Oracle9*i*

XML Developer's Kit ガイド - XDK

リリース 2（9.2）

2002 年 7 月

部品番号 : J06309-01

ORACLE®

Oracle9*i* XML Developer's Kit ガイド -XDK, リリース 2（9.2）

部品番号 : J06309-01

原本名：Oracle9*i* XML Developer's Kits Guide - XDK, Release 2 (9.2)

原本部品番号：A96621-01

原本著者：Jack Melnick

グラフィック・デザイナ：Valarie Moore

原本協力者：Mark Bauer、Shelley Higgins、Steve Muench、Mark Scardina、Jinyu Wang、Sandeepan Banerjee、Kishore Bhamidipati、Bill Han、K. Karun、Murali Krishnaprasad、Bruce Lowenthal、Anjana Manian、Meghna Mehta、Nick Montoya、Ravi Murthy、Den Raphaely、Blaise Ribet、Tarvinder Singh、Tomas Saulys、Tim Yu、Jim Warner、Simon Wong、Kongyi Zhou

目次

第 II 部　XDK for Java

4　XML Parser for Java

5 XSLT Processor for Java

6 XML Schema Processor for Java

7 XML Class Generator for Java

8 XML SQL Utility（XSU）

9 XSQL Pages パブリッシング・フレームワーク

10 XDK JavaBeans

11 XDK および SOAP の使用

12 Oracle TransX Utility

第 III 部 XDK for C/C++

13 XML Parser for C

14 XSLT Processor for C

15 XML Schema Processor for C

16 XML Parser for C++

17 XSLT Processor for C++

18 XML Schema Processor for C++

19 XML Class Generator for C++

第 IV 部 XDK for PL/SQL

20 XML Parser for PL/SQL

21 XSLT Processor for PL/SQL

22 XSU for PL/SQL

第 V 部　XDK をサポートするツール製品およびフレームワーク

23　JDeveloper を使用した XML アプリケーションの開発

24 BC4J の概要

25 User Interface XML（UIX）の概要

A XDK for Java: 仕様およびクイック・リファレンス

B XDK for PL/SQL: 仕様

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
- 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
- 表記規則

このマニュアルの内容

このマニュアルでは、Oracle9*i* 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。

XML の関連文書

XML アプリケーションの構築の詳細は、次のマニュアルを参照してください。

参照：

- 『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- 『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』

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/`

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文書は、OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

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

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

```
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
- コード例中の構文および表記規則

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。
固定幅フォントの大文字	固定幅フォントの大文字は、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、Recovery Manager 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
固定幅フォントの小文字	固定幅フォントの小文字は、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。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、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。

コード例中の構文および表記規則

構文例は、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
[ ]	構文例では、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`* `])`
{ }	構文例では、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}`
|	構文例では、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。 ■ 構文例で、引数を追加できる。	CREATE TABLE ... AS *subquery*; 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。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
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 / 小文字が区別されないため、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;

XDK 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。

- Oracle9i リリース 2（9.2）で導入された XDK の新機能
- Oracle9i リリース 1（9.0.1）で導入された XDK の新機能
- Oracle8i リリース 8.1.7 で導入された XDK の機能

Oracle9*i* リリース 2（9.2）で導入された XDK の新機能

XML Schema Processor for Java

W3C の XML Schema 勧告をサポートします。

スキーマ識別制約の検証に、外部 DocumentBuilder が不要になります。

XSLT スタイルシート

スレッド・セーフな XSLStylesheet オブジェクトをサポートします。

XSQL Servlet

<xsql:include-owa> のパフォーマンスを向上させる新しいオプションです。

<xsql:set-page-param> は、xpath="Expr" 属性をサポートします。

CLOB 列および VARCHAR2 列からの XML の挿入が簡単になります。

転送された XML を挿入するための新しい <xsql:include-posted-xml> アクション・ハンドラが追加されています。

Apache FOP リリース 0.19 をサポートします。

Cookie として設定された値の即時読込みをサポートします。

単一の SQL 文による複数のパラメータ値の設定をサポートします。

Class Generator for Java

DTD Class Generator にデータのバインド機能が追加されています。

XML インスタンス・ドキュメントを入力値として指定し、生成されたクラスにインスタンス・データをロードできます。

XDK for Java

XML SQL Utility（XSU）は、Simple API for XML（SAX）2.0、および SQL 問合せの XML Schema の生成をサポートします。

DOM レベルの圧縮をサポートします。

Oracle SOAP API が追加されています。

XML Parser for Java での SAX2 のサポートが拡張されています。

XDK for Java によって JAXP 1.1 がサポートされています。

Oracle TransX Utility は、データおよびテキストのロードに有効です。

XML Schema Processor for Java は、LAX モードと STRICT モードの両方をサポートします。

XML Parser for Java での XML の圧縮がサポートされています。

XDK for C

Linux 版でリリースされています。

XDK for C++

Linux 版でリリースされています。

XDK for JavaBeans

新しい XMLDiff Bean が追加されています。

SourceViewer Bean に内部 DTD サポートが追加されています。

OTN

新しい XDK のライブ・デモを次の URL から実行できます。

`http://otn.oracle.com/tech/xml/xdk_sample/xdkdemo_faq.html`

`http://otn.oracle.com/tech/xml/xdk_sample/xdkdemo_xsql.html`

最新の XDK のテクニカル・ペーパー『Building Server-Side XML Schema Validation』を次の URL から参照できます。

`http://otn.oracle.com/tech/xml/xdk_sample/xdksample_093001i.html`

Oracle9*i* リリース 1（9.0.1）で導入された XDK の新機能

Oracle9*i* リリース 1（9.0.1）で導入された XDK 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 4 章「XML Parser for Java」
  - 第 6 章「XML Schema Processor for Java」

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

  参照： 第 7 章「XML Class Generator for Java」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 Drivers 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。
  - 転送された 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 : すべての挿入リクエストまたは更新リクエストに含める列を、明示的にリストできます。
  - ページ・リクエスト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・リクエスト・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 9 章「XSQL Pages 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 XSLT 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 10 章「XDK JavaBeans」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 15 章「XML Schema Processor for C」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 18 章「XML Schema Processor for C++」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 20 章「XML Parser for PL/SQL」を参照してください。

XML SQL Utility（XSU）の機能

- 任意の SQL 問合せによる XML Schema の生成
- XMLType および URIRef 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 8 章「XML SQL Utility（XSU）」を参照してください。

Oracle8*i* リリース 8.1.7 で導入された XDK の機能

Oracle8*i* リリース 8.1.7 で導入された XDK の新機能では、次のコンポーネントが拡張および改善されています。

- XDK for Java
- XDK for C
- XDK for C++
- XDK for PL/SQL
- XML SQL Utility

第 I 部

XML Developer's Kit（XDK）

第 I 部では、Oracle の XML 対応テクノロジとその機能、Oracle XML Developer's Kit（XDK）と XML コンポーネント、および XDK のインストール方法について説明します。第 I 部に含まれる章は、次のとおりです。

1

XML Developer's Kit およびコンポーネントの概要

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

Oracle9*i* 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。

XDK 9.0.2（Oracle9*i*AS リリース 2（9.0.2）に付属）および XDK 9.2（Oracle9*i* リリース 2（9.2）に付属）では、XSLStylesheet はスレッド・セーフであり、複数の `XSLProcessor.processXSL` コールのスレッド間で使用できます。ただし、軽量なオブジェクトである XSLProcessor は、スレッド・セーフになりません。

次の XML コンポーネントは、Oracle9*i* および Oracle9*i* Application Server に付属しています。

- **XML Developer's Kit（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 文書の読込み、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 1-1 に、XDK コンポーネントを示します。
- **XML SQL Utility（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（またはその逆に）正規にマップすることによって、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 1-8「XDK for Java を使用した XML 文書の生成」
- 図 1-9「XDK for C を使用した XML 文書の生成」
- 図 1-10「XDK for C++ を使用した XML 文書の生成」
- 図 1-11「XDK for PL/SQL を使用した XML 文書の生成」

表 1-1 XDK コンポーネントの説明

XDK コンポーネント	言語	説明
XML Parser	Java、C、C++、PL/SQL	インターネット標準の 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形式に変換またはレンダリングします。
XML Schema プロセッサ	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。

表 1-1 XDK コンポーネントの説明（続き）

XDK コンポーネント	言語	説明
XML Transviewer Beans	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。
XML SQL Utility（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。
TransX Utility	Java	インストールに有効な追加の SQL 機能を使用して、XML でカプセル化されたデータをデータベースにロードします。
Oracle SOAP Server	Java	第 11 章「XDK および SOAP の使用」を参照してください。
XML Compressor	Java	4-9 ページの「XML Compressor」を参照してください。

開発ツールおよび XML 対応のその他の Oracle9*i* 機能

XML 対応の開発ツールは次のとおりです。

Oracle Text: 問合せ、検索および取得のためのツールです。

Oracle9*i* JDeveloper および Business Components for Java（BC4J）: Oracle9*i* JDeveloper 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は、Common Object Request Broker Architecture（CORBA）サーバー・オブジェクトまたは Enterprise JavaBeans（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。

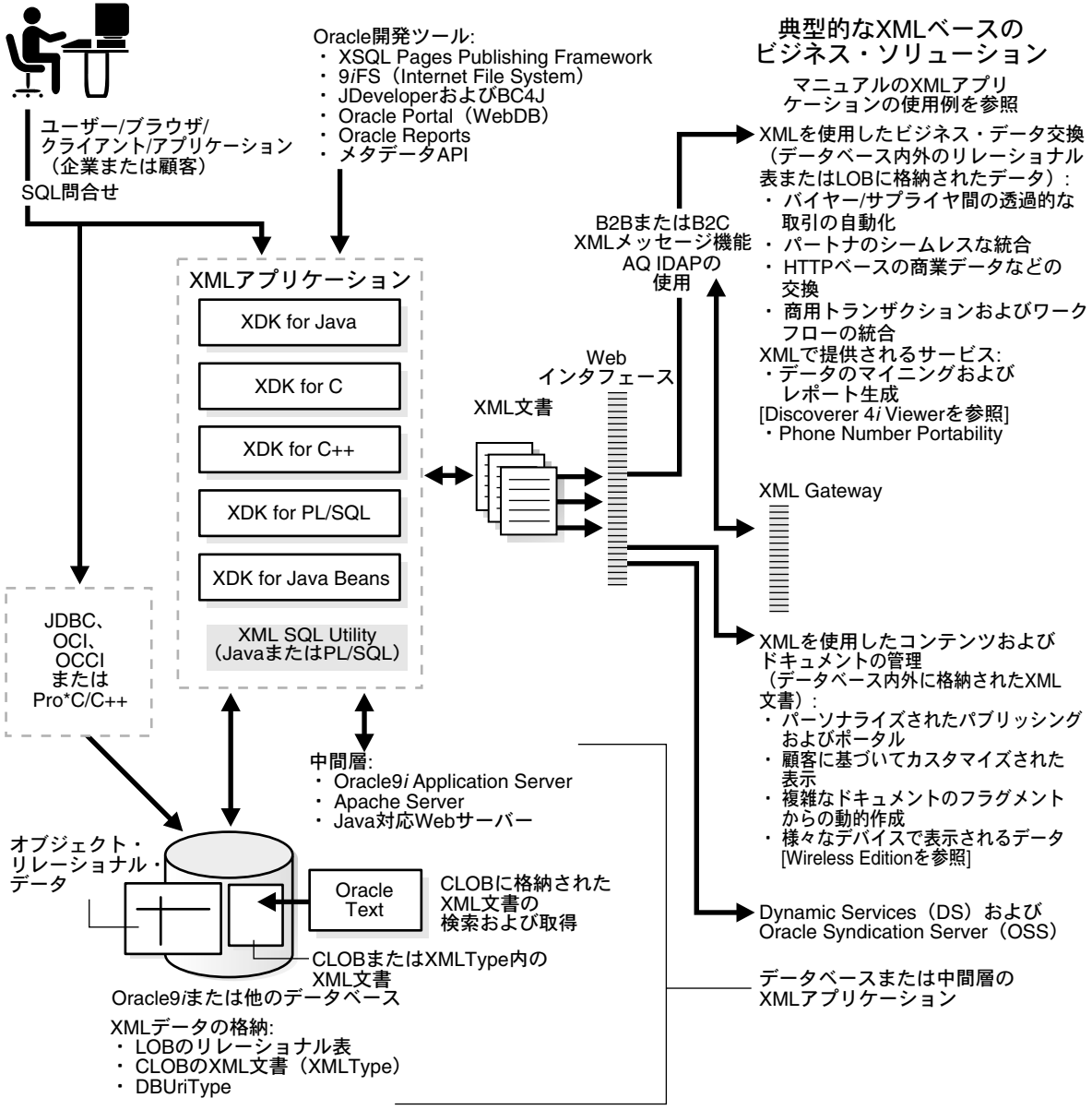
> **参照：** 第 21 章「XSLT Processor for PL/SQL」を参照してください。

- **Oracle9*i* Internet File System（9*i*FS）:** 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。9*i*FS 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、9*i*FS は XML のリポジトリとして機能します。9*i*FS 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 Portable Document Format（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形式で、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。

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



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:** インターネット標準の 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。XML を XML または他のテキストベース形式（HTML など）に変換する **XSL Transformation（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 バージョン 2 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。XML SQL Utility および XML Parser for Java を使用します。
- **XML SQL Utility（XSU）for Java:** 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得したデータを 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。
- **SOAP Server:** インターネットを介してレスポンスを送受信するためのプロトコルです。
- **TransX Utility:** 変換されたシード・データおよびメッセージをデータベースにロードする操作を簡単にします。
- **XML Compressor:** XML 文書が、XML パーサーによってバイナリ・ストリームに圧縮されます。

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。
- **XMLDiff Bean:** 2 つの XML DOM ツリーを比較します。2 つの XML ツリーを表示し、それらの XML ツリーの違いを示します。

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:** インターネット標準の 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。XML を XML または他のテキストベース形式（HTML など）に変換する **XSL Transformation（XSLT）プロセッサ**を搭載しています。
- **XSLT プロセッサ :** XML を、HTML や WML など他のテキストベースの形式に変換またはレンダリングします。

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++:** インターネット標準の 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。XML を XML または他のテキストベース形式（HTML など）に変換する **XSL Transformation（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。Oracle9*i* 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。XML Schema プロセッサは、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。
- **XSLT プロセッサ :** XML を、HTML や WML など他のテキストベースの形式に変換またはレンダリングします。

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:** インターネット標準の 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。XML を XML または他のテキストベース形式（HTML など）に変換する **XSL Transformation（XSLT）プロセッサ**を搭載しています。
- **XML SQL Utility（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得したデータを 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。
- **XSLT プロセッサ :** XML を、HTML や WML など他のテキストベースの形式に変換またはレンダリングします。

Oracle XML Parser

Oracle XML Parser には、Oracle9*i* 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 XSL Transformation: 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形式への 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 1-2 に、Oracle XML Parser for Java を示します。また、図 1-3 に、Oracle XML Parser の機能の概要を示します。

> **参照：** 第 4 章「XML Parser for Java」および付録 A「XDK for Java: 仕様およびクイック・リファレンス」を参照してください。

図 1-2 Oracle XML Parser for Java

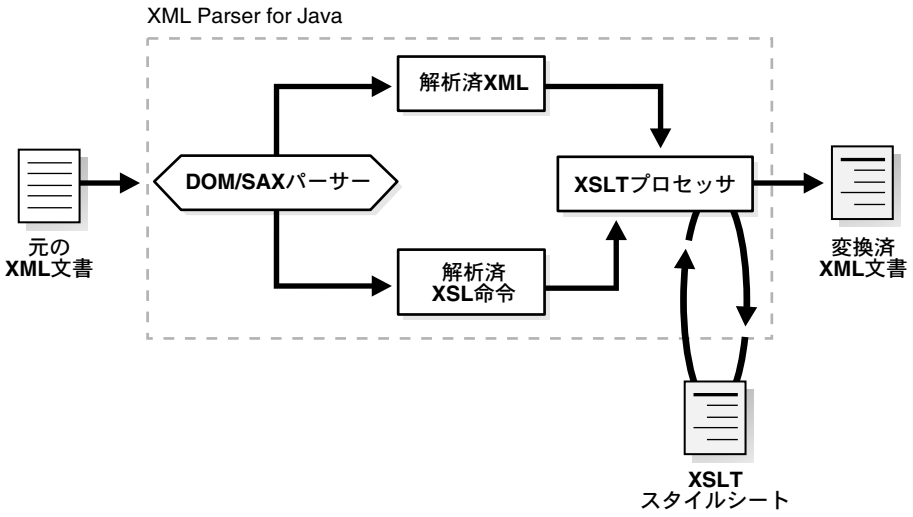
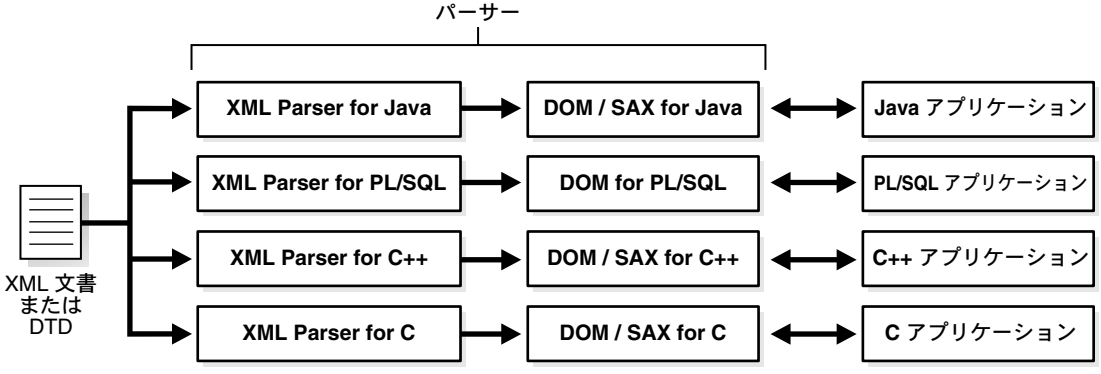


図 1-3 Oracle XML Parser: Java、C、C++、PL/SQL



XSL Transformation（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、業界標準に準拠した変換を行います。
- 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、ネイティブ・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、XSLT 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 XSL Transformation（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形式に変換できます。

XSLT プロセッサの使用方法については、第 4 章「XML Parser for Java」を参照してください。

> **参照：** 付録 A「XDK for Java: 仕様およびクイック・リファレンス」を参照してください。

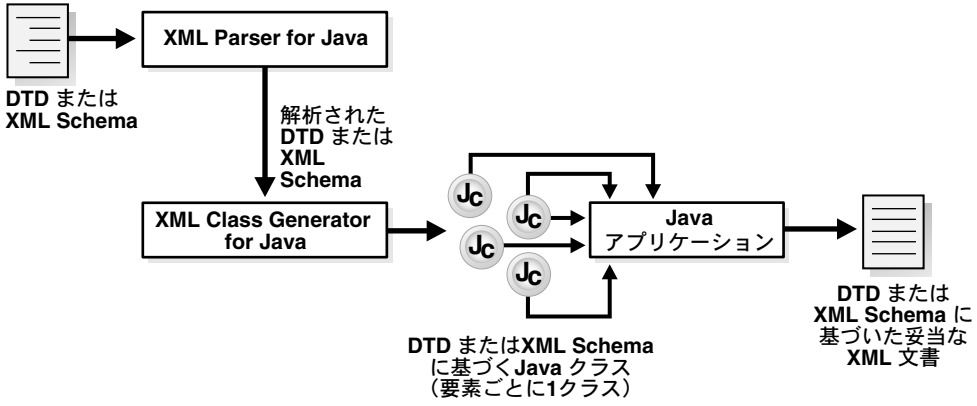
XML Class Generator

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 1-4 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 7 章「XML Class Generator for Java」
- 第 19 章「XML Class Generator for C++」

図 1-4 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 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チェック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、XSLT スタイルシートに基づいて、XML 文書を XSLT 変換します。これによって、XSLT スタイルシートを適用して、XML 文書を XML、HTML、データ定義言語（DDL）など、すべてのテキストベースの形式に変換できます。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 XSLT 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**
- **Compression Bean**
- **Differ 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。この機能については、第 10 章「XDK JavaBeans」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、XML SQL Utility および Oracle の XSL Transformation（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（OAS）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 JavaServer Pages（JSP）プラットフォーム

JavaServer Pages 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

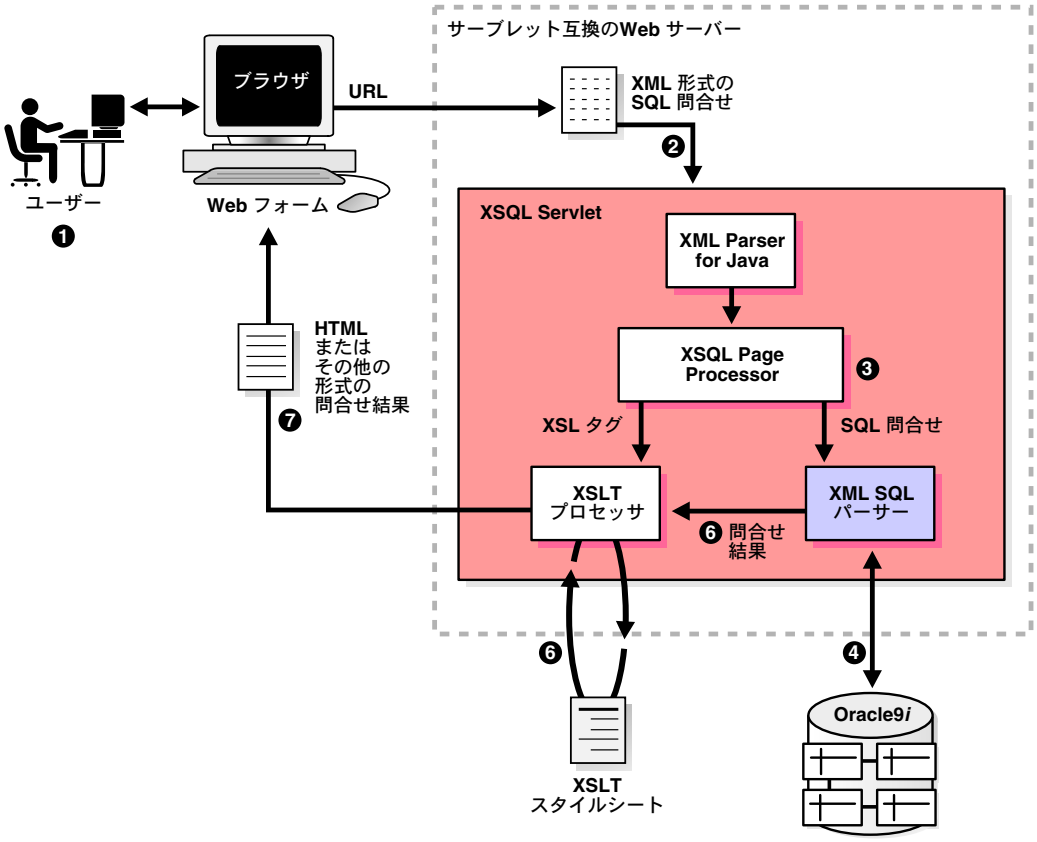
XSQL Servlet は、SQL 問合せを処理し、結果セットを XML として出力するツール製品です。このプロセッサは Java サーブレットとして実装され、埋込み SQL 問合せを含む XML ファイルを入力として受け入れます。これは、XML Parser for Java および XML SQL Utility を使用して、多くの操作を行います。

図 1-5 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 Uniform Resource Locator（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 XSLT 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 XML SQL Utility 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. XML SQL Utility が、Oracle9*i* 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 XML SQL Utility に戻します。
5. XML SQL Utility 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 XSLT 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 をリクエスト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 リクエスト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 9 章「XSQL Pages パブリッシング・フレームワーク」を参照してください。

図 1-5 XSQL Servlet の機能



Oracle XML SQL Utility（XSU）

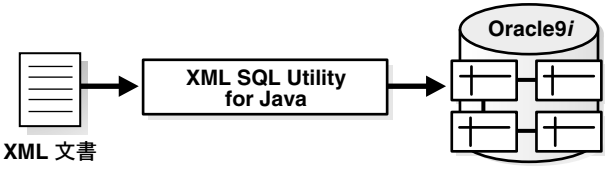
Oracle XML SQL Utility（XSU）は、Java および PL/SQL をサポートします。

- XML SQL Utility 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。

  XML SQL Utility 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、ドキュメント・オブジェクト・モデル（DOM）、Document Type Definition（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。
- XML SQL Utility for PL/SQL は、XML SQL Utility for Java をラップする PL/SQL パッケージで構成されます。

図 1-6 に、Oracle XML SQL Utility の機能の概要を示します。

図 1-6 Oracle XML SQL Utility の機能



XML SQL Utility 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。

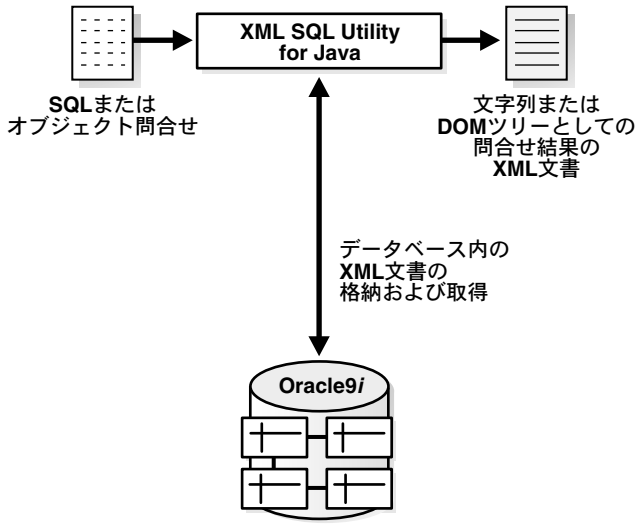
- 問合せをデータベースに渡し、結果または DTD から XML 文書（テキストまたは DOM）を生成します。DTD は、検証に使用できます。
  - XML データをデータベース表に書き込みます。

参照： 第 8 章「XML SQL Utility（XSU）」を参照してください。

問合せ結果からの XML の生成

図 1-7 に、XML SQL Utility 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 1-7 XML SQL Utility 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 ツリーとしての XML 文書の生成

XML SQL Utility（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

XML SQL Utility（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。1-10 ページの「XML Class Generator」を参照してください。

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・データが取得され、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 8 章「XML SQL Utility（XSU）」および『Oracle9*i* ケース・スタディ - XML アプリケーション』の「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 8 章「XML SQL Utility（XSU）」を参照してください。

TransX Utility

TransX Utility はデータ転送ユーティリティであり、データベースに多言語データを移入できます。TransX Utility は XML を使用してデータを指定するため、XML からデータベースへの容易なデータ転送、開発者と翻訳者の両方が直観的に使用できる単純なデータ形式、および以前の方法よりエラーの発生が低減される検証機能を利用できます。

参照： 第 12 章「Oracle TransX Utility」を参照してください。

Oracle Text

Oracle Text では、Oracle9*i* 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、Oracle9*i* 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、Oracle9*i* 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、次のマニュアルまたは URL 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`

XML Gateway

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。XML Gateway は、Oracle Advanced Queuing 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、任意のメッセージ転送エージェントを介して、ビジネス・パートナとの間で送受信されます。

参照： 次のマニュアルを参照してください。

- 『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* XML データベース開発者ガイド - Oracle XML DB』

Oracle の XML コンポーネント : XML 文書の生成

図 1-8 および図 1-11 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 Oracle9*i* 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 1-8 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
- XML SQL Utility（XSU）for Java

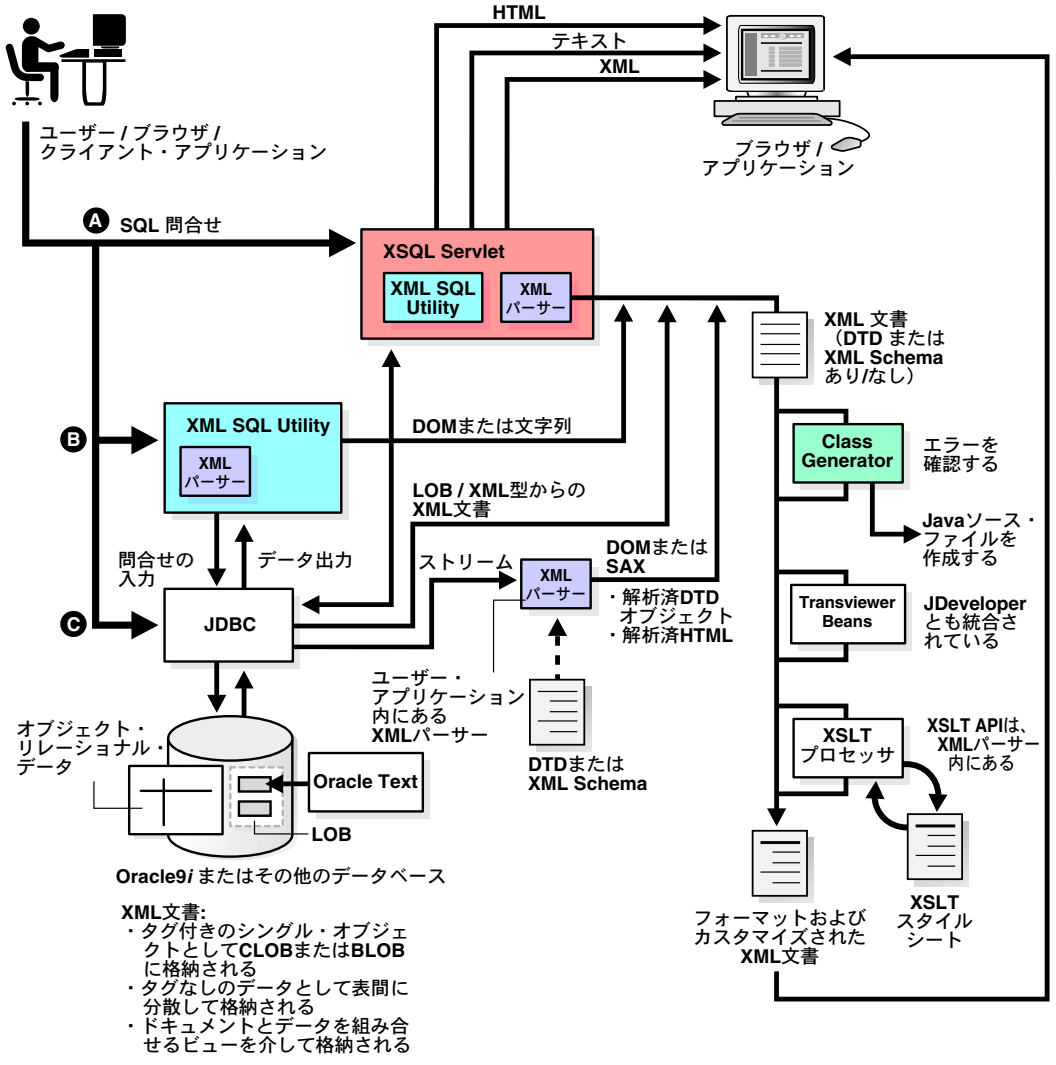
Java 環境では、ユーザー、クライアントまたはアプリケーションが問合せ（SQL）を送信した場合、Oracle の XML コンポーネントを使用した次の 3 つの方法でその問合せを処理できます。

- 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 1-8 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 1-9 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

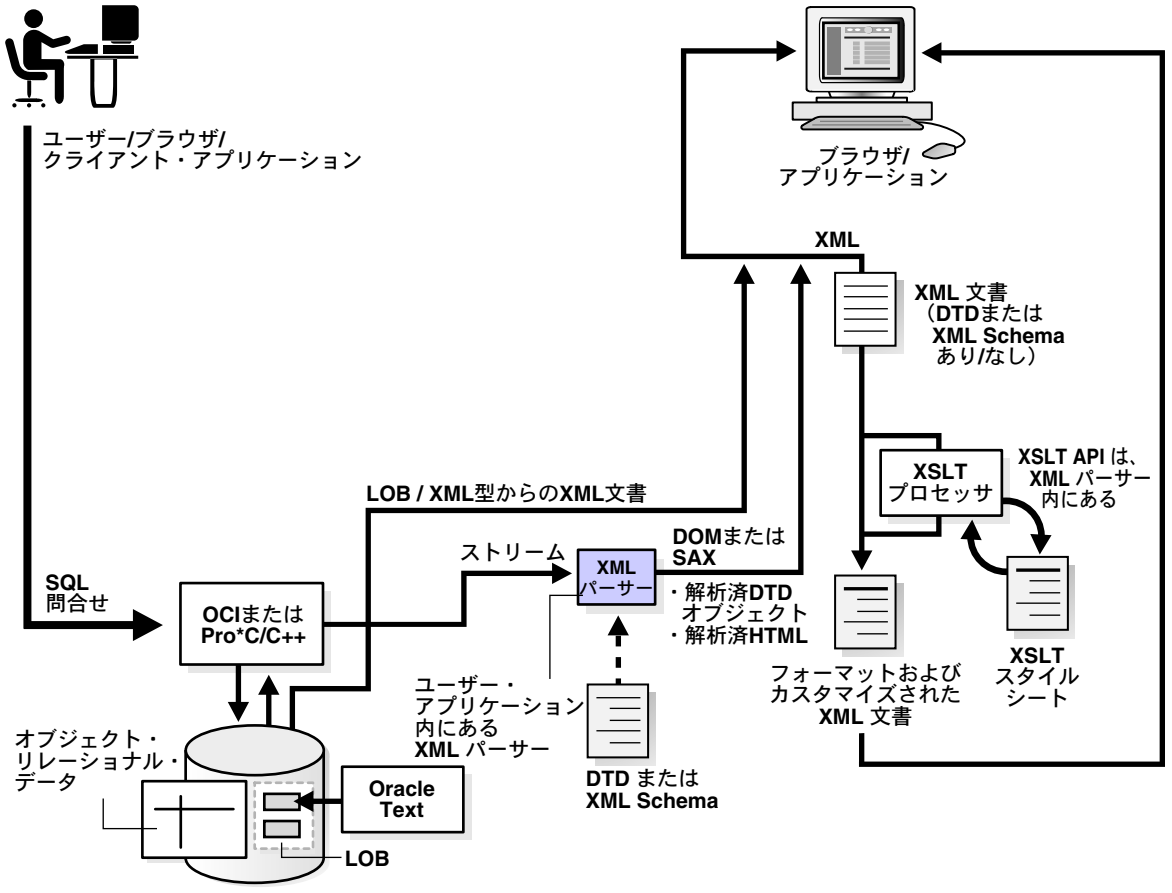
SQL 問合せは、OCI を介して、または Pro*C/C++ のプリコンパイラの埋込み文として、データベースに送信されます。

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 1-9 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 : C++

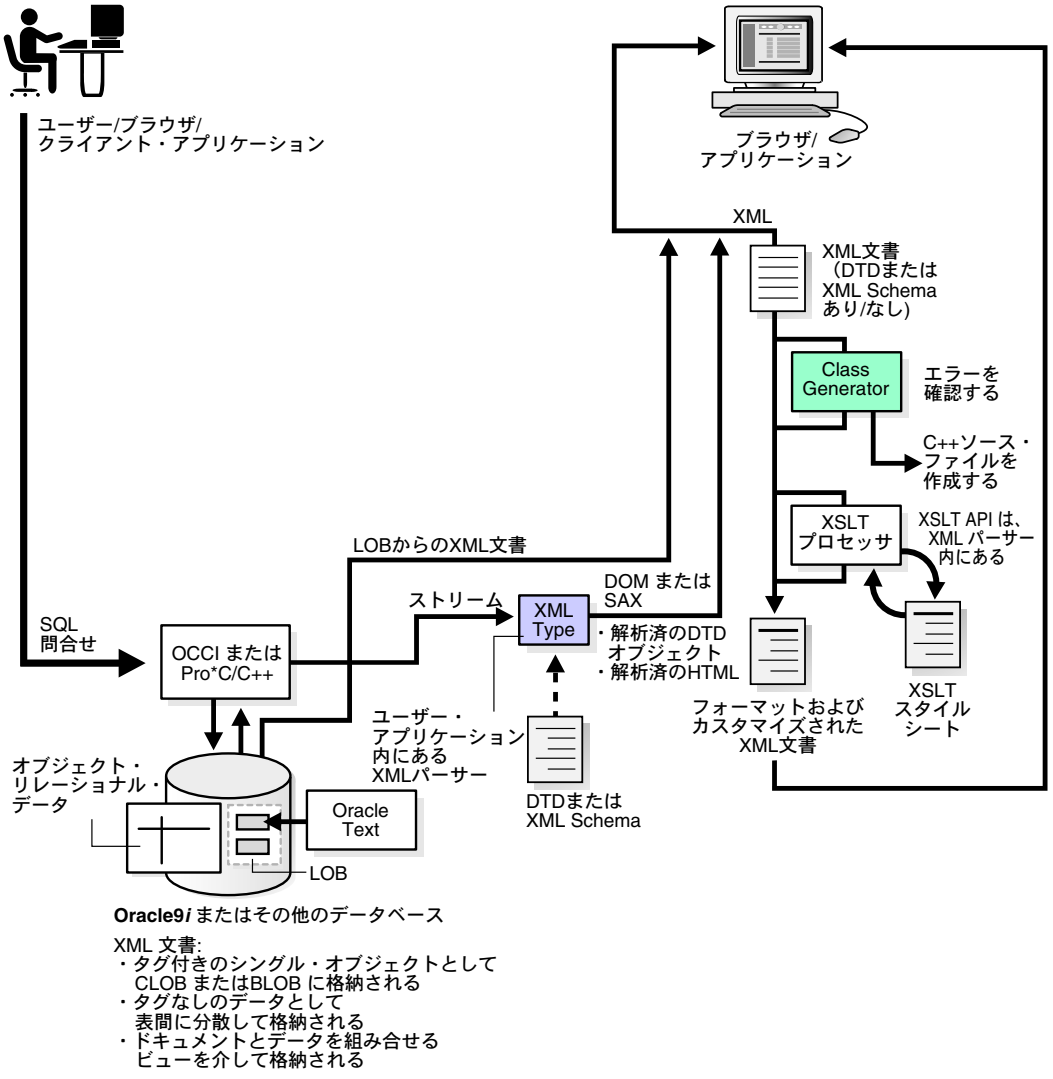
図 1-10 に、XML 文書の生成に使用される Oracle の XML コンポーネントを示します。次に、使用可能な XDK for C++ コンポーネントを示します。

- 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 1-10 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

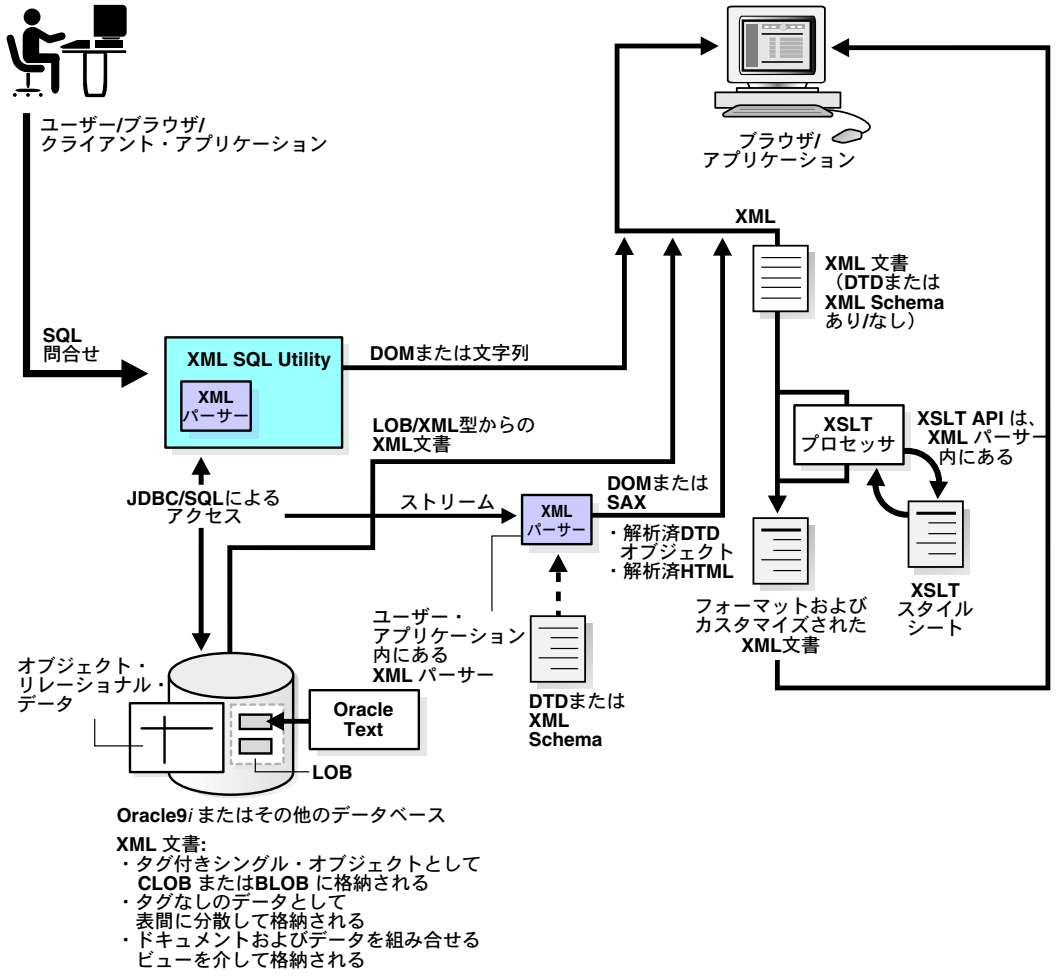
図 1-11 に、XML 文書の生成に使用される XDK for PL/SQL のコンポーネントを示します。次に、使用可能な PL/SQL 用コンポーネントを示します。

- XSLT を含む XML Parser for PL/SQL
- XML SQL Utility（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）
- XML SQL Utility（XSU）を介した処理

図 1-11 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 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

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「提案リクエスト」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 XML SQL Utility 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に必要なソフトウェア

現在、Solaris オペレーティング環境 2.6 上で Common Gateway Interface（CGI）、Perl および Oracle7 アプリケーションを使用していますが、これを XML/XSL、Java および Oracle アプリケーションに変換する予定です。Standard Generalized Markup Language（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. Oracle7 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 Virtual Machine（JVM）、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。XSQL Servlet を使用できます。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、JDBC ドライバを介して Oracle データベースに接続します。
2. Oracle7 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 8 章「XML SQL Utility（XSU）」を参照してください。
4. 回答の 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。

XML に関する質問

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/C++ で XML SQL Utility 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？

回答：

1. これは、アプリケーションの要件に依存します。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. 使用できます。
3. 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

XSLT が XML から XML、HTML またはその他のテキストベース形式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。非構造化テキスト形式の場合、次の Web サイトにある XFlat を試すことができます。

```
http://www.unidex.com/xflat.htm
```

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

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・ファイルおよびジェネレータ・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、Universal Modeling Language（UML）でのモデル化など、いくつかのガイドライン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は、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

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 : Oracle9*i* JDeveloper に、XML Schema や XSLT スタイルシートなどの XML Schema に基づく文書に使用できる、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 21 章「XSLT Processor for PL/SQL」を参照してください。

XML 文書を PDF 形式に変換する方法

Oracle8*i* 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 リリース 1.0.2 は、Apache FOP 0.14.0 と統合して、XML 入力または 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、次の Web サイトを参照してください。

```
http://xml.apache.org/fop/
```

```
http://www.xml.com/pub/rg/75
```

大規模な XML 文書をデータベースにロードする方法

大規模（27MB）でデータ中心の XML 文書があります。この文書が XML SQL Utility 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 プロセッサの実行中に、メモリー・リークが原因で 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 : XML 文書のロードが 1 回のみの場合、または 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。ローダー制御ファイルを構成するのみで使用できます。『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、ダイレクト・パスでは、SQL*Loader を使用した場合が最も高速です（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）。

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。

SQL*Loader によるネストのサポート

次の使用例について考えてみます。

```
...
     <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 はネストをサポートできません。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 -- field definition part of the SQL Loader control file

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、データベースのオブジェクト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（改行によって、完全なデータベース・レコード用のデータが区切られます）。

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・パーサー機能があります。大規模な XML 文書をロードする場合は、SQL*Loader を使用できます。

Oracle の以前のリリースに関する FAQ

異なるベンダーの XML パーサーの使用

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。

Oracle8 リリース 8.0.6 に対する XML のサポート

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。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、および XML SQL Utility などのユーティリティを含む Oracle XML Developer's Kit（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 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。

XML を使用した Oracle7 リリース 7.3.4 からサプライヤへのデータ転送

当社では Oracle7 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：Oracle7 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、XML SQL Utility を使用して、XML 形式でデータを取得することができます。

Oracle7 JDBC OCI ドライバおよび Thin JDBC ドライバについては、次の Web サイトを参照してください。

```
http://otn.oracle.com/tech/java/sqlj_jdbc/
```

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

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。

回答：XML SQL Utility および XSQL Pages フレームワークを含む Oracle XDK for Java、Oracle XDK for C および Oracle XDK for C++ は、データベース外で機能します。ライセンス（フリーのランタイム・ソフトウェアなど）については、変更はありません。最新のライセンスについては、OTN-J を参照してください。

Oracle の XML を使用した磁気テープ・ファイルの作成

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.....」のように特定の形式の文字列です。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：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に関する FAQ

XML をサポートするブラウザ

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

XML 標準に関する FAQ

XML が電子データ交換（EDI）より優れる点

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

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 Oracle Integration Server（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

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 Advanced Queueing（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。

Oracle9*i* 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、Open Applications Group、`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 に関するオラクル社の計画

回答 : 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 および XSL に関連する XML Schema、XML Query、XSL、XLink/XPointer、XML Information Set、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

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 : 次の Web サイトを参照してください。

```
http://www.xml.org
```

XML、CLOB および BLOB に関する FAQ

BLOB の XML メッセージのサポート

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形式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に関する FAQ

CLOB 格納時の XML ファイルの最大サイズ

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。サンプル・コードについては、次の URL を参照してください。

```
http://otn.oracle.com/tech/java/sqlj_jdbc/index2.htm?Code&files/advanced/advanced.htm
```

XML ファイルのサイズ制限

回答 : XML ファイルにサイズ制限はありません。

XML 文書の最大サイズ

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？

また、Oracle から XML 文書に生成された XML 文書の最大サイズを教えてください。

回答：

複数の表に PL/SQL 用のデータを指定する XML 文書の最大サイズは、1 つのオブジェクト・ビューに挿入できるサイズです。

Oracle から XML 文書に生成された XML 文書の最大サイズは、1 つのオブジェクト・ビューから取得できるサイズです。

表への XML データの挿入に関する FAQ

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

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 オペレーティング環境上で 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。

- XML SQL Utility
- XML Parser for Java
- JDBC ドライバ
- Java Developer's Kit（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 のパフォーマンスに関する FAQ

XML および Oracle のパフォーマンスに関する情報の参照先

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 文書のレコード取得

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得しました。同じレコードの取得を高速化するため、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、取得したリストがブラウザ・ウィンドウに表示されるまでに約 30 秒かかりました。count(*) を削除すると取得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 文書の取得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、オプティマイザは全表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。

複数の言語に関する FAQ

中国語の情報を XML に含める方法

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。

追加情報に関する 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/`
- `http://www.oasis-open.org/cover/`

XML および XSL 関連の推奨書籍

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』の最新の第 14 章は、`http://metalab.unc.edu/xml/books/bible/` からダウンロードできます。

  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。
- 『Oracle9*i* 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。

2

XDK for Java および XDK for JavaBeans を使用する前に

この章の内容は次のとおりです。

> **参照：** 第 3 章「XDK for C/C++ および XDK for PL/SQL を使用する前に」を参照してください。

XDK for Java のインストール

XDK for Java には、XML 文書の読込み、操作、変換および表示を行うための基本コンポーネントが含まれます。

> **注意：** 今回のリリースでは、XDK for Java および XDK for JavaBeans がバンドルされています。

Oracle XDK for Java は、次のコンポーネントで構成されます。

- XML Parser: DOM インタフェースまたは SAX インタフェースを使用した XML 文書の解析をサポートします。
- XSLT プロセッサ : XML Parser の一部として組み込まれており、XML 文書の変換をサポートします。
- XML Schema プロセッサ : XML Schema 定義ファイル（デフォルトの拡張子は `.xsd`）を使用した XML ファイルの解析および検証をサポートします。
- Class Generator: 入力 DTD または XML Schema に基づいて、一連の Java ソース・ファイルを生成します。
- XML SQL Utility: SQL 問合せから XML 文書を生成し、その文書をデータベースに挿入します。
- TransX Utility: 翻訳されたシード・データおよびメッセージをデータベースにロードする操作を簡単にします。
- XSQL Servlet: 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。

XDK for Java のインストール手順

XDK for Java は、Oracle データベースおよび Oracle Application Server に付属しています。また、XDK for Java の最新のベータ版または製品版は、OTN-J からダウンロードすることもできます。

Oracle データベースまたは Oracle9*i*AS を使用して XDK をインストールした場合は、次の手順を省略して、XDK ホーム・ディレクトリ（$XDK_HOME）に移動できます。

XDK を OTN-J からダウンロードする必要がある場合は、次の手順を実行します。

次の URL にアクセスします。

`http://otn.oracle.co.jp/tech/xml/xdk/index.html`

ページの左側にある「ダウンロード」アイコンをクリックします。

- OTN-J 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は、無償で登録できます）。

- ダウンロードするバージョンを選択します。
- ライセンス契約のすべての条件を承諾し、ソフトウェアをダウンロードします。次に、Solaris オペレーティング環境用のダウンロード・サイトに記載されている手順を示します。

```
Oracle XML Developer's Kit for Java on Sun Solaris™ Operating Environment- 9i
   Download the Complete File

   xdk_java_9_0_1_1_0A.tar.gz
   Directions

   Install  GNU gzip.

   Download the Oracle XDK for Java in .tar format

   Extract the distribution package into a directory.
   (Ex: #gzip -dc xdk_java.tar | tar xvf -)

   The result should be the following files and directories:
          /bin - xdk executables and utilities
          /lib - directory for libraries
          /xdk - top xdk directory
          /xdk/demo - directory for demo files
          /xdk/doc - directory for documentation
          /xdk/admin - direcory for dband config files
          /xdk/*html. - doc navigation files
          /xdk/license.html - copy of license agreement
```

- Windows NT の場合、.¥xdk ディレクトリおよびサブディレクトリを置くディレクトリ（Windows NT の C:¥ など）を選択し、そのディレクトリ（この場合は C:¥）に移動して、WinZip ビジュアル・ツールを使用してファイルを解凍します。

XDK for Java のコンポーネント

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-$XDK_HOME
    | - bin: executable files and setup script/batch files.
    | - lib: library files.
    | - xdk:
         | - admin: (Administration): XSU PL/SQL API setup SQL script
              and XSL Servlet Configuration file(XSQLConfig.xml).
         | - demo: demonstration code
         | - doc: documents including release notes and javadocs.
```

XDK for Java 内のすべてのパッケージは、JDK 1.2 または JDK 1.1.8 での動作が保証され、サポートされています。そのため、CLASSPATH に、次に示すすべての必要なライブラリが含まれていることを確認してください。

表 2-1 XDK for Java のライブラリ

コンポーネント	ライブラリ	注意
XML Parser	xmlparserv2.jar	XML Parser for Java バージョン 2。このパーサーには、JAXP 1.1 API、DOM API、SAX API および XSLT API が含まれます。
XSLT プロセッサ	xmlmesg.jar	XML パーサー用のメッセージ・ファイル。英語以外の言語で XML パーサーを使用する場合は、CLASSPATH にこの jar ファイルを設定する必要があります。
XML Schema プロセッサ	xschema.jar	XML Schema Processor for Java
XML SQL Utility	xsu12.jar	JDK 1.2 以上用の XML SQL Utility
	xsu111.jar	JDK 1.1.1 用の XML SQL Utility
XSQL Servlet	oraclesql.jar	Oracle XSQL Servlet
	xsqlserializers.jar	FOP/PDF 統合用の Oracle XSQL シリアライザ
	classgen.jar	XML Class Generator for Java
	transx.zip	Oracle TransX Utility

また、XML SQL Utility、XSQL Servlet および TransX Utility はいずれも、次の表に示す JDBC に依存します。

表 2-2 XDK for Java の依存ライブラリ

コンポーネント	ライブラリ	注意
JDBC	classes12.zip	JDK 1.2 以上用の JDBC
	classes111.zip	JDK 1.1.8 用の JDBC
Globalization	nls_charset12.jar	JDK 1.2 以上に対するグローバリゼーション・サポート
	nls_charset111.jar	JDK 1.1.8 に対するグローバリゼーション・サポート

XDK for Java の環境設定

次のファイルで環境を設定します。

UNIX: `$XDK_HOME/bin/env.csh`

Windows NT: `%XDK_HOME%¥bin¥env.bat`

次の表に、Windows NT での環境変数を示します。カスタマイズする必要がある変数には、「カスタマイズ」の列に「要」と記載しています。

表 2-3 Windows NT での環境設定

変数	注意	カスタマイズ
%JDBCVER%	Java 2 SDK, Standard Edition バージョン 1.3.1 のインストール・ディレクトリ	要
%JDKVER%	次の jar ファイルを含めます。 .;%XDK_HOME%¥lib¥xmmlparserv2.jar;%XDK_HOME%¥lib¥xsu12.jar;	要
%INSTALL_ROOT%	XDK のインストール・ルート（%XDK_HOME% というディレクトリ）	不要
%JAVA_HOME%	JAVA_HOME=C:¥JDK%JDKVER%	要
%CLASSPATHJ%	CLASSPATHJ=%ORACLE_HOME%¥jdbc¥lib¥classes%JDBCVER%.zip; %ORACLE_HOME%¥jdbc¥lib¥nls_charset%JDBCVER%.jar	要
%PATH%	PATH=%JAVA_HOME%¥bin;%ORACLE_HOME%¥bin;%PATH%; %INSTALL_ROOT%¥bin	不要
%CLASSPATH%	.;%CLASSPATHJ%;%INSTALL_ROOT%¥lib¥xmlparserv2.jar; %INSTALL_ROOT%¥lib¥xschema.jar; %INSTALL_ROOT%¥lib¥xsu%JDBCVER%.jar; %INSTALL_ROOT%¥lib¥oraclexsql.jar;%INSTALL_ROOT%¥lib¥classgen.jar	不要

次の表に、UNIX での環境変数を示します。カスタマイズする必要がある変数には、「要」と記載しています。

表 2-4 UNIX での環境設定

変数	注意	カスタマイズ
$JDBCVER	JDBC のバージョン。JDK 1.2 以上を使用する場合は、12 に設定する必要があります。 JDK 1.1.8 を使用する場合は、111 に設定する必要があります。	要
$JDKVER	次のコマンドで取得できる JDK のバージョン（デフォルトは 1.2.2_07） `Java -version`	要
$INSTALL_ROOT	XDK のインストール・ルート（$XDK_HOME）	不要
$JAVA_HOME	Java SDK, Standard Edition のインストール・ディレクトリ	要
$CLASSPATHJ	Java SDK にリンクされたパス。このパスは変更する必要があります。 ${ORACLE_HOME}/jdbc/lib/classes${JDBCVER}.zip: ${ORACLE_HOME}/jdbc/lib/nls_charset${JDBCVER}.jar Oracle のリレーショナル・データベース管理システム（RDBMS）がインストールされているシステム以外のシステムで XSU を実行している場合は、CLASSPATHJ パスを JDBC ライブラリ（classes12.jar）の正しい場所に更新する必要があります。特定のキャラクタ・セットをサポートするには、nls_charset12.jar が必要です。XDK for Java でのグローバリゼーション・サポートの設定を参照してください。 システムにこれらのライブラリが存在しない場合は、どちらのライブラリも、OTN-J（http://otn.oracle.co.jp）にある JDBC ドライバのダウンロードのページで入手できます。	要
$CLASSPATH	次の jar ファイルを含めます。 .:${CLASSPATHJ}:${INSTALL_ROOT}/lib/xmlparserv2.jar: ${INSTALL_ROOT}/lib/xschema.jar: ${INSTALL_ROOT}/lib/xsu${JDBCVER}.jar: ${INSTALL_ROOT}/lib/oraclexsql.jar: ${INSTALL_ROOT}/lib/classgen.jar	不要
$PATH	${JAVA_HOME}/bin:${PATH}:${INSTALL_ROOT}/bin	不要
$LD_LIBRARY_PATH	OCI/JDBC コネクション用 ${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}	不要

XSU の設定

XSU のインストールについては、3-29 ページの「XDK for PL/SQL のインストール」を参照してください。

XSQL Servlet の設定

XSQL Servlet は、任意のデータベースに対して、任意の JDBC ドライバを使用して、任意の JVM で実行できるように設計されています。実際には、最も一般的な構成でのみテストし、サポートしているテスト済の構成を記載しています。

XSQL Pages および XSQL Servlet は、次の JDK に対してのみテスト済です。

- JDK 1.1.8
- JDK 1.2.2
- JDK 1.3

これらの 3 つの JDK バージョンに対してのみ、XSQL Servlet が正常に動作することを確認済です。

注意： 多くのユーザーから、XSQL Pages および XSQL Servlet を JDK 1.1.7 で使用した場合に問題が発生することが報告されています。これらの問題は UTF-8 のキャラクタ・セット変換ルーチンで発生するため、XSQL Pages の処理には JDK 1.1.7 を使用できません。

サポートするサーブレット・コンテナ

この XSQL Servlet は、次のサーブレット・コンテナでテスト済です。

- Oracle9*i*AS Apache JServ Servlet Engine
- Oracle9*i*AS OC4J Servlet Engine
- Allaire JRun 2.3.3 および 3.0.0
- Apache 1.3.9 with JServ 1.0 および 1.1
- Apache 1.3.9 with Tomcat 3.1/3.2 Servlet Engine
- 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
- Oracle8*i* リリース 8.1.7 の Oracle Servlet Engine
- Sun JavaServer Web Development Kit（JSWDK）1.0.1 Web Server

サポートする JSP 実装

JavaServer Pages は、<jsp:forward> または <jsp:include>（あるいはその両方）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テスト済です。

- Oracle9*i*AS Apache JServ Servlet Engine
- Oracle9*i*AS OC4J Servlet Engine
- Apache 1.3.9 with Tomcat 3.1/3.2 Servlet Engine
- Apache Tomcat 3.1/3.2 Web Server + Tomcat 3.1/3.2 Servlet Engine
- Caucho Resin 1.1（ビルトイン JSP 1.0 サポート）
- NewAtlanta ServletExec 2.2/3.0 for IIS/PWS 4.0（ビルトイン JSP 1.0 サポート）
- Oracle8*i* Lite Web-to-Go Server および Oracle JSP 1.0
- Oracle8*i* リリース 8.1.7 の Oracle 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、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・コンテナ、および 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とともに動作します。

JDBC ドライバおよびデータベース

Oracle XSQL Page Processor は、Oracle JDBC Drivers で機能を最大限に活用できるように設計されていますが、妥当な JDBC ドライバを持つすべてのデータベースで有効に動作します。多くのユーザーから、他の様々な JDBC ドライバで XSQL Pages を正常に使用できることが報告されていますが、オラクル社でテスト済の JDBC ドライバは次のとおりです。

- Oracle8*i* リリース 8.1.5 の JDBC 1.*x* ドライバ
- Oracle8*i* リリース 8.1.6 の JDBC 1.*x* ドライバ
- Oracle8*i* リリース 8.1.7 の JDBC 1.*x* ドライバ
- Oracle8*i* Lite 4.0 の JDBC 1.x ドライバ
- Oracle8*i* リリース 8.1.6 の JDBC 2.0 ドライバ
- Oracle8*i* リリース 8.1.7 の JDBC 2.0 ドライバ
- Oracle9*i* リリース 1（9.0.1）の JDBC 2.0 ドライバ

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、.¥xdk¥admin¥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>
```

XSQL Pages 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. 次の XSQL Java アーカイブのリストを含めます。
   - xsu12.jar - Oracle XML SQL Utility
   - xmlparserv2.jar - Oracle XML Parser for Java
   - oraclexsql.jar- Oracle XSQL Pages
   - xsqlserializers.jar - FOP/PDF 統合用の Oracle XSQL シリアライザ
   - classes12.jar - Oracle JDBC Drivers、またはかわりに使用する JDBC ドライバ用の jar ファイル。
   - サーバーの CLASSPATH に、XSQLConfig.xml が存在するディレクトリ（デフォルトでは、./xdk/admin）も含めます。

2. oracle.xml.xsql.XSQLServlet サーブレット・クラスに、.xsql ファイル拡張子をマップします。
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 をマップします（オンライン・ヘルプおよびデモへのアクセス用）。

Oracle Internet Application Server Oracle9*i*AS リリース 1（9.0.1）以上は、XSQL Servlet を実行できるように事前構成されています。デフォルトでは、Apache JServ サーブレット・コンテナの jserv.conf には、XSQL の実行に必要な Java アーカイブを含めるためのすべての wrapper.classpath エントリが含まれています。XSQLConfig.xml ファイルは、Oracle9*i*AS のインストール・ホームである ./xdk/admin サブディレクトリにあります。

Oracle9*i*AS の Oracle Containers for Java（OC4J）サーブレット・コンテナ XSQL Servlet を Oracle9*i*AS の OC4J サーブレット・コンテナにインストールする最も簡単な方法は、XSQL Servlet をグローバル・アプリケーションとしてインストールすることです。たとえば、OC4J のインストール・ホームが C:¥j2ee¥home で、XDK ディストリビューションを C:¥xdk902 ディレクトリに解凍した場合は、次の設定手順を実行します。

1. 次の jar ファイルが C:¥j2ee¥home¥lib ディレクトリに存在することを確認します（これらのファイルは、インストール済である必要があります）。
   - xmlparserv2.jar - Oracle XML Parser for Java
   - classes12.jar - Oracle JDBC Drivers
2. 次の追加 jar ファイルを C:¥xdk902¥lib から C:¥j2ee¥home¥lib にコピーします。
   - xsu12.jar - Oracle XML SQL Utility
   - oraclexsql.jar - Oracle XSQL Pages
   - xsqlserializers.jar - FOP/PDF 統合用の Oracle XSQL シリアライザ
3. C:¥xdk¥admin¥XSQLConfig.xml 構成ファイルを C:¥j2ee¥home¥default-web-app¥WEB-INF¥classes ディレクトリにコピーします。
4. 次のとおり、サーバーの構成ファイル C:¥j2ee¥home¥config¥global-web-application.xml を編集して、<servlet> および <servlet-mapping> エントリを <web-app> 要素の子要素として追加します。

```
<orion-web-app ...and so on... >
  :
 etc
  :
  <web-app>
    <servlet>
      <servlet-name>xsql</servlet-name>
      <servlet-class>oracle.xml.xsql.XSQLServlet</servlet-class>
    </servlet>
```

```
        <servlet-mapping>
          <servlet-name>xsql</servlet-name>
          <url-pattern>/*.xsql</url-pattern>
        </servlet-mapping>
         :
        etc
         :
      </web-app>
    </web-app>
```

