

Oracle8*i*

アプリケーション開発者ガイド - XML Vol.1

リリース 8.1

2000 年 11 月

部品番号 : J02331-01

ORACLE®

部品番号 : J02331-01

原本名 : Oracle8i Application Developer's Guide -XML, Volume 1 Release 3 (8.1.7)

原本部品番号 : A86001-01

原本著者 : Shelley Higgins

原本協力者 : Sandeepan Banerjee, Kishore Bhamidipati, Stefan Buchta, Dipto Chakravarty (Artesia Technologies, Inc.), Robert Dell'immagine, Brajesh Goyal, Robert Hall, Karun K, Stefan Kiritzo, Vishu Krishnamurthy, Murali Krishnaprasad, Olivier LeDiouris, Bryn Llewellyn, Roger Medlin (Artesia Technologies, Inc.), Steve Muench, Visar Nimani, Paul Nock, Rajesh Raheja, Tomas Saulys, Mark Scardina, Flora Sun, Prabhu Thukkaram, Ari Adler, Omar Alonso, Phil Bates, Mark Bauer, Ravinder Booreddy, Steve Cave, Steve Corbett, Claire Dessaux, Janet Lee, Shailendra Mishra, Andy Page, Rahul Pathak, Padmini Ranganathan, Den Raphaely, Jim Rawles, David Saslav, Chitra Sharma, Ena Singh, Keith Swartz, Kurt Thompson, Melanie Watson, Jon Wilkinson

グラフィック・デザイナー : Valerie Moore

Copyright © 1996, 2000, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation（米国オラクル）または日本オラクル株式会社（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

目次

はじめに	xix
このマニュアルの内容	xx
対象読者	xxi
前提知識	xxi
関連マニュアル	xxi
機能範囲および可用性	xxi
このマニュアルの構成	xxii
このマニュアルの表記規則	xxv
略語	xxv

第 I 部 Oracle の XML の概要

1 Oracle の XML の概要

概要	1-3
Oracle の XML の使用 : アプリケーション	1-3
作成済 XML または生成される XML	1-3
Oracle の XML アプリケーション	1-3
このマニュアルのロードマップ	1-5
Oracle の XML	1-7
Oracle の XML の概要	1-7
Oracle の XML コンポーネント	1-7
Oracle の XML コンポーネントの動作	1-10
Oracle8i の XML の機能	1-10
Oracle8i の XML を使用する理由	1-13
統合ツールとしての Oracle Suite	1-13

Oracle JDeveloper 3.2 および Oracle Business Objects for Java (BC4J)	1-13
Internet File System (iFS)	1-14
Portal (WebDB)	1-14
Oracle Exchange	1-15
その他の取組み	1-15
Oracle <i>interMedia</i> Text を使用した XML ドキュメントの索引付けおよび検索	1-16
メッセージング・ハブおよび中間層コンポーネント	1-16
バックエンド、データベース、フロントエンドの統合	1-17
Oracle XML Parser が提供する API: DOM および SAX	1-18
カスタムの XML アプリケーションの作成	1-18
データベースへの XML のロード	1-19
Oracle8i からの XML データの格納および抽出	1-20
Oracle8i: オブジェクト・リレーショナル・インフラストラクチャ	1-20
Oracle8i: 拡張可能なアーキテクチャ	1-20
Oracle8i によるネイティブ Java のサポート	1-21
データベース内の XML: 生成される XML および作成済 XML	1-22
生成される XML	1-22
例: XML SQL Utility を使用した SQL 問合せ結果の XML 表現の取得	1-23
XML 記憶領域オプション	1-25
LOB 形式での格納: 作成済 XML ドキュメントの格納	1-25
オブジェクト・リレーショナル形式での格納: 生成される XML ドキュメントの格納	1-25
構造化 XML ドキュメント (生成された XML) の格納	1-26
XML 構造を保持した XML SQL Utility による XML データの格納	1-26
例: XML SQL Utility の Java API を使用したデータベースへの XML の挿入	1-26
XML ドキュメントのオブジェクト・リレーショナル形式での格納: メリット	1-27
XML ドキュメント・オブジェクト・リレーショナル形式での格納: デメリット	1-27
非構造化 XML ドキュメント (作成済 XML) の格納	1-28
<i>interMedia</i> Text の例: CONTAINS を使用したテキストおよび XML データの検索	1-28
マッピングの細分化を改善するためのハイブリッドな格納方法の使用	1-29
ハイブリッドな格納によるユーザー定義の細分化での格納	1-29
ハイブリッドな格納のメリット	1-30
生成される XML: XML の変換	1-31
XSLT を使用した問合せ結果の変換	1-31
XML Schema およびドキュメントのマッピング	1-31
XML Schema の例 1: 単純なデータ型の定義	1-32

XML Schema の例 2: XML Schema を使用した基礎となるスキーマへの 生成された XML ドキュメントのマッピング	1-32
作成済 XML: ドキュメントとしての XML ドキュメントの格納	1-34
XML ドキュメントの生成および格納	1-35
ビューを使用した XML ドキュメントとデータの組合せ	1-36
Web フォームのインフラストラクチャの生成方法	1-36
XML SQL Utility による格納	1-36
データ交換アプリケーションにおける一般的な XML 設計の問題	1-37
マッピングの細分化を改善するためのハイブリッドな格納方法の使用	1-37
Web フォームからデータベースへの XML データの送信	1-37
アプリケーション間での XML ドキュメントの通信	1-38
Oracle の XML のサンプルおよびデモ	1-39
Oracle の XML コンポーネントの実行要件	1-39

2 Oracle の XML を使用したビジネス・ソリューション

Oracle の XML を使用するアプリケーション	2-2
コンテンツおよびドキュメントの管理	2-3
データ表示のカスタマイズ	2-3
使用例 1 - コンテンツおよびドキュメントの管理:	
Oracle の XML を使用した複合ドキュメントの作成および公開	2-6
使用例 2 - コンテンツおよびドキュメントの管理:	
Oracle の XML を使用した個人情報の配信	2-7
使用例 3 - コンテンツ管理:	
XML を使用したデータ駆動のアプリケーションのカスタマイズ	2-9
B2B/B2C メッセージ機能	2-10
XML を使用した B2B/B2C メッセージ機能: 使用例	2-10
使用例 4 - B2B メッセージ機能:	
XML を使用した複数ベンダー用のオンライン・ショッピング・カート設計	2-11
使用例 5 - B2B メッセージ機能:	
インターネット・アプリケーションのための XML および Oracle AQ の使用	2-13
使用例 6 - B2B メッセージ機能:	
XML および Oracle AQ メッセージ機能を使用した複数アプリケーションの統合	2-15

3 Oracle の XML コンポーネントおよび一般的な FAQ

Oracle の XML コンポーネント: 概要	3-2
開発ツールおよび XML 対応のその他の Oracle8i 機能	3-3

XDK for Java	3-3
XDK for PL/SQL	3-3
XML Parser	3-4
XSL Transformation (XSLT) Processor	3-6
XML Class Generator	3-6
XML Transviewer Beans	3-7
Oracle XSQL Servlet	3-8
XSQL Servlet をサポートするサーブレット・コンテナ	3-9
XSQL Servlet をサポートする Java Server Pages (JSP) プラットフォーム	3-9
Oracle XML SQL Utility (XSU)	3-12
問合せ結果からの XML の生成	3-13
XML ドキュメントの構造: 要素への列のマッピング	3-13
Oracle <i>interMedia</i> Text	3-15
Oracle の XML アプリケーションを構築するためのツール	3-16
Oracle の XML コンポーネント: XML ドキュメントの生成	3-17
Oracle の XML コンポーネントを使用した XML ドキュメントの生成: Java	3-17
Oracle の XML コンポーネントを使用した XML ドキュメントの生成: PL/SQL	3-19
FAQ - XML 全般	3-21
XML の記述方法	3-21
XML および必要な Oracle のツール	3-21
XML 形式の発注書の収集: XML での RFP の作成	3-22
移植性: 異なるベンダーの XML パーサーの使用	3-23
XML をサポートするブラウザ	3-23
Oracle 8.0.x に対する XML サポート	3-24
EDI および XML	3-24
B2B の取引をサポートする Oracle のツール	3-25
XML に関するオラクル社の方針	3-26
XML および BLOB (XML メッセージ内)	3-27
CLOB の最大サイズ	3-27
Oracle 7.3.4: XML を使用した他のベンダーへのデータ転送	3-27
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	3-28
XML アプリケーションの構築に必要なソフトウェア	3-28
受注や出荷に使用する標準の DTD	3-29
DTD からデータベース・スキーマへの移行	3-29
XML へのスキーマのマッピング	3-30
データベース内の XML のパフォーマンス	3-30

より高速なレコード取出し	3-30
他の形式から XML への変換	3-31
XML ファイルのサイズ制限	3-31
XML ドキュメントの最大サイズ	3-32
Rational Rose ツールからのデータベース・スキーマの生成	3-32
追加情報	3-33
XML に関するその他の FAQ	3-33
XML/XSL 関連の推奨書籍	3-33

第 II 部 XML SQL Utility (XSU) : データベースに対する XML の格納および取出し

4 XML SQL Utility (XSU) の使用

XML SQL Utility へのアクセス	4-3
XML SQL Utility (XSU) の使用	4-4
XML SQL Utility (XSU) を使用する場合	4-4
XML SQL Utility (XSU) を実行できる場所	4-5
データベース内での XML SQL Utility の実行	4-5
中間層内での XML SQL Utility の実行	4-6
Web サーバー内での XML SQL Utility の実行	4-7
XML SQL Utility の動作要件	4-8
XSU の機能	4-8
XSU 使用のガイドライン	4-9
マッピングの手引き	4-9
マッピング: SQL からの XML の生成	4-9
マッピング: XML からの SQL での表への格納	4-12
XSU 用のコマンドラインのフロントエンドの使用	4-14
XSU のフロントエンドを使用した XML の生成	4-15
XSU のフロントエンドを使用した XML の挿入	4-17
XML SQL Utility for Java	4-19
XML の生成	4-19
XSU: SQL 問合せからの XML の基本的な生成	4-20
XSU の例 1: emp 表からの文字列の生成	4-20
XSU の例 2: emp 表からの DOM の生成 (Java)	4-23
結果ページの区切り: skipRows および maxRows	4-25
XSU の例 3: 結果ページの区切り: コール時の XML ページの生成 (Java)	4-26

ResultSet オブジェクトからの XML の生成	4-28
XSU の例 4: JDBC の ResultSets からの XML の生成 (Java)	4-28
XSU の例 5: プロシージャの戻り値 (REF CURSOR) からの XML の生成 (Java)	4-30
該当する行がない場合の例外の呼出し	4-31
XSU の例 6: 該当する行がない場合の例外 (Java)	4-32
XML の格納	4-32
挿入処理	4-33
XSU の例 7: すべての列への XML 値の挿入 (Java)	4-34
XSU の例 8: 特定の列への XML 値の挿入 (Java)	4-35
更新処理	4-36
XSU の例 9: keyColumn を使用した更新 (Java)	4-36
XSU の例 10: 指定された列のリストの更新 (Java)	4-37
削除処理	4-39
XSU の例 11: ROW ごとの削除操作 (Java)	4-39
XSU の例 12: 指定したキー値の削除 (Java)	4-40
XML SQL Utility for PL/SQL の使用	4-41
DBMS_XMLQuery を使用した XML の生成	4-41
XSU の例 13: 単純な問合せからの XML の生成 (PL/SQL)	4-41
XSU の例 13a: 出力バッファへの CLOB の出力	4-42
XSU の例 14: ROW タグ名および ROWSET タグ名の変更 (PL/SQL)	4-42
XSU の例 15: setMaxRows() および setSkipRows() を使用した結果のページ区切り	4-43
XSU へのスタイルシートの設定 (PL/SQL)	4-44
XSU のバインド値 (PL/SQL)	4-45
XSU の例 15a: SQL 文への値のバインド	4-45
DBMS_XMLSave を使用したデータベースへの XML の格納	4-46
PL/SQL での XSU による挿入処理	4-47
XSU の例 16: すべての列への値の挿入 (PL/SQL)	4-47
XSU の例 17: 特定の列への値の挿入 (PL/SQL)	4-48
更新処理	4-49
XSU の例 18: keyColumns を使用した XML ドキュメントの更新 (PL/SQL)	4-50
XSU の例 19: 更新する列のリストの指定 (PL/SQL)	4-50
削除処理	4-51
XSU の例 20: ROW ごとの削除操作 (PL/SQL)	4-51
XSU の例 21: キー値の指定による削除 (PL/SQL)	4-52
XSU の例 22: コンテキスト・ハンドルの再使用 (PL/SQL)	4-53
高度な使用方法	4-55

Java での例外処理	4-55
PL/SQL での例外処理	4-56
FAQ: XML SQL Utility (XSU)	4-57
XSU を使用して XML を格納する場合のスキーマ構造	4-57
複数表への XML データの格納	4-58
XML SQL Utility を使用した属性に格納された XML のロード	4-59
XML SQL Utility の大 / 小文字区別: 大 / 小文字区別なし機能の使用など	4-59
DTD からのデータベース・スキーマの生成	4-60
XML SQL Utility コマンドラインの使用	4-60
INSERT、DELETE、UPDATE 後の XML SQL Utility のコミット	4-60

第 III 部 XML を使用したコンテンツおよびドキュメントの管理

5 *interMedia Text* を使用した XML ドキュメントのデータ検索および取得

概要	5-2
<i>interMedia Text</i> の概要	5-2
<i>interMedia Text</i> のインストール	5-3
<i>interMedia Text</i> のユーザーおよびロール	5-3
CONTAINS 演算子を使用した問合せ	5-4
CONTAINS の例 1	5-5
CONTAINS の例 2: ラベルを指定した SCORE 演算子の使用	5-5
CONTAINS の例 3: SCORE 演算子の使用	5-5
この章の例に関する前提	5-6
<i>interMedia Text</i> を使用した XML ドキュメントの検索	5-7
<i>interMedia Text</i> 索引	5-7
CTX_DDL PL/SQL パッケージの使用	5-8
各 CTX パッケージに必要なロールのリスト表示	5-8
CTX_DDL プロシージャ	5-9
XML_SECTION_GROUP 属性セクション	5-10
AUTO_SECTION_GROUP	5-12
<i>interMedia Text</i> 索引の作成	5-13
必要なロールの決定および ctxapp 権限の付与	5-13
データの設定 (ない場合)	5-13
CONTAINS を使用するための <i>interMedia Text</i> 索引の作成	5-14
プリファレンスの作成: プリファレンスを使用したパラメータ化の表現	5-15

プリファレンスのパラメータ化	5-15
<i>interMedia Text</i> の例 1: 索引の作成 - プリファレンスの作成および適切なパラメータ化	5-16
<i>interMedia Text</i> の例 2: AUTO_SECTION_GROUP を使用したセクション・グループの作成	5-17
<i>interMedia Text</i> の例 3: XML_SECTION_GROUP を使用したセクション・グループの作成	5-18
<i>interMedia Text</i> を使用した問合せアプリケーションの構築	5-19
テキスト問合せ式	5-19
<i>interMedia Text</i> の例 4: テキスト問合せ式の使用	5-19
属性セクションでの問合せ	5-26
属性セクションの問合せに対する制約	5-26
SECTION GROUPS の問合せ	5-28
DOCTYPE 間のタグの区別	5-28
DOCTYPE 制限の指定によるタグの区別	5-28
セクション・グループ内での DOCTYPE 制限タグおよび未制限タグの混在	5-29
<i>interMedia Text</i> を使用した問合せアプリケーションの作成手順	5-30
CTX_OBJECTS 表および CTX_OBJECT_ATTRIBUTES ビューの使用	5-30
プリファレンスの作成	5-32
プリファレンスの属性の設定	5-32
CTX_DDL.add_zone_section の使用	5-33
CTX_DDL.Add_Attr_Section の使用	5-33
CTX_DDL.add_field_section の使用	5-34
CTX_DDL.add_special_section の使用	5-36
CtX_DDL.Add_Stop_Section の使用	5-36
問合せ構文の作成	5-37
<i>interMedia Text</i> の例 4: ドキュメントの問合せ	5-38
<i>interMedia Text</i> の例 5: 索引の作成およびテキスト問合せの実行	5-40
ドキュメント・タイプが区別される XML ドキュメントでのセクションの作成	5-42
セクションの繰返し	5-42
セクションのオーバーラップ	5-43
セクションのネスト	5-43
問合せ結果の表示	5-44
FAQ: <i>interMedia Text</i>	5-45
<i>interMedia Text</i> を使用した XML データの挿入および検索	5-45
<i>interMedia Text</i> : 属性の処理	5-45
CTXSYS/CTXSYS の ID およびパスワード	5-46
XML ドキュメントの問合せ	5-46
<i>interMedia Text</i> および Oracle8i	5-47

<i>interMedia</i> を使用した XML の索引付け	5-48
<i>interMedia Text</i> を使用した CLOB の検索	5-48
様々な DTD を使用した様々な XML ドキュメントの管理 : CLOB への XML の格納および CLOB での XML の検索 - <i>interMedia Text</i>	5-49
<i>interMedia Text</i> のロール (ORA-01919: ロール CTXSYS は存在しません。)	5-51
XML ドキュメントの検索およびゾーンの取得	5-51
CLOB への XML ドキュメントの格納 : <i>interMedia Text</i> の使用	5-52
データベースへの XML ドキュメントのロードおよび <i>interMedia Text</i> を使用した検索	5-53
WITHIN 演算子を使用した XML の検索	5-54
XML および <i>interMedia Text</i>	5-54
<i>interMedia Text</i> および XML: Add_field_section	5-55
<i>interMedia</i> および XML のサポート	5-55
Oracle8i Lite 4.0.0.2.0: <i>interMedia Text</i> の非サポート	5-57
<i>interMedia</i> コンテキストでの SQL	5-58
XML および <i>interMedia Text</i>	5-59
3 つの列に対する索引の作成	5-59
構造化 / 非構造化データの検索	5-60

6 XML によるコンテンツのカスタマイズ : Dynamic News アプリケーション

Dynamic News アプリケーションの概要	6-2
Dynamic News の主なタスク	6-2
Dynamic News アプリケーションの概要	6-2
Dynamic News の SQL の例 1: 項目スキーマ nisetup.sql	6-4
Dynamic News のサーブレット	6-4
Dynamic News の動作 : 鳥瞰図	6-5
静的なページ	6-6
一部動的なページ	6-8
動的なページ	6-10
コンテンツのパーソナライズ	6-11
エンド・ユーザー設定項目の取得	6-12
データベースからのニュース項目の取出し	6-16
ドキュメント構築のためのニュース項目の組合せ	6-18
表示のカスタマイズ	6-19
ニュース項目のインポートおよびエクスポート	6-22

7 XML によるデータ表示のパーソナライズ : Portal-to-Go

Oracle Portal-to-Go の概要	7-2
Portal-To-Go 1.0.2 の機能	7-3
Portal-to-Go 実行の要件	7-3
Portal-to-Go: サポートするデバイスおよびゲートウェイ	7-4
Portal-to-Go の動作	7-5
Portal-to-Go コンポーネント	7-6
Portal-to-Go サービス	7-6
Portal-to-Go Adapter	7-7
Portal-to-Go Transformer	7-8
XML でのデータ交換 : Portal-to-Go を使用したソースから XML、XML からターゲットへの配信	7-9
コンテンツの取出し	7-10
Web Integration Developer: スクリーン・スクレーパ	7-12
XML への変換	7-13
中間 XML フォーマットを使用する理由	7-13
Simple Result DTD の使用	7-13
Adapter による DTD 要素へのソース・コンテンツのマッピング	7-16
サンプルのアダプタ・クラス	7-18
Portal-to-Go Adapter の例 1: 株相場およびヘッドラインの XML への変換	7-18
Portal-to-Go Adapter の例 2: ユーザーの名前ごとの対応	7-19
ターゲットのマークアップ言語への XML の変換	7-22
Portal-to-Go: Java Transformer	7-23
Portal-to-Go Java Transformer の例 1: 他のフォーマットへの Simple Result 要素の変換	7-24
Portal-to-Go: XSL Stylesheet Transformer	7-26
Portal-to-Go XSL Stylesheet Transformer の例 1: プレーン・テキストへの Simple Result ドキュメントの変換	7-26
固有のトランスフォーマを必要とする各マークアップ言語	7-27
Portal-To-Go Stylesheet Transformer の例 2: WML1.1 トランスフォーマ用スタイルシートの カスタマイズ	7-29
Portal-to-Go の例 1: オンライン・ドラッグストアの販売市場の拡大	7-30
Portal-to-Go の例 2: 銀行サービスの拡大	7-30

8 XML および XSQL を使用した表示のカスタマイズ : Flight Finder

XML Flight Finder のサンプル・アプリケーション: 概要	8-2
必要なソフトウェア	8-2
Flight Finder の動作	8-3

Flight Finder のデータベースへの問合せ: 結果の XML への変換	8-6
XSQL Servlet を使用した問合せの処理および XML での結果の出力	8-7
スタイルシートを使用した XML のフォーマット	8-10
1 つのスタイルシートに対する 1 つのターゲット・デバイス	8-10
複数のスタイルシートに対する複数のターゲット・デバイス	8-12
出力のローカライズ	8-14
データベースへの XML の書込み	8-17
ユーザー入力の取得	8-17
ユーザーから取得した値のコード・パラメータへの割当て	8-19
処理成功のユーザーへの通知	8-19
Oracle Portal-to-Go	8-21

第 IV 部 XML を使用したデータ交換

9 Oracle Advanced Queuing を使用した XML データの交換

AQ の概要	9-2
AQ と XML の相互補完	9-2
AQ の例 1 (PL/SQL) : AQ メッセージの CLOB としての XML メッセージ	9-4
AQ 環境の設定	9-5
AQ の例 1: 実行されるタスク	9-5
AQ の例 1: PL/SQL コード	9-6
AQ の例 2 (Java) : JMS (パブリッシュ/サブスクライブ) を使用した XML メッセージの処理	9-8
AQ の例 2: JMS を使用した XML メッセージの処理 - 実行されるタスク	9-8
AQ の例 2: JMS を使用した XML メッセージの処理 - Java コード	9-9
FAQ: XML およびアドバンスト・キューイング	9-11
複数のフォーマットのメッセージ: オブジェクト型の作成および単一メッセージとしての格納 ..	9-11
メッセージがエンキューされた後の新しい受信者の追加	9-12
XML コンテンツを持つ JMS クライアントの AQ からの取出しおよび解析	9-13
エンキューが成功したかどうかの確認	9-14

10 B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供

Oracle Internet Procurement (iProcurement) の概要	10-2
様々なサプライヤによる総合カタログ表へのカタログのロード	10-2
Oracle Internet Procurement ソリューション	10-2
Internet Procurement および関連製品の詳細	10-3

バイヤーが提供するコンテンツ	10-3
サプライヤが提供するカタログおよび市場	10-5
iProcurement: XML Parser for Java	10-6
バイヤーが提供するカタログ	10-9
文書型定義 (DTD)	10-9
iProcurement の例 1: バイヤーが提供するカタログの DTD	10-10
DTD 管理情報: <ADMIN>	10-11
DTD スキーマ情報: <SCHEMA>	10-12
iProcurement の例 2: DTD<SCHEMA>: カテゴリおよび記述子のカテゴリへの追加	10-13
iProcurement の例 3: DTD<SCHEMA>: カテゴリまたは記述子の削除	10-13
iProcurement の例 4: DTD<SCHEMA>: カテゴリまたは記述子の更新	10-14
DTD: 品目情報	10-15
iProcurement の例 5: DTD ITEM: <ITEM ACTION="ADD"> を使用した品目の追加	10-16
iProcurement の例 6: DTD ITEM: <ITEM ACTION="DELETE"> を使用した品目の削除	10-16
iProcurement の例 7: DTD ITEM: <ITEM ACTION="UPDATE"> を使用した品目の更新	10-17
サプライヤが提供するカタログおよび市場	10-18
データ要素の定義	10-18
定義	10-19
標準コード	10-20
契約データ要素	10-20
品目データ要素	10-21
カテゴリ・データ要素	10-22
価格データ要素	10-23
サプライヤ・データ要素	10-24
追加属性	10-25
注文明細の XML 定義	10-26
すべてのデータを囲む CDATA タグ	10-26
iProcurement の例 8: 注文明細の XML Schema	10-26
iProcurement の例 9: XML: 完全なスキーマ仕様を含む 1 つの注文明細	10-34
iProcurement の例 10: XML: 2 品目のトランザクションの例	10-36
HTML 仕様	10-39
選択された品目の iProcurement への送信: 外部カタログの HTML ファイル形式	10-39
HTML 要素の説明	10-40
iProcurement の例 11: HTML/XML ファイル	10-41
ユーザー認証	10-42
iProcurement の例 12: 有効なセッションの XML ドキュメント	10-43

認証済 XML Schema	10-44
iProcurement の例 13: 認証済 XML Schema: 戻された注文ユーザーの XML ドキュメント	10-44
iProcurement の例 14: 認証済ユーザー: 戻された XML ドキュメント例	10-45
認証されていない XML Schema	10-46
iProcurement の例 15: 非認証ユーザー: XML Schema	10-46
iProcurement の例 16: 非認証ユーザー: XML サンプル・ドキュメント	10-46

11 XML メッセージ機能を使用した Phone Number Portability

Phone Number Portability のメッセージ機能の概要	11-2
Phone Number Portability アプリケーション構築の要件	11-3
SDP における Number Portability およびメッセージ機能アーキテクチャ	11-4
Communication Protocol Adapter	11-4
Order Processing Engine	11-4
Workflow Engine	11-5
Fulfillment Engine	11-5
Event Manager	11-5
SDP リポジトリ	11-5
Number Portability のプロセス	11-6
新しい電話サービスを注文した場合の処理	11-6
ローカル・サービス・プロバイダを変更した場合の処理	11-7
データ形式である XML: アドバンスト・キューイングの使用	11-8
このメッセージ機能に XML が使用される理由	11-8
ネットワーク要素の設置	11-9
Event Manager を使用したメッセージの非同期送受信	11-9
メッセージを送信するコード例	11-10
Internet Message Studio (iMessage) を使用したアプリケーションのメッセージ・セットの作成	11-11
コードの生成	11-11
メッセージ・セットの定義	11-11

第 V 部 Oracle の XML アプリケーションの開発

12 B2B XML アプリケーション: ステップ・バイ・ステップ

B2B XML アプリケーションの概要	12-2
B2B XML アプリケーション実行の要件	12-2
B2B XML アプリケーションの構築の概要	12-3

XML にデータを変換する理由	12-5
アドバンスト・キューイング (AQ) を使用する理由	12-6
B2B XML アプリケーション: 主なコンポーネント	12-7
B2B XML アプリケーションを実行するためのタスクの概要	12-8
B2B XML アプリケーションの実行のための環境設定	12-9
B2B アプリケーションの実行	12-11
B2B アプリケーション・セッションの終了	12-11
XML B2B アプリケーション: データベース・スキーマの設定	12-12
SQL のコール順序	12-13
小売業者およびサプライヤのスキーマの作成および構築	12-14
AQ 環境およびキュー表の作成	12-20
XSL Stylesheet 表を含む Broker スキーマの作成	12-22
環境のクリーン・アップおよびアプリケーション再実行の準備	12-25
キューを停止する SQL スクリプト	12-27
キューを削除する SQL スクリプト	12-27
キューを作成する SQL スクリプト	12-28
キューを開始する SQL スクリプト	12-28
dropOrder.sql	12-29
B2B XML アプリケーション: データ交換フロー	12-30
小売業者とサプライヤ間のトランザクション	12-31
小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする	12-31
小売業者が発注する	12-31
小売業者が確認して注文を送信する	12-32
AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する	12-32
サプライヤ・アプリケーションが、再フォーマット済の受信 XML 注文ドキュメントを解析し、 サプライヤ・データベースに注文を挿入する	12-33
サプライヤ・アプリケーションが保留注文をサプライヤに通知する	12-33
AQ Broker Transformer が小売業者の形式に従って XML 注文を変換する	12-34
小売業者アプリケーションが Ord 表および Line_Item 表を更新する	12-34
B2B XML アプリケーションの実行: 詳細手順	12-35
1 小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする	12-36
2 小売業者が発注する	12-48
3 「Validate」をクリックしてトランザクションをコミットし、小売業者アプリケーションが XML 注文を作成する	12-49
4 AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する	12-70

5 サプライヤ・アプリケーションがXMLドキュメントを解析し、サプライヤのデータベースに注文を挿入する	12-73
6a サプライヤ・アプリケーションが保留注文をサプライヤに通知する	12-74
6b サプライヤが商品を小売業者に出荷する	12-76
6c サプライヤ・アプリケーションがAQ Brokerに送信する新しいXMLメッセージを生成する	12-77
7 AQ Broker Transformer が小売業者の形式にXML注文を変換する	12-78
8 小売業者アプリケーションがOrd表を更新し、小売業者に新しい注文状態を表示する	12-79
B2B XML アプリケーションの停止	12-80
vieworder.sqlを使用した注文状態の確認	12-80
Java の例 - コール順序	12-81
XSL および XSL 管理スクリプト	12-82
XSL スタイルシートの例 1: HTML への変換 - html.xml	12-82
XSL スタイルシートの例 2: Palm Pilot ブラウザ用の変換 - pp.xml	12-88
Java の例 3: スタイルシートの管理 - GUIInterface.java	12-94
Java の例 4: GUIInterface_AboutBoxPanel.java	12-111
Java の例 5: GUIStylesheet.java	12-112
XSL 処理および管理スクリプト	12-114
Java の例 6: Main4XMLtoDMLv2.java	12-114
Java の例 7: ParserTest.java	12-117
Java の例 8: TableInDocument.java	12-119
Java の例 9: XMLFrame.java	12-120
Java の例 10: XMLProducer.java	12-122
Java の例 11: XMLtoDMLv2.java	12-123
Java の例 12: XMLGen.java	12-131
Java の例 13: XMLUtil.java	12-133
Java の例 14: XSLTWrapper.java	12-134
B2B XML アプリケーションで使用する他のスクリプト	12-140
XML の例 1: XSQL 構成 - XSQLConfig.xml	12-140
Java の例 15: メッセージ・ヘッダー・スクリプト - MessageHeaders.java	12-146
Java の例 16: Message Broker による使用定数の保持 - AppCste.java	12-147
小売業者のスクリプト	12-147
Java の例 17: 小売業者によるサプライヤからの更新状態待機 - UpdateMaster.java	12-147
AQ Broker Transformer およびアドバンスド・キューイングのスクリプト	12-155
Java の例 18: 1つのAQスレッド上でのAQ Brokerのリスニング - BrokerThread.java	12-155
Java の例 19: MessageBroker.java	12-160

Java の例 20: AQReader.java	12-166
Java の例 21: AQWriter.java	12-168
Java の例 22: B2BMessage.java	12-171
Java の例 23: ReadStructAQ.java	12-172
Java の例 24: StopAllQueues.java	12-173
Java の例 25: WriteStructAQ.java	12-174
サプライヤのスクリプト	12-177
Java の例 26: SupplierFrame.java	12-177
Java の例 27: エージェントによる小売業者からの注文受信の通知 - SupplierWatcher.java	12-183

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

JDeveloper 3.2 の概要	13-2
Business Components for Java (BC4J)	13-2
Oracle JDeveloper の XML 計画	13-3
JDeveloper 3.2 の動作環境	13-3
JDeveloper 3.2 の入手方法	13-3
BC4J での XML	13-4
BC4J を使用した XSQL クライアントの構築	13-6
Object Gallery	13-6
XSQL Element Wizard	13-8
Page Selector Wizard	13-9
JDeveloper 3.2 の XML 機能	13-10
Oracle XDK と Transviewer Beans の統合	13-10
Oracle XML Parser for Java	13-11
Oracle XSQL Servlet	13-11
XML Data Generator Web Bean	13-13
Portal-To-Go および JDeveloper を使用したモバイル・アプリケーション開発	13-13
JDeveloper を使用した XML アプリケーションの構築	13-14
JDeveloper の XML の例 1: BC4J メタデータ	13-14
JDeveloper 3.2 でのアプリケーションの構築手順	13-14
JDeveloper の XML Data Generator Web Bean の使用	13-15
JDeveloper での XSQL Servlet の使用	13-17
JDeveloper の XSQL の例 2: emp 表の従業員データ : emp.xsql	13-17
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	13-19
JDeveloper でのモバイル・アプリケーションの作成	13-20

BC4J アプリケーションの作成	13-21
BC4J アプリケーションに基づいた JSP ページの作成	13-22
データ読込みに必要なデバイスに従った XSLT スタイルシートの作成	13-23
FAQ: JDeveloper を使用した XML アプリケーションの構築	13-26
JSP での XML ドキュメントの作成	13-26
BC4J での XMLData の使用	13-27
JDeveloper 3.0 での XML Parser for Java の実行	13-27
複雑な XML ドキュメントのデータベースへの移動	13-30

14 Internet File System (iFS) を使用した XML アプリケーションの作成

Internet File System (iFS) の概要	14-2
iFS での XML の使用	14-2
ドキュメント記述子の供給	14-2
iFS パーサーの使用	14-3
標準の iFS パーサーおよびカスタム・パーサー	14-3
iFS 標準パーサーの使用	14-4
解析オプション	14-5
iFS カスタム・パーサーの使用	14-5
iFS XML の解析方法	14-5
パーサー・アプリケーションの作成	14-6
iFS での XML のレンダリング	14-7
XML およびビジネス・インテリジェンス	14-7
XML ファイルを使用した iFS の構成	14-7

15 XML を使用したリッチ・メディア管理用の n 層アーキテクチャの構築 : ArtesiaTech

n 層アーキテクチャ構築の概要	15-2
XML ベースの複数層通信	15-2
機能の移動 : 論理層と物理層の分離	15-4
XML を使用したオブジェクト指向のメッセージ機能	15-4
メッセージ機能への XML 使用の必要性	15-6
XML または IDL の使用の選択	15-6
XML のメッセージ処理	15-8
XML を使用したスクリプト	15-9
層内の通信	15-12
Web ベース・サービスのトランザクション状態管理	15-13

XML ベース・ソフトウェアの品質保証	15-13
XML ベースのデータ送信	15-14
XML 中心のデジタル資産管理	15-15
エンド・ユーザーのパーソナライズを容易にするデジタル資産集計	15-15
帯域幅の問題に対処するデジタル資産集計	15-17
まとめ	15-17

用語集

索引

はじめに

この章の内容は次のとおりです。

- [このマニュアルの内容](#)
- [対象読者](#)
- [前提知識](#)
- [関連マニュアル](#)
- [機能範囲および可用性](#)
- [このマニュアルの構成](#)
- [このマニュアルの表記規則](#)
- [略語](#)

このマニュアルに記載されているオラクル製品は、一部日本では出荷されていないものを含みます。ご利用の際は、必ず事前に日本オラクルまたは販売代理店にご確認ください。

また、C/C++ に関しましては、日本において対象外の機能です。

このマニュアルの内容

このマニュアルでは、Oracle の XML テクノロジ、およびその実装方法を説明します。また、例やサンプル・アプリケーションなども記載しています。例は、次のような主な機能（主な使用目的）に基づいて示されます。

- コンテンツまたはドキュメントの管理
- データ交換および B2B

Oracle の XML コンポーネントを使用して、データベース・ベースの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、アプリケーション推奨使用例を示し、XML コンポーネントの併用方法について説明します。

例およびサンプル・コード

このマニュアルで示す多くの例では、`$ORACLE_HOME/xdk/java/demo/` または `$ORACLE_HOME/xdk/java/demo/sample/` ディレクトリ、あるいは `$ORACLE_HOME/rdbm/demo` ディレクトリにあるソフトウェアを例として使用しています。

第 12 章「[B2B XML アプリケーション: ステップ・バイ・ステップ](#)」では、1 つのアプリケーションについて詳細に説明しています。このアプリケーションでは、XML データの変換およびカスタマイズされた表示アプリケーションの実装方法について説明します。

事前作成済 XML または生成される XML

XML ドキュメントは、次のいずれかの方法で処理されます。

- LOB に格納された事前作成済 XML
- 表内の関連列にマップされた XML タグを持つ、リレーショナル表に格納されて生成される XML

XML コンポーネント

Oracle の XML コンポーネントは、次の 2 つの言語での実装が可能です。

- Java: XDK for Java および XML SQL Utility for Java を使用します。
- PL/SQL: XDK for PL/SQL および XML SQL Utility for PL/SQL を使用します。

各 XML コンポーネントの使用方法については、[第 4 章「XML SQL Utility \(XSU\) の使用」](#)、第 VI 部および第 VII 部を参照してください。

対象読者

このマニュアルは、Oracle8i で XML アプリケーションを構築する開発者を対象にしています。

前提知識

このマニュアルを読むには、XML および XSL を理解しているのが理想ですが、必須ではありません。参考のため、付録に XML の手引きを記載しています。

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

関連マニュアル

詳細は、次のドキュメントを参照してください。

- Oracle JDeveloper オンライン・ヘルプ
- 『Oracle8i アプリケーション開発者ガイド 基礎編』
- 『Oracle8i 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』
- 『Oracle8i Integration Server 概要』
- 『Oracle8i XML リファレンス・ガイド』

機能範囲および可用性

このマニュアルの情報は、Oracle の XML テクノロジ・コンポーネントの現在の情報を表します。これらの情報は常に更新されています。

このマニュアルの構成

このマニュアルは7部編成で、20の章および5つの付録で構成されています。索引および用語集も含まれます。

第I部：OracleのXMLの概要

- 第1章「[OracleのXMLの概要](#)」では、OracleのXMLコンポーネント、XMLアプリケーションの構築に使用するツールおよび *interMedia Text* について説明します。また、Oracle8iでXMLアプリケーションを構築する場合の問題についても説明します。この章には、このマニュアルに記載されている情報へのロードマップが含まれます。
- 第2章「[OracleのXMLを使用したビジネス・ソリューション](#)」では、OracleのXMLコンポーネントを、典型的なコンテンツ / ドキュメント管理アプリケーションおよび B2Bメッセージ機能アプリケーションに使用する方法を簡単に説明します。
- 第3章「[OracleのXMLコンポーネントおよび一般的なFAQ](#)」では、OracleのXMLコンポーネント、XML Developer's Kit および XML SQL Utility の概要を説明します。また、Java および PL/SQL 用にXMLドキュメントを生成するための様々な方法の概要も示します。さらに、OracleのXMLに関する一般的な質問を含む、FAQも示します。

第II部：XML SQL Utility (XSU)

- 第4章「[XML SQL Utility \(XSU\) の使用](#)」では、Java および PL/SQL 用の XML SQL Utility を使用した XML ドキュメントの生成および格納方法、データベース内の XML ドキュメントに対する INSERT/UPDATE/DELETE の実行方法、コマンドライン・ツールの使用方法、および列に要素をマップする方法について説明します。この章に記載されている例は、\$ORACLE_HOME/rdbms/demo/xsu にあります。この章では、FAQも示します。

第III部：XMLを使用したコンテンツおよびドキュメントの管理

- 第5章「[interMedia Textを使用したXMLドキュメントのデータ検索および取得](#)」では、*interMedia Text*、CONTAINS 演算子の使用方法、*interMedia Text* 索引の作成方法、問合せの作成方法およびテキスト問合せ式について説明します。また、提供された PL/SQL パッケージ CTX_DDL および XML_SECTION_GROUP とその属性、AUTO_SECTION_GROUP の基本についても説明します。この章では、FAQも示します。
- 第6章「[XMLによるコンテンツのカスタマイズ：Dynamic News アプリケーション](#)」では、Dynamic News アプリケーション、アプリケーションで使用される3つのサーブレット、XML SQL Utility を使用して Oracle8i のニュース・データにアクセスする方法、およびユーザーの3つのカスタマイズ・レベル（静的、一部動的および動的）について説明します。また、データ表示のカスタマイズ方法も説明します。

- 第7章「XMLによるデータ表示のパーソナライズ: Portal-to-Go」では、Portal-to-Go コンポーネント、これを使用してブラウザの Web サイトのコンテンツを抽出する方法、抽出したコンテンツを XML に変換する方法、およびこのコンテンツを変換して様々なデバイスで表示する方法について説明します。
- 第8章「XML および XSQL を使用した表示のカスタマイズ: Flight Finder」では、Flight Finder というアプリケーションがデータベースとの間で XML を生成する方法、および XSQL Servlet を使用して問合せを処理し、結果を XML として出力する方法について説明します。また、Flight Finder がスタイルシートを使用して XML データのフォーマットを変換する方法についても説明します。

第 IV 部: XML を使用したデータ交換

- 第9章「Oracle Advanced Queuing を使用した XML データの交換」では、アドバンスド・キューイング (AQ) の概念、および AQ と XML が互いにどのように補足し合うかについて説明します。また、Java AQ および PL/SQL AQ の例も示します。FAQ も示します。
- 第10章「B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供」では、iProcurement の主なコンポーネント、および iProcurement が受け取ったサード・パーティ製カタログを XML Parser for Java を使用して解析および確認し、カタログのコンテンツを管理する方法について説明します。iProcurement で使用される DTD についても説明します。統合されたカタログは、データベースから抽出され、ロードされた後、PL/SQL プログラムを使用して Oracle Applications に送信されます。
- 第11章「XML メッセージ機能を使用した Phone Number Portability」では、Phone Number Portability アプリケーションの概要を説明し、XML メッセージ機能を iMessage Studio、Event Manager および Adapter で使用する方法について説明します。

第 V 部: Oracle の XML を使用したアプリケーション開発

- 第12章「B2B XML アプリケーション: ステップ・バイ・ステップ」では、XSQL Servlet を使用して B2B XML アプリケーションを構築および実装する方法、および様々なユーザーのデバイスに基づいて XML メッセージを変換する方法について説明します。このアプリケーションは、単純な AQ メッセージ機能も使用します。
- 第13章「JDeveloper を使用した Oracle の XML アプリケーションの作成」では、JDeveloper を使用して XML アプリケーションを構築する方法、JDeveloper での XSQL Servlet の使用方法、および JDeveloper を使用してモバイル・アプリケーションを構築する手順について説明します。FAQ も示します。
- 第14章「Internet File System (iFS) を使用した XML アプリケーションの作成」では、Internet File System (iFS) および XML の機能について説明します。
- 第15章「XML を使用したリッチ・メディア管理用の n 層アーキテクチャの構築: ArtesiaTech」では、ビデオ・クリップなどのデジタル資産を管理するための、複数層の高度な XML メッセージ機能アーキテクチャについて説明します。XML のオブジェクト指向のメッセージ機能について説明し、XML と IDL を比較します。

第 VI 部 : XDK for Java

- 第 16 章「[XML Parser for Java の使用](#)」では、XML Parser for Java および XSLT Processor の使用方法を説明します。ソフトウェアに含まれる例もリストしています。FAQ も示します。
- 第 17 章「[XML Class Generator for Java の使用](#)」では、XML Class Generator for Java の使用方法を説明します。ソフトウェアに含まれる例もリストしています。FAQ も示します。
- 第 18 章「[XSQL Servlet の使用](#)」では、XSQL Servlet の使用方法を説明します。図を使用して、XSQL Page Processor の動作についても示します。FAQ も示します。
- 第 19 章「[XML Transviewer Beans の使用](#)」では、XML Transviewer Beans およびその使用方法を説明します。ソフトウェアに含まれる例もリストしています。

第 VII 部 : XDK for PL/SQL

- 第 20 章「[XML Parser for PL/SQL の使用](#)」では、XML Parser for PL/SQL および XSLT Processor の使用方法を説明します。ソフトウェアに含まれる例もリストしています。FAQ も示します。

[付録 A「XML の手引き](#)」では、XML に関する基本情報および背景情報を説明します。

[付録 B「Oracle XML Parser および Class Generator の言語間比較](#)」では、実装言語に基づいて、Oracle XML Parser と Class Generator を比較します。

[付録 C「XDK for Java: 仕様および早見表](#)」では、XDK for Java コンポーネントの仕様を説明します。いくつかのトップレベル・クラスおよびメソッドもリストしています。

[付録 D「XDK for PL/SQL: 仕様および早見表](#)」では、XDK for PL/SQL コンポーネントの仕様を説明します。いくつかのトップレベル・ファンクションもリストしています。

[付録 E「XML SQL Utility \(XSU\) の仕様および早見表](#)」では、XML SQL Utility (XSU) for Java および XSU for PL/SQL の仕様を説明します。いくつかのトップレベル・メソッドおよびファンクションもリストしています。

このマニュアルの表記規則

このマニュアルで使用する表記規則は次のとおりです。

本文

このマニュアルの本文は、次の表記規則に従って記述されています。

大文字	大文字は、SQL キーワード、ファイル名および初期化パラメータを示します。
イタリック	イタリックは、SQL 文のパラメータを示します。
太字	太字は、用語の定義を示します。
固定幅フォント	固定幅フォントは、メソッド、ファンクションおよびその他の実行可能ファイルの名前を示します。

略語

次に、このマニュアルで使用される略語を示します。これらの用語の詳細は、「[用語集](#)」を参照してください。

頭字語	説明
API	アプリケーション・プログラム・インタフェース
B2B	Business-to-Business アプリケーション
B2C	Business-to-Consumer アプリケーション
BC4J	Business Components for Java
CDF	チャンネル定義形式 (Channel Definition Format) インターネット上のチャンネルに関する情報を交換する方法を提供する。
CSS	カスケーディング・スタイルシート
DTD	文書型定義 (Document Type Definition)
DOM	ドキュメント・オブジェクト・モデル
EDI	電子データ交換
HTML	ハイパー・テキスト・マークアップ言語
OAG	オープン・アプリケーション・グループ
OAI	Oracle Applications Integrator
OE	Oracle Exchange

頭字語	説明
PDA	パーソナル・デジタル・アシスタント
RDBMS	リレーショナル・データベース管理システム
RDF	リソース記述フレームワーク
SAX	Simple API for XML
SGML	標準一般化マークアップ言語 (Standard Generalized Mark Up Language)
W3C	World Wide Web Consortium
WBEM	Web ベース・エンタープライズ管理
XDK	Oracle XML Developer's Kit
XLink	XML リンク言語
XML	拡張可能マークアップ言語
XMLQuery	W3C が取り組む XML ドキュメントに対する問合せ言語の指定
XML Schema	W3C の仕様 XML Schema Processor は、XML ドキュメントおよびデータの妥当性を自動的に保証する。
XPath	XML ドキュメントを操作するためのデータ・モデルおよび文法を指定した W3C の勧告
XPointer	XML ドキュメント内の個別のエンティティまたはフラグメントの識別を指定した W3C の勧告
XSL	(W3C) 拡張可能スタイルシート言語 (eXtensible Stylesheet Language)
XSL スタイルシート	クラスのインスタンスの変換方法を記述して、XML ドキュメントのクラス表示を指定する。
XSLT	XSL Transformation

第 I 部

Oracle の XML の概要

第 I 部では、XML、Oracle の XML とその機能、Oracle の XML コンポーネントの用途、デモンストレーションと使用例、および Oracle の XML コンポーネント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。

- [第 1 章「Oracle の XML の概要」](#)
- [第 2 章「Oracle の XML を使用したビジネス・ソリューション」](#)
- [第 3 章「Oracle の XML コンポーネントおよび一般的な FAQ」](#)

Oracle の XML の概要

この章の内容は次のとおりです。

- 概要
- Oracle の XML の使用 : アプリケーション
- このマニュアルのロードマップ
- Oracle の XML
- Oracle8i の XML を使用する理由
- 統合ツールとしての Oracle Suite
- Oracle8i からの XML データの格納および抽出
- Oracle8i: オブジェクト・リレーショナル・インフラストラクチャ
- Oracle8i: 拡張可能なアーキテクチャ
- データベース内の XML: 生成される XML および作成済 XML
- XML 記憶領域オプション
- 構造化 XML ドキュメント (生成された XML) の格納
- 非構造化 XML ドキュメント (作成済 XML) の格納
- マッピングの細分化を改善するためのハイブリッドな格納方法の使用
- 生成される XML: XML の変換
- XML Schema およびドキュメントのマッピング
- 作成済 XML: ドキュメントとしての XML ドキュメントの格納
- XML ドキュメントの生成および格納

-
- データ交換アプリケーションにおける一般的な XML 設計の問題
 - Oracle の XML のサンプルおよびデモ
 - Oracle の XML コンポーネントの実行要件

概要

このマニュアルでは、Oracle8i の XML 対応のデータベース・テクノロジー、Oracle8i データベースでの XML データの格納、管理および問合せ方法、およびアプリケーションを構築するための Oracle8i の XML テクノロジーと適切な開発ツールの使用方法について説明します。

Oracle の XML の使用 : アプリケーション

作成済 XML または生成される XML

このマニュアルでは、Oracle の XML テクノロジーを使用して、データベースの XML アプリケーションを構築するために有効な情報を中心に説明します。アプリケーションでは、通常、次のどちらかの形式の XML ドキュメントが処理されることに注意してください。

- **作成済 XML ドキュメント**: XML ドキュメントとして、データベースの CLOB に格納された XML ドキュメントです。
- **生成される XML ドキュメント**: リレーショナル・データとして、データベースの表またはビューに格納された XML ドキュメントです。このデータは、必要に応じて、動的に XML 形式に再生成が可能です。

Oracle の XML アプリケーション

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

コンテンツおよびドキュメントの管理

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

参照： コンテンツおよびドキュメントの管理については、次の章を参照してください。

- 第 6 章「XML によるコンテンツのカスタマイズ : Dynamic News アプリケーション」
- 第 7 章「XML によるデータ表示のパーソナライズ : Portal-to-Go」
- 第 8 章「XML および XSQL を使用した表示のカスタマイズ : Flight Finder」

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

B2B および B2C メッセージ機能には、ビジネス・アプリケーション間のデータ交換が伴います。これらのアプリケーションでは、通常、生成される XML ドキュメント、または生成されるドキュメントと事前作成済 XML ドキュメントの組合せが処理されます。

参照： B2B については、次の章を参照してください。

- 第 9 章「Oracle Advanced Queuing を使用した XML データの交換」
- 第 10 章「B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供」
- 第 11 章「XML メッセージ機能を使用した Phone Number Portability」
- 第 12 章「B2B XML アプリケーション : ステップ・バイ・ステップ」

このマニュアルのロードマップ

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 : ロードマップ」に、B2B および B2C メッセージ機能に加えて、コンテンツおよびドキュメント管理のための XML ベースのビジネス・ソリューションを示します。

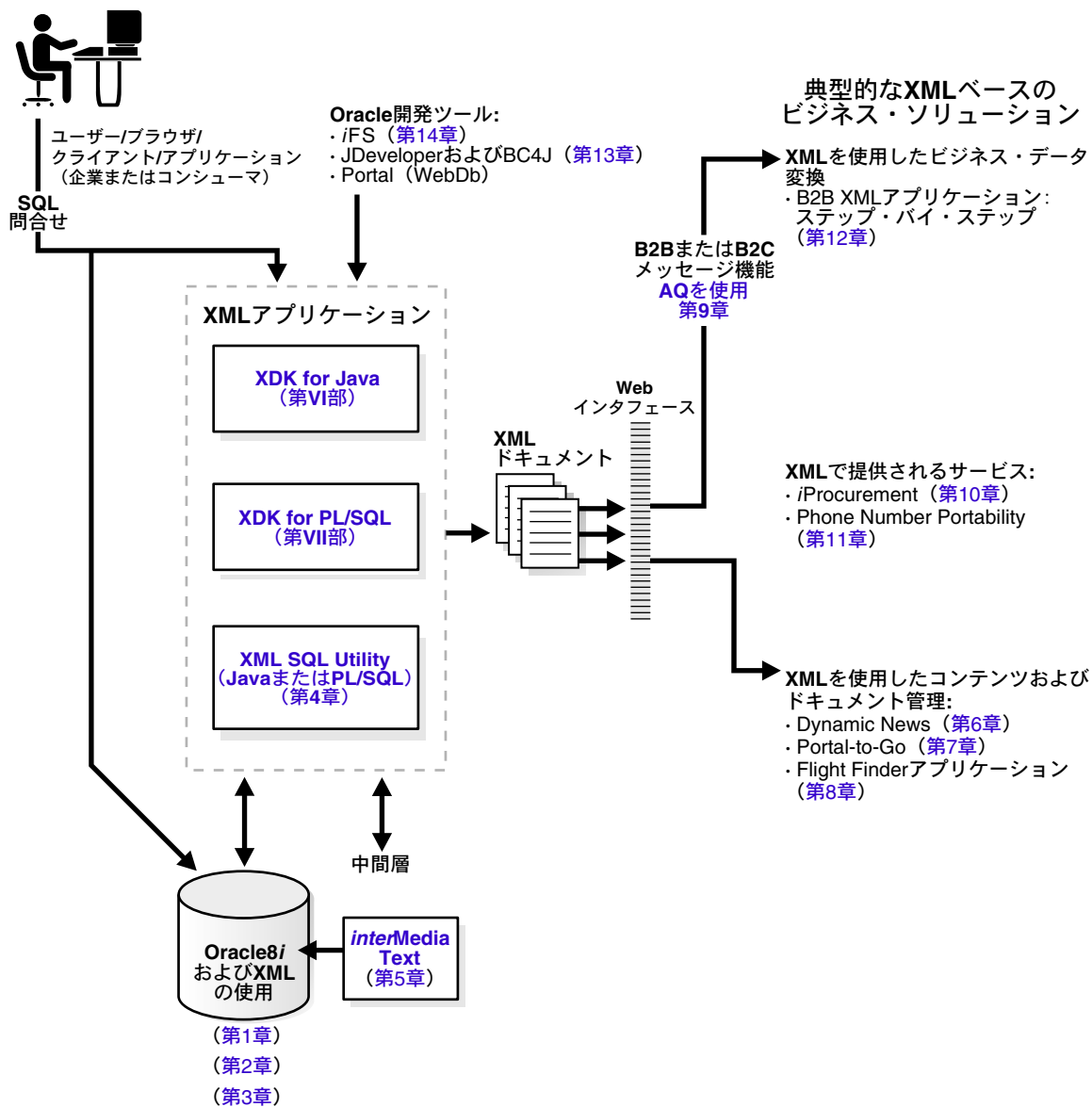
この図では、ユーザーがブラウザまたはアプリケーションから実行する SQL 問合せを示します。問合せは破線の中の「XML アプリケーション」で処理され、XML アプリケーションが Oracle8i にアクセスします。その後、XML ドキュメントが生成され、この XML ドキュメントが Web インタフェースを介して渡されるかどうかにかかわらず、データ交換ソリューションとして、サービスの提供、コンテンツやドキュメントの管理、同一ビジネス内の他のビジネスやアプリケーションに対する表示の多様化を行います。

図 1-1 では、Oracle の XML に含まれる主なコンポーネントの概要に加えて、このマニュアルのロードマップも示します。図には、マニュアルにおける主な参照先も示しています。

- まず、Oracle8i の使用および Oracle の XML コンポーネントの用途に関する概要および基本的な情報（第 1 章～第 3 章）、および *interMedia Text* を適用して、XML ドキュメントから情報を検索および取得する方法について説明します。
- 第 6 章～第 8 章では、コンテンツの管理および表示に関連する、典型的な XML ベースのビジネス・ソリューションについて説明します。
- 第 9 章～第 11 章では、B2B およびデータ交換に関連する、典型的な XML ベースのビジネス・ソリューションについて説明します。
- 第 12 章～第 14 章では、JDeveloper および iFS の使用に関する入門情報について説明し、コード中心のアプリケーション実装を、ステップごとに詳細に説明します。
- 最後に、最も重要な「XML アプリケーション」枠内に示す様々な Oracle の XML コンポーネントの使用方法について説明します。コンポーネントの使用方法は、第 4 章および第 15 章～第 20 章に記載されています。付録は、第 4 章および第 15 章～第 20 章の単なる捕捉であるため、ロードマップには記載されていません。

ロードマップに含まれていない第 15 章「XML を使用したリッチ・メディア管理用の n 層アーキテクチャの構築 : ArtesiaTech」では、N 層の XML アプリケーションを構築する方法について説明します。

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：ロードマップ



Oracle の XML

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

Oracle の XML の概要

XML は、構造化および半構造化データをモデル化し、インターネット・データに適したプライマリ・モデルとして認識されています。

Oracle8i は、複合データ、構造化データ、非構造化データおよび半構造化データをサポートするように拡張されています。また、XML に対応し、XML データの格納、問合せ、表示および操作を容易にします。

Oracle8i の XML による構造化および半構造化データのサポートの詳細は、次の章を参照してください。

- 第 4 章「XML SQL Utility (XSU) の使用」では、データベースから XML ドキュメントを生成する方法、およびデータベースにそのドキュメントを再度格納する方法について説明します。
- 第 15 章～第 19 章では、他の方法について説明します。特に、第 18 章「XSQL Servlet の使用」では、Oracle8i で Oracle の XML テクノロジおよびコンポーネントを使用する場合の実用的なガイドラインについて説明します。

Oracle の XML コンポーネント

図 1-2 に、「XML アプリケーション」枠内の Oracle の XML コンポーネントについて示します。

Oracle の XML コンポーネントは、次のコンポーネントで構成されます。

- XDK¹ for Java
 - XML Parser for Java および XSLT Processor
 - XML Class Generator for Java
 - XSQL Servlet
 - XML Transviewer Beans
 - XML Schema Processor²

¹ XDK - XML Developer's Kit

² Oracle XML Schema processor については、このマニュアルの将来のバージョンで説明する予定です。PL/SQL バージョンについては、今後リリースされる予定です。

- XDK for PL/SQL
 - XML Parser for PL/SQL
- XML SQL Utility (XSU) for Java および XSU for PL/SQL

図 1-2 では、次の XML ベースのビジネス・ソリューションについても示します。

■ **XML を使用したビジネス・データ交換**

- パイヤー / サプライヤ間の透過的な取引の自動化
- パートナのシームレスな統合および HTTP ベースのデータ交換
- データベース・インベントリへのアクセス、商用トランザクションおよびフローの統合
- XML で提供されるサービス
 - * Oracle iProcurement などを使用したセルフサービスによる調達
 - * Oracle Exchange および Oracle Application
 - * Phone Number Portability

これらのデータ交換には、Oracle Advanced Queuing (AQ) が使用されます。

■ **XML を使用したコンテンツおよびドキュメントの管理**

- パーソナライズされたパブリッシングおよびポータル
- カスタマイズされた表示 (Dynamic News の例、Portal-to-Go および Flight Finder)

図 1-2 Oracle の XML コンポーネントおよび E ビジネス・ソリューション：関連項目

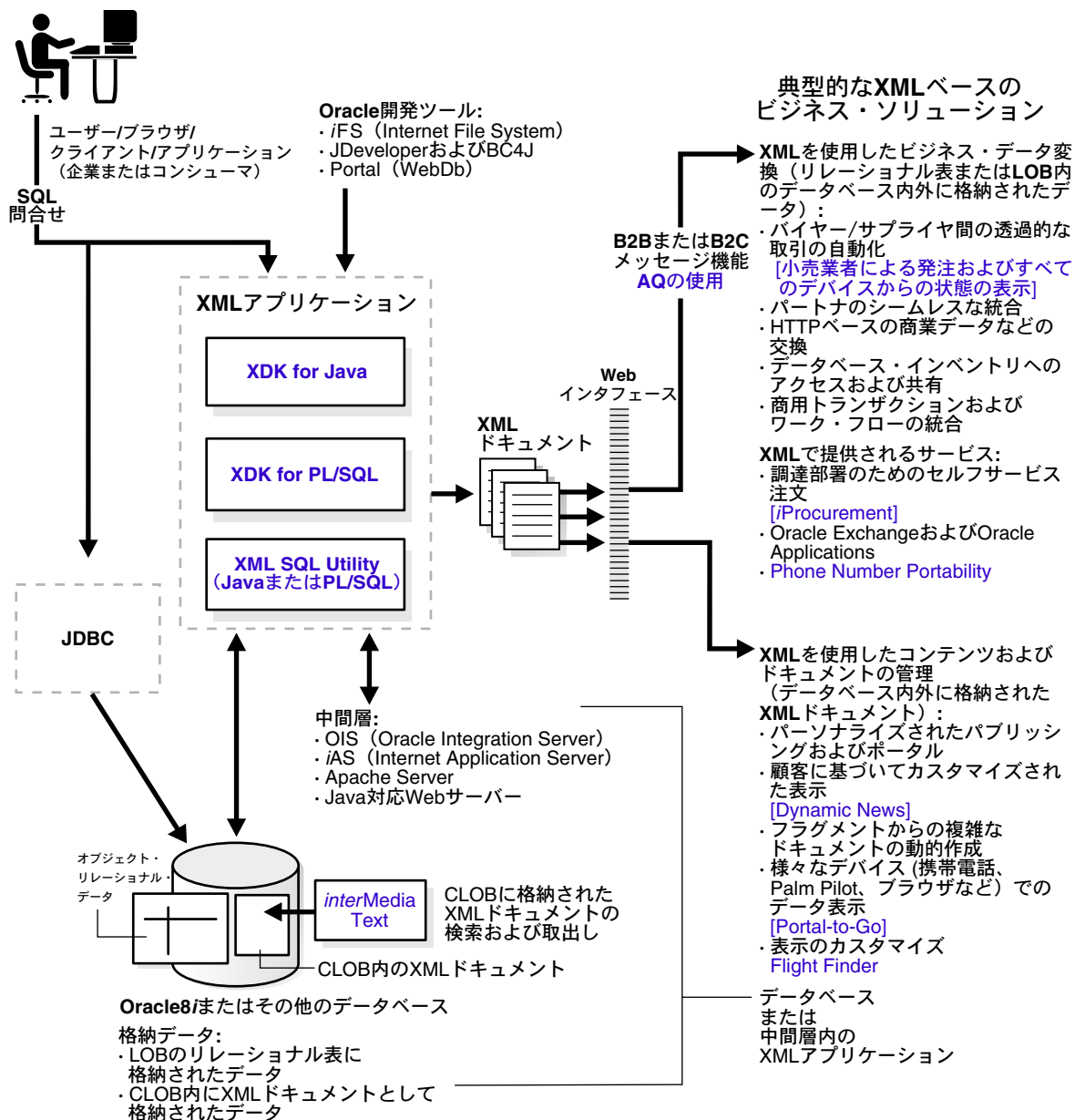


図 1-2 では、次のことについても示しています。

Oracle の開発ツール

Internet File System (iFS)、JDeveloper および Portal (WebDB) を使用して、XML アプリケーションを構築できます。これらのツールについては、1-13 ページの「[統合ツールとしての Oracle Suite](#)」を参照してください。

データベースおよび中間層

XML アプリケーションは、データベース、または OIS、iAS、Apache Server、その他の Java 対応 Web サーバーなどの中間層のどちらかに常駐できます。

データベースに格納されたデータ

データは、リレーショナル表または XML ドキュメントとして CLOB 内に格納されます。*interMedia Text* を使用して、CLOB に格納された XML ドキュメントを検索および取得します。

Oracle の XML コンポーネントの動作

Oracle の XML コンポーネントおよびコンポーネントの動作については、[第 2 章「Oracle の XML を使用したビジネス・ソリューション」](#) および [第 3 章「Oracle の XML コンポーネントおよび一般的な FAQ」](#) を参照してください。

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

Oracle8i の XML の機能

Oracle8i の XML の機能は次のとおりです。

- [XML オープン規格準拠](#)
- [配置およびアプリケーション](#)
- [複数の言語およびプラットフォームのサポート](#)
- [スケーラビリティおよびパフォーマンス](#)
- [アプリケーションおよびサーバーのサポート](#)

XML オープン規格準拠

Oracle の XML コンポーネントは、次の W3C の XML 勧告に基づいています。

- XML 1.0 Core
- DOM 1.0
- XSLT 1.0
- XML Namespaces 1.0
- XML Schema

このコンポーネントは、このマニュアルでは説明していません。将来のリリリースで説明する予定です。

- XPath

この勧告は、今回のリリリースの Oracle の XML では公開されていません。

W3C の活動の詳細は、<http://www.w3c.org> を参照してください。

配置およびアプリケーション

Oracle の XML テクノロジによって、柔軟な配置アーキテクチャでの使用が可能なインフラストラクチャが提供されています。次のものを組み合わせて使用します。

- リレーショナル・データベース
- 中間層システム
 - iAS (Internet Application Server) などの Web サーバー
 - Oracle Integration Server (OIS) などの統合サーバー
- Thin または Thick クライアント

複数の言語およびプラットフォームのサポート

Oracle の XML は、次のプラットフォームおよび言語をサポートします。

- プラットフォームに依存しない言語
 - Java および Java2
 - PL/SQL
- プラットフォーム
 - Solaris, Win32, Linux および HP-UX を含む、すべての Oracle8i プラットフォーム

スケーラビリティおよびパフォーマンス

Oracle8i の XML では、スケーラビリティおよびパフォーマンス関連の次の機能をサポートしています。

- 実行時のスタイルシートの適用によるオンデマンド表示
- サーバー側のプロセスを使用したクライアント / サーバー・パフォーマンス
- インテリジェント・キャッシュ

個別の XML コンポーネントは、スタイルシート、DTD およびドキュメントをキャッシュし、iCache などの中間層キャッシュとともにシームレスに動作します。

- 統合されたロード・バランシング

XML コンポーネントは層で構成されているため、ボトルネックを削減し、動的なロードが可能です。

- 接続プーリング

XML コンポーネントは、スケーラビリティを改善する接続プール管理のサポートを統合しています。

Oracle8i の XML を使用する理由

Oracle8i は、次の理由で XML データベース・アプリケーションの構築に適しています。

- [統合ツールとしての Oracle Suite](#)
- [Oracle interMedia Text を使用した XML ドキュメントの索引付けおよび検索](#)
- [メッセージング・ハブおよび中間層コンポーネント](#)
- [バックエンド、データベース、フロントエンドの統合](#)
- [Oracle XML Parser が提供する API: DOM および SAX](#)
- [データベースへの XML のロード](#)
- [Oracle の XML のサンプルおよびデモ](#)

統合ツールとしての Oracle Suite

Oracle8i では、E-Business アプリケーションを構築するために、次の統合ツール・スイートを提供しています。

- [Oracle JDeveloper 3.2 および Oracle Business Objects for Java \(BC4J\)](#)
- [Internet File System \(iFS\)](#)
- [Portal \(WebDB\)](#)
- [Oracle Exchange](#)
- [XML Gateway](#)

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

Oracle JDeveloper 3.2 および Oracle Business Objects for Java (BC4J)

Oracle JDeveloper 3.2 は、Oracle8i で Java および XML を使用して、アプリケーションを構築、配置およびデバッグするために統合された環境です。この環境によって、CORBA、EJB および Java ストアド・プロシージャでの Java1.1 または 1.2 の処理が簡単になります。

Oracle JDeveloper 3.2 を使用すると、次の操作が可能になります。

- Oracle の XML コンポーネントに直接アクセスして、複数層のアプリケーションを構築します。
- XML 情報を即時に処理する Java サーブレットを作成およびデバッグします。
- JDeveloper および BC4J コンポーネントを使用して、移植可能なアプリケーション論理を構築します。

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

- セルフサービスの Web 上での経費処理を含む iProcurement（セルフサービス・アプリケーション）。第 10 章「[B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供](#)」を参照してください。
- PDA へのコンテンツ配信。第 7 章「[XML によるデータ表示のパーソナライズ: Portal-to-Go](#)」を参照してください。
- オンライン・マーケットプレイス
- XML および AQ メッセージ機能を使用した小売業者 / サプライヤ間のトランザクション。第 12 章「[B2B XML アプリケーション: ステップ・バイ・ステップ](#)」を参照してください。

JDeveloper および XML アプリケーションの詳細は、第 13 章「[JDeveloper を使用した Oracle の XML アプリケーションの作成](#)」を参照してください。

Oracle Business Components for Java (BC4J) BC4J は、ビジネス・ロジックを再使用可能な Java コンポーネントのライブラリにカプセル化し、柔軟な SQL ベースの情報ビューを介してそのビジネス・ロジックを再使用するための、Oracle8i アプリケーション・フレームワークです。

Internet File System (iFS)

Oracle8i Internet File System (iFS) を使用すると、標準の Widows、および SMB、HTTP、FTP、SMTP、IMAP4 などのインターネット・プロトコルを介して、ファイルベースおよびフォルダベースの隠喩を使用したドキュメントおよびデータの編成およびアクセスが容易になります。

iFS を使用すると、Web ベースのアプリケーションを簡単に構築および管理できます。iFS は、Java 用アプリケーション・インタフェースであり、PowerPoint の PPT ファイルなどのドキュメントを Oracle8i にロードしたり、iAS や Apache Web サーバーなどの Web サーバーのドキュメントを表示することができます。第 14 章「[Internet File System \(iFS\) 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。

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

Portal (WebDB)

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

Oracle Portal の一部としてのポータル：Oracle Portal は、エンタープライズ・ポータルを構築および配置するためのソフトウェアです。エンタープライズ・ポータルとは、E-Business を実行するための Web サイトです。ブラウザ・インタフェースは、各ユーザーに必要なビジネス情報、Web コンテンツおよびアプリケーションを、編成およびパーソナライズされた方法で表示します。これには、WebDB 2.2 のサイト構築機能およびセルフサービス Web パブリッシング機能が含まれ、シングル・サインオン、パーソナライゼーションおよびコンテンツ分類などのエンタープライズ・ポータル機能が追加されています。Oracle Portal は Oracle8i を使用し、Oracle iAS に配置されます。Oracle Portal は、iAS にパッケージ化されています。

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

Oracle Exchange

Oracle Exchange プラットフォームは、Oracle8i に基づいています。業界全体または企業のサプライ・チェーンをサポートするために必要な、すべてのビジネス・トランザクションを提供します。Oracle Exchange は、Oracle の E-Business Suite に基づいています。E-Business Suite は、顧客候補からの初めての連絡から、製造計画および製造実施、および販売後の継続サービスおよびサポートまでのサプライ・チェーンをサポートします。

Oracle Exchange は、データ交換フォーマットおよびメッセージ・ペイロードとして XML およびアドバンスト・キューイングを使用します。

その他の取組み

これらのツールの他に、次の取組みを行っています。

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

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

アドバンスド・キューイングの XML サポート：信頼性の高い、非同期メッセージ機能に対するインターネットの使用

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

Oracle *interMedia* Text を使用した XML ドキュメントの索引付けおよび検索

interMedia Text は、CLOB およびその他のドキュメントに格納された XML を検索および取得するための、強力なオプションを提供します。表の列に格納された、最大 4GB の XML ドキュメントおよびドキュメント・セクションを索引付けおよび検索できます。

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

データベース内のテキスト CLOB に保存された XML ドキュメントまたはドキュメント・セクションは、Oracle8i *interMedia* Text のテキスト検索エンジンを使用して索引付けできます。開発者は、特定の XML 階層内のデータを正確に検索し、XML 要素の属性にある名前と値の組の場所を検索できます。

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

参照： 第 5 章「*interMedia* Text を使用した XML ドキュメントのデータ検索および取得」および『Oracle8i *interMedia* Text リファレンス』を参照してください。

メッセージング・ハブおよび中間層コンポーネント

Oracle の XML には、次のコンポーネントも含まれています。

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

Oracle8i JVM (Java 仮想マシン)

Oracle8i JVM は、Oracle マルチスレッド・サーバー (MTS)・アーキテクチャに基づいて構築されています。これは Java 1.2 準拠の仮想マシンであり、データ・サーバーがメモリー・アドレス領域を共有します。Oracle8i JVM には、次の機能があります。

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

Oracle8i JVM は、ネイティブ CORBA および EJB 標準と、Java を SQL および PL/SQL と簡単に統合するための Java スタッド・プロシージャをサポートします。

Oracle Integration Server (OIS)

OIS は、Oracle Advanced Queueing を使用して、XML ペイロード・メッセージを送信および受信します。Oracle Message Broker は、JMS ラッパーを使用して、XML メッセージをパッケージ化して配信します。

Oracle Internet Application Server (iAS)

Oracle iAS 8i は、イントラネットおよびインターネットの Web アプリケーションに対するサービスを提供します。Oracle8i と統合され、データ・キャッシュや Oracle Portal などの高度なサービスを提供します。

バックエンド、データベース、フロントエンドの統合

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

XML を使用すると、異なるベンダーのリレーショナル・データベースとオブジェクト・リレーショナル・データベース間のこのようなデータ交換がより単純になります。XML および AQ を使用したデータ交換の実装の例は、[第 12 章「B2B XML アプリケーション: ステップ・バイ・ステップ」](#)を参照してください。

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

高パフォーマンスへの影響

前述のソリューションを使用すると、次の理由からパフォーマンスが向上します。

- データベース・データおよび XML が同じサーバー上で同時に処理され、データ・アクセスによるネットワークの通信量が削減されます。
- Oracle8i の問合せエンジンが高速化され、Oracle8i JVM、iAS または OIS によってデータ・アクセス速度がさらに高速になります。

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

Oracle XML Parser が提供する API: DOM および SAX

Oracle XML Parser は、Java および PL/SQL で実装されます。Java バージョンは、Oracle8i JVM (Java 仮想マシン) 上で直接実行します。XML1.0 の仕様をサポートし、検証または非検証パーサーとして使用されます。

Oracle XML Parser は、開発者が XML ドキュメントを処理するために必要な、最も一般的な次の 2 つの API を提供します。

- **DOM:** W3C 勧告のドキュメント・オブジェクト・モデル (DOM)・インタフェースです。解析済ドキュメントの要素内容に対して、アクセスおよび編集するための標準的な手段を提供します。
- **SAX:** Simple API for XML インタフェースです。

詳細は、[第 16 章「XML Parser for Java の使用」](#)を参照してください。Oracle XML Parser と Class Generator の比較については、[付録 B「Oracle XML Parser および Class Generator の言語間比較」](#)を参照してください。

カスタムの XML アプリケーションの作成

Oracle8i 環境では、XML ドキュメントを処理するカスタム・アプリケーションを簡単に作成できます。これによって、すべての層に配置可能な言語を使用して、業界標準に準拠した移植可能なアプリケーションおよびコンポーネントを作成できます。

XML Parser は、Oracle8i がインストールされたすべてのオペレーティング・システム上の Oracle8i プラットフォームの一部です。

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

データベースへの XML のロード

次のオプションを使用して、XML データまたは DTD ファイルを Oracle8i にロードできます。

- dbms_lob などの LOB に対する PL/SQL ストアド・プロシージャの使用
- Java カスタム・コードの記述
- SQL*Loader の使用
- Oracle *interMedia* の使用
- XML SQL Utility

Internet File System (iFS) を使用して、XML ドキュメントをデータベースに挿入することもできます。ただし、DTD はサポートされません。DTD に代わる標準の XML Schema がサポートされています。

Oracle8i からの XML データの格納および抽出

XML は、Web 上でのデータ交換の標準として登場しました。Oracle8i は XML 対応であり、現在の市場のニーズに対応できます。

Oracle8i は、次のデータを格納できます。

- オブジェクト・リレーショナル・データとしての構造化 XML データ
- *interMedia Text* データとしての非構造化 XML ドキュメント

Oracle8i は、オブジェクト・リレーショナル・データを XML として自動的に抽出する機能を提供します。標準の SQL を使用した、XML データの効率的な問合せが簡単になります。

また、DOM API を使用して XML ドキュメントへアクセスする機能も提供されます。

Oracle8i: オブジェクト・リレーショナル・インフラストラクチャ

Oracle データベースは、SQL:1999 標準に準拠したオブジェクト・リレーショナル・エンジンへと拡張されています。Oracle のオブジェクト・リレーショナル・エンジンによって、オブジェクト型、コレクション型およびオブジェクト型への参照を定義できます。

このオブジェクト・リレーショナル・インフラストラクチャは、データベースへの構造化オブジェクト・インスタンスの格納をサポートします。本質的に構造化データ形式である XML は、オブジェクト・リレーショナル・インスタンスに簡単にマップできます。XML のマッピングの詳細は、1-31 ページの「[XML Schema およびドキュメントのマッピング](#)」を参照してください。

Oracle8i: 拡張可能なアーキテクチャ

通常、データベースは一連のサービスを提供します。たとえば、基本的な格納サービス、問合せ処理サービス、索引付け、問合せ最適化などのサービスです。

拡張可能なサービス

Oracle8i では、これらのサービスは拡張可能であり、データ・カートリッジが独自の実装を提供できます。データベースが提供するシステム固有のサービスに、特定のドメインに適さない部分がある場合、開発者はドメイン固有の実装を構築できます。

たとえば、地理情報システム用の空間データ・カートリッジを構築する場合、空間索引を作成するための機能が必要です。これを行うには、空間索引の作成、索引へのエントリの挿入、索引の更新、索引からの削除などを実行するルーチンを実装します。サーバーは、空間データに索引付けの機能が必要になるたびに、自動的に実装を起動します。基本的には、サーバーの索引付けサービスを拡張して、空間データを処理します。

拡張性と XML

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

interMedia Text 検索

拡張インフラストラクチャの例は、*interMedia Text* 検索です。LOB に保存されたテキストは、拡張索引付けインタフェースを使用して索引付けされます。*interMedia Text* は、CONTAINS などの演算子を提供します。これらの演算子を使用すると、テキスト内の一致した部分文字列を検索できます。

Oracle8i の拡張フレームワークは、特殊な XML カートリッジを構築するためのインフラストラクチャを提供します。この場合、カートリッジによって XML の索引付けおよび最適化が実現されます。

Oracle8i によるネイティブ Java のサポート

Oracle8i は、パフォーマンスおよびスケーラビリティを向上するためにデータベースと密接に統合されたネイティブ Java VM を提供することによって、DBMS での Java の固有のサポートを提供します。また、データベース・システムは、JDBC、SQLJ、ORB および EJB フレームワークを固有にサポートします。さらに、Oracle8i には HTTP リスナーが搭載されています。これによって、Oracle8i は Web サーバーとしても機能します。

オブジェクト・リレーショナル・フレームワークによって、アプリケーション・レベルの一連の Java クラスとデータ領域レベルのデータ・モデル間で、一貫した構造を保持するためのより自然な方法が提供されます。Oracle8i では、次のとおりオブジェクト・リレーショナル機能が Java 環境と密接に統合されています。

- サーバー・オブジェクト・リレーショナル・スキーマは、Java クラスにマップできます。JPublisher ユーティリティは、このマッピングを自動的に実行できます。
- Java は、オブジェクト型のメソッドおよびデータ・カートリッジを実装するために選択できる言語の 1 つです。
- オブジェクトは、JDBC または SQLJ を使用して操作（格納および取出し）できます。

パーサーなどの多くの XML インフラストラクチャは Java に対応し、サーバー内で簡単に使用できる必要があるため、データベース内での Java のサポートは必須です。また、Java で構築された XML 用のコンポーネントは、サーバー内またはアプリケーション層の外で実行できます。

データベース内の XML: 生成される XML および作成済 XML

Oracle8i は、データベースでの XML の使用に対して、様々な側面をサポートをします。

- **生成される XML:** XML を交換用フォーマットとして使用します。データベース内で XML を使用する最も一般的な方法は、XML を交換用フォーマットとして使用することです。ここで、既存のビジネス・データは XML 構造内にラップされます。この場合、XML 形式は交換処理自体のみに使用され、一時的です。
- **作成済 XML:** XML ドキュメントをデータベース内に格納し、問い合わせます。

生成される XML

XML は、XML SQL Utility (XSU) を使用して、データベース表およびビューに格納されたデータから生成できます。XML SQL Utility は、コマンドラインによるフロントエンド、Java API および PL/SQL API で構成されます。

XML SQL Utility (XSU) による XML への SQL 問合せ結果の変換

このユーティリティは、問合せの別名または列名を要素タグ名にマップし、オブジェクト型のネストを保持することによって、SQL 問合せの結果を XML に変換します。XML SQL Utility は、XML を文字列または DOM ツリー表現で戻すことができます。DOM ツリーは、後の操作に非常に有効な形式です。

XML とデータベース間のマッピング

構造化 XML インスタンスとオブジェクト・リレーショナル型の関係は、非常に明確です。

- 列は、最上位の要素にマップされます。
- スカラー値は、内容がテキストのみの要素にマップされます。
- 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されます。

例 : XML SQL Utility を使用した SQL 問合せ結果の XML 表現の取得

次の例では、XML SQL Utility の Java API が使用されます。例では、データベースに対して適用される SQL 問合せが指定されています。問合せ結果は、XML として戻されます。

```
public void testXML()
{
    DriverManager.registerDriver(
        new oracle.jdbc.driver.OracleDriver());

    //JDBC 接続を初期化します。
    Connection conn =
        DriverManager.getConnection(
            "jdbc:oracle:oci8:scott/tiger@");

    //OracleXMLQuery を初期化します。;
    OracleXMLQuery qry =
        new OracleXMLQuery(conn,
            "select * from purchaseOrderTab");

    //行セットの要素名を設定します。
    qry.setRowsetTag("PurchaseOrderList");
    //行の要素名を設定します。
    qry.setRowTag("PurchaseOrder");
    //XML 結果を取得します。
    String xmlString = qry.getXMLString();
    //結果を出力します。
    System.out.println(" OUPUT IS:\n"+xmlString);
}
```

このコードを実行すると、次の XML ドキュメントが生成されます。

```
<?xml version='1.0'?>
<PurchaseOrderList>
  <PurchaseOrder num="1">
    <purchaseNo>1001</purchaseNo>
    <purchaseDate>10-Jan-1999</purchaseDate>
    <customer>
      <custNo>100</custNo>
      <custName>Hose</custName>
      <custAddr>
        <street>200 Redwood Shrs</street>
        <city>Redwood City</city>
        <state>CA</state>
        <zip>94065</zip>
      </custAddr>
    </customer>
  </PurchaseOrder>
</PurchaseOrderList>
```

```
<lineItemList>
  <lineItem>
    <lineItemNo>901</lineItemNo>
    <lineItemName>Chair</lineItemName>
    <lineItemPrice>234.55</lineItemPrice>
    <lineItemQuan>10</lineItemQuan>
  </lineItem>
  <lineItem>
    <lineItemNo>991</lineItemNo>
    <lineItemName>Desk</lineItemName>
    <lineItemPrice>3456.63</lineItemPrice>
    <lineItemQuan>20</lineItemQuan>
  </lineItem>
</lineItemList>
</PurchaseOrder>
<PurchaseOrder>
  <!-- more purchase orders. -->
</PurchaseOrder>
</PurchaseOrderList>
```

作成される XML ドキュメントは、構造的に、問い合わせられたデータベース・オブジェクトの完全なレプリカです。オブジェクト・リレーショナル・ビューを使用しても、フラットなりレーショナル・スキーマに格納されたデータから、XSLT を使用せずに前述のような構造化 XML を取得できます。

XML 記憶領域オプション

次に、XML データの格納に使用できるオプションを示します。

- **LOB 形式での格納: 作成済 XML ドキュメントの格納**
- **オブジェクト・リレーショナル形式での格納: 生成される XML ドキュメントの格納**

LOB 形式での格納: 作成済 XML ドキュメントの格納

Oracle8i では、キャラクタ LOB (CLOB)、バイナリ LOB (BLOB) またはバイナリ・ファイル (BFILE) としてのラージ・オブジェクト (LOB) の格納がサポートされます。LOB は、作成済 XML またはネイティブ XML ドキュメントの格納に使用されます。

CLOB または BFILE を使用した非構造化データの格納

大きい文字データを格納できる CLOB は、非構造化 XML ドキュメントの格納に有効です。外部ファイル参照である BFILE も格納に使用できますが、BFILE はマルチメディア・データにより適しています。この場合、XML は RDBMS 外に格納され管理されますが、サーバー側の問合せに使用できます。ドキュメントのメタデータは、索引付けおよびアクセスを高速化するために、サーバーのオブジェクト・リレーショナル表に格納できます。

interMedia Text の索引付けによる XML 要素内の内容検索のサポート

Oracle8i では、外部ドキュメントを指す URL に加えて、LOB 列での *interMedia Text* 索引の作成が可能です。このテキスト・カートリッジは、拡張メカニズムを使用し、これらのドキュメントの全文索引を作成します。Oracle8i では、XML データも処理できるように、このメカニズムが拡張されています。

テキスト・カートリッジは XML タグを認識します。XML 要素内容の検索をサポートするために、セクションおよびサブセクション・テキストの検索が拡張されています。そのため、問合せを非構造化データに対して実行し、ドキュメント内の特定のセクションまたは要素に限定することができます。

オブジェクト・リレーショナル形式での格納: 生成される XML ドキュメントの格納

XML は、通常オブジェクト・リレーショナル・インスタンスとして格納します。オブジェクト・リレーショナル型システムは、XML のネストおよびリスト・セマンティクスを完全に表現できます。複雑な XML ドキュメントは、オブジェクト・リレーショナル・インスタンスとして格納し、効率的に索引付けできます。拡張インフラストラクチャを使用すると、パス索引などの新しいタイプの索引を作成して、XML ドキュメントの検索を高速化できます。

構造化 XML ドキュメント（生成された XML）の格納

構造が適切に定義され、すべてのインスタンスで同一であれば、データは構造化 XML ドキュメント形式にできます。また、ドキュメントは、リレーショナルまたはオブジェクト・リレーショナル構造に格納できます。また、オブジェクト・リレーショナル型システムでは、XML ドキュメントへの直接マッピングが可能です。

このマッピングは比較的容易です。Oracle XML SQL Utility (XSU) によって、XML ドキュメントを特定の表またはビューに直接マップする挿入メカニズムが提供されます。

XML 構造を保持した XML SQL Utility による XML データの格納

XML SQL Utility は、XML を再使用するために、次のとおり XML 構造を保持して XML データを格納します。

- タグ名は列にマップされます。
- テキストのみのデータ（要素）はスカラー列にマップされます。
- サブ要素を持つ要素はオブジェクト型にマップされます。
- サブ要素のリストを持つ要素はコレクション型にマップされます。

例 : XML SQL Utility の Java API を使用したデータベースへの XML の挿入

次に、XSU の Java API の単純な使用例を示します。ここでは、XML ドキュメントをデータベース表に挿入します。

```
public void testInsertXML()
{
    DriverManager.registerDriver(
        new oracle.jdbc.driver.OracleDriver());

    //JDBC 接続を初期化します。
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    //XML を含むファイルまたは URL に対する URL オブジェクトを作成します。
    URL url = sav.getURL("sample.xml");
    OracleXMLSave sav = new OracleXMLSave(conn, "purchaseOrderTab");
    System.out.println("Rows inserted:  "+sav.insertXML(xmlDoc));
}
```


XML ドキュメントのオブジェクト・リレーショナル形式での格納： メリット

XML ドキュメントをオブジェクト・リレーショナル・インスタンスとして格納する場合のメリットは、ドキュメントの構造がデータベース内でも保持されることです。これによって、ドキュメント上での XPath 検索に類似した方法で、XML ドキュメントを SQL で表示および全検索できます。

たとえば、次のような XPath 検索について考えてみます。

```
//purchase_order[pono=101]/shipaddr/street
```

これは、SQL で属性検索として簡単に表すことができます。

```
SELECT po.shipaddr.street
FROM purchase_order_tab po
WHERE po.pono = 100;
```

オブジェクト・リレーショナルへのマッピングによって、既存のデータベース・アプリケーションで XML データを処理できます。また、Oracle8i が提供するオブジェクト・リレーショナル列での索引付け、パーティション化、パラレル問合せなどの機能も使用できます。

XML ドキュメント・オブジェクト・リレーショナル形式での格納： デメリット

前述のようなマッピングを使用すると、元のドキュメントを完全に再生できなくなります。たとえば、コメントが失われます。ただし、これは次の項で説明するとおり、元のドキュメントのコピーを CLOB に格納し、問合せの効率を改善するためにマップされたオブジェクト・リレーショナル・データを使用することによって、回避できます。

要素の順序付けによる別の問題が発生する可能性があります。要素の順序を保持するには、基礎となる表に特別な列を作成し、その列を使用して結果を順序付けする必要があります。

非構造化 XML ドキュメント（作成済 XML）の格納

受け取った XML ドキュメントが特定の構造に準拠しない場合、ドキュメントを CLOB に格納する方が有効である場合があります。たとえば、XML メッセージ機能環境では、問合せでの各 XML メッセージの構造が異なる場合があります。

Oracle8i では、CLOB 列を索引付けするための *interMedia Text* が提供されています。これは、CONTAINS などの識別子を実装する拡張メカニズムを使用して、テキスト・データを検索します。*interMedia Text* は、セクションおよびサブセクションの検索を含む、XML ドキュメントの検索をサポートするように拡張されています。

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

この *interMedia Text* の例では、適切な索引が作成されていると想定します。

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

参照： *interMedia Text* の詳細は、第 5 章「[interMedia Text を使用した XML ドキュメントのデータ検索および取得](#)」を参照してください。

非構造化 XML ドキュメント（作成済 XML）の格納：メリット

XML ドキュメントの構造が不明、任意または動的である場合は、CLOB 格納が最適です。

非構造化 XML ドキュメント（作成済 XML）の格納：デメリット

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

マッピングの細分化を改善するためのハイブリッドな格納方法の使用

前述の項は、次のことを説明しました。

- 構造化 XML ドキュメントをオブジェクト・リレーショナル・インスタンスにマップする方法
- 非構造化 XML ドキュメントを LOB に格納する方法

多くの場合、マッピングの細分化をより適切に制御する必要があります。

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

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

ハイブリッドな格納によるユーザー定義の細分化での格納

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

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

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

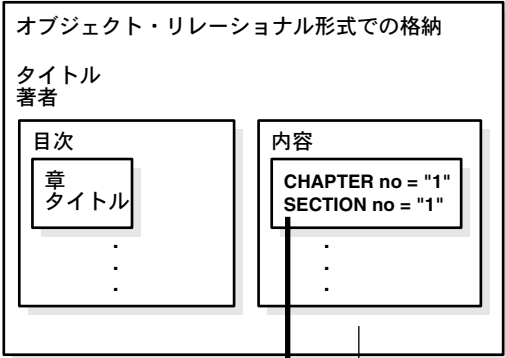
図 1-3 に、ハイブリッドな格納方法を示します。

図 1-3 ハイブリッドな方法：内容が CLOB に格納されている場合に、表の最上位要素を問い合わせる例

XMLドキュメント

```
<?xml version = '1.0'?>
<BOOK>
  <TITLE>Oracle PL/SQL</TITLE>
  <AUTHOR>Steve Feuerstein</AUTHOR>
  <TABLE_OF_CONTENTS>
    <CHAPTER>
      <CHAPTER_NUM>1</CHAPTER_NUM>
      <TITLE>概要</TITLE>
      <SECTIONS>
        ...
      </SECTIONS>
    </CHAPTER>
    ...
  </TABLE_OF_CONTENTS>
  <DETAILS>
    <CHAPTER no="1">
      <SECTION no="1" name="PL/SQLの概要">
        PL/SQLは、Oracleがサポートするプロ
        グラミング言語です。
      </SECTION>
      ...
    </CHAPTER>
  </DETAILS>
</BOOK>
```

列にマップ
された
最上位の要素



表またはビュー
である可能性もある

LOB形式での格納
PL/SQLは、Oracleが
サポートするプログ
ラミング言語です。

ハイブリッドな格納のメリット

- 次に、XML ドキュメントをハイブリッドな方法で格納するメリットを示します。
- 問合せ可能で有効な情報をオブジェクト・リレーショナル形式に格納でき、ドキュメント全体が分解されません。
 - ドキュメント全体が分解されないため、ドキュメントの再作成の時間が短縮されます。
 - LOB に格納された部分のドキュメントがテキスト検索できます。

生成される XML: XML の変換

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

XSLT を使用した問合せ結果の変換

元のドキュメントに問い合せて、結果をエンド・ユーザーまたはアプリケーションが要求する形式に変換します。たとえば、WML を使用してアプリケーションで携帯電話と通信する場合、生成される XML を、WML または携帯電話との通信に適した同様の標準形式に変換する必要があります。

これを行うには、結果の XML ドキュメントに対して XSLT による変換を適用します。XSLT による変換は、最適化されるように生成フェーズで行うことができます。データベース・サーバー内にあるスケーラブルで高パフォーマンスな XSLT 変換エンジンは、大量のデータを処理できます。

XML Schema およびドキュメントのマッピング

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

- XML Schema1: ドキュメント構造 (要素、属性、名前空間) の制約
- XML Schema2: 内容 (データ型、エンティティ、表記法) の制約

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

注意： XML Schema API は、今回のリリースのソフトウェアでは提供されていません。

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

たとえば、「[XML Schema の例 1: 単純なデータ型の定義](#)」に示すように、単純なデータ型を定義できます。

XML Schema の例 1: 単純なデータ型の定義

次に、XML Schema で単純なデータ型を定義する例を示します。

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

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

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

XML Schema の例 2: XML Schema を使用した基礎となるスキーマへの生成された XML ドキュメントのマッピング

XML Schema で表された単純な発注書の型について考えてみます。

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

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

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

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

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

これらの XML Schema は、1-23 ページで説明されているオブジェクト・リレーショナルでの発注書の例と一致するように作成されています。

XML Schema で提案された構造体と SQL:1999 ベースの型システムの類似性が重要です。そのような類似性によって、XML Schema をデータベースのオブジェクト・リレーショナル・スキーマにマップし、前述のスキーマに従って、スキーマに有効なドキュメントをデータベース・スキーマ内の行オブジェクトに簡単にマップできます。DTD より優れた XML Schema の表現力によって、マッピングが大幅に簡単になっています。

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

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

作成済 XML: ドキュメントとしての XML ドキュメントの格納

XML ドキュメントの内容がドキュメント全体の置換によってのみ更新されるような静的な内容である場合、元の XML ドキュメントを CLOB または BLOB に格納することをお薦めします。

作成済 XML の例

例には、雑誌記事、広告、本、契約書などのテキストがあります。

この種類のドキュメントは文書中心であり、データベースから全体が取り出されます。この種類のドキュメントを Oracle8i に格納すると、ファイル・システムでの格納に対して、データベースのメリットおよびその信頼性が得られます。

データベース外への格納

XML ドキュメントをデータベース外に格納する場合でも、Oracle8i の機能を使用して、索引付けおよび問合せを行い、また BFILE、URL およびテキストベースの索引付けを使用して、ドキュメントを効率的に取り出すことができます。

XML ドキュメントの生成および格納

XML ドキュメントの構造が適切に定義されており、その内容が更新可能または他の方法で使用される場合、そのドキュメントはデータ中心であるといえます。通常、XML ドキュメントには、複合構造の要素または属性が含まれます。

生成された XML の例

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

オブジェクト・リレーショナル拡張された Oracle8i には、オブジェクト型、オブジェクト参照およびコレクション型を使用して、データベース内のデータ構造を利用する機能があります。次に示すとおり、XML データの構造をオブジェクト・リレーショナル形式で格納および保存するための 2 つのオプションがあります。

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

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

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

XML ドキュメント構造に変換が必要な場合

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

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

ビューを使用した XML ドキュメントとデータの組合せ

構造化 XML データと非構造化データが組み合わされたときに、そのデータ全体を表示および操作する必要がある場合、Oracle8i のビューを使用します。

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

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

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

Web フォームのインフラストラクチャの生成方法

Web フォームのインフラストラクチャを生成するには、次の操作を行います。

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

XML SQL Utility による格納

XML SQL Utility を使用して XML データを格納する場合、XML を表またはビューにマップできます。XML SQL Utility は、列へのマップおよび列からのマップを正規に行います。

- メリット
 - オブジェクト、LOB およびすべての Oracle の型を処理できます。
 - オブジェクト・ビューは、構造化ドキュメントを複数の表にマップできます。
- デメリット
 - 正規の再マップしか行うことができません。ただし、XSLT の使用が可能のため、XML ドキュメントを生成する際に、XSLT による XML ドキュメントの変換を行うことができます。

データ交換アプリケーションにおける一般的な XML 設計の問題

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

- マッピングの細分化を改善するためのハイブリッドな格納方法の使用
- Web フォームからデータベースへの XML データの送信
- アプリケーション間での XML ドキュメントの通信

マッピングの細分化を改善するためのハイブリッドな格納方法の使用

詳細は、1-29 ページの「マッピングの細分化を改善するためのハイブリッドな格納方法の使用」を参照してください。

ハイブリッドな格納方法については、[図 1-3](#) を参照してください。

Web フォームからデータベースへの XML データの送信

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

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

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

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

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

アプリケーション間での XML ドキュメントの通信

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

内容は次のとおりです。

- 送信アプリケーションが XML ドキュメントを送信します。
- 受信アプリケーションが XML ドキュメントを受信します。
- **ファイル転送:** 受信アプリケーションが、FTP、NFS、SMB または他のファイル転送プロトコルを介して、送信アプリケーションから XML ドキュメントを要求します。ドキュメントが、受信アプリケーションのファイル・システムにコピーされます。アプリケーションが、ファイルを読み込み、処理します。
- **HTTP:** 受信アプリケーションが、サーブレットに HTTP 要求を行います。サーブレットが、XML ドキュメントを受信アプリケーションに戻し、受信アプリケーションが読みおよび処理を行います。
- **Web フォーム:** 送信アプリケーションが、Web フォームをレンダリングします。ユーザーは、ブラウザ内で実行する Java アプレットまたは Java スクリプトを介してフォームに記入し、情報を送信します。アプレットまたは Java スクリプトが、ユーザーのフォームを XML 形式で受信アプリケーションに転送し、受信アプリケーションが読みおよび処理を行います。受信アプリケーションが最終的にデータをデータベースに書き込む場合、送信アプリケーションはデータベースと互換性のある形式の XML を作成する必要があります。Oracle の XML 製品を使用してこれを行う方法については、1-37 ページの「[Web フォームからデータベースへの XML データの送信](#)」を参照してください。
- **アドバンスド・キューイング:** Oracle8i データベースが、Net8 および JDBC を介して、XML ドキュメントを 1 つ以上の受信アプリケーションに送信します。XML ドキュメントが、Oracle Advanced Queuing (AQ) を介してメッセージとして送信されます。受信アプリケーションが、XML メッセージをデキューし、処理します。

参照: 次の章およびマニュアルを参照してください。

- [第 9 章「Oracle Advanced Queuing を使用した XML データの交換」](#)
- [第 12 章「B2B XML アプリケーション: ステップ・バイ・ステップ」](#)
- 『Oracle8i Integration Server 概要』

Oracle の XML のサンプルおよびデモ

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

Oracle の XML コンポーネントの実行要件

Oracle8i には、Java や XML などインターネット標準の固有のサポートが含まれます。これらを使用して構築された Oracle の XML コンポーネントおよびアプリケーションは、Oracle8i に組み込まれた Java 仮想マシンである Oracle8i JVM を使用して、データベース内で実行できます。

Oracle8i Lite は、より小さいデータベースのフットプリントしか要求しないデバイスおよびアプリケーションに対して、XML データを格納および取り出すために使用します。

要件

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

- XDK for Java には、JDK/JRE 1.1 以上の Java VM が必要です。
- XDK for PL/SQL には、Oracle8x または PL/SQL カートリッジが必要です。

要件については、XDK の章および付録も参照してください。

Oracle の XML を使用した ビジネス・ソリューション

この章の内容は次のとおりです。

- Oracle の XML を使用するアプリケーション
- コンテンツおよびドキュメントの管理
 - 使用例 1 - コンテンツおよびドキュメントの管理: Oracle の XML を使用した複合ドキュメントの作成および公開
 - 使用例 2 - コンテンツおよびドキュメントの管理: Oracle の XML を使用した個人情報配信
 - 使用例 3 - コンテンツ管理: XML を使用したデータ駆動のアプリケーションのカスタマイズ
- B2B/B2C メッセージ機能
 - 使用例 4 - B2B メッセージ機能: XML を使用した複数ベンダー用のオンライン・ショッピング・カート設計
 - 使用例 5 - B2B メッセージ機能: インターネット・アプリケーションのための XML および Oracle AQ の使用
 - 使用例 6 - B2B メッセージ機能: XML および Oracle AQ メッセージ機能を使用した複数アプリケーションの統合

Oracle の XML を使用するアプリケーション

インターネット・アプリケーションでは、XML を様々な方法で使用できます。

次に、Oracle の XML コンポーネントの使用に適した 2 つのデータベースを使用したアプリケーションの分野を示します。

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

また、Oracle の XML コンポーネントは、これらを組み合わせて使用する場合にも適しています。このマニュアルでは、これらの 2 つのアプリケーション分野を中心に説明します。それぞれの分野を、いくつかの使用例を基に説明します。

コンテンツおよびドキュメントの管理

データ表示のカスタマイズ

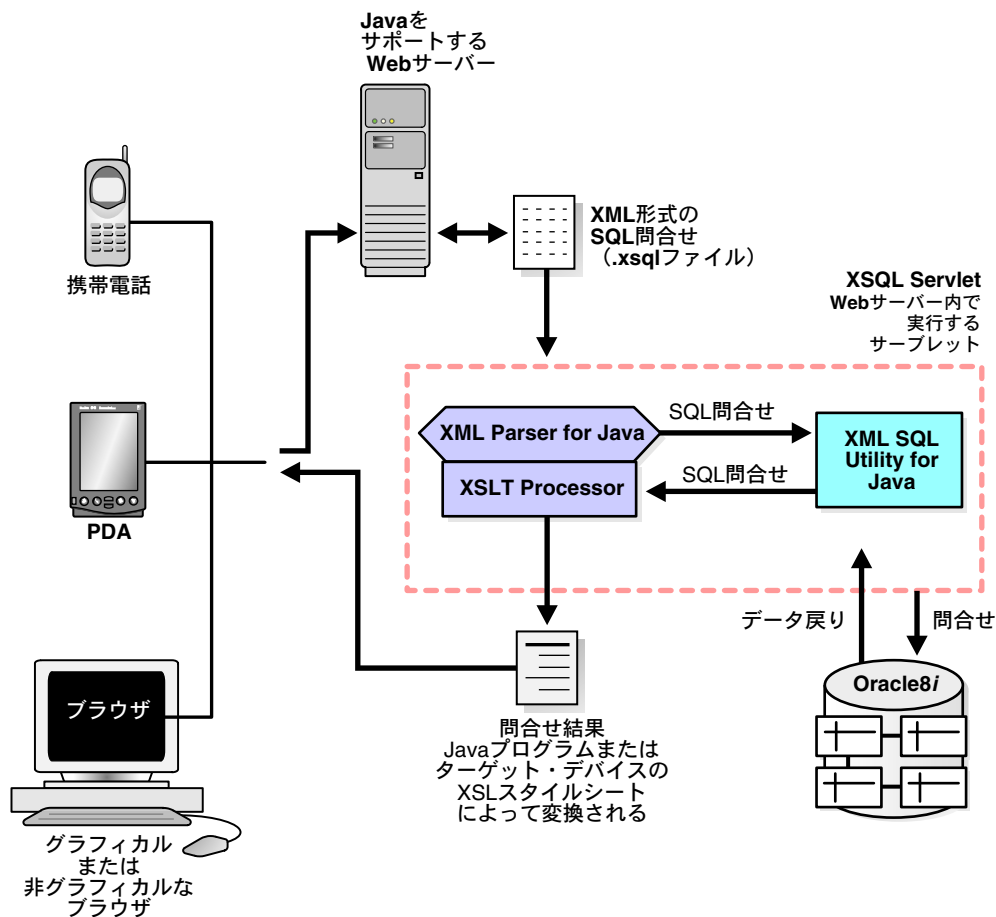
様々なブラウザ、デバイスおよびユーザーに合わせてデータ表示をカスタマイズするための XML の使用が増大しています。XML ドキュメントを XSL スタイルシートとともにクライアント、中間層またはサーバーで使用すると、個々のユーザー用にカスタマイズされた XML データを、次のような様々なクライアント・デバイスに合わせて変換、編成および表示できます。

- グラフィカルおよび非グラフィカルな Web ブラウザ
- Palm Pilot などのパーソナル・デジタル・アシスタント (PDA)
- デジタル式の携帯電話およびポケットベル
- TBD など

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

XML および XSL を使用すると、より簡単に動的 Web サイトを作成および管理できます。XSL スタイルシートを変更すると、基礎となるビジネス・ロジックまたはデータベース・コードを変更せずに、ルックアンドフィールを変更できます。新しいユーザーおよびデバイスをターゲットにする場合、必要に応じて、新しい XSL スタイルシートを設計するのみです。[図 2-1](#) を参照してください。

図 2-1 コンテンツ管理 : 表示のカスタマイズ



参照： 次の章を参照してください。

- 第 7 章「XML によるデータ表示のパーソナライズ : Portal-to-Go」
- 第 16 章「XML Parser for Java の使用」

次のコンテンツ管理の使用例について考えてみます。

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

これらの使用例では、Oracle の XML コンポーネントを使用します。また、それぞれのビジネス上の問題、ソリューション、主な実行タスクおよび使用する Oracle の XML コンポーネントについて説明します。

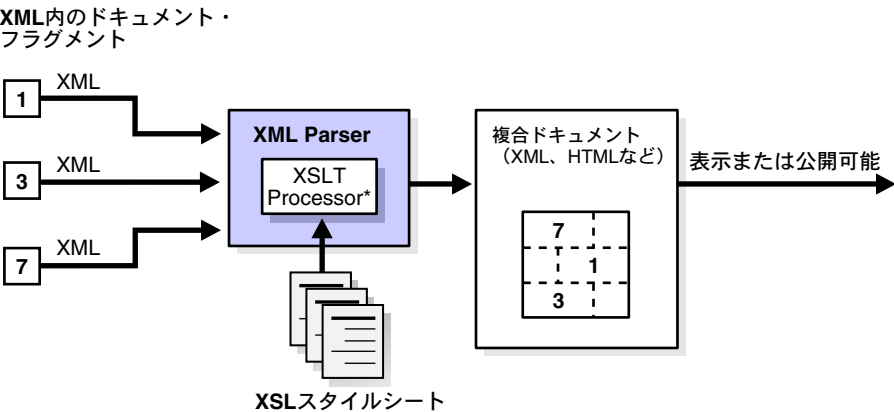
これらの使用例の詳細は、第 III 部の例を参照してください。

使用例 1 - コンテンツおよびドキュメントの管理：
Oracle の XML を使用した複合ドキュメントの作成および公開

問題: X 社には、断片的な SGML および XML マークアップ・テキストのドキュメント・リポジトリが多数あります。複合ドキュメントは、動的に公開される必要があります。

- **ソリューション:** X 社は、XSL スタイルシートを使用して、ドキュメントのセクションまたはフラグメントを組み合わせ、その複合ドキュメントをユーザーに電子送信できます。図 2-2 を参照してください。
- **主な実行タスク:** 使用例 1 で実行される主なタスクは次のとおりです。
 1. ドキュメントを、使用可能な小さいセクションまたはフラグメントに分割します。
 2. これらのセクションまたはフラグメントを、データベースの CLOB および BLOB に格納します。
 3. XSL スタイルシートを作成して、セクションまたはフラグメントを完全なドキュメントにレンダリングします。
- **使用する Oracle の XML:**
 - XML Parser および XSLT
 - XSU (セクションまたはフラグメントをデータベース内外に移動させます。)

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



*XSLT Processorは、複合ドキュメントをドキュメント・フラグメントに分割するためにも使用できます。

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

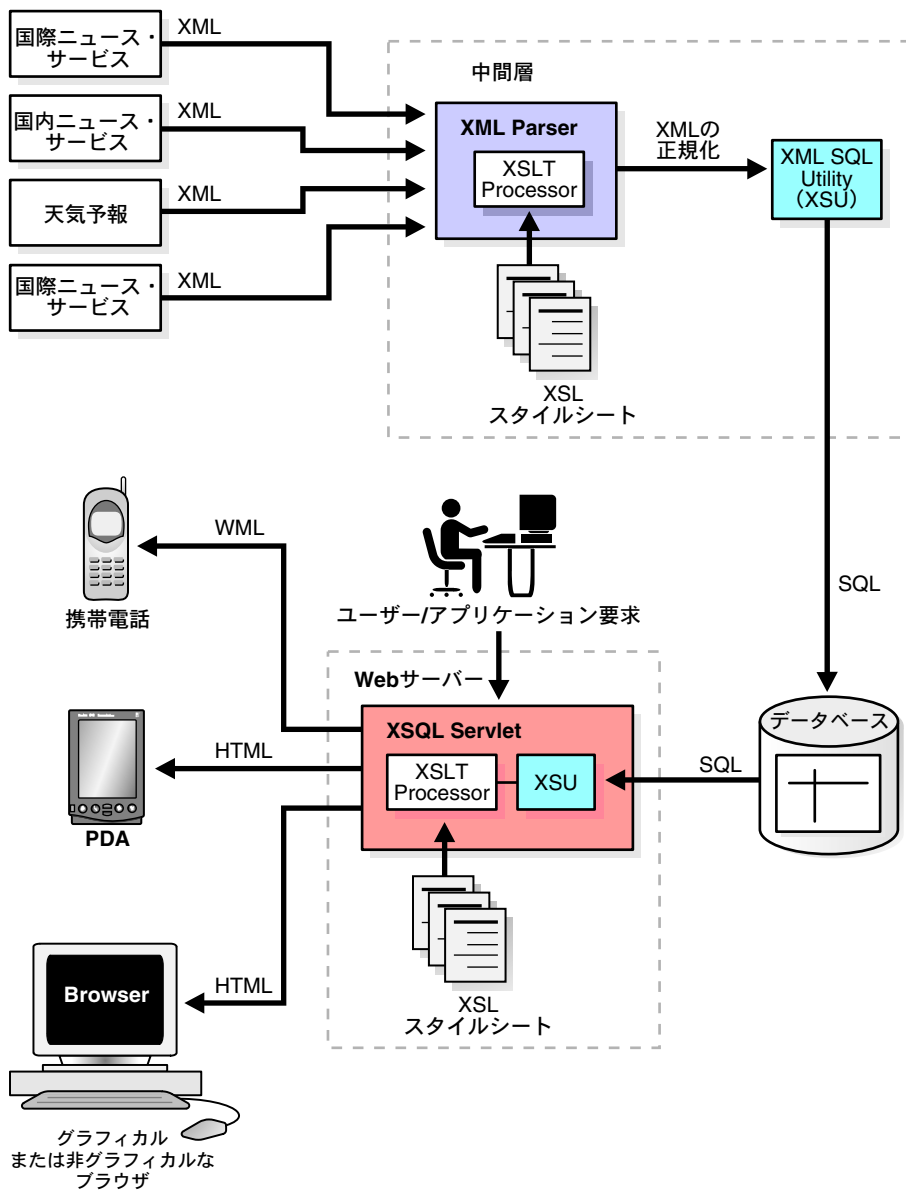
問題 : 大規模なニュース配信元は、様々なニュース・ソースからデータを受信します。このデータは、データベースに格納され、ニュース配信元との契約に従って、配信者およびユーザーがいつでも特定のニュースおよびカスタマイズ済ニュースを参照できるように、必要に応じてすべての配信者およびユーザーに送信する必要があります。配信元は、XSL を使用してデータを正規化し、データベースに格納します。格納されたデータは、いくつかの Web サイトおよびポータルで使用されます。これらの Web サイトおよびポータルは、様々な有線および無線クライアントからの HTTP 要求を受信します。

- **ソリューション :** XSL スタイルシートおよび XSQL Servlet を使用して、要求サービスに対して、動的に適切なレンダリングを行います。図 2-3 を参照してください。
- **主な実行タスク :** 使用例 2 で実行される主なタスクは次のとおりです。
 1. 出力が最適化されるように、データベース・スキーマのデータ・モデルを設計します。
 2. 各情報ソース用の XSL スタイルシートを作成して、正規化されたフォーマットに変換します。変換後、それをデータベースに格納します。
 3. XSL スタイルシートを XSQL ページとともに作成して、データを Web サイト上で公開します。

第 6 章「XML によるコンテンツのカスタマイズ : Dynamic News アプリケーション」も参照してください。

- **使用する Oracle の XML :**
 - XML Parser for Java V2
 - XML SQL Utility (XSU)
 - XSQL Servlet

図 2-3 使用例 2 - XML を使用したカスタマイズ済ニュース情報の配信



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

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

- **ソリューション:** データは、データベースから問い合わせられ、1 つ以上の XSL スタイルシートを介して動的にレンダリングされて、クライアント・アプリケーションに送信されます。データは、リレーショナル・データベースに XML 形式で LOB として格納されます。
- **使用する Oracle の XML:**
 - XML Parser for Java

B2B/B2C メッセージ機能

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

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

XML を使用した B2B/B2C メッセージ機能 : 使用例

次の B2B/B2C メッセージ機能の使用例について考えてみます。

- [使用例 4 - B2B メッセージ機能:XML を使用した複数ベンダー用のオンライン・ショッピング・カート設計](#)
- [使用例 5 - B2B メッセージ機能: インターネット・アプリケーションのための XML および Oracle AQ の使用](#)
- [使用例 6 - B2B メッセージ機能: XML および Oracle AQ メッセージ機能を使用した複数アプリケーションの統合](#)

これらの使用例では、Oracle の XML プラットフォーム・コンポーネントを使用します。各使用例について、問題、ソリューション、問題を解決するために実行する主なタスクなどを、簡単に説明します。

これらの使用例の詳細は、第 VI 部および第 V 部の例を参照してください。

使用例 4 - B2B メッセージ機能: XML を使用した複数ベンダー用のオンライン・ショッピング・カート設計

問題: X 社は、様々なベンダーから製品を購入するためのオンライン・ショッピング・カートを構築する必要があります。X 社は、注文書をオンラインで受信し、注文された製品に応じて、適切なベンダーに注文書を送信する必要があります。

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

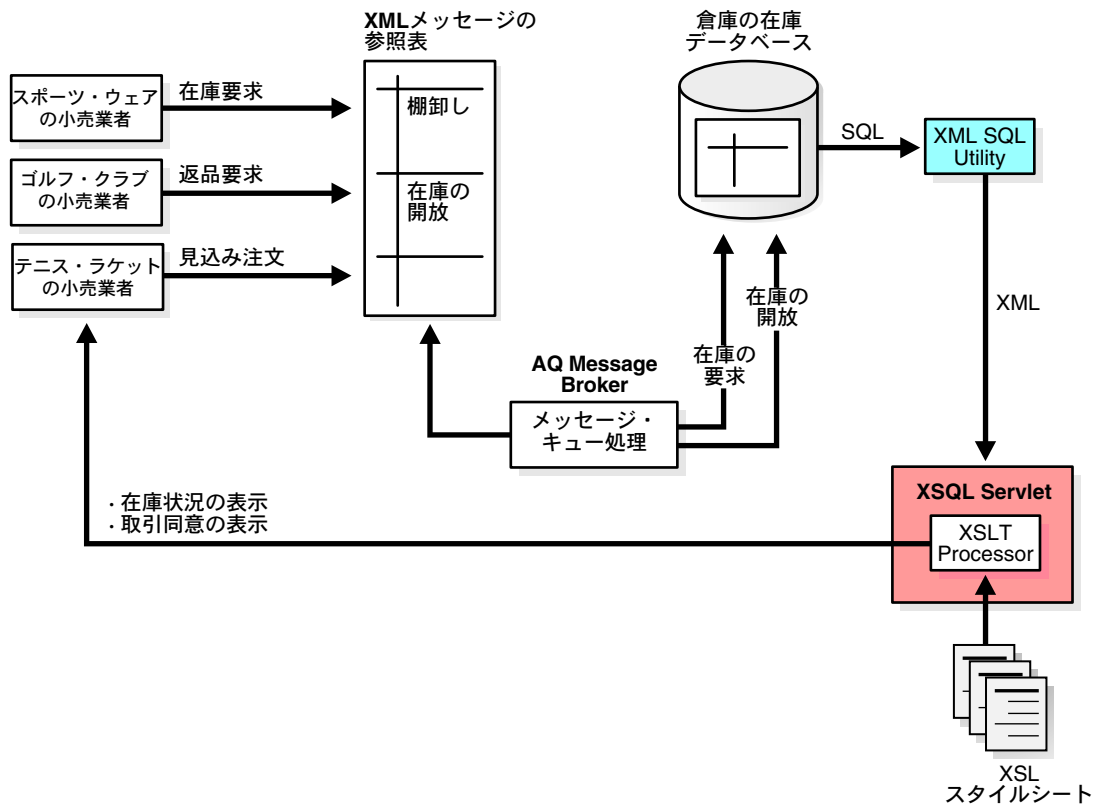
ユーザーは、ベンダーのサイトで商品をショッピング・バスケットに追加した後、ボタンをクリックしてショッピングを完了します。ベンダーは、ショッピング・バスケットの内容を、ユーザーが選択した部品番号、数量および価格情報を含む XML ファイルとして X 社のアプリケーションに返送します。

