BEA WebLogic Server™

WebLogic XML プログラミング ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

WebLogic XML プログラミング ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1

目次

このマニュアルの内容

1. XML の概要

2. WebLogic Server による XML アプリケーションの開発

3. XML プログラミング手法

4. WebLogic Server XML の管理

5. XML リファレンス

索引

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ XML ソフトウェアの使い方について説明します。XML ソフトウェアの使用に関連する概念を定義し、XML アプリケーションの開発プロセスについて述べています。また、このマニュアルでは、アプリケーション プログラミング インタフェース（API）、管理タスク、および XML ツールについても説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「XML の概要」は、XML ソフトウェアとそのコンポーネントの基本的な説明です。
- 第 2 章「WebLogic Server による XML アプリケーションの開発」では、WebLogic Server と XML ツールを使用して XML アプリケーションを開発する方法について説明します。
- 第 3 章「XML プログラミング手法」では、XML ドキュメントにおけるメッセージ駆動型 Bean および JMS キューの使用といった、特定のタスクを処理するためのプログラミング技術について説明します。
- 第 4 章「WebLogic Server XML の管理」では、Administration Console の XML レジストリ、および XML のコンフィグレーション タスクを行う方法について説明します。
- 第 5 章「XML リファレンス」では、この XML ソフトウェアでサポートしている仕様およびアプリケーション プログラミング インタフェースへのリンクを提供しています。

対象読者

このマニュアルは、XML アプリケーションを設計、開発、コンフィグレーション、および管理するシステム管理者およびプログラマを対象としています。Web 技術、XML、XSLT、Java プログラミング言語、サーブレット、および J2EE 仕様の JSP API に読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA ホームページの [製品のドキュメント] をクリックするか、WebLogic Server 製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/wls61）を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルのメイン トピックを一度に 1 つずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は、Adobe の Web サイト（http://www.adobe.co.jp）から無料で入手できます。

関連情報

XML の詳細については、1-17 ページの「XML について学習するには」を参照してください。

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例 :
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例 : `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。

表記法	適用
[ ]	構文内の任意指定の項目を示す。 例 : `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例 : `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 XML の概要

以下の節では XML テクノロジと WebLogic Server XML サブシステムについて概説します。

- XML とは
- XML ドキュメントの記述方法
- XML を使用する理由
- XSL と XSLT
- DOM と SAX とは
- JAXP とは
- XML と XSLT の一般的な使い方
- WebLogic Server XML の機能
- XML について学習するには

XML とは

Extensible Markup Language （XML）とは、ドキュメント内のデータの内容および構造を示すのに使用するマークアップ言語です。XML は、Standardized General Markup Language （SGML）を簡素化したものです。XML は、インターネット上でコンテンツを配信するための業界標準です。また、新しいタグを定義する機能があるので、拡張が可能です。

HTML と同じように、XML でもタグを使用してコンテンツを記述します。しかし、XML のタグはコンテンツの表示方法ではなく、データの意味と階層構造を表します。この機能を使用すれば、異なるプログラムやシステム間でのデータ交

換を効率化するために必要な高度なデータ型を定義できます。さらに、XML ではコンテンツと表現を分離できるため、コンテンツ（データ）を異種システム間で移植できます。

XML 構文では、一対の開始タグと終了タグ（<name> と </name> など）を使用して情報をマークアップします。タグで区切られる情報のことを、要素と呼びます。どの XML ドキュメントにもルート要素が 1 つあります。ルート要素は、他のすべての要素を含む最上位の要素です。他の要素内に含まれる要素をしばしば下位要素と言います。要素には、名前と値のペアという構造の属性が付くことがあります。属性は、要素の一部として要素をさらに定義するのに使用します。

次のサンプル XML ファイルでは、アドレス帳の内容を説明します。

```
<?xml version="1.0"?>

<address_book>
  <person gender="f">
    <name>Jane Doe</name>
    <address>
      <street>123 Main St.</street>
      <city>San Francisco</city>
      <state>CA</state>
      <zip>94117</zip>
    </address>
    <phone area_code=415>555-1212</phone>
  </person>
  <person gender="m">
    <name>John Smith</name>
    <phone area_code=510>555-1234</phone>
    <email>johnsmith@somewhere.com</email>
  </person>
</address_book>
```

XML ファイルのルート要素は、address_book です。アドレス帳には現在、person 要素の形で「Jane Doe」と「John Smith」の 2 つのエントリがあります。Jane Doe のエントリには住所と電話番号、John Smith のエントリには電話番号と電子メール アドレスがあります。この XML ドキュメントの構造では、要素の本体の下位要素ではなく、phone 要素に area_code 属性を指定して市外局番を格納するように定義しています。すべての下位要素が person 要素で必要なわけではないことにも注意してください。

XML ドキュメントの記述方法

XML ドキュメントの記述方法には、DTD および スキーマの 2 種類があります。

DTD（Document Type Definition : 文書型定義）は、XML ドキュメントの構造で基本的に必要なものを定義します。DTD では、XML ドキュメントで有効な要素と属性、およびそれらが有効となるコンテキストを定義します。言い換えれば、DTD では特定のタグ内で有効なタグ、およびオプションのタグと属性を指定します。

次の例では、上のアドレス帳のサンプル XML ドキュメントを定義する DTD を示します。

```
<!DOCTYPE address_book [
<!ELEMENT person (name, address?, phone?, email?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (street, city, state, zip)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>

<!ATTLIST person gender CDATA #REQUIRED>
<!ATTLIST phone area_code CDATA #REQUIRED>
]>
```

スキーマは、XML 仕様において最近開発され、DTD に取って代わるものとされています。スキーマでは、DTD および DTD 以外の XML ドキュメント自体より柔軟かつ詳細に XML ドキュメントを記述します。スキーマ仕様は、現在 World Wide Web Consortium（W3C）によって開発中であり、DTD の制限の多くを克服するものとされています。XML スキーマの詳細については、http://www.w3.org/TR/xmlschema-0/ を参照してください。

次の例では、上のアドレス帳のサンプル XML ドキュメントを定義するスキーマを示します。

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<xsd:element     name="address_book" type="bookType"/>

<xsd:complexType name="bookType">
  <xsd:element name=name="person"     type="personType"/>
</xsd:complexType>
```

```
<xsd:complexType name="personType">
  <xsd:element   name="name"     type="xsd:string"/>
  <xsd:element   name="address"  type="addressType"/>
  <xsd:element   name="phone"    type="phoneType"/>
  <xsd:element   name="email"    type="xsd:string"/>
  <xsd:attribute name="gender"   type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="addressType">

  <xsd:element   name="street"  type="xsd:string"/>
  <xsd:element   name="city"    type="xsd:string"/>
  <xsd:element   name="state"   type="xsd:string"/>
  <xsd:element   name="zip"     type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="phoneType">
  <xsd:restriction base="xsd:string"/>
  <xsd:attribute name="area_code" type="xsd:string"/>
</xsd:simpleType>

</xsd:schema>
```

XML ドキュメントでは、ドキュメント自体の一部として DTD またはスキーマを指定することも、DOCTYPE 宣言で外部 DTD またはスキーマを参照することも、DTD やスキーマをまったく指定または参照しないようにすることもできます。次の XML ドキュメントの抜粋では、`address.dtd` という外部 DTD を参照する方法を示しています。

