Oracle® Application Server Containers for J2EE

JSP タグ・ライブラリおよびユーティリティ・リファレンス

10*g* リリース 2（10.1.2）

部品番号 : B15621-02

2005 年 11 月

ORACLE®

Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス , 10*g* リリース 2（10.1.2）

部品番号 : B15621-02

原本名 : Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference, 10*g* Release 2 (10.1.2)

原本部品番号 : B14016-02

原本著者 : Brian Wright

原本協力者 : Michael Freedman, Jesse Anton, Ellen Barnes, Julie Basu, Charlie Berger, Fred Bethke, Carolyn Bruse, Kelly Chan, Matthieu Devin, Sumathi Gopalakrishnan, Ralph Gordon, Ping Guo, Hal Hildebrand, Christine Jacobs, Nilesh Junnarkar, Susan Kraft, Sunil Kunisetty, Song Lin, Angie Long, Peter Lubbers, Sharon Malek, Sheryl Maring, Kuassi Mensah, Jasen Minton, Charles Murray, Dmitry Nonkin, John O'Duinn, Sue Pelski, Olga Peschansky, Shiva Prasad, Jerry Schwarz, Sanjay Singh, Ingrid Snedecor, Deborah Steiner, Gael Stevens, Margaret Taft, George Tang, Jingwu Tang, Olaf van der Geest, YaQing Wang, Alex Yiu, David Zhang

目次

7 Web Object Cache のタグと API

8 ファイル・アクセスとメール用の Bean とタグ

9 JSP のユーティリティとユーティリティ・タグ

10 パーソナライズ・タグ

11 Web サービス・タグ

A JML コンパイル時構文とタグ

B サード・パーティ・ライセンス

索引

はじめに

JavaServer Pages（JSP）テクノロジは、Sun 社が先導する業界グループが定めるとおり、標準的な Java 2 Enterprise Edition（J2EE）のコンポーネントです。Oracle Application Server の J2EE コンポーネントは、Oracle Application Server Containers for J2EE（OC4J）と呼ばれます。

このマニュアルでは、Oracle Application Server 10*g* リリース 2（10.1.2）の OC4J に同梱されている JSP タグ・ライブラリとユーティリティに関する参照情報および概念について説明します。これらのライブラリは通常、JSP 仕様に準拠しています。

JSP タグ・ライブラリのフレームワークなど、OC4J JSP の実装に関する一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

この章には、次の項目が含まれます。

対象読者

このマニュアルは、サーブレットおよびJavaServer Pagesテクノロジを使用するWebアプリケーション開発者を対象としています。Web、サーブレットおよびJSP環境での作業経験があり、次の内容を理解していることが前提です。

- 一般的なWebテクノロジ
- Java
- HTML
- Java Servlet
- JavaServer Pages
- Webサーバーおよびサーブレット環境の構成
- Oracle JDBC（Oracle Databaseにアクセスする JSPアプリケーション用）

標準のJavaServer Pagesテクノロジ、Oracle JSPの実装およびタグ・ライブラリのサポートに関するバックグラウンド情報の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages開発者ガイド』を参照してください。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWebサイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかしJAWSは括弧だけの行を読まない場合があります。

外部Webサイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しないWebサイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらのWebサイトのアクセシビリティに関しての評価や言及は行っておりません。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

Oracle Java Platform グループからは、次の OC4J マニュアルを入手できます。

- 『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

  このマニュアルは、OC4J の概要および一般情報を提供します。サーブレット、JSP ページおよび EJB に関する初歩的な章が含まれ、一般的な構成とデプロイについて説明します。

- 『Oracle Application Server Containers for J2EE スタンドアロン・ユーザーズ・ガイド』

  このバージョンのユーザーズ・ガイドは、特にスタンドアロン・バージョンの OC4J を対象としており、OTN からスタンドアロン・バージョンをダウンロードするときに入手できます。OC4J スタンドアロンは開発環境で使用されますが、通常、本番環境では使用しません。

- 『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』

  このマニュアルは、サーブレット開発者向けに、OC4J でのサーブレットとサーブレット・コンテナの使用情報を提供します。これには、基本的なサーブレット開発、JDBC と EJB の使用、アプリケーションのビルドとデプロイ、サーブレットと Web サイトの構成などが含まれます。

- 『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』

  このマニュアルは、OC4J で独自のページを実行する JSP 開発者向けの情報を提供します。JSP 標準の一般的な概要とプログラミングに関する考慮事項も含まれます。また、OC4J 環境を初めて使用する方のために、Oracle の付加価値機能および手順についても説明します。

- 『Oracle Application Server Containers for J2EE サービス・ガイド』

  このマニュアルは、JTA、JNDI、JMS、JAAS および Oracle Application Server の Java Object Cache など、OC4J が提供する、標準に基づく Java サービスに関する情報を提供します。

- 『Oracle Application Server Containers for J2EE セキュリティ・ガイド』

  このマニュアルは、OC4J に特有のセキュリティ機能と実装に関する情報を提供します（『Oracle Application Server セキュリティ・ガイド』と混同しないでください）。これには、JAAS（Java Authentication and Authorization Service）や他の Java セキュリティ・テクノロジの使用方法が含まれます。

- 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』

  このマニュアルは、OC4J での EJB 実装および EJB コンテナに関する情報を提供します。

Oracle Java Platform グループからは、次のマニュアルも入手できます。

- 『Oracle Database Java 開発者ガイド』
- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- 『Oracle Database JPublisher ユーザーズ・ガイド』

Oracle Application Server グループからは次のマニュアルを入手できます。

- 『Oracle Application Server 管理者ガイド』
- 『Oracle Application Server セキュリティ・ガイド』
- 『Oracle Application Server パフォーマンス・ガイド』
- 『Oracle Enterprise Manager 概要』
- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Application Server グローバリゼーション・サポート・ガイド』
- 『Oracle Application Server Web Cache 管理者ガイド』
- 『Oracle Application Server Web Services 開発者ガイド』
- Oracle Application Server のアップグレードおよび互換性ガイド

Oracle JDeveloper グループからは次のドキュメントを入手できます。

- Oracle JDeveloper オンライン・ヘルプ
- OTN（Oracle Technology Network）上の Oracle JDeveloper ドキュメント

  `http://www.oracle.com/technology/products/jdev/content.html`

Oracle Server Technologies グループからは次のマニュアルを入手できます。

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML API リファレンス』
- 『Oracle Database アプリケーション開発者ガイド - 基礎編』
- 『PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Database SQL リファレンス』
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Advanced Security 管理者ガイド』
- 『Oracle Database リファレンス』

パーソナライズ・タグ・ライブラリの基礎である OracleAS Personalization の詳細は、OracleAS Personalization グループからの次のドキュメントを参照してください。

- 『Oracle Application Server Personalization 管理者ガイド』
- 『Oracle Application Server Personalization プログラマーズ・ガイド』

Java Servlet および JavaServer Pages に関する次の OTN Web サイトも利用できます。

`http://www.oracle.com/technology/tech/java/servlets/`

次のリソースは、Sun 社の Web サイトから入手できます。

- JavaServer Pages（最新仕様を含む）に関する Web サイト

  `http://java.sun.com/products/jsp/index.jsp`

- Java Servlet（最新仕様を含む）に関する Web サイト

  `http://java.sun.com/products/servlet/index.jsp`

- `jsp-interest` ディスカッション・グループ

  サブスクライブするには、メッセージ本文に次の行を入力して、`listserv@java.sun.com` に電子メールを送信してください。

  `subscribe jsp-interest` *`yourlastname yourfirstname`*

  ただし、送信された電子メールの日刊ダイジェストのみサブスクライブすることをお薦めします。その場合は、メッセージ本文に次の行も追加してください。

  `set jsp-interest digest`

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

オラクル社カスタマ・サポート・センター

オラクル製品サポートの購入方法、およびオラクル社カスタマ・サポート・センターへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

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

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

タグ・ライブラリとユーティリティの概要

このマニュアルでは、Oracle Application Server Containers for J2EE（OC4J）が提供するタグ・ライブラリ、JavaBeans およびその他のユーティリティについて説明します。これらは、JSP 標準に基づいて実装されています。また、JavaServer Pages 標準タグ・ライブラリ（JSTL）のサポートに関する説明、および OC4J 以外の Oracle コンポーネントが提供するタグ・ライブラリのサマリーも記載しています。

Oracle 固有の機能と、OC4J JSP コンテナ、標準 JSP テクノロジおよび標準 JSP 1.2 タグ・ライブラリの機能の概要は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』で説明しています。

この章には、次の項目が含まれます。

- OC4J が提供するタグ・ライブラリとユーティリティの概要
- Web アプリケーションに対する Oracle キャッシング・サポートのサマリー
- JavaServer Pages 標準タグ・ライブラリのサポート
- 他の Oracle コンポーネントのタグ・ライブラリの概要

最初の項で紹介するタグと JavaBeans には、型の拡張機能、XML/XSL との統合、データベース・アクセスおよび便利なプログラミング機能など、多様な機能が備わっています。

注意： 以前のリリースに含まれていたサンプル・アプリケーションに関する章はなくなりました。この章に記載されていたアプリケーションは、Oracle Technology Network（OTN）の次の URL にある OC4J デモで利用できます（OTN への登録が必要ですが無償です）。

```
http://www.oracle.com/technology/tech/java/oc4j/demos/
```

OC4J が提供するタグ・ライブラリとユーティリティの概要

次の各項で紹介する Oracle の拡張機能は、タグ・ライブラリまたはカスタム JavaBeans を使用して実装されます。この拡張機能は、JSP 標準と JavaBeans 標準に準拠しています。

- タグ構文の表記と意味
- 拡張型 JavaBeans の概要
- イベント処理用の JspScopeListener の概要
- XML および XSL との統合の概要
- データ・アクセス JavaBeans とタグ・ライブラリのサマリー
- JSP Markup Language（JML）のカスタム・タグ・ライブラリのサマリー
- Oracle Application Server Personalization タグ・ライブラリのサマリー
- Web サービス・タグ・ライブラリのサマリー
- ファイル・アクセス・タグとメール・タグのサマリー
- EJB タグのサマリー
- JSP ユーティリティ・タグのサマリー

OC4J が提供するカスタム・タグ・ライブラリのいくつか（XML、データ・アクセスおよび JML）は、JavaServer Pages 標準タグ・ライブラリ（JSTL）より前に開発されたもので、JSTL と重複する機能が含まれているので注意してください。標準に準拠するために、原則として、カスタム・ライブラリではなく、JSTL の使用をお薦めします。1-18 ページの「JavaServer Pages 標準タグ・ライブラリのサポート」を参照してください。

ただし、既存のライブラリもサポート対象です。JSTL で使用できないカスタム・ライブラリの機能のうち、有効と判断される機能は、必要に応じて、JSTL 標準に採用される予定です。

注意：

- キャッシング機能のサポート用に OC4J が提供するタグ・ライブラリの詳細は、1-13 ページの「Web アプリケーションに対する Oracle キャッシング・サポートのサマリー」を参照してください。
- この項で紹介する機能を使用したサンプル・アプリケーションは、OC4J のデモを参照してください。サンプル・アプリケーションは、次の URL で Oracle Technology Network（OTN）からダウンロードできます（OTN への登録が必要ですが無償です）。

  `http://www.oracle.com/technology/tech/java/oc4j/demos/`

タグ構文の表記と意味

このマニュアルでは、タグの説明の構文表記に、次の規則を使用しています。

- イタリックは、指定する必要がある値または文字列を示します。
- オプションの属性は、大カッコ（`[...]`）で囲まれています。
- オプションの属性のデフォルト値は、**太字**で示します。
- 属性の値に複数の選択肢がある場合は、縦線（|）で区切られています。
- 特に注記がないかぎり、`<%=` *jspExpression* `%>` という JSP の実行時の式を使用して、タグの属性値を設定できます。
- このマニュアルのタグ説明では、慣例に従って特定のタグ接頭辞を使用しています。ただし、`taglib` ディレクティブには、任意の接頭辞を指定できます。