この時点で、任意の仮想パスに存在する任意の XSQL ページを参照でき、このページは XSQL Servlet によって処理されます。XSQL の組込みサンプル、デモおよびオンライン・ヘルプを使用する場合は、次の追加手順を実行して、/xsql/ の仮想パスを C:¥xdk¥demo¥java¥xsql ディレクトリにマップする必要があります。

次のファイルがあります。

```
c:¥j2ee¥home¥application-deployments¥default¥defaultWebApp¥orion-web.xml
```

このファイルを編集して、次の <virtual-directory> エントリを追加します。

```
<orion-web-app ...and so on...>
    :
   etc
    :
   <virtual-directory
   virtual-path="/xsql"
   real-path="/c:/xdk/xdk/demo/java/xsql/" />
    :
   etc
    :
</orion-web-app>
```

これによって、次の URL を使用してデモを参照できます。

```
 http://yoursever:yourport/xsql/index.html
```

Apache JServ 1.0 または 1.1 サーバーの CLASSPATH を XSQL Servlet 用に正しく設定します。これは、JServ 構成ファイル jserv.properties を編集して行います。XSQL Servlet ファイルを C:¥ にインストールした場合、Oracle JDBC 1.x Driver を使用するには、次のエントリを追加する必要があります。

```
# Oracle XML SQL Utility (XSU)
wrapper.classpath=C:¥xdk902¥lib¥xsu111.jar
# Oracle XSQL Servlet
wrapper.classpath=C:¥xdk902¥lib¥oraclexsql.jar
# Oracle JDBC (8.1.6) -- JDBC 1.x driver
wrapper.classpath=directory_where_JDBC_Driver_resides¥classes111.zip
```

```
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:¥xdk902¥lib¥xmlparserv2.jar
# XSQLConfig.xml File location
wrapper.classpath=directory_where_XSQLConfig.xml_resides
# FOR Apache FOP Generation, Add
# wrapper.classpath=C:¥xdk902¥lib¥xsqlserializers.jar
# wrapper.classpath=FOPHOME/fop.jar
# wrapper.classpath=FOPHOME/lib/batik.jar
```

Oracle JDBC 2.0 Driver を使用する場合は、次のようなリストになります。

```
# Oracle XML SQL Utility (XSU)
wrapper.classpath=C:¥xdk902¥lib¥xsu12.jar
# Oracle XSQL Servlet
wrapper.classpath=C:¥xdk902¥lib¥oraclexsql.jar
# Oracle JDBC (8.1.6) -- JDBC 2.0 driver
wrapper.classpath=directory_where_JDBC_Driver_resides¥classes12.zip
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:¥xdk902¥lib¥xmlparserv2.jar
# XSQLConfig.xml File location
wrapper.classpath=directory_where_XSQLConfig.xml_resides
# FOR Apache FOP Generation, Add
# wrapper.classpath=C:¥xdk902¥lib¥xsqlserializers.jar
# wrapper.classpath=FOPHOME/fop.jar
# wrapper.classpath=FOPHOME/lib/w3c.jar
```

XSQL Servlet への .xsql ファイル拡張子のマッピング これを行うには、JServ 構成ファイル jserv.conf（JServ 1.0 では、一部のプラットフォームで mod_jserv.conf）を編集する必要があります。次の行を追加します。

```
# Executes a servlet passing filename with proper extension in PATH_TRANSLATED
# property of servlet request.
# Syntax: ApJServAction [extension] [servlet-uri]
# Defaults: NONE

ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

/xsql/ 仮想ディレクトリのマッピング この手順では、仮想パス /xsql/ を C:¥xdk902¥xdk¥demo¥java¥xsql¥（または XSQL Servlet ファイルをインストールしたディレクトリ）にマップします。これを行うには、Apache 構成ファイル httpd.conf を編集し、次の行を追加する必要があります。

```
Alias /xsql/ "C:¥xdk902¥xdk¥demo¥java¥xsql¥"
```

Apache Server を再起動し、次の URL を参照します。

```
http://localhost/xsql/index.html
```

Jakarta Tomcat 3.1 または 3.2

XSQL Servlet 用のサーバーの CLASSPATH の設定 これを行うには、Tomcat サーバーを起動する前に、次のとおり、./jakarta-tomcat/bin 内の Tomcat 起動スクリプト tomcat.bat を編集し、5 つの行を追加して、適切なエントリをシステムの CLASSPATH に追加します。

Oracle JDBC 1.x Driver の場合は、次のとおりです。

```
rem Set up the CLASSPATH that we need

set cp=%CLASSPATH%

set CLASSPATH=.
set CLASSPATH=%TOMCAT_HOME%¥classes
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥webserver.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥jasper.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥xml.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥servlet.jar
set CLASSPATH=%CLASSPATH%;%JAVA_HOME%¥lib¥tools.jar

REM Added for Oracle XSQL Servlet
REM ----------------------------
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xsu111.jar
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥oraclexsql.jar
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xmlparserv2.jar
set CLASSPATH=%CLASSPATH%;directory_where_JDBC_Driver_resides¥classes111.zip
set CLASSPATH=%CLASSPATH%;directory_where_XSQLConfig.xml_resides
REM FOR Apache FOP Generation, Add
REM set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xsqlserializers.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/fop.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/lib/batik.jar
```

Oracle JDBC 2.0 Driver の場合は、次のとおりです。

```
rem Set up the CLASSPATH that we need

set cp=%CLASSPATH%

set CLASSPATH=.
set CLASSPATH=%TOMCAT_HOME%¥classes
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥webserver.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥jasper.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥xml.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%¥lib¥servlet.jar
set CLASSPATH=%CLASSPATH%;%JAVA_HOME%¥lib¥tools.jar

REM Added for Oracle XSQL Servlet
REM ----------------------------
```

```
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xsu12.jar
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥oraclexsql.jar
set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xmlparserv2.jar
set CLASSPATH=%CLASSPATH%;directory_where_JDBC_Driver_resides¥classes12.zip
set CLASSPATH=%CLASSPATH%;directory_where_XSQLConfig.xml_resides
REM FOR Apache FOP Generation, Add
REM set CLASSPATH=%CLASSPATH%;C:¥xdk902¥lib¥xsqlserializers.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/fop.jar
REM set CLASSPATH=%CLASSPATH%;FOPHOME/lib/batik.jar
```

XSQL Servlet への .xsql ファイル拡張子のマッピング Tomcat は、任意の数の構成コンテキストの作成をサポートします。これによって、ユーザーのサイトでサポートする必要がある Web アプリケーションの編成が改善されます。各コンテキストは、仮想ディレクトリ・パスにマップされ、それぞれ個別のサーブレット構成情報を含みます。XSQL Servlet には、XSQL Servlet の設定を簡単にするために事前構成されたコンテキスト・ファイルが含まれています。

デフォルトでは、Tomcat 3.1 および 3.2 には、次のコンテキスト（./jakarta-tomcat/conf/server.xml ファイル内の <Context> エントリで定義）が事前構成されています。

- ルート・コンテキスト
- /examples
- /test
- /admin

これらのいずれかのコンテキストに XSQL Servlet をインストールできますが、説明を簡単にするために、XSQL Servlet ディストリビューションのインストール・ディレクトリにマップされた、XSQL Servlet 専用の新しいコンテキストを作成します。

./jakarta-tomcat/conf/server.xml ファイルを編集して、path= "/xsql" を含む次の <Context> エントリを追加します。

```
<Context path="/test" docBase="webapps/test" debug="0" reloadable="true" />

<!--
  |  Define a Servlet context for the XSQL Servlet
  |
  |  The XSQL Servlet ships with a .\WEB-INF directory
  |  with its web.xml file preconfigured for C:¥xdk902¥xdk¥demo¥java¥xsql
  |  installation.
 +-->
<Context path="/xsql" docBase="C:¥xdk902¥xdk¥demo¥java¥xsql"/>
```

docBase= "C:¥xsql" は、XSQL Servlet ディストリビューションがインストールされている物理ディレクトリを指すことに注意してください。次に、C:¥xdk902¥xdk¥demo¥java¥xsql ディレクトリ内に WEB-INF サブディレクトリを作成し、そのサブディレクトリ内に次の ./WEB-INF/web.xml ファイルを保存する必要があります。

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <servlet>
      <servlet-name>oracle-xsql-servlet</servlet-name>
      <servlet-class>oracle.xml.xsql.XSQLServlet</servlet-class>
   </servlet>
    <servlet-mapping>
      <servlet-name>oracle-xsql-servlet</servlet-name>
      <url-pattern> *.xsql </url-pattern>
   </servlet-mapping>
</web-app>
```

> **注意：** XSQL Servlet を既存のコンテキストに追加するには、前述の web.xml ファイル内にある servlet および servlet-mapping エントリを、対象コンテキスト用の web.xml ファイルに追加します。

/xsql/ 仮想ディレクトリのマッピング これは、前述の /xsql コンテキストを作成することによって、すでに実行されています。

Tomcat サーバーを再起動し、次の URL を参照します。

```
http://localhost:8080/xsql/index.html
```

Tomcat を、DOM レベル 1 インタフェースのみをサポートする XML パーサー（Sun 社の Crimson Parser など）とともに使用する場合、Oracle XML Parser のアーカイブ xmlparser.jar が、CLASSPATH 内の DOM レベル 1 パーサーのアーカイブより前になるように tomcat.bat を編集する必要があります。たとえば、tomcat.bat を編集して、次の行を追加します。

```
REM NEED TO PUT xmlparserv2.jar FIRST before parser.jar
set CP=C:¥xdk902¥lib¥xmlparserv2.jar;%CP%
```

これらの行は、次の行の直前に追加します。

```
echo Using CLASSPATH: %CP%
echo.
set CLASSPATH=%CP%
```

XDK for Java およびグローバリゼーション・サポート

この項では、グローバリゼーション・サポートに関連する設定の概要を示します。

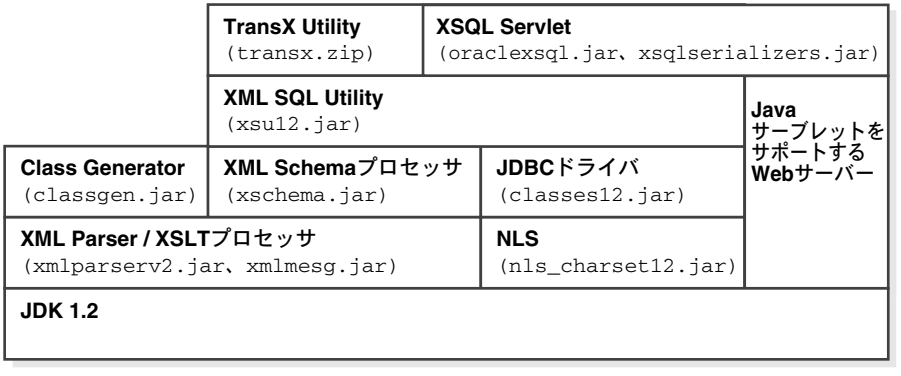
- `xmlmesg.jar` の使用 : 英語以外の言語を使用する場合は、`xmlmesg.jar` を `CLASSPATH` に設定して、パーサーがその言語で正しいメッセージを取得できるようにする必要があります。
- `nls_charset12.jar` の使用 : 次のキャラクタ・セット以外のマルチバイト・キャラクタ・セットを使用するとします。
  - UTF-8
  - ISO 8859-1
  - JA16SJIS

  この場合、XSU、TransX または XSQL Servlet のいずれかを使用した XML ファイルのロード中に、JDBC が入力ファイルのキャラクタ・セットをデータベース・キャラクタ・セットに変換できるように、この jar ファイルを Java の `CLASSPATH` に設定する必要があります。

XDK の依存性

次の図に、JDK 1.2 以上を使用する場合の XDK の依存性を示します。

図 2-1 JDK 1.2.x 以上を使用する場合の XDK の依存性



環境を正しく設定した後に、すべての必要な jar ファイルを `CLASSPATH` に含めます。これによって、Java プログラムを作成し、そのプログラムを次の `javac` コマンドを使用してコンパイルできるようになります。

```
javac your_program.java
```

エラーなしでコンパイルが完了した場合、コマンドラインまたは Web サーバーを使用してプログラムをテストできます。

> **参照：** XDK for Java のコンポーネントの詳細は、第 4 章「XML Parser for Java」を参照してください。

XDK for JavaBeans のインストール

XDK for JavaBeans を使用すると、XML アプリケーションにビジュアルまたは非ビジュアルなインタフェースを簡単に追加できます。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。

注意： 今回のリリースでは、XDK for Java および XDK for JavaBeans がバンドルされています。

Oracle XDK for JavaBeans は、次のコンポーネントで構成されます。

- DOMBuilder Bean: DOMParser をカプセル化し、XML 文書の非同期解析を提供します。
- XMLTreeViewer Bean: 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。
- XMLSourceViewer Bean: 参照および編集を簡単にするために、XML および XSL 書式のファイルの構文を色でハイライト表示します。
- XSLTransformer Bean: 入力 XML 文書を受け入れ、入力 XSLT スタイルシートで指定された変換を適用して、出力ファイルを作成します。
- XMLTransformPanel Bean: 前述の Bean を、XML ファイルの取得、変換および表示を行うためのアプリケーション・コンポーネントにカプセル化します。
- DBAccess Bean: `XMLType` をサポートし、XMLTransformPanel が対話モードで提供するすべての機能にプログラムでアクセスします。
- DBView Bean: XML およびスタイルシート変換を使用してデータベース情報をビジュアル化する必要があるすべてのアプリケーションで使用できます。
- XMLDiff Utility: 2 つの XML ファイルを比較し、その違いをビジュアル的に表すか、または生成した XSL コードによって表します。
- XMLCompression Utility: XML 文書を圧縮形式にシリアル化します。

XDK for JavaBeans は、Oracle データベースまたは Oracle9*i*AS に付属しています。また、XDK for JavaBeans の最新版は、OTN からダウンロードすることもできます。

Oracle データベースまたは Oracle9*i*AS を使用して XDK をインストールした場合は、次の手順を省略して、XDK ホーム・ディレクトリ（$XDK_HOME）を直接参照できます。

XDK を OTN からダウンロードする必要がある場合は、次の手順を実行します。

ブラウザで、次の URL にアクセスします。

http://otn.oracle.com/tech/xml/xdk_jbeans/index.html

ページの左側にある「Software」アイコンをクリックします。

- 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は、無償で登録できます）。
- ダウンロードするバージョンを選択します。
- ライセンス契約のすべての条件を承諾し、ソフトウェアをダウンロードします。次に、Solaris オペレーティング環境用のダウンロード・サイトに記載されている手順を示します。

```
Oracle XML Developer's Kit for Java on Sun Solaris™ Operating Environment- 9i
Download the Complete File
xdk_java_9_0_2_0_0C.tar.gz

Directions

      Install GNU gzip.
      Download the Oracle XDK for JavaBeans in .tar format
      Extract the distribution package into a directory.

(Ex: #gzip -dc xdk_java.tar | tar xvf -)

The result should be the following files and directories:
/bin - xdk executables and utilities
/lib - directory for libraries
/xdk - top xdk directory
/xdk/demo - directory for demo files
/xdk/doc - directory for documentation
/xdk/admin - direcory for dband config files
/xdk/*html - doc navigation files
/xdk/license.html - copy of license agreement
```

- Windows NT の場合、.¥xdk ディレクトリおよびサブディレクトリを置くディレクトリ（Windows NT の C:¥ など）を選択し、そのディレクトリ（この場合は C:¥）に移動して、WinZip ビジュアル・ツールを使用してファイルを解凍します。

XDK for JavaBeans のコンポーネント

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-$XDK_HOME
| - bin: executable files and setup script/batch files.
| - lib: library files.
| - xdk
     | - admin (Administration): XSU PL/SQL API setup SQL script and XSL Servlet
           Configuration file (XSQLConfig.xml).
     | - demo: demonstration code
     | - doc: documents including release notes and javadocs.
```

XDK for JavaBeans 内のすべてのパッケージは、JDK 1.2 または 1.1.8 での動作が保証され、サポートされています。そのため、CLASSPATH に、すべての必要なライブラリが含まれていることを確認してください。

JDK 1.2 未満のバージョンでは、次のページの「Java Foundation Classes (JFC)/Swing 1.0.3」にある Swing ライブラリ swingall.jar の他に、JDK ライブラリを CLASSPATH に含める必要があります。

http://java.sun.com/products/archive/index.html

次の表に、XDK for JavaBeans のライブラリを示します。

表 2-5 XDK for JavaBeans のライブラリ

コンポーネント	ライブラリ	注意
XML Parser	xmlparserv2.jar	XML Parser for Java バージョン 2。このパーサーには、JAXP 1.1 API、DOM API、SAX API および XSLT API が含まれます。
XSLT プロセッサ	xmlmesg.jar	XML パーサー用のメッセージ。英語以外の言語で XML パーサーを使用する場合は、CLASSPATH にこの jar ファイルを設定する必要があります。
XML Schema プロセッサ	xschema.jar	XML Schema Processor for Java

表 2-5 XDK for JavaBeans のライブラリ（続き）

コンポーネント	ライブラリ	注意
XML SQL Utility	xsu12.jar	JDK 1.2 以上用の XML SQL Utility
	xsu111.jar	JDK 1.1.8 用の XML SQL Utility
	oraclexsql.jar	Oracle XSQL Servlet
	xsqlserializers.jar	FOP/PDF 統合用の Oracle XSQL シリアライザ
Class Generator	classgen.jar	Class Generator for Java
TransX Utility	transx.zip	Oracle TransX Utility
JavaBeans	xmlcomp.jar	Oracle JavaBeans Utilities
	xmlcomp2.jar	

また、XML SQL Utility、XSQL Servlet および TransX Utility はいずれも、他のコンポーネントに依存します。次の表に、それらのコンポーネントのライブラリを示します。

表 2-6 XDK for JavaBeans: 依存ライブラリ

コンポーネント	ライブラリ	注意
JDBC	classes12.zip	JDK 1.2 以上用の JDBC
	classes111.zip	JDK 1.1.8 用の JDBC
Globalization	nls_charset12.jar	JDK 1.2 以上に対するグローバリゼーション・サポート
	nls_charset111.jar	JDK 1.1.8 に対するグローバリゼーション・サポート
XMLType	xdb_g.jar	Java API for XMLType（$ORACLE_HOME/rdbms/jlib）
Jdev Runtime	jdev-rt.zip	Java Graphical User Interface（GUI）ライブラリ

XDK for JavaBeans の環境設定

UNIX の場合は、次のスクリプト・ファイルを使用します。

```
$XDK_HOME/bin/env.csh
```

Windows の場合は、次のバッチ・ファイルを使用します。

```
%XDK_HOME%¥bin¥env.bat
```

次の表に、XDK の設定時に必要な環境変数を示します。前述のスクリプトまたはバッチ・ファイルを実行する前にカスタマイズする必要がある変数には、「カスタマイズ」の列に「要」と記載しています。

表 2-7 UNXI での JavaBeans の環境設定

変数	注意	カスタマイズ
$JDBCVER	JDBC のバージョン。JDK 1.2 以上を使用する場合は、12 に設定する必要があります。	要
$JDKVER	次のコマンドで取得できる JDK のバージョン（デフォルトは 1.2.2_07） `Java -version`	要
$INSTALL_ROOT	XDK のインストール・ルート（$XDK_HOME）	不要
$JAVA_HOME	Java SDK, Standard Edition のインストール・ディレクトリ	要
$CLASSPATHJ	Java SDK にリンクされたパス。このパスは変更する必要があります。 ${ORACLE_HOME}/jdbc/lib/classes${JDBCVER}.zip: ${ORACLE_HOME}/jdbc/lib/nls_charset${JDBCVER}.jar Oracle のリレーショナル・データベース管理システム（RDBMS）がインストールされているシステム以外のシステムで XSU を実行している場合は、CLASSPATHJ パスを JDBC ライブラリ（classes12.jar）の正しい場所に更新する必要があります。 特定のキャラクタ・セットをサポートするには、nls_charset12.jar が必要です。XDK for JavaBeans でのグローバリゼーション・サポートの設定を参照してください。 システムにこれらのライブラリが存在しない場合は、どちらのライブラリも、OTN-J（http://otn.oracle.co.jp）にある JDBC ドライバのダウンロードのページで入手できます。	要

表 2-7 UNXI での JavaBeans の環境設定（続き）

変数	注意	カスタマイズ
$CLASSPATH	次の jar ファイルを含めます。 .:${CLASSPATHJ}:${INSTALL_ROOT}/lib/xmlparserv2.jar: ${INSTALL_ROOT}/lib/xschema.jar: ${INSTALL_ROOT}/lib/xsu${JDBCVER}.jar: ${INSTALL_ROOT}/lib/oraclexsql.jar: ${INSTALL_ROOT}/lib/classgen.jar	不要
$PATH	${JAVA_HOME}/bin:${PATH}:${INSTALL_ROOT}/bin	不要
$LD_LIBRARY_PATH	JDBC/OCI コネクション用 ${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}	不要

Windows NT での設定については、次の表を参照してください。

表 2-8 Windows NT での JavaBeans の環境設定

変数名	注意	カスタマイズ
%JDBCVER%	JDBC のバージョン。JDK 1.2 以上を使用する場合は 12、JDK 1.1.8 を使用する場合は 111 です。	要
%JDKVER%	次のコマンドで取得される JDK のバージョン（デフォルトは 1.2.2_07） `Java -version`	要
%INSTALL_ROOT%	XDK のインストール・ルート（%XDK_HOME%）	不要
%JAVA_HOME%	Java SDK, Standard Edition のインストール・ディレクトリ。Java SDK にリンクされたパス。このパスは変更する必要があります。	要
%CLASSPATHJ%	CLASSPATHJ=%ORACLE_HOME%¥jdbc¥lib¥classes%JDBCVER%.zip; %ORACLE_HOME%¥jdbc¥lib¥nls_charset%JDBCVER%.jar	要
%PATH%	PATH=%JAVA_HOME%¥bin;%ORACLE_HOME%¥bin;%PATH%;%INSTALL_ROOT%¥bin	不要
%CLASSPATH%	.;%CLASSPATH%;%INSTALL_ROOT%¥lib¥xmlparserv2.jar; %INSTALL_ROOT%¥lib¥xschema.jar; %INSTALL_ROOT%¥lib¥xsu%JDBCVER%.jar;%INSTALL_ROOT%¥lib¥oraclexsql.jar;%INSTALL_ROOT%¥lib¥classgen.jar	不要

XDK for JavaBeans およびグローバリゼーション・サポート

この項では、グローバリゼーション・サポートに関連する設定の概要を示します。

- 英語以外の言語を使用する場合は、`xmlmesg.jar` を `CLASSPATH` に設定して、パーサーがその言語で正しいメッセージを取得できるようにします。
- 次のキャラクタ・セット以外のマルチバイト・キャラクタ・セットを使用するとします。
  - UTF-8
  - ISO 8859-1
  - JA16SJIS

  この場合、XML ファイルのロード中に、JDBC が入力ファイルのキャラクタ・セットをデータベース・キャラクタ・セットに変換できるように、`nls_charset12.jar` を Java の `CLASSPATH` に設定します。

  参照： XDK for JavaBeans のコンポーネントの詳細は、第 10 章「XDK JavaBeans」を参照してください。

3

XDK for C/C++ および XDK for PL/SQL を使用する前に

この章の内容は次のとおりです。

参照： 第 2 章「XDK for Java および XDK for JavaBeans を使用する前に」を参照してください。

XDK for C のインストール

XDK for C には、XML 文書の読込み、操作および変換を行うための基本コンポーネントが含まれます。

> **注意：** 今回のリリースでは、XDK for C および XDK for C++ がバンドルされています。

Oracle XDK for C は、次のコンポーネントで構成されます。

- XML Parser: DOM インタフェースまたは SAX インタフェースを使用した XML 文書の解析をサポートします。
- XSLT プロセッサ : XML 文書の変換をサポートします。
- XML Schema プロセッサ : XML Schema 定義ファイル（デフォルトの拡張子は `.xsd`）を使用した XML ファイルの解析および検証をサポートします。

XDK for C の入手

Oracle データベースまたは Oracle9*i*AS がインストールされている場合、XDK for C はすでにインストールされています。

XDK for C の最新版は、OTN からダウンロードすることもできます。

XDK を OTN からダウンロードするには、次の手順を実行します。

- ブラウザで次の URL にアクセスします。

  `http://otn.oracle.com/tech/xml/xdk_c/content.html`

- ページの左側にある「Software」アイコンをクリックします。
- 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は、無償で登録できます）。
- ダウンロードするバージョンを選択します。
- ライセンス契約のすべての条件を承諾します。
- 適切なファイルをクリックします。
- ダウンロードしたファイルを解凍します。

XDK（XDK for C を使用する場合）のダウンロードの詳細は、2-1 ページの「XDK for Java および XDK for JavaBeans を使用する前に」を参照してください。

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-$XDK_HOME
     | - bin: executable files
     | - lib: library files.
     |- nlsdata: Globalization Support data files(*.nlb)
     | - xdk
          | - demo: demonstration code
          | - doc: documents including release notes.
          | - include: header files.
          | - mesg: message files. (*.msb)
```

次に、XDK for C の UNIX バージョンに付属するすべてのライブラリを示します。

表 3-1 XDK for C のライブラリ

コンポーネント	ライブラリ	注意
XML Parser	libxml9.a	XML Parser for C バージョン 2。このパーサーには、DOM API、SAX API および XSLT API が含まれます。
XSLT プロセッサ		
XML Schema プロセッサ	libxsd9.a	XML Schema Processor for C

XDK for C（UNIX）は、次の表に示す Oracle の CORE ライブラリおよび Globalization Support ライブラリに依存します。

表 3-2 XDK for C の依存ライブラリ（UNIX）

コンポーネント	ライブラリ	注意
CORE ライブラリ	xmlparser	Oracle CORE ライブラリ
Globalization Support ライブラリ	libnls9.a	Oracle Globalization Support 共通ライブラリ
	libunls9.a	Unicode をサポートするための Oracle Globalization Support ライブラリ

UNIX での環境設定

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されているかどうかを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
setenv ORA_NLS33 ${ORACLE_HOME}/ocommon/nls/admin/data
```

Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
setenv ORA_NLS33 ${XDK_HOME}/nlsdata
```

環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されているかどうかを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
setenv ORA_NLS33 ${ORACLE_HOME}/xdk/mesg
```

Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

```
setenv ORA_NLS33 ${XDK_HOME}/xdk/mesg
```

現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

これで、Make ファイルを使用して、デモ・コードをコンパイルおよびリンクできます。

Windows NT での環境設定

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-%XDK_HOME%
   | - bin: executable files and dynamic libraries
   | - lib: static library files.
   |- nlsdata: Globalization Support data files (*.nlb)
   | - xdk
         | - demo: demonstration code
         | - doc: documents including release notes.
         | - include: header files.
         | - mesg: message files. (*.msb)
```

次に、XDK for C に付属する Widows NT ライブラリを示します。

表 3-3 XDK for C のライブラリ（Windows NT）

コンポーネント	ライブラリ	注意
XML Parser XSLT プロセッサ	oraxml9.lib oraxml9.dll	XML Parser for C バージョン 2。このパーサーには、DOM API、SAX API および XSLT API が含まれます。
XML Schema プロセッサ	oraxsd9.a oraxsd9.dll	XML Schema Processor for C

XDK for C（Windows NT）は、次の表に示す Oracle の CORE ライブラリおよび Globalization Support ライブラリに依存します。

表 3-4 XDK for C の依存ライブラリ（Windows NT）

コンポーネント	ライブラリ	注意
CORE ライブラリ	oracore9.a	Oracle CORE ライブラリ
Globalization Support ライブラリ	oranls9.a oranls9.dll	Oracle Globalization Support 共通ライブラリ
	oraunls9.a oraunls9.dll	Unicode をサポートするための Oracle Globalization Support ライブラリ

コマンドラインを使用するための環境

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥nlsrtl¥admin¥nlsdata
```

Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
set ORA_NLS33 =%XDK_HOME%¥nlsdata
```

環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されているかどうかを確認する必要があります。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥xdk¥mesg
```

Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

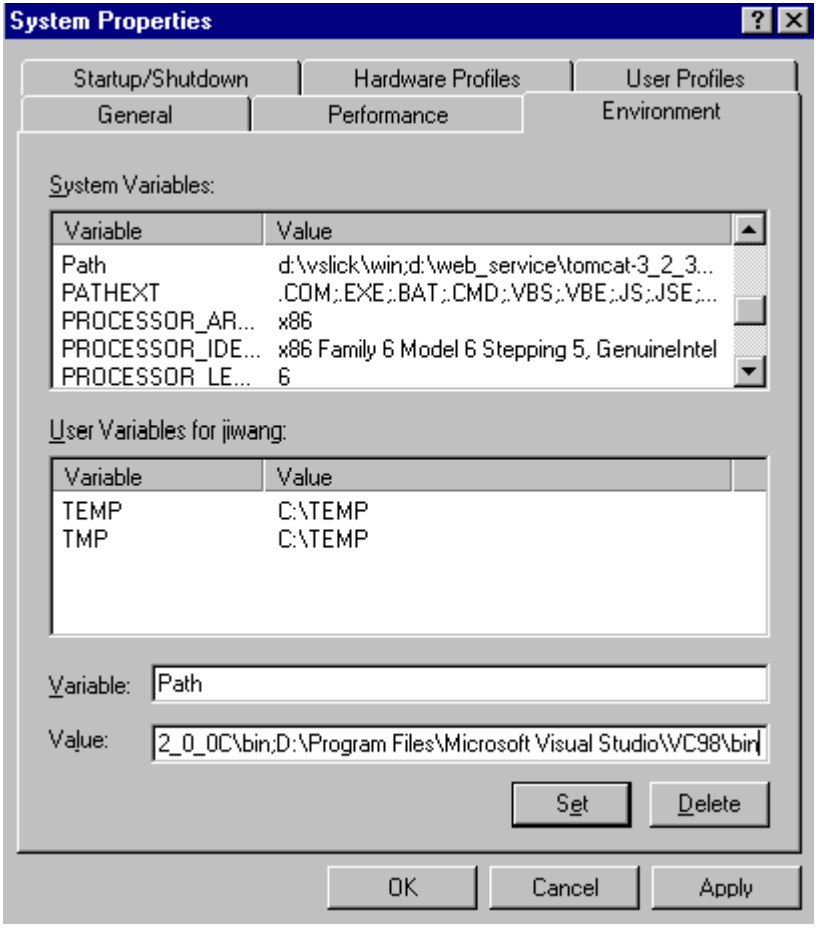
```
set ORA_NLS33 =%XDK_HOME%¥xdk¥mesg
```

現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

コマンドライン環境で（Make.bat を使用してコードをコンパイルする必要がある場合）は、cl コンパイラのパスを設定します。

「スタート」>「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、図 3-1「Windows NT での cl コンパイラのパスの設定」に示す Path 変数に cl.exe のパスを入力します。

図 3-1 Windows NT での cl コンパイラのパスの設定



次の Make.bat ファイルの例に示すとおり、COMPILE および LINK コマンドにライブラリおよびヘッダー・ファイルのパスを追加することによって、Make.bat を更新する必要があります。

```
...
: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 /Doratext=OraText -I.
-I..¥..¥..¥include -
ID:¥Progra~1¥Micros~1¥VC98¥Include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..¥..¥..¥..¥bin¥%filename%.exe /libpath:%ORACLE_HOME%¥lib
/libpath:D:¥Progra~1¥Micros~1¥VC98¥lib /libpath:..¥..¥..¥..¥lib %filename%.obj
oraxml9.lib oracore9.lib oranls9.lib oraunls9.lib user32.lib kernel32.lib msvcrt.lib
ADVAPI32.lib oldnames.lib winmm.lib
:EOF
```

ここで、各部分は次の内容を表します。

D:¥Progra~1¥Micros~1¥VC98¥Include: ヘッダー・ファイルのパス
D:¥Progra~1¥Micros~1¥VC98¥lib: ライブラリ・ファイルのパス

Visual C++ での XDK for C の使用

コンパイラに Microsoft Visual C++ を使用している場合について考えてみます。

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥nlsrtl¥admin¥nlsdata
```

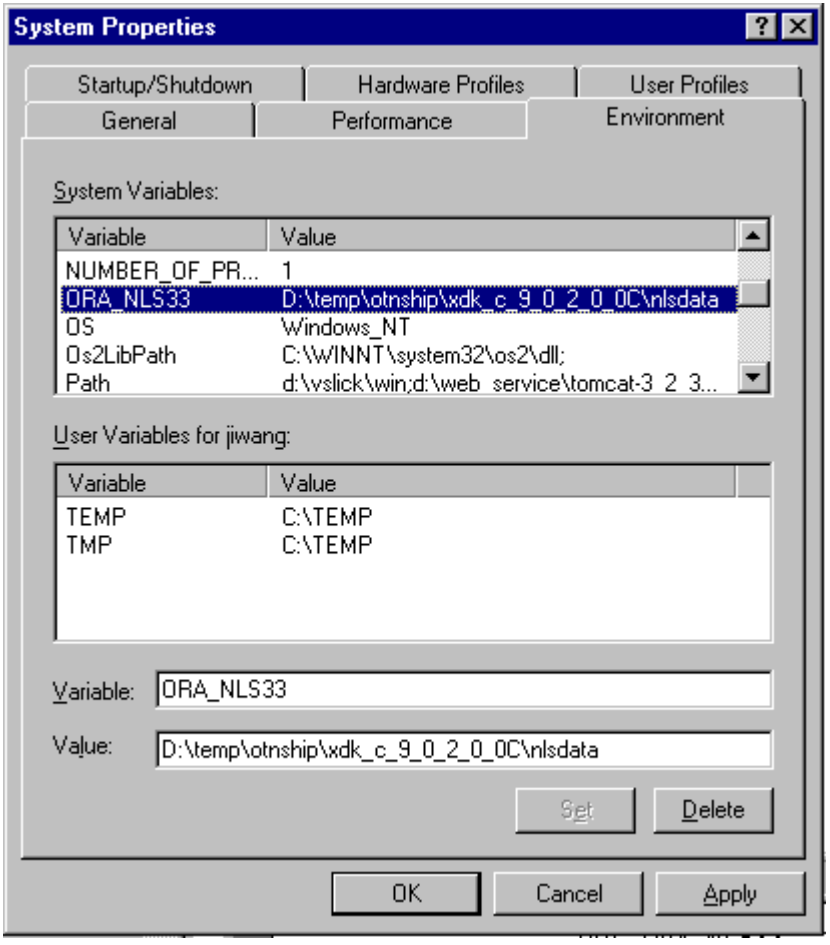
Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
set ORA_NLS33 =%XDK_HOME%¥nlsdata
```

Visual C++ を使用するには、Windows NT のシステム設定を使用して、この環境変数を定義する必要があります。

「スタート」>「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、ORA_NLS33 を入力します。

図 3-2 環境変数 ORA_NLS33 の設定



環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥xdk¥mesg
```

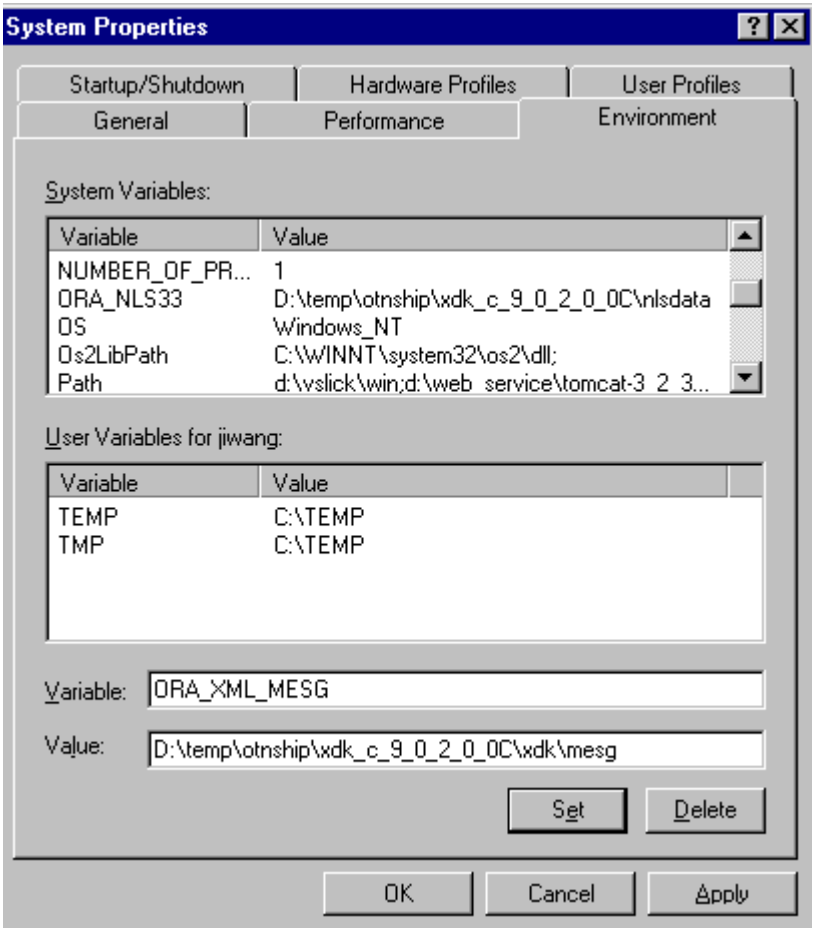
Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

```
set ORA_NLS33 =%XDK_HOME%¥xdk¥mesg
```

Visual C++ でこの環境変数を使用するには、Windows NT のシステム設定を使用して、この環境変数を定義する必要があります。

「スタート」>「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、ORA_XML_MESG を入力します。

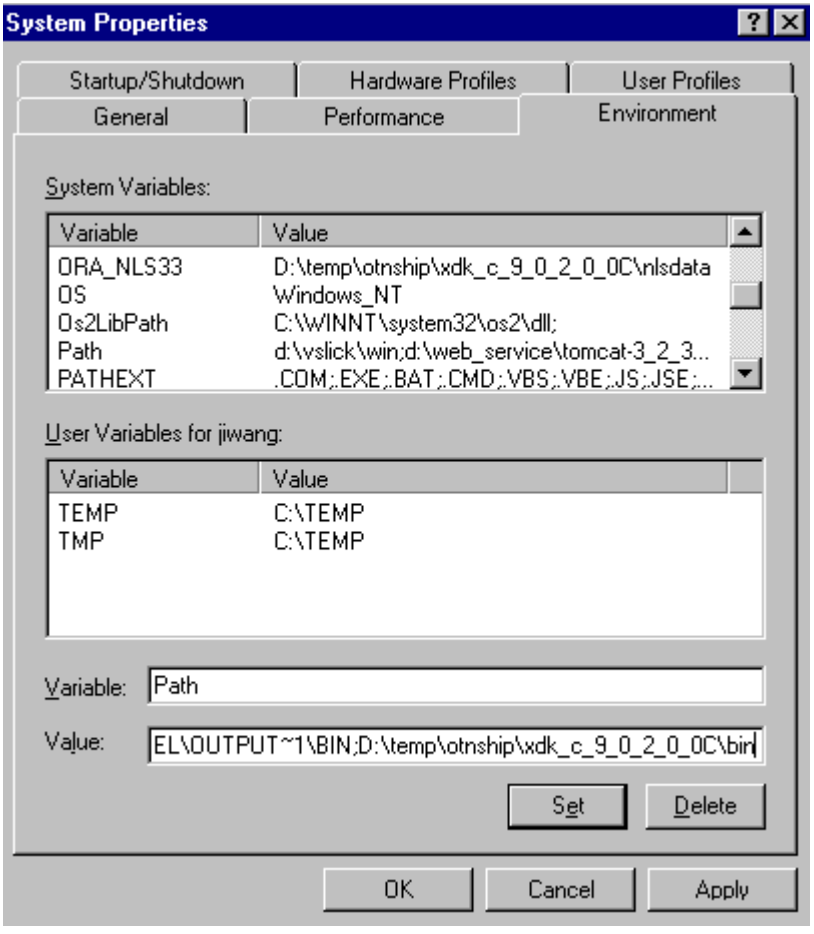
図 3-3 環境変数 ORA_XML_MESG の設定



現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

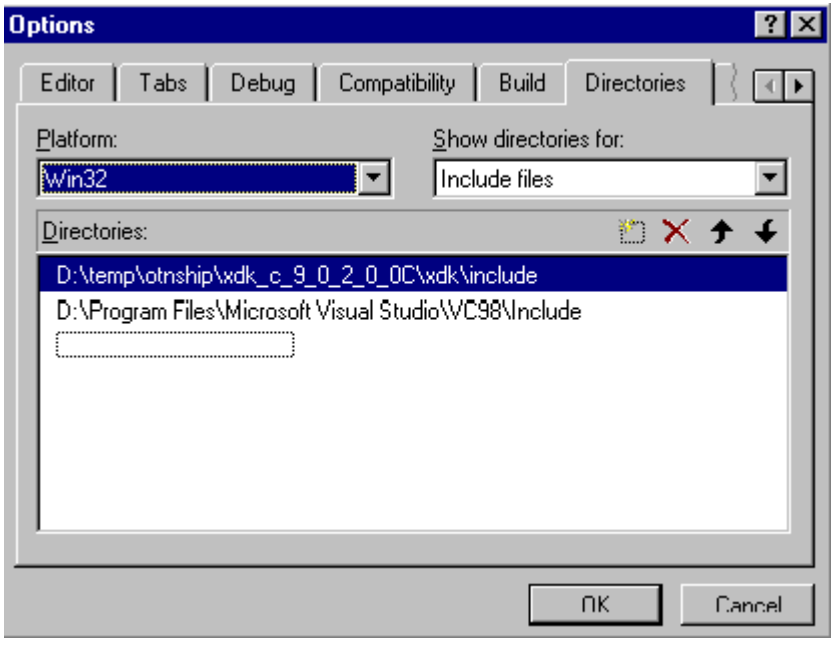
次の図に、Dynamic Link Library（DLL）のパスの設定を示します。

図 3-4 DLL のパスの設定



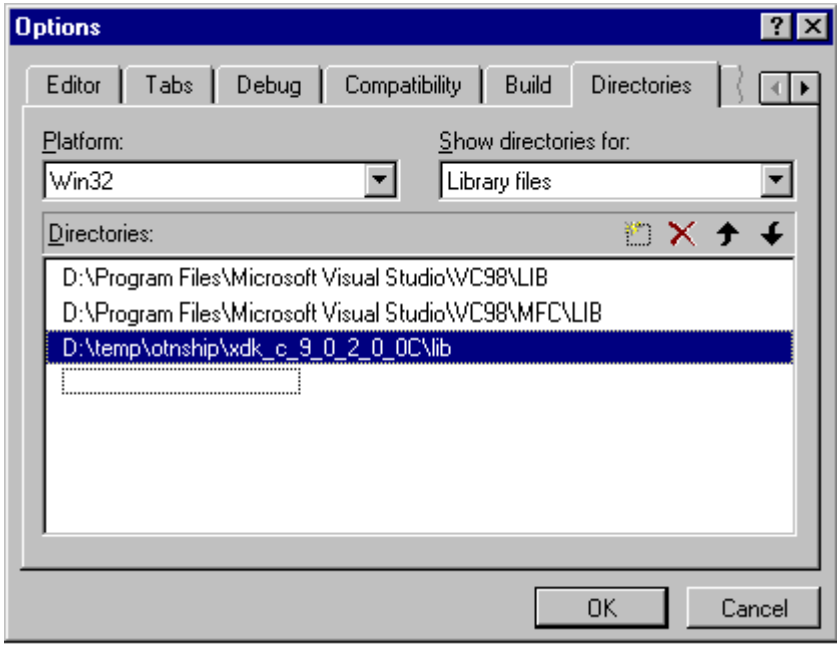
Visual C++ で作業領域をオープンし、プロジェクト用の *.c ファイルをインクルードした後で、プロジェクトのパスを設定する必要があります。「Tools」メニュー >「Options」を選択します。ウィンドウが表示されます。「Directories」タブを選択し、次の図に示すとおり、インクルード・パスを設定します。

図 3-5 Visual C++ でのインクルード・パスの設定



その後、次の図に示すとおり、ライブラリ・パスを設定します。

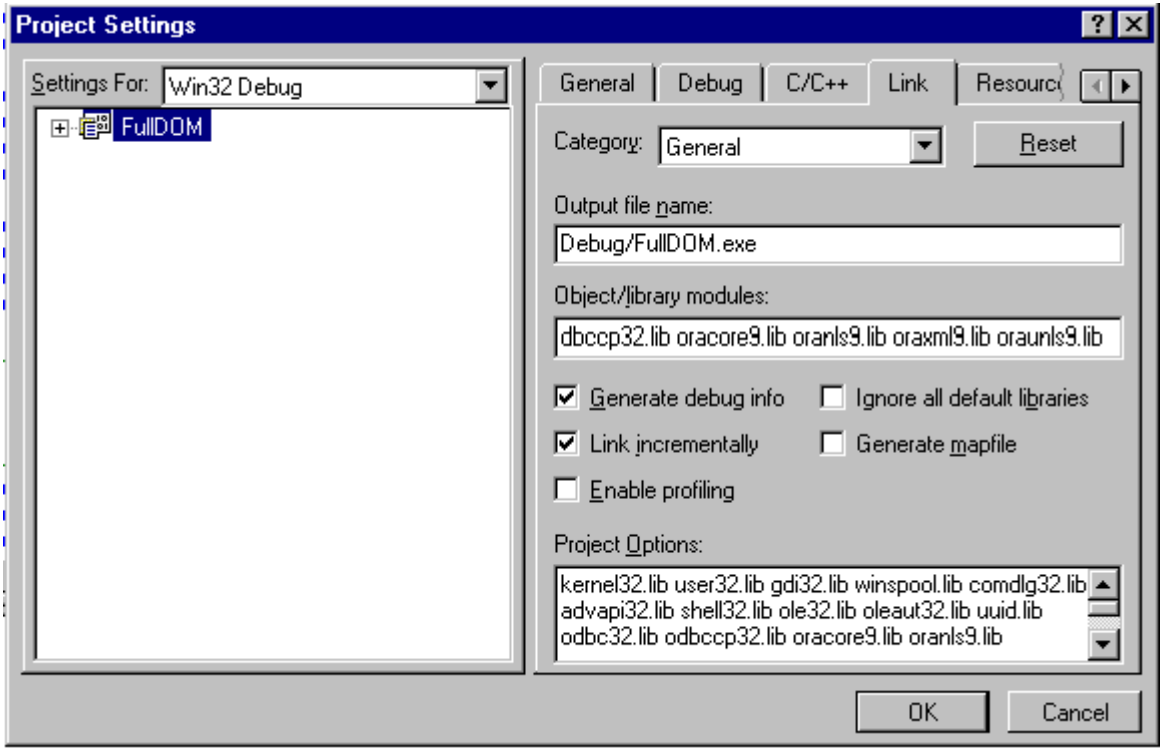
図 3-6 静的ライブラリ・パスの設定



%XDK_HOME%¥lib 内の静的ライブラリのパスを設定した後、Visual C++ のコンパイル環境でライブラリ名を設定する必要もあります。

メニュー・バーで「Project」メニュー >「Settings」を選択します。ウィンドウが表示されます。次の図に示すとおり、「Link」タブを選択し、「Object/library modules」フィールドに XDK for C ライブラリの名前を入力します。

図 3-7 Visual C++ プロジェクトでの静的ライブラリの設定



デモ・プログラムをコンパイルおよび実行し、XDK for C の使用を開始します。

参照： XDK for C のコンポーネントの詳細は、第 13 章「XML Parser for C」を参照してください。

XDK for C++ のインストール

XDK for C++ には、XML 文書の読込み、操作および変換を行うための基本コンポーネントが含まれます。

注意： 今回のリリースでは、XDK for C および XDK for C++ がバンドルされています。

Oracle XDK for C++ は、次のコンポーネントで構成されます。

- XML Parser: DOM インタフェースまたは SAX インタフェースを使用した XML 文書の解析をサポートします。
- XSLT プロセッサ : XML 文書の変換をサポートします。
- XML Schema プロセッサ : XML Schema 定義ファイル（デフォルトの拡張子は `.xsd`）を使用した XML ファイルの解析および検証をサポートします。
- Class Generator for C++: 入力 DTD または XML Schema に基づいて、一連の C++ ソース・ファイルを生成します。

XDK for C++ の入手

Oracle データベースまたは Oracle9*i*AS がインストールされている場合、XDK for C++ はすでにインストールされています。

XDK for C++ の最新版は、OTN からダウンロードすることもできます。

XDK を OTN からダウンロードするには、次の手順を実行します。

- ブラウザで次の URL にアクセスします。

  `http://otn.oracle.com/tech/xml/xdk_cpp/content.html`

- ページの左側にある「Software」アイコンをクリックします。
- 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は、無償で登録できます）。
- ダウンロードするバージョンを選択します。
- ライセンス契約のすべての条件を承諾します。
- 適切なファイルをクリックします。
- ダウンロードしたファイルを解凍します。

XDK（XDK for C++ を使用する場合）のダウンロードの詳細は、2-1 ページの「XDK for Java および XDK for JavaBeans を使用する前に」を参照してください。

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-$XDK_HOME
     | - bin: executable files
     | - lib: library files.
     |- nlsdata: Globalization Support data files(*.nlb)
     | - xdk
          | - demo: demonstration code
          | - doc: documents including release notes.
          | - include: header files.
          | - mesg: message files. (*.msb)
```

次の表に、XDK for C++ の UNIX バージョンに付属するライブラリを示します。

表 3-5 XDK for C++ のライブラリ（UNIX）

コンポーネント	ライブラリ	注意
XML Parser XSLT プロセッサ	libxml9.a	XML Parser for C++ バージョン 2。このパーサーには、DOM API、SAX API および XSLT API が含まれます。
XML Schema プロセッサ	libxsd9.a	XML Schema Processor for C++
Class Generator	libxmlg.a	Class Generator for C++

XDK for C++ パッケージは、次の表に示す Oracle の CORE ライブラリおよび Globalization Support ライブラリに依存します。

表 3-6 XDK for C++ の依存ライブラリ（UNIX）

コンポーネント	ライブラリ	注意
CORE ライブラリ	xmlparser	Oracle CORE ライブラリ
Globalization Support ライブラリ	libnls9.a	Oracle Globalization Support 共通ライブラリ
	libunls9.a	Unicode をサポートするための Oracle Globalization Support ライブラリ

UNIX での C++ 環境の設定

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
setenv ORA_NLS33 ${ORACLE_HOME}/ocommon/nls/admin/data
```

Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
setenv ORA_NLS33 ${XDK_HOME}/nlsdata
```

環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
setenv ORA_NLS33 ${ORACLE_HOME}/xdk/mesg
```

Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

```
setenv ORA_NLS33 ${XDK_HOME}/xdk/mesg
```

現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

これで、Make ファイルを使用して、デモ・コードをコンパイルおよびリンクし、UNIX プラットフォームで XDK for C++ を使用してプログラム開発を開始できます。

Windows NT での環境設定

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-%XDK_HOME%
   | - bin: executable files and dynamic libraries
   | - lib: static library files.
   |- nlsdata: Globalization Support data files (*.nlb)
   | - xdk
        | - demo: demonstration code
        | - doc: documents including release notes.
        | - include: header files.
        | - mesg: message files. (*.msb)
```

次に、XDK for C++ に付属する Widows NT ライブラリを示します。

表 3-7 XDK for C++ のライブラリ（Windows NT）

コンポーネント	ライブラリ	注意
XML Parser XSLT プロセッサ	oraxml9.lib oraxml9.dll	XML Parser for C++ バージョン 2。このパーサーには、DOM API、SAX API および XSLT API が含まれます。
XML Schema プロセッサ	oraxsd9.a oraxsd9.dll	XML Schema Processor for C++
Class Generator	oraxmlg.a oraxmlg.dll	Class Generator for C++

XDK for C++（Windows NT）は、次の表に示す Oracle の CORE ライブラリおよび Globalization Support ライブラリに依存します。

表 3-8 XDK for C++ の依存ライブラリ（Windows NT）

コンポーネント	ライブラリ	注意
CORE ライブラリ	oracore9.a oracore9.dll	Oracle CORE ライブラリ
Globalization Support ライブラリ	oranls9.a oranls9.dll	Oracle Globalization Support 共通ライブラリ
	oraunls9.a oraunls9.dll	Unicode をサポートするための Oracle Globalization Support ライブラリ

コマンドラインの使用方法

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥nlsrtl¥admin¥nlsdata
```

Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
set ORA_NLS33 =%XDK_HOME%¥nlsdata
```

環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥xdk¥mesg
```

Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

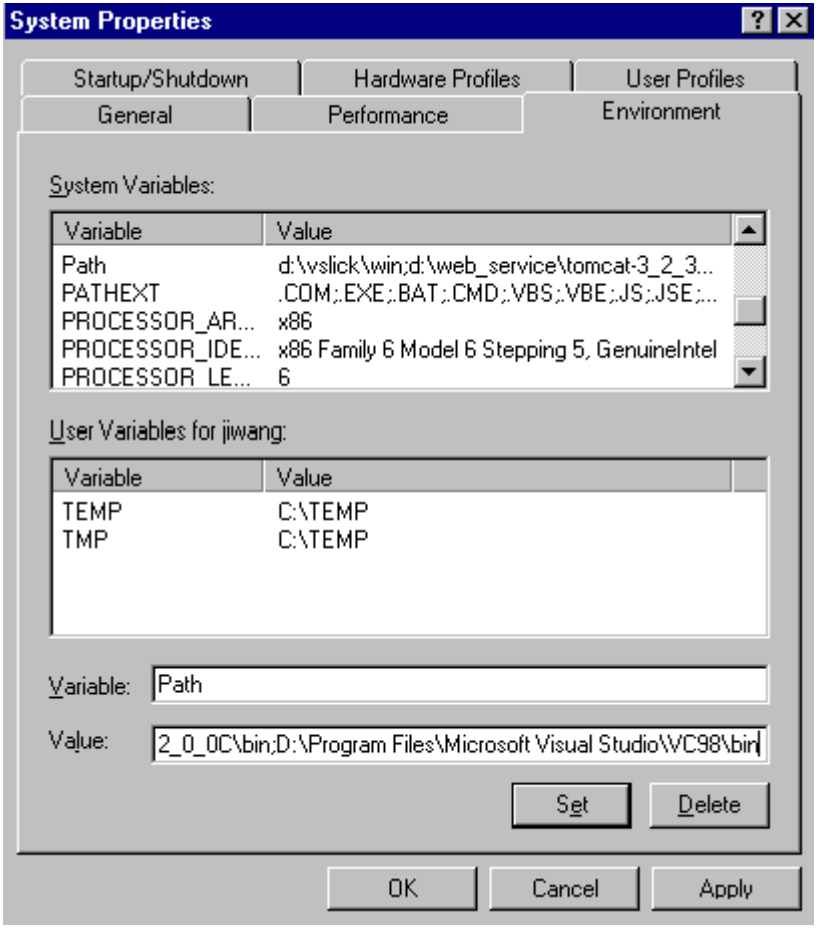
```
set ORA_NLS33 =%XDK_HOME%¥xdk¥mesg
```

現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

コマンドラインで make.bat を使用してコードをコンパイルする必要がある場合は、cl コンパイラのパスを設定します。

「スタート」メニュー >「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、図 3-8「cl コンパイラのパスの設定」に示す Path 変数に `cl.exe` のパスを入力します。

図 3-8 cl コンパイラのパスの設定



次に示すとおり、ライブラリおよびヘッダー・ファイルのパスを COMPILE および LINK コマンドに追加することによって、Make.bat ファイルを更新する必要があります。

```
...
: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 /Doratext=OraText -I.
-I..¥..¥..¥include -
ID:¥Progra~1¥Micros~1¥VC98¥Include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..¥..¥..¥..¥bin¥%filename%.exe /libpath:%ORACLE_HOME%¥lib
/libpath:D:¥Progra~1¥Micros~1¥VC98¥lib /libpath:..¥..¥..¥..¥lib %filename%.obj
oraxml9.lib oracore9.lib oranls9.lib oraunls9.lib user32.lib kernel32.lib msvcrt.lib
ADVAPI32.lib oldnames.lib winmm.lib

:EOF
...
```

ここで、各部分は次の内容を表します。

D:¥Progra~1¥Micros~1¥VC98¥Include: ヘッダー・ファイルのパス
D:¥Progra~1¥Micros~1¥VC98¥lib: ライブラリ・ファイルのパス

これで、XDK for C++ を使用した開発を開始できます。

Visual C++ での XDK for C++ の使用

環境変数 ORA_NLS33 が、グローバリゼーション・サポート・データ・ファイルの場所を指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥nlsrtl¥admin¥nlsdata
```

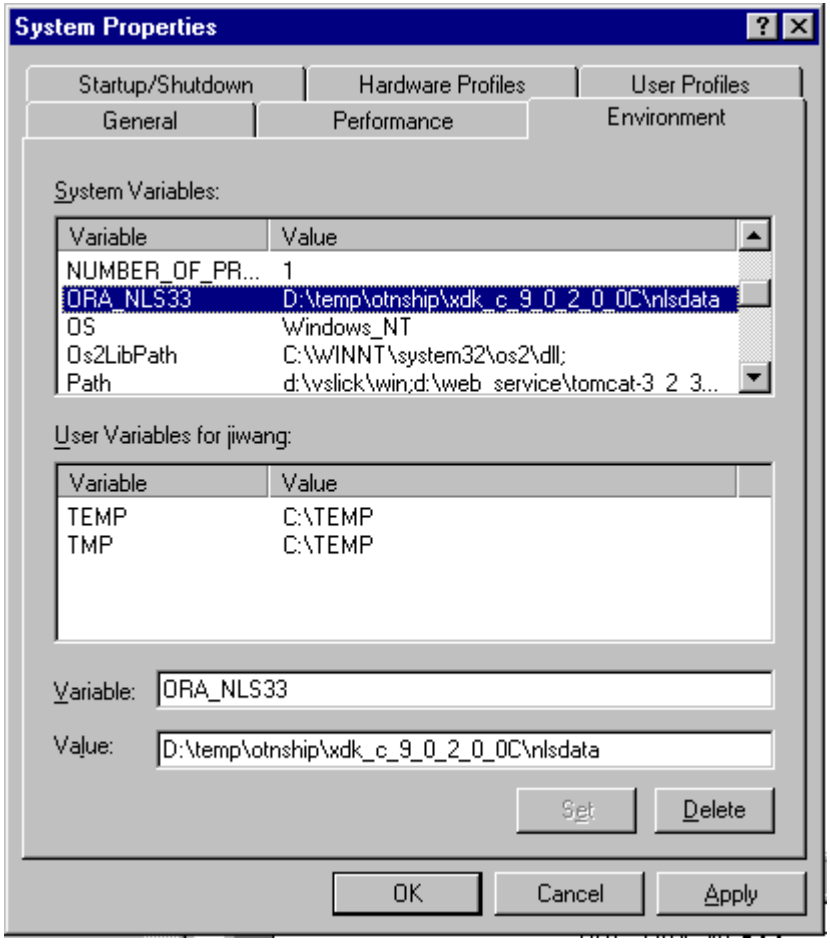
Oracle データベースをインストールしない場合は、次のとおり設定することによって、XDK リリースに付属するグローバリゼーション・サポート・データ・ファイルを使用できます。

```
set ORA_NLS33 =%XDK_HOME%¥nlsdata
```

Visual C++ でこの環境変数を認識するには、Windows NT のシステム設定を使用して、この環境変数を定義する必要があります。

「スタート」メニュー >「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、ORA_NLS33 を入力します。

図 3-9 環境変数 ORA_NLS33 の設定



環境変数 ORA_XML_MESG が、mesg ディレクトリの絶対パスを指すように設定されていることを確認します。

Oracle データベースをインストールする場合は、この環境変数を次のとおり設定できます。

```
set ORA_NLS33 =%ORACLE_HOME%¥xdk¥mesg
```

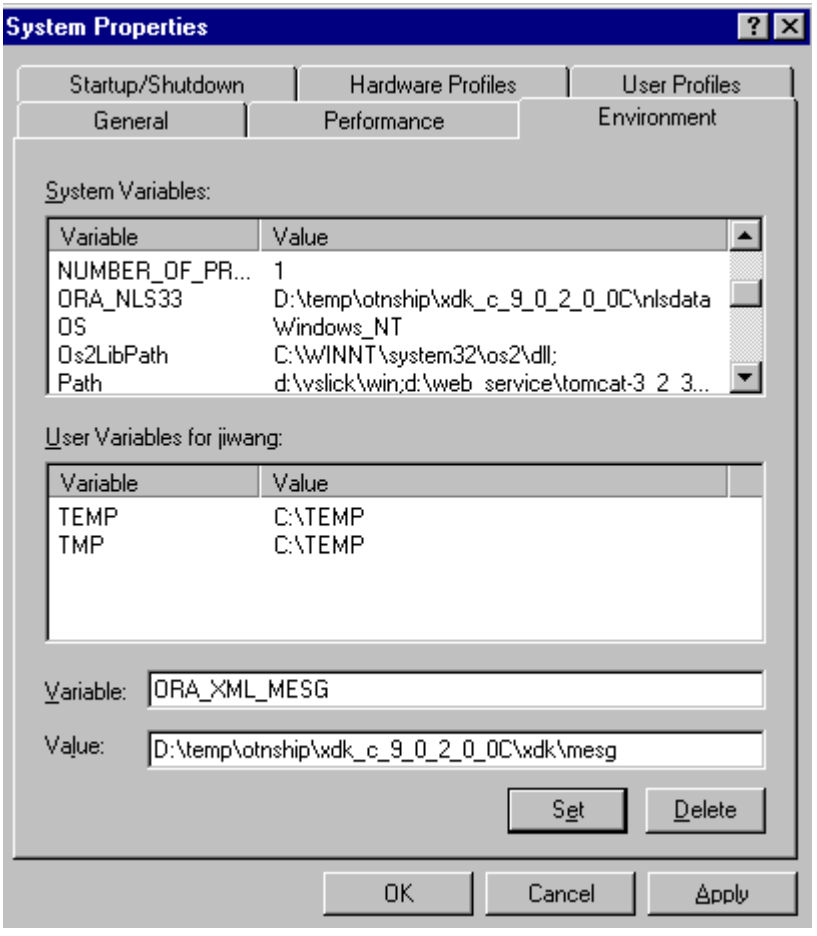
Oracle データベースをインストールしない場合は、次のとおり、この環境変数を XDK リリースに付属するエラー・メッセージ・ファイルのディレクトリに設定できます。