```
<?xml version=1.0?>
<!DOCTYPE address_book SYSTEM "address.dtd">
<address_book>
...
```

XML ドキュメントは、パーサで検証する場合、または複雑なタイプが含まれる場合にのみ DTD またはスキーマを付ける必要があります。XML ドキュメントは、1) 関連する DTD またはスキーマがある場合、および 2) 関連する DTD またはスキーマに定義されている制約に準拠している場合に有効であると見なされます。ただし、XML ドキュメントが整形式でなければならない場合は、ドキュメントに DTD またはスキーマを付ける必要はありません。W3C の XML 1.0 勧告のすべてのルールに従っている場合、ドキュメントは整形式と見なされます。XML 1.0 仕様の完全な説明については、http://www.w3.org/XML/ を参照してください。

XML を使用する理由

通常、ある業界では、その業界にとって意味がある固有のデータ交換方式を使用しています。電子商取引の誕生により、企業ではさまざまな業界と関係を結ぶようになっています。このため、こうした企業は、これらの業界で使用されるさまざまな電子通信プロトコルに関する専門知識を取得しなければなりません。

XML は拡張性に優れているため、さまざまな業界間のデータ交換フォーマットを標準化するための非常に効果的なツールとなっています。たとえば、複数の業界間または一企業内の複数の部門間でメッセージ ブローカとワークフロー エンジンがトランザクションを調整しなければならない場合、XML を使用すると、異なるソースからのデータを組み合わせて、すべての当事者が理解できる 1 つのフォーマットを作成できます。

XSL と XSLT

Extensible Stylesheet Language（XSL）は、XML ドキュメントに適用する表示のルールを記述するための W3C の規格です。XSL には、変換言語（XLST）とフォーマット言語の両方が含まれます。この 2 つの言語は、互いに独立して機能します。XSLT は XML ベースの言語で、XML ドキュメントを別の XML ドキュメントに、または HTML、PDF、その他の文書フォーマットに変換する方法を記述するための W3C の仕様です。

XSLT トランスフォーマは、XML ドキュメントと XSLT ドキュメントを入力として受け付けます。XSLT ドキュメントに定義されるテンプレート ルールには、そのルールの適用先となる XML ツリーを指定するパターンが含まれています。XSLT トランスフォーマは、XML ドキュメントをスキャンしてルールに一致するパターンを見つけ出し、元の XML ドキュメントの該当するセクションにテンプレートを適用します。

DOM と SAX とは

DOM および SAX は、XML データ解析のための 2 つの Java アプリケーション プログラミング インタフェース（API）です。この 2 つの API は、解析に対して異なるアプローチをとっており、それぞれの API に長所と短所があります。

SAX

SAX は Simple API for XML の略語です。SAX は、イベントベースの XML 解析用インタフェースです。SAX は、パーサが要素の最初または最後などの XML ドキュメントを読み取る際に発生するイベントを定義します。プログラマは、ドキュメント解析時に異なるイベントを処理するハンドラを提供します。

XML ドキュメントを解析するために SAX API を使用するプログラマは、これらのイベントが発生したときに起こることを完全に制御する権限を持ち、解析プロセスをカスタマイズできます。たとえば、プログラマは、ドキュメントが不正であることを示すエラーがパーサで発生したときに、ドキュメントがすべて解析されるまで待つのではなく、すぐに XML ドキュメントの解析を中止できるので、パフォーマンスが向上します。

WebLogic Server の組み込みパーサ（Apache Xerces）は、SAX バージョン 2.0 をサポートしています。SAX バージョン 1.0 を使用して XML ドキュメントを解析するプログラムを作成した場合は、2 つのバージョンの相違点を把握し、それに従ってプログラムを更新する必要があります。2 つのバージョンの相違点については、http://www.megginson.com/SAX/Java/changes.html を参照してください。

DOM

DOM は Document Object Model の略語です。DOM は、XML ドキュメントをメモリに読み込んでツリー表示します。ツリーの各ノードは、元の XML ドキュメントからの特定のデータの一部を表します。ツリー構造は、データを表すための標準プログラミング メカニズムで、Java を使用したツリーのナビゲートおよび操作を簡単に素早く、かつ効率的に実行できます。しかし、主な欠点として、

DOM でツリーを作成するために XML ドキュメント全体を読み取る必要があるため、XML ドキュメントのサイズが大きくなると、アプリケーションのパフォーマンスが低下するというものがあります。

WebLogic Server の組み込みパーサ（Apache Xerces）は、DOM Level 2.0 Core をサポートしています。DOM Level 1.0 を使用して XML ドキュメントを解析するプログラムを作成した場合は、2 つのバージョンの相違点を把握し、それに従ってプログラムを更新する必要があります。相違点の詳細については、http://www.w3.org/DOM/DOMTR を参照してください。

JAXP とは

前の節では、XML データの解析に使用できる API、SAX、DOM について説明しました。Java API for XML Processing（JAXP）では、これらのパーサにアクセスする手段を提供します。JAXP はまた、どのような仕様のパーサまたはトランスフォーマも使用できるプラグイン可能レイヤも定義します。

WebLogic Server では、XML アプリケーションの開発と、WebLogic Server で構築した XML アプリケーションを他の Web アプリケーション サーバに移植するのに必要な作業を容易にするために、Java API for XML Processing（JAXP）を実装しています。JAXP は、XML アプリケーションの移植性を高めるために Sun Microsystems で開発されました。JAXP では、Java プラットフォームの API の標準セットを介して XML ドキュメントを解析および変換するための基本機能がサポートされています。WebLogic Server 配布キット内の JAXP 1.1 は、組み込みパーサを使用するコンフィグレーションになっています。したがって、WebLogic Server で構築される XML アプリケーションでは、デフォルトで JAXP を使用します。

WebLogic Server 配布キットには、JAXP 1.1 で必要なインタフェースとクラスがあります。JAXP 1.1 には、SAX バージョン 2 および DOM Level 2 に対するサポートがあります。JAXP 用の Javadoc は、WebLogic Server のオンライン リファレンス マニュアルに含まれています。

JAXP パッケージ

JAXP には、以下の 2 つのパッケージがあります。

- javax.xml.parsers
- javax.xml.transform

javax.xml.parsers パッケージには、SAX バージョン 2.0 および DOM Level 2.0 モードで XML データを解析するためのクラスがあります。XML ドキュメントを SAX モードで解析するには、まず newInstance() メソッドで新しい SaxParserFactory オブジェクトのインスタンスを作成します。このメソッドは、明確に定義された場所のリストを基に、ロードするパーサの特定の実装をルックアップします。次に、SaxParserFactory から SaxParser インスタンスを取得し、parse() メソッドを実行して、解析する XML ドキュメントにそれを渡します。XML ドキュメントを DOM モードで解析する場合も同様ですが、この場合は DocumentBuilder クラスと DocumentBuilderFactory クラスを使用します。

JAXP による XML ドキュメント解析の詳細については、2-2 ページの「XML ドキュメントの解析」を参照してください。

javax.xml.transform パッケージには、XML ドキュメント、DOM ツリー、または SAX イベントなどの XML データを別の形式に変換するクラスがあります。トランスフォーマ クラスも、パーサ クラスと同じように機能します。XML ドキュメントを変換するには、まず newInstance() メソッドで新しい TransformerFactory オブジェクトのインスタンスを作成します。このメソッドは、明確に定義された場所のリストを基に、ロードする XSLT トランスフォーマの特定の実装をルックアップします。次に、特定の XSLT スタイルシートを基に新しい Transformer オブジェクトのインスタンスを作成し、transform() メソッドを実行して、変換する XML オブジェクトにそれを渡します。XML オブジェクトは、XML ファイルでも DOM ツリーなどでもかまいません。

JAXP による XML オブジェクト変換の詳細については、2-13 ページの「JAXP による XML データの変換」を参照してください。

XML と XSLT の一般的な使い方

XML と XSLT をどのように使用するかは、ビジネス上のニーズによって異なります。

XML と XSLT を使用したコンテンツと表示の分離

XML と XSLT は、複数のクライアント タイプをサポートするアプリケーションでよく使用されます。たとえば、ブラウザ ベースのクライアントと Wireless Application Protocol（WAP）クライアントの両方をサポートする Web ベース アプリケーションを使用しているとします。これらのクライアントは、それぞれ HTML と Wireless Markup Language（WML）という異なるマーク付け言語を理解しますが、アプリケーションでは両方に適したコンテンツを提供しなければなりません。

これを実現するには、クライアントに送信するデータを表す XML ドキュメントを最初に作成するようにアプリケーションを記述します。次に、アプリケーションは、データを表す XML ドキュメントを、クライアントのブラウザ タイプによって HTML または WML に変換します。アプリケーションでは、HTTP リクエストの `User-Agent` リクエスト ヘッダを調べることによって、クライアントのブラウザ タイプを識別できます。クライアントのブラウザ タイプを認識すると、適切な XSLT スタイル シートを使用して、適切なマーク付け言語にドキュメントを変換します。このヘッダ情報へのアクセス方法の例については、WebLogic Server 配布キットの `examples\servlets` ディレクトリに収められている SnoopServlet サンプルを参照してください。

このように、クライアント タイプごとに異なるマーク付け言語を使用して同じ XML ドキュメントを表示する方法を用いると、複数のクライアント タイプのサポートに必要な努力を、適切な XSLT スタイル シートの開発に集中させることができます。さらに、必要な場合は、アプリケーションを他のクライアント タイプに簡単に適合させることができます。

XSLT の詳細については、5-5 ページの「その他の情報源」を参照してください。

企業間通信用メッセージ フォーマットとしての XML

企業間（B2B）環境で、企業 A と企業 B は両者間の電子商取引に関する情報の交換を望んでいます。企業 A は大手の電子商取引サイトです。企業 B は、企業 A の製品を最適な顧客集団に販売する小規模な子会社です。企業 B が企業 A に顧客情報を送ると、企業 B は企業 A から 2 通りの方法で対価を受け取ることができます。企業 B は、金銭、および企業 B が提供した顧客と同じ購買層の他の顧客に関する情報を企業 A から受け取ります。情報を交換するためには、企業 A と企業 B はマシンが理解でき、どちらの企業のシステムでも簡単に処理できるデータ フォーマットについて合意する必要があります。

XML は、このシナリオで使用する論理データ フォーマットですが、このフォーマットの選択はほんの第一歩に過ぎません。次に両企業は、交換する XML メッセージのフォーマットについて合意しなければなりません。企業 A はその子会社と 1 対多の関係にあるため、やり取りする XML メッセージのフォーマットは企業 A 側で定義する必要があります。

XML メッセージ、つまり XML ドキュメントのフォーマットを定義するために、企業 A は 2 種類の文書型定義（DTD）を作成します。1 つは企業 A が提供する顧客情報を記述するもので、もう 1 つは企業 A が受け取る新しい子会社の情報を記述するものです。企業 B も、2 種類の DTD を作成します。1 つは企業 A から受け取る XML ドキュメントを処理するためのもの、もう 1 つは企業 A 側で処理できるフォーマットで XML ドキュメントを作成するためのものです。

WebLogic Server XML の機能

WebLogic Server は、WebLogic Server と WebLogic Server に基づく XML アプリケーションに適用可能な XML テクノロジを統合します。WebLogic Server XML サブシステムを使用すると、顧客は、標準パーサ、WebLogic FastParser、XSLT トランスフォーマ、DTD、および XML スキーマを使用して XML ファイルを処理および変換できます。

WebLogic Server XML サブシステムには、以下の機能があります。

- XML ドキュメント パーサ
- XML ドキュメント トランスフォーマ
- JAXP プラグイン可能レイヤの実装
- WebLogic サーブレット属性
- WebLogic XSLT JSP タグ ライブラリ
- パーサおよびトランスフォーマのコンフィグレーション用 XML レジストリ
- 外部エンティティの解決のコンフィグレーション用 XML レジストリ
- コード例

XML ドキュメント パーサ

WebLogic Server には、以下の 2 つのパーサがあります。

パーサ	説明
組み込み	組み込みパーサは、Apache Xerces パーサ バージョン 1.3.1 に基づいている。組み込みパーサは、Simple API For XML（SAX）モードまたは Document Object Model（DOM）モードのいずれかで使用できる。
WebLogic FastParser	WebLogic Web サービスに関連付けられた SOAP および WSDL ファイルなど、中小規模のドキュメントを処理するために特別に設計された高性能の XML パーサ。アプリケーションが中小規模（最大 10,000 要素）の XML ドキュメントを処理する場合、FastParser を使用するように WebLogic Server をコンフィグレーションする。 **注意：** WebLogic Server の旧バージョンには、カスタム パーサの作成機能がありました。カスタマイズされたパーサの対象となる XML ドキュメントのタイプに応じた WebLogic FastParser を使用できるので、WebLogic FastParser は、効果的にカスタマイズされたパーサ機能を置き換える。カスタマイズされたパーサの生成機能はなくなっている。 WebLogic FastParser の詳細については、2-9 ページの「WebLogic FastParser の使用」を参照。

Administration Console による XML レジストリのコンフィグレーションで、他の任意の XML パーサを使用できます。単一の WebLogic Server のインスタンスをコンフィグレーションすることによって、あるパーサを特定のアプリケーション用に使用し、別のパーサを異なるアプリケーション用に使用できます。

XML ドキュメント トランスフォーマ

WebLogic Server には、Apache Xalan XSLT トランスフォーマ バージョン 2.0.1 をベースとした組み込み XSLT トランスフォーマがあります。XML アプリケーションでは、この組み込み XSLT トランスフォーマまたは他の XSLT トランスフォーマを使用して、XML ドキュメントを変換できます。XML ドキュメントの詳細については、2-13 ページの「JAXP による XML データの変換」を参照してください。

JAXP プラグイン可能レイヤの実装

Java API for XML Processing（JAXP）1.1 は、Java 標準で、パーサに依存しない XML 用 API です。JAXP の詳細については、1-7 ページの「JAXP とは」を参照してください。

注意： WebLogic Server は、JAXP 1.1 仕様で定義されているように、システム プロパティを使用するのではなく、Administration Console からアクセスする XML レジストリを使用して、パーサおよびトランスフォーマをプラグインします。

WebLogic サーブレット属性

WebLogic Server は、以下の特殊なサーブレット属性をサポートします。

- `org.xml.sax.HandlerBase`
- `org.xml.sax.helpers.DefaultHandler`
- `org.w3c.dom.Document`

`ServletRequest` オブジェクトで、前述の属性を付けて `setAttribute` メソッド（SAX 解析の場合）および `getAttribute`（DOM 解析の場合）メソッドを呼び出すと、任意の XML ドキュメントが解析されます。

以下のコード セクションでは、これらのメソッドの使い方の例を示します。

```
request.setAttribute("org.xml.sax.helpers.DefaultHandler", new DefHandler());
org.w3c.dom.Document = (Document)request.getAttribute("org.w3c.dom.Document");
```

注意： `setAttribute` メソッドと `getAttribute` メソッドは便利な機能として用意されているだけであり、サーブレットから XML を解析する際には必須というわけではありません。

WebLogic XSLT JSP タグ ライブラリ

JSP タグ ライブラリは、WebLogic Server で動作する JavaServer Pages（JSP）内から組み込み XSLT トランスフォーマにアクセスするための単純なタグを提供します。現時点では、このタグは組み込み XSLT トランスフォーマだけをサポートしています。このタグを使用して、XML ドキュメントを JSP 内から異なるトランスフォーマを使って変換することはできません。

JSP タブ ライブラリは、`xmlx-tags.jar` に収められています。このファイルは、WebLogic Server 配布キットのインストール時に自動的にインストールされます。

注意： JSP タグ ライブラリは便利な機能として用意されているだけであり、JSP 内から XSLT トランスフォーマへのアクセスに必要というわけではありません。

パーサおよびトランスフォーマのコンフィグレーション用 XML レジストリ

XML レジストリは、管理タスクとコンフィグレーション タスクを XML アプリケーションから切り離すことによって、これらのタスクを簡素化します。WebLogic Server のインスタンスに対するパーサおよびトランスフォーマのコンフィグレーションには、Administration Console （WebLogic Server 管理用のグラフィカル ユーザ インタフェース（GUI））を使用します。

注意： 各 WebLogic Server ドメインにはレジストリを好きなだけ組み込むことができ、ドメイン内の各 WebLogic Server にはゼロまたは 1 つのレジストリを割り当てることができます。

XML レジストリを使用すると

- 構築時ではなくデプロイメント時にパーサまたはトランスフォーマを指定できます。
- アプリケーションにパーサ依存コードまたはトランスフォーマ依存コードを組み込む必要がありません。
- 単一のサーバで複数のパーサおよびトランスフォーマをより便利にサポートできます。

XML レジストリを使用すると、以下のタスクを行うことができます。

- WebLogic Server の本バージョンに付属の組み込みパーサの代わりに別の XML パーサをコンフィグレーションします。
- WebLogic Server の本バージョンに付属の組み込みトランスフォーマの代わりに別の XSLT トランスフォーマをコンフィグレーションします。
- 特定のドキュメント タイプの処理に使用する XML パーサをコンフィグレーションします。

上記のすべての機能は、アプリケーションが WebLogic Server の本バージョンに付属の標準 Java API for XML Processing（JAXP）を使用する場合に利用できます。これらの機能は、サーバサイドだけで使用できます。

外部エンティティの解決のコンフィグレーション用 XML レジストリ

WebLogic XML では、XML レジストリを使用して外部エンティティを解決できます。外部エンティティの例が、XML ドキュメントの検証に使用する DTD ファイルです。この機能を使用するには、Administration Console を開き、XML レジストリで、外部エンティティに関連付ける パブリック `ID` または システム `ID` を入力します。

外部エンティティをローカルに格納することに加えて、URL などの HTTP インタフェースをサポートする外部レポジトリから外部エンティティを取得およびキャッシュするように WebLogic Server をコンフィグレーションできます。WebLogic Server のコンフィグレーションで、メモリやディスクにある外部エンティティをキャッシュし、有効期限が過ぎたと判断するまでエンティティをキャッシュ内にどの程度の期間とどめるかを指定できます。

XML レジストリを使用した外部エンティティの解決の詳細については、4-12 ページの「外部エンティティのコンフィグレーション タスク」を参照してください。

コード例

WebLogic Server には、XML ドキュメントの解析および変換のサンプルがあります。

XML ドキュメント変換のサンプルは、***BEA_HOME*\samples\examples\xml** ディレクトリにあります。***BEA_HOME*** は、WebLogic Server のメイン インストール ディレクトリです。

サンプルの構築方法と実行方法の詳細については、ブラウザで Web ページ *BEA_HOME*\samples\examples\xml\package-summary.html にアクセスしてください。

XML ファイルの編集

XML ファイルを編集するには、完全な Java ベースの XML スタンドアロン エディタである BEA XML エディタを使用します。この XML エディタは、XML ファイルを作成および編集するためのシンプルでユーザフレンドリなツールです。このツールでは、XML ファイルの中身を、階層的な XML ツリー構造と実際の XML コードの両方で表示します。ドキュメントを 2 通りに表示することにより、以下の 2 つの方法で XML ドキュメントを編集できます。

- 階層ツリー表示では、階層 XML ツリー構造の各ポイントでいくつかの指定可能な機能を使用する形で、構造化された制約のある編集が可能です。指定可能な機能は、構文的に決定されており、指定されているものがある場合は XML ドキュメントの DTD またはスキーマに従っています。
- XML コード表示では、データを自由に編集できます。

BEA XML エディタは、指定した DTD または XML スキーマを基に XML コードを検証します。

BEA XML エディタの使用方法の詳細については、オンライン ヘルプを参照してください。

BEA XML エディタは、BEA dev2dev からダウンロードできます。

XML について学習するには

XML について学習するには、以下のオンラインコースおよびチュートリアルを参照してください。

- A Technical Introduction to XML
- XML Authoring Tutorial
- Working with XML and Java
- Tutorials for using the Java 2 platform and XML technology
- XML, Java, and the Future of the Web

- 『XML Bible』の第 14 章「XSL Transformations」
- 『XSL Tutorial』、Miloslav Nic 著
- SAX 2.0: The Simple API for XML
- Document Object Model (DOM)

2 WebLogic Server による XML アプリケーションの開発

以降の節では、Java プログラミング言語と WebLogic Server を使用して XML アプリケーションを開発する方法について説明します。ここでは、Java サーブレットと JavaServer Pages（JSP）を使用して Java アプリケーションを記述する方法について理解していることを前提にしています。サーブレットと JSP アプリケーションの記述方法については、『WebLogic HTTP サーブレット プログラマーズ ガイド』と『WebLogic JSP プログラマーズ ガイド』を参照してください。

- XML アプリケーションの開発 : 主な手順
- XML ドキュメントの解析
- XML ドキュメントの生成
- JAXP による XML データの変換
- JSP タグによる XML ドキュメントの変換
- 組み込みトランスフォーマ以外のトランスフォーマの使用

XML アプリケーションの開発 : 主な手順

WebLogic Server XML サブシステムを使用して XML アプリケーションを開発する場合、通常は以下のプログラミング タスクの一部または全部を実行します。

1. XML ドキュメントを解析します。

   XML ドキュメントは、いくつかのソースから構成されています。たとえば、クライアントから XML ドキュメントを受信するサーブレットを開発し、サーブレットまたはその他の EJB などから XML ドキュメントを受信する EJB を記述できます。各インスタンスで、データを操作できるように XML ドキュメントを解析しておくこともできます。

このタスクの詳細については、2-2 ページの「XML ドキュメントの解析」を参照してください。

2. 新しい XML ドキュメントを生成します。

   サーブレットまたは EJB が XML ドキュメントを受信および解析し、何らかの形でデータを操作したら、サーブレットまたは EJB は、新しい XML ドキュメントを生成してクライアントに返すか、他の EJB に渡さなければならないことがあります。

   このタスクの詳細については、2-10 ページの「XML ドキュメントの生成」を参照してください。

3. XML データを他の形式に変換します。

   XML ドキュメントを解析するか、新しい XML ドキュメントを生成したら、サーブレットまたは EJB は、その XML を HTML、WML、またはプレーンテキストなどの形式に変換しなければならないことがあります。

   このタスクの詳細については、2-13 ページの「JAXP による XML データの変換」を参照してください。

XML ドキュメントの解析

この節では、XML ドキュメントの解析方法について説明します。

前述のように、Administration Console の XML レジストリを使用して以下の項目をコンフィグレーションします。

- doctype ごとのパーサ。指定した doctype の組み込みパーサの代わりに使用されます。
- 外部エンティティの解決。XML ドキュメントの解析中に外部ファイルを見つけるように要求された場合に XML パーサが実行する処理です。

Administration Console でこれらのタスクを実行する方法については、第 4 章「WebLogic Server XML の管理」を参照してください。

SAX モードによる XML ドキュメント解析の完全なサンプルについては、*BEA_HOME*\samples\examples\xml\sax ディレクトリを参照してください。*BEA_HOME* は、WebLogic Server のメイン インストール ディレクトリです。

SAX モードで JAXP を使用した XML ドキュメントの解析

次のコード例は、SAX パーサ ファクトリをコンフィグレーションして検証パーサを作成する方法を示したものです。また、MyHandler クラスをパーサに登録する方法も示しています。MyHandler クラスは、SAX 解析イベントまたはエラーのカスタム動作を提供するために DefaultHandler クラスの任意のメソッドをオーバーライドできます。

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

...
MyHandler handler = new MyHandler();
// MyHandler は org.xml.sax.helpers.DefaultHandler を拡張

  //SAXParserFactory のインスタンスを取得
  SAXParserFactory spf = SAXParserFactory.newInstance();
  // 検証パーサを指定
  spf.setValidating(true); // DTD をロードする必要がある
  // ファクトリから SAX パーサのインスタンスを取得
  SAXParser sp = spf.newSAXParser();
  // ドキュメントを解析
  sp.parse("http://server/file.xml", handler);
...
```