拡張型 JavaBeans の概要

JSP ページでは、スカラー値の表現をコアとなる Java の型に依存しています。ただし、次に示す標準の型カテゴリは、いずれも JSP ページでの使用に適していません。

- プリミティブ型（int、float および double など）

  これらの型の値には、スコープを指定できません。つまり、JSP のスコープ・オブジェクト（page、request、session または application スコープ）に格納できません。これは、スコープ・オブジェクトに格納できるのはオブジェクトのみであるためです。

- 標準の java.lang パッケージにあるラッパー・クラス（Integer、Float および Double など）

  これらの型の値はオブジェクトであるため、理論上では JSP スコープ・オブジェクトに格納できます。ただし、ラッパー・クラスは JavaBean モデルに準拠しておらず、引数が 0（ゼロ）のコンストラクタを提供していないため、jsp:useBean 操作でこの型を宣言できません。

  さらに、ラッパー・クラスのインスタンスは不変です。値を変更するには、インスタンスを新たに作成して、適切に割り当てる必要があります。

JDK 1.4 以下の環境のこれらの制限事項を回避するために、OC4J は、oracle.jsp.jml パッケージに JmlBoolean、JmlNumber、JmlFPNumber および JmlString JavaBean クラスを用意し、最も一般的な Java の型をラップします。

詳細は、第 2 章「拡張型用の JavaBeans」を参照してください。

イベント処理用の JspScopeListener の概要

重要： JspScopeListener は OC4J 10.1.2 実装で廃止され、それ以降の実装でサポート外になる予定です。

OC4J は、JSP アプリケーション内にある様々なスコープの Java オブジェクトのライフ・サイクルを管理するため、JspScopeListener インタフェースを提供します。

標準のサーブレットと JSP のイベント処理は、javax.servlet.http.HttpSessionBindingListener インタフェースを介して実行されますが、これは、セッション・ベースのイベントのみを処理します。Oracle の JspScopeListener を HttpSessionBindingListener と統合すると、セッション・ベースのイベントの他に、ページ・ベース、リクエスト・ベースおよびアプリケーション・ベースのイベントも処理できます。

詳細は、9-2 ページの「JspScopeListener による JSP のイベント処理」を参照してください。

XML および XSL との統合の概要

JSP 構文を使用すると、HTML コードの他に、テキスト・ベースの MIME タイプを生成できます。特に、XML 出力を動的に作成できます。ただし、JSP ページを使用して XML 文書を生成する場合は、XML データをクライアントに送信する前に、XML データに適用するスタイルシートが必要になる場合があります。この場合、JavaServer Pages テクノロジでは、JSP ページに使用する標準の出力ストリームがサーバーを通じて直接書き戻されるため、この処理は困難です。

OC4J には、JSP ページのすべてまたは一部を、出力前に XSL のスタイルシートを使用して変換するように指定する特別のタグが用意されています。入力は、タグ・ボディまたは XML DOM オブジェクトから可能で、XML DOM オブジェクトからブラウザへの出力が可能です。

ページ内の異なる部分にスタイルシートを個別に指定する場合は、これらのタグを単一の JSP ページ内で繰り返し使用できます。

次の追加 XML サポートもあります。

- ユーティリティ・タグによって、データを入力ストリームから XML DOM オブジェクトに変換します。
- キャッシングや SQL の操作などの機能に使用するタグは、XML オブジェクトを入力として取得したり、出力として送信することができます。

XML ユーティリティ・タグのサマリーを、表 1-1 に示します。XML の機能は dbOpen SQL タグおよび cacheXMLObj Web Object Cache タグにもあります。詳細は、第 5 章「XML と XSL に関するタグのサポート」を参照してください。

標準 JSP 1.2 の XML サポートの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

表 1-1 XML ユーティリティ・タグのサマリー

タグ	説明	属性
transform	XML データを、XSL 変換で HTTP クライアントまたは指定した XML DOM オブジェクトに出力します。	href、fromXMLObjName、toXMLObjName、toWriter
styleSheet	transform タグと同じです。	href、fromXMLObjName、toXMLObjName、toWriter
parsexml	入力ストリームから XML DOM オブジェクトに変換します。	resource、toXMLObjName、validateResource、root

データ・アクセス JavaBeans とタグ・ライブラリのサマリー

OC4J には、Oracle Database にアクセスするための一連のカスタム JavaBeans が用意されています。oracle.jsp.dbutil パッケージには、次の Bean が含まれています。

- ConnBean は、データベース接続をオープンします。この Bean は、データ・ソースおよび接続プーリングもサポートします。
- ConnCacheBean は、Oracle の接続キャッシング実装を使用してデータベースに接続します。(JDBC 2.0 が必要です。)
- DBBean は、データベースの問合せを実行します。
- CursorBean は、問合せ（UPDATE、INSERT および DELETE 文）およびストアド・プロシージャ・コールに対する一般的な DML サポートを提供します。

詳細は、4-2 ページの「データ・アクセスに関する JavaBeans」を参照してください。

OC4J には、JSP プログラマ向けに、JavaBeans の機能をラップする SQL 機能用のカスタム・タグ・ライブラリが用意されています。これらのタグのサマリーを、表 1-2 に示します。詳細は、4-11 ページの「データ・アクセス用 SQL タグ」を参照してください。

表 1-2 データ・アクセス・タグ・ライブラリのサマリー

タグ	説明	属性
dbOpen	データベース接続をオープンします。このタグは、データ・ソースおよび接続プーリングもサポートします。	connId、scope、dataSource、user、password、URL、commitOnClose、
dbClose	データベース接続をクローズします。	connId、scope
dbQuery	問合せを実行します。	queryId、connId、scope、output、maxRows、skipRows、bindParams、toXMLObjName
dbCloseQuery	問合せのカーソルをクローズします。	queryId
dbNextRow	結果セットの行を処理します。	queryId
dbExecute	SQL 文（DML または DDL）を実行します。	connId、scope、output、bindParams
dbSetParam	パラメータを設定して、`dbQuery` タグまたは `dbExecute` タグにバインドします。	name、value、scope
dbSetCookie	Cookie を設定します。	name、value、domain、comment、maxAge、version、secure、path

JSP Markup Language（JML）のカスタム・タグ・ライブラリのサマリー

重要： JML タグ・ライブラリは OC4J 10.1.2 実装で廃止され、それ以降の実装でサポート外になる予定です。かわりに、標準の JSTL タグ・ライブラリを使用してください。

JSP 仕様では Java 以外のスクリプト言語もサポートされていますが、使用される主な言語は Java です。JavaServer Pages テクノロジは、動的な Java 開発作業と静的な HTML 開発作業を分離するように設計されていますが、Web 開発者に Java の知識がない場合、特に Java のエキスパートがいない小規模な開発グループでは、Java を使用することが障害になります。

OC4J は、Java のかわりとなる JSP Markup Language（JML）のカスタム・タグを提供します。Oracle の JML タグ・ライブラリによって JSP タグの追加セットが提供されるため、Java 文を使用せずに JSP ページのスクリプトを作成できます。JML は、変数宣言、制御フロー、条件分岐、反復ループ、パラメータ設定およびオブジェクトのコール用のタグを提供します。JML のタグ・ライブラリでは、前述のように、XML 機能もサポートします。

次の例は、JML の `for` タグの使用方法を示しています。この例では、「Hello World」を下位のヘッダー（H1、H2、H3、H4、H5）に段階的に繰り返し出力します。

```
<jml:for id="i" from="<%= 1 %>" to="<%= 5 %>" >
     <H<%=i%>>
            Hello World!
     </H<%=i%>>
</jml:for>
```

JML タグ・ライブラリのサマリーを、表 1-3 に示します。詳細は、第 3 章「JSP Markup Language タグ」を参照してください。

表 1-3 JSP Markup Language タグ・ライブラリのサマリー

タグ	説明	属性
useVariable	単純な変数を宣言するための `jsp:useBean` タグにかわる便利なタグです。	id、scope、type、value
useForm	変数を宣言してリクエストから渡された値に設定する、便利な構文を提供します。	id、scope、type、param
useCookie	変数を宣言して Cookie 内の値に設定する、便利な構文を作成します。	id、scope、type、cookie
remove	オブジェクトをそのスコープから削除します。	id、scope
if	単一条件文を評価します。条件が true の場合は、`if` タグのボディが実行されます。	condition
choose	`choose` タグは、`when` タグおよび `otherwise` タグと関連付けて、複数の条件文を提供します。	（なし）
when	`choose` タグとともに使用します。	condition
otherwise	必要に応じて、`choose` タグおよび `when` タグとともに使用します。	（なし）

表 1-3 JSP Markup Language タグ・ライブラリのサマリー（続き）

タグ	説明	属性
for	Java の `for` ループと同様に、ループを使用した反復機能を提供します。	id、from、to
foreach	Java 配列、`Enumeration` インスタンスまたは `Vector` インスタンスで同種の値セット間を反復する機能を提供します。	id、in、limit、type
return	このタグに到達すると、後続の処理を行わずに実行がページから戻されます。	（なし）
flush	ページ・バッファの現行のコンテンツをクライアントに書き戻します。ただし、このタグの適用はそのページがバッファされている場合のみです。そうでない場合、効果はありません。	（なし）

注意： JSP 仕様 1.1 より前の Oracle JSP コンテナのバージョンでは、JML タグ・ライブラリに関して Oracle 固有のコンパイル時実装が使用されています。この実装は、付録 A「JML コンパイル時構文とタグ」で説明しているように、標準の実行時実装の代替として現在もサポートされています。

Oracle Application Server Personalization タグ・ライブラリのサマリー

Web サイトのパーソナライズは、行動およびデモグラフィック・データに基づいて、サイトのユーザー向けにリコメンデーションを作成する機能です。リコメンデーションは、ユーザーの Web セッション中にリアルタイムで作成されます。ユーザーの動作はデータベース・リポジトリ内に保存され、将来のユーザーの動作を予測するためのモデルの構築に使用されます。

OracleAS Personalization は、Oracle Database のデータ・マイニング・アルゴリズムを使用して、ユーザーに最も関連するコンテンツを選択します。通常、過去および現在におけるユーザーの動作に関する大量のデータを使用して、OracleAS Personalization リコメンデーション・エンジンによって、リコメンデーションが計算されます。これは、常識的な経験則に依存し、システム内でルールを手動定義する必要がある他のアプローチよりも優れています。

OracleAS Personalization タグ・ライブラリにより、幅広い層の JSP 開発者が、この機能を HTML、XML または JavaScript ページで使用できるようになります。タグ・インタフェースのレイヤーは、リコメンデーション・エンジンの Java API の最上部に位置します。

表 1-4 に、OracleAS Personalization タグ・ライブラリのサマリーを示します。詳細は、第 10 章「パーソナライズ・タグ」を参照してください。

表 1-4 OracleAS Personalization タグ・ライブラリのサマリー

タグ	説明	属性
startRESession	OracleAS Personalization リコメンデーション・エンジンのセッションを開始します。	REName、REURL、RESchema、REPassword、RECacheSize、REFlushInterval、applicationSession、createSession、userType、userID、storeUserIDIn、disableRecording