```
set ORA_NLS33 =%XDK_HOME%¥xdk¥mesg
```

Visual C++ でこの環境変数を使用するには、Windows NT のシステム設定を使用して、この環境変数を定義する必要があります。

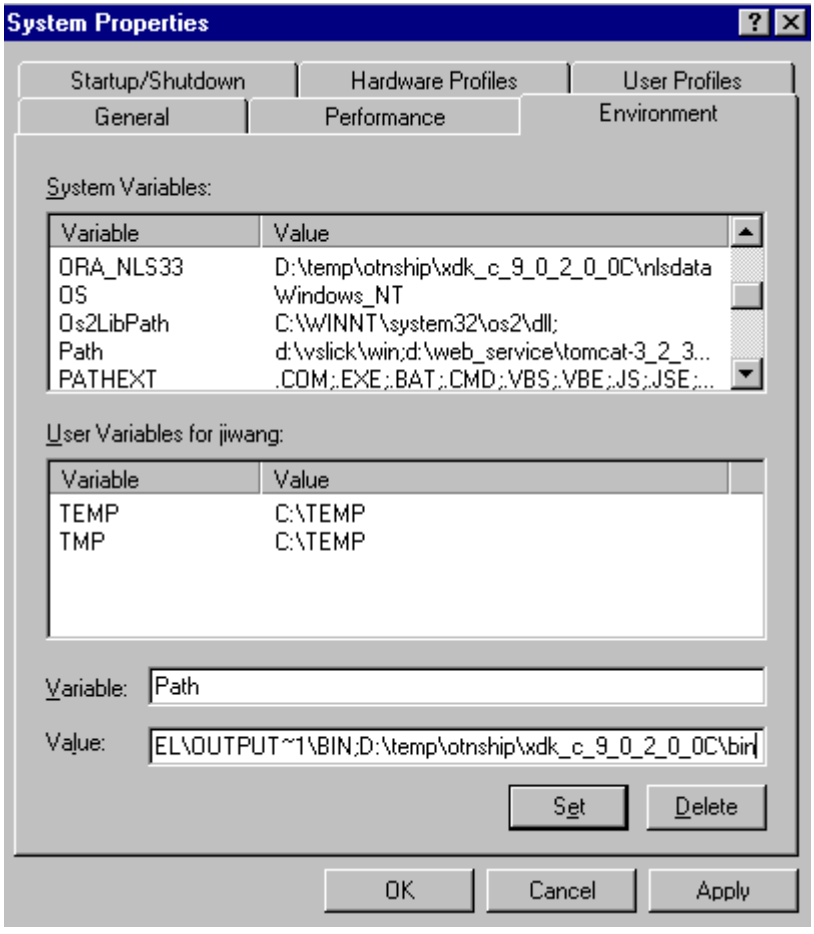
「スタート」メニュー >「設定」>「コントロール パネル」を選択します。「コントロール パネル」ウィンドウで「システム」アイコンを選択し、ダブルクリックします。「システムのプロパティ」ウィンドウが表示されます。「環境」タブを選択し、ORA_XML_MESG を入力します。

図 3-10 環境変数 ORA_XML_MESG の設定



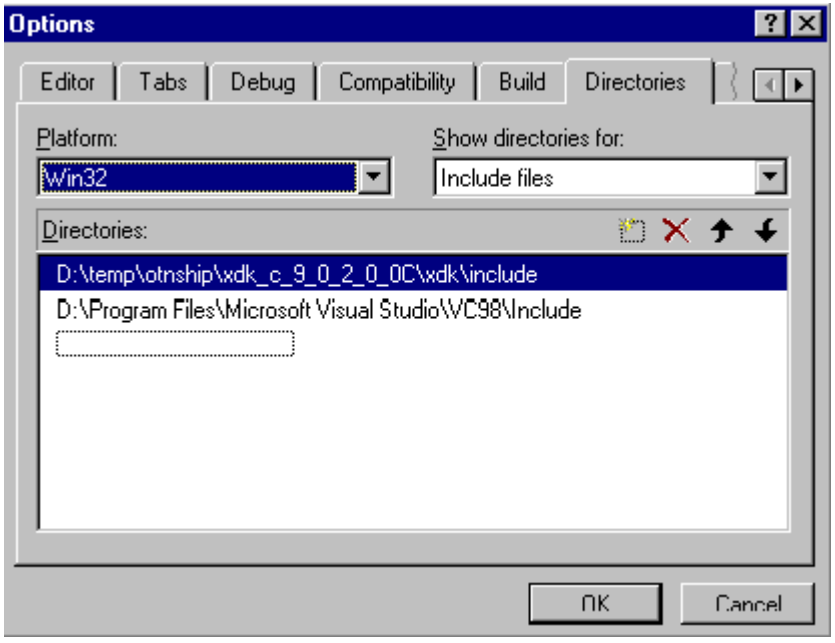
現在、すべてのメッセージ・ファイルは英語で作成されています。その他の言語用のメッセージ・ファイルは、将来のリリースで提供される予定です。

図 3-11 DLL のパスの設定



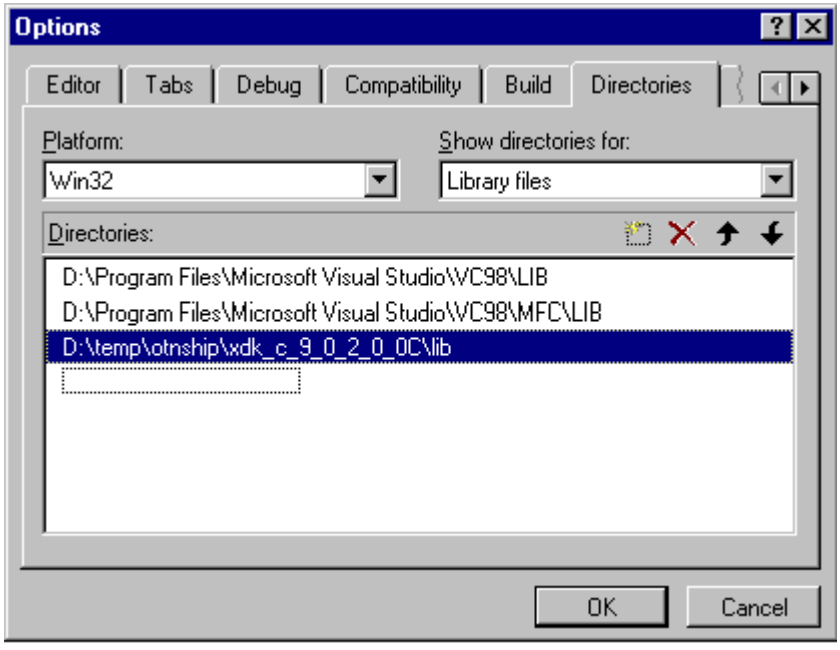
Visual C++ で作業領域をオープンし、プロジェクト用の *.c ファイルをインクルードした後で、プロジェクトのパスを設定する必要があります。「Tools」メニュー >「Options」を選択します。ウィンドウが表示されます。「Directories」タブを選択し、次の図に示すとおり、インクルード・パスを設定します。

図 3-12 Visual C++ でのインクルード・パスの設定



その後、次の図に示すとおり、ライブラリ・パスを設定します。

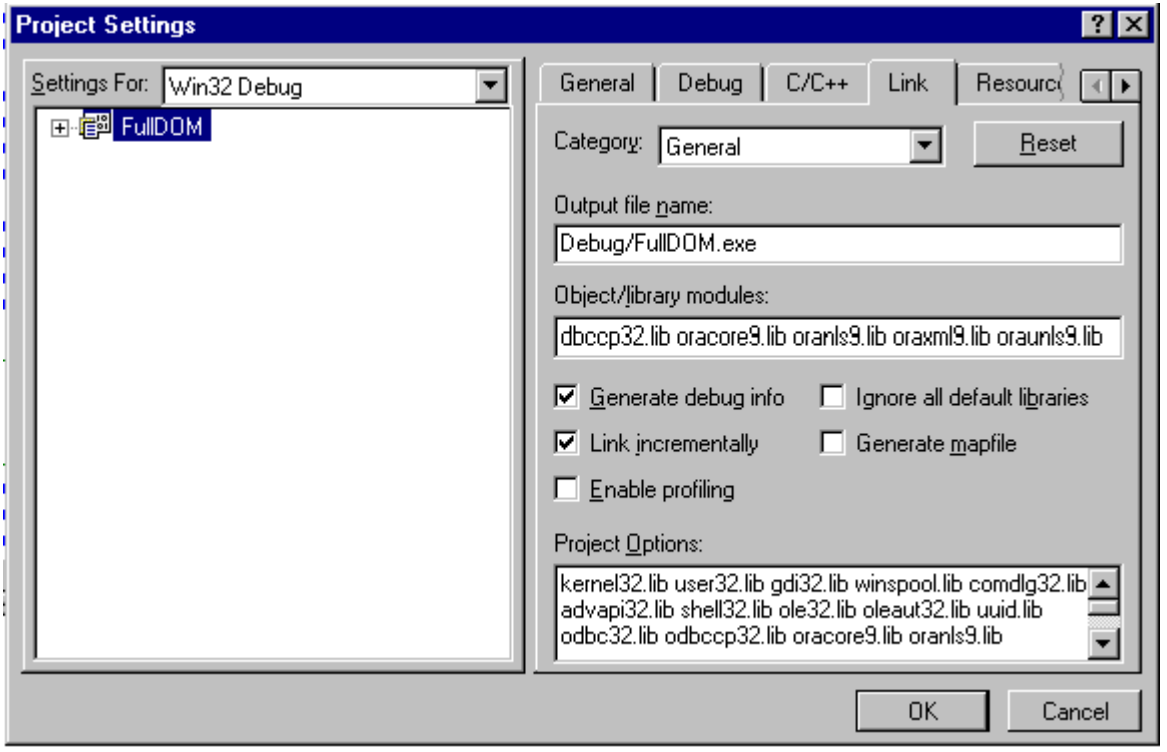
図 3-13 静的ライブラリ・パスの設定



%XDK_HOME%¥lib 内の静的ライブラリのパスを設定した後、Visual C++ のコンパイル環境でライブラリ名を設定する必要もあります。

メニュー・バーで「Project」メニュー >「Settings」を選択します。ウィンドウが表示されます。次の図に示すとおり、「Link」タブを選択し、「Object/library modules」フィールドに XDK for C++ ライブラリの名前を入力します。

図 3-14 Visual C++ プロジェクトでの静的ライブラリの設定



これで、デモ・プログラムをコンパイルおよび実行し、XDK for C++ の使用を開始できます。

> **参照：** XDK for C++ のコンポーネントの詳細は、第 16 章「XML Parser for C++」を参照してください。

XDK for PL/SQL のインストール

XDK for PL/SQL には、XML 文書の読込み、操作および変換を行うための基本コンポーネントが含まれます。Oracle XDK for PL/SQL は、次のコンポーネントで構成されます。

- XML Parser: DOM インタフェースを使用した XML 文書の解析をサポートします。
- XSLT プロセッサ : XML 文書の変換をサポートします。
- XML SQL Utility: SQL 問合せから XML 文書を生成し、その XML 文書をデータベースに挿入します。

XDK for PL/SQL の環境設定

Oracle データベースまたは Oracle9*i*AS がインストールされている場合、XDK for PL/SQL はすでにインストールされています。

XDK for PL/SQL の最新版は、OTN-J からダウンロードすることもできます。

XDK を OTN-J からダウンロードするには、次の手順を実行します。

- ブラウザで次の URL にアクセスします。

  `http://otn.oracle.co.jp/tech/xml/xdk/index.html`

- ページの左側にある「ダウンロード」アイコンをクリックします。
- OTN-J 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は、無償で登録できます）。
- ダウンロードするバージョンを選択します。
- ライセンス契約のすべての条件を承諾します。
- 適切なファイルをクリックします。
- ダウンロードしたファイルを解凍します。

XDK（XDK for PL/SQL を使用する場合）のダウンロードの詳細は、2-1 ページの「XDK for Java および XDK for JavaBeans を使用する前に」を参照してください。

XDK をインストールした後のディレクトリ構造は次のとおりです。

```
-$XDK_HOME
    | - bin: executable files and setup script/batch files.
    | - lib: library files.
    | - xdk:
        | - admin: (Administration): XSU PL/SQL API setup SQL script
              and XSL Servlet Configuration file(XSQLConfig.xml).
        | - demo: demonstration code
        | - doc: documents including release notes and javadocs.
```

次の表に、XDK for PL/SQL に付属するすべての Java ライブラリを示します。

表 3-9 XDK for PL/SQL のライブラリ

コンポーネント	ライブラリ	注意
XML Parser XSLT プロセッサ	xmlparserv2.jar	XML Parser for Java バージョン 2。このパーサーには、JAXP 1.1 API、DOM API、SAX API および XSLT API が含まれます。
	xmlmesg.jar	XML パーサー用のメッセージ・ファイル。英語以外の言語で XML パーサーを使用する場合は、CLASSPATH にこの jar ファイルを設定する必要があります。
XML Schema プロセッサ	xschema.jar	XML Schema Processor for Java
XML SQL Utility	xsu12.jar	JDK 1.2 以上用の XML SQL Utility
	xsu111.jar	JDK 1.1.8 用の XML SQL Utility
XML PL/SQL Package	xmlplsql.jar	XML PL/SQL パッケージ

次の表に、提供されている PL/SQL パッケージを示します。

表 3-10 XDK for PL/SQL のパッケージ

PL/SQL ライブラリ	パッケージ名	注意
XML Parser	xmlparser	XML パーサー
	xmldom	XML 用の DOM API
XSLT プロセッサ	xslprocessor	XSLT プロセッサ
XML SQL Utility	DBMS_XMLQuery	XML SQL Utility の PL/SQL パッケージは、Java クラス（OracleXMLQuery）の機能に類似しています。これは、SQL 問合せから XML を生成するために使用されます。
	DBMS_XMLSave	XML SQL Utility の PL/SQL パッケージは、Java クラス（OracleXMLSave）の機能に類似しています。これは、XML をデータベースに格納するために使用されます。

データベースへの XDK for PL/SQL のインストール

XDK for PL/SQL パッケージをデータベースにインストールする前に、パッケージおよび関連する Java ライブラリの状態を確認する必要があります。

PL/SQL パッケージの状態の確認

次のコマンドを使用して、現行のデータベース・スキーマに PL/SQL パッケージが含まれているかどうかを確認できます。

```
SQL*PLUS> desc package_name
```

次に例を示します。

```
SQL*PLUS> desc xmldom
```

パッケージの内容が表示された場合、そのパッケージはスキーマで使用可能な状態であり、残りのすべてのインストール手順をスキップできます。

一方、次のエラー・メッセージが表示されたとします。

```
SQL> desc xmldom
ERROR:
ORA-04043: オブジェクト "SYS"."XMLDOM" は存在しません。
```

これは、XDK for PL/SQL パッケージがデータベース・スキーマで定義されていないことを意味します。関連する Java ライブラリの状態を確認する必要があります。

Java ライブラリの状態の確認

xmlparserv2.jar、xmlplsql.jar、xsu12.jar（または xsu111.jar）などのライブラリをデータベースにロードする必要があります。SQL コマンドを使用すると、ライブラリに含まれるクラスごとに、特定のライブラリの状態を確認できます。

たとえば、xmlparserv2.jar の状態を確認するには、次の SQL 文を使用して、oracle.xml.parser.v2.DOMParser クラス内のクラスを確認できます。

```
SELECT SUBSTR(dbms_java.longname(object_name),1,35) AS class, status
     FROM   all_objects
     WHERE  object_type = 'JAVA CLASS'
     AND object_name = dbms_java.shortname('oracle/xml/parser/v2/DOMParser');
```

次の結果が表示されたとします。

```
CLASS                                               STATUS
--------------------------------------------------------
oracle/xml/parser/v2/DOMParser                       INVALID
```

この場合、次のコマンドを実行します。

```
ALTER JAVA CLASS _oracle/xml/parser/v2/DOMParser Resolve
```

この確認手順で SQL*Plus メッセージ「レコードが選択されませんでした。」が生成された場合は、3-33 ページの「XDK for PL/SQL のロード」に示す XDKLOAD ユーティリティを使用する必要があります。

前述の結果が表示された場合でも、状態が VALID であるときは、Oracle XML Parser for Java がすでにインストールされており、すぐに使用できることを意味します。すべての Java ライブラリがすでにデータベースにロードされている場合は、SQL スクリプトを実行して、PL/SQL パッケージを定義できます。

パッケージの他に、パブリック・シノニムを作成する必要がある SYS ユーザーの場合は、次のスクリプトを実行します。

XML Parser for PL/SQL の場合 :

```
$XDK_HOME/xdk/admin/xmlpkg.sql
$XDK_HOME/xdk/admin/xmlsyn.sql
```

XSU の場合 :

```
$XDK_HOME/xdk/admin/xsupkg.sql
$XDK_HOME/xdk/admin/xsusyn.sql
```

他のすべてのユーザーの場合は、次のスクリプトを実行します。

XML Parser for PL/SQL の場合 :

```
$XDK_HOME/xdk/admin/xmlpkg.sql
```

XSU の場合 :

```
$XDK_HOME/xdk/admin/xsupkg.sql
```

有効なライブラリが存在しない場合は、loadjava ユーティリティを次のとおり直接使用することによって、パッケージをロードできます。

```
loadjava -resolve -verbose -user  xdktemp/xdktemp xmlparserv2.jar
```

XDK for PL/SQL のロード

loadjava ユーティリティを使用して Java ライブラリをデータベース・スキーマにロードする前に、JVM を適切にインストールしている必要があります。LOADJAVA ユーティリティを実行する前に、INITJVM.SQL および INITDBJ.SQL スクリプトを実行して、Java 環境を初期化する必要があります。通常、これらのスクリプトは ORACLE_HOME ディレクトリの $ORACLE_HOME/javavm/install サブディレクトリにあります。

xdkload の使用

XDK for PL/SQL パッケージをデータベース・スキーマにロードするには、XDK が提供するスクリプトまたはバッチ・ファイルを使用します。

UNIX の場合 :

```
$XDK_HOME/bin/xdkload
```

Windows の場合 :

```
%XDK_HOME%¥bin¥xdkload.bat
```

xdkload コマンドの構文は次のとおりです。

```
xdkload -u username/password [-s] [-noverify] [-dbver]
```

-s	ロードされた Java API のパブリック・シノニムを作成します。これは、ターゲット・ユーザーが DBA 権限を所有している場合にのみ実行されます。
-noverify	古いバージョンのデータベースにロードし、不明なメソッドによるエラーが発生した場合（たとえば、XSU バージョン 9.0.1.0.0 を Oracle8*i* リリース 8.1.7 にロードしている場合）に使用します。
-dbver	XDK をロードするデータベースのバージョンを指定します。これは、Oracle のバージョンが XDK のバージョンより古い場合に指定する必要があります。このオプションは、-noverify オプションも設定します。

次に例を示します。

```
xdkload -u "system/manager" -s -dbver "816"
```

この例では、xdkload を使用して、XDK for PL/SQL パッケージをシステム・ユーザーにロードしています。

xdkload を使用する前に、xmlparserv2.jar、xmlxsql.jar、xsu12.jar（xsu111.jar）などのライブラリのいずれかが、すでにデータベースにロードされているかどうかを確認する必要があります。ロードされている場合は、xdkload を使用する前に、それらのライブラリを次のとおり削除する必要があります。

```
dropjava -verbose -user xdktemp/xdktemp  xmlparserv2.jar xschema.jar
```

また、XDK が提供する次のスクリプトまたはバッチ・ファイルを使用して、環境変数を設定する必要があります。

UNIX の場合 :

```
$XDK_HOME/bin/env.csh
```

Windows の場合 :

```
%XDK_HOME%¥bin¥env.bat
```

この環境設定の詳細は、2-1 ページの「XDK for Java および XDK for JavaBeans を使用する前に」を参照してください。

xdkload を実行するために使用したターゲット・ユーザー名が DBA 権限を所有している場合、xdkload スクリプトまたはバッチ・ファイルを実行すると、すべてのユーザーが XDK for PL/SQL パッケージを使用できるようになり、PL/SQL パッケージのパブリック・シノニムも作成されます。DBA 権限を所有していない場合は、ターゲット・ユーザーのみが XDK for PL/SQL パッケージを使用できます。

> **参照 :** XDK for PL/SQL のコンポーネントの詳細は、第 20 章「XML Parser for PL/SQL」を参照してください。

第 II 部

XDK for Java

第 II 部では、XDK for Java の入手方法および使用方法を説明します。

- 第 4 章「XML Parser for Java」
- 第 5 章「XSLT Processor for Java」
- 第 6 章「XML Schema Processor for Java」
- 第 7 章「XML Class Generator for Java」
- 第 8 章「XML SQL Utility（XSU）」
- 第 9 章「XSQL Pages パブリッシング・フレームワーク」
- 第 10 章「XDK JavaBeans」
- 第 11 章「XDK および SOAP の使用」
- 第 12 章「Oracle TransX Utility」

注意：

- XML SQL Utility（XSU）は、XDK for Java（および XDK for PL/SQL）の一部です。XSU の詳細は、第 8 章「XML SQL Utility（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 9 章「XSQL Pages パブリッシング・フレームワーク」を参照してください。

4

XML Parser for Java

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 パーサーは単独の 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（あるいは DTD または XML Schema）を解析します。この章では、XML Parser for Java についてのみ説明します。その他の言語バージョンの XML パーサーについては、後半の章を参照してください。

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:** 統合されたドキュメント・オブジェクト・モデル（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 勧告および Mutation Event
  - W3C の DOM 2.0 Traversal 勧告（TreeWalker、Node Iterator および Node Filter を含む）
  - DOM レベルの XML 圧縮

  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（Simple API for XML）API で、SAX 2.0 勧告および SAX2-ext 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・ドリブン・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 プロセッサがサポートされています。

  参照： `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。
  - XSL Transformation 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 プロセッサ : 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 プロセッサ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。
    * Part 0: Primer（入門書）
    * Part 1: Structures（構造）
    * Part 2: Datatypes（データ型）
- Oracle9*i* データベースおよび Oracle9*i* Application Server で実行します。

その他にも次の機能があります。

- 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
- JAXP 1.1 のサポート

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。

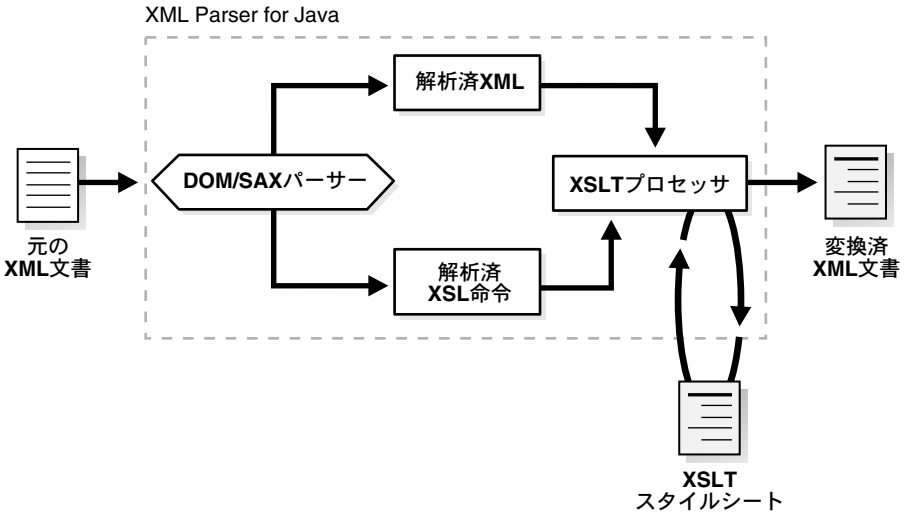
図 4-1 に、XML Parser for Java に読み込まれる XML 文書を示します。DOM または SAX パーサー・インタフェースが、XML 文書を解析します。解析済 XML は、アプリケーションに転送され、さらに処理が行われます。

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： 次の章または付録を参照してください。

- 付録 A「XDK for Java: 仕様およびクイック・リファレンス」
- 第 5 章「XSLT Processor for Java」
- 第 6 章「XML Schema Processor for Java」

図 4-1 Oracle XML Parser



DOM および SAX API については、4-7 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 4-4「XML Parser for Java: DOMParser()」
- 図 4-5「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 5-1「XSLT Processor for Java の使用」

XSL Transformation（XSLT）プロセッサ

XML パーサーには、XSLT スタイルシートを使用して XML データを変換するための XSL Transformation（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、HTML など、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 4-1 を参照してください。

参照： 詳細は、第 5 章「XSLT Processor for Java」を参照してください。

XML 名前空間のサポート

XML Parser for Java 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 Uniform Resource Identifier（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 は、ローカルの非修飾の要素型と属性名および汎用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：** 次の章またはマニュアルを参照してください。
>
> - 第 6 章「XML Schema Processor for Java」
> - 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

Oracle XML Parser の検証モード

XML Parser for Java 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 API リファレンス - XDK および Oracle XML DB』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

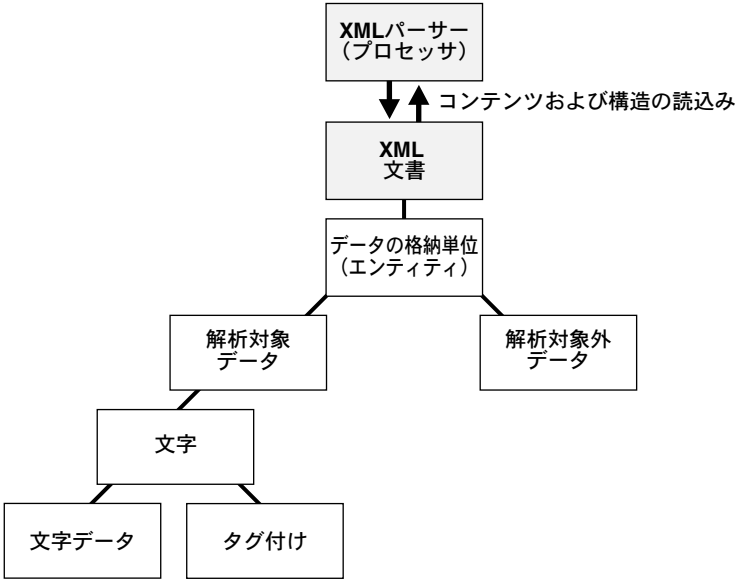
XML 文書は、エンティティというデータの記憶単位で構成され、各エンティティには解析対象または解析対象外のデータが含まれます。解析対象データは文字で構成され、中にはドキュメント内の文字データを形成したり、タグを形成するものもあります。

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 文書の読込み、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 4-2 に示します。

図 4-2 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 4-3 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。たとえば、前述の XML 文書では、DOM は図 4-3 に示すようなメモリー内のツリー構造を作成します。

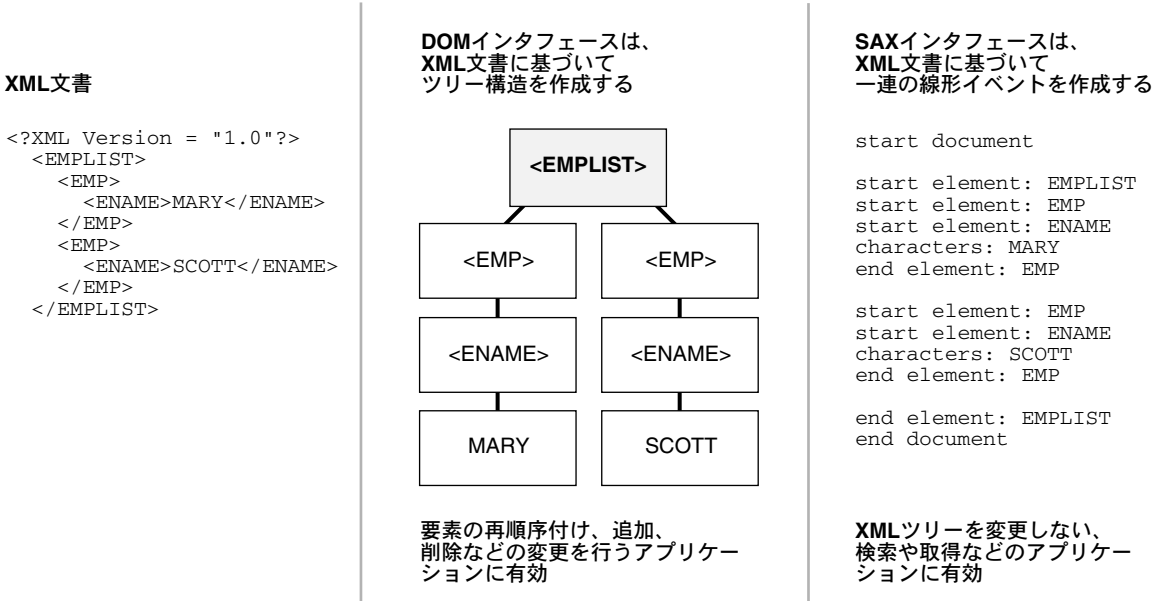
SAX: イベントベース API

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。

前述の XML 文書は、図 4-3 に示すような一連の線形のイベント群になります。

図 4-3 DOM（ツリーベース）API と SAX（イベントベース）API の比較



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 Compressor

今回のリリースでは、XML 文書のバイナリ圧縮がサポートされています。圧縮は、XML タグのトークン化に基づいています。これは、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。したがって、実行される圧縮は、入力文書のタイプに依存します。タグの数が多く、テキスト・コンテンツが少ないほど、圧縮率が向上します。

圧縮の目的は、DOM ツリーの構造情報および階層情報を失うことなく、XML 文書のサイズを小さくすることです。圧縮したストリームには、バイナリ形式から DOM ツリーを再作成するためのすべての有効な情報が含まれます。圧縮ストリームは、SAX イベントからも生成できます。DOM および SAX から生成されたバイナリ・ストリームには互換性があります。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。

圧縮機能を示すサンプル・プログラムが、デモに含まれています。

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・コンテンツが減り、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バイナリ・ラージ・オブジェクト（BLOB）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込まれると、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、再度読み込まれると、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。

XML Parser for Java のサンプルの実行

`demo/java/parser` ディレクトリには、Oracle XML Parser の使用方法を示すいくつかのサンプル XML アプリケーションが含まれています。

次に、サブディレクトリにあるサンプル Java ファイルを示します。

- XSLSample - SAX API を使用するサンプル・アプリケーション
- DOMSample - DOM API を使用するサンプル・アプリケーション
- DOMNamespace - 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
- DOM2Namespace - DOM レベル 2.0 API を使用するサンプル・アプリケーション
- DOMRangeSample - DOM Range API を使用するサンプル・アプリケーション
- EventSample - DOM Event API を使用するサンプル・アプリケーション
- NodeIteratorSample - DOM Iterator API を使用するサンプル・アプリケーション
- TreeWalkerSample - DOM TreeWalker API を使用するサンプル・アプリケーション
- SAXSample - SAX API を使用するサンプル・アプリケーション
- 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
- SAX2Namespace - SAX 2.0 を使用するサンプル・アプリケーション

- Tokenizer - XMLToken インタフェース API を使用するサンプル・アプリケーション

  Tokenizer アプリケーションは、XMLToken インタフェースを実装します。このインタフェースは、setTokenHandler() メソッドを使用して登録する必要があります。XMLToken のリクエストは、setToken() メソッドを使用して登録します。トークン化実行中、パーサーは、ドキュメントを検証せず、内部および外部のユーティリティの追加または読込みも行いません。

- DOMCompression - DOM ツリーを圧縮するサンプル・アプリケーション
- DOMDeCompression - 圧縮ストリームから DOM を再度読み込むサンプル・アプリケーション
- SAXCompression - SAX パーサーからの SAX 出力を圧縮するサンプル・アプリケーション
- SAXDeCompression - 圧縮ストリームから SAX イベントを再生成するサンプル・アプリケーション
- JAXPExamples - JAXP 1.1 API を使用して Oracle エンジンを実行するいくつかのサンプル・アプリケーション

これらのサンプル・プログラムを実行するには、次の手順に従います。

- `make` を使用して .class ファイルを生成します。
- `xmlparserv2.jar` および現在のディレクトリを `CLASSPATH` に追加します。
- 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
  ```

- 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
  ```

- Tokenizer API のサンプル・プログラムを次のとおり実行します。

  ```
  java Tokenizer sample_xml_file token_string
  ```

- DOM ツリーを圧縮するサンプル・プログラムを次のとおり実行します。

  ```
  java DOMCompression sample.dat
  ```

  圧縮した出力が「xml.ser」というファイル内に生成されます。

- 圧縮したストリームから DOM ツリーを構築するサンプル・プログラムを次のとおり実行します。

  ```
  java DeCompression xml.ser
  ```

- SAX イベントを圧縮するサンプル・プログラムを次のとおり実行します。

  ```
  java SAXCompression sample.dat
  ```

- 圧縮したストリームから SAX イベントを再生成するサンプル・プログラムを次のとおり実行します。

  ```
  java SAXDeCompression xml.ser
  ```

- JAXP 1.1 API のサンプル・プログラムを次のとおり実行します。

  ```
  java JAXPExamples
  ```

  .xml ファイルおよび .xsl ファイルは JAXPExamples.java に含まれています。

いくつかの .xml ファイルが、テスト用に common ディレクトリに含まれています。

XSLT スタイルシート iden.xsl は、common ディレクトリに存在する次の XML ファイルの変換を確認するために使用できます。

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 4-4 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。

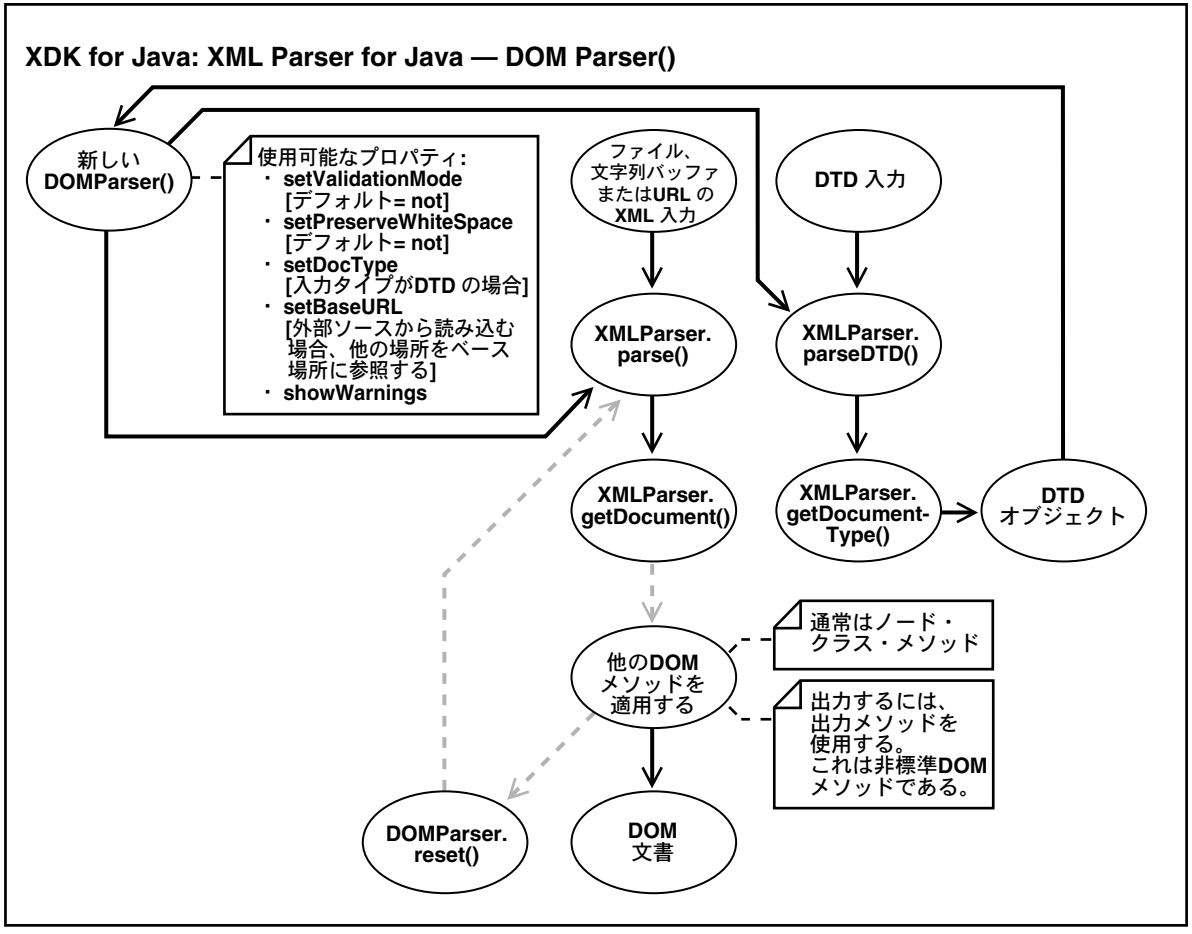
4. オプションで、次のような他の DOM メソッドを適用できます。
   * `print()`
   * `DOMNamespace()` メソッド
5. XML Parser for Java が、DOM ツリーとして XML（解析済）文書を出力します。
6. オプションで、XML Parser for Java が DOM ツリーの構築を終了した後で、`DOMParser.reset()` を使用してすべての内部データ構造を削除します。

■ **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()` クラスの使用方法については、4-17 ページの「XML Parser for Java の例 1: XML Parser for Java および DOM API の使用」を参照してください。

図 4-4 XML Parser for Java: DOMParser()



XML Parser for Java の例 1: XML Parser for Java および DOM API 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）、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 4-4 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 4-3 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 4-1 に、SAXParser() クラスのメソッドを示します。

表 4-1 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。

表 4-1 SAXParser() クラスのメソッド（続き）

メソッド	説明
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。

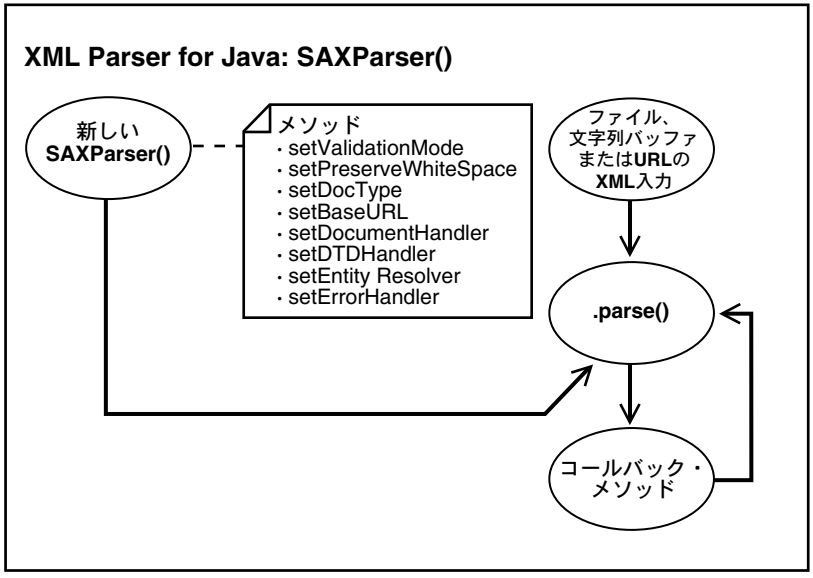
図 4-5 に、SAXParser() クラスを使用したコードを作成するための主な手順を示します。

1. 新しい `SAXParser()` クラスを宣言します。表 4-1 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 4-5 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 の例 4:（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);
    }
}
```

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。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。
- 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 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 4-2 に、oraxml のコマンドライン・オプションを示します。

表 4-2 oraxml: コマンドライン・オプション

オプション	用途
-comp ファイル名	入力 XML ファイルの圧縮
-decomp ファイル名	入力圧縮ファイルの解凍
-dtd ファイル名	DTD 検証を使用した入力ファイルの検証
-enc ファイル名	入力ファイルのエンコーディングの出力
-help	ヘルプ・メッセージの出力
-log ログ・ファイル	出力ファイルへのエラーおよびログの書込み
-novalidate ファイル名	入力ファイルが整形式かどうかの確認
-schema ファイル名	スキーマ検証を使用した入力ファイルの検証
-version	バージョン番号の出力
-warning	警告の表示

JAXP の使用

Java API for XML Processing（JAXP）によって、Java アプリケーションから SAX、DOM および XSLT プロセッサを使用できるようになります。また、特定の XML プロセッサ実装に依存しない API を使用して、アプリケーションで XML 文書を解析および変換できるようになります。

JAXP には、交換可能なレイヤーが含まれていて、このレイヤーを使用すると、プロセッサの実装をプラグインできます。JAXP API は、パーサーを交換可能にするための Thin レイヤーを提供する抽象クラスで構成された API 構造を持ちます。Oracle は、Sun 社のリファレンス実装に基づいて JAXP を実装しています。

> **参照：** その他の例については、次の URL またはディレクトリを参照してください。
>
> - `http://technet.oracle.com/tech/xml`
> - `xdk/demo/java/parser/jaxp` ディレクトリ

JAXP の例 : JAXPExamples.java

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

import java.io.*;
import java.util.*;
import java.net.URL;
import java.net.MalformedURLException;

import org.xml.sax.*;
import org.xml.sax.ext.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

public class JAXPExamples
{
        public static void main(String argv[])
        throws TransformerException, TransformerConfigurationException,
      IOException, SAXException,
      ParserConfigurationException, FileNotFoundException
        {
        try {
         URL xmlURL = createURL("jaxpone.xml");
         String xmlID = xmlURL.toString();
         URL xslURL = createURL("jaxpone.xsl");
```

```
        String xslID = xslURL.toString();
        //
        System.out.println("--- basic ---");
        basic(xmlID, xslID);
        System.out.println();
        System.out.println("--- identity ---");
        identity(xmlID);
        //
        URL generalURL = createURL("general.xml");
        String generalID = generalURL.toString();
        URL ageURL = createURL("age.xsl");
        String ageID = ageURL.toString();
        System.out.println();
        System.out.println("--- namespaceURI ---");
        namespaceURI(generalID, ageID);
        //
        System.out.println();
        System.out.println("--- templatesHandler ---");
        templatesHandler(xmlID, xslID);
        System.out.println();
        System.out.println("--- contentHandler2contentHandler ---");
        contentHandler2contentHandler(xmlID, xslID);
        System.out.println();
        System.out.println("--- contentHandler2DOM ---");
        contentHandler2DOM(xmlID, xslID);
        System.out.println();
        System.out.println("--- reader ---");
        reader(xmlID, xslID);
        System.out.println();
        System.out.println("--- xmlFilter ---");
        xmlFilter(xmlID, xslID);
        //
        URL xslURLtwo = createURL("jaxptwo.xsl");
        String xslIDtwo = xslURLtwo.toString();
        URL xslURLthree = createURL("jaxpthree.xsl");
        String xslIDthree = xslURLthree.toString();
        System.out.println();
        System.out.println("--- xmlFilterChain ---");
        xmlFilterChain(xmlID, xslID, xslIDtwo, xslIDthree);
    } catch(Exception err) {
      err.printStackTrace();
    }
}
//
public static void basic(String xmlID, String xslID)
   throws TransformerException, TransformerConfigurationException
{
```

```
   TransformerFactory tfactory = TransformerFactory.newInstance();
   Transformer transformer = tfactory.newTransformer(new StreamSource(xslID));
   StreamSource source = new StreamSource(xmlID);
   transformer.transform(source, new StreamResult(System.out));
}
//
public static void identity(String xmlID)
     throws TransformerException, TransformerConfigurationException
     {
     TransformerFactory tfactory = TransformerFactory.newInstance();
   Transformer transformer = tfactory.newTransformer();
   transformer.setOutputProperty(OutputKeys.METHOD, "html");
   transformer.setOutputProperty(OutputKeys.INDENT, "no");
   StreamSource source = new StreamSource(xmlID);
        transformer.transform(source, new StreamResult(System.out));
     }
//
public static void namespaceURI(String xmlID, String xslID)
   throws TransformerException, TransformerConfigurationException
{
   TransformerFactory tfactory = TransformerFactory.newInstance();
   Transformer transformer
      = tfactory.newTransformer(new StreamSource(xslID));
   System.out.println("default: 99");
   transformer.transform( new StreamSource(xmlID),
      new StreamResult(System.out));
   transformer.setParameter("{http://www.oracle.com/ages}age", "20");
   System.out.println();
   System.out.println("should say: 20");
   transformer.transform( new StreamSource(xmlID),
      new StreamResult(System.out));
   transformer.setParameter("{http://www.oracle.com/ages}age", "40");
   transformer.setOutputProperty(OutputKeys.METHOD, "html");
   System.out.println();
   System.out.println("should say: 40");
   transformer.transform( new StreamSource(xmlID),
      new StreamResult(System.out));
}
//
public static void templatesHandler(String xmlID, String xslID)
   throws TransformerException, TransformerConfigurationException,
   IOException, SAXException,
   ParserConfigurationException, FileNotFoundException
{
   TransformerFactory tfactory = TransformerFactory.newInstance();
   if (tfactory.getFeature(SAXTransformerFactory.FEATURE))
   {
```

```
            SAXTransformerFactory stfactory = (SAXTransformerFactory) tfactory;
            TemplatesHandler handler = stfactory.newTemplatesHandler();
            handler.setSystemId(xslID);
            // JDK 1.1.8
            Properties driver = System.getProperties();
            driver.put("org.xml.sax.driver", "oracle.xml.parser.v2.SAXParser");
            System.setProperties(driver);
            /** JDK 1.2.2
            System.setProperty("org.xml.sax.driver", "oracle.xml.parser.v2.SAXParser");
            */
            XMLReader reader = XMLReaderFactory.createXMLReader();
            reader.setContentHandler(handler);
            reader.parse(xslID);
            Templates templates = handler.getTemplates();
            Transformer transformer = templates.newTransformer();
            transformer.transform(new StreamSource(xmlID), new
StreamResult(System.out));
         }
      }
      //
      public static void reader(String xmlID, String xslID)
         throws TransformerException, TransformerConfigurationException,
         SAXException, IOException, ParserConfigurationException
      {
         TransformerFactory tfactory = TransformerFactory.newInstance();
         SAXTransformerFactory stfactory = (SAXTransformerFactory)tfactory;
         StreamSource streamSource = new StreamSource(xslID);
         XMLReader reader = stfactory.newXMLFilter(streamSource);
         ContentHandler contentHandler = new oraContentHandler();
         reader.setContentHandler(contentHandler);
         InputSource is = new InputSource(xmlID);
         reader.parse(is);
      }
      //
      public static void xmlFilter(String xmlID, String xslID)
         throws TransformerException, TransformerConfigurationException,
         SAXException, IOException, ParserConfigurationException
      {
         TransformerFactory tfactory = TransformerFactory.newInstance();
         XMLReader reader = null;
         try {
                javax.xml.parsers.SAXParserFactory factory=
                   javax.xml.parsers.SAXParserFactory.newInstance();
                factory.setNamespaceAware(true);
            javax.xml.parsers.SAXParser jaxpParser=
                   factory.newSAXParser();
                reader = jaxpParser.getXMLReader();
```

```
   } catch(javax.xml.parsers.ParserConfigurationException ex) {
     throw new org.xml.sax.SAXException(ex);
   } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
     throw new org.xml.sax.SAXException(ex1.toString());
   } catch(NoSuchMethodError ex2) {
   }
   if (reader == null)
      reader = XMLReaderFactory.createXMLReader();
   XMLFilter filter
      = ((SAXTransformerFactory) tfactory).newXMLFilter(new StreamSource(xslID));
   filter.setParent(reader);
   filter.setContentHandler(new oraContentHandler());
   filter.parse(new InputSource(xmlID));
}
//
public static void xmlFilterChain(
   String xmlID, String xslID_1,
   String xslID_2, String xslID_3)
   throws TransformerException, TransformerConfigurationException,
   SAXException, IOException
{
   TransformerFactory tfactory = TransformerFactory.newInstance();
   if (tfactory.getFeature(SAXSource.FEATURE))
   {
      SAXTransformerFactory stf = (SAXTransformerFactory)tfactory;
      XMLReader reader = null;
      try {
              javax.xml.parsers.SAXParserFactory factory =
           javax.xml.parsers.SAXParserFactory.newInstance();
         factory.setNamespaceAware(true);
              javax.xml.parsers.SAXParser jaxpParser =
                 factory.newSAXParser();
              reader = jaxpParser.getXMLReader();
      } catch(javax.xml.parsers.ParserConfigurationException ex) {
              throw new org.xml.sax.SAXException( ex );
      } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
              throw new org.xml.sax.SAXException( ex1.toString() );
      } catch(NoSuchMethodError ex2) {
      }
      if (reader == null ) reader = XMLReaderFactory.createXMLReader();
      XMLFilter filter1 = stf.newXMLFilter(new StreamSource(xslID_1));
      XMLFilter filter2 = stf.newXMLFilter(new StreamSource(xslID_2));
      XMLFilter filter3 = stf.newXMLFilter(new StreamSource(xslID_3));
      if (filter1 != null && filter2 != null && filter3 != null)
      {
         filter1.setParent(reader);
         filter2.setParent(filter1);
```

```
            filter3.setParent(filter2);
            filter3.setContentHandler(new oraContentHandler());
            filter3.parse(new InputSource(xmlID));
         }
      }
   }
   //
   public static void contentHandler2contentHandler(String xmlID, String xslID)
      throws TransformerException,
      TransformerConfigurationException,
      SAXException, IOException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();

      if (tfactory.getFeature(SAXSource.FEATURE))
      {
         SAXTransformerFactory stfactory = ((SAXTransformerFactory) tfactory);
         TransformerHandler handler
            = stfactory.newTransformerHandler(new StreamSource(xslID));
         Result result = new SAXResult(new oraContentHandler());
         handler.setResult(result);
         XMLReader reader = null;
         try {
                javax.xml.parsers.SAXParserFactory factory=
                   javax.xml.parsers.SAXParserFactory.newInstance();
            factory.setNamespaceAware(true);
                javax.xml.parsers.SAXParser jaxpParser=
                   factory.newSAXParser();
            reader=jaxpParser.getXMLReader();
         } catch( javax.xml.parsers.ParserConfigurationException ex ) {
                throw new org.xml.sax.SAXException( ex );
         } catch( javax.xml.parsers.FactoryConfigurationError ex1 ) {
                throw new org.xml.sax.SAXException( ex1.toString() );
         } catch( NoSuchMethodError ex2 ) {
         }
         if( reader == null ) reader = XMLReaderFactory.createXMLReader();
         reader.setContentHandler(handler);
         reader.setProperty("http://xml.org/sax/properties/lexical-handler",
handler);
         InputSource inputSource = new InputSource(xmlID);
         reader.parse(inputSource);
      }
   }
   //
   public static void contentHandler2DOM(String xmlID, String xslID)
      throws TransformerException, TransformerConfigurationException,
      SAXException, IOException, ParserConfigurationException
```

```
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();

      if (tfactory.getFeature(SAXSource.FEATURE)
         && tfactory.getFeature(DOMSource.FEATURE))
      {
         SAXTransformerFactory sfactory = (SAXTransformerFactory) tfactory;

         DocumentBuilderFactory dfactory
            = DocumentBuilderFactory.newInstance();
         DocumentBuilder docBuilder = dfactory.newDocumentBuilder();
         org.w3c.dom.Document outNode = docBuilder.newDocument();

         TransformerHandler handler
            = sfactory.newTransformerHandler(new StreamSource(xslID));
         handler.setResult(new DOMResult(outNode));

         XMLReader reader = null;

         try {
                javax.xml.parsers.SAXParserFactory factory =
                   javax.xml.parsers.SAXParserFactory.newInstance();
                factory.setNamespaceAware(true);
                javax.xml.parsers.SAXParser jaxpParser=
                factory.newSAXParser();
                reader = jaxpParser.getXMLReader();
         } catch(javax.xml.parsers.ParserConfigurationException ex) {
                throw new org.xml.sax.SAXException(ex);
         } catch(javax.xml.parsers.FactoryConfigurationError ex1) {
                throw new org.xml.sax.SAXException(ex1.toString());
         } catch(NoSuchMethodError ex2) {
         }
         if(reader == null ) reader = XMLReaderFactory.createXMLReader();
         reader.setContentHandler(handler);
         reader.setProperty("http://xml.org/sax/properties/lexical-handler",
handler);
         reader.parse(xmlID);
         printDOMNode(outNode);
      }
   }
   //
   private static void printDOMNode(Node node)
      throws TransformerException, TransformerConfigurationException, SAXException,
IOException,
      ParserConfigurationException
   {
      TransformerFactory tfactory = TransformerFactory.newInstance();
```

```
      Transformer serializer = tfactory.newTransformer();
      serializer.setOutputProperty(OutputKeys.METHOD, "xml");
      serializer.setOutputProperty(OutputKeys.INDENT, "no");
      serializer.transform(new DOMSource(node),
         new StreamResult(System.out));
   }
   //
   private 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;
   }
}
```

JAXP の例 : oraContentHandler.java

```
import org.xml.sax.ContentHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.Locator;

public class oraContentHandler implements ContentHandler
{
   private static final String TRADE_MARK = "Oracle 9i ";

   public void setDocumentLocator(Locator locator)
   {
      System.out.println(TRADE_MARK + "- setDocumentLocator");
   }

   public void startDocument()
      throws SAXException
   {
      System.out.println(TRADE_MARK + "- startDocument");
   }

   public void endDocument()
      throws SAXException
   {
      System.out.println(TRADE_MARK + "- endDocument");
   }

   public void startPrefixMapping(String prefix, String uri)
      throws SAXException
   {
      System.out.println(TRADE_MARK + "- startPrefixMapping: "
         + prefix + ", " + uri);
   }

   public void endPrefixMapping(String prefix)
      throws SAXException
   {
      System.out.println(TRADE_MARK + " - endPrefixMapping: "
         + prefix);
   }

   public void startElement(String namespaceURI, String localName,
      String qName, Attributes atts)
      throws SAXException
   {
      System.out.print(TRADE_MARK + "- startElement: "
```

```
            + namespaceURI + ", " + namespaceURI +
            ", " + qName);
         int n = atts.getLength();
         for(int i = 0; i < n; i++)
            System.out.print(", " + atts.getQName(i));
         System.out.println("");
      }

      public void endElement(String namespaceURI, String localName,
         String qName)
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- endElement: "
            + namespaceURI + ", " + namespaceURI
            + ", " + qName);
      }

      public void characters(char ch[], int start, int length)
         throws SAXException
      {
         String s = new String(ch, start, (length > 30) ? 30 : length);
         if(length > 30)
            System.out.println(TRADE_MARK + "- characters: \""
               + s + "\"...");
         else
            System.out.println(TRADE_MARK + "- characters: \""
               + s + "\"");
      }

      public void ignorableWhitespace(char ch[], int start, int length)
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- ignorableWhitespace");
      }

      public void processingInstruction(String target, String data)
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- processingInstruction: "
            + target + ", " + target);
      }

      public void skippedEntity(String name)
         throws SAXException
      {
         System.out.println(TRADE_MARK + "- skippedEntity: " + name);
      }
   }
```

DTD に関する FAQ

この項では、DTD についての質問および回答を示します。

XML パーサーが DTD ファイルを検出できない理由

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

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。

Oracle による DTD のキャッシュ

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));
```

XML Parser for Java が外部 DTD を認識する方法

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、Oracle9*i* 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 を読み込む場合に使用します。

jar ファイルから外部 DTD をロードする方法

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、次の手順を実行します。

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、次のコード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

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、後で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 文書と分離して DTD オブジェクトを解析する方法

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();
```

XML パーサーの大 / 小文字の区別

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、「XML-00148: 無効な要素 'xn:subjectcode'（'xn:Resource' の内容）です ...」というエラーが発生します。

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 / 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 / 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 / 小文字を区別します。これに準拠するために、Oracle XML Parser は大 / 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 セクションから埋込み XML を抽出する方法

次のコードがあるとします。

```
<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>
```

1. PAYLOAD を抽出して追加の処理を行う方法を教えてください。
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。原因を教えてください。
3. XSLT のみを使用して埋込み XML を抽出する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。

   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。

次のコードを使用します。

```
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 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コンテンツ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

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。

```
Attribute value should start with quote.
```

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、任意の値または %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 be 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

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 <![CDATA[指定で始まる XML ファイルの一部です。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 は、次のようなエンティティです。

```
<![CDATA[<!-- This is just an html comment -->]]>
```

B.xml は、次のような文書です。

```
 <?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：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 に関する FAQ

DOM API を使用してタグ付けされた要素数をカウントする方法

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ノード・リストを戻します。そのノード・リスト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。

DOM パーサーの動作方法

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てノードを作成する方法

回答 : ノードのタイプ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ノード値が 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。これによって、テキスト・ノードに値を格納できます。

XML ツリーを全検索する方法

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 ファイルから要素を抽出する方法

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

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、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。`setValidationMode()` を使用して、パーサーの検証モードを設定してください。

最初の子ノードの要素の値を検索する方法

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・ドリブンであるため、ドキュメント全体を解析する必要はありません。

DOCTYPE ノードを作成する方法

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() メソッドを使用する方法

`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またはサブツリーからコンテンツを抽出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` インタフェースを使用できます。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 を使用してデータ値を判断する方法

XML 文書の解析に SAX パーサーを使用しています。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 がメソッドをコールする方法

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使用します。イベント・ドリブンの API である SAX の基本的な説明については、http://www.megginson.com/SAX/index.html を参照してください。

DOMParser による org.xml.sax.Parser インタフェースの使用

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 を使用し、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 API を使用して新しいドキュメント・タイプ・ノードを作成する方法

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() を使用）、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方法を教えてください。NodeFactory には、DOCTYPE ノードを作成するメソッドがないことはわかっています。

回答：DOM API を使用して新しい DOCUMENT_TYPE_NODE オブジェクトを作成する方法はありません。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を問い合せる方法

XML Parser for Java を使用しています。ある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・ドリブンであるため、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

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";
```

この情報を XML 文書にする 2 つの方法は、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
```

DOM API を使用して要素タグのデータを出力する方法

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();
```

ハッシュテーブル値のペアから XML ファイルを構築する方法

キー値のペアのハッシュテーブルがありますが、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>
```

回答 :

1. ハッシュテーブル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テーブル・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

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？

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。

コードの一部分による WRONG_DOCUMENT_ERR の発生

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。

ノード値設定時に DOMException が発生する理由

次のようなエラーが戻ります。

```
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 : 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 パーサーに強制的に空白に続く文字を削除させないようにする方法

SAX パーサーを使用して添付ファイルを読み込むと、文字データが空白で始まる場合、characters() メソッドによって空白に続く文字が削除されます。

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。

検証に関する FAQ

DTD の配置規則

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 XML パーサーによって、このファイルが見つからないというメッセージが戻されます。

```
<!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。

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

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。

複数スレッドでのドキュメントのクローンの使用

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 バージョン 2 リリース 2.0.2.5 以上でサポートされています。

キャラクタ・セットに関する FAQ

特殊文字を含む ISO 8859-1 でエンコーディングされた文書を解析する方法

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、コンテンツも 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。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、Solaris オペレーティング環境 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各国語キャラクタ・ラージ・オブジェクト（NCLOB）に格納された XML を解析する方法

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, BrÃ¼ck_W</A>
</G>
```

このテキストは、次のようになります。

```
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妥当な 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() メソッドをコールする 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);
XMLDocument 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（0xc2, 0x82）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 Thin JDBC 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 文書ですべての処理を行うことができます。

サンプル XML を、Thin JDBC 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 内のグローバリゼーション・サポート

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 ツールキットを使用する動的 SQL プロシージャによって、Oracle9*i*AS およびブラウザを使用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 オペレーティング・システム、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。

アクセント付き文字を含むドキュメントを解析する方法

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 バージョン 2 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。

XML 文書内にアクセント付き文字を格納する方法

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、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込む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込む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）に設定すると解決できます。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。

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。

FAQ: 子としての XML 文書の追加

他の要素の子として XML 文書を追加する方法

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://.../pfgrfff.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。

回答 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。この問題は、DocumentFragment によって解決されるため、複数のルートに関する問題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 XML 文書のフラグメントを追加する方法

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.
```

結果ドキュメント・フラグメントが、ルート要素を 1 つのみ持つ整形式の XML 文書である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 : ドキュメント・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ドキュメント・フラグメントから 1 つのルート要素を抽出し、Element に捕捉します。

XML パーサーに関する一般的な FAQ

XML パーサーのインストール時にエラーが発生する理由

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/jdbc/driver/OracleDriver at oracle.aurora.server.tools.
```

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
```

Oracle8*i* 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
```

データベースから XML パーサーを削除する方法

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` コマンドが戻されます。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取得できます。

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 パーサーの役割

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 ファイルを HTML ファイルに変換する方法

回答 : XML を HTML にレンダリングするための XSLT 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

XML パーサーは XML Schema に対する検証を行いますか ?

回答 : 行います。

XML 文書にバイナリ・データを挿入する方法

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

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、次の Web サイトを参照してください。

`http://www.w3.org/TR/xmlschema-1/`

`http://www.w3.org/TR/xmlschema-2/`

XML Schema は、Oracle9*i* 以上でサポートされています。

XML/XSL 標準の定義へのオラクル社の参加

回答 : オラクル社は、XML/XSL に関連する XML Schema、XML Query、XSL、XLink/XPointer、XML Information Set、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 のバージョン番号を確認する方法

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 名前空間および XML Schema のサポート

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ています。

XML Parser for Java バージョン 2 以上での JDK 1.1.x の使用

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。

ページ内で結果をソートする方法

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 : 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 Oracle9*i*

回答 : XML Parser for Java は、サポートされている JVM 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 JVM 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 JVM 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

回答 : XML ファイルでは、エンコーディングを動的に設定できません。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。

文字列を解析する方法

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 文書を表示する方法

回答 : Internet Explorer 5 ブラウザを使用している場合は、直接 XML 文書を表示できます。それ以外のブラウザの場合は、Oracle XSLT プロセッサによって、XSLT 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.println() および特殊文字を使用する方法

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*/
```

XML 文書に <、>、=、'、" および & を挿入する方法

XML 文書に、不等号（より大きい）(>)、不等号（より小さい）(<)、アポストロフィ、二重引用符または等号（=）を挿入する方法を教えてください。

回答 : 等号（=）には実体参照 `&eq;`、不等号（より大きい）(>) には実体参照 `>`、および不等号（より小さい）(<) には実体参照 `<` を使用する必要があります。アポストロフィまたは一重引用符には '、二重引用符には "、およびアンパサンドには & を使用します。

タグで特殊文字を使用する方法

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 : 特殊文字を XML 名の一部として使用できます。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 仕様では、`NameChar` および `Name` は次のとおり定義されます。

```
NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar |Extender
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 `NameChars` 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。

これらの文字は読込み可能です。

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 に名前をマップする機能によって、`_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 XML を解析する方法

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 文書から文字列にデータを抽出する方法

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();
```

エスケープ出力の無効化のサポート

回答 : エスケープの出力の無効化はサポートされます。リリース 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。

特殊文字を使用した複数の XML 文書のデリミタ付け

複数の XML 文書を単一の文字列として読み込み、分離する必要があります。解決策の 1 つとして、プログラムによって生成されたいくつかの特殊文字（XML 文書内には存在しない）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、必要に応じて取得または解析できます。

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。

XML Parser for Java で実体参照を使用する方法

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。

回答 : 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または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。

DDL を挿入しないで XML 文書を分割して格納する方法

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 ?