注意： 組み込みパーサ以外のパーサを使用する場合は、WebLogic Server Administration Console を使用して XML レジストリでパーサを指定します。指定しなかった場合、SaxParserFactory.newInstance メソッドは組み込みパーサを返します。組み込みパーサ以外のパーサを使用する WebLogic Server のコンフィグレーション手順については、4-5 ページの「組み込み以外のパーサまたはトランスフォーマのコンフィグレーション」を参照してください。

SAX モードによる XML ドキュメント解析の完全なサンプルについては、*BEA_HOME*\samples\examples\xml\sax ディレクトリを参照してください。*BEA_HOME* は、WebLogic Server のメイン インストール ディレクトリです。

DOM モードで JAXP を使用した XML ドキュメントの解析

次のコード例では、XML ドキュメントを解析し、DocumentBuilder オブジェクトから org.w3c.dom.Document ツリーを作成する方法を示します。

```
import javax.xml.parsers.DocumentBuidler;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

...
//DocumentBuilderFactory のインスタンスを取得
DocumentBuilderFactory dbf =
                  DocumentBuilderFactory.newInstance();
// 検証パーサを指定
dbf.setValidating(true); // DTD をロードする必要がある
// ファクトリから DocumentBuilder のインスタンスを取得
DocumentBuilder db = dbf.newDocumentBuilder();
// ドキュメントを解析
Document doc = db.parse(inputFile);
...
```

注意： 組み込みパーサ以外のパーサを使用する場合は、WebLogic Server Administration Console でパーサを指定します。指定しなかった場合、DocumentBuilderFactory.newInstance メソッドは組み込みパーサを返します。組み込みパーサ以外のパーサを使用する WebLogic Server のコンフィグレーション手順については、4-5 ページの「組み込み以外のパーサまたはトランスフォーマのコンフィグレーション」を参照してください。

DOM モードによる XML ドキュメント解析の完全なサンプルについては、*BEA_HOME*\samples\examples\xml\dom ディレクトリを参照してください。*BEA_HOME* は、WebLogic Server のメイン インストール ディレクトリです。

サーブレットでの XML ドキュメントの解析

Java サーブレット仕様バージョン 2.2 で、setAttribute メソッドと getAttribute メソッドに対するサポートが追加されました。属性は、リクエストに関連付けられたオブジェクトです。リクエスト オブジェクトは、クライアント リクエストからの全情報をカプセル化します。HTTP プロトコルでは、この情報は、リクエストの HTTP ヘッダとメッセージ本文を基にクライアントからサーバに転送されます。

WebLogic Server では、以下のメソッドを使用して XML ドキュメントを解析できます。setAttribute メソッドは、SAX モードの解析に使用し、getAttribute メソッドは DOM モードの解析に使用します。

サーブレットでの XML ドキュメント解析の完全なサンプルについては、*BEA_HOME*\samples\examples\xml\attributes ディレクトリを参照してください。*BEA_HOME* は、WebLogic Server のメイン インストール ディレクトリです。

org.xml.sax.DefaultHandler 属性を使用したドキュメントの解析

次のサンプル コードでは、setAttribute メソッドの使い方を示します。

```
import weblogic.servlet.XMLProcessingException;
import org.xml.sax.helpers.DefaultHandler;
...
public void doPost(HttpServletRequest request,
                              HttpServletResponse response)
   throws ServletException, IOException  {
   try {
     request.setAttribute("org.xml.sax.helpers.DefaultHandler",
                          new DefaultHandler());
    } catch(XMLProcessingException xpe) {
      System.out.println("Error in processing XML");
      xpe.printStackTrace();
      return;
    }
...
```

また、現在は非推奨となっているものの、org.xml.sax.HandlerBase 属性を使用して XML ドキュメントを解析することもできます。

```
request.setAttribute("org.xml.sax.HandlerBase",
                     new HandlerBase());
```

注意： このサンプル コードは、SAX と setAttribute メソッドでドキュメントを解析する単純な方法を示したものです。ドキュメントを解析するこのメソッドは、他のサーブレット ベンダではサポートされていない WebLogic Server の便利な機能です。したがって、アプリケーションを他のサーブレット プラットフォームで実行する場合は、この機能を使用しないでください。

org.w3c.dom.Document 属性を使用したドキュメントの解析

次のサンプル コードでは、getAttribute メソッドの使い方を示します。

```
import org.w3c.dom.Document;
import weblogic.servlet.XMLProcessingException;

...

public void doPost(HttpServletRequest request,
                           HttpServletResponse response)
throws ServletException, IOException  {

try {
   Document doc = request.getAttribute("org.w3c.dom.Document");
 } catch(XMLProcessingException xpe) {
   System.out.println("Error in processing XML");
   xpe.printStackTrace();
   return;
 }
...
```

注意： このサンプル コードは、DOM と getAttribute メソッドでドキュメントを解析する単純な方法を示したものです。ドキュメントを解析するこのメソッドは、他のサーブレット ベンダではサポートされていない WebLogic Server の便利な機能です。したがって、アプリケーションを他のサーブレット プラットフォームで実行する場合は、この機能を使用しないでください。

非検証パーサの検証

前述のとおり、整形式ドキュメントは、W3C の XML 1.0 勧告のルールに従っている、構文的に正しいドキュメントのことです。有効なドキュメントは、DTD またはスキーマで指定した制約に従っているドキュメントです。

非検証パーサは、ドキュメントが整形式かどうかを検証しますが、有効かどうかは検証しません。2-9 ページの「WebLogic FastParser の使用」で説明している WebLogic FastParser は、デフォルトでは非検証パーサです。

（検証パーサの使用を前提に）ドキュメント解析時の検証を有効にするには、以下を実行する必要があります。

- 以下の例で示すように、`SAXParserFactory.setValidating()` メソッドを true に設定します。

  ```
  SAXParserFactory factory = SAXParserFactory.newInstance();
  factory.setValidating(true);
  ```

- 解析する XML ドキュメントで、（インラインまたは参照で）DTD またはスキーマが定義されているようにします。

XML ドキュメント解析時のエンティティ解決の処理

この節では、XML パーサによる外部エンティティの識別および解決方法、および XML アプリケーションによる外部エンティティ解決のパフォーマンスを向上させる WebLogic Server の機能など、外部エンティティに関する一般的な情報について説明します。

XML ドキュメント解析時の外部エンティティ解決の完全なサンプルについては、*`BEA_HOME`*`\samples\examples\xml\entityresolution` ディレクトリを参照してください。*`BEA_HOME`* は、WebLogic Server のメイン インストール ディレクトリです。

外部エンティティに関する一般的な情報

外部エンティティは、XML ドキュメント内に記述されていないものの、XML ドキュメント内で参照されるさまざまなテキストです。実際のテキストは、同じコンピュータの他のファイル、Web 上など、どこにあってもかまいません。パーサは、ドキュメントの解析時に外部エンティティ参照に出会うと、参照されたテキストをフェッチし、テキストを XML ドキュメント内に配置してから、解析を続行します。外部エンティティの例が DTD です。XML ドキュメント内に DTD のテキストがすべて記述されているのではなく、別のファイルに格納されている DTD への参照が XML ドキュメント内にあります。

外部エンティティの識別方法には、システム ID とパブリック ID の 2 つがあります。システム ID は、URI で指定された位置を基に外部エンティティを参照します。パブリック ID は、外部で宣言されている名前を使用して情報を参照します。

次の例では、パブリック ID を使用して、J2EE アプリケーション アーカイブ（*.ear）を示す application.xml ファイルの DTD を参照する方法を示します。

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems,
Inc.//DTD J2EE Application 1.2//EN">
```

次の例は、システム ID のみによる外部 DTD の参照を示します。

```
<!DOCTYPE application SYSTEM
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">
```

次の例は、パブリック ID およびシステム ID の両方を使用する参照です。SYSTEM キーワードが省略されています。