ショッピング・バスケット内の商品は、新規注文書に明細品目として自動的に追加されます。

顧客の注文書は、XML に変換され、適切なベンダーのデータベースに送信されて、処理されます。適切な転送を行うために、XSL を使用して、カートを変換および分割します。データは、リレーショナル・データベースに XML 形式で格納されます。[図 2-4](#) を参照してください。

- **主な実行タスク:** 同様の実装例については、次の章を参照してください。
 - [第 10 章「B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供」](#)
 - [第 12 章「B2B XML アプリケーション: ステップ・バイ・ステップ」](#)
- **使用する Oracle の XML:**
 - Oracle XML Parser
 - XML SQL Utility
 - XSQL Servlet
 - Java コード（ベンダーの Web サイトから受信するショッピング・カートを認証します。）

図 2-4 使用例 4 - XML を使用した複数ベンダー用のオンライン・ショッピング・カート設計

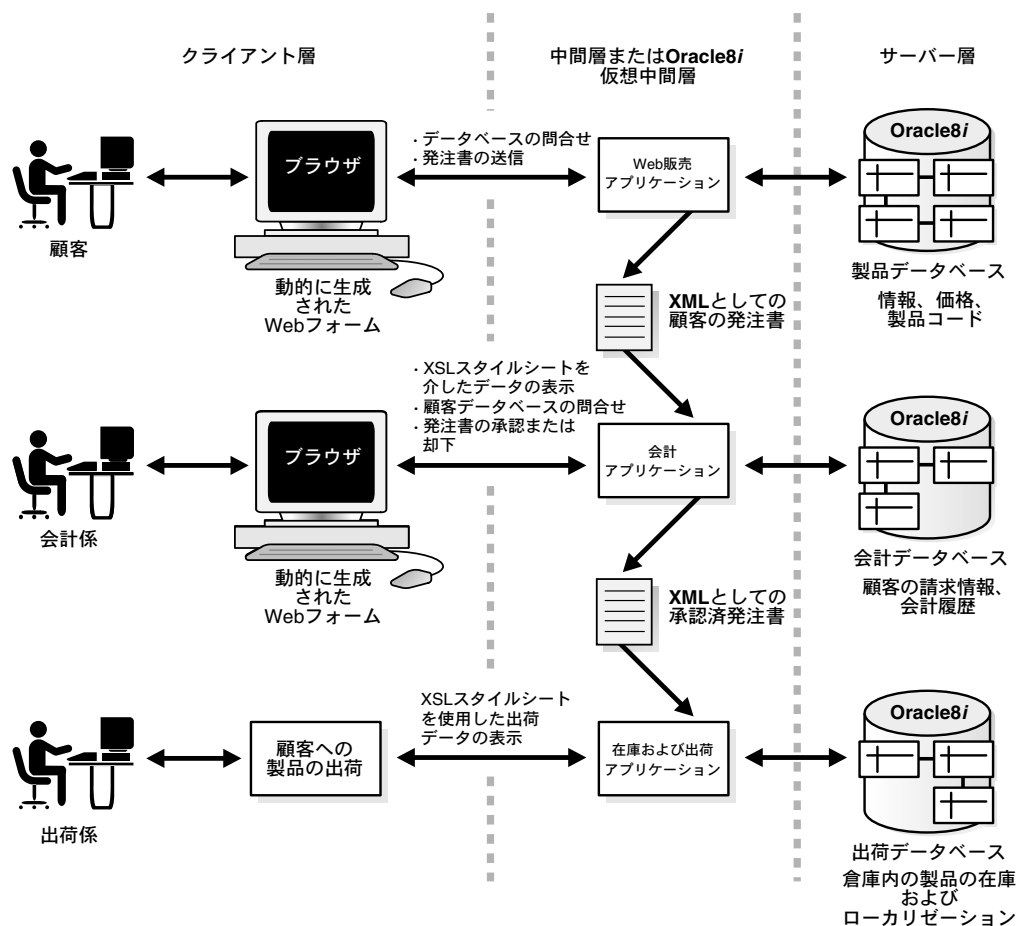


使用例 5 - B2B メッセージ機能： インターネット・アプリケーションのための XML および Oracle AQ の使用

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

- **ソリューション:** リソースがアクセスまたは解放されると、使用可能であることを示す XML メッセージが生成されます。次に、XSL を使用して、必要に応じてリソースを複数のクライアント形式に変換します。逆に、1 つのクライアントがリソースを取得すると、XML メッセージが他のクライアントに送信され、取消しが通知されます。メッセージは LOB に格納されます。データは、リレーショナル・データベースに格納され、XML で具体化されます。図 2-5 を参照してください。
- **主な実行タスク:** 第 11 章「XML メッセージ機能を使用した Phone Number Portability」も参照してください。
- **使用する Oracle の XML:**
 - XML Parser
 - XSLT Processor

図 2-5 使用例 5 - B2B インターネット・アプリケーションのための XML および Oracle AQ メッセージ機能の使用

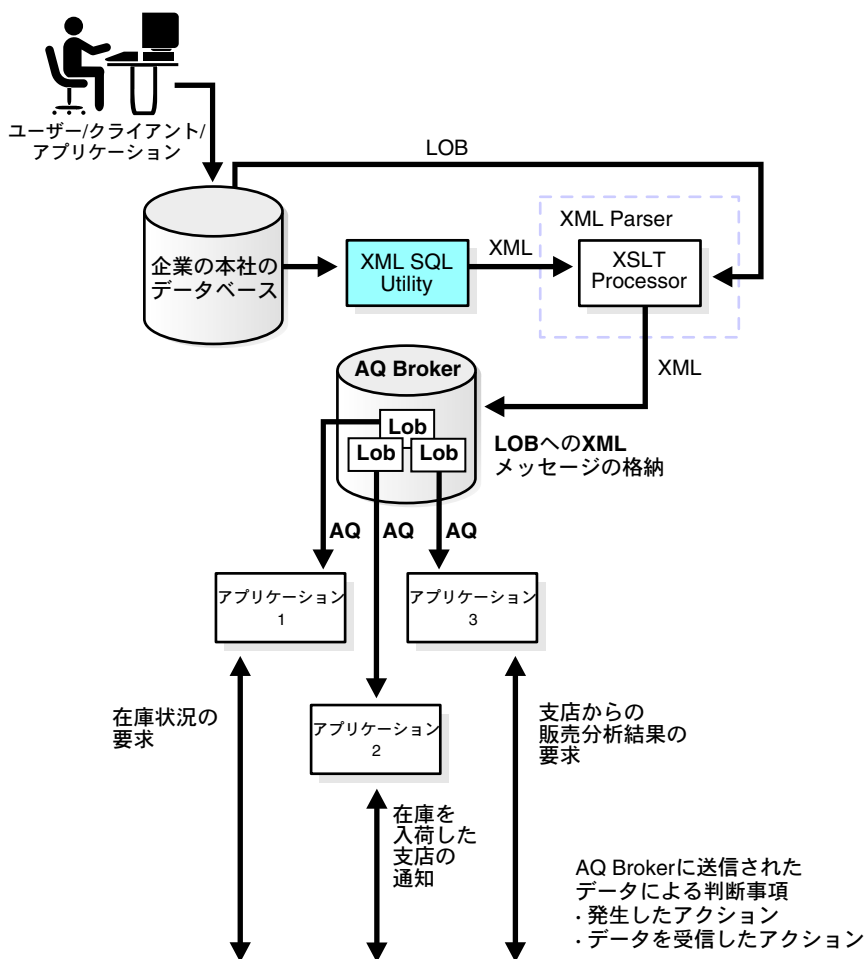


使用例 6 - B2B メッセージ機能： XML および Oracle AQ メッセージ機能を使用した複数アプリケーションの 統合

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

- **ソリューション:** XML をメッセージ・ペイロードとして使用します。XML は XSLT Processor を介して変換され、メッセージに含まれて転送されます。XML メッセージは、AQ Broker データベースに LOB として格納されます。このソリューションでは、コンテンツ管理 (XSL スタイルシートを使用した表示のカスタマイズ) も使用します。[図 2-6](#) を参照してください。
- **主な実行タスク:**
 1. ユーザーまたはアプリケーションが、要求を発信します。結果データが、XSU を使用して企業のデータベースから取り出されます。
 2. データは、XSLT Processor を介して変換され、AQ Broker に送信されます。
 3. AQ Broker は、このメッセージを読み込み、必要なアクションを決定します。AQ Broker は、アプリケーション 1、2 および 3 に適切に応答しその後の処理を行います。
- **使用する Oracle の XML:**
 - XML Parser
 - XSLT Processor
 - XML SQL Utility (XSU)

図 2-6 使用例 6 - XML および Oracle AQ メッセージ機能を使用した複数アプリケーションの統合



Oracle の XML コンポーネントおよび一般的な FAQ

この章の内容は次のとおりです。

- [Oracle の XML コンポーネント : 概要](#)
- [開発ツールおよび XML 対応のその他の Oracle8i 機能](#)
- [XML Parser](#)
- [XSL Transformation \(XSLT\) Processor](#)
- [XML Class Generator](#)
- [XML Transviewer Beans](#)
- [Oracle XSQL Servlet](#)
- [Oracle XML SQL Utility \(XSU\)](#)
- [Oracle interMedia Text](#)
- [Oracle の XML アプリケーションを構築するためのツール](#)
- [Oracle の XML コンポーネント : XML ドキュメントの生成](#)
- [Oracle の XML コンポーネントを使用した XML ドキュメントの生成 : Java](#)
- [Oracle の XML コンポーネントを使用した XML ドキュメントの生成 : PL/SQL](#)
- [Oracle の XML コンポーネントを使用した XML ドキュメントの生成 : PL/SQL](#)
- [FAQ - XML 全般](#)

Oracle の XML コンポーネント : 概要

Oracle8i は、XML テクノロジーを使用して 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。

Oracle8i では、次の XML コンポーネントが提供されます。

- **XML Developer’s Kit (XDK)¹**
Oracle XDK には、Java および PL/SQL 用があります。これらの開発キットには、XML ドキュメントの読み込み、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。
- **XML SQL Utility (XSU)**
XML SQL Utility for Java および XML SQL Utility for PL/SQL 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（取得）し、またデータベースに格納（挿入）もします。このユーティリティは、SQL 問合せの結果を XML に（またはその反対に）正規にマッピングすることによって、データ変換を行います。

表 3-1 に、Oracle の XML コンポーネントおよびそれに対応付けられた言語を示します。

表 3-1 XDK および XSU で使用可能な言語

言語	Java	PL/SQL
Parser	使用可能	使用可能
XSLT Processor	使用可能	使用可能
Class Generator	使用可能	---
XSQL	使用可能	使用不可
Transviewer Beans	使用可能	使用不可
XML SQL Utility	使用可能	使用可能

この章では、XML コンポーネントの概要について説明します。

¹ 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。

開発ツールおよび XML 対応のその他の Oracle8i 機能

- **Oracle8i interMedia Text:** データをドキュメントとして表示し、ドキュメントをデータとして処理できる Java のアプリケーション・インタフェースです。
- **Oracle JDeveloper:** 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。
- **Oracle8i Internet File System (iFS) :** データをドキュメントとして表示し、ドキュメントをデータとして処理できる Java のアプリケーション・インタフェースです。

XDK for Java

XDK for Java は、次のもので構成されます。

- **XML Parser for Java:** 業界標準である DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。XML を XML またはその他のテキストベース・フォーマット (HTML など) に変換する XSL Transformation (XSLT) Processor を搭載しています。
- **XML Class Generator for Java:** Java クラスを生成します。
- **XML Transviewer Beans:** Java を介して XML ドキュメントおよびデータを表示および変換します。
- **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。

XDK for PL/SQL

XDK for PL/SQL は、次のもので構成されます。

- **XML Parser for PL/SQL:** DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。XML を XML または他のテキストベース・フォーマット (HTML など) に変換する XSL Transformation (XSLT) Processor を搭載しています。

XML Parser

Oracle XML Parser には、Oracle8i が実行するすべてのプラットフォーム用の 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、XML1.0 仕様に準拠している妥当性検証 XML Parser 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 1.0 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。

- XPath¹
- ドキュメント・ノードの段階的な XSL Transformation (XSLT)

XSLT は、W3C 勧告のバージョン 1.0 に準拠しています。これをサポートすると、次の機能が使用可能になります。

- 他の XML 構造への XML ドキュメントの変換
- 他のテキストベース・フォーマットへの XML ドキュメントの変換

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

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

参照： [第 16 章「XML Parser for Java の使用」](#) および [付録 C「XDK for Java: 仕様および早見表」](#) を参照してください。

¹ XPath は、W3C 勧告であり、XSLT、Xlink および XML Query で使用される XML ドキュメント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。

図 3-1 Oracle XML Parser for Java

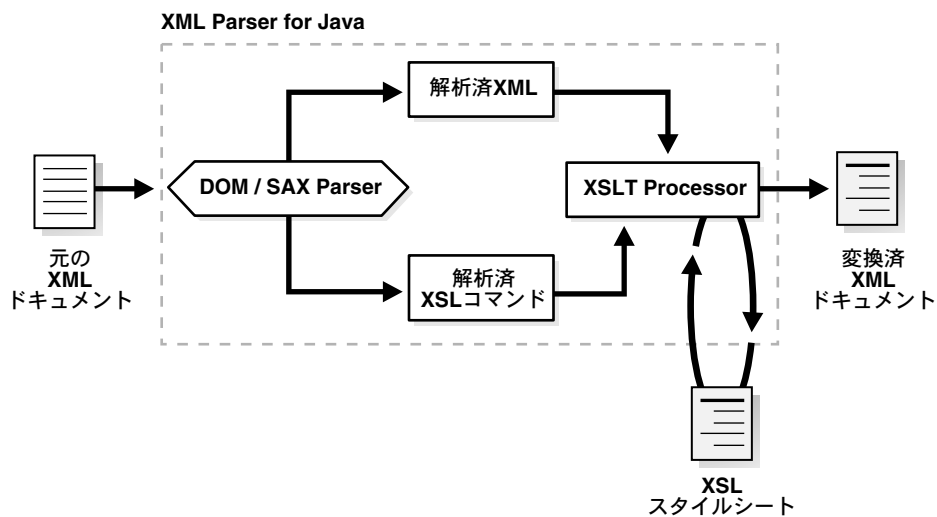
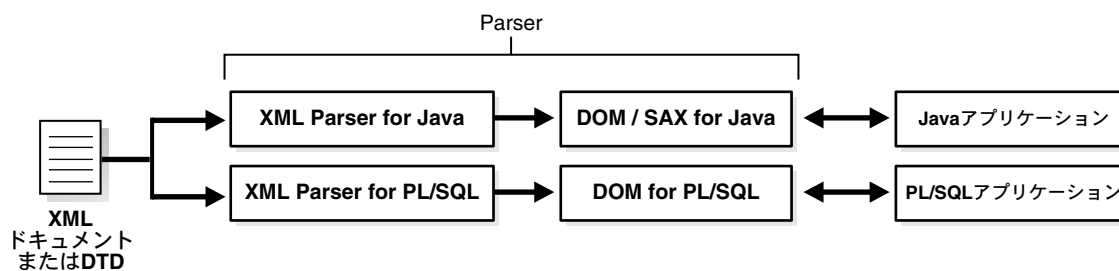


図 3-2 XML Parser: Java および PL/SQL



XSL Transformation (XSLT) Processor

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

- すべてのプラットフォーム上で、データベース内外の XML 情報に対して、標準に基づいた変換を行います。
- Java の拡張性をサポートします。また、パフォーマンス向上のため、システム固有のエンジンとして Oracle8i リリース 8.1.7 のデータベース内にネイティブ・コンパイルされています。

Oracle XML Parser には、XSL スタイルシートを使用して XML データを変換するための統合された XSL Transformation (XSLT) Processor が含まれます。XSLT Processor を使用すると、XML ドキュメントを XML から XML、HTML など、ほぼすべてのテキストベースのフォーマットに変換できます。

XSLT Processor の使用方法については、[第 16 章「XML Parser for Java の使用」](#)を参照してください。

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

XML Class Generator

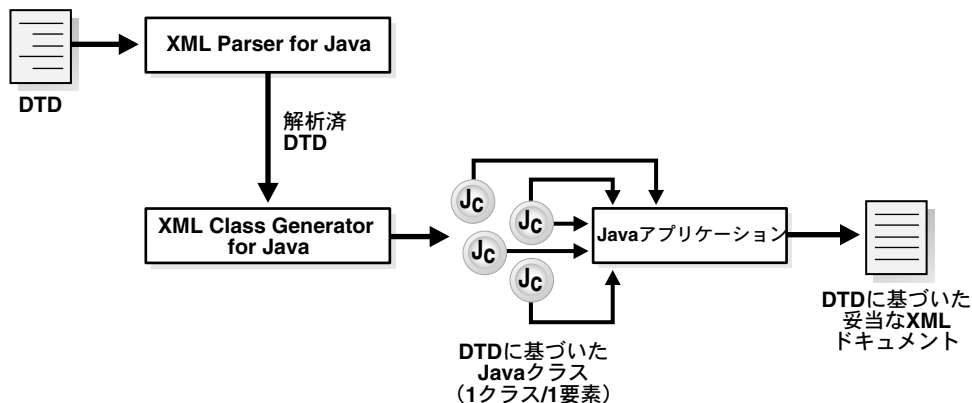
XML Class Generator は、入力 DTD に従った XML ドキュメントを作成するための一連の Java クラスを作成します。

[図 3-3](#) に、Oracle XML Class Generator の機能の概要を示します。

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

- [第 17 章「XML Class Generator for Java の使用」](#)

図 3-3 Oracle XML Class Generator for Java



XML Transviewer Beans

Oracle XML Transviewer Beans は、Java Beans のため 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 ドキュメント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 5 つの Bean が使用可能です。

■ DOM Builder 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。

■ XML Source Viewer Bean

XML ドキュメントを表示可能にする 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 ドキュメントを変更する場合に有効です。この Bean は簡単に DOM Builder Bean と統合でき、指定した DTD に対する事前および事後の解析および妥当性検証を行うことができます。

■ XML Tree Viewer Bean

XML パーサーを拡張および無効にする機能によって、XML ドキュメント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 ドキュメントを 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。

■ XSL Transformer Bean

XSLT Processor を Bean インタフェースでラップし、XSL スタイルシートに基づいて、XML ドキュメントを XSL 変換します。これによって、XSL スタイルシートを適用して、XML ドキュメントを XML、HTML、DDL など、ほぼすべてのテキストベースのフォーマットに変換できます。この Bean を他の Bean と統合した場合、アプリケーションまたはユーザーは、変換の結果をすぐに表示できます。この 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の基礎として使用し、XML ドキュメント（問合せ結果の XML 表示など）をブラウザで表示するために、HTML にレンダリングすることもできます。

■ XML TransPanel 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 ドキュメントおよび XSL スタイルシートをロードするためのファイル・インタフェースが含まれます。これは、次の Bean を使用します。

- XML ドキュメントを表示および編集（オプション）するための Bean
- スタイルシートを XML ドキュメントに適用し、その出力を表示するための変換用の Bean

これらの Bean は、標準の JavaBeans として、Oracle JDeveloper などの、すべてのグラフィカル Java 開発環境で使用できます。

Oracle XSQL Servlet

XSQL Servlet は、SQL 問合せを処理し、その結果セットを 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、入力として埋込み SQL 問合せを含む XML ファイルを取ります。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
- Oracle8i Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1 (JSP Patch 搭載)
- Sun Java Server Web Development Kit (JSWDK) 1.0.1 Web Server

XSQL Servlet をサポートする Java Server Pages (JSP) プラットフォーム

Java Server 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)
- Oracle8i Lite Web-to-Go Server および Oracle JSP 1.0
- 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。

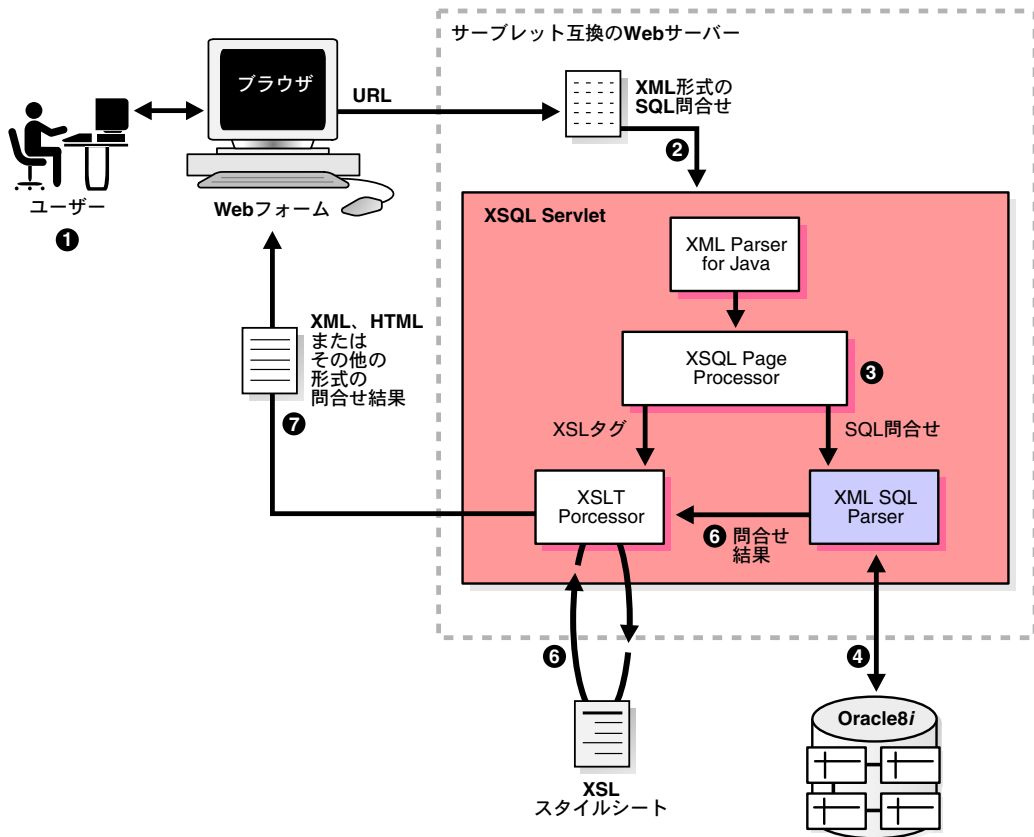
- Servlet 2.1 以上の仕様をサポートするすべてのサーブレット・エンジン
- Oracle JSP 1.0 リファレンス実装またはそれと同等の機能を持つその他のベンダー製品

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 サブレットとして実装され、入力として埋込み SQL 問合せを含む XML ファイルを取ります。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 3-4 に、クライアントからサブレット、およびサブレットからクライアントへのデータ・フローを示します。次に、イベントの順序を示します。

1. ユーザーが、ブラウザを介して URL を入力します。この URL は解析され、Web サーバーを介して XSQL Servlet に渡されます。この URL には、ターゲットの XSQL ファイル (.xsql) の名前が含まれます。また、オプションで、値や XSL スタイルシート名などのパラメータが含まれます。また、ブラウザおよび Web サーバーを介せずに、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 の内容にアクセスするための API を提供します。
3. サブレットのページ・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 Processor に渡します。
4. XML SQL Utility が、Oracle8i 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 Processor 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、指定した XSL スタイルシートを使用して、必要に応じ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 Processor は、最初に URL を要求したクライアントのタイプに基づいて、異なるスタイルシートを選択して適用できます。この HTTP_USER_AGENT の情報は、HTTP 要求を介してクライアントから取得されます。
7. XSLT Processor 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。

図 3-4 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。

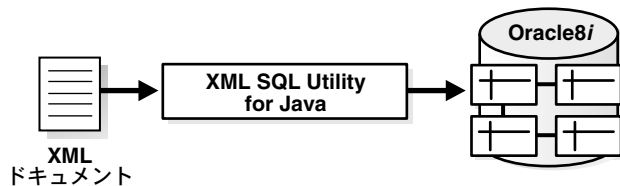
- XML SQL Utility は、任意の SQL 問合せを自動的に正規の XML にレンダリングするためのコア Java クラス・ライブラリで構成されます。このユーティリティには、次の機能が含まれます。
 - 豊富な構造のユーザー定義オブジェクト型およびオブジェクト・ビューでの問合せをサポートします。
 - 正規の構造を持つ XML を既存の表、ビュー、オブジェクト表またはオブジェクト・ビューに自動的に挿入します。XSLT 変換と組み合わせることによって、ほぼすべての XML ドキュメントをデータベースに自動的に挿入できます。

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 ドキュメント、ドキュメント・オブジェクト・モデル (DOM) または文書型定義 (DTD) を生成します。
- XML ドキュメント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 3-5 に、Oracle XML SQL Utility の機能の概要を示します。

図 3-5 Oracle XML SQL Utility の機能図



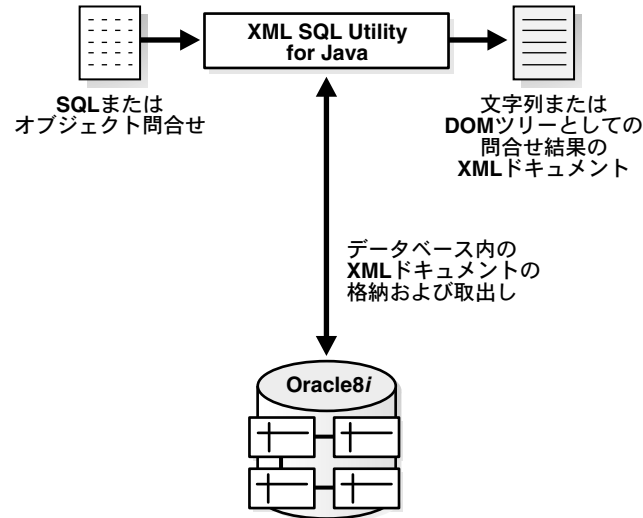
XML SQL Utility for Java は、次のタスクを実行する一連の Java クラスで構成されます。

- 問合せをデータベースに渡し、結果または DTD から XML ドキュメント（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。

問合せ結果からの XML の生成

図 3-6 に、XML SQL Utility が SQL 問合せを処理し、結果を XML ドキュメントとして戻す方法を示します。

図 3-6 XML SQL Utility が SQL 問合せを処理し、結果を XML ドキュメントとして戻す方法



XML ドキュメントの構造：要素への列のマッピング

結果として戻す XML ドキュメントの構造は、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ピングします。
- スカラー値は、内容がテキストのみの要素にマッピングします。
- オブジェクト型は要素にマッピングされ、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ピングします。

XSU による文字列または DOM 要素ツリーとしての XML ドキュメントの生成

XML SQL Utility (XSU) は、次のいずれかを生成します。

- XML ドキュメントの文字列表現

XML ドキュメントをリクエストに戻す場合、この表現を使用してください。

- 要素のメモリー内の 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 Processor 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 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。「[XML Class Generator](#)」も参照してください。

このインフラストラクチャに基づいて、Web フォームではユーザー・データが獲得され、データベース・スキーマと互換性のある XML ドキュメントが作成されます。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 4 章「[XML SQL Utility \(XSU\) の使用](#)」および第 12 章「[B2B XML アプリケーション: ステップ・バイ・ステップ](#)」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 ドキュメントをデータベース表に書き込むには、その XML ドキュメントを変換してから、データベースに書き込んでください。この方法については、第 4 章「[XML SQL Utility \(XSU\) の使用](#)」を参照してください。

Oracle *interMedia* Text

interMedia Text では、Oracle8i 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、Oracle8i を拡張します。

interMedia 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、Oracle8i に格納された XML ドキュメントを検索できます。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： *interMedia* Text および XML の使用方法の詳細は、[第 5 章「*interMedia* Text を使用した XML ドキュメントのデータ検索および取得」](#)を参照してください。

Oracle の XML アプリケーションを構築するためのツール

次に、Oracle の XML アプリケーションの開発に使用できるツールを示します。

- JDeveloper

第 13 章「[JDeveloper を使用した Oracle の XML アプリケーションの作成](#)」を参照してください。

- Internet file System (iFS)

第 14 章「[Internet File System \(iFS\) 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 の XML コンポーネント : XML ドキュメントの生成

図 3-7 および図 3-8 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 Oracle8i から SQL 問合せを介して XML ドキュメント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
- PL/SQL

Oracle の XML コンポーネントを使用した XML ドキュメントの生成 : Java

図 3-7 に、Oracle の XML の Java コンポーネント、およびそれを使用して XML ドキュメントを生成する方法を示します。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 Class Generator
 - 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。

A. XSL Servlet による処理 (XSU および XML Parser の使用も含む)

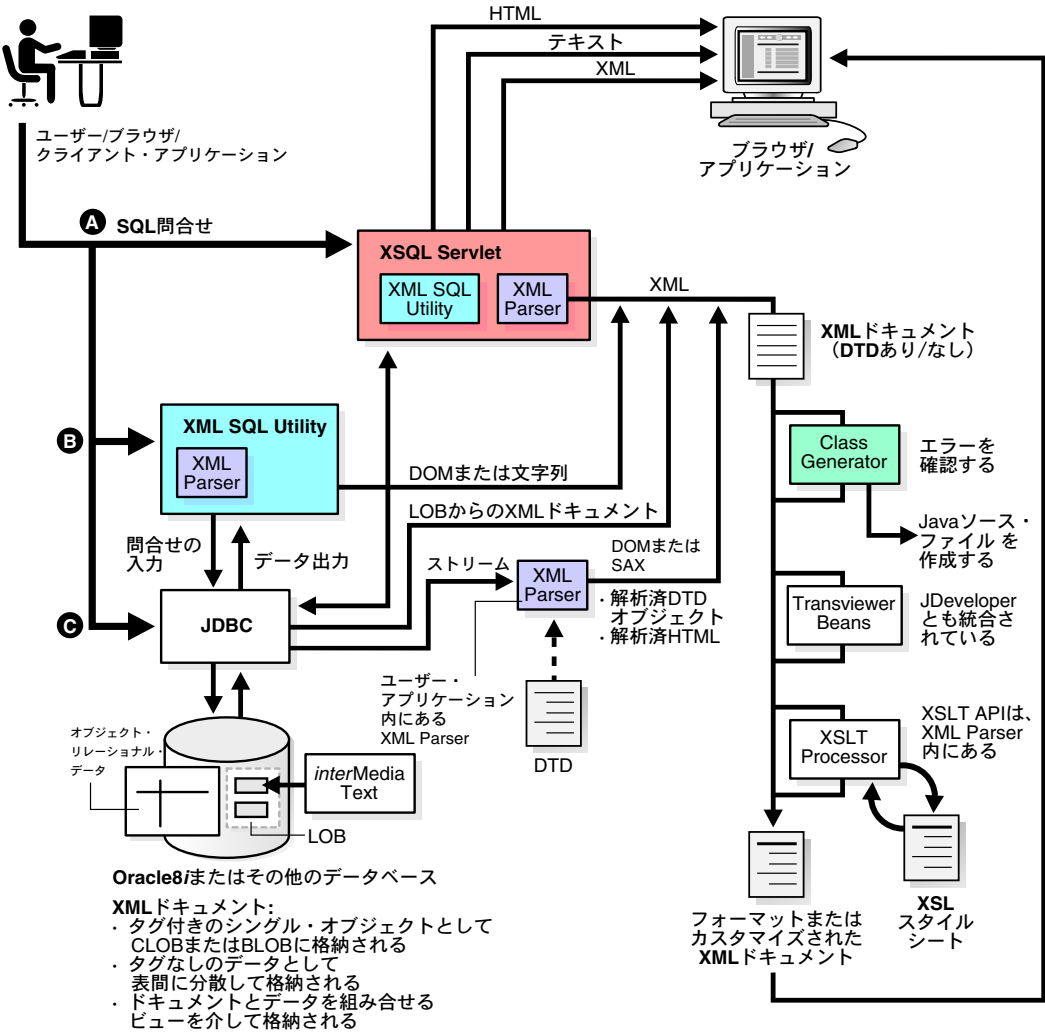
B. XSU による直接処理 (XML Parser の使用も含む)

C. JDBC による直接処理 (JDBC が XML Parser 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 Parser からの XML ドキュメント出力は、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 ドキュメントの用途に応じて、追加処理されます。

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

図 3-7 Oracle の XML コンポーネントを使用した XML ドキュメントの生成 - Java オプション



Oracle の XML コンポーネントを使用した XML ドキュメントの生成 : PL/SQL

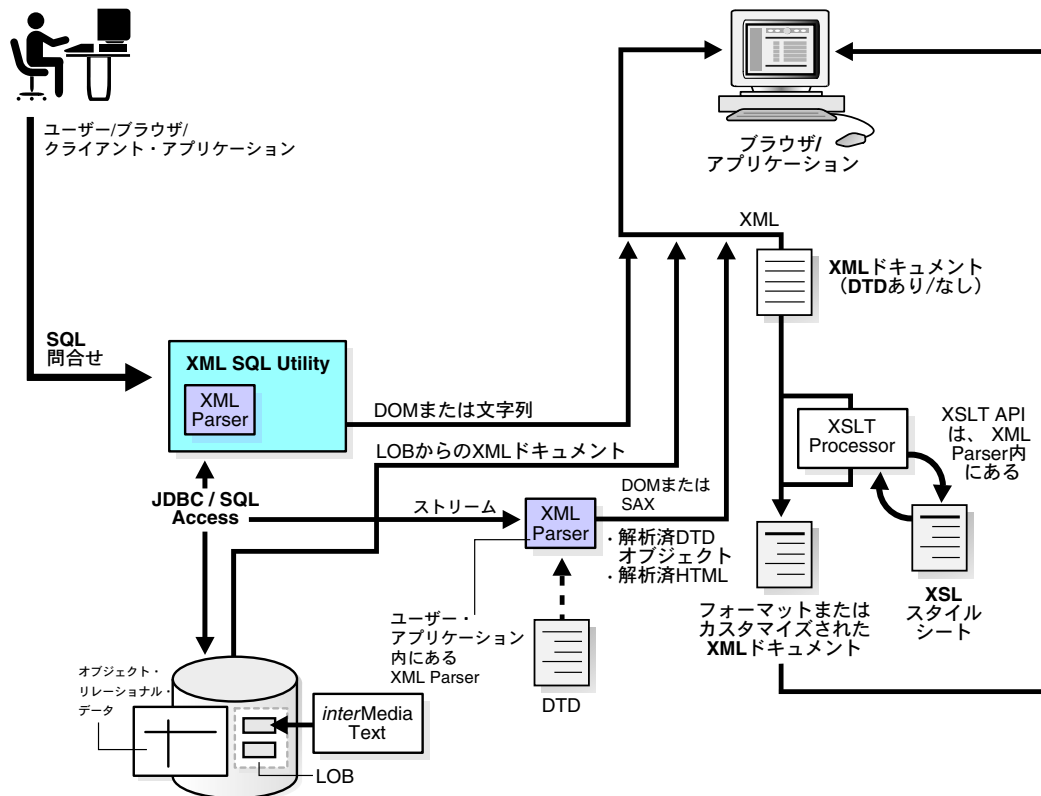
図 3-8 に、XML ドキュメントの生成に使用される PL/SQL 用 Oracle の XML コンポーネントを示します。次に、使用可能な PL/SQL 用 XML コンポーネントを示します。

- XDK for PL/SQL:
 - XSLT を含む XML Parser for PL/SQL バージョン 2
- 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 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 を使用した処理

図 3-8 Oracle の XML コンポーネントを使用した XML ドキュメントの生成 - PL/SQL オプション



Oracle8i またはその他のデータベース

XMLドキュメント:

- ・ タグ付きのシングル・オブジェクトとして
CLOBまたはBLOBに格納される
- ・ タグなしのデータとして
表間に分散して格納される
- ・ ドキュメントとデータを組み合わせる
ビューを介して格納される

FAQ - XML 全般

この項では、Oracle の XML 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。

XML の記述方法

質問

いくつかの XML の記事を読みましたが、XML を記述する方法がわかりません。C の場合は、次の手順で記述します。

1. UNIX で vi をオープンします。
2. ヘッダ・ファイルを挿入します。
3. メインをオープンします。
4. コードを記述します。
5. メインをクローズします。
6. コンパイルします (cc -o example example.c)。

XML では、これをどのように行いますか？XML は、多くの雑誌で取り上げられていますが、その記述方法は説明されていません。

回答

まずは、Oracle XSQL Servlet を使用して SQL を学習し、XML および XSLT 変換を実験的に使用することです。

XML および必要な Oracle のツール

質問

Oracle の XML テクノロジを使用してデータ・ファイルを生成するためのオプションを検討しています。生成されたデータ (XML) ・ファイルは、トランスレータを使用して、クライアント固有の EDI テキスト・ファイルに変換されます。

トランスレータは、クライアント側で XML ドキュメントの受入れ準備が整うまで使用されます。

1. 使用する必要があるデータベースの最新バージョンは何ですか？リリース 8.0.6 は持っていますが、Oracle8i が必要ですか？
2. XML をテキスト・ファイルに変換可能なトランスレータはありますか？XSL がそれに該当するツール製品ですか？
3. 変換用のマッピング情報の格納場所および格納方法を教えてください。

回答

中間層を使用せずにデータベースから直接 XML を生成する場合にのみ、Oracle8i が必要です。

PL/SQL から XML SQL Utility に Java コールを行うことができます。

XML ファイルはテキスト・ファイルですが、XSL を使用し、適切なスタイルシートを作成することによって、XML をほぼすべてのテキストベースのフォーマットに変換できます。

XML 形式の発注書の収集 : XML での RFP の作成

質問

XML および Oracle8i 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、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 8i 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、同社の優先ベンダーに対する別の「提案要求」を XML で作成する必要があります。そのため、データベースに対して挿入または更新問合せを実行する予定です。この場合、Oracle 8i にインストールする必要があるすべてのコンポーネントを教えてください。

回答

Java を使用した実装を前提とすると、XML Parser 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。

移植性：異なるベンダーの XML パーサーの使用

質問

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

- 異なるベンダー（Oracle、IBM など）の XML パーサーにはどのような違いがありますか？
- 現在、Oracle の XML Parser 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 /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利点です。

XML をサポートするブラウザ

質問

XML をサポートするブラウザのリストはありますか？

回答

次のブラウザが、XML 表示をサポートしています。

- Opera（バージョン 4.0 以上の XML）
- Citec Doczilla（XML および SGML ブラウザ）
- Indelv（XSL を使用してのみ XML ドキュメントを表示）
- 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

Oracle 8.0.x に対する XML サポート

質問

当社には、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る顧客がいます。その顧客は、ウォール街の大手機関です。同機関では、すべての現行システムがリリース 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。

この顧客から、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があるかどうか、また、これに対処するために使用できるアダプタ / カートリッジがあるかどうかについての質問が寄せられました。

回答

XML Parser、XSLT Processor、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 のすべてのコンポーネントは、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、これらのコンポーネントには、次の機能がありません。

- データベース内での XML コンポーネントの実行
- *interMedia* での XML の検索

これらは、両方とも Oracle8i 専用の機能です。

EDI および XML

質問

ベンダー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、難解なテクノロジーです。開発者が実際に読んだり、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 ドキュメントは簡単に読んだり、理解することができます。
- EDI 開発者の養成にはコストがかかります。

- 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 ドキュメントを別の XML ドキュメントに変換して、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（前述の項を参照）。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/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。

B2B の取引をサポートする Oracle のツール

質問

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

回答

オラクル社は、次の B2B の標準化機関に参加しています。

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

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

- 電子マーケットプレイスを実装するための独自のソリューションを提供する Oracle Exchange
- 組織内実装用の Oracle Integration Server（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出入れするための Oracle XML Gateway

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

XML に関するオラクル社の方針

質問

XML に関するオラクル社の方針を教えてください。

回答

XML に関するオラクル社の方針は、オラクル社がこれまでに蓄積したテクノロジー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 Oracle8i データベースおよび Advanced Queueing (AQ) と組み合わせて、ある程度の競合解消、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 Oracle8i のリリースを、競合解消、トランザクション確認、分散 2 フェーズ・コミット・トランザクションなどの点で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、オブジェクトに関連して、表または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。

オラクル社は、将来のリリースで、AQ を使用して XML ペイロードを送信する機能をサポートする計画です。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。現在、この計画を進めています。

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

XML Query

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

オラクル社は、XML Schema、XML Query、XSL、XLink/XPointer、XML Infoset、DOM および XML Core という、XML/XSL に関連した W3C のワーキング・グループに人材を積極的に派遣しています。

XML および BLOB（XML メッセージ内）

質問

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

回答

XML の場合、すべての文字を解析する必要があるため、RAW バイナリ・データを XML ドキュメント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。

CLOB の最大サイズ

質問

XML ファイルを CLOB として Oracle8i 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

最大サイズは 2GB です。LOB および CLOB の詳細は、『Oracle8i 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。

Oracle 7.3.4: XML を使用した他のベンダーへのデータ転送

質問

当社ではリリース 7.3.4 を使用しています。当グループでは、当社と取引先ベンダー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 サイトを見る限り、これを行うために、当グループは Oracle8i 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？当社が将来的に Oracle8 に移行することは確実ですが、現在進めているプロジェクトの次のフェーズの期間中（次の 3～4 か月）に移行するかどうかはわかりません。

回答

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

XML を使用して表にデータを挿入するために必要なソフトウェア

質問

表示するデータを選択し、XML を介して表にデータを挿入するためには、どのようなソフトウェアが必要ですか？現在、Solaris 上で Oracle8i 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
- JDK

最初の 3 つはオラクル社の製品です。

4 つ目は、Sun Microsystems 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

XML アプリケーションの構築に必要なソフトウェア

質問

現在、Solaris 2.6 上で CGI と PERL と Oracle7 によるアプリケーションを使用していますが、これを XML/XSL と JAVA と Oracle に変換する予定です。SGML、XML、JAVA など、ほぼすべてのテクノロジー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？

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. Oracle7.3 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 Parser が必要ですか？
4. Web サーバーと Oracle Database Server の間に、XSQL Servlet、Parser、JAVA VM、EJB、CORBA、SQLJ、JDBC、または UTL_HTTP などの Oracle パッケージを使用する必要がありますか？

回答

1. Apache を使用できます。Apache Web サーバーは、JDBC などの方法で Oracle とやりとりします。たとえば、XSQL Servlet があります。これは、サーブレット対応のすべての Web サーバー上で実行可能なサーブレットです。このサーブレットは Apache 上で動作し、Oracle データベースに対する JDBC ドライバを介してデータベースに接続します。
2. 今後も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せん。
3. 生成した XML の用途によって、必要である場合と必要でない場合があります。XML の生成および送信のみを行う場合、XML Parser 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 Parser が必要です。また、XML ドキュメントを受信し、それを解析して、いずれかの場所に格納する場合も、XML Parser が必要です。これについては、「XML SQL Utility」を参照してください。
4. 質問 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。

受注や出荷に使用する標準の DTD

質問

Oracle8i および XDK を実装済です。受注、出荷および通知のための基本的な標準の DTD は、どこで取得できますか？

回答

<http://xml.org> を参照してください。

DTD からデータベース・スキーマへの移行

質問

DTD からデータベース・スキーマに移行するためのツール製品はありますか？

回答

XML Schema 以外に、データ型を指定する方法がないため、現在オラクル社には DTD からデータベース・スキーマに移行するためのツール製品はありません。

XML へのスキーマのマップ

質問

当社のプロジェクトでは、クライアント用に、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。

表の設計および XML の生成（フラットな表またはオブジェクト / コレクション）に最適な方法を教えてください。

回答

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。

データベース内の XML のパフォーマンス

質問

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

回答

現在、XML 製品用のパフォーマンス標準 / 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。

より高速なレコード取出し

質問

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

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

回答

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

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

他の形式から XML への変換

質問

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

回答

HTML の場合、Tidy や JTidy などのユーティリティを使用して、XSLT によって変換可能である整形形式の HTML に変換できます。

ランダム・テキスト・フォーマットの場合、<http://www.unidex.com/xflat.htm> にある XFlat を試すことができます。XML99 で XFlat による表示を確認しましたが、問題はないようです。ただし、実証はされていません。

XML ファイルのサイズ制限

質問

XML ファイルのサイズに制限はありますか？

回答

ありません。

XML ドキュメントの最大サイズ

質問

1. CLOB を使用しない場合、表全体に PL/SQL（または SQL）用のデータを指定するための XML ドキュメントの最大サイズがありますか？
2. Oracle8i から XML ドキュメントに生成された XML ドキュメントの最大サイズは？

回答

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

Rational Rose ツールからのデータベース・スキーマの生成

質問

Oracle8i 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 / ジェネレータ（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、再使用できるように設計されていますが、大規模なフレームワークの中で開発されています。ユーザーは、いくつかのガイドライン（UML でのモデル化など）に従う必要があります。また、オラクル社製品のメリットを活かすには、基本となるクラスを使用する必要があります。

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

追加情報

XML に関するその他の FAQ

XML 関連の FAQ については、次のサイトも参照してください。

- <http://www.ucc.ie/xml/>
- <http://www.oasis-open.org/cover/>

XML/XSL 関連の推奨書籍

質問

XML/XSL 関連の推奨書籍があれば教えてください。

回答

WROX という出版社は多数の有用な書籍を出版しています。『XML Design and Implementation』（Paul Spencer 著）は、XML、XSL および開発について詳しく説明しています。

コメント

他にも、『XML Bible』は、XML および XSL に関する有効な書籍です。次の Web サイトで、最新の第 14 章を参照できます。

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

これを読むと、XSLT の理解を深めることができます。この章は無料でダウンロードできます。

第 II 部

XML SQL Utility (XSU) : データベースに対する XML の格納および取出し

第 II 部では、XML データを Oracle8i データベースに格納する方法およびデータベースから取り出す方法、および XML SQL Utility (XSU) を使用してこの作業を行う方法について説明します。

第 II 部に含まれる章は、次のとおりです。

- [第 4 章「XML SQL Utility \(XSU\) の使用」](#)

XML SQL Utility (XSU) の使用

この章の内容は次のとおりです。

- XML SQL Utility へのアクセス
- XML SQL Utility (XSU) の使用
- XML SQL Utility (XSU) を実行できる場所
- XSU 使用のガイドライン
 - マッピングの手引き
 - XSU 用のコマンドラインのフロントエンドの使用
 - ResultSet オブジェクトからの XML の生成
- XML SQL Utility for Java
 - 結果ページの区切り : skipRows および maxRows
 - ResultSet オブジェクトからの XML の生成
 - 該当する行がない場合の例外の呼出し
 - XML の格納
 - 挿入処理
 - 更新処理
 - 削除処理
- XML SQL Utility for PL/SQL の使用
 - XSU へのスタイルシートの設定 (PL/SQL)
 - XSU のバインド値 (PL/SQL)
 - DBMS_XMLSave を使用したデータベースへの XML の格納

-
- [PL/SQL での XSU による挿入処理](#)
 - [更新処理](#)
 - [削除処理](#)
 - [高度な使用方法](#)
 - [FAQ: XML SQL Utility \(XSU\)](#)

XML SQL Utility へのアクセス

XML SQL Utility (XSU) は Oracle8i に付属しており、次の 3 つのファイルで構成されています。

- \$ORACLE_HOME/rdbms/jlib/xsu12.jar - XSU を構成するすべての Java クラスを含みます。xsu12 には JDK1.2.x または JDBC2.x (あるいはその両方) が必要です。これは、データベース自体にロードされた XSU のバージョンです。
- \$ORACLE_HOME/rdbms/jlib/xsu111.jar - xsu12.jar と同じクラスを含みます。ただし、xsu111 には、JDK1.1.x および JDBC1.x が必要です。JDK1.1.8 は、クライアント側の Oracle がサポートする JDK の公式バージョンです。
- \$ORACLE_HOME/rdbms/admin/dbmsxsu.sql - XSU の PL/SQL API を構築する SQL スクリプトです。xsu12.jar は、dbmsxsu.sql が実行される前にデータベースにロードする必要があります。このスクリプトは、自動的にロードされます。

Oracle8i Installer は、デフォルトで XSU をファイル・システム (前述の指定場所) にインストールし、データベースにもロードします。最初のインストール時に Installer に XSU をインストールしないように指示した場合は、後で Installer を実行し、XSU および依存するコンポーネントをインストールできます。その場合は、XSU が自動的にデータベースにロードされることはありません。XSU をデータベースにロードするには、次の手順に従う必要があります。

1. XML Parser for Java をデータベースにロードしていない場合は、\$ORACLE_HOME/xdk/lib を選択します。このディレクトリにある xmlparserv2.jar を、データベースにロードする必要があります。この手順の詳細は、『Oracle8i Java スタッド・プロシージャ開発者ガイド』の「Java クラスのロード」を参照してください。
2. 次に、\$ORACLE_HOME/rdbms/admin を選択し、catxsu.sql スクリプトを実行します。

XML SQL Utility (XSU) の使用

XML は、データ交換のための形式として急速に確立されました。現在、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。通信のためには、これらのリレーショナル・データを XML に変換する必要があります。XML SQL Utility (XSU) は、すべての SQL 問合せ結果を XML に、または XML を SQL 問合せ結果に正規にマップすることによって、このデータ変換を簡単にします。

たとえば、scott のスキーマ (すべてのデータベースでデフォルト・スキーマとして使用可能) にある従業員表の結果を取り出すには、ユーティリティに対して次の問合せを発行します。

```
select * from scott.emp
```

デフォルト設定では、次の XML ドキュメントが結果として生成されます。

```
<?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>
```

同様のドキュメントのデータを同じ表に挿入することもできます。XSU は、XML ドキュメントを更新および削除する API も提供します。

XML SQL Utility (XSU) を使用する場合

XML SQL Utility (XSU) は、データベースからデータを取得およびデータベースへデータを挿入する基本的な機能を提供します。このユーティリティを使用すると、単純な変換操作 (各行に生成された ROW タグの名前の変更など) によって、正規マッピングを相互に行えます。複雑な変換は、業界標準の XSL (XML スタイルシート言語) Transformation をドキュメントに適用することによって行います。Oracle XML Parser は、変換を実行するために、強力な XSL プロセッサである XSLT をサポートしています。XSL Transformation は、XSU に直接登録できます。登録すると、XSU が生成した XML は自動的に変換されます。

注意： アプリケーションの主要な目的が Web ソースに対する XML ページの生成である場合は、XSQL Servlet の使用を検討してください。XSQL Servlet は、このプロセスを自動化する標準 Java サブレットです。このサブレットを使用すると、さらに単純な XML テンプレート形式の言語を使用して変換を指定できます。

XML SQL Utility (XSU) を実行できる場所

XML SQL Utility は Java で作成され、Java をサポートする次のようなすべての層で実行できます。

- データベース内
- Java をサポートする中間層アプリケーション・サーバー内 - 中間層サーバーから XSU をコールします。
- Web サーバー内 - Web サーバーの内部で実行するサブレットから XSU の Java API をコールします。
- クライアント側 - XSU の Java API を Java アプリケーションで使用するか、XSU のコマンドラインのフロントエンドを使用します。

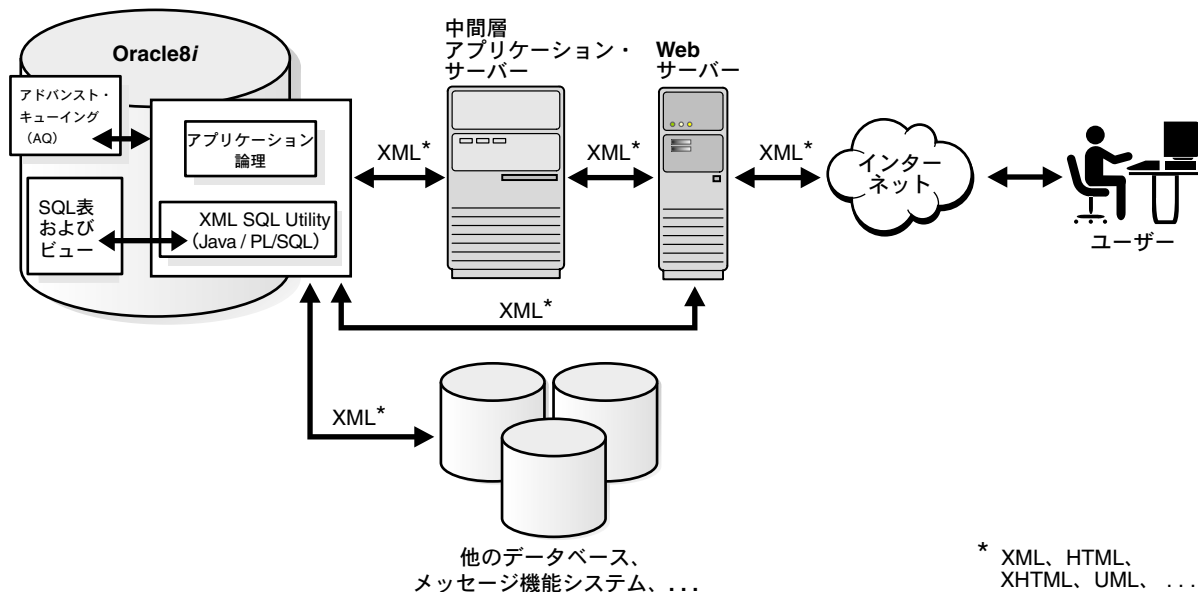
データベース内での XML SQL Utility の実行

XSU を構成する Java クラスは、Oracle8i JVM 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、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たり、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たり、また SQL を介して直接 XSU の機能を使用することができます。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 Oracle8i が必要です。

図 4-1 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ド・キュー内に置いて、他のシステムまたはクライアントにキューさせることができます。この XML は、データベース内のストアド・プロシージャから使用するか、Web サーバまたはアプリケーション・サーバを介して外部に送信できます。

図では、すべての矢印が両方向であることに注意してください。XSU は、データの挿入だけ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 4-1 データベース内での XML SQL Utility の実行



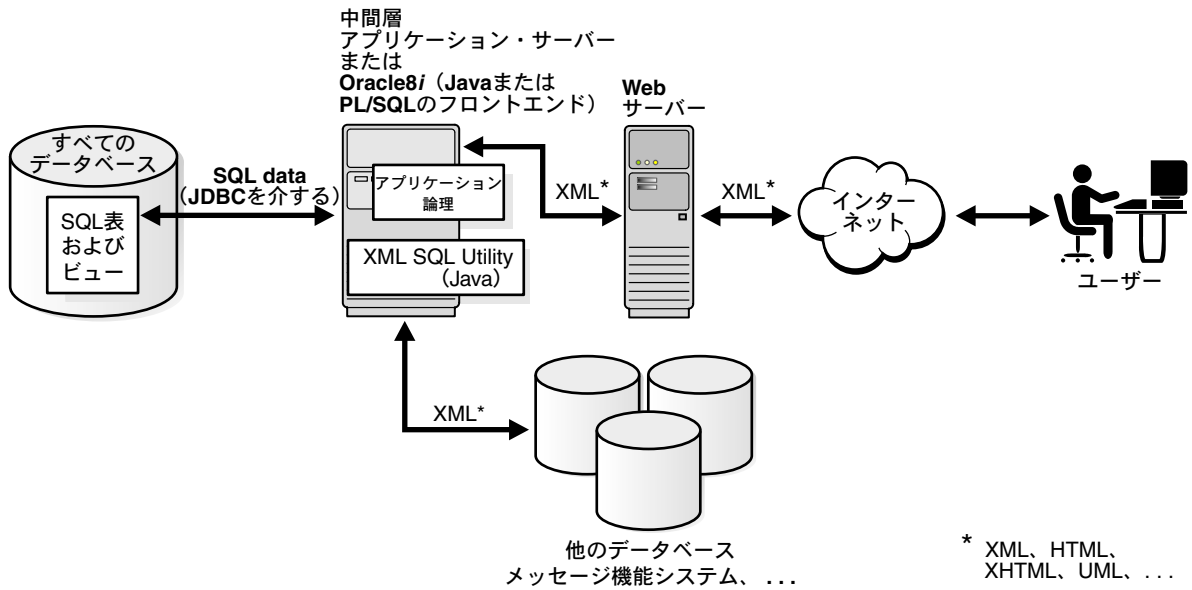
中間層内での XML SQL Utility の実行

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・サーバを使用する必要があるものもあります。このアプリケーション層には、Oracle データベース、Oracle Application Server、Java プログラムをサポートするサード・パーティ製のアプリケーション・サーバーなどがあります。

中間層で、SQL 問合せまたは ResultSet 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。この場合は、Java バージョンの XSU を使用して、中間層で実行している Java プログラムから直接 XSU の Java API をコールします。

図 4-2 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。JDBC ソースのデータは、中間層の XSU によって変換され、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。Oracle8i 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。

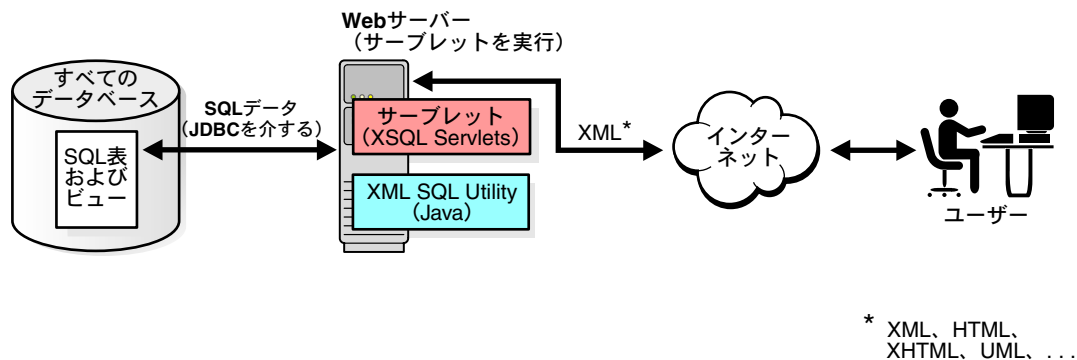
図 4-2 中間層内での XML SQL Utility の実行



Web サーバー内での XML SQL Utility の実行

Java バージョンの XSU は、Web サーバーが Java サブレットをサポートする場合は Web サーバー自体の中でも実行できます。XSU は Java サブレットから直接コールでき、XML を Web サーバーから生成したり戻すことができます。XSQL Servlet は Oracle が提供するサブレットであり、XML データを生成および保存するためにユーティリティをコールします。XSU を使用する主な目的が XML データの生成および保存である場合は、XSQL Servlet の使用をお勧めします。XSQL Servlet を使用すると大量のサブレット・コーディングを行う必要がなく、テンプレートを使用するより単純に、Web サーバーからの XML 処理を行うことができます。

参照： XSQL Servlet の使用については、[第 18 章「XSQL Servlet の使用」](#)を参照してください。



XML SQL Utility の動作要件

XML SQL Utility が動作するには、次のコンポーネントが必要です。

- **データ・ソース**: XML SQL Utility (XSU) には、JDBC ドライバが必要です。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 XSU に対していかなる保証およびサポートも行いません。
- **XML Parser**: XSU には、Oracle XML Parser が必要です。Oracle XML Parser は、Oracle8i の一部として提供されています。

XSU の機能

XSU は、コマンドラインのフロントエンド、Java API および PL/SQL API で構成されています。XSU の主な機能は次のとおりです。

- すべての SQL 問合せからの XML ドキュメントの生成をサポートします。Oracle8i 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 (文書型定義) の動的生成をサポートします。将来的には、XMLSchema もサポートする予定です。
- 生成での単純な変換 (ROW 要素のデフォルト・タグ名の変更など) をサポートします。XSL Transformation を登録して、生成した XML ドキュメントにその場で適用できます。
- XML ドキュメントを、ドキュメントの文字列または DOM 表現内で生成できます。
- データベース表 / ビューへの XML の挿入をサポートします。特定の XML ドキュメントの、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 ドキュメントは、これらの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わせることによって、簡単に生成し、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、Oracle8i のオブジェクト・リレーショナル機能を使用して、既存のリレーショナル・データから構造化データを作成できます。

XSU 使用のガイドライン

この項では、クライアント側のコマンドライン、Java API および PL/SQL API を含む、XSU の様々な機能の基本的な使用手順について説明します。詳細な例および使用方法については、この後の項を参照してください。

- マッピングの手引き
 - マッピング:SQL からの XML の生成
 - マッピング:XML からの SQL での表への格納 - 挿入、更新、削除
- XSU でのコマンドラインのフロントエンドの使用
- XSU の Java API の使用
- XSU の PL/SQL API の使用 - dbms_xmlquery および dbms_xmlsave

マッピングの手引き

様々な API の使用に関する手順を説明する前に、SQL から XML (XML の生成) および XML から SQL (XML の格納) へのデフォルト・マッピングについて理解する必要があります。前述のとおり、XML SQL Utility (XSU) は、SQL データから XML データへ (およびその逆) の正規マッピングを行います。次のマッピングについて説明します。

- マッピング:SQL からの XML の生成
- マッピング:XML からの SQL での表への格納

マッピング: SQL からの XML の生成

SQL 問合せが発行されると、XML ドキュメントを作成するために正規マッピングが行われます。scott スキーマにある、次のような構造を持つ 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
);
```

この表の要素を XML に変換するには、ユーティリティで使用可能な API の 1 つを使用して次の問合せを実行します。

```
select * from scott.emp;
```

この問合せを実行すると、ユーティリティは、すべての行の結果を含む ROWSET タグを含む XML ドキュメントを生成します。各行は、ROW タグ内にカプセル化されます。ROW タグには、各要素の行番号を識別する属性 num も含まれます。各スカラー要素は、XML 要素にマップされます。列名は、その要素のタグ名になります。この場合、有効な XML 識別子名ではないすべての列 (empno\$ や empno# など) は、SELECT 問合せで別名を指定して、有効な XML 名に変更する必要があります。

```
<?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>
```

デフォルト・マッピングは、API を介して様々なオプションを使用して変更できます。スカラー値は、フラットな XML ドキュメントにマップされます。

Oracle8i は、オブジェクト型、コレクション型およびオブジェクト参照の概念をサポートします。この概念によって、サーバー内で構造的なモデリングが行われます。オブジェクト型、コレクション型およびオブジェクト参照に対する XML へのマッピングによって、構造が保たれます。たとえば、部門の住所構造および従業員のリストを含む部門表について考えてみます。

AddressType は、住所オブジェクトの構造を定義するオブジェクト型です。

```
CREATE TYPE AddressType AS OBJECT (
  STREET VARCHAR2(20),
  CITY   VARCHAR2(20),
  STATE  CHAR(2),
  ZIP    VARCHAR2(10)
);
/
```

従業員の構造を定義する従業員型も示されています。従業員の住所が、住所型によってどのように定義されているかに注意してください。

```
CREATE TYPE EmployeeType AS OBJECT
(
    EMPNO NUMBER,
    ENAME VARCHAR2(20),
    SALARY NUMBER,
    EMPADDR AddressType
);
/
```

ここで、リスト型を定義することによって従業員のリストを作成できます。

```
CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;
/
```

これで、部門の住所および従業員のリストを含む部門表が作成されました。この表の各行には、従業員オブジェクトのネストしたコレクション型が含まれます。各オブジェクトには、名前、給与、住所などの従業員の説明が含まれます。

```
CREATE TABLE Dept
(
    DEPTNO NUMBER,
    DEPTNAME VARCHAR2(20),
    DEPTADDR AddressType,
    EMPLIST EmployeeListType
);
```

有効な値が部門表に格納されていると想定すると、表に対する SELECT 問合せの結果を、次のように XML ドキュメントにマップできます。

```
<?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>
      <EMPLOYEE_TYPE 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>
</EMPLOYEE_TYPE>
<!-- additional employee types within the employee list -->
</EMPLIST>
</ROW>
<!-- additional rows ... -->
</ROWSET>
```

前述の例からわかるように、オブジェクト型属性は XML ドキュメント内のネストした要素に、コレクション型は XML リストにマップされます。CURSOR 副問合せを使用しても、同様のネストが行われます。オブジェクト・ビューを使用すると、既存のリレーショナル表から同一の構造を再現できます。

マッピング: XML からの SQL での表への格納

XML SQL Utility を使用すると、XML ドキュメントを表の行にマップできます。また、最上位の列を更新したり、行を削除することができます。記憶域は、単純なマッピングを使用して要素のタグ名を列にマップします。XML 文字列は、デフォルト・マッピングを介して適切なデータ型に変換されます。XML 要素が構造化されている場合は、この要素を SQL オブジェクト型にマップできます。

挿入

たとえば、DEPTADDR 要素は、Dept 表に挿入すると AddressType SQL 型にマップされます。オブジェクト型およびコレクション型にマップする場合、一致する必要があるのは構造および名前のみです。したがって、AddressType2 型と AddressType 型の Address 要素の構造が同じであり、SQL 型属性名も同じである場合は、AddressType2 型の列から生成した XML を AddressType 型の列にマップできます。

挿入は、INSERT 文を起動し、その文の VALUES 句に要素のすべての値をバインドする場合のみで実行できます。各 ROW 要素の内容は、挿入する値の個別の集合としてマップします。このため、前項で示した XML ドキュメントを Dept 表に挿入するように XSU に要求すると、XSU は次の形式の INSERT 文を生成します。

```
INSERT INTO Dept (DEPTNO, DEPTNAME, DEPTADDR, EMPLIST) VALUES (?, ?, ?, ?)
```

その後、次のように値をバインドします。

```
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'), ...))
```

XML ドキュメントに ROW 要素が 2 つ以上ある場合は、XSU は各 ROW に対して値をバインドし、文を実行します。挿入処理は、パッチで挿入およびコミットするように最適化できます。これらの説明については、4-33 ページの「[挿入処理](#)」を参照してください。

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 ドキュメントの 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 が使用するキー列のリストを、ユーザーが提供するように想定されています。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 ドキュメントを記述します。

```
<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 文が起動されます。

```
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 ドキュメント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。

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 文はドキュメントにあるすべての列を一致させようとします。次のようなドキュメントについて考えてみます。

```
<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、ユーティリティは次のような DELETE 文を ROW 要素ごとに 1 つ起動します。

```
DELETE FROM Dept WHERE DEPTNO = ? AND DEPTNAME = ? AND DEPTADDR = ?
binding,
DEPTNO <- 100
DEPTNAME <- Sports
DEPTADDR <- AddressType('100 Redwood Shores Pkwy','Redwood City','CA','94065')
```

これらすべての使用方法については、4-55 ページの「[高度な使用方法](#)」を参照してください。

XSU 用のコマンドラインのフロントエンドの使用

XSU には、単純なコマンドラインによるフロントエンドがあります。これを使用すると、XSU の XML 生成機能および XML 挿入機能にすばやくアクセスできます。現時点では、XSU のフロントエンドは XSU の更新機能と削除機能をサポートしていません。

コマンドラインは、Java クラス OracleXML を介して利用できます。このオプションを起動するには、次をコールします。

```
java OracleXML
```

前述のコールによって、フロントエンドの使用情報が出力されます。

XSU のフロントエンドを実行できるようにするには、まず実行可能ファイルがある場所を指定する必要があります。この場所を指定するには、XSU の Java ライブラリ (xsu12.jar または xsu111.jar) を CLASSPATH に追加します。

XSU は Oracle XML Parser および JDBC ドライバに依存しているため、XSU を実行するには、ここでこれらのコンポーネントの場所も指定しておく必要があります。これらの場所を指定するには、CLASSPATH に Oracle XML Parser の Java ライブラリ (xmlparserv2.jar) および JDBC ライブラリ (xsu12.jar を使用している場合は classes12.jar または xsu111.jar を使用している場合は classes111.jar) を含める必要があります。

XSU のフロントエンドを使用した XML の生成

生成機能を使用するには、getXML パラメータを使用して XSU をコールします。たとえば、scott スキーマにある emp 表を問い合わせて XML ドキュメントを生成するには、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。

OracleXML - getXML オプション

- -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。これが指定されない場合は、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せん。

- -conn "<JDBC_connect_string>"

JDBC データベース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 ドキュメントとともに DTD も生成するように XSU に指示します。

- `-rowsetTag "<tag_name>"`

ROWSET タグ（問合せによって戻されたレコードに対応するすべての XML 要素を囲むタグ）を指定します。デフォルトの行セット・タグは、ROWSET です。行セット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。行 ID 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、行の ID 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、リストの要素の数を追跡します（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）。行の ID 属性に空の文字列（""）を指定すると、XSU は属性を省略します。
- `-useNullAttrId`

要素が NULL であることを示すために属性 "NULL (TRUE/FALSE)" を使用するように XSU に指示します。
- `-styleSheet "<stylesheet URI>"`

XML PI（処理命令）にスタイルシートを指定します。
- `-stylesheetType "<stylesheet type>"`

XML PI（処理命令）にスタイルシートの型を指定します。
- `-errorTag "<error tag name>"`

エラー・タグを指定します。エラー・タグは、XML に書式化されたエラー・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呼び出すように XSU に指示します。
- `-maxRows "<maximum number of rows>"`
XML に変換するために取り出される行の最大数を指定します。
- `-skipRows "<number of rows to skip>"`
スキップされる行の数を指定します。
- `-encoding "<encoding name>"`
生成される XML のキャラクタ・セットのエンコーディングを指定します。
- `-dateFormat "<date format>"`
XML ドキュメント内の日付値用の日付書式を指定します。
- `-fileName "<SQL query fileName>" | <sql query>`
問合せを含むファイル名か、問合せ自体を指定します。

XSU のフロントエンドを使用した XML の挿入

XML ドキュメントを 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 ドキュメントの読み込み
- XML ドキュメント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。

OracleXML - putXML オプション

putXML オプションを次に示します。

- `-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。自動コミットをオン（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すると、行を囲むタグが XML ドキュメントで使用されていないと通知されます。
- `-dateFormat "<date format>"`

XML ドキュメント内の日付値用の書式を指定します。
- `-ignoreCase`

列名とタグ名を大文字 / 小文字を区別せずに一致させます（たとえば、ignoreCase がオンになっている場合、「EmpNo」と「EMPNO」は一致します。）
- `-fileName "<file name>" | -URL "<url>" | -xmlDoc "<xml document>"`

挿入する XML ドキュメントを指定します。fileName オプションはローカル・ファイルを指定し、URL はドキュメントをフェッチする URL を指定します。また、xmlDoc オプションは、XML ドキュメントをコマンドライン上の文字列としてインラインに格納します。
- `<tableName>`

値を挿入する表の名前です。

XML SQL Utility for Java

次の2つのXSUクラスは、Java プログラマにとって便利なクラスです。

- `oracle.xml.sql.query.OracleXMLQuery`: 特定のSQL問合せでXMLドキュメントを生成します。
- `oracle.xml.sql.dml.OracleXMLSave`: XMLドキュメントを表およびビューに挿入します。

XML の生成

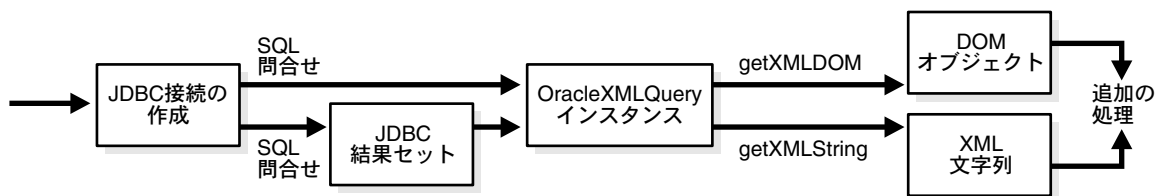
`OracleXMLQuery` クラスは、XSU の Java API の XML 生成部分を構成します。

図 4-3 に、`OracleXMLQuery` を使用する場合の基本的な処理の流れを示します。

XML を生成するには、次の手順に従います。

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 文字列のいずれかで取得します。

図 4-3 XML SQL Utility for Java を使用した XML の生成 : 基本的な処理の流れ

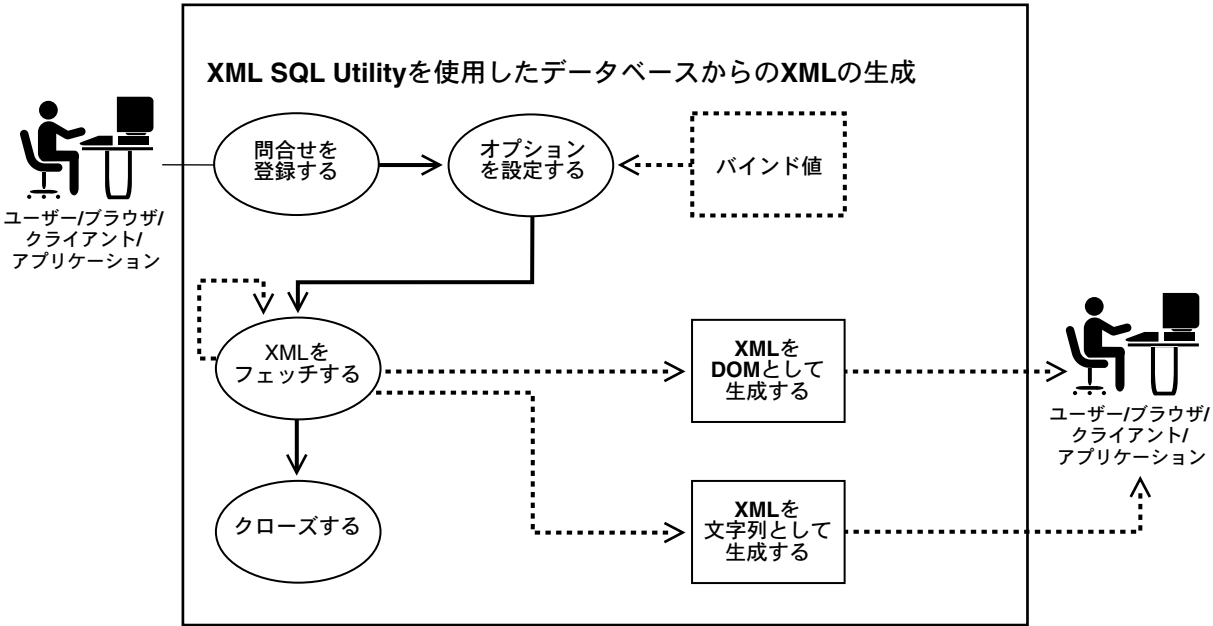


次に、単純な XML ドキュメントの生成方法を示します。

XSU: SQL 問合せからの XML の基本的な生成

これらの例では、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 ドキュメントを取得する方法を示します。図 4-4 を参照してください。

図 4-4 XML SQL Utility を使用した XML の生成: プロセスおよびオプション



XSU の例 1: emp 表からの文字列の生成

XML を取得する最初の手順は、データベースへの接続を作成することです。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。

```
//Oracle ドライバをインポートします。
import oracle.jdbc.driver.*;

//Oracle JDBC ドライバをロードします。
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
// 接続を作成します。
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` で識別される）に接続します。データベースへの接続には、JDBC Thin ドライバも使用できます。Thin ドライバは Pure Java で作成されており、アプレットやその他の Java プログラム内からコールできます。

参照： 詳細は、『Oracle8i Java 開発者ガイド』を参照してください。

Thin ドライバを使用した接続

次に、Thin ドライバを使用した接続の例を示します。

```
// 接続を作成します。
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.*;

//Oracle JDBC ドライバをロードします。
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。

OracleXMLQuery クラスのインスタンスの作成

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。

```
// 問合せ用クラスをインポートします。
import oracle.xml.sql.query.OracleXMLQuery;

OracleXMLQuery qry = new OracleXMLQuery (conn, "select * from emp");
```

これで、問合せクラスを使用できます。

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();
```

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、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。

次に、XML 文字列を抽出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.*;

// 文字列生成をテストするクラスです。
class testXMLSQL {

    public static void main(String[] argv)
    {

        try{
            // 接続を作成します。
            Connection conn = getConnection("scott","tiger");

            // 問合せ用クラスを作成します。
            OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

            //XML 文字列を取得します。
            String str = qry.getXMLString();

            //XML 出力を出力します。
            System.out.println(" The XML output is:\n"+str);
            // 問合せをクローズして、リソースを排除します。
            qry.close();
        }catch(SQLException e){
            System.out.println(e.toString());
        }
    }
}
```



```
// ユーザー名およびパスワードを指定して、接続を取得します。
private static Connection getConnection(String username, String password)
    throws SQLException
{
    //JDBC ドライバを登録します。
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    //OCI8 ドライバを使用して、接続を作成します。
    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 の例 2: emp 表からの DOM の生成 (Java)

DOM (ドキュメント・オブジェクト・モデル) は W3C が定義した標準であり、XML ドキュメント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、これらの子は子ノードになります。

ユーティリティが生成した XML から DOM ツリーを生成するには、ユーティリティに DOM ドキュメントを直接要求するだけです。ユーティリティは、ドキュメント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 ツリーを取得する例を示します。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{
            // 接続を作成します。
            Connection conn = getConnection("scott","tiger");

            // 問合せ用クラスを作成します。
            OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

            //DOM オブジェクトを取得します。実際の型は、Oracle XML Parser の DOM 表現です。
            // (XMLDocument)
            XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();

            //DOM から XML 出力を直接出力します。
            domDoc.print(System.out);

            // 文字列バッファに出力することもできます。
            StringWriter s = new StringWriter(10000);
            domDoc.print(new PrintWriter(s));
            System.out.println(" The string version ---> "+s.toString());

            qry.close(); // 問合せは必ずクローズする必要があります。
        }catch(Exception e){
            System.out.println(e.toString());
        }
    }

    // ユーザー名およびパスワードを指定して、接続を取得します。
    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 ResultSet 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ドキュメントを生成します。たとえば、一度に 100 行を取得するには、ユーザー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 設定および maxRows 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 を設定すると、XML に変換する行の数を制限できます。skipRows の値を 5 に設定し、maxRows の値を 10 に設定すると、ユーティリティは最初の 5 行をスキップし、次の 10 行に対する XML を生成します。

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、ユーティリティに、一度に 10 行ずつ変換して ResultSet 状態をアクティブに保つように指示します。こうするとユーティリティは、次に追加の結果を要求されたときに、前回の生成が終了した所から生成を開始します。

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ドキュメントを複数生成できます。

これは、maxRows パラメータおよび keepObjectOpen 機能を使用して処理できます。

keepObjectOpen 機能

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 機能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。

XSU の例 3: 結果ページの区切り : コール時の XML ページの生成 (Java)

次の例では、状態を保持する単純なクラスを作成し、このクラスがコールされるたびに次のページを作成します。

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;

public class pageTest
{
    Connection conn;
    OracleXMLQuery qry;
    ResultSet rset;
    Statement stmt;
    int lastRow = 0;

    public pageTest (String sqlQuery)
    {
        try{
            conn = getConnection("scott","tiger");
            //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            //                             ResultSet.CONCUR_READ_ONLY); // スクロール可能な結果セットを作成します。
            //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            //                             ResultSet.CONCUR_READ_ONLY); // スクロール可能な結果セットを作成します。
            stmt = conn.createStatement();
            ResultSet rset = stmt.executeQuery(sqlQuery); // 結果セットを取得します。
            rset.first();
            qry = new OracleXMLQuery(conn,rset); //OracleXMLQuery インスタンスを作成します。
            qry.keepCursorState(true); // 最初のフェッチの後に状態が消去されません。
            qry.setRaiseNoRowsException(true);
            qry.setRaiseException(true);
        }catch(SQLException e){
            System.out.println(e.toString());
        }
    }

    // 次の XML ページを戻します。
    public String getResult(int startRow, int endRow) throws SQLException
    {
        //rset.relative(lastRow-startRow); // 結果セット内をスクロールします。
        //rset.absolute(startRow); // 結果セット内をスクロールします。
        qry.setMaxRows(endRow-startRow); // 取り出す行の最大数を設定します。
        //System.out.println("before getxml");
        return qry.getXMLString();
    }
}
```

```
// 次のページを実行できるファンクションです。
public String nextPage() throws SQLException
{
    String result = getResult(lastRow,lastRow+10);
    lastRow+= 10;
    return result;
}

public void close() throws SQLException
{
    stmt.close();    // 文をクローズします。
    conn.close();    // 接続をクローズします。
    qry.close();     // 問合せをクローズします。
}

public static void main(String[] argv)
{
    String str;

    try{
        pageTest test = new pageTest("select e.* from emp e");

        int i = 0;
        //1 回に 1 ページのデータを取得します。
        while ((str = test.getResult(i,i+10))!= null)
        {
            System.out.println(str);
            i+= 10;
        }
        test.close();
    }catch(Exception e){
        e.printStackTrace(System.out);
    }
}

// ユーザー名およびパスワードを指定して、接続を取得します。
private static Connection getConnection(String user, String passwd)
    throws SQLException
{
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}
```

ResultSet オブジェクトからの XML の生成

これまでの、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、結果をページ区切形式で取り出す方法について説明しました。ただし Web では、次のページの結果のみでなく、前のページも取り出す必要がある場合があります。このスクロール可能機能を使用するには、Scrollable ResultSet を使用します。この Scrollable ResultSet オブジェクトを使用して結果セット内を移動し、移動するたびにユーティリティを使用して XML を生成します。

XSU の例 4: JDBC の ResultSets からの XML の生成 (Java)

この例では、JDBC の ResultSet の使用方法と、この ResultSet から XML を生成する方法について説明します。ResultSet は、ユーティリティが直接処理しない場合（バッチ・サイズの設定、値のバインディングなど）に使用が必要があることに注意してください。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); // スクロール可能な結果セットを作成します。

        ResultSet rset = stmt.executeQuery(); // 結果セットを取得します。
        qry = new OracleXMLQuery(conn,rset); // OracleXMLQuery インスタンスを作成します。
        qry.keepObjectOpen(true); // 最初のフェッチの後に状態が消去されません。
    }

    // 次の XML ページを戻します。
    public String getResult(int startRow, int endRow)
    {
        rset.scroll(lastRow-startRow); // 結果セット内をスクロールします。
        qry.setMaxRows(endRow-startRow); // 取り出す行の最大数を設定します。
        return qry.getXMLString();
    }
}
```

```
// 次のページを実行できるファンクションです。
public String nextPage()
{
    String result = getResult(lastRow,lastRow+10);
    lastRow+= 10;
    return result;
}

public void close()
{
    stmt.close();    // 文をクローズします。
    conn.close();    // 接続をクローズします。
    qry.close();     // 問合せをクローズします。
}

public void main(String[] argv)
{
    pageTest test = new pageTest("select * from emp");

    int i = 0;
    //1 回に 1 ページのデータを取得します。
    while ((str = test.getResult(i,i+10))!= null)
    {
        System.out.println(str);
        i+= 10;
    }
    test.close();
}
}
```

XSU の例 5: プロシージャの戻り値 (REF CURSOR) からの XML の生成 (Java)

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"); // 接続を作成します。

        //PL/SQL ファンクションをコールして、ResultSet オブジェクトを作成します。
        CallableStatement stmt =
            conn.prepareCall("begin ? := testRef.testRefCur(); end;");
```



```

stmt.registerOutParameter(1,OracleTypes.CURSOR); // 定義型を設定します。

stmt.execute(); // 文を実行します。
ResultSet rset = (ResultSet)stmt.getObject(1); //ResultSet を取得します。

OracleXMLQuery qry = new OracleXMLQuery(conn,rset); // 問合せクラスを準備します。
qry.setRaiseNoRowsException(true);
qry.setRaiseException(true);
qry.keepCursorState(true); // オプションを設定します (カーソルをアクティブの
                           // まま保持します)。
while ((str = qry.getXMLString())!= null)
    System.out.println(str);

qry.close(); // 問合せをクローズします。

// 文およびResultSet を指定したため、OracleXMLQuery インスタンスをクローズしても、
// これらはクローズされないことに注意してください。これらは、明示的にクローズする必要
// があります。
stmt.close();
conn.close();
}
// ユーザー名およびパスワードを指定して、接続を取得します。
private static Connection getConnection(String user, String passwd)
    throws SQLException
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
    return conn;
}
}

```

スタイルシートを適用するには、`applyStylesheet()` メソッドを使用します。これによって、出力が生成される前に強制的にスタイルシートが適用されます。

該当する行がない場合の例外の呼出し

ユーティリティは、処理する行がない場合は NULL 文字列を戻します。ただし、アプリケーションが例外ハンドラで処理できるように、生成される行がなくなるたびに例外を呼び出す方が効率的である場合があります。`setRaiseNoRowsException()` を設定すると、ユーティリティは、出力用に生成する行がなくなるたびに `OracleXMLSQLNoRowsException` を呼び出します。これはランタイムでの例外であり、必要がないときはキャッチする必要はありません。次のコードでは、NULL 文字列をチェックするかわりに例外を使用するように前述の例を拡張します。

XSU の例 6: 該当する行がない場合の例外 (Java)

```
public class pageTest {
    .... // クラス定義の残りです。

    public void main(String[] argv)
    {
        pageTest test = new pageTest("select * from emp");

        test.query.setRaiseNoRowsException(true); // 例外の生成を要求します。
        try
        {
            while(true)
                System.out.println(test.nextPage());
        }
        catch(oracle.xml.sql.OracleXMLNoRowsException)
        {
            System.out.println(" END OF OUTPUT ");
            test.close();
        }
    }
}
```

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。

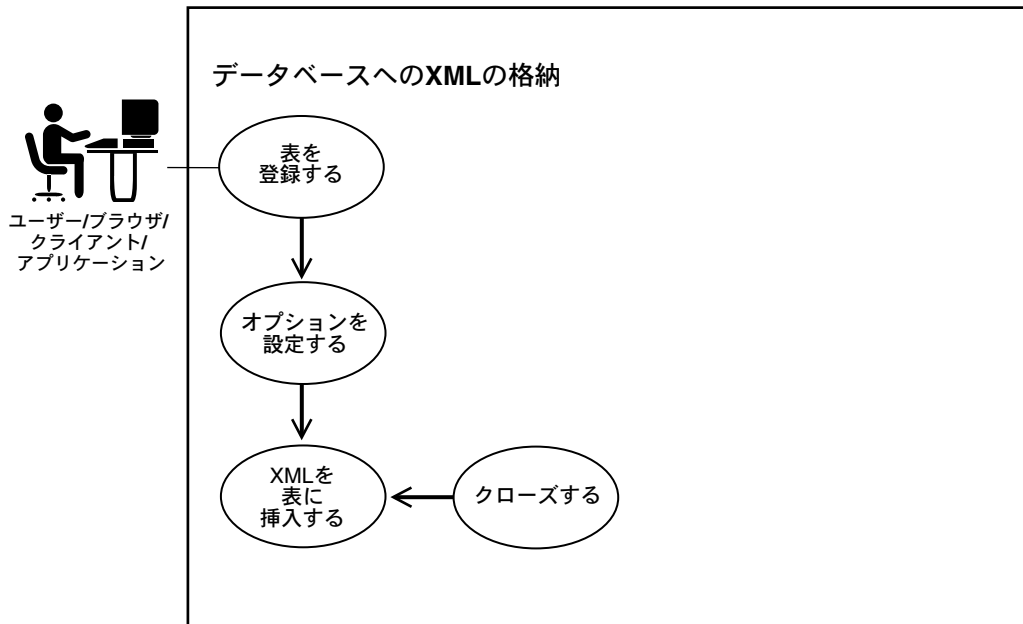
XML の格納

これまでの、問合せを XML へ変換する方法について説明しました。次に、ユーティリティを使用して 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 ドキュメント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 ドキュメントが解析され、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 4-5](#) に、XSU を使用した XML 格納のプロセスおよびオプションを示します。

図 4-5 XML SQL Utility を使用したデータベースへの XML の格納：プロセスおよびオプション



ドキュメントには 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。

挿入処理

表またはビューにドキュメントを挿入する手順は、表名またはビュー名を指定して、次にドキュメントを指定するのみです。ユーティリティはドキュメント（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、ユーティリティ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 表から生成したドキュメントを、簡単に emp 表に戻しています。

XSU 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。

```
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");
        // ユーザーがこのドキュメントを指定したと想定します。表に保存します。
        sav.insertXML(argv[1]);
        sav.close();
    }
    // ユーザー名およびパスワードを指定して、接続を取得します。
    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, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

列名に一致している入力 XML ドキュメント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 ドキュメントに送るとします。

```
<?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>
```

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 の例 8: 特定の列への 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[1] = "EMPNO";
        colNames[2] = "ENAME";
        colNames[3] = "JOB";

        sav.setUpdateColumnList(colNames); // 更新する列を設定します。

        // このドキュメントを最初の引数として指定すると想定します。
        sav.insertXML(argv[1]);
        sav.close();
    }
    // ユーザー名およびパスワードを指定して、接続を取得します。
    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、挿入したドキュメントに他の列の値（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、入力に存在する各 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。

更新処理

これまでは、XML ドキュメント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 ドキュメントを取得して、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 ... -->
  </ROW>
</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、ユーティリティ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 の例 9: keyColumn を使用した更新（Java）

この例では、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[1] = "EMPNO";
sav.setKeyColumnList(keyColNames);

// このドキュメントを最初の引数として指定すると想定します。
sav.updateXML(argv[1]);
sav.close();
}
// ユーザー名およびパスワードを指定して、接続を取得します。
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 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よくあります。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、ドキュメント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 ドキュメント内のすべての 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[1] = "EMPNO";
        sav.setKeyColumnList(keyColNames);

        // 更新する列のリストを作成します。
        // これを指定しない場合、XML ドキュメントの各 ROW 要素に対して新しい UPDATE 文を
        // 生成し、要素内にあるすべてのタグ値（キー列以外）を更新します。
        String[] updateColNames = new String[2];
        updateColNames[1] = "SAL";
        updateColNames[2] = "JOB";
        sav.setUpdateColumnList(updateColNames); // 更新する列を設定します。

        // このドキュメントを最初の引数として指定すると想定します。
        sav.updateXML(argv[1]);
        sav.close();
    }
    // ユーザー名およびパスワードを指定して、接続を取得します。
    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;
    }
}
```


削除処理

削除する場合は、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 ドキュメントの各 ROW 要素に対して作成されます。

XSU 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");

        // このドキュメントを最初の引数として指定すると想定します。
        sav.deleteXML(argv[1]);
        sav.close();
    }
    // ユーザー名およびパスワードを指定して、接続を取得します。
    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 ドキュメントと同じドキュメントを使用する場合は、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 ドキュメント内の各 ROW 要素にあるタグ名に基づいて構成されます。

XSU 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、setKeyColList() メソッド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[1] = "EMPNO";
        sav.setKeyColumnList(keyColNames);

        // このドキュメントを最初の引数として指定すると想定します。
        sav.deleteXML(argv[1]);
        sav.close();
    }
    // ユーザー名およびパスワードを指定して、接続を取得します。
    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、ドキュメント内のすべての ROW 要素に対して使用されます。

XML SQL Utility for PL/SQL の使用

XML SQL Utility PL/SQL API は、XML の生成および格納の方法が 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関クションの 1 つをコールしてコンテキストを作成し、ハンドルを取得してそのハンドルをすべての副問合せコールに使用します。

DBMS_XMLQuery を使用した XML の生成

XML を生成すると、XML ドキュメントを含む CLOB が生成されます。生成エンジンを使用する手順は次のとおりです。

1. DBMS_XMLQuery.getCtx ファンクションをコールし、このハンドルを問合せに (CLOB または VARCHAR2 として) 指定してコンテキスト・ハンドルを作成します。
2. DBMS_XMLQuery.bind ファンクションを使用して、考えられる値を問合せにバインドします。バインドは、名前を位置にバインドすることで行われます。たとえば、`select * from emp where empno = :EMPNO_VAR` のような問合せができます。ここでは、`setBindValue` ファンクションを使用して EMPNO_VAR 用の値をバインドします。
3. ROW タグ名、ROWSET タグ名、フェッチする行の数などの引数をオプションで設定します。
4. `getXML()` ファンクションを使用して、XML を CLOB としてフェッチします。`getXML` をコールして、DTD 付きまたは DTD なしで XML を生成できます。
5. コンテキストをクローズします。

次に、この PL/SQL パッケージを使用する例を示します。

XSU の例 13: 単純な問合せからの XML の生成 (PL/SQL)

この例では、`emp` 表から行を選択して XML ドキュメントを 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。ドキュメントのエンコーディングは、データベース・キャラクター・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
```

```
-- 問合せコンテキストを設定します。
queryCtx := DBMS_XMLQuery.newContext('select * from emp');

-- 結果を取得します。
result := DBMS_XMLQuery.getXMLClob(queryCtx);
-- 結果を使用して、表に挿入するか、またはメッセージとして送信します。
printClobOut(result);
DBMS_XMLQuery.closeContext(queryCtx); -- 問合せハンドルをクローズする必要があります。
end;
/
```

XSU の例 13a: 出力バッファへの CLOB の出力

printClobOut() は、CLOB を出力バッファへ出力する単純なプロシージャです。この PL/SQL コードを SQL*Plus で実行すると、CLOB の結果が画面に出力されます。結果を表示するには、serveroutput をオンに設定します。

次に、printClobOut を示します。

```
/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 の例 14: ROW タグ名および ROWSET タグ名の変更 (PL/SQL)

PL/SQL API は、ROW タグ名および ROWSET タグ名を変更する機能も提供します。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、およびドキュメント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。

-- ROW タグ名の設定

```
declare
  queryCtx DBMS_XMLQuery.ctxType;
  result CLOB;
```

```

begin
    -- 問合せコンテキストを設定します。
    queryCtx := DBMS_XMLQuery.newContext('select * from emp');

    DBMS_XMLQuery.setRowTag(queryCtx, 'EMP'); -- ROW タグ名を設定します。
    DBMS_XMLQuery.setRowSetTag(queryCtx, 'EMPSET'); -- ROWSET タグ名を設定します。

    result := DBMS_XMLQuery.getXML(queryCtx); -- 結果を取得します。

    printClobOut(result); -- 結果を出力します。
    DBMS_XMLQuery.closeContext(queryCtx); -- 問合せハンドルをクローズします。
end;
/

```

結果として生成される XML ドキュメントには、EMPSET ドキュメント要素、および EMP タグによって区切られた各行が含まれます。

XSU の例 15: setMaxRows() および setSkipRows() を使用した結果のページ区切り

問合せの生成の結果は、setMaxRows ファンクションおよび setSkipRows 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。skipRows パラメータは、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 に行が書き込まれないときには例外が呼び出されます。これは、終了条件としてキャッチおよび使用できます。

```

-- 結果のページ区切り

declare
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin
    -- 問合せコンテキストを設定します。
    queryCtx := DBMS_XMLQuery.newContext('select * from emp');

```

```
DBMS_XMLQuery.setSkipRows(queryCtx,3); -- スキップする行数を設定します。
DBMS_XMLQuery.setMaxRows(queryCtx,10); -- フェッチごとの行の最大数を設定します。

result := DBMS_XMLQuery.getXML(queryCtx); -- 最初の結果を取得します。

printClobOut(result); -- 結果を出力します。これはユーザーのルーチンです。
DBMS_XMLQuery.setSkipRows(queryCtx,0); -- これ以降は行をスキップしないでください。

DBMS_XMLQuery.setRaiseNoRowsException(queryCtx,true);
-- no rows 例外を呼び出します。

begin
  loop -- 永続的にループします。
    result := DBMS_XMLQuery.getXML(queryCtx); -- 次のバッチを取得します。
    printClobOut(result); -- 次のバッチの 10 行を出力します。
  end loop;
exception
  when others then
    -- dbms_output.put_line(sqlerrm);
    null; -- 終了条件です。何も行いません。
end;
DBMS_XMLQuery.closeContext(queryCtx); -- ハンドルをクローズします。
end;
/
```

XSU へのスタイルシートの設定 (PL/SQL)

PL/SQL API は、生成の前に、結果の XML にスタイル・ヘッダーを設定するか、またはスタイルシート自体を適用する機能を提供します。スタイルシート自体を適用する場合は、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 ドキュメントを 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。この場合、ユーティリティは内部で DOM ドキュメントを生成し、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。setStyleSheetHeader() プロシージャは、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。一方、useStyleSheet() 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)

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。変数の前には「:」を付けて、バインド変数であることを示す必要があります。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、ENAME のみを「Scott」に変更し、問合せを再実行するには次の文を実行します。

```
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 の例 15a: SQL 文への値のバインド

次の例は、SQL 文でのバインド変数の使用方法を示します。

```
declare
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin

    queryCtx := DBMS_XMLQuery.newContext(
        'select * from emp where empno = :EMPNO and ename = :ENAME');

    DBMS_XMLQuery.clearBindValues(queryCtx);
    DBMS_XMLQuery.setBindValue(queryCtx, 'EMPNO', 7566);
```

```
DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'JONES');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);

DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'Scott');

result := DBMS_XMLQuery.getXML(queryCtx);

--printClobOut(result);
end;
/
```

DBMS_XMLSave を使用したデータベースへの XML の格納

XML SQL Utility 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 ファンクションを使用して、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もオプションで指定できます。この場合は、キー列名と一致する XML ドキュメント内のタグが 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、指定したドキュメントの各 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 ドキュメントを指定します。
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。

PL/SQL での XSU による挿入処理

表またはビューにドキュメントを挿入する手順は、表名またはビュー名、およびドキュメントを指定するのみです。ユーティリティはドキュメント（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、ユーティリティ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 表から生成したドキュメントを、比較的簡単に emp 表に戻します。

XSU の例 16: すべての列への値の挿入 (PL/SQL)

この例では、プロシージャ insProc を作成します。このプロシージャは、XML ドキュメントを CLOB、およびそのドキュメントを挿入する表名として受け入れてから、このドキュメントを挿入します。

```
create or replace procedure insProc(xmlDoc IN CLOB, tableName IN VARCHAR2) is
    insCtx DBMS_XMLSave.ctxType;
    rows number;
begin
    insCtx := DBMS_XMLSave.newContext (tableName); -- コンテキスト・ハンドルを取得します。
    rows := DBMS_XMLSave.insertXML (insCtx, xmlDoc); -- ドキュメントが挿入されます。
    DBMS_XMLSave.closeContext (insCtx);           -- ハンドルがクローズされます。
end;
/
```

これでこのプロシージャは、すべての XML ドキュメント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 ドキュメント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 XML ドキュメントに送る場合は、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>
<!-- 追加の行 -->
</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 の例 17: 特定の列への値の挿入（PL/SQL）

値を挿入する必要がない列がある場合もあります。取得する値が完全なセットではない場合などで、残りの列用に使用するトリガーまたはデフォルト値が必要な場合などです。次のコードは、これを行う方法を示しています。

従業員番号、名前および仕事の値のみを取得し、給与、マネージャ、部門番号および雇用日のフィールドには自動的に値が挿入されると想定します。挿入を実行する列名のリストを作成し、このリストを DBMS_XMLSave プロシージャに渡します。これらの値は、setUpdateColumnName() プロシージャを繰り返しコールし、更新する列名を各コールごとに指定することによって設定できます。列名設定は、clearUpdateColumnNames() を使用して消去できます。

```
create or replace procedure testInsert( xmlDoc IN clob) is
    insCtx DBMS_XMLSave.ctxType;
    doc clob;
    rows number;
begin

    insCtx := DBMS_XMLSave.newContext('scott.emp'); -- 保存コンテキストを取得します。

    DBMS_XMLSave.clearUpdateColumnList(insCtx); -- 更新設定を消去します。

    -- 更新する列を値のリストとして設定します。
    DBMS_XMLSave.setUpdateColumn(insCtx, 'EMPNO');
    DBMS_XMLSave.setUpdateColumn(insCtx, 'ENAME');
    DBMS_XMLSave.setUpdatecolumn(insCtx, 'JOB');

    -- ドキュメントを挿入します。EMPNO、ENAME および JOB 列にのみ挿入されます。
    rows := DBMS_XMLSave.insertXML(insCtx, xmlDoc);
    DBMS_XMLSave.closeContext(insCtx);

end;
/
```

プロシージャをコールして CLOB をドキュメントとして指定すると、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、挿入したドキュメントに他の列の値 (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。

更新処理

これまでは XML ドキュメント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 ドキュメントを取得して、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、ユーティリティ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 の例 18: keyColumns を使用した XML ドキュメントの更新 (PL/SQL)

```

/.....

create or replace procedure testUpdate ( xmlDoc IN clob) is
    updCtx DBMS_XMLSave.ctxType;
    rows number;
begin

    updCtx := DBMS_XMLSave.newContext('scott.emp'); -- コンテキストを取得します。
    DBMS_XMLSave.clearUpdateColumnList (updCtx); -- 更新の設定を消去します。

    DBMS_XMLSave.setKeyColumn (updCtx, 'EMPNO'); -- EMPNO をキー列として設定します。
    rows := DBMS_XMLSave.updateXML (updCtx, xmlDoc); -- 表を更新します。
    DBMS_XMLSave.closeContext (updCtx);          -- コンテキストをクローズします。

end;
/

```

この例では、プロシージャが、前述のドキュメントを含む 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 の例 19: 更新する列のリストの指定 (PL/SQL)

列のリストを指定して更新する必要がある場合もあります。列のリストを指定すると、すべての ROW 要素に対して同じ UPDATE 文を使用できるため、より高速に処理できます。また、ドキュメント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 ドキュメント内のすべての ROW 要素と同じであることがわかっている場合は、setUpdateColumnNames() 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'); -- EMPNO をキー列として設定します。

```

```

-- 更新する列のリストを設定します。
DBMS_XMLSave.setUpdateColumn(updCtx, 'SAL');
DBMS_XMLSave.setUpdateColumn(updCtx, 'JOB');

rows := DBMS_XMLSave.updateXML(updCtx, xmlDoc); -- XMLドキュメントを更新します。
DBMS_XMLSave.closeContext(updCtx); -- ハンドルをクローズします。

end;
/

```

削除処理

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 ドキュメントの各 ROW 要素に対して作成されます。

XSU の例 20: ROW ごとの削除操作 (PL/SQL)

次の削除の例について考えます。

```

create or replace procedure testDelete(xmlDoc IN clob) is
  delCtx DBMS_XMLSave.ctxType;
  rows number;
begin
  delCtx := DBMS_XMLSave.newContext('scott.emp');
  DBMS_XMLSave.setKeyColumn(delCtx, 'EMPNO');

  rows := DBMS_XMLSave.deleteXML(delCtx, xmlDoc);
  DBMS_XMLSave.closeContext(delCtx);

end;
/

```

更新の例で示した XML ドキュメントと同じドキュメントを使用する場合は、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 ドキュメント内の各 ROW 要素にあるタグ名に基づいて構成されます。

XSU の例 21: キー値の指定による削除 (PL/SQL)

DELETE 文に述語としてキー値のみを使用する場合は、setKeyColumn ファンクションを使用してこれを設定できます。

```
create or replace package testDML AS
    saveCtx DBMS_XMLSave.ctxType := null;    -- 単一の静的変数です。

    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'); -- コンテキストを作成します。
    DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');    -- キー列名を設定します。
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、ドキュメント内のすべての ROW 要素に対して使用されます。

XSU の例 22: コンテキスト・ハンドルの再使用 (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、パッケージ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.ctxHandle := null;    -- 単一の静的変数です。

    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(xmlDoc);
    end;

    procedure update(xmlDoc in clob) is
    begin
        DBMS_XMLSave.updateXML(xmlDoc);
    end;

    procedure delete(xmlDoc in clob) is
    begin
        DBMS_XMLSave.deleteXML(xmlDoc);
    end;

    begin
        saveCtx := DBMS_XMLSave.getCtx('scott.emp'); -- コンテキストを作成します。
        DBMS_XMLSave.setKeyColumnName('EMPNO');      -- キー列名を設定します。
    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使用します。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** 表を更新できません。

```
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。

高度な使用方法

Java での例外処理

OracleXMLSQLException クラス

ユーティリティは、処理中に発生したすべての例外をキャッチし、ランタイム例外である `oracle.xml.sql.SQLException` を発生させます。このため、コールするプログラムはこの例外を常にキャッチする必要がありません。プログラムがこの例外をキャッチする場合は、適切な処置を行ってください。例外クラスは、エラー・メッセージおよび親である例外（ある場合）も取得するファンクションを提供します。たとえば、次に示すプログラムはランタイムの例外をキャッチし、親である例外を取得します。

OracleXMLNoRowsException クラス

この例外は、`setRaiseNoRowsException` が生成中に `OracleXMLQuery` クラスに設定されている場合に生成されます。これは、`OracleXMLSQLException` クラスのサブクラスであり、生成中の行処理に対する終了標識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");

        // 問題のある問合せであり、例外を生成します。
        OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp where sd
= 322323");

        qry.setRaiseException(true); // 例外の呼出しを要求します。

        try{
            String str = qry.getXMLString();
        }catch(oracle.xml.sql.SQLException e)
        {
            // 元の例外を取得します。
            Exception parent = e.getParentException();
            if (parent instanceof java.sql.SQLException)
            {
                // 他の操作を実行します。ここでは、単純に出力します。
                System.out.println(" Caught SQL Exception:"+parent.getMessage());
            }
        }
    }
}
```

```
    }
    else
        System.out.println(" Exception caught..." + e.getMessage());
    }
}
// ユーザー名およびパスワードを指定して、接続を取得します。
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;
}
}
```

PL/SQL での例外処理

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');

    -- Raise Exception を TRUE に設定します。
    DBMS_XMLQuery.setRaiseException(queryCtx, true);
    DBMS_XMLQuery.setRaiseNoRowsException(queryCtx, true);

    -- 元の例外を取得するために、Propagate Original Exception を TRUE に設定します。
    DBMS_XMLQuery.propagateOriginalException(queryCtx, true);
    result := DBMS_XMLQuery.getXML(queryCtx);

exception
    when others then
        -- 元の例外を取得します。
        DBMS_XMLQuery.getExceptionContent(queryCtx, errorNum, errorMsg);
        dbms_output.put_line(' Exception caught ' || TO_CHAR(errorNum)
                               || errorMsg );
end;
/
```

FAQ: XML SQL Utility (XSU)

XSU を使用して XML を格納する場合のスキーマ構造

質問

次の XML が customer.xml ファイルにあります。

```
<ROWSET>
  <ROW num="1">
    <CUSTOMER>
      <CUSTOMERID>1044</CUSTOMERID>
      <FIRSTNAME>Paul</FIRSTNAME>
      <LASTNAME>Astoria</LASTNAME>
      <HOMEADDRESS>
        <STREET>123 Cherry Lane</STREET>
        <CITY>SF</CITY>
        <STATE>CA</STATE>
        <ZIP>94132</ZIP>
      </HOMEADDRESS>
    </CUSTOMER>
  </ROW>
</ROWSET>
```

XSU を使用してこの XML を格納する場合に使用するデータベース・スキーマ構造を教えてください。

回答

この例ではレベルが複数であるため（ネストした構造であるため）、オブジェクト・リレーショナル・スキーマを使用する必要があります。前述の XML は、オブジェクト・リレーショナル・スキーマに正規マッピングされます。適切なデータベース・スキーマは次のとおりです。

```
create type address_type as object
(
  street varchar2(40),
  city varchar2(20),
  state varchar2(10),
  zip varchar2(10)
);
/
create type customer_type as object
(
  customerid number(10),
  firstname varchar2(20),
```

```
lastname varchar2(20),
homeaddress address_type
);
/
create table customer_tab ( customer customer_type);
```

XSU を介して customer.xml をリレーショナル・スキーマにロードする場合は、リレーショナル・スキーマのビューにオブジェクトを作成します。

たとえば、次のすべての情報を含むリレーショナル表があるとして。

```
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、最上位にカスタム・オブジェクトを持つカスタム・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、あまりお勧めできません。

複数表への XML データの格納

質問

XML SQL Utility は、複数表にまたがって XML データを格納できますか？

回答

現在 XML SQL Utility (XSU) は、単一表にのみ格納できます。このユーティリティは、XML ドキュメントの正規表現をすべての表 / 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せん。これを行うには、XSLT を使用していずれかのドキュメントを複数のドキュメントに変換し、これらのドキュメントを別々に挿入します。また、複数表にまたがってビュー（必要に応じてオブジェクト・ビュー）を定義し、そのビューに挿入することでも実行できます。このビューが更新不可能（複雑な結合などのために）な場合は、ビューにまたがって INSTEAD OF トリガーを使用して、挿入を実行できます。

XML SQL Utility を使用した属性に格納された XML のロード

質問

データのいくつかが属性に格納されている XML のロードに XML SQL Utility を使用したいのですが、XML SQL Utility 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 ドキュメント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）。XML SQL Utility 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。

XML SQL Utility の大 / 小文字区別 : 大 / 小文字区別なし機能の使用など

質問

次の XML ドキュメント（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、XML SQL Utility は大 / 小文字を区別します。このためユーティリティは、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、XML SQL Utility 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、または大 / 小文字区別なし機能を使用します。

DTD からのデータベース・スキーマの生成

質問

DTD を指定すると、XML SQL Utility はデータベース・スキーマを生成しますか？

回答

生成しません。DTD には多くの欠点があるため、この機能は使用できません。XML Schema が標準化されたら、この機能は実現可能になります。

XML SQL Utility コマンドラインの使用

質問

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 Driver 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

JDBC Thin ドライバ接続文字列の例は、
「jdbc:oracle:thin:<user>/<password>@<hostname>:<port number>:<DB SID>」です。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。有効な OCI8 接続文字列は、「jdbc:oracle:oci8:<user>/<password>@<hostname>」です。

INSERT、DELETE、UPDATE 後の XML SQL Utility のコミット

質問

XML SQL Utility は挿入 / 削除 / 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、XML SQL Utility は一度に多数の INSERT（または DELETE や UPDATE）文を実行します。バッチ処理で同時に実行される文の数は、setBatchSize 機能を使用してオーバーライドできます。

デフォルトでは、XML SQL Utility 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。

最後に、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。

第 III 部

XML を使用したコンテンツおよび ドキュメントの管理

第 III 部では、*interMedia Text* を使用して、XML データベース・アプリケーションの検索および取出しを拡張および高速化する方法について説明します。*interMedia Text* は、データベース以外のアプリケーションからの検索および取出しにも使用できます。

第 III 部には、*interMedia Text* の例および XML ベースのコンテンツ管理の例が含まれます。

第 III 部に含まれる章は、次のとおりです。

- 第 5 章「*interMedia Text* を使用した XML ドキュメントのデータ検索および取得」
- 第 6 章「XML によるコンテンツのカスタマイズ: *Dynamic News* アプリケーション」
- 第 7 章「XML によるデータ表示のパーソナライズ: *Portal-to-Go*」
- 第 8 章「XML および XSQL を使用した表示のカスタマイズ: *Flight Finder*」

interMedia Text を使用した XML ドキュメントの データ検索および取得

この章の内容は次のとおりです。

- 概要
- [interMedia Text の概要](#)
- [interMedia Text のインストール](#)
- [interMedia Text のユーザーおよびロール](#)
- [CONTAINS 演算子を使用した問合せ](#)
- [この章の例に関する前提](#)
- [interMedia Text を使用した XML ドキュメントの検索](#)
 - [interMedia Text 索引](#)
 - [CTX_DDL PL/SQL パッケージの使用](#)
 - [interMedia Text 索引の作成](#)
- [interMedia Text を使用した問合せアプリケーションの構築](#)
 - [テキスト問合せ式](#)
 - [属性セクションでの問合せ](#)
 - [SECTION GROUPS の問合せ](#)
 - [interMedia Text を使用した問合せアプリケーションの作成手順](#)
 - [ドキュメント・タイプが区別される XML ドキュメントでのセクションの作成](#)
 - [問合せ結果の表示](#)
- [FAQ: interMedia Text](#)

概要

この章では、次の *interMedia Text* 機能について説明します。

- 索引の作成
- *interMedia Text* を使用した XML 問合せアプリケーションの構築

注意： *interMedia Text* は、サーバーベースの実装です。

interMedia Text の概要

interMedia Text は、XML ドキュメントを検索するために使用できます。これは、Oracle8i に格納されたすべてのテキストまたはドキュメントを索引付けすることによって、Oracle8i を拡張します。また、オペレーティング・システム内のドキュメント（フラット・ファイル）および URL を検索することもできます。

interMedia Text を使用すると、次の操作を行うことができます。

- 一般的な標準 SQL を使用した内容ベースの問合せ（特定のワードを含むテキストおよびドキュメントの検索など）。
- Oracle8i を使用して、従来の関連情報を使用した統合的な方法によって、テキストおよびドキュメントを管理するためのファイルベースのテキスト・アプリケーション。
- 英語ドキュメントの概念検索。
- CTX_DOC PL/SQL パッケージを使用した英語ドキュメントのテーマ分析。
- 検索で一致したワードのハイライト。*interMedia Text* を使用すると、ドキュメントを様々な方法でレンダリングできます。たとえば、問合せ用語（英語の場合、ワード問合せのワードまたは ABOUT 問合せのテーマ）をハイライトさせてドキュメントを表示できます。これを行うには、CTX_DOC.MARKUP または HIGHLIGHT プロシージャを使用します。
- *interMedia Text* PL/SQL パッケージを使用したドキュメントの表示およびシソーラスのメンテナンス。

interMedia Text は、他の *interMedia* 製品（Web コンテンツ管理アプリケーションのためのイメージ、オーディオ、ビデオおよび地理検索サービス）とともにパッケージ化されています。

interMedia Text を使用しなくても、データベースに格納された XML データを直接問い合わせることはできます。ただし、*interMedia Text* を使用すると、問合せのパフォーマンスが向上します。

参照： *interMedia Text* の詳細は、『Oracle8i *interMedia Text* リファレンス』を参照してください。

interMedia Text のインストール

interMedia Text は、基本的に、CTXSYS が所有する一連のスキーマ・オブジェクトです。これらのオブジェクトは、Oracle カーネルにリンクされます。スキーマ・オブジェクトは、Oracle8i をインストールするときに存在します。

interMedia Text のユーザーおよびロール

interMedia Text では、CTXSYS ユーザーを使用してユーザーおよび CTXAPP ロールを管理し、*interMedia Text* のプリファレンスを作成および削除したり、次の *interMedia Text* PL/SQL パッケージを使用することができます。

CTXSYS ユーザー

このユーザーは、インストール時に作成されます。*interMedia Text* ユーザーをこのユーザーとして管理します。CTXSYS ユーザーには、次の権限があります。

- システム定義のプリファレンスを変更する権限
- 他のユーザーのプリファレンスを削除および変更する権限
- CTX_ADM PL/SQL パッケージのプロシージャをコールして、サーバーの起動およびシステム・パラメータの設定を行う権限
- ctxsrv サーバーを起動する権限
- すべてのシステム定義のビューを問い合わせる権限
- CTXAPP ロールを使用してユーザーのすべてのタスクを実行する権限

CTXAPP ロール

すべてのユーザーは、*interMedia Text* 索引を作成したり、テキスト問合せを発行することができます。その他のタスクを実行する場合は、CTXAPP ロールを使用してください。これは、システム定義のロールです。このロールを使用して、次のタスクを実行できます。

- *interMedia Text* のプリファレンスの作成および削除
- CTX_DDL パッケージなどの *interMedia Text* PL/SQL パッケージの使用

CONTAINS 演算子を使用した問合せ

interMedia Text の主な用途は、CONTAINS 演算子を使用することです。CONTAINS 演算子は、テキスト問合せ用の問合せ式を指定するために、SELECT 文の WHERE 句で使用されます。

参照：[「interMedia Text を使用した問合せアプリケーションの構築」](#)を参照してください。

CONTAINS 構文

CONTAINS 構文は次のとおりです。

