Oracle®

アプリケーション開発者ガイド - XML

10*g*（9.0.4）

部品番号 : B12341-01

2004 年 2 月

ORACLE®

Oracle アプリケーション開発者ガイド - XML, 10*g*（9.0.4）

部品番号 : B12341-01

原本名 : Oracle Application Developer's Guide - XML, 10*g* (9.0.4)

原本部品番号 : B12099-01

原本著者 : Shelley Higgins

原本協力者 : Omar Alonso, Sandeepan Banerjee, Neerja Bhaat, Kishore Bhamidipati, Stefan Buchta, Raghunath Chintalapati, Robert Dell'immagine, Brajesh Goyal, Robert Hall, Karun K, Stefan Kiritzo,Vishu Krishnamurthy, Murali Krishnaprasad, Olivier LeDiouris, Janet Lee, Wesley Lin, Bryn Llewellen, Colin McGregor, Ian Macky, Anjana Manian, Becca Martin, Shailendra Mishra, Steve Muench, Bhagat Nainani, Paul Narth, Visar Nimani, Paul Nock, Ami Parekh, Rajesh Raheja, Carol Roston, Frank Rovitto, Tomas Saulys, Mark Scardina, Flora Sun, Prabhu Thukkaram, Rodney Ward, Philipp Weckerle, Manh-Kiet (Allen) Yap, Ari Adler, Omar Alonso, Cathy Baird, Phil Bates, Catherine Bauer, Mark Bauer, Ravinder Booreddy, Steve Cave, Steve Corbett, Claire Dessaux, Roger Ford, William Gietz, Steven Leung, Anand Manikutty, Narayan Mantravadi, Jack Melnick, Jitendra Pandey, Andy Page, Rahul Pathak, Padmini Ranganathan, Den Raphaely, Jim Rawles, David Saslav, Chitra Sharma, Keith Swartz, Priya Vennapusa,Melanie Watson, Jon Wilkinson, Vikram Yavagal, Tim Yu, Kongyi Zhou, Valerie Moore

目次

2 Oracle の XML アプリケーションのモデリングおよび設計問題

3 XDK および XML コンポーネント : 概要および一般的な FAQ

4 XSL および XSLT の使用

第 II 部　データベースに対する XML の格納および取出し

5　XML に対するデータベース・サポート

6 DBURI 参照

7 XSU

8 Oracle Text を使用した XML データの検索

第 III 部　XML を使用したデータ交換

9　Oracle AQ を使用した XML データの交換

11 JDeveloper を使用した Oracle の XML アプリケーションの作成

12 BC4J および XML アプリケーションの作成

13 メタデータ API の使用

14 OracleAS Reports Services および XML

15 PDK を使用した Oracle Portal での XML データのビジュアル化

16 OE での XML の使用

17 Oracle XML Gateway の概要

第 V 部　OracleAS Dynamic Services（DS）および Oracle Syndication Server（OSS）

18　OracleAS Dynamic Services および XML

19　Oracle Syndication Server（OSS）および XML

第 VI 部　XDK for Java

20　XML Parser for Java の使用

21 XML Schema Processor for Java の使用

22 XML Class Generator for Java の使用

第 VII 部　XDK for JavaBeans

23　XML Transviewer Beans の使用

第 VIII 部　XDK for C

24　XML Parser for C の使用

25 XML Schema Processor for C の使用

第 IX 部　XDK for C++

26　XML Parser for C++ の使用

27 XML Schema Processor for C++ の使用

28 XML C++ Class Generator の使用

第 X 部　XDK for PL/SQL

29　XML Parser for PL/SQL の使用

A XML の手引き

B Oracle XML Parser および Class Generator の言語間比較

C XDK for Java: 仕様および早見表

D XDK for JavaBeans: 仕様および早見表

E XDK for C: 仕様および早見表

F XDK for C++: 仕様および早見表

G XDK for PL/SQL: 仕様および早見表

H XSU の仕様および早見表

用語集

索引

はじめに

ここでは、次の項目について説明します。

このマニュアルの内容

このマニュアルでは、Oracle の eXtensible Markup Language（XML）対応データベース・テクノロジについて説明します。Oracle の XML 対応テクノロジおよび適切な開発ツールを使用して、データベースで XML データを格納、管理および問合せする方法を説明します。

Oracle の XML アプリケーションを設計する場合の主な基準を説明した後、実在するビジネス・アプリケーションに基づいた使用例の概要を説明します。また、XML Developer's Kit（XDK）の概要、および XDK コンポーネントを併用してデータベースに XML データを生成および格納する方法も説明します。必要に応じて、例およびサンプル・アプリケーションを示します。

例およびサンプル・コード

このマニュアルで使用する多くの XDK の例は、ソフトウェアの次のディレクトリにあります。

- `$ORACLE_HOME/xdk/java/demo/`
- `$ORACLE_HOME/xdk/C/demo/` など
- `$ORACLE_HOME/xdk/java/sample/`
- `$ORACLE_HOME/rdbms/demo/`

構成済 XML または分解済（生成される）XML

通常、XML 文書は、次のいずれかの方法で処理されます。

- LOB に格納された構成済 XML 文書。
- データベース表内の関連列にマップされた XML タグを持つ、リレーショナル表に格納された分解済 XML 文書フラグメント。分解済（断片化された）XML 文書は、構成済 XML 文書に生成しなおすことができます。

Oracle の XML 対応テクノロジ

XDK は、Oracle の主要な XML 対応テクノロジ・コンポーネントです。XDK は、次の 4 つの言語での実装が可能です。

- Java: XDK for Java、XDK for JavaBeans および XML SQL Utility（XSU）for Java
- PL/SQL: XDK for PL/SQL および XSU for PL/SQL
- C: XDK for C
- C++: XDK for C++

対象読者

このマニュアルは、Oracle または Oracle Application Server（OracleAS）で XML アプリケーションを構築する開発者を対象としています。

前提知識

このマニュアルを使用するには、XML および eXtensible Stylesheet Language（XSL）を理解していることが理想ですが、必須ではありません。付録 A および第 3 章の FAQ の項に、推奨ドキュメントの参照を記載しています。付録 A には、XML の手引きがあります。

このマニュアル内の多くの例は、SQL、Java、PL/SQL、C または C++ で記述されています。そのため、このうちの 1 つ以上の言語の実用経験があることを前提としています。

XML は理解しているがデータベースの知識がない場合

次の手順を実行することをお薦めします。

1. 『Oracle9*i* データベース概要』を読んで、データベースを計画、モデリングおよび設計します。
2. 第 I 部「Oracle の XML 対応テクノロジの概要」および第 II 部「データベースに対する XML の格納および取出し」の章を読みます。
3. 次の Oracle Technology Network（OTN）のサイトにアクセスします。
   - http://otn.oracle.com: データベース機能の概要
   - http://otn.oracle.com/tech/xml: 使用可能な XDK の情報およびホワイト・ペーパーとデモ
4. 次の FAQ を参照します。
   - 3-26 ページの「FAQ: Oracle の XML 対応テクノロジ」
   - 7-58 ページの「FAQ: XSU」
   - 8-52 ページの「FAQ: Oracle Text」
   - 10-76 ページの「FAQ: XSQL Servlet」
   - 20-59 ページの「FAQ: XML Parser for Java」
5. 質問がある場合、オラクル社カスタマ・サポート・センターに問い合せるか、OTN の「Discussions」オプションにアクセスして質問をポストします。

6. アプリケーションに必要な言語およびアプリケーションの構築に必要な XDK コンポーネントを決定したら、XML コンポーネントの詳細およびその使用方法について、次の章およびマニュアルを参照します。
   - 第 7 章、第 10 章および第 IV 部「Oracle ベースの XML アプリケーションを構築するためのツール製品およびフレームワーク」～第 X 部「XDK for PL/SQL」
   - 『Oracle9*i* XML リファレンス』

データベースは理解しているが XML の知識がない場合

次の手順を実行することをお薦めします。

1. 付録 A「XML の手引き」、および第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」の最後に記載されている推奨書籍を読みます。XML の概要については、多くの推奨書籍および Web サイトがあります。付録 A では、これらの推奨書籍および Web サイトのいくつかを示しています。
2. 第 4 章「XSL および XSLT の使用」を読みます。
3. 次の FAQ を参照します。
   - 3-26 ページの「FAQ: Oracle の XML 対応テクノロジ」
   - 7-2 ページの「FAQ: XSU」
   - 8-52 ページの「FAQ: Oracle Text」
   - 10-76 ページの「FAQ: XSQL Servlet」
   - 20-59 ページの「FAQ: XML Parser for Java」
4. Oracle データベース固有の（新しい）XML サポートについて読む必要があります。第 I 部「Oracle の XML 対応テクノロジの概要」および第 II 部「データベースに対する XML の格納および取出し」を参照してください。
5. OTN の XML サイト（http://otn.oracle.com/tech/xml）にアクセスして、使用可能な XDK の情報およびホワイト・ペーパーとデモを参照します。
6. アプリケーションに必要な言語およびアプリケーションの構築に必要な XDK コンポーネントを決定したら、XML コンポーネントの詳細およびその使用方法について、次の章およびマニュアルを参照します。
   - 第 7 章、第 10 章および第 IV 部「Oracle ベースの XML アプリケーションを構築するためのツール製品およびフレームワーク」～第 X 部「XDK for PL/SQL」
   - 『Oracle9*i* XML リファレンス』
7. 質問がある場合、オラクル社カスタマ・サポート・センターに問い合せるか、OTN の「Discussions」オプションにアクセスして質問をポストします。

機能範囲および可用性

このマニュアルの情報は、Oracle の XML 対応テクノロジ・コンポーネントの現在の情報を表します。これらの情報は常に更新されています。最新の情報を参照するには、OTN（http://otn.oracle.com/tech/xml）にアクセスしてください。

このマニュアルの構成

このマニュアルは 10 部編成で、29 の章および 8 つの付録で構成されています。索引および用語集も含まれます。

このマニュアルのロードマップ

第 1 章「Oracle の XML 対応テクノロジ」に、詳細な図があります。

- **XML および Oracle データベースの概要 :** Oracle データベースの XML コンポーネント（第 1 章～第 8 章）の使用、データベースにおける XML のサポート、XMLType と URI 参照の使用、XSU、および Oracle Text を適用した XML 文書からの情報の検索および取出し方法に関する概要および基本情報を説明します。
  - 第 I 部「Oracle の XML 対応テクノロジの概要」
    - 第 1 章「Oracle の XML 対応テクノロジ」では、Oracle XDK、データベース固有の（新しい）XML サポート、XMLType と DBURI 参照、XML アプリケーションの構築に使用するツール製品、および Oracle Text（*inter*Media Text）の概要を説明します。
    - 第 2 章「Oracle の XML アプリケーションのモデリングおよび設計問題」では、XML の設計およびロードの問題について説明します。また、Oracle の XML コンポーネントを、典型的なコンテンツ / ドキュメント管理アプリケーションおよび B2B メッセージ機能アプリケーションに使用する方法も説明します。
    - 第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」では、Oracle の XML コンポーネント、XDK および XSU の概要を説明します。また、Java、C、C++ および PL/SQL 用に XML 文書を生成する方法の概要も示します。この章では、Oracle の XML 対応テクノロジの FAQ も示します。
    - 第 4 章「XSL および XSLT の使用」では、XML および eXtensible Stylesheet Language Transformation（XSLT）の概要を説明します。また、カスケーディング・スタイルシート（CSS）と XSL の違いについても説明します。この章では、FAQ も示します。

- 第 II 部「データベースに対する XML の格納および取出し」
  - 第 5 章「XML に対するデータベース・サポート」では、XMLType データ型（新機能）の概要を説明し、データベースで XML を生成および格納する場合に XMLType を使用する方法を説明します。また、extract()、existsnode() などのメンバー関数、SQL 問合せでの XML の生成に使用する DBMS_XMLGEN パッケージおよび SYS_XMLGEN 関数、XML データの集計に使用する SYS_XMLAGG 関数、XMLType の作成と管理、およびファンクション索引と表関数を使用した XML の問合せについても説明します。
  - 第 6 章「DBURI 参照」では、URI 参照と DBURI 参照、URIType およびサブタイプ DBUriType と HttpUriType の使用方法、新しい SYS_DBURIGEN() 演算子、URIFactory メソッド、オブジェクトへの Hypertext Transport Protocol（HTTP）アクセス、および OraDBUriServlet メカニズムについて説明します。
  - 第 7 章「XSU」では、Java および PL/SQL 用の XSU Application Program Interface（API）を使用して XML 文書を生成および格納する方法、データベース内の XML 文書に対して挿入、更新および削除を実行する方法、XSU コマンドライン・ツールを使用する方法、および列に要素をマップする方法を説明します。この章に記載されている例は、$ORACLE_HOME/rdbms/demo/xsu にあります。この章では、FAQ も示します。
  - 第 8 章「Oracle Text を使用した XML データの検索」では、Oracle Text（*inter*Media Text）の概要、CONTAINS 演算子の使用、Oracle Text のセクションおよび索引を作成する方法、および問合せを作成する方法を説明します。また、XML_SECTION_GROUP、AUTO_SECTION_GROUP、新しい PATH_SECTION_GROUP、INPATH 演算子および HASPATH 演算子の使用例およびガイドラインを示します。この章では、FAQ も示します。

- **B2B および XML データ交換**
  - 第 III 部「XML を使用したデータ交換」
    - 第 9 章「Oracle AQ を使用した XML データの交換」では、アドバンスト・キューイング（AQ）の概念、および AQ と XML が互いにどのように補足し合うかについて説明します。また、（新しい）Internet-Data-Access-Presentation（iDAP）メカニズム、AQXMLServlet、および HTTP を使用して AQXMLServlet にアクセスする方法も説明します。さらに、XMLType キューおよび XML AQ メッセージ変換についても説明します。この章では、FAQ も示します。

- **Oracle 開発ツール :** 第 10 章～第 17 章では、XSQL ページ・パブリッシング・フレームワーク、JDeveloper、Business Components for Java（BC4J）、メタデータ API（新機能）、Oracle Reports、Oracle Portal、Oracle Exchange（OE）および XML Gateway の概要または使用方法のいずれかを説明します。
  - 第 IV 部「Oracle ベースの XML アプリケーションを構築するためのツール製品およびフレームワーク」
    * 第 10 章「XSQL ページ・パブリッシング・フレームワーク」では、XSQL Servlet の使用方法を説明します。図を使用して、XSQL Page Processor の動作についても示します。この章では、FAQ も示します。
    * 第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」では、JDeveloper を使用して XML アプリケーションを構築する方法、JDeveloper で XSQL Servlet を使用する方法、および JDeveloper を使用してモバイル・アプリケーションを構築する手順について説明します。この章では、FAQ も示します。
    * 第 12 章「BC4J および XML アプリケーションの作成」では、BC4J の概要および BC4J フレームワークを使用して XML アプリケーションを構築する方法を説明します。
    * 第 13 章「メタデータ API の使用」では、メタデータ API の概要および使用方法を説明します。また、（新しい）DBMS_METADATA のプログラム・インタフェースとブラウザ・インタフェースについても説明し、詳細な例を示します。
    * 第 14 章「OracleAS Reports Services および XML」では、Oracle Reports を使用して、レポートを XML として生成およびカスタマイズする方法、およびレポートから XML データ・ソースを読み取る方法を説明します。また、（新しい）トランスポータブル XML データ・ソース（XML-PDS）の使用、および XSQL に対するサポートについても説明します。
    * 第 15 章「PDK を使用した Oracle Portal での XML データのビジュアル化」では、OracleAS Portal の機能および Portal Developer Kit（PDK）の概要を説明します。また、URL Services を使用して、XML アプリケーションを Web またはデータベースをベースとしたポートレットとして使用可能にする方法も説明します。
    * 第 16 章「OE での XML の使用」では、OE、その格納されたトランザクションとパス・スルー・トランザクション、XML 配布形式および E-Business ソリューション・アーキテクチャについて説明します。
    * 第 17 章「Oracle XML Gateway の概要」では、Oracle XML Gateway の概要を説明します。Oracle XML Gateway は、ビジネス・イベントによってトリガーされる XML メッセージを作成および使用するために、Oracle E-Business Suite との統合を簡単にするための一連のサービスです。

- 第 V 部「OracleAS Dynamic Services（DS）および Oracle Syndication Server（OSS）」
  - 第 18 章「OracleAS Dynamic Services および XML」では、OracleAS Dynamic Services、そのアーキテクチャ、Java、PL/SQL と JMS/HTTP の配置モード、および Dynamic Services サービスの開発の概要を説明します。
  - 第 19 章「Oracle Syndication Server（OSS）および XML」では、Oracle Syndication Server（OSS）の概要、ICE プロトコルの使用方法、OSS アーキテクチャ、および OSS が、OracleAS Dynamic Services、コンテンツ・プロバイダおよびサブスクライバとのインタフェースとして機能する方法を説明します。
- **XDK:** ロードマップの「XML アプリケーション」ボックスに、様々な XDK を示します。第 7 章および第 19 章～第 29 章では、XDK の使用方法を説明します。
  - 第 VI 部「XDK for Java」
    - 第 20 章「XML Parser for Java の使用」では、XML Parser for Java および XSLT プロセッサの使用方法を説明します。ソフトウェアに含まれる例もリストしています。この章では、FAQ も示します。
    - 第 21 章「XML Schema Processor for Java の使用」では、XML Schema の概要を説明し、XML Schema と Document Type Definition（DTD）を比較します。また、Oracle XML Schema Processor for Java の機能、使用方法、およびサンプル・プログラムの使用方法を説明します。この章では、FAQ も示します。
    - 第 22 章「XML Class Generator for Java の使用」では、XML Java Class Generator を DTD および XML Schema とともに使用する方法を説明します。ソフトウェアに含まれる例もリストしています。この章では、FAQ も示します。
  - 第 VII 部「XDK for JavaBeans」
    - 第 23 章「XML Transviewer Beans の使用」では、（新しい）DBViewer Bean および（新しい）DBAccess Bean を含む XML Transviewer Beans およびその使用方法を説明します。ソフトウェアに含まれる例もリストしています。
  - 第 VIII 部「XDK for C」
    - 第 24 章「XML Parser for C の使用」では、XML Parser for C および XSLT プロセッサの使用方法を説明します。ソフトウェアに含まれる例もリストしています。
    - 第 25 章「XML Schema Processor for C の使用」では、XML Schema Processor for C の機能、コール順序および提供されているサンプル・プログラムを実行する方法を説明します。提供されている例もリストしています。

- 第 IX 部「XDK for C++」
  - 第 26 章「XML Parser for C++ の使用」では、XML Parser for C++ および XSLT プロセッサの使用方法を説明します。ソフトウェアに含まれる例もリストしています。
  - 第 27 章「XML Schema Processor for C++ の使用」では、XML Schema Processor for C++ の機能、コール順序および提供されているサンプル・プログラムを実行する方法を説明します。提供されている例もリストしています。
  - 第 28 章「XML C++ Class Generator の使用」では、XML C++ Class Generator の使用方法を説明します。ソフトウェアに含まれる例もリストしています。
- 第 X 部「XDK for PL/SQL」
  - 第 29 章「XML Parser for PL/SQL の使用」では、XML Parser for PL/SQL および XSLT プロセッサの使用方法を説明します。ソフトウェアに含まれる例もリストしています。この章では、FAQ も示します。

ロードマップには、XML の手引き、XDK の早見表や仕様などの付録は記載されていません。

- 付録 A「XML の手引き」では、XML に関する基本情報および背景情報を説明します。
- 付録 B「Oracle XML Parser および Class Generator の言語間比較」では、実装言語に基づいて、Oracle XML Parser と Class Generator を比較します。
- 付録 C「XDK for Java: 仕様および早見表」では、XDK for Java コンポーネントの仕様を説明します。いくつかのトップレベル・クラスおよびメソッドもリストしています。
- 付録 D「XDK for JavaBeans: 仕様および早見表」では、XDK for JavaBeans（特に Transviewer Beans の早見表）について説明します。
- 付録 E「XDK for C: 仕様および早見表」では、XDK for C の仕様を説明します。トップレベル・ファンクションもリストしています。
- 付録 F「XDK for C++: 仕様および早見表」では、XDK for C++ コンポーネントの仕様を説明します。いくつかのトップレベル・クラスおよびメソッドもリストしています。
- 付録 G「XDK for PL/SQL: 仕様および早見表」では、XDK for PL/SQL コンポーネントの仕様を説明します。いくつかのトップレベル・ファンクションもリストしています。
- 付録 H「XSU の仕様および早見表」では、XSU for Java および XSU for PL/SQL の仕様を説明します。いくつかのトップレベル・メソッドおよびファンクションもリストしています。

関連ドキュメント

詳細は、次の Oracle リソースを参照してください。

- 『Oracle9*i* データベース新機能』
  Oracle9*i* と Oracle9*i* Enterprise Edition の違い、および使用可能な機能とオプションについて説明します。また、Oracle9*i* のすべての新機能についても説明します。
- 『Oracle9*i* データベース概要』
- Oracle JDeveloper オンライン・ヘルプ
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』
- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『Oracle Integration Server Overview』
- 『Oracle9*i* XML リファレンス』
- 『The Oracle XML Handbook』XML Core Development Team 著、Oracle Press 出版
- 『Building Oracle XML Applications』Steve Muench 著、O'Reilly 出版
- 『XML Bible』Elliotte Rusty Harold 著、IDG Books Worldwide 出版
- 『XML Unleashed』Morrison その他著、SAMS 出版
- 『Building XML Applications』St.Laurent、Cerami 著、McGraw-Hill 出版
- 『Building Web Sites with XML』Michael Floyd 著、Prentice Hall PTR 出版
- 『Building Corporate Portals with XML』Finkelstein、Aiken 著、McGraw-Hill 出版
- 『XML in a Nutshell』O'Reilly 出版
- 『Learning XML - （Guide to）Creating Self-Describing Data』Ray 著、O'Reilly 出版
- http://www.xml.com/pub/rg/46
- http://www.xmlmag.com/
- http://www.webmethods.com/
- http://www.clarient.org/
- http://www.xmlwriter.com/
- http://webdevelopersjournal.com/articles/why_xml.html
- http://www.w3schools.com/xml/
- http://www.w3.org/TR/REC-xml

リリース・ノート、インストール・マニュアル、ホワイトペーパーなどのダウンロード

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連書籍は、OTN に接続すれば無償でダウンロードできます。OTN を利用するには、オンライン登録をする必要があります。次の URL で登録できます。

http://otn.oracle.co.jp/membership/

すでに OTN のユーザー名およびパスワードを所有している場合は、OTN の Web サイトから無償でドキュメントを参照およびダウンロードすることが可能です。

http://otn.oracle.co.jp/document/

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

- 本文中の表記規則
- コード例中の表記規則

本文中の表記規則

本文では、特別な用語をより迅速に識別できるように、様々な表記規則を使用します。次の表に、それらの表記規則を説明し、その使用例を示します。

表記規則	意味	例
太字	太字は、本文中で定義されている用語または用語集にある用語（あるいはその両方）を示します。	この句を指定すると、**索引構成表**が作成されます。
大文字の固定幅フォント	大文字の固定幅フォントは、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、RMAN キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを取ることができます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。

表記規則	意味	例
小文字の固定幅フォント	小文字の固定幅フォントは、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名、データベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクト、データベース構造、列名、パッケージ、クラス、ユーザー名、ロール、プログラム・ユニットおよびパラメータの値も含まれます。 **注意：** 大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。
小文字の固定幅イタリック体	小文字の固定幅イタリック体は、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。ここで、*old_release* とはアップグレード前にインストールしたリリースを示します。

コード例中の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文を説明します。コード例は、固定幅フォントで表示され、この例に示すとおり通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用される表記規則を説明し、その使用例を示します。

表記規則	意味	例
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。オプションのうちの 1 つを入力します。縦線は、入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]

表記規則	意味	例
. . .	水平省略記号は、次のいずれかを示します。	
	■ 例に直接関連しないコードの一部が省略されている。	`CREATE TABLE ... AS subquery;`
	■ コードの一部を繰り返すことができる。	`SELECT col1, col2, ... , coln FROM employees;`
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおり入力する必要があります。	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
イタリック体	イタリック文は、プレースホルダまたは特定の値を指定する必要のある変数を示します。	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

大文字	大文字は、システムが提供する要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で示されます。用語が大カッコ内にないかぎり、表示されているとおりの順序および綴りで入力します。ただし、これらの用語は大文字 / 小文字が識別されないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザーが提供するプログラム要素を示します。たとえば、表名、列名またはファイル名などです。
	注意： 大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Oracle の XML 対応テクノロジの新機能

ここでは、次のリリースでの新機能について説明します。

- Oracle アプリケーション開発者ガイド - XML 10g（9.0.4）で導入された新機能

Oracle アプリケーション開発者ガイド - XML 10*g*（9.0.4）

10g（9.0.4）で導入された XML の新機能は次のとおりです。

- **XDK for Java**
  - XML Schema Processor for Java
  - XML Parser for Java - DOM 2.0 および SAX 2.0 のサポート
  - XSLT のパフォーマンスの向上

    参照： 次の章を参照してください。

    - 第 20 章「XML Parser for Java の使用」
    - 第 21 章「XML Schema Processor for Java の使用」

  - Class Generator for Java に含まれる DTD ベースおよび XML Schema ベースの Class Generator

    参照： 第 22 章「XML Class Generator for Java の使用」を参照してください。

- **XSQL Servlet および XSQL Pages**
  - データベース・バインド変数のサポート : 字句置換とデータベース・バインド変数の両方をサポートしているため、パフォーマンスがさらに向上しています。
  - Apache FOP を使用した PDF 出力のサポート : XSQL Pages を Apache FOP プロセッサと組み合せることで、すべての XML コンテンツを Adobe PDF 形式で出力できます。
  - XSLT スタイルシートに対するトラステッド・ホストのサポート : 新しいセキュリティ機能によって、非トラステッド・ホストからはスタイルシートを実行できなくなります。
  - Oracle 以外の JDBC ドライバの完全サポート : すべての問合せ、挿入、更新および削除機能を、Oracle JDBC ドライバと Oracle 以外の JDBC ドライバの両方で実行できます。
  - 動的に構築された XSQL ページの処理 : XSQLRequest API では、プログラムによって構築された XSQL ページを処理できます。
  - カスタムの Connection Manager の使用 : ユーザー独自の Connection Manager を実装して、必要に応じた方法でデータベース接続を処理できます。
  - インライン XML Schema の作成 : オプションで、XML 問合せ結果の構造を記述するインライン XML Schema を作成できます。

- 問合せ用のデフォルト日付書式の設定 : 日付書式マスクを設定して、日付データを書式化する方法のデフォルトを変更できます。
- カスタム・シリアライザの作成 : カスタム・シリアライザを作成および使用して、XSQL Page Processor がクライアントに戻す内容および方法を制御できます。
- スタイルシートの動的割当て : パラメータ、または SQL 問合せ結果に基づいて、スタイルシートを動的に割り当てることができます。
- ポストされた XML の更新または削除 : XML の挿入に加えて、更新および削除もサポートされます。
- ターゲット列のみの挿入または更新 : すべての挿入要求または更新要求に含める列を、明示的にリストできます。
- ページ要求範囲付きオブジェクト : アクション・ハンドラによって、ページ要求コンテキストでオブジェクトを取得および設定し、ページ内のアクション間で状態を共有できます。
- ServletContext へのアクセス : HttpRequest オブジェクトおよび HttpResponse オブジェクトに加えて、ServletContext にもアクセスできます。

  参照 : 第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

- **XDK for JavaBeans**
  - DBViewer Bean（新規）: データベースに対する問合せまたはすべての XML に XSL スタイルシートを適用して、結果の HTML をスクロール可能なパネルに表示します。
  - DBAccess Bean（新規）: DBAccess Bean は、複数の XML 文書およびテキスト・ドキュメントを保持する CLOB 表をメンテナンスします。

    参照 : 第 23 章「XML Transviewer Beans の使用」を参照してください。

- **XDK for C**
  - XML Parser for C - DOM 1.0 および DOM CORE 2.0（DOM のサブセット）
  - XML Schema Processor for C
  - XSLT のパフォーマンスの向上

    参照 : 第 25 章「XML Schema Processor for C の使用」を参照してください。

- **XDK for C++**
  - XML Parser for C++ - DOM 1.0 および DOM CORE 2.0（DOM のサブセット）
  - XML Schema Processor for C++
  - XSLT のパフォーマンスの向上

    参照： 第 27 章「XML Schema Processor for C++ の使用」を参照してください。

- **XDK for PL/SQL**
  - XSLT のパフォーマンスの向上

    参照： 第 29 章「XML Parser for PL/SQL の使用」を参照してください。

XSU の機能

- 任意の SQL 問合せによる XML Schema の生成
- XMLType および URI 参照のサポート
- SAX2 コールバックのストリームとしての XML の生成
- データベースからの XML 作成時における XML 属性のサポート : これによって、特定の列または列のグループを、XML 要素ではなく XML 属性に簡単にマップできます。

XSU は、XDK for Java および XDK for PL/SQL の一部でもあります。

参照： 第 7 章「XSU」を参照してください。

XML 関連のデータベースの拡張

XML は、World Wide Web Consortium（W3C）が策定した規格であり、Web で構造化データおよび非構造化データを表現するための標準フォーマットです。Universal Resource Identifier（URI）は、Web ページなどの Web 上のリソースを識別します。Oracle は、XML データおよび URI データを処理するための型、およびデータベースに格納されているデータにアクセスするための DBURI 参照という URI のクラスを提供します。また、データベースに外部 URI および内部 URI を格納およびアクセスするための、新しい一連の型も提供します。

XMLType

Oracle が提供するこの（新しい）型を使用して、データベースに XML データを格納したり、データベース内の XML データを問い合せることができます。XMLType に含まれるメンバー関数を使用すると、XPath 式を使用して XML データのアクセス、抽出および問合せを行うことができます。XPath も、W3C が策定した、XML 文書を全検索するための規格です。Oracle XMLType 関数は、W3C の XPath 式に準じたフォーマットをサポートします。Oracle は、既存のリレーショナル・データまたはオブジェクト・リレーショナル・データから XML 文書を作成するための、一連の SQL 関数（SYS_XMLGEN や SYS_XMLAGG など）および PL/SQL パッケージ（DBMS_XMLGEN など）も提供します。

XMLType はシステム定義型であるため、関数の引数、または表やビューの列のデータ型として使用できます。表に XMLType 列を作成すると、Oracle は内部的に CLOB を使用して、この列に対応付けられた実際の XML データを格納します。すべての CLOB データと同様に、更新は、XML 文書全体に対してのみ行えます。XMLType 列には、Oracle Text 索引または他のファンクション索引を作成できます。

UriType

Oracle は、継承階層によって関連付けられた UriType のグループ（UriType、DBUriType および HttpUriType）を提供します。UriType はオブジェクト型であり、その他は UriType のサブタイプです。

- HttpUriType を使用すると、外部 Web ページまたはファイルに対する URL を格納できます。この型は、HTTP を使用してこれらのファイルにアクセスします。
- DBUriType を使用すると、データベース内のデータを参照する DBURI 参照を格納できます。UriType はスーパータイプであるため、この型で作成した列に、DBUriType 型または HttpUriType 型のインスタンスを格納できます。これによって、データベースの内外に格納されたデータを参照し、一貫的にデータにアクセスできます。

  DBURI 参照は、XPath に類似した表現を使用して、データベース内のデータを参照します。データベースを XML ツリーとして考えると、表、行および列は XML 文書の要素とみなすことができます。たとえば、人事部のユーザー hr が、次のような XML ツリーを参照するとします。

```
<HR>
  <EMPLOYEES>
    <ROW>
      <EMPLOYEE_ID>205</EMPLOYEE_ID>
      <LAST_NAME>Higgins</LAST_NAME>
      <SALARY>12000</SALARY>
      .. <!-- other columns -->
    </ROW>
    ... <!-- other rows -->
  </EMPLOYEES>
  <!-- other tables..-->
</HR>
<!-- other user schemas on which you have some privilege on..-->
```

DBURI 参照は、この仮想 XML 文書では、単純な XPath 式です。したがって、従業員表 EMPLOYEES で従業員番号 205 の SALARY 値を参照するには、DBURI 参照を次のとおり指定します。

```
/HR/EMPLOYEES/ROW[EMPLOYEE_ID=205]/SALARY
```

このモデルを使用して、CLOB 列またはその他の列に格納されたデータを参照し、URL として外部に公開できます。Oracle は、このような URL を解析する標準 URI サーブレットを提供します。このサーブレットは、Oracle Servlet Engine（OSE）上でインストールおよび実行できます。

UriFactoryType

UriFactoryType は**ファクトリ型**です。この型は、他のオブジェクト型を作成し、戻すことができます。URL 文字列を指定すると、UriFactoryType は、UriType の様々なサブタイプのインスタンスを作成できます。この型は、URL 文字列を分析し、URL のタイプ（HTTP、DBUri など）を識別し、対応するサブタイプのインスタンスを作成します。

> **参照：** 次の章を参照してください。
>
> - 第 5 章「XML に対するデータベース・サポート」
> - 第 6 章「DBURI 参照」
> - 第 8 章「Oracle Text を使用した XML データの検索」

AQ 機能

AQ の新機能には、次の拡張 XML メッセージ機能オプションが含まれます。

- iDAP
- HTTP アクセスに使用する AQXMLServlet
- XMLType キュー
- XML AQ メッセージ変換

> **参照：** 第 9 章「Oracle AQ を使用した XML データの交換」を参照してください。

メタデータ API

メタデータ API（新規）によって、集中化された単純かつ柔軟な方法で、次の作業を実行できます。

- データベース・オブジェクトの完全な定義（メタデータ）の、XML または作成 DDL のいずれかとしての抽出
- 業界標準の XSLT によるメタデータの変換
- SQL DDL の生成によるデータベース・オブジェクトの再作成

メタデータ API は、インスタンスが操作可能である場合はいつでも、Oracle データベースで使用できます。Oracle Lite では使用できません。メタデータ API には、提供された PL/SQL パッケージ DBMS_METADATA（新規）が含まれます。

> **参照：** 第 13 章「メタデータ API の使用」を参照してください。

Oracle Text（*inter*Media Text/Context）の機能

新しい Oracle Text セクション・グループの PATH_SECTION_GROUP によって、より高度な新しいセクション検索を XML 文書に対して実行できます。PATH_SECTION_GROUP は、次の機能をサポートします。

- 大文字 / 小文字の区別
- 直接の親子関係を保持した複数タグのパスの検索
- ワイルドカード・レベルのパス検索
- 参照の最上位タグの検索
- 属性値を識別した検索（セクションの存在による検索）

新しい Oracle Text 演算子は次のとおりです。

- HASPATH()
- INPATH()

> **参照：** 第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

OracleAS Reports Services

- サーバーの拡張機能 : Java ベースのサーバーに対しては、Pure Java で再エンジニアリングされています。ステータス情報が、HTML および XML で使用できます。
- トランスポータブル・データソースおよび宛先（新規）: 独自のデータ・アクセス・モジュールを Java で作成できます。
  - PDS-API を使用した Reports のプラグイン
  - データ・モデルとのシームレスな統合
  - 単一レポートへの複数のデータソースの結合
  - 送信 PDS（XML、JDBC、Express および SQL）
- JSP ベースのランタイム
- 改善されたポータルの統合、レポートのバースト、電子メール、配布、および PDF のサポート
- イベント・ベースのレポート

参照： 第 14 章「OracleAS Reports Services および XML」を参照してください。

第 I 部

Oracle の XML 対応テクノロジの概要

第 I 部では、Oracle の XML 対応テクノロジとその機能、Oracle XDK と XML コンポーネント、モデリングと設計の問題、および使用例について説明します。第 4 章では、XSL および XSLT の使用に関する基本的な情報を示します。

第 I 部に含まれる章は、次のとおりです。

- 第 1 章「Oracle の XML 対応テクノロジ」
- 第 2 章「Oracle の XML アプリケーションのモデリングおよび設計問題」
- 第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」
- 第 4 章「XSL および XSLT の使用」

1

Oracle の XML 対応テクノロジ

この章の内容は次のとおりです。

XML の概要

付録 A「XML の手引き」では、XML に関する入門情報、W3C の XML 勧告、HTML と XML の違いおよび他の XML 構文について説明します。また、情報交換のためのインターネット標準である XML が、データベース・アプリケーションでの使用に適した重要な言語である理由についても説明します。

Oracle の XML 対応テクノロジの概要

XML は、構造化および半構造化データをモデル化します。Oracle は、構造化データおよび半構造化データの他に、複合データおよび非構造化データもサポートします。また、XML に対応し、XML データの格納、問合せ、表示および操作を固有に処理します。

Oracle の XML コンポーネント

図 1-1 に、「XML アプリケーション」枠内の Oracle の XML コンポーネントについて示します。Oracle の XML コンポーネントは、次のコンポーネントで構成されます。

- データベース内の XML サポート
  - `XMLType` - XML 文書の格納、問合せおよび取出しを行うための新しいデータ型
  - `SYS_XMLGEN` - XML 文書を作成するための SQL 関数
  - `SYS_XMLAGG` - 複数の XML 文書を集計するための SQL 関数
  - `DBMS_XMLGEN` - SQL 問合せから XML を作成するための組込みパッケージ
  - URI のサポート - グローバル参照およびデータベース内参照の格納および取出し
  - テキストのサポート - `XMLType` およびテキスト列での XPath のサポート
- XDK for Java
  - XML Parser for Java および XSLT プロセッサ
  - XML Schema Processor for Java
  - XML Class Generator for Java
  - XSQL Servlet
  - XSU for Java
- XDK for JavaBeans
  - XML Transviewer Beans
    - DOMBuilder Bean
    - XSLTransformer Bean
    - DBAccess Bean

    - TreeViewer Bean
    - SourceViewer Bean
    - XMLTransformPanel Bean
    - DBViewer Bean
- XDK for C
  - XML Parser for C
  - XML Schema Processor for C
- XDK for C++
  - XML Parser for C++
  - XML Schema Processor for C++
  - XML Class Generator for C++
- XDK for PL/SQL
  - XML Parser for PL/SQL
  - XSU for PL/SQL

図 1-1 では、次の XML ベースのビジネス・ソリューションについても示します。

- XML を使用したビジネス・データ交換
  - バイヤー / サプライヤ間の透過的な取引の自動化
  - パートナのシームレスな統合および HTTP ベースのデータ交換
  - データベース・インベントリへのアクセス、商用トランザクションおよびフローの統合
  - Oracle *i*Procurement などを使用したセルフサービスによる調達
  - Oracle Discoverer 3i Viewer を使用したデータのマイニングおよびレポート
  - Oracle Exchange および Oracle Application
  - Phone Number Portability
- XML を使用したコンテンツおよびドキュメントの管理
  - パーソナライズされたパブリッシングおよびポータル
  - カスタマイズされた表示（Dynamic News の例、Portal-to-Go および Flight Finder）

図 1-1 Oracle の XML コンポーネントおよび E-Business ソリューション : 関連項目

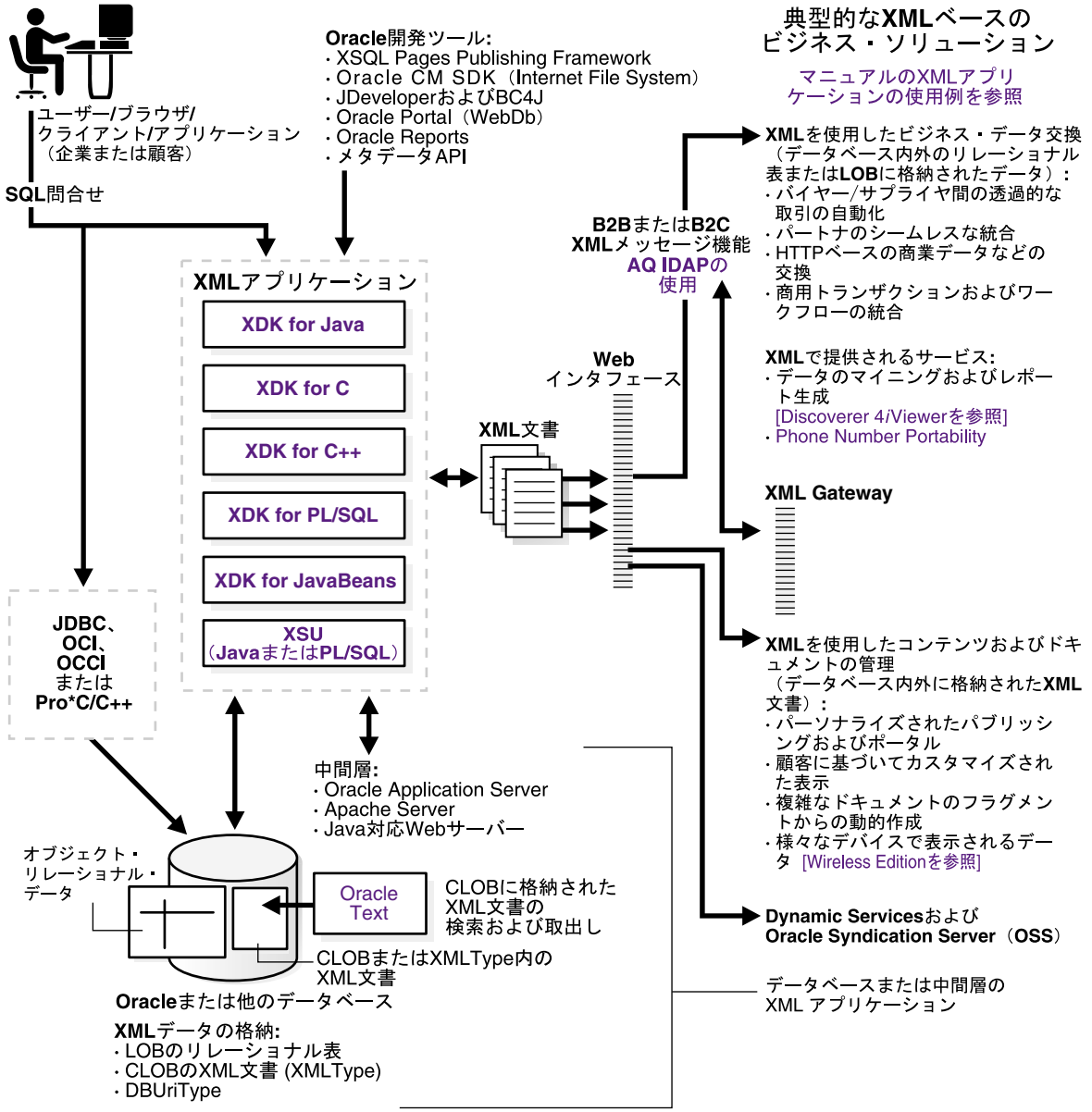


図 1-1 では、次のことについても示しています。

Oracle の開発ツールおよびフレームワーク

XSQL Servlet と XSQL Pages、Oracle CM SDK、JDeveloper、BC4J、Oracle Portal（WebDB）、Oracle Application Server Reports Services および OracleAS Dynamic Services を使用して、XML アプリケーションを構築できます。

注意： XSQL Servlet および XSQL Pages は、Oracle XDK for Java の一部です。

データベースおよび中間層

OracleAS、Apache Server やその他の Java 対応 Web サーバーのような XML アプリケーションは、データベース、または中間層のどちらかに常駐できます。

データベースに格納されたデータ

データは、オブジェクト・ビューを使用したリレーショナル表として、または XML 文書として、`XMLType` 列および CLOB 内に格納されます。Oracle Text（*inter*Media Text）を使用して、`XMLType` 列内または CLOB 列内に格納された XML 文書を効率的に検索できます。

Oracle からの XML データの格納および取出し

XML は、Web 上でのデータ交換の標準として登場しました。Oracle は XML 対応であり、XML を次のフォーマットで固有に格納、検索および取出しできます。

- **分解済 XML 文書 :** XML 文書は、構成フラグメントで格納されます。XML データはオブジェクト・リレーショナル形式で格納され、XSU または SQL 関数およびパッケージを使用して、これらのオブジェクト・リレーショナル・インスタンスから完全な（構成済の）XML 文書を生成できます。

  また、XSU または SQL 関数（`Extract()` や表関数など）を使用して XML を変換し、オブジェクト・リレーショナル（分解済）形式に戻すこともできます。

- **構成済（完全な）XML 文書 :** XML データを `XMLType` または CLOB/ バイナリ・ラージ・オブジェクト（BLOB）列内に格納し、`Extract()` や `ExistsNode()` などの `XMLType` 関数または Oracle Text の索引付けを使用して、これらの文書を検索します。

データベース内の XML サポート

Oracle データベースは、次の XML 機能をサポートします。

- **XML 文書の生成 :** Oracle は、クライアントまたはサーバー上での XML の生成をサポートします。クライアントまたはサーバー上では、既存のオブジェクト・リレーショナル・データを使用して、対応する XML を生成できます。XML は、問合せ結果から生成するか、または XSU の Java API や PL/SQL API を使用して SQL 問合せ自体の一部として生成することができます。

  今回のリリースでは、次の機能の提供によって、サーバーでの XML サポートが拡張されています。

  - XML を生成および集計するための新しい SQL 関数
  - サーバーにリンクされた、C バージョンの XSU

- **XML 文書の格納、問合せおよび取出し :** 以前のリリースでは、たとえば、XSU を使用して XML 文書の格納、問合せおよび取出しを行うことができました。今回のリリースでは、新しいデータ型である `XMLType` を使用できます。

  `XMLType` は、XML 文書を CLOB として格納します。その後、Oracle Text（*inter*Media Text）の索引付け機能を使用して `XMLType` 列を索引付けし、`CONTAINS` 演算子および XPath に類似した構文を使用してそれらの列を問い合せることができます。`XMLType` は、XML 文書からフラグメントを抽出するために使用できるメンバー関数もサポートします。

  参照 : 5-32 ページの「XMLType 列の索引付け」を参照してください。

XMLType および UriType

Oracle は、XML および URI データを処理するための新しい型を提供します。XML は、W3C が策定した規格であり、Web 上で構造化データおよび非構造化データを表現するための標準フォーマットです。

URI は、Web ページなどの Web 上のリソースを識別するために使用されます。Oracle は、データベース内に格納されているデータにアクセスするための新しい URI のクラスである、DBURI 参照を提供します。また、データベースから外部 URI および内部 URI を格納およびアクセスするための、新しい一連の型も提供します。

XMLType

Oracle が提供する型 XMLType は、データベース内に XML データを格納したり、XML データを問い合せるために使用することができます。XMLType は、XPath 式を使用して XML データにアクセスしたり、XML データを抽出および問い合せるためのメンバー関数を提供します。XPath も、W3C 委員会が策定した、XML 文書を全検索するための標準です。XMLType 関数は、XPath 式に準じたフォーマットのみをサポートします。また、既存のリレーショナル・データまたはオブジェクト・リレーショナル・データから XML 文書を作成するための、SYS_XMLGEN、SYS_XMLAGG などの一連の SQL 関数およびその他の PL/SQL パッケージ（DBMS_XMLGEN）も提供します。

XMLType はシステム定義型であるため、関数の引数、または表やビューの列として使用できます。表内に XMLType 列を作成すると、Oracle は内部的に CLOB を使用して、この列に対応付けられた実際の XML データを格納します。XMLType 列には、Oracle Text の索引およびその他のファンクション索引を作成できます。Oracle データベースでは、XMLType が CLOB として格納されるため、更新は文書全体に対してのみ実行できます。

参照： 次の章およびマニュアルを参照してください。

- 第 5 章「XML に対するデータベース・サポート」
- 第 8 章「Oracle Text を使用した XML データの検索」
- 第 9 章「Oracle AQ を使用した XML データの交換」
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』（Oracle AQ での XMLType の使用に関する情報）

UriType

Oracle は、次の UriType のグループを提供します。

- UriType
- DBUriType
- HttpUriType

これらは、継承階層によって関連付けられます。UriType は抽象型で、DBUriType および HttpUriType は UriType のサブタイプです。

- HttpUriType は、外部 Web ページまたはファイルへの URL を格納するために使用できます。このデータ型は、HTTP プロトコルを使用してこれらのファイルにアクセスします。
- DBUriType は、データベース内のデータを参照する DBURI 参照を格納するために使用できます。

UriType はスーパータイプであるため、この型の列を作成し、列内に DBUriType または HttpUriType のインスタンスを格納できます。これによって、データベース内外に格納されたデータを参照したり、それらのデータに一貫してアクセスすることができます。

DBURI 参照は、XPath に類似した表現を使用して、データベース内のデータを参照します。データベースを XML ツリーとして考えると、表、行および列は XML 文書内の要素とみなすことができます。たとえば、ユーザー scott が、次のようなツリーを参照するとします。

```
<SCOTT>
  <EMP>
    <ROW>
      <EMPNO>2100</EMPNO>
      <ENAME>John</ENAME>
      <SALARY>10000</SALARY>
      .. <!-- other columns -->
    </ROW>
    ... <!-- other rows -->
  </EMP>
  <!-- other tables..-->
</SCOTT>
<!-- other user schemas on which you have some privilege on..-->
```

DBURI 参照は、この仮想 XML 文書では、単純な XPath 式です。したがって、EMP 表で従業員番号 2100 の従業員の SALARY 値を参照するには、DBURI 参照を次のとおり指定します。

```
/SCOTT/EMP/ROW[EMPNO=2100]/SALARY
```

これを使用すると、CLOB またはその他の列に格納されたデータを参照し、URL として外部に公開できます。Oracle は、このような URL を解析する標準サーブレットを提供します。このサーブレットは、OSE でインストールおよび実行できます。

> **参照：** 第 6 章「DBURI 参照」を参照してください。

拡張性と XML

Oracle の拡張性によって、XML での特別な索引付け（セクション検索用の Oracle Text 索引など）、XML を処理するための特別な演算子、XML の集計、および XML を伴う問合せの特別な最適化が可能になります。

Oracle Text 検索

LOB に格納された XML テキストは、拡張索引付けインタフェースを使用して索引付けできます。Oracle は、CONTAINS や WITHIN などの演算子を提供します。これらの演算子を使用すると、XML テキストにおける部分文字列一致検索が可能です。

参照： 次の章を参照してください。

- 第 2 章「Oracle の XML アプリケーションのモデリングおよび設計問題」
- 第 8 章「Oracle Text を使用した XML データの検索」

Oracle ベースの XML アプリケーション

XML は、インターネット・アプリケーションにおいて様々な用途で使用できます。このマニュアルでは、Oracle の XML コンポーネントの使用に適した、次の 2 つの分野のデータベース中心のアプリケーションについて説明します。

コンテンツおよびドキュメントの管理

コンテンツ管理およびドキュメント管理には、データ表示のカスタマイズが含まれます。これらのアプリケーションでは、通常、ほぼ作成済の XML 文書が処理されます。このマニュアルでは、いくつかの例を記載しています。

参照：

- 「XML を使用したコンテンツのカスタマイズ : Dynamic News アプリケーション」
- 「OracleAS Wireless Edition と XML」
- 「XML および XSQL を使用した表示のカスタマイズ : Flight Finder」

B2B または Business-to-Consumer（B2C）メッセージ機能

B2B および B2C メッセージ機能には、ビジネス・アプリケーション間のデータ交換が伴います。これらのアプリケーションでは、通常、生成される XML 文書、または生成される XML 文書と構成済 XML 文書の組合せが処理されます。

参照： B2B については、次の章を参照してください。

- 第 9 章「Oracle AQ を使用した XML データの交換」

Oracle の XML コンポーネントの動作

Oracle の XML コンポーネントおよびコンポーネントの動作については、第 2 章「Oracle の XML アプリケーションのモデリングおよび設計問題」および第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」を参照してください。

このマニュアルの他の章では、Oracle の XML コンポーネントと Oracle の開発ツールの使用方法、およびこれらのツールを使用して、Web ベースのデータベース・アプリケーションを構築する方法について説明します。

Oracle の XML 対応テクノロジ・コンポーネントおよび機能

Oracle データベースは、XML データベース・アプリケーションの構築に適しています。Oracle の XML 対応テクノロジの機能は次のとおりです。

- Oracle Text（interMedia Text）を使用した XML 文書の索引付けおよび検索
- メッセージング・ハブおよび中間層コンポーネント
- バックエンド、データベース、フロントエンドの統合
- Oracle XDK が提供する API: DOM および SAX
- ツールおよびコンポーネントを統合した Oracle Suite
- Oracle の XML のサンプルおよびデモ

Oracle Text（*inter*Media Text）を使用した XML 文書の索引付けおよび検索

Oracle Text（*inter*Media Text）は、CLOB およびその他のドキュメントに格納された XML の検索および取出しのための、強力なオプションを提供します。表の列に格納された、最大 4GB の XML 文書およびドキュメント・セクションを索引付けおよび検索できます。

Oracle Text の XML 文書検索には、階層要素のコンテナ化、DOCTYPE 識別および XML 属性の検索が含まれます。これらの XML 文書検索は、標準の SQL 問合せ述語、または他の強力な語彙および全文検索オプションと組み合せて使用できます。

データベース内のテキスト CLOB に保存された XML 文書またはドキュメント・セクションは、Oracle Text のテキスト検索エンジンを使用して索引付けできます。開発者は、特定の XML 階層内のデータを正確に検索し、XML 要素の属性にある名前と値の組の場所を検索できます。

Oracle Text は、データベースおよび SQL 言語にシームレスに統合されているため、開発者は SQL を使用して、構造化データおよび索引付けされたドキュメント・セクションの両方を伴う問合せを簡単に実行できます。

参照： 次の章およびマニュアルを参照してください。

- 第 8 章「Oracle Text を使用した XML データの検索」
- 『Oracle Text リファレンス』

メッセージング・ハブおよび中間層コンポーネント

Oracle の XML には、次のコンポーネントも含まれています。

- **XML 対応メッセージング・ハブ :** これらのハブは、Oracle 以外のシステムと連結する B2B アプリケーションに必須です。第 9 章「Oracle AQ を使用した XML データの交換」を参照してください。
- **中間層システム :** OIS や OracleAS などの XML 対応のアプリケーション、Web または統合サーバーです。

Oracle JVM（Java VM）

Oracle JVM は、Oracle マルチスレッド・サーバー（MTS）・アーキテクチャに基づいて構築されています。これは Java 1.2 準拠の仮想マシンであり、データ・サーバーがメモリー・アドレス領域を共有します。Oracle JVM には、次の機能があります。

- 標準の JDBC インタフェースを使用して、Java および XML の処理コードをメモリー内データ・アクセス速度で実行します。
- Java バイトコードを固有にコンパイルして、数千の同時ユーザーにスケーラビリティを提供し、サーバー側の Java のパフォーマンスを向上させます。Oracle XDK コンポーネントは、事前にロードされ、固有にコンパイルされます。

Oracle JVM は、ネイティブ Common Object Request Broker API（CORBA）および Enterprise JavaBeans（EJB）標準と、Java を SQL および PL/SQL と簡単に統合するための Java ストアド・プロシージャをサポートします。

OracleAS

OracleAS は、イントラネットおよびインターネットの Web アプリケーションに対するサービスを提供します。Oracle と統合され、データ・キャッシュや Oracle Portal などの高度なサービスを提供します。また、OracleAS は、Oracle AQ、Oracle Message Broker（OMB）、Oracle Workflow、Oracle Reports Services、Dynamic Services などのその他のサービスも提供します。

参照： http://otn.oracle.com/products/ を参照してください。

バックエンド、データベース、フロントエンドの統合

開発における主な課題は、複数ベンダーのバックエンド ERP および CRM システムを、サプライ・チェーンのパートナ・システムおよびカスタマイズされたデータ・ウェアハウスと統合することです。

XML を使用すると、異なるベンダーのリレーショナル・データベースとオブジェクト・リレーショナル・データベース間のこのようなデータ交換がより単純になります。

Oracle の XML テクノロジおよび Oracle の XML 対応ツール、インタフェースおよびサーバーは、ほとんどのデータおよびアプリケーションの統合に対するソリューションを提供します。

高パフォーマンスへの影響

前述のソリューションを使用すると、次の理由からパフォーマンスが向上します。

- データベース・データおよび XML が同じサーバー上で同時に処理され、データ・アクセスによるネットワークの通信量が削減されます。
- Oracle の問合せエンジンが高速化され、Oracle JVM、OracleAS または OIS によってデータ・アクセス速度がさらに高速になります。
- XDK for C コンポーネントを使用すると、そのネイティブ XML 機能によって、パフォーマンスが向上します。

これによって、開発者は、多くの方法で Java、データベース・データ、機能を統合する XML ベースの Web ソリューションを構築できます。

Oracle XDK が提供する API: DOM および SAX

Oracle XDK は、Java、C、C++ および PL/SQL で実装されます。Java バージョンは、Oracle JVM 上で直接実行します。XML 1.0 の仕様をサポートし、検証または非検証パーサーとして使用されます。

Oracle XDK は、開発者が XML 文書を処理するために必要な、最も一般的な次の 2 つの API を提供します。

- **DOM 1.0 および 2.0:** W3C 勧告の DOM インタフェースです。解析済ドキュメントの要素内容に対して、アクセスおよび編集するための標準的な手段を提供します。
- **SAX 1.0 および 2.0:** SAX インタフェースです。

詳細は、第 20 章「XML Parser for Java の使用」を参照してください。Oracle XML Parser と Class Generator の比較については、付録 B「Oracle XML Parser および Class Generator の言語間比較」を参照してください。

カスタムの XML アプリケーションの作成

Oracle 環境では、XML 文書を処理するカスタム・アプリケーションを簡単に作成できます。これによって、すべての層に配置可能な言語を使用して、業界標準に準拠した移植可能なアプリケーションおよびコンポーネントを作成できます。

Oracle XML Parser は、Oracle データベースがインストールされたすべてのオペレーティング・システム（OS）上の Oracle プラットフォームの一部です。

Oracle XML Parser は、PL/SQL でも実装されます。このため、既存の PL/SQL アプリケーションを拡張して、Oracle の XML テクノロジのメリットを得ることができます。

ツールおよびコンポーネントを統合した Oracle Suite

Oracle では、E-Business アプリケーションを構築するために、次のツールおよびコンポーネントの統合スイートを提供しています。

- Oracle JDeveloper および BC4J
- Oracle CM SDK
- Oracle Portal
- Oracle Exchange

この統合スイートによって、アプリケーション開発におけるデータ・オブジェクトとドキュメント・オブジェクトの交換を簡略化でき、複数のシリアル化を排除できます。

Oracle JDeveloper および BC4J

Oracle JDeveloper は、Oracle で Java および XML を使用して、アプリケーションを構築、配置およびデバッグするために統合された環境です。この環境によって、CORBA、EJB および Java ストアド・プロシージャでの Java 1.1 または 1.2 の処理が簡単になります。Oracle JDeveloper を使用すると、次の操作が可能になります。

- Oracle の XML コンポーネントに直接アクセスして、複数層のアプリケーションを構築します。
- XML 情報を処理する Java Servlet を迅速に作成およびデバッグします。
- JDeveloper および BC4J コンポーネントを使用して、移植可能なアプリケーション・ロジックを構築します。

次に、Oracle JDeveloper を使用して構築されたアプリケーションの例を示します。

- セルフサービスの Web 上での経費処理を含む *i*Procurement（セルフサービス・アプリケーション）。
- オンライン・マーケットプレース。

JDeveloper および XML アプリケーションの詳細は、第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」を参照してください。

BC4J BC4J は、ビジネス・ロジックを再利用可能な Java コンポーネントのライブラリにカプセル化し、柔軟な SQL ベースの情報ビューを介してそのビジネス・ロジックを再利用するための、Oracle アプリケーション・フレームワークです。

注意： Oracle JDeveloper および BC4J は Oracle に含まれていません。BC4J Runtime のみが含まれています。JDeveloper は OTN からダウンロードできます。

Oracle CM SDK

Oracle CM SDK を使用すると、標準の Windows、および SMB、HTTP、FTP、SMTP、IMAP4 などのインターネット・プロトコルを介して、ファイルベースおよびフォルダベースのモデルを使用したドキュメントおよびデータの編成およびアクセスが簡単になります。

Oracle CM SDK を使用すると、Web ベースのアプリケーションを簡単に構築および管理できます。Oracle CM SDK は、Java 用アプリケーション・インタフェースであり、PowerPoint のファイルなどのドキュメントを Oracle9*i* にロードしたり、Oracle Application Server や Apache Web サーバーなどの Web サーバーのドキュメントを表示することができます。

Oracle CM SDK を使用すると、開発者は XML を使用して容易に作業ができます。これは、Oracle CM SDK が XML のリポジトリとして機能するためです。Oracle CM SDK は、自動的に XML を解析し、コンテンツを表および列に格納します。Oracle CM SDK は、選択された情報を含むファイルの配信が要求されると、Web などにコンテンツをレンダリングします。

詳細は、`http://otn.oracle.com/products/ifs/` を参照してください。

Oracle Portal

Oracle Portal は、たとえば、XML ベースの Rich Site Summary（RSS）形式のドキュメントを入力し、情報を XML スタイルシートとマージして、ブラウザ内でレンダリングできるようにします。この設計によって、情報の表示が情報自体から効率的に分割され、データの整合性を損なうことなく、ルックアンドフィール（外観と操作性）のカスタマイズが簡単にできるようになります。

Oracle Portal は、エンタープライズ・ポータルを構築および配置するためのソフトウェアです。エンタープライズ・ポータルとは、E-Business を実行するための Web サイトです。ブラウザ・インタフェースは、各ユーザーに必要なビジネス情報、Web コンテンツおよびアプリケーションを、編成およびパーソナライズされた方法で表示します。これには、WebDB 2.2 のサイト構築機能およびセルフサービス Web パブリッシング機能が含まれ、シングル・サインオン、パーソナリゼーションおよびコンテンツ分類などのエンタープライズ・ポータ

ル機能が追加されています。Oracle Portal は Oracle データベースを使用し、OracleAS に配置およびパッケージ化されています。

ポートレット： ポートレットは、Web ベースのリソースへのアクセスを提供する、再利用可能なインタフェース・コンポーネントです。ポートレットを介して、すべての Web ページ、アプリケーション、ビジネス・インテリジェンス・レポート、組織化されたコンテンツ配信、供給されたソフトウェア・サービスまたは他のリソースにアクセスし、Oracle Portal のサービスとして、これらをパーソナライズおよび管理できます。企業は、独自のポートレットを作成したり、ポートレットのサード・パーティ・プロバイダからポートレットを選択することができます。オラクル社は、開発者が PL/SQL、Java、HTML または XML を使用してポートレットを簡単に作成できるように、Portal Developer's Kit（PDK）を提供しています。

> **参照：** Oracle Portal の PDF および URL サービスについては、第 15 章「PDK を使用した Oracle Portal での XML データのビジュアル化」を参照してください。

Oracle Exchange

Oracle Exchange プラットフォームは、Oracle データベースに基づいています。業界全体または企業のサプライ・チェーンをサポートするために必要な、すべてのビジネス・トランザクションを提供します。Oracle Exchange は、Oracle の E-Business Suite に基づいています。E-Business Suite は、顧客候補からの初めての連絡から、製造計画および製造実施、および販売後の継続サービスおよびサポートまでのサプライ・チェーンをサポートします。

Oracle Exchange は、データ交換フォーマットおよびメッセージ・ペイロードとして XML および AQ を使用します。

> **参照：** 第 16 章「OE での XML の使用」を参照してください。

XML Gateway

XML Gateway は、ビジネス・イベントによってトリガーされる XML メッセージを作成および使用するために、Oracle E-Business Suite との統合を簡単にするための一連のサービスです。また、XML Gateway は、Oracle AQ と統合してメッセージのエンキューまたはデキューも行います。エンキューまたはデキューされたメッセージは、OMB などのメッセージ転送サービスを介してビジネス・パートナとの間で送受信されます。

> **参照：** 第 17 章「Oracle XML Gateway の概要」を参照してください。

メタデータ API

メタデータ API によって、集中化された単純かつ柔軟な方法で、次の作業を実行できます。

- データベース・オブジェクトの完全な定義（メタデータ）の、XML または作成 DDL のいずれかとしての抽出
- 業界標準の XSLT によるメタデータの変換
- SQL DDL の生成によるデータベース・オブジェクトの再作成

メタデータ API は、インスタンスが操作可能である場合はいつでも、Oracle データベースで使用できます。Oracle Lite では使用できません。

参照： 第 13 章「メタデータ API の使用」を参照してください。

XML に関するその他の取組み

これらのツールの他に、次の取組みを行っています。

XML Metadata Interchange（XMI）：ツールおよびデータ・ウェアハウス・メタデータの管理および共有

オラクル社、IBM 社および Unisys 社が提唱する XMI をサポートします。これによって、オラクル社および他社のアプリケーション開発ツールおよびデータ・ウェアハウス・ツールで共通のメタデータを交換できるようになり、アプリケーションおよびウェアハウスの設計を変更しなくても、すべてのツールを使用できます。

AQ の XML サポート：信頼性の高い、非同期メッセージ機能に対するインターネットの使用

Oracle AQ を使用すると、ペイロードとして XML 文書やドキュメントのセクションまたはフラグメントを含むメッセージなどの非同期メッセージを、安全な HTTP 上で高い信頼性をもって伝播できます。これによって、動的な取引が可能になり、企業間または代理店間のリンクを確立するための遅延および初期コストが削減されます。

参照： 第 9 章「Oracle AQ を使用した XML データの交換」を参照してください。

Oracle の XML のサンプルおよびデモ

このマニュアルには、Oracle の XML コンポーネントの使用例を記載しています。例は、1 つのスキーマに制限されたものではありません。例は、$ORACLE_HOME/rdbms/demo または $ORACLE_HOME/xdk/.../sample にあります。

Oracle の XML コンポーネントの実行要件

Oracle8*i* 以降には、Java や XML などのインターネット標準の固有のサポートが含まれます。これらを使用して構築された Oracle の XML コンポーネントおよびアプリケーションは、組込み Java VM である Oracle JVM を使用して、データベース内で実行できます。

Oracle Lite は、より小さいデータベースのフットプリント以外は要求しないデバイスおよびアプリケーションに対して、XML データの格納および取出しのために使用します。

Oracle の XML コンポーネントは、http://otn.oracle.com/tech/xml から無償でダウンロードできます。

XDK の要件

次に、XDK for Java および XDK for PL/SQL の要件を示します。

- XDK for Java には、Java Development Kit（JDK）/Java Runtime Environment（JRE）1.1 以上の Java VM が必要です。
- XDK for PL/SQL には、Oracle8.x 以降または PL/SQL カートリッジが必要です。

要件については、XDK に関する章である第 19 章～第 29 章および付録 C ～付録 G も参照してください。

Oracle データベースおよび OracleAS に含まれる XML コンポーネント

表 1-1 に、Oracle データベースおよび OracleAS に含まれる XDK コンポーネントのバージョンを示します。

表 1-1 Oracle データベースおよび OracleAS が提供する XDK コンポーネントのバージョン

XDK コンポーネント	Oracle データベース 10*g*（9.0.4）	OracleAS リリース 2（予定）
XDK for Java		
XML Parser for Java および XSLT プロセッサ	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for Java	9.0.1.0.0	9.0.1.0.0
XML Class Generator for Java	9.0.1.0.0	9.0.1.0.0
XSQL Servlet	9.0.1.0.0	9.0.1.0.0
XSU for Java	9.0.1.0.0	9.0.1.0.0
XDK for JavaBeans		
XML Transviewer Beans	9.0.1.0.0	9.0.1.0.0
XDK for C		
XML Parser for C および XSLT プロセッサ	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C	9.0.1.0.0	9.0.1.0.0
XDK for C++		
XML Parser for C++ および XSLT プロセッサ	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C++	9.0.1.0.0	9.0.1.0.0
XML Class Generator for C++	9.0.1.0.0	9.0.1.0.0
XDK for PL/SQL		
XML Parser for PL/SQL および XSLT プロセッサ	9.0.1.0.0	9.0.1.0.0
XSU for PL/SQL	9.0.1.0.0	9.0.1.0.0

XML の技術サポート

オラクル社カスタマ・サポート・センターの通常のサポート・チャネルの他に、Oracle の XML 対応テクノロジの技術サポートが、OTN の「Discussions」オプションから無償で利用できます。

```
http://otn.oracle.com/tech/xml
```

OTN の登録ユーザーでなくても、OTN Technical Discussion Forum で XML に関連する質問を送信したり、質問に回答することができます。OTN Technical Forum を利用するには、次の手順に従います。

1. OTN サイトの左側のナビゲーション・バーで、「Support」>「Discussions」と選択します。
2. 「Enter a Technical Forum」をクリックします。
3. 「Technologies」のセクションまでスクロールし、「XML」を選択します。
4. そこで、質問、コメント、リクエストまたはエラー・レポートをポストします。

OTN からの最新ソフトウェアのダウンロード

次の OTN の URL から、Oracle の XML コンポーネントの最新情報をダウンロードできます。

```
http://otn.oracle.com/software/
```

サイト上部の「Download Oracle Products, Drivers, and Utilities」の下にある「Select a Utility or Driver」プルダウン・メニューをスクロールし、表示される XML ユーティリティから任意のユーティリティを選択します。最新の XML Parser for Java および XML Parser for C++ をダウンロードする場合は、V2 を選択してください。

2

Oracle の XML アプリケーションのモデリングおよび設計問題

この章の内容は次のとおりです。

生成される XML または構成済 XML として格納可能な XML データ

Oracle では、XML データを次の方法で格納できます。

- **生成される XML:** XML データは、データベース内のオブジェクト・リレーショナル表に格納されるか、またはビューとして格納されます。このデータは、必要に応じて、動的に XML 形式に再生成が可能です。
- **構成済（作成済またはネイティブ）XML:** XML 文書は、CLOB 内に格納される場合と同様に格納されます。

生成される XML

XML は、オブジェクト・リレーショナル表およびビューから生成できます。純粋なリレーショナル構造ではなく、オブジェクト・リレーショナル表およびビューを使用するメリットは次のとおりです。

生成される XML は、XML が交換フォーマットであり、既存のビジネス・データが XML 構造（タグ）内にラップされる場合に使用されます。これは、データベース内で XML を使用する最も一般的な方法です。この場合、XML は交換処理自体のみに使用され、一時的です。

生成される XML の例

この種類のドキュメントの例には、販売注文書、請求書、運航スケジュールなどがあります。

オブジェクト・リレーショナル拡張された Oracle には、オブジェクト型、オブジェクト参照およびコレクション型を使用して、データベース内のデータ構造を利用する機能があります。次に示すとおり、XML データの構造をオブジェクト・リレーショナル形式で格納および保存するための 2 つのオプションがあります。

- 要素の属性をリレーショナル表に格納し、オブジェクト・ビューを定義して XML 要素の構造を実現します。
- 構造化 XML の要素をオブジェクト表に格納します。

データをオブジェクト・リレーショナル形式で格納すると、必要に応じて，SQL で簡単に更新、問合せ、再配置、再フォーマットできます。

生成される XML 文書のオブジェクト・リレーショナル形式での格納

複雑な XML 文書は、オブジェクト・リレーショナル・インスタンスとして格納し、効率的に索引付けできます。このようなインスタンスは、XML のネストおよびリスト・セマンティクスを完全に表現できます。Oracle の拡張インフラストラクチャを使用すると、パス索引などの新しいタイプの索引を作成して、XML 文書の検索を高速化できます。

XSU による XML の格納および XML への SQL 問合せ結果の変換

XSU は、基礎となるオブジェクト・リレーショナル形式での格納に対してその XML データをマップすることによって、XML 文書を格納します。また、オブジェクト・リレーショナル・データを XML 文書として取り出す機能も提供します。

XSU は、SQL 問合せの別名または列名を要素タグ名にマップし、オブジェクト型のネストを保持することによって、SQL 問合せの結果を XML に変換します。この結果をテキストまたは DOM ツリー形式にすることができます。DOM ツリーを生成すると、テキストを解析するオーバーヘッドを回避し、DOM ツリーが直接実現されます。

参照： 第 7 章「XSU」を参照してください。

構成済（作成済またはネイティブ）XML

Oracle8*i* 以上では、CLOB、BLOB または外部に格納されるバイナリ・ファイル（BFILE）としての LOB の格納がサポートされます。LOB は、構成済（作成済またはネイティブ）XML 文書の格納に使用されます。

構成済 XML データの CLOB または BFILE への格納

受け取った XML 文書が特定の構造に準拠しない場合、文書を CLOB に格納する方が有効である場合があります。たとえば、XML メッセージ機能環境では、キュー内の各 XML メッセージの構造が異なる場合があります。

CLOB は大きい文字データを格納し、構成済 XML 文書の格納に有効です。

外部ファイル参照である BFILE も格納に使用できますが、BFILE はアクセス頻度の少ないマルチメディア・データにより適しています。この場合、XML は Oracle 外に格納され管理されますが、サーバー側の問合せに使用できます。文書のメタデータは、索引付けおよびアクセスを高速化するために、サーバーのオブジェクト・リレーショナル表に格納できます。

XML 文書の内容が文書全体の置換によってのみ更新されるような静的な内容である場合、元の XML 文書を CLOB または BLOB に格納することをお薦めします。

- 構成済 XML の例には、雑誌記事、広告、本、契約書などのテキストがあります。この種類のドキュメントは文書中心であり、データベースから全体が取り出されます。この種類のドキュメントを Oracle に格納すると、ファイル・システムでの格納に対して、データベースのメリットおよびその信頼性が得られます。
- データベース外への格納。XML 文書をデータベース外に格納する場合でも、Oracle の機能を使用して、索引付けおよび問合せを行い、また BFILE、URL およびテキストベースの索引付けを使用して、文書を効率的に取り出すことができます。

Oracle Text の索引付けによる要素内容のファイングレイン検索

Oracle では、外部ドキュメントを指す URL に加えて、LOB 列での Oracle Text（*inter*Media Text）索引の作成が可能です。この索引付けメカニズムは、XML データも処理できます。

Oracle8*i* および Oracle データベースは、XML タグを認識します。また、XML 要素内容を検索するために、セクションおよびサブセクション・テキストも認識します。そのため、問合せを非構造化データに対して実行し、ドキュメント内の特定のセクションまたは要素に限定することができます。

Oracle Text の例 : CONTAINS を使用したテキストおよび XML データの検索

次の Oracle Text（*inter*Media Text）の例では、適切な索引がすでに作成されていると想定しています。

```
SELECT *
FROM   purchaseXMLTab
WHERE  CONTAINS(po_xml,"street WITHIN addr") >= 1;
```

> **参照：** Oracle Text の詳細は、第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

構成済（作成済）XML の格納のメリット

XML 文書の構造が不明または動的である場合は、CLOB に格納するのが最適です。

構成済 XML の格納のデメリット

多くの SQL 機能は、オブジェクト・リレーショナル列で使用できません。更新など、特定の操作の並行性が低下する場合があります。ただし、ドキュメントの完全コピーが保存されます。

マッピングの細分化を改善するためのハイブリッドな XML 格納方法の使用

前述の項では、次のことを説明しました。

- 構造化 XML 文書（生成される XML）をオブジェクト・リレーショナル・インスタンスにマップする方法
- 構成済 XML 文書（作成済 XML）を LOB に格納する方法

多くの場合、マッピングの細分化をより適切に制御する必要があります。

たとえば、本などのテキスト・ドキュメントを XML でマッピングする場合、すべての単一の要素を拡大し、オブジェクト・リレーショナルとして格納することを回避する必要があります。このようなドキュメントの場合、フォント情報および段落情報をオブジェクト・リレーショナル形式で格納しても、問合せに有効ではありません。

一方、テキスト・ドキュメント全体を CLOB に格納した場合、ドキュメント全体に SQL 問合せを効率的に実行できなくなります。

ハイブリッドな格納によるユーザー定義の細分化での格納

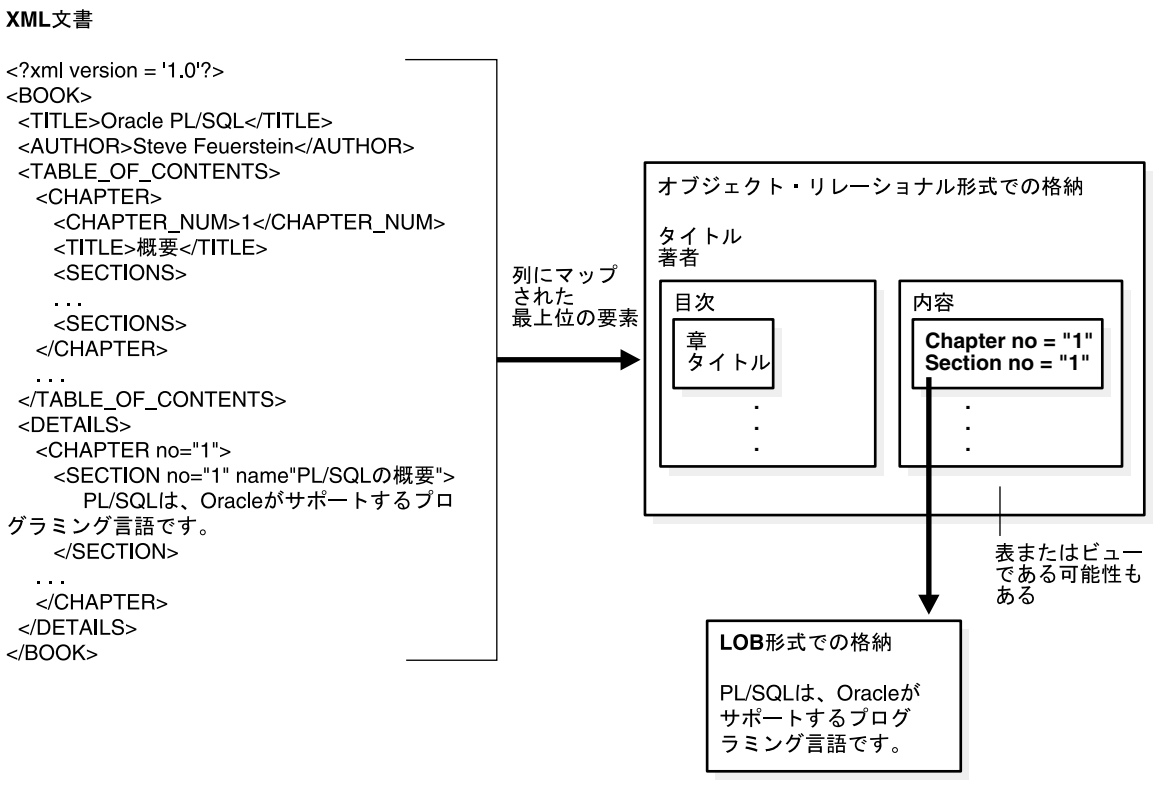
前述の問題を解決するには、ユーザー定義の細分化を利用して格納します。本の例では、次の操作を行う必要があります。

- 章、項、タイトルなどの最上位の要素は、問合せのためにオブジェクト・リレーショナル表に格納します。
- 本の各項の内容は CLOB に格納します。

表の定義時に、マッピングの細分化を指定できます。サーバーは様々なソースから XML を自動的に作成し、問合せを適切に構成できます。

図 2-1 に、この XML のハイブリッドな格納方法を示します。

図 2-1 XML のハイブリッドな格納方法 : 内容が CLOB に格納されている場合に、表の最上位要素を問い合せる例



ハイブリッドな格納のメリット

次に、XML 文書をハイブリッドな方法で格納するメリットを示します。

- 問合せ可能で有効な情報をオブジェクト・リレーショナル形式で格納でき、文書全体が分解されません。
- 文書全体が分解されないため、文書の再作成の時間が短縮されます。
- LOB に格納された部分の文書がテキスト検索できます。

生成される XML の変換

データベースから生成された XML は正規の形式であり、列が要素にマップされ、オブジェクト型がネストした要素にマップされます。ただし、状況によっては、アプリケーションでの XML 文書の表示を変更する必要がある場合があります。

XML 文書構造に変換が必要な場合

構造化された XML 文書の構造に基礎となるデータベース・スキーマの構造との互換性がない場合、データベースにデータを書き込む前に、正しい形式に変換する必要があります。これを行うには、次のいずれかの方法を実行します。

- XSL スタイルシートまたは他のプログラミング方法を使用します。
- データ中心の XML 文書を完全なシングル・オブジェクトとして格納します。
- 様々な XML 文書構造に対応するオブジェクト・ビューを定義し、INSTEAD OF トリガーを定義して、ベース・データの適切な変換および更新を行います。

ビューを使用した XML 文書とデータの結合

構造化 XML データと非構造化データが組み合されたときに、そのデータ全体を表示および操作する必要がある場合、Oracle のビューを使用します。

ビューを使用すると、様々な方法で格納された XML データを組み合せることによって、その場でオブジェクトを作成できます。次の操作を行うことができます。

- 従業員データ、顧客データなどの構造化データをオブジェクト・リレーショナル表内の 1 つの場所に格納します。
- 説明やコメントなどの関連した非構造化データを CLOB 内に格納します。

データ全体を取り出す必要がある場合、ビューの SELECT 文に型コンストラクタを使用して、データの様々なフラグメントから構造を単純に作成します。作成すると、XSU で、ビューから構造化データを単一の XML 文書として取り出せます。

変換の索引付けおよび問合せ

XML 文書の変換されたビューに索引を作成し、問合せを実行する必要がある場合があります。たとえば、XML メッセージ機能環境では、発注書メッセージのフォーマットが異なる場合があります。特定の問合せをすべての発注書メッセージに対して実行できるように、発注書メッセージを正規に問い合せる必要がある場合があります。

この場合、問合せは文書の変換されたビューに対して実行されます。ファンクション索引を作成するか、または通常のビューを使用して、これを実行できます。

索引付けの方法

XML 文書の解析、パス検索の実行およびフラグメントの抽出を行うには、`extract()` および `existsNode()` メンバー関数を固有に実装します。ただし、これはパフォーマンスを向上させる、スケーラブルなソリューションではありません。

2 つ目の方法は、Oracle Text（*inter*Media Text）の索引付けを使用することです。

> **参照：** 第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

また、拡張索引付けインフラストラクチャを使用して、`XMLType` 列に対して独自の索引付けメカニズムを構築することもできます。

> **参照：** 『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

XML Schema および XML 文書のマッピング

W3C は、スキーマ・ワーキング・グループを設立しました。スキーマ・ワーキング・グループの目的は、既存の DTD ベースのメカニズムの拡張として、構造化スキーマおよびデータ型に対する新しい XML ベースの表記法を提供することです。次に、XML Schema の使用目的を示します。

- XML Schema1: 文書構造（要素、属性、名前空間）の制約
- XML Schema2: 内容（データ型、エンティティ、表記法）の制約

データ型自体は、基本形（バイト、日付、整数、順序、間隔など）の場合もユーザー定義型（既存のデータ型から導出された型、およびベース型の範囲、精度、長さ、マスクなど、特定のプロパティを制約できる型）の場合もあります。アプリケーション固有の制約および記述も可能です。

XML Schema は、要素、属性およびデータ型定義の継承を提供します。標準の明確な構造体セマンティクスを簡単に利用するために、URI 参照用のメカニズムが提供されています。埋込み文書またはコメント用に、スキーマ言語も提供されています。

たとえば、次の例に示すとおり、単純なデータ型を定義できます。

XML Schema の例 1: 単純なデータ型の定義

次に、XML Schema で単純なデータ型を定義する例を示します。

```
<datatype name="positiveInteger"
            basetype="integer"/>
   <minExclusive> 0 </minExclusive>
</datatype>
```

この例からわかるように、XML Schema は DTD に対して、ベース型や最小値の制約など、多くの重要な新規の構造体を提供します。

データベースから動的データが生成された場合、通常、データベース型で表されます。Oracle では、これは前述のオブジェクト・リレーショナル型のシステムです。このシステムでは、NULL、NUMBER(7,2) などの変数精度、チェック制約、ユーザー定義型、継承、型間の参照、コレクション型のように、データ型がより精密になります。XML Schema は様々なスキーマ制約を利用できるため、データの基礎となる型によるシステムに対して、生成された文書の一致率が向上します。

XML Schema の例 2: 基礎となるスキーマへの生成された XML 文書のマップ

XML Schema で表された単純な発注書の型について考えてみます。

```
<type name="Address" >
   <element name="street" type="string" />
   <element name="city"   type="string" />
   <element name="state"  type="string" />
   <element name="zip"    type="string" />
</type>

<type name="Customer">
   <element name="custNo"
                    type="positiveInteger"/>
   <element name="custName" type="string" />
   <element name="custAddr" type="Address" />
</type>

<type name="Items">
   <element name="lineItem" minOccurs="0" maxOccurs="*">
    <type>
      <element name="lineItemNo" type="positiveInteger" />
      <element name="lineItemName" type="string" />
      <element name="lineItemPrice" type="number" />
      <element name="LineItemQuan">
        <datatype basetype="integer">
          <minExclusive>0</minExclusive>
        </datatype>
      </element>
```

```
    </type>
   </element>
</type>

<type name="PurchaseOrderType">
    <element name="purchaseNo"
                    type="positiveInteger" />
    <element name="purchaseDate"  type="date" />
    <element name="customer" type="Customer" />
    <element name="lineItemList"  type="Items" />
</type>
```

これらの XML Schema は、2-9 ページの「XML Schema の例 2: 基礎となるスキーマへの生成された XML 文書のマップ」で説明されているオブジェクト・リレーショナルでの発注書の例と一致するように作成されています。XML Schema で提案された構造体と SQL:1999 ベースの型システムの類似性が重要です。そのような類似性によって、XML Schema をデータベースのオブジェクト・リレーショナル・スキーマにマップし、前述のスキーマに従って、有効な文書をデータベース・スキーマ内の行オブジェクトに比較的簡単にマップできます。DTD より優れた XML Schema の表現力によって、マッピングが大幅に簡単になっています。

XML Schema が提供するスキーマ制約は、データ駆動のアプリケーション以外にも適用できます。動的な動作を示すドキュメント駆動のアプリケーションは、増加の一途をたどっています。

- 単純な例にはメモがあります。メモは、マークアップ・タグに基づいて様々な方法で転送されます。
- 高度な例には、大陸間を横断する航空機のテクニカル・サービス・マニュアルがあります。XML Schema が提供する複合制約に基づいて、このようなマニュアルの著者が必ず有効な部品番号を入力するようにできます。また、部品番号の妥当性を、インベントリ・レベル、需要と供給の変動基準、法的規制の変更などに対して動的にチェックするようにすることもできます。

一般的な XML: データ交換アプリケーションにおける設計の問題

この項では、データを交換するアプリケーションの、次の XML 設計問題について説明します。

- データベースに格納された XML データからの Web フォームの生成
- Web フォームからデータベースへの XML データの送信

データベースに格納された XML データからの Web フォームの生成

Web フォームのインフラストラクチャを生成するには、次の操作を行います。

1. XSU を使用して、問合せ中の基礎となる表のスキーマに基づいて、DTD を生成します。
2. 生成された DTD を XML Class Generator for Java への入力として使用します。XML Class Generator for Java は DTD 要素に基づいて、一連のクラスを生成します。
3. これらのクラスを使用する Java コードを記述し、Web ベースのフォームのインフラストラクチャを生成します。このインフラストラクチャに基づいて、Web フォームではユーザー・データが獲得され、データベース・スキーマと互換性のある XML 文書が作成されます。このデータは、追加の処理を行わずに関連のデータベース表またはオブジェクト・ビューに直接書き込むことができます。

Web フォームからデータベースへの XML データの送信

Web フォームを介して取得されたデータを基礎となるデータベース・スキーマにマップする 1 つの方法は、Web フォームおよびその基礎となる構造を設計し、スキーマと互換性のある DTD に基づいて、XML データを生成することです。この項では、XSU および XML Parser for Java を使用してこれを行う方法について説明します。次に、この処理の流れを示します。

1. Java アプリケーションが XSU を使用して、ターゲットのオブジェクト・ビューまたは表の形式と一致する DTD を生成します。
2. アプリケーションは、この DTD を XML Class Generator for Java に対して指定します。XML Class Generator for Java は、ユーザーに提示された Web フォームを設定するためのクラスを構築します。
3. 生成されたクラスを使用して、JSP、Java サーブレットまたはその他のコンポーネントによって、Web フォームが動的に構築されます。
4. ユーザーがフォームに記入してそれを送信すると、サーブレットがデータを適切な XML データ構造にマップし、XSU がデータをデータベースに書き込みます。

XSU の DTD 生成機能を使用して、ターゲットのオブジェクト・ビューまたは表にとって適切な XML 形式を判断できます。これを行うには、`SELECT * FROM` をオブジェクト・ビューまたは表に実行して、XML を生成します。

この結果では、DTD 情報が個別ファイルとして出力されるか、または XML ファイルの最初の DOCTYPE タグ内に埋め込まれます。

この DTD を XML Class Generator for Java への入力として使用し、DTD 要素に基づいて、一連のクラスを生成します。これらのクラスを使用する Java コードを記述し、Web ベースのフォームのインフラストラクチャを生成します。この結果、Web フォームを介して送信されたデータが、データベースに書込み可能な XML 文書に変換されます。

アプリケーション間の XML 文書の送信

アプリケーション間で XML 文書を送信する方法は数多くあります。この項では、一般的ないくつかの方法について説明します。

ここでは、次の内容を想定しています。

- 送信アプリケーションが XML 文書を送信します。
- 受信アプリケーションが XML 文書を受信します。

ファイル転送 : 受信アプリケーションが、FTP、NFS、SMB または他のファイル転送プロトコルを介して、送信アプリケーションから XML 文書を要求します。文書が、受信アプリケーションのファイル・システムにコピーされます。アプリケーションが、ファイルを読み込み、処理します。

HTTP: 受信アプリケーションが、サーブレットに HTTP 要求を行います。サーブレットが、XML 文書を受信アプリケーションに戻し、受信アプリケーションが読込みおよび処理を行います。

Web フォーム : 送信アプリケーションが、Web フォームをレンダリングします。ユーザーは、ブラウザ内で実行する Java アプレットまたは Java スクリプトを介してフォームに記入し、情報を送信します。アプレットまたは Java スクリプトが、ユーザーのフォームを XML 形式で受信アプリケーションに転送し、受信アプリケーションが読込みおよび処理を行います。受信アプリケーションが最終的にデータをデータベースに書き込む場合、送信アプリケーションはデータベースと互換性のある形式の XML を作成する必要があります。Oracle の XML 製品を使用してこれを行う方法については、2-11 ページの「Web フォームからデータベースへの XML データの送信」を参照してください。

AQ: Oracle データベースが、ネット・サービスの HTTP および JDBC を介して、XML 文書を 1 つ以上の受信アプリケーションに送信します。XML 文書が、Oracle AQ を介してメッセージとして送信されます。受信アプリケーションが、XML メッセージをデキューし、処理します。

参照： 次の章およびマニュアルを参照してください。

- 第 9 章「Oracle AQ を使用した XML データの交換」
- 『Oracle Integration Server Overview』
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』

データベースへの XML のロード

次のオプションを使用して、XML データまたは DTD ファイルを Oracle にロードできます。

- DBMS_LOB などの LOB に対する PL/SQL ストアド・プロシージャの使用
- Java（Pro*C、C++）カスタム・コードの記述
- SQL*Loader の使用
- Oracle *inter*Media の使用
- XSU

Oracle CM SDK を使用して、XML 文書をデータベースに挿入することもできます。ただし、DTD はサポートされません。DTD に代わる標準である XML Schema がサポートされます。

SQL*Loader の使用

SQL*Loader を使用すると、LOB をバルク・ロードできます。

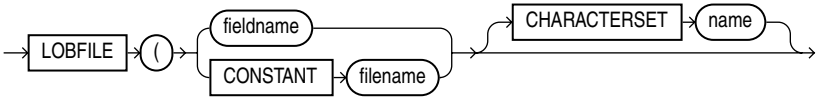
参照：

- SQL*Loader を使用した LOB のロードの詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。
- SQL*Loader の使用の概要および例については、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』の第 4 章の「LOB ロード時の SQL*Loader の使用」を参照してください。

SQL*Loader を使用した LOB への XML 文書のロード

LOB は非常に大きい場合があるため、SQL*Loader は、LOB データをメイン・データ・ファイル（残りのデータとともに行内にある）または LOBFILE からロードできます。図 2-2 に、LOBFILE 構文を示します。

図 2-2 LOBFILE 構文



LOB データは、非常に長いため、LOBFILE からロードすることが有効です。LOBFILE でも、LOB データ・インスタンスはフィールド内（事前にサイズが決まっている、デリミタ付き、長さ値）にあるとみなされますが、これらのフィールドはレコードに編成されません（LOBFILE 内にはレコードの概念が存在しません）。そのため、レコードを処理するオーバーヘッドが回避されます。このタイプのデータ編成は LOB のロードに最適です。

LOBFILE の LOB はメモリーに収まる必要はありません。SQL*Loader は、LOBFILE を 64KB のチャンクで読み取ります。64KB より大きい物理レコードをロードするには、`READSIZE` パラメータを使用して、より大きいサイズを指定できます。

XMLType 列、または CLOB 内に XML データを含む列をロードするには、LOBFILE の使用が最適です。

- **XML が妥当な場合 :** LOBFILE 内の XML データが大きく、妥当な XML であることがわかっている場合は、ダイレクト・パス・ロードを使用します。これによって、すべての XML 妥当性検証処理が回避されます。
- **XML の検証が必要な場合 :** XML データの妥当性検証が必要な場合は、従来型パス・ロードを使用します。ただし、これはダイレクト・パス・ロードより非効率的であることに注意してください。

従来型パス・ロードは、SQL `INSERT` 文を実行して、表を Oracle データベースに移入します。ダイレクト・パス・ロードは、Oracle データ・ブロックをフォーマットし、データ・ブロックをデータベース・ファイルに直接書き込むことによって、ほとんどの Oracle データベースのオーバーヘッドを排除します。

ダイレクト・パス・ロードでは、他のユーザーとのデータベース・リソースの競合が発生しません。そのため、通常、ディスク速度に近い速度でデータをロードできます。制限事項、セキュリティ、バックアップ問題などのダイレクト・パス・ロード固有の考慮点については、『Oracle9*i* データベース・ユーティリティ』の第 9 章を参照してください。

図 2-3 に、SQL*Loader のダイレクト・パス・ロードおよび従来型パス・ロードを示します。

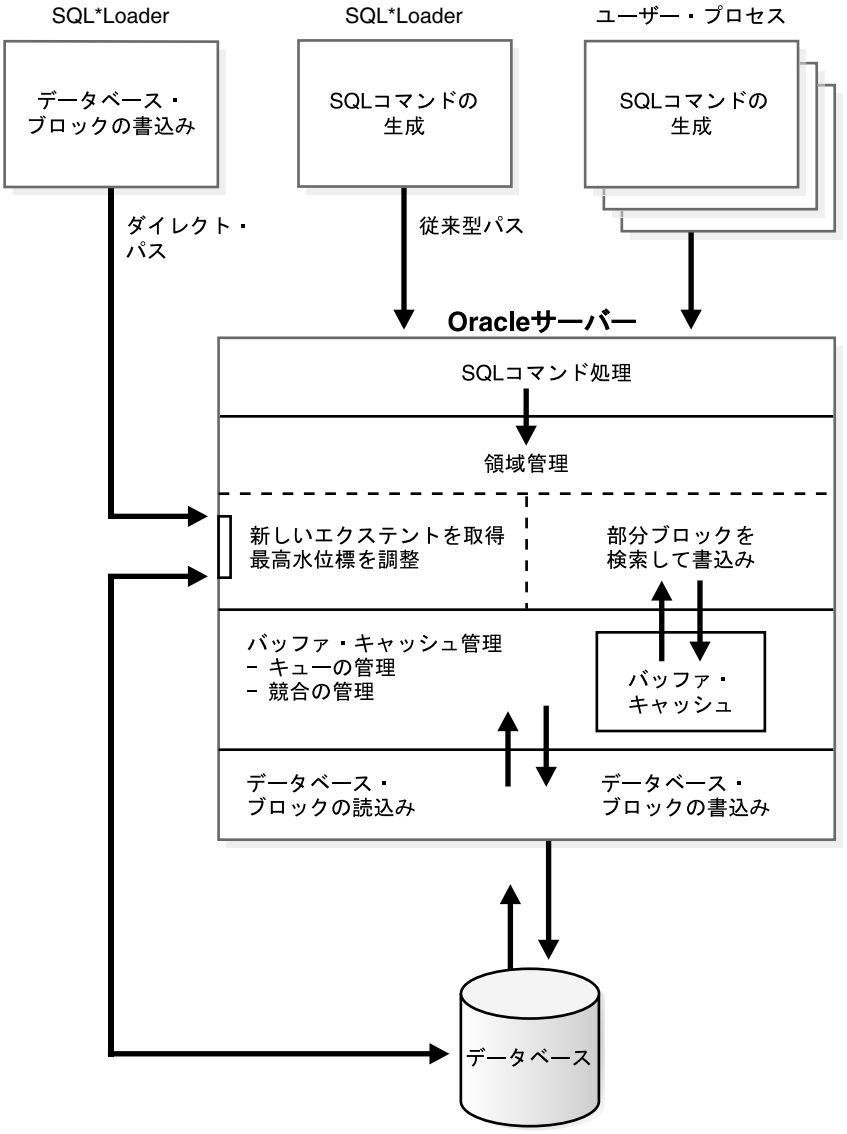
ロードする表は、データベース内の既存の表である必要があります。SQL*Loader は表を作成しません。SQL*Loader は、すでにデータを含んでいるか、または空である既存の表をロードします。

ロードには、次の権限が必要です。

- ロードする表に対する INSERT 権限が必要です。
- 表に新しいデータをロードする前に、REPLACE または TRUNCATE オプションを使用してその表内の古いデータを空にする場合は、ロードする表に対する DELETE 権限が必要です。

> **参照：** ロードおよびその例の詳細は、『Oracle9*i* データベース・ユーティリティ』の第 7 章および第 9 章を参照してください。

図 2-3 SQL*Loader: ダイレクト・パス・ロードおよび従来型パス・ロード



Oracle の XML 対応テクノロジを使用するアプリケーション

インターネット・アプリケーションでは、XML を様々な方法で使用できます。次に、Oracle の XML コンポーネントの使用に適した 2 つのデータベースを使用したアプリケーションの分野を示します。

- データ表示のカスタマイズを含む「XML 対応テクノロジによるコンテンツおよびドキュメントの管理」
- システム間またはシステム内アプリケーション間でデータを交換するための「B2B および B2C メッセージ機能」

また、Oracle の XML コンポーネントは、これらを組み合せて使用する場合にも適しています。このマニュアルでは、これらの 2 つのアプリケーション分野を中心に、それぞれ第 III 部「XML を使用したデータ交換」および第 IV 部「Oracle ベースの XML アプリケーションを構築するためのツール製品およびフレームワーク」で説明します。

この章では、それぞれのアプリケーション分野を、代表的な使用例を基に説明します。

XML 対応テクノロジによるコンテンツおよびドキュメントの管理

データ表示のカスタマイズ

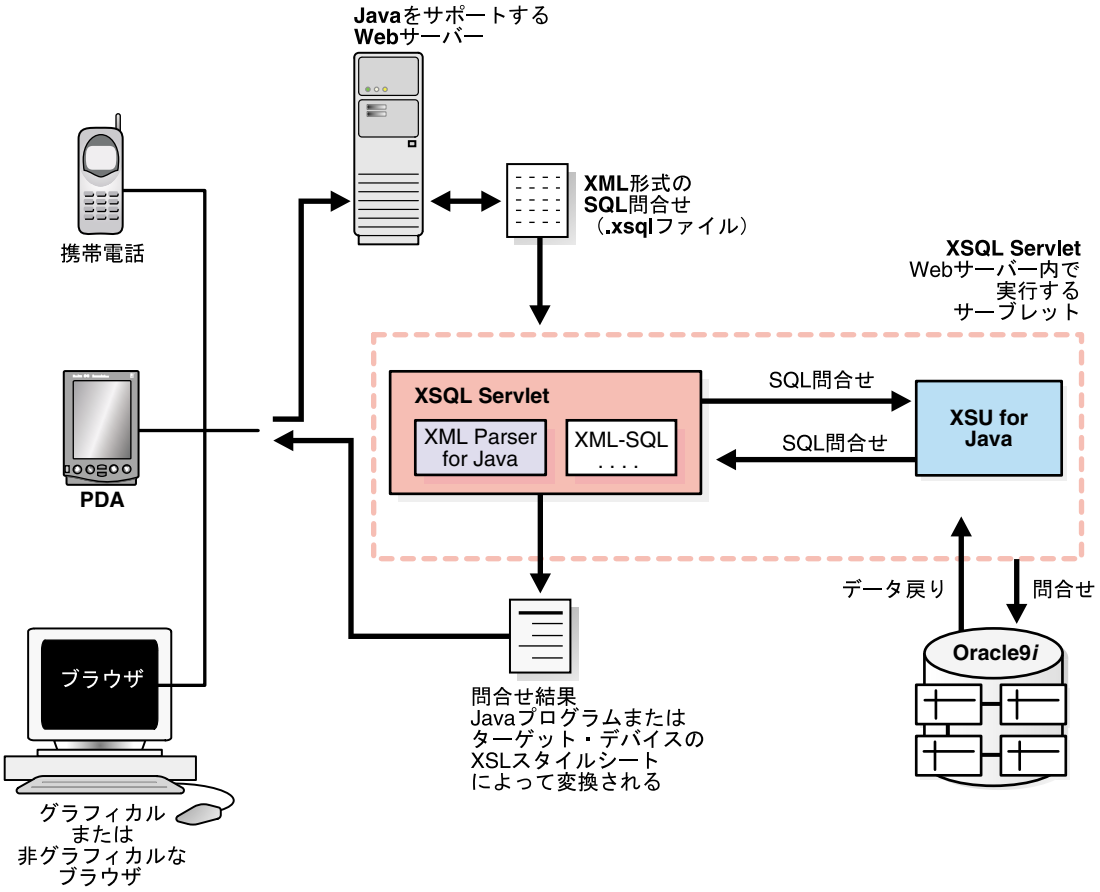
様々なブラウザ、デバイスおよびユーザーに合せてデータ表示をカスタマイズするための XML の使用が増大しています。XML 文書を XSL スタイルシートとともにクライアント、中間層またはサーバーで使用すると、個々のユーザー用にカスタマイズされた XML データを、次のような様々なクライアント・デバイスに合せて変換、編成および表示できます。

- グラフィカルおよび非グラフィカルな Web ブラウザ
- Palm Pilot などの PDA
- デジタル式の携帯電話およびポケットベル

これによって、様々な出力デバイスへの対応が容易になり、ビジネス・オペレーションに集中してビジネス・アプリケーションを作成できます。

XML および XSL を使用すると、より簡単に動的 Web サイトを作成および管理できます。XSL スタイルシートを変更すると、基礎となるビジネス・ロジックまたはデータベース・コードを変更することなく、ルックアンドフィールを変更できます。新しいユーザーおよびデバイスをターゲットにする場合、必要に応じて、新しい XSL スタイルシートを設計するのみです。図 2-4 を参照してください。

図 2-4 コンテンツ管理 : 表示のカスタマイズ



参照： 第 20 章「XML Parser for Java の使用」を参照してください。

Oracle の XML コンポーネントを使用した次のコンテンツ管理の使用例について考えてみます。

- 使用例 1 - コンテンツおよびドキュメントの管理 : Oracle の XML 対応テクノロジを使用した複合ドキュメントの公開
- 使用例 2 - コンテンツおよびドキュメントの管理 : Oracle の XML テクノロジを使用した個人情報の配信
- 使用例 3 - コンテンツ管理 : Oracle の XML テクノロジを使用したデータ駆動のアプリケーションのカスタマイズ

各使用例では、ビジネス上の問題、ソリューション、主な実行タスクおよび使用する Oracle の XML コンポーネントについて簡単に説明します。

これらの使用例の詳細は、『Oracle9*i* ケース・スタディ -XML アプリケーション』の「XML を使用したコンテンツおよびドキュメントの管理」を参照してください。

使用例 1 - コンテンツおよびドキュメントの管理 : Oracle の XML 対応テクノロジを使用した複合ドキュメントの公開

問題点

X 社には、断片的な SGML および XML 文書のドキュメント・リポジトリが多数あります。複合ドキュメントは、動的に公開される必要があります。

ソリューション

データベース・アプリケーションの設計を、適切なデータベースを設計することから始める必要があります。X 社は、まず適切なデータ・モデル化および設計のガイドラインを使用する必要があります。これによって、データに対するオブジェクト・ビューをより簡単に作成できます。

XMLType を使用して、ドキュメントを XML 形式で格納します。この形式では、リレーショナル・データを更新できます。Oracle CM SDK をデータ・リポジトリ・インタフェースとして使用します。Oracle CM SDK は、XML データ・リポジトリ管理および管理タスクの実装に有効です。

X 社は、XSL スタイルシートを使用して、ドキュメントのセクションまたはフラグメントを組み合せ、その複合ドキュメントをユーザーに電子送信できます。単一のソーシングおよびオーサリング、または複数チャネルの公開に、Arbortext および EPIC を使用することをお薦めします。複数チャネルの公開によって、同一ドキュメントを HTML、PDF、WORD、ASCII テキスト、SGML、FrameMaker などの様々なフォーマットで簡単に生成できるようになります。図 2-5 を参照してください。

> **参照：** Arbortext および EPIC 製品の詳細は、http://www.arbortext.com を参照してください。

主な実行タスク

使用例 1 のソリューションで実行される主なタスクは次のとおりです。

1. データベースを設計します。使用する XML タグおよび要素を決定します。
2. これらのセクションまたはフラグメントを、データベースの CLOB 内の XMLType 列に格納します。
3. XSL スタイルシートを作成して、セクションまたはフラグメントを完全なドキュメントにレンダリングします。

使用する Oracle の XML コンポーネント

- XML パーサーおよび XSLT

  第 20 章「XML Parser for Java の使用」または第 24 章「XML Parser for C の使用」を参照してください。

- XSQL Servlet

  第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

- XSU（セクションまたはフラグメントをデータベース内外に移動させます）

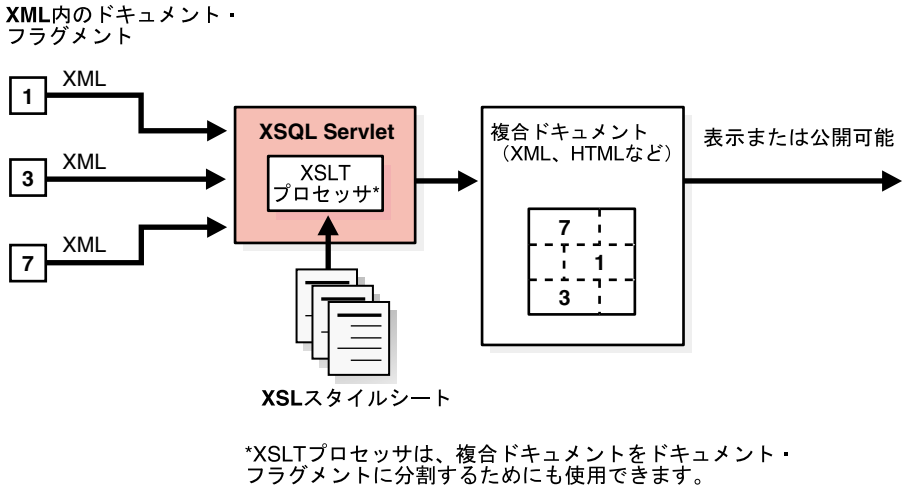
  第 7 章「XSU」を参照してください。

- Oracle Text（データ検索アプリケーションを拡張します）

  第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

- XMLType（CLOB）への XML の格納については、第 5 章「XML に対するデータベース・サポート」を参照してください。

図 2-5 使用例 1 - XML を使用した複合ドキュメントの作成および公開



使用例 2 - コンテンツおよびドキュメントの管理 : Oracle の XML テクノロジを使用した個人情報の配信

問題点

大規模なニュース配信元は、様々なニュース・ソースからデータを受信します。このデータは、データベースに格納され、ニュース配信元との契約に従って、配信者およびユーザーがいつでも特定のニュースおよびカスタマイズ済ニュースを参照できるように、必要に応じてすべての配信者およびユーザーに送信する必要があります。配信元は、XSL を使用してデータを正規化し、データベースに格納します。格納されたデータは、いくつかの Web サイトおよびポータルで使用されます。これらの Web サイトおよびポータルは、様々な有線および無線クライアントからの HTTP 要求を受信します。

ソリューション

XSL スタイルシートおよび XSQL Servlet を使用して、要求サービスに対して、動的に適切なレンダリングを行います。図 2-6 を参照してください。

主な実行タスク

使用例 2 で実行される主なタスクは次のとおりです。

1. 出力が最適化されるように、データベース・スキーマのデータ・モデルを設計します。
2. 各情報ソース用の XSL スタイルシートを作成して、正規化されたフォーマットに変換します。変換後、それをデータベースに格納します。
3. XSL スタイルシートを XSQL ページとともに作成して、データを Web サイト上で公開します。

使用する Oracle の XML コンポーネント

- XML Parser for Java

  第 20 章「XML Parser for Java の使用」を参照してください。

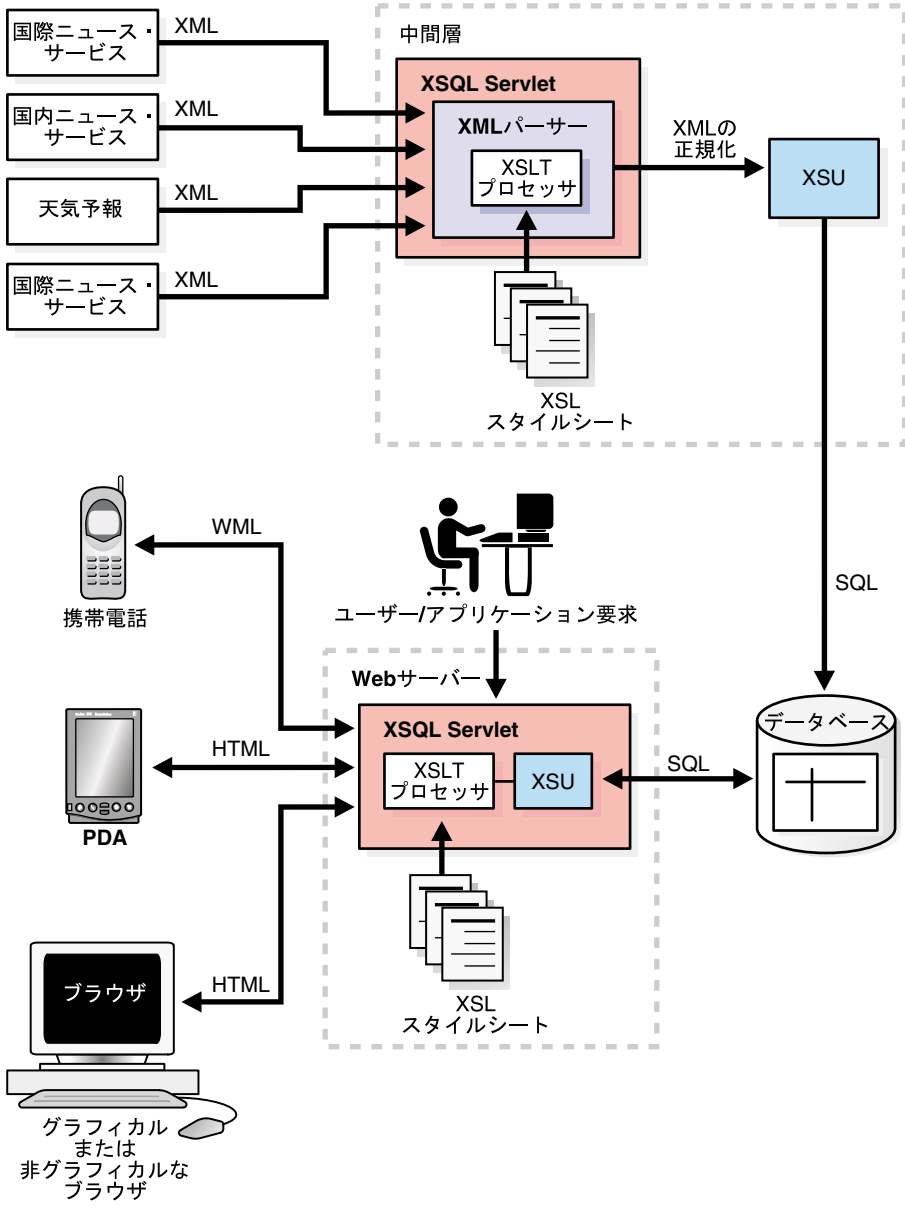
- XSU

  第 7 章「XSU」を参照してください。

- XSQL Servlet

  第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

図 2-6 使用例 2 - Oracle の XML コンポーネントを使用したカスタマイズ済ニュース情報の配信



使用例 3 - コンテンツ管理 : Oracle の XML テクノロジを使用したデータ駆動のアプリケーションのカスタマイズ

問題点

X 社では、相互的にデータを Thin クライアントに配信する必要があります。

ソリューション

問合せは、クライアントからデータベースに送信され、その出力が 1 つ以上の XSL スタイルシートを介して動的にレンダリングされて、クライアント・アプリケーションに送信されます。データは、リレーショナル・データベースに XML 形式で LOB として格納されます。

使用する Oracle の XML コンポーネント

- XML Parser for Java および XSLT プロセッサ

  第 20 章「XML Parser for Java の使用」および第 21 章「XML Schema Processor for Java の使用」を参照してください。

- LOB への XML の格納については、第 5 章「XML に対するデータベース・サポート」を参照してください。

B2B および B2C メッセージ機能

ビジネス・アプリケーション開発者にとっての課題は、異なるベンダーおよび異なるアプリケーション・ドメインのアプリケーションによって生成されたデータを結合することです。Oracle の XML 対応テクノロジは、データを特定のネットワークまたは通信プロトコルと結び付けずに、データおよびそのコンテキストに焦点を当てることによって、アプリケーション間のデータ交換を容易にします。

XML および XSL 変換を使用することで、アプリケーションは、独自または非互換のデータ形式を管理および解析せずに、データ交換できます。

Oracle の XML コンポーネントを使用した次の B2B/B2C メッセージ機能の使用例について考えてみます。

- 使用例 4 - B2B メッセージ機能 : XML を使用した複数サプライヤ用のオンライン・ショッピング・カート設計
- 使用例 5 - B2B メッセージ機能 : オンライン在庫管理アプリケーションのための Oracle の XML コンポーネントおよび AQ の使用
- 使用例 6 - B2B メッセージ機能 : Oracle の XML 対応テクノロジおよび AQ を使用した複数アプリケーションの統合

各使用例では、問題、ソリューション、問題を解決するために実行する主なタスク、および使用する Oracle の XML コンポーネントについて簡単に説明します。

使用例 4 - B2B メッセージ機能 : XML を使用した複数サプライヤ用のオンライン・ショッピング・カート設計

問題点

X 社は、様々なサプライヤから製品を購入するためのオンライン・ショッピング・カートを構築する必要があります。X 社は、注文書をオンラインで受信し、注文された製品に基づいて、適切なサプライヤに注文書を送信する必要があります。

ソリューション

XML を使用して、統合されたオンライン購入アプリケーションを作成します。ユーザーは、新しいハードウェアの新規注文書に記入するときに、直接コンピュータ・メーカーの Web サイトに進み、最新モデル、構成オプションおよび価格を参照できます。ユーザーのサイトから、注文書の参照番号および認証情報をサプライヤの Web サイトに送信します。

ユーザーは、サプライヤのサイトで商品をショッピング・カートに追加した後、ボタンをクリックしてショッピングを完了します。サプライヤは、ショッピング・カートの内容を、ユーザーが選択した部品番号、数量および価格情報を含む XML ファイルとして X 社のアプリケーションに返送します。

ショッピング・カート内の商品は、新規注文書に明細項目として自動的に追加されます。

顧客の注文書（XML 形式）は、適切なサプライヤのデータベースに送信されて、処理されます。適切な転送を行うために、XSL を使用して、ショッピング・カートを変換および分割します。データは、リレーショナル・データベースに XML 形式で格納されます。図 2-7 を参照してください。

> **参照：** 同様の実装例については、『Oracle9*i* ケース・スタディ -XML アプリケーション』の第 8 章「オンライン B2B XML アプリケーション : 手順」を参照してください。

使用する Oracle の XML コンポーネント

- XML パーサー

  第 20 章「XML Parser for Java の使用」を参照してください。

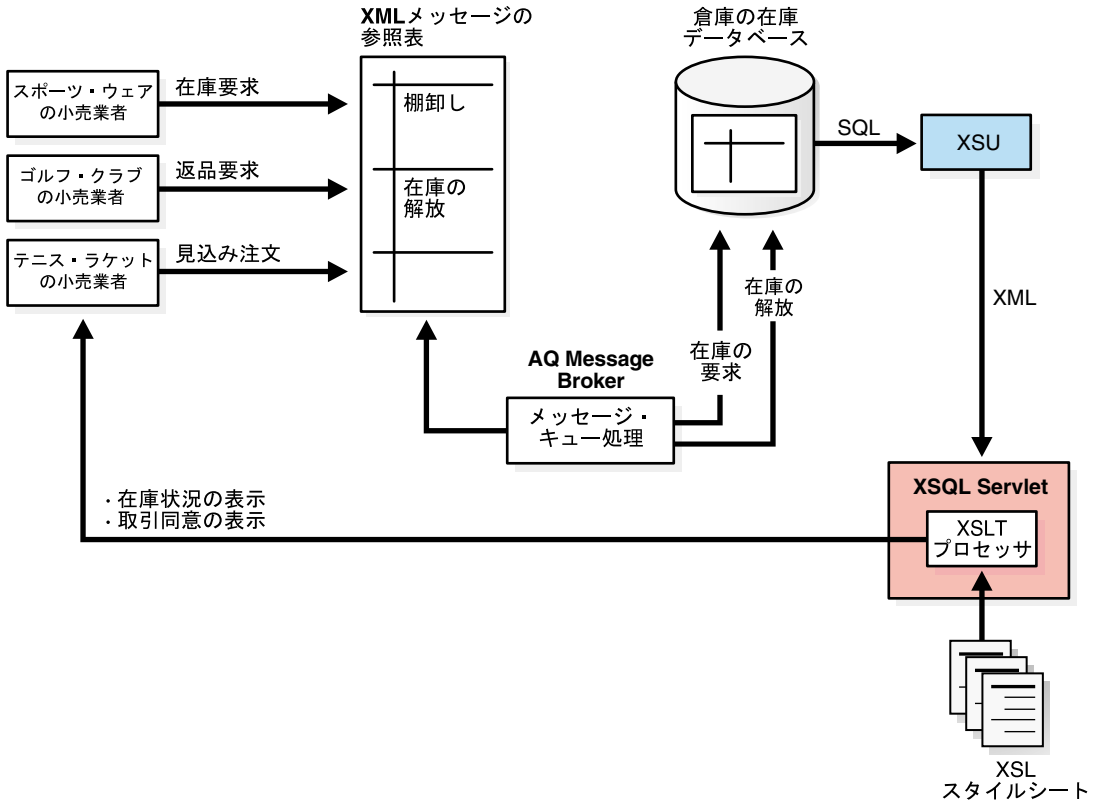
- XSU

  第 7 章「XSU」を参照してください。

- XSQL Servlet

  第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

図 2-7 使用例 4 - Oracle の XML コンポーネントを使用した複数ベンダー用のオンライン・ショッピング・カート設計



使用例 5 - B2B メッセージ機能 : オンライン在庫管理アプリケーションのための Oracle の XML コンポーネントおよび AQ の使用

問題点

クライアント / サーバーおよびサーバー / サーバーのアプリケーションは、データのリソースおよびインベントリをデータベース・リポジトリに格納します。このリポジトリは、企業間で共有されます。X 社は、データ・リソースがアクセスされるたびに、データがアクセスされたことを知る必要があります。また、システム上のすべてのユーザーおよび顧客は、データがアクセスされた時間および場所を知る必要があります。

ソリューション

リソースがアクセスまたは解放されると、使用可能であることを示す XML メッセージが生成されます。次に、XSL を使用して、必要に応じてリソースを複数のクライアント形式に変換します。逆に、1 つのクライアントがリソースを取得すると、XML メッセージが他のクライアントに送信され、取消しが通知されます。メッセージは LOB に格納されます。データは、リレーショナル・データベースに格納され、XML で具体化されます。図 2-8 を参照してください。

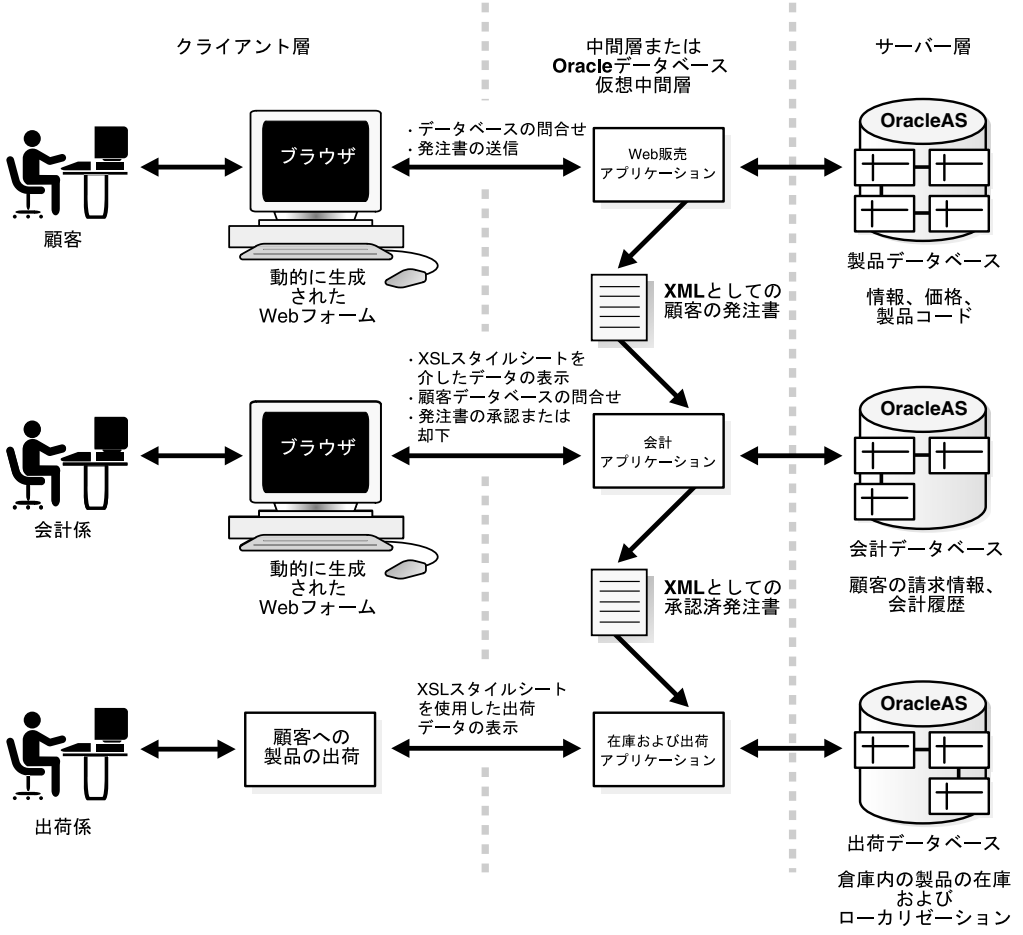
使用する Oracle の XML コンポーネント

- XML パーサーおよび XSLT プロセッサ

  第 20 章「XML Parser for Java の使用」、第 24 章「XML Parser for C の使用」、第 26 章「XML Parser for C++ の使用」または第 29 章「XML Parser for PL/SQL の使用」を参照してください。

- LOB への XML データの格納については、第 5 章「XML に対するデータベース・サポート」を参照してください。

図 2-8 使用例 5 - オンライン在庫管理アプリケーションでの Oracle の XML コンポーネントおよび AQ の使用



使用例 6 - B2B メッセージ機能 : Oracle の XML 対応テクノロジおよび AQ を使用した複数アプリケーションの統合

問題点

X 社は、ビジネスのワークフローおよびプロセスを統合するために、複数のアプリケーションを通信させ、データを共有させる必要があります。

ソリューション

XML をメッセージ・ペイロードとして使用します。XML は XSLT プロセッサを介して変換され、メッセージに含まれて転送されます。XML メッセージは、AQ Broker データベースに LOB として格納されます。Oracle Workflow を使用すると、メッセージの管理、データのルーティングおよび変換が簡単になります。このソリューションでは、コンテンツ管理（XSL スタイルシートを使用した表示のカスタマイズ）も使用します。図 2-9 を参照してください。

主な実行タスク

1. ユーザーまたはアプリケーションが、要求を発信します。結果データが、XSU を使用して企業のデータベースから取り出されます。
2. データは、XSLT プロセッサによって変換され、AQ Broker に送信されます。
3. AQ Broker は、このメッセージを読み取り、必要なアクションを決定します。AQ Broker は、アプリケーション 1、2 および 3 に適切に応答しその後の処理を行います。

使用する Oracle の XML コンポーネント

- XML パーサーおよび XSLT プロセッサ

  第 20 章「XML Parser for Java の使用」を参照してください。

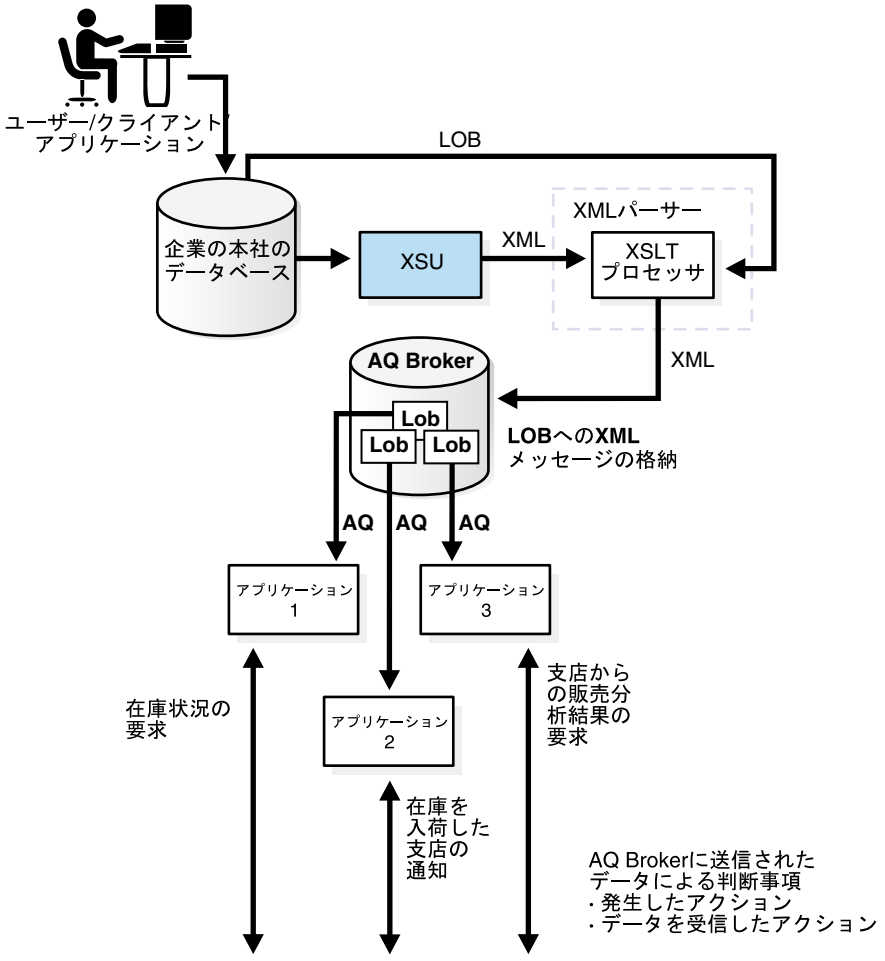
- XSU

  第 7 章「XSU」を参照してください。

- AQ

  第 9 章「Oracle AQ を使用した XML データの交換」を参照してください。

図 2-9 使用例 6 - Oracle の XML コンポーネントおよび AQ を使用した複数アプリケーションの統合



3

XDK および XML コンポーネント : 概要および一般的な FAQ

この章の内容は次のとおりです。

Oracle の XML コンポーネント : 概要

この章では、XML コンポーネントの概要について説明します。

Oracle は、XML テクノロジを使用して Web ベースのデータベース・アプリケーションを構築するために使用できる、いくつかのコンポーネント、ユーティリティおよびインタフェースを提供します。コンポーネントの使用基準は、アプリケーション要件、プログラミング作業環境、開発環境および配置環境に依存します。

次の XML コンポーネントは、Oracle および OracleAS に付属しています。

- **XDK:** Oracle XDK には、Java、C、C++ および PL/SQL 用があります。これらの開発キットには、XML 文書の読取り、操作、変換および表示を行うためのコンポーネントが含まれます。Oracle XDK はフル・サポートされており、商用再配布ライセンスを受けています。表 3-1 に、XDK コンポーネントを示します。
- **XSU:** このユーティリティには、Java 用および PL/SQL 用があります。XSU は、XML データを SQL 問合せ、結果セットまたは表のデータベースから生成し、またデータベースに格納します。このユーティリティは、SQL 問合せの結果を XML に（またはその反対に）正規にマッピングすることによって、データ変換を行います。

次の図に、XDK コンポーネントを使用して XML を生成する方法の概念を示します。

- 図 3-7「XDK for Java を使用した XML 文書の生成」
- 図 3-8「XDK for C を使用した XML 文書の生成」
- 図 3-9「XDK for C++ を使用した XML 文書の生成」
- 図 3-10「XDK for PL/SQL を使用した XML 文書の生成」

表 3-1 XDK コンポーネントの説明

XDK コンポーネント	言語	説明
XML パーサー	Java、C、C++、PL/SQL	業界標準の DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。
XSLT プロセッサ	Java、C、C++、PL/SQL	XML を、HTML や WML など他のテキストベースのフォーマットに変換またはレンダリングします。
XML Schema Processor	Java、C、C++	XML Schema 定義によって XML の単純型および複合型を使用可能にします。
XML Class Generator	Java、C++	DTD および XML Schema から自動的に Java クラスおよび C++ クラスを生成し、Web フォームまたはアプリケーションから XML データを送信します。
XML Transviewer JavaBean	Java	Java コンポーネントを使用して、XML 文書および XML データを表示および変換します。
XSU	Java、PL/SQL	SQL 問合せから、XML 文書、DTD および XML Schema を生成します。
XSQL Servlet	Java	サーバー内の XML、SQL および XSLT を組み合せて、動的 Web コンテンツを配信します。

開発ツールおよび XML 対応のその他の機能

XML 対応の開発ツールは次のとおりです。

- **Oracle Text（*inter*Media Text/Context）:** 問合せ、検索および取出しのためのツールです。詳細は、第 8 章「Oracle Text を使用した XML データの検索」を参照してください。
- **Oracle JDeveloper9*i* および BC4J:** JDeveloper9*i* は、Web ベースの Java アプリケーションを作成するための統合開発ツールです。Oracle BC4J は、Pure Java 対応で XML ベースのフレームワークです。このフレームワークによって、再利用可能なビジネス・コンポーネントから、複数層でデータベースを使用するアプリケーションの高生産性開発、移植可能な配置、および柔軟なカスタマイズが可能となります。このようなアプリケーション・サービスは、CORBA サーバー・オブジェクトか EJB Session Beans のいずれかとして、Java テクノロジをサポートする企業規模のサーバー・プラットフォーム上に配置できます。

  参照： 次の章を参照してください。

  - 第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」
  - 第 12 章「BC4J および XML アプリケーションの作成」

- **Oracle CM SDK:** データをドキュメントとして表示し、そのドキュメントをデータとして処理できるアプリケーション・インタフェースです。Oracle CM SDK を使用すると、開発者は容易に XML で作業ができます。この場合、Oracle CM SDK は XML のレポジトリとして機能します。Oracle CM SDK を使用すると、XML 文書で次のタスクを実行できます。
  - 自動的に XML を解析し、コンテンツを表および列に格納します。
  - XML ファイルのコンテンツをレンダリングします。

    参照： 『Oracle9*i* ケース・スタディ -XML アプリケーション』の「Oracle Internet File System を使用した XML アプリケーションの作成」を参照してください。

- **Oracle Reports:** Oracle Reports Developer および Oracle Reports Server を使用すると、動的に生成された高品質な Web レポートを作成および公開できます。主要なタスクは、ウィザードを使用して簡単に行えます。また、レポート・テンプレートおよびライブ・データ・プレビューを使用して、簡単にレポート構造をカスタマイズできます。標準の Web ブラウザを介して、HTML、HTML カスケーディング・スタイルシート（HTML CSS）、Adobe 社の PDF、デリミタ付きテキスト、Rich Text Format（RTF）、ポストスクリプト、PCL、XML などの選択した任意のフォーマットで、企業内でレポートを公開できます。また、Oracle Reports は Oracle Portal（WebDB）と統合できます。
  - レポートをスケジューリングして定期的に実行し、Oracle Portal サイトの情報を更新できます。さらに、レポートをユーザー用にパーソナライズできます。
  - Oracle Reports Developer は、Oracle の E-Business Intelligence Solution の一部であり、Oracle Discoverer および Oracle Express と統合されています。

    参照： 第 14 章「OracleAS Reports Services および XML」を参照してください。

XDK for Java

XDK for Java は、次のコンポーネントで構成されます。

- **XML Parser for Java:** 業界標準である DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。XML を XML またはその他のテキストベース・フォーマット（HTML など）に変換する **XSLT プロセッサ**を搭載しています。
- **XML Schema Processor for Java:** 単純型および複合型をサポートし、Oracle XML Parser for Java V2 に組み込まれています。
- **XML Class Generator for Java:** XML DTD または XML Schema 定義から、ソース・ファイルを作成します。

- **XSQL Servlet:** XSQL ファイル（拡張子は .xsql）に埋め込まれた SQL 問合せを処理します。結果を XML 形式で戻します。XSU および XML Parser for Java を使用します。
- **XSU for Java:** オブジェクト・リレーショナル・データベースの表またはビューから取り出したデータを XML に変換し、XML 文書からデータを抽出して、次のタスクを実行できます。
  - 正規マッピングを使用して、表やビューの適切な列や属性にデータを挿入します。
  - このデータを適用して、適切な列または属性の値を更新または削除します。

XDK for JavaBeans

XDK for JavaBeans は、次のコンポーネントで構成されます。

- **XML Transviewer Beans:** Java を介して XML 文書およびデータを表示および変換します。

XDK for C

XDK for C は、次のコンポーネントで構成されます。

- **XML Parser for C:** 業界標準である DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。XML を XML またはその他のテキストベース・フォーマット（HTML など）に変換する **XSLT プロセッサ**を搭載しています。

XDK for C++

XDK for C++ は、次のコンポーネントで構成されます。

- **XML Parser for C++:** 業界標準である DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。XML を XML またはその他のテキストベース・フォーマット（HTML など）に変換する **XSLT プロセッサ**を搭載しています。
- **XML Schema Processor for C++:** XML Parser for C++ とともに動作するコンポーネントです。Oracle データベースの XML アプリケーションで、単純型および複合型をサポートします。Schema Processor は、XML Schema の草案をサポートします。
- **XML C++ Class Generator:** XML DTD または XML Schema 定義から、ソース・ファイルを作成します。

XDK for PL/SQL

XDK for PL/SQL は、次のコンポーネントで構成されます。

- **XML Parser for PL/SQL:** 業界標準である DOM インタフェースおよびSAXインタフェースを使用して、XML を作成および解析します。XML を XML または他のテキストベース・フォーマット（HTML など）に変換する **XSLT プロセッサ**を搭載しています。
- **XSU for PL/SQL:** オブジェクト・リレーショナル・データベース表またはビューから取り出したデータを XML に変換し、XML 文書からデータを抽出して、次のタスクを実行できます。
  - 正規マッピングを使用して、表やビューの適切な列や属性にデータを挿入します。
  - このデータを適用して、適切な列または属性の値を更新または削除します。

XML パーサー

Oracle XML Parser には、Oracle データベースが実行するすべてのプラットフォーム用の C、C++、PL/SQL および Java の実装が含まれます。

`xml.com` 誌の適合性テストにおいて、Oracle XML Parser は、SAX および DOM の両インタフェースをサポートし、XML 1.0 仕様に準拠している妥当な XML パーサーとして、2 位以内にランクされました。SAX インタフェースおよび DOM インタフェースは、W3C 勧告 2.0 に準拠しています。

Oracle XML Parser は、次の機能を統合的にサポートします。

- XPath: XPath は、W3C 勧告であり、XSLT、Xlink および XML Query で使用される XML 文書をナビゲートするためのデータ・モデルおよび文法を指定します。
- ドキュメント・ノードの段階的な XSLT: XSLT は、W3C 勧告のバージョン 1.0 に準拠しています。これをサポートすると、次の機能が使用可能になります。
  - 他の XML 構造への XML 文書の変換
  - 他のテキストベース・フォーマットへの XML 文書の変換

Oracle XML Parser は、すべての Oracle プラットフォームで使用できます。

図 3-1 に、Oracle XML Parser for Java を示します。また、図 3-2 に、Oracle XML Parser の機能の概要を示します。

> **参照：** 第 20 章「XML Parser for Java の使用」および付録 C「XDK for Java: 仕様および早見表」を参照してください。

図 3-1 Oracle XML Parser for Java

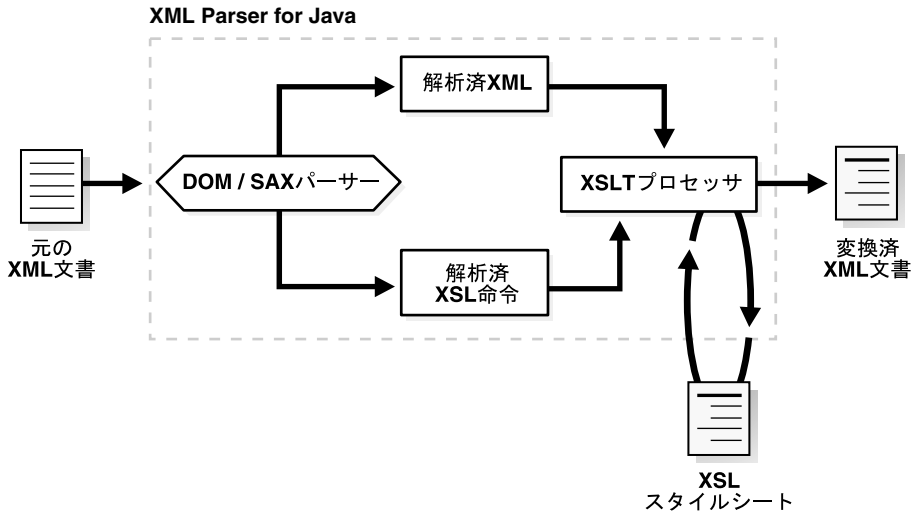
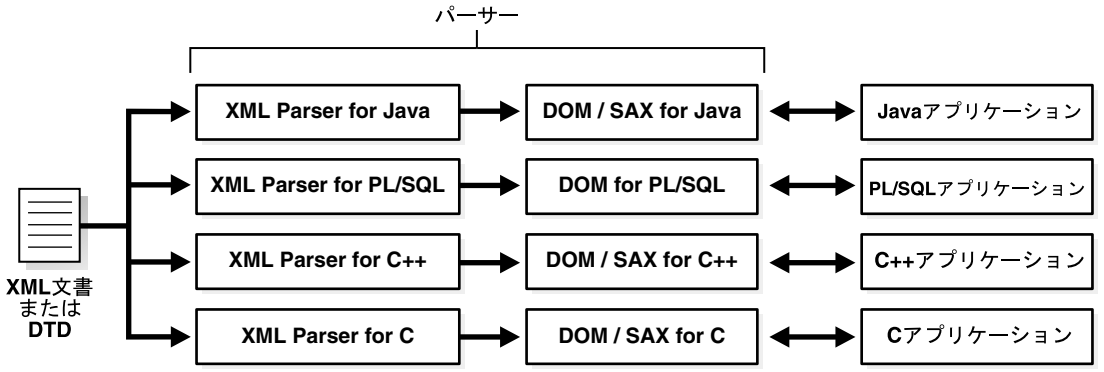


図 3-2 Oracle XML Parser: Java、C、C++、PL/SQL



XSLT プロセッサ

Oracle の XSLT エンジンは、W3C の 1.0 勧告である XSL Transformation をフル・サポートしています。この XSLT エンジンには、次の機能があります。

- すべてのプラットフォーム上で、データベース内外の XML 情報に対して、標準に基づいた変換を行います。
- Java の拡張性をサポートします。また、パフォーマンス向上のため、システム固有のエンジンとして Oracle8*i* リリース 8.1.7 以上にネイティブ・コンパイルされています。

Oracle XML Parser には、XSL スタイルシートを使用して XML データを変換するための統合化された XSLT プロセッサが含まれます。XSLT プロセッサを使用すると、XML 文書を XML から XML、HTML など、すべてのテキストベースのフォーマットに変換できます。

XSLT プロセッサの使用方法については、第 20 章「XML Parser for Java の使用」を参照してください。

参照： 付録 C「XDK for Java: 仕様および早見表」を参照してください。

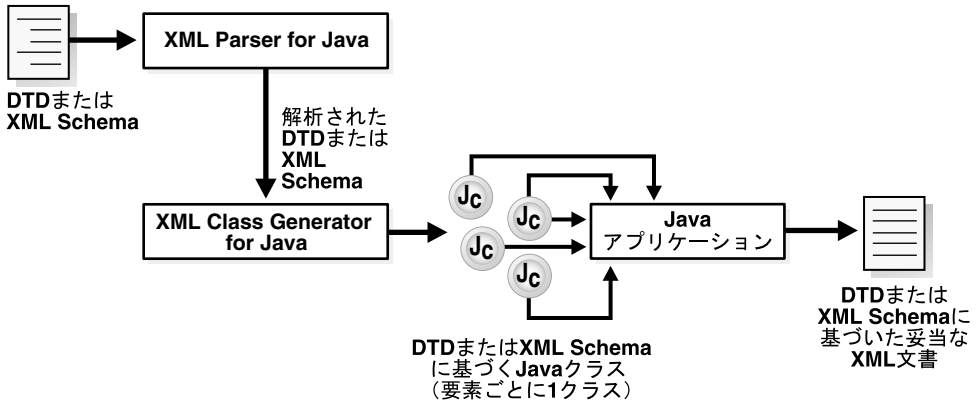
XML Class Generator

XML Class Generator は、入力 DTD または XML Schema に対応する XML 文書を作成するための一連の Java クラスまたは C++ クラスを作成します。図 3-3 に、Oracle XML Class Generator の機能の概要を示します。

XML Class Generator の使用方法については、次の章を参照してください。

- 第 22 章「XML Class Generator for Java の使用」
- 第 28 章「XML C++ Class Generator の使用」

図 3-3 Oracle XML Class Generator for Java



XML Transviewer Beans

Oracle XML Transviewer Beans は、JavaBeans 用の XML を構成する一連の XML コンポーネントです。これらのコンポーネントは、Java のアプリケーションまたはアプレットで XML 文書を表示および変換するために使用されます。

これらは、Oracle JDeveloper と統合できる、XML ベースのデータベース・アプリケーションの作成および配置を高速化する、ビジュアルおよび非ビジュアルの Java コンポーネントです。今回のリリースでは、次の 7 つの Bean が使用可能です。

- **DOMBuilder Bean**

  Java の XML（DOM）パーサーを Bean インタフェースでラップし、複数のファイルを同時に解析（非同期解析）できるようにします。リスナーを登録すると、Java アプリケーションは大規模または連続したドキュメントを解析し、制御をすぐにコール元に戻すことができます。

- **XMLSourceViewer Bean**

  XML 文書を表示可能にする JPanel を拡張した Bean です。XML および XSL 構文をカラーで強調することによって、XML および XSL ファイルの表示を改善します。また、編集アプリケーションを使用して XML 文書を変更する場合に有効です。この Bean は簡単に DOMBuilder Bean と統合でき、指定した DTD に対する事前および事後の解析および妥当性検証を行うことができます。

- **XMLTreeViewer Bean**

  XML パーサーを拡張および無効にする機能によって、XML 文書をツリー形式で表示できるように JPanel を拡張した Bean です。これは、XML 文書を DOM ツリーとして表示して、ユーザーがマウスを使用して簡単にツリーを操作し、選択されたブランチを非表示にしたり、表示できるようにします。

- **XSLTransformer Bean**

  XSLT プロセッサを Bean インタフェースでラップし、XSL スタイルシートに基づいて、XML 文書を XSL 変換します。これによって、XSL スタイルシートを適用して、XML 文書を XML、HTML、DDL など、すべてのテキストベースのフォーマットに変換できます。この Bean を他の Bean と統合した場合、アプリケーションまたはユーザーは、変換の結果をすぐに表示できます。XSLTransformer Bean は、サーバー側のアプリケーションまたはサーブレットが XML 文書（問合せ結果の XML 表現など）をブラウザで表示するために HTML でレンダリングするための基礎として使用できます。

- **XMLTransformPanel Bean**

  他の Bean を使用して、XML ファイルを処理できるサンプル・アプリケーションを作成します。この Bean には、XML 文書および XSL スタイルシートをロードするためのファイル・インタフェースが含まれます。これは、次の Bean を使用します。

  – XML 文書を表示および編集するための Bean

  – スタイルシートを XML 文書に適用し、その出力を表示するための変換用の Bean

- **DBAccess Bean**

- **DBViewer Bean**

これらの Bean は、標準の JavaBeans として、Oracle JDeveloper などの、すべてのグラフィカル Java 開発環境で使用できます。Oracle XML Transviewer Beans の機能については、第 23 章「XML Transviewer Beans の使用」を参照してください。

Oracle XSQL Page Processor および Oracle XSQL Servlet

XSQL Servlet は、SQL 問合せを処理し、結果セットを XML として出力するツール製品です。このプロセッサは Java サーブレットとして実装され、埋込み SQL 問合せを含む XML ファイルを入力として受け入れます。このプロセッサは、XML Parser for Java、XSU および Oracle の XSLT エンジンを使用して、多くの操作を行います。

XSQL Servlet を使用して、次のタスクを実行できます。

- 1 つ以上の SQL 問合せ結果から動的な XML データページを構築し、サーバー側の XSLT 変換を使用して、その結果を XML データグラムまたは HTML ページとして Web に表示します。

- Web サーバーに送信された XML を受信し、データベースに挿入します。

XSQL Servlet をサポートするサーブレット・コンテナ

XSQL Servlet は、次のサーブレット・コンテナでテスト済です。

- Allaire JRun 2.3.3
- Apache 1.3.9 with JServ 1.0 および 1.1
- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Servlet Engine
- Caucho Resin 1.1
- NewAtlanta ServletExec 2.2 for IIS/PWS 4.0
- Oracle9*i* Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1（JSP Patch 搭載）
- Oracle8*i* 8.1.7 Beta Aurora および Oracle9*i* 以上の Servlet Engine
- Sun JavaServer Web Development Kit（JSWDK）1.0.1 Web Server

XSQL Servlet をサポートする JSP プラットフォーム

JSP は、`<jsp:forward>` または `<jsp:include>` を使用し、アプリケーションの一部として、XSQL Pages とともに動作します。次の JSP プラットフォームでは、XSQL Servlet のサポートをテスト済です。

- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Tomcat 3.1 Beta1 Servlet Engine
- Caucho Resin 1.1（ビルトイン JSP 1.0 サポート）
- NewAtlanta ServletExec 2.2 for IIS/PWS 4.0（ビルトイン JSP 1.0 サポート）
- Oracle9*i* Lite Web-to-Go Server および Oracle JSP 1.0
- Oracle8*i* 8.1.7 Beta Aurora および Oracle JSP 1.0 搭載の Oracle9*i* 以上の Servlet Engine
- すべての Servlet Engine with Servlet API 2.1+ および Oracle JSP 1.0

通常、これは、次のものとともに動作します。

- Servlet 2.1 以上の仕様をサポートするすべてのサーブレット・エンジン
- Oracle JSP 1.0 リファレンス実装またはそれと同等の機能を持つその他のベンダー製品

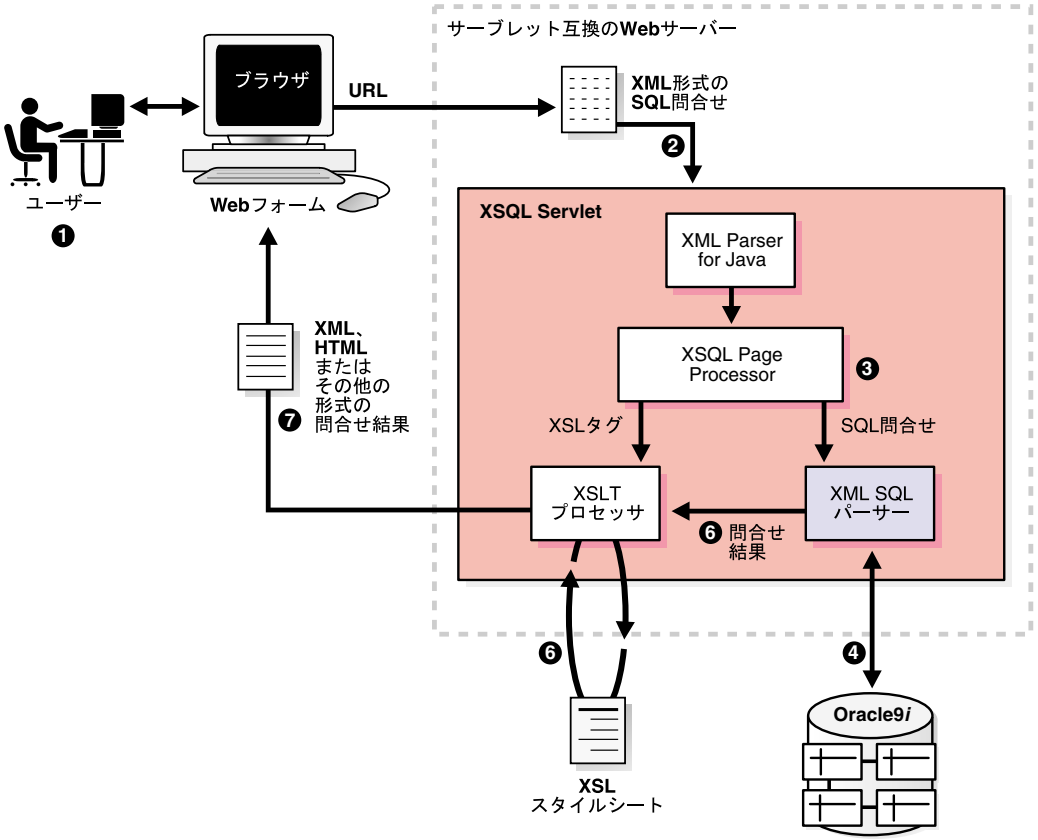
XSQL Servlet は、SQL 問合せを処理し、結果セットを XML として出力するツール製品です。このプロセッサは Java サーブレットとして実装され、埋込み SQL 問合せを含む XML ファイルを入力として受け入れます。これは、XML Parser for Java および XSU を使用して、多くの操作を行います。

図 3-4 に、クライアントからサーブレット、およびサーブレットからクライアントへのデータ・フローを示します。次に、イベントの順序を示します。

1. ユーザーが、ブラウザを介して URL を入力します。この URL は解析され、Web サーバーを介して XSQL Servlet に渡されます。この URL には、ターゲットの XSQL ファイル（.xsql）の名前が含まれます。また、オプションで、値や XSL スタイルシート名などのパラメータが含まれます。また、ブラウザおよび Web サーバーを介さずに、コマンドラインから XSQL Servlet を起動することもできます。
2. サーブレットが、XSQL ファイルを XML Parser for Java に渡します。XML Parser for Java は、XML を解析し、XML コンテンツにアクセスするための API を提供します。
3. サーブレットのページ・プロセッサ・コンポーネントがこの API を使用して、XML のパラメータおよび SQL 文（`<xsql:query></xsql:query>` タグで囲まれた部分）を XSU に渡します。ページ・プロセッサは、すべての XSL 処理文も XSLT プロセッサに渡します。
4. XSU が、Oracle データベースに SQL 問合せを送ります。このデータベースは、問合せ結果を XSU に戻します。
5. XSU が、問合せ結果を XML 形式のテキストとして XSLT プロセッサに戻します。結果は、XML ファイル内の元の `<xsql:query>` タグと同じ場所に埋め込まれます。
6. XSLT プロセッサが、必要に応じて、指定された XSL スタイルシートを使用して問合せ結果およびその他の XML データを変換します。データは、HTML、またはスタイルシートで定義されたその他の形式に変換できます。XSLT プロセッサは、最初に URL を要求したクライアントのタイプに基づいて、異なるスタイルシートを選択して適用できます。この `HTTP_USER_AGENT` の情報は、HTTP 要求を介してクライアントから取得されます。
7. XSLT プロセッサが、ユーザーに表示する完成したドキュメントをクライアントのブラウザに戻します。

> **参照：** 第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

図 3-4 XSQL Servlet の機能



Oracle XSU

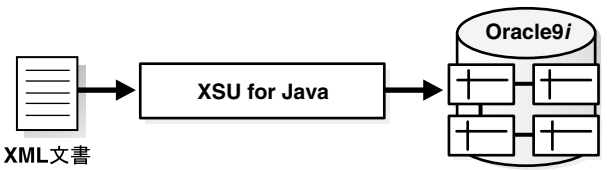
Oracle XSU は、Java および PL/SQL をサポートします。

- XSU は、任意の SQL 問合せを自動的かつ動的に正規の XML にレンダリングするためのコア Java クラス・ライブラリで構成されます。このユーティリティには、次の機能が含まれます。
  - 豊富な構造のユーザー定義オブジェクト型およびオブジェクト・ビューでの問合せをサポートします。
  - 正規の構造を持つ XML を既存の表、ビュー、オブジェクト表またはオブジェクト・ビューに自動的に挿入します。XSLT 変換と組み合せることによって、ほぼすべての XML 文書をデータベースに自動的に挿入できます。

  XSU の Java クラスは、次のタスクに使用できます。
  - SQL 問合せまたは結果セット・オブジェクトから、テキスト、XML 文書、DOM、DTD、または XML Schema を生成します。
  - XML 文書のデータを既存のデータベース・スキーマまたはビューにロードします。
- XSU for PL/SQL は、XSU for Java をラップする PL/SQL パッケージで構成されます。

図 3-5 に、Oracle XSU の機能の概要を示します。

図 3-5 Oracle XSU の機能



XSU for Java は、次のタスクを実行する一連の Java クラスで構成されます。

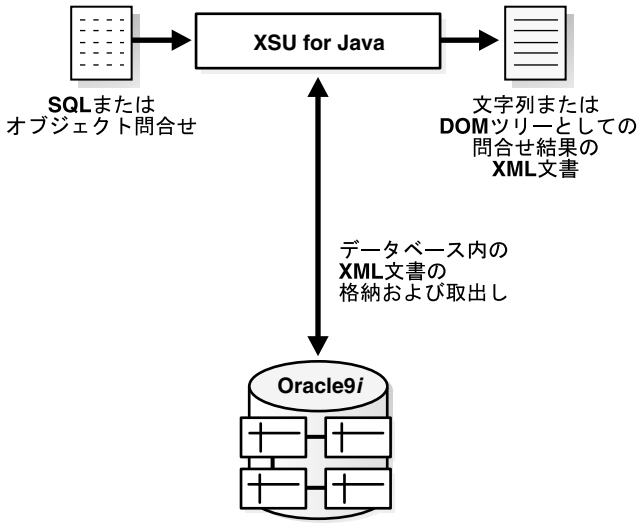
- 問合せをデータベースに渡し、結果または DTD から XML 文書（テキストまたは DOM）を生成します。DTD は、検証に使用できます。
  - XML データをデータベース表に書き込みます。

    参照： 第 7 章「XSU」を参照してください。

問合せ結果からの XML の生成

図 3-6 に、XSU が SQL 問合せを処理し、結果を XML 文書として戻す方法を示します。

図 3-6 XSU が SQL 問合せを処理し、結果を XML 文書として戻す方法



XML 文書の構造 : 要素への列のマップ

結果として戻る XML 文書の構造は、次に示すとおり、問合せ結果を戻すデータベース・スキーマの内部構造に基づいています。

- 列は、最上位の要素にマップします。
- スカラー値は、内容がテキストのみの要素にマップします。
- オブジェクト型は要素にマップされ、オブジェクト型の属性はサブ要素を構成します。
- コレクション型は、要素のリストにマップします。

XSU による文字列または DOM 要素ツリーとしての XML 文書の生成

XSU は、次のいずれかを生成します。

- XML 文書の文字列表現

  XML 文書をリクエスタに戻す場合、この表現を使用してください。

- DOM ツリー

  プログラムで XML を操作する場合、この表現を使用してください。たとえば、DOM メソッドで XML を検索または変更する XSLT プロセッサを使用して XML を変換する場合です。

XSU による問合せ対象の表のスキーマに基づいた DTD の生成

XSU を使用して、問合せ対象の表またはビューのスキーマに基づいて、DTD を生成することもできます。生成された DTD を Java または C++ 用の XML Class Generator への入力として使用します。XML Class Generator は、DTD 要素に基づいて一連のクラスを生成します。その後、これらのクラスを使用する Java コードを作成し、Web ベースのフォームのインフラストラクチャを生成します。3-8 ページの「XML Class Generator」も参照してください。

このインフラストラクチャに基づいて、Web フォームではユーザー・データが獲得され、データベース・スキーマと互換性のある XML 文書が作成されます。このデータは、追加の処理を行わずに関連するデータベース表またはオブジェクト・ビューに直接書き込むことができます。

> **参照：** この方法の詳細は、第 7 章「XSU」および『Oracle9*i* ケース・スタディ -XML アプリケーション』の「オンライン B2B XML アプリケーション : 手順」を参照してください。

> **注意：** XML データが基礎となる表の構造と一致しない場合に XML 文書をデータベース表に書き込むには、その XML 文書を変換してから、データベースに書き込んでください。これを行う方法については、第 7 章「XSU」を参照してください。

Oracle Text

Oracle Text（*inter*Media Text）では、Oracle に格納されるすべてのテキストまたはドキュメントに索引付けすることによって、Oracle データベースを拡張します。Oracle Text を使用し、XML をプレーン・テキストとして索引付けすることによって、Oracle データベースに格納された XML 文書を検索できます。また、WITHIN タイトル検索（タイトルはドキュメントのセクション）など、より正確な検索を行うために、XML をドキュメント・セクションとして索引付けすることもできます。

> **参照：** Oracle Text および XML の使用の詳細は、第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

Oracle の XML コンポーネント : XML 文書の生成

図 3-7 および図 3-10 に、Oracle の XML コンポーネントの関係、およびそれらが連携して Oracle から SQL 問合せを介して XML 文書を生成する方法を示します。次の言語別にオプションを示します。

- Java
- C
- C++
- PL/SQL

Oracle の XML コンポーネントを使用した XML 文書の生成 : Java

図 3-7 に、XML 文書の生成に使用される Java 用の Oracle の XML コンポーネントを示します。次に、使用可能な Java 用 XML コンポーネントを示します。

- XDK for Java:
  - XSLT を含む XML Parser for Java
  - XML Schema Processor for Java
  - XML Class Generator for Java
  - XSQL Servlet
  - XML Transviewer Beans
- XSU for Java

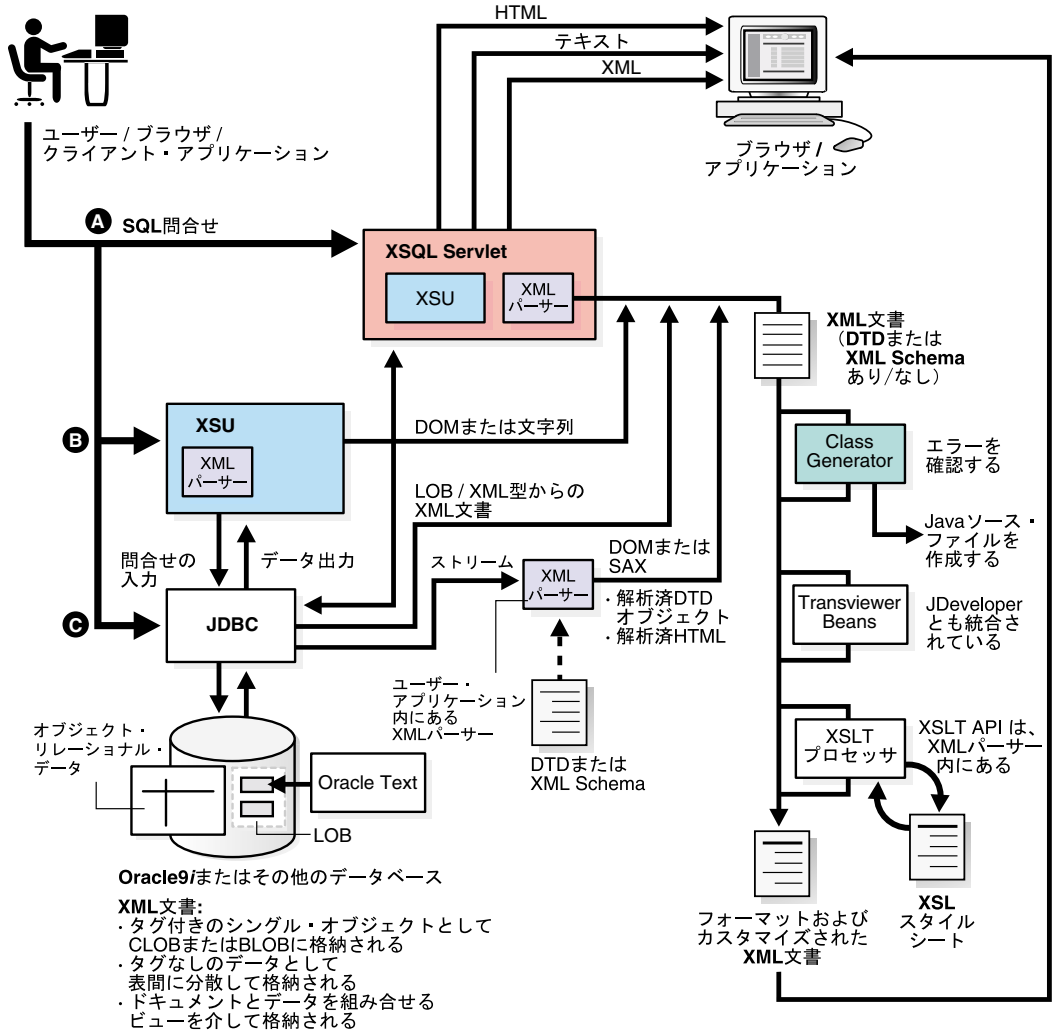
Java 環境では、ユーザー、クライアントまたはアプリケーションが問合せ（SQL）を送信した場合、Oracle の XML コンポーネントを使用した次の 3 つの方法でその問合せを処理できます。

- XSL Servlet による処理（XSU および XML パーサーの使用も含む）
- XSU による直接処理（XML パーサーの使用も含む）
- JDBC による直接処理（JDBC が XML パーサーにアクセスします）

格納された XML 変換済データがデータベースからどのように生成されたかにかかわらず、結果として戻る XML パーサーからの XML 文書出力は、ユーザーまたはユーザーが使用するアプリケーションの XML 文書の用途に応じて、追加処理されます。

XML 文書は、スタイルシートの適用によってフォーマットおよびカスタマイズされ、この際に XSLT によって処理されます。

図 3-7 XDK for Java を使用した XML 文書の生成



Oracle の XML コンポーネントを使用した XML 文書の生成 : C

図 3-8 に、XML 文書の生成に使用される C 言語用の Oracle の XML コンポーネントを示します。次に、使用可能な C 言語用の XML コンポーネントを示します。

- XML Parser/XSLT Processor for C
- XML Schema Processor for C

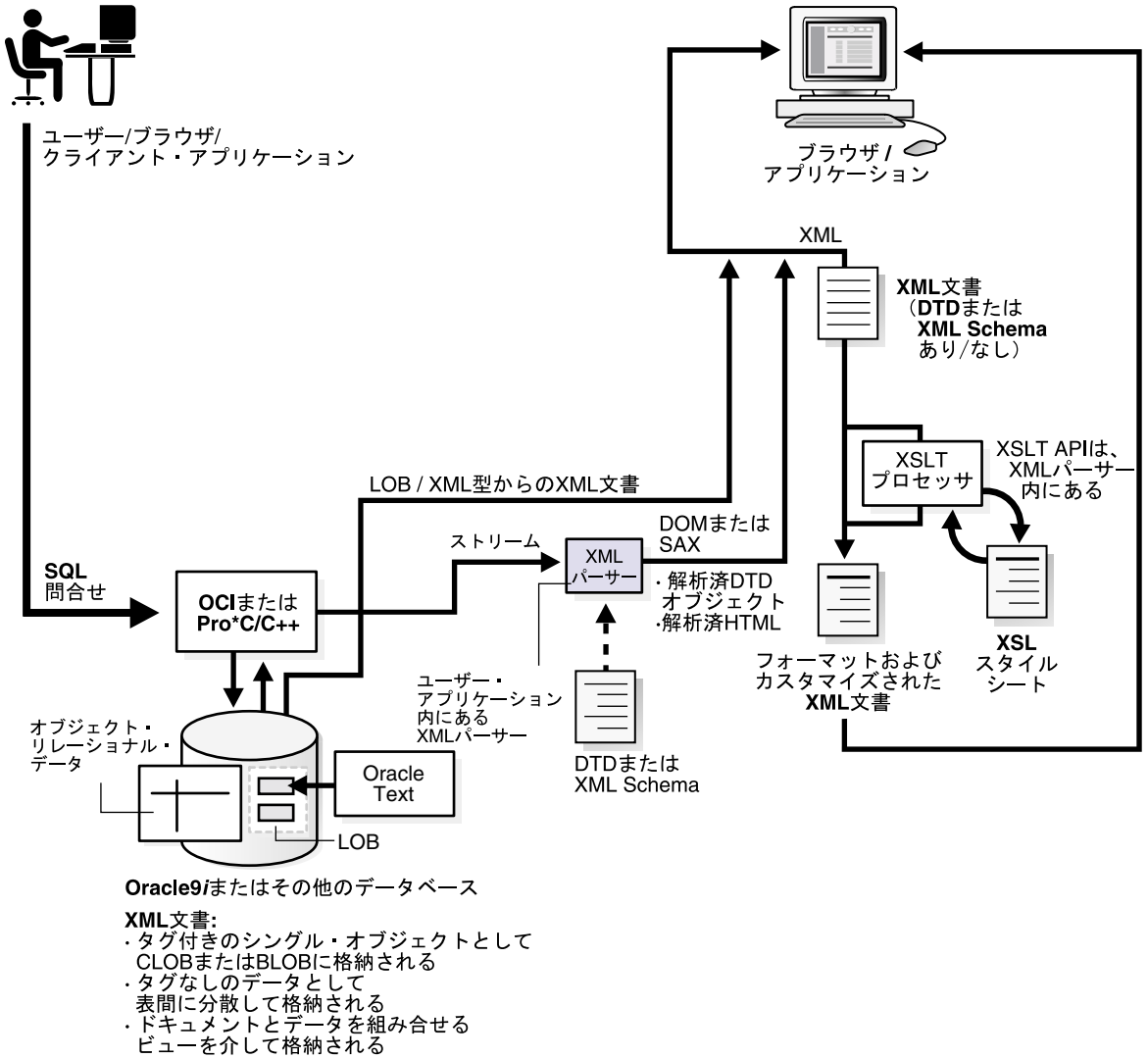
SQL 問合せは、OCI を介して、または Pro*C のプリコンパイラの埋込み文として、データベースに送信されます。

結果 XML データは、次の方法で処理できます。

- XML パーサーを使用した処理
- XML 文書としての CLOB からの処理

オプションで、この XML データを XSLT プロセッサで変換したり、XML 対応のブラウザから直接表示したり、追加処理のためにアプリケーションまたは AQ Broker へ送信することができます。

図 3-8 XDK for C を使用した XML 文書の生成



Oracle の XML コンポーネントを使用した XML 文書の生成 : C++

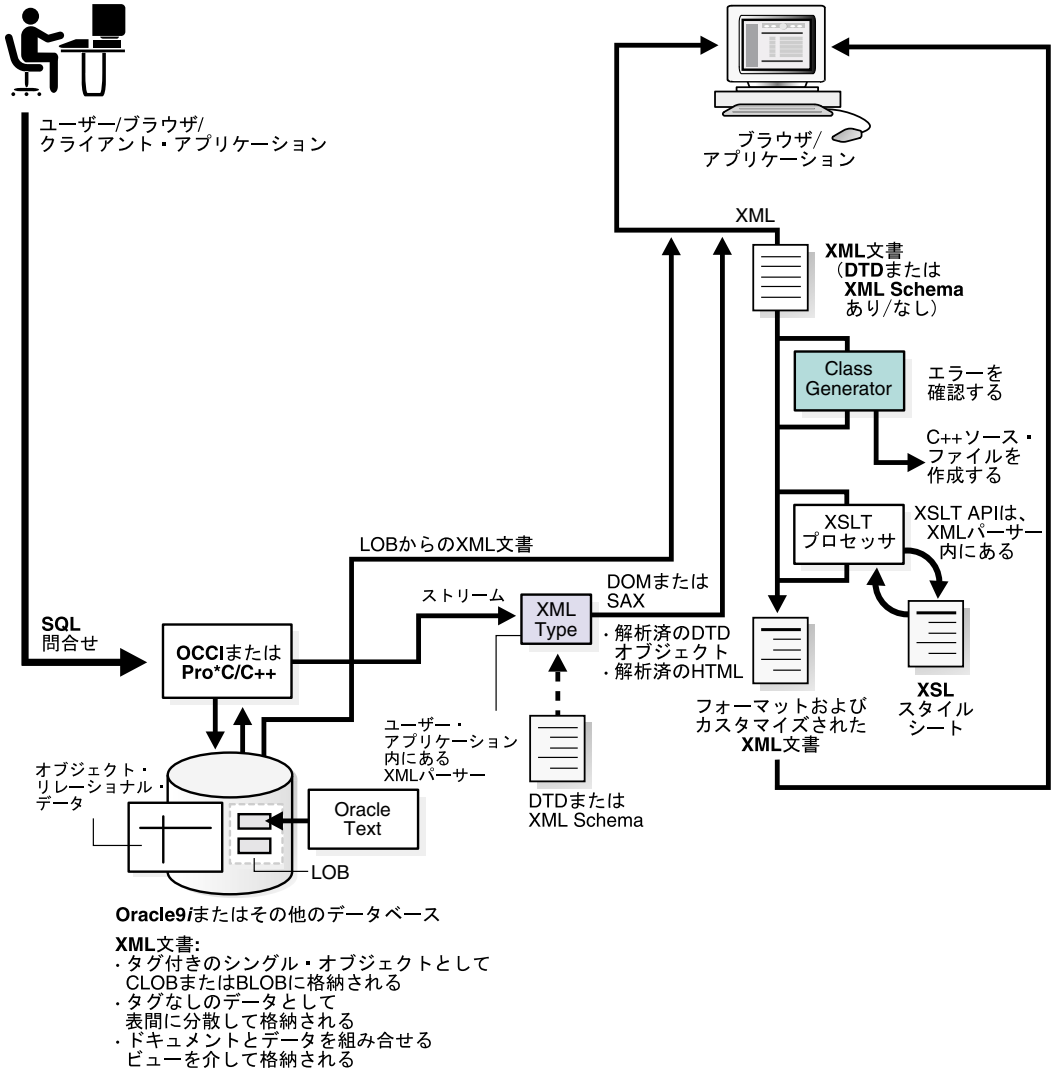
図 3-9 に、XML 文書の生成に使用される Oracle の XML コンポーネントを示します。次に、使用可能な XDK for C++ コンポーネントを示します。

- XSLT を含む XML Parser for C++
- XML Schema Processor for C++
- XML Class Generator for C++

C++ 環境では、ユーザー、クライアントまたはアプリケーションが SQL 問合せを送信した場合、XDK for C++ を使用した次の 2 つの方法でその問合せを処理できます。

- JDBC を使用した直接処理（JDBC が XML パーサーにアクセスします）
- Oracle C++ Call Interface（OCCI）または Pro*C/C++ プリコンパイラを介した処理

図 3-9 XDK for C++ を使用した XML 文書の生成



Oracle の XML コンポーネントを使用した XML 文書の生成 : PL/SQL

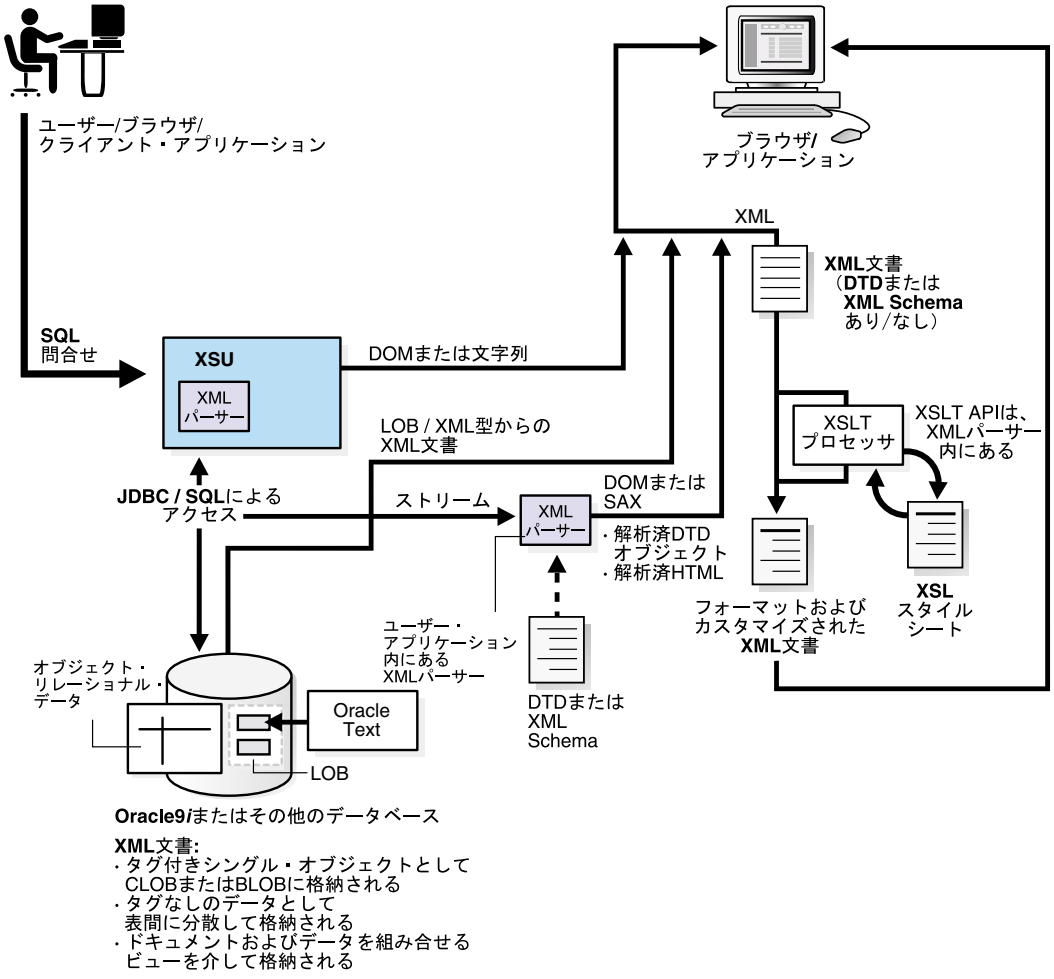
図 3-10 に、XML 文書の生成に使用される XDK for PL/SQL のコンポーネントを示します。次に、使用可能な PL/SQL 用コンポーネントを示します。

- XSLT を含む XML Parser for PL/SQL
- XSU for PL/SQL

PL/SQL 環境では、ユーザー、クライアントまたはアプリケーションが SQL 問合せを送信した場合、Oracle の XML コンポーネントを使用した次の 2 つの方法でその問合せを処理できます。

- JDBC を使用した直接処理（JDBC が XML パーサーにアクセスします）
- XSU を介した処理

図 3-10 XDK for PL/SQL を使用した XML 文書の生成



FAQ: Oracle の XML 対応テクノロジ

この項では、次のカテゴリごとに、Oracle の XML 対応テクノロジに関する一般的な FAQ を示します。

FAQ は、このマニュアルの他の章でも記載しています。

XDK 全般に関する FAQ

インストールする必要がある XML コンポーネント

質問

XML および Oracle を使用して小規模なアプリケーションを開発する計画です。内容は次のとおりです。A 社には中央発注システムがあるとします。A 社には、B、C および D 部門があり、同社は B、C および D から XML 形式の発注書を受け取ります。

現在、A 社はすべての発注書を収集し、それを Oracle データベースに格納する必要があります。また A 社は、そのデータベースから、同社の優先サプライヤに対する別の「提案要求」を XML で作成する必要があります。そのため、データベースに対して挿入または更新問合せを実行する予定です。どのような XML コンポーネントを Oracle にインストールする必要がありますか？

回答

Java を使用した実装を前提とすると、XML パーサーおよび XSU が必要です。Java ベースのフロントエンドを使用して発注書を生成する場合は、XML Class Generator によって、発注書を移入するために必要なクラスが提供されます。また、Web インタフェースの構築には XSQL Servlet が有効です。

XML アプリケーションの構築 : 必要なソフトウェア

質問

現在、Solaris 2.6 上で CGI と PERL と Oracle7 によるアプリケーションを使用していますが、これを XML/XSL と JAVA と Oracle に変換する予定です。SGML、XML、JAVA など、ほぼすべてのテクノロジを習得していますが、これを Oracle で使用する方法がわかりません。どのようなオラクル社製ソフトウェアが必要ですか ? 特に、次のことについて教えてください。

1. Oracle Web Server のかわりに Apache を使用できますか ? 使用できる場合は、その方法を教えてください。
2. Oracle リリース 7.3 で、どこまで対応できますか ?
3. すべての XML をプログラムで作成している場合も、XML パーサーが必要ですか ?
4. Web サーバーと Oracle DB サーバーの間に、XSQL Servlet、パーサー、Java VM、EJB、CORBA、SQLJ、JDBC、または `UTL_HTTP` などの Oracle パッケージを使用する必要はありますか ?

回答

1. Apache を使用できます。Apache Web サーバーは、JDBC などの方法で Oracle とやりとりします。たとえば、XSQL Servlet があります。これは、サーブレット対応のすべての Web サーバー上で実行可能なサーブレットです。このサーブレットは Apache 上で動作し、Oracle データベースに対する JDBC ドライバを介してデータベースに接続します。
2. Oracle リリース 7.3 は、ほとんどの処理に対応可能です。唯一の問題は、Java プログラムをサーバー内で実行できず、すべての XML ツールをサーバー内にロードできないことです。ただし、Oracle7 用の Oracle JDBC ユーティリティをダウンロードしてデータベースに接続し、すべてのプログラムをクライアント側のユーティリティとして実行することはできます。
3. すべての XML をプログラムで作成している場合も、XML パーサーが必要かどうかは、生成した XML の用途によって異なります。XML の生成および送信のみを行う場合、XML パーサーは必要ありません。ただし、XML DOM ツリーを生成する場合は、XML パーサーが必要です。また、XML 文書を受信し、それを解析して、いずれかの場所に格納する場合も、XML パーサーが必要です。これについては、第 7 章「XSU」を参照してください。
4. Web サーバーと Oracle DB サーバーの間には、回答 1 で説明したとおり、OCI または JDBC を介して Oracle とやりとりするサーブレット（または CGI）が必要です。

DTD からデータベース・スキーマへの移行

質問

DTD からデータベース・スキーマに移行するためのツール製品はありますか？

回答

XML Schema 以外に、データ型を指定する方法がないため、現在オラクル社には DTD からデータベース・スキーマに移行するためのツール製品はありません。OTN からダウンロードできる XSU および他の XML コンポーネントを使用すると、データベース・スキーマから DTD を生成できます。この DTD は、XML Class Generator への入力として使用できます。データベースが関連するため、この方法をお薦めします。詳細は、`http://otn.oracle.com/tech/xml` にある XML Discussion Forum などの OTN のリソースを参照してください。

XML へのスキーマのマップ

質問

当社のプロジェクトでは、顧客のために、マスターとディテール・データを XML に変換する必要があります。

1. 表の設計および XML の生成（フラットな表、オブジェクトまたはコレクション）に最適な方法を教えてください。
2. Pro*C で XSU を使用できますか？
3. データベースから生成する XML 文書にサイズの制限がありますか？Pro*C を使用して XSU をコールできますか？

回答

1. これは、アプリケーションの用件に依存します。一般的な方法は、オブジェクト・ビューを使用し、スキーマによって、データベース・データを含むタグ構造を要素の内容として定義することです。
2. 確認されている制限は、オブジェクト・ビューおよび基礎となる表の構造による制限のみです。

データを他の形式から XML に変換するための XDK ユーティリティ

質問

XDK には、データを特定の形式から XML に変換するためのユーティリティが含まれていますか？XSLT が XML から XML、HTML またはその他のテキストベース・フォーマットに変換することは知っています。逆の変換の場合はどうですか？

回答

HTML の場合、Tidy や JTidy などのユーティリティを使用して、XSLT によって変換可能である整形式の HTML に変換できます。ランダム・テキスト・フォーマットの場合、`http://www.unidex.com/xflat.htm` にある XFlat を試すことができます。

Rational Rose で生成された XML ファイルからのデータベース・スキーマの生成

質問

Oracle で、`CREATE TABLE` を含むスクリプトを使用して、Rational Rose 設計ツールで生成された XML ファイルからデータベース・スキーマを生成することはできますか？

回答

オラクル社のプロジェクトでは、すべてのパーサー・ファイル / ジェネレータ・ファイル（Petal-File、XML など）を開発しています。すべてのコンポーネントは、再利用できるように設計されていますが、大規模なフレームワークの中で開発されています。ユーザーは、いくつかのガイドライン（UML でのモデル化など）に従う必要があります。また、オラクル社製品のメリットを活かすには、基本となるクラスを使用する必要があります。

Oracle は、オブジェクト型を生成し、永続性レイヤーでの継承などの完全なオブジェクト指向の機能を提供するのみです。この機能を必要としない場合、Rational Rose（Petal-File）パーサー、および様々なジェネレータの基礎になる Oracle 独自のパッケージの使用を検討してください。

XML 文書を作成および編集するためのツール製品

質問

DTD または XML Schema の検証を使用して、XML 文書を作成（DTD または XML Schema 定義の DOM に基づく）および編集するためのツール製品はありますか？

回答

JDeveloper9*i* に、XML Schema や XSLT スタイルシートなどの XML Schema ベースのドキュメントに使用できる、統合された XML Schema 駆動のコード・エディタが搭載されています。また、XML Schema で定義された正しい要素および属性を簡単に入力できる Tag Insight 機能もあります。

> **参照：** 第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」を参照してください。

XML 文書を PDF フォーマットに変換する方法

質問

リリース 8.1.6 に格納されている XML 文書を PDF にフォーマットするように依頼されました。開発環境として JDeveloper リリース 3.1.1.2 を使用しており、クライアントは可能であれば Windows NT の OAS リリース 4.0.8.2 環境を変更しないことを望んでいます。何か提案または推奨のリソースはありますか？

回答

Oracle XSQL Pages バージョン 1.0.2 は、Apache FOP 0.14.0 と統合して、XML/SQL 入力からの PDF 出力のレンダリングをサポートします。

Formatting Object（FOP）を使用すると、XML を PDF にフォーマットできます。詳細は、`http://xml.apache.org/fop/` および `http://www.xml.com/pub/rg/75` を参照してください。

大規模な XML 文書をデータベースにロードする方法

質問

大規模（27MB）でデータ中心の XML 文書があります。この文書が XSU を使用してリレーショナル表に分割されているとき、この文書をデータベースにロードできませんでした。これは、XSLT プロセッサ実行中に、（メモリー・リークのため）DOM パーサーに障害が発生したためです。この問題の解決策はありますか？SAX パーサーを使用する必要がありますか？XSLT プロセッサおよび SAX パーサーの使用方法を教えてください。

回答

1. XML 文書のロードが 1 回のみの場合、または取得する XML 文書のタグが常に同一である場合、SQL*Loader（ダイレクト・パス）の使用を検討してください。これには、ローダー制御ファイルの構成のみが必要です（『Oracle9*i* データベース・ユーティリティ』の第 3 章などを参照してください）。`enclosed by` オプションを使用して、フィールドを記述できます。たとえば、ファイルのリストで次のように入力します。

   ```
   (empno    number(10)    enclosed by "<empno>" and "</empno>",...)
   ```

   データの解析はどのツールを使用しても処理時間に違いがありませんが、実際のデータベースへのロードは、SQL*Loader を使用した場合が最も高速です（ダイレクト・パスでは、中間のレイヤーをバイパスして、データが直接データ・ブロックに書き込まれるため）。

2. サブ文書の多数の繰返しによって、文書のサイズが 27MB になる場合、『Building Oracle XML Applications』（Steve Muench 著、O'Reilly 出版）の第 14 章のサンプル・コードを使用すると、任意のサイズの XML を、任意の数の表にロードできます。第 14 章「Advanced XML Loading Techniques」の例には、大規模 XML 文書をデータベースにロードするための XML ローダー・ユーティリティを作成する方法が示されています。

質問

SQL*Loader は、ネストを処理できますか？次のようなコードがあるとします。

```
...
     <something>
        <price>10.00</price>
     </something>
...
   ...
      ...
         <somethingelse>
           <price>55.00</price>
         </somethingelse>
```

2 つの <price> 要素を一意に識別する方法がありますか？

回答

厳密には、SQL*Loader はネストを処理できません。制御ファイルのフィールドの記述はネストされます。これは、オブジェクト・リレーショナル列のサポートの一部です。この記述がマップするデータ・レコードはフラットですが、SQL*Loader のすべてのデータ・フィールド記述機能を使用すると、多くのタスクを実行できます。たとえば、次のような記述があります。

sample.xml

```
<resultset>
    <emp>
        <first>...</first>
        <last>...</last>
        <middle>....</middle>
    <emp>
    <friend>
        <first>...</first>
        <last>...</last>
        <middle>....</middle>
    </friend>
</resultset>
```

sample.ctl -- SQL*Loader の制御ファイルのフィールド定義部分

```
field list ....
(
 emp  COLUMN OBJECT ....
   (
        first      char(30)   enclosed by "<first>" and "</first>",
        last      char(30)   enclosed by "<last>" and "</last>",
        middle     char(30)   enclosed by "<middle>" and </middle>"
   )
    friend COLUMN OBJECT ....
  (
        first      char(30)   enclosed by "<first>" and "</first>",
        last      char(30)   enclosed by "<last>" and "</last>",
        middle     char(30)   enclosed by "<middle>" and </middle>"
  )
```

COLUMN OBJECT のフィールド名は、データベースの ADT 列と一致する必要があることに注意してください。また、カスタムのレコード終了記号を使用する必要があります。使用しない場合、デフォルトで、改行が終了位置になります（改行ごとに、完全なデータベース・レコード用のデータがあるとみなされます）。

XML がより複雑で、選択したフィールドのみを抽出する場合、FILLER フィールドを使用して、スキャン・カーソルの位置を再設定できます。スキャン・カーソルは、スキャンが中断された位置から（最初のフィールドの場合はレコードの先頭から）、レコードの最後までスキャンします。

SQL*Loader には、強力なテキスト・パーサー機能があるため、多くの高度な処理を実行できます。タグに一貫性がある大規模な XML 文書をロードする場合は、SQL*Loader の使用を検討する必要があります。

> **参照：** XML のロードのガイドラインについては、2-13 ページの「データベースへの XML のロード」を参照してください。

以前のリリースの Oracle における移植性および XML サポート

異なるベンダーの XML パーサーの使用

質問

現在、SAX を検討しています。オラクル社および IBM 社の XML パーサーは、両方とも W3C 勧告の DOM および SAX を使用していると理解しています。

- 異なるベンダー（Oracle、IBM など）の XML パーサーにはどのような違いがありますか？
- 現在、Oracle の XML パーサーを使用していますが、他のベンダーのパーサーに切り替える場合、コードを変更する必要がありますか？

回答

実装用の SAX インタフェースまたは DOM インタフェースを変更しない場合、コードを変更する必要はありません。この点が、標準のインタフェースのメリットです。

Oracle 8.0.x に対する XML のサポート

質問

当社は、将来、XML ベースのインタフェースで実行するためのシステムの一部を設計しています。当社は、ウォール街の大手機関です。すべての現行システムは Oracle8 リリース 8.0.6 で実行しており、その需要の高さから、XML 概念の一部を既存システム上に実装する計画です。現在または将来、データベース内で XML ベースのコードをサポートする計画はありますか？また、これに対処するために使用できるアダプタまたはカートリッジを教えてください。

回答

XML パーサー、XSLT プロセッサ、XSQL Servlet や XSU などのユーティリティを含む、Oracle XDK のすべてのコンポーネントは、Oracle8 リリース 8.0.6 では、データベース外で問題なく機能します。ただし、データベース内で XML コンポーネントを実行したり、Oracle Text（*inter*Media Text）を使用して XML を検索することはできません。これらは両方とも Oracle8*i* 以上の機能です。

Oracle 7.3.4: XML を使用したサプライヤへのデータ転送

質問

当社では Oracle リリース 7.3.4 を使用しています。当グループでは、当社とサプライヤ間のデータ転送の一部に XML を使用することを検討中です。Web サイトを見るかぎり、これを行うために、当グループは Oracle8*i* に移行する必要があると思われます。Oracle7 を使用して XML を使用する方法はありますか？

回答

リリース 7.3.4 用の適切な JDBC 1.1 ドライバをお持ちの場合、XSU を使用して、XML 形式でデータを取り出すことができます。

Oracle7 JDBC OCI ドライバおよび JDBC Thin ドライバについては、`http://otn.oracle.com/tech/java/sqlj_jdbc/` を参照してください。

Oracle8*i* より前のバージョンにおける Oracle の XML ツール製品の使用

質問

1. Oracle8*i* より前のバージョンを使用している場合、Oracle の XML ツール製品を使用して XML ベースのアプリケーションを提供できますか？提供できる場合、ライセンスに関する条件について教えてください。
2. Oracle の XML テクノロジは、磁気テープ・ファイルの作成に適していますか？このファイルは、「`abcdefg.....`」のように特定のフォーマットの文字列です。このようなファイルを作成するスタイルシートを作成できますか？

回答

1. XSU および XSQL Pages フレームワークを含む XDK for Java、XDK for C および XDK for C++ は、データベース外で動作します。最新のライセンスについては、OTN を参照してください。
2. 作成できます。`<xsl:output method="text"/>` を使用するのみで、プレーン・テキストを出力できます。

XML をサポートするブラウザ

XML をサポートするブラウザ

質問

XML をサポートするブラウザのリストはありますか ?

回答

次のブラウザが、XML 表示をサポートしています。

- Opera （バージョン 4.0 以上の XML）
- Citec Doczilla （XML および SGML ブラウザ）
- Indelv （XSL を使用してのみ XML 文書を表示）
- Mozilla Gecko （XML、CSS1 および DOM1 をサポート）
- HP ChaiFarer （XML および CSS1 をサポートする埋込み環境）
- ICESoft （XML、DOM1、CSS1 および MathML をサポートする埋込みブラウザ）
- Microsoft IE5 以上 （完全な XML パーサーを含む）
- Netscape 5.x 以上

標準

XML が EDI より優れる点

質問

サプライヤおよび顧客との通信に必要なため、EDI を実装することを検討しています。ただし、小規模な企業の場合、XML の方がより低コストな方法であると理解しています。XML が EDI より優れる点について教えてください。

回答

これについては、次のことが考えられます。

- EDI は、難解なテクノロジです。開発者が簡単に読んだり、理解することができない形式でのマシン対マシン通信を許可します。
- EDI メッセージのデバッグは非常に困難です。XML 文書は簡単に読んだり、理解することができます。
- EDI は、柔軟性に欠けます。新しい取引先を既存システムの一部として追加することが非常に困難であり、新しい取引先ごとに個別のマッピングが必要です。XML は、柔軟性に富み、必要に応じて新しいタグを追加したり、XML 文書を別の XML 文書に変換して、2 つの異なる形式の発注書番号などをマップすることができます。
- EDI は、高コストです。開発者の養成にコストがかかります。また、非常に高性能なサーバーが必要であり、専用ネットワークに対する要件も高いため、配布にもコストがかかります（EDI は、高コストな VAN で実行します）。XML は、既存のインターネット接続上で、低コストな Web サーバーで動作します。

また、「今後、XML は EDI に取ってかわるか ?」という質問が浮かびますが、その可能性は低いといえます。EDI と XML は、少なくとも当面の間、共存する見込みです。現在、EDI に投資している大企業は、方向転換するのではなく、既存の EDI ベースの実装を拡張する方法として XML を使用する可能性が高いといえます。これによって、XML と EDI の統合という新しい問題が発生します。

小規模な組織および EDI の柔軟性に欠けるアプリケーションの場合、XML は非常に有効な方法です。

Oracle がサポートする B2B 標準および開発ツール

質問

Oracle は、どの B2B 用 XML の標準（ebXML、cxml、BizTalk など）をサポートしますか？オラクル社が提供する B2B による取引用のツールにはどのようなものがありますか？

回答

オラクル社は、次の B2B の標準化機関に参加しています。

- OBI（Open Buying on the Internet）
- ebXML（Electronic Business XML）
- RosettaNet（IT 業界におけるサプライ・チェーンのための E-Commerce）
- OFX（Open Financial Exchange for Electronic Bill Presentment and Payment）

オラクル社は、顧客のニーズに応じて、次の B2B による取引用オプションを提供します。

- 電子マーケットプレイスを実装するための独自のソリューションを提供する Oracle Exchange
- 組織内実装用の OIS（および主に Message Broker）
- データ・レベルでの情報交換のための Oracle Gateway
- Oracle E-Business Suite から XML ベースのメッセージを転送するための Oracle XML Gateway

Oracle インターネット・プラットフォームは、B2B による取引のための統合されたソリッドなプラットフォームを提供します。

XML に関するオラクル社の方針

質問

XML に関するオラクル社の方針を教えてください。

回答

XML に関するオラクル社の方針は、オラクル社がこれまでに蓄積したテクノロジを最大限に活用する方法で XML を使用することです。現在では、Oracle の XML コンポーネントを Oracle8*i* 以上のデータベースおよび AQ と組み合せて、競合解消、トランザクション確認などを実現できます。オラクル社では、将来のリリースを、これらの機能に加えて、分散 2 フェーズ・コミット・トランザクションなどの点でもよりシームレスなものにするための取組みを行っています。

XML データは、オブジェクト・リレーショナル表またはオブジェクト・リレーショナル・ビューに格納されるか、または CLOB として格納されます。XML トランザクションは、これらのデータ型の 1 つを伴うトランザクションで、ロールバック・セグメント、ロック、ロギングを含む、標準の Oracle メカニズムを使用して処理されます。

Oracle では、AQ を使用した XML ペイロード送信をサポートしています。これには、SQL からの XML の問合せを可能にする必要があります。

オラクル社は、特定の XML Schema を開発および登録するために、W3C の XML ワーキング・グループ、Java Extensions for XML、オープン・アプリケーション・グループ、`XML.org` など、すべての XML 標準化機関に積極的に参加しています。

XML Query

オラクル社は、XML Query の W3C のワーキング・グループに参加しています。オラクル社は、XQL 提案にあるような、XML データの問合せを可能にする言語を実装する計画を検討しています。XSLT は静的な XML 変換機能を提供しますが、問合せ言語は、SQL がリレーショナル・データの柔軟性を高めるように、データ問合せの柔軟性を高めます。

オラクル社は、XML/XSL に関連する XML Schema、XML Query、XSL、XLink/XPointer、XML Infoset、DOM および XML Core の W3C ワーキング・グループに積極的に参加しています。

受注や出荷などに使用できる標準の DTD

質問

Oracle8*i* および XDK を実装済です。受注、出荷および通知のための基本的な標準の DTD は、どこで取得できますか？

回答

`http://www.xml.org` を参照してください。

XML、CLOB および BLOB

BLOB の XML メッセージのサポート

質問

XML メッセージに BLOB を含める機能はサポートされていますか？または、MIME ラッパーを使用して、バイナリ・オブジェクトを UUENCODE などの適切なテキスト・フォーマットにエンコーディングし、アプリケーション・レベルで行う必要はありますか？

回答

XML の場合、すべての文字を解析する必要があるため、RAW バイナリ・データを XML 文書に挿入することはできません。ただし、データを UUENCODE にエンコーディングし、それを CDATA セクションに挿入することはできます。このエンコーディング方法に対する制限は、正当な文字のみが CDATA セクションに生成される必要があることです。

ファイルの最大サイズ

CLOB 格納時の XML ファイルの最大サイズ

質問

XML ファイルを CLOB として Oracle データベースに格納する場合の、ファイルの最大サイズを教えてください。

回答

ファイルの最大サイズは 2GB です。LOB および CLOB の詳細は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

XML ファイルのサイズ制限

質問

XML ファイルのサイズに制限はありますか？

回答

ありません。

XML 文書の最大サイズ

質問

1. CLOB を使用しない場合、表全体に PL/SQL（または SQL）用のデータを指定するための XML 文書の最大サイズがありますか？
2. Oracle から XML 文書に生成された XML 文書の最大サイズは？

回答

1. 最大サイズは、1 つのオブジェクト・ビューに挿入できるサイズです。
2. 最大サイズは、1 つのオブジェクト・ビューから取り出すことができるサイズです。

表への XML データの挿入

XML を使用して表にデータを挿入するために必要なソフトウェア

質問

表示するデータを選択し、XML を介して表にデータを挿入するためには、どのようなソフトウェアが必要ですか？現在、Solaris 上で Oracle8*i* を使用しています。

回答

次のソフトウェアが必要です。

- XSU
- XML Parser for Java
- JDBC ドライバ
- JDK

最初の 3 つはオラクル社の製品です。4 つ目は、Sun 社の製品です。これをブラウザから行う場合は、次のものも必要です。

- Java 準拠の Web サーバー
- XSQL Servlet

データベース内の XML: パフォーマンス

XML および Oracle のパフォーマンスに関する情報の参照先

質問

XML および Oracle のパフォーマンスに関するホワイト・ペーパーはありますか？

回答

現在、XML 製品用のパフォーマンス標準またはベンチマークがないため、正式なパフォーマンスの分析情報はありません。

より高速な XML 文書のレコード取出し

質問

何百万ものレコードを含むデータベースがあります。4、5 個のパラメータを使用して問合せを行い、一致するレコードを取り出しました。同じレコードの取出しを高速化するため、データベースに索引を追加しました。ただし、戻されるレコード数が多く、「前へ」と「次へ」のリンクを設定して、1 回に 10 レコードを表示させることにし、count(*) で一致するレコードの数を取得する必要がありました。

レコードが多すぎるため、count(*) では有効に索引が機能せず、取り出されたリストがブラウザ・ウィンドウに表示されるまでに約 20 ～ 30 秒かかりました。count(*) を削除すると取出しは非常に高速化されますが、count(*) に対する「前へ」と「次へ」のリンクは消去されます。

回答

より高速な XML 文書の取出し方法に関するご質問ですが、DOM のかわりに SAX を使用してみてください。

索引列の COUNT(*) を選択するようにしてください（索引の選択性が高いほど有効です）。この方法では、オプティマイザはフル・テーブル・スキャンのかわりに、数回の索引ブロック I/O のみで COUNT 問合せを実行できます。

異なる言語での XML の使用

質問

当社のアプリケーションは、異なる言語体系の外部のエンティティと通信する必要があります。別の言語（たとえば中国語）の情報を XML に含める必要がある場合、それらの言語を異なる方法で処理する必要がありますか？たとえば、使用されるエンコーディングに注意する必要がありますか？または、パーサーがそのエンコーディングを認識できますか？データベースを処理する場合、問題がありますか？

回答

XML は、本質的に単一ドキュメント内の複数の言語をサポートします。各エンティティは、他のエンティティとは異なるエンコーディングを使用できます。そのため、中国語のエンコーディングでエンコードされた中国語のエンティティを、残りのドキュメントに追加できます。また、使用する言語に関係なく、Unicode にエンコードすることによって、すべての部分を同様に扱うこともできます。最初の例では、XML テキスト宣言内でエンコーディング宣言を行う必要があります。

Oracle XML Parser は、ほとんどの外部エンティティを処理できるように設計されています。また、世界中で広く使用されているほとんどの言語コードを含む、様々な言語コードを認識できます。

データベースは、XML で使用する予定のすべての言語をサポートする必要があります。ZHS16GBK および ZHT16BIG5 などの中国語のキャラクタ・セットは ASCII のスーパーセットであるため、いずれかのキャラクタ・セットを使用して英語および中国語を処理できます。ただし、Unicode を使用して、より多くの言語を使用する必要がある場合もあります。

追加情報

XML に関するその他の FAQ

XML 関連の FAQ については、次のサイトも参照してください。

- http://www.ucc.ie/xml/

XML および XSL 関連の推奨書籍

質問

XML または XSL 関連の推奨書籍があれば教えてください。

回答

- WROX という出版社は多数の有用な書籍を出版しています。『XML Design and Implementation』（Paul Spencer 著）は、XML、XSL および開発について詳しく説明しています。
- 『Building Oracle XML Applications』（Steve Muench 著、O'Reilly 出版）については、http://www.oreilly.com/catalog/orxmlapp/ を参照してください。
- 『XML Bible』は、XML および XSL に関する有効な書籍です。次の Web サイトで、最新の第 14 章を参照できます。

  http://metalab.unc.edu/xml/books/bible/

  これを読むと、XSLT の理解を深めることができます。この章は無償でダウンロードできます。
- 『Oracle XML Handbook』（Oracle XML Product Development Team 著）については、http://www.osborne.com/oracle/ を参照してください。

4

XSL および XSLT の使用

この章の内容は次のとおりです。

XSL の概要

XML 文書には、構造はありますがフォーマットはありません。XSL は、XML 文書にフォーマットを追加します。

XSL は、XML セマンティクスを表示する方法を提供します。XSL を使用すると、XML 要素を HTML などの他のフォーマット言語にマップできます。

W3C の XSL 仕様

W3C は、スタイルシート活動の一部として XSL 仕様を策定しています。XSL には、スタイルの指定以外に、ドキュメントを操作する機能もあります。XSL は、XML のスタイルシート言語です。

1999 年 7 月の W3C の XSL 仕様は、次の 2 つのドキュメントに分割されています。

- XSL の構文およびセマンティクス
- XSL を使用してスタイルシートを適用し、ドキュメントを別のドキュメントに変換する方法

XSL で使用される書式設定オブジェクトは、CSS および文書スタイル意味指定言語（DSSSL）に関する以前の作業に基づいています。XSL は、DSSSL より簡単に使用できるように設計されています。

XSL 提案に定義されている機能によって、次の機能が使用可能になります。

- 祖先と子孫、位置、および一意性に基づいたソース要素のフォーマット
- フォーマット構造体の作成（生成されたテキストおよび図形を含む）
- 再利用可能な書式設定用マクロの定義
- 書込み先に依存しないスタイルシート
- 拡張可能な一連の書式設定オブジェクト

XSL 仕様提案

XSL 仕様では、XSL はスタイルシートを表す言語であると定義されています。任意の構造化 XML 文書またはデータ・ファイルのクラスに対して、設計者は、XSL スタイルシートを使用して、その構造化コンテンツの表示方法を表現します。XSL スタイルシートを使用すると、表示メディア（Web ブラウザや携帯端末のウィンドウ、カタログ、レポート、パンフレットや書籍の一連の物理的なページなど）におけるソース・コンテンツの表示スタイル、レイアウトおよびページ区切りの方法を指定できます。書式設定は、結果ツリーに書式設定セマンティクスを含めることによって実行できます。

書式設定セマンティクスは、書式設定オブジェクトのクラスのカタログで表されます。結果ツリーのノードが、書式設定オブジェクトです。書式設定オブジェクトのクラスは、ページ、段落、表など、出力に対する抽象的な概念を示します。

これらの抽象的概念の表示は、一連の書式設定プロパティ（インデントの制御、ワード間隔や文字間隔、およびウィドウ、オーファン、ハイフネーションの制御など）によって、より細かい制御が可能です。XSL では、書式設定オブジェクトおよび書式設定プロパティのクラスによって、目的の表示を表現するためのボキャブラリが提供されます。

これらを個別のプロセスとして提供する場合、実装は必須ではありません。また、概念的な XSL 処理モデルを使用して処理を行った場合と同じ結果が生成されるように、実装では任意の方法でソース・ドキュメントを処理できます。

XML の名前空間

名前空間とは、一意の識別子または名前です。XML 文書は、異なる DTD または XML Schema を使用して個別に作成できるため、名前空間が必要です。名前空間は、タグが生成された DTD または XML Schema を識別することによって、マークアップ・タグの競合を回避します。また、XML 要素を特定の DTD または XML Schema へリンクします。

`rml:`、`xhtml:`、`xsl:` などの名前空間マーカーを使用する前に、次の段落で説明する名前空間インジケータ `xmlns` を使用して、マーカーを識別する必要があります。

参照： http://w3.org/TR/REC-xml-names を参照してください。

XSL スタイルシートのアーキテクチャ

XSL スタイルシートには、次の構文を含める必要があります。

- スタイルシートを指定する開始タグ（`<xsl:stylesheet2>` など）。
- 名前空間インジケータ（XSL の名前空間インジケータ `xmlns:xsl="http//www.w3.org/TR/WD-xsl"`、書式設定オブジェクトの名前空間インジケータ `xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"` など）。
- フォントの種類と太さ、色、および改ページなどのテンプレート規則。テンプレートには、要素および要素値の制御指示が含まれます。
- スタイルシートの定義の終了を示すタグ（`</xsl:stylesheet2>`）。

XSLT

XSLT は、XSL の 1 つ目の部分として使用されるように設計されています。XSL には、XSLT に加えて、書式設定を指定するための XML ボキャブラリが含まれます。XSL は、XSLT を使用して XML 文書のスタイルを指定し、特定の XML 文書を、書式設定ボキャブラリを使用する別の XML 文書に変換する方法を記述します。

XSL の 2 つ目の部分では、XSL 書式設定オブジェクト、その属性、および書式設定オブジェクトを組み合せる方法を記述します。

参照： 第 20 章「XML Parser for Java の使用」を参照してください。

XSLT 1.1 仕様

W3C の XSL ワーキング・グループは、XSLT 1.1 仕様の要件を記述した仕様書を発表しました。XSLT 1.1 仕様の 1 つ目の目的は、スタイルシートの移植性の向上です。新しい草案については、`http://www.w3.org/TR/xslt11req` を参照してください。

ユーザーの要求のサポートに加えて、XSLT プロセッサは、XSLT 1.0 の拡張メカニズムを利用して、追加の組込み変換機能を提供しています。有効な組込み拡張機能の登場によって、ユーザーはそれらを取り入れ、信頼し始めています。

ただし、これらの拡張機能は、移植性を考慮していません。XSLT 1.0 は、拡張機能の実装に関する詳細または手引きを提供していないため、現在、ユーザー定義の拡張機能または組込み拡張機能は、必然的に単一の XSLT プロセッサでしか使用できません。

目的 1: スタイルシートの移植性の向上

XSLT 1.1 仕様の 1 つ目の目的は、スタイルシートの移植性を向上することです。この目的は、拡張機能の実装メカニズムを標準化し、XSLT の主要な仕様に 2 つの組込み拡張機能を含めることによって達成されます。次に示すこれらの機能は、XSLT プロセッサの既存ベンダーの多くによって、ユーザーの要求に応じて追加されています。

- 変換された複数の出力ドキュメントのサポート
- 追加処理のための、結果ツリーのフラグメントからノード・セットへの変換のサポート

  複数ベンダーの実装で提供されているこれらの拡張機能に関連する側面を標準化することによって、複数の XSLT プロセッサ間で動作するスタイルシートを作成する機能が飛躍的に向上します。

目的 2: 新しい XML 仕様のサポート

XSLT 1.1 仕様の 2 つ目の目的は、XML ベースの新しい仕様をサポートすることです。

XSLT 1.1 仕様提案には、これらの目的を実現するための要件が示されています。ワーキング・グループは、XSLT 1.1 の適用範囲を複数の XSLT 1.0 プロセッサに実装されている機能の標準化に限定することを決定し、最初に拡張機能の実装の標準化に取り組んでいます。

XSLT 2.0 では、拡張要素の標準化、および新しい XML Schema のデータ型を認識する機能のサポートが予定されています。

XPath

XSL に関連する別の仕様が、XPath バージョン 1.0 として公開されています。XPath は、XML 文書の一部を指定するための言語で、XSL によって変換される文書の一部を正確に指定する場合に必須です。たとえば、XPath を使用すると、章の要素に属するすべての段落を選択したり、特記事項の要素を選択することができます。XPath は、XSLT と XPointer の両方で使用できるように設計されています。XPath は、XSLT と XPointer の間で共有される機能に対して、共通の構文およびセマンティクスを提供するために設計されています。

CSS と XSL の比較

W3C は、相互運用可能な書式設定モデルの実装の実現に取り組んでいます。

CSS

CSS は、HTML ドキュメントのスタイルを指定するために使用します。CSS は、W3C のスタイルに関するワーキング・グループにより策定されています。CSS2 は、作成者またはユーザーが HTML ドキュメントや XML アプリケーションなどの構造化ドキュメントにスタイル（たとえば、フォント、間隔、音声キュー）を指定できるスタイルシート言語です。

CSS2 によって、ドキュメントの表示スタイルがドキュメントのコンテンツから分離されるため、Web オーサリングおよび Web サイトのメンテナンスが簡単になります。

XSL

一方、XSL はドキュメントを変換できます。たとえば、XSL を使用して、XML データを Web サーバー上で HTML/CSS ドキュメントへ変換できます。このように、2 つの言語は相互に補足し合っており、ともに使用できます。どちらの言語を使用しても、XML 文書のスタイルを指定できます。CSS および XSL は、基礎となる同一の書式設定モデルを使用するため、設計者は、両方の言語で同じ書式設定機能を使用できます。

XSL が画面上のドキュメントのレンダリングに使用するモデルは、ISO 標準の複雑なスタイル言語である DSSSL への長年の取組みに基づいて構築されています。XSL は、主に複雑なドキュメント作成プロジェクトを目的としており、目次、索引およびレポートの自動生成、およびその他のより複雑な出版タスクにおいても、様々な用途があります。

XSL に関する参照情報

XSL の使用例は、このマニュアル全体に記載されています。詳細は、『Oracle9*i* ケース・スタディ -XML アプリケーション』の次の章を参照してください。

- 「XML を使用したコンテンツのカスタマイズ : Dynamic News アプリケーション」
- 「OracleAS Wireless Edition と XML」
- 「XML および XSQL を使用した表示のカスタマイズ : Flight Finder」

参照 : 次の Web サイトを参照してください。

- http://www.mulberrytech.com/xsl/xsl-list/
- http://www.builder.com/Authoring/XmlSpot/?tag=st.cn.sr1.ssr.bl_xml

FAQ: XSL および XSLT

XSL で IF 文を記述してタグ内の値をテストする方法

質問

要素ではなく、要素の値を比較する構文を教えてください。私が読んだドキュメントでは、タグのテストについての説明はありますが、タグ内の値のテストについては説明されていません。XSL ドキュメントの一部を次に示します。

```
<xsl:template match="EmployeeList">
    <xsl:for-each select="employee">
         <xsl:value-of select="name"/>
         <xsl:value-of select="sal"/>
    </xsl:for-each>
<xsl:template>
```

IF 文を記述して給料が 5000 を超える従業員の情報を赤色で表示させる必要があります。IF 文に sal の値を挿入する方法を教えてください。

回答

次の IF 文を参照してください。

```
  <xsl:if expr="this.nodeTypedValue == 'INIZIATIVE'">
    .........
  </xsl:if>
```

XSL ドキュメントで特定の属性を選択する方法

質問

XML 文書を XSL スタイルシートとマージしていますが、XSL ドキュメントで次のような構文を使用すると、子の属性が戻されません。

```
<xsl:value-of select="Foo/Bar"/>
```

戻されない原因がわかりません。他の XML パーサーでは問題なく動作しているようです。

回答

XPath 式 Foo/Bar は、<Foo> 要素に含まれている <Bar> 要素の値のみを選択するように設計されています。この式は、<Bar> 要素のネストされたコンテンツにあるすべてのテキスト・ノードの連結を戻しますが、属性内のテキスト値を戻すようには設計されていません。

1 つの属性を選択するには、次の構文を使用する必要があります。

```
Foo/Bar/@SomeAttr
```

<Bar> のすべての属性を選択するには、次の構文を使用する必要があります。

```
Foo/Bar/@*
```

XML から HTML への変換時に表示される「Unexpected EOF」

質問

次の XML および XSLT を使用して、単純な XML 文書を HTML 形式へレンダリングしようとしています。XML Parser for Java に付属の XSLSample.java を使用すると、変換は失敗し、「Unexpected EOF」というメッセージが表示されます。XSLT から <td></td>（{ELEMENT} 型の XPath 式が含まれる）を削除すると、変換は問題なく実行できます。

XML は次のとおりです。

```
<ROWSET>
   <ROW>
         <ELEM0>Al</ELEM0>
         <ELEM1>Gore</ELEM1>
         <ELEM2></ELEM2>
         <ELEM3></ELEM3>
         <ELEM4></ELEM4>
   </ROW>
.....
```

XSLT は次のとおりです。

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
 <head>
   <title>Value Upload</title>
 </head>
   <body bgcolor="#FFFFFF">
     <form method="post" action="">
     <xsl:for-each select="ROWSET">
```

```
      <table border="1" cellspacing="0" cellpadding="0">
         <xsl:for-each select="ROW">
           <tr>
         <td><input type="text" name="elem0" value="{ELEM0}" size="10"
               maxlength="20"></td>
         <td><input type="text" name="elem1" value="{ELEM1}" size="10"
               maxlength="20"></td>
         ...
         </xsl:for-each>
    </form>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

回答

次に示すとおり、入力要素にスラッシュ（/）を挿入する必要があります。

```
<td> <input xxxx /> </td>
```

空白 : 2 倍になる結果の値

質問

次のコードに構文エラーがありますか？

djia.xml は次のとおりです。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet type="text/xsl" href="djia.xsl"?>
<djia>
  <company>ALCOA</company>
  <company>ExxonMobil</company>
  <company>McDonalds</company>
  <company>American Express</company>
</djia>
```

djia.xsl は次のとおりです。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"version="1.0">
<xsl:output method="xml" encoding="Shift_JIS"/>
<xsl:template match="/djia">
<page>
     <xsl:apply-templates/>
</page>
</xsl:template>
<xsl:template match="company">
     <xsl:value-of select="current()"/>
     <xsl:value-of select="position()"/>:
     <xsl:if test="current()=../company[last()]"> last one!
     </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

結果は次のとおりです。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<page>ALCOA2: ExxonMobil4: McDonalds6: American Express8: last one!</page>
```

結果の値が 2 倍になる原因を教えてください。

回答

空白が原因です。/djia のテンプレートに <xsl:apply-templates/> が指定されている場合、<djia> のすべての子ノードが選択されます。djia.xml は適切にインデントされているため、<djia> の子ノードは次のようになります。

1. 改行コード（CR）と、次の要素がインデントされているように表示するための空白を含むテキスト・ノード
2. <company> （値は ALCOA）
3. 改行コード（CR）と、次の要素がインデントされているように表示するための空白を含むテキスト・ノード
4. <company> （値は ExxonMobil）
5. 改行コード（CR）と、次の要素がインデントされているように表示するための空白を含むテキスト・ノード
6. <company> （値は McDonalds）

7. 改行コード（CR）と、次の要素がインデントされているように表示するための空白を含むテキスト・ノード
8. <company>　（値は American Express）
9. 改行コード（CR）と、次の行に </djia> を入れるための空白を含むテキスト・ノード

XSLT プロセッサは、この現行のノード・リストを処理するため、position() 関数は、現行のノード・リスト内の位置（<company> 要素に対して 2、4、6 および 8）になります。

最上位層に次のタグを追加することによって、この問題を解決できます。

```
<xsl:strip-space elements="*"/>
```

ただし、XDK for Java に不具合があるため、現在は正しく動作しません。したがって、解決策として次のタグを使用してください。

```
<xsl:apply-templates select="company"/>
```

この要素は、次のタグのかわりに使用します。

```
<xsl:apply-templates/>
```

XSL で NULL インジケータを指定する方法

質問

対応するソース XML が <mytag /> または <mytag NULL="YES"/> の場合に、XSLT で <mytag null="yes"/> と出力させる方法を教えてください。

回答

次の構文を使用してください。

```
<xsl:template match="mytag">
  <!-- If there are no child nodes -->
  <xsl:if test="not(node())">
    <mytag null="yes"/>
  </xsl:if>
</xsl:template>
```

XSLT でタグ名を変換する方法

質問

XSLT を使用して XML コードを変換する必要があります。元の XML コードは次のとおりです。

```
<REF_STATUS>
 ...
</REF_STATUS>
```

次の XML コードに変換します。

```
<REF index="STATUS">
...
</REF>
```

REF_VATCODE および REF_USFLG についても同様に変換します。最初に記述した次の構文は、問題なく動作しました。

```
<!-- fix REF_STATUS nodes -->
<xsl:template priority="1" match="REF_STATUS">
  <xsl:element name="REF">
   <xsl:attribute name="index">STATUS</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<!-- fix REF_USFLG nodes -->
<xsl:template priority="1" match="REF_USFLG">
  <xsl:element name="REF">
   <xsl:attribute name="index">USFLG</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<!-- fix REF_VATCODE nodes -->
<xsl:template priority="1" match="REF_VATCODE">
  <xsl:element name="REF">
   <xsl:attribute name="index">VATCODE</xsl:attribute>
    <xsl:apply-templates/>
 </xsl:element>
</xsl:template>
```

REF_ で始まるタグ名が 3 つあります。タグ名は REF に変換され、元のタグ名の REF_ を除いた部分と等しい index 属性を持ちます。これらすべてに一致する 1 つの規則を作成して、適切な変換を行う必要があります。次の構文を試してみました。

```
<xsl:template priority="1" match="starts-with(local-name(),'REF_')">
  <xsl:element name="REF">
   <xsl:attribute name="index">
    <xsl:value-of select="substring-after(local-name(),'REF_')"/>
   </xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

ただし、次のエラー・メッセージが表示されます。

```
Error occurred while processing elName.xsl: XSL-1013: 式：
'starts-with(local-name(),'REF_')' でエラーが発生しました。
```

前述の式は、何が間違っていますか？

回答

次のコードは正常に実行できます。

match="starts-with(..)" は、有効な一致パターンではないため不正です。この構文は、次の一致パターンを使用する必要があります。

```
match="*[starts-with(local-name(.),'REF_')]"
```

このパターンを使用した構文は次のとおりです。

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Identity Transform -->
  <xsl:template match="node()|@*">
    <!-- Copy the current node -->
    <xsl:copy>
      <!-- Including any attributes it has and any child nodes -->
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
 </xsl:template>

<xsl:template priority="2" match="*[starts-with(local-name(),'REF_')]">
  <REF index="{substring-after(local-name(.),'REF_')}">
    <xsl:apply-templates/>
  </REF>
 </xsl:template>
</xsl:stylesheet>
```

これによって、次のドキュメントが変換されます。

```
<foo>
   <bar>
     <REF_STATUS>
        <baz/>
     </REF_STATUS>
     <zoo>
        <REF_USFLG>
           <boo/>
        </REF_USFLG>
     </zoo>
   </bar>
</foo>
```

変換結果は次のとおりです。

```
<foo>
   <bar>
     <REF index="STATUS">
        <baz/>
     </REF>
     <zoo>
        <REF index="USFLG">
           <boo/>
        </REF>
     </zoo>
   </bar>
</foo>
```

XSL のノード・セットへ文字列を変換する方法

質問

受信する XML が、CDATA 表記法を使用した追加コードによってラップされています。XSL が、CDATA セクションの要素を認識しません。何を行う必要があるのか教えてください。

回答

<![CDATA[ ]]> 内は、XPath で問合せを行うための要素および属性ではありません。それらは、単なるリテラル文字（山カッコ、名前および引用符）です。それらは、要素や属性に見えますが、個別のノードとは異なり、情報セット・ツリー内にはありません。

CDATA 内には、単一のテキスト・ノードしかありません。XSL は、CDATA 内の要素は認識しません。最適な方法は、次のとおりです。

- CDATA の文字列の内容を一致させます。
- プログラムでドキュメントを解析し、プログラムで、CDATA ノードを CDATA ノードのコンテンツの解析結果で XML 文書として置換します。
- CDATA の文字列の内容を抽出および解析し、XSLT を使用して処理します。

質問

サンプルの XSL ファイル `toolbar.xsl` では、次の XSL を実行して文字列をノード・セットに変換しているように見えます。

```
<xsl:variable name="barns"select="my:nodeset($bar)"/>
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:my="http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.Extensions"e
xclude-result-prefixes="my">
    <xsl:template match="/">
      <xsl:call-template name="toolbar">
        <xsl:with-param name="bar">
          <toolbar>
             <button name="xxx" url="www.oracle.com"/>
          </toolbar>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:template>
...
```

この考えは正しいでしょうか？CDATA セクションを変数に抽出しましたが、それをノード・セットに変換する必要があります。Transviewer Beans の AsyncTransformSample.java を使用して変換しようとすると、次のエラーが発生します。

```
XSL-01045: 拡張ファンクション・エラー : クラスが見つかりません :
'oracle.xml.parser.v2.Extensions'
```

これは、標準のパッケージに含まれていますか？それともインポートする必要がありますか？次のインポート文は、AsyncTransformSample に含まれています。

```
import oracle.xml.parser.v2.*;
```

回答

文字列はノード・セットに変換されていません。ora:node-set() は、文字列ではなく、結果ツリーのフラグメントをノード・セットに変換します。文字列をノード・セットに変換するには、文字列を解析する必要があります。XSLT には parse-string() 組込み関数がないため、Java の拡張関数として関数を作成できます。Java XSLT 拡張関数の開発およびデバッグの詳細は、『Developing Oracle XML Applications』（Steve Muench 著、O'Reilly 出版）の第 16 章を参照してください。

次の Java クラスの例では、文字列が解析され、解析された XML 文書のルート・ノードを文字列に含むノード・セットが戻ります。解析中にエラーが発生した場合、空のノード・セットが戻ります。

```
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import oracle.xml.parser.v2.*;
import java.io.StringReader;
public class Util {
  public static NodeList parse (String s) {
    // Create a new parser
    DOMParser d = new DOMParser();
    try {
      // Parse the string into an in-memory DOM tree
      d.parse( new StringReader(s) );
      // Return a node list containing the root node
      return ((XMLDocument)d.getDocument()).selectNodes("/");
    }
    catch (Exception e) {
      // Return an empty nodelist in case of an error.
      return (new XMLDocument()).getChildNodes();
    }
  }
}
```

サンプルの message.xml ファイルを次に示します。このファイルは、XML 文書本体に CDATA セクションに囲まれた XML があり、質問のシナリオをシミュレートしています。

```
<message>
  <from>Steve</from>
  <to>Albee</to>
  <body><![CDATA[
    <order id="101">
      <item id="12" qty="10"/>
      <item id="13" qty="3"/>
    </order>
  ]]></body>
</message>
```

次に、サンプルのスタイルシートを示します。このスタイルシートは、<message> ドキュメントを処理し、XSL 変数のサブドキュメント（<body> 内で CDATA テキスト・ノードとしてエンコードされている）を解析および表現し、<xsl:for-each> を使用して、解析されたメッセージ本体を含む $body 変数から情報を選択します。ここでは、<order> の識別子のみを出力しますが、これは一般的な概念を示します。

```
<xsl:stylesheet  version="1.0"
  xmlns:util="http://www.oracle.com/XSL/Transform/java/Util"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!--
   | Above, we've associated the "util" namespace prefix
   | with the appropriate namespace URI that maps to
   | the "Util" class. The Util.java class is not in any
   | package, otherwise the URI would have looked like
   | http://www.oracle.com/XSL/Transform/java/my.pkg.Util
   +-->

  <xsl:template match="message">
    <!--
     | Use the parse() function in the util namespace
     | to parse the string value of the <body> child
     | element of the current <message> element, and
     | return the root node of the document
     +-->
    <xsl:variable name="body" select="util:parse(body)"/>
    <xsl:text>Items Ordered</xsl:text><xsl:text>&#xa;</xsl:text>
    <xsl:for-each select="$body/order/item">
      <xsl:value-of select="@id"/><xsl:text>&#xa;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

XSL で XML 文書のタグを HTML のリンクへ適切に変換する方法

質問

XSL について質問があります。次のような XML 文書があります。

```
<ROW num="1">
  <TITLE>New Java Classes</TITLE>
   <URL>/products/intermedia/</URL>
   <DESCRIPTION>&#60;a href=\"/products/intermedia/\">Java classes for
   Servlets and JSPs&#60;/a>are available.
   </DESCRIPTION>
</ROW>
```

XSL を使用して XML 文書を HTML で表示する場合、XML で DESCRIPTION を「`&#60;a href=\"/products/intermedia/\">`」として指定しても、DESCRIPTION はリンクとして表示されません。

XSL ファイルは次のとおりです。

```
<xsl:template>
  <P><FONT face="arial" size="4"><B>
  <xsl:value-of disable-output-escaping="yes" select="TITLE"/>
  </B> </FONT><BR></BR><FONT size="2">
  <xsl:value-of disable-output-escaping="yes" select="DESCRIPTION"/></FONT>
  </P>
</xsl:template>
```

回答

XSLT で、<a> タグを作成します。方法は次のとおりです。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html"/>
 <xsl:template match="/">
  <xsl:for-each select="rowset">
   <table border="1">
      <th>Category</th>
 <th>ID</th>
 <th>Title</th>
 <th>Thumbnail</th>
  <xsl:for-each select="row">
   <tr>
   <td><xsl:value-of select="category"/> </td>
   <td><xsl:value-of select="id"/> </td>
   <td><a href="Present.jsp?page=PRES_VIEW_SINGLE&amp;id={id}"><xsl:value-of
select="title"/> </a></td>
```

```
   <td><img src="/servlets/thumb?presentation={id}&amp;slide=0" /></td>
  </tr>
 </xsl:for-each>
 </table>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

WML 変換における適切な XSL ヘッダーの使用

質問

XDK for Java に付属の `oracle.xml.async.XSLTransformer` を使用して、XML 文書の XSL 変換を実行しています。WML を出力する必要があります。スタイルシートのコードは次のとおりです。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="no" encoding="ISO-8859-1"/>
<xsl:output doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"
doctype-public="-//WAPFORUM//DTD WML 1.1//EN"/>
...
```

サーブレットを使用して変換を確認しようとすると、WAP エミュレータで次のエラーが発生しました。

```
"Received HTTP status: 502 - WML Encoding Error, 1:com.sun.xml.parser/P-076
Malformed UTF-8 char
```

XML のエンコーディング宣言が不足していますか？生成された WML には、XML ヘッダーの情報が含まれていません。出力の先頭は、次のとおりです。

```
 <wml>
 <card id="gastronomia" title="Mis direcciones de gastronomia"><p>Mis direcciones de
gastronomia</p>
...
```

次のような XML ヘッダーを出力するための変換方法を教えてください。

```
"<?xml version="1.0" encoding="ISO-8859-1"?>"
```

回答

oracle.xml.parser.v2.XSLProcessor を使用します。また、スタイルシートに次のタグがあることを確認してください。

```
<xsl:output method="xml"/>
```

このタグは、<xsl:stylesheet> 内で、すべての <xsl:template> の外にある必要があります。

また、次の API を使用していることを確認してください。

```
processXSL (stylesheet,source,printwriter)
```

XSLT で DTD ファイルの位置を確認する方法

質問

BC4J のソース XML ファイルに、DTD を参照する次の行があります。

```
<!DOCTYPE ViewObject SYSTEM "jbo_03_01.dtd">
```

ファイルを変換する場合、この行で jbo_03_01.dtd が見つからないというエラーが発生します。DTD ファイルは、CLASSPATH 内に存在します。

回答

これには、次の 2 つのソリューションがあります。

- jbomt.zip から jbo_03_01.dtd を解凍して、仮想オブジェクト（VO）ファイルと同じディレクトリに格納します。VO ファイルが複数の異なるディレクトリ・レベルにある場合、この作業は複雑になります。
- プログラムで解析を行う場合は、BC4J 自体が独自の XML メタデータ・ファイルを解析するために使用している方法を使用します。

```
DOMParser d = new DOMParser();
       // read DTD as a resource from the classpath
       InputStream is = ...getResourceAsStream("/jbo_03_01.dtd");
       d.parseDTD( is );
       DTD dtd = d.getDoctype();
       d.setDoctype( dtd ); // set and cache the DTD to use.
       // Now, subsequences calls to d.parse() will
       // use the cached version of jbo_03_01.dtd
```

次に、XSL スタイールシートおよび XSLProcessor.process (style,source,printwriter) を使用して、この結果を変換します。

XSL で名前空間定義の繰返しを回避する方法

質問

名前空間に関連する質問があります。スタイルシートのコードの一部を次に示します。

```
<xsl:attribute name="data:text">
  <xsl:value-of select="@Name"/>@ipet:dataBindingObject
</xsl:attribute>
```

スタイルシートの最初に、marlin 名前空間を定義しました。

```
xmlns:data="http://xxx.us.yyy.com/cabo/marlin"
```

結果の XML ファイル（marlin UIX ファイル）では、名前空間の定義が、要素ごとに繰り返されています。

```
<messageTextInput id="Status" name="Status" prompt="Status"
required="yes"xmlns:data="http://xxx.us.yyy.com/cabo/marlin"
data:text="Status@ipet:dataBindingObject" rows="1" maximumLength="3" columns="3"/>
```

回答

XSLT ルート・テンプレートで、ドキュメント要素にデータの名前空間の接頭辞を定義することをお薦めします。接頭辞が結果ツリーのより上のレベルで定義されている場合、その定義が認識され、その下の各レベルの要素上には出力されません。

JDeveloper9*i* には、VO が含まれます。これらの VO によって、データベースの X$ ビューのような VO のメタデータが表示されます。したがって、これらの仮想メタデータ・ビューのいずれかに対して通常の VO.writeXML() メソッドを使用して、任意の VO の構造に基づいたデータ駆動の出力をレンダリングする操作を行うことができます。

Java プログラムから XSL スタイルシートへパラメータを渡す方法

質問

Oracle XSL Processor を使用して、Java プログラムから XSLT スタイルシートへパラメータを渡す方法がありますか？XSLT 標準には、「（中略）XSLT は、スタイルシートへパラメータを渡すためのメカニズムは定義しない」（http://www.w3.org/TR/xslt#variable-values を参照）と記載されています。これを行うことは可能ですが、ベンダーに依存した実装になります。ただし、OracleXSLprocessor のどの XSL コンストラクタを使用しても、これを実行できないようです。

スタイルシートへ整数を渡し、xsl:position() 関数を使用して、XML 文書からドキュメント・フラグメントを抽出する必要があります。次に例を示します。

```
<xsl:templatematch="ROW">
<xsl:if test="position()=1">
  SELECT DISTINCT sp.site_datatype_id
  FROM ref_hm_site_pcode sp
  WHERE sp.hm_site_code = '<xsl:value-ofselect='HM_SITE_CODE'/>'
  AND sp.hm_pcode = '<xsl:value-ofselect='HM_PCODE'/>'
</xsl:if>
</xsl:template>
```

ただし、position()=1 のかわりに、$1 などのパラメータに置き換える必要があります。

これを行う方法を教えてください。

回答

スタイルシートで、次のように最上位のパラメータを宣言しているとします。

```
<xsl:stylesheet ... >
   <!-- declare top-level $foo parameter, default value to 5 -->
   <xsl:param name="foo" select="5"/>
   <xsl:template match="/">
   <xsl:if test="$foo=10">
     :
```

この場合、oracle.xml.parser.v2.XSLStylesheet で次のメソッドを使用して、パラメータを制御できます。

- resetParams()
- setParam()

foo という名前のパラメータを、数字の 10 に設定するには、次の構文を使用します。

```
myStylesheet.setParam("foo","10");
```

foo を、文字列の ten に設定するには、文字列を引用符で囲む必要があります。

```
myStylesheet.setParam("foo","'ten'");
```

質問

Java プログラムでスタイルシートにパラメータを渡す必要がある場合、どの Java クラスを使用する必要がありますか？

現在、次のクラスを使用しています。

```
processXSL(XSLStylesheet xsl,XMLDocument xml)
```

パラメータを渡すために使用できるメソッドを教えてください。

回答

次のメソッドを使用してください。

- `XSLStylesheet.setParam()`
- `XSLStylesheet.resetParams()`

エラー「XSL-01009 属性 'XSL Version' が 'HTML' で見つかりません。」を解決する方法

質問

`http://metalink.oracle.com` の `Note:104675.1` を参照して、XDK を使用して、Oracle から XML データを取り出し、それを HTML に変換しました。

XML の出力ファイルは生成できましたが、次の引数を持つそのファイル `Emp.xsl` を使用して HTML の出力を生成しようとすると、「XSL-01009 属性 'XSL Version' が 'HTML' で見つかりません。」というエラーが発生しました。

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- XSL ファイルの正しい引数を教えてください。`<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0" version="1.0">` は、正常に実行できましたが、HTML 出力には、純粋なデータのみで HTML タグが含まれていません。
- XML パーサーで生成された HTML 出力を参照したことがないため、XML パーサーが自動的に HTML タグを生成するかどうかがわかりません。

HTML 出力ファイルは、どのようになるのでしょうか？

回答

`xsl:version="1.0"` 属性を、`<html>` 要素に追加する必要があります。

末端の子要素のみ検索する XPath 式

質問

末端の子要素（その下に子要素を持たない要素）のみを、指定された要素から検索する XPath 式を教えてください。たとえば、XPath 式を使用して、次に示す赤色でハイライトされた TABLE の子要素のみを戻す必要があります。

```
<TABLE>
     <ID>1</ID>
     <NAME>1</NAME>
     <SIZE>1</SIZE>
     <COLUMNS>
        <COLUMN>
            <ID>1</ID>
            <NAME>Customers</NAME>
        <COLUMN>
        <COLUMN>
            <ID>c</ID>
            <NAME>Categories</NAME>
        <COLUMN>
     <COLUMNS>
     <DATE_CREATED>01/10/2000</DATE_CREATED>
  </TABLE>
```

回答

1. 次の構文を使用してください。

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="TABLE">
  <xsl:apply-templates select="child::*[not(child::*)]"/>
</xsl:template>
</xsl:stylesheet>
```

2. 必要な XPath 式は、次のとおりです。

```
/TABLE/*[count(child::*) = 0]
```

または

```
/TABLE/*[not (child::*)]
```

child 軸は省略できるため、前述の式は次の式と同じです。

```
/TABLE/*[count(*) = 0]
```

または

```
/TABLE/*[not (*)]
```

XSL スタイルシート適用後に子の属性を戻す方法

質問

XML 文書を XSL スタイルシートとマージしています。XSL ドキュメントで次のような構文を使用する場合、子の属性が戻されません。

```
<xsl:value-of select="Foo/Bar"/>
```

これは、XML Spy や XML Stylus などの他の XML パーサーでは問題なく動作するようです。

回答

XPath 式 `Foo/Bar` は、`<Foo>` 要素に含まれている `<Bar>` 要素の値のみを選択するように設計されています。この式は、`<Bar>` 要素のネストされたコンテンツにあるすべてのテキスト・ノードの連結を戻しますが、属性内のテキスト値を戻すようには設計されていません。

1 つの属性を選択するには、次の構文を使用する必要があります。

```
Foo/Bar/@SomeAttr
```

`<Bar>` のすべての属性を選択するには、次の構文を使用する必要があります。

```
Foo/Bar/@*
```

第 II 部

データベースに対する XML の格納および取出し

第 II 部では、XML データを Oracle データベースに格納する方法とデータベースから取り出す方法、および XSU、SYS.XMLType および DBURI 参照を使用してこのタスクを行う方法について説明します。また、Oracle Text（*inter*Media Text）を使用して XML データの検索および取出しを微調整および高速化する方法も説明します。

第 II 部に含まれる章は、次のとおりです。

5

XML に対するデータベース・サポート

この章の内容は次のとおりです。

Oracle 固有の XML データベース機能

Oracle は、新しいシステム定義のオブジェクト型である XMLType をサポートします。XMLType には、XML データの作成、抽出および索引作成のための強力なメカニズムを提供する組込みメンバー関数があります。ユーザーは、SQL 関数の SYS_XMLGEN と SYS_XMLAGG、および PL/SQL パッケージの DBMS_XMLGEN を使用して、XMLType インスタンスとして XML 文書を動的に生成することもできます。

表 5-1 に、Oracle で固有にサポートされている XML の新機能の概要を示します。

表 5-1 Oracle 固有の XML サポート機能の概要

XML 機能	説明
XMLType	（新機能）XMLType は、XML データにアクセスするための事前定義メンバー関数を持つシステム定義のデータ型です。XMLType を使用して次のタスクを実行できます。 ■ XMLType の列を作成し、その型のインスタンスで XMLType メンバー関数を使用します。5-4 ページの「XMLType データ型」を参照してください。 ■ XMLType を引数および戻りパラメータとして使用して、PL/SQL ファンクションおよびプロシージャを作成します。5-9 ページの「XMLType を使用する場合」を参照してください。 ■ XML データを XMLType 列に格納して、索引を作成し、操作します。5-4 ページの「XMLType データ型」を参照してください。
DBMS_XMLGEN	（新機能）DBMS_XMLGEN は、SQL 問合せの結果を正規の XML 形式に変換し、それを XMLType または CLOB として戻す PL/SQL パッケージです。DBMS_XMLGEN は C で実装され、データベース・カーネルでコンパイルされます。DBMS_XMLGEN は、機能的に DBMS_XMLQuery パッケージと類似しています。 5-32 ページの「DBMS_XMLGEN」を参照してください。

表 5-1 Oracle 固有の XML サポート機能の概要（続き）

XML 機能	説明
SYS_XMLGEN	（新機能）SYS_XMLGEN は、SQL 問合せ内で XML を生成する SQL 関数です。DBMS_XMLGEN およびその他のパッケージは問合せレベルで機能し、問合せ全体の集計結果を戻します。SYS_XMLGEN は、SQL 問合せ内の単一の引数で機能し、結果を XML に変換します。 SYS_XMLGEN は、スカラー値、オブジェクト型または XMLType インスタンスを受け入れ、XML 文書に変換します。また、オプションの XMLGenFormatType オブジェクトを使用して、結果の書式設定オプションを指定します。SYS_XMLGEN は XMLType を戻します。 5-64 ページの「SYS_XMLGEN」を参照してください。
SYS_XMLAGG	（新機能）SYS_XMLAGG は、一連の XMLType で集計を行う集計関数です。SYS_XMLAGG はすべての入力 XML 文書またはフラグメントを集計し、XML フラグメントを連結して、最上位のタグを追加することによって 1 つの XML 文書を生成します。 5-73 ページの「SYS_XMLAGG」を参照してください。
UriType	（新機能）UriType 型グループは、データベースに URI 参照を格納し、問い合せることができます。SYS.UriType は抽象オブジェクト型で、URL が指すデータにアクセスするための機能を提供します。SYS.HttpUriType および SYS.DBUriType は UriType のサブタイプです。HttpUriType は HTTP の URL を格納でき、DBUriType はデータベース内参照を格納できます。また、SYS.UriType の独自のサブタイプを定義して、別の URL プロトコルを処理することもできます。 **UriFactory パッケージ :** これは、http:// や ftp:// などの接頭辞をスキャンすることによって、これらの UriType のインスタンスを自動的に生成できるファクトリ・パッケージです。また、サポートされる接頭辞を指定して、UriFactory に独自のサブタイプを登録することもできます。たとえば、gopher プロトコルを処理するためのサブタイプを UriFactory に登録し、「gopher://」という接頭辞を持つ URL をそのサブタイプで処理するように指定できます。その結果、UriFactory は、接頭辞 gopher で始まるすべての URL に対して、登録したサブタイプのインスタンスを生成します。 第 6 章「DBURI 参照」を参照してください。

XMLType データ型

XMLType は、表の列およびビューとして使用できる新しいサーバー・データ型です。XMLType の変数は、PL/SQL ストアド・プロシージャのパラメータ、戻り値などとして使用できます。XMLType は、PL/SQL、SQL および Java を介して、サーバー内で使用できます。現在のところ OCI を介してはサポートされていません。

XMLType には、XML コンテンツを操作する有効な組込みメンバー関数が含まれています。たとえば、Extract 関数は、XMLType インスタンスから特定のノード（複数の場合もあります）を抽出します。

XML 文書を戻す SYS_XMLGEN などの新しい SQL 関数は、文書を XMLType として戻します。これによって、SQL 文で厳密な型付けができます。SQL 問合せ内の XMLType は、システムの他のユーザー定義データ型と同じ方法で使用できます。

注意： 今回のリリースでは、XMLType は SQL、PL/SQL および Java を介してのみ、サーバーでサポートされています。XMLType をクライアント側で使用するには、クライアント側で OCI または OCCI、および getClobVal() などの関数を使用して、完全な XML 文書を取り出してください。

XMLType の使用方法

XMLType を使用すると、表の列を作成できます。XMLType の createXML() 関数を使用すると、挿入用の XMLType インスタンスを作成できます。XML 文書を XMLType として格納することによって、標準の SQL 問合せを使用して簡単に XML コンテンツを検索できます。

この項では、XMLType 列を作成し、それを SQL 文で使用する方法を示す簡単な例を示します。

XMLType 列の作成例

XMLType 列は、他のユーザー定義型の列と同様に作成できます。