回答 : 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。

問合せにおける XML 文書の階層の検索

回答 : 問合せにおいて、XML 文書の階層を検索することはできません。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。

XML 文書をマージする方法

回答 : 現在の DOM1 の仕様では、2 つの XML 文書をマージすることはできません。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は、次のとおり入力します。

```
$ 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>
```

この方法は、サイズが大きいファイルの場合は、2 つの表間での同等のデータベースの結合ほど効果的ではありません。ただし、XML ファイルのみを処理する場合は効果的です。

タグの値を取得する方法

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 で `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。

ユーザーに JAVASYSPRIV ロールを付与する方法

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 ファイルに外部 XML ファイルを挿入する方法

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。XML Parser for Java のバージョン 1 および 2 は、解析済外部エンティティをサポートしますか？

回答： 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 XML Parser for Java のバージョン 2 による解析の両方に、次の構文が使用できます。これは XML Parser for Java のバージョン 1 でも動作しますが、バージョン 1 より高速な最新のバージョンの XML Parser for Java を使用する必要があります。

```
File: a.xml
<?xml version="1.0" ?>
<!DOCTYPE a [<!ENTITY b SYSTEM "b.xml">]>
 <a>&b;</a>

 File: b.xml
 <ok/>
```

a.xml をブラウズまたは解析すると、次の文が戻されます。

```
<a>
  <ok/>
</a>
```

Oracle XML Parser に付属のユーティリティ（解析済出力の参照用）

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 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 で次の参照は可能ですか？それとも、次の参照を行うには他のコード例が必要ですか？

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可能なユーティリティが付属していますか？

回答： 断言はできません。解析済外部エンティティは、整形式のドキュメント・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 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

`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。これをダウンロードするには、次の Web サイトを参照してください。

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

Oracle による階層マッピングのサポート

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： Oracle Text および XML の使用方法の詳細は、次のマニュアルおよび URL 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`

2 つ目の点については、Oracle XML Parser 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・モデルを含む必要があるため、XML 1.0 準拠のパーサーはあいまいなコンテンツ・モデルは実装できません。

参照： http://www.xml.com/axml/target.html#determinism を参照してください。

XML/XSL に関する推奨書籍

XML および 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』および XML 仕様（`http://www.w3.org/XML/`）W3C
- 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（企業が XML データを交換できる、XML DTD の業界情報公開機関）作成（`http://www.ucc.ie/xml/ XML.org`）
- `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 XML Developer's Kit（XDK）

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 文書を圧縮する方法

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 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 : XDK for Java は、Oracle9*i* の圧縮メカニズムをサポートします。これは、ストリーム圧縮および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。

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および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。

次の URL にある XMLZip を参照してください。

`http://www.xmls.com/resources/xmlzip.xml?id=resources_xmlzip`

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

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 文書を生成する必要があります。この場合は、次のコードを使用します。

```
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、親子関係に対する妥当な 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>
```

5

XSLT Processor for Java

この章の内容は次のとおりです。

XML Parser for Java の使用 : XSLT プロセッサ

XSLT プロセッサは、変換する XML 文書および XML に対する変換の適用に使用する XSLT スタイルシートという 2 つの入力によって動作します。これらの 2 つの入力は、それぞれ実際には複数の入力にできます。1 つのスタイルシートを使用して複数の XML 入力を変換できます。また、複数のスタイルシートを 1 つの XML 入力にマップできます。

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 5-1 に、XSLProcessor クラスが使用する全体的なプロセスを示します。手順は次のとおりです。

- XSLProcessor オブジェクトの作成後、Java コードで次に示すメソッドを使用します。これらのメソッドは、使用可能なメソッドの一部です。
  - `removeParam()`: パラメータを削除します。
  - `resetParam()`: すべてのパラメータを削除します。
  - `setParam()`: 変換用のパラメータを設定します。
  - `setBaseURL()`: スタイルシート内の相対参照のベース URL を設定します。
  - `setEntityResolver()`: スタイルシート内の相対参照のエンティティ・リゾルバを設定します。
  - `setLocale()`: エラー・レポートのロケールを設定します。
- 次に示すファンクション `XSLProcessor.newXSLStylesheet()` への入力パラメータのいずれかを使用して、スタイルシート・オブジェクトを作成します。
  - `java.io.Reader`
  - `java.io.InputStream`
  - `XMLDocument`
  - `java.net.URL`

  これによって、スレッド・セーフで、複数の XSLT プロセッサで使用可能なスタイルシート・オブジェクトが作成されます。
- XML 入力で、入力パラメータのいずれかを使用します。

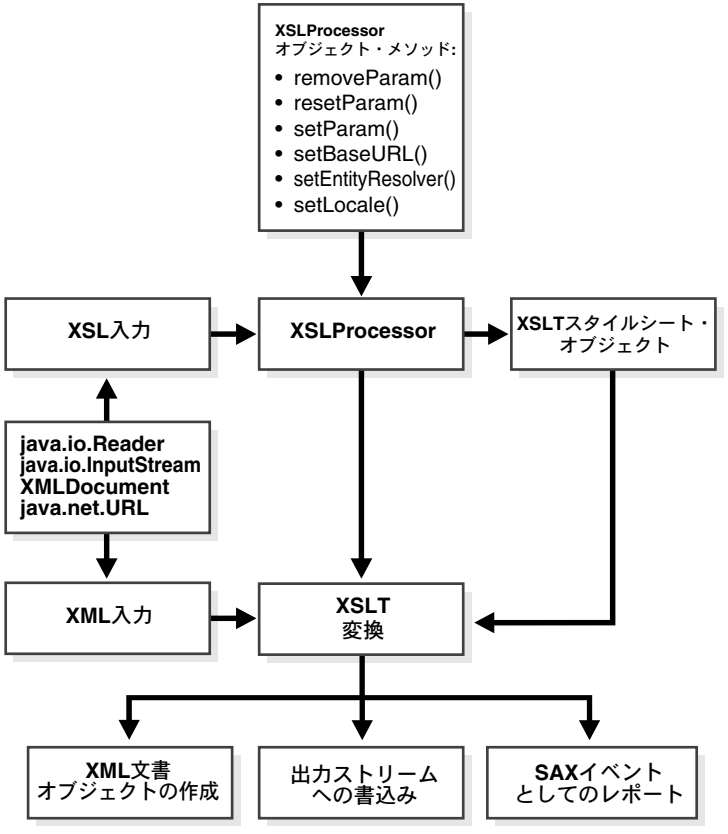
- XML 入力およびスタイルシート・オブジェクトが、XSLT プロセッサへの（それぞれ、前述の入力パラメータのいずれかを使用した）入力です。

  ```
  XSLProcessor.processXSL(xslstylesheet, xml instance)
  ```

  結果は、次のいずれかになります。

  - XML 文書オブジェクトの作成
  - 出力ストリームへの書込み
  - SAX イベントとしてのレポート

図 5-1 XSLT Processor for Java の使用



XSLT Processor for Java の例

次の例では、入力として 1 つの XML 文書および 1 つの XSLT スタイルシートを使用します。

```
public class XSLSample
{
   public static void main(String args[]) throws Exception
   {
      if (args.length < 2)
      {
         System.err.println("Usage: java XSLSample xslFile xmlFile.");
         System.exit(1);
      }

      // Create a new XSLProcessor.
      XSLProcessor processor = new XSLProcessor();

      // Register a base URL to resolve relative references
      // processor.setBaseURL(baseURL);

      // Or register an org.xml.sax.EntityResolver to resolve
      // relative references
      // processor.setEntityResolver(myEntityResolver);

      // Register an error log
      // processor.setErrorStream(new FileOutputStream("error.log"));

      // Set any global paramters to the processor
      // processor.setParam(namespace, param1, value1);
      // processor.setParam(namespace, param2, value2);

      // resetParam is for multiple XML documents with different parameters

      String xslFile = args[0];
      String xmlFile = args[1];

      // Create a XSLStylesheet
      //  The stylesheet can be created using one of following inputs:
      //
      // XMLDocument xslInput = /* using DOMParser; see below in this code */
      // URL         xslInput = new URL(xslFile);
      // Reader      xslInput = new FileReader(xslFile);

      InputStream xslInput = new FileInputStream(xslFile);

      XSLStylesheet stylesheet = processor.newXSLStylesheet(xslInput);

      // Prepare the XML instance document
```

```
        //   The XML instance can be given to the processor in one of
        // following ways:
        //
        // URL         xmlInput = new URL(xmlFile);
        // Reader      xmlInput = new FileReader(xmlFile);
        // InputStream xmlInput = new FileInputStream(xmlFile);
        // Or using DOMParser

        DOMParser parser = new DOMParser();
        parser.retainCDATASection(false);
        parser.setPreserveWhitespace(true);
        parser.parse(xmlFile);
        XMLDocument xmlInput = parser.getDocument();

        // Transform the XML instance
        //   The result of the transformation can be one of the following:
        //
        // 1. Return a XMLDocumentFragment
        // 2. Print the results to a OutputStream
        // 3. Report SAX Events to a ContentHandler

        // 1. Return a XMLDocumentFragment
        XMLDocumentFragment result;
        result = processor.processXSL(stylesheet, xmlInput);

        // Print the result to System.out
        result.print(System.out);

        // 2. Print the results to a OutputStream
        // processor.processXSL(stylesheet, xmlInput, System.out);

        // 3. Report SAX Events to a ContentHandler
        // ContentHandler cntHandler = new MyContentHandler();
        // processor.processXSL(stylesheet, xmlInput, cntHandler);

    }
}
```

参照： 4-7 ページの「SAX: イベントベース API」を参照してください。

XSLT Processor for Java: コマンドライン・インタフェース oraxsl

oraxsl - Oracle XSLT プロセッサ

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 5-1 を参照してください。

表 5-1 oraxsl: コマンドライン・オプション

オプション	用途
-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。
-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）。
-h	ヘルプ・モード（oraxsl 起動構文を出力します）。
-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）。

表 5-1 oraxsl: コマンドライン・オプション（続き）

オプション	用途
-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)。
-o 結果ディレクトリ	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・リスト	パラメータのリスト。
-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)。
-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)。
-t スレッドの数	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)。
-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)。
-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)。

XSLT プロセッサ用の XML 拡張関数

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。

XSLT プロセッサの拡張関数 : 概要

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。

static メソッドと非 static メソッドの比較

メソッドがクラスの非 static メソッドの場合、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 static メソッドの場合、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。

注意： XSL クラス・ローダーは、静的に追加された jar および wrapper.classpath によって指定された CLASSPATH のパスのみ認識します。JVM 内のリポジトリのキーワードを使用して動的に追加されたファイルは、XSLT プロセッサでは表示されません。

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
```

Oracle 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。

次のとおり入力したとします。

```
$ 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 enables all attributes
     |   that <xsl:output> enables, 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>
```

XSLT プロセッサおよび XSL に関する FAQ

この項では、XSL および XSLT プロセッサについての質問および回答を示します。

XSL で HTML エラーが発生する理由

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す。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 / 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。

XSL パーサーでの出力メソッド「html」のサポート

質問 1: 出力メソッド `html` は、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 ? `<BR>` 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 XSL 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 XSLT 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>`、`<BR>` などの整形式でないタグを XSLT 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 1: `<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 XSLT 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 `<BR>` 要素のかわりに、`<BR/>` 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 ドキュメントをリクエストします。XSLT エンジンは、整形式の XML を読み込む必要があります。

質問 2: 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 XSLT 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>
```

Internet Explorer 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 2: 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 で XSL がメタ・タグを戻さないようにする方法

<xsl output method="html" encoding="iso-8859-1" indent = “no” /> を使用しています。XSLT が HEAD 要素に <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。

ブラウザの表示エラーへの対処

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うため、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> セクションを含めないことです。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 / 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・メッセージに関する詳細情報の参照先

「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、通常はより明確です。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 の不等号（より小さい）（<）を生成する方法

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; は不等号（より大きい）（>）として生成されますが、lt; は #60; として生成されます。不等号（より小さい）（<）を生成する方法を教えてください。

回答 : 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

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。

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 : 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 の例の参照先

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 および 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 の機能のリストの参照先

Oracle XDK が使用する XSLT の機能のリストはありますか ?

回答 : XML Parser バージョン 2 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。

XSL を使用した XML 文書から他の形式への変換

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 XSL 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。

次の URI を使用すると解決します。

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

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 に関する詳細情報の参照先

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 の要点を簡単に説明しています。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 Web サイトとして、次のサイトもあります。

```
http://www.w3.org/style/XSL/
```

XSLT プロセッサでの複数の出力の生成

1 つの XML および XSL から複数の結果を作成する XSLT 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 : XML Parser バージョン 2 リリース 2.0.2.8 以上の `<ora:output>` のサポートによって処理されます。

6

XML Schema Processor for Java

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 で作成されました。XML Schema には、既存の Document Type Description（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

Document Type Definition（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。6-3 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 6-1「XML Schema の機能」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 6-1 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、定義済の基本形データ型です。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。 STRING（文字列）、BOOLEAN（ブーリアン）、FLOAT（浮動）、DECIMAL（小数）、DOUBLE（ダブル）、DURATION（時間）、DATETIME（日時）、TIME（時）、DATE（日付）、gYEARMONTH（年 / 月）、gYEAR（年）、gMonthDat（月 / 日）、gMonth、（月）、gDAY（日）、Base64Binary（BASE64 バイナリ）、HexBinary（16 進バイナリ）、ANY（任意）、URI、NOTATION（表記法）、QNAME（修飾名）。 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。	DTD は、文字列以外のデータ型をサポートしません。

表 6-1 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="gYear"> <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）、totalDigits（全桁数）、fractionDigits（小数部桁数）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。 Oracle XML Schema Processor for Java の最初のリリース以降、いくつかのファセットが変更されたことに注意してください。	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。

表 6-1 XML Schema の機能（続き）

XML Schema の機能	DTD
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、ベース型のコンテンツ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。
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、要素コンテンツ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。

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 Information set 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 A「XDK for Java: 仕様およびクイック・リファレンス」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

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。

- ストリーム（SAX）処理、一定のメモリー使用量および線形処理時間をサポートします。
- Oracle XML Parser for Java バージョン 2 に統合されています。
- W3C XML Schema 仕様の勧告候補（2000 年 10 月 24 日付）および勧告（2001 年 5 月 2 日付）を完全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 プロセッサは、Java で作成されています。これには、XML Parser for Java バージョン 2 が含まれています。

XML Schema Processor for Java のディレクトリ構造

表 6-2 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。

表 6-2 インストールした XML Schema プロセッサ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・ファイルのディレクトリ

XML Schema Processor for Java の使用方法

図 6-1 に示すとおり、Oracle XML Schema プロセッサは次の 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

スキーマ検証のモード

XML パーサーは、スキーマまたは DTD の検証で様々なモードをサポートします。`setValidationMode` メソッドを使用して、様々な検証パラメータを設定できます。スキーマの検証では、次のモードを使用できます。

- SCHEMA_VALIDATION: このモードを使用すると、スキーマ Validator がスキーマを検索および作成して、`schemaLocation` 属性および `noNamespaceSchemaLocation` 属性に基づいてインスタンス・ドキュメントの全体または一部を検証します。`XSDSample.java` のサンプル・コードを参照してください。
- SCHEMA_LAX_VALIDATION: Validator は、スキーマ定義が検出されなくなるまで、インスタンス・ドキュメントの一部またはすべての検証を試行します。定義が検出されない場合も、エラーは発生しません。`XSDLax.java` のサンプル・コードを参照してください。
- SCHEMA_STRICT_VALIDATION: Validator は、インスタンス・ドキュメント全体の検証を試行します。スキーマ定義が検出されない場合またはインスタンスが定義に準拠しない場合は、エラーが発生します。

Validator によってスキーマを作成する以外に、`XSDBuilder` を使用してスキーマを作成し、`setXMLSchema` メソッドを使用してそのスキーマを Validator に設定する方法もあります。`XSDSetSchema.java` のサンプル・コードを参照してください。`setXMLSchema` メソッドを使用することによって、検証モードが自動的に SCHEMA_STRICT_VALIDATION に設定され、`schemaLocation` 属性および `noNamespaceSchemaLocation` 属性は両方とも無視されます。検証モードを SCHEMA_LAX_VALIDATION に変更することもできます。

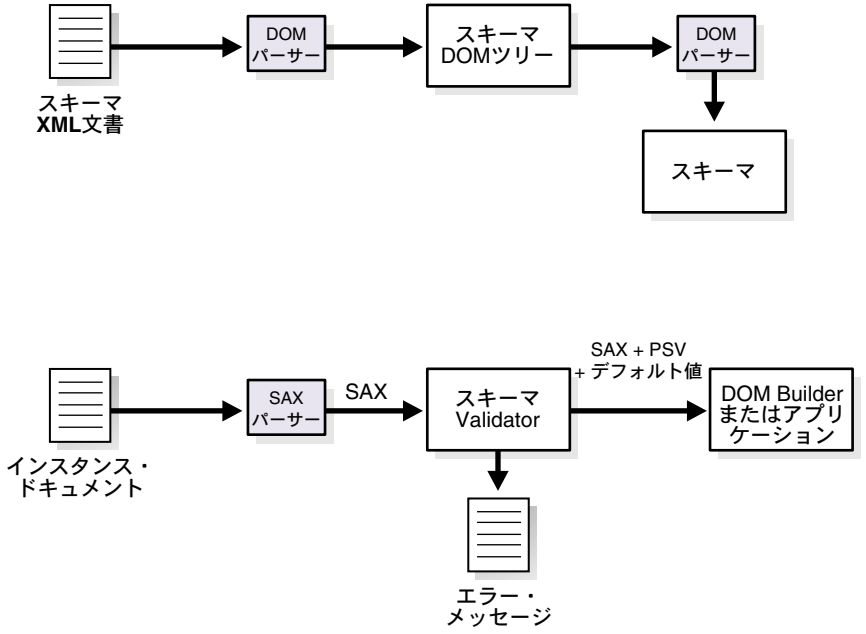
XML Schema API の使用

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。

- 6-18 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、6-20 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 6-1 XML Schema Processor for Java の使用方法



参照： 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の「XDK for Java」、「XML Schema プロセッサ」を参照してください。

XML Schema Processor for Java サンプル・プログラムを実行する方法

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。次に、README ファイルの抜粋を示します。

このディレクトリにあるサンプル Java ファイルは、次のとおりです。

`XSDSample`: 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。

`XSDSetSchema`: `schemaLocation` をオーバーライドして 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。

`XSDLax`: `XSDSetSchema` に基づきますが、LAX 検証モードを使用します。

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. プログラム 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. *.xml ファイルを使用して、サンプル・プログラムを実行します。

```
java XSDSample report.xml
java XSDSetSchema report.xsd report.xml
java XSDLax embeded_xsql.xsd embeded_xsql.xml
```

XML Schema プロセッサは、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。

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 プロセッサは、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。

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 for Java の Make ファイル

次に、ファイル Makefile を示します。

```
# Makefile for sample java files
#
# If not installed in ORACLE_HOME, set ORACLE_HOME to installation root
#
# ======================================================================

.SUFFIXES : .java .class

CLASSES = XSDSample.class XSDSetSchema.class XSDLax.class

# Change it to the appropriate separator based on the OS.
PATHSEP = :

# Assumes that the CLASSPATH contains JDK classes.
MAKE_CLASSPATH =
.$(PATHSEP)$(ORACLE_HOME)/lib/xmlparserv2.jar$(PATHSEP)$(ORACLE_HOME)/lib/xschema.ja
r$(PATHSEP)$(CLASSPATH)
```

```
.java.class:
@javac -classpath "$(MAKE_CLASSPATH)" $<

# make all class files
all: $(CLASSES)

demo: $(CLASSES)
@java -classpath "$(MAKE_CLASSPATH)" XSDSample report.xml > report.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSetSchema report.xsd report.xml > report.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSample catalogue.xml > catalogue.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSetSchema cat.xsd catalogue.xml >
catalogue.out

@java -classpath "$(MAKE_CLASSPATH)" XSDSample catalogue_e.xml > catalogue_e.out
@java -classpath "$(MAKE_CLASSPATH)" XSDSample report_e.xml > report_e.out

@java -classpath "$(MAKE_CLASSPATH)" XSDLax embeded_xsql.xsd embeded_xsql.xml>
embeded_xsql.out

clean:
@rm -f *.class
@rm -f *.out
```

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` プログラムに入力します。XML Schema プロセッサは、`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。

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.publishing.org/namespaces/Catalogue"
        elementFormDefault="qualified"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:cat="http://www.publishing.org/namespaces/Catalogue">

    <complexType name="PublicationType">
        <sequence>
            <element name="Title" type="string" minOccurs="1"
maxOccurs="unbounded"/>
            <element name="Author" type="string" minOccurs="1"
maxOccurs="unbounded"/>
            <element name="Date" type="year" minOccurs="1" maxOccurs="1"/>
        </sequence>
    </complexType>
    <element name="Publication" type="cat:PublicationType" abstract="true"/>
```

```
    <element name="Book" substitutionGroup="cat:Publication">
        <complexType>
          <complexContent>
            <extension base="cat:PublicationType">
               <sequence>
                  <element name="ISBN" type="string" minOccurs="1" maxOccurs="1"/>
                  <element name="Publisher" type="string" minOccurs="1"
maxOccurs="1"/>
               </sequence>
            </extension>
          </complexContent>
        </complexType>
    </element>
    <element name="Magazine" substitutionGroup="cat:Publication">
        <complexType>
          <complexContent>
             <restriction base="cat:PublicationType">
               <sequence>
                  <element name="Title" type="string" minOccurs="1"
maxOccurs="unbounded"/>
                  <element name="Author" type="string" minOccurs="0" maxOccurs="0"/>
                  <element name="Date" type="year" minOccurs="1" maxOccurs="1"/>
            </sequence>
           </restriction>
         </complexContent>
        </complexType>
    </element>
    <element name="Catalogue">
        <complexType>
            <sequence>
                <element ref="cat:Publication" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
</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 プロセッサは、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"?>
<Catalogue xmlns="http://www.publishing.org/namespaces/Catalogue"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation=
                     "http://www.publishing.org/namespaces/Catalogue
```

```
                    cat.xsd">
        <Magazine>
                <Title>Natural Health</Title>
                <Date>1999</Date>
        </Magazine>
        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</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 Java の例 3: catalogue_e.xml

XML Schema プロセッサは、`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"?>
<Catalogue xmlns="http://www.publishing.org/namespaces/Catalogue"
           xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
           xsi:schemaLocation=
                      "http://www.publishing.org/namespaces/Catalogue
                       cat.xsd">
        <Magazine>
                <Title>Natural Health</Title>
                <Date>1999</Date>
        </Magazine>
        <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>July 7, 1977</Date>
                <ISBN>0-440-34319-4</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>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp; Row</Publisher>
        </Book>
</Catalogue>
```

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 プロセッサは、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。

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report  report.xsd"
  period="P3M" periodEnding="1999-12-31">

 <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="455-BX" quantity="4"/>
  </zip>
 </regions>

 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </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 プロセッサは、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。

```
<schema targetNamespace="http://www.example.com/Report"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.example.com/Report"
        elementFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Report schema for Example.com
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>

 <element name="purchaseReport">
  <complexType>
   <sequence>
    <element name="regions" type="r:RegionsType">
     <keyref name="dummy2" refer="r:pNumKey">
      <selector xpath="r:zip/r:part"/>
      <field xpath="@number"/>
     </keyref>
    </element>

    <element name="parts" type="r:PartsType"/>
   </sequence>
   <attribute name="period"       type="duration"/>
   <attribute name="periodEnding" type="date"/>
  </complexType>

  <unique name="dummy1">
   <selector xpath="r:regions/r:zip"/>
   <field xpath="@code"/>
  </unique>

  <key name="pNumKey">
   <selector xpath="r:parts/r:part"/>
   <field xpath="@number"/>
  </key>
 </element>

 <complexType name="RegionsType">
  <sequence>
   <element name="zip" maxOccurs="unbounded">
```

```
    <complexType>
     <sequence>
      <element name="part" maxOccurs="unbounded">
       <complexType>
        <complexContent>
         <restriction base="anyType">
          <attribute name="number"   type="r:SKU"/>
          <attribute name="quantity" type="positiveInteger"/>
         </restriction>
        </complexContent>
       </complexType>
      </element>
     </sequence>
     <attribute name="code" type="positiveInteger"/>
    </complexType>
   </element>
  </sequence>
 </complexType>
 <simpleType name="SKU">
  <restriction base="string">
   <pattern value=" {3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
 <complexType name="PartsType">
  <sequence>
   <element name="part" maxOccurs="unbounded">
    <complexType>
     <simpleContent>
      <extension base="string">
       <attribute name="number" type="r:SKU"/>
      </extension>
     </simpleContent>
    </complexType>
   </element>
  </sequence>
 </complexType>
</schema>
```

XML Schema for Java の例 6: report_e.xml

XML Schema プロセッサは、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"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report  report.xsd"
  period="P3M" periodEnding="1999-11-31">

 <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="455-BX" quantity="4"/>
   <part number="235-JD" quantity="3"/>
  </zip>
 </regions>

 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </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.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;
   }

}
```

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 プロセッサは 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 プロセッサは 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 XSDSetSchema <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;
    }

}
```

XML Schema for Java の例 9: XSDLax.java

次に、XSDLax.java のリストを示します。

```
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 XSDLax
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 2)
      {
         System.out.println("Usage: java XSDLax <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_LAX_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;
   }

}
```

XML Schema for Java の例 10: embeded_xsql.xsd

次に、XSDLax.java への入力ファイルを示します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns = "http://xmlns.us.oracle.com/XDK/Example/XSQL/schema"
            targetNamespace = "http://xmlns.us.oracle.com/XDK/Example/XSQL/schema"
            elementFormDefault="qualified">

<xsd:element name="include-xml">
  <xsd:complexType>
     <xsd:simpleContent>
       <xsd:extension base="xsd:string">
          <xsd:attribute name="href" type="xsd:string"/>
       </xsd:extension>
     </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="XSQLBool">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="yes"/>
    <xsd:enumeration value="no"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="XSQLTagCase">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="lower"/>
    <xsd:enumeration value="upper"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="query">
  <xsd:complexType>
   <xsd:simpleContent>
    <xsd:extension base="xsd:string">
     <xsd:attribute name="bind-params" type="xsd:string"/>
     <xsd:attribute name="date-format" type="xsd:string"/>
```

```
      <xsd:attribute name="error-statement" type="XSQLBool"/>
      <xsd:attribute name="fetch-size" type="xsd:positiveInteger"/>
      <xsd:attribute name="id-attribute" type="xsd:string"/>
      <xsd:attribute name="id-attribute-column" type="xsd:string"/>
      <xsd:attribute name="include-schema" type="XSQLBool"/>
      <xsd:attribute name="max-rows" type="xsd:positiveInteger"/>
      <xsd:attribute name="null-indicator" type="XSQLBool"/>
      <xsd:attribute name="rowset-element" type="xsd:string"/>
      <xsd:attribute name="row-element" type="xsd:string"/>
      <xsd:attribute name="skip-rows" type="xsd:positiveInteger"/>
      <xsd:attribute name="tag-case" type="XSQLTagCase"/>
    </xsd:extension>
   </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

XML Schema for Java の例 11: embeded_xsql.xml

次に、XSDLax.java からの出力ファイルを示します。

```
<?xml version="1.0" ?>
<page connection="xdkdemo" xmlns:xsql="http://xmlns.us.oracle.com/XDK/Example/XSQL/
schema">
  <webpage title=" Search for XDK FAQ">
      <search>
       <xsql:include-xml href="xml/title.xml" />
      </search>
          <content>
            <question>
              <xsql:query fetch-size="50" null-indicator="no">
                     select question from xdkfaq
                     where contains(answer,'{@search}')>0
                   </xsql:query>
              </question>
              <time>
              <xsql:query tag-case="lower" max-rows="20">
                     select to_char(sysdate,'DD-MM-YYY') from dual
                   </xsql:query>
              </time>
          </content>
  </webpage>
</page>
```

7

XML Class Generator for Java

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、Oracle9*i* の XDK for Java に付属しています。Oracle XML Class Generator for Java は、`$ORACLE_HOME/xdk/java/classgen` にあります。OTN-J の Web サイト http://otn.oracle.co.jp/ からダウンロードすることもできます。

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。

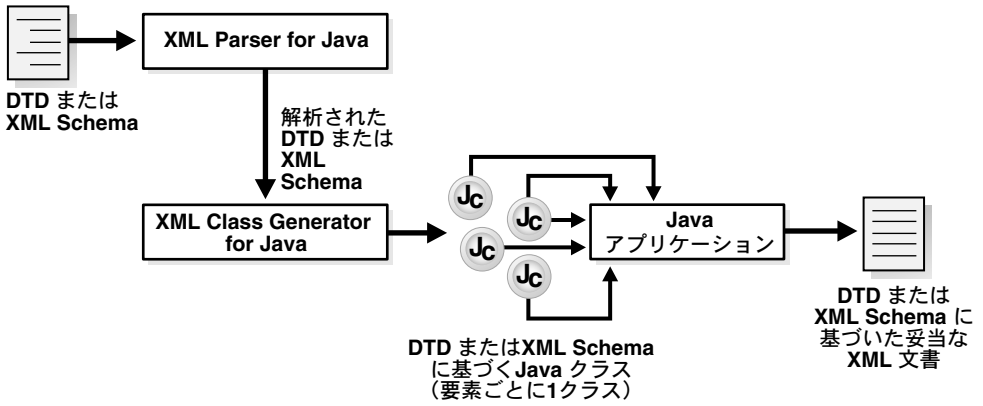
XML Class Generator for Java は、次の 2 つの Class Generator で構成されます。

- **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 7-1 に、XML Class Generator for Java の使用方法の概要を示します。

図 7-1 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。

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 7-1 に、`oracg` の引数を示します。

表 7-1 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
- help	ヘルプ・メッセージ・テキストを出力します。
- version	バージョン番号を出力します。
- dtd [-root]	入力ファイルは DTD ファイルまたは DTD に基づく XML ファイルです。
- schema	入力ファイルは Schema ファイルまたは Schema に基づく XML ファイルです。
- outputDir	Java ソースの生成先ディレクトリ名です。
- package	生成される Java クラスのパッケージ名（複数可）です。
- comment	生成される Java ソース・コードにコメントを生成します。

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。この競合を解決するために、パッケージ名に対応するディレクトリ内の types というサブディレクトリに、simpleType 要素および complexType 要素に対応するクラスが生成されます。
- 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 7-2 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。

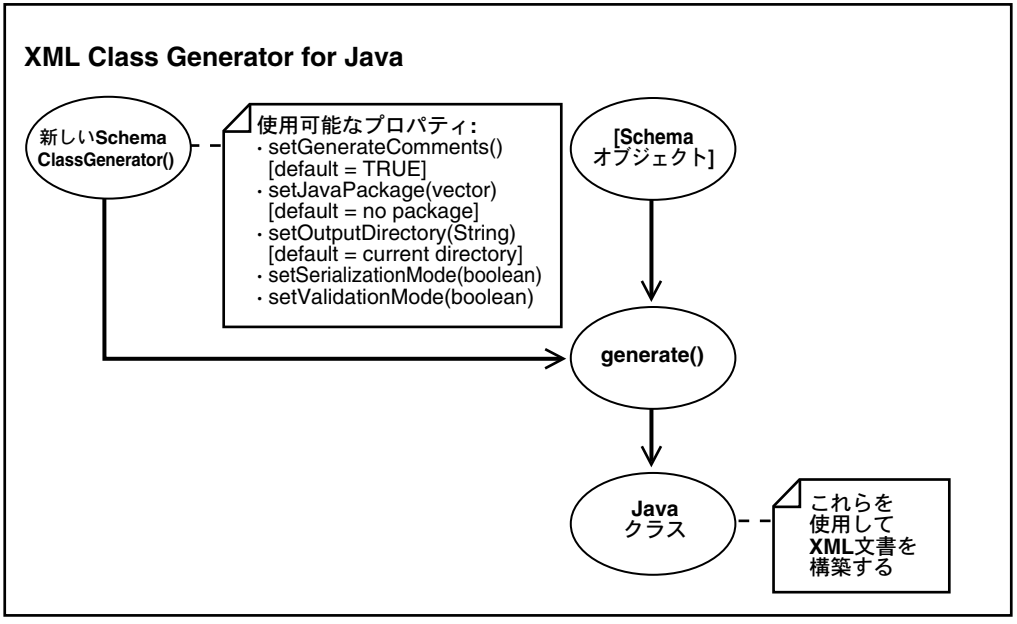
XML Schema を使用した XML Class Generator for Java の動作は次のとおりです。

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。
   - `setGenerateComments()`（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。4-16 ページの図 4-4「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。

- 7-6 ページの「最上位要素のクラスの生成」
- 7-7 ページの「最上位 ComplexType 要素のクラスの生成」
- 7-7 ページの「SimpleType 要素のクラスの生成」

図 7-2 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」を使用する要素は、要素と `complexType` の間でクラス名が競合します。
- 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。

- XML Schema オブジェクトを受け取り検証するための検証メソッドが要素内に提供されます。
- ノードを出力するための出力メソッドが要素内に提供されます。

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。

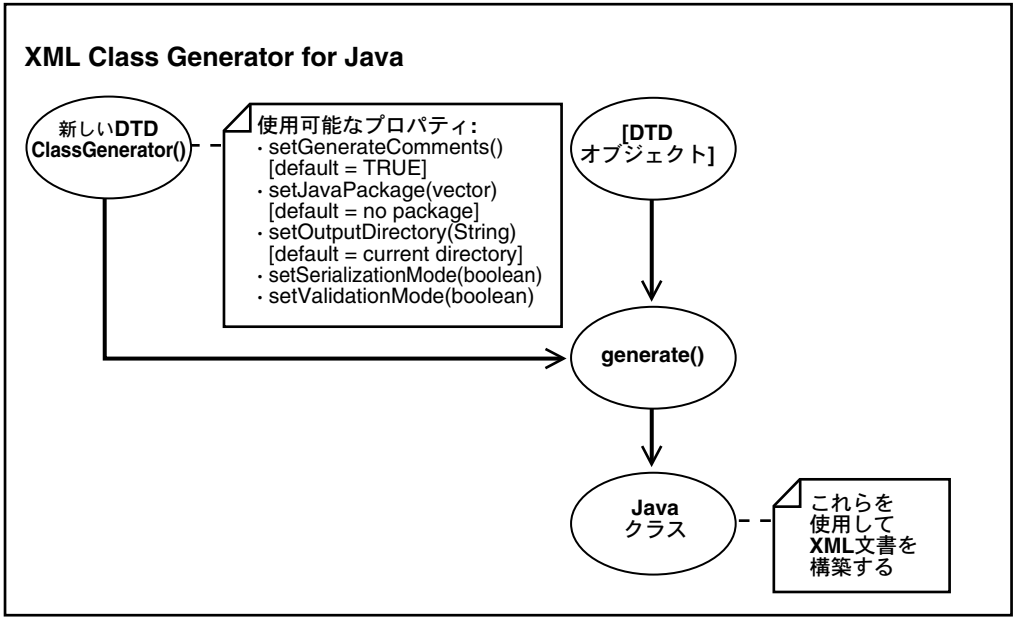
- 最上位の simpleType 要素ごとに 1 つのクラスが生成されます。
- simpleType 要素の階層は、生成されたクラス内に保持されます。simpleType 要素がベース・クラスを拡張する場合、その simpleType 要素に対応したクラスもベースの要素と一致するベース・クラスを拡張します。それ以外の場合は、simpleType 要素は CGXSDElement クラスを拡張します。
- simpleType 要素がスキーマのデータ型を拡張する場合、クラスもそのスキーマのデータ型に対応するクラスを拡張します。たとえば、ベース型が文字列の場合、そのスキーマと同等のクラスは XSDStringType 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 7-3 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。4-16 ページの図 4-4「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 7-3 XML Class Generator for Java および DTD を使用したクラスの生成



参照： 次の付録およびマニュアルを参照してください。

- 付録 A「XDK for Java: 仕様およびクイック・リファレンス」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

DTD および XML Schema を使用した XML Class Generator の使用例

表 7-2 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 7-2 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 target '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. 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 target '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。

Make ファイルを使用して 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

```
/**
 * 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: 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: 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。

- 7-19 ページの「XML Class Generator for Java の Schema の例 1b: アプリケーション - CarDealer.java」
- 7-10 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: 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。

- 7-22 ページの「XML Class Generator for Java の Schema の例 2b: BookCatalogue.java」
- 7-10 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: 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。

- 7-25 ページの「XML Class Generator for Java の Schema の例 3b: アプリケーション - TestPo.java」
- 7-10 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();
         }
      }
   }
```

XML Class Generator for Java に関する FAQ

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 for Java をインストールする方法

回答 : XML Class Generator は、XDK の一部としてパッケージ化されているため、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

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 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。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 バージョン 2 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

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 : サポートします。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。

「クラスが見つかりません」というエラーが発生する原因

XML Class Generator サンプル実行中の「クラスが見つかりません」というエラーの原因を教えてください。

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

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: Node doesn't belong to the current document
```

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、スター演算子（*）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

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

次の XML 文書を Java クラス内で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。XML SQL Utility 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 の一部として入手できます。XDK は、Oracle9*i* および Oracle9*i* Application Server 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。

8

XML SQL Utility（XSU）

この章の内容は次のとおりです。

XML SQL Utility（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。

XML SQL Utility（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得したデータを 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 XML SQL Utility 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：

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

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、Oracle9*i* 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 Oracle9*i* 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。

Oracle9*i* の XSU の新機能

Oracle9*i* 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： Oracle9*i* 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 による XMLType のサポート

Oracle9*i* リリース 2（9.2）以上では、XSU は XMLType をサポートします。XSU での XMLType の使用は、たとえば、オブジェクトまたは表内に XMLType 列が含まれている場合に有効です。

参照： XMLType が含まれている XSU の使用例については、『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』の特に XML の生成に関する章を参照してください。

XSU の依存関係およびインストール

依存関係

XSU は、次のコンポーネントによって異なります。

- 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 JDBC Drivers 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。
- Oracle XML Parser - xmlparserv2.jar。xmlparserv2.jar は、Oracle9*i* のインストールに含まれています。また、xmlparserv2.jar は、OTN-J の Web サイトからダウンロード可能な XDK for Java アーカイブの一部です。
- XSU は、xdb.jar および servlet.jar に含まれるクラスによっても異なります。これらのファイルは、Oracle9*i* のインストールに含まれます。また、OTN-J からダウンロード可能な XDK for Java アーカイブにも含まれます。

XSU のインストール

XSU は、Oracle9*i* ソフトウェアの CD に添付されています。また、OTN-J からダウンロード可能な XDK for Java の一部です。XSU は、次の 2 つのファイルで構成されています。

- $ORACLE_HOME/lib/xsu12.jar: XSU を構成する Java クラスを含むファイルです。xsu12 は、最小の JDK1.2.x および JDBC2.x が必要です。
- $ORACLE_HOME/rdbms/admin/dbmsmeta.sql: XSU の PL/SQL API を構築する SQL スクリプトです。xsu12.jar をデータベースにロードして、dbmsxsu.sql を実行します。

Oracle9*i* 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、XSU をデータベースにもロードします。最初の Oracle インストール時に XSU をインストールしない場合は、後でインストールできます。Oracle Installer を使用して、XSU およびその固有コンポーネントをインストールできます。また、OTN-J から最新の XDK for Java をダウンロードできます。

XSU をデータベースにロードするには、XSU のインストール方法に応じた次の手順のいずれかを実行する必要があります。

- Oracle Installer によるインストール：`ORACLE_HOME` ディレクトリに移動してから `rdbms/admin` に移動します。次に `initxml.sql` を実行します。
- OTN-J からのダウンロードによるインストール：ダウンロードし、拡張された XDK アーカイブの `bin` ディレクトリに移動します。次に `xdkload` スクリプトを実行します。Windows をご使用の場合は、`xdkload.bat` を実行します。

XML SQL Utility およびより大きな構図

XML SQL Utility（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 XML SQL Utility

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。

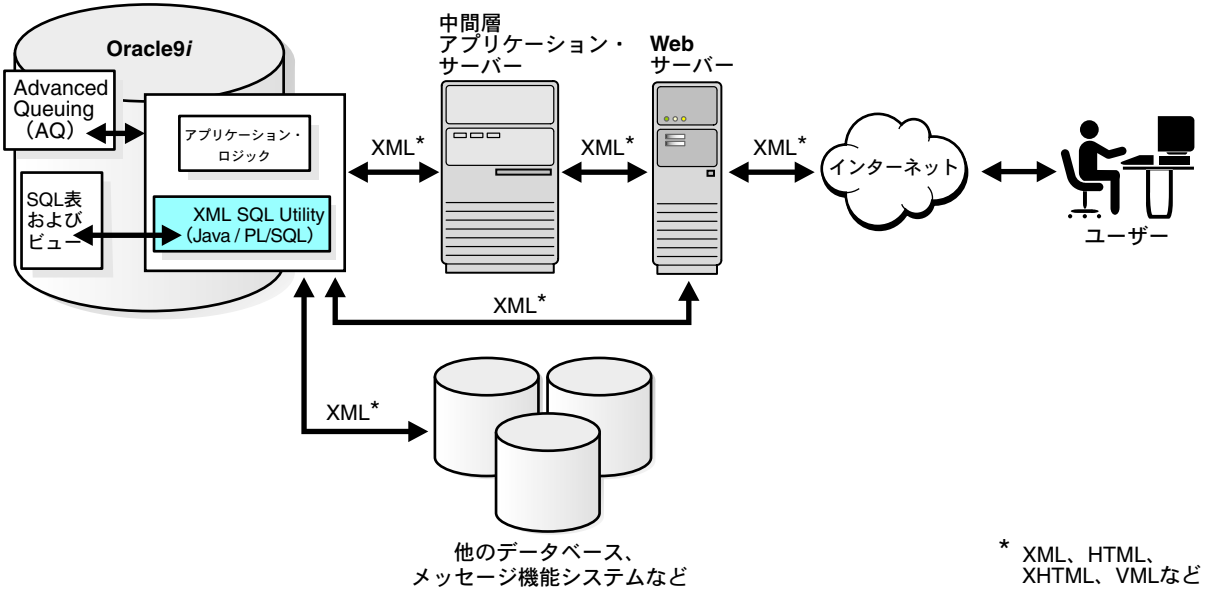
- データベース内で実行して XSU の Java API に直接アクセスする新しい Java アプリケーションを作成します。
- PL/SQL API を介して XSU にアクセスする PL/SQL アプリケーションを作成します。
- SQL を介して直接 XSU の機能にアクセスします。

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 8-1 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 サーバー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 8-1 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 8-1 データベース内での XML SQL Utility の実行



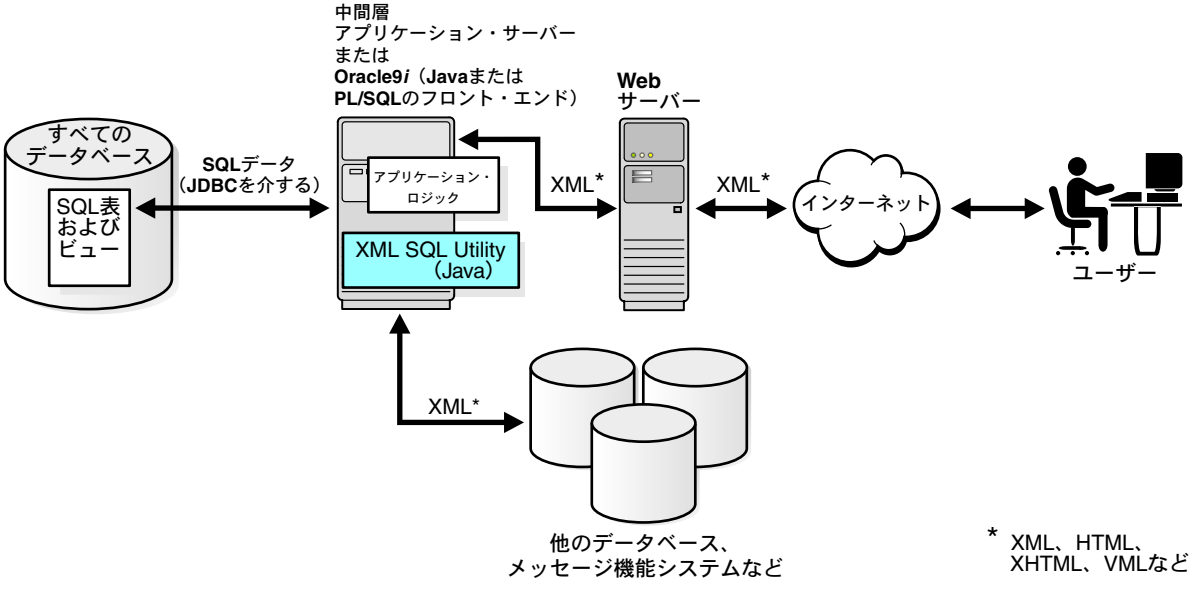
中間層内での XML SQL Utility

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、Oracle9*i* 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 8-2 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 8-2 中間層内での XML SQL Utility の実行



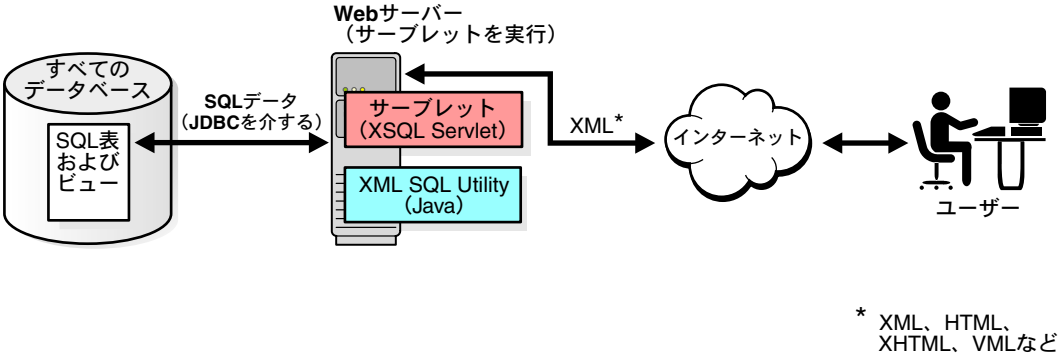
Web サーバー内での XML SQL Utility

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

図 8-3 Web サーバー内での XML SQL Utility の実行



クライアント層内での XML SQL Utility

XML SQL Utility 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、XML SQL Utility 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得したデータを 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コンテンツを持つ、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 EmployeeType 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捕捉多重集合構造を使用して、フラットなスキーマから取得した 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

XML SQL Utility 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 / 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：** 8-9 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。
- **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 / 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。

XML SQL Utility 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

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得されます。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。8-9 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得します。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：** 8-9 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、8-34 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。8-37 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')
```

8-39 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。

> **注意：** Oracle9*i* 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、次のコマンドをコールします。

```
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）
- XMLType 用の jar ファイル

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 8-1 に、OracleXML getXML オプションを示します。

表 8-1 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。
-withSchema	XML 文書とともにスキーマも生成するように 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。

表 8-1 XSU の OracleXML getXML オプション（続き）

getXML オプション	説明
-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。
-raiseNoRowsException	行が戻されなかった場合に例外を発生させるように XSU に指示します。
-maxRows "*maximum number of rows*"	XML に変換するために取得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。
-useTypeForCollElemTag	coll-elem タグに型名を使用します（デフォルトでは、XSU は *column-name_item* を使用します）。
-setXSLTRef *"URI"*	XSLT 外部実体参照を設定します。
-useLowerCase | useUpperCase	それぞれ小文字または大文字のタグ名を生成します。デフォルトでは、大 / 小文字は、タグ名の生成元の SQL オブジェクト名に一致します。
-withEscaping	SQL オブジェクト名では有効であっても、XML タグでは無効な文字が存在します。このオプションは、このような文字を検出した場合、例外を発生させるのではなく、エスケープすることを意味します。
-raiseException	デフォルトでは、XSU がエラーを捕捉し、エラー XML 文書を生成します。このオプションでは、発生した Java 例外を XSU が実際に発生させるように動作を変更できます。

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 8-2 に、putXML オプションを示します。

表 8-2 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。

表 8-2 XSU の OracleXML putXML オプション（続き）

putXML オプション	説明
`-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 / 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指定します。
`-tableName "table"`	値を挿入する表の名前です。
`-withEscaping`	文書の生成時に SQL から XML への名前のエスケープを使用すると、逆マッピングが有効になります。
`-setXSLT "URI"`	挿入前に XML 文書に適用する XSLT です。
`-setXSLTRef "URI"`	XSLT 外部実体参照を設定します。

XSU の Java API

XML SQL Utility の Java API は、次の 2 つのクラスで構成されます。

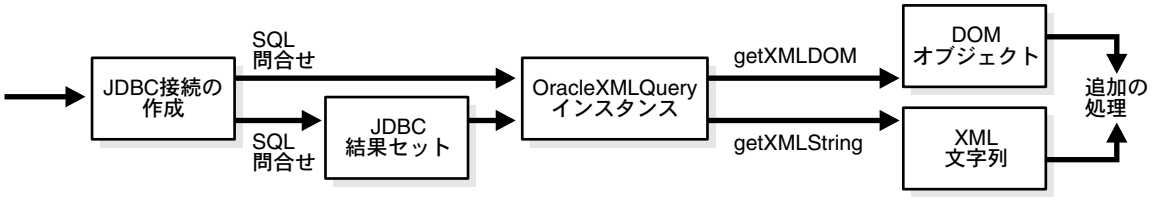
- XML 生成用の XSU API: `oracle.xml.sql.query.OracleXMLQuery`
- XML の保存、挿入、更新および削除用の XSU API: `oracle.xml.sql.dml.OracleXMLSave`

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 8-4 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。

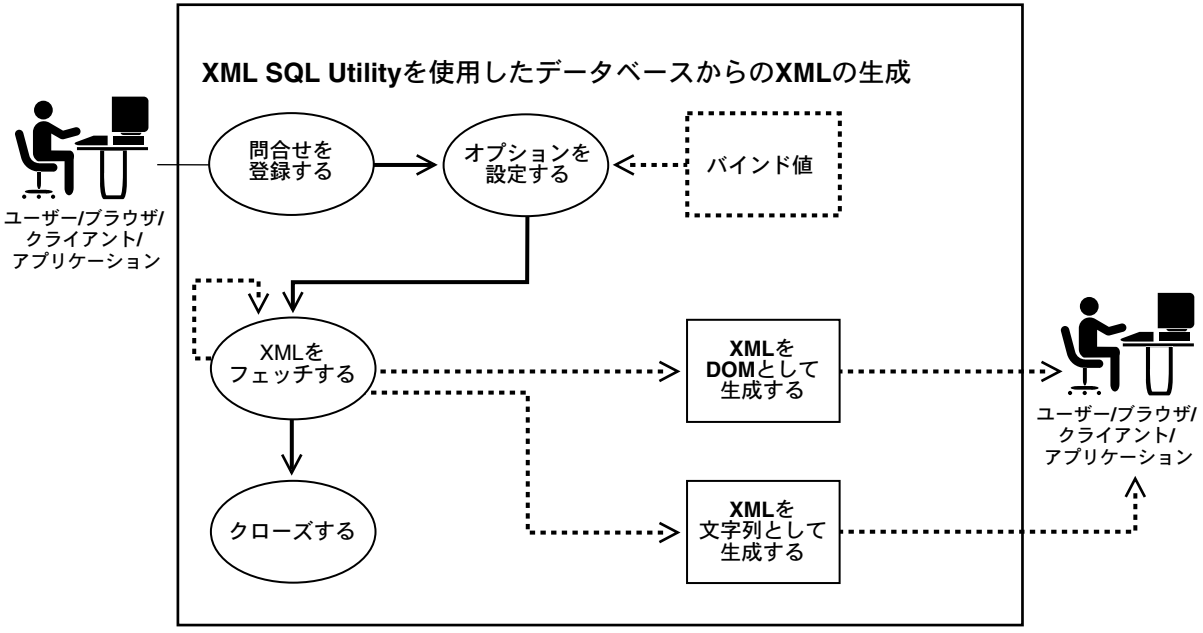
図 8-4 XML SQL Utility for Java を使用した XML の生成 : 基本的な処理の流れ



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 8-5 を参照してください。

図 8-5 XML SQL Utility 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、Thin JDBC 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、Thin JDBC 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: 表 emp 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 文書を直接リクエストします。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、XML SQL Utility（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、次に追加の結果をリクエストされたときに、前回の生成が終了した所から生成を開始します。8-28 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 : 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 : 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 oracle.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.*;
import oracle.xml.sql.query.*;
import oracle.xml.sql.dataset.*;
import oracle.xml.sql.docgen.*;

import java.sql.*;
import java.io.*;

public class b
{
  public static void main(String[] args) throws Exception
  {

@    DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

    Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    Statement stmt =
```

```
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                          ResultSet.CONCUR_READ_ONLY);

    String sCmd = "SELECT ENAME FROM SCOTT.EMP";
    ResultSet rs = stmt.executeQuery(sCmd);

    OracleXMLQuery xmlQry = new OracleXMLQuery(conn, rs);
    xmlQry.keepObjectOpen(true);
    //xmlQry.setRowIdAttrName("");
    xmlQry.setRowsetTag("ROWSET");
    xmlQry.setRowTag("ROW");
    xmlQry.setMaxRows(20);

    //rs.beforeFirst();
    String sXML = xmlQry.getXMLString();
    System.out.println(sXML);
  }
}
```

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得しました。ただし Web では、次のページの結果のみでなく、前のページも取得する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 active.
      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発生

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発生させる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` を発生させます。これはランタイムの例外であり、必要がないときは捕捉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 文字列を確認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();
     }
   }
}
```

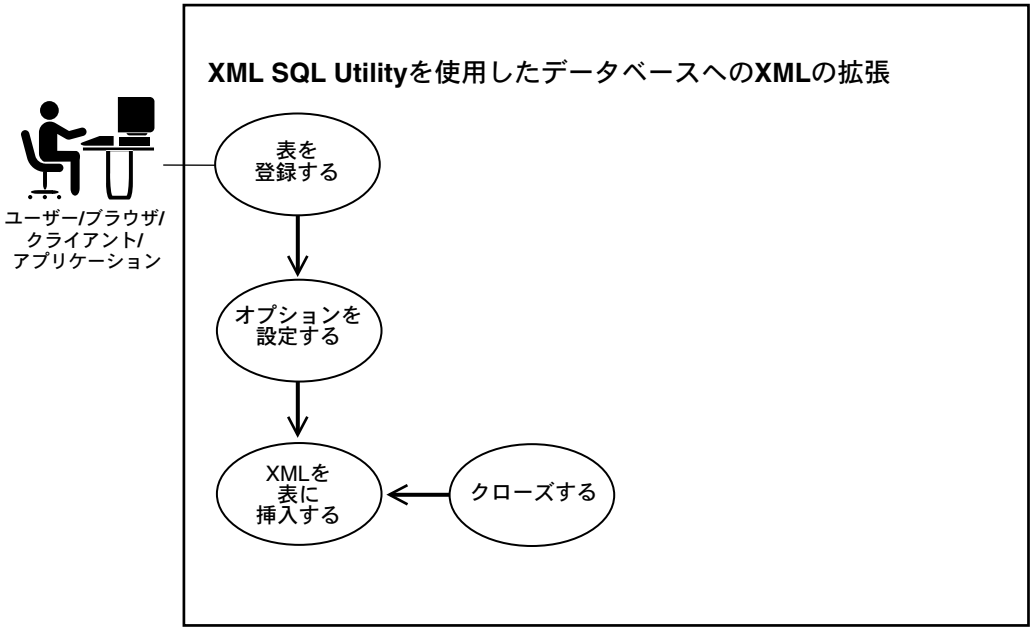
注意： 結果が null になるときから例外ハンドラになるときに、終了を確認する条件が変わったことに注意してください。

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 8-6 に、XSU を使用した XML 格納のプロセスを示します。

図 8-6 XML SQL Utility 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: 列への XML 値の挿入（Java）

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=?
```

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捕捉し、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捕捉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捕捉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捕捉し、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;
    }
}
```

XML SQL Utility（XSU）に関する FAQ

この項では、XML SQL Utility（XSU）に関する質問および回答を示します。

XSU を使用して XML を格納する場合のスキーマ構造

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

回答 : 現在、XML SQL Utility（XSU）は、データを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、複数表にまたがるビューを（必要に応じてオブジェクト・ビューを使用して）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

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 / 小文字の区別 : ignoreCase の使用

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 / 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 / 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 / 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

回答 : XSU を使用して、DTD からデータベース・スキーマは生成できません。DTD には多くのデメリットがあるため、この機能は使用できません。XML Schema の W3C 勧告はサポートされていますが、この機能は XSU ではサポートされていません。

XSU の Thin ドライバ接続文字列の例

XML SQL Utility 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、Thin JDBC 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

XML SQL Utility 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` 文、`DELETE` 文または `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

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。

9

XSQL Pages 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 Pages パブリッシング・フレームワークの概要

Oracle XSQL Pages パブリッシング・フレームワークは、XML 情報を必要な形式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 Pages 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形式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 Pages 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 Pagaes パブリッシング・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。

- リクエスト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コンテンツを変更することなく、表示を修正および改善できます。

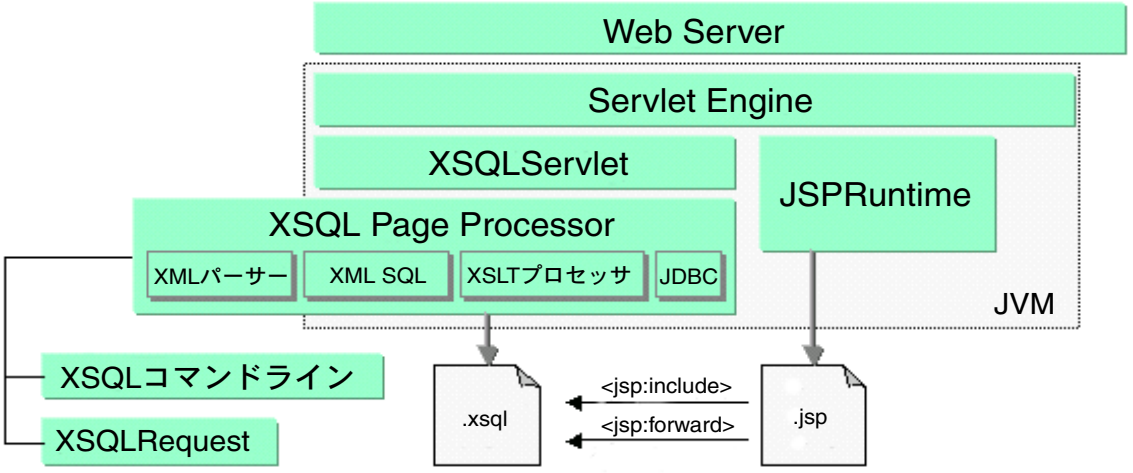
Oracle XSQL ページを使用して実行できる操作

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形式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 9-1 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 9-1 XSQL Pages パブリッシング・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リクエストを受信します。
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形式に変換します。

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・ドリブン・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（9-53 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 A「XDK for Java: 仕様およびクイック・リファレンス」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 ページの入手方法

XSQL Servlet は、Oracle9*i* 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 Pages 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、JVM（1.1.8、1.2.2 または 1.3）のみです。XSQL Pages パブリッシング・フレームワークには、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が必要です。

- Oracle XML Parser および Oracle XSLT プロセッサ（`xmlparserv2.jar`）
- Oracle XML SQL Utility（`xsu12.jar`）

どちらの Java アーカイブ・ファイルも、XSQL Pages パブリッシング・フレームワークを実行している CLASSPATH に含まれている必要があります。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 Pages パブリッシング・フレームワークには JDBC ドライバも必要です。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 Driver（`classes12.jar`）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込むため、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 Pages パブリッシング・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* Application Server リリース 1（9.0.1）およびリリース 2（9.0.2）
- 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 JavaServer Web Development Kit（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 Oracle9*i* 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。