```
CONTAINS (
    [schema.]column,
    text_query          VARCHAR2,
    [label              NUMBER])
RETURN NUMBER;
```

パラメータは次のとおりです。

表 5-1 CONTAINS 演算子：構文の説明

構文	説明
[schema.]column	検索するテキスト列を指定します。この列には、それに対応付けられた Text 索引が必要です。
text_query	列内での検索を定義する問合せ式を指定します。
label	オプションで、CONTAINS 演算子によって生成されたスコアを識別するラベルを指定します。
戻り値	<p>CONTAINS は、選択された各行について、そのドキュメント行が問合せにどの程度関連しているかを示す 0 ～ 100 の数値を返します。数値 0 は、Oracle がその行内で一致するデータを検索しなかったことを意味します。</p> <p>このスコアは、SCORE 演算子を使用して取得できます。</p> <p>注意：この数値を取得するには、ラベルを指定して SCORE 演算子を使用してください。</p>

CONTAINS の例 1

次の例は、SELECT 構文で CONTAINS 演算子を使用する方法を示します。

```
SELECT id FROM my_table
WHERE
  CONTAINS (my_column, 'receipts') > 0
```

CONTAINS 関数の 'receipts' パラメータは、テキスト問合せ式とも呼びます。この式の使用
方法の例については、「[テキスト問合せ式](#)」を参照してください。

注意： CONTAINS 関数を含む SQL 文を実行するには、Text 索引が必要です。

CONTAINS の例 2: ラベルを指定した SCORE 演算子の使用

次の例は、テキスト列内で「Oracle」というワードを含むすべてのドキュメントを検索しま
す。各行のスコアは、SCORE 演算子で 1 というラベルを使用して選択されます。

```
SELECT SCORE(1), title from newsindex
WHERE CONTAINS(text, 'oracle', 1) > 0;
```

CONTAINS 演算子の後には、> 0 の部分が続く必要があります。この構文は、CONTAINS
演算子が計算したスコアの値が 0（ゼロ）より大きい場合に、その行が選択されることを指
定します。

CONTAINS の例 3: SCORE 演算子の使用

SELECT 句などで SCORE 演算子がコールされると、演算子は次の例のとおり、ラベルの値
を参照する必要があります。

```
SELECT SCORE(1), title from newsindex
WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

この章の例に関する前提

XML テキストは、文字セマンティクスを持つ Oracle8i データベース表内の VARCHAR2 型または CLOB 型です。

interMedia Text は、ファイル・システムまたは URL 上のドキュメントも処理できます。ここでは、これらのドキュメント・タイプについては考慮しません。

この章に含まれる例を簡単にするために、*interMedia Text* オプションの一部分について考えてみます。

次のことを前提としています。

- すべての XML データが、US-ASCII の 7 ビット・キャラクタ・セットを使用して表されます。
- 「*」などの文字を空白として処理するか、またはワードの一部として処理するかについての問題は、含まれません。
- テキスト索引を実装する Oracle スキーマ・オブジェクトの格納特性については、考慮しません。
- CREATE INDEX 文または ALTER INDEX 文内の SECTION GROUP パラメータ¹に注目します。次に、CREATE INDEX 文で SECTION GROUP パラメータを使用した例を示します。

```
CREATE INDEX my_index
  ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' ) ;
```

- 特に、AUTO_SECTION_GROUP および XML_SECTION_GROUP を使用することに注目します。
- XML データ（タグ付けまたはマークアップされたデータ）の処理方法に注目します。*interMedia Text* は、他にも様々な種類のデータを処理します。

¹ CREATE INDEX 文および ALTER INDEX 文には、他のパラメータ・タイプも使用できます。使用可能なパラメータ・タイプは、DATASTORE、FILTER、LEXER、STOPLIST および WORDLIST です。

interMedia Text を使用した XML ドキュメントの検索

XML ドキュメントからデータを検索および取得するには、次の作業を行う必要があります。

- *interMedia Text* 索引の作成
- 問合せアプリケーションの構築

これらの作業の手順については、次の項を参照してください。

interMedia Text 索引

interMedia Text 用の索引を作成するには、次の手順に従います。

1. 使用する必要があるロールを決定し、ctxapp 権限を付与します。[「*interMedia Text* のユーザーおよびロール」](#)を参照してください。
2. 表およびデータがない場合は、これらを設定します。
3. 問合せに CONTAINS を使用するための *interMedia Text* 索引を作成します。
4. *interMedia Text* 索引をパラメータ化するためのプリファレンスを作成します。
5. CTX_DDL パッケージの Create_Section_Group および Add_Field_Section プロシージャを使用して、*interMedia Text* 索引をパラメータ化します。

CTX_DDL PL/SQL パッケージの使用

オラクル社が提供する CTX_DDL PL/SQL パッケージによって、*interMedia* Text 索引に必要なオブジェクトが作成および管理されます。CTX_DDL パッケージのストアド・プロシージャおよびストアド・ファンクションの詳細なリストは、『Oracle8i *interMedia* Text リファレンス』を参照してください。CTX_DDL パッケージを実行する場合は、CTXAPP ロールを使用してください。

この章では、次の CTX_DDL プロシージャを使用します。

- create_preference
- create_section_group
- add_attr_section
- add_field_section
- add_special_section
- add_zone_section

各 CTX パッケージに必要なロールのリスト表示

CTX パッケージで使用するロールが不明な場合は、次のスクリプトを実行します。これによって、各 CTX パッケージに必要なロールが表示されます。

```
connect ctxsys/ctxsys
column "Package" format a15
column "Role needed to execute Package" format a30

select dba_objects.object_name "Package",
       dba_tab_privs.grantee "Role needed to execute Package"
from dba_objects, dba_tab_privs
where dba_objects.object_name = dba_tab_privs.table_name (+)
      and dba_objects.object_type in ( 'PACKAGE', 'PROCEDURE', 'FUNCTION' )
      and dba_objects.object_name like 'CTX_%'
order by "Role needed to execute Package";
```

これによって、次の結果が出力されます。

Package	Role needed to execute Package
-----	-----
Ctx_Ddl	CTXAPP
Ctx_Output	CTXAPP
Ctx_Thes	CTXAPP
Ctx_Contains	PUBLIC
Ctx_Query	PUBLIC
Ctx_Doc	PUBLIC
Ctx_Adm	

CTX_DDL プロシージャ

次に、この章で使用する CTX_DDL プロシージャを、各プロシージャの引数および説明とともに示します。

procedure create_preference			
argument name	type	in/out	default?
-----	-----	-----	-----
preference_name	varchar2	in	
object_name	varchar2	in	
procedure create_section_group			
argument name	type	in/out	default?
-----	-----	-----	-----
group_name	varchar2	in	
group_type	varchar2	in	
procedure add_attr_section			
argument name	type	in/out	default?
-----	-----	-----	-----
group_name	varchar2	in	
section_name	varchar2	in	
tag	varchar2	in	
procedure add_field_section			
argument name	type	in/out	default?
-----	-----	-----	-----
group_name	varchar2	in	
section_name	varchar2	in	
tag	varchar2	in	
visible	boolean	in	default
procedure add_special_section			
argument name	type	in/out	default?
-----	-----	-----	-----
group_name	varchar2	in	
section_name	varchar2	in	
procedure add_zone_section			
argument name	type	in/out	default?
-----	-----	-----	-----
group_name	varchar2	in	
section_name	varchar2	in	
tag	varchar2	in	

XML_SECTION_GROUP 属性セクション

Oracle8i リリース 8.1.6 以降では、XML セクション・グループによって、属性値の範囲内で索引付けおよび検索を行う機能が提供されます。次の行を含むドキュメントについて考えてみます。

```
<comment author="jeeves">
  I really like interMedia Text
</comment>
```

ここで、XML セクション・グループが、1 つの属性セクションを提供します。これによって、たとえば次のとおり、索引に属性値を挿入できるようになります。

```
ctx_ddl.add_attr_section('mysg','author','comment@author');
```

構文は、他の add_section コールと同様です。最初の引数はセクション・グループの名前、2 番目はセクションの名前、3 番目は <タグ名>@<属性名> という形式のタグです。この場合、interMedia Text は、comment タグの author 属性の内容を「author」セクションとして索引付けすることを命令されます。

次に、問合せ構文を示します。これは、他のセクションの場合も同様です。

```
WHERE CONTAINS ( ... , 'jeeves WITHIN author...', ... ) ...
```

これによって、ドキュメントが検索されます。

属性値の識別によるセクション検索

属性セクションを使用すると、属性の内容を検索できます。ただし、属性値を使用して検索するセクションを指定することはできません。たとえば、次のドキュメントを想定します。

```
<comment author="jeeves">
  I really like interMedia Text
</comment>
```

このドキュメントは、次の問合せによって検索できます。

```
jeeves within comment@author
```

これは、author 属性の値に「jeeves」という単語を含む comment 要素を持つすべてのドキュメントを検索することに相当します。

ただし、次のような問合せを行うことはできません。

```
interMedia within comment where (@author = "jeeves")
```

この方法では、jeeves が author である comment 要素で interMedia が使用されているすべてのドキュメントが検索されます。この機能（属性値の識別によるセクション検索）は、Oracle8i の将来のリリースでサポートされる予定です。

動的追加セクション

セクション・グループは索引の作成前に定義されるため、リリース 8.1.5 では、構造化ドキュメント・セットの変更に対する機能が制限されます。ドキュメントで新しいタグを初めて使用する場合、または新しい DOCTYPE を初めて取得する場合は、索引を再作成して、それらのタグの使用を開始する必要があります。

8.1.6 以上のリリースでは、索引を再作成しないで、新しいセクションを既存の索引に追加できます。この場合、次に示すとおり、ALTER INDEX 文に ADD SECTION パラメータ文字列構文を使用します。

```
add zone section <section_name> tag <tag>
add field section <section_name> tag <tag> [ visible | invisible ]
```

たとえば、タグ title を使用して、tsec という名前の新しいゾーン・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add zone section tsec tag title')
```

タグ author を使用して、asec という名前の新しいフィールド・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author')
```

このフィールド・セクションは、add_field_section を使用した場合と同様、デフォルトで表示されません。これを参照可能なフィールド・セクションとして追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author visible')
```

動的追加セクションは、索引メタデータを変更するのみで、索引を再作成しません。これは、動的追加セクションが、操作後に索引付けされたすべてのドキュメントに影響し、既存のドキュメントには影響しないことを意味します。動的追加セクションを含むドキュメントにすでに索引が付いている場合、これらのドキュメントは、(通常、索引列をそれ自身に更新して) 再度索引付けするために、手動でマーク付けする必要があります。

この操作では、特殊セクションの追加はサポートされません。特殊セクションを追加するには、すべてのドキュメントを再度索引付けする必要があります。この操作は、再作成を使用してオンラインで行うことはできませんが、比較的高速な操作です。

AUTO_SECTION_GROUP

AUTO_SECTION_GROUP グループ・タイプを使用して、XML ドキュメント内の開始タグ / 終了タグの各組用のゾーン・セクションを自動的に作成します。XML タグから導出されるセクション名は、XML の場合と同様、大文字 / 小文字が区別されます。

属性セクションは、属性を含む XML タグ用に自動的に作成されます。属性セクションには、属性名 @ タグ名という形式の名前が付けられます。

停止セクション、空タグ、処理命令およびコメントは索引付けされません。

自動セクション・グループには、次の制限事項が適用されます。

- 自動セクション・グループには、ゾーン、フィールドまたは特殊セクションを追加できません。
- 自動セクション作成では、XML ドキュメント・タイプ（ルート要素）は索引付けされません。ただし、停止セクションは、ドキュメント・タイプで定義できます。
- 接頭辞および名前空間を含めた索引付きタグの長さは、64 文字以下である必要があります。
- 64 文字より長いタグは、索引付けされません。

XML ドキュメントでの自動セクション作成

次のコマンドは、AUTO_SECTION_GROUP グループ・タイプを使用して、`auto` というセクション・グループを作成します。このセクション・グループは、自動的に XML ドキュメント内のタグからセクションを作成します。

```
begin
ctx_ddl_create_section_group('auto', 'AUTO_SECTION_GROUP');
end;
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null_filter section group auto');
```

interMedia Text 索引の作成

次の例は、interMedia Text 索引の作成方法を示します。この例は、チュートリアル方式で示します。interMedia Text 索引を作成するには、次の手順に従います。

1. 必要なロールの決定および ctxapp 権限の付与
2. データの設定（ない場合）
3. CONTAINS を使用するための interMedia Text 索引の作成
4. プリファレンスの作成: プリファレンスを使用したパラメータ化の表現
5. プリファレンスのパラメータ化

必要なロールの決定および ctxapp 権限の付与

必要なロールを決定します。「各 CTX パッケージに必要なロールのリスト表示」を参照し、適切な権限を付与します。

```
CONNECT system/manager
GRANT ctxapp to scott;
CONNECT scott/tiger
```

データの設定（ない場合）

次の例では、my_table という表を作成し、この表にデータを挿入します。

-- いくつかのデータを設定します。

```
CREATE TABLE my_table (id number(5) primary key, my_column clob );

INSERT INTO my_table VALUES ( 1,
    '<author>Fred Smith</author>' ||
    '<document>I had a nice weekend in the mountains.</document>' );
INSERT INTO my_table VALUES ( 2,
    '<author>Jack Jones</author>' ||
    '<document>My month at the coast was relaxing.</document>' );
INSERT INTO my_table VALUES ( 3,
    '<publisher>Dog House</publisher>' ||
    '<document>His year in Provence was fulfilling.</document>' ||
    '<junk>banana</junk>' );
COMMIT;
```

CONTAINS を使用するための *interMedia Text* 索引の作成

次の例は、CONTAINS を使用するために Text 索引が必要であることを示します。

CONTAINS を使用するために必要な *interMedia Text* 索引

この例は、CONTAINS を使用するために *interMedia Text* 索引が必要であることを示します。

```
SELECT my_column FROM my_table
WHERE CONTAINS
      ( my_column, 'smith WITHIN author'
      ) > 0;
```

次のエラー・メッセージが取得されます。

DRG-10599: 列に索引が作成されていません。

デフォルトのパラメータ化 *interMedia Text* 索引の作成

この例は、デフォルトのパラメータ化された *interMedia Text* 索引が XML 機能をサポートしないことを示します。

```
CREATE INDEX my_index ON my_table ( my_column )
      INDEXTYPE IS Ctxsys.Context /* デフォルトを表します。*/;

SELECT my_column FROM my_table
WHERE CONTAINS
      ( my_column, 'smith WITHIN author'
      ) > 0;
```

次のエラー・メッセージが取得されます。

DRG-10837: セクション author が存在しません。

「[プリファレンスのパラメータ化](#)」を参照してください。

プリファレンスの作成：プリファレンスを使用したパラメータ化の表現

「my_section_group」というプリファレンスについて考えてみます。これは、参照する前に作成しておく必要があります。この例は、プリファレンスを使用してパラメータ化を表現する必要があることを示します。

```
DROP INDEX my_index;

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );
```

次のエラー・メッセージが取得されます。

DRG-12203: セクション・グループ my_section_group がありません。

プリファレンスのパラメータ化

'my_section_group' は、作成するのみでなく、パラメータ化する必要もあります。CTX_DDL.Create_Section_Group を使用します。

my_section_group プリファレンスのパラメータ化

この例は、my_section_group プリファレンスをパラメータ化する必要があることを示します。

```
DROP INDEX my_index;

BEGIN
  Ctx_Ddl.Create_Section_Group
    ( group_name => 'my_section_group',
      group_type => 'xml_section_group'
    );
end;
/

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS
    (my_column, 'smith WITHIN author'
    ) > 0;
```

次のエラー・メッセージが取得されます。

DRG-10837: セクション author が存在しません。

CTX_DDL.Create_Section_Group および CTX_DDL.Add_Field_Section を適切に使用した 'my_section_group' プリファレンスのパラメータ化

この例は、interMedia Text 索引および my_section_group プリファレンスを作成した後に、そのプリファレンスを適切にパラメータ化する必要があることを示します。

- まず、CTX_DDL.create_section_group を使用して、セクション・グループを作成します。
- 次に、問合せ文の WITHIN の後に使用されるすべてのタグに CTX_DDL.Add_Field_Section を使用します。

参照： CTX_DDL パッケージの概要については、「[CTX_OBJECTS 表および CTX_OBJECT_ATTRIBUTES ビューの使用](#)」を参照してください。また、CTX_DDL の詳細は、『Oracle8i interMedia Text リファレンス』を参照してください。

interMedia Text の例 1: 索引の作成 - プリファレンスの作成および適切なパラメータ化

```
DROP INDEX my_index;
BEGIN
Ctx_Ddl.Drop_Section_Group
  ( group_name => 'my_section_group'
  );
END;
/

BEGIN
Ctx_Ddl.Create_Section_Group /* HTML ではなく XML を処理しています。*/
  ( group_name => 'my_section_group',
    group_type => 'xml_section_group'
  );

Ctx_Ddl.Add_Field_Section /* これがキーです。*/
  ( group_name  =>'my_section_group',
    section_name =>'author', /* WITHIN の後に使用されるすべてのタグにこれを行います。*/
    tag         =>'author'
  );

Ctx_Ddl.Add_Field_Section /* これがキーです。*/
  ( group_name  =>'my_section_group',
    section_name =>'document', /*WITHIN の後に使用されるすべてのタグにこれを行います。
*/
    tag         =>'document'
  );
```



```

...
END;
/

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS Ctxsys.Context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS
    ( my_column, 'smith WITHIN author'
      ) > 0;

```

最後の例が正しい例です。

interMedia Text の例 2: AUTO_SECTION_GROUP を使用したセクション・グループの作成

次のコマンドは、AUTO_SECTION_GROUP グループ・タイプを使用して、autogroup というセクション・グループを作成します。このセクション・グループは、自動的に XML ドキュメント内のタグからセクションを作成します。

```

BEGIN
ctx_ddl.create_section_group('autogroup', 'AUTO_SECTION_GROUP');
END;

```

次の文を使用すると、ドキュメントを索引付けできます。

```

CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
  parameters('filter ctxsys.null_filter section group autogroup');

```

注意： XML セクション・グループのみに属性セクションを追加できません。AUTO_SECTION_GROUP を使用すると、属性セクションが自動的に作成されます。自動的に作成された属性セクションには、タグ名 @ 属性名という形式の名前が付けられます。

***interMedia* Text の例 3: XML_SECTION_GROUP を使用したセクション・グループの作成**

次のコマンドは、XML_SECTION_GROUP グループ・タイプを使用して、xmlgroup というセクション・グループを作成します。

```
BEGIN  
ctx_ddl.create_section_group('xmlgroup', 'XML_SECTION_GROUP');  
END;
```

CTX_DDL.ADD_SECTION を使用して、このグループにセクションを追加できます。

次の文を使用すると、ドキュメントを索引付けできます。

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context  
parameters('filter ctxsys.null_filter section group xmlgroup');
```

interMedia Text を使用した問合せアプリケーションの構築

この項の内容は次のとおりです。

- [テキスト問合せ式](#)
- [属性セクションでの問合せ](#)
- [SECTION GROUPS の問合せ](#)
- [interMedia Text を使用した問合せアプリケーションの作成手順](#)
- [ドキュメント・タイプが区別される XML ドキュメントでのセクションの作成](#)

テキスト問合せ式

テキスト問合せ式を使用すると、次の操作を実行できます。

- ワード・ベース基準の表現。これは、ブール演算子によるワード組合せ検索、近接検索、句検索などのテキスト検索業界標準です。
- (...CONTAINS (my_column,'About (customizing XML presentations)')...) などのテーマ・ベースの検索基準の実行。通常、「customizing XML presentations」という句は、取得されたドキュメントには現れません。

参照： テキスト問合せ式の詳細は、『Oracle8i *interMedia Text* リファレンス』を参照してください。

interMedia Text の例 4: テキスト問合せ式の使用

この例は、SELECT 文でテキスト問合せ式を設定および使用方法を示します。

- [シソーラスの作成](#)
- [christie 表の作成](#)
- [ctx_mutable 表の作成](#)

シソーラスの作成

-- ctxsys として実行します。

```
begin
  Ctx_Thes.Drop_Thesaurus ( 'default' );
exception
  when others then
    /* エラー発生条件
       DRG-11701: シソーラス default は存在しません。 */
    if instr ( SQLERRM, 'DRG-11701' ) != 0
    then
      null;
```

```
        else
            raise_application_error ( -20000, SQLERRM );
        end if;
    end;
/
begin
    Ctx_Thes.Create_Thesaurus (
        name      => 'default',
        casesens => false );

    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'crime' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'murder',
        rel       => 'NT',
        relname   => 'crime' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'death',
        rel       => 'RT',
        relname   => 'murder' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'kill',
        rel       => 'RT',
        relname   => 'murder' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'strangling',
        rel       => 'NT',
        relname   => 'murder' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => 'thirteen' );
    Ctx_Thes.Create_Phrase (
        tname     => 'default',
        phrase    => '13',
        rel       => 'SYN',
        relname   => 'thirteen' );
end;
/
```

christie 表の作成

```
Set Define Off
begin
  execute immediate
    'drop table christie';
exception
  when others then
    /* エラーの条件
       ORA-00942: 表またはビューが存在しません。 */
    if instr ( SQLERRM, 'ORA-00942' ) != 0
    then
      null;
    else
      raise_application_error ( -20000, SQLERRM );
    end if;
end;
/

create table christie ( id number, title varchar2(700) );
insert into christie ( title ) values ( '<T>Thirteen At Dinner</T> - <A>Agatha
Christie</A>' );
insert into christie ( title ) values ( '<T>The 4:50 from Paddington</T> - <A>Agatha
Christie</A>' );
insert into christie ( title ) values ( '<T>Blue Geranium</T> - <A>Agatha
Christie</A>' );
insert into christie ( title ) values ( '<T>The fiction of Agatha Christie</T> -
<A>John Smith</A>' );
insert into christie ( title ) values ( '<T>Caribbean with quite a few intervening
words between it and Mystery</T> - <A>Agatha Christie</A>' );
commit;
update christie set id = rownum;
commit;
alter table christie
  add constraint christie_pk primary key ( id )
  using index;
create unique index christie_title on christie ( title );
drop index christie_title;

begin
  Ctx_Ddl.Drop_Preference ( 'my_basic_lexer' );
exception
```

```

when others then
    /* エラーの条件
       プリファレンスが存在しません。*/
    if instr ( SQLERRM, 'DRG-10700' ) != 0
    then
        null;
    else
        raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
begin
    Ctx_Ddl.Drop_Section_Group ( 'my_basic_section_group' );
exception
when others then
    /* エラーの条件
       セクション・グループが存在しません。*/
    if instr ( SQLERRM, 'DRG-12203' ) != 0
    then
        null;
    else
        raise_application_error ( -20000, SQLERRM );
    end if;
end;
/
begin
    Ctx_Ddl.Create_Preference ( 'my_basic_lexer', 'basic_lexer'
                                );
    Ctx_Ddl.Set_Attribute      ( 'my_basic_lexer', 'index_themes', 'false' );
    Ctx_Ddl.Set_Attribute      ( 'my_basic_lexer', 'index_text',   'true'  );
    Ctx_Ddl.Create_Section_Group
    (
        group_name => 'my_basic_section_group',
        group_type => 'basic_section_group'
    );
    Ctx_Ddl.Add_Field_Section
    (
        group_name  => 'my_basic_section_group',
        section_name => 'title',
        tag         => 't',
        visible      => true
    );
    Ctx_Ddl.Add_Field_Section
    (
        group_name  => 'my_basic_section_group',
        section_name => 'author',

```

```
        tag          => 'a',
        visible      => true
    );
end;
/
create index christie_title on christie ( title )
    indextype is ctxsys.context
    parameters ( 'lexer my_basic_lexer section group my_basic_section_group' );
begin
    Ctx_Ddl.Drop_Preference ( 'my_basic_lexer' );
    Ctx_Ddl.Drop_Section_Group ( 'my_basic_section_group' );
end;
/
```

ctx_mutab 表の作成

```
Set Define Off
begin
    execute immediate
        'drop table ctx_mutab';
exception
    when others then
        /* エラーの条件
           ORA-00942: 表またはビューが存在しません。 */
        if instr ( SQLERRM, 'ORA-00942' ) != 0
        then
            null;
        else
            raise_application_error ( -20000, SQLERRM );
        end if;
end;
/
create table ctx_mutab
(
    query_id number constraint ctx_mutab_pk primary key,
    document clob
);

begin
    execute immediate
        'drop sequence ctx_mutab_seq';
exception
    when others then
        /* エラーの条件
           ORA-02289: 順序が存在しません。 */
```

```
if instr ( SQLERRM, 'ORA-02289' ) != 0
then
    null;
else
    raise_application_error ( -20000, SQLERRM );
end if;
end;
/
create sequence ctx_mutab_seq start with 1;
```

テキスト問合せ式の指定および問合せの実行

これは、テキスト問合せ式を指定して問合せを実行し、ヒットするたびに Ctx_Doc.Markup の出力を表示します。

```
Set Define Off
create or replace procedure Qry_And_Markup
(
    p_qry in varchar2 default null
)
is
    v_query_id    number;
    v_document    clob;
    v_amount      number;
    v_nof_hits    integer := 0;
begin
    if p_qry is not null
    then
        for j in
        (
            select score(0) s, id from christie
            where contains ( title, p_qry, 0 ) > 0
            order by s desc
        )
        loop
            select ctx_mutab_seq.nextval
            into v_query_id
            from dual;
            Ctx_Doc.Markup
            (
                index_name => 'christie_title',
                textkey     => to_char ( j.id ),
                text_query  => p_qry,
                restab      => 'ctx_mutab',
                query_id    => v_query_id,
                starttag    => Show.Start_Tag,
                endtag      => Show.End_Tag
            );
        end loop;
    end if;
end;
```



```
select document
  into v_document
  from ctx_mtab
  where query_id = v_query_id;

v_amount := 4000;

Show.Table_Row
(
  p_cell_1 => to_char ( j.s ),
  p_cell_2 =>
    Dbms_Lob.Substr
    (
      lob_loc    => v_document,
      amount     => v_amount,
      offset     => 1
    )
);
v_nof_hits := v_nof_hits + 1;
end loop;

if v_nof_hits < 1
then
  Show.Table_Row
  (
    p_cell_1 => 'No hits'
  );
end if;
else
  for j in
  (
    select title from christie
  )
  loop
    Show.Table_Row
    (
      p_cell_1 => j.title
    );
  end loop;
end if;
end Qry_And_Markup;
/
Show Errors
```

属性セクションでの問合せ

セクション・グループ・タイプとして XML_SECTION_GROUP または AUTOMATIC_SECTION_GROUP を使用して索引付けする場合、属性セクション内で問合せを実行できます。

次の XML ドキュメントがあると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

title@book セクションを属性セクションのタイトルとして定義できます。これは、CTX_DLL.ADD_ATTR_SECTION プロシージャを使用して行うか、または ALTER INDEX を使用して索引付けした後に動的に行います。

注意： AUTO_SECTION_GROUP を使用して XML ドキュメントを索引付けする場合、システムは自動的に属性セクションを作成し、それに属性名 @ タグ名という形式の名前を付けます。

XML_SECTION_GROUP を使用する場合、CTX_DDL.ADD_ATTR_SECTION を使用して、属性セクションに任意の名前を付けることができます。

属性セクション・タイトル内で Tale を検索するには、次の問合せを発行します。

```
'Tale WITHIN title'
```

属性セクションの問合せに対する制約

次の制約が、属性セクション内での問合せに適用されます。

- 通常の属性テキストの問合せは、WITHIN 句で修飾されていない場合、該当するドキュメントにヒットしません。次の XML ドキュメントがあると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

Tale の問合せは、WITHIN title@book で修飾されていない場合、それ自体では該当するドキュメントにヒットしません。この動作は、参照可能なフラグ・セットを FALSE に設定した場合のフィールド・セクションの動作と同様です。

- 属性セクションは、ネストした WITHIN 問合せでは使用できません。
- 句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

```
Now is the time for all good <word type="noun"> men </word> to come to the aid.
```

通常の good men の問合せを行うと、このドキュメントにヒットし、間に挿入されている属性テキストは無視されます。

WITHIN 問合せは、属性セクションの繰返しを区別できます。この動作は、ゾーン・セクションの動作と同様ですが、フィールド・セクションの動作とは異なります。たとえば、次のドキュメントの場合を考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>  
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

この場合、book がゾーン・セクション、book@author が属性セクションであると想定します。次の問合せについて考えてみます。

```
'(Tale and Bondage) WITHIN book@author'
```

Tale および Bondage は、属性セクション book@author の異なるオカレンスにあるため、この問合せは、ドキュメントにヒットしません。

SECTION GROUPS の問合せ

DOCTYPE 間のタグの区別

以前のリリースでは、XML セクション・グループは、異なる DTD のタグを区別できませんでした。たとえば、次の連絡先情報を格納するための DTD があるとして。

```
<!DOCTYPE contact>
<contact>
  <address>506 Blue Pool Road</address>
  <email>dudeman@radical.com</email>
</contact>
```

適切なセクションは、次のようになります。

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','address');
```

これは、同じ表内に異なる種類のドキュメントがない場合は問題ありません。

```
<!DOCTYPE mail>
<mail>
  <address>dudeman@radical.com</address>
</mail>
```

この場合、本来は住所用であったアドレス・セクションが、タグが重複するために、電子メール・アドレスを取得するようになります。

DOCTYPE 制限の指定によるタグの区別

Oracle8i リリース 8.1.5 以上では、DOCTYPE 制限を指定して、DOCTYPE 間のタグを区別できます。次のとおり、タグの前のカッコ内にドキュメント・タイプを指定します。

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','(contact)address');
ctx_ddl.add_field_section('mysg','email','(mail)address');
```

この場合、XML セクション・グループはアドレス・タグを認識し、そのタグを、ドキュメント・タイプが **contact** である場合は住所セクションとして、ドキュメント・タイプが **mail** である場合は電子メール・セクションとして索引付けします。

セクション・グループ内での DOCTYPE 制限タグおよび未制限タグの混在

次のとおり、1 つのセクション・グループ内に DOCTYPE 制限タグおよび未制限タグが両方あると想定します。

```
ctx_ddl.add_field_section('mysg','sec1','(type1)tag1');  
ctx_ddl.add_field_section('mysg','sec2','tag1');
```

この場合、制限タグは該当する DOCTYPE の場合に適用され、未制限タグはそれ以外のすべてのドキュメント・タイプの場合に適用されます。

これは、問合せには影響しません。問合せは、タグではなく、セクション名で行われるため、電子メール・アドレスの問合せは、次のとおり行われます。

```
radical WITHIN email
```

この場合、2 つの異なる種類のタグを同じセクション名にマップしているため、どちらのタグが電子メール・アドレスを表すために使用されているかに関係なく、ドキュメントが検索されます。

interMedia Text を使用した問合せアプリケーションの作成手順

interMedia Text を使用して問合せアプリケーションを作成するには、まず、次の索引付け手順に従います。これらの手順については、5-7 ページの「[interMedia Text 索引](#)」を参照してください。

1. ユーザーに `ctxapp` 権限を付与します。
2. パラメータ化テキスト索引を作成します。
3. ドキュメントを索引付けします。
4. 問合せに一致するドキュメントを表示します。

次に、問合せアプリケーションを作成します。これを行うには、次の手順に従います。

1. `CTX_DDL.create_preference` プロシージャを使用して、プリファレンスを作成します。
「[プリファレンスの作成](#)」を参照してください。
2. `CTX_DDL.Add_Attr_Section` などを使用して、プリファレンスの属性を設定します。
「[プリファレンスの属性の設定](#)」を参照してください。
3. 問合せ構文を作成します。

セクション・グループ・タイプとして `XML_SECTION_GROUP` または `AUTOMATIC_SECTION_GROUP` を使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

注意：

- ドキュメント内のすべてのものを検索できるわけではありません。まず、`.....add_.....section` を使用して何を検索できるかを確認してください。
 - 索引に追加したセクションが多いほど、検索に時間がかかります。
-
-

interMedia Text では、ネストしたタグの検索がサポートされます。

CTX_OBJECTS 表および CTX_OBJECT_ATTRIBUTES ビューの使用

`CTX_OBJECT_ATTRIBUTES` ビューは、各オブジェクトのプリファレンスに割り当てることができる属性を表示します。すべてのユーザーは、このビューを問い合わせることができます。

次の `DESCRIBE` コマンドを使用して、`CTX_OBJECTS` および `CTX_OBJECT_ATTRIBUTE` ビューの構造を確認してください。この章では、XML ドキュメントの問合せのみを目的としているため、`XML_SECTION_GROUP` および `AUTO_SECTION_GROUP` に注目します。

```
Describe ctx_objects
  SELECT obj_class, obj_name FROM ctx_objects
  ORDRR BY obj_class, obj_name;
```

結果は次のとおりです。

```
...
SECTION_GROUP          AUTO_SECTION_GROUP    <<==
SECTION_GROUP          BASIC_SECTION_GROUP
SECTION_GROUP          HTML_SECTION_GROUP
SECTION_GROUP          NEWS_SECTION_GROUP
SECTION_GROUP          NULL_SECTION_GROUP
SECTION_GROUP          XML_SECTION_GROUP    <<==
...
```

```
Describe ctx_object_attributes
SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'XML_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-----
ATTR
FIELD
SPECIAL
ZONE
```

```
SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'AUTO_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-----
STOP
```

プリファレンスの作成

まず、プリファレンスを作成する必要があります。これを行うには、CTX_DDL.Create_Preference プロシージャを使用します。

次に例を示します。

CTX_DDL.Create_Preference

```
CTX_DDL.Create_Preference (  
  preference_name => 'books' /* 任意の名前 */  
  object_name      => 'XML_SECTION_GROUP' /* XML_SECTION_GROUP または  
  AUTO_SECTION_GROUP のどちらか */);
```

このプリファレンスを削除するには、次の構文を使用します。

```
CTX_DDL.Drop_Preference (  
  preference_name => 'books');
```

プリファレンスの属性の設定

XML_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section
- Add_Special_Section

AUTO_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section

対応する CTX_DDL.drop セクションおよび CTX_DDL.remove セクション構文があります。

CTX_DDL.add_zone_section の使用

次に、CTX_DDL.add_zone_section の構文を示します。

```
CTX_DDL.Add_Zone_Section (
  group_name      => 'my_section_group' /* 前に指定した名前 */
  section_name    => 'author' /* このセクションの名前 */
  tag             => 'my_tag' /* XMLでの表現 */ );
```

ここで 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

add_zone_section についてのガイドライン

次に、add_zone_section についてのガイドラインを示します。

- 検索する必要がある XML ドキュメント内の各タグごとに CTX_DDL.Add_Zone_Section をコールします。

CTX_DDL.Add_Attr_Section の使用

次に、CTX_DDL.add_attr_section の構文を示します。

```
CTX_DDL.Add_Attr_Section ( /* 属性セクションを記述するたびにこれをコールします。 */
  group_name      => 'my_section_group' /* 前に指定した名前 */
  section_name    => 'author' /* このセクションの名前 */
  tag             => 'my_tag' /* XMLでの表現 */ );
```

ここで 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

add_attr_section についてのガイドライン

次に、add_attr_section についてのガイドラインを示します。

- 次の meta_data 属性 author について考えてみます。
 <meta_data author = "John Smith" title="How to get to Mars">

索引に追加したセクションが多いほど、検索に時間がかかります。

add_attr_section は、XML セクション・グループに属性セクションを追加します。このプロシージャは、XML ドキュメント内の属性をセクションとして定義する場合に有効です。これによって、WITHIN 演算子を使用して、XML 属性テキストを検索できるようになります。

次に、section_name についてのガイドラインを示します。

- 属性テキストの WITHIN 問合せに使用される名前です。
- コロン (:) またはドット (.) 文字を含むことができません。

- group_name 内で一意である必要があります。
- 大文字 / 小文字を区別しません。
- 64 バイト以下である必要があります。

このタグは、属性の名前をタグ名 @ 属性名形式で指定します。この場合、大文字 / 小文字が区別されます。

注意： add_attr_section プロシージャでは、問合せ時に、様々なタグを同じセクション名で表すことができます。したがって、キーワード WITHIN 検索の引数として使用される名前が、実際の XML タグ名と異なる場合があります。これは、問合せ時に、様々なタグを同じ名前にマップできることを意味します。この機能によって、問合せの検索性が向上します。

CTX_DDL.add_field_section の使用

次に、CTX_DDL.add_field_section の構文を示します。

```
CTX_DDL.Add_Field_Section (  
  group_name      => 'my_section_group' /* 前に指定した名前 */  
  section_name    => 'qq' /* このセクションの名前 */  
  tag             => 'my_tag' /* XML での表現 */ );  
  visible        => TRUE or FALSE );
```

add_field_section についてのガイドライン

次に、add_field_section についてのガイドラインを示します。

- add_field_section および add_zone_section 属性は、パフォーマンスの点で異なります。
- ドキュメント内では、タグを 2 回以上繰り返すことができますが、title などのいくつかのタグは、1 回しか現れません。DTD (または XML Schema) で、タグが現れる回数を定義します。
- **VISIBLE 属性：**これは、add_field_section では使用可能ですが、add_zone_section では使用できません。VISIBLE が TRUE に設定されているときに「... CONTAINS cat...」などの問合せを行うと、cat がタイトルまたは段落内に現れる場合、正しい検索が行われません。

再度、「... CONTAINS cat...」という問合せについて考えてみます。VISIBLE=TRUE に設定すると、ヒットしない場合があります。VISIBLE=FALSE の場合は、索引が小規模になるなど、機能の一部が失われますが、VISIBLE =TRUE の場合と比較して、パフォーマンスは向上します。

- add_zone_section を使用して索引を設定する場合
- add_field_section を使用して索引を設定する場合

注意： 1つのタグが2回以上現れる可能性がある場合、索引を構成することがさらに困難になります。

- XML ドキュメント内のタグが1回しか現れない場合は、単一の `add_field_section` プロシージャを使用してください。たとえば、「... CONTAINS cat and dog WITHIN title.....」などの場合です。
 - XML ドキュメント内のタグが2回以上現れる場合は、`add_zone_section` プロシージャを使用してください。たとえば、「... CONTAINS cat and dog WITHIN paragraph.....」などの場合です。多くの場合、タグは2回以上現れます。
-

属性セクションとフィールド・セクションの違い

属性セクションとフィールド・セクションは、次の点で異なります。

- 属性テキストは、表示されていないとみなされます。したがって、次の問合せは、フィールド・セクションと同様、ドキュメントを検索しません。

```
WHERE CONTAINS (... , '... jeeves',... )...
```

ただし、フィールド・セクションとは異なり、検索内の属性セクションはオカレンスを区別できます。次のドキュメントについて考えてみます。

```
<comment author="jeeves">
  I really like interMedia Text
</comment>
<comment author="bertram">
  Me too
</comment>
```

次の問合せを行うと想定します。

```
WHERE CONTAINS (... , '(cryil and bertram) WITHIN author', ... )...
```

「jeeves」および「bertram」が同じ属性テキスト内に現れないため、この問合せは、ドキュメントを検索しません。

- 複数のタグ名 @ 属性名を単一のセクション名にマップできますが、属性セクション名を、ゾーンまたはフィールド・セクション名とオーバーラップさせることはできません。属性セクションは、デフォルト値をサポートしません。次のドキュメントについて考えてみます。

```
<!DOCTYPE foo [
  <!ELEMENT foo (bar)>
  <!ELEMENT bar (#PCDATA)>
<!ATTLIST bar
  rev CDATA "8i">
```

```
]>
<foo>
  <bar>whatever</bar>
</foo>
```

また、次の属性セクションについて考えてみます。

```
ctx_ddl.add_attr_section('mysg', 'barrev', 'bar@rev');
```

問合せを実行します。

XML セマンティクスでは、bar 要素には rev 属性に対するデフォルト値がありますが、「8i within barrev」という問合せは、このドキュメントにヒットしません。

CTX_DDL.add_special_section の使用

次に、CTX_DDL.add_special_section の構文を示します。

```
CTX_DDL.Add_Special_Section (
  group_name      => 'my_section_group' /* 前に指定した名前 */
  section_name    => 'qq' /* このセクションの名前 */ );
```

add_special_section についてのガイドライン

次に、add_special_section についてのガイドラインを示します。

タグ・オプションが存在しません。開始タグおよび終了タグで定義されていないセクションは、暗黙的に定義されています。ほとんどタグ付けされていない文や段落などで構成されているドキュメントで、その文および段落を検索する必要がある場合は、このセクションを使用します。これらは、明示的に定義されています。

たとえば、問合せが「... CONTAINS cat and dog WITHIN sentence...」などの場合です。

CtX_DDL.Add_Stop_Section の使用

```
CtX_DDL.Add_Stop_Section (
  group_name      => 'my_section_group' /* 前に指定した名前 */
  section_name    => 'qq' /* このセクションの名前 */ );
```

問合せ構文の作成

問合せ文で CONTAINS 演算子を使用する方法については、5-4 ページの「[CONTAINS 演算子を使用した問合せ](#)」を参照してください。

属性セクション内での問合せ

セクション・グループ・タイプとして XML_SECTION_GROUP または AUTO_SECTION_GROUP を使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

次の XML ドキュメントがあると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

title@book セクションを属性セクションのタイトルとして定義できます。これは、CTX_DDL.Add_Attr_Section プロシージャを使用して行うか、または ALTER INDEX を使用して索引付けした後に動的に行います。

注意： AUTO_SECTION_GROUP を使用して XML ドキュメントを索引付けする場合、システムは、自動的に属性セクションを作成し、それに属性名 @ タグ名という形式の名前を付けます。

XML_SECTION_GROUP を使用する場合、CTX_DDL.Add_Attr_Section を使用して、属性セクションに任意の名前を付けることができます。

属性セクション・タイトル内で Tale を検索するには、次の問合せを発行します。

```
WHERE CONTAINS (...,'Tale WITHIN title', ...)
```

XML_SECTION_GROUP および add_attr_section を使用した問合せの支援

次のとおり、TITLE 属性を指定して BOOK タグを定義する XML ファイルについて考えてみます。

```
<BOOK TITLE="Tale of Two Cities">
It was the best of times. </BOOK>
<Author="Charles Dickens">
Born in England in the town, Stratford_Upon_Avon </Author>
```

次に、CTX_DDL.Add_Attr_Section 構文を示します。

```
CTX_DDL.Add_Attr_Section ( group_name, section_name, tag );
```

TITLE 属性を属性セクションとして定義するには、次のとおり、XML_SECTION_GROUP を作成し、属性セクションを定義します。

```
ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
ctx_ddl.add_attr_section('myxmlgroup', 'authors', 'author');
end;
```

TITLE 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合わせることができます。

```
... WHERE CONTAINS (...,'Cities WITHIN booktitle', ....)...
```

AUTHOR 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合わせることができます。

```
... WHERE 'England WITHIN authors'
```

interMedia Text の例 4: ドキュメントの問合せ

この例では、次の操作を行います。

1. res_xml 表の作成および移入
2. 索引 section_group およびプリファレンスの作成
3. プリファレンスのパラメータ化
4. res_xml に対するテスト問合せの実行

```
drop table res_xml;
```

```
CREATE TABLE res_xml (
  pk          NUMBER PRIMARY KEY ,
  text       CLOB
) ;
```

```
insert into res_xml values(111,
  'ENTITY chap8 "Chapter 8, <q>Keeping it Tidy: the XML Rule Book </q>"> this is the
document section');
commit;
```

```
---
```

```
--- res_xml に索引を作成するスクリプトです。
```

```
---
```

```
--- cleanup, in case we have run this before
DROP INDEX res_index ;
EXEC CTX_DDL.DROP_SECTION_GROUP ( 'res_sections' ) ;
```

```
--- セクション・グループを作成します。
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP ( 'res_sections', 'XML_SECTION_GROUP' );
  CTX_DDL.ADD_FIELD_SECTION ( 'res_sections', 'chap8', '<q>' );
END ;
/

begin
  ctx_ddl.create_preference
  (
    preference_name => 'my_basic_lexer',
    object_name      => 'basic_lexer'
  );
  ctx_ddl.set_attribute
  (
    preference_name => 'my_basic_lexer',
    attribute_name  => 'index_text',
    attribute_value => 'true'
  );
  ctx_ddl.set_attribute
  (
    preference_name => 'my_basic_lexer',
    attribute_name  => 'index_themes',
    attribute_value => 'false');
end;
/

CREATE INDEX res_index
  ON res_xml(text)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'lexer my_basic_lexer SECTION GROUP res_sections' );

前述の索引を次のようなテスト問合せでテストします。

SELECT pk FROM res_xml WHERE CONTAINS( text, 'keeping WITHIN chap8' )>0 ;
```

interMedia Text の例 5: 索引の作成およびテキスト問合せの実行

この例で使用する explain_ex 表の作成

```
drop table explain_ex;

create table explain_ex
(
    id          number primary key,
    text        varchar(2000)
);

insert into explain_ex ( id, text )
values ( 1, 'thinks thinking thought go going goes gone went' || chr(10) ||
        'oracle orackle oricle dog cat bird' || chr(10) ||
        'President Clinton' );
insert into explain_ex ( id, text )
values ( 2, 'Last summer I went to New England' || chr(10) ||
        'I hiked a lot.' || chr(10) ||
        'I camped a bit.' );

commit;
```

テーマ索引の作成

```
begin
    Ctx_Ddl.Drop_Preference ('my_lexer');
end;
/
begin
    Ctx_Ddl.Create_Preference ( 'my_lexer', 'basic_lexer' );
    Ctx_Ddl.Set_Attribute      ( 'my_lexer', 'index_text', 'true' );

    /* Experiment with 'index_themes' = 'false' */
    Ctx_Ddl.Set_Attribute      ( 'my_lexer', 'index_themes', 'true' );
end;
/

begin
    Ctx_Ddl.Drop_Stoplist ( 'my_stoplist' );
end;
/

begin
    Ctx_Ddl.Create_Stoplist ( 'my_stoplist' );
    Ctx_Ddl.Add_Stopword
    (
        stoplist_name => 'my_stoplist',
```



```

        stopword      => 'because'
    );
    Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'and' );
    Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'in' );
    Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'to' );
    Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'a' );
    Ctx_Ddl.Add_Stopword ( 'my_stoplist', 'I' );
end;
/

drop index explain_ex text;
select err_text from ctx_user_index_errors;
create index explain_ex_text on explain_ex ( text )
    indextype is ctxsys.context
    parameters ( 'lexer my_lexer stoplist ctxsys.empty_stoplist' );
select err_text from ctx_user_index_errors;

begin
    Ctx_Ddl.Drop_Preference ( 'my_lexer' );
    Ctx_Ddl.Drop_Stoplist ( 'my_stoplist' );
end;
/

```

テキスト問合せ式に ABOUT を使用するテキスト問合せ

```

Set Define Off
select text
    from explain_ex
    WHERE CONTAINS ( text,
        '( $( think & go ) , ?oracle ) & ( dog , ( cat & bird ) ) & about (mammal
                                during Bill Clinton)' ) > 0;

select text
    from explain_ex
    WHERE CONTAINS ( text, 'about ( camping and hiking in new england )' ) > 0;

```

ドキュメント・タイプが区別される XML ドキュメントでのセクションの作成

様々なドキュメント・タイプで宣言されている <book> タグを含む XML ドキュメント・セットについて考えてみます。ドキュメント・タイプごとに個別の book セクションを作成する必要があります。次のとおり、mydocname が XML ドキュメント・タイプ（ルート要素）として宣言されているとします。

```
<!DOCTYPE mydocname ... [...
```

mydocname 内で、要素 <book> が宣言されます。このタグ用に、次のとおり、タグのドキュメント・タイプで区別される mybooksec という名前のセクションを作成できます。

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');
end;
```

注意：

- Oracle8i では、セクション・グループの作成時に指定する group_type パラメータから、終了タグがどのように見えるかを認識しています。指定する開始タグは、セクション・グループ内で一意にしてください。
 - セクション名は、タグ間で一意である必要はありません。検索に対し詳細を透過的にして、同じセクション名を複数のタグに割り当てることができます。
-
-

セクションの繰返し

ゾーン・セクションは、繰り返すことができます。各オカレンスは、個別のセクションとして処理されます。たとえば、<H1> がヘッダー・セクションを示す場合、次のとおり、同じドキュメント内で繰り返すことができます。

```
<H1> The Brown Fox </H1>
<H1> The Gray Wolf </H1>
```

これらのゾーン・セクションに Heading という名前が付けられているとします。

次の問合せを行うとします。

```
WHERE CONTAINS (... , 'Brown WITHIN Heading', ...)...
```

この問合せは、このドキュメントを返します。

次の問合せを行うとします。

```
WHERE CONTAINS (...,' (Brown and Gray) WITHIN Heading',...)
```

この問合せは、このドキュメントを返しません。

セクションのオーバーラップ

ゾーン・セクションは相互にオーバーラップできます。たとえば、`` および `<I>` が 2 つの異なるゾーン・セクションを示す場合、次のとおり、これらはドキュメント内でオーバーラップできます。

```
plain <B> bold <I> bold and italic </B> only italic </I> plain
```

セクションのネスト

ゾーン・セクションは、次のとおり、ネストすることができます。

```
<TD>
  <TABLE>
    <TD>nested cell</TD>
  </TABLE>
</TD>
```

`WITHIN` 演算子を使用すると、セクション内のセクションのテキストを検索する問合せを作成できます。

ネストしたセクション問合せの例

たとえば、`BOOK1`、`BOOK2` および `AUTHOR` ゾーン・セクションが、次のとおり、ドキュメント `doc1` および `doc2` 内に現れるとします。

`doc1` は次のとおりです。

```
<book1><author>Scott Tiger</author> This is a cool book to read.</book1>
```

`doc2` は次のとおりです。

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

次のネストした問合せについて考えてみます。

```
'Scott WITHIN author WITHIN book1'
```

この問合せは、`doc1` のみを返します。

問合せ結果の表示

テキスト問合せアプリケーションを使用すると、問合せによって戻されたドキュメントを表示できます。通常、ヒットリストからドキュメントを選択し、アプリケーションがそのドキュメントを特定の形式で表示します。

interMedia Text を使用すると、ドキュメントを様々な方法でレンダリングできます。たとえば、問合せ用語をハイライトさせることができます。ハイライト表示できる問合せ用語は、英語の場合、ワード問合せのワード、または ABOUT 問合せのテーマです。このレンダリングを行うには、CTX_DOC.HIGHLIGHT または CTX_DOC.MARKUP プロシージャを使用します。

また、CTX_DOC.THEMES という PL/SQL パッケージを使用しても、ドキュメントからテーマ情報を取得できます。この他にも、問合せ結果を表示するためのいくつかの CTX_DOC プロシージャがあります。

参照： CTX_DOC PL/SQL パッケージの詳細は、『Oracle8i *interMedia Text* リファレンス』を参照してください。

FAQ: *interMedia* Text

interMedia Text を使用した XML データの挿入および検索

質問

interMedia は XML の階層構造を認識しませんが、次のような検索はできますか？

```
<report>
<day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

cause が hurricane であった LOB を検索したいのですが、可能ですか？

回答

interMedia の今回のリリースを使用すると、このレベルの検索は可能です。現在、ドキュメントを分割するには、Oracle XML Parser を XSLT とともに使用して、XML を DDL に変換するスタイルシートを作成する必要があります。iFS を使用すると、さらに高レベルのインタフェースが提供されます。

もう 1 つの方法は、JDBC プログラムを使用してドキュメントまたはドキュメント・フラグメントのテキストを CLOB 列または LONG 列に挿入し、索引を設定してから、CONTAINS() 演算子を使用して検索を行う方法です。

interMedia Text: 属性の処理

質問

現在、*interMedia* Text には、セクション・グループの内容に基づいて索引を作成するためのオプションがあります。ただし、ほとんどの XML 要素は、型の要素です。このため、検索のためのオプションは、属性値のみとなります。属性値の索引を作成する方法はありますか？

回答

リリース 8.1.6 以上では、属性を索引付けできます。

CTXSYS/CTXSYS の ID およびパスワード

質問

ユーザー / パスワードの CTXSYS/CTXSYS を介してデータベースにアクセスできません。Web サーバーのルート・ディレクトリから SQL PLUS を実行しても、SQL PLUS に通常の方法でアクセスしても、このデータベースにアクセスできません。SCOTT/TIGER または SYSTEM/MANAGER には、このどちらかの方法でアクセスできます。何か、このユーザー / 権限などを追加するために必要な手順が抜けていますか？手順が抜けている場合、その説明を参照できる場所を教えてください。受信したエラー・メッセージは次のとおりです。

```
ERROR:OCA-30017: error logging on to non-Oracle database [POL-5246] User does not exist.
```

現在、Oracle8i Lite および Windows NT 4 SP 4 を使用しています。

回答

Oracle8i Lite は *interMedia Text* をサポートしません。CTXSYS/CTXSYS は、*interMedia Text* スキーマ所有者用のデフォルトのユーザー名 / パスワードです。通常の Oracle8i データベースに対して実行するように設計されているデモを使用する場合、デモを Oracle8i Lite で実行すると、その他の問題が発生する場合があります。

XML ドキュメントの問合せ

質問

完全な状態の XML ドキュメントは、Oracle の XML ソリューションによって、CLOB または BLOB に格納されると理解しています。

1. CLOB/BLOB に格納された XML ドキュメントは、表スキーマのように問い合わせることができますか？次に例を示します。

```
[XML document stored in BLOB]...<name id="1111"><first>lee</first>
<second>jumee</second></name>...
```

この場合、XML ドキュメントの要素、属性および構造によって、値 (lee、jumee) を問い合わせることができますか？

2. 挿入 / 更新 / 削除できない要素または属性がある場合、ドキュメント全体を更新する必要がありますか？またドキュメントは、表スキーマと同様に、挿入 / 更新 / 削除できますか？

3. ロックに関する質問ですが、誰かが BLOB/CLOB に格納された XML ドキュメントを管理している場合、他の人は同じ XML ドキュメントにアクセスできないと聞きました。これは本当ですか？

回答

1. *interMedia Text* を使用すると、次の問合せによって、このドキュメントを検索できます。

```
lee within first or this:jumee within second or this:1111 within name@id
```

これらを次のとおり組み合わせることもできます。

```
lee within first and jumee within second or this:(lee within first) within name.
```

2. *interMedia Text* は、CLOB/BLOB を索引付けしますが、XML を明確に理解しているわけではないので、実際には個々の要素を変更することはできません。ドキュメント全体を編集する必要があります。
3. 他の CLOB と同様、誰かが CLOB に書き込んでいる場合、その CLOB はロックされ、他の人はそれに書き込むことができません。他のユーザーは、その CLOB を読み込むことはできますが、書き込むことはできません。これは、LOB の基本動作です。

別の方法として、XML ドキュメントを分解し、情報をリレーショナル・フィールドに格納する方法があります。これによって、個々の要素の変更、要素レベルでの同時アクセスなどが可能になります。この場合、USER_DATASTORE を使用すると、PL/SQL を使用して再度ドキュメントを XML に構成し、テキストを索引付けすることができます。これによって、テキストを XML として検索できますが、データはリレーショナル・データとして管理されます。

interMedia Text および Oracle8i

質問

interMedia Text は Oracle8i に含まれていますか？パッケージ名を教えてください。そのパッケージには、データベースに XML ドキュメントを挿入する機能および XML ドキュメントを検索する機能がありますか？

回答

現在、Context Cartridge は *interMedia Text* といい、Oracle8i *interMedia* オプションの一部です。*interMedia Text* には、データベースに XML ドキュメントを挿入する機能はありません。これは、XML ドキュメントを検索するのみです。

interMedia を使用した XML の索引付け

質問

interMediaText を使用して、次のような XML を索引付けできますか？

2/7/1968

また、次の問合せは処理できますか？

select name from person where hair.color = "BROWN" (髪の毛が茶色の人)

回答

構造条件に基づく検索は、現在 interMedia Text を介して使用できません。属性検索はリリース 8.1.6 からサポートされます。将来、XML Schema が勧告された場合にこれに準拠しないため、データを属性に挿入しないでください。

interMedia Text を使用した CLOB の検索

質問

CLOB 列内の「aorta」および「damage」を含むレコードを検索するには、interMedia パラメータをどのように定義すればよいですか？次に、XML（および DTD）の例を示します。

WellKnownFileName.gif echocardiogram aorta

これは、血管損傷のイメージです。単純（または複雑）な XML の interMedia での実装の例を参照できれば参考になります。

この場合、ZONE または FIELD を設定する必要はありませんか？

回答

XML ドキュメント・フラグメントを CLOB に保存し、それに対し interMedia Text XML 索引を使用可能にすると、次のとおり、CONTAINS() 演算子を使用して SQL 問合せを行うことができます。

次の保険金請求書のドキュメントがあるとします。

77804				
1999-01-01 00:00:00.0		8895		1044
Paul	Astoria			
123 Cherry Lane	SF		CA	94132
1999-01-05 00:00:00.0	7600		JOCK	
It was becace of Faulty Brakes				

この内容をドキュメント・フラグメントとして CLOB に格納すると、次の問合せを行うことができます（他のものはすべてリレーショナル表に格納するとします）。

```
REM Select the SUM of the amounts of
REM all settlement payments approved by "JCOX"
REM for claims whose relates to Brakes.
select sum(n.amount) as TotalApprovedAmount
  from insurance_claim_view v, TABLE(v.settlements) n
 where n.approver = 'JCOX'
 and contains(damageReport, 'Brakes within Cause') >
```

様々な DTD を使用した様々な XML ドキュメントの管理 : CLOB への XML の格納および CLOB での XML の検索 - *interMedia Text*

質問

XML を CLOB に格納し、必要に応じて、DOM または SAX を使用して XML を再解析することを薦められました。これは、現在抱えている問題（ドキュメント管理システムで、様々な DTD を使用して様々な XML ドキュメントを管理する方法）に対する最適なソリューションです。重大な問題は、このドキュメント・リポジトリを検索し、関連情報のある場所を特定することです。

この点で、*interMedia Text* は最適であるといえます。Oracle8i で *interMedia* を使用してこの問題を解決するための例（XML_SECTION_GROUP の定義方法、FIELD に対する ZONE の使用場所など）を参照できれば参考になります。

次に例を示します。

「WellKnownFileName.gif echo cardiogram aorta 」という XML（および DTD）で、CLOB 列内の「aorta」および「damage」を含むレコードを検索するには、*interMedia* パラメータをどのように定義すればよいですか？この XML は、血管損傷のイメージです。

回答

interMedia を使用して構造ベースの XML 検索を行うことはできません。特定の要素内のテキストを検索することはできますが、これより複雑な検索を行うことはできません。また、属性の場合も同様です。DOM Parser を使用して各ドキュメントをロードし、前述の方法で検索することはできますが、高パフォーマンスは望めません。オラクル社では、同様のニーズに対応するプロジェクトを進めています。オラクル社では、重要な検索を行う XML データのビットごとに表内に列を作成し、最初の XML 解析からそれをロードする方法を取っています。ただし、この方法は、任意の要素に対して構造化検索を行う必要がある場合は有効ではありません。

質問

リリース 8.1.6 以上では、属性テキスト内で検索を行うことができます。これは、「dog within book@author」のような検索です。当社では、次のような、属性値の識別による検索を実行しようとしています。

```
dog within book[@author = "Eric"]:
```

```
begin ctx_ddl.create_section_group('mygrp','basic_section_group');
  ctx_ddl.add_field_section('mygrp','keyword','keyword');
  ctx_ddl.add_field_section('mygrp','caption','caption');
end;
create index myidx on mytab(mytxtcolumn) indextype is ctxsys.contextparameters
('section group mygrp');
select * from mytab where contains(mytxtcolumn, 'aorta within keyword')>0;
options:
```

- タグに属性が含まれる場合、または大文字 / 小文字を区別したタグの検出が必要である場合は、基本セクション・グループのかわりに XML セクション・グループを使用します。
- セクションがオーバーラップする場合、またはインスタンスを区別する必要がある場合は、フィールド・セクションのかわりにゾーン・セクションを使用します。たとえば、keywords がフィールド・セクションの場合、「(aorta and echo cardiogram) within keywords」という問合せは、ドキュメントにヒットします。keywords がゾーン・セクションの場合、この問合せは、同じ keywords のインスタンスにないため、ドキュメントにヒットしません。

この例が最適であるとはいえません。この例では、同じレコード内に「aorta」を含む兄弟要素を持つ「damage」を含む要素のインスタンスを検索しているように見えます。「レコード」が意味するものが明確ではありません。

それぞれのレコードがこの例のレコードと同じで、単一の XML を格納した LOB に複数のレコードが存在する場合、*interMedia* を使用して、この検索を実行できますか？

CLOB/ 行ごとに 1 つのレコードのみがある場合、2 つの ConText 要素問合せを AND で組み合わせることによって、これを検索できます。ただし、この検索は、実際に必要な構造より、予想される制約がある XML 検索といえます。

回答

レコードが何を意味するかは明白です。例では、XML 全体が表内の CLOB 列に格納されています。このため、レコードは XML コードを含む表の行を意味しています。

interMedia Text のロール (ORA-01919: ロール CTXSYS は存在しません。)

質問

『Oracle8i interMedia Text 移行ガイド』の「ロールおよびユーザー」には、Oracle8i interMedia Text は、システム管理者およびアプリケーション開発者用に CTXSYS ロールおよび CTXAPP ロールという 2 つのロールを提供することが示されています。

ただし、CTXSYS ロールをユーザーに付与するために GRANT コマンドを発行すると、エラー・メッセージ (ORA-01919: ロール CTXSYS は存在しません。) が戻されます。また、sys.dba_roles ビューを問い合わせた場合、CTXSYS ロールが付与されません。

この問題を解決する方法を教えてください。

回答

interMedia Text がインストールされていない可能性があります。初期データベースを使用されましたか？または新しいデータベースを最初から作成しましたか？後者の場合、インストール中に interMedia Text を選択しましたか？

XML ドキュメントの検索およびゾーンの取得

質問

大規模な XML ファイルを Oracle8i に格納し、それを検索して、特定のタグ付けされた領域を取得する必要があります。大規模な XML ファイルを格納し、それを索引付けして、検索を可能にし、その検索によってタグ付けされたセクションを XML ファイルから取得するための明確な方法がわかりません。interMedia Text を使用して、次のことが実行できます。

- XML ファイルを CLOB フィールドに格納できます。
- ctxsys.context を使用して、XML ファイルを索引付けすることができます。
- <Zone> および <Field> を作成して、
ctx_ddl.add_zone_section(xmlgroup,"dublincore",dc); のように、XML ファイルにタグを表すことができます。
- ゾーンまたはフィールドでテキストを検索できます。たとえば、「Select title from mytable where CONTAINS(textField,"some words WITHIN dublincore")」という問合せが可能です。

テキスト検索によってゾーンまたはフィールドを取得する方法を教えてください。

回答

interMedia Text は、ヒットした項目のみを戻します。次に、CLOB を解析して、セクションを抽出する必要があります。

CLOB への XML ドキュメントの格納 : *interMedia Text* の使用

質問

XML ファイル（現時点でファイル・システム上に存在）をデータベースに格納する必要があります。ドキュメント全体を格納する必要があります。タグによってドキュメントを分割し、情報を別々の表 / フィールドに格納することは望ましくありません。逆に、様々な XML ドキュメントを格納するために使用できる汎用表が必要です。この場合、汎用表は内部で CLOB 型のフィールドに格納されると考えています。使用する XML ファイルは、常に ASCII データを含みます。

interMedia を使用して、このような処理を行うことができますか？そのためには、*interMedia Text* または *interMedia Annotator* のどちらを使用する必要がありますか？Annotator は入手済ですが、XML ドキュメントをデータベースに格納することができません。

XML ドキュメントを CLOB 列に格納しようとしています。基本的に、次のように定義されている表が 1 つあります。

```
CREATE TABLE xml_store_testing
(
    xml_doc_id  NUMBER,
    xml_doc     CLOB )
```

XML ドキュメントを `xml_doc` フィールドに格納する必要があります。

XML ドキュメントの内容を読み込むために、次に示す別の PL/SQL プロシージャを作成しました。XML ドキュメントは、ファイル・システム上で使用可能です。XML ドキュメントには ASCII データのみが含まれ、バイナリ・データはありません。

```
CREATE OR REPLACE PROCEDURE FileExec
(
    p_Directory      IN VARCHAR2,
    p_FileName       IN VARCHAR2)
AS
    v_CLOBLocator    CLOB;
    v_FileLocator     BFILE;
BEGIN
    SELECT xml_doc
    INTO    v_CLOBLocator
    FROM    xml_store_testing
    WHERE   xml_doc_id = 1
    FOR     UPDATE;
    v_FileLocator := BFILENAME(p_Directory, p_FileName);
    DBMS_LOB.FILEOPEN(v_FileLocator, DBMS_LOB.FILE_READONLY);
    dbms_output.put_line(to_char(DBMS_LOB.GETLENGTH(v_FileLocator)));
    DBMS_LOB.LOADFROMFILE(v_CLOBLocator, v_FileLocator,
        DBMS_LOB.GETLENGTH(v_FileLocator));
    DBMS_LOB.FILECLOSE(v_FileLocator);
END FileExec;
```

回答

XML ドキュメントを CLOB 列に挿入してから、XML セクション・グループを使用して、*interMedia Text* 索引をそれに追加してください。

質問

このプロシージャを実行すると、正常に実行されました。ただし、表から選択した場合、CLOB フィールド内の表に不明な文字が現れました。これは、オペレーティング・システム (XML ファイルを格納) とデータベース (CLOB データを格納) のキャラクタ・セットの違いが原因ですか？

回答

キャラクタ・セットの違いが原因です。キャラクタ・セットが異なる場合は、UTL_RAW.CONVERT を介してデータを渡し、キャラクタ・セットを変換してから CLOB に書き込む必要があります。

データベースへの XML ドキュメントのロードおよび *interMedia Text* を使用した検索

質問

XML ドキュメントをデータベースに挿入する方法を教えてください。

特に、XML ドキュメントを、現在の状態のまま、表内の CLOB データ型の列に挿入する必要があります。

回答

Oracle XML SQL Utility for Java は、XML データをロードするために使用可能なコマンドライン・ユーティリティを提供します。

XML ドキュメントは、すべてのテキスト・ファイルと同様に挿入できます。CLOB への挿入も、他のテキスト・ファイルの場合と変わりません。

質問

Oracle *interMedia Text* を使用して、CLOB に格納された XML を索引付けおよび検索できますか？これを行うための方法を教えてください。

回答

interMedia Text の前のリリースでは、タグ・ベースの検索のみが可能でした。Oracle8i の現行リリースでは、XML の構造ベースおよび属性ベースの検索が可能になっています。索引の作成方法および Oracle8i *interMedia* での SQL の使用方法に関するドキュメントを参照してください。

参照：『Oracle8i *interMedia Text* リファレンス』を参照してください。

WITHIN 演算子を使用した XML の検索

質問

次の XML があります。

```
<person>
  <name>efrat</name>
  <childrens>
    <child>
      <id>1</id>
      <name>keren</name>
    </child>
  </childrens>
</person>
```

keren という名前の子供を持ち、自分の名前が keren ではない人を検索する方法を教えてください。ネスト可能で、自分自身を含むことができる `add_zone_section` を使用して、すべてのタグを定義済であると想定します。

回答

タグのかわりに属性として ID を作成することをお勧めします。

XML および *interMedia Text*

質問

XML を *interMedia* で使用する方法を適切に示した例はどこで参照できますか？

回答

『Oracle8i *interMedia Text* リファレンス』を参照してください。

***interMedia* Text および XML: Add_field_section**

質問

XML および *interMedia* Text に関する質問です。XML ドキュメントを *interMedia* Text に送り込み、タグを認識させる方法がありますか？または、XML ドキュメント内の各タグごとに `add_field_section` コマンドを使用する必要がありますか？使用する XML ドキュメントには、何百ものタグが含まれています。これを簡単に行う方法がありますか？

回答

どのリリースのデータベースをご使用ですか？リリース 8.1.5 では、各タグごとに `add_field_section` コマンドを使用する必要がありますが、8.1.6 ではその必要はありません。リリース 8.1.6 では、`AUTO_SECTION_GROUP` を使用できます。

***interMedia* および XML のサポート**

質問

次の単純な作業を行う場合の例を教えてください。
XML ドキュメントを Oracle8i に送り込み、タグを使用して内容ベースの検索を行う必要があります。使用する XML ドキュメントには 100 を超えるタグが含まれています。各タグに対して `ADD_FIELD_SECTION` を使用する必要がありますか？必要ない場合、これについての説明があるドキュメントを教えてください。

回答

XSQL Servlet には、SQL スクリプトの完全な例（*interMedia* の使用部分は単純）が付属しています。このスクリプトは、オブジェクト型から複雑な XML データグラムを作成し、保険請求書型に格納された XML ドキュメント・フラグメントに対して *interMedia* Text 索引を作成します。

XSQL Servlet をダウンロードし、ファイル `./xsql/demo/insclaim.sql` を参照すると、ファイルの下にある *interMedia* の要素を参照できます。リリース 8.1.6 における *interMedia* の新しい主要機能は、`AUTO Sectioner for XML` です。この詳細は、前述の回答に示す URL を参照してください。リリース 8.1.5 では、フィールド・セクションを手動で作成する必要があります。

質問

「Hello World」の例を参照できる場所がありますか？XSQL Servlet のダウンロード・ページで Java スクリプト・エラーが発生しました。

回答

次に、前述のデモ・ファイルの内容を示します。

interMedia Text に関連する部分は、次の行で始まります。

```
ctx_ddl.drop_preference()
```

この例では、保険請求書に、XML ドキュメント・フラグメントである「DamageReport」が含まれています。最後の *interMedia* コードは、この「DamageReport」というドキュメント・フラグメントの <CAUSE> および <MOTIVE> タグに対する XML 検索用索引の設定方法を示しています。

```
set scan offset echo onset termout onREMREM $Author: smuench $REM $Date: 1999/11/27
14:48:10 $REM $Source: C:\\cvsroot/xsql/src/demo/insclaim.sql,v $REM $Revision: 1.3
$REMDrop synonym claim;drop table settlement_payments;drop view insurance_claim_
view;drop table insurance_claim;drop view policy_view;drop table policy;drop view
policyholder_view;
drop table policyholder;drop type insurance_claim_t;
drop type settlements_t;
drop type payment;
drop type policy_t;
drop type policyholder_t;
drop type address_t;
create type address_t as object( Street varchar2(80), City Varchar2(80), State
VARCHAR2(80),Zip NUMBER );
./create type policyholder_t as object( CustomerId number,
    FirstName varchar2(80),
    LastName varchar2(80),
    HomeAddress address_t);
./create type policy_t as object(
    policyID number, primaryinsured policyholder_t);
./create type payment as object(
    PayDate DATE, Amount NUMBER, Approver VARCHAR2(8));
./create type settlements_t as table of payment;
./create type insurance_claim_t as object (
    claimid number,filed date, claimpolicy policy_t,
    settlements settlements_t, damageReport varchar2(4000) /* XML */);
./create table policyholder( CustomerId number,
    FirstName varchar2(80), LastName varchar2(80),
    HomeAddress address_t, constraint policyholder_pk primary key (customerid));
insert into policyholder values ( 1044, 'Paul','Astoria',
    ADDRESS_T('123 Cherry Lane','SF','CA','94132'));
insert into policyholder values ( 1045, 'Martina','Boyle',
```



```

ADDRESS_T('55 Belden Place','SF','CA','94102'));
create or replace force view policyholder_view of policyholder_t
    with object OID
...
...
create or replace force view insurance_claim_view of insurance_claim_t      with
object OID (claimid)
    as select c.claimid,c.filed,
        (SELECT value(pv)
         from policy_view pv
         WHERE pv.policyid = c.claimpolicy),
        CAST(MULTISET(SELECT PAYMENT(sp.paydate,sp.amount,sp.approver)
         as Payment from settlement_payments sp
         WHERE sp.claimid = c.claimid) AS settlements_t),c.damagereport
    from insurance_claim c;commit;
begin ctx_ddl.drop_preference('Demo');
end;
/begin ctx_ddl.create_preference('Demo', 'basic_lexer');
ctx_ddl.set_attribute ('Demo', 'index_themes', '0');
ctx_ddl.set_attribute ('Demo', 'index_text', '1');
ctx_ddl.create_section_group('demo_xml', 'xml_section_group');
ctx_ddl.add_zone_section('demo_xml', 'CAUSE', 'CAUSE');
ctx_ddl.add_zone_section('demo_xml', 'MOTIVE', 'MOTIVE');
end;
/create index
ctx_xml_i on insurance_claim(damagereport) indextype is
ctxsys.contextparameters('LEXER Demo SECTION GROUP demo_xml');
create synonym claim for insurance_claim_view;

```

Oracle8i Lite 4.0.0.2.0: *interMedia Text* の非サポート

質問

次のとおり、データベースを初期化し、デモ・プログラム用の SQL スクリプトを実行することができません。

1. CTXSYS/CTXSYS として接続できません。*interMedia Text* パッケージのソースは何ですか？ CTXSYS ユーザーは、デフォルトのインストールで存在しますか？
2. GRANT QUERY REWRITE TO SCOTT を実行すると、構文エラーが発生します。
3. また、一部のスクリプト (airport.sql など) を実行すると、構文エラーが発生します。他のスクリプト (index.sql など) では、問題ありません。Oracle8i Lite 4.0.0.2.0 をはじめてインストールしています。

回答

1. *interMedia Text* は、Oracle8i の機能です。Oracle8i Lite を使用している場合、この機能は使用できません。
2. QUERY REWRITE 権限は、Oracle8i ではサポートされますが、Oracle8i Lite ではサポートされません。
3. AIRPORT.SQL の最後の文（UPPER（記述）にファンクション索引を作成する文）でエラーが発生します。ファンクション索引は、Oracle8i の機能であり、Oracle8i Lite では使用できません。また、はじめてインストールする場合、SQL スクリプトの実行中に、いくつかのエラー・メッセージが表示されます。スクリプトを確認すると、それが、表を作成する前に表を削除しようとするスクリプトであることがわかります。はじめてのインストールではこれらの表は存在しないため、エラーが発生します。

interMedia コンテキストでの SQL

質問

CLOB に格納した XML ドキュメントがあります。また、`section_group` などを使用して、タグに対する索引を作成済です。1 つのタグは、`<SALARY></SALARY>` です。たとえば、給与が 5000 を超えるすべてのレコードを選択する SQL 文を記述する必要があります。

これを行うには、どうすればよいですか？ `WITHIN` 演算子は使用できません。このタグ内の値を数値として解釈する必要があります。また、これは給与であるため、浮動小数点の場合もあります。

回答

これを *interMedia Text* で行うことはできません。実際には、範囲検索はテキスト操作ではありません。最適のソリューションは、別の Oracle の XML 解析ユーティリティを使用して、給与を `NUMBER` フィールドに変換することです。これによって、テキスト検索には *interMedia Text* を使用し、また、より構造化されたフィールドには通常の SQL 演算子を使用して、同じ結果を取得できます。

XML および *interMedia* Text

質問

XML 形式のすべてのドキュメントを CLOB に格納しています。Oracle には、ドキュメント全体を取得して、その構造を検索するのではなく、1つのフィールドごとに内容を一度に取得（フィールド名を指定して、タグ間のテキストを取得する）するために使用できるユーティリティはありますか？*interMedia* はそれに該当しますか？

回答

interMedia は、セクションの取出しを行いません。これについては、XML SQL Utility を参照してください。

3つの列に対する索引の作成

質問

7～8つの表に基づいてビューを作成し、このビューには、*custordnumber*、*product_dscr*、*qty*、*prdid*、*shipdate*、*ship_status* などの列があります。次の3つの列に対して1つの *interMedia* 索引を作成する必要があります。

- *custordnumber*
- *product_dsc*
- *ship_status*

これらの列に対して1つの索引を作成する方法はありますか？

回答

あります。次の2つのオプションがあります。

1. 索引付け実行中に、*USER_DATASTORE* オブジェクトを使用してその場で連結フィールドを作成します。
2. フィールドを連結し、1つの表の追加の CLOB フィールドに格納します。次に、その CLOB フィールドに対する索引を作成します。Oracle リリース 8.1.6 を使用している場合、連結前に XML タグを各フィールドの前後に置くオプションもあります。これによって、各フィールド内での検索が可能になります。

構造化 / 非構造化データの検索

質問

データを XML ファイルからデータベースに挿入する必要があります。作成済の表を使用して構造化データのみを挿入できると聞きました。これは本当ですか？

現在、法律プロジェクトに取り組んでおり、このプロジェクトでは、構造化データおよび非構造化データを含む法律データを格納し、*interMedia Text* を使用してそのデータを検索する必要があります。

非構造化データも挿入できますか？これを行うためには、カスタム・アプリケーションを作成する必要がありますか？また、構造化部分および非構造化部分を持つデータを格納した場合、*interMedia Text* を使用してそのデータを検索できますか？

非構造化部分を CLOB に格納し、その CLOB にタグが含まれている場合、特定のタグ内にあるデータのみを検索することはできますか？

回答

iFS の使用を検討してください。iFS を使用すると、ドキュメントを分割し、それを複数の表および LOB に格納できます。現在、*interMedia Text* は、タグを使用してデータ検索を行うことはできませんが、XML の階層構造は理解しません。リリース 8.1 以上では、*interMedia Text* に、この機能および名前 / 値のペア属性の検索機能が追加されています。

質問

前述の回答によると、カスタム・アプリケーションを開発しない場合、当面このようなドキュメントの分割は不可能であるということですか？*interMedia* では、XML の階層構造が認識されませんが、次のような検索はできますか？

```
<report>
  <day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

interMedia を使用して索引付けし、*cause* が hurricane である LOB を検索できますか？

回答

今回の *interMedia* のリリースを使用すると、このレベルの検索は可能です。現在、ドキュメントを分割するには、Oracle XML Parser を XSLT とともに使用して、XML を DDL に変換するスタイルシートを作成する必要があります。iFS を使用すると、さらに高レベルのインタフェースが提供されます。

もう 1 つの方法は、JDBC プログラムを使用してドキュメントまたはドキュメント・フラグメントのテキストを CLOB 列または LONG 列に挿入し、索引を設定してから、CONTAINS() 演算子を使用して検索を行う方法です。

XML によるコンテンツのカスタマイズ： Dynamic News アプリケーション

この章の内容は次のとおりです。

- [Dynamic News アプリケーションの概要](#)
- [Dynamic News の主なタスク](#)
- [Dynamic News アプリケーションの概要](#)
- [Dynamic News の SQL の例 1: 項目スキーマ nisetup.sql](#)
- [Dynamic News のサーブレット](#)
- [Dynamic News の動作 : 鳥瞰図](#)
- [静的なページ](#)
- [一部動的なページ](#)
- [動的なページ](#)
- [コンテンツのパersonナライズ](#)
- [エンド・ユーザー設定項目の取得](#)
- [データベースからのニュース項目の取出し](#)
- [ドキュメント構築のためのニュース項目の組合せ](#)
- [表示のカスタマイズ](#)
- [ニュース項目のインポートおよびエクスポート](#)

Dynamic News アプリケーションの概要

Dynamic News アプリケーションは、Oracle8i データベースとともに Oracle の XML プラットフォーム・コンポーネントを使用して Web ベースのニュース・サービスを構築します。

Java、XML、XSL、HTML および Oracle8i を組み合わせることで、Dynamic News が柔軟かつ強力になります。

- ニュース項目をデータベースに格納すると、ユーザー入力に基づいて問合せを実行し、コンテンツをパーソナライズできます。
- XML、XSL および HTML によって、複数のプラットフォーム用に表示をカスタマイズできます。
- Dynamic News アプリケーションは、できるだけ XML ドキュメントを事前生成し、パフォーマンスを向上させます。

問題: ブラウザで受信したニュースを、ユーザーの要求に従ってカスタマイズします。

ソリューション: このソリューションでは、Oracle の XML コンポーネント、Oracle8i データベースおよびカスタムのサブリットを使用します。詳細をこの章で説明します。

使用する Oracle の XML コンポーネント: XML Parser for Java および XML SQL Utility (XSU) for Java

Dynamic News の主なタスク

Dynamic News アプリケーションは、次のタスクを行う方法を提供します。

- データベースへのニュース・ヘッドラインの格納
- XML でのニュースの出力
- XSL スタイルシートを適用した新しいヘッドラインのフォーマット

Dynamic News アプリケーションの概要

Dynamic News は、ニュース項目（ヘッドライン）をデータベースから取り出し、HTML ページを構築します。HTML ページは、ユーザー設定項目に従ってカスタマイズされます。

このページには項目のリストが表示され、各項目は詳細記事にハイパーリンクされています。各ニュース項目には、次の属性があります。

- カテゴリ（スポーツやテクノロジーなど）
- サブカテゴリ（野球やソフトウェアなど）
- タイプ（特集記事や論評など）

3つのレベルのカスタマイズ: 静的、一部動的および動的

Dynamic News は、これらの属性を使用して次の3つのレベルのカスタマイズを行います。

- 静的
- 一部動的
- 動的

表 6-1 に、これらのレベルの選択基準を示します。

表 6-1 Dynamic News: カスタマイズの3つのレベル

カスタマイズ・レベル	説明
静的	<p>静的なページはカスタマイズされません。</p> <p>このレベルのエンド・ユーザーは、カテゴリ、サブカテゴリおよびタイプのそれぞれからすべての項目をリストしたページを受け取ります。</p> <p>ニュース・システム管理者は、管理サプレットを使用して定期的（たとえば1時間ごと）に静的XMLドキュメントを生成します。</p> <p>アプリケーションは、このようなページを必要に応じて構築できますが、ユーザーの要求ごとに問合せを実行して同じページを構築するよりも、事前生成したページを提示する方が高速です。</p>
一部動的	<p>一部動的なページは、事前生成した項目リストを組み合せます。</p> <p>エンド・ユーザーが1つ以上のカテゴリを選択すると、Dynamic News はこれらのカテゴリの項目をリストしたページを構築します。ニュース管理者は、管理サプレットを定期的に使用して、各カテゴリの項目のリストを事前生成します。</p> <p>一部動的なページは静的なページと同様に、パフォーマンス向上のために事前生成されたドキュメントから構築されます。</p>
動的	<p>動的なページは、エンド・ユーザーに要求されたときに構築されます。コンテンツは直接データベースから取り出されます。事前生成は行われません。</p> <p>まず、エンド・ユーザーがサプレットを起動してカテゴリ、サブカテゴリおよびタイプを選択します。次に、Dynamic News が前述の基準に一致する項目をデータベースから問い合わせ、その結果セットを使用してXMLドキュメントを構築します。その後、静的なページおよび一部動的なページと同様に、XSLT変換を適用してHTMLを生成します。</p>

注意：「動的」および「静的」という用語は、動作ではなくページのコンテンツを示します。

Dynamic News の SQL の例 1: 項目スキーマ nisetup.sql

ニュース項目の構造を定義する nisetup.sql の SQL を次に示します。

```
CREATE TABLE news.NEWS_ITEMS
( ID      NUMBER NOT NULL,
  TITLE    VARCHAR2(200),
  URL      VARCHAR2(200),
  DESCRIPTION  VARCHAR2(2000),
  ENTRY_DATE    DATE,
  CATEGORY_ID   NUMBER,
  SUB_CATEGORY_ID NUMBER,
  TYPE_ID      NUMBER,
  SUBMITTED_BY_ID NUMBER,
  EXPIRATION_DATE DATE,
  APPROVED_FLAG VARCHAR2(1)
);
```

Dynamic News のサーブレット

表 6-2 に、Dynamic News アプリケーションで使用するサーブレットを示します。これらのサーブレットは、アプリケーションのロジックを提供します。

表 6-2 Dynamic News サーブレット

サーブレット	説明	ファイル名
管理サーブレット	<ul style="list-style-type: none">■ データベースにニュース項目を追加します。■ ユーザー、タイプおよびカテゴリのリストを保持します。■ 静的（カスタマイズされない）ニュース・ページ用に XML および HTML を生成します。	xmlnews/admin/AdminServlet.java
一部動的ページ用サーブレット	エンド・ユーザーが選択したカテゴリ内のニュース項目のリストを生成します。	xmlnews/dynamic/SemiDynamicServlet.java
動的ページ用サーブレット	データベースからニュース項目を取り出し、エンド・ユーザーの設定項目に基づいてカスタム化されたページを生成します。	xmlnews/dynamic/DynamicServlet.java

Dynamic News の動作 : 鳥瞰図

HTML ページ構築のための XML ドキュメントの生成

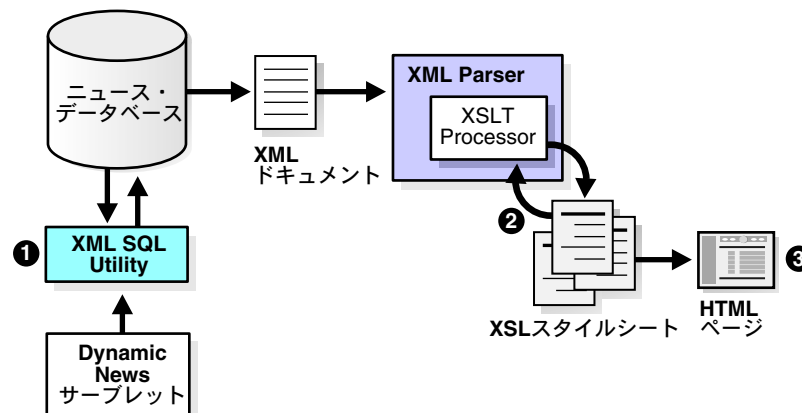
Dynamic News は、XML ドキュメントを生成して次のページの HTML を構築します。

- 静的なページ: ニュース・システム管理者が設定した間隔で、事前生成された XML ドキュメントから構築されます。
- 一部動的なページ: ユーザーが選択したカテゴリ内の項目がリストされている、事前生成された XML ドキュメントから構築されます。
- 動的なページ: ユーザーが選択したカテゴリ、サブカテゴリおよびタイプごとに項目がリストされている XML ドキュメントから必要に応じて構築されます。

図 6-1 に、Dynamic News がこれらの手順を実行する方法を示します。

1. Oracle XML SQL Utility (XSU) をコールします。このコールはデータベースからニュース項目を問い合わせ、結果を XML ドキュメントに書き込みます。これは次のモードで行われます。
 - 静的なページの場合はバッチ・モード
 - 一部動的なページの場合はバッチ・モード
 - 動的なページの場合は必要に応じて
2. Oracle XML Parser for Java の XSLT Processor を使用して、3 つの XSL スタイルシートのうちの 1 つを介して XML を HTML に変換します。スタイルシートには、Netscape Navigator 用スタイルシート、Internet Explorer 用スタイルシート、および他のすべてのブラウザ用の一般的なスタイルシートがあります。
3. Web サーバーを介して HTML ページをユーザーに配信します。

図 6-1 Dynamic News



静的なページ

Dynamic News は、静的なページを生成してすべての使用可能なニュース項目を表示します。これらのページはニュース・システム管理者が設定した間隔で（たとえば1時間ごとに）構築されます。管理者が間隔を設定しない場合は、これらのページは変更されません。

静的なページを使用する場合

静的なページは、データがあまり頻繁に変更されないすべてのアプリケーションで有効です。たとえば、ERP や顧客アプリケーションから日報を公開する場合などです。コンテンツが静的であるため、ユーザーの要求ごとにページを構築するよりも、ページを事前生成しておく方がより効率的です。

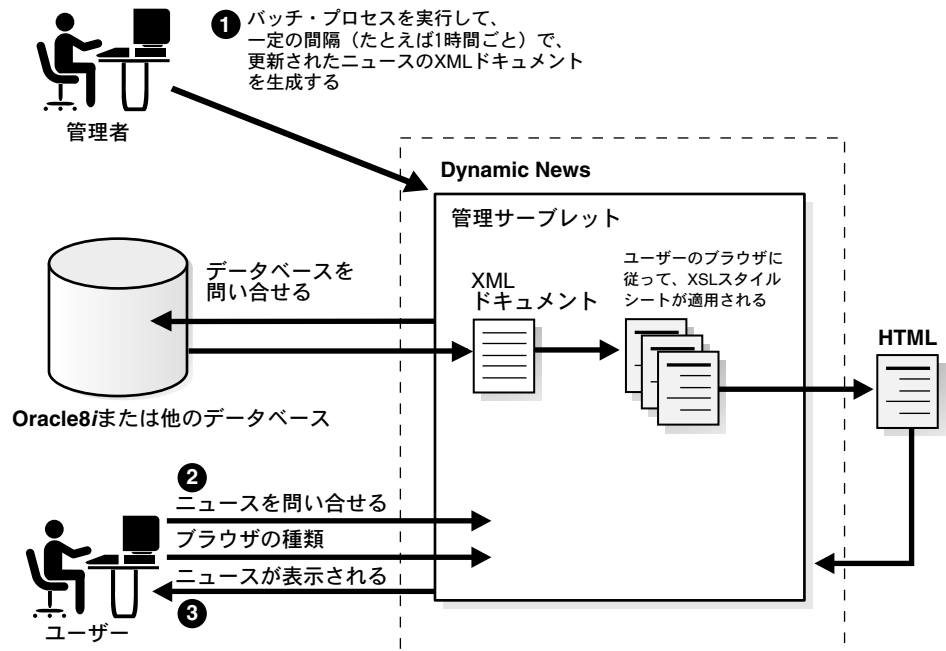
静的なページの動作

管理者は、管理サプレットから実行されるバッチ・プロセスを実行します。このプロセスはデータベースに問い合わせ、XML ドキュメントを生成します。エンド・ユーザーが Dynamic News を起動してすべてのニュースを表示すると、サプレットは HTTP 要求のユーザー・エージェント・ヘッダーからブラウザの種類を取得し、XML ドキュメントを読み込み、適切な XSL スタイルシートを適用します。

最後に、サプレットは図 6-2 に示すとおり、エンド・ユーザーのブラウザ用にフォーマットされた HTML ページを返します。

別の方法として、XSL スタイルシートをバッチ・プロセスの一部として適用し、スタイルシートごとに1つの HTML ファイルを生成できます。この場合は、管理するファイルが多くなりますが、実行するサプレットのサイズは小さくなります。

図 6-2 Dynamic News: 静的なページ - XML ドキュメントの生成



一部動的なページ

アプリケーションは、事前生成されたリストを組み合わせ一部動的なページを構築します。カテゴリごとの項目のリスト（各カテゴリに 1 つの XML ファイル）は管理者が事前生成しますが、このリストを含むページはユーザーごとにカスタマイズされます。エンド・ユーザーは、スポーツ、ビジネス、エンターテインメントなどのカテゴリを選択します。

一部動的なページを使用する場合

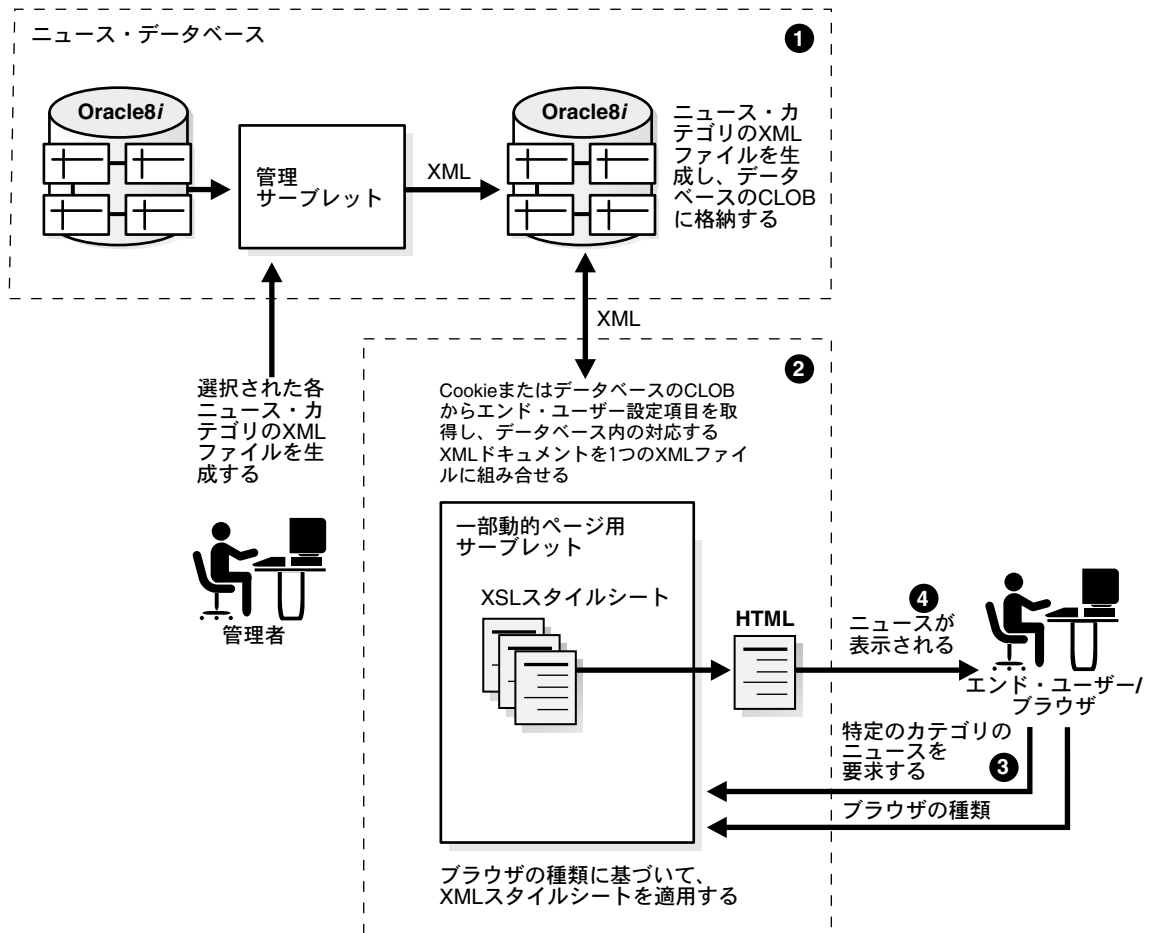
一部動的な方法は、データがあまり変更されず、エンド・ユーザーの選択肢を比較的少なくする場合に有効です。選択肢を提供するアプリケーションが提供する選択肢が多くなると、事前生成する必要があるドキュメントも多くなるため、利点が相対的に少なくなります。

一部動的なページの動作

図 6-3 に、一部動的なページの生成動作を示します。次の 2 つのフェーズがあります。

- **フェーズ 1 - 静的処理フェーズ:** 管理者は、管理サプレットを定期的を使用して XML ファイルを事前生成し、これらのファイルをデータベースの CLOB に格納します。これらのファイルは単純なフラット・ファイル・システムにも格納できますが、データベースのパフォーマンスは向上しません。
- **フェーズ 2 - 動的処理フェーズ:** このフェーズは、エンド・ユーザーが、指定したカテゴリからニュース項目を要求すると始まります。サプレットはデータベースから CLOB を取り出し、これらの CLOB を 1 つの XML ドキュメントに組み合わせます。サプレットは、ユーザー設定項目をデータベース側およびクライアント側の両方の Cookie に格納し、パフォーマンスを向上するために、できるだけ Cookie からこのユーザー設定項目を読み込みます。その後、エンド・ユーザーのブラウザに一致する XSL スタイルシートを使用して、XML ドキュメントを HTML ページに変換します。静的なページの場合と同様に、サプレットは HTTP 要求のユーザー・エージェント・ヘッダーからブラウザの種類を取得します。

図 6-3 Dynamic News: 一部動的なページ - XML ドキュメントの生成



動的なページ

アプリケーションは、データベースから項目を直接取り出すことで、必要に応じて動的なページを構築します。エンド・ユーザーは、「Create/Edit User Preference Page」にアクセスしてカテゴリ、サブカテゴリおよびタイプを選択します（たとえば、「エンターテインメント」- 「映画」- 「レビュー」）。

動的なページを使用する場合

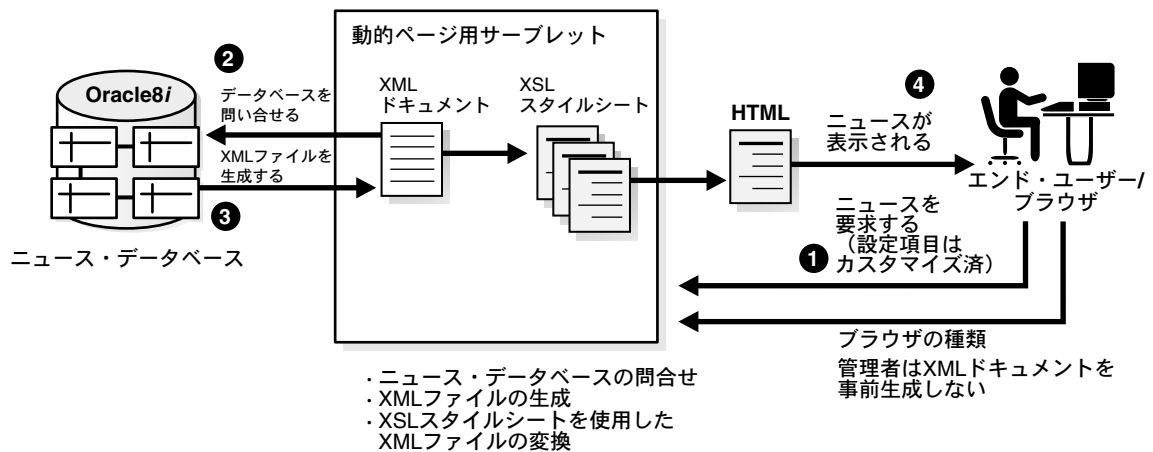
動的なページは、ニュース速報のような最新の情報の配信に有効です。このページは、特定の株における過去 10 年間での任意の日付の終値のような、過去のデータの配信にも有効です。すべてのありうる要求に対してドキュメントを事前作成しておくことは実用的ではありません。データベースから数値を取り出す方が単純で効率的です。

動的なページの動作

図 6-4 に、動的なページの生成動作を示します。他のモデルとは異なり、管理者は XML ドキュメントを事前生成しません。かわりに、動的ページ用サブルーットがエンド・ユーザーのカスタマイズの選択に基づいて、ニュース項目をデータベースから問い合わせます。

サブルーットはユーザー設定項目をデータベース側およびクライアント側の両方の Cookie に格納し、パフォーマンスを向上するために、できるだけ Cookie からこのユーザー設定項目を読み込みます。サブルーットは、問合せ結果を使用して XML ファイルを生成し、XSL スタイルシートを使用して、このファイルをユーザーのブラウザ用の HTML ページに変換します。他の方法と同様に、アプリケーションは HTTP 要求のユーザー・エージェント・ヘッダーからブラウザの種類を取得します。

図 6-4 Dynamic News: 動的なページ - XML ドキュメントの生成



コンテンツのパーソナライズ

Oracle8i によって、Dynamic News が柔軟になります。ニュース項目はデータベースに格納されるため、Dynamic News は必要に応じてコンテンツをカスタマイズできます。この項のコード例は、アプリケーションが、エンド・ユーザーが指定したカテゴリにあるニュース項目を取り出してページをパーソナライズする方法を示しています。主なタスクは次のとおりです。

1. エンド・ユーザー設定項目を取得します。
2. データベースからニュース項目を取り出します。
3. ニュース項目を組み合わせでドキュメントを構築します。
4. アプリケーションは、パーソナライズしたコンテンツを作成してからページの表示をカスタマイズし、エンド・ユーザーのブラウザ用にフォーマットします。フォーマットについてはこのマニュアルの後半で説明します。

エンド・ユーザー設定項目の取得

設定項目を処理するロジックは、アプリケーション内に分散されています。このロジックによって、データはデータベース側とクライアント側の Cookie の両方に格納されます。アプリケーションはパフォーマンスの向上のために、できるだけ設定項目データを Cookie から読み込みます。Cookie からデータを取得できない場合（たとえば、エンド・ユーザーがそのサイトに初めてアクセスする場合、またはエンド・ユーザーのブラウザが Cookie を受け入れない場合）は、データベースから設定項目データを読み込みます。

クライアント側の Cookie からの読み込み

次に、アプリケーションが、Cookie に格納されている設定項目データの処理に使用する 2 つのメソッドを示します。どちらのメソッドも、xmlnews.common.UserPreference にあります。Cookie の例を示します。

DynamicServlet=3\$0\$0#4\$2\$1***242

Cookie は、区切り記号として設定項目値にはドル記号、カテゴリにはシャープ記号を使用します。また、ユーザー ID と設定項目データを区切るトークンとして 3 つのアスタリスクを使用します。前述の Cookie の例は、ユーザー 242 がカテゴリ 3 および 4 の項目を要求していることを示しています。ユーザーは、カテゴリ 3 ではすべてのサブカテゴリにあるすべてのタイプの項目を要求しています（値 0（ゼロ）は、すべての項目を選択します）。ユーザーは、カテゴリ 4 ではサブカテゴリ 2 からの項目のみを要求しており、さらにサブカテゴリ 2 内のタイプ 1 の項目のみ要求しています。

サンプル・アプリケーションは、このような Cookie を次の手順で処理します。

1. `getNewsCookie` が、HTTP 要求を発行したブラウザから「DynamicServlet」の Cookie を取得します。
2. `loadPreferenceFromCookie` が「DynamicServlet」の Cookie を解析し、ユーザー ID と設定項目を含む文字列を取得します。

```
public Cookie getNewsCookie(HttpServletRequest request)
    throws Exception {
    Cookie c[] = request.getCookies();
    Cookie l_returnCookie = null;
    for (int i = 0; (c != null) && (i < c.length); i++) {
        if (c[i].getName().equals("DynamicServlet")) {
            l_returnCookie = c[i];
        }
    }
    return l_returnCookie;
}

public Vector loadPreferenceFromCookie(Cookie p_cookie) throws Exception {
    Vector l_prefId = new Vector(2);
```



```

String l_Preferences = p_cookie.getValue();
StringTokenizer l_stToken = new StringTokenizer(l_Preferences, "****");
String l_userId = "";
while (l_stToken.hasMoreTokens()) {
    // 最初のトークンはユーザー設定項目です。
    l_Preferences = l_stToken.nextToken();
    // 2 番目のトークンはユーザー ID です。
    l_userId = l_stToken.nextToken();
}
l_prefId.addElement(l_Preferences);
l_prefId.addElement(l_userId);
return l_prefId;
}

```

データベースの問合せ

アプリケーションは、Cookie から設定項目を読み込めない場合はデータベースを問い合わせます。xmlnews.common.GenUtility クラスが、データベースに接続してニュースのカテゴリ、サブカテゴリおよびタイプをフェッチするメソッドを実装します。

一部動的ページ用サブリットおよび動的ページ用サブリットの両方は、これらのメソッドと loadInitialPreference メソッドおよび constructUserPreference メソッドをコールします。これらは両方とも xmlnews/common/UserPreference.java 内で実装されます。

loadInitialPreference メソッドは getSubCategories をコールし、結果セットをループして、カテゴリ値を区切り文字と組み合わせて設定項目文字列を構築します。

```

public String loadInitialPreference(Vector p_category, Vector p_subcategory,
    Vector p_types, Connection p_con)
throws Exception {
    GenUtility m_general = new GenUtility();
    ...
    for (int i = 0; i < p_category.size(); i++) {
        String l_cat[] = (String []) p_category.elementAt(i);
        l_category = l_cat[0];
        Vector l_subcategory = m_general.getSubCategories(p_con, l_cat[0]);

        for(int l_j = 0, l_k = 0; l_j < l_subcategory.size(); l_j++, l_k++)
        {
            ...
            // 次の設定項目を構成文字列に追加します。
            l_userPref = l_userPref+"#"+l_category+"$"+l_subCat+"$"+l_typeStr;
        }
    }
}

```

```

...
    return l_userPref;
}

public static Vector getSubCategories(Connection p_conn, String p_categoryId)
    throws Exception {
    Vector l_subCats = new Vector();

    PreparedStatement l_pstmt = p_conn.prepareStatement(
        "Select id, name from sub_categories where category_id = ? ");
    l_pstmt.setString(1, p_categoryId);
    ResultSet l_rset = l_pstmt.executeQuery();

    while (l_rset.next()) {
        String[] l_subCat = new String[2];
        l_subCat[0] = new String(l_rset.getString(1));
        l_subCat[1] = new String(l_rset.getString(2));
        l_subCats.addElement(l_subCat);
    }
    l_pstmt.close();
    return l_subCats;
}

```

たとえば、次のコードは `xmlnews.dynamic.DynamicServlet.service` にあります。

このコードはこれらのメソッドをコールし、エンド・ユーザー設定項目をデータベースから読み込みます。その後、設定項目を使用して HTML ページを構築します。

```

public void service(HttpServletRequest p_request,
    HttpServletResponse p_response)
    throws ServletException {

    // 次の文は、他の場所でクラス変数として宣言され、
    // サブプレットの init メソッドで初期化されます。
    // GenUtility m_general = null;

    // m_general = new GenUtility();
    // UserPreference m_userPreference = null;
    // m_userPreference = new UserPreference();
    ...