```
CREATE TABLE warehouses(
  warehouse_id NUMBER(3),
  warehouse_spec SYS.XMLTYPE,
  warehouse_name VARCHAR2(35),
  location_id NUMBER(4));
```

XMLType 列への値の挿入例

XMLType 列に値を挿入するには、XMLType インスタンスをバインドする必要があります。XMLType インスタンスは、XMLType の createXML() 関数を使用して、VARCHAR2 または CLOB から簡単に作成できます。

```
INSERT into warehouses (warehouse_id, warehouse_spec)
   VALUES (1001, sys.XMLType.createXML(
                    '<Warehouse whNo="100">
                       <Building>Owned</Building>
                     </Warehouse>'));
```

この例では、文字列リテラルから XMLType インスタンスを作成しています。createXML への入力には、VARCHAR2 または CLOB を戻す、すべての式を使用できます。

createXML 関数は、入力 XML が整形式かどうかも確認します。XML の妥当性は確認しません。

SQL 文内での XMLType の使用例

次の単純な SELECT 文は、SQL 文で XMLType を使用する方法を示します。

```
SELECT
  w.warehouse_spec.extract('/Warehouse/Building/text()').getStringVal()
     "Building"
FROM warehouses w
```

ここで、warehouse_spec はメンバー関数 Extract() によって操作される XMLType 列です。この単純な問合せの結果は、次の文字列（VARCHAR2）になります。

```
Building
-----------------
Owned
```

参照： 5-5 ページの「XMLType の使用方法」を参照してください。

XMLType 列の更新例

今回のリリースでは、XMLType の XML 文書は CLOB 内に格納されます。したがって、更新で既存の文書を置換する必要があります。今回のリリースでは、XML のピース単位の更新はサポートされていません。

XML 文書を更新するには、XMLType インスタンスをバインドする場合を除き、標準の SQL UPDATE 文を発行します。

```
UPDATE warehouses SET warehouse_spec =
           sys.XMLType.createXML(
                    '<Warehouse whono="200">
                       <Building>Leased</Building>
                     </Warehouse>'));
```

この例では、文字列リテラルから XMLType インスタンスを作成し、warehouse_spec 列を新しい値で更新します。UPDATE 文ではすべてのトリガーが起動され、そのトリガー内で XML 値を確認および変更できます。

XMLType 列を含む行の削除例

XMLType 列を含む行を削除する方法は、他のデータ型の場合と同じです。

Extract 関数および ExistsNode 関数を使用しても、削除する行を識別できます。たとえば、ウェアハウスの建物が Leased になっているすべての warehouse 行を削除するには、次の文を使用します。

```
DELETE FROM warehouses e
   WHERE e.warehouse_spec.extract('//Building/text()').getStringVal()
           = 'Leased';
```

XMLType 列を使用する場合のガイドライン

XML データを XMLType 列に格納する場合のガイドラインは次のとおりです。

- **XMLType 列を定義します。**まず、XMLType 列を定義します。列定義にはオプションの記憶特性を含めることができます。
- **XMLType インスタンスを作成します。**XML データを列に挿入する前に、XMLType コンストラクタを使用して、XMLType インスタンスを作成します。SYS_XMLGEN 関数および SYS_XMLAGG 関数を使用して、直接 XMLType のインスタンスを作成することもできます。5-69 ページの「SYS_XMLGEN の例 3: XMLType インスタンスの変換」および 5-75 ページの「SYS_XMLAGG の例 2: 表に格納された XMLType インスタンスを集計する場合」を参照してください。
- **特定の XMLType インスタンスを選択または抽出します。**XMLType インスタンスは列から選択できます。XMLType では、メンバー関数 extract() を使用して特定のノードを抽出したり、メンバー関数 existsNode() を使用してノードが存在するかどうかを確認することができます。表 5-3「XMLType メンバー関数および静的関数」を参照してください。

  参照： 次の項を参照してください。

  - 5-30 ページの「XMLType の問合せ例 6: XMLType からのフラグメントの抽出」
  - 5-22 ページの「XMLType の問合せ例 1: CLOB としての XML 文書の取出し」

- **Oracle Text 索引を定義できます。**XMLType 列に Oracle Text（*inter*Media Text）索引を定義できます。これによって、その列で CONTAINS、HASPATH、INPATH およびその他のテキスト演算子を使用できます。LOB 列を操作するすべてのテキスト演算子および索引関数は、XMLType 列も操作します。

  参照： 次の項、章およびマニュアルを参照してください。

  - 5-32 ページの「XMLType 列の索引付け」
  - 第 8 章「Oracle Text を使用した XML データの検索」
  - 『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』

XMLType のメリット

XMLType を使用する場合のメリットは次のとおりです。

- XML と SQL が統合され、次のタスクを実行できます。
  - XML コンテンツに対する SQL 操作
  - SQL コンテンツに対する XML 操作
- 組込み関数、索引付けサポート、ナビゲーションなどが含まれているため、実装に便利です。

XMLType と他の SQL 構文の相互作用

XMLType は、他の列およびデータ型と組み合せて SQL 文で使用できます。たとえば、XMLType 列を問い合せ、抽出の結果をリレーショナル列と結合できます。Oracle は、これらの問合せを実行するための最適な方法を判断できます。

結果の XML に対するツリー形式またはシリアル化形式の選択

XMLType は、必要な場合以外は、XML データがツリー構造に生成されないように最適化されています。したがって、SQL 問合せで XMLType インスタンスを選択すると、シリアル化形式のみが関数を越えて交換されます。これらのインスタンスは、extract() や existsNode() などの操作が実行されるときのみ、ツリー形式に分解されます。XMLType の内部構造も、DOM に類似した最適化されたツリー構造です。

XMLType に対するファンクション索引および Text 索引の作成

Oracle Text 索引は、XMLType 列もサポートするように拡張されています。ExistsNode 関数および Extract 関数にもファンクション索引を作成して、問合せの評価を高速化できます。

> **参照：** 第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

XMLType を使用する場合

XMLType は次の場合に使用します。

- XML 全体をデータベースに格納し、取り出す必要がある場合。
- 文書の一部または全体に SQL 問合せを実行する必要がある場合。ExistsNode 関数および Extract 関数を使用すると、XML 文書に SQL 問合せを実行できます。
- SQL 文および PL/SQL ファンクションで厳密な型付けが必要な場合。厳密な型付けとは、渡される値が常に XML 値であり、任意のテキスト文字列ではないことを意味します。
- XML 文書を処理するために、Extract 関数および ExistsNode 関数で提供される XPath 機能が必要な場合。XMLType は、組込みの C 言語で書かれた XML パーサーおよび XSLT プロセッサを使用するため、サーバー内部で使用されるとパフォーマンスおよびスケーラビリティが向上することに注意してください。
- 文書の XPath 検索に索引付けが必要な場合。XMLType には、ファンクション索引を作成して検索を最適化するためのメンバー関数があります。
- 文書のピース単位の更新が必要ない場合。
- 記憶域モデルからのアプリケーションの保護が必要な場合。将来のリリースでは、XMLType は様々な種類の記憶域をサポートします。CLOB またはリレーショナル記憶域のかわりに XMLType を使用すると、アプリケーションでの問合せまたはデータ操作言語（DML）文に影響することなく、後でアプリケーションが様々な代替の記憶域に効率的に移行できます。
- 将来の最適化に準備する場合。XML 関連のすべての新機能は、XMLType のみをサポートします。サーバーは、XMLType が XML データしか格納できないことを固有に認識するため、将来はより優れた最適化および索引付けの方法を実現できます。XMLType を使用するアプリケーションを作成することによって、アプリケーションを後で作成しなおすことなく、これらの最適化および拡張を簡単に実現できます。

データベース内の XMLType 記憶領域

今回のリリースでは、XMLType は単一の CLOB 記憶領域オプションを提供します。将来のリリースでは、Oracle が BLOB や NCLOB など、他の記憶領域オプションを提供する可能性もあります。

XMLType 列を作成すると、XML データを格納するために自動的に非表示 CLOB 列が作成されます。XMLType 列自体は、この非表示 CLOB 列上の仮想列になります。CLOB 列には直接アクセスできませんが、XMLType 記憶域句を使用してその列の記憶特性を設定できます。

VARRAY は表に格納されるときには CLOB をサポートしないため、XMLType の VARRAY を作成して、データベースに格納することはできません。

注意： XMLType を含む VARRAY 型の列は作成できません。これは、Oracle が VARRAY 内の LOB ロケータをサポートせず、（現在）XMLType が常に XML データを CLOB に格納するためです。

XMLType の作成例 1: XMLType 列の作成

前述のとおり、XMLType を単にデータ型として使用することによって、XMLType 列を作成できます。

次の文では、XMLType の発注書列を作成します。

```
CREATE TABLE po_xml_tab(
  poid number,
  poDoc SYS.XMLTYPE);
```

XMLType の作成例 2: XMLType 列の追加

表を変更して XMLType 列を追加することもできます。この方法は、他のデータ型と同様です。

次の文では、表に新しい顧客ドキュメント列を追加します。

```
ALTER TABLE po_xml_tab add (custDoc sys.XMLType);
```

XMLType の作成例 3: XMLType 列の削除

他のデータ型の方法と同様に、表を変更して XMLType 列を削除できます。

次の文では、custDoc 列を削除します。

```
ALTER TABLE po_xml_tab drop (custDoc sys.XMLType);
```

XMLType 列に対する記憶特性の指定

前述のとおり、XMLType 列の XML データは CLOB 列として格納されます。CLOB 列には、LOB の記憶特性を指定することもできます。前述の例では、warehouse spec 列は XMLType 列です。

図 5-1 に、XMLType 記憶域句の構文を示します。

図 5-1 XMLType 記憶域句の構文

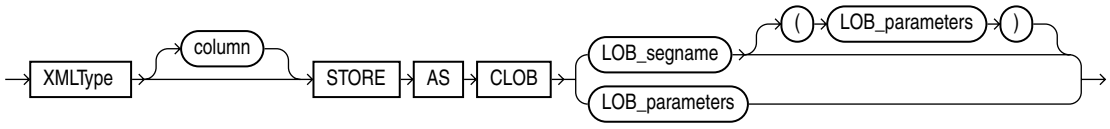


表 5-2 に、XMLType 記憶域句の構文について説明します。

表 5-2 XMLType 記憶域句の構文の説明（図 5-1 を参照）

構文メンバー	説明
column	表の表領域および記憶特性とは異なる表領域および記憶特性を明示的に定義する、LOB 列名または LOB オブジェクト属性を指定します。Oracle は、作成する列ごとに自動的にシステム管理索引を作成します。
LOB_segname	LOB データ・セグメントの名前を指定します。複数の *LOB_item* を指定する場合、*LOB_segname* は使用できません。
LOB_parameters	*LOB_parameters* 句を使用すると、LOB 記憶域の様々な要素を指定できます。 ■ ENABLE STORAGE IN ROW: 行の記憶域を使用可能にする場合、LOB 値の長さが約 4000 バイトからシステム制御情報を引いた長さより小さいときは、LOB 値が行内に格納されます。これはデフォルトです。 **制限 :** IOT の場合、*index_org_table_clause* に OVERFLOW セグメントを指定しないかぎり、このパラメータは指定できません。 ■ DISABLE STORAGE IN ROW: 行の記憶域を使用禁止にする場合、LOB 値の長さにかかわらず LOB 値が行外に格納されます。 **注意 :** LOB ロケータは、LOB 値の格納場所にかかわらず常に行内に格納されます。一度 STORAGE IN ROW の値を設定すると、表を移動しないかぎり変更できません。『Oracle9*i* SQL リファレンス』の「ALTER TABLE」の「move_table_clause」を参照してください。 ■ CHUNK *integer*: LOB の操作用に割り当てるバイトを指定します。*integer* がデータベース・ブロック・サイズの倍数でない場合、Oracle は次の倍数（バイト単位）に切り上げます。データベース・ブロック・サイズが 2048 で、*integer* が 2050 の場合、Oracle は 4096 バイト（2 ブロック）を割り当てます。最大値は 32768（32K）です。デフォルトの CHUNK サイズは Oracle データベース・ブロック 1 つ分です。一度 CHUNK の値を設定すると、変更できません。 **注意 :** CHUNK の値は NEXT（*storage_clause* のデフォルト値または指定された値）の値以下である必要があります。CHUNK が NEXT の値を超えると、Oracle はエラーを戻します。 ■ PCTVERSION *integer*: 新しいバージョンの LOB を作成するために使用する、LOB 記憶領域全体に対する割合の最大値を指定します。デフォルト値は 10 です。これは、古いバージョンの LOB データは、LOB 記憶領域全体の 10% が使用されるまで上書きされないことを意味します。

次のとおり、表を作成するときにこの列の記憶特性を指定できます。

```
CREATE TABLE po_xml_tab(
     poid NUMBER(10),
     poDoc SYS.XMLTYPE
     )
     XMLType COLUMN poDocument
        STORE AS CLOB (
            TABLESPACE lob_seg_ts
            STORAGE (INITIAL 4096 NEXT 4096)
            CHUNK 4096 NOCACHE LOGGING
         );
```

表に列を追加するときは、記憶域句もサポートされます。この表に新しい XMLType 列を追加し、その列に記憶域句を指定するには、次の構文を使用します。

```
ALTER TABLE po_xml_tab  add(
     custDoc SYS.XMLTYPE
   )
   XMLType COLUMN custDoc
      STORE AS CLOB (
           TABLESPACE lob_seg_ts
           STORAGE (INITIAL 4096 NEXT 4096)
           CHUNK 4096 NOCACHE LOGGING
         );
```

参照： LOB 記憶領域オプションの詳細は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

XMLType 列に対する制約の指定

XMLType 列には NOT NULL 制約を指定できます。次に例を示します。

```
CREATE TABLE po_xml_tab (
  poid number(10),
  poDoc sys.XMLType NOT NULL
);
```

これによって、次のような挿入が回避されます。

```
INSERT INTO po_xml_tab (poDoc) VALUES (null);
```

ALTER TABLE 文を使用して、他の型の列の場合と同じ方法で XMLType 列の NOT NULL 情報を変更することもできます。

```
ALTER TABLE po_tab MODIFY (poDoc NULL);
ALTER TABLE po_tab MODIFY (poDoc NOT NULL);
```

このデータ型では、デフォルト値および他のチェック制約はサポートされていません。

XMLType 関数

Oracle では、新しい SQL 関数の ExistsNode および Extract が導入されています。これらの関数は、XMLType の値を操作します。

existsNode() 関数は、その実装のために XMLType.existsNode() メンバー関数を使用します。ExistsNode 関数の構文は次のとおりです。

```
existsNode ( "XMLType_instance" IN sys.XMLType,
             "XPath_string" IN VARCHAR2) RETURN NUMBER
```

extract() 関数は XPath 式を適用し、結果の XML フラグメントを含む XMLType を戻します。構文は次のとおりです。

```
extract ( "XMLType_instance" IN sys.XMLType,
          "XPath_string" IN VARCHAR2) RETURN sys.XMLType;
```

注意： 今回のリリースでは、SQL 関数 existsNode() および extract() は機能の実装のみが行われています。将来のリリースでは、これらの関数には新しい索引が使用され、最適化がさらに進みます。

表 5-3 に、XMLType のすべての SQL 関数とメンバー関数、およびそれらの構文と説明をリストします。

SQL 文では、ExistsNode および Extract メンバー関数のかわりに SQL 関数を使用できます。すべての XMLType 関数は、組込みの C 言語で書かれた XML パーサーおよび XSLT プロセッサを使用して、XML データを解析して妥当性を検証し、そのデータに XPath 式を適用します。また、最適化されたメモリー内 DOM ツリーを使用して、抽出などの処理を実行します。

表 5-3 XMLType メンバー関数および静的関数

XMLType 関数	構文の概要	説明
createXML()	STATIC FUNCTION createXML(xmlval IN varchar2) RETURN sys.XMLType deterministic	文字列から XMLType インスタンスを作成する静的関数です。XML 値が整形式かどうかを確認します。 パラメータ : xmlval (IN) - XML 文書を含む文字列。 戻り値 : XMLType インスタンス。文字列には整形式の XML 文書が含まれている必要があります。 5-30 ページの「XMLType の問合せ例 6: XMLType からのフラグメントの抽出」、およびこの章の他の例を参照してください。
createXML()	STATIC FUNCTION createXML(xmlval IN clob) RETURN sys.XMLType deterministic	CLOB から XMLType インスタンスを作成する静的関数です。XML 値が整形式かどうかを確認します。 パラメータ : xmlval (IN) - XML 文書を含む CLOB。 戻り値 : XMLType インスタンス。CLOB には整形式の XML 文書が含まれている必要があります。 5-26 ページの「XMLType の問合せ例 2: extract() および existsNode() の使用」、およびこの章の他の例を参照してください。
existsNode()	MEMBER FUNCTION existsNode(xpath IN varchar2) RETURN number deterministic	任意の XPath 式で、文書に適用された XPath が有効なノードを戻すかどうかを確認します。 パラメータ : xpath (IN) - テストする XPath 式。 戻り値 : XPath 式がどのノードも戻さない場合は 0（ゼロ）、それ以外は 1。XPath 文字列が NULL か、文書が空の場合は、0（ゼロ）が戻されます。 5-26 ページの「XMLType の問合せ例 2: extract() および existsNode() の使用」を参照してください。
extract()	MEMBER FUNCTION extract(xpath IN varchar2) RETURN sys.XMLType deterministic	任意の XPath 式で、XPath を文書に適用し、フラグメントを XMLType として戻します。 パラメータ : xpath (IN) - 適用する XPath 式。 戻り値 : 結果ノードを含む XMLType インスタンス。XPath がどのノードも戻さない場合、NULL になります。 5-30 ページの「XMLType の問合せ例 6: XMLType からのフラグメントの抽出」を参照してください。

表 5-3 XMLType メンバー関数および静的関数（続き）

XMLType 関数	構文の概要	説明
isFragment()	MEMBER FUNCTION isFragment() RETURN number	文書がフラグメントかどうかを確認します。多くのノードを戻す可能性のある XML 文書で EXTRACT またはその他の操作を実行した場合、フラグメントが存在する可能性があります。 戻り値 : XML がフラグメントの場合は 1、整形式文書の場合は 0 を戻します。 5-30 ページの「XMLType の問合せ例 6: XMLType からのフラグメントの抽出」を参照してください。
getClobVal()	MEMBER FUNCTION getClobVal() RETURN clob deterministic	文書を CLOB として取得します。 戻り値 : シリアル化 XML 表現を含む CLOB。使用後はテンポラリ CLOB を解放してください。 5-57 ページの「XMLType の問合せ例 1: CLOB としての XML 文書の取出し」を参照してください。
getStringVal()	MEMBER FUNCTION getStringVal() RETURN varchar2 deterministic	XML 値を文字列として取得します。 戻り値 : シリアル化 XML 表現を含む文字列、またはテキスト・ノードの場合はそのテキスト。XML 文書が VARCHAR2 の最大サイズ（4000 バイト）より大きい場合、実行時にエラーが発生します。 5-20 ページの「XMLType の削除例 1: extract を使用した行の削除」および 5-5 ページの「XMLType の使用方法」を参照してください。
getNumberVal()	MEMBER FUNCTION getNumberVal() RETURN number deterministic	XMLType が数値として指す数値を取得します。 戻り値 : XMLType インスタンスが指すテキスト値からフォーマットされる数値。XMLType は、数値を含む有効なテキスト・ノードを指す必要があります。 5-26 ページの「XMLType の問合せ例 2: extract() および existsNode() の使用」を参照してください。

参照： extract()、existsNode()、getClobVal() およびその他の関数の使用方法については、5-5 ページの「XMLType の使用方法」にある例を参照してください。

XMLType 列の XML データの操作

XMLType はユーザー定義のデータ型で、ファンクションが定義されているため、XMLType にファンクションをコールし、結果を取得することができます。XMLType は、ユーザー定義型を使用している場所には常に使用できます。これには、表の列、ビュー、トリガー本体、型定義などが含まれます。

XMLType 列の XML データには次の操作（DML）を実行できます。

- XML データの挿入
- XML データの更新
- XML データの削除

XMLType 列への XML データの挿入

次の方法で、データを XMLType 列に挿入できます。

- INSERT 文（SQL、PL/SQL、C（OCI）および Java）を使用する方法
- SQL*Loader を使用する方法

XMLType 列には整形式の XML 文書のみ格納できます。このような列には、フラグメントおよびその他の非整形式の XML は格納できません。

INSERT 文の使用

INSERT 文を使用して XML データを XMLType に挿入する場合、最初に XML 文書を作成し、それを使用して挿入を実行する必要があります。次の方法で、挿入可能な XML 文書を作成できます。

1. XMLType コンストラクタ SYS.XMLType.createXML() を使用する方法。この方法は SQL、PL/SQL、C（OCI）および Java で実行できます。
2. SQL 関数 SYS_XMLGEN および SYS_XMLAGG を使用する方法。この方法は PL/SQL、SQL、C（OCI）および Java で実行できます。

XMLType の挿入例 1: CLOB での createXML() の使用

次の例では、INSERT...SELECT および XMLType.createXML() コンストラクタを使用して、最初に XML 文書を作成してから、その文書を XMLType 列に挿入します。

たとえば、po_clob_tab が XML 文書を格納する CLOB を含む表である場合、次の構文を使用します。

```
CREATE TABLE po_clob_tab
(
  poid number,
  poClob CLOB
);

-- some value is present in the po_clob_tab
INSERT INTO po_clob_tab
     VALUES(100, '<?xml version="1.0"?>
                    <PO pono="1">
                       <PNAME>Po_1</PNAME>
                       <CUSTNAME>John</CUSTNAME>
                       <SHIPADDR>
                         <STREET>1033, Main Street</STREET>
                         <CITY>Sunnyvalue</CITY>
                         <STATE>CA</STATE>
                       </SHIPADDR>
                    </PO>');
```

これによって、別の po_clob_tab に格納されている CLOB データから XML インスタンスを作成するのみで、発注書の XML 文書を po_tab 表に挿入できます。

```
INSERT INTO po_xml_tab
       SELECT poid, sys.XMLType.createXML(poClob)
       FROM po_clob_tab;
```

CLOB 値は、テンポラリ CLOB を作成できる関数、または他の表やビューから CLOB を選択できる関数を含むすべての式から取得できることに注意してください。

XMLType の挿入例 2: 文字列での createXML() の使用

次の例では、createXML() コンストラクタを使用して po_tab 表に発注書を挿入します。

```
insert into po_xml_tab
   VALUES(100, sys.XMLType.createXML('<?xml version="1.0"?>
                    <PO pono="1">
                       <PNAME>Po_1</PNAME>
                       <CUSTNAME>John</CUSTNAME>
                       <SHIPADDR>
                         <STREET>1033, Main Street</STREET>
                         <CITY>Sunnyvalue</CITY>
                         <STATE>CA</STATE>
                       </SHIPADDR>
                    </PO>'));
```

文字列リテラルを渡すことによって、createXML 関数を使用して XMLType が作成されています。

XMLType の挿入例 3: SYS_XMLGEN() の使用

次の例では、SQL 関数 SYS_XMLGEN() を使用して生成された PurchaseOrder（PO）を、po_tab 表に挿入します。SQL 関数 SYS_XMLGEN() については、この章の後半で説明します。PO は、発注書オブジェクトを含むオブジェクト・ビューと想定します。PO ビューの完全な定義については、5-57 ページの「DBMS_XMLGEN の例 5: XML 形式でのデータベースからの発注書の生成」を参照してください。

```
INSERT into po_xml_tab
  SELECT SYS_XMLGEN(value(p),
                sys.xmlgenformatType.createFormat('PO'))
  FROM po p
  WHERE p.pono=2001;
```

SYS_XMLGEN は発注書オブジェクトから XMLType を作成し、作成された XMLType は po_xml_tab 表に挿入されます。

XMLType 列の XML データの更新

XML 文書全体の更新のみが可能です。更新は、SQL、PL/SQL、C（OCI）または Java で実行できます。

Java を介して XMLType を更新する方法は、5-38 ページの「XMLType の Java の例 4: XMLType 列の要素の更新」を参照してください。

XMLType の更新例 1: createXML() を使用した更新

次の例では、createXML() コンストラクタを使用して XMLType を更新します。この例では、発注書番号が 2001 のドキュメントのみが更新されます。

```
update po_xml_tab e
set  e.poDoc = sys.XMLType.createXML(
'<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>Nance</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>')
WHERE e.po.extract('/PO/@PONO').getNumberVal() = 2001;
```

注意： 現在、UPDATE はドキュメント・レベルでのみサポートされています。したがって、特定のドキュメントの一部を更新するには、ドキュメント全体を更新する必要があります。

XML データの削除

XMLType 列を含む行での DELETE は、他のデータ型の場合と同じ方法で実行されます。

XMLType の削除例 1: extract を使用した行の削除

たとえば、発注書名が「Po_2」のすべての発注書行を削除するには、次の文を実行します。

```
DELETE from po_xml_tab e
WHERE e.poDoc.extract('/PO/PNAME/text()').getStringVal()='Po_2';
```

トリガー内での XMLType の使用

トリガー内で NEW バインドおよび OLD バインドを使用して、XMLType 列値を読み取り、変更することができます。INSERT 文および UPDATE 文の場合、NEW 値を変更して、挿入する値を変更できます。

XMLType トリガーの例 1

たとえば、発注書に納品先が記載されていない場合、トリガーを作成して、発注書を別のものに変更できます。

```
CREATE OR REPLACE TRIGGER po_trigger
    BEFORE insert or update on po_xml_tab for each row
  pono Number;
begin

   if INSERTING then
     if :NEW.poDoc.existsnode('//SHIPADDR') = 0 then
      :NEW.poDoc := sys.xmltype.createxml('<PO>INVALID_PO</PO>'); end if;
     end if;

     -- when updating, if the old poDoc has purchase order number
     -- different from the new one then make it an invalid PO.
     if UPDATING then

        if :OLD.poDoc.extract('//@PONO/text()').getNumberVal() !=
            :NEW.poDoc.extract('//@PONO/text()').getNumberVal() then

          :NEW.poDoc := sys.xmltype.createXML('<PO>INVALID_PO</PO>');
        end if;
     end if;
end;
/
```

この例は、単なる説明用です。XMLType 値を使用して、トリガー内で有効な操作（XML 文書が準拠する必要のあるビジネス・ロジックや規則の検証、監査など）を実行できます。

XML データの選択および問合せ

次の方法で、XMLType 列から XML データを問い合せることができます。

- SQL、PL/SQL、C（OCI）または Java を介して XMLType 列を選択する方法
- 直接 XMLType 列を問い合せ、extract() または existsNode()（あるいはその両方）を使用する方法
- Text 演算子を使用して XML コンテンツを問い合せる方法

XML データの選択

XMLType データは、PL/SQL または Java を介して選択できます。また、getClobVal()、getStringVal() または getNumberVal() 関数を使用して、XML をそれぞれ CLOB、VARCHAR2 または数値として取得することもできます。

XMLType の問合せ例 1: CLOB としての XML 文書の取出し

次の例は、SQL*Plus を介して XMLType 列を選択する方法を示します。

```
set long 2000

select e.poDoc.getClobval() AS poXML
from po_xml_tab e;

POXML
---------------------
<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>Nance</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>
```

XML データの問合せ

ExistsNode 関数および Extract 関数を使用して、XMLType データを問い合せて、その一部を抽出できます。これらの関数はいずれも W3C 標準である XPath の一部を使用して、ドキュメントをナビゲートします。

検索のための XPath 式の使用

XPath は、XML 文書をナビゲートするための W3C 標準の方法です。XPath は、XML 文書をノードのツリーとしてモデル化します。XPath は、ツリー内を移動し、述語およびノード・テスト関数を適用するための、豊富な操作を提供します。XPath 式を XML 文書に適用すると、ノードの集合が戻されます。たとえば、/PO/PONO は、文書の「PO」ルート要素の下にあるすべての「PONO」子要素を選択します。

XPath で使用する共通構造体は次のとおりです。

「/」は、XPath 式のツリーのルートを示します。たとえば、/PO は、名前が「PO」であるルート・ノードの子を示します。

「/」は、任意のノードの子ノードを識別するための区切りとしても使用します。たとえば、/PO/PNAME は、ルート要素の子である発注書名要素を示します。

「//」は、現行ノードのすべての子孫を識別するために使用します。たとえば、PO//ZIP は、「PO」要素の下にあるすべての「ZIP」要素と一致します。

「*」は、任意の子ノードと一致させるためのワイルド・カードとして使用します。たとえば、/PO/*/STREET は、「PO」要素の孫であるすべての「STREET」要素と一致します。

[ ] は、述語式を示すために使用します。XPath は OR、AND、NOT など、様々なバイナリ演算子をサポートしています。
たとえば、/PO[PONO=20 and PNAME="PO_2"]/SHIPADDR は、発注書番号が 20 で、発注書名が「PO_2」であるすべての発注書の出荷先要素を選択します。

[ ] は、リスト内の索引を示す場合にも使用します。たとえば、/PO/PONO[2] は「PO」ルート要素の下の 2 番目の発注書番号要素を示します。

サポートされている XPath 構造体

Oracle の Extract 関数および ExistsNode 関数は、限られた数の XPath 式のみサポートします。今回のリリースでサポートされている XPath 構造体は、次のとおりです。

- 子検索 : /PO/SHIPADDR/STREET
- 属性検索 : /PO/@PONO
- 索引アクセス : /PO/PONO[2]
- ワイルド・カード検索 : /PO/*/STREET
- 子孫検索 : PO//STREET
- ノード・テスト関数 : /PO/PNAME/text()

child 軸、descendant 軸などの非順序依存軸のみがサポートされています。sibling 軸または parent 軸はサポートされていません。

注意 : 現在、Extract 関数および ExistsNode 関数は、マルチバイト・キャラクタ・セットをサポートしていません。

述語の未サポート 今回のリリースでは、XMLType は述語をサポートしていません。述語のサポートが必要な場合、関数を、複数の関数と SQL で表現した述語に再作成できます（可能な場合）。

また、XPath は単一または一連の要素、テキストまたは属性ノードを識別する必要があります。XPath の結果はブール式であってはなりません。

XPath での ExistsNode 関数

XMLType に ExistsNode 関数を使用すると、任意の XPath 評価の結果が、少なくとも単一の XML 要素またはテキスト・ノードになるかどうかが確認されます。単一の XML 要素またはテキスト・ノードになる場合は数値 1 が戻り、それ以外の場合は 0（ゼロ）が戻ります。たとえば、次の XML 文書を考えてみます。

```
<PO>
  <PONO>100</PONO>
  <PNAME>Po_1</PNAME>
  <CUSTOMER CUSTNAME="John"/>
  <SHIPADDR>
    <STREET>1033, Main Street</STREET>
    <CITY>Sunnyvalue</CITY>
    <STATE>CA</STATE>
  </SHIPADDR>
</PO>
```

/PO/PNAME のような XPath 式の結果は、単一のノードになるため、ExistsNode はその XPath に対して TRUE を戻します。これは、結果が単一のテキスト・ノードになる /PO/PNAME/text() の場合も同じです。XPath の /PO/@pono も値を戻します。

/PO/POTYPE のような XPath 式はノードを戻さないため、この式では ExistsNode は数値の 0（ゼロ）を戻します。

したがって、ExistsNode() メンバー関数は次の場合に直接使用できます。

- 後述のいくつかの例の問合せ内
- ファンクション索引を作成して、問合せの評価を高速化するため

XPath での Extract 関数

XMLType に Extract 関数を使用すると、XPath 式によって識別された文書からノードまたはノードの集合が抽出されます。抽出されるノードは、要素、属性またはテキスト・ノードの場合があります。すべてのテキスト・ノードは、抽出時に単一のテキスト・ノード値に縮小されます。

Extract を介して XPath を適用した結果の XMLType は、整形式の XML 文書である必要はありませんが、場合によってはノードの集合または単純なスカラー・データを含んでいる可能性があります。XMLType に getStringVal() または getNumberVal() メソッドを使用すると、このスカラー・データを抽出できます。

たとえば、XPath 式 /PO/PNAME は、前述の XML 文書内の PNAME 要素を識別します。一方、式 /PO/PNAME/text() は、PNAME 要素のテキスト・ノードを参照します。後者の式でも、XMLType とみなされることに注意してください。実際には XMLType インスタンスにテキストしか含まれない場合でも、たとえば、EXTRACT(poDoc, '/PO/PNAME/text()') は XMLType インスタンスを戻します。getStringVal() を使用すると、VARCHAR2 の結果としてテキスト値を抽出できます。

getStringVal() または getNumberVal() を使用してテキスト・ノードを SQL データに変換する前に、text() ノード・テスト関数を使用して、要素内のテキスト・ノードを識別します。text() ノードを使用しない場合、XML フラグメントが生成されます。たとえば、XPath 式 /PO/PNAME は、フラグメント <PNAME>PO_1</PNAME> を識別します。一方、式 /PO/PNAME/text() は、テキスト値「PO_1」を識別します。

XML 文書内で要素が繰り返されている場合、索引作成メカニズムを使用して、個々の要素を識別できます。たとえば、次の XML 文書を考えてみます。

```
<PO>
  <PONO>100</PONO>
  <PONO>200</PONO>
</PO>
```

この場合、//PONO[1] を使用して文書内の最初の「PONO」要素（値は 100）を識別し、//PONO[2] を使用して 2 番目の「PONO」要素を識別できます。

抽出の結果は常に XMLType です。XPath を適用することによって空のセットが生成される場合、Extract は NULL 値を戻します。

したがって、extract() メンバー関数は、次のような多くの場合に使用できます。

- 数値を抽出し、その数値にファンクション索引を作成することによる処理の高速化
- SQL 文の FROM 句で使用するコレクション式の抽出
- 後で集計して別の文書を生成するためのフラグメントの抽出

XMLType の問合せ例 2: extract() および existsNode() の使用

発注書 ID および発注書 XML 列を含む po_xml_tab 表があり、この表に次の値を挿入すると想定します。

```
INSERT INTO po_xml_tab values (100,
   sys.xmltype.createxml('<?xml version="1.0"?>
                          <PO>
                            <PONO>221</PONO>
                            <PNAME>PO_2</PNAME>
                          </PO>'));

INSERT INTO po_xml_tab values (200,
   sys.xmltype.createxml('<?xml version="1.0"?>
                          <PO>
                            <PONAME>PO_1</PONAME>
                          </PO>'));
```

これで、EXTRACT 関数を使用して、発注書番号の数値を抽出できます。

```
SELECT e.poDoc.extract('//PONO/text()').getNumberVal() as pono
FROM po_xml_tab e
WHERE e.podoc.existsnode('/PO/PONO')  = 1 AND poid > 1;
```

ここで、extract() は、タグの内容である発注書番号「PONO」を抽出します。existsnode() は、「PO」の子として「PONO」が存在するノードを検出します。

text() 関数を使用すると、テキスト・ノードのみが戻されることに注意してください。getNumberVal() 関数は、テキスト値のみを数値に変換できます。

参照： 5-14 ページの「XMLType 関数」を参照してください。

XMLType の問合せ例 3: 一時 XMLtype データの問合せ

次の例は、XML データを選択して、それを PL/SQL 内で問い合せる方法を示します。

```
-- create a transient instance from the purchase order table and then perform
some extraction on it...
declare
   poxml SYS.XMLType;
   cust SYS.XMLType;
  val VARCHAR2;
begin

 -- select the adt instance
  select poDoc into poxml
     from po_xml_tab p where p.poid = 100;

  -- do some traversals and print the output
  cust := poxml.extract('//SHIPADDR');

   -- do something with the customer XML fragment
  val := cust.getStringVal();
  dbms_output.put_line(' The customer XML value is '|| val);

end;
/
```

XMLType の問合せ例 4: XML からのデータの抽出

次の例は、発注書 XML からデータを抽出し、そのデータを SQL リレーショナル表に挿入する方法を示します。

次のリレーショナル表を考えてみます。

```
CREATE TABLE cust_tab
(
  custid number primary key,
  custname varchar2(20)
);

insert into cust_tab values (1001, "John Nike");

CREATE table po_rel_tab
(
  pono number,
  pname varchar2(100),
  custid number refernces cust_tab
  shipstreet varchar2(100),
  shipcity varchar2(30),
  shipzip varchar2(20)
);
```

単純な PL/SQL ブロックを作成し、Extract 関数を使用して、次の形式の XML をリレーショナル表に変換できます。

```
<?xml version = '1.0'?>
<PO>
  <PONO>2001</PONO>
  <CUSTOMER CUSTNAME="John Nike"/>
  <SHIPADDR>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
  </SHIPADDR>
</PO>
```

次に SQL の例を示します（前述の XML が po_xml_tab 内にあると想定します）。

```
insert into po_rel_tab as
select p.poDoc.extract('/PO/PONO/text()').getnumberval(),
       p.poDoc.extract('/PO/PNAME/text()').getstringval(),
       -- get the customer id corresponding to the customer name
       ( SELECT custid
         FROM   cust_tab c
         WHERE  c.custname =
                p.poDoc.extract('/PO/CUSTOMER/@CUSTNAME').getstringval()
        ),
       p.poDoc.extract('/PO/SHIPADDR/STREET/text()').getstringval(),
       p.poDoc.extract('//CITY/text()').getstringval(),
       p.poDoc.extract('//ZIP/text()').getstringval(),

from po_xml_tab p;


The po_tab would now have the following values,

PONO    PNAME   CUSTID   SHIPSTREET         SHIPCITY   SHIPZIP
-----------------------------------------------------------------
2001            1001     323 College Drive  Edison     08820
```

入力 XML 文書で、PO の下に PNAME という要素がなかったため、PNAME が NULL になっていることに注意してください。また、//CITY を使用して、すべての階層で city 要素を検索していることにも注意してください。

PL/SQL ブロック内で等式を使用しても、同じタスクを実行できます。

```
declare
  poxml SYS.XMLType;
  cname varchar2(200);
  pono number;
  pname varchar2(100);
  shipstreet varchar2(100);
  shipcity varchar2(30);
  shipzip varchar2(20);

begin

 -- select the adt instance
  select poDoc into poxml from po_xml_tab p;

  cname := poxml.extract('//CUSTOMER/@CUSTNAME').getstringval();

  pono := poxml.extract('/PO/PONO/text()').getnumberval(),
  pname := poxml.extract('/PO/PNAME/text()').getstringval(),
  shipstreet := poxml.extract('/PO/SHIPADDR/STREET/text()').getstringval(),
  shipcity := poxml.extract('//CITY/text()').getstringval(),
  shipzip := poxml.extract('//ZIP/text()').getstringval(),

  insert into po_rel_tab
    values (pono, pname,
           (select custid from cust_tab c where custname = cname),
            shipstreet, shipcity, shipzip);
end;
/
```

XMLType の問合せ例 5: extract() を使用した検索

Extract 関数および ExistsNode 関数を使用すると、次に示すとおり、列に様々な操作を実行できます。

```
select e.poDoc.extract('/PO/PNAME/text()').getStringVal() PNAME
from po_xml_tab e
where e.poDoc.existsNode('/PO/SHIPADDR') = 1 and
      e.poDoc.extract('//PONO/text()').getNumberVal() = 300 and
      e.poDoc.extract('//@CUSTNAME').getStringVal() like '%John%';
```

この SQL 文は、出荷先を含み、発注書番号が 300 で、顧客名「CUSTNAME」に「John」という文字列が含まれているすべての文書の発注書要素 PO から、発注書名「PNAME」を抽出します。

XMLType の問合せ例 6: XMLType からのフラグメントの抽出

extract() メンバー関数は、XPath 式によって識別されるノードを抽出し、フラグメントを含む XMLType を戻します。この場合、検索結果は、ノードの集合、単一のノードまたはテキスト値のいずれかになります。XMLType に isFragment() 関数を使用すると、結果がフラグメントかどうかを確認できます。次に例を示します。

```
select e.po.extract('/PO/SHIPADDR/STATE').isFragment()
    from foo_tab e;
```

注意： フラグメントを XMLType 列に挿入することはできません。SYS_XMLGEN 関数を使用すると、囲みタグを追加することによってフラグメントを整形式の文書に変換できます。5-64 ページの「SYS_XMLGEN」を参照してください。ただし、様々な XMLType 関数を使用して、フラグメントに追加の問合せを実行することができます。

/PO/SHIPADDR/STATE を抽出すると、フラグメントではない単一の整形式ノードが戻るため、前述の SQL 文は、0（ゼロ）を戻します。

一方、/PO/SHIPADDR/STATE/text() などの XPath は、整形式の XML 文書ではないためにフラグメントとみなされます。

Text 演算子を使用した XMLType データの問合せ

Oracle Text 索引は CLOB 列および VARCHAR 列で機能します。Oracle では、Oracle Text 索引は XMLType 列でも機能するように拡張されています。デフォルトでは、Oracle Text 索引が XMLType 列に定義されると、自動的に XML セクションが作成されます。また、XPath をサポートするように拡張されている CONTAINS 演算子も提供します。

XMLType 列での Text 索引の作成

Text 索引は、他の CLOB 列または VARCHAR 列と同様に、INDEXTYPE を指定した CREATE INDEX を使用することによって作成できます。ただし、XMLType は仮想列として実装されるため、Text 索引はファンクション索引メカニズムを使用して作成されます。

この場合、問合せで Text 索引を作成および使用するには、索引を作成する権限および Text 索引を作成するために必要な権限の他に、ファンクション索引を作成するために必要な権限を取得し、必要な設定を行う必要があります。要件は次のとおりです。

- QUERY_REWRITE 権限 : 自スキーマ内の XMLType 列に Text 索引を作成するための権限を取得している必要があります。他のスキーマや他のスキーマに常駐している表の XMLType 列に Text 索引を作成する必要がある場合は、GLOBAL_QUERY_REWRITE 権限を取得している必要があります。
- QUERY_REWRITE_ENABLED パラメータを true に設定する必要があります。

- 問合せを再作成して Text 索引を使用するために、QUERY_REWRITE_INTEGRITY を trusted に設定する必要があります。

  参照： 次の章およびマニュアルを参照してください。

  - 第 8 章「Oracle Text を使用した XML データの検索」
  - 『Oracle Text リファレンス』
  - 『Oracle Text アプリケーション開発者ガイド』

CONTAINS と ExistsNode/Extract の違い

CONTAINS 内の XPath サポートと、`ExistsNode` および `Extract` 関数を介してサポートされる XPath の間には、次の相違点があります。

- 今回のリリースでは、Oracle Text 索引によってサポートされている XPath は、特定の等価述語も満たすため、ファンクション実装より強力です。
- Oracle Text 索引は空白を無視するため、空白が重要な場合、XPath 式で正確な結果を得られない場合があります。
- Oracle Text 索引は文字列の等価性を含む特定の述語式もサポートしていますが、数値および範囲の比較はできません。
- XML 文書にテキストのないタグ名および属性名のみが含まれている場合、Oracle Text 索引は不正な結果を戻す場合があります。このような文書の例を次に示します。

```
<A>
  <B>
      <C>
      </C>
  </B>
  <D>
      <E>
      </E>
   </D>
 </A>
```

  この場合、XPath 式の A/B/E は、この XML 文書と一致しますが、これは不正です。

- ファンクション索引および Oracle Text 索引は、両方ともナビゲーションをサポートします。そのため、Text 索引を 1 次フィルタとして使用して、基準と一致する可能性のあるすべての文書を効率的にフィルタし、その後に、残りの文書に `existsNode()` や `extract()` 操作などの 2 次フィルタを適用することができます。

XMLType 列の索引付け

XMLType を使用して次の索引を作成すると、問合せの評価を高速化できます。

- **XMLType を使用したファンクション索引 :** XML 文書の ExistsNode 関数や XML 文書の抽出された部分にファンクション索引を作成することによって、問合せを高速化できます。

  Extract 操作でのファンクション索引の例

  たとえば、次の問合せで検索を高速化するとします。

  ```
  SELECT * FROM po_xml_tab e
  WHERE e.poDoc.extract('//PONO/text()').getNumberVal()= 100;
  ```

  この場合、次のとおり、Extract 関数でファンクション索引を作成できます。

  ```
  CREATE INDEX city_index ON po_xml_tab
     (poDoc.extract('//PONO/text()').getNumberVal());
  ```

  この索引では、SQL 問合せは、行ごとに XML 文書を解析して XPath 式を評価するかわりに、ファンクション索引を使用して述語を評価します。

  ExistsNode 操作でのファンクション索引の例

  ビットマップ化されたファンクション索引を作成して、演算子の評価を高速化することもできます。特に ExistsNode は、XPath で示されたノードが文書内に存在するかどうかに応じて数値 1 または 0（ゼロ）を戻すため、最適です。

  たとえば、XML 文書のいずれかのレベルに SHIPADDR という要素が含まれているかどうか検索する、次の問合せを高速化するとします。

  ```
  SELECT * FROM po_xml_tab e
  WHERE e.poDoc.existsNode('//SHIPADDR') = 1;
  ```

  この場合、次のとおり、ExistsNode 関数でビットマップ化されたファンクション索引を作成します。

  ```
  CREATE INDEX po_index ON po_xml_tab
     (poDoc.existsNode('//SHIPADDR'));
  ```

  これによって、問合せ処理が高速化されます。

- **XMLType 列での Text 索引の作成 :** 前述のとおり、XMLType 列に Text 索引を作成できます。XMLType 列に索引付けをする場合、索引は、デフォルトのセクション・グループとして PATH_SECTION_GROUP を使用します。これはデフォルトであり、索引の作成時に上書きできます。

```
CREATE INDEX po_text_index ON
    po_xml_tab(poDoc) indextype is ctxsys.context;
```

この XMLType 列に、CONTAINS や SCORE などの Text 操作を実行できます。Oracle では、CONTAINS 関数は、新しい演算子 INPATH および HASPATH を使用する XPath をサポートするように拡張されています。

INPATH は、指定のパス内に任意のワードがあるかどうかを確認し、HASPATH は文書内に任意の XPath があるかどうかを確認します。

```
SELECT * FROM po_xml_doc  w
WHERE CONTAINS(w.poDoc,
              'haspath(/PO[./@CUSTNAME="John Nike"])') > 0;
```

XMLType（oracle.xdb.XMLType）への Java アクセス

XMLType には、Java の oracle.xdb.XMLType クラスを介してアクセスできます。このクラスには、PL/SQL の XMLType と同様の関数があります。oracle.xdb.XMLType は、Oracle JDBC の oracle.sql.OPAQUE クラスのサブクラスです。

サーバー内の XMLType は OPAQUE 型として実装されるため、JDBC で getOPAQUE コールを使用して JDBC の結果セットから OPAQUE 型のインスタンスを取得し、そのインスタンスから XMLType インスタンスを作成する必要があります。将来のリリースでは、JDBC は自動的に XMLType をインスタンス化するようになります。

XMLType は、java.sql.PreparedStatement インタフェースの setObject コールを使用して、JDBC の任意の XML データ・インスタンスにバインドできます。

oracle.xdb.XMLType クラスで定義される関数は、Java で XML データを取り出す場合に有効です。SQL 関数は、すべての問合せの実行に使用します。

XMLType の Java の例 1: Java での XMLType データの選択

Java で XMLType を選択する場合、次の 2 つの方法を使用できます。

- SQL で getClobVal() または getStringVal() を使用して、Java で oracle.sql.CLOB または java.lang.String として結果を取得します。次に、この操作を行う方法を示す Java コードの一部を示します。

```
 DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

 Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");

 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
          "select e.poDoc.getClobVal() poDoc, "+
                 e.poDoc.getStringVal() poString "+
          " from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;

// the first argument is a CLOB
oracle.sql.CLOB clb = orset.getCLOB(1);

// the second argument is a string..
String poString = orset.getString(2);

// now use the CLOB inside the program..
```

- PreparedStatement で getOPAQUE() コールを使用して XMLType インスタンス全体を取得し、XMLType コンストラクタを使用して、そのインスタンスから oracle.xdb.XMLType クラスを作成します。これによって、XMLType クラスで Java 関数を使用して、データにアクセスできます。

```
import oracle.xdb.XMLType;
...

 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
          "select e.poDoc from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;
```

```
// get the XMLType
XMLType poxml = XMLType(orset.getOPAQUE(1));

// get the XML as a string...
String poString = poxml.getStringVal();
```

XMLType の Java の例 2: Java での XMLType データの更新

Java で XMLType を挿入する場合、次の 2 つの方法を使用できます。

- CLOB または文字列を INSERT、UPDATE または DELETE 文にバインドし、SQL 内で createXML() コンストラクタを使用して XML インスタンスを作成します。

```
 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = sys.XMLType.createXML(?) ");

// the second argument is a string..
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";

// now bind the string..
stmt.setString(1,poString);
stmt.execute();
```

- PreparedStatement で setObject()（または setOPAQUE()）を使用して、XMLType インスタンス全体を設定します。

```
import oracle.xdb.XMLType;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = ? ");

// the second argument is a string..
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
XMLType poXML = XMLType.createXML(conn, poString);

// now bind the string..
stmt.setObject(1,poXML);
stmt.execute();
```

XMLType の Java の例 3: XMLType でのメタデータの取得

XMLType 値を選択する場合、JDBC が列を OPAQUE 型として記述します。列型の名前を選択し、それを「XMLTYPE」と比較すると、XMLType を処理しているかどうかを確認できます。

```
import oracle.sql.*;
import oracle.jdbc.*;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "select poDoc from po_xml_tab");

OracleResultSet rset = (OracleResultSet)stmt.exuecuteQuery();

// Now, we can get the resultset metadata
OracleResultSetMetaData mdata =
        (OracleResultSetMetaData)rset.getMetaData();

// Describe the column = the column type comes out as OPAQUE
// and column type name comes out as SYS.XMLTYPE
if (mdata.getColumnType(1) == OracleTypes.OPAQUE &&
    mdata.getColumnTypeName(1).oompareTo("SYS.XMLTYPE") == 0)
{
   // we know it is an XMLtype..
}
```

表 5-4 oracle.xdb.XMLType のメンバー関数および静的関数の概要

関数	構文の概要	説明
XMLType() (コンストラクタ)	PUBLIC oracle.xdb. XMLType(oracle.sql.OPAQUE opt)	OPAQUE インスタンスから oracle.xdb.XMLType インスタンスを作成するためのコンストラクタ。現在、JDBC は XMLType データを oracle.sql.OPAQUE インスタンスとして戻します。このコンストラクタを使用して、OPAQUE インスタンスから XMLType を作成します。 パラメータ : opq (IN) - 有効な OPAQUE インスタンス。
createXML()	PUBLIC STATIC oracle.xdb.XMLType createXML(Connection conn, String xmlval)	文字列から oracle.xdb.XMLType インスタンスを作成するための静的関数。XML 値が整形式かどうかは確認しません。XML 値に対するすべてのデータベース操作では、整形式かどうかが確認されます。 パラメータ : conn (IN) - 有効な Oracle Connection。 xmlval (IN) - XML 値を含む Java 文字列。 戻り値 : oracle.xdb.XMLType インスタンス。
createXML()	PUBLIC STATIC oracle.xdb.XMLType createXML(Connection conn, oracle.sql.clob xmlVal)	oracle.sql.CLOB から XMLType インスタンスを作成するための静的関数。XML 値が整形式かどうかは確認しません。すべてのデータベース操作では、整形式かどうかが確認されます。 パラメータ : conn (IN) - 有効な Oracle Connection。 xmlval (IN) - XML 値を含む Java 文字列。 戻り値 : oracle.xdb.XMLType インスタンス。
getClobVal()	PUBLIC oracle.sql.CLOB getClobVal()	文書を oracle.sql.CLOB として取得します。 戻り値 : シリアル化 XML 表現を含む CLOB。使用後はテンポラリ CLOB を解放してください。
getStringVal()	PUBLIC java.lang.String getStringVal()	XML 値を文字列として取得します。 戻り値 : シリアル化 XML 表現を含む文字列、またはテキスト・ノードの場合はそのテキスト。

XMLType の Java の例 4: XMLType 列の要素の更新

次の例では、XMLType 列に格納されている PO 内の「DISCOUNT」要素を更新します。この場合、Java（JDBC）および oracle.xdb.XMLType クラスを使用します。この例では、Java（JDBC）を使用して XMLType を挿入、更新および削除する方法も示します。パーサーを使用してメモリー内 DOM ツリーを更新し、更新された XML 値を列に書き込みます。

```
-- create po_xml_hist table to store old PurchaseOrders
create table po_xml_hist (
 xpo sys.xmltype
);

/*
   DESCRIPTION
    Example for oracle.xdb.XMLType

   NOTES
   Have classes12.zip, xmlparserv2.jar, and oraxdb.jar in CLASSPATH

*/

import java.sql.*;
import java.io.*;

import oracle.xml.parser.v2.*;
import org.xml.sax.*;
import org.w3c.dom.*;

import oracle.jdbc.driver.*;
import oracle.sql.*;

import oracle.xdb.XMLType;

public class tkxmtpje
{

  static String conStr = "jdbc:oracle:oci8:@";
  static String user = "scott";
  static String pass = "tiger";
  static String qryStr =
        "SELECT x.poDoc from po_xml_tab x "+
        "WHERE  x.poDoc.extract('/PO/PONO/text()').getNumberVal()=200";
```

```
static String updateXML(String xmlTypeStr)
 {
    System.out.println("\n==============================");
    System.out.println("xmlType.getStringVal():");
    System.out.println(xmlTypeStr);
    System.out.println("==============================");
    String outXML = null;
    try{
       DOMParser parser  = new DOMParser();
       parser.setValidationMode(false);
       parser.setPreserveWhitespace (true);

       parser.parse(new StringReader(xmlTypeStr));
       System.out.println("xmlType.getStringVal(): xml String is well-formed");

       XMLDocument doc = parser.getDocument();

       NodeList nl = doc.getElementsByTagName("DISCOUNT");

       for(int i=0;i<nl.getLength();i++){
          XMLElement discount = (XMLElement)nl.item(i);
          XMLNode textNode = (XMLNode)discount.getFirstChild();
          textNode.setNodeValue("10");
       }

      StringWriter sw = new StringWriter();
      doc.print(new PrintWriter(sw));

      outXML = sw.toString();

     //print modified xml
      System.out.println("\n==============================");
      System.out.println("Updated PurchaseOrder:");
      System.out.println(outXML);
      System.out.println("==============================");
     }
   catch ( Exception e )
   {
     e.printStackTrace(System.out);
   }
   return outXML;
 }
```

```
public static void main(String args[]) throws Exception
 {
   try{

       System.out.println("qryStr="+ qryStr);

       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

       Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:@", user, pass);

       Statement s = conn.createStatement();
       OraclePreparedStatement stmt;

       ResultSet rset = s.executeQuery(qryStr);
       OracleResultSet orset = (OracleResultSet) rset;

       while(orset.next()){

       //retrieve PurchaseOrder xml document from database
        XMLType xt = XMLType.createXML(orset.getOPAQUE(1));

        //store this PurchaseOrder in po_xml_hist table
        stmt = (OraclePreparedStatement)conn.prepareStatement(
                  "insert into po_xml_hist values(?)");

        stmt.setObject(1,xt);  // bind the XMLType instance
        stmt.execute();

        //update "DISCOUNT" element
        String newXML = updateXML(xt.getStringVal());

        // create a new instance of an XMLtype from the updated value
        xt = XMLType.createXML(conn,newXML);

       // update PurchaseOrder xml document in database
        stmt = (OraclePreparedStatement)conn.prepareStatement(
          "update po_xml_tab x set x.poDoc =? where "+
            "x.poDoc.extract('/PO/PONO/text()').getNumberVal()=200");

        stmt.setObject(1,xt);  // bind the XMLType instance
        stmt.execute();

        conn.commit();
        System.out.println("PurchaseOrder 200 Updated!");

       }
```

```
       //delete PurchaseOrder 1001
        s.execute("delete from po_xml x "+
           "where x.xpo.extract"+
              "('/PurchaseOrder/PONO/text()').getNumberVal()=1001");
        System.out.println("PurchaseOrder 1001 deleted!");
    }
    catch( Exception e )
    {
      e.printStackTrace(System.out);
    }
  }
}

----------------------
-- list PurchaseOrders
----------------------

set long 20000
set pages 100
select x.xpo.getClobVal()
from po_xml x;
```

XML での更新された PO の結果は次のとおりです。

```
<?xml version = '1.0'?>
<PurchaseOrder>
  <PONO>200</PONO>
  <CUSTOMER>
   <CUSTNO>2</CUSTNO>
   <CUSTNAME>John Nike</CUSTNAME>
   <ADDRESS>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
   </ADDRESS>
   <PHONELIST>
    <VARCHAR2>609-555-1212</VARCHAR2>
    <VARCHAR2>201-555-1212</VARCHAR2>
   </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
```

```
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
   <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1004">
     <PRICE>6750</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>1</QUANTITY>
    <DISCOUNT>10</DISCOUNT>
   </LINEITEM_TYP>
   <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1011">
     <PRICE>4500.23</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>2</QUANTITY>
    <DISCOUNT>10</DISCOUNT>
   </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR>
   <STREET>55 Madison Ave</STREET>
   <CITY>Madison</CITY>
   <STATE>WI</STATE>
   <ZIP>53715</ZIP>
  </SHIPTOADDR>
</PurchaseOrder>
```

oracle.xdb.XMLType クラスのインストールおよび使用

oracle.xdb.XMLType は、*ORACLE_HOME*/rdbms/jlib の **xdb_g**.jar ファイルにあります。ここで、*ORACLE_HOME* は、Oracle のホーム・ディレクトリを示します。

JVM 内での oracle.xdb.XMLType の使用

このクラスは JVM に事前ロードされており、SYS スキーマで使用可能です。

ただし、データベースを前のバージョンからアップグレードした場合はロードされていません。このクラスを JVM にアップロードする必要がある場合は、SYS として接続して、*ORACLE_HOME*/rdbms/admin ディレクトリの initxdbj.sql ファイルを実行する必要があります。

クライアント側での oracle.xdb.XMLType の使用

クライアント側で oracle.xdb.XMLType クラスを使用する必要がある場合、環境変数 CLASSPATH に xdb_g.jar ファイルがリストされていることを確認してください。

ネイティブ XML の生成

Oracle では、次のパッケージおよびファンクションを使用したネイティブ XML の生成をサポートしています。

- **DBMS_XMLGEN:** 提供される PL/SQL パッケージです。SQL 問合せから XML を取得します。これは、C で作成され、パフォーマンス向上のためにサーバーにリンクされています。
- SQL 問合せから XML を取得するための SQL 関数は次のとおりです。
  - **SYS_XMLGEN:** 行を操作し、XML 文書を生成します。
  - **SYS_XMLAGG:** 行のグループを操作し、複数の XML 文書を 1 つに集約します。

DBMS_XMLGEN

DBMS_XMLGEN は、データベース問合せの結果を XML にマップすることによって、SQL 問合せから XML 文書を作成します。DBMS_XMLGEN は、XML 文書を CLOB として取得します。DBMS_XMLGEN には「フェッチ」・インタフェースがあり、それによって行の最大数とスキップする行を指定できます。これは、Web アプリケーションでのページ区切り要件を満たすために有効です。DBMS_XMLGEN には、ROW、ROWSET などのタグ名を変更するオプションもあります。

DBMS_XMLGEN パッケージのパラメータは、取り出す行の数を制限し、タグ名を囲みます。PL/SQL パッケージ DBMS_XMLGEN の機能の概要は次のとおりです。

- SQL 問合せから XML 文書インスタンスを作成し、その文書を CLOB として取得します。
- 行の最大数とスキップする行を指定する「フェッチ」・インタフェースを提供します。たとえば、最初のフェッチで最大 10 行を取り出し、最初の 4 行をスキップできます。これは、Web ベース・アプリケーションでのページ区切り要件を満たすために有効です。
- ROW、ROWSET などのタグ名を変更するためのオプションを提供します。

参照： OracleXMLQuery と DBMS_XMLGEN の機能の比較については、7-21 ページの「XSU の OracleXMLQuery を使用した XML の生成」を参照してください。

問合せ結果の例

次に、データベースで「select * from scott.emp」問合せを実行した結果の例を示します。

```
<?xml version="1.0"?>
<ROWSET>
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Scott</ENAME>
  <SALARY>20000</SALARY>
</ROW>
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Mary</ENAME>
  <AGE>40</AGE>
</ROW>
</ROWSET>
```

DBMS_XMLGEN パッケージを使用した getXML() の結果は、CLOB になります。デフォルト・マッピングは次のとおりです。

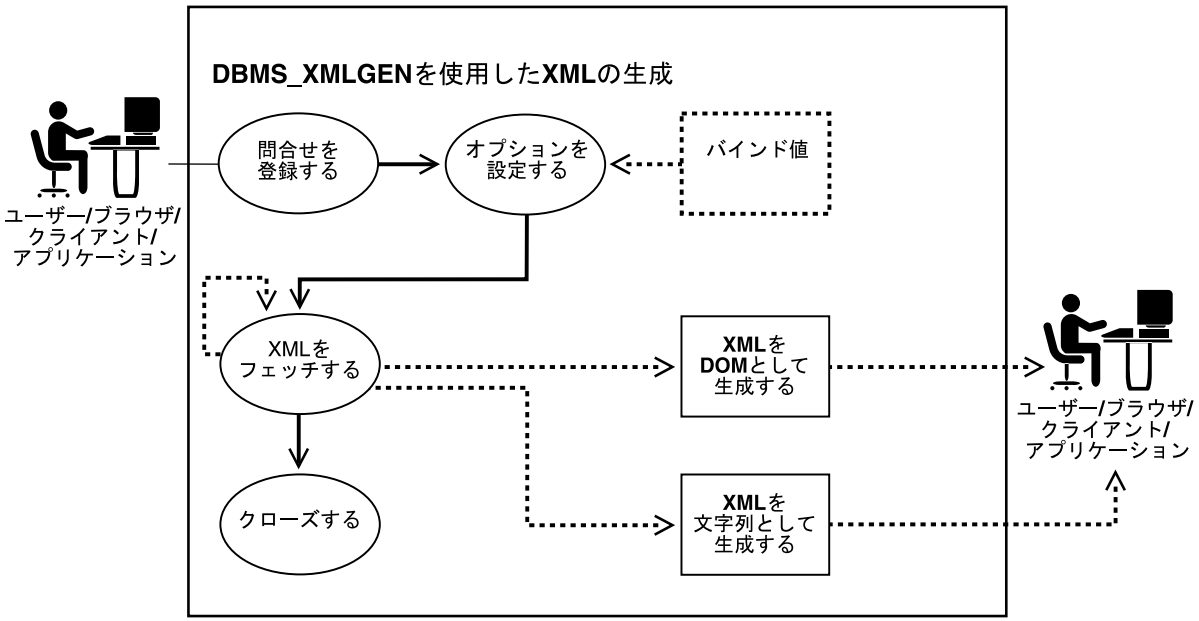
- 問合せ結果のすべての行は、デフォルトのタグ名「ROW」を持つ XML 要素にマップされます。
- 結果全体は、「ROWSET」要素で囲まれます。これらの名前は両方とも、DBMS_XMLGEN の setRowTagName() プロシージャおよび setRowSetTagName() プロシージャを使用して変更できます。
- SQL 問合せ結果の各列は、ROW 要素のサブ要素としてマップされます。
- CURSOR 式以外のすべてのデータ型が、DBMS_XMLGEN によってサポートされます。バイナリ・データは、16 進数表現に変換されます。

文書は CLOB にあるため、データベース・キャラクタ・セットと同じエンコーディングになります。データベース・キャラクタ・セットが SHIFTJIS の場合、XML 文書も SHIFTJIS になります。

DBMS_XMLGEN のコール順序

図 5-2 に、DBMS_XMLGEN のコール順序の概要を示します。

図 5-2 DBMS_XMLGEN のコール順序



DBMS_XMLGEN のコール順序は次のとおりです。

1. SQL 問合せを実行し、newContext() をコールすることによって、パッケージからコンテキストを取得します。
2. そのコンテキストをパッケージ内のすべてのプロシージャおよびファンクションに渡して、様々なオプションを設定します。たとえば、ROW 要素の名前を設定するには、setRowTag(ctx) を使用します。ここで、ctx は前の newContext() のコールから取得したコンテキストです。
3. getXML() を使用して XML 結果を取得します。setMaxRows() をコールしてフェッチごとに取り出す行の最大数を設定すると、このファンクションを繰り返しコールして、コールごとに最大数の行を取得できます。問合せに行が残っていない場合、このファンクションは NULL の CLOB を戻します。

   getXML() は、取り出す行がない場合でも、常に XML 文書を戻します。取り出す行があるかどうかを確認するには、getNumRowsProcessed() ファンクションを使用します。

4. 問合せをリセットして再度開始し、手順 3 を繰り返すことができます。
5. closeContext() をクローズし、内部に割り当てられているすべてのリソースを解放します。

表 5-5 に、DBMS_XMLGEN のファンクションおよびプロシージャの概要を示します。

表 5-5 DBMS_XMLGEN のファンクションおよびプロシージャ

ファクションまたはプロシージャ	説明
DBMS_XMLGEN 型定義 SUBTYPE ctxHandle IS NUMBER	すべてのファンクションによって使用されるコンテキスト・ハンドルです。 DTD または Schema の指定は次のとおりです。 ■ NONE CONSTANT NUMBER:= 0; 今回のリリースでサポートされています。 ■ DTD CONSTANT NUMBER:= 1; S ■ SCHEMA CONSTANT NUMBER:= 2; これらの指定を getXML ファンクションで使用して、DTD を指定するか、XML Schema を生成するか、または何も生成しないか（NONE）を指定できます。今回のリリースの getXML ファンクションでは、NONE 指定のみがサポートされています。
ファンクション・プロトタイプ newContext()	任意の問合せ文字列に対して、後続のファンクションで使用される新しいコンテキスト・ハンドルを生成します。
ファンクション newContext(queryString IN VARCHAR2)	新しいコンテキストを戻します。 パラメータ : queryString (IN) - XML に変換する必要がある問合せ文字列の結果。 戻り値 : コンテキスト・ハンドル。最初にこのファンクションをコールして、結果から XML を戻すために getXML() およびその他のファンクションで使用できるハンドルを取得します。
setRowTag()	すべての行を区切る要素の名前を設定します。デフォルト名は ROW です。

表 5-5 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファクションまたはプロシージャ	説明

プロシージャ
`setRowTag(ctx IN ctxHandle,`
`rowTag IN VARCHAR2);`
	パラメータ : ctx (IN) - newContext をコールして取得されるコンテキスト・ハンドル。
	rowTag (IN) - ROW 要素の名前。ROW 要素が必要ない場合、NULL を指定します。このファンクションをコールして、ROW 要素の名前を設定すると、デフォルト名の「ROW」が表示されません。この名前を NULL に設定して、ROW 要素自体を出力しないようにすることもできます。ROW および ROWSET の両方が NULL で、出力に複数の列または行がある場合、エラーになります。

`setRowSetTag()`	文書のルート要素の名前を設定します。デフォルト名は「ROWSET」です。

プロシージャ
`setRowSetTag(ctx IN ctxHandle,`
`rowSetTag IN VARCHAR2);`
	パラメータ : ctx (IN) - newContext をコールして取得されるコンテキスト・ハンドル。
	rowsetTag (IN) - 文書要素の名前。ROW 要素が必要ない場合、NULL を指定します。このファンクションをコールして、文書のルート要素の名前を設定すると、出力にデフォルト名の「ROWSET」が表示されません。この名前を NULL に設定して、この要素自体を出力しないようにすることもできます。ROW および ROWSET の両方が NULL で、出力に複数の列または行がある場合、エラーになります。

`getXML()`	指定されている最大数の行をフェッチすることによって XML 文書を取得します。また、その XML 文書を渡された CLOB に追加します。

プロシージャ
`getXML(ctx IN ctxHandle,`
`clobval IN OUT NCOPY clob,`
`dtdOrSchema IN number:= NONE);`
	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。
	clobval (IN/OUT) - XML 文書を追加する CLOB。
	dtdOrSchema (IN) - DTD または Schema のどちらを生成するかどうか。このパラメータはサポートされていません。
	このバージョンの getXML ファンクションを使用して、余分な CLOB コピーの生成を回避したり、後続のコールで同じ CLOB を再利用します。この getXML のコールは、LOB ロケータの作成を必要としますが、次のファンクションより効率的です。
	XML を生成するときは、setSkipRows コールに指定された数の行がスキップされ、setMaxRows コールに指定された最大数の行（または指定がない場合は結果全体）がフェッチされ、XML に変換されます。getNumRowsProcessed ファンクションを使用して、行が取り出されたかどうかを確認します。

表 5-5 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
`getXML()`	XML 文書を生成し、それを CLOB として戻します。
ファンクション `getXML(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN clob`	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。 dtdOrSchema (IN) - DTD または Schema のどちらかを生成するかどうか。このパラメータはサポートされていません。 戻り値 : 文書を含むテンポラリ CLOB。dbms_lob.freetemporary をコールして、このファンクションから取得したテンポラリ CLOB を解放します。
ファンクション `getXMLType(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN sys.XMLType`	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。 dtdOrSchema (IN) - DTD または Schema のどちらかを生成するかどうか。このパラメータはサポートされていません。 戻り値 : 文書を含む XMLType インスタンス。
`getNumRowsProcessed()`	getXML をコールして XML を生成するときに処理される、SQL の行数を取得します。この行数には、XML の生成前にスキップされる行数は含まれません。
ファンクション `getNumRowsProcessed(ctx IN ctxHandle) RETURN number`	パラメータ : queryString (IN) - XML に変換する必要がある結果の問合せ文字列。 戻り値 : getXML を最後にコールしたときに処理された行数。この行数にはスキップされた行数は含まれません。ループで getXML をコールする場合、このファンクションを使用して終了条件を決定します。getXML は、行がない場合でも常に XML 文書を生成することに注意してください。
`setMaxRows()`	getXML のコールごとに、SQL 問合せ結果からフェッチする行の最大数を設定します。
プロシージャ `setMaxRows(ctx IN ctxHandle, maxRows IN NUMBER);`	パラメータ : ctx (IN) - 実行されている問合せに対応するコンテキスト・ハンドル。 maxRows (IN) - getXML をコールするたびに取得する最大行数。 maxRows パラメータは、このユーティリティを使用して結果のページ区切りを生成するときに使用できます。たとえば、XML または HTML データのページを生成する場合、XML に変換する行数を制限し、後続のコールにおいて、次の一連の行を次々に取得することができます。これによって応答時間が短縮できます。

表 5-5 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
setSkipRows()	getXML ルーチンをコールするたびに、XML 出力を生成する前に任意の数の行をスキップします。
プロシージャ setSkipRows(ctx IN ctxHandle, skipRows IN NUMBER);	パラメータ : ctx (IN) - 実行されている問合せに対応するコンテキスト・ハンドル。 skipRows (IN) - getXML をコールするたびにスキップする行数。 skipRows パラメータは、このユーティリティを使用して、ステートレスな Web ページの結果の区切りページの結果を生成するときに使用できます。たとえば、XML または HTML データの最初のページを生成するときに、skipRows を 0（ゼロ）に設定できます。次の設定では、skipRows を最初に指定した行数に設定できます。
setConvertSpecialChars()	XML データの特殊文字を、XML の等価のエスケープ文字に変換する必要があるかどうかを設定します。たとえば、「<」符号は < に変換されます。デフォルトでは、変換が実行されます。
プロシージャ setConvertSpecialChars(ctx IN ctxHandle, conv IN boolean);	パラメータ : ctx (IN) - 使用するコンテキスト・ハンドル。 conv (IN) - 変換が必要かどうか。 入力データに、エスケープする必要がある <、>、"、' などの特殊文字が含まれていないことがわかっている場合、このファンクションを使用して XML 処理を高速化できます。実際に文字データをスキャンして特殊文字を置換するには、特に大量のデータを処理する場合、大きいコストがかかります。したがって、データが XML で問題ない場合、このファンクションをコールしてパフォーマンスを向上できます。
useItemTagsForColl()	コレクション要素の名前を設定します。コレクション要素のデフォルト名は、型名自体です。このファンクションを使用すると、列名を、列名に「_ITEM」タグを追加した名前に上書きできます。
プロシージャ useItemTagsForColl(ctx IN ctxHandle);	パラメータ : ctx (IN) - コンテキスト・ハンドル。 たとえば、NUMBER というコレクションがある場合、コレクション要素のデフォルト・タグ名は NUMBER になります。このプロシージャをコールすると、このデフォルト動作を変更して、デフォルト名に「_ITEM」タグを追加したコレクション列名を生成できます。
restartQuery()	問合せを再度開始し、最初の行から XML を生成します。

表 5-5 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファクションまたはプロシージャ	説明
プロシージャ `restartQuery(ctx IN ctxHandle);`	パラメータ : ctx (IN) - 現行の問合せに対応するコンテキスト・ハンドル。このプロシージャをコールすると、新しいコンテキストを作成することなく、問合せの実行を再度開始できます。
`closeContext()`	任意のコンテキストをクローズし、SQL カーソル、バインドや定義バッファなど、そのコンテキストに対応付けられているすべてのリソースを解放します。
プロシージャ `closeContext(ctx IN ctxHandle);`	パラメータ : ctx (IN) - クローズするコンテキスト・ハンドル。このハンドルに対応付けられているすべてのリソースをクローズします。クローズした後は、他のすべての DBMS_XMLGEN ファンクションをコールするときに、このハンドルを使用できなくなります。

DBMS_XMLGEN の例 1: 単純な XML の生成

この例では、オブジェクト・リレーショナル表から従業員データを選択して XML 文書を作成し、結果の CLOB を表に挿入します。

```
CREATE TABLE temp_clob_tab(result CLOB);

DECLARE
   qryCtx DBMS_XMLGEN.ctxHandle;
   result CLOB;
BEGIN
  qryCtx := dbms_xmlgen.newContext('SELECT * from scott.emp;');

  -- set the row header to be EMPLOYEE
  DBMS_XMLGEN.setRowTag(qryCtx, 'EMPLOYEE');

  -- now get the result
  result := DBMS_XMLGEN.getXML(qryCtx);

  INSERT INTO temp_clob_tab VALUES(result);
END;
/
```

この例から生成された XML は次のようになります。

```
select * from temp_clob_tab;

RESULT
-----------------------------------
<?xml version=''1.0''?>
<ROWSET>
 <EMPLOYEE>
  <EMPNO>7369</EMPNO>
  <ENAME>SMITH</ENAME>
  <JOB>CLERK</JOB>
  <MGR>7902</MGR>
  <HIREDATE>17-DEC-80</HIREDATE>
  <SAL>800</SAL>
  <DEPTNO>20</DEPTNO>
 </EMPLOYEE>
 <EMPLOYEE>
  <EMPNO>7499</EMPNO>
  <ENAME>ALLEN</ENAME>
  <JOB>SALESMAN</JOB>
  <MGR>7698</MGR>
  <HIREDATE>20-FEB-81</HIREDATE>
  <SAL>1600</SAL>
  <COMM>300</COMM>
  <DEPTNO>30</DEPTNO>
 </EMPLOYEE>
...
</ROWSET>
```

DBMS_XMLGEN の例 2: ページ区切りを使用した単純な XML の生成

すべての行に対して XML 全体を取得するかわりに、DBMS_XMLGEN が提供するフェッチ・インタフェースを使用して、1 回に固定数の行を取り出すことができます。これによって、特に多数の行がある場合に、応答時間が短縮され、結果の XML に DOM API を使用する必要があるアプリケーションをスケールするために有効です。

次の例は、DBMS_XMLGEN を使用して scott.emp 表から結果を取り出す方法を示します。

```
-- create a table to hold the results..!
create table temp_clob_tab ( result clob);

declare
   qryCtx dbms_xmlgen.ctxHandle;
   result CLOB;
begin
```

```
  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('select * from scott.emp');

  -- set the maximum number of rows to be 5,
  dbms_xmlgen.setMaxRows(qryCtx, 5);

  loop
    -- now get the result
    result := dbms_xmlgen.getXML(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- do some processing with the lob data..!
    -- Here, we are inserting the results
    -- into a table. You can print the lob out, output it to a stream,
    -- put it in a queure
    -- or do any other processing.
    insert into temp_clob_tab values(result);

  end loop;

end;
/
```

この場合、5 行ごとに XML 文書が生成されます。

DBMS_XMLGEN の例 3: 複雑な XML の生成

オブジェクト型を使用すると、ネストした構造を表す複雑な XML を生成できます。

```
 CREATE TABLE new_departments (
    department_id   NUMBER PRIMARY KEY,
    department_name VARCHAR2(20)
  );

 CREATE TABLE new_employees (
    employee_id    NUMBER PRIMARY KEY,
    last_name    VARCHAR2(20),
    department_id  NUMBER REFERENCES departments
  );

 CREATE TYPE emp_t AS OBJECT(
    "@employee_id" NUMBER,
     last_name VARCHAR2(20)
  );

 CREATE TYPE emplist_t AS TABLE OF emp_t;
```

```
 CREATE TYPE dept_t AS OBJECT(
    "@department_id" NUMBER,
     department_name VARCHAR2(20),
     emplist emplist_t
  );

 qryCtx := dbms_xmlgen.newContext
     ('SELECT dept_t(department_id, department_name,
              CAST(MULTISET
                   (SELECT e.employee_id, e.last_name
                    FROM   employees e
                    WHERE  e.department_id = d.department_id)
                            AS   emplist_t))  AS deptxml
         FROM departments d');
DBMS_XMLGEN.setRowTag(qryCtx, NULL);
```

生成された XML は次のようになります。

```
<ROWSET>
   <DEPTXML DEPARTMENT_ID="10">
      <DEPARTMENT_NAME>SALES</DEPARTMENT_NAME>
          <EMPLIST>
            <EMP_T EMPLOYEE_ID="30">
              <LAST_NAME>Scott</LAST_NAME>
            </EMP_T>
            <EMP_T EMPLOYEE_ID="31">
              <LAST_NAME>Mary</LAST_NAME>
            </EMP_T>
          </EMPLIST>
      </DEPTXML>
   <DEPTXML DEPARTMENT_ID="20">
   ...
</ROWSET>
```

これで、temp_clob_Tab 表から LOB データを選択し、結果を検証できます。結果は、5-44 ページの「問合せ結果の例」に示す結果の例に類似しています。

リレーショナル・データの場合、結果はフラットでネストしていない XML 文書になります。ネストした XML 構造にするには、オブジェクト・リレーショナル・データを使用します。この場合のマッピングは次のとおりです。

- オブジェクト型は、XML 要素としてマップされます。
- 型の属性は、親要素のサブ要素にマップされます。

注意： 複雑な構造にするには、オブジェクト型を使用し、オブジェクト・ビューまたはオブジェクト表を作成します。正規マッピングを使用して、オブジェクト・インスタンスを XML にマップしてください。

アットマーク（@）を列名または属性名で使用すると、マッピングで XML の囲み要素の属性に変換されます。

DBMS_XMLGEN の例 4: 複雑な XML の生成 - XML 文書内でネストするためのユーザー定義型の入力

ユーザー定義型の値を DBMS_XMLGEN 関数に入力すると、ユーザー定義型は正規マッピングを使用して XML 文書にマップされます。正規マッピングでは、ユーザー定義型の属性は XML 要素にマップされます。

「@」で始まる名前のすべての属性は、その前にある要素の属性にマップされます。

ユーザー定義型を使用して、結果の XML 文書内でネストすることができます。

たとえば、EMP および DEPT という 2 つの表を考えてみます。

```
CREATE TABLE DEPT
(
 deptno number primary key,
 dname varchar2(20)
);

CREATE TABLE EMP
(
  empno number primary key,
  ename varchar2(20),
  deptno number references dept
);
```

この場合、部門とその従業員のデータの階層ビューを生成するには、適切なオブジェクト型を定義してデータベース内に構造を作成できます。次に例を示します。

```
CREATE TYPE EMP_T AS OBJECT
(
  "@empno" number,  -- empno defined as an attribute!
   ename varchar2(20),
);
/
```

前にアットマーク（@）を付けた empno を定義して、Employee の囲み要素の属性としてマップする必要があることを示します。

```
CREATE TYPE EMPLIST_T AS TABLE OF EMP_T;
/
CREATE TYPE DEPT_T AS OBJECT
(
  "@deptno" number,
  dname varchar2(20),
  emplist emplist_t
);
    /
```

部門タイプの DEPT_T は、従業員のリストを含む部門を表します。これで、従業員表と部門表を問い合せ、結果を XML 文書として取得できます。次に例を示します。

```
declare
   qryCtx dbms_xmlgen.ctxHandle;
   result CLOB;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext(
     'SELECT
      dept_t(deptno,dname,
               CAST(MULTISET(select empno, ename
                    from emp e
                    where e.deptno = d.deptno) AS emplist_t))) AS deptxml
    FROM dept d');

  -- set the maximum number of rows to be 5,
  dbms_xmlgen.setMaxRows(qryCtx, 5);
```

```
  -- set no row tag for this result as we have a single ADT column
  dbms_xmlgen.setRowTag(qryCtx,null);

  loop
    -- now get the result
    result := dbms_xmlgen.getXML(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- do whatever with the result..!
  end loop;
end;
/
```

MULTISET 演算子は、その部門の従業員のサブセット結果を、リストとそれを囲む CAST として扱い、それを適切なコレクション型にキャストします。こうすることで、それを囲む部門インスタンスを作成し、DBMS_XMLGEN ルーチンをコールして、オブジェクト・インスタンスの XML を作成します。結果は次のようになります。

```
<?xml version="1.0"?>
<ROWSET>
 <DEPTXML deptno="10">
   <DNAME>Sports</DNAME>
   <EMPLIST>
    <EMP_T empno="200">
     <ENAME>John</ENAME>
    </EMP_T>
    <EMP_T empno="300">
     <ENAME>Jack</ENAME>
    </EMP_T>
   </EMPLIST>
 </DEPTXML>
   <DEPTXML deptno="20">
      <!-- .. other columns -->
   </DEPTXML>
 </ROWSET>
```

デフォルト名「ROW」は、NULL に設定しているために存在しません。deptno および empno は囲み要素の属性になっています。

DBMS_XMLGEN の例 5: XML 形式でのデータベースからの発注書の生成

この例では、DBMS_XMLGEN.getXMLType() を使用して、オブジェクト・ビューによってリレーショナル・データベースから PO を XML 形式で生成します。

```
-----------------------------------------------------
-- Create relational schema and define Object Views
-- Note: DBMS_XMLGEN Package maps UDT attributes names
--       starting with '@' to xml attributes
-----------------------------------------------------
-- Purchase Order Object View Model

-- PhoneList Varray object type
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20)
/

-- Address object type
CREATE TYPE Address_typ AS OBJECT (
  Street         VARCHAR2(200),
  City           VARCHAR2(200),
  State          CHAR(2),
  Zip            VARCHAR2(20)
  )
/

-- Customer object type
CREATE TYPE Customer_typ AS OBJECT (
  CustNo           NUMBER,
  CustName         VARCHAR2(200),
  Address          Address_typ,
  PhoneList        PhoneList_vartyp
)
/

-- StockItem object type
CREATE TYPE StockItem_typ AS OBJECT (
  "@StockNo"    NUMBER,
  Price      NUMBER,
  TaxRate    NUMBER
)
/

-- LineItems object type
CREATE TYPE LineItem_typ AS OBJECT (
  "@LineItemNo"   NUMBER,
```

```
  Item    StockItem_typ,
  Quantity     NUMBER,
  Discount     NUMBER
  )
/
-- LineItems Nested table
CREATE TYPE LineItems_ntabtyp AS TABLE OF LineItem_typ
/

-- Purchase Order object type
CREATE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO                 NUMBER,
  Cust_ref             REF Customer_typ,
  OrderDate            DATE,
  ShipDate             TIMESTAMP,
  LineItems_ntab       LineItems_ntabtyp,
  ShipToAddr           Address_typ
 )
/

-- Create Purchase Order Relational Model tables

--Customer table
CREATE TABLE Customer_tab(
  CustNo                NUMBER NOT NULL,
  CustName              VARCHAR2(200) ,
  Street                VARCHAR2(200) ,
  City                  VARCHAR2(200) ,
  State                 CHAR(2) ,
  Zip                   VARCHAR2(20) ,
  Phone1                VARCHAR2(20),
  Phone2                VARCHAR2(20),
  Phone3                VARCHAR2(20),
  constraint cust_pk PRIMARY KEY (CustNo)
)
ORGANIZATION INDEX OVERFLOW;

-- Purchase Order table
CREATE TABLE po_tab (
   PONo        NUMBER, /* purchase order no */
   Custno      NUMBER constraint po_cust_fk references Customer_tab,
                                /*  Foreign KEY referencing customer */
   OrderDate   DATE, /*  date of order */
   ShipDate    TIMESTAMP, /* date to be shipped */
   ToStreet    VARCHAR2(200), /* shipto address */
```

```
   ToCity      VARCHAR2(200),
   ToState     CHAR(2),
   ToZip       VARCHAR2(20),
   constraint po_pk PRIMARY KEY(PONo)
);

--Stock Table
CREATE TABLE Stock_tab (
  StockNo      NUMBER constraint stock_uk UNIQUE,
  Price        NUMBER,
  TaxRate      NUMBER
);

--Line Items Table
CREATE TABLE LineItems_tab(
  LineItemNo           NUMBER,
  PONo                 NUMBER constraint LI_PO_FK REFERENCES po_tab,
  StockNo              NUMBER ,
  Quantity             NUMBER,
  Discount             NUMBER,
  constraint LI_PK PRIMARY KEY (PONo, LineItemNo)
);

-- create Object Views

--Customer Object View
CREATE OR REPLACE VIEW Customer OF Customer_typ
   WITH OBJECT IDENTIFIER(CustNo)
   AS SELECT c.Custno, C.custname,
             Address_typ(C.Street, C.City, C.State, C.Zip),
             PhoneList_vartyp(Phone1, Phone2, Phone3)
        FROM Customer_tab c;

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
  WITH OBJECT IDENTIFIER (PONO)
   AS SELECT P.PONo,
             MAKE_REF(Customer, P.Custno),
             P.OrderDate,
             P.ShipDate,
             CAST( MULTISET(
                    SELECT LineItem_typ( L.LineItemNo,
                                StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                         L.Quantity, L.Discount)
                     FROM LineItems_tab L, Stock_tab S
```

```
                    WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
                AS LineItems_ntabtyp),
         Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
        FROM PO_tab P;

-- create table with XMLType column to store po in XML format
create table po_xml_tab(
  poid number,
  poDoc SYS.XMLType /* purchase order in XML format */
)
/

--------------------
-- Populate data
-------------------
-- Establish Inventory

INSERT INTO Stock_tab VALUES(1004, 6750.00, 2) ;
INSERT INTO Stock_tab VALUES(1011, 4500.23, 2) ;
INSERT INTO Stock_tab VALUES(1534, 2234.00, 2) ;
INSERT INTO Stock_tab VALUES(1535, 3456.23, 2) ;

-- Register Customers

INSERT INTO Customer_tab
  VALUES (1, 'Jean Nance', '2 Avocet Drive',
         'Redwood Shores', 'CA', '95054',
         '415-555-1212', NULL, NULL) ;

INSERT INTO Customer_tab
  VALUES (2, 'John Nike', '323 College Drive',
         'Edison', 'NJ', '08820',
         '609-555-1212', '201-555-1212', NULL) ;

-- Place Orders

INSERT INTO PO_tab
  VALUES (1001, 1, '10-APR-1997', '10-MAY-1997',
          NULL, NULL, NULL, NULL) ;

INSERT INTO PO_tab
  VALUES (2001, 2, '20-APR-1997', '20-MAY-1997',
         '55 Madison Ave', 'Madison', 'WI', '53715') ;
```

```
-- Detail Line Items

INSERT INTO LineItems_tab VALUES(01, 1001, 1534, 12,  0) ;
INSERT INTO LineItems_tab VALUES(02, 1001, 1535, 10, 10) ;
INSERT INTO LineItems_tab VALUES(01, 2001, 1004,  1,  0) ;
INSERT INTO LineItems_tab VALUES(02, 2001, 1011,  2,  1) ;


-------------------------------------------------------
-- Use DBMS_XMLGEN Package to generate PO in XML format
-- and store SYS.XMLType in po_xml table
-------------------------------------------------------

declare
   qryCtx dbms_xmlgen.ctxHandle;
   pxml SYS.XMLType;
   cxml clob;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('
                    select pono,deref(cust_ref) customer,p.OrderDate,p.shipdate,
                           lineitems_ntab lineitems,shiptoaddr
                    from po p'
            );

  -- set the maximum number of rows to be 1,
  dbms_xmlgen.setMaxRows(qryCtx, 1);
  -- set rowset tag to null and row tag to PurchaseOrder
  dbms_xmlgen.setRowSetTag(qryCtx,null);
  dbms_xmlgen.setRowTag(qryCtx,'PurchaseOrder');

  loop
    -- now get the po in xml format
    pxml := dbms_xmlgen.getXMLType(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- Store SYS.XMLType po in po_xml table (get the pono out)
    insert into po_xml_tab (poid, poDoc)
       values(
            pxml.extract('//PONO/text()').getNumberVal(),
            pxml);
  end loop;
end;
/
```

```
----------------------------
-- list xml PurchaseOrders
----------------------------

set long 100000
set pages 100
select x.xpo.getClobVal() xpo
from   po_xml x;

PurchaseOrder 1001:
```

これによって、次の発注書の XML 文書が生成されます。

```
<?xml version="1.0"?>
 <PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
   <CUSTNO>1</CUSTNO>
   <CUSTNAME>Jean Nance</CUSTNAME>
   <ADDRESS>
    <STREET>2 Avocet Drive</STREET>
    <CITY>Redwood Shores</CITY>
    <STATE>CA</STATE>
    <ZIP>95054</ZIP>
   </ADDRESS>
   <PHONELIST>
    <VARCHAR2>415-555-1212</VARCHAR2>
   </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>10-APR-97</ORDERDATE>
  <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
   <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1534">
     <PRICE>2234</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>12</QUANTITY>
    <DISCOUNT>0</DISCOUNT>
   </LINEITEM_TYP>
   <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1535">
     <PRICE>3456.23</PRICE>
```

```
      <TAXRATE>2</TAXRATE>
     </ITEM>
     <QUANTITY>10</QUANTITY>
     <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
   </LINEITEMS>
   <SHIPTOADDR/>
  </PurchaseOrder>

PurchaseOrder 2001:

<?xml version="1.0"?>
 <PurchaseOrder>
  <PONO>2001</PONO>
  <CUSTOMER>
   <CUSTNO>2</CUSTNO>
   <CUSTNAME>John Nike</CUSTNAME>
   <ADDRESS>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
   </ADDRESS>
   <PHONELIST>
    <VARCHAR2>609-555-1212</VARCHAR2>
    <VARCHAR2>201-555-1212</VARCHAR2>
   </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
   <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1004">
     <PRICE>6750</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>1</QUANTITY>
    <DISCOUNT>0</DISCOUNT>
   </LINEITEM_TYP>
   <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1011">
     <PRICE>4500.23</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
```

```
    <QUANTITY>2</QUANTITY>
    <DISCOUNT>1</DISCOUNT>
   </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR>
   <STREET>55 Madison Ave</STREET>
   <CITY>Madison</CITY>
   <STATE>WI</STATE>
   <ZIP>53715</ZIP>
  </SHIPTOADDR>
 </PurchaseOrder>
```

SYS_XMLGEN

Oracle では、SQL 問合せで XML を生成するための新しい SQL 関数 SYS_XMLGEN() が導入されています。DBMS_XMLGEN およびその他のパッケージは問合せレベルで機能し、問合せ全体の集計結果を戻します。SYS_XMLGEN は、SQL 問合せ内の単一の引数を取り、それ（結果）を XML に変換します。

SYS_XMLGEN は、XML 文書に変換するスカラー値、オブジェクト型または XMLType インスタンスを取ります。また、オプションの XMLGenFormatType オブジェクトも取ります。このオブジェクトを使用すると、結果の XML 文書に対する書式設定オプションを指定できます。

SYS_XMLGEN は XMLType を戻します。

SYS_XMLGEN を使用して、SQL 問合せ内で XML インスタンスを作成し、問い合せます。次に例を示します。

```
SQL> SELECT SYS_XMLGEN(employee_id)
            2  FROM employees WHERE last_name LIKE
             'Scott%';
```

結果の XML 文書は次のようになります。

```
<?xml version=''1.0''?>
<employee_id>60</employee_id>
```

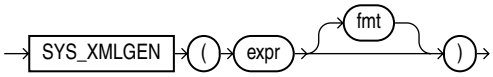
SYS_XMLGEN の構文

SYS_XMLGEN 関数は、データベースの特定の行および列に対して評価する式を使用して、XML 文書を含む XMLType 型のインスタンスを戻します。図 5-3 を参照してください。*expr* は、スカラー値、ユーザー定義型または XMLType インスタンスのいずれかになります。

- *expr* がスカラー値の場合、この関数は、スカラー値を含む XML 要素を戻します。
- *expr* が型の場合、この関数は、ユーザー定義型属性を XML 要素にマップします。
- *expr* が XMLType インスタンスの場合、この関数は、デフォルト・タグ名が ROW である XML 要素で文書を囲みます。

デフォルトでは、XML 文書の要素は *expr* の要素と一致します。たとえば、*expr* が列名に変換される場合、XML の囲み要素は同じ列名になります。XML 文書を別の方法でフォーマットする場合は *fmt* を指定します。これは、XMLGenFormatType オブジェクトのインスタンスです。

図 5-3 SYS_XMLGEN の構文



次の例では、サンプル表 oe.employees から、従業員（employee_id 値が 205）の電子メール ID を取り出し、EMAIL 要素を持つ XML 文書を含む XMLType のインスタンスを生成します。

```
SELECT SYS_XMLGEN(email).getStringVal()
   FROM employees
   WHERE employee_id = 205;

SYS_XMLGEN(EMAIL).GETSTRINGVAL()
--------------------------------------------------------------------
<EMAIL>SHIGGENS</EMAIL>
```

SYS_XMLGEN のメリット

SYS_XMLGEN() のメリットは次のとおりです。

- SQL 問合せ内で XML インスタンスを作成し、問い合せることができます。
- オブジェクト・リレーショナル・インフラストラクチャを使用して、単純なリレーショナル表から複雑なネストした XML インスタンスを作成できます。

SYS_XMLGEN() は、次のどちらかから XML 文書を作成します。

- ユーザー定義型インスタンス
- 渡されるスカラー値

また、文書に含まれる XMLType インスタンスを戻します。

また、SYS_XMLGEN() は、オプションで XMLGenFormatType オブジェクト型を入力します。このオブジェクト型を介して、SQL の結果をカスタマイズできます。書式設定オブジェクトが NULL の場合、デフォルトのマッピング動作が使用されます。

XMLGenFormatType オブジェクト

XMLGenFormatType を使用して、SYS_XMLGEN および SYS_XMLAGG 関数に対して書式設定引数を指定できます。表 5-6 に、XMLGenFormatType 属性を示します。

表 5-6 XMLGenFormatType 属性

XMLGenFormatType 属性	説明
enclTag	使用する囲みタグの名前です。
schemaType	現在はサポートされていません。ただし、将来のリリースでは、XML Schema を生成する方法を指定するために使用されます。
processingIns	文書の先頭に追加する必要があるスタイルシート命令などの処理命令を指定できます。
dburl	現在は使用されていません。
targetNameSpace	現在は使用されていません。ただし、将来のリリースでは、XML Schema の生成に使用されます。

createFormat を使用した書式設定オブジェクトの作成

静的メンバー関数 createFormat を使用して、書式設定オブジェクトを作成できます。この関数では、ほとんどの値がデフォルトで指定されています。次に例を示します。

```
-- function to create the formatting object..
 STATIC MEMBER FUNCTION createFormat(
      enclTag IN varchar2 := null,
       schemaType IN varchar2 := 'NO_SCHEMA'
       schemaName IN varchar2 := null,
       targetNameSpace IN varchar2 := null,
      dburl IN varchar2 := null,
       processingIns IN varchar2 := null)
  RETURN XMLGenFormatType;
```

SYS_XMLGEN の例 1: スカラー値から XML 文書要素のコンテンツへの変換

SYS_XMLGEN() にスカラー値を入力すると、SYS_XMLGEN() はそのスカラー値を、スカラー値を含む要素に変換します。次に例を示します。

```
select sys_xmlgen(empno) from scott.emp where rownum < 1;
```

次に示すとおり、SYS_XMLGEN() は要素として empno 値を含む XML 文書を戻します。

```
<?xml version="1.0"?>
<EMPNO>30</EMPNO>
```

囲み要素名（この場合は EMPNO）は、演算子に渡される列名から導出されます。SELECT 文の結果は、XMLType を含む行になることに注意してください。

注意： 現在、SQL*Plus は XMLType を適切に表示できません。そのため、LOB 値を抽出して、それを表示する必要があります。XMLType に対して getClobval() 関数を使用し、CLOB 値を取り出します。次に例を示します。

```
SELECT sys_xmlgen(empno).getclobval() FROM scott.emp
WHERE rownum < 1;
```

前述の例では、文書に列名「EMPNO」を使用しています。列名を直接導出できない場合、デフォルト名「ROW」が使用されます。次の構文を考えてみます。

```
select sys_xmlgen(empno).getclobval()
from scott.emp
where rownum < 1;
```

次の XML 出力が生成されます。

```
<?xml version="1.0"?>
<ROW>60</ROW>
```

これは、この関数が式の名前を推論できないためです。デフォルトの ROW タグは、XMLGenFormatType オブジェクトを演算子の最初の引数に指定することによって変更できます。

たとえば、前述の場合、タグ名として EMPNO を使用する結果を取得するには、次のとおり、書式設定引数を関数に指定します。

```
select sys_xmlgen(empno *2,
     sys.xmlgenformattype.createformat('EMPNO')).getClobVal()
from dual;
```

これによって、次の XML が出力されます。

```
<?xml version="1.0"?>
<EMPNO>60</EMPNO>
```

注意： 現在、CURSOR 式は入力としてサポートされていません。

SYS_XMLGEN の例 2: ユーザー定義型から XML への変換

ユーザー定義型の値を SYS_XMLGEN() に入力すると、ユーザー定義型は、正規マッピングを使用して XML 文書にマップされます。正規マッピングでは、ユーザー定義型の属性は XML 要素にマップされます。

「@」で始まる名前の型属性は、その前の要素の属性にマップされます。ユーザー定義型を使用して、結果の XML 文書内でネストすることができます。

DBMS_XMLGEN の項（5-54 ページの「DBMS_XMLGEN の例 4: 複雑な XML の生成 - XML 文書内でネストするためのユーザー定義型の入力」）に示す例を使用して、次のとおり、従業員および部門の例の階層 XML を生成できます。

```
SELECT SYS_XMLGEN(
  dept_t(deptno,dname,
        CAST(MULTISET(
            select empno, ename
            from emp e
            where e.deptno = d.deptno) AS emplist_t))).getClobVal()
     AS deptxml
FROM dept d;
```

MULTISET 演算子は、その部門の従業員のサブセット結果を、リストとそれを囲む CAST として扱い、それを適切なコレクション型にキャストします。こうすることで、それを囲む部門インスタンスを作成し、DBMS_XMLGEN ルーチンをコールして、オブジェクト・インスタンスの XML を作成します。

部門の行ごとの結果は次のとおりです。

```
<?xml version="1.0"?>
<ROW DEPTNO="100">
  <DNAME>Sports</DNAME>
  <EMPLIST>
    <EMP_T EMPNO="200">
      <ENAME>John</ENAME>
    <EMP_T>
    <EMP_T>
      <ENAME>Jack</ENAME>
    </EMP_T>
 </EMPLIST>
</ROW>
```

関数が入力オペランドの名前を直接推論できないため、デフォルト名「ROW」が存在します。

この例では、SYS_XMLGEN と DBMS_XMLGEN の違いが明確になります。

- SYS_XMLGEN は、SQL 問合せ内で動作し、行内の式および列を操作します。
- DBMS_XMLGEN は、結果セット全体で動作します。

SYS_XMLGEN の例 3: XMLType インスタンスの変換

XML 文書を SYS_XMLGEN() に渡す場合、SYS_XMLGEN は、タグ名がデフォルトの「ROW」である要素、または書式設定オブジェクトを介して渡される名前で、その文書（またはフラグメント）を囲みます。この機能を使用して、文書フラグメントを整形式の文書に変換できます。

たとえば、次の文書の抽出操作では、フラグメントを戻すことができます。次の文書から EMPNO 要素を抽出するとします。

```
<DOCUMENT>
  <EMPLOYEE>
    <ENAME>John</ENAME>
    <EMPNO>200</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ENAME>Jack</ENAME>
    <EMPNO>400</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
```

```
    <ENAME>Joseph</ENAME>
    <EMPNO>300</EMPNO>
  </EMPLOYEE>
</DOCUMENT>
```

次の文を使用します。

```
select e.xmldoc.extract('/DOCUMENT/EMPLOYEE/ENAME')
   from po_xml_tab e;
```

次の文書フラグメントが生成されます。

```
<ENAME>John</ENAME>
<ENAME>Jack</ENAME>
<ENAME>Joseph</ENAME>
```

このフラグメントを有効な XML 文書にするには、次のとおり、SYS_XMLGEN() をコールして、文書の前後に囲み要素を置きます。

```
select SYS_XMLGEN(e.xmldoc.extract('/DOCUMENT/EMPLOYEE/ENAME')).getclobval()
   from po_xml_tab e;
```

これによって、次のとおり、結果の前後に要素「ROW」が置かれます。

```
<?xml version="1.0"?>
<ROW>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</ROW>
```

注意： 入力が列の場合、その列名はデフォルトとして使用されます。囲み要素名は、書式設定オブジェクトを使用して変更できます。このオブジェクトは、追加の引数として関数に渡すことができます。

SYS_XMLGEN() の例 4: オブジェクト・ビューでの SYS_XMLGEN() の使用

```
-- create Purchase Order object type
CREATE OR REPLACE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO                  NUMBER,
  Customer              Customer_typ,
  OrderDate             DATE,
  ShipDate              TIMESTAMP,
  LineItems_ntab        LineItems_ntabtyp,
  ShipToAddr            Address_typ
 )
/

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
  WITH OBJECT IDENTIFIER (PONO)
   AS SELECT P.PONO,
             Customer_typ(P.Custno,C.CustName,C.Address,C.PhoneList),
             P.OrderDate,
             P.ShipDate,
             CAST( MULTISET(
                    SELECT LineItem_typ( L.LineItemNo,
                                 StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                          L.Quantity, L.Discount)
                     FROM LineItems_tab L, Stock_tab S
                     WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
                AS LineItems_ntabtyp),
         Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
        FROM PO_tab P, Customer C
        WHERE P.CustNo=C.custNo;

-----------------------------------------------------
-- Use SYS_XMLGEN() to generate PO in XML format
-----------------------------------------------------
set long 20000
set pages 100
SELECT SYS_XMLGEN(value(p),
               sys.xmlgenformatType.createFormat('PurchaseOrder')).getClobVal() PO
FROM po p
WHERE p.pono=1001;
```

これによって、XML 形式で PO が戻されます。

```
<?xml version="1.0"?>
<PurchaseOrder>
 <PONO>1001</PONO>
 <CUSTOMER>
  <CUSTNO>1</CUSTNO>
  <CUSTNAME>Jean Nance</CUSTNAME>
  <ADDRESS>
   <STREET>2 Avocet Drive</STREET>
   <CITY>Redwood Shores</CITY>
   <STATE>CA</STATE>
   <ZIP>95054</ZIP>
  </ADDRESS>
  <PHONELIST>
   <VARCHAR2>415-555-1212</VARCHAR2>
  </PHONELIST>
 </CUSTOMER>
 <ORDERDATE>10-APR-97</ORDERDATE>
 <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
 <LINEITEMS_NTAB>
  <LINEITEM_TYP LineItemNo="1">
   <ITEM StockNo="1534">
    <PRICE>2234</PRICE>
    <TAXRATE>2</TAXRATE>
   </ITEM>
   <QUANTITY>12</QUANTITY>
   <DISCOUNT>0</DISCOUNT>
  </LINEITEM_TYP>
  <LINEITEM_TYP LineItemNo="2">
   <ITEM StockNo="1535">
    <PRICE>3456.23</PRICE>
    <TAXRATE>2</TAXRATE>
   </ITEM>
   <QUANTITY>10</QUANTITY>
   <DISCOUNT>10</DISCOUNT>
  </LINEITEM_TYP>
 </LINEITEMS_NTAB>
 <SHIPTOADDR/>
</PurchaseOrder>
```

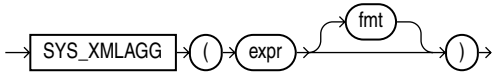
SYS_XMLAGG

SYS_XMLAGG は、すべての入力文書を集計し、単一の XML 文書を生成します。また、フラグメントを集計（連結）します。SYS_XMLAGG は、引数として XMLType を取り、行内のすべての XML 文書を、グループごとに 1 つの文書に集計（連結）します。SYS_XMLAGG() を使用して、次のいずれかのタスクを実行します。

- フラグメントの集計
- 関連 XML データの集計

図 5-4 に、SYS_XMLAGG 関数がすべての XML 文書またはフラグメント（*expr* によって表される）を集計し、単一の XML 文書を生成する方法を示します。この関数は、デフォルト名 ROWSET の新しい囲み要素を追加します。XML 文書を別の方法でフォーマットするには、*fmt* を指定します。これは SYS.XMLGenFormatType オブジェクトのインスタンスです。

図 5-4 SYS_XMLAGG の構文



次に例を示します。

```
SQL> SELECT SYS_XMLAGG(SYS_XMLGEN(last_name)
             2  ,      SYS.XMLGENFORMATTYPE.createFormat
             3         ('EmployeeGroup')).getClobVal()
             4  FROM employees
             5  GROUP BY department_id;
```

これによって、次の XML 文書が生成されます。

```
<EmployeeGroup>
     <last_name>Scott</last_name>
     <last_name>Mary<last_name>
</EmployeeGroup >
<EmployeeGroup >
     <last_name>Jack</last_name>
     <last_name>John>/last_name>
</EmployeeGroup >
```

SYS_XMLAGG の例 1: リレーショナル・データからの XML の集計

次の SELECT 文を考えてみます。

```
SELECT SYS_XMLAGG(SYS_XMLGEN(ename)).getClobVal() xml_val
FROM scott.emp
GROUP BY deptno;
```

これによって、次の XML 文書が戻されます。

```
xml_val
-------------
<ROWSET>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</ROWSET>

<ROWSET>
  <ENAME>Scott</ENAME>
  <ENAME>Adams</ENAME>
</ROWSET>

2 rows selected
```

ここでは、特定の部門に所属するすべての従業員をグループ化し、SYS_XMLGEN 関数によって生成されたすべての XML 文書を集計しています。前述の SQL 文で、各グループの結果を EMPLOYEE タグで囲むには、次の文を使用します。

```
select sys_xmlagg(sys_xmlgen(ename),
    sys.xmlgenformattype.createformat('EMPLOYEE')).getClobVal() xmlval
          from scott.emp
          group by deptno);
```

これによって、次の結果が戻されます。

```
xml_val
-------------
<EMPLOYEE>
 <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</EMPLOYEE>

<EMPLOYEE>
  <ENAME>Scott</ENAME>
  <ENAME>Adams</ENAME>
</EMPLOYEE>

2 rows selected
```

SYS_XMLAGG の例 2: 表に格納された XMLType インスタンスを集計する場合

表に格納された XMLType インスタンス、または関数から選択された XMLType インスタンスを集計することもできます。次のとおり定義された表があるとします。

```
CREATE TABLE po_tab
(
  pono  number primary key,
  orderdate date,
  poxml sys.XMLType;
);

insert into po_Tab values (100,'10-11-2000',
   '<?xml version="1.0"?><PO pono="100"><PONAME>Po_1</PONAME></PO>');
insert into po_Tab values (200,'10-23-1999',
   '<?xml version="1.0"?><PO pono="200"><PONAME>Po_2</PONAME></PO>');
```

これで、次のとおり、SYS_XMLAGG 関数を使用して発注書を単一の発注書に集計できます。

```
select SYS_XMLAGG(poxml,sys.xmlgenformattype.createformat('POSET'))
from po_Tab;
```

これによって、次の形式の単一の XML 文書が生成されます。

```
<?xml version="1.0"?>
<POSET>
  <PO pono="100">
    <PONAME>Po_1</PONAME>
  </PO>
  <PO pono="200">
    <PONAME>Po_2</PONAME>
  </PO>
</POSET>
```

SYS_XMLAGG の例 3: XMLType フラグメントの集計

XMLType の Extract() 関数を使用して、XML 文書のフラグメントを抽出できます。SYS_XMLAGG 関数を使用して、これらのフラグメントを集計することもできます。たとえば、前述の例で、PONAME 要素のみを抽出する場合、次のとおり、SYS_XMLAGG 関数を使用してこれらの要素を集計できます。

```
select SYS_XMLAGG(p.po_xml.extract('//PONAME')).getclobval()
from po_tab p;
```

これによって、次の XML 文書が生成されます。

```
<?xml version="1.0"?>
<ROWSET>
  <PONAME>Po_1</PONAME>
  <PONAME>Po_2</PONAME>
</ROWSET>
```

SYS_XMLAGG の例 4: すべての発注書から単一の XML 文書への集計

```
set long 20000
set pages 200
SELECT SYS_XMLAGG(SYS_XMLGEN(value(p),
            sys.xmlgenformatType.createFormat('PurchaseOrder'))).getClobVal() PO
FROM po p;
```

これによって、すべての発注書が単一の XML 文書（ROWSET 要素で囲まれた部分）に戻されます。

```
<?xml version="1.0"?>
<ROWSET>
<PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
    <CUSTNO>1</CUSTNO>
    <CUSTNAME>Jean Nance</CUSTNAME>
    <ADDRESS>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>95054</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>415-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>10-APR-97</ORDERDATE>
  <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS_NTAB>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1534">
        <PRICE>2234</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
```

```
      <QUANTITY>12</QUANTITY>
      <DISCOUNT>0</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
      <ITEM StockNo="1535">
        <PRICE>3456.23</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>10</QUANTITY>
      <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
  </LINEITEMS_NTAB>
  <SHIPTOADDR/>
</PurchaseOrder>
<PurchaseOrder>
  <PONO>2001</PONO>
  <CUSTOMER>
    <CUSTNO>2</CUSTNO>
    <CUSTNAME>John Nike</CUSTNAME>
    <ADDRESS>
      <STREET>323 College Drive</STREET>
      <CITY>Edison</CITY>
      <STATE>NJ</STATE>
      <ZIP>08820</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>609-555-1212</VARCHAR2>
      <VARCHAR2>201-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS_NTAB>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1004">
        <PRICE>6750</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>1</QUANTITY>
      <DISCOUNT>0</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
      <ITEM StockNo="1011">
```

```
        <PRICE>4500.23</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>2</QUANTITY>
      <DISCOUNT>1</DISCOUNT>
    </LINEITEM_TYP>
  </LINEITEMS_NTAB>
  <SHIPTOADDR>
    <STREET>55 Madison Ave</STREET>
    <CITY>Madison</CITY>
    <STATE>WI</STATE>
    <ZIP>53715</ZIP>
  </SHIPTOADDR>
</PurchaseOrder>
</ROWSET>
```

その他の集計方法

ROLLUP および CUBE

Oracle は、CUBE や ROLLUP などのオンライン分析処理（OLAP）操作に対する強力な機能を提供します。SYS_XMLAGG 関数は、これらの場合も機能します。たとえば、ROLLUP または CUBE 操作に基づいて様々な XML 文書を作成できます。

WINDOWING 関数

Oracle は、累積集計、移動集計および集中集計を計算するために使用する WINDOWING 関数を提供します。SYS_XMLAGG を使用して、ランクおよびパーティションに基づいて文書を作成することもできます。

表関数

前述の項では、Oracle が導入した新しい関数および演算子について説明しました。これらは、XPath に類似の構文を使用した XML インスタンスの問合せに有効です。ただし、XML を単純なリレーショナル・データまたはオブジェクト・リレーショナル・データに分解して、それを表に挿入したり、標準リレーショナル SQL 文を使用して問合せができるようにする必要もあります。これを行うには、表関数という強力なメカニズムを使用します。

表関数は、Oracle の新機能です。この関数を使用すると、任意のデータ（データベース内部のデータまたは外部ソースからのデータ）を SQL 行のコレクションとしてモデル化できます。表関数はパイプライン化されて、パラレル実行されるため、パフォーマンスが向上します。表関数を使用して、XML を SQL 行に分割できます。この SQL 行は、通常の SQL 問合せを実行したり、通常のリレーショナル表またはオブジェクト・リレーショナル表に挿入することができます。

Oracle8*i* では、コレクションを戻す関数を、SELECT 文の FROM 句で使用できます。ただし、この関数を外部問合せブロックで使用するには、コレクション全体を生成しておく必要があります。

表関数では、これらのコレクションはパイプライン化され、パラレルになっています。そのため、この関数はメモリー内でコレクション全体をインスタンス化する必要がなく、かわりに、結果を外部の問合せブロックにパイプライン化します。これによって、応答時間が短縮され、メモリー使用量が減ります。

> **参照：**『PL/SQL ユーザーズ・ガイドおよびリファレンス』の「PL/SQL のサブプログラム」を参照してください。

XML での表関数の使用

XML では、これらの表関数を使用して、XML を SQL 行に分割できます。この SQL 行は、通常の SQL 問合せを実行したり、通常のリレーショナル表またはオブジェクト・リレーショナル表に挿入することができます。この関数はパラレルで、パイプライン化されているため、このような操作のパフォーマンスが大幅に向上します。

XMLType または CLOB を取り、予約済コレクション型を戻すインスタンスの関数を定義できます。たとえば、この関数は、Oracle で使用可能な XML パーサーを使用して SAX 解析を実行し、結果を戻したり、`extract()` 関数を使用して XML 文書のフラグメントを抽出し、それを戻すことができます。

表関数の例 1: PO の分解によるリレーショナル表への格納

前述の項で説明した発注書があり、その文書を分解して発注書の詳細を含むリレーショナル表に格納する必要がある場合、最初に型を作成して、結果の構造を記述します。

```
create type poRow_type as object
(
  poname varchar2(20),
  postreet varchar2(20),
  pocity varchar2(20),
  postate char(2),
  pozip char(10)
);
/
create type poRow_list as TABLE of poRow_type;
/
```

これで、ODCI 実装タイプを作成して TABLE インタフェースを実装するか、ネイティブ PL/SQL を使用できます。

> **参照：** インタフェース定義の詳細および本体の例は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』の「PL/SQL のサブプログラム」を参照してください。

関数自体を作成することによって実装タイプの本体を PL/SQL で作成したと想定すると、表関数は、次の方法で定義できます。

```
create function poExplode_func (arg IN sys.XMLType) return poRow_list
pipelined is
  out_rec poRow_type;
  poxml sys.XMLType;
  i binary_integer := 1;
  argnew sys.XMLType := arg;
begin

  loop

   -- extract the i'th purchase order!
   poxml := argnew.extract('//PO['||i||']');
   exit when poxml is null;

   -- extract the required attributes..!!!
   out_rec.poname := poxml.extract('/PONAME/text()').getStringVal();
   out_rec.postreet := poxml.extract('/POADDR/STREET/text()').getStringVal();
   out_rec.pocity := poxml.extract('/POADDR/CITY/text()').getStringVal();
   out_rec.postate := poxml.extract('/POADDR/STATE/text()').getStringVal();
   out_rec.pozip := poxml.extract('/POADDR/ZIP/text()').getStringVal();
   PIPE ROW(out_rec);

  i := i+1;

  end loop;
  return;
end;
/
```

この関数は FROM リストで使用できます。また、Imp_t で定義されたインタフェースは自動的にコールされ、パイプライン化された方法で値を取得します。

```
select *
   from TABLE( CAST(
      poExplode_func(
       sys.XMLType.createXML(
         '<?xml version="1.0"?>
          <POLIST>
           <PO>
            <PONAME>Po_1</PONAME>
            <POADDR>
              <STREET>100 Main Street</STREET>
              <CITY>Sunnyvale</CITY>
              <STATE>CA</STATE>
              <ZIP>94086</ZIP>
            </POADDR>
           </PO>
           <PO>
            <PONAME>Po_2</PONAME>
            <POADDR>
              <STREET>200 First Street</STREET>
              <CITY>Oaksdale</CITY>
              <STATE>CA</STATE>
              <ZIP>95043</ZIP>
            </POADDR>
           </PO>
          </POLIST>')
   ) AS poRow_list));
```

> **注意：** 前述の例では、XMLType コンストラクタを使用して、XML 文書が構成されています。バインド変数または選択リスト副問合せを使用して、値を取得することもできます。

SQL 文は、次の値を戻します。

```
PONAME     POSTREET         POCITY     POSTATE    POZIP
-------------------------------------------------------------
Po_1      100 Main Street  Sunnyvale    CA        94086
Po_2      200 First Street Oaksdale    CA        95043
```

これらの値は、リレーショナル表に挿入するか、SQL で問い合せることができます。

FAQ: XMLType

XMLType でのレプリケーションおよびマテリアライズド・ビュー（MV）のサポート

質問

レプリケーションおよび MV での XMLType の使用に関して、何か問題がありますか？

回答

レプリケーションは、XMLType を別のユーザー定義型として扱います。そのため、XMLType は、別のユーザー定義型として機能します。dbms_defer_query およびユーザー定義の競合解消ルーチンでは、将来のリリースまで XMLType がサポートされません。したがって、今回のリリースでは、レプリケーションおよび MV は、XMLType で完全にサポートされません。

データベース・レコード内の XML タグの更新方法

質問

データベース表のレコード内の XML タグは更新できますか？また、XML タグに基づいて索引を作成できますか？

回答

今回のリリースでは、CLOB 記憶域を含む XMLType が提供されています。更新は単一の文書レベルで可能なため、文書全体を置換する必要があります。

XMLType は、Oracle Text（*inter*Media Text）を使用して索引付けできます。この種類の索引は、特定の XML 検索基準と一致する文書の検索には最適です。ただし、アプリケーションが「数量」や「価格」などのデータを選択して、（たとえば、グラフ作成ソフトウェアまたはデータ・ウェアハウス・ソフトウェアで）そのデータを処理する場合は、データ指向 XML を実表および実列として格納する方法が最適です。また、ファンクション索引を使用して、特定の予約済 XPath 式を高速化することもできます。

> **参照：** 次の章およびマニュアルを参照してください。
>
> - 第 8 章「Oracle Text を使用した XML データの検索」
> - 『Oracle Text アプリケーション開発者ガイド』
> - 『Oracle Text リファレンス』

Oracle8*i* における例については、『Building Oracle XML Applications』（Steve Muench 著、O'Reilly 出版）を参照してください。

XMLType による属性制約などのビジネス・ルールの規定のサポート

質問

Oracle の XML データ型では、データ内の要素の属性としてビジネス・ルールまたは制約を規定できますか？次に例を示します。

```
<Address>
  <Street type=string> </Street>
  <City type=string> </City>
  <State type=string size=2> </State>
  <Zip type=number size=5> </Zip>
</Address>
```

回答

Oracle では、トリガー・メカニズムを使用して制約を規定できます。現在、Oracle では記憶域を CLOB としてのみサポートしており、本質的に制約チェックは使用できません。将来のリリースでは、XML Schema の準拠性を実現する可能性があります。その場合は、XML Schema に準拠する列の作成などが実行できます。これによって、XML Schema が規定するすべての制約が使用できるようになります。

日本語用の適切なエンコーディングを使用した XML 文書の作成

質問

Oracle で、SYS_XMLGEN() を使用して日本語で文書を作成する必要があります。データベース・キャラクタ・セットは EUC_JP ですが、SYS_XMLGEN()、SYS_XMLAGG() および DBMS_XMLGEN() を使用して XML を作成すると、次の XML 宣言を含む文書が生成されます。

```
<?xml version="1.0"?>
```

次のような適切な宣言を含む XML 文書を作成できますか？

```
<?xml version="1.0" encoding="EUC_JP"?>
```

回答

Oracle では、エンコーディング・タグを使用して XML 出力を生成することができません。バイナリ・データ型のデータは、データベース・キャラクタ・セットに変換可能でない場合、変換が正常に実行されません。

エンコーディング宣言は、それ自体が文書の一部を構成しないことに注意してください。これは、プロセッサがそのエンコーディングを識別するために有効な識別子です。

XML をデータベースに格納するための最適な方法

質問

XML をデータベースに格納するための最適な方法を教えてください。XML データを NCLOB に格納できますか？または、XML データを BLOB に格納し、アプリケーションのみで、そのデータを操作する方が適切ですか？

回答

Oracle で XML を格納する最適な方法は、`XMLType` を使用する方法です。現在、Oracle では、CLOB 格納のみが固有にサポートされていますが、将来のリリースでは、NCLOB、BLOB などでの格納が固有にサポートされる可能性があります。

Oracle Database では、XML を次の方法で格納できます。

- XMLType: 推奨の方法です。これによって、サーバーがデータが XML であることを認識できます。XML ベースの問合せおよび索引付けを実行できます。
- BLOB: これによって、元の文書と同じエンコーディングで XML 文書をそのまま格納できます。デメリットは、すべてのエンコーディング上の問題を処理する必要があることです。
- NCLOB: XML を NCLOB として格納すると、データに対して SQL 操作を実行したり、コンテキスト検索を実行することができます。コンテキスト検索は、データベース・キャラクタ・セットが、NCHAR の適切なスーパーセットまたは変換可能セットである（データを消失することなく、NCLOB をデータベース・キャラクタ・セットに変換できる）場合にのみサポートされています。

6

DBURI 参照

この章の内容は次のとおりです。

URI 参照の概念

URI 参照の概要

URI 参照は、URL の概念を一般化したものです。今回のリリースでは、URI は、HTML や XML を含む、すべてのドキュメントを参照できます。また、URI 参照は、ドキュメントに対するポインタ・セマンティクスも提供します。URI 参照は、次の 2 つの部分で構成されます。

- 正規の URL 仕様に準拠した URL 部分
- ドキュメント内のフラグメントを識別するフラグメント部分

URI 参照は、対象ドキュメントの型に固有の言語で記述されます。フラグメント部分は、次の項に示す例の「#」以降の部分です。今回のリリースでは、フラグメント部分はサポートされません。

XML 文書のビューから作成された URL パス

図 6-2 に、データベース内のリレーショナル表 Emp に格納されている XML データのビュー、および XML 文書の要素にマップされているデータ列を示します。このマッピングは、「XML のビジュアル化」ともいいます。結果の URL パスは、XML 文書のビューから簡単に作成できます。

URI 参照の典型的な形式は次のとおりです。

- HTML の場合 : http://www.url.com/document1#A

  ここで、A はドキュメント内のアンカーです。

- XML の場合 : http://www.xml.com/xml_doc#//po/cust/custname

  ここで、各部分は次の内容を表します。

  - 「#」の前の部分は、文書自体の場所を示します。
  - 「#」の後の部分は、文書のフラグメントを示します。この部分は W3C の XPointer 仕様で定義されています。

URI 参照による異なるプロトコルを使用したデータの取出し

Oracle データベースでは、URI 参照オブジェクトを格納および取出しするための、新しいデータ型が導入されています。詳細は、6-3 ページの「URI 参照を格納するための新しいデータ型」を参照してください。この結果、URI 参照は、HTTP などの異なるプロトコルを使用してデータを取り出すことができます。

また、Oracle には、新しい概念である **DBURI 参照**も導入されています。これは、データベース自体にある表およびビューの列および行に対する参照です。

- HTTP の URL は、グローバルなオブジェクトを参照します。
- DBURI 参照は、ローカルなオブジェクトを参照します。

DBURI 参照のメカニズムを使用して、データベース内にあるすべての表またはビューの行または列にアクセスできます。DBURI 参照は、データベース内に格納されているすべてのデータに対してデータベース内 URL を提供します。6-5 ページの「DBURI 参照およびデータベース内参照」を参照してください。

DBURI 参照を使用するメリット

DBURI 参照のメリットは次のとおりです。

- **Web サーバーをバイパスすることによって、パフォーマンスが向上します。**URI 参照はスタイルシートをローカルに参照する方法の 1 つです。たとえば、データベース管理システムのメタデータは、DBURI 参照を使用して、多くのスタイルシートを参照します。XML 文書内に URL があり、それをデータベースへの参照に置換する必要がある場合、次の方法で結果を取得できます。
  - サーブレットの使用
  - DBURI 参照などの入力メカニズムの使用

  DBURI 参照を使用すると、Web サーバーを介すことなくデータベースと直接対話できるため、パフォーマンスが向上します。
- **SQL が必要がありません。**データベース内に格納されている XML 文書にアクセスするために、SQL の知識が必要ありません。DBURI 参照では、SQL セマンティクスを使用してデータベースの XML 文書にアクセスできます。この場合、SQL を使用する必要はありません。そのため、SQL プログラマでないユーザーも、データベース内に格納されている XML 文書に簡単にアクセスできます。

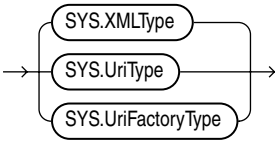
URI 参照を格納するための新しいデータ型

Oracle では、URI 参照を格納するための新しいデータ型が導入されています。

- **UriType:** HttpUriType または DBUriType のインスタンスを格納できる抽象オブジェクト型です。
- **DBUriType:** DBURI 参照が指すデータを取得できます。
- **HttpUriType:** リモート・ページにアクセスするための HTTP プロトコルを実装します。
- **UriFactoryType:**

参照： 図 6-3「UriType インスタンスを生成する UriFactory」を参照してください。

図 6-1 新しい UriType



これらのデータ型は、オブジェクトによって指されるオブジェクトまたはページにアクセスするために使用できるメンバー関数を持つオブジェクト型です。そのため、UriType を使用して次の操作が可能です。

- データベース内外のデータを指す列の作成
- UriType の提供する抽象関数を使用してのデータベース列の問合せ

UriType を使用するメリット

Oracle は、URL 参照のフェッチに対して、PL/SQL での UTL_HTTP および Java での java.net.URL をサポートしています。SQL で新しい UriType データ型を定義するメリットは次のとおりです。

- **型付けの改善によって、索引付け、ナビゲーションおよび問合せがより効率的になります。**URI 参照によって、データベースが表またはビューの列に格納可能な新しい URL 型を認識します。列の型付けの改善によって、URL を認識する新しい索引付けスキームが実現し、URL を介したより効率的なナビゲーション機能や問合せ機能が得られます。
- **列に対する XML 文書のマッピングが改善されます。**XML 文書をオブジェクト・リレーショナル列に分解するには、URI 参照のサポートが必要です。そのため、文書で指定された URI 参照は、データベース内の URL 列にマップできます。

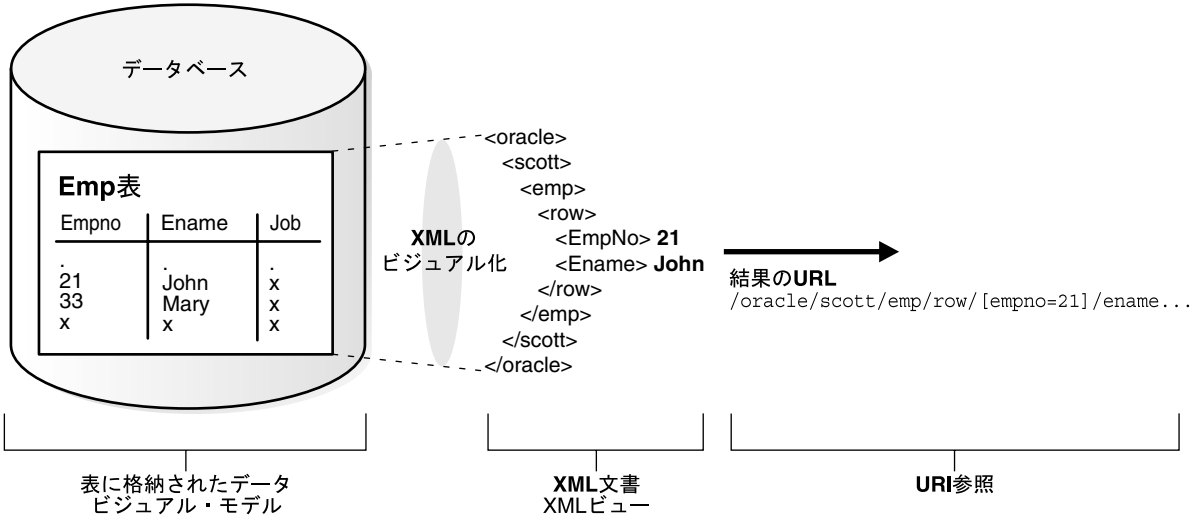
参照： 6-14 ページの「URI 参照型（UriType）の使用」を参照してください。

DBURI 参照およびデータベース内参照

DBURI 参照（データベース関連の URI）は、URI 参照メカニズムの特殊なケースです。この場合、DBURI 参照は、データベースおよびセッションのコンテキスト内で動作することが保証されています。DBURI 参照は、HTTP の URL のようなグローバルな参照ではなく、データベース内のローカル参照（ローカル URL）です。

また、DBURI 参照を処理できるサーブレットを識別する HTTP の URL パスに、DBURI 参照を「追加」することによって、この URL が指すオブジェクトへグローバルにアクセスすることもできます。詳細は、6-25 ページの「サーブレットを使用したブラウザから DBURI 参照へのアクセス」を参照してください。

図 6-2 DBURI 参照の説明



DBUri の表現

URL の構文は、データベースの仮想的な XML のビジュアル化で XPath に類似した構文を指定することによって取得されます。図 6-2「DBURI 参照の説明」を参照してください。

「ビジュアル・モデル」は、SQL スキーマ、表、行および列に関して現在接続中のユーザーが参照する階層的なビューです。

「XML ビュー」には、データベースにマップするルート要素が含まれます。ルート XML 要素には、子要素が含まれます。これらの子要素は、ユーザーが任意のオブジェクトに対する権限を取得しているスキーマです。スキーマ要素には、ユーザーが参照できる表およびビューが含まれます。たとえば、ユーザー *scott* は次の仮想文書を参照できます。

```
<?xml version='1.0'?>
<oradb SID="ORCL">
 <PUBLIC>
    <ALL_TABLES>
       ..
     </ALL_TABLES>
    <EMP>
      <!-- EMP table -->
    </EMP>
 <PUBLIC>
 <SCOTT>
  <ALL_TABLES>
    ....
  </ALL_TABLES>
  <EMP>
   <ROW>
     <EMPNO>1001</EMPNO>
     <ENAME>John</ENAME>
     <EMP_SALARY>20000</EMP_SALARY>
   </ROW>
   <ROW>
     <EMPNO>2001</EMPNO>
     <ENAME xsi:null="true"/>
     <EMP_SALARY xsi:null="true"/>
   </ROW>
  </EMP>
  <DEPT>
   <ROW>
     <DEPTNO>200</DEPTNO>
     <DNAME>Sports</DNAME>
```

```
    </ROW>
   </DEPT>
</SCOTT>
<JONES>
 <CUSTOMER_OBJ_TAB>
    <ROW>
      <NAME>xxx</NAME>
      <ADDRESS>
          <STATE>CA</STATE>
          <ZIP>94065</ZIP>
      </ADDRESS>
    </ROW>
  </CUSTOMER_OBJ_TAB>
 </JONES>
</database>
```

これは、アクセス時に取得している権限に基づいた仮想的な XML 文書であることに注意してください。

この例から、次のことがわかります。

- ユーザー scott は、scott スキーマおよび jones スキーマを参照できます。これらは、ユーザーが読み取ることができる表またはビューが存在するスキーマです。
- emp 表は、EMP として表示され、行の要素タグが含まれます。これはすべての表に対するデフォルトのマッピングです。dept 表および JONES スキーマの customer_obj_tab 表でも同様です。
- NULL 要素は空として表示されます。将来のリリースでは、NULL を示す特別な NULL 属性を指定する XML Schema 仕様に準拠するように変更されます。
- また、スキーマ修飾がなくてもアクセスできる表およびビューが存在する PUBLIC 要素もあります。たとえば、次のような SELECT 問合せ文を発行するとします。

  ```
  select * from emp;
  ```

  この場合、ユーザー scott による問合せでは、scott スキーマ内の emp 表と一致します。このような表が検出されない場合、使用可能なパブリック・シノニムとの一致を試みます。同様に、PUBLIC 要素には、ユーザーが自スキーマを介して参照できるすべての表やビュー、および PUBLIC シノニムを介して参照できるすべての表が含まれます。

DBUri の仕様

データベースが XML ツリーとしてビジュアル化される場合、仮想文書の任意の部分に対して XPath 検索を実行できます。これによって、データベース表およびビューにある行と列のすべての共通部分を取得できます。ビジュアル・モデル上で XPath を指定することによって、データベースのすべてのデータに対する参照を作成できます。

DBUri は、簡素化された XPath 形式で指定します。今回のリリースでは、DBURI 参照に対する XPath または XPointer が完全にサポートされません。次の項では、これらの DBUri の構造について説明します。

注意： グローバルな HTTP の URL を介して DBUri を公開する場合、XPath 構文では「]」や"」などの特定の文字を「エスケープ」する必要がある場合があります。エスケープ済の URL を取得するには、UriType 型の `getExternalUrl()` 関数を使用します。

前述のとおり、すべてのデータに対して DBUri を作成できます。次の単位の参照を使用できます。

- 列内の、スカラー、オブジェクトまたはコレクションのインスタンス
- オブジェクト型の属性

注意： 現在、スカラー、XMLType または LOB のデータ値内の参照はサポートされていません。

また、DBUri を使用して、グローバルに参照可能な URL を作成することもできます。詳細は、6-25 ページの「サーブレットを使用したブラウザから DBURI 参照へのアクセス」を参照してください。

DBUri の構文のガイドライン

参照を指定するために使用できる XPath 問合せの種類には制限があります。通常、DBURI 参照は、次の構文ガイドラインに従う必要があります。

- 特定のスキーマを持たない表名を変換するために、ユーザー・スキーマ名または PUBLIC を含める必要があります。
- 表名またはビュー名を含める必要があります。
- ROW 要素を識別するために ROW タグを含める必要があります。
- 抽出する列またはオブジェクト属性を識別する必要があります。
- パス内のスキーマ要素および表要素以外のレベルに述語を含める必要があります。
- 選択パスではなく、ROW 要素内に述語を指定する必要があります。たとえば、選択パスが /scott/purchase_obj_tab[..]/ROW/line_item_list で、述語 pono = 100 を指定する場合、ROW ノードとともに pono 述語を含める必要があります。次に例を示します。

```
/scott/purchase_obj_tab/ROW[pono=100]//line_item_list
```

- DBURI 参照は、単一のデータ値を正確に識別する必要があります。これらの値は、オブジェクト型またはコレクションの場合があります。データ値は、行全体になることもあります。この場合、ROW ノードを使用してそれを明示する必要があります。また、URI 参照は表全体を指すこともできます。

DBUri での述語（XPath）式の使用

述語式では、次の XPath 式を使用できます。

- ブール演算子 : AND、OR および NOT
- 関係演算子 : <、>、<=、!=、>=、=、mod、div、*（かけ算）

注意 :

- child 軸以外の XPath 軸は、サポートされていません。そのため、ワイルド・カード（*）、子孫（//）および他の操作は無効です。
- `text()` 以外の XPath 関数はサポートされていません。また、`text()` は、スカラー・ノード上でのみ有効です。そのため、`text()` ノードは、ROW レベルや表レベルなどでは適用できません。

述語は、スキーマ要素および表要素以外のすべての要素で定義できます。オブジェクト列がある場合、属性値も検索できます。たとえば、アドレスが emp 表内の列であり、州、都市、通り、郵便番号などの属性を含む場合、次の DBURI 参照は有効です。

```
/SCOTT/EMP/ROW[ADDRESS/STATE='CA' OR ADDRESS/STATE='OR']/ADDRESS[CITY='Portland' OR
./ZIPCODE=94404]/CITY
```

この DBURI 参照は、州が California か Oregon、または都市名が Portland か郵便番号が 94404 である、emp 表内のアドレス列の都市属性を識別します。

DBURI 参照の一般的な使用例

DBURI 参照は、表、特定の行、行内の特定の列やオブジェクト列の特定の属性など、様々なオブジェクトを識別できます。次に DBURI 参照の一般的な使用例を示します。

1. **表全体の指定** : この例では、表全体を取り出す XML 文書を戻します。囲みタグは表名です。行の値は「ROW」要素に囲まれます。

   次の構文を使用します。

   ```
   /<schemaname>/<tablename>
   ```

   次に例を示します。

   ```
   /SCOTT/EMP
   ```

   次の XML 文書が戻ります。

   ```
   <?xml version="1.0"?>
   <EMP>
    <ROW>
     <EMPNO>7369</EMPNO>
     <ENAME>Smith</ENAME>
       ... <!-- other columns -->
    </ROW>
    <!-- other rows -->
   </EMP>
   ```

2. **表の特定の行の識別** : この例では、表内の特定の ROW 要素を識別します。結果は、ROW 要素およびその子要素としての列を含む XML 文書です。

   次の構文を使用します。

   ```
   /<schemaname>/<tablename>/ROW[<predicate_expression>]
   ```

   次に例を示します。

   ```
   /SCOTT/EMP/ROW[EMPNO=7369]
   ```

次の XML 文書が戻ります。

```
<?xml version="1.0"?>
<ROW>
  <EMPNO>7369</EMPNO>
  <ENAME>SMITH</ENAME>
  <JOB>CLERK</JOB>
     <!-- other columns -->
</ROW>
```

注意： この場合、述語式は一意の行を識別する必要があります。

3. **ターゲット列の指定**: この場合、ターゲット列または列の属性が識別され、XML として取り出されます。

注意： ネストした表または VARRY 列に対する全検索はできません。

次の構文を使用します。

```
/<schemaname>/<tablename>/ROW[<predicate expression>]/<columnname>
/<schemaname>/<tablename>/ROW[<preciate expression>]/ <columnname>/ <attribute>*
```

例 1:

```
/SCOTT/EMP/ROW[EMPNO=7369 and DEPTNO =20]/ENAME
```

この例では、次のとおり、従業員番号が 7369 で部門番号が 20 の、emp 表内の ename 列を取り出します。

```
<?xml version="1.0"?>
<ENAME>SMITH</ENAME>
```

例 2:

```
/SCOTT/EMP/ROW[EMPNO=7369]/ADDRESS/STATE
```

この例では、次のとおり、従業員番号が 7369 である従業員の住所オブジェクト列内の州属性を取り出します。

```
<?xml version="1.0"?>
<STATE>CA</STATE>
```

4. **列のテキスト値の取出し**: 多くの場合、囲みタグを取り出さずに、列のテキスト値のみを取り出す方が有効です。たとえば、XSL スタイルシートが CLOB 列に格納されている場合、XML 文書を列名のタグで囲むことなく、そのような XML 文書を取り出すことができます。これらの場合、`text()` 関数を使用して、ノードのテキスト値のみを取り出すように指定できます。

   次の構文を使用します。

   ```
   /<schemaname>/<tablename>/ROW[<predicate expression>]/<columnname>/text()
   ```

   次に例を示します。

   ```
   /SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
   ```

   従業員番号が 7369 の従業員名のテキスト値のみが、XML タグなしで取り出されます。戻される値は XML 文書ではなく、「SMITH」という値を持つテキスト・ドキュメントです。

注意： XPath のみでは、有効な URI を構成できません。Oracle では、これを DBUri と呼びますが（データベース内で URI のように動作するため）、DBUri をグローバルに有効な URI 参照に変換することもできます。

注意： DBUri では、大文字 / 小文字が区別されます。scott.emp を指定するには、SCOTT/EMP を使用します。これは、Oracle ディクショナリでは、実際の表名が大文字で格納されているためです。データベース内に小文字の表名または列名を作成するには、" " で名前を囲みます。

DBUri とオブジェクト参照の相違点

DBURI 参照は、列レベルおよび属性レベルでアクセスでき、型付けも厳密ではありません。Oracle8 のオブジェクト機能で提供されるオブジェクト参照は、システム内の行オブジェクトに対する参照です。DBURI 参照は、本質的にこの参照メカニズムのスーパーセットです。

DBURI 参照は、特定の行を識別するのみでなく、行の列またはオブジェクト属性に対するアクセスも提供します。ただし、DBURI 参照は、オブジェクト参照と異なり、厳密に型付けされていません。URI 参照の全検索の結果は、システム内のオブジェクトになる可能性があります。

データベースおよびセッションに適用される DBURI 参照

DBURI 参照の有効範囲がデータベースおよびセッションということは重要です。DBURI 参照自体はセッション固有情報を何も保持しないため、ユーザーが特定のセッション・コンテキストでデータベースに接続し、そのコンテキストで URI 参照を解析していると想定されます。これは、オブジェクト参照の参照解除に参照オブジェクトに対する読取り権限が必要な、オブジェクト参照メカニズムと似ています。

注意： 使用するセッション・コンテキストによっては、特に PUBLIC パスが使用される場合、同じ DBURI 参照でも結果が異なる場合があります。

たとえば、/PUBLIC/FOO_TAB は、scott として接続している場合は SCOTT.FOO_TAB に変換され、JONES として接続している場合は JONES.FOO_TAB に変換されます。

DBURI 参照が使用可能な場合

URI 参照は、次のような多くのシナリオで使用できます。

関連ドキュメントに対する URL の格納

旅行体験記を表に格納している旅行体験記についての Web サイトでは、関連の体験記へのリンクを作成する場合があります。この場合、関連する体験記へのデータベース内リンクを作成するため、DBUri が有効です。

データベースへのスタイルシートの格納

アプリケーションは、XSL スタイルシートを使用して、XML を他の形式に変換できます。このデータは XML に変換されます。使用する XSL スタイルシートは、CLOB に格納されます。アプリケーションは、次の目的で DBURI 参照を使用できます。

- 解析中に、データベースに格納されている XSL スタイルシートにアクセスするため
- XSL スタイルシート自体にある、import/include などの関連 XSL スタイルシートに対する参照を取得するため

注意： DBURI 参照では、次のサポートは提供されません。

- XML データに対する汎用 XPointer メカニズムは提供されません。
- DBURI 参照は、データベース・オブジェクト参照の代替機能ではありません。参照の構文およびセマンティクスは、URI 参照型の構文およびセマンティクスとは異なります。
- DBURI 参照は、新しいセキュリティ・モデルまたは制限を規定または作成しません。権限の規定に対しては、基礎となるセキュリティ・アーキテクチャに依存します。

URI 参照型（UriType）の使用

この項では、URI 参照を使用してドキュメントへのポインタを格納する方法およびデータベース内のそのような URI 参照にアクセスする方法について説明します。

UriType を使用したドキュメントへのポインタの格納

前述のとおり、UriType は、列内にサブタイプのインスタンスを格納できる抽象型です。この型は、URI 参照文字列を含む単一の VARCHAR2 属性を含み、参照の全検索およびデータ抽出用の関数を含みます。

UriType の列を作成したり、URI 参照をデータベースに格納することができます。Oracle では、HTTP および DBUri の全検索用の標準クラスが提供されています。豊富なナビゲーション関数を使用して URI をナビゲートできます。

表 6-1 に、有効な UriType メソッドを示します。

注意： 継承メカニズムを使用して、任意の新しいプロトコルをプラグインできます。Oracle は、HTTP プロトコルの処理および DBURI 参照の解読のために、HttpUriType 型および DBUriType 型を提供します。たとえば、UriType のサブタイプを実装して、gopher などのプロトコルを処理できます。

表 6-1 UriType メソッド

UriType メソッド	説明
getClob	URL が指す値を CLOB 値として戻します。文字コードは、データベース・キャラクタ・セットの文字コードになります。
getUrl	UriType に格納されている URL を戻します。属性「url」を直接使用せず、かわりにこの関数を使用してください。この関数をサブタイプでオーバーライドし、正しい URL を取得できます。たとえば、HttpUriType は、URL のみを格納して接頭辞「http://」を格納しません。そのため、getUrl() は接頭辞を追加して値を戻します。
getExternalUrl	URL 仕様に準拠するために、URL の文字をエスケープするエスケープ・メカニズムをコールすることを除けば、getUrl と同じです（たとえば、空白はエスケープ値 %20 に変換されます）。

UriType の例

UriType の例 1: 発注書のリストに対する URL 参照の作成

発注書のリストを、発注書に対する URL 参照付きで作成できます。次に例を示します。

```
CREATE TABLE uri_tab
(
   poUrl SYS.UriType,  -- Note that we have created abstract type columns
-- if we knew what kind of uri's we are going to store, we can actually
-- create the appropriate types.
   poName VARCHAR2
);

 -- insert an absolute url into SYS.UriType..!
 -- the factory will create the correct instance (in this case a HttpUriType
INSERT INTO uri_tab VALUES
  (sys.UriFactory.getUri('http://www.oracle.com/cust/po'),'AbsPo');

-- insert a URL by directly calling the SYS.HttpUriType constructor.
-- Note this isstrongly discouraged. Note the absence of the
-- http:// prefix when creating SYS.HttpUriType instance through the default
-- constructor.
INSERT INTO uri_tab VALUES (sys.HttpUriType('proxy.us.oracle.com'),'RelPo');

-- Now extract all the purchase orders
SELECT e.poUrl.getClob(), poName FROM uri_tab e;

-- In PL/SQL
declare
   a SYS.UriType;
begin

 -- absolute URL
 SELECT poUrl into a from uri_Tab WHERE poName like 'AbsPo%';
 printDataOut(a.getClob());

 SELECT poUrl into a from uri_Tab WHERE poName like 'RelPo%';
 -- here u need to supply a prefix before u can get at the data..!
 printDataOut(a.getClob());
end;
/
```

参照： UriFactory の使用方法の詳細は、6-17 ページの「UriFactory パッケージ」を参照してください。

UriType の例 2: 置換メカニズムの使用

UriType の列を直接作成し、その列に HttpUriType と DBUriType の両方を挿入できます。また、参照されるドキュメントの格納場所がわからなくても、列を問い合せることができます。

たとえば、例 1 では、次のように、uri_tab 表に DBURI 参照を挿入することもできます。

```
INSERT INTO uri_tab VALUES
  (SYS.UriFactory.getUrl(
     '/SCOTT/PURCHASE_ORDER_TAB/ROW[PONO=1000]'),'ScottPo');
```

この INSERT 文では、SCOTT スキーマ内に発注書の表があると想定しています。この場合、表の URL 列には、HTTP を介してグローバルなドキュメントを指し、データベース内の仮想ドキュメントも指す値が含まれます。

`getClob()` メソッドを使用してその列に SELECT 文を実行すると、ドキュメントの格納場所にかかわらず、結果が CLOB として取り出されます。

```
select e.poURL.getclob() from uri_tab e;
```

この SELECT 文では、最初の行に格納されているグローバル HTTP アドレス、およびローカルな DBURI 参照から値が取り出されます。

HttpUriType および DBUriType の使用

HttpUriType および DBUriType は、UriType のサブタイプです。これらは、それぞれ HTTP 参照用および DBURI 参照用の関数を実装します。

注意： 今回のリリースでは、HttpUriType は相対 HTTP 参照を格納できません。

DBUriType の例

DBUriType の例 1: DBURI 参照の作成

次の例では、DBUriType 型の列を含む表を作成し、その表に値を割り当てます。

```
CREATE TABLE DBURiTab(DBUri SYS.DBUriType, dbDocName VARCHAR2(2000));

-- insert values into it..!
INSERT INTO DBUriTab VALUES
   (sys.UriFactory.createUri('/ORADB/SCOTT/EMP/ROW[EMPNO=7369]'),'emp1');

INSERT INTO DBUriTab VALUES
   (sys.DBUriType('/SCOTT/EMP/ROW[EMPNO=7369]/'),'emp2');

-- access the references
SELECT e.DBUri.getCLOB() from dual;
```

UriFactory パッケージ

UriFactory パッケージには、ファクトリ・メソッドが含まれます。このファクトリ・メソッドを使用すると、プログラムで実装をハードコードすることなく、UriType の適切なインスタンスを生成できます。図 6-3 を参照してください。

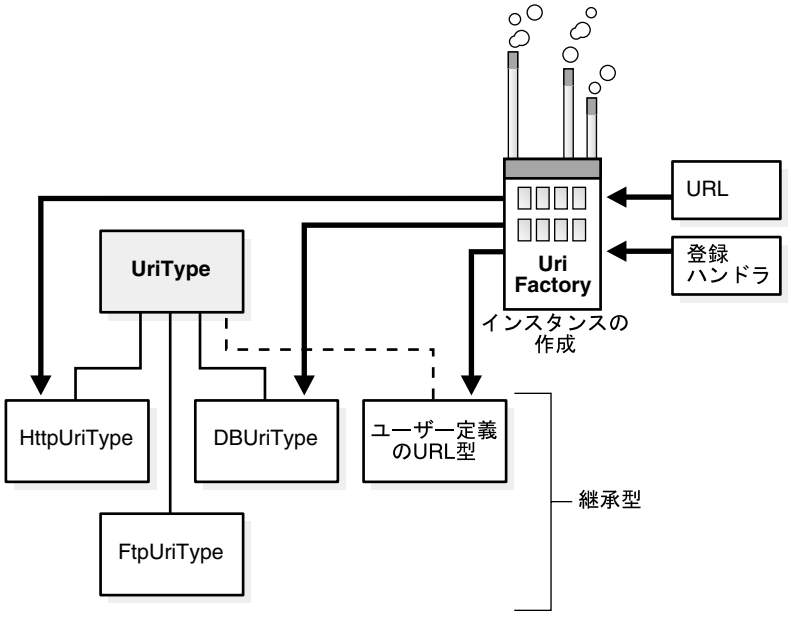
ファクトリ・メソッドは、様々な URL を表す文字列を取り、適切なサブタイプのインスタンスを戻すことができます。次に例を示します。

- 接頭辞が「http://」で始まる場合、SYS.HttpUriType を作成して、接頭辞「http://」を削除してから、そのインスタンスに対する参照を戻します。
- 文字列が接頭辞「/oradb/」で始まるか、または「http://」などの一般的な接頭辞に一致しない場合、SYS.DBUriType インスタンスを作成し、それを戻します。

UriType の新しいサブタイプの登録

UriFactory パッケージは、Oracle で現在サポートされていない他の様々なプロトコルを処理するために、UriType の新しいサブタイプを登録する機能も提供します。たとえば、新しいプロトコル「ecom://」を作成し、そのプロトコルを処理するための UriType のサブタイプを定義して、UriFactory にそのサブタイプを登録できます。登録後は、すべてのファクトリ・メソッドが、接頭辞 ecom を検出するたびに、その新しいサブタイプのインスタンスを生成します。

図 6-3 UriType インスタンスを生成する UriFactory



UriFactory の例 : ecom プロトコルの登録

新しいプロトコル「ecom://」を登録するには、次の作業を実行します。

- プロトコルを処理する型の定義
- UriFactory パッケージへの型の登録

次に例を示します。

```
create table url_tab (urlcol varchar2(20));

-- insert a Http reference
insert into url_tab values ('http://www.oracle.com');

-- insert a DBuri-ref reference
insert into url_tab values ('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]');

-- create a new type to handle a new protocol called ecom://
create type EComUriType under SYS.UriType
(
  overriding member function getClob() return clob,
```

```
  overriding member function getBlob() return blob, -- not supported
  overriding member function getExternalUrl() return varchar2,
  overriding member function getUrl() return varchar2,

  -- MUST NEED THIS for registering with the url handler
  static member function createUri(url in varchar2) return EcomUriType
);
/

-- register a new protocol handler.
begin

  -- register a new handler for ecom:// prefixes. The handler
  -- type name is ECOMURITYPE, schema is SCOTT
  -- Ignore the prefix case, when comparing and also strip the prefix
  -- before calling the createUri function
  SYS.urifactory.registerHandler('ecom://','SCOTT','ECOMURITYPE',
     true,true);
end;
/
insert into url_tab values ('ECOM://company1/company2=22/comp');

-- now use the factory to generate the instances.!
select SYS.urifactory.getUri(urlcol) from url_tab;

-- would now generate
HttpUriType('www.oracle.com'); -- a Httpuritype instance

SYS.DBUriType('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"],null); -- a SYS.DBUriType

EComUriType('company1/company2=22/comp'); -- a EComUriType instance
```

異なる URI 参照を使用する理由

前述のとおり、URI 参照は、異なるプロトコルを実装する様々な導出を持つ抽象クラスです。様々な導出を分離するメリットは、次の 2 つです。

- 列を表すためにサブタイプを選択する場合、その列に対して暗黙的な制約が規定され、特定のプロトコル型のインスタンスのみが含まれます。これは、その列に特定のプロトコルのスペシャライズ・インデックスを実装する場合に有効です。たとえば、DBURI 参照の場合、SQL 問合せを実行するかわりに、直接ディスク・ブロックに移動してデータをフェッチできるスペシャライズ・インデックスを実装できます。
- 関連の型に基づいて、列に異なる制約を規定できます。たとえば HTTP の場合、列に対してプロキシおよびファイアウォールの制約を潜在的に定義して、HTTP を介したすべてのアクセスでプロキシ・サーバーを使用するようにできます。

実装クラスと URI 参照の抽象クラスを分離すると、次のメリットがあります。

- モデリングの改善
- 拡張機能の改善

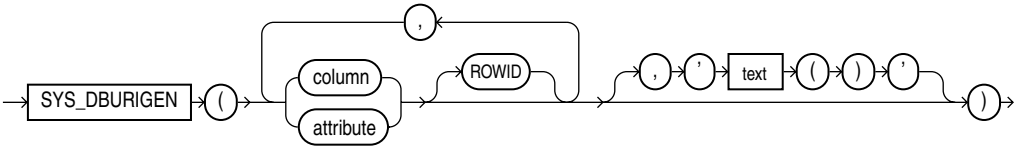
ナビゲーションや索引付けなどのために、独自のプロトコルを実装して、実際にデータベースで URI 参照として処理することができます。

SQL 関数 SYS_DBURIGEN()

DBURI 参照は、コンストラクタ・メソッドまたは UriFactory メソッドへのパス式を指定することによって作成できます。ただし、任意のターゲット列で、これらの DBURI 参照を動的に生成するためのメソッドも必要です。そのため、新しい SQL 関数 `SYS_DBURIGEN()` が導入されています。

図 6-4 に、SYS_DBURIGEN() の構文を示します。

図 6-4 SYS_DBURIGEN() の構文



次の例では、SYS_DBURIGEN() 関数を使用して、サンプル表 hr.employees にある employee_id = 206 の行の email 列に対して、データ型 DBUriType の URL を生成します。

```
SELECT SYS_DBURIGEN(employee_id, email)
   FROM employees
   WHERE employee_id = 206;

SYS_DBURIGEN(EMPLOYEE_ID,EMAIL)(URL, SPARE)
--------------------------------------------------------------------
DBURITYPE('/PUBLIC/EMPLOYEES/ROW[EMPLOYEE_ID = "206"]/EMAIL', NULL)
```

SYS_DBURIGEN() 関数は、引数として 1 つ以上の列または属性、およびオプションで ROWID を取り、特定の列または行オブジェクトに対してデータ型 DBUriType の URL を生成します。次に、その URL を使用して、データベースから XML 文書を取り出せます。また、この関数は、追加のパラメータを取り、ノードのテキスト値が必要かどうかを示します。

参照するすべての列または属性は、同じ表内に存在する必要があります。それらの列または属性は、主キーの役割を果たす必要があります。表の主キーに実際に一致する必要はありませんが、一意の値を参照する必要があります。複数の列を指定する場合、最後の列を除くすべての列はデータベース内の行を識別し、最後に指定する列はその行内の列を識別します。

デフォルトでは、URL はフォーマットされた XML 文書を指します。URL が文書のテキストのみを指すようにするには、オプションの 'text()' を指定します（この XML コンテキストでは、「text」はキーワードであり、構文のプレースホルダではありません）。

列または属性を含む表またはビューが、問合せのコンテキストで指定されたスキーマを含まない場合、Oracle は、その表名またはビュー名をパブリック・シノニムに変換します。

SYS_DBURIGEN() 関数に渡される列または属性は、次の規則に従う必要があります。

- **一意のマッピング :** 列またはユーザー定義型の属性は、元の表またはビューに一意にマップ可能である必要があります。そのため、仮想列は使用できません。唯一の例外は VALUE 演算子および REF 演算子です。列は、TABLE() 副問合せまたはオンライン・ビュー（そのオンライン・ビューが列の名前を変更しない場合）から取得される場合があります。
- **キー列 :** ROWID または一連のキー列のいずれかを指定する必要があります。キー列のリストは、列が結果内の特定の行を一意に識別できるかぎり、表のキーのリストと一致する必要はありません。
- **同一の表 :** SYS_DBURIGEN() 関数で参照されるすべての列は、同じ表またはビューから取得する必要があります。
- **PUBLIC 要素 :** ROWID またはキー列が指す表またはビューに指定されたデータベース・スキーマがない場合、スキーマのかわりに PUBLIC キーワードが使用されます。これは、DBUri がアクセスされたときに、表名に対する現在のバインド（表、シノニムまたはビュー）を使用するという効果があります。

- **TEXT 関数 :** デフォルトでは、DBUri は、結果を含む XML 文書を取り出します。テキスト値のみを取り出すには、関数の最後の引数として 'text()' キーワードを使用します。

  次に例を示します。

  ```
  select SYS_DBURIGEN(empno,ename,'text()') from scott.emp,
  ```

  この構文は、次の形式の URL を生成します。

  ```
  /SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
  ```

- **単一列の引数 :** 単一列の引数がある場合、列は、行を識別するキー列と参照列の両方として使用されます。次に例を示します。

  ```
  select SYS_DBURIGEN(empno) from emp;
  ```

  この構文は、empno をキー列と参照列の両方として使用し、次の形式の URL を生成します。

  ```
  /SCOTT/EMP/ROW[EMPNO=7369]/EMPNO,
  ```

  ここで、行は empno = 7369 です。

SYS_DBURIGEN の例 1 : データベース参照の挿入

```
CREATE TABLE doc_list_tab(docno number primary key, doc_ref SYS.DBUriType);

-- inserts /SCOTT/EMP/ROW[rowid='xxx']/EMPNO
INSERT INTO doc_list_tab(1001,
     (select SYS_DBURIGEN(rowid,empno) from emp where empno = 100);

-- insert a Uri-ref to point to the empname column of emp!
INSERT INTO doc_list_tab
   select empno, SYS_DBURIGEN(empno, ename) from emp));

-- result of the DBURIGEN looks like, /SCOTT/EMP/ROW[EMPNO=7369]/ENAME
```

SYS_DBURIGEN の例 2：部分的な結果の取得

LOB 列のような列の結果を選択する場合、結果の一部のみを取り出し、かわりに列に対する URL を作成することがあります。たとえば、旅行体験記についての Web サイトを考えてみます。表にすべての旅行体験記が格納されている場合、ユーザーは、その表を問い合せ、検索基準に基づいて、関連するすべての体験記を検索します。この場合、問合せ結果のページには、体験記の全文ではなく、最初の 100 文字または話の要点のみを表示し、実際の話の URL を戻す必要があります。

これを行うには、次の手順を実行します。

旅行体験記の表が、次のように定義されているとします。

```
create table travel_story
(
  story_name varchar2(100),
  story clob
);

-- insert some value..!
insert into travel_story values ('Egypt','This is my story of how I spent my time in
Egypt, with the pyramids in full view from my hotel room');
```

旅行体験記の最初の 20 文字のみを戻す関数を作成します。

```
create function charfunc(clobval IN clob ) return varchar2 is
 res varchar2(20);
 amount number := 20;
begin
  dbms_lob.read(clobval,amount,1,res);
  return res;
end;
/
```

旅行体験記の最初の 100 文字のみを選択し、story 列に対する DBURI 参照を戻すビューを作成します。

```
create view travel_view as select story_name, charfunc(story) short_story,
   SYS_DBURIGEN(story_name,story,'text()') story_link
from travel_story;
```

ビューに対する SELECT 文の結果は、次のようになります。

```
select * from travel_view;

STORY_NAME      SHORT_STORY             STORY_LINK
--------------------------------------------------------------------------------
Egypt           This is my story of h

SYS.DBUriType('/PUBLIC/TRAVEL_STORY/ROW[SHORT_STORY='Egypt']/STORY/text()')
```

SYS_DBURIGEN の例 3：URI 参照の取得

DML 文の RETURNING 句では、SYS_DBURIGEN を使用できます。これは、挿入されたオブジェクトの URL を取り出す場合に有効です。たとえば、表 clob_tab について考えてみます。

```
CREATE TABLE clob_tab ( docid number, doc clob);
```

文書を挿入する場合、その文書の URL を取り出して、別の表 uri_tab にその URL を格納する必要がある場合があります。これは、監査または他の目的に有効です。

```
CREATE TABLE uri_tab (docs sys.DBUriType);
```

これは、RETURNING 句を使用して、clob_tab に対する挿入の一部として実行できます。EXECUTE IMMEDIATE 構文を使用して、PL/SQL 内で SYS_DBURI 関数を実行できます。次に例を示します。

```
declare
  ret sys.dburitype;
begin
  -- exucute the insert and get the url
 EXECUTE IMMEDIATE
'insert into clob_tab values (1,''TEMP CLOB TEST'')
 RETURNING SYS_DBURIGEN(docid, doc, ''text()'') INTO :1 '
 RETURNING INTO ret;
 -- insert the url into uri_tab
insert into uri_tab values (ret);
end;
/
```

作成される URL は、次の形式になります。

```
/SCOTT/CLOB_TAB/ROW[DOCID="xxx"]/DOC/text()
```

注意： text() キーワードを最後に追加すると、CLOB 値のみが戻り、CLOB テキストを囲む XML 文書は戻されません。

サーブレットを使用したブラウザから DBURI 参照へのアクセス

DBURI 参照はデータベース参照です。次の方法を使用して、DBURI 参照をブラウザまたは他の Web サーバーからアクセスできるように拡張できます。

- DBURI 参照 URL を実行し値を戻すことができる、OSE または Apache JServ モジュールで実行するサーブレットを作成します。
- 同様のタスクを実行するために提供されているデフォルトのサーブレットを使用します。このサーブレットは、Java ネイティブ・インタフェース（JNI）を使用してサーバーにリンクされた DBUri を解釈するコードと通信するため、OSE でのみ実行可能です。

oracle.xml.dburi.OraDbUriServlet() サーブレット・メカニズム

前述の方法を実行するため、サーブレット `OraDBUriServlet()` クラスは Oracle サーブレット・エンジンで実行します。このサーブレットは、パス式で、サーブレット名に続いて DBURI 参照を取り、DBUri が指す文書を出力ストリームに出力します。このサーブレットは、次のいずれかを実行できます。

- 文書の MIME タイプを自動的に生成します。今回のリリースでは、「text/xml」および「text/plain」のみがサポートされています。DBUri の最後が text() 関数の場合は、「text/plain」の MIME タイプが使用されます。それ以外の場合は、XML 文書は「text/xml」の MIME タイプを使用して生成されます。
- サーブレットに対するコンテンツ型の引数を使用して、MIME 値をオーバーライドします。

たとえば、従業員表の empno 列を取り出すには、次のいずれかのような URL を記述できます。

- http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text() （text/plain のコンテンツ型を生成します）
- http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME（text/xml のコンテンツ型を生成します）

ここで、マシン machine.oracle.com は OSE を実行しており、Web サービスがポート 8080 で要求をリスニングしています。oradb は、OraDbUriServlet にマップする仮想パスです。

注意： HTTP アクセスでは、「]」、「[」、「&」、「|」などの特殊文字は、%xxx フォーマットを使用してエスケープする必要があります。ここで、xxx はその特殊文字の ASCII コードを指す 10 進数の数字です。エスケープ済の URL を取得するには、UriType 型の getExternalUrl() 関数を使用します。

OraDBUriServlet セキュリティ

OraDbServlet を公開する場合は、次のセキュリティ問題を考慮します。

- DBUser レルムでの OraDBServlet の公開
- レルム以外での OraDBServlet の公開

DBUser レルムでの OraDBServlet の公開

提供されている OraDBUri のサーブレットは、DBUser レルムで公開される場合、認証済ユーザーと自動的に切り替わり、その認証済ユーザーとして問合せを実行します。

次に例を示します。

- サーブレットが DBUser レルムで SYS として公開される場合、問合せは常にログイン・ユーザーとして実行されます。
- サーブレットが Rdbms レルムで公開される場合、サーブレットは、認証済ユーザーとしてではなく、発行者として実行されます。

サーブレットは、特に SYS スキーマ内で公開する場合、またはデータ・アクセスに対するセキュリティを施行する場合は、DBUser レルム以外のレルムで実行されないことに注意してください。

レルム以外での OraDBServlet の公開

レルム以外でサーブレットを公開する場合、ユーザーは、ユーザー名やパスワードを入力することなく直接サーブレットにアクセスできます。サーブレットは、発行ユーザーの権限で実行され、サーブレットのユーザーは、発行ユーザーが参照権限を取得しているすべてのデータにアクセスできます。

これは、権限の限定されたユーザーが所有するデータ（文書、デモ、サンプル・スキーマなど）に、すべてのユーザーがアクセスできるようにする場合に有効です。

たとえば、製品または企業の業務に関連するすべての有用な文書を所有する HELP ユーザーを作成し、そのスキーマ内でサーブレットを公開して、すべてのユーザーにすべての文書に対するアクセス権を付与することができます。

注意： DBUser レルム以外のレルムでサーブレットを公開する場合、問合せは発行ユーザーで実行されます。そのため、サーブレットの公開は DBUser レルムでのみ行うか、または他のレルム内で公開する場合は発行ユーザーの権限を制限するように注意してください。6-28 ページの「DBUri サーブレットの例 1: DBUriServer Web サービスの新規作成（tkxmsrv.ssh）」以降の例を参照してください。

SYS ユーザーに対しては、DBUser レルム以外のレルムでサーブレットを公開しないでください。DBUser レルム以外で公開した場合、サーブレットにアクセスするユーザーが、データベース内のすべてのデータに対するアクセス権を取得します。

OraDBUriServlet のインストール

OraDBUriServlet は、$ORACLE_HOME の rdbms/jlib/ ディレクトリにある jar ファイル OraDbUri.jar にあります。サーブレットをインストールするには、次のタスクを実行します。

サーブレットに対応する Java クラスは、SYS ユーザーでインストールされており、すべてのユーザーに実行権限が付与されているため、最初の 2 つの手順はスキップできます。ただし、サーブレットに対応する Java クラスを自スキーマ内に常駐させる場合、次の手順を実行します。

1. SQL*Plus を実行し、サーブレットをインストールするスキーマに接続します。SYS スキーマにインストールすることをお薦めします。
2. admin ディレクトリ内の inituris.sql スクリプトを実行し、jar ファイルを目的のスキーマにロードします。また、loadjava を使用して jar ファイルを目的のスキーマにロードすることもできます。dbms_java インタフェースの実行には、ファイルにアクセスするための IO 権限が必要です。
3. sess_sh を使用して、admin ポートでデータベースに接続します。詳細は、『Oracle9*i* Servlet Engine Developer's Guide』を参照してください。
4. 適切な Web サービスおよびレルムを設定します。サーブレット・デモに、DBUser レルムの設定例があります。
5. 必要なレルムおよびコンテキストを設定したら、サーブレットを公開します。次に、sess_sh シェルの例を示します。

```
publishservlet -virtualpath /oradb/*  -stateless /webdomains/default/contexts
DBUriServlet SYS:oracle.xml.uri.OraDbUriServlet
```

   - /oradb/* の使用に注意してください。「*」は、oradb 以降のすべてのパスが同じサーブレットにマップされることを示すために必要です。oradb は仮想パスとして公開されます。ここで、自スキーマ内にサーブレットをインストールしている場合、キーワード「SYS:」をサーブレットがインストールされているスキーマ名に変更します。
   - stateless パラメータは、サーブレット自体がステートレスであるため、同じ接続を使用してサーブレットを起動できることを示します。
   - /webdomains/default/contexts は、サーブレットが公開されているコンテキストです。コンテキストには、定義済の仮想パスを含む場合があることに注意してください。この場合、サーブレットを公開する仮想パスは、そのパスの下位になります。たとえば、この例のコンテキストの仮想パスが /servlets である場合、DBUriServlet は、パス /servlets/oradb/* によってアクセスできます。

- DBUriServlet はサーブレット名です。
- oracle.xml.uri.OraDbUriServlet は、SYS スキーマで使用するクラス名です。スキーマ名は、jar ファイルがロードされたスキーマに変更します。
- これで、mod_ose モジュールを使用して、Apache サーバーを介してサーブレットに直接アクセスできます。または、Web サービスがリスニングしているポートを介して、OSE に直接アクセスできます。

DBUri サーブレットの例 1: DBUriServer Web サービスの新規作成（tkxmsrv.ssh）

スクリプト tkxmsrv.ssh は、（スクリプト内にハードコードされた）ポート 8088 に DBUriServer の Web サービスを作成し、パラメータとしてスクリプトに渡された ID のユーザーに対してサービスの所有権を割り当てます。このスクリプトは、次のすべてのスクリプトを実行する前に実行する必要があります。このスクリプトが作成された環境では、AURORA_AWS_ADMIN_PORT は 8080 に設定されています。そのため、管理アクティビティを行うには、AURORA_AWS_ADMIN_PORT に接続する必要があります。スクリプトのこの部分を変更することによって、すべてのポートに対して Web サービスを構成できます。

```
#   USAGE :
#   sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
<user-name>"
#
# This script does the following :
#   1. Creates dburiServer service with -root /dburidomain.
#   2. Assigns ownership to user passed as parameter

destroywebdomain dburidomain
destroyservice   dburiServer

echo "Creating dburiService ..."
createwebservice -root /dburidomain dburiServer

#The following line requires this script to be run as SYS
rmendpoint -force dburiServer main
addendpoint -port 8088 -register dburiServer main

chown -R &1 /dburidomain

echo "Service creation complete"
```

DBUri サーブレットの例 2: DBUridomain の作成 - OraDbUriServlet の公開

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。デフォルト・コンテキストを使用して、このドメインで URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを SYS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsys.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsys.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Publishes the OraDbUriServlet servlet under SYS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Publishing servlet under default context .."

publishservlet /dburidomain/contexts/default -virtualpath /norealm/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 3: SYS での OraDbUriServlet の公開（tkxmsysd.ssh）

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。DBUser レルムにマップされたデフォルト・コンテキストを使用して、このドメイン内で URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを SYS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysd.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysd.ssh"
```

```
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates and protects dburirealm
#   3. Publishes the OraDbUriServlet servlet under SYS.
#   4. Grant permission to execute the servlet to SCOTT and ADAMS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain
#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>
echo "Domain creation complete"

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/default -add /oradb/* -scheme basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default -name
SYS -path /oradb/* + get,post
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default -name
SCOTT -path /oradb/* + get,post
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default -name
ADAMS -path /oradb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under default context .."

publishservlet /dburidomain/contexts/default -virtualpath /oradb/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

chmod +x SCOTT /dburidomain/contexts/default/named_servlets/DBUriServlet

chmod +x ADAMS /dburidomain/contexts/default/named_servlets/DBUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 4: ADAMS での OraDbUriServlet の公開

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。uritests コンテキストを使用して、このドメイン内で URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを SYS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadam.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadam.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates uritests context.
#   3. Publishes the OraDbUriServlet servlet under ADAMS using the class
#      under SYS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 5: SCOTT での OraDbUriServlet の公開（tkxmsctd.ssh）

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。DBUser レルムにマップされた uritests コンテキストを使用して、このドメイン内で URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを SYS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SCOTT"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsctd.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u scott/tiger -c "@tkxmsctd.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates uritests context.
#   3. Creates and protects dburirealm
#   4. Publishes the OraDbUriServlet servlet under SCOTT.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context and granting ownership to SCOTT ..."
createcontext -virtualpath /scottcon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/uritests -add /scottdb/* -scheme basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/uritests -name
PUBLIC -path /scottdb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under uritests context"
```

```
publishservlet /dburidomain/contexts/uritests -virtualpath /scottdb/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 6: dburirealm の作成およびマッピング - OraDbUriServlet

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。RDBMS レルムにマップされたデフォルト・コンテキストを使用して、このドメインで URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを SYS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysr.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysr.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates and protects dburirealm using RDBMS realm mapping.
#   3. Publishes the OraDbUriServlet servlet under SYS.
#   4. Grant permission to execute the servlet to SCOTT and ADAMS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain
#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>
echo "Domain creation complete"

echo "Creating and protecting dburirealm "

realm publish -w /dburidomain -r dburirealm -type rdbms
realm publish -w /dburidomain -add dburirealm -type rdbms
# create a user in the realm
realm user -w /dburidomain -realm dburirealm -add alex -p welcome
# create a group in the realm
realm group -w /dburidomain -realm dburirealm -add uriGroup -p welcome
# add 'alex' to the 'uriGroup'
realm parent -w /dburidomain -realm dburirealm -group uriGroup -add alex
# Allow 'uriGroup' to execute http requests with the GET,POST methods
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default -name
```

```
uriGroup -path /rdbrealm/* + get,post
# protect the resource '/rdbrealm'
realm map -s /dburidomain/contexts/default -add /rdbrealm/* -scheme Basic:dburirealm

echo "Realm creation complete"

echo "Publishing servlet under default context .."

publishservlet /dburidomain/contexts/default -virtualpath /rdbrealm/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

chmod +x SCOTT /dburidomain/contexts/default/named_servlets/DBUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 7: ADAMS スキーマでの OraDbUriServlet の公開

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。uritests コンテキストを使用して、このドメインで URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを ADAMS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadmn.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadmn.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates uritests context.
#   3. Publishes the OraDbUriServlet servlet under ADAMS using the class
#      under ADAMS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
```

```
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/* DBUriServlet
ADAMS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

DBUri サーブレットの例 8: ADAMS スキーマでの OraDbUriServlet の公開

このスクリプトは、tkxmsrv.ssh によって作成されたサービスに Web ドメインを作成します。DBUser レルムにマップされた uritests コンテキストを使用して、このドメインで URI サーブレットを公開します。このスクリプトを実行する前に、URI サーブレット・クラスを ADAMS スキーマ内にロードする必要があります。

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadmd.ssh"

#   USAGE :
#   sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadmd.ssh"
#   tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
#   1. Creates webdomain /dburidomain.
#   2. Creates uritests context.
#   3. Creates and protects dburirealm
#   4. Publishes the OraDbUriServlet servlet under ADAMS using the class
#      under ADAMS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context and granting ownership to ADAMS ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>
```

```
echo "Domain creation complete"

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/uritests -add /adamsdb/* -scheme basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/uritests -name
PUBLIC -path /adamsdb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/* DBUriServlet
ADAMS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

DBURI 参照を処理する UriFactory パッケージの構成

UriFactory（6-17 ページの「UriFactory パッケージ」を参照）は、任意の URL で、UriType の適切なサブタイプを生成して、特定のプロトコルを処理します。HTTP の URL の場合、UriFactory は HttpUriType のインスタンスを作成します。ただし、HTTP の URL が、DBURI 参照のメカニズムを使用してデータベース内を指している場合は、その URL をデータベースの DBUriType インスタンスとして格納および処理する方がより効率的です。

サーバー内では、HTTP URL のメカニズムを使用するより、DBUriType のインスタンスを使用して DBURI 参照を直接処理する方が常により効率的です。これは、HTTP URL メカニズムでは、JavaVM レイヤー、サーブレット・レイヤーおよび Web サーバー・レイヤーを介した追加のデータ転送を伴い、追加の文字変換が発生する可能性があるためです。

`http://machine-name/servlets/oradb/` のようなすべての URL を OraDBUriServlet で処理するために、OraDBUriServlet をインストールして DBURI 参照を処理する場合、その接頭辞を使用して、HttpUriType のかわりに DBUriType のインスタンスを作成するように UriFactory を構成できます。

```
begin
 -- register a new handler for the dburi prefix..
urifactory.registerHandler('http://machine-name/servlets/oradb'
        ,'SYS','DBURITYPE', true,true);
end;
/
```

一度セッションでこのブロックを実行すると、そのセッションで `UriFactory.getUri()` をコールするたびに、その接頭辞を含む HTTP の URL に対して自動的に DBUriType のインスタンスが作成されます。この方法では、真の DBUri URL を DBUriType のインスタンスに変換して、処理をより効率的にできます。

7

XSU

この章の内容は次のとおりです。

XSU の概要

XML は、データ交換のための形式として確立されています。一方で、大量のビジネス・データがオブジェクト・リレーショナル・データベースに保存されています。そのため、このリレーショナル・データを XML に変換する機能が必要です。

XSU を使用すると、次に示すようにリレーショナル・データを XML に変換できます。

- XSU は、オブジェクト・リレーショナル・データベースの表またはビューから取り出したデータを XML に変換できます。
- XSU は、XML 文書からデータを抽出し、正規マッピングを使用して、表またはビューの適切な列または属性にデータを挿入できます。
- XSU は、XML 文書からデータを抽出し、このデータを適用して適切な列または属性の値を更新または削除できます。

データベースからの XML の生成

たとえば、XML の生成側で、`SELECT * FROM emp` という問合せが発行されると、XSU がデータベースに問い合せ、結果として次の XML 文書を戻します。

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

データベースへの XML の格納

一方、前述の XML 文書では、XSU はこの XML 文書からデータを抽出し、データベース内の `scott.emp` 表に挿入します。

XSU の機能へのアクセス

次の方法で XSU の機能にアクセスできます。

- Java API
- PL/SQL API
- Java コマンドラインのフロントエンド

参照： 次の章およびマニュアルを参照してください。

- 付録 H「XSU の仕様および早見表」
- 『Oracle9*i* XML リファレンス』

XSU の機能

XSU は次のタスクを実行します。

- SQL 問合せから XML 文書を生成します。XSU は、Oracle データベース・サーバーがサポートするすべてのデータ型をサポートします。
- DTD を動的に生成します。
- 生成中、`ROW` 要素のデフォルト・タグ名の変更など、単純な変換を実行します。XSLT を登録して、生成した XML 文書に必要に応じて適用できます。
- XML 文書を、文書の文字列または DOM 表現で生成します。
- データベースの表またはビューに XML を挿入します。特定の XML 文書の、データベース・オブジェクトのレコードを更新または削除できます。
- ネストした複雑な XML 文書を簡単に生成できます。XSU は、フラットな表上にオブジェクト・ビューを作成し、これらのビューを問い合せることによって、ネストした複雑な XML 文書をリレーショナル表に格納できます。オブジェクト・ビューでは、Oracle8*i* および Oracle のオブジェクト・リレーショナル機能を使用して、既存のリレーショナル・データから構造化データを作成できます。

XSU の Oracle 機能

Oracle では、XSU は次のタスクも実行できます。

- 任意の SQL 問合せにおける XML Schema の生成。
- SAX2 コールバックのストリームとしての XML の生成。
- 生成中の XML 属性のサポート。これによって、特定の列または列のグループが XML 要素ではなく XML 属性にマップされるように、簡単に指定できます。

- SQL 識別子から XML 識別子へのエスケープ。列名が有効な XML タグ名ではない場合があります。これを回避するには、すべての列名に別名を付けるか、またはタグをエスケープします。

注意： Oracle では、PL/SQL パッケージ DBMS_XMLGen が提供されています。このパッケージは、以前は DBMS_XMLQuery で使用可能だった機能を提供します。DBMS_XMLGen はデータベース・コードに組み込まれているため、パフォーマンスが向上します。

XSU の依存性およびインストール

依存性

XSU には、次のコンポーネントが必要です。

- **Database Connectivity - JDBC ドライバ。**XSU は、どの JDBC ドライバでも動作しますが、Oracle の JDBC ドライバ用に最適化されています。オラクル社は、Oracle 以外のデータベースで実行されている XSU に対していかなる保証およびサポートも行いません。
- **XML パーサー - Oracle XML Parser。**Oracle XML Parser は、Oracle8*i* および Oracle に含まれています。また、XSU を OTN の Web サイトから、XSU インストールの一部としてダウンロードできます。

インストール

XSU は、Oracle8*i*（リリース 8.1.7 以上）および Oracle にパッケージ化されています。XSU は、次の 3 つのファイルで構成されています。

- $ORACLE_HOME/rdbms/jlib/xsu12.jar - XSU を構成するすべての Java クラスを含みます。xsu12 には、JDK1.2.x および JDBC2.x が必要です。これは、Oracle にロードされた XSU のバージョンです。
- $ORACLE_HOME/rdbms/jlib/xsu111.jar - xsu12.jar と同じクラスを含みます。ただし、xsu111 には、JDK1.1.x および JDBC1.x が必要です。
- $ORACLE_HOME/rdbms/admin/dbmsxsu.sql - XSU の PL/SQL API を構築する SQL スクリプトです。xsu12.jar は、dbmsxsu.sql が実行される前にデータベースにロードする必要があります。

Oracle Installer は、デフォルトで XSU を前述の指定場所にあるファイル・システムにインストールします。さらに、データベースにもロードします。

最初のインストール時に XSU をインストールしないと選択した場合は、後で XSU をインストールできます。ただし、インストールは少し複雑になります。後で XSU をインストールするには、まずシステム上に XSU および XSU 固有のコンポーネントをインストールします。これを行うには、Oracle Installer を使用します。次の手順に従います。

1. XML Parser for Java をデータベースにロードしていない場合は、`$ORACLE_HOME/xdk/lib` を選択します。このディレクトリにある `xmlparserv2.jar` を、データベースにロードする必要があります。この手順の詳細は、『Oracle9*i* Java Stored Procedures Developer's Guide』の「Loading Java Classes」を参照してください。
2. `$ORACLE_HOME/admin` を選択し、`catxsu.sql` を実行します。

注意： XSU は、OTN の Web サイト `http://otn.oracle.com/tech/xml` からもダウンロードできます。XSU の最新版は、この Web サイトで確認してください。

XSU および広がる実行可能場所

XSU は Java で作成されており、Java をサポートするすべての層で実行できます。

データベース内での XSU

XSU を構成する Java クラスは、Java 対応の Oracle8*i* 以上のサーバーにロードできます。また、XSU は、XSU の Java API を PL/SQL に公開して PL/SQL API を作成した PL/SQL ラッパーも含みます。このため、次の作業ができます。

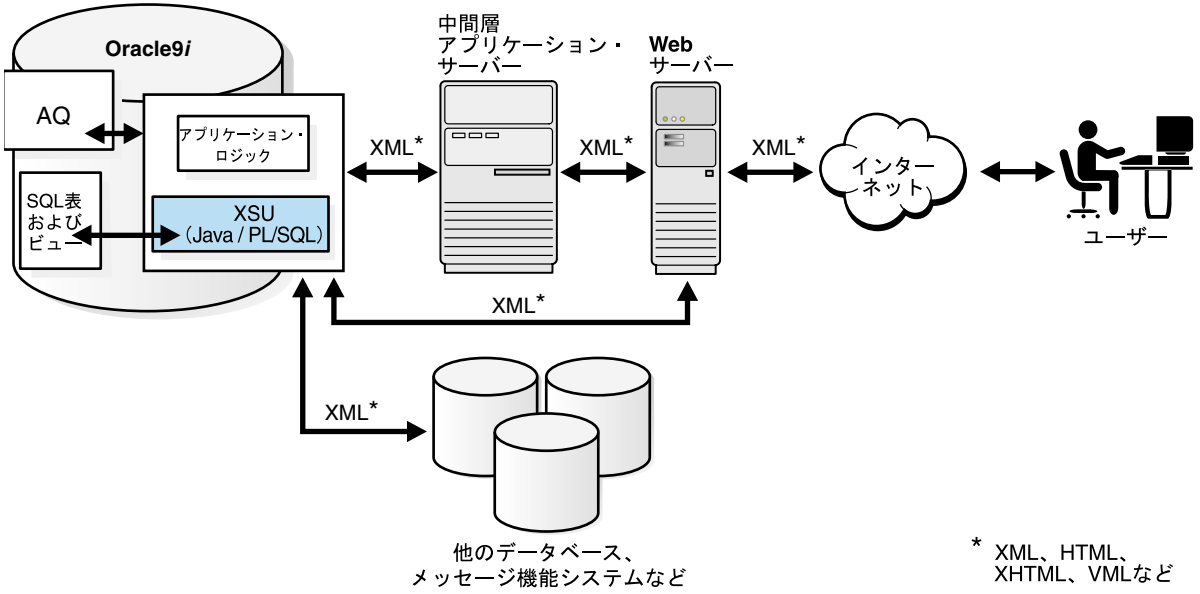
- データベース内で実行して XSU の Java API に直接アクセスする新しい Java アプリケーションを作成します。
- PL/SQL API を介して XSU にアクセスする PL/SQL アプリケーションを作成します。
- SQL を介して直接 XSU の機能にアクセスします。

注意： データベース内で Java をロードおよび実行するには、Java 対応の Oracle8*i* 以上のサーバーが必要です。

図 7-1 に、このようなシステム用の一般的なアーキテクチャを示します。データベース内で実行する XSU が生成した XML は、データベースのアドバンスト・キュー内に置いて、他のシステムまたはクライアントにキューさせることができます。この XML は、データベース内のストアド・プロシージャから使用するか、Web サーバまたはアプリケーション・サーバーを介して外部に送信できます。

注意： 図 7-1 では、すべての矢印が両方向です。XSU は、データの保存のみでなく生成も行うため、データベース内で実行する XSU は様々なソースからデータを取得したり、そのデータを適切なデータベース表に戻すことができます。

図 7-1 データベース内での XSU の実行



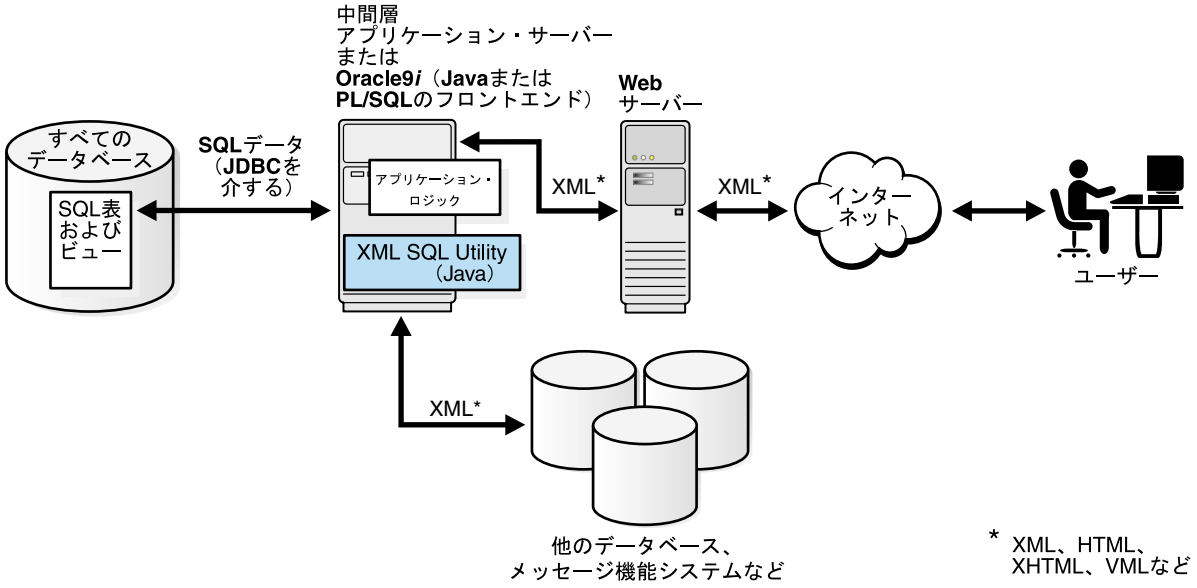
中間層内での XSU

アプリケーション・アーキテクチャによっては、データベースとは別の中間層にあるアプリケーション・サーバーを使用する必要があるものもあります。このアプリケーション層には、Oracle データベース、Oracle Application Server、Java プログラムをサポートするサード・パーティ製のアプリケーション・サーバーなどがあります。

中間層で、SQL 問合せまたは結果セットから XML を生成する必要がある場合もあります。たとえば、中間層で異なる JDBC データ・ソースを統合するとします。この場合、中間層に XSU をインストールし、Java プログラムで Java API を介して XSU を使用します。

図 7-2 に、XSU を中間層で実行する一般的なアーキテクチャを示します。中間層で、JDBC ソースのデータは XSU によって XML に変換され、Web サーバーまたは他のシステムに送信されます。ここでもすべてのプロセスは両方向であり、データは XSU を使用して JDBC ソース（データベース表またはビュー）に戻すことができます。Oracle データベース自体がアプリケーション・サーバーとして使用されている場合は、Java のかわりに PL/SQL のフロントエンドも使用できます。

図 7-2 中間層内での XSU の実行



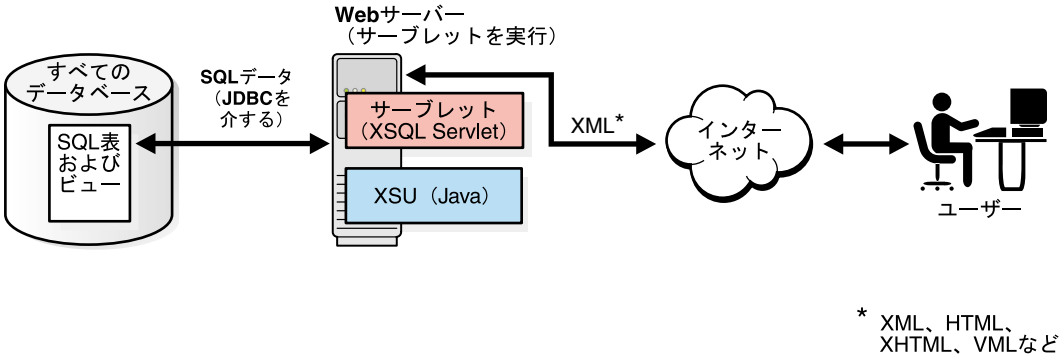
Web サーバー内での XSU

Web サーバーが Java サーブレットをサポートする場合は、XSU を Web サーバー内で実行できます。この場合、XSU を使用してタスクを実行する Java サーブレットを作成できます。

XSQL Servlet はこれを行います。XSQL Servlet は、Oracle が提供する標準のサーブレットです。XSQL Servlet は XSU の最上位に構築され、XSU の機能にテンプレート形式のインタフェースを提供します。Web サーバーで XML 処理することが目的である場合、サーブレットの複雑なプログラミングを省略するために、XSQL Servlet を使用します。

> **参照：** XSQL Servlet の使用については、第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

図 7-3 Web サーバー内での XSU の実行



クライアント層内での XSU

XSU をクライアント・システムにインストールし、XSU を使用する Java プログラムを作成することもできます。コマンドラインのフロントエンドを介して、直接 XSU を使用することもできます。

SQL から XML および XML から SQL へのマッピングの手引き

前述したとおり、XSU は、オブジェクト・リレーショナル・データベースの表またはビューから取り出したデータを XML に変換します。XSU は、XML 文書からデータを抽出し、指定されたマッピングを使用して、データベースの表またはビューの適切な列または属性にデータを挿入することもできます。この項では、SQL から XML または XML から SQL への正規マッピングまたは変換について説明します。

SQL から XML へのデフォルトのマッピング

emp 表について考えてみます。

```
CREATE TABLE emp
(
   EMPNO NUMBER,
   ENAME VARCHAR2(20),
   JOB VARCHAR2(20),
   MGR  NUMBER,
   HIREDATE DATE,
   SAL NUMBER,
   DEPTNO NUMBER
);
```

XSU は、`select * from emp` という問合せを指定して、次の XML 文書を生成できます。

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

生成された XML では、SQL 問合せによって戻された行が ROWSET タグで囲まれ、<ROWSET> 要素を構成します。この要素は、生成された XML 文書のルート要素でもあります。

- <ROWSET> 要素は、1 つ以上の <ROW> 要素を含みます。
- 各 <ROW> 要素には、戻されたデータベース表の行の 1 つからのデータが含まれています。各 <ROW> 要素には、SQL 問合せの SELECT リストで指定されたデータベースの列の名前および内容を持つ、1 つ以上の要素が含まれています。
- データベースの列に対応するこれらの要素は、列からのデータを含みます。

オブジェクト・リレーショナル・スキーマでの SQL から XML へのマッピング

次に、オブジェクト・リレーショナル・スキーマでのマッピングについて説明します。次に示す AddressType 型について考えてみます。これは、属性がすべてスカラー型のオブジェクト型で、次のように作成します。

```
CREATE TYPE AddressType AS OBJECT (
   STREET VARCHAR2(20),
   CITY   VARCHAR2(20),
   STATE  CHAR(2),
   ZIP    VARCHAR2(10)
);
/
```

次に示す EmplyeeType 型もオブジェクト型ですが、それ自体が AddressType というオブジェクト型である EMPADDR 属性を持ちます。EmployeeType は次のように作成します。

```
CREATE TYPE EmployeeType AS OBJECT
(
  EMPNO NUMBER,
  ENAME VARCHAR2(20),
  SALARY NUMBER,
  EMPADDR AddressType
);
/
```

次に示す EmployeeListType 型は、EmployeeType というオブジェクト型の要素のコレクション型です。EmployeeListType は次のように作成します。

```
CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;
/
```

最後に、dept は、オブジェクト型 AddressType の列およびコレクション型 EmployeeListType の列を含む表です。

```
CREATE TABLE dept
(
  DEPTNO NUMBER,
  DEPTNAME VARCHAR2(20),
  DEPTADDR AddressType,
  EMPLIST  EmployeeListType
)
NESTED TABLE EMPLIST STORE AS EMPLIST_TABLE;
```

dept 表に有効な値が格納されているとします。問合せ select * from dept に対して、XSU は次の XML 文書を生成します。

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>94065</ZIP>
    </DEPTADDR>
    <EMPLIST>
      <EMPLIST_ITEM num="1">
         <EMPNO>7369</EMPNO>
         <ENAME>John</ENAME>
         <SALARY>10000</SALARY>
         <EMPADDR>
           <STREET>300 Embarcadero</STREET>
           <CITY>Palo Alto</CITY>
           <STATE>CA</STATE>
           <ZIP>94056</ZIP>
         </EMPADDR>
      </EMPLIST_ITEM>
       <!-- additional employee types within the employee list -->
    </EMPLIST>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

最後の例では、マッピングは正規マッピングで、<ROWSET> は、列に対応する要素を含む <ROW> を含みます。前述したとおり、スカラー型の列に対応する要素は、単に列からのデータを含みます。

複雑な型の列から XML へのマッピング

複雑な型の列に対応する要素の場合、マッピングはより複雑になります。たとえば、<DEPTADDR> は、オブジェクト型 ADDRESS の DEPTADDR 列に対応しています。したがって、<DEPTADDR> は、ADDRESS 型で指定された属性に対応するサブ要素を含みます。これらのサブ要素は、対応する属性の型が単純か複雑かによって、それ自体がデータやサブ要素を含む場合があります。

XML へのコレクションのマッピング

データベースのコレクションに対応する要素を処理する場合は、マッピングの方法が異なります。たとえば、<EMPLIST> 要素は、EmployeeListType コレクション型の EMPLIST 列に対応します。したがって、<EMPLIST> 要素は、それぞれがコレクションの要素の 1 つに対応する <EMPLIST_ITEM> 要素のリストを含みます。

このマッピングについては、次のことにも注意してください。

- <ROW> 要素は、カーディナリティ属性 num を含みます。
- 特定の列または属性の値が NULL の場合、その行では、対応する XML 要素がすべて省略されます。
- 最上位のスカラー列の名前はアットマーク（@）で始まる場合、特定の列が XML 要素ではなく XML 属性にマップされます。

生成された XML のカスタマイズ : SQL から XML へのマッピング

多くの場合、特定の構造を持つ XML を生成する必要があります。生成された XML 文書のデフォルトの構造が要求する構造と異なる場合があるため、このプロセスにはある程度の柔軟性が必要になります。次のいずれかの方法を使用して、生成された XML 文書の構造をカスタマイズできます。

- 「ソースのカスタマイズ」
- 「マッピングのカスタマイズ」
- 「生成後のカスタマイズ」

ソースのカスタマイズ

ソースのカスタマイズは、問合せまたはデータベース・スキーマを変更することによって行います。最も単純で強力なソースのカスタマイズは次のとおりです。

- **データベース・スキーマで**、任意の XML 文書構造にマップするオブジェクト・リレーショナル・ビューを作成します。
- **問合せで、**次のことを行います。
  - カーソル副問合せまたはキャスト多重集合構造を使用して、フラットなスキーマから取り出した XML 文書でネストを実現します。
  - 列名または属性名に別名を付けて、任意の XML 要素を取得します。
  - アットマーク（@）で始まる識別子を使用して、最上位のスカラー型の列に別名を付け、これらの列を XML 要素ではなく XML 属性にマップします。たとえば、`select empno as "@empno",... from emp` は、<ROW> 要素が EMPNO 属性を持つ XML 文書になります。

マッピングのカスタマイズ

XSU を使用すると、SQL データを XML に変換するために使用するマッピングを変更できます。SQL から XML へのマッピングでは、次のように変更できます。

- <ROWSET> タグを変更または省略します。
- <ROW> タグを変更または省略します。
- num 属性を変更または省略します。これは、<ROW> 要素のカーディナリティ属性です。
- 生成された XML 要素名に大文字 / 小文字を指定します。
- コレクションの要素に対応する XML 要素が、カーディナリティ属性を持つように指定します。
- XML 文書の日付書式を指定します。
- XML 文書の NULL 値は、要素を省略せず、NULL 属性を使用して表すように指定します。

生成後のカスタマイズ

前述の方法で希望のカスタマイズが行えない場合、XSLT を記述して、XSU に登録できます。XSU に登録された XSLT がある場合、XSU は生成するすべての XML に XSLT を適用できます。

XML から SQL へのデフォルトのマッピング

XML から SQL へのマッピングは、単純に SQL から XML へのマッピングの逆の操作です。

> **参照：** 7-9 ページの「SQL から XML へのデフォルトのマッピング」を参照してください。

XML から SQL へのマッピングを SQL から XML へのマッピングと比較すると、次のような相違点があります。

- XML から SQL へのマッピングでは、XML 属性は無視されます。したがって、XML 属性は実際には SQL にマップされません。
- SQL から XML へのマッピングでは、SQL 問合せによって作成された結果セットから XML にマッピングされます。この場合、問合せは複数のデータベース表またはビューに対して実行できます。単一の結果セットが形成され、XML に変換されます。これは、XML から SQL へのマッピングでは異なります。XML から SQL へのマッピングでは、次のようになります。
  - 1 つの XML 文書を複数の表またはビューに挿入するには、ターゲットのスキーマにオブジェクト・リレーショナル・ビューを作成する必要があります。
  - このビューが更新不可能な場合は、INSTEAD-OF-INSERT トリガーを使用して対処します。

XML 文書がターゲットのデータベース・スキーマに完全にマップされない場合は、次の 3 つの方法で対処します。

- **ターゲットを変更します。**ターゲットのスキーマにオブジェクト・リレーショナル・ビューを作成し、そのビューを新しいターゲットにします。
- **XML 文書を変更します。**XSLT を使用して XML 文書を変換します。XSLT は XSU に登録できるため、受信する XML は、マッピングを試行する前に自動的に変換されます。ただし、この方法は、あまりお薦めできません。
- **XSU の XML から SQL へのマッピングを変更します。**XML 要素とデータベースの列または属性の一致で大文字 / 小文字を区別するように、XSU に対して指示できます。
  - デフォルト（ROW）でない場合、データベースの行に対応する要素の名前を使用するように、XSU に指示できます。
  - XSU に、XML 文書の日付を解析する場合に使用する日付書式を指示できます。

XSU の動作方法

この項では、次のタスクを実行する場合の XSU の動作方法について説明します。

- XSU を使用した選択（7-14 ページ）
- XSU を使用した挿入（7-15 ページ）
- XSU を使用した更新（7-15 ページ）
- XSU を使用した削除（7-16 ページ）

XSU を使用した選択

XSU の生成は単純です。SQL 問合せが実行され、データベースから結果セットが取り出されます。結果セットに関するメタデータが取得され、分析されます。7-9 ページの「SQL から XML へのデフォルトのマッピング」で説明するマッピングを使用して、SQL の結果セットが処理され、XML 文書に変換されます。

XSU を使用した挿入

XML 文書のコンテンツを特定の表またはビューに挿入するには、まず XSU でターゲットの表またはビューに関するメタデータを取り出します。そのメタデータに基づいて、XSU が SQL INSERT 文を生成します。XSU が XML 文書からデータを抽出し、適切な列または属性にバインドします。最後に、この文を実行します。

たとえば、ターゲット表が dept で、dept から XML 文書が生成されるとします。

> **参照：** 7-9 ページの「SQL から XML へのデフォルトのマッピング」を参照してください。

XSU は、次のような INSERT 文を生成します。

```
INSERT INTO Dept (DEPTNO, DEPTNAME, DEPTADDR, EMPLIST) VALUES (?,?,?,?)
```

次に、XSU は XML 文書を解析し、各レコードについて、適切な列または属性に適切な値をバインドし、文を実行します。

```
DEPTNO <- 100
DEPTNAME <- SPORTS
DEPTADDR <- AddressType('100 Redwood Shores Pkwy','Redwood Shores',
                        'CA','94065')

EMPLIST <- EmployeeListType(EmployeeType(7369,'John',100000,
             AddressType('300 Embarcadero','Palo Alto','CA','94056'),...)
```

挿入処理は、バッチで挿入およびコミットするように最適化できます。バッチ処理の詳細は、7-36 ページの「XSU を使用した挿入処理（Java API）」を参照してください。

XSU を使用した更新

更新および削除は、データベース表内の 2 つ以上の行に影響するという点で挿入と異なります。挿入では、表にトリガーまたは制約がない場合は、XML 文書の 1 つの ROW 要素の影響を受けるのは表内で最大 1 つの行のみです。

一方、更新および削除では、一致する列が表内のキー列でない場合は、XML 要素が 2 つ以上の行と一致する場合があります。更新の場合は、更新する行を識別するために XSU が必要とするキー列のリストを提供する必要があります。たとえば、DEPTNAME を Sports ではなく SportsDept に更新するには、次のような XML 文書を記述します。

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>SportsDept</DEPTNAME>
  </ROW>
</ROWSET>
```

キー列として DEPTNO を指定します。これによって、次の UPDATE 文が生成されます。

```
UPDATE DEPT SET DEPTNAME = ? WHERE DEPTNO = ?
```

その後、次のように値をバインドします。

```
DEPTNO <- 100
DEPTNAME <- SportsDept
```

更新の場合は、XML 文書にあるすべての要素を更新するのではなく、列の集合のみを更新するように選択できます。7-38 ページの「XSU を使用した更新処理（Java API）」を参照してください。

XSU を使用した削除

削除の場合は、削除する行を識別するために、キー列の集合を指定するように選択できます。キー列の集合を指定しないと、DELETE 文は文書にあるすべての列を一致させようとします。次のような XML 文書があるとします。

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>94065</ZIP>
    </DEPTADDR>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

削除するために、XSU は次のような DELETE 文（各 ROW 要素ごとに 1 つ）を実行します。

```
DELETE FROM Dept WHERE DEPTNO = ? AND DEPTNAME = ? AND DEPTADDR = ?
binding,
DEPTNO <- 100
DEPTNAME <- Sports
DEPTADDR <-  AddressType('100 Redwood Shores Pkwy','Redwood City','CA','94065')
```

7-41 ページの「XSU を使用した削除処理（Java API）」を参照してください。

XSU のコマンドラインのフロントエンド OracleXML の使用

XSU には、単純なコマンドラインのフロントエンドがあります。これを使用すると、XML の生成機能および挿入機能にすばやくアクセスできます。Oracle では、XSU のフロントエンドは、更新機能と削除機能をサポートしていません。XSU のコマンドライン・オプションは、Java クラス OracleXML を介して利用できます。このオプションを起動するには、次をコールします。

```
java OracleXML
```

これによって、フロントエンドの使用情報が出力されます。XSU のコマンドラインのフロントエンドを実行するには、まず実行可能ファイルがある場所を指定します。CLASSPATH に次の場所を追加します。

- XSU の Java ライブラリ（xsu12.jar または xsu111.jar）

XSU は Oracle XML Parser および JDBC ドライバに依存しているため、これらのコンポーネントの場所も指定します。これらの場所を指定するには、CLASSPATH に次の場所を含める必要があります。

- Oracle XML Parser の Java ライブラリ（xmlparserv2.jar）
- JDBC ライブラリ（xsu12.jar を使用している場合は classes12.jar、xsu111.jar を使用している場合は classes111.jar）

XSU のコマンドラインを使用した XML の生成

XSU の生成機能を使用するには、XSU の getXML パラメータを使用します。たとえば、scott スキーマにある emp 表を問い合せて XML 文書を生成するには、次の文を発行します。

```
java OracleXML getXML -user "scott/tiger" "select * from emp"
```

これによって、次のタスクが実行されます。

- 現在のデフォルト・データベースへの接続
- 問合せ select * from emp の実行
- 結果の XML への変換
- 結果の表示

getXML は、広範囲なオプションをサポートします。これらのオプションについては、次の項を参照してください。

XSU の OracleXML getXML オプション

表 7-1 に、OracleXML getXML オプションを示します。

表 7-1 XSU の OracleXML getXML オプション

getXML オプション	説明
-user "<username>/<password>"	データベースに接続するためのユーザー名およびパスワードを指定します。このオプションを指定しないと、デフォルトで scott/tiger が指定されます。接続文字列も指定されている場合は、ユーザー名およびパスワードはこの接続文字列の一部として指定できます。
-conn "<JDBC_connect_string>"	JDBC データベースの接続文字列を指定します。デフォルトの接続文字列は "jdbc:oracle:oci8:@" です。
-withDTD	XML 文書とともに DTD も生成するように XSU に指示します。
-rowsetTag "<tag_name>"	ROWSET タグ（問合せによって戻されたレコードに対応するすべての XML 要素を囲むタグ）を指定します。デフォルトの ROWSET タグは、ROWSET です。ROWSET に空の文字列を指定すると、XSU は ROWSET 要素を完全に省略します。
-rowTag "<tag_name>"	ROW タグ（データベース行に対応するデータを囲むタグ）を指定します。デフォルトの ROW タグは ROW です。ROW タグに空の文字列を指定すると、XSU は ROW タグを完全に省略します。
-rowIdAttr "<row_id-attribute-name>"	ROW 要素の属性に名前を付け、行の数を追跡します。デフォルトでは、この属性は num と呼ばれます。rowID 属性に空の文字列（""）を指定すると、XSU は属性を省略します。
-rowIdColumn "<row Id column name>"	問合せからのスカラー列の 1 つの値が、rowID 属性の値として使用されるように指定します。
-collectionIdAttr "<collection id attribute name>"	XML リスト要素の属性に名前を付け、行の数を追跡します（生成される XML リストは、カーソル問合せまたはコレクションのいずれかに対応することに注意してください）。rowID 属性に空の文字列（""）を指定すると、XSU は属性を省略します。
-useNullAttrId	要素が NULL であることを示すために属性 NULL （TRUE/FALSE）を使用するように XSU に指示します。
-styleSheet "<stylesheet URI>"	XML PI（処理命令）にスタイルシートを指定します。
-stylesheetType "<stylesheet type>"	XML PI（処理命令）にスタイルシートの型を指定します。
-errorTag "<error tag name>"	エラー・タグを指定します。エラー・タグは、XML にフォーマットされたエラー・メッセージを囲むタグです。

表 7-1 XSU の OracleXML getXML オプション（続き）

getXML オプション	説明
-raiseNoRowsException	行が戻されなかった場合に例外を呼び出すように XSU に指示します。
-maxRows "<maximum number of rows>"	XML に変換するために取り出される行の最大数を指定します。
-skipRows "<number of rows to skip>"	スキップされる行の数を指定します。
-encoding "<encoding name>"	生成される XML のキャラクタ・セットのエンコーディングを指定します。
-dateFormat "<date format>"	XML 文書内の日付値用の書式を指定します。
-fileName "<SQL query fileName>" | <sql query>	問合せを含むファイル名か、問合せ自体を指定します。

XSU のコマンドライン（putXML）を使用した XML の挿入

XML 文書を scott スキーマにある emp 表に挿入するには、次の構文を使用します。

```
java OracleXML putXML -user "scott/tiger" -fileName "/tmp/temp.xml" "emp"
```

これによって、次のタスクが実行されます。

- 現在のデータベースへの接続
- 特定のファイルからの XML 文書の読込み
- XML 文書の解析およびタグと列名の一致
- emp 表への値の適切な挿入

> **注意：** XSU のコマンドラインのフロントエンド putXML は、現在 XSU の挿入機能のみを実装しています。将来的には、XSU の更新機能および削除機能も実装される予定です。

XSU の OracleXML putXML オプション

表 7-2 に、putXML オプションを示します。

表 7-2 XSU の OracleXML putXML オプション

putXML オプション	説明
-user "<username>/<password>"	データベースに接続するためのユーザー名およびパスワードを指定します。このオプションを指定しないと、デフォルトで scott/tiger が指定されます。接続文字列も指定されている場合は、ユーザー名およびパスワードはこの接続文字列の一部として指定できます。
-conn "<JDBC_connect_string>"	JDBC データベースの接続文字列を指定します。デフォルトの接続文字列は "jdbc:oracle:oci8:@" です。
-batchSize "<batching size>"	バッチ・サイズを指定します。これによって、バッチされ、一度にデータベースに挿入される行の数を制御できます。バッチ処理を行うと、パフォーマンスが向上します。
-commitBatch "<commit size>"	コミットが実行される挿入レコードの数を指定します。自動コミットを true（デフォルト）にしている場合は、commitBatch を設定しても、処理は行われません。
-rowTag "<tag_name>"	ROW タグ（データベース行に対応するデータを囲むタグ）を指定します。デフォルトの ROW タグは ROW です。ROW タグに空の文字列を指定して、行を囲むタグが XML 文書で使用されていないときに通知するように XSU に指示します。
-dateFormat "<date format>"	XML 文書内の日付値用の書式を指定します。
-ignoreCase	列名とタグ名を大文字 / 小文字を区別せずに一致させます（たとえば、ignoreCase がオンになっている場合、「EmpNo」と「EMPNO」は一致します）。
-fileName "<file name>" | -URL "<url>" | -xmlDoc "<xml document>"	挿入する XML 文書を指定します。fileName オプションはローカル・ファイルを指定し、URL は文書をフェッチする URL を指定します。また、xmlDoc オプションは、XML 文書をコマンドライン上の文字列としてインラインに格納します。
<tableName>	値を挿入する表の名前です。

XSU の Java API

XSU の Java API は、次の 2 つのクラスで構成されます。

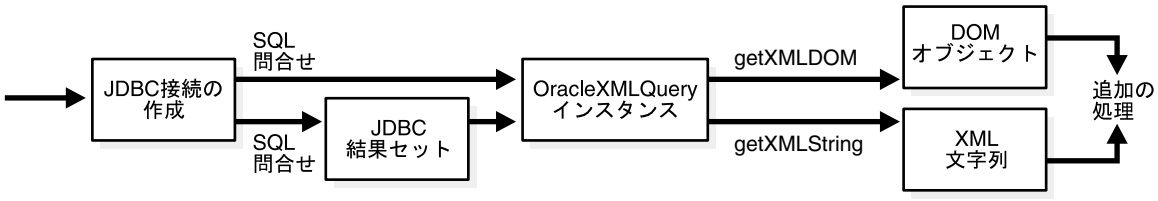
- XML 生成用の XSU API: `oracle.xml.sql.query.OracleXMLQuery`
- XML の保存、挿入、更新および削除用の XSU API: `oracle.xml.sql.dml.OracleXMLSave`

XSU の OracleXMLQuery を使用した XML の生成

`OracleXMLQuery` クラスは、XSU の Java API の XML 生成部分を構成します。図 7-4 に、OracleXMLQuery を使用する XML の生成に必要な基本的な手順を示します。

1. 接続を作成します。
2. SQL 文字列または `ResultSet` オブジェクトを指定して、`OracleXMLQuery` インスタンスを作成します。
3. 結果を DOM ツリーまたは XML 文字列として取得します。

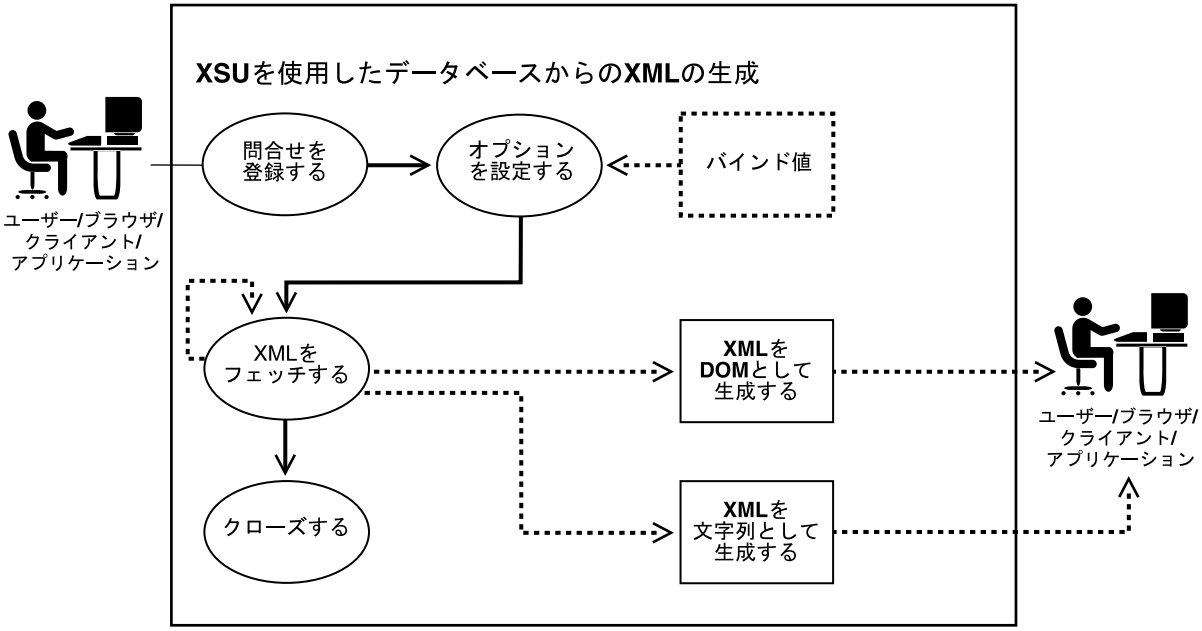
図 7-4 XSU for Java を使用した XML の生成 : 基本的な処理の流れ



XSU を使用した SQL 問合せからの XML の生成

次の例では、XSU を使用して、特定の SQL 問合せにおいて DOM 表現または文字列表現で XML 文書を生成する方法を示します。図 7-5 を参照してください。

図 7-5 XSU を使用した XML の生成



XSU を使用した XML 生成の例 1: emp 表からの文字列の生成（Java）

1. **接続を作成します。**

   XML を生成する前に、データベースへの接続を作成する必要があります。この接続は、JDBC 接続文字列を指定して作成できます。Oracle JDBC クラスを登録してから、接続を作成します。

```
// import the Oracle driver..
import oracle.jdbc.driver.*;

// Load the Oracle JDBC driver
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
// Create the connection.
Connection conn =
 DriverManager.getConnection("jdbc:oracle:oci8:@","scott","tiger");
```

これで、OCI8 JDBC ドライバを使用して接続が作成されました。パスワード tiger を指定して scott スキーマに接続できます。scott スキーマが現在のデータベース（環境変数 ORA_SID によって識別される）に接続します。データベースへの接続には、JDBC Thin ドライバも使用できます。Thin ドライバは Pure Java で作成されており、アプレットやその他の Java プログラム内からコールできます。

参照： 詳細は、『Oracle9*i* Java Developer's Guide』を参照してください。

- **Thin ドライバを使用した接続 :** 次に、JDBC Thin ドライバを使用した接続の例を示します。

  ```
  // Create the connection.
  Connection conn =
  DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL",
                              "scott","tiger");
  ```

  Thin ドライバには、ホスト名（dlsun489）、ポート番号（1521）、およびマシン上の特定の Oracle インスタンスを識別する Oracle SID（ORCL）を指定する必要があります。

- **接続が不要なサーバー内での実行 :** サーバー内で実行するコードであるサーバー側の Java コードを作成する場合は、サーバー側の内部ドライバはデフォルト・セッション内で実行するため、ユーザー名およびパスワードを使用して接続する必要はありません。ユーザーはすでに接続されています。この場合は、oracle.jdbc.driver.OracleDriver() クラスの defaultConnection() をコールし、現在の接続を取得します。

  ```
  import oracle.jdbc.driver.*;

  // Load the Oracle JDBC driver
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
  Connection conn =  new oracle.jdbc.driver.OracleDriver ().defaultConnection
  ();
  ```

この後の説明では、クライアントからの OCI8 接続か、またはユーザーが接続オブジェクトをすでに作成していると想定します。必要に応じて、適切な接続作成方法を使用してください。

2. **OracleXMLQuery クラスのインスタンスを作成します。**

   接続を登録したら、次の SQL 問合せを発行して OracleXMLQuery クラスのインスタンスを作成します。

```
// import the query class in to your class
import oracle.xml.sql.query.OracleXMLQuery;

OracleXMLQuery qry = new OracleXMLQuery (conn, "select * from emp");
```

   これで、問合せ用クラスを使用できます。

3. **結果を DOM ツリーまたは XML 文字列として取得します。**

   - **DOM オブジェクト出力 :** 文字列ではなく DOM オブジェクトを取得する場合は、次のように DOM 出力を指定します。

```
org.w3c.DOM.Document domDoc = qry.getXMLDOM();
```

     その後、DOM 検索を使用します。

   - **XML 文字列出力 :** 次の文によって、XML 文字列の結果を取得できます。

```
String xmlString = qry.getXMLString();
```

次に、XML 文字列を抽出（生成）するプログラムの詳細なリストを示します。このプログラムは文字列を取得し、標準出力に出力します。

```
Import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;

// class to test the String generation!
class testXMLSQL {

   public static void main(String[] argv)
   {

     try{
      // create the connection
      Connection conn  = getConnection("scott","tiger");

      // Create the query class.
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

      // Get the XML string
      String str = qry.getXMLString();

      // Print the XML output
```

```
      System.out.println(" The XML output is:\n"+str);
      // Always close the query to get rid of any resources..
     qry.close();
     }catch(SQLException e){
      System.out.println(e.toString());
     }
   }

   // Get the connection given the user name and password..!
   private static Connection getConnection(String username, String password)
     throws SQLException
   {
      // register the JDBC driver..
       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());


      // Create the connection using the OCI8 driver
       Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",username,password);

      return conn;
   }
}
```

このプログラムの実行方法

このプログラムを実行するには、次の手順に従います。

1. このプログラムを testXMLSQL.java というファイルに格納します。
2. Java コンパイラ javac を使用してこのプログラムをコンパイルします。
3. java testXMLSQL と指定してこのプログラムを実行します。

Java 実行可能ファイルがクラスを検索できるように、CLASSPATH がこのディレクトリを指すようにする必要があります。このプログラムのコンパイルおよび実行には、Oracle の JDeveloper などの様々な Java ツールも使用できます。このプログラムを実行すると、画面に XML ファイルが出力されます。

XSU を使用した XML 生成の例 2: emp 表からの DOM の生成（Java）

DOM は W3C が定義した標準です。DOM は、XML 文書を解析ツリーのような形式で表現します。各 XML エンティティは DOM のノードになります。したがって、XML 要素および属性は DOM のノードになり、子ノードになります。XSU が生成した XML から DOM ツリーを生成するには、XSU に DOM 文書を直接要求します。XSU は、XML 文書の文字列表現を作成するオーバーヘッドなしで、これを解析して DOM ツリーを生成します。

XSU は、XML パーサーをコールしてデータ値から直接 DOM ツリーを構築します。次に、DOM ツリーを生成する方法の例を示します。この例では、DOM ツリー内を移動し、すべてのノードを 1 つずつ出力します。

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;

 class domTest{

   public static void main(String[] argv)
   {
      try{
      // create the connection
      Connection conn  = getConnection("scott","tiger");

      // Create the query class.
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

      // Get the XML DOM object. The actual type is the Oracle Parser's DOM
      // representation. (XMLDocument)
      XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();

      // Print the XML output directly from the DOM
      domDoc.print(System.out);

      // If you instead want to print it to a string buffer you can do this..!
      StringWriter s = new StringWriter(10000);
      domDoc.print(new PrintWriter(s));
      System.out.println(" The string version ---> "+s.toString());

      qry.close(); // You should always close the query!!
      }catch(Exception e){
        System.out.println(e.toString());
      }
    }
```

```
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

結果ページの区切り : skipRows および maxRows

これまでに示した例では、XSU は結果セットまたは問合せを受け入れ、問合せのすべての行から完全な文書を生成します。一度に 100 行を取得するには、問合せを発行して最初の 100 行を取得し、別の問合せを発行して次の 100 行、のように繰り返す必要があります。また、たとえば問合せの最初の 5 行をスキップして結果を生成することはできません。

これらを必要に応じて取得するには、XSU の `skipRows` パラメータ設定および `maxRows` パラメータ設定を使用します。

- `skipRows` パラメータを設定すると、必要な行数を強制的にスキップさせて結果を生成できます。
- `maxRows` パラメータは、XML に変換する行の数を制限します。

たとえば、`skipRows` を 5 に設定し、`maxRows` を 10 に設定すると、XSU は最初の 5 行をスキップして、次の 10 行で XML を生成します。

ユーザーのセッション期間にわたるオブジェクトのオープン状態の保持

Web では、問合せオブジェクトをユーザーのセッション中オープンにしておく必要がある場合があります。たとえば、ユーザーの検索の結果をページ区切りの形式で表示する Web 検索エンジンについて考えてみます。これらのエンジンでは、最初のページに 10 個の結果、次のページに次の 10 個の結果、のように結果が表示されます。

このように表示するには、XSU に、一度に 10 行ずつ変換して結果セット状態をアクティブに保つように指示します。こうすると XSU は、次に追加の結果を要求されたときに、前回の生成が終了した所から生成を開始します。7-28 ページの「XSU を使用した XML 生成の例 3: 結果ページの区切り : コール時の XML ページの生成（Java）」を参照してください。

行数または行内の列数が多すぎる場合

行数または行内の列数が多すぎる場合もあります。この場合は、それぞれサイズが小さい文書を複数生成できます。

これは、maxRows パラメータおよび keepObjectOpen ファンクションを使用して処理できます。

keepObjectOpen ファンクション

通常、OracleXMLQuery は、発行された SQL 問合せを使用して ResultSet を作成している場合は、すべての結果が生成されるとその ResultSet をクローズします。これは、OracleXMLQuery が、それ以上の結果を必要としないと想定するためです。ただし、前述の例の場合は、その状態を保持する必要があるため、keepObjectOpen ファンクションをコールしてカーソルをアクティブにしておく必要があります。次の例を参照してください。

XSU を使用した XML 生成の例 3: 結果ページの区切り : コール時の XML ページの生成（Java）

次の例では、状態を保持する単純なクラスを作成し、このクラスがコールされるたびに次のページを作成します。

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class pageTest
{
   Connection conn;
   OracleXMLQuery qry;
   ResultSet rset;
   Statement stmt;
   int lastRow = 0;

   public pageTest(String sqlQuery)
   {
     try{
     conn  = getConnection("scott","tiger");
     //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     //             ResultSet.CONCUR_READ_ONLY);// create a scrollable Rset
     //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
     //             ResultSet.CONCUR_READ_ONLY);// create a scrollable Rset
     stmt = conn.createStatement();
     ResultSet rset = stmt.executeQuery(sqlQuery);  // get the result set..
     rset.first();
```

```
  qry = new OracleXMLQuery(conn,rset);   // create a OracleXMLQuery instance
  qry.keepCursorState(true); // Don't lose state after the first fetch
  qry.setRaiseNoRowsException(true);
  qry.setRaiseException(true);
  }catch(SQLException e){
   System.out.println(e.toString());
  }
}

// Returns the next XML page..!
public String getResult(int startRow, int endRow)  throws SQLException
{
  //rset.relative(lastRow-startRow);  // scroll inside the result set
  //rset.absolute(startRow);  // scroll inside the result set
  qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
  //System.out.println("before getxml");
  return qry.getXMLString();
}

// Function to still perform the next page.
public String nextPage() throws SQLException
{
  String result = getResult(lastRow,lastRow+10);
  lastRow+= 10;
  return result;
}

public void close() throws SQLException
{
  stmt.close();   // close the statement..
  conn.close();   // close the connection
  qry.close();    // close the query..
}

public static void main(String[] argv)
{
  String str;

  try{
  pageTest test = new pageTest("select e.* from emp e");

  int i = 0;
  // Get the data one page at a time..!!!!!
  while ((str = test.getResult(i,i+10))!= null)
  {
      System.out.println(str);
      i+= 10;
```

```
      }
      test.close();
     }catch(Exception e){
      e.printStackTrace(System.out);
     }
    }
 // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
    }

}
```

ResultSet オブジェクトからの XML の生成

これまでは、SQL 問合せを発行して結果を XML として取得する方法について説明しました。最後の例では、結果をページ区切りの形式で取り出しました。ただし Web では、次のページの結果のみでなく、前のページも取り出す必要がある場合があります。このスクロール可能機能を使用するには、スクロール可能な `ResultSet` を使用します。このスクロール可能な `ResultSet` オブジェクトを使用して結果セット内を移動し、移動するたびに XSU を使用して XML を生成します。次の例は、この方法を示します。

XSU を使用した XML 生成の例 4: JDBC の ResultSet からの XML の生成（Java）

この例では、JDBC の `ResultSet` を使用して XML を生成する方法を説明します。`ResultSet` は、バッチ・サイズの設定、値のバインディングなど、XSU が直接処理しない場合に使用する必要があることに注意してください。この例では、前の項で定義した `pageTest` を拡張し、どのページでも処理できるようにします。

```
public class pageTest()
{
   Connection conn;
   OracleXMLQuery qry;
   ResultSet rset;
   int lastRow = 0;

   public pageTest(String sqlQuery)
   {
     conn  = getConnection("scott","tiger");
```

```
    Statement stmt = conn.createStatement(sqlQuery);// create a scrollable Rset
    ResultSet rset = stmt.executeQuery();  // get the result set..
    qry = new OracleXMLQuery(conn,rset);   // create a OracleXMLQuery instance
    qry.keepObjectOpen(true); // Don't lose state after the first fetch
  }

  // Returns the next XML page..!
  public String getResult(int startRow, int endRow)
  {
    rset.scroll(lastRow-startRow);  // scroll inside the result set
    qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
    return qry.getXMLString();
  }

  // Function to still perform the next page.
  public String nextPage()
  {
    String result = getResult(lastRow,lastRow+10);
    lastRow+= 10;
    return result;
  }

  public void close()
  {
    stmt.close();   // close the statement..
    conn.close();   // close the connection
    qry.close();    // close the query..
  }

  public void main(String[] argv)
  {
    pageTest test = new pageTest("select * from emp");

    int i = 0;
    // Get the data one page at a time..!!!!!
    while ((str = test.getResult(i,i+10))!= null)
    {
        System.out.println(str);
        i+= 10;
    }
    test.close();
  }
}
```

XSU を使用した XML 生成の例 5: プロシージャの戻り値からの XML の生成

OracleXMLQuery クラスは、問合せ文字列または ResultSet に対してのみ XML 変換を行います。ただし、アプリケーションに PL/SQL プロシージャがあり、このプロシージャが REF カーソルを戻した場合の変換方法を考えてみます。

この場合は、前述の ResultSet の変換メカニズムを使用してタスクを実行できます。REF カーソルは、PL/SQL 内のカーソル・オブジェクトへの参照です。これらのカーソル・オブジェクトは有効な SQL 文であり、一連の値を取得するために繰り返されます。これらの REF カーソルは、Java では OracleResultSet オブジェクトに変換されます。

これらのプロシージャを実行して OracleResultSet オブジェクトを取得し、このオブジェクトを OracleXMLQuery オブジェクトに送信して必要な XML を取得できます。

次のように REF カーソルを定義し、そのカーソルを戻す PL/SQL ファンクションについて考えてみます。

```
CREATE OR REPLACE package body testRef is

  function testRefCur RETURN empREF is
  a empREF;
  begin
      OPEN a FOR select * from scott.emp;
      return a;
  end;
end;
/
```

これでこのファンクションは、コールされるたびに問合せ select * from emp に対してカーソル・オブジェクトをオープンし、そのカーソル・インスタンスを戻します。このインスタンスを XML に変換するには、次のようにします。

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class REFCURtest
{
   public static void main(String[] argv)
     throws SQLException
   {
      String str;
      Connection conn  = getConnection("scott","tiger"); // create connection

      // Create a ResultSet object by calling the PL/SQL function
      CallableStatement stmt =
         conn.prepareCall("begin ? := testRef.testRefCur(); end;");
```

```
      stmt.registerOutParameter(1,OracleTypes.CURSOR); // set the define type

      stmt.execute();   // Execute the statement.
      ResultSet rset = (ResultSet)stmt.getObject(1);  // Get the ResultSet

      OracleXMLQuery qry = new OracleXMLQuery(conn,rset); // prepare Query class
      qry.setRaiseNoRowsException(true);
      qry.setRaiseException(true);
      qry.keepCursorState(true);        // set options (keep the cursor alive..
      while ((str = qry.getXMLString())!= null)
           System.out.println(str);

      qry.close();    // close the query..!

      // Note since we supplied the statement and resultset, closing the
      // OracleXMLquery instance will not close these. We would need to
      // explicitly close this ourselves..!
      stmt.close();
      conn.close();
   }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }

}
```

スタイルシートを適用するには、`applyStylesheet()` メソッドを使用します。これによって、出力が生成される前に強制的にスタイルシートが適用されます。

該当する行がない場合の例外の呼出し

XSU は、処理する行がない場合は NULL 文字列を戻します。ただし、アプリケーションが例外ハンドラで処理できるように、生成される行がなくなるたびに例外を呼び出す方が効率的である場合があります。setRaiseNoRowsException() を設定すると、XSU は、出力用に生成する行がなくなるたびに oracle.xml.sql.OracleXMLSQLNoRowsException を呼び出します。これはランタイムの例外であり、必要がないときはキャッチする必要はありません。

XSU を使用した XML 生成の例 6: 該当する行がない場合の例外（Java）

次のコードでは、NULL 文字列をチェックするかわりに例外を使用するように前述の例を拡張します。

```
public class pageTest {
    .... // rest of the class definitions....

   public void main(String[] argv)
   {
     pageTest test = new pageTest("select * from emp");

     test.query.setRaiseNoRowsException(true); // ask it to generate exceptions
     try
     {
        while(true)
         System.out.println(test.nextPage());
     }
     catch(oracle.xml.sql.OracleXMLNoRowsException)
     {
       System.out.println(" END OF OUTPUT ");
       test.close();
     }
   }
}
```

注意： 結果が NULL になるときから例外ハンドラになるときに、終了を確認する条件が変わったことに注意してください。

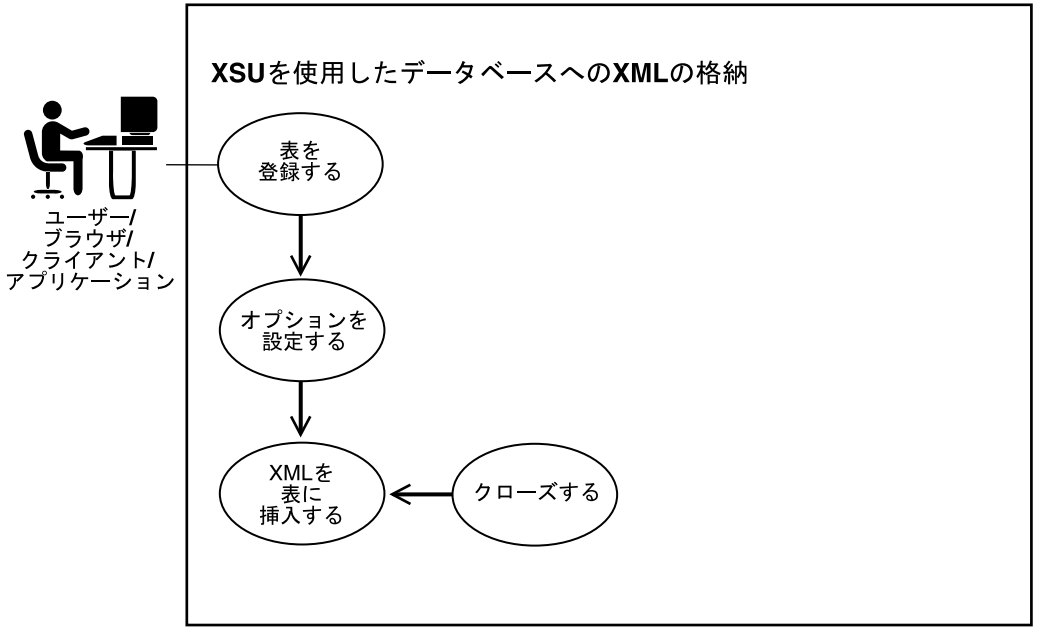
XSU の OracleXMLSave を使用したデータベースへの XML の格納

これまでは、問合せを XML へ変換する方法について説明しました。次に、XSU を使用して XML を表またはビューに戻す方法について説明します。

`oracle.xml.sql.dml.OracleXMLSave` クラスによって、この機能が提供されます。このクラスによって、XML を表に挿入したり、XML 文書で既存の表を更新したり、XML 要素の値に基づいて、表から行を削除することができます。

これらのすべての操作では、特定の XML 文書が解析され、要素のタグ名と、ターゲット表またはビューにある列名のタグ名が一致するかどうかが調べられます。要素はその後 SQL 型に変換され、適切な文にバインドされます。図 7-6 に、XSU を使用した XML 格納のプロセスを示します。

図 7-6 XSU を使用したデータベースへの XML の格納



XML 文書には ROW 要素のリストがあると想定します。各 ROW 要素は個々の DML 操作（表またはビューでの挿入、更新または削除）を構成します。

XSU を使用した挿入処理（Java API）

表またはビューに文書を挿入する手順は、表名またはビュー名、および文書を指定するのみです。XSU は文書（文字列が指定されている場合）を解析し、INSERT 文を作成してこの文にすべての値をバインドします。デフォルトでは、XSU は値を表またはビューのすべての列に挿入します。存在しない要素は NULL 値として処理されます。次の例は、emp 表から生成した XML 文書を、比較的簡単に表に格納する方法を示します。

XSU を使用した XML 挿入の例 7: すべての列への XML 値の挿入（Java）

この例では、XML 値をすべての列に挿入します。

```
// This program takes as an argument the file name, or a url to
// a properly formated XML document and inserts it into the SCOTT.EMP table.
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
   public static void main(String argv[])
     throws SQLException
  {
     DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
     Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@","scott","tiger");

     OracleXMLSave sav = new OracleXMLSave(conn, "emp");
     sav.insertXML(sav.getUrl(argv[0]));
     sav.close();
   }
}
```

次の形式の INSERT 文が生成されます。

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

列名と一致している入力 XML 文書内の要素タグが照合され、その値がバインドされます。

次の XML 文書をファイルに格納するとします。

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

このファイルを前述のプログラムに指定すると、値（7369、Smith、CLERK、7902、12/17/1980、800、20）を含む emp 表に新しい行が生成されます。行要素内に存在しない要素は、NULL 値として扱われます。

XSU を使用した XML 挿入の例 8: 特定の列への XML 値の挿入

値を挿入する必要がない列がある場合もあります。取得する値が完全なセットではない場合などです。残りの列用に使用するトリガーまたはデフォルト値が必要です。次のコードは、これを行う方法を示しています。

従業員番号、名前および仕事の値のみを取得し、給与、マネージャ、部門番号および雇用日のフィールドには自動的に値が挿入されると想定します。まず、挿入を実行する列名のリストを作成し、このリストを OracleXMLSave インスタンスに渡します。

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] colNames = new String[5];
      colNames[0] = "EMPNO";
      colNames[1] = "ENAME";
      colNames[2] = "JOB";

      sav.setUpdateColumnList(colNames); // set the columns to update..!
```

```
      // Assume that the user passes in this document as the first argument!
      sav.insertXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

次の形式の INSERT 文が生成されます。

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

前述の例では、挿入した文書に他の列の値（JOB、HIREDATE など）が含まれている場合は、これらの値は無視されることに注意してください。挿入は、入力にある各 ROW 要素に対しても実行されます。この挿入は、デフォルトでバッチ処理されます。

XSU を使用した更新処理（Java API）

これまでは XML 文書から表への値の挿入方法について説明しました。次に、特定の値のみを更新する方法について説明します。XML 文書を取得して、ある従業員の給与、およびその従業員が勤務する部門を更新するとします。

```
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <SAL>1800</SAL>
    <DEPTNO>30</DEPTNO>
  </ROW>
  <ROW>
    <EMPNO>2290</EMPNO>
    <SAL>2000</SAL>
    <HIREDATE>12/31/1992</HIREDATE>
  <!-- additional rows ... -->
</ROWSET>
```

XSU を使用して値を更新できます。更新の場合は、XSU にキー列のリストを指定する必要があります。キー列名は、UPDATE 文の WHERE 句の一部になります。前述の emp 表では、従業員番号（EMPNO）列がキーを構成します。これを使用して更新します。

XSU を使用した XML 更新の例 9: keyColumn を使用した表の更新（Java）

次の例では、keyColumn を使用して、emp 表を更新します。

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

この例では、2 つの UPDATE 文が生成されます。1 つ目の ROW 要素に対して、SAL フィールドおよび JOB フィールドを更新する UPDATE 文を次のように生成します。

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

2 つ目の ROW 要素には次の文を生成します。

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU を使用した XML 更新の例 10: 指定された列のリストの更新（Java）

列のリストを指定して更新する必要がある場合もあります。列のリストを指定すると、すべての ROW 要素に対して同じ UPDATE 文を使用できるため、より高速に処理できます。また、XML 文書内にある他のタグを無視できます。

> **注意：** 更新する列のリストを指定すると、更新する列に対応する要素が存在しない場合は、その要素は NULL として処理されることに注意してください。

更新されるすべての要素が XML 文書内のすべての ROW 要素と同じであることがわかっている場合は、setUpdateColumnNames() メソッドを使用して更新する列のリストを設定できます。

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // you create the list of columns to update..!
      // Note that if you do not supply this, then for each ROW element in the
      // XML document, you would generate a new update statement to update all
      // the tag values (other than the key columns)present in that element.
      String[] updateColNames = new String[2];
      updateColNames[0] = "SAL";
      updateColNames[1] = "JOB";
      sav.setUpdateColumnList(updateColNames); // set the columns to update..!