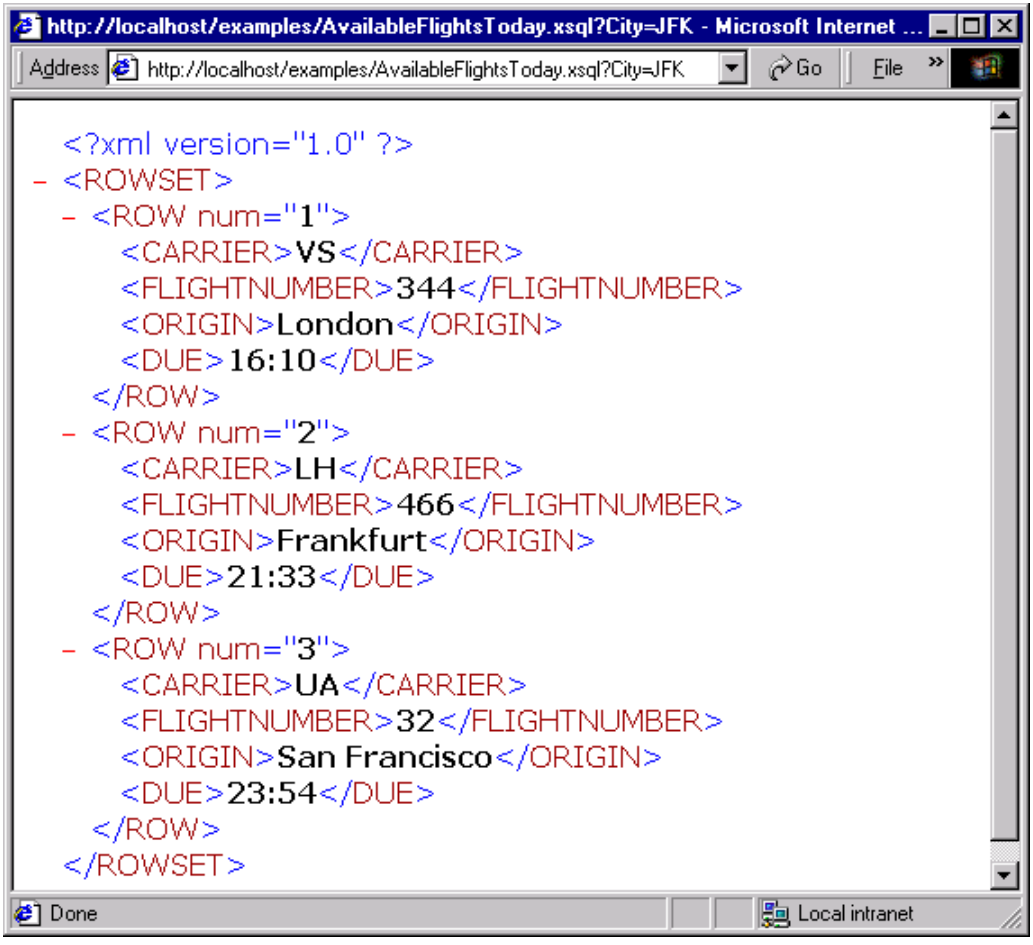
```
<?xml version="1.0"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
      FROM FlightSchedule
     WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
       AND Destination = ?   /* 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リクエストされます。ただし、Internet Explorer 5.0 などのブラウザを使用する場合、図 9-2 に示すような XML での結果を直接表示できます。

図 9-2 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。

ページ内の最初の（この場合は唯一の）要素 <xsql:query> に、名前空間の接頭辞 xsql を Oracle XSQL 名前空間識別子 urn:oracle-xsql のシノニ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、リクエスト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。

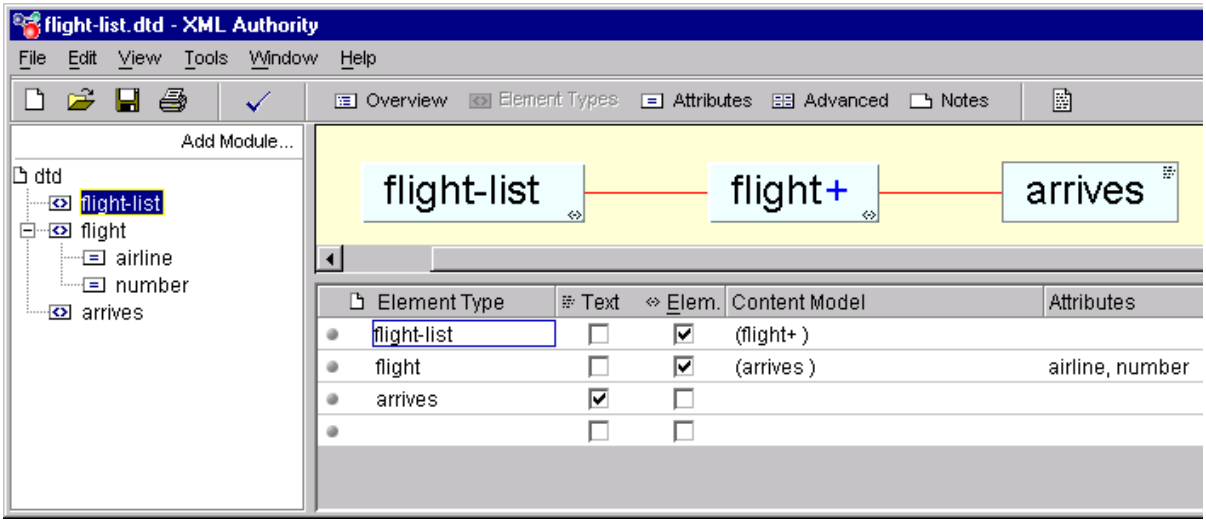
別の XML 形式への XML データグラムの変換

図 9-2 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 Document Type Definition（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 9-3 に示すとおり flight-list.dtd の構造をブラウズできます。

図 9-3 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。

- 1 つ以上の <flight-list> 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"

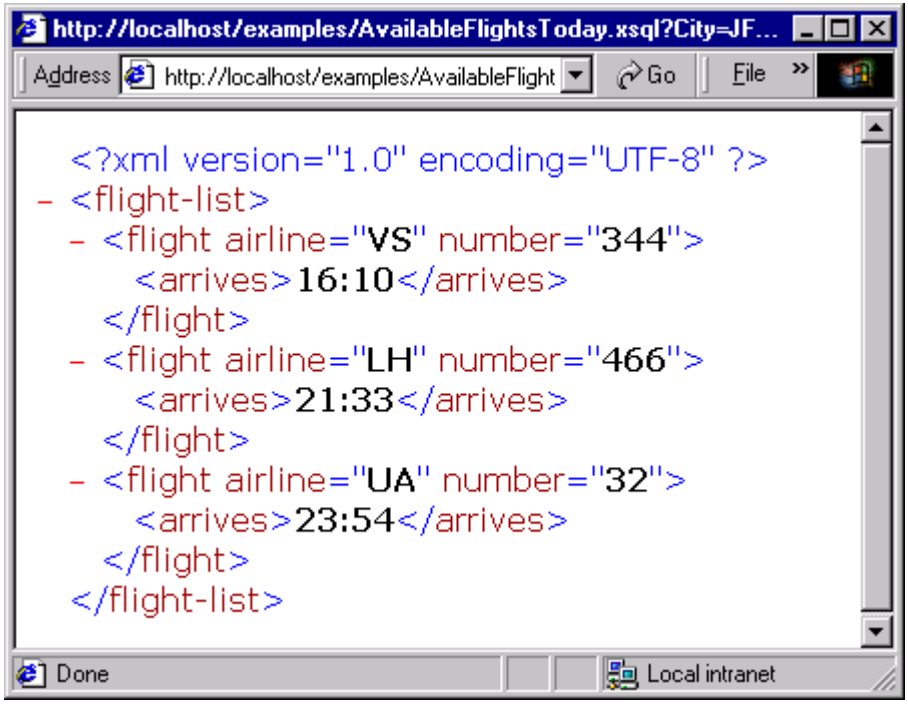
  この属性は、ドキュメントを XSLT 1.0 スタイルシートとして識別します。version 属性は、すべての XSLT スタイルシートが、妥当であると識別され、XSLT プロセッサによって認識されるために必要です。

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 9-4 に示すとおり、リクエスト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。

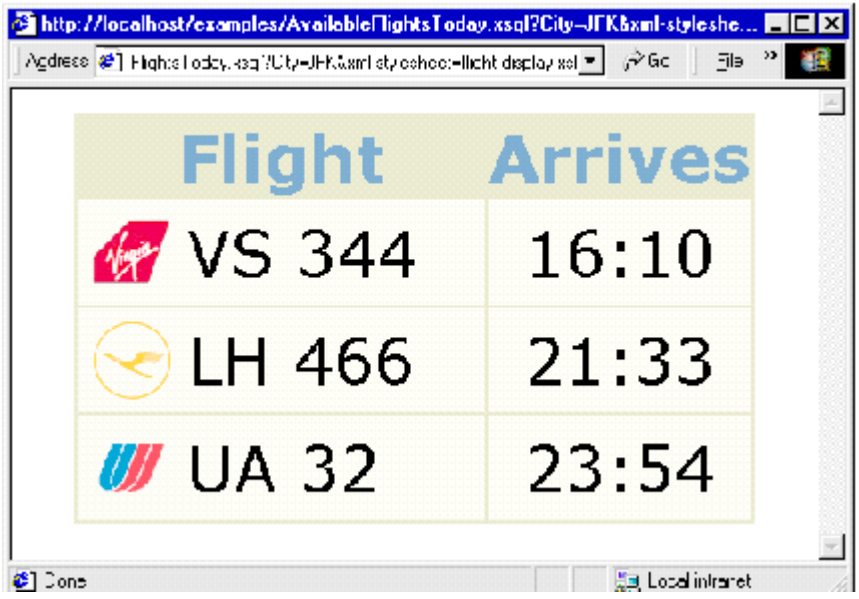
図 9-4 業界標準の XML 形式の XSQL ページ



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 9-5 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 9-5 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、ページをリクエストする適切な 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。Oracle9*i* 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 JVM 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 XML SQL Utility）
- `classes12.jar`（Oracle JDBC Drivers）

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.classes12.jar;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.jarclasses12.jar:/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、リクエスト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、リクエスト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、リクエスト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、次のようにネストした <xsql:no-rows-query> 要素を含めることによって、代替の問合せを作成できます。

```
<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 9-1 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得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 9-1 <xsql:query> の属性

属性名	説明
bind-params = "*string*"	順序付けられ、空白で区切られた 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 / 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。
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。

表 9-1 <xsql:query> の属性（続き）

属性名	説明
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 / 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 9-2 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 9-2 <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、空白で区切られた 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 9-1 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 9-3 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 9-3 <xsql:include-owa> の属性

属性名	説明
bind-params = "*string*"	順序付けられ、空白で区切られた 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、リクエスト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・リクエストで 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>
```

次のようなリクエストで顧客 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 データをリクエストできます。

```
http://yourserver.com/fin/CustomerPortfolio.xsql?custid=1001
```

bind-params 属性の値は、空白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 属性には空白で区切られた 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 をリクエストすると、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、次のとおりリクエストします。

```
$ 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、リクエスタが各リクエスト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リクエスト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、次のようなリクエストで 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リクエストした場合、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、次のようなリクエストを行ったと想定します。

```
http://yourserver.com/example.xsql?val=3
```

このリクエストは、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・リクエストは、次の結果を戻します。

```
<example>
  <rowset/>
</example>
```

これは、デフォルト値またはリクエスト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、リクエスト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 という名前のリクエスト・パラメータの値（指定されている場合）
3. 前述の値が設定されていない場合は、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 という名前のリクエスト・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

ユーザーがリクエスト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、リクエスト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 式を使用してリクエスト・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、リモート・リソースまたはデータベース・ドリブンの XML 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 または SQL 文で指定されます。

このアクションの構文は、次のとおりです。

```
<xsql:include-xml href="URL"/>
```

または

```
<xsql:include-xml>
  SQL select statement selecting a single row containing a single
  CLOB or VARCHAR2 column value
</xsql:include-xml>
```

URL には、他の Web サイトから XML を取得するための HTTP ベースの絶対 URL、または相対 URL を指定できます。href 属性と 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 9-5 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 9-4 <xsql:include-xml> の属性

属性名	説明
bind-params = "*string*"	順序付けられ、空白で区切られた 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。

<xsql:include-posted-xml> アクション

<xsql:include-posted-xml> アクションは、リクエスト内にポストされた XML 文書を XSQL ページに挿入します。XML 文書ではなく HTML 形式をポストした場合、挿入される XML は、<xsql:include-request-params> アクションによって挿入されるものと類似したものになります。

<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>
```

または

```
<xsql:set-page-param name="paramname" xpath="XPathExpression"/>
```

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。この方法では、<xsql:set-page-param> 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。

value 属性または SQL 文を指定するもう 1 つの方法として、xpath 属性を指定して、ページ・レベルのパラメータを XPath 式の値に設定できます。XPath 式は、XSQL Page Processor にポストされた XML 文書または HTML 形式に対して評価されます。xpath 属性の値には、任意の有効な XPath 式を指定できます。他の XSQL アクション・ハンドラ要素と同様に、XSQL パラメータを属性値の一部として使用して、オプションで作成できます。

ページのプライベート・パラメータが設定されると、後続のアクション・ハンドラは、この値を {@po_id} などの字句パラメータとして、または SQL 操作をサポートするアクション・ハンドラの bind-params 属性でその名前を参照することによって、SQL バインド・パラメータ値として使用できます。

name 属性を使用するのではなく、単一の SQL 文の結果に基づいて複数のセッション・パラメータ値を設定する必要がある場合は、names 属性を使用して、1 つ以上のセッション・パラメータ名を空白またはカンマで区切ったリストを指定できます。次に例を示します。

```
<xsql:set-page-param names="paramname1 paramname2 paramname3">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM table
   WHERE clause_identifying_a_single_row
</xsql:set-page-param>
```

name 属性または names 属性のいずれかが必要です。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 9-5 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 9-5 <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。
names = "*string string ...*"	値を設定するページ・パラメータ名を空白またはカンマで区切ったリスト。name 属性または names 属性のいずれかを使用します。両方は使用できません。
bind-params = "*string*"	順序付けられ、空白で区切られた 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。
xpath = "*XPathExpression*"	XSQL Page Processor にポストされた XML 文書または HTML 形式に対して評価された XPath 式に、パラメータの値を設定します。

<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 属性を使用するのではなく、単一の SQL 文の結果に基づいていくつかのセッション・パラメータ値を設定する必要がある場合は、names 属性を使用して、1 つ以上のセッション・パラメータ名を空白またはカンマで区切ったリストを指定できます。次に例を示します。

```
<xsql:set-session-param names="paramname1 paramname2 paramname3">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM table
   WHERE clause_identifying_a_single_row
</xsql:set-session-param>
```

name 属性または names 属性のいずれかが必要です。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 9-6 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 9-6 <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。
names = "*string string* ..."	値を設定するセッション・パラメータ名を空白またはカンマで区切ったリスト。name 属性または names 属性のいずれかを使用します。両方は使用できません。
bind-params = "*string*"	順序付けられ、空白で区切られた 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*"	セッション・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。Cookie に割り当てられる値には、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 属性を使用するのではなく、単一の SQL 文の結果に基づいて複数の Cookie 値を設定する必要がある場合、names 属性を使用して、1 つ以上の Cookie 名を空白またはカンマで区切ったリストを指定できます。次に例を示します。

```
<xsql:set-cookie names="paramname1 paramname2 paramname3">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM table
   WHERE clause_identifying_a_single_row
</xsql:set-cookie>
```

name 属性または names 属性のいずれかが必要です。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。選択リストの列数は、設定されている Cookie の数と一致している必要があります。一致していない場合は、エラー・メッセージが戻されます。

表 9-7 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 9-7 <xsql:set-cookie> の属性

属性名	説明
name = "*string*"	値を設定する Cookie の名前。
names = "*string string* ..."	値を設定する Cookie 名を空白またはカンマで区切ったリスト。name 属性または names 属性のいずれかを使用します。両方は使用できません。
bind-params = "*string*"	順序付けられ、空白で区切られた 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 の値が有効で読込み可能なドメイン。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*"	Cookie 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 の値が有効で読込み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。
immediate = "*boolean*"	Cookie の割当てを現在のページにすぐに表示するかどうかを指定します。通常、現行のリクエストで設定された Cookie は、ブラウザが後続のリクエストで Cookie をサーバーに送信するまでは、表示されません。有効値は、yes および no です。デフォルト値は no です。

<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。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 属性を使用するのではなく、単一の SQL 文の結果に基づいていくつかのスタイルシート・パラメータ値を設定する必要がある場合は、names 属性を使用して、1 つ以上のスタイルシート・パラメータ名を空白またはカンマで区切ったリストを指定できます。次に例を示します。

```
<xsql:set-stylesheet-param names="paramname1 paramname2 paramname3">
  SELECT expression_or_column1, expression_or_column2, expression_or_column3
    FROM table
   WHERE clause_identifying_a_single_row
</xsql:set-stylesheet-param>
```

name 属性または names 属性のいずれかが必要です。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 9-8 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 9-8 <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。
names = "*string string* ..."	値を設定する最上位のスタイルシート・パラメータ名を空白またはカンマで区切ったリスト。name 属性または names 属性のいずれかを使用します。両方は使用できません。
bind-params = "*string*"	順序付けられ、空白で区切られた 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*"	スタイルシート・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リクエスト・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 9-9 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 9-9 <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 文書の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。

XMLType の問合せ結果の反映

Oracle9*i* では、XML ベースのデータベース・コンテンツの格納および問合せに、XMLType の使用が導入されています。データベース XML 機能を使用して、XSQL ページに挿入する XML を生成できます。その方法は次の 2 つのいずれかです。

- <xsql:query>: XMLType 型の列を含む問合せを処理します。ただし、CLOB 列および VARCHAR2 列内の XML マークアップはリテラル・テキストとして処理されます。
- <xsql:include-xml>: 問合せから取得した 1 つの CLOB または文字列ベースの XML 文書を解析し、挿入します。

前述の 2 つの方法の相違点は、<xsql:include-xml> アクションは、CLOB または文字列値内のリテラル XML を解析し、すぐに要素および属性のツリーとして表現しますが、<xsql:query> アクションは、CLOB または文字列の値列内の XML マークアップをリテラル・テキストのままにしておくという点です。

もう 1 つの相違点は、<xsql:query> は任意の数の列および行の問合せ結果を処理しますが、<xsql:include-xml> は、1 つの行の 1 つの列を処理するように設計されている点です。このため、<xsql:include-xml> を使用すると、そこに含まれる SELECT 文は、1 つの列を含む 1 つの行を戻します。この列は、整形式の XML 文書を含む CLOB または VARCHAR2 値です。この XML 文書は、解析され、XSQL ページに挿入されます。

次の例では、ネストした xmlagg() 関数を使用して、部門およびネストした従業員を含む動的に作成された XML 文書の結果を、<DepartmentList> 要素でラップして 1 つの XML 結果文書に集計します。

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select XmlElement("DepartmentList",
           XmlAgg(
             XmlElement("Department",
               XmlAttributes(deptno as "Id"),
               XmlForest(dname as "Name"),
               (select XmlElement("Employees",
                         XmlAgg(
```

```
                        XmlElement("Employee",
                          XmlAttributes(empno as "Id"),
                          XmlForest(ename as "Name",
                                    sal   as "Salary",
                                    job   as "Job")
                        )
                      )
                    )
                  from emp e
                 where e.deptno = d.deptno
                )
              )
            )
          ) as result
   from dept d
  order by dname
</xsql:query>
```

別の例を考えてみます。MOVIES という XMLType の表に多数の <Movie> という XML 文書が格納されているとします。それぞれの文書は、次のようになっているとします。

```
<Movie Title="The Talented Mr.Ripley" RunningTime="139" Rating="R">
  <Director>
    <First>Anthony</First>
    <Last>Minghella</Last>
  </Director>
  <Cast>
    <Actor Role="Tom Ripley">
      <First>Matt</First>
      <Last>Damon</Last>
    </Actor>
    <Actress Role="Marge Sherwood">
      <First>Gwenyth</First>
      <Last>Paltrow</Last>
    </Actress>
    <Actor Role="Dickie Greenleaf">
      <First>Jude</First>
      <Last>Law</Last>
      <Award From="BAFTA" Category="Best Supporting Actor"/>
    </Actor>
  </Cast>
</Movie>
```

Oracle9*i* の組込み XPath 問合せ機能を使用すると、次の問合せを使用して、データベースに含まれるすべての映画から、アカデミー賞を受賞したすべての出演者の集計リストを抽出できます。

```
  select xmlelement("AwardedActors",
           xmlagg(extract(value(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]')))
    from movies m
```

この XMLType の問合せ結果を XSQL ページに挿入するには、<xsql:query> 要素の中に問合せを貼り付けて、次のように問合せ式の別名（後ろに「as result」を付けるなど）を挿入します。

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select xmlelement("AwardedActors",
           xmlagg(extract(value(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]')) as result
    from movies m
</xsql:query>
```

xmlelement() と xmlagg() を組み合せて使用して、データベースによって、問合せで識別されたすべての XML フラグメントを、単一の整形式の XML 文書に集計していることに注意してください。xmlelement() と xmlagg() を組み合せて使用すると、次のような整形式の結果が生成されます。

```
<AwardedActors>
  <Actor>...</Actor>
  <Actress>...</Actress>
</AwardedActors>
```

バインド変数を式に連結する場合は、XPath 式の途中で標準の XSQL ページのバインド変数機能を使用することもできます。たとえば、値「Oscar」を「award-from」という名前のパラメータにするには、XSQL ページを次のように使用します。

```
<xsql:query connection="orcl92" xmlns:xsql="urn:oracle-xsql"
            award-from="Oscar" bind-params="award-from">
  /* Using a bind variable in an XPath expression */
  select xmlelement("AwardedActors",
           xmlagg(extract(value(m),
                  '/Movie/Cast/*[Award[@From="'|| ? ||'"]]'))) as result
    from movies m
</xsql:query>
```

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 XML SQL Utility 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。

XML SQL Utility 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、XML SQL Utility は挿入、更新または削除操作（またはそのすべて）を自動化することができます。XML SQL Utility 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後、結果として戻る正規の XML を挿入、更新、削除するように XML SQL Utility にリクエストできます。

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>`

  リクエストでポストされ、オプションで変換された XML 文書を表に挿入します。表 9-10 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:update-request>`

  リクエストでポストされ、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 9-11 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:delete-request>`

  リクエストでポストされ、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 9-12 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。

- `<xsql:insert-param>`

  リクエスト・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 9-13 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 9-10 <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。指定する列名は空白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 9-11 <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。指定する列名は空白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。指定する列名は空白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 9-12 <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。指定する列名は空白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。
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。

表 9-13 <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。指定する列名は空白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。

異なる XML ポスト・オプションの理解

XSQL Pages パブリッシング・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、リクエストの本体に 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 リクエストを送信できます。

   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 パラメータ値のコンテンツが、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、ブラウザはリクエストの本体にエンコードされたすべての HTML フォーム・フィールドおよびその値を含み、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、このリクエストは 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）リクエスト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リクエスト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>
```

リクエスト・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 リクエスト・ドキュメントの未加工の XML が表示されます。この XML は、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、9-66 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・コンテンツまたは要素コンテンツを <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 9-14 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 9-14 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・ドリブンの 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、形式に 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。

表 9-14 XSQL Servlet の例（続き）

デモンストレーション名	説明
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。
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、リクエストで 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 / 詳細ビューを 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。

表 9-14 XSQL Servlet の例（続き）

デモンストレーション名	説明
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。
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および貼付けして、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。Oracle9*i* Text（interMedia 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 / 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

リクエスト中の現行の XSQL ページが許可する場合、リクエストで 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` パラメータをリクエスト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コンテンツ・タイプの制御

提供する情報のコンテンツの型を設定することは、非常に重要です。これによって、リクエスト側クライアントは、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、リクエスタにレスポンスする前に 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 XSLT 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 プロセッサなど）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 ページがリクエストを行うユーザー・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リクエスト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 / 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 9-15 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 9-15 <?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、リクエスト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 / 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。
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 9-16 に、設定可能なすべてのパラメータを定義します。

表 9-16 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 ヘッダーがリクエスト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 タイプをコンテンツとして指定した <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。
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> 要素のコンテンツとして、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リクエストが試行されると、XSQL Page Processor は XSQLConfig.xml ファイルを再読込みして、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。

表 9-16 XSQLConfig.xml の構成設定（続き）

構成設定名
`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得する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 Drivers 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。
`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。

表 9-16 XSQLConfig.xml の構成設定（続き）

構成設定名
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・リクエスト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リクエスト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。
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/owa/fetch-style <xsql:include-owa> アクションがデフォルトで使用する OWA ページ・バッファのフェッチ・スタイルを設定します。有効値は、CLOB または TABLE で、値を指定しない場合のデフォルトは CLOB です。 CLOB に設定すると、プロセッサは一時 CLOB を使用して OWA ページ・バッファを取得します。 TABLE に設定すると、プロセッサは、XSQL_OWA_ARRAY という Oracle ユーザー定義型を必要とする、より効率的な方法を使用します。この型は、次の DDL 文を使用して手動で作成する必要があります。 CREATE TYPE xsql_owa_array AS TABLE OF VARCHAR2(32767)
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。

表 9-16 XSQLConfig.xml の構成設定（続き）

構成設定名
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）を報告するコンテンツを持つアクション・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、リクエストで特殊な xml-stylesheet パラメータの値を指定し、リクエスト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。
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 プロセッサは、*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。

表 9-16 XSQLConfig.xml の構成設定（続き）

構成設定名
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。
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、XSL Formatting Objects（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。

```
<?xml version="1.0"?>
<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-reference="first">

  <fo:flow flow-name="xsl-region-body">

      <fo:block font-size="24pt" line-height="24pt" 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>
```

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 つの追加操作をリクエスト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リクエストに渡す操作

  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テーブルを渡すことは、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 9-17 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 9-17 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-posted-xml>`	`XSQLIncludePostedXMLHandler`
`<xsql:include-xsql>`	`XSQLIncludeXSQLHandler`

表 9-17 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:include-owa>	XSQLIncludeOWAHandler
<xsql:action>	XSQLExtensionActionHandler
<xsql:ref-cursor-function>	XSQLRefCursorFunctionHandler
<xsql:include-param>	XSQLGetParameterHandler
<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 文書の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・フラグメントのコンテンツに置き換えます。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 9-18 に、有効なメソッドを示します。

表 9-18 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・コンテンツを戻します。
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・リクエスト・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、リクエストのタイプが Servlet の場合は、`XSQLPageRequest` オブジェクトをより固有の `XSQLServletPageRequest` に捕捉し、`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。
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得します。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。

表 9-18 oracle.xml.xsql.SQLActionHandlerImpl に対する有効なメソッド（続き）

メソッド名	説明
appendSecondaryDocument	外部 XML 文書のコンテンツ全体をアクション・ハンドラの結果コンテンツのルートに追加します。
addResultElement	テキスト・コンテンツを含む単一の要素をアクション・ハンドラの結果コンテンツ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	空白で区切られた `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。
requiredConnectionProvided	このリクエスト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 文書の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 クラスを介してリクエスト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・リクエスト用の 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 によるルートからリクエスト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 のリクエスト、レスポンス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 / 小文字を区別）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 OutputStream に出力すると、バイナリ情報が Servlet の OutputStream 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。同一のリクエスト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、現行のリクエストが使用する 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。

XSQLConnectionManager のカスタム実装で、オプションの oracle.xml.xsql.XSQLConnectionManagerCleanup インタフェースも実装する場合は、Connection Manager で、割当て済のリソースをクリーンアップできます。たとえば、サーブレット・コンテナが XSQLServlet サーブレットに対して destroy() メソッドを起動すると（サーブレットのオンライン管理中などに発生）、Connection Manager はサーブレットの破棄プロセスの一部として、リソースをクリーンアップできます。

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、XSLT 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、引き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。

XSQL Servlet に関する FAQ

この項では、XSQL Servlet についての質問および回答を示します。

XSQL 出力を WML ドキュメントに変換中に DTD を指定する方法

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。次に例を示します。

```
<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、次のコードを含む XML 結果が生成されます。

```
<!DOCTYPE xxxx SYSTEM "your.dtd">
```

"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 の条件文の記述

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。

問合せで取得された値を他の問合せの WHERE 句で使用する方法

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、ページ・パラメータを使用して行うことができます。次の例を参照してください。

```
<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で XSQL Servlet を使用する方法

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 Drivers とともに使用する場合にのみ機能します。

XSQL Servlet を使用して JServ プロセスにアクセスする方法

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 cannot aquire a database connection to process page
```

リクエストがタイムアウトし、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 Oracle9*i* 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。XSQconnections.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 サーブレットを正常に実行できるかどうかをテストしてください。

Oracle8*i* Lite で XSQL を実行する方法

XSQL Servlet を、Windows 98 で Oracle8*i* Lite および Apache JServ Web サーバー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。Oracle8i 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

<><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 によってリクエスト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>
```

また、次のコードを使用して、複数値のページ・パラメータの値を SQL 文で 0（ゼロ）以外の前に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>
```

Oracle 7.3 で XSQL Servlet を使用する方法

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？XML SQL Utility（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 Oracle9*i* JDBC Drivers を使用していることを確認してください。

OUT 変数が <xsql:dml> でサポートされない理由

`<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すべての必要な 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、次のようなリクエストを実行できます。

```
http://yourserver.com/addmult.xsql?arg1=30&arg2=45
```

このリクエストは、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>
```

接続不能エラーの原因

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-007: Cannot acquire a database connection to process page.
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。

1. データベースがローカル・ホスト・マシン上にない。
2. データベースの SID が ORCL ではない。
3. 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

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

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、このページをリクエストすると、エラーが発生します。

```
XSQL-00005: XSQL page is not well-formed.
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（&）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。

- 、< などの実体参照を指定する一連の文字がアンパサンド（&）で始まること
- <SomeElement> などの要素を指定する一連の文字が不等号（<）で始まること

リテラル・アンパサンドまたは不等号を挿入するには、それぞれ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>
```

```
Alternatively, you can surround an entire block of text with a so-called CDATA
section that begins with the sequence <![CDATA[ and ends with a corresponding ]]>
sequence. All text contained in the CDATA section is treated as literal.
<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>
```

XML のポストを試行したときに「No Posted Document to Process」というエラーが発生する原因

<xsql:insert-request> タグを含む XSQL ページへのリンクをクリックすると、ページに「No Posted Document to Process」というメッセージが表示され、データベースにデータが挿入されません。原因は何ですか？

回答： XML 情報を処理するために XSQL ページにポストする場合は、HTTP POST メソッドで送信する必要があります。この XML 情報は、HTTP POST によってポストされた HTML 形式または HTTP POST によって送信された XML 文書になります。HTTP GET を使用すると、ポスト済の文書が存在しないため、「No Posted Document to Process」というエラーが発生します。正しく動作させるには、HTTP POST を使用してください。

XSQL での SOAP のサポート

XSQL ページを使用して SOAP サービスを実装し、クライアントが HTTP 経由でこれを使用することはできますか？

回答： できます。XSQL ページでは、`<xsql:set-page-param>` アクションの xpath="*XpathExpression*" 属性を使用して受信 SOAP メッセージのコンテンツにアクセスできます。また、`getPageRequest().getPostedDocument()` をコールして、カスタマ・アクション・ハンドラでポスト済の SOAP メッセージ本体に直接アクセスすることもできます。クライアントに戻す SOAP レスポンス本体を作成するには、XSLT スタイルシートまたはカスタム・シリアル化コードの実装を使用して、XML レスポンスを適切な SOAP エンコーディング形式で書き出します。

したがって、XSQL ページを使用した SOAP サービスの実装は、自動的には実行されませんが、実行可能です。XSQL ページを使用して SOAP ベースの Web サービスを実装する例については、XSQL ページ・フレームワークに付属の AirportSOAP デモを参照してください。

XSQL の接続を指定しない方法

リクエストで使用する XSQL の接続を指定しない必要があります。これは可能ですか？

回答： 可能です。ページの **connection** 属性で XSQL パラメータを参照し、接続名のデフォルト値に必ず同じ名前の属性を定義します。次に例を示します。

```
<xsql:query conn="testdb" connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
   :
</xsql:query>
```

パラメータを指定せずにこのページを取得する場合、conn パラメータの値は `testdb` となり、このページは `XSQLConfig.xml` ファイルに定義された `testdb` という接続を使用します。この方法のかわりに `conn=proddb` を使用してページをリクエストすると、`proddb` という接続が使用されます。

データベース接続およびパスワードの格納方法の制御

XSQL にデフォルトで用意されているユーザー名およびパスワードの管理方法（XSQLConfig.xml ファイルを使用）より高度な方法が必要な場合、デフォルトをオーバーライドすることは可能ですか？

回答：可能です。XSQL Page Processor によるデータベース接続の処理方法は、XSQLConnectionManager インタフェースのユーザー独自の実装を作成することによって完全に再定義できます。これを行うには、`oracle.xml.xsql.XSQLConnectionManagerFactory` インタフェースを実装するクラスおよび `oracle.xml.xsql.XSQLConnectionManager` インタフェースを実装するクラスを作成後、XSQLConfig.xml 構成ファイルで使用する XSQLConnectionManagerFactory クラスの名前を変更する必要があります。一度これを行うと、XSQL ページのデフォルト・スキームではなく、定義した接続管理スキームが使用されます。

カスタム Connection Manager の認証情報にアクセスする方法

HTTP 認証メカニズムを使用して、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取得する必要があります。カスタム Connection Manager の `getConnection()` メソッドでこのような情報は取得できますか？

回答：取得できます。`getConnection()` メソッドは、`XSQLPageRequest` インタフェースのインスタンスに渡されます。そのインスタンスから、次の方法によって HTTP リクエスト・オブジェクトを取得できます。

1. リクエストのタイプをテストして、「`Servlet`」であることを確認します。
2. `XSQLPageRequest` を `XSQLServletPageRequest` に捕捉します。
3. **2.** の結果に対して `getHttpServletRequest()` をコールします。

これで、HTTP リクエスト・オブジェクトから認証情報を取得できます。

現行の XSQL ページから名前を取得する方法

現行ページへのリンクを作成するため、実行時に XSQL ページからそのページの名前にアクセスするための一般的で効率的な方法はありますか？

回答：次のようなヘルパー・メソッドを使用すると、カスタム・アクション・ハンドラ内のページの名前を取得できます。

```
// Get the name of the current page from the current page's URI
private String curPageName(XSQLPageRequest req) {
  String thisPage = req.getSourceDocumentURI();;
  int pos = thisPage.lastIndexOf('/');
  if (pos >=0) thisPage = thisPage.substring(pos+1);
  pos = thisPage.indexOf('?');
  if (pos >=0) thisPage = thisPage.substring(0,pos-1);
  return thisPage;
}
```

FOP シリアル化コード使用時にエラーを解決する方法

FOP シリアル化コードを使用して XSQL ページから PDF 出力を生成しようとすると、エラーが発生します。原因は何ですか？

回答 : 通常、CLASSPATH にすべての必要な jar ファイルがそろっていないことが原因です。XSQLFOPSerializer クラスは別の `xsqlserializers.jar` ファイルに存在しますが、FOP の統合を使用するには、CLASSPATH 内に指定する必要があります。これによって、XSQLFOPSerializer クラス自体が Apache からのライブラリの一部と依存性を持ちます。例として、FOP ソフトウェアの Apache FOP 0.20.3RC リリース候補で動作する FOP シリアル化コードのソース・コードを次に示します。

```
package sample;
import org.w3c.dom.Document;
import org.apache.log.Logger;
import org.apache.log.Hierarchy;
import org.apache.fop.messaging.MessageHandler;
import org.apache.log.LogTarget;
import oracle.xml.xsql.XSQLPageRequest;
import oracle.xml.xsql.XSQLDocumentSerializer;
import org.apache.fop.apps.Driver;
import org.apache.log.output.NullOutputLogTarget;

/**
 * Tested with the FOP 0.20.3RC release from 19-Jan-2002
 */
public class SampleFOPSerializer implements XSQLDocumentSerializer {
  private static final String PDFMIME = "application/pdf";
  public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    try {
      // First make sure we can load the driver
      Driver FOPDriver = new Driver();
      // Tell FOP not to spit out any messages by default.
      // You can modify this code to create your own FOP Serializer
      // that logs the output to one of many different logger targets
      // using the Apache LogKit API
      Logger logger = Hierarchy.getDefaultHierarchy()
                               .getLoggerFor("XSQLServlet");
      logger.setLogTargets(new LogTarget[]{new NullOutputLogTarget()});
      FOPDriver.setLogger(logger);
      // Some of FOP's messages appear to still use MessageHandler.
      MessageHandler.setOutputMethod(MessageHandler.NONE);
      // Then set the content type before getting the reader/
      env.setContentType(PDFMIME);
      FOPDriver.setOutputStream(env.getOutputStream());
      FOPDriver.setRenderer(FOPDriver.RENDER_PDF);
      FOPDriver.render(doc);
    }
```

```
    catch (Exception e) {
      // Cannot write PDF output for the error anyway.
      // So maybe this stack trace will be useful info
      e.printStackTrace(System.err);
    }
  }
}
```

この FOP シリアル化コードは、実行時に、CLASSPATH に次の追加の Apache jar ファイルを必要とします。

1. `fop.jar`: Apache FOP レンダリング・エンジン
2. `batik.jar`: Apache Batik SVG レンダリング・エンジン
3. `avalon-framework-4.0.jar`: Apache Avalon Framework 用 API
4. `logkit-1.0.jar`: Apache Logkit 用 API

パフォーマンスを最適化するために XSQL ページを調整する方法

XSQL ページを最速で実行するにはどうしたらよいですか？

回答 : パフォーマンスに最も大きく影響するのは、問合せの対象データのサイズ（および問合せの純粋な速度）です。問合せがチューニング済で、SQL で可能な場所では常に、字句バインド変数ではなく ? バインド変数を使用している場合は、必要な結果を得るための最小量のデータのみが問合せの対象になっていることを確認します。

データを何千行も問い合せている場合、XSLT スタイルシートのみを使用して、ブラウザにこれらの行のうち 10 行のみを表示するようにフィルタリングすることはよい方法ではありません。データベースの機能を最大限に使用して行をフィルタし、可能なかぎり、必要な 10 行のみを戻すようにします。XSQL は、Oracle データベースと、変換言語としての XSLT の機能との間の単純な調整を行っているだけです。

他の接続プール実装における XSQL の使用方法

接続プールから接続を行うように XSQL ページを設定することはできますか？ たとえば、Weblogic Web サーバーで XSQL Servlet を実行中の場合、既存の接続プールから接続を行うには、接続の定義をどのように設定すればよいですか？

回答 : XSQL は独自の接続プーリングを実装しているため、通常は別の接続プールを使用する必要はありません。ただし、適切な形式の JDBC 接続文字列を指定するのみでは WebLogic プールの使用に不十分な場合、`XSQLConnectionManagerFactory` インタフェースおよび `XSQLConnectionManager` インタフェースを実装して、XSQL に独自のカスタム Connection Manager を作成できます。

CLOB に格納された XML 文書を挿入する方法

データベースの CLOB に格納された XML 文書を XSQL ページに挿入する方法を教えてください。

回答 : <xsql:include-xml> を、CLOB 値を取得する問合せとともに使用します。

JSP と XSQL を同じページに組み合せる方法

XSQL タグと JSP タグを同じページに組み合せることはできますか ? また、そのために include タグを使用する必要はありますか ?

回答 : JSP と XSQL とは、2 つの異なるモデルです。JSP は、OutputStream への文字のストリームの書出しに基づくモデルです。XSQL は、純粋な XML または XSLT ベースのモデルです。作業終了時に、HTML、XML などの結果をユーザーに戻します。コードを作成し、XML 文書を文字のストリームとして操作し、内部で大量の再解析を行った場合、XSQL で実装できて JSP で実装できない操作は存在しません。XSQL は、顧客が（XML で表現される）データのコンテンツと（XSLT スタイルシートで表現される）データ表現とを明確に分割する必要がある場合のアーキテクチャに適しています。XSQL は、XML または XSLT アーキテクチャ専用であるため、この目的に最適です。

たとえば、<jsp:include> または <jsp:forward> を使用して、JSP ページに XSQL ページを挿入するか、または JSP ページを XSQL ページに転送することができます。これが最良の方法です。

入力引数に基づいたスタイルシートの選択

入力引数に基づいてスタイルシートを動的に変更できますか ?

回答 : 変更できます。xml-stylesheet 処理命令の href 属性に字句パラメータを使用して実行できます。

```
<?xml-stylesheet type="text/xsl" href="{@filename}.xsl"?>
```

リクエストの一部としてパラメータの値を渡すか、<xsql:set-page-param> を使用して SQL 問合せに基づいてパラメータの値を設定できます。

10

XDK JavaBeans

この章では、Oracle XML で使用できる JavaBeans について説明します。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、Oracle9*i* Enterprise Edition および Oracle8*i* Standard Edition リリース 8.1.6 以上に、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・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 Beans 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。

XML SQL Utility（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。
- XSLT スタイルシートを使用して、XML を変換します。
- 結果の XML 文書を高速に取得するために、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 ファイルを取得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。

参照： 10-5 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 XSLT 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 XSLT 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。

参照： 10-9 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。

参照： 10-13 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 文書をテキスト・エディタで修正する場合、XML および XSL 書式のファイルの構文を色でハイライト表示します。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および検証を可能にします。

> **参照：** 10-16 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 XSLT 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。

> **参照：** 10-20 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、XSLT 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・ドキュメントを含む 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
- XMLDiff Bean の使用

参照：

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

DOMBuilder Bean の使用

DOMBuilder() クラスは、World Wide Web Consortium（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、応答時間が最大で 40% も高速になります。

DOMBuilder Bean の使用方法

図 10-1 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. これによって、`DOMBuilder.addDOMBuilderListener(DOMBuilderListener)` メソッドが入力され、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。

   参照： 適用する使用可能なメソッドのリストは、表 10-1 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するとコールされます。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 10-1 DOMBuilder Bean の使用方法

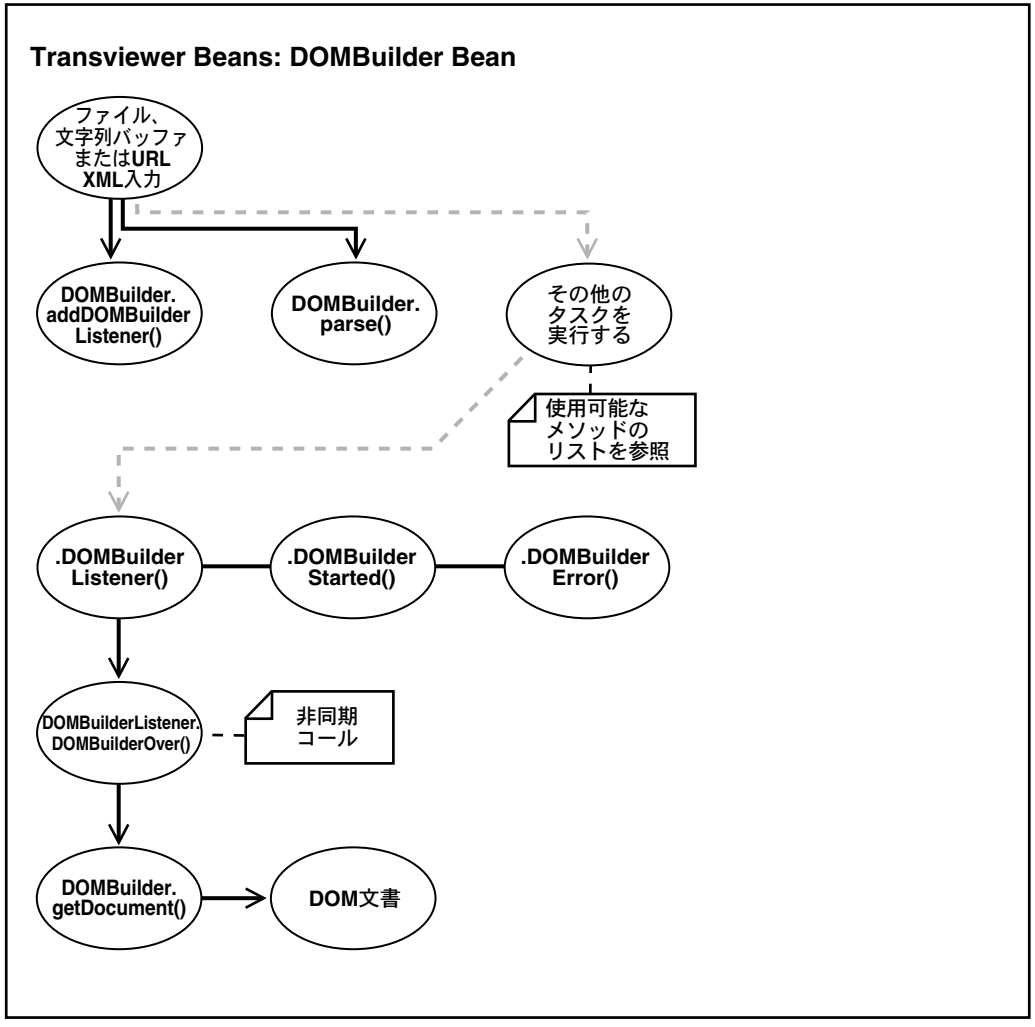


表 10-1 DOMBuilder Bean: メソッド

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 のバージョン番号を戻します。
	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)	与えられた入力ソースから XML を解析します。
	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。
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。
removeDOMBuilderListener(DOMBuilderListener)	DOMBuilderListener を削除します。

表 10-1 DOMBuilder Bean: メソッド（続き）

メソッド	説明
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 XSLT 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、XSLT 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

XSLT 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。

即時レスポンスが可能なユーザー・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、即時レスポンスが可能なユーザー・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、変換のリクエスト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: 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 XSLT 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、XSLT 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。

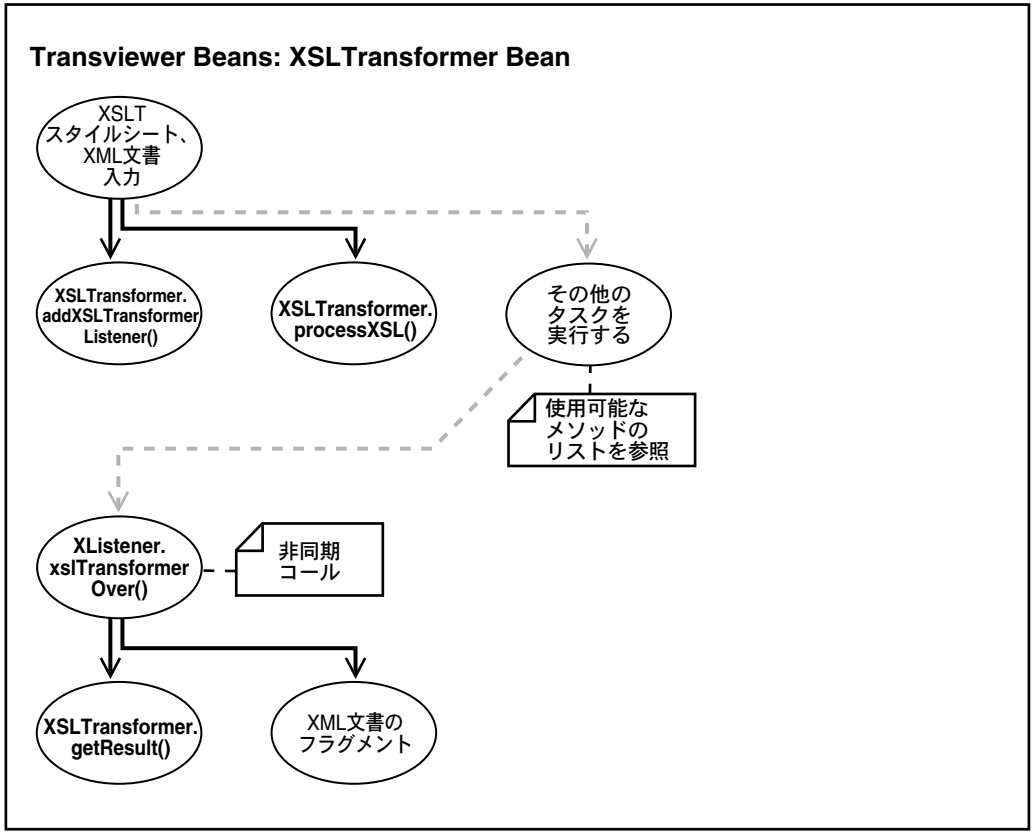
   CLOB 表に格納された HTML は、問合せ中の表に格納された最新のデータを常にミラー化します。JSP（JavaServer Pages）は HTML を表示できます。

   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ときにのみ再生成されます。

XSLTransformer Bean の使用方法

図 10-2 に、XSLTransformer Bean の使用方法を示します。この Bean を実装する例は、10-38 ページの「Transviewer Bean の例 1: AsyncTransformSample.java」を参照してください。

図 10-2 XSLTransformer Bean の使用方法



1. XSLTransfomer は、`XSLTransfomer.addXSLTransformerListener(XSLTransformerListener)` メソッドを使用して、XSLT 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、XSLT 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 10-2 に、XSLTransformer Bean メソッドを示します。

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に関する通知を受け取るには、このインタフェースを実装する必要があります。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 10-2 XSLTransformer Bean: メソッド

メソッド	説明
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。
getId()	一意の XSLTransformer 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 XSLT 変換を開始します。
processXSL(XSLstylesheet, Reader, URL)	バックグラウンドで XSLT 変換を開始します。
processXSL(XSLstylesheet, URL, URL)	バックグラウンドで XSLT 変換を開始します。
processXSL(XSLstylesheet, XMLDocument)	バックグラウンドで XSLT 変換を開始します。
processXSL(XSLstylesheet, XMLDocument, OutputStream)	バックグラウンドで XSLT 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()	
setErrorStream(OutputStream)	XSLT 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)	XSLT 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 10-3「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 10-3 XMLTreeViewer Bean 実行画面： XML 文書のツリーとしての表示

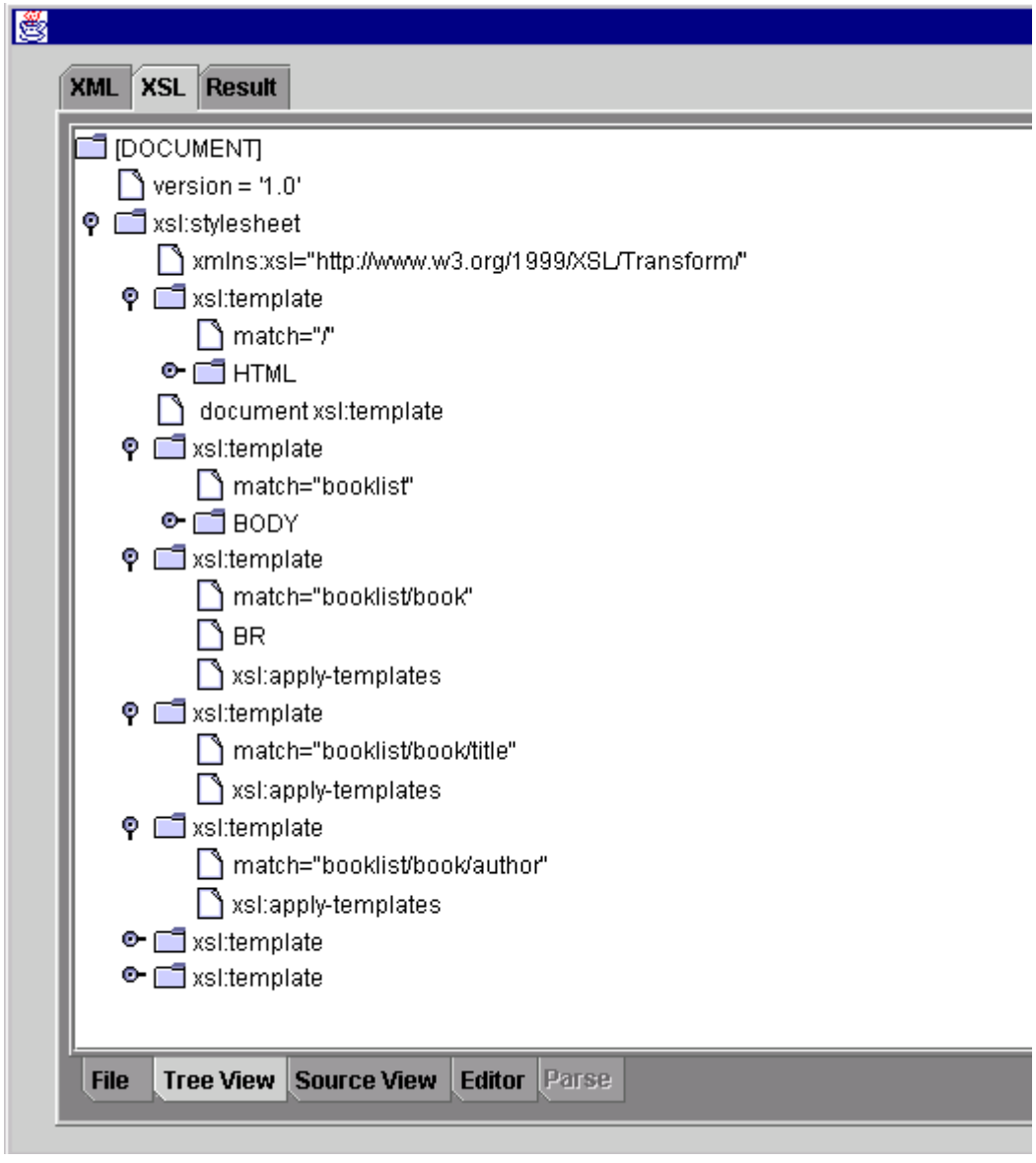
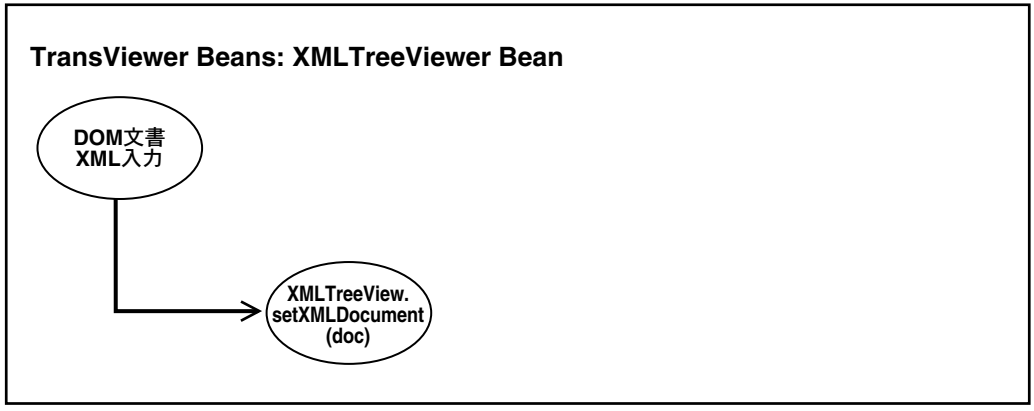


図 10-4 に、XMLTreeViewer Bean の使用方法を示します。DOM XML 文書は、`XMLTreeView.setXMLDocument(doc)` メソッドへの入力です。この入力によって、XML Tree Viewer が XML 文書に対応付けられます。TreeViewer Bean メソッドは次のとおりです。

- 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またはリフレッシュさせます。

図 10-4 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 10-5 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 10-6 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。10-44 ページの「Transviewer Bean の例 2: ViewSample.java」を参照してください。

図 10-5 XMLSourceViewer Bean 実行画面 : 色でハイライト表示された XML 文書の表示

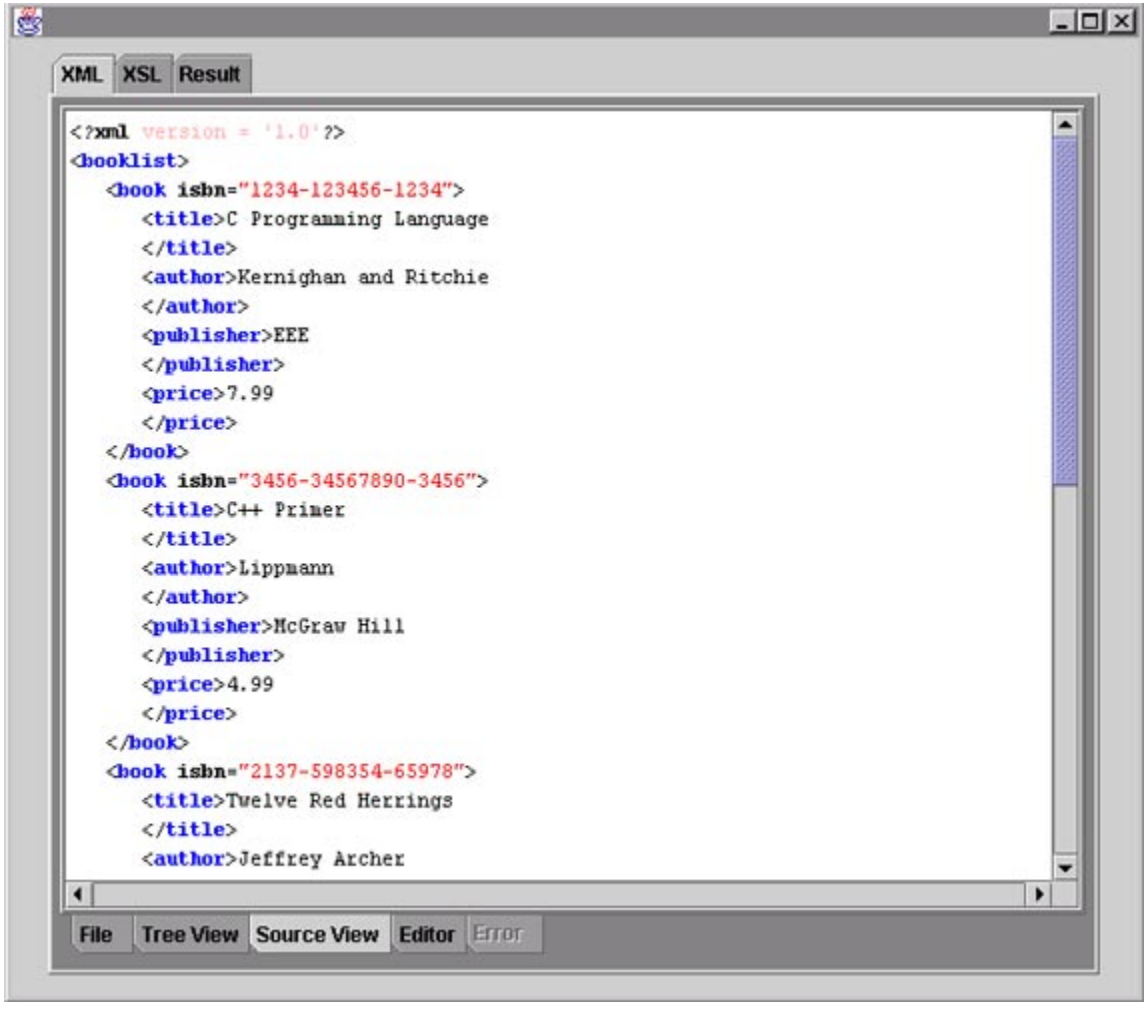


図 10-6 XMLSourceViewer Bean の使用方法

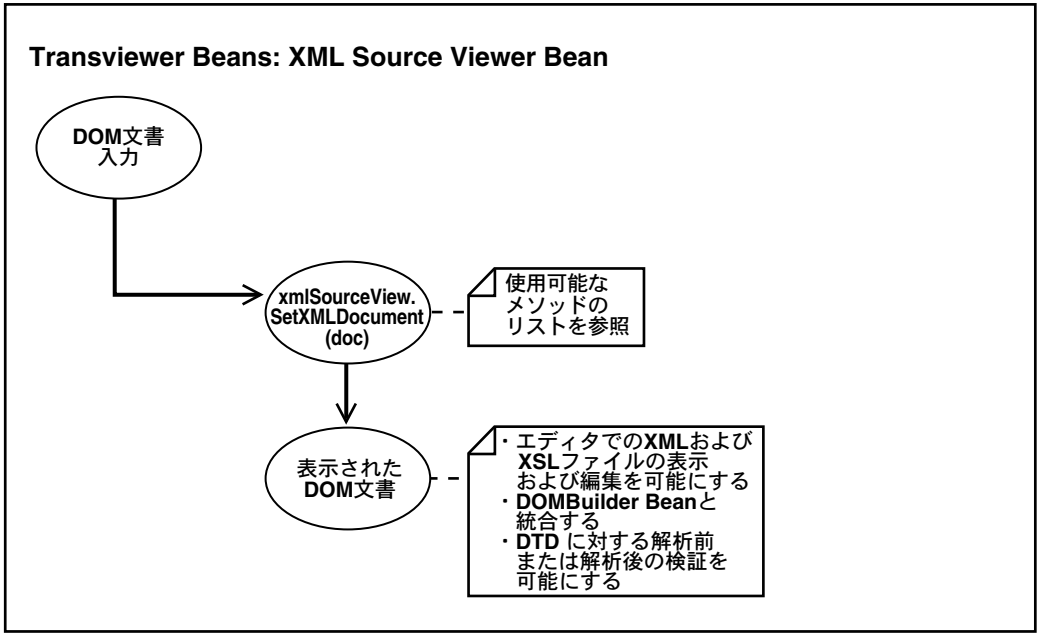


表 10-3 に、XMLSourceView Bean メソッドのリストを示します。

表 10-3 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。

表 10-3 XMLSourceView Bean メソッド（続き）

メソッド	説明
getCDATAFont()	CDATA のフォントを戻します。
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 名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リアンを戻します。
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。

表 10-3 XMLSourceView Bean メソッド（続き）

メソッド	説明
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。
setCDATAForeground(Color)	CDATA のフォアグラウンド・カラーを設定します。
setCommentDataFont(Font)	コメント文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リアンを設定します。
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、XSLT 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、Oracle9*i* 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得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 10-7「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。10-10 ページの「XSL Transviewer Bean の使用例 1: 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。
- XSLT 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。

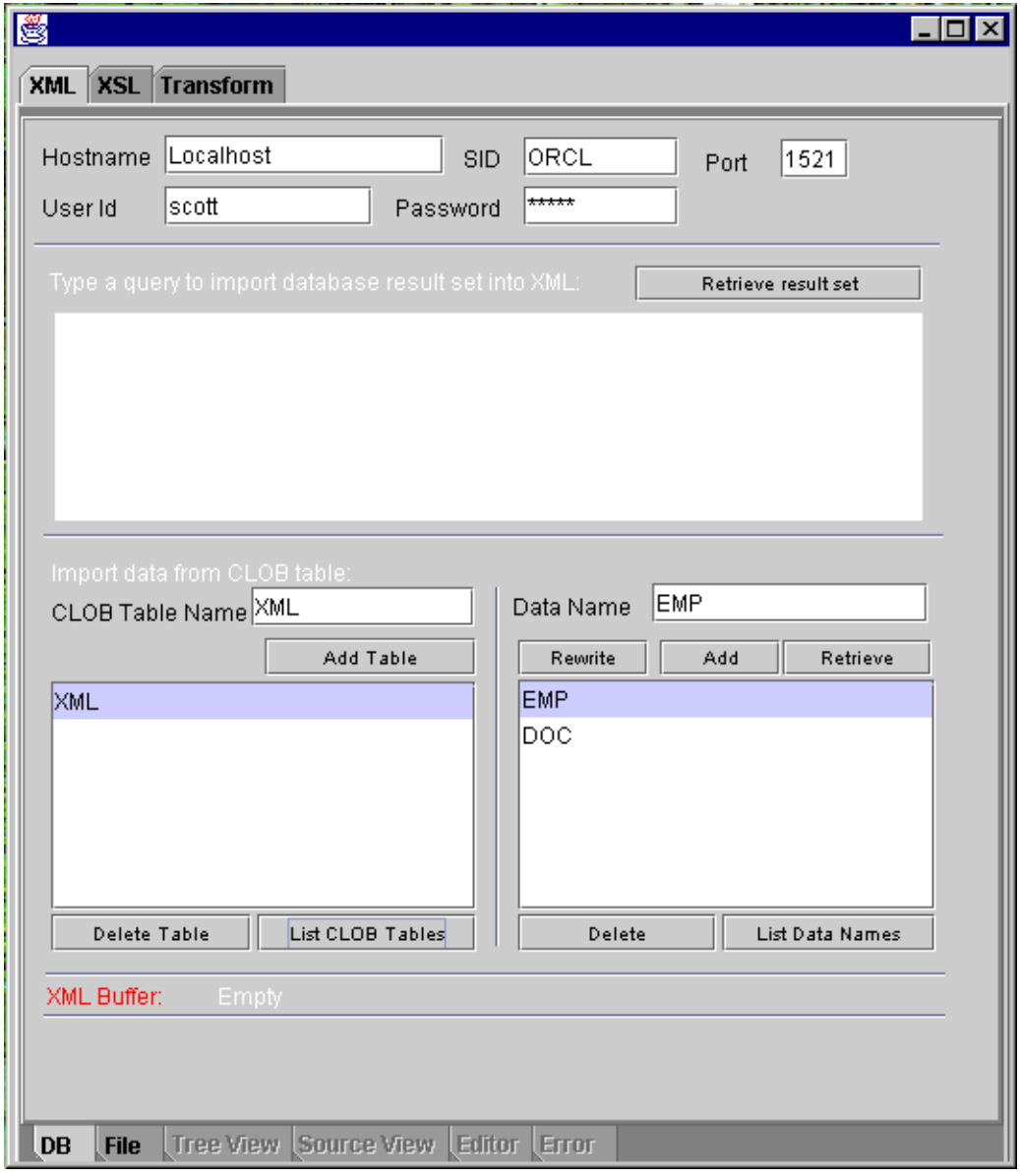
Transviewer Bean アプリケーション

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
- XSLT 変換の編集および適用
- ファイル・システムまたは Oracle9*i* にある XML、XSL および結果ファイルの取得および保存