    // データベース接続がクローズされている場合、再オープンします。
    if (m_connection == null || m_connection.isClosed())
        m_connection = m_general.dbConnection();
    ...
}

```

```
String l_preference = m_userPreference.loadInitialPreference(
    m_general.getCategories(m_connection),
    null, m_general.getTypes(m_connection),
        m_connection);

m_userPreference = m_userPreference.constructUserPreference
    ( l_preference,m_status);

// 動的なページを表示します。
    this.sendDynamicPage(l_browserType, p_response,
        l_username, m_userPreference,
        m_servletPath + "?REQUEST_TYPE=SET_ADVANCED_USER_PREFS",
        m_servletPath + "?REQUEST_TYPE=LOGIN_REQUEST",
        m_servletPath + "?REQUEST_TYPE=LOG_OUT_REQUEST",
        m_servletPath);
    ...
}
```

データベースからのニュース項目の取出し

xmlnews.admin.AdminServlet.performGeneration および
xmlnews.admin.AdminServlet.staticProcessingHtml にある次のコードは、アプリケーションが、使用可能な各カテゴリにあるニュース項目をデータベースに問い合わせ、それぞれの結果セットを XML ドキュメントに変換する方法を示しています。

データベースは、各カテゴリ用の XML を CLOB（キャラクタ・ラージ・オブジェクト）として格納します。このため、アプリケーションは非常に長いリストを処理できます。

```
public void performGeneration(String p_user, String p_genType,
    HttpServletResponse p_response)
    throws ServletException, IOException {
    ...
    try {
        String l_fileSep = System.getProperty("file.separator");
        String l_message = ""; // 状態メッセージを保持します。

        if (p_genType.equals("BATCH_GEN")) { // バッチ生成
            String l_htmlFile = "BatchGeneration";
            String l_xslFile = "BatchGeneration";
            String l_xmlFile = "BatchGeneration";

            //XML および HTML コンテンツを生成し、ファイルに保存します。
            this.staticProcessingHtml(
                m_dynNewsEnv.m_dynNewsHome+l_fileSep+l_htmlFile+".html",
                m_dynNewsEnv.m_dynNewsHome+l_fileSep+m_dynNewsEnv.m_batchGenXSL,
                m_dynNewsEnv.m_dynNewsHome+l_fileSep+l_xmlFile+".xml"
            );
            ...
        }
        ...
    }
}
```

xmlnews.admin.AdminServlet.staticProcessingHtml メソッドは、ニュース項目をフェッチする問合せを定義および実行します。その後このメソッドは、Oracle XML SQL Utility (XSU) を使用して結果セットから XML ドキュメントを構築し、XSLT 変換を適用することで HTML ページを作成します。

```
public void staticProcessingHtml(String p_htmlFile,String p_xslfile,
    String p_xmlfile) throws Exception {
    String l_query = "select a.id, a.title, a.URL, a.DESRIPTION, " +
        " to_char(a.ENTRY_DATE, 'DD-MON-YYYY'), a.CATEGORY_ID, b.name, " +
        a.SUB_CATEGORY_ID, c.name, a.Type_Id, d.name, " +
```

```

" a.Submitted_By_Id, e.name, to_char(a.expiration_date, 'DD-MON-YYYY'),
                                a.approved_flag " +
" from news_items a, categories b, sub_categories c, types d, users e where " +
" a.category_id is not null and a.sub_category_id is not null and "+
" a.type_id is not null and a.EXPIRATION_DATE is not null and "+
" a.category_id = b.id AND a.SUB_CATEGORY_ID = c.id AND a.Type_ID = d.id
                                AND " +
" a.SUBMITTED_BY_ID = e.id AND "+
" a.EXPIRATION_DATE > SYSDATE AND "+
" a.APPROVED_FLAG = 'A\' ORDER BY b.name, c.name ";

Statement l_stmt = m_connection.createStatement();
ResultSet l_result = l_stmt.executeQuery(l_query);
//Oracle XML SQL Utility を使用して、XML ドキュメントを作成します。
XMLDocument l_xmlDocument = m_xmlHandler.constructXMLDoc(l_result);
l_stmt.close();

// 対応する XSL を XML に適用して、HTML 文字列を取得します。
String l_htmlString = m_xmlHandler.applyXSLtoXML(l_xmlDocument,p_xslfile);

File l_file = new File(p_htmlFile);
FileOutputStream l_fileout = new FileOutputStream(l_file);
FileOutputStream l_xmlfileout = new FileOutputStream(new File(p_xmlfile));
l_fileout.write(l_htmlString.getBytes());
l_xmlDocument.print(l_xmlfileout);

l_fileout.close();
l_xmlfileout.close();
}

```

ドキュメント構築のためのニュース項目の組合せ

コンテンツのパーソナライズの最後の手順では、エンド・ユーザー設定項目に従って XML ドキュメントを HTML ページに変換します。

次のコードは、`xmlnews.generation.SemiDynamicGenerate.dynamicProcessing` にあります。

このコードはユーザーが選択したカテゴリに対応する CLOB を取り出し、各 CLOB を XML ドキュメントに変換し、これらのドキュメントを 1 つの XML ドキュメントに組み合わせます。XML ドキュメントを HTML ページに変換するプロセスについては、次の項を参照してください。

```
public XMLDocument semiDynamicProcessingXML(Connection p_conn, UserPreference p_prefs)
    throws Exception
{
    String l_htmlString = null ;
    XMLDocument l_combinedXMLDocument = null ;
    XMLDocument [] l_XMLArray = new XMLDocument[p_prefs.m_categories.size()];
    int l_arrayIndex = 0 ;

    PreparedStatement l_selectStmt = p_conn.prepareStatement(
        " SELECT PREGEN_XML FROM CATEGORIES_CLOB WHERE CATEGORY_ID = ?");
    // 各設定項目を処理します。
    for ( ; l_arrayIndex < p_prefs.m_categories.size(); ++l_arrayIndex ){
        l_selectStmt.setString(1, p_prefs.m_categories.elementAt(l_arrayIndex).toString());
        OracleResultSet l_selectRst = (OracleResultSet)l_selectStmt.executeQuery();
        if (l_selectRst.next()) {
            CLOB l_clob = l_selectRst.getCLOB(1);
            l_XMLArray[l_arrayIndex] = convertFileToXML(l_clob.getAsciiStream());
        } else
            l_XMLArray[l_arrayIndex] = null ;
    }
    l_selectStmt.close();

    XMLDocHandler l_xmlHandler = new XMLDocHandler();
    l_combinedXMLDocument = l_xmlHandler.combineXMLDocumnts(l_XMLArray );
    return l_combinedXMLDocument ;
}
```

表示のカスタマイズ

Dynamic News はデータベースからニュース項目をフェッチし、これらの項目を XML ドキュメントへ変換します。XML によって表示とコンテンツが分割されるため、カスタム HTML ページを簡単に構築できます。

Dynamic News は、次のような異なる XSL スタイルシートを使用して、XML ドキュメントをブラウザ用にカスタマイズされた HTML に変換します。

- Netscape Navigator 用のスタイルシート
- Microsoft Internet Explorer 用のスタイルシート
- 他のブラウザ用の一般的なスタイルシート

このプロセスは、次の手順で行われます。

1. ユーザーのブラウザの種類を取得します。
2. ニュース項目を取得します。
3. XML ドキュメントを構築します。
4. XML を HTML に変換します。

アプリケーションは HTTP 要求を受信するたびに、ユーザー・エージェント・ヘッダーを調べて要求を送信したブラウザの種類を判断します。

xmlnews.dynamic.DynamicServlet.service の次の行は、サーブレットが RequestHandler オブジェクトを作成し (xmlnews/common/RequestHandler.java 内で実装)、要求を解析してブラウザの種類を取得する方法を示しています。その後サーブレットは、この情報を使用して、エンド・ユーザー設定項目およびブラウザの種類に基づいて HTML ページを戻します。

```
public void service(HttpServletRequest p_request, HttpServletResponse p_response)
throws ServletException {
    ...
    // 要求ハンドラ（他の場所で宣言済）をインスタンス化します。
    m_reqHandler = new RequestHandler(m_userPreference, m_general, m_status);
    RequestParams l_reqParams = m_reqHandler.parseRequest(p_request, m_
connection);
    String l_browserType = l_reqParams.m_browserType;
    ...
    // 動的なページを表示します。
    this.sendDynamicPage(l_browserType, p_response, l_username, m_userPreference,
                        m_servletPath+"?REQUEST_TYPE=SET_ADVANCED_USER_PREFS",
                        m_servletPath+"?REQUEST_TYPE=LOGIN_REQUEST",
                        m_servletPath+"?REQUEST_TYPE=LOG_OUT_REQUEST",
                        m_servletPath);
    ...
}
```

ユーザー・エージェント・ヘッダーから実際にブラウザの種類を判別するコードは、`xmlnews.common.GenUtility.getBrowserType` 内にあります。このコードを次に示します。

```
public String getBrowserType(HttpServletRequest p_request) throws Exception
{
    // 要求に対応付けられたすべてのヘッダー名を取得します。
    Enumeration l_enum = p_request.getHeaderNames();

    String l_Version    = null;
    String l_browValue  = null;
    String l_browserType = null;

    while (l_enum.hasMoreElements()) {
        String l_name = (String)l_enum.nextElement();
        if (l_name.equalsIgnoreCase("user-agent"))
            l_browValue = p_request.getHeader(l_name);
    }

    // 値に「MSIE」という文字列が含まれている場合、ブラウザはInternet Explorerです。
    if (l_browValue.indexOf("MSIE") > 0 ) {
        StringTokenizer l_st = new StringTokenizer(l_browValue, ";");
        // Parse the Header to get the browser version.
        l_browserType = "IE";
        while (l_st.hasMoreTokens()) {
            String l_tempStr = l_st.nextToken();
            if (l_tempStr.indexOf("MSIE") > 0 ) {
                StringTokenizer l_st1 = new StringTokenizer(l_tempStr, " ");
                l_st1.nextToken();
                l_Version = l_st1.nextToken();
            }
        }
    }
    // 値に「en」という文字列が含まれている場合、ブラウザはNetscape です。
    } else if (l_browValue.indexOf("en") > 0 ) {
        l_browserType = "NET";
        String l_tVersion = l_browValue.substring(8);
        int l_tempInd  = l_tVersion.indexOf("(");
        l_Version = l_tVersion.substring(0, l_tempInd);
    }

    // 連結後、ブラウザの種類およびバージョンを戻します。
    return l_browserType + l_Version;
}
```


エンド・ユーザーのブラウザの種類を取得したら、DynamicServlet のサービス・メソッドはこの種類を `xmlnews.dynamic.DynamicServlet.sendDynamicPage` に渡します。

このメソッドは、データベースから XML ドキュメントをフェッチし、エンド・ユーザーのブラウザの種類に適切な XSL スタイルシートをこれらのドキュメントに適用して HTML に変換します。

```
public void sendDynamicPage(String p_browserType,HttpServletResponse p_response,
    String p_userName,UserPreference p_pref,String p_userPrefURL,
    String p_signOnURL,String p_logout,
    String p_servletPath) throws Exception {
    String l_finalHTML = ""; //HTMLを保持します。
    if (p_browserType.startsWith("IE4") || (p_browserType.startsWith("IE5"))) {
        //XML および XSL をパラメータとして送信し、HTML 文字列を取得します。
        l_finalHTML = m_handler.applyXSLtoXML(
            this.dynamicProcessingXML(m_connection, p_pref),
            m_dyEnv.m_dynNewsHome + "/DynamicIE.xml"
        );
    }
    String l_thisbit = m_general.postProcessing(l_finalHTML,p_userName,
        p_userPrefURL,p_signOnURL,p_logout,p_servletPath);
    PrintWriter l_output = p_response.getWriter();
    l_output.print(l_thisbit);
    l_output.close();
}
else if (p_browserType.startsWith("NET4") ||
    (p_browserType.startsWith("NET5"))) {
    // 同様の操作を、スタイルシート「/DynamicNS.xml」を適用して行います。
    ...
    // ブラウザが IE または Netscape 以外の場合。
} else {
    // 同様の操作を、スタイルシート「/Dynamic.xml」を適用して行います。
    ...
}
```

キーとなるメソッドは次のとおりです。

- `xmlnews.dynamic.DynamicServlet.dynamicProcessingXML`

このメソッドは、エンド・ユーザー設定項目に一致するニュース項目をデータベースに問い合わせます。その後、`xmlnews.common.XMLDocHandler.constructXMLDoc` をコールして、結果セットを XML ドキュメントに変換します。

- `xmlnews.common.XMLDocHandler.applyXSLtoXML`

このメソッドは、指定した XSL スタイルシートを使用して XML ドキュメントを HTML に変換します。このメソッドは、Oracle XML Parser の XSL Transformation 機能を使用します。具体的には、このメソッドは DOM パーサーを使用して、XML ドキュメントの構造を表すツリーを作成します。このメソッドは、最終的な HTML 文字列を構築するために、ツリーのルートとして機能する要素を作成し、解析済の DOM ドキュメントを追加します。

ニュース項目のインポートおよびエクスポート

Dynamic News は、Resource description framework Site Summary (RSS) 標準に準拠する XML ドキュメントをインポートおよびエクスポートできます。データ・チャネルを共有する方法として Netscape 社によって開発された RSS は、my.netscape.com や slashdot.org などの Web サイトで使用されています。

アプリケーションは、RSS を使用してニュース・ページを発表（ニュース・ページを RSS ホストに使用可能にする）したり、他の RSS サイトからのニュースを集約することができます。たとえば、Dynamic News には `xmlnews.admin.RSSHandler` クラスが含まれています。このクラスは、指定された DTD を使用して、指定されたファイルのニュース項目を解析および抽出します。その後、このクラスはニュース項目をハッシュ表に格納します。また、そのハッシュ表の要素を戻すこともできます。

XML によるデータ表示のパーソナライズ : Portal-to-Go

この章の内容は次のとおりです。

- [Oracle Portal-to-Go の概要](#)
- [Portal-To-Go 1.0.2 の機能](#)
- [Portal-to-Go 実行の要件](#)
- [Portal-to-Go: サポートするデバイスおよびゲートウェイ](#)
- [Portal-to-Go の動作](#)
- [Portal-to-Go コンポーネント](#)
- [XML でのデータ交換 : Portal-to-Go を使用したソースから XML、XML からターゲットへの配信](#)
- [コンテンツの取出し](#)
- [XML への変換](#)
- [サンプルのアダプタ・クラス](#)
- [ターゲットのマークアップ言語への XML の変換](#)
- [Portal-to-Go: Java Transformer](#)
- [Portal-to-Go: XSL Stylesheet Transformer](#)
- [Portal-to-Go の例 1: オンライン・ドラッグストアの販売市場の拡大](#)
- [Portal-to-Go の例 2: 銀行サービスの拡大](#)

Oracle Portal-to-Go の概要

現在、Web クライアントのほとんどは PC ですが、Meta Group 社は「2003 年までにはインターネット・アクセスの 50% が PC 以外からのアクセスになる」と予想しています。

Oracle Portal-to-Go (Portal-to-Go) は、次のサービスを可能にします。

- すべてのワイヤレス・デバイスで、セキュリティ対策済の E-Business アプリケーションを含む、既存の Web またはデータベース・アプリケーションやコンテンツすべてにアクセスできるようになります。
- 携帯電話サービス企業が、広域 E-Commerce サービス・プロバイダの事業を行うことができます。

Oracle のインターネット・プラットフォームのコンポーネントである Portal-to-Go は、対応するすべてのデバイスに Web コンテンツを配信するための機能をすべて備えたサーバー製品です。Portal-to-Go は、既存のコンテンツをデバイス固有のフォーマットに変換し、エンド・ユーザー用のポータル・インタフェースを提供します。

重要な要素である XML

XML は、コンテンツ・プロバイダが、様々なフォーマットで配信されるデータをモバイル・ユーザーの対象読者に届けるための重要な要素です。XML は、ターゲット・デバイスのフォーマットからソース・コンテンツのフォーマットを分離します。これによって、コンテンツ・プロバイダはすべてのソースからデータを取り出し、このデータをすべてのターゲットに配信できます。これらの XML ベース技術は、次のフォーマットから別のフォーマットへのデータ変換に使用されます。

- エンタープライズ・アプリケーションの統合
- ユーザー・プロファイルに基づいたコンテンツ配信のカスタマイズ
- 市場サプライヤの取引形式でのコンテンツとサービスの集計

Portal-to-Go コンポーネント

Portal-to-Go ポータルには、次のコンポーネントがあります。

- モバイル・デバイスにデータを配信するサービス
- HTML および RDBMS のコンテンツを XML に変換するアダプタ
- XML を適切なマークアップ言語に変換するトランスフォーマ

この章では、Portal-to-Go が XML を使用してすべてのデバイスで Web コンテンツを使用可能にする方法について説明します。株相場サービスおよびデータ交換のための、中間フォーマットとしての XML の役割についても説明します。

Oracle の XML コンポーネント

XML Parser for Java が、Portal-to-Go Adapter および Portal-to-Go Transformer で使用されています。XML Parser for Java の XSLT パッケージも使用されています。

Portal-To-Go 1.0.2 の機能

Portal-To-Go 1.0.2 の機能は次のとおりです。

- Apache および Apache JServ のサポート
- サービスごとの明示的なコンテンツ・タイプ設定
- 出力変数名の明示的な設定など、デバイス出力のカスタマイズの向上
- オプションの入力パラメータのサポート
- マルチバイト・キャラクタ・セット処理の改善
- 特殊文字（\$ など）の処理の改善
- 既存の WML Transformer の拡張
- 以前のリリースの、多くの異なるトランスフォーマの代替となる単一の WML Transformer
- 外部サイトを Portal-to-Go ポータルに含めるためのブックマーク機能
- マルチバイト・キャラクタ・セットを使用したポータルの作成

参照： リポジトリのアップグレードの詳細は、『Portal-to-Go インストール・ガイド』を参照してください。

Portal-to-Go 実行の要件

Portal-to-Go を実行するには、次のものがが必要です。

- Oracle8i リリース 8.1.5 以上
- 次のサーバーのうちの 1 つ
 - iAS (Internet Application Server)
 - Apache Web サーバー 1.3.9 および Apache JServ 1.1
- Java 構成要件
 - Service Designer。Portal-to-Go Service Designer には、JDK 1.2.2 が必要です。JDK 1.2.2 は、Portal-to-Go の CD-ROM からインストールできます。
 - Web Integration Developer。Web Integration Developer には、独自の Java Virtual Machine (JVM) があります。Java のセットアップは必要ありません。
 - サーバー・コンポーネント。Portal-to-Go サーバー・コンポーネントは、JDK 1.1 または 1.2 で実行します。JDK 1.2 のパフォーマンスは向上しています。

Portal-to-Go: サポートするデバイスおよびゲートウェイ

トランスフォーマ

Portal-to-Go は、次のベンダー製の WAP 準拠の最新デバイス用トランスフォーマを提供します。

- Alcatel
- Ericsson (R320 を含む)
- Motorola (Timeport を含む)
- Neopoint (NP1000 を含む)
- Nokia (7100 を含む)
- Samsung

ユーザー独自のトランスフォーマを作成し、Portal-to-Go でサポートする対象を他のデバイスにも拡張することができます。

WAP ゲートウェイ

Portal-to-Go の動作は、次の WAP ゲートウェイで実証済です。

- hone.com UP.link Gateway
- Nokia WAP Gateway
- Ericsson WAP Gateway
- Infinite Technologies WAPLite

Portal-to-Go の動作

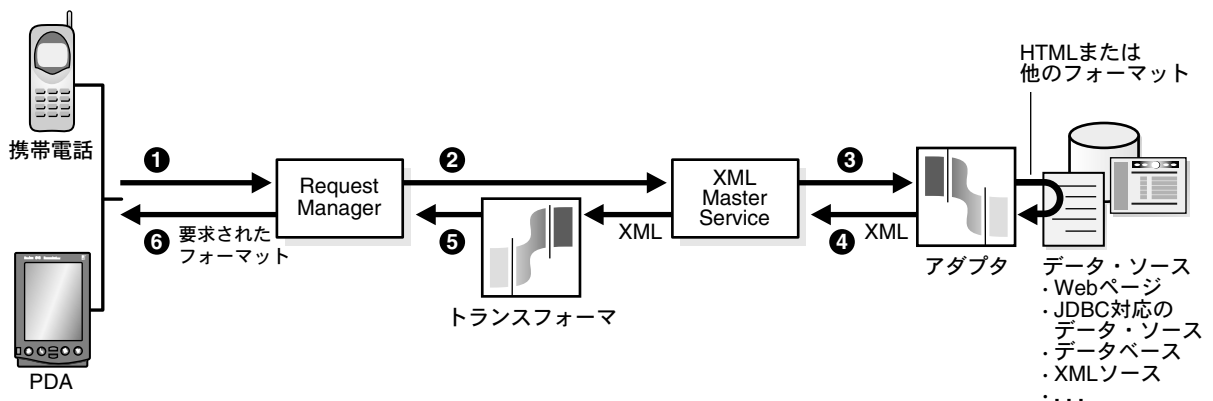
図 7-1 に、Portal-to-Go の動作を示します。エンド・ユーザーが Portal-to-Go サービスを要求すると、次のアクションが発生します。

1. Portal-to-Go の Request Manager は、認証を含むユーザー・レベルの事前処理を実行します。
2. Request Manager は、対応する Master Service に要求を送信します。
3. Master Service はアダプタを起動して、要求されたコンテンツを取り出します。
4. アダプタは、コンテンツを XML で返します。
5. トランスフォーマは、XML コンテンツをターゲット・デバイスに適合するフォーマットに変換します。
6. Request Manager は、情報をデバイスに返します。

XML および関連テクノロジーは、次の点で Portal-to-Go の機能の中心的な役割を果たします。

- XML は、表示とコンテンツを分離します。
- DTD は、XML タグをユーザー・インタフェース (UI) 要素へマップします。
- XSL スタイルシートは、結果のフォーマット、ソートおよびフィルタのためのルールを定義します。

図 7-1 Portal-to-Go の動作



Portal-to-Go コンポーネント

Portal-to-Go サービス

Portal-to-Go サービスは、Portal-to-Go クライアントが要求し、Portal-to-Go クライアントに配信される情報の単位をカプセル化します。サービスの例を次に示します。

- 株相場
- ニュース
- 地図
- 電子メール

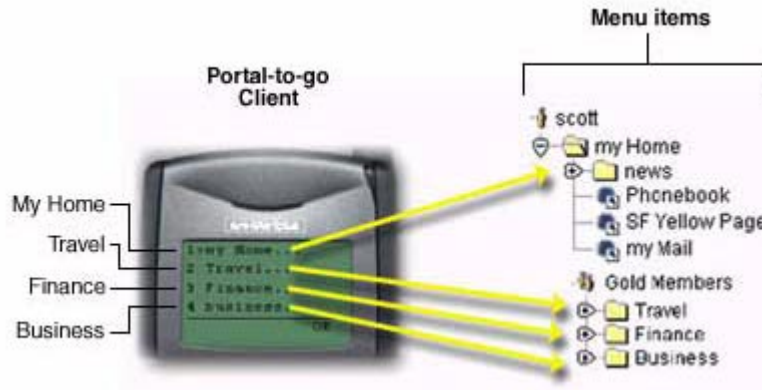
サービスは、既存の Web サイト、あらゆるデータベースへの問合せ、またはすべての XML ソースから構築できます。

Master Service

Master Service は、サービスを実装し特定のアダプタを起動する Portal-to-Go オブジェクトです。サービスは通常、エンド・ユーザーの携帯電話メニュー項目や Web ページ上のリンクに表示されます。エンド・ユーザーは、デバイスのインタフェース上のメニュー項目を選択して Master Service を起動します。Master Service は、次の種類のデータを戻します。

- 静的テキスト（映画のレビューなど）
- アプリケーション（航空券予約システムなど）

図 7-2 エンド・ユーザーに表示されるメニュー項目としてのサービス。Master Service はメニュー項目を選択すると起動されます。



Master Service は、Adapter をデバイスの Transformer にマップすることで、Portal-to-Go のコンテンツ・ソースを配信プラットフォームにリンクします。各 Master Service は、1つの Adapter に基づいています。Master Service は、使用する Adapter の独自のインスタンスを作成します。したがって、複数のサービスで同じタイプの Adapter を使用でき、各サービスは、サービス固有の引数値を渡すことができます。

Portal-to-Go Adapter

Portal-to-Go Adapter は、外部ソースからデータを取り出し、Portal-to-Go XML に渡す Java アプリケーションです。この Adapter は、Master Service によって起動されると、サービス・コンテンツを含む XML ドキュメントを返します。Adapter は、Portal-to-Go サーバーとコンテンツ・ソース間のインタフェースを提供します。

Adapter は次のタスクを実行します。

- データ・ソースへの接続
- コンテンツの取出し
- Portal-to-Go XML へのコンテンツの変換

Portal-to-Go は、一般的なコンテンツ・ソース（Web ページ、JDBC 対応データ・ソースを含む）用に事前作成済の Adapter を提供します。アダプタは、他のコンテンツ・ソースと動作するように変更できます。

すべてのアダプタは、Portal-to-Go XML を生成します。Portal-to-Go XML は、Portal-to-Go DTD に準拠し、整形形式および妥当な XML ドキュメントです。

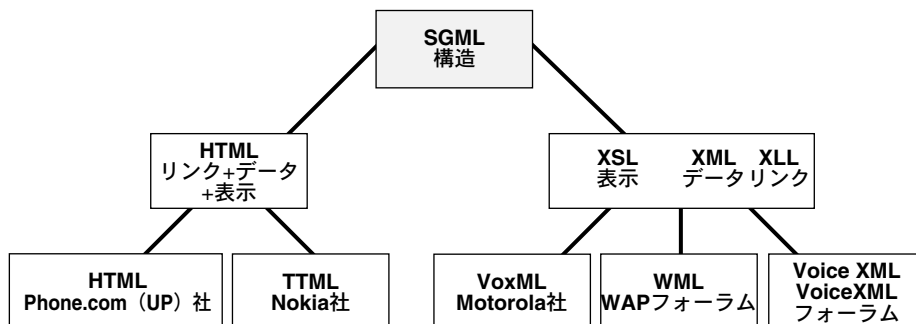
Portal-to-Go Transformer

Portal-to-Go Transformer は、XML ドキュメントをターゲットのフォーマットまたは他の Portal-to-Go フォーマットに変換する Java プログラムまたは XSLT スタイルシートです。Portal-to-Go Transformer は、テキストの再配置、フィルタおよび追加も行います。Transformer を使用すると、ターゲット・デバイスに最適なフォーマットでコンテンツを提示できます。Portal-to-Go は、次のマークアップ言語用のトランスフォーマを提供します。

- WML 1.1: WAP フォーラムで定義されたワイヤレス・マークアップ言語です。
- Tiny HTML: Palm Pilot など（電話以外の）のハンドヘルド・デバイスに適した HTML のサブセットです。
- VoxML: 音声によるアプリケーションとの対話を可能にする、Motorola 社が開発したマークアップ言語です。
- TTML: Tagged Text Mark-up Language (TTML) は、Nokia 社が開発した HTML のサブセットです。
- HDML: Handheld Devices Markup Language (HDML) は、ハンドヘルド・デバイス用に特別に設計された HTML の簡易バージョンです。
- プレーン・テキスト: ショート・メッセージ・サービス対応デバイスおよび電子メール・アプリケーションのコンテンツを変換します。

図 7-3 に、これらのマークアップ言語およびその派生言語を示します。

図 7-3 Portal-to-Go による複数の HTML ベースおよび XML ベースのマークアップ言語のサポート



Transformer を使用すると、あらゆるデバイス用にコンテンツの表示を最適化したり、新しいデバイス・プラットフォームをサポートすることができます。ほとんどの場合は、既存の Transformer を簡単に変更または再使用できます。

XML でのデータ交換 : Portal-to-Go を使用したソースから XML、XML からターゲットへの配信

XML を中間フォーマットとして、あらゆるソースからのデータをあらゆるデバイスへ配信できます。株相場情報およびヘッドラインを提供する Web アプリケーションがあり、これらの情報を携帯電話および PDA (Palm Pilot などのパーソナル・デジタル・アシスタント) に配信する場合について考えます。

コンテンツのフォーマットには各デバイス固有の要件があるため、各デバイスに同じデータを送信することはできません。この問題を解決するために、Portal-to-Go は、中間データ・フォーマットを XML で定義します。また、コンテンツ・プロバイダが次のタスクを実行できるようにするツールを提供します。

- ソース・コンテンツの取出し
- ソース・コンテンツの XML への変換
- 各デバイス用のマークアップ言語への XML の変換

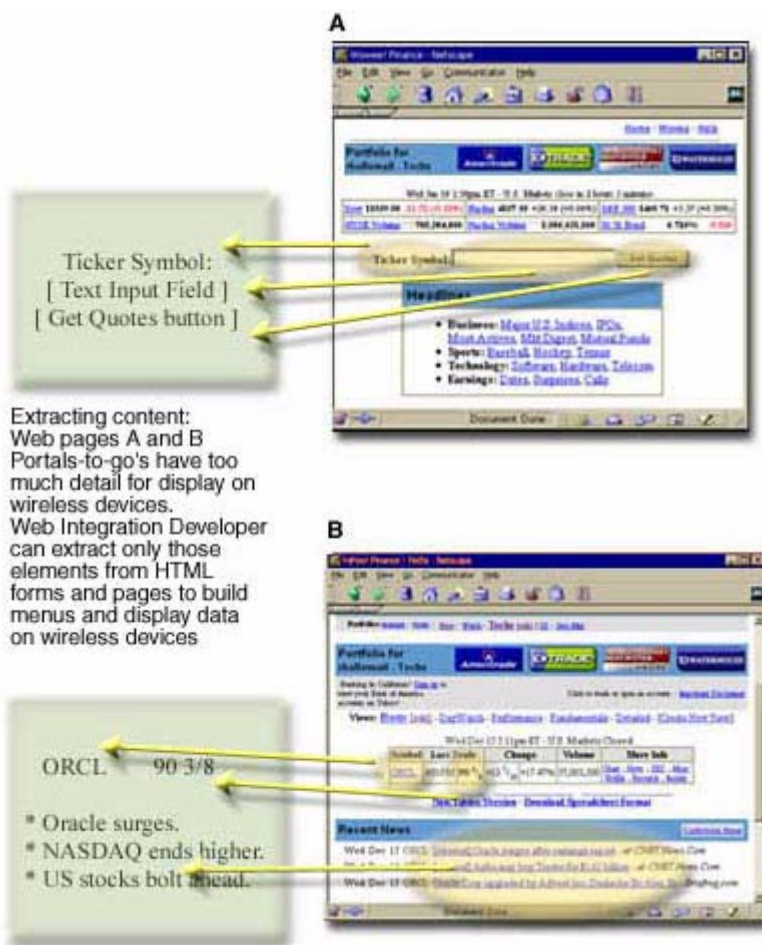
コンテンツの取出し

ハンドヘルド・デバイスには、デスクトップ・モニタほど多くの情報を表示することはできないため、情報を選択する必要があります。図 7-4 に、株式データ・アプリケーションの 2 つの Web ページを示します。これらは、意識的にわかりにくくしています。

- A: 1 つ目のページは、企業の株式相場表示機用記号を入力するフォームです。例えば、ORCL は、オラクル社の株式相場表示機用記号です。
- B: 2 つ目のページは、株価およびその企業のその他の情報を表示します。

どちらのページにも、多くの広告、ボタン、ハイパーリンク、関連記事などが配置されています。まず、サービスにアクセスさせる Web ページの要素を決定します。

図 7-4 ワイヤレス・デバイスに表示するための HTML ページからの要素の抽出



Web Integration Developer: スクリーン・スクレーパ

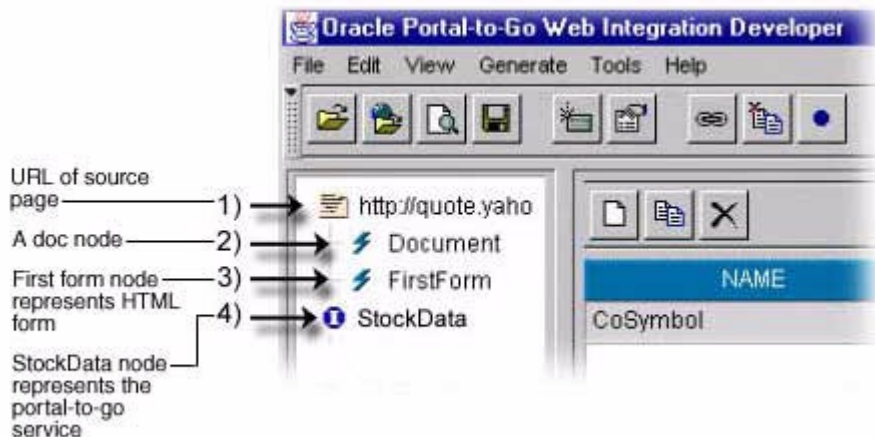
Portal-to-Go は、Web Integration Developer と呼ばれる GUI ツールを提供します。このツールは、Web ページからユーザー・インタフェース (UI) の要素を取り出す「スクリーン・スクレーパ」です。Web Integration Developer ツールおよび機能を使用すると、UI 要素を選択し、対応する出力パラメータおよび入力パラメータを定義できます。

図 7-5 に、Web Integration Developer の一部を示します。Document Browser パネルには、次の項目があります。

- ソース・ページの URL。
- ソース・ページのコンテンツ (段落、イメージ、リストおよび表) を表す Document ノード。
- HTML フォームを表す FirstForm ノード。Web Integration Developer は、ソース・ページにある各フォーム用にフォーム・ノードを作成します。
- Portal-to-Go サービスを表す StockData ノード。

次に、取り出した要素を XML に変換します。

図 7-5 Web Integration Developer を使用した UI 要素の取出し



XML への変換

Portal-to-Go Adapter は、ソースからコンテンツを取り出します。ここで示す例では、Portal-to-Go Adapter が特定の株相場およびヘッドラインを Web ページから取り出します。その後、Adapter はコンテンツを XML に変換します。

中間 XML フォーマットを使用する理由

ターゲット・デバイスのフォーマットに直接変換しない理由には、柔軟性と拡張性の 2 つがあります。ソースからターゲットに直接変換するには、それぞれのソースとターゲットのペア用のアダプタおよびトランスフォーマを効率的に作成する必要があります。中間フォーマットとして XML を使用すると、必要なのは各ソースに 1 つのアダプタ、および各デバイスに 1 つのトランスフォーマのみです。たとえば、2 つのコンテンツ・ソースと 3 つのターゲット・デバイスがあるとしたします。

- **XML を使用しない、ソースからターゲットへの変換:** それぞれ 6 つのアダプタとトランスフォーマ、合計 12 のコンポーネントが必要です。
- **XML を使用した、ソースからターゲットへの変換:** 2 つのアダプタと 3 つのトランスフォーマ、合計 5 つのコンポーネントのみ必要です。

Simple Result DTD の使用

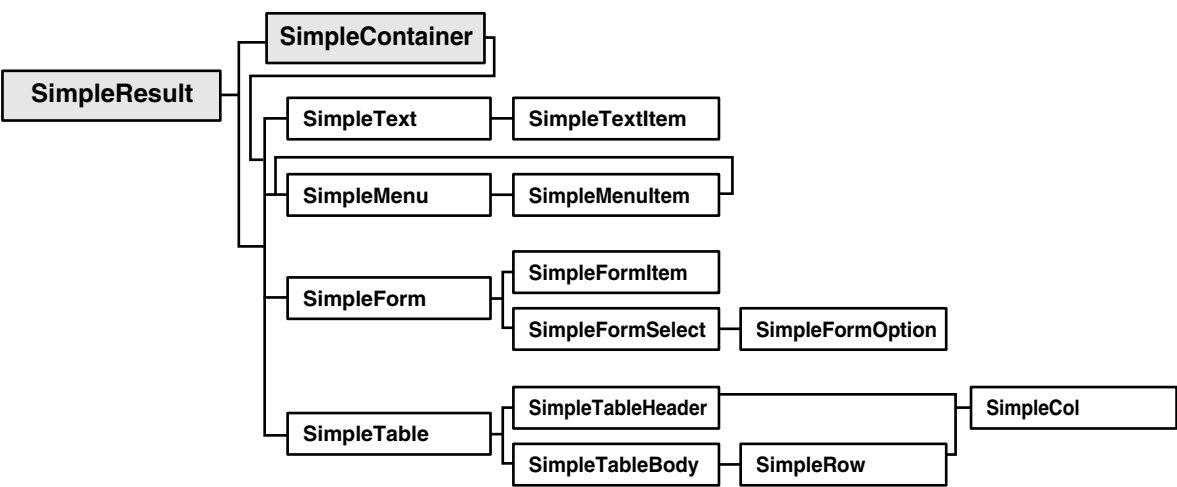
アダプタ出力を汎用にするには XML である必要があります。このためには、すべてのデータ型をあらゆるデバイスに表示できる XML ドキュメント・タイプを定義することが重要です。ドキュメント・タイプは、DTD (Document Type Definition) に定義されます。DTD は、使用可能な要素を記述することで、XML ドキュメントのクラスに文法を提供するファイルです。

Portal-to-Go は、Simple Result DTD を使用して、完全に汎用的な中間データ・フォーマットを作成します。Simple Result DTD にある要素は、取り出したユーザー・インタフェースの要素を表します。これらの要素は次のとおりです。

- テキスト項目
- メニュー
- フォーム
- 表

図 7-6 に、Simple Result DTD のコンテンツ・モデルを示します。

図 7-6 Simple Result DTD コンテンツ・モデル



次に、株式データの例に使用する要素を示す、SimpleResult.dtd の一部を示します。

```
<!--
エンティティ：「GENATTR」は、ほとんどの要素用の汎用属性を含みます。
属性：      「name」は要素の名前です。
           「title」は、要素のタイトルです。
...
-->
<!ENTITY % GENATTR "
            name    CDATA #IMPLIED
            title   CDATA #IMPLIED
            ...
            ">

<!--
要素：      「SimpleResult」は、結果の要素です。
使用方法：  結果を含めます。
子：        「SimpleText」は、結果のテキストです。
...
-->

<!ELEMENT   SimpleResult ((SimpleContainer|SimpleText|SimpleMenu|
SimpleForm|SimpleTable|SimpleImage|SimpleBreak)+) >
<!ATTLIST   SimpleResult %GENATTR;>
...

```



```

<!--
要素：      「SimpleText」は、1 ブロック以上のテキストを表示するための要素です。
使用方法：  ブレーン・テキストに使用します。
子：        「SimpleTextItem」は、テキストのブロックです。
-->

<!--ELEMENT   SimpleText (SimpleTextItem+)>
<!--ATTLIST   SimpleText %GENATTR;>

<!--
要素：      「SimpleTextItem」は、テキストのブロックです。
使用方法：  1 ブロックのテキスト（通常は 1 段落）を含みます。
子：        「#PCDATA」は、実際のテキストです。
-->

<!--ELEMENT   SimpleTextItem (#PCDATA)>
<!--ATTLIST   SimpleTextItem %GENATTR;>

...

<!--
要素：      「SimpleForm」は、1 つ以上の入力フィールドを表示するための要素です。
使用方法：  データ入力フォームとして使用します。
子：        「SimpleFormItem」は、各入力フィールドです。
属性：      「target」は、このフォームのリンク・ターゲットです。
            「section」は、セクション識別子です。

**** WIDL アダプタの特別なケース ****

-->
<!--ELEMENT   SimpleForm ((SimpleFormItem|SimpleFormSelect)+)>
<!--ATTLIST   SimpleForm %GENATTR;
              target  CDATA #REQUIRED
              section CDATA #IMPLIED>

<!--
要素：      「SimpleFormItem」は、単純なフォーム内の単一の入力項目です。
使用方法：  ユーザーの入力を取得します。
子：        「#PCDATA」は、サーバーにある記入済入力を含みます。

***** これは、デフォルトの属性をオーバーライドします。*****

属性：      「default」は、オプションのフィールドのデフォルト値を指定します。

```

**** デフォルト値は、フィールドが空の場合にのみ使用します。

「mandatory」は、フォーム項目が必須であることを示します。
「maxLength」は、入力の最大長を指定します。

```
-->  
<!ELEMENT   SimpleFormItem (#PCDATA)>  
<!ATTLIST   SimpleFormItem %GENATTR;  
            default   CDATA #IMPLIED  
            mandatory (yes|no) "no"  
            maxLength CDATA #IMPLIED>
```

...

Adapter による DTD 要素へのソース・コンテンツのマッピング

Portal-to-Go Adapter は、ソース・コンテンツを適切な SimpleResult 要素にマッピングします。

- 入力バインディングは、要求を完了するために必要なすべてのデータを、サービス URL 内のフォーム <input> タグおよび変数を使用して指定します。
- 出力バインディングは、リクエストに返される結果です。これらのバインディングは、サービスおよびデバイスに関連した HTML のフラグメントのみを選択します。

例として、表 7-1 に、仮定の StockData Adapter によって生成された入力フォーム用の XML (テキスト・ラベル、入力フィールドおよび送信ボタン) および結果ページ (株式相場表示機用記号、株価およびヘッドライン) を示します。

表 7-1 StockData Adapter が生成した入力ページおよび結果ページ用の XML

入力ページ用 XML	XML 結果ページ: 株相場およびヘッドライン・ページ
<pre><SimpleResult> <SimpleText> <SimpleTextItem name = "TickerField"> 株式相場表示機用記号 : </SimpleTextItem> </SimpleText> <SimpleForm title="Input Form"> <SimpleFormItem name="Ticker"> </SimpleFormItem> <SimpleFormButton name="submitBtn"> 株価を検索 </SimpleFormButton> </SimpleForm> </SimpleResult></pre>	<pre><SimpleResult> <SimpleText title="Quote Results"> <SimpleTextItem name="Ticker"> ORCL </SimpleTextItem> <SimpleTextItem name="Price"> 90 3/8 </SimpleTextItem> </SimpleText> <SimpleText title="Headlines"> <SimpleTextItem name = "Headline1"> * Oracle 躍進 </SimpleTextItem> <SimpleTextItem name = "Headline2"> * NASDAQ 高値で終了 </SimpleTextItem> <SimpleTextItem name = "Headline3"> * US 株前進 </SimpleTextItem> </SimpleText> </SimpleResult></pre>

サンプルのアダプタ・クラス

次の2つのコード例は、Adapter がいかに Java で実装されているかを示しています。このコードを参照して、Adapter の動作を理解してください。これらの Adapter を変更して、カスタムのコンテンツ・ソース用に独自の Adapter を作成できます。

- 「Portal-to-Go Adapter の例 1: 株相場およびヘッドラインの XML への変換」に、Adapter クラスが株相場およびヘッドラインを XML に変換する方法を示します。内容を明確にするために、補助メソッドは省略します。
- 「Portal-to-Go Adapter の例 2: ユーザーの名前ごとの対応」は、ユーザーの名前ごとに対応する単純で完全な Adapter 実装です。

Portal-to-Go Adapter の例 1: 株相場およびヘッドラインの XML への変換

複数のメソッドを実装する必要がある1つの Adapter クラスについて考えます。キーとなるメソッドは、invoke です。Master Service は、クライアントが要求を送信するたびに invoke をコールします。この例は、表 7-1 に示す株相場およびヘッドラインのページの XML 結果を生成する Adapter クラスの invoke を示しています。

```
public class StockQuoteAdapter implements Adapter {
    ...
    public Element invoke (ServiceRequest sr)
        throws AdapterException {
        Element result = XML.makeElement("SimpleResult");
        result.setAttribute("title", "Stock Data");

        Element quote = XML.makeElement("SimpleText");
        quote.setAttribute("title", "Quote");

        Element tickerSymbol = XML.makeElement("SimpleTextItem");
        tickerSymbol.setAttribute ("title", "Ticker");
        String t = sr.getArguments().getInputValue("Ticker");
        Text ticker = XML.makeText(t);
        tickerSymbol.appendChild(ticker);
        quote.appendChild(tickerSymbol);

        Element stockPrice = XML.makeElement("SimpleTextItem");
        tickerSymbol.setAttribute ("title", "Price");
        String p = sr.getArguments().getInputValue("Price");
        Text price = XML.makeText(p);
        stockPrice.appendChild(price);
        quote.appendChild(stockPrice);
        result.appendChild(quote)
    }
}
```

```

Element headlines = XML.makeElement("SimpleText");
headlines.setAttribute("title", "Headlines");
Element headline = XML.makeElement("SimpleTextItem");
int i = 0;
String argBase = "headline";
String h = "";
while (h != null) {
    h = sr.getArguments().getInputValue(argBase + i);
    headline.setAttribute("name", argbase + i);
    Text headText = XML.makeText(h);
    headline.appendChild(headText);
    headlines.appendChild(headline);
    i++;
}
result.appendChild(headlines);
return result;
}
...
}

```

Portal-to-Go Adapter の例 2: ユーザーの名前ごとの対応

ユーザーの名前ごとに対応するサービス用の単純な Adapter について考えます。次のような入力があります。

- 初期化パラメータ（対応に使用される文字列）
- 入力パラメータ（ユーザーの名前）

例 2 の Adapter は `invoke` メソッドを使用して、次のパッケージにあるメソッドを使用する Simple Result ドキュメントを構築します。

- `org.w3c.dom.Element`
- `org.w3c.dom.Text`

`invoke` メソッドは、次のタスクを実行します。

1. ルートの結果要素を作成します。
2. SimpleText 要素を作成します。title 属性を設定し、この要素をルート要素に追加します。Simple Result DTD に定義されているように、SimpleTextItem は SimpleText の必須子要素です。
3. 入力パラメータ値を取り出し、結果ドキュメントに追加します。
4. 結果を返します。

Adapter の実装を示します。

```
import org.w3c.dom.Element;
import org.w3c.dom.Text;
import oracle.panama.Argument;
import oracle.panama.Arguments;
import oracle.panama.ServiceRequest;
import oracle.panama.adapter.Adapter;
import oracle.panama.adapter.AdapterDefinition;
import oracle.panama.adapter.AdapterException;
import oracle.panama.adapter.AdapterHelper;

public class HelloAdapter implements Adapter {
    private boolean initialized = false;
    private String greeting = "Hello";
    public static final String GREETING = "greeting";
    public static final String NAME = "name";

    //Adapter がインスタンス化されると、1 回コールされます。
    public void init (Arguments args) throws AdapterException {
        synchronized (this) {
            if (!initialized) {
                initialized = true;
                greeting = args.getInputValue( GREETING );
            }
        }
    }

    public Element invoke (ServiceRequest sr)
        throws AdapterException {
        Element result = XML.makeElement("SimpleResult");
        Element st = XML.makeElement("SimpleText");
        st.setAttribute ("title",
            "Oracle Portal-to-Go Server HelloAdapter Sample");
        result.appendChild (st);
        Element sti = XML.makeElement("SimpleTextItem");
        sti.setAttribute ("name", "message");
        sti.setAttribute ("title", "Portal-to-Go says:");
        st.appendChild (sti);
        //ServiceRequest (sr) には、入力パラメータ (NAME など) が含まれます。
        String name = sr.getArguments().getInputValue (NAME);
        Text txt = XML.makeText( greeting + " " + name + "!!");
        sti.appendChild (txt);
        return result;
    }
    // このメソッドによって、Master Service は、Adapter が使用する
    // 初期化パラメータを決定できます。
```

```

private AdapterDefinition initDef = null;
public AdapterDefinition getInitDefinition() {
    if (initDef == null) {
        synchronized (this) {
            if (initDef == null) {
                initDef = AdapterHelper.createAdapterDefinition();
                initDef.createInit( Argument.SINGLE_LINE,
                                   GREETING,
                                   "Greeting phrase",
                                   null );
            }
        }
    }
    return initDef;
}
// このメソッドによって、adapter&#146;s ランタイム入力パラメータが定義されます。
private AdapterDefinition adpDef = null;
public AdapterDefinition getAdapterDefinition() throws AdapterException {
    if (adpDef == null ) {
        synchronized (this) {
            if (adpDef == null) {
                if (initDef == null)
                    throw new AdapterException ("Adapter is
                                                not properly initialized");
                adpDef = initDef;
                adpDef.createInput( Argument.SINGLE_LINE,
                                   NAME,
                                   "Name to greet",
                                   null );
            }
        }
    }
    return adpDef;
}
}

```

前述の Adapter は、入力パラメータ「Dolly」で起動されると、次の XML 結果を返します。

```

<SimpleResult>
  <SimpleText title="Oracle Portal-to-Go Server Hello Sample">
    <SimpleTextItem name="message" title="Portal-to-Go says:">
      Hello Dolly!
    </SimpleTextItem>
  </SimpleText>
</SimpleResult>

```

ターゲットのマークアップ言語への XML の変換

Portal-to-Go Transformer は、XML ドキュメントをターゲット・デバイス用のマークアップ言語に変換します。情報の表示に SimpleResult などの汎用内部 XML フォーマットを使用して、各クライアント・デバイスの UI 機能を最大限に活用できます。

Transformer は、SimpleResult DTD を使用して取り出した UI 要素をターゲットのフォーマットにマップします。Transformer は、用途に応じて Java または XSLT を使用して実装できます。

■ Java

Java を使用すると、VOX デバイス用の繰返し機能のような、デバイス固有の動作を追加できます（VOX デバイス用の繰返し機能は、画面に書き込むデバイスには必要ありません）。7-24 ページの「[Portal-to-Go Java Transformer の例 1: 他のフォーマットへの Simple Result 要素の変換](#)」を参照してください。

■ XSLT

XSL スタイルシートには、複雑なパターン一致および結果処理論理を含めることができます。これらのスタイルシートには通常、ターゲットのフォーマット用マークアップ・タグなどのリテラル結果要素が含まれます。Portal-to-Go は、デフォルトで XSL スタイルシートを使用します。7-26 ページの「[Portal-to-Go XSL Stylesheet Transformer の例 1: プレーン・テキストへの Simple Result ドキュメントの変換](#)」を参照してください。

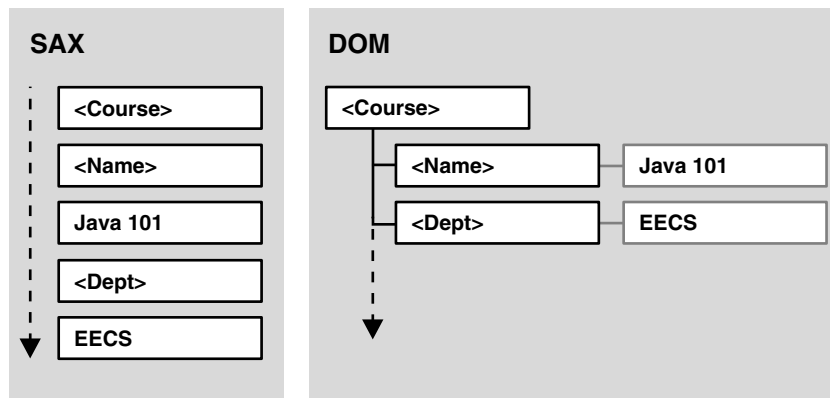
Portal-to-Go: Java Transformer

Java Transformer は、次の 2 つのインタフェースのどちらかを使用して作成できます。

- ツリーベースのドキュメント・オブジェクト・モデル (DOM) を操作する DOM インタフェース
- 解析プロセスのイベントと直接対話する Simple API for XML (SAX) インタフェース

図 7-7 に、これら 2 つのインタフェースを示します。

図 7-7 DOM インタフェースおよび SAX インタフェース



Portal-to-Go には、Simple Result ドキュメントをプレーン・テキストに変換する Java Transformer が含まれています。Transformer は、結果として生成されるドキュメント内にマークアップ・タグを作成しませんが、改行やタブのような単純なテキスト・フォーマットの要素を適用します。

Portal-to-Go Java Transformer の例 1: 他のフォーマットへの Simple Result 要素の変換

この例では、Simple Result 要素を他のフォーマットに変換する方法を簡単に説明します。

```
package oracle.panama.core.xform;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import oracle.panama.PanamaException;
import oracle.panama.core.LogicalDevice;
import oracle.panama.core.Service;
import oracle.panama.Arguments;
import oracle.panama.core.parm.PanamaRequest;
import oracle.panama.core.parm.AbstractRequest;

public class SimpleResultToText implements Transform {
    public SimpleResultToText() {}

    private String format(Element el) {
        if (el == null) {
            return "";
        }
        StringBuffer buf = new StringBuffer();
        String name = el.getTagName();
        if (name != null && name.length() > 0) {
            buf.append(name);
            buf.append(": ");
        }
        buf.append(el.getNodeValue());
        return buf.toString();
    }

    public String transform(Element element, LogicalDevice device)
        throws PanamaException {
        PanamaRequest req = AbstractRequest.getCurrentRequest();
        Service service = req.getService();
        StringBuffer buf =
            new StringBuffer((service == null) ? "" : service.getName());
        NodeList list = element.getElementsByTagName("*");
        Element el;
        String tag;
        boolean newRow = false;
        for (int i = 0; i
            el = (Element)list.item(i);
            tag = el.getTagName();
            if (tag.equals("SimpleRow")) {
                newRow = true;
            }
        }
```

```
        buf.append("\n");
    } else if (tag.equals("SimpleCol")) {
        if (!newRow) {
            buf.append("\t");
        } else {
            newRow = false;
        }
        buf.append(format(el));
    } else if (tag.equals("SimpleText") ||
               tag.equals("SimpleForm") ||
               tag.equals("SimpleMenu")) {
        newRow = true;
        buf.append("\n");
    } else if (tag.equals("SimpleTextItem") ||
               tag.equals("SimpleFormItem") ||
               tag.equals("SimpleMenuItem")) {
        if (!newRow) {
            buf.append("\n");
        } else {
            newRow = false;
        }
        buf.append(format(el));
    }
}
return buf.toString();
}
```

Portal-to-Go: XSL Stylesheet Transformer

XSL スタイルシートは、他の XML ドキュメントの処理ルールを指定する XML ドキュメントです。XSL スタイルシートは、Java Transformer と同様に特定の DTD に固有であり、その DTD に宣言されているすべての要素を処理する必要があります。このスタイルシートは、ソース・ドキュメント内で要素を検出した後、その要素に定義されているルールに従ってその要素の内容をフォーマットします。

Portal-to-Go XSL Stylesheet Transformer の例 1: プレーン・テキストへの Simple Result ドキュメントの変換

この XSL Transformer 例は、Portal-to-Go の初期リポジトリに含まれています。この例は、前述の Java Transformer の XSL パージョンです。この Transformer は、Simple Result ドキュメントをプレーン・テキストに変換します。

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
  <xsl:template match="/">
    <xsl:apply-templates/></xsl:apply-templates>
  </xsl:template>
  <xsl:template match="SimpleTextItem | SimpleFormItem | SimpleMenuItem">
    <xsl:text>
      </xsl:text>
    <xsl:value-of select="."/></xsl:value-of>
  </xsl:template>
  <xsl:template match="SimpleRow">
    <xsl:text></xsl:text>
    <xsl:for-each select="./SimpleCol">
      <xsl:text></xsl:text>
      <xsl:value-of select="."/></xsl:value-of>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

この例では、XSL スタイルシートは次のタスクを実行します。

1. パターン一致用セマンティクスを使用して Simple Result 要素を選択します。たとえば、要素「/」はドキュメントのルート要素に一致します。
2. `apply-templates` を使用してその要素の内容を処理します。
3. ソースの要素ツリー内を降順に移動し、各サブ要素を選択して処理します。`value-of` や `for-each` などの文字命令は、一致する要素の内容を処理します。
 - `value-of` 要素は、要素の実際の内容を抽出します。
 - `for-each` 要素は、反復処理を適用します。

固有のトランスフォーマを必要とする各マークアップ言語

独自のマークアップ言語には、それぞれ固有のトランスフォーマが必要です。株式データの例で、PDA と携帯電話が異なるマークアップ言語（Tiny HTML および WML）を使用します。このため、2 つのトランスフォーマが必要です。これらのトランスフォーマは一度作成されると、Simple Result XML を生成するすべての Adapter からのコンテンツを処理できます。

表 7-2 に、Adapter の SimpleResult XML コードおよび 2 つのトランスフォーマが生成したマークアップ言語を示します。

- PDA 用の Tiny HTML。株相場とヘッドラインの両方をフォーマットおよび表示できます。
- 携帯電話用の WML。株相場のみ表示できます。

表 7-2 固有のトランスフォーマを使用した Adapter の Simple Result XML の変換

Adapter の Simple Result XML	固有のトランスフォーマ
<pre><SimpleResult> <SimpleText title="Quote"> <SimpleTextItem name="Ticker"> ORCL </SimpleTextItem> <SimpleTextItem name="Price"> 90 3/8 </SimpleTextItem> </SimpleText> <SimpleText title="Headlines"> <SimpleTextItem name = "Headline1"> * Oracle 躍進 </SimpleTextItem> <SimpleTextItem name = "Headline2"> * NASDAQ 高値で終了 </SimpleTextItem> <SimpleTextItem name = "Headline3"> * US 株前進 </SimpleTextItem> </SimpleText> </SimpleResult></pre>	<p>PDA 用の Tiny HTML</p> <pre><html> <p> 株価 </p> <p> 株式相場表示機用記号 : ORCL</p> <p> 相場 : 90 3/8</p> <p> ヘッドライン :</p> <p>* Oracle 躍進 </p> <p>* NASDAQ 高値で終了 </p> <p>* US 株前進 </p> </html></pre> <p>携帯電話用の WML</p> <pre><?xml version "1.0"?> <!DOCTYPE WML PUBLIC "-//WAPFORUM/DTD WML 1.0//EN" "http://www.wapforum.org/DTD.wml.xml"> <WML> <CARD NAME="QUOTE_CARD" TITLE="Quote Card"> ORCL 90 3/8 </CARD> </WML></pre>

Portal-To-Go Stylesheet Transformer の例 2: WML1.1 トランスフォーマ用スタイルシートのカスタマイズ

Phone.com ブラウザでの WML 表示

Phone.com ブラウザを使用する場合のナビゲーション・モデルでは、処理の前に [Link] オプションを選択する必要があります。この動作を変更するには、スタイルシートをカスタマイズします。たとえば、次の文を WML1.1 トランスフォーマ用スタイルシートに追加できます。

```
| SimpleForm マッピング
+-->
<xsl:template match="SimpleForm">
<p>
  <xsl:variable name="theTarget">
    <xsl:value-of select="@target"/>
    <xsl:for-each select="SimpleFormItem | SimpleFormSelect">
      <xsl:text>&#38;</xsl:text>
      <xsl:value-of select="@name"/>
      <xsl:text>=</xsl:text>
      <xsl:value-of select="@name"/>
      <xsl:text></xsl:text>
    </xsl:for-each>
  </xsl:variable>
  <xsl:apply-templates/>

  <!-- [Link] の選択確認 -->
  <select>
    <option>
      <onevent type="onpick">
        <go href="{ $theTarget }"/>
      </onevent>
      <xsl:choose>
        <xsl:when test="boolean(@submit)">
          <xsl:value-of select="@submit"/>
        </xsl:when>
        <xsl:otherwise>Submit</xsl:otherwise>
      </xsl:choose>
    </option>
  </select>

  <!-- ここまでが [Link] の選択確認 -->
  <!--
    <a href="{ $theTarget }">
      <xsl:choose>
```

```
<xsl:when test="boolean(@submit)">
  <xsl:value-of select="@submit"/>
</xsl:when>
<xsl:otherwise>Submit</xsl:otherwise>
</xsl:choose>
</a>
<br/>
</p>
-->
</xsl:template>
```

Portal-to-Go の例 1: オンライン・ドラッグストアの販売市場の拡大

あるオンライン・ドラッグストアは、Oracle Portal-to-Go ワイヤレス・インターネット・ソフトウェアを使用して、ハンドヘルド・デバイスを介した利便性およびオンライン・ドラッグストアへの 24 時間アクセスの提供によって、販売市場を拡大しています。

Oracle Portal-to-Go を使用すると、既存のインターネット・サイトをハンドヘルド・ワイヤレス・デバイスに拡張できます。この例では、Portal-to-Go はオンライン・ストアに統合されています。このオンライン・ストアは、Oracle のインターネット・トランスフォーム上に構築されています。このソリューションによって、コンシューマはドラッグストアのすべての製品を、事実上どこからでも購入できます。

Portal-to-Go は、すべてのインターネット・コンテンツ・デバイスを個別に処理するため、PC 用に設計された既存のコンテンツに、インターネットに接続しているすべてのデバイスがアクセスできるようになります。デバイスには、パーソナル・デジタル・アシスタント (PDA)、ワイヤレス・アプリケーション・プロトコル (WAP) 電話、さらにポケットベルも含まれます。

Portal-to-Go の例 2: 銀行サービスの拡大

ある銀行では、顧客向けに携帯電話を使用したオンライン・サービスを開始しました。このサービスには、Oracle ワイヤレス・インターネット・サーバー製品の Oracle Portal-to-Go を使用しています。

銀行の顧客は、WAP 対応電話または標準 GSM 電話のどちらかを使用して経済情報、最寄りの支店を検索する機能、ローン返済計算機、イベント・カレンダー、天気予報などにアクセスできます。

銀行は、このワイヤレス・インターネット・サービスに、銀行取引サービスも追加する予定です。このサービスを追加することで、銀行の新しい WAP プラットフォームでは、顧客が携帯電話を介して、銀行のオンライン情報およびサービスへアクセスできるようになります。

XML および XSQL を使用した表示のカスタマイズ : Flight Finder

この章の内容は次のとおりです。

- [XML Flight Finder のサンプル・アプリケーション : 概要](#)
- [必要なソフトウェア](#)
- [Flight Finder のデータベースへの問合せ : 結果の XML への変換](#)
- [XSQL Servlet を使用した問合せの処理および XML での結果の出力](#)
- [スタイルシートを使用した XML のフォーマット](#)
- [データベースへの XML の書込み](#)
- [Oracle Portal-to-Go](#)

XML Flight Finder のサンプル・アプリケーション：概要

XML Flight Finder は、フライト・データをフェッチし、その結果をクライアント・デバイス（PC、携帯電話、PDA など）用にカスタマイズします。XML Flight Finder は Oracle8i にインストール済で、Oracle XSQL Servlet を利用します。このため、このアプリケーションは SQL 問合せを行い、XML、XSL および XSQL テキスト・ファイルを使用して出力フォーマットを定義できます。Java プログラミングまたはコードのコンパイルは必要ありません。このアプリケーションは、構築、カスタマイズおよびメンテナンスが容易です。

必要なソフトウェア

XML Flight Finder アプリケーションを構築および実行するには、次のソフトウェアが必要です。

- Java 1.2 以上
- Oracle8i リリース 8.1.5 以上
- ご使用のデータベースと互換性があるバージョンの SQL*Plus
- Oracle XSQL Servlet（Web-to-Go personal Web server for Windows NT を含む）。
- Flight Finder 用ファイル
- Web ブラウザ

最良の結果を得るには、Internet Explorer 5 などの XML を処理できるブラウザを使用します。

- （オプションとして）他のデバイス（携帯電話など）をコンピュータ上でシミュレートするソフトウェア
- （オプションとして）Apache または iAS

Windows NT のマシンでは、Web-to-Go があれば Flight Finder を実行できます。また Flight Finder は、Apache または iAS でも実行できます。

Flight Finder の動作

Flight Finder は、ある都市から他の都市へのフライト情報についてデータベースに問い合わせ、エンド・ユーザーのデバイス用にカスタマイズしたフォーマットで結果を返します。Flight Finder は Oracle8i にインストール済で、次の製品およびテクノロジーを使用します。

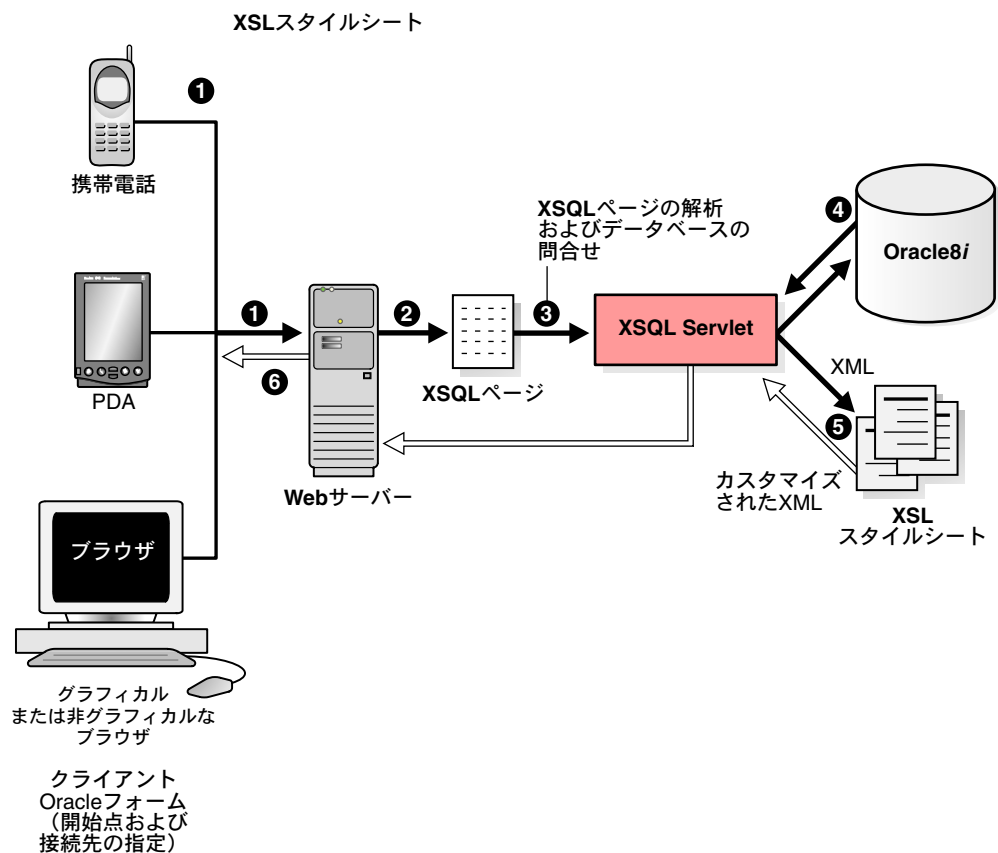
- SQL: ビジネス・データにアクセスする場合の標準ツールです。
- Oracle XSQL Servlet: XSQL ページに定義された問合せを処理します。XSQL ページは、SQL コードを含む XML ドキュメントです。XSQL Servlet は、結果セットを XML として出力します。
- XSLT: XML をターゲット・デバイス用に変換するオープン規格を定義します。

この章では、Flight Finder アプリケーションの実装方法について説明します。これらの方法は、次のタスクを行うすべての Web ベースのアプリケーションで使用できます。

- Web 上のすべてのクライアント・デバイスからの要求の受信
- 複数のデバイスへのデータベース・コンテンツの配信
- 複数のデバイスからの入力のデータベースへの書込み

[図 8-1](#) に、Flight Finder の動作を示します。

図 8-1 XML Flight Finder



1. エンド・ユーザーは、サポートされたクライアント・デバイスを使用してフォームに記入し、開始点および接続先を指定します。フォームのソース・コードには、エンド・ユーザーがフォームを送信したときに実行する XSQL ページを指定しておきます。
2. Web サーバーは、XSQL ページを使用して XSQL Servlet を起動します。
3. XSQL Servlet は、XSQL ページを解析し、データベースに問い合わせます。
4. データベースが問合せ結果を返します。XSQL Servlet はこの結果を XML ドキュメントに変換します。
5. XSQL Servlet は、エンド・ユーザーのクライアント・デバイスに適切な XSL スタイルシートを適用して XML を変換します。
6. Web サーバーが、カスタマイズされたドキュメントをクライアントに戻します。

Oracle8i では、Oracle の XML コンポーネント、およびこのコンポーネントを使用して構築されたアプリケーションを、データベース内で実行できます。データベースのフットプリントを小さくする必要があるデバイスおよびアプリケーションには、Oracle8i Lite を使用して XML データを格納および取出しできます。これらのコンポーネントは、Oracle Internet Application Server 8i などの中間層、またはクライアント上でも実行できます。

Flight Finder のデータベースへの問合せ : 結果の XML への変換

ここでは、Flight Finder がデータベースに問い合わせ、結果セットを XML ドキュメントに変換する方法について説明します。Flight Finder アプリケーションは、XSQL ページおよび XSL スタイルシートで構成されています。

- XSQL ページは問合せを定義します。
- XSL スタイルシートは問合せ結果を他のフォーマットに変換します。

Flight Finder には Java コードがありません。処理操作は、Oracle XSQL Servlet が行います。

Flight Finder は、フライト・データを AIRPORTS および FLIGHTS の 2 つの表に格納します。

- AIRPORTS では、CODE 列が主キーです。
- FLIGHTS では、CODE 列が主キーです。CODE_FROM 列および CODE_TO 列は、AIRPORTS.CODE を参照する外部キーです。

次の SQL コードは、これらの表の構造を示しています（太字の列名は主キーであり、イタリックの列名は外部キーです）。

```
create table airports
(
  code varchar2(3),
  name varchar2(64)
);

create table flights
(
code varchar2(6),
  code_from varchar2(3),
  code_to varchar2(3),
  schedule date,
  status varchar2(1),
  gate varchar2(2)
);
```

XSQL Servlet を使用した問合せの処理および XML での結果の出力

XSQL Servlet は、SQL 問合せを処理し、結果セットを XML で出力します。

XSQL Servlet は Java サブレットとして実装され、入力に XSQL ページを必要とします。これは、埋込み SQL 問合せを含む XML ファイルです。XML Parser for Java および XML SQL Utility for Java を使用して多くの処理を実行します。

例として、fly.xsql のコードを次に示します。これは、XSQL Servlet で解釈するために特別な <xsql> タグを使用した XML です。

flightFinderResult タグは、問合せ内のパラメータに値を割り当てる構造を定義します。このタグは、xsql キーワードを定義するネームスペースを識別し、XSQL Servlet に（事前定義済の）fly という名前のデータベース接続を使用するように指示します。

このコードは、<xsql:query> タグを使用して問合せを定義します。コードは、問合せ文の本体に FROM パラメータおよび TO パラメータを使用して、エンド・ユーザーが選択した都市の名前を格納します。

注意： XSQL ページは、XSLT 構文 {@param} を使用してパラメータを記述します。

図 8-2 に、Flight Finder ブラウザ・フォーム、これを使用した FROM 情報（ロサンゼルス）および TO 情報（サンフランシスコ）の入力方法を示します。

図 8-2 XSQL Servlet を使用した問合せの処理および XML での結果の出力 : Flight Finder ブラウザ・フォームでの FROM および TO の入力



```

<?xml version="1.0"?>
...
<flightFinderResult xmlns:xsql="urn:oracle-xsql" connection="fly" lang="english">
  <xsql:set-stylesheet-param name="lang" value="{@lang}"/>
  <xsql:query tag-case="upper">
    <![CDATA[
      select F.code, F.code_from, A1.name as "depart_airport",
             F.code_to, To_char(F.schedule, 'HH24:MI') as "Sched",
             A2.name as "arrive_airport",
             Decode(F.Status, 'A', 'Available', 'B', 'Full', 'Available')
             as "Stat", F.Gate
      from flights F, airports A1, airports A2
      where to_number(To_Char(F.schedule, 'HH24MI')) >
             to_number(To_Char(sysdate, 'HH24MI')) and
             F.code_from = '{@FROM}' and F.code_to = '{@TO}' and
             F.code_from = A1.code and F.code_to = A2.code
    ]]>
    ...
  </xsql:query>
  ...
</flightFinderResult>

```

次に、次の URL を処理することで XSQL Servlet が戻す XML の一部を示します。この URL の大文字と小文字は区別されます。

<http://localhost:7070/fly.xsql?FROM=LAX&TO=SFO&xml-stylesheet=none>

この URL は、XSQL Servlet を起動し、fly.xsql ファイルを処理して、スタイルシートを適用せずに LAX（ロサンゼルス）から SFO（サンフランシスコ）へのフライト情報を検索するようにサーバーに指示します（データベースからの XML コード（ある場合は、エラー・メッセージも含む）が示されるため、有効なデバッグ方法です）。

結果セット内の行のデータを含む XML ドキュメントが結果として戻されます（次の抜粋には 1 行目のみ示します）。

ROWSET タグおよび ROW タグは、XSQL Servlet が定義します。行セット内の各行のタグ（CODE、CODE_FROM、DEPART_AIRPORT など）は、データベース表内の列名から付けられます。


```
<?xml version="1.0" ?>
  <flightFinderResult lang="english">
    <ROWSET>
      <ROW NUM="1">
        <CODE>OA0307</CODE>
        <CODE_FROM>LAX</CODE_FROM>
        <DEPART_AIRPORT>Los Angeles</DEPART_AIRPORT>
        <CODE_TO>SFO</CODE_TO>
        <SCHED>12:04</SCHED>
        <ARRIVE_AIRPORT>San Francisco</ARRIVE_AIRPORT>
        <STAT>Available</STAT>
        <GATE>05</GATE>
      </ROW>
      ..
    </ROWSET>
    ...
  </flightFinderResult>
```

XML ドキュメントには、データを説明するデータおよびタグが含まれますが、データを表示するためのフォーマット方法に関する情報はありません。これは、最初は制限のように見えますが、実際は機能であり、XML を柔軟にする要素です。データを XML ドキュメントに変換したら、必要に応じた方法でどのようにでもフォーマットできます。

スタイルシートを使用した XML のフォーマット

Flight Finder は XSLT 変換を適用して、XML 結果をエンド・ユーザーのデバイスに適切なフォーマットにします。ここでは、そのプロセスについて説明します。

1つのスタイルシートに対する1つのターゲット・デバイス

Flight Finder は、問合せ結果を表示する XML ドキュメントのフォーマットの変換に XSL スタイルシートを使用します。スタイルシート自体も、他の XML ドキュメントのノードを処理する方法を指定する XML ドキュメントです。処理命令は、テンプレートと呼ばれる構造に定義されています。スタイルシートは、これらのテンプレートを選択したノードに適用することでドキュメントをフォーマットします。

たとえば、前述の XML ドキュメントには、ROWSET、ROW、CODE などの名前を持つノードが含まれています。次の (flyHTMLdefault.xml からの) コードは、スタイルシートが ROWSET 内の各 ROW 用に CODE ノード、DEPART_AIRPORT ノードおよび ARRIVE_AIRPORT ノードを選択し、テンプレートを適用して出力をフォーマットする方法を示しています。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
...
<xsl:template match="/">
<html>
...
  <xsl:for-each select="flightFinderResult/ROWSET/ROW">
    <tr>
      <td><xsl:apply-templates select="CODE"/></td>
      <td><xsl:apply-templates select="DEPART_AIRPORT"/></td>
      <td><xsl:apply-templates select="ARRIVE_AIRPORT"/></td>
      ...
    </tr>
  </xsl:for-each>
  ...
</html>
</xsl:template>
<xsl:template match="CODE">Fly Oracle Airlines <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="DEPART_AIRPORT">Leaving <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="ARRIVE_AIRPORT">
  for <xsl:value-of select="."/>
</xsl:template>
...
</xsl:stylesheet>
```

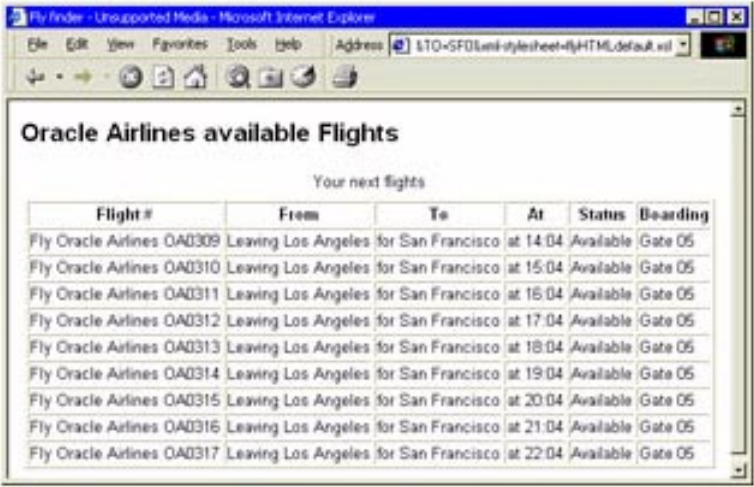
この例のフォーマット方法は単純であり、各ノードのコンテンツに文字列を付加するのみです。たとえば XSLT プロセッサは、CODE ノードを取得すると、そのノードの値に文字列「Fly Oracle Airlines」を付加します。その結果、次のような HTML が生成されます。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
...
<TR>
  <TD>Fly Oracle Airlines OA0309</TD>
  <TD>Leaving Los Angeles</TD>
  <TD>for San Francisco</TD>
  ...
</TR>
...
</HTML>
```

ブラウザで、URL
http://localhost:7070/fly/fly.xsql?FROM=LAX&TO=SFO&xml-stylesheet=flyHTMLdefault.xsl
を入力します。

図 8-3 に、スタイルシートが XML に適用された後でブラウザに表示される結果を示します。

図 8-3 Flight Finder: スタイルシートを使用して XML をフォーマットした後の結果



複数のスタイルシートに対する複数のターゲット・デバイス

XSL スタイルシートは、複数のデバイス、言語およびユーザー・インタフェースに重要です。複数の `<?xml-stylesheet?>` タグを XSQL ページの先頭に含めることができます。これらのタグは、それぞれ `media` 属性および `href` 属性を定義し、ユーザー・エージェントと XSL スタイルシートを対応付けることができます（HTTP 要求には、要求を行ったデバイスを識別するユーザー・エージェント・ヘッダーが含まれます）。`media` 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 / デフォルトとして使用できます。

たとえば、次に示す `fly.xsql` の XML コードには、複数の `<?xml-stylesheet?>` タグが含まれています。これらの中には、スタイルシート `flyVox.xml` を Motorola Voice Browser エージェントにマップするタグ、および `flyPP.xml` スタイルシートを HandHTTP (Palm Pilot) エージェントにマップするタグもあります。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="MSIE 5.0" href="flyHTML.xml"?>
<?xml-stylesheet type="text/xsl" media="Motorola Voice Browser" href="flyVox.xml"?>
<?xml-stylesheet type="text/xsl" media="UP.Browser" href="flyWML.xml"?>
<?xml-stylesheet type="text/xsl" media="HandHTTP" href="flyPP.xml"?>
<?xml-stylesheet type="text/xsl" href="flyHTMLdefault.xml"?>

<flightFinderResult xmlns:xsql="urn:oracle-xsql" connection="fly" lang="english">
  <xsql:stylesheet-param name="lang" value="{@lang}"/>
  <xsql:query tag-case="upper">
    ...
  </xsql:query>
  ...
</flightFinderResult>
```

次の 2 つのコードは、Palm Pilot (`flyPP.xml`) および Voice Browser デバイス (`flyVox.xml`) 用にそれぞれ 1 つの結果セット行をフォーマットする XSLT コードを示しています。

`flyPP.xml` の XSLT コード :

```
...
<xsl:for-each select="flightFinderResult/ROWSET/ROW">
  <tr>
    <td>
      <a>
        <xsl:attribute name="href">
          #<xsl:value-of select="CODE"/>
        </xsl:attribute>
        <b><xsl:value-of select="CODE"/></b>
      </a>
    </td>
    <td><xsl:apply-templates select="SCHED"/></td>
    <td><xsl:apply-templates select="GATE"/></td>
```

```

    </tr>
</xsl:for-each>
...
<xsl:template match="CODE">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="SCHED">
  at <b><xsl:value-of select="."/></b>
</xsl:template>
<xsl:template match="GATE">
  gate <b><xsl:value-of select="."/></b>
</xsl:template>
...

```

flyVox.xsl の XSLT コード :

```

...
<xsl:for-each select="flightFinderResult/ROWSET/ROW">
  <step><xsl:attribute name="name">
step<xsl:value-of select="position()" />
  </xsl:attribute>
  <prompt>
    <xsl:apply-templates select="CODE"/>
    <xsl:apply-templates select="SCHED"/>,
    <xsl:text>Do you take that one?</xsl:text>
  </prompt>
  <input type="OPTIONLIST" name="FLIGHT">
  <xsl:choose>
    <xsl:when test="position() = @NUM">
      <option>
        <xsl:attribute name="next">
          #<xsl:value-of select="CODE"/>
        </xsl:attribute>
        <xsl:text>Yes</xsl:text>
      </option>
      <xsl:if test="position() &lt; last()">
      <option>
        <xsl:attribute name="next">#step<xsl:value-of select="position() + 1"/>
        </xsl:attribute>
        <xsl:text>Next</xsl:text>
      </option>
      </xsl:if>
    <xsl:if test="position() &gt; 1">
      <option>
        <xsl:attribute name="next">#step<xsl:value-of select="position() - 1"/>
        </xsl:attribute>

```

```
<xsl:text>Previous</xsl:text>
</option>
</xsl:if>
</xsl:when>
</xsl:choose>
</input>
</step>
</xsl:for-each>
...
```

出力のローカライズ

ポータル (index.html) から Flight Finder を起動するとき、プロンプトおよびラベルの言語を選択できます。

Flight Finder では、英語、フランス語、スペイン語およびドイツ語がサポートされています。このため Flight Finder は、エンド・ユーザーが選択した言語を識別するパラメータを使用し、このパラメータを HTML から XSQL、そして XSL へと渡します。その後このパラメータは、翻訳されたメッセージのファイルから適切なテキストを選択します。例として、アプリケーションがユーザーの言語のプリファレンス（フランス語）を確認し、その言語のラベルを選択する方法の概要を示します。

1. index.html（ユーザーが、言語を選択するリンクをクリックします）：

```
<a href="http://localhost:7070/xsql/fly/index.xsql?lang=french">Fran is</a>
```

2. index.xsql（XSQL ページは、ユーザーの選択をパラメータに格納します）：

```
<xsql:set-stylesheet-param name="lang" value="{@lang}" />
```

3. flyHTML.xsl（スタイルシートは、言語選択パラメータを使用してメッセージ・ファイルからメッセージを選択します）：

```
<xsl:value-of select="document('messages.xml')/messages/msg[@id=101 and
@lang=$lang]" />
```

4. messages.xml（メッセージ・ファイルには、翻訳済メッセージが格納されています）：

```
<msg id="101" lang="french">Prochains vols sur Oracle Airlines</msg>
```

次に、これらのステップの前後関係を示します。

index.html には、各サポート言語の URL が設定された index.xsql を起動する HREF リンクが表示されます。

..

Web-to-Go の場合

```
<!-- Assumes default install to c:\xsql and Flight Finder files in c:\xsql\fly -->
<ul>
  <li type="disc">
    <a href="http://localhost:7070/xsql/fly/index.xsql">English</a>
  </li>
  <li type="disc">
    <a
href="http://localhost:7070/xsql/fly/index.xsql?lang=french">Fran&ccedil;ais</a>
  </li>
  <li type="disc">
    <a
href="http://localhost:7070/xsql/fly/index.xsql?lang=spanish">Espa&ntilde;ol</a>
  </li>
  <li type="disc">
    <a href="http://localhost:7070/xsql/fly/index.xsql?lang=german">Deutsch</a>
  </li>
</ul>
...
```

次に、ユーザーの選択が URL から抽出され、index.xsql 内のパラメータに挿入されます。この URL に言語が指定されていない場合は、コードの次の行によって、デフォルトで英語が設定されます。XSQL ページは、XSQL Servlet がデータベースに送信する問合せ（ここでは示していません）も定義します。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="indexHTML.xsl"?>
...
<index xmlns:xsql="urn:oracle-xsql" connection="fly" lang="english">
<xsql:set-stylesheet-param name="lang" value="{@lang}"/>
...
</index>
```

データベースが問合せ結果を戻すと、XSQL Servlet は XSLT 変換を適用してこれらの結果をフォーマットします。flyHTML.xsl スタイルシートの次のコードには、メッセージ・ファイル (messages.xml) をオープンし、指定された言語用にメッセージ 101 を選択する行が含まれています。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='1.0'>
  <xsl:output media-type="text/html" method="html"/>
  <xsl:param name="lang" select="@lang"/>
  <xsl:template match="/">
    <html>
...
    <body>
      ...
<!-- Next available flights -->
<xsl:value-of select=
  "document('messages.xml')/messages/msg[@id=101 and @lang=$lang]"/>
    ...
    </body>
  </html>
</xsl:template>
  ...
</xsl:stylesheet>
```

次の XML コードは、messages.xml の一部です。このファイル内のメッセージは、Flight Finder がクライアントに送信する情報 (ラベルまたはプロンプト) を示しています。メッセージは ID 番号によって識別されます。各メッセージは、サポートされたそれぞれの言語に翻訳されます。次のコードは、メッセージ 101 を 4 つの言語に翻訳したものです。ドイツ語バージョンのメッセージに見られるように、翻訳にインターナショナル・キャラクタ・セットのコードを設定できることに注意してください。ご使用のブラウザが、このようなキャラクタを表示するように設定する必要がある場合もあります。たとえば、Internet Explorer では、「表示」>「エンコード」>「西ヨーロッパ言語 (Windows)」を選択します。

```
<?xml version="1.0"?>
<messages>
...
  <msg id="101" lang="english">Oracle Airlines available flights</msg>
  <msg id="101" lang="french">Prochains vols sur Oracle Airlines</msg>
  <msg id="101" lang="spanish">Proximos vuelos sobre Oracle Airlines</msg>
  <msg id="101" lang="german">M&#246;gliche Fl&#252;ge mit Oracle Airlines</msg>
...
</messages>
```


データベースへの XML の書込み

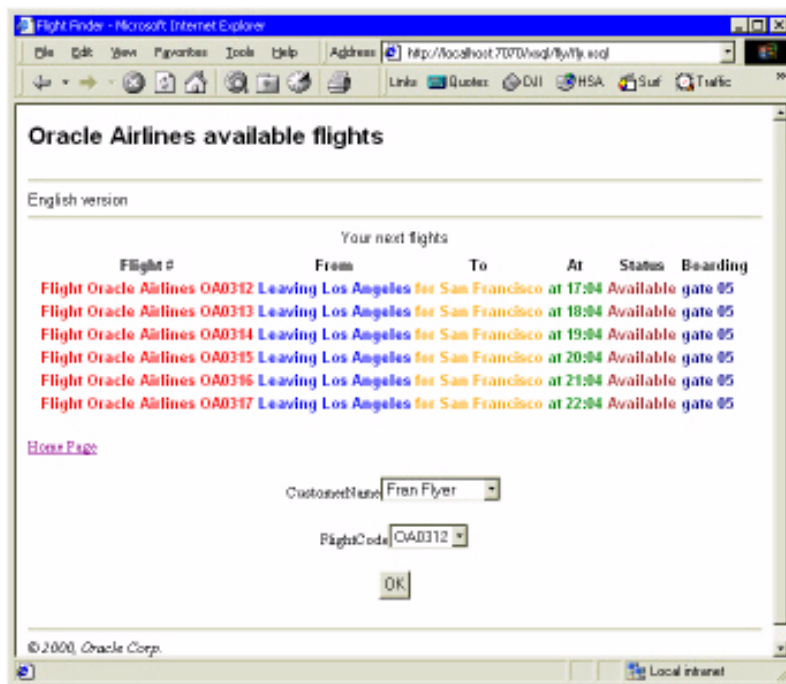
ここでは、Flight Finder がユーザーからの入力を取得し、この入力を XML へ変換して、データベースに書き込む方法について説明します。

ユーザー入力の取得

最初に、ユーザー入力を取得します。

図 8-4 に、ロサンゼルスからサンフランシスコへのフライト情報に関する問合せの結果を表示し、顧客の名前およびフライト・コードのドロップダウン・リストを表示する HTML を示します。ユーザーは名前およびコードを選択し、「OK」ボタンをクリックして顧客のフライトを予約します。アプリケーションは、この情報をデータベースへ書き込みます。アプリケーションのこの部分は、HTML および英語に対してのみ実装されます。

図 8-4 Flight Finder: ロサンゼルスからサンフランシスコへのフライト情報を問い合わせた結果を表示する HTML フォーム



次に、fly.xsql のコードを示します。このコードは、「CustomerName」および「FlightCode」という名前のドロップダウン・リストにデータベースからの値を移入します。<form> タグには、action 属性が含まれます。この属性では、ユーザーがフォームを送信したときに値を処理するために実行するファイルとして bookres.xsql が指定されます。

flyHTML.xsl ファイル（ここには示しません）には、図 8-4 に示すようなフォームのフォーマットを変換する XSLT の命令が記述されています。

...

```
<form action="bookres.xsql" method="post">
  <field name="CustomerName">
    <xsql:query rowset-element="dropDownList"
      row-element="listElem">
      <![CDATA[
        select unique name as "listItem"
          from customers
          order by name
        ]]>
    </xsql:query>
  </field>
  <field name="FlightCode">
    <xsql:query rowset-element="dropDownList"
      row-element="listElem">
      <![CDATA[
        select F.code as "listItem",
          F.code as "itemId",
          A1.name as "depart_airport",
          A2.name as "arrive_airport"
          from flights F,
          airports A1,
          airports A2
          where to_number('To_Char(F.schedule, 'HH24MI')') >
            to_number('To_Char(sysdate, 'HH24MI')') and
            F.code_from = '{@FROM}' and
            F.code_to = '{@TO}' and
            F.code_from = A1.code and
            F.code_to = A2.code
        ]]>
    </xsql:query>
  </field>
  <sendRequest type="button" label="OK"/>
</form>
```

...

ユーザーから取得した値のコード・パラメータへの割当て

ユーザーから値を取得したら、次にそれらの値をコード内のパラメータに割り当てます。次のコードは、bookres.xsql の一部です。

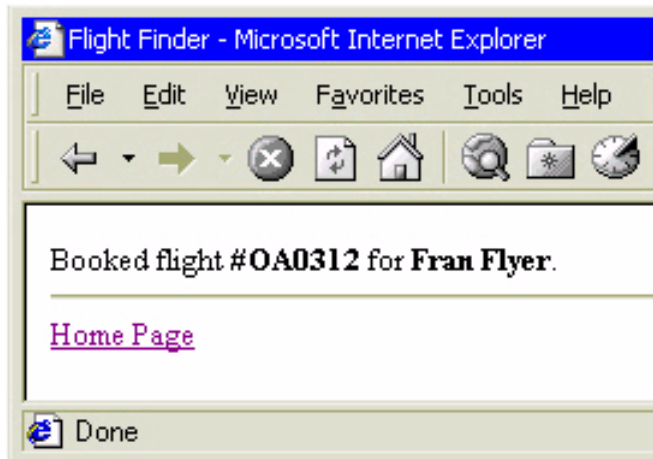
このコードは、ユーザーの選択を CustomerName および FlightCode という名前のパラメータに格納し、値を XSLT スタイルシートに渡すために cust および code という名前のパラメータを定義します。また、<xsql:dml> タグを使用して、行を CUSTOMERS 表に挿入する SQL 文を定義します。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="bookresHTML.xsl"?>
<?xml-stylesheet type="text/xsl" media="MSIE 5.0" href="bookresHTML.xsl"?>
  <bookFlight xmlns:xsql="urn:oracle-xsql" connection="fly">
    <xsql:set-stylesheet-param name="cust" value="{@CustomerName}"/>
    <xsql:set-stylesheet-param name="code" value="{@FlightCode}"/>
    <xsql:dml>
      <![CDATA[
        insert into customers values
          ('{@CustomerName}', tripseq.NEXTVAL, '{@FlightCode}')
      ]]>
    </xsql:dml>
    ...
  </bookFlight>
```

処理成功のユーザーへの通知

最後に、処理が正常に実行されたかどうかをユーザーに通知します。ここでは、[図 8-5](#) に示すとおり、フライトが予約されたかどうかを通知します。

図 8-5 Flight Finder: ユーザーへのフライト予約完了の通知



次のコードは、bookresHTML.xsl の一部です。

このコードは、cust および code という名前のパラメータを宣言し、bookres.xsql からこれらのパラメータに渡された値を格納します。その後、このコードはこれらのパラメータを使用して、ユーザーにメッセージを表示します。このようなパラメータを使用するための XSLT 構文は、\$param です。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output media-type="text/html"/>
  <xsl:param name="cust"/>
  <xsl:param name="code"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Flight Finder</title>
      </head>
      <body>
        Booked flight #<b><xsl:value-of select="$code"/></b>
        for <b><xsl:value-of select='$cust'/></b>.
        <hr/>
        <xsl:apply-templates select="bookFlight/returnHome"/>
      </body>
    </html>
  </xsl:template>
  ...
</xsl:stylesheet>
```

Oracle Portal-to-Go

XSQL および XSL コードを記述せずに、Oracle Portal-to-Go を使用できます。

Oracle のインターネット・プラットフォームのコンポーネントである Portal-to-Go は、すべての対応デバイスに Web コンテンツを配信するための機能をすべて備えています。Portal-to-Go は、既存のコンテンツをデバイス固有のフォーマットに変換し、ポータル・インタフェースをエンド・ユーザーに提供します。このポータルは、Oracle JDeveloper で開発できます。

Portal-to-Go は、XML を使用してコンテンツの収集をコンテンツ配信から分離します。

Portal-to-Go ポータルには、次のコンポーネントがあります。

- モバイル・デバイスにデータを配信するサービス
- HTML および RDBMS のコンテンツを XML に変換するアダプタ
- HTML、WML、Tiny HTML、音声マークアップ言語（VoxML）などの、適切なマークアップ言語に XML を変換するトランスフォーマ

参照： [第7章「XML によるデータ表示のパーソナライズ : Portal-to-Go」](#)
も参照してください。

第 IV 部

XML を使用したデータ交換

第 IV 部では、Oracle Advanced Queuing について説明し、B2B メッセージ機能アプリケーションで使用方法について説明します。B2B および B2C アプリケーションでの XML ベースのデータ交換を実装する方法を示すいくつかの例が含まれます。

第 IV 部に含まれる章は、次のとおりです。

- [第 9 章「Oracle Advanced Queuing を使用した XML データの交換」](#)
- [第 10 章「B2B: iProcurement による XML を使用した複数のカタログ製品のユーザー提供」](#)
- [第 11 章「XML メッセージ機能を使用した Phone Number Portability」](#)

Oracle Advanced Queuing を使用した XML データの交換

この章の内容は次のとおりです。

- [AQ の概要](#)
- [AQ と XML の相互補完](#)
- [AQ の例 1 \(PL/SQL\) : AQ メッセージの CLOB としての XML メッセージ](#)
- [AQ の例 2 \(Java\) : JMS \(パブリッシュ / サブスクライブ\) を使用した XML メッセージの処理](#)
- [FAQ: XML およびアドバンスト・キューイング](#)

AQ の概要

Oracle Advanced Queuing (AQ) には、次のデータベース統合化メッセージ・キューイング機能があります。

- メッセージを使用する 2 つ以上のアプリケーションの非同期通信を可能にし、その管理を行います。
- Point-to-Point 通信モデルおよびパブリッシュ / サブスクライブ通信モデルをサポートします。

Oracle8i データベースとメッセージ・キューイングの統合によって、Oracle8i の整合性、信頼性、リカバリ可能性、スケーラビリティ、パフォーマンスおよびセキュリティ機能がメッセージ・キューイングに提供されます。また、Oracle8i との統合によって、メッセージ・フローからのインテリジェント機能の抽出が容易になります。

AQ と XML の相互補完

XML の主要な利点は、アプリケーション間のデータ交換の共通フォーマットとして使用できることです。XML は自己記述型です。そのため、アプリケーションは、データを受信するアプリケーションに関して事前定義された知識がなくてもそのデータを共有できます。XML は、2 つ以上のアプリケーション間で通信されるメッセージを定義するために使用され、AQ はその通信を可能および管理するために使用されます。

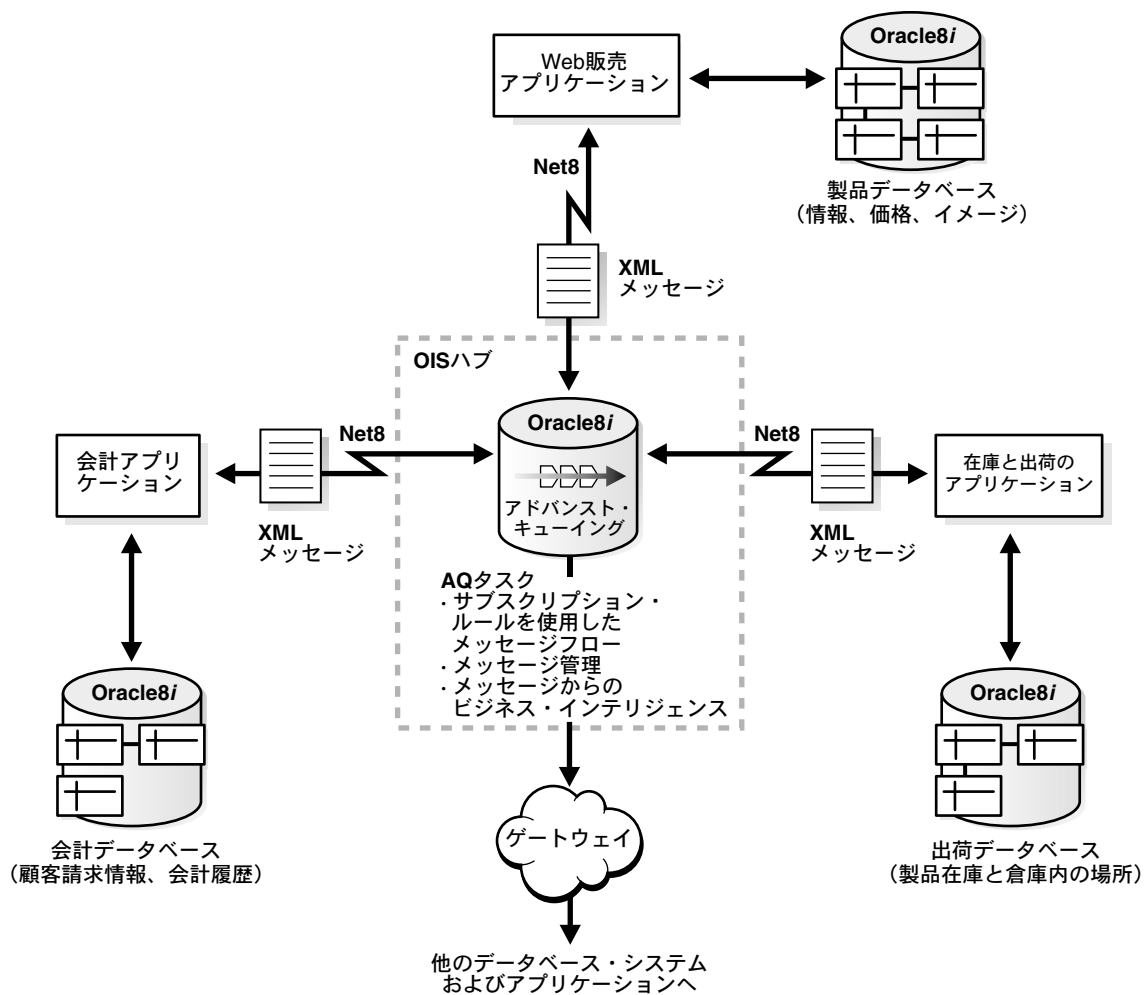
図 9-1 は、AQ を使用して 3 つのアプリケーションと通信する Oracle8i データベースを示します（メッセージ・ペイロードとして XML を使用します）。この使用例において AQ が実行する主なタスクは次の 3 つです。

- サブスクリプション・ルールを使用したメッセージ・フロー
- メッセージ管理
- メッセージからのビジネス・インテリジェンスの抽出

これは、XML メッセージが、AQ を介して組織内のアプリケーション間で非同期に渡される企業内業務での使用例です。このような使用例には、販売注文の遂行およびサプライチェーンの管理などがあります。

同様の使用例は、複数の統合ハブがインターネットのバックプレーンを介して通信する企業間業務処理にも適用できます。企業間業務での使用例は、旅行の予約、メーカーとサプライヤ間の調整、銀行間での資金の移動、保険請求の清算などです。Oracle は、これを企業アプリケーション統合製品で使用します。XML メッセージは、アプリケーションから AQ ハブ（図 9-1 では OIS ハブ）へ送信されます。このサーバーは、メッセージを必要とするすべてのアプリケーションに対してメッセージ・サーバーとして機能します。このハブ・アンド・スポーク・アーキテクチャを介して、XML メッセージを、疎結合された複数の受信側アプリケーションへ非同期に送信できます。

図 9-1 アドバンスト・キューイングおよびXML メッセージ・ペイロード



アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にする AQ

今日の企業が直面している重大な問題の 1 つは、アプリケーションの統合です。アプリケーションの統合には、複雑な業務トランザクションを実行するための、複数の部門アプリケーションの協調、調整および同期化が伴います。

アドバンスト・キューイングは、アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にします。このアーキテクチャを使用すると、統合化ソリューションの管理、構成およびビジネス・ニーズの変化に伴う変更が容易になります。

監査、追跡およびマイニング用に保存できるメッセージ

AQ が提供するメッセージ管理では、異なるアプリケーション間でのメッセージ・フローを管理するだけでなく、将来のビジネス・インテリジェンスの監査、追跡および抽出のためにメッセージを保存することもできます。

SQL ビューを使用したメッセージ内容の表示

AQ は、メッセージを参照するための SQL ビューも提供します。これらの SQL ビューは、システムにおける過去、現在および将来の動向を分析するために使用できます。

AQ を使用するメリット

AQ を使用すると、異なるアプリケーション間での通信を柔軟に構成できます。

XML メッセージは、AQ を使用して次の方法で通信できます。

- AQ ペイロードの XML メッセージが、VARCHAR または CLOB で格納されます。または、圧縮されて BLOB で格納されます。
- JMS を使用して、TextMessage または BytesMessage として格納されます。オブジェクトとして処理され、ObjectMessage または SQL ADT メッセージとして格納されることもできます。

AQ の例 1 (PL/SQL) : AQ メッセージの CLOB としての XML メッセージ

この例では、AQ メッセージの CLOB に格納されている XML メッセージの処理方法を説明します。この例は、PL/SQL で示しています。

参照：『Oracle8i 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』も参照してください。

AQ 環境の設定

AQ 環境を設定するには、次の操作を実行します。

- PL/SQL パッケージの DBMS_AQADM および DBMS_AQ に EXECUTE 権限を付与します。
- 次の表を作成します。
 - スキーマ
 - キュー・テーブル
 - キュー

AQ の例 1: 実行されるタスク

この AQ の例では、次のタスクが実行されます。

1. CLOB の XML メッセージを格納するための `xml_payload_type` を作成します。
2. `xml_payload_type` メッセージを保持するために AQ キュー・テーブルを作成します。
3. AQ キューを作成します。
4. サブスクライバを追加します。
5. エンキューおよびデキューのためのキューを開始します。

メッセージのエンキュー

1. 空の CLOB をエンキューします。
2. エンキューされたメッセージ ID を使用している CLOB ロケータを選択します。
3. DBMS_LOB パッケージを使用して、CLOB を移入します。

メッセージのデキュー

1. メッセージをデキューします。
2. DBMS_LOB パッケージを使用して、メッセージを読み込みます。

AQ の例 1: PL/SQL コード

次に例を示します。

```
CONNECT scott/tiger; -- DBMS_AQADM パッケージおよび DBMS_AQ パッケージに対する実行権限が
-- 必要です。

-- XML メッセージを格納する Oracle オブジェクト型を作成します。

CREATE OR REPLACE TYPE xml_payload_type AS OBJECT (
-- 必要に応じて、メッセージを解析して属性のいくつかを抽出し、AQ キューの内容ベースのルーティン
グを実行します。
xml_message CLOB;
);

-- XML ペイロードを持つ AQ キュー・テーブルを作成します。
BEGIN
dbms_aqadm.create_queue_table(
queue_table => 'xmlmsg_queueetable',
sort_list => 'priority,enq time',
comment => 'demonstrate XML message in an AQ queue',
multiple_consumers => TRUE,
queue_payload_type => 'xml_payload_type',
compatible => '8.1');
END;
/

-- XML メッセージ用の AQ キューを作成します。
BEGIN
dbms_aqadm.create_queue (
queue_name => 'xmlmsg_queue',
queue_table => 'xmlmsg_queueetable');
END;
/

-- サブスクライバを XML メッセージ・キューに追加します。
BEGIN
dbms_aqadm.add_subscriber(
queue_name => 'xmlmsg_queue',
subscriber => 'xml_subscriber');
END;
/

-- エンキューおよびデキューのためのキューを開始します。
BEGIN
dbms_aqadm.start_queue('xmlmsg_queue');
END;
```

```
/

-- XML メッセージをエンキューします。
DECLARE
  enqopt dbms_aq.enqueue_options_t;
  msgprop dbms_aq.message_properties_t;
  enq_msgid RAW(40);
  sales_order xml_payload_type;
  order_loc CLOB;
  XML_msg VARCHAR2(200);
  msgsize Number;
BEGIN
  sales_order := xml_payload_type (empty_clob());
  dbms_aq.enqueue(
    queue_name          => 'xmlmsg_queue', -- IN
    enqueue_options     => enqopt, -- IN
    message_properties  => msgprop, -- IN
    payload             => sales_order, -- IN
    msgid               => enq_msgid); -- OUT
  SELECT t.user_data.xml_message INTO order_loc
    FROM xmlmsg_queue_table t
   where t.msgid = enq_msgid;

  -- XML メッセージを LOB に挿入します。
  XML_msg := '<xml> </xml>';
  dbms_lob.write(order_loc, 12, 1, XML_msg);

  COMMIT;
END;
/

-- XML メッセージをデキューします。
declare
  dequeue_options dbms_aq.dequeue_options_t;
  message_properties dbms_aq.message_properties_t;
  message_handle RAW(16);
  message xml_payload_type;
  buffer varchar2(100);
  msglen number;
begin
  dequeue_options.consumer_name := 'xml_subscriber';
  dequeue_options.wait := dbms_aq.NO_WAIT;
  dbms_aq.dequeue(
    queue_name          => 'xmlmsg_queue',
    dequeue_options     => dequeue_options,
    message_properties  => message_properties,
```

```
        payload          => message,  
        msgid            => message_handle);  
    commit;  
    msglen := dbms_lob.getlength(message.xml_message);  
    dbms_lob.read(message.xml_message, msglen, 1,  
buffer);  
    dbms_output.put_line(buffer);  
end;  
/
```

AQ の例 2 (Java) : JMS (パブリッシュ / サブスクライブ) を使用した XML メッセージの処理

XML メッセージは、Java メッセージ機能標準である JMS を使用して AQ キューからエンキューおよびデキューできます。例 2 では、XML メッセージを `JMSTextMessage` としてパブリッシュおよびサブスクライブする方法を示します。これ以外の例については、『Oracle8i 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。

AQ の例 2: JMS を使用した XML メッセージの処理 - 実行されるタスク

AQ JMS 環境の設定

AQ JMS 環境の設定の詳細は、『Oracle8i アプリケーション開発者ガイド アドバンスト・キューイング』の第 13 章「JMS 管理インタフェース - 基本操作」を参照してください。

JMS を使用した XML メッセージのパブリッシュ (エンキュー)

次に、JMS を使用して XML メッセージをパブリッシュするために例 2 で実行されるタスクを示します。

1. あるサブジェクトのトピックの名前および場所を取得します。
2. 必要なトピックがある JMS プロバイダの `ConnectionFactory` を取得します。
3. `ConnectionFactory` を使用して JMS プロバイダへ接続します。
4. JMS セッションを作成します。
5. JMS セッションを使用してトピック・オブジェクトを取得します。
6. パブリッシュされるメッセージを作成します。
7. メッセージを送信するための `TopicPublisher` を作成します。
8. メッセージをトピックへパブリッシュします。

JMS を使用した XML メッセージの受信 (デキュー)

次に、JMS を使用して XML メッセージを受信 (サブスクライブ) するために例 2 で実行されるタスクを示します。

1. あるサブジェクトのトピックの名前および場所を取得します。
2. 必要なトピックがある JMS プロバイダの `ConnectionFactory` を取得します。
3. `ConnectionFactory` を使用して JMS プロバイダへ接続します。
4. JMS セッションを作成します。
5. JMS セッションを使用してトピック・オブジェクトを取得します。
6. 必要なメッセージを受信するための `TopicSubscriber` を作成します。
7. ブロッキング `receive` コールを使用するかまたは `MessageListener` を登録して、メッセージが受信されるまで待機します。

AQ の例 2: JMS を使用した XML メッセージの処理 - Java コード

次に、JMS を使用して XML メッセージを処理する Java コードを示します。

```
public void publishXMLMessage (String host, String ora_sid, int port, String driver)
throws JMS Exception
{
    TopicConnectionFactory tc_fact;
    TopicConnection        t_conn;
    TopicSession            jms_sess;
    Topic                   xmlmsg_topic;
    TextMessage             xmlmsg;
    TopicPublisher          xmlmsg_publisher;

    //ConnectionFactory を作成します。
    tc_fact = AQjmsFactory.createTopicConnectionFactory(
        host, ora_sid, port, driver);
    // トピック接続を作成します。
    t_conn = tc_fact.createTopicConnection ("scott", "tiger");
    //JMS セッションを作成します。
    jms_sess = t_conn.createTopicSession(true, 0);

    // トピック・オブジェクトを取得します。
    xmlmsg_topic = ((AQjmsSession)jms_sess).getTopic("scott", "xmlmsg_topic");
    // トピック・パブリッシャを作成します。
    xmlmsg_publisher = jms_sess.createPublisher(null);
    //XML テキスト・メッセージを作成します。
    xmlmsg = jms_sess.createTextMessage();
    // メッセージをパブリッシュします。
    xmlmsg_publisher.publish(xmlmsg_topic, xmlmsg);
}
```

```
jms_sess.commit();
}

public void receiveXMLMessage (String host, String ora_sid, int port, String driver)
throws JMS Exception
{
    TopicConnectionFactorytc_fact;
    TopicConnectiont_conn;
    TopicSessionjms_sess;
    Topicxmlmsg_topic;
    TextMessagexmlmsg;
    TopicSubscriberxmlmsg_subscriber;

    //ConnectionFactory を作成します。
    tc_fact = AQjmsFactory.createTopicConnectionFactory(
                                host, ora_sid, port, driver);
    // トピック接続を作成します。
    t_conn = tc_fact.createTopicConnection ("scott", "tiger");
    //JMS セッションを作成します。
    jms_sess = t_conn.createTopicSession(true, 0);

    // トピック・オブジェクトを取得します。
    xmlmsg_topic = ((AQjmsSession)jms_sess).getTopic("scott", "xmlmsg_topic");
    // トピック・サブスクライバを作成します。
    xmlmsg_subscriber = jms_sess.createDurableSubscriber(xmlmsg_topic, "XmlMsg_
Subscriber");
    //XML テキスト・メッセージを作成します。
    xmlmsg =(TextMessage)xmlmsg_subscriber.receive();
    // テキスト・メッセージを処理します。

    jms_sess.commit();
}
```

FAQ: XML およびアドバンスト・キューイング

複数のフォーマットのメッセージ: オブジェクト型の作成および単一メッセージとしての格納

質問

Oracle Advanced Queuing を使用して、XML ドキュメントをあるビジネス分野から別のビジネス分野へ変更する予定です。受信または送信される各メッセージには、XML ヘッダー、XML アタッチメント (XML データ・ストリーム)、DTD および PDF ファイルが含まれます。これらのすべての情報を、画像ファイルも含めてデータベース表 (この場合はキュー・テーブル) に格納する必要があります。

このメッセージを、1つのレコードまたは1つのピースとして Oracle キュー・テーブルにエンキューできますか? または、このメッセージを複数のレコード (XML データ・ストリームに対する CLOB 型としての1つのレコード、PDF ファイルに対する RAW 型としての1つのレコードなど) としてエンキューしてから、これらのレコードの相関を指定する必要がありますか? また、メッセージをデキューしたことを確認する必要もあります。

回答

次の方法で可能です。

- CLOB、RAW などの属性を持つオブジェクト型を定義し、これを単一メッセージとして格納します。
- AQ メッセージ・グループ化機能を使用して、該当するメッセージを複数のメッセージに格納します。ただし、メッセージのプロパティは1つのグループに対応付けられます。メッセージ・グループ化機能を使用するには、すべてのメッセージのペイロード型が同じである必要があります。

質問

最初にペイロード型を CLOB として指定し、すべてのピース、XML メッセージ・データ・ストリーム、DTD、PDF などを単一のメッセージ・ペイロードとしてエンキューしてキュー・テーブルに格納するのですか? その場合、このメッセージをデキューするときに、どのようにしてこの単一メッセージを個々に分割するのですか?