表 1-4 OracleAS Personalization タグ・ライブラリのサマリー（続き）

タグ	説明	属性
endRESession	リコメンデーション・エンジンのセッションを明示的に終了します。	（なし）
setVisitorToCustomer	匿名のビジターが、登録済のカスタマ・アカウントを作成する場合に、このタグを使用します。	customerID
getRecommendations	購入、ナビゲーションまたはレーティングに関する一連のリコメンデーションをリクエストします。	from、 fromHotPicksGroups、 storeResultsIn、 storeInterestDimensionIn、 maxQuantity、 tuningName、 tuningDataSource、 tuningInterestDimension、 tuningPersonalizationIndex、 tuningProfileDataBalance、 tuningProfileUsage、 filteringName、 filteringTaxonomyID、 filteringMethod、 filteringCategories、 sortOrder
getCrossSellRecommendations	リコメンデーションの基準として使用される一連の過去の項目（過去の購入など）に基づいて、購入、ナビゲーションまたはレーティングに関する一連のリコメンデーションをリクエストします。	storeResultsIn、 storeInterestDimensionIn、 fromHotPicksGroups、 inputItemList、 maxQuantity、 tuningName、 tuningDataSource、 tuningInterestDimension、 tuningPersonalizationIndex、 tuningProfileDataBalance、 tuningProfileUsage、 filteringName、 filteringTaxonomyID、 filteringMethod、 filteringCategories、 sortOrder
selectFromHotPicks	一連のホット・ピックス・グループのみからのリコメンデーションをリクエストします。	hotPicksGroups、 storeResultsIn、 storeInterestDimensionIn、 maxQuantity、 tuningName、 tuningDataSource、 tuningInterestDimension、 tuningPersonalizationIndex、 tuningProfileDataBalance、 tuningProfileUsage、 filteringName、 filteringTaxonomyID、 filteringMethod、 filteringCategories、 sortOrder

表 1-4 OracleAS Personalization タグ・ライブラリのサマリー（続き）

タグ	説明	属性
evaluateItems	タグに入力される一連の項目のみを評価します。	storeResultsIn、 taxonomyID、 inputItemList、 tuningName、 tuningDataSource、 tuningInterestDimension、 tuningPersonalizationIndex、 tuningProfileDataBalance、 tuningProfileUsage、 sortOrder
forItem	入力リストを必要とするタグに入力される個別の項目を選択します。	index、 itemList、 type、 ID
getNextItem	必要に応じて、一部のリコメンデーション・タグ内でこのタグを使用して、戻された項目にアクセスし、処理します。	storeTypeIn、 storeIDIn、 storeItemIn
recordNavigation	ナビゲーション項目をリコメンデーション・エンジン・セッションのキャッシュ内に記録します。	type、 ID、 index、 itemList
recordPurchase	購入項目をリコメンデーション・エンジン・セッションのキャッシュ内に記録します。	type、 ID、 index、 itemList
recordRating	レーティング項目をリコメンデーション・エンジン・セッションのキャッシュ内に記録します。	value、 type、 ID、 index、 itemList
recordDemographic	デモグラフィック項目をリコメンデーション・エンジン・セッションのキャッシュ内に記録します。	type、 value
removeNavigationRecord	このセッションで、以前にリコメンデーション・エンジン・セッションのキャッシュ内に記録されたナビゲーション項目を削除します。	type、 ID、 index、 itemList
removePurchaseRecord	このセッションで、以前にリコメンデーション・エンジン・セッションのキャッシュ内に記録された購入項目を削除します。	type、 ID、 index、 itemList
removeRatingRecord	このセッションで、以前にリコメンデーション・エンジン・セッションのキャッシュ内に記録されたレーティング項目を削除します。	value、 type、 ID、 index、 itemList
removeDemographicRecord	このセッションで、以前にリコメンデーション・エンジン・セッションのキャッシュ内に記録されたデモグラフィック項目を削除します。	type、 value

Web サービス・タグ・ライブラリのサマリー

OC4J が提供する Web サービス・タグ・ライブラリを使用すると、開発者は、Web サービスのクライアント・アプリケーション用 JSP ページを簡単に作成できます。実装には、RPC スタイルまたはドキュメント・スタイルのサービスをサポートする SOAP ベースの機能を使用します。クライアント・アプリケーションでは、Web Services Description Language（WSDL）文書にアクセスした後、WSDL 情報を使用して Web サービスの操作にアクセスします。

また、このタグ・ライブラリでは、Oracle による動的起動 API の実装を使用します。この実装の説明は、『Oracle Application Server Web Services 開発者ガイド』に記載されています。クライアント・アプリケーションで実行時に WSDL 文書を取得する場合は、動的起動 API を使用して、WSDL 文書に記述されている SOAP 操作を起動します。

この Web サービス・タグ・ライブラリのサマリーを表 1-5 に示します。詳細は、第 11 章「Web サービス・タグ」を参照してください。

表 1-5 Web サービス・タグ・ライブラリのサマリー

タグ	説明	属性
webservice	Web サービス・プロキシを作成します。このタグには、WSDL 文書の URL が必要です。プロキシの作成時にバインディングと SOAP 位置、またはサービス名とポートのいずれかを使用します。	wsdlUrl、 id、 scope、 binding、 soapLocation、 service、 port
map	Web サービス・プロキシは、`webservice` タグ内にネストされた `map` タグを使用して、SOAP/XML と Java との間の型マッピングのためのエントリを SOAP マッピング・レジストリに追加します。該当する型マッピングごとに 1 つの `map` タグを使用します。	localName、 namespaceUri、 javaType、 encodingStyle、 java2xmlClassName、 xml2javaClassName
property	必要に応じて、このタグを使用して名前 / 値ペアを指定し、Web サービスのクライアント・アプリケーションで使用できる複数のカスタム・プロパティを定義します。	name、 value
invoke	Web サービスの操作を起動します。`invoke` タグが Web サービス・プロキシにアクセスするには、`webservice` タグ内にネストされているか、あるいは `webservice` タグに作成された Web サービス・プロキシのスクリプト変数にアクセスする必要があります。	id、 operation、 webservice、 inputMsgName、 outputMsgName、 xmlToWriter、 toXMLObjName
part	操作の実行に入力メッセージ・パートの値が必要な場合は、各入力パートごとに 1 つの part タグを使用します。	name、 value

ファイル・アクセス・タグとメール・タグのサマリー

OC4J には、ファイルへのアクセス（アップロードとダウンロード）用とアプリケーションからの電子メール・メッセージ送信用のタグ・ライブラリが用意されています。

ファイルのアップロードには、`httpUpload` タグまたは `oracle.jsp.webutil.fileaccess.HttpUploadBean` JavaBean を使用できます。ファイルのダウンロードには、`httpDownload` タグまたは `HttpDownloadBean` JavaBean を使用できます。表 1-6 に、ファイル・アクセス・タグのサマリーを示します。詳細は、8-2 ページの「ファイル・アクセス JavaBeans とファイル・アクセス・タグ」を参照してください。

表 1-6 ファイル・アクセス・タグ・ライブラリのサマリー

タグ	説明	属性
httpUploadForm	このタグを使用すると、複数の部分に分割されてエンコードされたフォーム・データを使用して、アプリケーションにフォームを簡単に作成できます。このフォームによって、ユーザーはアップロードするファイルを指定できます。	formsAction、 maxFiles、 fileNameSize、 maxFileNameSize、 includeNumbers、 submitButtonText
httpUpload	クライアントからサーバーにファイルをアップロードします。ファイル・システムまたはデータベースにアップロードできます。	destination、 destinationType、 connId、 scope、 overwrite、 fileType、 table、 prefixColumn、 fileNameColumn、 dataColumn
httpDownload	サーバーからクライアントにファイルをダウンロードします。ファイル・システムまたはデータベースからダウンロードできます。	servletPath、 source、 sourceType、 connId、 scope、 recurse、 fileType、 table、 prefixColumn、 fileNameColumn、 dataColumn

電子メール・メッセージを、必要に応じて、サーバー・サイドまたはクライアント・サイドの添付とともに送信するには、`oracle.jsp.webutil.email.SendMailBean` JavaBean または `sendMail` タグを使用します。表 1-7 に `sendMail` タグのサマリーを示します。詳細は、8-18 ページの「メール JavaBean とメール・タグ」を参照してください。

表 1-7 sendMail タグのサマリー

タグ	説明	属性
sendMail	JSP ページから電子メール・メッセージを送信します。タグの機能には、グローバリゼーション・サポートが組み込まれています。	host、 sender、 recipient、 cc、 bcc、 subject、 contentType、 contentEncoding、 serverAttachment、 clientAttachment

EJB タグのサマリー

OC4J には、JSP ページでの Enterprise JavaBeans の使用を簡素化するカスタム・タグ・ライブラリが用意されています。OC4J EJB タグの機能は、J2EE 仕様に従っています。このタグを使用すると、`web.xml` ファイルの構成情報を使用して、EJB を名前でインスタンス化できます。

ホーム・インスタンスの作成、EJB インスタンスの作成および EJB のコレクション間の反復などに使用するタグがあります。表 1-8 に、EJB タグ・ライブラリのサマリーを示します。詳細は、9-11 ページの「EJB タグ」を参照してください。

表 1-8 EJB タグ・ライブラリのサマリー

タグ	説明	属性
useHome	EJB のホーム・インタフェースを参照して、そのインスタンスを作成します。	id、type、location、local
useBean	EJB をインスタンス化して使用します。このタグの機能は、JavaBean の標準の `jsp:useBean` タグに類似しています。	id、type、value、scope、local
createBean	EJB を最初にインスタンス化するときに、EJB `useBean` タグの `value` 属性を使用しない場合は、EJB `createBean` タグを `useBean` タグ内にネストし、EJB インスタンスの作成作業を実行する必要があります。	instance
iterate	EJB インスタンスのコレクション間の反復を行います（通常は、Entity Bean に多く使用されます）。	id、type、collection、max

JSP ユーティリティ・タグのサマリー

OC4J には、その他のユーティリティ・タグとして、日付を表示するタグ、適切な通貨で金額を表示するタグ、数値を表示するタグ、コレクション間を反復するタグ、ユーザーが特定のロールに属しているかどうかに応じてタグ・ボディを評価してインクルードするタグ、および現行のファイルの最終変更日付を表示するタグがあります。表 1-9 に、これらのタグのサマリーを示します。詳細は、9-17 ページの「一般的なユーティリティ・タグ」を参照してください。

表 1-9 一般的なユーティリティ・タグ・ライブラリのサマリー

タグ	説明	属性
displayCurrency	指定した金額をロケールの通貨として書式化して表示します。	amount、locale
displayDate	指定した日付をロケールに適した書式で表示します。	date、locale
displayNumber	指定した数値をロケールの書式および必要に応じて指定した書式で表示します。	number、locale、format
iterate	コレクション間を反復します。	id、type、collection、max

表 1-9 一般的なユーティリティ・タグ・ライブラリのサマリー（続き）

タグ	説明	属性
ifInRole	指定したアプリケーションのロールにユーザーが存在するかどうかに基づいて、タグ・ボディを評価し、JSP ページのボディにインクルードします。	role、include
lastModified	現行のファイルの最終変更日付を、ロケールに適した書式で表示します。	locale

Web アプリケーションに対する Oracle キャッシング・サポートのサマリー

この項では、次の情報を提供します。

- Oracle Application Server が全般的にサポートするキャッシング機能と OC4J JSP コンテナが特別にサポートするキャッシング機能の概要
- Oracle Application Server の他のキャッシング・コンポーネントに関連した OC4J Web Object Cache の役割の説明
- キャッシング機能に関連するタグ・ライブラリのサマリー

この項で紹介する Oracle タグ・ライブラリは、JSP 標準に準拠しています。

注意： この項で紹介する機能を使用したサンプル・アプリケーションは、OC4J のデモを参照してください。次の URL にアクセスして Oracle Technology Network（OTN）からダウンロードできます。

```
http://www.oracle.com/technology/tech/java/oc4j/demos/
```

Oracle Application Server と JSP のキャッシング機能

Oracle Application Server と OC4J には、次のキャッシング機能があります。

- Oracle Application Server Web Cache

  これは、アプリケーションの外部でメンテナンスされる HTTP レベルのキャッシュで、非常に高速なキャッシュ操作を行います。コンテンツ・ベースのキャッシュであるため、静的データ（HTML、GIF または JPEG ファイルなど）または動的データ（サーブレットや JSP の結果など）をキャッシュできます。このキャッシュが、アプリケーション外部にコンテンツ・ベースのキャッシュとして存在する場合は、オブジェクト（Java オブジェクトや XML DOM オブジェクトなど）を Java オブジェクト形式でキャッシュすることはできません。また、キャッシュ内のデータに適用可能な後処理操作は、Java ではコード化できず、キャッシュ自体で事前定義されます。