      // Assume that the user passes in this document as the first argument!
      sav.updateXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
      return conn;
   }
}
```

XSU を使用した削除処理（Java API）

XML 文書からの削除の場合は、キー列のリストを設定できます。これらの列は、DELETE 文の WHERE 句で使用されます。キー列名を指定しないと、DELETE 文の WHERE 句にある列のリストと ROW 要素にある列が一致する場合は、新しい DELETE 文が XML 文書の各 ROW 要素に対して作成されます。

XSU を使用した XML 削除の例 11: ROW ごとの削除操作（Java）

次の削除の例について考えます。

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      // Assume that the user passes in this document as the first argument!
      sav.deleteXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

更新の例で示した XML 文書と同じ文書を使用する場合は、次の 2 つの DELETE 文が生成されます。

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

DELETE 文は、XML 文書内の各 ROW 要素にあるタグ名に基づいて構成されます。

XSU を使用した XML 削除の例 12: 指定したキー値の削除（Java）

DELETE 文に述語としてキー値のみを使用する場合は、setKeyColumn ファンクションを使用してこれを設定できます。

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");
      OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

      String [] keyColNames = new String[1];
      keyColNames[0] = "EMPNO";
      sav.setKeyColumnList(keyColNames);

      // Assume that the user passes in this document as the first argument!
      sav.deleteXML(argv[0]);
      sav.close();
   }
   // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
      throws SQLException
    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
     return conn;
   }
}
```

次の形式の単一の DELETE 文が生成されます。

```
DELETE FROM scott.emp WHERE EMPNO=?
```

この文は、文書内のすべての ROW 要素に対して使用されます。

XSU の PL/SQL API

XSU の PL/SQL API は、XML 文書の生成およびデータベースへの格納の方法が Java API に似ています。DBMS_XMLQuery および DBMS_XMLSave の 2 つのパッケージは、Java クラス OracleXMLQuery および OracleXMLSave の機能に似ています。これらの両方のパッケージには、パッケージに対応付けられたコンテキスト・ハンドルがあります。コンストラクタに類似したファンクションの 1 つをコールしてコンテキストを作成し、ハンドルを取得してそのハンドルをすべての副問合せコールに使用します。

DBMS_XMLQuery() を使用した XML の生成

XML を生成すると、XML 文書を含む CLOB が生成されます。DBMS_XMLQuery および XSU 生成エンジンを使用する手順は次のとおりです。

1. DBMS_XMLQuery.getCtx ファンクションをコールし、このハンドルを問合せに（CLOB または VARCHAR2 として）指定してコンテキスト・ハンドルを作成します。
2. DBMS_XMLQuery.bind ファンクションを使用して、考えられる値を問合せにバインドします。バインドは、名前を位置にバインドすることで行われます。たとえば、select * from emp where empno = :EMPNO_VAR のような問合せができます。ここでは、setBindValue ファンクションを使用して EMPNO_VAR 用の値をバインドします。
3. ROW タグ名、ROWSET タグ名、フェッチする行の数などの引数をオプションで設定します。
4. getXML() ファンクションを使用して、XML を CLOB としてフェッチします。getXML() をコールして、DTD 付きまたは DTD なしで XML を生成できます。
5. コンテキストをクローズします。

次に、PL/SQL パッケージ DBMS_XMLQuery を使用する例を示します。

XSU を使用した XML 生成の例 13: 単純な問合せからの XML の生成（PL/SQL）

この例では、emp 表から行を選択して XML 文書を CLOB として取得します。まず、問合せを指定してコンテキスト・ハンドルを取得し、getXMLClob ルーチンをコールして CLOB 値を取得します。文書のエンコーディングは、データベース・キャラクタ・セットのエンコーディングと同じになります。

```
declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

  -- set up the query context...!
  queryCtx := DBMS_XMLQuery.newContext('select * from emp');
```

```
  -- get the result..!
  result := DBMS_XMLQuery.getXML(queryCtx);
  -- Now you can use the result to put it in tables/send as messages..
  printClobOut(result);
  DBMS_XMLQuery.closeContext(queryCtx);  -- you must close the query handle..
end;
/
```

XSU を使用した XML 生成の例 13a: 出力バッファへの CLOB の出力

printClobOut() は、CLOB を出力バッファへ出力する単純なプロシージャです。この PL/SQL コードを SQL*Plus で実行すると、CLOB の結果が画面に出力されます。結果を表示するには、serveroutput をオンに設定します。

```
/CREATE OR REPLACE PROCEDURE printClobOut(result IN OUT NOCOPY CLOB) is
xmlstr varchar2(32767);
line varchar2(2000);
begin
  xmlstr := dbms_lob.SUBSTR(result,32767);
  loop
    exit when xmlstr is null;
    line := substr(xmlstr,1,instr(xmlstr,chr(10))-1);
    dbms_output.put_line('| '||line);
    xmlstr := substr(xmlstr,instr(xmlstr,chr(10))+1);
  end loop;
end;
/
```

XSU を使用した XML 生成の例 14: ROW タグおよび ROWSET タグの変更（PL/SQL）

XSU の PL/SQL API では、ROW および ROWSET タグ名を変更することもできます。これらはデフォルトの名前であり、結果の各行の先頭と末尾、および文書全体の先頭と末尾にそれぞれ置かれます。これは、プロシージャ setRowTagName および setRowSetTagName によって次のように行われます。

```
--Setting the ROW tag names

declare
   queryCtx DBMS_XMLQuery.ctxType;
   result CLOB;
begin
   -- set the query context.
   queryCtx := DBMS_XMLQuery.newContext('select * from emp');
```

```
   DBMS_XMLQuery.setRowTag(queryCtx,'EMP'); -- sets the row tag name
   DBMS_XMLQuery.setRowSetTag(queryCtx,'EMPSET'); -- sets rowset tag name

   result := DBMS_XMLQuery.getXML(queryCtx); -- get the result

   printClobOut(result);  -- print the result..!
   DBMS_XMLQuery.closeContext(queryCtx);  -- close the query handle;
end;
/
```

結果として生成される XML 文書には、EMPSET 文書要素が含まれます。各行は、EMP タグによって区切られています。

XSU を使用した XML 生成の例 15: setMaxRows() および setSkipRows() の使用

問合せの生成の結果は、次のファンクションを使用してページを区切ることができます。

- setMaxRows ファンクション。このファンクションは、XML に変換する行の最大数を設定します。これは、最後の結果が生成された、現在の行の位置に関連します。
- setSkipRows ファンクション。このファンクションは、行値を XML に変換するときにスキップする行の数を指定します。

たとえば、emp 表の最初の 3 行をスキップして、残りの行を一度に 10 行ずつ出力するには、最初の 10 行のバッチの skipRows を 3 に設定し、残りのバッチの skipRows を 0（ゼロ）に設定します。

XSU の Java API の場合のように、keepObjectOpen() ファンクションをコールしてフェッチ間で状態が維持されるようにします。デフォルトでは、フェッチが終了すると状態がクローズされます。複数のフェッチの場合は、フェッチする行がなくなるときを判断する必要があります。これを行うには、setRaiseNoRowsException() を設定します。これによって、CLOB に行が書き込まれないときには例外が呼び出されます。これは、終了条件としてキャッチおよび使用できます。

```
-- Pagination of results

declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

  -- set up the query context...!
  queryCtx := DBMS_XMLQuery.newContext('select * from emp');

  DBMS_XMLQuery.setSkipRows(queryCtx,3); -- set the number of rows to skip
  DBMS_XMLQuery.setMaxRows(queryCtx,10); -- set the max number of rows per fetch

  result := DBMS_XMLQuery.getXML(queryCtx); -- get the first result..!
```

```
  printClobOut(result); -- print the result out.. This is you own routine..!
  DBMS_XMLQuery.setSkipRows(queryCtx,0); -- from now don't skip any more rows..!

  DBMS_XMLQuery.setRaiseNoRowsException(queryCtx,true);
                                        -- raise no rows exception..!
  begin
    loop  -- loop forever..!
      result := DBMS_XMLQuery.getXML(queryCtx); -- get the next batch
      printClobOut(result);             -- print the next batch of 10 rows..!
    end loop;
  exception
    when others then
    -- dbms_output.put_line(sqlerrm);
       null; -- termination condition, nothing to do;
  end;
  DBMS_XMLQuery.closeContext(queryCtx);  -- close the handle..!
end;
end;
/
```

XSU へのスタイルシートの設定（PL/SQL）

XSU の PL/SQL API は、次の方法で、生成した XML 文書にスタイルシートを設定する機能を提供します。

- 結果の XML にスタイルシート・ヘッダーを設定します。これを行うには、`setStylesheetHeader()` プロシージャを使用して、結果にスタイルシート・ヘッダーを設定します。このプロシージャは、単純に XML 処理命令を追加して、スタイルシートを含めます。
- 生成の前に、結果の XML 文書にスタイルシートを適用します。この機能を使用すると、パフォーマンスが大幅に向上します。この機能を使用しないと、XML 文書を CLOB として生成し、再度 XML パーサーに送信してからスタイルシートを適用する必要があります。XSU は DOM 文書を生成し、XML パーサーをコールし、スタイルシートを適用してから結果を生成します。結果の XML 文書にスタイルシートを適用するには、`useStyleSheet()` プロシージャを使用します。このプロシージャは、スタイルシートを使用して結果を生成します。

XSU のバインド値（PL/SQL）

XSU の PL/SQL API は、SQL 文に値をバインドする機能を提供します。SQL 文には、名前付きバインド変数を指定できます。この変数の前にはコロン（:）を付けて、バインド変数であることを宣言する必要があります。バインド変数を使用する手順は次のとおりです。

1. **問合せコンテキストを、バインド変数を含む問合せで初期化します。**たとえば、次の文は、バインド変数 :EMPNO および :ENAME を含む WHERE 句を使用して、emp 表から行を選択する問合せを登録します。これらのバインド変数には、後で従業員番号と従業員名の値をバインドします。

   ```
   queryCtx = DBMS_XMLQuery.getCtx('select * from emp where empno = :EMPNO and
   ename = :ENAME');
   ```

2. **バインド値のリストを設定します。**clearBindValues() は、すべてのバインド変数セットを消去します。setBindValue() は、単一のバインド変数に文字列値を設定します。たとえば、empno 値および ename 値を次のように設定します。

   ```
   DBMS_XMLQuery.clearBindValues(queryCtx);
   DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',20);
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','John');
   ```

3. **結果をフェッチします。**これによって、バインド値が文に適用され、述語 empno = 20 および ename = 'John' に対応する結果が取得されます。

   ```
   DBMS_XMLQuery.getXMLClob(queryCtx);
   ```

4. **必要に応じて値を再バインドします。**たとえば、次のように、ENAME のみを scott に変更し、問合せを再実行します。

   ```
   DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');
   ```

   ENAME の再バインドには、John のかわりに Scott が使用されます。

XSU を使用した XML 生成の例 15a: SQL 文への値のバインド

次の例は、SQL 文でのバインド変数の使用方法を示します。

```
declare
  queryCtx DBMS_XMLquery.ctxType;
  result CLOB;
begin

queryCtx := DBMS_XMLQuery.newContext(
       'select * from emp where empno = :EMPNO and ename = :ENAME');

DBMS_XMLQuery.clearBindValues(queryCtx);
DBMS_XMLQuery.setBindValue(queryCtx,'EMPNO',7566);
```

```
DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','JONES');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);

DBMS_XMLQuery.setBindValue(queryCtx,'ENAME','Scott');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);
end;
/
```

DBMS_XMLSave を使用したデータベースへの XML の格納

DBMS_XMLSave() および XSU 格納エンジンを使用する手順は次のとおりです。

1. DBMS_XMLSave.getCtx ファンクションをコールし、DML 操作に使用する表名をこのファンクションに指定してコンテキスト・ハンドルを作成します。
2. **挿入の場合は、**setUpdateColNames を使用して、挿入する列のリストを設定できます。デフォルトでは、すべての列に値が挿入されます。

   更新の場合は、キー列のリストを指定する必要があります。更新する列のリストを指定する必要がある場合もあります。この場合は、キー列名と一致する XML 文書内のタグが UPDATE 文の WHERE 句に使用され、更新列のリストと一致するタグが UPDATE 文の SET 句に使用されます。

   削除の場合は、デフォルトで、指定した文書の各 ROW 要素にあるすべてのタグ値と一致する WHERE 句が作成されます。この動作は、キー列のリストを設定することでオーバーライドできます。この場合は、タグ名がリスト内の列と一致するタグ値のみが、削除する行の識別に使用されます（DELETE 文の WHERE 句に使用して有効）。
3. 挿入の場合は insertXML ファンクション、更新の場合は updateXML ファンクション、および削除の場合は deleteXML ファンクションに、XML 文書を指定します。
4. 前回の操作を何度でも繰り返すことができます。
5. コンテキストをクローズします。

Java の場合の OracleXMLSave クラスの例と同じ例を使用します。

XSU を使用した挿入処理（PL/SQL API）

表またはビューに XML 文書を挿入する手順は、表名またはビュー名、および文書を指定するのみです。XSU は文書（文字列が指定されている場合）を解析し、INSERT 文を作成してこの文にすべての値をバインドします。デフォルトでは、XSU は値を表またはビューのすべての列に挿入します。存在しない要素は NULL 値として処理されます。

次のコードは、emp 表から生成した文書を、比較的簡単に emp 表に戻します。

XSU を使用した XML 挿入の例 16: すべての列への値の挿入（PL/SQL）

この例では、insProc プロシージャを作成します。このプロシージャは、次のものを受け入れます。

- CLOB としての XML 文書
- その文書を挿入する表名

次に、この XML 文書を表に挿入します。

```
create or replace procedure insProc(xmlDoc IN CLOB, tableName IN VARCHAR2) is
   insCtx DBMS_XMLSave.ctxType;
   rows number;
 begin
    insCtx := DBMS_XMLSave.newContext(tableName); -- get the context handle
    rows := DBMS_XMLSave.insertXML(insCtx,xmlDoc); -- this inserts the document
    DBMS_XMLSave.closeContext(insCtx);            -- this closes the handle
end;
/
```

これでこのプロシージャは、すべての XML 文書および表名を使用してコールできます。たとえば、次の形式のコールを使用します。

```
insProc(xmlDocument, 'scott.emp');
```

次の形式の INSERT 文が生成されます。

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

列名と一致している入力 XML 文書内の要素タグが照合され、その値がバインドされます。前述のコードのフラグメントを次の XML 文書に送る場合は、次の処理を行います。

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
```

```
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

emp 表に、値（7369、Smith、CLERK、7902、12/17/1980、800、20）を含む新しい行が作成されます。行要素内に存在しない要素は、NULL 値として扱われます。

XSU を使用した XML 挿入の例 17: 特定の列への値の挿入（PL/SQL）

値を挿入する必要がない列がある場合もあります。取得する値が完全なセットではない場合などで、残りの列用に使用するトリガーまたはデフォルト値が必要な場合などです。次のコードは、これを行う方法を示しています。

従業員番号、名前および仕事の値のみを取得し、給与、マネージャ、部門番号および雇用日のフィールドには自動的に値が挿入されると想定します。挿入を実行する列名のリストを作成し、このリストを DBMS_XMLSave プロシージャに渡します。これらの値は、setUpdateColumnName() プロシージャを繰り返しコールし、更新する列名を各コールごとに指定することによって設定できます。列名設定は、clearUpdateColumnNames() を使用して消去できます。

```
create or replace procedure testInsert( xmlDoc IN clob) is
  insCtx DBMS_XMLSave.ctxType;
  doc clob;
  rows number;
begin

   insCtx := DBMS_XMLSave.newContext('scott.emp'); -- get the save context..!

   DBMS_XMLSave.clearUpdateColumnList(insCtx); -- clear the update settings

   -- set the columns to be updated as a list of values..
   DBMS_XMLSave.setUpdateColumn(insCtx,'EMPNO');
   DBMS_XMLSave.setUpdateColumn(insCtx,'ENAME');
   DBMS_XMLSave.setUpdatecolumn(insCtx,'JOB');

   -- Now insert the doc. This will only insert into EMPNO,ENAME and JOB columns
   rows := DBMS_XMLSave.insertXML(insCtx, xmlDoc);
   DBMS_XMLSave.closeContext(insCtx);

end;
/
```

CLOB を文書として指定するプロシージャをコールすると、次の形式の INSERT 文が生成されます。

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

前述の例では、挿入した文書に他の列の値（JOB、HIREDATE など）が含まれている場合は、これらの値は無視されることに注意してください。

挿入は、入力にある各 ROW 要素に対しても実行されます。この挿入は、デフォルトでバッチ処理されます。

XSU を使用した更新処理（PL/SQL API）

これまでは XML 文書から表への値の挿入方法について説明しました。次に、特定の値のみを更新する方法について説明します。XML 文書を取得して、ある従業員の給与、およびその従業員が勤務する部門を更新するとします。

```
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <SAL>1800</SAL>
    <DEPTNO>30</DEPTNO>
  </ROW>
  <ROW>
    <EMPNO>2290</EMPNO>
    <SAL>2000</SAL>
    <HIREDATE>12/31/1992</HIREDATE>
  <!-- additional rows ... -->
</ROWSET>
```

更新処理をコールしてこれらの値を更新できます。更新の場合は、XSU にキー列名のリストを指定する必要があります。キー列名は、UPDATE 文の WHERE 句の一部になります。前述の emp 表では、従業員番号（EMPNO）列がキーを構成します。この列を更新に使用します。

XSU を使用した XML 更新の例 18: keyColumn を使用した XML 文書の更新

,.....

```
create or replace procedure testUpdate ( xmlDoc IN clob) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   updCtx := DBMS_XMLSave.newContext('scott.emp');  -- get the context
   DBMS_XMLSave.clearUpdateColumnList(updCtx); -- clear the update settings..

   DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column
```

```
   rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc);  -- update the table.
   DBMS_XMLSave.closeContext(updCtx);              -- close the context..!

end;
/
```

この例では、プロシージャが、前述の文書を含む CLOB 値を使用して実行されると、2 つの UPDATE 文が生成されます。最初の ROW 要素に対して、SAL フィールドおよび JOB フィールドを更新する UPDATE 文を次のように生成します。

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

2 つ目の ROW 要素には次の文を生成します。

```
update scott.emp SET SAL = 2000 and HIREDATE = '12/31/1992' WHERE EMPNO = 2290;
```

XSU を使用した XML 更新の例 19: 更新する列のリストの指定（PL/SQL）

列のリストを指定して更新する必要がある場合もあります。列のリストを指定すると、すべての ROW 要素に対して同じ UPDATE 文を使用できるため、より高速に処理できます。また、文書内にある他のタグを無視できます。更新する列のリストを指定すると、更新する列に対応する要素が存在しない場合は、その要素は NULL として処理されることに注意してください。

更新されるすべての要素が XML 文書内のすべての ROW 要素と同じであることがわかっている場合は、setUpdateColumnName() プロシージャを使用して更新する列のリストを設定できます。

```
create or replace procedure testUpdate(xmlDoc IN CLOB) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   updCtx := DBMS_XMLSave.newContext('scott.emp');
   DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column

   -- set list of columnst to update.
   DBMS_XMLSave.setUpdateColumn(updCtx,'SAL');
   DBMS_XMLSave.setUpdateColumn(updCtx,'JOB');

   rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc); -- update the XML document..!
   DBMS_XMLSave.closeContext(updCtx);   -- close the handle

end;
/
```

XSU を使用した削除処理（PL/SQL API）

削除の場合は、キー列のリストを設定できます。これらの列は、DELETE 文の WHERE 句の一部になります。キー列名を指定しないと、DELETE 文の WHERE 句にある列のリストと ROW 要素にある列が一致する場合は、新しい DELETE 文が XML 文書の各 ROW 要素に対して作成されます。

XSU を使用した XML 削除の例 20: ROW ごとの削除操作（PL/SQL）

次の削除の例について考えます。

```
create or replace procedure testDelete(xmlDoc IN clob) is
  delCtx DBMS_XMLSave.ctxType;
  rows number;
begin

   delCtx  := DBMS_XMLSave.newContext('scott.emp');
   DBMS_XMLSave.setKeyColumn(delCtx,'EMPNO');

   rows := DBMS_XMLSave.deleteXML(delCtx,xmlDoc);
   DBMS_XMLSave.closeContext(delCtx);
end;
/
```

更新の例で示した XML 文書と同じ文書を使用する場合は、次の 2 つの DELETE 文が生成されます。

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

DELETE 文は、XML 文書内の各 ROW 要素にあるタグ名に基づいて構成されます。

XSU を使用した XML 削除の例 21: キー値の指定による削除（PL/SQL）

DELETE 文に述語としてキー値のみを使用する場合は、setKeyColumn ファンクションを使用してこれを設定できます。

```
create or replace package testDML AS
   saveCtx DBMS_XMLSave.ctxType := null;   -- a single static variable

   procedure insertXML(xmlDoc in clob);
   procedure updateXML(xmlDoc in clob);
   procedure deleteXML(xmlDoc in clob);

 end;
/
```

```
create or replace package body testDML AS

  rows number;

  procedure insertXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.insertXML(saveCtx,xmlDoc);
  end;

  procedure updateXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.updateXML(saveCtx,xmlDoc);
  end;

  procedure deleteXML(xmlDoc in clob) is
  begin
    rows := DBMS_XMLSave.deleteXML(saveCtx,xmlDoc);
  end;

begin
  saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context once..!
  DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');      -- set the key column name.
end;
/
```

ここで、次の形式の DELETE 文が 1 つ生成されます。

```
DELETE FROM scott.emp WHERE EMPNO=?
```

この文は、文書内のすべての ROW 要素に対して使用されます。

XSU を使用した XML 削除の例 22: コンテキスト・ハンドルの再利用（PL/SQL）

前述の挿入、更新および削除の 3 つのすべての例では、複数の操作に同じコンテキスト・ハンドルが使用できます。同じコンテキストを作成したときに指定した同一の表に対してすべての挿入が行われる場合は、同じコンテキストを使用して複数の挿入が実行できます。コンテキストは、更新、削除および挿入を混合するためにも使用できます。

たとえば、次のコードは、ユーザーの入力に基づいて値を挿入、削除または更新するために、コンテキストおよび設定を使用する方法について示します。

この例では、すべてのファンクション・コールで同じコンテキストが使用されるように、PL/SQL パッケージの静的変数を使用してコンテキストを格納します。

```
create or replace package testDML AS
   saveCtx DBMS_XMLSave.ctxType := null;   -- a single static variable
```

```
   procedure insert(xmlDoc in clob);
   procedure update(xmlDoc in clob);
   procedure delete(xmlDoc in clob);

 end;
/

create or replace package body testDML AS

  procedure insert(xmlDoc in clob) is
  begin
    DBMS_XMLSave.insertXML(saveCtx, xmlDoc);
  end;

  procedure update(xmlDoc in clob) is
  begin
    DBMS_XMLSave.updateXML(saveCtx, xmlDoc);
  end;

  procedure delete(xmlDoc in clob) is
  begin
    DBMS_XMLSave.deleteXML(saveCtx, xmlDoc);
  end;

  begin
    saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context once..!
    DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');   -- set the key column name.
  end;
end;
/
```

前述のパッケージで、パッケージ全体（セッション）用にコンテキストを 1 回作成し、挿入、更新および削除の実行に同じコンテキストを再利用します。

> **注意：** キー列 EMPNO が、行を識別する方法として更新と削除の両方に使用されることに注意してください。

このパッケージのユーザーは、3 つのルーチンのどれをコールしても emp 表を更新できます。

```
testDML.insert(xmlclob);
testDML.delete(xmlclob);
testDML.update(xmlclob);
```

これらすべてのコールは、同じコンテキストを使用します。これによって、これらの操作が頻繁に実行される場合にパフォーマンスが向上します。

XSU の高度な使用方法

XSU の Java での例外処理

OracleXMLSQLException クラス

XSU は、処理中に発生したすべての例外をキャッチし、ランタイム例外である oracle.xml.sql.OracleXMLSQLException を発生させます。このため、プログラムがこの例外をキャッチして適切な処置を行う場合は、コールするプログラムはこの例外を常にキャッチする必要がありません。例外クラスは、エラー・メッセージおよび親である例外（ある場合）も取得するファンクションを提供します。たとえば、次に示すプログラムはランタイムの例外をキャッチし、親である例外を取得します。

OracleXMLNoRowsException クラス

この例外は、setRaiseNoRowsException が生成中に OracleXMLQuery クラスに設定されている場合に生成されます。これは、OracleXMLSQLException クラスのサブクラスであり、生成中の行処理に対する終了インジケータとして使用できます。

```
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;

public class testException
{
   public static void main(String argv[])
     throws SQLException
   {
      Connection conn = getConnection("scott","tiger");

      // wrong query this will generate an exception
      OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp where sd
 = 322323");

      qry.setRaiseException(true); // ask it to raise exceptions..!

      try{
        String str = qry.getXMLString();
      }catch(oracle.xml.sql.OracleXMLSQLException e)
      {
        // Get the original exception
        Exception parent = e.getParentException();
        if (parent instanceof java.sql.SQLException)
        {
           // perform some other stuff. Here you simply print it out..
           System.out.println(" Caught SQL Exception:"+parent.getMessage());
        }
```

```
        else
          System.out.println(" Exception caught..!"+e.getMessage());
      }
    }
     // Get the connection given the user name and password..!
     private static Connection getConnection(String user, String passwd)
       throws SQLException
     {
       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
       Connection conn =
           DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
      return conn;
    }
}
```

XSU の PL/SQL での例外処理

XSU の PL/SQL 例外処理の例を示します。

```
declare
  queryCtx DBMS_XMLQuery.ctxType;
  result clob;
  errorNum NUMBER;
  errorMsg VARCHAR2(200);
begin

  queryCtx := DBMS_XMLQuery.newContext('select * from emp where df = dfdf');

  -- set the raise exception to true..
  DBMS_XMLQuery.setRaiseException(queryCtx, true);
  DBMS_XMLQuery.setRaiseNoRowsException(queryCtx, true);

  -- set propagate original exception to true to get the original exception..!
  DBMS_XMLQuery.propagateOriginalException(queryCtx,true);
  result := DBMS_XMLQuery.getXML(queryCtx);

  exception
    when others then
      -- get the original exception
      DBMS_XMLQuery.getExceptionContent(queryCtx,errorNum, errorMsg);
      dbms_output.put_line(' Exception caught ' || TO_CHAR(errorNum)
                  || errorMsg );
end;
/
```

FAQ: XSU

XSU を使用して XML を格納する場合のスキーマ構造

質問

次の XML が customer.xml ファイルにあります。

```
<ROWSET>
 <ROW num="1">
  <CUSTOMER>
   <CUSTOMERID>1044</CUSTOMERID>
   <FIRSTNAME>Paul</FIRSTNAME>
   <LASTNAME>Astoria</LASTNAME>
   <HOMEADDRESS>
    <STREET>123 Cherry Lane</STREET>
    <CITY>SF</CITY>
    <STATE>CA</STATE>
    <ZIP>94132</ZIP>
   </HOMEADDRESS>
  </CUSTOMER>
 </ROW>
</ROWSET>
```

XSU を使用してこの XML を格納する場合に使用するデータベース・スキーマ構造を教えてください。

回答

この例ではレベルが複数であるため（ネストした構造であるため)、オブジェクト・リレーショナル・スキーマを使用する必要があります。前述の XML は、オブジェクト・リレーショナル・スキーマに正規マッピングされます。適切なデータベース・スキーマは次のとおりです。

```
create type address_type as object
 (
 street varchar2(40),
 city varchar2(20),
 state varchar2(10),
 zip varchar2(10)
 );
 /
 create type customer_type as object
 (
customerid number(10),
firstname varchar2(20),
```

```
lastname varchar2(20),
homeaddress address_type
 );
/
create table customer_tab ( customer customer_type);
```

XSU を介して customer.xml をリレーショナル・スキーマにロードする場合は、リレーショナル・スキーマのビューにオブジェクトを作成します。

たとえば、次のすべての情報を含むリレーショナル表があるとします。

```
create table cust_tab
 ( customerid number(10),
   firstname varchar2(20),
   lastname varchar2(20),
   state varchar2(40),
   city varchar2(20),
   state varchar2(20),
   zip varchar2(20)
 );
```

次に、最上位にカスタマ・オブジェクトを持つカスタマ・ビューを次のように作成します。

```
create view customer_view as
select customer_type(customerid, firstname, lastname,
address_type(state,street,city,zip))
from cust_tab;
```

最後に、XSLT を使用して XML をフラット化し、この XML をリレーショナル・スキーマに直接挿入します。ただし、この方法は、あまりお薦めできません。

XSU を使用した複数表への XML データの格納

質問

XSU は、複数表にまたがって XML データを格納できますか？

回答

現在、XSU は、単一表にのみ格納できます。XSU は、XML 文書の正規表現をすべての表およびビューにマップします。ただし、XSU を使用して XML を表にまたがって格納する方法はあります。これを行うには、XSLT を使用していずれかの文書を複数の文書に変換し、これらの文書を別々に挿入します。また、複数表にまたがってビュー（必要に応じてオブジェクト・ビュー）を定義し、そのビューに挿入することでも実行できます。このビューが更新不可能（複雑な結合などのために）な場合は、ビューにまたがって INSTEAD OF トリガーを使用して、挿入を実行できます。

XSU を使用した属性に格納された XML のロード

質問

データのいくつかが属性に格納されている XML のロードに XSU を使用したいのですが、XSU が XML 属性を無視します。対処方法を教えてください。

回答

現時点では、XSLT を使用して XML 文書を変換する必要があります（属性を要素に変更する必要があります）。XSU は、XML からデータベース・スキーマへの正規マッピングを想定しています。このため、柔軟性がなくなり、XSLT を使用する必要がある場合もあります。ただし、この場合、ユーザーがマッピングを指定する必要はありません。

XSU の大文字 / 小文字の区別 : ignoreCase の使用

質問

次の XML 文書（dual.xml）の挿入を試みました。

```
<ROWSET>
   <row>
      <DUMMY>X</DUMMY>
   </row>
</ROWSET>
```

挿入先の表は dual であり、XSU のコマンドラインのフロントエンドを次のように使用しました。

```
java OracleXML putxml -filename dual.xml dual
```

この結果、次のようなエラーが戻りました。

```
oracle.xml.sql.OracleXMLSQLException: No rows to modify -- the row enclosing tag
missing.  Specify the correct row enclosing tag.
```

回答

デフォルトでは、XSU は大文字 / 小文字を区別します。このため XSU は、デフォルトが ROW であるレコード区切りタグを検索しますが、検出されるのは row のみです。一般的なエラーとして、要素タグの大文字 / 小文字の不一致があります。たとえば、dual.xml にあるタグ DUMMY が実際は dummy であった場合、XSU は表 dual に一致する列が見つからないというエラーを戻します。したがって、正確な大文字 / 小文字を使用するか、または ignoreCase 機能を使用します。

XSU を使用した DTD からのデータベース・スキーマの生成

質問

DTD を指定すると、XSU はデータベース・スキーマを生成しますか？

回答

生成しません。DTD には多くの欠点があるため、この機能は使用できません。XML Schema の W3C 勧告がサポートされたら、この機能は実現可能になります。

XSU の Thin ドライバ接続文字列の例

質問

XSU のコマンドラインのフロントエンドを使用しています。接続文字列を渡すと TNS エラーが戻ります。Thin ドライバ接続文字列および OCI8 ドライバ接続文字列の例を教えてください。

回答

次に、JDBC Thin ドライバの接続文字列の例を示します。

```
jdbc:oracle:thin:<user>/<password>@<hostname>:<port number>:<DB SID>;
```

また、データベースにはアクティブな TCP/IP リスナーが必要です。次に、有効な OCI8 接続文字列の例を示します。

```
jdbc:oracle:oci8:<user>/<password>@<hostname>
```

INSERT、DELETE、UPDATE 後の XSU のコミット

質問

XSU は挿入、削除または更新後にコミットしますか？エラーが発生した場合、どうなりますか？

回答

デフォルトでは、XSU は一度に多数の INSERT（または DELETE や UPDATE）文を実行します。バッチ処理で同時に実行される文の数は、`setBatchSize` 機能を使用してオーバーライドできます。

デフォルトでは、XSU は明示的なコミットを行いません。自動コミットをオンにしている（JDBC 接続のデフォルト）場合は、文の各バッチが実行するたびにコミットされます。ユーザーは、自動コミットをオフにし、コミット前に実行する文の数を指定することでこれをオーバーライドできます。コミット前に実行する文の数は、`setCommitBatch` 機能を使用して指定できます。

エラーが発生した場合、XSU はコール前のターゲット表の状態、または XSU への現在のコール中に行われた前回のコミット直後の状態にロールバックします。

XSU を使用して表の行を XML 属性にマップする方法

質問

XSU を使用して表の行を XML 属性にマップする方法を教えてください。

回答

XSU リリース 2.1.0 以上では、特定の列または列のグループを XML 要素ではなく XML 属性にマップできます。このためには、列名に別名を作成し、この別名にアットマーク（@）を付加する必要があります。次に例を示します。

```
* Create a file called select.sql with the following content :
   SELECT empno "@EMPNO", ename, job, hiredate
   FROM emp
   ORDER BY empno

 * Call the XML SQL Utility :
   java OracleXML getXML -user "scott/tiger" \
           -conn "jdbc:oracle:thin:@myhost:1521:ORCL" \
           -fileName "select.sql"

 * As a result, the XML document will look like :
     <?xml version = '1.0'?>
```

```
<ROWSET>
   <ROW num="1" EMPNO="7369">
      <ENAME>SMITH</ENAME>
      <JOB>CLERK</JOB>
      <HIREDATE>12/17/1980 0:0:0</HIREDATE>
   </ROW>
   <ROW num="2" EMPNO="7499">
      <ENAME>ALLEN</ENAME>
      <JOB>SALESMAN</JOB>
      <HIREDATE>2/20/1981 0:0:0</HIREDATE>
   </ROW>
</ROWSET>
```

注意： すべての XML 属性の列を先にマップする必要があります。

XML 文書はストリーム形式で作成されるため、次のような問合せでは期待した結果が生成されません。

```
SELECT ename, empno "@EMPNO", ...
```

現在は、属性に格納された XML データをロードすることはできません。XSLT 変換を使用して、属性を要素に変更する必要があります。XSU は、XML からデータベース・スキーマへの正規マッピングを想定しています。

LOB での XMLGEN.insertXML の使用

質問

次の環境を使用しています。

- OS: Solaris 7
- DB: Oracle8*i* リリース 8.1.5

XSU の `insertXML` プロシージャを使用する必要があります。LOB を使用した経験は多くありません。次のスクリプトの問題点を教えてください。

`lob_temp` 表があります。

```
SQL> desc lob_temp
Name Null? Type
---------------- -------- ------------------  ----------
CHUNK CLOB

SQL> set long 100000
SQL> select * from lob_temp;
```

```
CHUNK
----------------------------------------  ----------
<DOCID> 91739.1 </DOCID>
<SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using  Intermedia
</SUBJECT>
<TYPE> PROBLEM </TYPE>
<CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
<STATUS> PUBLISHED </STATUS>
<CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
<LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
<LANGUAGE> USAENG </LANGUAGE>
```

次の表にも lob_temp 表のデータを挿入する必要があります。

```
SQL> desc metalink_doc
Name Null? Type
---------------- -------- ----------------- ----------
DOCID VARCHAR2(10)
SUBJECT VARCHAR2(100)
TYPE VARCHAR2(20)
CONTENT_TYPE VARCHAR2(20)
STATUS VARCHAR2(20)
CREATION_DATE DATE
LAST_REVISION_DATE DATE
LANGUAGE VARCHAR2(10)
```

スクリプトを次に示します。lob_temp 表からデータを読み取り、XML 文書から抽出したデータを metalink_doc 表に挿入するとします。

```
declare
xmlstr clob := null;
amount integer := 255;
position integer := 1;
charstring varchar2(255);
finalstr varchar2(4000) := null;
ignore_case constant number := 0;
default_date_format constant varchar2(21) := 'DD-MON-YYYY';
default_rowtag constant varchar2(10) := 'MDOC_DATA';
len integer;
insrow integer;
begin
select chunk into xmlstr from lob_temp;
dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
len := dbms_lob.getlength(xmlstr);
while position < len loop
dbms_lob.read(xmlstr,amount,position,charstring);
```

```
 if finalstr is not null then
 finalstr := finalstr||charstring;
 else
 finalstr := charstring;
 end if;
 position := position + amount;
 end loop;
 insrow := xmlgen.insertXML('metalink_doc',finalstr);
 dbms_output.put_line(insrow);
 dbms_lob.close(xmlstr);
 exception
 when others then
 dbms_lob.close(xmlstr);
 dbms_lob.freetemporary(xmlstr);
 end;
 /
```

次のようなエラーが戻ります。

```
ERROR at line 1:
ORA-22275: 指定された LOB ロケータが無効です。
ORA-06512: 485 行 "SYS.DBMS_LOB"
ORA-06512: 31 行
ORA-29532: 不明な Java 例外で Java コールが終了しました :
oracle.xml.sql.OracleXMLSQLException: Expected 'EOF'
```

使用しているユーザーは、両方の表および oraclexmlsqlload.csh の実行時に作成されたすべてのオブジェクトを所有しています。

回答

<ROWSET> および <ROW> タグで XML 文書を表に挿入する必要があります。修正後のプロシージャを次に示します。DATE 書式の解析時に問題が発生するため、VARCHAR2 を使用しています。

```
drop table lob_temp;
 create table lob_temp (chunk clob);
 insert into lob_temp values ('
 <ROWSET>
 <ROW>
 <DOCID> 91739.1 </DOCID>
 <SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using Intermedia
</SUBJECT>
 <TYPE> PROBLEM </TYPE>
 <CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
 <STATUS> PUBLISHED </STATUS>
 <CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
 <LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
```

```
<LANGUAGE> USAENG </LANGUAGE>
</ROW>
</ROWSET>
');

drop table metalink_doc;
create table metalink_doc (
DOCID VARCHAR2(10),
SUBJECT VARCHAR2(100),
TYPE VARCHAR2(20),
CONTENT_TYPE VARCHAR2(20),
STATUS VARCHAR2(20),
CREATION_DATE VARCHAR2(50),
LAST_REVISION_DATE varchar2(50),
LANGUAGE VARCHAR2(10)
);

create or replace procedure prtest as
xmlstr clob := null;
amount integer := 255;
position integer := 1;
charstring varchar2(255);
finalstr varchar2(4000) := null;
ignore_case constant number := 0;
default_date_format constant varchar2(21) := 'DD-MON-YYYY';
default_rowtag constant varchar2(10) := 'MDOC_DATA';
len integer;
insrow integer;
begin

select chunk into xmlstr from lob_temp;
dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
len := dbms_lob.getlength(xmlstr);

while position < len loop
dbms_lob.read(xmlstr,amount,position,charstring);
if finalstr is not null then
finalstr := finalstr||charstring;
else
finalstr := charstring;
end if;
position := position + amount;
end loop;

insrow := xmlgen.insertXML('metalink_doc',finalstr);
dbms_output.put_line(insrow);
```

```
IF DBMS_LOB.ISOPEN(xmlstr) = 1 THEN
dbms_lob.close(xmlstr);
END IF;

exception
when others then
IF DBMS_LOB.ISOPEN(xmlstr)=1 THEN
dbms_lob.close(xmlstr);
END IF;
end;
/
show err
```

返答

動作しました。ありがとうございました。

8

Oracle Text を使用した XML データの検索

この章では、次の Oracle Text（*inter*Media Text/Context）機能について説明します。

- XML 文書にセクション・グループおよび索引を作成する方法
- Oracle Text を使用して XML 問合せアプリケーションを構築し、XML 文書のデータを検索して取り出す方法

この章の内容は次のとおりです。

Oracle Text の概要

注意： Oracle Text は、サーバーベースの実装です。

参照： http://otn.oracle.com/products/text を参照してください。

Oracle Text を使用して、XML 文書を検索できます。これは、Oracle に格納されたすべてのテキストまたはドキュメントを索引付けすることによって、Oracle を拡張します。また、OS 内のドキュメント（フラット・ファイル）および URL も検索できます。

Oracle Text を使用すると、次の操作を行うことができます。

- 一般的な標準 SQL を使用した**内容ベースの問合せ**（特定のワードを含むテキストおよびドキュメントの検索など）。
- Oracle を使用して、従来の関連情報を使用した統合的な方法によって、**テキストおよびドキュメントを管理する**ためのファイルベースのテキスト・アプリケーション。
- 英語ドキュメントの**概念検索**。
- テーマ / 要点パッケージを使用した英語ドキュメントの**テーマ分析**。
- **検索で一致したワードのハイライト**。Oracle Text を使用すると、ドキュメントを様々な方法でレンダリングできます。たとえば、問合せ用語（英語の場合、ワード問合せのワードまたは ABOUT 問合せのテーマ）をハイライトさせてドキュメントを表示できます。これを行うには、CTX_DOC.MARKUP または HIGHLIGHT プロシージャを使用します。
- Oracle Text PL/SQL パッケージを使用した**ドキュメントの表示およびシソーラスのメンテナンス**。

Oracle Text は、他の *inter*Media 製品（Web コンテンツ管理アプリケーションのためのイメージ、オーディオ、ビデオおよび地理検索サービス）とともにパッケージ化されています。

Oracle Text を使用しなくても、データベースに格納された XML データを直接問い合せることはできます。ただし、Oracle Text を使用すると、問合せのパフォーマンスが向上します。

参照： 次のマニュアルまたは Web サイトを参照してください。

- 『Oracle Text リファレンス』
- 『Oracle Text アプリケーション開発者ガイド』
- http://otn.oracle.com/products/text

Oracle Text へのアクセス

Oracle Text を含む *inter*Media は、Oracle Standard Edition、Enterprise Edition および Personal Edition のすべてのライセンスに付属する標準機能です。この機能は、インストール時に選択する必要があります。特別なインストール手順は不要です。

Oracle Text は、基本的に、CTXSYS が所有する一連のスキーマ・オブジェクトです。これらのオブジェクトは、Oracle カーネルにリンクされます。スキーマ・オブジェクトは、Oracle のインストールに含まれます。

Oracle Text の詳細な例

Oracle Text およびセクション・グループ索引の作成の詳細な例は、次の Web サイトを参照してください。

http://otn.oracle.com/products/text

この章の例に関する前提

XML テキストは、文字セマンティクスを持つ Oracle データベース表内の VARCHAR2 型または CLOB 型です。Oracle Text は、ファイル・システムまたは URL 上のドキュメントも処理できますが、この章ではこれらのドキュメント・タイプについては考慮しません。

この章に含まれる例を簡単にするために、Oracle Text オプションの一部分について考えてみます。この章の例では、次のことを前提としています。

- すべての XML データが、US-ASCII の 7 ビット・キャラクタ・セットを使用して表されます。
- 「*」などの文字を空白として処理するか、またはワードの一部として処理するかについての問題は、含まれません。
- テキスト索引を実装する Oracle スキーマ・オブジェクトの記憶特性については、考慮しません。
- CREATE INDEX 文または ALTER INDEX 文内の SECTION GROUP パラメータに注目します。CREATE INDEX 文および ALTER INDEX 文に使用できる他のパラメータ・タイプは、DATASTORE、FILTER、LEXER、STOPLIST および WORDLIST です。

  参照： これらのパラメータ・タイプの詳細は、『Oracle Text リファレンス』を参照してください。

  次に、CREATE INDEX 文で SECTION GROUP パラメータを使用した例を示します。

```
CREATE INDEX my_index
  ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' ) ;
```

- 特に、AUTO_SECTION_GROUP、XML_SECTION_GROUP および PATH_SECTION_GROUP を使用することに注目します。
- XML データ（タグ付けまたはマークアップされたデータ）の処理方法に注目します。Oracle Text は、他にも様々な種類のデータを処理します。

参照：『Oracle Text アプリケーション開発者ガイド』を参照してください。

Oracle Text のユーザーおよびロール

Oracle Text では、次のユーザーおよびロールを使用できます。

- ユーザーを管理する CTXSYS ユーザー
- Oracle Text プリファレンスを作成および削除して Oracle Text PL/SQL パッケージを使用する CTXAPP ロール

CTXSYS ユーザー

このユーザーは、インストール時に作成されます。Oracle Text ユーザーをこのユーザーとして管理します。CTXSYS ユーザーには、次の権限があります。

- システム定義のプリファレンスを変更する権限
- 他のユーザーのプリファレンスを削除および変更する権限
- CTX_ADM PL/SQL パッケージのプロシージャをコールして、サーバーの起動およびシステム・パラメータの設定を行う権限
- ctxsrv サーバーを起動する権限
- すべてのシステム定義のビューを問い合せる権限
- CTXAPP ロールを使用してユーザーのすべてのタスクを実行する権限

CTXAPP ロール

すべてのユーザーは、Oracle Text 索引を作成したり、テキスト問合せを発行することができます。その他のタスクを実行する場合は、CTXAPP ロールを使用してください。これは、システム定義のロールです。このロールを使用して、次のタスクを実行できます。

- Oracle Text プリファレンスの作成および削除
- CTX_DDL パッケージなどの Oracle Text PL/SQL パッケージの使用

CONTAINS 演算子を使用した問合せ

Oracle Text の主な用途は、CONTAINS 演算子を使用することです。CONTAINS 演算子は、テキスト問合せ用の問合せ式を指定するために、SELECT 文の WHERE 句で使用されます。

参照： 8-23 ページの「Oracle Text を使用した XML 問合せアプリケーションの構築」を参照してください。

CONTAINS 構文

CONTAINS 構文は次のとおりです。

```
...WHERE CONTAINS([schema.]column,text_query VARCHAR2,[label NUMBER])
```

パラメータは次のとおりです。

表 8-1 CONTAINS 演算子 : 構文の説明

構文	説明
[schema.]column	検索するテキスト列を指定します。この列には、それに対応付けられた Text 索引が必要です。
text_query	列内での検索を定義する問合せ式を指定します。
label	オプションで、CONTAINS 演算子によって生成されたスコアを識別するラベルを指定します。

CONTAINS は、選択された各行について、そのドキュメント行が問合せにどの程度関連しているかを示す 0 ～ 100 の数値を戻します。数値 0（ゼロ）は、Oracle がその行内で一致するデータを検索しなかったことを意味します。このスコアは、SCORE 演算子を使用して取得できます。

注意： この数値を取得するには、ラベルを指定して SCORE 演算子を使用してください。

単純な SELECT 文の使用

次の例は、SELECT 構文で CONTAINS 演算子を使用する方法を示します。

```
SELECT id FROM my_table
   WHERE
   CONTAINS (my_column, 'receipts') > 0
```

CONTAINS 関数の 'receipts' パラメータは、テキスト問合せ式とも呼びます。

注意： CONTAINS 関数を含む SQL 文を実行するには、Text 索引が必要です。

関連性を取得するラベルを指定した SCORE 演算子の使用

次の例は、テキスト列内で「Oracle」というワードを含むすべてのドキュメントを検索します。各行のスコアは、SCORE 演算子で 1 というラベルを使用して選択されます。

```
SELECT SCORE(1), title from newsindex
          WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

CONTAINS 演算子の後には、> 0 の構文が続く必要があります。この構文は、CONTAINS 演算子が計算したスコアの値が 0（ゼロ）より大きい場合に、その行が選択されることを指定します。

SELECT 句などで SCORE 演算子がコールされると、演算子は例に示すとおり、ラベルの値を参照する必要があります。

問合せをドキュメント・セクションに限定する WITHIN 演算子の使用

HTML や XML のように内部構造を持つドキュメントの場合、索引付けを行う前に、埋込みタグを使用してドキュメント・セクションを定義できます。これによって、WITHIN 演算子を使用してセクション内で問合せできます。

注意： これは、XML_SECTION_GROUP のみで実行できます。AUTO_SECTION_GROUP または PATH_SECTION_GROUP では実行できません。

セクションを、セクション・グループの一部として定義し、WITHIN 演算子を使用して問合せをドキュメント・セクションに限定します。ドキュメント・セクションは、次のいずれかのタイプに指定できます。

- ゾーン・セクション
- フィールド・セクション
- 属性セクション
- 特殊セクション（文または段落）

セクション問合せのための WITHIN 構文

セクションを問い合せる WITHIN 構文は次のとおりです。

```
expression WITHIN section
```

この構文は、セクション内で式を検索します。XML_SECTION_GROUP を使用する場合、事前定義されているゾーン、フィールドまたは属性セクションには次の制限事項が適用されます。

- **ゾーン**・セクションの場合、ゾーン・セクションまたは特殊セクションを指定した 1 つ以上の WITHIN 演算子（ネストした WITHIN）を式に含めることができます。
- **フィールド**・セクションまたは**属性**セクションの場合、式に含めることができる WITHIN 演算子は 1 つのみです。

WITHIN 句は、組み合せてネストすることができます。XML のセクションをより詳細に検索するには、CONTAINS を使用した SELECT 文内に WITHIN 句を指定します。

WITHIN 演算子の制限事項

WITHIN 演算子には、次の制限事項があります。

- WITHIN 句を句に埋め込むことはできません。たとえば、term1 WITHIN section term2 のような句を作成することはできません。
- WITHIN 演算子を「$」、「!」、「*」などの拡張演算子と組み合せることはできません。
- WITHIN 演算子は予約語であるため、検索するには、中括弧を使用してそのワードをエスケープする必要があります。

> **参照：**『Oracle Text リファレンス』を参照してください。

XML_SECTION_GROUP、AUTO_SECTION_GROUP または PATH_SECTION_GROUP のいずれかのセクション・グループ・タイプを使用して索引付けする場合、属性セクション内で問合せを行うことができます。

次の XML 文書について考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

XML_SECTION_GROUP

XML_SECTION_GROUP を使用する場合、CTX_DDL.ADD_ATTR_SECTION を使用して、属性セクションに任意の名前を付けることができます。

セクション title@book を属性セクション booktitle として定義するには、次のいずれかの方法を使用します。

- **CTX_DLL.ADD_ATTR_SECTION プロシージャを使用します。**構文は次のとおりです。

  ```
  CTX_DDL.ADD_ATTR_SECTION(
    group_name      in    varchar2,
    section_name    in    varchar2,
    tag             in    varchar2);
  ```

  TITLE 属性を属性セクションとして定義するには、次のとおり、XML_SECTION_GROUP を作成し、属性セクションを定義します。

  ```
  EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
  ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
  ```

  TITLE 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

  ```
  'Cities within booktitle'
  ```

- **索引付けした後に、ALTER INDEX 文を使用して動的に定義します。**ALTER INDEX 文の構文は次のとおりです。

  ```
  ALTER INDEX [schema.]index REBUILD [ONLINE] [PARAMETERS (paramstring)];
  ```

  ここで、paramstring は次のとおりです。

  ```
  paramstring = 'replace [datastore datastore_pref]
                         [filter filter_pref]
                         [lexer lexer_pref]
                         [wordlist wordlist_pref]
                         [storage storage_pref]
                         [stoplist stoplist]
                         [section group section_group]
                            [memory memsize]
  |    ... add attr section section_name tag tag@attr
  |    add stop section tag'
  ```

`add attr section section_name tag tag@attr` 句は、属性セクション section_name を既存の索引に動的に追加します。XML のタグおよび属性を、タグ名 @ 属性名という形式で指定する必要があります。XML セクション・グループのみに属性セクションを追加できます。

追加された section_name セクションは、この操作の後に索引付けされたドキュメントのみに適用されます。したがって、変更を有効にするには、タグを含む既存のドキュメントを手動で再索引付けする必要があります。この文では索引は再作成されません。

AUTO_SECTION_GROUP または PATH_SECTION_GROUP

AUTO_SECTION_GROUP または PATH_SECTION_GROUP を使用して XML 文書を索引付けする場合、「属性名 @ タグ名」という名前の属性セクションが自動的に作成されます。

属性セクション `booktitle` 内で Tale を検索するには、次のとおり、SELECT 文に WITHIN 句を含めます。

- XML_SECTION_GROUP を使用している場合

```
... WHERE CONTAINS ('Tale WITHIN booktitle')>0;
```

- AUTO_SECTION_GROUP または PATH_SECTION_GROUP を使用している場合

```
... WHERE CONTAINS ('Tale WITHIN title@book')>0;
```

参照： 8-24 ページの「DOCTYPE 間のタグの区別」を参照してください。

属性セクションの問合せに対する制約

次の制約が、属性セクション内での問合せに適用されます。

- 通常の属性テキストの問合せは、WITHIN 句で修飾されていない場合、動作しません。次の XML 文書があると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

  Tale の問合せは、'WITHIN title@book' で修飾されていない場合、動作しません。

- 属性セクションは、ネストした WITHIN 問合せでは使用できません。
- 句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

```
....Now is the time for all good <word type="noun"> men </word> to come to the
aid......
```

  検索を行うと、通常の問合せによって「good men」が検出され、間に挿入されている属性テキストは無視されます。

問合せ検索用の INPATH 演算子または HASPATH 演算子の使用

INPATH 演算子および HASPATH 演算子は、PATH_SECTION_GROUP を使用して作成した索引にのみ使用します。

PATH_SECTION_GROUP を使用すると、パス検索ができます。パス検索は WITHIN 演算子の構文を拡張するため、セクション名のオペランド（右側）は、セクション名ではなくパスになります。

表 8-2 に、パス検索に INPATH 演算子を使用する様々な方法を示します。

表 8-2 INPATH 演算子を使用した XML 文書のパス検索

パス検索機能	構文	説明
単純なタグ検索	virginia INPATH (STATE) virginia INPATH (//STATE)	<STATE> と </STATE> の間に「virginia」というワードがあるすべての文書を検索します。文書構造のすべてのレベルにある STATE 要素が検索されます。
大文字 / 小文字区別	virginia INPATH (STATE) virginia INPATH (State)	パス検索では、タグ名および属性名の大文字 / 小文字が区別されます。virginia INPATH STATE では、<STATE>virginia</STATE> は検索されますが、<State>virginia</State> は検索されません。後者を検索するには、virginia INPATH State として検索する必要があります。
最上位タグの検索	virginia INPATH (Legal) virginia INPATH (/Legal)	最上位タグである Legal 要素に「virginia」があるすべての文書を検索します。「Legal」は、文書の最上位タグである必要があります。間に挿入されている他のタグに関係なく、「virginia」はこのタグのどこかにあります。次に例を示します。 <?xml version=″1.0″ standalone=″yes″?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address> ... </Legal>

表 8-2 INPATH 演算子を使用した XML 文書のパス検索（続き）

パス検索機能	構文	説明
任意のレベルのタグの検索	virginia INPATH (//Address)	「virginia」が「Address」タグのどこかにあります。他のタグの間にある場合もあります。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState>... </Legal>
直接の親子関係のパス検索	virginia INPATH (//CourtInformation/Location)	CourtInformation 要素の直接の子である Location 要素に「virginia」が含まれるすべての文書を検索します。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address>... </CourtInformation>

表 8-2 INPATH 演算子を使用した XML 文書のパス検索（続き）

パス検索機能	構文	説明
シングルレベル・ワイルド・カード検索	virginia INPATH(A/*/B) 'virginia INPATH (//CaseCaption/*/Location)'	A 要素の孫である B 要素に「virginia」があるすべての文書を検索します。たとえば、<A><D><B>virginia</B></D></A> などです。中間要素は索引付けされた XML タグである必要はありません。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState>... </Legal>

表 8-2 INPATH 演算子を使用した XML 文書のパス検索（続き）

パス検索機能	構文	説明
マルチレベル・ワイルド・カード検索	'virginia INPATH (Legal/*/Filing/*/*/CourtInformation)'	「Legal」が最上位タグで、タグ・レベルが「Legal」と「Filing」の間に 1 つのみ、「Filing」と「CourtInformation」の間に 2 つのみある必要があります。「virginia」は「CourtInformation」内のどこかに表示されています。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address> </Location> <CourtName> IN THE CIRCUIT COURT OF LOUDOUN COUNTY </CourtName> </CourtInformation>....
子孫の検索	virginia INPATH(A//B)	A 要素の子孫（任意のレベル）である B 要素に「virginia」があるすべての文書を検索します。
属性の検索	virginia INPATH(A/@B)	A 要素の B 属性に「virginia」があるすべての文書を検索します。
子孫 / 属性の存在テスト	virginia INPATH (A[B])	B 要素を直接の子とする A 要素に「virginia」があるすべての文書を検索します。 ■ virginia INPATH A[.//B] - B 要素を子孫（任意のレベル）とする A 要素に「virginia」があるすべての文書を検索します。 ■ virginia INPATH A[@B] - B 属性を持つ A 要素に「virginia」があるすべての文書を検索します。

表 8-2 INPATH 演算子を使用した XML 文書のパス検索（続き）

パス検索機能	構文	説明
属性値のテスト	virginia INPATH A[@B = "foo"]	値が「foo」である B 属性を持つ A 要素に「virginia」があるすべての文書を検索します。 ■ テストには等式のみサポートされています。範囲演算子および範囲関数はサポートされていません。 ■ 等式の左側には属性またはタグを指定する必要があります。リテラルは指定できません。 ■ 右側にはリテラルを指定する必要があります。タグおよび属性は指定できません。
内部等値	次のことを意味します。 デフォルトのレクサーおよびストップリストを使用すると、virginia INPATH (A[@B = "pot of gold"]) は次のいずれかに一致します。 ■ <A B="POT OF GOLD">virginia</A> ■ <A B="POT BLACK GOLD”>virginia</A> デフォルトでは、レクサーは大文字 / 小文字を区別しないため、「pot」は「POT」に一致します。 ■ <A B=" Pot OF Gold “>virginia</A> デフォルトでは、「of」はストップワードであるため、問合せでは、その位置にある任意のワードに一致します。	内部等値（8-16 ページの「パス検索への HASPATH 演算子の使用」を参照）は、テストの評価に使用されます。 空白は、テキスト索引付けではほとんの場合無視されます。レクサーは大文字 / 小文字を区別しません。 <A B=”pot_of_gold”>virginia</A> アンダースコアはアルファベット以外の文字であり、デフォルトでは結合文字ではありません。そのため、ほとんどの場合は空白として処理され、文字列は 3 つのワードに分割されます。
数値等値	virginia INPATH (A[@B = 5])	数値リテラルは、使用できますが、テキストとして処理されます。内部等値を使用して評価します。これは、問合せが一致しないことを意味します。 <A B=”5.0”>virginia</A> は、小数点付きの「5.0」が整数の 5 と同じであるとみなされないため、A[@B=5] に一致しません。

表 8-2 INPATH 演算子を使用した XML 文書のパス検索（続き）

パス検索機能	構文	説明
論理積のテスト	virginia INPATH (A[B AND C]) virginia INPATH (A[B AND @C = "foo"])...	述語を論理積として結合できます。
パスのテストとノードのテストの結合	virginia INPATH (A[@B = "foo"]/C/D) virginia INPATH(A//B[@C]/D[E])...	ノードのテストは、パス内のすべてのノードに適用できます。

パス検索への HASPATH 演算子の使用

> **注意：** HASPATH 演算子の動作は、XMLType の Existsnode() 演算子の動作に類似しています。第 5 章「XML に対するデータベース・サポート」を参照してください。

HASPATH 演算子は、PATH_SECTION_GROUP を使用して索引を作成した場合にのみ使用します。HASPATH 演算子の構文は次のとおりです。

- **WHERE CONTAINS(column, 'HASPATH(path)'...):** HASPATH は、XML 文書セットを検索し、パスが存在するすべての文書に対してスコア値 100 を戻します。親および子であるパスは、A/B/C のように、スラッシュ（/）文字で区切ります。次の問合せを行うとします。

  ```
  ...WHERE CONTAINS (col,'HASPATH(A/B/C)')>0;
  ```

  この問合せは、検索した次の文書に対してスコア 100 を戻します。

  ```
  <A><B><C>Virginia</C></B></A>
  ```

  この場合、「Virginia」は参照されません。

- **WHERE CONTAINS(column, 'HASPATH(A="value")'...):** HASPATH 句は、XML 文書セットを検索し、内容値および値のみを含む要素 A があるすべての文書に対してスコア値 100 を戻します。HASPATH を使用して、等価性をテストできます。これは、HASPATH 演算子の「セクション等価性テスト」機能です。次の問合せを行うとします。

  ```
  ...WHERE CONTAINS virginia INPATH A
  ```

この問合せは、<A>virginia</A> のみでなく <A>virginia state</A> も検索します。問合せを virginia という用語に限定するには、HASPATH 演算子によるセクション等価性テストを使用します。次に例を示します。

```
... WHERE CONTAINS (col,'HASPATH(A="virginia")'
```

この問合せは <A>virginia</A> のみを検索してスコア値 100 を戻します。<A>virginia state</A> は検索しません。

Oracle Text を使用した XML 文書の検索

Oracle Text を使用して XML 文書からデータを検索および取得するには、次のタスクを行う必要があります。

1. セクション・グループを作成します。
2. 作成したセクション・グループに基づいて Oracle Text 索引を作成します。
3. CONTAINS 演算子を使用して、問合せアプリケーションを構築します。

セクション・グループおよび Oracle Text 索引を作成する前に、必要なロールを決定し、適切な権限を付与する必要があります。8-5 ページの「Oracle Text のユーザーおよびロール」を参照し、適切な権限を付与してください。

データを作成および準備したら、次の手順に進みます。「手順 1: セクション・プリファレンスの作成」を参照してください。

作成したセクション・プリファレンスを使用して、Oracle Text 索引を作成します。「手順 2: 手順 1 で作成したセクション・プリファレンスを使用した索引の作成」を参照してください。

これで、問合せアプリケーションの構築が完了します。「Oracle Text を使用した XML 文書の検索」を参照してください。

まず、必要なロールを決定します。『Oracle Text リファレンス』および 8-5 ページの「Oracle Text のユーザーおよびロール」を参照して、次のとおり、適切な権限を付与します。

```
CONNECT system/manager
GRANT ctxapp to scott;
CONNECT scott/tiger
```

手順 1: セクション・プリファレンスの作成

ここでは、PATH_SECTION_GROUP、XML_SECTION_GROUP および AUTO_SECTION_GROUP を使用してセクション・プリファレンスを作成する基本的な方法を説明します。

表 8-3 に、XML 文書の索引付けに使用できるセクション・グループを示します。

表 8-3 Oracle Text のセクション・グループの比較

セクション・グループ	説明
XML_SECTION_GROUP	このグループ・タイプは、XML 文書を索引付けし、XML 文書内のセクションを定義するために使用します。
AUTO_SECTION_GROUP	このグループ・タイプは、XML 文書内の開始タグ / 終了タグの各組用のゾーン・セクションを自動的に作成するために使用します。XML タグから導出されるセクション名は、XML の場合と同様、大文字 / 小文字が区別されます。属性セクションは、属性を含む XML タグ用に自動的に作成されます。属性セクションには、「属性名 @ タグ名」という形式の名前が付けられます。停止セクション、空タグ、処理命令およびコメントは索引付けされません。自動セクション・グループには、次の制限事項が適用されます。 ■ 自動セクション・グループには、ゾーン、フィールドまたは特殊セクションを追加できません。 ■ 自動セクション作成では、XML 文書タイプ（ルート要素）は索引付けされません。ただし、停止セクションは、文書タイプで定義できます。 ■ 接頭辞および名前空間を含めた索引付きタグの長さは、64 文字以下である必要があります。64 文字より長いタグは、索引付けされません。
PATH_SECTION_GROUP	このグループ・タイプは、XML 文書を索引付けするために使用します。AUTO_SECTION_GROUP と同じように動作します。このセクション・グループを使用すると、INPATH 演算子および HASPATH 演算子によるパス検索を実行できます。問合せは、タグ名および属性名の大文字 / 小文字を区別します。 **AUTO_SECTION_GROUP との類似点** ■ ドキュメントは XML であると想定されます。 ■ デフォルトで、すべてのタグおよび属性が索引付けされます。 ■ 停止セクションを追加して、特定のタグが索引付けされないようにできます。 ■ 追加できるセクションは停止セクションのみです。ゾーン・セクション、フィールド・セクションおよび特殊セクションは追加できません。 ■ XML 文書のコレクションを索引付けする場合、セクションは自動的に定義されるため、ユーザーが明示的に定義する必要はありません。 **AUTO_SECTION_GROUP との相違点** ■ 新しい INPATH 演算子および HASPATH 演算子を使用して、問合せ時にパス検索を実行できます（8-16 ページの「例 : Oracle Text を使用したオンライン FAQ リストの検索」および「パス検索への HASPATH 演算子の使用」を参照）。 ■ 問合せ時に、タグ名および属性名の大文字 / 小文字が区別されます。

注意： AUTO_SECTION_GROUP または PATH_SECTION_GROUP を使用して XML 文書のコレクションを索引付けする場合、セクションは索引付け中に自動的に定義されるため、ユーザーが明示的に定義する必要はありません。

使用するセクション・グループの決定

アプリケーションに最適なセクション・グループを決定する方法は、アプリケーションによって異なります。表 8-4 に、XML 文書の索引付けに使用するセクション・グループ（XML_、AUTO_ または PATH_）を決定するための一般的なガイドライン、およびそのセクション・グループが適切な理由を示します。

表 8-4 XML_、AUTO_ または PATH_ セクション・グループ選択のガイドライン

アプリケーションの条件	XML_section_...	AUTO_section_...	PATH_section_...
XPath 検索機能を使用している			X
XML 文書のレイアウトおよび構造を理解し、ユーザーが検索を行うと想定されるセクションを事前定義できる	X		
ユーザーが検索するタグを特定できない		X	
一般的な問合せのパフォーマンス	最も速い	XML_section_... より少し遅い	AUTO_section_... より少し遅い
一般的な索引付けのパフォーマンス	最も速い	XML_section_... より少し遅い	AUTO_section_... より少し遅い
索引サイズ	最も小さい	XML_section_... より少し大きい	AUTO_section_... より少し大きい
その他の機能	1 つ以上の DTD のタグを 1 つのセクションにマップするようにマッピングを定義できます。DTD の拡張およびデータ集計に適しています。	最も単純です。マッピングを定義する必要がありません。add_stop_section を使用していくつかのセクションを無視できます。	より高度な XPath に類似した問合せのために設計されています。

XML_SECTION_GROUP を使用したセクション・プリファレンスの作成

次のコマンドは、XML_SECTION_GROUP グループ・タイプを使用して、xmlgroup というセクション・グループを作成します。

```
EXEC ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
```

CTX_DDL.ADD_SECTION を使用して、このグループにセクションを追加できます。次のとおり、TITLE 属性を指定して BOOK タグを定義する XML ファイルについて考えてみます。

```
<BOOK TITLE="Tale of Two Cities">  It was the best of times. </BOOK>
```

TITLE 属性を属性セクションとして定義するには、次のとおり、XML_SECTION_GROUP を作成し、属性セクションを定義します。

```
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
```

TITLE 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

```
'Cities within booktitle'
```

AUTO_SECTION_GROUP を使用したセクション・プリファレンスの作成

索引付け操作を設定し、セクション・グループ AUTO_SECTION_GROUP を使用して XML 文書からセクションを自動的に作成することができます。XML タグに対するゾーン・セクションは、Oracle が作成します。属性を持つタグには属性セクションが作成されます。これらの属性セクションには、「タグ名 @ 属性名」という形式の名前が付けられます。

次のコマンドは、AUTO_SECTION_GROUP グループ・タイプを使用して、autogroup というセクション・グループを作成します。このセクション・グループは、自動的に XML 文書内のタグからセクションを作成します。

```
EXEC ctx_ddl.create_section_group('autogroup', 'AUTO_SECTION_GROUP');
```

注意： XML セクション・グループのみに属性セクションを追加できます。AUTO_SECTION_GROUP を使用すると、属性セクションが自動的に作成されます。自動的に作成された属性セクションには、「タグ名 @ 属性名」という形式の名前が付けられます。

PATH_SECTION_GROUP を使用したセクション・プリファレンスの作成

パス・セクション検索を有効化するには、PATH_SECTION_GROUP を使用して XML 文書を索引付けします。次に例を示します。

```
EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');
```

手順 2: 手順 1 で作成したセクション・プリファレンスを使用した索引の作成

プリファレンスの作成に使用したセクション・グループに基づいて、索引を作成します。

XML_SECTION_GROUP を使用した索引の作成

XML_SECTION_GROUP を使用している場合、XML 文書を索引付けするには、次の文を使用します。

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
     parameters('section group xmlgroup');
```

> **参照：** 8-21 ページの「Oracle Text の例 1: XML_SECTION_GROUP を使用した索引の作成」を参照してください。

AUTO_SECTION_GROUP を使用した索引の作成

次の文は、AUTO_SECTION_GROUP を使用して、XML ファイルを含む列に myindex という索引を作成します。

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group autogroup');
```

PATH_SECTION_GROUP を使用した索引の作成

PATH_SECTION_GROUP を使用している場合、XML 文書を索引付けするには、次の文を使用します。

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group xmlpathgroup');
```

> **参照：** CTX_DDL の詳細は、『Oracle Text リファレンス』を参照してください。

Oracle Text の例 1: XML_SECTION_GROUP を使用した索引の作成

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');

/* ADDING A FIELD SECTION */
EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
  (  group_name   =>'my_section_group',
     section_name =>'author',/* do this for EVERY tag used after "WITHIN" */
     tag          =>'author'
  );

EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
      (  group_name   =>'my_section_group',
         section_name =>'document',/*do this for EVERY tag after "WITHIN" */
```

```
         tag          =>'document'
      );

  ...
/
/* ADDING AN ATTRIBUTE SECTION */
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');

/* The more sections you add to your index, the longer your search will take.*/
/* Useful for defining attributes in XML documents as sections. This allows*/
/* you to search XML attribute text using the WITHIN operator.*/
/*  The section name:
/* ** Is used for WITHIN queries on the attribute text.
   ** Cannot contain the colon (:) or dot (.) characters.
   ** Must be unique within group_name.
   ** Is case-insensitive.
   ** Can be no more than 64 bytes.
   **  The tag specifies the name of the attribute in tag@attr format. This is
       case-sensitive. */
/* Names used as arguments of the keyword WITHIN can be different from the
   actual XML tag names. Many tags can be mapped to the same name at query
   time.*/

/* ADDING A ZONE SECTION */
/* If You have an XML document that contains the <book> tag declared for */
/* different document types. You can create a distinct book section for each */ /*
document type. If mydocname is declared in your DTD as an XML document */
/* type (root element) as follows: */

<!DOCTYPE mydocname ... [...

/* Within mydocname, element <book> is declared. For this tag, you can create */ /*
a section named, mybooksec, that's sensitive to the tag's document type as */ /*
follows: */

EXEC ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');

/* Call CTX_DDL.Add_Zone_Section for each tag in your XML document that you need to
search on. */

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS(my_column, 'smith WITHIN author') > 0;
```

Oracle Text の例 2: AUTO_SECTION_GROUP を使用した索引の作成

```
ctx_ddl_create_section_group('auto', 'AUTO_SECTION_GROUP');

CREATE INDEX myindex ON docs(xmlfile_column)
    INDEXTYPE IS ctxsys.context
    PARAMETERS ('filter ctxsys.null_filter SECTION GROUP auto');

SELECT xmlfile_column FROM docs
    WHERE CONTAINS (xmlfile_column, 'virginia WITHIN title')>0;
```

Oracle Text の例 3: PATH_SECTION_GROUP を使用した索引の作成

```
EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');

CREATE INDEX myindex ON xmldocs(xmlfile_column)
    INDEXTYPE IS ctxsys.context
    PARAMETERS ('section group xmlpathgroup');

SELECT xmlfile_column FROM xmldocs
... WHERE CONTAINS (column, 'Tale WITHIN title@book')>0;
```

Oracle Text を使用した XML 問合せアプリケーションの構築

この項の内容は次のとおりです。

- 属性セクション内での問合せ
- XML 文書の問合せ
- Oracle Text を使用した問合せアプリケーションの作成手順
- ドキュメント・タイプが区別される XML 文書でのセクションの作成

参照： 次の項またはマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』
- 8-42 ページの「例 : Oracle Text を使用したオンライン FAQ リストの検索」

XML 文書の問合せ

DOCTYPE 間のタグの区別

以前のリリースでは、XML セクション・グループは、異なる DTD のタグを区別できませんでした。たとえば、次の連絡先情報を格納するための DTD があるとします。

```
<!DOCTYPE contact>
<contact>
  <address>506 Blue Pool Road</address>
  <email>dudeman@radical.com</email>
</contact>
```

適切なセクションは、次のようになります。

```
ctx_ddl.add_field_section('mysg','email',  'email');
ctx_ddl.add_field_section('mysg','address','address');
```

これは、同じ表内に異なる種類のドキュメントがない場合は問題ありません。

```
<!DOCTYPE mail>
 <mail>
   <address>dudeman@radical.com</address>
</mail>
```

この場合、本来は住所用であったアドレス・セクションが、タグが重複するために、電子メール・アドレスを取得するようになります。

DOCTYPE 制限の指定によるタグの区別

Oracle8*i* リリース 8.1.5 以上では、DOCTYPE 制限を指定して、DOCTYPE 間のタグを区別できます。次のとおり、タグの前のカッコ内にドキュメント・タイプを指定します。

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','(contact)address');
ctx_ddl.add_field_section('mysg','email','(mail)address');
```

この場合、XML セクション・グループはアドレス・タグを認識し、そのタグを、ドキュメント・タイプが、`contact` である場合は住所セクションとして、ドキュメント・タイプが `mail` である場合は電子メール・セクションとして索引付けします。

セクション・グループ内での DOCTYPE 制限タグおよび未制限タグの混在

次のとおり、1 つのセクション・グループ内に DOCTYPE 制限タグおよび未制限タグが両方あると想定します。

```
ctx_ddl.add_field_section('mysg','sec1','(type1)tag1');
ctx_ddl.add_field_section('mysg','sec2','tag1');
```

この場合、制限タグは該当する DOCTYPE の場合に適用され、未制限タグはそれ以外のすべてのドキュメント・タイプの場合に適用されます。

これは、問合せには影響しません。問合せは、タグではなく、セクション名で行われるため、電子メール・アドレスの問合せは、次のとおり行われます。

```
radical WITHIN email
```

この場合、2 つの異なる種類のタグを同じセクション名にマップしているため、どちらのタグが電子メール・アドレスを表すために使用されているかに関係なく、ドキュメントが検索されます。

属性セクション内での問合せ

次のいずれかのセクション・グループ・タイプを使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

- XML_SECTION_GROUP
- AUTO_SECTION_GROUP
- PATH_SECTION_GROUP

次の XML 文書について考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

title@book セクションを属性セクションのタイトルとして定義できます。これは、CTX_DLL.ADD_ATTR_SECTION プロシージャを使用して行うか、または ALTER INDEX を使用して索引付けした後に動的に行います。

注意：

- AUTO_SECTION_GROUP を使用して XML 文書を索引付けする場合、システムは、自動的に属性セクションを作成し、それに「属性名 @ タグ名」という形式の名前を付けます。
- XML_SECTION_GROUP を使用する場合、CTX_DDL.ADD_ATTR_SECTION を使用して、属性セクションに任意の名前を付けることができます。

属性セクション・タイトル内で Tale を検索するには、次の問合せを発行します。

```
'Tale WITHIN BOOK@TITLE'
```

属性セクションは、ネストした WITHIN 問合せでは使用できません。

句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

```
Now is the time for all good <word type="noun"> men </word> to come to the aid.
```

WITHIN 演算子を使用するには、検索するセクションの名前を指定する必要があります。定義されたセクションのリストは、CTX_SECTIONS または CTX_USER_SECTIONS ビューを使用して取得できます。

参照： 次のマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

XML_SECTION_GROUP 属性セクション

Oracle8*i* リリース 8.1.5 以上では、XML_SECTION_GROUP によって、属性値の範囲内で索引付けおよび検索を行う機能が提供されます。次の行を含むドキュメントについて考えてみます。

```
<comment author="jeeves">
    I really like Oracle Text
  </comment>
```

ここで、XML_SECTION_GROUP が、1 つの属性セクションを提供します。これによって、索引に属性値を挿入できるようになります。次に例を示します。

```
  ctx_ddl.add_attr_section('mysg','author','comment@author');
```

構文は、他の add_section コールと同様です。最初の引数はセクション・グループの名前、2 番目はセクションの名前、3 番目は「タグ名 @ 属性名」という形式のタグです。この場合、Oracle Text は、comment タグの author 属性の内容を「author」セクションとして索引付けすることを命令されます。

次に、問合せ構文を示します。これは、他のセクションの場合も同様です。

```
WHERE CONTAINS ( ... ,'jeeves WITHIN author...',...)...
```

これによって、ドキュメントが検索されます。

属性値の識別によるセクション検索

属性セクションを使用すると、属性の内容を検索できます。ただし、属性値を使用して検索するセクションを指定することはできません。たとえば、次のドキュメントを想定します。

```
<comment author="jeeves">
   I really like Oracle Text
</comment>
```

このドキュメントは、次の問合せによって検索できます。

```
jeeves within comment@author
```

これは、author 属性の値に「jeeves」という単語を含む comment 要素を持つすべてのドキュメントを検索することに相当します。

ただし、現在は、次のような要求を実行することはできません。

```
interMedia within comment where (@author = "jeeves")
```

この方法では、jeeves が author である comment 要素で *inter*Media が使用されているすべてのドキュメントが検索されます。この機能（属性値の識別によるセクション検索）は、Oracle9*i* の将来のリリースでサポートされる予定です。

動的追加セクション

セクション・グループは索引の作成前に定義されるため、Oracle8*i* リリース 8.1.5 では、構造化ドキュメント・セットの変更に対する機能が制限されます。ドキュメントで新しいタグを初めて使用する場合、または新しい DOCTYPE を初めて取得する場合は、索引を再作成して、それらのタグの使用を開始する必要があります。

Oracle8*i* リリース 8.1.6 以上では、索引を再作成しないで、新しいセクションを既存の索引に追加できます。この場合、次に示すとおり、ALTER INDEX 文に ADD SECTION パラメータ文字列構文を使用します。

```
add zone  section <section_name> tag <tag>
add field section <section_name> tag <tag> [ visible | invisible ]
```

たとえば、タグ title を使用して、tsec という名前の新しいゾーン・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add zone section tsec tag title')
```

タグ author を使用して、asec という名前の新しいフィールド・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author')
```

このフィールド・セクションは、add_field_section を使用した場合と同様、デフォルトで表示されません。これを参照可能なフィールド・セクションとして追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author visible')
```

動的追加セクションは、索引メタデータを変更するのみで、索引を再作成しません。これは、動的追加セクションが、操作後に索引付けされたすべてのドキュメントに影響し、既存のドキュメントには影響しないことを意味します。動的追加セクションを含むドキュメントにすでに索引が付いている場合、これらのドキュメントは、(通常、索引列をそれ自身に更新して) 再度索引付けするために、手動でマーク付けする必要があります。

この操作では、特殊セクションの追加はサポートされません。特殊セクションを追加するには、すべてのドキュメントを再度索引付けする必要があります。この操作は、再作成を使用してオンラインで行うことはできませんが、比較的高速な操作です。

属性セクションの問合せに対する制約

次の制約が、属性セクション内での問合せに適用されます。

- 通常の属性テキストの問合せは、WITHIN 句で修飾されていない場合、該当するドキュメントにヒットしません。次の XML 文書があると想定します。

  ```
  <book title="Tale of Two Cities">It was the best of times.</book>
  ```

  Tale の問合せは、WITHIN title@book で修飾されていない場合、それ自体では該当するドキュメントにヒットしません。この動作は、参照可能なフラグ・セットを FALSE に設定した場合のフィールド・セクションの動作と同様です。

- 属性セクションは、ネストした WITHIN 問合せでは使用できません。
- 句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

  ```
  Now is the time for all good <word type="noun"> men </word> to come to the aid.
  ```

  通常の good men の問合せを行うと、このドキュメントにヒットし、間に挿入されている属性テキストは無視されます。

WITHIN 問合せは、属性セクションの繰返しを区別できます。この動作は、ゾーン・セクションの動作と同様ですが、フィールド・セクションの動作とは異なります。たとえば、次のドキュメントの場合を考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

この場合、book がゾーン・セクション、book@author が属性セクションであると想定します。次の問合せについて考えてみます。

```
'(Tale and Bondage) WITHIN book@author'
```

Tale および Bondage は、属性セクション book@author の異なる箇所に出現しているため、この問合せは、ドキュメントにヒットしません。

Oracle Text を使用した問合せアプリケーションの作成手順

Oracle Text を使用して問合せアプリケーションを作成するには、まず、索引付け手順を実行します。

次に、問合せアプリケーションを作成します。これを行うには、次の手順を実行します。

1. CTX_DDL.create_preference プロシージャを使用して、プリファレンスを作成します。8-31 ページの「手順 1: プリファレンスの作成」を参照してください。
2. CTX_DDL.Add_Attr_Section などを使用して、プリファレンスの属性を設定します。8-31 ページの「手順 2: プリファレンスの属性の設定」を参照してください。
3. 問合せ構文を作成します。

セクション・グループ・タイプとして XML_SECTION_GROUP または AUTOMATIC_SECTION_GROUP を使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

注意：

- ドキュメント内のすべてのものを検索できるわけではありません。まず、.....add_....._section を使用して何を検索できるかを確認してください。
- 索引に追加したセクションが多いほど、検索に時間がかかります。

Oracle Text では、ネストしたタグの検索がサポートされます。

CTX_OBJECTS 表および CTX_OBJECT_ATTRIBUTES ビューの使用

CTX_OBJECT_ATTRIBUTES ビューは、各オブジェクトのプリファレンスに割り当てることができる属性を表示します。すべてのユーザーは、このビューを問い合せることができます。

次の DESCRIBE コマンドを使用して、CTX_OBJECTS および CTX_OBJECT_ATTRIBUTE ビューの構造を確認してください。この章では、XML 文書の問合せのみを目的としているため、XML_SECTION_GROUP および AUTO_SECTION_GROUP に注目します。

```
Describe ctx_objects
  SELECT obj_class, obj_name FROM ctx_objects
  ORDRR BY obj_class, obj_name;
```

結果は次のとおりです。

```
...
SECTION_GROUP                  AUTO_SECTION_GROUP    <<==
SECTION_GROUP                  BASIC_SECTION_GROUP
SECTION_GROUP                  HTML_SECTION_GROUP
SECTION_GROUP                  NEWS_SECTION_GROUP
SECTION_GROUP                  NULL_SECTION_GROUP
SECTION_GROUP                  XML_SECTION_GROUP     <<==

...

Describe ctx_object_attributes
SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'XML_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-------------
ATTR
FIELD
SPECIAL
ZONE

SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'AUTO_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-------------
STOP
```

手順 1: プリファレンスの作成

まず、プリファレンスを作成する必要があります。これを行うには、CTX_DDL.Create_Preference プロシージャを使用します。次に例を示します。

CTX_DDL.Create_Preference

```
CTX_DDL.Create_Preference (
 preference_name  => 'books' /* or whatever you want to call it */
 object_name      => 'XML_SECTION GROUP' /* either XML_SECTION_GROUP or AUTO_
SECTION_GROUP */);
```

このプリファレンスを削除するには、次の構文を使用します。

```
CTX_DDL.Drop_Preference (
 preference_name => 'books');
```

手順 2: プリファレンスの属性の設定

XML_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section
- Add_Special_Section

AUTO_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section

対応する CTX_DDL.drop セクションおよび CTX_DDL.remove セクション構文があります。

CTX_DDL.add_zone_section の使用

次に、CTX_DDL.add_zone_section の構文を示します。

```
CTX_DDL.Add_Zone_Section (
 group_name      => 'my_section_group' /* whatever you called it above */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

ここで 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

add_zone_section についてのガイドライン

次に、add_zone_section についてのガイドラインを示します。

- 検索する必要がある XML 文書内の各タグごとに CTX_DDL.Add_Zone_Section をコールします。

CTX_DDL.Add_Attr_Section の使用

次に、CTX_DDL.add_attr_section の構文を示します。

```
CTX_DDL.Add_Attr_Section ( /* call this as many times as you need to describe
                              the attribute sections */
 group_name      => 'my_section_group' /* whatever you did call it above */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

ここで 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

add_attr_section についてのガイドライン

次に、add_attr_section についてのガイドラインを示します。

- 次の meta_data 属性 author について考えてみます。

  <meta_data author = "John Smith" title="How to get to Mars">

索引に追加したセクションが多いほど、検索に時間がかかります。

add_attr_section は、XML セクション・グループに属性セクションを追加します。このプロシージャは、XML 文書内の属性をセクションとして定義する場合に有効です。これによって、WITHIN 演算子を使用して、XML 属性テキストを検索できるようになります。

次に、section_name についてのガイドラインを示します。

- 属性テキストの WITHIN 問合せに使用される名前です。
- コロン（:）またはドット（.）文字を含むことができません。
- group_name 内で一意である必要があります。
- 大文字 / 小文字を区別しません。
- 64 バイト以下である必要があります。

このタグは、属性の名前を「タグ名 @ 属性名」形式で指定します。この URL の大文字 / 小文字は区別されます。

> **注意：** add_attr_section プロシージャでは、問合せ時に、様々なタグを同じセクション名で表すことができます。したがって、キーワード WITHIN 検索の引数として使用される名前が、実際の XML タグ名と異なる場合があります。これは、問合せ時に、様々なタグを同じ名前にマップできることを意味します。この機能によって、問合せの検索性が向上します。

CTX_DDL.add_field_section の使用

次に、CTX_DDL.add_field_section の構文を示します。

```
CTX_DDL.Add_Field_Section (
 group_name     => 'my_section_group' /* whatever you called it above */
 section_name   => 'qq' /* what you want to call this section */
 tag            => 'my_tag' /* what represents it in XML */ );
 visible        => TRUE or FALSE );
```

add_field_section についてのガイドライン

次に、add_field_section についてのガイドラインを示します。

- add_field_section および add_zone_section 属性は、パフォーマンスの点で異なります。
- ドキュメント内では、タグを 2 回以上繰り返すことができますが、title などのいくつかのタグは、1 回しか現れません。DTD（または XML Schema）で、タグが現れる回数を定義します。

- **VISIBLE 属性 :** これは、add_field_section では使用可能ですが、add_zone_section では使用できません。VISIBLE が TRUE に設定されているときに、「... CONTAINS virginia...」などの問合せを行うと、virginia がタイトルまたは段落内に現れる場合、正しい検索が行われません。

  再度、「... CONTAINS virginia...」という問合せについて考えてみます。VISIBLE=TRUE に設定すると、ヒットしない場合があります。VISIBLE=FALSE の場合は、索引が小規模になるなど、機能の一部が失われますが、VISIBLE =TRUE の場合と比較して、パフォーマンスは向上します。

  - add_zone_section を使用して索引を設定する場合
  - add_field_section を使用して索引を設定する場合

  注意 : 1 つのタグが 2 回以上現れる可能性がある場合、索引を構成することがさらに困難になります。

  - XML 文書内のタグが 1 回しか現れない場合は、単一の add_field_section プロシージャを使用してください。たとえば、「.... CONTAINS virginia and state WITHIN title.....」などの場合です。
  - XML 文書内のタグが 2 回以上現れる場合は、add_zone_section プロシージャを使用してください。たとえば、「.... CONTAINS virginia and state WITHIN paragraph....」などの場合です。多くの場合、タグは 2 回以上現れます。

属性セクションとフィールド・セクションの違い

属性セクションとフィールド・セクションは、次の点で異なります。

- **属性テキストは、表示されないとみなされます。**したがって、次の問合せは、フィールド・セクションと同様、ドキュメントを検索しません。

```
WHERE CONTAINS (..., '... jeeves',...)...
```

ただし、フィールド・セクションとは異なり、検索内の属性セクションは出現箇所を区別できます。次のドキュメントについて考えてみます。

```
<comment author="jeeves">
  I really like Oracle Text
</comment>
<comment author="bertram">
  Me too
</comment>
```

次の問合せを行うと想定します。

```
WHERE CONTAINS (...,'(cryil and bertram) WITHIN author', ...)...
```

「jeeves」および「bertram」が同じ属性テキスト内に現れないため、この問合せは、ドキュメントを検索しません。

- 複数の「タグ名 @ 属性名」を単一のセクション名にマップできますが、属性セクション名を、ゾーンまたはフィールド・セクション名とオーバーラップさせることはできません。属性セクションは、デフォルト値をサポートしません。次のドキュメントについて考えてみます。

```
<!DOCTYPE foo [
  <!ELEMENT foo (bar)>
  <!ELEMENT bar (#PCDATA)>
<!ATTLIST bar
  rev CDATA "8i">
]>
<foo>
  <bar>whatever</bar>
</foo>
```

また、次の属性セクションについて考えてみます。

```
ctx_ddl.add_attr_section('mysg','barrev','bar@rev');
```

問合せを実行します。

XML セマンティクスでは、bar 要素には rev 属性に対するデフォルト値がありますが、「8i within barrev」という問合せは、このドキュメントにヒットしません。

CtX_DDL.Add_Stop_Section の使用

```
CtX_DDL.Add_Stop_Section (
 group_name     => 'my_section_group' /* whatever you called it above */
 section_name   => 'qq' /* what you want to call this section */ );
```

手順 3: 問合せ構文の作成

問合せ文で CONTAINS 演算子を使用する方法については、8-6 ページの「CONTAINS 演算子を使用した問合せ」を参照してください。

属性セクション内での問合せ

セクション・グループ・タイプとして XML_SECTION_GROUP または AUTO_SECTION_GROUP を使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

次の XML 文書があると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

title@book セクションを属性セクションのタイトルとして定義できます。これは、CTX_DLL.Add_Attr_Section プロシージャを使用して行うか、または ALTER INDEX を使用して索引付けした後に動的に行います。

注意： AUTO_SECTION_GROUP を使用して XML 文書を索引付けする場合、システムは、自動的に属性セクションを作成し、それに「属性名 @ タグ名」という形式の名前を付けます。

XML_SECTION_GROUP を使用する場合、CTX_DDL.Add_Attr_Section を使用して、属性セクションに任意の名前を付けることができます。

属性セクション・タイトル内で Tale を検索するには、次の問合せを発行します。

```
WHERE CONTAINS (...,'Tale WITHIN title', ...)
```

XML_SECTION_GROUP および add_attr_section を使用した問合せの支援

次のとおり、TITLE 属性を指定して BOOK タグを定義する XML ファイルについて考えてみます。

```
<BOOK TITLE="Tale of Two Cities">
It was the best of times. </BOOK>
<Author="Charles Dickens">
Born in England in the town, Stratford_Upon_Avon </Author>
```

次に、CTX_DDL.Add_Attr_Section 構文を示します。

```
CTX_DDL.Add_Attr_Section ( group_name, section_name, tag );
```

TITLE 属性を属性セクションとして定義するには、次のとおり、XML_SECTION_GROUP を作成し、属性セクションを定義します。

```
    ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
    ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
    ctx_ddl.add_attr_section('myxmlgroup', 'authors', 'author');
    end;
```

TITLE 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

```
... WHERE CONTAINS (...,'Cities WITHIN booktitle', ....)...
```

AUTHOR 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

```
... WHERE 'England WITHIN authors'
```

Oracle Text の例 4: ドキュメントの問合せ

この例では、次の操作を行います。

1. res_xml 表の作成および移入
2. 索引 section_group およびプリファレンスの作成
3. プリファレンスのパラメータ化
4. res_xml に対するテスト問合せの実行

```
drop table res_xml;

CREATE TABLE res_xml (
  pk            NUMBER PRIMARY KEY ,
  text          CLOB
  ) ;

insert into res_xml values(111,
  'ENTITY chap8 "Chapter 8, <q>Keeping it Tidy: the XML Rule Book </q>"> this is the
document section');
commit;

---
--- script to create index on res_xml
---
```

```
--- cleanup, in case we have run this before
DROP INDEX res_index ;
EXEC CTX_DDL.DROP_SECTION_GROUP ( 'res_sections' ) ;

--- create a section group
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP ( 'res_sections', 'XML_SECTION_GROUP' ) ;
  CTX_DDL.ADD_FIELD_SECTION ( 'res_sections', 'chap8', '<q>') ;
END ;
/

begin
  ctx_ddl.create_preference
    (
      preference_name => 'my_basic_lexer',
      object_name     => 'basic_lexer'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_text',
      attribute_value => 'true'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_themes',
      attribute_value => 'false');
end;
/

CREATE INDEX res_index
  ON res_xml(text)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'lexer my_basic_lexer SECTION GROUP res_sections' ) ;
```

前述の索引を次のようなテスト問合せでテストします。

```
SELECT pk FROM res_xml WHERE CONTAINS( text, 'keeping WITHIN chap8' )>0 ;
```

Oracle Text の例 5: 索引の作成およびテキスト問合せの実行

この例で使用する explain_ex 表の作成

```
drop table explain_ex;

create table explain_ex
  (
    id        number primary key,
    text      varchar(2000)
  );

insert into explain_ex ( id, text )
  values ( 1, 'thinks thinking thought go going goes gone went' || chr(10) ||
              'oracle orackle oricle dog cat bird'              || chr(10) ||
              'President Clinton' );
insert into explain_ex ( id, text )
  values ( 2, 'Last summer I went to New England'               || chr(10) ||
              'I hiked a lot.'                                  || chr(10) ||
              'I camped a bit.' );
commit;
```

テキスト問合せ式に ABOUT を使用するテキスト問合せ

```
Set Define Off
select text
  from explain_ex
  WHERE CONTAINS ( text,
  '( $( think & go ) , ?oracle ) & ( dog , ( cat & bird ) ) & about(mammal
                                                    during Bill Clinton)' ) > 0;

select text
  from explain_ex
  WHERE CONTAINS ( text, 'about ( camping and hiking in new england )'  ) > 0;
```

ドキュメント・タイプが区別される XML 文書でのセクションの作成

様々なドキュメント・タイプで宣言されている <book> タグを含む XML 文書セットについて考えてみます。ドキュメント・タイプごとに個別の book セクションを作成する必要があります。次のとおり、mydocname が XML 文書タイプ（ルート要素）として宣言されているとします。

```
<!DOCTYPE mydocname ... [...
```

mydocname 内で、要素 <book> が宣言されます。このタグ用に、次のとおり、タグのドキュメント・タイプで区別される mybooksec という名前のセクションを作成できます。

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');
end;
```

注意：

- Oracle は、セクション・グループの作成時に指定する group_type パラメータから、終了タグがどのように見えるかを認識しています。指定する開始タグは、セクション・グループ内で一意にしてください。
- セクション名は、タグ間で一意である必要はありません。検索に対し詳細を透過的にして、同じセクション名を複数のタグに割り当てることができます。

セクションの繰返し

ゾーン・セクションは、繰り返すことができます。各出現箇所は、個別のセクションとして処理されます。たとえば、<H1> がヘッダー・セクションを示す場合、次のとおり、同じドキュメント内で繰り返すことができます。

```
<H1> The Brown Fox </H1>
<H1> The Gray Wolf </H1>
```

これらのゾーン・セクションに Heading という名前が付けられているとします。

次の問合せを行うとします。

```
WHERE CONTAINS (..., 'Brown WITHIN Heading', ...)...
```

この問合せは、このドキュメントを戻します。

次の問合せを行うとします。

```
WHERE CONTAINS (...,' (Brown and Gray) WITHIN Heading',...)...
```

この問合せは、このドキュメントを戻しません。

セクションのオーバーラップ

ゾーン・セクションは相互にオーバーラップできます。たとえば、<B> および <I> が 2 つの異なるゾーン・セクションを示す場合、次のとおり、これらはドキュメント内でオーバーラップできます。

```
plain <B> bold <I> bold and italic </B> only italic </I>  plain
```

セクションのネスト

ゾーン・セクションは、次のとおり、ネストすることができます。

```
<TD>
  <TABLE>
  <TD>nested cell</TD>
  </TABLE>
</TD>
```

WITHIN 演算子を使用すると、セクション内のセクションのテキストを検索する問合せを作成できます。

ネストしたセクション問合せの例

たとえば、BOOK1、BOOK2 および AUTHOR ゾーン・セクションが、次のとおり、ドキュメント doc1 および doc2 内に現れるとします。

doc1 は次のとおりです。

```
<book1><author>Scott Tiger</author> This is a cool book to read.</book1>
```

doc2 は次のとおりです。

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

次のネストした問合せについて考えてみます。

```
'Scott WITHIN author WITHIN book1'
```

この問合せは、doc1 のみを戻します。

問合せ結果の表示

テキスト問合せアプリケーションを使用すると、問合せによって戻されたドキュメントを表示できます。通常、ヒットリストからドキュメントを選択し、アプリケーションがそのドキュメントを特定の形式で表示します。

Oracle Text を使用すると、ドキュメントを様々な方法でレンダリングすることができます。たとえば、問合せ用語をハイライトさせることができます。ハイライト表示できる問合せ用語は、英語の場合、ワード問合せのワード、または ABOUT 問合せのテーマです。このレンダリングを行うには、CTX_DOC.HIGHLIGHT または CTX_DOC.MARKUP プロシージャを使用します。

また、CTX_DOC.THEMES という PL/SQL パッケージを使用しても、ドキュメントからテーマ情報を取得できます。この他にも、問合せ結果を表示するためのいくつかの CTX_DOC プロシージャがあります。

> **参照：** CTX_DOC パッケージの詳細は、『Oracle Text リファレンス』を参照してください。

例 : Oracle Text を使用したオンライン FAQ リストの検索

会社で、各製品に複数の FAQ を設定している場合を考えてみます。各 FAQ は、次のような XML 文書です。

```
<?xml version="1.0"?>
    <FAQ OWNER="Billy Text">
      <TITLE>Oracle Text FAQ</TITLE>
        <DESCRIPTION>Everything you always wanted to know ...</DESCRIPTION>
        <QUESTION>What is Oracle Text?</QUESTION>
        <ANSWER>Oracle Text uses standard SQL to index, ...</ANSWER>
    </FAQ>
```

このサンプル FAQ の例では、Text、Performance および XML というタイトルを持つ 3 つの FAQ のみを使用します。サンプル FAQ のプログラムは、WITHIN 演算子を使用して FAQ の XML タグ内の情報を検索します。実行時に、FAQ のユーザー・インタフェース（UI）用のプルダウン・メニューが生成されます。図 8-1 に、オンライン FAQ 検索の UI を示します。プルダウン・メニューには、FAQ データの検索用に選択した、次の XML 要素が表示されます。

- Title
- Question
- FAQ
- Description
- Answer

「FAQ Owner」は、実際は要素の属性です。属性も検索可能です。ユーザーは、これらのタグまたは要素を使用してキーワード検索を詳細に調整し、必要な FAQ を検索することができます。

図 8-1 オンライン FAQ 検索の UI: 検索オプション

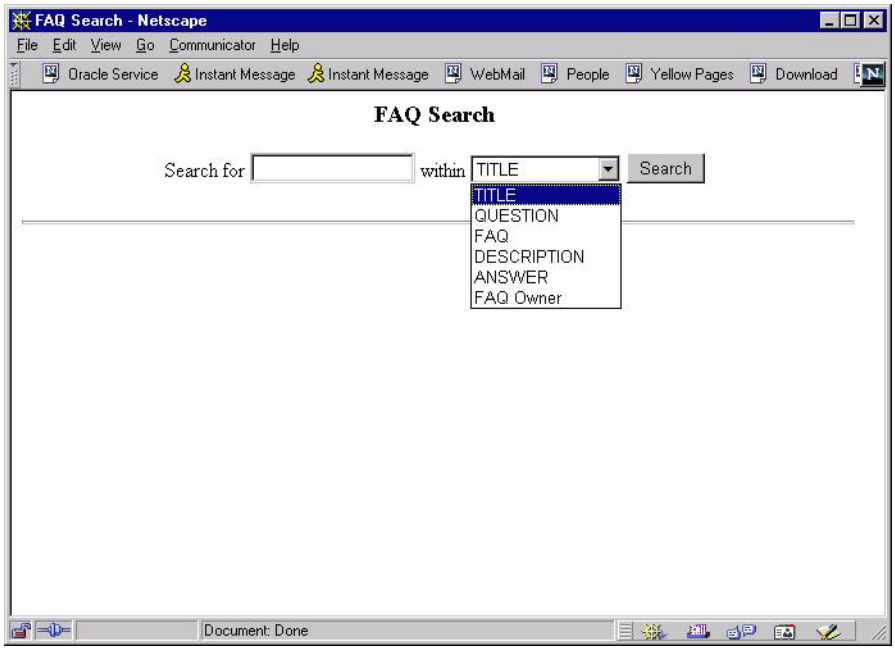


図 8-2 に、FAQ のすべての TITLE 要素でキーワード「Text」を検索するための入力方法を示します。その結果、タイトルに「Text...」が含まれる 1 つの FAQ が検索されます。

図 8-2 Oracle Text を使用したオンライン FAQ 検索の UI の作成 : TITLE 内の「XML」の検索

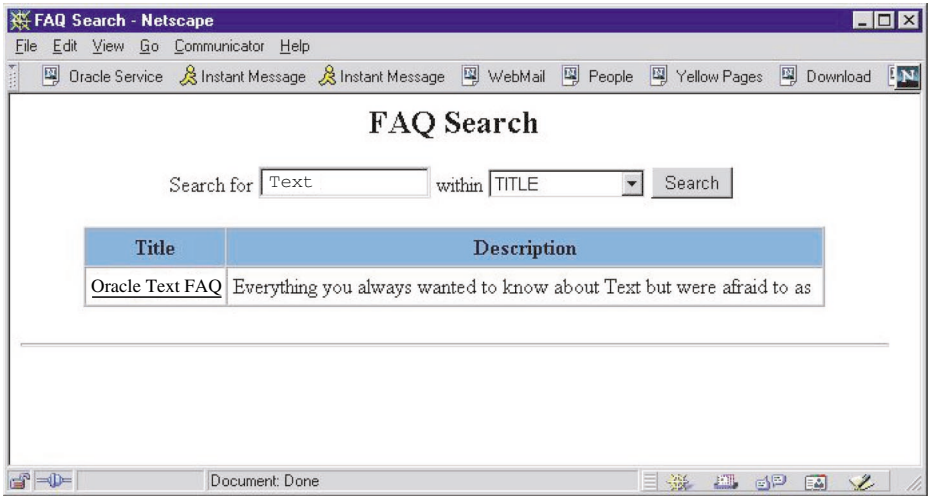
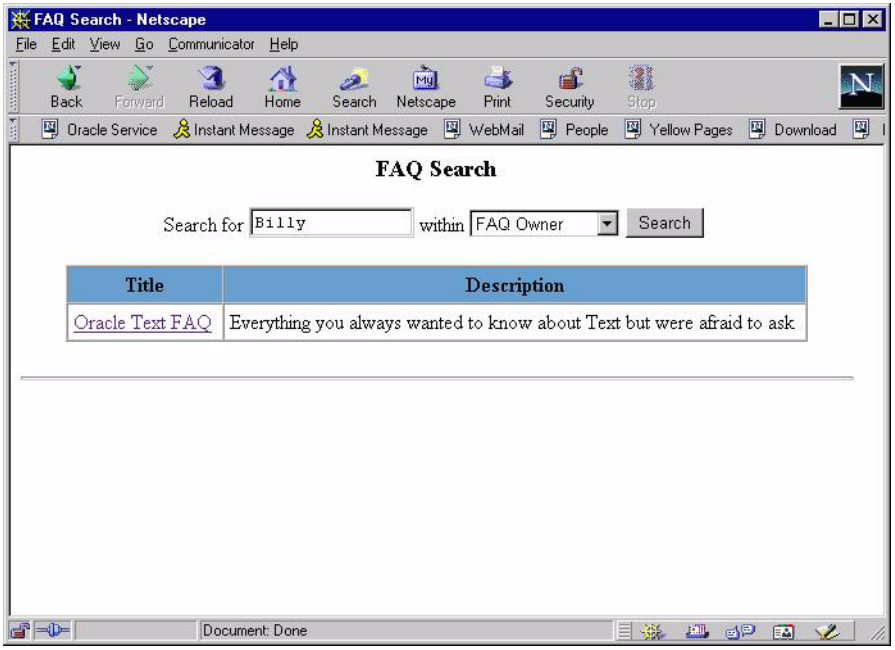


図 8-3 に、TITLE を使用して、FAQ Owner 内の「Billy」を検索する属性検索の入力方法を示します。このとき、TITLE は、FAQ Owner 要素の属性です。構文は、「...WHERE Billy WITHIN faq@owner」となります。

図 8-3 Oracle Text を使用したオンライン FAQ 検索 UI の作成 :「FAQ Owner」内の「Billy」を検索する属性検索



独自のオンライン FAQ 検索プログラムを作成するには、次の手順を実行します。

手順 1. FAQ 表の作成および移入 : AUTO_SECTION_GROUP および Oracle Text 索引の作成

手順 2. showxml.psp のコンパイル

手順 3. faqsearch.psp のコンパイル

手順 1. FAQ 表の作成および移入 : AUTO_SECTION_GROUP および Oracle Text 索引の作成

faqsearch_install.sql を実行します。このスクリプトは、次の操作を行います。

- FAQ 表を作成します。
- FAQ 表に 3 つのデータ・レコードを移入します。
- セクション・グループおよび Oracle Text 索引を作成します。

faqsearch_install.sql

```
insert into faq(tk,xml_desc)
 values(1,'<?xml version="1.0"?>
<FAQ OWNER="Billy Text">
<TITLE>Oracle Text FAQ</TITLE><DESCRIPTION>Everything you always wanted to know
about Text but were afraid to ask</DESCRIPTION>
<QUESTION>What is Oracle Text?</QUESTION>
    <ANSWER>Oracle Text uses standard SQL to index, search, and analyze text and
documents stored in the database, files or websites</ANSWER>
<QUESTION>What is ABOUT?</QUESTION>
    <ANSWER>ABOUT queries increase the number of relevant documents returned by a
query.</ANSWER>');

insert into faq(tk,xml_desc)
 values(2,'<?xml version="1.0"?><FAQ OWNER="Jack Performance">
<TITLE>Text Performance Guide</TITLE>
<DESCRIPTION>Oracle Text and interMedia Text performance guide</DESCRIPTION>
<QUESTION>What do we mean by query performance anyway?</QUESTION>
    <ANSWER>There are generally two measures of query performance - response time
(the time to get an answer to an individual query), and throughput (the number of
queries that can be run in any time period, eg queries per
second).</ANSWER></FAQ>');

insert into faq(tk,xml_desc)
 values(3,'<?xml version="1.0"?><FAQ OWNER="John XML">
<TITLE>XML FAQ</TITLE><DESCRIPTION>Oracle XML FAQ</DESCRIPTION>
<QUESTION>What is XML?</QUESTION>
    <ANSWER>XML stands for eXtensible Markup Language. XML s quickly becoming the
standard way to identify and describe data on the web because it has proved broadly
implementable and easy to deploy</ANSWER>
<QUESTION>What is the Oracle Kit?</QUESTION>
    <ANSWER>The Oracle XML Developer Kit (XDK) contains the basic building blocks
for reading, manipulating, transforming and viewing XML documents. To provide a
broad variety of deployment options, the Oracle XDK is available for Java, C, C++
and PL/SQL.</ANSWER></FAQ>');
```

```
commit;

begin
 ctx_ddl.create_section_group('faq_auto_section_group','auto_section_group');
end;
/

create index faq_idx on faq(xml_desc) indextype is ctxsys.context
 parameters('section group faq_auto_section_group')
/
```

手順 2. showxml.psp のコンパイル

PL/SQL Server Pages（PSP）を開発して配置するには、PL/SQL Web ゲートウェイの他に、リリース 8.1.6 以上の Oracle サーバーが必要です。現在、Web ゲートウェイには、Oracle Application Server、WebDB PL/SQL Gateway および OAS PL/SQL Cartridge があります。PSP の開発を始める前に、これらのゲートウェイの 1 つを使用して、データベース・サーバーと Web サーバーの両方にアクセスできる環境を準備する必要があります。

showxml.psp をコンパイルするには、次のコマンドを使用します。

```
loadpsp -replace -user username/passwd showxml.psp
```

> **参照：** PSP プログラムの使用の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

showxml.psp

```
<%@ plsql procedure="showxml" %>
<%@ plsql parameter="id" default="null" %>
<%! v_text      varchar2(32767); %>
<%! v_text_xml  varchar2(32767); %>

<%
  select xml_desc into v_text from faq where tk=id;
%>

<html>
<title>Show xml </title>
<body>
<h3>XML Content</h3>
<hr>

  <pre>
  <% v_text_xml := replace(v_text,'<','&lt;'); %>
```

```
  <%= v_text_xml %>
  </pre>

</body>
</html>
```

手順 3. faqsearch.psp のコンパイル

ブラウザで次の URL をオープンし、faqsearch.psp にアクセスします。

```
http://myserver_and_directory/faqsearch
```

faqsearch.psp

```
<%@ plsql parameter="query" default="null" %>
<%@ plsql parameter="tagvalue" default="null"  %>

<%! v_results numeric := 0; %>

<html>
<head>
   <title>FAQ Search </title>
</head>
<body>

<%
  If query is null Then
%>

  <center>
  <h3>FAQ Search </h3>
  <form method=post action="faqsearch">
  <p>
  Search for <input type=Text size=15 maxlength=25 name="query">
  within
  <select name="tagvalue">

  <%
-- generates the pull-down menu with the following select
      for c in (select token_text from DR$FAQ_IDX$I
                where token_type=2)
        loop
  %>
```

```
          <option value="<%= c.token_text %>"><%= c.token_text %>

  <%  end loop; %>

     <option value="faq@Owner">FAQ Owner
  </select>

  <input type=submit value="Search">
  </form>


<% Else %>
   <%!
     color varchar2(6) := 'ffffff';
   %>
   <center>
   <h3>FAQ Search</h3>
   <form method=post action="faqsearch">
   <p>
   Search for
   <input type=text size=15 maxlength=25 name="query" value=<%= query %>>
   within
   <select name="tagvalue">

   <%
-- generates the pull-down menu with the following select
      for c in (select token_text from DR$FAQ_IDX$I
                where token_type=2)
      loop
   %>
          <option value="<%= c.token_text %>"><%= c.token_text %>

   <% end loop; %>

      <option value="faq@Owner">FAQ Owner
   </select>

   <input type=submit value="Search">
   </form>

   <p>

   <%! v_query        varchar2(400);  %>
   <%! v_desc_substr  varchar2(1000); %>
   <%! v_desc_item    varchar2(1000); %>
   <%! v_desc_start   number;         %>
   <%! v_desc_final   number;         %>
```

```
   <%! v_title_substr varchar2(1000); %>
   <%! v_title_start  number;         %>
   <%! v_title_final  number;         %>


   <%

     v_query := query || ' WITHIN '|| tagvalue;

-- Text query using WITHIN for XML documents
     for doc in (select tk, xml_desc
                    from faq where contains(xml_desc,v_query) >0
                )
        loop
          v_results := v_results + 1;
          if v_results = 1 then
   %>

             <table border="1" cellpadding="4" cellspacing="0">
               <tr bgcolor="#6699CC">
                 <th>Title</th>
                 <th>Description</th>
               </tr>
   <%     end if; %>

          <tr bgcolor="#<%= color %>">

   <%
           v_title_start  := instr(doc.xml_desc,'<TITLE>');
           v_title_final  := instr(doc.xml_desc,'</TITLE>');
           v_title_substr := substr(doc.xml_desc,v_title_start+length('<TITLE>'),v_
title_final - length('</TITLE>') - v_title_start+1);

           v_desc_start   := instr(doc.xml_desc,'<DESCRIPTION>');
           v_desc_final   := instr(doc.xml_desc,'</DESCRIPTION>');
           v_desc_substr  := substr(doc.xml_desc,v_desc_
start+length('<DESCRIPTION>'),v_desc_final - length('</DESCRIPTION>') - v_desc_
start+1);
   %>

           <td>
             <a href="showxml?id=<%= doc.tk %>"><%= v_title_substr %></a>
           </td>

   <%      v_desc_item := replace(v_desc_substr,'<','&lt;'); %>

           <td>
```

```
            <%= v_desc_item %>
          </td>
         </tr>

   <%
         if (color = 'ffffff') then /* alternate row color */
            color := 'eeeeee';
          else
            color := 'ffffff';
         end if;
       end loop;
   %>

       </table>
     </center>

  <%
     if v_results = 0 then %>
       <center><h3>Found 0 records for your query</h3></center>
  <% end if; %>

<% End if;%>
<p>
<hr>
<p>
</body>
</html>
```

FAQ: Oracle Text

この項の内容は次のとおりです。

属性値の検索

属性値の索引の作成

質問

現在、Oracle Text（*inter*Media Text）には、セクション・グループの内容に基づいて索引を作成するためのオプションがあります。ただし、ほとんどの XML 要素は、型の要素です。このため、検索のためのオプションは、属性値のみとなります。属性値の索引を作成できますか？

回答

リリース 8.1.6 以上では、属性を索引付けできます。次の Web サイトを参照してください。
http://otn.oracle.com/products/intermedia/htdocs/text_training_816/Samples/imt_816_techover.html#SCN

Oracle Text に関する一般的な質問

表データと同様の方法を使用した XML 文書の問合せ

質問

完全な状態の XML 文書は、Oracle の XML ソリューションによって、CLOB に格納されると理解しています。

1. CLOB/BLOB に格納された XML 文書は、表スキーマのように問い合せることができますか？次に例を示します。

   ```
   [XML document stored in BLOB]...<name id="1111"><first>lee</first>
   <sencond>jumee</second></name>...
   ```

   この場合、XML 文書の要素、属性および構造によって、値（lee、jumee）を問い合せることができますか？

2. 挿入 / 更新 / 削除できない要素または属性がある場合、文書全体を更新する必要がありますか？また文書は、表スキーマと同様に、挿入 / 更新 / 削除できますか？
3. ロックに関する質問ですが、BLOB/CLOB に格納された XML 文書を管理している場合、他の人は同じ XML 文書にアクセスできないと聞きました。

回答

1. Oracle Text（*inter*Media Text）を使用すると、次のような問合せを使用してこの文書を検索できます。

   ```
   lee within first or this:jumee within second or this:1111 within name@id
   ```

   これらを次のとおり組み合せることもできます。

   ```
   lee within first and jumee within secondor this:(lee within first) within name
   ```

   詳細は、OTN の Web サイトからダウンロードできる『*inter*Media Text Technical Overview』を参照してください。
2. Oracle Text（*inter*Media Text）は、CLOB/BLOB を索引付けしますが、XML を明確に理解しているわけではないので、実際には個々の要素を変更することはできません。文書全体を編集する必要があります。
3. 他の CLOB と同様、誰かが CLOB に書き込んでいる場合、その CLOB はロックされ、他の人はそれに書き込むことができません。他のユーザーは、その CLOB を読み取ることはできますが、書き込むことはできません。これは、LOB の基本動作です。

   別の方法として、XML 文書を分解し、情報をリレーショナル・フィールドに格納する方法があります。これによって、個々の要素の変更、要素レベルでの同時アクセスなどが可能になります。この場合、USER_DATASTORE を使用すると、PL/SQL を使用して再度ドキュメントを XML に構成し、テキストを索引付けすることができます。これによって、テキストを XML として検索できますが、データはリレーショナル・データとして管理されます。詳細は、次のサイトの『*inter*Media Text Technical Overview』を参照してください。

   http://otn.oracle.com/products/text

構造条件に基づいた検索

質問

Oracle Text（*inter*Media Text）を使用して、2/7/1968 のような XML を索引付けできますか？また、次の問合せは処理できますか？

select name from person where hair.color = "BROWN"（髪の毛が茶色の人）

回答

構造条件に基づく検索は、現在 Oracle Text（*inter*Media Text）を介して使用できません。属性検索は、Oracle8*i* リリース 8.1.6 以上でサポートされています。将来、XML Schema が勧告された場合にこれに準拠しないため、データを属性に挿入しないでください。

XML 文書を検索してゾーンを取得する方法

質問

大規模な XML ファイルを Oracle8*i* に格納し、それを検索して、特定のタグ付けされた領域を取得する必要があります。Oracle Text（*inter*Media Text）を使用すると、次のことが実行できます。

- XML ファイルを CLOB フィールドに格納できます。
- ctxsys.context を使用して、XML ファイルを索引付けすることができます。
- <Zone> および <Field> を作成して、ctx_ddl.add_zone_section(xmlgroup,"dublincore", dc); のように、XML ファイルにタグを表すことができます。
- ゾーンまたはフィールドでテキストを検索できます。たとえば、「Select titile from mytable where CONTAINS(textField,"some words WITHIN doubleness")」という問合せが可能です。

テキスト検索によってゾーンまたはフィールドを取得する方法を教えてください。

回答

Oracle Text（*inter*Media Text）は、ヒットした項目のみを戻します。次に、CLOB を解析して、セクションを抽出する必要があります。

データベースへの XML 文書のロードおよび Oracle Text を使用した検索

質問

XML 文書をデータベースに挿入する方法を教えてください。XML 文書を、現在の状態のまま、表内の CLOB データ型の列に挿入する必要があります。

回答

Oracle XSU for Java は、XML データをロードするために使用可能なコマンドライン・ユーティリティを提供します。XSU の詳細は、http://otn.oracle.com/tech/xml および第 7 章「XSU」を参照してください。

XML 文書は、すべてのテキスト・ファイルと同様に挿入できます。CLOB への挿入も、他のテキスト・ファイルの場合と変わりません。

質問

Oracle Text（*inter*Media Text）を使用して、CLOB に格納された XML を索引付けおよび検索できますか？これを行うための方法を教えてください？

回答

Oracle8*i* リリース 8.1.6 より前のリリースの Oracle Text（*inter*Media Text）では、タグ・ベースの検索のみが可能でした。今回のリリースでは、XML の構造ベースおよび属性ベースの検索が可能になっています。索引の作成方法および Oracle Text での SQL の使用方法に関するドキュメントを参照してください。

参照：『Oracle Text リファレンス』を参照してください。

WITHIN 演算子を使用して XML を検索する方法

質問

次の XML があります。

```
<person>
  <name>efrat</name>
 <childrens>
     <child>
         <id>1</id>
         <name>keren</name>
     </child>
  </childrens>
</person>
```

keren という名前の子供を持ち、自分の名前が keren ではない人を検索する方法を教えてください。ネスト可能で、自分自身を含むことができる add_zone_section を使用して、すべてのタグを定義済であると想定します。

回答

タグのかわりに属性として ID を作成することをお薦めします。

Oracle Text（*inter*Media Text）および XML

質問

XML を Oracle Text で使用する方法を適切に示した例はどこで参照できますか？

回答

次のマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

Oracle Text（*inter*Media Text）および XML: Add_field_section

質問

Oracle Text（*inter*Media Text）は、XML 文書内のタグを認識できますか？または、XML 文書内のタグごとに add_field_section コマンドを使用する必要がありますか？使用する XML 文書には、何百ものタグが含まれています。これを簡単に行う方法はありますか？

回答

どのリリースのデータベースをご使用ですか？リリース 8.1.5 では、タグごとに add_field_section コマンドを使用する必要がありますが、Oracle8*i* リリース 8.1.6 ではその必要はありません。リリース 8.1.6 では、AUTO_SECTION_GROUP を使用できます。

XSQL Servlet には、SQL スクリプトの完全な例（*inter*Media の使用部分は単純）が付属しています。このスクリプトは、オブジェクト型から複雑な XML データグラムを作成し、保険請求書型に格納された XML 文書フラグメントに対して Oracle Text（*inter*Media Text）索引を作成します。

XSQL Servlet をダウンロードし、ファイル ./xsql/demo/insclaim.sql を参照すると、ファイルの下にある *inter*Media の要素を参照できます。Oracle8*i* リリース 8.1.6 における *inter*Media の新しい主要機能は、AUTO Sectioner for XML です。

Oracle Text を使用して範囲検索を行う方法

質問

CLOB に格納した XML 文書があります。また、section_group などを使用して、タグに対する索引を作成済です。1 つのタグは、<SALARY></SALARY> です。たとえば、給与が 5000 を超えるすべてのレコードを選択する SQL 文を記述する必要があります。これを行うには、どうすればよいですか？WITHIN 演算子は使用できません。このタグ内の値を数値として解釈する必要があります。また、これは給与であるため、浮動小数点の場合もあります。

回答

これは Oracle Text では行えません。実際には、範囲検索はテキスト操作ではありません。最適のソリューションは、別の Oracle の XML 解析ユーティリティを使用して、給与を NUMBER フィールドに変換することです。これによって、テキスト検索には Oracle Text（*inter*Media Text）を使用し、また、より構造化されたフィールドには通常の SQL 演算子を使用して、同じ結果を取得できます。

Oracle Text によるセクションの取出し

質問

XML 形式のすべてのドキュメントを CLOB に格納しています。Oracle には、ドキュメント全体を取得して、その構造を検索するのではなく、1 つのフィールドごとに内容を一度に取得（フィールド名を指定して、タグ間のテキストを取得する）するために使用できるユーティリティはありますか？Oracle Text はそれに該当しますか？

回答

Oracle Text は、セクションの取出しを行いません。これについては、第 7 章「XSU」を参照してください。

3 つの列に対する 1 つの Text 索引の作成

質問

7 ～ 8 つの表に基づいてビューを作成し、このビューには、custordnumber、product_dscr、qty、prdid、shipdate、ship_status などの列があります。次の 3 つの列に対して 1 つの Oracle Text 索引を作成する必要があります。

- custordnumber
- product_dsc
- ship_status

これらの列に対して 1 つの Text 索引を作成する方法はありますか？

回答

あります。次の 2 つのオプションがあります。

1. 索引付け実行中に、USER_DATASTORE オブジェクトを使用してその場で連結フィールドを作成します。
2. フィールドを連結し、1 つの表の追加の CLOB フィールドに格納します。次に、その CLOB フィールドに対する索引を作成します。Oracle8*i* リリース 8.1.6 以上を使用している場合、連結前に XML タグを各フィールドの前後に置くオプションもあります。これによって、各フィールド内での検索が可能になります。

Oracle によるテキストの索引付けの速度、およびブール検索の有効化

質問

mySQL を使用して、1 日に 900 万の Web ページの部分索引付けを行っています。4 つの CPU を搭載した SPARC 420 で実行していますが、全文索引を作成することができません。Oracle8*i* または Oracle9*i* ではこれを行うことができますか？

トランザクションの整合性、テキスト・ページへの特殊なフィルタの適用、単純なブール・ワード検索以外の検索（スコアリング、ステミング、ファジー検索、近接検索など）の実施には関心がありません。

次の質問があります。

- Oracle では、mySQL より高速で索引作成を行うことができますか？
- その場合、ブール・ワード検索以外のすべての索引作成機能を無効化する方法はありますか？

回答

あります。Oracle Text は、900 万の Web ページの全文索引を、高速で作成できます。Sun 社製の大型サーバーを使用したベンチマーク・テストでは、100GB の Web ページ（約 1,500 万）を 7 時間で索引付けできました。通常の DML またはパーティション化を使用して、部分索引付けを行うこともできます。

テーマ索引付けを無効化する「簡易索引付け」を行うこともできます。ドキュメントがすでに ASCII/HTML/XML である場合、そのドキュメントにフィルタを適用する必要はありません。最も一般的な拡張であるファジー、ステミング、近接は、問合せ時に行います。

異なる言語の XML 文書を索引付けする方法

質問

Oracle8*i* リリース 8.1.6 では複数の言語のレコードを同一の表（および列）に格納することができ、Oracle Text が（マルチレクサー機能を使用して）各行に設定された言語に基づいて索引を適切に処理することは知っています。

現在、表内の 1 列の CLOB 列を使用し、Oracle Text で索引付けしています。列の内容は XML 形式（タグ付けされた形式）で、フィールド、セクションおよびゾーン関数を使用して索引付けおよび検索を行います。データベース内のデータが単一言語（異なる言語およびサイトには別のデータベースを使用）である場合は、問題なく動作します。現在、Oralce8*i* リリース 8.1.5 を使用して、NLS_LANG が個々の言語に対して索引付けおよび検索を正しく行うように適切に設定しています。

現在、複数言語のデータを同一の表（および列）に格納する必要があります。個々のデータ要素が異なる言語である場合もあります。たとえば、タイトルがフランス語で著者がスペイン人である本に関するレコードなどです。現時点では、このようなすべての情報を、フィールド / セクションで分割して CLOB 列に格納しています。次の質問があります。

1. Oracle8*i* リリース 8.1.6 では索引内の個々のセクションに言語を指定することはできないと思うのですが、この考えは正しいでしょうか？
2. 異なる言語で使用される可能性があるすべてのフィールドを、同一の表の異なる列に分割し、これらの各列に対応する言語の列を作成し、マルチレクサー機能を使用して個別の索引を作成することができます。この前提は適切ですか？または、推奨されていますか？
3. 前述の作業を実行すると、列にまたがって検索する場合に複数の CONTAINS 句が必要です。これにより、パフォーマンスが低下する場合があります。
4. この問題に対して最適な対策を教えてください。

回答

1. 正しいです。Oracle8*i* リリース 8.1.6 では、索引内の個々のセクションに言語を指定することはできません。

2. ～ 3. では、潜在的な問題が正しく認識されています。

CLOB での XML 文書の検索

Oracle Text を使用して CLOB を検索する方法

質問

CLOB 列内の「aorta」および「damage」を含むレコードを検索するには、Oracle Text パラメータをどのように定義すればよいですか？次に、XML（および DTD）の例を示します。

```
WellKnownFileName.gif   echocardiogram   aorta
```

これは、血管損傷のイメージです。単純（または複雑）な XML の Oracle Text での実装の例を参照できれば参考になります。この場合、ZONE または FIELD を設定する必要はありませんか？

回答

XML 文書フラグメントを CLOB に保存し、それに対して Oracle Text XML 索引を使用可能にすると、次のとおり、CONTAINS() 演算子を使用して SQL 問合せを行うことができます。

次の保険金請求書のドキュメントがあるとします。

```
77804
1999-01-01 00:00:00.0                  8895                          1044
Paul               Astoria
123 Cherry Lane        SF         CA         94132
1999-01-05 00:00:00.0           7600            JCOX
It was becase of Faulty Brakes
```

この内容を文書フラグメントとして CLOB に格納すると、次の問合せを行うことができます（他のものはすべてリレーショナル表に格納するとします）。

```
REM Select the SUM of the amounts of
REM  all settlement payments approved by "JCOX"
REM  for claims whose  relates to Brakes.
select sum(n.amount) as TotalApprovedAmount
  from insurance_claim_view v, TABLE(v.settlements) n
  where n.approver = 'JCOX'
  and contains(damageReport,'Brakes within Cause') >
```

様々な DTD を使用して CLOB に格納された様々な XML 文書を検索する方法

質問

CLOB に XML を格納し、DOM または SAX を使用して後から必要に応じて XML を再解析するとします。このドキュメント・リポジトリを検索するには Oracle Text（*inter*Media Text）が最適ですか？ Oracle8*i* で *inter*Media を使用してこの問題を解決するための例（XML_SECTION_GROUP の定義方法、FIELD に対する ZONE の使用場所など）を参照できれば参考になります。次に例を示します。

「WellKnownFileName.gif　echo cardiogram aorta」という XML（および DTD）で、CLOB 列内の「aorta」および「damage」を含むレコードを検索するには、*inter*Media パラメータをどのように定義すればよいですか？この XML は、血管損傷のイメージです。

回答

Oracle8*i* リリース 8.1.6 以上では、属性テキスト内で検索を行うことができます。これは、「state within book@author」のような検索です。Oracle は、次のような属性値の識別による検索を実行できます。

```
state within book[@author = "Eric"]:
begin  ctx_ddl.create_section_group('mygrp','basic_section_group');
  ctx_ddl.add_field_section('mygrp','keyword','keyword');
  ctx_ddl.add_field_section('mygrp','caption','caption');
end;
create index myidx on mytab(mytxtcolumn)indextype is ctxsys.contextparameters
('section group mygrp');
select * from mytab where contains(mytxtcolumn, 'aorta within keyword')>0;
options:
```

- タグに属性が含まれる場合、または大文字 / 小文字を区別したタグの検出が必要である場合は、基本セクション・グループのかわりに XML セクション・グループを使用します。
- セクションがオーバーラップする場合、またはインスタンスを区別する必要がある場合は、フィールド・セクションのかわりにゾーン・セクションを使用します。たとえば、keywords がフィールド・セクションの場合、「(aorta and echo cardiogram) within keywords」という問合せは、ドキュメントにヒットします。keywords がゾーン・セクションの場合、この問合せは、同じ keywords のインスタンスにないため、ドキュメントにヒットしません。

この例が最適であるとはいえません。この例では、同じレコード内に「aorta」を含む兄弟要素を持つ「damage」を含む要素のインスタンスを検索しているように見えます。「レコード」が意味するものが明確ではありません。

それぞれのレコードがこの例のレコードと同じで、単一の XML を格納した LOB に複数のレコードが存在する場合、*inter*Media を使用して、この検索を実行できますか？

CLOB/ 行ごとに 1 つのレコードのみがある場合、2 つの ConText 要素問合せを AND で組み合せることによって、これを検索できます。ただし、この検索は、実際に必要な構造より、予想される制約がある XML 検索といえます。

Oracle Text を使用した CLOB への XML 文書の格納

質問

XML ファイル（現時点でファイル・システム上に存在）をデータベースに格納する必要があります。ドキュメント全体を格納する必要があります。タグによってドキュメントを分割し、情報を別々の表 / フィールドに格納することは望ましくありません。逆に、様々な XML 文書を格納するために使用できる汎用表が必要です。この場合、汎用表は内部で CLOB 型のフィールドに格納されると考えています。使用する XML ファイルは、常に ASCII データを含みます。

*inter*Media を使用して、このような処理を行うことができますか ? そのためには、Oracle Text（*inter*Media Text）または *inter*Media Annotator のどちらを使用する必要がありますか ? Annotator は入手済ですが、XML 文書をデータベースに格納することができません。

XML 文書を CLOB 列に格納しようとしています。基本的に、次のように定義されている表が 1 つあります。

```
CREATE TABLE xml_store_testing
(
    xml_doc_id  NUMBER,
    xml_doc     CLOB  )
```

XML 文書を xml_doc フィールドに格納する必要があります。

XML 文書の内容を読み取るために、次に示す別の PL/SQL プロシージャを作成しました。XML 文書は、ファイル・システム上で使用可能です。XML 文書には ASCII データのみが含まれ、バイナリ・データはありません。

```
CREATE OR REPLACE PROCEDURE FileExec
(
  p_Directory      IN VARCHAR2,
  p_FileName       IN VARCHAR2)
  AS   v_CLOBLocator  CLOB;
       v_FileLocator       BFILE;
BEGIN
    SELECT  xml_doc
    INTO      v_CLOBLocator
    FROM    xml_store_testing
    WHERE  xml_doc_id = 1
    FOR        UPDATE;
    v_FileLocator := BFILENAME(p_Directory, p_FileName);
    DBMS_LOB.FILEOPEN(v_FileLocator, DBMS_LOB.FILE_READONLY);
```

```
    dbms_output.put_line(to_char(DBMS_LOB.GETLENGTH(v_FileLocator)));
      DBMS_LOB.LOADFROMFILE(v_CLOBLocator, v_FileLocator,
      DBMS_LOB.GETLENGTH(v_FileLocator));
      DBMS_LOB.FILECLOSE(v_FileLocator);
END FileExec;
```

回答

XML 文書を CLOB 列に挿入してから、XML セクション・グループを使用して、Oracle Text（*inter*Media Text）索引をそれに追加してください。詳細は、http://otn.oracle.com/products/intermedia を参照してください。

質問

このプロシージャを実行すると、正常に実行されました。ただし、表から選択した場合、CLOB フィールド内の表に不明な文字が現れました。これは、オペレーティング・システム（XML ファイルを格納）とデータベース（CLOB データを格納）のキャラクタ・セットの違いが原因ですか？

回答

キャラクタ・セットの違いが原因です。キャラクタ・セットが異なる場合は、UTL_RAW.CONVERT を介してデータを渡し、キャラクタ・セットを変換してから CLOB に書き込む必要があります。

表の作成時における構造化データのみの挿入

質問

データを XML ファイルからデータベースに挿入する必要があります。作成済の表を使用して構造化データのみを挿入できると聞きました。これは本当ですか？

現在、法律プロジェクトに取り組んでおり、このプロジェクトでは、構造化データおよび非構造化データを含む法律データを格納し、Oracle Text（*inter*Media Text）を使用してそのデータを検索する必要があります。非構造化データも挿入できますか？これを行うためには、カスタム・アプリケーションを作成する必要がありますか？また、構造化部分および非構造化部分を持つデータを格納した場合、Oracle Text（*inter*Media Text）を使用してそのデータを検索できますか？非構造化部分を CLOB に格納し、その CLOB にタグが含まれている場合、特定のタグ内にあるデータのみを検索することはできますか？

回答

Oracle CM SDK の使用を検討してください。Oracle CM SDK を使用すると、ドキュメントを分割し、それを複数の表および LOB に格納できます。Oracle Text（*inter*Media Text）は、タグを使用してデータ検索を行うことができ、XML の階層構造も理解します。Oracle8*i* リリース 8.1.6 以上では、Oracle Text に、この機能および名前 / 値のペア属性の検索機能が追加されています。

質問

前述の回答によると、カスタム・アプリケーションを開発しない場合、当面このようなドキュメントの分割は不可能であるということですか？*inter*Media では、XML の階層構造が認識されませんが、次のような検索はできますか？

```
<report>
   <day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

Oracle Text を使用して索引付けし、cause が hurricane である LOB を検索できますか？

回答

Oracle Text（*inter*Media Text）の今回のリリースを使用すると、このレベルの検索は可能です。現在、ドキュメントを分割するには、Oracle XML Parser を XSLT とともに使用して、XML を DDL に変換するスタイルシートを作成する必要があります。Oracle CM SDK を使用すると、さらに高レベルのインタフェースが提供されます。

もう 1 つの方法は、JDBC プログラムを使用してドキュメントまたはドキュメント・フラグメントのテキストを CLOB 列または LONG 列に挿入し、索引を設定してから、CONTAINS() 演算子を使用して検索を行う方法です。

第 III 部

XML を使用したデータ交換

第 III 部では、Oracle AQ、新しい AQ の iDAP 機能、XML キュー、XML メッセージ変換について説明し、AQ および XML を B2B メッセージ機能アプリケーションで使用する方法を説明します。

第 III 部に含まれる章は、次のとおりです。

9

Oracle AQ を使用した XML データの交換

この章の内容は次のとおりです。

AQ の概要

Oracle AQ には、次のデータベース統合化メッセージ・キューイング機能があります。

- メッセージを使用する 2 つ以上のアプリケーションの非同期通信を可能にし、その管理を行います。
- Point-to-Point 通信モデルおよびパブリッシュ・サブスクライブ通信モデルをサポートします。

Oracle データベースとメッセージ・キューイングの統合によって、Oracle の整合性、信頼性、リカバリ能力、スケーラビリティ、パフォーマンスおよびセキュリティ機能がメッセージ・キューイングに提供されます。また、Oracle との統合によって、メッセージ・フローからのインテリジェント機能の抽出が容易になります。

AQ と XML の相互補完

XML は、企業間通信の標準フォーマットとして登場しました。XML は、ビジネス・アプリケーション間で通信されるデータを表示するだけでなく、XML でカプセル化されたビジネス・ロジックの表示にも使用されています。

Oracle では、AQ がネイティブ XML メッセージをサポートしているため、AQ 操作を XML ベースの iDAP 形式で定義できます。拡張可能なメッセージの起動プロトコルである iDAP は、インターネット標準に基づいて構築され、転送メカニズムとして HTTP プロトコルおよび電子メール・プロトコルを使用し、データ表示用の言語として XML を使用します。クライアントは、iDAP を使用して AQ にアクセスできます。

詳細は、9-6 ページの「iDAP」を参照してください。

AQ および XML メッセージ・ペイロード

図 9-1 に、AQ を使用して 3 つのアプリケーションと通信する Oracle データベースを示します（メッセージ・ペイロードとして XML を使用します）。この使用例において AQ が実行する一般的なタスクは、次のとおりです。

- サブスクリプション・ルールを使用したメッセージ・フロー
- メッセージ管理
- メッセージからのビジネス・インテリジェンスの抽出
- メッセージ変換

これは、XML メッセージが、AQ を使用してアプリケーション間で非同期に渡される、企業内業務および企業間業務での使用例です。

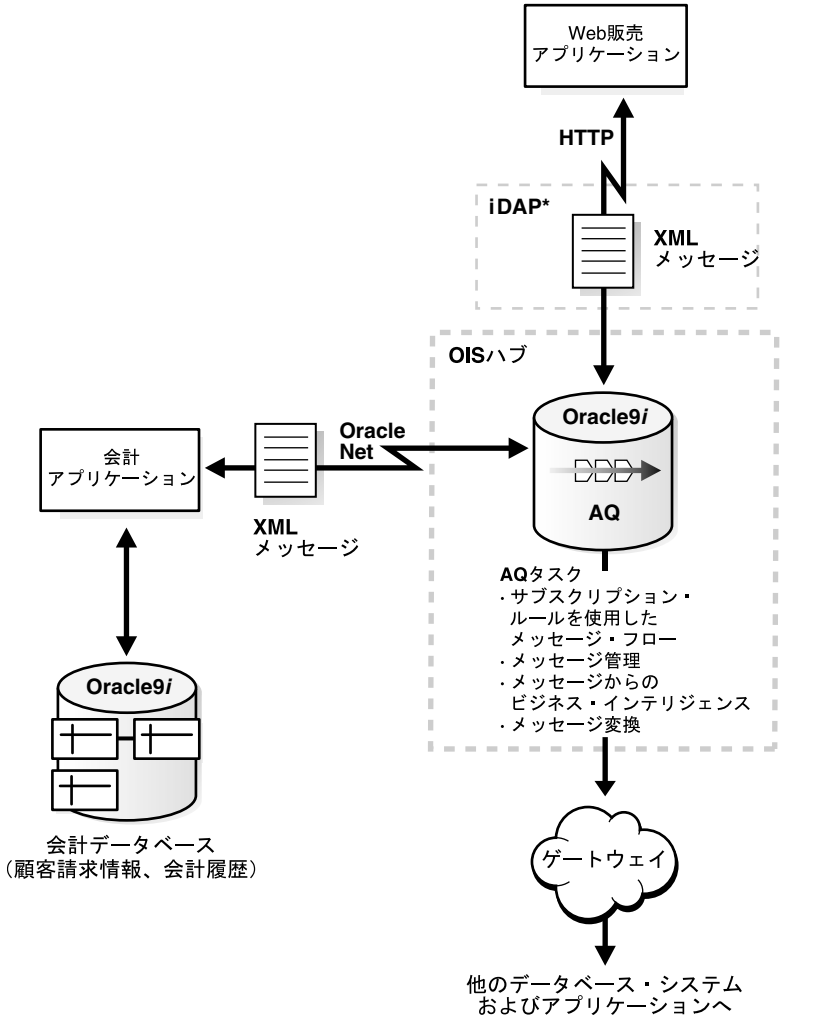
- 企業内業務の典型的な使用例には、販売注文の遂行およびサプライチェーンの管理などがあります。
- 企業間業務処理では、複数の統合ハブが、インターネットのバックプレーンを介して通信できます。企業間業務での使用例は、旅行の予約、メーカーとサプライヤ間の調整、銀行間での資金の移動、保険請求の清算などです。

  Oracle は、これを企業アプリケーション統合製品で使用します。XML メッセージは、アプリケーションから AQ ハブ（図 9-1 では OIS ハブ）へ送信されます。このサーバーは、メッセージを必要とするすべてのアプリケーションに対してメッセージ・サーバーとして機能します。このハブ・アンド・スポーク・アーキテクチャを介して、XML メッセージを、疎結合された複数の受信側アプリケーションへ非同期に送信できます。

図 9-1 に、AQ を使用して、次のアプリケーションを介して配送した XML ペイロード・メッセージを示します。

- iDAP を使用し、HTTP 接続を介した AQ 操作を行う Web ベースのアプリケーション
- Net* 接続を介して XML メッセージを伝播するため、AQ を使用する会計アプリケーション
- AQ を使用して、HTTP を介して iDAP/XML メッセージを直接データベースへ伝播する出荷および在庫アプリケーション

図 9-1 AQ および XML メッセージ・ペイロード



* iDAP = AQ操作 + データ

アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にする AQ

今日の企業が直面している重大な問題の 1 つは、アプリケーションの統合です。アプリケーションの統合には、複雑な業務トランザクションを実行するための、複数の部門アプリケーションの協調、調整および同期化が伴います。

AQ は、アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にします。このアーキテクチャを使用すると、統合化ソリューションの管理、構成およびビジネス・ニーズの変化に伴う変更が容易になります。

監査、追跡およびマイニング用に保存できるメッセージ

AQ が提供するメッセージ管理では、異なるアプリケーション間でのメッセージ・フローを管理するだけでなく、将来のビジネス・インテリジェンスの監査、追跡および抽出のためにメッセージを保存することもできます。

SQL ビューを使用したメッセージ内容の表示

AQ は、メッセージを参照するための SQL ビューも提供します。これらの SQL ビューは、システムにおける過去、現在および将来の動向を分析するために使用できます。

AQ を使用するメリット

AQ を使用すると、異なるアプリケーション間での通信を柔軟に構成できます。

iDAP

今回のリリースから、iDAP を使用して、インターネット上で AQ 操作を実行できるようになりました。iDAP は、XML を使用してメッセージ構造を定義します。iDAP によって構造化されたメッセージは、HTTP などの転送プロトコルを使用してインターネット上で転送されます。

XML および iDAP インタフェース

iDAP は、XML エンコーディングされたメソッド要求を本体に含むために Content-Type ヘッダーに `text/xml` を使用します。XML は、iDAP 要求および応答メッセージの表示方法を次のとおり定義します。

- すべてのプロトコル・タグは、iDAP 名前空間に有効です。
- 送信者は、iDAP 要素および属性に名前空間を含めます。
- 受信者は、適切な名前空間を含む iDAP メッセージを処理します。不適切な名前空間を持つ要求の場合、要求が無効であるというエラーを戻します。
- 受信者は、名前空間を含まない iDAP メッセージの場合でも、コンテキストが妥当であるときは、適切な名前空間を含む場合と同様に処理します。
- iDAP 名前空間の値は、`http://ns.oracle.com/AQ/schemas/envelope` です。
- iDAP 起動の要求を形成する XML 文書では、DTD または XML Schema を使用する必要はありません。

> **参照：** 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

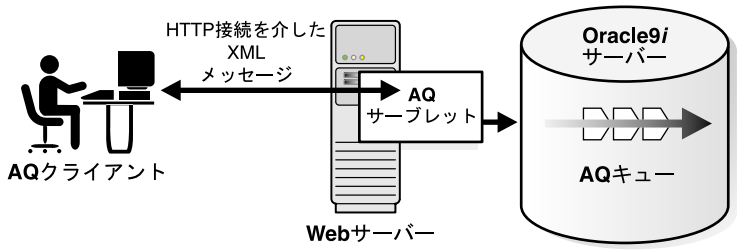
iDAP アーキテクチャ

図 9-2 に、HTTP メッセージを送信するために必要な次のコンポーネントを示します。

- iDAP 形式に準拠した XML メッセージを AQ サーブレットに送信するクライアント・プログラム。これには、Web ブラウザなど任意の HTTP クライアントが含まれます。
- Apache JServ または Tomcat など、受信した XML メッセージを解析するための AQ サーブレットのホスト Web サーバーまたはサーブレット・コンテナ。
- Oracle。AQ サーブレットは、このサーバーに接続し、キューに対して操作を実行します。

AQ クライアント・プログラムは、XML メッセージ（iDAP に準拠）を AQ サーブレットに送信します。Web ブラウザなど、任意の HTTP クライアントを使用できます。AQ サーブレットのホスト Web サーバー / サーブレット・コンテナ（Apache JServ、Tomcat など）は、受信した XML メッセージを解析します。AQ サーブレットは、Oracle データベース・サーバーに接続し、ユーザーのキューに対して操作を実行します。

図 9-2 HTTP を使用して AQ 操作を実行するための iDAP アーキテクチャ



iDAP メソッドの起動

メソッドの起動は、要求のヘッダーおよび本体を作成し、戻された応答のヘッダーおよび本体を処理することによって実行されます。要求および応答のヘッダーは、標準の転送プロトコル固有のヘッダーおよび拡張ヘッダーで構成されます。

HTTP ヘッダー

HTTP 要求のヘッダー内の POST メソッドによって、iDAP のメソッドが起動されます。要求には、ヘッダー IDAPMethodName を含める必要があります。このヘッダーの値は、ターゲットで起動するメソッドを示します。値は、次に示すとおり、URI の後に「#」が続き、その後にメソッド名が続きます（メソッド名に「#」を含めることはできません）。

```
IDAPMethodName: http://ns.oracle.com/AQ/schemas/access#AQXmlSend
```

インタフェースに使用する URI は、暗黙的な、またはペイロードの IDAP:Body 部分に指定された、メソッド名要素の名前空間修飾と一致する必要があります。

iDAP メッセージの構造

iDAP は、メッセージ要求または応答を次のとおり構成します。

- iDAP エンベロープ（XML ツリーのルート要素または最上位要素）
- iDAP ヘッダー（ルート下の最初の要素）
- iDAP 本体（AQ XML 文書）

iDAP エンベロープ

ルート要素であり、タグは IDAP:Envelope です。iDAP は、グローバル属性 IDAP:encodingStyle を定義します。これは、iDAP の指定で記述されたルールのかわりに使用されるシリアル化のルールを示します。この属性は、任意の要素に指定でき、その要素のすべての子要素にも（子要素自体はこの属性を含まない場合も）適用されます。IDAP:encodingStyle が省略されている場合、（親要素によってオーバーライドされないかぎり）型指定に従っていることを意味します。

名前空間で修飾されている場合、iDAP エンベロープには名前空間宣言およびその他の属性も含まれます。本体には、名前空間で修飾された他のサブ要素を続けることができます。

iDAP ヘッダー

ルート下の最初の要素であり、タグは IDAP:Header です。iDAP ヘッダーは、トランザクション ID などの必要な情報を要求とともに渡します。iDAP ヘッダーは、IDAP:Envelope という XML 要素の子としてエンコードされます。また、名前要素で識別され、名前空間で修飾されます。ヘッダー・エントリは、埋込み要素としてエンコードされます。

iDAP 本体

iDAP 本体は IDAP:Body でタグ付けされ、最初のサブ要素を含みます。このサブ要素の名前はメソッド名です。このメソッド要求要素には、各入力パラメータおよび出力パラメータ用の要素が含まれます。要素名はパラメータ名になります。本体には、エラーに関する情報を示す IDAP:Fault も含まれます。

AQ 操作を実行するには、iDAP 本体に AQ XML 文書を含める必要があります。AQ XML 文書の名前空間は、http://ns.oracle.com/AQ/schemas/access となります。

iDAP メソッドの起動本体 : iDAP ペイロード

iDAP メソッドの起動は、メソッド要求およびメソッド応答（オプション）で構成されます。iDAP のメソッド要求およびメソッド応答は、それぞれ HTTP 要求および応答であり、内容はルート要素および必須の本体要素で構成される XML 文書です。この章では、この XML 文書を iDAP ペイロードと呼びます。

iDAP ペイロードの定義は、次のとおりです。

- iDAP のルート要素は、XML ツリーの最上位要素です。
- iDAP ペイロードのヘッダーには、要求とともに送信する必要がある追加情報が含まれます。
- メソッド要求は、XML 要素およびパラメータの追加要素として表されます。メソッド要求は、IDAP:Body 要素の最初の子になります。この要求は、次の項に示す AQ XML クライアント要求のいずれかです。
- 応答は、クライアントに戻される戻り値、またはエラー / 例外です。エンコーディング規則は、次のとおりです。

要求 : 受信サイトでの結果

受信サイトでは、要求の結果は次のいずれかになります。

a. 受信サイトの HTTP インフラストラクチャが、要求を受信および処理しました。HTTP インフラストラクチャが、iDAP インフラストラクチャにヘッダーおよび本体を渡します。

b. 受信サイトの HTTP インフラストラクチャが、要求を受信および処理できませんでした。HTTP 応答の状態フィールドには HTTP エラーが含まれます。XML の本体は含まれません。

c. 受信サイトの iDAP インフラストラクチャが、入力パラメータをデコードし、サーバー・アドレスに示されている適切なサーバーにディスパッチし、メソッド要求に示されているメソッドに意味的に対応するアプリケーション・レベルのファンクションを起動しました。メソッド要求の結果は、応答またはエラーで構成されます。

d. 受信サイトの iDAP インフラストラクチャは、入力パラメータをデコードし、サーバー・アドレスに示されている適切なサーバーにディスパッチし、メソッド要求に示されているインタフェースまたはメソッドに意味的に対応するアプリケーション・レベルのファンクションを起動できませんでした。メソッドの結果はエラーになり、受信側のインフラストラクチャのディスパッチが正常終了できません。

(c) および (d) の場合、拡張性の目的で、要求の結果に追加のメッセージ・ヘッダーが表示される場合があります。

メソッド要求の結果

要求の結果は、要求 / 応答のフォームで提供されます。HTTP 応答では、Content-Type が text/xml である必要があります。

iDAP の結果は成功を示し、エラーは失敗を示します。メソッド応答には、結果とエラーの両方が含まれることはありません。様々な応答およびエラーのタイプについては、次の項で説明します。

AQ XML 文書である iDAP のメッセージ本体

iDAP メッセージの本体は、次のことを表す AQ XML 文書です。

- クライアントによるエンキュー、デキューおよび登録の要求
- クライアントによるエンキュー、デキューおよび登録の要求に対するサーバーの応答
- サーバーからクライアントへの通知

注意： AQ インターネット・アクセスは 8.1 形式のキューでのみサポートされています。iDAP を使用して 8.0 形式のキューにアクセスすることはできません。

クライアントによる iDAP エンキュー要求

クライアントによるエンキューの要求（SEND 要求および PUBLISH 要求）には、次のメソッドを使用します。

- AQXmlSend - シングル・コンシューマ・キューへのエンキュー
- AQXmlPublish - マルチ・コンシューマ・キュー / トピックへのエンキュー

表 9-1 に、AQXmlSend および AQXmlPublish が取る引数および引数属性を示します。必須の引数は太字で示します。

表 9-1 クライアントによる iDAP エンキュー要求 - AQXmlSend および AQXmlPublish の引数および属性

引数	属性
producer_options	**destination** - メッセージを送信するキュー / トピックを指定します。宛先要素には、宛先要素の値の解析方法を決める属性 lookup_type が含まれます。 ■ DATABASE（デフォルト）- 宛先は schema.queue_name と解析されます。 ■ LDAP - LDAP サーバーを使用して、宛先を解決します。 visibility ■ ON_COMMIT - エンキューは現行のトランザクションの一部になります。トランザクションをコミットすると、操作が完了します。これはデフォルトです。 ■ IMMEDIATE - 要求の完了直後に、エンキューが反映されます。エンキューは、現行のトランザクションの一部になりません。その操作のみで 1 つのトランザクションを構成します。 transformation - メッセージをエンキューする前に、PL/SQL 変換を開始します。
message_set - 1 つ以上のメッセージが含まれます。	各メッセージは、message_header および message_payload で構成されます。
■ message_header	message_id - デキュー中に提供される、メッセージの一意の識別子。 correlation - メッセージの相関識別子。 expiration - メッセージのデキューが可能な時間（秒）。このパラメータは、遅延に対するオフセットです。デフォルトでは、メッセージに期限はありません。 期限切れになる前にデキューされない場合、メッセージは EXPIRED 状態で例外キューに移されます。 delay - メッセージの処理が可能な時間（秒）。 priority - メッセージの優先順位。数値が小さいほど、優先順位が高いことを表します。優先順位には、負の数を含むすべての数値を指定できます。

表 9-1 クライアントによる iDAP エンキュー要求 - AQXmlSend および AQXmlPublish の引数および属性（続き）

引数	属性
	sender_id - アプリケーション固有の識別子。 ■ agent_name、address、protocol ■ agent_alias - 指定すると、LDAP を使用して、名前、アドレス、プロトコルが変換されます。 recipient_list - デフォルトのサブスクライバ・リストをオーバーライドします。lookup_type には、受信者が LDAP で指定または検索されるかどうかを指定できます。 ■ agent_name、address、protocol ■ agent_alias - 指定すると、LDAP を使用して、名前、アドレス、プロトコルが変換されます。 message_state - デキュー中にメッセージの状態が自動的に書き込まれます。 0: メッセージはすぐに処理できます。 1: メッセージ遅延に到達していません。 2: メッセージは処理され、保存されています。 3: メッセージは例外キューに移されています。 exception_queue - 例外が発生した場合に、正常に処理されなかったメッセージが移されるキューの名前。メッセージのデキューが正常に行われなかった回数が max_retries を超えたり、メッセージが期限切れになった場合、または例外キューのすべてのメッセージが EXPIRED 状態にある場合に、メッセージは移されます。 デフォルト値は、キュー・テーブルに対応付けられた例外キューです。メッセージの移動時に指定された例外キューが存在しない場合、メッセージはキュー・テーブルに対応付けられます。デフォルトの例外キューに移され、アラート・ファイルに警告が記録されます。デフォルトの例外キューが使用されると、デキュー時にパラメータが NULL 値を戻します。
■ message_payload	宛先キュー / トピックのペイロード型に基づいて、異なるサブ要素を持つことができます。様々なペイロード型については、次の項を参照してください。
AQXmlCommit	これは空要素です。指定すると、要求の終了時にユーザー・トランザクションがコミットされます。

メッセージ・ペイロード

AQ では、次のメッセージ型がサポートされています。

- RAW キュー
- Oracle オブジェクト型（ADT）キュー
- JMS 型のキュー / トピック

これらすべての型のキューに iDAP を使用してアクセスできます。キューに RAW、Oracle オブジェクトまたは JMS フォーマットのメッセージが含まれる場合、XML ペイロードはエンキュー中に適切な内部フォーマットに変換され、キューに格納されます。デキュー中、前述のいずれかのフォーマットのメッセージを含むキューからメッセージが取得された場合、そのメッセージはクライアントに送信される前に XML に変換されます。

メッセージ・ペイロード型は、操作が実行されているキューの型によって異なります。次の項では、キューの型について説明します。

RAW キュー

RAW キューの内容は、バイト列です。XML メッセージでは、メッセージ・ペイロードを 16 進数表示（たとえば、`<raw>023f4523</raw>`）にする必要があります。

Oracle オブジェクト型（ADT）キュー

JMS キューではない（`AQ$_JMS_*` 型ではない）ADT キューの場合、ペイロードの型は、キューを保持するキュー・テーブルの作成中に指定した型によって異なります。この場合に指定されている XML は、キュー・テーブルに対するペイロードの SQL 型にマップする必要があります。

> **参照：** XML に対する SQL 型のマッピングの詳細は、次の章を参照してください。
>
> - 第 5 章「XML に対するデータベース・サポート」
> - 第 7 章「XSU」

ADT キューの例 キューの型を EMP_TYP と定義し、次のような構造であると想定します。

```
create or replace type emp_typ as object (
     empno NUMBER(4),
     ename VARCHAR2(10),
     job VARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal   NUMBER(7,2),
     comm  NUMBER(7,2)
     deptno NUMBER(2));
```

対応する XML 表現は、次のとおりです。

```
<EMP_TYP>
  <EMPNO>1111</EMPNO>
  <ENAME>Mary</ENAME>
  <MGR>5000</MGR>
  <HIREDATE>1996-01-01 0:0:0</HIREDATE>
  <SAL>10000</SAL>
  <COMM>100.12</COMM>
  <DEPTNO>60</DEPTNO>
</EMP_TYP>
```

JMS 型のキュー / トピック

JMS 型のキュー（ペイロード型が AQ$_JMS_* のキュー）の場合、JMS 型に応じて 4 つの XML 要素があります。iDAP では、次の JMS 型を持つキュー / トピックがサポートされています。

- TextMessage
- MapMessage
- BytesMessage
- ObjectMessage

注意： StreamMessage 型のペイロードを持つ JMS キューは、iDAP を介してはサポートされません。

表 9-2 に、JMS 型および XML コンポーネントを示します。各列には、各 JMS 型に固有の XML 要素を示します。必須の要素は太字で示します。

表 9-2 JMS 型および XML コンポーネント

jms_text_message	jms_map_message	jms_bytes_message	jms_object_message
次のペイロード型のキュー / トピックに使用	-	-	-
AQ$_JMS_TEXT_MESSAGE	AQ$_JMS_MAP_MESSAGE	AQ$_JMS_BYTES_MESSAGE	AQ$_JMS_OBJECT_MESSAGE
oracle_jms_properties	-	-	-
user_properties	-	-	-
text_data - Text ペイロードを表す文字列	**map_data** - 次の名前と値の組（項目）の集合 ■ name ■ int_value string_value long_value double_value boolean_value float_value short_value byte_value	**bytes_data** - ペイロード・バイトの 16 進数表示	**ser_object_data** - シリアル化オブジェクトの 16 進数表示

すべての JMS メッセージは、次の共通の要素で構成されます。

- 次の要素で構成される oracle_jms_properties。
  - type - メッセージの型。
  - reply_to - agent_name、address および protocol で構成される返信先。
  - userid - AQ によって提供されるユーザー ID。クライアントは指定できません。
  - appid - アプリケーションの識別子。
  - groupid - グループの識別子。

- group_sequence - group_id で識別されるグループ内の順序。
- timestamp - メッセージが送信された時刻。エンキュー中は指定できません。デキューされるメッセージに自動的に移入されます。
- recv_timestamp - メッセージが受信された時刻。

- user_properties - 前述の事前定義のプロパティの他に、独自のメッセージ・プロパティを名前と値の組として指定できます。user_properties は、プロパティ要素のリストで構成されます。各プロパティは、次の名前と値の組で構成されます。
  - name - プロパティ名
  - int_value - プロパティ値（Integer）

    string_value - プロパティ値（String）

    long_value - プロパティ値（Long）

    double_value - プロパティ値（Double）

    boolean_value - プロパティ値（Boolean）

    float_value - プロパティ値（Float）

    short_value - プロパティ値（Short）

    byte_value - プロパティ値（Byte）

次の項では、様々なメッセージ型およびキュー型を使用したエンキュー要求の例を示します。

iDAP エンキュー要求の例 1 - シングル・コンシューマ・キューへの ADT メッセージ

キュー QS.NEW_ORDER_QUE のペイロード型は、ORDER_TYP です。

```
<?xml version="1.0"?>
  <Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
     <Body>
       <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
         <producer_options>
           <destination>QS.NEW_ORDERS_QUE</destination>
         </producer_options>

         <message_set>
           <message_count>1</message_count>

           <message>
             <message_number>1</message_number>
```

```
<message_header>
  <correlation>ORDER1</correlation>
  <sender_id>
    <agent_name>scott</agent_name>
  </sender_id>
</message_header>

<message_payload>

 <ORDER_TYP>
       <ORDERNO>100</ORDERNO>
       <STATUS>NEW</STATUS>
       <ORDERTYPE>URGENT</ORDERTYPE>
       <ORDERREGION>EAST</ORDERREGION>
       <CUSTOMER>
          <CUSTNO>1001233</CUSTNO>
          <CUSTID>MA1234555623212</CUSTID>
          <NAME>AMERICAN EXPRESS</NAME>
          <STREET>EXPRESS STREET</STREET>
          <CITY>REDWOOD CITY</CITY>
          <STATE>CA</STATE>
          <ZIP>94065</ZIP>
          <COUNTRY>USA</COUNTRY>
       </CUSTOMER>
       <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
       <ITEMS>
          <ITEMS_ITEM>
             <QUANTITY>10</QUANTITY>
             <ITEM>
                <TITLE>Perl</TITLE>
                <AUTHORS>Randal</AUTHORS>
                <ISBN>ISBN20200</ISBN>
                <PRICE>19</PRICE>
             </ITEM>
             <SUBTOTAL>190</SUBTOTAL>
          </ITEMS_ITEM>
          <ITEMS_ITEM>
             <QUANTITY>20</QUANTITY>
             <ITEM>
                <TITLE>XML</TITLE>
                <AUTHORS>Micheal</AUTHORS>
                <ISBN>ISBN20212</ISBN>
                <PRICE>59</PRICE>
             </ITEM>
             <SUBTOTAL>590</SUBTOTAL>
          </ITEMS_ITEM>
       </ITEMS>
```

```
                    <CCNUMBER>NUMBER01</CCNUMBER>
                    <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
                  </ORDER_TYP>
                </message_payload>
              </message>
            </message_set>
          </AQXmlSend>
        </Body>
</Envelope>
```

iDAP エンキュー要求の例 2 - マルチ・コンシューマ・キューへのメッセージ

マルチ・コンシューマ・キュー AQUSER.EMP_TOPIC のペイロード型は、EMP_TYP です。EMP_TYP の構造は、次のとおりです。

```
create or replace type emp_typ as object (
     empno NUMBER(4),
     ename VARCHAR2(10),
     job VARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal   NUMBER(7,2),
     comm  NUMBER(7,2)
     deptno NUMBER(2));
```

PUBLISH 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
          </producer_options>

              <message_set>
                <message_count>1</message_count>

                <message>
                  <message_number>1</message_number>

                  <message_header>
                     <correlation>NEWEMP</correlation>
                     <sender_id>
                        <agent_name>scott</agent_name>
                     </sender_id>
                  </message_header>
```

```
                    <message_payload>
                    <EMP_TYP>
                      <EMPNO>1111</EMPNO>
                      <ENAME>Mary</ENAME>
                      <MGR>5000</MGR>
                      <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                      <SAL>10000</SAL>
                      <COMM>100.12</COMM>
                      <DEPTNO>60</DEPTNO>
                    </EMP_TYP>
                   </message_payload>
                  </message>
                </message_set>
              </AQXmlPublish>
            </Body>
        </Envelope>
```

iDAP エンキュー要求の例 3 - JMS キューへのメッセージの送信

JMS キュー AQUSER.JMS_TEXTQ のペイロード型は、JMS の TextMessage（SYS.AQ$_JMS_TEXT_MESSAGE）です。SEND 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>

        <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.JMS_TEXTQ</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

              <message_header>
                <correlation>text_msg</correlation>
                  <sender_id>
                     <agent_name>john</agent_name>
                  </sender_id>
              </message_header>

              <message_payload>
```

```
                <jms_text_message>
                  <oracle_jms_properties>
                   <appid>AQProduct</appid>
                   <groupid>AQ</groupid>
                  </oracle_jms_properties>

                  <user_properties>
                    <property>
                      <name>Country</name>
                      <string_value>USA</string_value>
                    </property>
                    <property>
                       <name>State</name>
                      <string_value>California</string_value>
                    </property>
                   </user_properties>

                    <text_data>All things bright and beautiful</text_data>
                 </_text_message>
               </message_payload>
             </message>
           </message_set>
         </AQXmlSend>
       </Body>
</Envelope>
```

iDAP エンキュー要求の例 4 - 送信 / 公開およびコミット

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

               <message_header>
          <correlation>NEWEMP</correlation>
          <sender_id>
             <agent_name>scott</agent_name>
```

```
            </sender_id>
                </message_header>

                <message_payload>
                  <EMP_TYP>
                    <EMPNO>1111</EMPNO>
                    <ENAME>Mary</ENAME>
                    <MGR>5000</MGR>
                    <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                    <SAL>10000</SAL>
                    <COMM>100.12</COMM>
                    <DEPTNO>60</DEPTNO>
                  </EMP_TYP>
                </message_payload>
              </message>
            </message_set>

          <AQXmlCommit/>

          </AQXmlPublish>
        </Body>
</Envelope>
```

クライアントによる iDAP デキュー要求

クライアントによるデキューの要求には、AQXmlReceive メソッドを使用します。表 9-3 に、このメソッドが取る引数および引数属性を示します。必須の引数は太字で示します。

表 9-3 クライアントによる iDAP デキュー要求 - AQXmlReceive の引数および引数属性

引数	属性
consumer_options	**destination** - メッセージを受信するキュー / トピックを指定します。宛先要素には、宛先要素の値の解析方法を決める属性 lookup_type が含まれます。 ■ DATABASE（デフォルト）- 宛先は schema.queue_name と解析されます。 ■ LDAP - LDAP サーバーを使用して、宛先を解決します。 consumer_name - コンシューマの名前。コンシューマ名に一致するメッセージのみがアクセスされます。キューがマルチ・コンシューマ用に設定されていない場合は、このフィールドを指定しないでください。 wait_time - 検索基準に一致するメッセージが現在存在しない場合の待機時間（秒）。 selector - メッセージを選択するために使用する基準。次のいずれかに指定します。 ■ correlation - デキューするメッセージの相関識別子 ■ message_id - デキューするメッセージのメッセージ識別子 ■ condition - この条件を満たすデキュー・メッセージ 条件は、SQL 問合せの WHERE 句に類似した構文を使用して、ブール式で指定します。このブール式には、メッセージ・プロパティ、ユーザー・データ・プロパティ（オブジェクト・ペイロードのみ）および（SQL 問合せの WHERE 句に指定する）PL/SQL ファンクションまたは SQL 関数を含めることができます。メッセージ・プロパティには、priority、corrid およびキュー・テーブル内の他の列を含めることができます。 メッセージ・ペイロード（オブジェクト・ペイロード）にデキュー条件を指定するには、句にオブジェクト型の属性を使用します。各属性には、ペイロードを格納するキュー・テーブルの特定の列を示す修飾子として、接頭辞 tab.user_data を付ける必要があります。deq_condition パラメータは、4,000 文字に制限されています。 visibility ■ ON_COMMIT（デフォルト）- デキューは現行のトランザクションの一部になります。トランザクションをコミットすると、操作が完了します。 ■ IMMEDIATE - 要求の完了直後に、デキューが反映されます。デキューは、現行のトランザクションの一部になりません。その操作のみで 1 つのトランザクションを構成します。

表 9-3 クライアントによる iDAP デキュー要求 - AQXmlReceive の引数および引数属性（続き）

引数	属性
	dequeue_mode - デキューに対応付けられたロック動作を指定します。dequeue_mode は、次のいずれかに指定できます。 ■ REMOVE（デフォルト）- メッセージを読み取り、そのメッセージを更新または削除します。これはデフォルトです。メッセージは、保存プロパティに基づいて、キュー・テーブルに保存できます。 ■ BROWSE - メッセージのロックを取得することなく、メッセージを読み取ります。これは、SELECT 文と同等です。 ■ LOCKED: メッセージの書込みロックを読み取り、取得します。ロックはトランザクションが継続している間、有効です。これは、UPDATE 文の SELECT 文と同等です。 navigation_mode - 取り出すメッセージの位置を指定します。まず、位置を決めます。次に、検索基準を適用します。最後に、メッセージを取り出します。navigation_mode は、次のいずれかに指定できます。 ■ FIRST_MESSAGE - 使用可能で検索基準に一致する最初のメッセージを取り出します。これによって、位置がキューの先頭にリセットされます。 ■ NEXT_MESSAGE（デフォルト） - 使用可能で検索基準に一致する次のメッセージを取り出します。前のメッセージがメッセージ・グループに属する場合、AQ は、そのメッセージ・グループに属するメッセージの中から、検索基準に一致する次の使用可能なメッセージを取り出します。これはデフォルトです。 ■ NEXT_TRANSACTION - 現行のトランザクション・グループ（ある場合）の残りをスキップし、次のトランザクション・グループの最初のメッセージを取り出します。このオプションは、現行のキューでメッセージのグループ化が可能な場合のみに使用できます。 transformation - メッセージをデキューした後に、PL/SQL 変換を開始します。

次の項では、様々な AQXmlReceive 属性を使用したデキュー要求の例を示します。

iDAP デキュー要求の例 1 - シングル・コンシューマ・キューからのメッセージ

シングル・コンシューマ・キュー QS.NEW_ORDERS_QUE を使用する場合、RECEIVE 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

iDAP デキュー要求の例 2 - 特定の条件を満たすメッセージ

マルチ・コンシューマ・キュー AQUSER.EMP_TOPIC と、サブスクライバ APP1 および条件 deptno=60 を使用する場合、RECEIVE 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
            <consumer_name>APP1</consumer_name>
            <wait_time>0</wait_time>
            <selector>
                 <condition>tab.user_data.deptno=60</condition>
            </selector>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

iDAP デキュー要求の例 3 - メッセージの受信およびコミット

デキュー要求の例で、RECEIVE 要求の最後に AQXmlCommit を含めると、操作の完了後にトランザクションがコミットされます。9-24 ページの「iDAP デキュー要求の例 1 - シングル・コンシューマ・キューからのメッセージ」の場合、次に示すとおり、RECEIVE 要求にコミット・フラグを含めることができます。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
          </consumer_options>

          <AQXmlCommit/>

        </AQXmlReceive>
      </Body>
</Envelope>
```

iDAP デキュー要求の例 4 - メッセージのブラウズ

デフォルトでは、メッセージは REMOVE モードでデキューされます。BROWSE モードで QS.NEW_ORDERS_QUE からメッセージを受信するには、RECEIVE 要求を次のとおり変更します。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
            <dequeue_mode>BROWSE</dequeue_mode>
          </consumer_options>
         </AQXmlReceive>
      </Body>
</Envelope>
```

クライアントによる iDAP 登録要求

クライアントによる登録の要求には、AQXmlRegister メソッドを使用します。表 9-4 に、このメソッドが取る引数および引数属性を示します。必須の引数は太字で示します。

表 9-4 クライアント登録 - AQXmlRegister の引数および引数属性

引数	属性
register_options	**destination** - 通知を登録するキューまたはトピックを指定します。宛先要素には、宛先要素の値の解析方法を決める属性 lookup_type が含まれます。 ■ DATABASE（デフォルト）- 宛先は schema.queue_name と解析されます。 ■ LDAP - LDAP サーバーを使用して、宛先を解決します。 consumer_name - マルチ・コンシューマ・キューまたはトピックのコンシューマ名。シングル・コンシューマ・キューの場合は、このパラメータを指定しないでください。 **notify_url** - メッセージのエンキュー時に通知を送信する場所。フォームは、http://<url>、mailto://<email address> または plsql://<pl/sql procedure> です。

iDAP 登録要求の例 1 - 電子メール・アドレスでの通知の登録

キュー AQUSER.EMP_TOPIC の APP1 コンシューマにエンキューされているメッセージの電子メール・アドレスを通知する場合、REGISTER 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
      <Body>

        <AQXmlRegister xmlns = "http://ns.oracle.com/AQ/schemas/access">

          <register_options>
            <destination>AQUSER.EMP_TOPIC</destination>
            <consumer_name>APP1</consumer_name>
            <notify_url>mailto:app1@hotmail.com</notify_url>
          </register_options>

          <AQXmlCommit/>

        </AQXmlRegister>
      </Body>
</Envelope>
```

コミット要求

セッション中にユーザーが実行するすべてのアクションのコミット要求には、AQXmlCommit メソッドを使用します。

コミット要求の例

COMMIT 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlCommit xmlns="http://ns.oracle.com/AQ/schemas/access"/>
   </Body>
</Envelope>
```

ロールバック要求

セッション中にユーザーが実行するすべてのアクションのロールバック要求には、AQXmlRollback メソッドを使用します。visibility を IMMEDIATE に設定して実行するアクションは、ロールバックされません。

ロールバック要求の例

ROLLBACK 要求のフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlRollback xmlns="http://ns.oracle.com/AQ/schemas/access"/>
   </Body>
</Envelope>
```

サーバーによる iDAP エンキュー応答

シングル・コンシューマ・キューへのエンキュー要求の応答には、AQXmlSendResponse メソッドを使用します。表 9-5 に、応答のコンポーネントを示します。

表 9-5 シングル・コンシューマ・キューへのサーバーのエンキュー応答（AQXmlSendResponse）

応答	属性
status_response	status_code - 成功（0）または失敗（-1）
	error_code - エラーの Oracle コード
	error_message - エラーの説明
send_result	destination - メッセージの送信先
	message_id - 送信された各メッセージの識別子

サーバーによる iDAP 要求の例 1 - シングル・コンシューマ・キューへのエンキュー

シングル・コンシューマ・キュー QS.NEW_ORDERS_QUE への SEND 要求の結果のフォーマットは、次のとおりです。

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlSendResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <send_result>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <message_id>12341234123412341234</message_id>
         </send_result>
      </AQXmlSendResponse>
   </Body>
</Envelope>
```

マルチ・コンシューマ・キューまたはトピックへのエンキュー要求の応答には、AQXmlPublishResponse メソッドを使用します。表 9-6 に、応答のコンポーネントを示します。

表 9-6 マルチ・コンシューマ・キューまたはトピックへのサーバーのエンキュー応答（AQXmlPublishResponse）

応答	属性
status_response	status_code - 成功（0）または失敗（-1）
	error_code - エラーの Oracle コード
	error_message - エラーの説明
publish_result	destination - メッセージの送信先
	message_id - 送信された各メッセージの識別子

サーバーによる iDAP 要求の例 2 - マルチ・コンシューマ・キューへのエンキュー

マルチ・コンシューマ・キュー AQUSER.EMP_TOPIC への PUBLISH 要求の結果のフォーマットは、次のとおりです。

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlPublishResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <publish_result>
            <destination>AQUSER.EMP_TOPIC</destination>
            <message_id>23434435435456546546546546</message_id>
         </publish_result>
      </AQXmlPublishResponse>
   </Body>
</Envelope>
```

デキュー要求へのサーバー応答

デキュー要求への応答には、AQXmlReceiveResponse メソッドを使用します。表 9-7 に、応答のコンポーネントを示します。

表 9-7 キューまたはトピックからのデキューへのサーバー応答（AQXmlReceiveResponse）

応答	属性
status_response	status_code - 成功（0）または失敗（-1）
	error_code - エラーの Oracle コード
	error_message - エラーの説明
receive_result	destination - メッセージの送信先
	message_set - デキューされたメッセージの集合

サーバーによる iDAP デキュー応答の例 1 - ADT キューからのメッセージ

ペイロード型が EMP_TYP のキュー AQUSER.EMP_TOPIC に対する RECEIVE 要求の結果のフォーマットは、次のとおりです。

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlReceiveResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <receive_result>
            <destination>AQUSER.EMP_TOPIC</destination>
            <message_set>
               <message_count>1</message_count>
               <message>
                  <message_number>1</message_number>
                  <message_header>
                     <message_id>1234344545565667</message_id>
                     <correlation>TKAXAP10</correlation>
                     <priority>1</priority>
                     <delivery_count>0</delivery_count>
                     <sender_id>
                        <agent_name>scott</agent_name>
                     </sender_id>
                     <message_state>0</message_state>
                  </message_header>
                  <message_payload>
                    <EMP_TYP>
```

```
                        <EMPNO>1111</EMPNO>
                        <ENAME>Mary</ENAME>
                        <MGR>5000</MGR>
                        <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                        <SAL>10000</SAL>
                        <COMM>100.12</COMM>
                        <DEPTNO>60</DEPTNO>
                     </EMP_TYP>
                  </message_payload>
               </message>
            </message_set>
         </receive_result>
      </AQXmlReceiveResponse>
   </Body>
</Envelope>
```

登録要求へのサーバー応答

登録要求への応答には、AQXmlRegisterResponse メソッドを使用します。このメソッドは、status_response で構成されます（status_response については、表 9-7 を参照）。

コミットの応答

コミット要求への応答には、AQXmlCommitResponse メソッドを使用します。このメソッドは、status_response で構成されます（status_response については、表 9-7 を参照）。

例

COMMIT 要求への応答フォーマットは、次のとおりです。

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlCommitResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
      </AQXmlCommitResponse>
   </Body>
</Envelope>
```

ロールバック応答

ロールバック要求への応答には、AQXmlRollbackResponse メソッドを使用します。このメソッドは、status_response で構成されます（status_response については、表 9-7 を参照）。

通知

クライアントが登録したイベントが発生した場合、REGISTER 要求で指定した URL にクライアントへの通知が送信されます。AQXmlNotification は、次の要素で構成されます。

- 次の要素で構成される notification_options
  - destination - イベントが発生した宛先のキュー / トピック
  - consumer_name - イベントが発生したコンシューマ名（マルチ・コンシューマ・キュー / トピックの場合）
- message_set - メッセージ・プロパティの集合

エラー発生時の応答

前述のいずれかの要求でエラーが発生した場合、FAULT 要素が生成されます。FAULT 要素は、次の要素で構成されます。

- faultcode - 失敗のエラー・コード。
- faultstring - クライアント・エラーまたはサーバー・エラー。クライアント・エラーは、要求が無効であることを示します。サーバー・エラーは、AQ サーブレットが正しく設定されていないことを示します。
- 次の要素で構成される detail
  - status_response

iDAP スキーマおよび AQ XML Schema

iDAP は、iDAP スキーマおよび AQ XML Schema をクライアントに公開します。パーサーによって送信されるすべてのドキュメントは、これらのスキーマに対して妥当性が検証されます。

- **iDAP スキーマ** - `http://ns.oracle.com/AQ/schemas/envelope`

  これは、エンベロープ、ヘッダーおよび本体で構成されるドキュメントの構造を記述します。

- **AQ XML Schema** - `http://ns.oracle.com/AQ/schemas/access`

  これは、AQ 機能にインターネット・アクセスするための iDAP 本体の内容を記述します。

  参照：『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

AQXMLServlet

`AQXMLServlet` は、`oracle.AQ.xml.AQxmlServlet` クラスを拡張する Java クラスです。`AQxmlServlet` クラスは、`javax.servlet.http.HttpServlet` クラスを拡張するクラスです。

参照： `AQXMLServlet` の作成および配置の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

HTTP を使用した AQXMLServlet へのアクセス

参照： HTTP を使用して XML メッセージを受信するための AQ の設定については、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

HTTP を使用した AQ サーブレットへの AQ クライアント要求の手順

AQ クライアントが HTTP を使用して AQ サーブレットに要求を行う一般的な手順は、次のとおりです。

1. AQ クライアントがサーバーに HTTP(S) 接続します。次に例を示します。

   ```
   https://aq.us.oracle.com:8000/aqserv/servlet/AQTestServlet
   ```

   これによって、`aq.us.oracle.com` のポート 8000 への接続がオープンされます。

2. AQ クライアントが、次のいずれかの方法でサーバーにログインします。
   - HTTP 基本認証（SSL を使用する場合も、使用しない場合も含む）
   - SSL 証明書ベースのクライアント認証
3. AQ クライアントが、SEND、PUBLISH、RECEIVE または REGISTER 要求を表す XML メッセージを構成します。次に例を示します。

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
      <Body>
        <AQXmlSend  xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>OE.OE_NEW_ORDERS_QUE</destination>
          </producer_options>
          <message_set>
            <message_count>1</message_count>
            <message>
              <message_number>1</message_number>
              <message_header>
                <correlation>XML_ADT_SINGLE_ENQ</correlation>
                 <sender_id>
                     <agent_name>john</agent_name>
                 </sender_id>
              </message_header>
              <message_payload>
              <ORDER_TYP>
                     <ORDERNO>100</ORDERNO>
                     <STATUS>NEW</STATUS>
                     <ORDERTYPE>NORMAL</ORDERTYPE>
                     <ORDERREGION>EAST</ORDERREGION>
                     <CUSTOMER>
                        <CUSTNO>1001233</CUSTNO>
                        <CUSTID>JOHN</CUSTID>
                        <NAME>AMERICAN EXPRESS</NAME>
                        <STREET>EXPRESS STREET</STREET>
                        <CITY>REDWOOD CITY</CITY>
                        <STATE>CA</STATE>
                        <ZIP>94065</ZIP>
                        <COUNTRY>USA</COUNTRY>
                     </CUSTOMER>
                     <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
                     <ITEMS>
                        <ITEMS_ITEM>
                           <QUANTITY>10</QUANTITY>
                           <ITEM>
                              <TITLE>Perl</TITLE>
                              <AUTHORS>Randal</AUTHORS>
```

```
                                    <ISBN>ISBN20200</ISBN>
                                    <PRICE>19</PRICE>
                                  </ITEM>
                                  <SUBTOTAL>190</SUBTOTAL>
                                </ITEMS_ITEM>
                              </ITEMS>
                             <CCNUMBER>NUMBER01</CCNUMBER>
                             <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
                          </ORDER_TYP>
                        </message_payload>
                      </message>
                    </message_set>
                    </AQXmlSend>
                </Body>
</Envelope>
```

4. リモート・サーバーで、AQ クライアントが、サーブレットに HTTP の `POST` 要求を送信します。

HTTP を使用した AQ サーブレットによる要求の処理

AQ サーブレットが HTTP を使用して要求を行う一般的な手順は、次のとおりです。

1. サーバーが、HTTP(S) 接続を受け入れます。
2. サーバーが、クライアントによって指定されたユーザー（AQ エージェント）を認証します。
3. サーバーが `POST` 要求を受信します。
4. AQ サーブレットが起動します。

   この要求がこのサーブレットが処理する最初の要求である場合、サーブレットが初期化されます（サーブレットの `init( )` メソッドが起動します）。`init()` メソッドは、クライアントによって提供された `AQxmlDataSource` パラメータ（SID、ホスト、ポート、AQ サーブレットのスーパー・ユーザー名、パスワード）を使用して、Oracle サーバーへの接続プールを作成します。
5. AQ サーブレットが、次のとおりメッセージを処理します。
   a. この要求がこのクライアントからの最初の要求である場合、新しい HTTP セッションが作成されます。XML メッセージが解析され、メッセージの内容が検証されます。HTTP ヘッダーにクライアントによってセッション ID が渡された場合、この操作はそのセッションのコンテキストで実行されます。詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

b. サーブレットが、エージェントが操作を実行しようとしているオブジェクト（キュー / トピック）を判断します。たとえば、クライアント要求（9-33 ページの「HTTP を使用した AQ サーブレットへの AQ クライアント要求の手順」の手順 3）では、エージェント JOHN が OE.OE_NEW_ORDERS_QUE にアクセスしようとしています。

c. サーブレットが、（AQ$INTERNET_USERS ビューを使用して）この AQ エージェントにマップするデータベース・ユーザーのリスト全体を検索します。いずれかの db_user が要求に指定されたキュー / トピックへのアクセス権限を取得している場合、AQ サーブレットのスーパー・ユーザーは、この db_user のかわりにセッションを作成します。

d. たとえば、前述の例のエージェント JOHN が DBMS_AQADM.ENABLE_DB_ACCESS コールを使用してデータベース・ユーザー OE にマップされている場合、サーブレットは、データベース・ユーザー OE の権限でエージェント JOHN 用のセッションを作成します。

e. HTTP セッションにアクティブなトランザクションがない場合、新しいデータベース・トランザクションが開始します。明示的な COMMIT または ROLLBACK 要求が行われるまで、セッションの以降の要求は同じトランザクションの一部になります。

f. 要求された操作（SEND、PUBLISH、RECEIVE、REGISTER）が実行されます。

g. 応答が XML メッセージとしてフォーマットされ、クライアントに戻されます。たとえば、前述の要求に対する応答は、次のとおりです。

```
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
 <Body>
    <AQXmlSendResponse
       xmlns="http://ns.oracle.com/AQ/schemas/access">
       <status_response>
          <status_code>0</status_code>
       </status_response>
       <send_result>
          <destination>OE.OE_NEW_ORDERS_QUE</destination>
          <message_id>12341234123412341234123412341234</message_id>
       </send_result>
    </AQXmlSendResponse>
 </Body>
</Envelope>
```

応答には、セッション ID が Cookie として HTTP ヘッダーに含まれます。たとえば、Tomcat はセッション ID を JSESSIONID=239454ds2343 として戻します。

XMLType キュー

AQ による XML 文書の格納および問合せ

AQ は、XML 文書のキューへの格納をサポートし、XML 文書を問い合せる機能を提供します。Oracle AQ では、次の 2 つの場合に XML を使用します。

- **キューに格納された XML データ :** AQ キューは XML ペイロードをサポートします。XML メッセージは、XMLType データ型として格納できます。
- **ADT ペイロードまたは RAW ペイロードを持つ既存のキューから生成された XML データ :** ADT キューまたは RAW キューに格納されたデータに対して作成された新しいアプリケーションがメッセージ形式として XML を使用する必要がある場合、これらのデータを格納したアプリケーションによって使用されます。

> **参照 :** データベースでの XML のサポートの詳細は、第 5 章「XML に対するデータベース・サポート」を参照してください。

オブジェクト型を持つメッセージ・ペイロードの構造化および管理

Oracle AQ では、メッセージのペイロードを構造化して管理するためにオブジェクト型を使用できます。厳密な型指定を持つ内容（外部の型体系によって定義される形式を持つ内容）によって、次の機能が使用できます。

- **内容ベースのルーティング :** AQ が内容を調査し、その内容に基づいて自動的にメッセージを別のキューにルーティングできます。
- **内容ベースのサブスクリプション :** パブリッシュ・サブスクライブ・システムをメッセージ・システム上に組み込んで、内容ベースのサブスクリプションを作成できます。
- **問合せ :** メッセージ内容の問合せ機能によって、メッセージ・ウェアハウスを含む、様々なアプリケーションで必要な現行および処理済のメッセージを調査できます。

XMLType 属性を含むメッセージ・ペイロード・キューの作成

XMLType 属性を含むペイロードを持つキューを作成できます。これらのペイロードは、XML 文書を含むメッセージの転送および格納に使用できます。XMLType 属性を持つ Oracle オブジェクトを定義すると、次の操作を実行できます。

- 複数の型の XML 文書を同じキュー内に格納します。この文書は、CLOB として内部的に格納されます。
- `XMLType.existsNode()`、`XMLType.extract()` などの演算子を使用して XMLType 属性を持つメッセージを選択的にデキューします。

> **参照 :** XMLType 操作の詳細は、第 5 章「XML に対するデータベース・サポート」を参照してください。

- 変換を定義して、Oracle オブジェクトを XMLType に変換します。
- XMLType.existsNode() や XMLType.extract() などの XMLType 演算子を使用して、メッセージ内容を問い合せるルールベースのサブスクライバを定義します。

XMLType キューの例 1: キュー・オブジェクト用の XMLType キュー・テーブルの作成

BooksOnline のアプリケーションで、海外向け出荷サイトでは、XMLType 属性の発注情報を使用して、注文が ORDER_XML_TYP として表現されると想定します。注文入力サイトでは、注文が Oracle オブジェクトの ORDER_TYP として表現されます。

ORDER_XML_TYP は、XMLType 属性を含むコンポジット型です。

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  SYS.XMLTYPE);
```

海外向けキュー・テーブルおよびキューは、次のとおり作成されます。

```
BEGIN
dbms_aqadm.create_queue_table(
   queue_table        => 'OS_orders_pr_mqtab',
   comment            => 'Overseas Shipping MultiConsumer Orders queue table',
   multiple_consumers => TRUE,
   queue_payload_type => 'OS.order_xml_typ',
   compatible         => '8.1');
END;

BEGIN
dbms_aqadm.create_queue (
   queue_name   => 'OS_bookedorders_que',
   queue_table  => 'OS_orders_pr_mqtab');
END;
```

AQ XML メッセージ形式の変換

異なる Oracle 型とユーザー定義型の変換を指定できます。変換は、次のいずれかの方法で作成できます。

- PL/SQL ファンクション（コールアウトを含む）。9-39 ページの「AQ メッセージ変換の例 1: PL/SQL ファンクションの作成」を参照してください。
- SQL 式
- Java ストアド・プロシージャ

いずれの場合も、戻り型はターゲットの型です。

1 対 1 のメッセージ変換のみがサポートされます。メッセージがデータベース・メッセージ・システム内を移動するため、変換エンジンは、メッセージを容易に変換するために AQ と緊密に統合されています。

AQ アプリケーションは、アプリケーションで指定した形式のキューから、メッセージをエンキューまたはデキューできます。また、アプリケーションは、キューにサブスクライブする場合に、メッセージ形式を指定することもできます。

AQ プロパゲータは、リモート・サブスクリプションで指定されたとおり、メッセージを宛先キュー・メッセージの形式に変換します。変換機能では、データベースの状態を書き込んだり、現行のトランザクションをコミットまたはロールバックすることができません。変換は、スキーマ・エクスポートまたは全データベース・エクスポートを使用してエクスポートされます。

AQ メッセージ変換の例 1: PL/SQL ファンクションの作成

注文入力サイトでは、注文が Oracle オブジェクトの ORDER_TYP として表現されます。

海外向け出荷サイトでは、OE_BOOKEDORDERS_QUE キューのメッセージがサブスクライブされるため、メッセージが注文入力サイトから海外出荷向けサイトへ伝播される前に、変換が適用されます。

ORDER_XML_TYP は、XMLType 属性を含むコンポジット型です。

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  SYS.XMLTYPE);
```

変換は、次のとおり定義されます。

```
CREATE OR REPLACE FUNCTION CONVERT_TO_ORDER_XML(input_order TYPE OE.ORDER_TYP)
RETURN OS.ORDER_XML_TYP AS
  xdata SYS.XMLType;
  new_order OS.ORDER_XML_TYP;
BEGIN
  xdata := XMLType.createXML(input_order, NULL);
```

```
  new_order := OS.ORDER_XML_TYP(input_order.orderno, xdata);
  RETURN new_order;
END CONVERT_TO_ORDER_XML;

execute dbms_transform.create_transformation(
   schema =>         'OS',
   name   =>         'OE2XML',
   from_schema =>    'OE',
   from_type =>      'ORDER_TYP',
   to_schema =>      'OS',
   to_type =>        'ORDER_XML_TYP',
   transformation => 'CONVERT_TO_ORDER_XML(source.user_data)');

/*  Add a rule-based subscriber for Overseas Shipping to the Booked orders queues
with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
 subscriber     aq$_agent;
BEGIN
 subscriber := aq$_agent('Overseas_Shipping','OS.OS_bookedorders_que',null);

 dbms_aqadm.add_subscriber(
        queue_name     => 'OE.OE_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.orderregion = ''INTERNATIONAL'''
        transformation => 'OS.OE2XML');
END;
```

アプリケーションが、カナダの顧客の注文を処理すると想定します。このアプリケーションは、次のプロシージャを使用してメッセージをデキューできます。

```
/*  Create procedures to enqueue into single-consumer queues: */
create or replace procedure get_canada_orders() as
deq_msgid               RAW(16);
dopt                    dbms_aq.dequeue_options_t;
mprop                   dbms_aq.message_properties_t;
deq_order_data          OS.order_xml_typ;
no_messages             exception;
pragma exception_init   (no_messages, -25228);
new_orders              BOOLEAN := TRUE;

begin
        dopt.wait := 1;

/* Specify dequeue condition to select Orders for Canada */
        dopt.deq_condition := 'tab.user_data.xdata.extract(
''/ORDER_TYP/CUSTOMER/COUNTRY/text()'').getStringVal()=''CANADA''';
```

```
    dopt.consumer_name : = 'Overseas_Shipping';

  WHILE (new_orders) LOOP
    BEGIN
      dbms_aq.dequeue(
          queue_name         => 'OS.OS_bookedorders_que',
          dequeue_options    => dopt,
          message_properties => mprop,
          payload            => deq_order_data,
          msgid              => deq_msgid);
      commit;

      dbms_output.put_line(' Order for Canada - Order No: ' ||
                           deq_order_data.orderno);

    EXCEPTION
      WHEN no_messages THEN
           dbms_output.put_line (' ---- NO MORE ORDERS  ---- ');
           new_orders := FALSE;
    END;
  END LOOP;
end;
```

参照：

- DBMS_AQADM または Java（JDBC）のいずれかを使用した構造化メッセージ・ペイロード・アプリケーションの実装方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』の第 8 章を参照してください。
- DBMS_TRANSFORM の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

FAQ: XML および AQ

多くの PDF ファイルを持つ AQ XML メッセージを 1 つのレコードとして格納する方法

質問

Oracle AQ を使用して、XML 文書をあるビジネス分野から別のビジネス分野へ変更する予定です。受信または送信される各メッセージには、XML ヘッダー、XML アタッチメント（XML データ・ストリーム）、DTD および PDF ファイルが含まれます。これらのすべての情報を、画像ファイルも含めてデータベース表（この場合はキュー・テーブル）に格納する必要があります。

このメッセージを、1 つのレコードまたは 1 つのピースとして Oracle キュー・テーブルにエンキューできますか？または、このメッセージを複数のレコード（XML データ・ストリームに対する CLOB 型としての 1 つのレコード、PDF ファイルに対する RAW としての 1 つのレコードなど）としてエンキューしてから、これらのレコードの相関を指定する必要がありますか？また、メッセージをデキューしたことを確認する必要もあります。

回答

次の方法で可能です。

- CLOB、RAW などの属性を持つオブジェクト型を定義し、これを単一メッセージとして格納します。
- AQ メッセージ・グループ化機能を使用して、該当するメッセージを複数のメッセージに格納します。ただし、メッセージのプロパティは 1 つのグループに対応付けられます。メッセージ・グループ化機能を使用するには、すべてのメッセージのペイロード型が同じである必要があります。

質問

最初にペイロード型を CLOB として指定し、すべてのピース、XML メッセージ・データ・ストリーム、DTD、PDF などを単一のメッセージ・ペイロードとしてエンキューしてキュー・テーブルに格納するのですか？その場合、このメッセージをデキューするときに、どのようにしてこの単一メッセージを個々に分割するのですか？

回答

そうではありません。まず、次のとおりオブジェクト型を作成します。

```
CREATE TYPE mypayload_type as OBJECT (xmlDataStream CLOB, dtd CLOB, pdf BLOB);
```

次に、このオブジェクト型を単一のメッセージとして格納します。

メッセージをエンキューした後の新しい受信者の追加

質問

メッセージ割当てをサポートするためにキュー・テーブルを使用します。たとえば、他のビジネス分野は、メッセージを Oracle に送信するときに、これらのメッセージを処理するために誰が割り当てられるかは認識していません。ただし、そのメッセージが人事部宛てであることは認識しています。そのため、すべてのメッセージは人事部管理者へ送信されます。

この時点で、メッセージはキュー・テーブルにエンキューされています。このメッセージの受信者は人事部管理者のみであり、その他のすべての人事部社員はこのキューのサブスクライバとして事前定義されています。人事部管理者は新しい受信者であるその他の社員を、キュー・テーブル内の既存のメッセージに関する message_properties.recipient_list に追加することができますか？

メッセージがエンキューされるときには複数のコンシューマ（受信者）は存在しませんが、メッセージがキュー・テーブルにエンキューされた後に、古い受信者を置き換えるか、または新しい受信者を追加する必要があります。この新しい受信者が、新しいメッセージをデキューします。このようなことは可能ですか？または、メッセージを古い受信者から削除してから、同じメッセージ内容を新しい受信者にエンキューする必要がありますか？

回答

メッセージがエンキューされた後に受信者のリストを変更することはできません。受信者のリストを指定しない場合、サブスクライバはそのキューをサブスクライブし、メッセージをデキューできます。

この場合、新しい受信者がそのキューのサブスクライバである必要があります。それ以外の場合は、メッセージをデキューし、そのメッセージを新しい受信者が再度エンキューする必要があります。

Oracle での XML メッセージのエンキュー、デキューおよび処理方法

質問

OTN ドキュメント「Using XML in Oracle Database Applications、Part 4、Exchanging Business Data Among Applications」（1999 年 11 月）には、Oracle データベースは、XML メッセージをエンキュー、デキューおよび処理できると記載されています。これは、どのように実行されますか？

XSU を使用して、XML ファイルを処理する前に表に挿入する必要がありますか？または、XML ファイルを直接エンキューし、解析してから AQ プロセスを介してそのメッセージをディスパッチできますか？XML データを Oracle データベースに挿入または更新するたびに、XSU を使用する必要がありますか？

回答

AQ は、オブジェクトのエンキューおよびデキューをサポートしています。これらのオブジェクトは、XML 文書を含む XMLType 型の属性、およびメッセージとともに送信する必要がある、抽出されたメタデータ属性を持つことができます。詳細および例については、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

XML コンテンツを持つメッセージを AQ キューから解析する方法

質問

XML コンテンツを持つメッセージを AQ キューから解析し、ODS（Operational Data Store）にある表およびフィールドを更新するツールが必要です。XML 文書を取り出して解析し、特定のフィールドをデータベース表および列にマップする必要があります。

Oracle Text（*inter*Media Text/Context）で可能ですか？

回答

最も簡単な方法は、Oracle 内で AQ と同時に、Oracle XML Parser for Java および Java ストアド・プロシージャを使用することです。

質問

カスタム・ソリューションを使用すると XSU を使用できます。主な目的はサプライチェーンです。ある特定の XML タグ値の問合せに基づいて、AQ エンキュー / デキューの回数や JMS ヘッダー情報などのメタデータ情報を取得する必要があります。単純に CLOB に XML を格納して、Oracle Text（*inter*Media Text）を使用して問合せを発行できますか？

回答

- XML を CLOB で格納する場合、Oracle Text を使用して検索できますが、この場合は、ある基準に一致する特定のメッセージのみを検索できます。
- メタデータ上で集計操作を行う必要がある場合は、既存の関連ツールからメタデータを参照するか、または通常の SQL 述語をメタデータに使用します。この場合、CLOB に XML で格納するのみでは十分ではありません。

Oracle Text（*inter*Media Text）XML 検索と、抽出された列などの重複したメタデータ記憶域を組み合せ、通常の SQL 述語と Oracle Text（*inter*Media Text）の CONTAINS() 句を組み合せる SQL 文を使用すると、両方の最適な機能を利用できます。

> **参照：** 第 8 章「Oracle Text を使用した XML データの検索」を参照してください。

XML 文書が処理されるまでのリスナー停止の回避

質問

クライアントから XML メッセージを受信したら、すぐに処理する必要があります。各 XML 文書の処理には、約 15 秒かかります。PL/SQL を使用しています。

1 つの PL/SQL プロシージャがリスナーを起動し、メッセージをデキューし、別のプロシージャをコールして XML 文書を処理します。問題は、XML 文書が処理されるまで、リスナーが停止することです。その間、メッセージがキューに蓄積します。

これを処理する最適な方法を教えてください。リスナー・プログラムが、XML 処理プロシージャを非同期にコールし、リスニングへ戻る方法はありますか？現時点では、Java は使用しません。

回答

メッセージを受信後、DBMS_JOB パッケージを使用してジョブを送信します。ジョブは、異なるデータベース・セッションで非同期に起動されます。

Oracle では、AQ 通知フレームワークに PL/SQL コールバックが追加されています。これによって、メッセージがキューに入れられると非同期に起動される PL/SQL コールバックを登録できます。

第 IV 部

Oracle ベースの XML アプリケーションを構築するためのツール製品およびフレームワーク

第 IV 部では、XSQL Servlet Pages を使用する方法を説明します。XSQL Servlet は、XDK for Java の一部です。

また、JDeveloper、BC4J、メタデータ API、Oracle Reports および Oracle Portal を使用して Oracle ベースの XML アプリケーションを構築する方法も説明します。さらに、Oracle Exchange および Oracle XML Gateway についても説明します。

第 IV 部に含まれる章は、次のとおりです。

10

XSQL ページ・パブリッシング・フレームワーク

この章の内容は次のとおりです。

- XSQL Servlet の例の説明
  - デモ・データの設定
- XSQL ページの高度なトピック
  - クライアントによるスタイルシートのオーバーライド・オプションの理解
  - スタイルシートの処理方法の制御
  - XSQLConfig.xml を使用した環境のチューニング
  - FOP シリアル化コードを使用した PDF 出力の生成
  - XSQL Page Processor のプログラム的な使用
  - カスタム XSQL アクション・ハンドラの作成
  - カスタム XSQL シリアル化コードの作成
  - カスタム XSQL Connection Manager の作成
  - XSQL アクション・ハンドラ・エラーのフォーマット
- XSQL Servlet の制限事項
- FAQ: XSQL Servlet

XSQL ページ・パブリッシング・フレームワークの概要

Oracle XSQL ページ・パブリッシング・フレームワークは、XML 情報を必要なフォーマットで簡単に公開できる拡張可能なプラットフォームです。このフレームワークは、SQL、XML および XSLT の機能を組み合せてデータベース情報に基づいて動的 Web コンテンツを公開する操作を非常に簡単にします。

XSQL パブリッシング・フレームワークを使用すると、SQL を十分に理解しているユーザーは、XSQL ページという宣言テンプレートを作成および使用して、次の操作を行うことができます。

- パラメータ化された SQL 問合せに基づいて、動的 XML データグラムを作成します。
- 対応付けられた XSLT 変換を使用してこれらのデータ・ページを変換し、必要に応じて、XML、HTML または他のテキストベースのフォーマットで最終結果を生成します。

公開する情報の作成および変換には、プログラミングは必要ありません。ユーザーが必要とするほぼすべての一般的な操作は、宣言方式で簡単に実行できます。ただし、XSQL パブリッシング・フレームワークは拡張可能であるため、いずれかの組込み機能がユーザーのニーズに合わない場合は、Java を使用してフレームワークを簡単に拡張し、カスタム情報ソースを統合したり、サーバー側でカスタム処理を実行することができます。

XSQL ページ・フレームワークを使用すると、公開する情報の作成工程と表示工程を確実に分離できます。この単純なアーキテクチャによって、生産性における様々なメリットが得られます。次の操作を行うことができます。

- 要求の発信元であるクライアント・デバイス（ブラウザ、携帯電話、PDA など）の種類に合わせて適切に表示を調整するなど、同じ情報を複数の方法で表示できます。
- 既存のページを新しいページに集計して、情報を簡単に再利用できます。
- 表示されている情報の内容を変更することなく、表示を修正および改善できます。

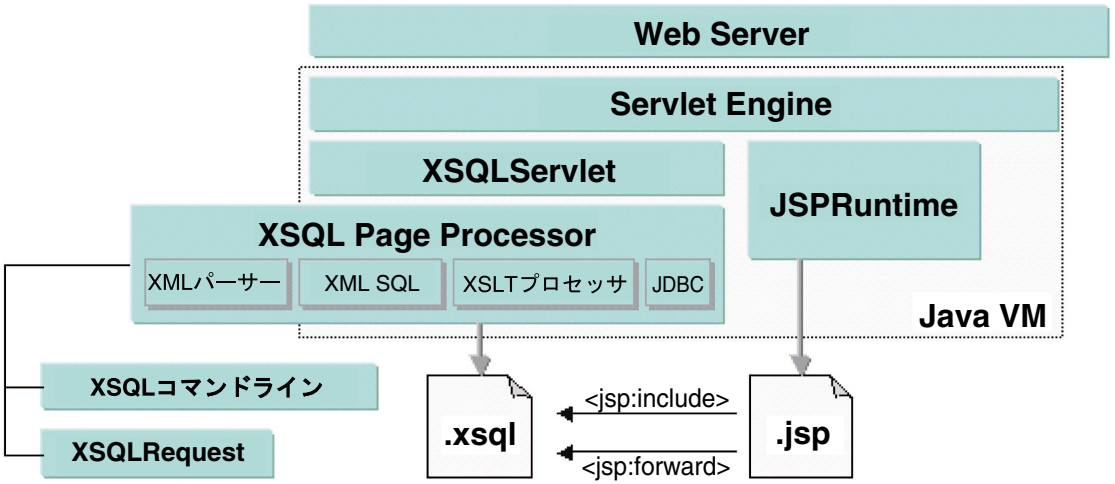
Oracle XSQL ページを使用して実行できる操作

サーバー側テンプレートは、その拡張子 `.xsql` から、「XSQL ページ」といいます。このテンプレートを使用すると、すべての情報を任意のフォーマットですべてのデバイスに公開できます。XSQL Page Processor Engine は、XSQL ページ・テンプレートのコンテンツを解析、キャッシュおよび処理します。図 10-1 に、コアである XSQL Page Processor Engine を次の 4 つの異なる方法で実行できることを示します。

- XSQL コマンドライン・ユーティリティを使用して、コマンドラインから、またはバッチで実行する方法
- 任意の Web サーバーにインストールした XSQL Servlet を使用して、Web 上で実行する方法

- `<jsp:include>` を使用してテンプレートを挿入し、JSP アプリケーションの一部として実行する方法
- `XSQLRequest` オブジェクト（XSQL Page Processor Engine の Java API）を使用して、プログラム的に実行する方法

図 10-1 XSQL ページ・フレームワークのアーキテクチャ



これらの使用例では、同じ XSQL ページ・テンプレートを使用できます。テンプレートの処理方法に関係なく、同じ基本手順によって結果が生成されます。XSQL Page Processor Engine は、次の操作を行います。

1. XSQL テンプレートの処理要求を受信します。
2. 1 つ以上の SQL 問合せの結果を使用して、XML データグラムを作成します。
3. この XML データグラムをリクエスタに戻します。
4. オプションで、データグラムを XML、HTML または他のテキスト・フォーマットに変換します。

この処理の変換手順中に、W3C の XSLT 1.0 標準に準拠するスタイルシートを使用して、作成されたデータグラムを次のようなドキュメント形式に変換できます。

- HTML - ブラウザ表示用
- WML - 無線デバイス用
- Scalable Vector Graphics（SVG）- データ駆動型のチャート、グラフおよびダイアグラム用
- eXtensible Stylesheet Language Formatting Object（XSLFO）- Adobe PDF 形式へのレンダリング用
- テキスト・ドキュメント（電子メール、SQL スクリプト、Java プログラムなど）
- 任意の XML ベースのドキュメント形式

XSQL ページを使用すると、基礎となる Oracle の XML コンポーネントの使用を自動化し、カスタム・プログラミングを行うことなく多くの一般的な問題を解決できます。ただし、カスタム・プログラミングのみが有効である場合（10-49 ページの「XSQL ページの高度なトピック」を参照）は、フレームワークの組込みアクションおよびシリアル化コードを追加して、任意のカスタム・ソースから XSQL データグラムを作成し、そのデータグラムを必要な形式にシリアル化できます。最初からパブリッシング・フレームワーク全体を作成する必要はありません。

参照：

- XSQL Servlet の仕様および早見表については、付録 C「XDK for Java: 仕様および早見表」を参照してください。
- OTN サイト http://otn.oracle.com/tech/xml の「XSQL Servlet Release Notes」も参照してください。

Oracle XSQL ページの取得方法

XSQL Servlet は、Oracle に付属しています。また、OTN サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

この章の例およびデモは、OTN にもあります。

XSQL ページの実行要件

Oracle XSQL ページ・パブリッシング・フレームワークをコマンドラインから実行するために必要なコンポーネントは、Java VM（1.1.8、1.2.2 または 1.3）のみです。XSQL ページ・フレームワークは、Oracle XDK に含まれている次の 2 つの基礎となるコンポーネントに依存し、これらのコンポーネントにバンドルされています。

- Oracle XML Parser および Oracle XSLT Processor（`xmlparserv2.jar`）
- Oracle XSU（`xsu12.jar`）

どちらの Java アーカイブ・ファイルも、XSQL ページ・フレームワークを実行している CLASSPATH にある必要があります。ほぼすべての XSQL ページはデータベースに接続して公開用の情報を問い合せるため、XSQL ページ・フレームワークは JDBC ドライバにも依存します。すべての JDBC ドライバがサポートされますが、Oracle に接続する場合は、Oracle JDBC ドライバ（`classes12.zip`）を使用すると、機能およびパフォーマンスが最大限に向上します。

また、XSQL パブリッシング・エンジンは `XSQLConfig.xml` という名前の自身の構成ファイルを Java リソースとして読み取るため、`XSQLConfig.xml` ファイルが格納されているディレクトリも CLASSPATH 内に含める必要があります。

XSQL ページ・フレームワークを Web パブリッシングに使用するには、前述のコンポーネントの他に、Java サーブレットをサポートする Web サーバーが必要です。次に、XSQL Servlet をテスト済のサーブレット機能を持つ Web サーバーを示します。

- Oracle9*i* Internet Application Server V1.x および V2.x
- Oracle9*i* Oracle Servlet Engine
- Allaire JRun 2.3.3 および 3.0.0
- Apache 1.3.9 以上（JServ 1.0/1.1 または Tomcat 3.1/3.2 Servlet Engine 付き）
- Apache Tomcat 3.1/3.2 Web Server + Servlet Engine
- Caucho Resin 1.1
- Java Web Server 2.0
- WebLogic Server 5.1
- NewAtlanta ServletExec 2.2/3.0 for IIS/PWS 4.0
- Oracle8*i* Lite Web-to-Go Server
- Sun JSWDK 1.0.1 Web Server

注意： セキュリティのため、本番 Web サーバーに XSQL Servlet をインストールする場合は、XSQLConfig.xml ファイルが Web サーバーの仮想ディレクトリ階層内のディレクトリに格納されていないことを確認してください。この予防措置を実行しなかった場合は、構成情報が Web 上で公開される危険があります。

インストール、環境の構成および XSQL Servlet の実行の詳細、およびその他の例とガイドラインについては、OTN サイト http://otn.oracle.com/tech/xml の「XSQL Servlet Release Notes」を参照してください。

XSQL ページの基本機能の概要

この項では、サーバー側の XSQL ページ・テンプレートで使用できる、次の最も基本的な機能について簡単に説明します。

- SQL 問合せからの XML データグラムの生成
- 別の XML 形式への XML データグラムの変換
- 表示用の HTML への XML データグラムの変換

SQL 問合せからの XML データグラムの生成

XSQL ページを使用して XML 形式のデータベース情報を Web 上で提供することは非常に簡単です。次に、今日 JFK 空港に到着するすべてのフライトのリアルタイムの XML データグラムを Oracle から簡単に提供する例を示します。Oracle JDeveloper（または任意のテキスト・エディタ）を使用して次のような XSQL ページ・テンプレートを作成し、それを AvailableFlightsToday.xsql という名前のファイルに保存します。

```
<?xml version="1.0"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
  ORDER BY ExpectedTime      /* to the value of the City parameter   */
</xsql:query>
```

XSQL Servlet が Web サーバー上に正しくインストールされている場合、前述の AvailableFlightsToday.xsql ファイルを Web サーバーの仮想ディレクトリ階層内のディレクトリにコピーし、Web ブラウザで次の URL を指定すると、保存したテンプレートにアクセスできます。

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=JFK
```

XSQL ページでの問合せの結果は、自動的に XML として生成され、リクエスタに戻されます。この XML ベースのデータグラムは、通常、他のサーバー・プログラムで処理するために要求されます。ただし、Internet Explorer 5.0 などのブラウザを使用する場合、図 10-2 に示すような XML での結果を直接表示できます。

図 10-2 XSQL ページ（AvailableFlightsToday.xsql）問合せの XML 結果

次に、使用した XSQL ページ・テンプレートの構造の詳細を説明します。XSQL ページが次の要素で始まることに注意してください。

```
<?xml version="1.0"?>
```

これは、XSQL テンプレート自体が、静的 XML コンテンツと XSQL アクション・ハンドラ要素の組合せを含む XML ファイル（拡張子は *.xsql）であるためです。前述の例の AvailableFlightsToday.xsql には、静的 XML 要素は含まれず、単一の XSQL アクション・ハンドラ要素 <xsql:query> のみが含まれています。これは、作成できる最も単純で有効な、単一の問合せを含む XSQL ページです。

ページ <xsql:query> 内の最初の（この場合は唯一の）要素に、名前空間の接頭辞 xsql を Oracle XSQL 名前空間識別子 urn:oracle-xsql のニックネームとして宣言する特殊な属性が含まれていることに注意してください。

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

この最初で最も外側にある要素（ドキュメント要素）にも、connection 属性が含まれています。この属性の値「demo」は、XSQLConfig.xml 構成ファイル内にある事前定義済接続の 1 つの名前です。

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

demo 接続に使用する、ユーザー名、パスワード、データベースおよび JDBC ドライバの詳細は、構成ファイルにすべて記載されます。これらの接続定義の設定については、この章の後半を参照してください。

また、<xsql:query> 要素には、bind-params 属性が含まれています。この属性は、要求内のパラメータの値を名前で対応付け、<xsql:query> タグに含まれる SQL 文内に疑問符で示されているパラメータをバインドします。

ページに複数の問合せを挿入する場合は、独自の XML 要素を作成して、次のとおり他の要素をラップする必要があります。

```
<?xml version="1.0"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="City">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
      ORDER BY ExpectedTime  /* to the value of the City parameter   */
  </xsql:query>
  <!-- Other xsql:query actions can go here inside <page> and </page> -->
</page>
```

この例では、connection 属性および xsql 名前空間宣言は常にドキュメント要素に指定されますが、bind-params は <xsql:query> アクションに固有であることに注意してください。

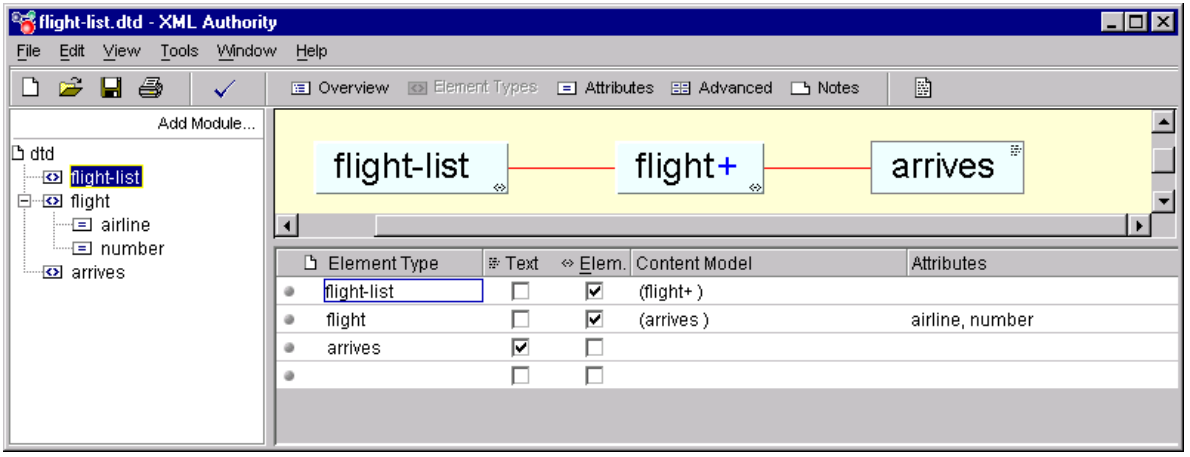
別の XML 形式への XML データグラムの変換

図 10-2 に示す正規の <ROWSET> および <ROW> XML 出力が必要な XML 形式ではない場合は、XSLT スタイルシートを任意の XSQL ページ・テンプレートと対応付けてこの XML データグラムをサーバー内で変換し、必要な別の形式で情報を戻すことができます。

他のプログラムとデータを交換する場合は、通常、交換する XML 形式を記述する特定の DTD を、交換先と決めておきます。DTD は、実際には、スキーマ定義です。DTD は、そのタイプのドキュメントが含むことができる XML の要素および属性を正式に定義します。

flight-list.dtd の定義が指定され、その DTD に準拠した形式で到着フライトのリストを生成する必要があるとします。Extensibility 社製の XML Authority などのビジュアル・ツールを使用して、図 10-3 に示すとおり flight-list.dtd の構造をブラウズできます。

図 10-3 Extensibility 社製の XML Authority を使用した業界標準の flight-list.dtd の調査



この図は、フライト・リスト用の標準の XML 形式が、次の要素を含むことを示します。

- <flight-list> 要素。
- 1 つ以上の <flight> 要素。airline 属性および number 属性を持ち、<flight-list> 要素に含まれます。
- <arrives> 要素。<flight> 要素に含まれます。

次の XSLT スタイルシート（flight-list.xsl）を XSQL ページと対応付けると、到着フライトのデフォルト形式である <ROWSET> および <ROW> を、業界標準の DTD 形式に変換できます。

```
<!-- XSLT Stylesheet to transform ROWSET/ROW results into flight-list format -->
<flight-list xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             xsl:version="1.0">
  <xsl:for-each select="ROWSET/ROW">
      <flight airline="{CARRIER}" number="{FLIGHTNUMBER}">
        <arrives><xsl:value-of select="DUE"/></arrives>
      </flight>
  </xsl:for-each>
</flight-list>
```

このスタイルシートは、結果として生成されたドキュメントで生成される必要がある <flight-list>、<flight>、<arrives> などのリテラル要素を含むテンプレートです。これらのリテラル要素は、次の操作を可能にする特殊な XSLT アクション・ハンドラ要素とともにテンプレート内に配置されます。

- <xsl:for-each> を使用して、ソース・ドキュメント内の一致する要素に対するループを行います。
- <xsl:value-of> を使用して、必要に応じてソース・ドキュメント要素の値をプラグインします。
- {something} を使用して、ソース・ドキュメント要素の値を属性値にプラグインします。

このスタイルシートでは、次の 2 つの属性が最上位の <flight-list> 要素に追加されていることに注意してください。

- xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

  これは、「xsl」という名前の XML 名前空間（xmlns）を定義し、XSLT 仕様を一意に識別する URL 文字列を識別します。これは、URL のように見えますが、文字列 http://www.w3.org/1999/XSL/Transform を、XSLT 1.0 仕様で定義されている要素セット用のグローバル主キーとして考えます。名前空間を定義すると、スタイルシートで <xsl:XXX> アクション・ハンドラ要素を使用し、必要に応じて値をループおよびプラグインできます。

- xsl:version="1.0"

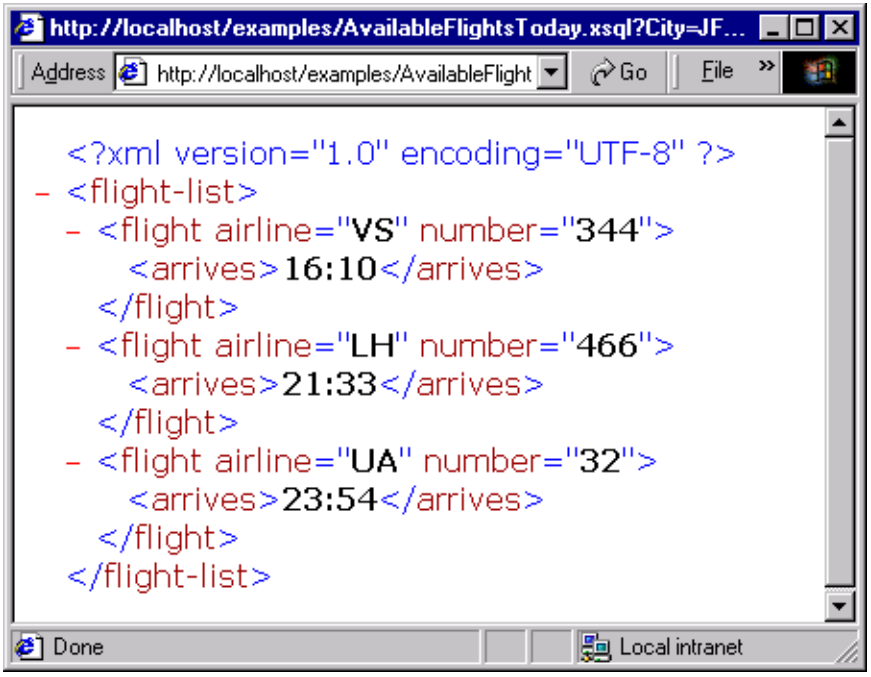
  この属性は、ドキュメントを XSLT 1.0 スタイルシートとして識別します。version 属性は、すべての XSLT スタイルシートが、妥当であると識別され、XSLT プロセッサによって認識されるために必要です。

次に示すとおり、<?xml-stylesheet?> 処理命令をページの上部に追加して、スタイルシートを XSQL ページに対応付けます。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="flight-list.xsl"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* The ? is a bind variable being bound */
      ORDER BY ExpectedTime  /* to the value of the City parameter   */
</xsql:query>
```

これは、スタイルシートを XML 文書と対応付ける W3C 標準のメカニズム（http://www.w3.org/TR/xml-stylesheet を参照）です。対応付けられた XSLT スタイルシートを XSQL ページに指定すると、図 10-4 に示すとおり、要求側プログラムまたはブラウザによって、提供された flight-list.dtd で指定された業界標準の形式で XML を参照できます。

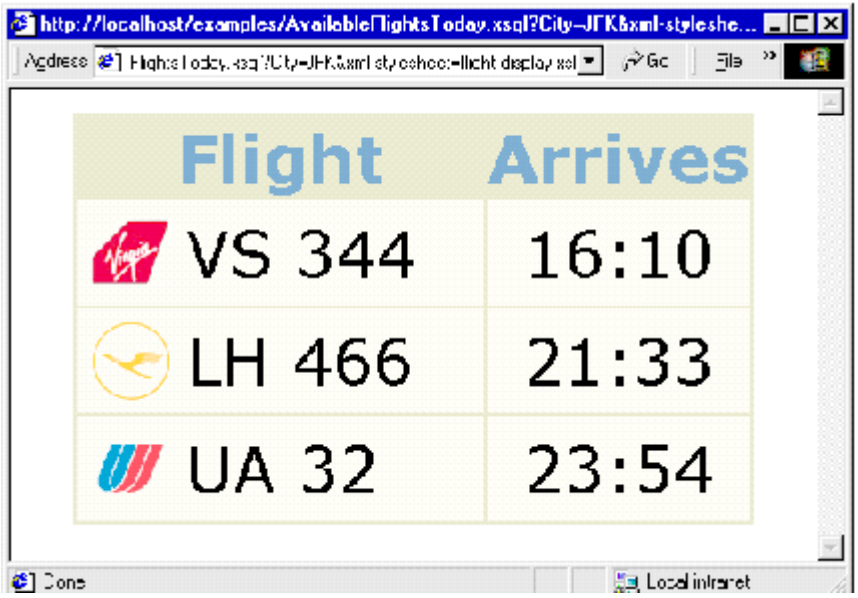
図 10-4 業界標準の XML 形式の XSQL ページ



表示用の HTML への XML データグラムの変換

同じ XML 情報を別の XML 形式ではなく HTML で戻すには、別の XSLT スタイルシートを使用します。スタイルシートで、<flight-list>、<flight> などの要素を生成するのではなく、<table>、<tr>、<td> などの HTML 要素を生成します。動的に問い合された情報の結果は、図 10-5 に示す HTML ページの表示に類似しています。XSQL ページは、未加工の XML 情報を戻すのではなく、サーバー側での XSLT による変換を使用して、ブラウザに送信するために情報を HTML としてフォーマットします。

図 10-5 対応付けられた XSLT スタイルシートを使用した HTML のレンダリング



flight-display.xsl スタイルシートは、flight-list.xsl スタイルシートの構文と同様に、テンプレートの HTML ページの表示に類似しています。これには、<xsl:for-each>、<xsl:value-of>、および <ROWSET> と <ROW> で構成された基礎となる XML 問合せ結果の動的な値をプラグインするための {DUE} などの属性値テンプレートが含まれます。

```
<!-- XSLT Stylesheet to transform ROWSET/ROW results into HTML -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><link rel="stylesheet" type="text/css" href="flights.css" /></head>
  <body>
```

```
    <center><table border="0">
      <tr><th>Flight</th><th>Arrives</th></tr>
      <xsl:for-each select="ROWSET/ROW">
        <tr>
          <td>
            <table border="0" cellspacing="0" cellpadding="4">
              <tr>
                <td><img align="absmiddle" src="images/{CARRIER}.gif"/></td>
                <td width="180">
                  <xsl:value-of select="CARRIER"/>
                  <xsl:text> </xsl:text>
                  <xsl:value-of select="FLIGHTNUMBER"/>
                </td>
              </tr>
            </table>
          </td>
          <td align="center"><xsl:value-of select="DUE"/></td>
        </tr>
      </xsl:for-each>
    </table></center>
  </body>
</html>
```

> **注意：** このスタイルシートは、HTML と同じに見えますが、相違点が 1 つあります。このスタイルシートは、整形式の HTML です。これは、それぞれの開始タグが適切にクローズされ（<td>...</td> など）、空のタグに、
 ではなく空の XML 要素構文
 が使用されることを意味します。

次の機能を組み合せて、前述のことを確認できます。

- Oracle データベースから必要な情報を選択するためのパラメータ化された SQL 文
- 移植可能な中間データ交換形式としての業界標準の XML
- XML ベースのデータ・ページを、必要に応じて任意の XML ベースまたは HTML ベースの形式に変換するための XSLT

これによって、非常に興味深く、有効な結果を迅速に実現できます。前述の操作は、XSQL ページを使用して実行できる操作の一部です。この章の後半で、さらに多くの操作について説明します。

> **注意：** XSLT、および様々な Oracle データベースの使用例に XSLT を適用する方法の詳細は、『Building Oracle XML Applications』（Steve Muench 著、O'Reilly 出版）を参照してください。

様々な環境での XSQL ページの設定および使用

XSQL ページは、様々な方法で開発および使用できます。最初に Oracle JDeveloper を使用した最も簡単な使用方法を説明し、次に本番環境で XSQL ページを使用するために理解しておく必要がある詳細を説明します。

Oracle JDeveloper による XSQL ページの使用

開発中に XSQL ページを処理する最も簡単な方法は、Oracle JDeveloper を使用することです。JDeveloper IDE のバージョン 3.1 以上は、カラー化された構文ハイライト表示、XML の構文検査および XSQL ページの簡単なテストをサポートします。また、JDeveloper 3.2 では XSQL ページのデバッグをサポートし、XSQL アクションの作成に有効な新しいウィザードが追加されています。

JDeveloper プロジェクトで XSQL ページを作成するには、次の操作を実行します。

- ナビゲータの上部にある「+」アイコンをクリックして、新規または既存の XSQL ページをプロジェクトに追加します。
- 「File」>「New...」を選択し、ギャラリーの「Web Objects」タブから「XSQL」を選択します。

`<xsql:query>` などの XSQL アクション・ハンドラ要素を XSQL ページに簡単に追加するには、新しい要素を挿入する位置にカーソルを置き、次のどちらかの操作を実行します。

- 右クリックして表示されるメニューから「XSQL Element...」を選択します。
- 「IDE」メニューから「Wizards」>「XSQL Element...」を選択します。

XSQL エレメント・ウィザードが示す手順に従って、使用する XSQL アクションおよび必要な属性を選択します。

XSQL ページ・テンプレートの構文検査を行うには、検査する XSQL ページの名前を選択した後で、ナビゲータ内で右クリックして表示されるメニューから「Check XML Syntax...」を選択します。XML に構文エラーがある場合は、メッセージ・ビューにエラーが表示され、カーソルが最初のエラーの位置に移動します。

XSQL ページをテストするには、ナビゲータ内でページを選択し、右クリックして表示されるメニューから「Run」を選択します。JDeveloper は、XSQL ページを実行できるように適切に構成されたローカルの Oracle Web-to-Go Server を自動的に起動し、ページを要求する適切な URL を指定したデフォルトのブラウザを起動してそのページをテストします。一度 XSQL ページを実行すると、IDE でそのページ（およびそのページが対応付けられているすべての XSLT スタイルシート）に変更を加えたり、そのファイルを IDE に保存した後で、すぐにブラウザで再表示して反映された変更を確認することができます。

JDeveloper を使用する場合は、CLASSPATH が適切に設定されるように、XSQL Runtime というライブラリをプロジェクトのライブラリ・リストに追加する必要があります。IDE は、ユーザーが New Object Gallery を使用して新しい XSQL ページを作成するときに自動的にこのエントリを追加します。このエントリは、「Project」>「Project Properties...」を選択して「Libraries」タブをクリックすることで、手動でプロジェクトに追加することもできます。

本番環境での CLASSPATH の適切な設定

JDeveloper 以外の環境では、XSQL Page Processor Engine が実行できるように適切に構成されていることを確認する必要があります。Oracle には、データベースに付属する Oracle HTTP Server に事前インストール済の XSQL Servlet が含まれていますが、その他の環境で XSQL を使用する場合は、Java CLASSPATH が適切に設定されていることを確認する必要があります。

XSQL Page Processor には、次の 3 つのエントリ・ポイントがあります。

- `oracle.xml.xsql.XSQLServlet`（サーブレット・インタフェース）
- `oracle.xml.xsql.XSQLCommandLine`（コマンドライン・インタフェース）
- `oracle.xml.xsql.XSQLRequest`（プログラム・インタフェース）

これらの 3 つのインタフェースおよびコアである XSQL Page Processor Engine 自体は、すべて Java で作成されているため、移植可能で、非常に簡単に設定できます。唯一の設定要件は、実行して XSQL ページを処理する JavaVM の CLASSPATH に、適切な jar ファイルが含まれていることです。jar ファイルには、次のものが含まれます。

- `oraclexsql.jar`（XSQL Page Processor）
- `xmlparserv2.jar`（Oracle XML Parser for Java）
- `xsu12.jar`（Oracle XSU）
- `classes12.zip`（Oracle JDBC ドライバ）

また、XSQL Page Processor の構成ファイル（`XSQLConfig.xml`）が格納されているディレクトリも、CLASSPATH 内のディレクトリとして示される必要があります。

Windows NT 上で配布された XSQL を `C:¥xsql` にインストールした場合、CLASSPATH は次のとおり表示されます。

```
C:¥xsql¥lib¥classes12.zip;C:¥xsql¥lib¥xmlparserv2.jar;
C:¥xsql¥lib¥xsu12.jar;C:¥xsql¥lib¥oraclexsql.jar;
directory_where_XSQLConfig.xml_resides
```

UNIX 上で配布された XSQL を /web ディレクトリに抽出した場合、CLASSPATH は次のとおり表示されます。

```
/web/xsql/lib/classes12.zip:/web/xsql/lib/xmlparserv2.jar:
/web/xsql/lib/xsu12.jar:/web/xsql/lib/oraclexsql.jar:
directory_where_XSQLConfig.xml_resides
```

XSQL Servlet を使用するには、.xsql ファイルの拡張子を XSQL Servlet の Java クラスである oracle.xml.xsql.XSQLServlet に対応付ける手順が必要になります。Web サーバーの Servlet 環境の CLASSPATH を設定する方法、および Servlet をファイル拡張子に対応付ける方法は、Web サーバーによって異なります。Oracle XSQL ページで使用する Web サーバー固有の設定情報については、Oracle XSQL Servlet のリリース・ノートを参照してください。

接続定義の設定

XSQL ページは、XSQL 構成ファイルに定義されている接続用のニックネームを使用して、データベース接続を参照します。接続名は、次のような XSQLConfig.xml ファイルの <connectiondefs> セクションに定義されています。

```
<connectiondefs>
  <connection name="demo">
    <username>scott</username>
    <password>tiger</password>
    <dburl>jdbc:oracle:thin:@localhost:1521:testDB</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <autocommit>true</autocommit>
  </connection>
  <connection name="lite">
    <username>system</username>
    <password>manager</password>
    <dburl>jdbc:Polite:POlite</dburl>
    <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
   </connection>
</connectiondefs>
```

各接続に対して、次の 5 つの情報を指定できます。

1. <username>
2. <password>
3. <dburl>（JDBC 接続文字列）
4. <driver>（使用する JDBC ドライバの完全修飾クラス名）
5. <autocommit>（オプションで、自動コミットを強制的にオンまたはオフにする要素）

<autocommit> 要素が指定されていない場合、XSQL Page Processor は JDBC ドライバの AutoCommit フラグのデフォルト設定を使用します。

任意の数の <connection> 要素をこのファイルで使用し、必要な接続を定義できます。個々の XSQL ページは、ページ内の最上位の要素（ドキュメント要素）に connection="*xxx*" 属性を挿入して、使用する接続を参照します。

注意： セキュリティのため、本番 Web サーバー上に XSQL Servlet をインストールする場合は、XSQLConfig.xml ファイルが Web サーバーの仮想ディレクトリ階層内のディレクトリに格納されていないことを確認してください。この予防措置を実行しなかった場合は、構成情報が Web 上で公開される危険があります。

XSQL コマンドライン・ユーティリティの使用

動的なページのコンテンツは、環境で頻繁に変更されないデータに基づいていることがほとんどです。Web パブリッシングのパフォーマンスを最適化するには、オペレーティング・システムの機能を使用して XSQL ページのオフライン処理をスケジューリングします。これによって、処理結果が Web サーバーで静的に提供されたままになります。

XSQL コマンドライン・ユーティリティを使用して、コマンドラインから任意の XSQL ページを処理できます。構文は次のとおりです。

```
$ java oracle.xml.xsql.XSQLCommandLine xsqlpage [outfile] [param1=value1 ...]
```

outfile を指定した場合、*xsqlpage* の処理結果がその出力ファイルに書き込まれます。*outfile* を指定しない場合、結果が標準出力されます。任意の数のパラメータを XSQL Page Processor に渡すことができます。また、要求の一部として処理中の XSQL ページはそれらのパラメータを参照できます。ただし、次のパラメータ名はコマンドライン・ユーティリティによって認識され、事前定義済の動作を行います。

- xml-stylesheet=*stylesheetURL*

  これは、要求に使用するスタイルシートの相対 URL または絶対 URL を指定します。また、文字列 none を指定すると、デバッグ操作のために XSLT スタイルシートの処理を抑制することもできます。

- posted-xml=*XMLDocumentURL*

  これは、要求の一部としてポストされたときと同様に処理する XML リソースの相対 URL または絶対 URL を指定します。

- useragent=*UserAgentString*

  これは、そのユーザー・エージェント・タイプに適切なスタイルシートがページのコマンドライン処理の一部として選択されるように、コマンドラインから特定の HTTP のユーザー・エージェント文字列をシミュレーションするために使用されます。

?/xdk/java/xsql/bin ディレクトリには、XSQL コマンドライン・ユーティリティのコールを自動化するための、プラットフォーム固有のコマンド・スクリプトが含まれています。このスクリプトは、oracle.xml.xsql.XSQLCommandLine クラスを実行するように Java Runtime を設定します。

XSQL ページのすべての機能の概要

前述の項では、単一の XSQL アクション・ハンドラ要素である <xsql:query> アクションについてのみ説明しました。これは最も一般的なアクションではありますが、XSQL ページ・フレームワークには他のアクションも組み込まれています。次の項では、XSQL ページで使用できるすべての機能について説明します。

コア組込みアクションの使用

この項では、コア組込みアクションのリストを示し、各アクションが実行する操作について簡単に説明します。また、各アクションがサポートするすべての必須およびオプションの属性も示します。

<xsql:query> アクション

<xsql:query> アクション・ハンドラ要素は、SQL の SELECT 文を実行し、問合せの結果セットの正規の XML 表示をデータ・ページに挿入します。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

このアクションの構文は、次のとおりです。

```
<xsql:query>
   SELECT Statement
</xsql:query>
```

すべての有効な SQL の SELECT 文を使用できます。使用した SELECT 文によって行が生成されない場合、次のようなネストした <xsql:no-rows-query> 要素を含めることによって、代替の問合せを指定できます。

```
<xsql:query>
  SELECT Statement
  <xsql:no-rows-query>
    SELECT Statement to use if outer query returns no rows
  </xsql:no-rows-query>
</xsql:query>
```

<xsql:no-rows-query> 要素自体が、任意のネスト・レベルまでネストした <xsql:no-rows-query> 要素を含むことができます。<xsql:no-rows-query> に対して使用可能なオプションは、<xsql:query> アクション・ハンドラ要素の場合と同じです。

デフォルトでは、問合せによって生成された XML はその結果セットの列構造を反映し、要素名は列名と一致します。次のようなネストした構造を持つ、結果内の列は、その構造を反映するネストした要素を生成します。

- オブジェクト型
- コレクション型
- CURSOR 式

異なる型の列を含み、1 行を戻す典型的な問合せの結果は、次のようになる場合があります。

```
<ROWSET>
  <ROW id="1">
    <VARCHARCOL>Value</VARCHARCOL>
    <NUMBERCOL>12345</NUMBERCOL>
    <DATECOL>12/10/2001 10:13:22</DATECOL>
    <OBJECTCOL>
       <ATTR1>Value</ATTR1>
       <ATTR2>Value</ATTR2>
    </OBJECTCOL>
    <COLLECTIONCOL>
       <COLLECTIONCOL_ITEM>
         <ATTR1>Value</ATTR1>
         <ATTR2>Value</ATTR2>
       </COLLECTIONCOL_ITEM>
       <COLLECTIONCOL_ITEM>
         <ATTR1>Value</ATTR1>
         <ATTR2>Value</ATTR2>
       </COLLECTIONCOL_ITEM>
    </COLLECTIONCOL>
    <CURSORCOL>
      <CURSORCOL_ROW>
        <COL1>Value1</COL1>
        <COL2>Value2</COL2>
      </CURSORCOR_ROW>
    </CURSORCOL>
  </ROW>
</ROWSET>
```

<ROW> 要素は、結果セット内の行ごとに繰り返されます。問合せでは、標準 SQL 列別名を使用して結果セット内の列の名前を変更できます。これによって、生成された XML 要素の名前も効果的に変更できます。このような列別名は、XML 要素には無効な名前を持つ列に対しては必須です。

たとえば、次のような <xsql:query> アクションでは、計算された式のデフォルト列名が無効な XML 要素名であるため、エラーが発生します。

```
<xsql:query>SELECT TO_CHAR(hiredate,'DD-MON') FROM EMP</xsql:query>
```

この問題は、次のような列別名を使用して解決できます。

```
<xsql:query>SELECT TO_CHAR(hiredate,'DD-MON') as hiredate FROM EMP</xsql:query>
```

表 10-1 に示すオプションの属性を指定すると、<xsql:query> アクションによって取り出されるデータおよび生成される XML を様々な面で制御できます。

表 10-1 <xsql:query> の属性

属性名	説明
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
date-format = "*string*"	問合せ中の XML 内の書式化された日付列 / 属性の値に対して使用する日付書式マスク。有効値は、java.text.SimpleDateFormat クラスの有効値です。
error-statement = "*boolean*"	no に設定すると、生成されたすべての <xsql-error> 要素への違反 SQL 文の挿入が抑制されます。有効値は、yes および no です。デフォルト値は yes です。
fetch-size = "*integer*"	データベース・ラウンドトリップごとにフェッチするレコードの数。指定しない場合は、XSQLConfig.xml 内の /XSQLConfig/processor/default-fetch-size 構成設定で指定されているデフォルト値が使用されます。
id-attribute = "*string*"	結果セット内の各行を一意に識別するためのデフォルトの num 属性のかわりに使用する XML 属性名。この属性の値が空の文字列である場合は、行の ID 属性が抑制されます。
id-attribute-column = "*string*"	結果セット内の列の大文字 / 小文字を区別した名前。この値は、行 ID の属性値として各行に指定する必要があります。デフォルトでは、行カウントが行 ID の属性値として使用されます。

表 10-1 <xsql:query> の属性（続き）

属性名	説明
include-schema = "*boolean*"	yes に設定すると、結果セットの構造を記述するインライン XML Schema が挿入されます。有効値は、yes および no です。デフォルト値は no です。
max-rows = "*integer*"	フェッチする行の最大数。オプションで、skip-rows 属性が示す行数をスキップした後にフェッチする行の最大数を指定することもできます。指定しない場合は、デフォルトですべての行がフェッチされます。
null-indicator = "*boolean*"	NULL="Y" 属性が列の要素に挿入され、列の値が NULL であることを通知するかどうかを指定します。デフォルトでは、値が NULL である列は出力されません。有効値は、yes および no です。デフォルト値は no です。
row-element = "*string*"	問合せ結果の行セット全体に対するデフォルトの <ROW> 要素名のかわりに使用する XML 要素名。結果セット内の各行に対して含まれる <ROW> 要素の生成を抑制するには、空の文字列を設定します。
rowset-element = "*string*"	問合せ結果の行セット全体に対するデフォルトの <ROWSET> 要素名のかわりに使用する XML 要素名。含まれる <ROWSET> 要素の生成を抑制するには、空の文字列を設定します。
skip-rows = "*integer*"	結果セットから行をフェッチする前にスキップする行の数。max-rows と組み合せて、問合せ結果のステートレス・ページングを行うこともできます。
tag-case = "*string*"	有効値は、lower および upper です。指定しない場合は、デフォルトで、問合せで対応する XML 要素名として指定されている列名の大文字 / 小文字が使用されます。

<xsql:dml> アクション

<xsql:dml> アクションを使用して、すべての DML または DDL 操作およびすべての PL/SQL ブロックを実行できます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

このアクションの構文は、次のとおりです。

```
<xsql:dml>
  DML Statement or DDL Statement or PL/SQL Block
</xsql:dml>
```

表 10-2 に、<xsql:dml> アクションに対して使用できるオプションの属性を示します。

表 10-2 <xsql:dml> の属性

属性名	説明
commit = "*boolean*"	yes に設定すると、DML 文が正常に実行された後に、現行の接続に対するコミットがコールされます。有効値は、yes および no です。デフォルト値は no です。
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
error-statement = "*boolean*"	no に設定すると、生成されたすべての <xsql-error> 要素への違反 SQL 文の挿入が抑制されます。有効値は、yes および no です。デフォルト値は yes です。

<xsql:ref-cursor-function> アクション

<xsql:ref-cursor-function> アクションを使用すると、PL/SQL ストアド・ファンクションを実行して結果セットが決定された問合せによって生成される XML 結果を挿入することができます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

PL/SQL の動的 SQL 機能を使用すると、結果セットへのカーソル・ハンドルを XSQL Page Processor に戻す前に、問合せをファンクションで動的にまたは条件付きで（あるいはその両方で）作成できます。このアクションの名前が示すとおり、コールされたファンクションの戻り値は REF CURSOR 型である必要があります。

このアクションの構文は、次のとおりです。

```
<xsql:ref-cursor-function>
  [SCHEMA.][PACKAGE.]FUNCTION_NAME(args);
</xsql:ref-cursor-function>
```