回答

そうではありません。まず、次のとおりオブジェクト型を作成します。

```
CREATE TYPE mypayload_type as OBJECT (xmlDataStream CLOB, dtd CLOB, pdf BLOB);
```

次に、このオブジェクト型を単一のメッセージとして格納します。

メッセージがエンキューされた後の新しい受信者の追加

質問

メッセージ割当てをサポートするためにキュー・テーブルを使用します。たとえば、他のビジネス分野は、メッセージを Oracle に送信するときに、これらのメッセージを処理するために誰が割り当てられるかは認識していません。ただし、そのメッセージが人事部宛てであることは認識しています。そのため、すべてのメッセージは人事部管理者へ送信されます。

この時点で、メッセージはキュー・テーブルにエンキューされています。このメッセージの受信者は人事部管理者のみであり、その他のすべての人事部社員はこのキューのサブスクライバとして事前定義されています。人事部管理者は新しい受信者であるその他の社員を、キュー・テーブル内の既存のメッセージに関する `message_properties.recipient_list` に追加することができますか？

メッセージがエンキューされるときには複数のコンシューマ（受信者）は存在しませんが、メッセージがキュー・テーブルにエンキューされた後に、古い受信者を置き換えるか、または新しい受信者を追加する必要があります。この新しい受信者が、新しいメッセージをデキューします。このようなことは可能ですか？または、メッセージを古い受信者から削除してから、同じメッセージ内容を新しい受信者にエンキューする必要がありますか？

回答

メッセージがエンキューされた後に受信者のリストを変更することはできません。受信者のリストを指定しない場合、サブスクライバはそのキューをサブスクライブし、メッセージをデキューできます。

この場合、新しい受信者がそのキューのサブスクライバである必要があります。それ以外の場合は、メッセージをデキューし、そのメッセージを新しい受信者が再度エンキューする必要があります。

XML コンテンツを持つ JMS クライアントの AQ からの取出しおよび解析

質問

メッセージ (XML コンテンツを持つ JMS クライアント) を AQ キューから解析し、ODS (Operational Data Store) にある表およびフィールドを更新するツールが必要です。XML ドキュメントを取り出して解析し、特定のフィールドをデータベース表および列にマップする必要があります。

interMedia Text で可能ですか？

回答

最も簡単な方法は、Oracle8i 内で AQ と同時に、Oracle XML Parser for Java および Java ストアド・プロシージャを使用することです。

質問

カスタム・ソリューションを使用すると XML SQL Utility を使用できます。主な目的はサブライチェーンです。ある特定の XML タグ値の問合せに基づいて、AQ エンキュー / デキューの回数や JMS ヘッダー情報などのメタデータ情報を取得する必要があります。単純に CLOB に XML を格納して、*interMedia Text* を使用して問合せを発行できますか？

回答

質問ではメッセージの解析と言及されていますが、質問の意図がよくわかりません。また、通常の表にメッセージ・メタデータを置くともあります。

- XML を CLOB で格納する場合、*interMedia Text* を使用して検索できますが、この場合は、ある基準に一致する特定のメッセージのみを検索できます。
- メタデータ上で集計操作を行う必要がある場合は、既存の関連ツールからメタデータを参照するか、または通常の SQL 述語をメタデータに使用します。この場合、CLOB に XML で格納するのみでは十分ではありません。

interMedia Text XML 検索と、抽出された列などの、重複したメタデータ記憶域を組み合わせ、通常の SQL 述語と *interMedia Text* CONTAINS() 句を組み合わせる SQL 文を使用すると、両方の最適な機能を利用できます。

エンキューが成功したかどうかの確認

質問

エンキューが成功したかどうかを確認する方法はありますか？AQ を Oracle8i に配置しています。エンキュー文が PL/SQL プロシージャで正常に実行されても、実際にはキューが開始していない場合があります。また、エンキュー後に、どのエラー・メッセージも戻されません。メッセージが失われてしまいます。

このような状況を解決する方法はありますか？

回答

次の処理を実行してみてください。

1. キューを作成した後、`dbms_aqadm.start_queue` コマンドを発行します。
2. メッセージをエンキューした後、`COMMIT` を発行します。

B2B: iProcurement による XML を使用した 複数のカタログ製品のユーザー提供

この章の内容は次のとおりです。

- [Oracle Internet Procurement \(iProcurement\) の概要](#)
- [iProcurement: XML Parser for Java](#)
- [バイヤーが提供するカタログ](#)
- [DTD 管理情報: <ADMIN>](#)
- [DTD スキーマ情報: <SCHEMA>](#)
- [DTD: 品目情報](#)
- [サプライヤーが提供するカタログおよび市場](#)
- [データ要素の定義](#)
- [注文明細の XML 定義](#)
- [HTML 仕様](#)
- [ユーザー認証](#)

Oracle Internet Procurement (iProcurement) の概要

Oracle iProcurement は、購入可能な製品やサービスの検索および注文を容易にする、Web ベースのカatalog・コンテンツ変換アプリケーションです。Oracle iProcurement を使用すると、商品の調達から支払いまでの販売サイクル全体を簡単に自動化できます。また、iProcurement には、エクスプレス・チェックアウトおよびパワー・チェックアウトのオプションを持つ Web ショッピング・インタフェース、およびサード・パーティ製の EPR システムのトランザクション用の組込み Internet Procurement Connector があります。

iProcurement は、Catalog・コンテンツを管理する複数の方法をサポートしています。ユーザーは、次のどの組合せでも選択できます。

- パイヤーが提供するコンテンツ
- サプライヤが提供するコンテンツ
- サード・パーティが提供する市場

これらのコンテンツ管理の方法は、Web ベースのデータ（コンテンツ）変換テクノロジーのために XML を使用します。この章では、これらのコンテンツ管理方法および関連する XML ドキュメントについてそれぞれ説明します。また、例をいくつか示します。

様々なサプライヤによる総合カATALOG表へのカATALOGのロード

iProcurement は XML を使用して、様々なサプライヤまたはカATALOG・サービス・プロバイダから受信したカATALOGを iProcurement データベースにある総合カATALOG表にロードします。図 10-1 を参照してください。

また XML は、外部のカATALOG提供企業と通信する場合にセキュリティ・オブジェクトまたは認証オブジェクトを変換します。

- **問題:** 複数のサプライヤの商品およびサービスのカATALOGを、Web を介して透過的に提供するカATALOGを構築します。
- **ビジネス・ソリューション:** iProcurement を、Oracle Exchange、Oracle の XML テクノロジー、PL/SQL プログラム、XML インタフェースを含む Oracle アプリケーションとともに使用します。
- **使用する Oracle の XML コンポーネント:** XML Parser for Java

Oracle Internet Procurement ソリューション

Oracle Internet Procurement ソリューションでは、Oracle Internet Procurement、Oracle Supplier Network および Oracle Exchange を組み合わせた機能を使用します。企業は、様々な種類の商品およびサービスを最適な価格で購入し、より適切な戦略的決定を行い、総収益を向上させることができます。このソリューションによって、注文および受領処理を分散させ、手配、承認処理、支払いなどの調達機能を自動化および集中化できます。

Oracle Exchange は、Oracle Supplier Network が提供するコンテンツおよび関連サービスを使用して、企業がどのような方法でも商品およびサービスを売買できるオープンな B2B オンライン・マーケットプレイスです。Oracle Exchange には業界の規模にかかわらず、すべての企業がアクセスでき、Oracle ソフトウェアは必要ありません。

Internet Procurement および関連製品の詳細

iProcurement の詳細は、次の Web サイトを参照してください。

<http://www.oracle.com/products/>

<http://www.oracle.com/applications/internetprocurement/index.html>

Oracle Exchange の詳細は、次のサイトを参照してください。

<http://www.oracle.com/applications/exchange/index.html>

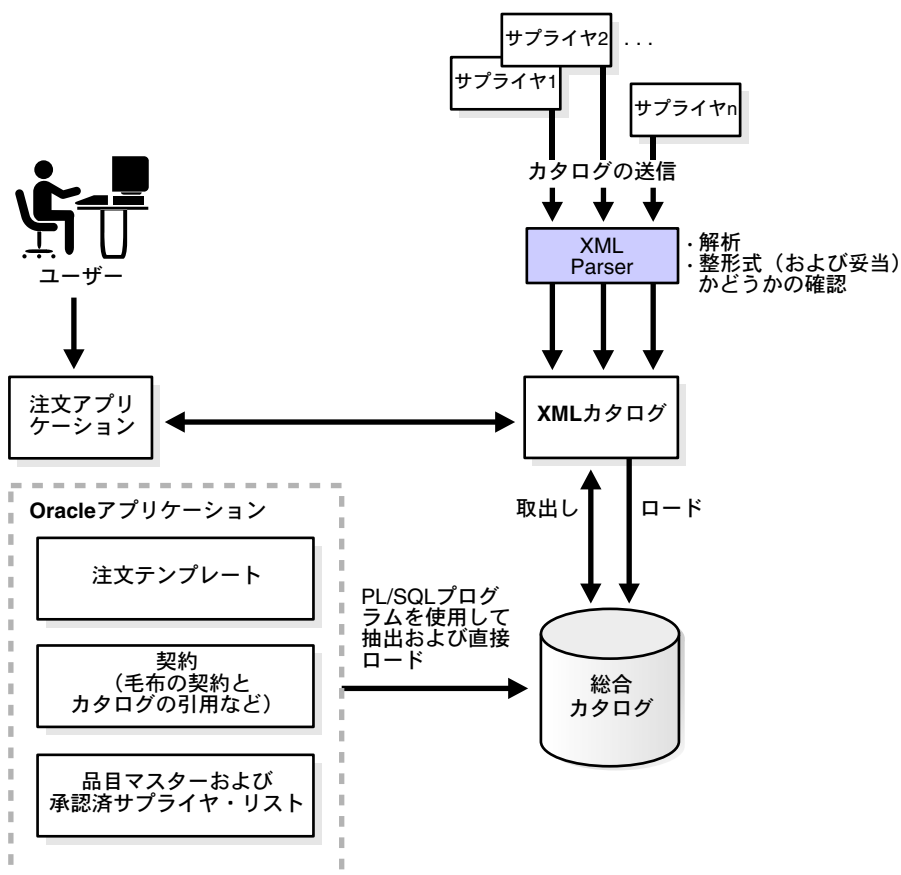
バイヤーが提供するコンテンツ

バイヤーが提供するコンテンツ管理を使用すると、サプライヤを最初を選択せずに、1 つの総合カタログにある製品またはサービスを検索することによって製品を注文できます。総合カタログのコンテンツは、次のいずれかのソースから受け取ります。

- Oracle アプリケーションから、次のようなコンテンツが直接ロードされます。
 - 品目マスター
 - 承認済サプライヤのリスト
 - 注文テンプレート
 - 契約情報
- この章で説明している XML インタフェースを介して、すべてのサード・パーティ・コンテンツ・プロバイダからロードします。たとえば、あるオラクル・パートナーは、XML インタフェースを使用して高品質な非公開電子サプライヤ・カタログを発行しています。

図 10-1 に、バイヤー提供のカタログ変換処理における XML の使用を示します。

図 10-1 バイヤーが提供するコンテンツの管理



Oracle アプリケーション・データは、PL/SQL コンカレント・プログラムによって抽出され、総合カタログへ直接ロードされます。そのため、この場合は XML ドキュメントを生成および解析する必要がありません。

サプライヤが提供するカタログおよび市場

ユーザーは、注文書の作成中に外部のサード・パーティのカタログにアクセスして品目を選択できます。これらの機能は、サプライヤまたはカタログのプロバイダによって提供されます。この場合、ユーザーは必要なカタログのリンクを選択して品目を選択し、iProcurement に戻って品目の追加、変更または注文を完了します。

外部にあるすべてのカタログを iProcurement へリンクするには、安全なユーザー認証および品目選択を提供する Oracle の XML インタフェースを使用できます。この XML インタフェースを使用すると、複数のサプライヤからの高品質なコンテンツを収集した、非公開の安全なカタログへアクセスできます。

図 10-2 サプライヤが提供するカタログの XML 変換

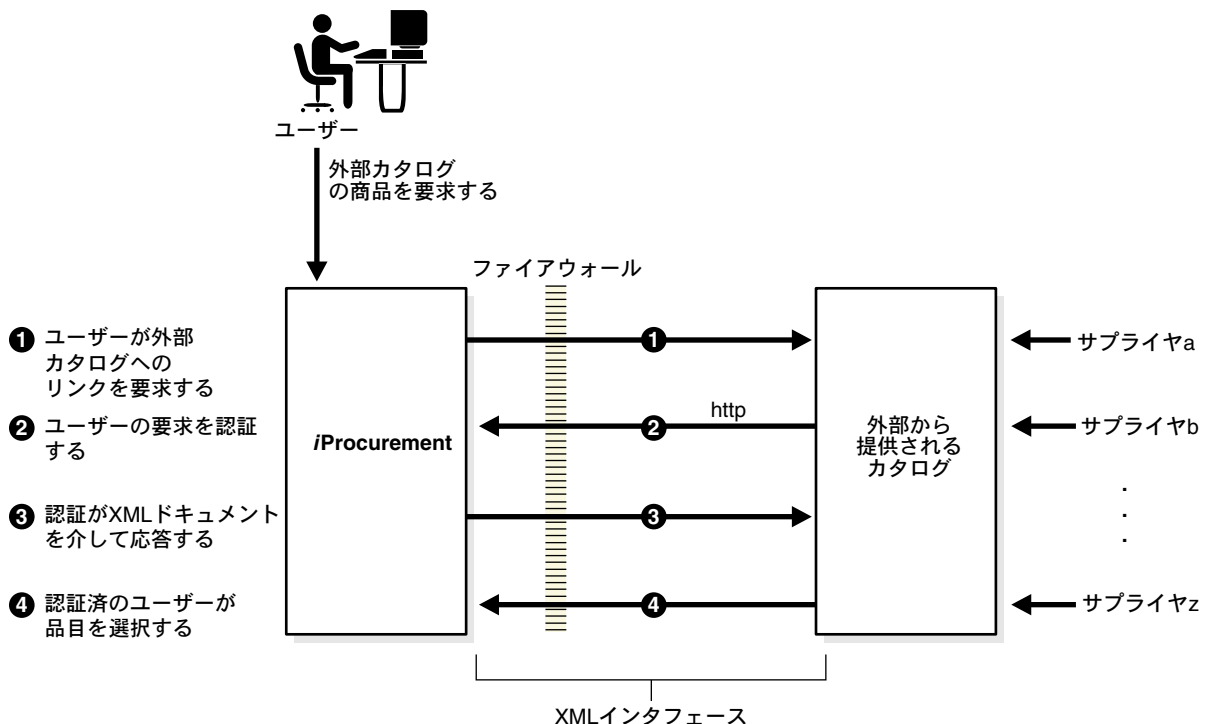


図 10-2 は、ユーザーが、サプライヤ提供のオンライン・カタログから品目を選択する場合のデータ変換処理を示しています。iProcurement での購入では、次の変換プロトコルが発生します。

1. リクエスト（ユーザー）は外部カタログへのリンクを選択し、移動します。このとき、暗号化されたセッション情報および認証 URL が提供されます。
2. 外部カタログが、安全な HTTP コールを介して、ユーザーの認証を iProcurement に要求します。
3. iProcurement が、ユーザーが正常に認証されたかどうかを示す XML ドキュメントによって、認証要求に応答します。正常に認証された場合は、注文者に関する追加情報が送信されます。
4. 認証されたユーザーは外部のサイトを参照して品目を選択し、最後にそれらの品目を注文書に追加します。これによって、外部カタログが、選択された品目を XML ドキュメントで iProcurement へ送信します。このドキュメントは解析され、注文書の明細に変換されます。ユーザーは iProcurement へ戻り、注文を完了します。

iProcurement: XML Parser for Java

iProcurement には、Oracle XML Parser for Java（中間層にインストールされていますが、Internet Application Server（iAS）とは分離されています）が含まれています。このパーサーは、XML ドキュメントが整形式かどうかを確認します。また、オプションで妥当かどうかも確認します。

国際化

iProcurement は NLS に完全に準拠していますが、リリース 10.7 および 11 では複数の言語をサポートしていません。このため、XML ドキュメント・データはインストール場所の基本言語で指定する必要があります。

注意： データベースと XML ドキュメントの間のキャラクタ・セット変換によるデータ損失を回避するために、Oracle XML Parser がサポートし、インストールされたデータベースのキャラクタ・セットと互換性のあるキャラクタ・セット・エンコーディングを指定してください。

言語の識別

XML ドキュメントは両方とも、W3C の XML 1.0 勧告の `xml:lang` 属性を使用した言語の指定がサポートされます。公開、草案または提案されているすべての勧告については、<http://www.w3.org/TR> を参照してください。

XML1.0 仕様から抜粋された次の表は、言語の識別方法の一般的な用語を示します。

```
LanguageID:=Langcode (- Subcode)*
Langcode:=ISO639Code | IanaCode | UserCode
ISO639Code:=( [a-z] | [A-Z] ) ( [a-z] | [A-Z] )
IanaCode:= ( 'i' | 'I' ) '-' ( [a-z] | [A-Z] ) +
UserCode:= ( 'x' | 'X' ) '-' ( [a-z] | [A-Z] ) +
SubCode:=( [a-z] | [A-Z] ) +
```

XML1.0 仕様では、Langcode は次の値をとることができます。

- ISO 639 「Codes for the representation of the names of languages」で定義されている 2 文字の言語コード
- Internet Assigned Numbers Authority (IANA) に登録されている言語識別子
- 団体間で非公式に共有する言語識別子

Subcode セグメントはいくつでも設定できますが、1 つ目のサブコードが 2 文字で構成されている場合は、そのコードは ISO 3166 「Codes for the representation of names of countries」の国コードである必要があります。

iProcurement 言語識別子

XML ドキュメントの言語仕様から Oracle Application 言語コードへ変換するには、次のコードを指定します。

- ISO 639 の言語コード
- ISO 3166 の国コード

次の例は、言語を英語、国をアメリカ合衆国に設定する方法を示しています。

```
<e xml:lang="EN-US">
```

`xml:lang` 宣言は、ある特定の属性にある `xml:lang` の別のインスタンスでオーバーライドされない限り、この宣言が指定されている要素のすべての属性およびコンテンツに適用されます。*iProcurement* のリリース 10.7 ~ 11i では単一言語を使用しているため、言語コードは各ドキュメント・タイプのルート要素で指定する必要があります。

iProcurement キャラクタ・セット・エンコーディング

XML Parser プロセッサには、XML 宣言のエンコーディング・パラメータを介して、XML ドキュメントで使用されているキャラクタ・セットが次のように通知されます。

```
<?xml . . . encoding='UTF-8' . . . ?>
```

XML 宣言にドキュメントのエンコーディングが指定されていない場合は、UTF-8 が使用されます。

JDK がサポートしている、他の ASCII または EBCDIC ベースのエンコーディングも使用できますが、これらは、IANA で定義されている公式なキャラクタ・セット名ではなく、JDK が必要とする形式で指定する必要があります。

XML Parser で現在サポートされているキャラクタ・セットおよびその他の Parser 仕様については、[付録 C「XDK for Java: 仕様および早見表」](#)を参照してください。

バイヤーが提供するカタログ

この項では、外部のソースから iProcurement 総合カタログをロードするための XML 仕様について説明します。この仕様では、次の機能をサポートしています。

- カatalog・スキーマ（品目分類データ）の修正
 - カテゴリの追加、改名または削除
 - 記述子（カテゴリ属性）の追加、改名または削除
- 品目の追加およびカテゴリの割当て
- 品目の削除および修正

文書型定義（DTD）

XML Parser を正常に機能させるためには、特殊文字および非 XML マークアップをエスケープする必要があります。特に、& および <> は、CDATA タグでエスケープする必要があります。

<![CDATA[**your data here**]]> 特殊文字用のすべてのタグに挿入します。

```
<SCHEMA>
  <CATEGORY ACTION="DELETE">
    <NAME><![CDATA[Pen & Pencil Gifts Sets]]></NAME>
  </CATEGORY>
</SCHEMA>
```

また、データ自体に引用符「"」が含まれている場合、文字列「"」で置き換える必要があります。

次に例を示します。

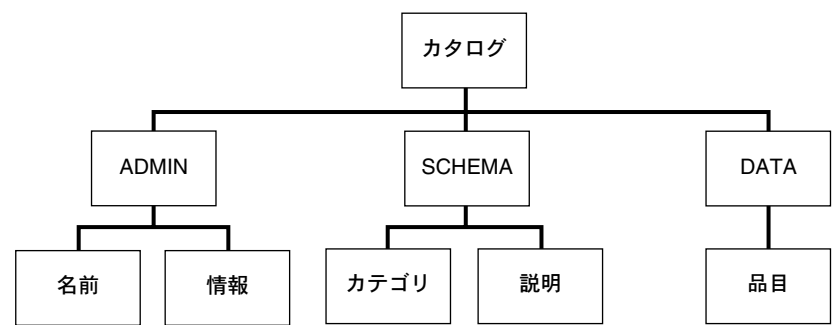
```
<itemDescription>6&quot; diameter pipe</itemDescription>
```

カタログ情報は、次の3つの主なカテゴリに分類されています。

- [DTD 管理情報: <ADMIN>](#) (10-11 ページ)
- [DTD スキーマ情報: <SCHEMA>](#) (10-12 ページ)
- [DTD: 品目情報](#) (10-15 ページ)

[図 10-3](#) に、カタログの DTD 階層図を示します。

図 10-3 バイヤーが提供するカタログ : DTD 階層図



iProcurement の例 1: バイヤーが提供するカタログの DTD

次にバイヤー提供カタログの DTD を示します。

```
<?xml version="1.0"?>
<!DOCTYPE CATALOG [

  <!ELEMENT CATALOG ( ADMIN, SCHEMA?, DATA? ) >
  <!ATTLIST CATALOG xml:lang NMTOKEN #IMPLIED >

  <!ELEMENT ADMIN ( NAME, INFORMATION ) >
  <!ELEMENT SCHEMA (CATEGORY | The DESCRIPTOR)* >
  <!ELEMENT DATA (ITEM)*>

  <!ELEMENT NAME (#PCDATA) >
  <!ELEMENT INFORMATION ( DATE, SOURCE ) >
  <!ELEMENT DATE (#PCDATA) >
  <!ELEMENT SOURCE (#PCDATA) >

  <!ELEMENT CATEGORY (NAME | KEY | TYPE | UPDATE )* >
  <!ATTLIST CATEGORY ACTION (ADD|DELETE|UPDATE) #REQUIRED>
  <!ELEMENT DESCRIPTOR (NAME | KEY | UPDATE | OWNER | TYPE )* >
  <!ATTLIST DESCRIPTOR ACTION (ADD|DELETE|UPDATE) #REQUIRED>
  <!ELEMENT OWNER (NAME?, KEY? ) >
  <!ELEMENT KEY (#PCDATA) >
  <!ELEMENT TYPE (#PCDATA) >

  <!ELEMENT ITEM (OWNER?, NAMEVALUE*, UPDATE ) >
  <!ATTLIST ITEM ACTION (ADD | DELETE | UPDATE) #REQUIRED>
```



```

<!ELEMENT UPDATE (NAME | KEY | NAMEVALUE ) * >

<!ELEMENT NAMEVALUE ( NAME, VALUE ) >
<!ELEMENT VALUE (#PCDATA) * >
]>

```

注意： iProcurement は解析処理中に DTD を検証しないため、コード・ツリー内に DTD のコピーを置く必要はありません。この例では、参照目的で記載しているのみです。

DTD 管理情報 : <ADMIN>

<ADMIN> セクションは、カタログを識別するために使用されます。このセクションは必須です。

```

<ADMIN>
  <NAME>Business Essential Reference Catalog</NAME>
  <INFORMATION>
    <DATE>![CDATA[02-FEB-99]]</DATE>
  <SOURCE>eContent Manager</SOURCE>
</INFORMATION>
</ADMIN>

```

このセクションにある情報はソース・カタログ固有であり、他のカタログ・データのロードには影響しません。表 10-1 に、<ADMIN> フィールドの説明を示します。

表 10-1 DTD: <ADMIN> フィールド

フィールド	必須かどうか	説明
Name	必須	カタログの名前
Date	必須	カタログが作成または修正された日付 注意： この日付は、アプリケーション・インスタンスのデフォルトのデータベース書式と同じである必要があります。
Source	必須	XML ドキュメントを作成した企業、人またはツール

DTD スキーマ情報 : <SCHEMA>

カタログでは、スキーマはカテゴリおよび記述子で構成されています。

- カテゴリ : カテゴリおよびそれらに含まれる品目には、対応付けられた説明的な属性があります。カテゴリは、ペン、紙、書籍、ソフトウェアなどの品目分類と考えることができます。
- 記述子 : カタログにはそれぞれ、「ベース」および「ローカル」の2種類の記述子があります。
 - ベース記述子は、カテゴリ（品目詳細、SKU、価格など）に関係なくカタログにあるすべての品目に適用されます。
 - ローカル記述子は、「コンピュータ」カテゴリの「CPU 速度」、「ペン」カテゴリの「インク・カラー」など、ある特定のカテゴリに属する品目にのみ適用されます。

カテゴリまたは記述子は、名前またはキー（あるいはその両方）で構成されます。キーが指定されている場合、カテゴリまたは記述子の識別の際に優先的に使用されます。[表 10-2](#)に、DTD の <SCHEMA> カテゴリ・フィールドおよび記述子フィールドの説明を示します。

表 10-2 DTD: <SCHEMA> カテゴリ・フィールドおよび記述子フィールド

フィールド	必須かどうか	説明
Name	条件付きで必須（カテゴリか記述子のどちらかが指定される必要があります）	カテゴリまたは記述子の名前
Key		カテゴリまたは記述子の一意識別子
Owner	記述子には必須、 カテゴリには任意	記述子の場合、記述子に対応付けるカテゴリ（名前またはキー）。Owner の値を Root または 0（ゼロ）に設定すると、記述子はすべての品目およびカテゴリに適用されます。これを、「ベース」または「ルート」記述子といいます。 カテゴリの場合、Owner を定義すると、目次で表される階層関係が作成されます。
Type	任意	記述子に使用される場合、記述子値のデータ型を表します。次の型が指定できます。 <ul style="list-style-type: none">■ STRING（デフォルト）■ NUMBER■ INTERNATIONAL（MLS が実装された場合、最終的にこの記述子値が翻訳されることを示します）

カタログ・スキーマは、次の順序で修正できます。<SCHEMA> タグには、コマンドを指定する属性、およびキーと動作可能なデータを定義するサブタグがあります。コマンドは次の通りです。

- ADD
- UPDATE
- DELETE

次の例は、<SCHEMA> タグで囲まれています。

iProcurement の例 2: DTD<SCHEMA>: カテゴリおよび記述子のカテゴリへの追加

次の例では、新しいカテゴリ、および新しいカテゴリに指定する記述子を追加します。

```
<SCHEMA>
  <CATEGORY ACTION="ADD">
    <NAME>Pen Gift Sets</NAME>
    <KEY>PEN_GIFT_SETS</KEY>
  </CATEGORY>
  <DESCRIPTOR ACTION="ADD">
    <OWNER>
      <NAME>Pen Gift Sets</NAME>
    </OWNER>
    <NAME>Package Type</NAME>
    <TYPE>String</TYPE>
  </DESCRIPTOR>
</SCHEMA>
```

iProcurement の例 3: DTD<SCHEMA>: カテゴリまたは記述子の削除

カテゴリおよび記述子には、削除するものを識別する NAME タグまたは KEY タグが必要です。記述子の場合、その記述子を含むカテゴリを識別するタグが必要です。

```
<SCHEMA>
  <CATEGORY ACTION="DELETE">
    <NAME>Pen Gifts Sets</NAME>
  </CATEGORY>
  <CATEGORY ACTION="DELETE">
    <NAME>Pen Gift Collections</NAME>
    <KEY>3245</KEY>
  </CATEGORY>

  <CATEGORY ACTION="DELETE">
    <KEY>Laptop Computer</KEY>
  </CATEGORY>
```

```
<DESCRIPTOR ACTION="DELETE">
  <OWNER>
    <NAME>Pens</NAME>
  </OWNER>
  <NAME>Barrel Color</NAME>
</DESCRIPTOR>

<DESCRIPTOR ACTION="DELETE">
  <OWNER>
    <KEY>22343</KEY>
  </OWNER>
  <NAME>Barrel Color</NAME>
  <KEY>887665</KEY>
</DESCRIPTOR>
</SCHEMA>
```

iProcurement の例 4: DTD<SCHEMA>: カテゴリまたは記述子の更新

カテゴリおよび記述子には、更新するものを識別する NAME タグまたは KEY タグが必要です。記述子の場合、記述子を含むカテゴリに対して NAME または KEY タグが必要です。次に例を示します。

```
<SCHEMA>
  <CATEGORY ACTION="UPDATE" >
    <NAME>Pen Gifts Set</NAME>
    <UPDATE>
      <NAME>Pen Gift Sets</NAME>
    </UPDATE>
  </CATEGORY>

  <CATEGORY ACTION="UPDATE">
    <KEY>C440911</KEY>
    <UPDATE>
      <NAME>Compressor Motors</NAME>
    </UPDATE>
  </CATEGORY>
</SCHEMA>
```

DTD: 品目情報

表 10-3 に、DTD の ITEM 情報フィールドの説明を示します。

表 10-3 DTD: < ITEM> 情報

フィールド	必須かどうか	説明
Owner	操作が削除の場合は任意。 それ以外の操作の場合は必須。	品目を特定のカテゴリに対応付けます。
Name/Value - Name	必須	品目を記述子に対応付けます。次のベース記述子名は Oracle iProcurement に よってシードされており、品目を追加する 場合に使用できます。 <ul style="list-style-type: none">■ Description■ UOM■ Sup Part Num■ Mfg Part Num■ Sup Name■ Mfg Name■ Price //Long Description および Picture は、 Oracle によってシードされていません が、ユーザー・インタフェースにおける 基本条件によって確実に処理されます。
Name/Value - Value	必須	指定された記述子名に対応付けられた値

データ項目はカテゴリに追加、削除および修正できます。すべてのデータ項目は、<DATA></DATA> タグの間にある必要があります。<NAME> および <KEY>（あるいはその両方）を指定する識別機能によって、ワイルド・カードを使用した効果的な更新による広範囲な値の変更が可能です。次の更新の例では、メーカー Bic のすべての品目に付いている Bic を、Bic, Inc に変更する方法を示します。

***i*Procurement の例 5: DTD ITEM: <ITEM ACTION="ADD"> を使用した品目の追加**

次の例では、<ITEM ACTION="ADD"> マーカーを使用して、品目を *i*Procurement DTD へ追加します。

```
<DATA>
  <ITEM ACTION="ADD">
    <OWNER>
      <NAME>Pens</NAME>
    </OWNER>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic</VALUE>
    </NAMEVALUE>
    <NAMEVALUE>
      <NAME>Barrel Color</NAME>
      <VALUE>Blue</VALUE>
    </NAMEVALUE>
  </ITEM>

  <ITEM ACTION="ADD">
    <OWNER>
      <NAME>Pencils</NAME>
    </OWNER>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic</VALUE>
    </NAMEVALUE>
  </ITEM>
</DATA>
```

***i*Procurement の例 6: DTD ITEM: <ITEM ACTION="DELETE"> を使用した品目の削除**

次の例では、<ITEM ACTION="DELETE"> マーカーを使用して、品目を *i*Procurement DTD から削除します。

```
<DATA>
  <ITEM ACTION="DELETE">
    <OWNER>
      <NAME>Pens</NAME>
    </OWNER>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic</VALUE>
    </NAMEVALUE>
    <!-- Pens から Bic mfg までのすべての品目を削除します。 -->
    <NAMEVALUE>
      <NAME>Barrel Color</NAME>
      <VALUE>Blue</VALUE>
    </NAMEVALUE>
  </ITEM>
</DATA>
```

```

        </NAMEVALUE>
    </ITEM>
</DATA>

```

iProcurement の例 7: DTD ITEM: <ITEM ACTION="UPDATE"> を使用した品目の更新

次の例では、<ITEM ACTION="UPDATE"> マーカーを使用して、品目を iProcurement DTD から更新します。

品目を更新する場合、Owner タグは任意です。これによって、特定の値で複数の品目を更新できます。

```

<DATA>
  <-- 品目ごとに更新 -->
  <ITEM ACTION="UPDATE">
    <OWNER>
      <NAME>Pens</NAME>
    </OWNER>
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic </VALUE>
    </NAMEVALUE>
    <NAMEVALUE>
      <NAME>Barrel Color</NAME>
      <VALUE>Blue</VALUE>
    </NAMEVALUE>
    <UPDATE>
      <NAMEVALUE>
        <NAME>Mfg Name</NAME>
        <VALUE>Bic Inc.</VALUE>
      </NAMEVALUE>
      <NAMEVALUE>
        <NAME>Barrel Color</NAME>
        <VALUE>Red</VALUE>
      </NAMEVALUE>
    </UPDATE>
  </ITEM>
  <-- 次のセクションでは、Mfg Name が Bic である品目を更新します。 -->
  <ITEM ACTION="UPDATE">
    <NAMEVALUE>
      <NAME>Mfg Name</NAME>
      <VALUE>Bic </VALUE>
    </NAMEVALUE>
  </ITEM>
</DATA>

```

サプライヤが提供するカタログおよび市場

この項では、選択された品目情報を Oracle に送信する外部カタログの仕様（データ要素およびファイル形式）について説明します。この項では、次の 4 つ項目について説明します。

- [データ要素の定義](#)
- [注文明細の XML 定義](#)
- 「[HTML 仕様](#)」 および XML データ送信の形式
- [ユーザー認証](#)

データ要素の定義

この項では、ユーザーがリモートのカタログからの品目選択を完了すると、サード・パーティのカタログから Oracle に送信される、注文書の明細ファイルのデータ要素について説明します。データ要素は、次の 6 つの論理セグメントに分類されます。

- **契約**: 選択された品目の契約情報
- **品目**: 品目、注文する量およびその他の品目情報
- **カテゴリ**: 品目のカテゴリ・コードおよびカタログ・ソース（カタログのプロバイダまたは提供者）
- **価格**: 品目の単価情報（カタログの通貨またはユーザーの目的に適した通貨）
- **サプライヤ**: 品目のサプライヤ
- **追加属性**: 品目に関する追加の顧客情報（Requisition Lines 表の Descriptive Flexfield 列にマップされます）

定義

すべてのデータ要素は、任意、必須または条件付きのいずれかです。

表 10-4 任意、必須または条件付きのデータ要素

データ要素の種類	説明
Required	Required フィールドは、サード・パーティによって指定される必要があります。このデータは、トランザクションを完了するために十分なコンテンツを含む必要があります。
Optional	Optional フィールドはトランザクションの精度を向上しますが、システムはこの値を導出、相互参照、計算またはデフォルト値で指定します。
Conditional	この状態は、ある条件の下でデータを指定する必要があることを示しています。これらの条件は、Required 列で指定されます。
Type	次の値が指定できます。 <ul style="list-style-type: none">■ 数値（実数または整数）■ 文字（文字列の長さはフィールドごとに定義されます。提供された文字列が、指定値より長い場合は切り捨てられます。）■ 日付（すべての日付は YYYYMMDD の書式である必要があります。）
Cross-Reference	認識される値を取得するために Oracle EDI Gateway への相互参照が必要かどうかを示します。

標準コード

表 10-5 に、Order Line で参照される標準コードを示します。

表 10-5 Order Line で参照される標準コード

コード	説明
D-U-N-S® Number	<p>Dun & Bradstreet 社は、EDI およびグローバルな E-Commerce トランザクションにおける企業の識別子として一般に使用されている 9 桁の識別シーケンスを開発しました。大規模な組織では、各業務場所がそれぞれ一意の識別子を持っているため、異なる D&B D-U-N-S 番号を多く持っています。</p> <p>Dun & Bradstreet, Inc. の詳細は、http://www.dnb.com を参照してください。</p>
UN/SPSC	<p>United Nations Development Programme および Dun & Bradstreet 社は共同で、製品およびサービスを分類するための標準化されたオープンなシステム、UN/Standard Product & Service Code (UN/SPSC) を定義しました。このシステムは、5 つのレベルおよび合計で約 8000 の分類を持つ階層構造を使用しています。</p> <p>UN/SPSC コードの詳細は、http://www.unspsc.org を参照してください。</p>

契約データ要素

表 10-6 契約データ要素

フィールド	必須かどうか	型	説明
Supplier Contract Number	条件付き： 1 つ以上の契約 番号が必要	CHAR(25)	サプライヤのシステムで定義されている 契約番号
Buyer Contract Number	指定	CHAR(20)	バイヤーのシステムで定義されている 契約番号
Buyer Contract Line Number	任意	NUMBER	バイヤーのシステムにおける契約の明細 項目番号 ASL ソース・ルールから導出されます。
Catalog Type	任意	CHAR(25)	次の値のどちらかを指定します。 'CONTRACTED' 'NONCONTRACTED'

品目データ要素

表 10-7 品目データ要素

フィールド	必須かどうか	型	説明
Line Type	任意	CHAR(25)	次の値のどちらかを指定します。 'GOODS': 価格、量および基本単位を指定します。これはデフォルトの値です。 'SERVICES QUANTITY': 料金ベースのサービスです。商品と同様に、サービスの単価および量を指定します。 'SERVICES AMOUNT': 量ベースのサービスです。Oracle Purchasing では Price は 1 に設定されており、実際の量は Quantity フィールドで入力されます。
Supplier Item Number	条件付き : 3 つの品目番号のうち 1 つ以上が必要	CHAR(25)	
Manufacturer Item Number	指定 (Supplier、Manufacturer または Buyer)	CHAR(25)	Manufacturer Item Number が指定されている場合は、Manufacturer Name が必要です。
Buyer Item Number		CHAR(25)	バイヤーのシステムで定義されている品目番号
Buyer Item Revision	任意	CHAR(25)	バイヤーのシステムで定義されている品目の改訂
Item Description	必須	CHAR(240)	
Quantity	任意	NUMBER	デフォルトは 1
Buyer Unit of Measure	条件付き : これらの 2 つのフィールドの 1 つ以上が必要	CHAR(25)	バイヤーのシステムで定義された基本単位のコードまたは説明 相互参照 : EDI Gateway
Supplier Unit of Measure		CHAR(25)	サプライヤのシステムで定義された基本単位のコードまたは説明 相互参照 : EDI Gateway

表 10-7 品目データ要素 (続き)

フィールド	必須かどうか	型	説明
Supplier Unit of Measure Quantity	任意	NUMBER	サプライヤの基本単位で対応付けられた量 相互参照 : EDI Gateway
Manufacturer Name	条件付き	CHAR(40)	Manufacturer Item Number が指定されている場合は必須
Hazard Class	任意	CHAR(40)	国連および運輸省が規格を規定しています。 相互参照 : EDI Gateway

カテゴリ・データ要素

表 10-8 カテゴリ・データ要素

フィールド	必須かどうか	型	説明
SPSC Category Code	条件付き : これらの 3 つ の値のうち 1 つ以上が必要	CHAR(30)	UN/SPSC 規格で定義されているカテゴリ・コード 相互参照 : EDI Gateway
Supplier Category Code		CHAR(30)	サプライヤのシステムのカテゴリ・コード
Buyer Category Code		CHAR(50)	バイヤーのシステムのカテゴリ・コード
Catalog Source	必須	CHAR(30)	カタログのプロバイダまたは提供者 (品目のサプライヤとは異なる場合があります) の名前またはコード。この値は、XML スキーマ・コードで「Catalog Trading Partner」とラベル付けされています。

価格データ要素

表 10-9 価格データ要素

フィールド	必須かどうか	型	説明
Currency	任意	CHAR(30)	デフォルトで、バイヤーのシステムで定義された（ユーザーの）目的に適した通貨に設定されます。 コードは ISO 定義に従います。 相互参照 : EDI Gateway
Unit Price	必須	NUMBER	価格が存在する場合、その価格は現在有効な契約と一致する必要があります。
Rate	任意	NUMBER	カタログの通貨からユーザーの用途に適した通貨へ変換するレート。指定された通貨が用途に適した通貨でない場合、レートは次のようになります。 品目が契約されていて、契約で固定変換レートが指定されている場合は、そのレートが使用されます。 サプライヤはレート情報を提供できません。 このフィールドが空白で、契約で固定レートが指定されていない場合、レートは内部参照スキーマ（その日のレートなど）を使用して取得されます。
Rate Date	条件付き： レートが指定されている場合にのみ必要	DATE	YYYYMMDD の書式を使用します。
Rate Type	条件付き： レートが指定されている場合にのみ必要		相互参照 : EDI Gateway

サプライヤ・データ要素

表 10-10 サプライヤ・データ要素

フィールド	必須かどうか	型	説明
Supplier DUNS	条件付き： これらの 4 つの値の うち 1 つ以上が必要	CHAR(30)	Dun & Bradstreet によって定義されたサプライヤ番号 相互参照 : EDI Gateway
Supplier Name		CHAR(80)	サプライヤの名前
Supplier Number		NUMBER	バイヤーのシステムで定義されているサプライヤ番号
Supplier Trading Partner Code		CHAR(30)	取引のパートナーとして設定するためにサプライヤに割 り当てられるカスタム・コード
Supplier Site	条件付き： Supplier DUNS が 指定されていない 場合に必要	CHAR(15)	
Contact Name	任意	CHAR(80)	このトランザクションに関する質問に対応する担当者
Contact Phone	任意	CHAR(20)	連絡電話番号

追加属性

表 10-11 追加属性

フィールド	必須かどうか	型	説明
Custom Attribute1	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute2	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute3	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute4	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute5	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute6	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute7	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute8	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute9	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute10	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute11	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute12	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute13	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute14	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義
Custom Attribute15	任意	CHAR(150)	バイヤーおよびサプライヤによって決定される定義

注文明細の XML 定義

後述する例では、前述の項で説明したデータ要素をフォーマットするためのテンプレートとして、XML Schema を使用しています。XML ファイルは、データを抽出し、iProcurement に注文明細を作成する Oracle XML Parser を介して渡されます。

すべての属性値は引用符で囲む必要があることに注意してください。たとえば、次のようなスキーマ文があるとしています。

```
<attribute name="categoryCodeIdentifier" atttype="ENUMERATION" values "SPSC SUPPLIER BUYER" />
```

この文は、次のような XML 文になります。

```
<category categoryCodeIdentifier='SPSC'>
```

すべてのデータを囲む CDATA タグ

すべてのデータは CDATA タグで囲む必要があります。データは特殊文字（引用符や二重引用符など）を含むことができるため、解析および解析中のデータの整合性を確実にするために必要です。たとえば、次のようなスキーマ文があるとしています。

```
<elementType id="manufacturerName">
  <string/>
  <description>the name of the manufacturer</description>
</elementType>
```

この文は、次のような XML 文になります。

```
<manufacturerName><![CDATA[Bob's Factory]]></manufacturerName>
```

iProcurement の例 8: 注文明細の XML Schema

```
<?xml version='1.0'?>
<s:schema id='OrderLinesDataItems'>
  <elementType id="catalogTradingPartner">
    <string/>
    <description>nique trading partner code in requisition system</description>
  </elementType>

  <elementType id="contractNumber">
    <string/>
    <description>contract in which the item exists</description>
  </elementType>
  <elementType id="buyerContractLineNum">
    <string/>
    <description> line number of the item on the buyer contract </description>
  </elementType>
```



```
<elementType id="catalogType">
  <string/>
  <description>catalog type: CONTRACTED/NONCONTRACTED</description>
</elementType>
<elementType id="supplierContract">
  <elementType ="#contractNumber"/>
  <description>supplier contract identifier for the line item </description>
</elementType>
<elementType id="buyerContract">
  <elementType ="#contractNumber"/>
  <description>buyer contract identifier for the line item </description>
</elementType>
<elementType id="contract">
  <attribute name=" contractNumberIdentifier" atttype="ENUMERATION" values=
"KNOWN UNKNOWN INFORMATIONAL NONE"/>
  <group groupOrder="OR">
    <elementType ="#supplierContract"/>
    <elementType ="#buyerContract"/>
    <elementType ="#buyerContractLineNumber"/>
    <elementType ="#catalogType" occurs "OPTIONAL "/>
  </group>
  <description>contract information for the line item </description>
</elementType>

<elementType id="itemID">
  <string/>
  <description>the item number in the chosen catalog/system</description>
</elementType>
<elementType id=" supplierItemNumber">
  <elementType ="#itemID"/>
  <description>supplier item number information</description>
</elementType>
<elementType id="manufacturerName">
  <string/>
  <description>the name of the manufacturer</description>
</elementType>
<elementType id=" manufacturerItemNumber">
  <elementType ="#itemID"/>
  <elementType ="#manufacturerName"/>
  <description>manufacturer item number information</description>
</elementType>
<elementType id="buyerItemRevision">
```

```

        <string/>
        <description> the buyer's item revision code(optional)</description>
    </elementType>
    <elementType id="buyerItemNumber">
        <elementType ="#itemID"/>
        <elementType ="#buyerItemRevision"/>
        <description>buyer item number information</description>
    </elementType>
    <elementType id="itemNumber">
        <group groupOrder="OR">
            <elementType ="#supplierItemNumber"/>
            <elementType ="#manufacturerItemNumber"/>
            <elementType ="#buyerItemNumber"/>
        </group>
        <description>the item number in the chosen catalog/system</description>
    </elementType>
    <elementType id="itemDescription">
        <string/>
        <description>the description of the item</description>
    </elementType>
    <elementType id="quantity">
        <number/>
        <description> quantity of the item (optional)</description>
    </elementType>
    <elementType id="buyerUnitOfMeasure">
        <string/>
        <description> unit of measure of the item on buyer system</description>
    </elementType>
    <elementType id="supplierUOMType">
        <string/>
        <description> unit of measure of the item on supplier system</description>
    </elementType>
    <elementType id="supplierUOMQuantity">
        <number/>
        <description> quantity associated with supplier unit of measure
        (optional)</description>
    </elementType>
    <elementType id="supplierUnitOfMeasure">
        <elementType ="#supplierUOMType"/>
        <elementType ="#supplierUOMQuantity" occurs "OPTIONAL "/>
        <description> item information on supplier system equivalent to buyer unit of
        measure</description>
    </elementType>
    <elementType id="UnitOfMeasure">
        <group groupOrder="OR">

```

```

        <elementType = "#buyerUnitOfMeasure" />
        <elementType = "#supplierUnitOfMeasure" />
    </group>
    <description> unit of measure of the item</description>
</elementType>
<elementType id="hazardClass">
    <string/>
    <description> the hazard class ID (optional)</description>
</elementType>
<elementType id="item">
    <attribute name = "lineType" atttype="ENUMERATION" values= "GOODS
AMOUNTBASEDSERVICES RATEBASEDSERVICES " default="GOODS">
    <elementType = "#itemNumber" />
    <elementType = "#itemDescription" />
    <elementType = "#quantity" occurs "OPTIONAL ">
        <default>1</default>
    <elementType>
    <elementType = "#unitOfMeasure" />
    <elementType = "#hazardClass" occurs "OPTIONAL ">
    <description>identifies the item</description>
</elementType>

<elementType id="categoryCode">
    <string/>
    <description>the code for the category</description>
</elementType>
<elementType id="category">
    <attribute name="categoryCodeIdentifier" atttype="ENUMERATION" values "SPSC
SUPPLIER BUYER" />
    <elementType = "#categoryCode" />
    <description>indicates item source catalog & category code</description>
</elementType>

<elementType id="currency">
    <string/>
    <description>ISO currency code</description>
</elementType>
<elementType id="unitPrice">
    <number/>
    <description>the price per unit of measure</description>
</elementType>
<elementType id="rateDate">
    < date/>
    <description> date of rate shown</description>
</elementType>

```

```

<elementType id="rateType">
  <String/>
  <description> type of rate</description>
</elementType>
<elementType id="rate">
  <attribute name="rateDefinition" atttype="ENUMERATION" values "FUNCTIONAL
CONTRACT SUPPLIER BUYER"/>
  <number/>
  <elementType ="#rateDate"/>
  <elementType ="#rateType"/>
  <description> conversion rate between currencies</description>
</elementType>
<elementType id="price">
  <elementType ="#currency"/>
  <elementType ="#unitPrice"/>
  <elementType ="#rate" occurs "OPTIONAL "/>
  <description>item unit price in appropriate currency</description>
</elementType>

<elementType id="supplierSite">
  <string/>
  <description>the supplier's site</description>
</elementType>
<elementType id="supplierDUNS">
  <string/>
  <description>the supplier's DUNS number</description>
</elementType>
<elementType id="supplierName">
  <string/>
  <elementType ="#supplierSite"/>
  <description>the supplier's name</description>
</elementType>
<elementType id="supplierNumber">
  <number/>
  <elementType ="#supplierSite"/>
  <description>supplier number as described in buyer system</description>
</elementType>
<elementType id="supplierTradingPartner">
  <string/>
  <description>supplier trading partner code</description>
</elementType>
<elementType id="contactName">
  <string/>
  <description> contact person at supplier site</description>
</elementType>

```

```
<elementType id="contactPhone">
  <string/>
  <description>phone number of contact</description>
</elementType>
<elementType id="supplier">
  <group groupOrder="OR">
    <elementType ="#supplierDUNS"/>
    <elementType ="#supplierName"/>
    <elementType ="#supplierNumber"/>
    <elementType ="#supplierTradingPartner"/>
  </group>
  <elementType ="#contactName" occurs "OPTIONAL "/>
  <elementType ="#contactPhone" occurs "OPTIONAL "/>
  <description>identifies suppliers</description>
</elementType>

<elementType id="languageCode">
  <string/>
  <description>the language code</description>
</elementType>
<elementType id="language">
  <elementType ="#languageCode" occurs "OPTIONAL "/>
  <description>language used to enter information for this item</description>
</elementType>

<elementType id="attribute1">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute2">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute3">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute4">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute5">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
```

```

<elementType id="attribute6">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute7">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute8">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute9">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute10">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute11">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute12">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute13">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute14">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="attribute15">
  <string/>
  <description>optional extra line attribute</description>
</elementType>
<elementType id="additionalAttributes">
  <elementType ="#attribute1" occurs "OPTIONAL "/>
  <elementType ="#attribute2" occurs "OPTIONAL "/>
  <elementType ="#attribute3" occurs "OPTIONAL "/>
  <elementType ="#attribute4" occurs "OPTIONAL "/>

```

```

<elementType = "#attribute5" occurs "OPTIONAL" />
<elementType = "#attribute6" occurs "OPTIONAL" />
<elementType = "#attribute7" occurs "OPTIONAL" />
<elementType = "#attribute8" occurs "OPTIONAL" />
<elementType = "#attribute9" occurs "OPTIONAL" />
<elementType = "#attribute10" occurs "OPTIONAL" />
<elementType = "#attribute11" occurs "OPTIONAL" />
<elementType = "#attribute12" occurs "OPTIONAL" />
<elementType = "#attribute13" occurs "OPTIONAL" />
<elementType = "#attribute14" occurs "OPTIONAL" />
<elementType = "#attribute15" occurs "OPTIONAL" />
<description>additional information about the item</description>
</elementType>

<elementType id="orderLine">
  <elementType = "#contract" />
  <elementType = "#item" />
  <elementType = "#category" />
  <elementType = "#price" />
  <elementType = "#supplier" />
  <elementType = "#language" />
  <elementType = "#additionalAttributes" />
  <description>Order line sent to requisition server</description>
</elementType>

<elementType id="OrderLinesDataElements">
  <elementType = "#catalogTradingPartner" />
  <elementType = "#orderLine" occurs "ZEROORMORE" />
  <description>complete order line sent to requisition server</description>
</elementType>
</s:schema>

```

注意： この例では、XML ヘッダーの 'catalogTradingPartner' は、各注文明細の 'catalog source' フィールドへマップされます。

スキーマで使用されているすべての用語の説明は、
<http://www.w3.org/TR/1998/NOTE-XML-data-0105/> にある XML データ仕様を参照してください。

Oracle では、列挙などのいくつかの定義を調整しています。列挙は、前述のスキーマのいくつかの要素で、CASE 文の切替えとして使用されます。これによって、要素に含まれるデータが決定されます。たとえば、「contract」では、列挙された属性値は、KNOWN、UNKNOWN、INFORMATIONAL または NONE のいずれかになります。要素が値としてのどのデータを含むかは、これらの値によって決まります。

- 値が KNOWN または INFORMATIONAL の場合: バイヤーの契約情報またはサプライヤの契約情報 (あるいはその両方) を含むことができます。
- 値が UNKNOWN または NONE の場合: フィールドがありません。

iProcurement の例 9: XML: 完全なスキーマ仕様を含む 1 つの注文明細

次に、1 つの注文明細が、任意のタグを含む完全なスキーマ仕様を説明する例を示します。この例には約 50 のタグがあり、大きさは 2.5KB です。

```
<?xml version='1.0'?>
<OrderLinesDataElements xml:lang='EN-US'>
<catalogTradingPartner><![CDATA[ABCCatalogServices]]></catalogTradingPartner>
<orderLine>
<contract contractNumberIdentifier='KNOWN'>
  <supplierContract>
    <contractNumber><![CDATA[12323634634]]></contractNumber>
  </supplierContract>
  <buyerContract>
    <contractNumber><![CDATA[987654321]]></contractNumber>
  </buyerContract>
  <buyerContractLineNumber><![CDATA[99]]></buyerContractLineNumber>
  <catalogType><![CDATA[CONTRACTED]]></catalogType>
</contract>

<item lineType='GOODS'>
  <itemNumber>
    <supplierItemNumber>
      <itemID><![CDATA[B1324]]></itemID>
    </supplierItemNumber>
    <manufacturerItemNumber>
      <itemID><![CDATA[X456]]></itemID>
      <manufacturerName><![CDATA[Bob's Factory]]></manufacturerName>
    </manufacturerItemNumber>
    <buyerItemNumber>
      <itemID><![CDATA[2222XY]]></itemID>
      <buyerItemRevision><![CDATA[4]]></buyerItemRevision>
    </buyerItemNumber>
  </itemNumber>
  <itemDescription><![CDATA[Purple and Red]]></itemDescription>
  <quantity><![CDATA[999]]></quantity>
  <unitOfMeasure>
    <buyerUnitOfMeasure><![CDATA[ea]]></buyerUnitOfMeasure>
    <supplierUnitOfMeasure>
      <supplierUOMType><![CDATA[each]]></supplierUOMType>
    </supplierUnitOfMeasure>
  </unitOfMeasure>
</item>
</orderLine>
</OrderLinesDataElements>
```



```
        <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
      </supplierUnitOfMeasure>
    </unitOfMeasure>
    <hazardClass><![CDATA[2768]]></hazardClass>
  </item>

  <category categoryCodeIdentifier='SPSC'>
    <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
  </category>

  <price>
    <currency><![CDATA[USD]]></currency>
    <unitPrice><![CDATA[50.99]]></unitPrice>
    <rate rateDefinition='CONTRACT'>
      <rateDate><![CDATA[19981210]]></rateDate>
      <rateType><![CDATA[corporate]]></rateType>
    </rate>
  </price>

  <supplier>
    <supplierName><![CDATA[ACME Hot Air Balloons]]></supplierName>
    <supplierSite><![CDATA[Kinshasa]]></supplierSite>
    <supplierTradingPartner><![CDATA[Traders R Us]]></supplierTradingPartner>
    <contactName><![CDATA[Ramanujam Kondetimanahalli]]></contactName>
    <contactPhone><![CDATA[3015061111]]></contactPhone>
  </supplier>

  <language>
    <languageCode><![CDATA[AmEnglish]]></languageCode>
  </language>

  <additionalAttributes>
    <attribute1><![CDATA[additional information 1]]></attribute1>
    <attribute2><![CDATA[additional information 2]]></attribute2>
    <attribute3><![CDATA[additional information 3]]></attribute3>
    <attribute4><![CDATA[additional information 4]]></attribute4>
    <attribute5><![CDATA[additional information 5]]></attribute5>
    <attribute6><![CDATA[additional information 6]]></attribute6>
    <attribute7><![CDATA[additional information 7]]></attribute7>
    <attribute8><![CDATA[additional information 8]]></attribute8>
    <attribute9><![CDATA[additional information 9]]></attribute9>
    <attribute10><![CDATA[additional information 10]]></attribute10>
    <attribute11><![CDATA[additional information 11]]></attribute11>
    <attribute12><![CDATA[additional information 12]]></attribute12>
    <attribute13><![CDATA[additional information 13]]></attribute13>
```

```

        <attribute14><![CDATA[additional information 14]]></attribute14>
        <attribute15><![CDATA[additional information 15]]></attribute15>
    </additionalAttributes>

</orderLine>
</OrderLinesDataElements>

```

iProcurement の例 10: XML: 2 品目のトランザクションの例

次の例は一般的な 2 品目のトランザクションを示しており、次の 2 つの品目を使用しています。

- 1 目の品目「ハード・ドライブ - 540MB IDE」は、価格 485.99、数量 34、サプライヤ名 A-1 Lighting です。
- 2 目の品目「高速アセンブリ・マシン」は、価格 12575.99、数量 9、サプライヤ名 JCN Technologies です。

「ABC Catalog Services」は、これら 2 つの品目のデータを Oracle システムへ送信するサード・パーティの外部カタログ・サービスです。次のデータが両方の品目に対して提供されます。

- サプライヤの契約番号
- サプライヤの部品番号
- サプライヤの基本単位

UN/SPSC は、両方の品目に使用されるカテゴリ・コードです。

```

<?xml version='1.0'?>
<OrderLinesDataElements>
<catalogTradingPartner><![CDATA[ABCCatalogServices]]></catalogTradingPartner>

<orderLine>
<contract contractNumberIdentifier='KNOWN'>
    <supplierContract>
        <contractNumber><![CDATA[111111112767-1]]></contractNumber>
        <contractLineNumber><![CDATA[12]]></contractLineNumber>
    </supplierContract>
</contract>

<item lineType='GOODS'>
    <itemNumber>
        <supplierItemNumber>
            <itemID><![CDATA[C13139]]></itemID>
        </supplierItemNumber>
    </itemNumber>

```

```
<itemDescription><![CDATA[Hard drive-540MB IDE]]></itemDescription>
<quantity><![CDATA[34]]></quantity>
<unitOfMeasure>
  <supplierUnitOfMeasure>
    <supplierUOMType ><![CDATA[Each]]></supplierUOMType>
    <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
  </supplierUnitOfMeasure>
</unitOfMeasure>
</item>

<category categoryCodeIdentifier='SPSC'>
  <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
</category>

<price>
  <currency><![CDATA[USD]]></currency>
  <unitPrice><![CDATA[485.99]]></unitPrice>
</price>

<supplier>
  <supplierName><![CDATA[A-1 Lighting]]></supplierName>
  <supplierSite><![CDATA[Washington]]></supplierSite>
</supplier>
</orderLine>

<orderLine>
<contract contractNumberIdentifier='KNOWN'>
  <supplierContract>
    <contractNumber><![CDATA[22222225678-2]]></contractNumber>
    <contractLineNumber><![CDATA[15]]></contractLineNumber>
  </supplierContract>
</contract>

<item lineType='GOODS'>
  <itemNumber>
    <supplierItemNumber>
      <itemID><![CDATA[P22378]]></itemID>
    </supplierItemNumber>
  </itemNumber>
  <itemDescription><![CDATA[High speed assembly machine]]></itemDescription>
  <quantity><![CDATA[9]]></quantity>
  <unitOfMeasure>
    <supplierUnitOfMeasure>
      <supplierUOMType ><![CDATA[Each]]></supplierUOMType>
      <supplierUOMQuantity><![CDATA[1]]></supplierUOMQuantity>
    </supplierUnitOfMeasure>
  </unitOfMeasure>
```

```
</item>

<category categoryCodeIdentifier='SPSC'>
  <categoryCode><![CDATA[5149-9908-00]]></categoryCode>
</category>

<price>
  <currency><![CDATA[USD]]></currency>
  <unitPrice><![CDATA[12575.99]]></unitPrice>
</price>

<supplier>
  <supplierName><![CDATA[JCN Technologies]]></supplierName>
  <supplierSite><![CDATA[New York]]></supplierSite>
</supplier>

</orderLine>
</OrderLinesDataElements>
```

HTML 仕様

ユーザーが選択した品目データを XML で iProcurement へ送信するための、外部カタログ・ソースの HTML 形式を次に示します。この形式では、XML ファイルをセグメントに断片化する必要があります。

選択された品目の iProcurement への送信 : 外部カタログの HTML ファイル形式

次に、選択された品目を外部カタログから iProcurement へ送信するときに使用される HTML 形式を示します。

```
<HTML>
  <BODY onLoad="document.orderForm.submit()">
    <FORM ACTION="URL of the Web Requisitions" METHOD="POST"
      NAME="orderForm">
      <INPUT type="hidden" name="REQ_TOKEN" value="Requisition token">
      <INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="N">
      <INPUT type="hidden" name="ITEM_XML_DATA1" value="First data segment">
      <INPUT type="hidden" name="ITEM_XML_DATA2" value="Second data segment">
      .....
      <INPUT type="hidden" name="ITEM_XML_DATA" value="Nth data segment">
    </FORM>
  </BODY>
</HTML>
```

HTML 要素の説明

表 10-12 に、関連の HTML タグ名でグループ化された HTML 要素を示します。

表 10-12 HTML 要素

HTML 要素	形式	説明
<BODY>	<code>onLoad="document.orderForm. submit()"</code>	クライアントのブラウザから iProcurement への HTTP リダイレクトを提供します。
<FORM>	<code>ACTION="URL of the Web Requisitions"</code> <code>METHOD="POST"</code> <code>NAME="orderForm"</code>	<p><code>orderForm</code> の <code>Action</code> 属性は、セルフサービス購入によって指定される URL に指定する必要があります。</p> <p>フォーム送信の種類</p> <p>送信されるフォームの名前</p>
<INPUT>	<code><INPUT type="hidden" name="REQ_TOKEN" value="Requisition token"></code> <code><INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="N"></code> <code><INPUT type="hidden" name="ITEM_XML_DATAN" value="Nth data segment"></code>	<p><code>Requisition Token</code> の非表示のフォーム要素を作成します。このトークンは、認証が正常に処理され、変更なしで iProcurement へ戻された後に、サード・パーティのカタログ・プロバイダへ送信されます。内部システムに関連したデータが含まれます。</p> <p>2つのシステム間で転送されるデータ・セグメントの数を送信する非表示のフォーム要素を作成します。値 <code>N</code> には、<code>ITEM_XML_DATA</code> フォーム要素の数が必要です。</p> <p>データ・セグメントを転送する非表示のフォーム要素を作成します。データ・セグメントは、XML ファイルを小さいファイルに分割して構成されます。各セグメントのサイズの設定は可変である必要があります。この変数の推奨値は、2000 文字です。名前の最後の文字によって、対応するデータ・セグメントの索引が構成されます。索引は 1 から始まり、<code>NO_OF_DATA_SEGMENTS</code> 値の <code>N</code> まで 1 ずつ増加します。</p>

iProcurement の例 11: HTML/XML ファイル

この例では、表 10-12 で定義された HTML 要素を使用します。

```
<HTML>
  <BODY onLoad="document.orderForm.submit()">
    <FORM ACTION="http://ap411sun.us.oracle.com:9999/OA_JAVA_
SERV/wp4/integrate/apps.Order" METHOD="POST" NAME="orderForm">
      <INPUT type="hidden" name="REQ_TOKEN"
value="template=tpn,action=addLines,function=addToOrder,por_req_session_
id=1,.....">
      <INPUT type="hidden" name="NO_OF_DATA_SEGMENTS" value="2">
      <INPUT type="hidden" name="ITEM_XML_DATA1" value="<?xml
version='1.0'?><OrderLinesDataElements><catalogTradingPartner><![CDATA[ABCCatalogSer
vices]]></catalogTradingPartner><orderLine>.....">
      <INPUT type="hidden" name="ITEM_XML_DATA2" value="
.....
</orderLine>
      <orderLine>
        <contract
.....
</OrderLinesDataElements>">
</FORM>
</BODY>
</HTML>
```

ユーザー認証

iProcurement に最初にログインしたときに、セッションに対するランダムなユーザー識別番号（セッション・チケット）が生成されます。この番号は、一方向暗号で暗号化され、Oracle Procurement Server に格納されます。

外部カタログへのリンクを選択すると、暗号化されたセッション・チケットおよびユーザー認証のための URL が送信されます。次に、カタログ・プロバイダで想定するコールの列を示します。

```
https://www.extsupplier.com?url=oas.us.oracle.com/wr41102/plsql/icx_ext_supplier.authenticate_user&ticket=128019274
```

このコンテンツの説明を次に示します。

- **www.extsupplier.com** は、外部カタログが指定する URL です。
- **ias.us.oracle.com/wr41102/plsql/icx_ext_supplier.authenticate_user** は、戻される顧客の URL です。（**ias.us.oracle.com** はファイアウォールの外にあるアプリケーション・サーバーの URL です。）

次に、カタログ・プロバイダは、Procurement Server へ HTTP コールを送信し、SSL を使用して、ユーザーが送信した URL アドレスで暗号化されたチケットを検証することを要求します。

```
https://ias.us.oracle.com/wr41102/plsql/icx_ext_supplier.authenticate_user?ticket=128019274
```

これは、実際には、クライアント・サイトのファイアウォール内にあるデータベースに格納された PL/SQL パッケージへのコールです。

外部カタログのプロバイダは、カタログ・プロバイダのデジタル証明を認証するファイアウォールの外にあるアプリケーション・サーバーへ接続し、内部アプリケーション・サーバーへのコールを許可します。ここで示している暗号化セッション・チケットは表に格納されたバージョンに対して検証され、残りのユーザー情報はカタログ・プロバイダへ戻されます。

iProcurement の例 12: 有効なセッションの XML ドキュメント

セッション・チケットが有効な場合、iProcurement はログインを渡します。これには、次のとおり名前、配達情報、企業、担当部門、注文番号および Requisition Server に戻される URL が含まれます。

```
<?xml version='1.0'?>
  <RequisitionUser>
    <userName>CBLACK</userName>
    <company>VIOP</company>
    <operatingUnit>Organization</operatingUnit>
    <shipTo>Philadelphia</shipTo>
    <deliverTo>Philadelphia</deliverTo>
    <reqToken>Req_Token</reqToken>
    <returnURL>ap411sun.us.oracle.com:5555/FJ_JAVA_SERV/faboujaw/PS/tpn_
redirect</returnURL>
  </RequisitionUser>
```

認証済 XML Schema

外部カタログのプロバイダがユーザーの認証を要求すると、iProcurement は、暗号化されたパラメータとしてセッション・チケットを使用して、PL/SQL プロシージャをコールしてそのユーザーを認証します。

iProcurement の例 13: 認証済 XML Schema: 戻された注文ユーザーの XML ドキュメント

次の XML スキーマは、戻された注文ユーザーの XML ドキュメントを示します。

```
<?xml version='1.0'?>
<s:schema id='RequisitionUser'>
  <elementType id="userName">
    <string/>
    <description>Unique user name of person in the requisition system</description>
  </elementType>
  <elementType id="company">
    <string/>
    <description>Unique company name in the requisition system</description>
  </elementType>
  <elementType id="operatingUnit">
    <string/>
    <description>Unique operating unit name in the requisition system</description>
  </elementType>
  <elementType id="shipTo">
    <string/>
    <description>shipTo account for the requisition</description>
  </elementType>
  <elementType id="deliverTo">
    <string/>
    <description>deliverTo account for the requisition </description>
  </elementType>
  <elementType id="reqToken">
    <string/>
    <description>Unique requisition ID used in the requisition system</description>
  </elementType>
  <elementType id="returnURL">
    <string/>
    <description>URL that ReqLines should be redirected to</description>
  </elementType>
  <elementType id="RequisitionUser">
    <elementType ="#userName"/>
    <elementType ="#company"/>
    <elementType ="#organisation" occurs "OPTIONAL"/>
    <elementType ="#reqToken"/>
    <elementType ="#shipTo" occurs "OPTIONAL"/>
  </elementType>
</s:schema>
```

```
<elementType = "#deliverTo" occurs "OPTIONAL" />
<elementType = "#returnURL" />
<description>object sent to external supplier for user
identification</description>
</elementType>
</s:schema>
```

iProcurement の例 14: 認証済ユーザー : 戻された XML ドキュメント例

この例にはデータがあります。次に、認証済ユーザーに対して生成され戻される XML ドキュメントの一般的な例を示します。

```
<?xml version='1.0'?>
<RequisitionUser>
  <userName>Fred Bloggs</userName>
  <company> Oracle Corporation</company>
  <operatingUnit>Manufacturing</operatingUnit>
  <shipTo>Oracle HQ</shipTo>
  <deliverTo>Kevin Miller</deliverTo>
  <reqToken> 1245</reqToken>
  <returnURL>http://reqs.us.oracle.com/order/75</returnURL>
</RequisitionUser>
```

認証されていない XML Schema

外部のサプライヤから送信されたユーザー ID が有効でない場合、次のスキーマが生成された XML を示します。

iProcurement の例 15: 非認証ユーザー : XML Schema

```
<?xml version='1.0'?>
<s:schema id='InvalidUser'>
  <elementType id="message">
    <string/>
    <description>Message generated when an invalid sessionID is sent</description>
  </elementType>
```

iProcurement の例 16: 非認証ユーザー : XML サンプル・ドキュメント

次に、非認証ユーザーに対して生成され戻される XML ドキュメントの一般的な例を示します。

```
<InvalidUser>
  <message>Invalid user</message>
</InvalidUser>
```

XML メッセージ機能を使用した Phone Number Portability

この章では、メッセージ・ペイロードとして XML を使用する Phone Number Portability アプリケーションの概要を説明します。

この章の内容は次のとおりです。

- [Phone Number Portability のメッセージ機能の概要](#)
- [SDP における Number Portability およびメッセージ機能アーキテクチャ](#)
- [Number Portability のプロセス](#)
- [ネットワーク要素の設置](#)
- [Event Manager を使用したメッセージの非同期送受信](#)
- [Internet Message Studio \(iMessage\) を使用したアプリケーションのメッセージ・セットの作成](#)

Phone Number Portability のメッセージ機能の概要

この章では、Phone Number Portability メッセージベースの製品（Number Portability）の概要を説明します。

Number Portability は、コンシューマが通信サービス・プロバイダを変更したり、引っ越して場所を移動したり、サービスを変更した場合でもその電話番号を保持できるメカニズムです。この概念は、競争の発生に取り組む調整機関によって決定され、電話番号を保持できると、コンシューマがサービス・プロバイダを積極的に変更することを示します。Number Portability は、米国の遠隔競争市場における劇的な成長の重要な要素として広く使用されています。

Number Portability メッセージベースのアプリケーションは、iMessage Studio、Event Manager および Adapter を使用します。アプリケーションは、XML をメッセージ・ペイロードとして使用し、通信業界で共通の B2B プロトコルを使用して 2 つのサービス・プロバイダ間で通信します。

これは、Oracle CRM Applications 11i にある Oracle Service Delivery Platform（OSDP または SDP）のメッセージ機能およびイベント管理機能を示しています。これは CRM の機能です。

高速な構成を可能にする Number Portability

Number Portability 製品を使用すると、コンサルタントは次のことを行えます。

- アプリケーションに XML メッセージ DTD を構成することによって、製品を高速に実装できます。
- XML メッセージの異なるノードを、Oracle データ・ソース、SQL 問合せ、またはストアド・プロシージャに割り当てることができます。
- SQL 問合せをネストできます。

たとえば、dept のリストおよび各 dept のすべての emp のリストを XML メッセージに取得するには、次の操作を実行します。

1. Number Portability アプリケーションで 2 つの問合せを実行します。
2. 提供される GUI に、メッセージをコーディングしないで構成します。

Number Portability のアドバンスト・キューイングは、データベースと外部システム・アダプタの通信用の標準フォーマットとして XML メッセージを使用します。

外部アダプタの概要

外部アダプタは、次のものに対するリスニングを実行する Java プログラムです。

- 実行するコマンドのデータベース・パイプ
- 複数のスレッドで処理するメッセージのアドバンスト・キューイング（AQ）

コマンドは XML 形式で `performControlMessageProcessing` メソッドへ送信されます。これによって、多くのパラメータがアダプタへ渡されます。

たとえば、アダプタを実行する 3 つのスレッドのデフォルトで起動するには、STARTUP コマンドは次のようになります。

```
<COMMAND>
  <MESSAGE_CODE>STARTUP</MESSAGE_CODE>
  <INITIAL_THREADS>5</INITIAL_THREADS>
</COMMAND>
```

これによって、アダプタのカスタマイズに制御性および柔軟性が与えられます。また、独自のコマンドを定義することもできます。XML メッセージを解析する場合の制限はありません。

参照：

- 『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
- ユーザー・インタフェースおよび iMessage Studio の詳細は、『Oracle Number Portability 11i User's Guide』を参照してください。
- 『Implementing Oracle SDP Number Portability』も参照してください。

この章で使用される用語

次は、この章で使用される用語の定義です。

- NPAC: Number Portability Administration Center
- NRC: Number Registration Center (NPAC ともいいます)
- SDP: Oracle Service Delivery Platform

Phone Number Portability アプリケーション構築の要件

Number Portability アプリケーションを構築するには、次のものがが必要です。

- Oracle Applications 11i
- Oracle SDP Number Portability 11i リリース 2
- Oracle8i

SDP における Number Portability およびメッセージ機能アーキテクチャ

Oracle Service Delivery Platform (SDP) フレームワークの Number Portability およびメッセージ機能アーキテクチャは、次のコンポーネントで構成されています。Event Manager が主要なコンポーネントです。

- Communication Protocol Adapter
- Order Processing Engine
- Workflow Engine
- Fulfillment Engine
- Event Manager
- SDP Repository

Communication Protocol Adapter

Communication Protocol Adapter は SDP と外部システム間のインタフェースです。これは次のメッセージ・フローを処理します。

- 受信された注文は、適切な Communication Protocol Adapter によって受信され、SDP へ渡されます。
- 送信メッセージは、SDP から外部システムへ渡されます。

次のアダプタがサポートされています。

- ファイル /FTP (バッチ・モード処理をサポートします)
- HTTP
- スクリプト
- Telnet セッションなどの Interactive Adapter

Order Processing Engine

注文管理システムからの注文は、SDP Work Item または論理明細品目に変換されます。また、注文はメッセージの処理によって、内部的にも作成されます。

これらの明細品目は、Dependency Manager または Order Analyzer によって分析されます。これによって、実行する Normalized Work Item の最終的なセットが作成されます。

Workflow Engine

このモジュールは実際の処理の流れ（Fulfillment Action）を指定して、アプリケーションの機能性を満たすために実行されます。このモジュールは Fulfillment Action を再使用して、NP Service Provider Mediation や NRC などの新しい機能をカスタマイズできます。

Workflow Engine は、Fulfillment Action を決定して各 Work Item に対して実行し、通信する Network Element を決定します。また、Workflow Engine は、フルフィルメント要素タイプ、フルフィルメント要素タイプのソフトウェア・バージョンおよびアダプタのタイプに基づいて適切なフルフィルメント・プロシージャを選択して実行します。

次に、フルフィルメント・プロシージャは、コードを生成した Internet Message Studio を使用して、メッセージを送信および処理します。Event Manager による Fulfillment Action の実行結果のイベント通知を取得すると、エンジンは処理を続行して Work Item を完了し、指定された注文のキューにある次の Work Item を選択します。このコンポーネントは、Oracle Workflow エンジンを使用します。

Fulfillment Engine

適用される Fulfillment Action および Network Element は、SDP がエンジンを提供することによって使用され、どの Fulfillment Program を実行するかを判断します。これは、データベースの PL/SQL エンジンを必ず使用して、ユーザー定義のプロシージャを実行します。

Event Manager

Event Manager は、関心のある様々なサブスクリバを異なるイベント型に登録する一般的なパブリッシュ・サブスクリブ・モジュールです。サブスクリバは、SDP Translator（イベントが新しい注文として伝播された場合）、Workflow Engine（外部イベントで待機している Workflow をイベントが再起動した場合）または API になることができます。Event Manager は、非同期アプリケーションのメッセージ機能を構築します。これには、柔軟性のある一連の API があり、開発者はこれらの API を使用してアプリケーションに基づく非同期メッセージを作成できます。

SDP リポジトリ

コア SDP リポジトリによって、ユーザーは注文を作成し、ネットワーク要素を構成できます。Work Item、Fulfillment Action、Fulfillment Program、Network Element の定義などがその例です。NP データベースには、強力な NP 固有のデータ（Subscription Version、Service Provider、Routing Number など）のエンティティが含まれています。

Number Portability のプロセス

Number Portability は次のタスクを実行します。

1. XML または DTD 要素を、SQL 表または PL/SQL ストアド・ファンクションのいずれかへリンクします。
2. バックグラウンドでストアド・プロシージャを動的に作成および構築します。また、追加の処理のために XML メッセージをエンキューします。これは、表から値を抽出 / 問い合わせることによって、または要素に対応付けられたストアド・ファンクションを動的に実行することによって XML を構築します。
3. 実行時に、ユーザーまたはプログラムは、この動的に実行されるプロシージャを実行します。このプロシージャには、XML メッセージを作成して追加の処理のためにそのメッセージをエンキューするインテリジェント機能があります。

Number Portability 製品は、ワークフロー・マネージャとして動作します。これは、顧客によって要求されたサービスを提供するために使用されます。

新しい電話サービスを注文した場合の処理

たとえば、新しい電話サービスを注文した場合、電話会社は注文を受け取り、アプリケーションを使用して注文情報を獲得します。

発生するイベントの流れは次のとおりです。Number Portability アプリケーションは次のすべての手順で使用されており、プロセスのインスタンスとして動作します。

1. 顧客がサービス（新しい電話の取付けなど）の注文を行います。
2. Provisioning アプリケーションは販売注文を獲得し、指定された検証プロセスおよび認可プロセスを開始します。
3. Provisioning アプリケーションは外部システムと通信します。たとえば、顧客の信用度評価を調べたり、その他の処理のためにサード・パーティのアプリケーションにアクセスします。

この通信には、2 つのシステム間で定義された XML 形式のプロトコルが使用されます。Oracle Work Flow を使用すると、コンサルタントは実行時でもグラフィカルな形式で業務処理を構成および参照できます。

ローカル・サービス・プロバイダを変更した場合の処理

Number Portability は、電話のローカル・サービス・プロバイダを変更した場合も動作します。次にそのプロセスを示します。

1. 顧客がローカル・サービス・プロバイダに連絡します。
2. ローカル・サービス・プロバイダは、以前のサービス・プロバイダとともにその要求を検証します。これは、独立した調整機関を通じて行われます。アメリカ合衆国では、この調整機関が NPAC（Number Portability Administration Center）です。
3. 長距離電話会社を変更すると、調整機関は音声を通してオンラインになります。ローカル・サービス・プロバイダを変更した場合は、電子的なメッセージ機能を通じてこれが行われます。
4. 新しいサービス・プロバイダはメッセージを NPAC へ送信するため、NPAC は要求を検証し、認証または認可が新しいサービス・プロバイダに付与され、そのプロバイダはこの新しい顧客を獲得できます。
5. NPAC がメッセージを以前のプロバイダ（提供者）へ送信します。プロバイダは注文を認証し、XML を使用してメッセージを NPAC へ送信します。
6. NPAC は、認可を新しいサービス・プロバイダ（受信者）へ送信します。
7. これで、注文が両方のプロバイダで認可されます。

注意： ここでのメッセージ機能はすべて、SDP Number Portability 製品における主な形式として XML を使用しています。別のプロトコルが必要な場合は、カスタム Adapter をプラグインして、XSL またはカスタム・コードを使用して変換してください。

8. 顧客がプロバイダを実際に変更する日には、NPAC はブロードキャスト・メッセージを国内すべての電話サービス・プロバイダへ送信します。このとき、すべての電話通信会社は、システム内のプロセスにあるネットワーク要素（ネットワーク・データベース）を更新する必要があります。

データ形式である XML: アドバンスト・キューイングの使用

XML は、すべてのメッセージ機能に使用されるデータ形式です。アドバンスト・キューイング (AQ) は、プロセス (およびシステム) の各場面で使用されます。

Message Builder モジュールは XML メッセージを作成し、エンキューします。Communication Protocol Adapter (Adapter) はメッセージのデキューを開始して、そのメッセージを外部システムへ送信します。

AQ は、キューにメッセージを格納するためのみに使用されます。これは、先入れ先出し (First In First Out: FIFO) のキューイング・システムとして動作します。メッセージを送信するために使用されるプロトコルはすべてのシステムで異なり、エンド・ユーザー指定 (Flat File/CORBA など) です。

次のことがいえます。

- ローカル・プロバイダまたは長距離電話会社を変更する注文が受信されると、注文要求は、XML または変換されたメッセージとして NPAC へ送信されます。
- 変更が認証および認可されると、NPAC は、国内すべての電話会社へ XML メッセージを送信します。次に、これらの電話会社がそのネットワーク要素を提供します。

このメッセージ機能に XML が使用される理由

XML が使用されるのは、必要に応じて他の形式に変更または変換できる柔軟性があるためです。

たとえば、メッセージを分配し、ネットワーク要素 (データベース) を設置 (更新) するために、フラット・ファイル・メッセージ形式を必要とする国もあります。XSL またはカスタム・コードを使用すると、生成された XML を必要なフラット・ファイル形式に簡単に変換できます。

この Number Portability アプリケーションは、ベルギーでこの方法で使用されており、正常に実行されています。ベルギーでは、フラット・ファイル・メッセージ形式が必要とされています。

ネットワーク要素の設置

電話システムには、エンド・ユーザーの個人電話番号以外に、設置（更新）される他のネットワーク要素があります。たとえば、交換機、Service Control Point（SCP）、ルーター、LDAP サーバーなどです。

次に、SDP Provisioning アプリケーションがどのように使用されるかについて別の例を示します。

1. 電話のローカル・サービス・プロバイダが、交換機の設置を要求をします。
2. Mediation Layer がその交換機と通信します。
3. XML 対応の既存のシステムである Service Delivery Platform（SDP）は、メッセージを Mediation Layer へ送信します。
4. SDP は、設置（更新）が完了すると Mediation Layer から応答を受信します。

また、このメッセージ機能は、メッセージ・ペイロードおよびアドバンスド・キューイング（AQ）として XML を使用します。ここで、AQ は主に XML キューの記憶メディアとして使用されます。将来のリリースでは、AQ 上で JMS インタフェースを使用して、標準インタフェースを提供する可能性があります。

Event Manager を使用したメッセージの非同期送受信

SDP は、Event Manager を使用してメッセージを非同期に送受信できます。[表 11-1](#) に、Event Management を実装する SDP のキューおよびサービスを示します。

表 11-1 Event Management を実装する SDP キューおよびサービス

キュー名	サービス名	備考
受信メッセージ・キュー	メッセージ・サーバー	すべての受信メッセージを処理します。
内部イベント・キュー	イベント・サーバー	内部で使用するために生成されるイベントは、高速に処理するために内部イベント・キューでエンキューされます。
発信メッセージ・キュー	通信アダプタ	発信メッセージ・キューからメッセージをデキューし、ピア・システムにそれを渡します。

Event Manager は、アプリケーション・システムに送信されるすべてのメッセージを処理します。システムへ送信されるメッセージは、要求メッセージへの応答またはリモート・システムからのイベント通知になります。

メッセージを送信するコード例

次に、PORT_IN メッセージを作成し、それを NRC コンシューマへ送信するコードのフラグメントの例を示します。NRC コンシューマは、メッセージを NRC システムへ送信する SDP アダプタです。

```
DECLARE
l_error_codeNUMBER ;
l_error_messageVARCHAR2(2000) ;
l_msg_headerXNP_MESSAGE.MSG_HEADER_REC_TYPE ;
l_msg_textVARCHAR2(4000) ;
l_fnd_messageVARCHAR2(4000) ;
BEGIN
/*
PORT_IN 要求メッセージを作成します。
*/
PORT_IN.CREATE_MSG(XNP$TN=>'3037505639'
    XNP$PORTING_ID=>1001,
    x_msg_header=>l_msg_header,
    x_msg_text=>l_msg_text,
    x_error_code=>l_error_code,
    x_error_message=>l_error_message,
    p_sender_name=>'TELIA') ;
/*
メッセージを NRC に送信する前に、顧客サポート・システムに通知して並行性を取得します。
*/
IF (l_error_code = 0) THEN

/*
顧客サポート・システムに通知するためのカスタム・プロトコルです。
*/
NOTIFY_CUSTOMER_CARE(l_msg_header,
    l_msg_text,
    l_error_code,
    l_error_message) ;
IF (l_error_code = 0) THEN
    XNP_MESSAGE.PUSH(P_MSG_HEADER=>l_msg_header,
        P_BODY_TEXT=>l_msg_text,
        P_QUEUE_NAME=>XNP_EVENT.C_OUTBOUND_MSG_Q,
        P_RECIPIENT_LIST=>'NRC_ADAPTER' );
...

```

メッセージは、受信メッセージ・キューまたは内部イベント・キューでもエンキューできますが、これらのキューはシングル・コンシューマ・キューです。各キューは、関連識別子を使用して、アプリケーション間で共有できます。

Internet Message Studio (iMessage) を使用したアプリケーションのメッセージ・セットの作成

iMessage ユーティリティは、アプリケーションまたはエンタープライズのメッセージ・セットを定義します。これによって、アプリケーションに基づくメッセージを簡単に開発でき、アプリケーション・メッセージの構築、公開、検証および処理に必要なすべてのコードが生成されます。

また、アプリケーション間でのメッセージの共有も可能になり、同じメッセージはエンタープライズにおける様々なアプリケーションで再定義する必要がありません。アプリケーションは、そのメッセージ機能が必要とする、実行時に生成されたすべてのプロシージャを実行できます。また、特定のビジネス・ロジックを含めるために必要な手段またはカスタマイズの要点を提供します。メッセージは標準 XML を使用して生成されます。

コードの生成

定義されたすべてのメッセージに対して、iMessage は、そのメッセージの名前を使用してパッケージを作成し、そのパッケージの一部として次のプロシージャを作成します。

- CREATE_MSG()
- SEND()
- PUBLISH()
- VALIDATE()
- PROCESS()
- DEFAULT_PROCESS()

メッセージ・セットの定義

図 11-1 「iMessage のデータ・ソース・ウィンドウを使用した XML メッセージ要素のデータ・ソースの定義 (Oracle Developer のフォーム)」に、iMessage を使用して XML メッセージを定義する方法を示します。この画面は、対応付けられたソースの SQL 問合せのみでなく、XML メッセージ要素および構造も示しています。

iMessage を使用して独自の XML メッセージ・セットを定義するには、次のように、いくつかの手順があります。

- **メッセージの定義**

メッセージは、すべての要素（属性）およびその構造関係を指定することによって定義できます。必須またはオプションの指定、データの最大長、デフォルト値など、その他の制約も指定できます。

■ メッセージの詳細の追加

Type フィールドです。Internet Message Studio は、アプリケーション・イベントの定義にも使用できます。メッセージとイベントの主な違いは、メッセージはアプリケーション・システム間の通信に使用されるのに対し、イベントは、ビジネス・オブジェクトの状態変化をブロードキャストまたはマルチキャストするために使用されます。さらに、Internet Message Studio は、タイマー・メッセージの定義にも使用できます。

Internet Message Studio を使用して定義されたイベントは、外部アプリケーション・システムおよび内部アプリケーション・システムの両方に公開されます。内部アプリケーションは、「event Publisher」画面または前述で定義された API を介して PL/SQL コールバック・プロシージャを登録でき、イベントが公開されたときに実行されます。定義ごとの外部アプリケーションはコールバック・プロシージャを登録しませんが、リモート・システムへ公開されたイベントを伝えるために実行するアダプタを持ちます。外部アプリケーションは、デフォルトのサブスクライバ画面を使用して、イベントに対してプロシージャを登録できます。内部アプリケーションの例は、単一の Oracle インスタンスで実行する Oracle の SDP および Installed Base です。

■ 説明

メッセージが使用されるコンテキストを説明します。

■ 表示名

メッセージを説明する名前です。

■ メッセージ要素の追加

■ メッセージ構造の構築

メッセージ構造は、メッセージ要素の階層関係を定義します。事前定義された要素のみがこの階層の一部になります。メッセージ構造の構築の詳細は、ユーザーズ・ガイドを参照してください。メッセージ構造は、ルートが最上位の要素である逆ツリーとして参照できます。

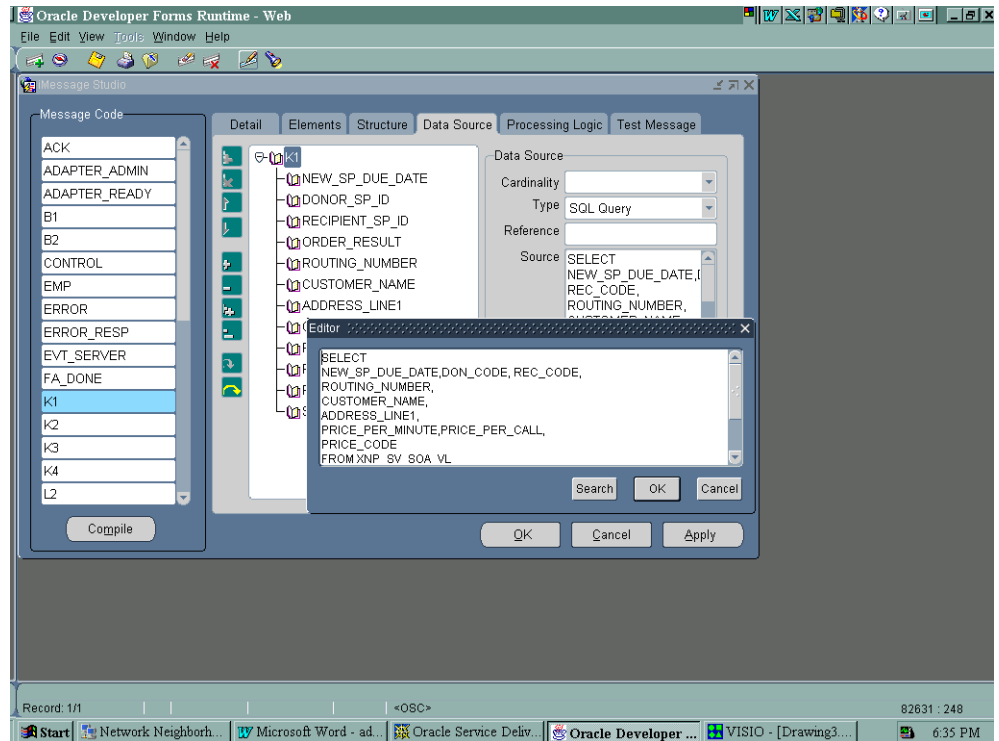
■ ルート要素

デフォルトでは、Internet Message Studio にはルート要素として「MESSAGE」があり、定義されているメッセージはルート要素の子です。ルート要素は参照できず、Internet Message Studio によって暗黙的に定義されることに注意してください。ルート要素は削除しないでください。構造にない要素はメッセージには現れません。

■ データ・ソースの定義

メッセージを定義する次の手順は、メッセージ要素のデータ・ソースを定義することです。メッセージ要素は、PL/SQL ファンクション・コールおよび SQL 問合せから値を取得できます。さらに、データは SDP Order パラメータ、SDP Work Item パラメータまたは Fulfillment Action パラメータとして取得できます。[表 11-1](#) を参照してください。

図 11-1 iMessage のデータ・ソース・ウィンドウを使用した XML メッセージ要素のデータ・ソースの定義 (Oracle Developer のフォーム)



サポートされているデータ型には、次のものがあります。

- PL/SQL ファンクション

PL/SQL ファンクションは実行時に実行されて、メッセージ要素の値を取得できます。指定された PL/SQL ファンクション・コールは、定義されたメッセージ要素のどれを参照しても引数を渡すことができます。また、PL/SQL ファンクションは、選択された列（これらの列は、上位レベルのメッセージ要素でデータ・ソースとして定義されています）のどれでも参照できます。ファンクションは、ユーザー・インタフェースのソース・フィールドで指定される必要があり、戻り型は、メッセージ要素に指定された型と同じである必要があります。

- SQL 問合せ

SQL 問合せは、メッセージ要素のデータを導出するために使用できます。SQL 問合せの列は、他のメッセージ要素の参照として使用でき、これらのメッセージ要素は、ツリー階層の下位レベルで定義されています。

その他のデータ型は SDP Order パラメータ、SADP Work Item パラメータおよび SDP Fulfilment Action パラメータです。

第 V 部

Oracle の XML アプリケーションの開発

第 V 部では、B2B XML アプリケーションを構築する方法を説明します。第 12 章では、様々なデバイスに同じ情報を表示する方法を説明します。

第 V 部の他の章では、JDeveloper および Internet File System (iFS) を使用して XML ベースのアプリケーションを構築する方法を説明します。

第 V 部に含まれる章は、次のとおりです。

- 第 12 章「B2B XML アプリケーション: ステップ・バイ・ステップ」
- 第 13 章「JDeveloper を使用した Oracle の XML アプリケーションの作成」
- 第 14 章「Internet File System (iFS) を使用した XML アプリケーションの作成」
- 第 15 章「XML を使用したリッチ・メディア管理用の n 層アーキテクチャの構築: ArtesiaTech」

B2B XML アプリケーション： ステップ・バイ・ステップ

この章の内容は次のとおりです。

- B2B XML アプリケーションの概要
- B2B XML アプリケーション実行の要件
- B2B XML アプリケーションの構築の概要
- XML にデータを変換する理由
- アドバンスト・キューイング (AQ) を使用する理由
- B2B XML アプリケーション: 主なコンポーネント
- B2B XML アプリケーションを実行するためのタスクの概要
- XML B2B アプリケーション: データベース・スキーマの設定
- B2B XML アプリケーション: データ交換フロー
- 小売業者とサプライヤ間のトランザクション
- B2B XML アプリケーションの実行: 詳細手順
- XSL および XSL 管理スクリプト
- XSL 処理および管理スクリプト
- B2B XML アプリケーションで使用する他のスクリプト
- 小売業者のスクリプト
- AQ Broker Transformer およびアドバンスト・キューイングのスクリプト
- サプライヤのスクリプト

B2B XML アプリケーションの概要

この章では、デモ B2B XML アプリケーション構築に必要なすべての手順およびスクリプトについて説明します。

B2B XML アプリケーション実行の要件

B2B XML アプリケーション構築および実行の要件は、次のとおりです。

- クライアント：
 - オペレーティング・システム : Windows NT
このアプリケーションに使用される 3 つのバッチ・ファイル（拡張子は .bat）は、Windows 固有です。ただし、これらのファイルを UNIX 用シェル・スクリプトで書き換えることはできます。
 - ツール : JDeveloper 3.1 以上（208MB）
 - XML エディタおよび XSL エディタ : すべての種類のエディタ
テキスト・エディタも、どの種類でも使用できます。
 - ブラウザ : Internet Explorer 5.0 や Netscape 5 以降などのブラウザ、および HandWeb などの PDA ブラウザ
- 中間層：
 - 開発環境には、513KB が必要です。
 - ランタイム環境には、135KB が必要です。
 - XSQL Servlet（XML Parser for Java や XSU for Java など）
 - HTTP リスナー
- サーバー：
 - Oracle および Java 対応のすべてのサーバー（Oracle8i リリース 8.1.7 以上など）

B2B XML アプリケーションの構築の概要

この XML アプリケーションおよびデモでは、コンテンツ管理および B2B メッセージ機能の実装を示しています。アプリケーションの主なトランザクションは、次のとおりです。

- 小売業者 (R) がブラウザ、携帯電話、PDA (パーソナル・デジタル・アシスタント) などのデバイスから発注します。
- サプライヤ (S) に、受注の受信が通知されます。サプライヤは、在庫および小売業者の貸方を確認した後、「Ship」ボタンをクリックします。
- 小売業者およびサプライヤが、注文の出荷状態をデバイスで参照します。

問題点

小売業者 (R) は複数のサプライヤ (S) の商品を自動発注する必要があり、すべてのデバイスで注文状態を参照できるようにする必要があります。

ソリューション

このソリューションとして、次のものを実装します。

- Oracle の XML コンポーネント
小売業者の Web サイトから受信した HTML (または他の形式) の注文データを XML ドキュメントに変換します。
- Oracle8i データベース
このソリューションでは、小売業者とサプライヤの両方が Oracle8i データベースにデータを格納していると想定しています。
- AQ Broker Transformer
この AQ アプリケーションは、小売業者とサプライヤ間の注文フローを管理します。小売業者は、注文を AQ キューに入れて送信します。関心があるサプライヤは、キューから注文を取得 (読み込みまたはデキュー) します。AQ は、注文フローに関する情報を抽出するためにも使用されます。それぞれの注文は、XML メッセージになります。このメッセージは、小売業者とサプライヤの両方が認識できる形式に変換されます。

タスク

図 12-2 に、主なタスクを示します。

1. 小売業者がブラウザ、PDA または携帯電話から発注します。
2. 小売業者が注文を検証すると、XSQL Servlet によって注文が XML に変換されます。
3. 小売業者アプリケーションが、AQ Broker に XML 注文を送ります。
4. AQ メッセージ機能によって、XML 注文データが送信されます。小売業者には、注文状態が「Pending」と表示されます。AQ Broker は、サブライヤが理解できる形式に XML 注文を再フォーマットします。
5. サブライヤ・アプリケーションが、サブライヤのデータベースに注文を挿入します。
6. サブライヤ・アプリケーションが注文を解析し、注文が受信され、処理待機中であることをサブライヤに通知します。
7. サブライヤが、サブライヤ・アプリケーション画面上の「Shipped」ボタンを押すと、AQ メッセージ機能によって AQ Broker に XML 注文状態データが戻されます。AQ Broker は、戻された XML 注文状態を、小売業者が認識できる形式に変換します。
8. サブライヤが、再フォーマットされた XML 注文状態メッセージを受信します。小売業者アプリケーションは、新しい注文状態を使用して小売業者のデータベースを更新します。小売業者が参照する注文状態が、「Shipped」に変わります。

詳細なタスク、参照する画面および使用するスクリプトについては、12-35 ページの「[B2B XML アプリケーションの実行: 詳細手順](#)」および図 12-2「[B2B XML アプリケーション: 主なコンポーネント](#)」を参照してください。

使用する XML および Oracle コンポーネント

- XSQL Servlet (XML SQL Utility (XSU)、XML Parser for Java および XSLT Processor を含む)
- Oracle8i

使用するツール

- JDeveloper

注意： 事前作成された（静的）XML ドキュメントは、この B2B XML アプリケーションでは使用されません。このアプリケーションのすべての XML ドキュメントは、データベースから動的に生成されます。

XML にデータを変換する理由

小売業者とサプライヤは、異なる形式を使用します。

小売業者からの異なる形式の注文フォームを、すべてのサプライヤが認識して処理できるように、小売業者の注文データは XML に変換されます。

サプライヤは、注文状態および受取り通知データに異なる形式を使用します。すべての小売業者が注文状態および受取り通知を認識できるように、このデータは XML に変換されます。

注意： このソリューションでは、規定の 2 つの顧客注文ドキュメント形式の限定セットを使用します。

- **小売業者の注文データ：**小売業者のデータベース R に格納される注文データは、適切な XSL スタイルシートを使用して、AQ Broker によって特定のサプライヤが認識する形式に変換されます。
- **サプライヤの注文状態データ：**このデータは、適切な XSL スタイルシートを使用して、AQ Broker によって特定の小売業者が認識する形式に変換されます。

注意： Transformer API および対応付けられた表は、小売業者やサプライヤのデータベースなど、どこにでも格納できます。

図 12-1 に、小売業者とサプライヤ間のトランザクションの全体的な流れを示します。小売業者が注文を入力します。

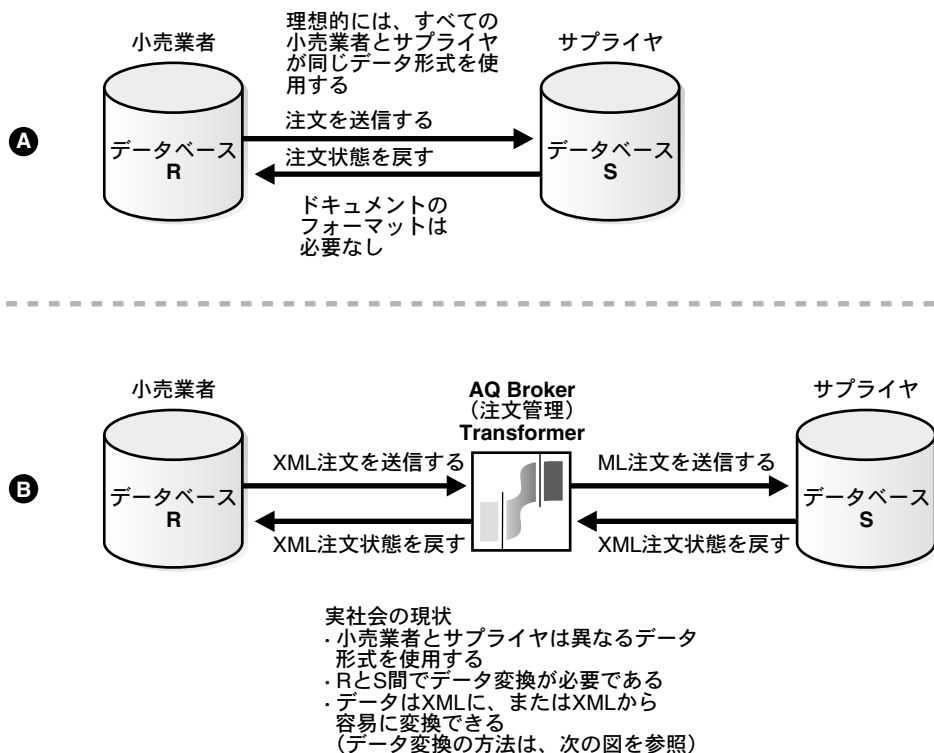
- すべての小売業者およびサプライヤの注文ドキュメント形式が同じという理想的な場合には、処理は A に示されるとおりに簡単になります。
- 実際は、各小売業者およびサプライヤの注文ドキュメント形式が異なることがほとんどです。これらのトランザクションは、データを XML へ変換することで透過的に行うことができます。データ形式に XSL スタイルシートを適用すると、デバイスに関係なく複数の形式にカスタマイズして表示できます。

アドバンスト・キューイング（AQ）を使用する理由

このアプリケーションで AQ を使用すると、次のようなメリットがあります。

- 小売業者からサプライヤへの注文フロー、およびサプライヤから小売業者への注文状態の更新および受取り通知を管理できます。
- 小売業者とサプライヤが区別されるため、すべての小売業者が同じキューで注文し、すべてのサプライヤがその同じキューから単純に注文を取得するようにできます。多対多の場合の実装が単純になります。
- 処理中の注文に関する情報を抽出できます。

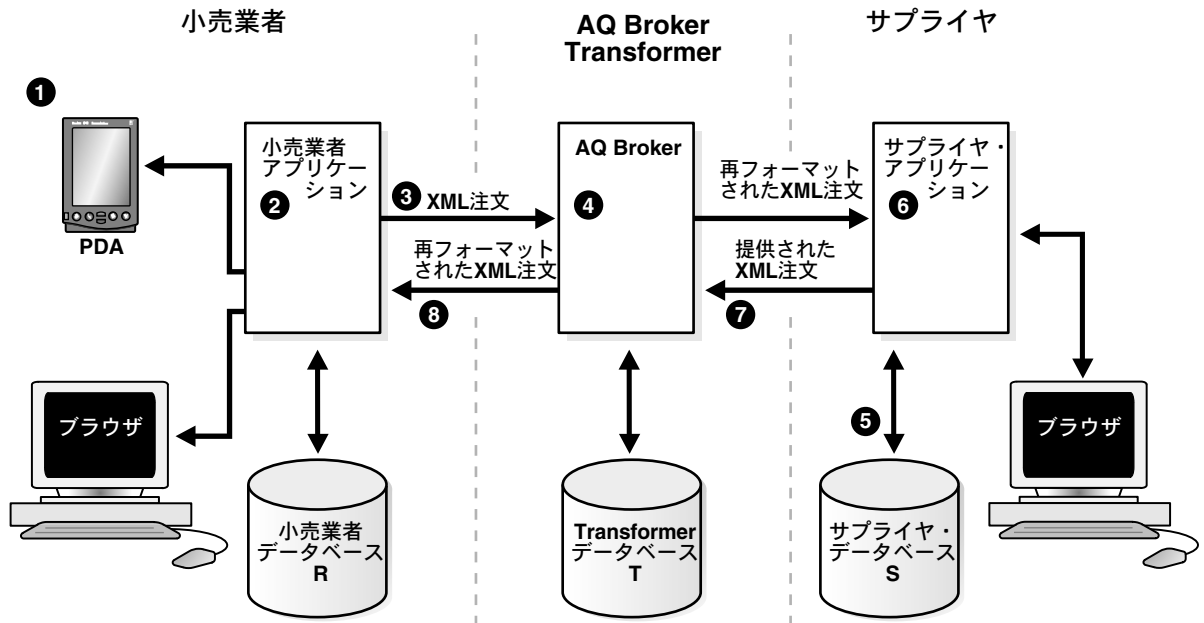
図 12-1 XML にデータを変換する理由：すべてのサプライヤが小売業者の注文データを認識でき、すべての小売業者がサプライヤの受注状態および受取り通知を認識できる



B2B XML アプリケーション：主なコンポーネント

図 12-2 に、B2B XML アプリケーションで使用する主なコンポーネントを示します。小売業者がサプライヤに商品を注文し、サプライヤから商品出荷の確認を受け取ります。このプロセスに関するトランザクションの詳細は、図 12-5 を参照してください。

図 12-2 B2B XML アプリケーション：主なコンポーネント



B2B XML アプリケーションを実行するためのタスクの概要

B2B XML アプリケーション構築に使用するスキーマは、[図 12-3](#) に示しています。

B2B XML アプリケーションを実行するには、次のタスクを実行します。

- 1. [B2B XML アプリケーションの実行のための環境設定](#)
- 2. [B2B アプリケーションの実行](#)
- 3. [B2B アプリケーション・セッションの終了](#)

ブラウザ上で B2B XML アプリケーションを実行する詳細は、12-35 ページの「[B2B XML アプリケーションの実行：詳細手順](#)」を参照してください。小売業者およびサプライヤが参照する典型的な画面も示しています。

図 12-3 B2B XML の小売業者（顧客）およびサプライヤのスキーマ

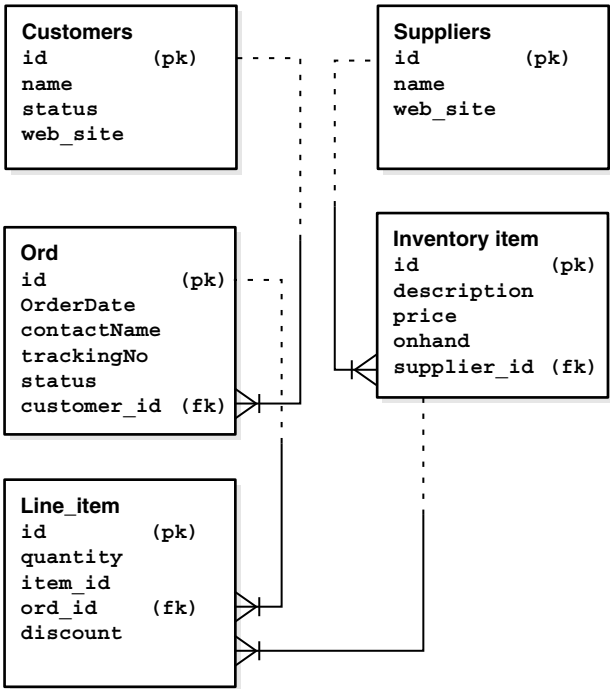
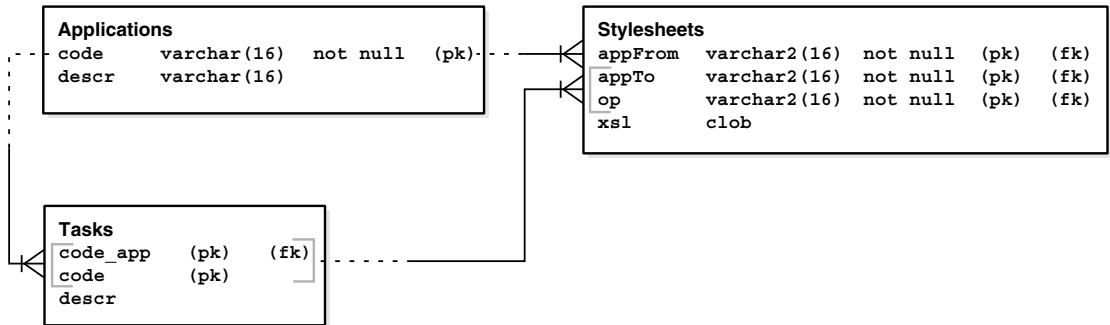


図 12-4 B2B XML の AQ Broker スキーマ : スタイルシート



B2B XML アプリケーションの実行のための環境設定

1. Apache または他の Web サーバーを起動します。
2. Internet Explorer 5 などのブラウザを起動します。
3. ログインします。
4. B2B XML アプリケーションの実行に必要なすべてのスキーマを設定するには、次の手順に従います。

小売業者およびサプライヤのスキーマを作成します。12-7 ページの「[B2B XML アプリケーション: 主なコンポーネント](#)」を参照してください。

- 任意の方法でデータベースに接続します。
 - **buildAll.sql** を実行します。スクリプトによって、要求されたユーザーを作成するためのシステム・パスワードが要求されます。
5. AQ スキーマを作成します。
 - SQL スクリプト **mkAQUser.sql** を任意のマシン上で実行します。
 - aqMessBrok/aqMessBrok として接続し、スクリプト **mkQ.sql** を実行します。
 6. XSL 表を作成します。
 - 接続状態のままで、スクリプト **mkSSTables.sql** を実行します。
 - **setup.sql** を実行して、データベースに XSL スタイルシートをインストールします。
 - 次の手順で説明されているとおりに接続を変更してから、GUIStylesheet Java クラスを実行してテストします。

7. 接続を変更します。
 - 次のファイルで JDBC 接続パラメータを変更します。
 - * AppCste.java
 - * retail.bat
 - * supplier.bat
 - * PlaceOrder.xsql
 - 最後に XSQLConfig.xml を変更して、retailer/retailer で接続する retail という名前の接続を作成します。
 - すべてのファイルを再コンパイルしてから、作業を続けます。
 8. **reset.sql** スクリプトを実行して AQ 環境をリセットし、B2B XML アプリケーションを実行します。
 9. プローカ、サプライヤおよび小売業者用の 3 つのバッチ・ファイルを変更してから実行します。
 - バッチ・ファイルを変更します。主に 3 つの使用ファイルがあり、次のバッチ・ファイルから起動します。
 - * Broker.bat (Messge Broker 用)
 - * Supplier.bat (サプライヤ用)
 - * Retail.bat (小売業者用)
- まず、次に示すとおりバッチ・ファイルを独自の環境用に変更します。
- * **verbose:** Y または TRUE に設定すると、受信したメッセージに関する多くの詳細情報が提供されます。
 - * **step:** Y または TRUE に設定すると、各処理手順の後でユーザーに [Enter] を押すように要求します。手順に数値がある場合、その数値は、各手順間での作業継続前の待ち時間（1000 分の 1 秒単位）とみなされます。

Retail.bat および Supplier.bat は、対象データベースに接続するための URL を示す -dbURL パラメータも受け入れます。デフォルトの URL は、jdbc:oracle:thin:@localhost:1521:ORCL です。

B2B アプリケーションの実行

1. **broker.bat**、**supplier.bat** および **retailer.bat** を実行します。
2. **GUIStyleSheet.class** を実行して、StyleSheet ユーティリティを確認します。

これらのスタイルシートは、受信するドキュメントを処理するために Broker が使用します。

ブラウザ上の画面表示などを含む、B2B XML アプリケーション実行の詳細は、12-35 ページの「[B2B XML アプリケーションの実行：詳細手順](#)」を参照してください。

B2B アプリケーション・セッションの終了

1. B2B XML アプリケーションを終了するには、Java クラス **stopAllQueues**、または **stopQ.bat** スクリプトを実行します。
2. Apache またはご使用の Web サーバーを停止します。

XML B2B アプリケーション：データベース・スキーマの設定

次のスキーマのスクリプトは、実行する順序で説明します。

- 小売業者およびサプライヤのスキーマの作成および構築
 - SQL の例 1: 小売業者およびサプライヤの環境設定 - BuildAll.sql
このスクリプトは、次のスクリプトをコールします。
 - SQL の例 2: 小売業者とサプライヤ間のスキーマの作成および移入 - BuildSchema.sql
- AQ 環境およびキュー表の作成
 - SQL の例 3: AQ の環境設定 - mkAQUser.sql
 - SQL の例 4: AQ キュー作成スクリプトのコール - mkQ.sql
このスクリプトは、次のスクリプトをコールします。
 - * SQL (PL/SQL) の例 5: AppOne_QTab 表の作成 - mkQueueTableApp1.sql
 - * SQL (PL/SQL) の例 6: AppTwo_QTab 表の作成 - mkQueueTableApp2.sql
 - * SQL (PL/SQL) の例 7: AppThree_QTab 表の作成 - mkQueueTableApp3.sql
 - * SQL (PL/SQL) の例 8: AppFour_QTab 表の作成 - mkQueueTableApp4.sql
- XSL Stylesheet 表を含む Broker スキーマの作成
 - SQL の例 9: Broker スキーマの作成 - mkSSTables.sql.
このスクリプトは、次のスクリプトをコールします。
 - SQL (PL/SQL) の例 10: CLOB への XSL データ入力および Broker スキーマの移入 - setup.sql
- 環境のクリーン・アップおよびアプリケーション再実行の準備
 - SQL の例 11: キュー・アプリケーションの停止と削除およびキュー・アプリケーションの起動 - reset.sql

SQL のコール順序

次に、SQL のコール順序の例を示します。各ファイルの拡張子 .sql は省略しています。「<---」はコールを意味します。たとえば「BuildAllsql <---- BuildSchema」は、BuildAllsql が BuildSchema をコールするという意味です。

- BuildAll.sql <---- BuildSchema.sql
- mkAQuser.sql
- mkQ.sql
 - <---- mkQueueTableApp1
 - <---- mkQueueTableApp2
 - <---- mkQueueTableApp3
 - <---- mkQueueTableApp4
- mkSSTables.sql <---- setup.sql
- reset.sql
 - <---- stopQueueApp1
 - <---- stopQueueApp2
 - <---- stopQueueApp3
 - <---- stopQueueApp4
 - <---- dropQueueApp1
 - <---- dropQueueApp2
 - <---- dropQueueApp3
 - <---- dropQueueApp4
 - <---- createQueueApp1
 - <---- createQueueApp2
 - <---- createQueueApp3
 - <---- createQueueApp4
 - <---- startQueueApp1
 - <---- startQueueApp2
 - <---- startQueueApp3
 - <---- startQueueApp4

小売業者およびサプライヤのスキーマの作成および構築

これらのスキーマのスクリプトは、小売業者およびサプライヤの環境、ユーザー、表領域、割当て制限などを設定します。スキーマの作成および移入も行います。

- [SQL の例 1: 小売業者およびサプライヤの環境設定 - BuildAll.sql](#)

このスクリプトは、次のスクリプトをコールします。

- [SQL の例 2: 小売業者とサプライヤ間のスキーマの作成および移入 - BuildSchema.sql](#)

SQL の例 1: 小売業者およびサプライヤの環境設定 - BuildAll.sql

BuildAll.sql は、小売業者およびサプライヤのスキーマの環境を設定します。このスクリプトは、小売業者およびサプライヤのスキーマを作成してからこれらのスキーマにデータを移入する BuildSchema.sql をコールします。

```
--
-- buildall.sql は、すべてのスキーマを作成します。
--
accept sysPswd prompt 'Enter the system password
> ' hide
accept cStr      prompt 'Enter the connect string if any, including '@' sign (ie
@atp-1) > '
connect system/&sysPswd&cStr
drop user retailer cascade
/
drop user supplier cascade
/
col tablespace_name head "Available Tablespaces"
select tablespace_name from dba_tablespaces
/
prompt
accept userTbsp prompt 'What is the DEFAULT Tablespace name ? > '
accept tempTbsp prompt 'What is the TEMPORARY Tablespace name ? > '
prompt

create user retailer identified by retailer
default tablespace &userTbsp
temporary tablespace &tempTbsp
quota unlimited on &userTbsp
/
grant connect, resource, create any directory to retailer
```

```

/
create user supplier identified by supplier
default tablespace &userTbsp
temporary tablespace &tempTbsp
quota unlimited on &userTbsp
/
grant connect, resource, create any directory to supplier
/
prompt Now populating Supplier, hit [Return]
pause
connect supplier/supplier&cStr
@buildSchema
prompt Now populating Retailer, hit [Return]
pause
connect retailer/retailer&cStr
@buildSchema
prompt done !