  参照： XML Transform Panel の使用方法の例は、10-48 ページの「Transviewer Bean の例 3: XMLTransformPanelSample.java」を参照してください。

図 10-7 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、XSLT 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 10-8「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 10-9「DBViewer Bean 実行画面 : XSLT 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。

- Oracle9*i* 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 10-8 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

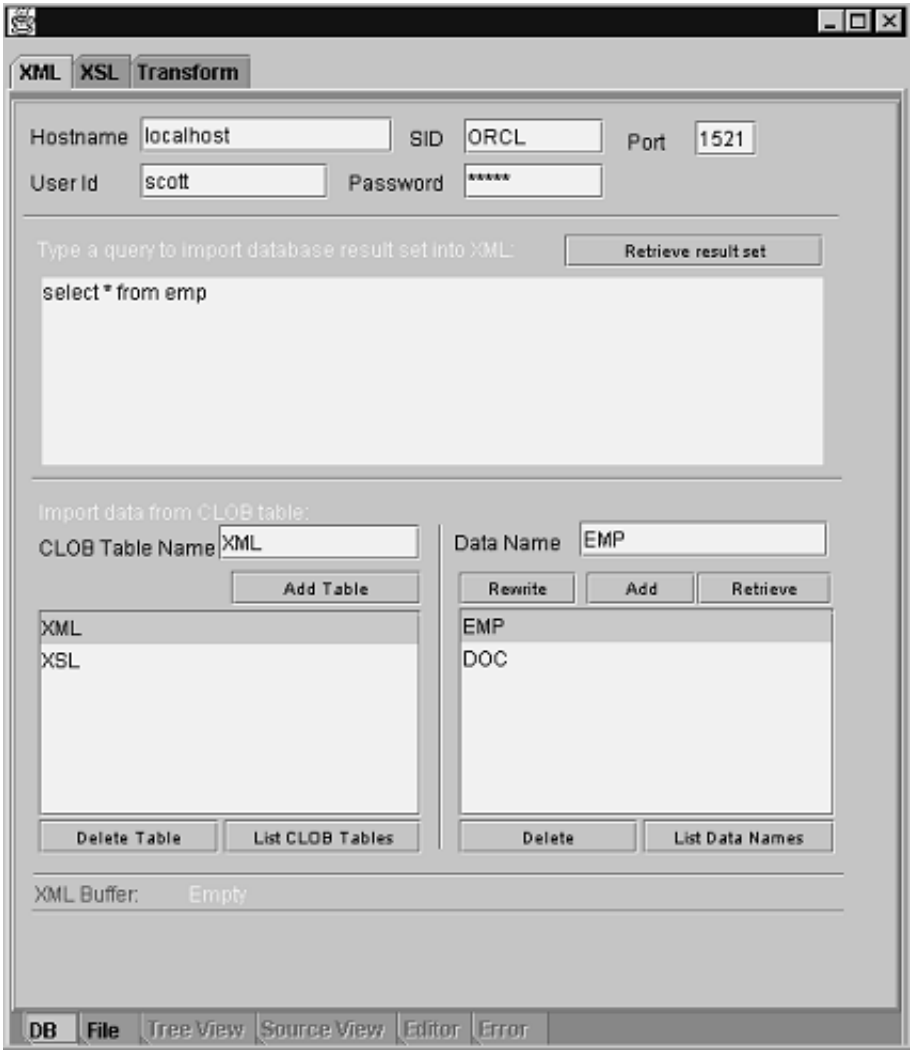
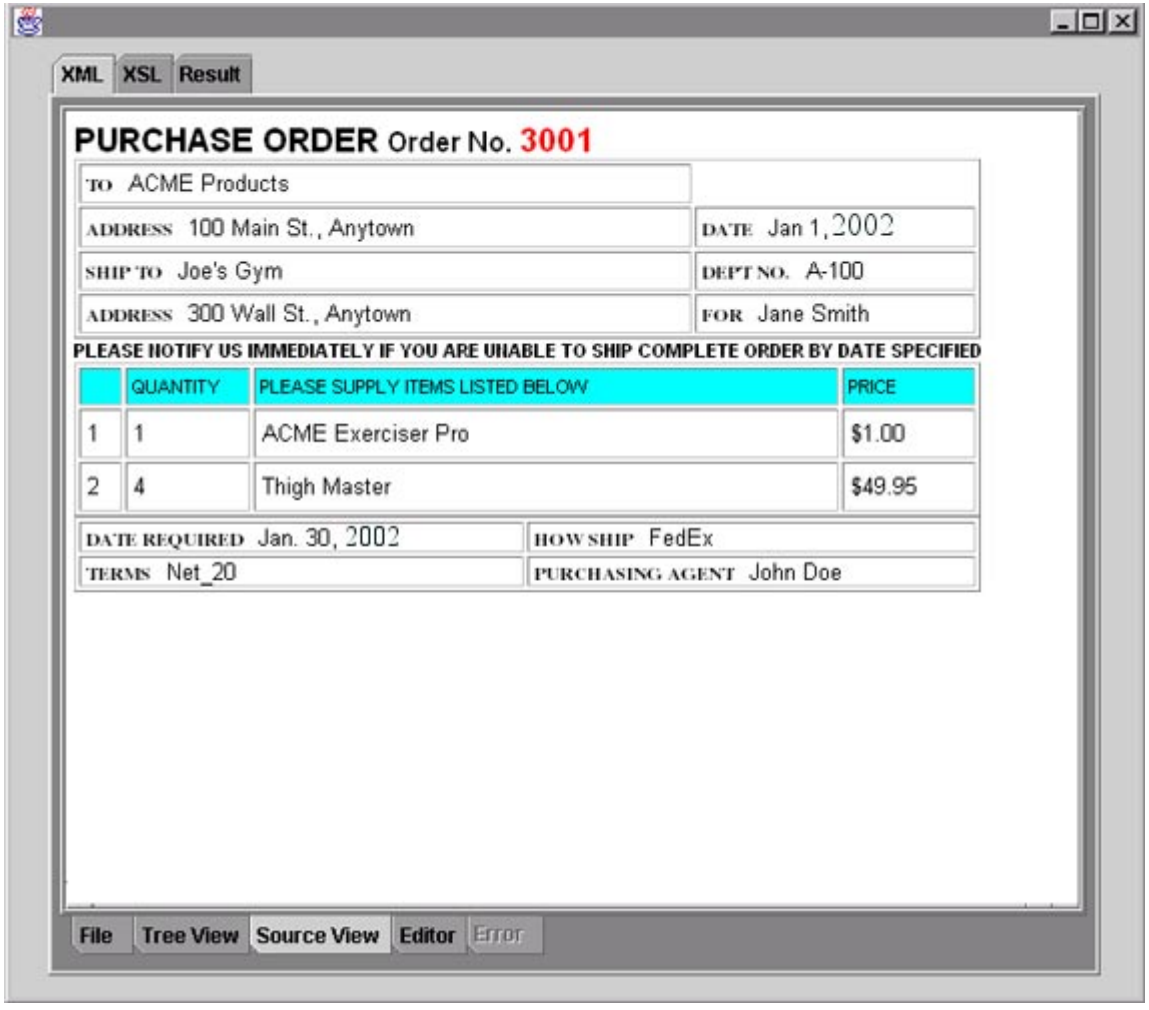


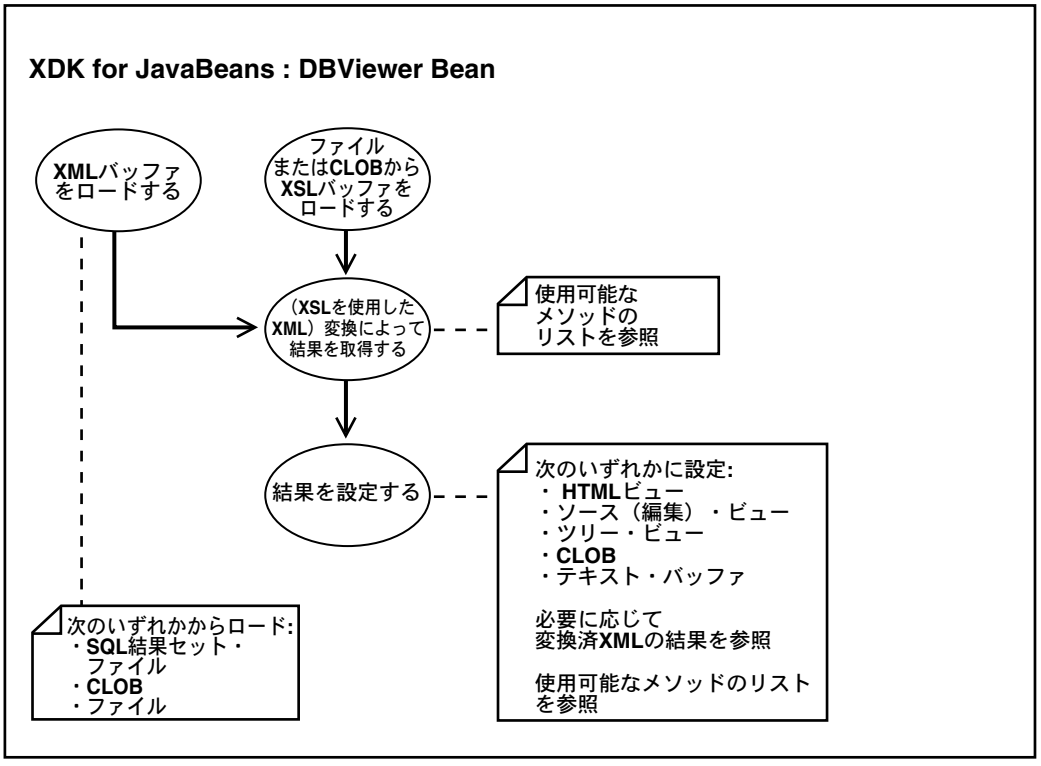
図 10-9 DBViewer Bean 実行画面 : XSLT スタイルシートを使用して HTML に変換した後の XML 文書の表示



DBViewer Bean の使用方法

図 10-10 に、DBViewer Bean の使用方法を示します。

図 10-10 DBViewer Bean の使用方法



DBViewer Bean メソッド

表 10-4 に、DBViewer Bean メソッドのリストを示します。

表 10-4 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。

表 10-4 DBViewer Bean メソッド（続き）

メソッド	説明
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。
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。

表 10-4 DBViewer Bean メソッド（続き）

メソッド	説明
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。
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。

表 10-4 DBViewer Bean メソッド（続き）

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 バッファの新しいテキストを設定します。
setXmlCLOBFileName(String)	XML の CLOB 表名を設定します。
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。

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)
```

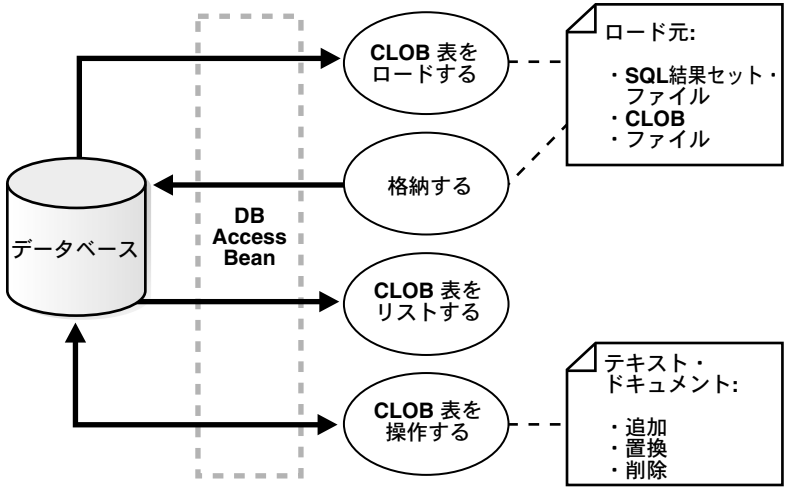
各 XML（またはテキスト）文書は表の行として格納されます。FILENAME フィールドには、その行を検索、更新または削除するためにキーとして使用される一意の文字列があります。文書のテキストは FILEDATA フィールドに格納されます。これは CLOB オブジェクトです。CLOB 表は Transviewer Bean が自動的にメンテナンスします。DBAccess Bean がメンテナンスする CLOB 表は、Transviewer Bean が後で使用します。DBAccess Bean は次のタスクを行います。

- 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 10-11 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 10-11 DBAccess Bean の使用方法



DBAccess Bean メソッド

表 10-5 に、DBAccess Bean メソッドのリストを示します。

表 10-5 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・ファイルを取得します。
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。

XMLDiff Bean の使用

XML Diff Bean は、2 つの XML DOM ツリーを比較します。2 つの XML ツリーを表示して、その違いを示します。ノードの挿入、削除、移動または変更ができます。これらの操作は、次のとおり、それぞれ異なる色またはスタイルで表示されます。

- 赤 : 変更されたノードまたは属性
- 青 : 新しいノードまたは属性
- 黒 : 削除されたノードまたは属性

移動は、削除操作または挿入操作としてビジュアル的に表示されます。

2 つの XML ツリーの違いは、XSL コードの形式で生成できます。生成した XSL コードを使用して、最初の XML ファイルを 2 つ目の XML ファイルに変換できます。

注意： 現在、GUI 表示はカスタマイズできません。

XMLDiff のメソッド

この項では、XMLDiff Bean に含まれているメソッドについて説明します。

boolean diff()

2 つの XML ファイルまたは 2 つの XMLDocument オブジェクトの違いを検索します。

void domBuilderError(DOMBuilderEvent p0)

DOM パーサーによってのみコールされる DOMBuilderErrorListener インタフェースを実装します。

void domBuilderErrorCalled(DOMBuilderErrorEvent p0)

解析中にエラーが発生すると、DOM パーサーによってのみコールされる DOMBuilderErrorListener インタフェースを実装します。

void domBuilderOver(DOMBuilderEvent p0)

解析の完了時に、DOM パーサー・スレッドによってのみコールされる DOMBuilderListener インタフェースを実装します。

void domBuilderStarted(DOMBuilderEvent p0)

解析の開始時に、DOM パーサーによってのみコールされる DOMBuilderListener インタフェースを実装します。

boolean equals(Node node1, Node node2)

2 つのノードを比較します。differ アルゴリズムによってコールされます。このファンクションをオーバーライドして、カスタマイズされた比較を行うことができます。

XMLDocument generateXSLDoc()

XSLT スタイルシートを、2 つの XML 文書セットの違いを最初に表す XMLDocument として生成します。

void generateXSLFile(java.lang.String filename)

最初に設定された 2 つの XML ファイルの違いを表す、入力ファイル名を持つ XSL ファイルを生成します。

javax.swing.JTextPane getDiffPane1()

テキスト・パネルを、最初の XML ファイルの違いをビジュアル的に表示する JTextPane オブジェクトとして取得します。

javax.swing.JTextPane getDiffPane2()

テキスト・パネルを、2 つ目の XML ファイルまたは XML 文書の違いをビジュアル的に表示する JTextPane オブジェクトとして取得します。

XMLDocument getDocument1()

最初の XML ツリーの XMLDocument オブジェクトとして文書ルートを取得します。

XMLDocument getDocument2()

2 つ目の XML ツリーの XMLDocument オブジェクトとして文書ルートを取得します。

void printDiffTree(int tree, BufferedWriter out)

アルゴリズムによって違うと識別されたノード名および値を含む diff ツリーを出力します。このメソッドは、デバッグに有効です。

void setDocuments(XMLDocument doc1, XMLDocument doc2)

比較が必要な XML 文書を設定します。

void setFiles(java.io.File file1, java.io.File file2)

比較が必要な XML ファイルを設定します。

void setIndentIncr(int spaces)

XSL の生成にインデントを設定します。これは、generateXSLFile() メソッドまたは generateXSLDoc() メソッドの前にコールします。インデントは、すべての属性に適用されます。属性に加えて、新しく挿入されたノードを識別する場合は、10-34 ページの「void setNewNodeIndentIncr(int spaces)」を参照してください。

void setInput1(java.io.File file1)

比較が必要な最初の XML ファイルを設定します。

void setInput1(XMLDocument doc1)

比較が必要な最初の XML 文書を設定します。

void setInput2(java.io.File file2)

比較が必要な 2 つ目の XML ファイルを設定します。

void setInput2(XMLDocument doc2)

比較が必要な 2 つ目の XML 文書を設定します。

void setNewNodeIndentIncr(int spaces)

XSL の生成にインデントを設定します。これは、generateXSLFile() メソッドまたは generateXSLDoc() メソッドの前にコールします。インデントは、新しく挿入されたすべてのノードに適用されます（属性を除く）。属性のインデント・サポートについては、10-33 ページの「void setIndentIncr(int spaces)」を参照してください。

void setNoMoves()

diff アルゴリズムによって検出される移動が存在しないとします。このファンクションは、diff() ファンクションの前にコールします。このメソッドを使用すると、パフォーマンスが向上します。

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、DBAccess Bean および XMLDiff Bean が含まれています。

表 10-6 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 10-6 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。

表 10-6 Transviewer Bean のサンプル・ファイル（続き）

ファイル名	説明
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 10-38 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 10-44 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 10-48 ページの「Transviewer Bean の例 3: XMLTransformPanelSample.java」	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。XSLT 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。 ■ 10-49 ページの「Transviewer Bean の例 4a: DBViewer Bean - DBViewClaims.java」 ■ 10-52 ページの「Transviewer Bean の例 4b: DBViewer Bean - DBViewFrame.java」 ■ 10-53 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。
XMLDiffSample 次の例を参照してください。 ■ 10-53 ページの「XMLDiffSample.java」 ■ 10-58 ページの「XMLDiffFrame.java」	ユーザーが、任意の 2 つの XML ファイルを図形で比較できるサンプル・ビジュアル・アプリケーションです。2 つのファイルの違いは、XSLT コードとして表示できます。生成した XSLT を使用して、最初の XML ファイルを 2 つ目の XML ファイルに変換できます。

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。ORACLE_HOME を設定していない場合、インストールした XDK JavaBeans のルート・ディレクトリに設定します。
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**
   - **gmake sample6**
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。

サンプル・ファイル XMLDiffData1.txt および XMLDiffData2.txt を使用して、XMLDiff Bean のデモ・サンプル 6 をテストします。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、XSLT 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 Oracle9*i* から取得および格納し、XSLT 変換ファイルを Oracle9*i* 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。

- Oracle9*i* または Oracle9*i*AS 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。

Oracle9*i* 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

makefile 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。
- XSLT 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、XSLT 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、XSLT 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();
  }
}
```

XMLDiffSample.java

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

import java.io.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.net.URL;
import java.net.MalformedURLException;
```

```
public class XMLDiffSample
{
  /**
   * Constructor
   */
  public XMLDiffSample() {
  }

  /**
   * main
   * @param args
   */
  public static void main(String[] args)
  {

    dfxApp = new XMLDiffSample();
    diffFrame = new XMLDiffFrame(dfxApp);
    diffFrame.addTransformMenu();
    xmlDiff = new XMLDiff();

    if (args.length == 3)
      outFile = args[2];
    /* Use the default outFile name = XMLDiffSample.xsl */
    if(args.length >= 2)
      dfxApp.showDiffs(new File(args[0]), new File(args[1]));

    diffFrame.setVisible(true);
 }


  public void showDiffs(File file1, File file2)
  {
   try
   {
      xmlDiff.setFiles(file1, file2);

      /* Check if files are equal */
      if(!xmlDiff.diff())
      {
        JOptionPane.showMessageDialog(diffFrame,
          "Files are equivalent in XML representation",
          "XMLDiffSample Message",
          JOptionPane.PLAIN_MESSAGE);
      }

     /* generate xsl file */
    xmlDiff.generateXSLFile(outFile);
```

```
  /* parse the xsl file created, alternately you can use
     generateXSLDoc to get the xsl as a document tree instead of a file */
  parseXSL();
  /* Display the document trees created by the xmlDiff object */
  diffFrame.makeSrcPane(xmlDiff.getDocument1(), xmlDiff.getDocument2());
  diffFrame.makeDiffSrcPane(new XMLDiffSrcView(xmlDiff.getDiffPane1()),
                            new XMLDiffSrcView(xmlDiff.getDiffPane2()));
  diffFrame.makeXslPane(xslDoc, "Diff XSL Script");
  diffFrame.makeXslTabbedPane();
  }catch (FileNotFoundException e)
  {
    JOptionPane.showMessageDialog(diffFrame,
        "File Not Found: "+e.getMessage(),
        "XMLDiffSample Error Message",
        JOptionPane.ERROR_MESSAGE);
  }
  catch (Exception e)
  {
    e.printStackTrace();
    JOptionPane.showMessageDialog(diffFrame,
        "Error: "+e.getMessage(),
        "XMLDiffSample Error Message",
        JOptionPane.ERROR_MESSAGE);
  }
}

public void doXSLTransform()
{
  try
  {
    doc1 = xmlDiff.getDocument1();
    doc2 = xmlDiff.getDocument2();

    XSLProcessor xslProc = new XSLProcessor();

    /* Using the xsl stylesheet generated (xslDoc), transform the first file
      (doc1) into the second file (resultDocFrag) */
     XMLDocumentFragment resultDocFrag = xslProc.processXSL(new XSLStylesheet
                                    (xslDoc, createURL(outFile)), doc1);
    XMLDocument resultDoc = new XMLDocument();
    /* The XML declaration has to be copied over to the transformed XML doc,
       the xsl will not generate it automatically */
    if (doc1.getFirstChild() instanceof XMLDeclPI)
    if (doc1.getFirstChild() instanceof XMLDeclPI)
    {
       XMLNode xmldecl = (XMLNode) resultDoc.importNode(doc1.getFirstChild(),
                                                        false);
```

```
        resultDoc.appendChild(xmldecl);
      }
     /* Create the DTD node in the transformed XML document  */
      if(doc1.getDoctype() != null)
      {
        DTD dtd = (DTD)doc1.getDoctype();
        resultDoc.setDoctype(dtd.getName(), dtd.getSystemId(), dtd.getPublicId());
      }
      /* Create the result document tree from the document fragment */
      resultDoc.appendChild(resultDocFrag);
      diffFrame.makeResultFilePane(resultDoc);
    } catch (XSLException e)
    {
      e.printStackTrace();
      JOptionPane.showMessageDialog(diffFrame,
          "Error: "+e.getMessage(),
          "XMLDiffSample Error Message",
          JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception e)
    {
      e.printStackTrace();
      JOptionPane.showMessageDialog(diffFrame,
       "Error:"+e.getMessage(),
       "XMLDiffSample Error Message",
       JOptionPane.ERROR_MESSAGE);
    }
  }

  /* Parse the XSL file generated into a DOM tree */
  protected void parseXSL()
  {
    try
    {
      BufferedReader xslFile = new BufferedReader(new FileReader(outFile));
      DOMParser domParser = new DOMParser();
      domParser.parse(xslFile);
      xslDoc = domParser.getDocument();

    }catch (FileNotFoundException e)
    {
      JOptionPane.showMessageDialog(diffFrame,
          "File Not Found: "+e.getMessage(),
          "XMLDiffSample Message",
          JOptionPane.PLAIN_MESSAGE);
    }
    catch (Exception e)
```

```
  {
    JOptionPane.showMessageDialog(diffFrame,
     "Error:"+e.getMessage(),
     "XMLDiffSample Error Message",
     JOptionPane.ERROR_MESSAGE);
  }
}

// create a URL from a file name
protected 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();
          // to handle Windows platform
          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)
       {
        JOptionPane.showMessageDialog(diffFrame,
        "Cannot create url for: " + fileName,
        "XMLDiffSample Error Message",
        JOptionPane.ERROR_MESSAGE);

       }
    }
    return url;
}
```

```
  protected XMLDocument doc1;   /* DOM tree for first file */
  protected XMLDocument doc2;   /* DOME tree for second file */
  protected static XMLDiffFrame diffFrame; /* GUI frame */
  protected static XMLDiffSample dfxApp;   /* XMLDiff sample application */
  protected static XMLDiff xmlDiff;        /* XML diff object */
  protected static XMLDocument xslDoc;     /* parsed xsl file */
  protected static String outFile = new String("XMLDiffSample.xsl"); /* output
                                                                  xsl file name */
}
```

XMLDiffFrame.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

import oracle.xml.parser.v2.*;
import oracle.xml.srcviewer.*;
import oracle.xml.differ.*;
import org.w3c.dom.*;

public class XMLDiffFrame extends JFrame implements ActionListener {

  public XMLDiffFrame(XMLDiffSample dfApp)
  {
    super();
    mydfApp = dfApp;
    init();
  }


  public void makeSrcPane(XMLDocument doc1, XMLDocument doc2)
  {
      //undo srcviewer highlighting here
      XMLSourceView XmlSrcView1 = new XMLSourceView();
      XmlSrcView1.setXMLDocument(doc1);
      XmlSrcView1.setTagForeground(Color.black);
      XmlSrcView1.setAttributeValueForeground(Color.black);
      XmlSrcView1.setPIDataForeground(Color.black);
      XmlSrcView1.setCommentDataForeground(Color.black);
      XmlSrcView1.setCDATAForeground(Color.black);

      XmlSrcView1.setBackground(Color.lightGray);
      XmlSrcView1.getJTextPane().setBackground(Color.white);
```

```
        XmlSrcView1.add(new JLabel(filename1,SwingConstants.CENTER),
                        BorderLayout.NORTH);

        XMLSourceView XmlSrcView2 = new XMLSourceView();
        XmlSrcView2.setXMLDocument(doc2);
        XmlSrcView2.setTagForeground(Color.black);
        XmlSrcView2.setAttributeValueForeground(Color.black);
        XmlSrcView2.setPIDataForeground(Color.black);
        XmlSrcView2.setCommentDataForeground(Color.black);
        XmlSrcView2.setCDATAForeground(Color.black);

        XmlSrcView2.setBackground(Color.lightGray);
        XmlSrcView2.getJTextPane().setBackground(Color.white);
        XmlSrcView2.add(new JLabel(filename2,SwingConstants.CENTER),
                        BorderLayout.NORTH);

        XmlSrcView2.updateUI();
        XmlSrcView1.updateUI();

        srcPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                            XmlSrcView1, XmlSrcView2);
        srcPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);
        srcPane.setDividerLocation(0.5);
        srcPane.validate();


    }

    public void makeDiffSrcPane(XMLDiffSrcView srcView1, XMLDiffSrcView srcView2)
    {
        srcView1.setBackground(Color.lightGray);
        srcView2.setBackground(Color.lightGray);

        srcView1.add(new JLabel(filename1,SwingConstants.CENTER),BorderLayout.NORTH);
        srcView2.add(new JLabel(filename2,SwingConstants.CENTER),BorderLayout.NORTH);

        JScrollBar vscrollBar = srcView2.getScrollPane().getVerticalScrollBar();

        // make the diffSrcView divider fixed.
        srcView1.getScrollPane().setVerticalScrollBar(vscrollBar);
        srcView1.getScrollPane().setMinimumSize(
          new
Dimension(FRAMEWIDTH/2,srcView1.getScrollPane().getPreferredSize().height));
        srcView2.getScrollPane().setMinimumSize(
          new
Dimension(FRAMEWIDTH/2,srcView2.getScrollPane().getPreferredSize().height));
```

```
      srcView2.getScrollPane().updateUI();
      srcView1.getScrollPane().updateUI();

      srcView2.getTextPane().updateUI();
      srcView1.getTextPane().updateUI();

      srcView2.updateUI();
      srcView1.updateUI();

      diffSrcPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                           srcView1, srcView2);
      diffSrcPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);
      diffSrcPane.setDividerLocation(0.5);
      diffSrcPane.validate();

  }
  public void makeTabbedPane()
  {
    tabbedPane = new JTabbedPane();

    tabbedPane.addTab("SourceView", null , srcPane, "Source View of Files being
Diffed");
    tabbedPane.addTab("SourceDiffView", null , diffSrcPane, "Source View of File
Diffs");
    tabbedPane.addTab("TreeDiffView", null , diffTreePane, "DOM Tree View of File
Diffs");
    tabbedPane.setSelectedIndex(1);
    tabbedPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);

    this.getContentPane().add(tabbedPane);
    this.setVisible(true);


  }

  public void makeXslPane(XMLDocument doc, String title)
  {
     xslSrcView = new XMLSourceView();
     xslSrcView.setXMLDocument(doc);
     xslSrcView.setTagForeground(Color.black);
     xslSrcView.setAttributeValueForeground(Color.black);
     xslSrcView.setPIDataForeground(Color.black);
     xslSrcView.setCommentDataForeground(Color.black);
     xslSrcView.setCDATAForeground(Color.black);

     xslSrcView.setBackground(Color.lightGray);
     xslSrcView.getJTextPane().setBackground(Color.white);
```

```
    xslSrcView.add(new JLabel(title,SwingConstants.CENTER),
                   BorderLayout.NORTH);
    this.enableTransformItem(true);
  }


  public void makeResultFilePane(XMLDocument doc)
  {
    resultDoc = doc;
    XMLSourceView resultSrcView = new XMLSourceView();
    resultSrcView.setXMLDocument(doc);
    resultSrcView.setTagForeground(Color.black);
    resultSrcView.setAttributeValueForeground(Color.black);
    resultSrcView.setPIDataForeground(Color.black);
    resultSrcView.setCommentDataForeground(Color.black);
    resultSrcView.setCDATAForeground(Color.black);

    resultSrcView.setBackground(Color.lightGray);
    resultSrcView.getJTextPane().setBackground(Color.white);
    resultSrcView.add(new JLabel("XSLT Result File",SwingConstants.CENTER),
                   BorderLayout.NORTH);

    tabbedPane.addTab("ResultSourceView", null , resultSrcView,
                      "Source View of XSLT on File1");
    tabbedPane.setSelectedIndex(3);
    this.enableSaveAsItem(true);
  }

  public void makeXslTabbedPane()
  {
    tabbedPane = new JTabbedPane();

    tabbedPane.addTab("SourceView", null , srcPane, "Source View of XML Files being
Diffed");
    tabbedPane.addTab("SourceDiffView", null , diffSrcPane, "Source View of File
Diffs");
    tabbedPane.addTab("XSL Script",null,xslSrcView, "Source View of Diff XSL
script");
    tabbedPane.setSelectedIndex(2);
    tabbedPane.setSize(FRAMEWIDTH,FRAMEHEIGHT);

    this.getContentPane().add(tabbedPane);
    this.setVisible(true);


  }
```

```
public void actionPerformed(ActionEvent evt)
{
  File selectedFile1, selectedFile2;
  BufferedReader file1, file2;
  String arg, temp;

  if(evt.getSource() instanceof JMenuItem)
  {

    arg = evt.getActionCommand();

    if(arg.equals("Compare XML Files"))
    {
      JFileChooser jFC = new JFileChooser();
    jFC.setCurrentDirectory(new File("."));
    int retval = jFC.showOpenDialog(this);

    switch (retval)
    {

      case JFileChooser.APPROVE_OPTION:
          selectedFile1 = jFC.getSelectedFile();
        temp = selectedFile1.getName();
          jFC.cancelSelection();
          jFC.updateUI();
        switch(jFC.showOpenDialog(this))
        {
          case JFileChooser.APPROVE_OPTION:
            selectedFile2 = jFC.getSelectedFile();
           filename2 = selectedFile2.getName();
            filename1 = temp;

            this.getContentPane().removeAll();
            this.enableSaveAsItem(false);

           mydfApp.showDiffs(selectedFile1, selectedFile2);
            break;

            case JFileChooser.CANCEL_OPTION:
           break; //filename1 = null; // filename1 also null
          }// switch (jFC.showOpenDialog(this))
        break;

        case JFileChooser.CANCEL_OPTION:
        break;
      }
    }// if(arg.equals("Compare XML Files"))
```

```
    else if(arg.equals("Apply XSL to 1st Input File"))
    {
      mydfApp.doXSLTransform();

    }
    else if(arg.equals("Save As"))
    {
      JFileChooser jFC = new JFileChooser();
      jFC.setCurrentDirectory(new File("."));
      int retval = jFC.showOpenDialog(this);

    if (retval == JFileChooser.APPROVE_OPTION)
      {
        File file = jFC.getSelectedFile();
        try
        {
          resultDoc.print(new FileOutputStream(file));
        }catch (IOException e)
        {
           JOptionPane.showMessageDialog(this,
           "Error:"+e.getMessage(),
           "XMLDiffer Message",
           JOptionPane.PLAIN_MESSAGE);
        }


      }
    }
    else if(arg.equals("Exit"))
    {
      System.exit(0);
    }

  }
}


public void addTransformMenu()
{
  JMenuItem  item;

  JMenu jmenu = new JMenu("Transform");

  item = new JMenuItem("Apply XSL to 1st Input File");
  item.addActionListener(this);
  item.setEnabled(false);
  jmenu.add(item);
```

```
    this.getJMenuBar().add(jmenu);

  }

  protected void enableTransformItem(boolean flag)
  {
    this.getJMenuBar().getMenu(1).getItem(0).setEnabled(flag);
  }

  protected void enableSaveAsItem(boolean flag)
  {
    this.getJMenuBar().getMenu(0).getItem(1).setEnabled(flag);
  }

  private void init()
  {
    try
    {
      this.setTitle("XMLDiffer");
      this.getContentPane().setLayout( new
BoxLayout(this.getContentPane(),BoxLayout.Y_AXIS));
      // make the Differ window non-resizable
      this.setResizable(false);
      this.getContentPane().setBackground(SystemColor.control);
      addMenu();

      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = this.getSize();

      // set Frame size based on screen size such that there is room around it
      FRAMEWIDTH = screenSize.width - 100;
      FRAMEHEIGHT = screenSize.height - 200;
      this.setSize(new Dimension(FRAMEWIDTH, FRAMEHEIGHT));

      // put Differ window in the center of the screen
      this.setLocation((screenSize.width - FRAMEWIDTH)/2, (screenSize.height -
FRAMEHEIGHT)/2);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) { System.exit(0); }});

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
```

```
  private void addMenu()
  {
    JMenuItem  item;

    JMenuBar jmenubar = new JMenuBar();
    JMenu jmenu = new JMenu("File");

    item = new JMenuItem("Compare XML Files");
    item.addActionListener(this);
    jmenu.add(item);

    item = new JMenuItem("Save As");
    item.addActionListener(this);
    item.setEnabled(false);
    jmenu.add(item);

    jmenu.addSeparator();

    item = new JMenuItem("Exit");
    item.addActionListener(this);
    jmenu.add(item);

    jmenubar.add(jmenu);
    this.setJMenuBar(jmenubar);

  }


  protected static int LEFT_TOP = 0;
  protected static int RIGHT_TOP = 1;
  protected static int CENTER = 2;

  private int FRAMEWIDTH =0;
  private int FRAMEHEIGHT =0;

  private XMLDocument resultDoc;
  private XMLSourceView xslSrcView;
  private XMLDiffSample mydfApp;
  private String filename1, filename2;
  private JTabbedPane tabbedPane;
  private JSplitPane diffTreePane, srcPane,diffSrcPane;
}
```

11

XDK および SOAP の使用

この章の内容は次のとおりです。

SOAP の概要

「Web サービス」という用語は、Web 上のエンティティによって利用可能な機能を意味します。Web サービスは、XML 標準を使用するアプリケーションで、Web を介して公開、検索および実行されます。

Simple Object Access Protocol（SOAP）は、リクエストおよびレスポンスをインターネット経由で送受信するためのプロトコルです。このプロトコルは XML および簡易転送プロトコル（HTTP など）に基づくもので、ファイアウォールによってブロックされず、簡単に使用できます。SOAP はオペレーティング・システム、実装言語および単一のオブジェクト・モデルに依存しません。

SOAP は、リモート・プロシージャ・コールをサポートします。SOAP によるメッセージは、次の 3 種類のいずれかです。

- サービスのリクエスト（入力パラメータを含む）
- リクエストされたサービスへのレスポンス（戻り値および出力パラメータを含む）
- エラー・コードおよびエラー情報を含む Fault

SOAP メッセージは、次の内容で構成されます。

- エンベロープ : メッセージを含み、その処理方法、処理者、およびその処理がオプションか必須かを指定します。
- エンコーディング規則 : アプリケーションで使用するデータ型を記述します。これらの規則では、アプリケーションのデータ型を XML へ、また XML をデータ型へ変換するシリアル化メカニズムが定義されます。
- リモート・プロシージャ・コール定義

SOAP 1.1 仕様は W3C ノートです。（W3C XML プロトコル・ワーキング・グループは、SOAP の後継となる規格を作成するために結成されました）。

SOAP は、転送プロトコルにもオペレーティング・システムにも依存しません。SOAP は、すべてのアプリケーションで標準の XML メッセージ形式を提供します。SOAP は、World Wide Web Consortium（W3C）の W3C XML Schema 仕様を使用しています。

> **参照 :** 次の Web サイトを参照してください。
>
> - `http://www.w3.org/TR/SOAP/`
> - `http://xml.apache.org/soap`

SOAP サービスのリモート・プロシージャ・コール（RPC）・リクエストおよびレスポンスの手順は、次のとおりです。

1. SOAP クライアントが、エディタまたは Oracle SOAP クライアント API を使用して、準拠する XML 文書にサービスへのリクエストを書き込みます。

2. SOAP クライアントは、サービスへのリクエストが書き込まれた文書を Web サーバーでサーブレットとして実行中の SOAP リクエスト・ハンドラに送信します。
3. Web サーバーはリクエストされたサービスを提供しながら、そのメッセージをサービス・リクエストとして適切なサーバー側アプリケーションにディスパッチします。
4. サポートされる部分がメッセージに含まれることを、アプリケーションで確認する必要があります。サービスからのレスポンスは、SOAP リクエスト・ハンドラ・サーブレットに戻された後、SOAP ペイロード形式を使用してコール側に戻されます。

UDDI および WSDL の概要

Universal Description, Discovery and Integration（UDDI）仕様では、プラットフォームに依存しないフレームワークを提供しています。このフレームワークでは、XML を使用してインターネット上でサービスを記述し、ビジネスを検出し、ビジネス・サービスを統合します。UDDI ビジネス・レジストリは、企業が登録されている公開データベースです。UDDI ビジネス登録は、XML ファイルで、次の 3 つのセクションを含みます。

- ホワイトページ : 住所、連絡先および既知の識別子を含みます。
- イエローページ : 業種分類を含みます。
- グリーンページ : 公開されているサービスについての技術情報を含みます。

Web Services Description Language（WSDL）は、インタフェースの記述、プロトコル・バインディングおよび Web サービスの詳細な配置のための汎用 XML 言語です。WSDL は、抽象インタフェースおよび任意のネットワーク・サービスを記述するメソッドを提供します。WSDL サービスは、UDDI レジストリに登録されるか、または埋め込まれます。

Web サービスで使用されるプロトコルのスタックを、次の表にまとめて示します。

プロトコル・スタック
ユニバーサル・サービス相互運用プロトコル（WSDL など）
Universal Description, Discovery Integration（UDDI）
Simple Object Access Protocol（SOAP）
XML、XML Schema
インターネット・プロトコル（HTTP、HTTPS、TCP/IP）

Oracle SOAP の概要

Oracle SOAP は、Simple Object Access Protocol の実装です。Oracle SOAP は、Apache Software Foundation によって開発された SOAP オープン・ソース実装に基づいています。

SOAP の動作

次の例について考えてみます。StockQuote サービスに GetLastTradePrice SOAP リクエストが送信されました。このリクエストは企業の銘柄記号である文字列パラメータを取り、SOAP レスポンスで float 型を戻します。XML 文書は SOAP メッセージを表します。SOAP エンベロープ要素は、XML 文書の最上位要素です。XML 名前空間は、SOAP 識別子とアプリケーション固有の識別子との区別に使用します。次の例では、HTTP を転送プロトコルとして使用します。SOAP の XML ペイロード形式を決定する規則は、このペイロードが HTTP で転送されるということには関係ありません。HTTP リクエストに埋め込まれる SOAP リクエスト・メッセージは、次のとおりです。

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope  xmlns:SOAP-   ENV="http://schemas.xmlsoap.org/soap/
envelope/"  SOAP-
ENV:encodingStyle="http://schemas.xnlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>ORCL</symbol>
<m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

レスポンスの HTTP メッセージは次のとおりです。

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope  xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap//envelope/ SOAP-
ENV:encodingStyle="http://schemas.xnlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP クライアントの概要

SOAP クライアント・アプリケーションとは、SOAP リクエストを行うユーザー作成アプリケーションを意味します。SOAP クライアントの機能は、次のとおりです。

- サービスの起動にすべての必要なパラメータを収集します。
- SOAP サービス・リクエスト・メッセージを作成します。これは、SOAP プロトコルに従って作成される XML メッセージで、XML でエンコードされたすべての入力パラメータのすべての値を含みます。このプロセスをシリアル化といいます。
- SOAP サーバーがサポートする任意の転送プロトコルを使用してリクエストを SOAP サーバーに送信します。
- SOAP レスポンス・メッセージを受信します。
- SOAP Fault 要素を処理して、リクエストの成否を判断します。
- XML から戻されたパラメータをネイティブなデータ型に変換します。このプロセスを非シリアル化といいます。
- 必要に応じて結果を使用します。

SOAP クライアント API

SOAP クライアントは、SOAP サービスへのリクエストを構成する XML 文書を生成し、SOAP レスポンスを処理します。Oracle SOAP は、有効な SOAP リクエストを送信するすべてのクライアントからのリクエストを処理します。クライアントが簡単に開発できるように、Oracle SOAP には、SOAP サービスを起動する一般的な方法を提供する SOAP クライアント API が含まれています。

SOAP クライアント API は、リクエストおよびレスポンス用の同期式起動モデルをサポートします。SOAP クライアント API によって、SOAP リクエストを行う Java クライアント・アプリケーションを簡単に作成できます。SOAP クライアント API は、SOAP リクエストの作成および基礎となる転送プロトコルを使用したリクエスト送信の詳細をカプセル化します。また、SOAP クライアント API は、交換可能な転送もサポートし、クライアントは転送を簡単に変更できます（HTTP および HTTPS による転送が可能です）。

SOAP サーバーの概要

SOAP サーバーの機能は、次のとおりです。

- サービス・リクエストを受信します。
- XML リクエストの解析後、メッセージを実行するか、拒否するかを決定します。
- メッセージを実行する場合、リクエストされたサービスが存在するかどうかを判断します。
- すべての入力パラメータを XML からサービスが認識できるデータ型に変換します。
- サービスを起動します。
- 戻りパラメータを XML に変換し、SOAP レスポンス・メッセージを生成します。
- レスポンス・メッセージをコール元に戻します。

Oracle SOAP のセキュリティ機能

Oracle SOAP は、転送時に、セキュリティ機能を使用して安全なアクセスをサポートし、他のセキュリティ機能もサポートします。たとえば、Oracle SOAP は、HTTPS を使用して、Secure Sockets Layer（SSL）での機密保護、認証および整合性を提供します。ロギング、認可などのその他のセキュリティ機能は、サービス・プロバイダから提供されます。

SOAP トランスポート

SOAP トランスポートは、SOAP メッセージを搬送するプロトコルです。Oracle SOAP は、次の転送をサポートします。

- HTTP: このプロトコルは、基本的な SOAP トランスポートです。Oracle SOAP リクエスト・ハンドラ・サーブレットは、HTTP リクエストを管理し、HTTP 上でレスポンスを直接行います。このプロトコルは、普及したため、標準になりました。
- HTTPS: Oracle SOAP リクエスト・ハンドラ・サーブレットは、HTTPS リクエストを管理し、サポートされている様々なセキュリティ・レベルでレスポンスを行います。

管理クライアント

SOAP 管理クライアントには、サービス・マネージャおよびプロバイダ・マネージャが含まれます。これらの管理クライアントは、新しいサービスおよび新しいプロバイダの動的配置をサポートするサービスです。

SOAP リクエスト・ハンドラ

SOAP リクエスト・ハンドラは、SOAP リクエストを受信し、適切なサービス・プロバイダを検索し、リクエストされたメソッド（サービス）を起動するサービス・プロバイダを処理して、SOAP レスポンスが存在する場合はそれを戻す Java サーブレットです。

SOAP プロバイダ・インタフェースとプロバイダ

Oracle SOAP には、Java クラスのプロバイダ実装が含まれており、他のプロバイダを追加できます。

プロバイダ・インタフェース

プロバイダ・インタフェースを使用すると、SOAP サーバーでプロバイダのタイプ（Java クラス、ストアド・プロシージャなど）に関係なくサービス・メソッドを一定の方法で起動できます。サービス・プロバイダのタイプごとに、1 つのプロバイダ・インタフェース実装があり、すべてのプロバイダ固有情報がカプセル化されます。プロバイダ・インタフェースでは、新しいタイプのサービス・プロバイダをサポートできるように、SOAP 実装を簡単に拡張することができます。

プロバイダ配置管理

Oracle SOAP は、プロバイダ配置管理クライアントを提供して、プロバイダ配置情報を管理します。

SOAP サービス

SOAP サービスは、SOAP アプリケーションの開発者によって提供されます。これらのサービスは、供給されるデフォルトの Java クラス・プロバイダまたはカスタム・プロバイダを使用すると利用できます。Oracle SOAP には、サービスを管理するサービスとして実行する、サービス配置管理クライアントが含まれます。SOAP サービス（Java サービスを含む）とは、リモート SOAP クライアントに提供されるユーザー作成アプリケーションを意味します。

JDeveloper による SOAP のサポート

Oracle9*i* JDeveloper は、WSDL、SOAP および UDDI をサポートします。

参照： 第 23 章「JDeveloper を使用した XML アプリケーションの開発」を参照してください。

開発者ガイドの参照

次に、『Oracle9*i* Application Server Oracle9*i*AS SOAP 開発者ガイド リリース 1.0.2.2』（2001 年 10 月、J04239-01）をオンラインで検索する方法を示します。

1. `http://otn.oracle.co.jp/document/products/9ias/index.html` にアクセスします。
2. 「R1.0.2.2 - プラットフォーム共通」を開きます。

参照： Oracle SOAP および Web サービスのドキュメントおよびダウンロードの詳細は、次のマニュアルおよび Web サイトを参照してください。

- http://otn.oracle.com/products/ias/daily/sept07.html
- Advanced Queueing（AQ）へのインターネット・アクセスについての詳細は、『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。
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- SOAP API は、プロダクト CD ディスク 1 のファイル `doc/readmes/ADDEN_rdbms.htm` にあります。

12

Oracle TransX Utility

この章の内容は次のとおりです。

TransX Utility の概要

TransX Utility を使用すると、翻訳済のシード・データおよびメッセージをデータベースに簡単にロードできます。また、次の機能によって、国際化のコストを削減できます。

- 翻訳する文字列の準備
- 文字列の翻訳
- データベースへの文字列のロード

TransX Utility を使用すると、翻訳データ形式エラーが最小になり、データベース内の事前に指定された場所に翻訳内容が正確にロードされます。TransX Utility には、その他に次のメリットがあります。

- 翻訳ベンダーは、慣れていない SQL および PL/SQL スクリプトを操作する必要がありません。
- 様々なグローバリゼーション・サポートの設定による構文エラーが発生しません。
- NCHAR データのピース単位の UNISTR コンストラクタが不要になります。

開発グループで、翻訳済のメッセージおよびシード・データをロードする必要がある場合、TransX Utility を使用すると、国際化の要件を満たすために必要な作業を簡単に行えます。データが事前定義済の形式になると、TransX Utility はその形式を検証します。

TransX によるロードでは、XML がエンコーディングを記述しているメリットとして、翻訳済のデータのロード時に正しいエンコーディングが自動的に選択されます。このため、データ・ファイルが XML 標準に準拠しているかぎり、不適切なエンコーディングによるロード時のエラーは発生しません。

TransX Utility の主な機能

この項では、TransX Utility が持つ次の機能について説明します。

- 多言語データ・ロードの簡略化
- 簡易データ形式のサポートおよびインタフェース
- 標準 XML 形式でのデータセットのロード
- 既存データの処理
- その他の TransX Utility 機能

多言語データ・ロードの簡略化

従来、一般的な翻訳データのロード方法では、ロードするファイルの切替えと同時に NLS_LANG の設定を切り替えていました。各ロード・ファイルは、特定の言語に適した特定のキャラクタ・セットでエンコードされます。翻訳は、元のファイルと同じファイル形式（通常 .sql スクリプト）で行う必要があるため、この操作が必要でした。

NLS_LANG の設定は、ファイルが言語に対応したキャラクタ・セットに適した形でロードされると同時に変更されます。TransX Utility ロード・ツールを使用すると、開発および翻訳グループは、XML を使用してデータをデータベースにロードするまでの間、正しいキャラクタ・セットを保持する必要がなくなります。

簡易データ形式のサポートおよびインタフェース

TransX Utility データ・ロード・ツールは、データベースにロードされるシード・データを表す正規の方法として定義されたデータ形式に準拠します。この形式は、直観的で、簡単に理解可能な形式です。また、翻訳グループ用に簡略化されています。フォーマット仕様は、データのロードを可能にするための、翻訳者によるデータの記述方法を定義します。

このデータ・ロード・ツールは、コマンドライン・インタフェースおよびプログラム可能な API を持ちます。どちらも簡単で、短時間で使用できるようになります。

標準 XML 形式でのデータセットのロード

正規の形式でデータセットを指定すると、そのデータは、TransX Utility によってデータベース内の指定された場所にロードされます。ただし、オブジェクト（データのロード先となる表も含む）は作成されません。ロードするデータの記述には、XML で表されるリテラル値以外に、次の式を使用できます。

定数式 定数式を使用すると、定数値を指定できます。各行に固定値を持つ列では、同じ値を繰り返す必要はありません。

順序 データベース内の順序から取得した値を使用して列をロードできます。

問合せ SQL 問合せを使用して列をロードできます。問合せでは、パラメータ（複数可）が使用できます。

既存データの処理

データ・ロード・ツールは、データベースに重複行が存在するかどうかを判断します。重複行の処理方法は、次のオプションから 1 つ選択できます。検索キーとして指定されたすべての列の値が同じ場合、行が重複しているとみなされます。処理オプションは、次のとおりです。

- 重複行をスキップするか、そのままにしておく（デフォルト）
- 指定されたデータセットのデータを使用して、重複行を更新またはオーバーライドする
- エラーを表示する

その他の TransX Utility 機能

次に、その他の TransX Utility 機能を説明します。

- コマンドライン・インタフェース : データ・ロード・ツールには、簡単に使用可能なコマンドが用意されています。
- ユーザー API: データ・ロード・ツールでは、Java API が表示されます。
- 検証 : データ・ロード・ツールは、データ形式を検証し、エラーをレポートします。
- 空白の処理 : データセットに含まれる空白文字は、特に指定されていないかぎり、重要ではありません。
- アンロード : データ・ロード・ツールは、問合せに基づいて、結果を標準データ形式にエクスポートします。
- 翻訳交換フォーマットとの親和性 : 翻訳交換フォーマットとの変換を行うように設計されています。
- ローカライズされたユーザー・インタフェース : メッセージが *N* 通りの言語で提供されています。

TransX Utility のインストール

この項では、TransX のインストール方法および TransX の依存性について説明します。

TransX の依存性

Oracle TransX Utility が機能するには、次のコンポーネントが必要です。

- Database Connectivity: JDBC ドライバ。このユーティリティは、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 TransX に対しては保証およびサポートを行いません。

- XML Parser: Oracle XML Parser バージョン 2。Oracle XML Parser は、Oracle8*i* および Oracle9*i* に含まれています。また、Oracle Technology Network（OTN）の Web サイトからダウンロードすることもできます。
- XML Schema プロセッサ : Oracle XML Schema プロセッサ。Oracle XML Schema プロセッサは、Oracle8*i* および Oracle9*i* に含まれています。また、Oracle Technology Network（OTN）の Web サイトからダウンロードすることもできます。
- XML SQL Utility: Oracle XML SQL Utility（XSU）。Oracle XSU は、Oracle8*i* および Oracle9*i* に含まれています。また、Oracle Technology Network（OTN）の Web サイトからも入手できます。

Oracle Installer を使用した TransX のインストール

TransX は、Oracle9*i* にパッケージ化されています。TransX ユーティリティは、次の 3 つの実行可能ファイルで構成されています。

- $ORACLE_HOME/rdbms/jlib/transx.zip: TransX を構成するすべての Java クラスを含みます。
- $ORACLE_HOME/rdbms/bin/transx: UNIX コマンドラインから TransX を起動するシェル・スクリプトです。
- $ORACLE_HOME/rdbms/bin/transx.bat: Windows コマンドラインから TransX を起動するバッチ・ファイルです。

Oracle9*i* Installer は、デフォルトで TransX 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。

OTN からダウンロードした TransX のインストール

Oracle Technology Network の Web サイト（http://otn.oracle.com）から、XDK for Java の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。

TransX のフロントエンドまたは Java API を使用するために必要な手順

`env.xxx` スクリプトを使用して、環境（CLASSPATH）を設定します。このスクリプトは、ダウンロードした XDK のアーカイブを解凍して作成されたディレクトリの下の bin ディレクトリに存在します。

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は、そのシェルの構文を使用でき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 ファイルを実行します。

TransX Utility コマンドラインの構文

次に、TransX Utility のコマンドラインの構文を示します。

```
java oracle.xml.transx.loader [options] connect_string username password datasource
[datasource]
java oracle.xml.transx.loader -v datasource [datasource]
java oracle.xml.transx.loader -x connect_string username password table [column]
java oracle.xml.transx.loader -s connect_string username password filename table
[column]
```

TransX Utility コマンドラインの例

次に、TransX Utility のコマンドラインの例を示します。

```
java oracle.xml.transx.loader "dlsun9999:1521:mydb" scott tiger foo.xml
java oracle.xml.transx.loader "jdbc:oracle:oci:@mydb" scott tiger foo.xml
java oracle.xml.transx.loader -v foo.xml
java oracle.xml.transx.loader -x "dlsun9999:1521:mydb" scott tiger emp
java oracle.xml.transx.loader -s "dlsun9999:1521:mydb" scott tiger emp.xml emp ename
job
```

TransX Utility コマンドライン・パラメータ

表 12-1 に、コマンドライン・パラメータを示します。

表 12-1 TransX Utility コマンドライン・パラメータ

パラメータ	意味
connect_string	JDBC 接続文字列。「@」記号を使用すると接続文字列情報を省略できます。この箇所に「jdbc:oracle:thin:@」が指定されます。
username	データベース・ユーザー名。
password	データベース・ユーザーのパスワード。
datasource	XML データ・ソース。
option	表 12-2「TransX Utility のコマンドライン・オプション」を参照してください。

TransX Utility のコマンドライン・オプション

表 12-2 TransX Utility のコマンドライン・オプション

オプション	意味	説明
-u	既存の行を更新します。	このオプションを指定すると、既存の行をスキップせず、更新します。更新操作の対象から列を除外するには、useforupdate 属性を「no」に指定します。
-e	行がデータベースにすでに存在する場合に例外を戻します。	このオプションを指定すると、重複行が検出されると例外を発生させます。デフォルトでは、単に重複行をスキップします。データベースおよびデータセットの検索キー列（複数可）の値が同じ場合は、行が重複しているとみなします。
-x	データベースのデータを、事前定義済の形式 * で出力します。	-s オプションと同様に、TransX がロードと逆の操作を実行します。ただし、-s オプションとは異なり、stdout に出力を行います。 **注意 :** この出力をファイルにリダイレクトする方法はお薦めしません。オペレーティング・システムの介入によって、予想外のトランスコーディングが発生してデータが失われる可能性があります。
-s	データベースのデータを、事前定義済の形式 * でファイルに保存します。	これはアンロードを実行するためのオプションです。データベースへの問合せを行って、その結果を事前定義済の XML 形式にフォーマットし、指定したファイル名で格納します。
-p	ロードする XML を出力します。	挿入するデータセットを正規の XSU 形式で出力します。
-t	更新のため XML を出力します。	更新するデータセットを正規の XSU 形式で出力します。
-o	検証を省略します（デフォルトでは、データは解析時に検証されます）。	TransX が形式の検証（デフォルトでは実行される）をスキップします。
-v	データ形式を検証し、ロードせずに終了します。	TransX が検証を行って終了します。
-w	すべての空白を保持します。	TransX が空白文字（\t、\r、\n、' ' など）を重要であるとみなします。文字列データ要素内の連続する空白文字は、デフォルトでは 1 つの空白文字とみなされます。

コマンドライン・オプションの例外 次に、コマンドライン・オプションの例外を示します。

- -u および -e は、相互に排他的です。
- -v は、次の例に示すとおり、後にデータを伴う唯一のオプションです。
- -x は、次の例に示すとおり、後に接続情報および SQL 問合せを伴う唯一のオプションです。

すべての引数を省略すると、フロントエンド使用情報が表に示されます。

> **参照：** Java API for TransX Utility の詳細は、『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

TransX Utility のサンプル・コード

次に、TransX Utility のサンプル・コードを示します。

```
String  datasrc[] = {"data1.xml", "data2.xml", "data3.xml"};

// instantiate a loader
TransX  transx = loader.getLoader();

// start a data loading session
transx.open( jdbc_con_str, usr, pwd );

// specify operation modes
transx.setLoadingMode( LoadingMode.SKIP_DUPLICATES );
transx.setValidationMode( false );

// load the dataset(s)
for ( int i = 0 ; i < datasrc.length ; i++ )
{
transx.load( datasrc[i] );
}

// cleanup
transx.close();
```

第 III 部

XDK for C/C++

第 III 部では、XML Developer's Kit（XDK）for C++ の入手方法および使用方法を説明します。

13

XML Parser for C

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 Oracle9*i* および Oracle9*i*AS 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、Solaris オペレーティング環境システムでは `$ORACLE_HOME/xdk/demo/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/tech/xml` の「XML Discussion Forum」にポストできます。

仕様

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 API リファレンス - XDK および Oracle XML DB』
- `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。

- `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 13-1 に、XML Parser for C で使用するデータ型を示します。

表 13-1 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	ネイティブな符号なし整数

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/xdk/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

参照： 使用可能な検証モードの詳細は、4-5 ページの「Oracle XML Parser の検証モード」を参照してください。

XML Parser for C の使用方法

図 13-1 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 `xmlterm()` をコールします。
   - 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. `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 13-1 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

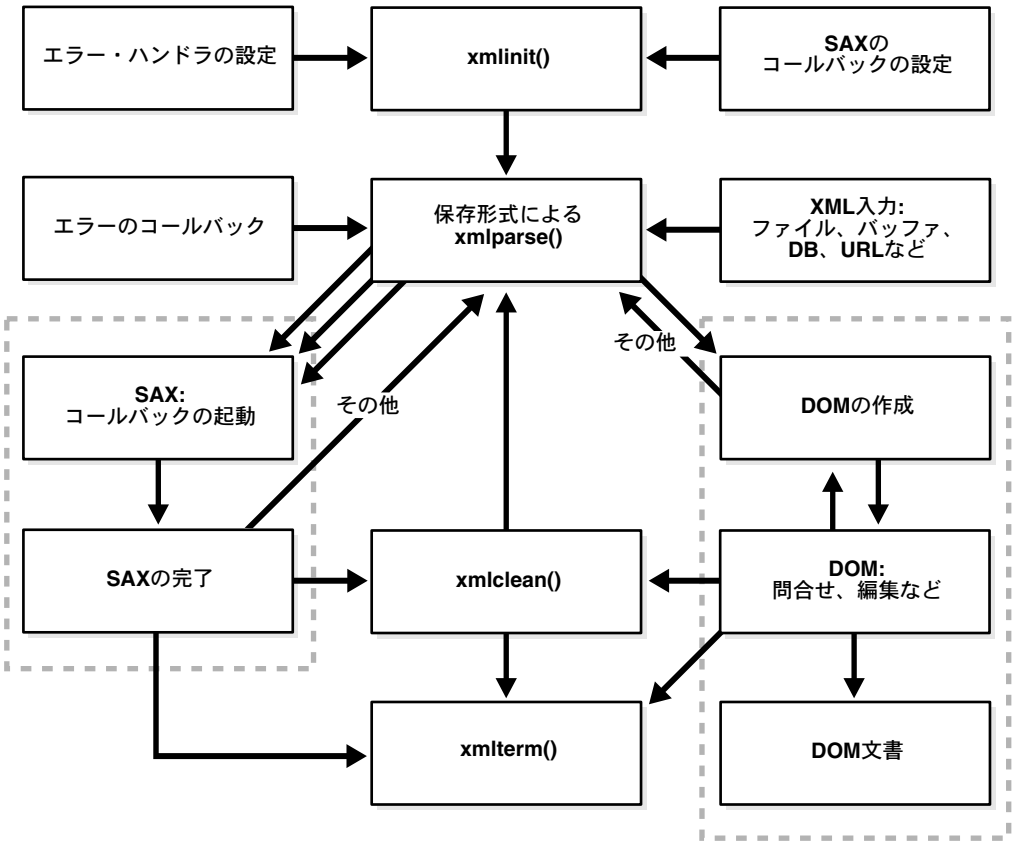
XMLParser に対するコール順序は、次のいずれかになります。

- `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 13-1 XML Parser for C のコール順序