  OracleAS Web Cache には、Edge Side Includes をサポートする ESI プロセッサが備わっており、Web サーバーから独立して、動的にコンテンツのアセンブリができる XML スタイルのマークアップ言語がサポートされます。このテクノロジによって、開発者は、必要に応じてキャッシュ可能なページを個別のキャッシュ・オブジェクトに分割できます。OC4J では、このテクノロジが JESI タグ・ライブラリによってサポートされています。

  Edge Side Includes と OracleAS Web Cache の概要、および JESI タグ・ライブラリの詳細は、第 6 章「Edge Side Includes 用の JESI タグ」を参照してください。

  OracleAS Web Cache の詳細は、『Oracle Application Server Web Cache 管理者ガイド』を参照してください。

- OC4J Web Object Cache

  これはアプリケーション・レベルのキャッシュで、Java Web アプリケーションに埋め込まれ、メンテナンスされます。Web ベースとオブジェクト・ベースの両方を混合したキャッシュです。カスタム・タグ・ライブラリまたは API を使用すると、ユーザーはページ・フラグメントの境界を定義でき、JSP ページやサーブレットの実行の中間結果と部分的な結果を、キャッシュ内のオブジェクトとして取得、格納、再利用、処理および管理できます。ページ・フラグメントごとに別個のキャッシュ・オブジェクトを生成できます。作成されるオブジェクトは、HTML または XML のテキスト・フラグメント、XML DOM オブジェクトまたは Java のシリアライズ化可能なオブジェクトなどです。これらのオブジェクトは、HTTP リクエストおよびセッション・セマンティクスと関連付けると簡単にキャッシュできます。あるいは、HTTP 外部で再利用できます。たとえば、アプリケーションでキャッシュ内の XML オブジェクトを Simple Mail Transfer Protocol（SMTP）、Java Message Service（JMS）、アドバンスト・キューイング（AQ）または Simple Object Access Protocol（SOAP）を介して出力する場合に再利用できます。

  詳細は、第 7 章「Web Object Cache のタグと API」を参照してください。

- Oracle Application Server Java Object Cache

  Oracle Application Server Java Object Cache は、プロセス内、プロセス間およびローカル・ディスク上で Java オブジェクトを管理するための汎用 Java キャッシュです。Java Object Cache は、取得や作成が困難でコストがかかるオブジェクトのローカル・コピーを管理することによって、アプリケーションのパフォーマンスを大幅に向上させます。デフォルトでは、OC4J Web Object Cache は、Oracle Application Server Java Object Cache を基礎となるキャッシュ・リポジトリとして使用します。

  Java Object Cache の詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

JSP Web Object Cache のロール

Web アプリケーション全体の設定における OC4J Web Object Cache のロールを理解していることが重要です。このキャッシュは、Java レベルで機能し、サーブレットと JSP アプリケーションの HTTP 環境と緊密に統合されています。一方、Oracle Application Server Java Object Cache は、Java オブジェクト・レベルで機能しますが、HTTP とは統合されていません。OracleAS Web Cache の場合は、HTTP と緊密に統合され、Web Object Cache と比べてはるかに高速です。ただし、Java レベルでは動作せず、Web アプリケーション・コードで直接起動することはできません。たとえば、このキャッシュでは、スタイルシートを J2EE コンテナにあるキャッシュ内の DOM オブジェクトに適用したり、他のプロトコルにあるキャッシュ内の結果を再利用したり、または DOM 操作を直接実行することはできません。ただし、OracleAS Web Cache では、スタイルシートを、DOM オブジェクトではなく、HTTP を使用して元の Web サーバーからキャッシュされたテキスト・ベースの XML 文書に適用できます。

Web Object Cache は、アプリケーションの主要な Web キャッシュとしては使用しません。サーブレットや JSP ページを実行する Java Virtual Machine 内に埋め込まれた補助的なキャッシュです。Web Object Cache のキャッシュ結果に対する検索パスには、JVM、JSP およびサーブレットの各エンジンが含まれているため、通常は、OracleAS Web Cache と比べると、Web Object Cache からの方がページの処理にはるかに時間がかかります。

Web Object Cache は、OracleAS Web Cache や Oracle Application Server Java Object Cache の必要性を否定したり、排除するものではありません。あくまでも Web アプリケーションの全体的なフレームワークにおける補足的なキャッシング・コンポーネントであり、必要に応じて、他のキャッシング製品とともに使用する必要があります。実際に、Web Object Cache では、Java Object Cache をそのデフォルトのリポジトリとして使用しています。また、OC4J JESI タグと Web Object Cache タグを組み合せて使用することによって、Web Object Cache と OracleAS Web Cache を同一ページで使用できます。

Web Object Cache と OracleAS Web Cache の比較

OracleAS Web Cache は、プライマリ・キャッシング・コンポーネントと考えてください。キャッシュ内のページを HTTP クライアントに直接供給し、大量の HTTP 通信量を迅速に処理し、ほとんどの Web サイトの要件に適応しています。OracleAS Web Cache を使用すると、Web ページの全体または一部を（JESI タグを使用して）格納できます。キャッシュ内のページは、クライアントへの送信前に、ある程度カスタマイズできます。たとえば、Cookie 置換やページ・フラグメントの連結などが対象となります。

OracleAS Web Cache をできるだけ多用して、レスポンスを高速化し、Web アプリケーション・サーバーやバックエンド・データベースの負荷を軽減することをお薦めします。大部分の Web ページのキャッシング・ニーズに対しても、OracleAS Web Cache のみで対処できます。

OracleAS Web Cache の補完として Web Object Cache を使用すると、JSP およびサーブレットの実行の中間結果を取得し、後でこのキャッシュ結果を Java アプリケーション・ロジックの他の部分で再利用できます。キャッシュ結果のオブジェクトを繰り返し再利用できず、クライアントへ送信するにはキャッシュ内のオブジェクトに対して後処理が必要である場合は、Web アプリケーションで Web Object Cache を使用するメリットはありません。

Web Object Cache と Oracle Application Server Java Object Cache の比較

Web Object Cache では、Oracle Application Server Java Object Cache と比べると、実行結果の一部を動的 Web ページに格納および維持することがはるかに容易になります。Java Object Cache は、一般的な Java アプリケーションにとって純粋なオブジェクト・ベースのフレームワークであるため、埋込みが可能な HTTP 環境を認識できません。たとえば、キャッシュ・キーは、HTTP の Cookie またはセッションに自動的には依存しません。Java Object Cache を Web アプリケーション内で直接使用する場合、Web ページに必要なすべてのインタフェースは開発者の責任で行います。Java Object Cache でキャッシュ・メンテナンス・ポリシーを指定するにはプログラミングが必要ですが、かわりに構成ファイルを介してポリシーを指定できます。

キャッシングに関するタグ・ライブラリのサマリー

OC4J には、Oracle Application Server のキャッシング機能で使用する次の 2 つのタグ・ライブラリがあります。

- JESI タグ・ライブラリ
- Web Object Cache タグ・ライブラリ

この項では、この 2 つのライブラリのサマリーを示します。

JESI タグ・ライブラリのサマリー

OC4J では、JESI タグ・ライブラリを、ESI タグと Web キャッシングに関する Edge Side Includes 機能への便利なインタフェースとして提供しています。開発者は、Web アプリケーションで ESI タグを直接使用できますが、JESI タグは、JSP 環境で使用したほうがはるかに便利です。

表 1-10 に、JESI タグ・ライブラリのサマリーを示します。詳細は、6-12 ページの「Oracle JESI タグの説明」を参照してください。

表 1-10 JESI タグ・ライブラリのサマリー

タグ	説明	属性
control	control/include の使用モデルで、JSP ページのキャッシング特性を制御します。JESI `control` タグは、トップレベルのページまたは任意のインクルード・ページで使用できます。	expiration、maxRemovalDelay、cache、control
include	標準の `jsp:include` タグと同様、インクルード・ページの出力をインクルード先ページの出力に動的に挿入できます。ただし、この結果、インクルード・ページが ESI プロセッサにより（通常は OracleAS Web Cache 内で）処理されアセンブルされます。	page、alt、ignoreError、copyParam、flush
param	JESI `include` タグのサブタグです。1 つ以上の JESI `param` タグを使用して、インクルード・ページに追加の問合せパラメータを渡すことができます。	name、value
template	JSP ページを個別のキャッシュ・フラグメントに分割するときに（JESI `fragment` タグとともに）使用します。JESI `template` タグでは、集計ページ（フラグメント以外）のキャッシング動作を指定します。	expiration、maxRemovalDelay、cache、control
fragment	1 つ以上の JESI `fragment` タグを、JESI `template` タグ内、つまり JESI `template` の開始タグと終了タグ間で使用して、独立したキャッシュ可能なフラグメントを指定します。	expiration、maxRemovalDelay、cache、control
codeblock	JESI `template` タグのサブタグです。JESI `codeblock` タグを使用して、テンプレート・コード内でコード・ブロックを条件付きで実行するように指定できます。	execute
invalidate	JESI `object` サブタグとともにこのタグを使用し、OracleAS Web Cache によりキャッシュされた 1 つ以上のオブジェクトを明示的に無効にします。	url、username、password、config、output
object	JESI `invalidate` タグに必須のサブタグ。完全な URI または URI 接頭辞に基づいて、無効にするキャッシュ内のオブジェクトを指定します。	uri、prefix、maxRemovalDelay
cookie	JESI `object` タグのサブタグ。必要に応じて、無効化の追加基準として Cookie 情報を使用します。	name、value

表 1-10 JESI タグ・ライブラリのサマリー（続き）

タグ	説明	属性
header	JESI `object` タグのサブタグ。必要に応じて、無効化の追加基準として HTTP/1.1 のヘッダー情報を使用します。	name、value
personalize	オブジェクトに対するすべてのリクエストの Cookie 値置換を実行するように ESI プロセッサに指示し、ページのカスタマイズを可能にします。	name、default

Web Object Cache タグ・ライブラリのサマリー

OC4J Web Object Cache は、Java で記述された Web アプリケーションが、JSP ページやサーブレットなどの動的 Web ページが生成した部分的な結果と中間結果を取得、格納、再利用、後処理およびメンテナンスできるようにする機能です。プログラミング・インタフェースについては、タグ・ライブラリ（JSP ページでの使用）と Java API（サーブレットでの使用）を備えています。

表 1-11 に、Web Object Cache タグ・ライブラリのサマリーを示します。詳細は、7-15 ページの「Web Object Cache タグの説明」を参照してください。

表 1-11 Web Object Cache タグ・ライブラリのサマリー

タグ	説明	属性
cache	JSP アプリケーション内でテキスト・フラグメントなどのオブジェクトをキャッシュします。ただし、XML DOM オブジェクトや Java シリアライズ可能オブジェクトのキャッシング用には別個のタグがあります。	policy、ignoreCache、invalidateCache、scope、autoType、selectedParam、selectedCookies、reusableTimeStamp、reusableDeltaTime、name、expirationType、TTL、timeInaDay、dayInaWeek、dayInaMonth、writeThrough、printCacheBlockInfo、printCachePolicy、cacheRepositoryName、reportException
cacheXMLObj	通常、XML DOM オブジェクトをキャッシングする場合に `cache` タグのかわりに使用します。`cacheXMLObj` タグは、`cache` タグのすべての属性と XML 固有の追加パラメータをサポートしています。	`cache` タグのすべての属性および次の属性：fromXMLObjName、toXMLObjName、toWriter
useCacheObj	シリアライズ可能なすべての Java オブジェクトをキャッシュします。`useCacheObj` タグは、すべての `cache` タグ・パラメータとその機能固有の追加属性をサポートしています。	`cache` タグのすべての属性および次の属性：type、id、cacheScope