```
<!DOCTYPE application
PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">
```

WebLogic Server のエンティティ解決機能の使用

以下の WebLogic Server 機能を使用すると、XML アプリケーションの外部エンティティ解決のパフォーマンスを向上させることができます。

- 外部エンティティのコピーを、WebLogic 管理サーバのホストとなるコンピュータに永続的に格納します。
- WebLogic Server が、URL などの HTTP インタフェースをサポートする外部レポジトリにある外部エンティティを自動的に取得およびキャッシュするように指定します。エンティティをキャッシュ先をメモリにするかディスクにするかを指定し、キャッシュされているエントリが自動的に更新されるキャッシュの有効期限を指定できます。

  この機能を使用すると、実際に外部エンティティをローカル コンピュータにコピーする必要はなくなります。XML アプリケーションは、ドキュメントを解析するたびにではなく、指定した期間でのみ外部エンティティを参照するので、アプリケーションのパフォーマンスを大幅に向上させることができ、必要に応じて外部エンティティを最新の状態で参照できます。

XML レジストリで、外部エンティティの場所（ローカルまたは URL）、およびどのキャッシュ オプションが Web 上のエンティティ用かを識別するエンティティ解決エントリを作成します。システム ID またはパブリック ID を使用して外

部エンティティ エントリを識別します。次に、XML ドキュメントで、この外部エンティティを参照すると、WebLogic Server は、ドキュメントの解析時にローカル コピーまたはキャッシュされたコピー（コンフィグレーションによる）を取り出します。

XML レジストリでの外部エンティティ レジストリ作成の詳細については、4-12 ページの「外部エンティティのコンフィグレーション タスク」を参照してください。

組み込みパーサ以外のパーサの使用

JAXP で XML ドキュメントを解析する場合、WebLogic Server XML レジストリ（Administration Console でコンフィグレーション）には以下の選択肢があります。

- 組み込みパーサをサーバ全体のパーサとして使用します。
- WebLogic FastParser をサーバ全体のパーサとしてコンフィグレーションします。
- 任意のパーサをサーバ全体のパーサとして使用します。
- システム ID、パブリック ID、またはルート タグに基づいた特定の DTD に対応するパーサをコンフィグレーションします。

XML レジストリで解析オプションをコンフィグレーションする手順については、4-4 ページの「XML パーサおよびトランスフォーマのコンフィグレーション タスク」を参照してください。

WebLogic FastParser の使用

WebLogic Server には、中小規模の XML ドキュメント（最大 10,000 個の要素）の検証用に特別に設計された高性能の非検証 XML パーサ（WebLogic FastParser）があります。大きなドキュメントになると、このパーサのパフォーマンスは、Apache Xerces などの標準パーサと変わらなくなります。

注意： WebLogic FastParser は、SAX スタイルの解析のみをサポートしています。

WebLogic FastParser を WebLogic Server 全体のパーサとして使用するように指定することもできます。または、4-4 ページの「XML パーサおよびトランスフォーマのコンフィグレーション タスク」で説明されているように、XML レジストリを使用することで特定の DOCTYPE で使用するように指定することもできます。[SAX パーサ ファクトリ] フィールドを `weblogic.xml.babel.jaxp.SAXParserFactoryImpl` に設定します。

注意： WebLogic Server の旧バージョンには、カスタム パーサの作成機能がありました。カスタマイズされたパーサの対象となる XML ドキュメントのタイプに応じた WebLogic FastParser を使用できるので、FastParser は、効果的にカスタマイズされたパーサ機能を置き換えます。カスタマイズされたパーサの生成機能はなくなっています。

XML ドキュメントの生成

この節では、DOM ドキュメント ツリーから JSP を使用して XML ドキュメントを生成する方法について説明します。

DOM ドキュメント ツリーからの XML の生成

この節では、DOM ドキュメント ツリーから XML ドキュメントを作成する以下の 2 つの方法について説明します。

- Apache `serialize` クラスを使用する場合
- JAXP `Transformer` クラスを使用する場合

Apache serialize クラスを使用する場合

DOM ドキュメント ツリーから XML ドキュメントを生成するには、`weblogic.apache.xml.serialize` クラスを使用すると、DOM ドキュメント ツリーを XML テキストに変換できます。このクラスの詳細については、`weblogic.apache.xml.serialize` の Javadoc を参照してください。

次のコード例では、このクラスの使い方を示します。

注意： 次の例では、JAXP を使用しない代わりに、Apache 固有の API を直接使用します。

```
package test;

import java.io.OutputStreamWriter;
import java.util.Date;
import java.text.DateFormat;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import weblogic.apache.xerces.dom.DocumentImpl;
import weblogic.apache.xml.serialize.DOMSerializer;
import weblogic.apache.xml.serialize.XMLSerializer;

public class WriteXML {

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

    // DOM ツリーを作成
    Document doc= new DocumentImpl();
    Element message = doc.createElement("message");
    doc.appendChild(message);
    Element text = doc.createElement("text");
    text.appendChild(doc.createTextNode("Hello world."));
    message.appendChild(text);
    Element timestamp = doc.createElement("timestamp");
    timestamp.appendChild(
      doc.createTextNode(
                DateFormat.getDateInstance().format(new Date()))
    );
    message.appendChild(timestamp);

    // DOM を XML テキストにシリアライズして stdout に出力
    DOMSerializer xmlSer =
      new XMLSerializer(new OutputStreamWriter(System.out),null);
    xmlSer.serialize(doc);
  }
}
```

JAXP トランスフォーマ クラスを使用する場合

コード例の次のセグメントで示すように、javax.xml.transform.Transformer クラスを使用して、DOM オブジェクトを XML ストリームにシリアライズできます。

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
```

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

...

TransformerFactory trans_factory =
TransformerFactory.newInstance();
Transformer xml_out = trans_factory.newTransformer();
Properties props = new Properties();
props.put("method", "xml");
xml_out.setOutputProperties(props);
xml_out.transform(new DOMSource(doc), new
StreamResult(System.out));
```

例では、Transformer.transform() は、DOM オブジェクトを XML ストリームに変換します。transform() メソッドは、doc 変数に格納されている DOM ツリーから作成された javax.xml.transform.dom.DOMSource オブジェクトを入力としてとり、それを javax.xml.transform.stream.StreamResult オブジェクトに変換して、結果の XML ドキュメントを標準出力に書き出します。

JSP での XML ドキュメントの生成

通常、JSP は HTML の生成に使用しますが、JSP を使用して XML ドキュメントを生成することもできます。

JSP による XML の生成には、以下のように JSP ページのコンテンツ タイプを設定する必要があります。

```
<%@ page contentType="text/xml"%>
    ... XML document
```

次のコードでは、JSP による XML ドキュメントの生成例を示します。

```
<?xml version="1.0">

<%@ page contentType="text/xml"
import="java.text.DateFormat,java.util.Date" %>

<message>
  <text>
    Hello World.
  </text>
  <timestamp>
<%
out.print(DateFormat.getDateInstance().format(new Date()));
%>
```

```
    </timestamp>
  </message>
```

JSP による XML の生成の詳細については、
http://java.sun.com/products/jsp/html/JSPXML.html を参照してください。

JAXP による XML データの変換

変換は、XML ドキュメント（変換元）を他の形式（変換結果）に変換することです。通常は、別の XML ドキュメント、HTML、無線用マーク付け言語（Wireless Markup Language : WML）に変換されます。バージョン 1.1 の JAXP では、プラグイン可能の変換が提供されています。つまり、JAXP 準拠の任意のトランスフォーマ エンジンを使用できます。

JAXP は、XML データをさまざまな形式に変換するために以下のインタフェースを提供します。

- `javax.xml.transform` ： このパッケージには、ドキュメントを変換するための汎用 API があります。SAX か DOM かに関係なく、変換元および変換結果の形式を推測し、できる限り少ない候補に絞り込みます。
- `javax.xml.transform.stream` : このパッケージは、ストリーム固有および URI 固有の変換 API を実装します。特に、`InputStreams` と URL を変換元、`OutputStreams` と URL を変換結果として指定できる `StreamSource` クラスおよび `StreamResult` クラスを定義します。
- `javax.xml.transform.dom` : このパッケージは、DOM 固有の変換 API を実装します。特に、DOM ツリーを変換元か変換結果、またはその両方として指定できる `DOMSource` クラスおよび `DOMResult` クラスを定義します。
- `javax.xml.transform.sax` : このパッケージは、SAX 固有の変換 API を実装します。特に、`org.xml.sax.ContentHandler` イベントを変換元か変換結果、またはその両方として指定できる `SAXSource` クラスおよび `SAXResult` クラスを定義します。

変換には、入力と出力の多くの組み合わせがあります。

XML ドキュメント変換の完全なサンプルについては、
`BEA_HOME``\samples\examples\xml\xslt` ディレクトリを参照してください。
`BEA_HOME` は、WebLogic Server のメイン インストール ディレクトリです。

JAXP による XML ドキュメント変換の例

抜粋した次のコード例では、JAXP を使用して myXMLdoc.xml を mystylesheet.xsl スタイルシートを使用する別の XML ドキュメント変換する方法を示します。

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

Transformer trans;
TransformerFactory factory = TransformerFactory.newInstance();
String stylesheet = "file://stylesheets/mystylesheet.xsl";
String xml_doc = "file://xml_docs/myXMLdoc.xml";

trans = factory.newTransformer(new StreamSource(stylesheet));
trans.transform(new StreamSource(xml_doc),
                new StreamResult(System.out));
```

DOM ドキュメントを XML ストリームに変換する方法の例については、2-11 ページの「JAXP トランスフォーマ クラスを使用する場合」を参照してください。

Xalan API から JAXP 1.1 API への変換

アプリケーションに Xalan 固有のコードが含まれている場合、それを JAXP を使用するように変更することをお勧めします。

この節では、Xalan API から JAXP に変換するために XML アプリケーションで必要な変更を簡単に説明します。この節では、よく似た変換タスクを実行する 2 つのコード セグメントを比較します。一方のコード セグメントでは Xalan API を直接使用し、もう一方では JAXP を使用します。

次の Java コード セグメントでは JAXP を使用しています。

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

...

Transformer trans;
TransformerFactory factory = TransformerFactory.newInstance();
```

```
String stylesheet = "file://stylesheets/mystylesheet.xsl";
String xml_doc = "file://xml_docs/myXMLdoc.xml";

trans = factory.newTransformer(new StreamSource(stylesheet));
trans.transform(new StreamSource(xml_doc),
                new StreamResult(System.out));
```

次の Java コード セグメントでは Xalan API を直接使用しています。

```
 /*
 * このサンプル コードは、Apache Software Foundation で提供され
 * ているコードからの引用。Apache SoftWare Foundation のために多
 * くの個人から有志の協力を得て完成した。元のソフトウェア著作権は以下の
 * とおり
 * copyright (c) 1999, Lotus Development Corporation.,
 * http://www.lotus.com。Apache Software Foundation に関する
 * 情報については <http://www.apache.org/> を参照
 */

import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessor;

...

XSLTProcessor processor = XSLTProcessorFactory.getProcessor();

String stylesheet = "file://stylesheets/mystylesheet.xsl";
String xml_doc = "file://xml_docs/myXMLdoc.xml";

processor.process(new XSLTInputSource(xml_doc),
                      new XSLTInputSource(stylesheet),
                      new XSLTResultTarget(System.out));
```

次の表では、前の例で XML ドキュメントの変換に使用した Xalan と JAXPI インタフェースおよびメソッドの名前を示します。この表を基に、既存の Xalan アプリケーションを完全な JAXP アプリケーションに変換してください。

注意： この表は、Xalan と JAXP の間のマッピング全体ではなく、前の例で使用したメイン クラスおよびメソッドのみを示します。各 API の詳細については、Apache および Sun の Web サイト（http://www.apache.org および http://java.sun.com/xml/index.html）を参照してください。

クラスまたはインタフェースの説明	Xalan 1.X	JAXP 1.1
XML ドキュメントの変換に使用するメイン クラス	`XSLTProcessor`	`Transformer`

クラスまたはインタフェースの説明	Xalan 1.X	JAXP 1.1
トランスフォーマ オブジェクトの作成に使用するファクトリ クラス	`XSLTProcessorFactory`	`TransformerFactory`
ファクトリの新しいインスタンスの作成に使用するメソッド	なし	`TransformerFactory.newInstance()`
新しいトランスフォーマ オブジェクトの作成に使用するメソッド	`XSLTProcessorFactory.getProcessor()`	`TransformerFactory.newTransformer()`
XML ドキュメントまたは XSL スタイルシートなどの変換元を保持するクラス	`XSLTInputSource`	`StreamSource`
変換結果を保持するクラス	`XSLTResultTarget`	`StreamResult`
変換を実行するメソッド	`XSLTProcessor.process()`	`Transformer.transform()`

JSP タグによる XML ドキュメントの変換

WebLogic Server では、JSP 内から XSLT トランスフォーマに簡単にアクセスするための小規模な JSP タグ ライブラリが提供されます。このタグを使用すると XML ドキュメントを HTML、WML などに変換できますが、必須ではありません。

JSP タグ ライブラリは、メイン タグの `x:xslt` と `x:xslt` タグの内部で使用できる 2 つのサブタグ（`x:stylesheet` および `x:xml`）で構成されます。

XSLT JSP タグ構文

XSLT JSP タグ構文は XML を基にしています。JSP タグは、開始タグ、本文（省略可能）、対応する終了タグで構成されます。開始タグには要素名と属性（省略可能）が含まれます。

次の構文では、WebLogic Server で提供される 3 つの XSLT JSP タグを JSP で使用する方法について示します。属性は省略可能です。また、`x:stylesheet` および `x:xml` タグも省略可能です。構文に続く表では、`x:xslt` および `x:stylesheet` タグの属性について示します。`x:xml` タグには属性はありません。