```

SQL の例 2: 小売業者とサプライヤ間のスキーマの作成および移入 - BuildSchema.sql

BuildSchema.sql は BuildAll.sql からコールされます。このスクリプトは、小売業者およびサプライヤのスキーマを作成、移入および構築します。

このスクリプトは、次の 5 つの表を作成および移入します。

- Customers
- Suppliers
- Inventory_item
- Ord
- Line_item

このスキーマの図は、[図 12-3](#) を参照してください。

```

--
-- buildSchema.sql は、B2B XML アプリケーション用のすべての表を削除してから再作成します。
--
drop trigger line_item_insert_trigger;
drop table line_item;
drop table ord;
drop table customer;
drop table inventory_item;

```

```
drop table supplier;
drop sequence ord_seq;
drop sequence customer_seq;
drop sequence line_item_seq;
drop sequence supplier_seq;
drop sequence inventory_item_seq;

prompt
prompt Creating sequences...
prompt
prompt
prompt Creating sequence ORD_SEQ
create sequence ord_seq start with 101;

prompt Creating sequence CUSTOMER_SEQ
create sequence customer_seq start with 201;

prompt Creating sequence LINE_ITEM_SEQ
create sequence line_item_seq start with 1001;

prompt Creating sequence SUPPLIER_SEQ
create sequence supplier_seq start with 301;

prompt Creating sequence INVENTORY_ITEM_SEQ
create sequence inventory_item_seq start with 401;

prompt
prompt
prompt Creating tables...
prompt
prompt
--
-- ***** CUSTOMERS 表を作成します。 *****
--
prompt Creating table CUSTOMER
create table customer(
  id          number,
  name        varchar2(30),
  status      varchar2(8),
  web_site    varchar2(40),
  constraint  customer_pk
              primary key (id)
);
--
```

```
-- ***** SUPPLIERS 表を作成します。*****
--
prompt Creating table SUPPLIER
create table supplier(
    id            number,
    name          varchar2(30),
    web_site      varchar2(40),
    constraint
        supplier_pk
        primary key (id)
);
--
-- ***** INVENTORY_ITEM 表を作成します。*****
--
prompt Creating table INVENTORY_ITEM
create table inventory_item(
    id            number,
    description    varchar2(30),
    price          number(8,2),
    onhand         number,
    supplier_id    number,
    constraint
        inventory_item_pk
        primary key (id),
    constraint
        supplied_by
        foreign key (supplier_id) references supplier
);
--
-- ***** ORD 表を作成します。*****
--
prompt Creating table ORD
create table ord (
    id            number,
    orderDate      date,
    contactName    varchar2(30),
    trackingNo     varchar2(20),
    status         varchar2(10),
    customer_id    number,
    constraint
        ord_pk
        primary key (id),
    constraint
        order_placed_by
        foreign key (customer_id) references customer
);
```

```
prompt Creating table LINE_ITEM
create table line_item(
  id          number,
  quantity    number,
  item_id     number,
  ord_id      number,
  discount    number,
  constraint
    line_item_pk
      primary key (id),
  constraint
    item_ordered_on
      foreign key (ord_id) references ord,
  constraint
    order_for_item
      foreign key (item_id) references inventory_item
);

prompt
prompt
prompt Inserting data...
prompt
prompt
prompt Inserting values into SUPPLIER and INVENTORY_ITEM
prompt

insert into supplier values( supplier_seq.nextval,'DELL','http://dell.com');
insert into inventory_item values( inventory_item_seq.nextval,'Optiplex GXPro',
1500, 27, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 7000', 2500,
49, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'PowerEdge 6300',
7500, 16, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 3000', 2500,
0, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval,'Inspiron 2000', 2500,
0, supplier_seq.currval );

insert into supplier values( supplier_seq.nextval, 'HP', 'http://hp.com');
insert into inventory_item values( inventory_item_seq.nextval, 'LaserJet 6MP', 899,
123, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, 'Jornada 2000', 450,
1198, supplier_seq.currval );
```

```
insert into inventory_item values( inventory_item_seq.nextval, 'HP 12C', 69, 801,
supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, 'LaserJet 2', 69, 3,
supplier_seq.currval );

insert into inventory_item values( inventory_item_seq.nextval, 'Jaz PCMCIA adapter',
125, 54, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, '8860 Digital phone',
499, 12, supplier_seq.currval );
insert into inventory_item values( inventory_item_seq.nextval, 'Jaz carrying bag',
20, 66, supplier_seq.currval );

insert into supplier values(supplier_seq.nextval, 'Intel', 'http://www.intel.com');
prompt Inserting values into CUSTOMER
prompt

insert into ord values( ord_seq.nextval, sysdate, 'George', 'AX'||ord_seq.currval,
'Pending', 201);
insert into line_item values (line_item_seq.nextval, 2, 410,ord_seq.currval, 0);
insert into line_item values (line_item_seq.nextval, 1, 402,ord_seq.currval, 0);
insert into line_item values (line_item_seq.nextval, 1, 406,ord_seq.currval, 0);

insert into ord values(ord_seq.nextval,sysdate,'Elaine','AX'||ord_seq.currval,
create trigger line_item_insert_trigger
before insert on line_item for each row
begin
select line_item_seq.nextval into :new.id from dual ;
end;
/

commit;
```

AQ 環境およびキュー表の作成

AQ スキーマ・スクリプトを、次のとおりに実行します。

- [SQL の例 3: AQ の環境設定 - mkAQUser.sql](#)
- [SQL の例 4: AQ キュー作成スクリプトのコール - mkQ.sql](#)

このスクリプトは、次のスクリプトをコールします。

- [SQL \(PL/SQL\) の例 5: AppOne_QTab 表の作成 - mkQueueTableApp1.sql](#)
- [SQL \(PL/SQL\) の例 6: AppTwo_QTab 表の作成 - mkQueueTableApp2.sql](#)
- [SQL \(PL/SQL\) の例 7: AppThree_QTab 表の作成 - mkQueueTableApp3.sql](#)
- [SQL \(PL/SQL\) の例 8: AppFour_QTab 表の作成 - mkQueueTableApp4.sql](#)

SQL の例 3: AQ の環境設定 - mkAQUser.sql

次の SQL スクリプトは、AQ を使用する環境を設定し、ユーザー aqMessBrok を作成してから、デフォルト表領域および一時表領域を作成し、AQ PL/SQL パッケージ dbms_aqadm および dbms_aq の実行権限を aqMessBrok へ付与します。

```
set ver off
set scan on
prompt Creating environment for Advanced Queuing
accept mgrPsw prompt 'Please enter the SYSTEM password
> ' hide
accept cStr prompt 'Please enter the the DB Alias if any, WITH the @ sign (ie
@Ora8i)> '
connect system/&mgrPsw&cStr

col tablespace_name head "Available Tablespaces"
select tablespace_name from dba_tablespaces
/
Prompt
accept userTbsp prompt 'What is the DEFAULT Tablespace name ? > '
accept tempTbsp prompt 'What is the TEMPORARY Tablespace name ? > '
prompt

prompt Creating aqMessBrok
create user aqMessBrok identified by aqMessBrok
default tablespace &userTbsp
temporary tablespace &tempTbsp
```



```
quota unlimited on &userTbsp
/
grant connect, resource, aq_administrator_role, create any directory to aqMessBrok
/
grant execute on dbms_aqadm to aqMessBrok
/
grant execute on dbms_aq to aqMessBrok
/
```

SQL の例 4: AQ キュー作成スクリプトのコール - mkQ.sql

このスクリプトは、AQ キュー表を作成するために次の 4 つのスクリプトをコールします。

```
@mkQueueTableApp1
@mkQueueTableApp2
@mkQueueTableApp3
@mkQueueTableApp4
```

SQL (PL/SQL) の例 5: AppOne_QTab 表の作成 - mkQueueTableApp1.sql

このスクリプトは mkQ.sql からコールされます。このスクリプトは、キュー表 1 AppOne_Qtab を作成するために、dbms_aqadm.create_queue_table プロシージャをコールします。

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppOne_QTab', queue_payload_
type => 'RAW');
```

SQL (PL/SQL) の例 6: AppTwo_QTab 表の作成 - mkQueueTableApp2.sql

このスクリプトは mkQ.sql からコールされます。このスクリプトは、キュー表 2 AppTwo_Qtab を作成するために、dbms_aqadm.create_queue_table プロシージャをコールします。

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppTwo_QTab', queue_payload_
type => 'RAW');
```

SQL (PL/SQL) の例 7: AppThree_QTab 表の作成 - mkQueueTableApp3.sql

このスクリプトは mkQ.sql からコールされます。このスクリプトは、キュー表 3 AppThree_Qtab を作成するために、dbms_aqadm.create_queue_table プロシージャをコールします。

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppThree_QTab', queue_
payload_type => 'RAW');
```

SQL (PL/SQL) の例 8: AppFour_QTab 表の作成 - mkQueueTableApp4.sql

このスクリプトは mkQ.sql からコールされます。このスクリプトは、キュー表 4 AppFour_Qtab を作成するために、dbms_aqadm.create_queue_table プロシージャをコールします。

```
execute dbms_aqadm.create_queue_table (queue_table => 'AppFour_QTab', queue_payload_
type => 'RAW');
```

XSL Stylesheet 表を含む Broker スキーマの作成

Stylesheet、Task および Application 表を作成し移入するには、次のスクリプトを実行します。

- [SQL の例 9: Broker スキーマの作成 - mkSSTables.sql](#)
このスクリプトは、次のスクリプトをコールします。
- [SQL \(PL/SQL\) の例 10: CLOB への XSL データ入力および Broker スキーマの移入 - setup.sql](#)

SQL の例 9: Broker スキーマの作成 - mkSSTables.sql

Broker スキーマを作成するには、mkSSTables.sql を実行します。このスクリプトは、次の 3 つの表を作成および移入します。

- Stylesheet
- Task
- Application

[図 12-4](#) に、このスキーマを示します。次にこのスクリプトは、setup.sql をコールします。

```
prompt Building Stylesheets management tables.
prompt Must be connected as aqMessBrok (like the borker)
accept cStr prompt 'ConnectionString (WITH @ sign, like @Ora8i) > '
connect aqMessBrok/aqMessBrok&cStr
```

```
drop table styleSheets
/
drop table tasks
/

drop table applications
/
create table applications
(
    code varchar2(16) not null,
```

```
        descr varchar2(256)
    )
/
alter table applications
    add constraint PK_APP
        primary key (code)
/
create table tasks
(
    code_app varchar2(16) not null,
    code      varchar2(16) not null,
    descr     varchar2(256)
)
/
alter table tasks
    add constraint PK_TASKS
        primary key (code_app,code)
/
alter table tasks
    add constraint TASK_FK_APP
        foreign key (code_app)
        references applications(code) on delete cascade
/
create table styleSheets
(
    appFrom varchar2(16) not null,
    appTo    varchar2(16) not null,
    op       varchar2(16) not null,
    xsl      clob
)
/
alter table styleSheets
    add constraint PK_SS
        primary key (appFrom,appTo,op)
/
alter table styleSheets
    add constraint SS_FK_FROM
        foreign key (appFrom)
        references applications(code)
/
alter table styleSheets
    add constraints SS_FK_TASK
        foreign key (appTo,op)
        references tasks(code_app,code)
/
@setup
```

SQL (PL/SQL) の例 10: CLOB への XSL データ入力および Broker スキーマの移入 - setup.sql

setup.sql は、Stylesheets 表の XSL 列 (CLOB) にスタイルシート・データをインストールします。このスクリプトは、loadlob プロシージャを作成します。このスクリプトは、PL/SQL パッケージ dbms_lob および dbms_output も使用します。

```
prompt Installing the stylesheets
-- cStr プロンプト 'ConnectionString (WITH @ sign, like @Ora8i) > ' を受け入れます。
-- aqMessBrok/aqMessBrok&cStr で接続します。
prompt Creating LoadLob procedure
create or replace procedure loadLob (imgDir in varchar2,
                                     fname in varchar2,
                                     app_From in varchar2,
                                     app_To in varchar2,
                                     oper in varchar2) as

    tempClob CLOB;
    fileOnOS BFILE := bfilename(imgDir, fname);
    ignore INTEGER;
begin
    dbms_lob.fileopen(fileOnOS, dbms_lob.file_readonly);
    select xsl
    into tempClob
    from StyleSheets S
    where s.APPFROM = app_From and
          s.APPTO = app_To and
          s.OP = oper
    for UPDATE;
    dbms_output.put_line('External file size is: ' || dbms_lob.getlength(fileOnOS));
    dbms_lob.loadfromfile(tempClob, fileOnOS, dbms_lob.getlength(fileOnOS));
    dbms_lob.fileclose(fileOnOS);
    dbms_output.put_line('Internal CLOB size is: ' || dbms_lob.getlength(tempClob));
exception
    When Others then
        dbms_output.put_line('Oooops : ' || SQLERRM);
end LoadLob;
/
show errors
set scan off

create or replace directory "LOB_DIR" as 'D:\xml817\references\olivier_new'
/
insert into applications values ('RETAIL', 'Origin')
/
insert into applications values ('SUPPLY', 'Destination')
```

```

/
insert into tasks values ('SUPPLY', 'NEW ORDER', 'Insert a new Order')
/
insert into tasks values ('RETAIL', 'UPDATE ORDER', 'Update an Order Status')
/

set serveroutput on
begin
insert into StyleSheets values ('RETAIL','SUPPLY','NEW ORDER',EMPTY_CLOB());
loadLob('LOB_DIR', 'one.xml', 'RETAIL','SUPPLY','NEW ORDER');
insert into StyleSheets values ('SUPPLY','RETAIL','UPDATE ORDER',EMPTY_CLOB());
loadLob('LOB_DIR', 'two.xml', 'SUPPLY','RETAIL','UPDATE ORDER');
exception
when others then
dbms_output.put_line('Error Occurred : ' || chr(10) || SQLERRM);
end;
/
commit
/

```

環境のクリーン・アップおよびアプリケーション再実行の準備

環境をクリーン・アップしてこのアプリケーションを再実行するには、reset.sql を実行します。

- [SQL の例 11: キュー・アプリケーションの停止と削除およびキュー・アプリケーションの起動 - reset.sql](#)

このスクリプトは、次の 16 個の PL/SQL スクリプトをコールします。

- [stopQueueApp1.sql](#)
- [stopQueueApp2.sql](#)
- [stopQueueApp3.sql](#)
- [stopQueueApp4.sql](#)
- [dropQueueApp1.sql](#)
- [dropQueueApp2.sql](#)
- [dropQueueApp3.sql](#)
- [dropQueueApp4.sql](#)
- [createQueueApp1.sql](#)
- [createQueueApp2.sql](#)

- [createQueueApp3.sql](#)
- [createQueueApp4.sql](#)
- [startQueueApp1.sql](#)
- [startQueueApp2.sql](#)
- [startQueueApp3.sql](#)
- [startQueueApp4.sql](#)

SQL の例 11: キュー・アプリケーションの停止と削除およびキュー・アプリケーションの起動 - reset.sql

reset.sql スクリプトは、まず stopQueueApp1 ～ 4 をコールして、4 つのすべてのキュー・アプリケーションを停止します。次に dropQueueApp1 ～ 4 をコールしてこれらのアプリケーションを削除してから、startQueueApp1 ～ 4 をコールして再起動します。

このスクリプトは、「Press [Return] to Exit」というプロンプトを表示します。

```
connect aqMessBrok/aqMessBrok
start stopQueueApp1
start stopQueueApp2
start stopQueueApp3
start stopQueueApp4
start dropQueueApp1
start dropQueueApp2
start dropQueueApp3
start dropQueueApp4
start createQueueApp1
start createQueueApp2
start createQueueApp3
start createQueueApp4
start startQueueApp1
start startQueueApp2
start startQueueApp3
start startQueueApp4
prompt Press [Return] to exit !
pause
exit
```

キューを停止する SQL スクリプト

次の 4 つのスク립トは `reset.sql` からコールされます。これらのスク립トは、キューを停止するために PL/SQL プロシージャ `dbms_aqadm.stop_queue` を使用します。

stopQueueApp1.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppOneMsgQueue');
```

stopQueueApp2.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppTwoMsgQueue');
```

stopQueueApp3.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppThreeMsgQueue');
```

stopQueueApp4.sql

```
execute dbms_aqadm.stop_queue(queue_name=>'AppFourMsgQueue');
```

キューを削除する SQL スクリプト

これらの 4 つのスク립トは `reset.sql` からコールされます。これらのスク립トは、キューを削除するために PL/SQL プロシージャ `dbms_aqadm.drop_queue` を使用します。

dropQueueApp1.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppOneMsgQueue');
```

dropQueueApp2.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppTwoMsgQueue');
```

dropQueueApp3.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppThreeMsgQueue');
```

dropQueueApp4.sql

```
execute dbms_aqadm.drop_queue (queue_name=>'AppFourMsgQueue');
```

キューを作成する SQL スクリプト

次の 4 つのスクリプトは `reset.sql` からコールされます。これらのスクリプトは、キューを作成するために PL/SQL プロシージャ `dbms_aqadm.create_queue` を使用します。

createQueueApp1.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppOneMsgQueue', queue_table=>'AppOne_QTab');
```

createQueueApp2.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppTwoMsgQueue', queue_table=>'AppTwo_QTab');
```

createQueueApp3.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppThreeMsgQueue', queue_table=>'AppThree_QTab');
```

createQueueApp4.sql

```
execute dbms_aqadm.create_queue (queue_name=>'AppFourMsgQueue', queue_table=>'AppFour_QTab');
```

キューを開始する SQL スクリプト

次の 4 つのスクリプトは `reset.sql` からコールされます。これらのスクリプトは、キューを開始するために PL/SQL プロシージャ `dbms_aqadm.start_queue` を使用します。

startQueueApp1.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppOneMsgQueue');
```

startQueueApp2.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppTwoMsgQueue');
```

startQueueApp3.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppThreeMsgQueue');
```

startQueueApp4.sql

```
execute dbms_aqadm.start_queue (queue_name=>'AppFourMsgQueue');
```


dropOrder.sql

この SQL スクリプトは、顧客 ID に従って小売業者 / サプライヤのデータベースの Customers 表から注文を削除します。

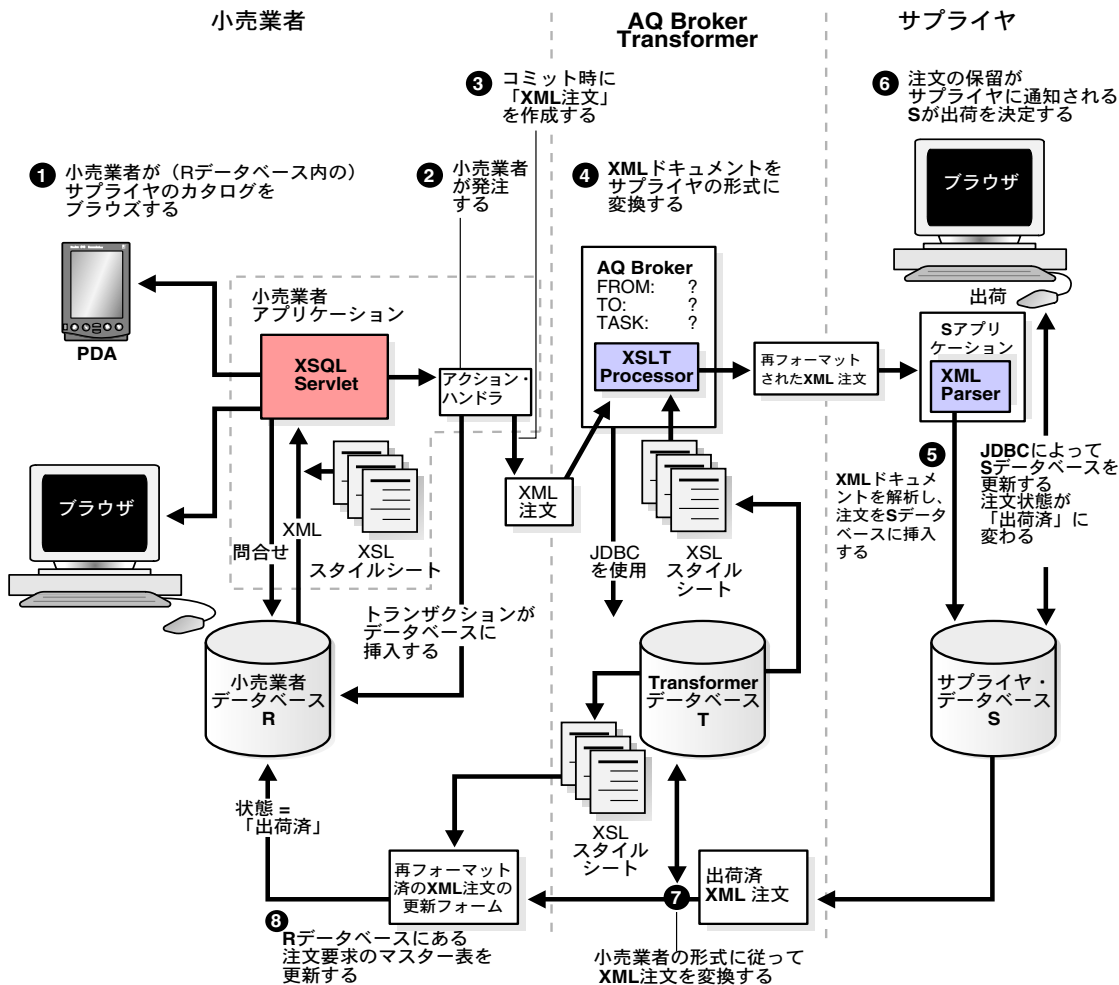
```
set ver off
accept CustName prompt 'Drop all for customer named > '

Delete LINE_ITEM I
Where I.ORD_ID in
(Select O.ID
 From ORD O
 Where O.CUSTOMER_ID in
 (Select C.ID
  From CUSTOMER C
   Where Upper(C.NAME) = Upper('&CustName')))
/
Delete ORD O
Where O.CUSTOMER_ID in
(Select C.ID
 From CUSTOMER C
  Where Upper(C.NAME) = Upper('&CustName'))
/
```

B2B XML アプリケーション：データ交換フロー

図 12-5 に、小売業者がサプライヤに商品を注文し、サプライヤから商品出荷の確認を受信する場合のトランザクションの詳細を示します。

図 12-5 企業間のデータ交換：XML と AQ を使用した、サプライヤへの小売業者の注文の送信およびサプライヤからの注文状態と受取り通知の受信



小売業者とサプライヤ間のトランザクション

図 12-5 に、小売業者とサプライヤ間のトランザクション・フローを示します。トランザクションは、次のとおりです。

1. 小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする
2. 小売業者が発注する
3. 小売業者が確認して注文を送信する
4. AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する
5. サプライヤ・アプリケーションが、再フォーマット済の受信 XML 注文ドキュメントを解析し、サプライヤ・データベースに注文を挿入する
6. サプライヤ・アプリケーションが保留注文をサプライヤに通知する
7. AQ Broker Transformer が小売業者の形式に従って XML 注文を変換する
8. 小売業者アプリケーションが Ord 表および Line_Item 表を更新する

トランザクションの詳細および B2B XML アプリケーションの実行方法については、12-35 ページの「[B2B XML アプリケーションの実行：詳細手順](#)」を参照してください。

小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする

次のとおり、小売業者のトランザクションが発生します。

1. 小売業者が XSQL Servlet を使用して、Web サイトにログインします。
2. 小売業者が、サプライヤのオンライン・カタログをブラウズし、商品および数量を選択します。

小売業者が発注する

小売業者は、発注するときに「Place Order」をクリックして注文およびコストを確認するか、「Give Up」をクリックして注文を取り消す必要があります。

小売業者が確認して注文を送信する

小売業者が「Place Order」をクリックして注文を確認すると、これによって注文データを含む XML ドキュメントが生成されます。小売業者アプリケーションは、AQ Broker Transformer アプリケーションを介して、この XML 注文ドキュメントをサプライヤに送信します。

XSQL Servlet のアクション・ハンドラ「[XSQL スクリプトの例 5: B2B プロセスの開始 - placeorder.xsql](#)」は、プロセス全体の主要なコンポーネントです。このハンドラによって、トランザクションが小売業者のデータベース表 Ord に確実に挿入されます。

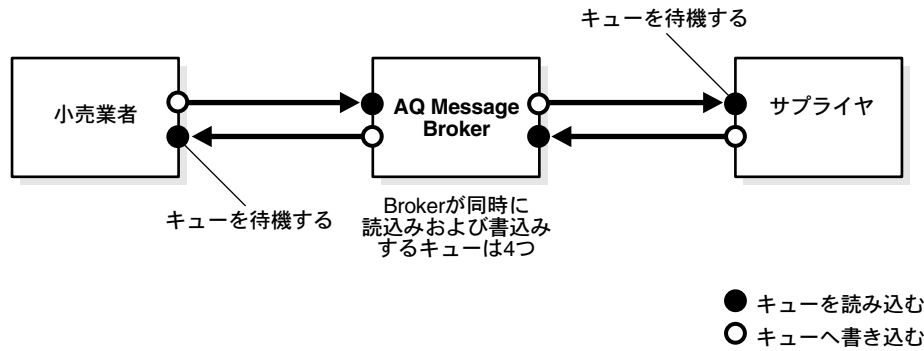
アクション・ハンドラは、AQ Broker Transformer への XML 注文の送信も行います。

AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する

AQ Broker Transformer が XML ドキュメントを受信すると、次の処理が行われます。

1. AQ Broker Transformer は、小売業者が注文を送信したという、読み込み用のキューを待機します。図 12-6 を参照してください。

図 12-6 B2B XML アプリケーション：AQ メッセージ機能フロー



2. AQ Broker が XML ドキュメントの注文メッセージを受信し、そのメッセージから次の情報を判断します。
 - FROM: メッセージの送信元（送信元の小売業者）
 - TO: メッセージの送信先（送信先のサプライヤ）
 - OPERATION または TASK: このメッセージを処理するために必要な操作

3. AQ Broker Transformer は Stylesheet 表を参照し、From、To および Task 基準に従って適切な XSL スタイルシートを選択します。スタイルシートは、Stylesheet 表の XSL 列の CLOB に格納されています。AQ Broker Transformer は、JDBC を介してデータベースおよびスタイルシートにアクセスします。
4. AQ Broker Transformer アプリケーションが、選択および取り出した XSL スタイルシートを、注文データを含む XML ドキュメントに適用するように XSLT Processor へ通知します。XSLT Processor は、再フォーマットされた XML 注文を出力します。
5. AQ Broker Transformer が、AQ を使用して、XML ドキュメントの書き込み用キューを送信先サプライヤに送信します。

注意： DTD (XML Schema) を使用する場合、DTD は AQ Broker フェーズで処理する前に適用されます。この例では、わかりやすいように、ドキュメントが常に同じ形式で送信されると想定しています。

図 12-4 に、AQ Broker Transformer が使用するスキーマを示します。

サプライヤ・アプリケーションが、再フォーマット済の受信 XML 注文ドキュメントを解析し、サプライヤ・データベースに注文を挿入する

サプライヤが、再フォーマットされた XML 注文ドキュメントを AQ Broker Transformer から受信すると、次の処理が行われます。

1. サプライヤは、小売業者からの注文が保留中であるという AQ Broker Transformer からのキューを待機します。サプライヤが AQ メッセージをデキューします。
2. サプライヤが XML ドキュメントを解析し、JDBC を介してサプライヤ・データベースに注文を挿入します。

サプライヤ・アプリケーションが保留注文をサプライヤに通知する

サプライヤ・アプリケーションがサプライヤのデータベースに XML ドキュメントを挿入すると、次の処理が行われます。

1. サプライヤ・アプリケーションがサプライヤに注文を通知します。注文状態は、「Pending」のままです。
2. サプライヤは、注文された商品の在庫有無および小売業者の貸方を確認後、商品の出荷を決定します。サプライヤが「Ship」をクリックします。
3. サプライヤ・アプリケーションが、サプライヤのデータベースの Ord 表の状態列を「Shipped」に更新します。

AQ Broker Transformer が小売業者の形式に従って XML 注文を変換する

1. AQ Broker Transformer は、サプライヤからの読み込み用キューを待機します。
2. AQ Broker Transformer は、XML 注文出荷ドキュメントを受信すると、Transformer データベースの Stylesheets 表を参照し、From、To および Task 基準に従って適切な XSL スタイルシートを選択します。スタイルシートは、Stylesheets 表の XSL 列の CLOB に格納されています。AQ Broker Transformer は、JDBC を介してデータベースおよびスタイルシートにアクセスします。
3. AQ Broker Transformer が、AQ を使用して、再フォーマット済の XML 注文更新ドキュメントの書き込み用キューを送信先の小売業者に送信します。

小売業者アプリケーションが Ord 表および Line_Item 表を更新する

1. 小売業者アプリケーションが、新しい「Shipped」状態情報によって小売業者のデータベースを更新します。Ord 表が更新されます。
2. 小売業者はどのデバイスからでもこの情報を参照できます。状態は、「Shipped」になっています。

B2B XML アプリケーションの実行 : 詳細手順

図 12-5 に、B2B XML アプリケーションのトランザクションおよびフローの詳細を示します。XML 注文ドキュメントは、AQ Broker Transformer を介して小売業者からサプライヤに送信され、小売業者に返送されます。

B2B XML アプリケーションは、12-8 ページの「[B2B XML アプリケーションを実行するためのタスクの概要](#)」で説明しているスキーマ作成スクリプトが実行されたことを確認してから実行してください。

アプリケーションの実行プロセスおよび方法を、次の手順で説明します。

1. 小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする
2. 小売業者が発注する
3. 「Validate」をクリックしてトランザクションをコミットし、小売業者アプリケーションが XML 注文を作成する
4. AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する
5. サプライヤ・アプリケーションが XML ドキュメントを解析し、サプライヤのデータベースに注文を挿入する
- 6a. サプライヤ・アプリケーションが保留注文をサプライヤに通知する
- 6b. サプライヤが商品を小売業者に出荷する
- 6c. サプライヤ・アプリケーションが AQ Broker に送信する新しい XML メッセージを生成する
7. AQ Broker Transformer が小売業者の形式に XML 注文を変換する
8. 小売業者アプリケーションが Ord 表を更新し、小売業者に新しい注文状態を表示する

1 小売業者がサプライヤのオンライン Hi-Tech Mall カタログをブラウズする

B2B XML アプリケーションの処理フローの詳細は、[図 12-5](#) を参照してください。

注意： この手順では、サプライヤのカタログは小売業者のデータベース内にあると想定しています。

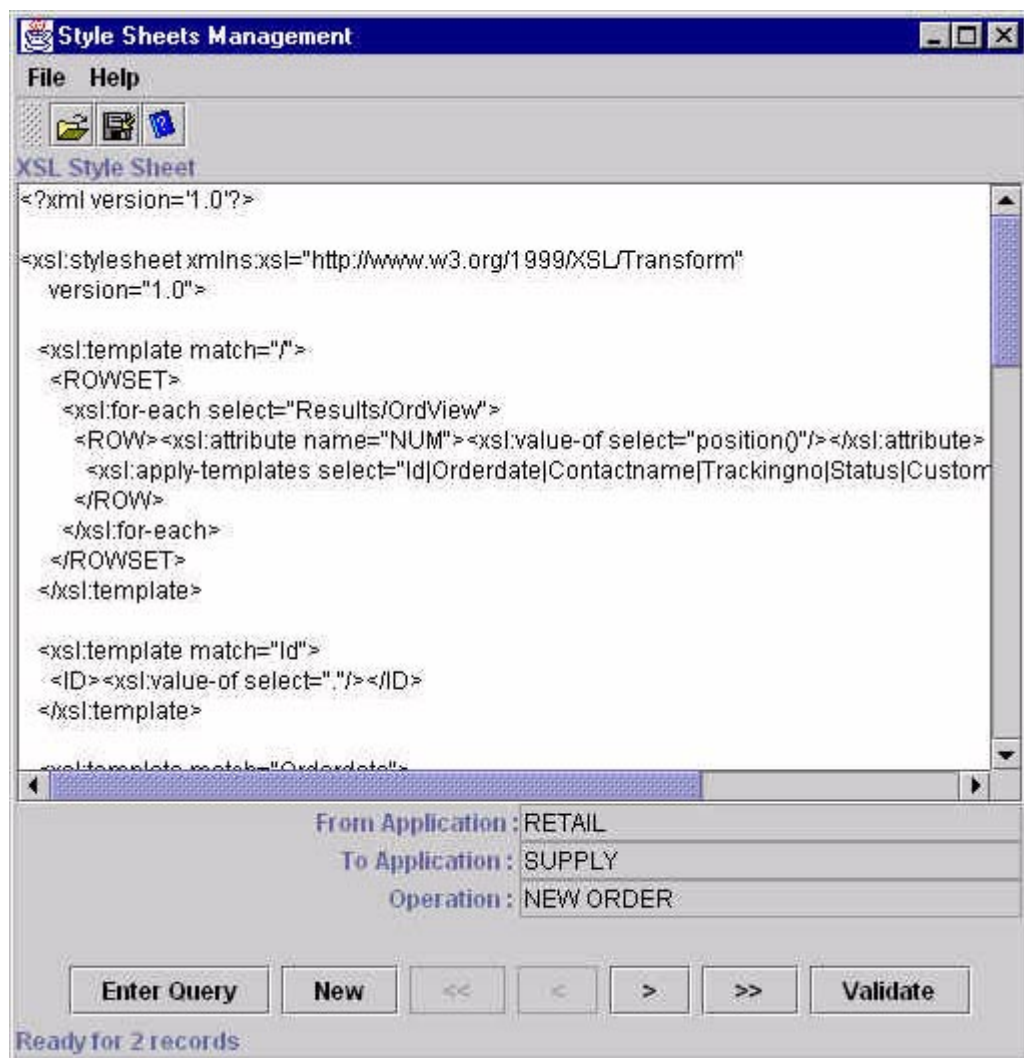
1. SS.bat を起動して、StyleSheet ユーティリティが動作することを確認します。

スタイルシートのバッチ・ファイル : SS.bat

```
@echo off
@echo Stylesheet Util
D:\jdev31\java\bin\java -mx50m -classpath "D:\xml817\references\olivier_new;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;
D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbsms\jlib\aqapi11.jar;
D:\Ora8i\rdbsms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip" B2BDemo.StyleSheetUtil.GUIStylesheet
```

このユーティリティを使用すると、スタイルシートが格納されている実際の表 Stylesheets をブラウズできます。これらのスタイルシートは、AQ Broker Transformer が受信したドキュメントを処理するために使用します。[図 12-7](#) を参照してください。

図 12-7 StyleSheet ユーティリティの確認



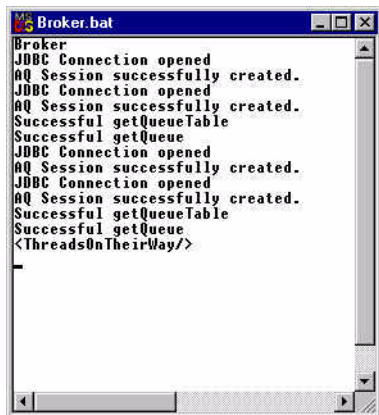
2. retailer.bat を実行して小売業者アプリケーションを起動します。図 12-8 を参照してください。

図 12-8 小売業者アプリケーションの起動



3. broker.bat を実行して AQ Broker Transformer アプリケーションを起動します。図 12-9 を参照してください。

図 12-9 AQ Broker Transformer アプリケーションの起動



4. `supplier.bat` を実行してサプライヤ・アプリケーションを起動します。図 12-10 を参照してください。

図 12-10 サプライヤ・アプリケーションの起動 : サプライヤ・ウォッチャ



小売業者アプリケーション、AQ Broker Transformer アプリケーションおよびサプライヤ・アプリケーション用の 3 つのバッチ・ファイルは、それぞれ次のとおりです。

retailer.bat

```
@echo off
@echo Retail Side
D:\jdev31\java\bin\java -mx50m -classpath "D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;
D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbs\jlib\aqapi11.jar;
D:\Ora8i\rdbs\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip" B2BDemo.Retailer.UpdateMaster -step=1000
-verbose=y -dbURL=jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL
```

broker.bat

```
@echo off
@echo Broker
D:\jdev31\java\bin\java -mx50m -classpath "D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdms\jlib\aqapi11.jar;
D:\Ora8i\rdms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip" B2BDemo.Broker.MessageBroker -step=1000
-verbose=y
```

supplier.bat

```
@echo off
@echo Supplier
D:\jdev31\java\bin\java -mx50m -classpath "D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdms\jlib\aqapi11.jar;
D:\Ora8i\rdms\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip" B2BDemo.Supplier.SupplierWatcher -step=1000
-verbose=y -dbURL=jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL
```

- 最後に、ブラウザ、Palm Pilot などの PDA、携帯電話または他のデバイスからクライアントを起動します。
- [小売業者]: ログインします。「Welcome!」画面が表示されます。[図 12-11](#) を参照してください。

XSQL スクリプト例 1: ログイン中のユーザー ID の確認: getlogged.xsql

```
<?xml version="1.0"?>

<!--
| 2 目のスクリプトがコールされます。
| データベース内でユーザーが認識されているかどうかを確認します。
| $ 作成者: olediou@us $
| $ 改訂番号: 1.1 $
+-->
<?xml-stylesheet type="text/xsl" media="HandHTTP" href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="HTML.xsl"?>

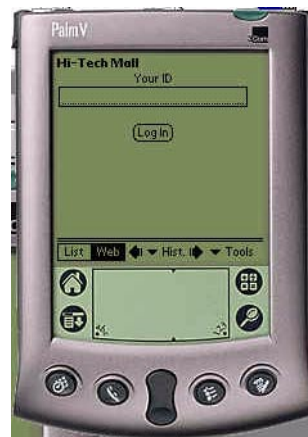
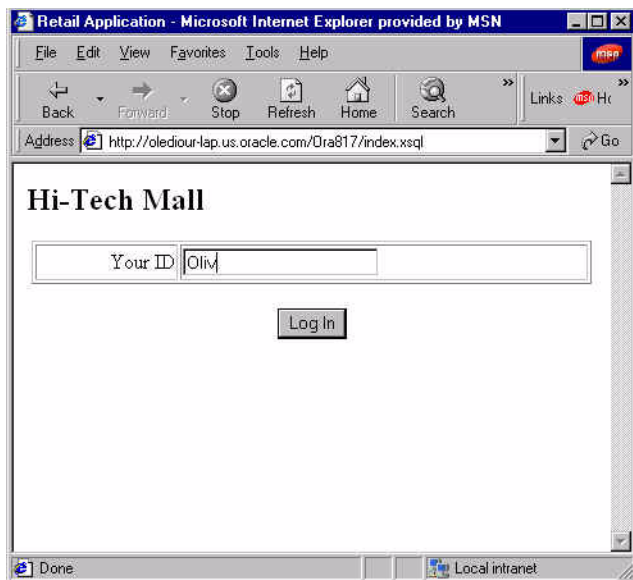
<loginResult xmlns:xsql="urn:oracle-xsql"
               connection="retail"
               custName="XXX">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <xsql:query tag-case="upper">
    <![CDATA[
      select C.ID, C.NAME
      from CUSTOMER C
      where Upper(C.NAME) = Upper('@custName')
    ]]>
    <xsql:no-rows-query>
      Select '@custName' as "unknown" from dual
    </xsql:no-rows-query>
  </xsql:query>
  <nextStep>inventory.xsql</nextStep>
  <returnHome>index.xsql</returnHome>

</loginResult>
```

この XSQL スクリプトは、次の XSL スクリプトをコールします。

- pp.xsl: 「[XSL スタイルシートの例 1: HTML への結果の変換 - html.xsl](#)」を参照してください。
- html.xsl: 「[XSL スタイルシートの例 2: Palm Pilot ブラウザ用の結果変換 - pp.xsl](#)」を参照してください。

図 12-11 [小売業者]: ブラウザまたは PDA からのログイン



7. [小売業者]: 「Please Enter the Mall」をクリックします。

XSQL スクリプト例 2: Hi-Tech Mall の最初の画面の表示 - index.xsql

```
<?xml version="1.0"?>

<!--
|   これは、アプリケーションのエントリ・ポイントです。
|   このスクリプトはデータベースにアクセスしないことに注意してください。
|   $Author: olediou@us $
|   $Revision: 1.1 $
+-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"      href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"       href="HTML.xsl"?>

<index xmlns:xsql="urn:oracle-xsql">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <form action="getLogged.xsql" method="post">
    <field type="text" name="custName" prompt="Your ID"/>
    <button type="submit" label="Log In"/>
  </form>
</index>
```

8. [小売業者]: 結果画面には、Hi-Tech Mall カタログの商品リストが表示されます。興味がある商品をクリックします。図 12-12 を参照してください。

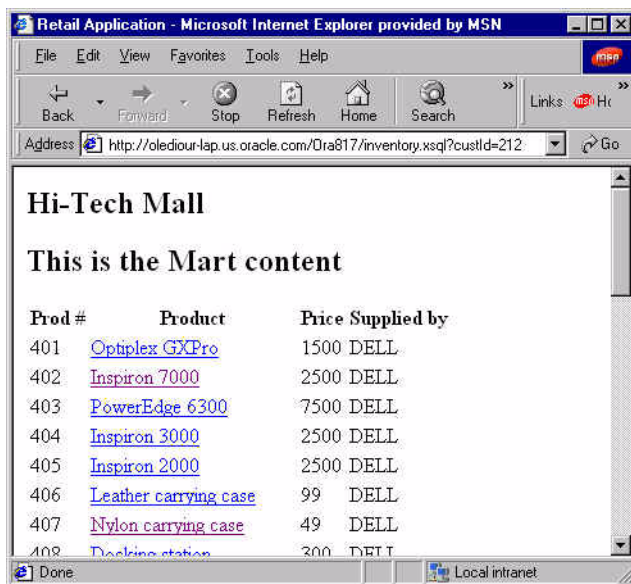
XSQL スクリプト例 3: カタログ商品のリスト - inventory.xsql

```
<?xml version="1.0"?>

<!--
|   これは、3 つ目にコールされるスクリプトです。
|   このスクリプトは、小売業者のデータベースからカタログを作成します。
|
|   $Author: olediou@us $
|   $Revision: 1.1 $
+-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"      href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"        href="HTML.xsl"?>

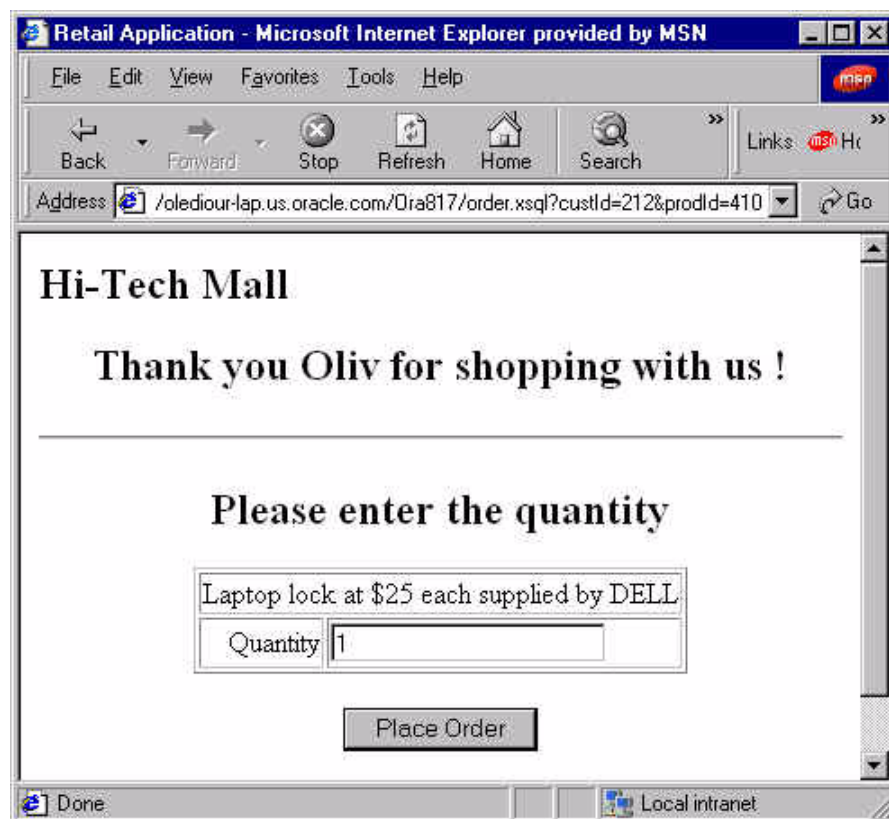
<inventory xmlns:xsql="urn:oracle-xsql"
            connection="retail"
            custId="000">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <form action="order.xsql" method="post">
    <hiddenFields>
      <xsql:include-param name="custId"/>
    </hiddenFields>
    <theMart>
      <xsql:query tag-case="upper">
        <![CDATA[
          select I.ID,
                 I.DESCRPTION,
                 I.PRICE,
                 S.NAME
          from INVENTORY_ITEM I,
               SUPPLIER S
          where I.SUPPLIER_ID = S.ID
        ]]>
      <xsql:no-rows-query>
        Select 'No items !' as "Wow" from dual
      </xsql:no-rows-query>
    </xsql:query>
    </theMart>
  </form>
  <returnHome>index.xsql</returnHome>
</inventory>
```

図 12-12 [小売業者]: Hi-Tech Mall (マート) カタログへのアクセス



9. [小売業者]: 必要な数量を入力し、「Place Order」ボタンをクリックします。図 12-13 を参照してください。

図 12-13 [小売業者]: 数量を入力し、「Place Order」ボタンをクリックする



10. [小売業者]: 「Go On」または「Give Up」をクリックします。図 12-14 を参照してください。

XSQL スクリプト例 4: 数量の入力 - order.xsql

```

<?xml version="1.0"?>
<!--
|   これは、4 つ目にコールされるスクリプトです。
|   このスクリプトは、数量を入力するプロンプトを表示します。
|
|   $Author: olediou@us $
|   $Revision: 1.1 $
+-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"      href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"       href="HTML.xsl"?>

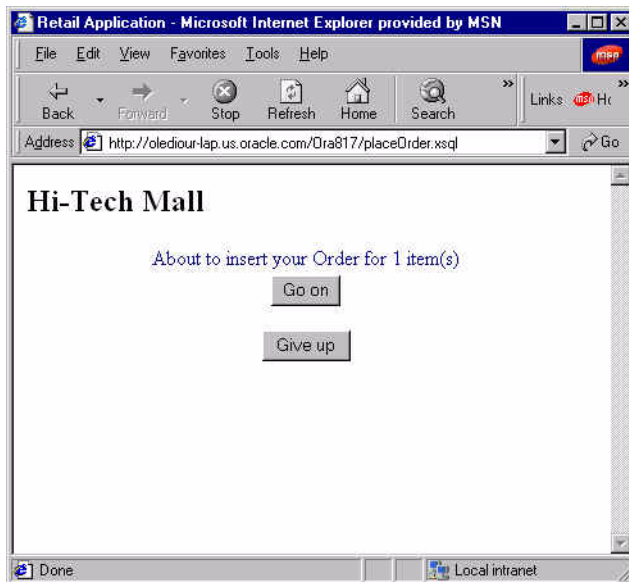
<order xmlns:xsql="urn:oracle-xsql"
        connection="retail"
        custId="000"
        prodId="000">
  <pageTitle>Hi-Tech Mall</pageTitle>
  <xsql:query tag-case      = "upper"
              rowset-element= ""
              row-element   = "cust">
    <![CDATA[
      select C.ID,
             C.NAME
      from CUSTOMER C
      where C.ID = '{@custId}'
    ]]>
    <xsql:no-rows-query>
      Select '{@custId}' as "unknown" from dual
    </xsql:no-rows-query>
  </xsql:query>

  <xsql:query tag-case="upper"
              rowset-element=""
              row-element="prod">
    <![CDATA[
      select I.ID,
             I.DESCRPTION,
             I.PRICE,
             S.NAME
      from INVENTIORY_ITEM I,
           SUPPLIER S
      where I.SUPPLIER_ID = S.ID and
            I.ID = '{@prodId}'
    ]]>
    <xsql:no-rows-query>
      Select '{@prodId}' as "unknown" from dual
    </xsql:no-rows-query>

```

```
</xsql:query>  
  
<returnHome>index.xsql</returnHome>  
</order>
```

図 12-14 [小売業者]: 「Go On」をクリックする



2 小売業者が発注する

小売業者は、「Go On」 ボタンをクリックしてから、「Validate」 を選択して注文を検証します。図 12-15 および図 12-16 を参照してください。

図 12-15 [小売業者]: 「Validate」 をクリックする

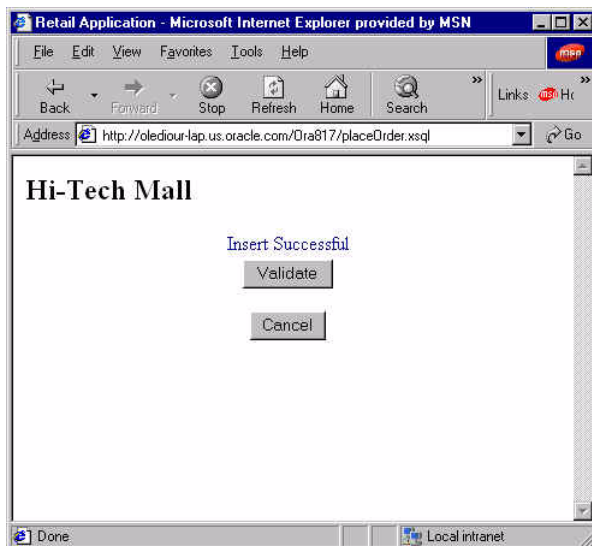
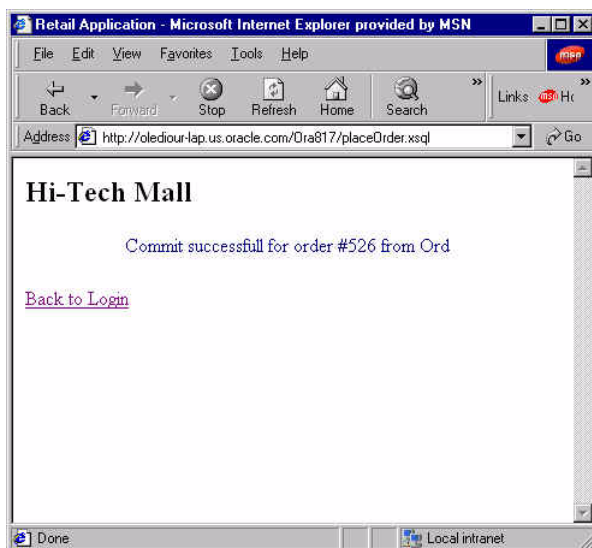


図 12-16 [小売業者]: 正常にコミットされ、Ord 表が更新されている



3 「Validate」をクリックしてトランザクションをコミットし、小売業者アプリケーションがXML注文を作成する

1. 「Validate」をクリックすると、XSQL Servlet アクション・ハンドラによって主な B2B プロセスがトリガーされます。これでクライアントの相互作用が終了します。

次のスクリプトは、B2B アプリケーション（デモ）によって実行されます。

- [XSQL スクリプトの例 5: B2B プロセスの開始 - placeorder.xsql](#)
- [Java の例 1: placeOrder.xsql がコールしたアクション・ハンドラ - RetailActionHandler.java](#)
- [Java の例 2: RetailActionHandler.java のセッション・コンテキストの保持 - SessionHolder.java](#)

XSQL スクリプトの例 5: B2B プロセスの開始 - placeorder.xsql

```
<?xml version="1.0"?>
<!--
|   これは、最後にコールされる 5 つ目のスクリプトですが、重要でないわけではありません。
|   このスクリプトは、B2B プロセス全体を実際に起動します。
|   このスクリプトは、XSQL Servlet のアクション・ハンドラ機能を使用します。
|
|   $Author: olediou@us $
|   $Revision: 1.1 $
+-->
<?xml-stylesheet type="text/xsl" media="HandHTTP"      href="PP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla"      href="HTML.xsl"?>

<placeOrder xmlns:xsql="urn:oracle-xsql"
  connection="retail"
  dbUrl      ="jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL"
  username   ="retailer"
  password   ="retailer"
  entity     ="Ord"
  operation  ="insert"
  custId     =" "
  ordId      =" "
  prodId     =" "
  qty        =" ">
  <xsql:include-request-params/>
  <pageTitle>Hi-Tech Mall</pageTitle>
  <pageSeparator/>
```

```
<xsql:action handler      ="B2BDemo.XSQLActionHandler.RetailActionHandler"
                dbUrl      ="{@dbUrl}"
                username    ="{@username}"
                password    ="{@password}"
                entity      ="{@entity}"
                operation    ="{@operation}"
                custId      ="{@custId}"
                ordId       ="{@ordId}"
                prodId      ="{@prodId}"
                qty         ="{@qty}"/>
<pageSeparator/>
<bottomLinks>
  <aLink href="placeOrder.xsql?operation=rollback">Rollback</aLink>
</bottomLinks>
<returnHome>index.xsql</returnHome>
</placeOrder>
```

Java の例 1: placeOrder.xsql がコールしたアクション・ハンドラ - RetailActionHandler.java

注意： この例は、約 20 ページあります。

```
package B2BDemo.XSQLActionHandler;
/**
 * placeOrder.xsql がコールしたアクション・ハンドラです。
 * B2B プロセス自体を実際に起動します。
 * SessionHolder を使用してトランザクション状態を保持します。
 *
 * @see SessionHolder
 * @see placeOrder.xsql
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Corp.
 */
import oracle.xml.xsql.*;
import oracle.xml.xsql.actions.XSQLIncludeXSQLHandler;
import javax.servlet.http.*;
import javax.servlet.*;
import org.w3c.dom.*;

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

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

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class RetailActionHandler extends XSQLActionHandlerImpl
{
    private static final boolean verbose    = false;
    private static final boolean debugFile = false;

    private Connection actionConnection = null;

    private String appUrl      = "";
    private String appUser     = "";
    private String appPassword = "";

    public static final String DBURL      = "dbUrl";
    public static final String USERNAME   = "username";
    public static final String PASSWORD   = "password";

    public static final String OPERATION  = "operation";

    public static final String ENTITY     = "entity";

    public static final String ORCID      = "ordId";
    public static final String ORDERDATE  = "orderDate";
    public static final String CONTACTNAME = "contactName";
    public static final String TRACKINGNO = "trackingNo";
    public static final String STATUS     = "status";
    public static final String CUSTID     = "custId";

    public static final String QTY        = "qty";
    public static final String PRODID     = "prodId";

    public static final String SELECT     = "select";
    public static final String INSERT     = "insert";
    public static final String BEGIN      = "begin";
    public static final String COMMIT     = "commit";
    public static final String ROLLBACK   = "rollback";

    XSQLActionHandler nestedHandler = null;
    String operation = null;

    String entity      = null;
```

```
String ordId      = null;
String orderDate  = null;
String contactName = null;
String trackingNo = null;
String status     = null;
String custId     = null;
String qty        = null;
String prodId     = null;

HttpServletRequest request = null;
HttpServletResponse response = null;
HttpSession session = null;

public void init(XSQLPageRequest xspRequest, Element action)
{
    super.init(xspRequest, action);
    // パラメータを取り出します。

    if (verbose)
        System.out.println("init Action Handler.....");

    appUrl      = getAttributeAllowingParam(DBURL,      action);
    appUser     = getAttributeAllowingParam(USERNAME,   action);
    appPassword = getAttributeAllowingParam(PASSWORD,   action);

    operation = getAttributeAllowingParam(OPERATION, action);
    entity    = getAttributeAllowingParam(ENTITY,      action);

    ordId      = getAttributeAllowingParam(ORDID,      action);
    orderDate  = getAttributeAllowingParam(ORDERDATE,  action);
    contactName = getAttributeAllowingParam(CONTACTNAME, action);
    trackingNo  = getAttributeAllowingParam(TRACKINGNO, action);
    status     = getAttributeAllowingParam(STATUS,     action);
    custId     = getAttributeAllowingParam(CUSTID,     action);
    prodId     = getAttributeAllowingParam(PRODID,     action);
    qty        = getAttributeAllowingParam(QTY,        action);
    //
    if (verbose)
    {
        System.out.println("OrdID > " + ordId);
        System.out.println("CustID > " + custId);
        System.out.println("ProdID > " + prodId);
    }

    final String HOLDER_NAME = "XSQLActionHandler.connection";
```



```

try
{
    if (xspRequest.getRequestType().equals("Servlet"))
    {
        XSQLServletPageRequest xspr = (XSQLServletPageRequest)xspRequest;
        HttpServletRequest req      = xspr.getHttpServletRequest();
        session = req.getSession(true); // true : 見つからない場合は作成する !!!
        if (verbose)
            System.out.println("Session Id = " + session.getId() + " - new : " +
                               session.isNew());
        SessionHolder sh = (SessionHolder) session.getValue(HOLDER_NAME);
        if (sh == null)
        {
            if (verbose)
                System.out.println("New SessionHandler > Getting connected at " +
                                   (new java.util.Date()));
            actionConnection = getConnected(appUrl, appUser, appPassword);
            sh = new SessionHolder(actionConnection);
            session.putValue(HOLDER_NAME, sh);
        }
        actionConnection = sh.getConnection();
        if (verbose)
        {
            System.out.println("Reusing Connection at " + (new java.util.Date()) +
                               " - Opened at " + sh.getOpenDate().toString());
            System.out.println("Driver      : " +
                               actionConnection.getMetaData().getDriverName());
            System.out.println("SessionId  : " + session.getId());
            System.out.println("AutoCommit : " +
                               actionConnection.getAutoCommit());
        }
    }
}
catch (Exception e)
{
    System.err.println("Error in retrieving session context \n" + e);
    e.printStackTrace();
}
}

// 結果は out パラメータです。
public void handleAction(Node result) throws SQLException
{
    XSQLPageRequest xpr = getPageRequest();
    if (xpr.getRequestType().equals("Servlet"))

```

```
{
    // サブレットのコンテキストおよびコンポーネントを取得します。
    XSQLErrorPageRequest xspr = (XSQLErrorPageRequest)xpr;
    request = xspr.getHttpServletRequest();
    response = xspr.getHttpServletResponse();

    Document doc = null;

    // CLASSPATH を表示します。
    XMLDocument myDoc = new XMLDocument();
    try
    {
        Element root = myDoc.createElement("root");
        myDoc.appendChild(root);

        Element cp = myDoc.createElement("ClassPath");
        root.appendChild(cp);

        // テキストは、そのノードの子です。
        Node cpTxt = myDoc.createTextNode("text#");
        cpTxt.setNodeValue(System.getProperty("java.class.path"));
        cp.appendChild(cpTxt);

        Element e = myDoc.getDocumentElement();
        e.getParentNode().removeChild(e);
        result.appendChild(e); // 結果を戻す前に、その結果に子を追加します。
    }
    catch (Exception e)
    {
        System.err.println("Building XMLDoc");
        e.printStackTrace();
    }
    try
    {
        // 操作値を保持するノードを追加します。
        XMLDocument xmlDoc = new XMLDocument();
        Element elmt = xmlDoc.createElement("requiredOperation");
        xmlDoc.appendChild(elmt);
        Node theText = xmlDoc.createTextNode("text#");
        theText.setNodeValue(operation);
        elmt.appendChild(theText);
        // 結果に追加します。
        Element e = xmlDoc.getDocumentElement();
        e.getParentNode().removeChild(e);
        result.appendChild(e); // 結果を戻す前に、その結果に子を追加します。
    }
```

```
}
catch (Exception e)
{
    System.err.println("Building XMLDoc (2)");
    e.printStackTrace();
}

try
{
    // ディスパッチします。
    if (operation.equals(SELECT))
    /* doc = manageSelect() */;
    else if (operation.equals(INSERT))
        doc = manageInsert();
    else if (operation.equals(BEGIN))
        doc = doBegin();
    else if (operation.equals(COMMIT))
        doc = doCommit();
    else if (operation.equals(ROLLBACK))
        doc = doRollback();
    else // 不正な操作です。
    {
        XMLDocument xmlDoc = new XMLDocument();
        Element elmt = xmlDoc.createElement("unknownOperation");
        xmlDoc.appendChild(elmt);
        Node theText = xmlDoc.createTextNode("text#");
        theText.setNodeValue(operation);
        elmt.appendChild(theText);
        // 結果に追加します。
        Element e = xmlDoc.getDocumentElement();
        e.getParentNode().removeChild(e);
        result.appendChild(e); // 結果を戻す前に、その結果に子を追加します。
    }
}
catch (Exception ex)
{
    // file://this.reportError(e);
    XMLDocument xmlDoc = new XMLDocument();
    Element elmt = xmlDoc.createElement("operationProblem");
    xmlDoc.appendChild(elmt);
    Node theText = xmlDoc.createTextNode("text#");
    theText.setNodeValue(ex.toString());
    elmt.appendChild(theText);
    // 結果に追加します。
    Element e = xmlDoc.getDocumentElement();
}
```

```
        e.getParentNode().removeChild(e);
        result.appendChild(e); // 結果を戻す前に、その結果に子を追加します。
    }

    try
    {
        if (doc != null)
        {
            Element e = doc.getDocumentElement();
            e.getParentNode().removeChild(e);
            result.appendChild(e); // 結果を戻す前に、その結果に子を追加します。
        }
    }
    catch (Exception e)
    {
        try
        {
            ServletOutputStream out = response.getOutputStream();
            out.println(e.toString());
        }
        catch (Exception ex) {}
    }
}

else // コマンドライン?
{
    System.out.println("Request type is [" + xpr.getRequestType() + "]);
}
}

/**
 * このデモでは不要なため削除しました。
 *
 * private Document manageSelect() throws Exception
 * {
 *     Document doc = null;
 *     String cStmt = "";
 *
 *     if (custId != null && custId.length() > 0)
 *         vo.setWhereClause("Customer_Id = '" + custId + "'");
 *     else
 *         vo.setWhereClause(null);
 *     vo.executeQuery();
 *     doc = data.getXMLDocument(); // 問合せが暗黙的に実行されました！
 *     return doc;
 * }
```

```

*/
private Document manageInsert() throws Exception
{
    Document doc = null;

    if (entity.equals("Ord"))
        doc = insertInOrd();
    else if (entity.equals("LineItem"))
        doc = insertInLine();
    else
    {
        doc = new XMLDocument();
        Element elmt = doc.createElement("operationQuestion");
        Attr attr = doc.createAttribute("opType");
        attr.setValue("insert");
        elmt.setAttributeNode(attr);
        doc.appendChild(elmt);
        Node txt = doc.createTextNode("text#");
        elmt.appendChild(txt);
        txt.setNodeValue("Don't know what to do with " + entity);
    }
    return doc;
}

private Document insertInOrd()
{
    Document doc = null;
    if (custId == null || custId.length() == 0)
    {
        doc = new XMLDocument();
        Element elmt = doc.createElement("operationProblem");
        Attr attr = doc.createAttribute("opType");
        attr.setValue("OrdInsert");
        elmt.setAttributeNode(attr);
        doc.appendChild(elmt);
        Node txt = doc.createTextNode("text#");
        elmt.appendChild(txt);
        txt.setNodeValue("Some element(s) missing for ord insert (custId)");
    }
    else
    {
        String seqStmt = "select Ord_Seq.nextVal from dual";
        String seqVal = "";
        try
        {

```

```
Statement stmt = actionConnection.createStatement();
ResultSet rSet = stmt.executeQuery(seqStmt);
while (rSet.next())
    seqVal = rSet.getString(1);
rSet.close();
stmt.close();
}
catch (SQLException e)
{
    System.err.println("Error reading ORD_SEQ Sequence : " + e.toString());
}
//
//              1              2              3              4
String cStmt = "insert into ORD values (?, sysdate, ?, 'AX' || ?,
                                           'Pending', ?)";

try
{
    if (verbose)
        System.out.println("Inserting Order # " + seqVal);
    PreparedStatement pStmt = actionConnection.prepareStatement(cStmt);
    pStmt.setString(1, seqVal);
    pStmt.setString(2, "Ora817"); // デフォルト値です。
    pStmt.setString(3, seqVal);
    pStmt.setString(4, custId);
    pStmt.execute();
    pStmt.close();
}
/**
try
{
    Statement stmt = actionConnection.createStatement();
    ResultSet rSet = stmt.executeQuery("SELECT * FROM ORD WHERE ID = " +
                                        seqVal);

    int i = 0;
    while (rSet.next())
        i++;
    if (verbose)
        System.out.println(i + " record found for " + seqVal);
    rSet.close();
    stmt.close();
}
catch (SQLException e)
{
    System.err.println("Error : " + e.toString());
}
*/
doc = new XMLDocument();
```

```
Element elmt = doc.createElement("operationResult");
Attr attr = doc.createAttribute("opType");
attr.setValue("insert");
elmt.setAttributeNode(attr);

attr = doc.createAttribute("Step");
attr.setValue(entity);
elmt.setAttributeNode(attr);

doc.appendChild(elmt);
Node txt = doc.createTextNode("text#");
elmt.appendChild(txt);
txt.setNodeValue("About to insert your Order for " + qty + " item(s)");

Element nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Go on");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
Element pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// 参照オブジェクト
Element prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("entity");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("LineItem");
prm.setAttributeNode(attr);
// 顧客 ID
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("custId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(custId);
prm.setAttributeNode(attr);
// 商品 ID
prm = doc.createElement("prm");
```

```
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("prodId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(prodId);
prm.setAttributeNode(attr);
// 数量
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("qty");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(qty);
prm.setAttributeNode(attr);
// ordID
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("ordId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(seqVal);
prm.setAttributeNode(attr);

nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Give up");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// 参照オブジェクト
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("operation");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("rollback");
```



```

        prm.setAttributeNode(attr);
    }
    catch (Exception e)
    {
        doc = new XMLDocument();
        Element elmt = doc.createElement("operationProblem");
        Attr attr = doc.createAttribute("opType");
        attr.setValue("insert");
        elmt.setAttributeNode(attr);

        attr = doc.createAttribute("Step");
        attr.setValue(entity);
        elmt.setAttributeNode(attr);

        doc.appendChild(elmt);
        Node txt = doc.createTextNode("text#");
        elmt.appendChild(txt);
        txt.setNodeValue(e.toString());
        if (verbose)
            System.out.println("Error : " + e.toString());
        Element prm = doc.createElement("parameters");
        elmt.appendChild(prm);
        // ID
        Element prmVal = doc.createElement("ID");
        prm.appendChild(prmVal);
        txt = doc.createTextNode("text#");
        prmVal.appendChild(txt);
        txt.setNodeValue(ordId);
        // CUSTOMER_ID
        prmVal = doc.createElement("CUSTOMER_ID");
        prm.appendChild(prmVal);
        txt = doc.createTextNode("text#");
        prmVal.appendChild(txt);
        txt.setNodeValue(custId);
    }
}
return doc;
}

private Document insertInLine()
{
    Document doc = null;
    if (custId == null || custId.length() == 0 ||
        qty == null || qty.length() == 0 ||
        prodId == null || prodId.length() == 0 ||

```

```
        ordId == null || ordId.length() == 0)
    {
        doc = new XMLDocument();
        Element elmt = doc.createElement("operationProblem");
        Attr attr = doc.createAttribute("opType");
        attr.setValue("lineInsert");
        elmt.setAttributeNode(attr);
        doc.appendChild(elmt);
        Node txt = doc.createTextNode("text#");
        elmt.appendChild(txt);
        txt.setNodeValue("Some element(s) missing for line insert (" +
            ((custId == null || custId.length() == 0)? "custId ":"") +
            ((qty == null || qty.length() == 0)? "qty ":"") +
            ((prodId == null || prodId.length() == 0)? "prodId ":"") +
            ((ordId == null || ordId.length() == 0)? "ordId ":"") + ")");

        Element subElmt = doc.createElement("custId");
        elmt.appendChild(subElmt);
        txt = doc.createTextNode("text#");
        subElmt.appendChild(txt);
        txt.setNodeValue(custId);

        subElmt = doc.createElement("qty");
        elmt.appendChild(subElmt);
        txt = doc.createTextNode("text#");
        subElmt.appendChild(txt);
        txt.setNodeValue(qty);

        subElmt = doc.createElement("prodId");
        elmt.appendChild(subElmt);
        txt = doc.createTextNode("text#");
        subElmt.appendChild(txt);
        txt.setNodeValue(prodId);

        subElmt = doc.createElement("ordId");
        elmt.appendChild(subElmt);
        txt = doc.createTextNode("text#");
        subElmt.appendChild(txt);
        txt.setNodeValue(ordId);
    }
    else
    {
        if (verbose)
            System.out.println("Inserting line : Ord>" + ordId + ", Prod>" + prodId
```

```

                                                                    + ", Qty>" + qty);
/**
try
{
    Statement stmt = actionConnection.createStatement();
    ResultSet rSet = stmt.executeQuery("SELECT * FROM ORD WHERE ID = " +
                                        ordId);

    int i = 0;
    while (rSet.next())
        i++;
    System.out.println(i + " record found for " + ordId);
    rSet.close();
    stmt.close();
}
catch (SQLException e)
{
    System.err.println("Error : " + e.toString());
}
*/
String cStmt = "insert into line_item values (Line_item_seq.nextVal, ?, ?,
                                                                    ?, 0)";

try
{
    PreparedStatement pStmt = actionConnection.prepareStatement(cStmt);
    pStmt.setString(1, qty);
    pStmt.setString(2, prodId);
    pStmt.setString(3, ordId);
    pStmt.execute();
    pStmt.close();

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("insert");
    elmt.setAttributeNode(attr);

    attr = doc.createAttribute("Step");
    attr.setValue(entity);
    elmt.setAttributeNode(attr);

    doc.appendChild(elmt);
    Node txt = doc.createTextNode("text#");
    elmt.appendChild(txt);
    txt.setNodeValue("Insert Successful");
}
```

```
Element nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Validate");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
Element pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// 操作
Element prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("operation");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("commit");
prm.setAttributeNode(attr);
// ordID
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
attr.setValue("ordId");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue(ordId);
prm.setAttributeNode(attr);

nextElmt = doc.createElement("nextStep");
elmt.appendChild(nextElmt);
attr = doc.createAttribute("Label");
attr.setValue("Cancel");
nextElmt.setAttributeNode(attr);

attr = doc.createAttribute("Action");
nextElmt.setAttributeNode(attr);
attr.setValue("placeOrder.xsql");
pList = doc.createElement("prmList");
nextElmt.appendChild(pList);
// 操作
prm = doc.createElement("prm");
pList.appendChild(prm);
attr = doc.createAttribute("name");
```

```
attr.setValue("operation");
prm.setAttributeNode(attr);
attr = doc.createAttribute("value");
attr.setValue("rollback");
prm.setAttributeNode(attr);
}
catch (Exception e)
{
    if (verbose)
        System.out.println("Error when inserting " + e.toString());

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationProblem");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("insert");
    elmt.setAttributeNode(attr);

    attr = doc.createAttribute("Step");
    attr.setValue(entity);
    elmt.setAttributeNode(attr);
    doc.appendChild(elmt);

    Node txt = doc.createTextNode("text#");
    elmt.appendChild(txt);
    txt.setNodeValue(e.toString());

    Element prm = doc.createElement("parameters");
    elmt.appendChild(prm);
    // ID
    Element prmVal = doc.createElement("ORD_ID");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
    txt.setNodeValue(ordId);
    // 数量
    prmVal = doc.createElement("QTY");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
    txt.setNodeValue(qty);
    // ITEM_ID
    prmVal = doc.createElement("ITEM_ID");
    prm.appendChild(prmVal);
    txt = doc.createTextNode("text#");
    prmVal.appendChild(txt);
```

```
        txt.setNodeValue(prodId);
    }
}
return doc;
}

private Document doCommit() throws Exception
{
    Document doc = null;
    actionConnection.commit();

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("commit");
    elmt.setAttributeNode(attr);
    doc.appendChild(elmt);
    Node txt = doc.createTextNode("dummy");
    elmt.appendChild(txt);
    txt.setNodeValue("Commit successfull for order #" + ordId + " from " + entity);

    if (ordId != null && ordId.length() > 0)
    {
        // AQ に送信する XML ドキュメントを生成します。
        // OrdID 値が - である Ord から開始します。

        AQWriter aqw = null;

        aqw = new AQWriter(AppCste.AQuser,
                           AppCste.AQpswd,
                           AppCste.AQDBUrl,
                           "AppOne_QTab",
                           "AppOneMsgQueue");

        String doc2send = XMLGen.returnDocument(actionConnection, ordId);
        // XMLDoc をキューに送信します。
        try
        {
            if (verbose)
                System.out.println("Doc : " + doc2send);
            if (debugFile)
            {
                BufferedWriter bw = new BufferedWriter(new FileWriter("debug.txt"));
                bw.write("Rows in " + entity);
            }
        }
    }
}
```

```
        bw.write(doc2send);
        bw.flush();
        bw.close();
    }
}
catch (Exception ex) {}

aqw.writeQ(new B2BMessage(MessageHeaders.APP_A,
                           MessageHeaders.APP_B,
                           MessageHeaders.NEW_ORDER,
                           doc2send));
aqw.flushQ(); // コミットします。
}

return doc;
}

private Document doRollback() throws Exception
{
    Document doc = null;
    actionConnection.rollback();

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("rollback");
    elmt.setAttributeNode(attr);
    doc.appendChild(elmt);
    Node txt = doc.createTextNode("dummy");
    elmt.appendChild(txt);
    txt.setNodeValue("Rollback successfull");

    return doc;
}

private Document doBegin() throws Exception
{
    Document doc = null;
    actionConnection.setAutoCommit(false);

    doc = new XMLDocument();
    Element elmt = doc.createElement("operationResult");
    Attr attr = doc.createAttribute("opType");
    attr.setValue("begin");
    elmt.setAttributeNode(attr);
```

```
        doc.appendChild(elmt);
        Node txt = doc.createTextNode("dummy");
        elmt.appendChild(txt);
        txt.setNodeValue("Begin successfull");

        return doc;
    }

    private static Connection getConnected(String connURL,
                                           String userName,
                                           String password)
    {
        Connection conn = null;
        try
        {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            conn = DriverManager.getConnection(connURL, userName, password);
            conn.setAutoCommit(false);
        }
        catch (Exception e)
        {
            System.err.println(e);
            System.exit(1);
        }
        return conn;
    }
}
```

Java の例 2: RetailActionHandler.java のセッション・コンテキストの保持 - SessionHolder.java

```
// Copyright (c) 2000 Oracle Corporation
package B2BDemo.XSQLActionHandler;
/**
 * XSQL アクション・ハンドラからの接続コンテキストを保持するために使用されます。
 * また、サープレットの期限が切れたときに接続をクローズします。
 *
 * @see RetailActionHandler
 */
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class SessionHolder implements HttpSessionBindingListener
{
    private Connection c;
    private java.util.Date d = null;
```



```
public SessionHolder(Connection conn)
{
    System.out.println("New SessionHandler");
    this.c = conn;
    this.d = new java.util.Date();
}

public Connection getConnection()
{
    return this.c;
}

public java.util.Date getOpenDate()
{
    return this.d;
}

public void valueBound(HttpSessionBindingEvent event)
{
    System.out.println("\nvalueBound ! " + event.getName() + "\nat " + (new
        java.util.Date()) + "\nfor " + event.getSession().getId());
}

public void valueUnbound(HttpSessionBindingEvent event)
{
    System.out.println("\nvalueUnbound ! " + event.getName() + "\nat " + (new
        java.util.Date()) + "\nfor " + event.getSession().getId());
    event.getSession().removeValue("XSQLActionHandler.connection");
    if (this.c != null)
    {
        try { this.c.close(); }
        catch (Exception e)
        {
            System.out.println("Problem when closing the connection from " +
                event.getName() +
                " for " +
                event.getSession().getId() +
                " :\n" +
                e);
        }
    }
}
```

4 AQ Broker Transformer がサプライヤの形式に従って XML ドキュメントを変換する

1. AQ Broker Transformer アプリケーションに、XML 注文が保留されていることが通知されます。
2. 注文の詳細情報を含む XML ドキュメントが、XML SQL Utility によって作成されます。このドキュメントは、アドバンスト・キューイングを使用して、伝播のために AQ Broker Transformer へ送信されています。

AQ Broker アプリケーションは、Stylesheet 表から次のことを認識しています。

- 送信元 : 小売業者
- 送信先 : サプライヤ
- 目的 : 新しい注文

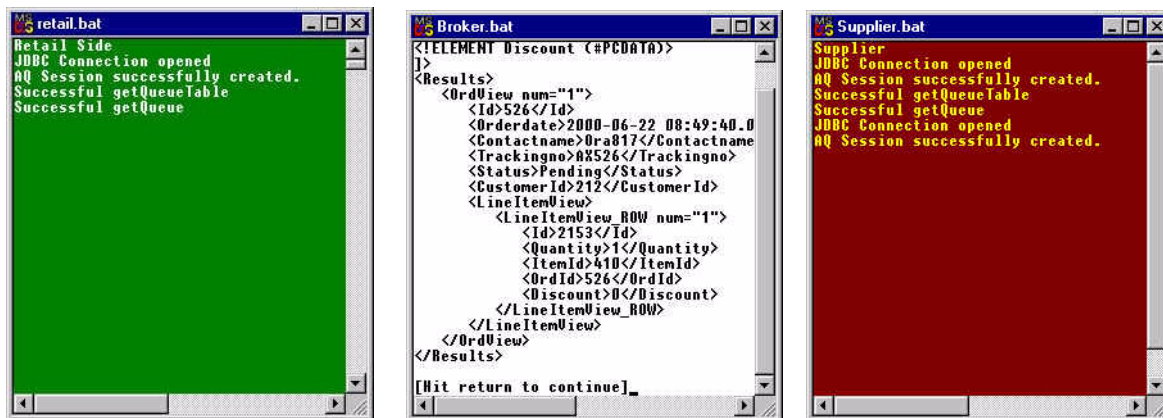
これらの要素は、Stylesheet 表で正しいスタイルシートを選択するために使用されます。XSLT Processor は変換処理を行います。図 12-17 を参照してください。

スクリプト

- MessageBroker.java は BrokerThread.java をコールします。
- BrokerThread.java は、AQReader.java および AQWriter.java をコールします。

AQReader.java および AQWriter.java は、メッセージ構造に B2BMessages.java を使用します。

図 12-17 [AQ Broker]: retailer.bat、broker.bat および supplier.bat のコンソール (1/3) の表示



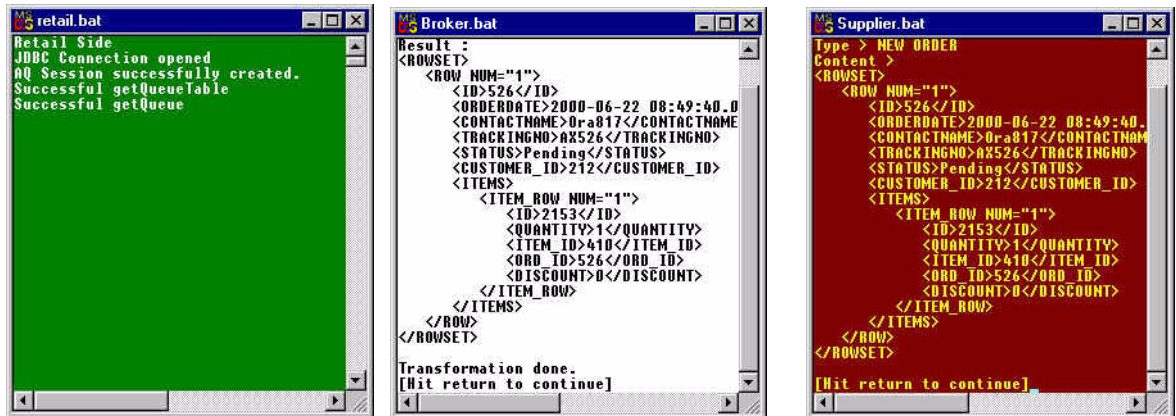
3. AQ Broker のコンソールで [Enter] を押します。
4. AppFrom 列、AppTo 列および Op 列の内容に従って、Stylesheets 表内で正しいスタイルシートが検出されます。XSL Transformation は、選択されたスタイルシートを使用して処理を続行します。これで、サプライヤ用に再フォーマットされた XML ドキュメントが準備できました。

注意： ここでは、XML + XSL = XML となります。

5. AQ Broker のコンソールで [Enter] を再度押します。図 12-18 を参照してください。broker.bat 画面は、変換処理に応じて変化します。結果は、XSLT 変換後に取得されます。

AQ Broker Transformer およびアドバンスト・キューイング・プロセスを実行するコードのリストは、12-155 ページの「AQ Broker Transformer およびアドバンスト・キューイングのスクリプト」を参照してください。

図 12-18 [AQ Broker]: retailer.bat、broker.bat および supplier.bat のコンソール (2/3) の表示



新しく再フォーマットされた XML ドキュメントは、アドバンスト・キューイング [WRITE] を介してサプライヤに送信されます。

注意： AQ Broker およびサプライヤのバッチ画面は、両方のアプリケーションが現時点で同じ XML ドキュメントを処理しているため、同じように表示されている必要があります。

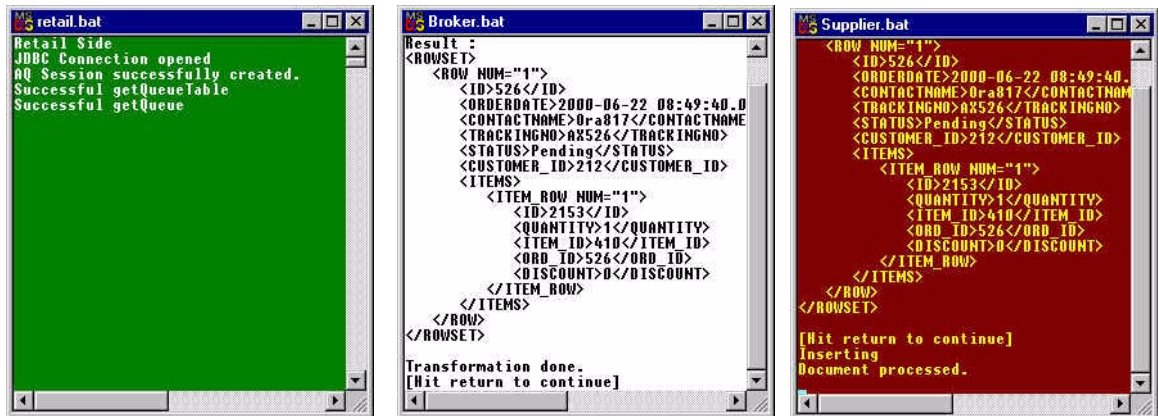
図 12-19 AQ Broker Transformer の XML ドキュメント出力例



5 サプライヤ・アプリケーションが XML ドキュメントを解析し、サプライヤのデータベースに注文を挿入する

1. サプライヤ・アプリケーションが XML ドキュメントを受信します。この XML ドキュメントに含まれるデータを解析する必要があります。このデータはその後データベースへ挿入されます。
2. サプライヤのコンソールで [Enter] を押します。図 12-20 を参照してください。

図 12-20 [AQ Broker]: retailer.bat、broker.bat および supplier.bat のコンソール (3/3) の表示



6a サプライヤ・アプリケーションが保留注文をサプライヤに通知する

1. ドキュメントが処理され、データが挿入されます。サプライヤ・アプリケーションの監視プログラムが、注文が保留されているという通知メッセージを送ります。図 12-21 を参照してください。
2. 「Supplier Watcher」ダイアログ・ボックスで「OK」をクリックします。図 12-22 を参照してください。

スクリプト

- `SupplierWatcher.java` は `SupplierFramer.java` をコールします。

図 12-21 サプライヤ・アプリケーションが保留注文をサプライヤに通知する : 「Wake Up!」

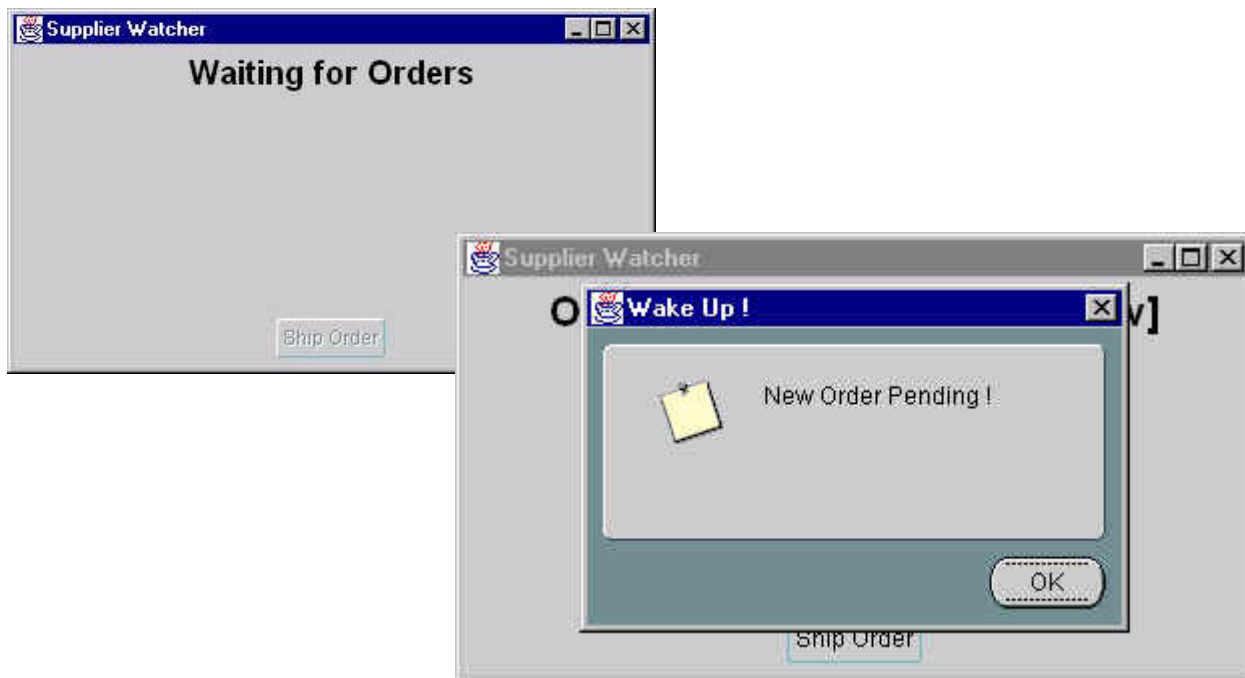
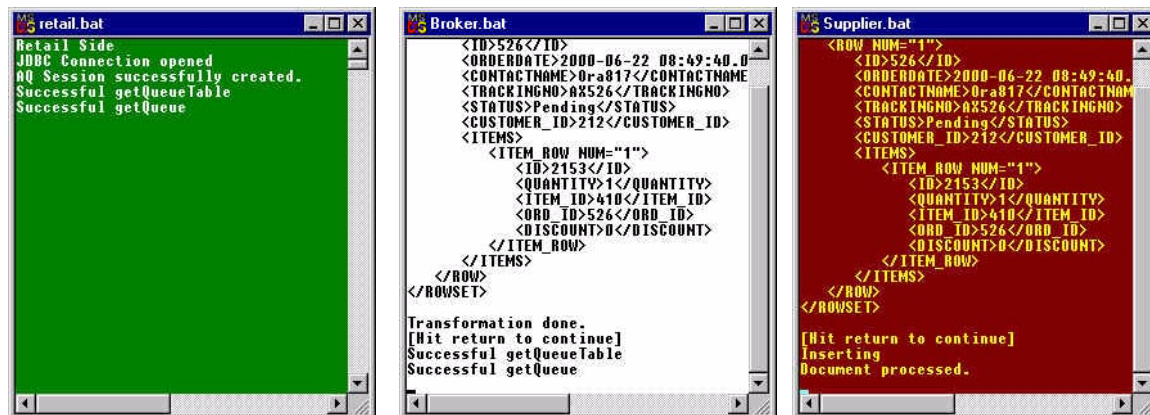


図 12-22 [サプライヤ]: retail.bat、broker.bat および supplier.bat のコンソール：「Wake Up」の「OK」をクリックした後



```
retail.bat
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue

Broker.bat
<ID>526</ID>
<ORDERDATE>2000-06-22 08:49:40.0
<CONTACTNAME>0ra817</CONTACTNAME>
<TRACKINGNO>AX526</TRACKINGNO>
<STATUS>Pending</STATUS>
<CUSTOMER_ID>212</CUSTOMER_ID>
<ITEMS>
  <ITEM_ROW NUM="1">
    <ID>2153</ID>
    <QUANTITY>1</QUANTITY>
    <ITEM_ID>410</ITEM_ID>
    <ORD_ID>526</ORD_ID>
    <DISCOUNT>0</DISCOUNT>
  </ITEM_ROW>
</ITEMS>
</ROW>
</ROWSET>
Transformation done.
[Hit return to continue]
Successful getQueueTable
Successful getQueue

Supplier.bat
<ROW NUM="1">
  <ID>526</ID>
  <ORDERDATE>2000-06-22 08:49:40.
  <CONTACTNAME>0ra817</CONTACTNAME>
  <TRACKINGNO>AX526</TRACKINGNO>
  <STATUS>Pending</STATUS>
  <CUSTOMER_ID>212</CUSTOMER_ID>
  <ITEMS>
    <ITEM_ROW NUM="1">
      <ID>2153</ID>
      <QUANTITY>1</QUANTITY>
      <ITEM_ID>410</ITEM_ID>
      <ORD_ID>526</ORD_ID>
      <DISCOUNT>0</DISCOUNT>
    </ITEM_ROW>
  </ITEMS>
</ROW>
</ROWSET>
[Hit return to continue]
Inserting
Document processed.
```

6b サプライヤが商品を小売業者に出荷する

1. サプライヤが、注文の出荷を決定します。ダイアログ・ボックスの「Ship Order」をクリックします。図 12-23 および図 12-24 を参照してください。

スクリプト : ここでも SupplierWatcher.java を使用します。

図 12-23 [サプライヤ] : 注文の出荷を決定する

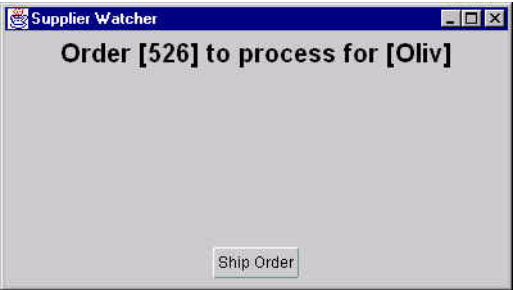
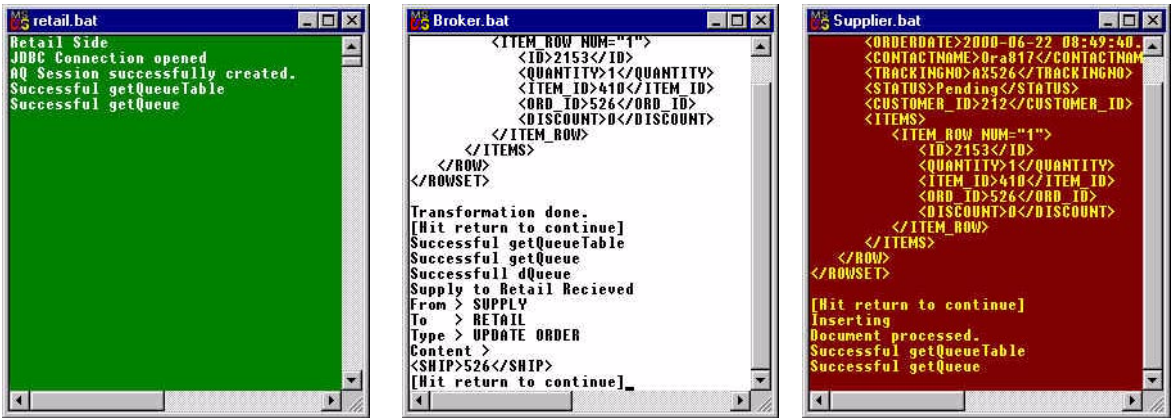


図 12-24 [サプライヤ] : 「Ship Order」クリック時の retailer.bat、broker.bat および supplier.bat のコンソールの表示



6c サプライヤ・アプリケーションが AQ Broker に送信する新しい XML メッセージを生成する

1. 注文を出荷するサプライヤ・アプリケーションが、ブローカに送信する新しいメッセージを生成します。
2. ブローカのコソールで [Enter] を押します。図 12-25 を参照してください。

図 12-25 [サプライヤ]: retailer.bat、broker.bat および supplier.bat のコソール - 新しい XML ドキュメントの構成

```

retailer.bat
Retail Side
JDBC Connection opened
AQ Session successfully created.
Successful getQueueTable
Successful getQueue

Broker.bat
<ITEM_ROW NUM="1">
  <ID>2153</ID>
  <QUANTITY>1</QUANTITY>
  <ITEM_ID>410</ITEM_ID>
  <ORD_ID>526</ORD_ID>
  <DISCOUNT>0</DISCOUNT>
</ITEM_ROW>
</ITEMS>
</ROW>
</ROWSET>

Transformation done.
[Hit return to continue]
Successful getQueueTable
Successful getQueue
Successful dQueue
Supply to Retail Recieved
From > SUPPLY
To > RETAIL
Type > UPDATE ORDER
Content >
<SHIP>526</SHIP>
[Hit return to continue]

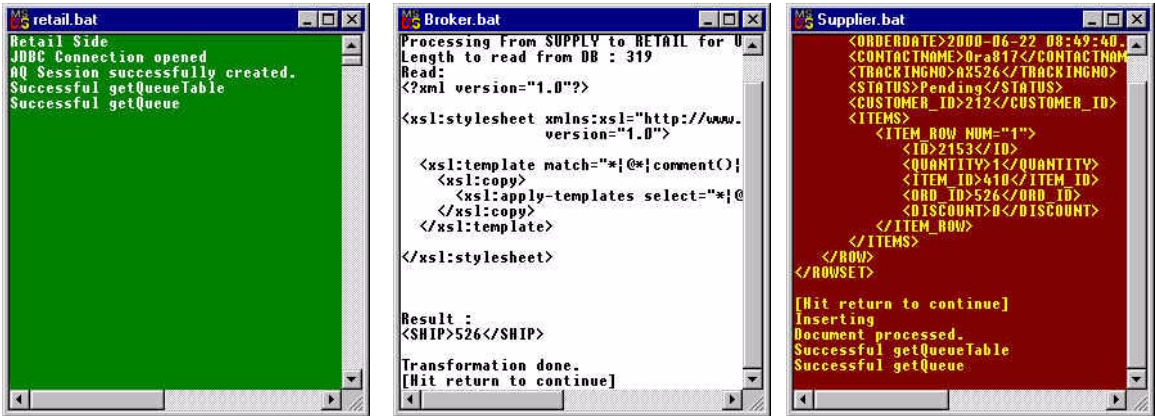
Supplier.bat
<ORDERDATE>2000-06-22 08:49:40</ORDERDATE>
<CONTACTNAME>0ra817</CONTACTNAME>
<TRACKINGNO>0X526</TRACKINGNO>
<STATUS>Pending</STATUS>
<CUSTOMER_ID>212</CUSTOMER_ID>
<ITEMS>
  <ITEM_ROW NUM="1">
    <ID>2153</ID>
    <QUANTITY>1</QUANTITY>
    <ITEM_ID>410</ITEM_ID>
    <ORD_ID>526</ORD_ID>
    <DISCOUNT>0</DISCOUNT>
  </ITEM_ROW>
</ITEMS>
</ROW>
</ROWSET>

[Hit return to continue]
Inserting
Document processed.
Successful getQueueTable
Successful getQueue
  
```

7 AQ Broker Transformer が小売業者の形式に XML 注文を変換する

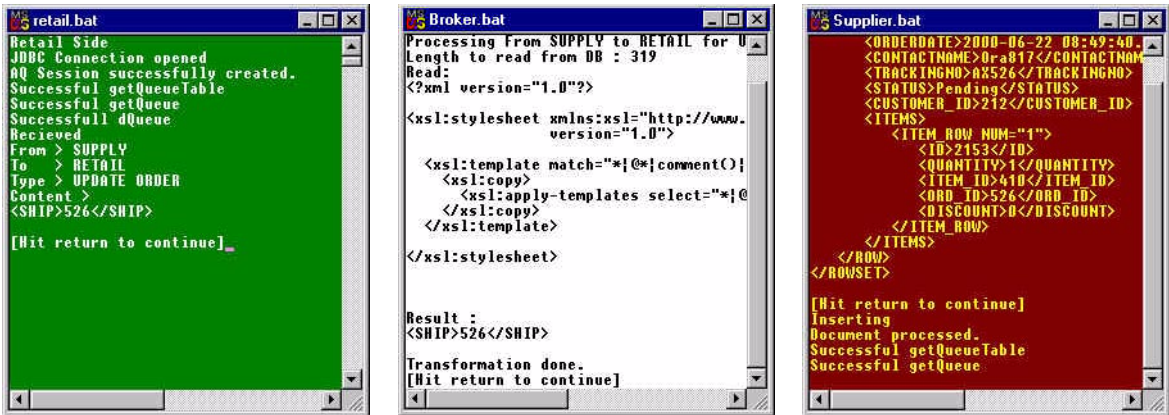
1. 手順 4 で示したように、再フォーマット済の XML ドキュメントを作成するために AQ Broker Transformer のデータベースからスタイルシートが選択され、XML ドキュメントに適用されます。
2. プローカのコソールで [Enter] を押します。図 12-26 を参照してください。

図 12-26 [AQ Broker]: retailer.bat、broker.bat および supplier.bat のコソール - XML ドキュメントの再フォーマット



3. ドキュメントが、小売業者アプリケーションに送信されます。
4. 小売業者のコソールで [Enter] を押します。図 12-27 を参照してください。これによって、XML 注文が解析されます。

図 12-27 [AQ Broker]: retailer.bat、broker.bat および supplier.bat のコソール - XML メッセージの送信



B2B XML アプリケーションの停止

B2B XML アプリケーション（デモ）を停止するには、[Java の例 3: stopQ.bat](#) を実行します。

Java の例 3: stopQ.bat

```
@echo off
@echo stopping all Qs
D:\jdev31\java\bin\java -mx50m -classpath
"D:\xml817\references\Ora817DevGuide;
D:\jdev31\lib\jdev-rt.zip;
D:\jdev31\jdbc\lib\oracle8.1.6\classes111.zip;
D:\jdev31\lib\connectionmanager.zip;
D:\jdev31\lib;D:\jdev31\lib\oraclexsql.jar;
D:\jdev31\lib\oraclexmlsql.jar;
D:\jdev31\lib\xmlparserv2_2027.jar;
D:\jdev31\jfc\lib\swingall.jar;
D:\jdev31\jswdk-1.0.1\lib\servlet.jar;
D:\Ora8i\rdbs\jlib\aqapi11.jar;
D:\Ora8i\rdbs\jlib\aqapi.jar;
D:\XMLWorkshop\xmlcomp.jar;
D:\jdev31\java\lib\classes.zip" B2BDemo.AQUtil.StopAllQueues
```

vieworder.sql を使用した注文状態の確認

注文状態をデータベースで直接参照するには、次の SQL スクリプトを実行します。

```
set ver off
select O.ID as "Order#",
       O.OrderDate as "Order Date",
       O.Status as "Status"
From ORD O,
       CUSTOMER C
Where O.CUSTOMER_ID = C.ID and
       Upper(C.NAME) = Upper('&CustName');
```

Java の例 - コール順序

次に、Java のコール順序の例を示します。各ファイルの拡張子 .java は省略しています。「<----」はコールを意味します。たとえば「AQReader <----- B2Bmessage」は、AQReader が B2BMessage をコールするという意味です。

- AQReader <---- B2BMessage
- AQWriter <---- B2BMessage
- UpdateMaster
 - <---- AQReader <----B2BMessage
 - <---- B2BMessage
 - <----- MessageHeaders
 - XMLFrame
- SupplierWatcher
 - <---- SupplierFrame
 - * <---- AQReader <----B2BMessage
 - * <----XML2DMLv2 <---- TableInDocument
 - * <---- TableInDocument
 - * <---- AQWriter <---- B2BMessage
 - * <---- B2BMessage
 - * <---- MessageHeaders
 - <---- XMLFrame
- MessageBroker
 - <---- AppCste
 - <---- BrokerThread
 - * <---- XSLTWrapper
 - * <---- AQWriter <---- B2BMessage
 - * <---- AQReader <----B2BMessage
 - <---- AQReader <----B2BMessage
 - <---- AQWriter <---- B2BMessage
 - <---- XMLFrame (MessageBroker によるコール)
- RetailActionHandler <---- SessionHolder

XSL および XSL 管理スクリプト

「B2B XML アプリケーションの実行 : 詳細手順」の例がわかりやすいように、XSL の例を個別に示します。

- [XSL スタイルシートの例 1: HTML への結果の変換 - html.xsl](#)
- [XSL スタイルシートの例 2: Palm Pilot ブラウザ用の結果変換 - pp.xsl](#)
- [Java の例 3: スタイルシートの管理 - GUIInterface.java](#)
- [Java の例 4: GUIInterface_AboutBoxPanel.java](#)
- [Java の例 5: GUIStylesheet.java](#)

XSL スタイルシートの例 1: HTML への結果の変換 - html.xsl

```
<?xml version="1.0"?>
<!--
| $Author: olediou@us $
| $Date: 04-May-2000
| xsl for html
| $Revision: 1.1 $
+-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 version="1.0">
  <xsl:output media-type="text/html" method="html" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Retail Application</title>
      </head>
      <body>
        <xsl:if test="//pageTitle">
          <h2><xsl:value-of select="//pageTitle"/></h2>
        </xsl:if>
        <xsl:choose>
          <xsl:when test="loginResult">
            <xsl:apply-templates select="loginResult"/>
          </xsl:when>
          <xsl:when test="index">
            <xsl:apply-templates select="index"/>
          </xsl:when>
          <xsl:when test="inventory">
```

```

        <xsl:apply-templates select="inventory"/>
      </xsl:when>
      <xsl:when test="order">
        <xsl:apply-templates select="order"/>
      </xsl:when>
      <xsl:when test="placeOrder">
        <xsl:apply-templates select="placeOrder"/>
      </xsl:when>
      <xsl:otherwise>
        <p align="center">
          <h3>This kind of XML Document cannot be processed...</h3>
        </p>
      </xsl:otherwise>
    </xsl:choose>
  </body>
</html>
</xsl:template>

<xsl:template match="loginResult">
  <xsl:if test="ROWSET/ROW/unknown">
    <table width="98%">
      <tr>
        <td bgcolor="yellow" align="center">
          <xsl:value-of select="ROWSET/ROW/unknown"/> is not allowed to log in !</td>
        </tr>
      </table>
    </xsl:if>
    <xsl:if test="ROWSET/ROW/NAME">
      <p align="center">
        <h2>Welcome <xsl:value-of select="ROWSET/ROW/NAME"/> !</h2>
      </p>
      <p align="center">
        <a>
          <xsl:attribute name="href">
            <xsl:value-of select="nextStep"/>?custId=<xsl:value-of select="ROWSET/ROW/ID"/>
          </xsl:attribute>
          Please enter the Mall !
        </a>
      </p>
    </xsl:if>
    <p>
      <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
    </p>
  </xsl:template>

```

```
<xsl:template match="index">
  <xsl:for-each select="form">
    <center>
      <form>
        <xsl:attribute name="action"><xsl:value-of select="./@action"/></xsl:attribute>
        <xsl:attribute name="method"><xsl:value-of select="./@method"/></xsl:attribute>
        <xsl:if test="./field">
          <table width="98%" border="1">
            <xsl:for-each select="./field">
              <tr>
                <td align="right"><xsl:value-of select="./@prompt"/></td>
                <td>
                  <input>
                    <xsl:choose>
                      <xsl:when test="./@type = 'text'">
                        <xsl:attribute name="type">text</xsl:attribute>
                      </xsl:when>
                    </xsl:choose>
                    <xsl:attribute name="name">
                      <xsl:value-of select="./@name"/></xsl:attribute>
                    </input>
                  </td>
                </tr>
              </xsl:for-each>
            </table>
          </xsl:if>
          <xsl:if test="./button">
            <p>
              <xsl:for-each select="./button">
                <input>
                  <xsl:choose>
                    <xsl:when test="./@type = 'submit'">
                      <xsl:attribute name="type">submit</xsl:attribute>
                    </xsl:when>
                  </xsl:choose>
                  <xsl:attribute name="value">
                    <xsl:value-of select="./@label"/>
                  </xsl:attribute>
                </input>
              </xsl:for-each>
            </p>
          </xsl:if>
        </form>
      </center>
```



```

    </xsl:for-each>
</xsl:template>

<xsl:template match="inventory">
  <h2>This is the Mart content</h2>
  <table>
    <tr>
      <th>Prod #</th>
      <th>Product</th>
      <th>Price</th>
      <th>Supplied by</th>
    </tr>
    <xsl:for-each select="form/theMart/ROWSET/ROW">
      <tr>
        <td><xsl:value-of select="ID"/></td>
        <td>
          <a>
            <xsl:attribute name="href">
              <xsl:value-of
select="../../../../../form/@action"/>?custId=<xsl:value-of
select="../../../../../form/hiddenFields/custId"/>&amp;prodId=<xsl:value-of
select="ID"/>
              </xsl:attribute>
              <xsl:value-of select="DESCRIPTION"/>
            </a>
          </td>
        <td><xsl:value-of select="PRICE"/></td>
        <td><xsl:value-of select="NAME"/></td>
      </tr>
    </xsl:for-each>
  </table>
  <p>
    <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
  </p>
</xsl:template>

<xsl:template match="order">
  <center>
    <h2>Thank you <xsl:value-of select="CUST/NAME"/> for shopping with us !</h2>
    <hr/>
    <h2>Please enter the quantity</h2>
    <form action="placeOrder.xsql" method="post">
      <input type="hidden" name="prodId">

```

```
<xsl:attribute name="value">
  <xsl:value-of select="PROD/ID"/>
</xsl:attribute>
</input>
<input type="hidden" name="custId">
  <xsl:attribute name="value">
    <xsl:value-of select="CUST/ID"/></xsl:attribute>
</input>
<table border="1">
  <tr>
    <td colspan="2"><xsl:value-of select="PROD/DESCRIPTION"/>
      at $<xsl:value-of select="PROD/PRICE"/> each
      supplied by <xsl:value-of select="PROD/NAME"/></td>
    </tr>
    <tr>
      <td align="right">Quantity</td>
      <td><input type="text" name="qty"/></td>
    </tr>
  </table>
  <p><input type="submit" value="Place Order"/></p>
</form>
</center>
<p>
  <a><xsl:attribute name="href">
    <xsl:value-of select="returnHome"/>
  </xsl:attribute>Back to Login</a>
</p>
</xsl:template>

<xsl:template match="placeOrder">
  <xsl:if test="operationResult">
    <table width="98%">
      <tr><td align="center">
        <font color="navy">
          <xsl:value-of select="operationResult/text()"/>
        </font></td></tr>
      <tr>
        <td align="center">
          <xsl:for-each select="operationResult/nextStep">
            <form method="post">
              <xsl:attribute name="action"><xsl:value-of
select="./@Action"/></xsl:attribute>
              <xsl:if test="prmList">
                <xsl:for-each select="prmList/prm">
                  <input type="hidden">
```

```

        <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
        <xsl:attribute name="value"><xsl:value-of
select="./@value"/></xsl:attribute>
        </input>
    </xsl:for-each>
</xsl:if>
    <input type="submit">
        <xsl:attribute name="value"><xsl:value-of
select="./@Label"/></xsl:attribute>
    </input>
</form>
</xsl:for-each>
</td>
</tr>
</table>
</xsl:if>
<xsl:if test="xsql-error">
    <table width="98%">
        <tr><td><xsl:value-of select="xsql-error/@action"/></td></tr>
        <tr><td><xsl:value-of select="xsql-error/statement"/></td></tr>
        <tr><td><xsl:value-of select="xsql-error/message"/></td></tr>
    </table>
</xsl:if>
<xsl:if test="operationProblem">
    <table width="98%">
        <tr>
            <td colspan="2" align="center">
                <font color="red"><b><xsl:value-of
select="operationProblem/text ()"/></b></font>
            </td>
        </tr>
        <xsl:for-each select="operationProblem/parameters/*">
            <tr>
                <td align="right"><xsl:value-of select="name()"/></td>
                <td align="left"><xsl:value-of select="."/></td>
            </tr>
        </xsl:for-each>
    </table>
</xsl:if>
<xsl:if test="bottomLinks">
    <xsl:choose>
        <xsl:when test="operationResult">
            </xsl:when>
        <xsl:otherwise>

```

```

        <p align="center">
            <xsl:for-each select="bottomLinks/aLink">
                [<a><xsl:attribute name="href"><xsl:value-of
select="."/ @href"/></xsl:attribute><xsl:value-of select="."/></a>]
            </xsl:for-each>
        </p>
    </xsl:otherwise>
</xsl:choose>
</xsl:if>
<xsl:choose>
    <xsl:when test="operationResult/nextStep">
        </xsl:when>
    <xsl:otherwise>
        <xsl:if test="returnHome">
            <p>
                <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
            </p>
        </xsl:if>
    </xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>

```

XSL スタイルシートの例 2: Palm Pilot ブラウザ用の結果変換 - pp.xsl

```

<?xml version="1.0"?>
<!--
| $Author: olediou@us $
| $Date: 04-May-2000
| xsl for html (Palm Pilot, HandWeb browser)
| $Revision: 1.1 $
+-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output media-type="text/html" method="html" encoding="ISO-8859-1"/>

    <xsl:template match="/">
        <html>
            <head>
                <title>Retail Application</title>
            </head>

```

```

<body>
  <xsl:if test="//pageTitle">
    <h2><xsl:value-of select="//pageTitle"/></h2>
  </xsl:if>
  <xsl:choose>
    <xsl:when test="loginResult">
      <xsl:apply-templates select="loginResult"/>
    </xsl:when>
    <xsl:when test="index">
      <xsl:apply-templates select="index"/>
    </xsl:when>
    <xsl:when test="inventory">
      <xsl:apply-templates select="inventory"/>
    </xsl:when>
    <xsl:when test="order">
      <xsl:apply-templates select="order"/>
    </xsl:when>
    <xsl:when test="placeOrder">
      <xsl:apply-templates select="placeOrder"/>
    </xsl:when>
    <xsl:otherwise>
      <p align="center">
        <h3>This kind of XML Document cannot be processed...</h3>
      </p>
    </xsl:otherwise>
  </xsl:choose>
</body>
</html>
</xsl:template>

<xsl:template match="loginResult">
  <xsl:if test="ROWSET/ROW/unknown">
    <table width="98%">
      <tr><td bgcolor="yellow" align="center"><xsl:value-of
select="ROWSET/ROW/unknown"/> is not allowed to log in !</td></tr>
    </table>
  </xsl:if>
  <xsl:if test="ROWSET/ROW/NAME">
    <p align="center">
      <h2>Welcome <xsl:value-of select="ROWSET/ROW/NAME"/> !</h2>
    </p>
    <p align="center">
      <a>
        <xsl:attribute name="href"><xsl:value-of
select="nextStep"/>?custId=<xsl:value-of select="ROWSET/ROW/ID"/></xsl:attribute>

```

```
        Please enter the Mall !
    </a>
</p>
</xsl:if>
<p>
    <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
</p>
</xsl:template>

<xsl:template match="index">
    <xsl:for-each select="form">
        <center>
            <form>
                <xsl:attribute name="action"><xsl:value-of
select="./@action"/></xsl:attribute>
                <xsl:attribute name="method"><xsl:value-of
select="./@method"/></xsl:attribute>
                <xsl:if test="./field">
                    <table width="98%" border="1">
                        <xsl:for-each select="./field">
                            <tr>
                                <td align="right"><xsl:value-of select="./@prompt"/></td>
                                <td>
                                    <input>
                                        <xsl:choose>
                                            <xsl:when test="./@type = 'text'">
                                                <xsl:attribute name="type">text</xsl:attribute>
                                            </xsl:when>
                                        </xsl:choose>
                                        <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
                                    </input>
                                </td>
                            </tr>
                        </xsl:for-each>
                    </table>
                </xsl:if>
                <xsl:if test="./button">
                    <p>
                        <xsl:for-each select="./button">
                            <input>
                                <xsl:choose>
                                    <xsl:when test="./@type = 'submit'">
```

```

        <xsl:attribute name="type">submit</xsl:attribute>
      </xsl:when>
    </xsl:choose>
    <xsl:attribute name="value"><xsl:value-of
select="..@label"/></xsl:attribute>
  </input>
</xsl:for-each>
</p>
</xsl:if>
</form>
</center>
</xsl:for-each>
</xsl:template>

<xsl:template match="inventory">
  <h2>This is the Mart content</h2>
  <xsl:for-each select="form/theMart/ROWSET/ROW">
    <xsl:value-of select="ID"/>
    <xsl:text> </xsl:text>
    <form method="post">
      <xsl:attribute name="action">
        <xsl:value-of select="../../../../../form/@action"/>
      </xsl:attribute>
      <input type="hidden" name="custId">
        <xsl:attribute name="value"><xsl:value-of
select="../../../../../form/hiddenFields/custId"/></xsl:attribute>
      </input>
      <input type="hidden" name="prodId">
        <xsl:attribute name="value"><xsl:value-of select="ID"/></xsl:attribute>
      </input>
      <input type="submit">
        <xsl:attribute name="value"><xsl:value-of
select="DESCRIPTION"/></xsl:attribute>
      </input>
    </form>
    <xsl:text> @ $</xsl:text><xsl:value-of select="PRICE"/><xsl:text>
each</xsl:text>
    <xsl:text> Supplied by </xsl:text><xsl:value-of select="NAME"/>
    <br/>
  </xsl:for-each>
  <p>
    <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
  </p>

```

```
</xsl:template>

<xsl:template match="order">
  <center>
    <h2>Thank you <xsl:value-of select="CUST/NAME"/> for shopping with us !</h2>
    <hr/>
    <h2>Please enter the quantity</h2>
    <form action="placeOrder.xsql" method="post">
      <input type="hidden" name="prodId">
        <xsl:attribute name="value"><xsl:value-of
select="PROD/ID"/></xsl:attribute>
      </input>
      <input type="hidden" name="custId">
        <xsl:attribute name="value"><xsl:value-of
select="CUST/ID"/></xsl:attribute>
      </input>
      <p>
        <xsl:value-of select="PROD/DESCRIPTION"/>
        at $<xsl:value-of select="PROD/PRICE"/> each
        supplied by <xsl:value-of select="PROD/NAME"/>
      <br/>
        Quantity :
      <br/>
      <input type="text" name="qty"/>
    </p>
    <p><input type="submit" value="Place Order"/></p>
  </form>
</center>
<p>
  <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>
</p>
</xsl:template>

<xsl:template match="placeOrder">
  <xsl:if test="operationResult">
    <center>
      <xsl:value-of select="operationResult/text()"/>
    <br/>
    <xsl:for-each select="operationResult/nextStep">
      <form method="post">
        <xsl:attribute name="action"><xsl:value-of
select="./@Action"/></xsl:attribute>
        <xsl:if test="prmList">
```



```

        <xsl:for-each select="prmList/prm">
            <input type="hidden">
                <xsl:attribute name="name"><xsl:value-of
select="./@name"/></xsl:attribute>
                <xsl:attribute name="value"><xsl:value-of
select="./@value"/></xsl:attribute>
            </input>
        </xsl:for-each>
    </xsl:if>
    <input type="submit">
        <xsl:attribute name="value"><xsl:value-of
select="./@Label"/></xsl:attribute>
    </input>
</form>
</xsl:for-each>
</center>
</xsl:if>
<xsl:if test="operationProblem">
    <table width="98%">
        <tr><td align="center"><font color="red"><xsl:value-of
select="operationProblem"/></font></td></tr>
    </table>
</xsl:if>
<xsl:if test="bottomLinks">
    <xsl:choose>
        <xsl:when test="operationResult">
            </xsl:when>
        <xsl:otherwise>
            <p align="center">
                <xsl:for-each select="bottomLinks/aLink">
                    [<a><xsl:attribute name="href"><xsl:value-of
select="./@href"/></xsl:attribute><xsl:value-of select="."/></a>]
                </xsl:for-each>
            </p>
        </xsl:otherwise>
    </xsl:choose>
</xsl:if>
<xsl:choose>
    <xsl:when test="operationResult/nextStep">
        </xsl:when>
    <xsl:otherwise>
        <xsl:if test="returnHome">
            <p>
                <a><xsl:attribute name="href"><xsl:value-of
select="returnHome"/></xsl:attribute>Back to Login</a>

```

```
</p>
</xsl:if>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Java の例 3: スタイルシートの管理 - GUIInterface.java

このスクリプトは、B2B XML アプリケーションで使用する GUI およびスタイルシートを作成および管理します。

```
package B2BDemo.StyleSheetUtil;
/**
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.sql.*;
import java.util.*;
// 新しい CLOB クラスおよび BLOB クラスが必要です。
import oracle.sql.*;
import oracle.jdbc.driver.*;
import java.beans.*;
import javax.swing.event.*;

import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class GUIInterface extends JFrame
{
    private boolean lite = false; // O8iLite を使用します。
    private boolean inserting = false;

    private final static int UPDATE = 1;
    private final static int INSERT = 2;

    private final static int ENTER_QUERY = 1;
    private final static int EXEC_QUERY = 2;

    int queryState = ENTER_QUERY;
```

```
String sqlStmt = "Select APPFROM, " +
                  "      APPTO, " +
                  "      OP, " +
                  "      XSL " +
                  "From styleSheets";

private static String connURL = AppCste.AQDBUrl;
private static String userName = AppCste.AQuser;
private static String password = AppCste.AQpswd;
private Connection conn = null;

private Vector recVect = null;
int currRec = 0;
XslRecord thisRecord = null;

BorderLayout borderLayout1 = new BorderLayout();
JPanel jPanel1 = new JPanel();
JMenuBar menuBar1 = new JMenuBar();
JMenu menuFile = new JMenu();
JMenuItem menuFileExit = new JMenuItem();
JMenu menuHelp = new JMenu();
JMenuItem menuHelpAbout = new JMenuItem();
JLabel statusBar = new JLabel();
JToolBar toolBar = new JToolBar();
JButton buttonOpen = new JButton();
JButton buttonClose = new JButton();
JButton buttonHelp = new JButton();
ImageIcon imageOpen;
ImageIcon imageClose;
ImageIcon imageHelp;
JPanel jPanel2 = new JPanel();
BorderLayout borderLayout2 = new BorderLayout();
JButton firstButton = new JButton();
JPanel jPanel3 = new JPanel();
JPanel jPanel4 = new JPanel();
BorderLayout borderLayout3 = new BorderLayout();
BorderLayout borderLayout4 = new BorderLayout();
JPanel jPanel5 = new JPanel();
JTextField fromAppValue = new JTextField();
JLabel fromApp = new JLabel();
JPanel jPanel6 = new JPanel();
BorderLayout borderLayout5 = new BorderLayout();
JLabel jLabel2 = new JLabel();
JScrollPane jScrollPane1 = new JScrollPane();
```

```
JTextArea XSLStyleSheet = new JTextArea();
JButton previousButton = new JButton();
JButton nextButton = new JButton();
JButton lastButton = new JButton();
JButton validateButton = new JButton();
GridLayout gridLayout1 = new GridLayout();
JLabel toApp = new JLabel();
JTextField toAppValue = new JTextField();
JLabel operationLabel = new JLabel();
JTextField opValue = new JTextField();
JButton newButton = new JButton();
JButton deleteButton = new JButton();
JButton queryButton = new JButton();

public GUIInterface()
{
    super();
    try
    {
        jbInit();
        buttonOpen.setIcon(imageOpen);
        buttonClose.setIcon(imageClose);
        buttonHelp.setIcon(imageHelp);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

private void getConnected() throws Exception
{
    try
    {
        if (lite)
        {
            Class.forName("oracle.lite.poljdbc.POLJDBCDriver");
            conn = DriverManager.getConnection("jdbc:Polite:POLite", "system",
"manager");
        }
        else
        {
            Class.forName ("oracle.jdbc.driver.OracleDriver");
            conn = DriverManager.getConnection (connURL, userName, password);
        }
    }
}
```

```
    }
    catch (Exception e)
    {
        System.err.println("Get connected failed : " + e);
        throw e;
    }
}

private void jbInit() throws Exception
{
    if (conn == null)
    {
        try { getConnected(); }
        catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, e.toString(),
                                         "Connection",
                                         JOptionPane.ERROR_MESSAGE);

            System.exit(1);
        }
    }
    imageOpen = new ImageIcon(GUInterface.class.getResource("openfile.gif"));
    imageClose = new ImageIcon(GUInterface.class.getResource("closefile.gif"));
    imageHelp = new ImageIcon(GUInterface.class.getResource("help.gif"));
    this.setTitle("Style Sheets Management");
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(511, 526));
    jPanel1.setLayout(borderLayout2);
    menuFile.setText("File");
    menuFileExit.setText("Exit");
    menuFileExit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            fileExit_ActionPerformed(e);
        }
    });
    menuHelp.setText("Help");
    menuHelpAbout.setText("About");
    menuHelpAbout.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            helpAbout_ActionPerformed(e);
        }
    });
    statusBar.setText("Initializing...");
    buttonOpen.setToolTipText("Open File");
    buttonClose.setToolTipText("Validate modifications");
}
```

```
buttonHelp.setToolTipText("About Style Sheet Manager");
firstButton.setText("<<");
jPanel5.setLayout(gridLayout1);
fromApp.setText("From Application :");
fromApp.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel2.setText("XSL Style Sheet");
previousButton.setText("<");
nextButton.setText(">");
lastButton.setText(">>");
validateButton.setText("Validate");
gridLayout1.setRows(4);
toApp.setText("To Application : ");
toApp.setHorizontalAlignment(SwingConstants.RIGHT);
operationLabel.setText("Operation : ");
operationLabel.setHorizontalAlignment(SwingConstants.RIGHT);
jPanel6.setLayout(borderLayout5);
jPanel4.setLayout(borderLayout4);
jPanel3.setLayout(borderLayout3);
menuFile.add(menuFileExit);
menuBar1.add(menuFile);
menuHelp.add(menuHelpAbout);
menuBar1.add(menuHelp);
this.setJMenuBar(menuBar1);
this.getContentPane().add(statusBar, BorderLayout.SOUTH);
toolBar.add(buttonOpen);
toolBar.add(buttonClose);
toolBar.add(buttonHelp);
this.getContentPane().add(toolBar, BorderLayout.NORTH);
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(queryButton, null);
jPanel2.add(newButton, null);
jPanel2.add(firstButton, null);
jPanel2.add(previousButton, null);
jPanel2.add(nextButton, null);
jPanel2.add(lastButton, null);
jPanel2.add(validateButton, null);
jPanel2.add(deleteButton, null);
jPanel1.add(jPanel3, BorderLayout.CENTER);
jPanel3.add(jPanel4, BorderLayout.NORTH);
jPanel3.add(jPanel5, BorderLayout.SOUTH);
jPanel5.add(fromApp, null);
jPanel5.add(fromAppValue, null);
jPanel5.add(toApp, null);
jPanel5.add(toAppValue, null);
```

```
jPanel5.add(operationLabel, null);
jPanel5.add(opValue, null);
jPanel3.add(jPanel6, BorderLayout.CENTER);
jPanel6.add(jLabel2, BorderLayout.NORTH);
jPanel6.add(jScrollPane1, BorderLayout.CENTER);
jScrollPane1.getViewport().add(XSLStyleSheet, null);

//
statusBar.setText("Connected...");
// レコードのベクトルを構築中です。
queryButton.setText("Enter Query");
queryButton.setActionCommand("query");
queryButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        queryButton_actionPerformed(e);
    }
});
buttonClose.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        buttonClose_actionPerformed(e);
    }
});
deleteButton.setText("Delete");
deleteButton.setToolTipText("Delete the current record");
deleteButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        deleteButton_actionPerformed(e);
    }
});
newButton.setText("New");
newButton.setToolTipText("Create a new record");
newButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        newButton_actionPerformed(e);
    }
});
validateButton.setToolTipText("Validate your modifications");
```

```
opValue.setEditable(false);
toAppValue.setEditable(false);
fromAppValue.setEditable(false);
validateButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        validateButton_actionPerformed(e);
    }
});
lastButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        lastButton_actionPerformed(e);
    }
});
firstButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        firstButton_actionPerformed(e);
    }
});
previousButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        previousButton_actionPerformed(e);
    }
});
nextButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        nextButton_actionPerformed(e);
    }
});
lastButton.setActionCommand("last");
lastButton.setToolTipText("Last record");
nextButton.setActionCommand("next");
nextButton.setToolTipText("Next record");
previousButton.setActionCommand("previous");
previousButton.setToolTipText("Previous record");
firstButton.setActionCommand("first");
```



```
firstButton.setToolTipText("First record");

// 問合せを実行し、ベクトルを構築します。
executeQuery(sqlStmt);

updateStatusBar();
}

void executeQuery(String theSqlStmt)
{
    recVect = new Vector();
    try
    {
        Statement stmt = conn.createStatement();
        ResultSet rSet = stmt.executeQuery(theSqlStmt);
        CLOB clob = null;
        while (rSet.next())
        {
            clob = ((OracleResultSet)rSet).getCLOB(4);
            String strLob = dumpClob(conn, clob);
            XslRecord xslRecord = new XslRecord(rSet.getString(1),
                                                rSet.getString(2),
                                                rSet.getString(3),
                                                strLob);

            recVect.addElement(xslRecord);
        }
        rSet.close();
        stmt.close();
        // フォームに最初のレコードを移入します。
        firstButton.setEnabled(false);
        previousButton.setEnabled(false);
        nextButton.setEnabled(false);
        lastButton.setEnabled(false);
        if (recVect.size() > 0)
        {
            currRec = 1;
            displayRecord(currRec);
        }
        if (recVect.size() > 1)
        {
            nextButton.setEnabled(true);
            lastButton.setEnabled(true);
        }
    }
    catch (Exception e)
```

```
{
    JOptionPane.showMessageDialog(null, e.toString(),
                                   "Executing request",
                                   JOptionPane.ERROR_MESSAGE);

    System.exit(1);
}
}

void displayRecord(int mk)
{
    XslRecord xslRecord = (XslRecord)recVect.elementAt(mk-1);
    thisRecord = new XslRecord(xslRecord.FROM,
                               xslRecord.TO,
                               xslRecord.TASK,
                               xslRecord.XSL);

    XSLStyleSheet.setText(xslRecord.XSL);
    fromAppValue.setText(xslRecord.FROM);
    toAppValue.setText(xslRecord.TO);
    opValue.setText(xslRecord.TASK);

    XSLStyleSheet.requestFocus();
    XSLStyleSheet.setCaretPosition(0);

    // ボタン
    firstButton.setEnabled(false);
    previousButton.setEnabled(false);
    nextButton.setEnabled(false);
    lastButton.setEnabled(false);
    if (mk > 1)
    {
        firstButton.setEnabled(true);
        previousButton.setEnabled(true);
    }
    if (mk < recVect.size())
    {
        nextButton.setEnabled(true);
        lastButton.setEnabled(true);
    }
}

void updateStatusBar()
{
    statusBar.setText("Ready for " + recVect.size() + " records");
}
```

```
private static String dumpClob(Connection conn, CLOB clob) throws Exception
{
    String returnString = "";

    OracleCallableStatement cStmt1 = (OracleCallableStatement) conn.prepareCall
("begin ? := dbms_lob.getLength (?); end;");
    OracleCallableStatement cStmt2 = (OracleCallableStatement) conn.prepareCall
("begin dbms_lob.read (?, ?, ?, ?); end;");

    cStmt1.registerOutParameter (1, Types.NUMERIC);
    cStmt1.setCLOB (2, clob);
    cStmt1.execute ();

    long length = cStmt1.getLong (1);
    long i = 0;
    int chunk = 80;

    while (i < length)
    {
        cStmt2.setCLOB (1, clob);
        cStmt2.setLong (2, chunk);
        cStmt2.registerOutParameter (2, Types.NUMERIC);
        cStmt2.setLong (3, i + 1);
        cStmt2.registerOutParameter (4, Types.VARCHAR);
        cStmt2.execute ();

        long read_this_time = cStmt2.getLong (2);
        returnString += cStmt2.getString (4);
        // System.out.print ("Read " + read_this_time + " chars: ");
        // System.out.println (string_this_time);
        i += read_this_time;
    }
    cStmt1.close ();
    cStmt2.close ();
    return returnString;
}

static void fillClob (Connection conn, CLOB clob, String str) throws SQLException
{
    OracleCallableStatement cStmt =
        (OracleCallableStatement) conn.prepareCall ("begin dbms_lob.write (?, ?, ?,
?); end;");

    int i = 0;
```

```
int chunk = 80;
int length = str.length();
long c, ii;

System.out.println("Length: " + length + "\n" + str);
while (i < length)
{
    cStmt.setClob (1, clob);
    c = chunk;
    cStmt.setLong (2, c);
    ii = i + 1;
    cStmt.setLong (3, ii);
    cStmt.setString (4, str.substring(i, i + chunk));
    cStmt.execute ();
    i += chunk;
    if (length - i < chunk)
        chunk = length - i;
}
cStmt.close ();
}

void fileExit_ActionPerformed(ActionEvent e)
{
    if (conn != null)
    {
        try { conn.close(); } catch (Exception ex) {}
    }
    System.exit(0);
}

void helpAbout_ActionPerformed(ActionEvent e)
{
    JOptionPane.showMessageDialog(this, new GUIInterface_AboutBoxPanell1(), "About",
JOptionPane.PLAIN_MESSAGE);
}

void nextButton_actionPerformed(ActionEvent e)
{
    checkRecordChange();
    currRec++;
    displayRecord(currRec);
}

void previousButton_actionPerformed(ActionEvent e)
{