表 1-11 Web Object Cache タグ・ライブラリのサマリー（続き）

タグ	説明	属性
cacheInclude	cache タグ（ただし、cacheXMLObj タグや useCacheObj タグを除く）の機能と標準の jsp:include タグの機能を組み合せます。	policy、page、printCacheBlockInfo、reportException
invalidateCache	キャッシュ・オブジェクトをプログラムで無効にします。invalidateCache タグのほとんどの属性の動作は、cache タグ内の同名の属性の動作と同じです。	policy、ignoreCache、scope、autoType、selectedParam、selectedCookies、name、invalidateNameLike、page、autoInvalidateLevel、cacheRepositoryName、reportException

JavaServer Pages 標準タグ・ライブラリのサポート

Oracle Application Server 10*g* リリース 2（10.1.2）の OC4J JSP 製品には、Sun 社の JavaServer Pages 標準タグ・ライブラリ（JSTL）仕様 1.0 に指定されている JSTL の実装が組み込まれています。次の各項では、JSTL の機能と OC4J サポートの概要について説明します。

- JSTL の概要および概念
- JSTL 式言語のサマリー
- JSTL タグと追加機能の概要
- JSTL の使用上の注意と将来の考慮事項

JSTL に関する詳細は、次のサイトの仕様を参照してください。

`http://www.jcp.org/aboutJava/communityprocess/first/jsr052/index.html`

注意： JSTL 1.0 には、JSP 1.2 環境が必要です。

JSTL の概要および概念

JSTL は、Java などのスクリプト言語に不慣れな JSP ページ作成者を対象にしています。以前は、JSP ページで動的データを処理するには、スクリプトレットを使用していました。JSTL では、JSTL タグを使用すると、スクリプトレットが不要になります。

以前のバージョンの OC4J JSP 製品を使用した経験のある読者であれば、JSTL と Oracle JavaServer Pages Markup Language（JML）タグ・ライブラリの目的が似ていることがわかります。JML タグ・ライブラリはそのままサポートされますが、標準 JSTL の使用をお薦めします。1-23 ページの「JSTL の使用上の注意と将来の考慮事項」も参照してください。

JSTL には、次の主要な機能が含まれます。

- JSTL 式言語（EL）

  この式言語によって、アプリケーション・データへのアクセスと操作に必要なコードが簡素化され、スクリプトレットおよびリクエスト時の式が不要になります。次項の「JSTL 式言語のサマリー」を参照してください。

- 式言語サポート、条件付きロジックとフロー制御、イテレータ操作および URL ベース・リソースへのアクセスに使用する core タグ
- XML 処理、フロー制御および XSLT 変換に使用するタグ

- データベース・アクセスに使用する SQL タグ
- I18N 対応の国際化と書式化に使用するタグ

  （「I18N」は国際標準を表します。）

タグ・サポートは、前述の機能に基づいて 4 つの JSTL サブライブラリに分割されます。表 1-12 に、標準 TLD URI と各サブライブラリの接頭辞を示します。

表 1-12 JSTL サブライブラリ

機能	URI	接頭辞
Core	`http://java.sun.com/jstl/core`	c:
XML 処理	`http://java.sun.com/jstl/xml`	x:
SQL データベース・アクセス	`http://java.sun.com/jstl/sql`	sql:
I18N 国際化と書式化	`http://java.sun.com/jstl/fmt`	fmt:

詳細は、1-21 ページの「JSTL タグと追加機能の概要」を参照してください。

注意： JSP 1.2 コンテナを使用するよう制約されている場合、式言語モデルとリクエスト時の式モデルの両方をサポートするには、JSTL 1.0 実装が必要です。それぞれのモデルに対応する JSTL サブライブラリを使用すると、両方のモデルをサポートできます。サブライブラリ（core、XML、SQL および I18N）ごとに個別の TLD があるため、2 つのモデルに別々の TLD URI があります。

ほとんどのユーザーは、前述の URI に対応する式言語モデルを使用すると考えられます。リクエスト時の式モデルを使用する場合は、各 URI に「`_rt`」を追加して、適切な TLD にアクセスしてください。慣例では、各接頭辞にも「`_rt`」を追加します（**例：**「`c_rt:`」）。

JSTL 式言語のサマリー

JSTL 式言語は、JSP ページ間での情報の受渡しに JSP のスコープ属性とリクエスト・パラメータが有効であることを応用しています。JSTL 式言語を使用すると、JSP スクリプトレットとリクエスト時の式が不要になります。

JSTL 1.0 では、JSTL タグ属性の値にのみ、式言語を使用できます。

次の例では、JSTL `c:if` タグを使用して、企業リストから鉄鋼メーカーを選び出しています。

```
<c:if test="${company.industry == 'steel'}">
   ...
</c:if>
```

この項の後半では、JSTL 式言語の構文について要約し、OC4J JSP アプリケーションで JSTL 式言語の評価を可能にする方法について説明します。

JSTL 式言語の構文

次に、JSTL 式言語の主要な構文機能の簡単なサマリーを示します。その後に簡単な例をいくつか示します。

- 起動

  JSTL 式言語の起動には、`${`*`expression`*`}` 構文を使用します。最も基本的なセマンティックは、名前付き変数 `${foo}` を起動すると、メソッド・コール `PageContext.findAttribute(foo)` の結果と同じになることです。

- データ構造アクセス

  JavaBeans 内のデータおよびリスト、マップ、配列などのコレクション内のデータにアクセスするために、式言語は「.」構成メンバーと「[]」構成メンバーをサポートしています。「.」構成メンバーを使用すると、標準 Java 識別子の名前を持つプロパティにアクセスできます。「[]」構成メンバーは、より一般的なアクセスに使用します。ただし、有効な Java 識別子に対しては、「.」構成メンバーと等価です。たとえば、式 `foo.bar` と式 `foo["bar"]` の結果は同じになります。

- 関係演算子

  式言語は、関係演算子 `==`（または `eq`）、`!=`（または `ne`）、`<`（または `lt`）、`>`（または `gt`）、`<=`（`le`）、`>=`（または `ge`）をサポートしています。

- 算術演算子

  式言語は、算術演算子 `+`、`-`、`*`、`/`（または `div`）、`%`（または、剰余あるいは Modulo の場合は `mod`）をサポートしています。

- 論理演算子

  式言語は、論理演算子 `&&`（または `and`）、`||`（または `or`）、`!`（または `not`）、`empty` をサポートしています。

例 : 基本 次に、関係演算子「<=」（以下）を含む、式言語の基本的な起動例を示します。

```
<c:if test="${auto.price <= customer.priceLimit}">
   The <c:out value="${auto.makemodel}"/> is in your price range.
</c:if>
```

例 : コレクションへのアクセス 次に、Sun 社の JSTL 仕様 1.0 から、「.」構成メンバーと「[]」構成メンバーの使用例を示します。

```
<%-- "productDir" is a Map object containing the description of
      products, "preferences" is a Map object containing the
      preferences of a user --%>
product:
<c:out value="${productDir[product.custId]}"/>
shipping preference:
<c:out value="${user.preferences['shipping']}"/>
```

JSTL 式言語の暗黙的なオブジェクト

JSTL では、次の暗黙的なオブジェクトを提供します。

- `pageScope`: ページ・スコープ変数にアクセスできます。
- `requestScope`: リクエスト・スコープ変数にアクセスできます。
- `sessionScope`: セッション・スコープ変数にアクセスできます。
- `applicationScope`: アプリケーション・スコープ変数にアクセスできます。
- `pageContext`: JSP ページのページ・コンテキストの全プロパティにアクセスできます。
- `param`: これは Java の `Map` オブジェクトです。この場合、`param["foo"]` は、リクエスト・パラメータ `foo` に関連付けられた最初の文字列の値を戻します。
- `paramValues`: たとえば、`paramValues["foo"]` は、リクエスト・パラメータ `foo` に関連付けられた全文字列の値の配列を戻します。
- `header`: `param` を使用する場合と同様に、リクエスト・ヘッダーに関連付けられた最初の文字列の値にアクセスできます。
- `headerValues`: `paramValues` を使用する場合と同様に、リクエスト・ヘッダーに関連付けられた全文字列の値にアクセスできます。
- `initParam`: コンテキスト初期化パラメータにアクセスできます。
- `cookie`: リクエストで受信した Cookie にアクセスできます。

JSTL 式言語の追加機能

式言語は次の機能も提供します。

- 式の評価に失敗してもリカバリ可能と判断された場合は、デフォルト値を表示できます。
- アプリケーションのデータが、タグ属性または式言語の演算子に必要な型と完全に一致しない場合に、結果値の型を必要な型に変換できます。

詳細は、JSTL 仕様 1.0 を参照してください。

JSTL タグと追加機能の概要

次の各項では、JSTL タグのサマリーを示し、JSTL の追加機能について説明します。

- スコープ変数
- 構成データと Config クラス
- JSTL タグのサマリー

スコープ変数

JSTL タグは、JSP のスコープ属性を使用してデータを使用可能にします。この属性はスコープ変数とも呼ばれ、スクリプト変数のかわりに使用されます。このようにデータを使用可能にできる JSTL タグには、その属性に `var` および `scope` が含まれ、次のように使用されます。

- `var`: 公開対象の変数を指定します。
- `scope`: `page`（デフォルト）、`request`、`session` または `application` のいずれかの変数のスコープを指定します。

`scope` 属性は、`NESTED` 変数（常に `page` スコープを持つ）には必要ありません。ただし、JSTL での変数は、`AT_END`（終了タグからそのページの終わりまで使用可能）になります。

次の例では、コア・ライブラリ・イテレータ操作タグ `forEach` と式言語サポート・タグ `out` を使用して、`employees` コレクションの現行の項目を公開します。

```
<c:forEach var="employee" items="${customers}">
   The current employee is <c:out value="${customer}"/>
</c:forEach>
```

構成データと Config クラス

JSTL には、スコープ変数を使用して特定スコープの JSP 構成データを動的にオーバーライドする機能が含まれています。この操作を行うには、`javax.servlet.jsp.jstl.core.Config` クラスの機能を使用します。

JSP 仕様によると、JSP ページ・コンテキスト内に存在するすべてのスコープ（`page`、`request`、`session` および `application`）は、単一の名前空間を形成する必要があります。つまり、スコープ変数の名前は、ページの実行ごとに一意である必要があります。

`Config` クラスには、構成パラメータ名を透過的に操作して、各スコープに独自の名前空間を設定する機能があります。この機能を効果的に使用すると、構成パラメータを特定のスコープに対してのみ設定できます。

詳細は、JSTL 仕様 1.0 を参照してください。

JSTL タグのサマリー

表 1-13 に、機能グループに編成された JSTL タグのサマリーを示します。各グループごとに JSTL 標準のタグ接頭辞について説明しています。

表 1-13 JavaServer Pages 標準タグ・ライブラリのサマリー

タグ・グループ	グループの説明	各タグ
Core、EL サポート	式の評価と現行の JspWriter オブジェクトへの結果の出力、スコープ変数の値やターゲット・オブジェクトのプロパティ値の設定、スコープ変数の削除、およびネストされた操作によってスローされる Throwable インスタンスの捕捉などを行うタグが含まれています。	c:out、 c:set、 c:remove、 c:catch
Core、条件付き	test 属性によって true と評価されたボディ・コンテンツの評価および手動による相互に排他的な条件付き実行パスの指定を行うタグが含まれています。when タグと otherwise タグは、choose タグとともに使用します。	c:if、 c:choose、 c:when、 c:otherwise
Core、イテレータ	ボディの実行をオブジェクトのコレクション間で（または指定した回数）反復し、指定のデリミタで区切られた一連のトークン間で反復するタグが含まれています。	c:forEach、 c:forTokens
Core、URL 関連	URL ベース・リソースのコンテンツのインポート、適切なリライティング・ルールを使用した URL の作成、HTTP リダイレクトのクライアントへの送信、およびリクエスト・パラメータの URL への追加などを行うタグが含まれています。param タグは、import タグ、url タグおよび redirect タグのサブタグです。	c:import、 c:url、 c:redirect、 c:param
XML、コア	XML 文書の解析、XPath 式の評価と現行の JspWriter オブジェクトへの結果の出力、および XPath 式の評価とスコープ変数への結果の格納などを行うタグが含まれています。（XPath に関しては、この表の次にある「注意」を参照してください。）	x:parse、 x:out、 x:set
XML、フロー制御	指定した XPath 式の評価とその式で true と評価されたコンテンツのレンダリング、手動による相互に排他的な条件付き実行パスの指定、および指定した XPath 式の評価と結果間でのボディ実行の反復などを行うタグが含まれます。when タグと otherwise タグは、choose タグとともに使用します。	x:if、 x:choose、 x:when、 x:otherwise、 x:forEach
XML、変換	XSLT スタイルシート変換の文書への適用および変換パラメータの設定を行うタグが含まれます。param タグは、transform タグのサブタグです。	x:transform、 x:param