```
<x:xslt  [xml="uri of XML file"]
         [media="media type to determine stylesheet"]
         [stylesheet="uri of stylesheet"]
    <x:xml>In-line XML goes here
    </x:xml>
    <x:stylesheet [media="media type to determine stylesheet"]
                  [uri="uri of stylesheet"]
    </x:stylesheet>
</x:xslt>
```

次の表に、x:xslt タグの属性を示します。

x:xslt タグの属性	必須	データ型	説明
xml	いいえ	String	変換する XML ファイルの場所を指定する。場所は、タグが使用される Web アプリケーションのドキュメント ルートへの相対パス。
media	いいえ	String	HTML または WML など、ドキュメントの出力タイプを定義する。このタイプによって、XML ドキュメントの変換時に使用するスタイルシートが決定される。 この属性は、x:xslt タグの本文の内部で x:stylesheet タグで囲まれた media 属性と組み合わせて使用できる。x:xslt タグの media 属性の値は x:stylesheet タグに囲まれた media 属性の値と比較される。値が等しい場合、x:stylesheet タグの uri 属性で指定されたスタイルシートが XML ドキュメントに適用される。 **注意 :** 同じ x:xslt タグ内で media 属性と stylesheet 属性の両方を設定するとエラーが発生する。
stylesheet	いいえ	String	XML ドキュメントの変換に使用するスタイルシートの場所を指定する。場所は、タグが使用される Web アプリケーションのドキュメント ルートへの相対パス。 **注意 :** 同じ x:xslt タグ内で media 属性と stylesheet 属性の両方を設定するとエラーが発生する。

次の表に、x:stylesheet タグの属性を示します。

x:stylesheet タグの属性	必須	データ型	説明
media	いいえ	String	HTML または WML など、ドキュメントの出力タイプを定義する。このタイプによって、XML ドキュメントの変換時に使用するスタイルシートが決定される。 この属性は x:xslt タグに囲まれた media 属性と組み合わせて使用する。x:xslt タグの media 属性の値は x:stylesheet タグに囲まれた media 属性の値と比較される。値が等しい場合、x:stylesheet タグの uri 属性で指定されたスタイルシートが XML ドキュメントに適用される。
uri	いいえ	String	media 属性の値が、x:xslt タグに囲まれた media 属性の値と一致する場合に使用するスタイルシートの場所を指定する。場所は、タグが使用される Web アプリケーションのドキュメント ルートへの相対パス。

XSLT JSP タグの使い方

x:xslt タグは本文の有無に関係なく使用でき、属性は省略可能です。この節では、本文や 1 つまたは複数の属性を指定したかどうかによってタグの動作を決定するルールについて説明します。

x:xslt タグが空タグ（本文なし）の場合、以下の説明が適用されます。

- 属性が設定されていない場合、XML ドキュメントはサーブレットのパスとデフォルトのメディア スタイルシートで処理されます。XML ファイルで、<?xml-stylesheet> 処理指示を使用してデフォルトのメディア スタイルシートを指定します。デフォルトのスタイルシートには media 属性はありません。

  この処理のタイプでは、XSLT 処理を実行するファイル サーブレットとしてタグ拡張を含む JSP ページを登録できます。

- media 属性のみが設定されている場合、XML ドキュメントは、サーブレットのパスと指定されたメディア タイプを使用して処理されます。x:xslt タグの media タイプ属性の値は、XML ドキュメントにある

<?xml-stylesheet> 処理指示の media 属性の値と比較されます。一致すると、対応する場合は、スタイルシートが適用されます。一致しない場合はデフォルトのメディア スタイルシートが使用されます。メディア タイプ属性は、ドキュメントの出力タイプ（XML、HTML、ポストスクリプト、WML など）の定義に使用します。この機能を使用すると、ドキュメントの出力タイプごとにスタイルシートをまとめることができます。

- xml 属性のみが設定されている場合、指定した XML ドキュメントはデフォルトのメディア スタイルシートで処理されます。
- media および xml 属性を設定すると、指定した XML ドキュメントは指定したメディア タイプを使用して処理されます。
- stylesheet 属性が定義されている場合、XML ドキュメントは指定したスタイルシートで処理されます。

警告： 同じ x:xslt タグ内で media 属性と stylesheet 属性の両方を設定するとエラーが発生します。

本文がある XSLT JSP タグは、<x:xml> タグまたは <x:stylesheet> タグを含んでいる場合があります。以下の説明が適用されます。

- <x:xml> タグを使用すると、XML ドキュメントをインライン処理用に指定できます。このタグには属性はありません。
- <x:stylesheet> タグは、属性を指定しない場合、デフォルトのスタイルシートをインラインで指定できます。
- <x:stylesheet> タグの uri 属性を使用して、デフォルト スタイルシートの場所を指定します。
- さまざまなメディア タイプごとに異なるスタイルシートを指定する場合は、media 属性に異なる値を指定して、複数の <x:stylesheet> タグを使用できます。タグの本文でメディア タイプごとのスタイルシートを指定したり、uri 属性を使ってスタイルシートの場所を指定したりできます。

XSLT JSP タグによる XML ドキュメントの変換

XSLT JSP タグで XML ドキュメントを変換するには、以下の手順を実行します。

1. *BEA Home*\wlserver6.0\ext ディレクトリの xmlx.zip ファイルを開いて、xmlx-tags.jar ファイルを Web アプリケーションの \lib ディレクトリに移動します。*BEA Home* は、WebLogic Server 配布キットをインストールした最上位ディレクトリです。
2. <taglib> エントリを web.xml ファイルに追加します。次に例を示します。

   ```
   <taglib>
     <taglib-uri>xmlx.tld</taglib-uri>
     <taglib-location>/WEB-INF/lib/xmlx-tags.jar</taglib-location>
   </taglib>
   ```

3. タグを使用するには、次の行を JSP ページに追加します。

   ```
   <%@ taglib uri="xmlx.tld" prefix="x"%>
   ```

4. トランスフォーマをコンフィグレーションします。以下の手順では、トランスフォーマのコンフィグレーションの一般的な方法を示します。

   a. 次のコード行を入力して xslt.jsp ファイルを作成します。

   ```
   <%@ taglib uri="xmlx.tld" prefix="x"%><x:xslt/>
   ```

   b. 次のように、xslt.jsp ファイルを web.xml ファイルに登録します。

   ```
   <servlet>
      <servlet-name>myxsltinterceptor</servlet-name>
      <jsp-file>xslt.jsp</jsp-file>
   </servlet>
   <servlet-mapping>
      <servlet-name>myxsltinterceptor</servlet-name>
      <url-pattern>/xslt/*</url-pattern>
   </servlet-mapping>
   ```

   c. XML、DTD、XSL ドキュメントまたはサーブレットを Web アプリケーションに配置します。

   d. xslt プレフィックスを XML ドキュメントのパス名に追加（たとえば、docs/fred.xml を xslt/docs/fred.xml に変更）し、ドキュメントにアクセスします。web.xml ファイルの <url-pattern> エントリによって、WebLogic Server は、XML ドキュメントで自動的に XSLT トランスフォーマを実行し、ドキュメントにデフォルトのスタイルシートを設定します。

e. メディア タイプを定義するには、XML ドキュメントのメディア タイプと出力のコンテンツ タイプを指定するコードを JSP に追加します。

f. メディア タイプを xslt タグに渡して、応答オブジェクトのコンテンツ タイプを設定します。

注意： 他の形の XSLT JSP タグは、スタイルシートが XML ドキュメントで指定されていない場合または XML スタイルシートがインラインで生成される場合に使用されます。

JSP での XSLT JSP タグの使用例

JSP から抜粋した次のコードは、XSLT JSP タグを使用し、JSP を要求するクライアントのタイプにしたがって、XML を HTML または WML に変換する方法を示します。JSP は、クライアントがブラウザの場合は HTML を返し、無線デバイスの場合は WML を返します。

最初に、JSP は HttpServletRequest オブジェクトの getHeader() メソッドを使用して、JSP を要求するクライアントのタイプを識別します。次に、*myMedia* 変数を wml または html に適切に設定します。JSP で *myMedia* 変数を html に設定した場合は、*content* 変数に含まれる XML ドキュメントに html.xsl スタイルシートが適用されます。同様に、JSP で *myMedia* 変数を wml に設定した場合は、wml.xsl スタイルシートが適用されます。

```
<%
   String clientType = request.getHeader("User-Agent");
   // デフォルトは WML クライアント
   String myMedia = "wml";

   // クライアントが HTML ブラウザの場合
   if (clientType.indexOf("Mozilla") != -1) {
      myMedia = "http"
   }
%>

<x:xslt media="<%=myMedia%>">
  <x:xml><%=content%></x:xml>
  <x:stylesheet media="html" uri="html.xsl"/>
  <x:stylesheet media="wml"  uri="wml.xsl"/>
</x:xslt>
```

組み込みトランスフォーマ以外のトランスフォーマの使用

WebLogic Server の XML レジストリ（Administration Console でコンフィグレーション）では、以下の項目をコンフィグレーションします。

- 組み込みトランスフォーマをサーバ全体のトランスフォーマとして使用します。
- 組み込みトランスフォーマ以外のトランスフォーマをサーバ全体のトランスフォーマとして使用します。トランスフォーマは JAXP 準拠でなくてはなりません。

XML レジストリで変換オプションをコンフィグレーションする手順については、4-4 ページの「XML パーサおよびトランスフォーマのコンフィグレーション タスク」を参照してください。

3　XML プログラミング手法

以下の節では、XML データを処理する J2EE アプリケーションを開発するための特定の XML プログラミング手法について説明します。

- サーブレットおよび JSP に対する XML の送受信
- JMS アプリケーションでの XML ドキュメントの処理
- HTTP インタフェースを持たない外部エンティティへのアクセス
- XML ドキュメント ヘッダ情報

サーブレットおよび JSP に対する XML の送受信

一般的な J2EE アプリケーションでは、クライアント アプリケーションは、XML データを処理するサーブレットまたは JSP に XML データを送信します。次に、サーブレットまたは JSP は、そのデータを JMS 送り先や EJB などの別の J2EE コンポーネントに送信するか、または処理した XML データを別の XML ドキュメントの形でクライアントに返します。

Java クライアントからの XML データを WebLogic Server がホストのサーブレットまたは JSP に対して送受信するには、`java.net.URLConnection` クラスを使用します。このクラスは、アプリケーションと URL（この場合は、サーブレットまたは JSP を呼び出す URL）の間の通信リンクを表します。`URLConnection` クラスのインスタンスは、HTTP POST メソッドを使用して XML ドキュメントを送信します。

WebLogic XML サンプルから抜粋した次の Java クライアント プログラムでは、JSP に対して XML を送受信する方法を示しています。

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Client {

  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.out.println("Usage:  java examples.xml.Client URL Filename");
    }
    else {
      try {
        URL url = new URL(args[0]);
        String document = args[1];
        FileReader fr = new FileReader(document);
        char[] buffer = new char[1024*10];
        int bytes_read = 0;
        if ((bytes_read = fr.read(buffer)) != -1)
          {
            URLConnection urlc = url.openConnection();
            urlc.setRequestProperty("Content-Type","text/xml");
            urlc.setDoOutput(true);
            urlc.setDoInput(true);
            PrintWriter pw = new PrintWriter(urlc.getOutputStream());

            // XML を JSP に送信
            pw.write(buffer, 0, bytes_read);
            pw.close();

            BufferedReader in = new BufferedReader(new
InputStreamReader(urlc.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
                      System.out.println(inputLine);

            in.close();
            }
          }
          catch (Exception e) {
          e.printStackTrace();
          }
       }
    }
}
```

この例ではまず、引数リストから URL を使用して JSP への URL 接続を開き、接続から出力ストリームを取得し、引数リストで提供されている XML ドキュメントを出力ストリームに出力して、XML データを JSP に送信する方法を示します。次に、`URLConnection` クラスの `getInputStream()` メソッドで、JSAP がクライアント アプリケーションに返す XML データを読み取る方法を示します。

サンプル JSP から抜粋した次のコード セグメントでは、JSP がクライアント アプリケーションから XML データを受信し、XML ドキュメントを解析して、XML データを返す方法を示しています。

```
BufferedReader br = new BufferedReader(request.getReader());
DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
DocumentBuilder db = fact.newDocumentBuilder();
Document doc = db.parse(new InputSource(br));

...

PrintWriter responseWriter = response.getWriter();
responseWriter.println("<?xml version='1.0'?>");
```

...

WebLogic サーブレットと JSP アプリケーションのプログラミングの詳細については、『WebLogic HTTP サーブレット プログラマーズ ガイド』と『WebLogic JSP プログラマーズ ガイド』を参照してください。

JMS アプリケーションでの XML ドキュメントの処理

WebLogic Server は、特に JMS アプリケーションで XML ドキュメントを処理するために一部の Java Message Service（JMS）クラスに対して以下のエクステンションを提供します。

- `weblogic.jms.extensions.WLSession`。JMS クラス `javax.jms.Session` を拡張します。
- `weblogic.jms.extensions.WLQueueSession`。JMS クラス `javax.jms.QueueSession` を拡張します。

- `weblogic.jms.extensions.WLTopicSession`。JMS クラス `javax.jms.TopicSession` を拡張します。
- `weblogic.jms.extensions.XMLMessage`。JMS クラス `javax.jms.TextMessage` を拡張します。

より一般的な `TextMessage` クラスではなく、`XMLMessage` クラスを使用して、JMS アプリケーションで XML ドキュメントを送受信する場合、XML 固有のメッセージ セレクタを使用して不要なメッセージをフィルタ処理できます。特に、`JMS_BEA_SELECT` メソッドを使用すると、XML ドキュメントの XML フラグメントを検索するための XPath クエリを指定できます。クエリの結果に基づいて、メッセージ コンシューマがメッセージを受信しないよう決定することもあります。これにより、ネットワーク トラフィックを軽減し、JMS アプリケーションのパフォーマンスを向上させることができます。

`XMLMessage` クラスを使用して JMS アプリケーションに XML メッセージを含めるには、`WLQueueSession` オブジェクトまたは `WLTopicSession` オブジェクトのいずれかを作成する必要があります。どちらのオブジェクトを作成するかは、JMS `Connection` を作成した後に、汎用オブジェクト `QueueSession` または `TopicSession` ではなく、JMS キューまたはトピックを使用するかどうかによります。次に、`WLSession` インタフェースの `createXMLMessage()` メソッドを使用して `XMLMessage` オブジェクトを使用します。

JMS アプリケーションで `XMLMessage` オブジェクトを使用する際の詳細については、『WebLogic JMS プログラマーズ ガイド』を参照してください。

HTTP インタフェースを持たない外部エンティティへのアクセス

WebLogic Server は、外部レポジトリにある外部エンティティに、エンティティを返す URL などの HTTP インタフェースがある限り、その外部エンティティを自動的に取得およびキャッシュできます。XML レジストリで外部エンティティをコンフィグレーションする際の詳細については、4-12 ページの「外部エンティティのコンフィグレーション タスク」を参照してください。