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

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、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;
```

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 13-2 に、コマンドライン・オプションを示します。

表 13-2 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、提供される API を使用するように C コードを記述することによっ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。プログラムの作成方法の詳細は、demo/c/parser 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/demo/c/parser/ 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 13-3 に、サンプル・ファイルを示します。

表 13-3 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
Makefile	サンプル・プログラムを作成するための Makefile
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	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

サンプル・ディレクトリ（Solaris オペレーティング環境の `$ORACLE_HOME/xdk/demo/c/parser`）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 13-4 に、サンプル・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 13-4 XML Parser for C: サンプル・ファイルによって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>	XSLT プロセッサの使用例。入力として、XML ファイルおよび XSLT スタイルシートの 2 つのファイル名を取ります。

14

XSLT Processor for C

この章の内容は次のとおりです。

XSLT for C の入手方法

XSLT for C は Oracle9*i* および Oracle9*i*AS 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。

XSLT for C は、$ORACLE_HOME/xdk/demo/c/parser にあります。

XSLT 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。

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

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 API リファレンス - XDK および Oracle XML DB』
- http://otn.oracle.com/tech/xml/

XSLT for C（DOM インタフェース）の使用方法

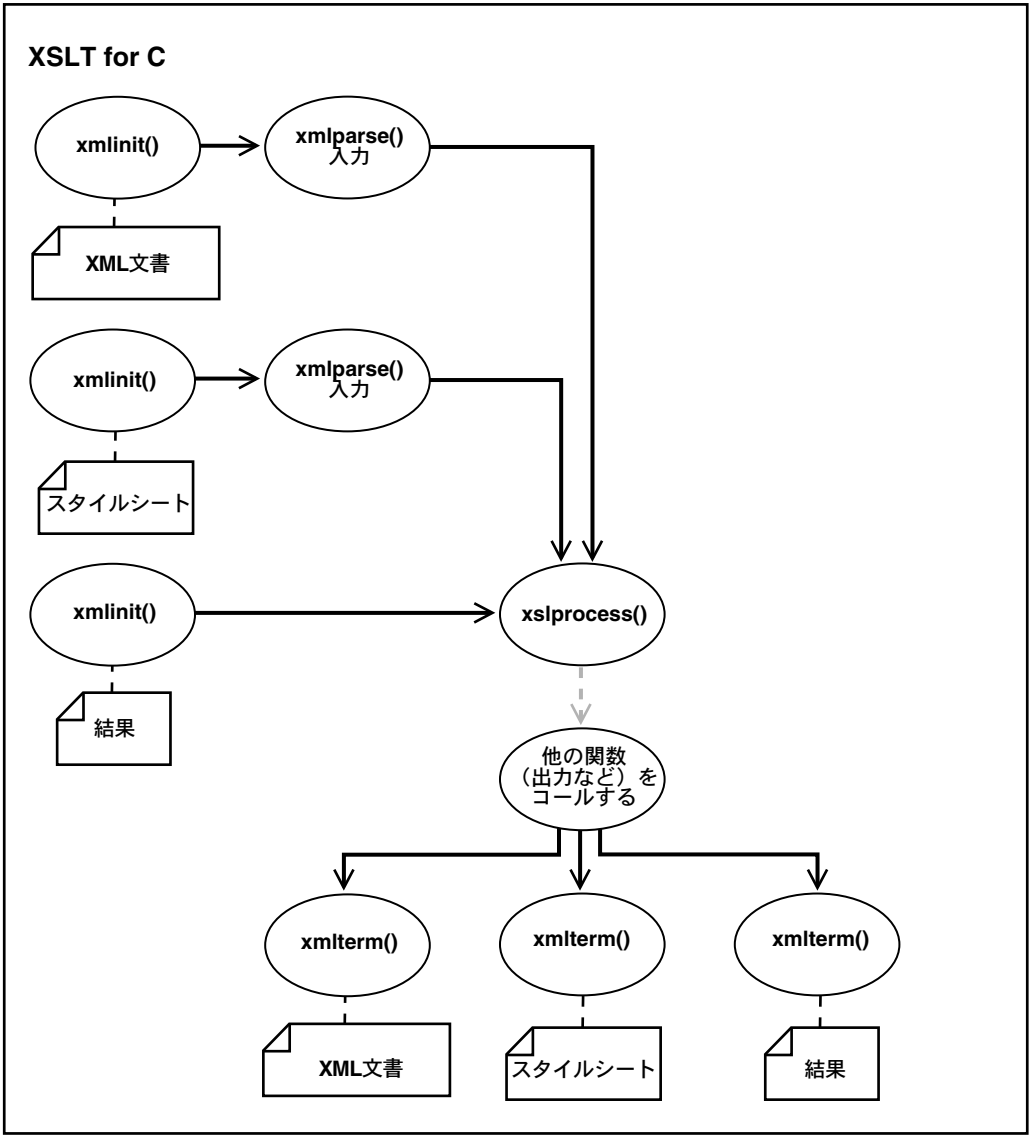
図 14-1 に、XSLT for C の機能を示します。

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
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 API リファレンス - XDK および Oracle XML DB』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、`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。

- 14-7 ページの「XSLT for C の例 2: C - XSLSample.c」
- 14-9 ページの「XSLT for C の例 3: C - XSLSample.std」

図 14-1 XSLT for C（DOM インタフェース）の使用方法



XSLT for C の起動

XSLT 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

XSLT 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 14-1 に、コマンドライン・オプションを示します。

表 14-1 XML Parser for C: コマンドライン・オプション

オプション	説明
-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。
-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。
-s	スタイルシート

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/demo/c/parser/` ディレクトリには、XSLT 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 14-2 に、サンプル・ファイルを示します。

表 14-2 XSLT for C のサンプル・ファイル

サンプル・ファイル名	説明
XSLSample.c	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

XSLT 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

サンプル・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 14-3 に、サンプル・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 14-3 XSLT for C: サンプル・ファイルによって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
XSLSample <xmlfile> <xsl ss>	XSLT プロセッサの使用例。入力として、XML ファイルおよび XSLT スタイルシートの 2 つのファイル名を取ります。

XSLT for C の例 1: 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>
```

XSLT for C の例 2: 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;
}
```

XSLT for C の例 3: 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>
```

15

XML Schema Processor for C

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、Oracle9*i* 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、W3C の XML Schema 勧告をサポートしています。このため、Oracle9*i* 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、Oracle9*i* 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 プロセッサが Oracle9*i* 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 4 章「XML Parser for Java」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 勧告をサポートします。

> **参照：** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

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
- W3C の XML Schema 勧告

XML Schema Processor for C: 提供されるソフトウェア

表 15-1 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 15-1 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
xdk/license.html	ライセンス契約
bin	スキーマ・プロセッサが実行可能なスキーマ
xdk/doc	API ドキュメント
xdk/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
xdk/mesg	エラー・メッセージ・ファイル
xdk/demo/c/schema	スキーマ・プロセッサの使用例

表 15-2 に、Solaris オペレーティング・システムに含まれるライブラリを示します。

表 15-2 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
libxml9.a	XML パーサー /XSLT プロセッサ
libxsd9.a	XML Schema プロセッサ
libcore9.a	CORE ファンクション
libnls9.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、xdk/demo/c/schema 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。

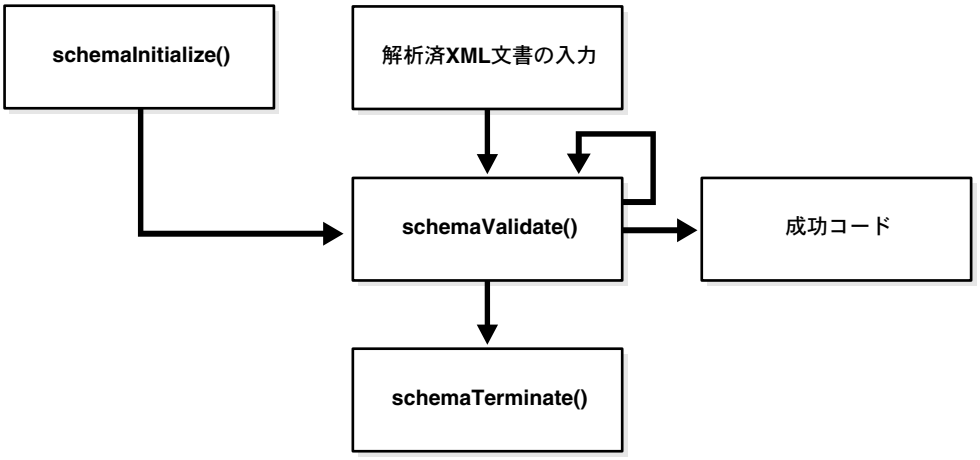
XML Schema Processor for C の使用方法

図 15-1 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 15-1 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

サンプル・ディレクトリには、API による Oracle XML Schema プロセッサ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 15-3 に、提供されるサンプル・ファイルを示します。

表 15-3 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
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
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

サンプル・プログラムを作成するには、make コマンドを実行します。

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 コマンドを実行します。

16

XML Parser for C++

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 Oracle9*i* および Oracle9*i*AS 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/demo/cpp/parser にあります。

XML Parser for C++ の機能

ドキュメントのルート・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/tech/xml の「XML Discussion Forum」にポストできます。

仕様

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 xdk/doc/cpp ディレクトリ
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- 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。

- 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 16-1 に、XML Parser for C++ で使用するデータ型を示します。

表 16-1 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	ネイティブな符号なし整数

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、xdk/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/xdk/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

参照： 使用可能な検証モードの詳細は、4-5 ページの「Oracle XML Parser の検証モード」を参照してください。

XML Parser for C++ の使用方法

図 16-1 に、XML Parser for C++ の機能を示します。

1. xmlinit() 関数によって、解析プロセスが初期化されます。
2. 入力は、XML 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。
   - 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. `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.xmlparseBuffer()` - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparseBuffer()` - `XMLParser.xmlclean()` - `XMLParser.xmlparse()` または

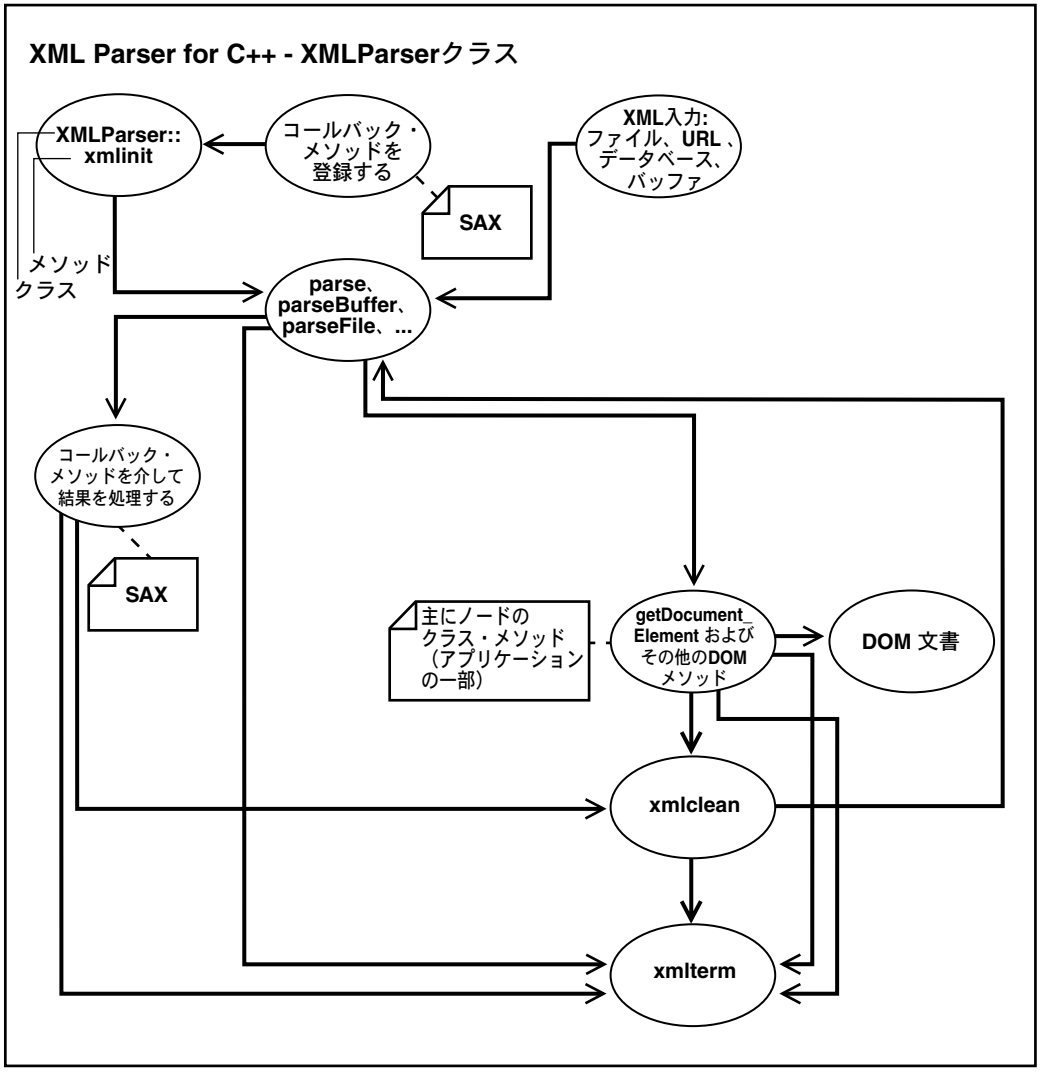
  `XMLParser.xmlparseBuffer()` - `XMLParser.xmlclean()` -... - `XMLParser.xmlterm()`

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparseBuffer()` - `XMLParser.xmlparse()` または

  `XMLParser.xmlparseBuffer()` -... - `XMLParser.xmlterm()`

図 16-1 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;
```

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 16-2 に、コマンドライン・オプションを示します。

表 16-2 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、提供される API を使用するようにコードを記述することによって起動することもできます。コードは、xdk/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、xdk/demo/cpp/parser 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/demo/cpp/parser/ 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 16-3 に、サンプル・ファイルを示します。

表 16-3 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.cpp	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

サンプル・ディレクトリ（Solaris オペレーティング環境の `$ORACLE_HOME/xdk/demo/cpp/parser`）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 16-4 に、サンプル・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 16-4 XML Parser for C++: サンプル・ファイルによって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>	XSLT プロセッサの使用例。入力として、XML ファイルおよび XSLT 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と、環境によっては出力の一部が欠落する場合があります。

17

XSLT Processor for C++

この章の内容は次のとおりです。

XSLT for C++ の入手方法

XSLT for C++ は Oracle9*i* および Oracle9*i*AS 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。

XSLT for C++ は、$ORACLE_HOME/xdk/demo/cpp/parser にあります。

XSLT for C++ の機能

ドキュメントのルート・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。

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

参照： 次のディレクトリおよびマニュアル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 API リファレンス - XDK および Oracle XML DB』

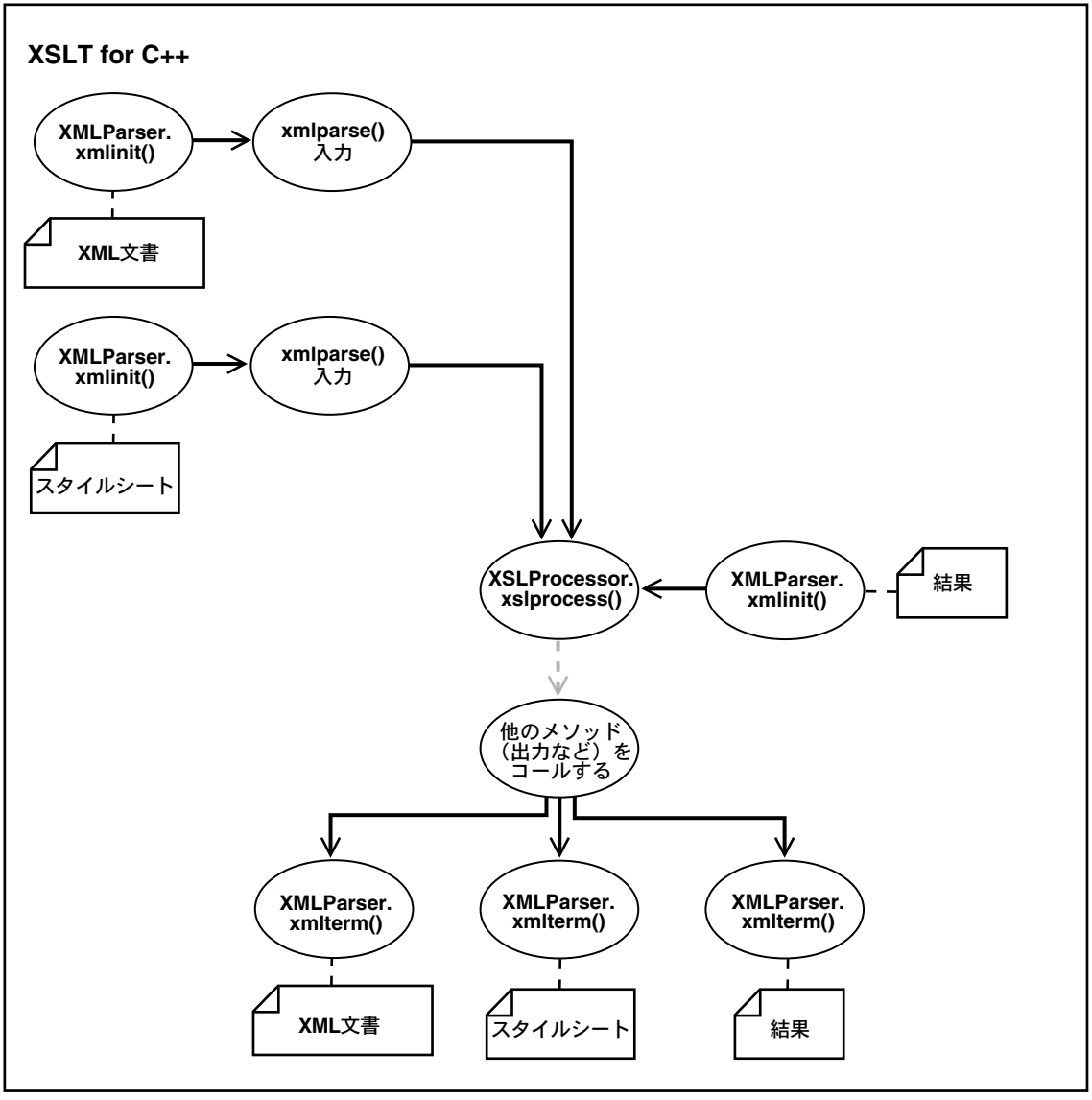
XSLT for C++（DOM インタフェース）の使用方法

図 17-1 に、XSLT for C++ の DOM インタフェース用の機能を示します。

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、XSLProcessor.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 API リファレンス - XDK および Oracle XML DB』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。

図 17-1 XSLT for C++ 機能（DOM インタフェース）の使用方法



XSLT for C++ の起動

XSLT 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

XSLT 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 17-1 に、コマンドライン・オプションを示します。

表 17-1 XXSLT for C++: コマンドライン・オプション

オプション	説明
-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。
-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。
-s	スタイルシート

提供される API を使用するための C++ コードの記述

XSLT for C++ は、提供される API を使用するようにコードを記述することによって起動することもできます。このコードは、xdk/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、xdk/demo/cpp/parser 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/demo/cpp/parser/ ディレクトリには、XSLT 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 17-2 に、サンプル・ファイルを示します。

表 17-2 XML Parser for C++ のサンプル・ファイル

サンプル・ファイル名	説明
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

XSLT 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

サンプル・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 17-3 に、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 17-3 XML Parser for C++: サンプル・ファイルによって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
XSLSample <xmlfile> <xsl ss>	XSLT プロセッサの使用例。入力として、XML ファイルおよび XSLT 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と、環境によっては出力の一部が欠落する場合があります。

18

XML Schema Processor for C++

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、Oracle9*i* 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、W3C の XML Schema 勧告をサポートしています。このため、Oracle9*i* 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、Oracle9*i* 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 プロセッサが Oracle9*i* 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 4 章「XML Parser for Java」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 勧告をサポートします。

XML Schema Processor for C++ のクラスは `XMLSchema` です。

> **参照：** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

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
- W3C の XML Schema 勧告

XML Schema Processor for C++: ソフトウェア

表 18-1 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 18-1 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
xdk/license.html	ライセンス契約
bin	スキーマ・プロセッサが実行可能なスキーマ
xdk/doc	API ドキュメント
xdk/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
xdk/mesg	エラー・メッセージ・ファイル
xdk/demo/cpp/schema	スキーマ・プロセッサの使用例

表 18-2 に、Solaris オペレーティング・システムに含まれるライブラリを示します。

表 18-2 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
libxml9.a	XML パーサー /XSLT プロセッサ
libxsd9.a	XML Schema プロセッサ
libcore9.a	CORE ファンクション
libnls9.a	グローバリゼーション・サポート

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、xdk/demo/cpp/schema 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。

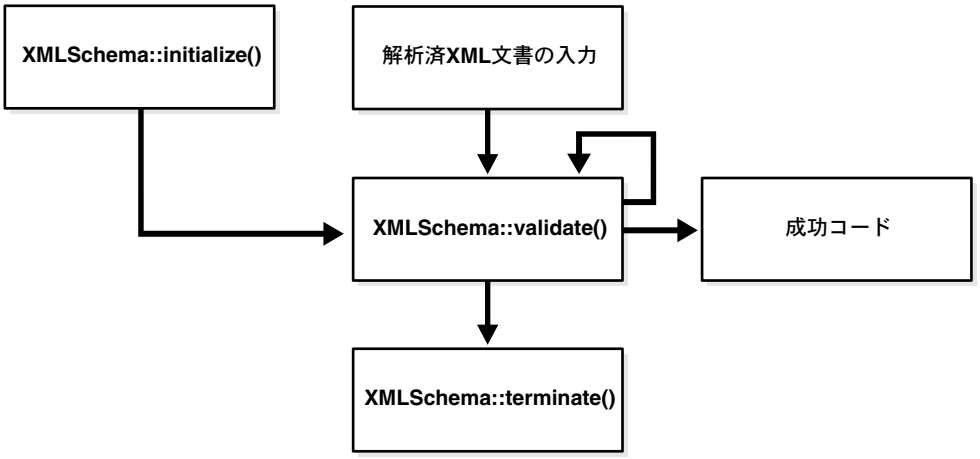
XML Schema Processor for C++ の使用方法

図 18-1 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 18-1 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

サンプル・ディレクトリには、API による Oracle XML Schema プロセッサ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 18-3 に、提供されるサンプル・ファイルを示します。

表 18-3 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
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
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

サンプル・プログラムを作成するには、make コマンドを実行します。

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 コマンドを実行します。

19

XML Class Generator for C++

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 は Oracle9*i* 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/demo/cpp/classgen` にあります。Class Generator の使用の詳細は、ソフトウェアに付属するマニュアルを参照してください。

XML C++ Class Generator の使用

XML C++ Class Generator は、XML DTD または XML Schema からソース・ファイルを作成します。XML C++ Class Generator は Document Type Definition（DTD）または XML Schema 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 または XML Schema 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、入力用 DTD を解析し、解析済文書を Class Generator に渡します。

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）をリクエストすることなく、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、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。19-5 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/xdk/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 19-1 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。

   `xmlcg [XML 文書ファイル名。ここでは xxxxx とします。]`

   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、19-3 ページの「XML C++ Class Generator への入力」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（および混合）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。

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。

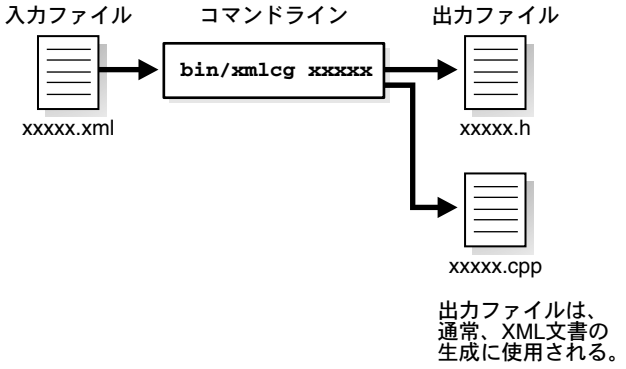
キャラクタ・セットのサポート

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* Database 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

注意： 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。

図 19-1 XML C++ Class Generator の機能



xmlcg の使用方法

単独のパーサーは、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。たとえば、次のような記述があります。

```
xmlcg [flags] <XML document or External DTD>
```

表 19-1 に、xmlcg オプション・フラグを示します。

表 19-1 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。
-s *name*	XML Schema - 指定された名前を持つ XML Schema を入力します。

ソフトウェアに含まれるサンプル・ファイルの使用

表 19-2 に、サンプル・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 19-2 サンプル・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
sample.cpp	サンプル・プログラム
sample.xml	DTD およびダミー・ドキュメントを含む XML ファイル
sample.dtd	sample.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。

- sample.xml に基づいて、クラス sample.h および sample.cpp を生成します。
- （sample.h を使用して）プログラム sample.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 sample.xml

XML ファイル sample.xml は、XML C++ Class Generator を入力します。sample.xml は、DTD ファイル sample.dtd を参照します。

```
<?xml version="1.0"?>
<!DOCTYPE sample SYSTEM "sample.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 sample.dtd

DTD ファイル sample.dtd は、XML ファイル sample.xml によって参照されます。sample.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: sample.cpp サンプル・プログラム

サンプル・プログラム sample.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.out 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"sample.xml"
#define OUT_DOCUMENT"sample.out"

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 sample.cpp
```

第 IV 部

XDK for PL/SQL

第 IV 部では、Oracle XML Developer's Kit（XDK）for PL/SQL の入手方法および使用方法を説明します。

- 第 20 章「XML Parser for PL/SQL」
- 第 21 章「XSLT Processor for PL/SQL」
- 第 22 章「XSU for PL/SQL」

注意： Oracle9*i* では、XML SQL Utility（XSU）for PL/SQL は XDK for PL/SQL の一部です。XSU の詳細は、第 8 章「XML SQL Utility（XSU）」を参照してください。

20

XML Parser for PL/SQL

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 Oracle9*i* に付属しています。また、OTN-J の Web サイト `http://otn.oracle.co.jp/tech/xml/xdk/index.html` 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 B「XDK for PL/SQL: 仕様」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、Oracle9*i* 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。

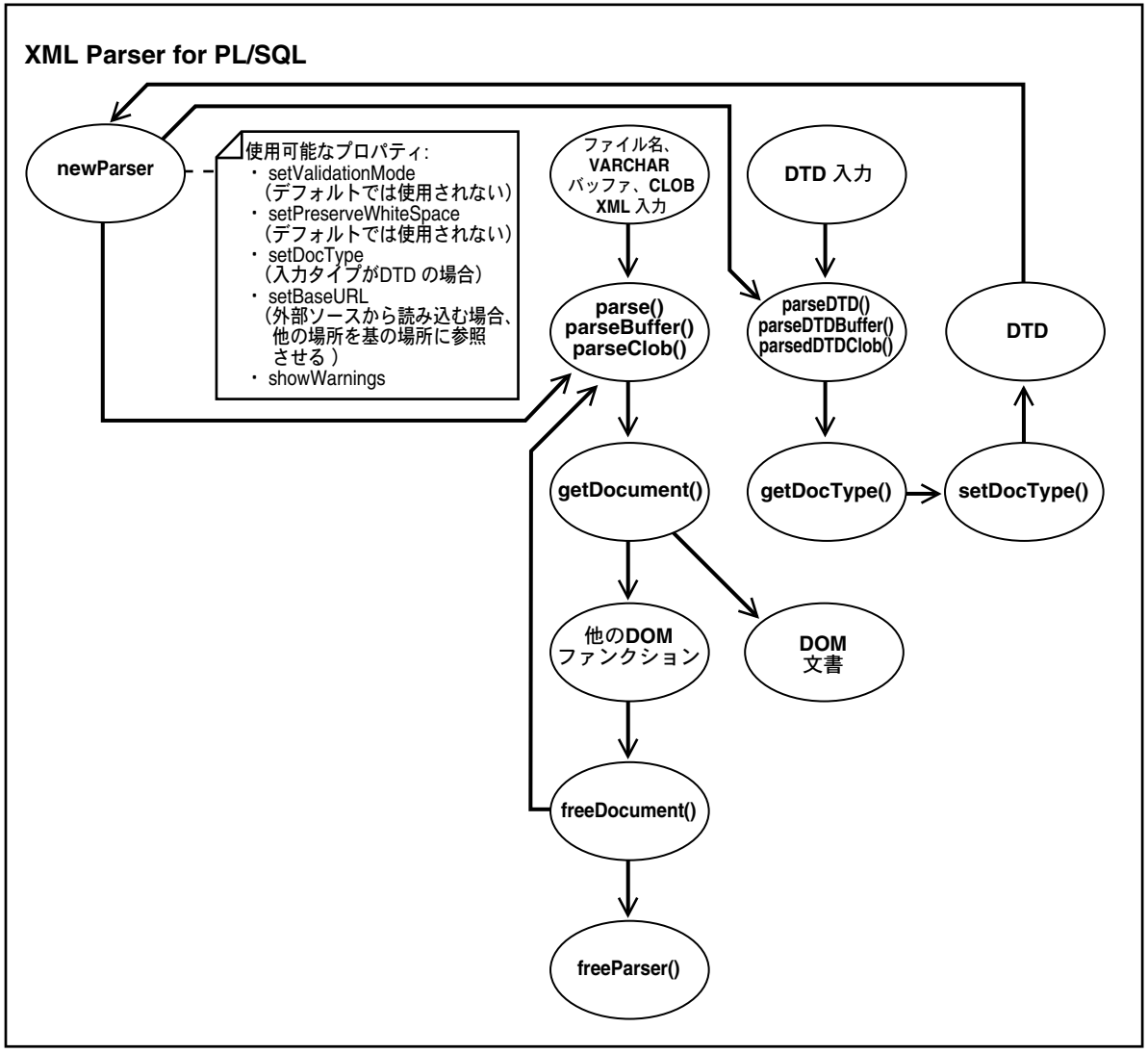
さらに、XML Parser for PL/SQL は、開発者が XML 文書を処理するために必要な、最も一般的な次の 2 つの API を提供します。

- W3C 勧告の DOM
- XSLT 勧告および XPath 勧告

これによって、Oracle9*i* 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、Oracle9*i* 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 Oracle9*i* 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 20-1 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 20-1 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 20-1 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 20-1 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。
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。

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 API リファレンス - XDK および Oracle XML DB』を参照してください。
- 『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』の PL/SQL API for XMLType に関する章も参照してください。

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 は妥当性を検証します。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。

sample/ ディレクトリの XML Parser for PL/SQL サンプル使用例

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;
   ```

   「権限付与が成功しました」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)
   ```

ここで、各部分は次の内容を表します。

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 )
   ```

   ここで、各部分は次の内容を表します。

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 XSLT 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:\/family.xml
Parsing XSL document c:\/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.co.jp/ の XML Discussion Forum に報告してください。

XML Parser for PL/SQL の例 : 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 : 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 : 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 : PL/SQL - xslsample.sql

```
-- This file demonstates a simple use of XSLT 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;
```

XML Parser for PL/SQL に関する FAQ

スレッド例外のパーサー・エラーが発生する理由

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 : 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。

PL/SQL で xmldom.GetNodeValue を使用する方法

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？

回答 : 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、要素を示します。

```
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
```

XML Parser for PL/SQL を使用して CLOB に含まれる DTD を解析する方法

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。

次のエラーが戻ります。

```
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; API
Documentation for xmlparser.parseDTDClob:

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が変更されてから、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。

最初に、./plsqlxmlparser_1.0.1/lib/sql/xmlparsercover.sql のバックアップを取ります。

次に、xmlparsercover.sql の 18 行目にある文字列 oracle.xml.parser.plsql.XMLParserCover.parseDTD を oracle.xml.parser.plsql.XMLParserCover.parseDTDClob に変更します。

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[])';
```

ファイルを保存して、SQL*Plus で xmlparsercover.sql を再実行します。Oracle XML Parser リリース 2.0.2.6 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 でローカル変数を使用する方法

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。

ユーザーへの JavaSysPriv 権限の付与時にセキュリティ・エラーが発生する理由

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。通常、次のコードで解決できます。

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO  youruser;
```

Oracle JVM オプションを使用して XML Parser for PL/SQL をインストールする方法

XML Parser for PL/SQL をダウンロードおよびインストールしました。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。Oracle9*i* 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。JVM 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。

XML Parser for PL/SQL に付属する domsample を使用する方法

XML Parser for PL/SQL の domsample を実行しようとしています。これは、XML Parser for PL/SQL に付属する例です。XML ファイル family.xml は、/hrwork/log/pqpd115CM/out ディレクトリにあると仮定します。

次のエラーが戻ります。

```
Usage of domsample is domsample(dir, inpfile, errfile)

    SQL>
    begin
    domsample('/hrwork/log/pqpd115CM/out','family.xml','errors.txt');
      end;
      /
    Error generated :
    begin
    *
    ERROR at line 1:
    ORA-20100: Error occurred while parsing No such file or directory
    ORA-06512: 22行目 APPS.XMLPARSER
    ORA-06512: 69行目 APPS.XMLPARSER
    ORA-06512: 69行目 APPS.DOMSAMPLE
    ORA-06512: 2行目
```

回答 : 内容からすると、エラーなしで sample および Readme のすべての手順を完了していないようです。xmlparserv2.jar がロードされていることを確認した後、注意して再度手順を完了してください。

CLOB の一部を抽出する方法

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？Oracle9*i* 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()`
- `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 のテキストを詳細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、内容を変更すると文字数が増加または減少する場合は、注意が必要です。

XML パーサーでメモリー不足のエラーが発生する理由

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。Oracle XML Parser で、次のようなメモリー不足のエラーが発生します。

```
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。

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。

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 または SAX 選択のガイドラインは、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。
- DOM は、より多くのメモリーを消費します。
- 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。

PL/SQL 使用時のメモリー要件

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

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 の使用に関する FAQ

XML Parser for PL/SQL の機能

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。

XML 文書でのエンコーディングの動的な設定

回答 : XML 文書では、エンコーディングを動的に設定できません。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。

特定のタグの要素数を取得する方法

パーサーを使用して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。

文字列を解析する方法

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。解決策として、次のいずれか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）

XML 文書を表示する方法

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 パーサーで XSLT プロセッサを使用すると、XSLT 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。

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

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

文字データからアンパサンド（&）を取得する方法

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。

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

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);
```

Linux 上でのパーサーの実行

回答 : JVM for Linux 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。

XML 名前空間および XML Schema のサポート

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。

パーサーが DTD ファイルを検出しない理由

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。

外部 DTD を使用した XML ファイルの検証

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。

パーサーの DTD キャッシュ機能

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。

XML 文書の解析後に DOCTYPE タグを挿入する方法

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。

XML DOM Parser の動作方法

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。

値を後で設定できるノードを作成する方法

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。

XML ファイルの要素を取得する方法

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。

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

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。

DOM で XML パーサーを使用中に実際のデータを取得できない理由

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得します。

XML Parser for PL/SQL による非 XML 文書の生成

回答 : XML Parser for PL/SQL は、非 XML 文書を作成できます。

サンプル・ファイルが実行できない理由（不適切なインストール方法の可能性）

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

CLOB 内の DTD の解析

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。

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 1: 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
clob_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 2: 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、ユーザー定義型を作成します。

   ```
   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。

前述の例は、『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。

ドキュメントの解析中にエラーが発生する理由

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。

```
ORA-20100: Error occurred while parsing: Unterminated sring
```

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。

```
ORA-20100: Error occurred while parsing: 無効な UTF8 エンコーディングです。
```

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。

XML Parser for PL/SQL を使用して指定された URL を解析する方法

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=3182DOMSample myxml.xml
```

XML パーサーを使用して HTML を解析する方法

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 1: 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得します）
2. `getNodeValue()`（文字列を戻します）
3. `setNodeValue()`（ノード値を設定します）

回答 2: 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。

XML Parser for PL/SQL および Oracle7 リリース 7.3.4 を使用して Web ブラウザへデータを送信する方法

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（Oracle7 リリース 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。

Oracle7 リリース 7.3.4 での XML Parser for Java の動作

XML Parser for Java は Oracle7 リリース 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 Oracle7 リリース 7.3.4 で動作します。
2. XML Parser for Java は XML SQL Utility に必要であるため、XML SQL Utility 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

`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 elementsfor i in 0..len-1 loop   n := xmlparser.item(nl, i);
   dbms_output.put(xmlparser.getNodeName(n)
```

回答 : 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。ただし、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を取得する方法

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得する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

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、ドキュメント・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、そのドキュメント・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
```

ドキュメント・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、それにドキュメント・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。

21

XSLT Processor for PL/SQL

この章の内容は次のとおりです。

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。

XSLT 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、XSLT 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 21-1 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()

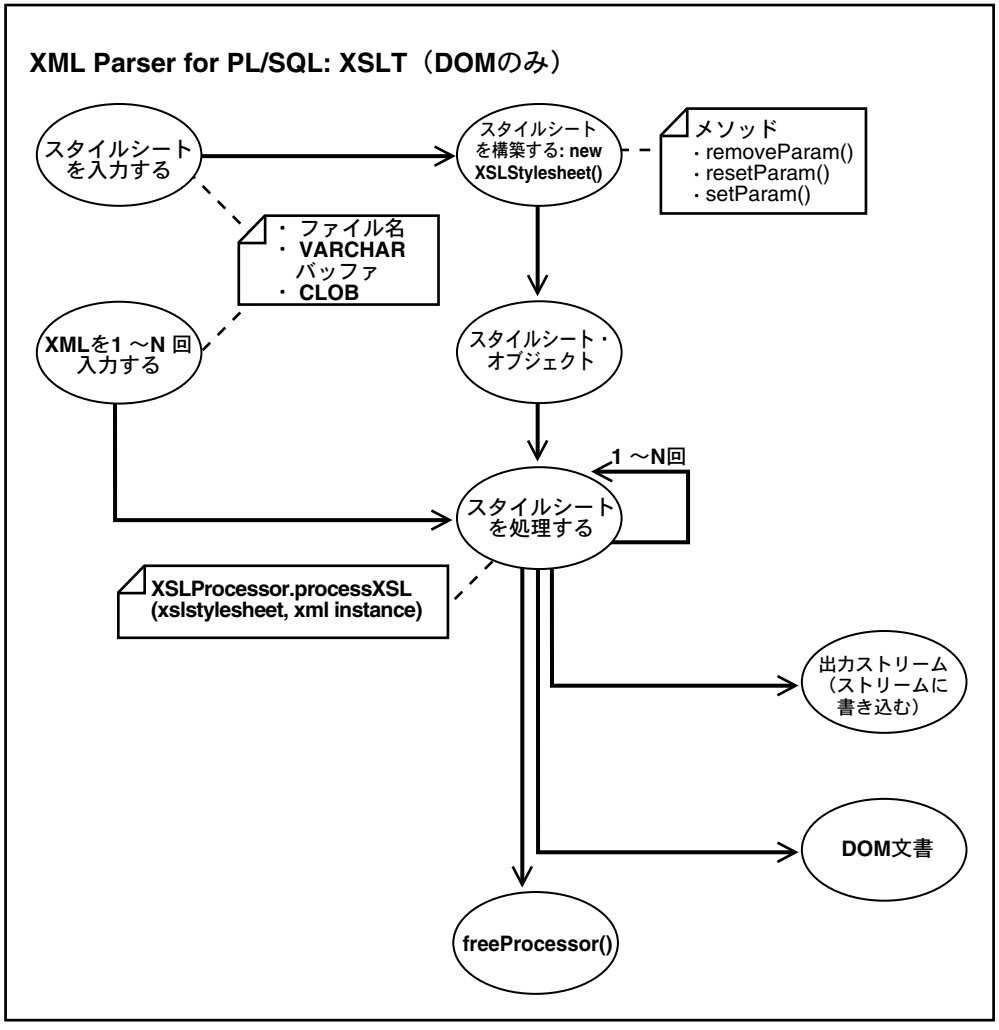
   これでスタイルシート・オブジェクトが作成され、XSLProcessor.processXSL(xslstylesheet,xmlinstance) プロシージャを使用して、「スタイルシートを処理する」手順が開始します。

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、XSLT 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 21-1 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。

XML Parser for PL/SQL の例 : 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>
```

22

XSU for PL/SQL

この章の内容は次のとおりです。

> **参照：** XSU の概要は、第 8 章「XML SQL Utility（XSU）」を参照してください。

XSU の PL/SQL API

XML SQL Utility（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。

XSU による XMLType のサポート

Oracle9*i* リリース 2（9.2）以上では、XSU は XMLType をサポートします。XSU での XMLType の使用は、オブジェクトまたは表内に XMLType 列が含まれる場合などに有効です。

> **参照：** XMLType と XSU の使用例については、『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』の特に XML の生成に関する章を参照してください。

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 またはスキーマの有無にかかわらず 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 1: 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 2: 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 3: ROW タグ名および ROWSET タグ名の変更

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 4: 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。

XML SQL Utility 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 が書き込まれない行があるときには例外が発生します。これは、終了条件として捕捉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;
/
```

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 5: 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');

--No longer needed:
--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 6: 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 7: 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 8: キー列を使用した XML 文書の更新（PL/SQL）

次の PL/SQL プロシージャについて考えてみます。

```
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 9: 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 10: 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 11: 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 12: 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 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;
/
```

XML SQL Utility（XSU）for PL/SQL に関する FAQ

この項では、XSU for PL/SQL に関する FAQ を示します。

LOB での XMLGEN.insertXML の使用

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込み、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'.
```

ログイン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
```

使用されなくなった XMLGEN API

XSU の最初のリリース（Oracle8*i* リリース 8.1.7）の前は、XSU の PL/SQL API は「XMLGEN」と呼ばれていました。XMLGEN を次の 3 つと混同しないでください。

a. XML を生成する SQL 関数 `SYS_XMLGEN`

b. XML を生成する PL/SQL パッケージ `DBMS_XMLGEN`

c. SQLX 標準関数 `XMLGen()`

XSU が最初に Oracle8*i* リリース 8.1.7 の本番リリースとして提供されたときに、`XMLGEN` パッケージは使用できなくなりました。引き続き Oracle ソフトウェアに添付されていますが、Oracle8*i* リリース 8.1.7 の本番コードの一部としては提供されません。そのため、`XMLGEN` という用語は、マニュアルには記載されません。

`XMLGEN` は、XML 生成に使用する `DBMS_XMLQuery`、および DML およびデータ操作に使用する `DBMS_XMLSave` に置き換えられます。Oracle9*i* リリース 2（9.2）以上の Oracle ソフトウェアに、`XMLGEN` は含まれません。

今回のリリースでは、`XMLGEN` パッケージを、XSU ダウンロード（XDK ダウンロードの一部）の一部として OTN-J からダウンロードできますが、最新の本番パッケージ `DBMS_XMLQuery` および `DBMS_XMLSave` へ移行することをお薦めします。メソッド名が同じであるため、簡単に移行できます。異なる点は、新しい XSU の PL/SQL API にメソッドが追加されていることのみです。すべてのメソッドは、最初の引数にコンテキスト・ハンドルを取ります。

第 V 部

XDK をサポートするツール製品およびフレームワーク

第 V 部に含まれる章は、次のとおりです。

23

JDeveloper を使用した XML アプリケーションの開発

この章の内容は次のとおりです。

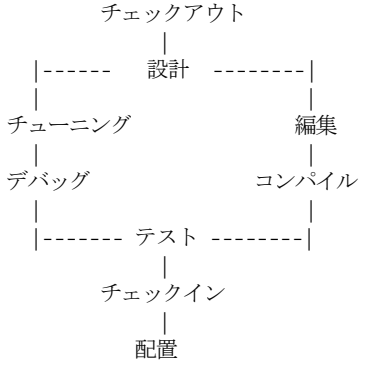
JDeveloper の概要

Oracle JDeveloper 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・トゥ・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。

生産性をより向上させるために、JDeveloper では、ソース・コードの制御、モデリングおよびコーディングからデバッグ、テスト、プロファイリングおよび配置までの一連の開発ライフ・サイクルをサポートする統合ツールを提供します。JDeveloper は、アプレット、JavaBeans、JavaServer Pages（JSP）、サーブレット、Enterprise JavaBeans（EJB）などの高品質な標準 J2EE コンポーネントを作成するためのウィザード、エディタ、ビジュアル設計ツールおよび配置ツールを提供することによって、J2EE の開発を容易にします。JDeveloper は、開発環境を拡張およびカスタマイズし、外部製品とシームレスに統合するためのパブリックなアドイン API も提供します。

JDeveloper による全開発ライフ・サイクルのサポート

Java は比較的新しい言語であり、Java 開発環境には従来のクライアント / サーバー・ツールが組み込まれています。開発者に必要なのは、すべての開発ライフ・サイクルをサポートする統合された開発環境です。



通常、開発者が JDeveloper を起動し、ソース制御システムからアプリケーションをチェックアウトし、開発プロセスを開始します。開発者によるアプリケーションの設計およびソース・コードの生成には、UML モデラーが有効です。JDeveloper は、機能性を向上させる視覚的かつコード・ベースのウィザードおよびエディタを提供します。また JDeveloper には、アプリケーションをコンパイル、テスト、デバックおよびチューニングするための様々なツールも含まれています。開発の完了後、開発者は、ソース制御システムにアプリケーションをチェックインし、目的の場所に配置できます。

Windows、Linux および Solaris オペレーティング環境での JDeveloper の実行

Oracle9*i* JDeveloper は、Java で完全に再作成されているため、Java Virtual Machine（JDK 1.3 以上）を搭載するすべてのオペレーティング・システムで動作します。また、Windows（NT、2000 および XP）、Linux および Solaris オペレーティング環境でサポートされます。

また、アドイン API を介して開発環境を完全に拡張できるというメリットもあります。これによって、顧客およびサード・パーティ・ベンダーは、製品を拡張したり、他の製品と統合することができます。

Java 以外に必要な言語

過去数年の間に、Java は、インターネット用のプログラミング言語として認知されるようになりました。Java が好まれる理由には、オペレーティング・システムに依存しないこと、単純であること、強力なコンポーネント・モデルであることなどがあります。

ただし、完全な E-Business アプリケーションを構築するには、開発者は、Java 以外の言語も必要とします。オラクル社では、Java、SQL および XML の結合に着目し、この結合に積極的に取り組んできました。ビジネス・ロジックおよびプレゼンテーション・ロジックのプログラミングには Java、データベースとの対話には SQL、疎結合アプリケーション間で情報を渡すためには XML を使用します。

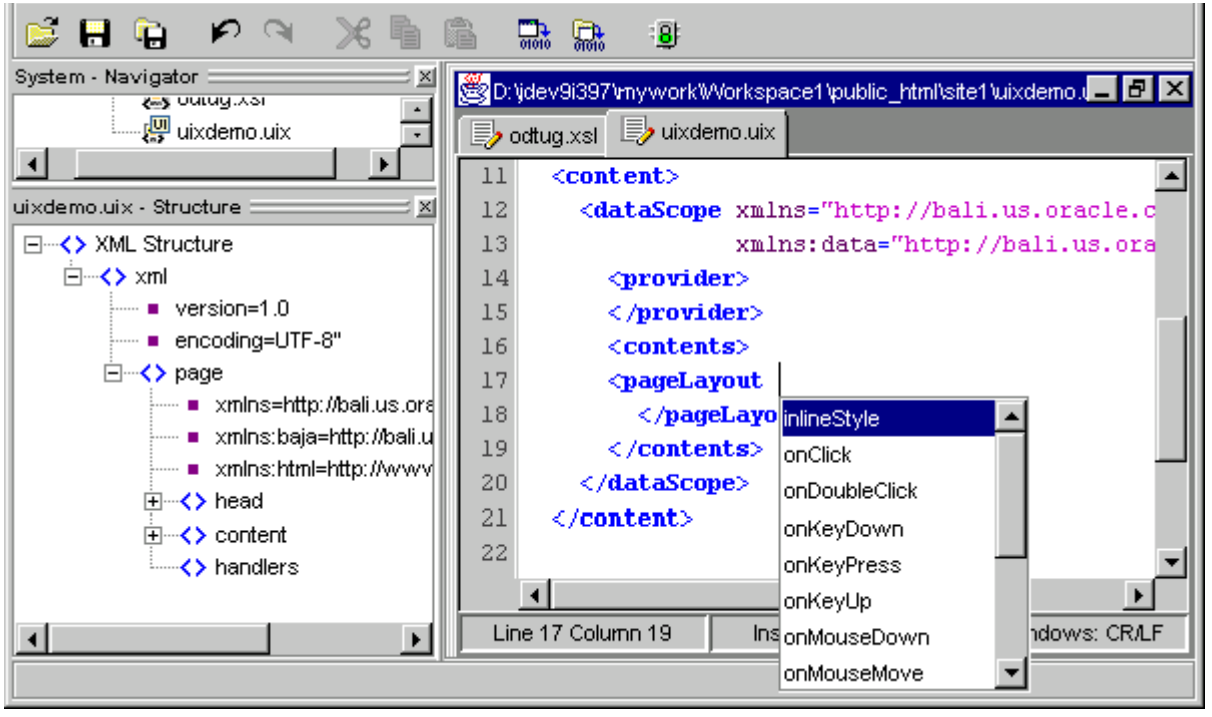
JDeveloper を使用すると、開発者は、Java、XML、HTML、SQL および PL/SQL を使用して E-Business アプリケーションを構築できます。JDeveloper は、これらの各言語に対して様々なコード・エディタおよびビジュアル・ツールを提供します。

JDeveloper の XML ツール

JDeveloper に統合された Oracle XDK は、XML を作成、処理および変換する多くの方法を提供します。たとえば、開発者は XSQL Servlet を使用して、データベースの情報に対して問合せおよび操作を実行し、XML 文書を生成し、XSLT スタイルシートを使用したドキュメントを変換して Web 上で使用できます。

JDeveloper には、スキーマ駆動の新しい XML エディタがあります。図 23-1 を参照してください。

図 23-1 動作中の JDeveloper スキーマ駆動 XML エディタ



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 JDeveloper 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 JDeveloper 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

  23-8 ページの「Page Selector 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

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： 次の章および Web サイトを参照してください。

- 第 9 章「XSQL Pages パブリッシング・フレームワーク」
- `http://jdeveloper.us.oracle.com`
- `http://otn.oracle.co.jp/products/jdev/`
- JDeveloper のオンライン・ディスカッション・フォーラムは、`http://www.oracle.com/forums` にあります。

BC4J

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。

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。

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。

参照： 第 24 章「BC4J の概要」を参照してください。

Web サービス開発の統合

JDeveloper は、標準 J2EE の開発方法を、最新の XML と新しい Web サービス標準（SOAP、UDDI、WSDL など）の両方、およびこれらと同等の Java ベースのサービスとシームレスに統合します。また、PL/SQL および J2EE アプリケーションにおけるこれまでの投資を保持するために、開発者が J2EE および PL/SQL アプリケーションから Web サービスを非常に簡単に作成、配置および使用するための、次の機能を提供しています。

- Java クラス、Enterprise JavaBeans および PL/SQL プロシージャからの Web サービスの作成
- Web サービスの作成時の WSDL ファイルおよび SOAP ディプロイメント・ディスクリプタの自動生成
- 1 クリックでの SOAP サービスの登録および登録解除
- Oracle9*i* SOAP および Apache SOAP 2.x の SOAP サーバーのサポート
- WSDL ファイルからの Web サービス・プロキシの作成
- 1 クリックでの WSDL ファイルからの Web サービス・プロキシの同期化
- WSDL ファイルからのサーバーの骨組みの作成

JDeveloper の動作環境

JDeveloper 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。JDeveloper を実行するには、128 MB 以上の RAM が必要です。

JDeveloper のシステム最小要件

『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 Oracle9*i* の実行
- ローカルでの Oracle9*i* 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）

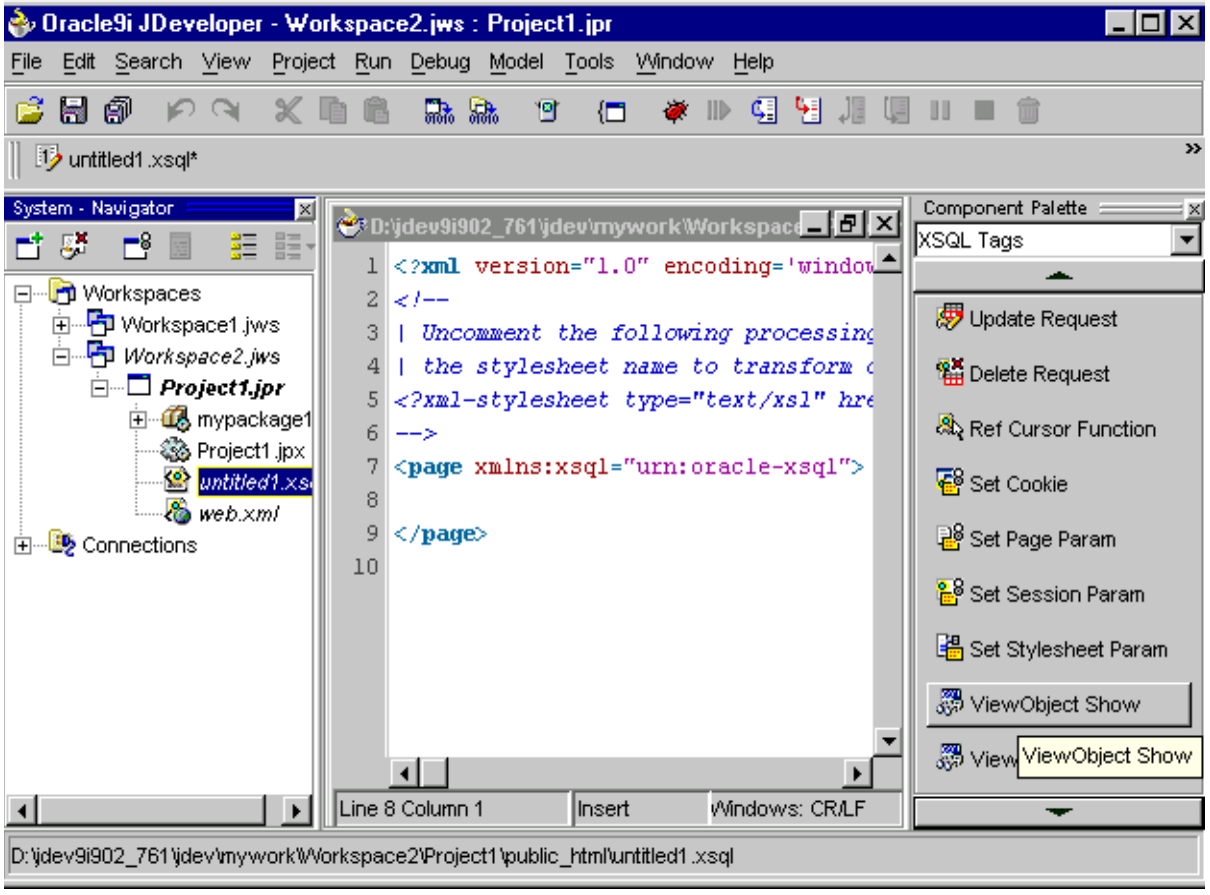
実際の CPU 使用率および必要なディスク領域の点から、これらもシステム要件になります。

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。

XSQL Component Palette

XSQL Component Palette は、データベース表または BC4J ビュー・オブジェクト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 23-2「JDeveloper の XSQL Component Palette」に、JDeveloper の XSQL Component Palette を示します。

図 23-2 JDeveloper の XSQL Component Palette



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 ビュー・オブジェクト上に作成できます。BC4J ビュー・オブジェクト上に 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。

JDeveloper の XDK 機能

JDeveloper がサポートする XDK for Java 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。

JDeveloper への Oracle XDK 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-J サイト http://otn.oracle.co.jp/tech/xml/xdk/index.html 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。

> **参照：** XML Transviewer Beans の使用方法の詳細は、第 10 章「XDK JavaBeans」を参照してください。

XSQL Pages を使用した JDeveloper での Web アプリケーションの開発

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。
- XSLT 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>
```

JDeveloper を使用すると、XSQL ファイルを簡単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 Pages 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。

> **参照：** オンライン「Help」メニューの『The JDeveloper Guide』を参照してください。

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 XDK 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>
...
```

JDeveloper でのアプリケーションの構築手順

JDeveloper で XSQL 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. 「File」>「New」を選択して、「OK」をクリックします。
4. 「WebObjects」>「XSQL」を選択します。
5. `<PAGE>` タグと `<?PAGE>` タグの間にカーソルを置きます。
6. Component Palette から、`ViewObjects Show` タグを選択します。
7. 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。

Page Wizard でのこれらの手順を終了すると、BC4J フレームワークのビュー・オブジェクトに基づいた 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 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 9 章「XSQL Pages 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。
- XSLT 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。

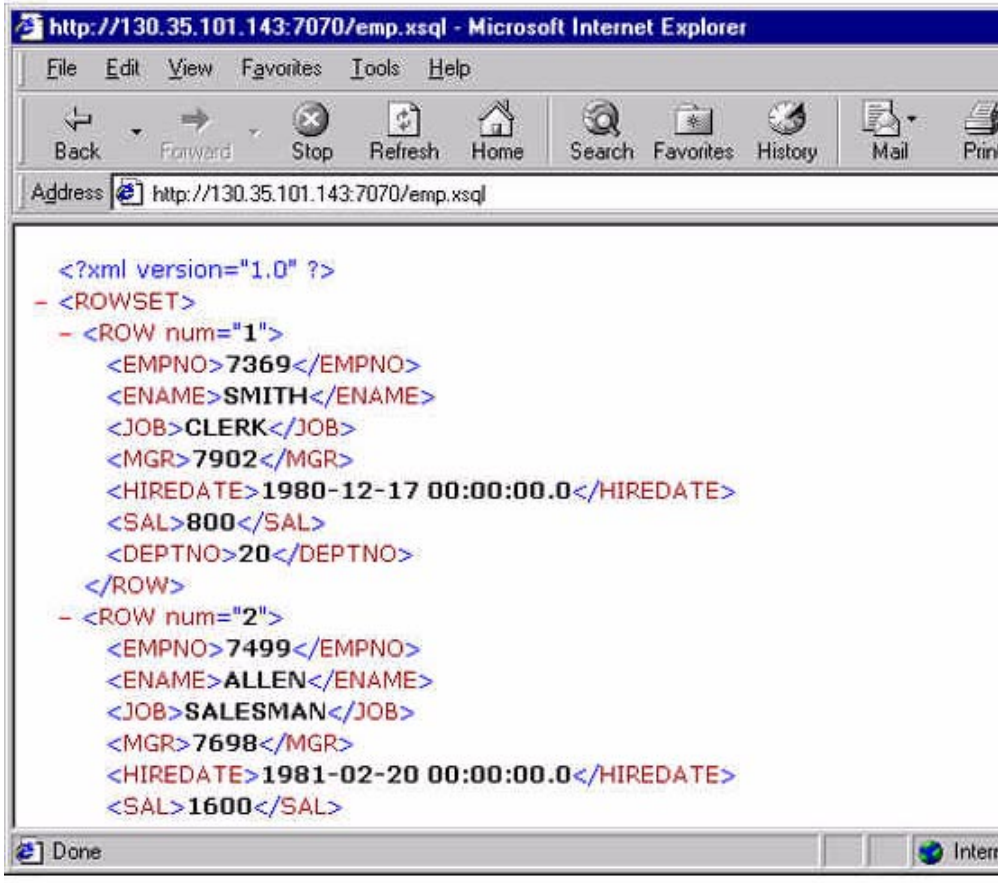
XSQL ファイルの簡単な問合せを考えてみます。この問合せでは、emp 表のすべての従業員の詳細情報が戻されます。この情報を取得する XSQL コードは、例 2 で示します。

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 23-3 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 23-3 未加工の XML 形式の従業員データ



データを表形式で出力する場合、XSQL コードを少し変更してスタイルシートを指定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>
```

結果は表になります。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 9 章「XSQL Pages 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 および XML アプリケーションに関する FAQ

この項では、JDeveloper についての質問および回答を示します。

JSP で XML 文書を作成する方法

XML Class Generator for Java が（DTD に基づいて）生成したクラスを使用し、XSLT 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リクエストで共有され、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、リクエスト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、すべてのリクエスト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リクエスト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。