<xsql:ref-cursor-function> アクションに使用できるオプションの属性は、fetch-size 属性を除き、表 10-1 に示す <xsql:query> アクションの属性と同じです。

たとえば、次の PL/SQL パッケージについて考えてみます。

```
CREATE OR REPLACE PACKAGE DynCursor IS
  TYPE ref_cursor IS REF CURSOR;
  FUNCTION DynamicQuery(id NUMBER) RETURN ref_cursor;
END;
CREATE OR REPLACE PACKAGE BODY DynCursor IS
  FUNCTION DynamicQuery(id NUMBER) RETURN ref_cursor IS
    the_cursor ref_cursor;
  BEGIN
   -- Conditionally return a dynamic query as a REF CURSOR
   IF id = 1 THEN
     OPEN the_cursor
      FOR 'SELECT empno, ename FROM EMP'; -- An EMP Query
   ELSE
     OPEN the_cursor
      FOR 'SELECT dname, deptno FROM DEPT'; -- A DEPT Query
   END IF;
   RETURN the_cursor;
  END;
END;
```

<xsql:ref-cursor-function> は、次の構文によって、このファンクションが戻した REF CURSOR の動的結果を挿入できます。

```
<xsql:ref-cursor-function>
  DynCursor.DynamicQuery(1);
</xsql:ref-cursor-function>
```

<xsql:include-owa> アクション

<xsql:include-owa> アクションを使用すると、データベース・ストアド・プロシージャによって生成された XML コンテンツを挿入できます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

ストアド・プロシージャは、標準の Oracle Web Agent（OWA）パッケージ（HTP および HTF）を使用して、XML タグをサーバー側ページ・バッファに出力します。その後、XSQL Page Processor が、動的に生成された XML コンテンツをフェッチおよび解析し、データ・ページに挿入します。ストアド・プロシージャは、整形式の XML ページを生成する必要があります。生成しなかった場合は、該当するエラーが表示されます。

このアクションの構文は、次のとおりです。

```
<xsql:include-owa>
   PL/SQL Block invoking a procedure that uses the HTP and/or HTF packages
</xsql:include-owa>
```

表 10-3 に、このアクションがサポートするオプションの属性を示します。

表 10-3 <xsql:include-owa> の属性

属性名	説明
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
error-statement = "*boolean*"	no に設定すると、生成されたすべての <xsql-error> 要素への違反 SQL 文の挿入が抑制されます。有効値は、yes および no です。デフォルト値は yes です。

バインド変数の使用

SQL バインド変数を使用すると、前述のアクションの結果をパラメータ化することができます。これによって、XSQL ページ・テンプレートは、要求で渡されたパラメータの値に基づいて異なる結果を生成できます。バインド変数を使用するには、SQL によってバインド変数が許可されている文内の任意の場所に疑問符を挿入します。たとえば、<xsql:query> アクションに次の SELECT 文が含まれる場合があります。

```
SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
  FROM latest_stocks s, customer_portfolio p
 WHERE p.customer_id = ?
   AND s.ticker = p.ticker
```

疑問符を使用して顧客 ID のバインド変数を作成します。このアクション・ハンドラ要素の bind-params 属性を指定すると、ページ内の SQL 文が実行されるたびに、パラメータ値がバインド変数にバインドされます。前述の例を使用すると、示されたバインド変数を次のようなページ要求で custid パラメータ値にバインドする XSQL ページを作成できます。

```
<!-- CustomerPortfolio.xsql -->
<portfolio connnection="prod" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="custid">
    SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
      FROM latest_stocks s, customer_portfolio p
     WHERE p.customer_id = ?
       AND s.ticker = p.ticker
  </xsql:query>
</portfolio>
```

次のような要求で顧客 ID パラメータを渡すと、特定の顧客のポートフォリオに対する XML データを要求できます。

```
http://yourserver.com/fin/CustomerPortfolio.xsql?custid=1001
```

bind-params 属性の値は、スペースで区切られたパラメータ名のリストです。これらのパラメータ名の左から右への順序は、その値がバインドされるバインド変数の文内での位置を示します。そのため、SQL 文内に 5 つの疑問符がある場合、bind-params 属性にはスペースで区切られた 5 つのパラメータ名のリストを指定する必要があります。疑問符で示された 1 つのバインド変数が複数回出現し、それに同一のパラメータ値をバインドする必要がある場合は、そのパラメータ名を bind-params 属性値の適切な位置に繰り返します。問合せ内にある疑問符と同数のパラメータ名が bind-params 属性に挿入されていない場合は、ページの実行時にエラーが発生します。

バインド変数は、SQL 文を必要とするすべてのアクションで使用できます。次のページは、その他の例を示します。

```
<!-- CustomerPortfolio.xsql -->
<portfolio connnection="prod" xmlns:xsql="urn:oracle-xsql">
  <xsql:dml commit="yes" bind-params="useridCookie">
     BEGIN log_user_hit(?); END;
  </xsql:dml>
  <current-prices>
    <xsql:query bind-params="custid">
      SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
        FROM latest_stocks s, customer_portfolio p
       WHERE p.customer_id = ?
         AND s.ticker = p.ticker
    </xsql:query>
  </current-prices>
  <analysis>
    <xsql:include-owa bind-params="custid userCookie">
      BEGIN portfolio_analysis.historical_data(?,5 /* years */, ?); END;
    </xsql:include-owa>
  </analysis>
</portfolio>
```

字句置換パラメータの使用

すべての XSQL アクション・ハンドラ要素で、任意の属性の値または含まれる SQL 文のテキストを字句置換パラメータに置き換えることができます。これによって、アクションの動作をパラメータ化したり、アクションが実行する SQL 文の一部を置き換えることができます。字句置換パラメータは、構文 {@*ParameterName*} を使用して参照されます。

次に、2 つの字句置換パラメータを使用した例を示します。1 つのパラメータはパラメータとして渡すことができる行の最大数を指定し、もう 1 つのパラメータは ORDER BY に対する列のリストを制御します。

```
<!-- DevOpenBugs.xsql -->
<open-bugs connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
    SELECT bugno, abstract, status
      FROM bug_table
     WHERE programmer_assigned = UPPER(?)
       AND product_id          = ?
       AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>
```

この例では、次の URL を要求すると、指定した開発者のオープン・エラー・リストの XML が表示されます。

```
http://yourserver.com/bug/DevOpenBugs.xsql?dev=smuench&prod=817
```

または、XSQL コマンドライン・ユーティリティを使用して、次のとおり要求します。

```
$ xsql DevOpenBugs.xsql dev=smuench prod=817
```

また、字句パラメータは XSQL ページ接続をパラメータ化したり、次のとおり、ページを処理するために使用されるスタイルシートのパラメータ化にも使用できます。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<!-- DevOpenBugs.xsql -->
<open-bugs connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
    SELECT bugno, abstract, status
      FROM bug_table
     WHERE programmer_assigned = UPPER(?)
       AND product_id          = ?
       AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>
```

バインド変数およびパラメータのデフォルト値の指定

多くの場合、バインド変数または置換パラメータのデフォルト値はページで直接指定すると有効です。これによって、リクエスタが各要求のすべての値を明示的に渡すことなく、ページをパラメータ化することができます。

パラメータのデフォルト値を挿入するには、そのパラメータと同じ名前の XML 属性をアクション・ハンドラ要素または任意の祖先クラス要素に追加します。任意のパラメータの値が要求内に挿入されていない場合、XSQL Page Processor は現行のアクション・ハンドラ要素で同じ名前の属性を検索します。検索されなかった場合は、ページのドキュメント要素に到達するまで、現行のアクション・ハンドラ要素の各祖先クラス要素で該当する属性が検索されます。

単純な例として、次のページでは、ページ内の両方の <xsql:query> アクションに対して max パラメータのデフォルト値を 10 に指定します。

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
</example>
```

次の例では、max パラメータのデフォルト値を、1 つ目の問合せでは 5 に、2 つ目の問合せでは 7 に、3 つ目の問合せでは 10 に指定します。

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max="5" max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max="7" max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE3</xsql:query>
</example>
```

これらのすべてのデフォルト値は、次のような要求で max パラメータの値を指定すると、オーバーライドされます。

```
http://yourserver.com/example.xsql?max=3
```

バインド変数のデフォルト値を指定する場合も、同じ規則に従います。次のようなページは、あまり有効ではありませんが、参考になる例です。

```
<example val="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

前述のページは、パラメータを指定せずに要求した場合、次の XML データグラムを戻します。

```
<example>
  <rowset>
    <row>
      <somevalue>10</somevalue>
    </row>
  </row>
</example>
```

一方、次のような要求を行ったと想定します。

```
http://yourserver.com/example.xsql?val=3
```

この要求は、次の結果を戻します。

```
<example>
  <rowset>
    <row>
      <somevalue>3</somevalue>
    </row>
  </row>
</example>
```

バインド変数の重要な点を示すために、次のとおり val 属性を削除して、val パラメータのデフォルト値をページから削除すると想定します。

```
<example connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

パラメータを指定していないページ要求は、次の結果を戻します。

```
<example>
  <rowset/>
</example>
```

これは、デフォルト値または要求で指定された値のどちらも設定されていないパラメータにバインドされているバインド変数が NULL にバインドされ、前述のページの例に示す WHERE 句が行を戻さないためです。

様々なパラメータの理解

XSQL ページでは、要求で指定されるパラメータの他に、ページのプライベート・パラメータを使用できます。このパラメータの名前および値は、ページ内のアクションが決定します。アクションが bind-params 属性または字句パラメータ参照内で *param* という名前のパラメータへの参照を検出すると、*param* パラメータの値が次の値を使用して解決されます。

1. *param* という名前の、ページのプライベート・パラメータの値（設定されている場合）
2. 前述の値が設定されていない場合は、*param* という名前の要求パラメータの値（指定されている場合）

3. 前述の値が指定されていない場合は、現行のアクション・ハンドラ要素またはその祖先クラス要素の 1 つに対する *param* という名前の属性によって指定されているデフォルト値
4. 前述の値が設定されていない場合は、バインド変数に対しては NULL 値、および字句パラメータに対しては空の文字列

XSQL Servlet が HTTP 上で処理する XSQL ページでは、その他に、HTTP セッション・レベル変数および HTTP Cookie という 2 つの HTTP 固有のパラメータを設定および参照できます。XSQL Servlet を介して処理される XSQL ページでは、パラメータ値解決スキームが追加され、*param* パラメータの値が次の値を使用して解決されます。

1. *param* という名前の、ページのプライベート・パラメータの値（設定されている場合）
2. 前述の値が設定されていない場合は、*param* という名前の Cookie の値（設定されている場合）
3. 前述の値が設定されていない場合は、*param* という名前のセッション変数の値（設定されている場合）
4. 前述の値が設定されていない場合は、*param* という名前の要求パラメータの値（指定されている場合）
5. 前述の値が設定されていない場合は、現行のアクション・ハンドラ要素またはその祖先クラス要素の 1 つに対する *param* という名前の属性によって指定されているデフォルト値
6. 前述の値が設定されていない場合は、バインド変数に対しては NULL 値、および字句パラメータに対しては空の文字列

ユーザーが要求にパラメータ値を指定して、HTTP セッションで設定されている同じ名前のパラメータ（存続期間は HTTP セッションの存続期間で、Web サーバーが制御）、または Cookie として設定されている同じ名前のパラメータ（ブラウザ・セッション中存続するように設定可能）をオーバーライドできないように、このような解決順序が設定されています。

<xsql:include-request-params> アクション

`<xsql:include-request-params>` アクションを使用すると、要求内のすべてのパラメータの XML 表示をデータグラムに挿入できます。これは、対応付けられた XSLT スタイルシートが XPath 式を使用して要求パラメータ値のいずれかを参照する必要がある場合に有効です。

このアクションの構文は、次のとおりです。

```
<xsql:include-request-params/>
```

XSQL Servlet を介してページを処理する場合は、次の形式の XML が挿入されます。

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
      :
  </parameters>
</request>
```

または、次の形式の XML が挿入されます。

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
      :
  </parameters>
  <session>
    <sessVarName>value1</sessVarName>
      :
  </session>
  <cookies>
    <cookieName>value1</cookieName>
      :
  </cookies>
</request>
```

このアクションには、必須またはオプションの属性はありません。

<xsql:include-param> アクション

<xsql:include-param> アクションを使用すると、単一のパラメータの XML 表示をデータグラムに挿入できます。これは、対応付けられた XSLT スタイルシートが XPath 式を使用してそのパラメータ値を参照する必要がある場合に有効です。

このアクションの構文は、次のとおりです。

```
<xsql:include-param name="paramname" />
```

この name 属性は必須で、値を挿入する必要があるパラメータの名前を指定します。このアクションには、オプションの属性はありません。

次の形式の XML が挿入されます。

```
<paramname>value1</paramname>
```

<xsql:include-xml> アクション

<xsql:include-xml> アクションは、ローカル・リソースまたはリモート・リソースの XML コンテンツをデータグラムに挿入します。リソースは、URL で指定されます。

このアクションの構文は、次のとおりです。

```
<xsql:include-xml href="URL"/>
```

URL には、他の Web サイトから XML を取り出すための HTTP ベースの絶対 URL、またはファイル・システム上のファイルから XML を挿入するための相対 URL を指定できます。href 属性は必須です。また、このアクションには、その他のオプションの属性はありません。

<xsql:set-page-param> アクション

<xsql:set-page-param> アクションは、ページのプライベート・パラメータの値を設定します。この値には、静的テキストと他のパラメータ値を組み合せて指定できます。また、別の方法として、SQL の SELECT 文の結果から指定することもできます。

このアクションの構文は、次のとおりです。

```
<xsql:set-page-param name="paramname" value="value"/>
```

または

```
<xsql:set-page-param name="paramname">
  SQL select statement
</xsql:set-page-param>
```

SQL 文を使用する方法では、単一行が結果セットからフェッチされ、パラメータに最初の列の値が割り当てられます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

name 属性は必須です。value 属性と含まれる SQL 文は、相互に排他的です。どちらか一方が指定されている場合は、他方を指定できません。

表 10-4 に、このアクションがサポートする属性を示します。太字の属性は必須です。

表 10-4 <xsql:set-page-param> の属性

属性名	説明
name = "*string*"	値を設定する、ページのプライベート・パラメータの名前。
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
ignore-empty-value = "*boolean*"	ページ・レベルのパラメータに割当て中の値が空の文字列である場合に、その割当てを無視するかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。

<xsql:set-session-param> アクション

<xsql:set-session-param> アクションは、HTTP セッション・レベルのパラメータの値を設定します。セッション・レベルのパラメータの値は、現行ブラウザ・ユーザーの HTTP セッションの存続期間中、存続します。HTTP セッションの存続期間は、Web サーバーによって制御されます。この値は、静的テキストと他のパラメータ値を組み合せて指定できます。また、別の方法として、SQL SELECT 文の結果から指定することもできます。

この機能は Java サーブレットに固有であるため、このアクションは、それが使用されている XSQL ページが XSQL Servlet によって処理されている場合にのみ有効です。このアクションは、XSQL コマンドライン・ユーティリティまたは XSQLRequest プログラム API によって処理されている XSQL ページで検出された場合は、no-op です。

このアクションの構文は、次のとおりです。

```
<xsql:set-session-param name="paramname" value="value"/>
```

または

```
<xsql:set-session-param name="paramname">
  SQL select statement
</xsql:set-session-param>
```

SQL 文を使用する方法では、単一行が結果セットからフェッチされ、パラメータに最初の列の値が割り当てられます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

name 属性は必須です。value 属性と含まれる SQL 文は、相互に排他的です。どちらか一方が指定されている場合は、他方を指定できません。

表 10-5 に、このアクションがサポートするオプションの属性を示します。

表 10-5 <xsql:set-session-param> の属性

属性名	説明
name = "*string*"	値を設定するセッション・レベルの変数の名前。
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
ignore-empty-value = "*boolean*"	ページ・レベルのパラメータに割当て中の値が空の文字列である場合に、その割当てを無視するかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。
only-if-unset = "*boolean*"	セッション変数が存在しない場合にのみセッション変数を割り当てるかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。

<xsql:set-cookie> アクション

<xsql:set-cookie> アクションは、HTTP Cookie の値を設定します。デフォルトでは、Cookie の値は現行のブラウザの存続期間中、存続します。ただし、その存続期間は、オプションの max-age 属性を指定して変更できます。この値は、静的テキストと他のパラメータ値を組み合せて指定できます。また、別の方法として、SQL の SELECT 文の結果から指定することもできます。

この機能は HTTP プロトコル固有であるため、このアクションは、それが使用されている XSQL ページが XSQL Servlet によって処理されている場合にのみ有効です。このアクションは、XSQL コマンドライン・ユーティリティまたは XSQLRequest プログラム API によって処理されている XSQL ページで検出された場合は、no-op です。

このアクションの構文は、次のとおりです。

```
<xsql:set-cookie name="paramname" value="value"/>
```

または

```
<xsql:set-cookie name="paramname">
  SQL select statement
</xsql:set-cookie>
```

SQL 文を使用する方法では、単一行が結果セットからフェッチされ、パラメータに最初の列の値が割り当てられます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

name 属性は必須です。value 属性と含まれる SQL 文は、相互に排他的です。どちらか一方が指定されている場合は、他方を指定できません。

表 10-6 に、このアクションがサポートするオプションの属性を示します。

表 10-6 <xsql:set-cookie> の属性

属性名	説明
name = "*string*"	値を設定する Cookie の名前。
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
domain = "*string*"	Cookie の値が有効で読取り可能なドメイン。ドメインが明示的に設定されていない場合は、デフォルトで、その Cookie を作成するドキュメントの完全修飾ホスト名（bigserver.yourcompany.com など）に設定されます。
ignore-empty-value = "*boolean*"	ページ・レベルのパラメータに割当て中の値が空の文字列である場合に、その割当てを無視するかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。
max-age = "*integer*"	Cookie の存続期間の最大値（秒）を設定します。デフォルトでは、Cookie が現行ブラウザ・ユーザー・セッションの終了時に期限切れになるように設定されます。
only-if-unset = "*boolean*"	Cookie が存在しない場合にのみ Cookie を割り当てるかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。
path = "*string*"	Cookie の値が有効で読取り可能なドメイン内の相対 URL パス。パスが明示的に設定されていない場合は、デフォルトで、その Cookie を作成するドキュメントの URL パスに設定されます。

<xsql:set-stylesheet-param> アクション

<xsql:set-stylesheet-param> アクションは、最上位の XSLT スタイルシート・パラメータの値を設定します。この値は、静的テキストと他のパラメータ値を組み合せて指定できます。また、別の方法として、SQL SELECT 文の結果から指定することもできます。スタイルシート・パラメータ値は、現行のページの処理中に使用されるすべてのスタイルシートに対して設定できます。

このアクションの構文は、次のとおりです。

```
<xsql:set-stylesheet-param name="paramname" value="value"/>
```

または

```
<xsql:set-stylesheet-param name="paramname">
  SQL select statement
</xsql:set-stylesheet-param>
```

SQL 文を使用する方法では、単一行が結果セットからフェッチされ、パラメータに最初の列の値が割り当てられます。このアクションを使用するには、そのアクションを使用する XSQL ページのドキュメント要素に connection="*connname*" 属性を指定し、データベース接続を指定する必要があります。

name 属性は必須です。value 属性と含まれる SQL 文は、相互に排他的です。どちらか一方が指定されている場合は、他方を指定できません。

表 10-7 に、このアクションがサポートするオプションの属性を示します。

表 10-7 <xsql:set-stylesheet-param> の属性

属性名	説明
name = "*string*"	値を設定する最上位のスタイルシート・パラメータの名前。
bind-params = "*string*"	順序付けられ、スペースで区切られた 1 つ以上の XSQL パラメータ名のリスト。この値は、SQL 文内の該当する順序位置にある JDBC バインド変数にバインドするために使用されます。
ignore-empty-value = "*boolean*"	ページ・レベルのパラメータに割当て中の値が空の文字列である場合に、その割当てを無視するかどうかを指定します。有効値は、yes および no です。デフォルト値は no です。

<xsql:include-xsql> を使用した情報の集計

<xsql:include-xsql> アクションを使用すると、XSQL ページの結果を他のページに簡単に挿入できます。これによって、作成済のページのコンテンツを簡単に集計し、別の用途に使用できます。次の例は、<xsql:include-xsql> の最も一般的な 2 つの使用方法を示します。

ディスカッション・フォーラムのカテゴリを示す次の XSQL ページがあると想定します。

```
<!-- Categories.xsql -->
<xsql:query connection="forum" xmlns:xsql="urn:oracle-xsql">
  SELECT name
    FROM categories
    ORDER BY name
</xsql:query>
```

このページの結果を、次のとおり、現行のフォーラムで最新のトピック 10 個を示すページに挿入できます。

```
<!-- TopTenTopics.xsql -->
<top-ten-topics connection="forum" xmlns:xsql="urn:oracle-xsql">
  <topics>
    <xsql:query max-rows="10">
      SELECT subject FROM topics ORDER BY last_modified DESC
    </xsql:query>
  </topics>
  <categories>
    <xsql:include-xsql href="Categories.xsql"/>
  </categories>
</top-ten-topics>
```

また、<xsql:include-xsql> を使用して既存のページを挿入し、それに対して XSLT スタイルシートを適用することもできます。たとえば、次の 2 つの異なる XSLT スタイルシートがあると想定します。

- cats-as-html.xsl（HTML でのトピックのレンダリング用）
- cats-as-wml.xsl（WML でのトピックのレンダリング用）

この場合、2 つの異なるタイプのデバイスに対応するための 1 つの方法は、各デバイス用に異なる XSQL ページを作成することです。1 つのページは次のとおりです。

```
<?xml version="1.0"?>
<!-- HTMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

このページは、Categories.xsql を集計し、cats-as-html.xsl スタイルシートを適用します。もう 1 つのページは次のとおりです。

```
<?xml version="1.0"?>
<!-- WMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

このページは、Categories.xsql を集計し、cats-as-wml.xsl スタイルシートを適用してワイヤレス・デバイスに配信します。この方法では、再利用可能な Categories.xsql ページのコンテンツを 2 つの異なる方法で別の用途に使用しました。

集計されているページに <?xml-stylesheet?> 処理命令が含まれている場合は、結果が集計される前にそのスタイルシートが適用されます。そのため、<xsql:include-xsql> を使用して、XSLT スタイルシートの適用を簡単に連鎖させることもできます。

1 つの XSQL ページが <xsql:include-xsql> を使用して他のページのコンテンツを集計する場合、ネストしたページは要求レベルのすべてのパラメータを参照できます。XSQL Servlet が処理するページでは、セッション・レベルのパラメータおよび Cookie も参照できます。集計側のページの *page-private* パラメータは、ネストしたページからは参照できません。

表 10-8 に、このアクションがサポートする属性を示します。太字の属性は必須です。

表 10-8 <xsql:include-xsql> の属性

属性名	説明
href = "*string*"	挿入する XSQL ページの相対 URL または絶対 URL。
reparse = "*boolean*"	挿入された XSQL ページの出力を挿入前に再解析するかどうかを指定します。これは、挿入側のページが要素として処理する必要がある XML 文書フラグメントのテキストを、挿入された XSQL ページが選択している場合に有効です。有効値は、yes および no です。デフォルト値は no です。

ポストされた情報の処理

XSQL ページ・フレームワークは、XML コンテンツの作成および変換だけでなく、ポストされた XML コンテンツの処理も簡単にします。組込みアクションによって、XML 文書形式および HTML 形式のポストされた情報の処理が簡単になり、その情報を Oracle XSU の基礎となる機能を使用してデータベース表に直接ポストすることができます。

XSU は、ターゲット表またはビューに必要な正規の形式の XML 文書のコンテンツに基づいて、データをデータベースに挿入、更新および削除する機能を提供します。特定のデータベース表では、そのデータの正規の XML 形式は、その表に対する SELECT * FROM *tablename* 問合せからの XML 出力の 1 行によって指定されます。この正規の形式の XML 文書では、XSU は挿入、更新または削除操作（あるいはそのすべて）を自動化することができます。XSU と XSLT 変換を組み合せると、すべての形式の XML を特定の表に対して適切な正規の形式に変換し、結果として戻る正規の XML を挿入、更新、削除するように XSU に要求できます。

次の組込み XSQL アクションを使用すると、この機能を XSQL ページ内から簡単に使用できます。

- <xsql:insert-request>

  要求でポストされ、オプションで変換された XML 文書を表に挿入します。表 10-9 に、このアクションがサポートする必須およびオプションの属性を示します。

- <xsql:update-request>

  要求でポストされ、オプションで変換された XML 文書を表またはビューで更新します。表 10-10 に、このアクションがサポートする必須およびオプションの属性を示します。

- `<xsql:delete-request>`

  要求でポストされ、オプションで変換された XML 文書を表またはビューから削除します。表 10-11 に、このアクションがサポートする必須およびオプションの属性を示します。

- `<xsql:insert-param>`

  要求パラメータの値としてポストされ、オプションで変換された XML 文書を表またはビューに挿入します。表 10-12 に、このアクションがサポートする必須およびオプションの属性を示します。

データベース・ビューを挿入のターゲットにする場合、そのビューに対する INSTEAD OF INSERT トリガーを作成し、ポストされた情報の処理をさらに自動化することができます。たとえば、ビューの INSTEAD OF INSERT トリガーは、PL/SQL を使用してレコードの有無を確認し、その確認結果に応じて INSERT または UPDATE のどちらを実行するかを効果的に選択できます。

表 10-9 <xsql:insert-request> の属性

属性名	説明
table = "*string*"	XML 情報の挿入に使用する表、ビューまたはシノニムの名前。
transform = "*URL*"	挿入されるドキュメントを正規の ROWSET/ROW 形式に変換するために使用する XSLT 変換の相対 URL または絶対 URL。
columns = "*string*"	値を挿入する 1 つ以上の列名のリスト。指定する列名はスペースまたはカンマで区切ります。列名を指定すると、指定された列のみが挿入されます。列名を指定しない場合は、すべての列が挿入され、値が XML 文書に表示されない列には NULL 値が指定されます。
commit-batch-size = "*integer*"	0（ゼロ）以外の正数である数値 N を指定すると、N 個のレコードが挿入されるたびにコミットされます。数値を指定しない場合は、デフォルトのバッチ・サイズは 0（ゼロ）で、途中でコミットされることはありません。
date-format = "*string*"	挿入中の XML 内の日付フィールド値を解析するために使用する日付書式マスク。有効値は、java.text.SimpleDateFormat クラスの有効値です。

表 10-10 <xsql:update-request> の属性

属性名	説明
table = "*string*"	XML 情報の挿入に使用する表、ビューまたはシノニムの名前。
key-columns = "*string*"	ポストされた XML 文書内の値が、更新する既存の行を識別するために使用される 1 つ以上の列名のリスト。指定する列名はスペースまたはカンマで区切ります。
transform = "*URL*"	挿入されるドキュメントを正規の ROWSET/ROW 形式に変換するために使用する XSLT 変換の相対 URL または絶対 URL。
columns = "*string*"	値を更新する 1 つ以上の列名のリスト。指定する列名はスペースまたはカンマで区切ります。列名を指定すると、指定された列のみが更新されます。列名を指定しない場合は、すべての列が更新され、値が XML 文書に表示されない列には NULL 値が指定されます。
commit-batch-size = "*integer*"	0（ゼロ）以外の正数である数値 N を指定すると、N 個のレコードを挿入するたびにコミットされます。数値を指定しない場合は、デフォルトのバッチ・サイズは 0（ゼロ）で、途中でコミットされることはありません。
date-format = "*string*"	挿入中の XML 内の日付フィールド値を解析するために使用する日付書式マスク。有効値は、java.text.SimpleDateFormat クラスの有効値です。

表 10-11 <xsql:delete-request> の属性

属性名	説明
table = "*string*"	XML 情報の挿入に使用する表、ビューまたはシノニムの名前。
key-columns = "*string*"	ポストされた XML 文書内の値が、更新する既存の行を識別するために使用される 1 つ以上の列名のリスト。指定する列名はスペースまたはカンマで区切ります。
transform = "*URL*"	挿入されるドキュメントを正規の ROWSET/ROW 形式に変換するために使用する XSLT 変換の相対または絶対 URL。
commit-batch-size = "*integer*"	0（ゼロ）以外の正数である数値 N を指定すると、N 個のレコードを挿入するたびにコミットされます。数値を指定しない場合は、デフォルトのバッチ・サイズは 0（ゼロ）で、途中でコミットされることはありません。

表 10-12 <xsql:insert-param> の属性

属性名	説明
name = "*string*"	挿入される XML を値に含むパラメータの名前。
table = "*string*"	XML 情報の挿入に使用する表、ビューまたはシノニムの名前。
transform = "*URL*"	挿入されるドキュメントを正規の ROWSET/ROW 形式に変換するために使用する XSLT 変換の相対 URL または絶対 URL。
columns = "*string*"	値を挿入する 1 つ以上の列名のリスト。指定する列名はスペースまたはカンマで区切ります。列名を指定すると、指定された列のみが挿入されます。列名を指定しない場合は、すべての列が挿入され、値が XML 文書に表示されない列には NULL 値が指定されます。
commit-batch-size = "*integer*"	0（ゼロ）以外の正数である数値 N を指定すると、N 個のレコードを挿入するたびにコミットされます。数値を指定しない場合は、デフォルトのバッチ・サイズは 0（ゼロ）で、途中でコミットされることはありません。
date-format = "*string*"	挿入中の XML 内の日付フィールド値を解析するために使用する日付書式マスク。有効値は、java.text.SimpleDateFormat クラスの有効値です。

異なる XML ポスト・オプションの理解

XSQL ページ・フレームワークは、ポストされた情報を 3 つの異なる方法で処理できます。

1. クライアント・プログラムは、要求の本体に XML 文書を含み、HTTP ヘッダーに text/xml の ContentType を指定した、XSQL ページをターゲットとする HTTP POST メッセージを送信できます。

   この場合、<xsql:insert-request>、<xsql:update-request> または <xsql:delete-request> アクションを使用できます。これによって、ポストされた XML のコンテンツが、指定されたとおりターゲット表で挿入、更新または削除されます。XSLT 変換を使用してポストされた XML 文書を変換する場合、ポストされた XML 文書がこの変換のソース・ドキュメントです。

2. クライアント・プログラムは、パラメータの 1 つに XML 文書を含む、XSQL ページに対する HTTP GET 要求を送信できます。

   この場合、<xsql:insert-param> アクションを使用できます。これによって、ポストされた XML パラメータ値の内容が、指定されたとおりターゲット表に挿入されます。XSLT 変換を使用してポストされた XML 文書を変換する場合、パラメータ値内の XML 文書がこの変換のソース・ドキュメントです。

3. ブラウザは、アクションが XSQL ページをターゲットとする、method="POST" が設定された HTML フォームを送信できます。この場合、通常、ブラウザは要求の本体にエンコードされたすべての HTML フォーム・フィールドとその値を含み、ContentType を `application/x-www-form-urlencoded` に指定した HTTP POST メッセージを送信します。

   この場合、この要求は XML 文書を含みません。かわりに、エンコードされたフォーム・パラメータを含みます。ただし、これらの 3 つの場合を同等にするために、XSQL Page Processor は（必要に応じて）要求に含まれるフォーム・パラメータ、セッション変数および Cookie の組合せから XML 文書を生成します。その後、XSLT 変換は、`<xsql:insert>`、`<xsql:update-request>` または `<xsql:delete-request>` を使用してそれぞれ挿入、更新または削除操作を行うために、この動的マテリアライズド XML 文書を正規の形式に変換します。

ポストされた HTML フォームを使用する場合、動的マテリアライズド XML 文書は次の形式になります。

```
<request>
  <parameters>
    <firstparamname>firstparamvalue</firstparamname>
      :
    <lastparamname>lastparamvalue</lastparamname>
  </parameters>
  <session>
    <firstparamname>firstsessionparamvalue</firstparamname>
       :
    <lastparamname>lastsessionparamvalue</lastparamname>
  </session>
  <cookies>
    <firstcookie>firstcookievalue</firstcookiename>
        :
    <lastcookie>firstcookievalue</lastcookiename>
  </cookies>
</request>
```

複数のパラメータが同じ名前でポストされた場合、それらのパラメータは自動的に「行化」され、それ以降の処理を簡単にします。たとえば、次のパラメータをポストまたは挿入する要求があると想定します。

- `id` = 101
- `name` = Steve
- `id` = 102
- `name` = Sita
- `operation` = update

これは、次のような一連の「行化」されたパラメータを作成します。

```
<request>
  <parameters>
    <row>
      <id>101</id>
      <name>Steve</name>
    </row>
    <row>
      <id>102</id>
      <name>Sita</name>
    </row>
    <operation>update</operation>
  </parameters>
       :
</request>
```

要求パラメータを含むこのマテリアライズド XML 文書をターゲット表の正規の形式に変換する XSLT スタイルシートを指定する必要があるため、次のような XSQL ページを独自に作成すると有効である場合があります。

```
<!--
 | ShowRequestDocument.xsql
 | Show Materialized XML Document for an HTML Form
 +-->
<xsql:include-request-params xmlns:xsql="urn:oracle-xsql"/>
```

このページを適切に配置すると、HTML フォームを一時的に変更して、`ShowRequestDocument.xsql` ページにポストすることができます。また、ブラウザで、マテリアライズド XML 要求ドキュメントの未加工の XML が表示されます。この XML を保存して、XSLT 変換を展開するために使用できます。

カスタム XSQL アクション・ハンドラの使用

組込みアクション・ハンドラが処理しないタスクを実行する必要がある場合は、XSQL ページ・フレームワークを使用すると、カスタム・アクションをコールして、ページ処理の一部として実行する必要があるすべての種類のジョブを実行できます。カスタム・アクションは、任意の XML コンテンツをデータ・ページに提供し、任意の処理を実行することができます。Java でのカスタム・アクション・ハンドラの作成の詳細は、10-63 ページの「カスタム XSQL アクション・ハンドラの作成」を参照してください。この項では、作成済のカスタム・アクション・ハンドラの使用方法について説明します。

カスタム・アクション・ハンドラをコールするには、組込みアクション・ハンドラ要素である <xsql:action> を使用します。このアクション・ハンドラには、handler という名前の単一の必須属性があります。この属性の値は、コールするアクションの完全修飾 Java クラス名です。このクラスは、oracle.xml.xsql.XSQLActionHandler インタフェースを実装する必要があります。次に例を示します。

```
<xsql:action handler="yourpackage.YourCustomHandler"/>
```

通常の方法で、任意の数の追加属性をハンドラに指定できます。たとえば、yourpackage.YourCustomHandler に param1 および param2 という名前の属性が必要である場合、次の構文を使用します。

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy"/>
```

一部のアクション・ハンドラでは、属性の他に、テキスト内容または要素内容を <xsql:action> 要素内で使用する必要がある場合があります。その場合は、次のとおり必要な構文を使用します。

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">
   Some Text Goes Here
</xsql:action>
```

または、

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">
  <some>
    <other/>
    <elements/>
    <here/>
  </some>
</xsql:action>
```

XSQL Servlet の例の説明

図 10-13 に、./demo ディレクトリ内にあるソフトウェア付属の XSQL Servlet のアプリケーション例を示します。

表 10-13 XSQL Servlet の例

デモンストレーション名	説明
Hello World ./demo/helloworld	最も単純な XSQL ページ。
Do You XML Site ./demo/doyouxml	XSQL ページが XSQL ページを使用してデータ駆動型の Web サイトを構築する方法を示します。問合せに SQL、XSQL の置換変数を使用し、フォーマットに XSLT を使用します。 <xsql:query> タグ内の SQL 文および <xsql:query> タグの属性に代替パラメータを使用して、問合せ結果を介してページングするために、表示またはスキップするレコードの数などを制御します。
Employee Page ./demo/emp	XSQL ページが、XSQL ページ・パラメータを使用して従業員およびデータのソートを制御し、EMP 表の XML データを表示します。 対応付けられた XSLT スタイルシートを使用して、結果を HTML バージョンの emp.xsql ページとしてフォーマットします。これは、構成アクションであるため、検索基準を微調整できます。
Insurance Claim Page ./demo/insclaim	構造化された Insurance Claim オブジェクト・ビューに対するサンプル問合せを示します。insclaim.sql は、INSURANCE_CLAIM_VIEW オブジェクト・ビューを設定し、サンプル・データを作成します。
Invalid Classes Page ./demo/classerr	XSQL ページが、invalidclasses.xsl を使用して、スキーマ内で発生した現在の Java クラスのコンパイル・エラーの最新リストをフォーマットします。.sql スクリプトは、デモ用の XSQLJavaClassesView オブジェクト・ビューを設定します。オブジェクト・ビューからのマスター情報 / ディテール情報は、サーバー内の invalidclasses.xsl スタイルシートによって HTML にフォーマットされます。
Airport Code Validation ./demo/airport	XSQL ページが、3 文字の空港コードに基づいて、空港情報のデータグラムを戻します。最初の問合せが行を戻さない場合、代替の問合せとして <xsql:no-rows-query> を使用します。XSQL ページは、渡された空港コードの一致検索を試みた後、空港の説明に基づいて、あいまい一致検索を試みます。 airport.htm ページは、Java スクリプトによって Internet Explorer 5.0 の組込み XML DOM 機能を使用して、Web ページからの airport.xsql ページの XML 結果を使用する方法を実例で示します。 Web ページ上で 3 文字の空港コードを入力すると、Java スクリプトは、入力したコードに対応する Web 上で XSQL Servlet から XML データグラムをフェッチします。戻り値が、一致するものがなかったことを示す場合、プログラムは、XSQL Servlet から XML データグラム形式で戻された情報に基づいて、一致する可能性があるもののリストを生成します。

表 10-13 XSQL Servlet の例（続き）

デモンストレーション名	説明
Airport Code Display ./demo/airport	Airport Code Validation の例と同じ XSQL ページを使用して実例を示しますが、要求で XSLT スタイルシート名を提供します。これによって、空港情報は、未加工の XML として戻されるのではなく、HTML フォームにフォーマットされます。
Emp/Dept Object Demo ./demo/empdept	オブジェクト・ビューを使用して、EMP や DEPT などの 2 つの既存フラット表からのマスター情報 / ディテール情報をグループ化する方法を示します。empdeptobjs.sql スクリプトが、オブジェクト・ビューおよび INSTEAD OF INSERT トリガーを作成し、マスター・ビュー / ディテール・ビューを xsql:insert-request の挿入ターゲットとして使用できるようにします。 empdept.xsl スタイルシートは、最上位に余分な xsl:stylesheet または xsl:transform のない HTML ページのように見える、単純な形式の XSLT スタイルシートの例を示します。リテラル結果要素をスタイルシートとして使用し、XSLT 1.0 仕様の一部がコールされます。 生成された HTML が、coolcolors.css ファイル内にある、集中化された HTML スタイルの情報用の CSS を十分に使用できるように、<link rel="stylesheet"> を含む HTML ページを生成する方法も示します。
Adhoc Query Visualization ./demo/adhocsql	パラメータとして使用する SQL 問合せおよび XSLT スタイルシートをサーバーに渡す方法を示します。 **注意 :** このデモ・ページを本番環境に配置する場合は、SCOTT ユーザー・アカウントがアクセスできる Web 上ですべての SQL 問合せの結果が XML 形式で公開されるため、十分な注意が必要です。
XML Document Demo ./demo/document	XML 文書をリレーショナル表に挿入する方法を示します。 docdemo.sql スクリプトが、XMLDOCFRAG という、CLOB 型の属性を含むユーザー定義型を作成します。 ■ 文書のテキストを /xsql/demo/xml99.xml に挿入し、xml99.xsl という名前をスタイルシートに指定します。 ■ スタイルシート relnotes.xsl を使用して、文書のテキストを ./xsql/demo/JDevRelNotes.xml に挿入します。 docstyle.xsql ページは、クライアントが提供するスタイルシート名を使用して最終出力を変換する前に、doc.xsql ページの出力を独自のページに挿入するためのアクション・ハンドラ要素である <xsql:include-xsql> の例を示します。 XML Document Demo は、Internet Explorer 5.0 のクライアント側 XML 機能を使用して、文書がサーバーにポストされる前に、その文書が整形式であるかどうかを確認します。

表 10-13 XSQL Servlet の例（続き）

デモンストレーション名	説明
XML Insert Request Demo ./demo/insertxml	クライアントからの XML を XSQL ページにポストします。この XSQL ページは、アクション・ハンドラ要素である <xsql:insert-request> を使用して、ポストされた XML 情報をデータベース表に挿入します。 このデモでは、moreover.com が提供する XML ベースのニュース形式の XML 文書を使用できます。XML をポストするプログラムは、Internet Explorer 5.0 および Java スクリプトから XMLHttpRequest オブジェクトを使用するクライアント側の Web ページです。 insertnewsstory.xsql ページのソースが、表名および XSLT 変換名を指定します。 moreover-to-newsstory.xsl スタイルシートは、受信した XML を OracleXMLSave ユーティリティが挿入できる正規の形式に変換します。<article> 要素の例を <moreovernews> 要素内に数回コピー・アンド・ペーストして、数個のニュース記事を一度に挿入します。 newsstory.sql は、受信データの処理方法、主キーのデフォルト値などをカスタマイズするために、XSQL ページにデータを挿入させる挿入先のデータベース・ビューで INSTEAD OF トリガーを使用する方法を示します。
SVG Demo ./demo/svg	deptlist.xsql ページが、SalChart.xsql ページへのハイパーリンクを使用して、単純な部門リストを表示します。 SalChart.xsql ページは、パラメータとして渡された特定の部門の従業員を問い合せ、SalChart.xsql スタイルシートを使用して、問合せの結果をその部門内の従業員の給与を比較する棒グラフの SVG にフォーマットします。
PDF Demo ./demo/fop	emptable.xsql ページが、単純な従業員リストを表示します。emptable.xsl スタイルシートは、データ・ページを XSLFO に変換します。XSLFO は、組込み FOP シリアル化コードと組み合されて、結果を Adobe PDF 形式にレンダリングします。

デモ・データの設定

デモ・データを設定するには、次の手順を実行します。

1. ディレクトリをマシン上の ./demo ディレクトリに変更します。
2. このディレクトリで SQL*Plus を実行します。Oracle Text（*inter*Media Text）パッケージのスキーマ所有者である CTXSYS/CTXSYS としてデータベースに接続し、次のコマンドを発行します。

```
GRANT EXECUTE ON CTX_DDL TO SCOTT;
```

3. SYSTEM/MANAGER としてデータベースに接続し、次のコマンドを発行します。

   ```
   GRANT QUERY REWRITE TO SCOTT;
   ```

   これによって、SCOTT は、デモの 1 つが空港の説明に基づく大文字 / 小文字を区別しない問合せを実行するために使用するファンクション索引を作成できるようになります。

4. SCOTT/TIGER としてデータベースに接続します。

5. スクリプト `install.sql` を `./demo` ディレクトリで実行します。このスクリプトは、すべてのデモ用のすべての SQL スクリプトを実行します。

   ```
   install.sql
   @@insclaim/insclaim.sql
   @@document/docdemo.sql
   @@classerr/invalidclasses.sql
   @@airport/airport.sql
   @@insertxml/newsstory.sql
   @@empdept/empdeptobjs.sql
   ```

6. ディレクトリを `./doyouxml` サブディレクトリに変更し、次のコマンドを実行します。

   ```
   imp scott/tiger file=doyouxml.dmp
   ```

   これは、Do You XML Site デモ用のサンプル・データをインポートするために実行します。

7. SVG のデモンストレーションを体験するには、Adobe 社製の SVG Plug-in などの SVG プラグインをブラウザにインストールします。

XSQL ページの高度なトピック

クライアントによるスタイルシートのオーバーライド・オプションの理解

要求中の現行の XSQL ページが許可する場合、要求で XSLT スタイルシートの URL を指定して、使用されるデフォルトのスタイルシートをオーバーライドしたり、デフォルトでスタイルシートが適用されない場合にスタイルシートを適用することができます。クライアントが起動するスタイルシート URL は、`xml-stylesheet` パラメータを要求の一部として指定することによって、指定されます。このパラメータの有効値は次のとおりです。

- 処理中の XSQL ページに対して相対的に解析されるすべての相対 URL
- HTTP プロトコル・スキームを使用し、トラステッド・ホスト（`XSQLConfig.xml` ファイルで定義）を参照するすべての絶対 URL
- リテラル値 `none`

この最後の値である xml-stylesheet=none は、開発中、XSLT スタイルシートの処理を一時的に短絡させてスタイルシートが実際に参照している XML データグラムを確認する場合に特に有効です。これは、スタイルシートが予期される結果を生成しない場合に、その理由を理解するうえで有効です。

クライアントによる XSQL ページ用スタイルシートのオーバーライドは、次のどちらかの方法によって禁止できます。

- XSQLConfig.xml ファイルで allow-client-style 構成パラメータを no に設定します。
- XSQL ページのドキュメント要素に対して、allow-client-style="no" 属性を明示的に挿入します。

XSQLConfig.xml 構成ファイルでクライアントによるスタイルシートのオーバーライドがデフォルトでグローバルに使用禁止にされている場合でも、すべてのページで、そのページのドキュメント要素に対して allow-client-style="yes" 属性を挿入することによって、クライアントによるオーバーライドを明示的に使用可能にできます。

スタイルシートの処理方法の制御

戻されたドキュメントの内容の型の制御

提供する情報の内容の型を設定することは、非常に重要です。これによって、要求側クライアントは、戻された情報を正しく解析できます。スタイルシートが <xsl:output> 要素を使用する場合、XSQL Page Processor は、<xsl:output> の media-type 属性および encoding 属性から、戻されたドキュメントのメディア・タイプおよびエンコーディングを推測します。

たとえば、次のスタイルシートは、<xsl:output> に対して media-type="application/vnd.ms-excel" 属性を使用し、emp 表に対する標準の問合せを含む XSQL ページの結果を Microsoft Excel のスプレッドシート形式に変換します。

```
<?xml version="1.0"?>
<!-- empToExcel.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" media-type="application/vnd.ms-excel"/>
  <xsl:template match="/">
   <html>
     <table>
       <tr><th>EMPNO</th><th>ENAME</th><th>SAL</th></tr>
       <xsl:for-each select="ROWSET/ROW">
         <tr>
           <td><xsl:value-of select="EMPNO"/></td>
           <td><xsl:value-of select="ENAME"/></td>
           <td><xsl:value-of select="SAL"/></td>
         </tr>
       </xsl:for-each>
     </table>
   </html>
  </xsl:template>
</xsl:stylesheet>
```

このスタイルシートを使用する XSQL ページは、次のようになります。

```
<?xml version="1.0"?>
<?xml-stylesheet href="empToExcel.xsl" type="text/xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  select * from emp order by sal desc
</xsql:query>
```

スタイルシートの動的割当て

前述のとおり、<?xml-stylesheet?> 処理命令を .xsql ファイルの最上部に挿入すると、XSQL Page Processor は結果として生成される XML データグラムの変換にそれを使用することを検討します。次に例を示します。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query>
    SELECT * FROM emp ORDER BY sal DESC
  </xsql:query>
</page>
```

このページは、emp.xsl スタイルシートを使用して、リクエスタに応答する前に EMP 問合せの結果をサーバー層で変換します。このスタイルシートは、<?xml-stylesheet?> 処理命令に対する href 擬似属性で指定される相対 URL または絶対 URL によってアクセスされます。

href 擬似属性の値に 1 つ以上のパラメータ参照を挿入すると、スタイルシートの名前を動的に決定できます。たとえば、次のページは、問合せを使用してページのプライベート・パラメータの値を割り当てることによって、表から使用するスタイルシートの名前を選択します。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:set-page-param bind-params="UserCookie" name="sheet">
    SELECT stylesheet_name
      FROM user_prefs
     WHERE username = ?
  </xsql:set-page-param>
  <xsql:query>
    SELECT * FROM emp ORDER BY sal DESC
  </xsql:query>
</page>
```

クライアントでのスタイルシートの処理

Microsoft 社製の Internet Explorer 5.0 以上などの一部のブラウザは、クライアントでの XSLT スタイルシートの処理をサポートします。これらのブラウザは、<?xml-stylesheet?> 処理命令を使用して、サーバー側の XSQL ページと同じ方法で XML 文書用に処理されるスタイルシートを認識します。これは偶然ではありません。この目的に <?xml-stylesheet?> を使用することは、1999 年 6 月 29 日に公開された W3C 勧告「Associating Stylesheets with XML Documents, Version 1.0」の一部です。

デフォルトでは、XSQL Page Processor は XSLT 変換をサーバーで実行します。ただし、擬似属性 client="yes" を XSQL ページの <?xml-stylesheet?> 処理命令に追加すると、Page Processor は、ドキュメントの最上部に現行の <?xml-stylesheet?> が挿入された状態で未加工の XML データグラムを提供し、クライアントに対する XSLT 処理を遅延します。

注意する必要がある重要な点は、1998 年後半に出荷された Internet Explorer 5.0 には、1998 年 12 月の XSLT 草案に準拠する XSL スタイルシート言語の実装が含まれていることです。1999 年 11 月に最終的に公開された XSLT 1.0 勧告は、Internet Explorer 5.0 が準拠する以前の草案から大幅に変更されています。これは、Internet Explorer 5.0 ブラウザが、XSLT 1.0 勧告の構文を実装する他のすべての XSLT プロセッサ（Oracle XSLT Processor など）とは異なる XSLT を理解することを意味します。

2000 年末にかけて、Microsoft 社は MSXML コンポーネントのバージョン 3.0 を Web 上からダウンロード可能なリリースとして公開しました。この最新バージョンは XSLT 1.0 標準を実装しますが、Internet Explorer 5.0 ブラウザ内で XSLT プロセッサとして使用するには、追加のインストール手順を実行する必要があります。ただし、サーバーは Internet Explorer 5.0 ブラウザが最新の XSLT コンポーネントをインストール済であるかどうかを検出できないため、デフォルトで最新のコンポーネントを含み、バージョン番号 6.0 を含む検出可能な異なるユーザー・エージェント文字列を送信する Internet Explorer 6.0 が公開されるまで、Internet Explorer 5.0 ブラウザに配信され、クライアントで処理されるスタイルシートは Internet Explorer 5.0 用の以前の XSL 機能を使用する必要があります。

XSQL ページが要求を行うユーザー・エージェントに応じて異なるスタイルシートを使用するように要求できる必要があります。XSQL ページ・フレームワークは、この操作を簡単にします。その方法については、次の項を参照してください。

複数のユーザー・エージェント固有のスタイルシートの提供

XSQL ページの最上部に複数の `<?xml-stylesheet?>` 処理命令を挿入し、そのすべての処理命令にオプションの `media` 擬似属性を含めることができます。`media` 擬似属性に値を指定すると、その値は大文字 / 小文字を区別して HTTP ヘッダーのユーザー・エージェント文字列の値と比較されます。`media` 擬似属性の値がユーザー・エージェント文字列の一部と一致した場合、プロセッサは現行の `<?xml-stylesheet?>` 処理命令を選択して使用します。一致しなかった場合は、その処理命令を無視して、検索を続けます。ドキュメント内の順序で最初に一致した処理命令が使用されます。`media` 擬似属性がない処理命令は、すべてのユーザー・エージェントと一致するため、代替またはデフォルトとして使用できます。

たとえば、.xsql ファイルの最上部に次の処理命令があると想定します。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="lynx" href="doyouxml-lynx.xsl" ?>
<?xml-stylesheet type="text/xsl" media="msie 5" href="doyouxml-ie.xsl" ?>
<?xml-stylesheet type="text/xsl" href="doyouxml.xsl" ?>
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
  :
```

この処理命令は、Lynx ブラウザ用に `doyouxml-lynx.xsl` を、Internet Explorer 5.0 または 5.5 ブラウザ用に `doyouxml-ie.xsl` を、および他のすべてのブラウザ用に `doyouxml.xsl` を使用します。

表 10-14 に、<?xml-stylesheet?> 処理命令に対して使用可能な、サポートされるすべての擬似属性の概要を示します。

表 10-14 <?xml-stylesheet?> の擬似属性

属性名	説明
type = "*string*"	対応付けられたスタイルシートの MIME タイプを指定します。XSLT スタイルシートの場合、この属性は文字列 text/xsl に設定する必要があります。 serializer 属性を使用する場合、この属性は、そのシリアル化コードをコールする前に XSLT スタイルシートを実行する必要があるかどうかによって、存在する場合と存在しない場合があります。
href = "*URL*"	使用する XSLT スタイルシートへの相対 URL または絶対 URL を指定します。http プロトコル・スキームを使用する絶対 URL を指定する場合は、リソースの IP アドレスが XSQLConfig.xml ファイルに示されている信頼できるホストである必要があります。
media = "*string*"	この属性はオプションです。この属性に値を指定すると、その値は、要求側デバイスが送信した HTTP ヘッダーのユーザー・エージェント文字列に対する大文字 / 小文字を区別した一致検索を実行するために使用されます。現行の <?xml-stylesheet?> 処理命令は、ユーザー・エージェント文字列に media 属性の値が含まれている場合にのみ、使用されます。含まれていない場合は、無視されます。
client = "*boolean*"	yes に設定すると、XSQL Page Processor は対応付けられた、クライアントに対する XSLT スタイルシートの処理を遅延します。ドキュメントの最上部に現行の <?xml-stylesheet?> 処理命令が挿入された状態で、未加工の XML データグラムがクライアントに送信されます。値を指定しない場合、デフォルトでは、変換がサーバーで実行されます。

表 10-14 <?xml-stylesheet?> の擬似属性（続き）

属性名	説明
serializer = "*string*"	デフォルトでは、XSQL Page Processor は次のシリアル化コードを使用します。 ■ XML DOM シリアル化コード（XSLT スタイルシートを使用しない場合） ■ XSLT プロセッサのシリアル化コード（XSLT スタイルシートを使用する場合） この擬似属性を指定すると、前述のコードのかわりにカスタム・シリアル化コードの実装を使用する必要があります。 有効値は、XSQLConfig.xml ファイルの <serializerdefs> セクションで定義されるカスタム・シリアル化コードの名前、または文字列 java:*fully.qualified.Classname* です。XSLT スタイルシートと serializer 属性の両方が存在する場合は、まず XSLT 変換が実行され、次にカスタム・シリアル化コードがコールされて最終結果が OutputStream または PrintWriter にレンダリングされます。

XSQLConfig.xml を使用した環境のチューニング

XSQL ページ環境をチューニングするには、XSQLConfig.xml ファイルを使用します。表 10-15 に、設定可能なすべてのパラメータを定義します。

表 10-15 XSQLConfig.xml の構成設定

構成設定名
XSQLConfig/servlet/output-buffer-size バッファ付き出力ストリーム・サイズ（バイト単位）を設定します。サーブレット・エンジンがすでに I/O をサーブレット出力ストリームにバッファリング済である場合は、0（ゼロ）に設定してさらなるバッファリングを回避することができます。 デフォルト値は 0（ゼロ）です。有効値は負ではない整数です。
XSQLConfig/servlet/suppress-mime-charset/media-type XSQL Servlet は、HTTP ContentType ヘッダーが要求に対して戻されているリソースの MIME タイプを示すように設定します。デフォルトでは、XSQL Servlet は MIME タイプにオプションのキャラクタ・セット情報を挿入します。特定の MIME タイプについて、必要な MIME タイプを内容として指定した <media-type> 要素を挿入すると、キャラクタ・セット情報の挿入を抑制できます。 任意の数の <media-type> 要素を指定できます。 有効値はすべての文字列です。

表 10-15 XSQLConfig.xml の構成設定（続き）

構成設定名
XSQLConfig/processor/character-set-conversion/default-charset デフォルトでは、XSQL Page Processor は HTTP パラメータの値に基づいてキャラクタ・セット変換を行い、ほぼすべてのサーブレット・エンジンが使用するデフォルトのキャラクタ・セットを補完します。変換に使用されるデフォルトのベース・キャラクタ・セットは、IANA の ISO-8859-1 キャラクタ・セットに対応する Java キャラクタ・セット 8859_1 です。サーブレット・エンジンが異なるキャラクタセットをベース・キャラクタ・セットとして使用する場合は、この構成設定でその値を指定できます。 キャラクタ・セット変換を抑制するには、<default-charset> 要素の内容として、キャラクタ・セット名ではなく空の要素である <none/> を指定します。これは、サーブレットのデフォルト・キャラクタ・セットを使用して適切に表示できるパラメータ値を使用する場合に有効であり、キャラクタ・セット変換の実行に関連するオーバーヘッドを削減します。 有効値は、すべての Java キャラクタ・セット名または要素 <none/> です。
XSQLConfig/processor/reload-connections-on-error XSQL Page Processor を起動すると、接続定義がキャッシュされます。この設定を yes に設定した場合、キャッシュされた接続のリストに含まれていない接続名の要求が試行されると、XSQL Page Processor は XSQLConfig.xml ファイルを再読取りして、接続定義を再ロードします。yes を設定すると、開発中、サーブレットが実行しているときに新しい <connection> 定義をファイルに追加する場合に有効です。メモリー内のキャッシュで接続名が見つからなかった場合の接続定義ファイルの再ロードを回避するには、no に設定します。 デフォルト値は yes です。有効値は、yes および no です。
XSQLConfig/processor/default-fetch-size SQL 問合せを使用してデータベースから情報を取り出すための行のフェッチ・サイズのデフォルト値を設定します。この設定は、Oracle JDBC ドライバを使用している場合にのみ有効です。それ以外の場合は無視されます。これは、異なる層で実行中のサーブレット・エンジンからデータベースへのネットワーク・ラウンドトリップの削減に有効です。 デフォルト値は 50 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/page-cache-size XSQL ページ・テンプレート用の XSQL キャッシュ・サイズを設定します。この設定は、キャッシュされる XSQL ページの最大数を決定します。この最大数を超えると、最も使用頻度の少ないページがキャッシュから削除されます。 デフォルト値は 25 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/stylesheet-cache-size XSQL スタイルシート用の XSQL キャッシュ・サイズを設定します。この設定は、キャッシュされるスタイルシートの最大数を決定します。この最大数を超えると、最も使用頻度の少ないスタイルシートがキャッシュから削除されます。 デフォルト値は 25 です。有効値は 0（ゼロ）以外の正の整数です。

表 10-15 XSQLConfig.xml の構成設定（続き）

構成設定名
XSQLConfig/processor/stylesheet-pool/initial キャッシュされた各スタイルシートは、実際はスループットを改善するためにキャッシュされたスタイルシート・インスタンスのプールです。この構成設定は、各スタイルシート・プールに対するスタイルシートの初期割当て数を設定します。 デフォルト値は 1 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/stylesheet-pool/increment サーバー上でのロードが増加したためスタイルシート・プールを拡大する必要がある場合に割り当てるスタイルシートの数を設定します。 デフォルト値は 1 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/stylesheet-pool/timeout-seconds プールが縮小して初期サイズに戻ろうとするときに、プール内のスタイルシート・インスタンスが削除され、リソースが解放される前に、経過する必要がある非活動時間（秒）を設定します。 デフォルト値は 60 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/connection-pool/initial XSQL Page Processor のデフォルトの Connection Manager は、接続プーリングを実装して、スループットを改善します。この設定は、各接続プールに対する JDBC 接続の初期割当て数を制御します。 デフォルト値は 2 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/connection-pool/increment サーバー上でのロードが増加したため接続プールを拡大する必要がある場合に割り当てる接続の数を設定します。 デフォルト値は 1 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/connection-pool/timeout-seconds プールが縮小して初期サイズに戻ろうとするときに、プール内の JDBC 接続が削除され、リソースが解放される前に、経過する必要がある非活動時間（秒）を設定します。 デフォルト値は 60 です。有効値は 0（ゼロ）以外の正の整数です。
XSQLConfig/processor/connection-pool/dump-allowed dump-pool=y パラメータをページ要求で渡すことによって接続プール・アクティビティの診断レポートを要求できるかどうかを決定します。 デフォルト値は no です。有効値は、yes または no です。

表 10-15 XSQLConfig.xml の構成設定（続き）

構成設定名
XSQLConfig/processor/connection-manager/factory XSQL Connection Manager ファクトリ実装の完全修飾 Java クラス名を指定します。値を指定しない場合は、デフォルトで oracle.xml.xsql.XSQLConnectionManagerFactoryImpl に設定されます。 デフォルト値は oracle.xml.xsql.XSQLConnectionManagerFactoryImpl です。有効値は、oracle.xml.xsql.XSQLConnectionManagerFactory インタフェースを実装するすべてのクラス名です。
XSQLConfig/processor/timing/page XSQL Page Processor が xsql-timing 属性を、ページの処理に必要な経過時間（秒）を報告する値を持つ、ページのドキュメント要素に追加するかどうかを決定します。 デフォルト値は no です。有効値は、yes または no です。
XSQLConfig/processor/timing/action XSQL Page Processor がページ内の、アクションの処理に必要な経過時間（秒）を報告する内容を持つアクション・ハンドラ要素のすぐ前にコメントを追加するかどうかを決定します。 デフォルト値は no です。有効値は、yes または no です。
XSQLConfig/processor/security/stylesheet/defaults/allow-client-style アプリケーションの開発中は、要求で特殊な xml-stylesheet パラメータの値を指定し、要求ごとにスタイルシートをオーバーライドする XSQL Page Processor の機能を使用すると有効である場合がよくあります。最も一般的な使用方法の 1 つは、xml-stylesheet=none の組合せを指定してスタイルシートのアプリケーションを一時的に使用禁止にし、デバッグ操作のために未処理の XSQL データ・ページを参照することです。 開発が完了すると、allow-client-style="no" 属性を各 XSQL ページのドキュメント要素に明示的に追加して、クライアントが本番アプリケーションでスタイルシートをオーバーライドしないようにできます。ただし、この構成設定を使用すると、allow-client-style のデフォルト動作を 1 つの場所でグローバルに変更できます。 この構成設定は、この動作のデフォルト設定のみを指定することに注意してください。特定の XSQL ページのドキュメント要素に対して allow-client-style="yes|no" 属性を明示的に指定すると、その値がこのグローバル・デフォルト値より優先されます。 有効値は、yes および no です。

表 10-15 XSQLConfig.xml の構成設定（続き）

構成設定名
XSQLConfig/processor/security/stylesheet/trusted-hosts/host XSLT スタイルシートは、拡張関数をコールできます。特に、XSQL Page Processor がすべての XSLT スタイルシートを処理するために使用する Oracle XSLT Processor は、Java 拡張関数をサポートします。通常、XSQL ページは相対 URL を使用して XSLT スタイルシートを参照します。XSQL Page Processor は、処理される XSLT スタイルシートへの絶対 URL が構成ファイルに示されているトラステッド・ホストからのものである必要があることを規定します。 <trusted-hosts> 要素内で任意の数の <host> 要素を指定できます。デフォルトでは、ローカル・マシン名、localhost および 127.0.0.1 がトラステッド・ホストとみなされます。 有効値は、すべてのホスト名または IP アドレスです。
XSQLConfig/http/proxyhost HTTP プロトコル・スキームを使用する URL を処理する場合に使用する HTTP プロキシ・サーバーの名前を設定します。 有効値は、すべてのホスト名または IP アドレスです。
XSQLConfig/http/proxyport HTTP プロトコル・スキームを使用する URL を処理する場合に使用する HTTP プロキシ・サーバーのポート番号を設定します。 有効値は 0（ゼロ）以外の整数です。
XSQLConfig/connectiondefs/connection XSQL Page Processor が使用する名前付き接続のニックネームおよび JDBC 接続の詳細を定義します。 <connectiondefs> には、任意の数の <connection> 子要素を指定できます。各接続定義には name 属性を指定する必要があります。また、適切な子要素 <username>、<password>、<driver>、<dburl> および <autocommit> を指定できます。
XSQLConfig/connectiondefs/connection/username 現行の接続のユーザー名を定義します。
XSQLConfig/connectiondefs/connection/password 現行の接続のパスワードを定義します。
XSQLConfig/connectiondefs/connection/dburl 現行の接続の JDBC 接続 URL を定義します。
XSQLConfig/connectiondefs/connection/driver 現行の接続に使用する JDBC ドライバの完全修飾 Java クラス名を指定します。値を指定しない場合は、デフォルトで oracle.jdbc.driver.OracleDriver に設定されます。

表 10-15 XSQLConfig.xml の構成設定（続き）

構成設定名
XSQLConfig/connectiondefs/connection/autocommit 現行の接続の自動コミット・フラグを明示的に設定します。値を指定しない場合、接続は JDBC ドライバの自動コミットのデフォルト設定を使用します。
XSQLConfig/serializerdefs/serializer 名前付きカスタム・シリアル化コードの実装を定義します。 <serializerdefs> には、任意の数の <serializer> 子要素を指定できます。各シリアル化コードには、<name> 子要素と <class> 子要素の両方を指定する必要があります。
XSQLConfig/serializerdefs/serializer/name 現行のカスタム・シリアル化コード定義の名前を定義します。
XSQLConfig/connectiondefs/connection/class 現行のカスタム・シリアル化コードの完全修飾 Java クラス名を指定します。このクラスは、oracle.xml.xsql.XSQLDocumentSerializer インタフェースを実装する必要があります。

FOP シリアル化コードを使用した PDF 出力の生成

XSQL ページ・フレームワークによるカスタム・シリアル化コードのサポートを使用すると、Apache の FOP プロセッサ（http://xml.apache.org/fop を参照）と統合するための oracle.xml.xsql.serializers.XSQLFOPSerializer が提供されます。FOP プロセッサは、XSLFO（http://www.w3.org/TR/xsl を参照）を含む XML 文書から PDF ドキュメントをレンダリングします。

たとえば、次の XSLT スタイルシート EmpTableFO.xsl を想定します。

```
<!-- EmpTableFO.xsl -->
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xsl:version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- defines the layout master -->
  <fo:layout-master-set>
    <fo:simple-page-master master-name="first"
                           page-height="29.7cm"
                           page-width="21cm"
                           margin-top="1cm"
                           margin-bottom="2cm"
                           margin-left="2.5cm"
                           margin-right="2.5cm">
      <fo:region-body margin-top="3cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <!-- starts actual layout -->
  <fo:page-sequence master-name="first">
```

```
  <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="24pt" font-family="Garamond" line-height="24pt"
                space-after.optimum="3pt" font-weight="bold"
                start-indent="15pt">
        Total of All Salaries is $<xsl:value-of select="sum(/ROWSET/ROW/SAL)"/>
      </fo:block>
      <!-- Here starts the table -->
      <fo:block border-width="2pt">
        <fo:table>
          <fo:table-column column-width="4cm"/>
          <fo:table-column column-width="4cm"/>
          <fo:table-body font-size="10pt" font-family="sans-serif">
            <xsl:for-each select="ROWSET/ROW">
              <fo:table-row line-height="12pt">
                <fo:table-cell>
                  <fo:block><xsl:value-of select="ENAME"/></fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block><xsl:value-of select="SAL"/></fo:block>
                </fo:table-cell>
              </fo:table-row>
            </xsl:for-each>
          </fo:table-body>
        </fo:table>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

指定された FOP シリアル化コード（XSQLConfig.xml で事前定義済）を次のような XSQL ページで使用すると、EMP 表に対する問合せの結果をフォーマットできます。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emptablefo.xsl" serializer="FOP"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT ENAME, SAL FROM EMP
    ORDER BY SAL asc
</xsql:query>
```

注意： XSQL FOP シリアル化コードを使用するには、サーバーの CLASSPATH に次の Java アーカイブを追加する必要があります。

- `xsqlserializers.jar` **- Oracle XSQL に付属の Java アーカイブ**
- `fop.jar` **- Apache バージョン 0.16 以上の Java アーカイブ**
- `w3c.jar` **- FOP 配布パッケージの** `./lib` **ディレクトリに含まれる Java アーカイブ**

XSQL Page Processor のプログラム的な使用

`XSQLRequest` クラスを使用すると、XSQL Page Processor Engine を独自のカスタム Java プログラム内から使用できます。API は簡単に使用できます。`XSQLRequest` のインスタンスを作成し、処理する XSQL ページを次のいずれかのオブジェクトとしてコンストラクタに渡します。

- ページへの URL を含む文字列
- ページの URL オブジェクト
- メモリー内の XML 文書

その後、次のメソッドのどちらかをコールして、ページを処理します。

- `process()` - 結果を PrintWriter または OutputStream に書き込む場合
- `processToXML()` - 結果を XML 文書として戻す場合

`XSQLConfig.xml` ベースの接続定義に基づいて JDBC 接続プーリングを実装する組込み XSQL Connection Manager を使用する場合、コンストラクタに渡す必要があるものは XSQL ページのみです。また、独自の Connection Manager 実装を使用する必要がある場合は、オプションで `XSQLConnectionManagerFactory` インタフェース用のカスタム実装を渡すこともできます。

処理する XSQL ページをメモリー内の XML 文書として渡す機能を使用すると、有効な XSQL ページを動的に生成して必要な方法で処理し、その後、そのページを XSQL エンジンに渡して評価することができます。

ページの処理時に、次の 2 つの追加操作を要求の一部として行う必要がある場合があります。

- 一連のパラメータを要求に渡す操作

  この操作は、`Dictionary` インタフェースを実装する任意のオブジェクトを `process()` メソッドまたは `processToXML()` メソッドに渡すことによって実行できます。パラメータを含むハッシュ表を渡すことは、一般的な方法の 1 つです。

- ページが XML 文書をポストされた XML メッセージの本体として処理するように設定する操作

  この操作は、XSQLRequest オブジェクトに対して `setPostedDocument()` メソッドを使用して実行できます。

次に、`XSQLRequest` を使用したページ処理の単純な例を示します。

```
import oracle.xml.xsql.XSQLRequest;
import java.util.Hashtable;
import java.io.PrintWriter;
import java.net.URL;
public class XSQLRequestSample {
  public static void main( String[] args) throws Exception {
     // Construct the URL of the XSQL Page
   URL pageUrl = new URL("file:///C:/foo/bar.xsql");
   // Construct a new XSQL Page request
   XSQLRequest req = new XSQLRequest(pageUrl);
   // Setup a Hashtable of named parameters to pass to the request
   Hashtable params = new Hashtable(3);
   params.put("param1","value1");
   params.put("param2","value2");
   /* If needed, treat an existing, in-memory XMLDocument as if
   ** it were posted to the XSQL Page as part of the request
   req.setPostedDocument(myXMLDocument);
   **
   */
   // Process the page, passing the parameters and writing the output
   // to standard out.
   req.process(params,new PrintWriter(System.out)
                        ,new PrintWriter(System.err));
  }
}
```

カスタム XSQL アクション・ハンドラの作成

タスクの実行にカスタム処理が必要であり、そのニーズに合う組込みアクションがない場合は、すべての XSQL が使用できる独自のアクションを作成してレパートリに追加できます。

XSQL Page Processor のコアは、アクション・ハンドラ要素を含む XML 文書を処理するエンジンです。XSQL Page Processor Engine は、`XSQLActionHandler` インタフェースを実装するすべてのアクションをサポートするように作成されています。すべての組込みアクションは、このインタフェースを実装します。

XSQL Page Processor は、ページ内のアクションを次の方法で処理します。XSQL Page Processor は、ページ内のアクションごとに、次の手順を実行します。

1. デフォルトのコンストラクタを使用して、アクション・ハンドラ・クラスのインスタンスを作成します。
2. 次のメソッドをコールして、アクション・ハンドラ要素オブジェクトおよび XSQL Page Processor コンテキストを含むハンドラ・インスタンスを初期化します。

```
init(Element actionElt,XSQLPageRequest context)
```

3. 次のメソッドを起動して、ハンドラがアクションを処理できるようにします。

```
handleAction (Node result)
```

組込みアクションの場合、XSQL Page Processor Engine は、そのアクション・ハンドラを実装する Java クラスへの XSQL アクション・ハンドラ要素名のマッピングを認識します。表 10-16 に、そのマッピングを示します。ユーザー定義のアクションの場合は、次の組込みアクションを使用します。

```
<xsql:action handler="fully.qualified.Classname" ... />
```

このアクションの handler 属性は、そのカスタム・アクション・ハンドラを実装する Java クラスの完全修飾名を指定します。

表 10-16 組込み XSQL アクション・ハンドラ要素およびアクション・ハンドラ・クラス

XSQL アクション・ハンドラ要素	oracle.xml.xsql.actions のハンドラ・クラス
<xsql:query>	XSQLQueryHandler
<xsql:dml>	XSQLDMLHandler
<xsql:set-stylesheet-param>	XSQLStylesheetParameterHandler
<xsql:insert-request>	XSQLInsertRequestHandler
<xsql:include-xml>	XSQLIncludeXMLHandler
<xsql:include-request-params>	XSQLIncludeRequestHandler
<xsql:include-xsql>	XSQLIncludeXSQLHandler
<xsql:include-owa>	XSQLIncludeOWAHandler
<xsql:action>	XSQLExtensionActionHandler
<xsql:ref-cursor-function>	XSQLRefCursorFunctionHandler
<xsql:include-param>	XSQLGetParameterHandler

表 10-16 組込み XSQL アクション・ハンドラ要素およびアクション・ハンドラ・クラス（続き）

XSQL アクション・ハンドラ要素	oracle.xml.xsql.actions のハンドラ・クラス
<xsql:set-session-param>	XSQLSetSessionParamHandler
<xsql:set-page-param>	XSQLSetPageParamHandler
<xsql:set-cookie>	XSQLSetCookieHandler
<xsql:insert-param>	XSQLInsertParameterHandler
<xsql:update-request>	XSQLUpdateRequestHandler
<xsql:delete-request>	XSQLDeleteRequestHandler

独自のアクション・ハンドラの作成

カスタム・アクション・ハンドラを作成するには、oracle.xml.xsql.XSQLActionHandler インタフェースを実装するクラスを作成する必要があります。ほぼすべてのカスタム・アクション・ハンドラは、init() メソッドのデフォルト実装および非常に有効な一連のヘルパー・メソッドを提供する oracle.xml.xsql.XSQLActionHandlerImpl を拡張する必要があります。

XSQL Page Processor がアクション・ハンドラの handleAction メソッドをコールすると、アクション実装が DOM 文書フラグメントのルート・ノードに渡されます。アクション・ハンドラは、ページに戻す必要がある動的に作成された任意の XML コンテンツをこのドキュメント・フラグメントに追加します。

XSQL Page Processor は、概念的に XSQL ページ・テンプレート内のアクション・ハンドラ要素をこのドキュメント・フラグメントの内容に置き換えます。ページに追加する XML コンテンツがない場合、アクション・ハンドラはこのドキュメント・フラグメントに何も追加しません。

カスタム・アクション・ハンドラの作成中に、XSQLActionHandlerImpl クラスに対していくつかのメソッドを使用すると、操作が簡単になります。表 10-17 に、有効なメソッドを示します。

表 10-17 oracle.xml.xsql.SQLActionHandlerImpl に対する有効なメソッド

メソッド名	説明
`getActionElement`	処理中の現行のアクション・ハンドラ要素を戻します。
`getActionElementContent`	すべての字句パラメータを適切に置き換えて、現行のアクション・ハンドラ要素のテキスト内容を戻します。
`getPageRequest`	現行の XSQL Page Processor コンテキストを戻します。このオブジェクトを使用すると、次のような操作を実行できます。 ■ `setPageParam()` ページ・パラメータ値を設定します。 ■ `getPostedDocument()/setPostedDocument()` ポストされた XML 文書を取得または設定します。 ■ `translateURL()` 相対 URL を絶対 URL に変換します。 ■ `getRequestObject()/setRequestObject()` 単一のページのアクション間で共有できるページ要求コンテキスト内のオブジェクトを取得または設定します。 ■ `getJDBCConnection()` このページが使用中の JDBC 接続（使用中の接続がない場合は NULL）を取得します。 ■ `getRequestType()` Servlet、Command Line または Programmatic のいずれのコンテキストで実行中かを検出します。たとえば、要求のタイプが Servlet の場合は、`XSQLPageRequest` オブジェクトをより固有の `XSQLServletPageRequest` にキャストし、`getHttpServletRequest`、`getHttpServletResponse`、`getServletContext` などの他の Servlet 固有のメソッドにアクセスできます。

表 10-17 oracle.xml.xsql.SQLActionHandlerImpl に対する有効なメソッド（続き）

メソッド名	説明
getAttributeAllowingParam	属性値で使用されているすべての XSQL 字句パラメータ参照を解決し、要素から属性値を取り出します。通常、このメソッドはアクション・ハンドラ要素自体に適用されますが、そのサブ要素の属性にアクセスすることも有効です。字句パラメータを許可せずに属性値にアクセスするには、DOM の Element インタフェースに対して標準の getAttribute() メソッドを使用します。
appendSecondaryDocument	外部 XML 文書のコンテンツ全体をアクション・ハンドラの結果内容のルートに追加します。
addResultElement	テキスト内容を含む単一の要素をアクション・ハンドラの結果内容のルートに追加する操作を簡単にします。
firstColumnOfFirstRow	渡された SQL 文の最初の行にある最初の列値を戻します。このメソッドを使用するには、現行のページのドキュメント要素に connection 属性が含まれている必要があります。含まれていない場合は、エラーが戻されます。
bindVariableCount	スペースで区切られた bind-params のリスト内にあるトークンの数を戻し、パラメータにバインドされるバインド変数の数を示します。
handleBindVariables	現行のアクション・ハンドラ要素の bind-params 属性で指定されるパラメータ値を使用して、プリコンパイルされた SQL 文で使用される JDBC バインド変数のバインディングを管理します。このメソッドをコールする前に SQL 文がすでに多くのバインド変数を使用している場合は、使用中である既存のバインド変数スロットの数も渡すことができます。
reportErrorIncludingStatement	問題の原因である違反（SQL）文を含むエラーを報告します。オプションで、エラーに数値エラー・コードが含まれる場合もあります。
reportFatalError	致命的エラーを報告します。
reportMissingAttribute	標準の <xsql-error> 要素を使用して、必須のアクション・ハンドラ属性が欠落していることを示すエラーを報告します。
reportStatus	標準の <xsql-status> 要素を使用して、アクション・ハンドラの状態を報告します。

表 10-17 oracle.xml.xsql.SQLActionHandlerImpl に対する有効なメソッド（続き）

メソッド名	説明
requiredConnectionProvided	この要求に使用できる接続の有無を確認し、使用可能な接続がない場合はページにエラーグラムを出力します。
variableValue	字句パラメータのデフォルト値を決定する場合があるすべての有効範囲決定規則を考慮して、字句パラメータの値を戻します。

次の例は、カスタム・アクション・ハンドラ MyIncludeXSQLHandler を示します。このハンドラは、組込みアクション・ハンドラの 1 つおよび任意の Java コードを使用し、そのハンドラによって戻された XML のドキュメント・フラグメントを変更してから、その結果を XSQL ページに追加します。

```
import oracle.xml.xsql.*;
import oracle.xml.xsql.actions.XSQLIncludeXSQLHandler;
import org.w3c.dom.*;
import java.sql.SQLException;
public class MyIncludeXSQLHandler extends XSQLActionHandlerImpl {
  XSQLActionHandler nestedHandler = null;
  public void init(XSQLPageRequest req, Element action) {
    super.init(req, action);
    // Create an instance of an XSQLIncludeXSQLHandler
    // and init() the handler by passing the current request/action
    // This assumes the XSQLIncludeXSQLHandler will pick up its
    // href="xxx.xsql" attribute from the current action element.
    nestedHandler = new XSQLIncludeXSQLHandler();
    nestedHandler.init(req,action);
  }
  public void handleAction(Node result) throws SQLException {
    DocumentFragment df=result.getOwnerDocument().createDocumentFragment();
    nestedHandler.handleAction(df);
    // Custom Java code here can work on the returned document fragment
    // before appending the final, modified document to the result node.
    // For example, add an attribute to the first child
    Element e = (Element)df.getFirstChild();
    if (e != null) {
      e.setAttribute("ExtraAttribute","SomeValue");
    }
    result.appendChild(df);
  }
}
```

ページが XSQL Servlet、XSQL コマンドライン・ユーティリティまたはプログラムで XSQLRequest クラスを介して要求されているかどうかに基づいて、異なる動作をする必要があるカスタム・アクション・ハンドラを作成する場合、アクション・ハンドラの実装で、getPageRequest() をコールして、現行のページ要求用の XSQLPageRequest インタフェースを参照できます。XSQLPageRequest オブジェクトに対して getRequestType() をコールすると、それぞれ Servlet、Command Line または Programmatic によるルートから要求を受信しているかどうかを確認できます。戻り値が Servlet の場合、次の構文によって、HTTP Servlet の要求、応答およびサーブレット・コンテキスト・オブジェクトにアクセスできます。

```
XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
if (xspr.getRequestType().equals("Servlet")) {
  HttpServletRequest     req  = xspr.getHttpServletRequest();
  HttpServletResponse   resp  = xspr.getHttpServletResponse();
  ServletContext        cont  = xspr.getServletContext();
  // do something fun here with req, resp, or cont however
  // writing to the response directly from a handler will
  // produce unexpected results. Allow the XSQL Servlet
  // or your custom Serializer to write to the servlet's
  // response output stream at the write moment later when all
  // action elements have been processed.
}
```

カスタム XSQL シリアル化コードの作成

ユーザー定義のシリアル化コード・クラスを作成し、最終 XSQL データページの XML 文書をテキストまたはバイナリ・ストリームにシリアル化する方法をプログラムで制御できます。ユーザー定義のシリアル化コードは、単一のメソッドを導出する oracle.xml.xsql.XSQLDocumentSerializer インタフェースを実装する必要があります。

```
void serialize(org.w3c.dom.Document doc, XSQLPageRequest env) throws Throwable;
```

今回のリリースでは、DOM ベースのシリアル化コードがサポートされます。将来のリリースでは、SAX2 ベースのシリアル化コードもサポートされる可能性があります。カスタム・シリアル化コード・クラスを使用する場合は、次のタスクを適切な順序で実行する必要があります。

1. 出力 PrintWriter（または OutputStream）に内容を出力する前に、シリアル化されたストリームの内容の型を設定します。

   シリアル化コードに渡された XSQLPageRequest に対して setContentType() をコールし、型を設定します。内容の型を設定する場合は、次のような MIME タイプのみを設定することができます。

   ```
   env.setContentType("text/html");
   ```

または、次のようにエンコーディング・キャラクタ・セットを含む MIME タイプを設定することもできます。

```
env.setContentType("text/html;charset=Shift_JIS");
```

2. XSQLPageRequest に対して getWriter() または getOutputStream() のどちらかをコールし（両方は不可）、それぞれ適切な PrintWriter または OutputStream を取得して、内容のシリアル化に使用します。

たとえば、次のカスタム・シリアル化コードは、現行の XSQL データ・ページのドキュメント要素名を含む HTML ドキュメントをシリアル化する単純な実装を示します。

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.PrintWriter;
import oracle.xml.xsql.*;

public class XSQLSampleSerializer implements XSQLDocumentSerializer {
  public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    String encoding = env.getPageEncoding();  // Use same encoding as XSQL page
                                              // template. Set to specific
                                              // encoding if necessary
    String mimeType = "text/html"; // Set this to the appropriate content type
    // (1) Set content type using the setContentType on the XSQLPageRequest
    if (encoding != null && !encoding.equals("")) {
      env.setContentType(mimeType+";charset="+encoding);
    }
    else {
      env.setContentType(mimeType);
    }
    // (2) Get the output writer from the XSQLPageRequest
    PrintWriter e = env.getWriter();
    // (3) Serialize the document to the writer
    e.println("<html>Document element is <b>"+
              doc.getDocumentElement().getNodeName()+
              "</b></html>");
  }
}
```

カスタム・シリアル化コードを使用するには、シリアル化の前に XSLT 変換を実行する必要があるかどうかによって、2 つの方法があります。カスタム・シリアル化コードを使用する前に XSLT 変換を実行するには、次のとおり、ページの最上部にある <?xml-stylesheet?> 処理命令内に serializer="java:*fully.qualified.ClassName*" を追加します。

```
<?xml version="1.0?>
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"
                 serializer="java:my.pkg.MySerializer"?>
```

カスタム・シリアル化コードのみが必要である場合は、次のとおり、type 属性および href 属性を省略します。

```
<?xml version="1.0?>
<?xml-stylesheet serializer="java:my.pkg.MySerializer"?>
```

また、XSQLConfig.xml ファイルの <serializerdefs> セクションでカスタム・シリアル化コードに短いニックネームを割り当て、serializer 属性でそのニックネーム（大文字 / 子文字を区別）を使用して、入力作業を省くことができます。たとえば、XSQLConfig.xml の内容が次のとおりであると想定します。

```
<XSQLConfig>
  <!-- etc. -->
  <serializerdefs>
    <serializer>
      <name>Sample</name>
      <class>oracle.xml.xsql.serializers.XSQLSampleSerializer</class>
    </serializer>
    <serializer>
      <name>FOP</name>
      <class>oracle.xml.xsql.serializers.XSQLFOPSerializer</class>
    </serializer>
  </serializerdefs>
</XSQLConfig>
```

この場合は、次の例で示すとおり、ニックネームの Sample または FOP（あるいはその両方）を使用できます。

```
<?xml-stylesheet type="text/xsl" href="emp-to-xslfo.xsl" serializer="FOP"?>
```

または

```
<?xml-stylesheet serializer="Sample"?>
```

XSQLPageRequest インタフェースは、getWriter() メソッドおよび getOutputStream() メソッドの両方をサポートします。カスタム・シリアル化コードは、getOutputStream() をコールして、OutputStream インスタンスを戻します。動的に生成された GIF イメージなどのバイナリ・データは、このインスタンスにシリアル化することができます。XSQL Servlet を使用する場合は、この出力ストリームに出力すると、バイナリ情報が Servlet の出力ストリームに出力されます。

たとえば、次のシリアル化コードは、動的 GIF イメージを出力する例を示します。この例では、GIF イメージは小さい静的な「OK」アイコンですが、より高度なイメージ・シリアル化コードに使用する基本的な方法を示します。

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.*;
import oracle.xml.xsql.*;

public class XSQLSampleImageSerializer implements XSQLDocumentSerializer {
   // Byte array representing a small "ok" GIF image
   private static byte[] okGif =
     {(byte)0x47,(byte)0x49,(byte)0x46,(byte)0x38,
      (byte)0x39,(byte)0x61,(byte)0xB,(byte)0x0,
      (byte)0x9,(byte)0x0,(byte)0xFFFFFF80,(byte)0x0,
      (byte)0x0,(byte)0x0,(byte)0x0,(byte)0x0,
      (byte)0xFFFFFFFF,(byte)0xFFFFFFFF,(byte)0xFFFFFFFF,(byte)0x2C,
      (byte)0x0,(byte)0x0,(byte)0x0,(byte)0x0,
      (byte)0xB,(byte)0x0,(byte)0x9,(byte)0x0,
      (byte)0x0,(byte)0x2,(byte)0x14,(byte)0xFFFFFF8C,
      (byte)0xF,(byte)0xFFFFFFA7,(byte)0xFFFFFFB8,(byte)0xFFFFFF9B,
      (byte)0xA,(byte)0xFFFFFFA2,(byte)0x79,(byte)0xFFFFFFE9,
      (byte)0xFFFFFF85,(byte)0x7A,(byte)0x27,(byte)0xFFFFFF93,
      (byte)0x5A,(byte)0xFFFFFFE3,(byte)0xFFFFFFEC,(byte)0x75,
      (byte)0x11,(byte)0xFFFFFF85,(byte)0x14,(byte)0x0,
      (byte)0x3B};

  public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    env.setContentType("image/gif");
    OutputStream os = env.getOutputStream();
    os.write(okGif,0,okGif.length);
    os.flush();
  }
}
```

XSQL コマンドライン・ユーティリティを使用する場合は、バイナリ情報がターゲット出力ファイルに出力されます。XSQLRequest プログラム API を使用する場合は、2 つのコンストラクタが存在します。これらのコンストラクタは、コール元がページ処理の結果用に使用するターゲット OutputStream を提供できるようにします。

シリアル化コードは、`getWriter()`（テキスト出力用）または `getOutputStream()`（バイナリ出力用）のどちらかをコールする必要があります。ただし、これらの両方をコールすることはできません。同一の要求で両方のメソッドをコールすると、エラーが発生します。

カスタム XSQL Connection Manager の作成

カスタム Connection Manager を作成し、組込み Connection Management メカニズムを置き換えることができます。カスタム Connection Manager の実装を提供するには、次のオブジェクトを提供する必要があります。

1. `oracle.xml.xsql.XSQLConnectionManagerFactory` インタフェースを実装する Connection Manager ファクトリ・オブジェクト
2. `oracle.xml.xsql.XSQLConnectionManager` インタフェースを実装する Connection Manager オブジェクト

`XSQLConfig.xml` ファイルの次のセクションにクラス名を指定すると、カスタム Connection Manager ファクトリをデフォルトの Connection Manager ファクトリとして使用するように設定できます。

```
<!--
 | Set the name of the XSQL Connection Manager Factory
 | implementation. The class must implement the
 | oracle.xml.xsql.XSQLConnectionManagerFactory interface.
 | If unset, the default is to use the built-in connection
 | manager implementation in
 | oracle.xml.xsql.XSQLConnectionManagerFactoryImpl
 +-->
<connection-manager>
  <factory>oracle.xml.xsql.XSQLConnectionManagerFactoryImpl</factory>
</connection-manager>
```

デフォルトの Connection Manager ファクトリを指定する他に、指定された API を使用して、カスタム接続ファクトリを個々の `XSQLRequest` オブジェクトに対応付けることもできます。

XSQLConnectionManagerFactory は、現行の要求が使用する XSQLConnectionManager のインスタンスを戻します。サーブレット・エンジンなどのマルチスレッド環境では、XSQLConnectionManager オブジェクトが、2 つの異なるスレッドによって単一の XSQLConnection インスタンスが使用されていないことを保証する役割を担います。これは、getConnection() メソッドの起動から releaseConnection() メソッドの起動までの期間中、接続を in use（使用中）としてマークすることによって、保証されます。デフォルトの XSQL Connection Manager 実装は、名前付き接続を自動的にプールし、このスレッド・セーフ・ポリシーに従います。

XSQL アクション・ハンドラ・エラーのフォーマット

XSQL アクション・ハンドラ要素の処理によって発生したエラーは、XSL スタイルシートがエラーの存在を検出し、オプションでそれらをフォーマットして表示できるように、XML 要素として決まった方法で通知されます。

エラーが発生したアクション・ハンドラ要素は、ページ内で次の構文によって置き換えられます。

```
<xsql-error action="xxx">
```

エラーによっては、<xsql-error> 要素に次のものが含まれます。

- ネストした <message> 要素
- 違反 SQL 文を含む <statement> 要素

エラー情報の画面表示

この項では、次の情報を使用してエラー情報を画面上に表示する XSLT スタイルシートの例を示します。

```
<xsl:if test="//xsql-error">
     <table style="background:yellow">
        <xsl:for-each select="//xsql-error">
           <tr>
            <td><b>Action</b></td>
            <td><xsl:value-of select="@action"/></td>
            </tr>
            <tr valign="top">
            <td><b>Message</b></td>
            <td><xsl:value-of select="message"/></td>
           </tr>
         </xsl:for-each>
     </table>
</xsl:if>
```

XSQL Servlet の制限事項

XSQL Servlet には、次の制限事項があります。

マルチバイト名を持つ HTTP パラメータ

マルチバイト名を持つ HTTP パラメータ（漢字表記の名前を持つパラメータなど）は、<xsql:include-request-params> を使用して XSQL ページに挿入すると、適切に処理されます。<xsql:query> タグの問合せ文内でマルチバイト名を持つパラメータの参照を試みると、パラメータの値に対して空の文字列が戻されます。

解決策

パラメータに非マルチバイト名を使用してください。このパラメータは、引続きマルチバイト値を持つことができ、このマルチバイト値は正しく処理できます。

SQL 文内の CURSOR() ファンクション

SQL 文で CURSOR() ファンクションを使用すると、CURSOR() 文がネストされ、問合せの最初の行が CURSOR() ファンクションに対して空の結果セットを戻した場合、「Exhausted Result Set」エラーが発生する場合があります。

FAQ: XSQL Servlet

XSQL 出力を WML ドキュメントに変換中に DTD を指定する方法

質問

XSQL 出力を WML および Vector Markup Language（VML）形式に変換するための独自のスタイルシートの作成を試みています。これらのプログラム（携帯電話シミュレータ）では、WML ドキュメントに特定の DTD を割り当てる必要があります。

XSQL 出力を WML ドキュメントに変換中に特定の DTD を指定する方法はありますか？

回答

指定する方法はあります。<xsl:output> という XSLT スタイルシートの組込み機能を使用します。

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output type="xml" doctype-system="your.dtd"/>
  <xsl:template match="/">
  </xsl:template>
     :
     :
</xsl:stylesheet>
```

これによって、XML 結果が生成されます。

```
<!DOCTYPE xxxx SYSTEM “your.dtd”>
```

この結果には、前述の宣言が含まれます。"your.dtd" には、任意の有効な絶対 URL または相対 URL を指定できます。

XSQL Servlet の条件文

質問

XSQL ファイルに条件文を記述できますか？できる場合、条件文を記述するための構文を教えてください。

次に例を示します。

```
<xsql:choose>
    <xsql:when test="@security='admin'">
      <xsql:query>
          SELECT ....
      </xsql:query>
   </xsq:when>
   <xsql:when test="@security='user'">
      <xsql:query>
          SELECT ....
      </xsql:query>
   </xsql:when>
</xsql:if>
```

回答

<xsql:ref-cursor-function> を使用し、条件付きで REF CURSOR を適切な問合せに戻す PL/SQL プロシージャをコールします。

問合せで取り出された値を他の問合せの WHERE 句で使用する方法

質問

次のとおり、XSQL ファイル内に 2 つの問合せがあります。

```
<xsql:query>
   select col1,col2
   from table1
</xsql:query>
<xsql:query>
   select col3,col4 from table2
   where col3 = {@col1}    => the value of col1 in the previous query
</xsql:query>
```

最初の問合せの SELECT 構文のリスト項目の値を 2 番目の問合せで使用する方法を教えてください。

回答

これは、次のとおりページ・パラメータを使用して行うことができます。

```
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
  <!-- Value of page param "xxx" will be first column of first row -->
  <xsql:set-page-param name="xxx">
    select one from table1 where ...
  </xsl:set-param-param>
  <xsql:query bind-params="xxx">
     select col3,col4 from table2
    where col3 = ?
  </xsql:query>
</page>
```

Oracle 以外のデータベースを使用する方法

質問

XSQL Servlet は JDBC をサポートするすべてのデータベースに接続できますか？

回答

接続できます。`XSQLConfig.xml` ファイルの接続定義に適切な JDBC ドライバ・クラスおよび接続 URL を指定します。ただし、オブジェクト・リレーショナル機能は、Oracle を Oracle JDBC ドライバとともに使用する場合にのみ機能します。

XSQL Servlet: JServ プロセスへのアクセス

質問

デモ helloworld.xsql を実行しています。最初の段階で、次のエラーが発生しました。

```
XSQL-00007 ページを処理するためのデータベース接続を取得できません。
```

要求がタイムアウトし、jserv/log/jserv.log ファイルに次のメッセージが現れます。

```
Connections from Localhost/127.0.0.1 are not allowed
```

これはセキュリティの問題でしょうか？XSQL ページを処理するための明示的な権限を付与する必要がありますか？必要な場合、その方法を教えてください。Apache Web サーバー、Apache JServ および Oracle をデータベースとして使用しています。Oracle クライアントをインストール済で、データベース接続を取得するための Tnsnames.ora ファイルも構成済です。XSQLConfig.xml ファイルも適切に構成されています。

回答

これは、一般的な JServ の問題のようです。jserv.properties の security.allowedAddresses=property が、現在のホストによる Java が実行している JServ プロセスへのアクセスを許可していることを確認する必要があります。任意の JServ Servlet を正常に実行できますか？

XSQL Servlet および Oracle8*i* Lite

質問

XSQL Servlet を Oracle8*i* Lite（Windows 98）および Apache Web サーバー /JServ とともに使用する予定です。CLASSPATH（POLJDBC ドライバを含む）に olite40.jar を設定していますが、エラー・メッセージ「no oljdbc40 in java.library.path」が表示されます。Oracle8*i* Lite に対して XSQL Servlet を実行するために必要な追加操作はありますか？

回答

次の命令を `jserv.properties` に挿入する必要があります。

```
wrapper.path=C:¥orant¥bin
```

この場合、`C:¥orant¥bin` は（デフォルトで）`OLJDBC40.DLL` が存在するディレクトリです。

これは `wrapper.classpath` ではなく `wrapper.path` であることに注意してください。

複数値の HTML フォーム・パラメータを処理する方法

質問

`<input type="checkbox">` に必要な複数値の HTML フォーム・パラメータを処理する方法はありますか？

回答

組み込まれた処理方法はありませんが、次のようなカスタム・アクション・ハンドラを使用できます。

```
// MultiValuedParam: XSQL Action Handler that takes the value of
// ----------------  a multi-valued HTTP request parameter and
//                   sets the value of a user-defined page-parameter
//                   equal to the concatenation of the multiple values
//                   with optional control over the separator used
```

```
//                  between values and delimiter used around values.
//                  Subsequent actions in the page can then reference
//                  the value of the user-defined page-parameter.
import oracle.xml.xsql.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
public class MultiValuedParam extends XSQLActionHandlerImpl {
  public void handleAction(Node root) {
    XSQLPageRequest req = getPageRequest();
    // Only bother to do this if we're in a Servlet environment
    if (req.getRequestType().equals("Servlet")) {
      Element      actElt = getActionElement();
      // Get name of multi-valued parameter to read from attribute
      String paramName = getAttributeAllowingParam("name",actElt);
      // Get name of page-param to set with resulting value
      String pageParam = getAttributeAllowingParam("page-param",actElt);
      // Get separator string
      String separator = getAttributeAllowingParam("separator",actElt);
      // Get delimiter string
      String delimiter = getAttributeAllowingParam("delimiter",actElt);
      // If the separator is not specified or is blank, use comma
      if (separator == null || separator.equals("")) {
        separator = ",";
      }
      // We're in a Servlet environment, so we can cast
      XSQLServletPageRequest spReq = (XSQLServletPageRequest)req;
      // Get hold of the HTTP Request
      HttpServletRequest httpReq = spReq.getHttpServletRequest();
      // Get the String array of parameter values
      String[] values = httpReq.getParameterValues(paramName);
      StringBuffer str = new StringBuffer();
      // If some values have been returned
      if (values != null) {
        int items = values.length;
        // Append each value to the string buffer
        for (int z = 0; z < items; z++) {
          // Add a separator before all but the first
          if (z != 0) str.append(separator);
          // Add a delimiter around the value if non-null
          if (delimiter != null) str.append(delimiter);
          str.append(values[z]);
          if (delimiter != null) str.append(delimiter);
        }
```

```
        // If page-param attribute not provided, default page param name
        if (pageParam == null) {
          pageParam = paramName+"-values";
        }
        // Set the page-param to the concatenated value
        req.setPageParam(pageParam,str.toString());
      }
    }
  }
}
```

このカスタム・アクションを次のようなページで使用できます。

```
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p1" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p2"
               delimiter="'" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p3"
               delimiter="&amp;quot;" separator=" " />
  <xsql:include-param name="p1"/>
  <xsql:include-param name="p2"/>
  <xsql:include-param name="p3"/>
</page>
```

このページが、複数値の属性を生成するために同じ名前の複数のパラメータを含む次の URL によって要求されたと想定します。

```
http://yourserver.com/page.xsql?guy=Curly&guy=Larry&guy=Moe
```

この場合、次のページが戻されます。

```
<page>
  <p1>Curly,Larry,Moe</p1>
  <p2>'Curly','Larry','Moe'</p2>
  <p3>"Curly" "Larry" "Moe"</p3>
</page>
```

ただし、次の構文によって、前述の複数値のページ・パラメータの値を SQL 文で使用することもできます。

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="list"
               delimiter="'" />
  <!-- Above custom action sets the value of page param named 'list' -->
  <xsql:query>
    SELECT * FROM sometable WHERE name IN ({@list})
  </xsql:query>
</page>
```

XSQL Servlet および Oracle7 リリース 7.3

質問

Oracle7 リリース 7.3 での XSQL Servlet の実行を妨げる要因はありますか？XSU は、クライアント側ユーティリティとして使用する場合、Oracle7 リリース 7.3 で使用できると理解しています。

回答

妨げる要因はありません。Oracle7 リリース 7.3 データベースに問題なく接続できる Oracle JDBC ドライバを使用していることを確認してください。

<xsql:dml> での OUT 変数の未サポート

質問

<xsql:dml> を使用して 1 つの OUT パラメータを含むストアド・プロシージャをコールしてみましたが、結果が戻りません。実行したコードは、結果として次の文を生成します。

```
<xsql-status action="xsql:dml" rows="0"/>
```

回答

今回のリリースでは、<xsql:dml> を使用して OUT 変数の位置にパラメータ値をバインドしても、パラメータ値を設定できません。バインディングは、IN パラメータに対してのみサポートされます。ユーザーは、HTP パッケージを使用して XML 要素を作成するラッパー・プロシージャを作成できます。XSQL ページは、かわりに <xsql:include-owa> を使用してそのラッパー・プロシージャを起動できます。

たとえば、次のプロシージャがあると想定します。

```
CREATE OR REPLACE PROCEDURE addmult(arg1        NUMBER,
                                    arg2        NUMBER,
                                    sumval  OUT NUMBER,
                                    prodval OUT NUMBER) IS
BEGIN
  sumval := arg1 + arg2;
  prodval := arg1 * arg2;
END;
```

次のプロシージャを作成してそれをラップし、前述のプロシージャに必要なすべての IN 引数を受け入れて、OWA ページ・バッファに出力する小規模な XML データグラムとして OUT 値をエンコーディングすることができます。

```
CREATE OR REPLACE PROCEDURE addmultwrapper(arg1 NUMBER, arg2 NUMBER) IS
  sumval  NUMBER;
  prodval NUMBER;
  xml     VARCHAR2(2000);
BEGIN
  -- Call the procedure with OUT values
  addmult(arg1,arg2,sumval,prodval);
  -- Then produce XML that encodes the OUT values
  xml := '<addmult>'||
         '<sum>'||sumval||'</sum>'||
         '<product>'||prodval||'</product>'||
         '</addmult>';
  -- Print the XML result to the OWA page buffer for return
  HTP.P(xml);
END;
```

これによって、ラッパー・プロシージャをコールする次のような XSQL ページを作成できます。

```
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:include-owa bind-params="arg1 arg2">
    BEGIN addmultwrapper(?,?); END;
  </xsql:include-owa>
</page>
```

これによって、次のような要求を実行できます。

```
http://yourserver.com/addmult.xsql?arg1=30&arg2=45
```

この要求は、次のような OUT 値を反映する XML データグラムを戻します。

```
<page>
  <addmult><sum>75</sum><product>1350</product></addmult>
</page>
```

接続不能エラー

質問

XSQL の使用を試みましたが、データベースに接続できず、helloworld.xsql の例を実行中に次のようなエラーが発生しました。

```
Oracle XSQL Servlet Page Processor 9.0.0.0.0 (Beta)
XSQL-00007: ページを処理するためのデータベース接続を取得できません。
Connection refused(DESCRIPTION=(TMP=)(VSNNUM=135286784)(ERR=12505)
(ERROR_STACK=(ERROR=(CODE=12505)(EMFI=4))))
```

これは、実際には構成ファイルが検索されていることを意味しますか？ユーザーは scott/tiger として設定済です。

回答

構成ファイルは検索されています。helloworld.xsql デモ・ページが変更されていないことを前提に、demo という名前の接続の <connectiondef> 情報に基づいて、JDBC 接続を実際に試みています。

XSQLConfig.xml ファイルには、デフォルトで次のような demo 接続のエントリが含まれています。

```
<connection name="demo">
  <username>scott</username>
  <password>tiger</password>
  <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
</connection>
```

したがって、次のことが原因でエラーが発生している可能性が高いといえます。

- データベースがローカル・ホスト・マシン上にない。
- データベースの SID が ORCL ではない。
- TNS リスナー・ポートが 1521 ではない。

これらの値がデータベースに適切であることを確認すると、正常に操作を行うことができます。

*.xsql 以外のファイル拡張子を使用する方法

質問

ユーザーには .html 拡張子を持つ HTML ファイルまたは .xml 拡張子を持つ XML ファイルにアクセスしているという印象を与えたまま、実際には XSQL を使用してユーザーに HTML および XML を表示する必要があります。XSQL Servlet は、デフォルトの .xsql 拡張子の他に、.html 拡張子または .xml 拡張子を持つファイルを認識できますか?

回答

認識できます。*.xsql 拡張子は絶対的なものではなく、単に XSQL ページを認識するために使用されるデフォルトの拡張子です。サーブレット・エンジンの構成設定を変更し、*.xsql 拡張子を対応付ける場合と同じ方法で、任意の拡張子を必要に応じて oracle.xml.xsql.XSQLServlet サーブレット・クラスに対応付けることができます。

XML 予約語を含む問合せのエラーを回避する方法

質問

次のようなページがあります。

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company,'&','and') company, balance
    FROM vendors
   WHERE outstanding_balance < 3000
</xsql:query>
```

ただし、このページを要求すると、エラーが発生します。

```
XSQL-00005: XSQL ページは正しいフォームではありません。
XML parse error at line 4, char 16
Expected name instead of '
```

何が間違っていますか?

回答

問題は、アンパサンド文字（&）および不等号（<）が次の理由から XML 内の予約語であることです。

-   や < などの実体参照を指定する一連の文字が & で始まること
- <SomeElement> などの要素を指定する一連の文字が < で始まること

リテラル・アンパサンド文字または不等号文字を挿入するには、各文字を次のような実体参照としてエンコードする必要があります。

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company,'&amp;','and') company, balance
    FROM vendors
   WHERE outstanding_balance &lt; 3000
</xsql:query>
```

または、一連の文字 <![CDATA[で始まり、対応する]]> で終わる CDATA セクションでテキストのブロック全体を囲むこともできます。CDATA セクションに含まれるすべてのテキストは、リテラルとして処理されます。

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
<![CDATA[
  SELECT id, REPLACE(company,'&','and') company, balance
    FROM vendors
   WHERE outstanding_balance < 3000
]]>
</xsql:query>
```

11

JDeveloper を使用した Oracle の XML アプリケーションの作成

この章の内容は次のとおりです。

JDeveloper9*i* の概要

Oracle JDeveloper9*i* は、E-Business アプリケーションの開発、デバッグおよび配置をエンド・ツー・エンドでサポートする J2EE 開発環境です。JDeveloper では、業界最速の Java デバッガ、新しいプロファイラ、コードのパフォーマンスを分析および改善する画期的な CodeCoach などの生産性の高いツールを使用して、より効率的に作業できます。

J2EE アプリケーションの開発の生産性をより向上するために、JDeveloper は BC4J を提供します。BC4J は、スケーラブルでパフォーマンスの高いインターネット・アプリケーションを作成するための、標準ベースのサーバー側フレームワークです。このフレームワークは、ビジネス・ロジックの構築および再利用を非常に簡単にする設計用機能および実行時サービスを提供します。

JDeveloper9*i* には、スキーマ駆動の新しい XML エディタがあります。図 11-1 を参照してください。XML 文書の構造を定義する XML Schema 仕様が、XML の検査や開発者による型付けの際にエディタで使用できます。この機能は Code Insight と呼ばれ、XML 文書内の要素や属性に対する有効な代替の要素や属性のリストを提供します。エディタでは、特定の言語に対してスキーマを指定するのみで、指定したマークアップ言語のドキュメントを効率的に作成できます。

Oracle JDeveloper9*i* によって、Java アプリケーション・コードと、XML データおよび XML 文書の同時使用が簡単になります。Oracle JDeveloper9*i* では、XML 開発モジュールをドラッグ・アンド・ドロップで使用できます。内容は次のとおりです。

- XML のカラー化された構文ハイライト表示
- XML および XSL 用の組込み構文確認
- XSQL Servlet のサポート

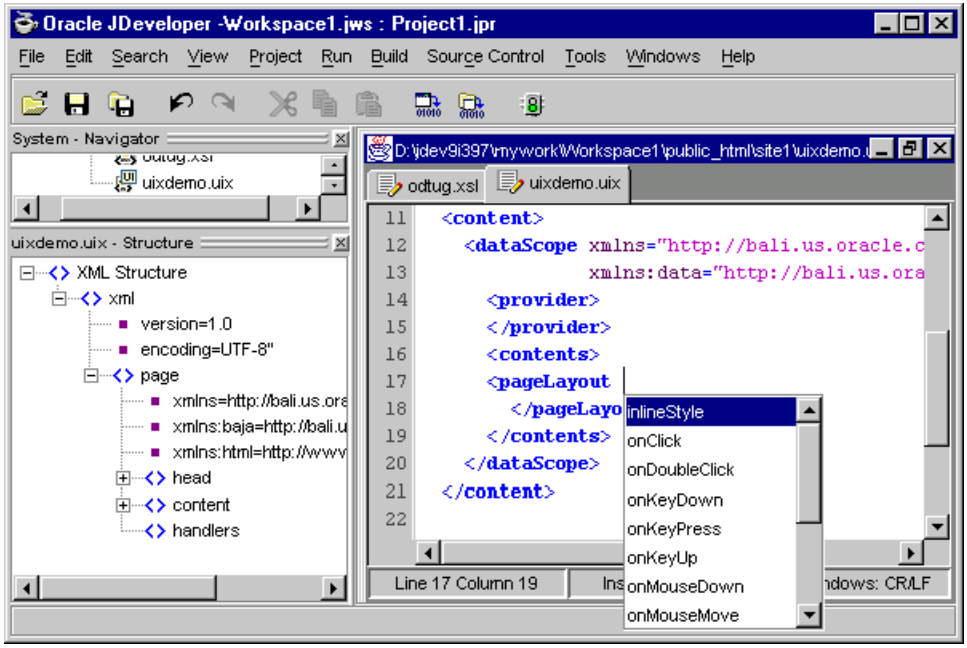
  開発者は、Oracle XSQL Servlet を編集およびデバッグしたり、コードを作成せずにデータベースに XML を挿入することができます。Oracle XSQL Servlet は、データベースに問い合せてフォーマットされた XML を戻すことができる Java プログラムです。統合されたサーブレット・エンジンによって、Java コードが生成した XML 出力を、ご使用のプログラム・ソースと同じ環境で参照できます。このため、迅速かつインタラクティブな開発およびテストが容易に行えます。
- Oracle XML Parser for Java
- XSLT プロセッサ
- XDK for JavaBeans の関連コンポーネント
- XSQL Page Wizard

  11-11 ページの「Page Selector Wizard」を参照してください。

- XSQL Element Wizard

  11-10 ページの「XSQL Element Wizard」を参照してください。
- XSQL アクション・ハンドラ
- スキーマ駆動 XML エディタ

図 11-1 動作中の JDeveloper9*i* スキーマ駆動 XML エディタ



JDeveloper に統合された Oracle XDK は、Java 開発者が XML を処理、作成および変換する際に有効な多くのユーティリティを提供します。たとえば、XSQL Servlet を使用して設計を行うと、データベースの情報を問合せおよび操作し、XML 文書を生成し、XSLT スタイルシートを使用してドキュメントを変換して Web 上で使用できます。

> **参照：** 第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

BC4J

BC4J は、Pure Java 対応で XML ベースのフレームワークです。このフレームワークによって、再利用可能なビジネス・コンポーネントから、複数層でデータベースを使用するアプリケーションの高生産性開発、移植可能な配置、および柔軟なカスタマイズが可能となります。

アプリケーション開発者は、Oracle Business Component フレームワークおよび Oracle JDeveloper の統合された設計時ウィザード、コンポーネント・エディタおよび生産性の高い Java 用コーディング環境を使用して、再利用可能なビジネス・コンポーネントからアプリケーション・サービスを作成およびテストします。

このようなアプリケーション・サービスは、CORBA サーバー・オブジェクトか EJB Session Beans のいずれかとして、Java テクノロジをサポートする企業規模のサーバー・プラットフォーム上に配置できます。

同じサーバー側のビジネス・コンポーネントは、JSP/Java Servlet アプリケーションまたは EJB のコンポーネントとして、変更せずに配置できます。このような柔軟な配置によって、開発者は同じビジネス・ロジックおよびデータ・モデルを再利用し、コードを再作成せずに様々なクライアント、ブラウザおよび無線インターネット・デバイスにアプリケーションを提供できます。

JDeveloper では、新しいビジュアル・ウィザードを使用して XML メタデータの記述を変更し、既存のビジネス・コンポーネントの機能をカスタマイズできます。

Oracle JDeveloper の XML 計画

Oracle JDeveloper9*i* は、XML アプリケーションの構築をサポートします。JDeveloper に統合されているスキーマ駆動の新しい XML コード・エディタでは、XML Schema ベースのドキュメントに対して次のような操作を実行できます。

- XML Schema の作成
- XSLT スタイルシートの作成

Tag Insight を使用すると、スキーマによって定義される正しい要素および属性を簡単に入力できます。JDeveloper の XML コード・エディタには、編集機能以外にも次の機能があります。

- エラーのハイライト
- プロパティの検査
- 構造ペイン内のツリー形式のビュー

JDeveloper の詳細情報

参照： 次の Web サイトおよびマニュアルを参照してください。

- http://otn.oracle.com/products/jdev/
- 『Oracle9*i* Java Developer's Guide』
- 『Oracle JavaServer Pages Developer's Guide and Reference』
- 『Oracle9*i* CORBA Developer's Guide and Reference』
- 『Oracle9*i* Enterprise JavaBeans Developer's Guide and Reference』
- 『Oracle9*i* Java Stored Procedures Developer's Guide』
- 『Oracle9*i* Java Tools Reference』
- 『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』
- 『Oracle9*i* JPublisher ユーザーズ・ガイド』
- 『Oracle9*i* Servlet Engine Developer's Guide』
- 『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』

JDeveloper9*i* の動作環境

JDeveloper9*i* は、Java で記述され、Windows NT、Windows 2000、Linux および Solaris オペレーティング・システムで動作する IDE です。JDeveloper9*i* を実行するには、128 MB 以上の RAM が必要です。

JDeveloper のシステム最低条件

『Oracle JDeveloper for Windows NT and Windows 2000 インストレーション・ガイド』を参照してください。同じマシンで動作する製品の数が増えると、システム要件は増加します。JDeveloper を実行するための一般的な開発環境は、次のとおりです。

- JDeveloper の実行
- ローカルでの Oracle の実行
- ローカルでの Oracle Application Server の実行
- その他のサード・パーティ・ツール（プロファイラ、バージョン・コントロール、モデラーなど）

実際の CPU 使用率および必要なディスク領域の点から、これらもシステム要件になります。

JDeveloper9*i* の入手方法

JDeveloper9*i* のベータ・リリースは、2001 年夏以降に OTN サイト（http://otn.oracle.com）から入手できます。

BC4J での XML

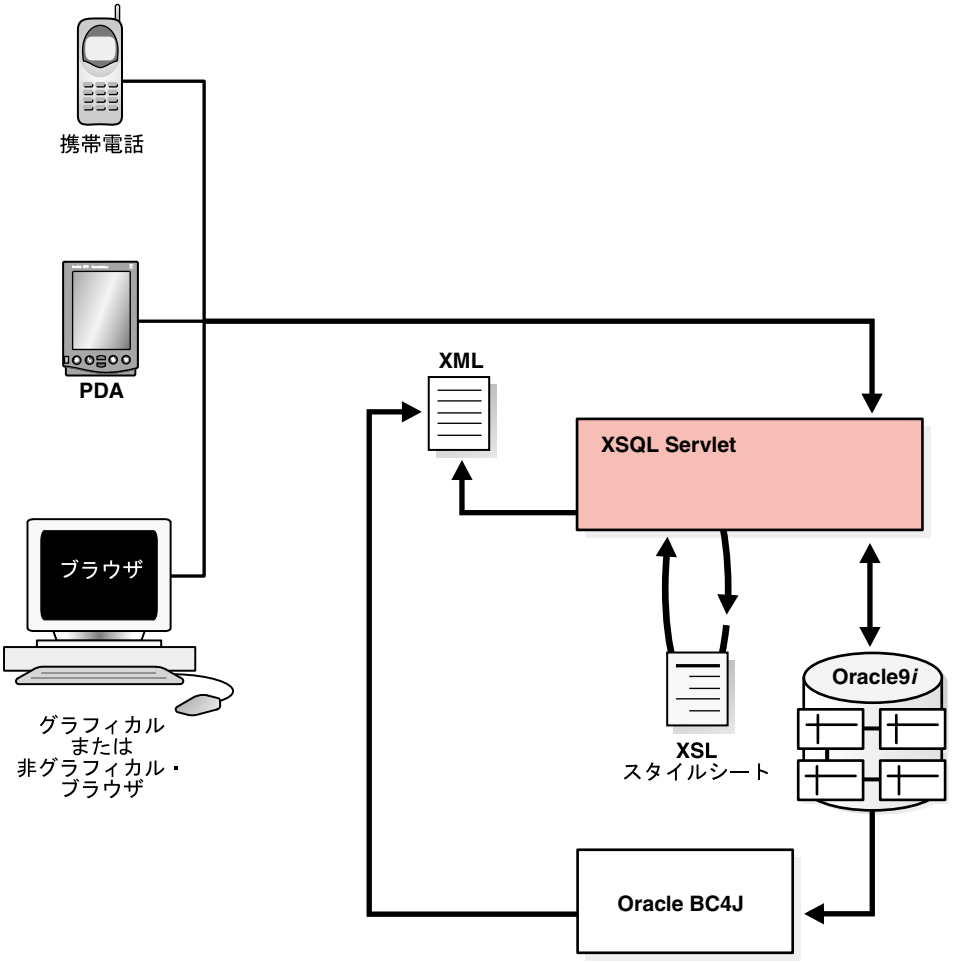
JDeveloper9*i* の BC4J フレームワークは、XML を使用して、オブジェクトの宣言設定および機能を表すメタデータを定義します。カスタムまたは複雑なビジネス・ロジックは、Java で実装できます。

- BC4J Tester を使用すると、View Object 内のデータを XML として参照できます。
- 検証規則などのビジネス・ルールは、Java ソース・コードではなく XML に格納されます。
- Java ソース・コードのかわりに XML を変更して、ビジネス・アプリケーションを容易にカスタマイズできます。
- ロジックを XML で抽象化することで、アプリケーションを簡単に読んだり、理解することができます。

BC4J による XML を使用したメタデータの格納　JDeveloper とともに提供される BC4J フレームワークは、XML を使用して、アプリケーションのコンポーネントに関するメタデータを格納します。重要な情報は、Java ソース・コードのかわりに構造化ドキュメントに格納されています。これによって、アプリケーションを簡単に理解およびカスタマイズできます。アプリケーションは、ソース・コードを変更せずにカスタマイズできます。図 11-2 に、XSQL Servlet が BC4J を使用して XML 文書を生成する方法を示します。

参照： http://otn.oracle.com/products/bc4j/ を参照してください。

図 11-2 BC4J の使用



ビジネス・ルールは、基礎となるコンポーネントのソース・コードへアクセスすることなく、その場で変更できます。

BC4J を使用した XSQL クライアントの構築

JDeveloper9*i* では、XSQL Servlet を構築できます。XSQL Servlet は、BC4J アプリケーション・モジュールと統合して、中間層から複数のクライアントへアプリケーション・ロジックを提供します。ユーザーは、対応するスタイルシートを適用するのみで、XML データを取り出し、すべてのクライアント・デバイスでデータを表示できます。

次の機能は、BC4J を使用した XSQL クライアントの構築に有効です。

- Object Gallery
- XSQL Element Wizard
- Page Selector Wizard

注意： JDeveloper9*i* の製品バージョンでは、これらの機能の表現が異なる場合があります。

参照： 第 12 章「BC4J および XML アプリケーションの作成」を参照してください。

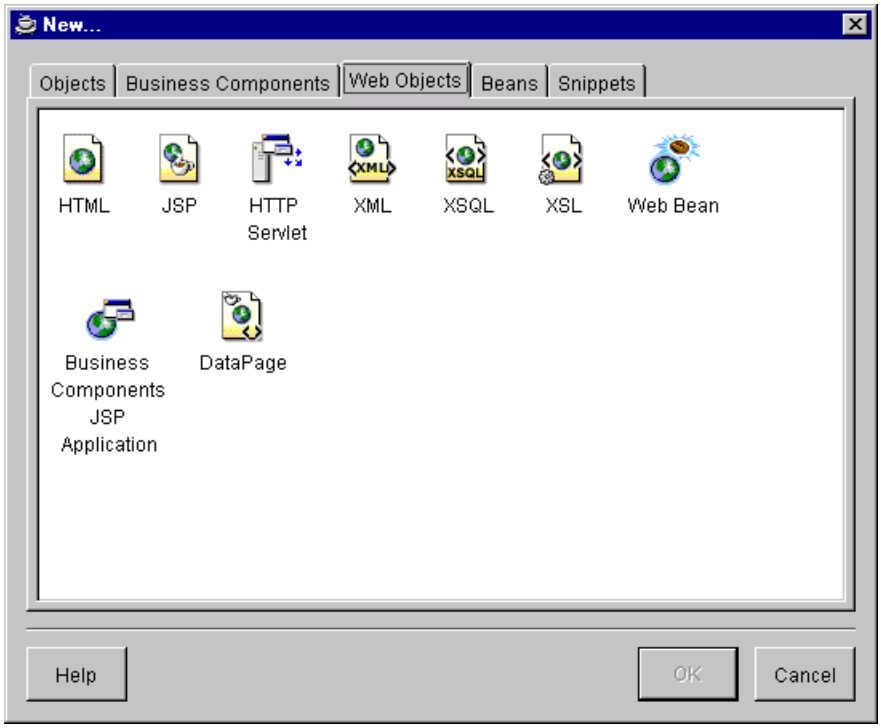
Object Gallery

Web Object Gallery には、XSQL、XML および XSL ドキュメントを簡単に作成するためのアイコンがあります。アイコンをクリックすると、これらのページの基本タグが生成されます。ユーザーは、これらのタグを拡張できます。

基本的な XSQL ページの生成後、XSQL Element Wizard を使用して、データにバインドされたタグを XSQL ページに挿入できるという点で、XSQL ページのアイコンは特に重要です。図 11-3 に、JDeveloper の Object Gallery を示します。

参照： 11-10 ページの「XSQL Element Wizard」を参照してください。

図 11-3 XSQL、XML および XSL アイコンがある JDeveloper の Object Gallery

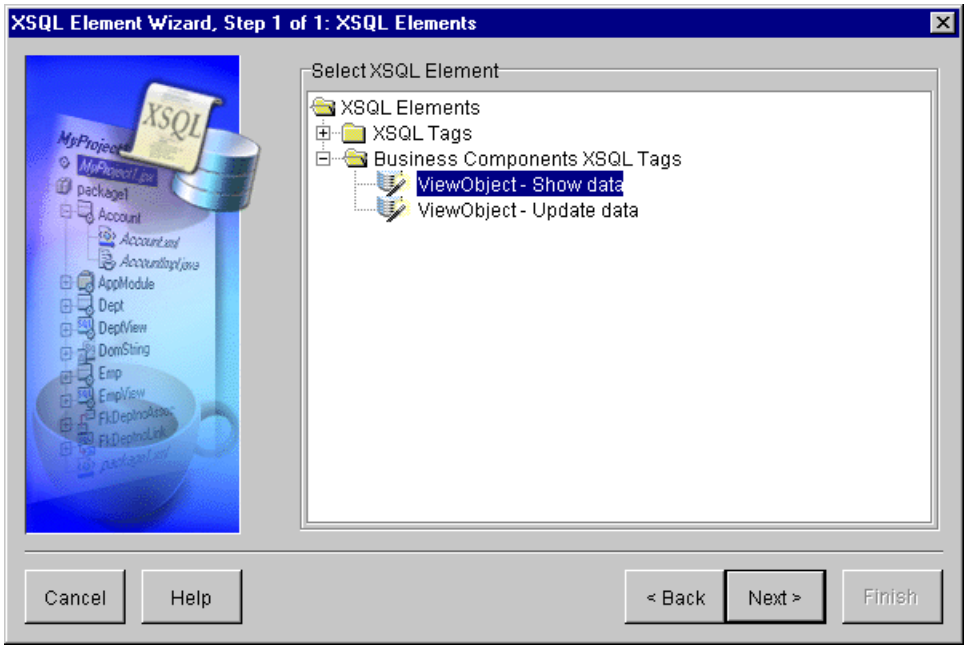


注意： 製品バージョンでは、これらの機能の表現（ウィザード）が異なる場合があります。

XSQL Element Wizard

XSQL Element Wizard は、データベース表または BC4J View Object へのアクセスを可能にするタグを追加するメカニズムを提供します。これらの表またはオブジェクトに問合せを実行するか、これらを介して基礎となるデータベース表を更新することができます。図 11-4 に、JDeveloper9*i* の XSQL Element Wizard を示します。

図 11-4 JDeveloper の XSQL Element Wizard



注意： 製品バージョンでは、これらの機能の表現（ウィザード）が異なる場合があります。

Page Selector Wizard

開発者が、Web アプリケーションの構築プロセスで XSQL ページを作成する必要がある場合、Page Wizard を起動します。このウィザードを使用して、XSQL ページをデータベース表上に直接、または BC4J View Object 上に作成できます。BC4J View Object 上に XSQL ページを作成する場合、リストからアプリケーション・モジュールを選択するか、新しいアプリケーション・モジュールを作成してから XSQL ページ・ベースのアプリケーションを作成するかを決定するプロンプトが表示されます。

参照：『Oracle9*i* Java Developer's Guide』を参照してください。

JDeveloper9*i* の XML 機能

JDeveloper9*i* がサポートする Oracle XDK for Java のコンポーネントは、次のとおりです。

- Oracle XML Parser for Java
- Oracle XSQL Servlet

XSLT プロセッサを含む Oracle XML Parser for Java および XSU は、Java で作成されているため、JDeveloper で使用できます。JDeveloper には、これらのコンポーネントがあります。

これらのツールの使用方法を説明するプログラムの例は、
`[JDeveloper]/Samples/xmlsamples` ディレクトリにあります。

Oracle XDK と Transviewer Beans の統合

Oracle XDK for Java は、次の XML ツールで構成されます。

- XML Parser for Java
- XSU for Java
- XML Class Generator for Java
- XSQL Servlet
- XML Transviewer Beans

これらすべてのユーティリティは Java で作成されているため、JDeveloper に簡単に組み込み、自由に使用できます。また、最新のバージョンを OTN サイト http://technet.oracle.com/tech/xml からダウンロードして XDK for Java のコンポーネントを更新できます。

Oracle XDK for Java には、XML Transviewer Beans も含まれています。これら一連の JavaBeans によって、XML アプリケーションにグラフィカルまたはビジュアルなインタフェースを簡単に追加できます。Beans は、JDeveloper から直接アクセスできるように、ドキュメントおよび記述子もカプセル化しています。これらの Beans を TOOLS パレットにドロップして、XML/XSL エディタなどのアプリケーション構築に使用できます。

> **参照：** Transviewer Beans の使用方法の詳細は、第 23 章「XML Transviewer Beans の使用」を参照してください。

Oracle XML Parser for Java

プロジェクトで Oracle XML Parser for Java を使用すると、XML 文書を検索および処理できるアプリケーションを作成できます。JDeveloper には Oracle XML Parser 用の組込みライブラリがあるため、1 回クリックするのみでプロジェクトに Oracle XML Parser を含めることができます。

Code Insight によって、コードをより簡単に理解および使用し、参照するクラスの Java ドキュメントへ適切にアクセスできます。XML Parser for Java は、次のインタフェースのいずれかを使用して XML 文書の処理を容易にします。

- DOM: W3C DOM のツリー
- SAX: SAX イベントのストリーム

Oracle XSQL Servlet

XSQL Servlet は SQL 問合せを処理し、結果セットを XML として出力するツールです。このプロセッサは Java サーブレットとして実装され、埋込み SQL 問合せを含む XML ファイルを入力として受け入れます。ほとんどの操作の実行には、XML Parser for Java および XSU を使用します。

XSQL Servlet を使用すると、生産的かつ簡単な方法で XML をデータベースから出し入れできます。単純なスクリプトを使用して、次のことが実行できます。

- 単純な XML 文書から複雑な文書まで生成できます。
- XSL スタイルシートを適用してあらゆるテキストの形式に生成できます。
- XML 文書を解析し、データをデータベースに格納できます。
- コードを 1 行もプログラムせずに、完全な動的 Web アプリケーションを作成できます。

JDeveloper の XSQL の例 1: emp.xsql

たとえば、次の XML の例を考えてみます。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
    select e.ename, e.sal, d.dname as department
    from dept d, emp e
    where d.deptno = e.deptno
  </xsql:query>
</FAQ>
```

次の XML が生成されます。

```
<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
  ...
</EMPLOYEES>
```

JDeveloper9*i* を使用すると、XSQL ファイルを容易に開発および実行できます。組込み Web サーバーおよびユーザーのデフォルト Web ブラウザが、結果ページの表示に使用されます。

XSQL Servlet でのアクション・ハンドラの使用

XSQL アクション・ハンドラは、XSQL Servlet アプリケーションから容易に起動できる Java クラスです。これらは Java クラスであるため、他の Java アプリケーションと同様に JDevloper でデバッグできます。

XSQL Pages アプリケーションを作成する場合、XSQL アクション・ハンドラを使用して、より複雑な作業を処理するための一連のアクションを拡張できます。このアクション・ハンドラをデバッグする必要があります。

XSQL Pages は、「HTML Source Directory」のプロジェクト・プロパティ「HTML Path」設定に指定したディレクトリに置く必要があります。

アクション・ハンドラをデバッグするには、次の手順に従います。

1. MyActionHandler というカスタムのアクション・ハンドラを参照する .xsql ファイルを作成したと想定します。
2. 計画どおりに動作しないため、このアクション・ハンドラをデバッグします。
3. Java ソース・ファイルでブレーク・ポイントを設定します。
4. .xsql ファイルを右クリックして、メニューの「Debug...」を選択します。

XML Data Generator Web Bean

Oracle JDeveloper には、XML Data Generator Web Bean があります。XML Data Generator Web Bean は、View Object のデータを含む XML を生成し、JSP 応答の出力ストリームにレンダリングします。

クライアントへの応答をレンダリングするために、XML および XSL を使用する JSP ページを作成できます。

XML Web Bean は JSP アプリケーションおよび Servlet アプリケーションで使用できます。XML Data Generator Web（Bean）は、ビジネス・コンポーネント（View Object）からデータを読み取り、適切な XML を作成します。Web Bean のメリットは、ビジネス・コンポーネント・アプリケーションを分析し、階層をナビゲートしてネストした XML を作成できることです。

XML Web Bean によって、XSL スタイルシートも指定できます。Web Bean は、XML に加えて、HTML、WML、変換済 XML およびその他のテキスト形式を生成できます。

Portal-to-Go および JDeveloper を使用したモバイル・アプリケーション開発

Portal-to-Go と Oracle JDeveloper を併用すると、モバイル・アプリケーション開発に非常に強力な環境が提供されます。開発者は、JDeveloper を使用してデータベースまたは BC4J アプリケーションから XML を生成し、Portal-to-Go を使用して Web ブラウザ、PDA または携帯電話にコンテンツを配信することができます。

JDeveloper を使用した XML アプリケーションの構築

次の例を考えてみます。この例では、データを提供するのではなく表示するために XML を使用する方法を説明します。ここでは、従業員と部門の間の多対 1 関係を示しています。

JDeveloper の XML の例 1: BC4J メタデータ

```
<Departments>
<Dept>
   <Deptno>10</Deptno>
   <Dname>Sales></Dname>
   <Loc>
   <Employees>
     <Employee>
        <Empno>1001></Empno>
        <Ename>Scott</Ename>
        <Salary>80000</Salary>
     </Employee>
   </Employees>
...
     </Employee>
   </Employees>
</Dept>
<Dept>
...
```

JDeveloper9*i* でのアプリケーションの構築手順

JDeveloper9*i* でこのプロジェクトを構築するには、次の手順を実行します。

1. 「File」>「New Project」を選択して、新規の JDeveloper プロジェクトを開始します。
2. BC4J アプリケーションを作成します。
3. Page Selector Wizard を起動して、BC4J アプリケーション・モジュールに基づいて XSQL ページを作成します。
4. 表示されたリストからアプリケーション・モジュールを選択します。
5. XSQL ページの基になる View Object を選択します。
6. 表示する列を選択します。

Page Wizard でのこれらの手順を終了すると、BC4J フレームワークの View Object に基づいた XSQL ページが作成されます。このページを実行すると、ブラウザに XML データが送信されます。データが希望どおりに表示されるようにフォーマットするために、オプションでスタイルシートを作成できます。また、PDA や携帯電話で表示されるように調整できます。

JDeveloper の XML Data Generator Web Bean の使用

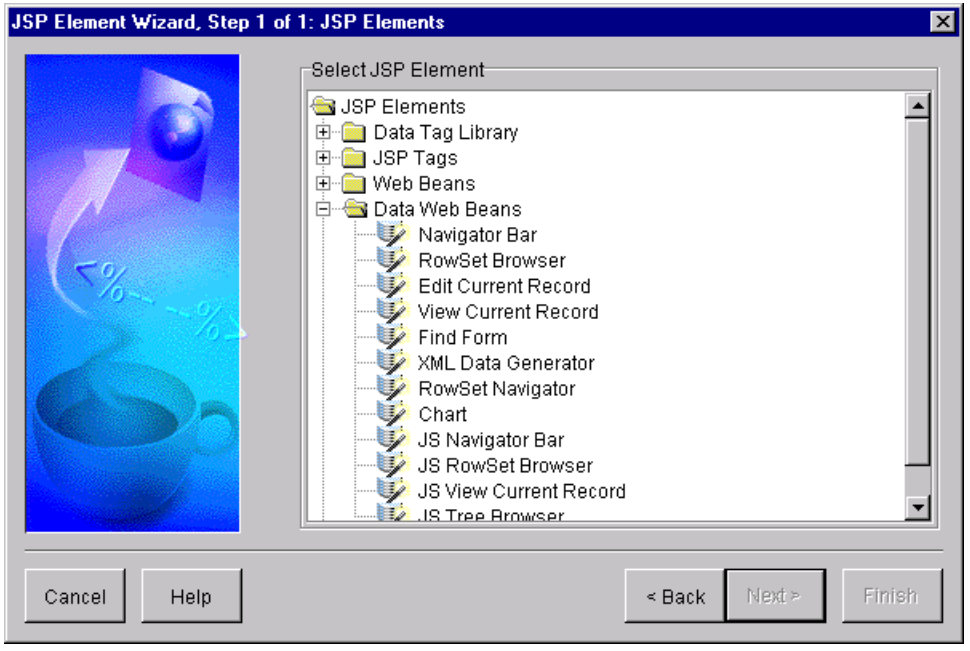
XML Data Generator Web（Bean）は、JSP および Servlet アプリケーションで使用できます。XML Data Generator Web（Bean）は、ビジネス・コンポーネント（View Object）からデータを読み取り、適切な XML を作成します。この Web Bean のメリットは、次のとおりです。

- ビジネス・コンポーネント・アプリケーションを分析し、階層をナビゲートしてネストした XML を作成します。
- XSL スタイルシートを指定できます。Web Bean は、HTML、WML、変換済 XML およびその他のテキスト・フォーマットを生成できます。

Data Generator Web Bean は、JSP Element Wizard の「Data Web Beans」カテゴリにあります。図 11-5 に、JSP Element Wizard から XML Data Generator Web（Bean）へアクセスする方法を示します。

要素を含めるページ上を右クリックして、JSP または XSQL Servlet の Element Wizard をコールします。Element Wizard のパラメータとしてスタイルシートを指定します。

図 11-5 JSP Element Wizard: XML Data Generator Bean

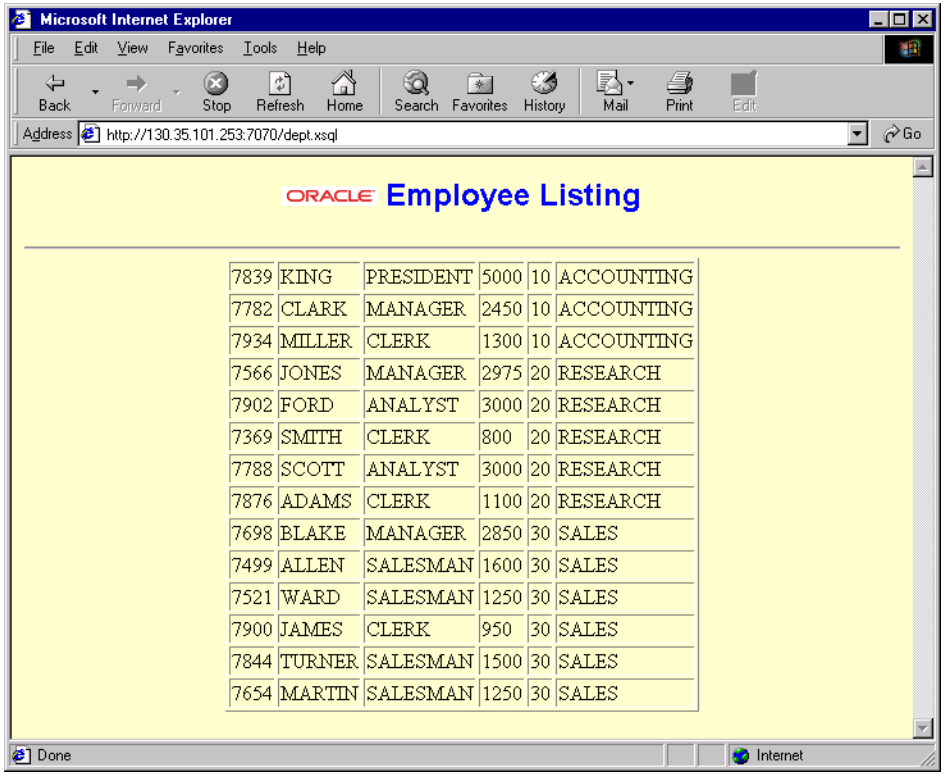


注意： 製品バージョンでは、これらの機能の表現（ウィザード）が異なる場合があります。

このウィザードを起動すると、生成する XML データに適用するスタイルシートを指定し、出力結果を参照できます。

図 11-6 に、従業員リストに XSL スタイルシートを適用して表示される出力例を示します。

図 11-6 ブラウザに HTML で表示される従業員リスト（XML+XSLT= HTML）



JDeveloper での XSQL Servlet の使用

XSQL Servlet を使用すると、効率的かつ簡単に XML をデータベースから出し入れできます。

> **参照：** XSQL Servlet の使用方法は、第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」および第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

JDeveloper で XSQL Servlet を使用する場合、XSQL Runtime というライブラリをプロジェクトに含める必要はありません。XSQL Runtime は、新しい XSQL ページまたは XSQL ウィザード・ベースのアプリケーションにはすでに含まれています。

単純なスクリプトを使用して、JDeveloper で次のことが実行できます。

- 単純な XML 文書から複雑な文書まで生成できます。
- XSL スタイルシートを適用して、あらゆるテキスト形式に生成できます。
- XML 文書を解析し、データをデータベースに格納できます。
- コードを 1 行もプログラムせずに、完全な動的 Web アプリケーションを作成できます。

XSQL ファイルの簡単な問合せを考えてみます。この問合せでは、emp 表のすべての従業員の詳細情報が戻されます。この情報を取得する XSQL コードは、例 2 で示します。

JDeveloper の XSQL の例 2: emp 表の従業員データ : emp.xsql

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
    order by empno
</xsql:query>
```

図 11-7 に、ブラウザに表示される従業員の未加工の XML データを示します。

図 11-7 未加工の XML 形式の従業員データ

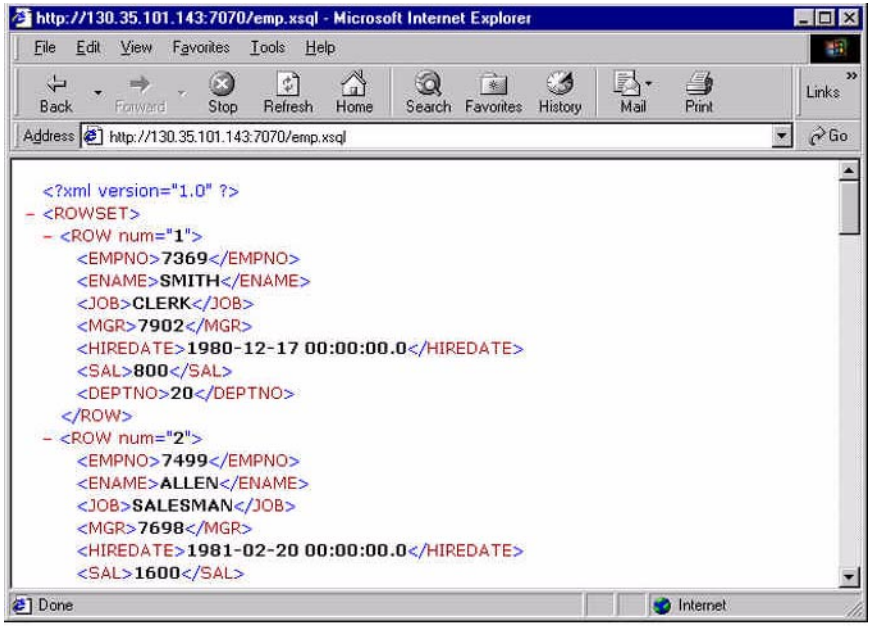


図 11-6 に示すとおりにデータを表形式で出力する場合、スタイルシートを指定するために XSQL コードを少し変更する必要があります。この例で行う変更は、次の例で強調されている部分です。

JDeveloper の XSQL の例 3: スタイルシートが追加された従業員データ

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
    order by empno
</xsql:query>
```

結果は、図 11-6 の表のようになります。XSQL Servlet を使用して、他にも多くのことが実行できます。

> **参照：** 第 10 章「XSQL ページ・パブリッシング・フレームワーク」、および OTN サイト http://otn.oracle.com/tech/xml の『XDK for Java, XSQL Servlet Release Notes』を参照してください。

JDeveloper でのモバイル・アプリケーションの作成

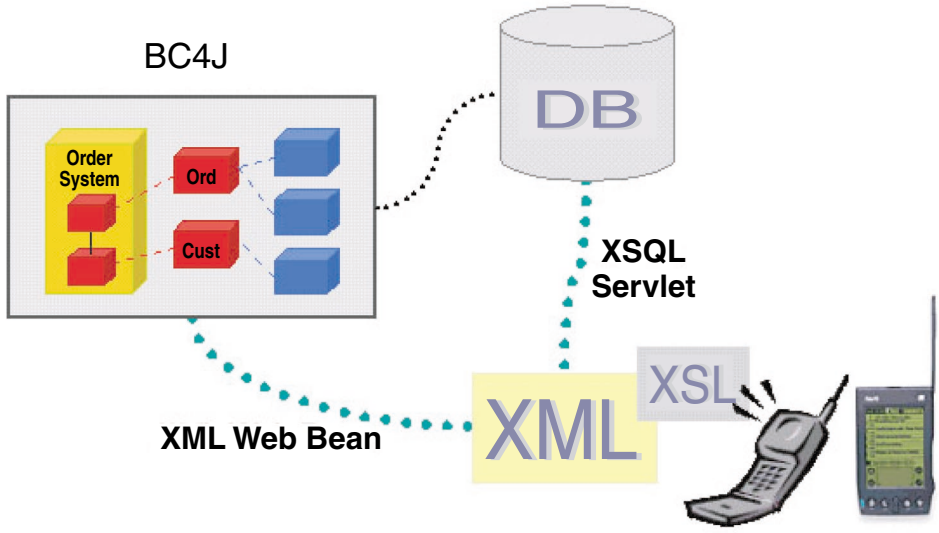
このモバイル・アプリケーションは、部門データベース・アプリケーションです。このアプリケーションは、BC4J および XML を使用して、ワイヤレス・デバイスでアクセスできるアプリケーションを開発できることを実証します。このアプリケーションは、主に次の 2 つの部分で構成されます。

- 1 つは BC4J フレームワークを使用して開発されるサーバー側のビジネス・ロジックであり、もう 1 つはクライアント部分です。ビジネス・ロジックは、SCOTT スキーマの DEPT 表に基づいた参照オブジェクトで構成されます。
- DEPT 表を問い合せて、ブラウザ、携帯電話、Palm Pilot などのクライアント・デバイスから更新するメカニズムです。Palm Pilot には、Windows NT で動作するエミュレータを使用します。

図 11-8 に、モバイル・アプリケーションが BC4J、XSQL Servlet、XSL スタイルシートおよび Oracle9*i* でどのように動作するかを概念的に示します。

同様のアプリケーションのより詳細なデモについては、http://otn.oracle.com/tech/xml を参照してください。

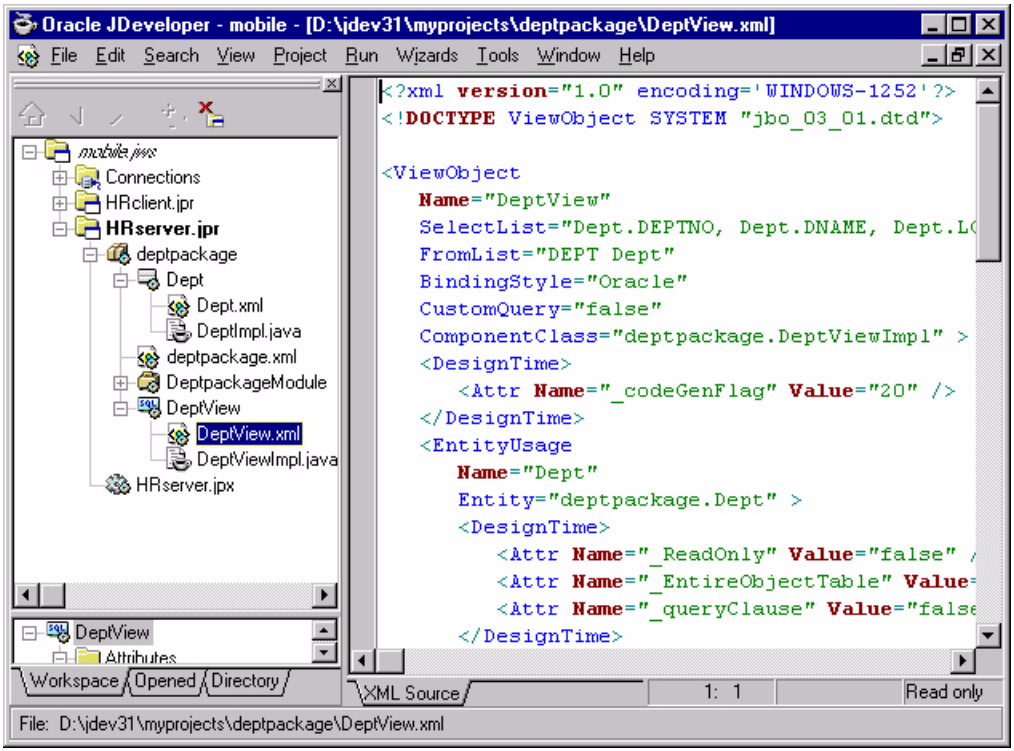
図 11-8 BC4J および XSQL Servlet を使用した JDeveloper でのモバイル・アプリケーションの作成



BC4J アプリケーションの作成

まず、BC4J アプリケーションを作成します。このアプリケーションは、Oracle9*i* データベースで SCOTT スキーマに接続します。図 11-9 に、DEPT オブジェクトに関するメタデータを含む XML ファイルを示します。11-15 ページの「JDeveloper の XML の例 1: BC4J メタデータ」を参照してください。

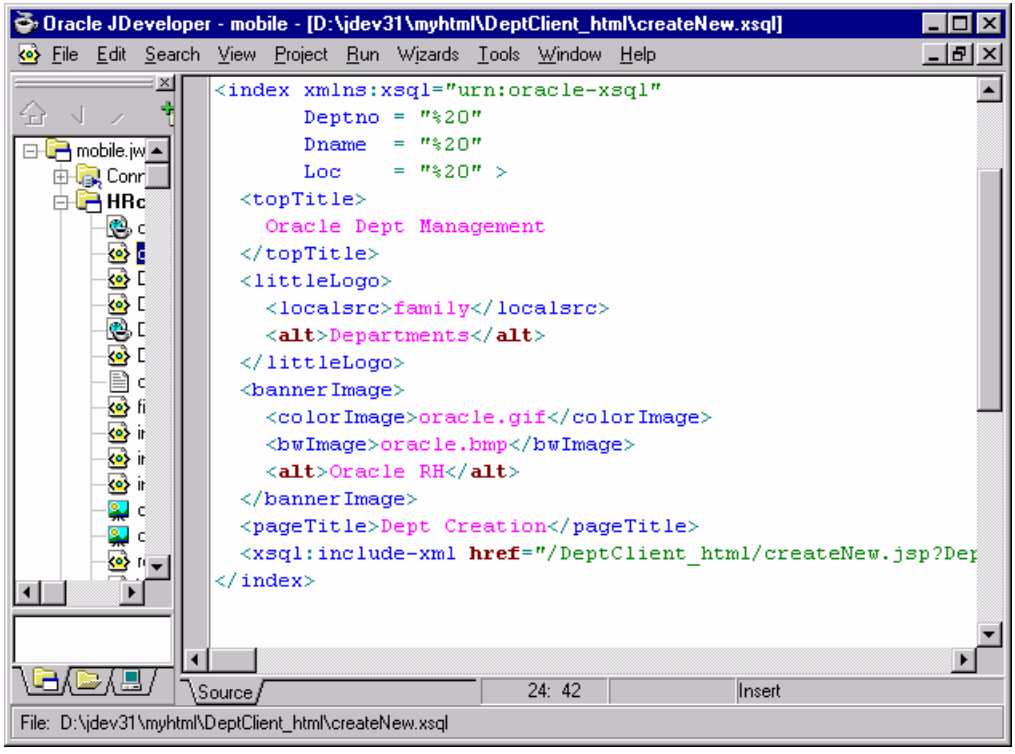
図 11-9 BC4J アプリケーション : DEPT View Object の XML ファイル



BC4J アプリケーションに基づいた JSP ページの作成

BC4J アプリケーションに基づいて JSP ページを作成します。JSP ページでは、XML Data Generator Web Bean を導入します。図 11-10 に、新しい部門を作成するために JSP ページをコールする XSQL ファイルを示します。

図 11-10 BC4J アプリケーション : JSP ページをコールする XSQL ファイル



データ読取りに必要なデバイスに従った XSLT スタイルシートの作成

アクセスするデータの様々なクライアント・デバイスに対応する XSLT スタイルシートを作成します。XSQL ファイルでは、スタイルシートのリストおよび対応するプロトコルを指定します。このプロトコルは、基本的にスタイルシートをクライアント・デバイスに結合させます。

図 11-11 に、Palm Pilot のエミュレータのブラウザ上に表示するために、XML データを HTML に変換するスタイルシート（indexPP.xsl）のコード例の一部を示します。

図 11-11 BC4J アプリケーション : XSL スタイルシート（indexPP.xsl）

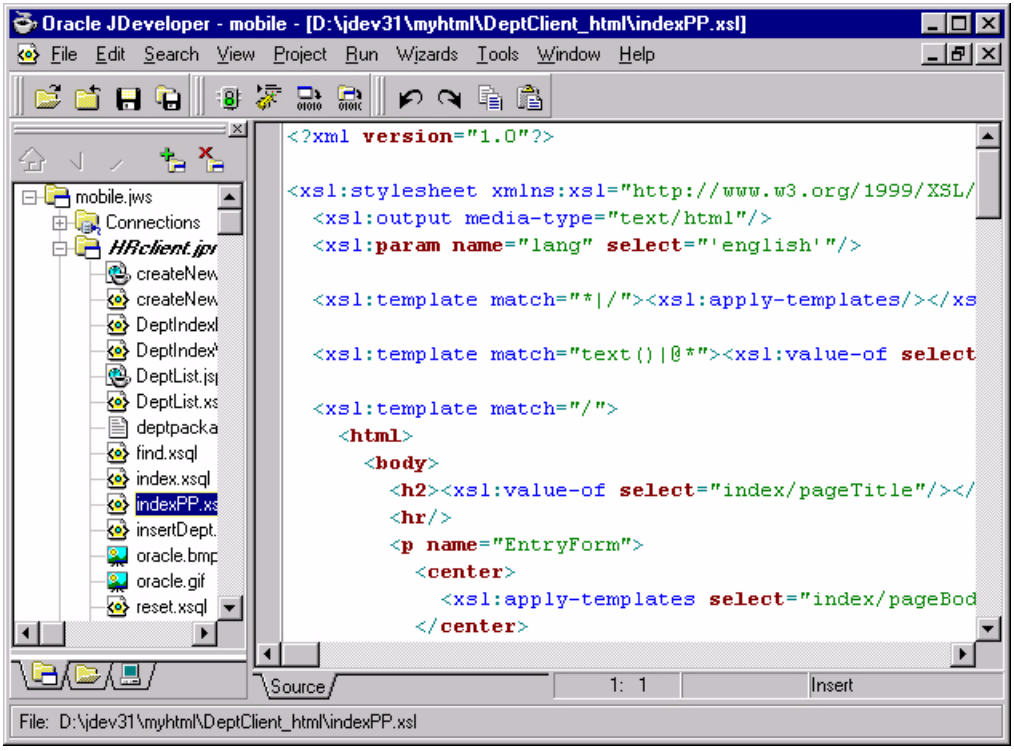


図 11-12 に、部門アプリケーションのクライアントを実行中の携帯電話のエミュレータを示します。携帯電話のエミュレータの設定画面も示しています。

図 11-12 部門アプリケーションのクライアントを実行中の携帯電話のエミュレータ

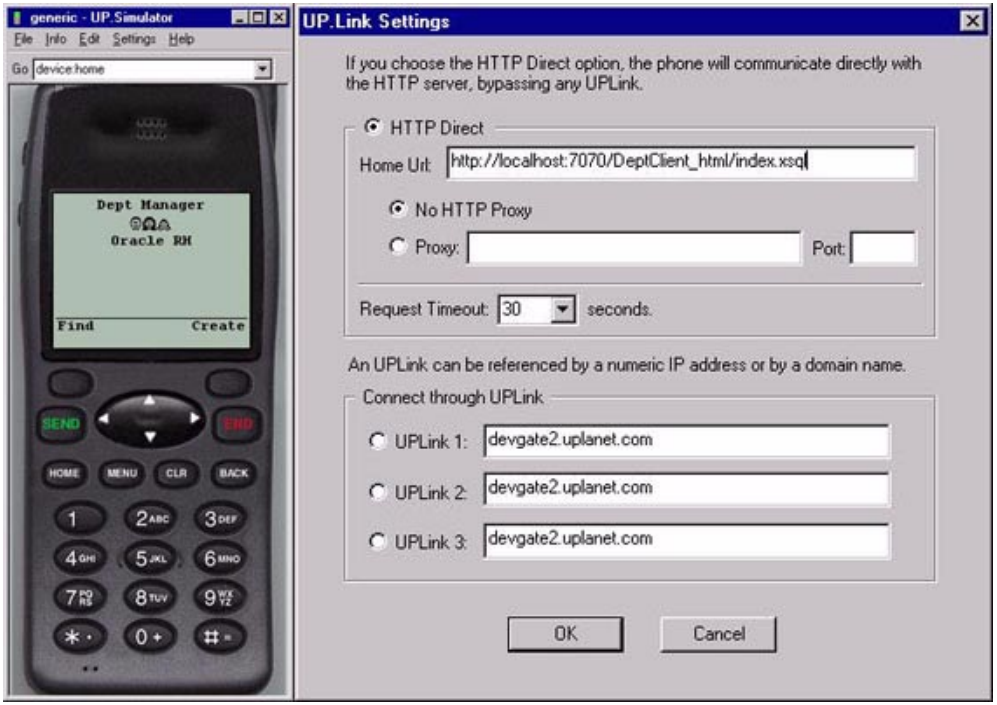


図 11-13 に、HandWeb ブラウザを介して部門アプリケーションにアクセス中の Palm Pilot のエミュレータを示します。

図 11-13 HandWeb ブラウザを介して部門アプリケーションにアクセス中の Palm Pilot のエミュレータ

FAQ: JDeveloper を使用した XML アプリケーションの構築

JSP での XML 文書の作成

質問

XML Class Generator for Java が（DTD に基づいて）生成したクラスを使用し、XSL スタイルシートを適用して HTML に変換して、JSP ページに XML 文書を（PL/SQL API が戻したデータ結果から）動的に作成しています。この方法は、JSP が初めてアクセス（および内部的にコンパイル）されたときは正常に動作しますが、その後同じページにアクセスするたびに失敗します。

次のようなエラーが表示されます。

```
"oracle.xml.parser.v2.XMLDOMException: Node doesn't belong to the current document"
```

再度動作させる唯一の方法は、JSP ページにタッチして JSP をコンパイルすることです。この方法が有効なのは 1 回のみです。Apache JServ を使用しています。

対処方法を教えてください。最上位のノード用に生成された Java クラスでの「static」を使用したコードが関係していますか？

回答

JSP に無効な状態を格納しているようです。XML パーサーがこの無効な状態を取得すると、前述の例外を発生させます。

CRM はアプリケーションで HTTP セッションを使用しません。ご質問の件も、この場合に該当すると思われます。メンバー変数を使用して、誤って無効な状態を格納した可能性があります。メンバー変数とは、次の構文で宣言された変数です。

```
<%! %>
```

次に例を示します。

```
<%! Document doc=null; %>
```

変数宣言のためにこの構文を使用する必要があると誤解している方が多いようですが、実際はこの構文を使用する必要はありません。ほとんどの場合、メンバー変数は必要ではありません。メンバー変数はすべての要求で共有され、JSP の存続期間中に 1 回のみ初期化されます。

ほとんどのユーザーには、スタック変数またはメソッド変数が必要です。これらの変数は、要求されるたびに作成および初期化されます。変数は次の例のとおり、スクリプトレット形式として宣言されます。

```
<% Document doc=null; %>
```

この場合、すべての要求には固有のドキュメント・オブジェクトがあり、このオブジェクトは要求されるたびに初期化されて NULL になります。

セッションに無効な状態も JSP にメソッド変数も格納していない場合、この問題には他の理由が考えられます。

BC4J での XMLData の使用

質問

BC4J のデータを取り出すために、XMLData を使用しています。JSP の XMLData ではなく、スタンドアロンの Java アプリケーションのものを使用しています。ターゲット・レコードに「R & D」という値があります。

XMLData が戻す「R & D」は、HTTP には適切ですがこの場合は適切ではありません。XMLData は文字をエスケープすることなく、データベースにあるものを戻せますか?

回答

XMLData はメモリー内 DOM を構築するため、問題なのは XML パーサーのシリアル化です。唯一考えられる対処方法は、次のとおりです。

1. 希望どおりに動作する DOM ツリー用のシリアル化コードを作成します。
2. テンプレートを 1 つ追加した認証変換を行い、disable-output-escaping=YES を設定したデータを書き込みます。

JDeveloper 3.0 での XML Parser for Java の実行

質問

JDeveloper をノートブック・コンピュータ（オペレーティング・システムは Windows 95）にダウンロードしました。XML パーサーのプログラム例（SimpleParse.java）を実行しようとしています。このプログラムは、org.w3c.dom.Document クラスをインポートします。正確なディレクトリで autoexe.bat に CLASSPATH を設定しました。DOS プロンプトでは、「java SimpleParse <filename>」コマンドでプログラムを実行できます。JDeveloper を介して同じプログラムを実行しようとすると、次のエラーが表示されます。

```
"identifier org.w3c.dom.Document not found"
```

何か不足しているものがありますか?

回答

プロジェクトに、Oracle XML Parser 2.0 という名前のライブラリがあることを確認してください。

このライブラリは JDeveloper 3.0 以上によって事前定義されています。このプロジェクトのライブラリ・リストを参照するには、「Project 」>「Properties」を選択し、「Paths」タブを選択します。

「(Add...)」ボタンをクリックし、リストから前述のライブラリを指定します。

org.w3c.dom.* インタフェースは、この Jar に含まれます。この W3C インタフェースは、XML ノードのツリーと動作するための、DOM 標準 API を定義します。

質問

@code をキーとして使用する場合、次の文を使用します。

```
<xsl:template match="aTextNode">
    ...
   <xsl:param name="labelCode" select="@code"/>
     <xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
    ...
   </xsl:template>
```

これでも動作しますが、@code を直接 document() 行に使用する方法はありませんか？

回答

これは、current() 関数が有効な場合の方法です。次の方法は使用しないでください。

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
```

かわりに次のように実行します。

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
                      and @lang = $lang]"/>
```

質問

messages.xml に格納されたデータをデータベースから取り出せますか？リスナーおよびサーブレットがデータベース内で動作するところでは、document() の命令はどうなりますか？

回答

データは取り出せます。仕様によって、XSLT エンジンは document() 関数で参照されるドキュメントを読取りおよびキャッシュします。渡された URI の文字列形式に基づいて解析ドキュメントをキャッシュし、次のとおりデータベース・ベースのメッセージ検索を行えます。

1. 次のように入力して、MESSAGES 表を作成します。

```
CREATE TABLE MESSAGES (lang VARCHAR2(2), code NUMBER, message VARCHAR2(200));
```

2. 次の msg.xsql のような XSQL ページを作成します。

```
<xsql:query lang="en" xmlns:xsql="urn:oracle-xsql" connection="demo"
            row-element="" rowset-element="">
      select message
        from messages
       where lang = '{@lang}'
         and code = {@code}
</xsql:query>
```

3. 次のような document() 関数に msg.xsql を使用するスタイルシートを作成します。

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
  <html><body>
    In English my name is
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
      </xsl:call-template><br/>
    En espanol mi nombre es
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">es</xsl:with-param>
      </xsl:call-template><br/>
    En fran&#231;ais, je m'appelle
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">fr</xsl:with-param>
      </xsl:call-template><br/>
    In italiano, mi chiamo
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
```

```
          <xsl:with-param name="lang">it</xsl:with-param>
        </xsl:call-template>
    </body></html>
    </xsl:template>
    <xsl:template name="msg">
      <xsl:param name="lang">en</xsl:param>
      <xsl:param name="code"/>
      <xsl:variable name="msgurl"
  select="concat('http://xml/msg.xsql?lang=',$lang,'&amp;code=',$code)"/>
      <xsl:value-of select="document($msgurl)/MESSAGE"/>
    </xsl:template>
  </xsl:stylesheet>
```

4. http://xml/testmessage.xsql で試行します。

これは、Web からメッセージをフェッチする場合に最適です。別の方法として、前述の msg.xsql を使用できますが、次の文が有効と考えられる場合は、XSQL ページに msg.xsql を含めます。

```
<xsql:include-xsql href="msg.xsql?lang={@lang}&amp;code={@code}"/>
```

または、独自のカスタムのアクション・ハンドラを作成し、JDBC を使用してメッセージをフェッチし、XSQL ページに含めることができます。

複雑な XML 文書のデータベースへの移動

質問

Oracle データベースに XML 文書を移動しています。文書は非常に複雑です。XML 文書および Oracle XDK は、XML 文書のデータベースへの格納方法に対応可能な DDL 形式、理想的にはオブジェクト・リレーショナル構造を生成できますか？これを実行できるツールはどれですか？

回答

最適な方法は、XML Class Generator for Java を使用することです。DTD ファイルがまだ作成されていない場合、XSU を使用します。マッピング・プログラムも作成する必要があります。

別の方法では、ビューおよびストアド・プロシージャを作成して複数の表を更新します。ただし、どちらの場合でも事前に表およびビューを作成する必要があります。

BC4J BC4J フレームワークは、データベースへ出入りする E-Commerce の XML メッセージをマッピングするための、一般的なメタデータ駆動のソリューションを提供します。BC4J の機能に関するテクニカル・ホワイト・ペーパーは、http://otn.oracle.com/products/jdev/info/techwp20/wp.html にあります。

Pure Java で XML ベースのビジネス・コンポーネントのフレームワークによって、より簡単に E-Commerce のアプリケーションを作成できます。これは単独で使用できる Java のフレームワークであり、JDeveloper 3.0 IDE 内にきめ細かな開発サポートが組み込まれています。JDeveloper 3.0 IDE は http://otn.oracle.com/software/ からダウンロードして入手できます。

BC4J によって、すべてのアプリケーション動作（ルールおよびプロセス）を一定の方法で管理するためのビジネス・コンポーネントに、SQL ベースの参照コンポーネントの階層を柔軟にマッピングできます。動的な機能がサポートされているため、ほとんどの機能は XML メタデータから実行できます。このフレームワークを使用して、あらゆる XML 文書をデータベースに、およびデータベースから柔軟にマッピングするレイヤーを構築できます。主なメリットは、XML 文書をシステムで使用すると、同じビジネス・ルールすべてが自動的に検証されることです。

12

BC4J および XML アプリケーションの作成

この章の内容は次のとおりです。

BC4J の概要

Oracle BC4J は、Java 対応で XML ベースのフレームワークです。このフレームワークによって、再利用可能なビジネス・コンポーネントからの複数層でデータベースを使用するアプリケーションの開発、移植可能な配置、および柔軟なカスタマイズが可能となります。

XML メッセージのマップ

BC4J フレームワークは、データベースへ出入りする E-Commerce の XML メッセージをマッピングするための、一般的なメタデータ駆動のソリューションを提供します。

JDeveloper を使用した BC4J アプリケーションのテスト

Oracle BC4J のフレームワーク、および Oracle JDeveloper のウィザードとコンポーネント・エディタを使用して、再利用可能なビジネス・コンポーネントからアプリケーション・サービスを作成およびテストします。

JDeveloper では、ビジュアル・ウィザードを使用して XML メタデータの記述を変更し、既存のビジネス・コンポーネントの機能をカスタマイズできます。

参照： 次の章、マニュアルおよび Web サイトを参照してください。

- 第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」
- 『Oracle9*i* Java Developer's Guide』
- http://otn.oracle.com/products/bc4j

BC4J の機能

BC4J の機能は、次のとおりです。

- 一度作成したらどこにでも配置できます。
- コンポーネント・ベースで複数層のエンタープライズ・アプリケーションを構築するための、Pure Java 対応のアプリケーション・フレームワークです。
- ビジネス・ロジックと UI を簡単に分離できます。
- XML および Java を使用して簡単にカスタマイズできます。
- 複合的なデータベースの相互作用の自動化に有効です。
- すべてのアプリケーション動作（ルールおよびプロセス）を一定の方法で管理するためのビジネス・コンポーネントに、SQL ベースの参照コンポーネントの階層を柔軟にマップできます。

- 動的な機能がサポートされているため、ほとんどの機能は XML メタデータから実行できます。
- このフレームワークを使用して、あらゆる XML 文書をデータベースに、およびデータベースから柔軟にマッピングするレイヤーを構築できます。XML 文書をシステムで使用すると、同じビジネス・ルールすべてが自動的に検証されます。

BC4J のメリット

BC4J のメリットは、次のとおりです。

- ビジネス・ロジックの再利用
- エンティティによるビジネス・ロジックのカプセル化
- ビジネス・コンポーネントの対応付けおよび構成
- 柔軟で更新可能な SQL ベースのビュー
- 最適なデータの取出しおよびキャッシュ
- ビジネス・ロジックとの自動調整
- Pure Java 対応
- ローカルおよびリモートでの CORBA および EJB Session Bean のサポート

柔軟な配置

BC4J で開発されたサービスは、CORBA サーバー・オブジェクトまたは EJB Session Bean のいずれかとして、Java テクノロジをサポートする企業規模のサーバー・プラットフォーム上に配置できます。

また、同じサーバー側のビジネス・コンポーネントは、JavaServer Pages/Servlet アプリケーションまたは EJB のコンポーネントとして、変更せずに配置できます。

このような柔軟な配置によって、ビジネス・ロジックおよびデータ・モデルを再利用し、コードを再作成することなく様々なクライアント、ブラウザおよびワイヤレス・インターネット・デバイスにアプリケーションを提供できます。

JDeveloper での BC4J XML アプリケーションの作成

JDeveloper9*i* の BC4J フレームワークは、XML を使用して、オブジェクトの宣言の設定および機能を表すメタデータを定義します。カスタムまたは複雑なビジネス・ロジックは、Java で実装できます。

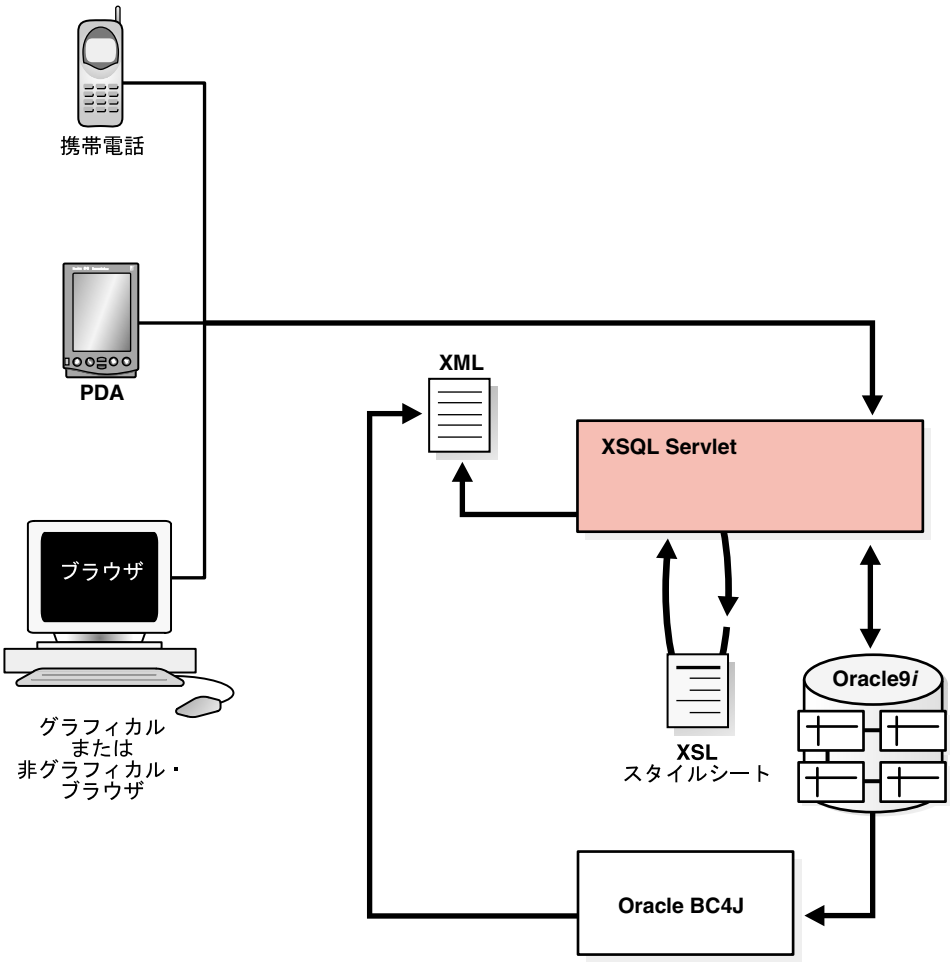
- BC4J Tester を使用すると、View Object 内のデータを XML として参照できます。
- 検証規則などのビジネス・ルールは、Java ソース・コードではなく XML に格納されます。
- Java ソース・コードのかわりに XML を変更して、ビジネス・アプリケーションを容易にカスタマイズできます。
- ロジックを XML で抽象化することで、アプリケーションを簡単に読んだり、理解することができます。

BC4J による XML を使用したメタデータの格納　JDeveloper とともに提供される BC4J フレームワークは、XML を使用して、アプリケーションのコンポーネントに関するメタデータを格納します。重要な情報は、Java ソース・コードのかわりに構造化ドキュメントに格納されています。これによって、アプリケーションを簡単に理解およびカスタマイズできます。

アプリケーションは、ソース・コードを変更することなくカスタマイズできます。

図 12-1 に、XSQL Servlet が BC4J を使用して XML 文書を生成する方法を示します。

図 12-1 BC4J の使用



ビジネス・ルールは、基礎となるコンポーネントのソース・コードへアクセスせずに、その場で変更できます。

BC4J による XSQL クライアントの作成

JDeveloper9*i* では、XSQL Servlet を構築できます。XSQL Servlet は、BC4J アプリケーション・モジュールと統合して、中間層から複数のクライアントへアプリケーション・ロジックを提供します。ユーザーは、対応するスタイルシートを適用するのみで、XML データを取り出し、すべてのクライアント・デバイスでデータを表示できます。

参照： 次の章およびマニュアルを参照してください。

- 第 11 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」
- 『Oracle9*i* Java Developer's Guide』
- 『Oracle9*i* ケース・スタディ -XML アプリケーション』

XML および Java アプリケーション作成時におけるコードの生成と管理の簡略化

BC4J を使用して XML アプリケーションを作成する場合の JDeveloper の一般的なコード要件は、次のとおりです。

- 各エンティティ・オブジェクトおよびビュー・オブジェクトに対する Java ファイルおよび XML ファイル
- 各関連オブジェクトおよびリンク・オブジェクトに対する Java ファイル
- アプリケーション・モジュールに対する Java ファイルおよび XML ファイル
- JDeveloper のナビゲータでこれらのファイルをダブルクリックして、ファイルの内容を参照します。

BC4J フレームワークは、XML と Java のコードの組合せを使用して各ビジネス・コンポーネントを表示します。

- **XML:** XML コードは、宣言の設定およびオブジェクトの機能を表すメタデータを定義します。
- **Java:** Java コードは、オブジェクトの動作を実装します。

生成されるその他の典型的なファイルは次のとおりです。

- エンティティの Java による実装
- ビューの XML ファイル
- ビューの Java 実装
- アプリケーション・モジュールの XML ファイル
- アプリケーション・モジュールの Java による実装

13

メタデータ API の使用

この章の内容は次のとおりです。

メタデータ API の概要

メタデータ API によって、集中化された、単純かつ柔軟な方法で、次の作業を実行できます。

- データベース・オブジェクトの完全な定義（メタデータ）の、XML または作成 DDL のいずれかとしての抽出
- 業界標準の XSLT によるメタデータの変換
- SQL DDL の生成によるデータベース・オブジェクトの再作成

メタデータ API は、インスタンスが操作可能である場合に、Oracle で使用できます。Oracle Lite では使用できません。

従来のメタデータの抽出方法

オブジェクトのメタデータは、定められた形式でディクショナリ全体に分散されています。以前のリリースでは、ユーザーはまずオブジェクトのメタデータがディクショナリ内でどのように表示されるか、どこに表示されるかを理解してから、複数の問合せを発行してオブジェクトの完全な表現を抽出する必要がありました。メタデータを抽出するには、通常、次のタスクを実行していました。

1. オブジェクトの表領域の変更、列のデータ型の変更、オブジェクトの所有者の変更などによるメタデータの変換
2. ソース・データベースまたは他のデータベースで実行するための、メタデータの SQL DDL テキストへの変換

以前のリリースでは、この 2 つの手順に対するサポートがありませんでした。

メタデータ API コンポーネント

メタデータ API の基礎は、一連のユーザー定義型および対応するオブジェクト・ビューで構成された Oracle ディクショナリのオブジェクト・モデルです。ユーザー定義型は、各オブジェクト・クラスのメタデータの集計を提供し、オブジェクト・ビューは、ユーザー定義型の属性をディクショナリ内の適切なベース・リレーショナル表にマップします。メタデータ API は、集計されたデータベース・オブジェクトの定義を取り出すために、これらのオブジェクト・ビューに対して問合せを生成します。

これらの問合せの結果は、XML SQL Utility（XSU、Oracle の新機能）によって XML 文書に変換されます。コール側が DDL 出力を要求すると、メタデータ API は Oracle サーバーに統合された XML パーサーおよび XSL プロセッサを使用して、その XML 文書を作成 DDL に変換します。

メタデータ API の機能

メタデータ API の機能は、次のとおりです。

- 詳細なプログラム制御または簡易ブラウザに対する強力な PL/SQL インタフェースの提供
- 次に示すオブジェクト・クラスに対する、完全で、集計済のデータベース・オブジェクト定義の検索のサポート
  - すべての表タイプ（リレーショナル表、オブジェクト表、索引構成表、ネストした表、一時表、パーティション表）
  - 索引（ファンクション索引およびドメイン索引）
  - ユーザー定義型
  - プロシージャ、ファンクションおよびパッケージ
  - 演算子
  - 索引タイプ
  - リレーショナル・ビューおよびオブジェクト・ビュー
  - トリガー
  - シノニム
  - 権限付与（オブジェクト権限の付与およびシステム権限の付与）
  - アウトライン
- オブジェクトの完全な表現のみの提供

  注意： 今回のリリースでは、オブジェクト属性のサブセット化は、XSLT 変換を介したサブセット化以外はサポートしていません。

- ダウンストリーム・プロセスによる XSLT を介した変換が容易に行える、XML 形式のデータベース・オブジェクト・メタデータの提供
- サポートするすべてのオブジェクトに対する、完全な Oracle 固有の作成 DDL のサポート
- 1 つの問合せで複数のオブジェクトを戻すことを可能にする、柔軟なオブジェクト選択
- 最初の変換の出力が次の変換の入力になる、連鎖した変換のサポート
- オブジェクト型固有の変換パラメータを介した DDL 出力のカスタマイズ

インターネット・コンピューティング

メタデータ API は、2 つのインターネット標準である XML および XSLT を使用して、オブジェクト・メタデータのエンコーディングおよび変換を実行します。メタデータのエンコーディングに、独自の形式ではなく業界標準の形式を使用するため、出力の解析および変換に標準ツールを使用できます。

データベース・メタデータに対する業界標準の XML モデルが現行では存在しないため、メタデータ API は、Oracle DDL を生成するために最適化されたモデルを使用します。ドキュメントの要素名は、Oracle ディクショナリ・モデルのユーザー定義型属性から直接導出されます。標準モデルが出現した場合、メタデータ API は、そのモデルをプラグインする機能をサポートしています。古いドキュメントは、XSLT を使用して新しいモデルに変換できます。

n 層インターネット・コンピューティングでは、メタデータ API は、メタデータに近いサーバーにバインドされます。したがって、メタデータ API は、Java などの他のすべての言語からコールできる PL/SQL で実装されます。

DBMS_METADATA の概要

DBMS_METADATA は、メタデータ API を実装する PL/SQL パッケージです。このパッケージによって、コール側はデータベース・ディクショナリからメタデータを取り出すことができます。また、柔軟で拡張可能な方法でオブジェクトを選択できます。さらに、XML および DDL のデータベース・オブジェクト・メタデータを抽出できます。

DBMS_METADATA には、次の 2 種類のインタフェースがあります。

- **プログラム・インタフェース**: きめの細かい、詳細な制御が可能です。
  - 次に示すルーチンが提供されています。詳細は、この章の後半で説明します。OPEN、SET_FILTER、SET_PARSE_ITEM、SET_COUNT、ADD_TRANSFORM、SET_TRANSFORM_PARAM、FETCH_xxx、CLOSE
  - メタデータは、XML として表されます。これによって、XSLT を使用した業界標準のメタデータ変換を実行できます。
  - メタデータを DDL として戻すように DBMS_METADATA に指定できます。メタデータ API は、XSL スクリプトを内部的に使用して、透過的に変換を実行します。
  - Oracle XML Parser またはサード・パーティ製のツールを使用して、XSL スクリプトをコールするだけで、XML 表示のオフライン変換を実行できます。

- **ブラウザ・インタフェース :** SQL*Plus などの SQL クライアント内で簡易的に使用できます。
  - GET_DDL および GET_XML

    たとえば、次の問合せは、現行のユーザーのスキーマにあるすべての表に対する DDL を示します。

    ```
    SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
    ```

  - セッション・レベル固有の変換パラメータ

DBMS_METADATA およびセキュリティ

Oracle メタデータ・モデルのオブジェクト・ビューは、セキュリティを実装しています。権限のないユーザーは、ユーザー自身のオブジェクトのメタデータ以外は参照できません。SYS ユーザーおよび SELECT_CATALOG_ROLE を持つユーザーは、すべてのオブジェクトを参照できます。権限のないユーザーは、自身に付与されているオブジェクト権限およびシステム権限、または他のユーザーに付与したオブジェクト権限およびシステム権限は取り出すことができます。これには、PUBLIC に付与されている権限も含まれます。

コール側が、取り出す権限を持たないオブジェクトを要求した場合、例外は呼び出されませんが、オブジェクトを取り出すことはできません。

注意 : 権限のないユーザーに、他のユーザーのスキーマ内のオブジェクトにアクセスするフォームが付与された場合、メタデータ API を介して権限付与仕様部を取り出すことができますが、オブジェクトの実際のメタデータを取り出すことはできません。

注意 : メタデータ API によって定義された型およびパブリック・インタフェースは、次の場所にあります。

```
$ORACLE_HOME/rdbms/admin/dbmsmeta.sql
```

参照 : 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

DBMS_METADATA プログラム・インタフェース

表 13-1 に、DBMS_METADATA プログラム・インタフェース・プロシージャを示します。

表 13-1 DBMS_METADATA プロシージャ : プログラム・インタフェース

PL/SQL プロシージャ	構文	説明
DBMS_ METADATA.OPEN()	FUNCTION **open** (object_type IN VARCHAR2, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE') RETURN NUMBER;	取り出されるオブジェクト型、オブジェクト・メタデータのバージョンおよびオブジェクト・モデルを指定します。戻り値は、後続のコールで使用される一連のオブジェクトの不透明なコンテキスト・ハンドルです。
DBMS_METADATA.SET_ FILTER()	PROCEDURE **set_filter** (handle IN NUMBER, name IN VARCHAR2, value IN VARCHAR2); PROCEDURE **set_filter** (handle IN NUMBER, name IN VARCHAR2, value IN BOOLEAN DEFAULT TRUE);	取り出されるオブジェクトに、オブジェクト名やスキーマなどの制限を指定します。INDEX、TRIGGER などの依存オブジェクトに対して、ベース・オブジェクトを指定できます。
DBMS_METADATA.SET_ COUNT()	PROCEDURE **set_count** (handle IN NUMBER, value IN NUMBER);	単一の FETCH_xxx コールで取り出されるオブジェクトの数を指定します。デフォルトでは、各 FETCH_xxx コールに 1 つのオブジェクトが戻されます。
DBMS_METADATA.GET_ QUERY()	FUNCTION **get_query** (handle IN NUMBER) RETURN VARCHAR2;	FETCH_xxx で使用される 1 つ以上の問合せのテキストを戻します。デバッグに有効です。
DBMS_METADATA.SET_ PARSE_ITEM()	PROCEDURE **set_parse_item** (handle IN NUMBER, name IN VARCHAR2);	出力の解析を有効にし、解析して戻すオブジェクト属性を指定します。これによって、コール側は、主要な属性に対して SQL DDL を解析する必要がなくなります。

表 13-1 DBMS_METADATA プロシージャ : プログラム・インタフェース（続き）

PL/SQL プロシージャ	構文	説明
DBMS_METADATA.ADD_TRANSFORM()	FUNCTION **add_transform** (handle IN NUMBER, name IN VARCHAR2 encoding IN VARCHAR2 DEFAULT NULL)) RETURN NUMBER;	取り出されたオブジェクトの XML 表示に対して FETCH_xxx が適用する変換を指定します。複数の変換を追加できます。デフォルトでは、変換が追加されることなく、オブジェクトが XML 文書として戻されます。戻された文書を変換する XSLT スクリプトを指定するには、ADD_TRANSFORM をコールします。「DDL」を指定すると、オブジェクトの作成 DDL が後続の FETCH_xxx コールから戻されます。ADD_TRANSFORM は、OPEN によって戻される変換ハンドルとは異なる不透明な変換ハンドルを戻します。 次のいずれかの場合は、エンコーディングを指定します。 ■ 外部 URL が指す XSL スタイルシートが、UTF-8 のサブセットではないキャラクタ・セットでエンコードされている。 ■ DB 内部 URL が指す XSL スタイルシートが、データベースのキャラクタ・セットのサブセットではないキャラクタ・セットでエンコードされている。
DBMS_METADATA.SET_TRANSFORM_PARAM()	PROCEDURE **set_transform_param** (transform_handle IN NUMBER, name IN VARCHAR2, value IN VARCHAR2); PROCEDURE **set_transform_param** (transform_handle IN NUMBER, name IN VARCHAR2, value IN BOOLEAN DEFAULT TRUE);	ADD_TRANSFORM から戻された *transform_handle* によって識別される XSLT スタイルシートにパラメータを指定します。 DDL 変換の場合、これらのパラメータで DDL のフォームを変更します。たとえば、列の制約または ALTER TABLE 文として、制約を要求することができます。

表 13-1 DBMS_METADATA プロシージャ : プログラム・インタフェース（続き）

PL/SQL プロシージャ	構文	説明
DBMS_METADATA.FETCH_xxx()	FUNCTION **fetch_xml** (handle IN NUMBER) RETURN XMLType; FUNCTION **fetch_ddl** (handle IN NUMBER) RETURN sys.ku$_ddls; FUNCTION **fetch_clob** (handle IN NUMBER) RETURN CLOB; PROCEDURE **fetch_clob** (handle IN NUMBER, doc IN OUT NOCOPY CLOB);	FETCH_xxx ルーチンは、OPEN、SET_FILTER、SET_COUNT、ADD_TRANSFORM などによって指定された規準に一致するオブジェクトのメタデータを戻します。 FETCH_XML および FETCH_DDL は、メタデータをそれぞれ XML および SQL DDL として戻します。FETCH_CLOB ルーチンは、指定された変換が示すとおり、XML または DDL のどちらか一方を戻します。 これらのルーチンが使用する型については、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
DBMS_METADATA.CLOSE()	PROCEDURE **close** (handle IN NUMBER);	OPEN によって戻されたハンドルを無効にし、それに対応付けられた状態をクリーンアップします。

参照 :

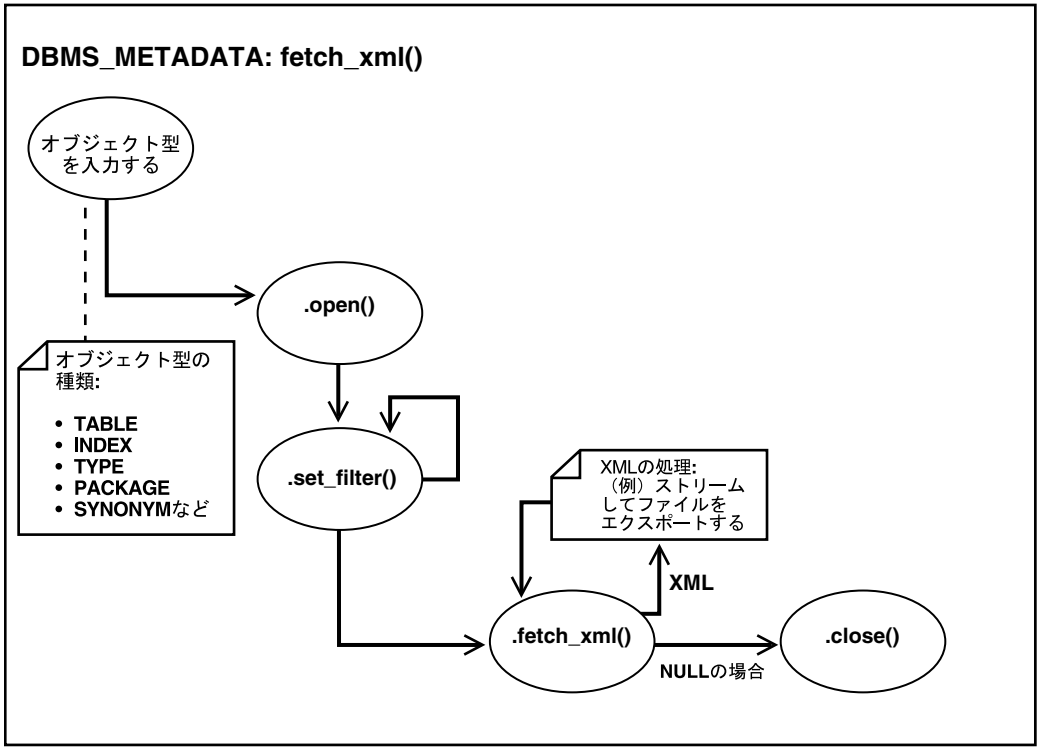
- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- XMLType の詳細は、第 5 章「XML に対するデータベース・サポート」を参照してください。

DBMS_METADATA.FETCH_XML

図 13-1 に、DBMS_METADATA.FETCH_XML() の使用方法を示します。

1. DBMS_METADATA.OPEN() を使用して、オブジェクト型をオープンします。
2. DBMS_METADATA.SET_FILTER() を使用して、取り出すオブジェクトを指定します。
3. DBMS_METADATA.FETCH_XML() を使用して、基準を満たす各オブジェクトのメタデータを XML 文書としてフェッチします。
4. この操作の結果が NULL の場合、DBMS_METADATA.CLOSE() を使用します。

図 13-1 DBMS_METADATA.FETCH_XML() の使用

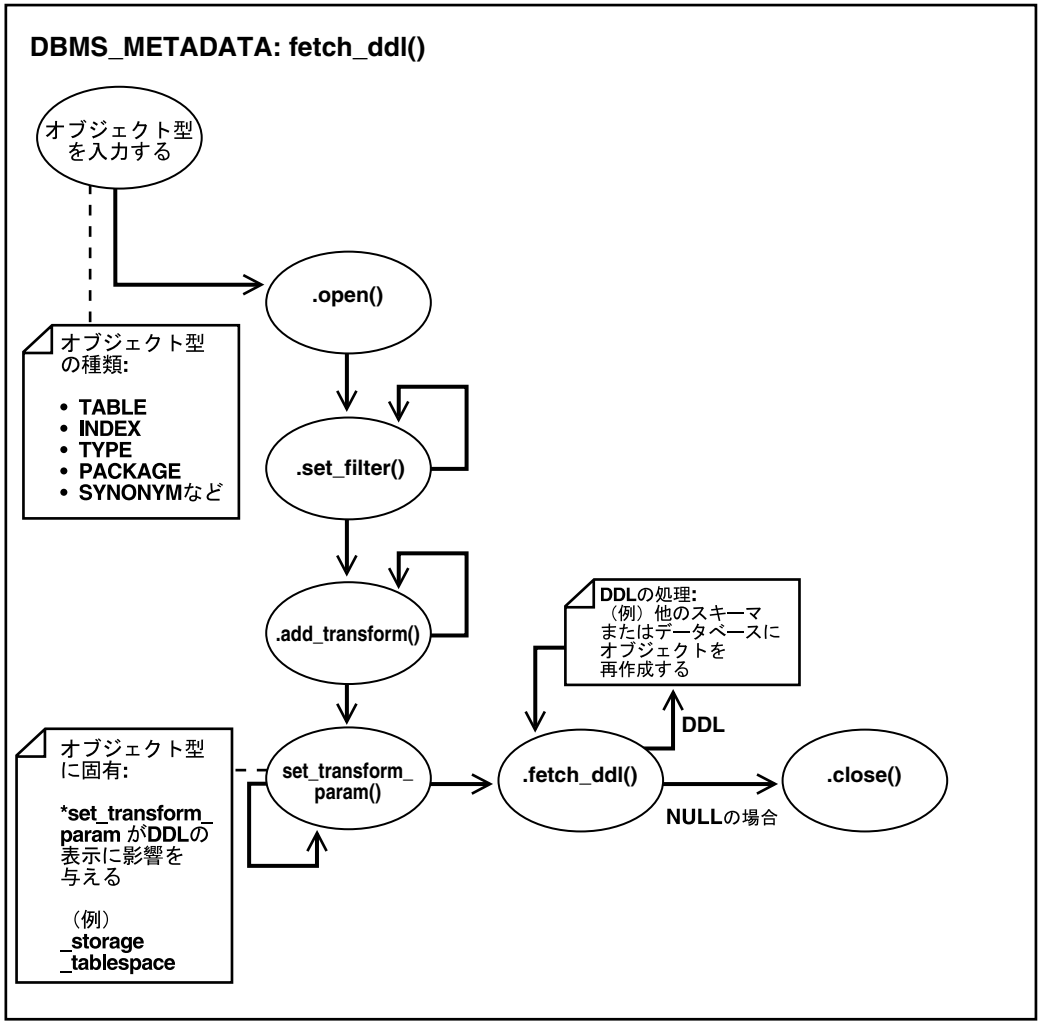


DBMS_METADATA.FETCH_DDL()

図 13-2 に、DBMS_METADATA.FETCH_DDL() の使用方法を示します。

1. DBMS_METADATA.OPEN() を使用して、オブジェクト型をオープンします。
2. DBMS_METADATA.SET_FILTER() を使用して、取り出すオブジェクトを指定します。
3. 出力に対して起動される変換を指定します。
   - 変換を追加するには、DBMS_METADATA.ADD_TRANSFORM() を使用します。最後に追加する変換は、「DDL」変換である必要があります。
4. DBMS_METADATA.SET_TRANSFORM_PARAM() を使用すると、DDL をカスタマイズできます。たとえば、表の定義の記憶域句を排除できます。変換パラメータは、選択されたオブジェクト型に固有です。
5. DBMS_METADATA.FETCH_DDL() を使用して、DDL をフェッチします。
6. この操作の結果が NULL の場合、DBMS_METADATA.CLOSE() を使用します。

図 13-2 DBMS_METADATA.FETCH_DDL() の使用



パフォーマンスのヒント

この項では、メタデータ API のプログラム・インタフェースを使用する場合に、パフォーマンスを向上させる方法について説明します。

1. 1 つの型のすべてのオブジェクトをフェッチしてから、次の型をフェッチします。たとえば、ユーザーのスキーマにあるすべてのオブジェクトの定義を取り出すには、まずすべての表、次にすべての索引、そしてすべてのトリガーのようにフェッチします。この方法は、OPEN コンテキストをネストする方法（1 つの表をフェッチして、その表のすべての索引、権限付与、トリガーをフェッチし、次の表をフェッチして、その表のすべての索引、権限付与、トリガーをフェッチする、という方法）より、はるかに高速にフェッチできます。この章の最後にある例では、多くのプログラム・コールを詳しく示すためのみに、後者（OPEN コンテキストをネストする方法）の非効率的な方法を使用しています。
2. SET_COUNT プロシージャを使用して、1 度に複数のオブジェクトを取り出します。これによって、多くの冗長なファンクション・コールを排除するだけでなく、サーバー・ラウンドトリップを最小限に抑えることができます。
3. FETCH_CLOB は、ファンクション・フォームではなくプロシージャ・フォームを使用します。プロシージャ・フォームは、IN OUT NOCOPY 指定子を介した参照によって出力 CLOB を戻します。ファンクション・フォームは、追加の LOB コピーを必要とする値によって出力 CLOB を戻します。
4. メタデータ API をコールする PL/SQL パッケージを作成する場合、LOB 変数および個別のファンクション内ではなく、パッケージの適用範囲に、SYS.KU$_DDLS などの LOB を格納するオブジェクトを宣言します。これによって、高コストな操作である、ファンクションの開始および終了における LOB の存続時間構造の作成および削除を排除できます。

   参照：『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

DBMS_METADATA ブラウザ・インタフェース

DBMS_METADATA ブラウザ・インタフェースは、GET_XML ファンクションおよび GET_DDL ファンクションによって提供されます。

表 13-2 に、ブラウザ API、その構文および簡単な説明を示します。

表 13-2 DBMS_METADATA プロシージャ：ブラウザ・インタフェース

PL/SQL プロシージャ名	構文	説明
DBMS_METADATA.GET_xxx()	FUNCTION **get_xml** (object_type IN VARCHAR2, name IN VARCHAR2, schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT NULL) RETURN CLOB; FUNCTION **get_ddl** (object_type IN VARCHAR2, name IN VARCHAR2, schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT 'DDL') RETURN CLOB;	単一オブジェクトのメタデータを戻す方法を指定します。各 GET_xxx コールは、OPEN、1 ～ 2 つの SET_FILTER コール、オプションとして ADD_TRANSFORM、FETCH_xxx、CLOSE で構成されています。 object_type パラメータは、OPEN と同じセマンティクスを持ちます。schema および name は、フィルタの適用のために使用されます。 変換を指定すると、セッション・レベルの変換フラグが継承されます。

例

次の SQL*Plus コマンドは、現行のユーザーのスキーマにあるすべての表に対する作成 DDL を表示します。

```
SQL> SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
```

メタデータ API の例 : 表に対する DDL の取得

ここでは、詳細なメタデータ API プログラミング例である PAYROLL_DEMO を示します。PAYROLL_DEMO は、「PAYROLL」で始まる MDDEMO スキーマ内のすべての表に対する DDL を取り出します。その後、表に定義されている権限付与、索引およびトリガーに対する DDL をフェッチします。このスクリプトは、ユーザーの Oracle ホーム・ディレクトリ内の rdbms/demo/mddemo.sql ファイルにあります。

mddemo.sql

```
-- This script demonstrates how to use the Metadata API. It first
-- establishes a schema (MDDEMO) and some payroll users, then creates three
-- payroll-like tables within it along with associated indexes, triggers
-- and grants.

-- It then creates a package PAYROLL_DEMO that shows common usage of the
-- Metadata API. The procedure GET_PAYROLL_TABLES retrieves the DDL for the
-- two tables in this schema that start with 'PAYROLL' then for each one,
-- retrieves the DDL for its associate dependent objects; indexes, grants
-- and triggers. All the DDL is written to a table named "MDDEMO"."DDL".

-- First, Install the demo. cd to rdbms/demo:
-- > sqlplus system/manager
-- SQL> @mddemo

-- Then, run it.
-- > sqlplus mddemo/mddemo
-- SQL> set long 40000
-- SQL> set pages 0
-- SQL> call payroll_demo.get_payroll_tables();
-- SQL> select ddl from DDL order by seqno;

Rem Set up schema for demo pkg. PAYROLL_DEMO.

connect system/manager
drop user mddemo cascade;
drop user mddemo_clerk cascade;
drop user mddemo_mgr cascade;

create user mddemo identified by mddemo;
GRANT resource, connect, create session
     , create table
     , create procedure
     , create sequence
     , create trigger
     , create view
     , create synonym
```

```
     , alter session
TO mddemo;

create user mddemo_clerk identified by clerk;
create user mddemo_mgr identified by mgr;

connect mddemo/mddemo

Rem Create some payroll-like tables...

create table payroll_emps
( lastname varchar2(60) not null,
  firstname varchar2(20) not null,
 mi varchar2(2),
 suffix varchar2(10),
 DOB date not null,
 badge_no number(6) primary key,
 exempt varchar(1) not null,
 salary number (9,2),
 hourly_rate number (7,2)
)
/
create table payroll_timecards
 badge_no number(6) references payroll_emps (badge_no),
 week number(2),
job_id number(5),
hours_worked number(4,2)
)
/
-- This is a dummy table used only to show that tables NOT starting with
-- 'PAYROLL' are NOT retrieved by payroll_demo.get_payroll_tables

create table audit_trail
(action_time DATE,
lastname VARCHAR2(60),
action LONG
)
/

Rem Then, create some grants...

grant update (salary,hourly_rate) on payroll_emps to mddemo_clerk;
grant ALL on payroll_emps to mddemo_mgr with grant option;

grant insert,update on payroll_timecards to mddemo_clerk;
grant ALL on payroll_timecards to mddemo_mgr with grant option;
```

```
Rem Then, create some indexes...

create index i_payroll_emps_name on payroll_emps(lastname);
create index i_payroll_emps_dob on payroll_emps(DOB);

create index i_payroll_timecards_badge on payroll_timecards(badge_no);

Rem Then, create some triggers (and required procedure)...

create or replace procedure check_sal( salary in number) as
begin
  return;  -- Fairly loose security here...
end;
/

create or replace trigger salary_trigger before insert or update of salary on
payroll_emps
for each row when (new.salary > 150000)
call check_sal(:new.salary)
/

create or replace trigger hourly_trigger before update of hourly_rate on payroll_
emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/

--
-- Set up a table to hold the generated DDL
--
CREATE TABLE ddl (ddl CLOB, seqno NUMBER);

Rem Finally, create the PAYROLL_DEMO package itself.

CREATE OR REPLACE PACKAGE payroll_demo AS

   PROCEDURE get_payroll_tables;
END;
/
CREATE OR REPLACE PACKAGE BODY payroll_demo AS

-- GET_PAYROLL_TABLES: Fetch DDL for payroll tables and their dependent objects

PROCEDURE  get_payroll_tables IS

tableOpenHandle NUMBER;
depObjOpenHandle NUMBER;
```

```
tableTransHandle  NUMBER;
indexTransHandle NUMBER;
schemaName VARCHAR2(30);
tableName VARCHAR2(30);
tableDDLs sys.ku$_ddls;
tableDDL sys.ku$_ddl;
parsedItems   sys.ku$_parsed_items;
depObjDDL CLOB;
seqNo NUMBER := 1;

TYPE obj_array_t IS VARRAY(3) OF VARCHAR2(30);

-- Load this array with the dependent object classes to be retrieved...
obj_array obj_array_t := obj_array_t('OBJECT_GRANT', 'INDEX', 'TRIGGER');

BEGIN

-- Open a handle for tables in the current schema.
  tableOpenHandle := dbms_metadata.open('TABLE');

-- Tell mdAPI to retrieve one table at a time. This call is not actually
-- necessary since 1 is the default... just showing the call.
  dbms_metadata.set_count(tableOpenHandle, 1);

-- Retrieve tables whose name starts with 'PAYROLL'. When the filter is
-- 'NAME_EXPR', the filter value string must include the SQL operator. This
-- gives the caller flexibility to use LIKE, IN, NOT IN, subqueries, etc.
  dbms_metadata.set_filter(tableOpenHandle, 'NAME_EXPR', 'LIKE ''PAYROLL%''');

-- There are no index-organized tables in the MDDEMO schema, so tell the API.
-- This eliminates one of the views it'll need to look in.
  dbms_metadata.set_filter(tableOpenHandle, 'IOT', FALSE);

-- Tell the mdAPI to parse out each table's schema and name separately so we
-- can use them to set up the calls to retrieve its dependent objects.
  dbms_metadata.set_parse_item(tableOpenHandle, 'SCHEMA');
  dbms_metadata.set_parse_item(tableOpenHandle, 'NAME');

-- Add the DDL transform so we get SQL creation DDL
  tableTransHandle := dbms_metadata.add_transform(tableOpenHandle, 'DDL');

-- Tell the XSL stylesheet we don't want physical storage information (storage,
-- tablespace, etc), and that we want a SQL terminator on each DDL. Notice that
-- these calls use the transform handle, not the open handle.
  dbms_metadata.set_transform_param(tableTransHandle,
     'SEGMENT_ATTRIBUTES', FALSE);
  dbms_metadata.set_transform_param(tableTransHandle,
```

```
    'SQLTERMINATOR', TRUE);

-- Ready to start fetching tables. We use the FETCH_DDL interface (rather than
-- FETCH_XML or FETCH_CLOB). This interface returns a SYS.KU$_DDLS; a table of
-- SYS.KU$_DDL objects. This is a table because some object types return
-- multiple DDL statements (like types / pkgs which have create header and
-- body statements). Each KU$_DDL has a CLOB containing the 'CREATE foo'
-- statement plus a nested table of the parse items specified. In our case,
-- we asked for two parse items; Schema and Name. (NOTE: See admin/dbmsmeta.sql
-- for a more detailed description of these types)

  LOOP
    tableDDLs := dbms_metadata.fetch_ddl(tableOpenHandle);
    EXIT WHEN tableDDLs IS NULL;  -- Get out when no more payroll tables

-- In our case, we know there is only one row in tableDDLs (a KU$_DDLS tbl obj)
-- for the current table. Sometimes tables have multiple DDL statements;
-- eg, if constraints are applied as ALTER TABLE statements, but we didn't ask
-- for that option. So, rather than writing code to loop through tableDDLs,
-- we'll just work with the 1st row.
--
-- First, write the CREATE TABLE text to our output table then retrieve the
-- parsed schema and table names.
    tableDDL := tableDDLs(1);
    INSERT INTO ddl VALUES(tableDDL.ddltext, seqNo);
    seqNo := seqNo + 1;
    parsedItems := tableDDL.parsedItems;

-- Must check the name of the returned parse items as ordering isn't guaranteed
    FOR i IN 1..2 LOOP
      IF parsedItems(i).item = 'SCHEMA'
      THEN
        schemaName := parsedItems(i).value;
      ELSE
        tableName  := parsedItems(i).value;
      END IF;
    END LOOP;

-- Now, we want to retrieve all the dependent objects defined on the current
-- table: indexes, triggers and grants. Since all 'dependent' object types
-- have BASE_OBJECT_NAME and BASE_OBJECT_SCHEMA in common as filter criteria,
-- we'll set up a loop to get all objects of the 3 types, just changing the
-- OPEN context in each pass through the loop. Transform parameters are
-- different for each object type, so we'll only use one that's common to all;
-- SQLTERMINATOR.

    FOR i IN 1..3 LOOP
```

```
      depObjOpenHandle := dbms_metadata.open(obj_array(i));
      dbms_metadata.set_filter(depObjOpenHandle,'BASE_OBJECT_SCHEMA',
       schemaName);
      dbms_metadata.set_filter(depObjOpenHandle,'BASE_OBJECT_NAME',tableName);

-- Add the DDL transform and say we want a SQL terminator
      indexTransHandle := dbms_metadata.add_transform(depObjOpenHandle, 'DDL');
      dbms_metadata.set_transform_param(indexTransHandle,
        'SQLTERMINATOR', TRUE);

-- Retrieve dependent object DDLs as CLOBs and write them to table DDL.
      LOOP
        depObjDDL := dbms_metadata.fetch_clob(depObjOpenHandle);
        EXIT WHEN depObjDDL IS NULL;
        INSERT INTO ddl VALUES(depObjDDL, seqNo);
        seqNo := seqNo + 1;
      END LOOP;

-- Free resources allocated for current dependent object stream.
      dbms_metadata.close(depObjOpenHandle);

    END LOOP; -- End of fetch dependent objects loop

  END LOOP;   -- End of fetch table loop

-- Free resources allocated for table stream and close output file.
  dbms_metadata.close(tableOpenHandle);
  RETURN;

END;  -- of procedure get_payroll_tables

END payroll_demo;
/
```

PAYROLL_DEMO の出力

ここでは、mddemo.payroll_demo.get_payroll_tables プロシージャを実行して取得した出力を示します。この出力は、次の問合せをユーザー MDDEMO として実行することによって取得されます。

```
SQL> SELECT ddl FROM ddl ORDER BY seqno;

CREATE TABLE "MDDEMO"."PAYROLL_EMPS"
   (    "LASTNAME" VARCHAR2(60) NOT NULL ENABLE,
        "FIRSTNAME" VARCHAR2(20) NOT NULL ENABLE,
        "MI" VARCHAR2(2),
        "SUFFIX" VARCHAR2(10),
        "DOB" DATE NOT NULL ENABLE,
        "BADGE_NO" NUMBER(6,0),
        "EXEMPT" VARCHAR2(1) NOT NULL ENABLE,
        "SALARY" NUMBER(9,2),
        "HOURLY_RATE" NUMBER(7,2),
 PRIMARY KEY ("BADGE_NO") ENABLE
   ) ;

  GRANT UPDATE ("SALARY") ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_CLERK";
  GRANT UPDATE ("HOURLY_RATE") ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_CLERK";
  GRANT ALTER ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT DELETE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INDEX ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INSERT ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT SELECT ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT UPDATE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT REFERENCES ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT ON COMMIT REFRESH ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT
OPTION;
  GRANT QUERY REWRITE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTI
ON;

  CREATE INDEX "MDDEMO"."I_PAYROLL_EMPS_DOB" ON "MDDEMO"."PAYROLL_EMPS" ("DOB")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;



  CREATE INDEX "MDDEMO"."I_PAYROLL_EMPS_NAME" ON "MDDEMO"."PAYROLL_EMPS" ("LASTN
AME")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
```

```
  CREATE OR REPLACE TRIGGER hourly_trigger before update of hourly_rate on payro
ll_emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/
ALTER TRIGGER "MDDEMO"."HOURLY_TRIGGER" ENABLE;

  CREATE OR REPLACE TRIGGER salary_trigger before insert or update of salary on
payroll_emps
for each row  WHEN (new.salary > 150000)  CALL check_sal(:new.salary)
/
ALTER TRIGGER "MDDEMO"."SALARY_TRIGGER" ENABLE;


CREATE TABLE "MDDEMO"."PAYROLL_TIMECARDS"
   (    "BADGE_NO" NUMBER(6,0),
        "WEEK" NUMBER(2,0),
        "JOB_ID" NUMBER(5,0),
        "HOURS_WORKED" NUMBER(4,2),
 FOREIGN KEY ("BADGE_NO")
  REFERENCES "MDDEMO"."PAYROLL_EMPS" ("BADGE_NO") ENABLE
   ) ;

  GRANT INSERT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_CLERK";
  GRANT UPDATE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_CLERK";
  GRANT ALTER ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT DELETE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION
;
  GRANT INDEX ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INSERT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION
;
  GRANT SELECT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION
;
  GRANT UPDATE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION
;
  GRANT REFERENCES ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OP
TION;
  GRANT ON COMMIT REFRESH ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH G
RANT OPTION;
  GRANT QUERY REWRITE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT
 OPTION;

  CREATE INDEX "MDDEMO"."I_PAYROLL_TIMECARDS_BADGE" ON "MDDEMO"."PAYROLL_TIMECAR
DS" ("BADGE_NO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
```

14

OracleAS Reports Services および XML

この章の内容は次のとおりです。

OracleAS Reports Services および XML の概要

OracleAS Reports Services は、次の XML サポートを提供します。

- レポートの XML での出力およびカスタマイズ（Reports 6*i* 以上）
  - OracleAS Reports Services を使用した実行中における XML 出力の作成

    14-4 ページの「OracleAS Reports Services を使用した実行中における XML 出力の作成」を参照してください。
  - 実行時のレポート定義のカスタマイズ

    14-6 ページの「実行時のレポート定義のカスタマイズ」を参照してください。
  - XML レポート定義を使用したレポートの一括変更

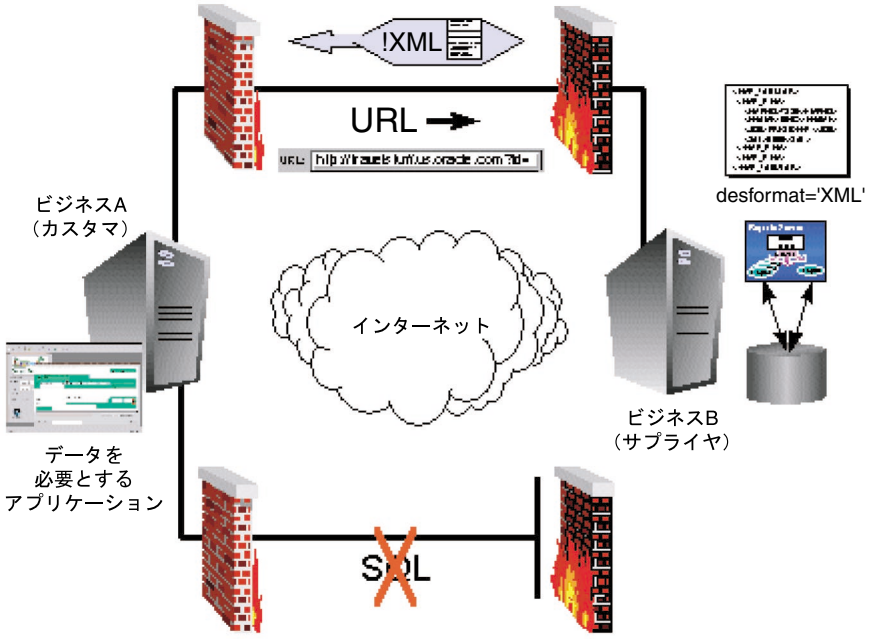
    14-12 ページの「XML レポート定義を適用したレポートの一括変更」を参照してください。
- レポートからの XML データ・ソースの読取り（Reports 9*i* 以上）

  14-17 ページの「データソースとしての XML の使用」を参照してください。

B2B データ交換 : レポートで XML を使用する理由

図 14-1 に、パートナとの情報の共有、およびより適時にデータを送信するための XML の使用方法を示します。OracleAS Reports Server を使用すると、XML ファイルを自動的に生成できます。Web 上で Oracle Reports を起動するために必要なことは、URL を指定することのみです。データを問い合せるためのレポート・モジュールを定義することで、サプライヤは、コール側の E-Commerce アプリケーションへリアルタイムで情報をストリームできます。

図 14-1 OracleAS Reports Services で XML を使用する理由



参照：

- 『Oracle9*i* ケース・スタディ -XML アプリケーション』の「XSL を使用した Discoverer 4*i* Viewer のカスタマイズ」を参照してください。
- 『Oracle Reports Developer レポート作成ガイド リリース 6*i*』以降のマニュアルを参照してください。
- 『Oracle Reports Developer パブリッシング・レポート リリース 6*i*』以降のマニュアルを参照してください。
- 『Oracle Reports Developer Release 6*i* Reference Manual』以降のマニュアルを参照してください。
- オンライン・ヘルプも参照してください。オンライン・ヘルプには、様々な方法でアクセスできます。デスクトップからは、「スタート」>「Oracle Forms and Reports Doc」を選択して起動します。

OracleAS Reports Services の実行要件

OracleAS Reports Service（Oracle Reports）は、OracleAS の一部です。OracleAS Reports Service は、OracleAS が提供する環境およびその他のサービスとシームレスに統合されています。

参照： http://otn.oracle.com/products/reports/ を参照してください。

OracleAS Reports Services を使用した実行中における XML 出力の作成

データ・モデルは、レポート用の XML 出力の構造を決定します。XML 出力は、ビジュアルなレイアウトに依存しません。

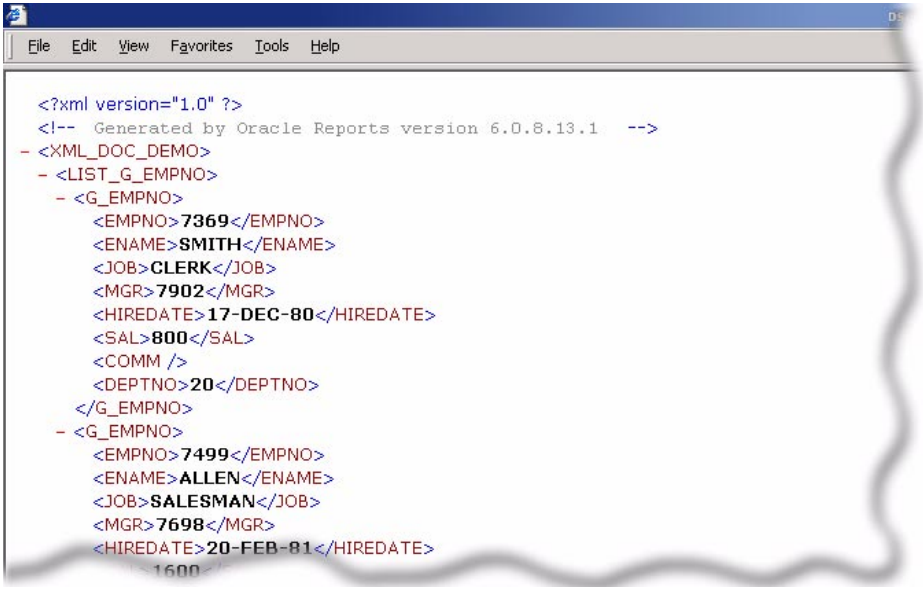
- データ・モデル・オブジェクト用の Property パレットで使用可能な XML プロパティを編集することによって、レポートの XML 出力構造を変更できます。
- 現在、Report Builder は、XML 出力の表示情報（カラー、フォント、物理レイアウトなど）をサポートしていません。XSL を使用して、独自の表示情報を提供できます。これはプロローグで指定できます。
- コマンドラインから、引数 `DESFORMAT`=XML を使用して、XML 出力を生成できます。

参照： `DESFORMAT` の詳細は、『Oracle Reports Developer Release 6*i* Reference Manual』を参照してください。

データ交換フォーマットとしての XML

B2B パートナへ、レポートを XML で送信する必要がある場合があります。図 14-2 に、XML のレポート例を示します。これは、1 つのパラメータを変更して、HTML ではなく XML でレポートを出力することを Reports サーバーに通知することによって取得されます。

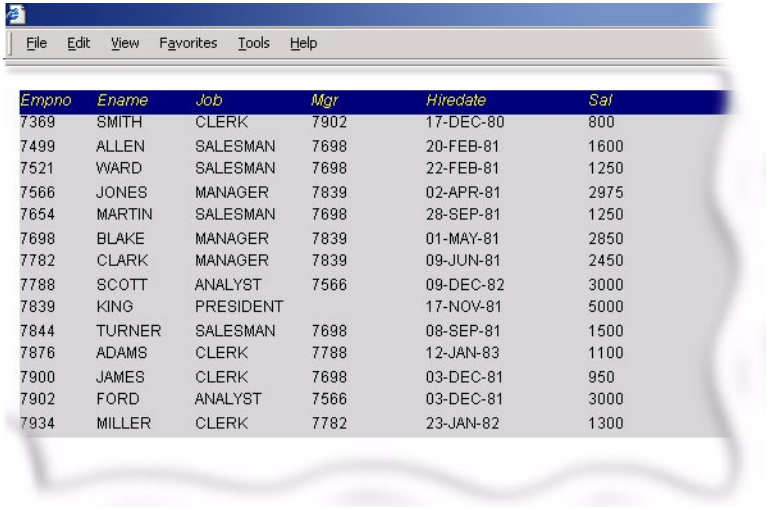
図 14-2 XML で示されたレポート例



XSL スタイルシートを使用した XML 出力のフォーマット

図 14-3 に、図 14-2 に示す同じ XML レポートに XSL スタイルシートを適用した結果を示します。この場合、データのレポート例が、HTML で要求されています。

図 14-3 XSL スタイルシート適用後のレポート例



Empno	Ename	Job	Mgr	Hiredate	Sal
7369	SMITH	CLERK	7902	17-DEC-80	800
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600
7521	WARD	SALESMAN	7698	22-FEB-81	1250
7566	JONES	MANAGER	7839	02-APR-81	2975
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250
7698	BLAKE	MANAGER	7839	01-MAY-81	2850
7782	CLARK	MANAGER	7839	09-JUN-81	2450
7788	SCOTT	ANALYST	7566	09-DEC-82	3000
7839	KING	PRESIDENT		17-NOV-81	5000
7844	TURNER	SALESMAN	7698	08-SEP-81	1500
7876	ADAMS	CLERK	7788	12-JAN-83	1100
7900	JAMES	CLERK	7698	03-DEC-81	950
7902	FORD	ANALYST	7566	03-DEC-81	3000
7934	MILLER	CLERK	7782	23-JAN-82	1300

実行時のレポート定義のカスタマイズ

OracleAS Reports Services によって、各レポート固有のバージョンを作成するかわりに、変更を別のカスタマイズ・ファイルに格納し、外部化できます。元のレポート定義を変更することなく、レポートの出力を固有のユーザーやグループ用にカスタマイズできます。

カスタマイズ・ファイルは、既存のレポート（.RDF または .XML）に適用されるレポート定義です。フィールドの日付書式マスクまたはバックグラウンド・カラーなどの、既存レポート・オブジェクトの特定の特性を変更できます。カスタマイズ・ファイルを使用して、別のレポートに新しいオブジェクトを追加することもできます。

> **参照：** 14-17 ページの「データソースとしての XML の使用」を参照してください。

XML カスタマイズの適用

XML レポート定義を実行時に .RDF または .XML ファイルに適用するには、次のオプションのいずれかを使用します。

- **ランタイム・オプション:** CUSTOMIZE コマンドライン引数を使用します。CUSTOMIZE は、RWCLI60、RWRUN60、RWBLD60、RWCON60 および URL レポート要求で使用できます。
- **組込みオプション:** 提供された PL/SQL パッケージ SRW.APPLY_DEFINITION（組込みファンクション）を使用します。

単一のカスタマイズ・ファイルの適用

次のコマンドラインは、XML のレポート定義である emp.xml を .RDF ファイルの emp.rdf に適用するジョブ要求を、OracleAS Reports Services に送信します。

```
rwcli60 report=emp.rdf customize=e:¥myreports¥emp.xml
  userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
  server=repserver
```

Runtime によるレポート OracleAS Reports Services Runtime を使用している場合、同等のコマンドラインは次のとおりです。

```
rwrun60 userid=username/password@mydb report=emp.rdf
   customize=e:¥myreports¥emp.xml destype=file desname=emp.pdf
   desformat=PDF
```

XML レポート定義をテストする場合、追加の引数を使用してレポート要求を実行し、トレース・ファイルを作成すると効果的です。たとえば、次のコマンドを実行します。

```
tracefile=emp.log tracemode=trace_replace traceopt=trace_app
```

トレース・ファイルから、レポート・オブジェクトの作成およびフォーマットの詳細なリストを得ることができます。

複数の XML カスタマイズ・ファイルの適用

CUSTOMIZE コマンドライン引数でリストを指定することによって、複数の XML レポート定義を実行時にレポートに適用できます。次のコマンドラインは、2 つの XML レポート定義である emp0.xml および emp1.xml を .RDF ファイルの emp.rdf に適用するジョブ要求を、OracleAS Reports Services に送信します。

```
rwcli60 report=emp.rdf
  customize="(e:¥corp¥myreports¥emp0.xml,
  e:¥corp¥myreports¥emp1.xml)"
  userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
  server=repserver
```

Runtime によるレポート　OracleAS Reports Services Runtime を使用している場合、同等のコマンドラインは次のとおりです。

```
rwrun60 report=emp.rdf
  customize="(e:¥corp¥myreports¥emp0.xml,
  e:¥corp¥myreports¥emp1.xml)"
  userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
```

PL/SQL を使用した XML カスタマイズ・ファイルの適用

XML レポート定義を PL/SQL を使用して .RDF ファイルに適用するには、Before Form トリガーまたは After Form トリガーに、提供された PL/SQL パッケージ（組込みファンクション）である SRW.ADD_DEFINITION および SRW.APPLY_DEFINITION を使用します。

- **ファイルに格納された XML 定義の適用（組込み）:** ファイル・システムに格納された XML をレポートに適用するには、レポートの Before Form トリガーまたは After Form トリガーに、提供された PL/SQL パッケージである SRW.APPLY_DEFINITION を使用します。

  ```
  SRW.APPLY_DEFINITION ('d:¥orant¥tools¥doc60¥us¥rbbr60¥cond.xml');
  ```

  レポートを実行すると、トリガーが実行され、指定した XML ファイルがレポートに適用されます。

- **メモリーに格納された XML 定義の適用:** メモリーに XML レポート定義を作成するには、SRW.ADD_DEFINITION を使用してドキュメント・バッファに定義を追加し、SRW.APPLY_DEFINITION を使用して適用します。

XML を使用した実行時のレポートのカスタマイズ

Reports 6*i* 以上の Oracle Reports Developer を使用すると、実行時にレポートの外観および内容を変更できます。これを行うには、XML タグから構築されたレポート定義ファイル（RDF）と既存の RDF ファイルを実行時にマージして、この組合せを実行します。XML レポート定義は、次のような他のタスクにも使用できます。

- レポート変換ツール（RWCON60）による XML レポート定義を使用した一括変更。RWCON60 は、提供された PL/SQL パッケージ（組込みファンクション）です。
- 既存の RDF ファイルを使用することなく、それ自体で実行できる XML での完全なレポート定義の構築。

次の例は、実行時に XML を使用してレポートを変更する方法を示しています。

- 項目のビジュアルな属性の変更およびその書式マスクの変更
- 非データベース・フィールドの属性の修正
- 1 つの RDF ファイルからの複数言語レポートの作成
- RDF ファイルの定義を使用しない、XML ファイルからのレポートの作成

カスタマイズの準備

次の例 1 ～ 5 を使用する前に、XML カスタマイズが適用されるデフォルトのレポート（「select*from emp」、「select all columns」、「Corporate 1」テンプレートを使用して作成した表レポート）を作成する必要があります。このレポートには任意の名前を付けることができますが、次の例では、emp_report といいます。

これらの変更を有効にするには、`modify.xml` というファイルを作成し、それに現在の例の XML コードをコピーすると最適です。このように、次のコマンドラインを使用して、いつでもレポートおよび変更を実行できます

```
rwrun60 report=emp_report userid=scott/tiger customize=modify.xml
```

次に示すスクリプトを変更する場合、XML 固有のすべての制限事項（特に、大文字 / 小文字の区別）に注意してください。

XML を使用したレポートのカスタマイズ例 1: F_EMPNO の変更および設定

次の例は、F_EMPNO フィールドを変更し、カラーを赤に設定します。

```
<report name="emp_report" DTDVersion="1.0">
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
  </section>
 </layout>
</report>
```

XML を使用したレポートのカスタマイズ例 2: F_EMPNO のテキスト・カラーの変更

例 2 は、F_EMPNO フィールドのテキスト・カラーを赤に変更し、F_HIREDATE フィールドの日付書式をドイツ語表記法に設定します。

```
<report name="emp_report" DTDVersion="1.0">
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
   <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
  </section>
 </layout>
</report>
```

XML を使用したレポートのカスタマイズ例 3: ボイラープレート・テキスト・オブジェクトの変更

この例は、ボイラープレート・テキスト・オブジェクトの変更方法を示しています。通常のレポート・レイアウト要素は、<layout>...</layout> タグで囲まれ、ボイラープレート要素および論理要素は、<customize>...</customize> タグで囲まれています。

```
<report name="emp_report" DTDVersion="1.0">
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
   <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
  </section>
 </layout>
 <customize>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Anst.Dat. </property>
   </properties>
  </object>
 </customize>
</report>
```

XML を使用したレポートのカスタマイズ例 4: SELECT * 問合せの置換

レポート定義のその他の要素と同様に、問合せは、XML ファイルによってカスタマイズできる要素です。この例では、レポート作成時に使用された問合せ（select * from emp）が、WHERE 句（select * from emp where deptno = 10）を使用する問合せによって置換されます。

ウィザードを使用して作成したレポート定義で最初に使用していたデータソース名と同じデータソース名を使用する必要があることに注意してください。それ以外のデータソース名は、どの反復フレームにも関連がないため使用されません。

```
<report name="emp_report" DTDVersion="1.0">
 <data>
  <dataSource name="Q_1">
   <select>
    select * from emp where deptno = 10
   </select>
  </dataSource>
 </data>
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
   <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
```

```
  </section>
 </layout>
 <customize>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Anst.Dat. </property>
   </properties>
  </object>
 </customize>
</report>
```

XML を使用したレポートのカスタマイズ例 5: フィールド S_SAL へのトリガーの追加

レポートのビジュアルな属性の変更に加えて、PL/SQL コードもカスタマイズすることによって、その論理を簡単に変更できます。この例では、作成したレポートにフォーマット・トリガーは含まれていません。この XML ファイルを適用することによって、:SAL の値が 2,500 未満の場合フィールドを非表示にする F_SAL フィールドへ、トリガーが追加されます。

```
<report name="emp_report" DTDVersion="1.0">
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
   <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
   <field name="F_SAL" source="SAL" formatTrigger="SAL_FORMAT"/>
  </section>
 </layout>
 <programUnits>
 <function name="SAL_FORMAT">
  <![CDATA[
    function sal_format return boolean
    is
    begin
     if :SAL > 2500 then
        return (true);
     else
        return (false);
     end if;
    end;
  ]]>
 </function>
 </programUnits>
 <customize>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
```

```
   <properties>
    <property name="textSegment"> Anst.Dat. </property>
   </properties>
  </object>
 </customize>
</report>
```

XML レポート定義を適用したレポートの一括変更

変更を外部化するレポートの機能を使用することによって、アップグレードおよびアプリケーションをサイト固有にする作業が簡単になります。XML 定義を適用し、結果の定義を新しい一意のモジュールとして保存できます。また、Reports Builder で各ファイルをオープンして変更することなく、簡単に更新およびアップグレードできます。

多数のレポートを更新するには、レポート変換ツールの RWCON60 で CUSTOMIZE コマンドライン引数を使用し、一括変更を行います。多数のレポートを繰返し変更する場合（たとえば、フィールドの書式マスクを変更する場合）、一括変更が有効です。Oracle Report Builder から、RWCON60 を一度実行して多数のレポートに同じ変更を実行できます。

次の例では、2 つの XML レポート定義 translate.xml および customize.xml を、3 つの .RDF ファイル inven.rdf、inven2.rdf および manu.rdf に適用します。

これは、変更された定義を次の新規ファイルに保存します。

- inven1_new.rdf
- inven2_new.rdf
- manu_new.rdf

```
rwcon60 username/password@mydb
  stype=rdffile
  source="(inven1.rdf, inven2.rdf, manu.rdf)"
  dtype=rdffile
  dest="(inven1_new.rdf, inven2_new.rdf, manu_new.rdf)"
  customize="(e:¥apps¥trans¥translate.xml,
  e:¥apps¥custom¥customize.xml)" batch=yes
```

1 つのマスターからの変更された RDF の作成

たとえば、ERP ベンダーが、顧客が各自のレポートを変更できるようにする必要があるとします。これには、フォント、カラーなどを変更する必要があります。Reports XML ベースのカスタマイズを使用することによって、ベンダーは、1 つの手順でアプリケーション全体にこのような変更を行えます。

これを行うには、前述の例に示すように、RWCON60 を使用して XML カスタマイズ・ファイルを適用し、そのファイルから新規の RDF ファイルまたは REP ファイルを作成します。これによって、異なるバージョンのレポートをメンテナンスする必要なく、各顧客に対してカスタマイズされたアプリケーションを作成できます。基本のレイアウトを含む 1 つのマスター・レポートのみ、またはオプションでロゴや会社名など顧客固有の情報に対するプレースホルダ・オブジェクトを作成し、固有のレポートを構築する各顧客用のカスタマイズ・ファイルを適用します。

単一の RDF からの複数バージョン・レポートの作成

複数言語レポートの作成は、常に困難な作業です。ベース・レポートを開発し、それをすべてのサポート言語に変換すると、メンテナンスの問題が発生する可能性があります。OracleAS Reports Services の XML カスタマイズ機能を使用すると、これを簡単に実行できます。

基本言語でレポートを作成し、ラベル・テキスト、データ・フォーマットおよび数値フォーマットなどの言語固有の設定を含む、様々な XML カスタマイズ・ファイルを適用します。これによって、様々な言語バージョンのレポートを簡単に作成できます。

XML を使用したレポートのカスタマイズ例 6: 様々な言語バージョンの作成

この例は、1 つのレポート定義から様々な言語バージョンを簡単に作成できる方法を示しています。複雑なコードを作成したり、同じレポートの複数のバージョンを作成する必要はありません。1 つのレイアウトを作成し、「言語 XML」を適用することによってそれをローカライズします。

この例では、emp_report を修正して、ドイツ語のボイラープレートおよび日付書式を取得します。これは、使用する地域の言語に簡単に変更できます。ドイツ語の用語を、必要な言語で使用されている用語に置き換えるだけで、そのレポートをローカライズできます。

```
<report name="emp_report" DTDVersion="1.0">
 <layout>
  <section name="main">
   <field name="F_EMPNO" source="EMPNO" textColor="red"/>
   <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
   <field name="F_DATE1_SEC2" source="Current Date" formatMask="dd.mm.yyyy"/>
  </section>
 </layout>
 <customize>
  <object name="B_EMPNO" type="REP_GRAPHIC_TEXT">
   <properties>
```

```
    <property name="textSegment"> Pers.No. </property>
   </properties>
  </object>
  <object name="B_ENAME" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Name </property>
   </properties>
  </object>
  <object name="B_JOB" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Pos. </property>
   </properties>
  </object>
  <object name="B_MGR" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Vorges. </property>
   </properties>
  </object>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Anst.Dat. </property>
   </properties>
  </object>
  <object name="B_SAL" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Geh. </property>
   </properties>
  </object>
  <object name="B_COMM" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Prov. </property>
   </properties>
  </object>
  <object name="B_DEPTNO" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Abt. </property>
   </properties>
  </object>
  <object name="B_DATE1_SEC2" type="REP_GRAPHIC_TEXT">
   <properties>
    <property name="textSegment"> Stand vom </property>
   </properties>
  </object>
 </customize>
</report>
```

アプリケーションが言語を動的に切り替える必要がある場合、実行時にカスタマイズを適用することもできます。

XML でのレポート定義の作成

XML レポート定義を使用すると、レポート定義全体を XML で作成し、他のレポートとは別に実行できます。これによって、Oracle Report Builder を使用することなくレポートを構築できます。独自のフロントエンドを使用して、必要な XML を生成することもできます。これによってユーザーは、独自のレポートを動的に構築できます。

OracleAS Reports Services を使用すると、Report-Builder でレポートを作成して XML 形式で保存し、開始点として使用できます。その後、XML を変更するか、または XML を作成するアプリケーション用のテンプレートとして使用できます。

XML を使用したレポートのカスタマイズ例 7: XML 定義のみからのレポート

この例には、空の RDF ファイルが必要です。空のレポートを作成し、空の .rdf として保存します。次に、必要な変更を含む次の XML を適用し、XML 定義のみからレポートを作成します。

```
rwrun60 report=empty userid=scott/tiger customize=modify.xml
```

この例では、テンプレート corp1.tdf を使用して、EMPNO、ENAME、SAL および COMM などの列を表示する単純なレポートを作成します。このレポートは、レポート・ウィザードを使用して作成したレポートと同じに見えます。

```
<report name="emp_report" DTDVersion="1.0">
<data>
 <dataSource name="Q_EMP">
  <select>
   select empno, ename, sal, comm from emp
  </select>
 </dataSource>
</data>
<layout>
 <section name="main">
  <groupLeft name="M_emp" template="corp1.tdf">
   <group>
    <field name="F_EMPNO" source="empno"/>
    <field name="F_ENAME" source="ename"/>
    <field name="F_SAL" source="sal"/>
    <field name="F_COMM" source="comm"/>
   </group>
  </groupLeft>
 </section>
</layout>
</report>
```

XML レポート定義の実行

XML レポート定義を作成すると、次の方法で使用できます。

- 「XML カスタマイズの適用」

  CUSTOMIZE コマンドライン引数または SRW.APPLY_DEFINITION 組込みファンクションを指定することによって、XML レポート定義を実行時に .RDF またはその他の .XML ファイルに適用します。

- 「XML レポート定義のみの実行」

  REPORT コマンドライン引数を指定することによって、XML レポート定義のみを（別のレポートなしで）実行します。

XML レポート定義のみの実行

XML レポート定義のみを実行するには、REPORT 引数に XML ファイル指定して要求を送信します。次の方法で可能です。

- コマンドラインから、ジョブ要求を OracleAS Reports Services に送信して emp.xml レポート自体を実行するには、次の構文を使用します。

```
rwcli60 userid=username/password@mydb
  report=e:¥corp¥myreports¥emp.xml
  destype=file desname=emp.pdf desformat=PDF
  server=repserver
```

- OracleAS Reports Services Runtime から、同等のジョブ要求を送信するためのコマンドは、次のとおりです。

```
rwrun60 userid=username/password@mydb
  report=e:¥corp¥myreports¥emp.xml
  destype=file desname=emp.pdf desformat=PDF
```

  XML レポート定義をこのように実行する場合、ファイル拡張子は .XML である必要があります。CUSTOMIZE 引数を使用して、XML カスタマイズ・ファイルをこのレポートに適用できます。

レポート定義を格納するために JSP で使用する XML

Reports 9*i* では、RDF および XML 他に、レポートを保存するための形式として JSP も使用できます。JSP 内では、データ・モデルおよびレイアウトなどのレポート要素が、XML 形式で格納されます。

JSP を使用して何を行うかによって、データ・モデルおよび Web ソースのみを含めたり、レイアウトも含めることができます。たとえば、この場合は、同じファイルから PDF ドキュメントを生成できます。

データソースとしての XML の使用

OracleAS Reports Services では、トランスポータブル・データ・ソース（PDS）の概念を導入しています。PDS を使用すると、独自のデータ・ソースにインタフェースを作成できるため、レポートをこのデータにアクセスさせ、単一のレポート内で、他の PDS からのデータとともに使用できます。

OracleAS Reports Services は、公開されたインタフェースを介して PDS と通信し、それを使用して指定されたソースからデータをフェッチします。Reports の PDS は、ユーザーに対して透過的です。Reports の PDS は、同じデータ・モデル内で並列に使用し、相互にリンクすることができます。

PDS は、Java で作成されています。OracleAS Reports Services にリンクさせるには、構成ファイルを使用します。

トランスポータブル・データ・ソースである XML-PDS

XML-PDS は、OracleAS Reports Services に付属する PDS の 1 つです。XML-PDS を使用すると、ファイルから XML データへまたはインターネットからのライブ・ストリームへアクセスできます。XML データの構造は、DTD または XML Schema 定義（XSD）に従う必要があります。XML Schema を使用するメリットは、次のとおりです。

- XML Schema 定義では、列に様々なデータ型を作成できます。
- DTD を使用する通常の XML 文書では、構造および単純な構文固有の情報しか格納できません。使用できる値は文字型のみです。

OracleAS Reports Services 構成ファイルでの XML の使用例

OracleAS Reports Services の構成ファイルは複雑になったため、XML 形式に変更されました。これによって、変更が簡単になり、可読性も向上しています。

PDS の構成

構成情報を格納する必要があるため、XML-PDS および JDBC-PDS では、XML ファイルを使用してプリファレンス設定を格納しています。

サーバーの構成

サーバーの構成ファイルも XML 形式に変更されました。Reports 6*i* などの以前のバージョンから移行する場合、サーバーは、以前の構成ファイルを読み取り、XML 形式でファイルを作成します。

次に、単純な構成ファイルの例を示します。

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<!DOCTYPE server PUBLIC "-//Oracle Corp.//DTD Reports Server Configuration 9i//EN"
"file:/d:/orawin70/report70/server/jasmine.dtd">
<server>
   <cache class="oracle.reports.cache.RWCache">
      <property name="cacheSize" value="50"/>
      <property name="cacheDir" value="d:¥orawin70¥report70¥server¥cache"/>
   </cache>
   <!--Please do not change the id for reports engine.-->
   <!--The class specifies below is subclass of _EngineImplBase and implements
EngineInterface.-->
   <engine id="rwEng" class="oracle.reports.engine.EngineImpl" initEngine="1"
maxEngine="1" minEngine="0" engLife="50" maxIdle="30" callbackTimeOut="60000">
      <property name="cacheDir" value="d:¥orawin70¥report70¥server¥cache"/>
   </engine>
   <job jobType="report" engineId="rwEng"/>
   <log option="noJob"/>
   <trace traceOpts="trace_all" traceFile="foo.txt" traceMode="trace_replace"/>
   <connection maxConnect="20" idleTimeOut="15">
      <orbClient id="RWClient" publicKeyFile="clientpub.key"/>
      <cluster publicKeyFile="serverpub.key" privateKeyFile="serverpri.key"/>
   </connection>
   <queue maxQueueSize="10000"/>
   <persistFile fileName="d:¥orawin70¥report70¥server¥demo-pc.c7.dat"/>
</server>
```

配布ファイル

OracleAS Reports Services の配布機能は、大幅に拡張されています。拡張に対応し、配布ファイルの管理をさらに容易にするために、XML 形式を使用しています。

新しい配布ファイルは次のとおりです。

```
<?xml version = "1.0" encoding = "UTF-8"?>
<destination>
 <file id = "1" name = "Testfile.html" section = "main" format = "htmlcss">
   <include src = ""/>
 </file>
<mail id = "2" to = "test@myserver.com">
  <attach format = "pdf" srctype = "report">
```