表 1-13 JavaServer Pages 標準タグ・ライブラリのサマリー（続き）

タグ・グループ	グループの説明	各タグ
SQL	データベースの問合せ、データベースの更新（UPDATE/INSERT/DELETE）、問合せと更新用トランザクション・コンテキストの設定、スコープ変数またはデータ・ソース構成変数へのデータ・ソースのエクスポート、SQL 文でのパラメータ・プレースホルダ（?）の値の設定、および java.util.Date 型の場合におけるパラメータ・プレースホルダ値の設定などを行うタグが含まれます。param タグと dateParam タグは、query タグと update タグのサブタグです。	sql:query、 sql:update、 sql:transaction、 sql:setDataSource、 sql:driver、 sql:param、 sql:dateParam
I18N、国際化	指定したロケールのロケール構成変数への格納、タグ内で使用する I18N ローカライゼーション・コンテキストの作成、タグ外で使用するローカライゼーション・コンテキストの作成と格納、リソース・バンドル内のローカル・メッセージの参照、およびリクエスト文字エンコーディングの設定などを行うタグが含まれます。param タグを message タグとともに使用すると、message タグのパラメータを置換できます。	fmt:locale、 fmt:bundle、 fmt:message、 fmt:param、 fmt:requestEncoding
I18N、書式化	書式化や解析用タイムゾーンの指定、指定済タイムゾーンのスコープ変数またはタイムゾーン構成変数への格納、ロケールや特別なカスタマイズに対する数値の適切な書式化、書式化済数値の文字列表現の解析、ロケールまたは特別なカスタマイズに対する日付または時間の書式化、書式化済日付または時間の文字列表現の解析などを行うタグが含まれます。	fmt:timeZone、 fmt:setTimeZone、 fmt:formatNumber、 fmt:parseNumber、 fmt:formatDate、 fmt:parseDate

---

注意： XML 処理用の JSTL タグは、XPath（XML パス）に関する W3C 勧告に基づいています。XPath は、XML 文書の各部分を指定および選択するための簡潔な表記規則を提供します。詳細は、次の Web サイトを参照してください。

```
http://www.w3.org/TR/xpath
```

JSTL の使用上の注意と将来の考慮事項

次の考慮事項に注意してください。

- Oracle Application Server 10*g* リリース 2（10.1.2）の JSTL 実装は、Jakarta 1.0.3 JSTL バージョンに基づいており、OC4J での使用に適しています。Jakarta の詳細は、次の URL を参照してください。

  ```
  http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html
  ```

- OC4J が提供している JML、XML およびデータ・アクセス（SQL）のカスタム・タグ・ライブラリは、JSTL より前に開発されたもので、JSTL と重複する機能が含まれています。標準に準拠するために、原則として、カスタム・ライブラリではなく、JSTL の使用をお薦めします。JML ライブラリは、将来のリリースでサポート外になる予定です。JSTL で使用できないカスタム・ライブラリの機能のうち、有効と判断される機能は、必要に応じて、JSTL 標準に採用される予定です。

- JSTL XML タグのフィルタ機能が OC4J で動作するには、OC4J の起動時に SAX ドライバを設定する必要があります。設定しないと、JSTL デモの `Filter.jsp` で次の例外が発生します。

  ```
  javax.servlet.jsp.JspException: System property org.xml.sax.driver not specified
  ```

  OC4J をスタンドアロンで起動する際に、次のオプションをコマンドラインで使用します。

  ```
  -Dorg.xml.sax.driver=oracle.xml.parser.v2.SAXParser
  ```

  Oracle Application Server の起動時に、この設定はシステム・プロパティを使用して指定できます。

他の Oracle コンポーネントのタグ・ライブラリの概要

OC4J 以外の他の多くの Oracle コンポーネントにも、JSP タグ・ライブラリが備わっています。この項では次のライブラリのサマリーを示します。

- Oracle ADF Business Components タグ・ライブラリ
- Oracle JDeveloper User Interface Extension（UIX）タグ・ライブラリ
- Oracle ADF Business Components UIX タグ・ライブラリ
- Oracle Application Server Wireless Location タグ・ライブラリ
- Oracle Application Server MapViewer タグ・ライブラリ
- Oracle Ultra Search タグ・ライブラリ
- Oracle Application Server Portal タグ・ライブラリ
- Oracle Business Intelligence Beans タグ・ライブラリ
- Oracle Application Server マルチメディア・タグ・ライブラリ

この項で紹介する Oracle タグ・ライブラリは、JSP 標準に準拠しています。

以降の説明は、基礎となるコンポーネントを事前に理解していることを前提としています。

Oracle ADF Business Components タグ・ライブラリ

Oracle Application Development Framework（Oracle ADF）は、Oracle JDeveloper の一部として提供され、コンポーネント・テクノロジの Oracle ADF Business Components が組み込まれています。これには、カスタムの Business Components データ・タグのライブラリが含まれます。

これらのデータ・タグは、Business Components データ・ソースとの相互作用に対して単純なタグ・ベースのアプローチを提供するため、表示、編集および完全な DML 制御機能によるビジネス・コンポーネントへの完全なアクセスが可能になります。Business Components による JSP アプリケーションの構築にタグ・ベースのアプローチを使用すると、大規模な Java プログラミングが不要になり、HTML ページのコーディングに類似したアプローチとなります。

詳細は、Oracle JDeveloper オンライン・ヘルプまたは Oracle Technology Network（OTN）の次の URL を参照してください。

```
http://www.oracle.com/technology/products/jdev/content.html
```

Oracle JDeveloper User Interface Extension（UIX）タグ・ライブラリ

Oracle JDeveloper には、User Interface Extension（UIX）タグと呼ばれるカスタム・タグのセットが用意されています。これらのタグは、UIX の制御機能を起動し、タブ、ボタン、表、ヘッダーおよび Oracle Browser Look and Feel を実装する他のレイアウトとナビゲーショナル・コンポーネントをレンダリングするための HTML を生成します。

タグは、いくつかのパレット・ページ（UIX JSP Border Layout、UIX JSP Form、UIX JSP Layout、UIX JSP Message Components、UIX JSP Page Layout、UIX JSP Simple Components および UIX JSP Table）に含まれています。

詳細は、Oracle JDeveloper オンライン・ヘルプまたは Oracle Technology Network（OTN）の次の URL を参照してください。

```
http://www.oracle.com/technology/products/jdev/content.html
```

Oracle ADF Business Components UIX タグ・ライブラリ

UIX JSP ページには、Business Components データ・タグとデータのプレゼンテーションを簡素化する Business Components UIX convenience タグの両方を含めることができます。

Business Components UIX convenience タグは、`ApplicationModule` データ・タグに依存して、Business Components アプリケーション・モジュールからデータ・ソースを取得します。Business Components UIX タグに加えて、UIX JSP ページの（UIX 以外の）Business Components タグも使用できます。

詳細は、Oracle JDeveloper オンライン・ヘルプまたは Oracle Technology Network（OTN）の次の URL を参照してください。

```
http://www.oracle.com/technology/products/jdev/content.html
```

Oracle Reports タグ・ライブラリ

Oracle Reports は、Oracle Reports Developer（Oracle Developer Suite のコンポーネント）および Oracle Reports Services（Oracle Application Server のコンポーネント）で構成されています。Oracle Reports Developer には、JSP レポートの作成に使用されるデータ・モデル・オブジェクトにインクルードされるタグが含まれています。Oracle Reports カスタム・タグを使用して、レポート・ブロックおよびグラフを既存の JSP ファイルに迅速に追加できます。これらのタグをテンプレートとして使用することにより、独自のデータ駆動 Java コンポーネントを構築し、JSP ベースの Web レポート用 HTML ページに挿入できます。

主要タグは、レポート・ブロックを区切り、定義します。これらのタグ内に含まれる他のタグは、レポート・データのコンテンツおよびルック・アンド・フィールを定義します。

詳細は、Reports Builder の「ヘルプ」メニューを使用して表示可能な Oracle Reports Developer オンライン・ヘルプを参照してください。Oracle Reports の詳細は、Oracle Technology Network（OTN）を参照してください。

```
http://www.oracle.com/technology/products/reports/index.html
```

Oracle Application Server Wireless Location タグ・ライブラリ

ロケーション・ベースのアプリケーションの開発者には、次のような特別なサービスが必要です。

- ジオコーディング : 地理座標を住所に関連付けます。
- マッピング : ポイント、ポイントのセット、ルーティングまたはドライブ操作をグラフィック・マップに提供します。
- ルーティング : 運転案内を提供します。
- ビジネス・リスト（職業別電話帳のようなもの）: ビジネスをリージョン別とカテゴリ別、またはリージョン別と名前別に表示します。
- 交通 : 交通事故、工事など、交通の流れに影響を与える出来事に関する情報を提供します。

OracleAS Wireless のロケーション・アプリケーション・コンポーネントは、ジオコーディングの実行、運転案内およびビジネス・リストの参照に使用する一連の API です。既存の主要プロバイダを API にマップするサービス・プロキシが組み込まれています。また、将来、プロバイダの追加が予定されています。

JSP 開発者用のタグ・ライブラリが用意されています。詳細は、『Oracle Application Server Wireless 開発者ガイド』を参照してください。

Oracle Application Server MapViewer タグ・ライブラリ

OracleAS MapViewer は、Oracle Spatial または Oracle Locator（単に Locator とも呼ぶ）により管理される空間データを使用してマップをレンダリングするためのプログラム可能なツールです。OracleAS MapViewer には、複雑な空間データ問合せやマップのレンダリングを表示しないツールと、上級ユーザー向けにカスタマイズ可能なオプションを提供するツールが用意されています。これらのツールはプラットフォームに依存せずに配布でき、マップ・レンダリング・アプリケーションと統合するように設計されています。

利便性を考慮して、OracleAS MapViewer にはマップ・リクエストの発行に使用できる JSP タグ・ライブラリが組み込まれています。

詳細は、『Oracle Application Server MapViewer ユーザーズ・ガイド』を参照してください。

Oracle Ultra Search タグ・ライブラリ

Oracle Ultra Search には、開発時にコンテンツ検索機能を JSP アプリケーションに取り込むために開発者が使用するカスタム・タグ・ライブラリがあります。このライブラリには、次の機能が組み込まれています。

- 拡張問合せフォームのレンダリングに使用する検索属性、グループ、言語および値リスト（LOV）を取得する機能
- ヒットした結果セット間を反復し、結果ページのレンダリングに使用するドキュメントの属性とプロパティを取得する機能
- 関連性取得による検索機能と、合計ヒット件数の予測を実行する機能

詳細は、『Oracle Ultra Search 管理者ガイド』を参照してください。または、Oracle Ultra Search JSP タグ・ライブラリの Oracle Ultra Search オンライン・ドキュメントを参照してください。

Oracle Application Server Portal タグ・ライブラリ

OracleAS Portal を使用すると、開発者は次の項目を実行できます。

- 関連情報やアプリケーションをカスタマ、従業員およびパートナに配信するためのインターネット・ポータルの構築とデプロイ
- 効率的なオンライン・ツールを使用した、コードなしのポータルの迅速な開発
- シングル・サインオンとセルフサービス公開によるユーザーの生産性の向上
- オープン標準に基づいた 250 を超えるビルトイン・ポートレットによる迅速な値の追加