```

```
        checkRecordChange();
        currRec--;
        displayRecord(currRec);
    }

    void firstButton_actionPerformed(ActionEvent e)
    {
        checkRecordChange();
        currRec = 1;
        displayRecord(currRec);
    }

    void lastButton_actionPerformed(ActionEvent e)
    {
        checkRecordChange();
        currRec = recVect.size();
        displayRecord(currRec);
    }

    void validateButton_actionPerformed(ActionEvent e)
    {
        validateRec();
    }

    void validateRec()
    {
        thisRecord = new XslRecord(fromAppValue.getText(),
                                   toAppValue.getText(),
                                   opValue.getText(),
                                   XSLStyleSheet.getText());
        if (saveChanges(thisRecord, (inserting?INSERT:UPDATE)))
            JOptionPane.showMessageDialog(null, "All right!");
    }

    void deleteRec()
    {
        thisRecord = new XslRecord(fromAppValue.getText(),
                                   toAppValue.getText(),
                                   opValue.getText(),
                                   XSLStyleSheet.getText());
        String sqlStmt = "delete styleSheets where fromApp = ? and " +
                        "                                toApp = ? and " +
                        "                                op      = ?";

        try
        {
```

```
        PreparedStatement pstmt = conn.prepareStatement(sqlStmt);
        pstmt.setString(1, thisRecord.FROM);
        pstmt.setString(2, thisRecord.TO);
        pstmt.setString(3, thisRecord.TASK);
        pstmt.execute();
        conn.commit();
        System.out.println("Deleted !");
        pstmt.close();
        // ベクトルから削除します。
        recVect.removeElementAt(currRec - 1);
        updateStatusBar();
        if (currRec >= recVect.size())
            currRec--;
        displayRecord(currRec);
        JOptionPane.showMessageDialog(null, "All right!");
    }
    catch (SQLException sqlE)
    {
        JOptionPane.showMessageDialog(null, sqlE.toString(),
                                      "Deleting record",
                                      JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception e)
    {
        JOptionPane.showMessageDialog(null, e.toString(),
                                      "Deleting record",
                                      JOptionPane.ERROR_MESSAGE);
    }
}

void checkRecordChange()
{
    thisRecord = new XslRecord(fromAppValue.getText(),
                              toAppValue.getText(),
                              opValue.getText(),
                              XSLStyleSheet.getText());
    if (!thisRecord.equals((XslRecord) recVect.elementAt(currRec-1)))
    {
        int result = JOptionPane.showConfirmDialog(null, "Record has changed\nDo you
        want to save the modifications ?");
        if (result == JOptionPane.YES_OPTION)
        {
            saveChanges(thisRecord, UPDATE);
            JOptionPane.showMessageDialog(null, "All right!");
        }
    }
}
```

```

    }
}

boolean saveChanges(XslRecord rec,
                    int operation)
{
    boolean ret = true;
    if (operation == this.UPDATE)
    {
        String theSqlStmt = "update styleSheets set xsl = ? where appFrom = ? and
appTo = ? and op = ?";
        try
        {
            PreparedStatement pStmt = conn.prepareStatement(theSqlStmt);
            pStmt.setString(1, rec.XSL);
            pStmt.setString(2, rec.FROM);
            pStmt.setString(3, rec.TO);
            pStmt.setString(4, rec.TASK);
            pStmt.execute();
            conn.commit();
            System.out.println("Updated !");
            pStmt.close();
            // ベクトルに再挿入します。
            recVect.setElementAt(rec, currRec - 1);
        }
        catch (SQLException sqlE)
        {
            JOptionPane.showMessageDialog(null, sqlE.toString(),
                                         "Saving record",
                                         JOptionPane.ERROR_MESSAGE);

            ret = false;
        }
    }
    else
    {
        System.out.println("Inserting new record");
        String sqlStmt = "insert into styleSheets " +
            "          ( appFrom,  " +
            "              appTo,    " +
            "              op,        " +
            "              xsl        " +
            "          ) values    " +
            "          (?, ?, ?, ?)";
        String sqlGetLob = "select xsl from styleSheets " +
            "where appFrom = ? and " +

```

```
        "        appTo    = ? and " +
        "        op       = ?";

try
{
    PreparedStatement pStmt = conn.prepareStatement(sqlStmt);
    pStmt.setString(1, rec.FROM);
    pStmt.setString(2, rec.TO);
    pStmt.setString(3, rec.TASK);
    pStmt.setString(4, ""); // LOB 内の NULL は、後で記入されます。
    pStmt.execute();
    System.out.println("Inserted !");
    pStmt.close();

    PreparedStatement fillLOBStmt = conn.prepareStatement(sqlGetLob);
    fillLOBStmt.setString(1, rec.FROM);
    fillLOBStmt.setString(2, rec.TO);
    fillLOBStmt.setString(3, rec.TASK);
    ResultSet lobRSet = fillLOBStmt.executeQuery();
    while (lobRSet.next())
    {
        CLOB clob = ((OracleResultSet)lobRSet).getCLOB(1);
        fillClob(conn, clob, rec.XSL);
    }
    conn.commit();

    // ベクトルに追加します。
    recVect.addElement(rec);
    currRec = recVect.size();
    displayRecord(currRec);
}
catch (SQLException sqlE)
{
    JOptionPane.showMessageDialog(null, sqlE.toString(),
                                   "Inserting record",
                                   JOptionPane.ERROR_MESSAGE);

    ret = false;
}

inserting = false;

fromAppValue.setEditable(false);
toAppValue.setEditable(false);
opValue.setEditable(false);
}
updateStatusBar();
```



```
        return ret;
    }

    void buttonClose_actionPerformed(ActionEvent e)
    {
        validateRec();
    }

    void newButton_actionPerformed(ActionEvent e)
    {
        fromAppValue.setEditable(true);
        toAppValue.setEditable(true);
        opValue.setEditable(true);
        inserting = true;
        XSLStyleSheet.setText("");
        fromAppValue.setText("");
        toAppValue.setText("");
        opValue.setText("");
    }

    void deleteButton_actionPerformed(ActionEvent e)
    {
        deleteRec();
    }

    void queryButton_actionPerformed(ActionEvent e)
    {
        if (queryState == ENTER_QUERY)
        {
            queryState = EXEC_QUERY;
            queryButton.setText("Execute Query");
            fromAppValue.setEditable(true);
            toAppValue.setEditable(true);
            opValue.setEditable(true);

            XSLStyleSheet.setEditable(false);
            statusBar.setText("Entering query");
            XSLStyleSheet.setText("");
            fromAppValue.setText("");
            toAppValue.setText("");
            opValue.setText("");

            newButton.setEnabled(false);
            firstButton.setEnabled(false);
            previousButton.setEnabled(false);
        }
    }
}
```

```
        nextButton.setEnabled(false);
        lastButton.setEnabled(false);
        validateButton.setEnabled(false);
        deleteButton.setEnabled(false);
    }
    else
    {
        queryState = ENTER_QUERY;
        queryButton.setText("Enter Query");
        statusBar.setText("Executing query");

        fromAppValue.setEditable(false);
        toAppValue.setEditable(false);
        opValue.setEditable(false);
        XSLStyleSheet.setEditable(true);

        newButton.setEnabled(true);
        firstButton.setEnabled(true);
        previousButton.setEnabled(true);
        nextButton.setEnabled(true);
        lastButton.setEnabled(true);
        validateButton.setEnabled(true);
        deleteButton.setEnabled(true);

        // 問合せを実行します。
        String stmt = sqlStmt;
        boolean firstCondition = true;
        if (fromAppValue.getText().length() > 0)
        {
            stmt += ((firstCondition?" where ":" and ") + "fromApp like '" +
fromAppValue.getText() + "' ");
            firstCondition = false;
        }
        if (toAppValue.getText().length() > 0)
        {
            stmt += ((firstCondition?" where ":" and ") + "toApp like '" +
toAppValue.getText() + "' ");
            firstCondition = false;
        }
        if (opValue.getText().length() > 0)
        {
            stmt += ((firstCondition?" where ":" and ") + "op like '" +
opValue.getText() + "' ");
            firstCondition = false;
        }
    }
}
```

```

        executeQuery(stmt);
        updateStatusBar();
        displayRecord(currRec);
    }
}
}

```

Java の例 4: GUIInterface_AboutBoxPanel.java

```

package B2BDemo.StyleSheetUtil;
/**
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import oracle.jdeveloper.layout.*;

public class GUIInterface_AboutBoxPanel1 extends JPanel
{
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JLabel jLabel4 = new JLabel();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    Border border1 = new EtchedBorder();

    public GUIInterface_AboutBoxPanel1()
    {
        try
        {
            jbInit();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception
    {
        jLabel1.setText("Stored Style Sheets management.");
        jLabel2.setText("Olivier LE DIOURIS");
    }
}

```

```
jLabel3.setText("Copyright (c) 1999");
jLabel4.setText("Oracle Corp.");
this.setLayout(gridBagLayout1);
this.setBorder(border1);
this.add(jLabel1, new GridBagConstraints2(0, 0, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(5,5,0,5),0,0));
this.add(jLabel2, new GridBagConstraints2(0, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,0,5),0,0));
this.add(jLabel3, new GridBagConstraints2(0, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,0,5),0,0));
this.add(jLabel4, new GridBagConstraints2(0, 3, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0,5,5,5),0,0));
    }
}
```

Java の例 5: GUIStylesheet.java

```
package B2BDemo.StyleSheetUtil;
/**
 * データベースのAQスキーマに格納されたスタイルシートを操作するためのグラフィカル・
 * ユーティリティです。スタイルシートは、受信ドキュメントを送信ドキュメントに変換するために
 * 使用されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

public class GUIStylesheet
{
    private static final boolean useBali = false;

    public GUIStylesheet()
    {
        Frame frame = new GUIInterface();
        // ウィンドウを中央に配置します。
        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.setVisible(true);
}

public static void main(String[] args)
{
    new GUIStylesheet();
}
}
```

XSL 処理および管理スクリプト

B2B XML アプリケーションで使用する XML 処理スクリプトおよび管理スクリプトは、次のとおりです。

- [Java の例 6: Main4XMLtoDMLv2.java](#)
- [Java の例 7: ParserTest.java](#)
- [Java の例 8: TableInDocument.java](#)
- [Java の例 9: XMLFrame.java](#)
- [Java の例 10: XMLProducer.java](#)
- [Java の例 11: XMLtoDMLv2.java](#)
- [Java の例 12: XMLGen.java](#)
- [Java の例 13: XMLUtil.java](#)
- [Java の例 14: XSLTWrapper.java](#)

Java の例 6: Main4XMLtoDMLv2.java

```
package B2BDemo.XMLUtil;
/**
 * テスト用の主要部分です。
 * XMLtoDMLv2 ユーティリティは、複数の表に挿入されるデータを含むことができる XML ドキュメント
 * を取得します。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.io.*;
import java.net.*;

public class Main4XMLtoDMLv2 extends Object
{
    // ユーザー入力を管理します。
    private static BufferedReader _stdin = new BufferedReader(new
InputStreamReader(System.in));
    private static String _buf = "";

    private static String _userInput(String prompt) throws Exception
    {
        String retString;
        System.out.print(prompt);
        try { retString = _stdin.readLine(); }
    }
}
```

```

        catch (Exception e)
        {
            System.out.println(e);
            throw(e);
        }
        return retString;
    }
    // テスト用
    public static void main(String args[])
    {
        XMLtoDMLv2 x2d = new XMLtoDMLv2("scott", "tiger",

jdbc:oracle:thin:@olediour-lap.us.oracle.com:1521:Ora8i");

        String xmldocname = "";
        try { xmldocname = userInput("XML file name > "); }
        catch (Exception e) {}
        String xmldoc = readURL(createURL(xmldocname));

        TableInDocument d[] = new TableInDocument[2];
        d[0] = new TableInDocument("ROWSET", "ROW", "DEPT");
        d[1] = new TableInDocument("EMP", "EMP_ROW", "EMP");

        try
        {
            x2d.insertFromXML(d, xmldoc);
            System.out.println(xmldocname + " processed.");
        }
        catch (Exception e)
        {
            System.err.println("Oops:\n" + e);
        }

        try { _buf = _userInput("End of task..."); } catch (Exception ioe) {}
    }

    public 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();
    // この大量のコードは、getAbsolutePath の戻り値に一貫性がないために、
    // Windows プラットフォームで URL を有効にするために必要です。
    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.err.println("Cannot create url for: " + fileName);
    System.exit(0);
}
}
return url;
}

public static String readURL(URL url)
{
    URLConnection newURLConn;
    BufferedInputStream newBuff;
    int nBytes;
    byte aByte[];
    String resultBuff = "";

    aByte = new byte[2];
    try
    {
        // System.out.println("Calling " + url.toString());
        try
        {
            newURLConn = url.openConnection();
            newBuff = new BufferedInputStream(newURLConn.getInputStream());
            resultBuff = "";
```



```

        while (( nBytes = newBuff.read(aByte, 0, 1)) != -1)
            resultBuff = resultBuff + (char)aByte[0];
    }
    catch (IOException e)
    {
        System.err.println(url.toString() + "\n : newURLConnection failed :\n" + e);
    }
}
catch (Exception e) {}
return resultBuff;
}

private static String userInput(String prompt) throws Exception
{
    String retString;
    System.out.print(prompt);
    try { retString = _stdin.readLine(); }
    catch (Exception e)
    {
        System.out.println(e);
        throw(e);
    }
    return retString;
}
}

```

Java の例 7: ParserTest.java

```

package B2BDemo.XMLUtil;

import org.xml.sax.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;
/**

```

```
* テスト用の主要部分のみです。
* XML ドキュメントから ID および CUSTOMER_ID を取り出す方法を示します。
*
* @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
*/
public class ParserTest extends Object
{

    static DOMParser parser = new DOMParser();

    static String XMLDoc =
"<ROWSET>" +
"  <ROW NUM=\"1\">" +
"    <ID>23321</ID>" +
"    <ORDERDATE>2000-05-03 00:00:00.0</ORDERDATE>" +
"    <CONTACTNAME>JDevBC4J</CONTACTNAME>" +
"    <TRACKINGNO>AX23321</TRACKINGNO>" +
"    <STATUS>Pending</STATUS>" +
"    <ITEMS>" +
"      <ITEM_ROW NUM=\"1\">" +
"        <ID>1242</ID>" +
"        <QUANTITY>2</QUANTITY>" +
"        <ITEM_ID>403</ITEM_ID>" +
"        <ORD_ID>23321</ORD_ID>" +
"        <DISCOUNT>0</DISCOUNT>" +
"      </ITEM_ROW>" +
"    </ITEMS>" +
"  </ROW>" +
"</ROWSET>";
/**
 * コンストラクタ
 */
public ParserTest()
{
}

public static void main(String[] args)
{
    parser.setValidationMode(false);
    try
    {
        parser.parse(new InputSource(new ByteArrayInputStream(XMLDoc.getBytes())));
        XMLDocument xml = parser.getDocument();
        XMLElement elmt = (XMLElement)xml.getDocumentElement();
```

```

NodeList nl = elmt.getElementsByTagName("ID"); // ORD ID
for (int i=0; i<nl.getLength(); i++)
{
    XMLElement ordId = (XMLElement)nl.item(i);
    XMLNode theText = (XMLNode)ordId.getFirstChild();
    String ordIdValue = theText.getNodeValue();
    System.out.println(ordIdValue);
    break;
}
nl = elmt.getElementsByTagName("CUSTOMER_ID"); // 顧客 ID
for (int i=0; i<nl.getLength(); i++)
{
    XMLElement ordId = (XMLElement)nl.item(i);
    XMLNode theText = (XMLNode)ordId.getFirstChild();
    String custIdValue = theText.getNodeValue();
    System.out.println(custIdValue);
}
}
catch (SAXParseException e)
{
    System.out.println(e.getMessage());
}
catch (SAXException e)
{
    System.out.println(e.getMessage());
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
}
}

```

Java の例 8: TableInDocument.java

```

package B2BDemo.XMLUtil;
/**
 * このクラスは、XMLtoDMLv2.java クラスが使用します。
 * このクラスは、XML ドキュメントと SQL 表の一致を記述します。
 * 複数レベルの XML ドキュメント（マスター・ディテール）を管理するために作成されています。
 *
 * @see XMLtoDMLv2
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */

```

```
public class TableInDocument extends Object
{
    public String rowSet = "ROWSET";
    public String row    = "ROW";
    public String table  = "";

    public TableInDocument (String rset,
                            String r,
                            String t)
    {
        this.rowSet = rset;
        this.row    = r;
        this.table  = t;
    }
}
```

Java の例 9: XMLFrame.java

```
// Copyright (c) 2000 Oracle Corporation
package B2BDemo.XMLUtil;

import javax.swing.*.*;
import java.awt.*.*;
import oracle.xml.srcviewer.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

/**
 * Swing ベースの最上位ウィンドウ・クラスです。
 * Transviewer Bean の Code View を実装します。
 * あるコンポーネントから他のコンポーネントへ伝播される XML コードを拡張するために
 * デモ内で使用されます。
 *
 * @author Olivier LE DIOURIS
 */
public class XMLFrame extends JFrame
{
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    BorderLayout borderLayout2 = new BorderLayout();
    XMLSourceView xmlSourceViewPanel = new XMLSourceView();
}
```

```
private String frameTitle = "";
private XSLTWrapper xsltw = new XSLTWrapper();
/**
 * 新しいインスタンスを構築します。
 */
public XMLFrame(String fTitle)
{
    super();
    this.frameTitle = fTitle;
    try
    {
        jbInit();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

/**
 * このインスタンスの状態を初期化します。
 */
private void jbInit() throws Exception
{
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    jPanel1.setLayout(borderLayout2);
    this.setTitle(this.frameTitle);
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(xmlSourceViewPanel, BorderLayout.CENTER);
}

public void setXMLDocument(String xmlContent) throws Exception
{
    xmlSourceViewPanel.setXMLDocument(xsltw.parseDocument(xmlContent));
}
}
```

Java の例 10: XMLProducer.java

```
package B2BDemo.XMLUtil;
/**
 * XML SQL Utility のラッパーです。
 * SQL 問合せの後に XML ドキュメントを作成するために、
 * サーブレットのみでなくどの Java オブジェクトからでもコールできます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.xml.sql.query.*;

public class XMLProducer
{
    Connection conn = null;
    String rowset = null;
    String row = null;

    public XMLProducer(Connection conn)
    {
        this.conn = conn;
    }

    public String getXMLString(ResultSet rSet)
    {
        return getXMLString(rSet, "N");
    }

    public String getXMLString(ResultSet rSet,
                               String DTD)
    {
        String finalDoc = "";

        try
        {
            boolean dtdRequired = false;
            if (DTD != null && DTD.length() > 0 && DTD.toUpperCase().equals("Y"))
                dtdRequired = true;
            // スキル //////////////////////////////////////
            OracleXMLQuery oXmlq = new OracleXMLQuery(conn, rSet); //
            // oXmlq.useUpperCaseTagNames(); //
            if (this.rowset != null)
                oXmlq.setRowsetTag(this.rowset);
            if (this.row != null)
```

```

        oXmlq.setRowTag(this.row);
        finalDoc = oXmlq.getXMLString(dtdRequired); //
        // これで終わりです。////////////////////////////////////
    }
    catch (Exception e)
    {
        System.err.println(e);
    }
    return finalDoc;
}

public void setRowset(String rSet)
{
    this.rowset = rSet;
}
public void setRow(String row)
{
    this.row = row;
}
}

```

Java の例 11: XMLtoDMLv2.java

```

package B2BDemo.XMLUtil;
/**
 * このクラスは、入力として XML ドキュメントを受け入れ、
 * データベースへの挿入を実行します。
 * 複数レベルの XML ドキュメントはサポートしていますが、
 * 次のように、1つの要素が複数の子を持つ場合はサポートしていません。
 *
 *      <elem1>
 *          <elem11/>
 *          <elem12/>
 *      </elem1>
 *
 * @see TableInDocument
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import org.xml.sax.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

```

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

public class XMLtoDMLv2 extends Object
{
    static DOMParser parser = new DOMParser();
    Connection conn = null;
    String username = "";
    String password = "";
    String connURL = "";

    public XMLtoDMLv2(String username,
                      String password,
                      String connURL)
    {
        this.username = username;
        this.password = password;
        this.connURL = connURL;
    }

    public void insertFromXML(TableInDocument tInDoc[],
                             String document) throws Exception
    {
        if (conn == null)
            getConnected();

        String xmlString = "";
        try
        { xmlString = readURL(createURL(document)); }
        catch (Exception e)
        { xmlString = document; }

        // System.out.println("xml2Insert = \n" + xmlString);

        // 戻された文字列はXMLドキュメントに変換されます。
        XMLDocument xmlDoc = parseDocument(xmlString);
        // また、このドキュメントのルートに関する参照を取り出します。
        XMLElement e = (XMLElement) xmlDoc.getDocumentElement();

        // ROW ノード内を検索します。
        NodeList nl = e.getChildrenByTagName(tInDoc[0].row); // "ROW"
        // System.out.println("This document has " + nl.getLength() + " ROW(s)");

        Vector sqlStmt = new Vector();
        scanLevel(0, tInDoc, nl, sqlStmt);
    }
}
```



```

// ここで、ベクトル内のすべての文を逆順 (FK...) に実行します。
int i = sqlStmt.size();
Enumeration enum = sqlStmt.elements();
while (i > 0)
{
    i--;
    String s = (String)sqlStmt.elementAt(i);
    // System.out.println("Executing " + s);
    executeStatement(s);
}
}

// これは再帰的です。
private int scanLevel(int level,
                      TableInDocument tInDoc[],
                      NodeList nl,
                      Vector sqlStmt) throws Exception
{
    int nbRowProcessed = 0;
    Vector columnNames = new Vector();
    Vector columnValues = null;
    String[] colTypes = null;

    String columns = "", values = "";
    // ツリー内でループします。
    boolean firstLoop = true;
    for (int i=0; i<nl.getLength(); i++) //XML ドキュメントのすべての行でループします。
    {
        columnValues = new Vector();
        XMLElement aRow = (XMLElement) nl.item(i);
        // String numVal = aRow.getAttribute("num");
        // System.out.println("NUM = " + numVal);
        NodeList nlRow = aRow.getChildNodes();
        // System.out.println("a Row has " + nlRow.getLength() + " children");
        for (int j=0; j<nlRow.getLength(); j++)
        {
            XMLElement anXMLElement = (XMLElement)nlRow.item(j);
            if (anXMLElement.getChildNodes().getLength() == 1 &&
                (level == (tInDoc.length - 1) || (level < (tInDoc.length - 1) &&
                !(anXMLElement.getNodeName().equals(tInDoc[level+1].rowSet)))) )
            {
                // System.out.println("Element " + (j+1) + "=" + anXMLElement.getNodeName());
                // System.out.print(anXMLElement.getNodeName());
            }
        }
    }
}

```

```
        if (firstLoop)
            columnNames.addElement(anXMLElement.getNodeName());
        // 値
        XMLNode nodeValue = (XMLNode) anXMLElement.getFirstChild();
        // System.out.println("\t" + nodeValue.getNodeValue());
        columnValues.addElement(nodeValue.getNodeValue());
    }
    else
    {
        // System.out.println(anXMLElement.getNodeName() + " has " +
        anXMLElement.getChildNodes().getLength() + " children");
        // System.out.println("Comparing " + anXMLElement.getNodeName() + " and " +
        tInDoc[level+1].rowSet);
        if (level < (tInDoc.length - 1) &&
        anXMLElement.getNodeName().equals(tInDoc[level+1].rowSet))
        {
            // System.out.println("Searching for " + tInDoc[level+1].row);
            NodeList nl2 = anXMLElement.getChildrenByTagName(tInDoc[level+1].row);
            // 行
            if (nl2 == null)
                System.out.println("Nl2 is null for " + tInDoc[level+1].row);
            scanLevel(level + 1, tInDoc, nl2, sqlStmt);
        }
    }
}

// System.out.println("INSERT INTO " + tableName + " (" + columns + ") VALUES ("
+ values + ")");
try
{
    if (firstLoop)
    {
        firstLoop = false;
        String selectStmt = "SELECT ";
        boolean comma = false;
        Enumeration cNames = columnNames.elements();
        while (cNames.hasMoreElements())
        {
            columns += ((comma?", ":"") + (String)cNames.nextElement());
            if (!comma)
                comma = true;
        }
        selectStmt += columns;
        selectStmt += (" FROM " + tInDoc[level].table + " WHERE 1 = 2"); // 取り出
        された行はありません。
        Statement stmt = conn.createStatement();
    }
}
```

```

//      System.out.println("Executing: " + selectStmt);
      ResultSet rSet = stmt.executeQuery(selectStmt);
      ResultSetMetaData rsmd = rSet.getMetaData();
      colTypes = new String[rsmd.getColumnCount()];
      for (int ci=0; ci<(rsmd.getColumnCount()); ci++)
      {
        // System.out.println("Col " + (ci+1) + ":" + rsmd.getColumnName(ci+1) + ",
" + rsmd.getColumnTypeName(ci+1));
        colTypes[ci] = rsmd.getColumnTypeName(ci+1);
      }
      rSet.close();
      stmt.close();
    }
    // 値部分を構築します。
    int vi = 0;
    Enumeration cVal = columnValues.elements();
    boolean comma = false;
    while (cVal.hasMoreElements())
    {
      if (comma)
        values += ", ";
      comma = true;
      if (colTypes[vi].equals("DATE"))
        values += ("TO_DATE(SUBSTR(");
      values += ("'" + cVal.nextElement() + "'");
      if (colTypes[vi].equals("DATE"))
        values += ("", 1, 19), 'YYYY-MM-DD HH24:MI:SS')");
      vi++;
    }
    // 実行します。
    // System.out.println("Stmt:" + "INSERT INTO " + tInDoc[level].table + " (" +
columns + ") VALUES (" + values + ")");
    sqlStmt.addElement("INSERT INTO " + tInDoc[level].table + " (" + columns +
") VALUES (" + values + ")");
    nbRowProcessed++;
  }
  catch (Exception execE)
  {
    //      System.err.println("Executing " + execE);
    throw execE;
  }
  values = "";
}
// System.out.println("End of Loop");
return nbRowProcessed;

```

```
}

public static XMLDocument parseDocument(String documentStream) throws Exception
{
    XMLDocument returnXML = null;
    try
    {
        parser.parse(new InputSource(new
ByteArrayInputStream(documentStream.getBytes())));
        returnXML = parser.getDocument();
    }
    catch (SAXException saxE)
    {
        // System.err.println("Parsing XML\n" + "SAX Exception:\n" + saxE.toString());
        // System.err.println("For:\n" + documentStream + "\nParse failed SAX : " +
saxE);
        throw saxE;
    }
    catch (IOException e)
    {
        // System.err.println("Parsing XML\n" + "Exception:\n" + e.toString());
        // System.err.println("Parse failed : " + e);
        throw e;
    }
    return returnXML;
}
// ファイル名から URL を作成します。
private static URL createURL(String fileName) throws Exception
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex) // 有効な URL ではありません。ファイルである可能性があ
ります。
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            // この大量のコードは、getAbsolutePath の戻り値に一貫性がないために、
            // Windows プラットフォームで URL を有効にするために必要です。

```

```
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.err.println("Cannot create url for: " + fileName);
    throw e;    // これもファイルではありません。
}
}
return url;
}

private static String readURL(URL url) throws Exception
{
    URLConnection newURLConn;
    BufferedInputStream newBuff;
    int nBytes;
    byte aByte[];
    String resultBuff = "";

    aByte = new byte[2];
    try
    {
        // System.out.println("Calling " + url.toString());
        try
        {
            newURLConn = url.openConnection();
            newBuff = new BufferedInputStream(newURLConn.getInputStream());
            resultBuff = "";
            while ((nBytes = newBuff.read(aByte, 0, 1)) != -1)
                resultBuff = resultBuff + (char)aByte[0];
        }
        catch (IOException e)
        {
            // System.err.println("Opening locator\n" + e.toString());
            // System.err.println(url.toString() + "\n : newURLConn failed :\n" + e);
        }
    }
}
```

```
        throw e;
    }
}
catch (Exception e)
{
    // System.err.println("Read URL\n" + e.toString());
    throw e;
}
return resultBuff;
}

private void executeStatement(String strStmt) throws SQLException, Exception
{
    if (conn == null)
        throw new Exception("Connection is null");
    try
    {
        Statement stmt = conn.createStatement();
        stmt.execute(strStmt);
        stmt.close();
    }
    catch (SQLException e)
    {
        System.err.println("Failed to execute statement\n" + strStmt);
        throw e;
    }
}

private void getConnected() throws Exception
{
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(connURL, username, password);
    }
    catch (Exception e)
    {
        // System.err.println(e);
        throw e;
    }
}

public Connection getConnection()
{
    return conn;
}
}
```

Java の例 12: XMLGen.java

```

package B2BDemo.XMLUtil;

import java.sql.*;
/**
 * このクラスは、placeOrder.xsql の XSQL Servlet によってコールされたアクション・
 * ハンドラが使用します。このクラスは、ブローカに送信される、元の XML ドキュメント
 * を生成します。
 *
 * @see B2BMessage
 * @see XMLProducer
 * @see RetailActionHandler
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class XMLGen extends Object
{
    static Connection conn = null;
    // デフォルトの接続パラメータ
    static String appURL      = "jdbc:oracle:thin:@localhost:1521:ORCL";
    static String appUser     = "retailer";
    static String appPassword = "retailer";

    static String XMLStmt =
        "SELECT O.ID as \"Id\", \"          +
        \"      O.ORDERDATE as \"Orderdate\", \" +
        \"      O.CONTACTNAME as \"Contactname\", \" +
        \"      O.TRACKINGNO as \"Trackingno\", \" +
        \"      O.STATUS as \"Status\", \" +
        \"      O.CUSTOMER_ID as \"CustomerId\", \" +
        \"      CURSOR (SELECT L.ID as \"Id\", \" +
        \"                  L.QUANTITY as \"Quantity\", \" +
        \"                  L.ITEM_ID as \"ItemId\", \" +
        \"                  L.ORD_ID as \"OrdId\", \" +
        \"                  L.DISCOUNT as \"Discount\" \" +
        \"                  FROM LINE_ITEM L \" +
        \"                  WHERE L.ORD_ID = O.ID) as \"LineItemView\" \" +
        \"FROM ORD O \" +
        \"WHERE O.ID = ?\";

    public static String returnDocument (Connection c, String ordId)
    {
        String XMLDoc = "";
        try
        {
            if (c != null)
                conn = c;

```

```
        if (conn == null)
            _getConnected(appURL, appUser, appPassword);
        XMLProducer xmlp = null;
        xmlp = new XMLProducer(conn); // XML プロシージャ
        xmlp.setRowset("Results");
        xmlp.setRow("OrdView");
        PreparedStatement stmt = conn.prepareStatement(XMLStmt);
        stmt.setString(1, ordId);
        ResultSet rSet = stmt.executeQuery();

        XMLDoc = xmlp.getXMLString(rSet, "Y");
        rSet.close();
        stmt.close();
        if (c == null)
        {
            conn.close();
            conn = null;
        }
    }
    catch (SQLException e)
    {}
    return XMLDoc;
}

private static void _getConnected(String connURL,
                                   String userName,
                                   String password)
{
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(connURL, userName, password);
    }
    catch (Exception e)
    {
        System.err.println(e);
        System.exit(1);
    }
}

public static void main (String[] args) // テスト用のみです。
{
    System.out.println(returnDocument(null, "28004"));
}
}
```


Java の例 13: XMLUtil.java

```
package B2BDemo.XMLUtil;
/**
 * AQ スキーマ内の Stylesheet 表のレコードを一致させます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class XslRecord
{
    public String FROM;
    public String TO;
    public String TASK;
    public String XSL;

    public XslRecord(String FROM,
                     String TO,
                     String TASK,
                     String XSL)
    {
        this.FROM = FROM;
        this.TO = TO;
        this.TASK = TASK;
        this.XSL = XSL;
    }

    public boolean equals(XslRecord x)
    {
        if (this.FROM.equals(x.FROM) &&
            this.XSL.equals(x.XSL) &&
            this.TASK.equals(x.TASK) &&
            this.TO.equals(x.TO))
            return true;
        else
            return false;
    }
}
```

Java の例 14: XSLTWrapper.java

```
package B2BDemo.XMLUtil;
/**
 * いくつかの解析機能をラップします。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.net.*;
import java.io.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

/**
 * このクラスは、Oracle XML Parser for Java V2 とともに提供される XSL Transformer
 * 用のラッパーです。
 *
 * このクラスは、文字列、ファイルまたは URL から XSL 変換を処理します。
 *
 * @author Olivier Le Diouris. Partner Services. Oracle Corp.
 * @version 1.0
 */
public class XSLTWrapper
{
    DOMParser parser;

    String xml    = "";
    String xsl    = "";
    String result = "";

    private static boolean _debug = false;

    public XSLTWrapper()
    {
        parser = new DOMParser();
    }

    public void process() throws Exception
    {
        if (xml.length() == 0)
            throw new Exception("XML Document is empty");
        if (xsl.length() == 0)
```

```
        throw new Exception("XSL Document is empty");
        result = processTransformation(xml, xsl);
    }

    public void putXml(String xml) throws Exception
    {
        if (_debug) System.out.println("Recieved XML : \n" + xml);
        this.xml = xml;
    }
    public void putXsl(String xsl) throws Exception
    {
        this.xsl = xsl;
        if (_debug) System.out.println("Recieved XSL: \n" + xsl);
    }
    public String getXml() throws Exception
    {
        return xml;
    }
    public String getXsl() throws Exception
    {
        return xsl;
    }
    public String getProcessResult() throws Exception
    {
        return result;
    }
}

// 文字列をXMLドキュメントに変換します。
public XMLDocument parseDocument(String documentStream) throws Exception
{
    XMLDocument returnXML = null;
    try
    {
        parser.parse(new InputSource(new
        ByteArrayInputStream(documentStream.getBytes())));
        returnXML = parser.getDocument();
    }
    catch (SAXException saxE)
    {
        if (_debug) System.err.println("For:\n" + documentStream + "\nParse failed SAX
: " + saxE);
        throw new Exception("Parsing XML\n" + "SAX Exception:\n" + saxE.toString());
    }
    catch (IOException e)
```

```
{
    if (_debug) System.err.println("Parse failed : " + e);
    throw new Exception("Parsing XML\n" + "IOException:\n" + e.toString());
}
return returnXML;
}

private XMLDocument processXML(XMLDocument xml,
                                XMLDocument xslDoc) throws Exception
{
    XMLDocument out = null;
    URL xslURL = null;

    try
    {
        parser.setPreserveWhitespace(true);
        parser.setValidationMode(false);          // 検証します (オプション)。
        // スタイルシートをインスタンス化します。
        XSLStylesheet xsl = new XSLStylesheet(xslDoc, xslURL);
        XSLProcessor processor = new XSLProcessor();

        // 発生する可能性がある警告を表示します。
        processor.showWarnings(true);
        processor.setErrorStream(System.err);

        // XSL を処理します。
        DocumentFragment result = processor.processXSL(xsl, xml);

        // 結果を保持する出力ドキュメントを作成します。
        out = new XMLDocument();
        /*
        // 出力ドキュメント用のダミー・ドキュメント要素を作成します。
        Element root = out.createElement("root");
        out.appendChild(root);
        // ダミー・ドキュメント要素に変換ツリーを追加します。
        root.appendChild(result);
        */
        out.appendChild(result);
        // 変換済ドキュメントを出力します。
        // out.print(System.out);
    }
    catch (Exception e)
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        PrintWriter pw = new PrintWriter(baos);
```

```

        e.printStackTrace(pw);
        e.printStackTrace();
        throw new Exception("ProcessXML\n" + baos.toString());
    }
    return(out);
}

/**
 * 入力としての XML 文字列および XSL 文字列です。
 * 入力文字列には次が含まれる場合があります。
 *      URL の名前
 *      ファイルの名前 (ローカル・ファイル・システム上)
 *      ドキュメント自体
 * 出力としての XML 文字列です。
 */
public String processTransformation(String xmlStream,
                                   String xslStream) throws Exception
{
    String xmlContent = "";
    String xslContent = "";

    try
    { xmlContent = readURL(createURL(xmlStream)); }
    catch (Exception e)
    { xmlContent = xmlStream; }

    try
    { xslContent = readURL(createURL(xslStream)); }
    catch (Exception e)
    { xslContent = xslStream; }

    if (_debug) System.out.println("xmlStream = " + xmlContent);
    if (_debug) System.out.println("xslStream = " + xslContent);

    XMLDocument xml = parseDocument(xmlContent);
    XMLDocument xsl = parseDocument(xslContent);

    XMLDocument out = processXML(xml, xsl);
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    try
    { out.print(baos); }
    catch (IOException ioE)
    {
        if (_debug) System.err.println("Exception:" + ioE);
        throw new Exception("XML Processing throws IOException\n" + ioE.toString());
    }
}

```

```
    }
    return (baos.toString());
}

// ファイル名から URL を作成します。
private static URL createURL(String fileName) throws Exception
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex) // これは有効な URL ではありません。ファイルである可能性
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            // この大量のコードは、getAbsolutePath の戻り値に一貫性がないために、
            // Windows プラットフォームで URL を有効にするために必要です。
            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)
        {
            if (_debug) System.err.println("Cannot create url for: " + fileName);
            throw e; // ファイルではありません。
        }
    }
    return url;
}

private static String readURL(URL url) throws Exception
{

```

```
URLConnection newURLConn;
BufferedInputStream newBuff;
int nBytes;
byte aByte[];
String resultBuff = "";

aByte = new byte[2];
try
{
// System.out.println("Calling " + url.toString());
try
{
    newURLConn = url.openConnection();
    newBuff = new BufferedInputStream(newURLConn.getInputStream());
    resultBuff = "";
    while (( nBytes = newBuff.read(aByte, 0, 1)) != -1)
        resultBuff = resultBuff + (char)aByte[0];
}
catch (IOException e)
{
// System.err.println("Opening locator\n" + e.toString());
// System.err.println(url.toString() + "\n : newURLConn failed :\n" + e);
throw e;
}
}
catch (Exception e)
{
// System.err.println("Read URL\n" + e.toString());
throw e;
}
return resultBuff;
}
}
```

B2B XML アプリケーションで使用する他のスクリプト

XML の例 1: XSQL 構成 - XSQLConfig.xml

```
<?xml version="1.0" ?>
<!--
| $Author: smuench $
| $Date: 2000/03/14 10:36:42 $
| $Source: C:\cvsroot\xsql/src/XSQLConfig.xml,v $
| $Revision: 1.11 $
+-->
<XSQLConfig>

  <!--
  |
  | このセクションでは、XSQL Servlet に固有な構成設定を
  | 定義します。
  |
  +-->
  <servlet>

    <!--
    |
    | バッファされた出力ストリームのサイズ（バイト単位）を設定します。
    | サブレット・エンジンが、サブレット出力ストリームへの I/O を
    | すでにバッファしている場合、追加のバッファを防ぐために 0（ゼロ）
    | を設定できます。
    |
    |      <output-buffer-size>10000</output-buffer-size>
    |
    +-->
    <output-buffer-size>0</output-buffer-size>

    <!--
    |
    | 次に示すように、<media-type> 要素を追加して、XSQL Servlet が
    | Media/Content-type の「charset=XXX」部分を送信することを
    | 防止できます。
    |
    | たとえば、image/svg ドキュメント用のキャラクタ・セットを送信すると、
    | 既存の SVG プラグインが混乱するようです。
    |
    |      <suppress-mime-charset>
    |        <media-type>image/svg</media-type>
    |      </suppress-mime-charset>
```



```

+-->

<suppress-mime-charset>
  <media-type>image/svg</media-type>
</suppress-mime-charset>

</servlet>

<!--
|   このセクションでは、XSQL Page Processor の構成設定を定義します。
|
+-->
<processor>

  <!--
  |   接続定義（次の <connectiondefs> を参照）は、
  |   XSQL Page Processor の初期化時にキャッシュされます。
  |
  |   「YES」に設定すると、キャッシュされた接続リストに名前がない
  |   接続が要求された場合に、接続定義を再ロードするために
  |   XSQLConfig.xml が再読み込みされます。「YES」設定は、開発時に、
  |   サブレット実行中に新しい <connection> 定義をファイルに追加する場合に
  |   有効です。「NO」に設定すると、接続名がメモリ内のキャッシュにないときの
  |   接続定義ファイルの再ロードを防ぐことができます。
  |
  +-->

  <reload-connections-on-error>yes</reload-connections-on-error>

  <!--
  |   データベースへの SQL 問合せから情報を取り出すために
  |   行フェッチ・サイズのデフォルト値を設定します。
  |   Oracle JDBC Driver を使用する場合にのみ有効です。
  |   使用していない場合は、設定は無視されます。
  |   異なる層で実行しているサブレット・エンジンから
  |   データベースへのネットワークのラウンドトリップを削減
  |   するために使用できます。
  |
  |   <default-fetch-size>50</default-fetch-size>
  |
  +-->

  <default-fetch-size>50</default-fetch-size>

```

```
<!--
|
|   XSQL ページの XSQL LRU キャッシュ値を設定します。
|   このスクリプトは、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。
|   この数を超えてキャッシュしようとする、最低使用頻度シートが
|   キャッシュから消去されます。
|
|   <page-cache-size>25</page-cache-size>
|
+-->

<page-cache-size>25</page-cache-size>

<!--
|
|   XSL スタイルシートの XSQL LRU キャッシュ値を設定します。
|   このスクリプトは、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。
|   この数を超えてキャッシュしようとする、最低使用頻度シートが
|   キャッシュから消去されます。
|
|   <stylesheet-cache-size>25</stylesheet-cache-size>
|
+-->

<stylesheet-cache-size>25</stylesheet-cache-size>

<!--
|
|   スタイルシート・プールを制御するパラメータを設定します。
|
|   キャッシュされた各スタイルシートは、実際にはスタイルシート・
|   インスタンスがキャッシュされたプールです。これらの値は、
|   プール内のスタイルシート・インスタンスの初期数を制御します。
|   プール内のインスタンスが少なくなると、この数を追加 / 増分して
|   プールを大きくする必要があります。また、スタイルシート・
|   インスタンスの数を初期値に戻すためにインスタンスを削除するまでの、
|   アクティビティなしの経過時間（秒数）も制御します。
|
|   <stylesheet-pool>
|       <initial>1</initial>
|       <increment>1</increment>
|       <timeout-seconds>60</timeout-seconds>
|   </stylesheet-pool>
|
+-->
```

```

<stylesheet-pool>
  <initial>1</initial>
  <increment>1</increment>
  <timeout-seconds>60</timeout-seconds>
</stylesheet-pool>

<!--
| データベース接続プールを制御するパラメータを設定します。
|
| このパラメータを使用すると、定義されている名前付き接続は、
| それぞれ接続インスタンスのプールを持つことができ、要求を
| 共有できます。これらの値は、プール内のスタイルシート・インスタンスの初期数
| を制御します。プール内のインスタンスが少なくなると、この数を追加 / 増分して
| プールを大きくする必要があります。また、スタイルシート・
| インスタンスの数を初期値に戻すためにインスタンスを削除するまでの、
| アクティビティなしの経過時間（秒数）も制御します。
|
| 「dump-allowed」要素に「YES」が設定されている場合、
| 接続プールの現在の状態をダンプするブラウザ・ベースの
| 状態レポートが使用可能になります。
|
| <connection-pool>
|   <initial>2</initial>
|   <increment>1</increment>
|   <timeout-seconds>60</timeout-seconds>
|   <dump-allowed>no</dump-allowed>
| </connection-pool>
+-->

<connection-pool>
  <initial>2</initial>
  <increment>1</increment>
  <timeout-seconds>60</timeout-seconds>
  <dump-allowed>no</dump-allowed>
</connection-pool>

```

```
<!--
|
| タイミング情報を含めます (1000 分の 1 秒単位)。
|
| <timing-info>
|   <page>yes</page>
|   <action>yes</action>
| </timing-info>
|
+-->

<timing-info>
  <page>no</page>
  <action>no</action>
</timing-info>

</processor>

<!--
|
| このセクションでは、<xsql:include-xml> アクションが使用する
| HTTP プロキシ・サーバーの名前およびポートを定義します。
| <xsql:include-xml> を使用して、ファイアウォール外の URL にある
| XML を含める場合は、次のセクションのコメント化を解除し、proxyhost
| および proxyport を必要に応じて変更します。
|
| <http>
|   <proxyhost>your-proxy-server.yourcompany.com</proxyhost>
|   <proxyport>80</proxyport>
| </http>
|
| コメントのままにしておくと、XSQL ページ・プロセッサはプロキシ・サーバーを
| 使用しません。
|
+-->

<!--

<http>
  <proxyhost>your-proxy-server.yourcompany.com</proxyhost>
  <proxyport>80</proxyport>
</http>

-->
```

```

<!--
| このセクションでは、便利なニックネームを1つまたは複数の
| データベース接続に定義します。<connectiondefs> 要素内に
| いくつでも <connection> 要素を設定できます。XSQL ページは、
| そのページのドキュメント要素に関する connection 属性にある
| 名前によってこれらの接続を参照します。
+-->

<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="xmlbook">
    <username>xmlbook</username>
    <password>xmlbook</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>
  <connection name="retail">
    <username>retailer</username>
    <password>retailer</password>
    <dburl>jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
  </connection>

</connectiondefs>

<!--
| この接続は、事前定義要素の名前およびハンドラ・クラスを
| ユーザー定義 XSQL ページ・アクション用に登録します。
|
| このセクションは次のように記述されます。
|

```

```

| <actiondefs>
|   <action>
|     <elementname>myAction</elementname>
|     <handlerclass>mypackage.MyActionHandler</handlerclass>
|   </action>
|   :
| </actiondefs>
|
| アクション・ハンドラ・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。
|
| ユーザー定義アクションは、ここで一度登録すると、組み込み XSQL アクションと同様に
| 使用できます。たとえば、ページ内に <xsql:myAction> 要素を含めることができます。
|
+-->
<actiondefs>
  <action>
    <elementname>param</elementname>

    <handlerclass>oracle.xml.xsql.actions.ExampleGetParameterHandler</handlerclass>
  </action>
  <action>
    <elementname>current-date</elementname>

    <handlerclass>oracle.xml.xsql.actions.ExampleCurrentDBDateHandler</handlerclass>
  </action>
</actiondefs>

</XSQLConfig>

```

Java の例 15: メッセージ・ヘッダー・スクリプト - MessageHeaders.java

メッセージ・ヘッダー・スクリプトは、次のとおりです。

```

package B2BDemo;
/**
 * メッセージに使用されるヘッダーについて説明します。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class MessageHeaders extends Object
{
    public static String APP_A          = "RETAIL";
    public static String APP_B          = "SUPPLY";
}

```

```

    public static String BROKER          = "BROKER";
    public static String EXIT            = "EXIT";
    public static String NEW_ORDER       = "NEW ORDER";
    public static String UPDATE_ORDER    = "UPDATE ORDER";
}

```

Java の例 16: Message Broker による使用定数の保持 - AppCste.java

```

package B2BDemo;
/**
 * Message Broker が使用する制約を保持します。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class AppCste extends Object
{
    public final static String AQDBUrl =
        "jdbc:oracle:thin:@atp-1.us.oracle.com:1521:ORCL";
    public final static String AQuser  = "aqMessBrok";
    public final static String AQpswd  = "aqMessBrok";
}

```

小売業者のスク립ト

小売業者は、次のスク립トを使用します。

- [Java の例 17: 小売業者によるサプライヤからの更新状態待機 - UpdateMaster.java](#)

Java の例 17: 小売業者によるサプライヤからの更新状態待機 - UpdateMaster.java

```

package B2BDemo.Retailer;
/**
 *
 * このクラスは、サプライヤが出荷後に行う状態の更新を、小売業者側で待機する
 * コンポーネントを実装します。受信されたドキュメントは解析され、その内容を
 * 使用してデータベースが容易に更新されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */

```

```
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

public class UpdateMaster extends Object
{
    private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

    private static boolean stepByStep = false;
    private static boolean verbose    = false;
    private static Integer pauseTime  = null;

    AQReader aqr;

    private static final String userName = "retailer";
    private static final String password = "retailer";
    private static String url          = "jdbc:oracle:thin:@localhost:1521:ORCL"; // これは
デフォルト値です。
    private static Connection conn = null;

    String currOrdId = "";

    DOMParser parser = new DOMParser();
    /**
     * コンストラクタ
     */
}
```



```

public UpdateMaster()
{
    XMLFrame frame = new XMLFrame("Retailer");
    /**
    try
    {
        OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
        UIManager.setLookAndFeel(new OracleLookAndFeel());
        SwingUtilities.updateComponentTreeUI(frame);
        frame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
        System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    // ウィンドウを中央に配置します。
    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;
    }
    */
    frameSize.width = screenSize.width / 3;

    // frame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height -
    frameSize.height)/2);
    frame.setLocation(0, (screenSize.height - frameSize.height)/2);
    // frame.addWindowListener(new WindowAdapter() { public void
    windowClosing(WindowEvent e) { System.exit(0); } });
    frame.setVisible(true);

    // AQ リーダーを初期化します。
    aqr = new AQReader(AppCste.AQuser,
                      AppCste.AQpswd,
                      AppCste.AQDBUrl,
                      "AppFour_QTab",
                      "AppFourMsgQueue");

```

```

boolean go = true;
while (go)
{
    String ordIdValue = "";
    B2BMessage sm = aqr.readQ();
    if (verbose)
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                            "\nTo > " + sm.getTo() +
                            "\nType > " + sm.getType() +
                            "\nContent >\n" + sm.getContent());
    else
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                            "\nTo > " + sm.getTo() +
                            "\nType > " + sm.getType());
    String xmlDoc = sm.getContent();
    if (xmlDoc != null && xmlDoc.length() > 0)
    {
        try { frame.setXMLDocument(sm.getContent()); }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    if (stepByStep)
    {
        if (pauseTime != null)
        {
            System.out.println("Waiting for " + pauseTime.longValue() + "
milliseconds");
            try { Thread.sleep(pauseTime.longValue()); } catch (InterruptedException
e) {}
        }
        else
            try { String s = _userInput("[Hit return to continue]"); } catch
(Exception e) {}
    }

    if (sm.getType().equals(MessageHeaders.EXIT))
        go = false;
    else
    {
        System.out.println("Updating");
        try
        {

```

```

        parser.parse(new InputSource(new
ByteArrayInputStream(sm.getContent().getBytes())));
        XMLDocument xml = parser.getDocument();
        XMLElement elmt = (XMLElement)xml.getDocumentElement();
        NodeList nl = elmt.getElementsByTagName("SHIP"); // ORD ID
        for (int i=0; i<nl.getLength(); i++)
        {
            XMLElement ordId = (XMLElement)nl.item(i);
            XMLNode theText = (XMLNode)ordId.getFirstChild();
            currOrdId = theText.getNodeValue();
            System.out.println("Gonna update " + currOrdId);
            try
            {
                if (conn == null)
                    getConnected(url, userName, password);
                String strStmt = "update ORD set STATUS = 'Shipped' where ID = ?";
                PreparedStatement pStmt = conn.prepareStatement(strStmt);
                pStmt.setString(1, currOrdId);
                pStmt.execute();
                conn.commit();
                pStmt.close();
                System.out.println("Done !");
            }
            catch (SQLException e)
            {
                System.out.println("Pb updating the ORD\n" + e.toString());
            }
        }
    }
    catch (SAXParseException e)
    {
        System.out.println(e.getMessage());
    }
    catch (SAXException e)
    {
        System.out.println(e.getMessage());
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
}
}
frame.setVisible(false);
System.exit(0);

```

```

    }

    private static void getConnected(String connURL,
                                     String userName,
                                     String password)
    {
        try
        {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            conn = DriverManager.getConnection(connURL, userName, password);
        }
        catch (Exception e)
        {
            System.err.println(e);
            System.exit(1);
        }
    }

    private String _userInput(String prompt) throws Exception
    {
        String retString;
        System.out.print(prompt);
        try { retString = stdin.readLine(); }
        catch (Exception e)
        {
            System.out.println(e);
            throw(e);
        }
        return retString;
    }

    private static void setRunPrm(String[] prm)
    {
        for (int i=0; i<prm.length; i++)
        {
            if (prm[i].toLowerCase().startsWith("-verbose"))
                verbose = isolatePrmValue(prm[i], "-verbose");
            else if (prm[i].toLowerCase().startsWith("-help"))
            {
                System.out.println("Usage is:");
                System.out.println("\tjava B2BDemo.Retailer.MessageBroker");
                System.out.println("\tparameters can be -dbURL -verbose, -step, -help");
                System.out.println("\tdbURL contains a string like
jdbc:oracle:thin:@localhost:1521:ORCL");
                System.out.println("\tparameters values can be (except for -help):");
            }
        }
    }

```

```
System.out.println("\t\tnone - equivalent to 'y'");
System.out.println("\t\tty");
System.out.println("\t\ttrue - equivalent to 'y'");
System.out.println("\t\ttn");
System.out.println("\t\tfalse - equivalent to 'n'");
System.out.println("\t\tstep can take a value in milliseconds");
System.exit(0);
}
else if (prm[i].toLowerCase().startsWith("-step"))
{
    String s = getPrmValue(prm[i], "-step");
    try
    {
        pauseTime = new Integer(s);
        System.out.println("Timeout " + pauseTime);
        stepByStep = true;
    }
    catch (NumberFormatException nfe)
    {
        pauseTime = null;
        if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
            stepByStep = true;
        else
            stepByStep = false;
    }
}
else if (prm[i].toLowerCase().startsWith("-dburl"))
{
    url = getPrmValue(prm[i], "-dbURL");
}
else
    System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
}
}

private static boolean isolatePrmValue(String s, String p)
{
    boolean ret = true;
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
        if (s.indexOf("=") > -1)
        {
            String val = s.substring(s.indexOf("=") + 1);
            if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
                ret = true;
        }
    }
}
```

```
        else if (val.toUpperCase().equals("N") || val.toUpperCase().equals("FALSE"))
            ret = false;
        else
        {
            System.err.println("Unrecognized value for " + p + ", set to Y");
            ret = true;
        }
    }
}
return ret;
}

private static String getPrmValue(String s, String p)
{
    String ret = "";
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
        if (s.indexOf("=") > -1)
        {
            ret = s.substring(s.indexOf("=") + 1);
        }
    }
    return ret;
}

/**
 * メイン
 * @パラメータ引数
 */
public static void main(String[] args)
{
    if (args.length > 0)
        setRunPrm(args);
    UpdateMaster updateMaster = new UpdateMaster();
}
}
```

AQ Broker Transformer およびアドバンスト・キューイングのスクリプト

AQ Broker Transformer は、次のスクリプトを使用します。

- [Java の例 18: 1 つの AQ スレッド上での AQ Broker のリスニング - BrokerThread.java](#)
- [Java の例 19: MessageBroker.java](#)
- [Java の例 20: AQReader.java](#)
- [Java の例 21: AQWriter.java](#)
- [Java の例 22: B2BMessage.java](#)
- [Java の例 23: ReadStructAQ.java](#)
- [Java の例 24: StopAllQueues.java](#)
- [Java の例 25: WriteStructAQ.java](#)

Java の例 18: 1 つの AQ スレッド上での AQ Broker のリスニング - BrokerThread.java

```
package B2BDemo.Broker;

import java.sql.*;
import oracle.AQ.*;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;

import B2BDemo.AQUtil.*;
import B2BDemo.XMLUtil.*;
import B2BDemo.*;

/**
 * このクラスは、1 つの AQ 上でリスニングするスレッドを実装します。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class BrokerThread extends Thread
{
    private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

    private static boolean stepByStep = false;
    private static boolean verbose    = false;
```

```
AQReader aqReader;
AQWriter aqWriter;
String threadName;
XSLTWrapper wrapper;
Connection conn;
Integer pause;
XMLFrame frame;
/**
 * コンストラクタ
 */
public BrokerThread(String name,
                    AQReader aqr,
                    AQWriter aqw,
                    XSLTWrapper wrap,
                    Connection c,
                    boolean v,
                    boolean s,
                    Integer p,
                    XMLFrame f)
{
    this.aqReader = aqr;
    this.aqWriter = aqw;
    this.threadName = name;
    this.wrapper = wrap;
    this.conn = c;

    this.verbose = v;
    this.stepByStep = s;
    this.pause = p;
    this.frame = f;
}

public void run()
{
    boolean go = true;
    while (go)
    {
        B2BMessage sm = this.aqReader.readQ();
        if (verbose)
            System.out.println(this.threadName + " Recieved\nFrom > " + sm.getFrom() +
                               "\nTo > " + sm.getTo() +
                               "\nType > " + sm.getType() +
                               "\nContent >\n" + sm.getContent());
        else
            System.out.println(this.threadName + " Recieved\nFrom > " + sm.getFrom()
+

```



```
        "\nTo  > " + sm.getTo() +
        "\nType > " + sm.getType());
String xmlDoc = sm.getContent();
if (xmlDoc != null && xmlDoc.length() > 0)
{
    try { this.frame.setXMLDocument(sm.getContent()); }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
if (stepByStep)
{
    if (pause != null)
    {
        System.out.println("Waiting for " + pause.longValue() + "
                               milliseconds");
        try { Thread.sleep(pause.longValue()); } catch (InterruptedException
                                                         e) {}
    }
    else
        try { String s = _userInput("[Hit return to continue]"); } catch
            (Exception e) {}
}
if (sm.getType().length() >= MessageHeaders.EXIT.length() &&
    sm.getType().equals(MessageHeaders.EXIT))
    go = false;
else
{
    // 変換します。
    String processedXMLDoc = "";
    String xslDoc = getXSL(sm.getFrom(),
                           sm.getTo(),
                           sm.getType());

    if (verbose)
        System.out.println("Read:\n" + xslDoc);
    try
    {
        processedXMLDoc = wrapper.processTransformation(sm.getContent(),
                                                         xslDoc /*defaultStyleSheet*/);

        if (verbose)
            System.out.println("\nResult :\n" + processedXMLDoc);
        System.out.println("Transformation done.");
    }
    catch (Exception e)
    {
        System.err.println("Oops...\n");
    }
}
```

```
        e.printStackTrace();
    }
    if (stepByStep)
    {
        if (pause != null)
        {
            System.out.println("Waiting for " + pause.longValue() + "
                                milliseconds");
            try { Thread.sleep(pause.longValue()); } catch (InterruptedException
                                                                e) {}
        }
        else
            try { String s = _userInput("[Hit return to continue]"); } catch
                (Exception e) {}
    }

    // 新しいドキュメントを接続先に送信します。
    this.aqWriter.writeQ(new B2BMessage(sm.getFrom(),
                                        sm.getTo(),
                                        sm.getType(),
                                        processedXMLDoc));

    this.aqWriter.flushQ();
}
}
if (frame.isVisible())
    frame.setVisible(false);
System.exit(0);
}

private String getXSL(String from,
                      String to,
                      String task)
{
    if (verbose)
        System.out.println("Processing From " + from + " to " + to + " for " + task);
    String xsl = "";
    String stmt = "SELECT XSL FROM STYLESHEETS WHERE APPFROM = ? AND APPTO = ? AND
OP = ?";

    try
    {
        PreparedStatement pStmt = conn.prepareStatement(stmt);
        pStmt.setString(1, from);
        pStmt.setString(2, to);
        pStmt.setString(3, task);
        ResultSet rSet = pStmt.executeQuery();
    }
}
```

```
        while (rSet.next())
        {
            xsl = _dumpClob(conn, ((OracleResultSet)rSet).getCLOB(1));
            rSet.close();
            pstmt.close();
        }
    } catch (SQLException e)
    { }
    catch (Exception e)
    { }
    return xsl;
}

static String _dumpClob (Connection conn, CLOB clob)
    throws Exception
{
    String returnStr = "";

    OracleCallableStatement cStmt1 =
        (OracleCallableStatement)
        conn.prepareCall ("begin ? := dbms_lob.getLength (?); end;");
    OracleCallableStatement cStmt2 =
        (OracleCallableStatement)
        conn.prepareCall ("begin dbms_lob.read (?, ?, ?, ?); end;");

    cStmt1.registerOutParameter (1, Types.NUMERIC);
    cStmt1.setClob (2, clob);
    cStmt1.execute ();

    long length = cStmt1.getLong (1);
    long i = 0;
    int chunk = 100;
    if (verbose)
        System.out.println("Length to read from DB : " + length);

    while (i < length)
    {
        cStmt2.setClob (1, clob);
        cStmt2.setLong (2, chunk);
        cStmt2.registerOutParameter (2, Types.NUMERIC);
        cStmt2.setLong (3, i + 1);
        cStmt2.registerOutParameter (4, Types.VARCHAR);
        cStmt2.execute ();

        long readThisTime = cStmt2.getLong (2);
        String stringThisTime = cStmt2.getString (4);
```

```
//      System.out.print ("Read " + read_this_time + " chars: ");
      returnStr += stringThisTime;
      i += readThisTime;
    }

    cStmt1.close ();
    cStmt2.close ();

    return returnStr;
  }

  private String _userInput(String prompt) throws Exception
  {
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
      System.out.println(e);
      throw(e);
    }
    return retString;
  }
}
```

Java の例 19: MessageBroker.java

```
package B2BDemo.Broker;
/**
 * AQ Broker Transformer を実装します。
 * この Message Broker は、Oracle8i Advanced Queueing
 * が提供する 4 つのメッセージ・キューを使用します。
 * AQ Broker は、BrokerThread に記述されているスレッドを使用します。
 * 各スレッドは、1 つのキュー上で待機し、他のキューに書き込みます。
 * Message Broker は、このデモでは次の 2 つのスレッドを使用します。
 *   小売業者からサプライヤへのスレッド 1 つ
 *   サプライヤから小売業者へのスレッド 1 つ
 * 2 つのスレッドは、4 つのキューです。
 *
 * ブローカは、メッセージを受信すると次を認識します。
 *   送信元
 *   送信先
 *   目的 (操作)
```

```
* これらの3つの要素は、受信ドキュメントを、送信先アプリケーションの
* 要件を満たす発信ドキュメントに変換するために、データベースから適切な
* XSL スタイルシートをフェッチするための（AQ スキーマに属する）Stylesheet 表
* の主キーとして使用されます。
*
* @see BrokerThread
* @see B2BMessage
* @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
*/
import java.sql.*;
import oracle.AQ.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

public class MessageBroker extends Object
{
    private static boolean stepByStep = false;
    private static boolean verbose    = false;
    private static Integer pauseTime  = null;

    XSLTWrapper wrapper = null;

    // CLOB からスタイルシートを取得します。
    Connection conn = null;
    String userName = AppCste.AQuser;
    String password = AppCste.AQpswd;
    String dbUrl    = AppCste.AQDBUrl;

    public MessageBroker()
    {
        XMLFrame frame = new XMLFrame("Message Broker");
        /**
        try
        {

```

```
        OracleLookAndFeel.setColorScheme(Color.cyan);
//    OracleLookAndFeel.setColorScheme("Titanium");
    UIManager.setLookAndFeel(new OracleLookAndFeel());
    SwingUtilities.updateComponentTreeUI(frame);
    frame.setBackground(UIManager.getColor("darkIntensity"));
}
catch (Exception e)
{
    System.err.println("Exception for Oracle Look and Feel:" + e );
}
*/
// ウィンドウを中央に配置します。
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;
}
*/
frameSize.width = screenSize.width / 3;
// frame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
    frame.setLocation(frameSize.width, (screenSize.height - frameSize.height)/2);
// frame.addWindowListener(new WindowAdapter() { public void
windowClosing(WindowEvent e) { System.exit(0); } });
    frame.setVisible(true);

AQReader aqr = null;
AQWriter aqw = null;
// AQ リーダーおよびライターを初期化します。
aqw = new AQWriter(AppCste.AQuser,
                    AppCste.AQpswd,
                    AppCste.AQDBUrl,
                    "AppTwo_QTab",
                    "AppTwoMsgQueue");
aqr = new AQReader(AppCste.AQuser,
                    AppCste.AQpswd,
                    AppCste.AQDBUrl,
                    "AppOne_QTab",
```

```
        "AppOneMsgQueue" );
wrapper = new XSLTWrapper();
if (conn == null)
    _getConnected();

BrokerThread retail2supply = new BrokerThread("Retail to Supply",
        aqr,
        aqw,
        wrapper,
        conn,
        verbose,
        stepByStep,
        pauseTime,
        frame);

aqw = new AQWriter(AppCste.AQuser,
        AppCste.AQpswd,
        AppCste.AQDBUrl,
        "AppFour_QTab",
        "AppFourMsgQueue");
aqr = new AQReader(AppCste.AQuser,
        AppCste.AQpswd,
        AppCste.AQDBUrl,
        "AppThree_QTab",
        "AppThreeMsgQueue");
BrokerThread supply2retail = new BrokerThread("Supply to Retail",
        aqr,
        aqw,
        wrapper,
        conn,
        verbose,
        stepByStep,
        pauseTime,
        frame);

retail2supply.start();
supply2retail.start();

System.out.println("<ThreadsOnTheirWay/>");
}

private void _getConnected()
{
    try
    {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
```

```
        conn = DriverManager.getConnection (dbUrl, userName, password);
    }
    catch (Exception e)
    {
        System.out.println("Get connected failed : " + e);
        System.exit(1);
    }
}

private static void setRunPrm(String[] prm)
{
    for (int i=0; i<prm.length; i++)
    {
        if (prm[i].toLowerCase().startsWith("-verbose"))
            verbose = isolatePrmValue(prm[i], "-verbose");
        else if (prm[i].toLowerCase().startsWith("-help"))
        {
            System.out.println("Usage is:");
            System.out.println("\tjava Intel.iDevelop.MessageBroker");
            System.out.println("\tparameters can be -verbose, -step, -help");
            System.out.println("\tparameters values can be (except for -help):");
            System.out.println("\t\tnone - equivalent to 'y'");
            System.out.println("\t\ty");
            System.out.println("\t\ttrue - equivalent to 'y'");
            System.out.println("\t\tn");
            System.out.println("\t\tfalse - equivalent to 'n'");
            System.out.println("\t\t-step can take a value in milliseconds");
            System.exit(0);
        }
        else if (prm[i].toLowerCase().startsWith("-step"))
        {
            String s = getPrmValue(prm[i], "-step");
            try
            {
                {
                    pauseTime = new Integer(s);
                    System.out.println("Timeout " + pauseTime);
                    stepByStep = true;
                }
            }
            catch (NumberFormatException nfe)
            {
                pauseTime = null;
                if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
                    stepByStep = true;
                else
                    stepByStep = false;
            }
        }
    }
}
```



```
    }
  }
  else
    System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
}
}

private static boolean isolatePrmValue(String s, String p)
{
  boolean ret = true;
  if (s.length() > (p.length() + 1)) // +1 : "="
  {
    if (s.indexOf("=") > -1)
    {
      String val = s.substring(s.indexOf("=") + 1);
      if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
        ret = true;
      else if (val.toUpperCase().equals("N") || val.toUpperCase().equals("FALSE"))
        ret = false;
      else
      {
        System.err.println("Unrecognized value for " + p + ", set to y");
        ret = true;
      }
    }
  }
  return ret;
}

private static String getPrmValue(String s, String p)
{
  String ret = "";
  if (s.length() > (p.length() + 1)) // +1 : "="
  {
    if (s.indexOf("=") > -1)
    {
      ret = s.substring(s.indexOf("=") + 1);
    }
  }
  return ret;
}

public static void main(String args[])
{

```

```
// java B2BDemo.OrderEntry.MessageBroker -verbose[=[y|true|n|false]]
-step[=[y|true|n|false]] -help
if (args.length > 0)
    setRunPrm(args);

    new MessageBroker();
}
}
```

Java の例 20: AQReader.java

```
package B2BDemo.AQUtil;
/**
 * このクラスは、Oracle8i のアドバンスト・キューイング機能のラッパーです。
 * メッセージのデキューに使用されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

public class AQReader extends Object
{
    Connection conn = null;
    AQSession aqSess = null;

    String userName = "";
    String qTableName = "";
    String qName = "";

    AQQueueTable aqTable = null;
    AQQueue aq = null;

    public AQReader(String userName,
                    String password,
                    String url,
                    String qTable,
                    String qName)
    {
        this.userName = userName;
        this.qTableName = qTable;
        this.qName = qName;
        aqSess = createSession(userName, password, url);
    }
}
```

```
aqTable = aqSess.getQueueTable(userName, qTableName);
System.out.println("Successful getQueueTable");
// q へのハンドルです。
aq = aqSess.getQueue(userName, qName);
System.out.println("Successful getQueue");
}

public AQSession createSession(String userName,
                               String pswd,
                               String url)
{
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(url, userName, pswd);
        System.out.println("JDBC Connection opened");
        conn.setAutoCommit(false);
        // Oracle8i AQ Driver をロードします。
        Class.forName("oracle.AQ.AQOracleDriver");
        // AQ セッションを作成します。
        aqSess = AQDriverManager.createAQSession(conn);
        System.out.println("AQ Session successfully created.");
    }
    catch (Exception e)
    {
        System.out.println("Exception : " + e);
        e.printStackTrace();
    }
    return aqSess;
}

public B2BMessage readQ() throws AQException
{
    AQMessage message;
    AQRawPayload rawPayload;

    // REMOVE オプションを指定して読み込みます。
    AQDequeueOption dqOption = new AQDequeueOption();
    dqOption.setDequeueMode(AQDequeueOption.DEQUEUE_REMOVE);
    message = aq.dequeue(dqOption);

    System.out.println("Successfull dQueue");
    rawPayload = message.getRawPayload();
}
```

```
        try
        {
            conn.commit(); // REMOVE をコミットします。
        }
        catch (Exception sqle)
        {
            System.err.println(sqle.toString());
        }

        return (B2BMessage)deserializeFromByteArray(rawPayload.getBytes());
    }

    private static Object deserializeFromByteArray (byte[] b)
    {
        ByteArrayInputStream inputStream = new ByteArrayInputStream(b);
        try
        {
            ObjectInputStream ois = new ObjectInputStream(inputStream);
            return ois.readObject();
        }
        catch (Exception e)
        {
            System.err.println("deserializeFromByteArray failed :" + e);
            return null;
        }
    }
}
```

Java の例 21: AQWriter.java

```
package B2BDemo.AQUtil;

/**
 * このクラスは、Oracle8i のアドバンスト・キューイング機能のラッパーです。
 * メッセージのエンキューに使用されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

public class AQWriter extends Object
{
    Connection conn = null;
```

```
AQSession aqSess = null;

String userName  = "";
String qTableName = "";
String qName     = "";

public AQWriter(String userName,
                String password,
                String url,
                String qTable,
                String qName)
{
    this.userName = userName;
    this.qTableName = qTable;
    this.qName = qName;
    aqSess = createSession(userName, password, url);
}

public void flushQ()
{
    if (conn != null)
    {
        try { conn.commit(); } catch (SQLException e) {}
    }
}

public void closeConnection()
{
    if (conn != null)
    {
        try { conn.close(); }
        catch (SQLException e)
        { }
    }
}

public AQSession createSession(String userName,
                               String pswd,
                               String url)
{
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(url, userName, pswd);
        System.out.println("JDBC Connection opened");
    }
}
```

```
        conn.setAutoCommit(false);
        // Oracle8i AQ Driver をロードします。
        Class.forName("oracle.AQ.AQOracleDriver");
        // AQ セッションを作成します。
        aqSess = AQDriverManager.createAQSession(conn);
        System.out.println("AQ Session successfully created.");
    }
    catch (Exception e)
    {
        System.out.println("Exception : " + e);
        e.printStackTrace();
    }
    return aqSess;
}

public void writeQ(B2BMessage sm) throws AQException
{
    AQQueueTable      qTable;
    AQQueue           q;

    qTable = aqSess.getQueueTable(userName, qTableName);
    System.out.println("Successful getQueueTable");
    // q へのハンドルです。
    q = aqSess.getQueue(userName, qName);
    System.out.println("Successful getQueue");

    // Q が認識されています。書き込みます。
    AQMessage message;
    AQRawPayload rawPayload;

    message = q.createMessage();
    byte[] bArray = serializeToByteArray(sm);
    rawPayload = message.getRawPayload();
    rawPayload.setStream(bArray, bArray.length);
    AQEnqueueOption eqOption = new AQEnqueueOption();
    q.enqueue(eqOption, message);
}

private static byte[] serializeToByteArray (Object o)
{
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    try
    {
        ObjectOutputStream oos = new ObjectOutputStream(outputStream);
        oos.writeObject(o);
    }
```

```
        return outputStream.toByteArray();
    }
    catch (Exception e)
    {
        System.err.println("serialize2ByteArray failed : " + e);
        return null;
    }
}
}
```

Java の例 22: B2BMessage.java

```
package B2BDemo.AQUtil;
/**
 * このクラスは、このデモで使用するメッセージの構造を記述します。
 * 8.1.7 では変更される場合があります。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
import java.io.Serializable;

public class B2BMessage extends Object implements Serializable
{
    String from;
    String to;
    String type;
    String content;

    public B2BMessage(String f,
                      String t,
                      String typ,
                      String c)
    {
        this.from    = f;
        this.to      = t;
        this.type     = typ;
        this.content  = c;
    }

    public String getFrom()
    { return this.from; }
    public String getTo()
    { return this.to; }
    public String getType()
```

```
    { return this.type; }  
    public String getContent()  
    { return this.content; }  
  
}
```

Java の例 23: ReadStructAQ.java

```
package B2BDemo.AQUtil;  
/**  
 * テスト用の主要部分です。デモ自体には使用されません。  
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.  
 */  
import java.sql.*;  
import oracle.AQ.*;  
import java.io.*;  
  
import B2BDemo.*;  
  
public class ReadStructAQ extends Object  
{  
    public static void main(String[] args)  
    {  
        AQReader aqr = new AQReader(AppCste.AQuser,  
                                     AppCste.AQpswd,  
                                     AppCste.AQDBUrl,  
                                     "objMsgsStruct_QTab",  
                                     "structMsgQueue");  
  
        // EXIT が受信されていないときはループします。  
        boolean goLoop = true;  
        while (goLoop)  
        {  
            B2BMessage sm = aqr.readQ();  
            System.out.println("Recieved\nFrom > " + sm.getFrom() +  
                              "\nTo    > " + sm.getTo() +  
                              "\nType  > " + sm.getType() +  
                              "\nContent >\n" + sm.getContent());  
  
            if (sm.getType().equals("EXIT"))  
                goLoop = false;  
        }  
        System.out.println("<bye/>");  
    }  
}
```


Java の例 24: StopAllQueues.java

```
package B2BDemo.AQUtil;

import B2BDemo.*;

/**
 * キュー、およびキュー上で待機しているアプリケーションを停止するために、デモで使用されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class StopAllQueues extends Object
{
    /**
     * コンストラクタ
     */
    public StopAllQueues()
    {
        AQWriter aqw1 = new AQWriter(AppCste.AQuser,
                                     AppCste.AQpswd,
                                     AppCste.AQDBUrl,
                                     "AppOne_QTab",
                                     "AppOneMsgQueue");
        aqw1.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                   MessageHeaders.APP_A,
                                   MessageHeaders.EXIT,
                                   ""));
        aqw1.flushQ();
        AQWriter aqw2 = new AQWriter(AppCste.AQuser,
                                     AppCste.AQpswd,
                                     AppCste.AQDBUrl,
                                     "AppTwo_QTab",
                                     "AppTwoMsgQueue");
        aqw2.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                   MessageHeaders.APP_A,
                                   MessageHeaders.EXIT,
                                   ""));
        aqw2.flushQ();
        AQWriter aqw3 = new AQWriter(AppCste.AQuser,
                                     AppCste.AQpswd,
                                     AppCste.AQDBUrl,
                                     "AppThree_QTab",
                                     "AppThreeMsgQueue");
        aqw3.writeQ(new B2BMessage(MessageHeaders.APP_B,
                                   MessageHeaders.APP_A,
```

```
        MessageHeaders.EXIT,
        "");
    aqw3.flushQ();
    AQWriter aqw4 = new AQWriter(AppCste.AQuser,
        AppCste.AQpswd,
        AppCste.AQDBUrl,
        "AppFour_QTab",
        "AppFourMsgQueue");
    aqw4.writeQ(new B2BMessage(MessageHeaders.APP_B,
        MessageHeaders.APP_A,
        MessageHeaders.EXIT,
        ""));
    aqw4.flushQ();
}

/**
 * メイン
 * @パラメータ引数
 */
public static void main(String[] args)
{
    StopAllQueues stopAllQueues = new StopAllQueues();
}
}
```

Java の例 25: WriteStructAQ.java

```
package B2BDemo.AQUtil;
/**
 * テスト用の主要部分です。デモ自体では使用されません。
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
/**
 * テスト用の主要部分です。
 */
import java.sql.*;
import oracle.AQ.*;
import java.io.*;

import B2BDemo.*;

public class WriteStructAQ extends Object
{
```

```
private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

static AQWriter aqw = null;

public static void main(String[] args)
{
    try
    {
        aqw = new AQWriter(AppCste.AQuser,
                            AppCste.AQpswd,
                            AppCste.AQDBUrl,
                            "objMsgsStruct_QTab",
                            "structMsgQueue");

        String messSubject    = "";
        String messTxt        = "";
        String messOrigin     = "";
        String messDestination = "";
        try
        {
            messOrigin      = userInput("Message Origin      > ");
            messDestination = userInput("Message Destination > ");
            messSubject     = userInput("Message Subject    > ");
            messTxt         = userInput("Message Text      > ");
        } catch (Exception e) {}

        // キューを作成します。
        B2BMessage sm = new B2BMessage(messOrigin,
                                       messDestination,
                                       messSubject,
                                       messTxt);

        aqw.writeQ(sm);
        try { String s = userInput("Written"); }
        catch (Exception ne) {}
        aqw.closeConnection();
        try { String s = userInput("Closed !"); }
        catch (Exception ne) {}
    }
    catch (Exception e)
    {
        System.err.println("Arghh : " + e);
        e.printStackTrace();
        try { String s = userInput("..."); }
        catch (Exception ne) {}
    }
}
```

```
    }  
}  
  
private static String userInput(String prompt) throws Exception  
{  
    String retString;  
    System.out.print(prompt);  
    try { retString = stdin.readLine(); }  
    catch (Exception e)  
    {  
        System.out.println(e);  
        throw(e);  
    }  
    return retString;  
}  
}
```

サプライヤのスク립ト

サプライヤは、次のスク립トを使用します。

- [Java の例 26: SupplierFrame.java](#)
- [Java の例 27: エージェントによる小売業者からの注文受信の通知 - SupplierWatcher.java](#)

Java の例 26: SupplierFrame.java

```
package B2BDemo.Supplier;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import oracle.xml.sql.query.*;
import oracle.xml.sql.dml.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import org.xml.sax.*;

import B2BDemo.AQUtil.*;
import B2BDemo.*;
import B2BDemo.XMLUtil.*;

/**
 * このクラスは、注文の出荷を促すフレームを実装します。
 *
 * @see SupplierWatcher in the same package.
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class SupplierFrame extends JFrame
{
    private BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
    private static boolean stepByStep = false;
    private static boolean verbose    = false;
```

```
private static Integer pause      = null;
private XMLFrame frame           = null;

AQReader aqr;
XMLtoDMLv2 x2d = null;

String userName = "supplier";
String password = "supplier";
String url      = null;

String currOrdId = "";

DOMParser parser = new DOMParser();

AQWriter aqw = null;

BorderLayout borderLayout1 = new BorderLayout();
JPanel jPanel1 = new JPanel();
BorderLayout borderLayout2 = new BorderLayout();
JPanel southPanel = new JPanel();
JButton shipButton = new JButton();
JPanel centerPanel = new JPanel();
JLabel ordMessage = new JLabel();

/**
 * 新しいインスタンスを構築します。
 */
public SupplierFrame(boolean v, boolean s, Integer p, XMLFrame f, String url)
{
    super();
    this.verbose = v;
    this.stepByStep = s;
    this.pause = p;
    this.frame = f;
    this.url = url;
    try
    {
        jbInit();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

```
/**
 * このインスタンスの状態を初期化します。
 */
private void jbInit() throws Exception
{
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    shipButton.setText("Ship Order");
    shipButton.setEnabled(false);
    shipButton.addActionListener(new java.awt.event.ActionListener()
    {

        public void actionPerformed(ActionEvent e)
        {
            shipButton_actionPerformed(e);
        }
    });
    ordMessage.setText("Waiting for Orders");
    ordMessage.setFont(new Font("Dialog", 1, 20));
    jPanel1.setLayout(borderLayout2);
    this.setTitle("Supplier Watcher");
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(southPanel, BorderLayout.SOUTH);
    southPanel.add(shipButton, null);
    jPanel1.add(centerPanel, BorderLayout.CENTER);
    centerPanel.add(ordMessage, null);
}

public void enterTheLoop()
{
    // AQ リーダーを初期化します。
    aqr = new AQReader(AppCste.AQuser,
                      AppCste.AQpswd,
                      AppCste.AQDBUrl,
                      "AppTwo_QTab",
                      "AppTwoMsgQueue");
    // XSL Transformer を初期化します。
    x2d = new XMLtoDMLv2(userName,
                        password,
                        url);
    // AQ ライターを初期化します。
    aqw = new AQWriter(AppCste.AQuser,
                      AppCste.AQpswd,
                      AppCste.AQDBUrl,
                      "AppThree_QTab",
```

```

        "AppThreeMsgQueue");

boolean go = true;
while (go)
{
    String ordIdValue = "";
    String custIdValue = "";
    B2BMessage sm = aqr.readQ();
    if (verbose)
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                            "\nTo > " + sm.getTo() +
                            "\nType > " + sm.getType() +
                            "\nContent >\n" + sm.getContent());
    else
        System.out.println("Recieved\nFrom > " + sm.getFrom() +
                            "\nTo > " + sm.getTo() +
                            "\nType > " + sm.getType());
    String xmlDoc = sm.getContent();
    if (xmlDoc != null && xmlDoc.length() > 0)
    {
        try { this.frame.setXMLDocument(sm.getContent()); }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    if (stepByStep)
    {
        if (pause != null)
        {
            System.out.println("Waiting for " + pause.longValue() + " milliseconds");
            try { Thread.sleep(pause.longValue()); } catch (InterruptedException e) {}
        }
        else
            try { String s = _userInput("[Hit return to continue]"); } catch
(Exception e) {}
    }
    if (sm.getType().equals(MessageHeaders.EXIT))
        go = false;
    else
    {
        System.out.println("Inserting");
        TableInDocument d[] = new TableInDocument[2];
    }
}

```



```
d[0] = new TableInDocument("ROWSET", "ROW", "ORD");
d[1] = new TableInDocument("ITEMS", "ITEM_ROW", "LINE_ITEM");
try
{
    String XMLDoc = sm.getContent();
    x2d.insertFromXML(d, XMLDoc);
    System.out.println("Document processed.");
    // 要素を読み込む必要があります。
    parser.setValidationMode(false);
    try
    {
        parser.parse(new InputSource(new
ByteArrayInputStream(XMLDoc.getBytes())));
        XMLDocument xml = parser.getDocument();
        XMLElement elmt = (XMLElement)xml.getDocumentElement();
        NodeList nl = elmt.getElementsByTagName("ID"); // ORD ID
        for (int i=0; i<nl.getLength(); i++)
        {
            XMLElement ordId = (XMLElement)nl.item(i);
            XMLNode theText = (XMLNode)ordId.getFirstChild();
            ordIdValue = theText.getNodeValue();
            currOrdId = ordIdValue;
            break; // 1 つ目のみです。
        }
        nl = elmt.getElementsByTagName("CUSTOMER_ID"); // 顧客 ID
        for (int i=0; i<nl.getLength(); i++)
        {
            XMLElement ordId = (XMLElement)nl.item(i);
            XMLNode theText = (XMLNode)ordId.getFirstChild();
            custIdValue = theText.getNodeValue();
        }
    }
    catch (SAXParseException e)
    {
        System.out.println(e.getMessage());
    }
    catch (SAXException e)
    {
        System.out.println(e.getMessage());
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
}
```

```
        catch (Exception e)
        {
            System.err.println("Oops:\n" + e);
        }
        this.shipButton.setEnabled(true);
        String custName = "";
        try
        {
            PreparedStatement pstmt = x2d.getConnection().prepareStatement("Select
C.NAME from CUSTOMER C where C.ID = ?");
            pstmt.setString(1, custIdValue);
            ResultSet rSet = pstmt.executeQuery();
            while (rSet.next())
                custName = rSet.getString(1);
            rSet.close();
            pstmt.close();
        }
        catch (SQLException e)
        {}

        this.ordMessage.setText("Order [" + ordIdValue + "] to process for [" +
custName + "]);
        JOptionPane.showMessageDialog(this, "New Order Pending !", "Wake Up !",
JOptionPane.INFORMATION_MESSAGE);
    }
}
frame.setVisible(false);
}

void shipButton_actionPerformed(ActionEvent e)
{
    // メッセージを送信します。
    String doc2send = "<SHIP>" + currOrdId + "</SHIP>";
    // XMLDoc をキューに送信します。
    aqw.writeQ(new B2BMessage(MessageHeaders.APP_B,
MessageHeaders.APP_A,
MessageHeaders.UPDATE_ORDER,
doc2send));
    aqw.flushQ(); // コミットします。

    // ボタンを無効にします。
    this.shipButton.setEnabled(false);
    // 待機メッセージを表示します。
    this.ordMessage.setText("Waiting for orders...");
}
```

```
private String _userInput(String prompt) throws Exception
{
    String retString;
    System.out.print(prompt);
    try { retString = stdin.readLine(); }
    catch (Exception e)
    {
        System.out.println(e);
        throw(e);
    }
    return retString;
}
```

Java の例 27: エージェントによる小売業者からの注文受信の通知 - SupplierWatcher.java

```
package B2BDemo.Supplier;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//import oracle.bali.ewt.border.UIBorderFactory;
//import oracle.bali.ewt.olaf.OracleLookAndFeel;

import B2BDemo.XMLUtil.*;

/**
 * このクラスは、注文が送信されるキュー上で待機するエージェントを実装します。
 * メッセージが読み込まれると、エージェントが注文の出荷を通知し、促します。
 * 注文を出荷すると新しい B2B プロセスが起動され、小売業者データベース内の
 * 注文状態が更新されます。
 *
 * @author Olivier LE DIOURIS - Partner Technical Services - Oracle Copr.
 */
public class SupplierWatcher
{
    private static boolean stepByStep = false;
    private static boolean verbose    = false;
```

```
private static Integer pauseTime = null;
private static String url = "jdbc:oracle:thin:@localhost:1521:ORCL"; // デフォルト値
/**
 * コンストラクタ
 */
public SupplierWatcher()
{
    XMLFrame xmlFrame = new XMLFrame("Supplier");
    /*
    try
    {
        OracleLookAndFeel.setColorScheme(Color.cyan);
//      OracleLookAndFeel.setColorScheme("Titanium");
        UIManager.setLookAndFeel(new OracleLookAndFeel());
        SwingUtilities.updateComponentTreeUI(xmlFrame);
        xmlFrame.setBackground(UIManager.getColor("darkIntensity"));
    }
    catch (Exception e)
    {
        System.err.println("Exception for Oracle Look and Feel:" + e );
    }
    */
    // ウィンドウを中央に配置します。
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = xmlFrame.getSize();
    if (frameSize.height > screenSize.height)
    {
        frameSize.height = screenSize.height;
    }
    /**
    if (frameSize.width > screenSize.width)
    {
        frameSize.width = screenSize.width;
    }
    */
    frameSize.width = screenSize.width / 3;
//    xmlFrame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height
- frameSize.height)/2);
    xmlFrame.setLocation((2 * frameSize.width), (screenSize.height -
frameSize.height)/2);
//    xmlFrame.addWindowListener(new WindowAdapter() { public void
windowClosing(WindowEvent e) { System.exit(0); } });
    xmlFrame.setVisible(true);
}
```

```

        SupplierFrame frame = new SupplierFrame(verbose, stepByStep, pauseTime,
xmlFrame, url);
        /*
        try
        {
            OracleLookAndFeel.setColorScheme(Color.cyan);
//      OracleLookAndFeel.setColorScheme("Titanium");
            UIManager.setLookAndFeel(new OracleLookAndFeel());
            SwingUtilities.updateComponentTreeUI(frame);
            frame.setBackground(UIManager.getColor("darkIntensity"));
        }
        catch (Exception e)
        {
            System.err.println("Exception for Oracle Look and Feel:" + e );
        }
        */
        // ウィンドウを中央に配置します。
        screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        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);

        frame.enterTheLoop();
        frame.setVisible(false);
        xmlFrame.setVisible(false);
        System.exit(1);
    }

    private static void setRunPrm(String[] prm)
    {
        for (int i=0; i<prm.length; i++)
        {
            if (prm[i].toLowerCase().startsWith("-verbose"))
                verbose = isolatePrmValue(prm[i], "-verbose");
        }
    }

```

```

else if (prm[i].toLowerCase().startsWith("-help"))
{
    System.out.println("Usage iB2BDemo.Supplier.MessageBroker");
    System.out.println("\tparameters can be -dbURL -verbose, -step, -help");
    System.out.println("\tdbURL contains a string like
jdbc:oracle:thin:@localhost:1521:ORCL");
    System.out.println("\tparameters values can be (except for -help):");
    System.out.println("\t\tnone - equivalent to 'y'");
    System.out.println("\t\ty");
    System.out.println("\t\ttrue - equivalent to 'y'");
    System.out.println("\t\tn");
    System.out.println("\t\tfalse - equivalent to 'n'");
    System.out.println("\t\t-step can take a value in milliseconds");
    System.exit(0);
}
else if (prm[i].toLowerCase().startsWith("-step"))
{
    String s = getPrmValue(prm[i], "-step");
    try
    {
        pauseTime = new Integer(s);
        System.out.println("Timeout " + pauseTime);
        stepByStep = true;
    }
    catch (NumberFormatException nfe)
    {
        pauseTime = null;
        if (s.toUpperCase().equals("Y") || s.toUpperCase().equals("TRUE"))
            stepByStep = true;
        else
            stepByStep = false;
    }
}
else if (prm[i].toLowerCase().startsWith("-dburl"))
{
    url = getPrmValue(prm[i], "-dbURL");
}
else
    System.err.println("Unknown parameter [" + prm[i] + "], ignored.");
}

private static boolean isolatePrmValue(String s, String p)
{
    boolean ret = true;

```

```
if (s.length() > (p.length() + 1)) // +1 : "="
{
    if (s.indexOf("=") > -1)
    {
        String val = s.substring(s.indexOf("=") + 1);
        if (val.toUpperCase().equals("Y") || val.toUpperCase().equals("TRUE"))
            ret = true;
        else if (val.toUpperCase().equals("N") || val.toUpperCase().equals("FALSE"))
            ret = false;
        else
        {
            System.err.println("Unrecognized value for " + p + ", set to y");
            ret = true;
        }
    }
}
return ret;
}

private static String getPrmValue(String s, String p)
{
    String ret = "";
    if (s.length() > (p.length() + 1)) // +1 : "="
    {
        if (s.indexOf("=") > -1)
        {
            ret = s.substring(s.indexOf("=") + 1);
        }
    }
    return ret;
}

/**
 * メイン
 * @パラメータ引数
 */
public static void main(String[] args)
{
    if (args.length > 0)
        setRunPrm(args);

    try
    {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
}
```

```
    }  
    catch (Exception e)  
    {  
        e.printStackTrace();  
    }  
    new SupplierWatcher();  
}  
}
```

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

この章の内容は次のとおりです。

- [JDeveloper 3.2 の概要](#)
- [JDeveloper 3.2 の動作環境](#)
- [BC4J での XML](#)
- [BC4J を使用した XSQL クライアントの構築](#)
- [JDeveloper 3.2 の XML 機能](#)
- [JDeveloper を使用した XML アプリケーションの構築](#)
- [JDeveloper の XML Data Generator Web Bean の使用](#)
- [JDeveloper での XSQL Servlet の使用](#)
- [JDeveloper でのモバイル・アプリケーションの作成](#)
- [FAQ: JDeveloper を使用した XML アプリケーションの構築](#)

JDeveloper 3.2 の概要

JDeveloper バージョン 3.2 は、Java アプリケーションおよび XML アプリケーションの構築および配置用に、統合された完全装備のアプリケーション開発ツールを提供します。

JDeveloper を使用すると、XML データおよび XML ドキュメントを作成および処理するインターネット・アプリケーションを構築、デバッグおよび配置できます。

Oracle JDeveloper 3.2 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 ドキュメントの同時使用が容易になります。Oracle JDeveloper 3.2 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わせ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 Processor
- 関連の JavaBeans コンポーネント
- XSQL Page Wizard
13-9 ページの「[Page Selector Wizard](#)」を参照してください。
- XSQL Element Wizard
13-8 ページの「[XSQL Element Wizard](#)」を参照してください。
- XSQL Action Handler

Business Components for Java (BC4J)

Oracle Business Components for Java 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、再使用可能なビジネス・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、再使用可能なビジネス・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、JavaServer Pages/Servlet アプリケーションまたは Enterprise JavaBeans 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・モデルを再使用し、コードを再作成せずに様々なクライアント、ブラウザおよび無線インターネット・デバイスにアプリケーションを提供できます。

JDeveloper では、新しいビジュアル・ウィザードを使用して XML メタデータの記述を変更し、既存のビジネス・コンポーネントの機能をカスタマイズできます。

Oracle JDeveloper の XML 計画

この章では、JDeveloper で XML コンポーネントを使用してモバイル・アプリケーションを開発する方法について説明します。JDeveloper での XML サポートのロードマップも示します。JDeveloper で現在使用可能な XML サポートの他に、JDeveloper の将来的な展開についても説明します。

JDeveloper 3.2 の動作環境

JDeveloper 3.2 は、128MB 以上の RAM を搭載した Windows NT バージョン 4.0 で動作します。

JDeveloper のシステム最低条件

『Oracle JDeveloper for Windows NT リリース・ノート』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 Oracle8i の実行
- ローカルでの iAS (Internet 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。

JDeveloper 3.2 の入手方法

JDeveloper 3.2 の入手方法については、オラクル社カスタマ・サポート・センターにお問い合せください。

BC4J での XML

JDeveloper 3.2 の BC4J フレームワークは、XML を使用して、オブジェクトの宣言設定および機能を表すメタデータを定義します。カスタムまたは複雑なビジネス・ロジックは、Java で実装できます。

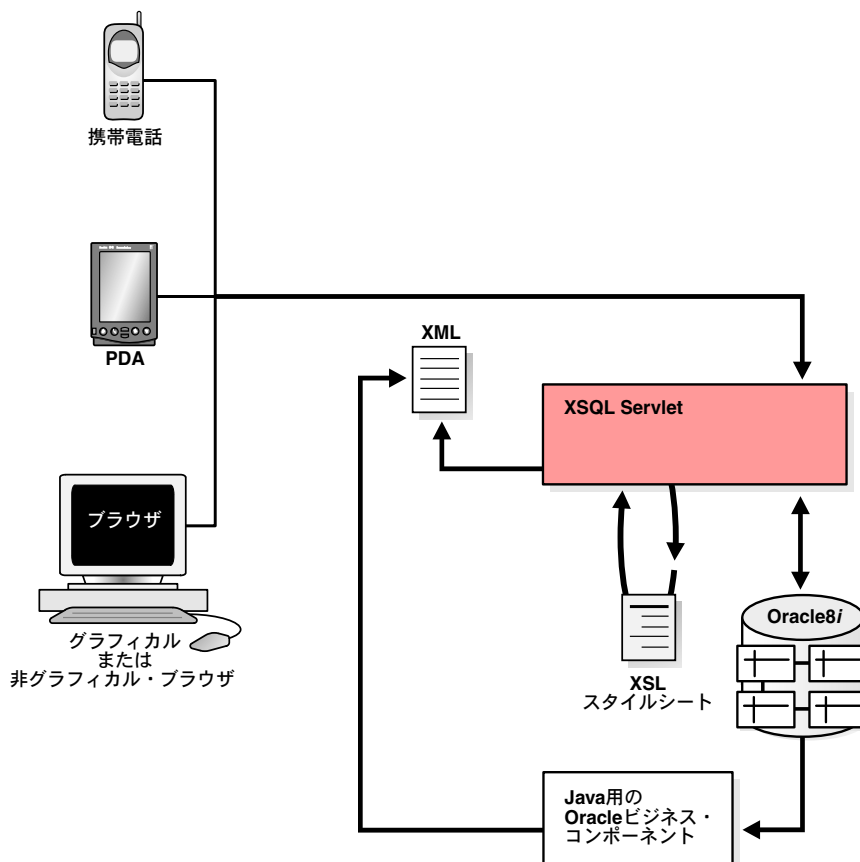
- BC4J Tester を使用すると、View Object 内のデータを XML として参照できます。
- 検証ルールなどのビジネス・ルールは、Java ソース・コードではなく XML に格納されます。
- Java ソース・コードのかわりに XML を変更して、ビジネス・アプリケーションを容易にカスタマイズできます。
- ロジックを XML で抽象化することで、アプリケーションを簡単に読んだり、理解することができます。

BC4J による XML を使用したメタデータの格納 JDeveloper とともに提供される Java フレームワーク用のビジネス・コンポーネントは、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 13-1 に、XSQL Servlet が BC4J を使用して XML ドキュメントを生成する方法を示します。

図 13-1 BC4J の使用



ビジネス・ルールは、基礎となるコンポーネントのソース・コードへアクセスせずに、その場で変更できます。

BC4J を使用した XSQL クライアントの構築

JDeveloper 3.2 では、XSQL Servlet を構築できます。XSQL Servlet は、BC4J アプリケーション・モジュールと統合して、中間層から複数のクライアントへアプリケーション・ロジックを提供します。ユーザーは、対応するスタイルシートを適用するのみで、XML データを取り出し、すべてのクライアント・デバイスでデータを表示できます。

次の機能は、BC4J を使用した XSQL クライアントの構築に有効です。

- Object Gallery
- XSQL Element Wizard
- Page Selector Wizard

注意： JDeveloper 3.2 の製品バージョンでは、これらの機能の表現が異なる場合があります。

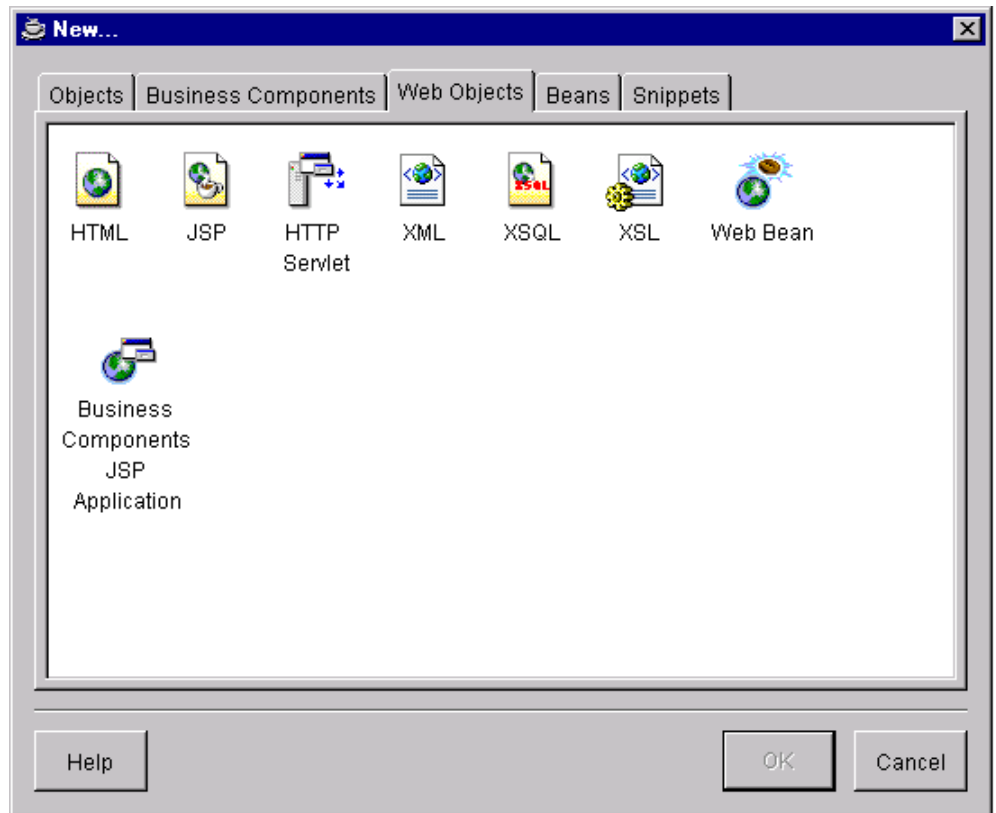
Object Gallery

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 3 つの新しいアイコンがあります。アイコンをクリックすると、これらのページの基本タグが生成されます。ユーザーは、これらのタグを拡張できます。

基本的な XSQL ページの生成後、XSQL Element Wizard を使用して、データにバインドされたタグを XSQL ページに挿入できるという点で、XSQL ページのアイコンは特に重要です。

図 13-2 に、Object Gallery を示します。

図 13-2 新しい XSQL、XML および XSL アイコンがある JDeveloper の Object Gallery

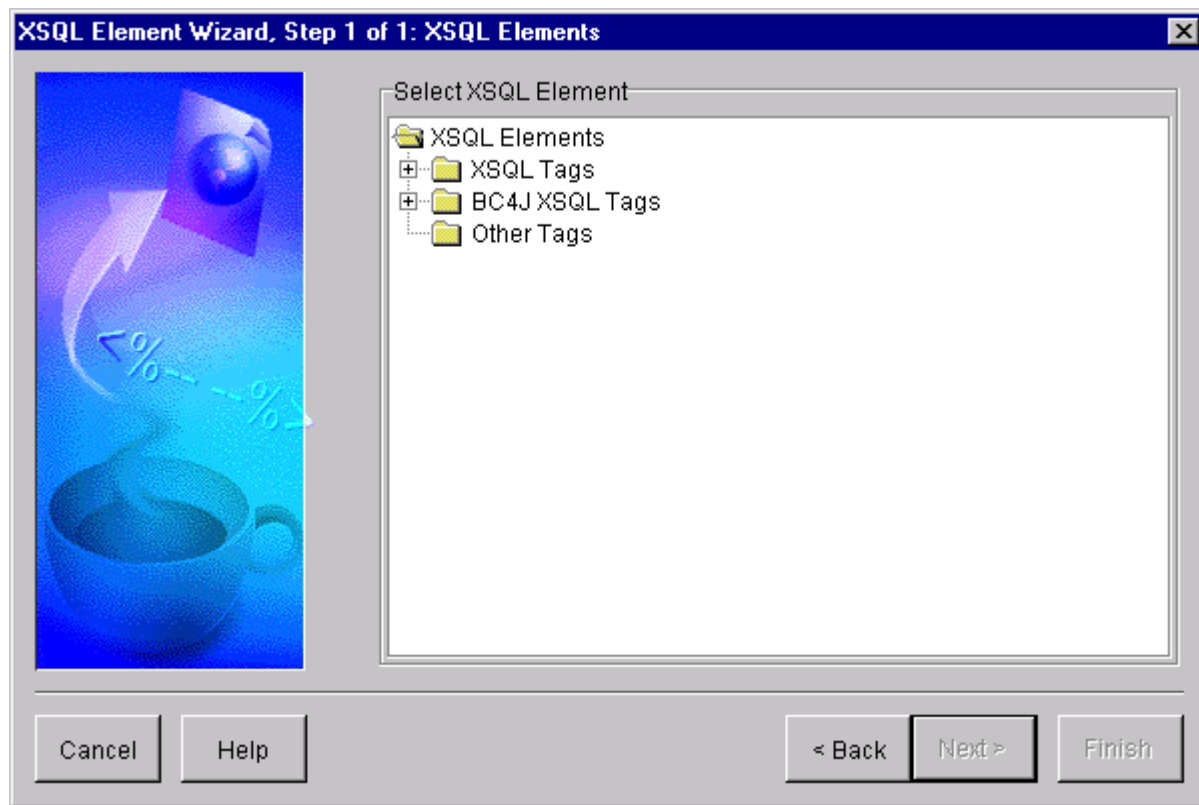


XSQL Element Wizard

XSQL Element Wizard は、データベース表または BC4J View Object へのアクセスを可能にするタグを追加するメカニズムを提供します。これらの表またはオブジェクトに問合せを実行するか、これらを介して基礎となるデータベース表を更新することができます。

図 13-3 に、JDeveloper 3.2 の XSQL Element Wizard を示します。

図 13-3 JDeveloper の XSQL Element Wizard

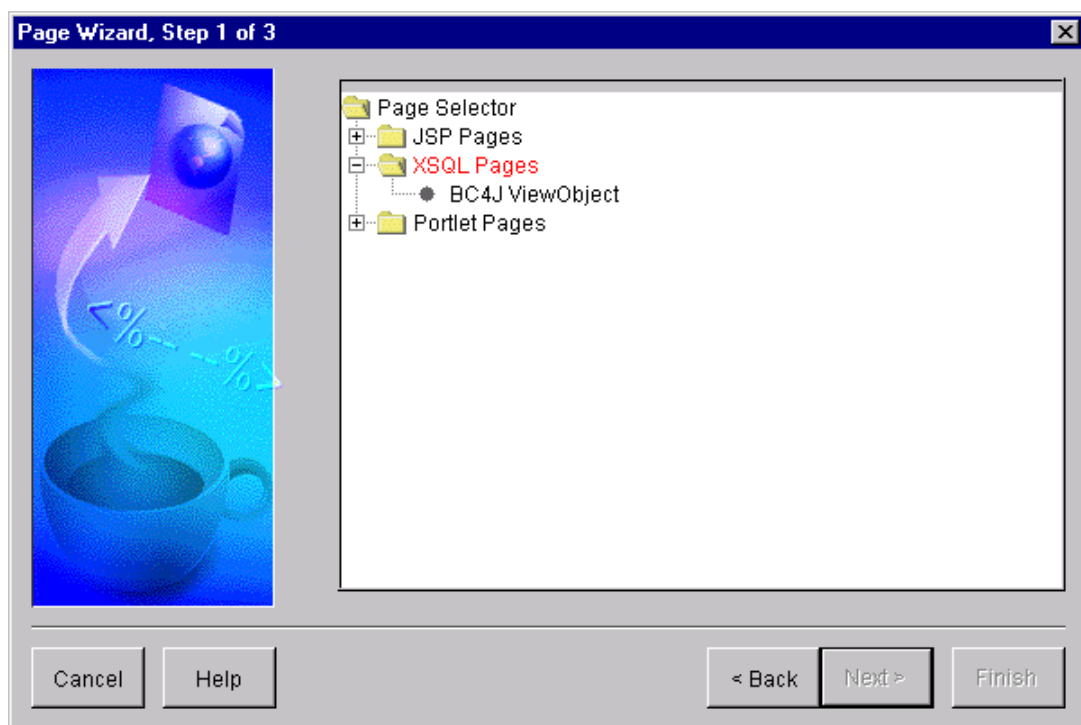


Page Selector Wizard

開発者が、Web アプリケーションの構築プロセスで完全な XSQL ページを作成する必要がある場合、Page Wizard を簡単に起動します。このウィザードを使用して、XSQL ページをデータベース表上に直接、または BC4J View Object 上に作成できます。BC4J View Object 上に XSQL ページを作成する場合、リストからアプリケーション・モジュール (JDeveloper ドキュメントを参照) 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。

図 13-4 に、JDeveloper の Page Selector Wizard を示します。

図 13-4 JDeveloper の Page Selector Wizard



JDeveloper 3.2 の XML 機能

JDeveloper 3.2 がサポートする Oracle XML Developer's Kit for Java (XDK for Java) のコンポーネントは、次のとおりです。

- Oracle XML Parser for Java
- Oracle XSQL Servlet

XSLT Processor を含む Oracle XML Parser for Java および XML SQL Utility は、Java で作成されているため、JDeveloper で使用できます。JDeveloper には、これらのコンポーネントがあります。

これらのツールの使用方法を説明するプログラムの例は、
[JDeveloper]/Samples/xmlsamples ディレクトリにあります。

Oracle XDK と Transviewer Beans の統合

Oracle XDK for Java は、次の XML ツールで構成されます。

- XML Parser for Java
- XML SQL Utility 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。

Oracle XDK for Java には、XML Transviewer Beans も含まれています。これら一連の Java Beans によって、XML アプリケーションにグラフィカルまたはビジュアルなインタフェースを簡単に追加できます。Beans は、JDeveloper から直接アクセスできるように、ドキュメントおよび記述子もカプセル化しています。これらの Beans を TOOLS パレットにドロップして、XML/XSL エディタなどのアプリケーション構築に使用できます。

参照： Transviewer Beans の使用方法の詳細は、[第 19 章「XML Transviewer Beans の使用」](#)を参照してください。

Oracle XML Parser for Java

プロジェクトで Oracle XML Parser for Java を使用すると、XML ドキュメントを検索および処理できるアプリケーションを作成できます。JDeveloper には Oracle XML Parser 用の組込みライブラリがあるため、1 回クリックするのみでプロジェクトに Oracle XML Parser を含めることができます。

Code Insight によって、コードをより簡単に理解および使用し、参照するクラスの Java ドキュメントへ適切にアクセスできます。XML Parser for Java は、次のインタフェースのいずれかを使用して XML ドキュメントの処理を容易にします。

- DOM: W3C DOM のツリー
- SAX: SAX イベントのストリーム

Oracle XSQL Servlet

XSQL Servlet は SQL 問合せを処理し、結果セットを XML として出力するツールです。このプロセッサは Java サーブレットとして実装され、埋込み SQL 問合せを含む XML ファイルを入力として受け入れます。ほとんどの操作の実行には、XML Parser for Java および XML SQL Utility 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 ドキュメントから複雑なドキュメントまで生成できます。
- XSL スタイルシートを適用してあらゆるテキストの形式に生成できます。
- XML ドキュメントを解析し、データをデータベースに格納できます。
- コーディングなしで、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>
    ...
  </EMP>
</EMPLOYEES>
```

JDeveloper 3.2 を使用すると、XSQL ファイルを容易に開発および実行できます。組み込み Web サーバーおよびユーザーのデフォルト Web ブラウザが、結果ページの表示に使用されます。

XSQL Servlet での Action Handler の使用

XSQL Action Handler 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 JDeveloper 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 Action Handler 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 Action Handler 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。

Action Handler 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 Action Handler 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 Action Handler をデバッグします。
3. Java ソース・ファイルでブレークポイントを設定します。
4. .xsql ファイルを右クリックして、メニューの「Debug...」を選択します。

XML Data Generator Web Bean

Oracle JDeveloper には、XML Data Generator Web Bean があります。XML Data Generator Web Bean は、View Object のデータを含む XML を生成し、JSP 応答の出力ストリームにレンダリングします。

クライアントへの応答をレンダリングするために、XML および XSL を使用する JSP ページを作成できます。

XML Web Bean は JSP アプリケーションおよび Servlet アプリケーションで使用できます。XML Web Bean は、ビジネス・コンポーネント (View Object) からデータを読み込み、適切な XML を生成します。Web Bean のメリットは、ビジネス・コンポーネント・アプリケーションを分析し、階層をナビゲートしてネストした XML を作成できることです。

XML Web Bean によって、XSL スタイルシートも指定できます。Web Bean は、XML に加えて、HTML、WML、変換済 XML およびその他のテキスト形式を生成できます。

Portal-To-Go および JDeveloper を使用したモバイル・アプリケーション開発

Portal-to-Go および Oracle JDeveloper を併用すると、モバイル・アプリケーション開発に非常に強力な環境が提供されます。開発者は、JDeveloper を使用してデータベースまたは BC4J アプリケーションから XML を生成し、Portal-To-Go を使用して Web ブラウザ、PDA または携帯電話にコンテンツを配信することができます。

JDeveloper を使用した XML アプリケーションの構築

次の例を考えてみます。この例では、データを提供するのではなく表示するために XML を使用する方法を説明します。ここでは、従業員と部門の間の多対 1 関係を示しています。

JDeveloper の XML の例 1: BC4J メタデータ