HTTP インタフェースを持たないレポジトリに格納されている外部エンティティにアクセスする場合、それを 1 つ作成する必要があります。たとえば、システム ID、パブリック ID、DTD のテキストの列を持つデータベース テーブルに XML ドキュメント用の DTD を格納しているとします。WebLogic XML アプリケーションから外部エンティティとして DTD にアクセスする場合、JDBC を使用してデータベースの DTD にアクセスするサーブレットを作成できます。

URL でサーブレットを呼び出すので、ここで外部エンティティに対する HTTP インタフェースを持っていることになります。XML レジストリでエンティティ レジストリ エントリを作成する場合、外部エンティティの場所としてサーブレットを呼び出す URL を指定します。WebLogic Server は、この外部エンティティへの参照を含む XML ドキュメントを解析しているときにサーブレットを呼び出し、サーブレットがデータベースのクエリに内部で使用するパブリック ID およびシステム ID をサーブレットに渡します。

XML ドキュメント ヘッダ情報

ドキュメント内の実際のデータをすべて取得するのではなく、ルート要素、システム ID、パブリック ID などの XML ドキュメントに関する情報のみが必要なことがあります。この場合、ドキュメントの完全な解析は不要です。特に、XML ドキュメントのサイズが非常に大きな場合は、アプリケーションのパフォーマンスが低下する可能性があります。

XML ドキュメントを解析する代わりに、`org.xml.sax.InputSource` クラスに対する WebLogic Server の拡張である `weblogic.xml.sax.XMLInputSource` クラスを使用すると、XML ドキュメントに関するヘッダ情報を取得できます。次のサンプル コード セグメントでは、このクラスの使い方を示します。

```
import weblogic.xml.sax.XMLInputSource;
...

    String inputXML = "file://xml_docs/myXMLdoc.xml";
    XMLInputSource xis = new XMLInputSource(inputXML);
    String docType = xis.getRootTag();
    String publicID = xis.getPublicId();
    String systemID = xis.getSystemId();
    String namespaceURI = xis.getNamespaceURI();
```

`weblogic.xml.sax.XMLInputSource` クラスの詳細については、WebLogic Server API リファレンスを参照してください。

4 WebLogic Server XML の管理

以下の節では、WebLogic Server での XML の管理について説明します。

- WebLogic Server XML の管理の概要
- XML パーサおよびトランスフォーマのコンフィグレーション タスク
- 外部エンティティのコンフィグレーション タスク

WebLogic Server XML の管理の概要

Administration Console から XML レジストリにアクセスし、XML レジストリで WebLogic Server を XML アプリケーション用にコンフィグレーションします。

ブラウザで Administration Console を呼び出すには、次の URL を指定します。

```
http://host:port/console
```

この URL の説明は以下のとおりです。

- *host* は、WebLogic 管理サーバが動作しているコンピュータです。
- *port* は、接続リクエストに対して WebLogic 管理サーバがリスニングを行うポート番号です。WebLogic 管理サーバのデフォルト ポート番号は 7001 です。

XML の管理タスク

XML レジストリは、Administration Console を通じて作成、コンフィグレーション、および使用します。Administration Console の XML レジストリを使用するメリットは以下のとおりです。

- XML アプリケーションで JAXP を使用している場合、XML レジストリのコンフィグレーションの変更が実行時に自動的に有効になります。
- XML レジストリを変更する場合、XML アプリケーション コードを変更する必要はありません。
- エンティティの解決がローカルで実行されます。XML レジストリでは、エンティティのローカル コピーを定義できます。または、WebLogic Server が Web からエンティティを指定した期間キャッシュし、Web 上のエンティティではなく、そのキャッシュしたコピーを使用するように指定することもできます。

XML レジストリを使用すると、以下の指定が可能になります。

- 組み込みパーサの代わりとなるサーバ全体の XML パーサ。
- ドキュメント単位の XML パーサ。
- 組み込みトランスフォーマの代わりとなるサーバ全体のトランスフォーマ。
- エンティティのローカル コピーで解決される外部エンティティ。外部エンティティを指定すると、管理サーバでは、エンティティのローカル コピーがファイル システムに保存され、解析時にサーバのパーサに自動的に配布されます。この機能を使用すれば、SAX EntityResolvers の作成と設定が不要になります。
- Web からの取得後に WebLogic Server によってキャッシュされる外部エンティティ。WebLogic Server が再取得するまでこれらの外部エンティティがキャッシュされる期間、および WebLogic が最初にエンティティを取得するタイミング（アプリケーションの実行時か、WebLogic Server の起動時か）を指定します。

これらの機能は、サーバサイドだけで使用できます。

XML レジストリの仕組み

XML レジストリを必要な数だけ作成できますが、WebLogic Server の特定のインスタンスに関連付けることができる XML レジストリの数は 1 つだけです。

WebLogic Server のインスタンスに関連付けられている XML レジストリがない場合、ドキュメントの解析および変換に組み込みパーサおよびトランスフォーマが使用されます。さらに、XML アプリケーションのパフォーマンスを向上させるために外部エンティティ解決をコンフィグレーションすることはできません。

XML レジストリを WebLogic Server のインスタンスに関連付けると、すべての XML コンフィグレーション オプションが、そのサーバを使用している XML アプリケーションで利用可能になります。

指定した XML レジストリに対して以下の 2 種類のエントリをコンフィグレーションできます。

- パーサおよびトランスフォーマをコンフィグレーション
- 外部エンティティの解決をコンフィグレーション

注意： XML レジストリでは、大文字と小文字が区別されます。たとえば、ルート要素が <CAR> の XML ドキュメント タイプのパーサをコンフィグレーションしている場合、[ルート要素タグ] フィールドには、「car」または「Car」ではなく「CAR」と入力する必要があります。

XML レジストリ内のパーサの選択

JAXP で XML アプリケーションを記述する場合は、常に XML レジストリに自動的にアクセスします。WebLogic Server は、以下のようなルックアップ順序に従って、ロードするパーサのクラスを決定します。

1. 特定のドキュメント タイプ用に定義したパーサを使用します。
2. WebLogic Server インスタンスに関連付けられている XML レジストリで定義されたサーバ全体の代替パーサを使用します。
3. 組み込み Xerces パーサを使用します。

特定のドキュメント タイプに対応するトランスフォーマを定義することはできないので最初の手順は除きますが、それ以外ではこのプロセスはトランスフォーマにも当てはまります。

さらに、WebLogic Server の起動時に、SAX エンティティ リゾルバは、レジストリで宣言されたエンティティを解決するように自動的に設定されます。したがって、使用するパーサを制御したり、外部エンティティのローカル コピーの場所を設定したりするために、XML アプリケーション コードを変更する必要はありません。使用するパーサ、および外部エンティティの位置は、XML レジストリで制御します。

注意 : パーサによって JAXP の代わりに提供された API を使用する場合、XML レジストリは XML ドキュメントの処理に影響を与えません。このため、XML アプリケーションではなるべく JAXP を使用してください。

XML パーサおよびトランスフォーマのコンフィグレーション タスク

デフォルトでは、WebLogic Server は、組み込みパーサとトランスフォーマで XML ドキュメントを解析および変換するようにコンフィグレーションされています。リリース 6.1 では、組み込み XML パーサは Apache Xereces、組み込みトランスフォーマは Apache Xalan です。デフォルト コンフィグレーションを使用する場合、XML アプリケーション用のコンフィグレーション タスクは不要です。組み込み以外のパーサまたはトランスフォーマを使用する場合は、以下の節で説明するように XML レジストリでパーサおよびトランスフォーマをコンフィグレーションする必要があります。

組み込み以外のパーサまたはトランスフォーマのコンフィグレーション

次の手順ではまず、SAX および DOM パーサおよびトランスフォーマを定義する XML レジストリの作成方法を説明します。次に、サーバが新しいパーサとトランスフォーマを使用するために、新しい XML レジストリを WebLogic Server のインスタンスに関連付ける方法を説明します。

1. WebLogic 管理サーバを起動し、Administration Console をブラウザで起動します。Administration Console 開始の詳細については、4-1 ページの「WebLogic Server XML の管理の概要」を参照してください。
2. 左ペインの [サービス] ノードの下にある [XML] ノードを右クリックし、ドロップダウン メニューから [新しい XML Registry のコンフィグレーション] を選択します。以下のように、新しい XML レジストリの作成用ウィンドウが表示されます。

図 4-1 Administration Console のメイン XML レジストリ ウィンドウ

3. ユニークなレジストリ名を [名前] フィールドに入力し、[Document Builder ファクトリ] フィールド、[SAX パーサ ファクトリ] フィールド、および [Transformer ファクトリ] フィールドに適切なファクトリ パーサおよびトランスフォーマのクラスを設定します。

   たとえば、WebLogic FastParser の場合は、次の情報を入力します。

[名前] : WebLogic FastParser
[Document Builder ファクトリ]:
[SAX パーサ ファクトリ] : weblogic.xml.babel.jaxp.SAXParserFactoryImpl
[Transformer ファクトリ] :

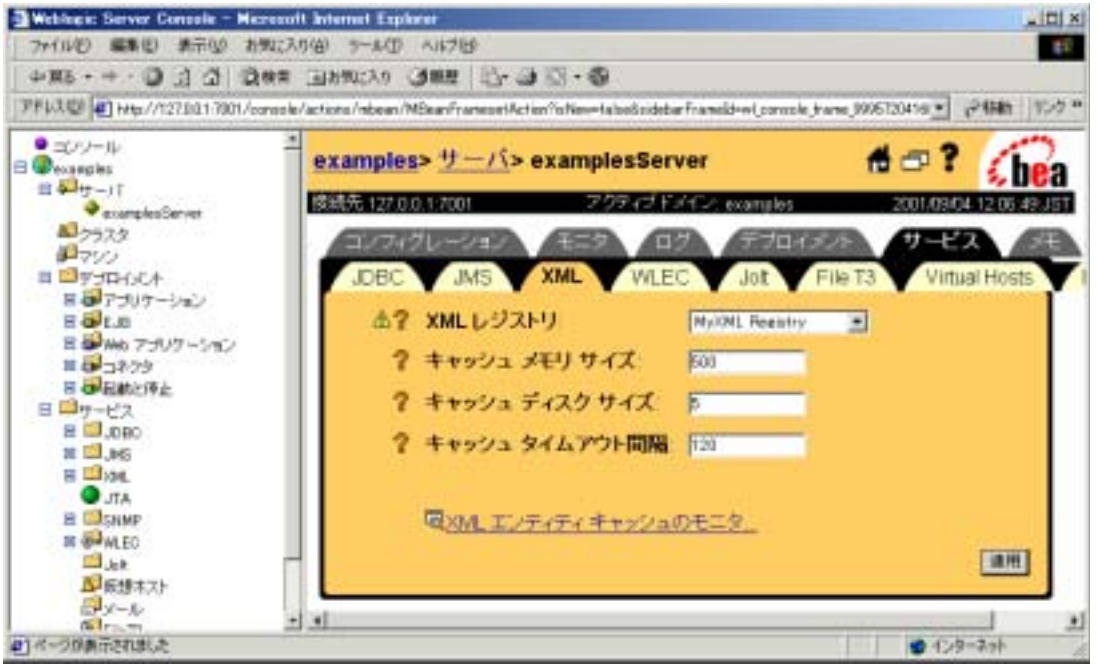
   上の例では、[Document Builder ファクトリ] および [Transformer ファクトリ] は空白のまま残されています。これは、DOM 解析および変換では、それぞれ組み込みパーサおよびトランスフォーマが使用されることを意味します。WebLogic FastParser は、SAX 解析でのみ使用されます。

Apache Xerces パーサおよび Xalan トランスフォーマを直接指定する場合、以下の情報のいずれかを指定します。

[名前] : Apache Xerces/Xalan Registry
[Document Builder ファクトリ] : org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
[SAX パーサ ファクトリ] : org.apache.xerces.jaxp.SAXParserFactoryImpl
[Transformer ファクトリ] : org.apache.xalan.processor.TransformerFactoryImpl

4. [作成] ボタンをクリックします。左ペインの [XML] ノードの下に、XML レジストリが作成されて表示されます。
5. 左ペインの [サーバ] ノード下で、新しい XML レジストリに関連付けるサーバの名前をクリックします。
6. 右ペインで、[サービス] タブを選択します。
7. [XML] タブを選択します。以下のように、WebLogic Server の XML プロパティのコンフィグレーション用ウィンドウが右ペインに表示されます。

図 4-2 Administration Console の XML プロパティ コンフィグレーション用ウィンドウ



8. [XML レジストリ] フィールドで、このサーバに関連付ける XML レジストリ名を選択し、[適用] ボタンをクリックします。
9. サーバを再起動して新しい設定内容を有効にします。

特定のドキュメント タイプに対応したパーサのコンフィグレーション

特定のドキュメント タイプに対応したパーサをコンフィグレーションする場合、ドキュメントのシステム ID、パブリック ID、ルート要素タグを使用して、ドキュメント タイプを識別できます。

注意： 次の手順では、これから新しい XML レジストリを作成して、必要なパーサ レジストリ エントリを追加し、サーバに関連付けることを前提としています。既存の XML レジストリを既にサーバに関連付けている場合は、手順 5. にスキップしてください。