```
    <include src = "test.pdf"/>
  </attach>
</mail>
</destination>
```

Reports 9*i* XML-PDS が XSQL Servlet をサポートする方法

XML-PDS によって、XSQL Servlet ファイルを XML データ・ソースとして指定できます。

XSQL ファイルの URL（ローカルまたはリモート）を、XML- PDS の URL データ・ソースとして指定できます。次に、XML-PDS は、URL の要求を Web サーバーに送信します。XSQL ファイルを処理するには、URL で示された Web サーバーを、XSQL Servlet を使用して構成する必要があります。URL で示されたファイルの固有の拡張子（この場合 .xsql）を識別することによって（Web サーバーで構成可能）、XSQL Servlet が起動されます。

XSQL Servlet は、データベース接続に関する情報を XSQL Pages 構成ファイルに格納しています。指定された XSQL ページを処理し、SQL 問合せを JDBC インタフェースを介してデータベースに送信し、受信した結果セットを XML 形式に変換します。XSQL Servlet は、この結果を XML-PDS に送信します。今後、XML-PDS は、この XML を他の XML データ・ソースとして扱い、レポート内で処理します。

XSQL をデータ・ソースとして使用する場合、XML Schema または DTD を、データ定義として使用する必要があります。

レポートの例

最新の XML ストリームを提供する方法

発展し続ける B2B 環境において、時間は非常に重要な要因となっています。多くのビジネスが、大量の在庫および必要な固定資本の増加を回避するために、ジャストインタイム方式の配送に依存しています。したがって、販売業者は、サプライヤが何を在庫に持っているかを常に把握することが必須です。

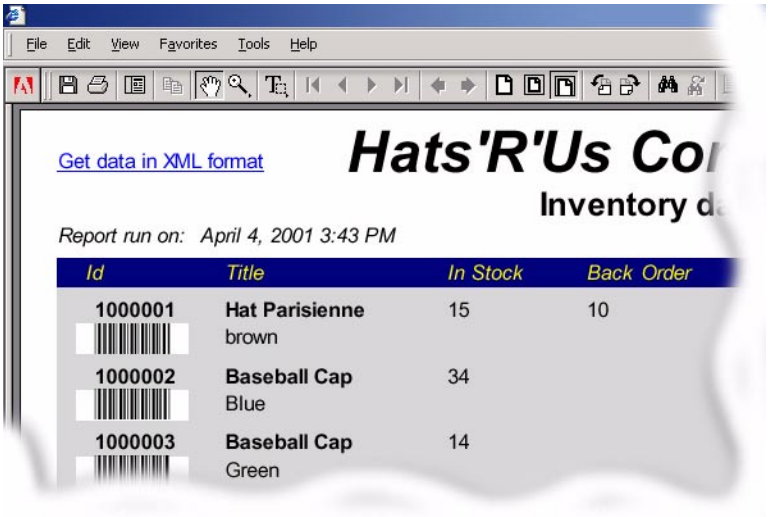
このようなサービスを提供するために、サプライヤは在庫情報を Web ページに提供できます。ほとんどの場合、このような情報は、追加の処理で使用できるように XML などの形式である必要があります。同じレポートから HTML、PDF および XML のレポートを生成できるツールがあるとさらに効果的です。これによって、販売業者は次のいずれかを行うことができます。

- サプライヤの Web ページにアクセスし、情報を検索します。
- URL を使用して、XML 形式で情報を取得し、独自のシステムまたは要求時にデータにアクセスするシステムへその情報をインポートできます。

この場合、OracleAS Reports Services が、理想的なソリューションを提供します。HTML または PDF 出力に最適なレポートを設計し、同時にこれを使用して XML ストリームを生成できます。

図 14-4 に、OracleAS Reports Services によって生成された、HTML 形式のレポート例を示します。

図 14-4 HTML および PDF 出力を生成する在庫データのレポート例



注意： レポート出力の左上の隅にあるリンクをクリックすると、宛先タイプに XML を指定して、レポートがコールされます。これによって、このレポートが XML 形式で出力されます。

図 14-5 に、図 14-4 に示した在庫データを XML ストリームとして表示するレポート例を示します。

図 14-5 XML ストリームとしての在庫データのレポート例

提供された XML データの利用方法

販売業者として、サプライヤが提供する在庫データを使用し、独自の在庫データと結合して仮想の在庫リストを作成できます。

OracleAS Reports Services では、次のモジュールを使用できます。

- 在庫データにアクセスするための、SQL-PDS モジュール
- サプライヤの在庫データにアクセスするための、XML-PDS モジュール

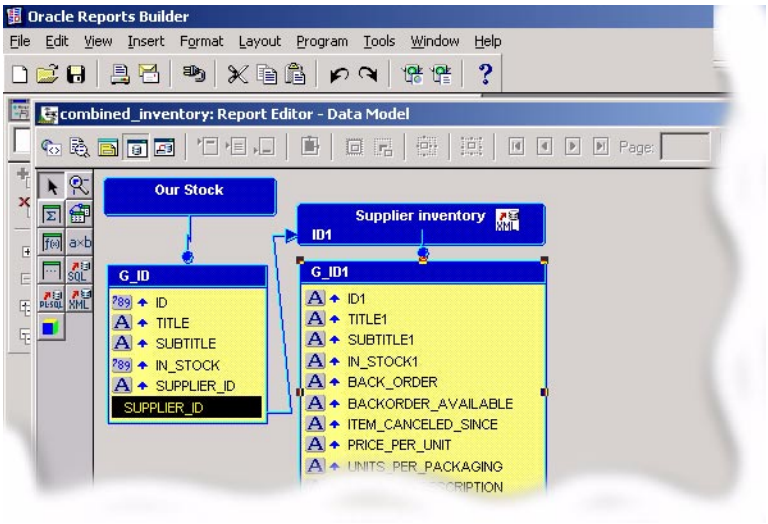
これを行うには、サプライヤの XML ストリームへの URL、および XML ストリームが提供するデータを記述する DTD または XML Schema 定義があると効果的です。たとえば、DTD は次のようになります。

```
<?xml version='1.0' encoding='UTF-8' ?>
<!ELEMENT INVENTORY (INVENTORY_ITEM+)>
<!ELEMENT INVENTORY_ITEM
  ( ID ,
    TITLE ,
    SUBTITLE ,
    IN_STOCK ,
    BACK_ORDER ,
    BACKORDER_AVAILABLE ,
    ITEM_CANCELED_SINCE ,
    PRICE_PER_UNIT ,
    UNITS_PER_PACKAGING ,
    PACKAGE_DESCRIPTION
  )>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT IN_STOCK (#PCDATA)>
<!ELEMENT BACK_ORDER (#PCDATA)>
<!ELEMENT BACKORDER_AVAILABLE (#PCDATA)>
<!ELEMENT ITEM_CANCELED_SINCE (#PCDATA)>
<!ELEMENT PRICE_PER_UNIT (#PCDATA)>
<!ELEMENT UNITS_PER_PACKAGING (#PCDATA)>
<!ELEMENT PACKAGE_DESCRIPTION (#PCDATA)>
```

PDS は透過的であるため、様々な PDS（この場合、SQL PDS および XML PDS）が、データ・モデルでともにシームレスに動作し、SQL PDS のように（結合したように）処理できます。

図 14-6 に、結合した SQL データおよび XML データの両方を使用した、Oracle Report Builder 上のデータ・モデルを示します。

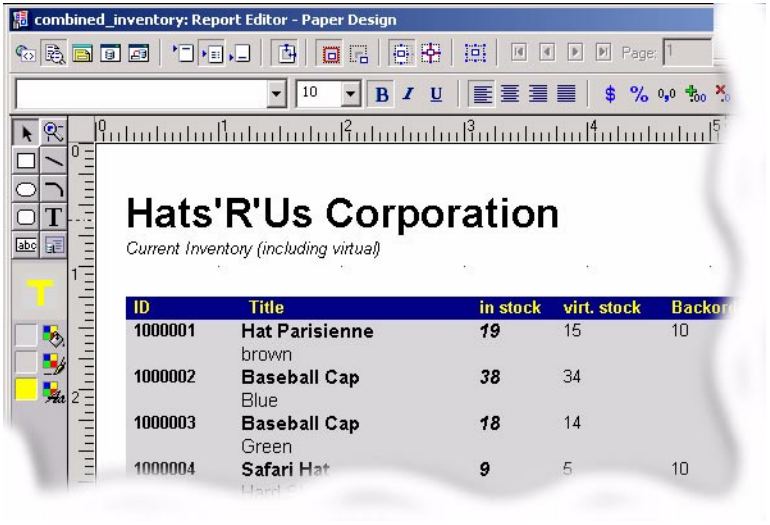
図 14-6 Report Builder: 結合した SQL データ・ソースおよび XML データ・ソースを示すデータ・モデル



両方のデータ・ソースからのデータを使用するレイアウトも作成できます。この場合、「virtual stock」列および「backorder」列は XML ストリームから提供され、この固有の項目に対するデータがサプライヤのインベントリに示されます。データは、レポートが実行されるたびに、サプライヤのサイト上のレポートへの URL を使用してフェッチされます。これによって、サプライヤの在庫データの XML ストリームが生成されます。

図 14-7 に、完成した在庫データのレポートを示します。このレポートには、販売業者独自のインベントリから取り出したデータなどの SQL データ、およびサプライヤから取り出した在庫データなどのリモートの XML ストリーム（仮想データ）が結合されています。

図 14-7 SQL データとリモートの XML ストリームからのデータを結合する在庫データのレポート例



FAQ: レポートおよび XML

データベース・インタフェースを介した年度末レポートからの XML の出力

質問

AU/SG 年度末レポートを処理しています。磁気テープ・ファイルのアーカイブおよび生成も行っています。EOY レポートの情報はありますか？EOY レポート用に XML を使用する方法を説明するサイトまたはドキュメントはありますか？具体的な例はありますか？高速計算式で DBI を使用して、多数の YE 情報を取得しています。XML で DBI を使用またはアクセスできますか？

回答

DESFORMAT パラメータを変更するだけで、Reports 6*i* から XML を出力できます。このパラメータを、HTML または PDF ではなく、XML に設定します。これによって、質問のレポートを処理できます。他の操作は必要ありません。

XML 出力の生成（スタイルシートの適用）以外の質問も、Reports 6*i* によって簡単に処理できます。ドキュメント化されていない PL/SQL 組込みファンクション、`SRW.SET_XML_PROLOG` を使用して、XML のプロローグ行を設定できます。これを行う方法については、Bug#1265291 を参照してください。テストには約 5 分かかります。

Reports 6*i* を使用すると、レイアウトが整った印刷可能な出力（PDF、ポストスクリプトなど）、または Web 上の HTML や HTMLCSS などを生成できると同時に、パラメータを 1 つのみ変更することで、質問のような B2B 環境で使用できる XML を出力できます。

XML レポートの生成は、次のとおり、様々な方法で実行できます。

- OracleAS Reports Services および既存のレポート構造の使用。XML を、レポートの出力として生成します（Oracle Reports 6*i* 機能）。次に、スタイルシートを適用し、指定されたスキーマに従って全体を作成し、レポートをフォーマットします。
- Java および Oracle XSU の使用。この場合、必要な形式ですべての情報を取り出すために、Java ストアド・プロシージャを作成します。Oracle XDK については、http://otn.oracle.com/tech/xml/ を参照してください。
- XSQL Servlet の使用。有効な情報については、http://otn.oracle.com/docs/tech/xml/oracle_xsu/doc_library/relnotes.html を参照してください。
- 現行の磁気テープ処理の使用。磁気テープ処理では、すでに必要なすべてのデータが生成されています。問題は、XML コードが生成されていないことです。この場合の解決策として、磁気テープ処理用の標準の高速計算式による方法を使用した、（XML 形式の）新規ファイルを作成します。

DBI に関する質問については、XDK または XSQL Servlet を使用して、データベース上のすべての表またはビューにアクセスできます。これによって、データベース・アイテム（FF_DATABASE_ITEMS 表）からの情報を使用して XML コードを生成できます。

レポート・テンプレートの変更

質問

あるテンプレートを使用しているレポートに別のテンプレートを適用する必要があります。レポートをコピーして、テンプレートを変更していますが、ウィザードの使用以外にレポートを変更する方法はありますか？ウィザードを使用すると、レポートのレイアウトが変更されます。

回答

まず、テンプレートを適用する必要があります。これは、Report Wizard を使用して新規のテンプレートを適用すると、既存のレイアウトを上書きする新規のレイアウトを作成するためです。他の方法はありません。

ただし、XML を使用して、レイアウトを変更せずに実行時にレポートを変更できます。実行時に XML を使用してレポートを変更する単純な例については、14-8 ページの「XML を使用した実行時のレポートのカスタマイズ」を参照してください。

REP-06106:Error in the XML report definition at line 1

質問

XML を RDF ファイルに適用し、ボイラープレート・テキストを英語から中国語に変更しようとしました。XML は次のとおりです。

```
  <report name="am01.rdf" DTDVersion="1.0">
  <layout>
  <section name="main">
  <field name="f_title" source="title" textColor="red" fontSize="16"
fontStyle="bold"/>
  </section>
  </layout>
  <customize>
  <object name="B_4" type="REP_GRAPHIC_TEXT">
  <properties>
  <property name="textSegment">µçÓºÃû</property>
  </properties>
  </object>
  </customize>
  </report>
```

この XML ファイルを Unicode 形式で保存しました。RWRUN60 を介して、コマンドラインから XML を RDF に適用しようとすると、Report Builder が次のメッセージを表示します。

```
REP-06106:Error in the XML report definition at line 1 in 'c:¥am01.xml'
Start of root element expected instead of TEXT 'null'
```

Oracle Report Server/Developer 6*i* を使用し、Apache Web サーバーで実行しています。

回答

質問の回答については、14-10 ページの「XML を使用したレポートのカスタマイズ例 4: SELECT * 問合せの置換」を参照してください。

15

PDK を使用した Oracle Portal での XML データのビジュアル化

この章の内容は次のとおりです。

Oracle Portal の概要

OracleAS Portal は、Oracle のコンポーネントであり、OracleAS および Oracle データベースにセキュリティ、信頼性およびスケーラビリティを提供します。OracleAS Portal には、アプリケーション開発、データ・ウェアハウス、ビジネス・インテリジェンス、アプリケーションの統合およびモバイル・コンピューティングに簡単に使用できるポータル・ソフトウェアと製品で構成されています。

ポートレットの概要

ポートレットは、Oracle Portal の Web ページに含まれているリージョンです。ポートレットは、他の Web サイトの抜粋を表示し、主要な情報のサマリーを生成する Web コンポーネントとして考えることができます。ユーザーが、頻繁に使用するサイトおよび情報へ簡単にアクセスできるように、1 つのページに複数のポートレットを置くことができます。

ポートレットは、Web ページの他の部分と同様に、Web ブラウザによってレンダリングされます。通常、ポートレットは、標準の HTML を使用してユーザーに情報を表示しますが、これらのインタフェースは、CSS、XSL、Java スクリプト、Java アプレットなどのブラウザ対応テクノロジを使用するように拡張できます。

ポートレットは、企業のイントラネットに格納されたファイルや業務アプリケーションが管理しているデータに関するレポートから、インターネット上のニュースおよび株相場まで、Web アクセス可能なほぼすべての種類の情報へのアクセスに使用できます。この動的な性質によって、ポートレットは、重要な情報の選択、新規の開発に関するユーザーへの通知、および主要なデータの要約に使用されます。

次の 3 種類のポートレットがあります。

- **組込みポートレット :** Oracle Portal では、Web アプリケーション開発、Web パブリッシングおよび外部サイトの統合に、すぐに使用できる機能を搭載した一連の組込みポートレットを提供します。
- **データベース・ポートレット :** ストアド・プロシージャとして実装され、データベースで実行されます。PL/SQL、または PL/SQ の DATABASE PORTLETS でラップした Java ストアド・プロシージャで作成できます。このポートレットは、ポートレットにデータベースとの大量な対話が必要である場合、または開発チームに Oracle を使用した経験がある場合に使用します。データベース・ポートレットの作成方法は、次のとおりです。
  - API がポートレットを適切に表示するために必要なメソッドを公開するパッケージを作成し、データベース・プロバイダを作成します。
  - HTML 表のセル内にレンダリングできるテクノロジ（HTML、Java スクリプト、アプレット、特定のプラグインなど）を生成するポートレットをコーディングします。

  - プロバイダをポータルに登録して、アクセスおよび使用できるようにします。これによって、ポートレットのリポジトリがリフレッシュされます。
  - プロバイダおよびプロバイダが所有するポートレットの情報を格納するポートレット・リポジトリをリフレッシュします。この手順は、プロバイダの登録後にポートレットの情報を変更した場合に実行する必要があります。
- **Web ポートレット :** Portal の外部の Web サーバーで、Web 対応言語を使用して実装します。PDK には、Web ポートレットの構築を容易にする Java クラスが含まれます。

一般的なポートレットの適用例

次に、一般的なポートレットの適用例を示します。

- **インターネット・サイトへのアクセスの集中化 :** Oracle Portal を使用すると、ユーザーが探しているものを簡単に見つけることができるように、多くのサイトへのリンクを一箇所に収集し、これらのアクセス・ポイントを容易に編成できます。ユーザーが独自にパーソナライズしたページから、頻繁に使用するサイトへ簡単にアクセスできるように、これらのリンクの集合をポートレットとして公開できます。
- **情報およびドキュメントの公開 :** Web 上のコンテンツをフォルダに配置し、ポータル・ページにポートレットとして置くことができます。これによって、必要な固有のコンテンツを混在および一致させることができます。
- **動的データ・サービスの統合 :** ポートレットは、外部のデータ・ソースから提供されたコンテンツのレンダリングおよびポータル・ページ上の情報の表示に有効です。たとえば、ポートレットによって、リアルタイムのニュースを XML データ・ソースからポータル内に表示できます。
- **Web アプリケーションへのインタフェースの提供 :** ポートレットを使用して、頻繁に使用するアプリケーション（またはデータ・ストア）に自動的にログインし、その情報のサマリーを取り出すことができます。次に、ポートレットは、ポータル・ページ上の情報を表示します。
- **他の企業システムとの統合 :** 組織内の複数のシステムに、互換性がない場合もあります。ポートレットによって、これらのシステムのインタフェースを、Oracle Portal 環境内に一貫した方法で提示できます。

Oracle PDK

Oracle PDK には、ポートレットを開発することによって OracleAS Portal のフレームワークを拡張するサービスおよびツールが含まれます。これらのサービスを次に示します。

PDK Integration Services（PDKIS）

PDKIS は、OracleAS Portal が提供するサービスの 1 つです。PDKIS によって、表示する前に認証を必要とする URL を含め、コンテンツを URL から直接ポートレットへ入れることができます。PDKIS は次のいずれかのタスクを実行できます。

- HTML コンテンツの直接解析およびポートレットへの配置
- XSL スタイルシートを使用した追加処理のための、XHTML への HTML コンテンツの変換

PDKIS によって、コンテンツをポータルへ入れ、デフォルトの XSL スタイルシートを変更して、表示内容および表示方法を選択できます。

PDK URL サービス

PDK for Java（JPDK）を拡張して、すべての言語で URL ベースのポートレットを作成できます。既存のアプリケーションを使用して、コードを変更することなくポートレットを作成できます。これらのサービスは、JPDK 1.4 以上を使用して、すべてのマシンにインストールできます。

URL サービスの実行要件

URL を実行するために、次のものが必要です。

- OracleAS Portal 3.0.8.9.8 以上。PDK URL サービスのほぼすべての機能は、以前のバージョンで動作しますが、動作が保証されているのは、OracleAS Portal のこのバージョンに対してのみです。認証ポートレットなどの特定の機能は、OracleAS Portal の以前のバージョンでは動作しません。
- JPDK 1.4 以上。

PDK URL サービスの概要

Oracle PDK は、現在、Java および PL/SQL 用のサービスを提供しています。これらのサービスによって、開発者は、Java クラスとサーブレット、JSP および PL/SQL を、Portal API を使用する OracleAS Portal 内のポートレットとして統合できます。PDK および JPDK（PDK Services for Java）は、PL/SQL および Java で簡単にポートレットを開発するためのサンプル、ユーティリティおよびドキュメントを提供していますが、C、C++、Perl、ASP などのその他の言語で作成されたアプリケーション開発者用のソリューションは提供していません。すべての言語でのポートレット開発を容易にするために、Oracle PDK は、PDK URL サービスを提供しています。

PDK URL サービスによって、開発者は、すべての言語で作成されたアプリケーションを使用し、統合ポートレットを簡単に作成できます。URL サービスは、アプリケーションの URL を使用して、コンテンツを解析し、JPDK フレームワークを使用してポートレットを作成します。このプロセスによって、ポートレットの各表示モードを、様々なアプリケーションおよび言語でレンダリングできます。たとえば、ポートレットには、PERL を使用してレンダリングされた表示モード、ASP を使用してレンダリングされた編集モード、HTML を使用してレンダリングされたヘルプ・モード、JSP を使用してレンダリングされた詳細モードなどを含めることができます。

- 現行の JPDK フレームワークでは、`provider.xml` ファイルを介してポートレットを定義およびリストすることができ、必要なプログラミングの量を制限します。
- PDK URL サービスは、JPDK フレームワークを拡張し、その使用しやすさおよび簡潔さのメリットを得ることができます。そのため、URL ポートレットの作成手順は、いくつかの例外を除き、Web ポートレットの作成に必要な手順および構成と同じです。

URL ポートレットの作成

URL ポートレットを作成するには、次の主な手順を実行します。

1. 任意の言語でアプリケーションを作成します。
2. URL を介してアクセスできるように、アプリケーションを構成します。
3. URL を提供することによって、`provider.xml` を介したアプリケーションを定義します。
4. OracleAS Portal を介して、プロバイダを Web Provider として登録します。
5. ポートレットをページに追加します。

Web Provider

Web Provider は、Web アプリケーションとして作成されています。Web サーバーでインストールおよびホストされ、Oracle Portal からはリモートにあります。Web Provider は、一連のポートレットも所有し、管理します。PDK Java サービスは、DefaultProvider というプロバイダ・ランタイムを提供します。これは、プロバイダの機能を実装します。DefaultProvider は、一連のポートレットを所有し、管理します。初期化ファイルを使用して、`provider.xml` という一連のポートレットを管理します。

`provider.xml` というファイルは、プロバイダおよびそのポートレットの情報を格納する静的ファイルです。`provider.xml` の構成を理解することによって、Web ポートレットをリストし、説明を記述する、固有のファイルを作成できます。

URL サービス・アーキテクチャ

URL サービスは、URL からのコンテンツのレンダリングが必要になると、既存の JPDK クラスを使用して、フレームワークを拡張します。PDK URL サービスでは、`provider.xml` ファイル内に、URL ポートレットを定義およびリストすることができます。

また、ポートレットの作成、統合および OracleAS Portal との通信を処理するデフォルトのランタイムを含めることによって、追加のプログラミングを行う必要がなくなります。PDK URL サービスの 3 つの主なコンポーネントは、次のとおりです。

- URL サービス・インタフェース
- URL サービス・ランタイム
- `provider.xml`

URL サービス・インタフェース

URL サービス・インタフェースは、JPDK に含まれる Web Provider インタフェースに追加されています。URL のレンダリングに固有のインタフェースは、oracle.portal.provider.v1.ContentFilter です。これは、URL コンテンツをフィルタリングするための仕様です。

URL サービス・ランタイム

URL サービス・ランタイムは、現行の Web Provider（JPDK）ランタイムからクラスを拡張し、新規の URL レンダリング機能に適合させます。URL サービス・ランタイムは、次のランタイム・クラスで構成されています。

- oracle.portal.provider.v1.http.DefaultURLProvider は、Web Provider ランタイム・クラスである DefaultProvider を拡張します。このクラスは、URL サービスを使用してレンダリングされた、すべてのポートレットのプロバイダを表します。
- oracle.portal.provider.v1.http.PortletNodeHandler は、Web Provider ランタイム・クラスである DefaultPortlet を拡張します。このクラスは、URL レンダリングを参照する、`provider.xml` 内のすべての XML コンテンツを解析します。
- oracle.portal.provider.v1.http.URLSecurityManager は、oracle.portal.provider.v1.PortletSecurityManager によって定義された、Web Provider インタフェースを実装します。このクラスは、ポートレットのアクセスおよびセキュリティを管理します。
- oracle.portal.provider.v1.http.URLPageRenderer は、oracle.portal.v1.PortletRenderer によって定義された、Web Provider インタフェースを実装します。URLPageRenderer によって、URL を使用して、要求の内容をレンダリングできます。
- oracle.portal.provider.v1.http.XMLFilter は、oracle.portal.provider.v1.ContentFilter によって定義された、Web Provider インタフェースを実装します。このフィルタは、URL から受信した HTML コンテンツを XHTML に変換します。
- oracle.portal.provider.v1.http.HtmlFilter は、oracle.portal.provider.v1.ContentFilter によって定義された、Web Provider インタフェースを実装します。このフィルタは、HTML コンテンツを OracleAS に準拠する HTML に変換します。

`DefaultXhtml.xsl` は、XHTML を OracleAS Portal に準拠する XHTML に変換する、デフォルトのスタイルシートです。

provider.xml

`provider.xml` は、情報を階層で格納し、使用可能なポートレットを定義およびリストします。`provider.xml` は、1 つのプロバイダのみと対応付けられています。XML ファイル上のデフォルトのプロバイダは、`oracle.portal.provider.v1.http.DefaultProvider` です。URL サービスを利用するには、`oracle.portal.provider.v1.http.DefaultURLProvider` を指定する必要があります。`DefaultURLProvider` は、`provider.xml` ファイル内の追加されたタグおよび更新されたタグを解析します。`DefaultURLProvider` は、標準の `provider.xml` タグを処理できますが、DefaultProvider は、URL サービス情報を含む `provider.xml` 内のタグを処理できません。

provider.xml タグ

表 15-1 に、provider.xml で追加または変更された provider.xml タグを示します。

表 15-1 provider.xml タグ

provider.xml タグ	説明
プロバイダ・タグ	次に、プロバイダ・タグ内で追加または変更されたタグを示します。
authentication	プロバイダ・タグに追加された、必須のタグ。使用される認証の型に関する情報を指定します。
proxyInfo	必須のタグであり、プロキシ・サーバーの情報を指定します。
authorizatio	必須のタグであり、認証のタイプに関する情報を指定します。
redirectUrl	オプションのタグであり、認証が正常に終了した後のリダイレクションのために、外部のアプリケーションが使用するパラメータ名が含まれます。
ポートレット・タグ	次に、ポートレット・タグ内で追加または変更されたタグを示します。
registrationPortlet	オプションのタグであり、このポートレットがプロバイダの登録ポートレットかどうかを指定します。
portletRenderer	URL ページをアクセプトするために変更されています。各表示モードに対するフィルタリングを行います。

provider.xml の使用

provider.xml は、記述情報を含む宣言ファイルです。ポートレットをリストおよび表示するために使用されます。DefaultProvider は、provider.xml を解析し、ファイルから情報を収集します。provider.xml に示される各ポートレットに、ポートレット・インスタンス（Java オブジェクト）を作成します。また、DefaultProvider は、レンダリング・モード、パーソナリゼーションおよびセキュリティ情報を、provider.xml から取り出します。この情報を、作成する各ポートレット・インスタンスに連結し、他のクラス・ファイルに配布します。

DefaultProvider は、レンダリング・モードに関する情報を PortletRender に、パーソナリゼーションに関する情報を PortletPersonalizationManager に、セキュリティ情報を PortletSecurityManager に配布します。これによって、各クラスの名前または場所が不明な場合、情報を provider.xml から取り出すことができます。

DefaultProvider が provider.xml を解析すると、インスタンスが停止するまでその情報を格納します。provider.xml からの情報を更新、追加または削除する場合、Oracle HTTP Server を停止して再起動し、Portal リポジトリをリフレッシュする必要があります。

XML パーサーは、テキストを含む XML 要素を囲む空白を保持します。そのため、provider.xml を編集する場合、空白および改行の使用に注意する必要があります。次の例について考えてみます。

```
<showEdit> true </showEdit>
<showEditDefault> true </showEditDefault>
<hasHelp> true </hasHelp>
```

これらのタグ内のブール文字列値は、左右に空白があるため認識されません。したがって、FALSE と評価されます。次のようにタグを指定する必要があります。

```
<showEdit>true</showEdit>
<showEditDefault>true</showEditDefault>
<hasHelp>true</hasHelp>
```

provider.xml の構成

provider.xml は、Web Provider が使用する情報を格納します。情報は階層で格納され、最上位の項目は「Provider」になります。ファイルは、DefaultProvider がファイルを解析するために必要なプロセスを簡単にするように構成されます。また、このファイルは、可読性が高くなるように構成されています。

この項では、provider.xml 内の情報について説明します。独自の provider.xml を作成する場合、DefaultProvider がファイルを正常に解析するために必要な階層および構文に従う必要があります。

プロバイダ・タグ

プロバイダ・タグは、provider.xml 内の最初のタグです。このタグは、oracle.portal.provider.v1.Provider を実装するクラスを指定します。この指定によって、Provider Adapter が、対応するプロバイダを指します。プロバイダ・タグには、次の 2 つの属性があります。

- class は、オプションの属性であり、oracle.portal.provider.v1.Provider を実装する Java クラスを指定します。
- oracle.portal.provider.v1.http.DefaultProvider は、クラスが指定されていない場合のデフォルトです。

session は、オプションの属性であり、DefaultProvider 内の initSession メソッドを無効にするために使用されます。指定したセッションが無効である場合、DefaultProvider は、サーブレット・セッションを作成します。この属性のデフォルトは、TRUE です。

provider.xml からのプロバイダ・タグの例を次に示します。この例では、プロバイダ・タグは、DefaultProvider を、oracle.portal.provider.v1.Provider を実装し、サーブレット・セッションを作成するクラスとして宣言します。

```
<provider class="oracle.portal.provider.v1.http.DefaultProvider" session="true">
```

プロバイダ・タグは、単一のタグを管理します。useOldStyleHeaders は、オプションです。JPDK 1.3 を使用する Oracle Portal 3.0.6.6.5 以降を使用し、プロバイダのポートレットがカスタマイズ、ヘルプおよびバージョン情報へのリンクをサポートし、以前の 3.0.6 スタイルのヘッダーおよびフッターを保持する必要がある場合、これを TRUE に設定します。

たとえば、プロバイダに次の行を含めます。

```
tag: <useOldStyleHeaders>true</useOldStyleHeaders>
```

ポートレット・タグ

Web Provider には、ポートレットというタグがあります。プロバイダが管理する各ポートレットに対して、1 つのポートレット・タグがあります。ポートレット・タグは、oracle.portal.provider.v1.Portlet を実装するクラスを宣言します。このタグには、このプロバイダが管理する一連のポートレットおよびその説明をリストします。ポートレット・タグには、次の 3 つの属性があります。

- class は、オプションの属性であり、oracle.portal.provider.v1.Portlet を実装する Java クラスを指定します。
- oracle.portal.provider.v1.http.DefaultPortlet は、クラスが指定されていない場合のデフォルトです。
- resource は、オプションの属性であり、リソース・バンドルを表すクラス・ファイルを指定します。リソース・バンドルは、メタデータ情報のポートレット文字列をローカライズする、コンパイル済 Java ソースです。リソース・バンドルは、ポートレットに関する情報をローカルの環境に格納します。ポートレット名、ポートレット・タイトル、ポートレットの説明などの情報を格納できます。リソース・バンドルに情報を指定した場合、provider.xml で、その情報の値に対するタグを作成する必要はありません。この属性のデフォルトは、NULL です。

version は、オプションの属性であり、ポートレットが実装する PDK のバージョンを指定します。これは、ポートレットが依存するポートレット・インタフェースのバージョンです。現在、この値は 1 に指定する必要があります。バージョンが指定されていない場合、この属性の値はデフォルトで 1 になります。

provider.xml からの 3 つの属性を使用する、ポートレット・タグの例を次に示します。この例では、ポートレット・タグは、DefaultPortlet をポートレットを実装するクラスとして宣言し、HelloWorldBundle というリソース・バンドル、およびバージョン 1 を宣言します。

```
<portlet class="oracle.portal.provider.v1.http.DefaultPortlet"
resource="oracle.portal.sample.devguide.helloworld.resource.HelloWorldBundle"
version="1" >
```

ポートレット・タグが管理するタグは 12 個あります。各タグは、ポートレットの属性を説明します。表 15-2 に、これらのタグを示します。

.

表 15-2 provider.xml の構成 : ポートレット・タグ

ポートレット・タグ	説明
id	必須のタグであり、ポートレットの ID 番号を指定します。この番号は、このポートレット・プロバイダ内で一意である必要があり、開発者が指定します。番号を連続させる必要はありません。DefaultProvider が LONG として番号を受信するため、長さの制限はありません。このタグのデフォルト値はありません。
name	必須のタグであり、ポートレットの名前を指定します。このポートレット名は一意である必要があり、空白または特殊文字を含めることはできません。このタグのデフォルト値はありません。
title	推奨タグです。タイトルは、ポートレットの表示名であり、ポートレットにアクセスするユーザーに対して表示されます。タイトルには、空白および特殊文字を含めることができます。このタグのデフォルト値はありません。
description	推奨タグです。ポートレットをページに追加するユーザーに、説明を表示します。このタグのデフォルト値はありません。
imageURL	オプションであり、ポートレット・イメージが参照する URL を指定します。このタグのデフォルト値はありません。
thumbnailURL	オプションであり、サムネール・イメージが参照する URL を指定します。このタグのデフォルトはありません。
timeout	オプションであり、ポートレットに対して待機しているポータルがタイムアウトするまでの時間（秒）を指定します。この値が指定されていない場合、プロバイダのタイムアウト値が使用されます。
timeoutMsg	オプションであり、ポートレットがタイムアウトする場合に表示するタイムアウト・メッセージを指定します。タイムアウト・メッセージが指定されていない場合、プロバイダのタイムアウト・メッセージを使用します。
showEdit	オプションのフラグであり、ポートレットに「カスタマイズ」リンクが含まれるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。

表 15-2 provider.xml の構成 : ポートレット・タグ（続き）

ポートレット・タグ	説明
showEditPublic	オプションのフラグであり、パブリック・ユーザーがポートレットを編集できるかどうかを指定します。デフォルトでは、ユーザーがログインしていないパブリック・ページは、カスタマイズ・リンクを表示しません。パブリック・ユーザーがポートレットをカスタマイズする必要がある場合、このタグに TRUE を指定します。この値はブール値で、デフォルトは FALSE です。
showEditDefault	オプションのフラグであり、カスタマイズ・ページに EditDefaults リンクが含まれるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。
showPreview	オプションのフラグであり、ポートレットをページに追加する場合、ポートレットにプレビュー・オプションがあるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。
showDetails	オプションのフラグであり、ポートレットをブラウザ・ページ全体に表示できるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。
hasHelp	オプションのフラグであり、ポートレットに「ヘルプ」リンクが含まれるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。
hasAbout	オプションのフラグであり、ポートレットに「バージョン情報」リンクが含まれるかどうかを指定します。この値はブール値で、デフォルトは FALSE です。
defaultLocale	オプションのタグであり、ポートレットがデフォルトで使用する言語を指定します。ロケールを、java.util.Locale クラスにあるロケールのリストに従って、2 文字の言語および 2 文字の国名で指定します。たとえば、en.US のように指定します。
acceptContentTypes	オプションのタグであり、ポートレットが認識する MIME タイプを指定します。このタグは配列であり、item という 1 つのタグが含まれます。

item	acceptContentTypes の下に指定する必須のタグであり、MIME タイプを指定します。ポートレットが認識する MIME タイプごとに、1 つの item を指定します。次に、ポートレットが認識する MIME タイプである HTML および XML をリストする例を示します。
	`<item>text/html</item>`
	`<item>text/xml</item>`

portletRenderer	必須のタグであり、ポートレット・ページをレンダリングするクラスを指定します。portletRenderer タグは、配列であり、1 つの属性と 11 個のタグがあります。class は、オプションの属性であり、oracle.portal.provider.v1.PortletRenderer を実装する Java クラスを指定します。クラスが指定されていない場合、デフォルトは、oracle.portal.provider.v1.http です。
PageRenderer. appPath	必須のタグであり、ポートレットをレンダリングするページのルートへの仮想パスを指定します。
appRoot	必須のタグであり、ポートレットをレンダリングするページのルートへの物理パスを指定します。

表 15-2 provider.xml の構成 : ポートレット・タグ（続き）

ポートレット・タグ	説明
showPage	必須のタグであり、ポートレットをレンダリングするページを指定します。
aboutPage	オプションのタグであり、ポートレットに関する情報を提供するために使用するページを指定します。helpPage はオプションのタグであり、ポートレットのヘルプ・ページを指定します。
editPage	オプションのタグであり、ポートレットのカスタマイズを表示するために使用するページを指定します。
editDefaultsPage	オプションのタグであり、ポートレットのデフォルトの設定をカスタマイズするために、管理者が使用するページを指定します。
previewPage	オプションのタグであり、ポートレットのプレビュー・ページを指定します。
showDetailsPage	オプションのタグであり、ポートレットをブラウザ・ウィンドウ全体に表示するために使用するページを指定します。
pageParameterName	オプションのタグであり、追加のページをレンダリングするためのパラメータ名を指定します。このタグによって、PortletRenderer は、画面遷移をサポートします。

renderContainer	オプションのタグであり、このポートレットのタイトル・バーおよび境界線をレンダリングするかどうかを指定するフラグです。デフォルト値は TRUE です。次の例では、PageRenderer を実装クラスとして宣言し、レンダリング・ページおよびいくつかの追加の表示モードを表示します。このポートレットは、コンテナもレンダリングします。
	`<portletRenderer class="oracle.portal.provider.v1.http.PageRenderer" >`
	`<appPath>/lottery</appPath>`
	`<appRoot>E:¥jpdk¥htdocs¥lottery</appRoot>`
	`<showPage>lotto.jsp</showPage>`
	`<editPage>custom.jsp</editPage>`
	`<aboutPage>about.html</aboutPage>`
	`<helpPage>help.html</helpPage>`
	`<renderContainer>true</renderContainer>`
	`</portletRenderer>`

portletPersonalizationManager	オプションのタグであり、ユーザーのカスタマイズを処理するクラスを指定します。portletPersonalizationManager は配列であり、1 つの属性と 2 つのタグがあります。
class	オプションの属性であり、oracle.portal.provider.v1.PortletPersonalizationManager を実装する Java クラスを指定します。クラスが指定されていない場合、デフォルトは、oracle.portal.provider.v1.http.DefaultPortletPersonalizationManager です。

表 15-2 provider.xml の構成 : ポートレット・タグ（続き）

ポートレット・タグ	説明
dataClass	オプションのタグであり、CustomizationObject を実装する Java クラスを参照します。クラスが指定されていない場合、デフォルトは、oracle.portal.provider.v1.http.BaseCustomization です。
multiLangStringClass	オプションのタグであり、カスタマイズ文字列を格納するために使用する言語を実装する Java クラスを参照します。クラスが指定されていない場合、デフォルトは、Java VM の言語です。次の例では、実装クラスを DefaultPortletPersonalizationManager、カスタマイズ・クラスを BaseCustomization、言語クラスを HashMLString として宣言します。 `<portletPersonalizationManager class="oracle.portal.provider.v1.http.DefaultPortletPersonalizationManager" > <dataClass> oracle.portal.provider.v1.http.BaseCustomization </dataClass> <multiLangStringClass> oracle.portal.provider.v1.HashMLString </multiLangStringClass> </portletPersonalizationManager>`
portletSecurityManager	オプションのタグであり、PortletSecurityManager を実装する Java クラスを指定します。このタグのデフォルト値はありません。

参照 : 次の Web サイトおよびマニュアルを参照してください。

- http://otn.oracle.co.jp/products
- 『Oracle WebDB インストレーション・ガイド リリース 2.2』
- 『Oracle WebDB チュートリアル リリース 2.2』
- 『Oracle WebDB コンポーネント・ユーザーズガイド リリース 2.2』
- 『Oracle WebDB コンポーネント・リファレンス リリース 2.2』
- 『Oracle WebDB サイト・ユーザーズガイド リリース 2.2』
- 『Oracle WebDB サイト・リファレンス リリース 2.2』

OracleAS Portal へのテクノロジの統合

OracleAS Portal には、OracleAS Portal に固有に含まれていない次のテクノロジをシームレスに統合できます。

- Dynamic Services を使用したポートレットの開発。第 18 章「OracleAS Dynamic Services および XML」および http://otn.oracle.co.jp/products を参照してください。
- HTT を使用したポートレットの開発。HTT は、テンプレート使用の動的 Web ページを構築するために、Oracle が開発したユーティリティであり、表示、論理およびデータを完全に分割します。オラクル社のコンサルタントは、HTT を使用してインターネット・ソリューションおよびイントラネット・ソリューションを構築しています。HTT は、簡単に理解および使用できます。このテクノロジによって、動的 Web ページを簡単で迅速に開発できます。また、コードおよびテンプレートの再利用に加えて、複数のテンプレートの集合を作成できます。

16

OE での XML の使用

この章の内容は次のとおりです。

OE と XML

OE トランザクションの多くは、XML 形式で実行するように選択できます。トランザクションの内容は次のとおりです。

- 発注書の受信。発注書の発信は、OE に格納されません。
- サービス注文書の受信および発信
- 発注書の受領
- サービス注文書の受領
- 事前出荷通知
- 請求書

OE では、通信方法（webMethod を使用した HTTP、SMTP（電子メール）など）を指定して、ドキュメントを XML 形式で送信および受信できます。OE は現在、次の 2 つのトランザクション・モデルをサポートしています。

- **格納されたトランザクション :** このトランザクション・モデルでは、トランザクションは、OE にマップおよび格納されます。
- **パススルー・トランザクション :** このトランザクション・モデルでは、OE は、サプライヤとバイヤー間のドキュメントに対するマッピングおよびルーティング・ハブとして機能します。

発信トランザクションまたは受信トランザクション

OE でのトランザクションは、OE と相対して、「発信」または「受信」としてラベル付けされます。

- **発信 :** OE からサプライヤまたはバイヤーに送信されるドキュメントはすべて、「発信」トランザクションと呼ばれます。
- **受信 :** サプライヤのシステムまたはバイヤーのシステムによって生成され、OE へ送信されるドキュメントはすべて、受信トランザクションと呼ばれます。

格納されたトランザクション

OE で作成されるドキュメントおよび受信発注書はすべて、OE に格納されます。これらの XML 文書は、OE データ・モデルでマップおよび格納されます。

- **発信発注書 :** バイヤーが OE でカタログ購入を行うと、OE は、バイヤーの選択した通信方法で発注書をバイヤーに送信します。発信発注書は、バージョン 1.0 以上でサポートされています。
- **オークションからの発信発注書 :** バイヤーは、オークションで落札すると、オークションの項目に対して 1 つ以上の発注書を作成できます（オークションの落札者に対して、各サプライヤの発注書は 1 枚）。OE は、バイヤーの選択した通信方法で発注書をバイヤーに送信します。オークションからの発信発注書は、バージョン 5.2 以上でサポートされています。
- **受信発注書 :** バイヤーのシステムによって作成され、OE へ送信される発注書は、受信発注書と呼ばれます。OE は、サプライヤの選択した通信方法でバイヤーからの受信発注書をサプライヤへ転送します。受信発注書は、バージョン 5.2 以上でサポートされています。
- **発信販売注文書 :** バイヤーがサプライヤのカタログから購入する場合、OE は、サプライヤの選択した通信方法で販売注文書をサプライヤに送信します。発信販売注文書は、バージョン 1.0 以上でサポートされています。
- **発信発注書の受領 :** サプライヤが発注書を受領した後、OE は、発注書の状態を更新し、バイヤーの選択した通信方法で発注書の受領をサプライヤからバイヤーへ転送します。発信発注書の受領は、バージョン 1.0 以上でサポートされています。
- **受信発注書の受領 :** 受信発注書の受領は、サプライヤのシステムによって作成され、OE に送信されます。OE は、サプライヤの選択した通信方法でバイヤーからの受信発注書をサプライヤへ転送します。受信発注書は、バージョン 5.2 以上でサポートされています。

パススルー・トランザクション

OE は、次のトランザクションに対する、ドキュメント・ルーティング・ハブとして機能します。受信した XML 文書はすべて、マップおよび転送されます。

- **受信事前出荷通知 :** サプライヤは、事前出荷通知を作成して、バイヤーに出荷を通知します。受信事前出荷通知は、バージョン 5.2 以上でサポートされています。
- **発信事前出荷通知 :** OE は、サプライヤの受信事前出荷通知をバイヤーに転送します。発信事前出荷通知は、バージョン 5.2 以上でサポートされています。

- **受信請求書 :** サプライヤは、発注書に対する請求書を生成し、OE に送信します。受信請求書は、バージョン 5.2 以上でサポートされています。
- **発信請求書 :** OE は、バイヤーの選択した通信方法でサプライヤの受信請求書をバイヤーに転送します。発信請求書は、バージョン 5.2 以上でサポートされています。

バイヤーおよびサプライヤは、OE を介して、XML 文書を送信および受信するために、独自のシステム・インフラストラクチャを設定する必要があります。

XML 配布形式

OE は、Open Applications Group（OAG）XML を使用してドキュメントを転送します。XML は、Web 上の構造化ドキュメントおよびデータ（スプレッドシート、住所録および金融トランザクションなど）の汎用フォーマットです。OAG は、E-Business およびアプリケーションの統合のために最適な慣習およびプロセス・ベースの XML コンテンツに注目した、独立した標準化団体です。

OAG XML は、XML 形式の特定の機能です。OAG XML 標準は、XML を使用するビジネス・トランザクションを定義および送信するためのオープン標準です。OE は、ビジネス・ソフトウェア・アプリケーションの相互運用性に関する業界のコンセンサス・ベースである XML フレームワークを支持することによって、XML トランザクションを標準化する取組みとして、OAG XML 形式を使用しています。

> **参照 :** OAG の詳細は、http://www.openapplications.org を参照してください。

E-Business ソリューション・アーキテクチャ

OE によって、次の実装の 1 つを使用できます。

- Oracle Message Broker およびアダプタ
- webMethod

webMethod の実装については、16-5 ページの「OE の Availability to Promise（ATP）」を参照してください。

OE の Availability to Promise（ATP）

ATP の機能によって、バイヤーは製品の在庫情報を取得できます。バイヤーからの要求は、webMethod B2B サーバーを介して、XML 文書としてサプライヤに送信されます。サプライヤは、要求を処理し、必要な情報を XML 文書としてバイヤーに戻します。

ATP は、次の 4 層のクライアント / サーバー・アーキテクチャ上で、Java と webMethod の組合せによって実装されます。

ブラウザ（クライアント）<-> Exchange サーバー <-> webMethod B2B サーバー <-> サプライヤ・サーバー

Java クラスは次のとおりです。

- `ATPDataService.java`
- `ATPItem.java`
- `ATPRecord.java`
- `ATPService.java`
- `ATPSupplierInfo.java`
- `ATPThreadService.java`*
- `BuildXML.java`

* ATPThreadService.java は、webMethod によって生成されます。

webMethod サービス

ATP Package には、次のサービスが含まれます。

```
ATPRequest Interface
    ---> ATPThreadService (Java Service)
    ---> httpPost (Flow Service)
    ---> httpTimer (Flow Service)
```

OE からサプライヤへ送信される XML

OE からサプライヤへ送信される XML 文書は、次のとおりです。

```
Control Section:
     Sender: Supplier information is available in this section.
     Logical Id: is the supplier Id.
     Auth Id: is the supplier name.
Data Section:
     User Area:
     Name1: Exchange Id.
     Name2: Exchange Name.
```

```
<?xml version = '1.0' standalone = 'no'?>
<GET_PRODAVAIL_002>
   <CNTROLAREA>
      <BSR>
         <VERB>
            <![CDATA[GET]]>
         </VERB>
         <NOUN>
            <![CDATA[PRODAVAIL]]>
         </NOUN>
         <REVISION>
            <![CDATA[002]]>
         </REVISION>
      </BSR>
      <SENDER>
         <LOGICALID>
            <![CDATA[8821]]>
         </LOGICALID>
         <COMPONENT>
            <![CDATA[SALES]]>
         </COMPONENT>
         <TASK>
            <![CDATA[ATP]]>
         </TASK>
         <REFERENCEID>
            <![CDATA[786957]]>
         </REFERENCEID>
         <CONFIRMATION>
            <![CDATA[1]]>
         </CONFIRMATION>
         <LANGUAGE>
            <![CDATA[EN]]>
         </LANGUAGE>
         <CODEPAGE>
            <![CDATA[CPXML]]>
         </CODEPAGE>
         <AUTHID>
            <![CDATA[CHRIS]]>
         </AUTHID>
      </SENDER>
      <DATETIME qualifer='CREATION'>
         <YEAR>
            <![CDATA[1999]]>
         </YEAR>
         <MONTH>
            <![CDATA[01]]>
```

```
        </MONTH>
        <DAY>
           <![CDATA[01]]>
        </DAY>
        <HOUR>
           <![CDATA[00]]>
        </HOUR>
        <MINUTE>
           <![CDATA[00]]>
        </MINUTE>
        <SECOND>
           <![CDATA[00]]>
        </SECOND>
        <SUBSECOND>
           <![CDATA[0000]]>
        </SUBSECOND>
        <TIMEZONE>
           <![CDATA[-0600]]>
        </TIMEZONE>
     </DATETIME>
</CNTROLAREA>
<DATAAREA>
     <GET_PRODAVAIL>
        <PRODAVAIL returndata='1'>
           <DATETIME qualifer='REQUIRED'>
              <YEAR>
                 <![CDATA[2000]]>
              </YEAR>
              <MONTH>
                 <![CDATA[6]]>
              </MONTH>
              <DAY>
                 <![CDATA[6]]>
              </DAY>
              <HOUR>
                 <![CDATA[15]]>
              </HOUR>
              <MINUTE>
                 <![CDATA[32]]>
              </MINUTE>
              <SECOND>
                 <![CDATA[21]]>
              </SECOND>
              <SUBSECOND>
                 <![CDATA[0000]]>
              </SUBSECOND>
              <TIMEZONE>
```

```
                    <![CDATA[-8]]>
                 </TIMEZONE>
              </DATETIME>
              <QUANTITY qualifer='ORDERED'>
                 <VALUE>
                    <![CDATA[1]]>
                 </VALUE>
                 <NUMOFDEC>
                    <![CDATA[0]]>
                 </NUMOFDEC>
                 <SIGN>
                    <![CDATA[+]]>
                 </SIGN>
                 <UOM>
                    <![CDATA[Each]]>
                 </UOM>
              </QUANTITY>
              <ITEM>
                 <![CDATA[652EGBA002]]>
              </ITEM>
              <SITELEVEL index='1'>
                 <![CDATA[200]]>
              </SITELEVEL>
              <USERAREA>
<NAME index='1'>
  <![CDATA[2]]>
</NAME>
<NAME index='2'>
  <![CDATA[auto-xchange]]>
</NAME>
              </USERAREA>
           </PRODAVAIL>
        </GET_PRODAVAIL>
     </DATAAREA>
</GET_PRODAVAIL_002>
```

XML Messaging Services

Oracle は、XML Messaging Services を使用する他の主要な ERP ベンダーと、OAG に準拠した重要な XML 文書を送受信できます。

Oracle XML Messaging Services は、Oracle E-Business Suite と取引パートナ間での、整形式で妥当な XML メッセージの作成および使用を可能にするツールです。XML Messaging Services は、B2B および Application to Application（A2A）による統合の必要性によって決定された、企業の統合要件をサポートするアプリケーションの相互運用性および統合を可能にします。XML Messaging Services は、OAG に準拠した XML 文書を取引パートナと送受信するために OE で使用される主要なテクノロジです。

XML Messaging Services の機能は次のとおりです。

- 一貫した単一の XML ベース・アプリケーション統合ツールの提供
- メッセージのマッピングおよび作成用のリポジトリ・ベースの設計時 UI およびメッセージ処理用のランタイム・エンジンの提供
- 企業と取引のパートナ間のビジネス・プロセスを並列に処理するための、イベント駆動メッセージ処理のサポート
- 標準に依存しない、オープンなメッセージ開発アプローチのサポート
- OAGI 仕様に準拠した、事前作成済の XML メッセージの提供
- 指定された DTD、ファイルまたは XML Schema に基づいた、メッセージの妥当性の検証
- ルールベースの例外処理のサポート
- 発信メッセージの配置および受信メッセージの抽出のための、Oracle AQ への統合

XML Message Designer およびランタイム実行エンジン

Message Designer は、Java の UI であり、これによって、XML Messaging Services リポジトリへロードできる XML メッセージ・マップを素早く簡単に作成できます。

新しい XML Schema に準拠した XML の生成

XSQL Servlet を使用して、データベース・スキーマを反映した構造を持つ XML 文書を生成できます。XML Schema を様々なカテゴリのデータ（Time Series ML、News ML、FpML など）に対して定義または適用する場合、これらの XML Schema に準拠する XML を生成できる必要があります。

業界標準の XML Schema を作成するには、次のコンポーネントが有効です。

- XML Messaging Services。OE の一部であり、スタンドアロン・コンポーネントではありません。
- XSLT スタイルシート。構造的な変換には有効ですが、一意の ID および相互参照などの内容にはあまり効果がありません。様々なツールを使用してスタイルシートを作成できますが、それらのツールはエディタを拡張したのみであるため、生成やリポジトリなどの操作は簡単ではありません。
- Mercator や Constellar などの特別な変換ツール。これらの高価なツールには、ドラッグ・アンド・ドロップ機能があり、ウィザード・ベースの開発をサポートします。

17

Oracle XML Gateway の概要

この章の内容は次のとおりです。

XML Gateway の概要

Oracle XML Gateway は、Oracle E-Business Suite リリース 11*i*.4 とともに、Oracle のアプリケーション統合フレームワークの主要なコンポーネントとして登場しました。XML Gateway の一連のサービスによって、Oracle E-Business Suite と簡単に統合し、ビジネス・イベントがトリガーする XML メッセージを作成および使用できます。Oracle XML Gateway は、Oracle AQ と統合し、メッセージをエンキュー / デキューします。これらのメッセージは、任意のメッセージ・トランスポート・エージェントを介して、ビジネス・パートナ間で送受信されます。

Oracle XML Gateway サービス

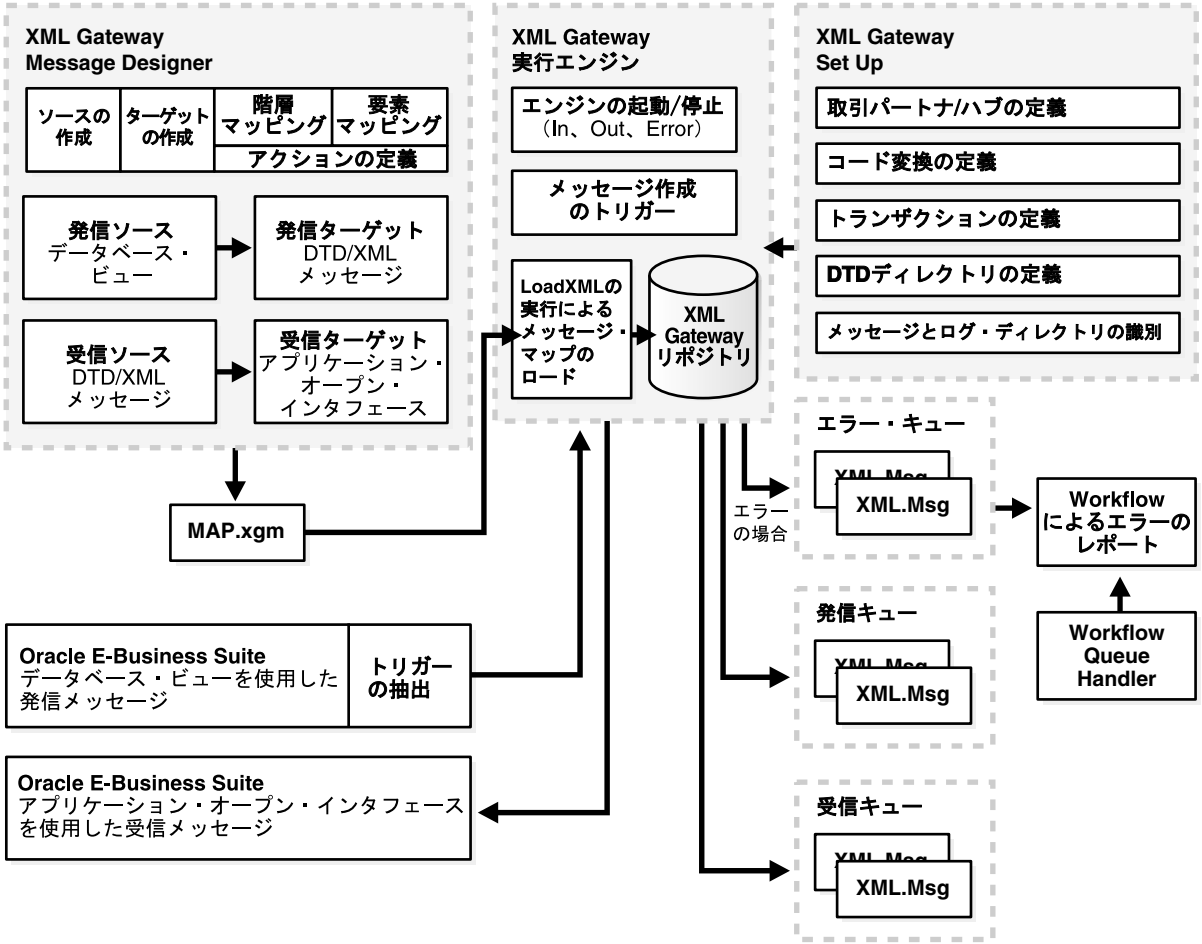
Oracle XML Gateway は、次のサービスを提供します。

- データ・ソースとデータ・ターゲットの定義、階層マップと要素マップの作成およびデータ変換とプロセス制御に対するアクション定義のための、ウィザード形式を使用した、GUI 指向でリポジトリ・ベースの Message Designer。
- ビジネス・イベントに基づいて XML メッセージを作成および使用するために、Oracle E-Business Suite と統合された強力な実行エンジン。メッセージは、XML Gateway リポジトリに格納された（取引パートナと対応付けられた）メッセージ・マップに基づいて、作成または使用されます。
- ハブ、ハブを介してデータを交換するすべての取引パートナ、または固有のビジネス・パートナに対応するための、柔軟な取引パートナの定義。
- メッセージ機能の有効化、メッセージ・マップの識別および通信プロトコルの識別のための、サイト・レベルの取引パートナ・ディレクトリ・サービス。
- 送信者と受信者間のコード変換値およびトランザクション名を定義するための、柔軟なメッセージ設定。
- XML Gateway が、（DTD が使用可能な場合）整形式かつ妥当な XML メッセージを作成または使用することを確認するための、XML パーサーとの統合。
- XML Gateway 実行エンジン、Oracle AQ またはトランスポート・エージェントが検出したエラーをレポートするための、Oracle Workflow を介したアクティブな通知。
- 発信 XML メッセージまたは受信 XML メッセージをエンキュー / デキューするための、Oracle AQ との統合。アクティブなエラー通知をサポートするために、エラー・メッセージのエンキュー / デキューに Oracle AQ が使用されます。
- XML メッセージを送受信するための、Oracle AQ と Oracle Workflow の統合。

Oracle XML Gateway アーキテクチャ

Oracle XML Gateway がサポートするサービスは、Message Designer、Message Set Up および実行エンジンの 3 つの主要コンポーネントに分類されます。図 17-1 に、これら 3 つのコンポーネントの関係を示します。

図 17-1 XML Gateway アーキテクチャ



XML Gateway コンポーネントは、次の Oracle ツール製品およびテクノロジを使用して統合され、完全なソリューションを提供します。

- XML メッセージが（DTD または XML Schema が使用可能な場合）妥当かつ整形式であることを検証する XML パーサー
- 処理エラーをレポートする Oracle Workflow
- XML メッセージをエンキュー / デキューする Oracle AQ
- XML メッセージを配信 / 受信する Oracle Workflow およびトランスポート・エージェント

XML Gateway サービス - Message Designer

XML Gateway Message Designer は、ウィザード形式を使用した、GUI 指向のリポジトリ・ベース・ツール製品です。XML Gateway Message Designer を使用すると、次の操作を実行できます。

- **データ・ソースの作成 :** 受信者に有意なターゲット・メッセージを作成するには、各メッセージのデータをデータ・ソースから取得する必要があります。Message Designer によってサポートされている使用可能なデータ・ソースは、データベース表、データベース・ビュー、DTD またはサンプルの XML メッセージです。発信メッセージの一般的なデータ・ソースは、データベース・ビュー（新規または既存）およびメッセージが必要とする関連ビュー列です。受信メッセージの一般的なデータ・ソースは、任意の XML 標準機関からの DTD またはサンプルの XML メッセージです。

  既存の実装またはレガシー・システムから Oracle E-Business Suite へ移行している場合、サンプルの XML メッセージをデータ・ソースとして選択すると効果的です。識別されている各ソース列に対して、Message Designer を使用して次の操作を実行できます。

  - ソースが DTD または XML Schema である場合のドキュメント・レベルの定義（DTD にはドキュメント・レベルの概念がないため）
  - デフォルト値の設定
  - コード変換の有効化
  - 列が必須であるかどうかの指定
  - 必要に応じた兄弟（同じ階層）または子（次のレベルのディテール）ノードの追加および削除

- **データ・ターゲットの作成 :** データ・ソースと同様に、すべてのメッセージにデータ・ターゲットが必要です。Message Designer がサポートする、使用可能なデータ・ターゲットは、データ・ソースがサポートするものと同一です。発信メッセージの一般的なデータ・ターゲットは、XML 標準機関からの DTD またはサンプルの XML メッセージです。受信メッセージの一般的なデータ・ターゲットは、アプリケーション・オープン・インタフェース表（新規または既存）およびメッセージが必要とする関連の列です。既存の実装またはレガシー・システムから Oracle E-Business Suite へ移行している場合、サンプルの XML メッセージをデータ・ターゲットとして選択すると効果的です。識別されている各ターゲット列に対して、Message Designer を使用して次の操作を実行できます。
  - ターゲットが DTD である場合のドキュメント・レベルの定義（DTD はドキュメント・レベルの概念がないため）
  - デフォルト値の設定
  - 列が必須であるかどうかの指定
  - 必要に応じた兄弟（同じ階層）または子（次のレベルのディテール）ノードの追加および削除

  ソース値を受信者が必要とする値に変換する場合のみ、データ・ソースでコード変換が行われます。
- **階層マッピングの実行 :** データ・ソースおよびデータ・ターゲットを作成したら、Message Designer の階層マッピング機能を使用して、ソース・データ構造をターゲット・データ構造に関連付けます。ソースとターゲット間のドキュメント・レベルが異なる場合、必要に応じて、Message Designer を使用してドキュメント・レベルを拡張または縮小します。
- **要素マッピングの実行 :** ソースおよびターゲットを作成し、ソースとターゲット間の階層を定義したら、Message Designer を使用してソース・データ要素をターゲット・データ要素にマップします。Message Designer は、データ・ソースを左側のペインに、データ・ターゲットを右側のペインに表示します。ソースとターゲット間で単純にドラッグ・アンド・ドロップすることによって、マップ関係が作成されます。マップ関係を識別するために、ターゲット・データ要素名の横にソース・データ要素名が表示されます。
- **アクションの定義 :** 階層または要素のマッピング・プロセスの一部として、Message Designer を使用して、データ変換またはプロセス制御のためのアクションを定義できます。アクションは、次のとおり定義できます。
  - ソースまたはターゲットで定義
  - データ要素、ドキュメントまたはルート・レベルで適用
  - メッセージの作成 / 使用前、作成 / 使用中、作成 / 使用後に適用

  アクションは、事前定義された条件に基づいて適用できます。条件が定義されていない場合、アクションは常に適用されます。

表 17-1 に、XML Gateway がサポートするアクションを示します。

表 17-1 XML Gateway がサポートするアクション

アクションのカテゴリ	アクションの説明
割当て	グローバル変数の作成 別の変数からの値の割当て
数学関数	加算 減算 乗算 除算
文字列関数	部分文字列 連結
データベース関数	次の順序値の割当て WHERE 句の追加 データベース表への挿入
プロシージャ・コール	送信パラメータおよび戻りパラメータを使用したプロシージャの実行
ファンクション・コール	ファンクションの実行およびファンクション戻り値の割当て
XSLT 変換	XSLT 変換を実行するためのプロシージャの実行
OAG 標準変換	■ Oracle の日付を OAG の日付フォーマットに変換 ■ Oracle の操作量を OAG の操作量フォーマットに変換 ■ Oracle の数量を OAG の数量フォーマットに変換 ■ Oracle の量を OAG の量フォーマットに変換 ■ OAG の日付を Oracle の日付フォーマットに変換 ■ OAG の操作量を Oracle の操作量フォーマットに変換 ■ OAG の数量を Oracle の数量フォーマットに変換 ■ OAG の量を Oracle の量フォーマットに変換
送信者へのエラー・コードの返信	送信者へのエラー・コードおよびエラー・メッセージの返信

表 17-1 XML Gateway がサポートするアクション（続き）

アクションのカテゴリ	アクションの説明
グローバル変数値の取得	グローバル変数値の取得 ■ DOCUMENT_ID ■ RETURN_CODE ■ RETURN_MESG ■ SENDER_TP_ID ■ RECEIVER_TP_ID ■ CODE_CONVERSION
その他	プログラムの終了

最も一般的なアクションは、日付値、操作量値、数量値および量値に対する Oracle の表示を OAG フォーマットに（またはその反対に）変換する、OAG 標準変換です。実行エンジンの状態を問い合せることができるため、エラーの重大性に基づいた柔軟なプロセス制御が可能です。重大なエラーが発生した場合、プロセスが強制終了され、Oracle Workflow プロセスを介してエラー・メッセージが送信者に戻されきます。既存のプロシージャまたはファンクションへのコールアウトによって、Oracle E-Business Suite と緊密に統合できます。定義したメッセージ・マップは、XML Gateway リポジトリへロードされ、実行エンジンによる発信 XML メッセージの作成または受信 XML メッセージの使用に使用されます。

XML Gateway サービス - Message Set Up

取引パートナと交換するメッセージを実装するには、XML Gateway サービスの Message Set Up を使用して、取引パートナまたはハブ、コード変換値およびトランザクション名の相互参照を定義します。さらに、ファイル・システム上での DTD、XML メッセージおよびプロセス・ログ・ファイルの格納場所を識別できます。

- **取引パートナ / ハブの定義 :** E-Business は、ビジネス・パートナ（通常は取引パートナ）と直接行う場合と、OE のようなハブを介して行う場合があります。ハブには、多くのバイヤーおよびサプライヤが集まり、E-Commerce を行います。

  Oracle XML Gateway サービスを使用して、ハブまたは実際のビジネス・パートナを取引パートナとして定義できます。ハブを取引パートナとして定義する場合、ハブ上でビジネスを行っているすべてのバイヤーまたはサプライヤを、ハブに対する取引パートナとして識別できます。

取引パートナ / ハブの定義には、次の情報が含まれます。

- 取引パートナ / ハブ名
- 使用可能なメッセージ
- メッセージの作成または使用のためのメッセージ・マップ
- 必要に応じた通信プロトコル（SMTP、HTTP、HTTPS）およびユーザー名 / パスワード
- 取引パートナ固有のコード変換値

- **コード変換の定義 :** コード変換用の Oracle XML Gateway サービスでは、受信者に有意であるように、Oracle コードの変換先の項目を指定できます。送信者のコードの変換先項目も指定できます。コード変換を必要とする、Oracle E-Business Suite の共通コードは、基本単位および通貨コードです。Oracle XML Gateway は、すべてのメッセージのすべてのデータ要素に適用できる、シードされたコード変換値のマスター・リストを提供しています。シードされたリストが十分でない場合、追加のコード変換値をマスター・リストに追加できます。また、特定の取引パートナのみに適用される固有のコード変換値を定義できます。
- **トランザクションの定義 :** Oracle XML Gateway を使用して、Oracle トランザクション名と受信者に有意なトランザクション名間の相互参照を定義できます。Oracle E-Business Suite で配信される事前作成済のメッセージについては、相互参照される名前は、OAG Business Object Document（OAG BOD）によって定義されている動詞 / 名詞（Process PO または Show Delivery など）の組合せです。
- **DTD ディレクトリの識別 :** Oracle XML Gateway を使用して、ファイル・システム上のディレクトリを識別し、メッセージの実装に使用される DTD を格納できます。XML Gateway 実行エンジンおよび XML パーサーは、このディレクトリに格納されている DTD を使用して、すべての発信メッセージおよび受信メッセージを検証し、これらが整形式かつ妥当であることを確認します。
- **XML メッセージおよびプロセス・ログ・ディレクトリの識別 :** Oracle XML Gateway を使用して、XML Gateway 実行エンジンが XML メッセージのコピーおよびそれに関連するプロセス・ログ・ファイルを格納する、ファイル・システム上のディレクトリを識別します。XML メッセージとプロセス・ログ・ファイルは両方とも、トラブルシューティング用にアーカイブまたは使用できます。

XML Gateway サービス - 実行エンジン

XML Gateway 実行エンジンは、Oracle E-Business Suite、XML Gateway Set Up および Oracle AQ とのインタフェースであり、次の機能を実行します。

- 受信キュー、発信キューおよびエラー・キューに対応付けられたエンジンの起動または停止を実行します。
- メッセージの作成が、Oracle E-Business Suite のビジネス・イベントによってトリガーされない場合、ユーザーは、メッセージの作成を手動でトリガーできます。
- Oracle E-Business Suite との相互作用によって発信メッセージを生成します。
- 取引パートナ / ハブを検証します。取引パートナが定義されていない場合、またはドキュメントが取引パートナに対して定義されていない場合、メッセージは生成されません。
- メッセージ・マップをリポジトリから取得します。取引パートナと対応付けられているメッセージ・マップが、XML Gateway リポジトリで使用できない場合、メッセージは作成されません。
- アプリケーション・データを収集します。取引パートナが有効で、メッセージ・マップがリポジトリに存在する場合、XML Gateway 実行エンジンは、メッセージ・マップで識別されるデータベース・ビューおよび列を使用して、Oracle E-Business Suite からアプリケーション・データを収集します。
- コード変換を適用します。コード変換に使用可能なソース列に、コード変換を適用します。
- ドキュメントまたは要素レベルで定義されている場合、アクションを適用します。
- メッセージ・マップおよびアプリケーション・データを使用して、XML メッセージを作成します。
- XML パーサーを使用して、新しく作成されたメッセージが整形式かつ妥当であることを検証します。（DTD ディレクトリに格納された DTD に基づいて）整形式ではない、または妥当でないメッセージは、発信キューでエンキューされません。
- 整形式かつ妥当なメッセージを発信キューにエンキューします。このメッセージは、トランスポート・エージェントによって取り出され、取引パートナに配信されます。
- Oracle E-Business Suite との相互作用によって、受信メッセージを使用します。
- 受信キューからメッセージをデキューします。
- 追加の処理を行う前に、XML パーサーを使用して受信メッセージを検証し、それが整形式かつ妥当であるかどうかを（DTD ディレクトリに格納された DTD に基づいて）確認します。

- 取引パートナ / ハブを検証します。受信メッセージが整形式かつ妥当である場合、実行エンジンは、取引パートナおよびドキュメントが定義されていることを検証します。取引パートナが定義されていない場合またはドキュメントが取引パートナに対して定義されていない場合、メッセージは処理できません。
- メッセージ・マップをリポジトリから取得します。取引パートナと対応付けられているメッセージ・マップが XML Gateway リポジトリで使用できない場合、メッセージは処理できません。
- コード変換に対して有効なソース列に、コード変換を適用します。
- アプリケーション・オープン・インタフェース表へのデータの挿入などのアクションが（ドキュメントまたは要素レベルで）定義されている場合、そのアクションを適用します。次に、オープン・インタフェース API を実行して、ベース・アプリケーション表を移入します。
- 処理エラーを検出し、レポートします。エラーは、Oracle XML Gateway 実行エンジン、Oracle AQ またはトランスポート・エージェントによって検出される場合があります。エラーに関する情報は、エラー・キューにエンキューされます。Oracle Workflow を介して通知が送信され、取引パートナに対してはデータ・エラー、XML Gateway システム管理者に対してはシステム / プロセス・エラーが通知されます。

システム / プロセス・エラーの場合、レポートされたエラーのトラブルシューティングに使用できるように、XML メッセージのコピーが XML メッセージ・ディレクトリに置かれます。取引パートナに関するデータ・エラーの場合、取引パートナは XML メッセージのコピーを参照できます。XML Gateway リスナーは、メッセージに対してアクティブにポーリングを行い、トランザクション・キューに何かが届いたことを検出すると、処理を開始します。XML Gateway 実行エンジンは、ドキュメント情報をトランザクション・キューから取得し、前述のとおり、XML メッセージの作成または使用の処理を開始します。

XML 標準について

DTD は、複数の標準機関（EbXML、Rosettanet、SOAP、iFX など）から公開されています。各機関は、他の DTD より優れていると主張しています。メッセージ・コンテンツの管理に強力な標準もあれば、メッセージ・コンテンツとそれに関連するプロセスの両方の管理に優れている標準もあります。

すべての業界をサポートするソフトウェアのプロバイダとして、オラクル社は、広範囲に実装できるメッセージ機能を持つ OAG の XML 標準を採用しています。OAG は、Oracle の顧客ベースで幅広く採用されている標準でもあります。Oracle E-Business Suite で配信されている、Oracle のすべての事前作成済メッセージは、OAG 標準に基づいています。ただし、DTD または XML Schema 定義が使用可能である場合、事前作成済のメッセージはすべて、XML Gateway Message Designer を使用して、選択した標準に再マッピングできます。

第 V 部

OracleAS Dynamic Services（DS）および Oracle Syndication Server（OSS）

第 V 部に含まれる章は、次のとおりです。

18

OracleAS Dynamic Services および XML

OracleAS Dynamic Services（DS）を使用すると、開発者は、生産性を向上するサービスとして、Web アプリケーションの機能およびコンテンツを利用できます。DS は、Oracle に、Web サービスの配置および機能を追加します。開発者は、Oracle9*i* Dynamic Services を使用して、ユーザーのロール、プロトコルおよび配信デバイスに応じて、Web サービスの構成、カタログへの追加、管理およびパーソナライズを行うことができます。

この章の内容は次のとおりです。

OracleAS Dynamic Services の概要

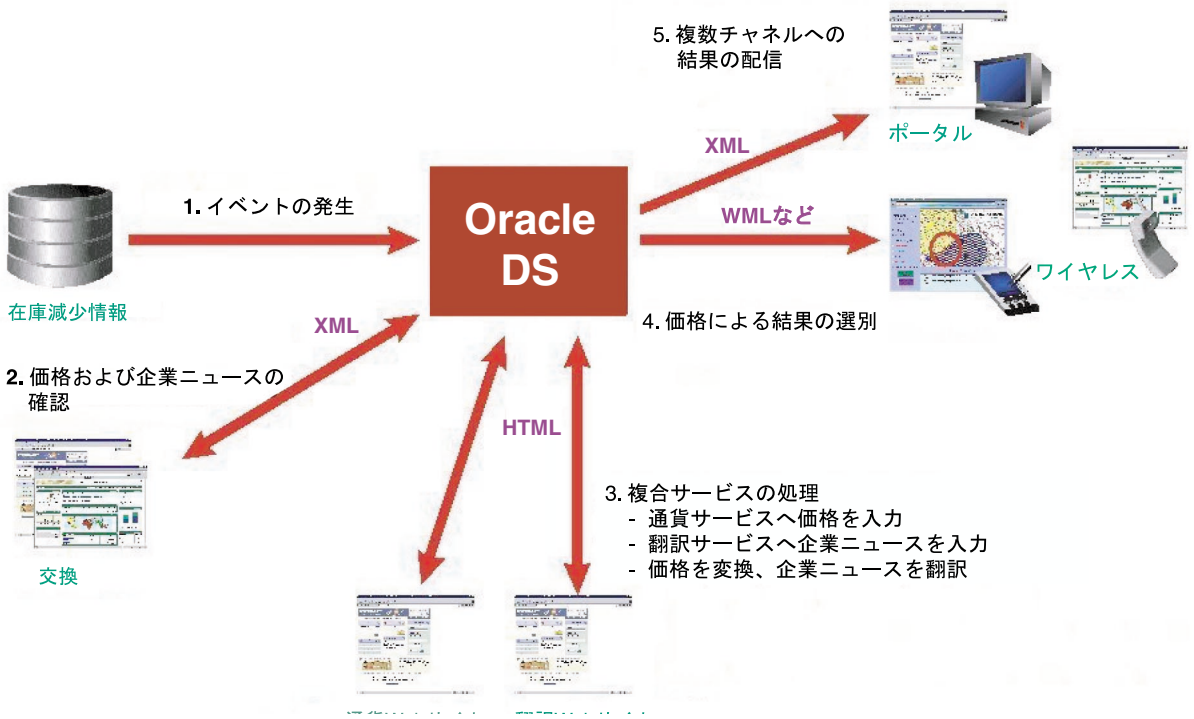
OracleAS Dynamic Services（DS）は、インターネットおよびイントラネット・サービスを組み込み、管理および配置するための、Java ベースのプログラム用フレームワークです。DS は、インターネットを情報ソースとして使用します。DS を使用すると、Web サイト、ローカルのデータベースおよび独自のシステムからアプリケーションへサービスを素早く組み込むことができます。

たとえば、オンラインの財政投資アプリケーションに DS を使用すると、様々なリソース・プロバイダからの株相場や為替相場などのインターネット金融サービスを統合し、現在の株価を外貨で計算することができます。図 18-1 および 18-19 ページの「Dynamic Services のコンシューマ・アプリケーション : 株式投資の例」を参照してください。DS は、サービスの品質を低下することなく、動的なビジネス・モデルを処理します。DS のサービスは、HTTP、JDBC、SOAP などの標準インターネット・プロトコルを介して、情報またはアプリケーションの機能へのアクセスを提供します。また、他の DS のサービスを集約して、固有の実行フローがある複合サービスを形成することもできます。DS サービスには変換および条件ロジックを含めることができ、均一なインタフェースを介してアクセスできます。

図 18-1 に、Dynamic Services（DS）の一般的な使用例を示します。この図は、アプリケーションで DS サービスを使用してデータの取出しおよび管理を行うワークフローを示しています。ユーザーは、必要なセマンティクスおよび出力形式を決定するのみです。

1. データベースが、特定の部品の在庫が減少していることを検出します。これによって、特定の部品の減少を通知するイベントが発生します。
2. DS サービスが起動され、サプライヤにアクセスし、情報にアクセスするサービスを起動します。サプライヤ ABC の Web サイトは、外貨または異なるフォーマットで作成されている場合もあります。在庫がある部品のカタログのみでなく、サプライヤに関するデータも記録されます。
3. 言語および通貨を在庫アプリケーションで必要な言語および通貨に変換する、別のサービスが起動されます。
4. 在庫アプリケーションは、他のアプリケーションにも接続できます。手順 4 および 5 で、結果が複数のチャネルに配信されます。
5. 結果データがターゲット・ユーザーのデバイスに応じてフォーマットされます。DS サービスにはスタイルシートを使用できるため、結果をあらゆる形式にレンダリングできます。データは主に XML に変換されます。

図 18-1 Dynamic Services の一般的な使用例



開発者に対する Dynamic Services（DS）のメリット

DS は、アプリケーション開発者が必要とする設計基準を満たしています。DS を使用すると、開発者には次のようなメリットがあります。

- 次の機能に基づく、サービスへのアクセス、管理、実行および送信のための単一のプログラム用フレームワーク。
  - **アクセス :** 様々なプロトコルを介した、様々なソースへの均一な XML ベースのアクセス。
  - **管理 :** サービス、セッション、キャッシュ・ポリシーおよびイベントの集中管理。
  - **実行 :** サービスの集約およびフェイルオーバーを可能にする高度な実行モジュール。
  - **配信 :** 様々な出力形式、デバイスおよびユーザー・グループへのマルチ・チャネル配信。
- 各企業は、パートナに標準化を強制することなく、社内のサービス・インタフェースを定義できます。パートナが様々なインタフェース、言語、通貨などを使用している場合でも、データ交換のためにパートナに変更を依頼する必要はありません。

詳細情報

OracleAS Dynamic Services の詳細は、次のマニュアルおよび Web サイトを参照してください。

参照 :

- 『Oracle9*i* Directory Service 統合および配置ガイド』
- 『Oracle Dynamic Services ユーザーズおよび管理者ガイド』
- http://otn.oracle.com/products/dynamic_services/index.html

OracleAS Dynamic Services の実行要件

OracleAS Dynamic Services を実行するには、次のコンポーネントが必要です。

- Oracle9*i* Enterprise Edition、OracleAS。
- Java2 準拠（JDK 1.2.2 以上）の JVM がインストールされたプラットフォーム（Oracle8*i* JVM 以上を含む）。これは、DS エンジンが Java で実装されているためです。
- JDK 1.2.2 以上。<JAVA2_HOME> は、JDK 1.2.2 以上のインストール・ディレクトリです。

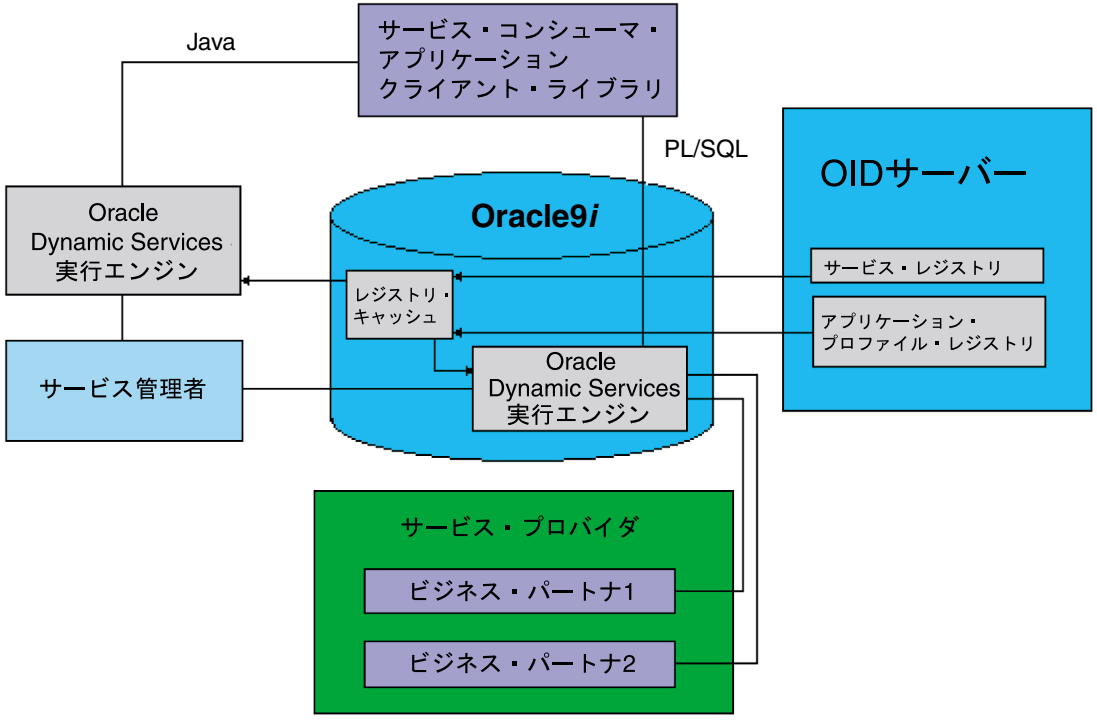
少なくとも、完全（標準）インストールを実行した Oracle があることを確認してください。

Dynamic Services（DS）のアーキテクチャ概要

図 18-2 に、Oracle Dynamic Services（DS）のアーキテクチャを示します。この図には、次のことを示します。

- サービス・プロバイダ（ビジネス・パートナおよびアプリケーション開発者）が提供するサービスを、サービス管理者が DSAdmin ユーティリティ（サービス管理者）を使用してサービス・レジストリに登録するプロセス。
- アプリケーション開発者は、サービス管理者がアプリケーション・プロファイル・レジストリに登録したアプリケーション・プロファイルを使用してアプリケーションを作成できます。レジストリは、Oracle Internet Directory（OID）の LDAP サーバーです。このサーバーの内容は、パフォーマンスの最適化のため、Oracle データベースにもミラー化されています。

図 18-2 Dynamic Services（DS）のアーキテクチャ



Dynamic Services は次の方法で配置できます。

- Oracle JVM 内に配置する場合、PL/SQL インタフェースを使用します（図 18-4 を参照）。
- アプリケーション（Thick クライアント・ライブラリ）をホストするローカルのマシンに配置する場合、Java インタフェースを使用します（図 18-3 を参照）。
- Java サーブレット内の中間層 Java エンジンとして配置する場合、リモートの Java インタフェースを使用します。アプリケーションとは、Dynamic Services の Thin クライアント・ライブラリを介して通信できます（図 18-5 を参照）。

Dynamic Services（DS）の実装の概要

OracleAS Dynamic Services（DS）は、現在、次の配置モードを提供しています。

- **Java 配置**ビュー（18-7 ページの「Dynamic Services の Java 配置」を参照）
- **PL/SQL 配置**ビュー（18-8 ページの「Dynamic Services の PL/SQL 配置」を参照）
- **Java（HTTP/JMS）配置**ビュー（18-9 ページの「Dynamic Services の Java HTTP/JMS 配置」を参照）

DS の主なコンポーネント

次に、各配置モードに必要な DS の主なコンポーネントを示します。

- **Dynamic Services エンジン :** Dynamic Services エンジンは、次に示す 3 つのうちのいずれかのエンジン・タイプとして配置できます。
  - アプリケーション（Thick クライアント・ライブラリ）をホストするローカルのマシン上で実行される Java エンジン（図 18-3 を参照）
  - Java サーブレット内の中間層 Java エンジン（図 18-5 を参照）
  - Oracle JVM 内で実行されるエンジン（図 18-4 を参照）

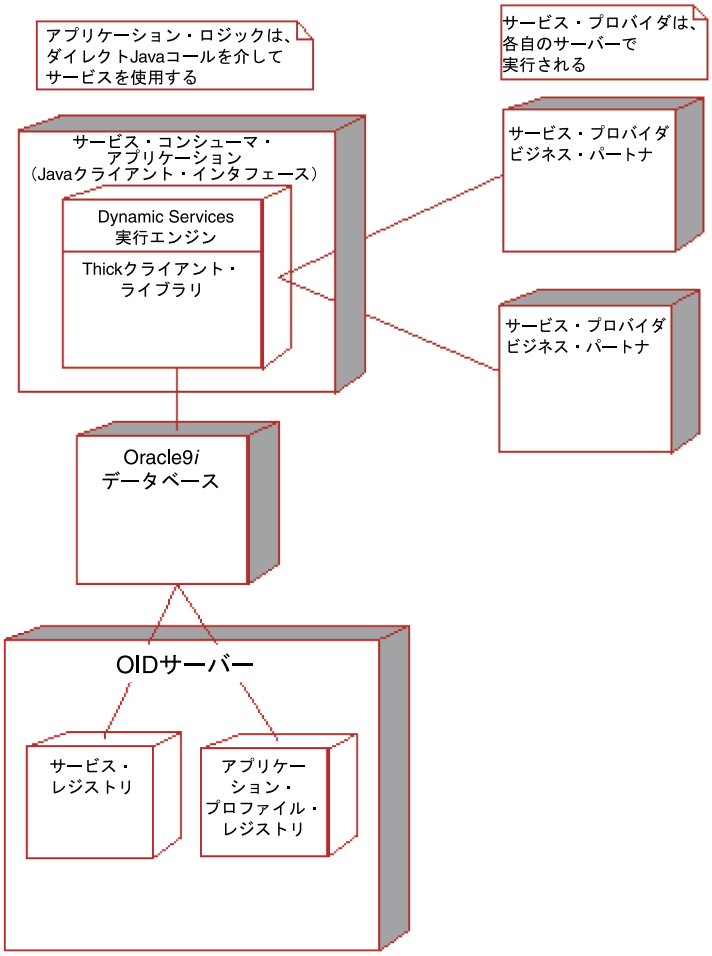
  アプリケーションを再コンパイルまたは再起動することなく、環境を切り替えることができます。これによって、様々なオプションを使用できます。
- **DS サービス・レジストリおよびアプリケーション・プロファイル・レジストリ :** サービス・レジストリおよびアプリケーション・プロファイル・レジストリは、OID サーバーのディレクトリとして使用します。アクセス制御に OID の Access Control List を使用すると、サービス管理者は、サービスが特定のサービス・コンシューマ・アプリケーションに表示されるように選択できます。
- **サービス・コンシューマ・アプリケーションと Dynamic Services エンジン間の通信 :** サービス・コンシューマ・アプリケーションと Dynamic Services エンジン間の通信は、Dynamic Services のクライアント・ライブラリで処理されます。Dynamic Services ドライバを登録すると、サービス・コンシューマ・アプリケーションは、クライアント・ライブラリが Dynamic Services エンジンとの通信に使用する基礎となる通信プロトコルを動的に変更できます。サポートされた通信プロトコルには、HTTP、AQ/JMS、ダイレクト Java アクセスなどがあります。

Dynamic Services の Java 配置

図 18-3 に、Dynamic Services の Java 配置ビューを示します。Oracle はレジストリ・キャッシュとして機能し、レジストリをホストする OID の LDAP サーバーと通信します。サービス・コンシューマ・アプリケーションには、ダイレクト Java コールを介してサービスを使用するアプリケーション・ロジックが含まれます。

サービス・コンシューマ・アプリケーションは、Dynamic Services 実行エンジンを含む DS の Thick クライアント・ライブラリを使用します。サービス・プロバイダは、それぞれのサーバーで実行されます。

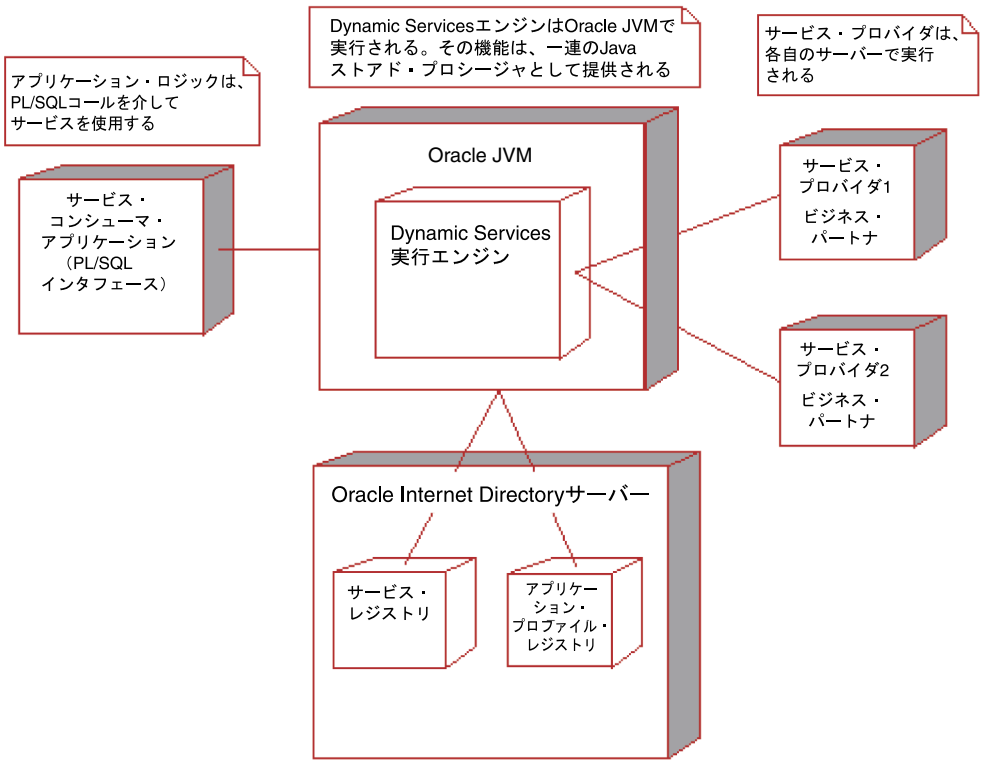
図 18-3 Oracle Dynamic Services: Java 配置



Dynamic Services の PL/SQL 配置

図 18-4 に、Oracle Dynamic Services の PL/SQL 配置を示します。Dynamic Services エンジンは、一連の Java ストアド・プロシージャとして提供されているファンクションを使用して、Oracle JVM で実行されます。Oracle データベースは、レジストリ・キャッシュとして機能します。データベースは、レジストリをホストする Oracle Internet Directory の LDAP サーバーと通信します。サービス・コンシューマ・アプリケーションのロジックは、PL/SQL コールを介してサービスを利用します。サービス・プロバイダは、それぞれのサーバーで実行されます。

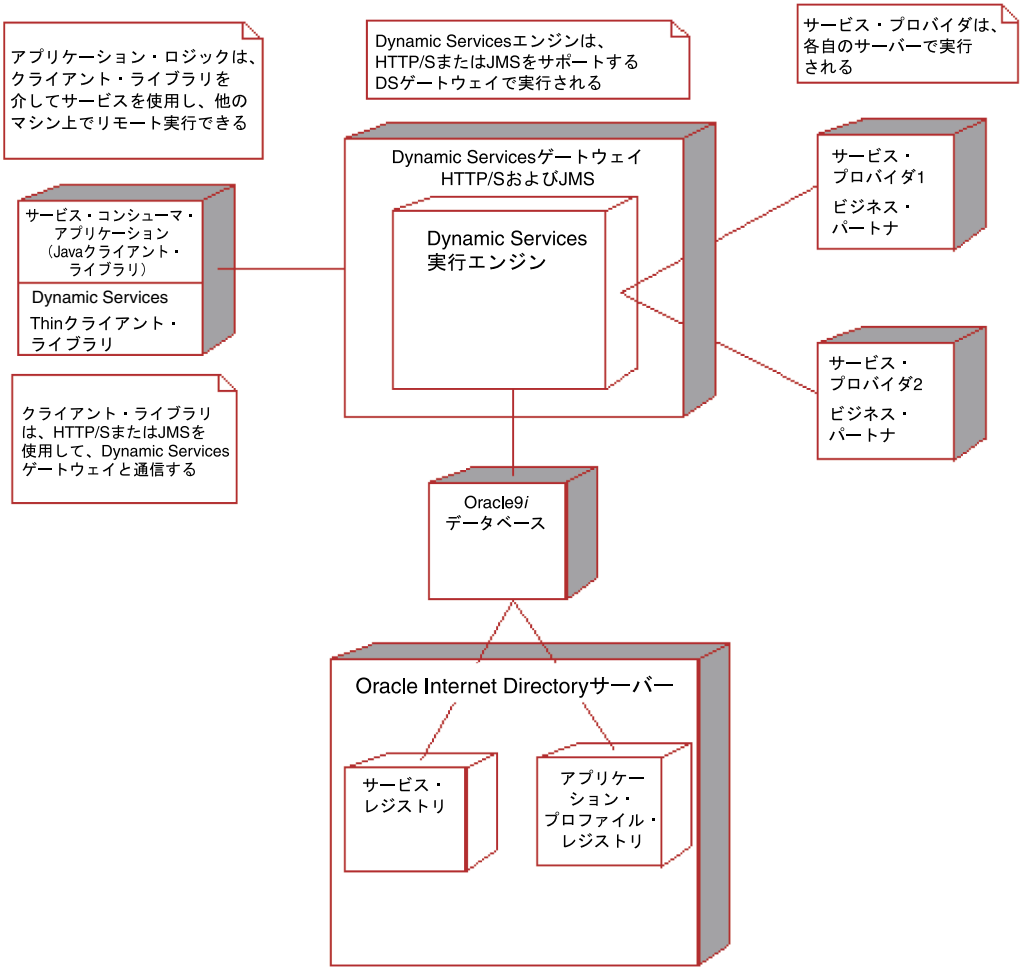
図 18-4 Oracle Dynamic Services: PL/SQL 配置



Dynamic Services の Java HTTP/JMS 配置

図 18-5 に、Oracle DS の Java（HTTP/JMS）配置ビューを示します。Dynamic Services ゲートウェイ（中間層）で実行されている Dynamic Services エンジンは、HTTP、HTTPS および JMS 通信プロトコルをサポートします。Oracle データベースはレジストリ・キャッシュとして機能し、レジストリをホストする Oracle Internet Directory の LDAP サーバーと通信します。サービス・コンシューマ・アプリケーションのロジックは、Dynamic Services の Thin Java クライアント・ライブラリを介してサービスを利用し、他のシステムでサービスをリモートで実行します。サービスの実行要求は Dynamic Services ゲートウェイに転送されます。これによって、サービスが実行され、応答が戻されます。サービス・コンシューマ・アプリケーションとゲートウェイ間の通信は、Dynamic Services の Thin クライアント・ライブラリで処理されます。

図 18-5 Oracle Dynamic Services: Java（HTTP/JMS）配置



たとえば、非同期配置通信（JMS）で、DS の JMS ドライバを使用すると、（Dynamic Services ゲートウェイを使用する）JMS デーモンという形式でサービスへの非同期アクセスが可能になります。このドライバを使用すると、リモート・データベースの AQ/JMS キューに非同期で要求を提出できます。このドライバは、いずれかの場所で実行されている前述の JMS デーモンが存在し、要求が送信されたキューと非同期でリスニングすることを前提としています。

JMS デーモンは、Dynamic Services エンジンとして機能します。JMS デーモンは、要求を処理し、応答を生成し、DS の JMS ドライバが非同期でリスニングする他のキューに応答を送信します。サービス・コンシューマ・アプリケーション側では、応答が戻されるとリスナーに通知されるように登録できます。

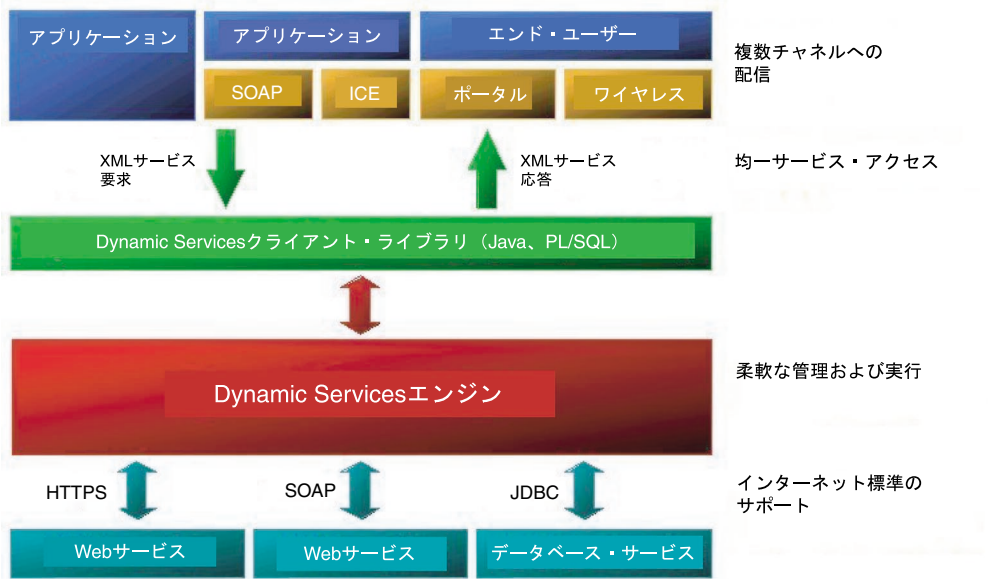
> **参照：** Dynamic Services の Java、PL/SQL および JMS 配置の詳細は、次のマニュアルを参照してください。
>
> - 『Oracle9*i* Directory Service 統合および配置ガイド』
> - 『Oracle Dynamic Services ユーザーズおよび管理者ガイド』

DS のマルチ・チャネル機能

図 18-6 に、DS を配信できる複数のチャネルを示します。次のチャネルが含まれます。

- Oracle Portal
- OracleAS WE
- Oracle Syndication Server（OSS）を介した ICE
- Oracle Dynamic Services の SOAP リスナーを介した SOAP

図 18-6 Dynamic Services の概要



Dynamic Services の機能

Dynamic Services（DS）には次のような機能が含まれます。

サービスの管理

Dynamic Services（DS）には、次のようなサービス管理機能が含まれます。DS のビジネス関係管理は、次のようなビジネス・タスクを処理します。

- サービスのポリシー、優先順位の指定
- パートナ・セキュリティまたは認証モデルの使用
- 使用の追跡およびイベントを介した請求の実施
- 著作権、ロゴおよびサービス記述子内にある他のプロバイダ情報の保持

将来のリリースでは、DS は Oracle Enterprise Manager を介して管理される予定です。

サービス検出

Dynamic Services のサービス検出機能は、実行時のイントラネット検出（LDAP）をサポートします。たとえば、セキュリティ、集中管理および LDAP 検索のために、サービス記述子を Oracle Internet Directory（OID）に格納できます。このサービス記述子に、ミラー化された Oracle インスタンスからアクセスすると、実行時のパフォーマンスが向上します。

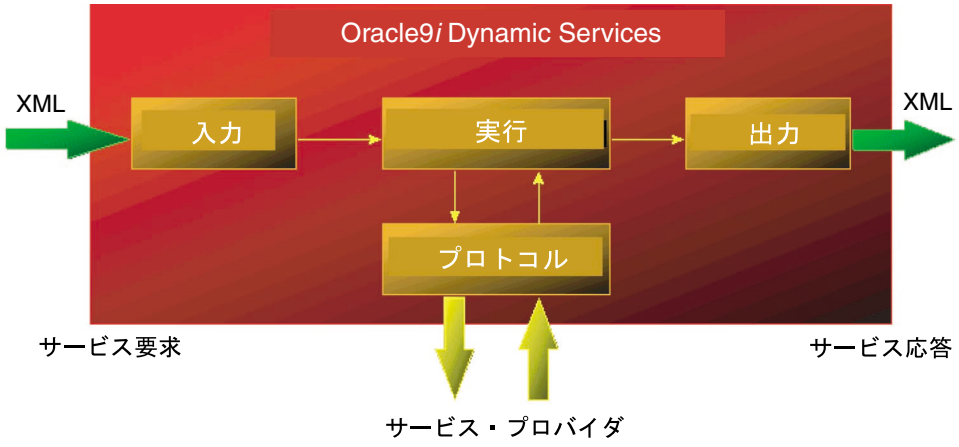
サービスの実行

Dynamic Services には、次のようなサービス実行機能が含まれます。

- サービスの実行（概要については、図 18-7 を参照）
- 次に示す高度な実行モジュール
  - フェイルオーバー
  - 複合
  - 条件付きサービス
- イベント / トリガー

次の項および図では、これらの DS サービス実行機能について説明します。

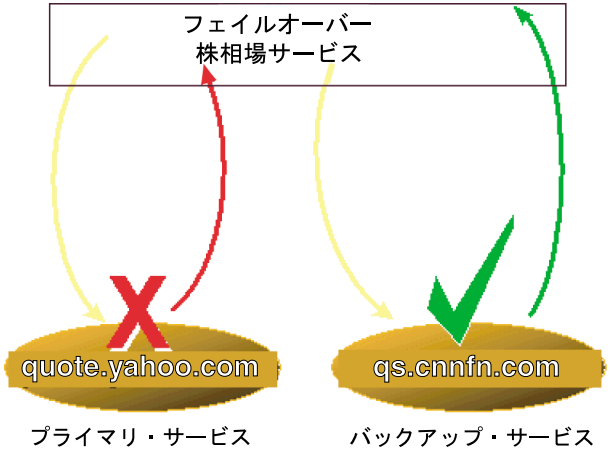
図 18-7 DS: サービスの実行 - 概要



フェイルオーバー・サービス

DS のフェイルオーバー・サービスは、バックアップと同等のサービスの優先順位を示すリストです。図 18-8 に、株相場アプリケーションで使用されるフェイルオーバー・サービスの例を示します。

図 18-8 DS: フェイルオーバー・サービスの例

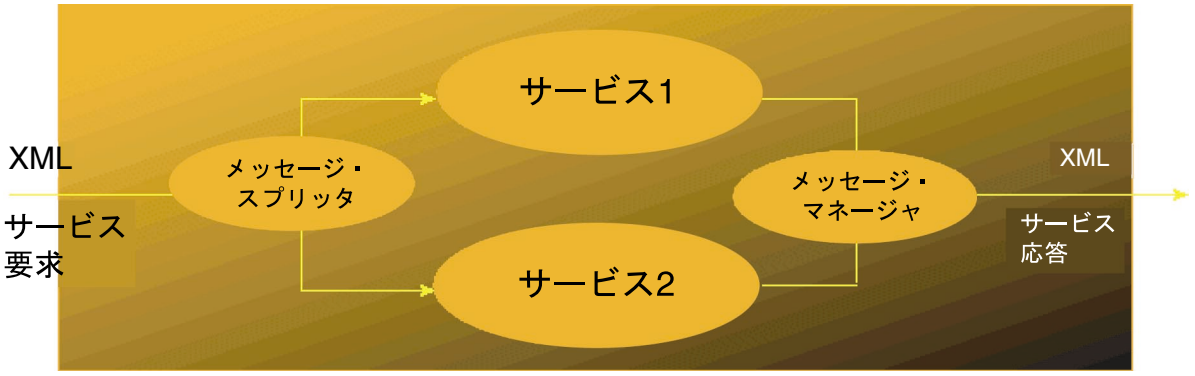


複合サービス

DS を使用すると、サービスを集約し、パラレルで実行する必要があるサービスおよびシリアルで実行する必要があるサービスを指定できます。

図 18-9 に、サービス 1 とサービス 2 という複数のサービスを実行し、これらのサービスで操作（マージまたは分割）を実行して結果を結合し、単一の結果を生成する方法を示します。1 つのサービスの出力を、他方のサービスに入力することもできます。

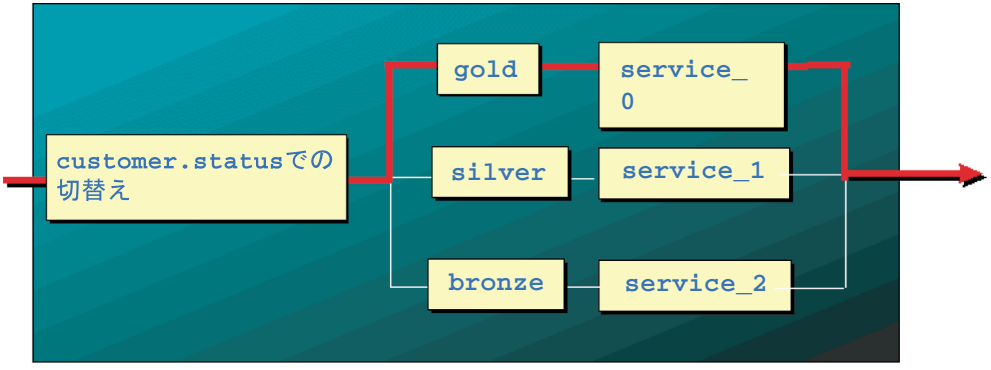
図 18-9 DS: 複合サービス



条件付きのサービス

Dynamic Services は、ビジネス要件に応じてサービスを実行できます。要件は、サービス要件またはユーザー・プロファイルのプロパティに基づいて作成できます。図 18-10 に、DS を使用して上得意客に対して特定のサービスに切り替える方法を示します。

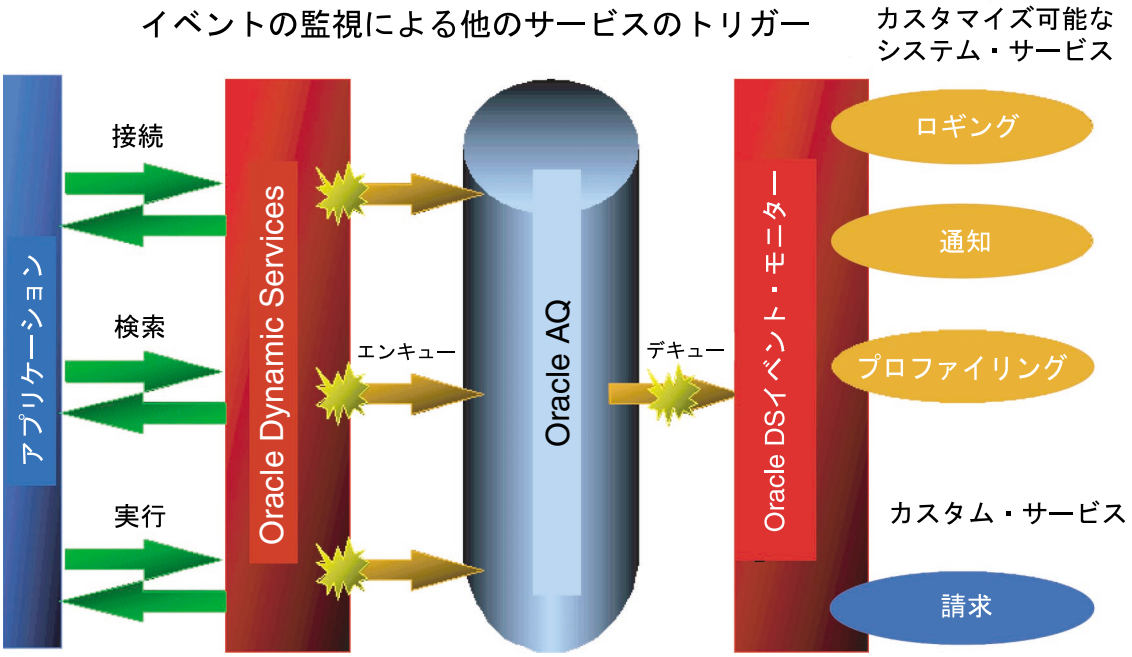
図 18-10 DS: 条件付きのサービス



イベントまたはトリガー

DS サービスの実行中、監視アプリケーションで獲得できる多数のイベントが生成されます。監視アプリケーションは、受信したイベントに応じて、「監視サービス」と呼ばれる他の DS サービスを実行します。図 18-11 に、監視サービスの例、および監視されたイベントが、ロギング、通知、プロファイリングおよび請求サービスなどの様々なサービスを実行する方法を示します。

図 18-11 DS: イベントまたはトリガー・サービス



Dynamic Services と他の Oracle 製品との統合

Dynamic Services（DS）は、次のような他の Oracle 製品と統合できます。

- **JDeveloper:** JDeveloper では、DS を作成、デバッグおよび配置できます。JDeveloper を使用すると、Java、JSP、XSQL および SOAP でのアプリケーション統合が容易になります。
- **OracleAS WE:** DS は、OracleAS WE に統合されています。これは以前、Portal-to-Go と呼ばれていたものです。OracleAS WE のためのアダプタを作成すると、モバイル・アプリケーションで Dynamic Services を利用できます。
- **Oracle Portal:** Oracle Portal JPDK を介して Oracle Portal と統合し、Dynamic Services を使用する Java Web Provider を作成できます。これによって、Dynamic Services をポートレットとして公開できます。

  「Portal Development Kit（PDK）」で PDK（「click here」）を選択し、次に「Integrating Technologies」を選択します。詳細は、『Developing Portlets with Dynamic Services Portlet』を参照してください。

サービス・コンシューマによる Dynamic Services の使用方法

Dynamic Services のクライアント・ライブラリは、サービス・コンシューマ（アプリケーション開発者）に、Dynamic Services エンジンの機能へのアクセスに使用できる Java API を提供します。

> **参照：** Oracle Dynamic Services に付属しているサンプル・サービスの一部に関してサービス要求の作成に使用されるクライアント Java コードの例、およびその実行方法は、次のマニュアルを参照してください。
>
> - 『Oracle9*i* Directory Service 統合および配置ガイド』
> - 『Oracle Dynamic Services ユーザーズおよび管理者ガイド』

詳細は、`<$ORACLE_HOME>/ds/demo/consumer` ディレクトリにあるサンプル・コードおよび `<$ORACLE_HOME>/ds/doc/` の Javadoc API（`apidoc.zip`）を参照してください。

Dynamic Services のサービスの開発

Dynamic Services において、サービスは、特別な付加価値機能を提供するインターネット・コンピューティング・モデル内のコンポーネントです。サービスは、単純なサービス・パッケージにバンドルされ、ローカルのディレクトリとして構造化されます。Dynamic Services を使用する一般的なタスクを次に示します。

- 永続監査サービスまたはイベント監視サービスの有効化
- イベントのロギングおよびイベントの監視の有効化
- イベント・ロガー監視サービスの使用
- ロガー・イベントの問合せ
- キャッシュ・パラメータ値の変更

サービス応答のキャッシュ

Dynamic Services エンジンは、Oracle データベースを使用して、サービス応答をキャッシュします。特定のサービスに対するキャッシュ・ポリシーは、サービス記述子の配置パラメータによって制御されます。サービスを登録する前に、サービス管理者はこれらのパラメータを再確認し、必要に応じて変更できます。キャッシュ・パラメータは、サービス記述子の SERVICE_HEADER 要素、DEPLOYMENT 要素および CACHING 要素で定義されます。

特定のサービスのキャッシュ・パラメータを変更するには、そのサービスの登録を解除し、新しいパラメータ設定で再登録する必要があります。使用可能なキャッシュ・パラメータは次のとおりです。

- MAX_AGE: サービス応答がキャッシュされた状態にある時間（秒）を指定します。指定した時間が経過すると、キャッシュされていた応答が廃棄されます。MAX_AGE の値を 0（ゼロ）以下に指定すると、サービス応答はキャッシュされません。
- SESSION_PRIVATE: ブール値（TRUE または FALSE）を取り、このサービスに関するキャッシュされた応答を、現行セッションの間のみ表示するか、すべての実行が終了するまで表示するかを指定します。『Oracle Dynamic Services ユーザーズおよび管理者ガイド』表 7-1 に、サービス応答の 4 つのキャッシュ方法における動作の概要が示されています。

参照： 次のマニュアルを参照してください。

- 『Oracle9*i* Directory Service 統合および配置ガイド』
- 『Oracle Dynamic Services ユーザーズおよび管理者ガイド』（Dynamic Services サービスの開発の詳細は、このマニュアルの第 3 章、第 6 章および付録 C を参照）

Oracle Syndication Server（OSS）

OSS は、Dynamic Services のアプリケーションです。第 19 章「Oracle Syndication Server（OSS）および XML」を参照してください。

Dynamic Services のコンシューマ・アプリケーション : 株式投資の例

Dynamic Services のソフトウェア・パッケージには、Java を使用して、Dynamic Services エンジンを介して Java を使用し、Yahoo の株式投資サービスを起動するためのサンプル・コードが含まれています。株式投資サービスは、銘柄記号を入力すると、応答として株相場情報が戻されるサービスです。このサービスは、Yahoo が提供しています。

Dynamic Services ソフトウェアには、次のファイルが含まれます。

- `SampleStock.java`: Java で Dynamic Services を起動するための Java コードの例
- `yapfl_req.xml`: サービス要求ファイルのサンプル
- readme.txt

SampleStock.java のコンパイル

`SampleStock.java` をコンパイルする前に、CLASSPATH に次のライブラリを含めます。なお、C:¥Oracle¥Ora81 は、ご使用の $ORACLE_HOME に置き換えてください。また、次に示すライブラリの他に赤字で表示されるライブラリがある場合は、適切に修正してください。

- DS: C:¥Oracle¥Ora81¥ds¥lib¥ds.jar
- Oracle XML Parser 2.0.2.9: C:¥Oracle¥Ora81¥ds¥lib¥xmlparserv2.jar
- Oracle XMLSchema Parser 1.0: C:¥Oracle¥Ora81¥ds¥lib¥xschema.jar
- Oracle AQ および JMS:

  C:¥Oracle¥Ora81¥RDBMS¥jlib¥jmscommon.jar
  C:¥Oracle¥Ora81¥RDBMS¥jlib¥aqapi.jar

- JSSE 1.0:

  C:¥Oracle¥Ora81¥ds¥lib¥jcert.jar
  C:¥Oracle¥Ora81¥ds¥lib¥jsse.jar
  C:¥Oracle¥Ora81¥ds¥lib¥jnet.jar

- LDAP JNDI:

  C:¥Oracle¥Ora81¥ds¥lib¥providerutil.jar
  C:¥Oracle¥Ora81¥ds¥lib¥ldap.jar
  C:¥Oracle¥Ora81¥ds¥lib¥jndi.jar

- ODS - XSQL 1.0.3 および依存ライブラリ :

  C:¥Oracle¥Ora81¥ds¥lib¥sax2.jar
  C:¥Oracle¥Ora81¥ds¥lib¥oraclexsql.jar
  C:¥Oracle¥Ora81¥ds¥lib¥xsu12.jar

コマンドラインで次のとおり入力し、SampleStock をコンパイルします。

```
javac SampleStock.java
```

サンプルを実行およびテストするには、コマンドラインに適切な引数を指定します。

```
java SampleStock <HOST_URL> <SID> <SymbolList>
```

各引数の意味は、次のとおりです。

表 18-1 SampleStock のコマンドラインの引数

引数	説明
HOST_URL	egroup-dev3.us.oracle.com など
SID	db816 など
SymbolList	'ORCL INTC MSFT' など

SampleStock の出力は、次のような要求された銘柄に関するデータの表である必要があります。

```
|-----+--------------------|
|     | Time  | 12:19PM    |
|     |-------+------------|
|     | Price | 30 3/16    |
|ORCL |-------+------------|
|     | Change| +0.42%     |
|     |-------+------------|
|     | Volume| 34,272,000 |
|-----+--------------------|
|     | Time  | 12:19PM    |
|     |-------+------------|
|     | Price | 35 15/64   |
|INTC |-------+------------|
|     | Change| -2.80%     |
|     |-------+------------|
|     | Volume| 22,499,200 |
|-----+--------------------|
|     | Time  | 12:19PM    |
|     |-------+------------|
|     | Price | 63         |
|MSFT |-------+------------|
|     | Change| +0.10%     |
|     |-------+------------|
|     | Volume| 24,091,600 |
|-----+--------------------|
```

Dynamic Services の例 1: SampleStock（Java）

SampleStock は、コマンドライン・パラメータを取り、このパラメータをクラス・データ・メンバーに保持します。次に、getQuotes() をコールして、株相場データを取り出します。

SampleStock は、次の操作によって、Yahoo 証券取引 DS サービスの実行プロセスを抽出します。

- 株式相場のリストを入力し、すべての関連情報を Java.util.Hashtable の形式で戻します。
- 「ORCL INTC MSFT」のように空白で区切られたいくつかの銘柄記号で構成されたパラメータ、symbolList A 文字列を入力します。

- 戻されたハッシュ表は、記号名で索引付けされています。ハッシュ表の各エントリは、それぞれが情報ラベル「時刻」、「価格」、「変更」または「ボリューム」で索引付けされたハッシュ表でもあります。
- SampleStock は、printQuotes() をコールして、前述の形式でハッシュ表に結果を表示します。

```
import java.io.*;
import java.util.*;

// imports for handling XML docs
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

// Dynamic Services imports
import oracle.ds.*;
import oracle.ds.comm.*;
import oracle.ds.registry.*;
import oracle.ds.comm.message.*;
import oracle.ds.driver.*;
import oracle.ds.utils.*;

/**
* SampleStock code. It opens a Dynamic Service
* connection, looks up a stock quote service,
* executes it and stores the results in a hash table.
*/
public class SampleStock implements CommunicationMessageConstants
{
 // Usage message
 private static final String USAGE =
 (
  "Usage: java SampleStock <HOST_URL> <SID> <Symbol list>\n "+
  "where, \n"+
  "\tHOST_URL: i.e. egroup-dev3.us.oracle.com\n"+
  "\tSID: i,e. db816\n" +
  "\tSymbol list: i.e. 'ORCL MSFT SEBL'\n"
 );

 // The following are actually the parameters in the
 // command line and some are hard coded.
 private static String ms_szJdbcURL;
 private static String ms_szUsername;
 private static String ms_szPassword;
 private static String ms_szServiceID;
 private static String ms_szSymbolList;
```

```
// A utility to resolve the NamespacePrefix
// In our case, we hard coded the namespace of the prortfolio service
static private NSResolver ms_nsResolver = new NSResolver()
{
  public String resolveNamespacePrefix(String szPrefix)
    { return "http://www.portfolio.org/Portfolio/Response"; }
};

/**
 * The main function
 */
public static void main(String[] argv)
{
  // Strict: Take only 3 arguments
  if (argv.length != 3) { System.err.println(USAGE); return; }

  // Extract the command line arguments
  int  iArgCounter = 0;
  ms_szJdbcURL = "jdbc:oracle:thin:@" + argv[iArgCounter++]
                          + ":1521:" + argv[iArgCounter++];
  ms_szUsername = "dssys";
  ms_szPassword = "dssys";

  // Hardcoded service ID for "Yahoo StockQuote service"
  ms_szServiceID = "urn:com.yahoo:finance.portfolio03";
  ms_szSymbolList = argv[iArgCounter++];

  // Get quotes
  Hashtable ht = getQuotes(ms_szSymbolList);

  // Print quotes in a form
  printQuotes(ht);
} // end of main

/**
 * This function abstracts the process of executing the YahooPortofolio DS
 * service by taking in an input of a list of stock tickers and returning
 * all the relevant information in a java.util.Hashtable object.
 *
 * @param symbolList A white space delimited string containing the stock
 *                   symbols ( e.g. "ORCL INTC MSFT" ).
 * @return A Hashtable indexed by the stock symbol, and each entry of the
 *         Hashtable is also a Hashtable which is indexed by the information
 *         label, i.e. "Time", "Price", "Change", "Volume".
 */
```

```
 public static Hashtable getQuotes(String symbolList)
 {
   Hashtable ht = new Hashtable();

   // If we are to use a session, a header field has to be set
   DSConnection dsconn = null;

   try
   {
     // First open the connection with the Direct Driver
     DSDriverManager.registerDriver("oracle.ds.driver.DSDirectDriver");
     dsconn = DSDriverManager.getConnection(ms_szJdbcURL);

     // Connect using your specified username/password
     dsconn.connect(ms_szUsername, ms_szPassword);
     System.err.println("==> Opened connection for "+ms_szUsername);

     // Lookup a service and obtaing the service request/response schemas
     DService dsServ = dsconn.lookupService(ms_szServiceID);

     // Make an XML request string from the symbol list
     String xmlRequest =
"<?xml version=\"1.0\"?>                                  \n" +
"<!-- Sample request of the Yahoo! portfolio service --> \n" +
"<PortfolioReq                                            \n" +
"  xmlns=\"http://www.portfolio.org/Portfolio/Request\" >\n";

     StringTokenizer st = new StringTokenizer(ms_szSymbolList);
     while (st.hasMoreTokens()) {
xmlRequest = xmlRequest + "<Symbol>" + st.nextToken() + "</Symbol>\n";
     }

     xmlRequest = xmlRequest + "</PortfolioReq>\n";

     // Create a Request by requesting for a default request context
     // from our Dynamic Services Connection
     DSRequest dsReq = dsconn.createDSRequest(ms_szServiceID,
       new StringReader(xmlRequest));

     // Execute synchronously, get the response and print it
     DSResponse dsResp = null;
     dsResp = dsconn.executeSynch(dsReq);

     // Get the result XML
     StringWriter sw = new StringWriter();
     dsResp.writeResponse(sw);
```

```
      // Instantiate a DOM Parser to parse the result
      // to eventually get a Document
      DOMParser xmlp = new DOMParser();
      xmlp.parse( new StringReader(sw.toString()));
      XMLDocument xmldoc = xmlp.getDocument();

      // Get the list of "Quote" nodes
      NodeList nlist = xmldoc.getElementsByTagName("Quote");
      int nsym = nlist.getLength();

      // For each 'Quote' node
      for (int i = 0 ; i < nsym ; i ++)
      {
XMLNode qnode = (XMLNode) nlist.item(i);

// Make an entry from a quote node
Hashtable entry = new Hashtable();

entry.put("Time", qnode.valueOf("./P:Time", ms_nsResolver));
entry.put("Price", qnode.valueOf("./P:Price", ms_nsResolver));
entry.put("Change", qnode.valueOf("./P:Change", ms_nsResolver));
entry.put("Volume", qnode.valueOf("./P:Volume", ms_nsResolver));

// Insert into hash table
ht.put(qnode.valueOf("./P:Symbol", ms_nsResolver), entry);
      } // end of for
    }
    catch (Exception e) { e.printStackTrace(); }

    // Clean up job
    finally
    {
      // If I have a valid connection then do clean-up
      if(dsconn != null)
      {
// Now close the connectin
try { dsconn.close(); }
catch(Exception e) { e.printStackTrace(); }
      }
    }

    // Return the final result
    return ht;
 } // end of getQuotes
```

```
/**
 * This function outputs quotes in a Hashable to a form like
 * |-----+-------------------|
 * |     | Time  | 12:19PM   |
 * |     |-------+-----------|
 * |     | Price | 30 3/16   |
 * |ORCL |-------+-----------|
 * |     | Change| +0.42%    |
 * |     |-------+-----------|
 * |     | Volume| 34,272,000|
 * |-----+-------------------|
 * |     | Time  | 12:19PM   |
 * |     |-------+-----------|
 * |     | Price | 35 15/64  |
 * |INTC |-------+-----------|
 * |     | Change| -2.80%    |
 * |     |-------+-----------|
 * |     | Volume| 22,499,200|
 * |-----+-------------------|
 * |     | Time  | 12:19PM   |
 * |     |-------+-----------|
 * |     | Price | 63        |
 * |MSFT |-------+-----------|
 * |     | Change| +0.10%    |
 * |     |-------+-----------|
 * |     | Volume| 24,091,600|
 * |-----+-------------------|
 *
 * @param ht A Java.util.Hastable that is the result of method getQuotes().
 */
public static void printQuotes(Hashtable ht)
{

  System.out.println("|-----+-------------------------|");

  Enumeration e = ht.keys();

  // For each key in the Hashtable
  while (e.hasMoreElements())
  {
    // Key is a symbol
    String symbol = (String) e.nextElement();

    // Get an entry
    Hashtable entry = (Hashtable) ht.get(symbol);
```

```
      // -- Time
      System.out.println("|     | Time  | " +
        (String) entry.get("Time") + "\t\t|" );
      System.out.println("|     |-------+-----------------|");

      // -- Price
      System.out.println("|     | Price | " +
        (String) entry.get("Price") + "      \t|" );
      System.out.println("|" + symbol + " |-------+-----------------|");

      // -- Change
      System.out.println("|     | Change| " +
        (String) entry.get("Change") + "\t\t|" );
      System.out.println("|     |-------+-----------------|");

      // -- Volume
      System.out.println("|     | Volume| " +
        (String) entry.get("Volume") + "\t|" );
      System.out.println("|-----+-------------------------|");
    } // end of while
  } // end of printQuotes
}
```

SampleStock.java のいくつかの一般的な XML コードを示します。

```
<PortfolioReq xmlns="http://www.portfolio.org/Portfolio/Request">
  <Symbol>ORCL</Symbol>
  <Symbol>INTC</Symbol>
</Portfolio>
```

FAQ: Dynamic Services

キューイングおよび順序付けコマンドの言語を設定する方法

質問

キューイングおよび順序付けコマンド用の簡単な言語の設定を検討しています。XML には明白なメリットがあるため、この用途に既存の DTD を使用する場合について教えてください。難しくないことは明白ですが、すでに標準の DTD が存在している場合、初めからやりなおす必要がありますか？ 必要なコマンドは次のとおりです。

- キューからのフェッチ
- XSLT 変換の実行
- XSQL 処理の実行
- キューへの挿入
- 電子メールの送信
- メッセージの記録
- 構成パラメータの設定

将来的には他のコマンドも必要になる予定です。

回答

Dynamic Services を使用すると、前述の XML ベースのサービスであるすべてのコマンドをモデル化できます。

これらのサービスを、実行フローと同様に定義できます。Dynamic Services エンジンは、フロー（エンジン内で使用可能な SMTP サービスがあるため、電子メールの送信も含む）に従って、完全な応答を戻します。エンジンは、Java API または PL/SQL API を介して直接コールできます。Dynamic Services ゲートウェイまたは同等の機能を使用して、SOAP または他の SOAP と同様の XML-RPC コールを使用して HTTP 上で要求を受け入れることもできます。エンジンの大部分は拡張可能なため、必要に応じてあらゆる変換、プロトコルまたは実行フローをプラグインできます。

その他の FAQ

その他の FAQ は、次の場所にある DS ソフトウェア・パッケージを参照してください。

```
$ORACLE_HOME/ds/doc/dsfaq.txt
```

19

Oracle Syndication Server（OSS）および XML

Oracle Syndication Server（OSS）には、様々なコンテンツ・リソースについて 1 つの単純なカタログを生成し、すべてのインターネット・データを集約する、コンテンツ・シンジケーション・ソリューションがあります。このソリューションは、更新されたコンテンツをどこにでも自動的に送信します。

この章の内容は次のとおりです。

Oracle Syndication Server（OSS）の概要

コンテンツ・シンジケーションは、B2B サプライ・チェーンでのカタログ交換ソリューションを含む多くのアプリケーション・シナリオ、および複数のポータルへのコンテンツ・プロバイダのシンジケーションに適用できます。Oracle Syndication Server（OSS）には、拡張可能でスケーラブルなコンテンツ・シンジケーション・ソリューションがあります。このソリューションは、次のサポートを提供します。

- 様々なコンテンツ・リソースに対して 1 つの単純なカタログを生成します。
- インターネット・データを集約します。
- 更新されたコンテンツをどこにでも自動的に送信します。

このため、OSS を使用すると B2B、B2C および B2E でのコンテンツ・シンジケーションが可能になり、プロバイダ間でのコンテンツ交換、プロバイダからコンシューマへのコンテンツの配信および組織間でのコンテンツの共有が容易になります。

参照： 次の Web サイトおよび章を参照してください。

- http://otn.oracle.com/products
- 第 18 章「OracleAS Dynamic Services および XML」

OSS の機能 : E-Business のコンテンツの集約、交換およびシンジケーション

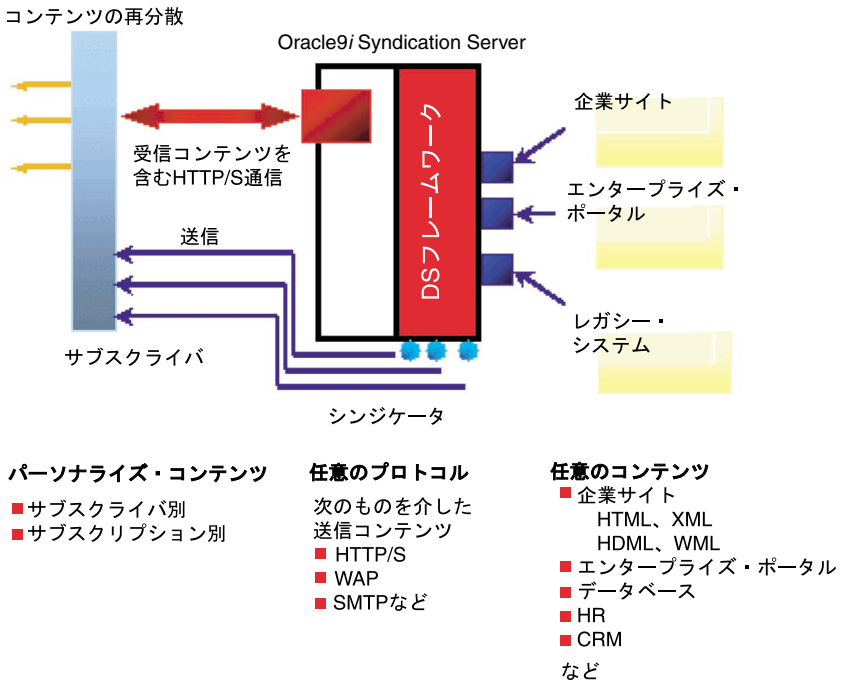
OSS は、E-Businesses のために、コンテンツの集約、交換およびシンジケーションを提供します。OSS によって、場所や時間に関係なくデータを使用できることが保証されます。OSS の主な機能は次のとおりです。

- **あらゆるソースからのコンテンツの集約 :** OSS は、OracleAS Dynamic Services を使用して、既存の Web サイト、データベース、エンタープライズ・アプリケーション、電子メール・リポジトリおよび既存のシステムを含むあらゆるソースからの情報を抽出または適用し、サブスクライバにシンジケートします。これらの情報およびコンテンツのソースは、すべてのサブスクライバに対して、「コンテンツ・カタログ」と呼ばれる 1 つのアクセス・ポイントに統合されます。OracleAS Dynamic Services の詳細は、第 18 章「OracleAS Dynamic Services および XML」を参照してください。
- **サブスクリプションおよび各サブスクライバへのコンテンツの配信のパーソナライズ :** OSS を使用すると、シンジケータで、サブスクライバのプロファイルに基づいて配信するコンテンツをパーソナライズし、サブスクリプション配信ポリシーに基づいて更新されたコンテンツを配信できます。

- **コンテンツの配信の自動化** : OSS は、関連する情報が変更された場合に、サブスクライバにコンテンツを送信します。シンジケータは、サブスクライバへの送信をスケジューリングできます。OSS は、Oracle9*i* Dynamic Services のフレームワークを使用して、複数のチャネルにまたがってコンテンツを送信します。
- **サブスクライバのフォーマットに適合させるためのコンテンツの変換** : OracleAS Dynamic Services のフレームワークによって、OSS は、フォーマットされたあらゆるソースから各サブスクライバに適したマークアップ言語にコンテンツを変換できます。

図 19-1 に、OSS の機能の概要を示します。OSS は、あらゆるプロトコルを介して、シンジケータとサブスクライバ間でコンテンツを転送および再分配するために使用されています。また、コンテンツのパーソナライゼーションも提供しています。OSS によってデータの送受信が容易になることが示されています。OSS には、サブスクライバのプロファイル、コンテンツ・リソースのプロファイル、確立されたサブスクリプションおよびシステムの監視を管理する管理ツールが含まれます。

図 19-1 Oracle Syndication Server（OSS）の機能



コンテンツ・シンジケーション

コンテンツ・シンジケーションには、2 つの役割があります。

- **コンテンツ・サブスクライバ :** 二者のうち、シンジケータから情報およびコンテンツを取得する側。これには、サブスクライバ（人）、コンテンツの供給を必要とする組織のアプリケーション、またはサブスクライブするコンテンツを利用する組織のいずれかに指定できます。
- **コンテンツ・シンジケータ** : 二者のうち、情報およびコンテンツをサブスクライバに送信する側。サード・パーティのコンテンツ・プロバイダからコンテンツを取り出すか、または組織内でコンテンツを提供します。

Oracle Syndication Server（OSS）は、コンテンツ・シンジケータまたはコンテンツ・サブスクライバのいずれかとして、またはその両方として機能するように構成できます。

ICE プロトコル

ICE プロトコルは、コンテンツ・シンジケーションのプロセスにおいて、シンジケーション関係の確立、データ交換および結果の分析を管理および自動化する、XML ベースの仕様です。

業界固有のボキャブラリと組み合せると、ICE は、情報のプロバイダとサブスクライバ間で行うあらゆるタイプの情報のシンジケートに完全なソリューションを提供します。そのため、ICE は、ネットワークで結ばれたパートナや子会社との間で電子資産の制御された交換および管理を行うことを容易にします。ICE ベースのアプリケーションを使用すると、情報およびコンテキストの交換ネットワークを確立することにより、企業は、シンジケートされた公開ネットワーク、Web スーパーストアおよびオンライン販売チャネルを簡単に構成できます。ICE には、次のような機能が実装されています。

- **メッセージの定義 :** ICE のメッセージは、XML ベースです。ICE 1.1 仕様は、XML DTD を使用してこれらのメッセージの書式を定義するため、プロトコルの文法はこのメッセージの定義によって確立されます。
- **コンテンツの転送** : ICE プロトコルは、XML 文書の交換に基づいて設計されています。各プロトコル・メッセージは有効な XML 文書で構成され、プロトコルはシンジケータとサブスクライバ間でこれらの文書を送受信します。仕様によって ICE が特定の通信メカニズムに制限されることはありません。
- **セキュリティ** : ICE を実装すると、トランスポート・レベルで暗号化などのメソッドを使用して、セキュリティを実現できます。また、ICE プロトコルによる項目であるデジタル署名されたコンテンツを、アプリケーションで合意した後に送信することもできます。また、シンジケータおよびサブスクライバが、証明書を使用して相互に認証することもできます。

ICE の操作タイプ

ICE プロトコルは、4 つのタイプの操作を処理します。

- サブスクリプションの確立および管理。ICE では、サブスクリプションを確立することによって、シンジケータとサブスクライバの関係が開始します。ICE では、サブスクライバは、通常、シンジケータから使用可能なサブスクリプション（実際にはサブスクリプションの提供）のカタログを取得することによって開始します。ICE プロトコルで定義されたカタログは、サブスクリプションの提供グループで構成されます。各提供グループには、サブスクライバが選択に使用する最も詳細な単位である提供のセットが含まれます。ICE プロトコルは、各提供に対して、対応付けられた配信ポリシー、使用レポート、表現制約および取引条件の構造をサポートおよび定義します。
- データ配信。サブスクライバは、可能なかぎり、プロトコル・パラメータ・ネゴシエーションを行って配信メソッドおよびスケジュールを相互に合意し、特定のサブスクリプションにサブスクライブします。
- イベント・ログ。主要なメッセージ交換の重点がデータ配信に置かれている場合、関係は安定した状態になります。ICE は、共通のデータ項目のコンテナ・メカニズムとして、パッケージ・コンセプトを使用します。ICE は、順序付けされたパッケージ・モデルを定義して、シンジケータが段階的更新モデルと完全更新モデルの両方をサポートできるようにします。また、送受信データの転送モデルを定義します。例外条件の管理および問題の診断機能は、シンジケーション管理の重要な要素です。ICE は、（相互に合意した）サブスクライバとシンジケータ間でイベント・ログを自動的に交換できるメカニズムを定義します。

ICE のその他の操作

ICE は、確立した関係でプロトコル・パラメータを再調整する機能、システム間での一方的な不定期の通知（テキストによるメッセージ）を送信する機能（管理者を最終的な送信先とする場合が多い）、関係の状態を問合せおよび確認する機能など、その他の操作をサポートする多くのメカニズムを提供します。

OSS のアーキテクチャ

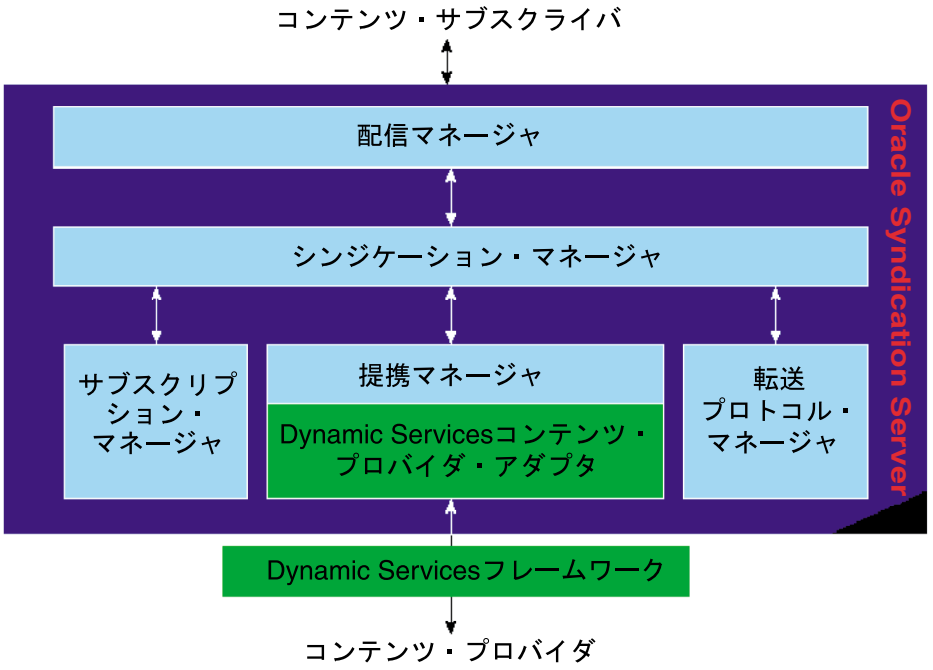
OSS は OracleAS Dynamic Services 上に構築され、Web サイト、インターネット・アプリケーションまたはデータベースからのあらゆるコンテンツ / 情報に XML 表現を適用します。また、XSL スタイルシートを使用して、XML をサブスクライバ固有のマークアップ言語に変換します。

各コンテンツ・ソースは、OracleAS Dynamic Services のサービスのセットとしてモデル化されます。サブスクライバ、コンテンツ・リソースおよび既存のサブスクリプションに関する情報は、OSS のレジストリに格納されます。実行時、OSS のエンジンは、関連するコンポーネントを起動してサブスクライバのコンテンツ要求を処理し、コンテンツ応答を戻します。コンテンツの更新が通知されると、OSS のエンジンは、関連するサブスクライバがあるローカルのレジストリを確認し、各サブスクライバに対してサブスクリプション内で指定さ

れた配信ポリシーに応じて更新された情報を送信します。OSS からのイベントは、Oracle データベースに記録されます。

図 19-2 に、Oracle Syndication Server のアーキテクチャの概要および主なコンポーネントを示します。Oracle Syndication Server（OSS）は、XML および Java を使用して、OracleAS Dynamic Services のフレームワーク上に構築されます。OSS は、業界標準の Web リスナーとサーブレット・コンテナに配置されている Java サーブレットを使用して通信を行い、メッセージを送受信します。サブスクライバおよびコンテンツ・リソースに関する情報は、サービスとして Oracle 上の OSS のレジストリに格納されます。

図 19-2 Oracle Syndication Server: アーキテクチャの概要



OSS のコンポーネントは、2 つの部分に分割されます。

- 直接または OracleAS Dynamic Services のフレームワークを介してコンテンツ・プロバイダと相互作用するコンポーネント
- コンテンツ・サブスクライバとコンテンツ・プロバイダ間の対話を管理する OSS のエンジン

コンテンツ・プロバイダとの対話

OSS は、OracleAS Dynamic Services（DS）を使用してコンテンツ・プロバイダと対話します。OSS は、Dynamic Services サービスのセットとして、コンテンツ・プロバイダを公開します。サブスクライバの要求の処理は、指定された Dynamic Services エンジン内の対応するサービスの実行にマップされます。OSS は、デジタル資産のハブとして機能し、場所、アクセス・プロトコルまたはコンテンツの形式に関係なく、あらゆるタイプのプロバイダからのコンテンツを集約します。

Dynamic Services Content Provider Adapter（DSCPA）

DSCPA は、OSS のエンジンが DS サービスのセットとして登録されている各コンテンツ・プロバイダにアクセスするためのインタフェースを提供します。OSS によって施行された場合、公開された各コンテンツ・プロバイダ・サービスは、事前定義された DS インタフェースに準拠している必要があります。実行時、OSS は指定された DSCPA を起動し、サブスクライバの要求から DS サービス要求を構成します。次に、DSCPA は、サービス要求を実行するため指定された DS エンジンに送信します。DS サービス・エンジンの応答は、DSCPA によって収集され、コンテンツ・プロバイダにトランザクションの終了をマークします。OSS では、各コンテンツ・プロバイダに少なくとも次に示す DS サービスのセットがある必要があります。

- DS のカタログ・サービス。コンテンツ・プロバイダに対して使用すると、使用可能なサブスクリプション提供に関するカタログ情報が提供されます。
- DS のサブスクリプション認証サービス。特定のコンテンツ・プロバイダに使用すると、OSS に格納する前にサブスクリプションを認証できます。サービスは、OSS で転送されたサブスクリプション要求を入力として受け入れ、コンテンツ・プロバイダがすべての条件について合意すると、認証されたサブスクリプションを戻します。
- DS のコンテンツ・アクセス・サービス。サブスクライバは、OSS へのコンテンツの送信だけでなく、OSS からコンテンツの受信を開始できます。
- DS のサブスクリプション取消しサービス。サブスクライバは、それぞれのサブスクリプションを取り消すことができます。取消しが発生すると、OSS は、コンテンツ・プロバイダによって実装された基礎となる「サブスクライブしない」DS サービスに、この取消しを転送します。これによって、コンテンツ・プロバイダは、サブスクリプションのクリーンアップを実行できます。

コンテンツ・サブスクライバとの対話

OSS が通信の基礎となるメカニズムとして ICE に実装されると、Content Subscriber Development Kit（CSDK）が OSS とバンドルされ、ICE を使用するコンテンツ・シンジケータと通信するアプリケーションをサブスクライバが容易に実装できるようになります。CSDK は、サブスクライバの観点から ICE メッセージの構成をある程度まで抽出できるクライアント・ライブラリを含みます。これによって、サーバーとの通信の基礎となるプロトコルと組み合せることなく、アプリケーションを簡単に開発できます。また、クライアント・ライブラリ以外にも、コンテンツ・アプリケーションの開発プロセスをさらに容易にする API ドキュメントのセットが提供されています。

コンテンツのサブスクライバへの配信

Transport Protocol Manager（TPM）は、OSS からサブスクライバへの送受信コンテンツの配信を処理します。コンテンツの送信では、サブスクライバはプロトコル固有の言語（多くの場合 XML ベースのマークアップ言語）を使用できます。

特定のサブスクライバにデータを配信する場合、TPM は XML 形式のコンテンツを適切なサブスクライバ固有のマークアップ言語に変換し、指定されたトランスポート層にまたがってコンテンツを配送します。これは、DS サービス・モデルによって行われ、各プロトコルに対して 1 つの配信サービスが適用されます。実行時、TPM は指定されたプロトコルに応じて DS 配信サービスを選択し、基礎となる DS フレームワークを介して実行します。

第 VI 部

XDK for Java

第 VI 部では、XDK for Java の XML コンポーネントの入手方法および使用方法を説明します。第 VI 部に含まれる章は、次のとおりです。

- 第 20 章「XML Parser for Java の使用」
- 第 21 章「XML Schema Processor for Java の使用」
- 第 22 章「XML Class Generator for Java の使用」

注意：

- XSU は、XDK for Java（および XDK for PL/SQL）の一部です。XSU の詳細は、第 7 章「XSU」を参照してください。
- XSQL Servlet は、XDK for Java の一部です。XSQL Servlet の詳細は、第 10 章「XSQL ページ・パブリッシング・フレームワーク」を参照してください。

次の章の最後には、FAQ が含まれます。

- 第 7 章 : 7-58 ページの「FAQ: XSU」
- 第 10 章 : 10-76 ページの「FAQ: XSQL Servlet」
- 第 20 章 : 20-59 ページの「FAQ: XML Parser for Java」
- 第 22 章 : 22-30 ページの「FAQ: Class Generator for Java」

20

XML Parser for Java の使用

この章の内容は次のとおりです。

XML Parser for Java の機能

オラクル社では、XML Parser for Java、XML Parser for C、XML Parser for C++ および XML Parser for PL/SQL を提供しています。これらの各 XML パーサーはスタンドアロンの XML コンポーネントであり、アプリケーションで処理できるように、XML 文書（または DTD や XML Schema）を解析します。ライブラリおよびコマンドラインは、次の標準および機能をサポートしています。

- **XML:** W3C の XML 1.0 勧告に準拠しています。
- **DOM:** 統合された DOM API で、次の勧告に準拠しています。
  - W3C の DOM 1.0 勧告
  - W3C の DOM 2.0 Core 勧告
  - W3C の DOM 2.0 Traversal 勧告（TreeWalker、Node Iterator および Node Filter を含む）

  これらの API によってアプリケーションは、XML 文書をツリー構造としてメモリー内でアクセスし操作できます。このインタフェースは、エディタなどのアプリケーションで使用されます。
- **SAX:** 統合された SAX API で、SAX 2.0 勧告に準拠しています。これらの API によってアプリケーションは、イベント駆動モデルを使用して XML 文書を処理できます。
- **W3C の XML Namespace 1.0 勧告案 :** これによって名前の競合が回避され、再利用性が向上し、アプリケーション統合が容易になります。また、Oracle XML Schema Processor がサポートされています。
  http://www.w3.org/TR/1999/REC-xml-names-19990114/ も参照してください。
- **XSLT:** XSLT Processor for Java には、次の機能があります。
  - W3C の XSLT 1.1 草案を統合サポートします。
  - XSLT を SAX 出力として取得するための新しい API を提供します。
- **XML Schema Processor:** 第 21 章「XML Schema Processor for Java の使用」を参照してください。XML Schema 定義ファイル（.xsd）を使用して XML ファイルを解析および検証する XML Schema Processor がサポートされています。このプロセッサには、次の機能があります。
  - XML Parser for Java に統合されています。
  - W3C の XML Schema 草案に含まれている次の 3 つのパートをサポートしています。
    * XML Schema Part 0: Primer（入門書）
    * XML Schema Part 1: Structures（構造）
    * XML Schema Part 2: Datatypes（データ型）
- Oracle および OracleAS で実行します。

その他にも次の機能があります。

- 検証モードおよび非検証モード
- 致命的エラーが発生するまでの組込みエラー・リカバリ
- ドキュメント作成のための DOM 拡張 API

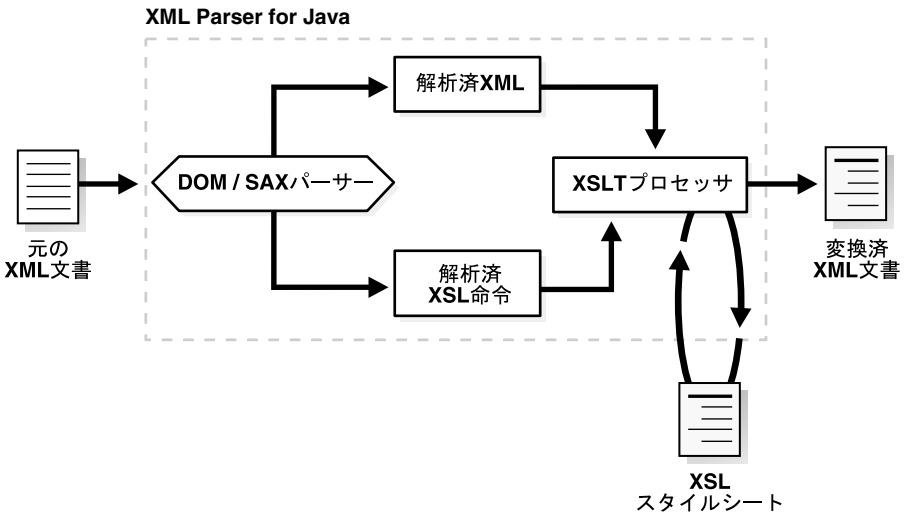
XML パーサーは、すべての Oracle プラットフォームで使用できます。

図 20-1 に、XML Parser for Java に読み取られる XML 文書を示します。DOM または SAX パーサー・インタフェースが、XML 文書を解析します。解析済 XML は、アプリケーションに転送され、さらに処理が行われます。

また、スタイルシートを使用する場合、DOM または SAX インタフェースは XSL コマンドを解析および出力します。これらは、解析済 XML とともに XSLT プロセッサに送信されます。XSLT プロセッサでは、選択したスタイルシートが適用され、変換済の（新しい）XML 文書が出力されます。

参照： 付録 C「XDK for Java: 仕様および早見表」を参照してください。

図 20-1 Oracle XML Parser



DOM および SAX API については、20-7 ページの「DOM API および SAX API」を参照してください。

XML 文書の解析に使用されるクラスおよびメソッドは、次の図を参照してください。

- 図 20-4「XML Parser for Java: DOMParser()」
- 図 20-5「SAXParser() クラスの使用」

XSLT プロセッサがスタイルシートの適用に使用するクラスおよびメソッドについては、次の図を参照してください。

- 図 20-6「XSLProcessor クラス・プロセス」

XSLT プロセッサ

XML パーサーには、XSL スタイルシートを使用して XML データを変換するための XSLT プロセッサが統合されています。XSLT プロセッサを使用すると、XML 文書を XML から XML、XML から HTML、またはほぼすべてのテキストベース形式の文書に変換できます。図 20-1 を参照してください。

XSLT プロセッサは、次の標準および機能をサポートします。

- W3C の XSL Transform 1.0 勧告案に準拠しています。
- W3C の XPath 1.0 勧告案に準拠しています。
- XML パーサーに統合され、パフォーマンスおよびスケーラビリティを向上します。
- Java、C、C++ および PL/SQL 用のライブラリおよびコマンドラインで使用可能です。

XML 名前空間のサポート

XML Parser for Java、XML Parser for C および XML Parser for C++ は、XML 名前空間もサポートしています。XML 名前空間は、XML 文書内にある要素タイプ（タグ）または属性間の名前の競合を解決または回避するメカニズムです。

このメカニズムは、汎用の名前空間での要素型および属性名を提供します。このマニュアルでは、これらについては詳しく説明していません。

このようなタグは、次のような URI によって修飾されます。

```
<oracle:EMP xmlns:oracle="http://www.oracle.com/xml"/>
```

たとえば、名前空間を使用して、オラクル社の <EMP> データ要素を、他社の <EMP> データ要素の定義と区別して識別できます。

これによってアプリケーションは、処理する要素および属性をより簡単に識別できます。XML Parser for Java、XML Parser for C および XML Parser for C++ は、汎用の要素型と属性名、およびローカルの非修飾の要素型と属性名を識別および解析できるようにすることで、名前空間をサポートします。

> **参照：** 第 21 章「XML Schema Processor for Java の使用」を参照してください。

Oracle XML Parser による 4 つの検証モードのサポート

XML Parser for Java、XML Parser for C および XML Parser for C++ は、検証モードまたは非検証モードで XML を解析できます。

- **非検証モード :** XML パーサーは、XML が整形式であることを確認し、データを DOM API が操作できるオブジェクトのツリーに解析します。
- **DTD 検証モード :** XML パーサーは、XML が整形式であることを確認し、その XML データを DTD（存在する場合）に対して妥当であるかどうかを検証します。
- **部分検証モード :** DTD または XML Schema が存在する場合、入力 XML 文書は DTD に対して検証されます。DTD または XML Schema が存在しない場合、XML パーサーは非検証モードになります。
- **スキーマ検証モード :** XML 文書は、その文書に対して指定された XML Schema のとおりに検証されます。
- **自動検証モード :** XML パーサーは、使用可能なすべてのものを使用して最適な検証を行います。DTD が使用可能である場合、XML パーサーは DTD_VALIDATION（DTD 検証）モードに設定されます。また、XML Schema が存在する場合は、SCHEMA_VALIDATION（スキーマ検証）モードに設定されます。どちらも使用可能でない場合は、NON_VALIDATING（非検証）モードに設定されます。

検証には、属性名および要素タグが正当であるかどうか、ネストした要素が適切な場所にあるかどうかなどの確認も含まれます。

参照 : 『Oracle9*i* XML リファレンス』を参照してください。

XML パーサーによる XML 文書のコンテンツおよび構造へのアクセス

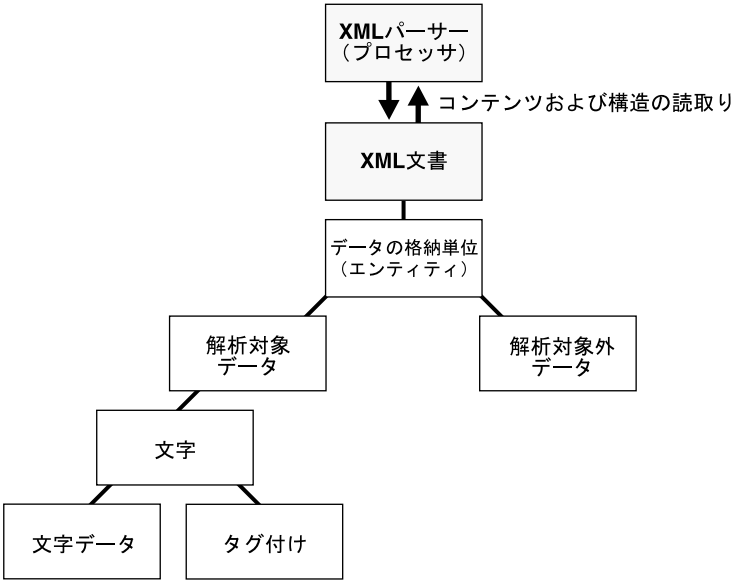
XML 文書は、エンティティというデータの格納単位で構成され、各エンティティには解析対象または解析対象外のデータが含まれます。解析対象データは文字で構成され、なかには文書内の文字データを形成したり、タグを形成するものもあります。

タグ付けは、文書のデータ格納のレイアウトおよび論理構造の記述をエンコーディングします。XML は、データ格納レイアウトおよび論理構造を制約するメカニズムを提供します。

XML 文書の読取り、およびその文書のコンテンツと構造へのアクセスには、XML プロセッサというソフトウェア・モジュールが使用されます。XML プロセッサは、アプリケーションという他のモジュールのかわりに作業を行います。

この解析プロセスを、図 20-2 に示します。

図 20-2 XML の解析プロセス



DOM API および SAX API

XML API は、通常、次のカテゴリのいずれかに分類されます。

- イベントベース
- ツリーベース

図 20-3 を参照してください。次の簡単な XML 文書について考えてみます。

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
     <ENAME>MARY</ENAME>
    </EMP>
    <EMP>
     <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

DOM: ツリーベース API

DOM などのツリーベース API は、XML 文書のメモリー内にツリーを構築します。この API は、アプリケーションがツリーを操作および処理するためのクラスおよびメソッドを提供します。

通常、DOM インタフェースは、要素の再順序付け、要素および属性の追加および削除、要素の名前の変更などの XML ツリーの構造的な操作に最も有効です。たとえば、前述の XML 文書では、DOM は図 20-3 に示すようなメモリー内のツリー構造を作成します。

SAX: イベントベース API

SAX などのイベントベース API は、コールを使用してアプリケーションに解析イベントを通知します。アプリケーションは、カスタマイズしたイベント・ハンドラによってこれらのイベントを処理します。イベントには、要素および文字の開始および終了も含まれます。

イベントベース API は、ツリーベース API とは異なり、通常 XML 文書のメモリー内ツリー表現を構築しません。したがって、SAX は、XML ツリーの操作が必要でない検索操作などのアプリケーションに有効です。

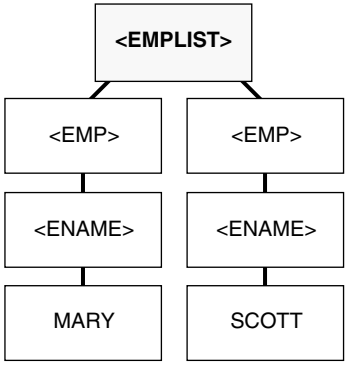
前述の XML 文書は、図 20-3 に示すような一連の線形のイベント群になります。

図 20-3 DOM（ツリーベース）API と SAX（イベントベース）API の比較

XML文書

```
<?XML Version = "1.0"?>
  <EMPLIST>
    <EMP>
      <ENAME>MARY</ENAME>
    </EMP>
    <EMP>
      <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

DOMインタフェースは、
XML文書に基づいて
ツリー構造を作成する



要素の再順序付け、追加、
削除などの変更を行うアプリケーションに有効

SAX インタフェースは、
XML文書に基づいて
一連の線形イベントを作成する

```
start document

start element: EMPLIST
start element: EMP
start element: ENAME
characters: MARY
end element: EMP

start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP

end element: EMPLIST
end document
```

XMLツリーを変更しない、
検索や取出しなどのアプリケーションに有効

DOM API および SAX API 使用時のガイドライン

この項では、DOM API および SAX API 使用時のガイドラインについて説明します。

DOM:

- DOM API は、ランダム・アクセスが必要なときに使用します。
- DOM は、より多くのメモリーを消費します。
- DOM は、変更を行う場合に使用します。
- DOM は、ツリーを反復したり、ドキュメント・ツリー全体を移動する場合に使用します。
- DOM インタフェースを使用する場合、パイプ・サイズが小さくなるように、XML 内で要素より多くの属性を使用してください。

SAX:

SAX API は、ほとんどのデータがストリーム形式である場合に使用します。

XML パーサーおよびデータ圧縮

Oracle XML Parser は、XML 文書を圧縮することもできます。圧縮機能を使用すると、メモリー内の DOM ツリー、または XML 文書から生成された SAX イベントを圧縮し、圧縮したバイナリ出力を生成できます。

DOM および SAX から生成された圧縮ストリームには互換性があるため、SAX から生成された圧縮ストリームを使用して DOM ツリーを生成したり、DOM から生成された圧縮ストリームを使用して SAX イベントを生成することができます。圧縮は、XML タグのトークン化に基づいています。これは、通常、XML ファイルにはタグの繰返しがあり、それらのタグをトークン化すると、データが圧縮されることを前提としています。圧縮は、入力 XML 文書のタイプに依存します。タグの数が多いほど、テキスト内容が減り、圧縮率が向上します。

通常、XML 文書と同様に、圧縮した XML データ出力はデータベースの CLOB として格納できます。

XML のシリアル化 / 圧縮

XML 文書は、メモリー内の DOM ツリーのシリアル化によって、バイナリ・ストリームに圧縮できます。大規模な XML 文書が解析され、それに対応してメモリー内に DOM ツリーが作成されると、メモリー要件を満たすことが困難になり、パフォーマンスが低下する場合があります。XML 文書は、バイト・ストリームに圧縮して、メモリー内の DOM ツリーに格納できます。これは、圧縮したストリーム内に格納された XML データに対する検証を行うことなく、後で拡張できます。

圧縮したストリームはシリアル化されたストリームとして処理できますが、Java のデフォルトのシリアル化によって実装される圧縮と比較すると、ストリーム内の情報はより厳密に制御および管理されることに注意してください。

今回のリリースには、次の 2 種類の XML 圧縮ストリームがあります。

- **SAX ベースの圧縮 :** 圧縮ストリームは、XML ファイルが SAX パーサーを使用して解析されたときに生成されます。SAX パーサーによって生成された SAX イベントは、SAX 圧縮ユーティリティによって処理されます。このユーティリティは、SAX イベントを処理し、圧縮したバイナリ・ストリームを生成します。バイナリ・ストリームが再度読み取られると、SAX イベントが生成されます。
- **DOM ベースの圧縮 :** 解析済 XML 文書に対応するメモリー内 DOM ツリーがシリアル化され、圧縮した XML 出力ストリームが生成されます。このシリアル化されたストリームは、再度読み取られると、DOM ツリーを再生成します。

SAX イベントを使用して生成された圧縮ストリームは、DOM のシリアル化を使用して生成された圧縮ストリームと互換性があります。SAX イベントによって生成された圧縮ストリームを使用して DOM ツリーを生成したり、DOM ツリーによって生成された圧縮ストリームを使用して SAX イベントを生成することができます。使用される圧縮アルゴリズムは、XML タグのトークン化に基づいています。これは、すべての XML ファイルにはタグの繰返しがあるため、それらのタグをトークン化すると、データが大幅に圧縮されることを前提としています。

参照： 次の項および章を参照してください。

- 2-13 ページの「データベースへの XML のロード」
- 第 5 章「XML に対するデータベース・サポート」

XDK for Java のアップグレード

以前のリリースから Oracle への XDK for Java のアップグレード

XDK for Java がすでにインストールされており、それを Oracle にアップグレードする場合は、次の手順を実行します。

1. Oracle JVM（JServer）が正常にアップグレードされていることを確認します。
2. *ORACLE_HOME*/rdbms/admin ディレクトリに移動します。
3. SQL*Plus を起動します。
4. SYSDBA 権限を持つユーザーとしてデータベース・インスタンスに接続します。
5. STARTUP を次のとおり実行します。

   ```
   SQL> STARTUP
   ```

   PFILE オプションを使用して、初期化パラメータ・ファイルの場所を指定する必要がある場合があります。

6. アップグレードする前のリリースに応じて、適切なアップグレード・スクリプトを実行します。

   リリース 8.1.5 からアップグレードする場合は、xmlu815.sql を次のとおり実行します。

   ```
   SQL> @xmlu815.sql
   ```

   リリース 8.1.6 からアップグレードする場合は、xmlu816.sql を次のとおり実行します。

   ```
   SQL> @xmlu816.sql
   ```

   リリース 8.1.7 からアップグレードする場合は、xmlu817.sql を次のとおり実行します。

   ```
   SQL> @xmlu817.sql
   ```

7. SHUTDOWN を次のとおり実行して、すべてのインスタンスを停止します。

   ```
   SQL> SHUTDOWN
   ```

8. SQL*Plus を終了します。

これによって、XDK for Java コンポーネントが新しいリリースにアップグレードされます。

セッション・ネームスペース、CORBA および OSE のアップグレード

1. JServer および XDK for Java が正常にアップグレードされていることを確認します。
2. 新しいリリースの Oracle ホーム・ディレクトリ所有者としてシステムにログインします。
3. システム・プロンプトで、*ORACLE_HOME*/javavm/install ディレクトリへ移動します。
4. SQL*Plus を起動します。
5. SYSDBA 権限を持つユーザーとしてデータベース・インスタンスに接続します。
6. インスタンスが実行されている場合は、SHUTDOWN IMMEDIATE を使用してインスタンスを停止します。

   ```
   SQL> SHUTDOWN IMMEDIATE
   ```

> **注意：** Oracle9*i* Real Application Clusters に対して、CLUSTER_DATABASE 初期化パラメータを false に設定します。アップグレード処理が完了した後、true に戻すことができます。

7. RESTRICT モードでインスタンスを起動します。

```
SQL> STARTUP RESTRICT
```

PFILE オプションを使用して、初期化パラメータ・ファイルの場所を指定する必要がある場合があります。

8. ロールバック・セグメント領域に 100MB 以上の空き領域があることを確認します。

> **参照：** ロールバック・セグメントの管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

9. リスナーを起動します。
10. アップグレード元のリリースに対応する適切なアップグレード・スクリプトを実行します。

リリース 8.1.5 からアップグレードする場合は、jisu815.sql を実行します。

```
SQL> @jisu815.sql
```

リリース 8.1.6 からアップグレードする場合は、jisu816.sql を実行します。

```
SQL> @jisu816.sql
```

リリース 8.1.7 からアップグレードする場合は、jisu817.sql を実行します。

```
SQL> @jisu817.sql
```

11. インスタンスを停止します。

```
SQL> SHUTDOWN
```

12. SQL*Plus を終了します。

注意： Enterprise JavaBeans のアップグレードはサポートされていません。以前のリリースで Enterprise JavaBeans を配置している場合は、Oracle9*i* リリース 1（9.0.1）用に再配置する必要があります。詳細は、『Oracle9*i* Enterprise JavaBeans Developer's Guide and Reference』を参照してください。

JSP のアップグレード

Oracle システムに JSP がインストールされている場合は、次の手順を実行します。

1. JServer、XDK for Java、セッション・ネームスペース、CORBA および OSE が正常にアップグレードされていることを確認します。
2. 新しいリリースの Oracle ホーム・ディレクトリ所有者としてシステムにログインします。
3. システム・プロンプトで、*ORACLE_HOME*/javavm/install ディレクトリへ移動します。
4. SQL*Plus を起動します。
5. SYSDBA 権限を持つユーザーとしてデータベース・インスタンスに接続します。
6. インスタンスが実行されている場合は、SHUTDOWN IMMEDIATE を使用してインスタンスを停止します。

   ```
   SQL> SHUTDOWN IMMEDIATE
   ```

   注意： Oracle9*i* Real Application Clusters に対して、CLUSTER_DATABASE 初期化パラメータを false に設定します。アップグレード処理が完了した後、true に戻すことができます。

7. RESTRICT モードでインスタンスを起動します。

   ```
   SQL> STARTUP RESTRICT
   ```

   PFILE オプションを使用して、初期化パラメータ・ファイルの場所を指定する必要がある場合があります。

8. ロールバック・セグメント領域に 100MB 以上の空き領域があることを確認します。

   参照： ロールバック・セグメントの管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

9. アップグレード元のリリースに対応する適切なアップグレード・スクリプトを実行します。

   リリース 8.1.5 からアップグレードする場合は、jspu815.sql を実行します。

   ```
   SQL> @jspu815.sql
   ```

   リリース 8.1.6 からアップグレードする場合は、jspu816.sql を実行します。

   ```
   SQL> @jspu816.sql
   ```

   リリース 8.1.7 からアップグレードする場合は、jspu817.sql を実行します。

   ```
   SQL> @jspu817.sql
   ```

10. インスタンスを停止します。

    ```
    SQL> SHUTDOWN
    ```

11. SQL*Plus を終了します。

Oracle8*i* リリース 8.1 へのダウングレード

『Oracle9*i* データベース移行ガイド』の第 13 章を参照してください。

XML Parser for Java のサンプルの実行

表 20-1 に、XDK for Java ソフトウェアに付属している XML Parser for Java のサンプルを示します。これらのサンプルは、sample/ サブディレクトリにあります。これらのサンプルは、Oracle XML Parser for Java の使用方法を示しています。

表 20-1 XML Parser for Java のサンプル

サンプル・ファイル名	説明
DOMSample.java	DOM API を使用するサンプル・アプリケーション
SAXSample.java	SAX API を使用するサンプル・アプリケーション
XSLSample.java	XSL API を使用するサンプル・アプリケーション
DOMNamespace.java	DOM API に名前空間による拡張を使用するサンプル・アプリケーション
SAXNamespace.java	SAX API に名前空間による拡張を使用するサンプル・アプリケーション

サンプル・プログラムを実行するには、次の手順を実行します。

1. 「**make**」を使用して .class ファイルを生成します。
2. xmlparserv2.jar および現在のディレクトリを CLASSPATH に追加します。
3. DOM/SAX API のサンプル・プログラムを次のとおり実行します。

```
java <classname> <sample xml file>
```

4. XSL API のサンプル・プログラムを次のとおり実行します。

```
java XSLSample <sample xsl file> <sample xml file>
```

class.xml、empl.xml、family.xml などのいくつかの XML ファイルがテスト用に提供されています。

XSL スタイルシート iden.xsl は、提供された次の XML ファイルの変換を確認するために使用できます。

- class.xml
- NSExample.xml
- family.xml
- empl.xml

XML Parser for Java - XML の例 1: class.xml

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for Java - XML の例 2: DTD（employee）の使用 - employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee [
<!ELEMENT employee (Name, Dept, Title)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
]>
<employee>
<Name>John Goodman</Name>
<Dept>Manufacturing</Dept>
<Title>Supervisor</Title>
</employee>
```

XML Parser for Java - XML の例 3: DTD（family.dtd）の使用 - family.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
<member memberid="m1">Sarah</member>
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

DTD: family.dtd

```
<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>
```

XML Parser for Java - XSL の例 1: XSL （iden.xsl）

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

XML Parser for Java - DTD の例 1: （NSExample）

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:a1 CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:a1 = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
</child>
</doc>
```

XML Parser for Java の使用 : DOMParser() クラス

DOM ベースのパーサー・アプリケーションを作成するには、次のクラスを使用します。

- `DOMNamespace()` クラス
- `DOMParser()` クラス
- `XMLParser()` クラス

DOMParser は XMLParser の拡張であるため、XMLParser のすべてのメソッドを DOMParser にも使用できます。図 20-4 に、`DOMParser()` クラスを使用したコードを作成する際に必要な主な手順を示します。

- **DTD を入力しない場合**

  1. 新しい `DOMParser()` クラスがコールされます。このクラスに使用可能なプロパティは次のとおりです。
     - `setValidateMode`
     - `setPreserveWhiteSpace`
     - `setDocType`
     - `setBaseURL`
     - `showWarnings`
  2. 1. の結果が、XML 入力とともに XMLParser.parse() に渡されます。XML 入力は、ファイル、文字列バッファまたは URL のいずれかになります。
  3. XMLParser.getDocument() メソッドを使用します。
  4. オプションで、次のような他の DOM メソッドを適用できます。
     - `print()`
     - `DOMNamespace()` メソッド
  5. XML Parser for Java が、DOM ツリーとして XML（解析済）文書を出力します。
  6. オプションで、XML Parser for Java が DOM ツリーの構築を終了した後で、`DOMParser.reset()` を使用してすべての内部データ構造を削除します。

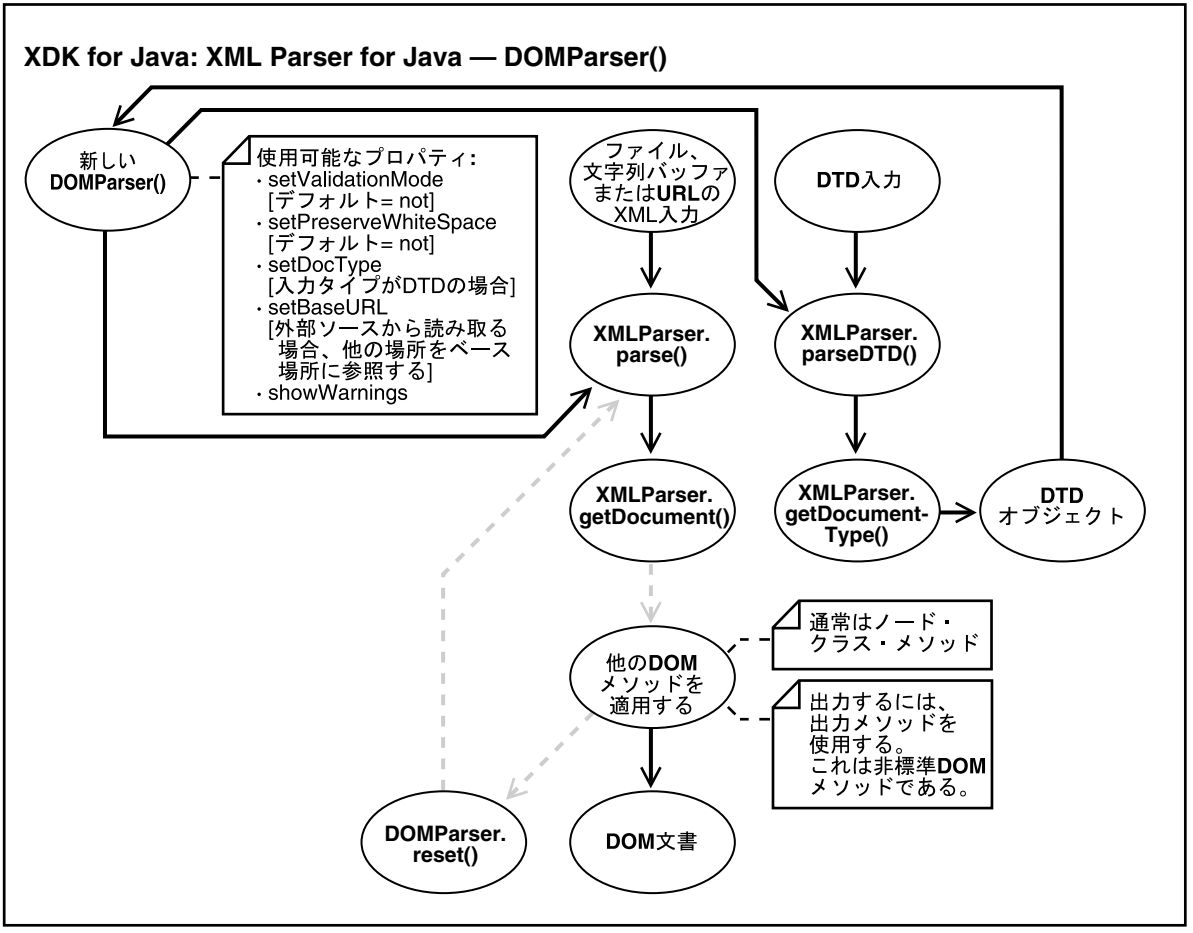
- **DTD を入力する場合**

  1. 新しい DOMParser() クラスがコールされます。このクラスに適用可能なプロパティは次のとおりです。
     - setValidateMode
     - setPreserveWhiteSpace
     - setDocType
     - setBaseURL
     - showWarnings
  2. 1. の結果が、DTD 入力とともに XMLParser.parseDTD() メソッドに渡されます。
  3. XMLParser.getDocumentType() メソッドが、DTD の結果オブジェクトを新しい DOMParser() に戻します。このプロセスは、DTD の適用が終了するまで続きます。

DOMParser() クラスの使用方法を、次の例に示します。

- 20-21 ページの「XML Parser for Java の例 1: XML Parser for Java および DOM API（DomSample.java）の使用」

図 20-4 XML Parser for Java: DOMParser()



XML Parser for Java の例 1: XML Parser for Java および DOM API（DomSample.java）の使用

次の例では、コードの作成方法を示すため、Java コーディング標準（拡張されたすべてのインポートなど）を、メソッドの前のドキュメント・ヘッダーなどを使用する例を示します。

```
// This file demonstates a simple use of the parser and DOM API.
// The XML file given to the application is parsed.
// The elements and attributes in the document are printed.
// This demonstrates setting the parser options.
//

import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;

import oracle.xml.parser.v2.*;

public class DOMSample
{
   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
         {
            // Must pass in the name of the XML file.
            System.err.println("Usage: java DOMSample filename");
            System.exit(1);
         }

         // Get an instance of the parser
         DOMParser parser = new DOMParser();

         // Generate a URL from the filename.
         URL url = createURL(argv[0]);

         // Set various parser options: validation on,
         // warnings shown, error stream set to stderr.
         parser.setErrorStream(System.err);
         parser.setValidationMode(DTD_validation);
         parser.showWarnings(true);
```

```
      // Parse the document.
      parser.parse(url);

      // Obtain the document.
      XMLDocument doc = parser.getDocument();

      // Print document elements
      System.out.print("The elements are: ");
      printElements(doc);

      // Print document element attributes
      System.out.println("The attributes of each element are: ");
      printElementAttributes(doc);
      parser.reset();
   }
   catch (Exception e)
   {
      System.out.println(e.toString());
   }
}

static void printElements(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Node n;

   for (int i=0; i<nl.getLength(); i++)
   {
      n = nl.item(i);
      System.out.print(n.getNodeName() + " ");
   }

   System.out.println();
}

static void printElementAttributes(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element e;
   Node n;
   NamedNodeMap nnm;
```

```
   String attrname;
   String attrval;
   int i, len;

   len = nl.getLength();
   for (int j=0; j < len; j++)
   {
      e = (Element)nl.item(j);
      System.out.println(e.getTagName() + ":");
      nnm = e.getAttributes();
      if (nnm != null)
      {
         for (i=0; i<nnm.getLength(); i++)
         {
            n = nnm.item(i);
            attrname = n.getNodeName();
            attrval = n.getNodeValue();
            System.out.print(" " + attrname + " = " + attrval);
         }
      }
      System.out.println();
   }
}

static URL createURL(String fileName)
{
   URL url = null;
   try
   {
      url = new URL(fileName);
   }
   catch (MalformedURLException ex)
   {
      File f = new File(fileName);
      try
      {
         String path = f.getAbsolutePath();
         String fs = System.getProperty("file.separator");
         if (fs.length() == 1)
         {
            char sep = fs.charAt(0);
            if (sep != '/')
               path = path.replace(sep, '/');
            if (path.charAt(0) != '/')
               path = '/' + path;
         }
```

```
            path = "file://" + path;
            url = new URL(path);
          }
          catch (MalformedURLException e)
          {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
          }
        }
        return url;
    }
}
```

DOMParser() の例 1 に関するコメント

図 20-4 を参照してください。次に、例 1 に関するコメントを示します。

1. 新しい DOMParser() を宣言します。例 1 には次の行があります。

   ```
   DOMParser parser = new DOMParser();
   ```

   このクラスでは、いくつかのプロパティが使用できます。プロパティの指定例を示します。

   ```
   parser.setErrorStream(System.err);
   parser.setValidationMode(DTD_validation);
   parser.showWarnings(true);
   ```

2. XML 入力は、次のとおり宣言される URL です。

   ```
   URL url = createURL(argv[0])
   ```

3. XML 文書は、URL として入力されます。この文書は、次のとおり parser.parse() を使用して解析されます。

   ```
   parser.parse(url);
   ```

4. 文書が取得されます。

   ```
   XMLDocument doc = parser.getDocument();
   ```

5. 他の DOM メソッドを適用します。この例では、次のメソッドを適用できます。
   - ノード・クラス・メソッド
     - getElementsByTagName()
     - getAttributes()
     - getNodeName()
     - getNodeValue()
   - createURL() メソッドは、文字列名を URL に変換します。
6. DOM ツリーが作成されると、解析プロセス中に作成されたすべてのデータ構造を削除するために parser.reset() がコールされます。
7. アプリケーションで追加の処理を行うために、DOM ツリー（解析済 XML）としてドキュメントが生成されます。

注意： 例 1 では、DTD 入力は示されていません。

XML Parser for Java の使用 : DOMNamespace() クラス

図 20-4 に、DOM インタフェースを使用した XML 文書の解析の主なプロセスを示します。DOMNamespace() メソッドは、パーサー・プロセスの「他の DOM メソッドを適用する」と書かれた楕円の位置で適用されます。DOMNamespace() の使用方法を、次の例に示します。

- 「XML Parser for Java の例 2: URL の解析 - DOMNamespace.java」

XML Parser for Java の例 2: URL の解析 - DOMNamespace.java

```
// This file demonstates a simple use of the parser and Namespace
// extensions to the DOM APIs.
// The XML file given to the application is parsed and the
// elements and attributes in the document are printed.
//

import java.io.*;
import java.net.*;

import oracle.xml.parser.v2.DOMParser;

import org.w3c.dom.*;
import org.w3c.dom.Node;
```

```
// Extensions to DOM Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLAttr;


public class DOMNamespace
{
   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
         {
            // Must pass in the name of the XML file.
            System.err.println("Usage: DOMNamespace filename");
            System.exit(1);
         }

         // Get an instance of the parser
         Class cls = Class.forName("oracle.xml.parser.v2.DOMParser");
         DOMParser parser = (DOMParser)cls.newInstance();

         // Generate a URL from the filename.
         URL url = createURL(argv[0]);

         // Parse the document.
         parser.parse(url);

         // Obtain the document.
         Document doc = parser.getDocument();

         // Print document elements
         printElements(doc);

         // Print document element attributes
         System.out.println("The attributes of each element are: ");
         printElementAttributes(doc);
      }
      catch (Exception e)
      {
         System.out.println(e.toString());
      }
   }
```

```
static void printElements(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   XMLElement nsElement;

   String qName;
   String localName;
   String nsName;
   String expName;

   System.out.println("The elements are: ");
   for (int i=0; i < nl.getLength(); i++)
   {
      nsElement = (XMLElement)nl.item(i);

      // Use the methods getQualifiedName(), getLocalName(), getNamespace()
      // and getExpandedName() in NSName interface to get Namespace
      // information.

      qName = nsElement.getQualifiedName();
      System.out.println("  ELEMENT Qualified Name:" + qName);

      localName = nsElement.getLocalName();
      System.out.println("  ELEMENT Local Name    :" + localName);

      nsName = nsElement.getNamespace();
      System.out.println("  ELEMENT Namespace     :" + nsName);

      expName = nsElement.getExpandedName();
      System.out.println("  ELEMENT Expanded Name :" + expName);
   }

   System.out.println();
}

static void printElementAttributes(Document doc)
{
   NodeList nl = doc.getElementsByTagName("*");
   Element e;
   XMLAttr nsAttr;
   String attrname;
   String attrval;
   String attrqname;
```

```
      NamedNodeMap nnm;
      int i, len;
      len = nl.getLength();
      for (int j=0; j < len; j++)
      {
         e = (Element) nl.item(j);
         System.out.println(e.getTagName() + ":");
         nnm = e.getAttributes();

         if (nnm != null)
         {
            for (i=0; i < nnm.getLength(); i++)
            {
               nsAttr = (XMLAttr) nnm.item(i);

               // Use the methods getQualifiedName(), getLocalName(),
               // getNamespace() and getExpandedName() in NSName
               // interface to get Namespace information.

               attrname = nsAttr.getExpandedName();
               attrqname = nsAttr.getQualifiedName();
               attrval = nsAttr.getNodeValue();

               System.out.println(" " + attrqname + "(" + attrname + ")" + " = " +
attrval);
            }
         }
         System.out.println();
      }
   }

   static URL createURL(String fileName)
   {
      URL url = null;
      try
      {
         url = new URL(fileName);
      }
      catch (MalformedURLException ex)
      {
         File f = new File(fileName);
         try
         {
            String path = f.getAbsolutePath();
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
```

```
            char sep = fs.charAt(0);
            if (sep != '/')
               path = path.replace(sep, '/');
            if (path.charAt(0) != '/')
               path = '/' + path;
         }
         path = "file://" + path;
         url = new URL(path);
      }
      catch (MalformedURLException e)
      {
         System.out.println("Cannot create url for: " + fileName);
         System.exit(0);
      }
   }
   return url;
  }
}
```

注意 : 例 2 では、DTD 入力は示されていません。

XML Parser for Java の使用 : SAXParser() クラス

アプリケーションは、SAX ハンドラを登録して様々なパーサー・イベントの通知を受信できます。XMLReader は、XML パーサーの SAX2 ドライバが実装する必要があるインタフェースです。このインタフェースによって、アプリケーションは XML パーサーの機能およびプロパティを問合せおよび設定したり、イベント・ハンドラを登録してドキュメントを処理したり、ドキュメントの解析を開始することができます。

すべての SAX インタフェースは、同期操作を前提としています。そのため、解析メソッドは解析が完了するまで結果を戻さず、リーダーは次のイベントを通知する前にイベント・ハンドラのコールバックが結果を戻すまで待機する必要があります。

このインタフェースは（現在は使用されない）SAX 1.0 パーサー・インタフェースに代わるものです。XMLReader インタフェースには、SAX 1.0 パーサー・インタフェースに対する次の 2 つの重要な拡張が含まれています。

- 機能およびプロパティを問合せおよび設定する標準の方法が追加されています。
- 多くのより高水準の XML 標準に必要な名前空間をサポートします。

表 20-2 に、SAXParser() クラスのメソッドを示します。

表 20-2 SAXParser() クラスのメソッド

メソッド	説明
getContentHandler()	現行のコンテンツ・ハンドラを戻します。
getDTDHandler()	現行の DTD ハンドラを戻します。
getEntityResolver()	現行のエンティティ・リゾルバを戻します。
getErrorHandler()	現行のエラー・ハンドラを戻します。
getFeature(java.lang.String name)	機能の値を検索します。
getProperty(java.lang.String name)	プロパティの値を検索します。
setContentHandler(ContentHandler handler)	アプリケーションがコンテンツ・イベント・ハンドラを登録できるようにします。
setDocumentHandler(DocumentHandler handler)	SAX 2.0 では使用できず、setContentHandler に代わります。
setDTDHandler(DTDHandler handler)	アプリケーションが DTD イベント・ハンドラを登録できるようにします。
setEntityResolver(EntityResolver resolver)	アプリケーションがエンティティ・リゾルバを登録できるようにします。
setErrorHandler(ErrorHandler handler)	アプリケーションがエラー・イベント・ハンドラを登録できるようにします。
setFeature(java.lang.String name, boolean value)	機能の状態を設定します。
setProperty(java.lang.String name, java.lang.Object value)	プロパティの値を設定します。

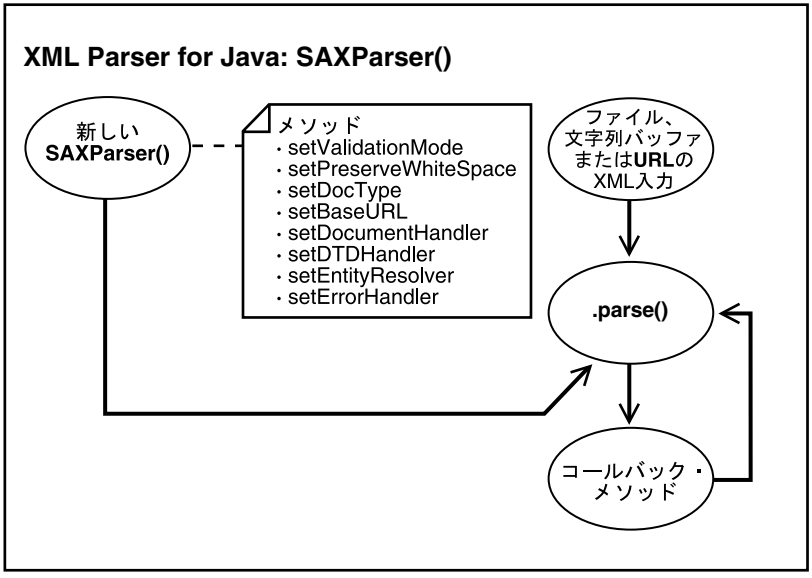
図 20-5 に、SAXParser() クラスを使用したコードを作成するための主な手順を示します。

1. 新しい SAXParser() クラスを宣言します。表 20-2 に、使用可能なメソッドを示します。
2. 1. の結果が、ファイル、文字列または URL 形式の XML 入力とともに parse() に渡されます。
3. 解析が完了すると、解析メソッドが結果を戻します。一方、このプロセスは、次のイベントを通知する前にイベント・ハンドラのコールバックが結果を戻すまで待機します。
4. 解析済 XML 文書は、アプリケーションでさらに処理を行うことができます。

SAXParser() クラスおよびいくつかのハンドラ・インタフェースの使用方法を、次の例に示します。

- 「XML Parser for Java の例 3: XML Parser for Java および SAX API（SAXSample.java）の使用」

図 20-5　SAXParser() クラスの使用



XML Parser for Java の例 3: XML Parser for Java および SAX API（SAXSample.java）の使用

```
// This file demonstates a simple use of the parser and SAX API.
// The XML file given to the application is parsed and
// prints out some information about the contents of this file.
//

import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXSample extends HandlerBase
{
   // Store the locator
   Locator locator;

   static public void main(String[] argv)
   {
      try
      {
         if (argv.length != 1)
         {
            // Must pass in the name of the XML file.
            System.err.println("Usage: SAXSample filename");
            System.exit(1);
         }
         // (1) Create a new handler for the parser
         SAXSample sample = new SAXSample();

         // (2) Get an instance of the parser
         Parser parser = new SAXParser();

         // (3) Set Handlers in the parser
         parser.setDocumentHandler(sample);
         parser.setEntityResolver(sample);
         parser.setDTDHandler(sample);
         parser.setErrorHandler(sample);
```

```
      // (4) Convert file to URL and parse
      try
      {
         parser.parse(fileToURL(new File(argv[0])).toString());
      }
      catch (SAXParseException e)
      {
         System.out.println(e.getMessage());
      }
      catch (SAXException e)
      {
         System.out.println(e.getMessage());
      }
   }
   catch (Exception e)
   {
      System.out.println(e.toString());
   }
}

static URL fileToURL(File file)
{
   String path = file.getAbsolutePath();
   String fSep = System.getProperty("file.separator");
   if (fSep != null && fSep.length() == 1)
      path = path.replace(fSep.charAt(0), '/');
   if (path.length() > 0 && path.charAt(0) != '/')
      path = '/' + path;
   try
   {
      return new URL("file", null, path);
   }
   catch (java.net.MalformedURLException e)
   {
      throw new Error("unexpected MalformedURLException");
   }
}
```

```
//////////////////////////////////////////////////////////////////////
// (5) Sample implementation of DocumentHandler interface.
//////////////////////////////////////////////////////////////////////

public void setDocumentLocator (Locator locator)
{
   System.out.println("SetDocumentLocator:");
   this.locator = locator;
}

public void startDocument()
{
   System.out.println("StartDocument");
}
public void endDocument() throws SAXException
{
   System.out.println("EndDocument");
}

public void startElement(String name, AttributeList atts)
                                              throws SAXException
{
   System.out.println("StartElement:"+name);
   for (int i=0;i<atts.getLength();i++)
   {
      String aname = atts.getName(i);
      String type = atts.getType(i);
      String value = atts.getValue(i);

      System.out.println("   "+aname+"("+type+")"+"="+value);
   }

}
public void endElement(String name) throws SAXException
{
   System.out.println("EndElement:"+name);
}

public void characters(char[] cbuf, int start, int len)
{
   System.out.print("Characters:");
   System.out.println(new String(cbuf,start,len));
}
```

```
public void ignorableWhitespace(char[] cbuf, int start, int len)
{
   System.out.println("IgnorableWhiteSpace");
}


public void processingInstruction(String target, String data)
          throws SAXException
{
   System.out.println("ProcessingInstruction:"+target+" "+data);
}

//////////////////////////////////////////////////////////////////////
// (6) Sample implementation of the EntityResolver interface.
//////////////////////////////////////////////////////////////////////

public InputSource resolveEntity (String publicId, String systemId)
                  throws SAXException
{
   System.out.println("ResolveEntity:"+publicId+" "+systemId);
   System.out.println("Locator:"+locator.getPublicId()+" "+
               locator.getSystemId()+
               " "+locator.getLineNumber()+" "+locator.getColumnNumber());
   return null;
}

//////////////////////////////////////////////////////////////////////
// (7) Sample implementation of the DTDHandler interface.
//////////////////////////////////////////////////////////////////////

public void notationDecl (String name, String publicId, String systemId)
{
   System.out.println("NotationDecl:"+name+" "+publicId+" "+systemId);
}

public void unparsedEntityDecl (String name, String publicId,
      String systemId, String notationName)
{
   System.out.println("UnparsedEntityDecl:"+name + " "+publicId+" "+
      systemId+" "+notationName);
}
```

```
    ////////////////////////////////////////////////////////////////////
    // (8) Sample implementation of the ErrorHandler interface.
    ////////////////////////////////////////////////////////////////////

    public void warning (SAXParseException e)
          throws SAXException
    {
       System.out.println("Warning:"+e.getMessage());
    }

    public void error (SAXParseException e)
          throws SAXException
    {
       throw new SAXException(e.getMessage());
    }


    public void fatalError (SAXParseException e)
          throws SAXException
    {
       System.out.println("Fatal error");
       throw new SAXException(e.getMessage());
    }
}
```

XML Parser for Java の使用 : XSLT プロセッサ

XML Parser for Java に XSLT プロセッサを実装するには、`XSLProcessor` クラスを使用します。

図 20-6 に、`XSLProcessor` クラスが使用する全体的なプロセスを示します。

1. 新しい `XSLProcessor()` クラスを宣言すると、XSLT プロセスが開始します。
2. 2 つの入力があります。
   - **スタイルシート :** 最初に、スタイルシートが構築されます。新しい `XSLStylesheet()` クラスは、次の使用可能なメソッドのいずれかを使用して宣言されます。
     - `removeParam()`
     - `resetParam()`
     - `setParam()`
   - **XML 入力 :** これは、特定のスタイルシートに対して 1 ～ N 回繰り返すことができます。これによって、「スタイルシートを処理する」の手順が行われます。

   どちらの入力も、次の 4 つのタイプのいずれかになります。
   - 入力ストリーム
   - URL
   - XML 文書
   - リーダー
3. 次のようにして、スタイルシートの結果オブジェクトおよび XML 入力が「スタイルシートを処理する」の手順に送られます。

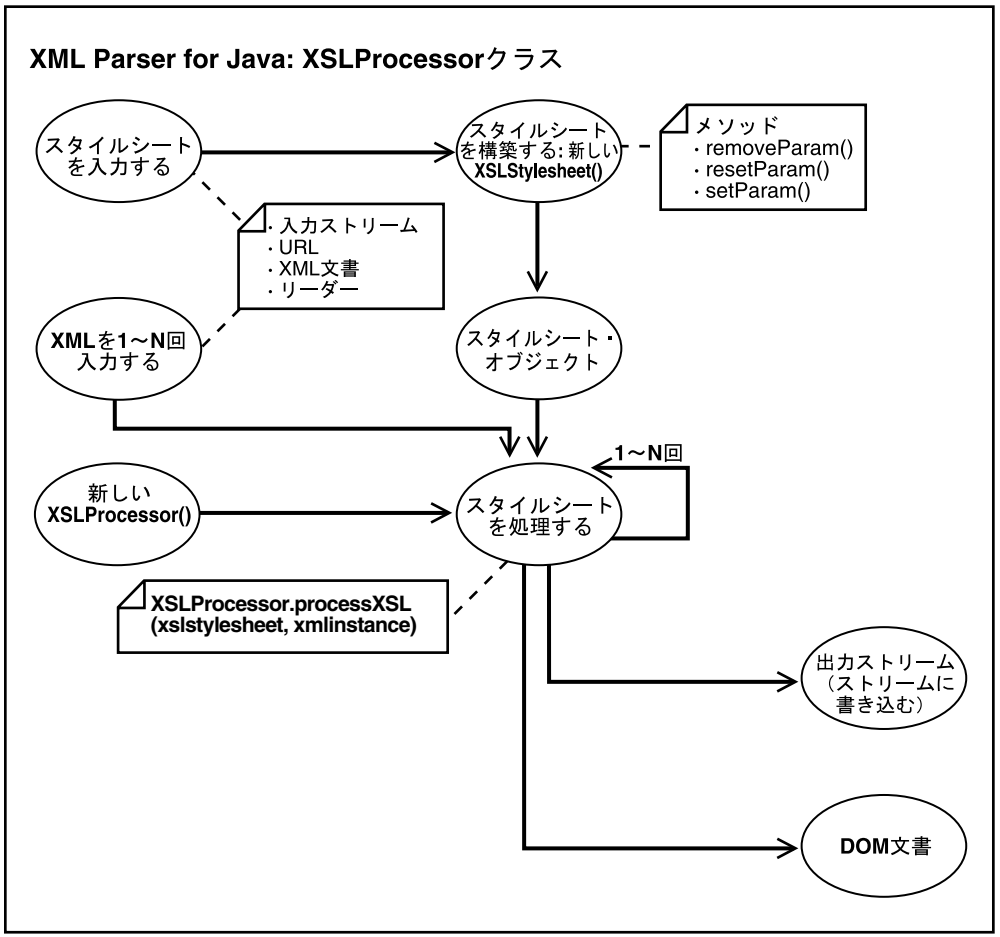
   ```
   XSLProcessor.processXSL(xslstylesheet, xmlinstance)
   ```

4. `XSLProcessor.processXSL()` メソッドが、選択されたスタイルシートを使用して、XML 入力を 1 ～ N 回処理します。
5. `XSLProcessor.processXSL()` が、出力ストリームか DOM 文書のいずれかを出力します。

XML Parser for Java の XSLT プロセッサを、次の例に示します。

- 「XML Parser for Java の例 4: （XSLSample.java）」
- 「XML Parser for Java の例 5: DOM API および XSLT プロセッサの使用」

図 20-6 XSLProcessor クラス・プロセス



XML Parser for Java の例 4: （XSLSample.java）

```
/**
 * This file gives a simple example of how to use the XSL processing
 * capabilities of the Oracle XML Parser V2.0. An input XML document is
 * transformed using a given input stylesheet
 */

import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class XSLSample
{
   /**
    * Transforms an xml document using a stylesheet
    * @param args input xml and xml documents
    */
   public static void main (String args[]) throws Exception
   {
      DOMParser parser;
      XMLDocument xml, xsldoc, out;
      URL xslURL;
      URL xmlURL;

      try
      {

         if (args.length != 2)
         {
            // Must pass in the names of the XSL and XML files
            System.err.println("Usage: java XSLSample xslfile xmlfile");
            System.exit(1);
         }

         // Parse xsl and xml documents

         parser = new DOMParser();
         parser.setPreserveWhitespace(true);
```

```
            // parser input XSL file
            xslURL = createURL(args[0]);
            parser.parse(xslURL);
            xsldoc = parser.getDocument();

            // parser input XML file
            xmlURL = createURL(args[1]);
            parser.parse(xmlURL);
            xml = parser.getDocument();

            // instantiate a stylesheet
            XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);
            XSLProcessor processor = new XSLProcessor();

            // display any warnings that may occur
            processor.showWarnings(true);
            processor.setErrorStream(System.err);

            // Process XSL
            DocumentFragment result = processor.processXSL(xsl, xml);

            // create an output document to hold the result
            out = new XMLDocument();

            // create a dummy document element for the output document
            Element root = out.createElement("root");
            out.appendChild(root);

            // append the transformed tree to the dummy document element
            root.appendChild(result);

            // print the transformed document
            out.print(System.out);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
```

```
   // Helper method to create a URL from a file name
   static URL createURL(String fileName)
   {
      URL url = null;
      try
      {
         url = new URL(fileName);
      }
      catch (MalformedURLException ex)
      {
         File f = new File(fileName);
         try
         {
            String path = f.getAbsolutePath();
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
               char sep = fs.charAt(0);
               if (sep != '/')
                  path = path.replace(sep, '/');
               if (path.charAt(0) != '/')
                  path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
         }
         catch (MalformedURLException e)
         {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
         }
      }
      return url;
   }
}
```

XML Parser for Java の例 5: DOM API および XSLT プロセッサの使用

このコード例は、sample/ サブディレクトリにはありません。次の Java コードでは、XML Parser for Java を使用して次のタスクを実行します。

- XML 文書を解析します。
- DOM API を使用して XML データを操作します。
- XSLT プロセッサを使用してデータを変換します。

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class XSLTransform
{
   public static void main (String args[]) throws Exception
   {
      DOMParser parser;
      XMLDocument xml, xsldoc, out;

      URL xslURL;
      URL xmlURL;

      try
      {
         if (args.length != 2)
         {
      // Pass in the names of the XSL and XML files
         System.err.println("Usage: java XSLTransform
            xslfile xmlfile");
         System.exit(1);
         }

      // Parse XSL and XML documents
         parser = new DOMParser();
         parser.setPreserveWhitespace(true);

         xslURL = createURL(args[0]);
         parser.parse(xslURL);
         xsldoc = parser.getDocument();
```

```
            xmlURL = createURL(args[1]);
            parser.parse(xmlURL);
            xml = parser.getDocument();

         // Instantiate the stylesheet
            XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);

            XSLProcessor processor = new XSLProcessor();

         // Display any warnings that may occur
            processor.showWarnings(true);
            processor.setErrorStream(System.err);

         // Process XSL
            DocumentFragment result = processor.processXSL(xsl, xml);

         // Create an output document to hold the result
            out = new XMLDocument();

         // Create a dummy document element for the output document
            Element root = out.createElement("root");
            out.appendChild(root);

         // Append the transformed tree to the dummy document element
             root.appendChild(result);

         // Print the transformed document
            out.print(System.out);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
   }
```

XSLT の例 5 に関するコメント

図 20-4 および図 20-6 を参照してください。次に、例 5 に関するコメントを示します。

1. プログラムでは、次の 2 つの URL ドキュメントが入力されています。
   - URL xmlURL
   - URL xslURL
2. 2 つのドキュメントを解析し、空白を保存するためのプロパティを設定します。

```
parser = new DOMParser();
parser.setPreserveWhitespace(true);
```

3. XSL および XML 文書を取得します。

```
xslURL = createURL(args[0]);
parser.parse(xslURL);
xsldoc = parser.getDocument();

xmlURL = createURL(args[1]);
xmlURL = createURL(args[1]);
parser.parse(xmlURL);
xml = parser.getDocument();
```

4. 新しい `XSLStylesheet` および `XSLProcessor` クラスを初期化します。

```
XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);

XSLProcessor processor = new XSLProcessor();
processor.setErrorStream(System.err);
```

5. スタイルシートを処理します。

```
DocumentFragment result = processor.processXSL(xsl, xml);
```

6. DOM として解析済 XML 文書を出力します。

```
out = new XMLDocument();
Element root = out.createElement("root");
out.appendChild(root);
root.appendChild(result);
```

XML Parser for Java の使用 : SAXNamespace() クラス

SAXNamespace() クラスの使用方法を、次の例に示します。

XML Parser for Java の例 6: （SAXNamespace.java）

```
// This file demonstrates a simple use of the Namespace extensions to
// the SAX APIs.

import org.xml.sax.*;
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;

// Extensions to the SAX Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLDocumentHandler;
import oracle.xml.parser.v2.DefaultXMLDocumentHandler;
import oracle.xml.parser.v2.NSName;
import oracle.xml.parser.v2.SAXAttrList;

import oracle.xml.parser.v2.SAXParser;

public class SAXNamespace {
  static public void main(String[] args) {
     String fileName;

     //Get the file name
     if (args.length == 0)
     {
        System.err.println("No file Specified!!!");
        System.err.println("USAGE: java SAXNamespace <filename>");
        return;
     }
     else
     {
        fileName = args[0];
     }
```

```
    try {
       // Create handlers for the parser
       // Use the XMLDocumentHandler interface for namespace support
       // instead of org.xml.sax.DocumentHandler
       XMLDocumentHandler xmlDocHandler = new XMLDocumentHandlerImpl();

       // For all the other interface use the default provided by
       // Handler base
       HandlerBase defHandler = new HandlerBase();

       // Get an instance of the parser
       SAXParser parser = new SAXParser();

       // Set Handlers in the parser
       // Set the DocumentHandler to XMLDocumentHandler
       parser.setDocumentHandler(xmlDocHandler);

       // Set the other Handler to the defHandler
       parser.setErrorHandler(defHandler);
       parser.setEntityResolver(defHandler);
       parser.setDTDHandler(defHandler);

       try
       {
          parser.parse(fileToURL(new File(fileName)).toString());
       }
       catch (SAXParseException e)
       {
          System.err.println(args[0] + ": " + e.getMessage());
       }
       catch (SAXException e)
       {
          System.err.println(args[0] + ": " + e.getMessage());
       }
    }
    catch (Exception e)
    {
       System.err.println(e.toString());
    }
}
```

```
static public URL fileToURL(File file)
   {
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
      path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')
      path = '/' + path;
    try {
      return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e) {
      /* According to the spec this could only happen if the file
 protocol were not recognized. */
      throw new Error("unexpected MalformedURLException");
    }
  }

  private SAXNamespace() throws IOException
   {
   }

}
   /**************************************************************************
     Implementation of XMLDocumentHandler interface. Only the new
     startElement and endElement interfaces are implemented here. All other
     interfaces are implemented in the class HandlerBase.
     *************************************************************************/

class XMLDocumentHandlerImpl extends DefaultXMLDocumentHandler
{

   public void XMLDocumentHandlerImpl()
   {
   }


   public void startElement(NSName name, SAXAttrList atts) throws SAXException
   {
```

```
    // Use the methods getQualifiedName(), getLocalName(), getNamespace()
    // and getExpandedName() in NSName interface to get Namespace
    // information.
    String qName;
    String localName;
    String nsName;
    String expName;
    qName = name.getQualifiedName();
    System.out.println("ELEMENT Qualified Name:" + qName);
    localName = name.getLocalName();
    System.out.println("ELEMENT Local Name    :" + localName);

    nsName = name.getNamespace();
    System.out.println("ELEMENT Namespace     :" + nsName);

    expName = name.getExpandedName();
    System.out.println("ELEMENT Expanded Name :" + expName);

    for (int i=0; i<atts.getLength(); i++)
    {

    // Use the methods getQualifiedName(), getLocalName(), getNamespace()
    // and getExpandedName() in SAXAttrList interface to get Namespace
    // information.
       qName = atts.getQualifiedName(i);
       localName = atts.getLocalName(i);
       nsName = atts.getNamespace(i);
       expName = atts.getExpandedName(i);

       System.out.println(" ATTRIBUTE Qualified Name   :" + qName);
       System.out.println(" ATTRIBUTE Local Name       :" + localName);
       System.out.println(" ATTRIBUTE Namespace        :" + nsName);
       System.out.println(" ATTRIBUTE Expanded Name    :" + expName);

       // You can get the type and value of the attributes either
       // by index or by the Qualified Name.
       String type = atts.getType(qName);
       String value = atts.getValue(qName);

       System.out.println(" ATTRIBUTE Type             :" + type);
       System.out.println(" ATTRIBUTE Value            :" + value);
       System.out.println();
    }
}
```

```
  public void endElement(NSName name) throws SAXException
   {
      // Use the methods getQualifiedName(), getLocalName(), getNamespace()
      // and getExpandedName() in NSName interface to get Namespace
      // information.
      String expName = name.getExpandedName();
      System.out.println("ELEMENT Expanded Name  :" + expName);
   }
}
```

XML Parser for Java: コマンドライン・インタフェース

oraxml - Oracle XML Parser

oraxml は、XML 文書を解析し、XML 文書が整形式であるか、および妥当であるかを確認するコマンドライン・インタフェースです。

oraxml を使用するには、次の条件を確認する必要があります。

- 環境変数 CLASSPATH が、Oracle XML Parser for Java に付属する xmlparserv2.jar ファイルを指すように設定されている。
- 環境変数 PATH が、JDK 1.1.x または JDK 1.2 に付属する Java インタプリタを検索できる。
- oraxml はスキーマ検証をサポートするため、xschema.jar も CLASSPATH に含める。

oraxml を起動するには、次の構文を使用します。

```
oraxml options* source
```

oraxml には、通常、解析する XML ファイルが指定されます。表 20-3 に、oraxml のコマンドライン・オプションを示します。

表 20-3 oraxml: コマンドライン・オプション

オプション	用途
-h	ヘルプ・モード（oraxml 起動構文を出力します。）
-v	部分検証モード（このオプションを使用しない場合は、解析では整形式であるかどうかのみが確認されます。）
-s	厳密な検証モード
-w	警告の表示（デフォルトでは、オフになっています。）
-debug	デバッグ・モード（デフォルトでは、オフになっています。）
-e < エラー・ログ >	エラーを書き込むファイル（エラーおよび警告を書き込むログ・ファイルを指定します。）

oraxsl - Oracle XSL Processor

oraxsl は、複数の XML 文書へのスタイルシートの適用に使用されるコマンドライン・インタフェースです。このインタフェースには、動作を指定する多くのコマンドライン・オプションがあります。

oraxsl を使用するには、次の条件を確認する必要があります。

- 環境変数 CLASSPATH が、Oracle XML Parser for Java に付属する xmlparserv2.jar ファイルを指すように設定されている。
- 環境変数 PATH が、JDK 1.1.x または JDK 1.2 に付属する Java インタプリタを検索できる。

oraxsl を起動するには、次の構文を使用します。

```
oraxsl options* source? stylesheet? result?
```

oraxsl には、通常、スタイルシート、変換する XML ファイルおよび結果ファイル（オプション）が指定されます。結果ファイルが指定されない場合、このインタフェースは、変換済文書を標準出力として出力します。複数の XML 文書をスタイルシートによって変換する必要がある場合は、かわりに -l または -d オプションを -s および -r オプションと組み合せて使用します。これらのオプションおよび他のオプションについては、表 20-4 を参照してください。

表 20-4 oraxsl: コマンドライン・オプション

オプション	用途
-h	ヘルプ・モード（oraxsl 起動構文を出力します）。
-v	冗長モード（デバッグ情報のいくつかが出力されます。この出力は、処理中に発生した問題の追跡に有効な場合があります）。
-w	警告の表示（デフォルトでは、オフになっています）。
-debug	新規オプション - デバッグ・モード（デフォルトでは、オフになっています）。
-e < エラー・ログ >	エラーを書き込むファイル（エラーおよび警告を書き込むログ・ファイルを指定します）。
-t < スレッド数 >	処理に使用するスレッド数（複数のスレッドを使用すると、複数文書の処理時のパフォーマンスを向上できます）。
-l <XML ファイル・リスト >	変換するファイルのリスト（処理するファイルを明示的に示すことができます）。
-d < ディレクトリ >	変換するファイルがあるディレクトリ（デフォルト動作では、ディレクトリ内のすべてのファイルが処理されます）。そのディレクトリ内にあるファイルの特定のサブセット（1 つのファイルのみなど）を処理する必要がある場合、-l を使用して処理するファイルのみを指定し、この動作を変更する必要があります。-x または -i を使用して拡張子に基づいてファイルを選択することで、この動作を変更することもできます。
-x < ソースの拡張子 >	排除する拡張子（-d とともに使用します。指定した拡張子を持つすべてのファイルが排除されます）。
-i < ソースの拡張子 >	挿入する拡張子（-d とともに使用します。指定した拡張子を持つファイルのみが選択されます）。
-s < スタイルシート >	使用するスタイルシート（-d または -l を指定する場合、このオプションを使用して、使用するスタイルシートを指定する必要があります。完全なパスを指定する必要があります）。
-r < 結果の拡張子 >	結果に使用する拡張子（-d または -l を指定する場合、このオプションを指定して、変換の結果に使用する拡張子を指定する必要があります。このため、拡張子「out」を指定して入力文書「foo」を変換すると、「foo.out」になります。デフォルトでは、結果は現在のディレクトリに出力されます。この動作は、結果を保存するディレクトリを指定する -o を使用して変更できます）。
-o < 結果のディレクトリ >	結果を保存するディレクトリ（-r とともに使用する必要があります）。
-p	パラメータのリスト。

XSLT プロセッサ用の XML 拡張関数

XSLT プロセッサの拡張関数 : 概要

XSLT プロセッサ用の XML 拡張関数を使用すると、XSLT プロセッサのユーザーは、特定の Java メソッドを XSL の式からコールできます。Java の拡張関数は、次の URL で始まる名前空間にあります。

```
http://www.oracle.com/XSL/Transform/java/
```

次の名前空間にある拡張関数を想定します。

```
http://www.oracle.com/XSL/Transform/java/classname
```

これは、`classname` クラスにあるメソッドを参照します。たとえば、次の名前空間があるとします。

```
http://www.oracle.com/XSL/Transform/java/java.lang.String
```

この名前空間は、XSL の式から `java.lang.String` メソッドをコールするために使用できます。

静的メソッドと非静的メソッドの比較

メソッドがクラスの非静的メソッドである場合、最初のパラメータはメソッドが起動されるインスタンスとして使用され、残りのパラメータがメソッドに渡されます。

拡張関数が静的メソッドである場合、その拡張関数のすべてのパラメータが、静的関数へのパラメータとして渡されます。

XML Parser for Java - XSL の例 1: 静的関数

次に、XSL の静的関数の例を示します。

```
<xsl:stylesheet
xmlns:math="http://www.oracle.com/XSL/Transform/java/java.lang.Math">
  <xsl:template match="/">
  <xsl:value-of select="math:ceil('12.34')"/>
</xsl:template>
</xsl:stylesheet>
```

「13」が出力されます。

コンストラクタ拡張関数

拡張関数「new」は、クラスの新しいインスタンスを作成し、コンストラクタとして動作します。

XML Parser for Java - XSL の例 2: コンストラクタ拡張関数

次に、コンストラクタ関数の例を示します。

```
<xsl:stylesheet
xmlns:jstring="http://www.oracle.com/XSL/Transform/java/java.lang.String">
  <xsl:template match="/">
  <!-- creates a new java.lang.String and stores it in the variable str1 -->
  <xsl:variable name="str1" select="jstring:new('Hello World')"/>
  <xsl:value-of select="jstring:toUpperCase($str1)"/>
</xsl:template>
</xsl:stylesheet>
```

「HELLO WORLD」が出力されます。

戻り値拡張関数

拡張関数の結果は、XSL に定義されている次の 5 つの型を含むどの型でも出力できます。

- NodeList
- Boolean
- String
- Number
- Resulttree

これらの結果は、変数に格納するか、他の拡張関数に渡すことができます。

結果の型が XSL に定義されている 5 つの型のうちの 1 つである場合、その結果は XSL の式の結果として戻すことができます。

XML Parser for Java - XSL の例 3: 戻り値拡張関数

次に、戻り値拡張関数を表す XSL の例を示します。

```
<!-- Declare extension function namespace -->
<xsl:stylesheet xmlns:parser =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.DOMParser"
xmlns:document =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.XMLDocument" >

<xsl:template match ="/"> <!-- Create a new instance of the parser, store it in
myparser variable -->
<xsl:variable name="myparser" select="parser:new()"/>
<!-- Call a non-static method of DOMParser. Since the method is anon-static method,
the first parameter is the instance on which themethod is called. This is equivalent
to $myparser.parse('test.xml') -->
<xsl:value-of select="parser:parse($myparser, 'test.xml')"/>
<!-- Get the document node of the XML Dom tree -->
<xsl:variable name="mydocument" select="parser:getDocument($myparser)"/>
<!-- Invoke getelementsbytagname on mydocument -->
<xsl:for-each select="document:getElementsByTagName($mydocument,'elementname')">
......
</xsl:for-each> </xsl:template>
</xsl:stylesheet>
```

データ型拡張関数

パラメータ数および型に基づいたオーバーロードがサポートされています。暗黙的な型変換は、XSL に定義されたとおりに 5 つの XSL 型の間で行われます。

型変換は、String、Number、Boolean、ResultTree 間、および NodeSet から String、Number、Boolean、ResultTree へ暗黙的に行われます。

相互に暗黙的な変換が可能な 2 つの型に基づくオーバーロードは許可されません。

XML Parser for Java - XSL の例 4: データ型拡張関数

次のオーバーロードでは、String および Number は相互に暗黙的な変換が可能なため、XSL でエラーが発生します。

- abc(int i){}
- abc(String s){}

XSL 型と Java 型間のマッピングは次のとおり行われます。

```
String -> java.lang.String
Number -> int, float, double
Boolean -> boolean
NodeSet -> XMLNodeList
ResultTree -> XMLDocumentFragment
```

ora XSLT 組込み拡張 : ora:node-set および ora:output

次の例は、動作中の ora:node-set と ora:output の両方を示します。

次のコードを実行するとします。

```
$ oraxsl foo.xml slides.xsl toc.html
```

ここで、foo.xml は任意の XML ファイルです。これによって、次のものを取得します。

- 目次を含む toc.html スライド
- スライド 1 を含む slide01.html ファイル
- スライド 2 を含む slide02.html ファイル

```
<!--
                                        | Illustrate using ora:node-set and
ora:output
                                        |
                                        | Both extensions depend on defining a
namespace
                                        | with the uri of
"http://www.oracle.com/XSL/Transform/java"
                                        +-->
                                       <xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:ora="http://www.oracle.com/XSL/Transform/java">

                                        <!-- <xsl:output> affects the primary
result document -->
                                        <xsl:output mode="html" indent="no"/>
```

```
                                        <!--
                                         |   <ora:output> at the top-level allows
all attributes
                                         |   that <xsl:output> allows, but you
must provide the
                                         |   additional "name" attribute to
assign a name to
                                         |   these output settings to be used
later.
                                         +-->
                                        <ora:output name="myOutput" mode="html"
indent="no"/>
                                        <!--
                                         | This top-level variable is a
result-tree fragment
                                         +-->
                                        <xsl:variable name="fragment">
                                          <slides>
                                            <slide>
                                              <title>First Slide</title>
                                              <bullet>Point One</bullet>
                                              <bullet>Point Two</bullet>
                                              <bullet>Point Three</bullet>
                                            </slide>
                                            <slide>
                                              <title>Second Slide</title>
                                              <bullet>Point One</bullet>
                                              <bullet>Point Two</bullet>
                                              <bullet>Point Three</bullet>
                                            </slide>
                                          </slides>
                                        </xsl:variable>
                                        <xsl:template match="/">
                                          <!--
                                           | We cannot "de-reference" a
result-tree-fragment to
                                           | navigate into it with an XPath
expression. However, using
                                           | the ora:node-set() built-in
extension function, you can
                                           | "cast" a result-tree fragment to a
node-set which *can*
```

```
                                        | then be navigated using XPath. Since
we'll use the node-set
                                        | of <slides> twice below, we save the
node-set in a variable.
                                        +-->
                                       <xsl:variable name="slides"
select="ora:node-set($fragment)"/>
                                       <!--
                                        | This <html> page will go to the
primary result document.
                                        | It is a "table of contents" for the
slide show, with
                                        | links to each slide. The "slides"
will each be generated
                                        | into *secondary* result documents,
each slide having
                                        | a file name of "slideNN.html" where
NN is the two-digit
                                        | slide number
                                        +-->
                                       <html>
                                         <body>
                                           <h1>List of All Slides</h1>
                                           <xsl:apply-templates
select="$slides" mode="toc"/>
                                         </body>
                                       </html>
                                       <!--
                                        | Now go apply-templates to format
each slide
                                        +-->
                                       <xsl:apply-templates select="$slides"/>
                                     </xsl:template>
                                     <!-- In 'toc' mode, generate a link to
each slide we match -->
                                     <xsl:template match="slide" mode="toc">
                                       <a
href="slide{format-number(position(),'00')}.html">
                                         <xsl:value-of select="title"/>
                                       </a><br/>
                                     </xsl:template>
```

```
                                        <!--
                                         | For each slide matched, send the
output for the current
                                         | <slide> to a file named
"slideNN.html". Use the named
                                         | output style defined above called
"myOutput".
                                         +-->
                                        <xsl:template match="slide">
                                          <ora:output use="myOutput"

href="slide{format-number(position(),'00')}.html">
                                            <html>
                                              <body>
                                                <xsl:apply-templates
select="title"/>
                                                <ul>
                                                  <xsl:apply-templates
select="*[not(self::title)]"/>
                                                </ul>
                                              </body>
                                            </html>
                                          </ora:output>
                                        </xsl:template>
                                        <xsl:template match="bullet">
                                          <li><xsl:value-of select="."/></li>
                                        </xsl:template>
                                        <xsl:template match="title">
                                          <h1><xsl:value-of select="."/></h1>
                                        </xsl:template>
                                      </xsl:stylesheet>
```

FAQ: XML Parser for Java

XML Parser for Java の FAQ は、次の項目で構成されています。

DTD

XML 文書と相対位置にある DOCTYPE の DTD ファイル

質問

XML パーサーが DTD ファイルを検出できません。

回答

<!DOCTYPE> 宣言に定義されている DTD ファイルは、入力 XML 文書と相対位置にある必要があります。相対位置にない場合は、setBaseURL(url) ファンクションを使用してベース URL を設定し、入力が InputStream からの場合に DTD の相対アドレスを解決する必要があります。

外部 DTD を使用した XML ファイルの検証

質問

外部 DTD を使用して XML ファイルを検証できますか？

回答

XML 文書に、適用可能な DTD への参照を含める必要があります。この参照がないと、XML パーサーは検証に使用する DTD を検出できません。参照の指定方法は、外部 DTD を指定するときに使用する XML の標準方法です。参照を指定しない場合、XML 文書に DTD を埋め込む必要があります。

DTD のキャッシュ機能

質問

DTD キャッシュ機能はありますか？DTD をキャッシュするために XML パーサーを使用して DTD を設定する方法を教えてください。

回答

キャッシュ機能はありますが、オプションであり、自動的には有効になりません。

DTD を設定するメソッドは、setDoctype() です。次に例を示します。

```
// Test using InputSource
parser = new DOMParser();
parser.setErrorStream(System.out);
parser.showWarnings(true);

FileReader r = new FileReader(args[0]);
InputSource inSource = new InputSource(r);
inSource.setSystemId(createURL(args[0]).toString());
parser.parseDTD(inSource, args[1]);
dtd = (DTD)parser.getDoctype();

r = new FileReader(args[2]);
inSource = new InputSource(r);
inSource.setSystemId(createURL(args[2]).toString());
parser.setDoctype(dtd);
parser.setValidationMode(DTD_validation);
parser.parse(inSource);

doc = (XMLDocument)parser.getDocument();
doc.print(new PrintWriter(System.out));
```

外部 DTD の認識

質問

XML Parser for Java は、サーバーから実行している場合はどのようにして外部 DTD を認識しますか？Java コードは loadjava によってロードされ、Oracle サーバー・プロセスで実行しています。XML ファイルには外部 DTD 参照があります。

1. SAXParser による方法のような、DTD がデータベースにある場合にストリームや文字列などにリダイレクトするためのより一般的な方法はありますか？
2. SAXParser の resolveEntity() による方法のような、DTD をリダイレクトするためのより一般的な方法はありますか？

回答

1. 現在使用できるメソッドは、setBaseURL() のみです。
2. 次の方法で、希望する結果を得ることができます。
   a. DOMParser の parseDTD() メソッドを使用して外部 DTD を解析します。
   b. getDoctype() をコールして oracle.xml.parser.v2.DTD のインスタンスを取得します。
   c. DTD をプログラム的に設定するドキュメントに、setDoctype(yourDTD); を使用します。この方法は、オラクル社の製品である JAR ファイル以外から DTD を読み取る場合に使用します。

jar ファイルからの外部 DTD のロード

質問

すべての DTD を jar ファイルに挿入して、XML パーサーに DTD が必要なときに、jar から取得するように設定する必要があります。現行の XML パーサーはベース URL（setBaseURL()）をサポートしていますが、この URL は、DTD が公開されている場所を指すのみです。

回答

この問題を解決するには、次の操作を組み合せて使用します。

1. 次の構文を使用して DTD を InputStream としてロードします。

```
InputStream is = YourClass.class.getResourceAsStream("/foo/bar/your.dtd");
```

   これによって、DTD が指定されている CLASSPATH 上の最初の相対位置にある /foo/bar/your.dtd がオープンされます。/foo/bar/your.dtd が CLASSPATH にある場合は、jar 以外の場所からオープンされます。

2. 次のコードによって DTD を解析します。

```
DOMParser d = new DOMParser();
d.parseDTD(is, "rootelementname");
d.setDoctype(d.getDoctype());
```

3. ここで、次の構文を使用して文書を解析します。

```
d.parse("yourdoc");
```

DTD を使用した XML 文書の正確性の確認

質問

Java オブジェクトを XML にエクスポートしています。XML 文書で DOM を構築し、出力メソッドを使用して DOM をエクスポートすることはできますが、これらの文書の DTD を設定できません。パーサーを構築して DTD を解析し、
Document doc = parser.getDocument() および DocType dtd = doc.getDocumentType() を使用してその DTD を取得します。

新しく構築した XML 文書の DTD を設定し、この DTD を後で使用して文書の正確性を確認する方法を教えてください。

回答

DTD オブジェクトの取得方法は正しいです。ただし、DOM API を使用して DOM ツリーを作成している間は検証を行うことができません。このため、文書に DTD を設定しても、構築した DOM ツリーの検証には有効ではありません。XML ファイルを検証する唯一の方法は、DOMParser または SAXParser を使用して XML 文書を解析することです。

XML 文書と分離した DTD オブジェクトの解析

質問

XML 文書の解析とは別に、DTD オブジェクトを解析および取得する方法を教えてください。

回答

parseDTD() メソッドを使用すると、DTD ファイルを個別に解析し、DTD オブジェクトを取得できます。これを行うコード例を次に示します。

```
DOMParser domparser = new DOMParser();
domparser.setValidationMode(DTD_validation);
/* parse the DTD file */
domparser.parseDTD(new FileReader(dtdfile));
DTD dtd = domparser.getDocType();
```

DTD に対する解析検証での大文字 / 小文字の区別

質問

XML ファイルに、<xn:subjectcode> というタグがあります。DTD では、このタグは <xn:subjectCode> と定義されています。このファイルをこの DTD に対して解析および検証すると、次のエラーが発生します。

```
XML-00148: 無効な要素 'xn:subjectcode' ('xn:Resource' の内容) です ...
```

要素名を <xn:subjectcode> から <xn:subjectCode> に変更すると、このエラーは発生しなくなります。パーサーは、DTD に対する検証では大文字 / 小文字を区別しますか ? それとも、要素のタグ定義内に名前空間があり、その名前空間とともに要素が定義されると、大文字 / 小文字の区別が有効になるのですか ?

回答

XML は基本的に、大文字 / 小文字を区別します。これに準拠するために、Oracle XML Parser は大文字 / 小文字を区別します。非検証モードで実行すると、整形式であるかどうかのみが確認されます。ただし、<test></Test> は、非検証モードでもエラーになります。

CDATA セクションからの埋込み XML フォームの取得

質問

1. PAYLOAD を取り出して追加の処理を行う必要があります。
2. PAYLOAD の値を選択しても、データが CDATA セクション内にあるため解析されません。
3. XSLT のみを使用して埋込み XML を取り出す方法を教えてください。以前は SAX を使用してこれを行うことができましたが、現在の設定では XSLT しか使用できません。

回答

1. ご希望の操作は、次のとおり行うことができます。

```
<PAYLOAD>
<![CDATA[<?xml version = '1.0' encoding = 'ASCII' standalone = 'no'?>
<ADD_PO_003>
   <CNTROLAREA>
      <BSR>
         <VERB value="ADD">ADD</VERB>
         <NOUN value="PO">PO</NOUN>
         <REVISION value="003">003</REVISION>
      </BSR>
   </CNTROLAREA>
</ADD_PO_003>]]>
</PAYLOAD>
```

CDATA による方法は、ある意味で不適切です。CDATA 内にテキストとして挿入されているネストした XML 文書には、異なるエンコーディングを使用できません。このため、埋込み文書の XML 宣言を取得することは、あまり意味がありません。XML 宣言を必要としない場合は、CDATA の動作を示すテキスト・チャンクのかわりに、実際の要素としてメッセージを <PAYLOAD> に埋め込むことをお薦めします。

これは次のとおり行います。

```
String s = YourDocumentObject.selectSingleNode("/OES_MESSAGE/PAYLOAD");
```

2. 結果を大きいテキスト・チャンクで戻すように指定しているため、データは解析されません。このテキスト・チャンクは、ユーザーが次のとおり解析する必要があります (CDATA 方法を使用しないメリットの 1 つです)。

```
  YourParser.parse( new StringReader(s));
```

s は、前述の手順で戻された文字列です。

3. CDATA に特別な要素はありません。テキストのみです。山カッコをエスケープしないでテキストの内容を出力する必要がある場合は、次の構文を使用します。

```
 <xsl:value-of select="/OES_MESSAGE/PAYLOAD" disable-output-escaping="yes"/>
```

DOMParser.parseDTD() のコール時にエラーが発生する理由

質問

Oracle XML Parser for Java を使用して DTD を作成および解析できません。
DOMParser.parseDTD() メソッドをコールすると、次のエラーが発生します。

属性値は引用符で開始してください。

DTD を確認し、何が問題であるかを教えてください。

```
<?xml version = "1.0" encoding="UTF-8" ?>
<!-- RCS_ID = "$Header: XMLRenderer.dtd 115.0 2000/09/18 03:00:10 fli noship $" -->
<!-- RCS_ID_RECORDED = VersionInfo.recordClassVersion(RCS_ID,
"oracle.apps.mwa.admin") -->
<!--  Copyright: This DTD file is owned by Oracle Mobile Application Server   Group.
-->
  <!ELEMENT    page    (header?,form,footer?) >
  <!ATTLIST    page
               name    CDATA   #REQUIRED
               lov     (Y|N)   'N' >
  <!ELEMENT    header EMPTY >
  <!ATTLIST    header
               name    CDATA   #REQUIRED
               title   CDATA
               home    (Y|N)   'N'
               portal  (Y|N)   'N'
               logout  (Y|N)   'N' >
  <!ELEMENT    footer EMPTY >
  <!ATTLIST    footer
               name    CDATA   #REQUIRED
               home    (Y|N)   'N'
               portal  (Y|N)   'N'
               logout  (Y|N)   'N'
               copyright (Y|N) 'N' >

  <!ELEMENT    form
  (styledText|textInput|list|link|menu|submitButton|table|separator)+ >
  <!ATTLIST    form
               name    CDATA    #REQUIRED
               title    CDATA
               type     CDATA >
```

```
<!ELEMENT    styledText    (#PCDATA) >

<!ELEMENT    textInput    EMPTY >
<!ATTLIST    textInput
             name    CDATA    #REQUIRED
             prompt    CDATA    #IMPLIED
             password    (Y|N)    'N'
             required    (Y|N)    'N'
             maxlength    #IMPLIED
             size    #IMPLIED
             format    #IMPLIED
             default    #IMPLIED >

<!ELEMENT    link (postfield*) >
<!ATTLIST    link
             name    CDATA    #REQUIRED
             title    CDATA    #REQUIRED
             baseurl    CDATA    #REQUIRED >
```

回答

DTD 構文が有効ではありません。CDATA によって ATTLIST を宣言する場合は、#REQUIRED、#IMPLIED、#FIXED、任意の値、%paramatic_entity を挿入する必要があります。たとえば、DTD に次のコードが含まれているとします。

```
<!ELEMENT  header EMPTY >
<!ATTLIST  header
           name    CDATA   #REQUIRED
           title   CDATA
           home    (Y|N)   'N'
           portal  (Y|N)   'N'
           logout  (Y|N)   'N' >
```

このコードは、次のとおり変更する必要があります。

```
<!ELEMENT  header EMPTY >
<!ATTLIST  header
           name   CDATA #REQUIRED
           title  CDATA #REQUIRED <- can replaced by #FIXED, #IMPLIED, or "title1"
           home    (Y|N)   'N'
           portal  (Y|N)   'N'
           logout  (Y|N)   'N' >
```

XML 文書内の外部実体参照に使用する標準の拡張子

質問

XML 文書内で参照されている外部エンティティに使用する標準の拡張子（.xml または .txt 以外）はありますか？これらの外部エンティティは完全な XML ファイルではなく、その一部のみで、<![CDATA[で始まります。ほとんどの XML ファイルには HTML または Java スクリプト・コードが含まれていますが、プレーン・テキストのみが含まれる場合もあります。次に、XML 文書 B.xml で参照されている外部エンティティ A.txt の例を示します。

A.txt:

```
<![CDATA[<!-- This is just an html comment -->]]>
```

B.xml:

```
 <?xml version="1.0"?>
 <!DOCTYPE B[
 <!ENTITY htmlComment SYSTEM "A.txt">
]>

<B>
  &htmlComment;
</B>
```

現在、このようなすべてのエンティティの拡張子として .txt を使用していますが、それを変更する必要があります。変更しない場合、翻訳チームは、翻訳が不要なこれらのファイルを翻訳する必要があるとみなします。この場合に使用できる標準の拡張子はありますか？

回答

DTD 内の DTD 構文を赤字（太字）でマークアップしました。外部エンティティのファイル拡張子は重要でないため、任意の有効な拡張子（拡張子なしも含む）に変更できます。

DOM API および SAX API

DOM API の使用

質問
パーサーを使用して特定のタグの要素数を取得する方法を教えてください。

回答
getElementsByTagName() メソッドを使用します。このメソッドは、指定したタグ名を持つすべての子要素の NodeList を戻します。その NodeList 内の要素数を調べて、特定のタグの要素数を判断できます。

DOM パーサーの動作

質問
XML DOM Parser はどのように動作しますか？

回答
XML DOM Parser は XML 形式のドキュメントを受け入れ、構造に基づいて DOM ツリーをメモリー内に構築します。ドキュメントが整形式であるかどうかを確認し、オプションで DTD に準拠しているかどうかを確認します。DOM レベル 1 および 2 をサポートするメソッドも提供されています。

後で設定する値を使用したノードの作成

質問
値を後で設定できるノードを作成する方法を教えてください。

回答
ノード型を説明する表に関する DOM 仕様を確認すると、要素ノードを作成する場合に nodeValue が NULL になるため、ノードを設定できないことがわかります。ただし、テキスト・ノードを作成し、要素ノードへ追加できます。テキスト・ノードに値を格納できます。

XML ツリーの全検索

質問
XML ツリーの全検索方法を教えてください。

回答
ツリーは、DOM API を使用して全検索できます。または、XPath 構文を取る selectNodes() メソッドを使用して、XML 文書内を操作できます。selectNodes() は、oracle.xml.parser.v2.XMLNode の一部です。

XML ファイルからの要素の取得

質問
XML ファイルから要素を取得する方法を教えてください。

回答
DOM を使用している場合、getElementsByTagName() メソッドを使用してドキュメント内のすべての要素を取得できます。

DTD による DOM ツリーの検証

質問
XML 文書に DTD を追加すると、この DTD は DOM ツリーを検証しますか？

回答
検証しません。DOM API を使用して DOM ツリーを作成している間は検証を行うことができません。このため、文書に DTD を設定しても、作成した DOM ツリーの検証には有効ではありません。XML ファイルを検証する唯一の方法は、DOMParser または SAXParser を使用して XML 文書を解析することです。

最初の子ノードの要素の値

質問

DOM ツリーを介さずに、要素の最初の子ノードの値を効率的に取得する方法を教えてください。

回答

ツリー全体が必要ない場合、SAX インタフェースを使用して必要なデータを戻します。SAX インタフェースはイベント駆動であるため、ドキュメント全体を解析する必要はありません。

DOCTYPE ノードの作成

質問

DOCTYPE ノードの作成方法を教えてください。

回答

DOCTYPE ノードは現在、parseDTD メソッドを使用してのみ作成できます。たとえば、emp.dtd には次の DTD があります。

```
<!ELEMENT employee (Name, Dept, Title)>
 <!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
 <!ELEMENT Title (#PCDATA)>
```

次のコードを使用して DOCTYPE ノードを作成できます。

```
parser.parseDTD(new FileInputStream(emp.dtd), "employee");
dtd = parser.getDocType();
```

XMLNode.selectNodes() メソッド

質問

selectNodes() メソッドは XMLNode クラスでどのように使用すればいいですか？

回答

selectNodes() メソッドは、XMLElement ノードおよび XMLDocument ノードに使用します。このメソッドは、XSL が許可する選択パターンに基づいてツリー / サブツリーから内容を取得するために使用されます。selectNodes のオプションの 2 つ目のパラメータは、名前空間の接頭辞を解決するために使用されます（接頭辞付きの展開された名前空間の URL を戻します）。XMLElement は、2 つ目のパラメータとして送信されるように NSResolver を実装します。XMLElement は、入力ドキュメントに基づいて接頭辞を解決します。名前空間の定義をオーバーライドする必要がある場合、NSResolver インタフェースを実装できます。次に、selectNodes を使用したコード例を示します。

```
public class SelectNodesTest  {
public static void main(String[] args) throws Exception {
String pattern = "/family/member/text()";
String file    = args[0];

if (args.length == 2)
  pattern = args[1];

DOMParser dp = new DOMParser();

dp.parse(createURL(file));  // Include createURL from DOMSample
XMLDocument xd = dp.getDocument();
XMLElement e = (XMLElement) xd.getDocumentElement();
NodeList nl = e.selectNodes(pattern, e);
for (int i = 0; i < nl.getLength(); i++) {
   System.out.println(nl.item(i).getNodeValue());
    }
  }
}

> java SelectNodesTest family.xml
Sarah
Bob
Joanne
Jim

> java SelectNodesTest family.xml //member/@memberid
m1
m2
m3
m4
```

SAX API を使用したデータ値の取得

質問

XML 文書の解析に SAX を使用しています。データ値の取得方法を教えてください。

回答

SAX による解析の間、要素の値は、startElement イベントの後から対応する endElement イベントがコールされるまでに通知された文字の連結になります。

SAXSample.java

質問

SAXSample のプログラム内に、setDocumentLocator およびその他のメソッドを明示的にコールする行がありませんが、これらのメソッドが実行されています。これらのメソッドはいつ、どこからコールされるのですか？

回答

SAX は、イベントベースの XML 解析用の標準インタフェースです。このパーサーは、解析イベントを setDocumentLocator() や startDocument() などのコールバックのメソッドを介して直接通知します。アプリケーション（この場合は SAXSample）は、異なるイベントを処理するハンドラを実装します。http://www.megginson.com/SAX/index.html では、イベント駆動の API である SAX の基本的な説明を参照できます。

DOMParser によるパーサー・インタフェースの実装

質問

XML パーサーの DOMParser は、org.xml.sax.Parser インタフェースを実装しますか？ドキュメントでは、DOMParser は XMLConstants を実装し、API にはこのクラスが含まれないと記載されています。

回答

SAX を処理し、org.xml.sax.Parser インタフェースを実装させるためには、oracle.xml.parser.v2.SAXParser が必要です。

DOM を使用した新しいドキュメント・タイプ・ノードの作成

質問

XML ファイルを作成しようとしています。NodeFactory を使用してドキュメントを構築し（createDocument()）、その後 setStandalone("no") および setVersion("1.0") を指定します。appendChild(new XMLNode("test", Node.DOCUMENT_TYPE_NODE)) を使用して DOCTYPE ノードを追加しようとすると、ClassCastException が発生します。このタイプのノードを追加するメカニズムを教えてください。NodeFactory には、DOCTYPE ノードを作成するメカニズムがないことはわかっています。

回答

DOM API を介して新しい DOCUMENT_TYPE_NODE オブジェクトを作成するメカニズムはありません。DTD オブジェクトを取得する唯一の方法は、DTD ファイルまたは XML ファイルを DOMParser を使用して解析し、getDocType() メソッドを使用することです。

新しい XMLNode("test",Node.DOCUMENT_TYPE_NODE) は、DTD オブジェクトを作成しないことに注意してください。これは、タイプを DOCUMENT_TYPE_NODE に設定して XMLNode オブジェクトを作成します。この方法は、実際には使用しないでください。appendChild が（タイプに基づいて）DTD オブジェクトを予期するため、ClassCastException が発生します。

また、DOM API を使用して DOM ツリーを作成している間は、検証を行うことができません。このため、ドキュメントに DTD を設定しても、作成した DOM ツリーの検証には有効ではありません。XML ファイルを検証する唯一の方法は、DOMParser または SAXParser を使用して XML 文書を解析することです。

特定のタグの最初の子ノード値の問合せ

質問

XML Parser for Java を使用しています。「微分積分」、「数学」、「Jim Green」、「Jack」、「Mary」および「Paul」を含む XML 文書があるとします。特定のタグの最初の子ノードの値を取得する必要があります。これを効率的に行うメソッドが見つかりません。最も近いメソッドは、下位ツリー全体を全検索する getElementsByTag("Name") です。

回答

ツリー全体が必要ない場合、最適な方法は、SAX インタフェースを使用して必要なデータを戻すことです。SAX インタフェースはイベント駆動であるため、ドキュメント全体を解析する必要はありません。

変数のデータからの XML 文書の生成

質問

単純な変数に含まれた情報から XML 文書を生成する例がありますか？クライアントが Java フォームに記入し、特定のデータを含む XML 文書を取得する必要がある場合のような例です。

回答

ご質問の内容を 2 通りに解釈して回答を示します。Java で指定した 2 つの変数があるとします。

```
String firstname = "Gianfranco";
String lastname = "Pietraforte";
```

これを XML 文書にする 2 つの基本的な方法は、次のとおりです。

1. XML 文書を文字列にして、解析します。

```
String xml = "<person><first>"+firstname+"</first>"+
                  "<last>"+lastname+"</last></person";
 DOMParser d = new DOMParser();
 d.parse( new StringReader(xml));
 Document xmldoc = d.getDocument();
```

2. DOM API を使用して文書を構築し、それぞれを統合します。

```
Document xmldoc = new XMLDocument();
Element e1 = xmldoc.createElement("person");
xmldoc.appendChild(e1);
Element e2 = xmldoc.createElement("first");
e1.appendChild(e2);
Text t = xmldoc.createText(firstname);
e2.appendChild(t);
// and so on
```

要素タグへのデータの出力 : DOM API

質問

DOM API を使用して次の要素を Java で出力する方法を教えてください。

```
<name>macy</name>
```

「macy」を出力する必要がありますが、使用するクラスおよびメソッドがわかりません。「name」は正常にコンソールに出力できました。

回答

DOM の場合、<name>macy</name> は、値が「macy」である子ノード（テキスト・ノード）を持つ「name」という名前の要素であることを理解してください。

したがって、次のコードを実行できます。

```
String value = myElement.getFirstChild().getNodeValue();
```

ハッシュ表値のペアからの XML ファイルの構築

質問

キー値のペアのハッシュ表がありますが、DOM API を使用してこの表から XML ファイルを構築する方法を教えてください。hashtable{key = value,name = george,zip = 20000} を設定しています。これを構築する方法を教えてください。

```
<key>value</key><name>george</name><zip>20000</zip>
```

これを自動的に行うユーティリティがありますか？

回答

1. ハッシュ表からキーの一覧を取得します。
2. enum.hasMoreElements() 中にループします。
3. 一覧内の各キーに対して、DOM 文書に createElement() を使用して、そのキーにハッシュ表エントリの *value* の値を指定している子テキスト・ノードを持つキー名によって、要素を作成します。

XML Parser for Java: Node.appendChild() の wrong_document_err

質問

XML パーサーの実装について質問があります。次の使用例について考えてみます。

```
Document doc1 = new XMLDocument();
Element element1 = doc1.creatElement("foo");
Document doc2 = new XMLDocument();
Element element2 = doc2.createElement("bar");
element1.appendChild(element2);
```

appendChild() ルーチンをコールすると、WRONG_DOCUMENT_ERR の DOMException が発生するのは正常なのでしょうか？XML パーサーに付属する XSLSample.java を見て疑問に思いました。ご意見をお聞かせください。

回答

element1 の所有者ドキュメントは doc1 であり、element2 の所有者ドキュメントは doc2 であるため、このエラーは必ず発生します。appendChild() は、単一ツリー内でのみ動作しますが、例では 2 つの異なるツリーを操作しています。

質問

XML パーサーに付属する XSLSample.java では、次のように記述されています。

```
DocumentFragment result = processor.processXSL(xsl, xml);
// create an output document to hold the result
  out = new XMLDocument();
// create a dummy document element for the output document
  Element root = out.createElement("root");
  out.appendChild(root);
// append the transformed tree to the dummy document element
   root.appendChild(result);
```

ノード・ルートおよび結果が異なる XML 文書から作成されました。これでは、結果を追加するときに WRONG_DOCUMENT_ERR が発生しませんか？

回答

この例では、ルート・ノードを持たないドキュメント・フラグメントを使用しています。したがって、XML 文書は 2 つではありません。

質問

ドキュメント・フラグメントをノードに追加するとき、ドキュメント・フラグメントの子ノードのみが挿入され、ドキュメント・フラグメント自体は挿入されません。パーサーは、これらの子ノードの所有者ドキュメントを確認しますか？

コメント

ドキュメント・フラグメントは、定義上、単なるノードのリストであることが多いため、ルート・ノードにバインドしないでください。ルート・ノード（存在する場合）は、単一の子として考える必要があります。たとえば、請求書自体ではなく、請求書ドキュメントのすべての行を取り、ProviderOrder ドキュメントに追加するとします。ドキュメント・フラグメントを他のドキュメントに追加するために、XSLT プロセッサが行うように、ルートなしでドキュメント・フラグメントを作成するにはどうすればいいでしょうか？

ノードの作成 : ノード値設定時の DOMException

質問

次のエラーが戻ります。

```
oracle.xml.parser.XMLDOMException: Node cannot be modified while trying to set the
value of a newly created node as below:
  String eName="Mynode";
  XMLNode aNode = new XMLNode(eName, Node.ELEMENT_NODE);
  aNode.setNodeValue(eValue);
```

後で値を設定できるノードを作成する方法を教えてください。

回答

ノード・タイプに関する DOM の注記を参照してください。要素ノードを作成している場合、nodeValue が NULL であり、設定できないことがわかります。

SAX 使用時にパーサーに強制的に空白を削除させないようにする方法

質問

SAX（Oracle XML Parser リリース 2.0.2.9.0）を使用して添付ファイルを読み取ると、文字データが空白で始まる場合、characters( ) メソッドは空白に続く文字を削除します。

これはエラーですか？それとも、パーサーに強制的にそれらの文字を削除させないようにすることができますか？

回答

XMLParser.setPreserveWhitespace(true) を使用して、パーサーに強制的に空白を削除させないようにします。

検証

DTD: DOCTYPE および妥当性を検証するパーサーの理解

質問

次の DTD への参照を含む XML 文字列があります。この DTD は、プログラムを起動するディレクトリ内に物理的に格納されています。妥当性を検証するパーサーによって、このファイルが見つからないと通知されます。

```
<!DOCTYPE xyz SYSTEM "xyz.dtd" >
```

DTD をディスク上に置くときの規則を教えてください。DOCTYPE 属性について説明してください。

回答

解析しているのは InputStream ですか、それとも URL ですか？InputStream を解析している場合、パーサーはその InputStream の送信元を識別しないため、現在のファイルと同じディレクトリにある DTD を検索できません。これを解決するには、setBaseURL() を設定してパーサーに URL 情報の一部を与え、DTD を取得するときに残りの情報を導出するようにします。

複数スレッドでの XSLProcessor/XSLStylesheet の使用

質問

複数スレッドは、単一の XSLProcessor/XSLStylesheet インスタンスを使用して同時変換を実行できますか？

回答

XML ファイルごとに 1 つの XSLProcessor/XSLStylesheet のみを使用して複数ファイルを処理している場合、複数スレッドを使用して同時に処理できます。bin ディレクトリにある readme.html ファイルに、複数スレッド処理用のスレッド・パラメータを持つ ORAXSL についての説明があります。

複数スレッドでドキュメントのクローンを使用する場合の安全性

質問

ドキュメントのクローンを複数スレッドで使用することは安全ですか？public void setParam(String ,String) によって発生する oracle.xml.parser.v2.XSLStylesheet クラスの XSLException メソッドはサポートされていますか？サポートされていない場合、実行時に XSLT プロセッサにパラメータを渡す別の方法はありますか？

回答

コンストラクタが設定したグローバル領域を他のスレッドにコピーする場合は、安全です。

そのメソッドは、Oracle XML Parser V2.0.2.5 以上でサポートされています。

コメント

XSLException メソッドはドキュメント内にありますが、XSLStylesheet クラス（Windows 用 zip 版）には実装されていません。まず、最新の ZIP ファイルをダウンロードしてください。

```
public static void serve(Document template, Document data,Element
userdata,PrintWriter out)
{
 XMLDocument clone = (XMLDocument)data.cloneNode(true);
    clone.getDocumentElement().appendChild(userdata.cloneNode(true));
    serve(template, clone, out);
}
```

キャラクタ・セット

XML パーサーでの ISO 8859-1 エンコーディング

質問

エンコーディングが ISO 8859-1 の XML 文書があります。これらの文書を XML パーサーの SAX API によって解析しようとしています。文字（char[], int, int）では、内容も ISO 8859-1（Latin1）で出力する必要があります。

System.out.println() を使用すると、正しく出力できません。ドイツ語のウムラウトが、出力ストリームでは「?」になります。内部では、「_」、「÷」、「'」、「ý」、「_」、「›」、「?」がそれぞれ「65508」、「65526」、「65532」、「65476」、「65494」、「65500」、「65503」として格納されます。Latin1 で出力する方法を教えてください。使用しているホスト・システムは、SPARC Solaris 2.6 です。

回答

System.out.println() を使用することはできません。OutputStreamWriter など、エンコーディングを識別する出力ストリームを使用する必要があります。

`OutputStreamWriter` を構築して、次のとおり write(char[], int, int) メソッドを使用できます。

```
print.Ex:OutputStreamWriter out = new OutputStreamWriter(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

UTF-8 エンコーディングで NCLOB に格納された XML の解析

質問

UTF-8 エンコーディングを使用して NCLOB 列に格納された XML は解析できません。次のコンポーネントを実行しています。

- Windows NT 4.0 Server Pack
- Oracle8*i* リリース 8.1.5
- JDeveloper 3.0
- JDK 1.1.8
- Oracle XML Parser（2.0.2.5?）

データベースにロードした次の XML の例には、2 つの UTF-8 マルチバイト・キャラクタが含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<G>
<A>GÂ,otingen, BrÃ ck_W</A>
</G>
```

G(0xc2, 0x82)otingen, Br(0xc3, 0xbc)ck_W

これらのマルチバイト・キャラクタは両方とも有効な UTF-8 エンコーディングであり、ISO 8859-1 では次のとおり定義されます。

```
0xC2 LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0xFC  LATIN SMALL LETTER U WITH DIAERESIS
```

デフォルト接続オブジェクトを使用してデータベースに接続し、SELECT 問合せを実行し、OracleResultSet を取得し、getCLOB メソッドをコールして CLOB オブジェクトに関する getAsciiStream() をコールする Java ストアド・ファンクションを作成しました。その後、このファンクションは、次のコードのフラグメントを実行して XML を DOM オブジェクトに変換します。

```
DOMParser parser = new DOMParser();
parser.setPreserveWhitespace(true);
parser.parse(istr);
// istr getAsciiStreamXMLDocument xmldoc = parser.getDocument();
```

ストアド・ファンクションが次のタスクを行う前に、このコードによって前述の XML に「無効な UTF-8 エンコーディング」が含まれるという例外が発生します。

- 最初のマルチバイト・キャラクタ（Oxc2,Ox82）を XML から削除すると、正常に解析されます。
- この文字を削除しないで、Oracle JDBC Thin ドライバ（ここではストアド・ファンクションとして RDBMS 内で実行していないことに注意してください）を介して接続すると、XML は正常に解析され、XML 文書であらゆる処理を行うことができます。

サンプル XML を、JDBC Thin ドライバを使用してデータベースにロードしました。2 つのデータベース構成 WE8ISO8859P1/WE8ISO8859P1 および WE8ISO8859P1/UTF8 を試してみましたが、どちらでも同じ問題が発生しました。

回答

確かに文字（0xc2,0x82）は有効な UTF-8 です。getAsciiStream() がコールされたときに文字に異常が発生したのではないかと思います。getAsciiStream() のかわりに、getUnicodeStream() および getBinaryStream() を使用してみてください。

正常に実行できない場合は、手順 parser.parse(istr) で文字がパーサーに送信される前に、文字が異常でないことを確認するために出力してみてください。

XML 内の NLS サポート

質問

データベースの nvarchar2 フィールド内に日本語のデータを格納しています。PL/SQL Web ツールキットを利用する動的 SQL プロシージャによって、OAS およびブラウザを介してデータにアクセスできます。このプロシージャは XML パーサーを使用して、ブラウザに戻す前に、結果セットを XML に適切にフォーマットします。

問題は、戻された日本語データがさかさまの疑問符としてブラウザに表示されることです。このデータを正確に戻して漢字として表示する方法を教えてください。

回答

Java および XML のデフォルト・キャラクタ・セットは UTF-8 ですが、UTF-8 OS、データベースでの UTF-8 の使用、または UTF-8 を使用した Web ページの作成については聞いたことがありません。つまり、文字コード変換の問題があります。漢字を正常に表示することはできます。XML Parser for PL/SQL および XML Parser for Java は、両方とも日本語で動作します。ただし、ご質問の件に対して簡単には説明できません。

XML Parser for Java での UTF-16 エンコーディング

質問

次のような XML 文書があります。

Documento de Prueba de gestin de contenidos. Roberto P_rez Lita

次の方法でこの文書を解析します。

```
DOMParser parser=new DOMParser();
parser.setPreserveWhitespace(true);
parser.setErrorStream(System.err);
parser.setValidationMode(false);
parser.showWarnings(true);
parser.parse ( new FileInputStream(new File("PruebaA3Ingles.xml")));
```

次のようなエラーが戻ります。

```
XML-00231 : エンコーディング 'UTF-16' は現在サポートされません。
```

XML Parser for Java リリース 2.0.2.5 を使用しています。マニュアルには、このバージョンのパーサーは UTF-16 エンコーディングをサポートしていると記載されています。スペイン語を含む文書を解析する方法はありますか？

回答

オラクル社では、XML Parser for Java の新しいリリースをアップロードしたばかりです。この最新のリリースは UTF-16 をサポートしますが、他のユーティリティでの UTF-16 エンコーディングの使用には問題があります。

アクセント付き文字を読み込む方法

質問

アクセント付き文字を XML 文書内に格納する必要があります。é などのアクセント付き文字を XML ファイルに手動で追加し、Oracle XML Parser for Java を使用して XML 文書を解析しようとすると、次の例外が発生します。

```
'Invalid UTF-8 encoding'
```

次に、XML ヘッダー内のエンコーディング宣言を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
```

また、UTF-16 をデフォルトのエンコーディングに指定すると、Oracle XML Parser for Java は、UTF-16 は現在サポートされていないというメッセージを表示します。Java プログラム内から、Java 文字列オブジェクトを次のとおり定義するとします。

```
String name = "éééé";
```

また、プログラムで XML 文書を生成し、それをファイルに保存すると、é 文字は正しくファイルに書き出されます。アクセント付き文字で構成される文字データを正常に読み取る方法を教えてください。アクセント付き文字は、一度 XML 文書内で 16 進または 10 進フォーマットで表すと読み取ることができることは知っています。次に例を示します。

```
&#xe9;
```

ただし、この方法は実行したくありません。

回答

XML ファイルの作成時に使用したキャラクタ・セットに基づいてエンコーディングを設定する必要があります。この問題は、エンコーディングを ISO 8859-1（西欧語 ASCII）に設定して解決しました。ただし、使用しているツールまたはオペレーティング・システム（あるいはその両方）によっては、異なるエンコーディングを使用する必要がある場合があります。

エンコーディングを明示的に UTF-8 に設定した場合（またはエンコーディングを指定しない場合）、XML パーサーはアクセント付き文字（127 より大きい ASCII 値の文字）を UTF-8 マルチバイト文字列の最初のバイトとして解析します。後続のバイトが有効な UTF-8 文字列を構成していない場合、このエラーが発生します。

回答

このエラーは、エディタがファイルを UTF-8 エンコーディングを使用して保存していないことを意味します。たとえば、エディタは ISO 8859-1 エンコーディングを使用してファイルを保存している場合があります。エンコーディングは Unicode のキャラクタ番号表現をディスクに書き込むために使用される特定のスキーマであることを理解しておいてください。文字列を次のように文書の最上部に追加するとします。

```
<?xml version="1.0" encoding="UTF-8"?>
```

この場合、エディタはファイルを表すバイトを UTF-8 エンコーディングを使用してディスクに書き出しません。メモ帳は UTF-8 を使用すると思われるため、それを試してみてください。

子としての XML 文書の追加

他の要素の子としての XML 文書の追加

質問

XML 文書を既存の要素の子として追加しています。次に例を示します。

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class ggg {public static void main (String [] args) throws Exception
 {
new ggg().doWork();;
public void doWork() throws Exception {XMLDocument doc1 = new XMLDocument();
Element root1=doc1.createElement("root1");
XMLDocument doc2= new XMLDocument();Element root2=doc2.createElement("root2");
root1.appendChild(root2);
doc1.print(System.out);};};
```

次のように通知されます。

```
D:\Temp\Oracle\sample>c:\jdk1.2.2\bin\javac -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;.
ggg.javaD:\Temp\Oracle\sample>c:\jdk1.2.2\bin\java -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;. gggException in thread "main"
java.lang.NullPointerException        at
oracle.xml.parser.v2.XMLDOMException.(XMLDOMException.java:67)        at
oracle.xml.parser.v2.XMLNode.checkDocument(XMLNode.java:919)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java, Compiled Code)        at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java:494)        at
ggg.doWork(ggg.java:20)        at ggg.main(ggg.java:12)
```

回答

1. 次のコードは正常に実行できます。

```
DocumentFragment rootNode = new XMLDocumentFragment(); DOMParser d  = new
DOMParser(); d.parse("http://.../stuff.xml");
Document doc = d.getDocument();
Element e = doc.getDocumentElement();
// Important to remove it from the first doc
// before adding it to the other doc. doc.removeChild(e);
rootNode.appendChild(e);
```

文書が保持できるルート・ノードは 1 つのみであるため、これを行うには DocumentFragment クラスを使用する必要があります。

2. すべてのノードには、ノードが作成された文書への参照が含まれているため、他の文書で作成されたノードの追加には特に問題はありません。文書フラグメントによってこれは解決されるため、1 つのルートに関する問題のみではないと思われます。com.w3c.dom.Document を org.w3c.dom.DocumentFragment に変換する簡単な方法はあるのでしょうか?

XML 文書の子としての XML 文書フラグメントの追加

質問

次のようなコードのフラグメントがあります。

```
XSLStylesheet XSLProcessorStylesheet = new XSLStylesheet(XSLProcessorDoc,
XSLProcessorURL);
XSLStylesheet XSLRendererStylesheet = new XSLStylesheet(XSLRendererDoc,
XSLRendererURL);
XSLProcessor processor = new XSLProcessor();
// configure the processorprocessor.showWarnings(true);
processor.setErrorStream(System.err);
XMLDocumentFragment processedXML = processor.processXSL(XSLProcessorStylesheet,
XMLInputDoc);
XMLDocumentFragment renderedXML = processor.processXSL(XSLRendererStylesheet,
processedXML);
Document resultXML = new XMLDocument();
resultXML.appendChild(renderedXML);
```

最後の行で、main スレッド oracle.xml.parser.v2 に例外が発生します。

```
XMLDOMException: Node of this type cannot be added.
```

結果文書フラグメントが、整形式の XML 文書であることがわかっていても（およびルート要素を 1 つのみ持つことがわかっていても）、毎回ルート要素を作成する必要がありますか？

回答

文書フラグメントは 2 つ以上のルート要素（より適切な用語がないため）を含むことができるため、この問題が発生します。これに対処するには、Node クラスのメソッドを使用して文書フラグメントから 1 つのルート要素を取得し、Element にキャストします。

XML パーサーのアンインストール

データベースからの XML パーサーの削除

質問

あるバージョンの XML パーサーをアンインストールし、新しいバージョンをインストールする方法を教えてください。dropjava などのコマンドでは、スキーマにロードされている他のパッケージが残ります。以前のバージョンを完全に削除してから新しいバージョンを正しい方法でインストールする必要があります。

回答

USER_OBJECTS 表に次の SQL 文を記述する必要があります。

```
SELECT 'drop java class '''&#0124; &#0124;       dbms_java.longname(object_
name)&#0124; &#0124;''';
from user_objects where
OBJECT_TYPE = 'JAVA CLASS'and DBMS_JAVA.LONGNAME(OBJECT_NAME)     LIKE
'oracle/xml/parser/%'
```

これによって、一連の DROP JAVA CLASS コマンドが起動されます。これらのコマンドは、SQL*Plus の SPOOL コマンドを使用してファイルに獲得できます。

その後、そのスプール・ファイルを SQL スクリプトとして実行すると、すべての適切なクラスが削除されます。

XML Parser for Java: インストール

XML パーサーがインストールできない場合

質問

XML パーサーをインストールしようとすると次のエラー・メッセージが戻されます。

```
loadjava -user username/manager -r -v xmlparserv2.jar
Error:
Exception in thread "main" java.lang.NoClassDefFounderr:
oracle.aurora.server.tools at oracle/jdbc/driver/OracleDriver.. etc..
```

回答

これは、CLASSPATH 内に JDBC classes111.zip が検出されなかったためのエラーです。loadjava ユーティリティは、データベースに接続し、JDBC ドライバを使用してクラスをロードします。

「loadjava」を確認すると、classes111.zip へのパスは次のとおりでした。

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
```

リリース 8.1.6 では、classes111.zip は次のディレクトリにあります。

```
<ORACLE_HOME>/jdbc/admin
```

XML パーサーに関連する一般的な質問

XML パーサーの動作

質問

XML パーサーの役割は何ですか？

回答

XML パーサーは、XML 文書を受け入れて、文書の要素と属性、およびイベント API（SAX）に対してアクセスまたは変更を行うツリーベース API（DOM）を戻します。イベント API は、登録するリスナーを提供し、特定の要素または属性、およびその他の文書イベントを通知します。

XML ファイルの HTML ファイルへの変換

質問

XML ファイルを HTML ファイルに変換する方法を教えてください。

回答

XML を HTML にレンダリングするための XSL スタイルシートを作成する必要があります。任意の形式で HTML ドキュメントを作成し、ダミー・データを移入します。スタイルシートを記述した XML 文書のデータを HTML に移入する XSLT コマンドによってこのデータを置き換えます。

XML パーサーによる XML Schema に対する検証

質問

XML パーサーは XML Schema に対する検証を行いますか？

回答

XML パーサー は、検証モードおよび非検証モードの両方をサポートします。XML Schema は、W3C の XML Schema 委員会で作成中です。Oracle は XML Schema をサポートしています。現在、XML Parser for Java は、DTD および XML Schema に対する非検証モード、DTD 検証モード、部分検証モードおよびスキーマ検証モードでの検証、非検証および部分検証をサポートしています。

XML 文書へのバイナリ・データの挿入

質問

XML 文書にバイナリ・データを挿入する方法を教えてください。

回答

文書に直接バイナリ・データを挿入する方法はありません。ただし、これに対処するには 2 つの方法があります。

- バイナリ・データは、異なるファイルに常駐する未解析の外部エンティティとして参照できます。
- バイナリ・データは、非エンコーディング（バイナリ・データを ASCII データに変換）をして CDATA セクションに含めることができます。エンコーディング方法には、CDATA セクションに正当な文字のみを作成するように確認する必要があるという制限があります。

XML Schema の概要

質問

XML Schema とは何ですか？

回答

XML Schema とは、データ型の概念を XML 文書に適用するために W3C が策定している XML 標準であり、DTD 構文を XML に基づく構文に置き換えるものです。詳細は、http://www.w3.org/TR/xmlschema-1/ および http://www.w3.org/TR/xmlschema-2/ を参照してください。XML Schema は、Oracle 以上でサポートされています。

XML/SQL 標準の定義へのオラクル社の参加

質問
オラクル社は XML/XSL 標準の定義に参加していますか？

回答
オラクル社は、XML/XSL に関連する XML Schema、XML Query、XSL、XLink/XPointer、XML Infoset、DOM および XML Core の W3C ワーキング・グループに積極的に参加しています。

XDK のバージョン番号

質問
ダウンロードした XDK ツールキットのバージョン番号はどのようにしてわかりますか？

回答
アーカイブ内、および Release Notes ページにリンクされた readme.html を参照すると、完全なバージョン番号を確認できます。

XML 文書への <、>、>= および <= の挿入

質問
XML 文書に >、<、>= および <= を挿入する方法を教えてください。

回答
< にはエンティティ <、および > にはエンティティ > を使用する必要があります。

XML 名前空間および XML Schema のサポート

質問
XML 名前空間および XML Schema はサポートされていますか？

回答
現在の XML パーサーは XML 名前空間をサポートします。XML Schema は、Oracle9*i* 以上でサポートされます。

XML Parser for Java バージョン 2 以上での JDK 1.1.x の使用

質問

XML Parser for Java バージョン 2 以上で JDK 1.1.x を使用できますか？

回答

XML Parser for Java のバージョン 2 は、Java2 とは関連がありません。これは、XML Parser for Java バージョン 1 とは下位互換性がなく、XSLT をサポートしていることを示します。XML Parser for Java バージョン 2 以上では JDK 1.1.x は正常に動作します。

ページでの結果のソート

質問

100 のレコードを、一度に 10 ずつ表示するとします。各列名にリンクを作成し、その列に基づいて、クリックによってページごとにデータをソートする必要があります。どうすればいいですか？

回答

目的によって方法は異なります。Internet Explorer 5.0 専用にページを作成して XML データを受け取る場合、Microsoft 社の XSL を使用してページ内でデータをソートできます。その他のブラウザ用にページを作成し、そのブラウザがデータを HTML で受け取る場合、XSQL スクリプトでソート・パラメータを設定し、このパラメータを ORDER BY 句で使用する必要があります。skip-rows パラメータとともに渡します。

XML Parser for Java の実行に必要な Oracle

質問

XML Parser for Java の実行に Oracle は必要ですか？

回答

XML Parser for Java は、サポートされている Java VM のすべてのバージョンで使用できます。Oracle と異なる点は、データベースにロードして、内部 Java VM である Oracle JVM を使用できることのみです。その他のデータベースのバージョンまたはサーバーでは、外部 Java VM 内で実行し、必要に応じて JDBC を介してデータベースに接続します。

XML ファイルでのエンコーディングの動的な設定

質問

XML 文書でエンコーディングを動的に設定できますか？

回答

できません。仕様のとおりに、文書内で適切なエンコーディング宣言を含める必要があります。setEncoding() を使用して、文書の入力にエンコーディングを設定することはできません。setEncoding() および oracle.xml.parser.v2.XMLDocument を使用して、出力に正しいエンコーディングを設定します。

文字列の解析

質問

文字列を解析する方法を教えてください。

回答

文字列に含まれている XML 文書を直接解析する方法は現在はありません。文字列は、解析する前に InputStream または InputSource に変換する必要があります。簡単な方法は、文字列内のバイトを使用して ByteArrayInputStream を生成することです。

XML 文書の表示

質問

XML 文書を表示する方法を教えてください。

回答

Internet Explorer 5 ブラウザを使用している場合、直接 XML 文書を表示できます。それ以外のブラウザの場合、XML パーサーの XSLT プロセッサを使用すると、XSL スタイルシートを使用して HTML ドキュメントを作成できます。Oracle XML Transviewer Beans を使用しても、XML 文書を表示できます。

System.out.primtln() および特殊文字

質問

System.out.println() を特殊文字のエンコーディングに使用すると問題があります。

回答

System.out.println() は使用できません。OutputStreamWriter など、エンコーディングを識別する出力ストリームを使用する必要があります。OutputStreamWrite を構築して、write(char[], int, int) メソッドを使用して出力します。

```
/* Example */
OutputStreamWriter out = new OutputStreamWriter
(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

文字データからのアンパサンド（&）の取得

質問

文字データからアンパサンド（&）を取得する方法を教えてください。

回答

XML データでは、アンパサンドをそのまま使用できません。かわりに、エンティティ & を使用する必要があります。このエンティティは、XML 標準で定義されています。

タグで特殊文字を使用する方法

質問

XML に < 会社名 > というタグがあります。

「A&B」を使用しようとすると、XML パーサーは文字が無効であるというエラーを通知します。会社名タグを解析する場合に特殊文字を使用する方法を教えてください。Oracle XML Parser for C を使用しています。

回答

1. リテラルを次のとおり表す必要があります。

   & は & として表します。

   < は < として表します。

2. 特殊文字を XML 名の一部として使用する必要がある場合があります。たとえば、<A&B>abc</A&B> のような例です。

   この場合、名前エンティティを使用しても問題は解決されません。XML 1.0 仕様（http://www.w3.org/TR/2000/REC-xml-20001006）では、NameChar および Name は次のとおり定義されます。

   [4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar |Extender

   [5] Name ::= (Letter | '_' | ':') (NameChar)*

   「&」、「$」、「#」などの特殊文字は、NameChar として使用できません。そのため、XML 文書を最初から作成する場合は、有効な NameChar のみを使用すると、解決策になります。たとえば、<A_B>、<AB>、<A_AND_B> などです。

   これらの文字は読取り可能です。

   ただし、データベース表などの外部データ・ソースから XML を生成する場合は、これが問題となります。XML 1.0 は、この問題に対応していません。

   Oracle では、新しい型である XMLType が SQL の名前を XML の名前にマップする機能を提供し、この問題を解決します。XMLType は、この問題をアプリケーション・レベルで解決します。SQL から XML に名前をマッピングする機能によって、_XHHHH_ という形式（HHHH は無効な文字の Unicode 値）の無効な XML NameChar がエスケープされます。たとえば、表名「V$SESSION」は XML の名前「V_X0024_SESSION」にマップされます。

   無効な文字のエスケープは、名前を他の場所に再ロードできるようにシリアル化する方法を提供する有効な手段です。

文字列データ型からの XML の解析

質問

文字列データ型から XML を解析する方法を教えてください。

回答

次の例を参照してください。

```
/* xmlDoc is a String of xml */
byte aByteArr [] = xmlDoc.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream (aByteArr, 0, aByteArr.length);
domParser.parse(bais);
```

XML 文書から文字列へのデータの取得

質問

XML 文書から文字列にデータを取得する方法を教えてください。

回答

次に例を示します。

```
XMLDocument Your Document;
/* Parse and Make Mods */
:
StringWriter sw = new StringWriter();
PrintWriter  pw = new PrintWriter(sw);
YourDocument.print(pw);
String YourDocInString = sw.toString();
```

エスケープの出力の無効化

質問

XML Parser for Java はエスケープの出力の無効化をサポートしますか？

回答

サポートします。バージョン 2.022 以上の XML Parser for Java は、xsl:text によってエスケープの出力を無効にするオプションを提供します。

Oracle8 リリース 8.0.5 での XML Parser for Java の使用

質問

XML Parser for Java は、Oracle9*i* 以外では使用できないのですか？Oracle8 リリース 8.0.5 で使用できますか？

回答

XML Parser for Java は、サポートされている Java VM のすべてのバージョンで使用できます。Oracle9*i* と異なる点は、データベースにロードして、内部 Java VM である Oracle JVM を使用できることのみです。リリース 8.0.5 では、外部で実行し、JDBC を介して接続します。

複数の XML 文書のデリミタ付け

質問

複数の XML 文書を単一の文字列として読み取る（および分離する）必要があります。解決策の 1 つとして、XML 文書内には存在しないことが確実ないくつかの（プログラムによって生成された）特殊文字を使用して、これらの XML 文書にデリミタを付けることができます。その後、個々の文書を簡単にトークン化して、必要に応じて取得 / 解析できます。

このような方法は行われたことがありますか？デリミタとして使用できる文字の提案はありますか（たとえば、#x0 ～ #x8 の範囲の文字は XML 文書内に存在できますか）？

回答

正当性に関しては、文字を 8 ビットに制限する場合は #x0 ～ #x8、#xB、#xC、#xE および #xF は正当ではありません。ただし、これは文書を事前処理し、すべてのパーサーがすべての不当な文字を拒否するわけではないという、例外に依存しない場合です。

その後、要素を文書に追加できます。

XML および実体参照 : XML Parser for Java

質問

1. XML Parser for Java は、&[whatever] などの実体参照を拡張しません。かわりに、すべての値を NULL にします。拡張させる方法を教えてください。
2. 内部エンティティに（スウェーデン語の文字 …„ などの）国際化文字は指定できないようです。この問題を解決する方法を教えてください。

回答

1. 実体参照の処理に関しては、多くのリグレッション・テストが正常に行われているため、エンティティの定義 / 使用に単純なエラーがあると思われます。単純なエラーには、「]> Alpha, then &status」などがあります。
2. キャラクタ・セットのエンコーディングを何語に設定する必要があるのですか？

DDL を挿入しないで XML 文書を分割して格納する方法

質問

1. 任意の XML 文書を分割して、挿入する DDL を作成しないでデータベースに格納することができますか？
2. 問合せにおいて、XML 文書の階層を検索することは可能ですか？

回答

1. できません。既存のスキーマがあるか、または XML から DDL を作成するスタイルシートが存在している必要があります。
2. Oracle8*i* リリース 8.1.6 以上の *inter*Media Text（現在の Oracle Text）では、これを行うことができます。

XML 文書のマージ

質問

2 つの XML 文書をマージする方法を教えてください。

回答

現在の DOM の仕様では行うことができません。DOM2 の仕様では、できるようになる可能性があります。

これを行うには、DOM による方法か、XSLT ベースの方法を使用できます。DOM を使用する場合、所有者エラーを回避するために、他の文書にノードを追加する前に、1 つの文書からノードを削除する必要があります。

次に、XSLT ベースの方法の例を示します。次の 2 つの XML ソース・ファイルがあるとします。

demo1.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
  </msg>
</messages>
```

demo2.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <text>This is another Message</text>
  </msg>
</messages>
```

一致する「<key>」の値に基づいて demo1.xml を demo2.xml に結合するスタイルシートを次に示します。

demomerge.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:variable name="doc2" select="document('demo2.xml')"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
<xsl:template match="msg">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
      <text><xsl:value-of select="$doc2/messages/msg[key=current()/key]/text"/>
</text>
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

コマンドライン「oraxsl」を使用してこれをテストする場合、次のとおり行います。

```
$ oraxsl demo1.xml demomerge.xsl
```

次のようなマージ結果を取得します。

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
    <text>This is another Message</text>
  </msg></messages>
```

2 つの表の間にある同等のデータベースによる結合ほど、サイズが大きいファイルには効果的ではありません。ただし、単に「Cannot Find Class（クラスが見つかりません）」というエラーに対処するために XML ファイルを使用する場合には有効です。

タグの値の取得

質問

XML 文書の解析に SAX を使用しています。特定のタグの値を取得する方法を教えてください。たとえば、Java の場合、title の値を取得する方法を教えてください。startElement メソッド、endElement メソッドおよび characters メソッドがあることは知っています。

回答

SAX による解析の間、要素の値は、startElement イベントの後から対応する endElement イベントがコールされるまでに通知された文字の連結になります。

ユーザーへの JAVASYSPRIV の付与

質問

Windows NT 4.0 上で Oracle XML Parser for Java を使用しています。外部 DTD を使用して XML 文書を解析しているとき、次のエラーが戻されます。

```
<!DOCTYPE listsamplereceipt SYSTEM
"file:/E:/ORACLE/utl_file_dir/dadm/ae.dtd">
java.lang.SecurityExceptionat
oracle.aurora.rdbms.SecurityManagerImpl.checkFile(SecurityManagerImpl.java)at
oracle.aurora.rdbms.SecurityManagerImpl.checkRead(SecurityManagerImpl.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
sun.net.www.MimeTable.load(MimeTable.java)at
sun.net.www.MimeTable.<init>(MimeTable.java)at
sun.net.www.MimeTable.getDefaultTable(MimeTable.java)at
sun.net.www.protocol.file.FileURLConnection.connect(FileURLConnection.java)at
sun.net.www.protocol.file.FileURLConnection.getInputStream(FileURLConnection.
java)at
java.net.URL.openStream(URL.java)at
oracle.xml.parser.v2.XMLReader.openURL(XMLReader.java:2313)at
oracle.xml.parser.v2.XMLReader.pushXMLReader(XMLReader.java:176)at
...
```

この原因を教えてください。

回答

このコードを実行しているユーザーが外部ファイルまたは URL をオープンできるように、このユーザーに JAVASYSPRIV を付与します。

他の XML ファイルへの外部 XML ファイルの挿入 : 解析済外部エンティティ

質問

1. 外部 XML ファイルを他の XML ファイルへ挿入しようとしています。Oracle XML Parser for Java は、解析済外部エンティティをサポートしますか ?
2. 当社のアプリケーションのリリース 10.7 および 11.0 を持つ顧客に出荷された Oracle XML Parser for Java のバージョンが 1.0 であるため、バージョン 1.0 を使用しています。バージョン 1.0 でこれが可能ですか ? それとも、これを行う他のコード例がありますか ?