OracleAS Portal タグ・ライブラリを使用すると、開発者はさらに簡単にカスタマイズ可能なインターネット・ポータルを構築できます。開発者は、内部 JSP ページを作成してポータル・データベース内に格納し、ポータルの実行時にダウンロードするか、外部 JSP ページを作成してファイル・システム内に格納するか、またはこの 2 つの方法を組み合せて使用できます。

詳細は、Oracle Technology Network の次の URL から入手可能な『Oracle Application Server Integrating JavaServer Pages with OracleAS Portal』を参照してください。

```
http://www.oracle.com/technology/products/ias/portal/pdf/pd_jsps_10g.pdf
```

Oracle Business Intelligence Beans タグ・ライブラリ

Oracle Business Intelligence Beans（OracleBI Beans）製品は、分析アプリケーションを迅速に開発するための Java コンポーネント、ユーティリティおよび JSP タグ・ライブラリで構成されています。OracleBI Beans アプリケーションは、Oracle Database の OLAP 機能を利用します。OracleBI Beans を使用すると、HTML と Java の両方のクライアント・アプリケーションを開発できます。

注意： OracleBI Beans 製品は Oracle Developer Suite（OracleDS）のコンポーネントで、Oracle JDeveloper で使用します。

OracleBI Beans には、次の Java コンポーネント・グループが組み込まれています。

- プレゼンテーション Bean: データを表示、操作、印刷するための Bean
- OLAP Bean: Oracle OLAP データ・ソースと相互作用する Bean
- 永続性サービス : OracleBI Beans カタログ内のオブジェクト定義の格納と取得をサポートする一連の Java パッケージ

開発者の利便性を考慮して、OracleBI Beans には JSP タグ・ライブラリが組み込まれています。JDeveloper を使用すると OracleBI Beans JSP ページを作成できます。JDeveloper ウィザードでは、使用するタグの関連情報が要求され、コード化されたタグが JSP ページに挿入されます。

詳細は、OracleBI Beans のオンライン・ヘルプを参照してください。「Web モジュールの構築」で、「JSP タグの使用方法」、「BI Beans JSP タグのリスト」を順にクリックします。

Oracle Application Server マルチメディア・タグ・ライブラリ

Oracle Application Server には、マルチメディア・タグ・ライブラリが用意されています。これはカスタム JSP タグ・ライブラリであり、開発者や Web ページ作成者が JSP ページ内でマルチメディア HTML タグを生成し、マルチメディア・データを *inter*Media オブジェクトにアップロードするときに使用します。

Oracle *inter*Media を使用すると、Oracle Database でイメージ、オーディオ、ビデオおよび他のメディア・データを格納、取得、管理および操作し、他のエンタープライズ情報と統合できます。特に、Oracle *inter*Media では、Oracle により管理されてバイナリ・ラージ・オブジェクト、ファイル・ベース・ラージ・オブジェクト、メディア・データを含む URL および専用サーバーに格納されるメディア・データの格納、取得、管理および操作がサポートされます。Oracle *inter*Media には、アプリケーションからリレーショナル・インタフェースおよびオブジェクト・インタフェースを介してアクセスできます。

Oracle *inter*Media では、Java クラスに似たオブジェクト・タイプを使用してメディア・データが記述されます。これらの *inter*Media オブジェクトには、共通のメディア・データ格納モデルがあります。また、Oracle *inter*Media には、ユーザーが *inter*Media オブジェクトを使用して Java アプリケーションを記述できるように Java クラスも用意されています。Oracle Database インスタンスとの間でメディア・データを容易に取得およびアップロードできるように、サーブレットと JavaServer Pages 用の Oracle *inter*Media Java クラスも用意されています。

マルチメディア・タグ・ライブラリには、メディア・データ取得用の一連のタグと、メディア・データ・アップロード用の一連のタグが含まれています。メディア取得用のマルチメディア JSP タグには、一連の共通属性とタグ固有のメディア・レンダリング属性があります。共通属性は、カスタム取得属性、データベース接続属性、メディア・アクセス属性、メディア・キャッシュ制御属性、表および列属性です。メディア・レンダリング属性は、各メディア取得タグを使用して記述されます。

マルチメディア JSP タグの詳細は、『Oracle Application Server Multimedia Tag Library for JSP 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。

2

拡張型用の JavaBeans

この章では、OC4J が提供する JavaBeans について説明します。この JavaBeans は拡張型として使用します。JSP ページの場合、この拡張型は、Java のプリミティブ型や `java.lang` 型と比較して多くのメリットがあります。

この章には、次の項目が含まれます。

JML 拡張型の概要

JSP ページでは、スカラー値の表現をコアとなる Java の型に依存しています。ただし、次の型カテゴリは、いずれも JSP ページでの使用に適していません。

- プリミティブ型（int、float および double など）

  これらの型の値には、スコープを指定できません。つまり、JSP のスコープ・オブジェクト（page、request、session または application スコープ）に格納できません。これは、スコープ・オブジェクトに格納できるのはオブジェクトのみであるためです。

- 標準の java.lang パッケージにあるラッパー・クラス（Integer、Float および Double など）

  これらの型の値はオブジェクトであるため、理論上では JSP スコープ・オブジェクトに格納できます。ただし、ラッパー・クラスは JavaBean モデルに準拠しておらず、引数が 0（ゼロ）のコンストラクタを提供していないため、jsp:useBean 操作でこの型を宣言できません。

  さらに、ラッパー・クラスのインスタンスは不変です。値を変更するには、インスタンスを新たに作成して、適切に割り当てる必要があります。

JDK 1.4 以下では、JML 拡張型により、これらの制限事項を回避するための便利なオプションが提供されます。OC4J では、最も一般的な Java の型に対してラッパーとして動作する、次の JavaBean クラスを oracle.jsp.jml パッケージに用意しています。

- boolean 値を表す JmlBoolean
- int 値を表す JmlNumber
- double 値を表す JmlFPNumber
- String 値を表す JmlString

これらのクラスには、それぞれ 1 つの属性 value があり、値の取得、各種フォーマットへの入力値の設定、その値が複数のフォーマットのいずれかに指定した値と等しいかどうかのテスト、およびその値の文字列への変換などを行うメソッドが組み込まれています。

あるいは、getValue() メソッドおよび setValue() メソッドを使用せずに、他の Bean のように、jsp:getProperty タグおよび jsp:setProperty タグを使用することもできます。

次の例では、application スコープを持つ count と呼ばれる JmlNumber インスタンスを作成します。

```
<jsp:useBean id="count" class="oracle.jsp.jml.JmlNumber" scope="application" />
```

値が他の場所で設定済の場合は、その値に次のようにしてアクセスできます。

```
<h3> The current count is <%=count.getValue() %> </h3>
```

次の例では、request スコープを持つ maxSize と呼ばれる JmlNumber インスタンスを作成し、そのインスタンスを setProperty を使用して設定します。

```
<jsp:useBean id="maxSize" class="oracle.jsp.jml.JmlNumber" scope="request" >
    <jsp:setProperty name="maxSize" property="value" value="<%= 25 %>" />
</jsp:useBean>
```

JML 拡張型の説明

この項では、4 つの拡張型（JmlBoolean、JmlNumber、JmlFPNumber および JmlString）のパブリック・メソッドについて説明し、例を示します。

> **注意：** JML 拡張型を使用するには、ojsputil.jar ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J に含まれています。

JmlBoolean 型

JmlBoolean オブジェクトは、Java boolean 値を表します。

getValue() メソッドと setValue() メソッドは、Bean の value プロパティを Java boolean 値として取得または設定します。

- boolean getValue()
- void setValue(boolean)

setTypedValue() メソッドには複数のシグネチャがあり、value プロパティを文字列（「true」や「false」など）、java.lang.Boolean 値、Java boolean 値または JmlBoolean 値から設定できます。文字列の入力については、java.lang.Boolean クラスの valueOf() メソッドと同じルールに従って、文字列の変換が実行されます。

- void setTypedValue(String)
- void setTypedValue(Boolean)
- void setTypedValue(boolean)
- void setTypedValue(JmlBoolean)

equals() メソッドは、value プロパティが、指定した Java boolean 値と等しいかどうかをテストします。

- boolean equals(boolean)

typedEquals() メソッドには複数のシグネチャがあり、value プロパティの値が、指定した文字列（「true」や「false」など）、java.lang.Boolean 値または JmlBoolean 値と等しいかどうかをテストします。

- boolean typedEquals(String)
- boolean typedEquals(Boolean)
- boolean typedEquals(JmlBoolean)

toString() メソッドは、value プロパティを java.lang.String 値（「true」または「false」のいずれか）として戻します。

- String toString()

JmlNumber 型

JmlNumber オブジェクトは、Java int 値に等しい 32 ビットの数値を表します。

getValue() メソッドと setValue() メソッドは、Bean の value プロパティを Java int 値として取得または設定します。

- int getValue()
- void setValue(int)

setTypedValue() メソッドには複数のシグネチャがあり、value プロパティを文字列、java.lang.Integer 値、Java int 値または JmlNumber 値から設定できます。文字列の入力については、java.lang.Integer クラスの decode() メソッドと同じルールに従って、文字列の変換が実行されます。

- void setTypedValue(String)
- void setTypedValue(Integer)
- void setTypedValue(int)
- void setTypedValue(JmlNumber)

equals() メソッドは、value プロパティが、指定した Java int 値と等しいかどうかをテストします。

- boolean equals(int)

typedEquals() メソッドには複数のシグネチャがあり、value プロパティの値が、指定した文字列（「1234」など）、java.lang.Integer 値または JmlNumber 値と等しいかどうかをテストします。

- boolean typedEquals(String)
- boolean typedEquals(Integer)
- boolean typedEquals(JmlNumber)

toString() メソッドは、value プロパティを等価の java.lang.String 値（「1234」など）として戻します。このメソッドの機能は、java.lang.Integer クラスの toString() メソッドの機能と同じです。

- String toString()

JmlFPNumber 型

JmlFPNumber オブジェクトは、Java double 値に等しい 64 ビットの浮動小数点の数値を表します。

getValue() メソッドと setValue() メソッドは、Bean の value プロパティを Java double 値として取得または設定します。

- double getValue()
- void setValue(double)

setTypedValue() メソッドには複数のシグネチャがあり、value プロパティを文字列（「3.57」など）、java.lang.Integer 値、Java int 値、java.lang.Float 値、Java float 値、java.lang.Double 値、Java double 値または JmlFPNumber 値から設定できます。文字列の入力については、java.lang.Double クラスの valueOf() メソッドと同じルールに従って、文字列の変換が実行されます。

- void setTypedValue(String)
- void setTypedValue(Integer)
- void setTypedValue(int)
- void setTypedValue(Float)
- void setTypedValue(float)

- void setTypedValue(Double)
- void setTypedValue(double)
- void setTypedValue(JmlFPNumber)

equals() メソッドは、value プロパティが、指定した Java double 値と等しいかどうかをテストします。

- boolean equals(double)

typedEquals() メソッドには複数のシグネチャがあり、value プロパティの値が、指定した文字列（「3.57」など）の、java.lang.Integer 値、Java int 値、java.lang.Float 値、Java float 値、java.lang.Double 値、Java double 値または JmlFPNumber 値と等しいかどうかをテストします。

- boolean typedEquals(String)
- boolean typedEquals(Integer)
- boolean typedEquals(int)
- boolean typedEquals(Float)
- boolean typedEquals(float)
- boolean typedEquals(Double)
- boolean typedEquals(JmlFPNumber)

toString() メソッドは、value プロパティを java.lang.String 値（「3.57」など）として戻します。このメソッドの機能は、java.lang.Double クラスの toString() メソッドの機能と同じです。

- String toString()