@code を直接 document() 行に使用する方法

@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]"/>
```

messages.xml からデータを取得する方法

messages.xml に格納されたデータをデータベースから取得できますか？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 : データは取得できます。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み込み、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 文書をデータベースへ移動する方法

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。

24

BC4J の概要

この章の内容は次のとおりです。

- Business Components for Java（BC4J）の概要
- XML メッセージ機能の実装
- JDeveloper でのモバイル・アプリケーションの作成
- BC4J に関する FAQ

Business Components for Java（BC4J）の概要

Business Components for Java は、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を構築するための JDeveloper のプログラミング・フレームワークです。通常、このアプリケーションは、次の要素で構成されます。

- Java または HTML（あるいはその両方）で作成されたクライアント側ユーザー・インタフェース
- ビジネス・ロジックおよびビジネス・オブジェクトのビューを提供する、1 つ以上のビジネス・ロジック層コンポーネント
- 基礎となるデータを格納するデータベース・サーバー上の表

Business Components for Java フレームワークを使用して構築された複数層アプリケーションは、クライアントによって共有可能なコンポーネントにビュー、ビジネス・ルールおよびカスタム・コードを配置します。Business Components for Java フレームワークを使用すると、これらのコンポーネントの構築と保持、使用と再利用、およびカスタマイズを簡単に行うことができます。サポートされているどのプラットフォームに配置する場合も、コンポーネントを変更する必要はありません。

> **参照：** 複数層のアプリケーションの例は、図 24-1「BC4J の使用」を参照してください。

この方法は、次のような多くの機能およびメリットを提供します。

表 24-1 BC4J の機能およびメリット

機能	説明
カプセル化されたビジネス・ロジック	検証を含むビジネス・ロジックがビジネス・ロジック層に格納され、実行されます。これによって、真の Thin クライアント、簡単なカスタマイズおよび再利用が可能になります。
柔軟なデータのビュー	データのビューは SQL ベースで、基礎となるエンティティから完全に切り離されています。これによって、柔軟な表示スキームが提供されます。
Thin クライアント	BC4J は、Thin クライアント（ビジネス・ロジック、およびビジネス・ロジック層によって処理されるデータのビューに対する簡単なウィンドウ）をサポートします。
柔軟な配置	CORBA サーバー・オブジェクトおよび EJB Session Bean として、ローカルで配置するか、または標準サーバー・プラットフォーム上に配置します。
データベースとの対話	BC4J のコンポーネント・ベースのフレームワークは、マスターとディテールの調整およびロックなど、繰り返し行われる多くのコード作成タスクを処理します。

表 24-1 BC4J の機能およびメリット（続き）

機能	説明
トランザクション管理	Business Components for Java は、キャッシュの変更を管理し、データベースへの変更の転送を処理します。

BC4J は、ドメイン固有のコンポーネントを構築およびカスタマイズするためのフレームワークを構成します。開発者は、フレームワークによって提供されたクラスおよびインタフェースからオブジェクトを派生させ、カスタム・コードを追加してアプリケーションに固有の機能を実装します。この処理をサポートするために、次のビジネス・コンポーネントが使用されます。

表 24-2 BC4J のビジネス・コンポーネント

オブジェクト	説明
エンティティ・オブジェクト	エンティティ・オブジェクトは、データベース表、ビューまたはシノニム用のビジネス・ロジックをカプセル化します。クライアントは、1 つ以上のビュー・オブジェクトを介してエンティティ・オブジェクトのデータにアクセスします。指定されたエンティティ・オブジェクトは、任意の数のビュー・オブジェクトが使用できます。エンティティ・オブジェクト間の関連は、対応付けを使用して表されます。
ビュー・オブジェクト	ビュー・オブジェクトは、SQL 問合せを使用して、エンティティ・オブジェクトからフィルタリングされた属性のサブセットを指定します。クライアントは、結果セットを介してナビゲートし、属性値を取得および設定することによって、データを操作します。ビュー・オブジェクト間の関連は、ビュー・リンクを使用して表されます。
アプリケーション・モジュール	アプリケーション・モジュールは、他のアプリケーション・モジュールによって指定されたビュー・オブジェクト、ビュー・リンクおよびトランザクションのインスタンスのための論理コンテナです。

作成する各ビジネス・コンポーネントは、1 つの XML ファイルと 1 つ以上の Java ファイルによって表されます。XML ファイルは、メタデータ（設計時にウィザードを使用して宣言したアプリケーションの機能および設定についての定義情報）を格納し、Java ファイルは、オブジェクトのコード（アプリケーション固有の動作）を格納します。各オブジェクトは、Java で作成されたパッケージのディレクトリ・ベースのセマンティクスを使用して、パッケージに編成されます。

ビジネス・コンポーネントを表す Java ファイルおよび XML ファイルでは、同様の構文を使用して、これらのファイルが含まれるパッケージを識別します。

表 24-3 BC4J によって使用される Java 構文および XML 構文

Java	XML

package d2ePackage;
...
public class DeptViewImpl extends oracle.jbo.server.ViewObjectImpl {
...
}
	<ViewObject
	Name="DeptView"
	...
	ComponentClass="d2ePackage.DeptViewImpl">

ビジネス・コンポーネント・フレームワークの概要

ビジネス・コンポーネント・フレームワークは、oracle.jbo.* に存在する、組込みアプリケーション機能を持つクラス・ライブラリです。このフレームワークを使用するには、ベース・クラスを特化してアプリケーション固有の動作を導入し、フレームワークによってオブジェクト間の多くの基本的な相互作用を調整できるようにする必要があります。

Business Components for Java の設計時ウィザードおよびエディタを使用すると、コンポーネントの特性（属性、関連およびビジネス・ルール）を定義することによって、ビジネス・ロジックを構築できます。Business Components for Java は、指定された動作を実装するための Java ソース・コードおよび XML メタデータを生成します。コードはフレームワークから継承されるため、Java ソース・ファイルは、大量のコードが含まれない簡潔なファイルになります。そのため、ビジネスをモデル化するコードを追加する場所を簡単に確認できます。JDeveloper を使用すると、配置プラットフォームに関係なく、Java コードを追加して動作を拡張または変更したり、アプリケーション・サービスを簡単にテストすることができます。

ビジネス・コンポーネントの使用

JDeveloper は、Business Components for Java フレームワークの統合サポートを提供します。ウィザード、プロパティ・エディタなどの設計ツールを使用して、オブジェクトの特性（属性、関連およびビジネス・ルール）を定義します。次に、JDeveloper によって実行可能な Java コードおよび XML が生成され、コンポーネントに対して定義した動作が実装されます。

理論上は、このコードを独自に作成することもできますが、実際は、すべての必要なコードの生成と、すべての依存性の指定を確実に行うウィザードを使用する方が効率的です。その後、生成されたコードを、アプリケーションの特定の要件を満たすように編集します。JDeveloper では特定の方法を使用する必要はありませんが、通常、開発プロセスで次の項目を確認する必要があります。

- 使用するエンティティおよびビジネス・オブジェクト。エンティティ・オブジェクトを単独で使用（顧客など）するか、または複数のエンティティ・オブジェクトを組み合せる（ヘッダー、明細項目、出荷および配送という項目を含む発注書など）ことができます。
- エンティティの関連。たとえば、部門と従業員間の多対 1 関連を定義できます。
- 検証規則。たとえば、5 年以上勤務している従業員の給与の最低額を指定するビジネス・ルールを規定できます。ルールは、属性、エンティティおよびビジネス・オブジェクトに適用できます。
- 表示および操作されるデータ。ビューを作成することによって、エンティティのデータを選択およびフィルタリングする SQL 問合せを定義し、ネットワークの通信量およびクライアント側の処理要件を最小限にすることができます。

BC4J 設計時のメリット

1. データベースの表に格納されたデータを、実在するエンティティ（従業員など）を使用して表すことができます。
2. JDeveloper は、表のデータおよびメタデータを使用して、エンティティを表す Java クラスを作成します。この Java コードを編集して、デフォルトの属性および動作を変更できます。
3. JDeveloper は、メタデータを、カスタマイズ可能な XML ファイルとして表示します。
4. JDeveloper は、データを選択するための基準を指定するデフォルトのビュー・オブジェクトを作成します。デフォルトの他に（またはデフォルトのかわりに）、ユーザー独自のオブジェクトを定義できます。
5. JDeveloper は、各ビュー・オブジェクトに対してカスタマイズ可能な Java クラス（ビュー・オブジェクト定義に対するクラスおよび行に対するクラス）を生成します。また、各ビュー・オブジェクトの XML ファイルを生成します。
6. 開発者は、ウィザードを使用して、アプリケーション・モジュールを定義します。アプリケーション・モジュールは、関連オブジェクトの論理コンテナです。アプリケーション・モジュールは、トランザクションを定義および実行するためのコンテキストを提供します。
7. ビジネス・コンポーネントで構成されたアプリケーション・サービスを設計、構築、テストおよびデバッグした後、このアプリケーション・サービスを配置できます。

BC4J 実行時のメリット

1. クライアント・コードによってアプリケーション・モジュールが初期化され、このアプリケーション・モジュールに含まれるエンティティおよびビューがロードされます。
2. ビュー・オブジェクトによって実行時に問合せが実行され、対応する 1 つ以上のエンティティのデータに対して操作が行われます。
3. 各ビュー・オブジェクトは、結果セットのナビゲートに使用可能なデフォルトのイテレータを提供します。
4. 問合せによって 1 つ以上の結果行がフェッチされる場合、個々の行は、行オブジェクトで表されます。結果行の各列値には、行オブジェクトの属性を介してアクセスできます。
5. クライアント・フォームでの制御によって、ユーザーによるデータの表示および編集が可能になります。この制御では、ビュー・オブジェクトの行が表示されます。これらの行自体は、基礎となるエンティティ・オブジェクトにバインドされています。ユーザーがある制御で値を変更すると、Business Components for Java フレームワークによってビュー・オブジェクトにこのアクションが送信されます。このアクションは、ビュー・オブジェクトによってエンティティ・オブジェクトに送信されます。エンティティ・オブジェクトに対してビジネス・ルールが指定されている場合、フレームワークによってデータベースに新しい値が送信される前に、このビジネス・ルールによってこの値が検証されます。

XML メッセージ機能の実装

Business Components for Java（BC4J）フレームワークは、データベースとの間で送受信される E-Commerce の XML メッセージをマップするための、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。

Sun 社では、Java Message Service（JMS）API を提供しています。Oracle9*i* は、Advanced Queueing（AQ）API を提供します。これらを BC4J とともに使用して、XML メッセージ機能を実装できます。

これを行うには、`ViewObjectImpl` クラスおよび `ViewRowImpl` クラスのビジネス・コンポーネント・フレームワーク・メソッドを使用して、XML データの正規形式の読込みおよび書込みを可能にします。

- `writeXML()`: 現行のオブジェクトを XML 要素に書き込みます。これによって、たとえば XML メッセージのペイロードとして、任意の XML 文書に現行のオブジェクトを追加できます。
- `createXMLDefinition()`: ViewObject または ViewRow に対する XML DTD を作成します。
- `readXML()`: XML 要素からこのオブジェクトの属性値または行を読み込みます。XML 要素は、XML 文書または XML メッセージから派生されます。

XML メッセージ機能の例に、実際のメッセージ・システムの実装方法を示します。さらに、XML メッセージ機能を実装するために実行する必要がある一般的な手順を示します。`$ORACLE_HOME\BC4J\samples` を参照してください。

ビジネス・コンポーネント・メソッドの詳細は、Javadoc を参照してください。JMS の詳細は、JavaSoft 社の Web サイトを参照してください。Advanced Queueing 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。

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 21 章「XSLT Processor for PL/SQL」
- 『Oracle9*i* Java Developer's Guide』
- http://otn.oracle.com/products/bc4j

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 Business Components for Java 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 24-1「BC4J の使用」に、BC4J を使用して XML 文書を生成する方法を示します。

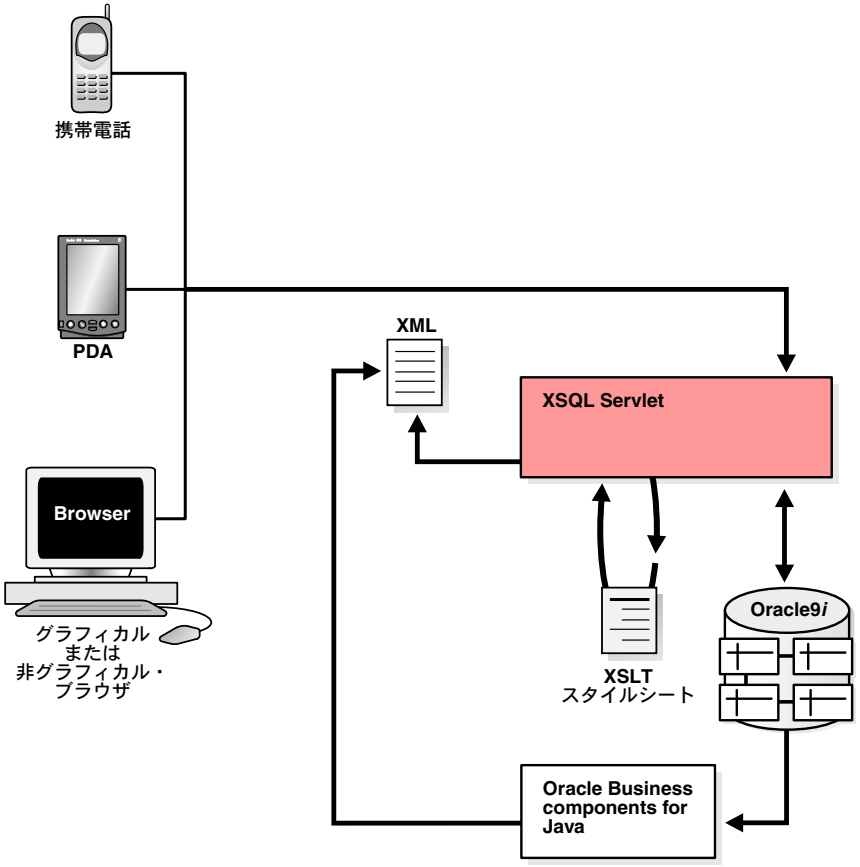
BC4J フレームワークは、データベースとの間で送受信される E-Commerce の XML メッセージをマップするための、一般的なメタデータ駆動のソリューションを提供します。BC4J の機能に関するテクニカル・ホワイト・ペーパーは、次の Web サイトにあります。

```
http://otn.oracle.co.jp/products/jdev/
```

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 に組み込まれることで詳細な開発をサポートします。この Java フレームワークは同じ Web サイトからダウンロード可能です。

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。

図 24-1 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。

JDeveloper でのモバイル・アプリケーションの作成

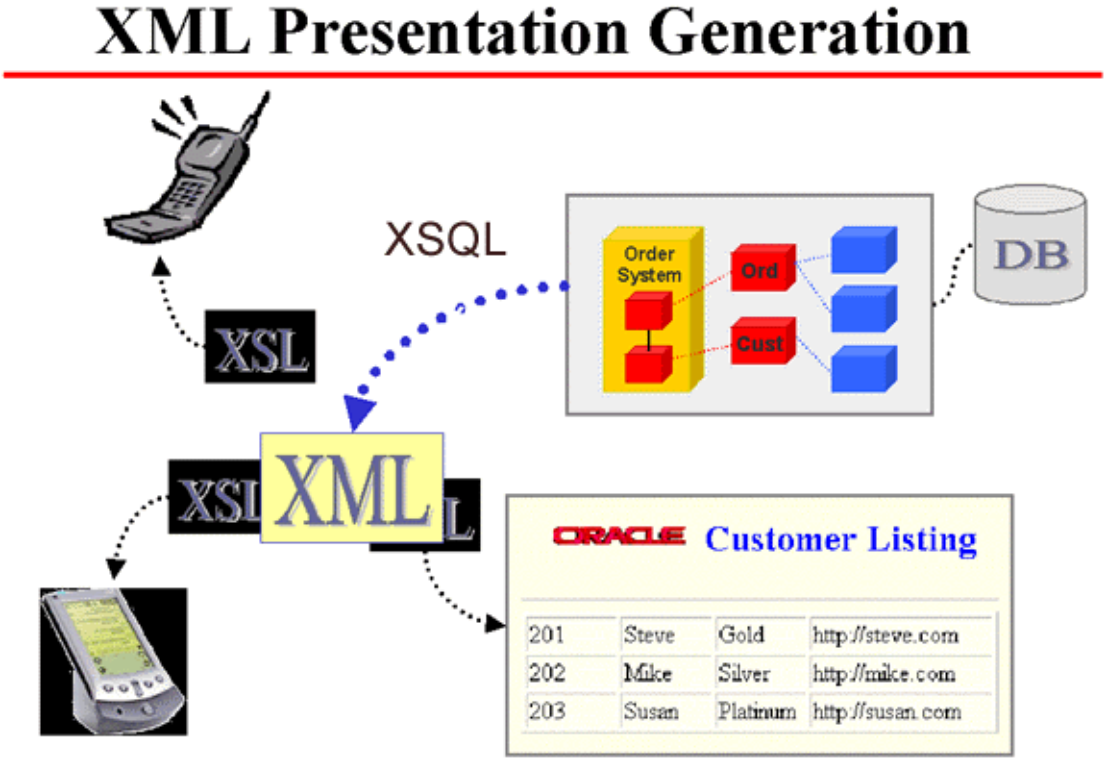
このモバイル・アプリケーションは、部門データベース・アプリケーションです。このアプリケーションは、Business Components for Java（BC4J）および XML を使用して、ワイヤレス・デバイスでアクセスできるアプリケーションを開発できることを実証します。このアプリケーションは、主に次の 2 つの部分で構成されます。

- 1 つは Business Components for Java（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 24-2「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」に、モバイル・アプリケーションが BC4J、XSQL Servlet、XSLT 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。

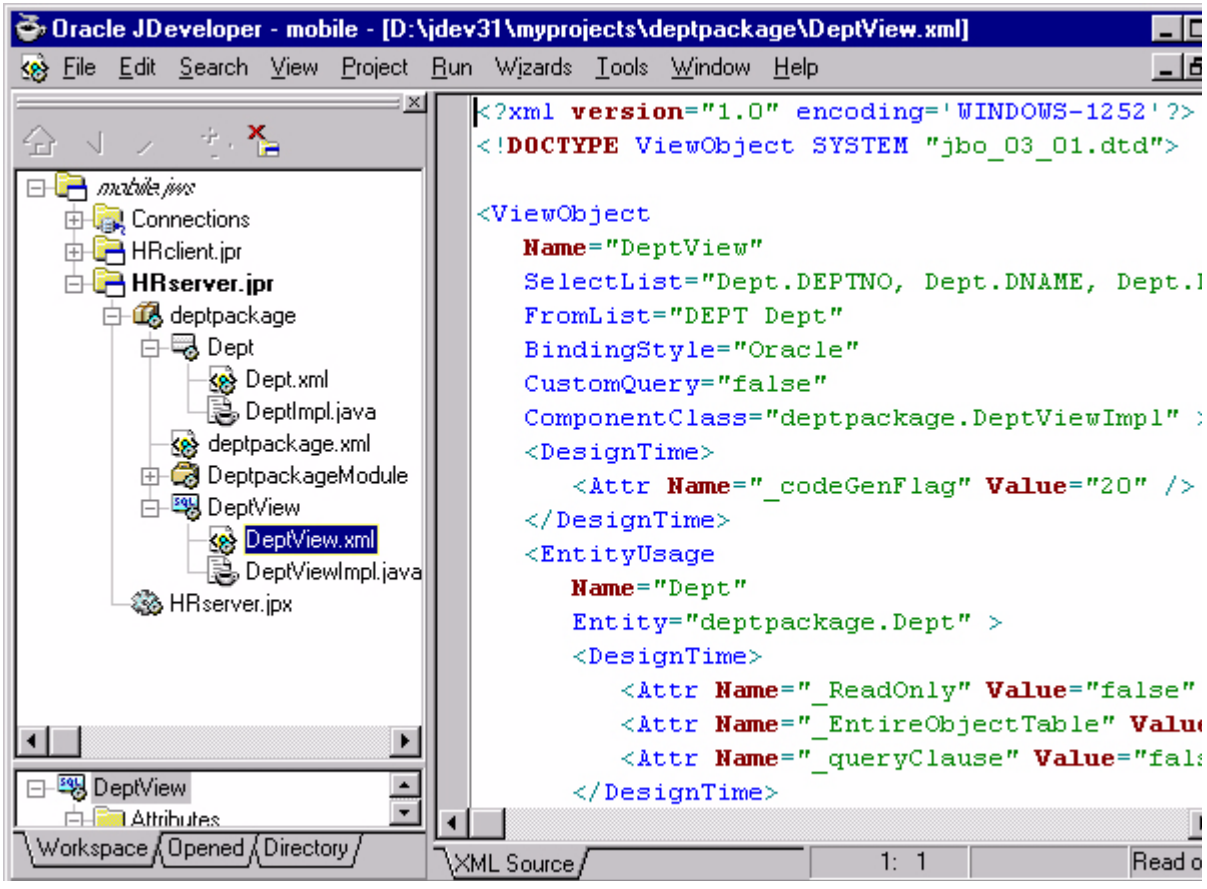
図 24-2 BC4J および XSQL Servlet を使用した JDeveloper でのモバイル・アプリケーションの作成



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 24-3「BC4J アプリケーション : DEPT ビュー・オブジェクトの XML ファイル」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。23-11 ページの「JDeveloper の XDK の例 1: BC4J メタデータ」を参照してください。

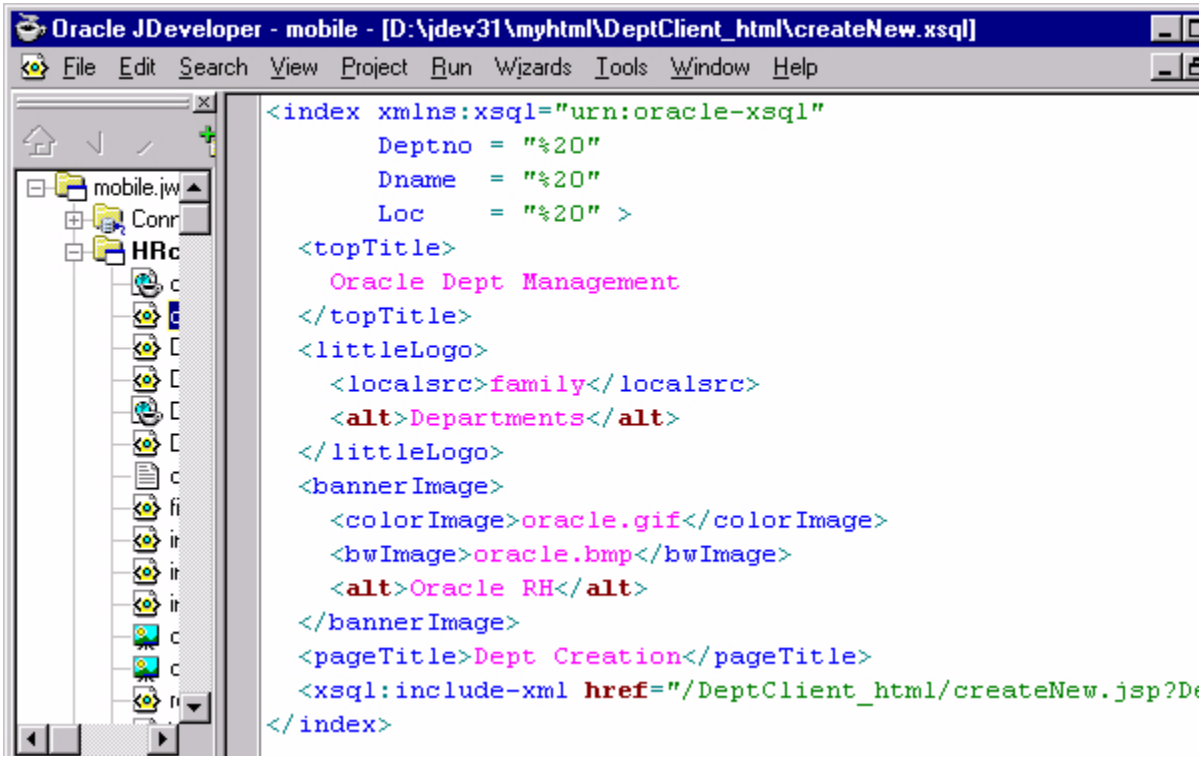
図 24-3 BC4J アプリケーション : DEPT ビュー・オブジェクトの XML ファイル



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 24-4 に、新しい部門を作成するための JSP ページをコールする XSQL ファイルを示します。

図 24-4 BC4J アプリケーション : JSP ページをコールする XSQL ファイル



データ読込みを行うデバイスに従った 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 24-5 に、Palm Pilot のエミュレータのブラウザに表示するために、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 24-5 BC4J アプリケーション : XSLT スタイルシート（indexPP.xsl）

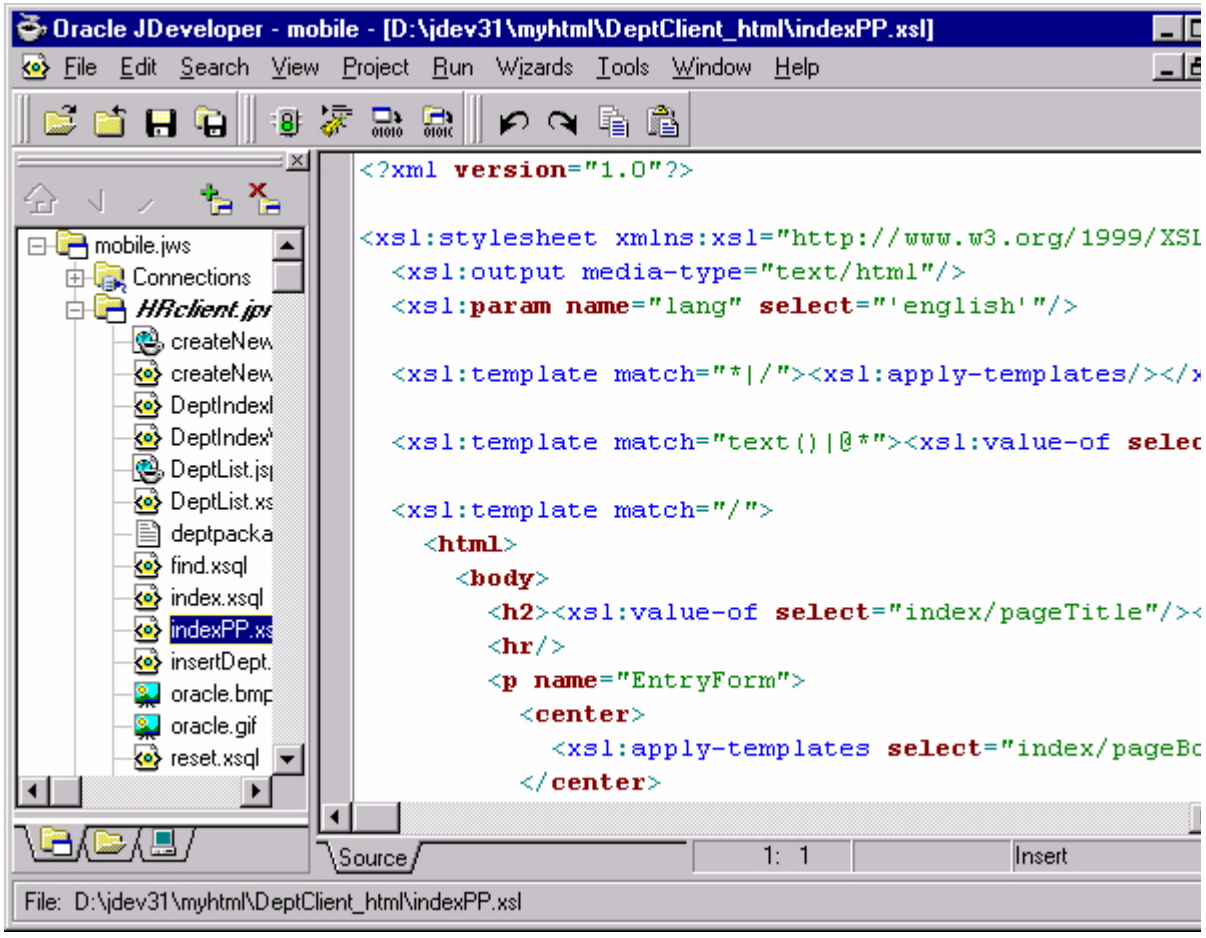


図 24-6 に、部門アプリケーションのクライアントを実行中の携帯電話のエミュレータを示します。携帯電話のエミュレータの設定画面も示しています。

図 24-6 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

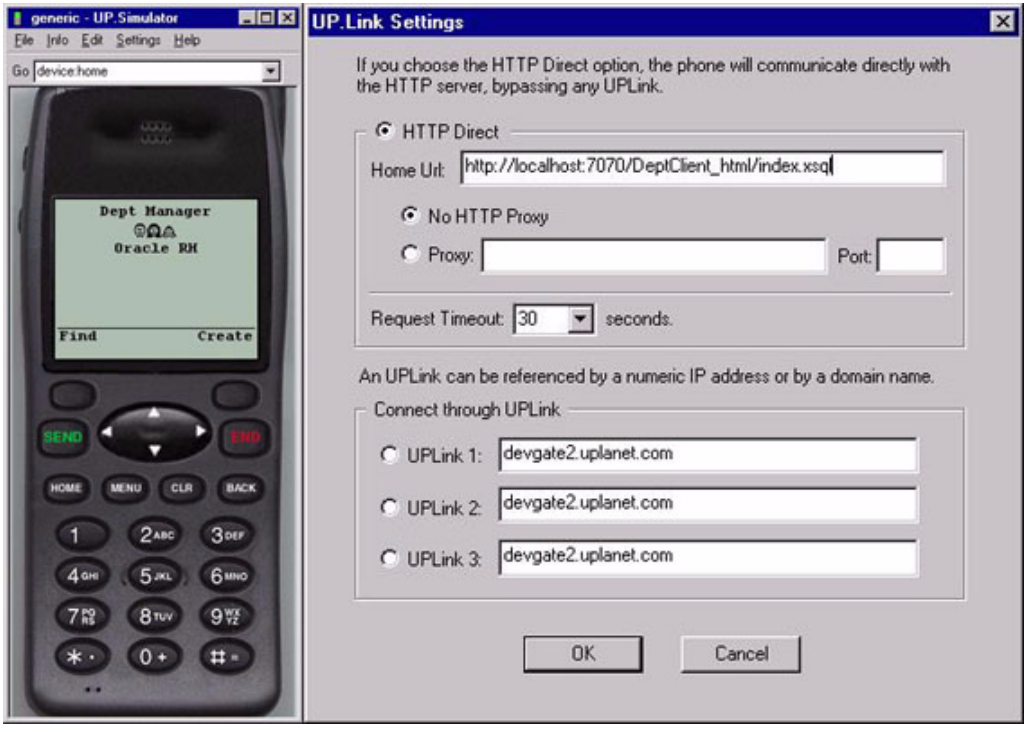


図 24-7 に、HandWeb ブラウザを介して部門アプリケーションにアクセス中の Palm Pilot のエミュレータを示します。

図 24-7 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

BC4J による XSQL クライアントの作成

Oracle9*i* JDeveloper 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 データを取得し、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。

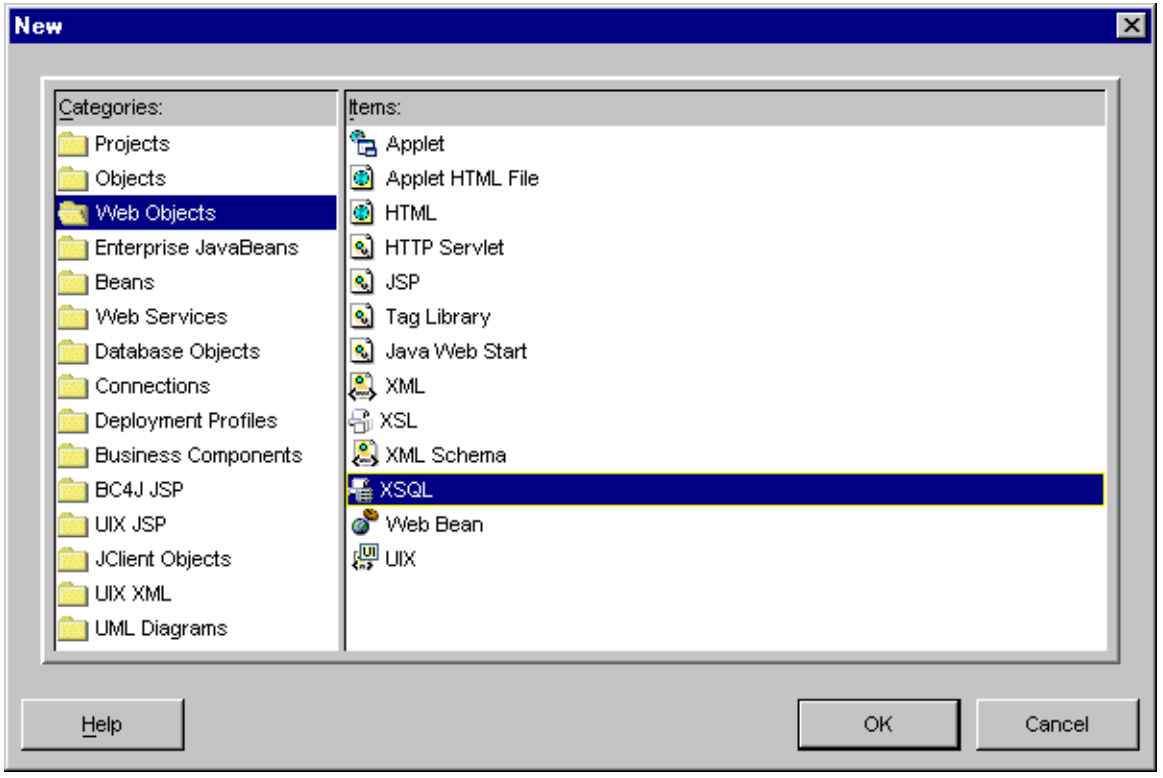
- 第 21 章「XSLT Processor for PL/SQL」
- 『Oracle9*i* Java Developer's Guide』
- 『Oracle9*i* ケース・スタディ - XML アプリケーション』

Web 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 Component Palette 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ため、XSQL ページのアイコンは特に重要です。図 24-8 に、JDeveloper の Web Object Gallery を示します。

図 24-8 XSQL、XML および XSL アイコンがある JDeveloper の Object Gallery



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および管理

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

BC4J に関する FAQ

この項では、BC4J についての質問および回答を示します。

J2EE 準拠のコンテナにおける BC4J アプリケーションの動作

回答 : BC4J フレームワークは、すべての J2EE 準拠のアプリケーション・サーバーで動作します。Oracle9*i* JDeveloper IDE は、任意の J2EE 1.2 コンテナに配置するために、BC4J ベースの J2EE アプリケーションの自動パッケージングをサポートしています。さらに、Oracle9*i*AS または WebLogic コンテナを使用している場合、このパッケージング機能の他に、ツールによって自動で配置を行うこともできます。

データベースにおける BC4J アプリケーションの動作

回答 : BC4J を使用して構築されたアプリケーションは、SQL92 と互換性のあるすべてのデータベースで動作します。

デフォルトでは、BC4J フレームワークは、Oracle データベースの固有のメリットおよび Oracle JDBC Drivers の機能を使用して、アプリケーションのパフォーマンスを最大限に引き出します。ただし、実行時に変更可能な SQL 用のパラメータを使用することによって、BC4J アプリケーションを Oracle 以外のデータベースで使用することもできます。特に、BC4J フレームワークの Oracle9*i* JDeveloper リリースは、（Merant DataDirect ドライバを使用して）IBM 社製の DB2 データベースおよび Microsoft 社製の SQL Server データベースに対してテスト済です。

使用していないフレームワーク機能による実行時オーバーヘッドの発生

回答 : オーバーヘッドは発生しません。BC4J フレームワークは、使用していないフレームワーク機能に対して実行時オーバーヘッドが発生しないように、慎重に設計および最適化されています。たとえば、BC4J エンティティ・オブジェクトが、ビジネス・ロジックをカプセル化し、永続性を処理するように設計されているとします。永続性を処理するためのみに BC4J フレームワークを使用する場合、データベース内の既存のデータベース・トリガーによってビジネス・ロジックが施行されるようにすると、使用していないビジネス・ロジックの施行によって実行時オーバーヘッドが発生することはありません。同様に、BC4J フレームワークでは、関心があるフレームワークのライフ・サイクル・イベントが発生した場合に開発者が通知を受信できる、様々な軽量のリスナーがサポートされています。この場合も、イベント・サブスクリプションが存在しない場合、関連するオーバーヘッドは発生しません。

BC4J に関する詳細情報の参照先

回答 : BC4J および JDeveloper の詳細は、次のサイトを参照してください。

```
http://jdeveloper.us.oracle.com
```

J2EE および EJB の開発者が、BC4J を使用して生産性を向上させる方法については、次のサイトを参照してください。

```
http://otn.oracle.com/products/jdev/htdocs/j2ee_bc4j.html
```

25

User Interface XML（UIX）の概要

この章の内容は次のとおりです。

UIX の概要

User Interface XML（UIX）は、Web アプリケーションを構築するフレームワークを構成する一連のテクノロジです。UIX は、主に、アプリケーションのユーザー・プレゼンテーション・レイヤーを対象としたテクノロジですが、イベントおよびアプリケーション・フローの状態を管理するための追加機能もあります。UIX は、統合開発環境（IDE）などの高度な対話を必要とするフル機能のアプリケーションではなく、ページ・ベースのナビゲーションが行われるオンライン人事管理アプリケーションなどを作成するために設計されています。

アプリケーションは、意思決定点と呼ばれる事前定義された場所で UIX と対話できます。この場所では、オペレータによって意思決定が行われるか、または特定のアクション・ルーチンが自動的にトリガーされます。アクションは、新しい意思決定点が検出されると終了します。アプリケーションの構造は、構成ファイル（ASCII ファイル、データベース、ソース・ファイルなど）として UIX に提供されます。

UIX には、Java クラス・ライブラリ、API、XML 言語、および Web ベースのアプリケーションの様々な要素を開発するためのその他のテクノロジが含まれます。開発する Web アプリケーションの要素によって、これらのテクノロジの一部またはすべてを使用できます。UIX テクノロジを最大限に活用するには、すべての UIX テクノロジについてよく理解しておくことが理想です。

UIX を使用する場合

次に、開発をより迅速に行うための UIX の機能を示します。

- UIX は、開発のためのオープンで柔軟なフレームワークを提供します。開発要件によって、様々な UIX テクノロジから選択できます。たとえば、ページのレンダリングには、UIX Components を使用できます。または、レンダリングに独自の HTML または JavaServer Pages（JSP）を使用し、UIX のその他の機能を利用することもできます。また、要件に適したすべてのバックエンドのデータ・テクノロジを使用できます。
- UIX テクノロジは、Java プログラミング言語およびその他の移植可能な Web テクノロジを使用して実装されるため、プラットフォームに依存しません。
- UIX は、広範囲のクライアント・エージェントをサポートします。また、様々なブラウザおよびロケール用に表示を調整します。携帯情報端末用のレンダリングもサポートします。
- UIX テクノロジ・スタックに作成されたアプリケーションでは、一貫した表現が保持されます。UIX レンダリング・プロジェクトでは、高水準のユーザー・インタフェース制御が実装されます。これは、アプリケーション（および UIX を使用するその他のアプリケーション）内で一貫してレンダリングされます。
- UIX アプリケーションは、複数レベルでカスタマイズされます。ページ・レイアウト、スタイル、イメージングなど、アプリケーションの多くの要素を個々に変更できます。環境によっては、単純なカスタマイズを簡単に行ったり、より複雑なカスタマイズを行うこともできます。

- UIX 開発の大部分を宣言方式で使用することもできます。これは、フレームワークを使用すると、プログラミングまたはコンパイルを行うことなく、ページ・レイアウト、スタイルおよびその他の多くの機能を XML 文書から派生させることができるためです。
- UIX アーキテクチャは、ローカライゼーションおよび国際化のサポートに留意して開発されています。UIX アーキテクチャのレンダリング・テクノロジでは、ターゲット・クライアントのロケールに対して自動的に調整が行われます。また、フレームワークは、ローカライズ可能なコンテンツをユーザー・インタフェースから分離できるように構築されています。
- フレームワークは、共有リソースのキャッシュおよび再利用など、高パフォーマンスを実現するように設計されています。

UIX を使用しない場合

UIX の使用に適さない場合があります。

- ターゲット・ユーザーの環境が Java 要件を満たしていない標準の Web ブラウザまたは携帯情報端末である場合、UIX の使用が適さない場合があります。これは、UIX が Java で構築されているため、アプリケーションの配置環境で Java Virtual Machine（JVM）がサポートされている必要があるためです。
- ユーザー・インタフェースにドラッグ・アンド・ドロップ、コード編集、ビジュアル設計などの高度な相互作用が必要である場合、クライアント側 Java など、UIX が提供するインタフェース・テクノロジよりさらに高度なユーザー・インタフェースを使用する必要があります。

UIX テクノロジの概要

UIX テクノロジを使用すると、Web アプリケーションのプレゼンテーション・レイヤー全体を実装できます。ただし、一部の機能のみが必要な場合、UIX のサブセットのみを使用できます。UIX は、Web アプリケーションの開発プロジェクトの様々な側面を対象とした「サブ製品」としてモジュール化されています。次に、各サブ製品の概要を説明します。

UIX Components

UIX Components は、ページ、特に Web アプリケーションのフロントエンド（ユーザー・インタフェース）として使用されるページのコンテンツを生成するためのクラス・ライブラリで構成されます。このテクノロジは、ページ間またはこれらのページに提供されているデータ間でのナビゲーションの管理は行いません。この操作は、その他のソース（その他の UIX テクノロジなど）によって行われます。UIX Components テクノロジは、主にページ自体のレンダリングを行います。このレンダリングでは、ブラウザ・ページの場合は HTML、携帯情報端末の場合は WML などのその他のテクノロジで出力できます。

UIX Components テクノロジは、ページ・レイアウト、および表、タブ、ボタンなどの標準ユーザー・インタフェース・オブジェクトを作成するための Web Bean（ノード）のコレクションを含めることによって、このレンダリングを行います。UIX Components テクノロジには、これらの Bean を使用してブラウザなどの特定のデバイスへ出力を生成するための一連のレンダリング・クラス（レンダラ）も含まれます。

UIX Components は、別のビジュアル・スタイル（ルックアンドフィール）を使用した同じページのレンダリングを可能にする、トランスポータブル・レンダリング・アーキテクチャを持ちます。デフォルトのレンダラは、Oracle Browser Look and Feel（BLAF）に準拠する HTML を出力しますが、携帯情報端末にはその他のレンダラを使用可能です。また、必要に応じて追加のレンダラを作成し、フレームワークに追加できます。

UIX Components をサポートするすべての Java コードおよびクラスは、`oracle.cabo.ui` パッケージおよびそのサブパッケージにあります。

UIX Controller

UIX Controller は、Web アプリケーション・フローを開発するためのフレームワークです。UIX Controller は、Java 2 Enterprise Edition の標準機能である Java サーブレット・テクノロジに基づいています。UIX Components は指定されたページのレンダリングを主に行うのに対し、UIX Controller はアプリケーション内のすべてのページ間でのナビゲーションを管理するように設計されています。これらのページのレンダリングは UIX Controller では行われず、その他のテクノロジ（UIX Components など）によって行われます。

UIX Controller は、アプリケーションが HTML イベントを処理する方法を標準化し、エラー・ページのループ、ログインのサポート、ファイルのアップロードなどの組込みサービスを提供します。UIX Controller は、UIX Components、JSP、eXtensible Stylesheet Language Transformations（XSLT）など、個々のページのレンダリングに使用されるテクノロジから独立して動作しますが、UIX Components などのテクノロジが使用される場合に開発を容易にする組込みサポートを提供します。

UIX Controller をサポートするすべての Java コードおよびクラスは、`oracle.cabo.servlet` パッケージおよびそのサブパッケージにあります。

UIX Language

UIX Language は、Java ベースの UIX Components の Bean または UIX Controller の Java コード（あるいはその両方）を使用して Web アプリケーションをプログラム的に作成するかわりに、宣言方式で使用できます。UIX Language は、UIX Components および UIX Controller の上に構築され、UIX Components のページ・レイアウトおよび UIX Controller のサーバー側イベントを指定するための XML 言語を提供します。基本的に、UIX Language を使用すると、Java プログラミングではなく XML 文書を使用して UIX Components のページおよび UIX Controller のイベントを作成できます。

UIX Language はページおよびページ・フローを作成する別の方法を提供しますが、これは UIX Components および UIX Controller に自動的に変換されるため、UIX による処理は同じになります。

UIX Language をサポートするすべての Java コードおよびクラスは、`oracle.cabo.ui.xml` パッケージ、`oracle.cabo.servlet.xml` パッケージおよびそのサブパッケージにあります。

UIX Dynamic Images

UIX Dynamic Images は、ボタンおよびタブの組込みサポートなど、テキストを含むイメージを生成するためのユーティリティを記述します。UIX Dynamic Images は、ローカライゼーションおよびアクセス性のサポート、およびパフォーマンスを向上させるためのキャッシュのサポートを提供する他に、アプリケーションのイメージをカラー化してカラー・スキームをサポートできます。UIX Dynamic Images は、イメージ、および必要に応じて、これらのイメージに対するイメージ・マップを生成します。

UIX Dynamic Images を使用すると、テキストがイメージとは別に処理されるため、より簡単および効率的にローカライゼーションを行うことができます。変換ツールではテキストのみが処理され、イメージの編集は行われません。変換されたテキストは（リソース・ファイルなどに）個別に格納し、必要に応じて取り出してイメージと組み合せることができます。このようにテキストとイメージを個別に処理することによって、漢字などの複雑な文字のテキスト・サイズを大きくする、色覚異常などの視覚障害を持つ人のためにビジュアルな属性を調整するなど、特別な目的のために個別のテキスト・スタイルやサイズを使用することができます。

UIX Dynamic Images をサポートするすべての Java コードおよびクラスは、`oracle.cabo.image` パッケージおよびそのサブパッケージにあります。UIX Components によってレンダリングされるイメージは、UIX Dynamic Images に依存します。

UIX Styles

UIX Styles は、様々なエンド・ユーザー環境（ロケール、ブラウザ、プラットフォームなど）に応じてスタイルシートを定義およびカスタマイズするためのアーキテクチャを提供します。スタイルシートは、ページの外観を、これらのページに含まれるコンテンツとは別に定義および変更するための、集中管理されたメカニズムを提供します。

UIX Styles には、環境固有のスタイルシートを定義するための新しい XML スタイルシート言語（XSS）が含まれます。XSS は、カスケーディング・スタイルシート（CSS）に基づいています。UIX Styles には、実行時に CSS スタイルシートを動的に生成する機能など、スタイルの情報を管理するためのサーバー側 API も含まれます。

UIX Styles をサポートするすべての Java コードおよびクラスは、`oracle.cabo.style` パッケージおよびそのサブパッケージにあります。UIX Components および UIX Dynamic Images のスタイル情報は、UIX Styles に依存します。

UIX Share

すべての UIX プロジェクトは、UIX Share によって提供される共通のユーティリティ・クラスに依存します。

UIX Share クラスには、構成のサポートおよびローカライゼーションなど、すべての UIX Web アプリケーションに有効な機能が含まれます。UIX Share をサポートするすべての Java コードおよびクラスは、`oracle.cabo.share` パッケージおよびそのサブパッケージにあります。

使用する UIX テクノロジ

UIX テクノロジ・スタックはオープンかつ柔軟であるため、必要に応じて多くのサブ製品を使用できます。ただし、いくつかの UIX サブ製品は、それ以外の UIX サブ製品とあわせて使用する必要があることに注意してください。たとえば、UIX Components では、ページのイメージをレンダリングするために UIX Dynamic Images を使用し、スタイルシートをレンダリングするために UIX Styles を使用するため、これらのテクノロジが搭載されている必要があります。ただし、これらのサブ製品は UIX Components によって内部的に使用されるため、ユーザーが使用する必要はありません。

ただし、様々な UIX テクノロジが併用されるように設計されているということに注意してください。ある UIX プロジェクトを実行することによって、別の UIX テクノロジをより簡単に使用できる場合があります。たとえば、UIX Controller では組込みサポートによって、UIX Language に指定されたページの作成およびキャッシュが自動で行われます。UIX Components を使用せずに UIX を使用する場合、UIX Language ドキュメントをロードおよび表示するコードを作成する必要があります。逆に、UIX Language を使用せずに UIX Components を使用する場合、独自のページの表示方法を UIX のフレームワークに通知するコードを作成する必要があります。そのため、UIX テクノロジを個別に使用するより、UIX テクノロジ・スタック全体を使用する方が適しています。

次に、使用するテクノロジに関する推奨項目を示します。

- Web アプリケーションを新規に作成する場合

  UIX Controller を使用してアプリケーション・フローを管理し、UIX Components および UIX Language を使用してページのレイアウトおよびイベントを指定することをお薦めします。これによって、最小限の作業で、最大限の UIX の機能を得ることができます。

- 既存のアプリケーション・フロー管理テクノロジを使用する必要があるが、ページのレンダリングは変更できる場合

  UIX Components および UIX Language を使用して、ページを作成およびレンダリングすることをお薦めします。これによって、UIX スタック全体を使用できない場合でも、UIX Components のメリット（エージェント・ベースのレンダリング・アーキテクチャ、高水準のページ Beans、ローカライゼーションなど）を得ることができます。ただし、UIX Controller のいくつかのコードは、UIX Controller 全体を使用しない場合でも、サーバー側フロー管理に使用可能であることに注意してください。たとえば、UIX Controller には、通常は Java サーブレット・ベースのアプリケーションに対して有効な、ファイルのアップロードを処理するためのユーティリティが含まれます。

- サーブレットを介して管理する必要があるページ（JSP または動的 HTML）が存在する場合

  ログイン管理およびエラー処理に UIX Controller サーブレットを使用し、基本的なサーブレット・アーキテクチャに含まれていないその他のユーティリティを補うことをお薦めします。これによって、UIX Components に基づいてページを簡単に追加できるようになります。

- HTML または JSP で設計されたページが存在するが、アプリケーションに新しいページを実装する必要がある場合

  すべてのページに UIX Components および UIX Language を使用することをお薦めします。UIX Components および UIX Language は、パススルー機能を使用して、既存の JSP および HTML などの他のコンテンツを同じページ上に配置する容易な方法を提供するためです。これによって、後でページを 1 つのテクノロジに統合できます。

- UIX Components Bean を使用する必要があるが、ページの他の部分にすでに Java ベースのレンダリングを使用している場合

  Java Web Bean クラスを介して同じページ上で UIX Components Bean を使用できます。生成された出力は、既存の Java ベースのページ出力にマージできます。ページの管理に UIX Controller を使用するかどうかは、この選択に依存しません。

- 現行のページのレンダリング・テクノロジは変更できないが、Web アプリケーションのイメージをローカライズする必要がある場合

  UIX Dynamic Images を使用して、ローカライズされるテキストを含むイメージを生成することをお薦めします。

- 現行のページ・レンダリング・テクノロジは変更できないが、各ビューアのブラウザ、ロケール、プリファレンス用に調整された製品に応じたスタイルシートが必要な場合

  UIX Styles を使用して、改良型に基づいて個々のスタイルシートを生成することをお薦めします。

UIX の詳細

次に、UIX に関する参照先を示します。

参照： UIX 用の JDeveloper デモンストレーション・コードの例については、次の Web サイトを参照してください。

- `http://otn.oracle.com/sample_code/products/jdev/content.html`
- 『UIX 開発者ガイド』の全文は、JDeveloper のオンライン・ヘルプに含まれています。

A

XDK for Java: 仕様およびクイック・リファレンス

この付録では、XDK for Java の仕様、および Java 用の各 XML コンポーネントのクイック・リファレンスを示します。クイック・リファレンスには、XDK for Java の各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 のクイック・リファレンス

注意： XML Parser for Java のメソッドは、次のリソースを参照してください。

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- http://technet.oracle.com/tech/xml
- doc/ 下にインストール済のソフトウェア

XML Parser for Java の仕様

Oracle XML Parser for Java 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 および 2 の API を統合します。
- SAX 1.0 API および 2.0 API を統合します。
- W3C の XML Namespace 勧告をサポートします。

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 xdk/doc/ ディレクトリにあります。

リリース固有の注意事項

ドキュメントのルート・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。

- eXtensible Markup Language（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）
- ドキュメント・オブジェクト・モデル（DOM）レベル 1
  （http://www.w3.org/TR/REC-DOM-Level-1/ を参照）
- ドキュメント・オブジェクト・モデル（DOM）レベル 2
  （http://www.w3.org/TR/DOM-Level-2-Core/ を参照）
- XML Path Language（XPath）1.0（http://www.w3.org/TR/1999/REC-xpath-19991116 を参照）
- XML Transformation（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。

- Simple API for XML（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（および XSQL Servlet）は現在、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 がリクエスト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は処理を継続します。

XDK for Java: XML Schema プロセッサ

参照 :

- 第 6 章「XML Schema Processor for Java」を参照してください。
- インストール済のソフトウェアの xdk/doc/java/schema ディレクトリにある readme.html 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.co.jp/tech/xml/xdk/index.htm からダウンロードすることもできます。

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。

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.co.jp/tech/xml/xdk/index.htm

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。

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、.¥xdk¥admin¥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 Pages 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、./xdk/admin）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 バージョン 2、Oracle XSU for Java 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・クラスにマップします。
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 をマップします（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 および 2 の 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。

B

XDK for PL/SQL: 仕様

この付録では、Oracle XDK for PL/SQL の仕様を説明します。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記憶単位で構成され、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 文書の読込み、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 レベル 1 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・ドリブン・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。
- Oracle9*i* および Oracle9*i* Application Server で実行できます。
- 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 プロセッサによるドキュメント作成のための DOM 拡張 API

Oracle XML Parser には、XSLT スタイルシートを使用して XML データを変換するための XSL Transformation（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 20 章「XML Parser for PL/SQL」を参照してください。

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込み方法の点で、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 は妥当性を検証します。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 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。Oracle9*i* 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 Oracle9*i* 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 xdk/doc/plsql/parser ディレクトリにあります。また、『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』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 eXtensible Markup Language（XML）1.0 勧告
  （http://www.w3.org/TR/1998/REC-xml-19980210 を参照）
- W3C のドキュメント・オブジェクト・モデル（DOM）レベル 1 勧告
  （http://www.w3.org/TR/REC-DOM-Level-1/ を参照）

パーサーは現在、SAX または Namespace をサポートしていません。将来のバージョンでサポートされる予定です。

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。

> **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 API リファレンス - XDK および Oracle XML DB』
> - 第 8 章「XML SQL Utility（XSU）」
> - `http://otn.oracle.co.jp/tech/xml/xdk/index.html`

用語集

9*i*FS

「Oracle 9iFS」を参照。

ACE

アクセス制御エントリ。「アクセス制御エントリ（Access Control Entry: ACE）」を参照。

ACL

アクセス制御リスト。「アクセス制御リスト（Access Control List: ACL）」を参照。

API

Application Program Interface。「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。JDeveloper に付属する J2EE アプリケーション開発フレームワーク。BC4J は、J2EE Design Patterns を実装するオブジェクト・リレーショナル・マッピング・ツールである。

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 Architecture（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 で作成された関数のライブラリ。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可能なコードを作成できる。

CORBA

「Common Object Request Broker Architecture（CORBA）」を参照。

CSS

「カスケーディング・スタイルシート（Cascading Style Sheets: CSS）」を参照。

DBUriType

データ型のインスタンスを格納するために使用される Oracle9*i* データベースのデータ型。このデータ型では、データベース・スキーマの XPath を使用したナビゲーションが可能になる。

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。

DOM 再現性（DOM fidelity）

データの整合性および精度を確認するため、Oracle XML DB に格納された XML 文書を再生成するときなどに、Oracle XML DB では DOM 再現性というデータ整合メカニズムが使用される。DOM 再現性とは、特に DOM 検索の目的で、戻された XML 文書が元の XML 文書と同一であることを意味する。Oracle XML DB は、バイナリ属性 `SYS_XDBPD$` を使用して DOM 再現性を確認する。

DTD

「Document Type Definition（DTD）」を参照。

EDI

電子データ交換。

Enterprise JavaBeans（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。

existNode

`XMLType` 内に `XPath` が存在するかどうかに基づいて TRUE または FALSE を戻す SQL 演算子。

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 World Wide Web Consortium（W3C）によって開発され、インターネットでの使用目的で設計された。

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 文書の表示方法を指定する。XSL は、スタイルシートを表す言語である。XSL は、次の 2 つのもので構成される。

- 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）

XSLT 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。「FOP」を参照。

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。

extract

`XMLType` として格納された XML 文書のフラグメントを取得する SQL 演算子。

FOP

XSL Formatting Objects によって駆動される出力フォーマッタ。FOP は、書式設定オブジェクト・ツリーを読み込んで、結果ページを指定された出力にレンダリングする Java アプリケーションである。出力形式には、現在、PDF、PCL、PS、SVG、XML（領域ツリー表現）、Print、AWT、MIF および TXT がサポートされている。主な出力ターゲットは PDF である。

HASPATH

Oracle Text に含まれ、特定の XPath の存在を確認するために XMLType データ型の問合せに使用される SQL 演算子。

HTML

「Hypertext Markup Language（HTML）」を参照。

HTTP

「Hypertext Transfer Protocol（HTTP）」を参照。

HTTPUriType

データ型のインスタンスを格納するために使用されるデータ型。このデータ型では、リモート・データベース内でデータベース・スキーマの XPath を使用したナビゲーションが可能になる。

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。

INPATH

Oracle Text に含まれ、特定の XPath の特定のテキストを検索するために XMLType データ型の問合せに使用される SQL 演算子。

***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 Oracle 内でのアクセス。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。

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。

J2EE

「Java 2 Platform Enterprise Edition（J2EE）」を参照。

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される。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 2 Platform Enterprise Edition（J2EE）

複数層のエンタープライズ・コンピューティングを定義する Java プラットフォーム（Sun 社）。

Java API for XML Processing（JAXP）

特定の XML プロセッサの実装に依存しない API を使用して、アプリケーションで XML 文書の解析および変換を可能にする。

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 Naming and Directory Interface（JNDI）

Sun 社が提供するプログラム・インタフェース。Java プログラムを、DNS、LDAP、NDS などのネーミング・サービスおよびディレクトリ・サービスに接続する。Oracle XML DB Resource API for Java/JNDI は、JNDI をサポートする。

Java Runtime Environment（JRE）

プラットフォーム上で Java Virtual Machine 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（JVM）

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、イントラネット上、Oracle9*i*AS などのアプリケーション・サーバー上、または Oracle などのデータベース・サーバー内で実行できる。

JavaBean

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。JavaBean ともいう。サーバー側の同等のモジュールは、Enterprise JavaBeans（EJB）という。「Enterprise JavaBeans（EJB）」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 サーバー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 で最初にリクエスト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。

JAXP

「Java API for XML Processing（JAXP）」を参照。

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、統合 UML クラス・モデラーなどを含む。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。

JNDI

「Java Naming and Directory Interface（JNDI）」を参照。

JVM

「Java Virtual Machine（JVM）」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 iStudio 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。

OASIS

「Organization for the Advancement of Structured Information（OASIS）」を参照。

Object Request Broker（ORB）

クライアント側のリクエスト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・リクエスト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、JCORB および EJB である。「Common Object Request Broker Architecture（CORBA）」を参照。

OC4J

Oracle9*i*AS Containers for J2EE。JDeveloper に付属する J2EE 開発ツール製品。

OCT

「Ordered Collection in Tables（OCT）」を参照。

OE

Oracle Exchange。

OIS

「Oracle Integration Server（OIS）」を参照。

Oracle 9*i*FS

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。

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 の製品。OIS には、AQ および Oracle Workflow を搭載した Oracle8i 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 JVM。JVM は、Oracle8*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 Text

Oracle のツール製品。XPath に類似した検索機能とともに、文書の全文索引および複数の文書にまたがった SQL 問合せを実行する機能を提供する。

Oracle XML DB

Oracle データベース・サーバーが提供する、高パフォーマンスな XML 格納および取得テクノロジ。W3C の XML データ・モデルに基づく。

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。

Oracle9*i* Application Server（Oracle9*i*AS）

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。

Oracle9*i*AS

「Oracle9i Application Server（Oracle9iAS）」を参照。

ORB

「Object Request Broker（ORB）」を参照。

Ordered Collection in Tables（OCT）

VARRAY の要素が異なる表に格納されている場合、これらの要素を Ordered Collection in Tables という。

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

SQL を拡張した、Oracle 手続き型言語。データベース内で実行可能なプログラムを作成するために使用する。

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。

Simple Object Access Protocol（SOAP）

非集中型の分散環境で情報を交換するための XML ベースのプロトコル。

SOAP

「Simple Object Access Protocol（SOAP）」を参照。

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。

SYS_XMLAGG

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。

SYS_XMLGEN

渡された SQL 問合せの結果を XML として戻すシステム固有の SQL 関数。これは、XMLType のインスタンス化にも使用できる。

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。

TransXUtility

翻訳されたシード・データおよびメッセージのデータベースへのロードを簡単にする Java API。

UDDI

「Universal Description, Discovery and Integration（UDDI）」を参照。

UIX

「User Interface XML（UIX）」を参照。

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。

Universal Description, Discovery and Integration（UDDI）

この仕様は、XML を使用してサービスの記述、ビジネスの検出およびインターネット上のビジネス・サービスの統合を行う、プラットフォーム非依存フレームワークを提供する。

URI

「Uniform Resource Identifier（URI）」を参照。

URL

「Uniform Resource Locator（URL）」を参照。

User Interface XML（UIX）

Web アプリケーションを構築するフレームワークを構成する一連のテクノロジ。

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。

WebDAV

「World Wide Web Distributed Authoring and Versioning（WebDAV）」を参照。

Web サービス記述言語（Web Services Description Language: WSDL）

Web サービスのインタフェース、プロトコル・バインドおよび配置の詳細を記述するための汎用 XML 言語。

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。

World Wide Web Distributed Authoring and Versioning（WebDAV）

Web 上での協調的なオーサリングのための Internet Engineering Task Force（IETF）標準。Oracle XML DB のフォルダリングおよびセキュリティ機能は、WebDAV に準拠している。

WSDL

「Web サービス記述言語（Web Services Description Language: WSDL）」を参照。

XDBbinary

バイナリ・データを含む Oracle XML DB スキーマで定義された XML 要素。XDBbinary 要素は、完全に構造化されていないバイナリ・データが Oracle XML DB にアップロードされたときに、リポジトリに格納される。

XDK

「XML Developer's Kit（XDK）」を参照。

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 Parser、XSLT プロセッサ、XML Class Generator、Transviewer JavaBeans および XSQL Servlet が含まれる。

XML Gateway

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。

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 プロセッサは、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。

XMLType

XMLType 列は、データベース内の基礎となる CLOB 列を使用して XML データを格納する。

XMLType ビュー（XMLType views）

Oracle XML DB は、既存のリレーショナル・データおよびオブジェクト・リレーショナル・データベースを XML 形式にラップする方法を提供する。特に、レガシー・データが XML ではなく、それを XML 形式に移行する必要がある場合などに有効。

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 文書の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。

アクセス制御エントリ（Access Control Entry: ACE）

アクセス制御リスト内のエントリ。指定されたプリンシパルへのアクセスを許可または禁止する。

アクセス制御リスト（Access Control List: ACL）

アクセス制御エントリのリスト。指定されたリソースへのアクセス権を所有するプリンシパルを判別する。

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。「Oracle9i Application Server（Oracle9iAS）」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、Oracle8 以上の 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。

階層的な索引付け（hierarchical indexing）

フォルダをその子に関連付けているデータは、Oracle XML DB の階層索引によって管理される。この索引によって、オペレーティング・システムのファイル・システムで使用されるディレクトリ・メカニズムに類似した、パス名の評価を高速化するメカニズムが提供される。パス名ベースのアクセスでは、通常、Oracle XML DB の階層索引が使用される。

カスケーディング・スタイルシート（Cascading Style Sheets: CSS）

スタイル（フォント、カラー、間隔など）を Web ドキュメントに追加するための単純なメカニズム。

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。

コンテンツ（Content）

リソースの本体。ファイルなどのリソースを処理する場合にその内容をリクエストすると戻されるもの。コンテンツは、常に XMLType である。

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、リクエスト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、同時実行（マルチスレッド）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。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。

遅延型変換（lazy type conversions）

Oracle XML DB によって使用されるメカニズム。Java アプリケーションが最初に Java 用の XML データをリクエストしたときに、そのデータのみを変換する。これによって、JDBC での型変換による一般的なボトルネックが低減される。

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 Oracle Net 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、日付言語などのグローバリゼーション・サポート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。

名前レベル・ロック（name-level locking）

Oracle XML DB では、コレクション・レベル・ロックではなく名前レベル・ロックが提供される。名前をコレクションに追加すると、コレクションには排他書込みロックが設定されず、コレクション内の名前のみがロックされる。名前の変更はキューに入れられ、コレクションはコミット時にのみロックされ、変更される。

ノード（node）

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。

パーサー（parser）

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文字列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。

パス名（pathname）

リポジトリ階層内でのリソースの位置を反映したリソースの名前。パス名は、ルート要素（最初の /）、要素セパレータ（/）および様々なサブ要素（またはパス要素）で構成される。パス要素は、（\ または /）を除いて、データベース・キャラクタ・セットのすべての文字で構成できる。これらの文字は、Oracle XML DB で特別な意味を持つ。スラッシュは、パス名内のデフォルトの名前セパレータであり、バックスラッシュは、エスケープ文字として使用される。

表記法宣言（Notation Attribute Declaration）

XML では、パーサーが認識できない内容の型の宣言。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。

ファンクション索引（function-based index）

データベース索引。これを作成すると、一般的な問合せの結果をより高速に戻すことができる。

フォルダ（Folder）

リソースを含む、または含むことができる Oracle XML DB リポジトリ内のディレクトリまたはノード。フォルダはリソースでもある。

フォルダリング（Foldering）

コンテンツをリソースの階層構造に格納できるようにする Oracle XML DB の機能。

プリンシパル（principal）

Oracle XML DB リソースへのアクセス制御権限を付与されたエンティティ。Oracle XML DB は、次のプリンシパルをサポートする。

- データベース・ユーザー。
- データベース・ロール。データベース・ロールは、グループとして認識される。たとえば、DBA ロールは、DBA ロールを付与されたすべてのユーザーの DBA グループを表す。

LDAP サーバーからインポートされたユーザーおよびロールも、データベースの通常の認証モデルの一部としてサポートされる。

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。

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。

リソース（resource）

リポジトリ階層内のオブジェクト。

リソース名（resource name）

親フォルダ内のリソースの名前。リソース名は、フォルダ内で一意（大 / 小文字が区別されない場合もある）である必要がある。リソース名のキャラクタ・セットは、常に UTF-8（NVARCHAR）である。

リポジトリ（repository）

パス名にマップされる、スキーマ内の一連のデータベース・オブジェクト。リポジトリには、それぞれパス名を持つ、一連のリソースを含む 1 つのルート（「/」）がある。

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

R

S

T

U

W

X

あ

い

す

せ

そ

た

ち

つ

て

と

な

の

は