   ファイル b.xml を次の形式にすることができません。

   ```
   <?xml version="1.0" ?>
   <b>
     <ok/>
   </b>
   ```

   Oracle XML Parser には、XML ファイルを解析して解析済の出力を参照できるユーティリティが付属していますか ?

回答

1. IE 5.0 は、XML ファイルを解析し解析済の出力を表示します。HTML ページをロードするようにファイルをロードしてください。

   IE5 でのブラウズおよび Oracle XML Parser による解析の両方に、次の構文が使用できます。これは Oracle XML Parser のバージョン 1.0 でも確実に動作しますが、バージョン 1.0 より高速な最新のバージョンの Oracle XML Parser を使用する必要があります。

   ファイル : a.xml

   ```
   <?xml version="1.0" ?>
   <!DOCTYPE a [<!ENTITY b SYSTEM "b.xml">]>
    <a>&b;</a>
   ```

   ファイル : b.xml

   ```
    <ok/>
   ```

   a.xml をブラウズ / 解析すると、次の文が戻されます。

   ```
   <a>
     <ok/>
   </a>
   ```

2. 断言はできません。解析済外部エンティティは、整形式の文書フラグメントである必要があるのみです。CLASSPATH にある次のプログラム（バージョン 1.0 の xmlparser.jar の場合）には、解析および解析済文書の出力が示されています。ここでは、文字列から解析していますが、URL が指定される場合は、ファイルからの解析にも同じメカニズムが使用されます。

```
import oracle.xml.parser.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.xml.sax.*;
/*
** Simple Example of Parsing an XML File from a String
** and, if successful, printing the results.
**
** Usage: java ParseXMLFromString <hello><world/></hello>
*/
public class ParseXMLFromString {
  public static void main( String[] arg ) throws IOException, SAXException {
    String theStringToParse =
       "<?xml version='1.0'?>"+
       "<hello>"+
       "  <world/>"+
       "</hello>";
    XMLDocument theXMLDoc = parseString( theStringToParse );
    // Print the document out to standard out
    theXMLDoc.print(System.out);
  }
  public static XMLDocument parseString( String xmlString ) throws
   IOException, SAXException {
   XMLDocument theXMLDoc     = null;
    // Create an oracle.xml.parser.v2.DOMParser to parse the document.
    XMLParser theParser = new XMLParser();
    // Open an input stream on the string
    ByteArrayInputStream theStream =
         new ByteArrayInputStream( xmlString.getBytes() );
    // Set the parser to work in non-Validating mode
    theParser.setValidationMode(DTD_validation);
    try {
      // Parse the document from the InputStream
      theParser.parse( theStream );
      // Get the parsed XML Document from the parser
      theXMLDoc = theParser.getDocument();
    }
    catch (SAXParseException s) {
      System.out.println(xmlError(s));
      throw s;
```

```
    }
    return theXMLDoc;
  }
  private static String xmlError(SAXParseException s) {
     int lineNum = s.getLineNumber();
     int  colNum = s.getColumnNumber();
     String file = s.getSystemId();
     String  err = s.getMessage();
     return "XML parse error in file " + file +
            "\n" + "at line " + lineNum + ", character " + colNum +
            "\n" + err;
  }
}
```

XML パーサーのコマンドライン・インタフェース OraXSL をダウンロードできる場所

質問

oracle.xml.parser.v2.OraXSL はどこからダウンロードできますか？

回答

これは、統合されている XML Parser for Java バージョン 2.0 以上の一部です。XML パーサー、DOM、XPath の実装および XSLT エンジンは、単一のパッケージに統合されています。http://otn.oracle.com/tech/xml/xdk_java/ を参照してください。

Oracle による階層マッピングのサポート

質問

Oracle データベースを使用して主に XML を格納することを考えています。受信した XML 文書を解析し、データおよびタグをデータベース内に格納する必要があります。ただし、Oracle の XML について、次の 2 つのことを懸念しています。

1 つ目は、解析済 XML データのリレーショナル・マッピングです。解析済 XML データを階層的に格納する必要がありますが、これは可能でしょうか？Oracle9*i* の XMLType はこの問題に対応していますか？

2 つ目は、Oracle XML Parser for Java に「あいまいな」コンテンツ・モードがないことです。これは、当社のビジネスにとって制約となっています。Oracle XML Parser for Java に「あいまいな」コンテンツ・モードを追加する計画はありますか？

回答

1 つ目は、多くの顧客によって最初に懸念される問題です。これは、格納する XML データの種類によって異なります。実際には単に発注書などのリレーショナル情報のエンコーディングである XML データグラムを格納する場合は、XML 文書に含まれるデータをリレーショナル表に格納し、特定のデータを抽出する必要がある場合に、必要に応じて XML 形式を再生成するためのパフォーマンスおよび（SQL を介した）問合せの柔軟性が大幅に向上します。

訴訟手続き、書籍の章、リファレンス・マニュアルなどのより複合的なコンテンツを含むドキュメントを格納する場合は、それらのドキュメントをチャンクで格納し、Oracle Text の XML 検索機能を使用して検索することが最適な方法です。

『Building Oracle XML Applications』（Steve Muench 著）は、これらの格納方法および検索方法を多くの例を示して説明しています。

> **参照：** 次の章およびマニュアルを参照してください。
>
> - 第 8 章「Oracle Text を使用した XML データの検索」
> - 『Oracle Text リファレンス』

2 つ目の点については、Oracle の XML パーサーはすべての XML 1.0 標準を実装し、XML 1.0 標準では XML 文書に明白なコンテンツ・モデルがある必要があるため、XML 1.0 準拠のパーサーはあいまいなコンテンツ・モデルを実装できません。
http://www.xml.com/axml/target.html#determinism を参照してください。

XSLT プロセッサおよび XSL スタイルシート

XSL での HTML エラー

質問

何が問題であるのかわかりません。次に news_xsl.xsl ファイルを示します。

```
<?xml version ="1.0"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <HTML>
    <HEAD>
        <TITLE>   Sample Form    </TITLE>
        </HEAD>
     <BODY>
      <FORM>
       <input type="text" name="country" size="15">    </FORM>
    </BODY>
```

```
 </HTML>
</xsl:template>
</xsl:stylesheet>


ERROR:End tag 'FORM' does not match the start tag 'input'. Line 14, Position 12
</FORM>-
----------^news.xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="news_xsl.xsl"?>
<GREETING/>
```

回答

HTML とは異なり、XML ではすべての開始タグに終了タグが必要であることを理解する必要があります。したがって、この場合の入力にも、対応する終了タグを含める必要があります。これを解決するには次のとおりスクリプトを変更します。

```
<FORM>
<input type="text" name="country" size="15"> </input>
</FORM>
```

または

```
<FORM>
<input type="text" name="country" size="15"/>
</FORM>
```

さらに、HTML とは異なり XML ではタグの大文字 / 小文字が区別されることを理解しておく必要もあります。これらのことに注意してください。

<xsl:output method="html"/> のサポート

質問

出力メソッド html は、XML/XSL パーサーの最近のバージョンでサポートされていますか？
 タグを <xsl:output method="xml"/> 宣言で使用しようとしましたが、XML 文書が整形式でないという XSLException エラー・メッセージが表示されました。そこで、出力メソッド宣言 <xsl:output method="html"/> を試してみましたが、同じ結果になりました。

使用した単純な XSL スタイルシートを次に示します。

```
<?xml version="1.0"?> <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="html"/>
<xsl:template match="/">      <HTML>         <HEAD></HEAD>         <BODY>
<P>           Blah blah<BR>           More blah blah<BR>         </P>
</BODY>      </HTML>   </xsl:template>
```

<IMG>、
 などの整形式でないタグを XSL スタイルシートでどのように使用するのかということです。

回答

<xsl:output> のすべてのオプションはサポートされています。ここでの問題は、使用している XSL スタイルシートが整形式の XML 文書である必要があることです。このため、使用しているすべての
 要素のかわりに、
 を使用する必要があります。<xsl:output method="html"/> は、XSLT エンジンが変換の結果を書き出すときに、適切な HTML ドキュメントを要求します。XSLT エンジンは、整形式の XML を読み取る必要があります。

質問

前述の質問の回答に関して質問があります。XML から HTML への変換を行う XSL スタイルシートがあります。すべて正常に動作しますが、整形式でない HTML タグに対してのみ正常に動作しません。前述の例を使用して次のとおり指定するとします。

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
......
<input type="text" name="{NAME}" size="{DISPLAY_LENGTH}" maxlength="{LENGTH}">
</input>
......
</xsl:stylesheet>
```

これは HTML を次の書式でレンダリングします。

```
<HTML>......<input type="text" name="in1" size="10" maxlength="20"/>
......
</HTML>
```

IE はこれを処理できますが、Netscape は処理できません。ブラウザ間で完全に互換性のある HTML を、XSL を使用して生成する方法はありますか？

回答

質問において、次のタグを使用しているとします。

```
<input ... />
```

次のタグは使用していません。

```
<input>
```

この場合は、HTML 出力は行われていないようなので、XSLProcessor.processXSL() をコールする方法が誤っているようです。次の構文を使用してください。

```
void processXSL(style,sourceDoc,PrintWriter)
```

次の構文は使用しないでください。

```
DocumentFragment processXSL(style,sourceDoc)
```

これで正常に動作します。

Netscape 4.0: XSL による <meta> タグの出力の回避

質問

<xsl:output method="html" encoding="iso-8859-1" indent = "no" /> を使用しています。XSLT が先頭に <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> を出力しないようにできますか？Netscape 4.0 ではこの文の処理に問題があります。ページが 2 回レンダリングされます。

回答

XSLT 1.0 勧告のセクション 16.2（「HTML Output Method」）では、HEAD 要素がある場合は、HTML 出力メソッドは、実際に使用されているキャラクタのエンコーディングを指定して、HEAD 要素の開始タグの直後に META 要素を追加すると記述されています。

次に例を示します。

```
<HEAD><META http-equiv="Content-Type" content="text/html; charset=EUC-JP">
```

したがって、すべての XSLT 1.0 互換のエンジンにはこれを追加する必要があります。

質問

Netscape 4.0 には次のようなエラーがあります。

Mozilla は、メタエンコーディング・タグを検出すると、ページのレンダリングを停止してリフレッシュを行います。このため、画面の表示がちらつきます。おそらくサーブレット Outputstream で置換を行う必要があるのですが、実行したくありません。代替策はありますか？

回答

次のような代替策が可能です。

- HTML ページに <HEAD> セクションを指定しないことです。XSLT の仕様によると、これによって <META> タグの挿入が抑制されます。
- 出力に method="HTML" は使用しないでください。<HTML>（大文字 / 小文字の組合せにかかわらず）で始まる結果ツリーの仕様に従ってデフォルトで HTML になるため、明示的に method="xml" または method="text" を指定する必要があります。

どちらも簡単ではありませんが、対応策にはなります。

XSL エラー・メッセージ

質問

XSL エラー・メッセージに関する詳細情報の参照先を教えてください。「XSL-01900: 例外が発生しました」というエラーが表示されます。これは何を意味していますか？この例外の原因をどこで調べることができますか？

回答

Java を使用している場合、例外ルーチンを作成してエラーをトラップできます。JDeveloper などのツールも有効です。

Oracle コンポーネントのエラー・メッセージは、通常はより簡単に理解できるようにしています。XSL-01900 は、発生可能な内部エラーまたは不適切な使用を示します。

HTML の生成 :「<」文字

質問

user_tab_columns 表の列名および次の XSL コードを使用して、データを入力するための HTML フォームを生成しようとしています。

```
<xsl:template match="ROW">
<xsl:value-of select="COLUMN_NAME"/>
<: lt;INPUT NAME="<xsl:value-of select="COLUMN_NAME"/>>
</xsl:template>
```

「gt;」は「>」として生成されますが、「lt;」は「#60;」として生成されます。「<」文字を生成する方法を教えてください。

コメント

次の構文を使用すると正常に生成されます。

```
<xsl:text disable-output-escaping="yes">entity-reference</xsl:text>
```

oraxsl では正常に行われるが XSLSample.java では正常に行われない HTML での「<」の変換

質問

XML から HMTL を表示できません。XML ファイルの XML タグ内に、HTML フラグメントを次のとおり格納しています。

```
<PRE>
<body.htmlcontent>
<&#60;table width="540" border="0" cellpadding="0"
cellspacing="0">&#60;tr>&#60;td>&#60;font face="Helvetica, Arial" size="2">&#60;!--
STILL IMAGE GOES HERE -->&#60;img src="graphics/imagegoeshere.jpg"  width="200"
height="175" align="right" vspace="0" hspace="7">&#60;!-- END STILL IMAGE TAG
-->&#60;!-- CITY OR TOWN NAME GOES FIRST FOLLOWED BY TWO LETTER STATE ABBREVIATION
-->&#60;b>City, state abbreviation&#60;/b> - &#60;!-- CITY OR TOWN NAME ENDS HERE
-->&#60;!-- STORY TEXT STARTS HERE -->Story text goes here.. &#60;!-- STORY TEXT
ENDS HERE -->&#60;/font>&#60;/td>&#60;/tr>&#60;/table>
</body.htmlcontent>
</PRE>
```

次の構文を XML で使用します。

```
<xsl:value-of select="body.HTMLcontent" disable-output-escaping="yes"/>
```

それでも、HTML 出力は次のとおりになります。

```
<PRE>&#60;</PRE>
```

すべての HTML タグがブラウザに表示されます。HTML を適切に表示する方法を教えてください。

コメント

正常に表示されているようには見えません。すべての「<」は「#60;」になり、コード内では「#60;」の前にアンパサンド（&）が追加されます。ブラウザでもそのように表示されます。

さらに理解できないことに、これは oraxsl では正常に表示されますが、XSLSample.java では正常に表示されません。

回答

これには理由があります。次に原因を示します。

- oraxsl は、内部的に void XSLProcessor.processXSL (style,source,printwriter); を使用します。
- XSLSample.java は、DocumentFragment XSLProcessor.processXSL (style,source); を使用します。

oraxsl は、<xsl:output>、および妥当な XML でない可能性がある出力の書出しに関連するすべてのオプション（出力のエスケープの無効化を含みます）をサポートします。
XSLSample.java は、完全な XML 対 XML ツリーから戻されます。したがって、出力がなく、結果の DOM ツリー・フラグメントのみが戻されているため、<xsl:output> またはエスケープの無効化は使用されません。

XSLT の例

質問

XSLT の良い例または簡単なチュートリアルがあるサイトはありますか？

回答

次のサイトには、XML/XSLT/XPath に関する多くのチュートリアルがあります。

http://zvon.vscht.cz/ZvonHTML/Zvon/zvonTutorials_en.html

XSLT の機能

質問

1. Oracle XDK が実装する XSLT の機能のリストはありますか？
2. XML パーサー は、IE5 より多くの機能勧告を実装しますか？XML パーサーの方が多くの機能を実装していると思います。<xsl:choose... および <xsl:if... は、XML パーサーでは正常に動作しますが、IE5 では見慣れないメッセージが戻されます。

回答

1. XML パーサーは、http://www.w3.org/TR/XSLT にある W3C の XSLT バージョン 1.0 勧告をサポートしています。
2. そのとおりです。XML パーサーは XSLT 勧告に準拠しています。

XSL を使用した XML 文書から他の形式への変換

質問

XML 文書をある形式から他の形式に、XSL（または XSLT）スタイルシートを使用して変換しようとしています。これを Java コードに組み込む前に、次のコマンドラインから変換をテストしようとしました。

```
> java oracle.xml.parser.v2.oraxsl jwnemp.xml jwnemp.xsl newjwnemp.xml
```

問題は、前述のコマンドを使用すると、変換済の XML ファイル（newjwnemp.xml）ではなく、jwnemp.xsl からのコードを含むファイルが戻されることです。この原因がわかりません。2 つの入力ファイルを添付しました。

```
<?xml version="1.0"?>
<employee_data>
   <employee_row>
      <employee_number>7950</employee_number>
      <employee_name>CLINTON</employee_name>
      <employee_title>PRESIDENT</employee_title>
      <manager>1111</manager>
      <date_of_hire>20-JAN-93</date_of_hire>
      <salary>125000</salary>
      <commission>1000</commission>
      <department_number>10</department_number>
   </employee_row>
</employee_data>
```

```
<?xml version='1.0'?>
<ROWSET xmlns:xsl="HTTP://www.w3.org/1999/XSL/Transform">
   <xsl:for-each select="employee_data/employee_row">
   <ROW>
      <EMPNO><xsl:value-of select="employee_number"/></EMPNO>
      <ENAME><xsl:value-of select="employee_name"/></ENAME>
      <JOB><xsl:value-of select="employee_title"/></JOB>
      <MGR><xsl:value-of select="manager"/></MGR>
      <HIREDATE><xsl:value-of select="date_of_hire"/></HIREDATE>
      <SAL><xsl:value-of select="salary"/></SAL>
      <COMM><xsl:value-of select="commission"/></COMM>
      <DEPTNO><xsl:value-of select="department_number"/></DEPTNO>
   </ROW>
   </xsl:for-each>
</ROWSET>
```

回答

xmlns:xsl="..." 名前空間宣言に、不適切な XSL 名前空間の URI が指定されているため、ほとんどの場合この問題が発生します。

xmlns:xsl=http://www.w3.org/1999/XSL/Transform を使用すると解決します。

xmlns:xsl=" 任意の他の文字列 " を使用すると、ご質問のような問題が発生します。

XSL に関する情報

質問

XSL の使用に関する情報が入手できません。どこで入手できますか？XML および XSL のファイルを入手して、このテクノロジを使用して実現できることを会社に示す必要があります。XML のみでは、ユーザーにとってはあまり印象的ではありません。

回答

入門的な XSL 例は、次のページにあります。

http://metalab.unc.edu/xml/books/bible/updates/14.html

このページでは、XSL の要点が英語で説明されています。XSL は、実際は XML ファイル以上のものではありません。したがって、顧客に示してもそれほど印象的ではないと思います。XSL に関する主要な Web サイト http://www.w3.org/style/XSL/ もあります。

XSL プロセッサおよび複数の出力

質問

1 つの XML および XSL から複数の結果を作成する XSL プロセッサについての記述を見たことがあります。これはどのように行われるのですか？

回答

Oracle XML Parser 2.0.2.8 の <ora:output> のサポートによって処理されます。

XML/XSL に関する推奨書籍

質問

XML/XSL を理解するための推奨書籍はありますか？

回答

XML テクノロジについて詳しく説明した記事、ホワイトペーパーおよび書籍は数多くあり、Web から入手できます。次に、最も有効なリソースを示します。

- 『XML, Java, and the Future of the Web by Jon Bosak』Sun 社（http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm）
- 『XML for the Absolute Beginner』Mark Johnson、JavaWorld 著（http://www.javaworld.com/jw-04-1999/jw-04-xml_p.html）
- 『XML And Databases』Ronald Bourret 著（ダルムシュタット工科大学）（http://www.rpbourret.com/index.htm）
- 『XMLAndDatabases.htm』W3C
- XML 仕様（http://www.w3.org/XML/）
- XML.com: XML リソースおよびコメントの広範囲なコレクション（http://www.xml.com/）
- 『Annotated XML Specification』Tim Bray、XML.com 著（http://www.xml.com/axml/testaxml.htm）
- 『The XML FAQ』W3C XML Special Interest Group 作成（http://www.ucc.ie/xml/ XML.org）：企業が XML データを交換できる、XML DTD の業界情報公開機関
- http://xml.org/
- xDev: DataChannel XML 開発者のページ（http://xdev.datachannel.com/）

HP/UX プラットフォーム用の XDK

質問

HP/UX プラットフォーム用の XML パーサーまたは XDK をリリースする計画がありますか？

回答

XML Parser for C/C++ および Class Generator for C++ の HP-UX 版は入手可能です。http://otn.oracle.com/ 上のお知らせを参照してください。

大量の XML 文書の圧縮

質問

XML 文書は、CLOB としてデータベースに保存するときに圧縮できますか？XML 文書を圧縮した場合、文書に対して Oracle Text（*inter*Media Text）を使用するときの含意事項を教えてください。最大 1MB の大規模な XML 文書があり、それらを最小化する必要があります。

主な要件は、格納された XML 文書が履歴情報（データ・ウェアハウス環境）であるため、ディスク記憶域の点でコストを節約することです。格納前に文書を圧縮できると、多くのディスク領域を節約できます。検索機能は二次的なものですが、大きいメリットにはなります。

回答

1. XDK for Java は、Oracle の圧縮メカニズムをサポートします。これは、ストリーム圧縮 / 圧縮解除をサポートします。圧縮は、XML 文書内のマークアップを削除することによって実行されます。初期バージョンは圧縮したデータの検索をサポートしません。この機能は、将来のリリースでサポートされる予定です。

2. XML 文書を格納および検索する必要がある場合は、Oracle Text でこれを処理できます。Oracle Text では、個々の文書のサイズは問題になりません。

   1MB の文書を圧縮してディスク領域 / コストを節約する必要がある場合、Oracle Text は圧縮した XML 文書を自動的に処理できません。

   唯一の懸念は、圧縮解除の実行によるパフォーマンスへの影響です。クライアントとサーバー間の XML 転送のみが懸念される場合は、HTTP 圧縮の方がより簡単である場合があります。

2 つの表に基づいて XML 文書を生成する方法

質問

マスターとディテールの関係にある 2 つの表に基づいて XML 文書を生成する必要があります。次の 2 つの表があるとします。

- ID 列および PARENT_NAME 列（キー列 = ID）を含む PARENT 表
- PARENT_ID 列、CHILD_ID 列および CHILD_NAME 列（キー列 = PARENT_ID + CHILD_ID）を含む CHILD 表

PARENT および CHILD は、マスターとディテールの関係にあります。次のようなドキュメントを生成する方法を教えてください。

```
<?xml version = '1.0'?>
  <ROWSET>
     <ROW num="1">
       <parent_name>Bill</parent_name>
         <child_name>Child 1 of 2</child_name>
         <child_name>Child 2 of 2</child_name>
      </ROW>
      <ROW num="2">
       <parent_name>Larry</parent_name>
         <child_name>Only one child</child_name>
      </ROW>
  </ROWSET>
```

回答

オブジェクト・ビューを使用して、マスター / ディテール構造から XML 文書を生成できます（生成する必要があります)。この場合は、次のようになります。

```
create type child_type is object
(child_name <data type child_name>) ;
/
create type child_type_nst
is table of child_type ;
/

create view parent_child
as
select p.parent_name
, cast
  ( multiset
    ( select c.child_name
      from   child c
```

```
      where  c.parent_id = p.id
    ) as child_type_nst
  ) child_type
from parent p
/
```

SQL から XML へのマッピング・ユーティリティによって処理される SELECT * FROM parent_child は、親子関係に対する有効な XML 文書を生成します。ただし、その構造は質問で提示されたようなものではなく、次のような構造になります。

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <PARENT_NAME>Bill</PARENT_NAME>
      <CHILD_TYPE>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Child 1 of 2</CHILD_NAME>
         </CHILD_TYPE_ITEM>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Child 2 of 2</CHILD_NAME>
         </CHILD_TYPE_ITEM>
      </CHILD_TYPE>
  </ROW>
   <ROW num="2">
      <PARENT_NAME>Larry</PARENT_NAME>
      <CHILD_TYPE>
         <CHILD_TYPE_ITEM>
            <CHILD_NAME>Only one child</CHILD_NAME>
         </CHILD_TYPE_ITEM>
      </CHILD_TYPE>
  </ROW>
</ROWSET>
```

21

XML Schema Processor for Java の使用

この章の内容は次のとおりです。

XML Schema の概要

XML Schema は、XML 文書のコンテンツおよび構造を XML で記述するために W3C で作成中です。XML Schema には、既存の DTD を XML Schema に変換できるようにすべての DTD 機能が含まれています。また、XML Schema には DTD にはない追加機能があります。

DTD と XML Schema との相違

DTD は、XML 1.0 が提供する、XML マークアップの制約を宣言するメカニズムです。DTD によって、次のものを指定できます。

- XML 文書に出現できる要素
- 要素に含めることができる要素
- 要素が出現できる順序

XML Schema は DTD と同様の目的を果たしますが、XML 文書の制約の指定についてはより柔軟であり、特定のアプリケーションではより有効である可能性があります。21-3 ページの「DTD の制限事項」を参照してください。

次の XML 文書について考えてみます。

```
<?XML version="1.0">
<publisher pubid="ab1234">
  <publish-year>2000</publish-year>
     <title>The Cat in the Hat</title>
     <author>Dr. Seuss</author>
     <artist>Ms. Seuss</artist>
     <isbn>123456781111</isbn>
</publisher>
```

前述の XML 文書に対する典型的な DTD について考えてみます。

```
<!ELEMENT publisher (year,title, author+, artist?, isbn)>
<!ELEMENT publish-year (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
...
```

DTD の制限事項

XML マークアップ宣言で知られる DTD は次の処理を含む特定のアプリケーションには不十分であると考えられています。

- 文書の作成および公開
- メタデータの交換
- E-Commerce
- データベース間の操作

DTD には、次のような制限事項があります。

- DTD は名前空間テクノロジと統合されないため、ユーザーはコードをインポートおよび再利用できません。
- DTD は文字データ以外のデータ型をサポートしないため、メタデータ標準およびデータベース・スキーマの記述が制限されます。
- アプリケーションは、DTD が可能にする水準より柔軟に文書構造の制約を指定する必要があります。

XML Schema の機能

表 21-1 に、XML Schema の機能を示します。XML Schema の機能には DTD の機能が含まれていることに注意してください。

表 21-1 XML Schema の機能

XML Schema の機能	DTD
組込みデータ型	
XML Schema では、一連の組込みデータ型を指定できます。これらの一部は、自身がコールする基本形データ型によって定義されます。これらのデータ型は、次の型システムの基礎を構成します。	DTD は、文字列以外のデータ型をサポートしません。
STRING（文字列）、BOOLEAN（ブール）、FLOAT（浮動）、DECIMAL（小数）、DOUBLE（実数値）、TIMEDURATION（時間）、TIMEINSTANT（時刻）、TIME（時）、DATE（日付）、YEARMONTH（年 / 月）、YEAR（年）、MONTHDAY（月 / 日）、DAY（日）、MONTH（月）、BINARY（バイナリ）、URIREFERENCE（URI 参照）、ID、IDREF（ID 参照）、ENTITY（エンティティ）、QNAME。	
その他のデータ型は、基本型で定義された導出データ型です。	

表 21-1 XML Schema の機能 （続き）

XML Schema の機能	DTD

ユーザー定義のデータ型
ユーザーは、組込みデータ型から独自のデータ型を導出できます。データ型を導出するには、制限、リスト、共用体という 3 つの方法があります。制限は、制約ファセットをベース型に適用することによって、より制限されたデータ型を定義します。リストは、単純にその項目型の値のリストを許可します。共用体は新しい型を定義し、その値はメンバー型のいずれかにすることができます。
たとえば、publish-year 型の値が特定の範囲内になるように指定するには、次のとおり指定します。
`<SimpleType name = "publish-year">`
`  <restriction base="year">`
`  <minInclusive value="1970"/>`
`  <maxInclusive value="2000"/>`
`  </restriction>`
`</SimpleType>`
制約ファセットには、length（長さ）、minLength（最小長）、maxLength（最大長）、pattern（パターン）、enumeration（列挙）、whiteSpace（空白）、maxInclusive（最大内含値）、maxExclusive（最大排他値）、minInclusive（最小内含値）、minExclusive（最小排他値）、precision（精度）、scale（スケール）、encoding（エンコーディング）があります。一部のファセットは、特定のベース型にのみ適用されます。
	DTD の例では、publish-year 要素をさらに制約することはできません。

出現インジケータ（コンテンツ・モデルまたは構造） XML Schema では、インスタンス・ドキュメントまたは要素の構造（complexType）がモデル・グループおよび属性グループで定義されます。属性グループには属性が含まれますが、モデル・グループにはさらにモデル・グループまたは小要素が含まれる場合があります。モデル・グループと属性グループの両方で、ワイルドカードを使用して任意の要素または属性を指定することができます。モデル・グループには順序、全体および選択という 3 つの種類があり、それぞれ小要素間の順序関係、結合関係および分離関係を表します。また、各小要素の出現回数の範囲も指定できます。 データ型と同様に、complexType も他の型から導出できます。導出方法には、制限または拡張があります。導出された型は、ベース型の内容および対応する変更を継承します。継承の他に、型定義は他のコンポーネントを参照することができます。この機能によって、一度定義したコンポーネントを他の様々な構造で使用できます。 XML Schema の型宣言および型定義メカニズムは、DTD より柔軟で強力です。	DTD によって制御される要素内の子要素数は、次の記号で割り当てられます。 ■ ? = 0（ゼロ）または 1。前述の DTD の例では、artist? はアーティストがオプションで、0（ゼロ）または 1 人であることを示しています。 ■ * = 0（ゼロ）以上。 ■ + = 1 以上（前述の DTD の例では、author+ は 1 人以上の作者が存在する可能性を示しています）。 ■ (none) = 1。

表 21-1 XML Schema の機能 （続き）

XML Schema の機能	DTD
ID 制約 XML Schema は、一意性、キーおよびキー参照の宣言によって XML ID/IDREF メカニズムの概念を拡張します。これらは型定義の一部であり、属性のみでなく、要素内容もキーとして許可します。各制約には有効範囲があり、字句文字列ではなく値で比較が行われます。	
インポート / エクスポート・メカニズム（スキーマのインポート、追加および変更） 単一スキーマ・ファイルでは、スキーマのすべてのコンポーネントを定義する必要はありません。XML Schema は、複数のスキーマを作成するメカニズムを提供します。異なる名前空間のスキーマを統合する場合はインポート機能を使用し、同じ名前空間のコンポーネントを追加する場合は追加機能を使用します。また、コンポーネントは追加時に再定義して変更することもできます。	外部スキーマで定義された構造体は使用できません。
拡張メカニズム XML Schema はより柔軟性があり、次の 3 つのモデルをサポートできます。 オープン・モデル - 要素に対して宣言された内容および属性は必須で、その他の内容および属性も使用できます。 精錬可能モデル - 要素に対して内容および属性が宣言されます。このモデルでは、内容および属性を精錬されたサブタイプで宣言できます。 クローズ・モデル - DTD と同様に、要素宣言にない子要素および属性は追加できません。	要素のインスタンスには、スキーマの要素宣言で指定されていない子要素および属性を追加できません。

XML Schema を使用すると、XML 文書のクラスを定義できます。インスタンス・ドキュメントは、特定のスキーマに準拠する XML 文書を示します。

これらのインスタンスおよびスキーマは、特にドキュメントとして存在する必要はありませんが、一般にファイルと呼ばれます。これらは、次のいずれかとして存在する場合があります。

- バイトのストリーム
- データベース・レコード内のフィールド
- XML Infoset の情報項目のコレクション

参照： 次の Web サイト、章およびマニュアルを参照してください。

- http://www.w3.org/TR/xmlschema-0/
- 付録 C「XDK for Java: 仕様および早見表」
- 『Oracle9*i* XML リファレンス』

XML Schema Processor for Java の機能

Oracle XML Schema Processor for Java には次の機能があります。

- 単純型および複合型をサポートします。
- Oracle XML Parser for Java に統合されています。
- 次に示す W3C の XML Schema 草案をサポートしています。
  - XML Schema Part 0: Primer（入門書）
  - XML Schema Part 1: Structures（構造）
  - XML Schema Part 2: Datatypes（データ型）

サポートするキャラクタ・セット

XML Schema Processor for Java は、次のエンコーディングを使用するドキュメントをサポートします。

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO 2022-JP
- ISO 2022-KR
- ISO 8859-1 ～ ISO 8859-9
- ISO 10646-UCS-2
- ISO 10646-UCS-4

- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

XML Schema Processor for Java の実行要件

XML Schema Processor for Java を実行するには、次のコンポーネントが必要です。

- オペレーティング・システム : Java 1.1.x をサポートするオペレーティング・システム
- Java: JDK 1.1.x 以上

オンライン・ドキュメント

Oracle XML Schema Processor for Java のドキュメントは、インストール場所の doc/ ディレクトリにあります。

リリース固有の注意事項

アーカイブのルート・ディレクトリにある readme.html ファイルには、不具合修正や追加の API などのリリース固有の情報が含まれています。

Oracle XML Schema Processor は、Java で作成されています。これには、XML Parser for Java が含まれています。

標準への準拠

XML Schema Processor は、次の W3C の標準に準拠しています。

- XML Schema Part 0: Primer（入門書）
- XML Schema Part 1: Structures（構造）
- XML Schema Part 2: Datatypes（データ型）

XML Schema Processor for Java のディレクトリ構造

表 21-2 に、Windows NT 上に XML Schema Processor for Java をインストールした後のディレクトリ構造を示します。UNIX の場合も、同じ構造がインストールされます。

表 21-2 Windows NT 上にインストールした XML Schema Processor のディレクトリ構造

ディレクトリおよびファイル	説明
license.html	ライセンス契約のコピー
readme.html	リリース・ノートおよびインストール時の注意
doc¥	ドキュメントのディレクトリ
lib¥	クラス・ファイルのディレクトリ
sample¥	サンプル・コード・ファイル

XML Schema Processor for Java の使用方法

図 21-1 に示すとおり、Oracle XML Schema Processor は次の 2 つの主要タスクを実行します。

- Builder が、スキーマ XML 文書からスキーマを作成します。
- Validator が、スキーマを使用してインスタンス・ドキュメントを検証します。

スキーマを作成する場合、Builder はまず DOM パーサーをコールして、スキーマ XML 文書を対応する DOM ツリーに解析します。次に、それを内部スキーマ・オブジェクトにコンパイルします。Validator は、SAX パーサーとインスタンス・ドキュメント用アプリケーションの間のフィルタとして機能します。Validator は、インスタンス・ドキュメントの SAX イベントを入力として取り、そのイベントをスキーマに対して検証します。Validator は妥当でない XML コンポーネントを検出すると、エラー・メッセージを送信します。Validator の出力は次のとおりです。

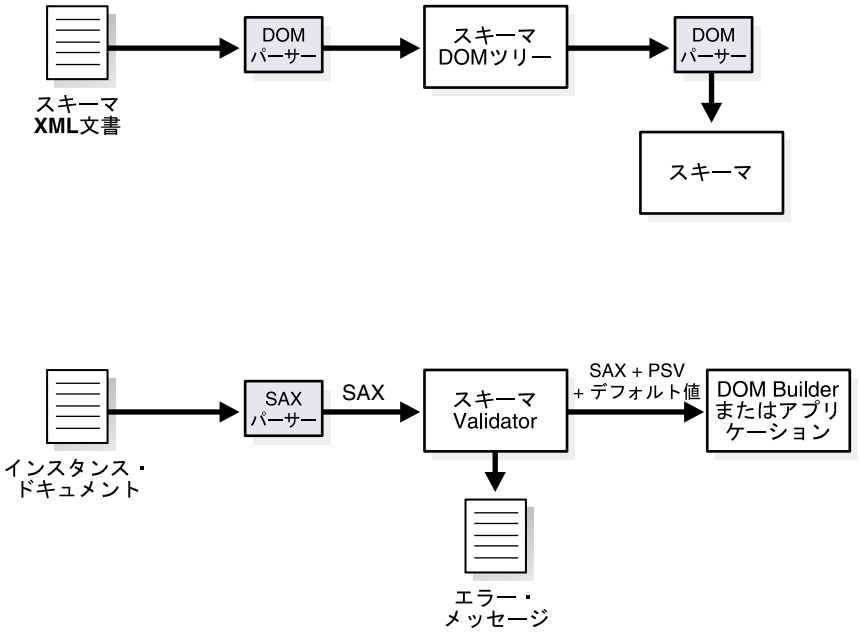
- 入力 SAX イベント
- 提供するデフォルト値
- スキーマ検証後（PSV）の情報

XML Schema Processor for Java の API は単純です。次のいずれかの操作を行うことができます。

- 21-16 ページの「XML Schema for Java の例 7: XSDSample.java」に示すとおり、`DOMParser` で `setSchemaValidationMode()` を使用します。
- `XSDBuilder` を使用してスキーマを明示的に作成し、21-18 ページの「XML Schema for Java の例 8: XSDSetSchema.java」に示すとおり、`XMLParser` に対してスキーマを設定します。

`xmlclean` と同様のクリーンアップ・コールはありません。新しい XML 文書を検証する前にすべてのメモリーを解放し、状態をリセットする必要がある場合は、コンテキストを終了し、最初からやりなおします。

図 21-1 XML Schema Processor for Java の使用方法



参照：『Oracle9*i* XML リファレンス』の第 I 部「XDK for Java パッケージ」を参照してください。

XML Schema Processor for Java サンプル・プログラムの実行

XML Schema Processor for Java の sample/ ディレクトリには、サンプル XML アプリケーションがあります。これは、Oracle XML Parser を XML Schema Processor for Java とともに使用する方法を示します。このディレクトリで提供されるサンプル Java ファイルは XSDSample です。これは、XML インスタンス・ドキュメントを処理するサンプル・ドライバです。このサンプル・プログラムを実行するには、次の手順を実行します。

1. 「**make**」を実行して .class ファイルを生成します。
2. xmlparserv2.jar、xschema.jar および現在のディレクトリを CLASSPATH に追加します。
3. サンプル・プログラムを **report.xml** ファイルで次のとおり実行します。

   ```
   java XSDSample report.xml
   java XSDSetSchema report.xsd report.xml
   ```

   XML Schema Processor は、report.xsd の XMLSchema 仕様を使用して、report.xml のコンテンツを検証します。
4. サンプル・プログラムを **catalogue.xml** ファイルで次のとおり実行します。

   ```
   java XSDSample   catalogue.xml
   java XSDSetSchema cat.xsd  catalogue.xml
   ```

   XML Schema Processor は、cat.xsd の XMLSchema 仕様を使用して、catalogue.xml のコンテンツを検証します。
5. 次に、XML Schema エラーの例を示します。

   ```
   java XSDSample catalogue_e.xml
   java XSDSample report_e.xml
   ```

   XML Schema Processor は、エラー・メッセージを生成します。

Make ファイル

```
# Makefile for sample java files
# ========================================================================

.SUFFIXES : .java .class

CLASSES = XSDSample.class

# Change it to the appropriate separator based on the OS.
PATHSEP = :
```

```
# XML Parser V2 jar file
XMLPARSER = ../lib/xmlparserv2.jar

# XMLSchema jar file
XSCHEMA = ../lib/xschema.jar

# Assumes that the CLASSPATH contains JDK classes.
CLASSPATH := $(CLASSPATH)$(PATHSEP)$(XMLPARSER)$(PATHSEP)$(XSCHEMA)

%.class: %.java
/usr/local/packages/jdk1.2/bin/javac -classpath "$(CLASSPATH)" $<

# make all class files
all: $(CLASSES)
```

XML Schema for Java の例 1: cat.xsd

cat.xsd は、サンプル XML Schema 定義ファイルです。このファイルは、XSDSetSchema.java プログラムを入力します。XML Schema Processor は、cat.xsd の XMLSchema 仕様を使用して、catalogue.xml のコンテンツを検証します。

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
            targetNamespace="http://www.somewhere.org/BookCatalogue"
            xmlns:catd = "http://www.somewhere.org/Digest"
            xmlns:cat  = "http://www.somewhere.org/BookCatalogue">

<import namespace = "http://www.somewhere.org/Digest"
        schemaLocation = "catd.xsd" />

  <element name="BookCatalogue">
     <complexType>
        <all>
        <element ref="cat:Book" minOccurs="0" maxOccurs="*"/>
        <element name="Digest" type="catd:Digest" minOccurs="0" maxOccurs="*"/>
        </all>
     </complexType>
  </element>
  <element name="Book">
     <complexType content="mixed">
        <group ref="cat:Book"/>
        <attribute name="number" type="integer"/>
        <attribute name="volumeName" type="string"/>
        <attribute name="volumeNumber" type="integer"/>
     </complexType>
  </element>
  <group name="Book">
```

```
      <all>
         <element ref="cat:Title"/>
         <element ref="cat:Author" minOccurs="0" maxOccurs="1"/>
         <element ref="cat:Date"/>
         <element ref="cat:ISBN"/>
         <element ref="cat:Publisher"/>
      </all>
   </group>
   <element name="Title" type="string"/>
   <element name="Author" type="string"/>
   <element name="Date" type="date"/>
   <element name="ISBN" type="string"/>
   <element name="Publisher" type="string"/>
</schema>
```

XML Schema for Java の例 2: catalogue.xml

catalogue.xml は、サンプル XML ファイルです。XML Schema Processor は、XSDSetSchema.java プログラムを使用して、このファイルを XML Schema 定義ファイル cat.xsd に対して検証します。

```
<?xml version="1.0"?>
<BookCatalogue xmlns =
                       "http://www.somewhere.org/BookCatalogue"
               xmlns:xsi =
                       "http://www.w3.org/1999/XMLSchema/instance"
               xsi:schemaLocation =
                       "http://www.somewhere.org/BookCatalogue/
                        cat.xsd">

        <Book number="11" volumeName="any" volumeNumber="1">
           <Date>July, 1998</Date>
           <Title>My Life and Times</Title>
           <Author>Paul McCartney</Author>
           <ISBN>1111-12021-43892</ISBN>
           <Publisher>McMillin Publishing</Publisher>
        </Book>
        <Digest>
           <Title>Book Revwiew</Title>
           <Volume>42</Volume>
           <Publisher>McMillin Publishing</Publisher>
        </Digest>
</BookCatalogue>
```

XML Schema for Java の例 3: catalogue_e.xml

XML Schema Processor は、XSDSample.java を使用してこのサンプル XML ファイルを処理すると、XML Schema エラーを生成します。

```
<?xml version="1.0"?>
<BookCatalogue xmlns =
                        "http://www.somewhere.org/BookCatalogue"
               xmlns:xsi =
                        "http://www.w3.org/1999/XMLSchema/instance"
               xsi:schemaLocation =
                        "http://www.somewhere.org/BookCatalogue/
                         cat.xsd">

        <Book number="11k" volumeName="any" volumeNumber="1">
           <Date>July, 1998</Date>
           <Title>My Life and Times</Title>
           <Author>Paul McCartney</Author>
           <ISBN>1111-12021-43892</ISBN>
           <Publisher>McMillin Publishing</Publisher>
        </Book>
        <Digest>
           <Title>Book Revwiew</Title>
           <Volume>42</Volume>
           <Author>McMillin Publishing</Author>
        </Digest>
</BookCatalogue>
```

XML Schema for Java の例 4: report.xml

report.xml は、サンプル XML ファイルです。XML Schema Processor は、XSDSetSchema.java プログラムを使用して、このファイルを XML Schema 定義ファイル report.xsd に対して検証します。

```
<?xml version="1.0"?>
<purchaseReport
  xmlns='http://www.example.com/Report'
  period="P3M" periodEnding="1999-12-31"
  xmlns:xsi = "http://www.w3.org/1999/XMLSchema/instance"
   xsi:schemaLocation="http://www.example.com/Report/report.xsd">

 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
```

```
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="755-KY" quantity="4"/>
  </zip>
 </regions>

 <parts>
  <partSpec number="872-AA">Lawnmower</partSpec>
  <partSpec number="926-AA">Baby Monitor</partSpec>
  <partSpec number="833-AA">Lapis Necklace</partSpec>
  <partSpec number="455-BX">Sturdy Shelves</partSpec>
  <partSpec number="755-KY">Sturdy Shelves</partSpec>
 </parts>

</purchaseReport>
```

XML Schema for Java の例 5: report.xsd

report.xsd は、サンプル XML Schema 定義ファイルです。このファイルは、XSDSetSchema.java プログラムを入力します。XML Schema Processor は、report.xsd の XMLSchema 仕様を使用して、report.xml のコンテンツを検証します。

```
<schema targetNamespace='http://www.example.com/Report'
        xmlns='http://www.w3.org/1999/XMLSchema'
        xmlns:r='http://www.example.com/Report'
        xmlns:xipo='http://www.example.com/IPO'
elementFormDefault="qualified">

 <element name="purchaseReport">
  <complexType>
   <element name="regions" type="r:RegionsType"/>
   <element name="parts" type="r:PartsType"/>
   <attribute name="period" type="timeDuration"/>
   <attribute name="periodEnding" type="date"/>
  </complexType>
    <unique name="pZipCode">
   <selector>regions/zip</selector>
   <field>@code</field>
  </unique>
  <key name="pNumKey">
   <selector>parts/part</selector>
   <field>@number</field>
  </key>
```

```
 <keyref name="pKeyRef" refer="pNumKey">
  <selector>regions/zip/part</selector>
  <field>@number</field>
 </keyref>
</element>
<complexType name="RegionsType">
 <element name="zip" minOccurs="1" maxOccurs="unbounded">
  <complexType>
   <element name="part" maxOccurs="unbounded">
    <complexType content="empty">
     <attribute name="number" type="r:Sku"/>
     <attribute name="quantity" type="positiveInteger"/>
    </complexType>
   </element>
   <attribute name="code" type="positiveInteger"/>
  </complexType>
 </element>
</complexType>

<complexType name="PartsType">
 <element name="partSpec" minOccurs="1" maxOccurs="unbounded">
  <complexType content="textOnly">
    <attribute name="number" type="r:Sku"/>
  </complexType>
 </element>
</complexType>
<simpleType name="Sku" base="string">
    <pattern value="\d{3}-[A-Z]{2}"/>
</simpleType>
</schema>
```

XML Schema for Java の例 6: report_e.xml

XML Schema Processor は、XSDSample.java を使用してこのサンプル XML ファイルを処理すると、XML Schema エラーを生成します。

```
<purchaseReport
  xmlns='http://www.example.com/Report'
  period="P3M" periodEnding="1999-12-31"
  xmlns:xsi = "http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation="http://www.example.com/Report  report.xsd">
 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
```

```
   <part number="926-AA" quantity="1"/>
   <part number="833-AAA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="755-KY" quantity="4"/>
  </zip>
 </regions>
 <parts>
  <partSpec number="872-AA">Lawnmower</partSpec>
  <partSpec number="926-AA">Baby Monitor</partSpec>
  <partSpec number="833-AA">Lapis Necklace</partSpec>
  <partSpec number="455-BX">Sturdy Shelves</partSpec>
  <partSpec number="755-KY">Sturdy Shelves</partSpec>
 </parts>
</purchaseReport>
```

XML Schema for Java の例 7: XSDSample.java

```
//import oracle.xml.parser.schema.*;
import oracle.xml.parser.v2.*;
import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSample
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 1)
      {
         System.out.println("Usage: java XSDSample <filename>");
         return;
      }
      process (args[0]);
   }

   public static void process (String xmlURI) throws Exception
   {

      DOMParser dp  = new DOMParser();
      URL       url = createURL (xmlURI);
```

```
    // Set Schema Validation to true
    dp.setSchemaValidationMode(true);
    dp.setValidationMode(false);
    dp.setPreserveWhitespace (true);

    dp.setErrorStream (System.out);

    try
    {
       System.out.println("Parsing "+xmlURI);
       dp.parse (url);
       System.out.println("The input file <"+xmlURI+"> parsed without errors");
    }
    catch (XMLParseException pe)
    {
       System.out.println("Parser Exception: " + pe.getMessage());
    }
    catch (Exception e)
    {
       System.out.println("NonParserException: " + e.getMessage());
    }

}

 // Helper method to create a URL from a file name
 static URL createURL(String fileName)
 {
    URL url = null;
    try
    {
       url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
       File f = new File(fileName);
       try
       {
          String path = f.getAbsolutePath();
          // This is a bunch of weird code that is required to
          // make a valid URL on the Windows platform, due
          // to inconsistencies in what getAbsolutePath returns.
          String fs = System.getProperty("file.separator");
```

```
            if (fs.length() == 1)
            {
               char sep = fs.charAt(0);
               if (sep != '/')
                  path = path.replace(sep, '/');
               if (path.charAt(0) != '/')
                  path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
         }
         catch (MalformedURLException e)
         {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
         }
      }
      return url;
   }

}
```

XML Schema for Java の例 8: XSDSetSchema.java

この例を cat.xsd および catalogue.xml で実行すると、XML Schema Processor は cat.xsd の XML Schema 仕様を使用して catalogue.xml のコンテンツを検証します。

この例を report.xsd および report.xml で実行すると、XML Schema Processor は report.xsd の XML Schema 仕様を使用して report.xml のコンテンツを検証します。

```
import oracle.xml.parser.schema.*;
import oracle.xml.parser.v2.*;

import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSetSchema
{
   public static void main(String[] args) throws Exception
   {
```

```
   if (args.length != 2)
   {
      System.out.println("Usage: java XSDSample <schema_file> <xml_file>");
      return;
   }

   XSDBuilder builder = new XSDBuilder();
   URL    url =  createURL(args[0]);

   // Build XML Schema Object
   XMLSchema schemadoc = (XMLSchema)builder.build(url);
   process(args[1], schemadoc);
}

public static void process(String xmlURI, XMLSchema schemadoc)
throws Exception
{

   DOMParser dp  = new DOMParser();
   URL       url = createURL (xmlURI);

   // Set Schema Object for Validation
   dp.setXMLSchema(schemadoc);
   dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
   dp.setPreserveWhitespace (true);

   dp.setErrorStream (System.out);

   try
   {
      System.out.println("Parsing "+xmlURI);
      dp.parse (url);
      System.out.println("The input file <"+xmlURI+"> parsed without errors");
   }
   catch (XMLParseException pe)
   {
      System.out.println("Parser Exception: " + pe.getMessage());
   }
   catch (Exception e)
   {
      System.out.println ("NonParserException: " + e.getMessage());
   }

}
```

```
    // Helper method to create a URL from a file name
    static URL createURL(String fileName)
    {
       URL url = null;
       try
       {
          url = new URL(fileName);
       }
       catch (MalformedURLException ex)
       {
          File f = new File(fileName);
          try
          {
             String path = f.getAbsolutePath();
             // This is a bunch of weird code that is required to
             // make a valid URL on the Windows platform, due
             // to inconsistencies in what getAbsolutePath returns.
             String fs = System.getProperty("file.separator");
             if (fs.length() == 1)
             {
                char sep = fs.charAt(0);
                if (sep != '/')
                   path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                   path = '/' + path;
             }
             path = "file://" + path;
             url = new URL(path);
          }
          catch (MalformedURLException e)
          {
             System.out.println("Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }

}
```

22

XML Class Generator for Java の使用

この章の内容は次のとおりです。

XML Class Generator for Java の入手方法

Oracle XML Class Generator for Java は、Oracle の XDK for Java に付属しています。OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

Oracle XML Class Generator for Java は、`$ORACLE_HOME/xdk/java/classgen` にあります。

XML Class Generator for Java の概要

XML Class Generator for Java は、XML DTD または XML Schema 定義から Java ソース・ファイルを作成します。これは、次のような状況で有効です。

- アプリケーションが、DTD または XML Schema に基づいて他のアプリケーションに XML メッセージを送信する場合
- XML 文書を構成するための Web フォームのバックエンドとして使用する場合

生成されたクラスを使用して、XML 文書をプログラム的に構築できます。また、XML Class Generator for Java は、生成されたソース・ファイル上にオプションで javadoc コメントを生成できます。XML Class Generator for Java を使用する場合は、XML Parser for Java および XML Schema Processor for Java が必要です。XML Class Generator for Java は、XML Parser for Java と連携して機能します。XML Parser for Java は、DTD または XML Schema を解析し、解析済 XML 文書を XML Class Generator for Java に渡します。

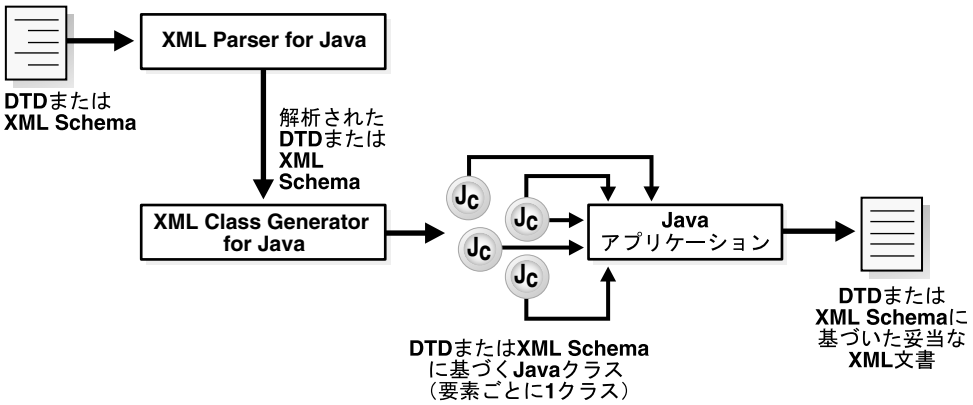
XML Class Generator for Java は、次の 2 つの Class Generator で構成されます。

- **DTD Class Generator**
- **XML Schema Class Generator**

これらは、どちらも `oracg` コマンドライン・ユーティリティから起動できます。

図 22-1 に、XML Class Generator for Java の使用方法の概要を示します。

図 22-1 XML Class Generator for Java: 概要



注意： 「要素ごとに 1 クラス」という部分は、XML Schema Class Generator for Java には適用されません。

参照： 3-19 ページの図 3-7「XDK for Java を使用した XML 文書の生成」を参照してください。

oracg コマンドライン・ユーティリティ

`oracg` コマンドライン・ユーティリティを使用すると、入力する引数に応じて DTD Class Generator for Java または XML Schema Class Generator for Java を起動できます。表 22-1 に、`oracg` の引数を示します。

表 22-1 Class Generator for Java: oracg コマンドライン・ユーティリティの引数

oracg の引数	説明
-h	ヘルプ・メッセージ・テキストの出力
- d <DTD ファイル >	DTD ファイル（.dtd ファイル）
- s <Schema ファイル >	Schema ファイル（.xsd ファイル）
- o < 出力ディレクトリ名 >	出力ディレクトリ
- c	コメント・オプション
- p < パッケージ名 >	名前空間に対応するパッケージ名

Class Generator for Java: XML Schema

XML Class Generator for Java の XML Schema Class Generator の機能を次に示します。

- 最上位の要素であるグローバル要素、simpleType 要素および complexType 要素に対して、それぞれ Java クラスを生成します。
- 最上位の要素であるグローバル要素に対応したクラスは、CGXSDElement を拡張します。
- 要素間の型の階層は、生成された Java クラス内に保持されます。complexType 要素または simpleType 要素が、他の complexType 要素および simpleType 要素を拡張する場合、それらの要素に対応するクラスがベース型の simpleType 要素または complexType 要素を拡張します。それ以外の場合は、CGXSDElement のクラスを拡張します。

名前空間の機能

XML Schema Class Generator は、次の名前空間の機能もサポートします。

- **パッケージ名の作成。**各名前空間に対して 1 つのパッケージが作成されます。このパッケージは、名前空間の要素に対応します。Java クラスは、このパッケージ内に生成されます。
  - 名前空間が未定義の場合は、クラスはデフォルトのパッケージ内に生成されます。
  - スキーマ内に targetNamespace が指定されている場合、クラスを生成するためにはパッケージ名が必要です。
- 名前空間が定義されている場合は、コマンドライン・ユーティリティを介してパッケージ名を指定する必要があります。指定されたパッケージの数は、そのパッケージ名に対応するコマンドライン引数と一致する必要があります。
- **シンボル空間。**単独のシンボル空間が、XML Schema で識別された定義および宣言の構成要素にそれぞれ指定された、ターゲットの名前空間内で使用されます。ただし、simpleType と complexType 要素間でシンボル空間が共有されている場合は例外です。

  指定されたシンボル空間内では、それぞれの名前は一意ですが、複数のシンボル空間内に同じ名前が出現しても競合は発生しません。たとえば、型定義および要素の宣言の両方に同じ名前が使用できます。競合も発生せず、2 つの名前の間にリレーションも必要ありません。競合が発生する場合、simpleType/complexType 要素に対応して生成されたクラス名には「Type」キーワードが追加されます。
- 競合を回避するために、要素の「型」（対応した Java クラスが生成されている）をパラメータとして取るメソッドには、パッケージ名との競合が解決済の完全名を使用します。

XML Schema を使用した XML Class Generator for Java の使用

図 22-2 に、XML Schema を使用した XML Class Generator for Java によってクラスを生成する場合のコール順序を示します。

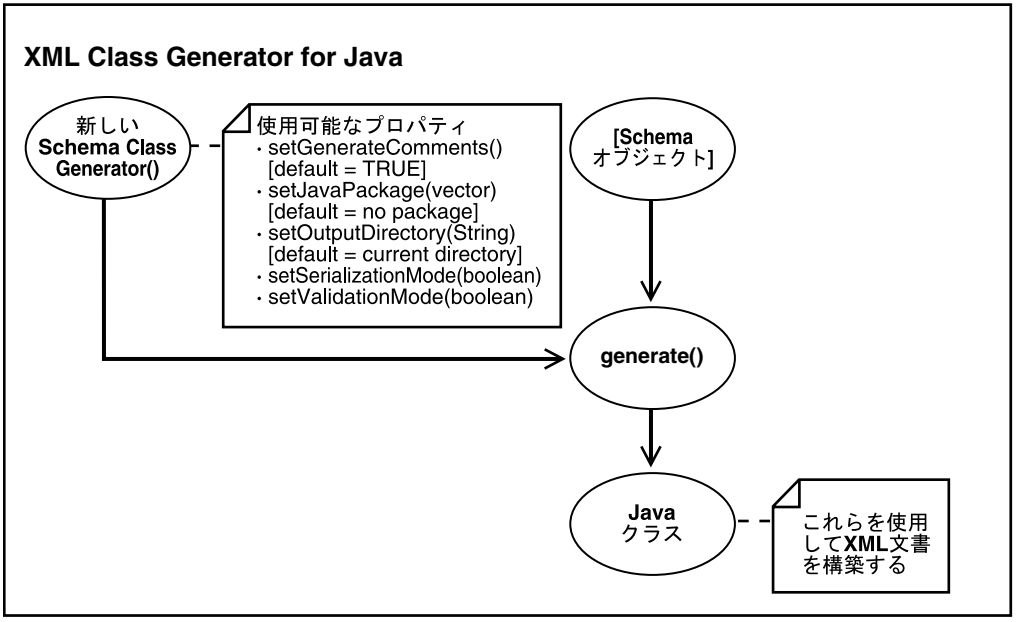
XML Schema を使用した XML Class Generator for Java の動作は次のとおりです。

1. 新しい `SchemaClassGenerator()` クラスが開始され、`generate()` メソッドをコールします。使用可能な `SchemaClassGenerator()` クラスのプロパティは次のとおりです。
   - `setGeneraterComments()`（デフォルト = TRUE）
   - `setJavaPackage(string)`（デフォルト = パッケージ指定なし）
   - `setOutputDirectory(string)`（デフォルト = 現在のディレクトリ）
2. XML Schema を使用する場合、`parseSchema()` メソッドから `getDocType()` を使用して戻される Schema オブジェクトも入力されます。20-20 ページの図 20-4「XML Parser for Java: DOMParser()」を参照してください。
3. `generate()` メソッドが Java クラスを生成します。この Java クラスは、XML 文書を構築するために使用できます。

XML Class Generator for Java で XML Schema を使用してクラスを生成するには、次の項で説明するガイドラインに従います。

- 22-6 ページの「最上位要素のクラスの生成」
- 22-7 ページの「最上位 ComplexType 要素のクラスの生成」
- 22-7 ページの「SimpleType 要素のクラスの生成」

図 22-2　Class Generator for Java および XML Schema を使用したクラスの生成



最上位要素のクラスの生成

XML Schema Class Generator for Java を使用して最上位要素のクラスを生成する場合のガイドラインを次に示します。

- 要素名に対応するクラスは、名前空間に対応したパッケージ内に生成されます。
- 要素には、要素クラス内の要素の型を設定する `setType` というメソッドがあります。競合を回避するために、`setType` は解決済の完全なパッケージ名を取ります。
- 要素にインラインの `simpleType` または `complexType` がある場合、要素クラス内に、`simpleType`/`complexType` 内で指定するすべての規則に従う静的パブリック・クラスが作成されます。静的パブリック・クラスの名前は、Type という接尾辞がついた要素名です。たとえば、要素名が `PurchaseOrder` であり、PurchaseOrder がインラインの `complexType` 定義を持つ場合、静的パブリック・インナー・クラスの名前は `PurchaseOrder_Type` になります。
- 接尾辞として「Type」を使用するクラス名は、要素と complexType の間で競合します。
- 要素名および名前空間は、（シリアル化および妥当性の検証に使用する）要素クラスの内部に格納されます。

- XMLSchema オブジェクトを受け取り検証するには、検証メソッドを要素内に指定します。
- ノードを出力するには、出力メソッドを要素内に指定します。

最上位 ComplexType 要素のクラスの生成

XML Schema Class Generator for Java を使用して最上位 complexType 要素のクラスを生成する場合のガイドラインを次に示します。

- complexType 要素が最上位要素の場合、クラスはその名前空間に対応するパッケージ内に生成されます。complexType 要素がベース型の要素を拡張する場合、その complexType 要素に対応したクラスもベース型の要素を拡張します。それ以外の場合は、CGXSDElement のクラスを拡張します。
- クラスは、属性に対応したフィールドを含みます。これらのフィールドは保護付きで作成され、サブタイプからアクセスできます。フィールドは、ベース型で出現しない属性に対してのみ追加されます。
- クラスには、属性を指定および取得するメソッドが含まれます。
- 各ローカル要素に対して、最上位の要素と同様の静的パブリック・クラスが作成されます。ただし、この静的パブリック・クラスは complexType クラス内に作成されます。

SimpleType 要素のクラスの生成

XML Schema Class Generator for Java を使用して最上位の simpleType 要素のクラスを生成する場合のガイドラインを次に示します。

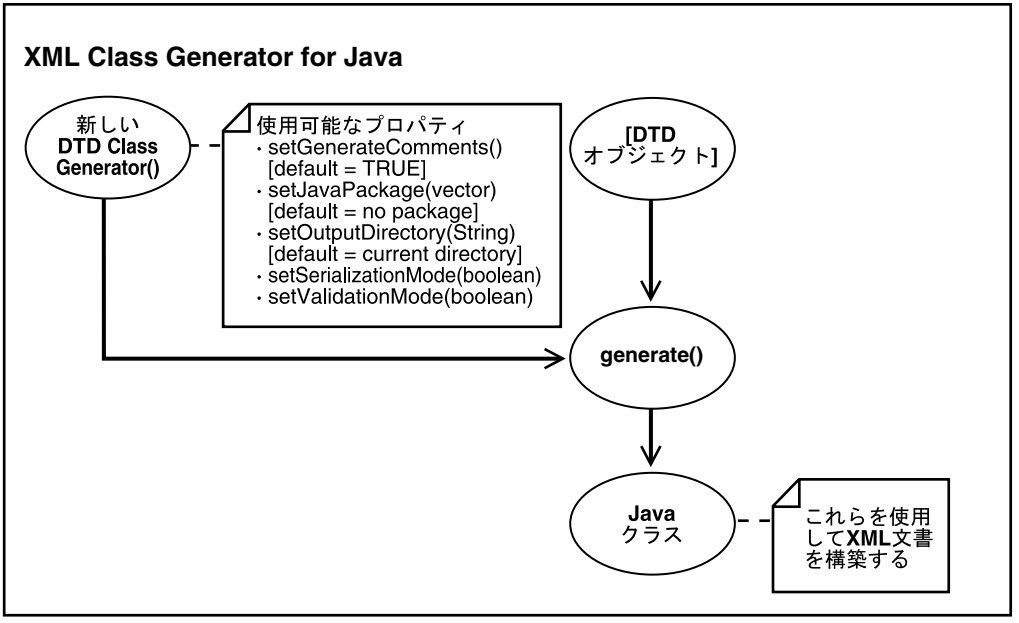
- 最上位の simpleType 要素ごとに 1 つのクラスが生成されます。
- simpleType 要素の階層は、生成されたクラス内に保持されます。simpleType 要素がベース・クラスを拡張する場合、その simpleType 要素に対応したクラスもベースの要素と一致するベース・クラスを拡張します。それ以外の場合は、simpleType 要素は CGXSDElement クラスを拡張します。
- simpleType 要素がスキーマのデータ型を拡張する場合、クラスもそのスキーマのデータ型に対応するクラスを拡張します。たとえば、ベース型が文字列の場合、そのスキーマと同等のクラスは XSDStringType になります。
- クラスには、simpleType の値を格納するフィールドが含まれます。
- simpleType 要素のクラスのコンストラクタは、スキーマのファセットを設定します。
- コンストラクタは、ファセットに対する検証の後に、simpleType のデータ値（XSDDataValue）をコンストラクタ内に設定します。

DTD を使用した XML Class Generator for Java の使用

図 22-3 に、DTD を使用した XML Java Class Generator のコール順序を示します。

1. 新しい `DTDClassGenerator()` クラスが開始され、`generate()` メソッドをコールします。使用可能な `DTDClassGenerator()` クラスのプロパティは次のとおりです。
   - `setGeneraterComments()`（デフォルト = TRUE）
   - `setJavaPackage(string)`（デフォルト = パッケージ指定なし）
   - `setOutputDirectory(string)`（デフォルト = 現在のディレクトリ）
2. DTD を使用する場合、`parseDTD()` メソッドから `getDocType()` を使用して戻される DTD オブジェクトも入力されます。20-20 ページの図 20-4「XML Parser for Java: DOMParser()」を参照してください。
3. `generate()` メソッドが Java クラスを生成します。この Java クラスは、XML 文書を構築するために使用できます。

図 22-3 XML Class Generator for Java および DTD を使用したクラスの生成



参照： 次の項、付録およびマニュアルを参照してください。

- 3-17 ページの「Oracle の XML コンポーネントを使用した XML 文書の生成 : Java」
- 付録 C「XDK for Java: 仕様および早見表」
- 『Oracle9*i* XML リファレンス』

DTD および XML Schema を使用した XML Class Generator の使用例

表 22-2 に、$ORACLE_HOME で提供されているサンプル・ファイルおよびディレクトリを示します。

表 22-2 XML Class Generator for Java のサンプル・ファイル

サンプル・ファイル	説明
MakeFile	UNIX でデモをコンパイルおよび実行するための Make ファイル
Make.bat	Windows でデモをコンパイルおよび実行するための Make ファイル
SampleMain.java	DTD に基づいて Java ソース・ファイルを生成するためのサンプル・アプリケーション
Widl.dtd	サンプル DTD
Widl.xml	Widl.dtd に基づくサンプル XML ファイル
TestWidl.java	SampleMain によって生成された Java ソース・ファイルを使用して、XML 文書を構成するためのサンプル・アプリケーション
car.xsd	サンプル XML Schema
CarDealer.java	car.xsd から生成された Java ソースを使用して、XML 文書を構成するためのサンプル・アプリケーション
book.xsd	サンプル XML Schema
BookCatalogue.java	book.xsd から生成された Java ソースを使用して、XML 文書を構成するためのサンプル・アプリケーション
po.xsd	サンプル XML Schema
TestPo.java	po.xsd から生成された Java ソースを使用して、XML 文書を構成するためのサンプル・アプリケーション

XML Class Generator for Java の実行 - DTD の例

XML Class Generator for Java の DTD のサンプル・プログラムを実行するには、次のコマンドを使用します。

```
make 'dtd'
```

このスクリプトにより、次の手順が実行されます。

1. SampleMain をコンパイルおよび実行して、Java ソース・ファイルを生成します。次のコマンドを使用します。

   ```
   javac SampleMain.java
   java SampleMain -root WIDL Widl.dtd
   ```

   または

   ```
   java SampleMain Widl.xml
   ```

2. 「classgen.jar」、「xmlparser.jar」および現在のディレクトリを含むように CLASSPATH を設定します。
3. SampleMain で生成された Java ソース・ファイルをコンパイルします。コンパイルするファイルは、CONDITION.java、REGION.java、SERVICE.java、VARIABLE.java および WIDL.java です。次のコマンドを使用します。

   ```
   javac *.java
   ```

4. テスト・アプリケーションを実行し、次のコマンドを使用して XML 文書を出力します。

   ```
   javac TestWidl.java
   java TestWidl
   ```

   出力は、Widl_out.txt に格納されます。

XML Class Generator for Java の実行 - XML Schema の例

XML Class Generator for Java の Schema のサンプル・プログラムを実行するには、次のコマンドを使用します。

```
make 'schema'
```

Schema のサンプルには、car.xsd、book.xsd および po.xsd の 3 種類があります。

クラスは、`oracg` ユーティリティを使用して生成されます。たとえば、car.xsd に対応するクラスは次のコマンドラインから生成されます。

```
oracg -c -s car.xsd -p package1
```

この場合、クラスは package1 ディレクトリ内に生成されます。

`Makefile` を使用して Schema Class Generator のデモを実行する場合のガイドラインを次に示します。

- car.xsd に対応するクラスは、package1 ディレクトリ内に生成されます。デモ・プログラム CarDealer.java が、生成されたクラスをテストします。CarDealer.java の出力は、car_out.txt ファイルに格納されます。
- book.xsd に対応するクラスは、package2 ディレクトリ内に生成されます。デモ・プログラム BookCatalogue.java が、生成されたクラスをテストします。出力は、book_out.txt ファイルに格納されます。
- po.xsd に対応するクラスは、package3 ディレクトリ内に生成されます。デモ・プログラム TestPo.java が、生成されたクラスをテストします。出力は、po_out.txt ファイルに格納されます。

DTD を使用した Class Generator の例は次のとおりです。

- XML Class Generator for Java の DTD の例 1a: アプリケーション - SampleMain.java
- XML Class Generator for Java の DTD の例 1b: DTD 入力 - Widl.dtd
- XML Class Generator for Java の DTD の例 1c: 入力 - Widl.xml
- XML Class Generator for Java の DTD の例 1d: TestWidl.java
- XML Class Generator for Java の DTD の例 1e: XML 出力 - Widl.out

XML Class Generator for Java の DTD の例 1a: アプリケーション - SampleMain.java

```
/**
 * This program generates the classes for a given DTD using
 * XML DTD Class Generator. A DTD file or an XML document which is
 * DTD compliant is given as input parameters to this application.
 */

import java.io.File;
import java.net.URL;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.DTD;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.classgen.DTDClassGenerator;

public class SampleMain
{

   public SampleMain()
   {
   }

   public static void main (String args[])
   {
      // Validate the input arguments
      if (args.length < 1)
      {
         System.out.println("Usage: java SampleMain "+
                            "[-root <rootName>] <fileName>");
         System.out.println("fileName\t   Input file, XML document or " +
                            "external DTD file");
         System.out.println("-root <rootName>   Name of the root Element " +
                            "(required if the input file is an external DTD)");
         return ;
      }

      // ty to open the XML Document or the External DTD File
      try
      {
         // Instantiate the parser
         DOMParser   parser = new DOMParser();
         XMLDocument doc    = null;
         DTD         dtd    = null;

         if (args.length == 3)
         {
```

```
            parser.parseDTD(fileToURL(args[2]), args[1]);
            dtd = (DTD)parser.getDoctype();
         }
         else
         {
            parser.setValidationMode(true);
            parser.parse(fileToURL(args[0]));
            doc = parser.getDocument();
            dtd = (DTD)doc.getDoctype();
         }

         String doctype_name = null;

         if (args.length == 3)
         {
            doctype_name = args[1];
         }
         else
         {
            // get the Root Element name from the XMLDocument
            doctype_name = doc.getDocumentElement().getTagName();
         }

         // generate the Java files...
         DTDClassGenerator generator = new DTDClassGenerator();

         // set generate comments to true
         generator.setGenerateComments(true);

         // set output directory
         generator.setOutputDirectory(".");

         // set validating mode to true
         generator.setValidationMode(true);

         // generate java src
         generator.generate(dtd, doctype_name);

      }
      catch (Exception e)
      {
         System.out.println ("XML Class Generator: Error " + e.toString());
         e.printStackTrace();
      }
   }
```

```
   static public URL fileToURL(String sfile)
   {
      File file = new File(sfile);
      String path = file.getAbsolutePath();
      String fSep = System.getProperty("file.separator");
      if (fSep != null && fSep.length() == 1)
         path = path.replace(fSep.charAt(0), '/');
      if (path.length() > 0 && path.charAt(0) != '/')
         path = '/' + path;
      try
      {
         return new URL("file", null, path);
      }
      catch (java.net.MalformedURLException e)
      {
         // According to the spec this could only happen if the file
         // protocol were not recognized.
         throw new Error("unexpected MalformedURLException");
      }
   }
}
```

XML Class Generator for Java の DTD の例 1b: DTD 入力 - Widl.dtd

次の例で、`Widl.dtd` は、SampleMain.java が使用する DTD ファイルです。

```
<!ELEMENT WIDL ( SERVICE | BINDING )* >
<!ATTLIST WIDL
     NAME       CDATA   #IMPLIED
     VERSION (1.0 | 2.0 | ...) "2.0"
     BASEURL    CDATA   #IMPLIED
     OBJMODEL (wmdom | ...) "wmdom"
>

<!ELEMENT SERVICE EMPTY>
<!ATTLIST SERVICE
     NAME       CDATA   #REQUIRED
     URL        CDATA   #REQUIRED
     METHOD (Get | Post) "Get"
     INPUT      CDATA   #IMPLIED
     OUTPUT     CDATA   #IMPLIED
>

<!ELEMENT BINDING ( VARIABLE | CONDITION | REGION )* >
<!ATTLIST BINDING
     NAME       CDATA   #REQUIRED
     TYPE (Input | Output) "Output"
```

```
>

<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE
     NAME       CDATA   #REQUIRED
     TYPE (String | String1 | String2) "String"
     USAGE (Function | Header | Internal) "Function"
     VALUE      CDATA   #IMPLIED
     MASK       CDATA   #IMPLIED
     NULLOK    (True | False) #REQUIRED
>

<!ELEMENT CONDITION EMPTY>
<!ATTLIST CONDITION
     TYPE  (Success | Failure | Retry) "Success"
     REF        CDATA   #REQUIRED
     MATCH      CDATA   #REQUIRED
     SERVICE    CDATA   #IMPLIED
>

<!ELEMENT REGION EMPTY>
<!ATTLIST REGION
     NAME       CDATA   #REQUIRED
     START      CDATA   #REQUIRED
     END        CDATA   #REQUIRED
>
```

XML Class Generator for Java の DTD の例 1c: 入力 - Widl.xml

この XML ファイルは、SampleMain.java を入力します。このファイルは Widl.dtd に基づいています。

```
<?xml version="1.0"?>
<!DOCTYPE WIDL SYSTEM "Widl.dtd">
<WIDL>
    <SERVICE NAME="sname" URL="surl"/>
    <BINDING NAME="bname"/>
</WIDL>
```

XML Class Generator for Java の DTD の例 1d: TestWidl.java

TestWidl.java は、SampleMain.java によって生成された Java ソース・ファイルを使用して XML 文書を構成します。

```
/**
 * This is a sample application program which is built using the
 * classes generated by the XML DTD Class Generator. The External DTD
 * File "Widl.dtd" or the XML document which "Widl.xml" which is compliant
 * to  Widl.dtd is used to generate the classes. The application
 * SampleMain.java is used to generate the classes which takes the DTD
 * or XML document as input parameters to generate classes.
 */

import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.CGDocument;
import oracle.xml.classgen.DTDClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.DTD;

public class TestWidl
{
   public static void main (String args[])
   {
      try
      {
         WIDL w1 = new WIDL();
         DTD dtd = w1.getDTDNode();

         w1.setNAME("WIDL1");
         w1.setVERSION(WIDL.VERSION_1_0);

         SERVICE s1 = new SERVICE("Service1", "Service_URL");
         s1.setINPUT("File");
         s1.setOUTPUT("File");

         BINDING b1 = new BINDING("Binding1");
         b1.setTYPE(BINDING.TYPE_INPUT);

         BINDING b2 = new BINDING("Binding2");
         b2.setTYPE(BINDING.TYPE_OUTPUT);

         VARIABLE v1 = new VARIABLE("Variable1", VARIABLE.NULLOK_FALSE);
         v1.setTYPE(VARIABLE.TYPE_STRING);
         v1.setUSAGE(VARIABLE.USAGE_INTERNAL);
         v1.setVALUE("value");

         VARIABLE v2 = new VARIABLE("Variable2", VARIABLE.NULLOK_TRUE);
```

```
            v2.setTYPE(VARIABLE.TYPE_STRING1);
            v2.setUSAGE(VARIABLE.USAGE_HEADER);

            VARIABLE v3 = new VARIABLE("Variable3", VARIABLE.NULLOK_FALSE);
            v3.setTYPE(VARIABLE.TYPE_STRING2);
            v3.setUSAGE(VARIABLE.USAGE_FUNCTION);
            v3.setMASK("mask");

            CONDITION c1 = new CONDITION("CRef1", "CMatch1");
            c1.setSERVICE("Service1");
            c1.setTYPE(CONDITION.TYPE_SUCCESS);

            CONDITION c2 = new CONDITION("CRef2", "CMatch2");
            c2.setTYPE(CONDITION.TYPE_RETRY);

            CONDITION c3 = new CONDITION("CRef3", "CMatch3");
            c3.setSERVICE("Service3");
            c3.setTYPE(CONDITION.TYPE_FAILURE);

            REGION r1 = new REGION("Region1", "Start", "End");

            b1.addNode(r1);
            b1.addNode(v1);
            b1.addNode(c1);
            b1.addNode(v2);

            b2.addNode(c2);
            b2.addNode(v3);

            w1.addNode(s1);
            w1.addNode(b1);
            w1.addNode(b2);
            w1.validateContent();
            w1.print(System.out);
         }
         catch (Exception e)
         {
            System.out.println(e.toString());
            e.printStackTrace();
         }
      }
   }
```

XML Class Generator for Java の DTD の例 1e: XML 出力 - Widl.out

この XML ファイル Widl.out は、TestWidl.java によって構成および出力されます。

```
<?xml  version = '1.0' encoding = 'ASCII'?>
<!DOCTYPE WIDL SYSTEM "file:/oracore/java/xml/ORACORE_MAIN_SOLARIS_990115_
XMLCLASSGEN/sample/out/WIDL.dtd">
<WIDL NAME="WIDL1" VERSION="1.0">
   <SERVICE NAME="Service1" URL="Service_URL" INPUT="File" OUTPUT="File"/>
   <BINDING NAME="Binding1" TYPE="Input">
      <REGION NAME="Region1" START="Start" END="End"/>
      <VARIABLE NAME="Variable1" NULLOK="False" TYPE="String" USAGE="Internal"
VALUE="value"/>
      <CONDITION REF="CRef1" MATCH="CMatch1" SERVICE="Service1" TYPE="Success"/>
      <VARIABLE NAME="Variable2" NULLOK="True" TYPE="String1" USAGE="Header"/>
   </BINDING>
   <BINDING NAME="Binding2" TYPE="Output">
      <CONDITION REF="CRef2" MATCH="CMatch2" TYPE="Retry"/>
      <VARIABLE NAME="Variable3" NULLOK="False" TYPE="String2" USAGE="Function"
MASK="mask"/>
   </BINDING>
</WIDL>
```

XML Schema を使用した Class Generator の例は次のとおりです。

- XML Class Generator for Java の Schema の例 1a: XML Schema - car.xsd
- XML Class Generator for Java の Schema の例 1b: アプリケーション - CarDealer.java
- XML Class Generator for Java の Schema の例 2a: XML Schema - book.xsd
- XML Class Generator for Java の Schema の例 2b: BookCatalogue.java
- XML Class Generator for Java の Schema の例 3a: XML Schema - po.xsd
- XML Class Generator for Java の Schema の例 3b: アプリケーション - TestPo.java

XML Class Generator for Java の Schema の例 1a: XML Schema - car.xsd

サンプル Schema の car.xsd を oracg コマンド内で使用し、クラスを生成します。生成されたクラスがプログラム CarDealer.java を入力し、このプログラムが XML 文書を作成します。次のコマンドを使用します。

```
oracg -c -s car.xsd -p package1
```

使用方法については、次の項を参照してください。

- 22-20 ページの「XML Class Generator for Java の Schema の例 1b: アプリケーション - CarDealer.java」
- 22-11 ページの「XML Class Generator for Java の実行 - XML Schema の例」

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
targetNamespace = "http://www.CarDealers.com/" elementFormDefault="qualified">
<element name="Car">
  <complexType>
    <element name="Model">
      <simpleType base="string">
        <enumeration value = "Ford"/>
        <enumeration value = "Saab"/>
        <enumeration value = "Audi"/>
      </simpleType>
    </element>
    <element name="Make">
       <simpleType base="string">
         <minLength value = "1"/>
         <maxLength value = "30"/>
       </simpleType>
    </element>
    <element name="Year">
       <complexType content="mixed">
          <attribute name="PreviouslyOwned" type="string" use="required"/>
          <attribute name="YearsOwned" type="integer" use="optional"/>
       </complexType>
    </element>
    <element name="OwnerName" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Condition">
       <complexType base="string" derivedBy="extension">
          <attribute name="Automatic">
            <simpleType base="string">
               <enumeration value = "Yes"/>
               <enumeration value = "No"/>
            </simpleType>
          </attribute>
       </complexType>
```

```
    </element>
    <element name="Mileage">
      <simpleType base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="20000"/>
      </simpleType>
    </element>
    <attribute name="RequestDate" type="date"/>
  </complexType>
 </element>
</schema>
```

XML Class Generator for Java の Schema の例 1b: アプリケーション - CarDealer.java

```
/**
 * This is a sample application program that creates an XMl document. It is
 * built using the classes generated by XML Schema Class Generator. XML
 * Schema "car.xsd", is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package1 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s car.xsd -p package1
 */

import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import package1.*;

public class CarDealer
{
   static OutputStream output  = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main(String args[])
   {
      CarDealer cardealer = new CarDealer();
      try
      {
         Car.Car_Type ctype = new Car.Car_Type();
         ctype.setRequestDate("02-09-00");
         Car.Car_Type.Model model = new Car.Car_Type.Model();
```

```
Car.Car_Type.Model.Model_Type modelType =
         new Car.Car_Type.Model.Model_Type("Ford");
model.setType(modelType);
ctype.addModel(model);

Car.Car_Type.Make make = new Car.Car_Type.Make();
Car.Car_Type.Make.Make_Type makeType =
         new Car.Car_Type.Make.Make_Type("F150");
make.setType(makeType);
ctype.addMake(make);

Car.Car_Type.Year year = new Car.Car_Type.Year();
Car.Car_Type.Year.Year_Type yearType =
        new Car.Car_Type.Year.Year_Type();
yearType.addText("1999");

year.setType(yearType);
ctype.addYear(year);

Car.Car_Type.OwnerName owner1 = new Car.Car_Type.OwnerName();
owner1.setType("Joe Smith");
ctype.addOwnerName(owner1);

Car.Car_Type.OwnerName owner2 = new Car.Car_Type.OwnerName();
owner2.setType("Bob Smith");
ctype.addOwnerName(owner2);

String str = "Small dent on the car's right bumper.";
Car.Car_Type.Condition condition = new Car.Car_Type.Condition();
Car.Car_Type.Condition.Condition_Type conditionType =
        new Car.Car_Type.Condition.Condition_Type(str);

Car.Car_Type.Condition.Condition_Type.Automatic automatic =
      new Car.Car_Type.Condition.Condition_Type.Automatic("Yes");
conditionType.setAutomatic(automatic);

condition.setType(conditionType);
ctype.addCondition(condition);

Car.Car_Type.Mileage mileage = new Car.Car_Type.Mileage();
Car.Car_Type.Mileage.Mileage_Type mileageType =
        new Car.Car_Type.Mileage.Mileage_Type("10000");
mileage.setType(mileageType);
ctype.addMileage(mileage);

Car car = new Car();
car.setType(ctype);
```

```
            car.print(out);

            out.writeNewLine();
            out.flush();
         }
         catch(InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch(Exception e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
      }
}
```

XML Class Generator for Java の Schema の例 2a: XML Schema - book.xsd

サンプル Schema の book.xsd を oracg コマンド内で使用し、クラスを生成します。生成されたクラスがプログラム CarDealer.java を入力し、このプログラムが XML 文書を作成します。oracg コマンドは次のとおりです。

```
oracg -c -s book.xsd -p package2
```

使用方法については、次の項を参照してください。

- 22-23 ページの「XML Class Generator for Java の Schema の例 2b: BookCatalogue.java」
- 22-11 ページの「XML Class Generator for Java の実行 - XML Schema の例」

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/BookCatalogue"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue"
        elementFormDefault="qualified">

    <complexType name="Pub">
      <sequence>
        <element name="Title" type="cat:titleType" maxOccurs="*"/>
        <element name="Author" type="string" maxOccurs="*"/>
        <element name="Date" type="date"/>
      </sequence>
      <attribute name="language" type="string" use="default" value="English"/>
    </complexType>
```

```
    <complexType name="titleType" base="string" derivedBy="extension">
      <attribute name="old" type="string" use="default" value="false"/>
    </complexType>

    <element name="Catalogue" type="cat:Pub"/>
</schema>
```

XML Class Generator for Java の Schema の例 2b: BookCatalogue.java

```
/**
 * This is a sample application program built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "book.xsd" is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package2 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s book.xsd -p package2
 */

import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import package2.*;

public class BookCatalogue
{
   static OutputStream output  = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main(String args[])
   {
      BookCatalogue bookCatalogue = new BookCatalogue();
      try
      {
         Pub pubType = new Pub();

         TitleType titleType = new TitleType("Natural Health");
         titleType.setOld("true");

         Pub.Title title = new Pub.Title();
         title.setType(titleType);
         pubType.addTitle(title);
```

```
            Pub.Author author = new Pub.Author();
            author.setType("Richard> Bach");
            pubType.addAuthor(author);

            Pub.Date date = new Pub.Date();
            date.setType("1977");
            pubType.addDate(date);
            pubType.setLanguage("English");

            Catalogue catalogue = new Catalogue();
            catalogue.setType(pubType);

            catalogue.print(out);
            out.writeNewLine();
            out.flush();
         }
         catch(InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch(Exception e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
      }
   }
```

XML Class Generator for Java の Schema の例 3a: XML Schema - po.xsd

サンプル Schema の po.xsd を oracg コマンド内で使用し、クラスを生成します。生成されたクラスがプログラム TestPo.java を入力し、このプログラムが XML 文書を作成します。使用する oracg コマンドは次のとおりです。

```
oracg -c -s po.xsd -p package3
```

使用方法については、次の項を参照してください。

- 22-26 ページの「XML Class Generator for Java の Schema の例 3b: アプリケーション - TestPo.java」
- 22-11 ページの「XML Class Generator for Java の実行 - XML Schema の例」

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/PurchaseOrder"
```

```
        xmlns:po = "http://www.somewhere.org/PurchaseOrder">

<element name="comment" type="string"/>

<element name="PurchaseOrder">
  <complexType>
      <element name="shipTo" type="po:Address"/>
      <element name="billTo" type="po:Address"/>
      <element ref="po:comment" minOccurs="0"/>
      <element name="items" type="po:Items"/>
      <attribute name="orderDate" type="date"/>
      <attribute name="shipDate" type="date"/>
      <attribute name="receiveDate" type="date"/>
  </complexType>
</element>

<complexType name="Address">
  <element name="name" type="string"/>
  <element name="street" type="string"/>
  <element name="city" type="string"/>
  <element name="zip" type="decimal"/>
  <attribute name="country" type="NMTOKEN"
      use="fixed" value="US"/>
</complexType>

<complexType name="Items">
   <element name="item" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <element name="productName" type="string"/>
        <element name="quantity" type="int"/>
        <element name="price" type="decimal"/>
        <element name="shipDate" type="date" minOccurs='0'/>
        <attribute name="partNum" type="string"/>
      </complexType>
   </element>
</complexType>

</schema>
```

XML Class Generator for Java の Schema の例 3b: アプリケーション - TestPo.java

```
/**
 * This is a sample application program which is built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "po.xsd" is used to generate the classes using the oracg
 * command line utility. The classes are generated in a package called
 * package3 which is specified as command line option. The following
 * oracg command line options are used to generate the classes:
 * oracg -c -s po.xsd -p package3
 */

import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;
import package3.*;

public class TestPo
{
   static OutputStream output = System.out;
   static XMLOutputStream out = new XMLOutputStream(output);

   public static void main (String args[])
   {
      TestPo testpo = new TestPo();
      try
      {
         // Create Purchase Order
         PurchaseOrder po = new PurchaseOrder();

         // Create Purchase Order Type
         PurchaseOrder.PurchaseOrder_Type poType =
                new PurchaseOrder.PurchaseOrder_Type();

         // Set purchase order date
         poType.setOrderDate("December 17, 2000");
         poType.setShipDate("December 19, 2000");
         poType.setReceiveDate("December 21, 2000");

         // Create a PurchaseOrder shipTo item
         PurchaseOrder.PurchaseOrder_Type.ShipTo shipTo =
                new PurchaseOrder.PurchaseOrder_Type.ShipTo();
```

```
// Create Address
Address address = new Address();

// Create the Name for the address and add
// it to addresss
Address.Name name = new Address.Name();
name.setType("Mary Smith");
address.addName(name);

// Create the Stree name for the address and add
// it to the address
Address.Street street = new Address.Street();
street.setType("Laurie Meadows");
address.addStreet(street);

// Create the city name for the address and add
// it to the address
Address.City city = new Address.City();
city.setType("San Mateo");
address.addCity(city);

// Create the zip name for the address and add
// it to the address
Address.Zip zip = new Address.Zip();
zip.setType(new Double("11208"));
address.addZip(zip);

// Set the address of the shipTo object
shipTo.setType(address);
// Add the shipTo to the Purchase Type object
poType.addShipTo(shipTo);

// Create a Purchase Order BillTo item
PurchaseOrder.PurchaseOrder_Type.BillTo billTo =
      new PurchaseOrder.PurchaseOrder_Type.BillTo();

// Create a billing Address
Address billingAddress = new Address();

// Create the name for billing address, set the
// name and add it to the billing address
Address.Name name1 = new Address.Name();
name1.setType("John Smith");
billingAddress.addName(name1);
```

```
// Create the street name for the billing address,
// set the street name value and add it to the
// billing address
Address.Street street1 = new Address.Street();
street1.setType("No 1. North Broadway");
billingAddress.addStreet(street1);

// Create the City name for the address, set the
// city name value and add it to the billing address
Address.City city1 = new Address.City();
city1.setType("New York");
billingAddress.addCity(city1);

// Create the Zip for the address, set the zip
// value and add it to the billing address.
Address.Zip zip1 = new Address.Zip();
zip1.setType(new Double("10006"));
billingAddress.addZip(zip1);

// Set the type of the billTo object to billingAddress
billTo.setType(billingAddress);

// Add the billing address to the PurchaseOrder type
poType.addBillTo(billTo);

PurchaseOrder.PurchaseOrder_Type.Items pItem =
      new PurchaseOrder.PurchaseOrder_Type.Items();

Items items = new Items();
Items.Item item = new Items.Item();
Items.Item.Item_Type itemType = new Items.Item.Item_Type();

Items.Item.Item_Type.ProductName pname =
    new Items.Item.Item_Type.ProductName();
pname.setType("Perfume");
itemType.addProductName(pname);

Items.Item.Item_Type.Quantity qty =
    new Items.Item.Item_Type.Quantity();
qty.setType(new Integer("1"));
itemType.addQuantity(qty);

Items.Item.Item_Type.Price price =
    new Items.Item.Item_Type.Price();
price.setType(new Double("69.99"));
itemType.addPrice(price);
```

```
            Items.Item.Item_Type.ShipDate sdate =
                new Items.Item.Item_Type.ShipDate();
            sdate.setType("Feb 14. 2000");
            itemType.addShipDate(sdate);

            itemType.setPartNum("ITMZ411");

            item.setType(itemType);
            items.addItem(item);

            pItem.setType(items);

            poType.addItems(pItem);

            // Set the type of the Purchase Order object to
            // Purchase Order Type
            po.setType(poType);
            po.print(out);

            out.writeNewLine();
            out.flush();
         }
         catch (InvalidContentException e)
         {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
         catch (Exception e)
         {
            System.out.println(e.toString());
            e.printStackTrace();
         }
      }
   }
```

FAQ: Class Generator for Java

この項では、XML Class Generator for Java についての質問および回答を示します。

XML Class Generator のインストール方法

質問

XML Class Generator のインストール方法を教えてください。

回答

XML Class Generator は、XDK の一部としてパッケージ化されてるため、個別にダウンロードする必要はありません。classgen.jar、xmlparserv2.jar および xschema.jar を CLASSPATH に含むように設定してください。これらは、bin/ ディレクトリ内ではなく lib/ ディレクトリ内にあります。

XML Class Generator for Java の役割

質問

XML Class Generator for Java の役割を教えてください。XML Class Generator for Java を使用した XML データの取得方法を教えてください。

回答

XML Class Generator for Java は、XML DTD から Java ソース・ファイルを作成します。これは、アプリケーションが DTD に従って、または構成する Web フォームおよび XML 文書のバックエンドとして他のアプリケーションに XML メッセージを送信する必要がある場合に有効です。Java アプリケーションは、これらのクラスを使用して、入力用 DTD に準拠する XML 文書を構成、検証および出力できます。この Class Generator は、Oracle XML Parser for Java と連携して機能します。XML Parser for Java は、DTD を解析し、解析済文書を Class Generator に渡します。

XML データを取得するには、まず、JDBC の結果セットを使用してデータベースからデータを取得します。次に、XML Class Generator によって生成されたクラスを使用して、オブジェクトをインスタンス化します。

DTD のサポート

質問

XML Class Generator for Java は、すべての種類の DTD をサポートしますか？

回答

はい。XML Class Generator for Java は、XML 1.0 に準拠するすべての種類の DTD をサポートします。

XML Class Generator サンプル実行中のエラー

質問

「クラスが見つかりません」というエラーの修正方法を教えてください。

回答

classgen.jar、xmlparserv2.jar および xschema.jar を CLASSPATH に含めてください。

XML Class Generator: 2 回以上のルート・オブジェクトの作成

質問

Class Generator を使用して、DTD から Java クラスのセットを生成しました。その後、引数として渡されたデータから XML ファイルを作成するために、これらのクラスを使用する Java アプリケーションの作成を試みました。次のエラー・メッセージが表示されるため、CGDocument から導出されたオブジェクトであるルート・オブジェクトを 2 回以上作成できません。

```
oracle.xml.parser.XMLDOMException: ノードがカレント・ドキュメントに属していません。
```

スター演算子（*）の処理方法を教えてください。アプリケーション起動時には、この要素が何回現れるかはわかりません。このため、element.addNode() のシーケンスを作成するための静的ループは構築していません。問題は、これらの一部が空になり、空の属性を持つ空の要素のセットを含む XML 文書を取得することです。

回答

毎回コンストラクタをコールすることによって、後続の XML 文書を作成できます。整形式の XML 文書は、複数のルート・ノードを持つことができません。このため、文書ルートとして指定する要素に対しては、「*」を使用できません。

DOM API を使用した XML ファイルの新規作成

質問

DOM API を使用して XML ファイルを作成する必要があります。テキスト・エディタで入力しないで、次の XML ファイルを作成する必要があります。

```
<xml>
  <future>is great</future>
</xml>
```

かわりに、DOM API を使用して XML ファイルを作成する方法を教えてください。入力ファイルがある場合に、DOM を使用して XML ファイルを操作する例はありますが、入力ファイルがなく、タグ名およびその値がわかっている場合に、DOM を使用して XML ファイルを最初から作成するという例はありません。

回答

最も簡単な方法は、XML Class Generator for Java をダウンロードし、その Class Generator に、作成する必要がある XML 文書の DTD を指定することです。この Class Generator は、DOM クラスを作成し、プログラムによって XML 文書を作成します。ソフトウェアに付属のサンプルがあります。

Java クラス内への XML 文書の作成

質問

次の Java クラス内に XML 文書を作成する必要があります。

```
<?xml version = '1.0' encoding = 'WINDOWS-1252'?>
    <root>
     <listing>
       <one> test </one>
       <two> test </two>
     </listing>
    </root>
```

XMLDocument クラスを使用して XML 文書を作成できますか。XSU は使用できますが、このユーティリティは SQL 問合せに基づく XML を作成するのみで、今回の状況には適用できません。何か方法例がありますか。

回答

XML Class Generator でこの作業が可能です。この XML Class Generator は、Oracle XDK for Java の一部として http://otn.oracle.com/tech/xml から入手できます。XDK は、Oracle および OracleAS 製品にも付属しています。Class Generator は、DTD 内の各要素に対して Java クラスを作成します。これらのクラスを使用して、実行時に直接 XML 文書を作成できます。ダウンロードした Class Generator にはサンプル・コードが付属しています。

第 VII 部

XDK for JavaBeans

第 VII 部では、XDK for JavaBeans の使用方法を説明します。

第 VII 部に含まれる章は、次のとおりです。

23

XML Transviewer Beans の使用

この章の内容は次のとおりです。

Oracle XML Transviewer Beans の入手方法

Oracle XML Transviewer Beans は、Oracle Enterprise Edition および Oracle Standard Edition のリリース 1（9.0.1）に、XDK for JavaBeans の一部として付属しています。最新の XDK for JavaBeans は、OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードできます。

XDK for Java: XML Transviewer Beans の機能

XML Transviewer Beans を使用すると、XML アプリケーションにグラフィカルまたはビジュアル的なインタフェースを追加できます。

JDeveloper からの直接アクセス

Bean のカプセル化には、JDeveloper などの Java 統合開発環境から直接アクセスできるドキュメントおよび記述子が含まれます。

Transviewer Bean のサンプル・アプリケーション

ソフトウェアには、すべての Bean を使用して簡単な XML エディタおよび XSLT による変換を行うサンプル・アプリケーションが含まれています。

XSU に含まれるこのサンプル・アプリケーションは、次のタスクを行います。

- データベースを問い合せて、XML を生成します。
- XSL スタイルシートを使用して、XML を変換します。
- 結果の XML 文書を高速に取り出すために、データベースに格納します。

データベースへの接続性

XML Transviewer Beans にはデータベースへの接続性もあります。XML Transviewer Beans は JDBC 対応のデータベースに直接接続して、XML ファイルや XSL ファイルを取出しおよび格納できます。

XML Transviewer Beans

XML Transviewer Beans は、次の 7 つの Bean で構成されています。

DOMBuilder Bean

DOMBuilder Bean は、ビジュアル的ではない Bean です。この Bean は、XML 文書から DOM ツリーを構築します。

DOMBuilder Bean は、Bean インタフェースを使用して XML Parser for Java の DOMParser クラスをカプセル化し、その機能を拡張して非同期解析を可能にします。リスナーを登録すると、Java アプリケーションは大規模または連続したドキュメントを解析し、制御をすぐにコール元に戻すことができます。23-5 ページの「DOMBuilder Bean の使用」を参照してください。

XSLTransformer Bean

XSLTransformer Bean は、ビジュアル的ではない Bean です。この Bean は XML ファイルを受け入れ、入力の XSL スタイルシートによって指定された変換を適用し、結果の出力ファイルを作成します。

XSLTransformer Bean を使用すると、適切な XSL スタイルシートを適用することによって、XML、HTML、DDL などのほぼすべてのテキスト・フォーマットに XML 文書を変換できます。

- この Bean を他の Bean と統合した場合、アプリケーションまたはユーザーは、変換の結果をすぐに表示できます。
- XSLTransformer Bean は、サーバー側のアプリケーションまたはサーブレットが XML 文書（問合せ結果の XML 表現など）をブラウザで表示するために HTML でレンダリングするための基礎として使用できます。

23-9 ページの「XSLTransformer Bean の使用」を参照してください。

XMLTreeViewer Bean

XMLTreeViewer Bean は、XML 書式のファイルをツリーとして図形で表示します。このツリーのブランチおよびリーフはマウスで操作できます。23-13 ページの「XMLTreeViewer Bean の使用」を参照してください。

XMLSourceViewer Bean

XMLSourceViewer Bean は、ビジュアル的な JavaBean です。XMLSourceViewer Bean を使用すると、XML 文書をビジュアルに編集できます。XMLSourceViewer Bean では、XML 文書を編集アプリケーションで修正するときに、XML および XSL 書式のファイルの構文を色でハイライトして表示します。これによって、ファイルの参照および編集が容易になります。この Bean は DOMBuilder Bean と容易に統合でき、指定された DTD に対する解析前または解析後のビジュアル化を可能にします。23-16 ページの「XMLSourceViewer Bean の使用」を参照してください。

XMLTransformPanel Bean

XMLTransformPanel Bean はビジュアル的な JavaBean です。XMLTransformPanel Bean を使用すると、XML 文書に XSL 変換を適用し、その結果を表示できます。これによって、XML 入力ファイルおよび XSL 入力ファイルを編集できます。23-20 ページの「XMLTransformPanel Bean の使用」を参照してください。

DBViewer Bean

DBViewer Bean はビジュアル的な JavaBean です。DBViewer Bean を使用すると、XSL スタイルシートを適用し、結果の HTML をビジュアル化して、スクロール可能な Swing パネル内にデータベース問合せまたは XML を表示することができます。DBViewer Bean には、XML ツリー・バッファおよび XSL ツリー・バッファの他に、結果バッファもあります。DBViewer Bean を使用すると、コール元のプログラムは次の操作が可能になります。

- Oracle データベースの CLOB 表やファイル・システムなどの、様々なソースからバッファをロードまたは保存できます。ファイル・システムとデータベースのユーザー・スキーマの間で、ファイルを移動させるための制御も使用できます。
- XSL バッファのスタイルシートを使用し、XML バッファにスタイルシート変換を適用できます。

結果は、結果バッファに格納されます。XML バッファおよび XSL バッファの内容は、ソースまたはツリー構造として表示されます。結果バッファの内容も、HTML としてレンダリングし、ソースまたはツリー構造として表示できます。XML バッファは、データベース問合せからロードできます。

DBAccess Bean

DBAccess Bean は、複数の XML 文書およびテキスト・ドキュメントを保持する CLOB 表をメンテナンスします。

XML Transviewer Beans の使用

XML Transviewer Beans を使用する場合のガイドラインは、次の項を参照してください。

- DOMBuilder Bean の使用
- XSLTransformer Bean の使用
- XMLTreeViewer Bean の使用
- XMLSourceViewer Bean の使用
- XMLTransformPanel Bean の使用
- DBViewer Bean の使用
- DBAccess Bean の使用

参照： 次の章およびマニュアルを参照してください。

- 『Oracle9*i* XML リファレンス』
- 第 3 章「XDK および XML コンポーネント : 概要および一般的な FAQ」

DOMBuilder Bean の使用

DOMBuilder() クラスは、W3C の勧告に準拠した XML 1.0 のパーサーを実装しており、XML 文書を解析して DOM ツリーを構築します。

解析は別々のスレッドで行われ、ツリーが構築されたときの通知には DOMBuilderListener インタフェースが使用される必要があります。

バックグラウンドでの非同期解析への使用

DOMBuilder Bean は、Bean インタフェースを使用して、XML Parser for Java をカプセル化します。DOMBuilder Bean は、機能を拡張して非同期解析を行うことができます。Java アプリケーションは、リスナーを登録することによってドキュメントを解析し、コール元に制御を戻すことができます。

バックグラウンド・スレッドでの非同期解析は、ビジュアル的なアプリケーションで対話式で使用されます。たとえば、通常のパーサーで大きいファイルを解析する場合、解析が完了するまでユーザー・インタフェースが操作不可能になることがあります。DOMBuilder Bean を使用すると、これを回避できます。DOMBuilder Bean の解析メソッドをコールすると、制御がすぐにアプリケーションに戻され、「Parsing, please wait」というメッセージが表示されます。「Cancel」ボタンがある場合は操作を取り消すこともできます。バックグラウンドの解析タスクが完了したときに `domBuilderOver()` メソッドが DOMBuilder Bean によってコールされた場合でも、アプリケーションは続行できます。

DOMBuilder Bean による多くのファイルの高速解析

多くのファイルを解析する場合、DOMBuilder Bean を使用すると時間を大幅に短縮できます。ファイルを 1 つずつ解析する場合と比較すると、最大で 40% も高速になります。

DOMBuilder Bean の使用方法

図 23-1 に、DOMBuilder Bean の使用方法を示します。

1. 解析される XML 文書は、ファイル、文字列バッファまたは URL として入力されます。
2. 前述の入力によって、`DOMBuilder.addDOMBuilderListener(DOMBuilderListener)` メソッドが入力されます。これによって DOMBuilderListener が追加されます。
3. DOMBuilder.parse() メソッドが XML 文書を解析します。
4. オプションで、解析済の結果にその他の処理を行うことができます。適用する使用可能なメソッドのリストは、表 23-1 を参照してください。
5. DOMBuilderListener API は、`DOMBuilderOver()` メソッドを使用してコールされます。DOMBuilderListener API は、アプリケーションから非同期コールを受信したときにコールされます。このインタフェースは、非同期解析中のイベントに関する通知を受信するために実装される必要があります。このインタフェースを実装しているクラスは、`addDOMBuilderListener` メソッドを使用して DOMBuilder に追加される必要があります。

   使用可能な DOMBuilderListener メソッドは次のとおりです。

   - `domBuilderError(DOMBuilderEvent)`
     このメソッドは、解析エラーが発生した場合にコールされます。
   - `domBuilderOver(DOMBuilderEvent)`
     このメソッドは、解析が完了したときにコールされます。
   - `domBuilderStarted(DOMBuilderEvent)`
     このメソッドは、解析が開始したときにコールされます。
6. `DOMBuilder.getDocument()` は結果の DOM 文書をフェッチし、DOM 文書を出力します。

図 23-1 DOMBuilder Bean の使用方法

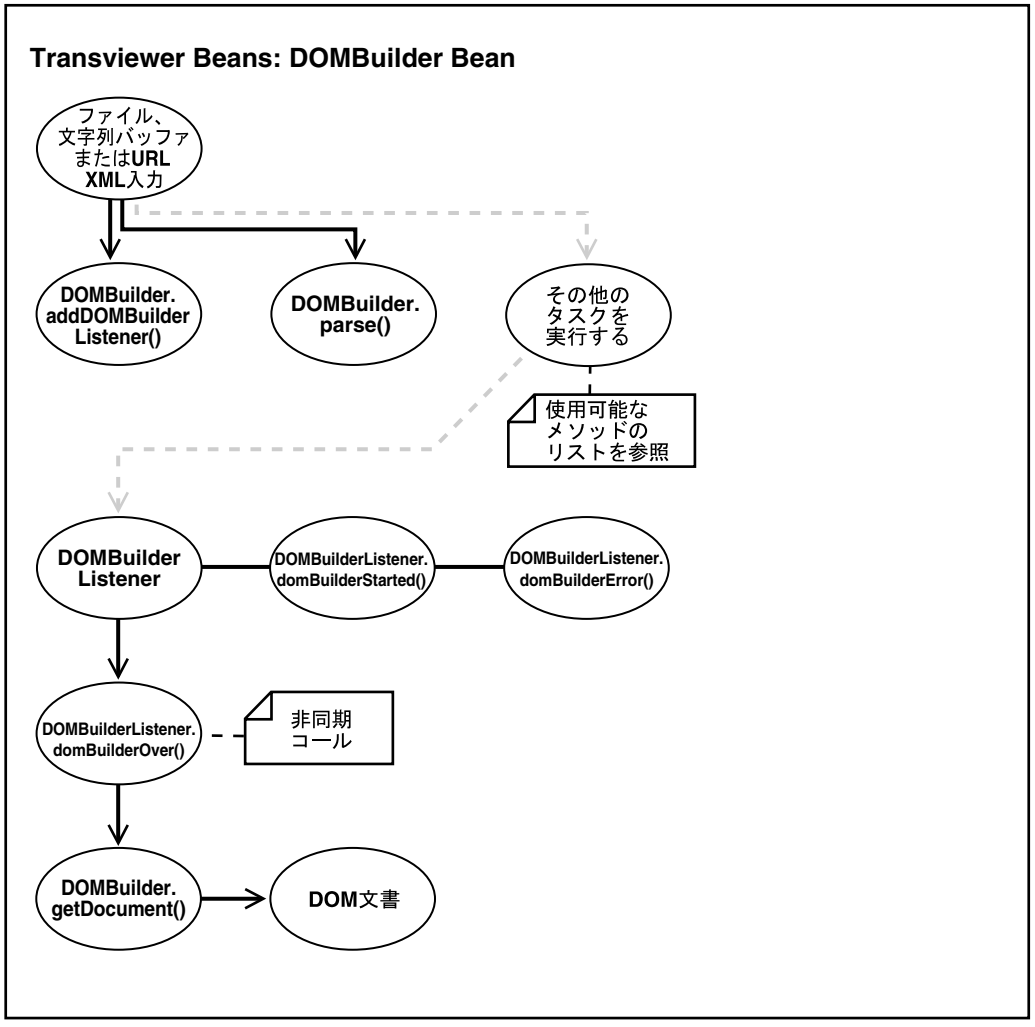


表 23-1 DOMBuilder Bean: メソッド

メソッド	説明
addDOMBuilderErrorListener(DOMBuilderErrorListener)	DOMBuilderErrorListener を追加します。
addDOMBuilderListener(DOMBuilderListener)	DOMBuilderListener を追加します。
	DTD を取得します。
getDocument()	ドキュメントを取得します。
getId()	パーサーのオブジェクト ID を戻します。
getReleaseVersion()	Oracle XML Parser のリリース・バージョンを戻します。
	ドキュメントを取得します。
getValidationMode()	検証モードの設定を戻します。
parse(InputSource)	指定された入力ソースからの XML を解析します。
	指定された入力ストリームからの XML 外部 DTD を解析します。
parse(Reader)	指定された入力ストリームからの XML 外部 DTD を解析します。
parse(String)	指定された URL からの XML 外部 DTD を解析します。
parse(URL)	指定された URL が指す XML 外部 DTD 文書を解析し、それに対応する XML 文書の階層を作成します。
parseDTD(InputSource, String)	指定した入力ソースから XML 外部 DTD を解析します。
parseDTD(InputStream, String)	指定した入力ストリームから XML 外部 DTD を解析します。
parseDTD(Reader, String)	指定した入力ストリームから XML 外部 DTD を解析します。
	指定した URL から XML 外部 DTD を解析します。
parseDTD(URL, String)	指定した URL が指す XML 外部 DTD を解析し、対応する XML 文書の階層を作成します。
removeDOMBuilderErrorListener(DOMBuilderErrorListener)	DOMBuilderErrorListener を削除します。

表 23-1 DOMBuilder Bean: メソッド（続き）

メソッド	説明
removeDOMBuilderListener(DOMBuilder Listener)	DOMBuilderListener を削除します。
run()	スレッドで実行します。
	外部エンティティおよび DTD をロードするためのベース URL を設定します。
setDebugMode(boolean)	ドキュメントのデバッグ情報を有効にするフラグを設定します。
setDoctype(DTD)	DTD を設定します。
setErrorStream(OutputStream)	エラーおよび警告を出力するための出力ストリームを設定します。
setErrorStream(OutputStream, String)	エラーおよび警告を出力するための出力ストリームを設定します。
setErrorStream(PrintWriter)	エラーおよび警告を出力するための出力ストリームを設定します。
	ノード・ファクトリを設定します。
setPreserveWhitespace(boolean)	空白保持モードを設定します。
setValidationMode(boolean)	検証モードを設定します。
showWarnings(boolean)	警告を出力するかどうかを決定します。

XSLTransformer Bean の使用

XSLTransformer Bean は XML ファイルを受け入れ、入力 XSL スタイルシートによって指定された変換を適用して、ファイルを作成および出力します。この Bean を使用すると、XSL スタイルシートを適用することによって、XML、HTML、DDL などのほぼすべてのテキストベースの形式に XML 文書を変換できます。

この Bean を他の Bean と統合した場合、アプリケーションまたはユーザーは、変換の結果をすぐに表示できます。

XSLTransformer Bean は、サーバー側のアプリケーションまたはサーブレットが XML 文書（問合せ結果の XML 表現など）をブラウザで表示するために HTML でレンダリングするための基礎として使用できます。

XSLTransformer Bean は、Bean インタフェースを使用して XML Parser for Java の XSLT 処理エンジンをカプセル化し、その機能を拡張して非同期変換を許可します。

Java アプリケーションは、リスナーを登録することによって、コール元にすぐに制御を返し、大規模または連続したドキュメントを解析できます。

多くのファイルの変換に最適な XSLTransformer Bean

XSL 変換は時間がかかる場合があります。多くのファイルを変換するアプリケーションでは XSLTransformer Bean を使用してください。この Bean は、複数のファイルを同時に変換できます。

即時応答が可能なユーザー・インタフェースを提供する XSLTransformer Bean

XSLTransformer Bean をビジュアル的なアプリケーションに使用すると、即時応答が可能なユーザー・インタフェースが提供されます。この場合、DOMBuilder Bean と同様の問題があります。

`XSLTransformerListener()` メソッドを実装することによって、変換の完了が、コール元のアプリケーションに通知されます。アプリケーションは、変換の要求から受信の間に自由にその他のタスクを実行できます。

XSL Transviewer Bean の使用例 1: 基礎となるデータが変更される場合の HTML の再生成

この使用例では、XSLTransformer Bean を適用する方法の 1 つを示します。

1. SQL 問合せを作成します。選択された XML データを CLOB 表へ格納します。
2. XSL Transfomer Bean を使用して XSL スタイルシートを作成し、適切なデータ表示が行われるまで、対話式でこのスタイルシートを XML データに適用します。この表示は、XSL 変換によって生成される HTML である場合もあります。
3. 必要な SQL（データ選択）および XSL（データ表示）を得た後、その SQL 問合せが使用する表またはビューにトリガーを作成します。このトリガーは、ストアド・プロシージャを実行できます。たとえば、ストアド・プロシージャは、次のような操作を行います。
   - 問合せの実行
   - スタイルシートの適用
   - 結果の HTML の CLOB 表への格納
4. ソース・データの表が更新された場合、この処理を再度実行できます。

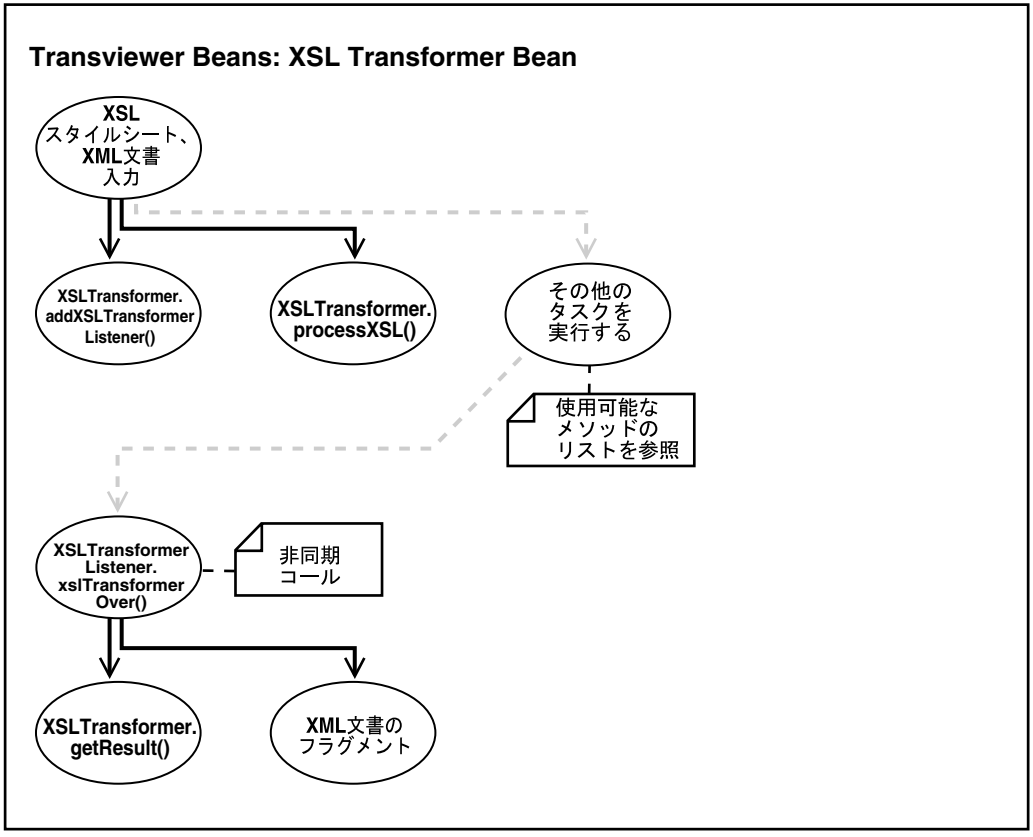
   CLOB 表に格納された HTML は、問合せ中の表に格納された最新のデータを常にミラー化します。JSP は HTML を表示できます。

   この使用例では、複数のエンド・ユーザーが複数のデータ問合せを作成することはないため、データベースに大きな負荷はかかりません。HTML は、基礎となるデータが変更された後にのみ再生成されます。

XSLTransformer Bean の使用方法

図 23-2 に、XSLTransformer Bean の使用方法を示します。この Bean を実装する例は、23-36 ページの「Transviewer Bean の例 1: AsyncTransformSample.java」を参照してください。

図 23-2 XSLTransformer Bean の使用方法



1. XSLTransfomer は、`XSLTransfomer.addXSLTransformerListener(XSLTransformerListener)` メソッドを使用して、XSL スタイルシートおよび XML 文書を入力します。このメソッドはリスナーを追加します。
2. `XSLTransfomer.processXSL()` メソッドは、XSL 変換をバックグラウンドで開始します。
3. オプションで、その他の作業を XSLTransformer Bean に割り当てることができます。表 23-2 は、使用可能な XSLTransformer Bean メソッドを示しています。
4. 変換が完了すると非同期コールが行われ、`XSLTransformerListener.xslTransformerOver()` メソッドがコールされます。非同期変換中のイベントの通知を受信するには、このインタフェースが実装されている必要があります。このインタフェースを実装するクラスは、`addXSLTransformerListener` メソッドを使用して XSLTransformer イベント・キューに追加される必要があります。
5. `XSLTransformer.getResult()` メソッドは、結果の文書に対して XML 文書のフラグメントを戻します。
6. このメソッドは、XML 文書のフラグメントを出力します。

表 23-2 XSLTransformer Bean: メソッド

メソッド	説明
addXSLTransformerErrorListener(XSL TransformerErrorListener)	エラー・イベント・リスナーを追加します。
addXSLTransformerListener(XSLTransformer Listener)	リスナーを追加します。
getId()	一意の XSLTransformer ID を戻します。
getResult()	結果文書の文書フラグメントを戻します。
processXSL(XSLStylesheet, InputStream, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, Reader, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, URL, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, XMLDocument)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, XMLDocument, OutputStream)	バックグラウンドで XSL 変換を開始します。

表 23-2 XSLTransformer Bean: メソッド（続き）

メソッド	説明
removeDOMTransformerErrorListener(XSL TransformerErrorListener)	エラー・イベント・リスナーを削除します。
removeXSLTransformerListener(XSL TransformerListener)	リスナーを削除します。
run()	
setErrorStream(OutputStream)	XSL プロセッサが使用するエラー・ストリームを設定します。
showWarnings(boolean)	XSL プロセッサが使用する showWarnings フラグを設定します。

XMLTreeViewer Bean の使用

XMLTreeViewer Bean は、XML 文書をツリーとして表示します。この Bean は、次の XML DOM ノードを認識します。

- タグ
- 属性名
- 属性値
- コメント
- CDATA
- PCDATA
- PI（処理命令）データ
- PI（処理命令）名
- 表記法

この Bean は、入力として org.w3c.dom.Document オブジェクトを取ります。

図 23-3「XMLTreeViewer Bean 実行画面 : XML 文書のツリーとしての表示」に、XMLTreeViewer Bean による XML 文書および編集オプションの表示方法を示します。

図 23-3 XMLTreeViewer Bean 実行画面 : XML 文書のツリーとしての表示

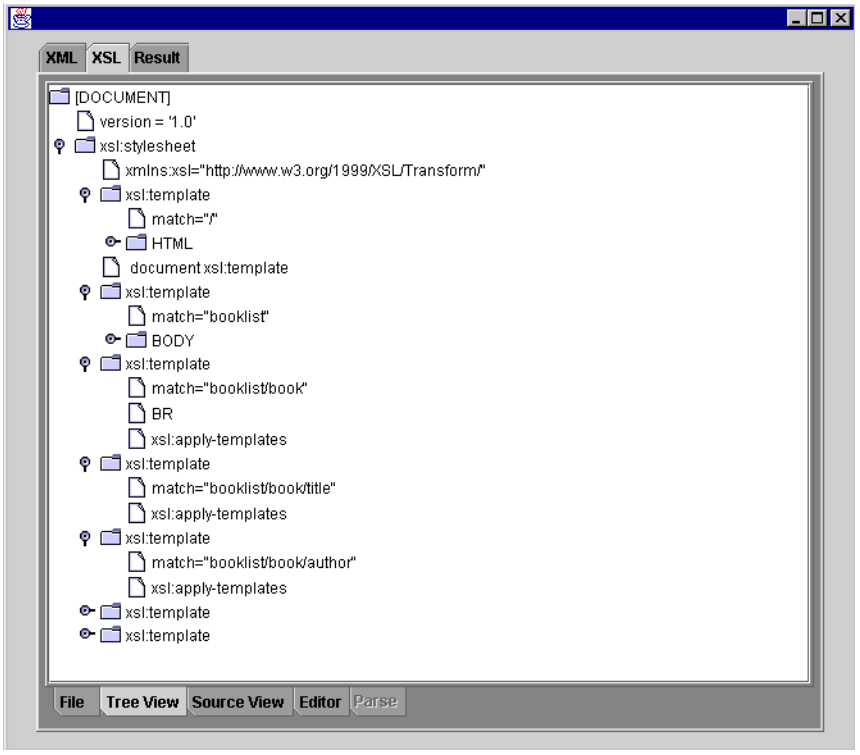
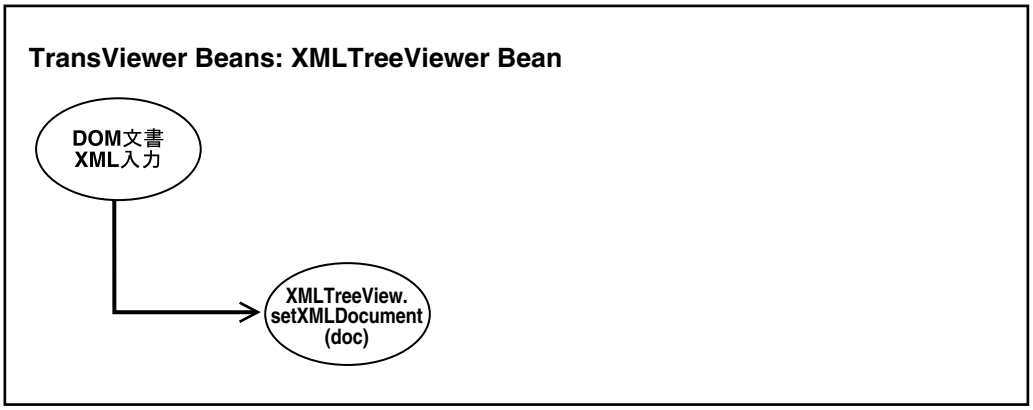


図 23-4 に、XMLTreeViewer Bean の使用方法を示します。DOM XML 文書は、`XMLTreeView.setXMLDocument(doc)` メソッドへの入力です。この入力によって、XML Tree Viewer が XML 文書に対応付けられます。使用可能な TreeViewer Bean メソッドは次のとおりです。

- getPreferredSize() - XMLTreeView の推奨サイズを戻します。
- setXMLDocument(Document) - XMLTreeViewer を XML 文書に対応付けます。
- updateUI() - XMLTreeView に強制的にユーザー・インタフェースを更新 / リフレッシュさせます。

図 23-4 XMLTreeViewer Bean の使用方法



XMLSourceViewer Bean の使用

XMLSourceViewer Bean は、XML 文書を表示するビジュアル的な JavaBean です。この Bean は、XML/XSL 構文を色でハイライトすることによって、XML ファイルおよび XSL ファイルの参照を容易にします。また、編集モードを提供します。XMLSourceViewer Bean は、DOMBuilder Bean と簡単に統合できます。この Bean は、指定された DTD に対する解析前または解析後のビジュアル化および検証を可能にします。

XMLSourceViewer Bean は、次の XML トークン型を認識します。

- タグ
- 属性名
- 属性値
- コメント
- CDATA
- PCDATA
- PI（処理命令）データ
- PI（処理命令）名
- 表記法

各トークン型には、フォアグラウンド・カラーおよびフォアグラウンド・フォントがあります。デフォルトのカラーおよびフォントの設定は、ユーザーが変更できます。この Bean は、入力として org.w3c.dom.Document オブジェクトを取ります。

XMLSourceViewer Bean の使用方法

図 23-6 に、XMLSourceViewer Bean の使用方法を示します。これは、`oracle.xml.srcviewer` API の一部です。DOM 文書は、XMLSourceView.SetXMLDocument(Doc) を入力します。結果の DOM 文書が表示されます。23-42 ページの「Transviewer Bean の例 2: ViewSample.java」を参照してください。

図 23-5「XMLSourceViewer Bean 実行画面 : 色でハイライトされた XML 文書の表示」に、タグを青、タグの内容を黒、属性を赤で表示した XML 文書を示します。

図 23-5 XMLSourceViewer Bean 実行画面 : 色でハイライトされた XML 文書の表示

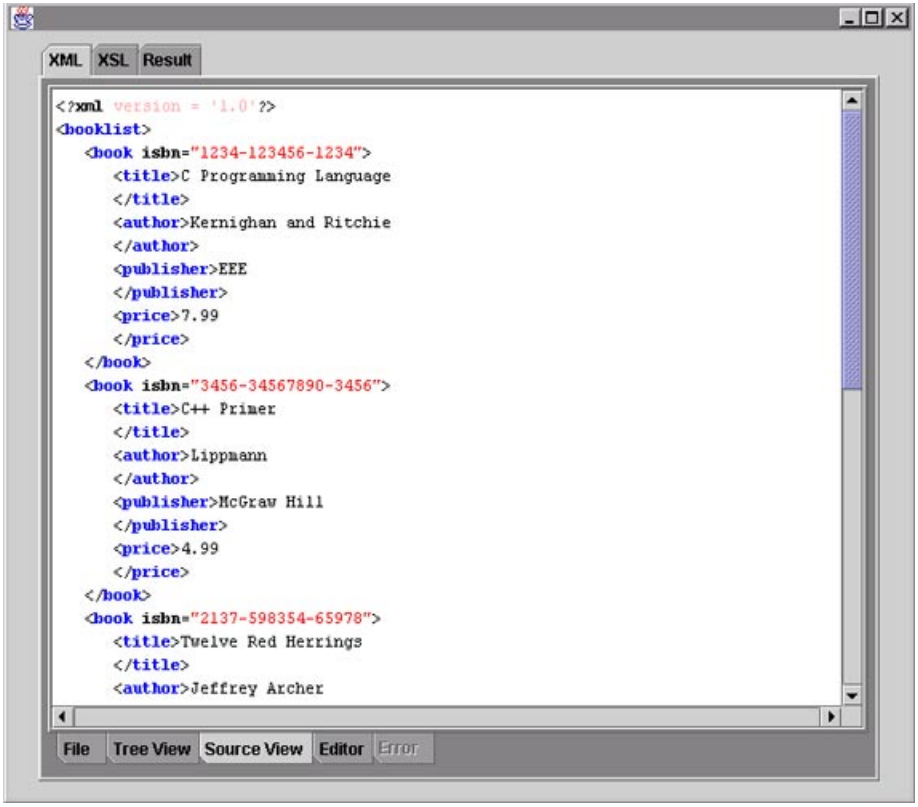


図 23-6 XMLSourceViewer Bean の使用方法

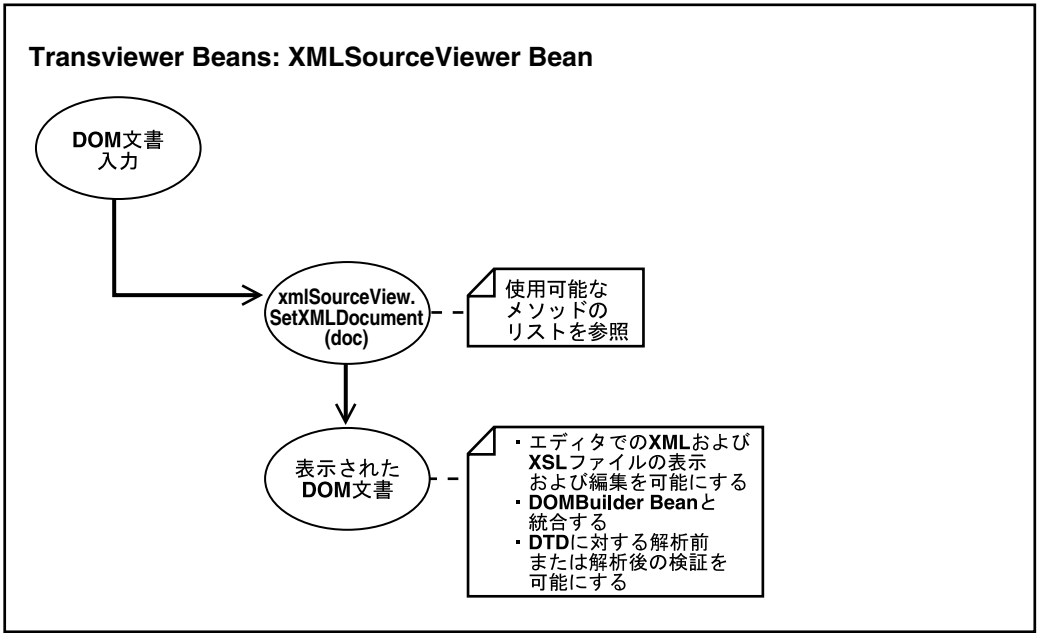


表 23-3 に、XMLSourceView メソッドのリストを示します。

表 23-3 XMLSourceView Bean メソッド

メソッド	説明
fontGet(AttributeSet)	指定された属性セットからフォントを取得して戻します。
fontSet(MutableAttributeSet, Font)	属性セットのフォントを設定します。
getAttributeNameFont()	属性名のフォントを戻します。
getAttributeNameForeground()	属性名のフォアグラウンド・カラーを戻します。
getAttributeValueFont()	属性値のフォントを戻します。
getAttributeValueForeground()	属性値のフォアグラウンド・カラーを戻します。
getBackground()	バックグラウンド・カラーを戻します。
getCDATAFont()	CDATA のフォントを戻します。

表 23-3 XMLSourceView Bean メソッド （続き）

メソッド	説明
getCDATAForeground()	CDATA のフォアグラウンド・カラーを戻します。
getCommentDataFont()	コメントのフォントを戻します。
getCommentDataForeground()	コメントのフォアグラウンド・カラーを戻します。
getEditedText()	編集済テキストを戻します。
getJTextPane()	JTextPane ビューアのコンポーネントを戻します。
getMinimumSize()	XMLSourceView の最小サイズを戻します。
getNodeAtOffset(int)	指定されたオフセットにある XML ノードを戻します。
getPCDATAFont()	PCDATA のフォントを戻します。
getPCDATAForeground()	PCDATA のフォアグラウンド・カラーを戻します。
getPIDataFont()	PI データのフォントを戻します。
getPIDataForeground()	PI データのフォアグラウンド・カラーを戻します。
getPINameFont()	PI 名のフォントを戻します。
getPINameForeground()	PI 名のフォアグラウンド・カラーを戻します。
getSymbolFont()	表記法のフォントを戻します。
getSymbolForeground()	表記法のフォアグラウンド・カラーを戻します。
getTagFont()	タグのフォントを戻します。
getTagForeground()	タグのフォアグラウンド・カラーを戻します。
getText()	XML 文書を文字列として戻します。
isEditable()	このオブジェクトが編集可能かどうかを示すブールを戻します。
selectNodeAt(int)	オフセット i にある XML ノードにカーソルを移動します。
setAttributeNameFont(Font)	属性名のフォントを設定します。
setAttributeNameForeground(Color)	属性名のフォアグラウンド・カラーを設定します。
setAttributeValueFont(Font)	属性値のフォントを設定します。
setAttributeValueForeground(Color)	属性値のフォアグラウンド・カラーを設定します。
setBackground(Color)	バックグラウンド・カラーを設定します。
setCDATAFont(Font)	CDATA のフォントを設定します。

表 23-3 XMLSourceView Bean メソッド （続き）

メソッド	説明
setCDATAForeground(Color)	CDATA のフォアグラウンド・カラーを設定します。
setCommentDataFont(Font)	コメントのフォントを設定します。
setCommentDataForeground(Color)	コメントのフォアグラウンド・カラーを設定します。
setEditable(boolean)	このオブジェクトを編集可能にするかどうかを示すために指定されたブールを設定します。
setPCDATAFont(Font)	PCDATA のフォントを設定します。
setPCDATAForeground(Color)	PCDATA のフォアグラウンド・カラーを設定します。
setPIDataFont(Font)	PI データのフォントを設定します。
setPIDataForeground(Color)	PI データのフォアグラウンド・カラーを設定します。
setPINameFont(Font)	PI 名のフォントを設定します。
setPINameForeground(Color)	PI 名のフォアグラウンド・カラーを設定します。
setSelectedNode(Node)	選択された XML ノードにカーソル位置を設定します。
setSymbolFont(Font)	表記法のフォントを設定します。
setSymbolForeground(Color)	表記法のフォアグラウンド・カラーを設定します。
setTagFont(Font)	タグのフォントを設定します。
setTagForeground(Color)	タグのフォアグラウンド・カラーを設定します。
setXMLDocument(Document)	XMLviewer と XML 文書を対応付けます。

XMLTransformPanel Bean の使用

XMLTransformPanel Bean はビジュアル的な Bean であり、XSL 変換を XML 文書に適用します。この Bean は結果をビジュアル化し、入力の XML 文書やファイルおよび XSL ドキュメントやファイルの編集を可能にします。XMLTransformPanel Bean は、プログラムによる入力を必要としません。これは直接ユーザーと対話するコンポーネントであり、カスタマイズはできません。

XMLTransformPanel Bean の機能

XMLTransformPanel Bean には次のような機能があります。

- ファイル・システムからは XML ファイルおよび XSL ファイルを、Oracle からは XML ファイル、XSL ファイルおよび HTML ファイルを格納および取得できます。Oracle9*i* では、XMLTransformerPanel Bean は 2 列の CLOB 表を使用します。最初の列はデータ名（ファイル名）を格納し、2 番目の列はデータ・テキスト（ファイルのデータ）を CLOB で格納します。Bean はスキーマにあるすべての CLOB 表をリストします。表をクリックすると、Bean はそのファイル名を示します。また、表の作成や削除、表からのファイルの取出しおよび表へのファイルの追加ができるため、情報の編成に有効です。図 23-7「XMLTransformPanel Bean 実行画面 : CLOB 表およびデータ名の表示」を参照してください。

  注意： XSLTransformer Bean が作成した CLOB 表は、トリガー・ベースのストアド・プロシージャによって使用され、データベースの表またはビューをこれらの CLOB 表にある HTML データへミラー化できます。23-10 ページの「XSL Transviewer Bean の使用例 1: 基礎となるデータが変更される場合の HTML の再生成」を参照してください。

- 複数のデータベース接続をサポートします。
- データベースの結果セットから XML を作成します。この機能によって、すべての SQL 問合せを現在接続しているデータベースへ送信できます。この Bean は結果セットを XML へ変換し、追加の処理のために、自動的にこの XML データを Bean の XML バッファへロードします。
- この Bean にロードされた XML データおよび XSL データを編集します。
- XSL 変換を XML バッファへ適用し、結果を表示します。この Bean を使用すると、ファイル・システムまたはデータベースの CLOB へ結果をエクスポートできます。

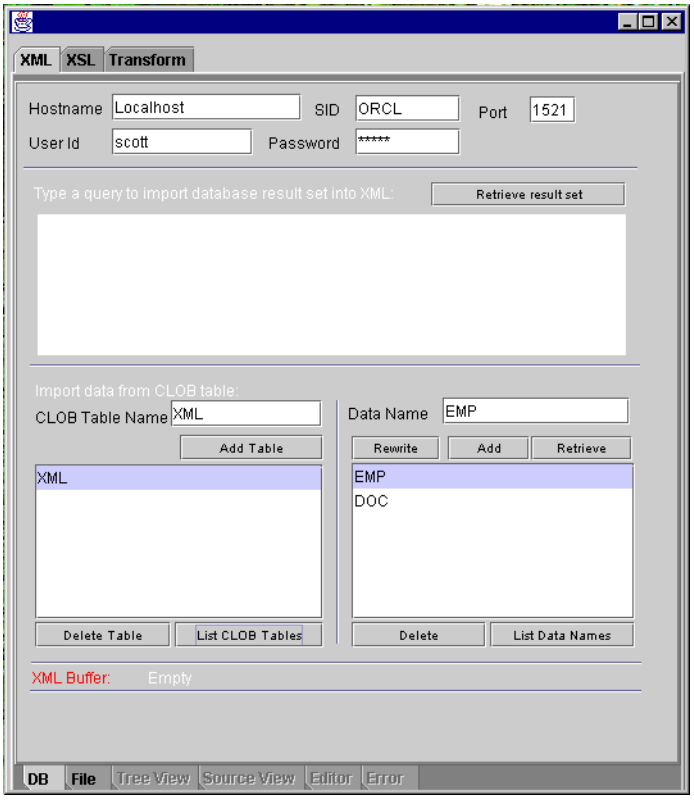
Transviewer Bean アプリケーション

Transviewer Bean は、XMLTransformPanel Bean の使用方法を示すアプリケーションです。Transviewer Bean は、次の操作を実行するためにコマンドラインから使用できます。

- XML ファイルの編集および解析
- XSL 変換の編集および適用
- ファイル・システムまたは Oracle にある XML、XSL および結果ファイルの取出しおよび保存

  参照： XMLTransformPanel Bean の使用方法の例は、23-46 ページの「Transviewer Bean の例 3: XMLTransformPanelSample.java」を参照してください。

図 23-7 XMLTransformPanel Bean 実行画面 : CLOB 表およびデータ名の表示



DBViewer Bean の使用

DBViewer Bean を使用すると、XSL スタイルシートを適用し、結果の HTML をビジュアル化して、スクロール可能な Swing パネル内にすべての XML 文書に対するデータベース問合せを表示できます。次の図を参照してください。

- 図 23-8「DBViewer Bean 実行画面 : データベース問合せの入力による XML の生成」
- 図 23-9「DBViewer Bean 実行画面 : XSL スタイルシートを使用して HTML に変換した後の XML 文書の表示」

DBViewer Bean には、次の 3 つのバッファがあります。

- XML
- XSL
- 結果バッファ

DBViewer Bean API を使用すると、コール元のプログラムは様々なソースからバッファをロードまたは保存し、XSL バッファのスタイルシートを使用して、XML バッファにスタイルシート変換を適用できます。結果は、結果バッファに格納できます。

内容の表示

XML バッファおよび XSL バッファの内容は、ソースまたはツリー構造として表示できます。結果バッファの内容も、HTML としてレンダリングし、ソースまたはツリー構造として表示できます。

バッファのロードおよび保存

XML バッファは、データベース問合せを使用してロードできます。次の場所から、すべてのバッファをロードし、ファイルを保存することができます。

- Oracle の CLOB 表
- ファイル・システム

このため、ファイル・システムとデータベースのユーザー・スキーマの間で、ファイルを移動させるための制御を使用することもできます。

図 23-8 DBViewer Bean 実行画面 : データベース問合せの入力による XML の生成

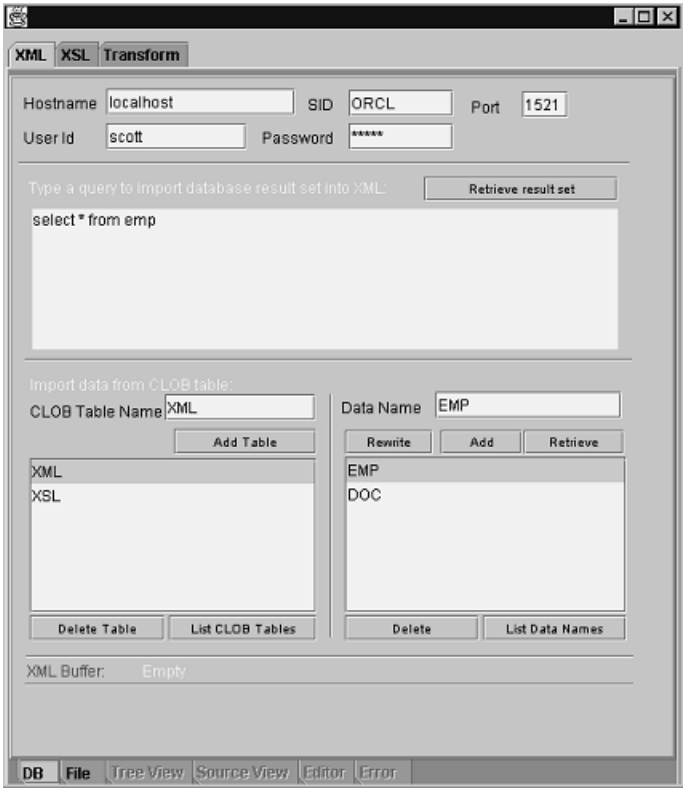
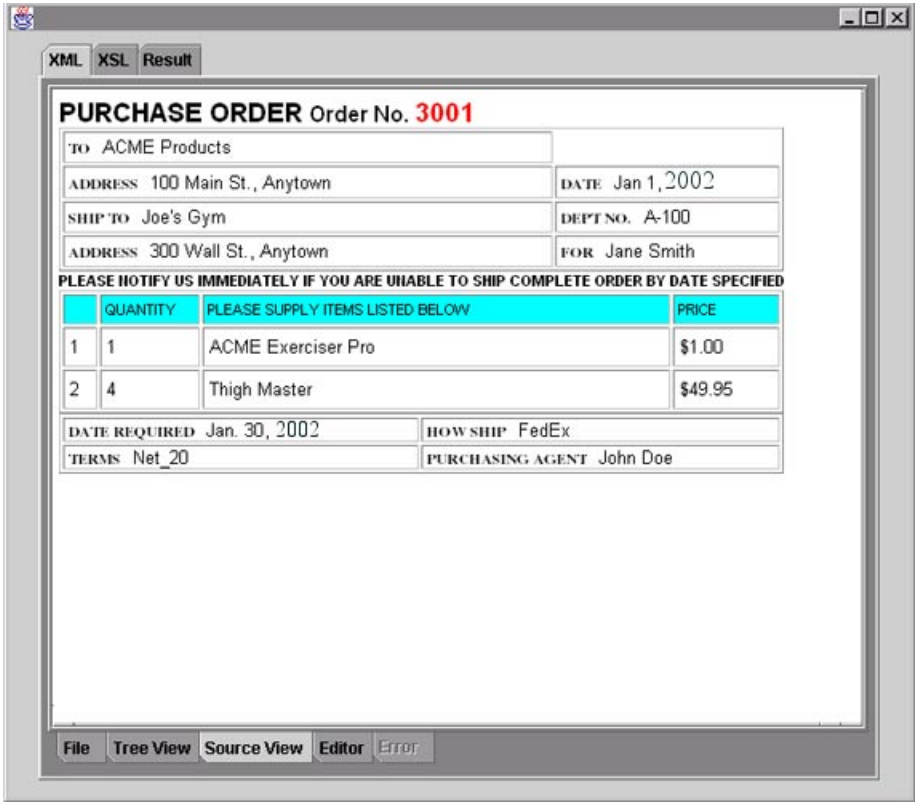


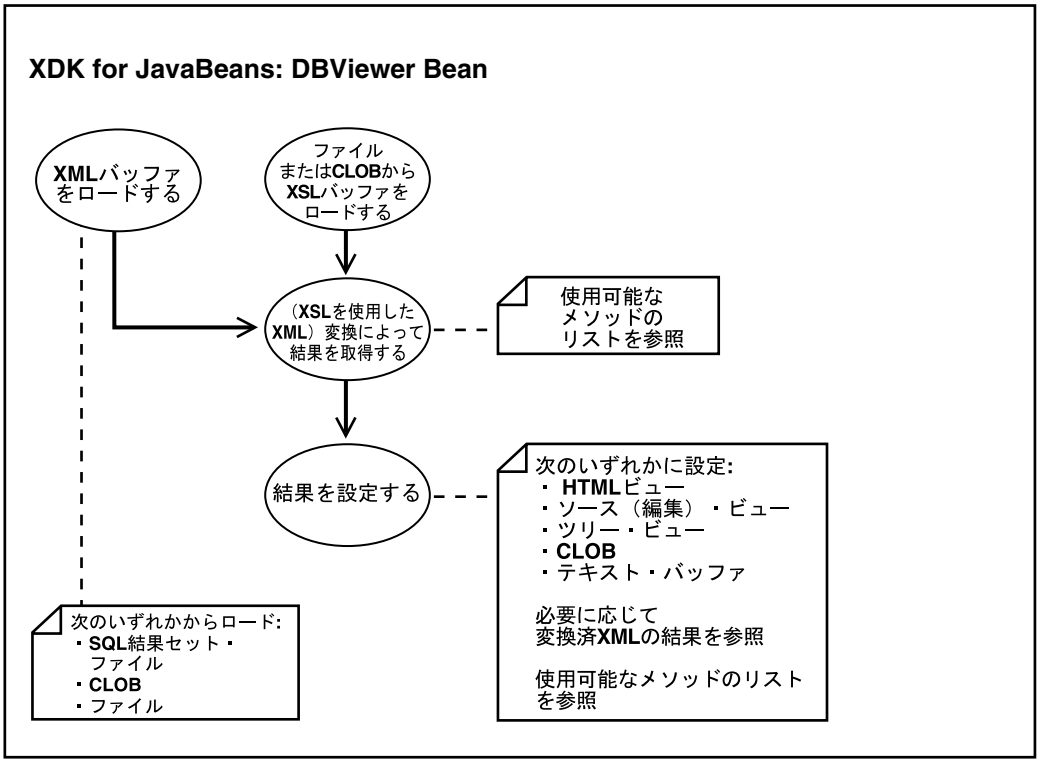
図 23-9 DBViewer Bean 実行画面 : XSL スタイルシートを使用して HTML に変換した後の XML 文書の表示



DBViewer Bean の使用方法

図 23-10 に、DBViewer Bean の使用方法を示します。

図 23-10 DBViewer Bean の使用方法



DBViewer Bean メソッド

表 23-4 に、DBViewer Bean メソッドのリストを示します。

表 23-4 DBViewer Bean メソッド

メソッド	説明
DBViewer()	新しいインスタンスを作成します。
getHostname()	データベースのホスト名を取得します。
getInstancename()	データベース・インスタンス名を取得します。
getPassword()	ユーザー・パスワードを取得します。
getPort()	データベースのポート番号を取得します。
getResBuffer()	結果バッファの内容を取得します。
getResCLOBFileName()	結果の CLOB ファイル名を取得します。
getResCLOBTableName()	結果の CLOB 表名を取得します。
getResFileName()	結果ファイル名を取得します。
getUsername()	ユーザー名を取得します。
getXmlBuffer()	XML バッファの内容を取得します。
getXmlCLOBFileName()	XML の CLOB ファイル名を取得します。
getXmlCLOBTableName()	XML の CLOB 表名を取得します。
getXmlFileName()	XML ファイル名を取得します。
getXMLStringFromSQL(String)	SQL 問合せから結果セットの XML 表示を取得します。
getXslBuffer()	XSL バッファの内容を取得します。
getXslCLOBFileName()	XSL の CLOB ファイル名を取得します。
getXslCLOBTableName()	XSL の CLOB 表名を取得します。
getXslFileName()	XSL ファイル名を取得します。
loadResBuffer(String)	ファイルから結果バッファをロードします。
loadResBuffer(String, String)	CLOB ファイルから結果バッファをロードします。
loadResBufferFromClob()	CLOB ファイルから結果バッファをロードします。

表 23-4 DBViewer Bean メソッド（続き）

メソッド	説明
loadResBufferFromFile()	ファイルから結果バッファをロードします。
loadXmlBuffer(String)	ファイルから XML バッファをロードします。
loadXmlBuffer(String, String)	CLOB ファイルから XML バッファをロードします。
loadXmlBufferFromClob()	CLOB ファイルから XML バッファをロードします。
loadXmlBufferFromFile()	ファイルから XML バッファをロードします。
loadXMLBufferFromSQL(String)	SQL 結果セットから XML バッファをロードします。
loadXslBuffer(String)	ファイルから XSL バッファをロードします。
loadXslBuffer(String, String)	CLOB ファイルから XSL バッファをロードします。
loadXslBufferFromClob()	CLOB ファイルから XSL バッファをロードします。
loadXslBufferFromFile()	ファイルから XSL バッファをロードします。
parseResBuffer()	結果バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
parseXmlBuffer()	XML バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
parseXslBuffer()	XSL バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
saveResBuffer(String)	結果バッファをファイルに保存します。
saveResBuffer(String, String)	結果バッファを CLOB ファイルに保存します。
saveResBufferToClob()	結果バッファを CLOB ファイルに保存します。
saveResBufferToFile()	結果バッファをファイルに保存します。
saveXmlBuffer(String)	XML バッファをファイルに保存します。
saveXmlBuffer(String, String)	XML バッファを CLOB ファイルに保存します。
saveXmlBufferToClob()	XML バッファを CLOB ファイルに保存します。
saveXmlBufferToFile()	XML バッファをファイルに保存します。
saveXslBuffer(String)	XSL バッファをファイルに保存します。

表 23-4 DBViewer Bean メソッド（続き）

メソッド	説明
saveXslBuffer(String, String)	XSL バッファを CLOB ファイルに保存します。
saveXslBufferToClob()	XSL バッファを CLOB ファイルに保存します。
saveXslBufferToFile()	XSL バッファをファイルに保存します。
setHostname(String)	データベースのホスト名を設定します。
setInstancename(String)	データベース・インスタンス名を設定します。
setPassword(String)	ユーザー・パスワードを設定します。
setPort(String)	データベースのポート番号を設定します。
setResBuffer(String)	結果バッファの新しいテキストを設定します。
setResCLOBFileName(String)	結果の CLOB ファイル名を設定します。
setResCLOBTableName(String)	結果の CLOB 表名を設定します。
setResFileName(String)	結果ファイル名を設定します。
setResHtmlView(boolean)	結果バッファをレンダリング済 HTML として表示します。
setResSourceEditView(boolean)	結果バッファを XML ソースとして表示し、編集モードに入ります。
setResSourceView(boolean)	結果バッファを XML ソースとして表示します。
setResTreeView(boolean)	結果バッファを XML ツリー・ビューとして表示します。
setUsername(String)	ユーザー名を設定します。
setXmlBuffer(String)	XML バッファの新しいテキストを設定します。
setXmlCLOBFileName(String)	XML の CLOB ファイル名を設定します。
setXmlCLOBTableName(String)	XML の CLOB 表名を設定します。
setXmlFileName(String)	XML ファイル名を設定します。
setXmlSourceEditView(boolean)	XML バッファを XML ソースとして表示し、編集モードに入ります。
setXmlSourceView(boolean)	XML バッファを XML ソースとして表示します。
setXmlTreeView(boolean)	XML バッファをツリーとして表示します。
setXslBuffer(String)	XSL バッファに新しいテキストを設定します。
setXslCLOBFileName(String)	XSL の CLOB ファイル名を設定します。

表 23-4 DBViewer Bean メソッド（続き）

メソッド	説明
setXslCLOBTableName(String)	XSL の CLOB 表名を設定します。
setXslFileName(String)	XSL ファイル名を設定します。
setXslSourceEditView(boolean)	XSL バッファを XML ソースとして表示し、編集モードに入ります。
setXslSourceView(boolean)	XSL バッファを XML ソースとして表示します。
setXslTreeView(boolean)	XSL バッファをツリーとして表示します。
transformToDoc()	XSL バッファのスタイルシートを適用して、XML バッファの内容を変換します。
transformToRes()	XML バッファ内の XML に XSL バッファのスタイルシート変換を適用し、その結果を結果バッファに格納します。
transformToString()	XSL バッファのスタイルシートを適用して、XML バッファの内容を変換します。

DBAccess Bean の使用

DBAccess Bean は、複数の XML およびテキスト・ドキュメントを保持する CLOB 表をメンテナンスします。各表は、次の文を使用して作成されます。

```
CREATE TABLE tablename FILENAME CHAR( 16) UNIQUE, FILEDATA CLOB) LOB(FILEDATA)
STORE AS (DISABLE STORAGE IN ROW)
```

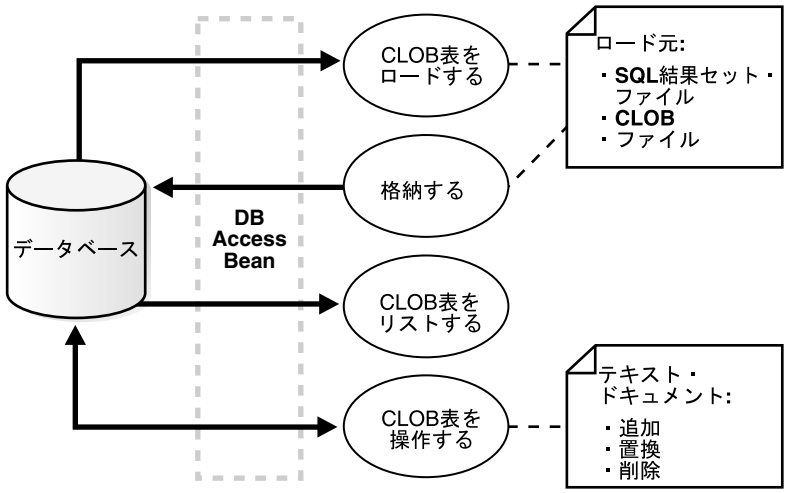
各 XML（またはテキスト）文書は表の行として格納されます。FILENAME フィールドには、その行を検索、更新または削除するためにキーとして使用される一意の文字列があります。文書のテキストは FILEDATA フィールドに格納されます。これは CLOB オブジェクトです。CLOB 表は Transviewer Bean が自動的にメンテナンスします。DBAccess Bean がメンテナンスする CLOB 表は、Transviewer Bean が後で使用します。DBAccess Bean は次のタスクを行います。

- CLOB 表の作成および削除
- CLOB 表の内容の表示
- CLOB 表にあるテキスト・ドキュメントの追加、置換または削除

DBAccess Bean の使用方法

図 23-11 に、DBAccess Bean の使用方法を示します。DBAccess Bean が、CLOB に格納された XML 文書をメンテナンスおよび操作する方法を示します。

図 23-11 DBAccess Bean の使用方法



DBAccess Bean メソッド

表 23-5 に、DBAccess Bean メソッドのリストを示します。

表 23-5 DBAccess Bean メソッド

メソッド	説明
createXMLTable(Connection,String)	XML 表を作成します。
deleteXMLName(Connection,String,String)	XML 表からテキスト・ファイルを削除します。
dropXMLTable(Connection,String)	XML 表を削除します。
getNameSize()	ファイル名が格納されているフィールドのサイズを戻します。
getXMLData(Connection,String,String)	XML 表からテキスト・ファイルを取り出します。
getXMLNames(Connection,String)	XML 表のすべてのファイル名を戻します。
getXMLTableNames(Connection,String)	指定された文字列で始まる名前を持つすべての XML 表を取得します。
insertXMLData(Connection,String,String, String)	テキスト・ファイルを XML 表の行として挿入します。
isXMLTable(Connection,String)	表が XML 表かどうかを確認します。
replaceXMLData(Connection,String,String, String)	テキスト・ファイルを XML 表の行として置換します。
xmlTableExists(Connection,String)	XML 表が存在するかどうかを確認します。

サンプル Transviewer Bean の実行

XDK for Java の Transviewer Bean の sample/ ディレクトリには、Oracle Transviewer Beans の使用方法を示すサンプル Transviewer Bean アプリケーションが含まれています。Oracle Transviewer Beans には、DOMBuilder Bean、XMLSourceViewer Bean、XMLTreeViewer Bean、XSLTransformer Bean、XMLTransformPanel Bean、DBViewer Bean および DBAccess Bean が含まれています。

表 23-6 に、sample/ ディレクトリにあるサンプル・ファイルを示します。

表 23-6 sample/ にある Transviewer Bean のサンプル・ファイル

ファイル名	説明
booklist.xml	例 1、2 または 3 で使用されるサンプル XML ファイルです。
doc.xml	例 1、2 または 3 で使用されるサンプル XML ファイルです。
doc.html	例 1、2 または 3 で使用されるサンプル HTML ファイルです。
doc.xsl	例 1、2 または 3 で使用されるサンプル入力 XSL ファイルです。doc.xsl は XSLTransformer に使用されます。
emptable.xsl	例 1、2 または 3 で使用されるサンプル入力 XSL ファイルです。
tohtml.xsl	例 1、2 または 3 で使用されるサンプル入力 XSL ファイルです。booklist.xml を変換します。
AsyncTransformSample.java 23-36 ページの「Transviewer Bean の例 1: AsyncTransformSample.java」を参照	XSLTransformer Bean および DOMBuilder Bean を使用するビジュアルでないサンプル・アプリケーションです。これは、doc.xsl で指定された XSLT スタイルシートを現在のディレクトリにあるすべての *.xml ファイルに適用します。結果は、拡張子が .log であるファイルにあります。
ViewSample.java 23-42 ページの「Transviewer Bean の例 2: ViewSample.java」を参照	XMLSourceViewer Bean および XMLTreeViewer Bean を使用するビジュアル的なサンプル・アプリケーションです。これは XML 文書ファイルをビジュアル化します。
XMLTransformPanelSample.java 23-46 ページの「Transviewer Bean の例 3: XMLTransformPanelSample.java」を参照	XMLTransformPanel Bean を使用するビジュアル的なアプリケーションです。この Bean は、前述の 4 つの Bean すべてを使用します。XSL 変換を XML 文書に適用し、結果をビジュアル化して、XML および XSL 入力ファイルの編集を可能にします。
DBViewSample 次の例を参照してください。 ■ 23-47 ページの「Transviewer Bean の例 4a: DBViewer Bean - DBViewClaims.java」 ■ 23-50 ページの「Transviewer Bean の例 4b: DBViewer Bean - DBViewFrame.java」 ■ 23-51 ページの「Transviewer Bean の例 4c: DBViewer Bean - DBViewSample.java」	DBViewer Bean を使用するビジュアル的なサンプル・アプリケーションです。単純な保険請求処理アプリケーションを実装します。

サンプル Transviewer Bean のインストール

Transviewer Beans は JDK 1.1.6 以上を必要とし、JDK 1.2 のすべてのバージョンでも使用できます。

1. Transviewer Beans が使用する次のコンポーネントをダウンロードして、インストールします。
   - Oracle JDBC Driver for Thin Client（jar ファイル classes111.zip）
   - Oracle XSU（jar ファイル oraclexmlsql.jar）

   コンポーネントをインストールした後、classes111.zip および oraclexmlsql.jar を CLASSPATH に含めてください。
2. Bean およびサンプルは Swing 1.1 を使用します。JDK 1.2 を使用している場合は、手順 3 へ進んでください。JDK 1.1 を使用している場合は、Sun 社から Swing 1.1 をダウンロードしてください。Swing をダウンロードした後、swingall.jar を CLASSPATH に追加します。
3. `Make` ファイルの JDKPATH を、JDK パスを指すように変更します。さらに、Windows NT の場合は、`Make` ファイルに記述されているとおりにファイル・セパレータを変更します。
4. scott/tiger アカウントを含むデフォルトのデータベースを使用していない場合、Make ファイルのユーザー ID およびパスワードを変更して、サンプル 4 を実行します。
5. 「**`make`**」を実行して .class ファイルを生成します。
6. 次のコマンドを使用して、サンプル・プログラムを実行します。
   - **gmake sample1**
   - **gmake sample2**
   - **gmake sample3**
   - **gmake sample4**
7. ViewSample を使用して、.log ファイル内の結果をビジュアル化します。
8. 「./tohtml.xsl」の XSLT ドキュメントを使用して、「./booklist.xml」の XML 文書を変換します。

いくつかの .xml ファイルが、テスト用に提供されています。XSLTransformer は、XSL スタイルシート「doc.xsl」を使用します。

注意： sample1 が XMLTransViewer プログラムを実行するため、XML ファイルを Oracle から取得および格納し、XSL 変換ファイルを Oracle に保持して、スタイルシートを XML に対話的に適用できます。

データベース接続機能の使用

このプログラムでデータベース接続機能を使用するには、次のことを確認する必要があります。

- Oracle または OracleAS を実行するコンピュータのネットワーク名
- ポート（通常は 1521）
- Oracle インスタンス名（通常は orcl）

また、CREATE TABLE 権限を持つアカウントも必要です。

Oracle システムでデフォルトのアカウント scott（パスワードは tiger）が使用可能な場合は、このアカウントを使用できます。

Make ファイルの実行

Make ファイル・スクリプトを次に示します。

```
# Makefile for sample java files

.SUFFIXES : .java .class

CLASSES = ViewSample.class  AsyncTransformSample.class XMLTransformPanelSample.class

# Change it to the appropriate separator based on the OS
PATHSEP= :

# Change this path to your JDK location. If you use JDK 1.1, you will need
# to download also Swing 1.1 and add swingall.jar to your classpath.
# You do not need to do this for JDK 1.2 since Swing is part of JDK 1.2
JDKPATH = /usr/local/packages/jdk1.2

# Make sure that the following product jar/zip files are in the classpath:
# - Oracle JDBC driver for thin client (file classes111.zip)
# - Oracle XML SQL Utility (file oraclexmlsql.jar)
# You can download this products from technet.us.oracle.com

#
CLASSPATH
:=$(CLASSPATH)$(PATHSEP)../lib/xmlparserv2.jar$(PATHSEP)../lib/xmlcomp.jar$(PATHSEP)
../lib/jdev-rt.zip$(PATHSEP).$(PATHSEP)
%.class: %.java
$(JDKPATH)/bin/javac -classpath "$(CLASSPATH)" $<

# make all class files
all: $(CLASSES)
```

```
sample1: XMLTransformPanelSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" XMLTransformPanelSample
sample2: ViewSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" ViewSample
sample3: AsyncTransformSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" AsyncTransformSample
```

Transviewer Bean の例 1: AsyncTransformSample.java

この例では、複数の XML ファイルを非同期に変換するための DOMBuilder Bean および XSLTransformer Bean の使用方法を示します。

```
import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.Vector;

import org.w3c.dom.DocumentFragment;
import org.w3c.dom.DOMException;

import oracle.xml.async.DOMBuilder;
import oracle.xml.async.DOMBuilderEvent;
import oracle.xml.async.DOMBuilderListener;
import oracle.xml.async.DOMBuilderErrorEvent;
import oracle.xml.async.DOMBuilderErrorListener;
import oracle.xml.async.XSLTransformer;
import oracle.xml.async.XSLTransformerEvent;
import oracle.xml.async.XSLTransformerListener;
import oracle.xml.async.XSLTransformerErrorEvent;
import oracle.xml.async.XSLTransformerErrorListener;
import oracle.xml.async.ResourceManager;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XSLStylesheet;
import oracle.xml.parser.v2.*;

public class AsyncTransformSample
{
  /**
   *  uses DOMBuilder bean
```

```
*/
void runDOMBuilders ()
{
   rm = new ResourceManager (numXMLDocs);

   for (int i = 0; i < numXMLDocs; i++)
   {
      rm.getResource();

      try
      {
         DOMBuilder builder = new DOMBuilder(i);

         URL  xmlURL = createURL(basedir + "/" +
                              (String)xmlfiles.elementAt(i));
         if (xmlURL == null)
            exitWithError("File " + (String)xmlfiles.elementAt(i) +
                          " not found");

         builder.setPreserveWhitespace(true);
         builder.setBaseURL (createURL(basedir + "/"));
         builder.addDOMBuilderListener (new DOMBuilderListener() {
            public void domBuilderStarted(DOMBuilderEvent p0) {}
            public void domBuilderError(DOMBuilderEvent p0) {}
            public synchronized void domBuilderOver(DOMBuilderEvent p0)
            {
               DOMBuilder bld = (DOMBuilder)p0.getSource();
               runXSLTransformer (bld.getDocument(), bld.getId());
            }
         });
         builder.addDOMBuilderErrorListener (new DOMBuilderErrorListener() {
            public void domBuilderErrorCalled(DOMBuilderErrorEvent p0)
            {
               int id = ((DOMBuilder)p0.getSource()).getId();
               exitWithError("Error occurred while parsing " +
                  xmlfiles.elementAt(id) + ": " +
                  p0.getException().getMessage());
            }
         });
         builder.parse (xmlURL);

         System.err.println("Parsing file " + xmlfiles.elementAt(i));
      }
      catch (Exception e)
      {
         exitWithError("Error occurred while parsing " +
                       (String)xmlfiles.elementAt(i) + ": " +
```

```
                        e.getMessage());
      }
   }
}

/**
 *  uses XSLTransformer bean
 */
void runXSLTransformer (XMLDocument xml, int id)
{
   try
   {
      XSLTransformer processor = new XSLTransformer (id);
      XSLStylesheet  xsl       = new XSLStylesheet (xsldoc, xslURL);

      processor.showWarnings (true);
      processor.setErrorStream (errors);
      processor.addXSLTransformerListener (new XSLTransformerListener() {
         public void xslTransformerStarted (XSLTransformerEvent p0) {}
         public void xslTransformerError(XSLTransformerEvent p0) {}
         public void xslTransformerOver (XSLTransformerEvent p0)
         {
            XSLTransformer trans = (XSLTransformer)p0.getSource();
            saveResult (trans.getResult(),  trans.getId());
         }
      });
processor.addXSLTransformerErrorListener (new XSLTransformerErrorListener() {
public void xslTransformerErrorCalled(XSLTransformerErrorEvent p0)
         {
            int i = ((XSLTransformer)p0.getSource()).getId();
           exitWithError("Error occurred while processing " +
                 xmlfiles.elementAt(i) + ": " +
                 p0.getException().getMessage());
          }
      });
      processor.processXSL (xsl, xml);
      // transform xml document
   }
   catch (Exception e)
   {
      exitWithError("Error occurred while processing " + xslFile + ": " +
                    e.getMessage());
   }
}

 void saveResult (DocumentFragment result, int id)
 {
```

```
   System.err.println("Transforming '" + xmlfiles.elementAt(id) +
            "' to '" + xmlfiles.elementAt(id) + ".log'" +
            " applying '" + xslFile);

   try
   {
       File resultFile = new File((String)xmlfiles.elementAt(id) + ".log");

       ((XMLNode)result).print(new FileOutputStream(resultFile));
   }
   catch (Exception e)
   {
      exitWithError("Error occurred while generating output : " +
                    e.getMessage());
   }

   rm.releaseResource();
}

void makeXSLDocument ()
{
   System.err.println ("Parsing file " + xslFile);
   try
   {
      DOMParser parser = new DOMParser();
      parser.setPreserveWhitespace (true);
      xslURL = createURL (xslFile);
      parser.parse (xslURL);
      xsldoc = parser.getDocument();
   }
   catch (Exception e)
   {
      exitWithError("Error occurred while parsing " + xslFile + ": " +
                    e.getMessage());
   }
}

private URL createURL(String fileName) throws Exception
{
   URL url = null;

   try
   {
      url = new URL(fileName);
   }
   catch (MalformedURLException ex)
   {
```

```
      File f = new File(fileName);

      try
      {
         String path = f.getAbsolutePath();
         // This is a bunch of weird code that is required to
         // make a valid URL on the Windows platform, due
         // to inconsistencies in what getAbsolutePath returns.
         String fs = System.getProperty("file.separator");
         if (fs.length() == 1)
         {
            char sep = fs.charAt(0);
            if (sep != '/')
               path = path.replace(sep, '/');
            if (path.charAt(0) != '/')
               path = '/' + path;
         }
         path = "file://" + path;
         url = new URL(path);
      }
      catch (MalformedURLException e)
      {
         exitWithError("Cannot create url for: " + fileName);
      }
   }

   return url;
}

boolean init () throws Exception
{
   File     directory = new File (basedir);
   String[] dirfiles = directory.list();
   for (int j = 0; j < dirfiles.length; j++)
   {
      String dirfile = dirfiles[j];

      if (!dirfile.endsWith(".xml"))
          continue;

       xmlfiles.addElement(dirfile);
   }

   if (xmlfiles.isEmpty()) {
   System.out.println("No files in directory were selected for processing");
      return false;
   }
```

```
   numXMLDocs = xmlfiles.size();

   return true;
}

private void exitWithError(String msg)
{
   PrintWriter errs = new PrintWriter(errors);
   errs.println(msg);
   errs.flush();
   System.exit(1);
}

void asyncTransform () throws Exception
{
   System.err.println (numXMLDocs +
            " XML documents will be transformed" +
            " using XSLT stylesheet specified in " + xslFile +
            " with " +  numXMLDocs + " threads");

   makeXSLDocument ();
   runDOMBuilders ();

   // wait for the last request to complete
   while (rm.activeFound())
      Thread.sleep(100);

}
String       basedir = new String (".");
OutputStream errors = System.err;

Vector xmlfiles = new Vector();
int    numXMLDocs = 1;

String      xslFile = new String ("doc.xsl");
URL         xslURL;
XMLDocument xsldoc;

private ResourceManager rm;

/**
 *   main
 */
public static void main (String args[])
{
   AsyncTransformSample inst = new AsyncTransformSample();
```

```
      try
      {
         if (!inst.init())
            System.exit(0);

         inst.asyncTransform ();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }

      System.exit(0);
   }
}
```

Transviewer Bean の例 2: ViewSample.java

この例では、XML ファイルをビジュアル的に表すための XMLSourceViewer Bean および XMLTreeViewer Bean の使用方法を示します。

```
import java.awt.*;
import oracle.xml.srcviewer.*;
import oracle.xml.treeviewer.*;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ViewSample
{

  public static void main(String[] args)
  {
     String fileName = new String ("booklist.xml");
     if (args.length > 0) {
        fileName = args[0];
     }

     JFrame        frame         = setFrame ("XMLViewer");
     XMLDocument   xmlDocument   = getXMLDocumentFromFile (fileName);
```

```
    XMLSourceView xmlSourceView = setXMLSourceView (xmlDocument);
    XMLTreeView   xmlTreeView   = setXMLTreeView (xmlDocument);
    JTabbedPane   jtbPane       = new JTabbedPane ();

    jtbPane.addTab ("Source", null, xmlSourceView, "XML document sorce view");
    jtbPane.addTab ("Tree", null, xmlTreeView, "XML document tree view");
    jtbPane.setPreferredSize (new Dimension(400,300));
    frame.getContentPane().add (jtbPane);

    frame.setTitle    (fileName);
    frame.setJMenuBar (setMenuBar());
    frame.setVisible  (true);
}

static JFrame setFrame (String title)
{
  JFrame frame = new JFrame (title);
  //Center the window
  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize  = frame.getSize();
  if (frameSize.height > screenSize.height) {
     frameSize.height = screenSize.height;
  }
  if (frameSize.width > screenSize.width) {
    frameSize.width = screenSize.width;
  }
  frame.setLocation ((screenSize.width - frameSize.width)/2,
                     (screenSize.height - frameSize.height)/2);
  frame.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent e) {
        System.exit(0);
     }
  });
  frame.getContentPane().setLayout (new BorderLayout());
  frame.setSize(new Dimension(400, 300));
  frame.setVisible (false);
  frame.setTitle (title);

  return frame;
}

static JMenuBar setMenuBar ()
{
    JMenuBar menuBar = new JMenuBar();
    JMenu    menu    = new JMenu ("Exit");
    menu.addMenuListener ( new MenuListener () {
      public void menuSelected (MenuEvent ev) { System.exit(0); }
```

```
      public void menuDeselected (MenuEvent ev) {}
      public void menuCanceled (MenuEvent ev) {}
    });
    menuBar.add (menu);
    return menuBar;
}

/**
 * creates  XMLSourceView object
 */
static XMLSourceView setXMLSourceView(XMLDocument xmlDocument)
{
  XMLSourceView xmlView = new XMLSourceView();

  xmlView.setXMLDocument(xmlDocument);
  xmlView.setBackground(Color.yellow);
  xmlView.setEditable(true);
  return xmlView;
}
/**
 * creates  XMLTreeView object
 */
static XMLTreeView setXMLTreeView(XMLDocument xmlDocument)
{
  XMLTreeView xmlView = new XMLTreeView();

  xmlView.setXMLDocument(xmlDocument);
  xmlView.setBackground(Color.yellow);
  return xmlView;
}

static XMLDocument getXMLDocumentFromFile (String fileName)
{
  XMLDocument doc = null;

  try  {
    DOMParser parser = new DOMParser();
    try {
       String dir= "" ;
       FileInputStream in = new FileInputStream(fileName);
       parser.setPreserveWhitespace(false);
       parser.setBaseURL(createURL(dir));
       parser.parse(in);
       in.close();
    } catch (Exception ex) {
       ex.printStackTrace();
       System.exit(0);
```

```
    }

    doc = (XMLDocument)parser.getDocument();

    try {
       doc.print(System.out);
    } catch (Exception ie) {
       ie.printStackTrace();
       System.exit(0);
    }

  }
  catch (Exception e) {
    e.printStackTrace();
  }
  return doc;
}

static URL createURL(String fileName)
{
    URL url = null;
    try
    {
       url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
       File f = new File(fileName);
       try
       {
          String path = f.getAbsolutePath();
          String fs = System.getProperty("file.separator");
          if (fs.length() == 1)
          {
             char sep = fs.charAt(0);
             if (sep != '/')
                path = path.replace(sep, '/');
             if (path.charAt(0) != '/')
                path = '/' + path;
          }
          path = "file://" + path;
          url = new URL(path);
       }
       catch (MalformedURLException e)
       {
          System.out.println("Cannot create url for: " + fileName);
          System.exit(0);
```

```
        }
      }
      return url;
    }
}
```

Transviewer Bean の例 3: XMLTransformPanelSample.java

この例では、XMLTransformPanel Bean を使用して、次の操作を行う対話型アプリケーションを示します。

- データベース問合せから XML を生成します。
- XSL スタイルシートを使用して XML を変換します。
- 結果を表示します。
- データベースの CLOB 表に結果を格納します。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import oracle.xml.transviewer.XMLTransformPanel;


public class XMLTransformPanelSample
{
  XMLTransformPanel transformPanel = new XMLTransformPanel();

  /**
   * Adjust frame size and add transformPanel to it.
   */
  public XMLTransformPanelSample ()
  {
    Frame     frame      = new JFrame();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setSize(510,550);
    transformPanel.setPreferredSize(new Dimension(510,550));
    Dimension frameSize = frame.getSize();

    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation ((screenSize.width - frameSize.width)/2,
                       (screenSize.height - frameSize.height)/2);
    frame.addWindowListener(new WindowAdapter() {
```

```
      public void windowClosing(WindowEvent e) { System.exit(0); }
    });
    frame.setVisible(true);

    ((JFrame)frame).getContentPane().add (transformPanel);
    frame.pack();
  }

  /**
   *  main(). Only creates XMLTransformPanelSample object.
   */
  public static void main (String[] args)
  {
    new XMLTransformPanelSample ();
  }
}
```

Transviewer Bean の例 4a: DBViewer Bean - DBViewClaims.java

次に、保険請求書の名前または契約を対話的に入力する例を示します。XML 問合せの結果セットから、XML バッファとして適切な請求書がロードされます。次に、XSL スタイルシートがファイル・システムからロードされます。DBViewer Bean は、XSL スタイルシートを使用して、XML バッファを参照可能な HTML に変換します。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;
import oracle.xml.dbviewer.*;

public class DBViewClaims extends JPanel {
  DBViewer dbPanel= new DBViewer();
  JButton searchButton = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JLabel titleLabel = new JLabel();
  JLabel nameLabel = new JLabel();
  JLabel policyLabel = new JLabel();
  JTextField nameTF = new JTextField();
  JTextField policyTF = new JTextField();
  JButton viewXMLButton = new JButton();
  JButton viewXSLButton = new JButton();
  JButton viewHTMLButton = new JButton();
  public DBViewClaims() {
    super();
    try  {
      jbInit();
    }
```

```
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
    setBackground(SystemColor.controlLtHighlight);
    this.setLayout(xYLayout1);
    searchButton.setText("searchButton");
    searchButton.setLabel("Search");
    xYLayout1.setHeight(464);
    xYLayout1.setWidth(586);
    titleLabel.setText("List of Claims");
    titleLabel.setHorizontalAlignment(SwingConstants.CENTER);
    titleLabel.setBackground(new Color(192, 192, 255));
    titleLabel.setFont(new Font("Dialog", 1, 16));
    nameLabel.setText("Last Name");
    policyLabel.setText("Policy:");
    viewXMLButton.setText("viewXMLButton");
    viewXMLButton.setLabel("view XML");
    viewXMLButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        viewXMLButton_actionPerformed(e);
      }
    });
    viewXSLButton.setText("viewXSLButton");
    viewXSLButton.setLabel("view XSL");
    viewXSLButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        viewXSLButton_actionPerformed(e);
      }
    });
    viewHTMLButton.setText("viewHTMLButton");
    viewHTMLButton.setLabel("view HTML");
    viewHTMLButton.addActionListener(new java.awt.event.ActionListener() {

      public void actionPerformed(ActionEvent e) {
        viewHTMLButton_actionPerformed(e);
      }
    });

     searchButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          searchButton_actionPerformed(e);
        }
     });

    this.add(dbPanel, new XYConstraints(16, 55, 552, 302));
```

```
    this.add(searchButton, new XYConstraints(413, 415, 154, 29));
    this.add(titleLabel, new XYConstraints(79, 10, 413, 31));
    this.add(nameLabel, new XYConstraints(333, 373, 72, -1));
    this.add(policyLabel, new XYConstraints(334, 395, 59, -1));
    this.add(nameTF, new XYConstraints(413, 368, 155, -1));
    this.add(policyTF, new XYConstraints(413, 391, 156, -1));
    this.add(viewXMLButton, new XYConstraints(19, 359, 94, 29));
    this.add(viewXSLButton, new XYConstraints(19, 390, 94, 29));
    this.add(viewHTMLButton, new XYConstraints(19, 421, 94, 29));
    updateUI();
  }
  void searchButton_actionPerformed(ActionEvent e) {
    String sqlText="select * from s_claim c ";
    try {
      if (!nameTF.getText().equals("")) {
         sqlText=sqlText+" where c.claimpolicy.primaryinsured.lastname="+
                "'"+nameTF.getText()+"'";
      } else if (!policyTF.getText().equals("")) {
         sqlText=sqlText+" where c.claimpolicy.policyid="+
                policyTF.getText();
      }
      dbPanel.setUsername("scott");
      dbPanel.setPassword("tiger");
      dbPanel.setInstancename("orcl");
      dbPanel.setHostname("localhost");
      dbPanel.setPort("1521");
      dbPanel.loadXMLBufferFromSQL(sqlText);
      dbPanel.loadXslBuffer("xslfiles","CLAIM.XSL");
      dbPanel.transformToRes();
      dbPanel.setResHtmlView(true);
    } catch (Exception e1) {
      System.out.println(e1);
    }
  }
  void viewXMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXmlSourceEditView(true);
  }
  void viewXSLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXslSourceEditView(true);
  }
  void viewHTMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setResHtmlView(true);
  }
}
```

Transviewer Bean の例 4b: DBViewer Bean - DBViewFrame.java

この例では、DBView の請求書機能にアクセスするために、フレームにメニュー・バーを挿入します。その後、請求書をロードし、HTML で表示します。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;

public class DBViewFrame extends JFrame {
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenuItem menuListCustomerClaims = new JMenuItem();

  public DBViewFrame() {
    super();
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
    this.getContentPane().setLayout(new GridLayout(1,1));
    this.setSize(new Dimension(600, 550));
    menuFile.setText("File");
    menuFileExit.setText("Exit");
    menuListCustomerClaims.setText("List Claims");
    menuFileExit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        fileExit_ActionPerformed(e);
      }
    });
    menuListCustomerClaims.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         ListCustomerClaims_ActionPerformed(e);

      }
    });
    menuFile.add(menuFileExit);
    menuFile.add(menuListCustomerClaims);
    menuBar1.add(menuFile);
    this.setJMenuBar(menuBar1);
    this.setBackground(SystemColor.controlLtHighlight);
  }
```

```
  void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
  }
  void ListCustomerClaims_ActionPerformed(ActionEvent e) {
    this.getContentPane().removeAll();
    this.getContentPane().add(new DBViewClaims());
    this.getContentPane().paintAll(this.getGraphics());
  }
}
```

Transviewer Bean の例 4c: DBViewer Bean - DBViewSample.java

この例では、DBViewFrame をインスタンス化する主なファンクションを提供し、固有のルックアンドフィールを実現します。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DBViewSample {
  public DBViewSample() {
    DBViewFrame frame = new DBViewFrame();
    frame.setVisible(true);
  }
  public static void main(String[] args) {
    try  {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    new DBViewSample();
  }
}
```

第 VIII 部

XDK for C

第 VIII 部では、XDK for C の入手方法および使用方法を説明します。第 VIII 部に含まれる章は、次のとおりです。

- 第 24 章「XML Parser for C の使用」
- 第 25 章「XML Schema Processor for C の使用」

24

XML Parser for C の使用

この章の内容は次のとおりです。

XML Parser for C の入手方法

XML Parser for C は Oracle および OracleAS に付属しています。OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

XML Parser for C は、`$ORACLE_HOME/xdk/c/parser` にあります。

XML Parser for C の機能

ソフトウェア・アーカイブのルート・ディレクトリにある `readme.html` ファイルには、不具合修正や追加の API などのリリース固有の情報が含まれています。

XML Parser for C は、XML 文書が整形式であるか、または DTD に対して妥当であるかどうか（オプション）を確認します。XML Parser for C は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。

質問、コメントまたは不具合のレポートは、OTN の Web サイト http://otn.oracle.com/tech/xml の「XML Discussion Forum」にポストできます。

仕様

XML Parser for C の仕様およびメソッドのリストは、付録 E「XDK for C: 仕様および早見表」を参照してください。

> **参照：** 次のディレクトリ、マニュアルおよび Web サイトを参照してください。
>
> - インストール場所の doc ディレクトリ
> - 『Oracle9*i* XML リファレンス』
> - http://otn.oracle.com/tech/xml/

メモリー割当て

独自のメモリー割当てを使用する場合は、メモリー・コールバック関数 `memcb` を使用します。この方法を使用するには、すべての関数を指定する必要があります。

SAX コールバックに渡されるパラメータに割り当てられたメモリー、または DOM 解析ツリーとともに格納されたノードおよびデータに割り当てられたメモリーは、次のいずれかの操作が完了すると解放されます。

- `xmlparse()` または `xmlparsebuf()` のコールによる、他のファイルまたはバッファの解析
- `xmlclean()` のコール
- `xmlterm()` のコール

スレッド・セーフティ

コールの初期化 / 解析 / 終了シーケンスの途中でスレッドが分岐している場合、不適切な動作および結果が発生する場合があります。

データ型索引

表 24-1 に、XML Parser for C で使用するデータ型を示します。

表 24-1 XML Parser for C で使用するデータ型

データ型	説明
oratext	文字列ポインタ
xmlctx	マスター XML のコンテキスト
xmlmemcb	メモリー・コールバック構造（オプション）
xmlsaxcb	SAX コールバック構造（SAX のみ）
ub4	32 ビット以上の符号なし整数
uword	システム固有の符号なし整数

エラー・メッセージ・ファイル

エラー・メッセージ・ファイルは、`mesg/` サブディレクトリにあります。このメッセージ・ファイルは、`$ORACLE_HOME/oracore/mesg` ディレクトリにもあります。環境変数 `ORA_XML_MESG` を設定し、`mesg/` サブディレクトリへの絶対パスを指定することもできます。ただし、これは必須ではありません。

検証モード

使用可能な検証モードの詳細は、20-5 ページの「Oracle XML Parser による 4 つの検証モードのサポート」を参照してください。

XML Parser for C の使用方法

図 24-1 に、次に示す XML Parser for C のコール順序を示します。

1. `XMLinit()` 関数によって、解析プロセスが初期化されます。
2. 解析済項目は、XML 文書（ファイル）または文字列バッファになります。次の関数を入力します。
   - `xmlparser()`（入力が XML ファイルの場合）
   - `xmlparserbuf()`（入力が文字列バッファの場合）
3. DOM API または SAX API を起動します。

   DOM: DOM インタフェースを使用する場合は、次の手順が含まれます。

   - `xmlparse()` 関数または `xmlparseBuffer()` 関数が、`.getDocumentElement()` をコールします。他の DOM 関数が適用されていない場合は、`xmlterm()` をコールできます。
   - 必要に応じて、他の DOM 関数をオプションでコールします。これらは、通常、ノード関数または出力関数です。これらの関数によって、DOM 文書が出力されます。
   - 完了後、プロセスが `xmlterm()` をコールします。
   - オプションで、最初に `xmlclean()` をコールして、解析プロセス中に作成されたすべてのデータ構造を削除できます。その後、`xmlterm()` をコールします。

   SAX: SAX インタフェースを使用する場合は、次の手順が含まれます。

   - コールバック関数を介して、`xmlparse()` または `xmlparseBuf()` からの解析の結果を処理します。
   - コールバック関数を登録します。
4. オプションで `xmlclean()` を使用し、解析中に使用したメモリーおよび構造を削除して手順 5 に進みます。または、手順 2 に戻ります。
5. `xmlterm()` を使用して解析プロセスを終了します。

図 24-1 に、XML Parser for C の使用方法の詳細を示します。

XMLParser のコール順序

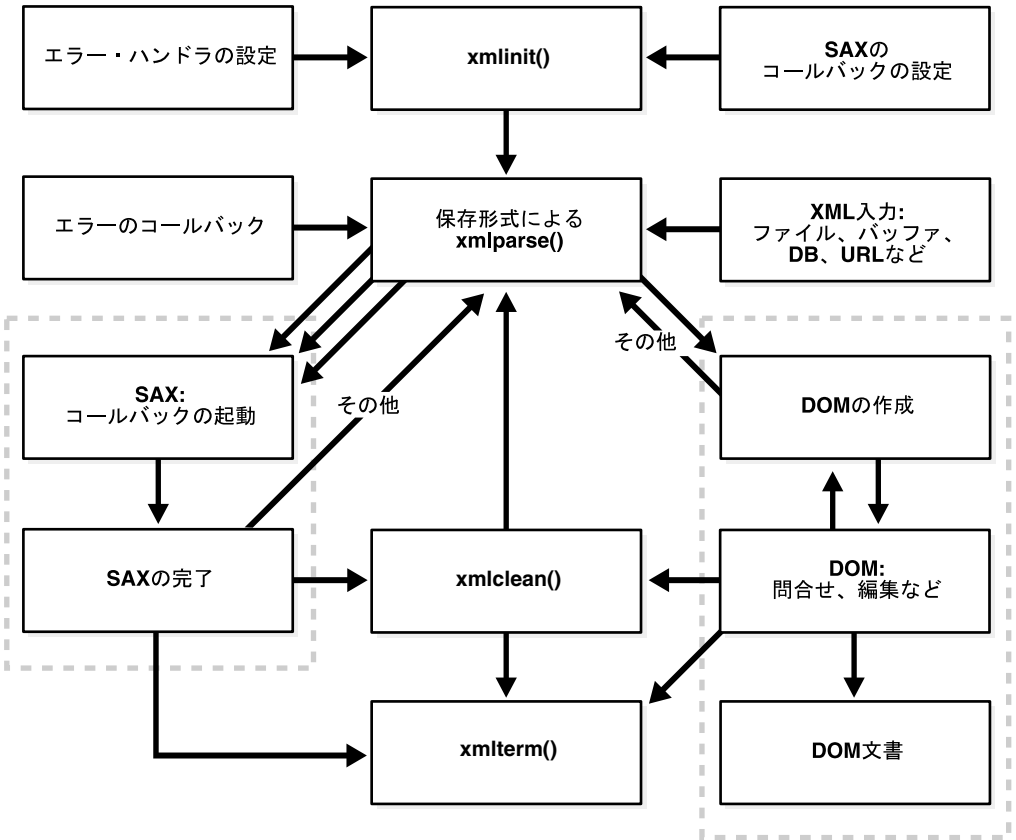
XMLParser に対するコール順序は、次のいずれかになります。

- `xmlinit() - xmlparse()` または

  `xmlparsebuf() - xmlterm()`

- `xmlinit()` - `xmlparse()` または
  `xmlparsebuf()` - `xmlclean()` - `xmlparse()` または
  `xmlparsebuf()` - `xmlclean()` -... - `xmlterm()`
- `xmlinit()` - `xmlparse()` または
  `xmlparsebuf()` - `xmlparse()` または
  `xmlparsebuf()` -... - `xmlterm()`

図 24-1 XML Parser for C のコール順序



XML Parser for C の XSLT（DOM インタフェース）の使用方法

図 24-2 に、XML Parser for C の XSLT 機能を示します。

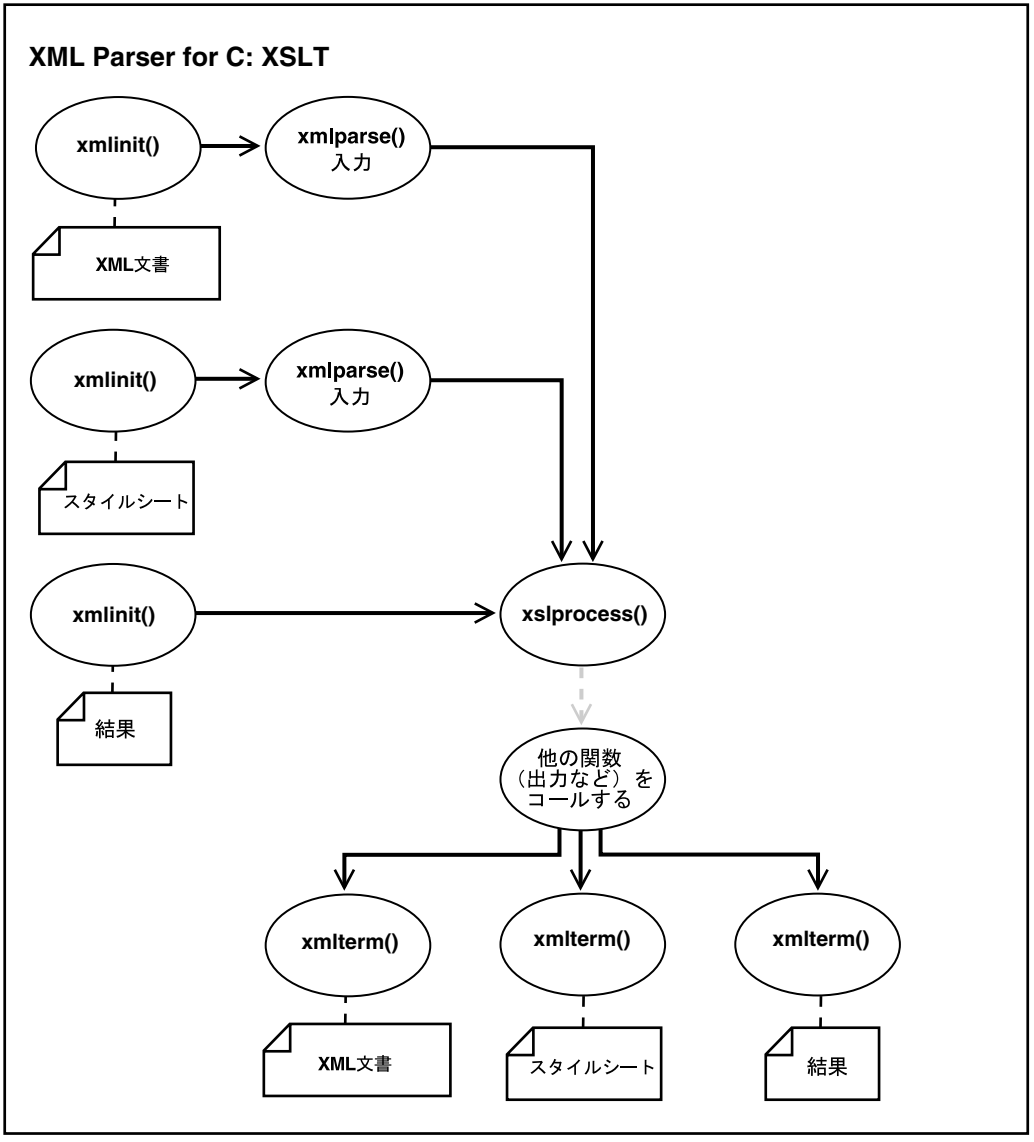
1. `xmlparse()` には、次の 2 つの入力があります。
   - XML 文書に適用するスタイルシート
   - XML 文書

   `xmlparse()` の出力である解析済スタイルシートおよび解析済 XML 文書は、`xslprocess()` 関数へ送られ、処理されます。
2. `xmlinit()` が XSLT プロセスを初期化します。また、`xslprocess()` の結果も初期化します。
3. オプションで、`xslprocess()` が、出力関数などの他の関数をコールします。使用可能な関数のリストは、OTN の Web サイトまたは『Oracle9*i* XML リファレンス』を参照してください。
4. 結果のドキュメント（XML、HTML、VML など）は、通常、アプリケーションへ送られ、さらに処理されます。
5. アプリケーションが、`xmlterm()` を宣言して XSLT プロセスを終了し、XML 文書、スタイルシートおよび最終結果を生成します。

XML Parser for C の XSLT 機能については、次の例を参照してください。

- 24-50 ページの「XML Parser for C の例 16: C - XSLSample.c」
- 24-52 ページの「XML Parser for C の例 17: C - XSLSample.std」

図 24-2 XML Parser for C: XSLT（DOM インタフェース）の使用方法



XML Parser for C のデフォルト動作

XML Parser for C のデフォルト動作は、次のとおりです。

- キャラクタ・セットのエンコーディングは UTF-8 です。ドキュメントがすべて ASCII 形式である場合は、パフォーマンスを向上するために、エンコーディングを US-ASCII に設定することをお薦めします。
- メッセージは、msghdlr が指定されないかぎり、stderr に出力されます。
- saxcb が SAX コールバック API を使用するように設定すると、DOM API がアクセスできる解析ツリーは構築されません。不要な SAX コールバック関数は、すべて NULL に設定できることに注意してください。
- パーサーは、デフォルト動作では、入力が整形式であることは確認しますが、妥当であるかどうかは確認しません。フラグ XML_FLAG_VALIDATE を設定することで、入力の妥当性を検証できます。空白処理のデフォルト動作は、XML 1.0 仕様に完全に準拠しています。すべての空白は、無視できる空白が明示された状態でアプリケーションに通知されます。ただし、アプリケーションによっては、XML_FLAG_DISCARD_WHITESPACE を設定し、要素の終了タグと次の要素の開始タグの間のすべての空白を削除する方が適切な場合もあります。

注意： シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 25% 向上します。

DOM API および SAX API

Oracle XML Parser for C は、XML 文書が整形式であるかどうか、および DTD に対して妥当であるかどうか（オプション）を確認します。このパーサーは、次のいずれかのインタフェースでアクセス可能なオブジェクト・ツリーを構築します。

- DOM インタフェース
- SAX インタフェースを介した順次操作

これらの XML API の説明を次に示します。

- DOM: ツリーベース API。ツリーベース API は、XML 文書を内部ツリー構造にコンパイルします。これによって、アプリケーションは、XML および HTML ドキュメント用のツリーベースの標準 API である DOM を使用してツリー内をナビゲートできます。

- SAX: イベントベース API。イベントベース API は、コールバックを使用して、要素の開始や終了などの解析イベントをアプリケーションに直接通知します。通常は、内部ツリーを構築しません。アプリケーションは、ハンドラを実装して様々なイベントを処理します。これは、Graphical User Interface（GUI）によるイベントの処理に類似しています。

ツリーベース API は広範囲なアプリケーションで有効ですが、特に文書のサイズが大きい場合、より多くのシステム・リソースが消費される場合があります（詳細に制御された環境では、簡単な方法でツリーを構築し、この問題のいくつかを回避できる場合もあります）。さらに、いくつかのアプリケーションではそれぞれ独自のデータ・ツリーを構築する必要があります。この場合、新規のツリーにマップするためにのみ解析ノードのツリーを構築することは、非効率的です。

どちらの場合も、イベントベース API は、XML 文書に対してより単純で低レベルのアクセスを提供します。したがって、使用可能なシステム・メモリーよりサイズの大きい文書を解析し、コールバック・イベント・ハンドラを使用して独自のデータ構造を構築できます。

SAX API の使用

SAX を使用するために、xmlsaxcb 構造が関数ポインタによって初期化され、xmlinit() コールに渡されます。ユーザー定義のコンテキスト構造に対するポインタを含むこともできます。コンテキストのポインタは、各 SAX 関数に渡されます。

SAX コールバック構造

SAX コールバック構造を次に示します。

```
typedef struct
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
             const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
             const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
             const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
             const oratext *publicId,
             const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
             const oratext *local, const oratext *nsp,
             const struct xmlnodes *attrs);
} xmlsaxcb;
```

DOM API の使用

24-17 ページの「XML Parser for C の例 7: C - DOMSample.std」を参照してください。

XML Parser for C の起動

XML Parser for C は、次の 2 つの方法で起動できます。

- コマンドラインで実行可能ファイルを起動する
- C コードを記述し、提供される API を使用する

コマンドラインの使用方法

XML Parser for C は、`bin/xml` をコールすることによって、実行可能ファイルとしてコールできます。

表 24-2 に、コマンドライン・オプションを示します。

表 24-2 XML Parser for C: コマンドライン・オプション

オプション	説明
-c	規格一致性の確認のみ。妥当性は検証しません。
-e encoding	入力ファイルのエンコーディングを指定します。
-h	ヘルプ - 使用方法のヘルプを表示します。
-n	数値 - DOM ツリーを検索し、要素の数をレポートします。
-p	解析後にドキュメントおよび DTD 構造を出力します。
-x	SAX インタフェースを実行し、ドキュメントを出力します。
-v	バージョン - パーサーのバージョンを表示し、終了します。
-w	空白 - すべての空白を保持します。

提供される API を使用するための C コードの記述

XML Parser for C は、C コードを記述し、提供される API を使用することによって起動することもできます。コードは、`include/` サブディレクトリにあるヘッダーを使用してコンパイルし、`lib/` サブディレクトリ内のライブラリにリンクする必要があります。プログラムの作成方法の詳細は、`sample/` サブディレクトリにある `Makefile` を参照してください。

ソフトウェアに含まれるサンプル・ファイルの使用

$ORACLE_HOME/xdk/c/parser/sample/ ディレクトリには、DOM インタフェースおよび SAX インタフェースによる XML Parser for C の使用方法を示す XML アプリケーションがあります。

表 24-3 に、sample/ ディレクトリにあるサンプル・ファイルを示します。

表 24-3 XML Parser for C のサンプル・ファイル

サンプル・ファイル名	説明
DOMNamespace.c	DOMNamespace プログラムのソース
DOMNamespace.std	DOMNamespace からの予想される出力
DOMSample.c	DOMSample プログラムのソース
DOMSample.std	DOMSample からの予想される出力
FullDOM.c	DOM インタフェースの使用例
FullDOM.std	FullDOM からの予想される出力
Make.bat	サンプル・プログラムを作成するためのバッチ・ファイル
NSExample.xml	名前空間を使用したサンプル XML ファイル
SAXNamespace.c	SAXNamespace プログラムのソース
SAXNamespace.std	SAXNamespace からの予想される出力
SAXSample.c	SAXSample プログラムのソース
SAXSample.std	SAXSample からの予想される出力
XSLSample.c	XSLSample プログラムのソース
XSLSample.std	XSLSample からの予想される出力
class.xml	XSLSample で使用できる XML ファイル
iden.xsl	XSLSample で使用できるスタイルシート
cleo.xml	シェイクスピアの戯曲（『アントニーとクレオパトラ』）の XML バージョン

XML Parser for C サンプル・プログラムの実行

サンプル・プログラムの作成

sample/ ディレクトリに移動し、README ファイルを参照してください。このファイルには、サンプル・プログラムの作成方法がプラットフォームごとに記載されています。

サンプル・プログラム

表 24-4 に、sample/ ディレクトリにあるサンプル・ファイルによって作成されたプログラムを示します。

表 24-4 XML Parser for C: sample/ ファイルで作成されたサンプル・プログラム

作成されたプログラム	説明
DOMSample	DOM API を使用するサンプル・アプリケーション。戯曲（cleo.xml）の概要（XML 要素の ACT および SCENE）を表示します。
SAXSample [word]	SAX API を使用するサンプル・アプリケーション。ワードが指定されると、戯曲（cleo.xml）内の、そのワードを含むすべてのせりふを表示します。ワードを指定しない場合は、「death」が使用されます。
DOMNamespace	DOM インタフェースを使用する以外は、SAXNamespace と同じです。
SAXNamespace	SAX API に名前空間による拡張を使用するサンプル・アプリケーション。名前空間の完全な情報とともに、NSExample.xml のすべての要素および属性を出力します。
FullDOM	DOM インタフェース全体の使用例。すべてのコールを実行しますが、それ以外の操作は行われません。
XSLSample <xmlfile> <xsl ss>	XSL プロセッサの使用例。入力として、XML ファイルおよび XSL スタイルシートの 2 つのファイル名を取ります。

XML Parser for C の例 1: XML - class.xml

class.xml は、XSLSample.c を入力する XML ファイルです。

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for C の例 2: XML - cleo.xml

cleo.xml は、DOMSample.c および SAXSample.c を入力します。

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
    <!ELEMENT PLAY     (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
PROLOGUE?, ACT+, EPILOGUE?)>
    <!ELEMENT TITLE    (#PCDATA)>
    <!ELEMENT FM       (P+)>
    <!ELEMENT P        (#PCDATA)>
    <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
    <!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
    <!ELEMENT PERSONA  (#PCDATA)>
    <!ELEMENT GRPDESCR (#PCDATA)>
    <!ELEMENT SCNDESCR (#PCDATA)>
    <!ELEMENT PLAYSUBT (#PCDATA)>
    <!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
    <!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
    <!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
```

```
    <!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT SPEAKER  (#PCDATA)>
    <!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
    <!ELEMENT STAGEDIR (#PCDATA)>
    <!ELEMENT SUBTITLE (#PCDATA)>
    <!ELEMENT SUBHEAD  (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>

<PERSONAE>
<TITLE>Dramatis Personae</TITLE>


<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>


<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>
```

```
...
...