特定のドキュメント タイプに対応したパーサをコンフィグレーションするには、以下の手順を実行します。

1. WebLogic 管理サーバを起動し、Administration Console をブラウザで起動します。

   Administration Console 開始の詳細については、4-1 ページの「WebLogic Server XML の管理の概要」を参照してください。

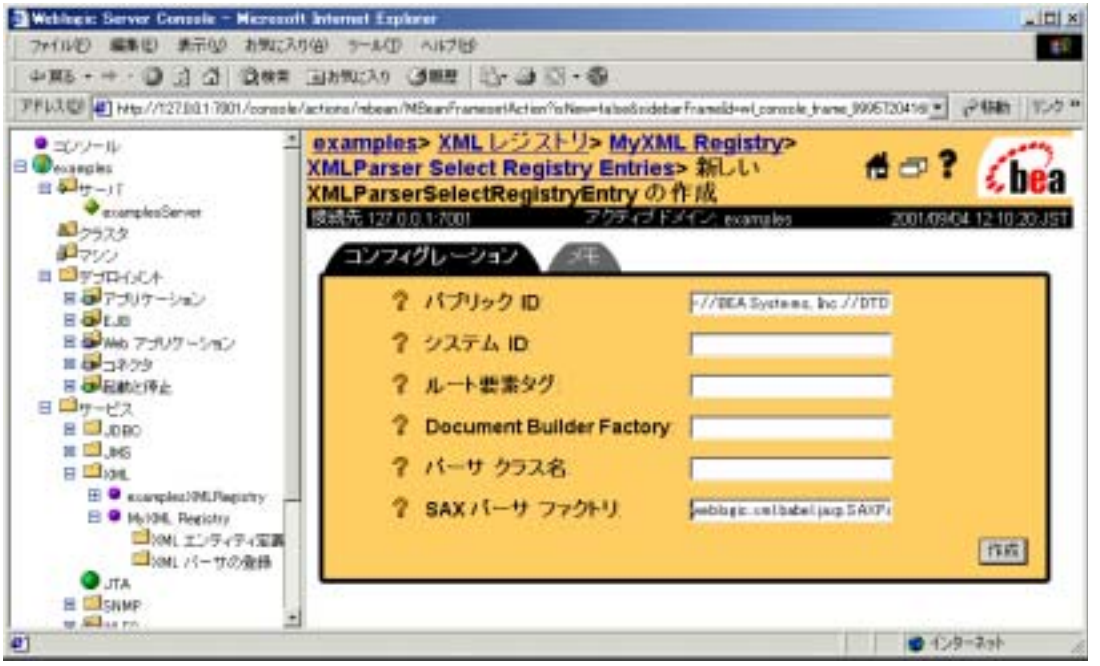
2. 左ペインの [サービス] ノードの下にある [XML] ノードを右クリックし、ドロップダウン メニューから [新しい XML Registry のコンフィグレーション] を選択します。図 4-1 のように、新しい XML レジストリの作成用ウィンドウが表示されます。

3. [名前] フィールドにユニークなレジストリ名を入力します。サーバに対してデフォルト パーサおよびトランスフォーマをコンフィグレーションする場合、[Document Builder ファクトリ]、[SAX パーサ ファクトリ]、および [Transformer ファクトリ] フィールドにファクトリ クラス名を入力します。そうでない場合は、フィールドを空白のままにします。

4. [作成] ボタンをクリックします。左ペインの [XML] ノードの下に、XML レジストリが作成されて表示されます。

5. 左ペインの [XML] ノード下で、XML レジストリの [XML パーサの登録] ノードを右クリックします。ドロップダウン メニューから [新しい XMLParserSelectRegistryEntry のコンフィグレーション] を選択します。以下のように、ドキュメント タイプ情報を入力するための空のウィンドウが右ペインに表示されます。

図 4-3 Administration Console による XML パーサのコンフィグレーション



6. 以下のいずれかの方法でドキュメント タイプ情報を入力します。

   a. [パブリック ID] フィールドまたは [システム ID] フィールドのいずれかに doctype を指定します。たとえば、`car.xml`（コード リスト 4-1 を参照）の場合、[パブリック ID] フィールドに「`-//BEA Systems, Inc.//DTD for cars//EN`」と入力します。

   b. [ルート要素タグ] にドキュメントのルート要素タグ名を指定します。`car.xml` の例では、[ルート要素タグ] フィールドに「`CAR`」と入力します。

      XML ドキュメントでネームスペースを定義する場合、`VEHICLES:CAR` のように完全修飾のルート要素タグを入力してください。

コード リスト 4-1 car.xml File

```
<?xml version="1.0"?>
<!-- 以下の XML ドキュメントでは自動車を説明 -->
<!DOCTYPE CAR PUBLIC "-//BEA Systems, Inc.//DTD for cars//EN"
"http://www.bea.com/dtds/car.dtd">
<CAR>
<MAKE>Toyota</MAKE>
<MODEL>Corrolla</MODEL>
<YEAR>1998</YEAR>
<ENGINE>1.5L</ENGINE>
<HP>149</HP>
</CAR>
```

7. [Document Builder ファクトリ] または [SAX パーサ ファクトリ] フィールドに、適切なファクトリ パーサ クラスを指定します。

   たとえば、このドキュメント タイプを WebLogic FastParser で解析するように指定する場合は、[SAX パーサ ファクトリ] フィールドに「`weblogic.xml.babel.jaxp.SAXParserFactoryImpl`」と入力します。

   注意 : [パーサ クラス名] フィールドに情報を入力しないでください。このフィールドは、旧バージョンの WebLogic Server との下位互換性を保つためのものです。

8. [作成] ボタンをクリックします。XMLParserSelect レジストリ エントリが作成されます。
9. 左ペインの [サーバ] ノード下で、新しい XML レジストリに関連付けるサーバの名前をクリックします。
10. 右ペインで、[サービス] タブを選択します。
11. [XML] タブを選択します。図 4-2 のように、WebLogic Server の XML プロパティのコンフィグレーション用ウィンドウが右ペインに表示されます。
12. [XML レジストリ] フィールドで、このサーバに関連付ける XML レジストリ名を選択し、[適用] ボタンをクリックします。
13. サーバを再起動して新しい設定内容を有効にします。

外部エンティティのコンフィグレーション タスク

XML レジストリを使用すると、外部エンティティ解決をコンフィグレーションし、外部エンティティ キャッシュをコンフィグレーションおよびモニタできます。

外部エンティティの解決のコンフィグレーション

WebLogic Server では、以下のいずれかの方法で外部エンティティの解決をコンフィグレーションできます。

- 物理的にエンティティ ファイルを、WebLogic 管理サーバからアクセス可能なディレクトリにコピーし、外部エンティティが XML ドキュメントで参照されている場合は常に管理サーバがそのローカル コピーを使用するように指定します。
- 管理サーバが起動したときか、または外部エンティティが最初に参照されたときに、管理サーバを基準にした相対 URL またはパス名で参照される外部エンティティを管理対象の WebLogic Server がキャッシュするように指定します。

  外部エンティティを管理対象の WebLogic Server にキャッシュすれば、アクセス時間を節約でき、ネットワークまたは管理サーバがダウンしたために XML ドキュメントの解析中に管理サーバにアクセスできない場合、ローカル バックアップを利用できます。

  キャッシュされたエンティティに対して、WebLogic Server が URL または管理サーバからエンティティを再取得して再キャッシュする有効期限をコンフィグレーションできます。

注意： 次の手順では、これから新しい XML レジストリを作成して、必要な外部エンティティ解決のエントリを追加し、サーバに関連付けることを前提としています。既存の XML レジストリを既にサーバに関連付けている場合は、手順 5. にスキップしてください。

外部エンティティの解決をコンフィグレーションするには、以下の手順を実行します。

1. WebLogic 管理サーバを起動し、Administration Console をブラウザで起動します。

   Administration Console 開始の詳細については、4-1 ページの「WebLogic Server XML の管理の概要」を参照してください。

2. 左ペインの [サービス] ノードの下にある [XML] ノードを右クリックし、ドロップダウン メニューから [新しい XML Registry のコンフィグレーション] を選択します。図 4-1 のように、新しい XML レジストリの作成用ウィンドウが表示されます。

3. [名前] フィールドに、ユニークなレジストリ名を入力します。サーバに対してデフォルト パーサおよびトランスフォーマをコンフィグレーションする場合、[Document Builder ファクトリ]、[SAX パーサ ファクトリ]、および [Transformer ファクトリ] フィールドにファクトリ クラス名を入力します。そうでない場合は、フィールドを空白のままにします。

4. [作成] ボタンをクリックします。左ペインの [XML] ノードの下に、XML レジストリが作成されて表示されます。

5. 左ペインの [XML] ノード下で、XML レジストリの [XML エンティティ定義] ノードを右クリックします。ドロップダウン メニューから [新しい XMLEntitySpecRegistryEntry のコンフィグレーション] を選択します。以下のように、エンティティ解決の情報を入力するための空のウィンドウが右ペインに表示されます。

図 4-4 Administration Console による外部エンティティのコンフィグレーション

6. XML ドキュメントで外部エンティティを参照するのに使用する [システム ID] または [パブリック ID] のどちらかを入力します。たとえば、次の `car.xml` ファイルの場合は、[システム ID] に対して「`http://www.bea.com/dtds/car.dtd`」と入力します。

コード リスト 4-2 car.xml ファイル

```
<?xml version="1.0"?>
<!-- 以下の XML ドキュメントでは自動車を説明 -->
<!DOCTYPE CAR PUBLIC "-//BEA Systems, Inc.//DTD for cars//EN"
"http://www.bea.com/dtds/car.dtd">
<CAR>
<MAKE>Toyota</MAKE>
<MODEL>Corrolla</MODEL>
<YEAR>1998</YEAR>
<ENGINE>1.5L</ENGINE>
```

```
<HP>149</HP>
</CAR>
```

7. [エンティティ URI] フィールドで、以下のいずれかのエンティティ パスを入力します。
   a. 管理サーバ内のエンティティ ファイルのコピーのパス名。このパス名は、ドメイン コンフィグレーション ディレクトリの *BEAHome*\wlserver6.1\config*domain*\xml\registries*reg_name* というように、レジストリ エンティティ ディレクトリを基準にした相対パスにする必要があります。このパス名では、*BEAHome* は、*WebLogic Server* ソフトウェアがインストールされている最上位ディレクトリ、*domain* は、WebLogic Server ドメイン名、*reg_name* は、新しい XML レジストリ名です。たとえば、car.xml ファイルの場合、[エンティティ URI] フィールドに「dtds\car.dtd」と入力できます。
   b. Web 上の外部エンティティを示す URL、またはレポジトリに保存されているエンティティ。たとえば、「http://java.sun.com/j2ee/dtds/application_1_2.dtd」と入力して、J2EE エンタープライズ アプリケーションの説明に使用する application.xml ファイルの DTD を参照するか、jdbc: を使用してデータベース内の任意のエンティティを参照します。

      http://、file://、jdbc:、または ftp:// のプロトコル宣言を使用して、外部エンティティを指定します。
8. [キャッシュのタイミング] リスト ボックスから以下のオプションのいずれか 1 つを選択します。
   - [参照時] ― WebLogic Server は、最初にエンティティが XML ドキュメントで参照されたときに URL で参照される外部エンティティをキャッシュします。
   - [初期化時] ― WebLogic Server は、サーバ起動時にエンティティをキャッシュします。
   - [レジストリの設定通り] ― WebLogic Server は、デフォルトのキャッシュ設定を使用します。デフォルト キャッシュ設定のコンフィグレーションの詳細については、4-17 ページの「外部エンティティ キャッシュのコンフィグレーション」を参照してください。

9. [キャッシュ タイムアウト間隔] フィールドで、キャッシュされた外部エンティティが古くなる（期限切れになる）までの秒数を入力します。キャッシュされたコピーがこの期間を超えた場合に、WebLogic Server は、管理サーバを基準に指定した相対 URL またはパス名から外部エンティティを再取得します。

   このフィールドのデフォルト値は -1 で、WebLogic Server のグローバル タイムアウト値が使用されます。グローバル キャッシュ タイムアウト設定のコンフィグレーションの詳細については、4-17 ページの「外部エンティティ キャッシュのコンフィグレーション」を参照してください。

10. [作成] ボタンをクリックします。XMLEntitySpec レジストリ エントリが作成されます。

11. 左ペインの [サーバ] ノード下で、新しい XML レジストリに関連付けるサーバの名前をクリックします。

12. 右ペインで、[サービス] タブを選択します。

13. [XML] タブを選択します。図 4-2 のように、WebLogic Server の XML プロパティのコンフィグレーション用ウィンドウが右ペインに表示されます。

14. [XML レジストリ] フィールドで、このサーバに関連付ける XML レジストリ名を選択し、[適用] ボタンをクリックします。

15. サーバを再起動して新しい設定内容を有効にします。

16. Web からのキャッシュではなく、エンティティのローカル コピーを使用するように指定した場合は、エンティティ ファイルをエンティティ ディレクトリをコピーします。たとえば、`car.dtd` ファイルを *`BEAHome`*`\wlserver6.1\config\`*`domain`*`\xml\registries\`*`reg_name`*`\dtds` ディレクトリにコピーします。このパス名では、*`BEAHome`* は、*`WebLogic Server`* ソフトウェアがインストールされている最上位ディレクトリ、*`domain`* は、WebLogic Server ドメイン名、*`reg_name`* は、新しい XML レジストリ名です。

外部エンティティ キャッシュのコンフィグレーション

外部エンティティ キャッシュの以下のプロパティをコンフィグレーションできます。

- キャッシュ メモリのサイズ（KB）。このプロパティのデフォルト値は、500KB です。
- 永続ディスク キャッシュのサイズ（MB）。このプロパティのデフォルト値は、5MB です。
- WebLogic Server にキャッシュされた外部エンティティが期限切れになるまでの秒数。これは、サーバ全体のデフォルト値です。エンティティをコンフィグレーションする場合に、特定の外部エンティティに対応するようにこの値をオーバーライドできます。このプロパティのデフォルト値は、120 秒（2 分）です。