```
<Departments>
<Dept>
  <Deptno>10</Deptno>
  <Dname>Sales</Dname>
  <Loc>
  <Employees>
    <Employee>
      <Empno>1001</Empno>
      <Ename>Scott</Ename>
      <Salary>80000</Salary>
    </Employee>
  </Employees>
  ...
  </Employee>
</Employees>
</Dept>
<Dept>
...
```

JDeveloper 3.2 でのアプリケーションの構築手順

JDeveloper 3.2 でこのプロジェクトを構築するには、次の手順に従います。

1. 「File」>「New Project」を選択して、新規の JDeveloper プロジェクトを開始します。
2. BC4J アプリケーションを作成します。
3. Page Selector Wizard を起動して、BC4J アプリケーション・モジュールに基づいて XSQL ページを作成します。[図 13-4](#) を参照してください。
4. 表示されたリストからアプリケーション・モジュールを選択します。
5. XSQL ページの基になる View Object を選択します。
6. 表示する列を選択します。

Page Wizard でのこれらの手順を終了すると、BC4J フレームワークの View Object に基づいた XSQL ページが作成されます。このページを実行すると、ブラウザに XML データが送信されます。

データが希望どおりに表示されるようにフォーマットするために、オプションでスタイルシートを作成できます。または、PDA または携帯電話で表示されるように調整できます。

JDeveloper の XML Data Generator Web Bean の使用

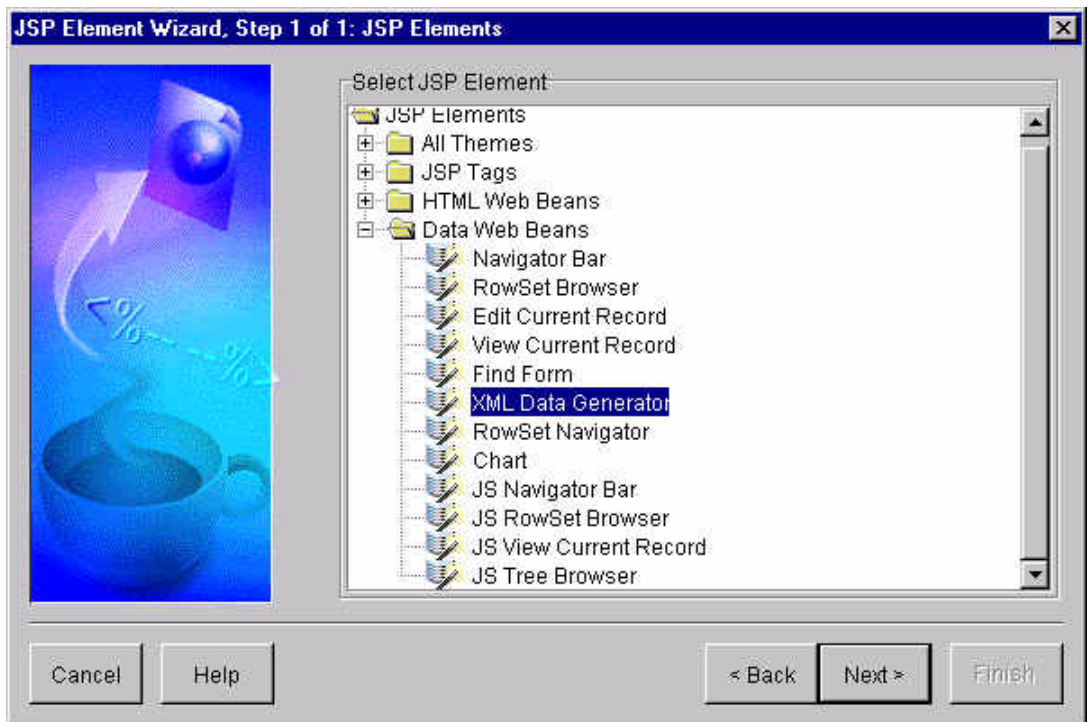
XML Data Generator Web (Bean) は、JSP および Servlet アプリケーションで使用できます。XML Data Generator Web (Bean) は、ビジネス・コンポーネント (View Object) からデータを読み込み、適切な XML を作成します。この Web Bean のメリットは、次のとおりです。

- ビジネス・コンポーネント・アプリケーションを分析し、階層をナビゲートしてネストした XML を作成します。
- XSL スタイルシートを指定できます。Web Bean は、HTML、WML、変換済 XML およびその他のテキストの形式を生成できます。

Data Generator Web Bean は、JSP Element Wizard の「Data Web Beans」カテゴリにあります。図 13-5 に、JSP Element Wizard から XML Data Generator Web (Bean) へアクセスする方法を示します。

要素を含めるページ上を右クリックして、JSP または XSQL Servlet の Element Wizard をコールします。Element Wizard のパラメータとしてスタイルシートを指定します。

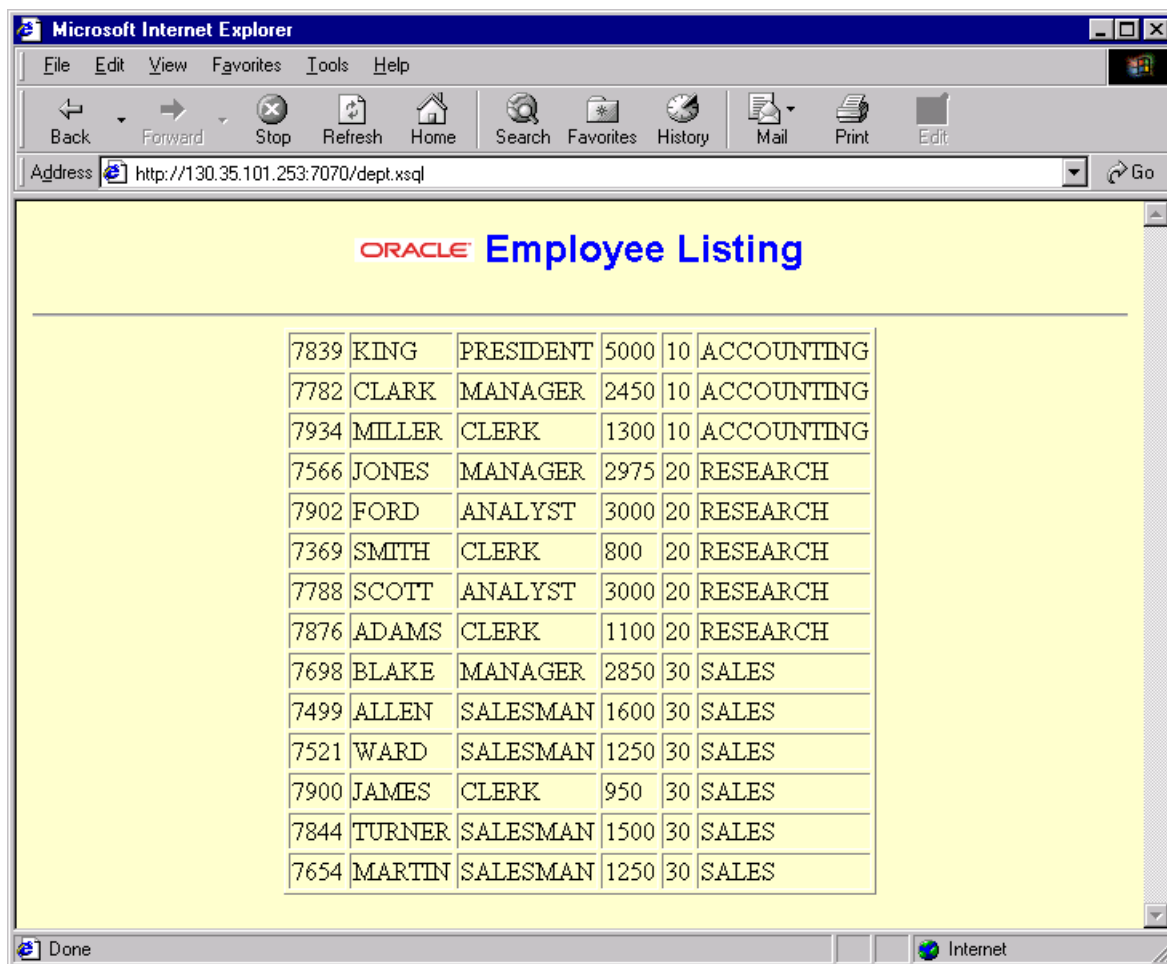
図 13-5 JSP Element Wizard: XML Data Generator Bean



このウィザードを起動すると、生成する XML データに適用するスタイルシートを指定し、出力結果を参照できます。

図 13-6 に、従業員リストに XSL スタイルシートを適用して表示される出力例を示します。

図 13-6 ブラウザに HTML で表示される従業員リスト (XML+XSLT= HTML)



JDeveloper での XSQL Servlet の使用

XSQL Servlet を使用すると、効率的かつ簡単に XML をデータベースから出し入れできます。

参照： XSQL Servlet の使用方法は、第 3 章「[Oracle の XML コンポーネントおよび一般的な FAQ](#)」および第 18 章「[XSQL Servlet の使用](#)」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 ドキュメントから複雑なドキュメントまで生成できます。
- XSL スタイルシートを適用して、あらゆるテキスト形式に生成できます。
- XML ドキュメントを解析し、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 13-7 に、ブラウザに表示される従業員の XML データを示します。

図 13-7 RAW XML 形式の従業員データ

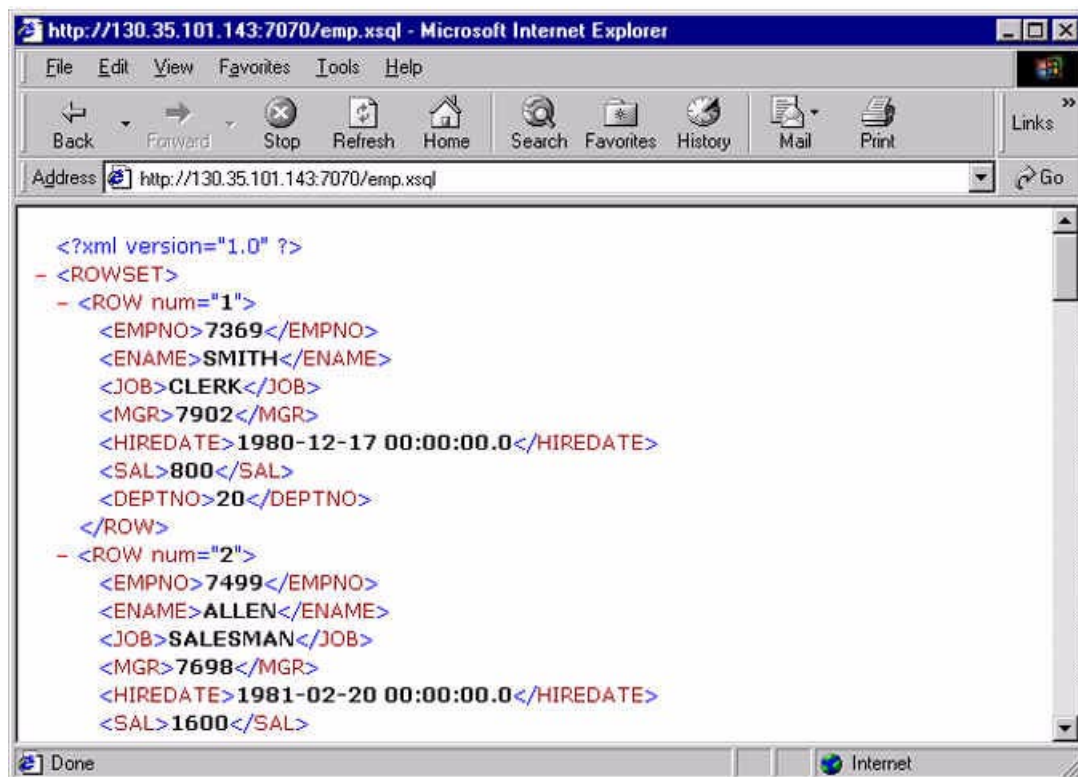


図 13-6 に示すとおりデータを表形式で出力する場合、スタイルシートを指定するために XSQL コードを少し変更する必要があります。この例で行う変更は、次の例で強調されている部分です。

JDeveloper の XSQL の例 3: スタイルシートが追加された従業員データ

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
  select *
  from emp
  order by empno
</xsql:query>
```

結果は、[図 13-6](#) の表のようになります。XSQL Servlet を使用して、他にも多くのことが実行できます。

参照： [第 18 章「XSQL Servlet の使用」](#) を参照してください。

JDeveloper でのモバイル・アプリケーションの作成

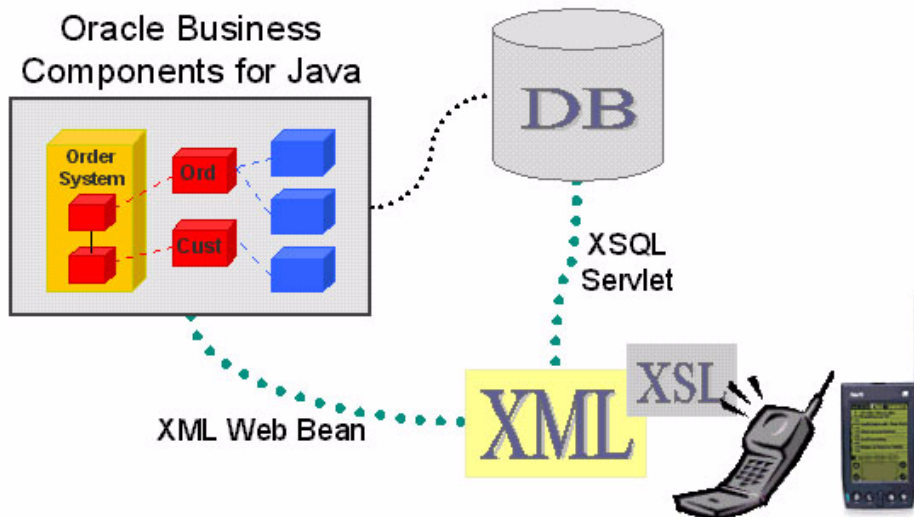
このモバイル・アプリケーションは、基本的に部門データベース・アプリケーションです。このアプリケーションの主な目的は、BC4J および XML を使用して、無線デバイスでアクセスできるアプリケーションを開発できることを実証することです。

このアプリケーションは、主に次の 2 つの部分で構成されます。

- 1 つは BC4J フレームワークを使用して開発されるサーバー側のビジネス・ロジックであり、もう 1 つはクライアント部分です。ビジネス・ロジックは、SCOTT スキーマの DEPT 表に基づいた参照オブジェクトで構成されます。
- DEPT 表を問い合わせ、ブラウザ、携帯電話や 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 13-8 に、モバイル・アプリケーションが BC4J、XSQL Servlet、XSL スタイルシートおよび Oracle8i でどのように動作するかを概念的に示します。

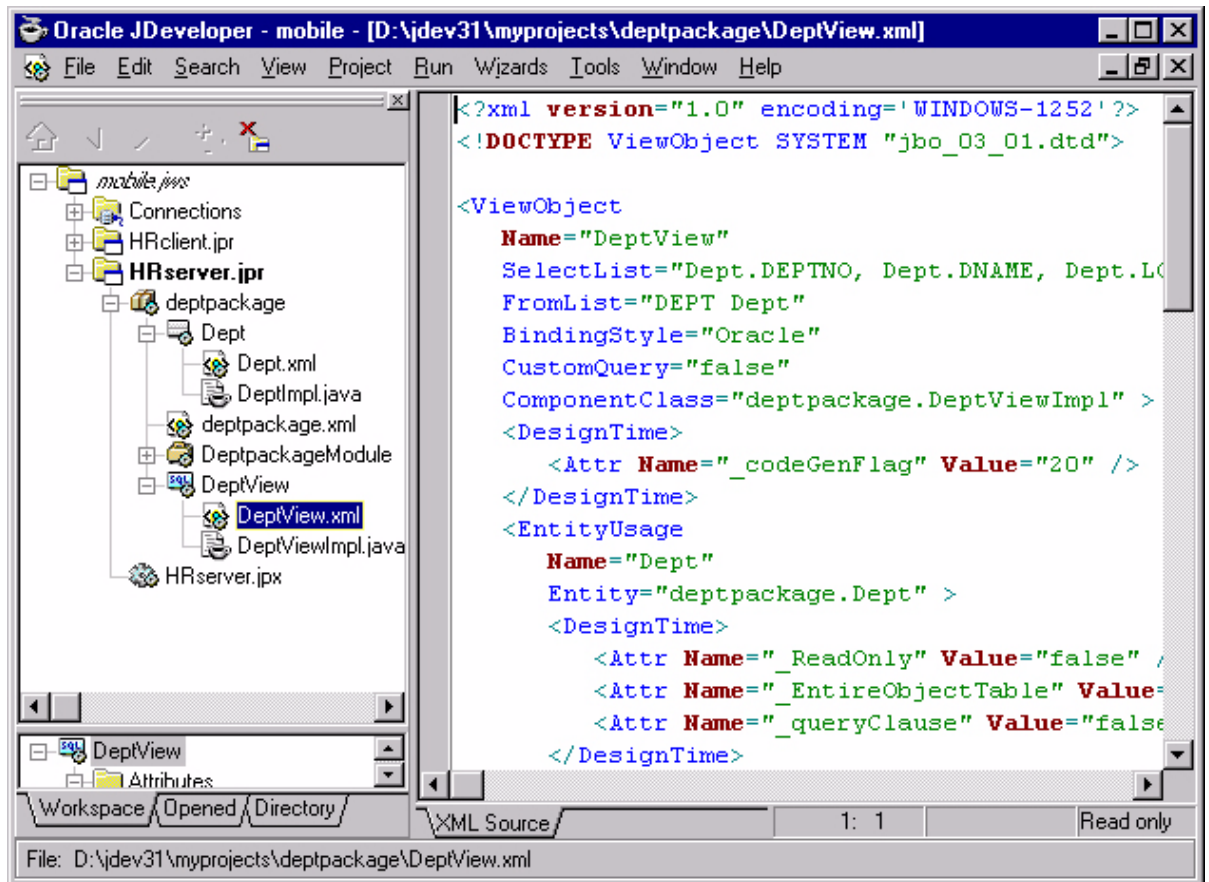
図 13-8 BC4J および XSQL Servlet を使用した JDeveloper でのモバイル・アプリケーションの作成



BC4J アプリケーションの作成

まず、BC4J アプリケーションを作成します。このアプリケーションは、Oracle8i データベースで SCOTT スキーマに接続します。図 13-9 に、DEPT オブジェクトに関するメタデータを含む XML ファイルを示します。13-14 ページの「JDeveloper の XML の例 1: BC4J メタデータ」を参照してください。

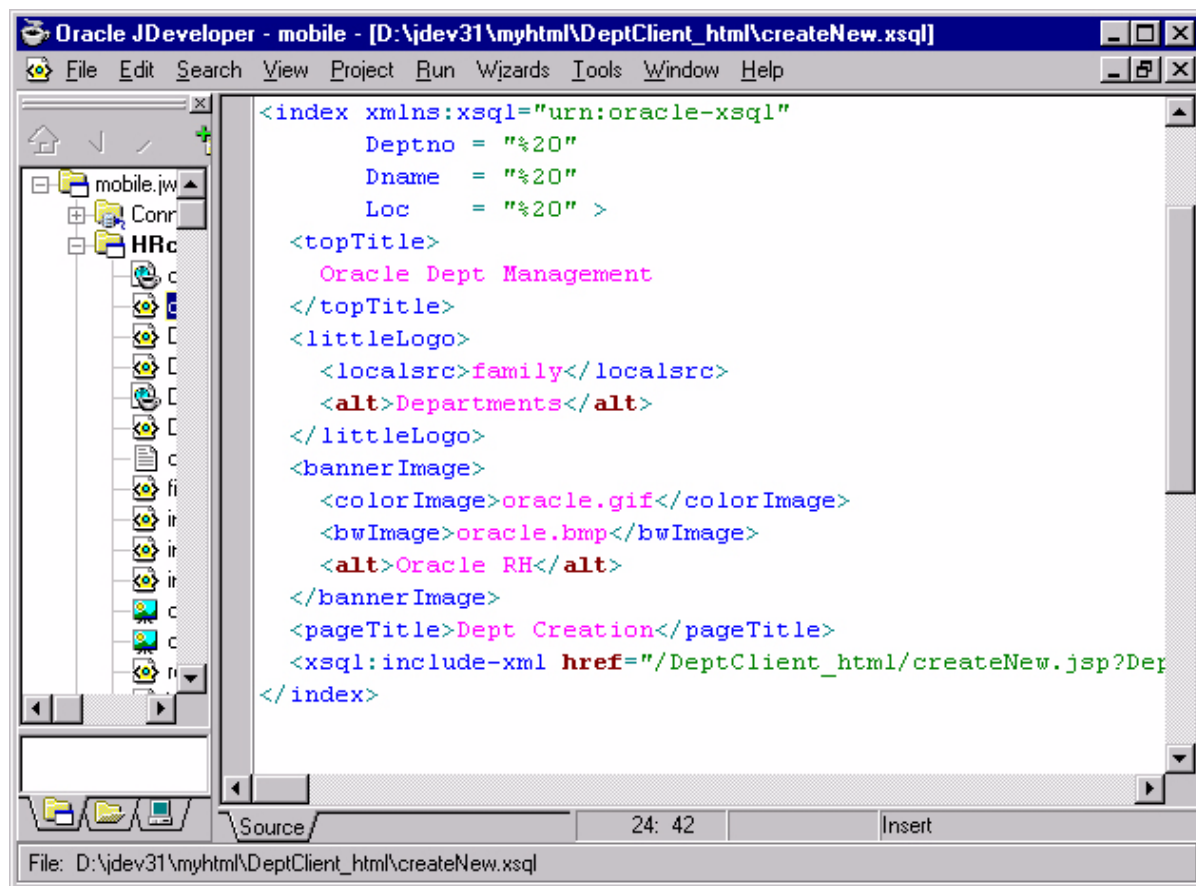
図 13-9 BC4J アプリケーション : DEPT View Object の XML ファイル



BC4J アプリケーションに基づいた JSP ページの作成

BC4J アプリケーションに基づいて JSP ページを作成します。JSP ページでは、XML Data Generator Web Bean を導入します。図 13-10 に、新しい部門を作成するために JSP ページをコールする XSQL ファイルを示します。

図 13-10 BC4J アプリケーション：JSP ページをコールする XSQL ファイル



データ読込みに必要なデバイスに従った XSLT スタイルシートの作成

アクセスするデータの様々なクライアント・デバイスに対応する XSLT スタイルシートを作成します。XSQL ファイルでは、スタイルシートのリストおよび対応するプロトコルを指定します。このプロトコルは、基本的にスタイルシートをクライアント・デバイスに結合させます。

図 13-11 に、Palm Pilot のエミュレータのブラウザ上に表示するために、XML データを HTML に変換するスタイルシート (indexPP.xml) のコード例の一部を示します。

図 13-11 BC4J アプリケーション: XSL スタイルシート (indexPP.xml)

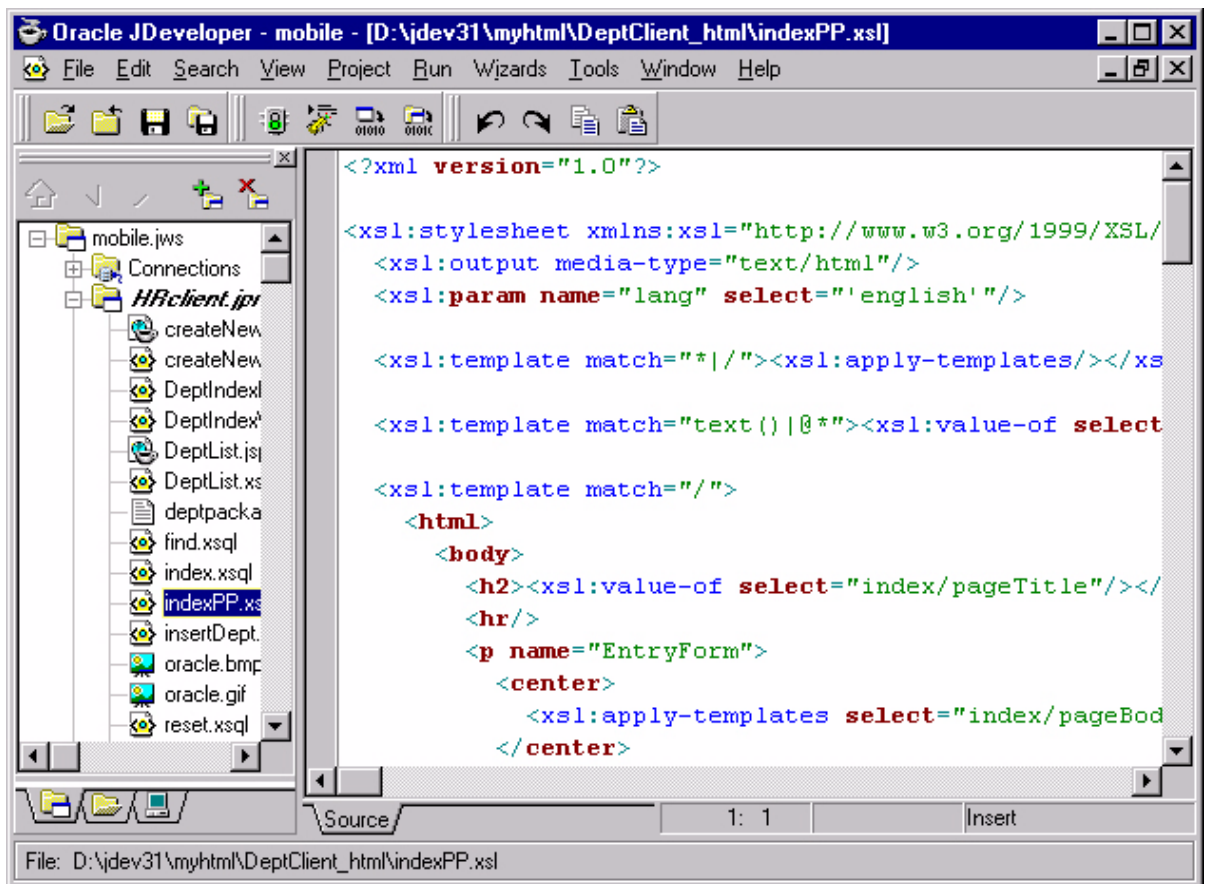


図 13-12 に、部門アプリケーションのクライアントを実行中の携帯電話のエミュレータを示します。携帯電話のエミュレータの設定画面も示しています。

図 13-12 部門アプリケーションのクライアントを実行中の携帯電話のエミュレータ

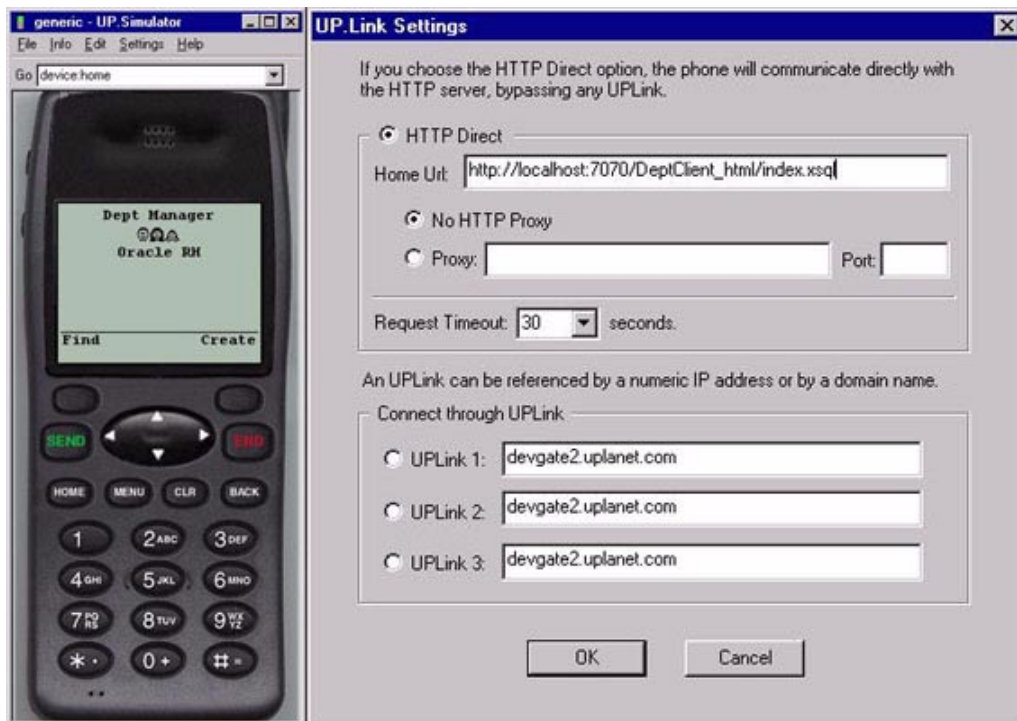


図 13-13 に、HandWeb ブラウザを介して部門アプリケーションにアクセス中の Palm Pilot のエミュレータを示します。

図 13-13 HandWeb ブラウザを介して部門アプリケーションにアクセス中の Palm Pilot のエミュレータ



FAQ: JDeveloper を使用した XML アプリケーションの構築

JSP での XML ドキュメントの作成

質問

XML Class Generator for Java が (DTD に基づいて) 生成したクラスを使用し、XSL スタイルシートを適用して HTML に変換して、JSP ページに XML ドキュメントを (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 Parser がこの無効な状態を取得すると、前述の例外を発生させます。

CRM はアプリケーションで HTTP セッションを使用しません。ご質問の件も、この場合に該当すると思われます。メンバー変数を使用して、誤って無効な状態を格納した可能性があります。メンバー変数とは、次の構文で宣言された変数です。

```
<%! %>
```

次に例を示します。

```
<%! Document doc=null; %>
```

変数宣言のためにこの構文を使用する必要があると誤解している方が多いようですが、実際はこの構文を使用する必要はありません。ほとんどの場合、メンバー変数は必要ではありません。メンバー変数はすべての要求で共有され、JSP の存続期間中に 1 回のみ初期化されます。

ほとんどのユーザーには、スタック変数またはメソッド変数が必要です。これらの変数は、要求されるたびに作成および初期化されます。変数は次の例のとおり、スクリプトレット形式として宣言されます。

```
<% Document doc=null; %>
```

この場合、すべての要求には固有のドキュメント・オブジェクトがあり、このオブジェクトは要求されるたびに初期化されて NULL になります。

セッションに無効な状態も JSP にメソッド変数も格納していない場合、この問題には他の理由が考えられます。

BC4J での XMLData の使用

質問

BC4J のデータを取り出すために、XMLData を使用しています。JSP の XMLData ではなく、スタンドアロンの Java アプリケーションのものを使用しています。ターゲット・レコードに「R & D」という値があります。

XMLData が戻す「R & D」は、HTTP には適切ですがこの場合は適切ではありません。XMLData は文字をエスケープせずに、データベースにあるものを戻せますか？

回答

XMLData はメモリ内 DOM を構築するため、問題なのは XML Parser のシリアル化です。唯一考えられる対処方法は、次のとおりです。

1. 希望どおりに動作する DOM ツリー用のシリアル化コードを作成します。
2. テンプレートを 1 つ追加した認証変換を行い、disable-output-escaping=YES を設定したデータを書き込みます。

JDeveloper 3.0 での XML Parser for Java の実行

質問

JDeveloper 3.0 をノートブック・コンピュータ（オペレーティング・システムは Windows 95）にダウンロードしました。XML Parser プログラム例（SimpleParse.java）を実行しようとしています。このプログラムは、org.w3c.dom.Document クラスをインポートします。正確なディレクトリで autoexe.bat に CLASSPATH を設定しました。DOS プロンプトでは、「java SimpleParse <filename>」コマンドでプログラムを実行できます。JDeveloper を介して同じプログラムを実行しようとする、次のエラーが表示されます。

「identifier org.w3c.dom.Document not found」

何か不足しているものがありますか？

回答

プロジェクトに、Oracle XML Parser 2.0 という名前のライブラリがあることを確認してください。

このライブラリは JDeveloper 3.0 によって事前定義されています。このプロジェクトのライブラリ・リストを参照するには、「Project | Properties...」を選択し、「Paths」タブを選択します。

「(Add...)」ボタンをクリックし、リストから前述のライブラリを指定します。

org.w3c.dom.* インタフェースは、この Jar に含まれます。この W3C インタフェースは、XML ノードのツリーと動作するための、DOM 標準 API を定義します。

質問

@code をキーとして使用する場合、次の文を使用します。

```
<xsl:template match="aTextNode">
  ...
  <xsl:param name="labelCode" select="@code"/>
  <xsl:value-of
    select="document('messages.xml')/messages/msg[@id=$labelCode and
      @lang=$lang]" />
  ...
</xsl:template>
```

それでも動作しますが、@code を直接 document() 行に使用する方法はありませんか？

回答

これは、current() 関数が有効な場合の方法です。次の方法は使用しないでください。

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
  select="document('messages.xml')/messages/msg[@id=$labelCode and
    @lang=$lang]" />
```

かわりに次のように実行します。

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
and @lang = $lang]" />
```

質問

messages.xml に格納されたデータをデータベースから取り出せますか？リスナーおよびサーバーレットがデータベース内で動作するところでは、document() の命令はどうなりますか？

回答

データは取り出せます。仕様によって、XSLT エンジンでは document() 関数で参照されるドキュメントを読み込みおよびキャッシュします。渡された URI の文字列形式に基づいて解析ドキュメントをキャッシュし、次のとおりデータベース・ベースのメッセージ検索を行います。

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>
    </body></html>
  </template>
</stylesheet>
```

```

        </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, '&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}&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 ドキュメントのデータベースへの移動

質問

Oracle データベースに XML ドキュメントを移動しています。ドキュメントは非常に複雑です。XML ドキュメントおよび Oracle XML Developer's Kit (XDK) は、XML ドキュメント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、XML SQL Utility を使用します。マッピング・プログラムも作成する必要があります。

別の方法では、ビューおよびストアド・プロシージャを作成して複数の表を更新します。ただし、どちらの場合でも事前に表およびビューを作成する必要があります。

BC4J BC4J フレームワークは、データベースへ出入りする E-Commerce の XML メッセージをマッピングするための、一般的なメタデータ駆動のソリューションを提供します。

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。BC4J は独自に使用可能な Java フレームワークですが、きめ細かな開発サポートが JDeveloper 3.0 IDE に組み込まれています。

BC4J によって、すべてのアプリケーション動作（ルールおよびプロセス）を一定の方法で管理するためのビジネス・コンポーネントに、SQL ベースの参照コンポーネントの階層を柔軟にマッピングできます。動的な機能がサポートされているため、ほとんどの機能は XML メタデータ以外から実行できます。

このフレームワークを使用して、あらゆる XML ドキュメントをデータベースに、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 ドキュメント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。

Internet File System (iFS) を使用した XML アプリケーションの作成

この章の内容は次のとおりです。

- Internet File System (iFS) の概要
- iFS での XML の使用
- iFS パーサーの使用
- iFS 標準パーサーの使用
- iFS カスタム・パーサーの使用
- iFS XML の解析方法
- パーサー・アプリケーションの作成
- iFS での XML のレンダリング
- XML およびビジネス・インテリジェンス
- XML ファイルを使用した iFS の構成

Internet File System (iFS) の概要

Internet File System (iFS) は、標準の Windows、および SMB、HTTP、FTP、SMTP、IMAP4 などのインターネット・プロトコルを介して、ファイルベースおよびフォルダベースの隠喩を使用したドキュメントおよびデータの編成やアクセスを容易にします。

iFS は、Web ベースのアプリケーションの構築および管理を支援します。iFS は Java 用アプリケーション・インタフェースであり、Powerpoint の .ppt ファイルなどのドキュメントを Oracle8i にロードしたり、Oracle8i JVM などの Web サーバーのドキュメントを表示することができます。

iFS での XML の使用

iFS を使用すると、開発者は容易に XML を使用した作業ができます。これは、iFS が XML のリポジトリとして機能するためです。iFS は XML ドキュメントで次のタスクを行えます。

- 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。

ファイルが要求されると、選択された情報を Web などに配信します。

iFS は、新しいファイル型を定義する拡張可能な方法をサポートし、XML ドキュメントのファイル型を定義、解析およびレンダリングする組込みサポートも提供します。

Oracle iFS は、XML データを格納するのみでなく、ビジネス・インテリジェンスの実装、動的コンテンツの生成およびアプリケーション間でのデータ共有のために、XML のすべての可能性を追求します。

ドキュメント記述子の供給

iFS では、XML ベースのファイル型を登録する場合、次のものを指定するドキュメント記述子を指定する必要があります。

- ファイル型の XML ドキュメント構造
- データベースでの格納方法

たとえば、ドキュメントを完全な形式でデータベースのラージ・オブジェクト (LOB) に保存できます。

Oracle iFS のドキュメント記述子は、XML ベースの構文を使用して、XML ベースのファイル型の構文 (またはスキーマ) を記述します。

ファイルが iFS に保存または送信される場合、iFS はドキュメントを使用中のファイル型として認識し、XML を解析して、ドキュメント記述子で指定したとおりに表にデータを格納します。

ファイル型の特定のインスタンスが iFS のサポートされたプロトコルを介して要求される場合、同じ情報が XML ドキュメントのレンダリング（配信用の再組立て）に使用されます。

参照：『Oracle Internet File System Developer's Guide』を参照してください。

iFS パーサーの使用

iFS が構造を理解できる新しい XML ファイルをフォルダにドロップする場合、Oracle XML Parser for Java は XML ファイルを解析し、個々の属性を iFS スキーマに格納できます。iFS サブクラスを定義すると、どの属性がどの列へ配置されるかが指定されます。iFS は、デフォルトの属性と列間のマッピングも行えます。

一度解析されると、ファイル内の属性はファイルの属性となります。これは、ファイル・システムの検索基準として編集および使用できる追加のメタデータです。

XML ベースの企業向け保険金支払請求標準フォームを考えてみます。XML の支払請求ファイルを解析し、それぞれのファイルの属性タグ情報を取り出して、これらのチャンクを個別に表に格納するように iFS に指示できます。

その後、ファイルの属性を検索する場合と同様に、地域やエージェントなどの XML 属性を検索できます。データは、保険業界の分析ツールなどの関連アプリケーションにも使用可能です。

XML ファイルは、解析されても使用できなくなるわけではありません。元の XML ファイルがオープンされると、XML レンダラはファイルを再構築します。14-7 ページの「[iFS での XML のレンダリング](#)」を参照してください。

標準の iFS パーサーおよびカスタム・パーサー

iFS では、アプリケーションが作成した文書書式に必要な場合、XML アプリケーションがカスタム・パーサーを要求します。

iFS パーサーを明示的に起動する必要がある場合

iFS パーサーの起動方法は、アプリケーションが作成したドキュメントを iFS に登録した方法によって異なります。

- プロトコル・サーバーを使用して XML ドキュメントがアップロードされる場合、iFS XML パーサーはプロトコル・サーバーによって自動的に起動されます。

- XML ドキュメントがアプリケーションによってアップロードされる場合、iFS XML パーサーまたはカスタム・パーサーのどちらかを明示的に起動する必要があります。アップロードされない場合、XML ドキュメントはオブジェクトに解析されるかわりにロー・データとして読み込まれます。
- アプリケーションが XML 以外の形式でドキュメントを生成するカスタム・クラスを定義する場合、iFS Java API の一部として提供されるクラスおよびメソッドを使用して、カスタム・パーサーを作成します。

カスタム・パーサーは、カスタム・クラスの iFS リポジトリ・オブジェクトを作成します。

たとえば、ドキュメント・クラスをサブクラス化するメモ・クラスを定義したとします。メモ・クラスには、カスタム属性の宛先、伝言者、日付および本文（メモの内容）が含まれています。

iFS にメモ・オブジェクトを格納するには、パーサーが必要です。

- メモ・ドキュメントが XML の場合、iFS SimpleXmlParser() を使用して、属性を取り出すことができます。
- メモ・ドキュメントが特別な形式で格納されている場合、カスタム・パーサーを作成し属性の取出し方法を指定します。

iFS 標準パーサーの使用

iFS には、アプリケーションの作成用のいくつかの標準パーサーがあります。表 14-1 に、iFS 標準パーサー・クラスを示します。

表 14-1 iFS 標準パーサー・クラス

クラス	説明
SimpleXmlParser	XML ドキュメント本体の iFS リポジトリでオブジェクトを作成します。iFS に格納されるすべての XML ドキュメントのデフォルトのパーサーです。SimpleXmlParser は XmlParser を拡張します。
XmlParser	カスタム XML パーサー開発用のベース・クラスです。
SimpleTextParser	カスタム・パーサーを作成する開発者の出発点となります。SimpleTextParser は、名前 = 値という単純な構文を使用します。
ClassSelectionParser	カスタム属性を指定形式のすべてのファイルに追加します。実際には解析しません。

解析オプション

iFS には次の解析オプションがあります。

- **SimpleXmlParser** (最小限のカスタマイズ) : FTP、SMB、Windows インタフェース、および iFS Web ユーザー・インタフェースで動作し、ドラッグ・アンド・ドロップを使用してアップロードできます。
- **ClassSelectionParser**: 実際には解析しません。ファイルがリポジトリに格納される前に、すべての .doc ファイルなど、特定のファイル拡張子を持つファイルに 1 つ以上のカスタム属性を追加できます。特定のファイル形式にクラスをマッピングします。

iFS カスタム・パーサーの使用

.doc や .xls などの、XML 以外のドキュメントを解析する場合、または Vcard 用の .vcf などのカスタム・タイプを定義した場合、これらのドキュメントのデータベース・オブジェクトを作成するためにカスタム・パーサーを作成する必要があります。カスタム・パーサーの作成には、既存の iFS パーサーをサブクラス化するか、最初からカスタム・クラスを作成して `oracle.ifs.beans.parsers.Parser` インタフェースを実装することができます。

iFS XML の解析方法

iFS にドキュメントの XML 表示を配置すると、`SimpleXmlParser()` がコールされ、ドキュメント・オブジェクトが作成されます。

`gking.vcf` というドキュメント・インスタンスが XML ファイル `gking.xml` に変換され、エンド・ユーザーが iFS Windows インタフェースを使用しているとします。

1. `gking.xml` などの XML ドキュメントのインスタンスを iFS フォルダ `/ifs/system/vcards` にドラッグしたとします。
2. SMB はファイル拡張子 `.xml` に基づいて、パーサー検索を行います。SMB は、Microsoft Windows 95/98 および Windows NT のクライアントを介して iFS にアクセスするプロトコルです。Oracle iFS からファイルをドラッグして出し入れしたり、iFS で直接ファイルを編集することができます。
3. これは XML ファイルであるため、パーサー検索は一致するものを検出し `SimpleXmlParser()` を起動します。
4. Vcard カスタム・クラスの定義ファイルは iFS に事前に格納されているため、`SimpleXmlParser()` は `gking.xml` を Vcard ドキュメントとして認識します。
5. `SimpleXmlParser()` は `gking.xml` を解析して、`gking` という Vcard オブジェクトを作成します。

ただし、ドキュメント・インスタンス `gking.vcf` が使用されている場合、手順 2 のパーサー検索は、SMB がカスタム・パーサー `VcardParser` を起動する結果になります。

パーサー・アプリケーションの作成

パーサー・アプリケーションを作成するには、次の手順に従います。

1. パーサー・クラスを作成します。
2. パーサーを配置します。
3. パーサーをパーサー・アプリケーションで起動します。
4. ParserCallback を作成します（オプション）。

パーサーの詳細は、iFS Java ドキュメントで次のクラスを参照してください。

```
oracle.ifs.beans.parsers.SimpleXmlParser  
oracle.ifs.beans.parsers.XmlParser  
oracle.ifs.beans.parsers.SimpleTextParser
```

iFS での XML のレンダリング

デフォルトでは、iFS XML のレンダラは XML 属性タグを含めて元のファイルを再構築します。ファイルをダブルクリックすると、iFS はファイルを再構築し、XML エディタでオープンします。

ファイルの再構築も重要ですが、ファイルを使用して様々なことができます。たとえば、XML ベースの保険金支払請求を解析する場合、顧客が Web ベースのセルフサービスのアプリケーションを介してアクセスすると、顧客には元のファイルの一部のみが表示されるようにする必要があります場合があります。さらに、合計、総数、平均などの値をファイル内容から計算し、ファイルに計算結果を追加する場合があります。さらに、ファイル形式を変更して、読みにくい XML ではなく RTF または HTML として表示する場合があります。

これらのすべてのタスクは、iFS レンダラを介して実行できます。XML ファイルの解析済コンテンツは、iFS での動的な再組立てを待機しており、ユーザーまたはアプリケーションに必要な情報の範囲、形式または型を示します。XML ファイルのクラスに新しいレンダラを作成および登録して、iFS によるファイルの表示方法を変更できます。

XML およびビジネス・インテリジェンス

解析済の XML ファイルは、各属性のデータを iFS スキーマの識別可能な個別の列に格納するため、Oracle iFS を使用すると、XML ファイルで事前に定義されているビジネス・インテリジェンスに、データ・マイニング・アプリケーションからアクセスできます。

保険金支払請求例に戻ると、ビジネス・インテリジェンスはそれぞれの請求に格納され、システムに格納されるすべての請求のすべての情報の集計値は、部分合計よりも大きくなります。エージェントが請求ファイルを挿入または更新する状態で、マネージャリアル・タイムで動向を追跡するには、iFS ファイルを使用してファイルを解析し、解析ファイルの内容をデータ・マイニング・ツールへ送ります。

XML ファイルを使用した iFS の構成

iFS を構成するには、XML を作成してファイル・システムをカスタマイズします。たとえば、Oracle iFS SDK の機能によって、ファイルおよびフォルダをサブクラス化できます。XML ベースの発注書をアプリケーションで個別の型のファイルとして識別する場合、iFS サブクラスを定義します。サブクラスの作成は、次のとおり簡単です。

1. サブクラスを定義する iFS 構文に続けて、XML ファイルを記述します。
2. この XML ファイルを iFS にあるフォルダにドラッグ・アンド・ドロップします。

このような種類の構成ファイルによって、アプリケーション配置のために複雑なスクリプトを作成する必要がなくなります。ただし、XML 構成ファイルなどすべての必要ファイルを新しい iFS インスタンスにドラッグ・アンド・ドロップする必要があります。

XML を使用したリッチ・メディア管理用の n 層アーキテクチャの構築 : ArtesiaTech

この章の内容は次のとおりです。

- n 層アーキテクチャ構築の概要
- XML ベースの複数層通信
- 機能の移動 : 論理層と物理層の分離
- XML を使用したオブジェクト指向のメッセージ機能
- XML のメッセージ処理
- XML を使用したスクリプト
- XML ベース・ソフトウェアの品質保証
- XML ベースのデータ送信
- XML 中心のデジタル資産管理
- まとめ

n 層アーキテクチャ構築の概要

幅広いテクノロジーによって、Web がリッチ・メディアへの導入口となり、マルチメディア・コンテンツ管理には、本質的な課題に取り組むためのシステム・アーキテクチャが必要になっています。この章では、XML を使用して、リッチ・メディア管理に対する信頼性が高くスケーラブルで、コスト・パフォーマンスが高いソフトウェア・エンジニアリング・ソリューションを構築するためのヒントおよび方法を説明します。

この章では、価値を持つデジタル・リソースをデジタル資産と呼び、写真およびオーディオやビデオなどのストリーム・メディア・タイプを含むマルチメディア・コンテンツをリッチ・メディアと呼びます。

デジタル資産管理の課題

リッチ・メディア管理には、サイズ、動作およびアクセス方法が多様に変化する無数のデータ型など、多くの課題があります。デジタル資産管理のための企業規模のアプリケーションを作成する場合、最初の課題は、表示特性から独立したコンテンツを管理可能な、非常にスケーラブルなアーキテクチャを提供することです。これによって、コンテンツを異なるシステム間のすべての場所で表現および交換できるようになります。

表示における課題

表示特性の処理では、エンド・ユーザーのパーソナライズを容易にするために、特別なビジュアル的な支援（レンダラ、ビューア、プレーヤなど）を指定する必要があることがよくあります。また、リッチ・メディアをローカルまたは広域ネットワーク上に転送すると、パフォーマンスが低下する場合がありますため、帯域幅と転送の問題もあります。

企業全体に及ぶ複数層のリッチ・メディア管理における分散環境で見られるこのような課題は、XML の使用によってすぐに解決できます。

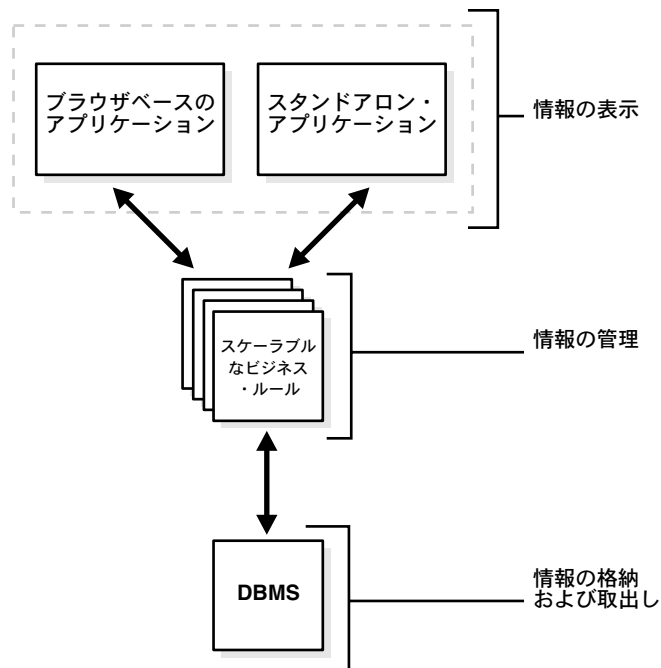
XML ベースの複数層通信

企業規模のデジタル資産管理システムを構築する場合、スケーラビリティ、柔軟性および拡張性という 3 つの基本目標があります。この目標を達成するために、最新のシステム・アーキテクチャは、通常、次の論理的なタスクに分割されています。

- 情報の表示 : ユーザー・インタフェースを構成します。
- 情報の管理 : アプリケーション・サーバーに組み込まれたビジネス・ルールおよび手順によって制御されます。
- 情報の格納 / 取出し : 通常、Oracle8i などのデータベース管理システムに委託されています。

図 15-1 に、この 3 層アーキテクチャを示します。これは、ArtesiaTech の非常にスケーラブルな企業規模のデジタル資産管理システムで、TEAMS と呼ばれます。このシステムは、アーキテクチャ基盤に基づいて設計および実装されています。

図 15-1 デジタル資産管理のための ArtesiaTech TEAMS の 3 層アーキテクチャ



機能の移動：論理層と物理層の分離

論理層と物理層を分離することによって、システムの長期稼働が可能になります。

適切に設計された柔軟性のあるシステム・アーキテクチャでは、3層の論理的な分離によって、基礎となるハードウェアに制約が課せられることはありません。演算論理は、ロード・バランシングのための実行時ベースの機能か、サブシステム間に発生する通信量を最小化するために機能領域が再編成される、システムの拡大に伴って発生する処理であるかにかかわらず、必要に応じて、簡単にクライアント・マシンに移動可能である必要があります。これを機能の移動といいます。

機能の移動とは、論理的なサービスを、そのサービスが最適に実行できるシステムの物理的な領域に移動するプロセスです。サービス間の通信量が多い場合、実際のシステムの配置および使用に固有の構成をすることが理想的ですが、サービスが同じマシンに常駐する方が効果的です。

機能の移動の基本的なコンポーネントは、様々な無数のオペレーティング・プラットフォーム間で、特定のサービスの制御論理を迅速に再配置する機能です。Javaによってこれを実現できるプラットフォームもありますが、実装者がその方法で処理できるのは、プログラミング言語固有の枠内のみです。

オブジェクト指向のシステム動作は、最終的にはオブジェクト間で渡されたメッセージによって制御されます。機能移動の目的を考える場合、オブジェクトを実装する方法は、メッセージをクロス・プラットフォーム互換にレンダリングするためにシリアライズ化する方法、およびメッセージを最終的に処理する方法に比べるとあまり重要ではありません。XMLおよびその派生的な標準では、両方が効果的に実行される適切なフレームワークが提供されます。

XML を使用したオブジェクト指向のメッセージ機能

メッセージとは、イベント、要求、応答などを説明する特別にフォーマットされたデータであり、メッセージ機能とは、システムの機能領域間でメッセージを交換するプロセスのことです。XML ベースのメッセージ機能によって、オペレーティング・システム、言語またはコンパイラに関係なく、アプリケーションまたは固有のサービスが異なるプログラミング環境で通信できます。これは、それぞれの環境のプロセスでは、共通のメッセージ形式およびプロトコルが理解されるだけでよいからです。最も基本的なレベルでは、それぞれの環境が API を介して XML パーサーにアクセスできればよく、Windows、Mac OS、Solaris、Linux などのすべての主なオペレーティング・システム上で、C、C++、Java などのよく知られたプログラミング言語によって簡単に行えます。

メッセージは、本質的には単なるデータ構造であり、様々な方法でシリアライズ化できます。たとえば、5つの整数の配列である次の C++ データ構造を考えてみます。

```
int array[5] = {-9,0,-2,1,0};
```

表 15-1 に、メッセージの論理データ構造を示します。

表 15-1 XML を使用したオブジェクト指向のメッセージ機能：メッセージの論理データ構造

配列 サイズ	索引 0 値	索引 1 値	索引 2 値	索引 3 値	索引 4 値
5	-9	0	-2	1	0

ただし、物理的には 8 ビット・マシンでの実際のビットは、表 15-2 のように格納されます。

表 15-2 XML を使用したオブジェクト指向のメッセージ機能：メッセージの物理データ構造

配列 サイズ	索引 0 ビット	索引 1 ビット	索引 2 ビット	索引 3 ビット	索引 4 値
00000101	11110111	00000000	11111110	00000001	00000000

ここで、マシン A にあるオブジェクト 1 からマシン B にあるオブジェクト 2 へのメッセージの一部として、この配列が渡される場合、このデータの送信先オブジェクトのビューが送信元オブジェクトの解析と同一であるとは限らないため、特に注意して同一かどうかを確認する必要があります。

たとえば、マシン 2 の整数のデフォルトが定義されていない場合があります。そのような場合のデータ構造の論理ビューを表 15-3 に示します。

表 15-3 XML を使用したオブジェクト指向のメッセージ機能：デフォルト整数が定義されていないメッセージの論理データ構造

配列 サイズ	索引 0 値	索引 1 値	索引 2 値	索引 3 値	索引 4 値
5	247	0	254	1	0

さらに、プラットフォームによっては、整数は 16 ビット、32 ビットなどの複数バイトで、近い将来には 64 ビットになります。これにビッグエンディアン / リトルエンディアンの不確実性を考えると、言語、コンパイラまたはプラットフォーム固有のメッセージのシリアライズ化は機能移動を促進するソリューションではないことは明らかです。

XML の同一配列のシリアル化は、非常に単純です。

```
<array size='5'>
  <integer value='-9'>/>
  <integer value='0'>/>
  <integer value='-2'>/>
  <integer value='1'>/>
  <integer value='0'>/>
</array>
```

メッセージ機能への XML 使用の必要性

メッセージ機能に XML を使用する必要があるのでしょうか。従来、プラットフォーム間の転送用データ形式の定義は、CORBA のようなミドルウェア・ソリューションを使用したインタフェース定義言語（Interface Definition Language: IDL）によって指定されていました。自己記述型で人間が理解できるデータによるわずかなオーバーヘッドを伴う XML と異なり、IDL はマシンが理解できる一時データ・パケットを定義します。

ただし、XML および IDL ではオブジェクト状態を保存できますが、シリアル化されたコンポーネントはクラス定義への変更によって実行できない場合があるため、人間が理解できて編集可能なオブジェクトの永続状態形式が求められます。

- IDL を使用すると、シリアル化されたオブジェクトはクラス・メソッドを介してアクセスされます。
- Java プログラムまたはデータベース・スキーマと同様に自己記述型の XML を使用すると、ソフトウェアが正常に実行されない場合、オブジェクト・データを簡単に調べられます。

XML または IDL の使用の選択

IDL か XML のどちらを使用するかについては、意見が分かれます。XML を使用して分散システムのコンポーネントをバインドするのは、高価なうえ無意味だという場合もあります。ただし、XML がオブジェクトの永続性をエンコーディングするために、プラットフォームに中立的でプログラム言語に依存せず、自己記述型のメカニズムを提供することは重要です。

すべてのアプリケーション・プログラムを、第 3 世代または第 4 世代言語ではなく、アセンブリ言語を使用して作成することはほとんどありません。同様に、オブジェクト指向のメッセージ機能形式として、IDL がすべての場合に XML に変更される必要もありません。プロファイラを使用すると、実装にかかる時間 / コストの比率、メンテナンスしやすさ、パフォーマンスなどのソフトウェア開発者が直面する問題に基づいて、それぞれを適切に判断できます。

ユーザー・インタフェース戦略の一部としての XML のシリアライズ化

Artesia Technologies 社のソフトウェア開発者は、Java Swing ライブラリに提供されたオブジェクトのシリアライズ化のかわりに、ユーザー・インタフェース配置戦略の一部として XML をシリアライズ化します。従来、新バージョンは旧バージョンが生成したシリアライズ化とは互換性がありませんでした。GUI コンポーネントを XML にシリアライズ化する場合、この問題はなくなりました。

オブジェクトの XML シリアライズ化に対する標準には、次のものがあります。

- Bean マークアップ言語 (BML)
- XML BeanMaker
- XML-CORBA Link (XORBA)
- Koala オブジェクト・マークアップ言語 (KOML)
- Coins (JavaBeans の XML ベース代替)

Artesia Technologies 社の XML Persistent Interchange Syntax for Assets (PISA) などの製品固有の形式の他に、RPC、RMI、HTTP などのプロトコルを併用しても簡単に転送できます。

XML のメッセージ処理

XML でのオブジェクト指向メッセージのシリアル化を正当化するには、最終的にコストを認識する必要があります。ソフトウェア・エンジニアリングの分野では、実際のコード開発にかかるコストは、総コストの一部にすぎないことが認識されてきています。つまり、コードのメンテナンスがコストの大部分となります。したがって、企業は、専用の社内ソリューションを作成するのではなく、できるだけ標準に準拠したソフトウェアを統合用に用意する必要があります。

たとえば、N 個のデジタル資産を表すメッセージのデータ構造を考えてみます。それぞれにはメタデータおよびコンテンツが含まれています。これは、ASCII ベースの専用形式では次のように表されます。

```
[ASSET/METADATA]
NAME=Daytona 500
DATE=February, 2000
```

```
[ASSET/CONTENT]
FILE=daytona.jpg
```

メリット

この方法には、明らかに多くのデメリットがあります。最終的に、データ形式の処理には、2 種類の検証が必要です。

- 構造
- データ型

構造検証ルールは、データ・メンバーの順序およびカーディナリティを含むものを指定します。コンテンツの検証には、指定パラメータとの対応を保証するために検証対象フィールドが必要です。

- 対象は日付フィールドかどうか。日付フィールドである場合、指定された日付は有効かどうか（2 月 31 日は無効）。
- 数値フィールドかどうか。数値フィールドである場合、規定制限内かどうか。
- フィールドは可変文字データかどうか。可変文字データである場合、長さに上限はあるかどうか。
- フィールドは列挙型であるか。列挙型の場合、有効値は何か。

フィールドの始まりと終わりはどこか、「フィールド・グループ化」したものの1つのセクションの始まりと終わりはどこか、抑制の定義は何か、空白が重要な場所はどこで、各行の終わりをどのように処理するか、データ解析の場合、何がリカバリ可能なエラーを表すかなどは、さらに基本的な問題です。前述の ASCII ベースの例では、動作指定の責任はファイル形式設計者にあります。XML などの業界標準に準拠していれば、標準がこれらの質問の答えとなります。回答された基本的な設計決定事項に基づいて、ソフトウェア開発者はソリューションの実装に取り組めます。

多くの XML パーサーが使用可能です。逐次またはランダム・アクセス・モードで XML を処理する現行の API は、それぞれ SAX (Simple API for XML) および DOM (文書ドキュメント・モデル) です。SAX はイベント駆動インタフェースで、発生する可能性があるイベントの制御論理をプログラマーが指定します。XML パーサーはイベント状態を検知すると、SAX が制御をイベント・ハンドラに渡すことで特定イベントの規定コードを起動します。

XML データの構造的な妥当性を保証する仕様が DTD です。これは、SGML から継承されたユーティリティで、実際の XML 構文を使用して、構造的な妥当性基準を記述する様々なスキーマ提案です。DTD による検証は、すべての妥当性検証 XML パーサーが行います。スキーマ・ベースの検証は、特定のスキーマ仕様の特性をサポートするソース・ライブラリを介して使用できます。

XML Schema (ワーキング・ドラフト) などの、W3C が勧告したスキーマは、構造検証およびデータ型検証の両方を行います。最終的にすべての戦略では、検証ルールを定義するための適切に定義された手順を提供します。これらは、実際のランタイム・コード内で、または個別に列挙できます。ルールは自己記述型であるため、ソフトウェア開発者または統合者が簡単に変更できます。正しく定義された方法であるため、通常、すぐにツールを使用することができます。このため何行にもわたるコードを書いたり、メンテナンスする必要がなく、ソフトウェア開発の全体でのコスト削減にもつながります。

XML を使用したスクリプト

最終的に機能の移動に必要なことは、データを記述および処理するための手順です。実際の論理はコンポーネントまたはオブジェクト内でカプセル化され、移動可能である必要があります。ただし、論理コントローラがデータ駆動の場合、ランタイム・コード自体は移動可能である必要はありません。XML によって、システム動作を制御するために使用するデータの処理方法が提供されます。

デジタル資産管理の基本について考えてみると、資産上で実行できる特別な CRUD (作成、読み込み、更新、削除) 操作が関係してきます。それぞれが固有の動作で構成されます。

表 15-4 に、CRUD 操作を分類するマトリクスを示します。

表 15-4 XML スクリプト : CRUD (作成、読み込み、更新、削除) 操作

作成			読み込み			更新		削除
インポート	バージョン	チェック イン	検索	エクスポート	チェック アウト	置換	削除	ページ

オブジェクト指向の観点では、多くの操作は他の操作を特殊化したものです。たとえば、バージョンはインポート（作成）を特殊化したものであり、新しいデジタル資産が作成され、レンダリング前のメタデータまたはコンテンツ（またはその両方）が引き継がれます。チェックイン操作は、バージョンを特殊化したものであり、ロックを解放する手順が追加されています。この操作ではデジタル資産がチェックアウトされると想定するため、チェックインの前にロックされます。

XML Persistent Interchange for Assets (PISA)

Artesia Technologies 社の XML Persistent Interchange Syntax for Assets (PISA) は、N 個のデジタル資産で構成されるデータ構造を定義します。実際には、N の上限は何千にもなります。GB サイズの XML データ・ストリームでは、膨大なメモリーが必要となるため、DOM を使用できません。したがって、Artesia Technologies 社の DAM システム TEAM は、資産ごとにトランザクション・レベルを持つ SAX を使用しています。

SAX によって PISA ファイル内で表現されるデジタル資産のコンポーネントを集計するには、デフォルトのドキュメント・ハンドラがサブクラス化され、PISA 形式固有にエンコーディングされる必要があります。前述の CRUD 操作およびトランザクション・レベルは資産ごとであるため、一般的な *AssetProcessor* クラスは、資産に関するすべての操作が発生するところで定義されます。したがって、すべての操作は *AssetProcessor* の定義によって異なります。資産の XML 終了タグが検出されると、ドキュメント・ハンドラは資産を表すメモリー内のデータ構造の構築を完了し、その後、その資産の *AssetProcessor* の処理方法にコールできるようになります。

どの *AssetProcessor* が適用されるかは、ドキュメント・ハンドラのコンストラクタに渡される実際の型によって異なります。これは通常、資産レベルの操作を起動する固有のコールバックによって制御されています。したがって、単一の XML ドキュメント・ハンドラを作成する場合、すべての資産操作がバルク操作として、デジタル資産管理システムで使用可能です。この方法はデザイン・パターンと呼ばれます。XML 処理の設計、実装およびメンテナンスが簡素化されるため、単一のデータ形式 PISA およびドキュメント・ハンドラのみが必要となり、ソフトウェア全体におけるコストを大幅に削減できます。

固有のコールバックを使用した *AssetProcessor* の選択には、1 つのデメリットがあります。それは、プロセッサの型が処理時間中、固定されてしまうことです。ただし、特定の資産のキュー処理には有効で、それぞれは資産によって異なる操作をします。単一のバッチ・ジョブとしてこのキューを処理するためには、*AssetProcessor* を変更してイベントを起動できるように、XML ファイルをエンコーディングする必要があります。これを実現するために、Artesia Technologies 社は、XML の処理命令を使用した埋込みスクリプト言語を設計しました。この処理命令では、現在アクティブな *AssetProcessor* など、システム動作の制御を構文的に列挙するように要求されます。

ターゲットに続く処理命令の内部データには、最小限の構文制限があります。したがって、データは要素属性と同様の構文で次のようにカプセル化できます。

```
<?PITarget attr1='some value' attr2='another value'?>
```

SAX の使用

処理命令を処理する SAX イベントが、命令をターゲット名と残りのデータに適切に分割するため、要素属性と同様の構文の解析には、最小限のドキュメント・ハンドラを作成する必要があります。次のように、処理命令は要素に書き直され、XML パーサーに送られるためです。

```
<PITarget attr1='some value' attr2='another value' />
```

開始要素を処理する SAX イベントは、属性名によって索引付けできる属性値のリストを返します。したがって、処理命令は一貫した方法で簡単に処理されるスクリプト言語として使用でき、XML パーサーをアプリケーション・フレームワークとして使用します。このため、XML を使用して、プログラマ・レベルの API がこれまでに開始したすべてのサービスを提供可能な、データ駆動のイベントベース・システムを簡単に構築できます。

層内の通信

デジタル資産管理システムは、永続ストレージ・レイヤーからのデータの出入れが可能な中間構築層で、最終的にアプリケーション・サービスを提供します。このレイヤーは通常、データベース管理またはオペレーティング・システム（またはその両方）専用で、固有なものです。XML をアプリケーション・フレームワークとして使用することで、新しいプラットフォームが作成されても迅速に再配置できる DAM システムを構築できます。これが可能なのは、XML が SGML 系統から論理データおよび物理データの表示を分離する概念を継承しているためです。

論理的に、デジタル資産はメタデータおよびそのコンテンツで構成されます。ただし、物理的には、これら 2 つのコンポーネントは何種類もの方法で格納できます。ここで、XML のエンティティの概念が有効になります。デジタル資産の概念は、次のとおり分類できます。

```
<!DOCTYPE teams:digital-asset [  
<!ENTITY metadata PUBLIC "-//TEAMS//TEXT digital asset metadata//EN" "metadata.ent">  
<!ENTITY content PUBLIC "-//TEAMS//TEXT digital asset content//EN" "content.ent" >  
>  
<teams:digital-asset>  
  &metadata;  
  &content;  
</teams:digital-asset>
```

XML パーサーがエンティティ参照を検出すると、エンティティの宣言を調べ、実際のエンティティ・データをどこから引き出すかを決定します。宣言でのエンティティ参照は、エンティティ・マネージャによって決定されます。エンティティ管理は実際のパーサーから独立しているという概念も XML によって受け継がれます。したがって、異なるエンティティ管理の置換えによって、基礎となる多くの物理記憶スキーマがサポートされます。

デジタル資産のメタデータの最も基本的な永続ストレージ・メカニズムは、ファイル・システムへの XML シリアル化です。したがって、エンティティ・マネージャはメタデータの PUBLIC 識別子を固有のホスト、パスおよびファイル名へマッピングする必要があるのみです。

同じデータを、リレーショナル・データベースまたはオブジェクト指向データベース、SGML/XML コンポーネント・マネージャおよび独占データ型のスキーマ内に格納できます。ある特定の実装は、すべての状況においてコスト、パフォーマンスなどの点で最適であるとは限らないため、DAM システムは永続ストレージ・システムを簡略化するために簡単に構成できる必要があります。

考えられる格納方法からデータを取り出せるようにエンティティ・マネージャを作成できるため、XML は永続ストレージとビジネス・ロジック間の間接的なレベルとして機能します。

エンティティが変換され、永続ストレージからデータを引き出すと、情報は再び XML としてシリアル化化されて中間層アプリケーションのサービスへ渡されて処理されます。これによって、データをウェアハウスに格納する有名および無名の方法から独立したデータ処理のメカニズムが提供されます。

Web ベース・サービスのトランザクション状態管理

デジタル資産管理サービスに Web ベースでアクセスする場合、確実に処理されるデータ形式への多数のウィザード（ウィザードはオプションの場合もあります）からのユーザー入力を維持および集計する問題が発生します。エンティティは実際のまたは空のデータ・ストリームへ変換されるため、XML は骨組みとして配置されます。そこでは、ウィザードの数にかかわらず、送られたデータが個別の物理記憶単位として維持され、その後 XML パーサーによって集計されます。

たとえば、HTML ページで表示される 3 つのウィザードについて考えてみます。ユーザー入力は HTML 形式データとして順次提供され、ユーザーが HTML ページ（ウィザード）から次のページへ移動すると、処理サプレットへ送られます。1 ページから次のページへコンテンツを保持するためには様々な方法があります。データは Cookie 内に永続的に格納されるか、または非表示型データとしてページ間で渡されますが、XML はさらに優れたソリューションを提供します。次の XML ベース・トランザクションの骨組みについて考えてみます。

```
<transaction>
  &wizard1;
  &wizard2;
  &wizard3;
</transaction>
```

各ウィザードのデータはエンティティ参照によって表され、通常、単一ファイルのように簡単に格納しているシステムから変換されます。ウィザード横断の順序にかかわらず、あるページから移動するたびに、XML データ・ストリームとしてエンコーディングされたトランザクションが生成されます。このストリームが送られる準備は、DTD が割り当てたルールによって簡単に制御されます。データは HTTP セッションの一部としてページ間を移動する必要がないため、トランザクション状態は効率的に維持および更新されます。

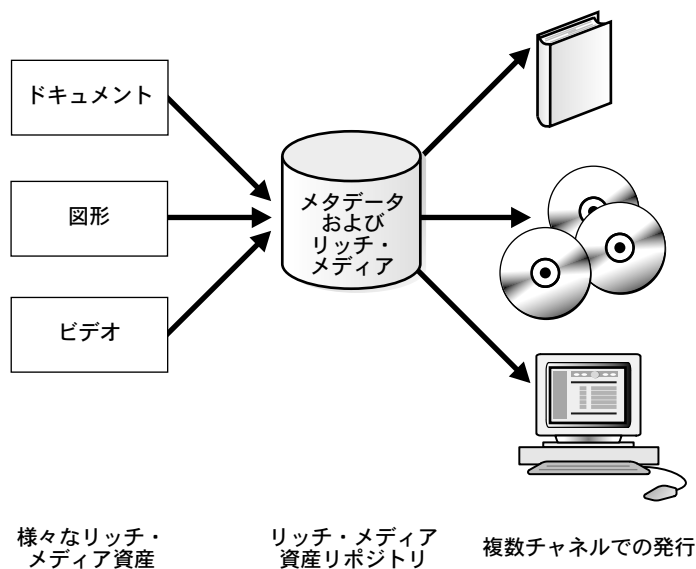
XML ベース・ソフトウェアの品質保証

規定または予期しない入力が行われたときでも製品が正常に動作するかどうか、そのソフトウェアの品質を決定します。品質レベルまたは欠陥の割合の判断は、従来非常に面倒で人間の力に頼っていました。最終的には、限界があるマシンと考えられているソフトウェアに処理が送られ、初期状態および考えられる一連の入力を指定することによって、入力結果としての新しい状態が確認されます。XML を使用して実行時にプログラムで状態を切り替えられる場合、DTD などの正式な文法によって、プログラムが規定の仕様に準拠していることが保証されます。

XML ベースのデータ送信

高品質で完全にスケーラブルな企業デジタル資産管理システムを構築するために有益な XML 標準も、リッチ・メディア・コンテンツおよび対応付けられたメタデータの様々な販路への配布に適用できます。デジタル資産はリポジトリまたはディレクトリからレンダリングされるため、XML によって再びメタデータとコンテンツがバインドされます。その後は、変換プログラムを適用して、メタデータを拡張 / フィルタしてコンテンツを別の型に変換できます。図 15-2 に示すとおり、その後、ページ・レイアウトのプログラム、CD マスタリング・ツール、Web サイト管理ソフトウェアなどの個別のメディアに対応するために、デジタル資産が集められます。

図 15-2 XML ベースのデータ送信



XML 中心のデジタル資産管理

静的テキストやイメージなど、非ストリームで自己完結型のコンテンツの集計作業は、これまで容易に理解されてきました。複雑なドキュメントのレンダリングが可能なページ・レイアウトのプログラムおよび最新のワード・プロセッサはたくさんありますが、オーディオおよびビデオの放送で使用するストリーム・メディアの種類には、さらなる問題点があります。同期化されたリッチ・メディアの表示に独立したマルチメディア・オブジェクトを統合すると、メディア・オブジェクト内で円滑に動作する機能の他に、時間の（時間ベースの）動作およびレイアウト表示を指定する機能が必要となります。

エンド・ユーザーのパーソナライズを容易にするデジタル資産集計

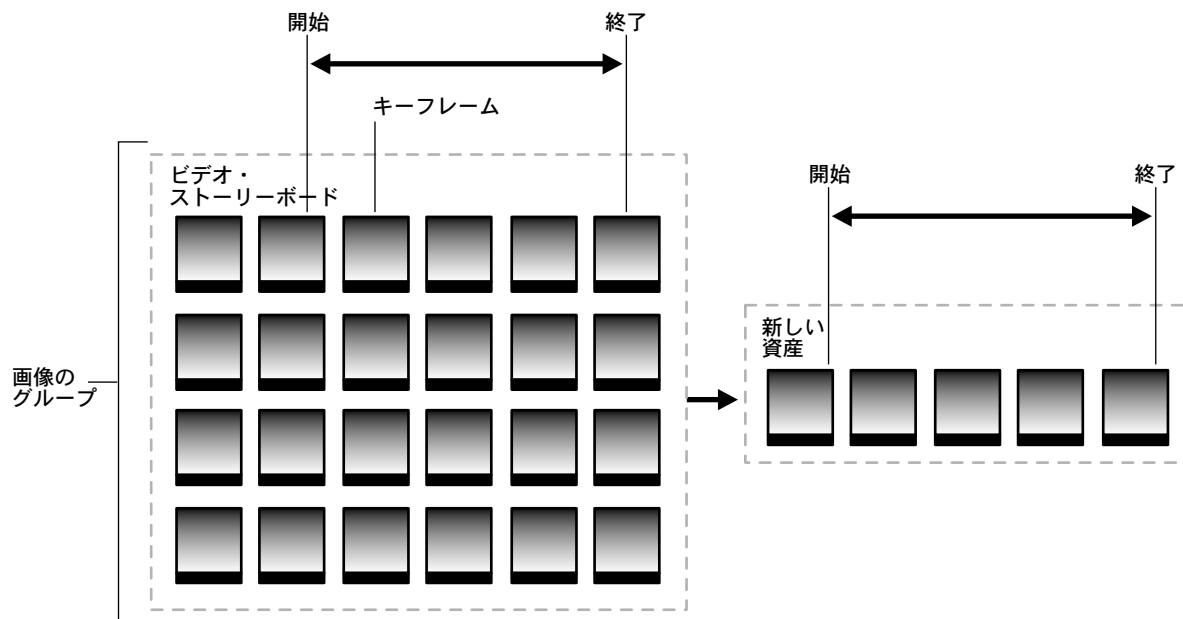
SMIL（同期化マルチメディア統合言語）は XML アプリケーションであり、統合および同期化されたリッチ・メディアの表示としてマルチメディア・オブジェクト（オーディオ、ビデオ、静止画像）をバインドする際の固有の問題に対処するように設計されました。

メディア・オブジェクトのタイミングは、SMIL を使用してレンダリング調整され、それぞれのコンテンツ・オブジェクトが URL によって識別されるため、表示のリッチ・メディア・コンポーネントは多数のソースからストリームできます。

図 15-3 に、SMIL を使用して、ストーリーボードのビデオ・クリップの再生リストを実装する方法を示します。このビデオ・クリップによって、ユーザーは多数のビデオ・セグメントを選択し、その開始点および終了点を調整して、シームレスに表示できます。

これは、アプリケーションに、選択されたビデオ・クリップの連続的な再生を実現する SMIL ファイルを生成させることによって実現できます。ユーザーは、選択、カットおよびビデオの流れに満足するまで簡単に表示の微調整ができます。さらに、編集ソフトウェアを使用してフレーム・トランジションを微調整する前に、簡単にデジタル・ビデオを大まかに編集することもできます。

図 15-3 SMIL を使用したビデオ・ストーリーボードからのビデオ・クリップ選択方法



使用するビジュアル支援（レンダラ、ビューア、プレーヤなど）のユーザー設定項目も管理する必要があります。XML は、パーソナリゼーションを可能にするには非常に適切です。

このような設定項目の情報量が比較的小さい場合、エンド・ユーザーのパーソナリゼーションを効果的に作成および問い合わせるために、通常、DOM API が配置されます。

設定項目は XML でエンコーディングされるため、基礎となる永続ストレージ層にかかわらず、透過的に管理されシステム内で交換されます。SMIL などの XML ベースのリッチ・メディア集計標準が使用される場合、最小限の演算のオーバーヘッドによるパーソナライゼーションのシリアル化がこのような固有のユーザー設定項目に簡単に埋め込まれます。

帯域幅の問題に対処するデジタル資産集計

遍在的なモバイル性のための標準が多数あります。デジタル資産管理に関連する2つの標準は、次のとおりです。

- WML（無線マークアップ言語）はXMLアプリケーションであり、Web ページのテキストを携帯電話、ポケットベル、PDA およびその他の無線モバイル・アクセス・デバイスにレンダリングする場合に効果的です。WML は、無線アプリケーション・プロトコル（WAP）のサブセットです。WAP は、無線通信ネットワーク上のアプリケーション開発の業界標準を定義するための取り組みです。WML は、限られた表示およびユーザー入力機能、限られた演算リソース、狭帯域ネットワーク接続などの無線デバイスで直面する制約に対処するようになっています。デジタル資産がシステムから発行されると、適切なトランスフォーマを起動してリッチ・メディアのリポジトリ内に格納されたドキュメントを WML に変換できます。
- SyncML も XML アプリケーションであり、異なるネットワークのデバイスおよびアプリケーションを同期化するために、遍在的なデータ形式を提供しています。SyncML を使用すると、電子メール、カレンダーおよび連絡情報などの個人的なデータは同期化され、情報の格納場所にかかわらず一貫性およびアクセス性が保証されます。たとえば、SyncML を使用して、PDA 内に格納された電子メールを PC ベースのメール・クライアントへ転送できます。SyncML は帯域幅が最小化され、つながりにくい接続や待機時間の長いネットワークなどの問題も解消します。

まとめ

XML は業界標準に準拠した商業ソフトウェア開発に、最も推奨されるテクノロジーの1つです。XML は実社会に則したリッチ・メディア管理ソフトウェア・アプリケーションを設計する手間を大幅に削減するのみでなく、これまでのあらゆるデータ交換標準の中で最高の相互運用性を発揮します。

互いに深く影響し合う放送、娯楽およびコンピュータ産業にリッチ・メディアが登場した中で、XML は、互換性のないメディア / データ形式が混乱する問題に対する「データの接着剤」として位置付けられています。

近い将来、デジタル・メディアの非互換性および多様性が高くなるほど、データ交換プロトコルとして XML を使用するシステムを実装したことによる投資収益（ROI）を実感できるでしょう。

無線の世界と同様に有線の世界からも高く評価され、柔軟なモジュール・システム開発の全体を支援する輝かしい実績を考えると、XML はリッチ・メディアのすばらしい新世界で支持されるオープン・テクノロジーの1つといえます。

用語集

API

「アプリケーション・プログラム・インタフェース（Application Program Interface: API）」を参照。

B to B（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。

B to C（Business-to-Consumer: B2C）

商品販売およびサービス提供における企業とコンシューマ間のコミュニケーションを示す用語。

BC4J

Business Components for Java。

BFILE

オペレーティング・システム内に常駐するデータベース表領域の外に存在する外部バイナリ・ファイル。BFILE はデータベース・セマンティクスから参照され、外部 LOB ともいう。

BLOB

「バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）」を参照。

CDATA

「文字データ（Character Data: CDATA）」を参照。

CDF

チャンネル定義形式。インターネット上のチャンネルに関する情報を交換する方法を提供する。

CGI

「[共通ゲートウェイ・インタフェース \(Common Gateway Interface: CGI\)](#)」を参照。

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 Oracle Runtime Environment (CORE)

C で作成されたファンクションのライブラリ。これによって開発者は、事実上すべてのプラットフォームおよびオペレーティング・システムに簡単に移植できるコードを作成できる。

CORBA

「[共通オブジェクト要求ブローカ API \(Common Object Request Broker API: CORBA\)](#)」を参照。

CSS

カスケーディング・スタイルシート。

DOCTYPE

XML ドキュメント内に 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。

DOM

「[ドキュメント・オブジェクト・モデル \(Document Object Model: DOM\)](#)」を参照。

DTD

「[文書型定義 \(Document Type Definition: DTD\)](#)」を参照。

EDI

電子データ交換。

Enterprise JavaBean (EJB)

サーバー上の JVM 内で実行する独立プログラム・モジュール。CORBA によって EJB のインフラストラクチャが提供され、コンテナ・レイヤーによって、サポートされたサーバーにセキュリティ、トランザクション・サポートおよびその他の共通機能が提供される。

eXtensible Stylesheet Language Formatting Object (XSLFO)

書式設定セマンティクスを指定するための XML 用語を定義する、W3C の標準仕様。

eXtensible Stylesheet Language Transformation (XSLT)

XSLT とも書く。XML ドキュメントを別のドキュメントに変換する変換言語を定義する、W3C の XSL 標準仕様。

HTML

「[ハイパー・テキスト・マークアップ言語 \(Hypertext Markup Language: HTML\)](#)」を参照。

HTTP

「[ハイパー・テキスト転送プロトコル \(Hypertext Transport Protocol: HTTP\)](#)」を参照。

IDE

「[統合開発環境 \(Integrated Development Environment: IDE\)](#)」を参照。

iFS

「[Internet File System \(iFS\)](#)」を参照。

interMedia

複合データ型のコレクションおよびこのコレクションの Oracle8i 内でのアクセスを示す用語。これには、テキスト、ビデオ、時系列および空間データ型が含まれる。

Internet File System (iFS)

Oracle8i データベース内または中間層上で実行する、Oracle のファイル・システムおよび Java ベースの開発環境。単一のデータベース・リポジトリに複数の型のドキュメントを作成、格納および管理する方法を提供する。

Internet Inter-ORB Protocol (IIOP)

インターネットなどの TCP/IP ネットワーク上で、メッセージを交換するために CORBA が使用するプロトコル。

Java

Sun Microsystems 社によって開発およびメンテナンスされた高水準のプログラミング言語。Java では、アプリケーションが JVM という仮想マシン内で実行する。JVM は、オペレーティング・システムに対するすべてのインタフェースの役割を担う。開発者は、このアーキテクチャによって、JVM を搭載するすべてのオペレーティング・システムまたはプラットフォームで実行する Java アプリケーションおよびアプレットを開発できる。

Java 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 Developer's Kit (JDK)

Java 開発環境を確立する Java バージョン用の、Java クラス、ランタイム、コンパイラ、デバッガおよびソース・コードのコレクション。JDK はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

Java Runtime Environment (JRE)

プラットフォーム上で Java 仮想マシンを構成する、コンパイル済クラスのコレクション。JRE はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

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。サーバー側の同等のモジュールは、Enterprise JavaBean (EJB) という。「[Enterprise JavaBean \(EJB\)](#)」を参照。

JavaServer Page (JSP)

Web ページへの単純なプログラム・インタフェースを可能にする、サーブレットの拡張機能。JSP は、特殊タグ、および Web またはアプリケーション・サーバーで実行される埋込み Java コードを含む HTML ページであり、HTML ページに動的機能を提供する。JSP は、サーバーの JVM で最初に要求および実行されるときに、サーブレットにコンパイルされる。

Java 仮想マシン (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、Web 上、OAS などのアプリケーション・サーバー上、または Oracle8i などのデータベース・サーバー内で実行できる。

JDBC

「[Java Database Connectivity \(JDBC\)](#)」を参照。

JDeveloper

アプリケーション、アプレットおよびサーブレットの開発を可能にする Oracle の Java IDE。エディタ、コンパイラ、デバッガ、構文チェッカ、ヘルプ・システムなどを含む。バージョン 3.1 の JDeveloper は、エディタで XML がサポートされるとともに、簡単に使用できるように統合された Oracle XDK for Java を搭載して、XML ベースの開発をサポートするように拡張されている。

JDK

「[Java Developer's Kit \(JDK\)](#)」を参照。

JVM

「[Java 仮想マシン \(Java virtual machine: JVM\)](#)」を参照。

LAN

「[ローカル・エリア・ネットワーク \(Local Area Network: LAN\)](#)」を参照。

LOB

「[ラージ・オブジェクト \(Large Object: LOB\)](#)」を参照。

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 つのクライアント層とともに、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。

OAS

「[Oracle Applilcation Server \(OAS\)](#)」を参照。

OASIS

「[Organization for the Advancement of Structured Information \(OASIS\)](#)」を参照。

OE

Oracle Exchange。

OIS

「[Oracle Integration Server \(OIS\)](#)」を参照。

Oracle Applilcation Server (OAS)

Oracle アプリケーション・サーバー。オープン標準フレームワーク内で高パフォーマンスの N 層トランザクション指向 Web アプリケーションを構築、配置および管理するために必要な、すべての主要なサービスおよび機能を統合する。

Oracle Integration Server (OIS)

アプリケーション統合のためのメッセージング・ハブとして機能する Oracle のサーバー製品。OIS には、AQ および Oracle Workflow を搭載した 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ーフェースが含まれる。

ORACLE_HOME

アプリケーションで使用するために、Oracle データベースのインストール場所を識別するオペレーティング・システムの環境変数。

Oracle8i JVM

Oracle8i データベースのメモリー領域内で実行する Java 仮想マシン。JVM は、Oracle 8i リリース 8.1.5 では Java 1.1 準拠であったが、リリース 8.1.6 では Java 1.2 準拠である。

ORB

「[オブジェクト要求ブローカ \(Object Request Broker: ORB\)](#)」を参照。

Organization for the Advancement of Structured Information (OASIS)

会議、セミナー、展示会およびその他の教育イベントを通じて、パブリック情報標準の普及促進を目的として設立されたメンバーの組織。XML は、SGML と同様に、OASIS が活発に普及を促進している標準である。

PCDATA

「[解析対象文字データ \(Parsed Character Data: PCDATA\)](#)」を参照。

PDA

Palm Pilot などのパーソナル・デジタル・アシスタント。

PL/SQL

データベース内で実行できるプログラムを作成するために SQL を拡張した、Oracle プロシージャ型言語。

PUBLIC

後に続く参照の、インターネット上の場所を指定する用語。

RDF

リソース記述フレームワーク。

SAX

「[Simple API for XML \(SAX\)](#)」を参照。

SGML

「[標準汎用マークアップ言語 \(Standard Generalized Markup Language: SGML\)](#)」を参照。

Simple API for XML (SAX)

XML パーサーによって提供され、イベント駆動型のアプリケーションによって使用される XML 標準インタフェース。

SQL

「[構造化問合せ言語 \(Structured Query Language: SQL\)](#)」を参照。

SSI

「[サーバー側インクルード \(Server-Side Include: SSI\)](#)」を参照。

SSL

「[セキュア・ソケット・レイヤー \(Secure Sockets Layer: SSL\)](#)」を参照。

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

「[トランスミッション・コントロール・プロトコル / インターネット・プロトコル \(Transmission Control Protocol/Internet Protocol: TCP/IP\)](#)」を参照。

URI

「[ユニフォーム・リソース識別子 \(Uniform Resource Identifier: URI\)](#)」を参照。

URL

「[ユニフォーム・リソース・ロケータ \(Uniform Resource Locator: URL\)](#)」を参照。

W3C

「[World Wide Web Consortium \(W3C\)](#)」を参照。

WAN

「[ワイド・エリア・ネットワーク \(Wide Area Network: WAN\)](#)」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。

World Wide Web Consortium (W3C)

1994 年に設立された、Web の標準を確立するための国際的な産業組合。W3C のサイトは、www.w3c.org である。

XLink

XML ドキュメント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

「拡張可能マークアップ言語 (eXtensible Markup Language: XML)」を参照。

XML 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 ドキュメントを作成するために使用できる一連のクラスである。

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、XSL Processor、XML Class Generator、Transviewer Beans および XSQL Servlet が含まれる。

XML Query

W3C が取り組む、XML ドキュメントを問い合わせるための言語および構文の標準。

XML Schema

W3C が取り組む、XML ドキュメント内で単純なデータ型および複合構造を表すための標準。データ型の定義や妥当性など、現在 DTD で不足している領域に取り組んでいる。Oracle XML Schema Processor は、オンライン取引などの E-Business アプリケーションで使われる、XML ドキュメントおよびデータの妥当性を自動的に確認する。XML Schema は、XML ドキュメントに単純および複雑なデータ型を追加し、DTD の機能を XML Schema 定義の XML ドキュメントに置き換える。

XML Transviewer Beans

XDK for Java に含まれる Oracle XML Java Beans を示す Oracle 用語。これらの Bean には、XML Source View Bean、Tree View Bean、DOMParser Bean、Transformer Bean および TransViewer Bean が含まれる。

XML パーサー (XML parser)

XML で、XML ドキュメントを入力として受け入れ、ドキュメントが整形形式であり、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。

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 ドキュメント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 ドキュメント・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 ドキュメント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

(W3C) の XML 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 ドキュメントを別の XML ドキュメントに変換し、XSL Formatting Objects は XML ドキュメントの表示を指定する。XSL は、スタイルシートを表す言語である。XSL は、次の 2 つで構成される。

- XML ドキュメントを変換するための言語 (XSLT)
- 書式設定セマンティクスを指定するための XML ボキャブラリ (XSLFO)

XSL スタイルシートは、書式設定用ボキャブラリを使用する XML ドキュメントへのクラスの実体の変換方法を記述して、XML ドキュメントのクラス表示を指定する。

XSL

「[拡張可能スタイルシート言語 \(eXtensible Stylesheet Language: XSL\)](#)」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 ドキュメントを生成し、XML スタイルシートを使用して、サーバー内のドキュメントを変換する (オプション) 機能を提供する。

アプリケーション・サーバー (Application Server)

アプリケーションおよびその環境をホストするために設計されたサーバーであり、サーバー・アプリケーションの実行を許可する。代表的な例は OAS で、リモート・クライアントがインタフェースを制御する場合に、Java、C、C++ および PL/SQL アプリケーションをホストできる。「[Oracle Application Server \(OAS\)](#)」を参照。

アプリケーション・プログラム・インタフェース (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。

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。

オブジェクト要求ブローカ (Object Request Broker: ORB)

クライアント側の要求元プログラムとサーバー側のオブジェクト間のメッセージ通信を管理するソフトウェア。ORB は、アクション要求およびそのパラメータをオブジェクトに渡し、結果を戻す。共通の実装は、CORBA および EJB である。「[共通オブジェクト要求ブローカ API \(Common Object Request Broker API: CORBA\)](#)」を参照。

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 または Oracle8i 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。interMedia 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。

拡張可能スタイルシート言語 (eXtensible Stylesheet Language: XSL)

XML ドキュメント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。XSL スタイルシートには、XSL Transformations (XSLT) および XSL Formatting Objects (XSLFO) という 2 つの W3C 勧告がある。

拡張可能マークアップ言語 (eXtensible Markup Language: XML)

データ記述のオープン標準。SGML 構文のサブセットを使用して W3C によって開発され、インターネットでの使用のために設計された。現行の標準はバージョン 1.0 で、1998 年 2 月に W3C 勧告として公開された。

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、interMedia 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。

共通オブジェクト要求ブローカ API (Common Object Request Broker API: CORBA)

ネットワーク全体の分散オブジェクト間の通信用の、Object Management Group による標準。これらの自己完結型ソフトウェア・モジュールは、異なるプラットフォームまたはオペレーティング・システム上で実行しているアプリケーションで使用できる。CORBA オブジェクトとそのデータ形式、およびファンクションは、Java、C、C++、Smalltalk、COBOL などの様々な言語にコンパイルできるインタフェース定義言語 (Interface Definition Language: IDL) で定義される。

共通ゲートウェイ・インタフェース (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。

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。

構造化問合せ言語 (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。

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全体が含まれた要素。たとえば、`<Parent><Child></Child></Parent>` は、Child 要素がその Parent 要素内にネストされていることを示す。

サーバー側インクルード (Server-Side Include: SSI)

データまたは他の内容を、要求元ブラウザに送信する前に Web ページに置く HTML コマンド。

サーブレット (servlet)

サーバー（通常は Web またはアプリケーション・サーバー）内で実行する Java アプリケーション。サーブレットは、CGI スクリプトに相当する Java である。

スキーマ (schema)

データベース内の構造およびデータ型の定義。スキーマは、XML Schema の W3C 勧告をサポートする XML ドキュメントも示す。

スタイルシート (Stylesheet)

XML では、XML 処理命令で構成される XML ドキュメントを示す用語。XML 処理命令は、入力 XML ドキュメントを出力 XML ドキュメントに変換またはフォーマットするために、XML プロセッサによって使用される。

スレッド (thread)

プログラミングにおける、Windows、UNIX、Java などの複数のオペレーティング・システムをサポートするオペレーティング・システム内の、単一のメッセージまたはプロセス実行パス。

整形式 (well-formed)

XML ドキュメントが、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。

セキュア・ソケット・レイヤー (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。

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 ドキュメントの構造および要素内容が、参照 DTD または内部 DTD で宣言されたものと一貫している状態を示す用語。

データグラム (datagram)

XSQL Servlet が処理した SQL 問合せから、HTML ページに埋め込まれたリクエストに戻るテキストのフラグメント。XML 形式の場合もある。

データベース・アクセス記述子 (Database Access Descriptor: DAD)

データベース・アクセスに使用される、名前付きの一連の構成値。DAD は、データベース名や SQL*Net V2 サービス名などの情報、ORACLE_HOME ディレクトリ、および言語、ソート型、日付言語などの NLS 構成情報を指定する。

統合開発環境 (Integrated Development Environment: IDE)

ソフトウェアの開発を支援するために設計された、単一のユーザー・インタフェースから実行されるプログラム・セット。JDeveloper は、Java 開発用の IDE であり、エディタ、コンパイラ、デバッグ、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 ドキュメント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 ドキュメントの 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。

トランスミッション・コントロール・プロトコル/インターネット・プロトコル (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。

名前空間 (namespace)

XML ドキュメント内にある、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 ドキュメントまたは 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。

バイナリ・ラージ・オブジェクト (Binary Large Object: BLOB)

内容がバイナリ・データで構成されたラージ・オブジェクト・データ型。このデータは、データ構造がデータベースに認識されないため、RAW 型とみなされる。

ハイパー・テキスト (hypertext)

ユーザーがハイパーリンクとして指定されたワードまたは句を選択して、他のドキュメントまたは図形間を操作できるテキスト・ドキュメントを作成および公開する方法。

ハイパー・テキスト転送プロトコル (Hypertext Transport 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。

ハイパー・テキスト・マークアップ言語 (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。

表記法 (NOTATION)

XML では、パーサーが理解できない内容の型の定義。これらの型には、オーディオ、ビデオおよび他のマルチメディアが含まれる。

標準汎用マークアップ言語 (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。

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 ドキュメントの最初の部分。

文書型定義 (Document Type Definition: DTD)

XML ドキュメント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 ドキュメント内に含めることができるテキスト・ファイルである。

モード (mode)

XML では、DOM ツリー内のアドレス指定可能な各エンティティを示す用語。

文字データ (Character Data: CDATA)

ドキュメント内の解析対象外のテキストは、CDATA セクションに格納される。これによって、&、<、> などの、他に特別な機能を持つ文字を含めることができる。CDATA セクションは、要素の内容または属性内で使用できる。

ユーザー・インタフェース (User Interface: UI)

メニュー、スクリーン、キーボード・コマンド、マウス・クリック、およびユーザーによるソフトウェア・アプリケーションとの対話方法を定義するコマンド言語の組合せ。

ユニフォーム・リソース識別子 (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。

ユニフォーム・リソース・ロケータ (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。

要素 (element)

XML ドキュメントの基本論理単位。子、データ、属性とその値などの他の要素に対するコンテナとして機能する。要素は、開始タグ <name> および終了タグ </name>、または空要素の場合、<name/> によって識別される。

ラージ・オブジェクト (Large Object: LOB)

内部 LOB および外部 LOB に分割された SQL データ型のクラス。内部 LOB には BLOB、CLOB および NCLOB が含まれ、外部 LOB には BFILE が含まれる。「[BFILE](#)」、「[バイナリ・ラージ・オブジェクト \(Binary Large Object: BLOB\)](#)」および「[キャラクタ・ラージ・オブジェクト \(Character Large Object: CLOB\)](#)」を参照。

ラッパー (Wrapper)

通常、汎用またはオブジェクト・インタフェースを提供するために、他のデータまたはソフトウェアをラップするデータ構造またはソフトウェアを示す用語。

リスナー (listener)

入力プロセスを監視する個別のアプリケーション・プロセス。

ルート要素 (root element)

XML ドキュメント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グとエピログの間に存在する。XML ドキュメントには、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。

ローカル・エリア・ネットワーク (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。

ワーキング・グループ (Working Group: WG)

特定のインターネット・テクノロジー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。

ワイド・エリア・ネットワーク (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。

数字

2 品目のトランザクションの例, 10-36

A

API, 用語集 -1
AppCste.java, 12-147
AQ Broker-Transformer, 12-32
AQReader.java, 12-166
AQWriter.java, 12-168
AQ 環境の設定, 9-5
AQ キュー作成のスクリプト, 12-21
AQ 使用例, 9-2
AQ スキーマ・スクリプト, 12-20
ArtesiaTech, 15-1

B

B2B, 用語集 -1
B2B XML アプリケーション, 12-2
B2B XML アプリケーションの実行, 12-35
B2B XML アプリケーションの停止, 12-80
B2B XML アプリケーションの要件, 12-2
B2BMessage.java, 12-171
B2B の定義, 用語集 -1
B2B メッセージ機能, 2-10, 2-11, 2-13, 2-15
B2C の定義, 用語集 -1
B2C メッセージ機能, 2-10
BC4J
 XSQL クライアント, 13-6
BC4J (Business Component for Java), 13-4
BC4J の定義, 用語集 -1
Bean マークアップ言語 (BML), 15-7
BLOB の定義, 用語集 -14

BML, 15-7
BrokerThread.java, 12-155
Broker-Transformer, 12-70
Broker スキーマの移入, 12-24
Broker スキーマの作成, 12-22
BuildAll.sql, 12-14
BuildSchema.sql, 12-15
Business Components for Java の定義, 用語集 -1
Business Component for Java
 XSQL クライアント, 13-6
Business Component for Java (BC4J), 13-4

C

catalogTradingPartner, 10-33
CDATA の定義, 用語集 -15
CGI の定義, 用語集 -12
CLASSPATH の定義, 用語集 -2
CLOB としての XML メッセージ, 9-4
CLOB の定義, 用語集 -11
Coins, 15-7
Common Oracle Runtime Environment の定義,
 用語集 -2
CORBA の定義, 用語集 -11
CORE の定義, 用語集 -2
CRUD, 15-9
CTX_DDL PL/SQL パッケージ, 5-8

D

DAD の定義, 用語集 -14
DBMS_XMLSave, 4-46
DOCTYPE の定義, 用語集 -2
DOM, 15-9
DOM の定義, 用語集 -14

dropOrder.sql, 12-29
DTD の定義, 用語集 -15
Dun & Bradstreet, 10-20
D-U-N-S, 10-20
Dynamic News アプリケーション, 6-2
 主なタスク, 6-2
 概要, 6-2
 サブルーット, 6-4
 動作, 6-5

E

EJB の定義, 用語集 -2
Enterprise Java Bean の定義, 用語集 -2
Event Manager
 非同期送受信, 11-9
EXECUTE 権限, 9-5
eXtensible Stylesheet Language Formatting Object の定義, 用語集 -3
eXtensible Stylesheet Language Transformation の定義, 用語集 -3

F

FAQ, 3-21
 interMedia Text, 5-45
 JDeveloper, 13-20
 XML アプリケーション, 13-26
 XSU, 4-57
FAQ, XML および AQ, 9-11
Flight Finder, 8-1
 概要, 8-2
 スタイルシートを使用した XML のフォーマット, 8-10
 データベースへの XML の書込み, 8-17
 問合せ, 8-6
 動作, 8-3

G

getXML, 4-15

H

HandWeb, 12-2
HTML の定義, 用語集 -15
HTML 要素, 10-40

HTTP の定義, 用語集 -14
HTTP リスナー, 12-2

I

IDE の定義, 用語集 -14
IDL または XML、選択, 15-6
IIOP の定義, 用語集 -3
iMessage (Internet Message Studio), 11-11
INFORMATIONAL, 10-34
interMedia, 3-15, 5-2
 CONTAINS 演算子, 5-4
 問合せ, 5-4
interMedia Text
 概要, 5-2
 問合せ, 5-30
 問合せアプリケーション, 5-19
 ユーザーおよびロール, 5-3
interMedia Text 索引, 5-7
 作成, 5-13
interMedia の定義, 用語集 -3
Internet File System の定義, 用語集 -3
Internet Message Studio (*iMessage*), 11-11
iProcurement, 10-2

J

Java Beans, 3-7
Java Bean の定義, 用語集 -4
Java Database Connectivity の定義, 用語集 -4
Java Runtime Environment の定義, 用語集 -4
Java の定義, 用語集 -3
JDBC の定義, 用語集 -4
JDeveloper, 12-2, 13-1
 3.2, 13-2
 FAQ, 13-26
 XML Data Generator Web Bean, 13-15
 XML 機能, 13-10
 XSQL Servlet の使用, 13-17
 概要, 13-2
 動作環境, 13-3
 モバイル・アプリケーション, 13-20
JDeveloper の定義, 用語集 -4
JDK の定義, 用語集 -4
JMS, 9-8
JMS を使用した XML メッセージの処理, 9-8
JRE の定義, 用語集 -4

JSP の定義, 用語集 -4
JVM の定義, 用語集 -4

K

KNOWN, 10-34
Koala オブジェクト・マークアップ言語, 15-7
KOML, 15-7

L

LAN の定義, 用語集 -16
LOB の定義, 用語集 -16

M

maxRows, 4-25
MessageBroker.java, 12-160
MessageHeaders.java, 12-146
mkAQUser.sql, 12-20
mkQ.sql, 12-21
mkSSTables.sql, 12-22

N

no rows 例外, 4-31
number portability, 11-4
Number Portability のプロセス, 11-6
n 層アーキテクチャ, 15-1
n 層アーキテクチャ構築, 15-2
N 層の定義, 用語集 -5

O

OAG の定義, 用語集 -5
OAI の定義, 用語集 -5
OASIS の定義, 用語集 -6
OAS の定義, 用語集 -6
OE の定義, 用語集 -5
OIS の定義, 用語集 -6
Open Applications Group の定義, 用語集 -5
Oracle Applilcation Server の定義, 用語集 -6
Oracle Exchange の定義, 用語集 -5
Oracle Integration Server の定義, 用語集 -6
Oracle *interMedia*, 3-15
Oracle *interMedia* Text, 5-2
Oracle Internet Procurement, 10-2

ORACLE_HOME の定義, 用語集 -6
Oracle8i JVM の定義, 用語集 -6
Oracle8i の XML を使用する理由, 1-13
Oracle の XML, 1-7
ORB の定義, 用語集 -10

P

ParserTest.java, 12-117
PCDATA の定義, 用語集 -11
PDA ブラウザ, 12-2
Persistent Interchange for Assets, 15-10
Persistent Interchange Syntax for Assets (PISA), 15-7
phone number portability のメッセージ機能, 11-2
PISA, 15-7, 15-10
PL/SQL
 CTX_DDL パッケージ, 5-8
 XSU, 4-41
 XSU のバインド値, 4-45
PL/SQL の定義, 用語集 -6
Point-to-Point, 9-2
Portal-to-Go, 8-21
 Java Transformer, 7-23
 XML でのデータ交換, 7-9
 XML への変換, 7-22, 7-13
 XSL Stylesheet Transformer, 7-26
 概要, 7-2
 機能, 7-3
 コンテンツの抽出, 7-10
 コンポーネント, 7-6
 サポートするデバイスおよびゲートウェイ, 7-4
 サンプル・アダプタ・クラス, 7-18
 実行要件, 7-3
 ターゲット・マークアップ言語, 7-22
 動作, 7-5
 例 1, 7-30
 例 2, 7-30
PUBLIC の定義, 用語集 -6
putXML, 4-17

R

ReadStructAQ.java, 12-172
reset.sql, 12-25, 12-26

S

SAX, 15-9
SAX の定義, 用語集 -7
Schema, 認証されていない, 10-46
SDP
 number portability, 11-4
 メッセージ機能アーキテクチャ, 11-4
SDP (Service Delivery Platform), 11-4
Service Delivery Platform (SDP), 11-4
setup.sql, 12-24
SGML の定義, 用語集 -15
Simple API for XML, 15-9
Simple API for XML の定義, 用語集 -7
skipRows, 4-25
SQL のコール順序, 12-13
SQL の定義, 用語集 -12
SSI の定義, 用語集 -12
SSL, 10-42
SSL の定義, 用語集 -13
StopAllQueues.java, 12-173
Stylesheet 表, XSL, 12-22
SupplierFrame.java, 12-177
SupplierWatcher.java, 12-183
SYSTEM の定義, 用語集 -7

T

TCP/IP の定義, 用語集 -14
Transformer API, 12-5
Type フィールド, 10-12

U

UI の定義, 用語集 -15
UNKNOWN, 10-34
URI の定義, 用語集 -15
URL の定義, 用語集 -15

W

W3C の定義, 用語集 -8
WAN の定義, 用語集 -16
Web Bean
 XML Data Generator, 13-15
Web Request Broker の定義, 用語集 -8
Web からデータベース, 1-37

WG の定義, 用語集 -16
World Wide Web Consortium の定義, 用語集 -8
WRB の定義, 用語集 -8
WriteStructAQ.java, 12-174

X

XDK の定義, 用語集 -8
XLink の定義, 用語集 -8
XML
 Business Compornent for Java, 13-4
 Oracle の XML, 1-7
 記憶領域オプション, 1-25
 作成, 1-22
 生成, 1-22
 設計問題, 1-37
 表示のカスタマイズ, 8-1
XML BeanMaker, 15-7
XML Class Generator, 3-6
XML Class Generator の定義, 用語集 -8
XML Data Generator, 13-15
XML Developer's Kit の定義, 用語集 -8
XML Flight Finder のサンプル・アプリケーション,
 8-2
XML Parser, 3-4
XML Query の定義, 用語集 -8
XML Schema, 1-31
XML Schema の定義, 用語集 -8
XML SQL Utility, 4-3, 4-41
XML SQL Utility for Java, 4-19
XML SQL Utility (XSU), 3-12
XML Transviewer Beans, 3-7
XML Transviewer Beans の定義, 用語集 -8
XML-CORBA Link, 15-7
XML アプリケーション, 1-3, 3-16, 13-1
 JDeveloper, 13-26
 JDeveloper の使用, 13-14
XML 機能
 JDeveloper 3.2, 13-10
XML コンポーネント, 3-2
 XML ドキュメントの生成, 3-17
XML データ
 送信, 1-37
XML データの送信, 1-37
XML ドキュメント, 3-17
 interMedia, 5-7
 格納, 1-35

生成, 1-35
セクション, 5-42
通信, 1-38
XML ドキュメントの格納, 1-35
XML ドキュメントの生成, 1-35
XML の格納, 1-20, 4-17, 4-32
XML のシリアルライズ化, 15-7
XML のスクリプト, 15-9
XML の生成, 4-15, 4-28
XML の抽出, 1-20
XML の定義, 用語集 -11
XML パーサーの定義, 用語集 -9
XML へのデータ変換, 理由, 12-5
XML または IDL, 選択, 15-6
XML メッセージ機能
 phone number portability, 11-1
XORBA, 15-7
XPath の定義, 用語集 -9
XPather の定義, 用語集 -9
XSL Stylesheet 表, 12-22
XSL Transformation Processor, 3-6
XSLFO の定義, 用語集 -3
XSLT の定義, 用語集 -3
XSL 管理スクリプト, 12-82
XSL の定義, 用語集 -11
XSQL Page Processor, 3-8
XSQL Servlet, 3-8, 12-2, 13-17
XSQLConfig.xml, 12-140
XSQL の定義, 用語集 -9
XSU, 3-12, 4-3
 FAQ, 4-57
 PL/SQL, 4-41
 PL/SQL での挿入処理, 4-47
 XML の生成, 4-15
 クライアント側, 4-14
 コマンドラインの使用, 4-14
 実行できる場所, 4-5
 使用のガイドライン, 4-9
 使用方法, 4-4
 スタイルシート, 4-44
 バインド値, 4-45
 マッピングの手引き, 4-9

あ

アクション・ハンドラ, 12-50
アドバンスト・キューイングのスクリプト, 12-155

アドバンスト・キューイングの定義, 9-2
アドバンスト・キューイング, 使用, 12-6
アプリケーション, 2-2
 XML ドキュメントの通信, 1-38
アプリケーション・サーバー, 用語集 -10
アプリケーションのメッセージ設定
 作成, 11-11
アプリケーション・プログラム・インタフェースの定
 義, 用語集 -1

い

一部動的なページ, 6-8
インスタンス化の定義, 用語集 -10
インストール
 interMedia Text, 5-3

え

エンキュー後の新しい受信者の追加, 9-12
エンティティの定義, 用語集 -10
エンド・ユーザー設定項目, 6-12

お

オブジェクト・ビューの定義, 用語集 -10
オブジェクト・リレーショナル・インフラストラク
 チャ, 1-20
オブジェクト・リレーショナルの定義, 用語集 -10
親要素の定義, 用語集 -10

か

カートリッジの定義, 用語集 -11
開発ツール, 3-3
価格データ要素, 10-23
価格の定義, 10-18
拡張可能なアーキテクチャ, 1-20
拡張可能スタイルシート言語の定義, 用語集 -11
カスケードリング・スタイルシート, 用語集 -2
カタログ・ソース, 10-33
カタログ表, 10-2
各国語キャラクタ・ラージ・オブジェクトの定義,
 用語集 -11
カテゴリ・データ要素, 10-22
カテゴリの定義, 10-18
空要素の定義, 用語集 -11

環境のクリーン・アップ, 12-25
監査, 9-4
管理
 コンテンツおよびドキュメント, 2-3
管理情報 (ADMIN), 10-11
管理スクリプト, 12-114

き

記憶領域オプション
 XML, 1-25
キュー・アプリケーションの起動, 12-26
キュー・アプリケーションの削除, 12-26
キュー・アプリケーションの停止および削除, 12-26
キュー作成のスクリプト, 12-21
キュー・テーブル, 9-5
キューを開始する SQL スクリプト, 12-28
キューを削除する SQL スクリプト, 12-27
キューを作成する SQL スクリプト, 12-28
キューを停止する SQL スクリプト, 12-27
共通オブジェクト要求ブローカ API の定義,
 用語集 -11

く

クライアント / サーバーの定義, 用語集 -12

け

契約情報, 10-3
契約の定義, 10-18
結果セット・オブジェクト, 4-28
結果セットの定義, 用語集 -12
結果のページ区切り, 4-25
言語の識別, 10-7
検索
 XML ドキュメント, 5-7

こ

更新処理, 4-36, 4-49
構造化 XML ドキュメント, 1-26
小売業者とサブライヤ間のスキーマ, 12-15
小売業者とサブライヤ間のトランザクション, 12-31
小売業者のスキーマ, 12-14
小売業者のスクリプト, 12-147
小売業者の発注, 12-48

コール順序、SQL, 12-13
コールバックの定義, 用語集 -12
国際化, 10-6
コンテンツおよびドキュメントの管理, 2-3
コンテンツ管理, 2-3
コンテンツ管理、iProcurement, 10-2
コンテンツのパーソナライズ, 6-11

さ

サーバー側インクルードの定義, 用語集 -12
サブレット
 Dynamic News アプリケーション, 6-4
サブレットの定義, 用語集 -12
削除処理, 4-39, 4-51
作成済 XML, 1-22
 格納, 1-28, 1-34
作成済 XML の格納, 1-34
作成、読み込み、更新、削除 (CRUD), 15-9
サブライヤが提供するカタログ, 10-5, 10-18
サブライヤ・データ要素, 10-24
サブライヤのスキーマ, 12-14
サンプル, 1-39

し

状態管理, 15-13
承認済サブライヤのリスト, 10-3
使用方法, 4-55
処理
 PL/SQL での挿入, 4-47
 更新, 4-36, 4-49
 削除, 4-39, 4-51
 挿入, 4-33
処理および管理スクリプト, 12-114
シリアル化、XML, 15-7

す

スキーマ情報 (SCHEMA), 10-12
スキーマ・スクリプト, 12-20
スキーマ、認証済, 10-44
スキーマの定義, 用語集 -12
スキーマ表, 9-5
スタイルシート
 XSU, 4-44
スタイルシートの定義, 用語集 -13

スレッドの定義, 用語集 -13

せ

整形式の定義, 用語集 -13

生成される XML, 1-22, 1-31, 3-21

格納, 1-26

静的なページ, 6-6

設計問題, 1-37

セッション・チケット, 10-42

セッションの定義, 用語集 -13

そ

層内の通信, 15-12

挿入処理, 4-33

属性の定義, 用語集 -13

ソフトウェアの品質保証, 15-13

た

タグの定義, 用語集 -13

妥当の定義, 用語集 -13

ち

チャンネル定義書式の定義, 用語集 -1

注文テンプレート, 10-3

注文明細の XML 定義, 10-26

つ

追加情報, 3-33

追加属性の定義, 10-18

追跡, 9-4

ツール, 3-16

て

提起するデジタル資産集計, 15-17

定数保持, Message Broker, 12-147

データグラムの定義, 用語集 -13

データ交換アプリケーション, 1-37

データ交換フロー B2B アプリケーション, 12-30

データ送信, 15-14

データ表示のカスタマイズ

データ表示

データのカスタマイズ, 2-3

データベース・アクセス記述子の定義, 用語集 -14

データベースへの XML の格納, 4-46

テキスト

表記規則, xxv

テキスト問合せ式, 5-19

デジタル資産管理, 15-15

デジタル資産の定義, 15-2

デモ, 1-39

電子データ交換の定義, 用語集 -2

と

問合せ

SECTION GROUPS, 5-28

結果, 5-44

属性セクション, 5-26

問合せアプリケーション, 5-30

統合開発環境の定義, 用語集 -14

統合ツール, 1-13

動的なページ, 6-10

ドキュメント

Java, 3-17

PL/SQL, 3-19

ドキュメント・オブジェクト・モデル, 15-9

ドキュメント・オブジェクト・モデルの定義,
用語集 -14

ドキュメント管理, 2-3

ドキュメントのマッピング, 1-31

トランザクション状態管理, 15-13

な

名前空間の定義, 用語集 -14

に

ニュース項目, 6-16, 6-18

インポート, 6-22

エクスポート, 6-22

認証されていない XML Schema, 10-46

認証済 XML Schema, 10-44

認証, ユーザー, 10-42

ね

ネットワーク要素
提供, 11-9
ネットワーク要素の設置, 11-9

は

パーソナル・デジタル・アシスタントの定義, 用語集
-6
バイナリ・ラージ・オブジェクトの定義, 用語集 -14
ハイパー・テキスト転送プロトコルの定義, 用語集 -14
ハイパー・テキストの定義, 用語集 -14
ハイパー・テキスト・マークアップ言語の定義,
用語集 -15
ハイブリッドな格納, 1-29, 1-37
バイヤーが提供するカタログ, 10-9
バイヤーが提供するコンテンツ, 10-3
ハブおよび spoke アーキテクチャ, 9-4
パブリッシュ / サブスクライブ, 9-2

ひ

非構造化 XML ドキュメント, 1-28
非同期メッセージ
Event Manager, 11-9
送受信, 11-9
表記法の定義, 用語集 -15
表示のカスタマイズ, 6-19
品質保証、XML ベース, 15-13
品目情報、DTD, 10-15
品目の定義, 10-18
品目マスター, 10-3

ふ

複数層通信, 15-2
プロローグの定義, 用語集 -15
文書型定義 (DTD), 10-9
文書型定義の定義, 用語集 -15

へ

変換, 1-31

ま

マイニング, 9-4

め

メッセージ機能
B2B および B2C, 2-10
phone number portability, 11-1
メッセージ機能アーキテクチャ, 11-4
メッセージ・サーバー, 9-2
メッセージ保存, 9-4

も

モードの定義, 用語集 -15
モバイル・アプリケーション
JDeveloper, 13-20

ゆ

ユーザー・インタフェースの定義, 用語集 -15
ユニフォーム・リソース識別子の定義, 用語集 -15
ユニフォーム・リソース・ロケータの定義, 用語集 -15

よ

用語集, 用語集 -1
要素の定義, 用語集 -16

ら

ラッパーの定義, 用語集 -16

り

リスナーの定義, 用語集 -16
リソース記述フレームワークの定義, 用語集 -7
リッチ・メディアの定義, 15-2

る

ルート要素の定義, 用語集 -16

れ

列挙定義, 10-33

レンダラの定義, 用語集 -16

ろ

ローカル・エリア・ネットワークの定義, 用語集 -16

ロードマップ, 1-5

わ

ワイド・エリア・ネットワークの定義, 用語集 -16