<SCNDESCR>SCENE  In several parts of the Roman empire.</SCNDESCR>

<PLAYSUBT>ANTONY AND CLEOPATRA</PLAYSUBT>

<ACT><TITLE>ACT I</TITLE>

<SCENE><TITLE>SCENE I.  Alexandria. A room in CLEOPATRA's palace.</TITLE>
<STAGEDIR>Enter DEMETRIUS and PHILO</STAGEDIR>

<SPEECH>
<SPEAKER>PHILO</SPEAKER>
<LINE>Nay, but this dotage of our general's</LINE>
<LINE>O'erflows the measure: those his goodly eyes,</LINE>
<LINE>That o'er the files and musters of the war</LINE>
<LINE>Have glow'd like plated Mars, now bend, now turn,</LINE>
<LINE>The office and devotion of their view</LINE>
<LINE>Upon a tawny front: his captain's heart,</LINE>
<LINE>Which in the scuffles of great fights hath burst</LINE>
<LINE>The buckles on his breast, reneges all temper,</LINE>
<LINE>And is become the bellows and the fan</LINE>
<LINE>To cool a gipsy's lust.</LINE>
<STAGEDIR>Flourish. Enter ANTONY, CLEOPATRA, her Ladies,
the Train, with Eunuchs fanning her</STAGEDIR>
<LINE>Look, where they come:</LINE>
<LINE>Take but good note, and you shall see in him.</LINE>
<LINE>The triple pillar of the world transform'd</LINE>
<LINE>Into a strumpet's fool: behold and see.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>If it be love indeed, tell me how much.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>There's beggary in the love that can be reckon'd.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>I'll set a bourn how far to be beloved.</LINE>
</SPEECH>
```

```
<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>Then must thou needs find out new heaven, new earth.</LINE>
</SPEECH>
...
...
...
<SPEAKER>DOLABELLA</SPEAKER>
<LINE>Here, on her breast,</LINE>
<LINE>There is a vent of blood and something blown:</LINE>
<LINE>The like is on her arm.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>
```

XML Parser for C の例 3: XSL - iden.xsl

iden.xsl はスタイルシートの例です。XSLSample.c の入力に使用できます。

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

XML Parser for C の例 4: XML - FullDOM.xml（DTD）

FullDOM.xml は DTD の例です。FullDOM.c を入力します。

```
<!DOCTYPE doc [
    <!ELEMENT p (#PCDATA)>
    <!ATTLIST p xml:space (preserve|default) 'preserve'>
    <!NOTATION notation1 SYSTEM "file.txt">
    <!NOTATION notation2 PUBLIC "some notation">
    <!ELEMENT doc (p*)>
    <!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>">
]>
<doc xml:lang="foo">&example;</doc>
```

XML Parser for C の例 5: XML - NSExample.xml

NSExample.xml は、名前空間を使用します。

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:a1 CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:a1 = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
```

```
<child>
This element inherits the default Namespace of doc.
</child>
</doc>
```

XML Parser for C の例 6: C - DOMSample.c

DOMSample.c には、DOMSample の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */
/*
   NAME
     DOMSample.c - Sample DOM usage
   DESCRIPTION
     Sample usage of C XML parser via DOM interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif
#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT        (oratext *) "cleo.xml"

void dump(xmlctx *ctx, xmlnode *node);
void dumppart(xmlctx *ctx, xmlnode *node, boolean indent);

int main()
{
    xmlctx     *ctx;
    uword       ecode;

    puts("XML C DOM sample");
    puts("Initializing XML package...");
    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
                        (void (*)(void *, const oratext *, uword)) 0,
                        (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
return 1;
```

```
    }

    printf("Parsing '%s' ...\n", DOCUMENT);
    if (ecode = xmlparse(ctx, DOCUMENT, (oratext *) 0,
XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
    {
printf("Parse failed, error %u\n", (unsigned) ecode);
return 1;
    }

    puts("Outlining...");
    dump(ctx, getDocumentElement(ctx));

    xmlterm(ctx);

    return 0;
}

void dump(xmlctx *ctx, xmlnode *node)
{
    const oratext *name;
    void    *nodes;
    uword    i, n_nodes;

    name = getNodeName(node);
    if (!strcmp((char *) name, "ACT"))
        dumppart(ctx, node, FALSE);
    else if (!strcmp((char *) name, "SCENE"))
        dumppart(ctx, node, TRUE);
    if (hasChildNodes(node))
    {
        nodes = getChildNodes(node);
        n_nodes = numChildNodes(nodes);
        for (i = 0; i < n_nodes; i++)
            dump(ctx, getChildNode(nodes, i));
    }
}

void dumppart(xmlctx *ctx, xmlnode *node, boolean indent)
{
    void *title = getFirstChild(node);

    if (indent)
       fputs("    ", stdout);
    puts((char *) getNodeValue(getFirstChild(title)));
}

/* end of DOMSample.c */
```

XML Parser for C の例 7: C - DOMSample.std

DOMSample.std は、DOMSample.c からの予想される出力を表示します。

```
XML C DOM sample
Initializing XML package...
Parsing 'cleo.xml' ...
Outlining...
ACT I
    SCENE I.  Alexandria. A room in CLEOPATRA's palace.
    SCENE II.  The same. Another room.
    SCENE III.  The same. Another room.
    SCENE IV.  Rome. OCTAVIUS CAESAR's house.
    SCENE V.  Alexandria. CLEOPATRA's palace.
ACT II
    SCENE I.  Messina. POMPEY's house.
    SCENE II.  Rome. The house of LEPIDUS.
...
...
...
 ACT V
    SCENE I.  Alexandria. OCTAVIUS CAESAR's camp.
    SCENE II.  Alexandria. A room in the monument.
```

XML Parser for C の例 8: C - SAXSample.c

SAXSample.c には、SAXSample の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
   NAME
     SAXSample.c - Sample SAX usage

   DESCRIPTION
     Sample usage of C XML parser via SAX interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif
```

```
#define DOCUMENT"cleo.xml"
#define DEFAULT_KEYWORD"death"

char    *keyword;
size_t   keylen;
oratext *elem;
oratext speaker[80];

oratext *findsub(oratext *buf, size_t bufsiz, oratext *sub, size_t subsiz);
void     savestr(oratext *buf, oratext *s, size_t len);

/* SAX callback functions */

sword startDocument(void *ctx);
sword endDocument(void *ctx);
sword startElement(void *ctx, const oratext *name,
                   const struct xmlnodes *attrs);
sword endElement(void *ctx, const oratext *name);
sword characters(void *ctx, const oratext *ch, size_t len);

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    startElement,
    endElement,
    characters
};

int main(int argc, char **argv)
{
    xmlctx     *ctx;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    puts("XML C SAX sample");
    keyword = (argc > 1) ? argv[1] : DEFAULT_KEYWORD;
    keylen = strlen(keyword);
    puts("Initializing XML package...");

    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
(void (*)(void *, const oratext *, uword)) 0,
(void *) 0, &saxcb, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
```

```
        (void) printf("Failed to initialize XML parser, error %u\n",
                      (unsigned) ecode);
        return 1;
    }

    printf("Parsing '%s' and looking for lines containing '%s'...\n",
DOCUMENT, keyword);
    elem = (oratext *) "";
    if (ecode = xmlparse(ctx, (oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    (void) xmlterm(ctx);/* terminate XML package */

    return 0;
}

sword startDocument(void *ctx)
{
    puts("startDocument");
    return 0;
}

sword endDocument(void *ctx)
{
    puts("endDocument");
    return 0;
}

sword startElement(void *ctx, const oratext *name,
                   const struct xmlnodes *attrs)
{
    elem = (oratext *) name;
    return 0;
}

sword endElement(void *ctx, const oratext *name)
{
    elem = (oratext *) "";
    return 0;
}

sword characters(void *ctx, const oratext *ch, size_t len)
{
    if (!strcmp((char *) elem, "SPEAKER"))
savestr(speaker, (oratext *) ch, len);
    else if (findsub((oratext *) ch, len, (oratext *) keyword, keylen))
printf("    %s: %.*s\n", speaker, len, ch);
```

```
    return 0;
}

oratext *findsub(oratext *buf, size_t bufsiz, oratext *sub, size_t subsiz)
{
    uword i;

    if (!buf || !bufsiz || (subsiz > bufsiz))
return (oratext *) 0;
    if (!sub || !subsiz)
return buf;
    for (i = 0; i < bufsiz - subsiz; i++, buf++)
    {
if (!memcmp(buf, sub, subsiz))
    return buf;
    }
    return (oratext *) 0;
}

void savestr(oratext *buf, oratext *s, size_t len)
{
    memcpy(buf, s, len);
    buf[len] = 0;
}

/* End of SAXSample.c */
```

XML Parser for C の例 9: C - SAXSample.std

SAXSample.std は、SAXSample.c からの予想される出力を表示します。

```
XML C SAX sample
Initializing XML package...
Parsing 'cleo.xml' and looking for lines containing 'death'...
startDocument
    MARK ANTONY: Who tells me true, though in his tale lie death,
    DOMITIUS ENOBARBUS: if they suffer our departure, death's the word.
    DOMITIUS ENOBARBUS: mettle in death, which commits some loving act upon
    MARK ANTONY: The death of Fulvia, with more urgent touches,
    MARK ANTONY: Is Fulvia's death.
    CLEOPATRA: In Fulvia's death, how mine received shall be.
    EROS: the poor third is up, till death enlarge his confine.
    SCARUS: Where death is sure. Yon ribaudred nag of Egypt,--
    EROS: Her head's declined, and death will seize her, but
    MARK ANTONY: I'll make death love me; for I will contend
    MARK ANTONY: Married to your good service, stay till death:
```

```
    MARK ANTONY: Than death and honour. Let's to supper, come,
    First Soldier: The hand of death hath raught him.
    CLEOPATRA: And bring me how he takes my death.
    MARK ANTONY: She hath betray'd me and shall die the death.
    MARK ANTONY: Than she which by her death our Caesar tells
    EROS: Of Antony's death.
    MARK ANTONY: A bridegroom in my death, and run into't
    DERCETAS: Thy death and fortunes bid thy followers fly.
    MARK ANTONY: Sufficing strokes for death.
    DIOMEDES: His death's upon him, but not dead.
    MARK ANTONY: I here importune death awhile, until
    CLEOPATRA: To rush into the secret house of death,
    CLEOPATRA: Ere death dare come to us? How do you, women?
    CLEOPATRA: And make death proud to take us. Come, away:
    OCTAVIUS CAESAR: And citizens to their dens: the death of Antony
    CLEOPATRA: What, of death too,
    CLEOPATRA: Where art thou, death?
    CLEOPATRA: The stroke of death is as a lover's pinch,
    CHARMIAN: Now boast thee, death, in thy possession lies
    OCTAVIUS CAESAR: Took her own way. The manner of their deaths?
endDocument
```

XML Parser for C の例 10: C - DOMNamespace.c

DOMNamespace.c には、DOMNamespace の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/**
 ** This file demonstates a simple use of the parser and Namespace
 ** extensions to the DOM APIs.
 ** The XML file that is given to the application is parsed and the
 ** elements and attributes in the document are printed.
 **/

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT         "NSExample.xml"
```

```
/*---------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------*/
static void    DOMNSprint(xmlctx *ctx);
static void    printElements(xmlctx *ctx, xmlnode *n);
static void    printAttrs(xmlctx *ctx, xmlnode *n);

/*---------------------------------------------------------------------
                                MAIN
  ---------------------------------------------------------------------*/
int main()
{
   xmlctx     *ctx;
   oratext    *encoding, *doc;
   void       *saxcbctx;
   const xmlsaxcb *saxcb;
   uword       ecode;
   ub4         flags;

   encoding = doc = (oratext *) 0;
   saxcbctx = (void *) 0;
   saxcb = (const xmlsaxcb *) 0;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
   doc = (oratext *)DOCUMENT;

   /* initialize LPX context */
   if (!(ctx = xmlinit(&ecode, encoding,
                       (void (*)(void *, const oratext *, uword)) 0,
                       (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
                       (void *) 0, (const oratext *) 0)))
   {
      printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
      return -1;
   }

   /* parse the document */

   printf("\nParsing '%s' ...\n", doc);
   ecode = xmlparse(ctx, doc, encoding, flags);

   if (ecode)
     printf("Parse failed, code %u\n", (unsigned) ecode);
   else
     printf("Parse succeeded.\n");

   /* print results */
```

```
   printf("Printing results ...\n");
   DOMNSprint(ctx);

   /* terminate */

   (void) xmlterm(ctx);

   return (ecode ? -1 : 0);
}

/*---------------------------------------------------------------------------
                            DOMNSprint
  ---------------------------------------------------------------------------*/
static void DOMNSprint(xmlctx *ctx)
{
    xmlnode *root;

    root = getDocumentElement(ctx);
    printf("\nThe elements are:\n");
    printElements(ctx, root);
}

/*---------------------------------------------------------------------------
                            printElements
  ---------------------------------------------------------------------------*/
static void printElements(xmlctx *ctx, xmlnode *n)
{
    xmlnodes *nodes;
    uword     i;
    size_t    nn;

    const oratext   *qname;
    const oratext   *namespace;
    const oratext   *local;
    const oratext   *prefix;

    if (n == (xmlnode*)NULL)
       return;

    if (nodes = getChildNodes(n))
    {
       for (nn = numChildNodes(nodes), i = 0; i < nn; i++)
       {
          /* get node qualified name, local name, namespace, and prefix */

          qname = namespace = local = prefix = (oratext*)" ";
```

```
            if (getNodeQualifiedName(n) != (oratext*)NULL)
               qname = getNodeQualifiedName(n);
              if (getNodePrefix(n) != (oratext*)NULL)
               prefix = getNodePrefix(n);

            if (getNodeLocal(n) != (oratext *)NULL)
               local = getNodeLocal(n);

            if (getNodeNamespace(n) != (oratext*)NULL)
               namespace = getNodeNamespace(n);

            printf("  ELEMENT Qualified Name: %s\n", qname);
            printf("  ELEMENT Prefix Name   : %s\n", prefix);
            printf("  ELEMENT Local Name    : %s\n", local);
            printf("  ELEMENT Namespace     : %s\n", namespace);

            printAttrs(ctx, n);
            printElements(ctx, (xmlnode *) getChildNode(nodes, i));
         }
      }
}

/*---------------------------------------------------------------------------
                              printAttrs
  ---------------------------------------------------------------------------*/
static void printAttrs(xmlctx *ctx, xmlnode *n)
{
    xmlnodes  *attrs;
    xmlnode   *a;
    uword      i;
    size_t     na;

    const oratext   *value;
    const oratext   *qname;
    const oratext   *namespace;
    const oratext   *local;
    const oratext   *prefix;

    if (attrs = getAttributes(n))
    {
       printf("\n    ATTRIBUTES: \n");
       for (na = numAttributes(attrs), i = 0; i < na; i++)
       {
          /* get attr qualified name, local name, namespace, and prefix */

          a = getAttributeIndex(attrs, i);
```

```
            qname = namespace = local = prefix = value = (oratext*)" ";

            if (getAttrQualifiedName(a) != (oratext*)NULL)
               qname = getAttrQualifiedName(a);
            if (getAttrNamespace(a) != (oratext*)NULL)
               namespace = getAttrNamespace(a);
            if (getAttrLocal(a) != (oratext*)NULL)
               local = getAttrLocal(a);
            if (getAttrPrefix(a) != (oratext*)NULL)
               prefix = getAttrPrefix(a);
            if (getAttrValue(a) != (oratext*)NULL)
               value = getAttrValue(a);

            printf("      %s = %s\n", qname, value);
            printf("      Namespace : %s\n", namespace);
            printf("      Local Name: %s\n", local);
            printf("      Prefix    : %s\n\n", prefix);
         }
      }
      printf("\n");
}
```

XML Parser for C の例 11: C - DOMNamespace.std

DOMNamespace.std は、DOMNamespace.c からの予想される出力を表示します。

```
Parsing 'NSExample.xml' ...
Parse succeeded.
Printing results ...

The elements are:
  ELEMENT Qualified Name: doc
  ELEMENT Prefix Name   :
  ELEMENT Local Name    : doc
  ELEMENT Namespace     : http://www.w3c.org

    ATTRIBUTES:
      nsprefix:a1 = v1
      Namespace : http://www.oracle.com
      Local Name: a1
      Prefix    : nsprefix

      xmlns = http://www.w3c.org
      Namespace :
      Local Name: xmlns
```

```
      Prefix    :

      xmlns:nsprefix = http://www.oracle.com
      Namespace :
      Local Name: nsprefix
      Prefix    : xmlns


  ELEMENT Qualified Name: child
  ELEMENT Prefix Name   :
  ELEMENT Local Name    : child
  ELEMENT Namespace     : http://www.w3c.org
```

XML Parser for C の例 12: C - SAXNamespace.c

SAXNamespace.c には、SAXNamespace の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/**
 ** This file demonstrates a simple use of the Namespace extensions to
 ** the SAX APIs.
 **/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

#define DOCUMENT         "NSExample.xml"

/*---------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------*/

static int sax_startdocument(void *ctx);
static int sax_enddocument(void *ctx);
static int sax_endelement(void *ctx, const oratext *name);
static int sax_nsstartelement(void *ctx, const oratext *qname,
                              const oratext *local,
                              const oratext *namespace,
```

```
                                const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb sax_callback = {
    sax_startdocument,
    sax_enddocument,
    0,
    sax_endelement,
    0,
    0,
    0,
    0,
    0,
    sax_nsstartelement,
    0, 0, 0, 0, 0, 0, 0, 0
};

/* SAX callback context */

typedef struct {
    xmlctx  *ctx;
    uword    depth;
} cbctx;


/*---------------------------------------------------------------------------
                                MAIN
  ---------------------------------------------------------------------------*/

int main()
{
   xmlctx     *ctx;
   uword       i;
   oratext    *doc, *encoding;
   xmlsaxcb   *saxcb;
   cbctx       saxctx;
   void       *saxcbctx;
   ub4         flags;
   uword       ecode;

   doc = encoding = (oratext *)0;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

   doc = (oratext *)DOCUMENT;

   /* set up SAX callbacks */
```

```
   saxcb = &sax_callback;
   saxcbctx = (void *) &saxctx;

   /* initialize LPX context */
   if (!(ctx = xmlinit(&ecode, encoding,
                       (void (*)(void *, const oratext *, uword)) 0,
                       (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
                       (void *) 0, (const oratext *) 0)))
   {
      printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
      return -1;
   }

   /* parse the document */

   printf("\nParsing '%s' ...\n", doc);
   ecode = xmlparse(ctx, doc, encoding, flags);

   if (ecode)
     printf("\nParse failed, code %u\n", (unsigned) ecode);
   else
     printf("\nParse succeeded.\n");

   /* terminate */

   (void) xmlterm(ctx);

   return (ecode ? -1 : 0);
}


/*---------------------------------------------------------------------------
                            SAX Interface
  ---------------------------------------------------------------------------*/

static int sax_startdocument(void *ctx)
{
    printf("\nStartDocument\n\n");
    return 0;
}


static int sax_enddocument(void *ctx)
{
    printf("\nEndDocument\n");
    return 0;
```

```
}


static int sax_endelement(void *ctx, const oratext *name)
{
    printf("\nELEMENT Name  : %s\n", name);
    return 0;
}


static int sax_nsstartelement(void *ctx, const oratext *qname,
                              const oratext *local,
                              const oratext *namespace,
                              const struct xmlnodes *attrs)
{
    cbctx   *saxctx = (cbctx *) ctx;
    xmlnode *attr;
    size_t   i;

    const oratext *aqname;
    const oratext *aprefix;
    const oratext *alocal;
    const oratext *anamespace;
    const oratext *avalue;

    /*
     * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
     * getXXXNamespace() to get Namespace information.
     */

    if (qname == (oratext*)NULL)
       qname = (oratext*)" ";
    if (local == (oratext*)NULL)
       local = (oratext*)" ";
    if (namespace == (oratext*)NULL)
       namespace = (oratext*)" ";

    printf("ELEMENT Qualified Name: %s\n", qname);
    printf("ELEMENT Local Name    : %s\n", local);
    printf("ELEMENT Namespace     : %s\n", namespace);

    if (attrs)
    {
       for (i = 0; i < numAttributes(attrs); i++)
       {
          attr = getAttributeIndex(attrs, i);
```

```
            aqname = aprefix = alocal = anamespace = avalue = (oratext*)" ";

            if (getAttrQualifiedName(attr) != (oratext*)NULL)
               aqname = getAttrQualifiedName(attr);

            if (getAttrPrefix(attr) != (oratext*)NULL)
               aprefix = getAttrPrefix(attr);

            if (getAttrLocal(attr) != (oratext*)NULL)
               alocal = getAttrLocal(attr);

            if (getAttrNamespace(attr) != (oratext*)NULL)
               anamespace = getAttrNamespace(attr);

            if (getAttrValue(attr) != (oratext*)NULL)
               avalue = getAttrValue(attr);

            printf(" ATTRIBUTE Qualified Name   : %s\n", aqname);
            printf(" ATTRIBUTE Prefix           : %s\n", aprefix);
            printf(" ATTRIBUTE Local Name       : %s\n", alocal);
            printf(" ATTRIBUTE Namespace        : %s\n", anamespace);
            printf(" ATTRIBUTE Value            : %s\n", avalue);
            printf("\n");
         }
      }
      return 0;
}
```

XML Parser for C の例 13: C - SAXNamespace.std

SAXNamespace.std は、SAXNamespace.c からの予想される出力を表示します。

```
Parsing 'NSExample.xml' ...

StartDocument

ELEMENT Qualified Name: doc
ELEMENT Local Name    : doc
ELEMENT Namespace     : http://www.w3c.org
 ATTRIBUTE Qualified Name   : nsprefix:a1
 ATTRIBUTE Prefix           : nsprefix
 ATTRIBUTE Local Name       : a1
 ATTRIBUTE Namespace        : http://www.oracle.com
 ATTRIBUTE Value            : v1
```

```
 ATTRIBUTE Qualified Name   : xmlns
 ATTRIBUTE Prefix           :
 ATTRIBUTE Local Name       : xmlns
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.w3c.org

 ATTRIBUTE Qualified Name   : xmlns:nsprefix
 ATTRIBUTE Prefix           : xmlns
 ATTRIBUTE Local Name       : nsprefix
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.oracle.com

ELEMENT Qualified Name: child
ELEMENT Local Name    : child
ELEMENT Namespace     : http://www.w3c.org

ELEMENT Name  : child

ELEMENT Name  : doc

EndDocument

Parse succeeded.
```

XML Parser for C の例 14: C - FullDOM.c

FullDOM.c には、FullDOM の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*
   NAME
     FullDOM.c

   DESCRIPTION
     Sample code to test full DOM interface
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
```

```
#endif

#define TEST_DOCUMENT(oratext *) "FullDOM.xml"

void dump(xmlnode *node, uword level);
void dumpnode(xmlnode *node, uword level);

static char *ntypename[] = {
    "0",
    "ELEMENT",
    "ATTRIBUTE",
    "TEXT",
    "CDATA",
    "ENTREF",
    "ENTITY",
    "PI",
    "COMMENT",
    "DOCUMENT",
    "DTD",
    "DOCFRAG",
    "NOTATION"
};

#define FAIL { puts("Failed!"); exit(1); }

int main()
{
    xmlctx     *ctx;
    xmldtd     *dtd;
    xmlnode    *doc, *elem, *node, *text, *pi, *comment, *entref,
*subelem, *subtext, *cdata, *attr1, *attr2, *clone,
*deep_clone, *frag, *fragelem, *fragtext, *sub2,
*fish, *food, *gleep1, *gleep2, *repl;
    xmlnodes   *subs, *nodes, *attrs, *notes, *entities;
    uword       i, ecode, level;

    puts("XML C Full DOM test");

    puts("Initializing XML parser...");

    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
                        (void (*)(void *, const oratext *, uword)) 0,
                        (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
(const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
```

```
return 1;
    }

    puts("\nCreating new document...");
    if (!(doc = createDocument(ctx)))
FAIL

    puts("Document from root node:");
    dump(getDocument(ctx), 0);

    puts("\nCreating 'ROOT' element...");
    if (!(elem = createElement(ctx, (oratext *) "ROOT")))
FAIL

    puts("Setting as 'ROOT' element...");
    if (!appendChild(ctx, doc, elem))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("Adding 7 children to 'ROOT' element...");
    if (!(text = createTextNode(ctx, (oratext *) "Gibberish")) ||
        !appendChild(ctx, elem, text))
FAIL

    if (!(comment = createComment(ctx, (oratext*) "Bit warm today, innit?")) ||
        !appendChild(ctx, elem, comment))
FAIL

    if (!(pi = createProcessingInstruction(ctx, (oratext *) "target",
(oratext *) "PI-contents")) ||
        !appendChild(ctx, elem, pi))
FAIL

    if (!(cdata = createCDATASection(ctx, (oratext *) "See DATA")) ||
        !appendChild(ctx, elem, cdata))
FAIL

    if (!(entref = createEntityReference(ctx, (oratext *) "EntRef")) ||
        !appendChild(ctx, elem, entref))
FAIL

    if (!(fish = createElement(ctx, (oratext *) "FISH")) ||
!appendChild(ctx, elem, fish))
FAIL
```

```
    if (!(food = createElement(ctx, (oratext *) "FOOD")) ||
!appendChild(ctx, elem, food))
FAIL

    puts("Document from 'ROOT' element with its 7 children:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node insertion...");
    puts("Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node
...");
    if (!(node = createTextNode(ctx, (oratext *) "Pre-Gibberish")) ||
        !insertBefore(ctx, elem, node, text))
FAIL

    if (!(node = createComment(ctx, (oratext *) "Ask about the weather:")) ||
        !insertBefore(ctx, elem, node, comment))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node removal by name ...");
    puts("Removing 'FISH' element");
    if (!(nodes = getChildNodes(elem)) ||
!removeNamedItem(nodes, (oratext *) "FISH"))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting nextSibling links starting at first child...");
    for (node = getFirstChild(elem); node; node = getNextSibling(node))
dump(node, 1);

    puts("\nTesting previousSibling links starting at last child...");
    for (node = getLastChild(elem); node; node = getPreviousSibling(node))
dump(node, 1);

    puts("\nTesting setting node value...");
    puts("Original node:");
    dump(pi, 1);
    setNodeValue(pi, (oratext *) "New PI contents");
    puts("Node after new value:");
    dump(pi, 1);

    puts("\nAdding another element level, i.e., 'SUB' ...");
    if (!(subelem = createElement(ctx, (oratext *) "SUB")) ||
```

```
!insertBefore(ctx, elem, subelem, cdata) ||
!(subtext = createTextNode(ctx, (oratext *) "Lengthy SubText")) ||
        !appendChild(ctx, subelem, subtext))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nAdding a second 'SUB' element...");
    if (!(sub2 = createElement(ctx, (oratext *) "SUB")) ||
!insertBefore(ctx, elem, sub2, cdata))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nGetting all SUB nodes - note the distinct hex addresses ...");
    if (!(subs = getElementsByTagName(ctx, (xmlnode *) 0, (oratext *) "SUB")))
FAIL
    for (i = 0; i < getNodeMapLength(subs); i++)
dumpnode(getChildNode(subs, i), 1);

    puts("\nTesting parent links...");
    for (level = 1, node = subtext; node; node = getParentNode(node), level++)
dumpnode(node, level);

    puts("\nTesting owner document of node...");
    dumpnode(subtext, 1);
    dumpnode(getOwnerDocument(subtext), 1);

    puts("\nTesting node replacement...");
    if (!(node = createTextNode(ctx, (oratext *) "REPLACEMENT, 1/2 PRICE")) ||
        !replaceChild(ctx, pi, node))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nTesting node removal...");
    if (!removeChild(entref))
FAIL

    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nNormalizing...");
    normalize(ctx, elem);
```

```
    puts("Document from 'ROOT' element:");
    dump(getDocumentElement(ctx), 0);

    puts("\nCreating and populating document fragment...");
    if (!(frag = createDocumentFragment(ctx)) ||
!(fragelem = createElement(ctx, (oratext *) "FragElem")) ||
!(fragtext = createTextNode(ctx, (oratext *) "FragText")) ||
!appendChild(ctx, frag, fragelem) ||
!appendChild(ctx, frag, fragtext))
FAIL
    dump(frag, 1);

    puts("Insert document fragment...");
    if (!insertBefore(ctx, elem, frag, comment))
FAIL
    dump(elem, 1);

    puts("\nCreate two attributes...");
    if (!(attr1 = createAttribute(ctx,(oratext*)"Attr1",(oratext*)"Value1")) ||
!(attr2 = createAttribute(ctx,(oratext*)"Attr2",(oratext*)"Value2")))
FAIL
    puts("Setting attributes...");
    if (!setAttributeNode(ctx, subelem, attr1, NULL) ||
!setAttributeNode(ctx, subelem, attr2, NULL))
FAIL
    dump(subelem, 1);

    puts("\nAltering attribute1 value...");
    setAttrValue(attr1, (oratext *) "New1");
    dump(subelem, 1);

    puts("\nFetching attribute by name (Attr2)...");
    if (!(node = getAttributeNode(subelem, (oratext *) "Attr2")))
FAIL
    dump(node, 1);

    puts("\nRemoving attribute by name (Attr1)...");
    removeAttribute(subelem, (oratext *) "Attr1");
    dump(subelem, 1);

    puts("\nAdding new attribute...");
    if (!setAttribute(ctx, subelem, (oratext *) "Attr3", (oratext *) "Value3"))
FAIL
    dump(subelem, 1);

    puts("\nRemoving attribute by pointer (Attr2)...");
```

```
    if (!removeAttributeNode(subelem, attr2))
FAIL
    dump(subelem, 1);

    puts("\nAdding new attribute w/same name (test replacement)...");
    dump(subelem, 1);
    if (!(node = createAttribute(ctx, (oratext*)"Attr3", (oratext*)"Zoo3")))
FAIL
    if (!setAttributeNode(ctx, subelem, node, NULL))
FAIL
    dump(subelem, 1);

    puts("\nTesting node (attr) set by name ...");
    puts("Adding 'GLEEP' attribute and printing out hex addresses of node set");
    attrs = getAttributes(subelem);
    if (!(gleep1=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep1")) ||
!setNamedItem(ctx, attrs, gleep1, NULL))
FAIL
    dump(subelem, 1);

    puts("\nTesting node set by name ...");
    puts("Replacing 'GLEEP' attribute - note the changed hex address");
    if (!(gleep2=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep2")) ||
!setNamedItem(ctx, attrs, gleep2, &repl))
FAIL
    dump(subelem, 1);
    puts("Replaced node was:");
    dump(repl, 1);

    puts("\nOriginal SubROOT...");
    dump(subelem, 1);
    puts("Cloned SubROOT (not deep)...");
    clone = cloneNode(ctx, subelem, FALSE);
    dump(clone, 1);
    puts("Cloned SubROOT (deep)...");
    deep_clone = cloneNode(ctx, subelem, TRUE);
    dump(deep_clone, 1);

    puts("\nSplitting text...");
    dump(subelem, 1);
    splitText(ctx, subtext, 3);
    dump(subelem, 1);

    puts("\nTesting string operations...");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Setting new data...");
    setCharData(subtext, (oratext *) "0123456789");
```

```
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    printf("    CharLength = %d\n", getCharLength(subtext));
    printf("    Substring(0,5) = \"%s\"\n",
substringData(ctx, subtext, 0, 5));
    printf("    Substring(8,2) = \"%s\"\n",
substringData(ctx, subtext, 8, 2));
    puts("Appending data...");
    appendData(ctx, subtext, (oratext *) "ABCDEF");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Inserting data...");
    insertData(ctx, subtext, 10, (oratext *) "*foo*");
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Deleting data...");
    deleteData(ctx, subtext, 0, 10);
    printf("    CharData = \"%s\"\n", getCharData(subtext));
    puts("Replacing data...");
    replaceData(ctx, subtext, 1, 3, (oratext *) "bamboozle");
    printf("    CharData = \"%s\"\n", getCharData(subtext));

    puts("Cleaning up...");
    xmlclean(ctx);

    if (getDocument(ctx))
    {
puts("Problem, document is not gone!!");
return 1;
    }

    puts("Parsing test document...");
    if (ecode = xmlparse(ctx, TEST_DOCUMENT, (oratext *) 0, 0))
    {
printf("Parse failed, code %d\n", (int) ecode);
return 1;
    }

    puts("Document from root node:");
    dump(getDocument(ctx), 0);

    dtd = getDocType(ctx);

    puts("Testing getDocTypeNotations...");
    if (notes = getDocTypeNotations(dtd))
    {
size_t n_notes = numChildNodes(notes);

printf("# of notations = %d\n", (int) n_notes);
for (i = 0; i < n_notes; i++)
```

```
    dump(getChildNode(notes, i), 1);
    }
    else
puts("No defined notations\n");

    puts("Testing getDocTypeEntities...");
    if (entities = getDocTypeEntities(dtd))
    {
size_t n_entities = numChildNodes(entities);

printf("# of entities = %d\n", (int) n_entities);
for (i = 0; i < n_entities; i++)
    dump(getChildNode(entities, i), 1);
    }
    else
puts("No defined entities\n");

    puts("Cleaning up...");
    xmlclean(ctx);

    if (getDocument(ctx))
    {
puts("Problem, document is not gone!!\n");
return 1;
    }

    puts("\nTerminating parser...");
    xmlterm(ctx);

    puts("Success.");
    return 0;
}

void dump(xmlnode *node, uword level)
{
    xmlnodes *nodes;
    uword     i, n_nodes;

    if (node)
    {
dumpnode(node, level);
if (hasChildNodes(node))
{
    nodes = getChildNodes(node);
    n_nodes = numChildNodes(nodes);
    for (i = 0; i < n_nodes; i++)
dump(getChildNode(nodes, i), level + 1);
```

```
}
   }


void dumpnode(xmlnode *node, uword level)
{
    const oratext  *name, *value;
    xmlntype  type;
    xmlnodes *attrs;
    xmlnode  *attr;
    uword     i, n_attrs;

    if (node)
    {
for (i = 0; i <= level; i++)
    fputs("    ", stdout);
type = getNodeType(node);
fputs(ntypename[type], stdout);
if ((name = getNodeName(node)) && (*name != '#'))
    printf(" \"%s\"", (char *) name);
if (value = getNodeValue(node))
    printf(" = \"%s\"", (char *) value);
if ((type == ELEMENT_NODE) && (attrs = getAttributes(node)))
{
    fputs(" [", stdout);
    n_attrs = numAttributes(attrs);
    for (i = 0; i < n_attrs; i++)
    {
if (i) fputs(", ", stdout);
attr = getAttributeIndex(attrs, i);
fputs((char *) getAttrName(attr), stdout);
if (getAttrSpecified(attr))
    putchar('*');
printf("=\"%s\"", (char *) getAttrValue(attr));
    }
    putchar(']');
}
putchar('\n');
    }
}

/* end of FullDOM.c */
```

XML Parser for C の例 15: C - FullDOM.std

FullDOM.std は、FullDOM.c からの予想される出力を表示します。

```
XML C Full DOM test
Initializing XML parser...

Creating new document...
Document from root node:
    DOCUMENT

Creating 'ROOT' element...
Setting as 'ROOT' element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
    ELEMENT "ROOT"
        TEXT = "Gibberish"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node ...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node removal by name ...
Removing 'FISH' element
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
```

```
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Testing nextSibling links starting at first child...
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Testing previousSibling links starting at last child...
        ELEMENT "FOOD"
        ENTREF "EntRef"
        CDATA = "See DATA"
        PI "target" = "PI-contents"
        COMMENT = "Bit warm today, innit?"
        COMMENT = "Ask about the weather:"
        TEXT = "Gibberish"
        TEXT = "Pre-Gibberish"

Testing setting node value...
Original node:
        PI "target" = "PI-contents"
Node after new value:
        PI "target" = "New PI contents"

Adding another element level, i.e., 'SUB' ...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Adding a second 'SUB' element...
Document from 'ROOT' element:
```

```
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Getting all SUB nodes - note the distinct hex addresses ...
        ELEMENT "SUB"
        ELEMENT "SUB"

Testing parent links...
        TEXT = "Lengthy SubText"
            ELEMENT "SUB"
                ELEMENT "ROOT"
                    DOCUMENT

Testing owner document of node...
        TEXT = "Lengthy SubText"
        DOCUMENT

Testing node replacement...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FOOD"

Testing node removal...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
```

```
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FOOD"

Normalizing...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-GibberishGibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FOOD"

Creating and populating document fragment...
        DOCFRAG
            ELEMENT "FragElem"
            TEXT = "FragText"
Insert document fragment...
        ELEMENT "ROOT"
            TEXT = "Pre-GibberishGibberish"
            COMMENT = "Ask about the weather:"
            ELEMENT "FragElem"
            TEXT = "FragText"
            COMMENT = "Bit warm today, innit?"
            TEXT = "REPLACEMENT, 1/2 PRICE"
            ELEMENT "SUB"
                TEXT = "Lengthy SubText"
            ELEMENT "SUB"
            CDATA = "See DATA"
            ELEMENT "FOOD"

Create two attributes...
Setting attributes...
        ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Altering attribute1 value...
        ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
```

```
            TEXT = "Lengthy SubText"

Fetching attribute by name (Attr2)...
        ATTRIBUTE "Attr2" = "Value2"

Removing attribute by name (Attr1)...
        ELEMENT "SUB" [Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Adding new attribute...
        ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Removing attribute by pointer (Attr2)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Adding new attribute w/same name (test replacement)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Testing node (attr) set by name ...
Adding 'GLEEP' attribute and printing out hex addresses of node set
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep1"]
            TEXT = "Lengthy SubText"

Testing node set by name ...
Replacing 'GLEEP' attribute - note the changed hex address
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Replaced node was:
        ATTRIBUTE "GLEEP" = "gleep1"

Original SubROOT...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (not deep)...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (deep)...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Lengthy SubText"

Splitting text...
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
```

```
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
            TEXT = "Leng"
            TEXT = "thy SubText"

Testing string operations...
    CharData = "Leng"
Setting new data...
    CharData = "0123456789"
    CharLength = 10
    Substring(0,5) = "01234"
    Substring(8,2) = "89"
Appending data...
    CharData = "0123456789ABCDEF"
Inserting data...
    CharData = "0123456789*foo*ABCDEF"
Deleting data...
    CharData = "*foo*ABCDEF"
Replacing data...
    CharData = "*bamboozle*ABCDEF"
Cleaning up...
Parsing test document...
Document from root node:
    DOCUMENT
        DTD "doc"
        ELEMENT "doc" [xml:lang*="foo"]
            ELEMENT "p" [xml:space="preserve"]
                TEXT = "An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;)."
Testing getDocTypeNotations...
# of notations = 2
        NOTATION "notation1"
        NOTATION "notation2"
Testing getDocTypeEntities...
# of entities = 1
        ENTITY "example" = "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>"
Cleaning up...

Terminating parser...
Success.
```

XML Parser for C の例 16: C - XSLSample.c

XSLSample.c には、XSLSample の C ソース・コードが含まれます。

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
   NAME
     XSLSample.c - Sample function for XSL
   DESCRIPTION
     Sample usage of C XSL Processor
*/

#include <stdio.h>
#ifndef ORATYPES
# include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
# include <oraxml.h>
#endif

int main(int argc, char *argv[])
{
    xmlctx     *xctx, *xslctx, *resctx;
    xmlnode    *result;
    uword       ecode;
    /* Check for correct usage */
    if (argc < 3)
        {
        puts("Usage is XSLSample <xmlfile> <xslfile>\n");
        return 1;
        }

    /* Parse the XML document */
    if (!(xctx = xmlinit(&ecode, (const oratext *) 0,
                         (void (*)(void *, const oratext *, uword)) 0,
                         (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                         (const xmlmemcb *) 0, (void *) 0,
                         (const oratext *) 0)))
    {
        printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
        return 1;
    }

    printf("Parsing '%s' ...\n", argv[1]);
    if (ecode = xmlparse(xctx, (oratext *)argv[1], (oratext *) 0,
                         XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
```

```
{
    printf("Parse failed, error %u\n", (unsigned) ecode);
    return 1;
}

/* Parse the XSL document */
if (!(xslctx = xmlinit(&ecode, (const oratext *) 0,
                    (void (*)(void *, const oratext *, uword)) 0,
                    (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                    (const xmlmemcb *) 0, (void *) 0,
                    (const oratext *) 0)))
{
    printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
    return 1;
}

printf("Parsing '%s' ...\n", argv[2]);
if (ecode = xmlparse(xslctx, (oratext *)argv[2], (oratext *) 0,
                    XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
{
    printf("Parse failed, error %u\n", (unsigned) ecode);
    return 1;
}

/* Initialize the result context */
if (!(resctx = xmlinit(&ecode, (const oratext *) 0,
                    (void (*)(void *, const oratext *, uword)) 0,
                    (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                    (const xmlmemcb *) 0, (void *) 0,
                    (const oratext *) 0)))
{
    printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
    return 1;
}

/* XSL processing */
printf("XSL Processing\n");
if (ecode = xslprocess(xctx, xslctx, resctx, &result))
{
    printf("Parse failed, error %u\n", (unsigned) ecode);
    return 1;
}

/* Print the result tree */
printres(resctx, result);

/* Call the terminate functions */
```

```
    (void)xmlterm(xctx);
    (void)xmlterm(xslctx);
    (void)xmlterm(resctx);

    return 0;
}
```

XML Parser for C の例 17: C - XSLSample.std

XSLSample.std は、XSLSample.c からの予想される出力を表示します。

```
Parsing 'class.xml' ...
Parsing 'iden.xsl' ...
XSL Processing
<root>
   <course>
      <Name>Calculus</Name>
      <Dept>Math</Dept>
      <Instructor>
         <Name>Jim Green</Name>
      </Instructor>
      <Student>
         <Name>Jack</Name>
         <Name>Mary</Name>
         <Name>Paul</Name>
      </Student>
   </course>
</root>
```

25

XML Schema Processor for C の使用

この章の内容は次のとおりです。

XML Schema Processor for C の概要

XML Schema Processor for C は、XML Parser for C とともに動作するコンポーネントです。XML Schema Processor for C を使用すると、Oracle の XML アプリケーションで単純および複雑なデータ型をサポートできます。

XML Schema Processor for C は、XML Schema 草案が W3C 勧告になった場合、勧告に完全に準拠することを目標として、XML Schema をサポートしています。このため、Oracle 環境では、XML 文書を処理するカスタム・アプリケーションを簡単に作成できます。また、Oracle が移植されたすべてのオペレーティング・システムには、標準に準拠した XML Schema Processor が Oracle プラットフォームの一部として搭載されています。

> **参照：** XML Schema の詳細および XML Schema を使用する理由については、第 21 章「XML Schema Processor for Java の使用」を参照してください。

XML Schema Processor for C の機能

XML Schema Processor for C には、次の機能があります。

- 単純型および複合型をサポートします。
- XML Parser for C に統合されています。
- W3C の XML Schema 草案をサポートします。

> **参照：** 次のマニュアルおよび章を参照してください。
>
> - 『Oracle9*i* XML リファレンス』
> - 付録 E「XDK for C: 仕様および早見表」

要件

XML Schema Processor for C は、次のオペレーティング・システム上で動作します。

- Linux
- Solaris
- HP/UX
- Windows NT 4.0/Service Pack 3 以上

オンライン・ドキュメント

Oracle XML Schema Processor for C のドキュメントは、インストール場所の doc/ ディレクトリにあります。

標準への準拠

Oracle XML Parser for C は、次の標準に準拠しています。

- W3C の XML 1.0 勧告
- W3C の DOM レベル 1.0 勧告
- W3C の XML Namespace 勧告案
- SAX 1.0
- W3C の XSLT 勧告
- W3C の XPath 勧告

サポートされているキャラクタ・セットの使用

XML Parser for C は、現在、次のエンコーディングをサポートしています。

- BIG5
- EBCDIC-CP-BE
- EBCDIC-CP-CA
- EBCDIC-CP-CH
- EBCDIC-CP-DK
- EBCDIC-CP-ES
- EBCDIC-CP-FI
- EBCDIC-CP-FR
- EBCDIC-CP-GB
- EBCDIC-CP-HE
- EBCDIC-CP-IS
- EBCDIC-CP-IT

- EBCDIC-CP-NL
- EBCDIC-CP-NO
- EBCDIC-CP-ROECE
- EBCDIC-CP-SE
- EBCDIC-CP-US
- EBCDIC-CP-WT
- EBCDIC-CP-YU
- EUC-JP
- GB2312
- ISO-10646-UCS-2
- ISO-8859-1 ～ ISO-8859-9
- KOI8-RUTF-8
- SHIFT_JIS
- US-ASCII
- UTF-16

> **参照：** 『Oracle9*i* グローバリゼーション・サポート・ガイド』の付録 A の「キャラクタ・セット」で指定されているキャラクタ・セットも使用できます。

これらのエンコーディングを使用するには、次の設定が必要です。

- 環境変数 ORACLE_HOME が、Oracle のインストール場所を指すように設定する必要があります。
- 環境変数 ORA_NLS、ORA_NLS32 および ORA_NLS33 が、NLS データ・ファイルの場所を指すように設定する必要があります。これらのデータ・ファイルは、通常、次の場所にあります。
  - UNIX システム : `$ORACLE_HOME/ocommon/nls/admin/data`
  - Windows NT: `$ORACLE_HOME¥nlsrtl¥admin¥nlsdata`

デフォルトでは、UTF-8 がエンコーディングになります。シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 2 倍向上します。

XML Schema Processor for C: ソフトウェア

表 25-1 に、今回のリリースで提供されるファイルおよびディレクトリを示します。

表 25-1 XML Schema Processor for C: 提供されるファイル

ディレクトリおよびファイル	説明
license.html	ライセンス契約
bin/	Schema Processor が実行可能なスキーマ
doc/	API ドキュメント
include/	ヘッダー・ファイル
lib/	XML/XSL/Schema およびサポート・ライブラリ
mesg/	エラー・メッセージ・ファイル
sample/	スキーマ・プロセッサの使用例

表 25-2 に、含まれるライブラリを示します。

表 25-2 XML Schema Processor for C: 提供されるライブラリ

含まれるライブラリ	説明
libxml8.a	XML パーサー /XSL プロセッサ
libxsd8.a	XML Schema Processor
libcore8.a	CORE の機能
libnls8.a	National Language Support

XML Schema Processor for C の起動

XML Schema Processor for C は、インストール場所の bin/schema をコールすることによって、実行可能ファイルとしてコールされます。これは次の 2 つの引数を取ります。

- XML インスタンス・ドキュメント
- デフォルト・スキーマ（オプション）

XML Schema Processor for C は、提供される API を使用するためのコードを記述して起動することもできます。このコードは、include/ サブディレクトリにあるヘッダーを使用してコンパイルし、lib/ サブディレクトリにあるライブラリにリンクする必要があります。プログラムを作成する方法の詳細は、sample/ サブディレクトリにある Makefile を参照してください。

エラー・メッセージ・ファイルは、mesg/ サブディレクトリにあります。現在、メッセージ・ファイルは英語表記のみです。他の言語のメッセージ・ファイルは、将来のリリースで提供される予定です。

環境変数 ORA_XML_MESG の設定による絶対パスの指定

環境変数 ORA_XML_MESG を設定し、mesg/ サブディレクトリの**絶対**パスを指定する必要があります。または、$ORACLE_HOME をインストールしている場合、mesg/ サブディレクトリの内容を $ORACLE_HOME/oracore/mesg ディレクトリにコピーすることもできます。

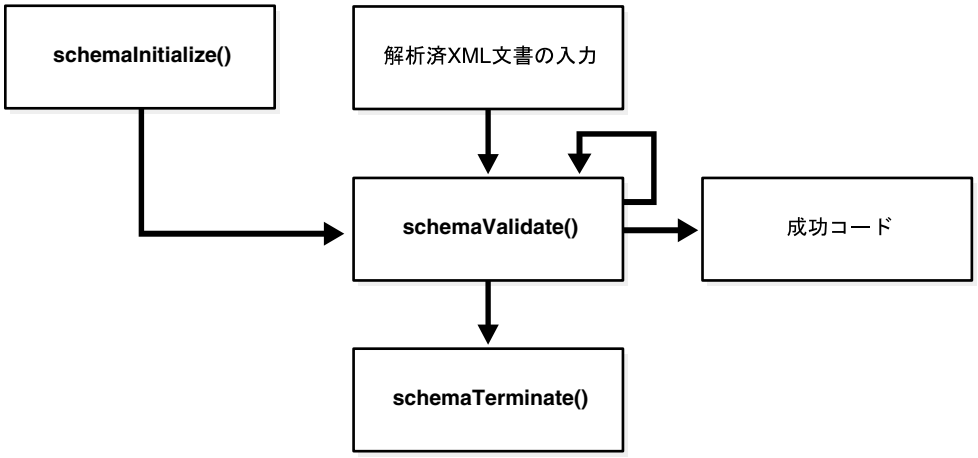
XML Schema Processor for C の使用方法

図 25-1 に、次に示す XML Schema Processor for C のコール順序を示します。

XML Schema Processor for C へのコールは、初期化、検証、検証、...、検証、終了の順序で実行されます。

1. 初期化コールが、セッションの開始時に一度コールされ、セッションで使用されるスキーマ・コンテキストを戻します。
2. 検証するインスタンス・ドキュメントが、まず XML パーサーによって解析されます。
3. インスタンスに対する XML コンテキストが、オプションのスキーマの URL とともにスキーマ検証関数に渡されます。
4. インスタンス・ドキュメントにスキーマが明示的に定義されていない場合は、デフォルトのスキーマが使用されます。
5. 同じスキーマ・コンテキストを使用して、複数のドキュメントを検証できます。
6. セッションが終了すると、スキーマ・メモリー解放関数がコールされます。これによって、ロードされたスキーマによって割り当てられていたすべてのメモリーが解放されます。

図 25-1 XML Schema Processor for C の使用方法



XML Schema Processor for C サンプル・プログラムの実行

sample/ ディレクトリには、API による Oracle XML Schema Processor の使用方法を示すサンプル XML Schema アプリケーションがあります。表 25-3 に、提供されるサンプル・ファイルを示します。

表 25-3 XML Schema Processor for C のサンプル・ファイル

サンプル・ファイル	説明
Makefile	サンプル・プログラムを作成および実行し、適切な出力を確認する Make ファイル
xsdtest.c	XML Schema for C の API をコールする一般的なプログラム
car.{xsd,xml,std}	xsdtest を実行した後のサンプル・スキーマ、インスタンス・ドキュメントおよび予想される出力。次の例を参照してください。 25-11 ページの「XML Schema for C の例 2: car.xsd」 25-12 ページの「XML Schema for C の例 3: car.xml」 25-13 ページの「XML Schema for C の例 4: car.std」

表 25-3 XML Schema Processor for C のサンプル・ファイル（続き）

サンプル・ファイル	説明
aq.{xsd,xml,std}	xsdtest を実行した後の 2 つ目のサンプル・スキーマ、インスタンス・ドキュメントおよび予想される出力。次の例を参照してください。 25-13 ページの「XML Schema for C の例 5: aq.xsd」 25-22 ページの「XML Schema for C の例 6: aq.xml」 25-23 ページの「XML Schema for C の例 7: aq.std」
pub.{xsd,xml,std}	xsdtest を実行した後の 3 つ目のサンプル・スキーマ、インスタンス・ドキュメントおよび予想出力。次の例を参照してください。 25-23 ページの「XML Schema for C の例 8: pub.xsd」 25-25 ページの「XML Schema for C の例 9: pub.xml」 25-26 ページの「XML Schema for C の例 10: pub.std」

サンプル・プログラムを作成するには、「**make**」を実行します。

プログラムを作成および実行し、実際の出力と予想される出力を比較するには、「**make sure**」を実行します。

Make.bat

```
:: ############################################################################
:: # Batch script to build Oracle XML parser C sample programs
:: ############################################################################
set opt_flg=-Ox -Oy-
if (%2) == (D) set opt_flg=-Z7 -Od
if (%2) == (D) set link_dbg=/debug /debugtype:both /pdb:none

if (%1) == () goto :XSDTEST
if (%1) == (all) goto :XSDTEST
if (%1) == (xsdtest) goto :XSDTEST
if (%1) == (clean) goto :CLEAN
if (%1) == (sure) goto :SURE
goto :EOF

:CLEAN
del *.obj
del *.out
del ..\bin\xsdtest.exe
goto :EOF

:XSDTEST
```

```
call :compile xsdtest
call :link xsdtest
if (%1) == (xsdtest) goto :EOF

:SURE
..\bin\xsdtest.exe car.xml > car.out
comp car.std car.out < NUL:
..\bin\xsdtest.exe pub.xml > pub.out
comp pub.std pub.out < NUL:
..\bin\xsdtest.exe aq.xml > aq.out
comp aq.std aq.out < NUL:
goto :EOF

:COMPILE
set filename=%1
cl -c -Fo%filename%.obj %opt_flg%  /DCRTAPI1=_cdecl /DCRTAPI2=_cdecl /nologo /Zl /Gy
/DWIN32 /D_WIN32 /DWIN_NT /DWIN32COMMON /D_DLL /D_MT /D_X86_=1 -I. -I..\include
%filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..\bin\%filename%.exe /libpath:%ORACLE_HOME%\lib
/libpath:..\lib  %filename%.obj oraxml8.lib oraxsd8.lib oracore8.lib oranls8.lib
user32.lib kernel32.lib msvcrt.lib ADVAPI32.lib oldnames.lib winmm.lib

:EOF
```

XML Schema for C の例 1: xsdtest.c

```
/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*
   NAME
     validate.c - Sample Schema validation

   DESCRIPTION
     Sample usage of C XML Schema processor
*/

#include <stdio.h>

#ifndef ORATYPES
# include <oratypes.h>
#endif
#ifndef ORAXML_ORACLE
# include <oraxml.h>
```

```
#endif

#ifndef ORAXSD_ORACLE
# include <oraxsd.h>
#endif

int main(int argc, char **argv)
{
    xmlctx     *ctx;
    xsdctx     *scctx;
    char       *doc, *schema;
    uword       ecode;

    puts("XML C Schema processor");

    if ((argc < 2) || (argc > 3))
    {
puts("usage: validate <xml document> [schema]");
return -1;
    }
    doc = argv[1];
    schema = (argc > 2) ? argv[2] : 0;

    puts("Initializing XML package...");

    if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
                        (void (*)(void *, const oratext *, uword)) 0,
                        (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                        (const xmlmemcb *) 0, (void *) 0,
                        (const oratext *) 0)))
    {
        printf("Failed to initialze XML parser, error %u\n", (unsigned) ecode);
        return 1;
    }

    printf("Parsing '%s' ...\n", doc);
    if (ecode = xmlparse(ctx, (oratext *) doc, (oratext *) 0,
 XML_FLAG_DISCARD_WHITESPACE))
    {
        printf("Parse failed, error %u\n", (unsigned) ecode);
        return 2;
    }

    puts("Initializing Schema package...");

    if (!(scctx = schemaInitialize(ctx, &ecode)))
    {
```

```
printf("Failed, code %u!\n", ecode);
return 3;
    }

    puts("Validating document...");
    if (ecode = schemaValidate(scctx, ctx, (oratext *) schema))
    {
        printf("Validation failed, error %u\n", (unsigned) ecode);
        return 4;
    }

    puts("Document is valid.");
    return 0;
}
```

XML Schema for C の例 2: car.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.CarDealers.com/">
    <element name="Car">
<complexType>
    <element name="Model">
<simpleType base="string">
    <enumeration value = "Ford"/>
    <enumeration value = "Saab"/>
    <enumeration value = "Audi"/>
</simpleType>
    </element>
    <element name="Make">
<simpleType base="string">
    <minLength value = "1"/>
    <maxLength value = "30"/>
</simpleType>
    </element>
    <element name="Year">
<complexType content="mixed">
    <attribute name="PreviouslyOwned" type="string"
       use="required"/>
    <attribute name="YearsOwned" type="integer"
       use="optional"/>
</complexType>
    </element>
    <element name="OwnerName" type="string"
     minOccurs="0" maxOccurs="unbounded"/>
    <element name="Condition">
```

```
<complexType base="string" derivedBy="extension">
    <attribute name="Automatic">
<simpleType base="string">
    <enumeration value = "Yes"/>
    <enumeration value = "No"/>
</simpleType>
    </attribute>
</complexType>
    </element>
    <element name="Mileage">
<simpleType base="integer">
    <minInclusive value="0"/>
    <maxInclusive value="2000000"/>
</simpleType>
    </element>
    <attribute name="RequestDate" type="date"/>
</complexType>
    </element>
</schema>
```

XML Schema for C の例 3: car.xml

```
<?xml version="1.0"?>
<car:Car xmlns:car="http://www.CarDealers.com/"
     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
     xsi:schemaLocation="http://www.CarDealers.com/ car.xsd"
     RequestDate="2000-12-6">

    <Model>Ford</Model>
    <Make>Explorer</Make>
    <Year PreviouslyOwned="You betcha">1999</Year>
    <OwnerName>Joe Smith</OwnerName>
    <OwnerName>Bob Jones</OwnerName>
    <Condition Automatic="No">Small dent on right bumper.</Condition>
    <Mileage>1999999</Mileage>

</car:Car>
```

XML Schema for C の例 4: car.std

```
XML C Schema processor
Initializing XML package...
Parsing 'car.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```

XML Schema for C の例 5: aq.xsd

```
<?xml version="1.0"?>

<!-- ****************** AQ xml schema  ****************** -->

<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.oracle.com/AQXmlDocument"
        xmlns:aq = "http://www.oracle.com/AQXmlDocument"
        xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
elementFormDefault="qualified">

   <element name="AQXmlDocument">
      <complexType content="mixed">
        <choice>
              <group ref="aq:client_operation" minOccurs="0"/>
              <group ref="aq:server_response"/>
        </choice>
      </complexType>
    </element>


   <!-- ****************** Client Operations Group  ****************** -->
   <group name="client_operation">
       <sequence>
          <element ref="aq:client_operation" minOccurs="0" maxOccurs="1"/>
      <choice>
              <element ref="aq:producer_options" maxOccurs="1"/>
              <element ref="aq:consumer_options" maxOccurs="1"/>
              <element ref="aq:register_options" maxOccurs="1"/>
            </choice>
          <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
       </sequence>
    </group>

   <!-- ****************** Server Response Group ****************** -->
   <group name="server_response">
```

```
        <sequence>
           <element ref="aq:server_response" minOccurs="0" maxOccurs="1"/>
           <element ref="aq:receive_result" maxOccurs="1"/>
           <choice  minOccurs="0" >
               <element ref="aq:send_result" maxOccurs="1"/>
               <element ref="aq:publish_result" maxOccurs="1"/>
               <element ref="aq:receive_result" maxOccurs="1"/>
               <element ref="aq:sequence_num_result" maxOccurs="1"/>
           </choice>
        </sequence>
    </group>

    <!-- ***************** Server Propagation Group ***************** -->
    <group name="server_prop_operation">
        <sequence>
           <element ref="aq:server_prop_operation" minOccurs="0" maxOccurs="1"/>
           <choice>
               <element ref="aq:push" maxOccurs="1"/>
               <element ref="aq:notification" maxOccurs="1"/>
               <element ref="aq:sequence_num_request" maxOccurs="1"/>
           </choice>
        </sequence>
    </group>

    <!-- ***************** Client Operation ***************** -->
    <element name="client_operation">
       <complexType content="mixed">
         <element ref="aq:txid" minOccurs="0"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
       </complexType>
    </element>

    <!-- ***************** Server Response ***************** -->
    <element name="server_response">
       <complexType content="mixed">
         <element ref="aq:txid" minOccurs="0"/>
         <element ref="aq:status_response" minOccurs="1"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
       </complexType>
    </element>

    <!-- ***************** Server Propagation Operation ***************** -->
    <element name="server_prop_operation">
       <complexType content="mixed">
         <element ref="aq:txid" minOccurs="0"/>
<attribute name="prop_opcode" use="required" type="aq:prop_opcode_type"/>
       </complexType>
```

```
   </element>

   <element name="txid" type="string"/>

   <element name="destination">
     <complexType base='string' derivedBy="extension">
     <attribute name="lookup_type" type="aq:dest_lookup_type"
        use="default" value="NORMAL"/>
     </complexType>
   </element>

   <!-- **** destination lookup type ******* -->
   <!-- lookup_type can be specified to either lookup LDAP or use  -->
   <!-- the destination directly in NAME::ADDRESS::PROTOCOL format -->
   <simpleType name="dest_lookup_type" base="string">
      <enumeration value="NORMAL"/>
      <enumeration value="LDAP"/>
    </simpleType>

   <!-- ***************** Producer Options ***************** -->
   <element name="producer_options">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:priority" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:expiration" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:recipient_list" minOccurs="0" maxOccurs="1"/>
<attribute name="visibility" type="aq:visibility_type"
           use="default" value="ON_COMMIT"/>
<attribute name="delivery_mode" type="aq:del_mode_type"
           use="default" value="PERSISTENT"/>
      </complexType>
    </element>

   <!-- ***************** Consumer Options ***************** -->
   <element name="consumer_options">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:wait_time" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:selector" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:batch_size" minOccurs="0" maxOccurs="1"/>

<attribute name="visibility" type="aq:visibility_type"
           use="default" value="ON_COMMIT"/>
<attribute name="dequeue_mode" type="aq:deq_mode_type"
           use="default" value="REMOVE"/>
<attribute name="navigation" type="aq:nav_mode_type"
```

```
            use="default" value="NEXT_MESSAGE"/>
        </complexType>
      </element>


    <!-- ***************** Register Options ***************** -->
    <element name="register_options">
       <complexType content="mixed">
         <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
         <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:notify_url" minOccurs="1" maxOccurs="1"/>
         <element ref="aq:qos" minOccurs="1" maxOccurs="1"/>
         <element ref="aq:batch_size" minOccurs="0" maxOccurs="1"/>
       </complexType>
     </element>

    <element name="recipient_list">
      <complexType content="mixed">
<element ref="aq:recipient" minOccurs="1" maxOccurs="*"/>
      </complexType>
    </element>

    <!-- ***************** Message Set ************************ -->
    <element name="message_set">
       <complexType content="mixed">
         <element ref="aq:message_count" minOccurs="1"/>
         <element ref="aq:message" minOccurs="0" maxOccurs="*"/>
       </complexType>
    </element>

    <!-- ***************** Message ************************ -->
    <element name="message">
       <complexType content="mixed">
         <element ref="aq:message_number" minOccurs="0"/>
         <element ref="aq:message_header" minOccurs="1" maxOccurs="1"/>
         <element ref="aq:message_payload" minOccurs="0" maxOccurs="1"/>
       </complexType>
    </element>

    <!-- ***************** Message header ***************** -->
    <element name="message_header">
       <complexType content="mixed">
         <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:correlation" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:delay" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:priority" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:delivery_count" minOccurs="0" maxOccurs="1"/>
```

```
      <element ref="aq:message_state" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:sender_id" minOccurs="1" maxOccurs="1"/>
      <element ref="aq:exception_queue" minOccurs="0" maxOccurs="1"/>
    </complexType>
  </element>

<!-- ***************** Oracle JMS properties ***************** -->
<element name="oracle_jms_properties">
    <complexType content="mixed">
      <element ref="aq:type" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:reply_to" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:userid" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:appid" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:groupid" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:group_sequence" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:timestamp" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:recv_timestamp" minOccurs="0" maxOccurs="1"/>
    </complexType>
  </element>

<!-- ***************** Message payload ***************** -->
<element name="message_payload">
   <complexType>
      <choice>
        <element ref="aq:jms_text_message" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:jms_map_message" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:jms_bytes_message" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:jms_object_message" minOccurs="0" maxOccurs="1"/>
   <any minOccurs="0" maxOccurs="*" processContents="skip"/>
      </choice>
   </complexType>
</element>

...
<!-- ***************** Status response ***************** -->
<element name="status_response">
    <complexType content="mixed">
      <element ref="aq:acknowledge" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:status_code" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:error_code" minOccurs="0" maxOccurs="1"/>
      <element ref="aq:error_message" minOccurs="0" maxOccurs="1"/>
    </complexType>
</element>

<!-- ***************** Send result ***************** -->
<element name="send_result">
    <complexType content="mixed">
```

```
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
      </complexType>
    </element>

    <!-- ***************** Publish result ***************** -->
    <element name="publish_result">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
      </complexType>
    </element>

    <!-- ***************** Receive result ***************** -->
    <element name="receive_result">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
      </complexType>
    </element>

....
    <!-- ***************** Push messages ************************** -->
    <element name="push">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:sequence_number" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:message_set" minOccurs="1" maxOccurs="1"/>
      </complexType>
    </element>

    <!-- ***************** Notification ************************** -->
    <element name="notification">
      <complexType content="mixed">
        <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:message" minOccurs="0" maxOccurs="1"/>
      </complexType>
    </element>

    <element name="priority" type="integer"/>
    <element name="expiration" type="integer"/>
    <element name="consumer_name" type="string"/>
    <element name="wait_time" type="integer"/>
    <element name="batch_size" type="integer"/>
    <element name="qos" type="string"/>
```

```
    <element name="notify_url" type="string"/>
    <element name="message_id" type="string"/>
    <element name="message_state" type="string"/>

    <element name="sequence_number" type="integer"/>
    <element name="message_number" type="integer"/>
    <element name="message_count" type="integer"/>

    <element name="correlation" type="string"/>
    <element name="delay" type="integer"/>
    <element name="delivery_count" type="integer"/>
    <element name="exception_queue" type="string"/>

    <element name="type" type="string"/>
    <element name="userid" type="string"/>
    <element name="appid" type="string"/>
    <element name="groupid" type="string"/>
    <element name="group_sequence" type="integer"/>
    <element name="timestamp" type="date"/>
    <element name="recv_timestamp" type="date"/>

    <element name="recipient">
      <complexType base='string' derivedBy="extension">
      <attribute name="lookup_type" type="aq:dest_lookup_type"
            use="default" value="NORMAL"/>
      </complexType>
    </element>

    <element name="sender_id">
      <complexType base='string' derivedBy="extension">
      <attribute name="lookup_type" type="aq:dest_lookup_type"
            use="default" value="NORMAL"/>
      </complexType>
    </element>

    <element name="reply_to">
      <complexType base='string' derivedBy="extension">
      <attribute name="lookup_type" type="aq:dest_lookup_type"
            use="default" value="NORMAL"/>
      </complexType>
    </element>

    <element name="selector">
      <complexType>
<choice>
          <element ref="aq:correlation_id" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
```

```
        <element ref="aq:complex_selector" minOccurs="0" maxOccurs="1"/>
      </choice>
   </complexType>
</element>

<element name="correlation_id" type="string"/>
<element name="complex_selector" type="string"/>

 <simpleType name="visibility_type" base="string">
   <enumeration value="ON_COMMIT"/>
   <enumeration value="IMMEDIATE"/>
 </simpleType>

<simpleType name="del_mode_type" base="string">
   <enumeration value="PERSISTENT"/>
   <enumeration value="NONPERSISTENT"/>
 </simpleType>

<simpleType name="deq_mode_type" base="string">
   <enumeration value="BROWSE"/>
   <enumeration value="LOCKED"/>
   <enumeration value="REMOVE"/>
   <enumeration value="REMOVE_NO_DATA"/>
 </simpleType>

<simpleType name="nav_mode_type" base="string">
   <enumeration value="FIRST_MESSAGE"/>
   <enumeration value="NEXT_MESSAGE"/>
   <enumeration value="NEXT_TRANSACTION"/>
 </simpleType>

<simpleType name="opcode_type" base="string">
   <enumeration value="SEND"/>
   <enumeration value="RECEIVE"/>
   <enumeration value="PUBLISH"/>
   <enumeration value="REGISTER"/>
   <enumeration value="COMMIT"/>
   <enumeration value="ROLLBACK"/>
   <enumeration value="SEQ_NUM_REQUEST"/>
</simpleType>

<simpleType name="prop_opcode_type" base="string">
   <enumeration value="SEND"/>
   <enumeration value="NOTIFICATION"/>
   <enumeration value="COMMIT"/>
   <enumeration value="ROLLBACK"/>
   <enumeration value="SEQ_NUM_REQUEST"/>
```

```
   </simpleType>

   <element name="acknowledge">
     <complexType content="empty">
     </complexType>
   </element>
   <element name="status_code" type="string"/>
   <element name="error_code" type="string"/>
   <element name="error_message" type="string"/>

    <simpleType name="prop_type" base="string">
      <enumeration value="STRING"/>
      <enumeration value="NUMBER"/>
    </simpleType>

   <element name="name" type="string"/>
   <element name="value" type="string"/>

   <!-- ***************** JMS text message ***************** -->
   <element name="jms_text_message">
      <complexType content="mixed">
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:text_data" minOccurs="1" maxOccurs="1"/>
      </complexType>
   </element>
   <element name="text_data" type="string"/>

....

   <!-- ***************** JMS object message ***************** -->
   <element name="jms_object_message">
      <complexType content="mixed">
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:ser_object_data" minOccurs="1" maxOccurs="1"/>
      </complexType>
   </element>

   <element name="ser_object_data" type="string"/>
</schema>
```

XML Schema for C の例 6: aq.xml

```
<AQXmlDocument xmlns="http://www.oracle.com/AQXmlDocument"
       xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
       xsi:schemaLocation="http://www.oracle.com/AQXmlDocument aq.xsd">

    <client_operation opcode="SEND">
        <txid> sdasdfdsf </txid>
    </client_operation>

    <producer_options delivery_mode="PERSISTENT">
        <destination lookup_type="NORMAL"> queue1 </destination>
        <priority>23</priority>
        <recipient_list>
            <recipient> abc </recipient>
            <recipient lookup_type="LDAP"> abc </recipient>
        </recipient_list>
    </producer_options>

    <message_set>
        <message_count>1</message_count>
        <message>
            <message_number>1</message_number>
            <message_header>
                <correlation>XML_40_NEW_TEST</correlation>
                <delay>10</delay>
                <sender_id>scott::home::0</sender_id>
            </message_header>
            <message_payload>
                <jms_map_message>
                    <oracle_jms_properties>
                        <reply_to>oracle::redwoodshores::100</reply_to>
                        <userid>scott</userid>
                        <appid>AQProduct</appid>
                        <groupid>AQ</groupid>
                    </oracle_jms_properties>
                    <user_properties>
                        <property property_type="STRING">
                            <name>country</name>
                            <value>USA</value>
                        </property>
                        <property property_type="STRING">
                            <name>State</name>
                            <value>california</value>
                        </property>
                    </user_properties>
                    <map_data>
```

```
                        <item item_type="STRING">
                            <name>Car</name>
                            <value>Toyota</value>
                        </item>
                        <item item_type="STRING">
                            <name>Color</name>
                            <value>Blue</value>
                        </item>
                        <item item_type="STRING">
                            <name>Shape</name>
                            <value>Circle</value>
                        </item>
                        <item item_type="NUMBER">
                            <name>Price</name>
                            <value>20000</value>
                        </item>
                    </map_data>
                </jms_map_message>
            </message_payload>
        </message>
    </message_set>
</AQXmlDocument>
```

XML Schema for C の例 7: aq.std

```
XML C Schema processor
Initializing XML package...
Parsing 'aq.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```

XML Schema for C の例 8: pub.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/2000/08/XMLSchema"
        targetNamespace = "http://www.somewhere.org/BookCatalogue"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue"
        elementFormDefault="qualified">

    <complexType name="Pub">
      <sequence>
        <element name="Title" type="cat:titleType" maxOccurs="*"/>
        <element name="Author" type="string" maxOccurs="*"/>
        <element name="Date" type="date"/>
      </sequence>
```

```
  <attribute name="language" type="string" use="default" value="English"/>
  <anyAttribute namespace="##local"/>
</complexType>

<element name="Publication" type="cat:Pub"  abstract="true"/>
<element name="Book" substitutionGroup="cat:Publication">
    <complexType>
       <complexContent>
           <extension base="cat:Pub" >
              <sequence>
                 <element name="ISBN" type="string" default="123456789"/>
                 <element name="Publisher" type="string"/>
              </sequence>
           </extension>
       </complexContent>
  </complexType>
</element>
<complexType name="titleType">
   <simpleContent>
      <extension base="string" >
         <attribute name="old" type="string" use="default" value="false"/>
      </extension>
   </simpleContent>
</complexType>
<element name="Magazine" substitutionGroup="cat:Publication">
    <complexType>
       <complexContent>
          <extension base="cat:Pub">
              <sequence>
                 <element name="Volume" type="cat:VolumeType"/>
                 <element name="htmlTable">
                    <complexType>
                       <any namespace="##other"
                            processContents="skip"
                            minOccurs="0" maxOccurs="2"/>
                    </complexType>
                  </element>
              </sequence>
          </extension>
       </complexContent>
    </complexType>
</element>
<simpleType name="VolumeType">
   <restriction base="integer" >
      <minInclusive value = "1"/>
      <maxInclusive value = "12"/>
   </restriction>
```

```
    </simpleType>
    <element name="Catalogue">
        <complexType>
           <sequence>
             <element ref="cat:Publication" minOccurs="0" maxOccurs="*"/>
           </sequence>
        </complexType>
    </element>
</schema>
```

XML Schema for C の例 9: pub.xml

```
<?xml version="1.0"?>

<Catalogue xmlns = "http://www.somewhere.org/BookCatalogue"
   xmlns:cat = "http://www.somewhere.org/BookCatalogue"
   xmlns:html = "http://www.somewhere.org/HTMLCatalogue"
   xmlns:xsi = "http://www.w3.org/1999/XMLSchema-instance"
   xsi:schemaLocation =
                      "http://www.somewhere.org/BookCatalogue pub.xsd">

        <cat:Magazine>
                <Title>Natural Health</Title>
                <Author>October</Author>
                <Date>1999-12</Date>
                <Volume>12</Volume>
                <htmlTable>
                   <table  xmlns = "http://www.somewhere.org/HTMLCatalogue">
                      <tr>....</tr>
                   </table>
                   <html:table>
                      <html:tr>....</html:tr>
                   </html:table>
                </htmlTable>
        </cat:Magazine>

        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN></ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Book>

        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
```

```
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp;amp; Row</Publisher>
        </Book>

</Catalogue>
```

XML Schema for C の例 10: pub.std

```
XML C Schema processor
Initializing XML package...
Parsing 'pub.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```

第 IX 部

XDK for C++

第 IX 部では、XDK for C++ の入手方法および使用方法を説明します。第 IX 部に含まれる章は、次のとおりです。

- 第 26 章「XML Parser for C++ の使用」
- 第 27 章「XML Schema Processor for C++ の使用」
- 第 28 章「XML C++ Class Generator の使用」

26

XML Parser for C++ の使用

この章の内容は次のとおりです。

XML Parser for C++ の入手方法

XML Parser for C++ は Oracle に付属しています。OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

XML Parser for C++ は、`$ORACLE_HOME/xdk/cpp/parser` にあります。

XML Parser for C++ の機能

アーカイブのルート・ディレクトリにある `readme.html` ファイルには、不具合の修正や追加の API などのリリース固有の情報が含まれています。

XML Parser for C++ は、XML 文書が整形式であるか、または DTD に対して妥当であるかどうか（オプション）を確認します。XML Parser for C++ は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。

質問、コメントまたは不具合のレポートは、OTN の Web サイト http://otn.oracle.com の「XML Discussion Forum」にポストできます

仕様

XML Parser for C++ の仕様およびメソッドのリストは、付録 F「XDK for C++: 仕様および早見表」を参照してください。

> **参照：** 次のディレクトリ、マニュアルおよび Web サイトを参照してください。
>
> - インストール場所の doc ディレクトリ
> - 『Oracle9*i* XML リファレンス』

メモリー割当て

独自のメモリー割当てを使用する場合は、メモリー・コールバック関数 `memcb` を使用できます。この方法を使用するには、すべての関数を指定する必要があります。

SAX コールバックに渡されるパラメータに割り当てられたメモリー、または DOM 解析ツリーとともに格納されたノードおよびデータに割り当てられたメモリーは、次のいずれかの操作が完了すると解放されます。

- `xmlparse()` または `xmlparsebuf()` のコールによる、他のファイルまたはバッファの解析
- `xmlclean()` のコール
- `xmlterm()` のコール

スレッド・セーフティ

コールの初期化 / 解析 / 終了シーケンスの途中でスレッドが分岐している場合、不適切な動作および結果が発生する場合があります。

データ型索引

表 26-1 に、XML Parser for C++ で使用するデータ型を示します。

表 26-1 XML Parser for C++ で使用するデータ型

データ型	説明
oratext	文字列ポインタ
xmlctx	マスター XML のコンテキスト
xmlmemcb	メモリー・コールバック構造（オプション）
xmlsaxcb	SAX コールバック構造（SAX のみ）
ub4	32 ビット以上の符号なし整数
uword	システム固有の符号なし整数

エラー・メッセージ・ファイル

エラー・メッセージ・ファイルは、`mesg/` サブディレクトリにあります。このメッセージ・ファイルは、`$ORACLE_HOME/oracore/mesg` ディレクトリにもあります。環境変数 `ORA_XML_MESG` を設定して、`mesg/` サブディレクトリへの絶対パスを指定することもできます。ただし、これは必須ではありません。

検証モード

使用可能な検証モードの詳細は、20-5 ページの「Oracle XML Parser による 4 つの検証モードのサポート」を参照してください。

XML Parser for C++ の使用方法

図 26-1 に、XML Parser for C++ の機能を示します。

1. 解析プロセスは、`xmlinit()` メソッドによって開始します。
2. XML 入力は、ファイルまたは文字列バッファになります。次のメソッドを入力します。
   - `XMLParser.xmlparse()`（入力が XML ファイルの場合）
   - `XMLParser.xmlparseBuffer()`（入力が文字列バッファの場合）

3. DOM API または SAX API を起動します。

   DOM: DOM インタフェースを使用する場合は、次の手順が含まれます。

   - `XMLParser.xmlparse()` メソッドまたは `XMLParser.xmlparserBuffer()` メソッドが `XMLParser.getDocumentElement()` をコールします。他の DOM メソッドが適用されていない場合は、`XMLParser.xmlterm()` をコールできます。
   - 必要に応じて、他の DOM メソッドをオプションでコールします。これらは、通常、ノードのクラス・メソッドまたは出力メソッドです。これらのメソッドによって、DOM 文書が出力されます。
   - 完了後、プロセスが `XMLParser.xmlterm()` をコールします。
   - オプションで、最初に `XMLParser.xmlclean()` をコールして、解析プロセス中に作成されたすべてのデータ構造を削除できます。その後 `XMLParser.xmlterm()` をコールします。

   SAX: SAX インタフェースを使用する場合は、次の手順が含まれます。

   - コールバック・メソッドを介して、`XMLParser.xmlparse()` または `XMLParser.xmlparseBuffer()` からの解析の結果を処理します。
   - コールバック・メソッドを登録します。

4. オプションで `XMLParser.xmlclean()` を使用し、解析中に使用したメモリーおよび構造を削除して手順 5 に進みます。または、手順 2 に戻ります。
5. `XMLParser.xmlterm()` を使用して解析プロセスを終了します。

XMLParser のコール順序

XMLParser に対するコール順序は、次のいずれかになります。

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` - `XMLParser.xmlparse()` または

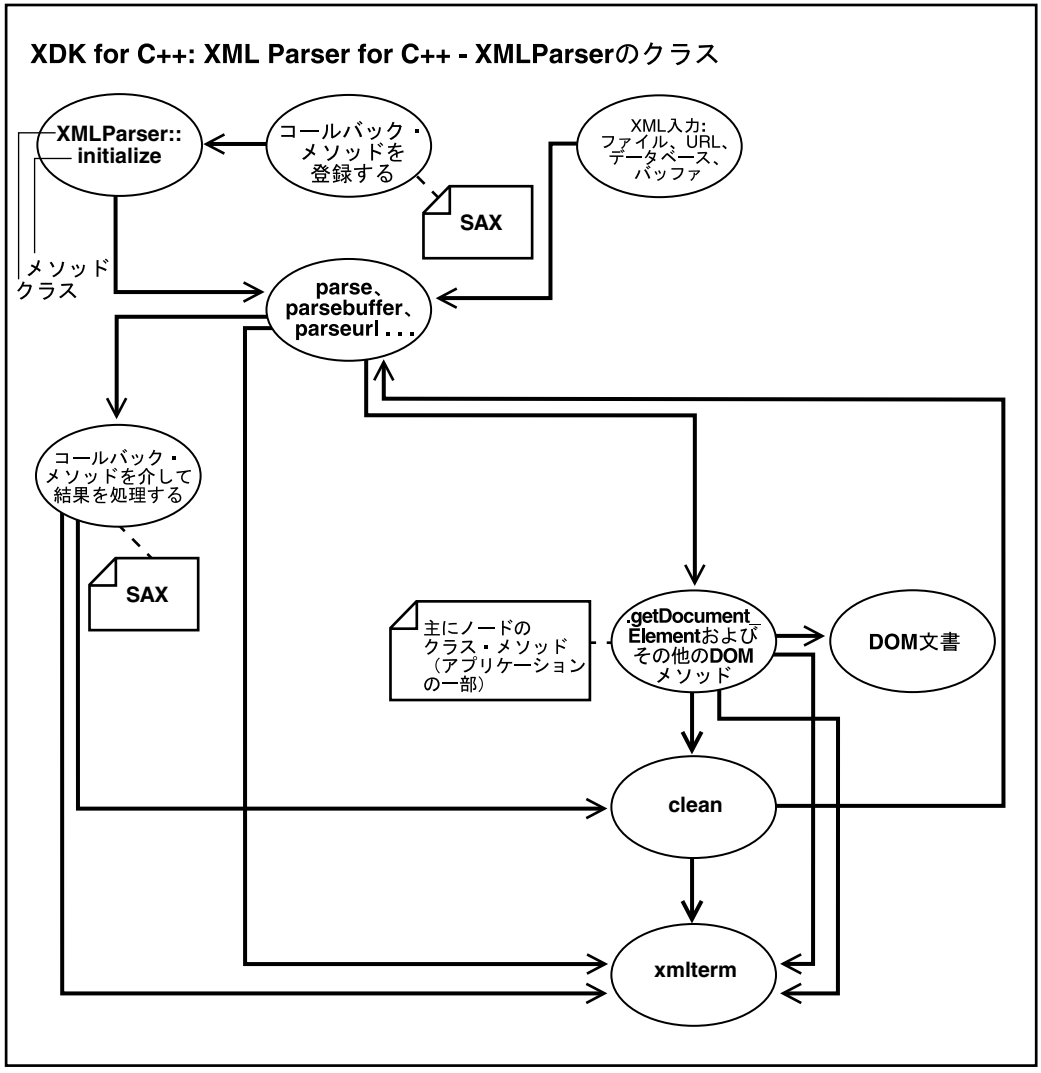
  `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` -... - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparsebuf()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparsebuf()` -... - `XMLParser.xmlterm()`

図 26-1 XML Parser for C++（DOM および SAX インタフェース）の使用方法



XML Parser for C++ の XSLT（DOM インタフェース）の使用方法

図 26-2 に、XML Parser for C++ の DOM インタフェース用の XSLT 機能を示します。

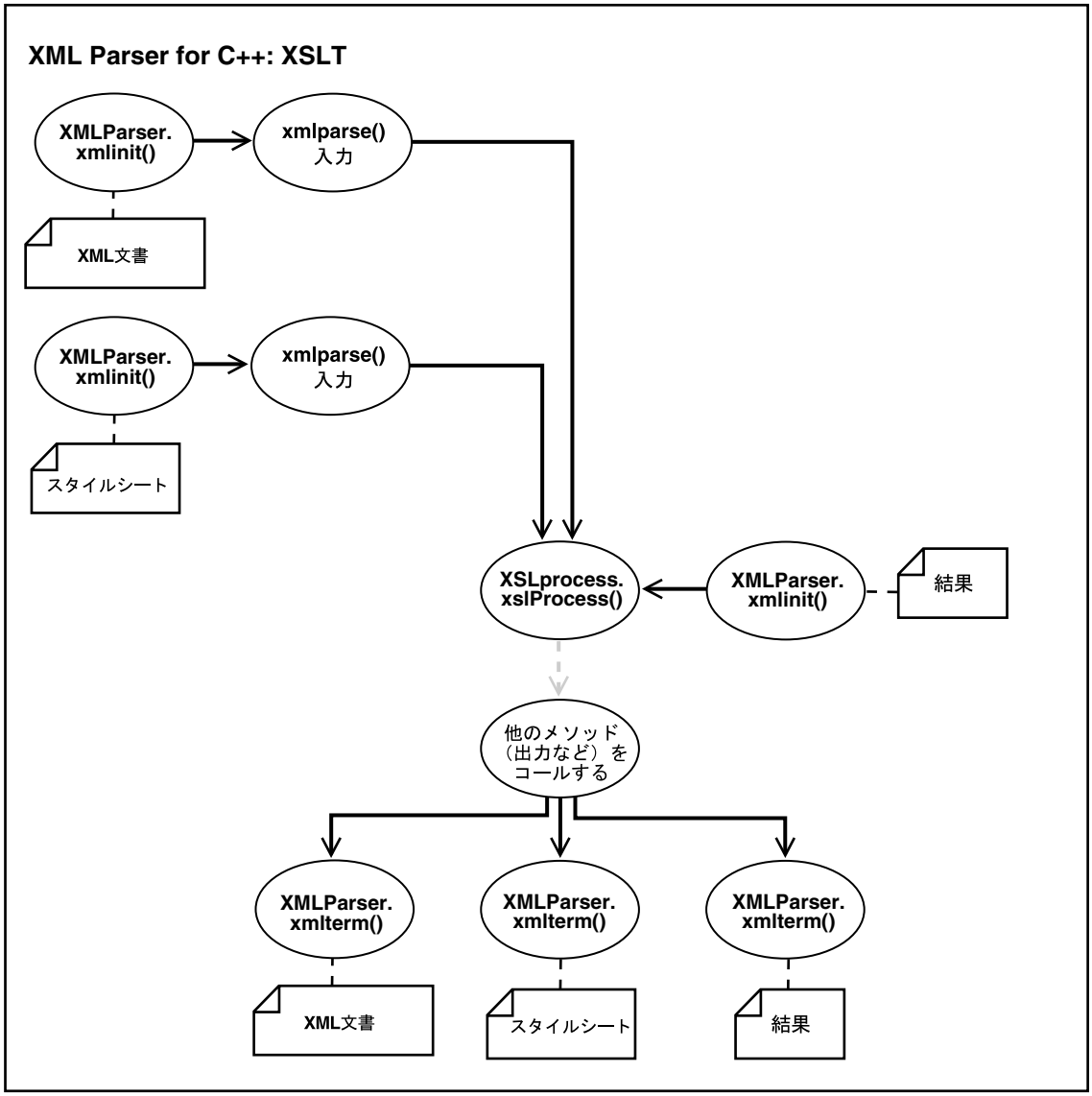
1. `XMLParser.xmlparse()` には、次の 2 つの入力があります。
   - XML 文書に適用するスタイルシート
   - XML 文書

   `XMLParser.xmlparse()` の出力である解析済スタイルシートおよび解析済 XML 文書は、`XSLProcess.xslprocess()` メソッドへ送られ、処理されます。
2. `XMLParser.xmlinit()` が XSLT プロセスを初期化します。また、`xslprocess()` の結果も初期化します。
3. オプションで、`XSLProcess.xslProcess()` が、出力メソッドなどの他のメソッドをコールします。使用可能なメソッドのリストは、OTN の Web サイトまたは『Oracle9*i* XML リファレンス』を参照してください。
4. 結果のドキュメント（XML、HTML、VML など）は、通常、アプリケーションに送られ、さらに処理されます。
5. アプリケーションが、`XMLParser.xmlterm()` を宣言して XSLT プロセスを終了し、XML 文書、スタイルシートおよび最終結果を生成します。

XML Parser for C の XSLT 機能については、次の例を参照してください。

- 26-51 ページの「XML Parser for C++ の例 16: C++ - XSLSample.cpp」
- 26-53 ページの「XML Parser for C++ の例 17: C++ - XSLSample.std」

図 26-2 Parser for C++: XSLT 機能（DOM インタフェース）の使用方法



XML Parser for C++ のデフォルト動作

XML Parser for C++ のデフォルト動作は、次のとおりです。

- キャラクタ・セットのエンコーディングは UTF-8 です。ドキュメントがすべて ASCII 形式である場合は、パフォーマンスを向上するために、エンコーディングを US-ASCII に設定することをお薦めします。
- メッセージは、msghdlr が指定されていないかぎり、stderr に出力されます。
- saxcb が SAX コールバック API を使用するように設定すると、DOM API がアクセスできる解析ツリーは構築されません。不要な SAX コールバック関数は、すべて NULL に設定できることに注意してください。
- パーサーは、デフォルト動作では、入力が整形式であることは確認しますが、妥当であるかどうかは確認しません。フラグ XML_FLAG_VALIDATE を設定することで、入力の妥当性を検証できます。空白処理のデフォルト動作は、XML 1.0 仕様に完全に準拠しています。すべての空白は、無視できる空白が明示された状態でアプリケーションに通知されます。ただし、アプリケーションによっては、XML_FLAG_DISCARD_WHITESPACE を設定し、要素の終了タグと次の要素の開始タグの間のすべての空白を削除する方が適切な場合もあります。

注意： シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 25% 向上します。

DOM API および SAX API

XML Parser for C++ は、XML 文書が整形式であるかどうか、および DTD に対して妥当であるかどうか（オプション）を確認します。このパーサーは、次のいずれかのインタフェースでアクセス可能なオブジェクト・ツリーを構築します。

- DOM インタフェース
- SAX インタフェースを介した順次操作

これらの XML API の説明を次に示します。

- DOM: ツリーベース API。ツリーベース API は、XML 文書を内部ツリー構造にコンパイルします。これによって、アプリケーションは、XML および HTML ドキュメント用のツリーベースの標準 API である DOM を使用してツリー内をナビゲートできます。

- SAX: イベントベース API。イベントベース API は、コールバックを使用して、要素の開始や終了などの解析イベントをアプリケーションに直接通知します。通常は、内部ツリーを構築しません。アプリケーションは、ハンドラを実装して様々なイベントを処理します。これは、GUI によるイベント処理に類似しています。

ツリーベース API は広範囲なアプリケーションで有効ですが、特に文書が大きい場合、より多くのシステム・リソースが消費される場合があります（詳細に制御された環境では、簡単な方法でツリーを構築し、この問題のいくつかを回避できる場合もあります）。さらに、いくつかのアプリケーションではそれぞれ独自のデータ・ツリーを構築する必要があります。この場合、新規のツリーにマップするためにのみ解析ノードのツリーを構築することは、非効率的です。

どちらの場合も、イベントベース API は、XML 文書に対してより単純で低レベルのアクセスを提供します。したがって、使用可能なシステム・メモリーよりサイズの大きい文書を解析し、コールバック・イベント・ハンドラを使用して独自のデータ構造を構築できます。

SAX API の使用

SAX を使用するために、xmlsaxcb 構造が関数ポインタによって初期化され、xmlinit() コールに渡されます。ユーザー定義のコンテキスト構造に対するポインタを含むこともできます。コンテキストのポインタは、各 SAX 関数に渡されます。

SAX コールバック構造

SAX コールバック構造を次に示します。

```
typedef struct
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
            const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
            const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
            const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
            const oratext *publicId,
            const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
            const oratext *local, const oratext *nsp,
            const struct xmlnodes *attrs);
} xmlsaxcb;
```

DOM API の使用

26-16 ページの「XML Parser for C++ の例 6: C++ - DOMSample.cpp」を参照してください。

XML Parser for C++ の起動

XML Parser for C++ は、次の 2 つの方法で起動できます。

- コマンドラインで実行可能ファイルを起動する
- C++ コードを記述し、提供される API を使用する

コマンドラインの使用方法

XML Parser for C++ は、`bin/xml` をコールすることによって、実行可能ファイルとしてコールできます。

表 26-2 に、コマンドライン・オプションを示します。

表 26-2 XML Parser for C++: コマンドライン・オプション

オプション	説明
-c	規格一致性の確認のみ。妥当性は検証しません。
-e encoding	入力ファイルのエンコーディングを指定します。
-h	ヘルプ - 使用方法のヘルプを表示します。
-n	数値 - DOM ツリーを検索し、要素の数をレポートします。
-p	解析後にドキュメントおよび DTD 構造を出力します。
-x	SAX インタフェースを実行し、ドキュメントを出力します。
-v	バージョン - パーサーのバージョンを表示し、終了します。
-w	空白 - すべての空白を保持します。

提供される API を使用するための C++ コードの記述

XML Parser for C++ は、コードを記述し、提供される API を使用することによって起動することもできます。コードは、include/ サブディレクトリにあるヘッダーを使用してコンパイルし、lib サブディレクトリにあるライブラリにリンクする必要があります。プログラムの作成方法の詳細は、sample/ サブディレクトリにある `Makefile` を参照してください。

ソフトウェアに含まれるサンプル・ファイルの使用

$ORACLE_HOME/xdk/cpp/parser/sample/ ディレクトリには、DOM インタフェースおよび SAX インタフェースによる XML Parser for C++ の使用方法を示す XML アプリケーションがあります。

表 26-3 に、sample/ ディレクトリにあるサンプル・ファイルを示します。

表 26-3 XML Parser for C++ のサンプル・ファイル

サンプル・ファイル名	説明
DOMNamespace.cpp	DOMNamespace プログラムのソース
DOMNamespace.std	DOMNamespace からの予想される出力
DOMSample.cpp	DOMSample プログラムのソース
DOMSample.std	DOMSample からの予想される出力
FullDOM.c	DOM インタフェースの使用例
FullDOM.std	FullDOM からの予想される出力
Make.bat	サンプル実行可能ファイルを作成するためバッチ・ファイル
Makefile	サンプル・プログラム用の Make ファイル
NSExample.xml	名前空間を使用したサンプル XML ファイル
SAXNamespace.cpp	SAXNamespace プログラムのソース
SAXNamespace.std	SAXNamespace からの予想される出力
SAXSample.cpp	SAXSample プログラムのソース
SAXSample.std	SAXSample からの予想される出力
XSLSample.cpp	SAXSample プログラムのソース
XSLSample.std	XSLSample からの予想される出力
class.xml	XSLSample で使用できる XML ファイル
iden.xsl	XSLSample で使用できるスタイルシート
cleo.xml	シェークスピアの戯曲の XML バージョン

XML Parser for C++ サンプル・プログラムの実行

サンプル・プログラムの作成

sample/ ディレクトリに移動し、README ファイルを参照してください。このファイルには、サンプル・プログラムの作成方法がプラットフォームごとに記載されています。

サンプル・プログラム

表 26-4 に、sample/ ディレクトリにあるサンプル・ファイルによって作成されたプログラムを示します。

表 26-4 XML Parser for C++: sample/ ファイルで作成されたサンプル・プログラム

作成されたプログラム	説明
SAXSample	SAX API を使用するサンプル・アプリケーション。戯曲（cleo.xml）シーンごとのすべての話者（各 SCENE 要素内のすべての一意の SPEAKER 要素）を出力します。
DOMSample [speaker]	DOM API を使用するサンプル・アプリケーション。指定された話者のすべてのせりふを出力します。話者を指定しない場合は、「Soothsayer」が使用されます。主要な役名はすべて大文字（「CLEOPATRA」など）で表され、端役の名前は頭文字だけが大文字（「Attendant」など）で表されることに注意してください。SAXSample の出力を参照してください。
SAXNamespace	SAX API に名前空間による拡張を使用するサンプル・アプリケーション。名前空間の完全な情報とともに NSExample.xml のすべての要素および属性を出力します。
DOMNamespace	DOM インタフェースを使用する以外は、SAXNamespace と同じです。
FullDOM	DOM インタフェース全体の使用例。すべてのコールを実行しますが、それ以外の操作は行われません。
XSLSample <xmlfile> <xsl ss>	XSL プロセッサの使用例。入力として、XML ファイルおよび XSL スタイルシートの 2 つのファイル名を取ります。 **注意** : このプログラムの stdout をファイルにリダイレクトする場合、環境によっては出力の一部が欠落する場合があります。

XML Parser for C++ の例 1: XML - class.xml

class.xml は、XSLSample.cpp を入力する XML ファイルです。

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for C++ の例 2: XML - cleo.xml

cleo.xml は、DOMSample.cpp および SAXSample.cpp を入力します。

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
    <!ELEMENT PLAY     (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
PROLOGUE?, ACT+, EPILOGUE?)>
    <!ELEMENT TITLE    (#PCDATA)>
    <!ELEMENT FM       (P+)>
    <!ELEMENT P        (#PCDATA)>
    <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
    <!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
    <!ELEMENT PERSONA  (#PCDATA)>
    <!ELEMENT GRPDESCR (#PCDATA)>
    <!ELEMENT SCNDESCR (#PCDATA)>
    <!ELEMENT PLAYSUBT (#PCDATA)>
    <!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
    <!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
    <!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
    <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
```

```
    <!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
    <!ELEMENT SPEAKER  (#PCDATA)>
    <!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
    <!ELEMENT STAGEDIR (#PCDATA)>
    <!ELEMENT SUBTITLE (#PCDATA)>
    <!ELEMENT SUBHEAD  (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>

<PERSONAE>
<TITLE>Dramatis Personae</TITLE>

<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>

<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>
...
...
<SPEECH>
```

```
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>
```

XML Parser for C++ の例 3: XSL - iden.xsl

iden.xsl はスタイルシートの例です。XSLSample.cpp の入力に使用できます。

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
          <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

XML Parser for C++ の例 4: XML - FullDOM.xml（DTD）

FullDOM.xml は DTD の例です。FullDOM.cpp を入力します。

```
<!DOCTYPE doc [
    <!ELEMENT p (#PCDATA)>
    <!ATTLIST p xml:space (preserve|default) 'preserve'>
    <!NOTATION notation1 SYSTEM "file.txt">
    <!NOTATION notation2 PUBLIC "some notation">
    <!ELEMENT doc (p*)>
    <!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>">
]>
<doc xml:lang="foo">&example;</doc>
```

XML Parser for C++ の例 5: XML - NSExample.xml

NSExample.xml は、名前空間を使用します。

```
<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:a1 CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:a1 = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
</child>
</doc>
```

XML Parser for C++ の例 6: C++ - DOMSample.cpp

DOMSample.cpp には、DOMSample の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

///////////////////////////////////////////////////////////////////////////
// NAME
//   DOMSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XML parser via DOM interface
//
// PUBLIC FUNCTION(S)
```

```
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
///////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <string.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT"cleo.xml"
#define DEFAULT_SPEAKER"Soothsayer"

void dump(Node *node);
void dumpspeech(Node *node);

char *speaker;
char *act, *scene;
uword n_speech;

int main(int argc, char **argv)
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XML C++ DOM sample\n";

    speaker = (argc > 1) ? argv[1] : DEFAULT_SPEAKER;

    cout << "Initializing XML package...\n";

    if (ecode = parser.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode;
        return 1;
    }

    cout << "Parsing '" << DOCUMENT << "'...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;
```

```
    cout << "Dumping " << speaker << " speeches...\n";
    cout.flush();
    cout << "-------------------------------------------------------------\n";
    act = scene = "";
    n_speech = 0;
    dump(parser.getDocumentElement());

    (void) parser.xmlterm();// terminate LPX package

    return 0;
}

void dump(Node *node)
{
    Node *title, *speak;
    char *name, *who;
    uword i, n_nodes;

    name = (char *) node->getName();
    if (!strcmp((char *) name, "ACT"))
    {
title = node->getFirstChild();
act = (char *) title->getFirstChild()->getValue();
    }
    else if (!strcmp((char *) name, "SCENE"))
    {
title = node->getFirstChild();
scene = (char *) title->getFirstChild()->getValue();
    }
    else if (!strcmp((char *) name, "SPEECH"))
    {
speak = node->getFirstChild();
who = (char *) speak->getFirstChild()->getValue();
if (!strcmp(who, speaker))
    dumpspeech(node);
    }

    if (node->hasChildNodes())
    {
n_nodes = node->numChildNodes();
for (i = 0; i < n_nodes; i++)
    dump(node->getChildNode(i));
    }
}

// <SPEECH>
```

```
// <SPEAKER>Soothsayer</SPEAKER>
// <LINE>Your will?</LINE>
// </SPEECH>

// <SPEECH>
// <SPEAKER>CLEOPATRA</SPEAKER>
// <LINE><STAGEDIR>Aside to DOMITIUS ENOBARBUS</STAGEDIR>  What means this?</LINE>
// </SPEECH>

void dumpspeech(Node *node)
{
    Node    *kid, *part, *partkid;
    uword    i, j, n_node, n_part;
    oratext *partname, *partval;

    if (n_speech++)
cout << "\n";
    cout << act << ", " << scene << "\n";
    n_node = node->numChildNodes();
    for (i = 0; i < n_node; i++)// skip speaker
    {
kid = node->getChildNode(i);// line #i
if (!strcmp((char *) kid->getName(), "LINE"))
{
    n_part = kid->numChildNodes();
    for (j = 0; j < n_part; j++)
    {
part = kid->getChildNode(j);
if (part->getType() == TEXT_NODE)
    cout << "    " << (char *) part->getValue() << "\n";
else
{
    partname = part->getName();
    partval = part->getFirstChild()->getValue();
    if (!strcmp((char *) partname, "STAGEDIR"))
cout << "    [" << (char *) partval << "]\n";
    else
cout << "    {" << (char *) partval << "}\n";
}
    }
}
    }
    cout.flush();
}

// end of DOMSample.c
```

XML Parser for C++ の例 7: C++ - DOMSample.std

DOMSample.std は、DOMSample.cpp からの予想される出力を表示します。

```
XML C++ DOM sample
Initializing XML package...
Parsing 'cleo.xml'...
Dumping Soothsayer speeches...
-------------------------------------------------------
ACT I, SCENE II.  The same. Another room.
    Your will?

ACT I, SCENE II.  The same. Another room.
    In nature's infinite book of secrecy
    A little I can read.

ACT I, SCENE II.  The same. Another room.
    I make not, but foresee.

ACT I, SCENE II.  The same. Another room.
    You shall be yet far fairer than you are.

ACT I, SCENE II.  The same. Another room.
    You shall be more beloving than beloved.

ACT I, SCENE II.  The same. Another room.
    You shall outlive the lady whom you serve.

ACT I, SCENE II.  The same. Another room.
    You have seen and proved a fairer former fortune
    Than that which is to approach.

ACT I, SCENE II.  The same. Another room.
    If every of your wishes had a womb.
    And fertile every wish, a million.

ACT I, SCENE II.  The same. Another room.
    Your fortunes are alike.

ACT I, SCENE II.  The same. Another room.
    I have said.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    Would I had never come from thence, nor you Thither!

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    I see it in
    My motion, have it not in my tongue: but yet
```

```
    Hie you to Egypt again.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    Caesar's.
    Therefore, O Antony, stay not by his side:
    Thy demon, that's thy spirit which keeps thee, is
    Noble, courageous high, unmatchable,
    Where Caesar's is not; but, near him, thy angel
    Becomes a fear, as being o'erpower'd: therefore
    Make space enough between you.

ACT II, SCENE III.  The same. OCTAVIUS CAESAR's house.
    To none but thee; no more, but when to thee.
    If thou dost play with him at any game,
    Thou art sure to lose; and, of that natural luck,
    He beats thee 'gainst the odds: thy lustre thickens,
    When he shines by: I say again, thy spirit
    Is all afraid to govern thee near him;
    But, he away, 'tis noble.
```

XML Parser for C++ の例 8: C++ - SAXSample.cpp

SAXSample.cpp には、SAXSample の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
////////////////////////////////////////////////////////////////////////////
// NAME
//   SAXSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XML parser via SAX interface
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
////////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <string.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif
```

```
#define DOCUMENT"cleo.xml"
#define MAX_STRING128
#define MAX_SPEAKER20

oratext  elem[MAX_STRING], last_elem[MAX_STRING];
uword    n_speaker;
oratext *speakers[MAX_SPEAKER];
size_t   speakerlen[MAX_SPEAKER];

/* SAX callback functions */

sword startDocument(void *ctx);
sword endDocument(void *ctx);
sword startElement(void *ctx, const oratext *name,
   const struct xmlnodes *attrs);
sword endElement(void *ctx, const oratext *name);
sword characters(void *ctx, const oratext *ch, size_t len);

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    startElement,
    endElement,
    characters
};

int main()
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XML C++ SAX sample\n";

    cout << "Initializing XML package...\n";

    if (ecode = parser.xmlinit((oratext *) 0,
// encoding
  (void (*)(void *, const oratext *, ub4)) 0,
  (void *) 0,
// msghdlr ctx
  (xmlsaxcb *) &saxcb))
// SAX callback
    {
    cout << "Failed to initialize XML parser, error " << ecode;
    return 1;
```

```
    }

    cout << "Parsing '" << DOCUMENT << "' and showing speakers by scene...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    (void) parser.xmlterm();
// terminate LPX package

    return 0;
}

sword startDocument(void *ctx)
{
    cout << "startDocument\n";
    return 0;
}

sword endDocument(void *ctx)
{
    cout << "endDocument\n";
    return 0;
}

sword startElement(void *ctx, const oratext *name,
   const struct xmlnodes *attrs)
{
    strcpy((char *) last_elem, (char *) elem);
    strcpy((char *) elem, (char *) name);
    return 0;
}

sword endElement(void *ctx, const oratext *name)
{
    uword i;

    if (!strcmp((char *) name, "SCENE"))
    {
for (i = 0; i < n_speaker; i++)
{
    cout << "    ";
    cout.write(speakers[i], speakerlen[i]);
    cout << "\n";
}
    }
    return 0;
```

```
}

sword characters(void *ctx, const oratext *ch, size_t len)
{
    uword i;

    if (!strcmp((char *) elem, "TITLE"))
    {
if (!strcmp((char *) last_elem, "ACT"))
{
    cout << "\n--- ";
    cout.write(ch, len);
    cout << " ---\n\n";
}
else if (!strcmp((char *) last_elem, "SCENE"))
{
    n_speaker = 0;
    cout << "  ";
    cout.write(ch, len);
    cout << "\n";
}
    }
    else if (!strcmp((char *) elem, "SPEAKER"))
    {
if (n_speaker < MAX_SPEAKER)
{
    for (i = 0; i < n_speaker; i++)
if ((len == speakerlen[i]) && !strncmp((char *) speakers[i],
(char *) ch, len))
    break;
    if (!n_speaker || (i == n_speaker))
    {
speakers[n_speaker] = (oratext *) ch;
speakerlen[n_speaker++] = len;
    }
}
    }
    return 0;
}

// end of SAXSample.cc
```

XML Parser for C++ の例 9: C++ - SAXSample.std

SAXSample.std は、SAXSample.cpp からの予想される出力を表示します。

```
XML C++ SAX sample
Initializing XML package...
Parsing 'cleo.xml' and showing speakers by scene...
startDocument

--- ACT I ---

  SCENE I.  Alexandria. A room in CLEOPATRA's palace.
    PHILO
    CLEOPATRA
    MARK ANTONY
    Attendant
    DEMETRIUS
  SCENE II.  The same. Another room.
    CHARMIAN
    ALEXAS
    Soothsayer
    DOMITIUS ENOBARBUS
    IRAS
    CLEOPATRA
    Messenger
    MARK ANTONY
    First Attendant
    Second Attendant
    Second Messenger
  SCENE III.  The same. Another room.
    CLEOPATRA
    CHARMIAN
    MARK ANTONY
  SCENE IV.  Rome. OCTAVIUS CAESAR's house.
    OCTAVIUS CAESAR
    LEPIDUS
    Messenger
  SCENE V.  Alexandria. CLEOPATRA's palace.
    CLEOPATRA
    CHARMIAN
    MARDIAN
    ALEXAS

--- ACT II ---

...
...
--- ACT V ---
```

```
  SCENE I.  Alexandria. OCTAVIUS CAESAR's camp.
    OCTAVIUS CAESAR
    DOLABELLA
    DERCETAS
    AGRIPPA
    MECAENAS
    Egyptian
    PROCULEIUS
    All
  SCENE II.  Alexandria. A room in the monument.
    CLEOPATRA
    PROCULEIUS
    GALLUS
    IRAS
    CHARMIAN
    DOLABELLA
    OCTAVIUS CAESAR
    SELEUCUS
    Guard
    Clown
    First Guard
    Second Guard
endDocument
```

XML Parser for C++ の例 10: C++ - DOMNamespace.cpp

DOMNamespace.cpp には、DOMNamespace の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
////////////////////////////////////////////////////////////////////////////
// NAME
//   DOMNamespace.cpp
//
// DESCRIPTION
//   This file demonstates a simple use of the parser and Namespace
//   extensions to the DOM APIs.
//
//   The XML file that is given to the application is parsed and the
//   elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
////////////////////////////////////////////////////////////////////////////
```

```
#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT         "NSExample.xml"

void dump(Node *node);
void dumpattrs(Node *node);

//
// main
//

int main()
{
    XMLParser   parser;
    ub4         flags;
    uword       ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "\nXML C++ DOM Namespace\n";
    cout << "Initializing XML package...\n";
    if (ecode = parser.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode;
        return 1;
    }

    cout << "Parsing '" << DOCUMENT << "'...\n";
    cout.flush();
    if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

    cout << "\nThe elements are:\n";
    dump(parser.getDocumentElement());
    (void) parser.xmlterm();// terminate LPX package
    return 0;
}
```

```
//
// dump
//

void dump(Node *node)
{
    uword i, n_nodes;
    NodeList *nodes;
    size_t   nn;

    String qName;
    String localName;
    String nsName;
    String prefix;

    if (node == NULL)
       return;
    if (nodes = node->getChildNodes())
    {
       for (nn = node->numChildNodes(), i=0; i < nn; i++)
       {
          // Use the methods getQualifiedName(), getLocalName(),
          // getPrefix(), and getNamespace() to get Namespace
          // information.

          qName = prefix = localName = nsName = (oratext *)" ";

  if (node->getQualifiedName() != (oratext *)NULL)
             qName = node->getQualifiedName();

  if (node->getPrefix() != (oratext *)NULL)
             prefix = node->getPrefix();

  if (node->getLocal() != (oratext *)NULL)
             localName = node->getLocal();

  if (node->getNamespace() != (oratext *)NULL)
             nsName = node->getNamespace();
          cout << "  ELEMENT Qualified Name: " << (char *)qName << "\n";
          cout << "  ELEMENT Prefix        : " << (char *)prefix << "\n";
          cout << "  ELEMENT Local Name    : " << (char *)localName << "\n";
          cout << "  ELEMENT Namespace     : " << (char *)nsName << "\n";
          dumpattrs(node);
          dump(node->getChildNode(i));
       }
    }
}
```

```
//
// dumpattrs
//

void dumpattrs(Node *node)
{
    NamedNodeMap  *attrs;
    Attr          *a;
    uword          i;
    size_t         na;

    oratext   *qname;
    oratext   *namespce;
    oratext   *local;
    oratext   *prefix;
    oratext   *value;
    if (attrs = node->getAttributes())
    {
       cout << "\n    ATTRIBUTES: \n";
       for (na = attrs->getLength(), i = 0; i < na; i++)
       {
          /* get attr qualified name, local name, namespace, and prefix */

          a = (Attr *)attrs->item(i);
          qname = namespce = local = prefix = value = (oratext*)" ";
          if (a->getQualifiedName() != (oratext*)NULL)
             qname = a->getQualifiedName();
          if (a->getNamespace() != (oratext*)NULL)
             namespce = a->getNamespace();
          if (a->getLocal() != (oratext*)NULL)
             local = a->getLocal();
          if (a->getPrefix() != (oratext*)NULL)
             prefix = a->getPrefix();
          if (a->getValue() != (oratext*)NULL)
             value = a->getValue();

          cout << "      " << (char*)qname << " = " << (char*)value << "\n";
          cout << "      Namespace : " << (char*)namespce << "\n";
          cout << "      Local Name: " << (char*)local << "\n";
          cout << "      Prefix    : " << (char*)prefix << "\n\n";
       }
    }
    cout << "\n";
}
```

XML Parser for C++ の例 11: C++ - DOMNamespace.std

DOMNamespace.std は、DOMNamespace.cpp からの予想される出力を表示します。

```
XML C++ DOM Namespace
Initializing XML package...
Parsing 'NSExample.xml'...

The elements are:
  ELEMENT Qualified Name: doc
  ELEMENT Prefix        :
  ELEMENT Local Name    : doc
  ELEMENT Namespace     : http://www.w3c.org

    ATTRIBUTES:
      nsprefix:a1 = v1
      Namespace : http://www.oracle.com
      Local Name: a1
      Prefix    : nsprefix

      xmlns = http://www.w3c.org
      Namespace :
      Local Name: xmlns
      Prefix    :

      xmlns:nsprefix = http://www.oracle.com
      Namespace :
      Local Name: nsprefix
      Prefix    : xmlns

  ELEMENT Qualified Name: child
  ELEMENT Prefix        :
  ELEMENT Local Name    : child
  ELEMENT Namespace     : http://www.w3c.org
```

XML Parser for C++ の例 12: C++ - SAXNamespace.cpp

SAXNamespace.cpp には、SAXNamespace の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
/////////////////////////////////////////////////////////////////////////////
// NAME
//   DOMNamespace.cpp
//
// DESCRIPTION
//   This file demonstates a simple use of the parser and Namespace
//   extensions to the SAX APIs.
//   The XML file that is given to the application is parsed and the
```

```
//   elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
/////////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define DOCUMENT         "NSExample.xml"

/*---------------------------------------------------------------------
                          FUNCTION PROTOTYPES
  ---------------------------------------------------------------------*/
int startDocument(void *ctx);
int endDocument(void *ctx);
int endElement(void *ctx, const oratext *name);
int nsStartElement(void *ctx, const oratext *qname,
                       const oratext *local,
                       const oratext *nsp,
                       const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb saxcb = {
    startDocument,
    endDocument,
    0,
    endElement,
    0,
    0,
    0,
    0,
    0,
    nsStartElement,
    0, 0, 0, 0, 0, 0, 0, 0
};

/* SAX callback context */
/*
```

```
typedef struct {
    xmlctx  *ctx;
    uword    depth;
} cbctx;
*/

/*---------------------------------------------------------------------------
                                MAIN
  ---------------------------------------------------------------------------*/
int main()
{
   XMLParser   parser;
   ub4         flags;
   uword       ecode;
   flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
   cout << "XML C++ SAX Namespace\n";
   cout << "Initializing XML package...\n";
   if (ecode = parser.xmlinit((oratext *) 0,           // encoding
                              (void (*)(void *, const oratext *, ub4)) 0,
                              (void *) 0,           // msghdlr ctx
                              (xmlsaxcb *) &saxcb)) // SAX callback
   {
       cout << "Failed to initialize XML parser, error " << ecode;
       return 1;
   }

   /* parse the document */

   cout << "Parsing '" << DOCUMENT << "'...\n";
   cout.flush();
   if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
      return 1;

   (void) parser.xmlterm();// terminate LPX package

   return 0;
}

/*---------------------------------------------------------------------------
                            SAX Interface
  ---------------------------------------------------------------------------*/
int startDocument(void *ctx)
{
    cout << "\nStartDocument\n\n";
    return 0;
}
```

```
int endDocument(void *ctx)
{
    cout << "\nEndDocument\n";
    return 0;
}

int endElement(void *ctx, const oratext *name)
{
    cout << "\nELEMENT Name  : " << (char*)name << "\n";
    return 0;
}

int nsStartElement(void *ctx, const oratext *qname, const oratext *local,
                   const oratext *nsp, const struct xmlnodes *attrs)
{
    xmlnode *attr;
    uword    i;
    oratext *aqname;
    oratext *alocal;
    oratext *anamespace;
    oratext *aprefix;
    oratext *avalue;

    /*
     * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
     * getXXXNamespace() to get Namespace information.
     */

    if (qname == (oratext*)NULL)
       qname = (oratext*)" ";
    if (local == (oratext*)NULL)
       local = (oratext*)" ";
    if (nsp == (oratext*)NULL)
       nsp = (oratext*)" ";

    cout << "ELEMENT Qualified Name: " << (char*)qname << "\n";
    cout << "ELEMENT Local Name    : " << (char*)local << "\n";
    cout << "ELEMENT Namespace     : " << (char*)nsp << "\n";

    if (attrs)
    {
       for (i = 0; i < numAttributes(attrs); i++)
       {
          attr = getAttributeIndex(attrs,i);
          aqname = alocal = anamespace = aprefix = avalue = (oratext*)" ";

          if (getAttrQualifiedName(attr))
```

```
            aqname = (oratext *) getAttrQualifiedName(attr);
         if (getAttrPrefix(attr))
            aprefix = (oratext *) getAttrPrefix(attr);
         if (getAttrLocal(attr))
            alocal = (oratext *) getAttrLocal(attr);
         if (getAttrNamespace(attr))
            anamespace = (oratext *) getAttrNamespace(attr);
         if (getAttrValue(attr))
            avalue = (oratext *) getAttrValue(attr);

        cout << " ATTRIBUTE Qualified Name   : " << (char*)aqname << "\n";
        cout << " ATTRIBUTE Prefix           : " << (char*)aprefix << "\n";
        cout << " ATTRIBUTE Local Name       : " << (char*)alocal << "\n";
        cout << " ATTRIBUTE Namespace        : " << (char*)anamespace << "\n";
        cout << " ATTRIBUTE Value            : " << (char*)avalue << "\n";
        cout << "\n";
      }
   }
   return 0;
}
```

XML Parser for C++ の例 13: C++ - SAXNamespace.std

SAXNamespace.std は、SAXNamespace.cpp からの予想される出力を表示します。

```
XML C++ SAX Namespace
Initializing XML package...
Parsing 'NSExample.xml'...

StartDocument

ELEMENT Qualified Name: doc
ELEMENT Local Name    : doc
ELEMENT Namespace     : http://www.w3c.org
 ATTRIBUTE Qualified Name   : nsprefix:a1
 ATTRIBUTE Prefix           : nsprefix
 ATTRIBUTE Local Name       : a1
 ATTRIBUTE Namespace        : http://www.oracle.com
 ATTRIBUTE Value            : v1

 ATTRIBUTE Qualified Name   : xmlns
 ATTRIBUTE Prefix           :
 ATTRIBUTE Local Name       : xmlns
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.w3c.org

 ATTRIBUTE Qualified Name   : xmlns:nsprefix
```

```
 ATTRIBUTE Prefix           : xmlns
 ATTRIBUTE Local Name       : nsprefix
 ATTRIBUTE Namespace        :
 ATTRIBUTE Value            : http://www.oracle.com

ELEMENT Qualified Name: child
ELEMENT Local Name    : child
ELEMENT Namespace     : http://www.w3c.org

ELEMENT Name  : child
ELEMENT Name  : doc
EndDocument
```

XML Parser for C++ の例 14: C++ - FullDOM.cpp

FullDOM.cpp には、FullDOM の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
////////////////////////////////////////////////////////////////////////////
// NAME
//   FullDOM.cpp
//
// DESCRIPTION
//   Sample code to test full C++ DOM interface
////////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#define TEST_DOC(oratext *) "FullDOM.xml"

void dump(Node *node, uword level);
void dumpnode(Node *node, uword level);

static char *ntypename[] = {
    "0",
    "ELEMENT",
    "ATTRIBUTE",
    "TEXT",
    "CDATA",
    "ENTREF",
    "ENTITY",
    "PI",
```

```
    "COMMENT",
    "DOCUMENT",
    "DTD",
    "DOCFRAG",
    "NOTATION"
};

#define FAIL { cout << "Failed!\n"; return 1; }

int main()
{
    XMLParser     parser;
    Document     *doc;
    Element      *root, *elem, *subelem;
    Attr         *attr, *attr1, *attr2, *gleep1, *gleep2;
    Text         *text, *subtext;
    Node         *node, *pi, *comment, *entref, *cdata, *clone,
 *deep_clone, *frag, *fragelem, *fragtext, *sub2,
 *fish, *food, *food2, *repl;
    NodeList     *subs, *nodes;
    NamedNodeMap *attrs, *notes, *entities;
    DocumentType *dtd;
    uword         i, ecode, level;

    cout << "XML C++ Full DOM test\n";
    cout << "Initializing XML parser...\n";

    if (ecode = parser.xmlinit())
    {
cout << "Failed to initialze XML parser, error " << ecode << "\n";
return 1;
    }

    cout << "\nCreating new document...\n";
    if (!(doc = parser.createDocument()))
FAIL

    cout << "Document from root node:\n";
    dump(parser.getDocument(), 0);

    cout << "\nCreating root element ('ROOT')...\n";
    if (!(elem = doc->createElement((oratext *) "ROOT")))
FAIL

    cout << "Setting as root element...\n";
    if (!doc->appendChild(elem))
FAIL
```

```
    cout << "Document from 'ROOT' element:\n";
    dump(root = parser.getDocumentElement(), 0);
    cout << "Adding 7 children to 'ROOT' element...\n";
    if (!(text = doc->createTextNode((oratext *) "Gibberish")) ||
        !elem->appendChild(text))
FAIL
    if (!(comment = doc->createComment((oratext*) "Bit warm today, innit?")) ||
        !elem->appendChild(comment))
FAIL

    if (!(pi = doc->createProcessingInstruction((oratext *) "target",
(oratext *) "PI-contents")) ||
       !elem->appendChild(pi))
FAIL

    if (!(cdata = doc->createCDATASection((oratext *) "See DATA")) ||
        !elem->appendChild(cdata))
FAIL

    if (!(entref = doc->createEntityReference((oratext *) "EntRef")) ||
        !elem->appendChild(entref))
FAIL

    if (!(fish = doc->createElement((oratext *) "FISH")) ||
!elem->appendChild(fish))
FAIL

    if (!(food = doc->createElement((oratext *) "FOOD")) ||
!elem->appendChild(food))
FAIL
    cout << "Document from 'ROOT' element with its 7 children:\n";
    dump(root, 0);

    cout << "\nTesting node insertion...\n";
    cout << "Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment
node...\n";
    if (!(node = doc->createTextNode((oratext *) "Pre-Gibberish")) ||
        !elem->insertBefore(node, text))
FAIL

    if (!(node = doc->createComment((oratext *) "Ask about the weather:")) ||
        !elem->insertBefore(node, comment))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
```

```
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nTesting nextSibling links starting at first child...\n";
    for (node = elem->getFirstChild();
    node;
    node = node->getNextSibling())dump(node, 1);
    cout << "\nTesting previousSibling links starting at last child...\n";
    for (node = elem->getLastChild();
    node;
    node = node->getPreviousSibling())dump(node, 1);

    cout << "\nTesting setting node value...\n";
    cout << "Original node:\n";
    dump(pi, 1);
    pi->setValue((oratext *) "New PI contents");
    cout << "Node after new value:\n";
    dump(pi, 1);

    cout << "\nAdding another element level, i.e., 'SUB'...\n";
    if (!(subelem = doc->createElement((oratext *) "SUB")) ||
!elem->insertBefore(subelem, cdata) ||
!(subtext = doc->createTextNode((oratext *) "Lengthy SubText")) ||
        !subelem->appendChild(subtext))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);

    cout << "\nAdding a second 'SUB' element...\n";
    if (!(sub2 = doc->createElement((oratext *) "SUB")) ||
!elem->insertBefore(sub2, cdata))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);

    cout << "\nGetting all SUB nodes - note the distinct hex addresses...\n";
    if (!(subs = doc->getElementsByTagName(root, (oratext *) "SUB")))
FAIL
    for (i = 0; i < subs->getLength(); i++)
dumpnode(subs->item(i), 1);

    cout << "\nTesting parent links...\n";
    for (level = 1, node = subtext; node; node = node->getParentNode(), level++)
dumpnode(node, level);
```

```
    cout << "\nTesting owner document of node...\n";
    dumpnode(subtext, 1);
    dumpnode(subtext->getOwnerDocument(), 1);

    cout << "\nTesting node replacement...\n";
    if (!(node = doc->createTextNode((oratext *) "REPLACEMENT, 1/2 PRICE")) ||
        !pi->replaceChild(node))
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);

    cout << "\nTesting node removal...\n";
    if (!entref->removeChild())
FAIL

    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nNormalizing...\n";
    elem->normalize();
    cout << "Document from 'ROOT' element:\n";
    dump(root, 0);
    cout << "\nCreating and populating document fragment...\n";
    if (!(frag = doc->createDocumentFragment()) ||
!(fragelem = doc->createElement((oratext *) "FragElem")) ||
!(fragtext = doc->createTextNode((oratext *) "FragText")) ||
!frag->appendChild(fragelem) ||
!frag->appendChild(fragtext))
FAIL
    dump(frag, 1);
    cout << "Insert document fragment...\n";
    if (!elem->insertBefore(frag, comment))
FAIL
    dump(elem, 1);

    cout << "\nCreate two attributes...\n";
    if (!(attr1 = doc->createAttribute((oratext*)"Attr1",(oratext*)"Value1")) ||
!(attr2 = doc->createAttribute((oratext*)"Attr2",(oratext*)"Value2")))
FAIL
    cout << "Setting attributes...\n";
    if (!subelem->setAttributeNode(attr1, NULL) ||
!subelem->setAttributeNode(attr2, NULL))
FAIL
    dump(subelem, 1);

    cout << "\nAltering attribute1 value...\n";
```

```
    attr1->setValue((oratext *) "New1");
    dump(subelem, 1);

    cout << "\nFetching attribute by name (Attr2)...\n";
    if (!(node = subelem->getAttributeNode((oratext *) "Attr2")))
FAIL
    dump(node, 1);

    cout << "\nRemoving attribute by name (Attr1)...\n";
    subelem->removeAttribute((oratext *) "Attr1");
    dump(subelem, 1);

    cout << "\nAdding new attribute...\n";
    if (!subelem->setAttribute((oratext *) "Attr3", (oratext *) "Value3"))
FAIL
    dump(subelem, 1);

    cout << "\nRemoving attribute by pointer (Attr2)...\n";
    if (!subelem->removeAttributeNode(attr2))
FAIL
    dump(subelem, 1);

    cout << "\nAdding new attribute w/same name (test replacement)...\n";
    dump(subelem, 1);
    if (!(attr = doc->createAttribute((oratext*)"Attr3", (oratext*)"Zoo3")))
FAIL
    if (!subelem->setAttributeNode(attr, NULL))
FAIL
    dump(subelem, 1);

    cout << "\nTesting node (attr) set by name...\n";
    cout << "Adding 'GLEEP' attr and printing out hex addresses of node set\n";
    attrs = subelem->getAttributes();
    if (!(gleep1=doc->createAttribute((oratext*)"GLEEP",(oratext*)"GLEEP1")) ||
!attrs->setNamedItem(gleep1, NULL))
FAIL
    dump(subelem, 0);

    cout << "\nTesting node (attr) set by name...\n";
    cout << "Replacing 'GLEEP' element - note the changed hex address\n";
    if (!(gleep2=doc->createAttribute((oratext*)"GLEEP",(oratext*)"GLEEP2")) ||
!attrs->setNamedItem(gleep2, &repl))
FAIL
    dump(subelem, 0);
    cout << "Replaced node was:\n";
    dump(repl, 1);
```

```
    cout << "\nTesting node removal by name...\n";
    cout << "Removing 'GLEEP' attribute\n";
    if (!attrs->removeNamedItem((oratext *) "GLEEP"))
FAIL
    dump(subelem, 0);

    cout << "\nOriginal SubROOT...\n";
    dump(subelem, 1);
    cout << "Cloned SubROOT (not deep)...\n";
    clone = subelem->cloneNode(FALSE);
    dump(clone, 1);
    cout << "Cloned SubROOT (deep)...\n";
    deep_clone = subelem->cloneNode(TRUE);
    dump(deep_clone, 1);

    cout << "\nSplitting text...\n";
    dump(subelem, 1);
    subtext->splitText(3);
    dump(subelem, 1);

    cout << "\nTesting string operations...\n";
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Setting new data...\n";
    subtext->setData((oratext *) "0123456789");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "    CharLength = " << (int) subtext->getLength() << "\n";
    cout << "    Substring(0,5) = \"" <<
(char *) subtext->substringData(0, 5) << "\"\n";
    cout << "    Substring(8,2) = \"" <<
(char *) subtext->substringData(8, 2) << "\"\n";
    cout << "Appending data...\n";
    subtext->appendData((oratext *) "ABCDEF");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Inserting data...\n";
    subtext->insertData(10, (oratext *) "*foo*");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Deleting data...\n";
    subtext->deleteData(0, 10);
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";
    cout << "Replacing data...\n";
    subtext->replaceData(1, 3, (oratext *) "bamboozle");
    cout << "    CharData = \"" << (char *) subtext->getData() << "\"\n";

    cout << "Cleaning up...\n";
    parser.xmlclean();

    if (parser.getDocument())
```

```
    {
cout << "Problem, document is not gone!!\n";
return 1;
    }

    cout << "Parsing test document...\n";
    if (ecode = parser.xmlparse(TEST_DOC, (oratext *) 0, 0))
    {
cout << "Parse failed, code " << ecode << "\n";;
return ecode;
    }

    cout << "Document from root node:\n" << flush;
    dump(parser.getDocument(), 0);

    cout << "Testing getNotations...\n" << flush;
    dtd = parser.getDocType();
    if (notes = dtd->getNotations())
    {
cout << "# of notations = " << notes->getLength() << "\n" << flush;
for (i = 0; i < notes->getLength(); i++)
    dump(notes->item(i), 1);
    }
    else
cout << "No defined notations\n" << flush;

    cout << "Testing getEntities...\n" << flush;
    if (entities = dtd->getEntities())
    {
cout << "# of entities = " << entities->getLength() << "\n" << flush;
for (i = 0; i < entities->getLength(); i++)
    dump(entities->item(i), 1);
    }
    else
cout << "No defined entities\n" << flush;

    cout << "Cleaning up...\n";
    parser.xmlclean();

    if (parser.getDocument())
    {
cout << "Problem, document is not gone!!\n";
return 1;
    }

    cout << "\nTerminating parser...\n";
    parser.xmlterm();
```

```
    cout << "Success.\n";
    return 0;
}

void dump(Node *node, uword level)
{
    NodeList *nodes;
    uword     i, n_nodes;

    if (node)
    {
dumpnode(node, level);
if (node->hasChildNodes())
{
    nodes = node->getChildNodes();
    n_nodes = node->numChildNodes();
    for (i = 0; i < n_nodes; i++)
dump(nodes->item(i), level + 1);
}
    }
}

void dumpnode(Node *node, uword level)
{
    const oratext *name, *value;
    short          type;
    NamedNodeMap  *attrs;
    Attr          *attr;
    uword          i, n_attrs;

    if (node)
    {
for (i = 0; i <= level; i++)
    cout << "    ";
type = node->getType();
cout << (char *) ntypename[type];
if ((name = node->getName()) && (*name != '#'))
    cout << " \"" << (char *) name << "\"";
if (value = node->getValue())
    cout << " = \"" << (char *) value << "\"";
if ((type == ELEMENT_NODE) && (attrs = node->getAttributes()))
{
    cout << " [";
    n_attrs = attrs->getLength();
    for (i = 0; i < n_attrs; i++)
    {
```

```
if (i) cout << ", ";
attr = (Attr *) attrs->item(i);
cout << (char *) attr->getName();
if (attr->getSpecified())
    cout << "*";
cout << "=\"" << (char *) attr->getValue() << "\"";
    }
    cout << "]";
}
cout << "\n";
    }
}

// end of FullDOM.cpp
```

XML Parser for C++ の例 15: C++ - FullDOM.std

FullDOM.std は、FullDOM.cpp からの予想される出力を表示します。

```
XML C++ Full DOM test
Initializing XML parser...

Creating new document...
Document from root node:
    DOCUMENT

Creating root element ('ROOT')...
Setting as root element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
    ELEMENT "ROOT"
        TEXT = "Gibberish"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
```

```
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing nextSibling links starting at first child...
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "PI-contents"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing previousSibling links starting at last child...
        ELEMENT "FOOD"
        ELEMENT "FISH"
        ENTREF "EntRef"
        CDATA = "See DATA"
```

```
        PI "target" = "PI-contents"
        COMMENT = "Bit warm today, innit?"
        COMMENT = "Ask about the weather:"
        TEXT = "Gibberish"
        TEXT = "Pre-Gibberish"

Testing setting node value...
Original node:
        PI "target" = "PI-contents"
Node after new value:
        PI "target" = "New PI contents"

Adding another element level, i.e., 'SUB'...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Adding a second 'SUB' element...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        PI "target" = "New PI contents"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Getting all SUB nodes - note the distinct hex addresses...
        ELEMENT "SUB"
        ELEMENT "SUB"
```

```
Testing parent links...
        TEXT = "Lengthy SubText"
            ELEMENT "SUB"
                ELEMENT "ROOT"
                    DOCUMENT

Testing owner document of node...
        TEXT = "Lengthy SubText"
        DOCUMENT

Testing node replacement...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ENTREF "EntRef"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Testing node removal...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-Gibberish"
        TEXT = "Gibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Normalizing...
Document from 'ROOT' element:
    ELEMENT "ROOT"
        TEXT = "Pre-GibberishGibberish"
        COMMENT = "Ask about the weather:"
        COMMENT = "Bit warm today, innit?"
```

```
        TEXT = "REPLACEMENT, 1/2 PRICE"
        ELEMENT "SUB"
            TEXT = "Lengthy SubText"
        ELEMENT "SUB"
        CDATA = "See DATA"
        ELEMENT "FISH"
        ELEMENT "FOOD"

Creating and populating document fragment...
        DOCFRAG
            ELEMENT "FragElem"
            TEXT = "FragText"
Insert document fragment...
        ELEMENT "ROOT"
            TEXT = "Pre-GibberishGibberish"
            COMMENT = "Ask about the weather:"
            ELEMENT "FragElem"
            TEXT = "FragText"
            COMMENT = "Bit warm today, innit?"
            TEXT = "REPLACEMENT, 1/2 PRICE"
            ELEMENT "SUB"
                TEXT = "Lengthy SubText"
            ELEMENT "SUB"
            CDATA = "See DATA"
            ELEMENT "FISH"
            ELEMENT "FOOD"

Create two attributes...
Setting attributes...
        ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Altering attribute1 value...
        ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Fetching attribute by name (Attr2)...
        ATTRIBUTE "Attr2" = "Value2"

Removing attribute by name (Attr1)...
        ELEMENT "SUB" [Attr2*="Value2"]
            TEXT = "Lengthy SubText"

Adding new attribute...
        ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
            TEXT = "Lengthy SubText"
```

```
Removing attribute by pointer (Attr2)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"

Adding new attribute w/same name (test replacement)...
        ELEMENT "SUB" [Attr3*="Value3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Testing node (attr) set by name...
Adding 'GLEEP' attr and printing out hex addresses of node set
    ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP1"]
        TEXT = "Lengthy SubText"

Testing node (attr) set by name...
Replacing 'GLEEP' element - note the changed hex address
    ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP2"]
        TEXT = "Lengthy SubText"
Replaced node was:
        ATTRIBUTE "GLEEP" = "GLEEP1"

Testing node removal by name...
Removing 'GLEEP' attribute
    ELEMENT "SUB" [Attr3*="Zoo3"]
        TEXT = "Lengthy SubText"

Original SubROOT...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (not deep)...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
Cloned SubROOT (deep)...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"

Splitting text...
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Lengthy SubText"
        ELEMENT "SUB" [Attr3*="Zoo3"]
            TEXT = "Leng"
            TEXT = "thy SubText"

Testing string operations...
    CharData = "Leng"
Setting new data...
```

```
    CharData = "0123456789"
    CharLength = 10
    Substring(0,5) = "01234"
    Substring(8,2) = "89"
Appending data...
    CharData = "0123456789ABCDEF"
Inserting data...
    CharData = "0123456789*foo*ABCDEF"
Deleting data...
    CharData = "*foo*ABCDEF"
Replacing data...
    CharData = "*bamboozle*ABCDEF"
Cleaning up...
Parsing test document...
Document from root node:
    DOCUMENT
        DTD "doc"
        ELEMENT "doc" [xml:lang*="foo"]
            ELEMENT "p" [xml:space="preserve"]
                TEXT = "An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;)."
Testing getNotations...
# of notations = 2
        NOTATION "notation1"
        NOTATION "notation2"
Testing getEntities...
# of entities = 1
        ENTITY "example" = "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>"
Cleaning up...

Terminating parser...
Success.
```

XML Parser for C++ の例 16: C++ - XSLSample.cpp

XSLSample.cpp には、XSLSample の C++ ソース・コードが含まれます。

```
// Copyright (c) Oracle Corporation 1999. All Rights Reserved.

/////////////////////////////////////////////////////////////////////////////
// NAME
//   XSLSample.cpp
//
// DESCRIPTION
//   Sample usage of C++ XSL processor
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
//   none
/////////////////////////////////////////////////////////////////////////////

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

int main(int argc, char **argv)
{
    XMLParser     xmlpar, xslpar, respar;
    XSLProcessor  xslproc;
    Node         *result;
    ub4           flags;
    uword         ecode;
    flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

    cout << "XSL processor sample\n";

    if (argc < 3)
    {
cout << "Usage is XSLSample <xmlfile> <stlyesheet>\n";
return 1;
    }

    // Parse the XML file
    cout << "Parsing XML file " << argv[1] << "\n";
    if (ecode = xmlpar.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
```

```
    }
    if (ecode = xmlpar.xmlparse((oratext *) argv[1], (oratext *) 0, flags))
        return 1;

    // Parse the Stylesheet file
    cout << "Parsing Stylesheet " << argv[2] << "\n";
    if (ecode = xslpar.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
    }
    if (ecode = xslpar.xmlparse((oratext *) argv[2], (oratext *) 0, flags))
        return 1;

    // Initialize the result context
    cout << "Initializing the result context\n";
    if (ecode = respar.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode << "\n";;
        return 1;
    }

    // XSL Processing
    cout << "XSL Processing\n";
    if (ecode = xslproc.xslprocess(&xmlpar, &xslpar, &respar, &result))
    {
        cout << "Failed in XSL Processing, error " << ecode << "\n";;
        return 1;
    }

    // print the resultant tree
    cout.flush();
    xslproc.printres(&respar, result);


    // Terminate the parsers
    (void) xmlpar.xmlterm();
    (void) xslpar.xmlterm();
    (void) respar.xmlterm();

    return 0;
}
```

XML Parser for C++ の例 17: C++ - XSLSample.std

XSLSample.std は、XSLSample.cpp からの予想される出力を表示します。

```
<xsl:param name="size"/>
<xsl:param name="data"/>
<xsl:choose><xsl:when test="number(number($size) < string-length(string($data)))">
   <xsl:value-of select="substring(string($data), 1, number($size))"/></xsl:when>
<xsl:otherwise>
<xsl:value-of select="string($data)"/>
</xsl:otherwise></xsl:choose>
<xsl:if test="number(number($size) > string-length(string($data)))">
<xsl:call-template name="pad">
<xsl:with-param name="padsize" select="number($size) -
string-length(string($data))"/>
</xsl:call-template></xsl:if></xsl:template><xsl:template match="/">
<xsl:text>&#13;&#10;</xsl:text><xsl:apply-templates select="//ROWSE
T/ROW/CUSTOMER"/>
   <xsl:text>&#13;&#10;</xsl:text>
</xsl:template><xsl:template match="CUSTOMER">
<xsl:call-template name="truncateorpad">
<xsl:with-param name="size" select="31"/>
<xsl:with-param name="data" select =","/>
</xsl:call-template>
</xsl:template>
</xsl:stylesheet>
```

27

XML Schema Processor for C++ の使用

この章の内容は次のとおりです。

XML Schema Processor for C++ の機能

XML Schema Processor for C++ は、XML Parser for C++ とともに動作するコンポーネントです。XML Schema Processor for C++ を使用すると、Oracle の XML アプリケーションで単純および複雑なデータ型をサポートできます。

XML Schema Processor for C++ は、XML Schema 草案が W3C 勧告になった場合、勧告に完全に準拠することを目標として、XML Schema をサポートしています。このため、Oracle 環境では、XML 文書を処理するカスタム・アプリケーションを簡単に作成できます。また、Oracle が移植されたすべてのオペレーティング・システムには、標準に準拠した XML Schema Processor が Oracle プラットフォームの一部として搭載されています。

> **参照：** XML Schema の詳細および XML Schema を使用する理由については、第 21 章「XML Schema Processor for Java の使用」を参照してください。

XML Schema Processor for C++ には、次の機能があります。

- 単純型および複合型をサポートします。
- XML Parser for C++ に統合されています。
- W3C の XML Schema 草案（2000 年 4 月 7 日のバージョン）に基づいています。

XML Schema Processor for C++ のクラスは `XMLSchema` です。この章で説明するバージョンは、Oracle XML Schema Processor for C++ 1.0.1.0.0 です。Oracle XML Schema Processor は、C++ ラッパーを使用して C で作成されています。これには XML Parser for C が含まれています。

> **参照：** 『Oracle9*i* XML リファレンス』を参照してください。

要件

XML Schema Processor for C++ は、次のオペレーティング・システム上で動作します。

- Linux
- Solaris
- HP/UX
- Windows NT4.0/Service Pack 3 以上

オンライン・ドキュメント

Oracle XML Schema Processor for C++ のドキュメントは、インストール場所の doc/ ディレクトリにあります。

標準への準拠

XML Schema Processor for C++ は、次の標準に準拠しています。

- W3C の XML 1.0 勧告
- W3C の DOM レベル 1.0 勧告
- W3C の XML Namespace 勧告案
- SAX 1.0
- W3C の XSLT 勧告
- W3C の XPath 勧告

サポートされているキャラクタ・セットの使用

XML Parser for C++ は、現在、次のエンコーディングをサポートしています。

- BIG5
- EBCDIC-CP-BE
- EBCDIC-CP-CA
- EBCDIC-CP-CH
- EBCDIC-CP-DK
- EBCDIC-CP-ES
- EBCDIC-CP-FI
- EBCDIC-CP-FR
- EBCDIC-CP-GB
- EBCDIC-CP-HE
- EBCDIC-CP-IS
- EBCDIC-CP-IT
- EBCDIC-CP-NL

- EBCDIC-CP-NO
- EBCDIC-CP-ROECE
- EBCDIC-CP-SE
- EBCDIC-CP-US
- EBCDIC-CP-WT
- EBCDIC-CP-YU
- EUC-JP
- GB2312
- ISO-10646-UCS-2
- ISO-8859-1 ～ ISO-8859-9
- KOI8-RUTF-8
- SHIFT_JIS
- US-ASCII
- UTF-16

> **参照：**『Oracle9*i* グローバリゼーション・サポート・ガイド』の付録 A の「キャラクタ・セット」で指定されているキャラクタ・セットも使用できます。

これらのエンコーディングを使用するには、次の設定が必要です。

- 環境変数 ORACLE_HOME が、Oracle のインストール場所を指すように設定する必要があります。
- 環境変数 ORA_NLS、ORA_NLS32 および ORA_NLS33 が、NLS データ・ファイルの場所を指すように設定する必要があります。これらのデータ・ファイルは、通常、次の場所にあります。
  - UNIX システム : `$ORACLE_HOME/ocommon/nls/admin/data`
  - Windows NT: `$ORACLE_HOME¥nlsrtl¥admin¥nlsdata`

デフォルトでは、UTF-8 がエンコーディングになります。シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 2 倍向上します。

XML Schema Processor for C++: ソフトウェア

表 27-1 に、今回のリリースで提供されるファイルおよびディレクトリを示します。

表 27-1 XML Schema Processor for C++: 提供されるファイル

ディレクトリおよびファイル	説明
license.html	ライセンス契約
readme.html	このファイル
bin/	Schema Processor が実行可能なスキーマ
doc/	API ドキュメント
include/	ヘッダー・ファイル
lib/	XML/XSL/Schema およびサポート・ライブラリ
mesg/	エラー・メッセージ・ファイル
sample/	スキーマ・プロセッサの使用例

表 27-2 に、含まれるライブラリを示します。

表 27-2 XML Schema Processor for C++: 提供されるライブラリ

含まれるライブラリ	説明
libxml8.a	XML パーサー /XSL プロセッサ
libcore8.a	CORE の機能
libnls8.a	National Language Support

XML Schema Processor for C++ の起動

XML Schema Processor for C++ は、インストール場所の **`bin/schema`** をコールすることによって、実行可能ファイルとしてコールされます。これは次の 2 つの引数を取ります。

- XML インスタンス・ドキュメント
- デフォルト・スキーマ（オプション）

XML Schema Processor for C++ は、提供される API を使用するためのコードを記述して起動することもできます。このコードは、`include/` サブディレクトリにあるヘッダーを使用してコンパイルし、`lib/` サブディレクトリにあるライブラリにリンクする必要があります。プログラムを構築する方法の詳細は、`sample/` サブディレクトリにある `Makefile` を参照してください。

エラー・メッセージ・ファイルは、mesg/ サブディレクトリにあります。現在、メッセージ・ファイルは英語表記のみです。他の言語のメッセージ・ファイルは、将来のリリースで提供される予定です。

環境変数 OR_XML_MESG の設定による絶対パスの指定

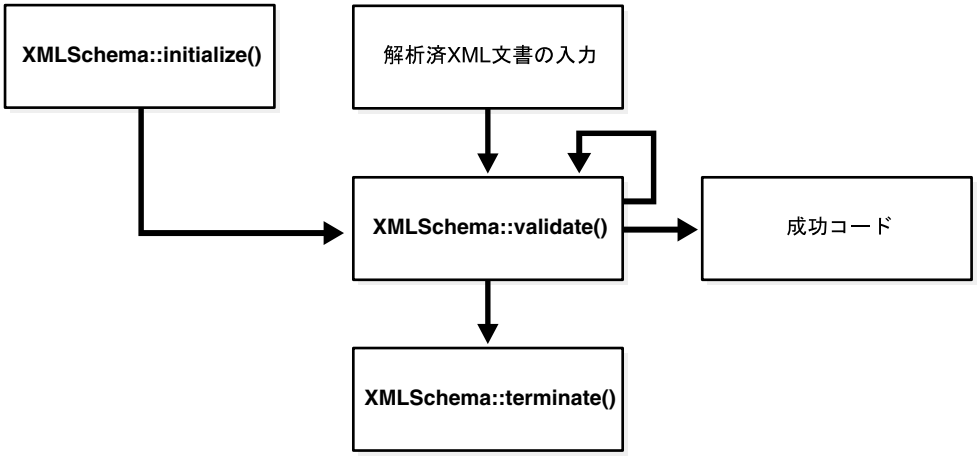
環境変数 ORA_XML_MESG を設定し、mesg/ サブディレクトリの**絶対**パスを指定する必要があります。または、$ORACLE_HOME をインストールしている場合、mesg/ サブディレクトリの内容を $ORACLE_HOME/oracore/mesg ディレクトリにコピーすることもできます。

XML Schema Processor for C++ の使用方法

図 27-1 に、次に示す XML Schema Processor for C++ のコール順序を示します。

1. XMLSchema.initialize() メソッドが処理を初期化します。
2. 1 つ以上の解析済 XML 文書が、XML Schema Processor for C++ を入力します。
3. XMLSchema.validate() が、1 つ以上の解析済 XML 文書を、成功コードが戻るまで検証します。
4. 検証が完了すると、XMLSchema.terminate() メソッドが処理を終了します。

図 27-1 XML Schema Processor for C++ の使用方法



XML Schema Processor for C++ サンプル・プログラムの実行

sample/ ディレクトリには、API による Oracle XML Schema Processor の使用方法を示すサンプル XML Schema アプリケーションがあります。表 27-3 に、提供されるサンプル・ファイルを示します。

表 27-3 XML Schema for C++ のサンプル・ファイル

サンプル・ファイル	説明
Makefile	サンプル・プログラムを作成および実行し、適切な出力を確認する Make ファイル
xsdtest.cpp	XML Schema for C++ の API をコールする一般的なプログラム
car.{xsd,xml,std}	xsdtest を実行した後のサンプル・スキーマ、インスタンス・ドキュメント、および予想される出力。次の例を参照してください。 27-9 ページの「XML Schema for C++ の例 2: car.xsd」 27-11 ページの「XML Schema for C++ の例 3: car.xml」 27-11 ページの「XML Schema for C++ の例 4: car.std」
aq.{xsd,xml,std}	xsdtest を実行した後の 2 つ目のサンプル・スキーマ、インスタンス・ドキュメント、および予想される出力。次の例を参照してください。 27-11 ページの「XML Schema for C++ の例 5: aq.xsd」 27-16 ページの「XML Schema for C++ の例 6: aq.xml」 27-17 ページの「XML Schema for C++ の例 7: aq.std」
pub.{xsd,xml,std}	xsdtest を実行した後の 3 つ目のサンプル・スキーマ、インスタンス・ドキュメント、および予想される出力。次の例を参照してください。 27-17 ページの「XML Schema for C++ の例 8: pub.xsd」 27-19 ページの「XML Schema for C++ の例 9: pub.xml」 27-20 ページの「XML Schema for C++ の例 10: pub.std」

サンプル・プログラムを作成するには、「**make**」を実行します。

プログラムを作成および実行し、実際の出力と予想される出力を比較するには、「**make sure**」を実行します。

エラー・メッセージ（英語）

エラー・メッセージ・ファイルは mesg サブディレクトリにあります。現在、メッセージ・ファイルは英語表記のみです。他の言語のメッセージ・ファイルは、将来のリリースで提供される予定です。

環境変数 ORA_XML_MESG を設定し、mesg サブディレクトリの絶対パスを指定する必要があります。または、$ORACLE_HOME をインストールしている場合、mesg サブディレクトリの内容を $ORACLE_HOME/oracore/mesg ディレクトリにコピーすることもできます。

XML Schema for C++ の例 1: xsdtest.cpp

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

///////////////////////////////////////////////////////////////////////////
//   NAME validate.cpp
//   DESCRIPTION Sample usage of C++ XML Schema processor
///////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <string.h>

#ifndef ORAXML_CPP_ORACLE
# include <oraxml.hpp>
#endif

#ifndef ORAXSD_CPP_ORACLE
# include <oraxsd.hpp>
#endif

int main(int argc, char **argv)
{
    XMLSchema   schema;
    XMLParser   parser;
    xmlctx      *ctx;
    char        *doc, *uri;
    uword       ecode;

    cout << "XML C++ Schema processor\n";

    if ((argc < 2) || (argc > 3))
    {
       cout << "usage: validate <xml document> [schema]\n";
       return -1;
    }
    doc = argv[1];
    uri = (argc > 2) ? argv[2] : 0;
```

```
    cout << "Initializing XML package...\n";

    if (ecode = parser.xmlinit())
    {
        cout << "Failed to initialize XML parser, error " << ecode;
        return 1;
    }

    cout << "Parsing '" << doc << "'...\n";
    if (ecode = parser.xmlparse((oratext *) doc, (oratext *) 0,
      XML_FLAG_DISCARD_WHITESPACE))
    {
        cout << "Parse failed, error " << ecode << "\n";
        return 2;
    }

    cout << "Initializing Schema package...\n";

    if (ecode = schema.initialize(&parser))
    {
        cout << "Failed, code " << ecode << "!\n";
        return 3;
    }

    cout << "Validating document...\n";
    if (ecode = schema.validate(&parser, (oratext *) uri))
    {
        cout << "Validation failed, error " << ecode << "\n";
        return 4;
    }

    cout << "Document is valid.\n";
    schema.terminate();
    return 0;
}
```

XML Schema for C++ の例 2: car.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.CarDealers.com/">
    <element name="Car">
  <complexType>
    <element name="Model">
  <simpleType base="string">
```

```
   <enumeration value = "Ford"/>
   <enumeration value = "Saab"/>
   <enumeration value = "Audi"/>
 </simpleType>
   </element>
   <element name="Make">
 <simpleType base="string">
   <minLength value = "1"/>
   <maxLength value = "30"/>
</simpleType>
   </element>
   <element name="Year">
<complexType content="mixed">
   <attribute name="PreviouslyOwned" type="string"
      use="required"/>
   <attribute name="YearsOwned" type="integer"
      use="optional"/>
</complexType>
   </element>
   <element name="OwnerName" type="string"
     minOccurs="0" maxOccurs="unbounded"/>
   <element name="Condition">
<complexType base="string" derivedBy="extension">
   <attribute name="Automatic">
<simpleType base="string">
   <enumeration value = "Yes"/>
   <enumeration value = "No"/>
</simpleType>
   </attribute>
</complexType>
   </element>
   <element name="Mileage">
<simpleType base="integer">
   <minInclusive value="0"/>
   <maxInclusive value="2000000"/>
</simpleType>
   </element>
   <attribute name="RequestDate" type="date"/>
</complexType>
   </element>
</schema>
```

XML Schema for C++ の例 3: car.xml

```
<?xml version="1.0"?>
<car:Car xmlns:car="http://www.CarDealers.com/"
     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
     xsi:schemaLocation="http://www.CarDealers.com/ car.xsd"
     RequestDate="2000-12-6">

    <Model>Ford</Model>
    <Make>Explorer</Make>
    <Year PreviouslyOwned="You betcha">1999</Year>
    <OwnerName>Joe Smith</OwnerName>
    <OwnerName>Bob Jones</OwnerName>
    <Condition Automatic="No">Small dent on right bumper.</Condition>
    <Mileage>1999999</Mileage>
</car:Car>
```

XML Schema for C++ の例 4: car.std

```
XML C++ Schema processor
Initializing XML package...
Parsing 'car.xml'...
Initializing Schema package...
Validating document...
Document is valid.
```

XML Schema for C++ の例 5: aq.xsd

```
<?xml version="1.0"?>
<!-- ****************** AQ xml schema  ****************** -->
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.oracle.com/AQXmlDocument"
        xmlns:aq = "http://www.oracle.com/AQXmlDocument"
        xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
elementFormDefault="qualified">

   <element name="AQXmlDocument">
      <complexType content="mixed">
        <choice>
              <group ref="aq:client_operation" minOccurs="0"/>
              <group ref="aq:server_response"/>
        </choice>
      </complexType>
    </element>
```

```
    <!-- ***************** Client Operations Group  ***************** -->
    <group name="client_operation">
        <sequence>
           <element ref="aq:client_operation" minOccurs="0" maxOccurs="1"/>
          <choice>
                  <element ref="aq:producer_options" maxOccurs="1"/>
                  <element ref="aq:consumer_options" maxOccurs="1"/>
                  <element ref="aq:register_options" maxOccurs="1"/>
                </choice>
           <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
        </sequence>
     </group>

    <!-- ***************** Server Response Group ***************** -->
    <group name="server_response">
        <sequence>
           <element ref="aq:server_response" minOccurs="0" maxOccurs="1"/>
           <element ref="aq:receive_result" maxOccurs="1"/>
           <choice  minOccurs="0" >
               <element ref="aq:send_result" maxOccurs="1"/>
               <element ref="aq:publish_result" maxOccurs="1"/>
               <element ref="aq:receive_result" maxOccurs="1"/>
               <element ref="aq:sequence_num_result" maxOccurs="1"/>
           </choice>
        </sequence>
    </group>

    <!-- ***************** Server Propagation Group ***************** -->
    <group name="server_prop_operation">
        <sequence>
           <element ref="aq:server_prop_operation" minOccurs="0" maxOccurs="1"/>
           <choice>
               <element ref="aq:push" maxOccurs="1"/>
               <element ref="aq:notification" maxOccurs="1"/>
               <element ref="aq:sequence_num_request" maxOccurs="1"/>
           </choice>
        </sequence>
    </group>

    <!-- ***************** Client Operation ***************** -->
    <element name="client_operation">
       <complexType content="mixed">
         <element ref="aq:txid" minOccurs="0"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
       </complexType>
    </element>
```

```
   <!-- ****************** Server Response ****************** -->
   <element name="server_response">
      <complexType content="mixed">
        <element ref="aq:txid" minOccurs="0"/>
        <element ref="aq:status_response" minOccurs="1"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
      </complexType>
   </element>

   <!-- ****************** Server Propagation Operation ****************** -->
   <element name="server_prop_operation">
      <complexType content="mixed">
        <element ref="aq:txid" minOccurs="0"/>
<attribute name="prop_opcode" use="required" type="aq:prop_opcode_type"/>
      </complexType>
   </element>

   <element name="txid" type="string"/>
....

   <!-- ****************** Message payload ****************** -->
   <element name="message_payload">
     <complexType>
        <choice>
          <element ref="aq:jms_text_message" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:jms_map_message" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:jms_bytes_message" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:jms_object_message" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="*" processContents="skip"/>
        </choice>
     </complexType>
   </element>

   <!-- ****************** User-defined properties ****************** -->
   <element name="user_properties">
      <complexType content="mixed">
        <element ref="aq:property" minOccurs="0" maxOccurs="*"/>
      </complexType>
   </element>

   <!-- ****************** Property  ****************** -->
   <element name="property">
      <complexType content="mixed">
        <element ref="aq:name" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:value" minOccurs="1" maxOccurs="1"/>
<attribute name="property_type" type="aq:prop_type"/>
       </complexType>
```

```
 </element>

<!-- ***************** Status response ***************** -->
<element name="status_response">
   <complexType content="mixed">
     <element ref="aq:acknowledge" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:status_code" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:error_code" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:error_message" minOccurs="0" maxOccurs="1"/>
   </complexType>
</element>

<!-- ***************** Send result ***************** -->
<element name="send_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ***************** Publish result ***************** -->
<element name="publish_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ***************** Receive result ***************** -->
<element name="receive_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

.
.
<!-- ***************** JMS text message ***************** -->
<element name="jms_text_message">
   <complexType content="mixed">
     <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:text_data" minOccurs="1" maxOccurs="1"/>
   </complexType>
</element>
```

```
   <element name="text_data" type="string"/>

   <!-- ***************** JMS map message ***************** -->
   <element name="jms_map_message">
      <complexType content="mixed">
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:map_data" minOccurs="1" maxOccurs="1"/>
      </complexType>
   </element>

   <!-- ***************** Map data ***************** -->
   <element name="map_data">
      <complexType content="mixed">
        <element ref="aq:item" minOccurs="0" maxOccurs="*"/>
      </complexType>
   </element>

   <!-- ***************** Map Item  ***************** -->
   <element name="item">
      <complexType content="mixed">
        <element ref="aq:name" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:value" minOccurs="1" maxOccurs="1"/>
<attribute name="item_type" type="aq:prop_type"/>
       </complexType>
    </element>

   <!-- ***************** JMS bytes message ***************** -->
   <element name="jms_bytes_message">
      <complexType content="mixed">
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:bytes_data" minOccurs="1" maxOccurs="1"/>
      </complexType>
   </element>

   <element name="bytes_data" type="string"/>


   <!-- ***************** JMS object message ***************** -->
   <element name="jms_object_message">
      <complexType content="mixed">
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:ser_object_data" minOccurs="1" maxOccurs="1"/>
      </complexType>
```

```
    </element>
    <element name="ser_object_data" type="string"/>

</schema>
```

XML Schema for C++ の例 6: aq.xml

```
<AQXmlDocument xmlns="http://www.oracle.com/AQXmlDocument"
       xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
       xsi:schemaLocation="http://www.oracle.com/AQXmlDocument aq.xsd">
    <client_operation opcode="SEND">
        <txid> sdasdfdsf </txid>
    </client_operation>

    <producer_options delivery_mode="PERSISTENT">
        <destination lookup_type="NORMAL"> queue1 </destination>
        <priority>23</priority>
        <recipient_list>
            <recipient> abc </recipient>
            <recipient lookup_type="LDAP"> abc </recipient>
        </recipient_list>
    </producer_options>

    <message_set>
        <message_count>1</message_count>
        <message>
            <message_number>1</message_number>
            <message_header>
                <correlation>XML_40_NEW_TEST</correlation>
                <delay>10</delay>
                <sender_id>scott::home::0</sender_id>
            </message_header>
            <message_payload>
                <jms_map_message>
                    <oracle_jms_properties>
                        <reply_to>oracle::redwoodshores::100</reply_to>
                        <userid>scott</userid>
                        <appid>AQProduct</appid>
                        <groupid>AQ</groupid>
                    </oracle_jms_properties>
                    <user_properties>
                        <property property_type="STRING">
                            <name>country</name>
                            <value>USA</value>
                        </property>
                        <property property_type="STRING">
                            <name>State</name>
```

```
                        <value>california</value>
                    </property>
                </user_properties>
                <map_data>
                    <item item_type="STRING">
                        <name>Car</name>
                        <value>Toyota</value>
                    </item>
                    <item item_type="STRING">
                        <name>Color</name>
                        <value>Blue</value>
                    </item>
                    <item item_type="STRING">
                        <name>Shape</name>
                        <value>Circle</value>
                    </item>
                    <item item_type="NUMBER">
                        <name>Price</name>
                        <value>20000</value>
                    </item>
                </map_data>
            </jms_map_message>
        </message_payload>
    </message>
</message_set>
</AQXmlDocument>
```

XML Schema for C++ の例 7: aq.std

```
XML C++ Schema processor
Initializing XML package...
Parsing 'aq.xml'...
Initializing Schema package...
Validating document...
Document is valid.
```

XML Schema for C++ の例 8: pub.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/2000/08/XMLSchema"
        targetNamespace = "http://www.somewhere.org/BookCatalogue"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue"
        elementFormDefault="qualified">
    <complexType name="Pub">
      <sequence>
        <element name="Title" type="cat:titleType" maxOccurs="*"/>
```

```
    <element name="Author" type="string" maxOccurs="*"/>
    <element name="Date" type="date"/>
  </sequence>
  <attribute name="language" type="string" use="default" value="English"/>
  <anyAttribute namespace="##local"/>
</complexType>
<element name="Publication" type="cat:Pub"  abstract="true"/>
<element name="Book" substitutionGroup="cat:Publication">
    <complexType>
       <complexContent>
           <extension base="cat:Pub" >
              <sequence>
                 <element name="ISBN" type="string" default="123456789"/>
                 <element name="Publisher" type="string"/>
              </sequence>
           </extension>
       </complexContent>
   </complexType>
</element>
<complexType name="titleType">
   <simpleContent>
      <extension base="string" >
         <attribute name="old" type="string" use="default" value="false"/>
      </extension>
   </simpleContent>
</complexType>
<element name="Magazine" substitutionGroup="cat:Publication">
    <complexType>
       <complexContent>
          <extension base="cat:Pub">
              <sequence>
                 <element name="Volume" type="cat:VolumeType"/>
                 <element name="htmlTable">
                    <complexType>
                       <any namespace="##other"
                            processContents="skip"
                            minOccurs="0" maxOccurs="2"/>
                    </complexType>
                  </element>
              </sequence>
          </extension>
       </complexContent>
    </complexType>
</element>
```

```
    <simpleType name="VolumeType">
       <restriction base="integer" >
          <minInclusive value = "1"/>
          <maxInclusive value = "12"/>
       </restriction>
    </simpleType>
    <element name="Catalogue">
        <complexType>
           <sequence>
             <element ref="cat:Publication" minOccurs="0" maxOccurs="*"/>
           </sequence>
        </complexType>
    </element>
</schema>
```

XML Schema for C++ の例 9: pub.xml

```
<?xml version="1.0"?>
<Catalogue xmlns = "http://www.somewhere.org/BookCatalogue"
   xmlns:cat = "http://www.somewhere.org/BookCatalogue"
   xmlns:html = "http://www.somewhere.org/HTMLCatalogue"
   xmlns:xsi = "http://www.w3.org/1999/XMLSchema-instance"
   xsi:schemaLocation =
             "http://www.somewhere.org/BookCatalogue pub.xsd">
        <cat:Magazine>
                <Title>Natural Health</Title>
                <Author>October</Author>
                <Date>1999-12</Date>
                <Volume>12</Volume>
                <htmlTable>
                   <table  xmlns = "http://www.somewhere.org/HTMLCatalogue">
                      <tr>....</tr>
                   </table>
                   <html:table>
                      <html:tr>....</html:tr>
                   </html:table>
                </htmlTable>
        </cat:Magazine>
        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN></ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Book>
```

```
        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp;amp; Row</Publisher>
        </Book>
</Catalogue>
```

XML Schema for C++ の例 10: pub.std

```
XML C++ Schema processor
Initializing XML package...
Parsing 'pub.xml'...
Initializing Schema package...
Validating document...
Document is valid.
```

28

XML C++ Class Generator の使用

この章の内容は次のとおりです。

- XML C++ Class Generator の入手方法
- XML C++ Class Generator の使用
- XML C++ Class Generator の使用方法
- xmlcg の使用方法
- sample/ にある XML C++ Class Generator のサンプル・ファイルの使用

XML C++ Class Generator の入手方法

XML C++ Class Generator は、Oracle に付属しています。OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

XML C++ Class Generator は、`$ORACLE_HOME/xdk/cpp/classgen` にあります。

XML C++ Class Generator の使用

XML C++ Class Generator は、XML DTD からソース・ファイルを作成します。XML C++ Class Generator は DTD を使用し、定義された各要素用のクラスを生成します。これらのクラスを C++ プログラムで使用し、DTD に準拠する XML 文書を作成します。

これは、アプリケーションが DTD に従って、または XML 文書を構成する Web フォームのバックエンドとして、他のアプリケーションに XML メッセージを送信する必要がある場合に有効です。C++ アプリケーションは、これらのクラスを使用して、入力用 DTD に準拠する XML 文書を構成、検証および出力できます。

XML C++ Class Generator は、Oracle XML Parser for C++ と連携して機能します。XML Parser for C++ は、DTD を解析し、解析済文書を Class Generator に渡します。

> **参照：** 詳細は、3-22 ページの「Oracle の XML コンポーネントを使用した XML 文書の生成 : C++」を参照してください。

外部 DTD の解析

XML C++ Class Generator は、完全なドキュメント（ダミー・ドキュメント）を要求することなく直接、外部 DTD を解析することもできます。これには、Oracle XML Parser for C++ ルーチンの `xmlparsedtd()` を使用します。

提供されているコマンドライン・プログラム `xmlcg` には、新しいオプション「-d」があります。これは、外部 DTD の解析に使用されます。詳細は、28-4 ページの「xmlcg の使用方法」を参照してください。

エラー・メッセージ・ファイル

エラー・メッセージ・ファイルは、`mesg/` サブディレクトリにあります。このメッセージ・ファイルは、`$ORACLE_HOME/oracore/mesg` ディレクトリにもあります。環境変数 `ORA_XML_MESG` を設定して、`mesg/` サブディレクトリへの絶対パスを指定することもできます。ただし、これは必須ではありません。

XML C++ Class Generator の使用方法

図 28-1 に、XML C++ Class Generator の使用方法の概要を示します。

1. bin ディレクトリから、コマンドラインに次のとおり入力します。

   ```
   xml [XML document file name, such as xxxxx]
   ```

   XML 文書ファイル名は、処理される解析済 XML 文書または DTD の名前です。XML 文書には、DTD が対応付けられている必要があります。

   XML C++ Class Generator への入力は、DTD を含む XML 文書、または外部 DTD です。文書本体自体は無視され、DTD のみが使用されますが、文書は DTD に準拠している必要があります。

   入力として使用できるキャラクタ・セット・エンコーディングについては、付録 F「XDK for C++: 仕様および早見表」を参照してください。

2. 2 つのソース・ファイル、xxxxx.h ヘッダー・ファイルおよび xxxxx.cpp C++ ファイルが出力されます。これらのファイルには、DTD ファイルと同じ名前が付けられます。

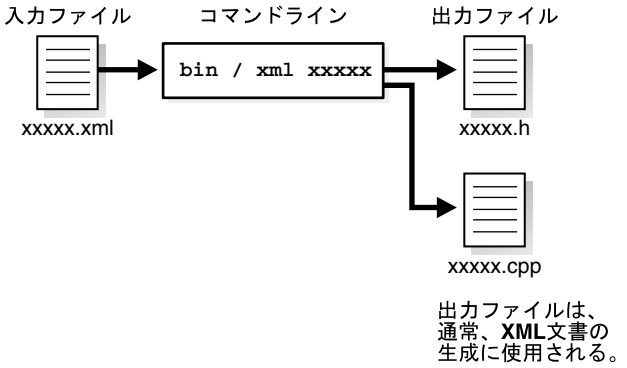
3. 出力ファイルは、通常、XML 文書の生成に使用されます。

各クラス（要素）にコンストラクタが提供されているため、次の 2 つの方法でオブジェクトを作成できます。

- まず空のオブジェクトを作成し、後で子またはデータを追加します。
- 最初からすべての子または初期データを含むオブジェクトを作成します。

#PCDATA（および複合）要素には、データおよび適切な場合は要素の属性を設定するためのメソッドが提供されています。

図 28-1 XML C++ Class Generator の機能



xmlcg の使用方法

スタンドアロンのパーサーは、bin/xmlcg をコールすることによって、実行可能ファイルとしてコールできます。次に例を示します。

```
xmlcg [flags] <XML document or External DTD>
```

表 28-1 に、xmlcg オプション・フラグを示します。

表 28-1 xmlcg オプション・フラグ

xmlcg オプション・フラグ	説明
-d name	DTD - 指定された名前を持つ外部 DTD を入力します。
-o directory	生成されたファイル用の出力ディレクトリ - デフォルトは、現在のディレクトリです。
-e encoding	エンコーディング - デフォルトの入力ファイルのエンコーディングを指定します。
-h	ヘルプ - 使用方法のヘルプを表示します。
-v	バージョン - XML C++ Class Generator のバージョンを表示します。

sample/ にある XML C++ Class Generator のサンプル・ファイルの使用

表 28-2 に、sample/ ディレクトリにある XML C++ Class Generator のサンプル・ファイルを示します。

表 28-2 sample/ ディレクトリにある XML C++ Class Generator のサンプル・ファイル

サンプル・ファイル名	説明
CG.cpp	サンプル・プログラム
CG.xml	DTD およびダミー・ドキュメントを含む XML ファイル
CG.dtd	CG.xml が参照する DTD ファイル
Make.bat（Windows NT） Makefile（UNIX）	クラスを生成し、サンプル・プログラムを構築する、バッチ・ファイル（Windows NT の場合）またはスクリプト・ファイル（UNIX の場合）
README	前述のファイルの説明を含む README ファイル

make.bat バッチ・ファイル（Windows NT の場合）または Makefile（UNIX の場合）では、次の操作が実行できます。

- CG.xml に基づいて、クラスを Sample.h および Sample.cpp に生成します。
- （Sample.h を使用して）プログラム CG.cpp をコンパイルし、サンプル・オブジェクトとともに ..¥bin（または ../bin）ディレクトリにある CG.exe という実行可能ファイルにリンクします。

XML C++ Class Generator の例 1: XML - Class Generator の入力ファイル CG.xml

XML ファイル CG.xml は、XML C++ Class Generator を入力します。CG.xml は、DTD ファイル CG.dtd を参照します。

```
<?xml version="1.0"?>
<!DOCTYPE Sample SYSTEM "CG.dtd">
  <Sample>
    <B>Be!</B>
    <D attr="value"></D>
    <E>
      <F>Formula1</F>
      <F>Formula2</F>
    </E>
  </Sample>
```

XML C++ Class Generator の例 2: DTD - Class Generator の入力ファイル CG.dtd

DTD ファイル CG.dtd は、XML ファイル CG.xml によって参照されます。CG.xml は、XML C++ Class Generator を入力します。

```
<!ELEMENT Sample (A | (B, (C | (D, E))) | F)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA | F)*>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ATTLIST D attr CDATA #REQUIRED>
<!ELEMENT E (F, F)>
<!ELEMENT F (#PCDATA)>
```

XML C++ Class Generator の例 3: CG サンプル・プログラム

CG サンプル・プログラム CG.cpp は、次のとおり実行します。

1. XML パーサーを初期化します。
2. DTD を含むファイルのダミー・ドキュメント部分以外を解析することによって、DTD をロードします。
3. 生成されたクラスを使用して複数のオブジェクトを作成します。
4. 生成されたクラスが DTD に一致するかどうかを検証する、検証関数をコールします。
5. 作成されたドキュメントを Sample.xml に書き込みます。

```
////////////////////////////////////////////////////////////////////////
// NAME        CG.cpp
// DESCRIPTION Demonstration program for C++ Class Generator usage
////////////////////////////////////////////////////////////////////////

#ifndef ORAXMLDOM_ORACLE
# include <oraxmldom.h>
#endif

#include <fstream.h>

#include "Sample.h"

#define DTD_DOCUMENT"CG.xml"
#define OUT_DOCUMENT"Sample.xml"
```

```
int main()
{
    XMLParser parser;
    Document *doc;
    Sample   *samp;
    B        *b;
    D        *d;
    E        *e;
    F        *f1, *f2;
    fstream  *out;
    ub4       flags = XML_FLAG_VALIDATE;
    uword     ecode;

    // Initialize XML parser
    cout << "Initializing XML parser...\n";
    if (ecode = parser.xmlinit())
    {
cout << "Failed to initialize parser, code " << ecode << "\n";
        return 1;
    }

    // Parse the document containing a DTD; parsing just a DTD is not
    // possible yet, so the file must contain a valid document (which
    // is parsed but we're ignoring).
    cout << "Loading DTD from " << DTD_DOCUMENT << "...\n";
    if (ecode = parser.xmlparse((oratext *) DTD_DOCUMENT, (oratext *)0, flags))
    {
cout << "Failed to parse DTD document " << DTD_DOCUMENT <<
    ", code " << ecode << "\n";
return 2;
    }

    // Fetch dummy document
    cout << "Fetching dummy document...\n";
    doc = parser.getDocument();

    // Create the constituent parts of a Sample
    cout << "Creating components...\n";
    b = new B(doc, (String) "Be there or be square");
    d = new D(doc, (String) "Dit dah");
    d->setattr((String) "attribute value");
    f1 = new F(doc, (String) "Formula1");
    f2 = new F(doc, (String) "Formula2");
    e = new E(doc, f1, f2);
```

```
    // Create the Sample
    cout << "Creating top-level element...\n";
    samp = new Sample(doc, b, d, e);

    // Validate the construct
    cout << "Validating...\n";
    if (ecode = parser.validate(samp))
    {
cout << "Validation failed, code " << ecode << "\n";
return 3;
    }

    // Write out doc
    cout << "Writing document to " << OUT_DOCUMENT << "\n";
    if (!(out = new fstream(OUT_DOCUMENT, ios::out)))
    {
cout << "Failed to open output stream\n";
return 4;
    }
    samp->print(out, 0);
    out->close();

    // Everything's OK
    cout << "Success.\n";

    // Shut down
    parser.xmlterm();
    return 0;
}

// end of CG.cpp
```

第 X 部

XDK for PL/SQL

第 X 部では、XDK for PL/SQL の入手方法および使用方法を説明します。第 X 部に含まれる章は、次のとおりです。

- 第 29 章「XML Parser for PL/SQL の使用」

> **注意：** XSU for PL/SQL は XDK for PL/SQL の一部です。XSU の詳細は、第 7 章「XSU」を参照してください。

29-19 ページの「FAQ: XML Parser for PL/SQL」に FAQ が含まれます。

29

XML Parser for PL/SQL の使用

この章の内容は次のとおりです。

XML Parser for PL/SQL の入手方法

XML Parser for PL/SQL は Oracle に付属しています。また、OTN の Web サイト http://otn.oracle.com/tech/xml からダウンロードすることもできます。

XML Parser for PL/SQL は、`$ORACLE_HOME/xdk/plsql/parser` にあります。

XML Parser for PL/SQL の実行の要件

付録 G「XDK for PL/SQL: 仕様および早見表」に、XML Parser for PL/SQL を実行するための仕様および要件をリストします。ここには、構文の早見表も含まれています。

XML Parser for PL/SQL の使用（DOM インタフェース）

XML Parser for PL/SQL を使用すると、Oracle を使用して、簡略化および標準化されたプロセスで XML アプリケーションを開発できます。PL/SQL インタフェースを使用すると、PL/SQL を十分に理解している Oracle の開発者は既存のアプリケーションを拡張し、必要に応じて XML を利用できます。

XML Parser for PL/SQL は PL/SQL および Java で実装されているため、Oracle JVM で自由に実行できます。

XML Parser for PL/SQL は、W3C の XML 1.0 仕様をサポートしています。この仕様に 100% 準拠することを目標としています。XML Parser for PL/SQL は、妥当性を検証するパーサーまたは検証しないパーサーとして使用できます。

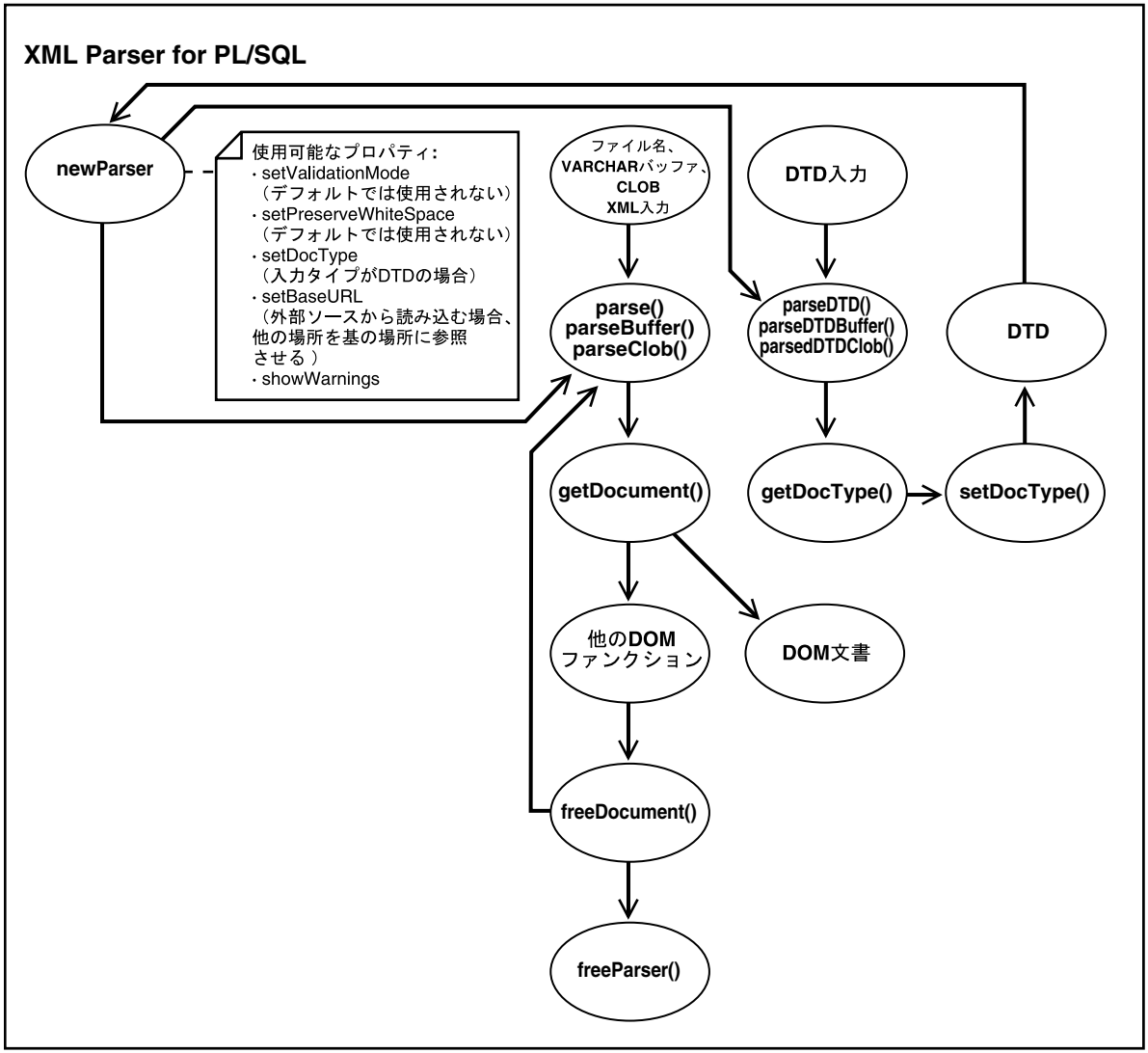
さらに、XML Parser for PL/SQL は、開発者が XML 文書を処理するために必要な、最も一般的な次の 2 つの API を提供します。

- W3C 勧告の DOM
- W3C 勧告の XSLT および XPath

これによって、Oracle 環境で、XML 文書を処理するカスタム・アプリケーションを簡単に作成できます。これは、Oracle がインストールされたすべてのオペレーティング・システムでは、標準に準拠した XML パーサーが Oracle のプラットフォームの一部として搭載されていることを示します。

図 29-1 に、XML Parser for PL/SQL の使用方法および解析プロセスを示します。

図 29-1 XML Parser for PL/SQL の機能（DOM インタフェース）



1. 必要に応じて newParser 宣言を行い、XML 文書および DTD の解析プロセスを開始します。

   表 29-1 に、newParser プロシージャに使用可能なプロパティを示します。

表 29-1 XML Parser for PL/SQL: newParser() のプロパティ

プロパティ	説明
setValidationMode	デフォルトでは使用されません。
setPreserveWhiteSpace	デフォルトでは使用されません。
setDocType	入力タイプが DTD の場合に使用します。
setBaseURL	外部ソースから読み込む場合、基の場所に他の場所を参照させます。
showWarnings	警告のオンまたはオフを切り替えます。

2. XML および DTD は、ファイル、VARCHAR バッファまたは CLOB として入力できます。XML 入力は、次のプロシージャによってコールされます。
   - parse()（XML 入力がファイルの場合）
   - parseBuffer()（XML 入力が VARCHAR バッファの場合）
   - parserClob()（XML 入力が CLOB の場合）

   DTD も入力される場合、次のプロシージャによってコールされます。
   - parseDTD()（入力が DTD ファイルの場合）
   - parseDTDBuffer()（DTD 入力が VARCHAR バッファの場合）
   - parserDTDClob()（DTD 入力が CLOB の場合）

XML 入力の場合 : XML 入力の場合、Parse()、ParserBuffer() または ParserClob() プロシージャからの解析結果は、GetDocument() へ送られます。

3. getDocument() プロシージャは、次の操作を実行します。
   - 解析済 XML 文書を、PL/SQL アプリケーションで通常使用される DOM 文書として出力します。
   - 必要に応じて、他の DOM ファンクションを適用します。

     参照 : 使用可能なオプションの DOM ファンクションのリストは、『Oracle9*i* XML リファレンス』を参照してください。

4. freeDocument() ファンクションを使用してパーサーを解放し、次の XML 入力を解析します。
5. freeParser() ファンクションを使用して、解析プロセス中に作成された一時文書構造を解除します。

DTD 入力の場合 : parseDTD()、parseDTDBuffer() または parseDTDClob() からの解析結果は、getDocType() ファンクションが使用します。

6. getDocType() は、setDocType() を使用して DTD オブジェクトを生成します。
7. setDocType() を使用して DTD オブジェクトをパーサーに送り、対応付けられた DTD をオーバーライドできます。

XML Parser for PL/SQL: デフォルト動作

XML Parser for PL/SQL のデフォルト動作は、次のとおりです。

- DOM API によってアクセス可能な解析済ツリーが構築されます。
- DTD が検出された場合、XML Parser for PL/SQL は妥当性の検証を行います。検出されない場合は、検証を行わないパーサーになります。
- エラー・ログが指定されないかぎり、エラーは記録されません。ただし、解析が正常に実行されない場合、アプリケーション・エラーが発生します。

このマニュアルに記載されているタイプおよびメソッドは、PL/SQL パッケージ xmlparser() に付属しています。

XML Parser for PL/SQL の使用 : XSLT プロセッサ（DOM インタフェース）

XSLT は、ソース・ツリーを結果ツリーへ変換する規則を表します。XSLT で表現される変換を、スタイルシートといいます。

指定された変換は、スタイルシートに定義したテンプレートとパターンを対応付けることによって実行されます。結果ツリーの一部を作成するために、テンプレートはインスタンス化されます。

XSL プロセッサの PL/SQL 実装は、W3C の XSLT 草案（WD-xslt-19990813 改訂）に準拠し、XSLT スタイルシートの読込み方法および影響する変換の点で、XSL プロセッサに必要な動作を含みます。

このマニュアルに記載されているタイプおよびメソッドは、PL/SQL パッケージ xslprocessor() に付属しています。

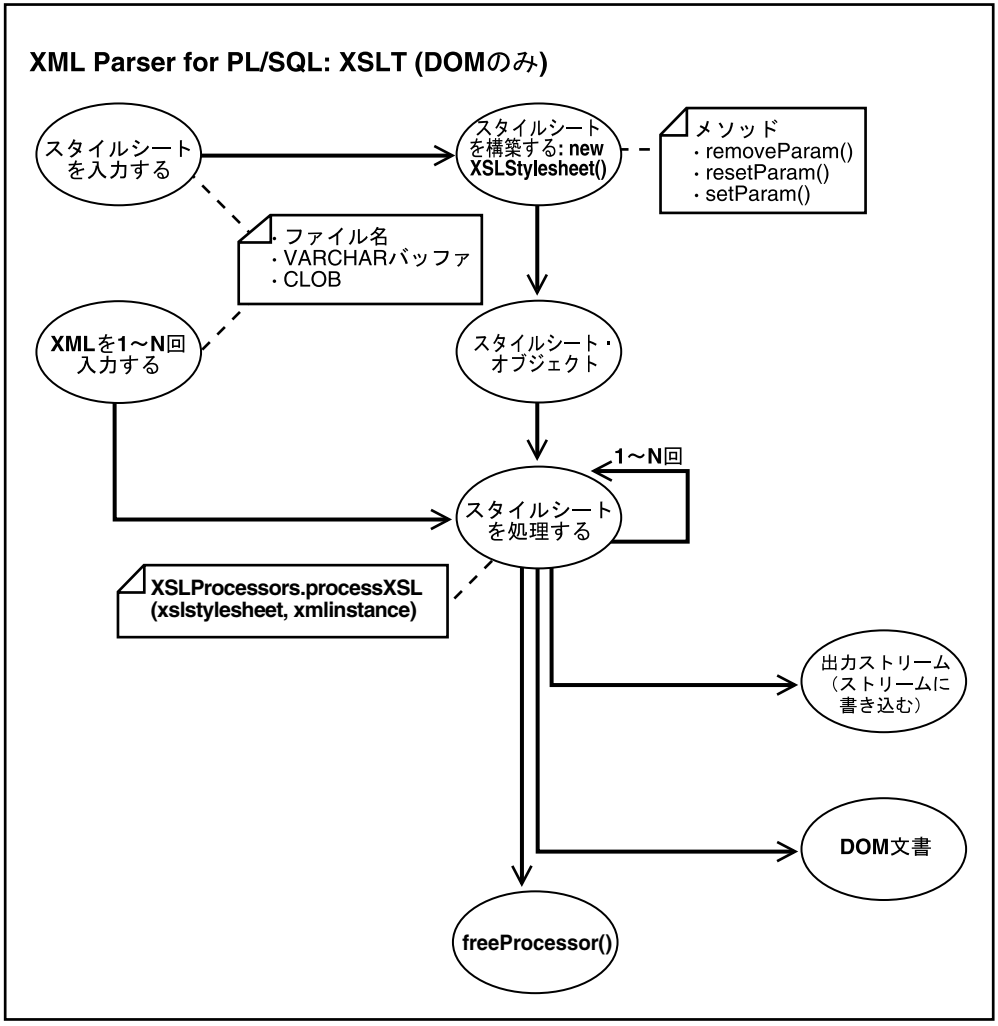
図 29-2 に、XML Parser for PL/SQL の XSLT プロセッサの主な機能を示します。

1. 「スタイルシートを処理する」のプロセスでは、XML 文書および選択されたスタイルシート（XML 文書に示されるかどうかにかかわらず）からの入力を受け取ります。スタイルシートおよび XML 文書は、次のいずれかになります。
   - ファイル名
   - VARCHAR バッファ
   - CLOB

   XML 文書は、1 ～ N 回入力できます。
2. 解析済 XML 文書は、`XSLProcessor.processXSL(xslstylesheet,xmlinstance)` プロシージャを入力します。ここで、引数は次のとおりです。
   - 引数「xmlinstance」には、XML 文書が示されます。
   - 引数「xslstylesheet」には、スタイルシートの入力が示されます。
3. `XSLStylesheet()` プロシージャへのスタイルシート入力を使用して、スタイルシートを構築します。このプロシージャに使用できるメソッドは次のとおりです。
   - `removeParam()`
   - `resetParam()`
   - `setParam()`

   これでスタイルシート・オブジェクトが作成され、`XSLProcessor.processXSL(xslstylesheet,xmlinstance)` プロシージャを使用して、「スタイルシートを処理する」手順が開始します。
4. 「スタイルシートを処理する」手順は、1 ～ N 回繰り返すことができます。同じスタイルシートを複数の解析済 XML 文書に適用して、XML 文書、HTML ドキュメントまたは他のテキストベース形式のいずれかに変換できます。
5. 結果の解析および変換済の文書は、ストリームまたは DOM 文書として出力されます。
6. XSLT プロセスが完了すると、freeProcessor() プロシージャをコールして、XSL 変換プロセスに使用された一時構造および XSLProcessor プロシージャを解除します。

図 29-2 XML Parser for PL/SQL: XSLT プロセッサ（DOM インタフェース）



XML Parser for PL/SQL: XSLT プロセッサ - デフォルト動作

XML Parser for PL/SQL の XSLT プロセッサのデフォルト動作は、次のとおりです。

- DOM API がアクセスできる結果ツリーが構築されます。
- エラー・ログが指定されないかぎり、エラーは記録されません。ただし、解析が正常に実行されない場合、アプリケーション・エラーが発生します。

sample/ での XML Parser for PL/SQL の使用例

sample/ サンプル・プログラムの動作環境の設定

$ORACLE_HOME/xdk/plsql/parser/sample/ ディレクトリには、2 つのサンプル XML アプリケーションがあります。

- domsample
- xslsample

これらは、XML Parser for PL/SQL の使用方法を説明します。

サンプル・プログラムを実行するには、次の手順に従います。

1. データベースに XML Parser for PL/SQL をロードします。ロードするには、lib ディレクトリにある README ファイルの指示に従います。
2. ファイル・システムのファイルから読込みおよび書込みを行うには、適切な Java セキュリティ権限が必要です。このためには、まず SQL*Plus（通常 $ORACLE_HOME/bin にあります）を起動し、「internal」などの管理権限を持つユーザーとして接続します。

   次に例を示します。

   ```
   % sqlplus
   SQL> connect / as sysdba
   ```

3. 「internal」または管理権限を持つ適切なユーザーのパスワードが必要です。管理権限を持つユーザーとしてログインできない場合、システム管理者、データベース管理者またはオラクル社カスタマ・サポート・センターに連絡してください。
4. このサンプルを実行するユーザーに、特別な権限を付与します。この権限は、手順 1 で jar ファイルおよび plsql ファイルをロードしたときのものと同じにする必要があります。

   ユーザー「scott」の場合 :

   ```
   SQL> grant javauserpriv to scott;
   SQL> grant javasyspriv to scott;
   ```

「Grant succeeded.」というメッセージが 2 つ表示されます。メッセージが表示されない場合、システム管理者、データベース管理者またはオラクル社カスタマ・サポート・センターに連絡してください。

手順 1 で XML Parser for PL/SQL をロードしたユーザーとして、再度接続します。たとえば、ユーザー「scott」（パスワードは「tiger」）で接続します。

```
SQL> connect scott/tiger
```

domsample の実行

domsample を実行するには、次の手順に従います。

1. 次のとおり、SQL*Plus で domsample.sql スクリプトをロードします（SQL*Plus が起動されていない場合、起動して、このサンプルを実行するユーザーとして接続します）。

   ```
   SQL> @domsample
   ```

   domsample.sql スクリプトは、次の構文で domsample プロシージャを定義します。

   ```
   domsample(dir varchar2, inpfile varchar2, errfile varchar2)
   ```

   各引数の意味は、次のとおりです。

引数	説明
dir	外部ファイル・システム上の有効なディレクトリを指す必要があります。この引数は、完全なパス名で指定する必要があります。
inpfile	dir にあるファイルを指し、解析対象の XML 文書を含む必要があります。
errfile	エラー記録用に使用するファイルを指す必要があります。このファイルは dir 内に作成されます。

2. 適切な引数 dir、inpfile および errfile を指定して、SQL*Plus 内で domsample プロシージャを実行します。次に例を示します。

   UNIX では、次のとおり実行できます。

   ```
   SQL>execute domsample('/private/scott', 'family.xml', 'errors.txt');
   ```

   Windows NT では、次のとおり実行できます。

   ```
   SQL>execute domsample('c:¥xml¥sample', 'family.xml', 'errors.txt');
   ```

   family.xml はテスト用に提供されています。

3. 次の出力が表示されます。

   - 要素は、family member member member member です。
   - 各要素の属性は次のとおりです。

```
family:
lastname = Smith
  member:
  memberid = m1
  member:
  memberid = m2
  member:
  memberid = m3 mom = m1 dad = m2
  member:
  memberid = m4 mom = m1 dad = m2
```

xslsample の実行

xslsample を実行するには、次の手順に従います。

1. 次のとおり、SQL*Plus で xslsample.sql スクリプトをロードします（SQL*Plus が起動されていない場合、起動して、このサンプルを実行するユーザーとして接続します）。

```
SQL>@xslsample
```

   xslsample.sql スクリプトは、次の構文で xslsample プロシージャを定義します。

```
xslsample ( dir varchar2, xmlfile varchar2, xslfile varchar2, resfile varchar2,
errfile varchar2 )
```

   各引数の意味は、次のとおりです。

引数	説明
dir	外部ファイル・システム上の有効なディレクトリを指す必要があります。この引数は、完全なパス名で指定する必要があります。
xmlfile	dir にあるファイルを指し、解析対象の XML 文書を含む必要があります。
xskfile	dir にあるファイルを指し、適用する XSL スタイルシートを含む必要があります。
resfile	変換済文書を置く dir 内のファイルを指す必要があります。
errfile	エラー記録用に使用するファイルを指す必要があります。このファイルは dir 内に作成されます。

2. 適切な引数 dir、xmlfile、xslfile および errfile を指定して、SQL*Plus 内で xslsample プロシージャを実行します。

   次に例を示します。

   - UNIX では、次のとおり実行できます。

     ```
     SQL>execute xslsample('/private/scott', 'family.xml', 'iden.xsl',
     'family.out', 'errors.txt');
     ```

   - Windows NT では、次のとおり実行できます。

     ```
     SQL>execute xslsample('c:¥xml¥sample', 'family.xml', 'iden.xsl',
     'family.out', 'errors.txt');
     ```

3. family.xml および iden.xsl はテスト用に提供されています。

4. 次の出力が表示されます。

   ```
   Parsing XML document c:¥xml¥sample¥family.xml
   Parsing XSL document c:¥xml¥sample¥iden.xsl
   XSL Root element information
   Qualified Name: xsl:stylesheet
   Local Name: stylesheet
   Namespace: http://www.w3.org/XSL/Transform/1.0
   Expanded Name: http://www.w3.org/XSL/Transform/1.0:stylesheet
   A total of 1 XSL instructions were found in the stylesheet
   Processing XSL stylesheet
   Writing transformed document
   ```

5. family.out の内容は次のとおりです。

   ```
   <family lastname="Smith">
   <member memberid="m1">Sarah</member>
   <member memberid="m2">Bob</member>
   <member memberid="m3" mom="m1" dad="m2">Joanne</member>
   <member memberid="m4" mom="m1" dad="m2">Jim</member>
   </family>
   ```

   このプロシージャを初めて実行したとき、出力の取得に時間がかかる場合があります。これは、Oracle JVM が様々な初期化タスクを行ってから、Java ストアド・プロシージャ（JSP）を実行するためです。次回からは、速く起動できます。

   エラーが発生する場合、ディレクトリ名がファイル・システム上の完全なパスで指定されていることを確認してください。

   注意： 現在、SQL ディレクトリの別名および共有ディレクトリ構文「¥¥」はサポートされていません。

エラーを解決できない場合、その問題点を http://otn.oracle.com/ の XML Discussion Forum に報告してください。

XML Parser for PL/SQL の例 1: XML - family.xml

family.xml は、domsample.sql を入力します。

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
<member memberid="m1">Sarah</member>
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

XML Parser for PL/SQL の例 2: DTD - family.dtd

family.dtd は、family.xml で参照されます。

```
<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>
```

XML Parser for PL/SQL の例 3: XSL - iden.xsl

iden.xsl は、xslsample.sql を入力します。

```
<?xml version="1.0"?>

<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="*|@*|comment()|processing-instruction()|text()">
  <xsl:copy>
  <xsl:apply-templates select="*|@*|comment()|processing-instruction()|text()"/>
  </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

XML Parser for PL/SQL の例 4: PL/SQL - domsample.sql

```
-- This file demonstrates a simple use of the parser and DOM API.
-- The XML file that is given to the application is parsed and the
-- elements and attributes in the document are printed.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure domsample(dir varchar2, inpfile varchar2,
                                      errfile varchar2) is
p xmlparser.parser;
doc xmldom.DOMDocument;

-- prints elements in a document
procedure printElements(doc xmldom.DOMDocument) is
nl xmldom.DOMNodeList;
len number;
n xmldom.DOMNode;

begin
   -- get all elements
   nl := xmldom.getElementsByTagName(doc, '*');
   len := xmldom.getLength(nl);

   -- loop through elements
   for i in 0..len-1 loop
      n := xmldom.item(nl, i);
      dbms_output.put(xmldom.getNodeName(n) || ' ');
   end loop;

   dbms_output.put_line('');
end printElements;

-- prints the attributes of each element in a document
procedure printElementAttributes(doc xmldom.DOMDocument) is
nl xmldom.DOMNodeList;
len1 number;
len2 number;
n xmldom.DOMNode;
e xmldom.DOMElement;
nnm xmldom.DOMNamedNodeMap;
attrname varchar2(100);
attrval varchar2(100);
```

```
begin

   -- get all elements
   nl := xmldom.getElementsByTagName(doc, '*');
   len1 := xmldom.getLength(nl);

   -- loop through elements
   for j in 0..len1-1 loop
      n := xmldom.item(nl, j);
      e := xmldom.makeElement(n);
      dbms_output.put_line(xmldom.getTagName(e) || ':');

      -- get all attributes of element
      nnm := xmldom.getAttributes(n);

     if (xmldom.isNull(nnm) = FALSE) then
        len2 := xmldom.getLength(nnm);

        -- loop through attributes
        for i in 0..len2-1 loop
           n := xmldom.item(nnm, i);
           attrname := xmldom.getNodeName(n);
           attrval := xmldom.getNodeValue(n);
           dbms_output.put(' ' || attrname || ' = ' || attrval);
        end loop;
        dbms_output.put_line('');
     end if;
   end loop;

end printElementAttributes;

begin

-- new parser
   p := xmlparser.newParser;

-- set some characteristics
   xmlparser.setValidationMode(p, FALSE);
   xmlparser.setErrorLog(p, dir || '/' || errfile);
   xmlparser.setBaseDir(p, dir);

-- parse input file
   xmlparser.parse(p, dir || '/' || inpfile);

-- get document
   doc := xmlparser.getDocument(p);
```

```
-- Print document elements
   dbms_output.put('The elements are: ');
   printElements(doc);

-- Print document element attributes
   dbms_output.put_line('The attributes of each element are: ');
   printElementAttributes(doc);

-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
   raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
   raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
   raise_application_error(-20120, 'Hierarchy request error');

when xmldom.WRONG_DOCUMENT_ERR then
   raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
   raise_application_error(-20120, 'Invalid Char error');

when xmldom.NO_DATA_ALLOWED_ERR then
   raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
   raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
   raise_application_error(-20120, 'Not found error');

when xmldom.NOT_SUPPORTED_ERR then
   raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
   raise_application_error(-20120, 'In use attr error');

end domsample;
/
show errors;
```

XML Parser for PL/SQL の例 5: PL/SQL - xslsample.sql

```
-- This file demonstates a simple use of XSL-T transformation capabilities.
-- The XML and XSL files that are given to the application are parsed,
-- the transformation specified is applied and the transformed document is
-- written to a specified result file.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure xslsample(dir varchar2, xmlfile varchar2,
                                      xslfile varchar2, resfile varchar2,
                                      errfile varchar2) is
p xmlparser.Parser;
xmldoc xmldom.DOMDocument;
xmldocnode xmldom.DOMNode;
proc xslprocessor.Processor;
ss xslprocessor.Stylesheet;
xsldoc xmldom.DOMDocument;
docfrag xmldom.DOMDocumentFragment;
docfragnode xmldom.DOMNode;
xslelem xmldom.DOMElement;
nspace varchar2(50);
xslcmds xmldom.DOMNodeList;

begin

-- new parser
   p := xmlparser.newParser;

-- set some characteristics
   xmlparser.setValidationMode(p, FALSE);
   xmlparser.setErrorLog(p, dir || '/' || errfile);
   xmlparser.setPreserveWhiteSpace(p, TRUE);
   xmlparser.setBaseDir(p, dir);

-- parse xml file
   dbms_output.put_line('Parsing XML document ' || dir || '/' || xmlfile);
   xmlparser.parse(p, dir || '/' || xmlfile);

-- get document
   xmldoc := xmlparser.getDocument(p);

-- parse xsl file
   dbms_output.put_line('Parsing XSL document ' || dir || '/' || xslfile);
   xmlparser.parse(p, dir || '/' || xslfile);
```

```
-- get document
   xsldoc := xmlparser.getDocument(p);

   xslelem := xmldom.getDocumentElement(xsldoc);
   nspace := xmldom.getNamespace(xslelem);

-- print out some information about the stylesheet
   dbms_output.put_line('XSL Root element information');
   dbms_output.put_line('Qualified Name: ' ||
                        xmldom.getQualifiedName(xslelem));
   dbms_output.put_line('Local Name: ' ||
                        xmldom.getLocalName(xslelem));
   dbms_output.put_line('Namespace: ' || nspace);
   dbms_output.put_line('Expanded Name: ' ||
                        xmldom.getExpandedName(xslelem));

   xslcmds := xmldom.getChildrenByTagName(xslelem, '*', nspace);
   dbms_output.put_line('A total of ' || xmldom.getLength(xslcmds) ||
                        ' XSL instructions were found in the stylesheet');
-- make stylesheet
   ss := xslprocessor.newStylesheet(xsldoc, dir || '/' || xslfile);

-- process xsl
   proc := xslprocessor.newProcessor;
   xslprocessor.showWarnings(proc, true);
   xslprocessor.setErrorLog(proc, dir || '/' || errfile);

   dbms_output.put_line('Processing XSL stylesheet');
   docfrag := xslprocessor.processXSL(proc, ss, xmldoc);
   docfragnode := xmldom.makeNode(docfrag);

   dbms_output.put_line('Writing transformed document');
  xmldom.writeToFile(docfragnode, dir || '/' || resfile);

-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
   raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
   raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
   raise_application_error(-20120, 'Hierarchy request error');
```

```
when xmldom.WRONG_DOCUMENT_ERR then
   raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
   raise_application_error(-20120, 'Invalid Char error');

when xmldom.NO_DATA_ALLOWED_ERR then
   raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
   raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
   raise_application_error(-20120, 'Not found error');

when xmldom.NOT_SUPPORTED_ERR then
   raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
   raise_application_error(-20120, 'In use attr error');

end xslsample;
/
show errors;
```

FAQ: XML Parser for PL/SQL

スレッド・パーサー・エラーの例外

質問

`oraxsl` を使用しようとすると、次のとおり「main」スレッドの例外が発生します。

```
java.lang.NoClassDefFoundError" oracle/xml/parser/v2/oraxsl.
```

対処方法を教えてください。

回答

構成および使用方法について、詳しく教えてください。データベース外で実行中の場合、xmlparserv2.jar が単にディレクトリではなく、明示的に CLASS_PATH 内にあることを確認する必要があります。データベース内で実行中の場合、xmlparserv2.jar が適切にロードされ、Oracle JVM が初期化されていることを確認する必要があります。

Java VM が現在サポートしないエンコーディング「8859_1」

質問

XML Parser for PL/SQL を使用して XML 文書を解析し、setNodeValue を使用して DOM 文書のいくつかのノード値を変更しました。「writeToBuffer」または「dataToFile」を使用して、変更した DOM 文書をバッファまたはファイルに書き込もうとした場合、両方のコマンドでエラーが表示されました。

コメント

initjvm.sql を再インストールして、XML Parser for PL/SQL の最新バージョンもインストールしました。すべて正常に動作しています。

PL/SQL での xmldom.GetNodeValue

質問

PL/SQL XMLDOM を使用して、要素値を取得できません。次に、コードのフラグメントを示します。

```
...nl := xmldom.getElementsByTagName(doc, '*');
len := xmldom.getLength(nl)
;-- loop through elements
   for i in 0..len-1 loop      n := xmldom.item(nl, i);
     elename := xmldom.getNodeName(n);
eleval := xmldom.getNodeValue(n);
...elename is Ok, but eleval is NULL.
```

テキスト・ノードとの対応付けが正常に実行できないようですが、方法が間違っているのでしょうか？次のようなコンパイル・エラーが発生します。

```
...t xmldom.DOMText;
...t := xmldom.makeText(n);
eleval := xmldom.getNodeValue(t);
```

何が間違っているのでしょうか？

コメント

問題点がわかりました。要素ノードに対応付けられたテキスト・ノード値を取得するには、xmldom.getFirstChild(n) を介して新しくノードのナビゲーションを実行する必要があります。

参考に、次のとおり DOMSample.sql の printElements() を変更します。

```
begin
-- get all elements
nl := xmldom.getElementsByTagName(doc, '*');
 len := xmldom.getLength(nl);
  -- loop through elements
for i in 0..len-1 loop      n := xmldom.item(nl, i);
      dbms_output.put(xmldom.getNodeName(n));
      -- get the text node associated with the element node
      n := xmldom.getFirstChild(n);
     if xmldom.getNodeType(n) = xmldom.TEXT_NODE then         dbms_output.put('='
&#0124; &#0124; xmldom.getNodeValue(n));
      end if;
     dbms_output.put(' ');
   end loop;
   dbms_output.put_line('');
end printElements;
```

この結果、次の出力が生成されます。

要素は次のとおりです。

```
family member=Sarah member=Bob member=Joanne member=Jim
```

各要素の属性は次のとおりです。

```
family:familylastname val=Smithmember:membermemberid val=m1member:membermemberid
val=m2member:membermemberid val=m3 mom val=m1 dad val=m2member:membermemberid val=m4
mom val=m1 dad val=m2
```

CLOB（PL/SQL）XML に含まれる DTD の解析

質問

CLOB に含まれる DTD ファイルの解析が正常に実行できません。XML Parser for PL/SQL によって提供される API「xmlparser.parseDTDClob」を使用しています。

次のエラーが発生しています。

```
ORA-29531: メソッド parseDTD はクラス oracle/xml/parser/plsql/XMLParserCover にはありません。
```

xmlparser.parseDTDClob プロシージャは、Java ストアド・プロシージャ xmlparsercover.parseDTDClob をコールし、このストアド・プロシージャが別の Java ストアド・プロシージャ xmlparsercover.parseDTD をコールします。

クラス・ファイル「oracle.xml.parser.plsql.XMLParserCover」が、データベースにロードされ、公開されていることは確認済です。そのため、エラー・メッセージの意味が理解できません。「xmlparser.parseDTDClob」をコールするために使用したプロシージャは、次のとおりです。

```
create or replace procedure parse_my_dtd as
p xmlparser.parser;
l_clob clob;
begin   p := xmlparser.newParser;
  select content into l_clob from dca_documents where doc_id = 1;
  xmlparser.parseDTDClob(p,l_clob,'site_template');
end;
```

xmlparser.parseDTDClob の API ドキュメントは次のとおりです。

```
parseDTDClob
PURPOSE
  Parses the DTD stored in the given clob
SYNTAX
  PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2);
PARAMETERS
  p        (IN)-  parser instance
  dtd      (IN)-  dtd clob to parse
  root     (IN)-  name of the root element
RETURNS
  Nothing
COMMENTS
```

パーサーのデフォルト動作への変更が有効になってから、このプロシージャをコールします。なんらかの理由で解析が正常に実行されない場合、アプリケーション・エラーが発生します。

dca_documents 表の内容は次のとおりです。

```
 DOC_ID          NOT NULL   NUMBER
 DOC_NAME        NOT NULL   VARCHAR2(350)
 DOC_TYPE                   VARCHAR2(30)
 DESCRIPTION                VARCHAR2(4000)
 MIME_TYPE                  VARCHAR2(48)
 CONTENT         NOT NULL   CLOB
 CREATED_BY      NOT NULL   VARCHAR2(30)
 CREATED_ON      NOT NULL   DATE
 UPDATED_BY      NOT NULL   VARCHAR2(30)
 UPDATED_ON      NOT NULL   DATE
```

DTD の内容は次のとおりです。

```
<!ELEMENT site_template (component*)>
<!ATTLIST site_template template_id CDATA #REQUIRED>
<!ATTLIST site_template template_name CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component component_id ID #REQUIRED>
<!ATTLIST component parent_id ID #REQUIRED>
<!ATTLIST component component_name ID #REQUIRED>
```

回答

これは、XML Parser for PL/SQL リリース 1.0.1 で検出された問題です。この問題を解決するには、次の手順に従います。

1. ./plsqlxmlparser_1.0.1/lib/sql/xmlparsercover.sql のバックアップを取ります。
2. xmlparsercover.sql の 18 行目にある、oracle.xml.parser.plsql.XMLParserCover.parseDTD を oracle.xml.parser.plsql.XMLParserCover.parseDTDClob に変更します。
3. 18 行目が、次のようになっているかどうかを確認します。
   procedure parseDTDClob(id varchar2, DTD CLOB, root varchar2, err in out varchar2) is language java
   name'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob(java.lang.String, oracle.sql.CLOB, java.lang.String, java.lang.String[])'
4. ファイルを保存します。
5. SQL*Plus で xmlparsercover.sql を再実行します。Oracle XML Parser バージョン 2.0.2.6 以上がデータベースにロードされていると想定すると、これで問題が解決するはずです。

XML Parser for PL/SQL

質問

XML Parser for PL/SQL を使用し始めたばかりです。開始タグと終了タグの間のテキストを、ローカル変数にすることができません。このような例はありますか？

回答

次の文を使用すると解決できます。

```
selectSingleNode("pattern");
getNodeValue()
```

要素ノードから値を取得する場合、子ノード #text まで移動して、getFirstChild.getNodeValue() などを実行します。

xmldom.DOMNode n の開始タグと終了タグの間にあるテキストを取得する必要がある場合、次の 2 行のみで取得できます。

```
n_child:=xmldom.getFirstChild(n);
text_value:=xmldom.getNodeValue(n_child));
```

n_child は、xmldom.DOMNode 型です。

text_value は、VARCHAR2 型です。

セキュリティ : ORA-29532 - ユーザーへの JavaSysPriv 権限の付与

質問

XML Parser for PL/SQL を使用して、XML 文書を解析しようとしています。次の Java セキュリティ・エラーが発生します。

```
ORA-29532: 不明な Java 例外で Java コールが終了しました : java.lang.SecurityException
ORA-06512: 0 行 NSEC.XMLPARSERCOVER
ORA-06512: 79 行 NSEC.XMLPARSER
ORA-06512: 36 行 NSEC.TEST1_XML
ORA-06512: 5 行
```

ユーザーに権限を付与する必要がありますか ? 構文は正確なようです。デモを実行しても、エラーが発生します。

回答

解析する文書に <!DOCTYPE が含まれ、file:// や http:// などのプロトコルでシステム URI が指定されている場合、データベースへのアクセス権、ファイルまたは URL.CONNECT SYSTEM/MANAGER でストリームをオープンする権限を、現行のデータベース・ユーザーに付与する必要があります。

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO  youruser;
```

これで解決できます。

XML Parser for PL/SQL のインストール : Oracle JVM オプション

質問

README では、loadjava を使用して、xmlparserv2.jar および plsql.jar を順番にアップロードすることが記載されています。Oracle に jar ファイルをアップロードするために、次のコマンドを使用して xmlparserv2.jar のロードを試みました。

loadjava -user test/test -r -v xmlparserv2.jar

大部分をアップロードした後、次のエラー・メッセージが表示されました。

```
identical: oracle/xml/parser/v2/XMLConstants is unchanged from previously loaded
fileidentical: org/xml/sax/Locator is unchanged from previously loaded fileloading
: META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while creating resource
META-INF/MANIFEST.MF    ORA-29547: Java システム・クラスが使用できません :
oracle/aurora/rdbms/Compilerloading  : oracle/xml/parser/v2/mesg/XMLErrorMesg_en_
US.propertiescreating : oracle/xml/parser/v2/mesg/XMLErrorMesg_en_US.propertiesError
while creating
...
```

そのため、前述のコマンドから -r を削除しました。

```
loadjava -user test/test -v xmlparserv2.jar
```

まだエラーが発生しますが、4 つに減りました。

```
.identical: org/xml/sax/Locator is unchanged from previously loaded fileloading  :
META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while creating
...
```

Oracle JVM は、データベースに正常にインストールされていると思います。

回答

loadjava の実行中にこのようなエラーが発生する場合、Oracle JVM オプションが正常にインストールされていません。Java VM を適切にインストールするには、INITJVM.SQL および INITDBJ.SQL を実行する必要があります。これらは通常、ORACLE_HOME の ./javavm サブディレクトリにあります。

CLOB 内の XML

質問

Oracle9*i* のデータベースには、1MB 以下の整形式の XML 文書を含む CLOB があります。

文書全体を処理するかわりに、CLOB（XML 文書）の一部のみを取得し、それを変更して、データベースに返す機能が必要です。

次に、この処理をデータベース層全体で実行する必要があります。

これを行うには、どの製品またはツールが必要ですか？Oracle に付属する JVM で実行できますか？ストアド・プロシージャによってこれを実行できる PL/SQL ツールはありますか？

回答

次のいずれかを使用して実行できます。

- Oracle XML Parser for PL/SQL
- Oracle XML Parser for Java を使用して作成したコード上に、カスタム Java ストアド・プロシージャ・ラッパーを作成します。

XML Parser for PL/SQL には、次のメソッドがあります。

- xmlparser.parseCLOB()

他にも次のメソッドがあります。

- xslProcessor.selectNodes()（検索している文書の部分を検出します）
- xmldom.*（XML 文書の内容を操作します）
- xmldom.writeToCLOB()（再度書き込みます）

CLOB のテキストを部分的に更新する必要がある場合、DBMS_LOB.* ルーチンを使用する必要があります。ただし、内容の変更に文字数の増加または減少が伴う場合、注意が必要です。

oracle.xml.parser でのメモリー不足のエラー

質問

次のとおり、oracle.xml.parser でのメモリー不足のエラーが発生しています。

```
last entry at 2000-04-26 10:59:27.042:
VisiBroker for Java runtime caught exception:
java.lang.OutOfMemoryError
   at oracle.xml.parser.v2.XMLAttrList.put(XMLAttrList.java:251)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:260)
   at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:228)
   at cars.XMLServer.processEXL(XMLServer.java:122)
```

新しい XML 属性の作成を試みながら、OutOfMemoryError でクラッシュしています。

50MB の XML ファイルを解析中です。200MB の shared_pool_size を使用して、java_pool_size を 150MB に増加しています。

回答

50MB の XML ファイルの解析には、DOM パーサーを使用しないでください。SAX パーサーは実行中にノードのメモリー内ツリーを作成しないため、任意のサイズのファイルを解析する SAX パーサーを使用する必要があります。

SAX または DOM のどちらのパーサーを使用していますか？DOM を使用している場合、ファイルを表すメモリー内ツリーを構築するかわりに、順次 XML ファイルを処理する SAX に移行することをお薦めします。

SAX を使用すると、非常に少量のメモリー量で、180MB を超える XML ファイルを問題なく処理できます。

DOM および SAX 使用のガイドラインは、次のとおりです。

DOM:

- ランダム・アクセスが必要な場合に適しています。
- より多くのメモリーを使用します。
- 変換を行う場合も適しています。
- ツリーを反復したり、ドキュメント・ツリー全体を移動する場合も適しています。
- DOM インタフェースを使用する場合は、（パイプ・サイズを小さくするために）XML 内の要素より多くの属性を使用できるかどうかを確認してください。

SAX:

- SAX インタフェースは、データが（I/P ストリームを使用して）ストリーム形式で受信される場合に使用します。

C 言語ベースの XML Parser for PL/SQL

質問

C 言語ベースの XML Parser for PL/SQL がありますか？

回答

現在、C 言語ベースの XML Parser for PL/SQL はありませんが、XML Parser for C をベースにした XML Parser for PL/SQL を提供する予定です。

XML Parser for PL/SQL 使用時のメモリー要件

質問

PL/SQL 使用時のメモリー要件は何ですか？

回答

メモリー使用は、ドキュメントのサイズに直接依存しますが、XML Parser for PL/SQL は Java パーサーを使用するため、Oracle JVM が実行中であることに注意してください。Oracle JVM には、その構成によって通常 40 ～ 60MB のメモリーが必要です。

XML Parser for PL/SQL 実行時の Oracle JVM の必要性

質問

XML Parser for PL/SQL を実行するには、Oracle JVM をインストールする必要がありますか？

回答

データベースでパーサーを実行中の場合、現在、XML Parser for PL/SQL はバックグラウンドで XML Parser for Java を使用するため、Oracle JVM をインストールする必要があります。Oracle JVM は、Standard および Enterprise の両方のバージョンに付属しています。C をバックグラウンドで使用する XML Parser for PL/SQL の次期バージョンを、JVM にアクセスしないアプリケーション用に開発中です。

DOM API の使用

質問
XML Parser for PL/SQL の機能を教えてください。

回答
XML パーサーは、すべての XML 文書を受け入れ、文書の要素および属性にアクセスしたり、それらを変更するためのツリーベースの API（DOM）を提供します。XML 文書から別の XML 文書への変換を可能にする XSLT もサポートしています。

質問
XML 文書に対して、エンコーディングを動的に設定できますか？

回答
できません。仕様のとおりに、文書内で適切なエンコーディング宣言を含める必要があります。setCharset(DOMDocument) を使用して、文書の入力にエンコーディングを設定できません。SetCharset(DOMDocument) および oracle.xml.parser.v2.XMLDocument を使用して、出力に正しいエンコーディングを設定します。

質問
XML パーサーを使用して特定のタグの要素数を取得する方法を教えてください。

回答
getElementByTagName (elem DOMElement, name IN VARCHAR2) メソッドを使用します。このメソッドは、指定されたタグ名を持つすべての子要素の DOMNodeList を戻します。その DOMNodeList 内の要素数を調べて、特定のタグの要素数が判断できます。

質問
文字列を解析する方法を教えてください。

回答
文字列に含まれている XML 文書を直接解析する方法は現在はありません。かわりに、次のファンクションを使用できます。

- parse (Parser, VARCHAR2) ファンクション（指定された URL またはファイルに格納された XML データを解析します）
- parseBuffer (Parser, VARCHAR2) ファンクション（指定されたバッファに格納された XML データを解析します）
- parseCLOB (Parser, VARCHAR2) ファンクション（指定された CLOB に格納された XML データを解析します）

質問
XML 文書を表示する方法を教えてください。

回答
Internet Explorer 5 ブラウザを使用している場合、直接 XML 文書を表示できます。それ以外のブラウザの場合、XML パーサーの XSLT プロセッサを使用すると、XSL スタイルシートを使用して HTML ドキュメントを作成できます。Java で実装された XML Transviewer Beans を使用しても、XML 文書を表示できます。

質問
特殊キャラクタ・セットを使用して XML データを再度書き込む方法を教えてください。

回答
ファイルまたはバッファに書き込むための、キャラクタ・セットを指定します。CLOB に書き込む場合、書き込むデータベース用のデフォルトのキャラクタ・セットを使用します。使用するメソッドは次のとおりです。

- writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2); プロシージャ
- writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2); プロシージャ
- writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2); プロシージャ

質問

文字データからアンパサンド（&）を取得する方法を教えてください。

回答

XML データでは、アンパサンドをそのまま使用できません。かわりに、エンティティ & を使用する必要があります。このエンティティは、XML 標準で定義されています。

質問

ファイルからドキュメント・オブジェクトを生成する方法を教えてください。

回答

次の例を参照してください。

```
inpPath VARCHAR2;
inpFile VARCHAR2;
p xmlparser.parser;
doc xmldom.DOMDocument;

-- initialize a new  parser object;
p := xmlparser.newParser;
-- parse the file
xmlparser.parse(p, inpPath || inpFile);
-- generate a document object
doc := xmlparser.getDocument(p);
```

質問

パーサーは Linux 上で実行できますか？

回答

Java VM for Linux のバージョン 1.1.x または 1.2.x がインストールされている場合、Linux 上で Oracle XML Parser for Java を実行できます。インストールされていない場合、C または C++ 用の XML Parser for Linux を使用します。

質問

XML パーサーを使用して、>、<、>= または <= を比較する方法を教えてください。

回答

< にはエンティティ < および、> にはエンティティ > を使用する必要があります。

質問
XML 名前空間および XML Schema はサポートされていますか？

回答
現在の XML パーサーは XML 名前空間をサポートします。XML Schema のサポートは、将来のリリースに含まれる予定です。

質問
パーサーが DTD ファイルを検出しません。

回答
<!DOCTYPE> 宣言で定義された DTD ファイルは、入力 XML 文書の位置に相対的である必要があります。相対的でない場合は、setBaseDir(Parser,VARCHAR2) ファンクションを使用してベース URL を設定し、DTD の相対アドレスを解決する必要があります。

質問
外部 DTD を使用して、XML ファイルを検証できますか？

回答
XML 文書に、適用可能な DTD への参照を含める必要があります。この参照がないと、パーサーは検証に使用する DTD を検出できません。参照の指定方法は、外部 DTD を指定するときに使用する XML の標準方法です。参照を指定しない場合、XML 文書に DTD を埋め込む必要があります。

質問
DTD キャッシュ機能はありますか？

回答
DTD キャッシュはオプションであり、自動的に有効にはなりません。

質問
XML 文書を解析してから、DOCTYPE タグを挿入する方法を教えてください。

回答
ファイルにいくつかの事前処理を行い、再度 DOM パーサーに通す必要があります。これによって、DOCTYPE タグを含む、妥当および整形式の XML 文書が生成されます。

質問
XML DOM Parser の動作方法を教えてください。

回答
XML DOM Parser は XML 形式のドキュメントを受け入れ、構造に基づいて DOM ツリーをメモリー内に構築します。ドキュメントが整形式であるかどうかを確認し、オプションで DTD に準拠しているかどうかを確認します。XML DOM Parser は、ツリーを全検索して、ツリーからデータを返す方法も提供します。

質問
値を後で設定できるノードを作成する方法を教えてください。

回答
ノード・タイプを説明する表を参照する DOM 仕様を確認すると、要素ノードを作成する場合に nodeValue が NULL になるため、ノードを設定できないことがわかります。ただし、テキスト・ノードを作成し、要素ノードへ追加することはできます。そのテキスト・ノードには値を格納できます。

質問
XML ファイルの要素を取得する方法を教えてください。

回答
DOM を使用している場合は、NamedNodeMap メソッドを使用して要素を取得できます。

質問
XML Parser for PL/SQL を使用して、テキスト・ノードを DOMElement に追加する方法を教えてください。

回答
createTextNode() メソッドを使用して、新しいテキスト・ノードを作成します。次に、makeNode() を使用して、DOMElement を DOMNode に変換します。これで、appendChild() を使用して、テキスト・ノードを DOMElement に追加できます。

質問
DOM で XML パーサーを使用中ですが、実際のデータを取得できません。何が間違っていますか？

回答
データがどのレベルに常駐しているのかを確認する必要があります。次に例を示します。

- <?xml version=1.0 ?>
- <greeting>Hello World!</greeting>

テキストは、ドキュメントにある最初の DOM 要素の最初の子ノードです。DOM レベル 1 仕様によると、要素ノードの値は NULL で、getNodeValue() メソッドは常に要素型ノードに NULL を戻します。要素の子である TEXT を取得し、getNodeValue() メソッドを使用して、ノードから実際のテキストを取り出します。

質問
XML Parser for PL/SQL は、HTML などの非 XML 文書を生成するスタイルシートを処理できますか？

回答
処理できます。

サンプルの使用

質問
サンプル・ファイルを実行できません。インストール方法を間違えたのでしょうか？

回答
次に、XML Parser for PL/SQL のインストール時に実行し忘れることが多い処理を示します。

- Oracle JVM の初期化（`$ORACLE_HOME/javavm/install/initjvm.sql` の実行）
- パーサー・アーカイブからのインクルード jar ファイルのロード

XML Parser for PL/SQL: CLOB 内の DTD の解析

質問

CLOB に含まれる DTD ファイルの解析が正常に実行できません。XML Parser for PL/SQL によって提供された API、「xmlparser.parseDTDClob」を使用しています。

次のエラーが発生しています。

ORA-29531: メソッド parseDTD はクラス oracle/xml/parser/plsql/XMLParserCover にはありません。

次のとおり実行するようにしています。

xmlparser.parseDTDClob プロシージャは、Java ストアド・プロシージャ xmlparsercover.parseDTDClob をコールし、このストアド・プロシージャが別の Java ストアド・プロシージャ xmlparsercover.parseDTD をコールします。

クラス・ファイル「oracle.xml.parser.plsql.XMLParserCover」が、データベースにロードされ、公開されていることを確認しています。そのため、エラー・メッセージの意味が理解できません。

解析方法に問題があるのか、パーサー API にエラーがあるのかが判断できません。

```
The procedure use to call "xmlparser.parseDTDClob" :
-----------------------------------------------------
create or replace procedure parse_my_dtd as
p xmlparser.parser;
l_clob clob;
begin
  p := xmlparser.newParser;
  select content into l_clob from dca_documents where doc_id = 1;
  xmlparser.parseDTDClob(p,l_clob,'site_template');
end;
```

xmlparser.parseDTDClob の API ドキュメントは次のとおりです。

```
parseDTDClob
PURPOSE
  Parses the DTD stored in the given clob
SYNTAX
  PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2);
PARAMETERS
  p        (IN)-  parser instance
 dtd      (IN)-  dtd clob to parse
 root     (IN)-  name of the root element
RETURNS
  Nothing
COMMENTS
```

パーサーのデフォルト動作への変更が有効になってから、このプロシージャをコールします。なんらかの理由で解析が正常に実行されない場合、アプリケーション・エラーが発生します。

dca_documents 表の内容は次のとおりです。

```
DOC_ID          NOT NULL    NUMBER
DOC_NAME        NOT NULL    VARCHAR2(350)
DOC_TYPE                    VARCHAR2(30)
DESCRIPTION                 VARCHAR2(4000)
MIME_TYPE                   VARCHAR2(48)
CONTENT         NOT NULL    CLOB
CREATED_BY      NOT NULL    VARCHAR2(30)
CREATED_ON      NOT NULL    DATE
UPDATED_BY      NOT NULL    VARCHAR2(30)
UPDATED_ON      NOT NULL    DATE
```

DTD の内容は次のとおりです。

```
<!ELEMENT site_template (component*)>
<!ATTLIST site_template template_id CDATA #REQUIRED>
<!ATTLIST site_template template_name CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component component_id ID #REQUIRED>
<!ATTLIST component parent_id ID #REQUIRED>
<!ATTLIST component component_name ID #REQUIRED>
```

回答 a

LOB を使用して何を行いますか? LOB には一時 LOB または永続 LOB があります。永続 LOB の場合、表に値を挿入する必要があります。一時 LOB の場合、プログラムでインスタンス化できます。

次に例を示します。

```
persistant lob
declare
  clob_var CLOB;
begin
  insert into tab_xxx values(EMPTY_CLOB()) RETURNING clob_col INTO lob_var;
      dbms_lob.write(,,,,);
 // send to AQ
end;
temp lob -----

 declare
   a clob;
 begin
```

```
    dbms_lob.createtemporary(a,DBMS_LOB.SESSION);
    dbms_lob.write(...);
   // send to AQ

end;
/
```

『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』も参照してください。PL/SQL の場合、次の文を実行するのみです。

```
myClob CLOB = clob();
```

`DBMS_LOB.createtemporary()` を試みましたが、正常に実行できます。

回答 b

AQ で LOB を使用している場合は、次の手順に従います。

1. 1 つのフィールドを CLOB 型に指定して、ADT を作成します。

   ```
   create type myAdt (id NUMBER, cdata CLOB);
   ```

   キュー・テーブルは myAdt 型であることを宣言する必要があります。

2. オブジェクトをインスタンス化します。LOB フィールドを empty_clob() に指定します。

   ```
   myMessage := myAdt(10, EMPTY_CLOB());
   ```

3. メッセージをエンキューします。

   ```
   clob_loc clob;
   enq_msgid RAW(16);
   DBMS_AQ.enqueue('queue1', enq_opt, msg_prop, myMessage, enq_msgid)
   ```

4. LOB ロケータを取得します。

   ```
   select t.user_data.cdata into clob_loc
   from qtable t  where t.msgid
   = enq_msgid;
   ```

5. dbms_lob.write を使用して、CLOB を移入します。

6. コミットします。

コミットの例は『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。Java API for AQ を使用している場合、手順は少し複雑になります。

ドキュメント解析中のエラー

質問

XML Parser for PL/SQL インタフェースを使用しています。3 つのタグで構成される XML ファイルがあり、解析時にエラーが発生します。

ドキュメントを個々のタグで分割すると、2 つのタグは問題ないのですが、3 番目のタグにエラーが発生します。

1. データを分割した場合、エラーが異なるのはどうしてですか?
2. ドキュメント内の無限文字列を見つけることができません。
3. これがデータが入力される唯一の方法であり、別のパーサーを試す時間もありません。

回答

ドキュメントが 3 つのタグで構成されている場合、複数のルート要素があるためドキュメントは整形式ではありません。3 つのタグの前後に、開始タグおよび終了タグを挿入してください。

PLXML: 指定された URL の解析

質問

Windows NT 上で XML Parser for PL/SQL を使用しています。パーサー API のドキュメントによると、「指定された URL/ ファイルに格納された XML を解析し、構築された DOM 文書を戻します。」のとおり、指定された URL も解析できるとあります。ファイルからの解析は正常に実行されますが、すべての形式の URL で、「ORA-29532: 不明な Java 例外で Java コールが終了しました : java.io.FileNotFoundException」が発生します。

コール例を教えてください。

回答

外部 URL にアクセスするには、プロキシ・ホストおよびポートを設定する必要があります。たとえば、次のタイプの構文を使用します。

```
java -Dhttp.proxyHost=myproxy.mydomain.com -Dhttp.proxyPort=3182 DOMSample myxml.xml
```

XML パーサーを使用した HTML の解析

質問

次のとおりに HTML ファイルを解析する必要があります。

1. 各「a href」を検出します。
2. 検出された各 a href に対して、リンク先のファイル / パス名を取得します。
3. ファイル / パス名をパラメータとして指定し、a href にデータベース・プロシージャ・コールを置き換えます。

XML Parser for PL/SQL を使用して、これを実行できますか ? 実行できる場合、その難易度および実行方法を教えてください。

回答 a

HTML ファイルは整形式の XML 文書である必要はありませんが、本当に XML パーサーを使用する必要がありますか ? PERL を使用することを検討してください。XML Parser for PL/SQL が次のメソッドをサポートするかどうか確かではありませんが、参考までに示します。

1. `getElementsByTagName()`（すべての一致するノードを取り出します）
2. `getNodeValue()`（文字列を戻します）
3. `setNodeValue()`（ノード値を設定します）

回答 b

XML Parser for PL/SQL はこれらのメソッドをサポートしますが、整形式ではない HTML ファイルではサポートしません。

Oracle 7.3.4: Web ブラウザへのデータの送信（PL/SQL）

質問

次のコンポーネントを使用して、クライアント側の Web ブラウザにデータを送信します。ただし、すべての処理はサーバー（Oracle 7.3.4）上で行われる必要があります。

- XML Parser for PL/SQL
- XSQL Servlet

これを行うには、この 2 つのコンポーネントで十分ですか？

回答

XSQL Servlet 実行の要件は次のとおりです。

- Oracle XML Parser
- Oracle XSU for Java
- Java Servlet をサポートする Web サーバー
- JDBC ドライバ

XSQL Servlet 自体も必要です。

Oracle 7.3.4 および XML

質問

XML Parser for Java は Oracle 7.3.4 で動作しますか？

XSU は XML Parser for Java の一部ですか？または個別にダウンロードする必要がありますか？

回答

1. 適切な JDBC ドライバがあり、中間層またはクライアント側の VM で実行する場合、XML Parser for Java は Oracle 7.3.4 で動作します。
2. XML Parser for Java は XSU に必要であるため、XSU のダウンロードにコピーが含まれています。

getNodeValue(): DomNode 値の取得

質問

xmlparser() を使用した後、XML タグの間の値を取得できません。DOMSAMPLE.SQL のコードは、次のとおりです。

```
-- loop through elements
for i in 0..len-1 loop
   n := xmlparser.item(nl, i);
   dbms_output.put(xmlparser.getNodeName(n);
```

コメント

同様の問題が以前発生し、要素ノードの getNodeValue() が NULL を戻すことがわかりました。また、テキスト・ノードの getNodeValue() は値を戻します。

ノードのすべての子または孫の取出し

質問

DOM API を使用して、DOM ツリー内の特定ノードのすべての子や孫などを取り出す方法を教えてください。これを行うための解決策はありますか？XML Parser for PL/SQL を使用しています。

回答

次の文を実行します。

```
DECLARE  nodeList   xmldom.DOMNodeList;
theElement xmldom.DOMElement;
BEGIN    :nodeList := xmldom.getElementsByTagName( theElement,'*');
:END;
```

これによって、ルート要素が「theElement」であるすべての子ノードが取得されます。

ORA-29532「不明な Java 例外で Java コールが終了しました : java.lang.ClassCastException」が発生する原因

質問

XML を解析し、XSL を適用して、その変換結果を XML 文書の形式で取得する必要があります。XML Parser for PL/SQL を使用しています。スクリプトでは、変換結果に、処理命令（PI）<?xml version="1.0"?> が追加されません。

`XSLProcessor.processXSL` メソッドは、文書フラグメント・オブジェクトを戻します。

finaldoc := xmldom.MakeDocument(docfragnode) を使用して、その文書フラグメント・オブジェクトから DOM 文書を作成します。

xmldom.DOMDocument 型の finaldoc が作成された場所で、結果ファイルに書き込みます。

```
xmldom.writeToFile(finaldoc, dir || '/' || resfile);
```

このメソッドは DOM 文書で使用可能ですが、次のエラーが発生します。

```
ora-29532 不明な Java 例外で Java コールが終了しました :java.lang.ClassCastException
```

文書フラグメントを文書オブジェクトに変換すると命令「<?xml version="1.0"?>」が追加されるのか、または XSL を介して、この命令を追加する必要があるのかを教えてください。

回答

新しい DOM 文書を作成して、それに文書フラグメントを追加した場合、`xmldom.writeToBuffer()` または類似のルーチンを使用して、XML 宣言とともに所定の位置にシリアル化できます。

A

XML の手引き

この付録の内容は次のとおりです。

XML の概要

XML は、Web 上のデータを識別および記述する標準の方法です。広範囲に実装可能で、簡単に配置できます。

XML は、階層構造を持つデータを記述するために使用する、人間およびマシンが理解できる一般的な構文で、様々なアプリケーション、データベース、E-Commerce、Java、Web 開発、検索などに適用できます。

カスタム・タグを使用すると、アプリケーション間および組織間のデータを定義、送信、検証および解析できます。

タグ

XML 要素は、開始タグ（<）および終了タグ（>）を使用します。たとえば、<author> の場合、タグ名 author が開始タグおよび終了タグで囲まれています。タグには、任意の名前を付けることができます。

属性

属性は、各 XML 要素に関する情報を追加します。属性を使用すると、データをエンコーディングまたは表現する方法を記述したり、リンクまたは外部リソースの位置を示したり、アプレット、サーブレットなどの外部プロセスを識別およびコールすることができます。また、ドキュメント内に要素インスタンスを指定して、ドキュメントの検索中に、その要素インスタンスをすばやく検索できるようにすることもできます。属性は、XML 文書のコンテンツまたは他の要素に関する追加情報を提供することもできます。属性は、フォント、カラーまたは他のスタイルやフォーマットを指定するためには使用できません。

XML 属性は、開始タグと終了タグの組の開始タグ、または空タグ内に保持されます。名前と値の組の XML 属性（たとえば、<image="adx10.jpg" ada_txt="XSQL Description"/>）を指定することができます。属性は、引用符で囲む必要があります。

属性および属性の内容は、DTD または XML Schema で定義されます。

要素

<author>charles kopman</author> は、XML 文書内の要素の例です。要素には、開始タグ、終了タグおよび開始タグと終了タグの間にあるテキストが含まれます。

すべての XML 文書には、ルート要素または最上位の要素が必要です。これは、最も外側にある要素で、すべての他の要素を含みます。ルート要素の名前は、任意に選択できます。HTML では、ルート要素は常に <html>....</html> です。

エンティティ

エンティティとは、図形、テキスト、サウンド・ファイルおよびバイナリ・データを格納できる仮想の格納単位です。XML では、エンティティは文字列で表されます。独自のエンティティを作成することができます。次の 5 つの内部エンティティが、XML で使用できるように定義されています。

不等号（より小さい）（<）には（<）を使用します。

不等号（より大きい）（>）には（>）を使用します。

アンパサンド（&）には（&）を使用します。

一重引用符またはアポストロフィ（'）には（'）を使用します。

二重引用符（"）には（"）を使用します。

XML マークアップの基本規則

次に、8 つの XML マークアップ基本規則を示します。

- 最初に、XML を宣言します。XML 文書の第 1 行目には、W3C の XML 勧告に準拠した XML 文書であることを示す XML 宣言（たとえば、<?xml version="1.0" standalone="yes" ?>）が必要です。
- 1 つの最上位タグ（「ドキュメント要素」）またはルート・タグを使用します。すべてのタグおよび XML コンテンツは、この最上位タグの中（下）に含まれます。
- すべての要素には、開始タグおよび終了タグ（たとえば、<author>charles kopman</author>）が必要です。
- 空要素は、/> で終わります（たとえば、<author name="charles kopman" />）。
- 要素が、正しい階層に適切にネストされていることを確認します。
- すべての属性値は、一重または二重引用符で囲みます（たとえば、<author name = "charles kopman">）。
- すべての XML タグは、< で始まります。すべての XML エンティティは、& で始まり、; で終わります。
- 前述の 5 つの内部エンティティを覚えておきます。A-3 ページの「エンティティ」を参照してください。
- XML 文書の最上部で、XML 宣言にどの文字コードを使用しているかを XML パーサーに指定できます（<?xml version="1.0" encoding="ISO-8859-9" ?>）。

  エンコーディングの完全なリストについては、http://www.iana.org/assignments/character-sets を参照してください。

W3C の XML 勧告

次に示す W3C の XML 勧告は、互いに連動する一連の仕様で、その数は現在も増加しています。

- **XML 1.0**

  1998 年 2 月に W3C によって勧告されました。その結果、多数の W3C ワーキング・グループ、Java プラットフォーム拡張エキスパート・グループ、および電子データ交換（EDI）などの大量のデータを交換する標準の XML 変換が生まれました。HTML の次期バージョンは、XHTML として知られる XML アプリケーションになります。

- **XML Namespace**

  W3C 勧告であり、複数の名前空間を持つ整形式の XML アプリケーション内にある要素のあいまいさを取り除くことを目的としています。

- **XML Query**

  W3C が策定している標準で、XML 文書に対する問合せ言語を指定します。

- **XML Schema**

  W3C が策定している標準で、単純または複雑なデータ型を XML 文書に追加し、DTD の機能を XML Schema 定義の XML 文書に置き換えます。

- **XSL**

  次の 2 つの W3C 勧告で構成されます。

  - XSL Transformation（XML 文書を他の XML 文書に変換します）
  - XSL Formating Object（XML 文書の表示を指定します）

- **XPath**

  W3C 勧告であり、XSLT、XLink および XML Query によって使用される XML 文書をナビゲートするためのデータ・モデルおよび文法を指定します。

- **XPointer**

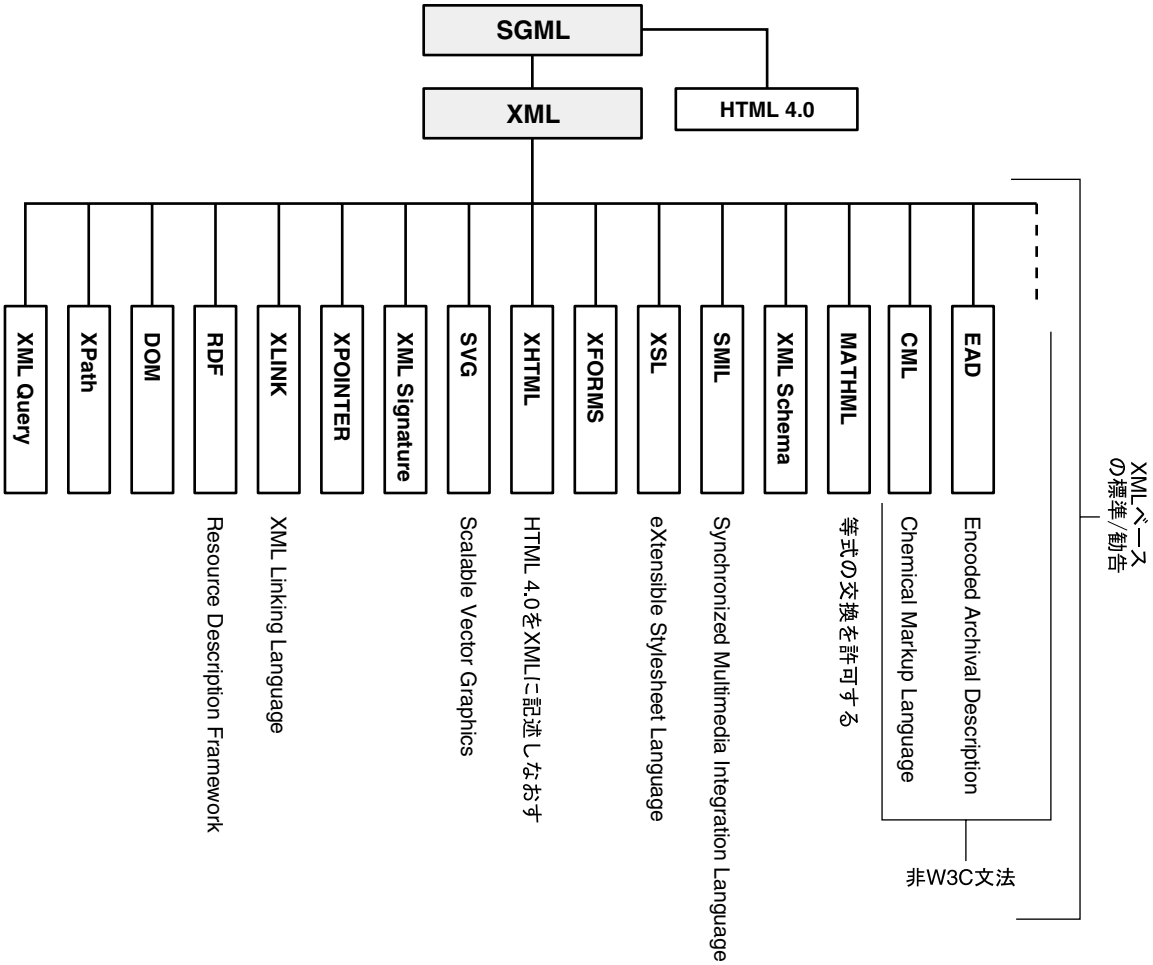
  W3C 勧告であり、XPath ナビゲーションを使用して、XML 文書内の個別のエンティティまたはフラグメントの識別を指定します。この W3C 勧告は、http://www.w3.org/TR/WD-xptr で定義されています。