外部エンティティ キャッシュをコンフィグレーションするには、次の手順に従います。

1. WebLogic 管理サーバを起動し、Administration Console をブラウザで起動します。

   Administration Console 開始の詳細については、4-1 ページの「WebLogic Server XML の管理の概要」を参照してください。

2. 左ペインの [サーバ] ノード下で、外部エンティティ キャッシュをコンフィグレーションする WebLogic Server の名前をクリックします。
3. 右ペインの [サービス] タブを選択します。
4. [XML] タブを選択します。図 4-2 のように、WebLogic Server の XML プロパティのコンフィグレーション用ウィンドウが右ペインに表示されます。
5. [キャッシュ メモリ サイズ] フィールドで、キャッシュ メモリのサイズ（KB）を入力します。
6. [キャッシュ ディスク サイズ] フィールドで、永続ディスク キャッシュのサイズ（MB）を入力します。

7. [キャッシュ タイムアウト間隔] フィールドで、エンティティが期限切れになるまでの秒数を入力します。
8. [適用] ボタンをクリックします。

外部エンティティ キャッシュのモニタ

外部エンティティ キャッシュを説明する一連の統計情報を使用して、キャッシュの効果をモニタできます。統計では、以下の情報を示します。

- キャッシュの現在の状態。
- 現在のセッションの累積アクティビティ。
- キャッシュ作成後、通常は WebLogic Server 起動後の累積アクティビティ。

統計情報にアクセスするには、J2EE Java Management Extension（JMX）仕様と WebLogic Server Management API を一緒に使用して、Management Bean（MBean）を作成およびデプロイし、WebLogic Server の外部エンティティ キャッシュをモニタします。以下の MBean インタフェースを使用します。

- `weblogic.management.runtime.EntityCacheCumulativeRuntimeMBean`
- `weblogic.management.runtime.EntityCacheCurrentStateRuntimeMBean`
- `weblogic.management.runtime.EntityCacheRuntimeMBean`

WebLogic Server Management API は、Javadoc でオンライン ドキュメント化されています。

以下の表で、外部エンティティ キャッシュの現在の状態に関する統計情報の取得に使用できるメソッドについて説明します。

表 4-1 キャッシュ統計の現在の状態

メソッド	説明
getMemoryUsage	すべてのメモリ常駐エントリの保存に使用するバイト数を返す。
getDiskUsage	すべてのディスク常駐エントリの保存に使用するバイト数を返す。
getTotalCurrentEntries	キャッシュ内のエントリ合計数を返す。
getTotalPersistentCurrentEntries	キャッシュ内の永続エントリ数を返す。
getTotalTransientCurrentEntries	キャッシュ内の一時エントリ数を返す。
getAvgPercentTransient	エントリの中で一時エントリが占めるパーセントを返す。
getAvgPercentPersistent	エントリの中で永続エントリが占めるパーセントを返す。
getAvgTimeout	エントリの平均タイムアウト値を返す。
getMinEntryTimeout	現在のエントリの最小タイムアウト値を返す。
getMaxEntryTimeout	現在のエントリの最大タイムアウト値を返す。
getAvgPerEntryMemorySize	現在のエントリの平均メモリ サイズを返す。
getMaxEntryMemorySize	現在のエントリの最大メモリ サイズを返す。
getMinEntryMemorySize	現在のエントリの最小メモリ サイズを返す。
getAvgPerEntryDiskSize	現在のエントリの平均ディスク サイズを返す。

以下の表で、外部エンティティ キャッシュの累積アクティビティに関する統計情報の取得に使用できるメソッドについて説明します。

表 4-2 キャッシュの累積アクティビティ

メソッド	説明
getTotalCurrentEntries	キャッシュ内のエントリ合計数を返す。
getTotalPersistentCurrentEntries	キャッシュ内の永続エントリ数を返す。
getTotalTransientCurrentEntries	キャッシュ内の一時エントリ数を返す。
getAvgPercentTransient	エントリの中で一時エントリが占めるパーセントを返す。
getAvgPercentPersistent	エントリの中で永続エントリが占めるパーセントを返す。
getAvgTimeout	エントリの平均タイムアウト値を返す。
getMinEntryTimeout	現在のエントリの最小タイムアウト値を返す。
getMaxEntryTimeout	現在のエントリの最大タイムアウト値を返す。
getAvgPerEntryMemorySize	現在のエントリの平均メモリ サイズを返す。
getMaxEntryMemorySize	現在のエントリの最大メモリ サイズを返す。
getMinEntryMemorySize	現在のエントリの最小メモリ サイズを返す。
getAvgPerEntryDiskSize	現在のエントリの平均ディスク サイズを返す。
getTotalNumberMemoryPurges	完了したメモリ パージ数を返す。
getTotalItemsMemoryPurged	すべてのメモリ パージでパージされた項目の合計数を返す。
getAvgEntrySizeMemoryPurged	メモリ パージされた項目の平均サイズをバイト数で返す。
getMostRecentMemoryPurge	最後のメモリ パージの時刻を返す。
getMemoryPurgesPerHour	1 時間ごとの平均メモリ パージ数を返す。
getTotalNumberDiskPurges	完了したディスク パージ数を返す。

表 4-2 キャッシュの累積アクティビティ

メソッド	説明
getTotalItemsDiskPurged	すべてのディスク パージでパージされた項目の合計数を返す。
getAvgEntrySizeDiskPurged	ディスク パージされた項目の平均サイズをバイト数で返す。
getMostRecentDiskPurge	最後のディスク パージの時刻を返す。
getDiskPurgesPerHour	1 時間ごとの平均ディスク パージ数を返す。
getTotalNumberOfRejections	拒否されたエントリ数を返す。
getTotalSizeOfRejections	拒否された全項目の合計サイズをバイト数で返す。
getPercentRejected	拒否された挿入をパーセントで返す。
getTotalNumberOfRenewals	期限切れエントリが更新された回数を返す。

5　XML リファレンス

以下の節では、WebLogic Server でサポートされる XML 仕様、アプリケーション プログラミング インタフェース（API）、およびツールについて説明します。

- Extensible Markup Language （XML）1.0 仕様
- Simple API for XML （SAX）2.0
- Document Object Model （DOM）Level 2 API
- W3C XML ネームスペース 1.0 勧告
- Java API for XML Processing (JAXP) 1.1.1
- Apache Xerces Java パーサ API
- Apache Xalan XML Stylesheet Language Transformer （XSLT）API
- その他の情報源

Extensible Markup Language （XML）1.0 仕様

W3C の XML 勧告では以下の抜粋のとおりに規定されています。

「Extensible Markup Language（XML）は SGML のサブセットであり、このドキュメントで完全に規定されています。XML の目的は、現在 HTML で実現されているように、汎用的な SGML を Web 上で配信、受信、および処理できるようにすることです。XML は実装が容易で、SGML および HTML との相互運用性を持つように設計されています。」

完全な XML 仕様は、http://www.w3.org/TR/REC-xml.html で公開されています。

Simple API for XML （SAX）2.0

SAX API は、プラットフォームと言語に依存しない API です。SAX API は、XML-DEV メーリング リストのメンバーによって共同開発された、イベントベースの XML 解析用の標準インタフェースです。

SAX アプリケーションは、パーサ オブジェクトを作成し、ハンドラを XML イベントに関連付けることによって XML ドキュメントを処理します。これらのタスクが完了すると、パーサはイベントの発生時にドキュメントを読み取り、それらをハンドラに渡します。イベントはドキュメント内部のエンティティを表します。たとえば、ドキュメントの開始、ドキュメントの終了、要素の開始、要素の終了などです。SAX インタフェースは単純なコーディング モデルを提供し、比較的単純な階層構造を持つ XML ドキュメントを処理するのに役立ちます。このため、SAX インタフェースは組み込みパーサで XML ドキュメントを解析するために必要なものを備えています。

SAX の詳細については、http://www.megginson.com/SAX/index.html を参照してください。Javadoc のドキュメントについては、SAX（Simple API for XML）を参照してください。

Document Object Model （DOM）Level 2 API

DOM API は、プラットフォームと言語に依存しないインタフェースです。DOM API を使用すると、プログラムとスクリプトは XML ドキュメントのコンテンツ、構造、およびスタイルに動的にアクセスしてそれらを更新できます。DOM を使用すると、XML ドキュメントに階層オブジェクト モデルとして格納される情報にアクセスできます。このモデルは、ドキュメントのルート要素をルート ノードとするツリーによく似ています。DOM インタフェースを使用すると、XML ドキュメントのさまざまな部分にアクセスし、それらを参照して変更や追加を行うことができます。

アプリケーションが DOM パーサを呼び出すと、パーサはドキュメント全体を処理し、インメモリ オブジェクト モデルを作成します。アプリケーションは、このモデルを任意の方法で処理できます。DOM 方式は、より複雑なドキュメントを扱うのに最も役立ちます。開発者が各要素を解釈する必要がないからです。

DOM の詳細については、DOM（Document Object Model）Level 2 仕様を参照してください。Javadoc のドキュメントについては、DOM（Document Object Model）を参照してください。

W3C XML ネームスペース 1.0 勧告

以下に W3C XML ネームスペース勧告からの抜粋を示します。

「XML ネームスペースは、Extensible Markup Language（XML）ドキュメントで使用される要素および属性の名前を、Universal Resource Identifier （URI）参照で識別するネームスペースに関連付け、これによりそれらの要素名および属性名を修飾する単純な方法を提供します。」

XML ネームスペース 1.0 勧告は、インターネット上（http://www.w3.org/TR/REC-xml-names/）で公開されています。

Java API for XML Processing (JAXP) 1.1.1

JAXP は、Sun の XML 解析および変換用 Java API です。JAXP は、標準化された Java プラットフォーム API セットを介して XML ドキュメントの解析、操作、および変換の基本的なサポートを提供します。このため、JAXP を使用して XML ドキュメントを処理するアプリケーションはプラットフォーム間で移植できます。

JAXP は、SAX API または DOM API に取って代わるものではありません。その代わり、SAX API と DOM API を使用するアプリケーションを移植可能にするための便利なメソッドを追加します。

JAXP 1.1 は、XML データ解析および変換用のインタフェース、クラス、メソッドを含む `javax.xml.parsers` と `javax.xml.transform` パッケージで構成されています。

JAXP 仕様は、インターネット上（`http://java.sun.com/xml/`）で公開されています。JAXP Javadoc は、http://java.sun.com/xml/jaxp/dist/1.1/docs/api/index.html で公開されています。

Apache Xerces Java パーサ API

Apache Xerces Java パーサ パッケージには、最も一般的な XML プログラミング インタフェースである SAX と DOM 用の API ドキュメントが収められています。また、このパーサには、SAX API と DOM API の一部ではないが、パーサ プログラムの作成に役立つクラスのドキュメントも付属しています。

Apache Xerces Java パーサの詳細については、http://xml.apache.org/xerces-j/index.html を参照してください。Javadoc のドキュメントについては、Apache Xerces Java Parser を参照してください。

Apache Xalan XML Stylesheet Language Transformer （XSLT）API

Apache Xalan-Java XSLT トランスフォーマは、XML ドキュメントの変換に使用されます。このプロセッサは、1999 年 11 月 16 日付 W3C 勧告「XSL Transformations (XSLT) Version 1.0」を実装しています。XSLT は、XML ドキュメントを別の XML ドキュメント、HTML ドキュメント、または他のドキュメント タイプに変換するためのスタイルシート言語です。この言語では、XSL 変換ボキャブラリと XPath （XML ドキュメントの一部を処理するための言語）が使用されます。XSL スタイルシートには、XML 入力のノード ツリーを別のノード ツリーに変換する方法が定義されます。

Apache Xalan XSLT トランスフォーマの詳細については、http://xml.apache.org/xalan-j/index.html を参照してください。Javadoc のドキュメントについては、Apache Xalan XSLT Transformer を参照してください。

その他の情報源

この節では、WebLogic XML を使ったプログラミングの学習に役立つさまざまなオンライン情報源を示します。

- コード例
- 関連する WebLogic マニュアル
- 一般的な XML 情報
- チュートリアルとオンライン コース
- その他の XML 仕様

コード例

XML サンプル コードとサポート ドキュメントは、*BEA Home*\wlserver6.1\samples\examples\xml にある WebLogic Server 配布キットに収められています。*BEA Home* は、WebLogic Server ソフトウェアがインストールされているディレクトリです。

関連する WebLogic マニュアル

- 『WebLogic Web サービス プログラマーズ ガイド』
- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』
- 『WebLogic JMS プログラマーズ ガイド』
- 『WebLogic JSP プログラマーズ ガイド』

- 『WebLogic HTTP サーブレット プログラマーズ ガイド』
- 『BEA WebLogic Server Wireless Application 開発プログラマーズ ガイド』

一般的な XML 情報

- W3C （World Wide Web Consortium）
- XML.com
- XML FAQ
- XML.org, The XML Industry Portal
- W3C:Extensible Stylesheet Language

チュートリアルとオンライン コース

- A Technical Introduction to XML
- XML Authoring Tutorial
- Working with XML and Java
- Tutorials for using the Java 2 platform and XML technology
- Developing XML Solutions with JavaServer Pages Technology
- XML, Java, and the Future of the Web
- 『XML Bible』の第 14 章「XSL Transformations」
- 『XSL Tutorial』、Miloslav Nic 著
- XML Schema Part 0: Primer

その他の XML 仕様

- Extensible Stylesheet Language (XSL) 1.0 仕様

- JSR-000031 XML Data Binding 仕様
- XML Path Language （XPath）Version 1.0 仕様
- XML Linking Language （XLink）仕様
- XML Pointer Language （XPointer）仕様
- XML Schema Part 1:Structures
- XML Schema Part 2:Datatypes

索引

い

か

く

ゆ