- **DOM**

  W3C 勧告であり、プログラム・アクセス用の API を含む XML 文書の文書オブジェクト・モデルを指定します。

図 A-1 に、XML アプリケーション・グループを示します。

図 A-1 XML アプリケーション・グループ（XML ベースの標準を含む）



XML の機能

XML の機能は次のとおりです。

- **構造化データから非構造化データへのデータ交換**

  XML は、データ交換の汎用標準構文であるといえます。XML は、非常に正確なテキストベースの方法を指定してデータ固有の構造を表すため、一意に作成および解析されます。XML の単純でタグベースの仕組は、HTML と同様に簡単に理解できますが、高度に構造化されたデータベース・レコードから構造化されていないドキュメントまで、「デジタル資産」全般に対応できる柔軟で拡張可能な機能を提供します。

- **特にドキュメント用に設計された SGML - あらゆるデータ用に設計された XML**

  SGML は、特にドキュメント用に設計されました。Web 中心の XML は、他言語の記述に使用できるツールキットのようなものです。XML は、ドキュメントに限ったものではありません。ツリーで記述可能なすべてのデータは、XML でプログラムできます。

- **データ・オブジェクトのクラス - SGML の制限付き形式**

  www.oasis-open.org では、XML を次のように説明しています。
  （前略）XML は XML 文書というデータ・オブジェクトのクラスを記述し、ドキュメントを処理するコンピュータ・プログラムの動作を部分的に記述します。XML はアプリケーション・プロファイル、または SGML の制限付き形式です。構成的に、XML 文書は SGML に準拠したドキュメントです。

- **多くの XML の使用方法**

  W3C.org プレス・リリースでは、XML を次のように説明しています。
  （前略）XML は本来、業界固有のマークアップ、ベンダーに中立的なデータ交換、メディアに依存しない公開、1 対 1 のマーケティング、協調的なオーサリング環境でのワークフロー管理、およびインテリジェント・クライアントによる Web ドキュメントの処理に対する大規模な Web コンテンツ・プロバイダの要件を満たすために開発されました。

- ***メタデータ***

  XML は、メタデータ・アプリケーション内でも使用できます。

- **国際化**

  XML は、UTF-8 および UTF-16 エンコーディングで Unicode キャラクタ・セットをサポートするために必要なすべての対応プロセッサを装備し、ヨーロッパ言語およびアジア言語に完全に対応しています。主に、電子発行およびデータ交換に使用されます。

- **解析対象または解析対象外格納エンティティ**

  W3C.org の XML 仕様提案の抜粋を示します。
  （前略）XML 文書は、エンティティというデータの格納単位で構成され、各エンティティには解析対象または解析対象外のデータが含まれます。解析対象データは文字で構成され、中には文書内の文字データを形成したり、タグを形成するものもあります。タグ付けは、文書のデータ格納のレイアウトおよび論理構造の記述をエンコーディングします。

- **XML プロセッサによる XML 文書の読取り**

  （前略）XML は、データ格納レイアウトおよび論理構造を制約するメカニズムを提供します。XML 文書の読取り、およびその文書のコンテンツと構造へのアクセスには、XML プロセッサというソフトウェア・モジュールが使用されます。XML プロセッサは、アプリケーションという他のモジュールのかわりに作業を行います。

- **インターネットのオープン標準**

  XML は、情報交換用のプラットフォームおよびアプリケーションに中立的な形式として、IBM 社、Sun 社、Microsoft 社、Netscape 社、SAP 社、CISCO 社などの他ベンダーから幅広い業界サポートを獲得しつつあります。

このマニュアルでは、XML の構文については詳しく説明しませんが、XML に関する主な項目の概要を次に示します。XML 構文の詳細は、A-15 ページの「XML 関連のリソース」に示すリソースを参照してください。

XML と HTML の違い

HTML と同様に、XML は SGML のサブセットであり、Web 上での配信に最適です。

ブラウザ上で表示するために Web ページの要素をタグ付けする（たとえば、<bold>Oracle</bold>）HTML とは異なり、XML は、データとして要素をタグ付けします（たとえば、<company>Oracle</company>）。たとえば、XML を使用して Web ページ内の単語および値に意味を持たせることができ、単純なテキストまたは数字要素としてではなく、データとして認識させます。

HTML のコード例を次に示します。その後、対応する XML のコード例を示します。例では、次の従業員データを示しています。

- 従業員数
- 名前
- 職種
- 給与

HTML の例 1

```
<table>
   <tr><td>EMPNO</td><td>ENAME</td><td>JOB</td><td>SAL</td></tr>
   <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>1250</td></tr>
   <tr><td>7788</td><td>SCOTT</td><td>ANALYST</td><td>3000</td></tr>
 </table>
```

XML の例 1

XML コードでは、XML データ・タグが追加され、要素の構造がネストしていることに注意してください。

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
     <EMPNO>7654</EMPNO>
     <ENAME>MARTIN</ENAME>
     <JOB>SALESMAN</JOB>
     <SAL>1250</SAL>
    </EMP>
    <EMP>
     <EMPNO>7788</EMPNO>
     <ENAME>SCOTT</ENAME>
     <JOB>ANALYST</JOB>
     <SAL>3000</SAL>
    </EMP>
  </EMPLIST>
```

HTML の例 2

タグを使用して表の行内のデータを表す次の HTML を考えてみます。「Java Programming」が本の名前であるか、大学の科目名であるか、仕事のスキルであるかは、ページ上のデータおよびタグを見るだけでは判断できません。これを計算するコンピュータ・プログラムについて考えてみます。

```
<HTML>
  <BODY>
    <TABLE>
     <TR>
     <TD>Java Programming</TD>
     <TD>EECS</TD>
     <TD>Paul Thompson</TD>
     <TD>Ron<BR>Uma<BR>Lindsay</TD>
     </TR>
    </TABLE>
  </BODY>
</HTML>
```

XML の例には同様のデータが含まれますが、タグは表示方法ではなく、データが何の情報を表すかを示します。「Java Programming」が大学の科目名であることがはっきりしましたが、表示方法についての説明はありません。

XML の例 2

```
<?xml version="1.0"?>
  <Course>
      <Name>Java Programming</Name>
    <Department>EECS</Department>
    <Teacher>
      <Name>Paul Thompson</Name>
    </Teacher>
    <Student>
      <Name>Ron</Name>
    </Student>
    <Student>
      <Name>Uma</Name>
    </Student>
    <Student>
      <Name>Lindsay</Name>
    </Student>
  </Course>
```

XML および HTML は両方とも情報を表します。

- XML は情報の内容を表します。
- HTML はその内容の外観を表します。

XML と HTML の違いの概要

図 A-1 に、XML と HTML の違いの概要を示します。

表 A-1 XML と HTML の違い

XML	HTML
情報の内容を表します。	内容の外観を表します。
ユーザー定義のタグを含みます。	標準で定義されたタグの固定セットを含みます。
すべての開始タグには終了タグがあります。	現行のブラウザでは、<P>、<B> などのタグに終了タグが必須ではありません。
属性は、一重または二重引用符で囲む必要があります。	現行のブラウザでは、この要件が必須ではありません。
空要素は、明確に示します。	現行のブラウザでは、この要件が必須ではありません。
要素名および属性は、大文字 / 小文字が区別されます。	要素名および属性は、大文字 / 小文字が区別されません。

スタイルシートを使用した XML の表示

データソースとして XML を使用する主なメリットは、XML の外観（Web ページなど）を構造および内容から切り離せることです。

- **外観 :** 適用したスタイルシートが、表示を定義します。XML データは、異なるスタイルシートを適用するだけで、外観および構成を様々な方法で表すことができます。
- **構造および内容 :** XML データは、構造および内容を定義します。

スタイルシートの使用

次のスタイルシートの使用方法を検討してみます。

- それぞれの表示スタイル用に異なるスタイルシートを定義することによって、ユーザー・プロファイル、ブラウザの種類または他の基準に基づいて、異なるユーザーに異なるインタフェースが表示されます。
- スタイルシートを使用して、XML データを、データを受信および処理する特定のアプリケーション用に調整された形式に変換します。

スタイルシートは、サーバー側またはクライアント側で適用できます。XSLT プロセッサは、XML 形式を、XML または HTML など他のテキストベースの形式に変換します。すべての Oracle XML Parser には、XSLT プロセッサが搭載されています。

スタイルシートの適用方法および XSLT プロセッサの使用方法については、次の章を参照してください。

- 第 4 章「XSL および XSLT の使用」
- 第 20 章「XML Parser for Java の使用」
- 『Oracle9*i* ケース・スタディ -XML アプリケーション』の「XSL を使用した Discoverer 4*i* Viewer のカスタマイズ」の章

XSL

XML のスタイルシート言語である XSL は、W3C の勧告です。XSL が提供するスタイルシートを使用すると、次の操作が可能です。

- XML を、XML または HTML など他のテキストベース形式に変換できます。
- データを再配置またはフィルタできます。
- XML データを、別の DTD に準拠する XML に変換できます。これは、異なるアプリケーション間でデータを共有するための重要な機能です。

CSS

W3C 仕様の CSS1 は、本来、HTML ドキュメントで使用するために作成されました。CSS を使用すると、ドキュメントの外観に関する次の点を制御できます。

- 間隔、要素の見え方、位置およびサイズ
- カラーおよびバックグラウンド
- フォントおよびテキスト

CSS2 は 1998 年に W3C によって公開され、次の機能が追加されています。

- システムのフォントおよびカラー
- 自動ナンバリング
- ページ区切りメディアのサポート
- 表および音声スタイルシート

「カスケード」とは、1 つのドキュメントに複数のスタイルシートを適用できることを意味します。たとえば、CSS を配置する Web ページ上では、3 つのスタイルシートを適用またはカスケードできます。

1. ユーザーが要求するスタイルシートが優先されます。
2. CSS
3. ブラウザ・スタイルシート

参照： 第 4 章「XSL および XSLT の使用」を参照してください。

拡張性および DTD

HTML より XML を使用する別のメリットとして、タグの仕様および使用方法に制約がないことがあります。データの意味および構造を表す独自のタグを作成して、XML 文書を作成します。

タグは、XML 文書中で使用して定義するか、または形式的に DTD で定義できます。データまたはアプリケーションの要件が変更になった場合、タグを変更または追加して新しいデータ・コンテキストを反映するか、または既存のタグを拡張できます。

次に、前述の XML 例の単純な DTD を示します。

```
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, SAL)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
```

注意： DOCTYPE 宣言は、DTD が XML コードに埋め込まれる場合にのみ使用されます。

整形式および妥当な XML 文書

整形式の XML 文書

XML の構造規則および表記規則に準拠する XML 文書は、整形式とみなされます。整形式の XML 文書は DTD を含んだり参照する必要はなく、データ要素およびその関係を暗黙的に定義できます。整形式の XML 文書は、次の規則に従う必要があります。

- 文書は、XML 宣言 <?xml version="1.0"?> で始まります。
- すべての要素は、1 つのルート要素内に含まれる必要があります。
- すべての要素は、オーバーラップすることなくツリー構造にネストする必要があります。
- 空要素以外のすべての要素には、開始および終了タグが必要です。

妥当な XML 文書

DTD にも準拠する整形式の XML 文書は、妥当であるとみなされます。DTD を含んだり参照する XML 文書を解析する場合、解析アプリケーションは、XML が DTD に準拠し、妥当であるかどうかを検証できます。これによって、解析アプリケーションは、すべてのデータ要素およびその内容が DTD に定義された規則に準拠していることを確認して、XML 文書を処理できます。

XML を使用する理由

情報交換のインターネット標準である XML は、次の理由から有効といえます。

- **データ交換の問題を解決します。**次の場合に、効率的なデータ通信が容易に行えます。
  - データが様々な形式およびプラットフォームにある場合
  - データを異なるプラットフォームに送信する必要がある場合
  - データを異なる形式および外観で表示する必要がある場合
  - データを様々なエンド・デバイスで表示する必要がある場合

  このように、XML は、アプリケーションのデータ交換の問題を解決します。XML を使用すると、他の企業およびワークフロー・コンポーネントと容易に通信できます。XML によるデータ交換問題の解決方法の詳細および例は、第 2 章～第 20 章を参照してください。

  Web、データベース、ネットワーキングおよびミドルウェアの相互作用に有効な XML を使用して、Web ベースのアプリケーションを構築できます。XML は、データ送信用に構造化された形式を提供します。

- **業界固有のデータ・オブジェクトは、XML を使用して設計されています。**OAG や XML.org などの組織は、XML を使用して業界ごとのデータ・オブジェクトを標準化しています。これによって、B2B のデータ交換がさらに容易になります。
- **XML を使用すると、データベース内のデータに簡単にアクセス、変換および格納できます。**データベースは、優れたデータの問合せ能力、スケーラビリティおよび可用性を発揮するため、大量のビジネス・データをリレーショナル表およびオブジェクト・リレーショナル表に格納します。このデータは、XML 形式から変換され、オブジェクト・リレーショナル・データベース構造および純粋なリレーショナル・データベース構造に格納されるか、追加の処理のために再度 XML に生成されます。

XML を使用するその他のメリット

次のように、他にも XML を使用するメリットがあります。

- 独自のタグを作成できます。
- 多くのツールが XML をサポートします。
- XML はオープン標準です。
- オープン標準に準拠して構築された XML パーサーは、相互運用が可能なパーサーで、単独のベンダーに拘束されません。XML 仕様は、様々な業界で承認されています。
- XML では、データの表示がデータの構造および内容から切り離されています。データの表示は、簡単にカスタマイズできます。A-10 ページの「スタイルシートを使用した XML の表示」および 2-17 ページの「データ表示のカスタマイズ」を参照してください。
- 汎用性があります。XML は、自己記述型の方法でデータを表示できるため、様々な場所で使用されます。
- 永続性があります。XML 文書としてデータを具体化することによって、プログラムによるアクセスおよび操作を可能にしながらデータは存続できます。
- プラットフォームおよびアプリケーションに依存しません。
- スケーラブルです。

XML 関連のリソース

次に、XML に関する情報のリソースを示します。

- 『The Oracle XML Handbook』Ben Chang、Mark Scardina 他著、Oracle Press 出版
- 『Building Oracle XML Applications』Steve Muench 著、O'Reilly 出版
- 『XML Bible』Elliotte Rusty Harold 著、IDG Books Worldwide 出版
- 『XML Unleashed』Morrison その他著、SAMS 出版
- 『Building XML Applications』St.Laurent、Cerami 著、McGraw-Hill 出版
- 『Building Web Sites with XML』Michael Floyd 著、Prentice Hall PTR 出版
- 『Building Corporate Portals with XML』Finkelstein、Aiken 著、McGraw-Hill 出版
- 『XML in a Nutshell』O'Reilly 出版
- 『Learning XML -（Guide to）Creating Self-Describing Data』Ray 著、O'Reilly 出版
- http://www.xml.com/pub/rg/46
- http://www.xmlmag.com/
- http://www.webmethods.com/
- http://www.infoshark.com
- http://www.clarient.org/
- http://www.xmlwriter.com/
- http://webdevelopersjournal.com/articles/why_xml.html
- http://www.w3schools.com/xml/
- http://www.w3.org/TR/REC-xml
- http://msdn.microsoft.com/xml/default.asp
- http://www.w3.org/TR に、W3C のテクニカル・レポートがあります。
- http://www.w3.org/xml に、W3C の活動概要を示しています。
- http://www.xml.com に、XML に関する業界最新ニュースを示しています。
- http://www.xml-cml.org に、Chemical Markup Language（CML）に関する情報があります。CML ドキュメントは、Jumbo ブラウザ上で参照および編集できます。
- http://www.loc.gov/ead/ に、アメリカ連邦議会図書館向けに開発された Encoded Archival Description（EAD）の情報があります。

- http://www.docuverse.com/xlf に、Extensible Log Format（XLF）の情報があります。XLF は、ログ・ファイル管理を簡素化するために、ログ・ファイルを XML に変換するプロジェクトです。
- http://www.w3.org/Math に、アプリケーション間で等式を交換する方法を提供する MathML に関する情報があります。
- http://www.w3.org/AudioVideo/ に、SMIL に関する情報があります。
- オラクル社は OASIS の公式スポンサーです。OASIS（http://www.oasis-open.org を参照）は、XML アプリケーションの標準化に尽力する世界最大の独立非営利団体です。OASIS はすべての産業からの参加を促進し、競合企業および重複する標準本体のまとめ役を担っています。

B

Oracle XML Parser および Class Generator の言語間比較

この付録では、Oracle XML Parser および Class Generator を言語間で比較します。この付録の内容は次のとおりです。

Oracle XML Parser の比較

表 B-1 に、Oracle XML Parser 機能の言語間の比較を示します。

図 B-1 Oracle XML Parser の比較

Java	C	C++	PL/SQL
Parser V2			
DOM API 2.0 を含む	DOM API 1.0 および CORE 2.0 を含む	DOM API 1.0 および CORE 2.0 を含む	DOM API 2.0 を含む
SAX API 2.0 を含む	SAX API を含む	SAX API を含む	利用不可
XSLT プロセッサ	XSLT プロセッサ	XSLT プロセッサ	XSLT プロセッサ
XML Schema Processor	XML Schema Processor	XML Schema Processor	利用不可
Namespace 1.0 をサポートする	Namespace 1.0 をサポートする	Namespace 1.0 をサポートする	Namespace 1.0 をサポートする
XPath 1.0 をサポートする	XPath 1.0 をサポートする	XPath 1.0 をサポートする	XPath 1.0 をサポートする
ドキュメントが整形式かどうかを確認する	ドキュメントが整形式かどうかを確認する	ドキュメントが整形式かどうかを確認する	ドキュメントが整形式かどうかを確認する
検証および非検証をサポートする	検証および非検証をサポートする	検証および非検証をサポートする	検証および非検証をサポートする

図 B-1 Oracle XML Parser の比較（続き）

Java	C	C++	PL/SQL
キャラクタ・セット（15）： BIG 5 EBCDIC-CP-* EUC-JP EUC-KR GB2312 ISO-2022-JP ISO-2022-KR ISO-8859-1 〜 ISO-8859-9 ISO-10646-UCS-2 ISO-10646-UCS-4 KOI8-R Shift_JIS US-ASCII UTF-8 UTF-16	キャラクタ・セット（15）： BIG 5 EBCDIC-CP-* EUC-JP EUC-KR GB2312 ISO-2022-JP ISO-2022-KR ISO-8859-1 〜 ISO-8859-9 ISO-10646-UCS-2 ISO-10646-UCS-4 KOI8-R Shift_JIS US-ASCII UTF-8 UTF-16	キャラクタ・セット（15）： BIG 5 EBCDIC-CP-* EUC-JP EUC-KR GB2312 ISO-2022-JP ISO-2022-KR ISO-8859-1 〜 ISO-8859-9 ISO-10646-UCS-2 ISO-10646-UCS-4 KOI8-R Shift_JIS US-ASCII UTF-8 UTF-16	キャラクタ・セット（12）： BIG 5 EBCDIC-CP-* EUC-JP EUC-KR GB2312 ISO-2022-JP ISO-2022-KR ISO-8859-1 〜 ISO-8859-9 KOI8-R Shift_JIS US-ASCII UTF-8
デフォルトのキャラクタ・セット： UTF-8	デフォルトのキャラクタ・セット： UTF-8	デフォルトのキャラクタ・セット： UTF-8	デフォルトのキャラクタ・セット： UTF-8
オペレーティング・システム： すべて Oracle のプラットフォーム	オペレーティング・システム： すべて Oracle のプラットフォーム	オペレーティング・システム： すべて Oracle のプラットフォーム	オペレーティング・システム： すべて Oracle のプラットフォーム
致命的エラー以外のエラー・リカバリ	利用不可	利用不可	致命的エラー以外のエラー・リカバリ

Oracle XML Class Generator の比較

表 B-2 に、Class Generator 機能の言語間の比較を示します。

図 B-2 Class Generator の比較

Java	C	C++	PL/SQL
Class Generator	利用不可	-	利用不可
oracle.xml.classgen oracg コマンドライン・ユーティリティ	利用不可	xmlcg コマンドライン・ユーティリティ	利用不可
CGDocument CGNode ClassGenerator InvalidContentException DTD および XML Schema をサポートする	利用不可	-	利用不可
キャラクタ・セット（8）： EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	利用不可	キャラクタ・セット（8）： EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	利用不可
デフォルトのキャラクタ・セット： US-ASCII	利用不可	デフォルトのキャラクタ・セット： US-ASCII	利用不可

C

XDK for Java: 仕様および早見表

この付録では、XDK for Java の仕様、および Java 用の各 XML コンポーネントの早見表を示します。早見表には、各 XDK for Java コンポーネント用の主要な API、クラスおよび対応付けられたメソッドを示します。

この付録の内容は次のとおりです。

XML Parser for Java の早見表

表 C-1 および表 C-2 に、XML Parser for Java の最上位クラスをそれぞれ簡単に説明します。次の表には、他の XML Parser for Java クラスの概要を示します。

- 表 C-3「XML Parser for Java: DTD() クラス・メソッド」
- 表 C-4「XML Parser for Java: ElementDecl() クラス」
- 表 C-5「XML Parser for Java: NodeFactory() クラス」
- 表 C-6「XML Parser for Java: NSName() and NSResolver クラス」
- 表 C-7「XMLParser for Java: SAXAttrList() クラス」
- 表 C-8「XML Parser for Java: SAXParser() クラス」
- 表 C-9「XML Parser for Java: XMLParser() クラス」

注意： これらの表には、すべての XML Parser for Java メソッドが示されているわけではありません。詳細は、次のリソースを参照してください。

- 『Oracle9*i* XML リファレンス』
- http://otn.oracle.com/tech/xml
- doc/ 下にインストール済のソフトウェア

表 C-1 XML Parser for Java: oracle.xml.parser.v2 クラス

クラスのサマリー	説明
インタフェース	
NSName	要素名および属性名用の名前空間をサポートします。
NSResolver	名前空間の解決をサポートします。
XMLDocumentHandler	org.xml.sax.DocumentHandler インタフェースを拡張します。
XMLToken	XMLToken 用の基本インタフェースです。
クラス	
AttrDecl	DTD 内の属性リストに宣言されている各属性の情報を保持します。
Oracle.xml.parser.v2 パッケージ	XMLDocumentHandler インタフェースのデフォルトの動作を実装します。
DOMParser	W3C 勧告に準拠した、XML 1.0 のパーサーを実装します。

表 C-1 XML Parser for Java: oracle.xml.parser.v2 クラス（続き）

クラスのサマリー	説明	
DTD	DOM DocumentType インタフェースを実装し、XML 文書の DTD 情報を保持します。	
ElementDecl	DTD の要素宣言を表します。	
NodeFactory	解析中に構築される DOM ツリーの様々なノードを作成するメソッドを指定します。	
oraxml	XML ファイルを検証するためのコマンドライン・インタフェースを提供します。	
	java oraxml options* source	
	-h	このメッセージの出力
	-v	部分検証モード
	-s	厳密な検証モード
	-w	警告の表示
	-debug	デバッグ・モード
	-e < エラー・ログ >	エラーを書き込むファイル
oraxsl	複数の XML 文書で適用されるスタイルシートのコマンドライン・インタフェースを提供します。	
	java oraxsl options* source? stylesheet? result?	
	-w	警告の表示
	-e < エラー・ログ >	エラーを書き込むファイル
	-l <XML ファイル・リスト >	変換するファイルのリスト
	-d < ディレクトリ >	変換するファイルがあるディレクトリ
	-x < ソースの拡張子 >	排除する拡張子
	-i < ソースの拡張子 >	挿入する拡張子
	-s < スタイルシート >	使用するスタイルシート
	-r < 結果の拡張子 >	結果に使用する拡張子
	-o < 結果の拡張子 >	結果を保存するディレクトリ
	-p < パラメータ・リスト >	パラメータのリスト
	-t < スレッド数 >	使用するスレッド数
	-v	冗長モード

表 C-1 XML Parser for Java: oracle.xml.parser.v2 クラス（続き）

クラスのサマリー	説明
SAXAttrList	SAX AttributeList インタフェースを実装し、名前空間をサポートします。
SAXParser	W3C 勧告に準拠した、XML 1.0 の SAX パーサーを実装します。
XMLAttr	DOM Attr インタフェースを実装し、要素の各属性情報を保持します。
XMLCDATA	DOM CDATASection インタフェースを実装します。
XMLComment	DOM Comment インタフェースを実装します。
XMLDocument	DOM Document インタフェースを実装し、XML 文書全体を表します。また、DOM ツリーのルートとして機能します。
XMLDocumentFragment	DOM の DocumentFragment インタフェースを実装します。
XMLElement	DOM の Element インタフェースを実装します。
XMLEntityReference	-
XMLNode	DOM の Node インタフェースを実装し、DOM 全体のプライマリ・データ型として機能します。
XMLParser	DOMParser クラスおよび SAXParser クラスのベース・クラスとして機能します。
XMLPI	DOM の Processing Instruction インタフェースを実装します。
XMLText	DOM の Text インタフェースを実装します。
XMLTokenizer	W3C 勧告に準拠した、XML 1.0 のパーサーを実装します。
XSLProcessor	事前に作成された XSLStylesheet を使用して、入力 XML 文書を変換するメソッドを提供します。
XSLStylesheet	テンプレート、キー、変数、属性セットなどの XSL スタイルシートに関する情報を保持します。
例外	
XMLParseException	XML 文書の処理中に、解析例外が発生したことを示します。
XSLException	XSL 変換中に例外が発生したことを示します。

表 C-2 XML Parser for Java: DOMParser() メソッド

メソッド	説明
コンストラクタ	
DOMParser()	新しいパーサー・オブジェクトを作成します。W3C 勧告に準拠した XML 1.0 のパーサーを実装しており、XML 文書を解析して DOM ツリーを構築します。
メソッド	
getDoctype()	DTD を取得します。
getDocument()	ドキュメントを取得します。
parseDTD(InputSource,String)	指定した入力ソースから XML 外部 DTD を解析します。
parseDTD(InputStream,String)	指定した入力ストリームから XML 外部 DTD を解析します。
parseDTD(Reader,String)	指定した入力ストリームから XML 外部 DTD を解析します。
parseDTD(String,String)	指定した URL から XML 外部 DTD を解析します。
parseDTD(URL,String)	指定した URL が指す XML 外部 DTD を解析し、対応する XML 文書の階層を作成します。
setErrorStream(OutputStream)	エラーおよび警告を出力するための出力ストリームを設定します。
setErrorStream(OutputStream, String)	エラーおよび警告を出力するための出力ストリームを設定します。
setErrorStream(PrintWriter)	エラーおよび警告を出力するための出力ストリームを設定します。
setNodeFactory(NodeFactory)	ノード・ファクトリを設定します。
showWarnings(boolean)	警告を出力するかどうかを決定します。

表 C-3 XML Parser for Java: DTD() クラス・メソッド

DTD クラス・メンバー	説明
クラス	
DTD()	DOM DocumentType インタフェースを実装し、XML 文書の DTD 情報を保持します。
メソッド	
cloneNode(boolean)	このノードの複製を戻し、ノードの汎用コピー・コンストラクタとして機能します。
findElementDecl(String)	指定するタグ名に対する要素宣言を検索します。
findEntity(String,boolean)	DTD の名前付きエンティティを検出します。
findNotation(String)	DTD から名前表記法を取り出します。
getChildNodes()	このノードのすべての子を含むノード・リストを取得します。
getElementDecls()	DTD 内の要素宣言を含む名前付きノード・マップを取得します。
getEntities()	DTD で宣言されている外部および内部の汎用エンティティを含む名前付きノード・マップを取得します。
getName()	DTD の名前（DOCTYPE キーワードの直後の名前）を取得します。
getNotations()	DTD で宣言されている表記法を含む名前付きノード・マップを取得します。
getPublicId()	DTD に対応する公開識別子（指定されている場合）を取得します。
getSystemId()	DTD に対応するシステム識別子（指定されている場合）を取得します。
hasChildNodes()	ノードに子ノードがあるかどうかを判断します。
printExternalDTD(OutputStream)	このドキュメントの内容を特定の出力ストリームに書き込みます。
printExternalDTD(OutputStream,String)	外部 DTD の内容を特定の出力ストリームに書き込みます。
printExternalDTD(PrintWriter)	このドキュメントの内容を特定の出力ストリームに書き込みます。

表 C-4 XML Parser for Java: ElementDecl() クラス

ElementDecl() メンバーのサマリー	説明
クラス	
ElementDecl()	DTD の要素宣言を表します。
フィールド	
ANY	要素内容の型 - 子は、すべての要素になることができます。
ASTERISK	ContentModelParseTreeNode 型 - 「*」ノード（1 つの子を持ちます）
COMMA	ContentModelParseTreeNode 型 - 「,」ノード（2 つの子を持ちます）
ELEMENT	ContentModelParseTreeNode 型 - 「リーフ」・ノード（子を持ちません）
ELEMENTS	要素内容型 - 子は、内容モデルに従って要素になることができます。
EMPTY	要素内容型 - 子を持ちません。
MIXED	要素内容型 - 子は、内容モデルに従って PCDATA および要素になることができます。
OR	ContentModelParseTreeNode 型 - 「|」ノード（2 つの子を持ちます）
PLUS	ContentModelParseTreeNode 型 - 「+」ノード（1 つの子を持ちます）
QMARK	ContentModelParseTreeNode 型 - 「?」ノード（1 つの子を持ちます）
メソッド	
expectedElements(Element)	要素に追加できる要素名の一覧をベクトルで戻します。
findAttrDecl(String)	属性宣言のオブジェクトを取得します。オブジェクトが検出されない場合は NULL を取得します。
getAttrDecls()	列挙された属性宣言を取得します。
getContentElements()	この要素に追加できる要素のベクトルを戻します。
getContentType()	要素の内容モデルを戻します。
getParseTree()	内容モデルを解析したツリーのルート・ノードを戻します。
validateContent(Element)	要素ノードの内容を検証します。

表 C-5 XML Parser for Java: NodeFactory() クラス

NodeFactory() メンバーのサマリー	説明
コンストラクタ	
NodeFactory()	解析中に構築される DOM ツリーの様々なノードを作成するメソッドを指定します。アプリケーションは、これらのメソッドをオーバーライドして、解析中に DOM ツリーに追加する独自のカスタム・クラスを作成します。アプリケーションは、XMLParser の setNodeFactory() メソッドを使用して、独自のノード・ファクトリを登録する必要があります。これらのメソッドが NULL ポインタを戻した場合、ノードは DOM ツリーに追加されません。
メソッド	
createAttribute(String,String)	指定したタグおよびテキストを含む属性ノードを作成します。
createCDATASection(String)	指定したテキストを含む CDATA ノードを作成します。
createComment(String)	指定したテキストを含むコメント・ノードを作成します。
createDocument()	ドキュメント・ノードを作成します。
createElement(String)	指定したタグを含む要素ノードを作成します。
createProcessingInstruction(String,String)	指定したタグおよびテキストを含む処理命令ノードを作成します。
createTextNode(String)	指定したテキストを含むテキスト・ノードを作成します。

表 C-6 XML Parser for Java: NSName() and NSResolver クラス

メンバーのサマリー	説明
クラス	
NSName	要素名および属性名用の名前空間をサポートします。
メソッド	
getExpandedName()	この名前を完全に決定する名称を取得します。
getLocalName()	この名前のローカル名を取得します。
getNamespace()	この名前の解決済名前空間を取得します。
getPrefix()	この名前の接頭辞を取得します。
getQualifiedName()	修飾名を取得します。
クラス	
NSResolver	名前空間の解決をサポートします。
メソッド	
resolveNamespacePrefix(String)	与えられた名前空間の接頭辞の有効範囲内で、名前空間の定義を検索します。

表 C-7 XMLParser for Java: SAXAttrList() クラス

メンバーのサマリー	説明
クラス	
SAXAttrList()	SAX `AttributeList` インタフェースを実装し、名前空間をサポートします。名前空間のサポートが必要なアプリケーションは、Oracle のパーサー・クラスが戻すいずれかの属性リストを明示的に `SAXAttrList` に割り当てることによって、次のメソッドを使用できます。また、属性（SAX 2.0）インタフェースも実装します。 このインタフェースによって、次の 3 つの方法で属性リストにアクセスできます。 ■ 属性索引 ■ 名前空間の修飾名 ■ （接頭辞付き）修飾名 リストには、#IMPLIED と宣言されている属性が含まれます。ただし、開始タグに指定されている属性は含まれません。http://xml.org/sax/features/namespace-prefixes 機能を true に設定しないかぎり（デフォルトは false）、名前空間宣言（xmlns*）として使用される属性も含まれません。 名前空間の接頭辞の機能（前述を参照）が false の場合、修飾名を使用してアクセスできません。http://xml.org/sax/features/namespace-prefixes 機能が false の場合、名前空間の修飾名を使用してアクセスできません。 このインタフェースは、現在は使用できない SAX1 インタフェースに置き換わります。SAX1 インタフェースには、名前空間のサポートがありません。このインタフェースは、名前空間のサポートの他に、（次の）getIndex メソッドも追加します。リスト内の属性順序は指定されておらず、実装ごとに異なります。

表 C-7 XMLParser for Java: SAXAttrList() クラス（続き）

メンバーのサマリー	説明
メソッド	
getExpandedName(int)	リスト内の任意の位置の属性の拡張名を取得します。
getLength()	このリスト内の属性数を戻します。
getLocalName(int)	リスト内の任意の位置の属性のローカル名を取得します。
getName(int)	このリスト内の位置の属性名を戻します。
getNamespace(int)	リスト内の任意の位置の属性の解決済名前空間を取得します。
getPrefix(int)	リスト内の任意の位置の属性の名前空間接頭辞を取得します。
getQualifiedName(int)	リスト内の任意の位置の属性の修飾名を取得します。
getType(int index)	索引によって属性の型を検索します。属性の型は、「CDATA」、「ID」、「IDREF」、「IDREFS」、「NMTOKEN」、「NMTOKENS」、「ENTITY」「ENTITIES」または「NOTATION」の文字列（常に大文字）のいずれかです。 パーサーが属性宣言を読み取らない場合、またはパーサーが属性型をレポートしない場合は、XML 1.0 勧告（3.3.3「Attribute-Value Normalization」）に記述されているとおり、「CDATA」値を戻す必要があります。 表記法ではない列挙された属性については、パーサーが「NMTOKEN」として型をレポートします。
getType(java.lang.String qName)	XML 1.0 の修飾名によって属性の型を検索します。
getType(java.lang.String uri, java.lang.String localName)	名前空間名によって属性の型を検索します。
getType(String)	リスト内の任意の属性名の属性の型を戻します。
getValue(int)	リスト内の任意の位置の属性値を戻します。
getValue(String)	リスト内の任意の属性名の属性値を戻します。

表 C-8 XML Parser for Java: SAXParser() クラス

メンバーのサマリー	説明
コンストラクタ	
SAXParser()	新しいパーサー・オブジェクトを作成します。W3C 勧告に準拠した、XML 1.0 の SAX パーサーを実装します。アプリケーションは、SAX ハンドラを登録して、様々なパーサー・イベントの通知を受け取ることができます。XMLReader は、XML パーサーの SAX2 ドライバが実装する必要があるインタフェースです。このインタフェースによって、アプリケーションが、パーサーの機能およびプロパティを設定および問合せしたり、ドキュメント処理用のイベント・ハンドラを登録したり、ドキュメントの解析を開始することができます。 すべての SAX インタフェースが同期していると想定されます。解析が完了するまで、解析メソッドは戻されません。また、リーダーは、イベント・ハンドラのコールバックが戻るまで待ってから、次のイベントをレポートする必要があります。このインタフェースは、(現在は使用できない) SAX 1.0 パーサー・インタフェースに置き換わります。XMLReader インタフェースには、次のとおり、以前のパーサー・インタフェースにはない 2 つの重要な拡張機能が含まれています。 ■ 機能およびプロパティを問い合せたり、設定する標準の方法の追加 ■ 多くのより高レベルの XML 標準に必要な名前空間サポートの追加
メソッド	
setDocumentHandler(DocumentHandler)	SAX アプリケーションは、このメソッドを使用して、新しいドキュメントのイベント・ハンドラを登録します。
setDTDHandler(DTDHandler)	SAX アプリケーションは、このメソッドを使用して、新しい DTD のイベント・ハンドラを登録します。
setEntityResolver(EntityResolver)	SAX アプリケーションは、このメソッドを使用して、新しいエンティティ・リゾルバを登録します。
setErrorHandler(ErrorHandler)	SAX アプリケーションは、このメソッドを使用して、新しいエラーのイベント・ハンドラを登録します。

表 C-9 XML Parser for Java: XMLParser() クラス

メンバーのサマリー	説明
クラス	
XMLParser()	DOMParser クラスおよび SAXParser クラスのベース・クラスとして機能します。W3C 勧告に準拠した、XML 1.0 文書を解析するメソッドが含まれます。このクラスは、インスタンス化できません（アプリケーションは、要件に応じて DOM パーサーまたは SAX パーサーを使用します）。
メソッド	
getReleaseVersion()	Oracle XML Parser のバージョン番号を戻します。
getValidationMode()	検証モードを戻します。
parse(InputSource)	与えられた入力ソースから XML を解析します。
parse(InputStream)	与えられた入力ストリームから XML を解析します。
parse(Reader)	与えられた入力ストリームから XML を解析します。
parse(String)	指定した URL から XML を解析します。
parse(URL)	与えられた URL が指す XML 文書を解析し、それに対応する XML 文書の階層を作成します。
setBaseURL(URL)	外部エンティティおよび DTD をロードするためのベース URL を設定します。
setDoctype(DTD)	DTD を設定します。
setLocale(Locale)	アプリケーションは、これを使用して、エラー・レポート用のロケールを設定できます。
setPreserveWhitespace(boolean)	空白保持モードを設定します。
setValidationMode(boolean)	検証モードを設定します。

XML Parser for Java の入手方法

Oracle XML Parser は、Oracle Enterprise Edition および Oracle Standard Edition リリース 1（9.0.1）に付属しています。最新の XML パーサーは、http://otn.oracle.com/tech/xml/ からダウンロードすることもできます。

XML Parser for Java V2 のインストール

この項では、Windows NT および UNIX バージョンの XML Parser for Java V2 をインストールする方法を説明します。

XML Parser for Java V2: Windows NT 上でのインストール　Windows NT 上で Oracle XML Parser for Java V2 をインストールするには、次の手順に従います。

1. JDK 1.1.x. 以上および ZIP 形式ファイルの解凍ユーティリティ（Winzip など）をインストールします。
2. ZIP 形式の Oracle XML Parser をダウンロードします。
3. ディレクトリに xmlparser.zip を解凍します。次に例を示します。

   ```
   C:¥[ディレクトリ名 ]> unzip xmlparser.zip
   ```

4. 解凍すると、次のファイルおよびディレクトリが取得されます。
   * license.html - ライセンス契約のコピー
   * readme.html - リリース・ノートおよびインストレーション・ノート
   * doc¥ - ドキュメントのディレクトリ
   * lib¥ - パーサー・クラス・ファイルのディレクトリ
   * sample¥ - サンプル・コード・ファイル

XML Parser for Java V2: UNIX 上でのインストール　UNIX 上で Oracle XML Parser for Java V2 をインストールするには、次の手順に従います。

1. JDK 1.1.x 以上および GNU gzip をインストールします。
2. .tar.gz 形式の Oracle XML Parser をダウンロードします。
3. ディレクトリに配布パッケージを解凍します。次に例を示します。

   ```
   #gzip -dc xmlparser.tar.gz | tar xvf -
   ```

4. 解凍すると、次のファイルおよびディレクトリが取得されます。
   * license.html - ライセンス契約のコピー
   * readme.html - リリース・ノートおよびインストレーション・ノート
   * doc/ - ドキュメントのディレクトリ
   * lib/ - パーサー・クラス・ファイルのディレクトリ
   * sample/ - サンプル・コード・ファイル

サンプル・コード

サンプル・コードおよび XML パーサーの使用方法については、第 20 章「XML Parser for Java の使用」を参照してください。

XML Parser for Java V2 の仕様

Oracle XML Parser for Java V2 の仕様は、次のとおりです。

- 高パフォーマンスの新しいアーキテクチャを提供します。
- W3C の XSLT 1.0 勧告を統合サポートします。
- 検証および非検証モードをサポートします。
- 致命的エラーが発生するまで組込みエラー・リカバリを行います。
- DOM レベル 1.0 API および 2.0 API を統合します。
- SAX 1.0 API および 2.0 API を統合します。
- W3C 勧告の XML Namespace をサポートします。

要件

オペレーティング・システム : Java 1.1.x をサポートするオペレーティング・システム

JAVA: JDK 1.1.x 以上

Windows および UNIX バージョンの内容は同じです。これらは、オペレーティング・システムの互換性およびユーザーにとっての便宜上、異なる形態でアーカイブされています。

オンライン・ドキュメント

Oracle XML Parser for Java のドキュメントは、インストール場所の doc/ ディレクトリにあります。

リリース固有の注意事項

アーカイブのルート・ディレクトリにある readme.html ファイルには、不具合の修正や追加の API などのリリース固有の情報が含まれています。

Oracle XML Parser は、Java で作成されています。XML 文書が整形式であるか、また妥当であるかどうか（オプション）を確認します。Oracle XML Parser は、アクセス可能な Java オブジェクト・ツリーを構築します。Oracle XML Parser には、XML 文書を変換するための、統合された XSLT プロセッサも含まれています。

標準への準拠

XML パーサーは、次の W3C 勧告に準拠しています。

- XML 1.0
  （http://www.w3.org/TR/1998/REC-xml-19980210 を参照）
- XML Namespace
  （http://www.w3.org/TR/REC-xml-names/ を参照）
- DOM レベル 1 1.0
  （http://www.w3.org/TR/REC-DOM-Level-1/ を参照）
- DOM レベル 2
  （http://www.w3.org/TR/DOM-Level-2-Core/ を参照）
- XPath 1.0
  （http://www.w3.org/TR/1999/REC-xpath-19991116 を参照）
- XSLT 1.0
  （http://www.w3.org/TR/1999/REC-xslt-19991116 を参照）

XML パーサーは、次の W3C 勧告案にも準拠しています。

- XML Schema Part 1: Structures（構造）
  （http://www.w3.org/TR/xmlschema-1 を参照）
- XML Schema Part 2: Datatypes（データ型）
  （http://www.w3.org/TR/xmlschema-2 を参照）

さらに、XML パーサーは、XML 開発団体によって定義された次のインタフェースを実装しています。

- SAX 1.0 および 2.0
  （http://www.megginson.com/SAX/index.html を参照）

キャラクタ・セット・エンコーディングのサポート

XML Parser for Java は、次のエンコーディングをサポートしています。

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

デフォルト： デフォルトでは、UTF-8 がエンコーディングになります。JDK がサポートする他の ASCII ベースまたは EBCDIC ベースのエンコーディングも使用できます。ただし、IANA が定義する公式キャラクタ・セット名ではなく、JDK が要求する形式で指定する必要があります。

エラー・リカバリ

Oracle XML Parser は、エラー・リカバリも行います。ほとんどのエラーからのリカバリを行い、致命的なエラーが発生するまで処理を継続します。

Oracle XML Parser V1 および Oracle XML Parser V2

XML Parser for Java V2 では、XSLT プロセッサが組み込まれたことの他に、V1 から再構築されています。その結果、特に名前空間をサポートするクラス名が大幅に変更されています。V1 から V2 のコード変換で注意する必要がある変更の概要を、次に示します。

> **注意：** この概要は、Oracle XML Parser V1.0.14（V1）および V2.0.00（V2）に基づいています。

新しいクラス構造

oracle.xml.parser パッケージの名前は、oracle.xml.parser.v2 に変更されています。

新しいインタフェースは次のとおりです。

- NSName
- XMLDocumentHandler

次のインタフェースは削除されています。

- NSAttr
- NSAttributeList
- NSDocumentHandler
- NSElement

V2 の新しいクラスは次のとおりです。

- DOMParser
- DefaultXMLDocumentHandler
- SAXAttrList
- SAXParser
- XSLProcessor
- XSLStylesheet
- XSLException

表 C-10 に、再編成された XDK for Java クラスを示します。

表 C-10 XML Parser for Java: クラスの再編成および変更

V1	V2
クラスの再編成	
XMLParser	■ XMLParser（DOM および SAX のアクセスに適用可能なメソッドを含みます） ■ DOMParser（DOM アクセスに固有のメソッドを含みます） ■ SAXParser（SAX アクセスに固有のメソッドを含みます）
NSDocumentHandler	XMLDocumentHandler
NSAttr	XMLAttr（名前空間をサポートし、NSAttr インタフェースは削除されています）
NSAttributeList	SAXAttrList（NSAttributeList および org.xml.sax.AttributeList のメソッドを含みます）
NSElement	XMLElement（名前空間をサポートし、NSElelement インタフェースは削除されています）
パブリック・クラス / 変数 / コンストラクタ / メソッドの変更	
AttrDecl	
getName()	削除されています。XMLNode.getNodeName を使用してください。
getPresence()	getAttrPresence()
getType()	getAttrType()
getValues()	getEnumerationValues()
ElementDecl	
ASTERISK	将来の実装のために予約されています。
COMMA	将来の実装のために予約されています。
ELEMENT	将来の実装のために予約されています。
OR	将来の実装のために予約されています。
PLUS	将来の実装のために予約されています。

表 C-10 XML Parser for Java: クラスの再編成および変更（続き）

V1	V2
QMARK	将来の実装のために予約されています。
getParseTree()	将来の実装のために予約されています。
XMLAttr	新しいコンストラクタ - XMLAttr(String,String,String,String) 新しいメソッド: ■ cloneNode() ■ getPrefix()
XMLElement	新しいコンストラクタ - XMLElement(String,String,String) 新しいメソッド: ■ checkNamespace(String,String) ■ getElementsByTagName(String,String) ■ resolveNamespacePrefix(String)
XMLNode	新しいメソッド - transformNode()
XMLText	新しいメソッド - getNodeValue()

XDK for Java: XML Schema Processor

表 C-12 に、XML Schema Processor for Java のクラス、コンストラクタおよびメソッドを示します。

参照:

- 第 21 章「XML Schema Processor for Java の使用」を参照してください。
- インストール済のソフトウェアの doc/ ディレクトリにある readme.txt ファイルを参照してください。このソフトウェアは、http://otn.oracle.com/tech/xml からダウンロードすることもできます。

表 C-11 XML Schema Processor for Java: クラス、コンストラクタおよびメソッド

メンバー	説明
クラス	
XMLSchema	最上位の XMLSchema ドキュメントの宣言と定義、およびスキーマの場所とスキーマのターゲット名前空間を設定します。XMLSchema ドキュメントを処理した結果、XSDBuilder によって XMLSchema オブジェクトが作成されます。たとえば、XMLSchema オブジェクトは、XML 文書の検証のために XSDParser によって使用され、インポート済スキーマとして XSDBuilder によって使用されます。
コンストラクタ	
XMLSchema()	
XMLSchema(int)	
クラス	
XDSBuilder	XMLSchema ドキュメントから XMLSchema オブジェクトを構築します。XMLSchema オブジェクトは、最上位のスキーマ宣言と定義に対応した、オブジェクトのセット（情報セットの項目）です。スキーマ・ドキュメントは、解析され、DOM ツリーに変換される「XML」です。このスキーマの DOM ツリーは、次の順序で解析される「スキーマ」です。 最初に、（存在する場合）スキーマ・オブジェクトを構築し、参照可能にします。次に、（存在する場合）対応する DOM ツリーに置換します。次に、最上位の宣言と定義は、現行のスキーマ情報セットの項目として登録されます。最後に、ツリーの最上位要素（情報セットの項目）が、解析されたスキーマになります。XMLSchema 結果オブジェクトは、オブジェクト（最上位の入力要素）のセット（情報セット）です。オブジェクトの内容は、数の情報（min/maxOccurs）を含む、SNode 型のノード / オブジェクトの前の低レベル element/group decls/refs に対応するノードを持つツリーです。
メソッド	
build(InputStream,URL)	XMLSchema オブジェクトを構築します。
build(Reader,URL)	XMLSchema オブジェクトを構築します。
build(String)	XMLSchema オブジェクトを構築します。
build(String,String)	XMLSchema オブジェクトを構築します。
build(String,URL)	XMLSchema オブジェクトを構築します。
build(URL)	XMLSchema オブジェクトを構築します。

表 C-11 XML Schema Processor for Java: クラス、コンストラクタおよびメソッド（続き）

メンバー	説明
build(XMLDocument,URL)	XML 文書から XMLSchema を構築します。
getObject()	スキーマ・オブジェクトを戻します。
setError(XMLError)	XMLError オブジェクトを設定します。
setLocale(Locale)	エラー・レポートのロケールを設定します。
XSDException	XMLSchema 検証中に例外が発生したことを示します。
Methods getMessage()	エラー ID およびエラー・パラメータから、エラー・メッセージを作成するために、getMessage をオーバーライドします。
getMessage(XMLError)	パラメータとして送信される XMLError に基づいて、ローカライズされたメッセージを取得します。

XDK for Java: XML Class Generator for Java

Oracle XML Class Generator for Java には、Oracle XML Parser for Java が必要です。生成されたクラスによって出力された XML 文書は、W3C の XML 1.0 勧告に準拠します。Oracle XML Class Generator は、オプションで Java ソース・ファイルを検証できます。また、Oracle XML Class Generator は、オプションでソース・ファイル内に Javadoc コメントを生成します。

Oracle XML Class Generator は、XML 文書を出力するための次のエンコーディングをサポートしています。

UTF-8、UTF-16、ISO-10646-UCS-2、ISO-10646-UCS-4、US-ASCII、EBCDIC-CP-US、ISO-8859-1 および Shift_SJIS

デフォルトのエンコーディングは、ASCII です。JDK がサポートする他のすべての ASCII ベースまたは EBCDIC ベースのエンコーディングを使用できます。

XML Class Generator for Java のインストール

次の項では、Oracle XML Class Generator for Java のインストールについて説明します。

XML Class Generator for Java: Windows NT 上でのインストール

Windows NT 上で Oracle XML Class Generator をインストールするには、次の手順に従います。

1. JDK 1.1.x. 以上および ZIP 形式ファイルの解凍ユーティリティ（Winzip など）をインストールします。
2. ZIP 形式の Oracle XML Class Generator for Java をダウンロードします。

   http://otn.oracle.com/tech/xml/xdk_java/content.html
3. ディレクトリに xmlclassgenV1_0_0.zip を解凍します。次に例を示します。

   ```
   C:¥[ディレクトリ名 ]> unzip xmlclassgenV1_0_0.zip
   ```
4. 解凍すると、次のファイルおよびディレクトリが取得されます。
   - license.html - ライセンス契約のコピー
   - readme.html - リリース・ノートおよびインストレーション・ノート
   - doc¥ - ドキュメントのディレクトリ
   - lib¥ - classgen クラス・ファイルのディレクトリ
   - sample¥ - サンプル・コード・ファイル

XML Class Generator for Java: UNIX 上でのインストール

UNIX 上で Oracle XML Class Generator for Java をインストールするには、次の手順に従います。

1. JDK 1.1.x 以上および GNU gzip をインストールします。
2. .tar.gz 形式の Oracle XML Class Generator for Java をダウンロードします。

   http://otn.oracle.com/tech/xml/xdk_java/content.html
3. ディレクトリに配布パッケージを解凍します。次に例を示します。

   ```
   #gzip -dc xmlclassgenV1_0_0.tar.gz | tar xvf -
   ```
4. 解凍すると、次のファイルおよびディレクトリが取得されます。
   - license.html - ライセンス契約のコピー
   - readme.html - リリース・ノートおよびインストレーション・ノート
   - doc/ - ドキュメントのディレクトリ
   - lib/ - classgen クラス・ファイルのディレクトリ
   - sample/ - サンプル・コード・ファイル

XML Class Generator for Java の早見表

表 C-12 に、XML Class Generator for Java の主な API および最上位クラスをそれぞれ簡単に説明します。表 C-13 に、XML Class Generator for Java メソッドを示します。

表 C-12 XML Class Generator for Java: API およびクラス

クラス	説明
クラス	
CGDocument	Class Generator によって生成されたクラスのベース・ドキュメント・クラスとして機能します。DTD のルート要素に対するコンストラクタです。 パラメータ : `doctype` - DTD のルート要素の名前。 `dtd` - クラスの生成に使用される DTD。
CGNode	Class Generator によって生成されたノードのベース・クラスとして機能します。
DTDClassGenerator	Class Gnerator によって、DTD に対するクラスの生成に使用されます。
SchemaClassGenerator	Class Gnerator によって、Schema に対するクラスの生成に使用されます。
CGXSDElement	Schema Class Generator によって生成されたクラス用のベース・クラスです。このクラスは、スキーマの最上位要素に対応するクラスによって拡張されます。このクラスには、初期化、および検証に必要なスキーマ・ファイルからスキーマを構築するためのコードが含まれます。 **注意 :** 検証はサポートされていないため、スキーマ・ファイル読取り用の静的ブロックは使用できません。CLASSPATH の有効範囲内では、すべての場所からスキーマ・ファイルを読み取ることができます。getResource を使用して、スキーマ・ファイルを読み取る必要があります。
例外	
InvalidContentException	dtdcompiler クラスによって発生する InvalidContentException の定義です。

表 C-13 XML Class Generator for Java: メソッド

メソッド	説明
クラス	
CGDocument(String,DTD)	DTD のルート要素のコンストラクタです。パブリック抽象クラス CGDocument は、CGNode を拡張します。DTD コンパイラによって生成されたクラスのベース・ドキュメント・クラスとして機能します。
メソッド	
print(OutputStream)	作成した XML 文書を出力します。
print(OutputStream,String)	作成した XML 文書を出力します。
public abstract class CGNode	Object クラスを拡張します。XML Class Generator によって生成されたノードのベース・クラスとして機能します。
クラス	
generate(DTD,String)	doctype 要素をルートとして DTD を全検索し、Java クラスを生成します。
メソッド	
setGenerateComments(boolean)	Java ドキュメント・コメントを生成するかどうかを決定します。デフォルトは TRUE です。
setJavaPackage(String)	生成されたクラスのパッケージを設定します。デフォルトはパッケージなしです。
setOutputDirectory(String)	出力ディレクトリを設定します。デフォルトは現在のディレクトリです。
setSerializationMode(boolean)	DTD をシリアル化されたオブジェクトとして保存するか、またはテキスト・ファイルとして保存するかを決定します。
setValidationMode(boolean)	生成されたクラスが、構成中の XML 文書を検証する必要があるかどうかを決定します。デフォルトは TRUE です。
クラス	
CGNode(String)	DOM ツリーの要素に対するコンストラクタです。
メソッド	
addCDATASection(string)	要素に CDATA セクションを追加します。
addData(String)	要素に PCDATA を追加します。
addNode(CGNode)	要素にノードを子として追加します。
getCGDocument()	ベース・ドキュメント（ルート要素）を取得します。

表 C-13 XML Class Generator for Java: メソッド（続き）

メソッド	説明
getDTDNode()	ベース・ドキュメントから静的 DTD を取得します。
setAttribute(String,String)	属性値を設定します。
setDocument(CGDocument)	ベース・ドキュメント（ルート要素）を設定します。
storeID(String,String)	IDREF 値を後で検証できるように、ID 識別子の値を格納します。
storeIDREF(String,String)	対応する ID が定義されたかどうかを後で検証できるように、ID 識別子の値を格納します。
validateContent()	DTD で指定されたコンテンツ・モデルのとおりに、要素のコンテンツが有効かどうかを確認します。
validEntity(String)	ENTITY 識別子が有効かどうかを確認します。
validID(String)	ID 識別子が有効かどうかを確認します。
validNMTOKEN(String)	NMTOKEN 識別子が有効かどうかを確認します。
クラス	
CGXSDElement	Schema Class Generator によって生成された XML Schema に対応する、すべての生成されたクラスのベース・クラスとして機能します。
メソッド	
addAttribute(String,String)	ハッシュ表に任意のノードの属性を追加します。 パラメータ : `attName` - 属性名。 `attValue` - 属性値。
addElement(Object)	要素に対応するベクトルに、任意の要素ノードの要素を追加します。 パラメータ : `elem` - 追加する必要があるオブジェクト。
getAttributes()	public java.util.Hashtable getAttributes()。属性を戻します。 戻り値 : 属性名および属性値を含むハッシュ表の属性
getChildElements()	public java.util.Vector getChildElements()。すべてのローカル要素を持つベクトルを取得します。 戻り値 : elemChild ベクトル
getNodeValue()	public java.lang.String getNodeValue()。ノード値を戻します。
getType()	public java.lang.Object getType()。型を戻します。

表 C-13 XML Class Generator for Java: メソッド（続き）

メソッド	説明
print(XMLOutputStream)	public void print(oracle.xml.parser.v2.XMLOutputStream. out)。要素ノードを出力します。 パラメータ : `out` - 出力先のストリーム。
printAttributes(XMLOutputStream,String)	public void printAttributes(oracle.xml.parser.v2.XMLOutputStream out,java.lang.String name)。属性ノードを出力します。 パラメータ : `out` - 出力先のストリーム。 `name` - 属性名。
setNodeValue(String)	protected void setNodeValue(java.lang.String value)。要素のノード値を設定します。 パラメータ : `value` - ノード値。

oracg コマンドライン・ユーティリティ

`oracg` は、指定された入力引数に応じて DTD または Schema Class Generator for Java を起動し、それぞれ DTD および Schema に基づいたクラスを生成します。表 C-14 に、oracg の引数を示します。

表 C-14 oracg コマンドライン・ユーティリティ

oracg の引数	説明
-h	ヘルプ・メッセージ・テキストの出力
- d<DTD ファイル >	DTD ファイル（.dtd ファイル）
- s <Schema ファイル >	Schema ファイル（.xsd ファイル）
- o < 出力ディレクトリ名 >	出力ディレクトリ
- c	コメント・オプション
- p < パッケージ名 >	名前空間に対応するパッケージ名

XDK for Java: XSQL Servlet

XSQL Servlet のダウンロードおよびインストール

OTN からの XSQL Servlet のダウンロード

XSQL Servlet 配布パッケージは、次の Web サイトからダウンロードできます。

http://otn.oracle.com/tech/xml/xsak_java

1. ページの上部にある「Software」アイコンをクリックします。
2. OTN ユーザー名およびパスワードでログインします（アカウントがない場合、無償で登録できます）。
3. Windows NT 用をダウンロードするか UNIX 用をダウンロードするかを選択します（両方ともファイルは同じです）。
4. ライセンス契約とサーベイを承諾します。
5. xsqlservlet_v1.0.2.0.tar.gz または xsqlservlet_v1.0.2.0.zip をクリックします。

ダウンロードしたファイルの解凍

XSQL Servlet 配布パッケージの内容を解凍するには、次の手順に従います。

1. .¥xsql ディレクトリおよびそのサブディレクトリを格納するディレクトリを選択（C:¥ など）します。
2. ディレクトリを C:¥ に変更し、ダウンロードした XSQL のアーカイブ・ファイルを解凍します。次に例を示します。

   UNIX の場合 :

   ```
   tar xvfz xsqlservlet_v1.0.2.0.tar.gz
   ```

   Windows NT の場合 :

   ```
   pkzip25 -extract -directories xsqlservlet_v1.0.2.0.zip
   ```

   pkzip25 コマンドライン・ツールまたは WinZip（ビジュアル・アーカイブ解凍ツール）を使用します。

Windows NT: Web-to-go Server の使用

XSQL Servlet には、Oracle Web-to-go Server がバンドルされています。Oracle Web-to-go Server は、XSQL Pages を使用できるように事前構成されています。Oracle Web-to-go Server はシングル・ユーザー・サーバーで、Servlet 2.1 API をサポートし、モバイル・アプリケーションの配置および開発に使用されます。XSQL Pages を実行する目的で他のサーブレット・エンジンの構成の詳細を調べる前に、Oracle Web-to-go Server を使用して Windows マシン上で XSQL Pages を試行することをお薦めします。

> **注意：** Oracle Web-to-go Server は、モバイル・アプリケーション用の Oracle の開発および配置プラットフォームの一部です。Web-to-go の詳細は、http://www.oracle.com/mobile を参照してください。

次の手順に従うと、Windows NT のユーザーはすぐに XSQL Pages を使用できます。

1. .¥xsql ディレクトリでスクリプト xsql-wtg.bat を実行します。
2. http://localhost:7070/xsql/index.html をブラウズします。

このスクリプトの起動時にエラーが発生する場合、xsql-wtg.bat ファイルを編集して、2 つの環境変数 JAVA および XSQL_HOME を、使用中のマシンに適切な値に設定します。

```
REM ------------------------------------------------
REM Set the 'JAVA' variable equal to the full path
REM of your Java executable.
REM ------------------------------------------------
set JAVA=J:¥java1.2¥jre¥bin¥java.exe
set XSQL_HOME=C:¥xsql
REM ------------------------------------------------
REM Set the 'XSQL_HOME' variable equal to the full
REM path of where you install the XSQL Servlet
REM distribution.
REM ------------------------------------------------
```

編集後、前述の 2 つの手順を繰り返します。

デモの実行時にデータベース接続エラーが発生する場合、次の項を参照して、XSQLConfig.xml ファイルのデータベース接続情報を正しく設定した後で、再度前述の手順を実行してください。

環境用のデータベース接続定義の設定

デモは、ローカル・マシン（Web サーバーを実行中のマシン）上で、データベースの SCOTT スキーマを使用するように設定されます。ローカル・データベースの実行中で、SCOTT アカウント（パスワードは TIGER）がある場合は、準備が完了しています。そうでない場合、.¥xsql¥lib¥XSQLConfig.xml ファイルを編集して、「username」、「password」および「dburl」に適切な値を設定し、「demo」接続に適切なドライバ値を設定する必要があります。

```
<?xml version="1.0" ?>
    <XSQLConfig>
         :
      <connectiondefs>
        <connection name="demo">
          <username>scott</username>
          <password>tiger</password>
          <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
          <driver>oracle.jdbc.driver.OracleDriver</driver>
        </connection>
        <connection name="lite">
          <username>system</username>
          <password>manager</password>
          <dburl>jdbc:Polite:POlite</dburl>
          <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
        </connection>
      </connectiondefs>
         :
</XSQLConfig>
```

UNIX: XSQL Page 実行のためのサーブレット・エンジンの設定

Web サーバーに XSQL Servlet をインストールする必要がある UNIX ユーザーを含むすべてのユーザーは、使用する Web サーバーに基づいて、次の手順を実行する必要があります。いずれの場合でも、3 つの基本手順があります。

1. サーバー CLASSPATH に XSQLConfig.xml が常駐するディレクトリ（デフォルトでは、./xsql/lib）の他に、XSQL Java アーカイブのリストも含めます。

   注意： 便宜上、xsqlservlet_v1.0.2.0.tar.gz および xsqlservlet_v1.0.2.0.zip 配布パッケージは、Oracle XSQL Pages 独自の .jar アーカイブとともに、Oracle XML Parser for Java V2、Oracle XSU for Java および Oracle9*i* リリース 1（9.0.1）の JDBC ドライバ用の .jar ファイルを .\lib サブディレクトリに含んでいます。

2. .xsql ファイルの拡張を、oracle.xml.xsql.XSQLServlet サーブレット・クラスにマッピングします。
3. XSQL ファイルを解凍したディレクトリに、仮想ディレクトリ /xsql をマッピングします（オンライン・ヘルプおよびデモへのアクセス用）。

XSQL Servlet の仕様

XSQL Servlet の仕様は次のとおりです。

- 1 つ以上の SQL 問合せに基づいて、動的 XML 文書を生成します。
- XSLT を使用して、サーバーまたはクライアント側の、結果として生成される XML 文書を変換します（オプション）。
- W3C の XML 1.0 勧告をサポートします。
- DOM レベル 1.0 API および 2.0 API をサポートします。
- W3C の XSLT 1.0 勧告をサポートします。
- W3C の XML Namespace 勧告をサポートします。

キャラクタ・セットのサポート

XSQL Servlet は、次のキャラクタ・セット・エンコーディングをサポートします。

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

XDK for Java: XSQL Servlet の早見表

表 C-15 および表 C-16 に、XSQL Servlet の API および最上位クラスをそれぞれ簡単に説明します。

表 C-15 XSQL Servlet クラス

クラスのサマリー	説明
インタフェース	
XSQLActionHandler	すべての XSQL Action Element Handler が実装する必要があるインタフェースです。
XSQLPageRequest	XSQL Pages に対する要求を表すインタフェースです。
クラス	
XSQLActionHandlerImpl	ユーザー独自のカスタム・ハンドラを作成するために拡張可能な XSQLActionHandler の基本実装です。
XSQLCommandLine	XSQL ページを処理するコマンドライン・ユーティリティです。
XSQLPageRequestImpl	新しい種類のページ要求実装の導出に使用可能な、XSQLPageRequest インタフェースの基本的な実装です。
XSQLParserHelper	一般的な XML 解析ルーチンです。
XSQLRequest	XSQL ページの要求のプログラム処理
XSQLServlet	XSQL Pages に対して HTTP の GET メソッドおよび POST メソッドを使用可能にするサーブレットです。
XSQLServletPageRequest	サーブレット・ベースの XSQL ページ要求のための XSQLPageRequest の実装です。
XSQLStylesheetProcessor	XSLT スタイルシート処理エンジンです。

表 C-16 XSQLPageRequest インタフェース・メソッド

メソッド	説明
createNestedRequest(URL,Dictionary)	ネストした要求のインスタンスを戻します。
getConnectionName()	この要求に使用されている接続の名前を戻します。接続が設定または使用されてない場合、NULL になる場合があります。
getErrorWriter()	この要求を処理するエラーを出力するための PrintWriter を戻します。
getJDBCConnection()	この要求に対して JDBC 接続を使用中にします。（NULL になる可能性があります。）
getPageEncoding()	この要求に対応付けられたソース XSQL ページのエンコーディングを戻します。
getParameter(String)	要求されたパラメータ値を戻します。
getPostedDocument()	この要求に対してポストされた XML の内容を、XML 文書として戻します。
getRequestParamsAsXMLDocument()	要求パラメータの内容を、XML 文書として戻します。
getRequestType()	作成しているページ要求のタイプを識別する文字列を戻します。
getSourceDocumentURI()	要求されたドキュメントの URI の文字列表現を戻します。
getStylesheetParameter(String)	スタイルシート・パラメータを名前で取得します。
getStylesheetParameters()	スタイルシート・パラメータの名前の列挙を取得します。
getStylesheetURI()	結果処理に使用されるスタイルシートの URI を戻します。
getUserAgent()	要求しているプログラムの文字列識別子を戻します。
getWriter()	ページ要求の結果の出力に使用される PrintWriter を戻します。
getXSQLConnection()	この要求に使用されている XSQLConnection オブジェクトを取得します。NULL になる場合があります。
isIncludedRequest()	この要求が別の要求に含まれている場合、TRUE を戻します。

表 C-16 XSQLPageRequest インタフェース・メソッド（続き）

メソッド	説明
isOracleDriver()	現行の接続が Oracle JDBC ドライバを使用している場合、TRUE を戻します。
printedErrorHeader()	Error Header が出力されたかどうかを戻します。
requestProcessed()	ページ要求が要求終了処理を実行できるようにします。
setConnectionName(String)	この要求に使用する接続名を設定します。
setContentType(String)	結果ページの Content Type を設定します。
setIncludingRequest(XSQLPageRequest)	この要求に対して Including Page Request オブジェクトを設定します。
setPageEncoding(String)	この要求に対応付けられたソース XSQL ページのエンコーディングを設定します。
setPageParam(String,String)	動的ページ・パラメータ値を設定します。
setPostedDocument(Document)	ポストされたドキュメントのプログラムを設定できるようにします。
setPrintedErrorHeader(boolean)	Error Header が出力されたかどうかを設定します。
setStylesheetParameter(String,String)	対応付けられたスタイルシートへ渡されるパラメータ値を設定します。
setStylesheetURI(String)	結果処理に使用されるスタイルシートの URI を設定します。
translateURL(String)	この要求のベース URI に対して相対 URI に変換された絶対 URL を表す文字列を戻します。
useConnectionPooling()	この要求に対して接続プーリングが必要とされる場合、TRUE を戻します。
useHTMLErrors()	この要求に対して HTML 形式のエラー・メッセージが必要とされる場合、TRUE を戻します。

XSU for Java の早見表

付録 H「XSU の仕様および早見表」を参照してください。

D

XDK for JavaBeans: 仕様および早見表

この付録では、XDK for JavaBeans の仕様および早見表について説明します。

この付録の内容は次のとおりです。

XDK for JavaBeans: Transviewer Bean の早見表

次の表に、XDK for JavaBeans に含まれる Transviewer Bean の主なクラスおよびメソッドを示します。

- 表 D-1「DOMBuilder Bean のクラスおよびメソッド」
- 表 D-2「XSLTransformer Bean のクラスおよびメソッド」
- 表 D-3「XMLTreeView Bean のクラスおよびメソッド」
- 表 D-4「XMLTransformPanel Bean のクラスおよびメソッド」
- 表 D-5「DBViewer Bean のクラスおよびメソッド」
- 表 D-6「XMLSourceView Bean のクラスおよびメソッド」
- 表 D-7「DBAccess Bean のクラスおよびメソッド」

DOMBuilder Bean の早見表

表 D-1 に、DOMBuilder Bean の主なクラスおよびメソッドを示します。

表 D-1 DOMBuilder Bean のクラスおよびメソッド

クラスのサマリー	説明
インタフェース	
DOMBuilderErrorListener	解析中にエラーが検出された場合に通知を受け取るためには、このインタフェースを実装する必要があります。
DOMBuilderListener	非同期解析中のイベントに関する通知を受け取るためには、このインタフェースを実装する必要があります。
クラス	
DOMBuilder	XML 1.0 のパーサーをカプセル化して、XML 文書を解析し、DOM ツリーを構築します。解析は別々のスレッドで行われ、ツリーが構築されたときの通知には DOMBuilderListener インタフェースが使用される必要があります。
DOMBuilderBeanInfo	-
DOMBuilderErrorEvent	解析例外の発生時に送信されるエラー・イベントを定義します。
DOMBuilderEvent	DOMBuilder が、すべての登録リスナーに解析イベントについて通知するために使用するイベント・オブジェクトです。
ResourceManager	固定数の論理リソースへのアクセスを維持する単純セマフォです。

XSLTransformer Bean の早見表

表 D-2 に、XSLTransformer Bean のクラスおよびメソッドを示します。

表 D-2 XSLTransformer Bean のクラスおよびメソッド

メンバーのサマリー	説明
クラス	
XSLTransformer	バックグラウンド・スレッドで XSL 変換を適用します。
コンストラクタ	
XSLTransformer()	XSLTransformer コンストラクタです。
XSLTransformer(int)	XSLTransformer コンストラクタです。
メソッド	
addXSLTransformerErrorListener(XSLTransformerErrorListener)	エラー・イベント・リスナーを追加します。
addXSLTransformerListener(XSLTransformerListener)	リスナーを追加します。
	一意の XSLTransformer の ID を戻します。
getResult()	結果ドキュメントのドキュメント・フラグメントを戻します。
processXSL(XSLStylesheet, InputStream, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, Reader, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, URL, URL)	バックグラウンドで XSL 変換を開始します。
processXSL(XSLStylesheet, XMLDocument, OutputStream)	バックグラウンドで XSL 変換を開始します。
removeDOMTransformerErrorListener(XSLTransformerErrorListener)	エラー・イベント・リスナーを削除します。
removeXSLTransformerListener(XSLTransformerListener)	リスナーを削除します。
run()	XSL プロセッサが使用するエラー・ストリームを設定します。
showWarnings(boolean)	XSL プロセッサが使用する showWarnings フラグを設定します。

表 D-2 XSLTransformer Bean のクラスおよびメソッド（続き）

メンバーのサマリー	説明
クラス	
XSLTransformerErrorEvent	XSLTransformer が、すべての登録リスナーに変換エラー・イベントについて通知するために使用するエラー・イベント・オブジェクトです。
XSLTransformerEvent	XSLTransformer が、すべての登録リスナーに変換イベントについて通知するために使用するイベント・オブジェクトです。
XSLTransformerErrorListener	非同期変換中のエラー・イベントに関する通知を受け取るためには、このインタフェースを実装する必要があります。
XSLTransformerListener	非同期変換中のイベントに関する通知を受け取るためには、このインタフェースを実装する必要があります。

XMLTreeView Bean の早見表

表 D-3 に、XMLTreeView Bean の主なクラスおよびメソッドを示します。

表 D-3 XMLTreeView Bean のクラスおよびメソッド

クラスまたはメソッド	説明
クラス	
XMLTreeView()	クラス・コンストラクタです。
メソッド	
getPreferredSize()	XMLTreeView の推奨サイズを戻します。
setXMLDocument(Document)	XMLTreeViewer を XML 文書に対応付けます。
updateUI()	XMLTreeView に強制的に UI を更新 / リフレッシュさせます。
クラス	
XMLTreeViewBeanInfo()	
メソッド	
getIcon(int)	
getPropertyDescriptors()	

XMLTransformPanel の早見表

表 D-4 に、XMLTransformPanel Bean の主なクラスおよびメソッドを示します。

表 D-4 XMLTransformPanel Bean のクラスおよびメソッド

クラスおよびメソッド	説明
クラス	
XMLTransformPanel	XMLTransformPanel はビジュアル的な Bean であり、XSL 変換を XML 文書に適用します。この Bean は結果をビジュアル化し、入力の XML 文書やファイルおよび XSL ドキュメントやファイルの編集を可能にします。
コンストラクタ	
XMLTransformPanel()	public XMLTransformPanel() クラス・コンストラクタです。XMLTransformPanel 型のオブジェクトを作成します。
XMLTransformPanelBeanInfo	-
コンストラクタ	
XMLTransformPanelBeanInfo()	public XMLTransformPanelBeanInfo()
メソッド	
getIcon(int)	public java.awt.Image getIcon(int iconKind) オーバーライド先 : java.beans.SimpleBeanInfo クラスの java.beans.SimpleBeanInfo.getIcon(int) をオーバーライドします。
getPropertyDescriptors()	public java.beans.PropertyDescriptor[] getPropertyDescriptors() オーバーライド先 : java.beans.SimpleBeanInfo クラスの java.beans.SimpleBeanInfo.getPropertyDescriptors() をオーバーライドします。

DBViewer Bean の早見表

表 D-5 に、DBViewer Bean の主なクラスおよびメソッドを示します。

表 D-5 DBViewer Bean のクラスおよびメソッド

クラスおよびメソッド	説明
クラス	
DBViewer()	新しいインスタンスを作成します。XSL スタイルシートを適用し、スクロール可能な swing パネルに結果の HTML をビジュアル化することによって、データベース問合せまたは任意の XML を表示します。この Bean には、XML バッファ、XSL バッファおよび結果バッファの 3 つのバッファがあります。DBViewer Bean を使用すると、プログラムをコールして、様々なソースからバッファをロードまたは保存したり、XSL バッファ内のスタイルシートを使用して、XML バッファにスタイルシート変換を適用することができます。結果は、結果バッファに格納できます。XML バッファおよび XSL バッファの内容は、ソースまたはツリー構造として表示できます。結果バッファの内容は、HTML にレンダリングして、ソースまたはツリー構造で表示することもできます。XML バッファは、データベースの問合せからロードできます。すべてのバッファには、Oracle データベース内の CLOB 表のファイル、およびファイル・システムのファイルをロードして保存できます。そのため、ファイル・システムとデータベース内のユーザー・スキーマ間のファイルの移動を制御できます。
メソッド	
getHostname()	データベースのホスト名を取得します。
getInstancename()	データベースのインスタンス名を取得します。
getPassword()	ユーザー・パスワードを取得します。
getPort()	データベースのポート番号を取得します。
getResBuffer()	結果バッファの内容を取得します。
getResCLOBFileName()	結果の CLOB ファイル名を取得します。
getResCLOBTableName()	結果の CLOB 表名を取得します。
getResFileName()	結果ファイル名を取得します。

表 D-5 DBViewer Bean のクラスおよびメソッド （続き）

クラスおよびメソッド	説明
getUsername()	ユーザー名を取得します。
getXmlBuffer()	XML バッファの内容を取得します。
getXmlCLOBFileName()	XML の CLOB ファイル名を取得します。
getXmlCLOBTableName()	XML の CLOB 表名を取得します。
getXmlFileName()	XML ファイル名を取得します。
getXMLStringFromSQL(String)	SQL 問合せから結果セットの XML 表示を取得します。
getXslBuffer()	XSL バッファの内容を取得します。
getXslCLOBFileName()	XSL の CLOB ファイル名を取得します。
getXslCLOBTableName()	XSL の CLOB 表名を取得します。
getXslFileName()	XSL ファイル名を取得します。
loadResBuffer(String)	ファイルから結果バッファをロードします。
loadResBuffer(String, String)	CLOB ファイルから結果バッファをロードします。
loadResBufferFromClob()	CLOB ファイルから結果バッファをロードします。
loadResBufferFromFile()	ファイルから結果バッファをロードします。
loadXmlBuffer(String)	ファイルから XML バッファをロードします。
loadXmlBuffer(String, String)	CLOB ファイルから XML バッファをロードします。
loadXmlBufferFromClob()	CLOB ファイルから XML バッファをロードします。
loadXmlBufferFromFile()	ファイルから XML バッファをロードします。
loadXMLBufferFromSQL(String)	SQL 結果セットから XML バッファをロードします。
loadXslBuffer(String)	ファイルから XSL バッファをロードします。
loadXslBuffer(String, String)	CLOB ファイルから XSL バッファをロードします。
loadXslBufferFromClob()	CLOB ファイルから XSL バッファをロードします。
loadXslBufferFromFile()	ファイルから XSL バッファをロードします。

表 D-5 DBViewer Bean のクラスおよびメソッド （続き）

クラスおよびメソッド	説明
parseResBuffer()	結果バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
parseXmlBuffer()	XML バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
parseXslBuffer()	XSL バッファを解析して、ツリー・ビューおよびソース・ビューをリフレッシュします。
saveResBuffer(String)	結果バッファをファイルに保存します。
saveResBuffer(String, String)	結果バッファを CLOB ファイルに保存します。
saveResBufferToClob()	結果バッファを CLOB ファイルに保存します。
saveResBufferToFile()	結果バッファをファイルに保存します。
saveXmlBuffer(String)	XML バッファをファイルに保存します。
saveXmlBuffer(String, String)	XML バッファを CLOB ファイルに保存します。
saveXmlBufferToClob()	XML バッファを CLOB ファイルに保存します。
saveXmlBufferToFile()	XML バッファをファイルに保存します。
saveXslBuffer(String)	XSL バッファをファイルに保存します。
saveXslBuffer(String, String)	XSL バッファを CLOB ファイルに保存します。
saveXslBufferToClob()	XSL バッファを CLOB ファイルに保存します。
saveXslBufferToFile()	XSL バッファをファイルに保存します。
setHostname(String)	データベースのホスト名を設定します。
setInstancename(String)	データベースのインスタンス名を設定します。
setPassword(String)	ユーザー・パスワードを設定します。
setPort(String)	データベースのポート番号を設定します。
setResBuffer(String)	結果バッファに新しいテキストを設定します。
setResCLOBFileName(String)	結果の CLOB ファイル名を設定します。
setResCLOBTableName(String)	結果の CLOB 表名を設定します。
setResFileName(String)	結果ファイル名を設定します。

表 D-5 DBViewer Bean のクラスおよびメソッド （続き）

クラスおよびメソッド	説明
setResHtmlView(boolean)	結果バッファをレンダリング済 HTML として表示します。
setResSourceEditView(boolean)	結果バッファを XML ソースとして表示し、編集モードに入ります。
setResSourceView(boolean)	結果バッファを XML ソースとして表示します。
setResTreeView(boolean)	結果バッファを XML ツリー・ビューとして表示します。
setUsername(String)	ユーザー名を設定します。
setXmlBuffer(String)	XML バッファに新しいテキストを設定します。
setXmlCLOBFileName(String)	XML の CLOB ファイル名を設定します。
setXmlCLOBTableName(String)	XML の CLOB 表名を設定します。
setXmlFileName(String)	XML ファイル名を設定します。
setXmlSourceEditView(boolean)	XML バッファを XML ソースとして表示し、編集モードに入ります。
setXmlSourceView(boolean)	XML バッファを XML ソースとして表示します。
setXmlTreeView(boolean)	XML バッファをツリーとして表示します。
setXslBuffer(String)	XSL バッファに新しいテキストを設定します。
setXslCLOBFileName(String)	XSL の CLOB ファイル名を設定します。
setXslCLOBTableName(String)	XSL の CLOB 表名を設定します。
setXslFileName(String)	XSL ファイル名を設定します。
setXslSourceEditView(boolean)	XSL バッファを XML ソースとして表示し、編集モードに入ります。
setXslSourceView(boolean)	XSL バッファを XML ソースとして表示します。
setXslTreeView(boolean)	XSL バッファをツリーとして表示します。
transformToDoc()	XSL バッファのスタイルシートを適用して、XML バッファの内容を変換します。
transformToRes()	XML バッファ内の XML に XSL バッファのスタイルシート変換を適用し、その結果を結果バッファに格納します。
transformToString()	XSL バッファのスタイルシートを適用して、XML バッファの内容を変換します。

XMLSourceView Bean の早見表

表 D-6 に、XMLSourceView Bean の主なクラスおよびメソッドを示します。

表 D-6 XMLSourceView Bean のクラスおよびメソッド

クラスおよびメソッド	説明
クラス	
XMLSourceView	XML 文書を表示します。タグ、属性名、属性値、コメント、CDATA、PCDATA、PI（処理命令）データ、PI（処理命令）名および表記法の XML トークン型を認識します。各トークン型には、フォアグラウンド・カラーおよびフォアグラウンド・フォントがあります。デフォルトのカラーおよびフォントの設定は、ユーザーが変更できます。この Bean は、入力として org.w3c.dom.Document オブジェクトを取ります。
フィールド	
inputDOMDocument、jScrollPane、jTextPane、xmlStyledDocument	-
コンストラクタ	
XMLSourceView()	クラス・コンストラクタです。XMLSourceView 型のオブジェクトを作成します。
メソッド	
fontGet(AttributeSet)	指定された属性セットからフォントを取得して戻します。 パラメータ : attributeset - ソースの属性セット。 戻り値 : フォント・オブジェクト
fontSet(MutableAttributeSet, Font)	属性セットのフォントを設定します。 パラメータ : mutableattributeset - 更新する属性セット。 font - 属性セット用の新しいフォント。
getAttributeNameFont()	属性名のフォントを戻します。 戻り値 : フォント・オブジェクト。

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
getAttributeNameForeground()	属性名のフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getAttributeValueFont()	属性値のフォントを戻します。 戻り値 : フォント・オブジェクト。
getAttributeValueForeground()	public java.awt.Color getAttributeValueForeground() 属性値のフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getBackground()	public java.awt.Color getBackground() バックグラウンド・カラーを戻します。 オーバーライド先 : java.awt.Component クラスの java.awt.Component.getBackground() をオーバーライドします。 戻り値 : カラー・オブジェクト。
getCDATAFont()	public java.awt.Font getCDATAFont() CDATA のフォントを戻します。 戻り値 : フォント・オブジェクト。
getCDATAForeground()	public java.awt.Color getCDATAForeground() CDATA のフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getCommentDataFont()	public java.awt.Font getCommentDataFont() コメントのフォントを戻します。 戻り値 : フォント・オブジェクト。
getCommentDataForeground()	public java.awt.Color getCommentDataForeground() コメントのフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
getEditedText()	public java.lang.String getEditedText() 編集済テキストを戻します。 戻り値 : 編集済テキストを含む文字列オブジェクト。
getJTextPane()	public javax.swing.JTextPane getJTextPane() JTextPane ビューアのコンポーネントを戻します。 戻り値 : XMLSourceViewer が使用する JTextPane オブジェクト。
getMinimumSize()	public java.awt.Dimension getMinimumSize() XMLSourceView の最小サイズを戻します。 オーバーライド先 : javax.swing.JComponent クラスの javax.swing.JComponent.getMinimumSize() をオーバーライドします。 戻り値 : XMLSourceView の最小サイズを含むディメンション・オブジェクト。
getNodeAtOffset(int)	public org.w3c.dom.Node getNodeAtOffset(int i) 指定されたオフセットにある XML ノードを戻します。 パラメータ : i - ノード・オフセット。 戻り値 : オフセット i からのノード・オブジェクト。
getPCDATAFont()	public java.awt.Font getPCDATAFont() PCDATA のフォントを戻します。 戻り値 : フォント・オブジェクト。
getPCDATAForeground()	public java.awt.Color getPCDATAForeground() PCDATA のフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getPIDataFont()	public java.awt.Font getPIDataFont() PI データのフォントを戻します。 戻り値 : フォント・オブジェクト。

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
getPIDataForeground()	public java.awt.Color getPIDataForeground() PI データのフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getPINameFont()	public java.awt.Font getPINameFont() PI 名のフォントを戻します。 戻り値 : フォント・オブジェクト。
getPINameForeground()	public java.awt.Color getPINameForeground() PI データのフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getSymbolFont()	public java.awt.Font getSymbolFont() 表記法のフォントを戻します。 戻り値 : フォント・オブジェクト。
getSymbolForeground()	public java.awt.Color getSymbolForeground() 表記法のフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getTagFont()	public java.awt.Font getTagFont() タグのフォントを戻します。 戻り値 : フォント・オブジェクト。
getTagForeground()	public java.awt.Color getTagForeground() タグのフォアグラウンド・カラーを戻します。 戻り値 : カラー・オブジェクト。
getText()	public java.lang.String getText() XML 文書を文字列として戻します。 戻り値 : XML 文書を含む文字列オブジェクト。

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
isEditable()	public boolean isEditable() このオブジェクトが編集可能かどうかを示すブールを戻します。
selectNodeAt(int)	public void selectNodeAt(int i) オフセット i にある XML ノードにカーソルを移動します。 パラメータ : i - ノード・オフセット。
setAttributeNameFont(Font)	public void setAttributeNameFont(java.awt.Font font) 属性名のフォントを設定します。 パラメータ : font - 属性名用の新しいフォント。
setAttributeNameForeground (Color)	public void setAttributeNameForeground(java.awt.Color color) 属性名のフォアグラウンド・カラーを設定します。 パラメータ : color - 属性名用の新しいカラー。
setAttributeValueFont(Font)	public void setAttributeValueFont(java.awt.Font font) 属性値のフォントを設定します。 パラメータ : font - 属性名用の新しいフォント。
setAttributeValueForeground (Color)	public void setAttributeValueForeground(java.awt.Color color) 属性値のフォアグラウンド・カラーを設定します。 パラメータ : color - 属性値用の新しいカラー。
setBackground(Color)	public void setBackground(java.awt.Color color) バックグラウンド・カラーを設定します。 オーバーライド先 : javax.swing.JComponent クラスの javax.swing.JComponent.setBackground(java.awt.Color) をオーバーライドします。 パラメータ : color - 新しいバックグラウンド・カラー。
setCDATAFont(Font)	public void setCDATAFont(java.awt.Font font) CDATA のフォントを設定します。 パラメータ : font - CDATA 用の新しいフォント。

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
setCDATAForeground(Color)	public void setCDATAForeground(java.awt.Color color) CDATA のフォアグラウンド・カラーを設定します。 パラメータ : color - CDATA 用の新しいカラー。
setCommentDataFont(Font)	public void setCommentDataFont(java.awt.Font font) コメント文のフォントを設定します。 パラメータ : font - XML コメント用の新しいフォント。
setCommentDataForeground (Color)	public void setCommentDataForeground（java.awt.Color color) コメントのフォアグラウンド・カラーを設定します。 パラメータ : color - コメント用の新しいカラー。
setEditable(boolean)	public void setEditable(boolean edit) このオブジェクトを編集可能にするかどうかを示すために指定されたブールを設定します。 パラメータ : doc - 新しいブール値。
setPCDATAFont(Font)	public void setPCDATAFont(java.awt.Font font) PCDATA のフォントを設定します。 パラメータ : font - PCDATA 用の新しいフォント。
setPCDATAForeground(Color)	public void setPCDATAForeground(java.awt.Color color) PCDATA のフォアグラウンド・カラーを設定します。 パラメータ : color - PCDATA 用の新しいカラー。
setPIDataFont(Font)	public void setPIDataFont(java.awt.Font font) PI データのフォントを設定します。 パラメータ : font - PI データ用の新しいフォント。
setPIDataForeground(Color)	public void setPIDataForeground(java.awt.Color color) PI データのフォアグラウンド・カラーを設定します。 パラメータ : color -PI データ用の新しいカラー。

表 D-6 XMLSourceView Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
setPINameFont(Font)	public void setPINameFont(java.awt.Font font) PI 名のフォントを設定します。 パラメータ : font - PI 名用の新しいフォント。
setPINameForeground(Color)	public void setPINameForeground(java.awt.Color color) PI 名のフォアグラウンド・カラーを設定します。 パラメータ : color - PI 名用の新しいカラー。
setSelectedNode(Node)	public void setSelectedNode(org.w3c.dom.Node node) 選択された XML ノードにカーソル位置を設定します。 パラメータ : node - 選択したノード。
setSymbolFont(Font)	public void setSymbolFont(java.awt.Font font) 表記法のフォントを設定します。 パラメータ : font - 表記法用の新しいフォント。
setSymbolForeground(Color)	public void setSymbolForeground(java.awt.Color color) 表記法のフォアグラウンド・カラーを設定します。 パラメータ : color - 表記法用の新しいカラー。
setTagFont(Font)	public void setTagFont(java.awt.Font font) タグのフォントを設定します。 パラメータ : font - XML タグ用の新しいフォント。
setTagForeground(Color)	public void setTagForeground(java.awt.Color color) タグのフォアグラウンド・カラーを設定します。 パラメータ : color - XML タグ用の新しいカラー。
setXMLDocument(Document)	public void setXMLDocument(org.w3c.dom.Document document) XMLviewer と XML 文書を対応付けます。 パラメータ : document - 表示するドキュメント・オブジェクト。 参照 : getText()

DBAccess Bean の早見表

表 D-7 に、DBAccess Bean の主なクラスおよびメソッドを示します。

表 D-7 DBAccess Bean のクラスおよびメソッド

クラスおよびメソッド	説明
クラス	
DBAccess	このクラスは、複数の XML およびテキスト・ドキュメントを保持する CLOB 表を保持します。各表は、次の文を使用して作成されます。 CREATE TABLE tablename FILENAME CHAR(16) UNIQUE, FILEDATA CLOB) LOB(FILEDATA) STORE AS (DISABLE STORAGE IN ROW) 各 XML（またはテキスト）文書が表に行として格納されます。FILENAME フィールドには、その行を検索、更新または削除を行うためにキーとして使用される一意の文字列があります。ドキュメント・テキストは FILEDATA フィールドに格納されます。これは CLOB オブジェクトです。この CLOB 表は Transviewer Bean が自動的に保持します。このクラスが保持する CLOB 表は、Transviewer Bean が後で使用します。このクラスは、CLOB 表の作成および削除、CLOB 表の内容の表示、および CLOB 表にあるテキスト・ドキュメントの追加、置換、削除を行います。
コンストラクタ	
DBAccess()	public DBAccess()
メソッド	
createBLOBTable(Connection, String)	BLOB 表を作成します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : 正常終了した場合は TRUE。
createXMLTable(Connection, String)	XML 表を作成します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : 正常終了した場合は TRUE。

表 D-7 DBAccess Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
deleteBLOBName(Connection, String, String)	BLOB 表からバイナリ・ファイルを削除します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 戻り値 : 正常終了した場合は TRUE。
deleteXMLName(Connection, String, String)	XML 表からテキスト・ファイルを削除します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 戻り値 : 正常終了した場合は TRUE。
dropBLOBTable(Connection, String)	public boolean dropBLOBTable(java.sql.Connection con, java.lang.String tablename) BLOB 表を削除します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : 正常終了した場合は TRUE。
dropXMLTable(Connection, String)	public boolean dropXMLTable(java.sql.Connection con, java.lang.String tablename) XML 表を削除します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : 正常終了した場合は TRUE。
getBLOBData(Connection, String, String)	public byte[] getBLOBData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname) BLOB 表からバイナリ・ファイルを取り出します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 戻り値 : バイト配列としてのファイル。

表 D-7 DBAccess Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
getNameSize()	public int getNameSize() ファイル名が格納されているフィールドのサイズを戻します。 戻り値 : ファイル名のサイズ。
getXMLData(Connection, String, String)	public java.lang.String getXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname) XML 表からテキスト・ファイルを取り出します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 戻り値 : 文字列としてのファイル。
getXMLNames(Connection, String)	public java.lang.String[] getXMLNames(java.sql.Connection con, java.lang.String tablename) XML 表のすべてのファイル名を戻します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : この表にあるすべてのファイル名を含む文字列の配列。
getXMLTableNames (Connection, String)	public java.lang.String[] getXMLTableNames(java.sql.Connection con, java.lang.String tablePrefix) 指定された文字列で始まる名前を持つすべての XML 表を取得します。 パラメータ : con - 接続オブジェクト。 tablePrefix - 表の接頭辞の文字列。 戻り値 : tablePrefix で始まるすべての XML 表の配列。

表 D-7 DBAccess Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
insertBLOBData(Connection, String, String, byte[])	public boolean insertBLOBData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, byte[] xmldata) バイナリ・ファイルを BLOB 表の行として挿入します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 xmldata - ファイル・データを含むバイト配列。 戻り値 : 正常終了した場合は TRUE。
insertXMLData(Connection, String, String, String)	public boolean insertXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, java.lang.String xmldata) テキスト・ファイルを XML 表の行として挿入します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 xmldata - ファイル・データを含むバイト配列。 戻り値 : 正常終了した場合は TRUE。
isXMLTable(Connection, String)	public boolean isXMLTable(java.sql.Connection con, java.lang.String tablename) 表が XML 表であるかどうかをチェックします。 パラメータ : con - 接続オブジェクト。 tablename - テストする表名。 戻り値 : XML 表である場合は TRUE。

表 D-7 DBAccess Bean のクラスおよびメソッド（続き）

クラスおよびメソッド	説明
replaceXMLData(Connection, String, String, String)	public boolean replaceXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, java.lang.String xmldata) テキスト・ファイルを XML 表の行として置換します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 xmlname - ファイル名。 xmldata - ファイル・データを含む文字列。 戻り値 : 正常終了した場合は TRUE。
xmlTableExists(Connection, String)	public boolean xmlTableExists(java.sql.Connection con, java.lang.String tablename) XML 表が存在するかどうかを確認します。 パラメータ : con - 接続オブジェクト。 tablename - 表名。 戻り値 : 表が存在する場合は TRUE。

参照： 第 23 章「XML Transviewer Beans の使用」を参照してください。

E

XDK for C: 仕様および早見表

この付録の内容は次のとおりです。

XML Parser for C 仕様

オラクル社では、XML Parser for Java、XML Parser for C、XML Parser for C++ および XML Parser for PL/SQL を提供しています。これらの各 XML パーサーはスタンドアロンの XML コンポーネントであり、アプリケーションで処理できるように、XML 文書（または DTD）を解析します。ライブラリおよびコマンドラインは、次の標準および機能をサポートしています。

- DOM サポートは、W3C の DOM 1.0 勧告に準拠しています。これらの API によってアプリケーションは、XML 文書をツリー構造としてメモリー内でアクセスし操作できます。このインタフェースは、エディタなどのアプリケーションで使用されます。
- SAX のサポートは、SAX 1.0 仕様に準拠しています。これらの API によってアプリケーションは、イベント駆動モデルを使用して XML 文書を処理できます。
- W3C の XML Namespace 1.0 勧告もサポートされています。これによって名前の競合が回避され、再利用性が向上し、アプリケーション統合が容易になります。
- 検証および非検証モードをサポートします。
- W3C の XML 1.0 勧告をサポートします。
- W3C の XSLT 1.0 勧告を統合サポート します。

検証モードおよび非検証モードのサポート

XML Parser for C は、検証モードまたは非検証モードで XML を解析できます。

- **非検証モード**では、XML Parser for C は、XML が整形式であることを確認し、データを、DOM API が操作できるオブジェクトのツリーに解析します。
- **検証モード**では、XML Parser for C は、XML が整形式であることを確認し、その XML データを DTD（存在する場合）に対して検証します。

検証には、属性名および要素タグが正当であるかどうか、ネストした要素が適切な場所にあるかどうかなどの確認も含まれます。

サンプル・コード

サンプル・コードおよび XML Parser for C の使用方法については、第 24 章「XML Parser for C の使用」を参照してください。

オンライン・ドキュメント

Oracle XML Parser for C のドキュメントは、`$ORACLE_HOME/xdk/c/parser/doc` ディレクトリにあります。

リリース固有の注意事項

アーカイブのルート・ディレクトリにある readme.html ファイルには、不具合の修正や追加の API などのリリース固有の情報が含まれています。

Oracle XML Parser for C は、C で作成されています。XML 文書が整形式であるか、または DTD に対して妥当であるかどうか（オプション）を確認します。Oracle XML Parser for C は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。

標準への準拠

XML Parser for C は、次の標準に準拠しています。

- W3C の XML 1.0 勧告
  （http://www.w3.org/TR/1998/REC-xml-19980210 を参照）
- W3C の DOM レベル 1 1.0 勧告
  （http://www.w3.org/TR/REC-DOM-Level-1/ を参照）
- W3C の XML Namespace 勧告
  （http://www.w3.org/TR/1998/PR-xml-names-19981117 を参照）
- SAX 1.0
  （http://www.megginson.com/SAX/index.html を参照）
- W3C の XSLT 1.0 勧告
  （http://www.w3.org/TR/xslt を参照）

キャラクタ・セット・エンコーディングのサポート

XML Parser for C は、『Oracle9*i* グローバリゼーション・サポート・ガイド』の付録 A の「キャラクタ・セット」に記載されているエンコーディングの他に、次のエンコーディングのドキュメントをサポートします。

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

デフォルト： デフォルトでは、UTF-8 がエンコーディングになります。シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 25% 向上します。

XML Parser for C のリリース履歴

表 E-1 に、XML Parser for C のリリース履歴をリストします。

表 E-1 XML Parser for C: リリース履歴

リリース	説明
Oracle XML Parser for C 2.0.4.0.0	これは、V2 の最初のリリースです。このリリースでの主な変更点は、不具合の修正です。 XML パーサーでは、次の不具合が修正されています。 ■ Bug#1352943 XMLPARSE() はファイル名を正常に処理できない場合があります。 ■ Bug#1302311 パラメータ・エンティティの処理に問題があります。 ■ Bug#1323674 XML Parser for C のエラー操作に一貫性がありません。 ■ Bug#1328871 LPXPRINTBUFFER によって、無条件で出力に XML コメントが付加されます。 ■ Bug#1349962 解放されたメモリー場所を使用すると、TLPXVNSA31.DIF が生成されます。oraxmldom.h は oradom.h に名前が変更されました。 XSLT プロセッサでは、次の不具合が修正されています。 ■ Bug#1225546 無効なエラー・メッセージに詳細が必要です。 ■ Bug#1267616 TLPXST14.DIF: LPXXP.C:LPXXPSUBSTRING() の DBL_MAX が SBIG_ORAMAXVAL に置換されます。 ■ Bug#1289228 デバッグに、ファイル名、行番号、ファンクションなどのエラー・コンテキストが必要です。 ■ Bug#1289214 xsl:choose が正常に実行されません。 ■ Bug#1298028 XPath の構造体 NOT（POSITION()=LAST()）が正常に動作していません。 ■ Bug#1298193 XPath ファンクションがパラメータの暗黙的な型変換を行いません。 ■ Bug#1323665 XML Parser for C がスタイルシートの解析用のベース・ディレクトリまたは URI を設定できません。 ■ Bug#1325452 XSL プロセスで過度のメモリー消費またはメモリー・リークがあります。 ■ Bug#1333693 XSL Processor for C での連鎖変換が正常に実行されていません（LPX-00002）。

表 E-1 XML Parser for C: リリース履歴

リリース	説明
Oracle XML Parser for C 2.0.3.0.0	SAX のメモリー使用量 : 大幅に減少しています。入力サイズおよび複数の解析（メモリー・リークは解消）に対する使用量は同じです。 XSLT のメモリー使用量 : 改善されています。 検証の警告 : 妥当性制約（VC）エラーは、警告に変更され、解析を中断しません。以前の動作（警告およびエラーでの中断）との互換性のために、新しいフラグ XML_FLAG_STOP_ON_WARNING（または XML プログラムでは「-W」）が追加されています。 パフォーマンスの向上 : 有限オートマタによる VC 構造の検証に切り替えることによって、パフォーマンスが 10% 向上しています。 HTTP サポート : HTTP の URI がサポートされています。次回のリリースで FTP がサポートされる予定です。他のアクセス方法として、ユーザーが新しい xmlaccess() API を使用して、独自のコールバックを定義することもできます。
Oracle XML Parser for C 2.0.2.0.0	XSLT の改善 : XSLT プロセッサの様々な不具合が修正されています。また、エラー・メッセージが改善されています。さらに、xsl:number、xsl:sort、xsl:namespace-alias、xsl:decimal-format、将来の xsl:version と互換性がある処理、およびスタイルシートとしてのリテラル結果要素が使用可能です。コア XPath ライブラリに、XSLT 固有の current()、format-number()、generate-id() および system-property() が追加されています。 不具合の修正 : 妥当性の問題、および開始タグと終了タグが SAX と一致しない問題が修正されています（Bug#1227096）。また、外部エンティティ内のパラメータ・エンティティの処理に関する不具合が修正されています（Bug#1225219）。
Oracle XML Parser 2.0.1.0.0 (C)	パフォーマンスの向上 : 前回からのパフォーマンスの主な向上点は、解析速度が、UTF-8 の解析では 2.5 倍、ASCII の解析では約 4 倍、高速化されていることです。次の表に、DOM の解析時間、および様々なスタンドアロン・ファイルの検証時間（SPARC Ultra 1 型の CPU 時間）について、前回のリリースとの比較を示します。

ファイル・サイズ	旧 UTF-8	新 UTF-8	速度向上率	旧 ASCII	新 ASCII	速度向上率
42KB	180ms	70ms	2.6	120ms	40ms	3.0
134KB	510ms	210ms	2.4	450ms	100ms	4.5
247KB	980ms	400ms	2.5	690ms	180ms	
3.81MB	2860ms	1130ms	2.5	1820ms	380ms	4.82
7MB	10550ms	4100ms	2.6	7450ms	1930ms	3.9
10.5MB	42250ms	16400ms	2.6	29900ms	7800ms	3.8

表 E-1 XML Parser for C: リリース履歴

リリース	説明
	準拠の改善 : XML 1.0 仕様にさらに厳密に準拠させたことによって、標準テスト・スイート（Jim Clark、Oasis など）において高いスコアを示しています。 配列からリストへの切替え : 内部パーサーのデータ構造は、均一にリストされます。配列は廃止されています。そのため、numChildNodes/getChildNode ではなく、firstChild/nextSibling 形式のループによるアクセスがより適切です。 DTD の解析 : 新しい API コール xmlparsedtd() が追加されています。これは、ドキュメントを囲むことなく、外部 DTD を直接解析します。主に、Class Generator で使用されます。 エラー・レポート : エラー・メッセージが改善されています。内容がより明確になり、メッセージ数も約 2 倍です。エラーの位置は行番号 / エンティティの組のスタックで示され、そのエラーの最終位置と中間の内容（たとえば、ファイルの X 行、エンティティの Y 行）を示します。 **注意 :** このリリースに付属の新しいエラー・メッセージ・ファイル（lpxus.msb）を使用してください。以前のリリースのエラー・メッセージ・ファイルとは互換性がありません。次のリリースの説明を参照してください。 XSL の改善 : XSLT プロセッサの様々な不具合が修正されています。現在、xsl:call-template が完全にサポートされています。
Oracle XML Parser for C 2.0.0.0.0	Oracle XML Parser V2 はベータ・リリース版で、C で作成されています。Oracle XML Parser V1 との主な違いは、統合化された XSLT プロセッサを介したスタイルシートに従って XML 文書をフォーマットできることです。Oracle XML Parser は、XML 文書が整形式であるか、また DTD に対して妥当であるかどうか（オプション）を確認します。Oracle XML Parser は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。 Solaris 2.6、Linux 2.2、HP-UX 11.0 および Windows NT 4.0 Service Pack 3 以上のオペレーティング・システムがサポートされています。使用許諾契約を読んでから、この製品を使用してください。

XML Parser for C: パーサー関数

表 E-2 に、XML Parser for C のパーサー関数、説明および構文を示します。

表 E-2 XML Parser for C: パーサー関数

関数	説明	構文およびコメント
xmlinit	XML パーサーを初期化します。	xmlctx *xmlinit (uword *err, const oratext *encoding, void (*msghdlr)(void *msgctx, const oratext *msg, ub4 errcode), void *msgctx, const xmlsaxcb *saxcb, void *saxcbctx, const xmlmemcb *memcb, void *memcbctx, const oratext *lang);
xmlclean	解析中に使用されたメモリーをクリーンアップします。	void xmlclean(xmlctx *ctx); 複数のファイルを解析する場合に、解析に使用されたメモリーを解放してから、xmlparse() または xmlparsebuf() をコールする場合に使用します。
xmlparse	ファイルを解析します。	uword xmlparse(xmlctx *ctx, const oratext *filename, const oratext *encoding, ub4 flags); フラグ・ビットは、OR 計算して、XML パーサーのデフォルトの動作をオーバーライドする必要があります。次のフラグ・ビットを設定できます。 ■ XML_FLAG_VALIDATE: 検証を開始します。 ■ XML_FLAG_DISCARD_WHITESPACE: 重要でない空白を削除します。 デフォルト動作では、入力が検証されません。デフォルト動作では、空白の処理が XML 1.0 仕様に完全に準拠します。すべての空白は無視できる空白が明示された状態でアプリケーションに戻されます。
xmlparsebuf	バッファを解析します。	uword xmlparsebuf(xmlctx *ctx, const oratext *buffer, size_t len, const oratext *encoding, ub4 flags);
xmlterm	XML パーサーを終了します。	uword xmlterm(xmlctx *ctx);
createDocument	新しいドキュメントを作成します。	xmlnode* createDocument(xmlctx *ctx) XML 文書は、常に DOCUMENT_NODE 型のノードがルートになります。このファンクションによって、ルート・ノードが作成され、コンテキスト内に設定されます。
isStandalone	ドキュメントのスタンドアロン・フラグを戻します。	boolean isStandalone(xmlctx *ctx) ドキュメントのスタンドアロン・フラグのブール値を、<?xml?> 処理命令で指定したとおりに戻します。

XML Parser for C: DOM API 関数

表 E-3 に、XML Parser for C の DOM API 関数を示します。

表 E-3 XML Parser for C: DOM API 関数

関数	説明
appendChild	子ノードを現行のノードに追加します。
appendData	文字データをノードの現行データの最後に追加します。
cloneNode	現行のノードと同一の新しいノードを作成します。
createAttribute	要素ノードの新しい属性を作成します。
createCDATASection	CDATA_SECTION ノードを作成します。
createComment	COMMENT ノードを作成します。
createDocumentFragment	DOCUMENT_FRAGMENT ノードを作成します。
createElement	ELEMENT ノードを作成します。
createEntityReference	ENTITY_REFERENCE ノードを作成します。
createProcessingInstruction	PROCESSING_INSTRUCTION（PI）ノードを作成します。
createTextNode	TEXT ノードを作成します。
deleteData	ノードの文字データから部分文字列を削除します。
getAttrName	属性名を戻します。
getAttrSpecified	属性の指定フラグの値を戻します。[DOM getSpecified]
getAttrValue	属性の値を戻します。
getAttribute	属性の値を戻します。
getAttributeIndex	指定された索引の要素の属性を戻します。
getAttributeNode	指定された名前の要素の属性ノードを取得します。[DOM getName]
getAttributes	要素の属性の配列を戻します。
getCharData	TEXT ノードの文字データを戻します。[DOM getData]
getCharLength	TEXT ノードの文字データの長さを戻します。[DOM getLength]
getChildNode	ノードの配列から索引付けされたノードを戻します。[DOM item]
getChildNodes	子ノードの配列を戻します。

表 E-3 XML Parser for C: DOM API 関数（続き）

関数	説明
getContentModel	DTD から要素の内容モデルを戻します。[DOM extension]
getDocument	最上位の DOCUMENT ノードを戻します。[DOM extension]
getDocumentElemen	最上位（ルート）の ELEMENT ノードを戻します。
getDocType	現行の DTD を戻します。
getDocTypeEntities	DTD の汎用エンティティの配列を戻します。
getDocTypeName	DTD の名前を戻します。
getDocTypeNotations	DTD の表記法の配列を戻します。
getElementsByTagName	一致する名前を持つ要素のリストを戻します。
getEntityNotation	エンティティの NDATA を戻します。[DOM getNotation]
getEntityPubID	エンティティの公開識別子を戻します。[DOM getPublicId]
getEntitySysID	エンティティのシステム識別子を戻します。[DOM getSystemId]
getFirstChild	ノードの最初の子を戻します。
getImplementation	DOM 実装の構造（定義済の場合）を戻します。
getLastChild	ノードの最後の子を戻します。
getModifier	内容モデルのノードの修飾子（「?」、「*」または「+」）を戻します。[DOM extension]
getNextSibling	ノードの次の兄弟関係を戻します。
getNamedItem	ノード・リストから名前付きノードを戻します。
getNodeMapLength	NodeMap 内のエントリ数を戻します。[DOM getLength]
getNodeName	ノード名を戻します。
getNodeType	ノードの型コード（列挙）を戻します。
getNodeValue	ノードの値（文字データ）を戻します。
getNotationPubID	表記法の公開識別子を戻します。[DOM getPublicId]
getNotationSysID	表記法のシステム識別子を戻します。[DOM getSystemId]
getOwnerDocument	指定されたノードを含む DOCUMENT ノードを戻します。
getPIData	処理命令のデータを戻します。[DOM getData]
getPITarget	処理命令のターゲットを戻します。[DOM getTarget]

表 E-3 XML Parser for C: DOM API 関数（続き）

関数	説明
getParentNode	ノードの親ノードを戻します。
getPreviousSibling	ノードの以前の兄弟関係を戻します。
getTagName	現行の名前と同じであるノードの「タグ名」を戻します。
hasAttributes	要素ノードに属性があるかどうかを判断します。[DOM extension]
hasChildNodes	ノードが子ノードを持っているかどうかを判断します。
hasFeature	DOM の実装によって固有の機能がサポートされるかどうかを判断します。
insertBefore	指定された参照ノードの前に新しい子ノードを挿入します。
insertData	ノードの既存のデータに新しい文字データを挿入します。
isStandalone	ドキュメントがスタンドアロンかどうかを判断します。[DOM extension]
nodeValid	現行の DTD に対してノードを検証します。[DOM extension]
normalize	隣接する TEXT ノードをマージしてノードを正規化します。
numAttributes	要素ノードの属性数を戻します。[DOM extension]
numChildNodes	ノードの子ノードの数を戻します。[DOM extension]
removeAttribute	指定された名前の要素の属性を削除します。
removeAttributeNode	指定されたポインタの要素の属性を削除します。
removeChild	子ノードの親リストからノードを削除します。
removeNamedItem	ノードのリストから指定された名前のノードを削除します。
replaceChild	1 つのノードを他のノードで置換します。
replaceData	ノードの文字データの部分文字列を、別の文字列 setAttribute で置換します。指定された属性名および値 setAttributeNode を持つ要素ノードに、新しい属性を設定（追加または置換）します。新しい属性への指定されたポインタを持つ要素ノードに、新しい属性を設定（追加または置換）します。
setNamedItem	子ノードの親リストに新しいノードを設定（追加または置換）します。
setNodeValue	ノードの「値」（文字データ）を設定します。
setPIData	処理命令のデータを設定します。[DOM setData]

XML Parser for C: Namespace API 関数

表 E-4 に、XML Parser for C の Namespace 関数を示します。

表 E-4 XML Parser for C: Namespace API 関数

関数	説明
getAttrLocal(xmlattr *attrs)	属性のローカル名を戻します。
getAttrNamespace(xmlattr *attr)	属性の名前空間（URI）を戻します。
getAttrPrefix(xmlattr *attr)	属性の接頭辞を戻します。
getAttrQualifiedName(xmlattr *attr)	属性の完全修飾名を戻します。
getNodeLocal(xmlnode *node)	ノードのローカル名を戻します。
getNodeNamespace(xmlnode *node)	ノードの名前空間（URI）を戻します。
getNodePrefix(xmlnode *node)	ノードの接頭辞を戻します。
getNodeQualifiedName(xmlnode *node)	ノードの修飾名を戻します。

XML Parser for C: XSLT API 関数

表 E-5 に、XML Parser for C の XSLT 関数を示します。

表 E-5 XML Parser for C: XSLT API の関数

関数	説明
xslprocess() xslprocess(xmlctx *docctx, xmlctx *xslctx, xmlctx *resctx, xmlnode **result)	XSL スタイルシートを XML 文書ソースで処理して、成功コードまたはエラー・コードを戻します。

XML Parser for C: SAX API 関数

表 E-6 に、XML Parser for C の SAX API 関数を示します。

表 E-6 XML Parser for C: SAX API 関数

SAX API 関数	説明
characters(void *ctx, const oratext *ch, size_t len)	要素内の文字データの通知を受け取ります。
endDocument(void *ctx)	ドキュメントの終わりの通知を受け取ります。
endElement(void *ctx, const oratext *name)	要素の終わりの通知を受け取ります。
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	要素内容にある無視可能な空白の通知を受け取ります。
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	表記法宣言の通知を受け取ります。
processingInstruction(void *ctx, const oratext *target, const oratext *data)	処理命令の通知を受け取ります。
startDocument(void *ctx)	ドキュメントの始まりの通知を受け取ります。
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	要素の始まりの通知を受け取ります。
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	解析対象外のエンティティ宣言の通知を受け取ります。
非 SAX コールバック関数	
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	要素の名前空間の始まりの通知を受け取ります。

F

XDK for C++: 仕様および早見表

この付録の内容は次のとおりです。

XML Parser for C++ の仕様

オラクル社では、XML Parser for Java、XML Parser for C、XML Parser for C++ および XML Parser for PL/SQL を提供しています。これらの各 XML Parser はスタンドアロンの XML コンポーネントであり、アプリケーションで処理できるように、XML 文書（または DTD）を解析します。ライブラリおよびコマンドラインは、次の標準および機能をサポートしています。

- DOM サポートは、W3C の DOM 1.0 勧告に準拠しています。これらの API によってアプリケーションは、XML 文書をツリー構造としてメモリー内でアクセスし操作できます。このインタフェースは、エディタなどのアプリケーションで使用されます。
- SAX のサポートは、SAX 1.0 仕様に準拠しています。これらの API によってアプリケーションは、イベント駆動モデルを使用して XML 文書を処理できます。
- W3C の XML Namespace 1.0 勧告もサポートされています。これによって名前の競合が回避され、再利用性が向上し、アプリケーション統合が容易になります。
- 検証および非検証モードをサポートします。
- W3C の XML 1.0 勧告をサポートします。
- W3C XSLT 1.0 勧告を統合サポートします。

検証モードおよび非検証モードのサポート

XML Parser for C++ は、検証モードまたは非検証モードで XML を解析できます。

- **非検証モード**では、XML Parser for C++ は、XML が整形式であることを確認し、データを、DOM API が操作できるオブジェクトのツリーに解析します。
- **検証モード**では、XML Parser for C++ は、XML が整形式であることを確認し、その XML データを DTD（存在する場合）に対して検証します。

検証には、属性名および要素タグが正当であるかどうか、ネストした要素が適切な場所にあるかどうかなどの確認も含まれます。

サンプル・コード

サンプル・コードおよび XML Parser for C++ の使用方法については、第 26 章「XML Parser for C++ の使用」を参照してください。

リリース固有の注意事項

アーカイブのルート・ディレクトリにある readme.html ファイルには、不具合の修正や追加の API などのリリース固有の情報が含まれています。

Oracle XML Parser for C++ は、C++ ラッパーを使用して C で作成されています。XML 文書が整形式であるか、または DTD に対して妥当であるかどうか（オプション）を確認します。Oracle XML Parser for C++ は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。

標準への準拠

XML Parser for C++ は、次の標準に準拠しています。

- W3C の XML 1.0 勧告
（http://www.w3.org/TR/1998/REC-xml-19980210 を参照）
- W3C の DOM レベル 1 1.0 勧告
（http://www.w3.org/TR/REC-DOM-Level-1/ を参照）
- W3C の XML Namespace 勧告
（http://www.w3.org/TR/1998/PR-xml-names-19981117 を参照）
- SAX 1.0
（http://www.megginson.com/SAX/index.html を参照）
- W3C の XSL Transform 1.0 勧告
（http://www.w3.org/TR/xslt を参照）

キャラクタ・セット・エンコーディングのサポート

XML Parser for C++ は、『Oracle9*i* グローバリゼーション・サポート・ガイド』の付録 A の「キャラクタ・セット」に記載されているエンコーディングの他に、次のエンコーディングのドキュメントをサポートします。

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

デフォルト： デフォルトでは、UTF-8 がエンコーディングになります。シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 25% 向上します。

XML Parser for C++ のリリース履歴

表 F-1 に、XML Parser for C++ のリリース履歴を示します。

表 F-1 XML Parser for C++: リリース履歴

リリース	説明
Oracle XML Parser for C++ 2.0.4.0.0	これは、V2 の最初のリリースです。このリリースでの主な変更点は、不具合の修正です。 XML Parser では、次の不具合が修正されています。 ■ Bug#1352943 XMLPARSE() はファイル名を正常に処理できない場合があります。 ■ Bug#1302311 パラメータ・エンティティの処理に問題があります。 ■ Bug#1323674 XML Parser for C のエラー操作に一貫性がありません。 ■ Bug#1328871 LPXPRINTBUFFER によって、無条件で出力に XML コメントが付加されます。 ■ Bug#1349962 解放されたメモリー場所を使用すると、TLPXVNSA31.DIF が生成されます。oraxmldom.h は oradom.h に名前が変更されました。 XSLT プロセッサでは、次の不具合が修正されています。 ■ Bug#1225546 無効なエラー・メッセージに詳細が必要です。 ■ Bug#1267616 TLPXST14.DIF: LPXXP.C:LPXXPSUBSTRING() の DBL_MAX が SBIG_ORAMAXVAL に置換されます。 ■ Bug#1289228 デバッグに、ファイル名、行番号、ファンクションなどのエラー・コンテキストが必要です。 ■ Bug#1289214 xsl:choose が正常に実行されません。 ■ Bug#1298028 XPath の構造体 NOT（POSITION()=LAST()）が正常に動作していません。 ■ Bug#1298193 XPath ファンクションがパラメータの暗黙的な型変換を行いません。 ■ Bug#1323665 XML Parser for C がスタイルシートの解析用のベース・ディレクトリまたは URI を設定できません。 ■ Bug#1325452 XSL プロセスで過度のメモリー消費またはメモリー・リークがあります。 ■ Bug#1333693 XSL Processor for C での連鎖変換が正常に実行されていません（LPX-00002）。

表 F-1 XML Parser for C++: リリース履歴 （続き）

リリース	説明
Oracle XML Parser for C++ 2.0.3.0.0	SAX のメモリー使用量 : 減少しています。入力サイズおよび複数の解析（メモリー・リークは解消）に対する使用量は同じです。 XSLT のメモリー使用量 : 改善しています。 検証の警告 : 妥当性制約（VC）エラーは、警告に変更され、解析を中断しません。以前の動作（警告およびエラーでの中断）との互換性のために、新しいフラグ XML_FLAG_STOP_ON_WARNING（または XML プログラムでは「-W」）が追加されています。 HTTP サポート : HTTP の URI がサポートされています。次回のリリースで FTP がサポートされる予定です。他のアクセス方法として、ユーザーが新しい xmlaccess() API を使用して、独自のコールバックを定義することもできます。
Oracle XML Parser for C++ 2.0.2.0.0	XSLT の改善 : XSLT プロセッサの様々な不具合が修正されています。また、エラー・メッセージが改善されています。さらに、xsl:number、xsl:sort、xsl:namespace-alias、xsl:decimal-format、将来の xsl:version と互換性がある処理、およびスタイルシートとしてのリテラル結果要素が使用可能です。コア XPath ライブラリに、XSLT 固有の current()、format-number()、generate-id() および system-property() が追加されています。 不具合の修正 : 妥当性の問題、および開始タグと終了タグが SAX と一致しない問題が修正されています。また、外部エンティティ内のパラメータ・エンティティの処理に関する不具合が修正されています。
Oracle XML Parser for C++ 2.0.1.0.0	パフォーマンスの向上 : 前回からのパフォーマンスの主な向上点は、解析速度が、UTF-8 の解析では 2.5 倍、ASCII の解析では約 4 倍、高速化されていることです。次の表に、DOM の解析時間、および様々なスタンドアロン・ファイルの検証時間（SPARC Ultra 1 型の CPU 時間）について、前回のリリースとの比較を示します。

ファイル・サイズ	旧 UTF-8	新 UTF-8	速度向上率	旧 ASCII	新 ASCII	速度向上率
42KB	180ms	70ms	2.6	120ms	40ms	3.0
134KB	510ms	210ms	2.4	450ms	100ms	4.5
247KB	980ms	400ms	2.5	690ms	180ms	3.8
1MB	2860ms	1130ms	2.5	1820ms	380ms	4.8
2.7MB	10550ms	4100ms	2.6	7450ms	1930ms	3.9
10.5MB	42250ms	16400ms	2.6	29900ms	7800ms	3.8

準拠の改善 : XML 1.0 仕様にさらに厳密に準拠させたことによって、標準テスト・スイート（Jim Clark、Oasis など）において高いスコアを示しています。

表 F-1 XML Parser for C++: リリース履歴 （続き）

リリース	説明
	配列からリストへの切替え : 内部パーサーのデータ構造は、均一にリストされます。配列は廃止されています。そのため、numChildNodes/getChildNode ではなく、firstChild/nextSibling 形式のループによるアクセスがより適切です。 DTD の解析 : 新しい API コール xmlparsedtd() が追加されています。これは、ドキュメントを囲むことなく、外部 DTD を直接解析します。主に、Class Generator で使用されます。 エラー・レポート : エラー・メッセージが改善されています。内容がより明確になり、メッセージ数も約 2 倍です。エラーの位置は行番号 / エンティティの組のスタックで示され、そのエラーの最終位置と中間の内容（たとえば、ファイルの X 行、エンティティの Y 行）を示します。 **注意 :** このリリースに付属の新しいエラー・メッセージ・ファイル（lpxus.msb）を使用してください。以前のリリースのエラー・メッセージ・ファイルとは互換性がありません。次のリリースの説明を参照してください。 XSL の改善 : XSLT プロセッサの様々な不具合が修正されています。現在、xsl:call-template が完全にサポートされています。
Oracle XML Parser for C++ 2.0.0.0.0	Oracle XML Parser V2 はベータ・リリース版で、C++ ラッパーを使用して C で作成されています。Oracle XML Parser V1 との主な違いは、統合化された XSLT プロセッサを介したスタイルシートに従って XML 文書をフォーマットできることです。Oracle XML Parser は、XML 文書が整形式であるか、また DTD に対して妥当であるかどうか（オプション）を確認します。Oracle XML Parser は、DOM インタフェースを介してアクセス可能なオブジェクト・ツリーを構築するか、SAX インタフェースを介して順次操作します。

XML Parser for C++: XMLParser() API

表 F-2 に、XML Parser for C++ の主な XMLParser() クラスのメソッドをそれぞれ簡単に説明します。XMLParser() クラスには、次のことを実行できるトップレベル・メソッドが含まれます。

- パーサーを起動します。
- ドキュメントに関する高レベルの情報を戻します。

表 F-2 XML Parser for C++: XMLParser() クラス

XMLParser() メソッド	説明
xmlinit	XML パーサーを初期化します。 uword xmlinit(oratext *encoding, void (*msghdlr)(void *msgctx, oratext *msg, ub4 errcode), void *msgctx, lpxsaxcb *saxcb, void *saxcbctx, oratext *lang)
xmlterm	XML パーサーを終了します。
xmlparse	ファイルからドキュメントを解析します。
xmlparseBuffer	バッファからドキュメントを解析します。
getContent	要素の内容モデルを戻します。
getModifier	内容モデルのノードの修飾子（「?」、「*」または「+」）を戻します。
getDocument	解析済ドキュメントのルート・ノードを戻します。
getDocumentElement	解析済ドキュメントのルート要素（ノード）を戻します。
getDocType	ドキュメント・タイプ文字列を戻します。
isStandalone	スタンドアロン・フラグの値を戻します。ドキュメントが、<?xml?> 行上にスタンドアロンで指定されている場合、TRUE を戻します。それ以外の場合は、FALSE を戻します。

XML Parser for C++: DOM API

表 F-3 に、XML Parser for C ++ の DOM API メソッドを示し、それぞれ簡単に説明します。

表 F-3 XML Parser for C++: DOM API クラス（サブクラス）

クラス（サブクラス）	メソッド	説明
Attr (Node)		
このクラスには、シングル・ドキュメント・ノード属性の名前および値にアクセスするためのメソッドが含まれます。		
	getName	属性の名前を戻します。
	getValue	属性の「値」（定義）を戻します。
	getSpecified	属性の「指定された」フラグ値を戻します。
	setValue	属性の値を設定します。
CDATASection (Text)		
このクラスは、サブクラスが Text である CDATA ノード・タイプを実装します。メソッドはありません。		
CharacterData (Node)		
このクラスには、テキスト・ノードに対応付けられたデータにアクセスし、変更するためのメソッドが含まれます。		
	appendData	このノードのデータに文字列を追加します。
	deleteData	このノードのデータから部分文字列を削除します。
	getData	テキスト・ノードのデータ（値）を取得します。
	getLength	テキスト・ノードのデータの長さを戻します。
	insertData	このノードのデータに文字列を挿入します。
	replaceData	このノードのデータの部分文字列を置換します。
	substringData	このノードのデータの部分文字列をフェッチします。
Comment (CharacterData)		
このクラスは、サブクラスが CharacterData である COMMENT ノード・タイプを実装します。メソッドはありません。		

表 F-3 XML Parser for C++: DOM API クラス（サブクラス）（続き）

クラス（サブクラス）	メソッド	説明
Document (Node)		
このクラスには、ノードを作成および取り出すためのメソッドが含まれます。		
	createAttribute	ATTRIBUTE ノードを作成します。
	createCDATASection	CDATA ノードを作成します。
	createComment	COMMENT ノードを作成します。
	createDocumentFragment	DOCUMENT_FRAGMENT ノードを作成します。
	createElement	ELEMENT ノードを作成します。
	createEntityReference	ENTITY_REFERENCE ノードを作成します。
	createProcessingInstruction	PROCESSING_INSTRUCTION ノードを作成します。
	createTextNode	TEXT ノードを作成します。
	getElementsByTagName	タグ名に基づいてノードを選択します。
	getImplementation	ドキュメントの DTD を戻します。
DocumentFragment (Node)		
このクラスは、サブクラスが Node である DOCUMENT_FRAGMENT ノード・タイプを実装します。		
DocumentType (Node)		
このクラスには、ドキュメントの DTD に関する情報にアクセスするためのメソッドが含まれます。		
	getName	DTD の名前を戻します。
	getEntities	DTD の（汎用）エンティティの名前付きノード・マップを戻します。
	getNotations	DTD の表記法の名前付きノード・マップを戻します。
DOMImplementation		
このクラスには、XML Parser によってサポートされる特定の DOM 実装に関連するメソッドが含まれます。		
	hasFeature	名前付き機能がサポートされているかどうかを検出します。

表 F-3 XML Parser for C++: DOM API クラス（サブクラス）（続き）

クラス（サブクラス）	メソッド	説明
Element (Node)		
このクラスには、要素ノードに関連するメソッドが含まれます。		
	getTagName	ノードのタグ名を戻します。
	getAttribute	指定された名前の属性を選択します。
	setAttribute	指定された名前および値の新しい属性を作成します。
	removeAttribute	指定された名前の属性を削除します。
	getAttributeNode	指定された名前の属性を削除します。
	setAttributeNode	新しい属性ノードを追加します。
	removeAttributeNode	属性ノードを削除します。
	getElementsByTagName	指定されたタグ名の要素ノードのリストを戻します。
	normalize	要素を正規化します（隣接したテキスト・ノードをマージします）。
Entity (Node)		
このクラスは、サブクラスが Node である ENTITY ノード・タイプを実装します。		
	getNotation	エンティティの NDATA（表記法名）を戻します。
	getPublicId	エンティティの公開識別子を戻します。
	getSystemId	エンティティのシステム識別子を戻します。
EntityReference (Node)		
このクラスは、サブクラスが Node である ENTITY_REFERENCE ノード・タイプを実装します。		
NamedNodeMap		
このクラスには、ノード・マップ内のノード番号にアクセスし、個々のノードをフェッチするためのメソッドが含まれます。		
	item	マップ内の *n* 番目のノードを戻します。
	getLength	マップ内のノード数を戻します。
	getNamedItem	名前ごとにノードを選択します。
	setNamedItem	ノードをマップに設定します。
	getLength	マップから指定のノードを削除します。

表 F-3 XML Parser for C++: DOM API クラス（サブクラス）（続き）

クラス（サブクラス）	メソッド	説明
Node		
このクラスには、ドキュメント・ノードの詳細なメソッドが含まれます。		
	appendChild	現行ノードの子のリストの最後に、新しい子を追加します。
	cloneNode	既存のノード、およびそのすべての子（オプション）を複製します。
	getAttributes	すべての定義済ノード属性を含む構造を戻します。
	getChildNode	指定されたノードの索引付けされた特定の子を戻します。
	getChildNodes	指定されたノードのすべての子ノードを含む構造を戻します。
	getFirstChild	指定されたノードの最初の子を戻します。
	getLastChild	指定されたノードの最後の子を戻します。
	getLocal	ノードのローカル名を戻します。
	getNamespace	ノードの名前空間を戻します。
	getNextSibling	ノードの次の兄弟関係を戻します。
	getName	ノード名を戻します。
	getType	ノードの数値型コードを戻します。
	getValue	ノードの値（データ）を戻します。
	getOwnerDocument	ノードを含むドキュメント・ノードを戻します。
	getParentNode	指定されたノードの親ノードを戻します。
	getPrefix	ノードの名前空間の接頭辞を戻します。
	getPreviousSibling	現行ノードの以前の兄弟関係を戻します。
	getQualifiedName	指定されたノードの名前空間で修飾されたノードを戻します。
	hasAttributes	ノードに定義済属性があるかどうかを判断します。
	hasChildNodes	ノードに子があるかどうかを判断します。

表 F-3 XML Parser for C++: DOM API クラス（サブクラス）（続き）

クラス（サブクラス）	メソッド	説明
	insertBefore	ノードの子のリストに新しい子ノードを挿入します。
	numChildNodes	指定されたノードの子ノード数のカウントを戻します。
	removeChild	現行ノードの子のリストからノードを削除します。
	replaceChild	子ノードを別のノードと置換します。
	setValue	ノードの値（データ）を設定します。
NodeList		
このクラスには、NodeList からノードを取得するためのメソッドが含まれます。		
	item	リスト内の *n* 番目のノードを戻します。
	getLength	リスト内のノード数を戻します。
Notation (Node)		
このクラスは、サブクラスが Node である NOTATION ノード・タイプを実装します。		
	getData	表記法のデータを戻します。
	getTarget	表記法のターゲットを戻します。
	setData	表記法のデータを設定します。
ProcessingInstruction (Node)		
このクラスは、サブクラスが Node である PROCESSING_INSTRUCTION ノード・タイプを実装します。		
	getData	PI のデータを戻します。
	getTarget	PI のターゲットを戻します。
	setData	PI のデータを設定します。
Text (CharacterData)		
このクラスには、テキスト・ノードに対応付けられたデータにアクセスし、変更するためのメソッドが含まれます（サブクラス CharacterData）。		
	splitText	テキスト・ノードのデータ（値）を取得します。

XML Parser for C++: XSLT API

XSLT は、XML 文書を他の XML 文書に変換するための言語です。XSLT は、XML 用のスタイルシート言語である XSL の一部として使用するために設計されています。XSL には、XSLT の他に、書式設定用の XML ボキャブラリが含まれます。XSL は、XSLT を使用して XML 文書のスタイルを指定し、その文書を、書式設定用ボキャブラリを使用する別の XML 文書へ変換する方法を記述します。

XSLT は、XSL から独立して使用できるようにも設計されています。ただし、XSLT は、完全な XML 変換用の汎用言語ではありません。XSL の一部として XSLT を使用する場合に必要な変換処理を目的として設計されています。

XSLT で表現される変換は、ソース・ツリーから結果ツリーへ変換する規則を表します。この変換は、テンプレートとパターンを対応付けることによって実行されます。パターンは、ソース・ツリー内の要素と一致します。結果ツリーの一部を作成するために、テンプレートはインスタンス化されます。結果ツリーは、ソース・ツリーから切り離されています。結果ツリーは、ソース・ツリーの構造と異なる構造にできます。結果ツリーを構築するときに、ソース・ツリーの要素をフィルタしたり、再順序付けすることができます。また、任意の構造を追加することもできます。

スタイルシート

XSLT で表現される変換を、スタイルシートといいます。これは、XSLT を XSL の書式設定用ボキャブラリに変換するときに、この変換がスタイルシートとして機能するためです。

スタイルシートには、一連のテンプレート規則が含まれます。テンプレート規則には、次の 2 つの部分があります。

- ソース・ツリー内のノードと一致するパターン。
- インスタンス化して結果ツリーの一部を形成するテンプレート。これによって、類似したソース・ツリー構造を持つ様々なドキュメントに、スタイルシートを適用できます。

テンプレートの処理方法

テンプレートは、特定のソース要素に対してインスタンス化され、結果ツリーの一部を作成します。テンプレートには、リテラル結果要素の構造を指定する要素を含めることができます。また、結果ツリー・フラグメントを作成するための命令である XSLT 名前空間の要素を含めることもできます。テンプレートがインスタンス化されると、各命令が実行され、作成された結果ツリー・フラグメントによって置き換えられます。

命令は、子のソース要素を選択および処理します。子の要素を処理すると、適用可能なテンプレート規則を検索し、そのテンプレートをインスタンス化することによって、結果ツリー・フラグメントが作成されます。ただし、要素が処理されるのは、命令の実行によって選択された場合のみであることに注意してください。結果ツリーは、ルート・ノード用のテンプレート規則を検索し、そのテンプレートをインスタンス化することによって構築されます。

XSL プロセッサと呼ばれるソフトウェア・モジュールは、XML 文書を読み取り、異なるスタイルの他の XML 文書に変換します。

XSL プロセッサの XML Parser for C++ 実装は、XSL Transformations 標準（バージョン 1.0、1999 年 11 月 16 日）に準拠し、XSLT 仕様に指定されている XSL プロセッサに必要な動作を含みます。

表 F-4 に、XSLProcessor クラス・メソッドおよび構文の概要を示します。

表 F-4 XML Parser for C++: XSL プロセッサ・クラス

クラス	メソッド
XSLProcessor このクラスには、XSL プロセッサを起動するトップレベル・メソッドが含まれます。	xslprocess() XML 文書ソースで XSL スタイルシートを処理します。 構文 : uword xslprocess(XMLParser *docctx, XMLParser *xslctx, XMLParser *resctx, Node **result); パラメータ : docctx (IN/OUT) - XML ドキュメント・コンテキスト xslctx (IN) - XSL スタイルシート・コンテキスト resctx (IN) - 結果ドキュメント・フラグメント・コンテキスト result (IN/OUT) - 結果ドキュメント・フラグメント・ノード

XML Parser for C++: SAX API

SAX API は、コールバックに基づいています。DOM インタフェースが、ドキュメント全体を解析し、参照できるデータ構造に変換するのに対し、SAX インタフェースはシリアルに動作します。ドキュメントが処理されると、適切な SAX のユーザー・コールバック関数が起動されます。各コールバック関数は、エラー・コードを戻します。0（ゼロ）は成功、0（ゼロ）以外の値は失敗を表します。0（ゼロ）以外のコードが戻された場合、ドキュメント処理は停止します。

SAX を使用するために、xmlsaxcb 構造はファンクション・ポインタで初期化され、xmlinit() コールに渡されます。ユーザー定義のコンテキスト構造体へのポインタも含まれるため、コンテキスト・ポインタは各 SAX 関数に渡されます。

この SAX 機能は、XML Parser for C の機能と同じです。

表 F-5 に、XML Parser for C++ の SAX API 関数を示します。

表 F-5 XML Parser for C++: SAX API 関数

SAX 関数	簡単な説明
characters(void *ctx, const oratext *ch, size_t len)	要素内の文字データの通知を受け取ります。
endDocument(void *ctx)	ドキュメントの終わりの通知を受け取ります。
endElement(void *ctx, const oratext *name)	要素の終わりの通知を受け取ります。
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	要素内容にある無視可能な空白の通知を受け取ります。
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	表記法宣言の通知を受け取ります。
processingInstruction(void *ctx, const oratext *target, const oratext *data)	処理命令の通知を受け取ります。
startDocument(void *ctx)	ドキュメントの始まりの通知を受け取ります。
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	要素の始まりの通知を受け取ります。
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	解析対象外のエンティティ宣言の通知を受け取ります。
非 SAX コールバック関数	
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	要素の名前空間の始まりの通知を受け取ります。

XML C++ Class Generator の仕様

XML Class Generator は、XML Parser for C++ とともに動作させると、入力 DTD に基づいて一連の C++ ソース・ファイルを生成します。生成された C++ ソース・ファイルを使用して、指定された DTD に準拠した XML 文書を作成、検証（オプション）および出力できます。Class Generator は、デバッグに有効な検証モードをサポートします。

XML C++ Class Generator への入力

入力は、DTD を含む XML 文書です。文書本体自体は無視されます。DTD のみが関連しますが、ダミー・ドキュメントは DTD に準拠する必要があります。基礎となる XML パーサーのみが、ドキュメントのファイル名およびそれに対応付けられた外部エンティティを受け入れます。将来のリリースでは、ダミー・ドキュメントは必要なくなり、追加プロトコルの URI が受け入れられます。

キャラクタ・セットのサポート

次に、XML C++ Class Generator へのファイル入力用にサポートされているキャラクタ・セット・エンコーディングを示します。『Oracle9*i* グローバリゼーション・サポート・ガイド』の付録 A の「キャラクタ・セット」に記載されているエンコーディングの他に、次のエンコーディングのドキュメントをサポートします。

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

デフォルト: デフォルトでは、UTF-8 がエンコーディングになります。シングルバイト・キャラクタ・セット（US-ASCII、ISO-8859 キャラクタ・セットのいずれか）のみを使用している場合、明示的にデフォルトのエンコーディングを設定することをお薦めします。UTF-8 などのマルチバイト・キャラクタ・セットを使用した場合より、パフォーマンスが 25% 向上します。

XML C++ Class Generator への出力

XML Parser for C++ 出力は、DTD 後に名前付けされた .cpp と .h の C++ ソース・ファイルの組です。各クラス（要素）用に提供されているコンストラクタを使用すると、オブジェクトを作成できます。オブジェクトを作成するには、最初は空で作成し、その後に子またはデータを追加するか、または最初から子や初期データの完全なセットで作成する 2 つの方法があります。#PCDATA（および Mixed）要素用に提供されているメソッドを使用すると、データを設定したり、適切な場合は、要素の属性を設定することができます。

標準への準拠

XML C++ Class Generator は、次の標準に準拠しています。

- W3C の XML 1.0 勧告
- W3C の DOM レベル 1 1.0 勧告
- W3C の XML Namespace 勧告
- SAX 1.0

ディレクトリ構造

XML C++ Class Generator には、次のファイルおよびディレクトリ構造があります。

- `license.html`: ライセンス契約
- `bin/`: スタンドアロン Class Generator "xmlcg"
- `doc/`: API ドキュメント
- `include/`: ヘッダー・ファイル
- `lib/`: XML およびサポートするライブラリ
- `mesg/`: エラー・メッセージ・ファイル（.msg には原因 / 処置に関する情報が含まれる）
- `sample/`: 使用例

表 F-6 に、XML C++ Class Generator に含まれているライブラリを示します。

表 F-6 XML C++ Class Generator ライブラリ

XML C++ Class Generator ライブラリ	説明
libxml8.a	XML パーサー /XSL プロセッサ
libxmlg8.a	XML Class Generator
libxmlc8.a	Oracle8*i* リリース 8.1.5 にリンクする必要がある互換性ライブラリ
libcore8.a	CORE ファンクション
libnls8.a	National Language Support

G

XDK for PL/SQL: 仕様および早見表

この付録では、Oracle XDK for PL/SQL の仕様および構文の早見表を示します。この付録の内容は次のとおりです。

XML Parser for PL/SQL

XML 文書は、エンティティというデータの格納単位で構成され、各エンティティには解析対象または解析対象外のデータが含まれます。解析対象データは文字で構成され、中にはドキュメント内の文字データを形成したり、タグを形成するものもあります。タグ付けは、ドキュメントのデータの格納レイアウトおよび論理構造の記述をエンコーディングします。XML は、データ格納レイアウトおよび論理構造を制約するメカニズムを提供します。

XML 文書の読取り、およびそのドキュメントのコンテンツと構造へのアクセスには、XML プロセッサというソフトウェア・モジュールが使用されます。XML プロセッサは、アプリケーションという他のモジュールのかわりに作業を行います。

Oracle XML Parser の機能

XML Parser for PL/SQL は、アプリケーションで処理できるように、XML 文書（または DTD）を解析します。ライブラリおよびコマンドラインは、次の標準および機能をサポートしています。

- DOM サポートは、W3C の DOM 1.0 勧告に準拠しています。これらの API によってアプリケーションは、XML 文書をツリー構造としてメモリー内でアクセスし操作できます。このインタフェースは、エディタなどのアプリケーションで使用されます。
- SAX のサポートは、SAX 1.0 仕様に準拠しています。これらの API によってアプリケーションは、イベント駆動モデルを使用して XML 文書を処理できます。
- XML Namespace 1.0 もサポートされています。これによって名前の競合が回避され、再利用性が向上し、アプリケーション統合が容易になります。
- Oracle および OracleAS を実行できます。
- C および C++ バージョンは、Windows、Solaris および Linux で使用できます。

その他にも次の機能があります。

- 検証モードおよび非検証モード
- 致命的なエラーが発生するまでの組込みエラー・リカバリ
- Oracle XSLT Processor によるドキュメント作成のための DOM 拡張 API

Oracle XML Parser には、XSL スタイルシートを使用して XML データを変換するための XSLT プロセッサが統合されています。XSLT プロセッサを使用すると、XML 文書を XML、HTML またはほぼすべてのテキストベース形式に変換できます。XSLT プロセッサは、次の標準および機能をサポートします。

- W3C の XSL Transform 1.0 勧告案に準拠しています。
- W3C の XPath 1.0 勧告案に準拠しています。
- XML パーサーに統合され、パフォーマンスおよびスケーラビリティを向上します。
- Java、C、C++ および PL/SQL 用のライブラリおよびコマンドラインで使用可能です。

XML 名前空間のサポート

XML Parser for Java、XML Parser for PL/SQL、XML Parser for C および XML Parser for C++ は XML 名前空間もサポートしています。XML 名前空間は、XML 文書内にある要素型（タグ）または属性間の名前の競合を解決または回避するメカニズムです。このメカニズムは、汎用の名前空間要素型および属性名を提供します。このマニュアルでは、これらの一部のみを説明しています。タグは、<oracle:EMP xmlns:oracle="http://www.oracle.com/xml"/> などの URI で修飾されています。たとえば、名前空間を使用して、オラクル社の <EMP> データ要素を、他社の <EMP> データ要素の定義と区別して識別できます。これによってアプリケーションは、処理する要素および属性をより簡単に識別できます。XML Parser for Java、XML Parser for C および XML Parser for C++ は、汎用の要素型と属性名、およびローカルの非修飾の要素型と属性名を識別および解析できるようにすることで、名前空間をサポートします。

検証モードおよび非検証モードのサポート

XML Parser for Java、XML Parser for PL/SQL、XML Parser for C および XML Parser for C++ は、検証モードまたは非検証モードで XML を解析できます。非検証モードでは、これらのパーサーは、XML が整形式であることを確認し、データを、DOM API が操作できるオブジェクトのツリーに解析します。検証モードでは、これらのパーサーは、XML が整形式であることを確認し、その XML データを DTD（存在する場合）に対して検証します。検証には、属性名および要素タグが正当であるかどうか、ネストした要素が適切な場所にあるかどうかなどの確認も含まれます。

サンプル・コード

サンプル・コードおよびXMLパーサーの使用方法については、第29章「XML Parser for PL/SQLの使用」を参照してください。

XML Parser for PL/SQLのディレクトリ構造

XML Parser for PL/SQLの$ORACLE_HOME/xdk/plsql/parser内のディレクトリ構造は、次のとおりです。

- Windows NT
  - license.html - ライセンス契約のコピー
  - readme.html - リリース・ノートおよびインストレーション・ノート
  - doc¥ - パーサーAPIのディレクトリ
  - lib¥ - パーサーのSQLおよびクラス・ファイルのディレクトリ
  - sample¥ - サンプル・コード・ファイル
- UNIX
  - license.html - ライセンス契約のコピー
  - readme.html - リリース・ノートおよびインストレーション・ノート
  - doc/ - パーサーAPIのディレクトリ
  - lib/ - パーサーのSQLおよびクラス・ファイルのディレクトリ
  - sample/ - サンプル・コード・ファイル

DOM APIおよびSAX API

XML APIは、通常、イベントベースおよびツリーベースの2つのカテゴリに分類されます。SAXなどのイベントベースAPIは、コールバックを使用してアプリケーションに解析イベントを通知します。アプリケーションは、カスタマイズしたイベント・ハンドラによってこれらのイベントを処理します。イベントには、要素および文字の開始および終了が含まれます。イベントベースAPIは、ツリーベースAPIとは異なり、通常、XML文書のメモリー内ツリー表現を構築しません。したがって、SAXは、XMLツリーの操作が必要でない検索操作などのアプリケーションに有効です。たとえば、次のXML文書を考えてみます。

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
     <ENAME>MARTIN</ENAME>
    </EMP>
    <EMP>
     <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

この XML 文書は、次のような一連の線形イベントになります。

```
start document
start element: EMPLIST
start element: EMP
start element: ENAME
characters: MARTIN
end element: EMP
start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP
end element: EMPLIST
end document
```

DOM などのツリーベースの API は、XML 文書のメモリー内にツリーを構築します。この API は、アプリケーションがツリーを操作および処理するためのクラスおよびメソッドを提供します。通常、DOM インタフェースは、要素の再順序付け、要素および属性の追加および削除、要素の名前の変更などの XML ツリーの構造的な操作に最も有効です。

XML Parser for PL/SQL の仕様

XML Parser for PL/SQL の仕様は次のとおりです。

- 検証および非検証モードをサポートします。
- 致命的エラーが発生するまで組込みエラー・リカバリを行います。
- W3C の XML 1.0 勧告をサポートします。
- W3C の XSLT 最終草案をサポートします。

XML プロセッサ（またはパーサー）の PL/SQL 実装は、W3C の XML 仕様（REC-xml-19980210 改訂）に準拠し、アプリケーションに提供する必要がある XML データおよび情報の読取り方法の点で、XML プロセッサに必要な動作を含みます。

XML Parser for PL/SQL: デフォルト動作

XML Parser for PL/SQL のデフォルト動作は、次のとおりです。

- DOM API によってアクセス可能な解析済ツリーが構築されます。
- DTD が検出された場合、XML Parser for PL/SQL は妥当性の検証を行います。検出されない場合は、検証を行わないパーサーになります。
- エラー・ログが指定されないかぎり、エラーは記録されません。ただし、解析が正常に実行されない場合、アプリケーション・エラーが発生します。

このマニュアルに記載されている型およびメソッドは、PL/SQL パッケージ xmlparser に付属しています。

- 統合された DOM レベル 1.0 API

キャラクタ・セット・エンコーディングのサポート

次の Oracle データベース・エンコーディングのドキュメントをサポートします。

- BIG 5
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1 ～ ISO-8859-9
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8

デフォルト: デフォルトでは、UTF-8 がエンコーディングになります。Oracle データベースがサポートする他の ASCII ベースまたは EBCDIC ベースのエンコーディングも使用できます。

要件

Java オプションが使用可能な Oracle データベースが必要です。

オンライン・ドキュメント

Oracle XML Parser for PL/SQL のドキュメントは、インストール場所の doc/ ディレクトリにあります。また、『Oracle9*i* XML リファレンス』も参照してください。

リリース固有の注意事項

Oracle XML Parser for PL/SQL は、PL/SQL および Java で作成されています。XML 文書が整形式であるか、また妥当であるかどうか（オプション）を確認します。Oracle XML Parser for PL/SQL は、PL/SQL インタフェースでアクセス可能なオブジェクト・ツリーを構築します。

標準への準拠

XML Parser for PL/SQL は、次の標準に準拠しています。

W3C の XML 1.0 勧告
（http://www.w3.org/TR/1998/REC-xml-19980210 を参照）

W3C の DOM レベル 1 1.0 勧告
（http://www.w3.org/TR/REC-DOM-Level-1/ を参照）

エラー・リカバリ

Oracle XML Parser は、エラー・リカバリも行います。ほとんどのエラーからのリカバリを行い、致命的なエラーが発生するまでは処理を継続します。

注意：Windows 版および UNIX 版の内容は同じです。これらは、オペレーティング・システムの互換性およびユーザーにとっての便宜上、異なる形態でアーカイブされています。

XML Parser for PL/SQL: Parser() API

表 G-1 に、XML Parser for PL/SQL の Parser() API 関数を示します。

表 G-1 XML Parser for PL/SQL: Parser() API

Parser() 関数	説明
parse(VARCHAR2)	指定した URL ファイルに格納された XML を解析し、構築された DOM 文書を戻します。
newParser	新しいパーサーのインスタンスを戻します。
parse(Parser, VARCHAR2)	指定した URL/ ファイルに格納された XML を解析します。
parseBuffer(Parser, VARCHAR2)	指定したバッファに格納された XML を解析します。
parseClob(Parser, CLOB)	指定した CLOB に格納された XML を解析します。
parseDTD(Parser, VARCHAR2, VARCHAR2)	指定した URL ファイルに格納された XML を解析します。
parseDTDBuffer(Parser, VARCHAR2, VARCHAR2)	指定したバッファに格納された XML を解析します。
parseDTDClob(Parser, CLOB, VARCHAR2)	指定した CLOB に格納された XML を解析します。
setBaseDir(Parser, VARCHAR2)	関連 URL の解決に使用されるベース・ディレクトリを設定します。
showWarnings(Parser, BOOLEAN)	警告のオン / オフを切り替えます。
setErrorLog(Parser, VARCHAR2)	エラーが指定ファイルに送信されるように設定します。
setPreserveWhitespace(Parser, BOOLEAN)	空白保存モードを設定します。
setValidationMode(Parser, BOOLEAN)	検証モードを設定します。
getValidationMode(Parser)	検証モードを取得します。
setDoctype(Parser, DOMDocumentType)	DTD を設定します。
getDoctype(Parser)	DTD を取得します。
getDocument(Parser)	DOM 文書を取得します。
freeParser(Parser)	パーサー・オブジェクトを解放します。

XML Parser for PL/SQL: XSLT プロセッサ API

表 G-2 に、XML Parser for PL/SQL の XSLT プロセッサ API 関数を示します。

次のインタフェースについて示します。

- プロセッサ・インタフェース型 : Processor()
- スタイルシート・インタフェース型 : Stylesheet()

表 G-2 XML Parser for PL/SQL: XSLT Processor() API の機能

XSLT プロセッサ関数	説明
newProcessor	新しいプロセッサ・インスタンスを戻します。
processXSL(Processor, Stylesheet, DOMDocument)	指定した DOMDocument およびスタイルシートを使用して、入力された XML 文書を変換します。
processXSL(Processor, Stylesheet, DOMDocumentFragment)	指定した DOMDocumentFragment およびスタイルシートを使用して、入力された XML 文書フラグメントを変換します。
showWarnings(Processor, BOOLEAN)	警告のオン / オフを切り替えます。
setErrorLog(Processor, VARCHAR2)	エラーが指定ファイルに送信されるように設定します。
newStylesheet(DOMDocument, VARCHAR2)	指定した DOMDocument および参照 URL を使用して、新しいスタイルシートを戻します。
newStylesheet(VARCHAR2, VARCHAR2)	指定した入力および参照 URL を使用して、新しいスタイルシートを戻します。
transformNode(DOMNode, Stylesheet)	指定したスタイルシートを使用して、DOM ツリー内のノードを変換します。
selectNodes(DOMNode, VARCHAR2)	指定したパターンに一致する DOM ツリーのノードを選択します。
selectSingleNodes(DOMNode, VARCHAR2)	指定したパターンに一致するツリーの最初のノードを選択します。
valueOf(DOMNode, VARCHAR2)	指定したパターンに一致するツリーの最初のノード値を取り出します。
setParam(Stylesheet, VARCHAR2, VARCHAR2)	最上位のスタイルシートのパラメータ値を設定します。
removeParam(Stylesheet, VARCHAR2)	最上位のスタイルシートのパラメータを削除します。
resetParams(Stylesheet)	最上位のスタイルシートのパラメータをリセットします。
freeStylesheet(Stylesheet)	スタイルシート・オブジェクトを解放します。

表 G-2 XML Parser for PL/SQL: XSLT Processor() API の機能（続き）

XSLT プロセッサ関数	説明
freeProcessor(Processor)	プロセッサ・オブジェクトを解放します。

XML Parser for PL/SQL: W3C DOM API - 型

DOM は、HTML ドキュメントおよび XML 文書に対する API です。ドキュメントの論理構造、およびドキュメントへのアクセス方法および操作方法を定義します。DOM 仕様では、「ドキュメント」は広い意味で使用される用語です。XML は、様々なシステムに格納される様々な情報を表す方法として、ますます需要が高まっています。従来、この情報のほとんどは、ドキュメントではなくデータとして考えられてきましたが、XML では、データをドキュメントとして表し、このデータを管理するために DOM が使用されます。

XML Parser for PL/SQL の W3C DOM API の詳細は、OTN の Web サイト http://otn.oracle.com/tech/xml を参照してください

表 G-3 XML Parser for PL/SQL: W3C DOM API 型

DOM ノード型	DOMException 型	DOM インタフェース型
ELEMENT_NODE	INDEX_SIZE_ERR	DOMNode
ATTRIBUTE_NODE	DOMSTRING_SIZE_ERR	DOMNamedNodeMap
TEXT_NODE	HIERARCHY_REQUEST_ERR	DOMNodeList
CDATA_SECTION_NODE	WRONG_DOCUMENT_ERR	DOMAttr
ENTITY_REFERENCE_NODE	INVALID_CHARACTER_ERR	DOMCDataSection
ENTITY_NODE	NO_DATA_ALLOWED_ERR	DOMCharacterData
PROCESSING_INSTRUCTION_NODE	NO_MODIFICATION_ALLOWED_ERR	DOMComment
COMMENT_NODE	NOT_FOUND_ERR	DOMDocumentFragment
DOCUMENT_NODE	NOT_SUPPORTED_ERR	DOMElement
DOCUMENT_TYPE_NODE	INUSE_ATTRIBUTE_ERR	DOMEntity
DOCUMENT_FRAGMENT_NODE	INDEX_SIZE_ERR	DOMEntityReference
NOTATION_NODE	DOMSTRING_SIZE_ERR	DOMNotation
		DOMProcessingInstruction
		DOMText
		DOMImplementation
		DOMDocumentType
		DOMDocument

XML Parser for PL/SQL: W3C DOM API - ノード・メソッド、ノード型および DOM インタフェース型

ノード・メソッド

次に、DOM API のノード・メソッドを示します。

- FUNCTION isNull(n DOMNode) RETURN BOOLEAN;
- FUNCTION makeAttr(n DOMNode) RETURN DOMAttr;
- FUNCTION makeCDataSection(n DOMNode) RETURN DOMCDataSection;
- FUNCTION makeCharacterData(n DOMNode) RETURN DOMCharacterData;
- FUNCTION makeComment(n DOMNode) RETURN DOMComment;
- FUNCTION makeDocumentFragment(n DOMNode) RETURN DOMDocumentFragment;
- FUNCTION makeDocumentType(n DOMNode) RETURN DOMDocumentType;
- FUNCTION makeElement(n DOMNode) RETURN DOMElement;
- FUNCTION makeEntity(n DOMNode) RETURN DOMEntity;
- FUNCTION makeEntityReference(n DOMNode) RETURN DOMEntityReference;
- FUNCTION makeNotation(n DOMNode) RETURN DOMNotation;
- FUNCTION makeProcessingInstruction(n DOMNode) RETURN DOMProcessingInstruction;
- FUNCTION makeText(n DOMNode) RETURN DOMText;
- FUNCTION makeDocument(n DOMNode) RETURN DOMDocument;
- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2);
- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2);
- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB);
- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2, charset VARCHAR2);

- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2, charset VARCHAR2);
- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB, charset VARCHAR2);
- FUNCTION getNodeName(n DOMNode) RETURN VARCHAR2;
- FUNCTION getNodeValue(n DOMNode) RETURN VARCHAR2;
- PROCEDURE setNodeValue(n DOMNode, nodeValue IN VARCHAR2);
- FUNCTION getNodeType(n DOMNode) RETURN NUMBER;
- FUNCTION getParentNode(n DOMNode) RETURN DOMNode;
- FUNCTION getChildNodes(n DOMNode) RETURN DOMNodeList;
- FUNCTION getFirstChild(n DOMNode) RETURN DOMNode;
- FUNCTION getLastChild(n DOMNode) RETURN DOMNode;
- FUNCTION getPreviousSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getNextSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getAttributes(n DOMNode) RETURN DOMNamedNodeMap;
- FUNCTION getOwnerDocument(n DOMNode) RETURN DOMDocument;
- FUNCTION insertBefore(n DOMNode, newChild IN DOMNode, refChild IN DOMNode) RETURN DOMNode;
- FUNCTION replaceChild(n DOMNode, newChild IN DOMNode, oldChild IN DOMNode) RETURN DOMNode;
- FUNCTION removeChild(n DOMNode, oldChild IN DOMNode) RETURN DOMNode;
- FUNCTION appendChild(n DOMNode, newChild IN DOMNode) RETURN DOMNode;
- FUNCTION hasChildNodes(n DOMNode) RETURN BOOLEAN;
- FUNCTION cloneNode(n DOMNode, deep boolean) RETURN DOMNode;

DOM ノード型

次に、DOM API のノード型を示します。

- ELEMENT_NODE
- ATTRIBUTE_NODE
- TEXT_NODE
- CDATA_SECTION_NODE
- ENTITY_REFERENCE_NODE
- ENTITY_NODE
- PROCESSING_INSTRUCTION_NODE
- COMMENT_NODE
- DOCUMENT_NODE
- DOCUMENT_TYPE_NODE
- DOCUMENT_FRAGMENT_NODE
- NOTATION_NODE

DOMException 型

次に、DOMException 型を示します。

- INDEX_SIZE_ERR
- DOMSTRING_SIZE_ERR
- HIERARCHY_REQUEST_ERR
- WRONG_DOCUMENT_ERR
- INVALID_CHARACTER_ERR
- NO_DATA_ALLOWED_ERR
- NO_MODIFICATION_ALLOWED_ERR
- NOT_FOUND_ERR
- NOT_SUPPORTED_ERR
- INUSE_ATTRIBUTE_ERR

DOM インタフェース型

次に、DOM インタフェース型を示します。

- DOMNode
- DOMNamedNodeMap
- DOMNodeList
- DOMAttr
- DOMCDataSection
- DOMCharacterData
- DOMComment
- DOMDocumentFragment
- DOMElement
- DOMEntity
- DOMEntityReference
- DOMNotation
- DOMProcessingInstruction
- DOMText
- DOMImplementation
- DOMDocumentType
- DOMDocument

H

XSU の仕様および早見表

この付録の内容は次のとおりです。

XSU のインストール

XSU 配布パッケージの内容

表 H-1 に、XSU の配布アーカイブ（ZIP ファイル）の内容を示します。

表 H-1 XSU 配布パッケージの内容

ファイル（相対位置）	説明
relnotes.html	リリース・ノートです。
env.csh	このファイルはヘルパー csh シェル・スクリプトで、XSU を正しく実行するために必要な、すべての環境変数を設定できます。ユーザーは、ディレクトリ情報を正しく設定する必要があります（JDK のインストール場所の指定など）。
env.bat	このファイルは、Windows のプラットフォーム用に作成されていることを除き、env.csh と同じです。
lib/oraclexmlsql.jar	XSU 用のすべての Java ファンクションを含む jar ファイルです。
lib/xmlparserv2.jar	XSU に同梱された Oracle XML Parser V2 です。
lib/oraclexmlsqlload.csh (UNIX) lib/oraclexmlsqlload.bat (Windows)	Oracle データベースへの XSU のロードを支援する csh および bat スクリプトです。これらのスクリプトは、loadjava をコールして jar ファイルをデータベースにロードし、xmlgenpkg.sql を実行して PL/SQL フロントエンド・ラッパーを作成します。
lib/xmlgenpkg.sql	PL/SQL フロントエンド・ラッパーを作成するための SQL スクリプトを含みます。

XSU のインストール : 手順

XSU をインストールするには、次の手順に従います。

1. 必要なソフトウェアをロードしたことを確認します。
2. XSU ファイルを解凍します。
3. クライアント側の環境を正しく設定します。
   - CLASSPATH を設定します。
   - データベースが起動されているかどうかを確認します。
4. サーバー側の環境を正しく設定します。

OTN からダウンロードした XSU のインストール

OTN の Web サイト（http://otn.oracle.com）から、適切な配布パッケージのアーカイブをダウンロードし、ダウンロードしたアーカイブを解凍します。使用例に応じて、次のとおりインストールを実行します。

XSU のクライアント側のフロントエンドまたは Java API を使用する場合は、次の手順が必要です。

1. 環境を設定します。（CLASSPATH の設定）
   - **UNIX ユーザー** : env.csh 内のパス名が正しいことを確認し、env.csh ファイルをソースに指定します。csh または tcsh 以外のシェルを使用している場合、そのシェルの構文を使用するようにファイルを編集します。
   - **Windows ユーザー** : env.bat 内のパス名が正しいことを確認し、env.bat ファイルをソースに指定します。

XSU の PL/SQL API を使用するか、または XSU Java API に JSP を作成するには、次の手順を実行します。

1. `xsuload.xxx` 内の USER_PASSWORD マクロが、XSU をロードする正しいスキーマ名（デフォルトは「scott/tiger」）を指定していることを確認します。
   - **UNIX ユーザー** : xsuload.csh 内を確認します。
   - **Windows ユーザー** : xsuload.bat 内を確認します。
2. XSU をロードする Oracle データベースが Java 対応で、起動されていることを確認します。
3. 適切な xsuload.xxx ファイルを実行します。
4. Oracle XML Parser for Java をデータベースにロードします。Oracle XML Parser for Java がすでにデータベースにロードされている場合、パーサーをロードする xsuload.xxx の行をコメント・アウトできます。
5. XSU の Java クラス（xsu12.jar または xsu111.jar）をロードし、XSU の PL/SQL API をロードします（PL/SQL スクリプト `dbmsxsu.sql` を実行します）。

XSU 実行の要件

XSU には、xsu111.jar および xsu12.jar の 2 つのバージョンがあります。xsu111.jar は JDK 1.1.x と互換性があり、xsu12.jar は JDK 1.2 と互換性があります。

XSU は Oracle8*i* リリース 8.1.7 以上および Oracle にパッケージ化されています。XSU は、次の 3 つのファイルで構成されています。

- `$ORACLE_HOME/rdbms/jlib/xsu12.jar - XSU` を構成するすべての Java クラスを含みます。xsu12 には、JDK 1.2.x および JDBC 2.x が必要です。これは、Oracle にロードされる XSU のバージョンです。

- $ORACLE_HOME/rdbms/jlib/xsu111.jar - xsu12.jar と同じクラスを含みます。ただし、xsu111 には、JDK 1.1.x および JDBC 1.x が必要です。
- $ORACLE_HOME/rdbms/admin/dbmsxsu.sql - XSU の PL/SQL API を構築する SQL スクリプトです。xsu12.jar は、`dbmsxsu.sql` が実行される前にデータベースにロードする必要があります。

Oracle Installer は、デフォルトで XSU をファイル・システム（前述の指定場所）にインストールし、データベースにもロードします。

最初のインストール時に XSU をインストールしない場合は、後でインストールを実行できますが、操作は複雑になります。後で XSU をインストールする場合は、最初に XSU およびその固有コンポーネントをシステムにインストールします。これは、Oracle Installer を使用して実行できます。その後、次の手順を実行します。

1. XML Parser for Java をデータベースにロードしていない場合は、$ORACLE_HOME/xdk/lib を選択します。このディレクトリにある xmlparserv2.jar を、データベースにロードする必要があります。操作の詳細は、『Oracle9*i* Java Stored Procedures Developer's Guide』の「Loading JAVA Classes」を参照してください。
2. 次に、`$ORACLE_HOME/admin` を選択し、`catxsu.sql` スクリプトを実行します。

注意： XSU は、OTN の Web サイト http://otn.oracle.com/tech/xml でも入手できます。XSU のアップデートについては、この Web サイトを参照してください。

XSU の要件

XSU をインストールする前に、ニーズに応じて正しいバージョンの XSU を選択する必要があります。たとえば、JDK 1.1.x のみを使用する場合、xsu111.jar ファイルをダウンロードします。JDK および JDBC ドライバがない場合、それらを正しくダウンロードして、インストールする必要があります。

XSU ファイルの解凍

ZIP ファイルをダウンロードした後、C:¥xml などの任意のディレクトリに内容を解凍します。XSU のバージョンによって、ファイルは xsu111 または xsu112 というサブディレクトリに展開されます。

XSU for Java の早見表

次の表に、XSU の Java API クラスおよびメンバーを示します。

- 表 H-2「XSU Java API: OracleXMLQuery クラス」
- 表 H-3「XSU Java API: OracleXMLSave クラス」
- 表 H-4「XSU Java API: OracleXMLSQLException クラス」
- 表 H-5「XSU Java API: OracleXMLSQLNoRowsException クラス」

表 H-2 XSU Java API: OracleXMLQuery クラス

メソッド、コンストラクタなど	説明
クラス	
OracleXMLQuery	public class OracleXMLQuery extends java.lang.Object(java.lang.Object = oracle.xml.sql.query.OracleXMLQuery) SQL 問合せを指定して、データベースから XML を生成します。
フィールド	
DTD	public static final int DTD DTD の生成を指定します。
ERROR_TAG	public static final java.lang.String ERROR_TAG ERROR ドキュメントのデフォルトのタグ名を指定します。
MAXROWS_ALL	public static final int MAXROWS_ALL すべての行を結果に含めます。
MAXROWS_DEFAULT	public static final int MAXROWS_DEFAULT XSU V2.0 以降は使用できません。かわりに、MAXROWS_ALL を使用します。
MAXROWS_NONE	public static final int MAXROWS_NONE XSU V2.0 以降は使用できません。かわりに、0（ゼロ）を使用します。
NONE	public static final int NONE DTD を生成しないことを指定します。
ROW_TAG	public static final java.lang.String ROW_TAG ROW 要素のデフォルトのタグ名を指定します。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
ROWIDATTR_TAG	public static final java.lang.String ROWIDATTR_TAG ROW 要素のデフォルトのタグ名を指定します。
ROWSET_TAG	public static final java.lang.String ROWSET_TAG ドキュメントのデフォルトのタグ名を指定します。
SCHEMA	public static final int SCHEMA XML Schema を生成しないことを指定します。
SKIPROWS_ALL	public static final int SKIPROWS_ALL 結果内ですべての行をスキップします。
SKIPROWS_DEFAULT	public static final int SKIPROWS_DEFAULT XSU V2.0 以降は使用できません。かわりに、0（ゼロ）を使用します。
SKIPROWS_NONE	public static final int SKIPROWS_NONE XSU V2.0 以降は使用できません。かわりに、0（ゼロ）を使用します。
コンストラクタ	
OracleXMLQuery(Connection, ResultSet)	public OracleXMLQuery(java.sql.Connection conn, java.sql.ResultSet rset) OracleXMLQueryObject のコンストラクタです。 パラメータ : conn - データベース接続。 rset - JDBC 結果セット・オブジェクト。
OracleXMLQuery(Connection, String)	public OracleXMLQuery(java.sql.Connection conn, java.lang.String query) OracleXMLQueryObject のコンストラクタです。 パラメータ : conn - データベース接続。 query - SQL 問合せ文字列。
OracleXMLQuery(OracleXMLDataSet)	public OracleXMLQuery(oracle.xml.sql.dataset.OracleXMLDataSet dset) OracleXMLQueryObject のコンストラクタです。 パラメータ : conn - データベース接続。 dset - データ・セット。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
メソッド	
close()	public void close() Oracle XML エンジンで作成された、すべてのオープン・リソースをクローズします。ただし、ユーザーが提供するインスタンスの結果セットはクローズしません。
getNumRowsProcessed()	public long getNumRowsProcessed() 処理された行の数を戻します。 戻り値 : 処理された行の数。
getXML(OracleXMLDocGen, boolean)	public void getXML(oracle.xml.sql.docgen.OracleXMLDocGen doc, boolean withDTD) XSU V2.0 以降は使用できません。
getXMLDOM()	public org.w3c.dom.Document getXMLDOM() コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。 戻り値 : XML 文書の DOM 表現。
getXMLDOM(boolean)	public org.w3c.dom.Document getXMLDOM(boolean withDTD) XSU V1.2.1 以降は使用できません。かわりに、getXMLDOM(int) を使用します。
getXMLDOM(int)	public org.w3c.dom.Document getXMLDOM(int metaType) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。metaType 引数は、XSU が XML とともに生成する XML メタデータの型を指定するために使用されます。現在、この値は無視され、XML メタデータは生成されません。 パラメータ : metaType - XML メタデータの型（NONE、SCHEMA）。 戻り値 : XML 文書の文字列表現。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
getXMLDOM(Node)	public org.w3c.dom.Document getXMLDOM(org.w3c.dom.Node root) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。NULL でない場合、root 引数は、XML 文書のルート要素とみなされます。 パラメータ : root - 新しい XML を追加するルート・ノード。 戻り値 : XML 文書の文字列表現。
getXMLDOM(Node, int)	public org.w3c.dom.Document getXMLDOM(org.w3c.dom.Node root, int metaType) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。NULL でない場合、root 引数は、XML 文書のルート要素とみなされます。metaType 引数は、XSU が XML とともに生成する XML メタデータの型を指定するために使用されます。現在、この値は無視され、XML メタデータは生成されません。 パラメータ : root - 新しい XML を追加するルート・ノード。 metaType - XML メタデータの型（NONE、SCHEMA）。 戻り値 : XML 文書の文字列表現。
getXMLMetaData(int, boolean)	public java.lang.String getXMLMetaData(int metaType, boolean withVer) getXML のコールによって生成された可能性がある XML 文書の DTD または XML Schema を戻します。metaType パラメータは、生成する XML メタデータの型を指定します。withVer パラメータは、バージョン・ヘッダーを生成するかどうかを指定します。 パラメータ : metaType - 生成する XML メタデータの型（NONE または DTD）。 withVer - バージョンの処理命令を生成するかどうか。
getXMLSAX(ContentHandler)	public void getXMLSAX(org.xml.sax.ContentHandler sax) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。 パラメータ : sax - 登録する ContentHandler オブジェクト。
getXMLSchema()	public org.w3c.dom.Document getXMLSchema() 指定した問合せに対応する XML Schema を生成します。 戻り値 : XML Schema

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
getXMLString()	public java.lang.String getXMLString() コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。 戻り値 : XML 文書の文字列表現。
getXMLString(boolean)	public java.lang.String getXMLString(boolean withDTD) XSU V1.2.1 以降は使用できません。かわりに、getXMLString(int) を使用します。
getXMLString(int)	public java.lang.String getXMLString(int metaType) getXML のコールによって生成された可能性がある XML 文書の DTD または XML Schema を戻します。metaType パラメータは、生成する XML メタデータの型を指定します。metaType 引数の有効値は、NONE および DTD（このクラスの静的フィールド）です。 パラメータ : metaType - XML メタデータの型（NONE、DTD または SCHEMA）。 戻り値 : XML 文書の文字列表現。
getXMLString(Node)	public java.lang.String getXMLString(org.w3c.dom.Node root) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。NULL でない場合、root 引数は、XML 文書のルート要素とみなされます。 パラメータ : root - 新しい XML を追加するルート・ノード。 戻り値 : XML 文書の文字列表現。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
getXMLString(Node, int)	public java.lang.String getXMLString(org.w3c.dom.Node root, int metaType) コンストラクタで指定されたオブジェクト・リレーショナル・データを、XML 文書に変換します。NULL でない場合、root 引数は、XML 文書のルート要素とみなされます。metaType 引数は、XSU が XML とともに生成する XML メタデータの型を指定するために使用されます。metaType 引数の有効値は、NONE および DTD（このクラスの静的フィールド）です。root 引数が NULL でない場合、DTD は要求されても生成されません。 パラメータ : root - 新しい XML を追加するルート・ノード。 metaType - XML メタデータの型（NONE、DTD または SCHEMA）。 戻り値 : XML 文書の文字列表現。
keepCursorState(boolean)	public void keepCursorState(boolean alive) XSU V1.2.1 以降は使用できません。かわりに、keepObjectOpen を使用します。
keepObjectOpen(boolean)	public void keepObjectOpen(boolean alive) ResultSet オブジェクトを取らないすべての getXML ファンクションのデフォルトの動作は、結果セット・オブジェクトおよび文オブジェクトをコールの終わりにクローズすることです。getXML を繰り返しコールして次の行セットを取得する永続機能を使用する必要がある場合、このファンクションを true に設定してコールすることで、このデフォルトの動作を停止する必要があります。OracleXMLQuery は、getXML のコール後、結果セット・オブジェクトおよび文オブジェクトをクローズしません。close() をコールして、明示的にカーソル状態をクローズします。 パラメータ : alive - オブジェクトのオープン状態を保持するかどうか。
removeXSLTParam(String)	public void removeXSLTParam(java.lang.String name) 最上位のスタイルシートのパラメータ値を削除します。スタイルシートが登録されてない場合、このメソッドは動作しません。 パラメータ : name - パラメータ名。
setCollIdAttr(String)	public void setCollIdAttr(java.lang.String collIdAttr) XSU V1.2.1 以降は使用できません。かわりに、setCollIdAttrName を使用します。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
setCollIdAttrName(String)	public void setCollIdAttrName(java.lang.String attrName) コレクション要素の区切りタグの ID 属性名を設定します。このタグに、NULL または空の文字列を設定すると、ROWID の属性が省略されます。 パラメータ : attrName - 属性名。
setDataHeader(Reader, String)	public void setDataHeader(java.io.Reader header, java.lang.String docTag) XML データ・ヘッダーを設定します。このデータ・ヘッダーは、XML のエンティティであり、問合せによって生成された XML エンティティ（行セット）の先頭に追加されます。2 つのエンティティは、docTag 引数で指定されたタグで囲まれます。最後に指定したデータ・ヘッダーが、使用されるデータ・ヘッダーです。また、ヘッダー・パラメータに NULL を指定すると、データ・ヘッダーの設定が解除されます。 パラメータ : header - ヘッダー。 tag - データ・ヘッダーおよび行セットを囲むタグ。
setDateFormat(String)	public void setDateFormat(java.lang.String mask) XML 文書に生成日付の書式を設定します。日付書式パターン（日付マスク）の構文は、java.text.SimpleDateFormat クラスの要件を満たす必要があります。マスクに NULL または空の文字列を設定すると、日付マスクの設定が解除されます。 パラメータ : mask - 日付マスク。
setEncoding(String)	public void setEncoding(java.lang.String enc) XML 文書にエンコーディングを設定します。エンコーディングとして NULL または空の文字列が指定された場合、デフォルトのキャラクタ・セットが処理命令のエンコーディングに指定されます。 パラメータ : enc - XML 文書のキャラクタ・セットのエンコーディング。
setErrorTag(String)	public void setErrorTag(java.lang.String tag) XML のエラー・ドキュメントを囲むタグを設定します。 パラメータ : tag - タグ名。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
setException(Exception)	public void setException(java.lang.Exception e) ユーザーに例外の指定を許可し、XSU に対処させます。 パラメータ : e - XSU が処理する例外。
setMaxRows(int)	public void setMaxRows(int rows) XML に変換する行の最大数を設定します。デフォルトでは、最大数は設定されていません。無制限の最大数を明示的に指定する方法は、MAXROWS_ALL を参照してください。 パラメータ : rows - 生成する行の最大数。
setMetaHeader(Reader)	public void setMetaHeader(java.io.Reader header) XML メタ・ヘッダーを設定します。ヘッダーが設定されると、このオブジェクトが生成した各 XML 文書のメタデータ部（DTD または XML Schema）の先頭に挿入されます。最後に指定したメタ・ヘッダーが使用されることに注意してください。また、ヘッダー・パラメータに NULL を指定すると、メタ・ヘッダーの設定が解除されます。 パラメータ : header - ヘッダー。
setRaiseException(boolean)	public void setRaiseException(boolean flag) 例外を発生させるように XSU に指示します。このコールが行われなかった場合、または flag 引数を false に指定した場合、XSU は SQL 例外をキャッチし、その例外メッセージから XML 文書を生成します。 パラメータ : flag - 例外を発生させるかどうか。
setRaiseNoRowsException(boolean)	public void setRaiseNoRowsException(boolean flag) 生成された XML 文書が空の場合に、理由にかかわらず OracleXMLNoRowsException が発生するかどうかを XSU に通知します。デフォルトでは、例外は発生しません。 パラメータ : flag - データが見つからなかった場合に、OracleXMLNoRowsException が発生するかどうか。
setRowIdAttrName(String)	public void setRowIdAttrName(java.lang.String attrName) 行の囲みタグの ID 属性名を設定します。このタグに、NULL または空の文字列を設定すると、ROWID 属性が省略されます。 パラメータ : attrName - 属性名。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
setRowIdAttrValue(String)	public void setRowIdAttrValue(java.lang.String colName) 列値が行の囲みタグの ID 属性に割り当てられる値スカラー列を指定します。colName に NULL または空の文字列を指定すると、ROWID 属性に行カウントの値（0、1、2 など）が割り当てられます。 パラメータ : colName - 列値が ROWID 属性に割り当てられる列。
setRowIdColumn(String)	public void setRowIdColumn(java.lang.String colName) XSU V1.2.1 以降は使用できません。かわりに、setRowIdAttrValue を使用します。
setRowsetTag(String)	public void setRowsetTag(java.lang.String tag) XML データセットを囲むタグを設定します。 パラメータ : tag - タグ名。
setRowTag(String)	public void setRowTag(java.lang.String tag) データベース・レコードに対応する XML 要素を囲むタグを設定します。 パラメータ : tag - タグ名。
setSkipRows(int)	public void setSkipRows(int rows) スキップする行数を設定します。デフォルトでは、0（ゼロ）行がスキップされます。すべての行をスキップするには、SKIPROWS_ALL を使用します。 パラメータ : rows - スキップする行数。
setStyleSheet(String)	public void setStyleSheet(java.lang.String uri) XSU V2.0 以降は使用できません。かわりに、setStylesheetHeader を使用します。
setStyleSheet(String, String)	public void setStyleSheet(java.lang.String uri, java.lang.String type) XSU V2.0 以降は使用できません。かわりに、setStylesheetHeader を使用します。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
setStylesheetHeader(String)	public void setStylesheetHeader(java.lang.String uri) 生成された XML 文書にスタイルシート・ヘッダー（スタイルシート処理命令）を設定します。uri 引数に NULL を指定すると、スタイルシート・ヘッダーおよびスタイルシートの型の設定が解除されます。 パラメータ : uri - スタイルシートの URI。
setStylesheetHeader(String, String)	public void setStylesheetHeader(java.lang.String uri, java.lang.String type) 生成された XML 文書にスタイルシート・ヘッダー（スタイルシート処理命令）を設定します。uri 引数に NULL を指定すると、スタイルシート・ヘッダーおよびスタイルシートの型の設定が解除されます。 パラメータ : uri - スタイルシートの URI。 type - スタイルシートの型（デフォルトは「text/xsl」）。
setXSLT(Reader, String)	public void setXSLT(java.io.Reader stylesheet, java.lang.String ref) 生成された XML に適用する XSL 変換を登録します。スタイルシートがすでに登録されている場合は、新しいスタイルシートに置き換えられます。スタイルシートの登録を解除するには、stylesheet 引数に NULL を指定します。 パラメータ : stylesheet - スタイルシート。 ref - 挿入、インポートおよび外部エンティティの URL。
setXSLT(String, String)	public void setXSLT(java.lang.String stylesheet, java.lang.String ref) 生成された XML に適用する XSL 変換を登録します。スタイルシートがすでに登録されている場合は、新しいスタイルシートに置き換えられます。スタイルシートの登録を解除するには、stylesheet 引数に NULL を指定します。 パラメータ : stylesheet - スタイルシート。 ref - 挿入、インポートおよび外部エンティティの URL。

表 H-2 XSU Java API: OracleXMLQuery クラス（続き）

メソッド、コンストラクタなど	説明
setXSLTParam(String, String)	public void setXSLTParam(java.lang.String name, java.lang.String value) 最上位のスタイルシートのパラメータ値を設定します。パラメータ値は、有効な XPath 式が想定されます（したがって、文字列リテラル値を明示的に囲む必要があります)。スタイルシートが登録されてない場合、このメソッドは動作しません。 パラメータ : name - パラメータ名。 value - XPath 式としてのパラメータ値。
useLowerCaseTagNames()	public void useLowerCaseTagNames() すべてのタグ名を小文字に設定します。すべての必要なタグを設定してから、このメソッドをコールします。
useNullAttributeIndicator(boolean)	public void useNullAttributeIndicator(boolean flag) NULL を示すために、XML 属性を使用するか、XML 文書内の特定のエンティティを省略するかを指定します。 パラメータ : flag - NULL を示す属性を使用するかどうか。
useTypeForCollElemTag(boolean)	public void useTypeForCollElemTag(boolean flag) デフォルトでは、コレクション要素のタグ名は、コレクションのタグ名の後に「_item」が続く名前になります。引数に true を指定してこのメソッドをコールすると、XSU はコレクション要素の型名をコレクション要素のタグ名として使用します。 パラメータ : flag - コレクション要素の型を、そのタグ名として使用するかどうか。
useUpperCaseTagNames()	public void useUpperCaseTagNames() すべてのタグ名を大文字に設定します。すべての必要なタグを設定してから、このメソッドをコールします。

表 H-3 XSU Java API: OracleXMLSave クラス

メソッド、コンストラクタなど	説明
クラス	
OracleXMLSave	public class OracleXMLSave extends java.lang.Object(java.lang.Object = oracle.xml.sql.dml.OracleXMLSave) XML からオブジェクト・リレーショナル表またはオブジェクト・リレーショナル・ビューへの正規マッピングをサポートします。挿入、更新および削除がサポートされます。最初に、DML 操作を実行する必要がある表名を渡すことによって、クラスを作成します。作成後、ユーザーはこの表に対して自由に挿入、更新および削除を実行できます。このクラスで提供されている有効なファンクションは、更新または削除のためのキー列を識別したり、更新中の列を制限するために有効です。
フィールド	
DATE_FORMAT	public static final java.lang.String DATE_FORMAT setDateFormat で使用される日付書式です。
DEFAULT_BATCH_SIZE	public static int DEFAULT_BATCH_SIZE 挿入時のデフォルトのバッチ・サイズは 17 です。
コンストラクタ	
OracleXMLSave(Connection, String)	public OracleXMLSave(java.sql.Connection oconn, java.lang.String tabName) Save クラスに対するパブリック・コンストラクタです。 パラメータ : oconn - 接続オブジェクト（データベースへの接続）。 tabName - 更新する必要がある表の名前。
メソッド	
cleanLobList()	public void cleanLobList()
close()	public void close() このオブジェクトに対応付けられたすべてのコンテキストのクローズまたは割当て解除を行います。
createURL(String)	public java.net.URL createURL(java.lang.String fileName) XSU V2.0 以降は使用できません。かわりに、このメソッドの静的なバージョンを使用します。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
deleteXML(Document)	public int deleteXML(org.w3c.dom.Document doc) XML 文書に基づいて表の行を削除します。 パラメータ : xmlDoc - DOM 形式の XML 文書。 戻り値 : 処理された XML の ROW 要素の数 参照 : deleteXML(URL)。
deleteXML(InputStream)	public int deleteXML(java.io.InputStream xmlStream) XML 文書に基づいて表の行を削除します。 パラメータ : xmlDoc - ストリーム形式の XML 文書。 戻り値 : 処理された XML の ROW 要素の数。 参照 : deleteXML(URL)。
deleteXML(Reader)	public int deleteXML(java.io.Reader xmlStream) XML 文書に基づいて表の行を削除します。 パラメータ : xmlDoc - ストリーム形式の XML 文書。 戻り値 : 処理された XML の ROW 要素の数。 参照 : deleteXML(URL)。
deleteXML(String)	public int deleteXML(java.lang.String xmlDoc) XML 文書に基づいて表の行を削除します。 パラメータ : xmlDoc - 文字列形式の XML 文書。 戻り値 : 処理された XML の ROW 要素の数。 参照 : deleteXML(URL)。
deleteXML(URL)	public int deleteXML(java.net.URL url) 提供された XML 文書内の要素の値に基づいて、指定された表から行を削除します。デフォルトでは、削除処理によって、すべての要素値とそれに対応する列名が一致します。入力ドキュメントの各 ROW 要素は、表に対する個別の DELETE 文として受け入れられます。 パラメータ : url - 表の行の削除に使用するドキュメントの URL。 戻り値 : 処理された XML の行要素の数。この数は、XML 文書を介して選択された行が、表内の行を一意に識別するかどうかによって削除されたデータベースの行数と同等になる場合とならない場合があります。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
finalize()	protected void finalize() オーバーライド先 : java.lang.Object クラスの java.lang.Object.finalize() をオーバーライドします。
getURL(String)	public static java.net.URL getURL(java.lang.String target) 指定されたファイル名または URL の URL オブジェクトを戻します。渡された引数が有効な URL 形式（http:// や file:// など）でない場合、このメソッドは引数に「file://」を追加して、引数の修正を試みます。NULL または空の文字列が渡された場合、NULL を戻します。 パラメータ : target - ファイル名または URL 文字列。 戻り値 : ターゲット・エンティティを識別する URL オブジェクト。
insertXML(Document)	public int insertXML(org.w3c.dom.Document doc)
insertXML(InputStream)	public int insertXML(java.io.InputStream xmlStream)
insertXML(Reader)	public int insertXML(java.io.Reader xmlStream)
insertXML(String)	public int insertXML(java.lang.String xmlDoc)
insertXML(URL)	public int insertXML(java.net.URL url) 指定した URL から指定した表に XML 文書を挿入します。デフォルトでは、挿入ルーチンが、要素名と列名を一致させることによって表に値を挿入し、その入力ドキュメント内で欠落したすべての要素に NULL 値を挿入します。setUpdateColumnList() を使用して挿入する列のリストを設定すると、これらの列にのみ値が挿入され、その他の列にはデフォルト値が挿入されるように設定できます。詳細は、第 7 章「XSU」および『Oracle9*i* XML リファレンス』を参照してください。 パラメータ : url - 表への行の挿入に使用するドキュメントの URL。 戻り値 : 挿入された行の数。
removeXSLTParam(String)	public void removeXSLTParam(java.lang.String name) 最上位のスタイルシートのパラメータ値を削除します。スタイルシートが登録されてない場合、このメソッドは動作しません。 パラメータ : name - パラメータ名。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
setBatchSize(int)	public void setBatchSize(int size) DML 操作中に使用されるバッチ・サイズを変更します。挿入、更新および削除を実行する場合、別々の文ではなく、1 回の処理でデータベースの操作が実行されるように、バッチ処理を行う方が適しています。ただし、その操作を実行する前に、すべてのバインド値を保持するためのメモリーがさらに多く必要になります。バッチ処理を行う場合、バッチ処理の実行後にのみコミットが発生することに注意してください。そのため、バッチ内の 1 文が正常に実行されなかった場合、バッチ処理全体がロールバックされます。この動作を防止するには、バッチ・サイズを 1 に設定します。デフォルトのバッチ・サイズは DEFAULT_BATCH_SIZE です。 パラメータ : size - すべての DML に使用するバッチ・サイズ
setCommitBatch(int)	public void setCommitBatch(int size) コミットのバッチ・サイズを設定します。コミットのバッチ・サイズとは、実行後にコミットする必要がある挿入の回数または記録を意味します。commitBatch が 1 未満またはセッションが自動コミット・モードの場合、XSU は明示的なコミットを行いません。コミットのデフォルトのバッチ・サイズは 0（ゼロ）です。 パラメータ : size - コミットのバッチ・サイズ。
setDateFormat(String)	public void setDateFormat(java.lang.String mask) XML 文書の日付書式を、XSU に通知します。デフォルトでは、`OracleXMLSave` は、日付書式を 'MM/dd/yyyy HH:mm:ss' であると想定しています。このファンクションをコールすると、このデフォルトの書式をオーバーライドできます。日付書式パターン（日付マスク）の構文は、`java.text.SimpleDateFormat` クラスの要件を満たす必要があります。マスクを NULL または空の文字列に設定すると、デフォルトのマスク `OracleXMLSave.DATE_FORMAT` が使用されます。 パラメータ : mask - 日付マスク。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
setIgnoreCase(boolean)	public void setIgnoreCase(boolean ignore) XSU は、要素名（XML タグ）に基づいて XML 要素をデータベースの列または属性にマップします。このファンクションは、XSU が大文字 / 小文字を区別しないように通知します。このように大文字 / 小文字の区別をなくすと、保存オブジェクトの作成時に実行されるメタデータのキャッシュに影響を与える場合があります。 パラメータ : ignore - XML 文書内のタグで大文字 / 小文字を無視するかどうか（0 - FALSE、1 - TRUE）。
setKeyColumnList(String[])	public void setKeyColumnList(java.lang.String[] keyColNames) 更新または削除中に、データベース表内の特定の行を識別するために使用する、列のリストを設定します。このコールは、挿入の場合は無視されます。更新を実行する前に、キー列を指定する必要があります。削除用のオプションです。このキー列を指定すると、XML 文書内のこれらのタグの値は、更新または削除するデータベースの行を識別するために使用されます。現在、XML 文書内でキー列の大文字 / 小文字を指定できないため、キー列自体の値を更新する方法はありません。 パラメータ : keyColNames - キーとして使用される列のリスト名。
setRowTag(String)	public void setRowTag(java.lang.String rowTag) XML 文書で使用するタグを指定し、各行の値に対応する XML 要素を囲みます。この値を NULL に指定すると、ROW タグが存在せず、ドキュメントの最上位の要素が行自体に対応することを表します。 パラメータ : rowtag - タグ名。
setUpdateColumnList(String[])	public void setUpdateColumnList(java.lang.String[] updColNames) 更新する列の値を設定します。これは、挿入および更新でのみ有効です。削除では無視されます。挿入の場合、デフォルトでは、表のすべての列に値が挿入されます。更新の場合、デフォルトでは XML 文書の ROW 要素にあるタグに対応する列のみが更新されます。列を指定すると指定した列のみが、UPDATE 文または INSERT 文で更新されます。ドキュメント内の他のすべての要素は無視されます。 パラメータ : updColNames - 更新する列の文字列リスト。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
setXSLT(Reader, String)	public void setXSLT(java.io.Reader stylesheet, java.lang.String ref) 生成された XML に適用する XSL 変換を登録します。スタイルシートがすでに登録されている場合は、新しいスタイルシートに置き換えられます。スタイルシートの登録を解除するには、stylesheet 引数に NULL を指定します。 パラメータ : stylesheet - スタイルシート。 ref - 挿入、インポートおよび外部エンティティの URL。
setXSLT(String, String)	public void setXSLT(java.lang.String stylesheet, java.lang.String ref) 生成された XML に適用する XSL 変換を登録します。スタイルシートがすでに登録されている場合は、新しいスタイルシートに置き換えられます。スタイルシートの登録を解除するには、stylesheet 引数に NULL を指定します。 パラメータ : stylesheet - スタイルシート。 ref - 挿入、インポートおよび外部エンティティの URL。
setXSLTParam(String, String)	public void setXSLTParam(java.lang.String name, java.lang.String value) 最上位のスタイルシートのパラメータ値を設定します。パラメータ値は、有効な XPath 式が想定されます（したがって、文字列リテラル値を明示的に囲む必要があります）。スタイルシートが登録されてない場合、このメソッドは動作しません。 パラメータ : name - パラメータ名。 value - XPath 式としてのパラメータ値。
updateXML(Document)	public int updateXML(org.w3c.dom.Document doc) DOM ツリー形式の XML 文書を指定して、表を更新します。 パラメータ : xmlDoc - DOM ツリー形式の XML 文書。 戻り値 : 処理された XML 要素の数。 参照 : updateXML(URL)。

表 H-3 XSU Java API: OracleXMLSave クラス（続き）

メソッド、コンストラクタなど	説明
updateXML(InputStream)	public int updateXML(java.io.InputStream xmlStream) ストリーム形式の XML 文書を指定して、表を更新します。 パラメータ : xmlStream - ストリーム形式の XML 文書。 戻り値 : 処理された XML 要素の数。 参照 : updateXML(URL)。
updateXML(Reader)	public int updateXML(java.io.Reader xmlStream) ストリーム形式の XML 文書を指定して、表を更新します。 パラメータ : xmlStream - ストリーム形式の XML 文書。 戻り値 : 処理された XML 要素の数。 参照 : updateXML(URL)。
updateXML(String)	public int updateXML(java.lang.String xmlDoc) 文字列形式の XML 文書を指定して、表を更新します。 パラメータ : xmlDoc - 文字列形式の XML 文書。 戻り値 : 処理された XML 要素の数。 参照 : updateXML(URL)。
updateXML(URL)	public int updateXML(java.net.URL url) 提供された XML 文書内の要素の値に基づいて、データベース内の列を更新します。更新には、指定された表内で更新する行を一意に識別するために使用する、キー列のリストが必要です。デフォルトでは、更新はキー列のリストを使用し、XML 文書内の対応する値と一致させて特定の行を識別した後、その XML 文書内に同等の要素が存在する表のすべての列を更新します。 パラメータ : url - 表の更新に使用するドキュメントの URL。 戻り値 : 処理された XML 行要素の数。この数は、XML 文書を介して選択された行が、表内の行を一意に識別するかどうかによって変更されたデータベースの行数と同等になる場合とならない場合があります。

表 H-4 XSU Java API: OracleXMLSQLException クラス

コンストラクタおよびメソッド	説明
クラス	
OracleXMLSQLException	public class OracleXMLSQLException extends java.lang.RuntimeException
コンストラクタ	
OracleXMLSQLException(Exception)	public OracleXMLSQLException(java.lang.Exception e)
OracleXMLSQLException(Exception, String)	public OracleXMLSQLException(java.lang.Exception e, java.lang.String errorTagName)
OracleXMLSQLException(String)	public OracleXMLSQLException(java.lang.String message)
OracleXMLSQLException(String, Exception)	public OracleXMLSQLException(java.lang.String message, java.lang.Exception e)
OracleXMLSQLException(String, Exception, String)	public OracleXMLSQLException(java.lang.String message, java.lang.Exception e, java.lang.String errorTagName)
OracleXMLSQLException(String, int)	public OracleXMLSQLException(java.lang.String message, int errorCode)
OracleXMLSQLException(String, int, String)	public OracleXMLSQLException(java.lang.String message, int errorCode, java.lang.String errorTagName)
OracleXMLSQLException(String, String)	public OracleXMLSQLException(java.lang.String message, java.lang.String errorTagName)
メソッド	
getErrorCode()	public int getErrorCode()
getParentException()	public java.lang.Exception getParentException() 元の例外がある場合、それを戻します。ない場合は、NULL を戻します。
getXMLErrorString()	public java.lang.String getXMLErrorString() XML エラー文字列を、指定したエラー・タグ名で出力します。
getXMLSQLErrorString()	public java.lang.String getXMLSQLErrorString() エラー・メッセージにも SQL パラメータを出力します。
setErrorTag(String)	public void setErrorTag(java.lang.String tagName) XML エラー・レポートを生成するために、getXMLErrorString および getXMLSQLErrorString で使用されるエラー・タグ名を設定します。

表 H-5 XSU Java API: OracleXMLSQLNoRowsException クラス

コンストラクタ	説明
クラス	
OracleXMLSQLNoRowsException	public class OracleXMLSQLNoRowsException extends OracleXMLSQLException
コンストラクタ	
OracleXMLSQLNoRowsException()	public OracleXMLSQLNoRowsException()
OracleXMLSQLNoRowsException(String)	public OracleXMLSQLNoRowsException(java.lang.String errorTag)

XSU for PL/SQL の早見表

XSU for PL/SQL は、次の PL/SQL パッケージを提供します。

- DBMS_XMLQuery（DB_to_XML 型の機能を提供します）
- DBMS_XMLSave（XML_to_DB 型の機能を提供します）

DBMS_XMLQuery PL/SQL パッケージ

表 H-6 に、DBMS_XMLQuery のプロシージャ、ファンクション、型および定数を示します。

表 H-6 DBMS_XMLQuery のプロシージャ、ファンクション、型および定数

プロシージャ、ファンクション、型および定数	説明
型	
ctxType	問合せのコンテキスト・ハンドルの型です。DBMS_ XMLQuery.newContext() の戻り型です。
定数	
DEFAULT_ROWSETTAG	ほとんどの場合、ルート・ノードのタグ名で、ROWSET と表されます。
DEFAULT_ERRORTAG	ERROR と表されます。
DEFAULT_ROWIDATTR	NUM と表されます。
DEFAULT_ROWTAG	ROW と表されます。
DEFAULT_DATE_FORMAT	'MM/dd/yyyy HH:mm:ss' と表されます。
ALL_ROWS	すべての行を出力する必要があることを示します。
NONE	DTD などを含めないことを指定します。
DTD	DTD の生成を要求します。
LOWER_CASE	小文字のタグを使用します。
UPPER_CASE	大文字のタグを使用します。
プロシージャ	
closeContext(ctxType)	特定の問合せコンテキストのクローズまたは割当て解除を行います。
ファンクション	
getDTD(ctxType, BOOLEAN := false)	

表 H-6 DBMS_XMLQuery のプロシージャ、ファンクション、型および定数 （続き）

プロシージャ、ファンクション、型および定数	説明
プロシージャ	
getDTD(ctxType, CLOB, BOOLEAN := false)	初期化に使用される SQL 問合せに基づいて、DTD を生成します。
getExceptionContent(ctxType, NUMBER, VARCHAR2)	-
ファンクション	
getXML(ctxType, NUMBER := NONE)	-
プロシージャ	
getXML(ctxType, CLOB, NUMBER := NONE)	XML 文書を生成します。
ファンクション	
newContext(VARCHAR2)	問合せコンテキストを作成し、コンテキスト・ハンドルを戻します。 戻り値 : ctxType。
ファンクション	
newContext(CLOB)	問合せコンテキストを作成し、コンテキスト・ハンドルを戻します。 戻り値 : ctxType。
プロシージャ	
propagateOriginalException(ctxType, BOOLEAN)	例外が発生した場合、例外を OracleXMLSQLException でラップするかわりに、呼び出された例外を発生させる必要があることを XSU に通知します。
setBindValue(ctxType, VARCHAR2, VARCHAR2)	特定したバインド名の値を設定します。
setCollIdAttrName(ctxType, VARCHAR2	コレクション要素の区切りタグの ID 属性名を設定します。
setDataHeader(ctxType, CLOB := null, VARCHAR2 := null)	XML データ・ヘッダーを設定します。
setDateFormat(ctxType, VARCHAR2)	XML 文書に生成日付の書式を設定します。
setErrorTag(ctxType, VARCHAR2)	XML のエラー・ドキュメントを囲むためのタグを設定します。

表 H-6 DBMS_XMLQuery のプロシージャ、ファンクション、型および定数 （続き）

プロシージャ、ファンクション、型および定数	説明
setMaxRows (ctxType, NUMBER)	XML に変換する行の最大数を設定します。
setMetaHeader(ctxType, CLOB := null)	XML メタ・ヘッダーを設定します。
setRaiseException(ctxType, BOOLEAN)	呼び出された例外を発生させるように XSU に通知します。
setRaiseNoRowsException(ctxType, BOOLEAN)	生成された XML 文書が空の場合に、理由にかかわらず OracleXMLNoRowsException が発生するかどうかを XSU に通知します。
setRowIdAttrName(ctxType, VARCHAR2)	行の囲みタグの ID 属性名を設定します。
setRowIdAttrValue(ctxType, VARCHAR2)	列値が行の囲みタグの ID 属性に割り当てられるスカラー列を指定します。
setRowsetTag(ctxType, VARCHAR2)	XML データセットを囲むタグを設定します。
setRowTag(ctxType, VARCHAR2)	データベースに対応する XML 要素を囲むタグを設定します。
setSkipRows(ctxType, NUMBER)	スキップする行数を設定します。
setStylesheetHeader(ctxType, VARCHAR2, VARCHAR2 := 'text/xsl')	スタイルシート・ヘッダーを設定します。
setTagCase(ctxType, NUMBER)	生成された XML タグの大文字 / 小文字を設定します。
setXSLT(ctxType, VARCHAR2, VARCHAR2 := null)	生成された XML に適用するスタイルシートを登録します。
setXSLT(ctxType, CLOB, VARCHAR2 := null)	生成された XML に適用するスタイルシートを登録します。
useNullAttributeIndicator(ctxType, BOOLEAN)	NULL であることを示すために、XML 属性を使用するか、XML 文書内の特定のエンティティを省略するかを指定します。

DBMS_XMLSave PL/SQL パッケージ

表 H-7 に、DBMS_XMLSave のプロシージャ、ファンクション、型および定数を示します。

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数

プロシージャ、ファンクション、型および定数	説明
型	
ctxType	問合せのコンテキスト・ハンドルの型です。DBMS_ XMLSave.newContext() の戻り型です。
定数	
DEFAULT_ROWTAG	データベース・レコードに対応する要素のデフォルトのタグ名で、ROW と表されます。
DEFAULT_DATE_FORMAT	デフォルトの日付マスクで、'MM/dd/yyyy HH:mm:ss' と表されます。
MATCH_CASE	XML 要素をデータベースのエンティティへマッピングする場合、XSU が大文字 / 小文字を区別するように指定します。
IGNORE_CASE	XML 要素をデータベースのエンティティへマッピングする場合、XSU が大文字 / 小文字を区別しないように指定します。
プロシージャ	
clearKeyColumnList(ctxType)	キー列リストを消去します。
clearUpdateColumnList(ctxType)	更新列リストを消去します。
closeContext(ctxType)	特定の保存コンテキストのクローズまたは割当て解除を行います。
ファンクション	
deleteXML(ctxType, CLOB)	コンテキストの作成時に指定された表から、XML 文書のデータで指定したレコードを削除します。 戻り値 : NUMBER
deleteXML(ctxType, VARCHAR2)	コンテキストの作成時に指定された表から、XML 文書のデータで指定したレコードを削除します。 戻り値 : NUMBER
プロシージャ	
getExceptionContent(ctxType, NUMBER, VARCHAR2)	発生した例外のエラー・コードおよびエラー・メッセージを引数を介して戻します。

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数（続き）

プロシージャ、ファンクション、型および定数	説明
ファンクション	
insertXML(ctxType, CLOB)	コンテキストの作成時に指定した表に、XML 文書を挿入します。 戻り値 : NUMBER
insertXML(ctxType, VARCHAR2)	コンテキストの作成時に指定した表に、XML 文書を挿入します。 戻り値 : NUMBER
newContext(targetTable IN VARCHAR2)	保存コンテキストを作成し、コンテキスト・ハンドルを戻します。 戻り値 : ctxType
プロシージャ	
propagateOriginalException(ctxType, BOOLEAN)	例外が発生した場合、例外を OracleXMLSQLException でラップするかわりに、呼び出された例外を発生させる必要があることを XSU に通知します。
setBatchSize(ctxType, NUMBER)	DML 操作中に使用されるバッチ・サイズを変更します。
setCommitBatch(ctxType, NUMBER)	コミットのバッチ・サイズを設定します。
setDateFormat(ctxType, VARCHAR2)	XML 文書の日付書式を、XSU に通知します。
setIgnoreCase(ctxType, NUMBER)	XSU は、XML 要素をデータベースにマッピングします。
setKeyColumn(ctxType, VARCHAR2)	キー列リストに列を追加します。
setRowTag(ctxType, VARCHAR2)	データベースに対応する XML 要素を囲むために XML 文書で使用するタグを指定します。
setUpdateColumn(ctxType, VARCHAR2)	更新列リストに列を追加します。
getExceptionContent(ctxType, NUMBER, VARCHAR2)	XML 文書のデータを使用して、コンテキストの作成時に指定した表を更新します。
propagateOriginalException(ctxType, BOOLEAN)	XML 文書のデータを使用して、コンテキストの作成時に指定した表を更新します。

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数（続き）

プロシージャ、ファンクション、型および定数	説明
ファンクション	
newContext(targetTable IN VARCHAR2)	保存コンテキストを作成し、コンテキスト・ハンドルを戻します。 パラメータ : targetTable - XML 文書をロードするターゲット表。 戻り値 : ctxType - コンテキスト・ハンドル
プロシージャ	
closeContext(ctxHdl IN ctxType)	特定の保存コンテキストのクローズまたは割当て解除を行います。 パラメータ : ctxHdl - コンテキスト・ハンドル。
setRowTag(ctxHdl IN ctxType, tag IN VARCHAR2)	データベースのレコードに対応する XML 要素を囲むために XML 文書で使用するタグを指定します。 パラメータ : ctxHdl - コンテキスト・ハンドル。 tag - タグ名。
setIgnoreCase(ctxHdl IN ctxType, flag IN NUMBER)	XSU は、要素名（XML タグ）に基づいて XML 要素をデータベースの列または属性にマップします。このファンクションは、XSU が大文字 / 小文字を区別しないように通知します。 パラメータ : ctxHdl- コンテキスト・ハンドル。 flag - XML 文書内でタグの大文字 / 小文字を無視するかどうか（0 - FALSE、1 - TRUE）。
setDateFormat(ctxHdl IN ctxType, mask IN VARCHAR2)	XML 文書の日付書式を、XSU に通知します。日付書式パターン（日付マスク）の構文は、`java.text.SimpleDateFormat` クラスの要件を満たす必要があります。マスクに NULL または空の文字列を設定すると、デフォルトのマスク `OracleXMLSave.DATE_FORMAT` が使用されます。 パラメータ : ctxHdl - コンテキスト・ハンドル。 mask - 日付マスク。

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数（続き）

プロシージャ、ファンクション、型および定数	説明
setBatchSize(ctxHdl IN ctxType, batchSize IN NUMBER);	DML 操作中に使用されるバッチ・サイズを変更します。挿入、更新および削除を実行する場合、別々の文ではなく、1 回の処理でそれらの操作が実行されるように、バッチ処理を行う方が適しています。欠点は、その操作を実行する前に、すべてのバインド値を保持するためのメモリーがさらに多く必要になることです。バッチ処理を行う場合、バッチ処理の実行後にのみコミットが発生することに注意してください。そのため、バッチ内の 1 文が正常に実行されなかった場合、バッチ処理全体がロールバックされます。バッチ処理によるパフォーマンスの向上とを考えると、大きな問題ではありませんが、これを防止するには、バッチ・サイズを 1 に設定します。 パラメータ : ctxHdl - コンテキスト・ハンドル。 batchSize - バッチ・サイズ。 参照 : DEFAULT_BATCH_SIZE
setCommitBatch(ctxHdl IN ctxType, batchSize IN NUMBER);	コミットのバッチ・サイズを設定します。コミットのバッチ・サイズとは、実行後にコミットする必要がある挿入の回数または記録を意味します。commitBatch が 1 未満またはセッションが自動コミット・モードの場合、XSU は明示的なコミットを行いません。コミットのデフォルトのバッチ・サイズは 0（ゼロ）です。 パラメータ : ctxHdl - コンテキスト・ハンドル。 ParambatchSize - コミットのバッチ・サイズ。
setUpdateColumn(ctxHdl IN ctxType, colName IN VARCHAR2);	更新列リストに列を追加します。挿入の場合、デフォルトでは、表のすべての列に値が挿入されます。更新の場合、デフォルトでは XML 文書の ROW 要素にあるタグに対応する列のみが更新されます。更新列リストを指定すると、このリストを構成する列のみが更新または挿入されます。 パラメータ : ctxHdl - コンテキスト・ハンドル。 colName - 更新列リストに追加される列。
clearUpdateColumnList(ctxHdl IN ctxType)	更新列リストを消去します。 パラメータ : ctxHdl - コンテキスト・ハンドル。 参照 : setUpdateColumn

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数（続き）

プロシージャ、ファンクション、型および定数	説明
setKeyColumn(ctxHdl IN ctxType, colName IN VARCHAR2)	キー列リストに列を追加します。更新または削除の場合、UPDATE 文または DELETE 文の WHERE 句を構成するキー列リストの列です。キー列リストは更新される前に指定する必要がありますが、削除操作ではオプションです。 パラメータ : ctxHdl - コンテキスト・ハンドル。 colName - キー列リストに追加される列。
clearKeyColumnList(ctxHdl IN ctxType)	キー列リストを消去します。 パラメータ : ctxHdl - コンテキスト・ハンドル。 参照 : setKeyColumn
ファンクション	
insertXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	コンテキストの作成時に指定された表に、XML 文書を挿入します。 戻り値 : NUMBER - 挿入された行の数。 パラメータ : ctxHdl- コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。
insertXML(ctxHdl IN ctxType, xDoc IN CLOB)	コンテキストの作成時に指定された表に、XML 文書を挿入します。 戻り値 : NUMBER - 挿入された行の数。 パラメータ : ctxHdl - コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。
updateXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	XML 文書のデータを使用して、コンテキストの作成時に指定された表を更新します。 戻り値 : NUMBER - 更新された行の数。 パラメータ : ctxHdl - コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。
updateXML(ctxHdl IN ctxType, xDoc IN CLOB)	XML 文書のデータを使用して、コンテキストの作成時に指定された表を更新します。 戻り値 : NUMBER - 更新された行の数。 パラメータ : ctxHdl - コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。

表 H-7 DBMS_XMLSave のプロシージャ、ファンクション、型および定数（続き）

プロシージャ、ファンクション、型および定数	説明
deleteXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	コンテキストの作成時に指定した表から、XML 文書のデータで指定したレコードを削除します。 戻り値 : NUMBER - 削除された行の数。 パラメータ : ctxHdl - コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。
deleteXML(ctxHdl IN ctxType, xDoc IN CLOB)	コンテキストの作成時に指定した表から、XML 文書のデータで指定したレコードを削除します。 戻り値 : NUMBER - 削除された行の数。 パラメータ : ctxHdl - コンテキスト・ハンドル。 xDoc - XML 文書を含む文字列。
プロシージャ	
propagateOriginalException(ctxHdl IN ctxType, flag IN BOOLEAN)	例外が発生した場合、例外を OracleXMLSQLException でラップするかわりに、呼び出された例外を発生させる必要があることを XSU に通知します。 パラメータ : ctxHdl - コンテキスト・ハンドル。 flag - 元の例外を継承するかどうか（0 - FALSE、1 - TRUE）。
getExceptionContent(ctxHdl IN ctxType, errNo OUT NUMBER, errMsg OUT VARCHAR2)	発生した例外のエラー・コードおよびエラー・メッセージ（SQL エラー・コード）を引数を介して戻します。これによって、例外が呼び出されるたびに Oracle JVM が例外を発生させることを防止できます。そのため、PL/SQL が元の例外にアクセスできません。 パラメータ : ctxHdl - コンテキスト・ハンドル。 errNo - エラー番号。 errMsg - エラー・メッセージ。

参照： 次のマニュアル、章および Web サイトを参照してください。

- 『Oracle9*i* XML リファレンス』
- 第 7 章「XSU」
- http://otn.oracle.com/tech/xml

用語集

API

「Application Program Interface（API）」を参照。

Application Program Interface（API）

一連のパブリック・プログラム・インタフェース。オペレーティング・システム、またはデータベース、Web サーバー、JVM などの他のプログラム環境と通信するための言語およびメッセージ形式で構成される。通常、これらのメッセージは、アプリケーション開発に使用可能なファンクションおよびメソッドをコールする。

BC4J

Business Components for Java。

BFILE

オペレーティング・システム内に常駐するデータベース表領域の外に存在する外部バイナリ・ファイル。BFILE はデータベース・セマンティクスから参照され、外部 LOB ともいう。

BLOB

「バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）」を参照。

Business-to-Business（B2B）

商品販売およびサービス提供における企業間の相互のコミュニケーションを示す用語。これを実現するソフトウェア・インフラストラクチャが取引である。

Business-to-Consumer（B2C）

商品販売およびサービス提供における企業と顧客間のコミュニケーションを示す用語。

CDATA

「文字データ（character Data: CDATA）」を参照。

CDF

チャネル定義形式。インターネット上のチャネルに関する情報を交換する方法を提供する。

CGI

「Common Gateway Interface（CGI）」を参照。

Class Generator

入力ファイルを受け入れ、対応する機能を持つ一連の出力クラスを作成するユーティリティ。XML Class Generator の場合、入力ファイルは DTD であり、出力は、その DTD に準拠する XML 文書を作成するために使用できる一連のクラスである。

CLASSPATH

JVM がアプリケーションの実行に必要なクラスを検索するために使用する、オペレーティング・システムの環境変数。

CLOB

「キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）」を参照。

Common Gateway Interface（CGI）

Web サーバーが他のプログラムを実行し、ブラウザに送信された HTML ページ、図形、オーディオおよびビデオに出力を渡すことを可能にするプログラム・インタフェース。

Common Object Request Broker API（CORBA）

ネットワーク全体の分散オブジェクト間の通信用の、Object Management Group による標準。これらの自己完結型ソフトウェア・モジュールは、異なるプラットフォームまたはオペレーティング・システム上で実行しているアプリケーションで使用できる。CORBA オブジェクトとそのデータ形式、およびファンクションは、Java、C、C++、Smalltalk、COBOL などの様々な言語にコンパイルできるインタフェース定義言語（IDL）で定義される。

Common Oracle Runtime Environment（CORE）

C で作成されたファンクションのライブラリ。これによって開発者は、事実上すべてのプラットフォームおよびオペレーティング・システムに簡単に移植できるコードを作成できる。

CORBA

「Common Object Request Broker API（CORBA）」を参照。

CSS

カスケーディング・スタイルシート。

DOCTYPE

XML 文書内に DTD またはその参照を指定するタグ名として使用される用語。たとえば、<!DOCTYPE person SYSTEM "person.dtd"> では、ルート要素名が person として、また外部 DTD が person.dtd としてファイル・システム内に宣言される。内部 DTD は、DOCTYPE 宣言内で宣言される。

Document Type Definition（DTD）

XML 文書の使用可能な構造を定義する一連の規則。DTD は、SGML から書式を導出し、DOCTYPE 要素を使用するか、または DOCTYPE 参照を介して外部ファイルを使用して XML 文書内に含めることができるテキスト・ファイルである。

DOM

「ドキュメント・オブジェクト・モデル（Document Object Model: DOM）」を参照。

DTD

「Document Type Definition（DTD）」を参照。

EDI

電子データ交換。

Enterprise JavaBean（EJB）

サーバー上の JVM 内で実行する独立プログラム・モジュール。CORBA によって EJB のインフラストラクチャが提供され、コンテナ・レイヤーによって、サポートされたサーバーにセキュリティ、トランザクション・サポートおよびその他の共通機能が提供される。

eXtensible Markup Language（XML）

データ記述のオープン標準。SGML 構文のサブセットを使用して W3C によって開発され、インターネットでの使用のために設計された。現行の標準はバージョン 1.0 で、1998 年 2 月に W3C 勧告として公開された。

eXtensible Stylesheet Language（XSL）

XML 文書を変換またはレンダリングするために、スタイルシート内で使用される言語。（W3C）の XSL は、XSL Transformations および XSL Formatting Objects という 2 つの W3C 勧告で構成されている。XSL Transformations は 1 つの XML 文書を別の XML 文書に変換し、XSL Formatting Objects は XML 文書の表示を指定する。XSL は、スタイルシートを表す言語である。XSL は、次の 2 つで構成される。

- XML 文書を変換するための言語（XSLT）
- 書式設定セマンティクスを指定するための XML ボキャブラリ（XSLFO）

XSL スタイルシートは、書式設定用ボキャブラリを使用する XML 文書へのクラスのインスタンスの変換方法を記述して、XML 文書のクラス表示を指定する。

eXtensible Stylesheet Language Formatting Object（XSLFO）

書式設定セマンティクスを指定するための XML 用語を定義する、W3C の標準仕様。

eXtensible Stylesheet Language Transformation（XSLT）

XSLT とも書く。XML 文書を別のドキュメントに変換する変換言語を定義する、W3C の XSL 標準仕様。

HTML

「Hypertext Markup Language（HTML）」を参照。

HTTP

「Hypertext Transfer Protocol（HTTP）」を参照。

Hypertext Markup Language（HTML）

Web ブラウザに送信するファイルを作成するために使用し、Web の基礎として機能するマークアップ言語。HTML の次のバージョンは xHTML と呼ばれ、XML アプリケーションになる予定である。

Hypertext Transfer Protocol（HTTP）

インターネットを介して、Web サーバーとブラウザ間で HTML ファイルを転送するために使用するプロトコル。

IDE

「統合開発環境（Integrated Development Environment: IDE）」を参照。

*inter*Media

複合データ型のコレクションおよびコレクションの Oracle9*i* 内でのアクセスを示す用語。これには、テキスト、ビデオ、時系列および空間データ型が含まれる。

Oracle Internet File System

Oracle9*i* データベース内または中間層上で実行する、Oracle のファイル・システムおよび Java ベースの開発環境。単一のデータベース・リポジトリに複数の型のドキュメントを作成、格納および管理する方法を提供する。

Internet Inter-ORB Protocol（IIOP）

インターネットなどの TCP/IP ネットワーク上で、メッセージを交換するために CORBA が使用するプロトコル。

Java

Sun 社によって開発およびメンテナンスされた高水準のプログラミング言語。Java では、アプリケーションが JVM という仮想マシン内で実行する。JVM は、オペレーティング・システムに対するすべてのインタフェースの役割を担う。開発者は、このアーキテクチャによって、JVM を搭載するすべてのオペレーティング・システムまたはプラットフォームで実行する Java アプリケーションおよびアプレットを開発できる。

Java Bean

JVM 内で実行する独立プログラム・モジュール。通常、クライアント側のユーザー・インタフェースを作成するために使用される。サーバー側の同等のモジュールは、Enterprise JavaBean（EJB）という。「Enterprise JavaBean（EJB）」を参照。

Java Database Connectivity（JDBC）

Java アプリケーションが、SQL 言語を介してデータベースにアクセスできるようにするプログラム API。JDBC ドライバは、プラットフォームに依存しないように Java で作成されるが、各データベースに固有である。

Java Development Kit（JDK）

Java 開発環境を確立する Java バージョン用の、Java クラス、ランタイム、コンパイラ、デバッガおよびソース・コードのコレクション。JDK はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

Java Runtime Environment（JRE）

プラットフォーム上で Java 仮想マシンを構成する、コンパイル済クラスのコレクション。JRE はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

Java Virtual Machine（Java VM）

コンパイル済 Java バイトコードをプラットフォームのマシン言語に変換し、それを実行する Java インタプリタ。JVM は、クライアント側、ブラウザ内、中間層内、Web 上、OAS などのアプリケーション・サーバー上、または Oracle9*i* などのデータベース・サーバー内で実行できる。

Java VM

「Java Virtual Machine（Java VM）」を参照。

JavaServer Page（JSP）

Web ページへの単純なプログラム・インタフェースを可能にする、サーブレットの拡張機能。JSP は、特殊タグ、および Web またはアプリケーション・サーバーで実行される埋込み Java コードを含む HTML ページであり、HTML ページに動的機能を提供する。JSP は、サーバーの JVM で最初に要求および実行されるときに、サーブレットにコンパイルされる。

JDBC

「Java Database Connectivity（JDBC）」を参照。

JDeveloper

アプリケーション、アプレットおよびサーブレットの開発を可能にする Oracle の Java IDE。エディタ、コンパイラ、デバッガ、構文チェッカ、ヘルプ・システムなどを含む。バージョン 3.1 の JDeveloper は、エディタで XML がサポートされるとともに、簡単に使用できるように統合された Oracle XDK for Java を搭載して、XML ベースの開発をサポートするように拡張されている。

JDK

「Java Development Kit（JDK）」を参照。

LAN

「Local Area Network（LAN）」を参照。

LOB

「ラージ・オブジェクト（Large Object: LOB）」を参照。

Local Area Network（LAN）

限定された地域内のユーザー用のコンピュータ通信ネットワーク。LAN は、サーバー、ワークステーション、通信ハードウェア（ルーター、ブリッジ、ネットワーク・カードなど）およびネットワーク・オペレーティング・システムで構成される。

NCLOB

「各国語キャラクタ・ラージ・オブジェクト（National Character Large Object: NCLOB）」を参照。

N 層（N-tier）

クライアントおよびサーバーで構成される 1 つ以上の層で構成される、コンピュータ通信ネットワーク・アーキテクチャの指定。通常、2 層システムは 1 つのクライアント・レベルおよび 1 つのサーバー・レベルで構成される。3 層システムは、通常、1 つのクライアン層とともに、データベース・サーバーと Web またはアプリケーション・サーバーの 2 つのサーバー層を使用する。

OAG

Open Applications Group。

OAI

Oracle Applications Integrator。CRM アプリケーションを Oracle ERP に加えて他の ERP システムと統合するための、Oracle *i*Studio 開発ツールを含むランタイム。固有の API は、「メッセージ対応」である必要がある。標準の拡張フックを使用して、他のアプリケーション・システムと交換された XML ストリームを生成または解析する。開発中。

OAS

「Oracle Application Server（OAS）」を参照。

OASIS

「Organization for the Advancement of Structured Information（OASIS）」を参照。

Object Request Broker（ORB）

クライアント側の要求元プログラムとサーバー側のオブジェクト間のメッセージ通信を管理するソフトウェア。ORB は、アクション要求およびそのパラメータをオブジェクトに渡し、結果を戻す。共通の実装は、CORBA および EJB である。「Common Object Request Broker API（CORBA）」を参照。

OE

Oracle Exchange。

OIS

「Oracle Integration Server（OIS）」を参照。

Oracle Application Server（OAS）

Oracle アプリケーション・サーバー。オープン標準フレームワーク内で高パフォーマンスの N 層トランザクション指向 Web アプリケーションを構築、配置および管理するために必要な、すべての主要なサービスおよび機能を統合する。

Oracle CM SDK

「Oracle Internet File System」を参照。

Oracle Integration Server（OIS）

アプリケーション統合のためのメッセージング・ハブとして機能する Oracle のサーバー製品。OIS には、AQ および Oracle Workflow を搭載した Oracle8*i* データベース、および Oracle Message Broker を使用して、アプリケーション間で XML 形式のメッセージを転送するアプリケーションへのインタフェースが含まれる。

Oracle JVM

Oracle データベースのメモリー領域内で実行する Java VM。JVM は、Oracle 8*i* リリース 8.1.5 では Java1.1 準拠であり、リリース 8.1.6 以降では Java1.2 準拠である。

ORACLE_HOME

アプリケーションで使用するために、Oracle データベースのインストール場所を識別するオペレーティング・システムの環境変数。

ORB

「Object Request Broker（ORB）」を参照。

Organization for the Advancement of Structured Information（OASIS）

会議、セミナー、展示会およびその他の教育イベントを通じて、パブリック情報標準の普及促進を目的として設立されたメンバーの組織。XML は、SGML と同様に、OASIS が活発に普及を促進している標準である。

PCDATA

「解析対象文字データ（Parsed Character Data: PCDATA）」を参照。

PDA

Palm Pilot などのパーソナル・デジタル・アシスタント。

PL/SQL

データベース内で実行できるプログラムを作成するために SQL を拡張した、Oracle 手続き型言語。

PUBLIC

後に続く参照の、インターネット上の場所を指定する用語。

RDF

Resource Definition Framework。

SAX

「Simple API for XML（SAX）」を参照。

Secure Sockets Layer（SSL）

インターネット上のプライマリ・セキュリティ・プロトコル。ブラウザとサーバー間の暗号化形式に、公開鍵 / 秘密鍵を使用する。

SGML

「Standard Generalized Markup Language（SGML）」を参照。

Simple API for XML（SAX）

XML パーサーによって提供され、イベント駆動型のアプリケーションによって使用される XML 標準インタフェース。

SQL

「Structured Query Language（SQL）」を参照。

SSI

「サーバー側インクルード（Server-side Include: SSI）」を参照。

SSL

「Secure Sockets Layer（SSL）」を参照。

Standard Generalized Markup Language（SGML）

マークアップおよびDTDを使用して実装された、テキスト・ドキュメントの書式を定義するためのISO標準。

Structured Query Language（SQL）

リレーショナル・データベース内のデータをアクセスおよび処理するために使用する標準言語。

SYSTEM

後に続く参照の、ホスト・オペレーティング・システム上の場所を指定する用語。

TCP/IP

「Transmission Control Protocol/Internet Protocol（TCP/IP）」を参照。

Transmission Control Protocol/Internet Protocol（TCP/IP）

TCPで構成される通信ネットワーク・プロトコル。TCPは、トランスポート機能、およびルーティング・メカニズムを提供するIPを制御する。TCP/IPは、インターネット通信の標準である。

Uniform Resource Identifier（URI）

URLおよびXPathを作成するために使用するアドレス構文。

Uniform Resource Locator（URL）

インターネット上のファイルの場所およびルートを定義するアドレス。URLは、Webをナビゲートするためにブラウザによって使用され、プロトコル接頭辞、ポート番号、ドメイン名、ディレクトリ名とサブディレクトリ名、およびファイル名で構成される。たとえば、http://technet.oracle.com:80/tech/xml/index.htmでは、Web上のOTNのXMLサイトを検索するためにブラウザが移動する、場所およびパスが指定されている。

URI

「Uniform Resource Identifier（URI）」を参照。

URL

「Uniform Resource Locator（URL）」を参照。

W3C

「World Wide Web Consortium（W3C）」を参照。

WAN

「Wide Area Network（WAN）」を参照。

Web Request Broker（WRB）

URL を処理し、適切なカートリッジに送信する OAS 内のカートリッジ。

Wide Area Network（WAN）

州や国などの広域内のユーザー用のコンピュータ通信ネットワーク。WAN は、サーバー、ワークステーション、通信ハードウェア（ルーター、ブリッジ、ネットワーク・カードなど）およびネットワーク・オペレーティング・システムで構成される。

World Wide Web Consortium（W3C）

1994 年に設立された、Web の標準を確立するための国際的な産業組合。W3C のサイトは、www.w3c.org である。

XLink

XML 文書内でのハイパーリンクの使用を制御する規則で構成された XML Linking 言語。これらの規則は、W3C の勧告プロセス下の XML Linking Group によって開発されている。XLink は、XML がドキュメントの表示およびハイパーリンクの管理にサポートする 3 つの言語（Xlink、Xpointer および XPath）の 1 つである。

XML

「eXtensible Markup Language（XML）」を参照。

XML Developer's Kit（XDK）

ソフトウェア開発者に、アプリケーションを XML 対応にするための標準ベースの機能を提供する、一連のライブラリ、コンポーネントおよびユーティリティ。Oracle XDK for Java には、XML パーサー、XSL プロセッサ、XML Class Generator、Transviewer Beans および XSQL Servlet が含まれる。

XML Query

W3C が取り組む、XML 文書を問い合せるための言語および構文の標準。

XML Schema

W3C が取り組む、XML 文書内で単純なデータ型および複合構造を表すための標準。データ型の定義や妥当性など、現在 DTD で不足している領域に取り組んでいる。Oracle XML Schema Processor は、オンライン取引などの E-Business アプリケーションで使用される、XML 文書およびデータの妥当性を自動的に検証する。XML Schema は、XML 文書に単純型および複合型を追加し、DTD の機能を XML Schema 定義の XML 文書に置き換える。

XML Transviewer Beans

XDK for Java に含まれる Oracle XML JavaBeans を示す Oracle 用語。これらの Bean には、XML Source View Bean、Tree View Bean、DOMParser Bean、Transformer Bean および TransViewer Bean が含まれる。

XPath

XSL および XPointer で使用されるドキュメント内で要素を指定するための、オープン標準の構文。XPath は、現在 W3C 勧告である。XSLT、XLink および XML Query に使用される XML 文書を操作するためのデータ・モデルおよび文法を指定する。

XPointer

XML 文書フラグメントへの参照を記述するための W3C 勧告。XPointer は、XPath 形式の URI の終わりに使用できる。XPath ナビゲーションを使用して、XML 文書内の個別のエンティティまたはフラグメントの識別を指定する。

XSL

「eXtensible Stylesheet Language（XSL）」を参照。

XSLFO

「eXtensible Stylesheet Language Formatting Object（XSLFO）」を参照。

XSLT

「eXtensible Stylesheet Language Transformation（XSLT）」を参照。

XSQL

Oracle XSQL Servlet によって使用される指定。1 つ以上の SQL 問合せから動的 XML 文書を生成し、XML スタイルシートを使用して、サーバー内のドキュメントを変換する（オプション）機能を提供する。

アプリケーション・サーバー（Application Server）

アプリケーションおよびその環境をホストするために設計されたサーバーであり、サーバー・アプリケーションの実行を許可する。代表的な例は OAS で、リモート・クライアントがインタフェースを制御する場合に、Java、C、C++ および PL/SQL アプリケーションをホストできる。「Oracle Application Server（OAS）」を参照。

インスタンス化（instantiate）

Java や C++ などのオブジェクト・ベース言語で使用される用語で、特定のクラスのオブジェクト作成を示す。

エンティティ（entity）

別の文字列、またはドキュメントのキャラクタ・セットに属さない特殊文字を表すことができる文字列。エンティティ、およびパーサーによってエンティティの代替となるテキストは、DTD に宣言される。

オブジェクト・ビュー（Object View）

1 つ以上のオブジェクト表または他のビューに含まれるデータの、調整された外観。オブジェクト・ビュー問合せの出力は、表として扱われる。オブジェクト・ビューは、表が使用されているほとんどの場所で使用できる。

オブジェクト・リレーショナル（object-relational）

テキスト・ドキュメント、オーディオ・ファイル、ビデオ・ファイル、ユーザー定義オブジェクトなどの高順序のデータ型を格納および操作できるリレーショナル・データベース・システムを示す用語。

親要素（parent element）

子要素という別の要素を囲む要素。たとえば、<Parent><Child></Child></Parent> は、Parent 要素がその Child 要素をラップしていることを示す。

カートリッジ（cartridge）

Java または PL/SQL のストアド・プログラム。データベースが新しいデータ型を理解および処理するために必要な機能を追加する。カートリッジは、Oracle 内の拡張フレームワークでインタフェースの役割を担う。Oracle Text はこの種類のカートリッジであり、データベース内に格納されたテキスト・ドキュメントの読込み、書込みおよび検索のサポートを追加する。

解析対象文字データ（Parsed Character Data: PCDATA）

解析する必要があるが、タグまたは解析対象外データの一部ではないテキストで構成される要素内容。

各国語キャラクタ・ラージ・オブジェクト（National Character Large Object: NCLOB）

データベースの各国語キャラクタ・セットに対応する文字データで構成された値を持つ LOB データ型。

空要素（empty element）

テキスト内容または子要素のない要素。属性およびその値のみを含む場合がある。空要素の書式は、<name/>、または <name></name>（タグの間には空白なし）である。

キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）

データベース・キャラクタ・セットに対応する文字データで構成された値を持つ LOB データ型。CLOB は、Oracle Text の検索エンジンを使用して索引付けおよび検索できる。

クライアント / サーバー（client-server）

実際のアプリケーションはクライアント側で実行するが、ネットワークを介して、サーバー側のデータまたは他の外部プロセスにアクセスするアプリケーション・アーキテクチャを表す用語。

結果セット（result set）

1 行以上のデータで構成される SQL 問合せの出力。

コールバック（callback）

1 つのプロセスに他のプロセスを開始させ、それを継続させるプログラム方法。2 番目のプロセスは、アクションの結果、値または他のイベントとして 1 番目のプロセスをコールする。この方法は、継続的な対話を許可するユーザー・インタフェースを持つほとんどのプログラムに使用されている。

コマンドライン（command line）

ユーザーがコマンド・インタプリタのプロンプトにコマンドを入力するインタフェース・メソッド。

子要素（child element）

親要素という別の要素内に完全に含まれた要素。たとえば、
<Parent><Child></Child></Parent> は、Child 要素がその Parent 要素内にネストされていることを示す。

サーバー側インクルード（Server-side Include: SSI）

データまたは他の内容を、要求元ブラウザに送信する前に Web ページに置く HTML コマンド。

サーブレット（servlet）

サーバー（通常は Web またはアプリケーション・サーバー）内で実行する Java アプリケーション。サーブレットは、CGI スクリプトに相当する Java である。

スキーマ（schema）

データベース内の構造およびデータ型の定義。スキーマは、XML Schema の W3C 勧告をサポートする XML 文書も示す。

スタイルシート（Stylesheet）

XML では、XML 処理命令で構成される XML 文書を示す用語。XML 処理命令は、入力 XML 文書を出力 XML 文書に変換またはフォーマットするために、XML プロセッサによって使用される。

スレッド（thread）

プログラミングにおける、Windows、UNIX、Java などの複数のオペレーティング・システムをサポートするオペレーティング・システム内の、単一のメッセージまたはプロセス実行パス。

整形式（well-formed）

XML 文書が、XML 宣言で宣言された XML バージョンの構文に準拠している状態を示す用語。これには、ルート要素が単一か、タグが適切にネストされているかなどが含まれる。

セッション（session）

2 つの層の間のアクティブ接続。

属性（attribute）

要素のプロパティ。等号で区切られた名前および値で構成され、開始タグ内の要素名の後に含まれる。たとえば、<Price units='USD'>5</Price> では、units（単位）が属性で USD がその値である。値は、一重または二重引用符で囲む必要がある。属性は、ドキュメントまたは DTD 内に格納できる。属性は、ドキュメントまたは DTD 内に格納できる。要素には多くの属性を指定できるが、その取得順序は定義されない。

タグ（tag）

XML マークアップの単一のピース。要素の開始または終了を指定する。タグは、< で始まり > で終わる。XML には、開始タグ（<name>）、終了タグ（</name>）および空タグ（<name/>）がある。

妥当（valid）

XML 文書の構造および要素内容が、参照 DTD または内部 DTD で宣言されたものと一貫している状態を示す用語。

データグラム（datagram）

XSQL Servlet が処理した SQL 問合せから、HTML ページに埋め込まれたリクエスタに戻されるテキストのフラグメント。XML 形式の場合もある。

データベース・アクセス記述子（Database Access Descriptor: DAD）

データベース・アクセスに使用される、名前付きの一連の構成値。DAD は、データベース名や SQL*Net V2 サービス名などの情報、ORACLE_HOME ディレクトリ、および言語、ソート型、日付言語などの National Language Support（NLS）構成情報を指定する。

統合開発環境（Integrated Development Environment: IDE）

ソフトウェアの開発を支援するために設計された、単一のユーザー・インタフェースから実行されるプログラム・セット。JDeveloper は、Java 開発用の IDE であり、エディタ、コンパイラ、デバッガ、構文チェッカ、ヘルプ・システムなどを含む。JDeveloper を使用すると、単一のユーザー・インタフェースを介して Java ソフトウェアを開発できる。

ドキュメント・オブジェクト・モデル（Document Object Model: DOM）

XML 文書のメモリー内ツリーベースのオブジェクト表現。要素および属性へのプログラム・アクセスを可能にする。DOM オブジェクトおよびそのインタフェースは、W3C 勧告である。プログラム・アクセス用の API など、XML 文書の DOM を指定する。DOM は、解析対象ドキュメントをオブジェクトのツリーとして表示する。

名前空間（namespace）

XML 文書内にある、関連する一連の要素名または属性を示す用語。名前空間の構文およびその使用方法は、W3C 勧告によって定義されている。たとえば <xsl:apply-templates/ > 要素は、XSL 名前空間の一部として識別される。名前空間は、XML 文書または DTD 内で、属性の構文 xmlns:xsl="http://www.w3.org/TR/WD-xsl" を宣言してから宣言される。

パーサー

XML で、XML 文書を入力として受け入れ、ドキュメントが整形式であり、また妥当（オプション）であるかどうかを判断するソフトウェア・プログラム。Oracle XML Parser は、SAX および DOM インタフェースの両方をサポートする。

バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）

内容がバイナリ・データで構成されたラージ・オブジェクト・データ型。このデータは、データ構造がデータベースに認識されないため、RAW 型とみなされる。

ハイパー・テキスト（hypertext）

ユーザーがハイパーリンクとして指定されたワードまたは句を選択して、他のドキュメントまたは図形間を操作できるテキスト・ドキュメントを作成および公開する方法。

表記法（NOTATION）

XML では、パーサーが理解できない内容の型の定義。これらの型には、オーディオ、ビデオおよび他のマルチメディアが含まれる。

プロローグ（prolog）

XML 宣言および DTD、またはドキュメントを処理するために必要な他の宣言を含む、XML 文書の最初の部分。

モード（mode）

XML では、DOM ツリー内のアドレス指定可能な各エンティティを示す用語。

文字データ（character Data: CDATA）

ドキュメント内の解析対象外のテキストは、CDATA セクションに格納される。これによって、&、<、> などの、他に特別な機能を持つ文字を含めることができる。CDATA セクションは、要素の内容または属性内で使用できる。

ユーザー・インタフェース（user interface: UI）

メニュー、スクリーン、キーボード・コマンド、マウス・クリック、およびユーザーによるソフトウェア・アプリケーションとの対話方法を定義するコマンド言語の組合せ。

要素（element）

XML 文書の基本論理単位。子、データ、属性とその値などの他の要素に対するコンテナとして機能する。要素は、開始タグ <name> および終了タグ </name>、または空要素の場合、<name/> によって識別される。

ラージ・オブジェクト（Large Object: LOB）

内部 LOB および外部 LOB に分割された SQL データ型のクラス。内部 LOB には BLOB、CLOB および NCLOB が含まれ、外部 LOB には BFILE が含まれる。「BFILE」、「バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）」および「キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）」を参照。

ラッパー（Wrapper）

通常、汎用またはオブジェクト・インタフェースを提供するために、他のデータまたはソフトウェアをラップするデータ構造またはソフトウェアを示す用語。

リスナー（listener）

入力プロセスを監視する個別のアプリケーション・プロセス。

ルート要素（root element）

XML 文書内にある他のすべての要素を囲む要素。オプションのプロローグとエピローグの間に存在する。XML 文書には、1 つのルート要素のみ置くことができる。

レンダラ（renderer）

ドキュメントを指定された形式に出力するソフトウェア・プロセッサ。

ワーキング・グループ（Working Group: WG）

特定のインターネット・テクノロジ分野における勧告を実行する業界のメンバーで構成された W3C の委員会。

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

あ

い

え

お

か

き

く

け

こ

さ

も

よ

ら

り

る

れ

ろ