JmlString 型

JmlString オブジェクトは、java.lang.String 値を表します。

getValue() メソッドと setValue() メソッドは、Bean の value プロパティを java.lang.String 値として取得または設定します。setValue() コール内の入力が null の場合、value プロパティは、空（0（ゼロ）長）の文字列に設定されます。

- String getValue()
- void setValue(String)

toString() メソッドの機能は、getValue() メソッドの機能と同じです。

- String toString()

setTypedValue() メソッドは、value プロパティを、指定した JmlString 値に従って設定します。JmlString 値が null の場合、value プロパティは、空（0（ゼロ）長）の文字列に設定されます。

- void setTypedValue(JmlString)

isEmpty() メソッドは、value プロパティが空（0（ゼロ）長）の文字列かどうかをテストします。

- boolean isEmpty()

equals() メソッドには 2 つのシグネチャがあり、value プロパティが、指定した java.lang.String 値または JmlString 値と等しいかどうかをテストします。

- boolean equals(String)
- boolean equals(JmlString)

JML 拡張型の例

この例では、スコープで単純な型を管理する JML 拡張型 JavaBeans の使用例を示します。このページでは、4 つのセッション・オブジェクト（各 JML 型に対して 1 つ）を宣言します。また、このページは開発者が各型に値を入力できるフォームを提供しています。新しい値が入力されたフォームには、新しい値と以前設定された値の両方が表示されます。この出力の生成中に、ページはセッション・オブジェクトを新しいフォーム値によって更新します。

```
<jsp:useBean id = "submitCount" class = "oracle.jsp.jml.JmlNumber" scope = "session" />

<jsp:useBean id = "bool" class = "oracle.jsp.jml.JmlBoolean" scope = "session" >
        <jsp:setProperty name = "bool" property = "value" param = "fBoolean" />
</jsp:useBean>

<jsp:useBean id = "num" class = "oracle.jsp.jml.JmlNumber" scope = "session" >
        <jsp:setProperty name = "num" property = "value" param = "fNumber" />
</jsp:useBean>

<jsp:useBean id = "fpnum" class = "oracle.jsp.jml.JmlFPNumber" scope = "session" >
        <jsp:setProperty name = "fpnum" property = "value" param = "fFPNumber" />
</jsp:useBean>

<jsp:useBean id = "str" class = "oracle.jsp.jml.JmlString" scope = "session" >
        <jsp:setProperty name = "str" property = "value" param = "fString" />
</jsp:useBean>


<HTML>

<HEAD>
        <META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=iso-8859-1">
        <META NAME="GENERATOR" Content="Visual Page 1.1 for Windows">
        <TITLE>Extended Datatypes Sample</TITLE>
</HEAD>

<BODY BACKGROUND="images/bg.gif" BGCOLOR="#FFFFFF">

<% if (submitCount.getValue() > 1) { %>
        <h3> Last submitted values </h3>
        <ul>
                <li> bool: <%= bool.getValue() %>
                <li> num: <%= num.getValue() %>
                <li> fpnum: <%= fpnum.getValue() %>
                <li> string: <%= str.getValue() %>
        </ul>
<% }

   if (submitCount.getValue() > 0) { %>

        <jsp:setProperty name = "bool" property = "value" param = "fBoolean" />
        <jsp:setProperty name = "num" property = "value" param = "fNumber" />
        <jsp:setProperty name = "fpnum" property = "value" param = "fFPNumber" />
        <jsp:setProperty name = "str" property = "value" param = "fString" />

        <h3> New submitted values </h3>
        <ul>
                <li> bool: <jsp:getProperty name="bool" property="value" />
                <li> num: <jsp:getProperty name="num" property="value" />
                <li> fpnum: <jsp:getProperty name="fpnum" property="value" />
                <li> string: <jsp:getProperty name="str" property="value" />
        </ul>
<% } %>
```

```
<jsp:setProperty name = "submitCount" property = "value"
                 value = "<%= submitCount.getValue() + 1 %>" />

<FORM ACTION="index.jsp" METHOD="POST" ENCTYPE="application/x-www-form-urlencoded">
<P> <pre>
 boolean test: <INPUT TYPE="text" NAME="fBoolean" VALUE="<%= bool.getValue() %>" >
  number test: <INPUT TYPE="text" NAME="fNumber" VALUE="<%= num.getValue() %>" >
fpnumber test: <INPUT TYPE="text" NAME="fFPNumber" VALUE="<%= fpnum.getValue() %>" >
  string test: <INPUT TYPE="text" NAME="fString" VALUE= "<%= str.getValue() %>" >
</pre>

<P> <INPUT TYPE="submit">

</FORM>

</BODY>

</HTML>
```

3

JSP Markup Language タグ

重要： JML タグ・ライブラリは OC4J 10.1.2 実装で廃止され、それ以降の実装でサポート外になる予定です。かわりに、標準の JSTL タグ・ライブラリを使用してください。

この章では、Oracle JSP Markup Language（JML）タグ・ライブラリについて説明します。このタグ・ライブラリには、開発者が JSP ページのスクリプトを Java 文を使用せずに作成できる一連の JSP タグが含まれています。JML ライブラリには、変数宣言、制御フロー、条件分岐、反復ループ、パラメータ設定およびオブジェクトのコールなどに使用するタグが含まれています。

この章には、次の項目が含まれます。

注意： この章で説明するライブラリは、標準の実行時実装を使用していますが、Oracle 固有のコンパイル時実装によってもサポートされています。コンパイル時実装の構文とタグは、付録 A「JML コンパイル時構文とタグ」で説明されています。実行時タグではなく、コンパイル時タグを使用する場合の一般的な考慮事項は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

JSP Markup Language（JML）タグ・ライブラリの概要

OC4J で提供される JSP Markup Language（JML）タグ・ライブラリは、JSP 標準に従って開発されています。JML タグは、Java に不慣れな JSP 開発者による構文のコーディングを簡素化するためのタグです。JML タグには、ロジック / フロー制御と Bean バインドという 2 つの主要なカテゴリがあります。

この 2 つの項目については次の項を参照してください。

- JML タグ・ライブラリの基本的な考え方
- JML タグ・カテゴリ

JML タグを使用する場合は、次の要件に注意してください。

- `ojsputil.jar` ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J のインストール時に予約済のタグ・ライブラリ・ディレクトリにインストールされます。
- タグ・ライブラリ・ディスクリプタ・ファイル `jml.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`jml.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jml.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意： OC4J が提供する JML カスタム・タグ・ライブラリは、JavaServer Pages 標準タグ・ライブラリ（JSTL）より前に開発され、JSTL と重複する機能が含まれています。標準に準拠するために、原則として、JSTL の使用をお薦めします。1-18 ページの「JavaServer Pages 標準タグ・ライブラリのサポート」を参照してください。

JML タグ・ライブラリの基本的な考え方

JavaServer Pages テクノロジは、次の 2 つの異なる開発者層を対象にしています。

- プライマリ・スキルが Java プログラミングである開発者
- プライマリ・スキルが静的コンテンツの設計（特に HTML での）で、スクリプト作成の経験が不十分な開発者

JML タグ・ライブラリは、Java の知識の有無にかかわらず、ほとんどの Web 開発者が、プログラム・フロー制御機能を完全に使用して、JSP アプリケーションのアセンブルが行えるよう設計されています。

このモデルでは、別の Java 開発者によって開発された JavaBeans 内にビジネス・ロジックが格納されていると想定しています。

JML タグ・カテゴリ

JML タグ・ライブラリには、表 3-1 のサマリーに示すように、2 つの機能カテゴリに分割される機能セットがあります。

表 3-1 JML タグの機能カテゴリ

タグ・カテゴリ	機能	タグ
Bean バインディング・タグ	指定した JSP スコープで JavaBean の宣言または宣言解除を行います。3-3 ページの「Bean バインディング・タグの説明」を参照してください。	useVariable、useForm、useCookie、remove
ロジック / フロー制御タグ	反復ループや条件分岐などに使用するコード・フローを定義する構文を簡素化します。3-6 ページの「ロジックおよびフロー制御タグの説明」を参照してください。	if、choose..when..[otherwise]、foreach、return、flush

JSP Markup Language（JML）タグの説明

次の各項では、現行の JSP 実行時実装でサポートされている JML タグについて説明します。

- Bean バインディング・タグの説明
- ロジックおよびフロー制御タグの説明

注意：

- このタグ構文では、接頭辞「jml:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を `taglib` ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

Bean バインディング・タグの説明

次の各項では、Bean バインディング操作で使用する次の JML タグについて説明します。

- JML useVariable タグ
- JML useForm タグ
- JML useCookie タグ
- JML remove タグ

JML useVariable タグ

単純な変数を宣言するための `jsp:useBean` タグにかわる便利なタグです。

構文

```
<jml:useVariable id = "beanInstanceName"
             [ scope = "page" | "request" | "session" | "application" ]
               type = "string" | "boolean" | "number" | "fpnumber"
             [ value = "stringLiteral" ] />
```

属性

- id（必須）: 宣言する変数を指定します。
- scope: 変数の継続時間またはスコープを定義します（jsp:useBean タグと同じです）。デフォルトのスコープは、「page」です。
- type（必須）: 変数の型を指定します。型指定については、JmlString、JmlBoolean、JmlNumber または JmlFPNumber を参照してください。
- value: 変数を、文字列リテラルまたは <%=...%> 構文で囲んだ JSP 式として、宣言内に直接設定します。指定しない場合、この値は、最後に設定した値のまま（すでに存在する場合）か、あるいはデフォルト値で初期化されます。指定すると、この宣言がオブジェクトをインスタンス化する宣言か、または単に名前付きスコープからオブジェクトを取得する宣言かに関係なく、指定した値が必ず設定されます。

例 次に例を示します。

```
<jml:useVariable id = "isValidUser" type = "boolean" value = "<%= dbConn.isValid() %>"
scope = "session" />
```

これは、次の例と同じです。

```
<jsp:useBean id = "isValidUser" class = "oracle.jsp.jml.JmlBoolean" scope = "session" />
<jsp:setProperty name="isValidUser" property="value" value = "<%= dbConn.isValid() %>" />
```

JML useForm タグ

変数を宣言してリクエストから渡された値に設定する、便利な構文を提供します。

構文

```
<jml:useForm id = "beanInstanceName"
           [ scope = "page" | "request" | "session" | "application" ]
           [ type = "string" | "boolean" | "number" | "fpnumber" ]
             param = "requestParameterName" />
```

属性

- id（必須）: 宣言または参照する変数を指定します。
- scope: 変数の継続時間またはスコープを定義します（jsp:useBean タグと同じです）。デフォルトは、「page」です。
- type: 変数の型を指定します。型指定については、JmlString、JmlBoolean、JmlNumber または JmlFPNumber を参照してください。デフォルトは、「string」です。
- param（必須）: 変数の設定に使用する値を持つリクエスト・パラメータの名前を指定します。リクエスト・パラメータが存在する場合は、この宣言によって生じた変数かどうかに関係なく、その変数値は必ず更新されます。リクエスト・パラメータが存在しない場合、変数値はそのままです。

例 次の例では、JmlString 型の user という名前のセッション変数を、user という名前のリクエスト・パラメータの値に設定します。

```
<jml:useForm id = "user" type = "string" param = "user" scope = "session" />
```

これは、次の例と同じです。

```
<jsp:useBean id = "user" class = "oracle.jsp.jml.JmlString" scope = "session" />
<jsp:setProperty name="user" property="value" param = "user" />
```

JML useCookie タグ

変数を宣言して Cookie 内の値に設定する、便利な構文を作成します。

構文

```
<jml:useCookie id = "beanInstanceName"
            [ scope = "page" | "request" | "session" | "application" ]
            [ type = "string" | "boolean" | "number" | "fpnumber" ]
              cookie = "cookieName" />
```

属性

- id（必須）: 宣言または参照する変数を指定します。
- scope: 変数の継続時間またはスコープを定義します。この属性はオプションです。デフォルトは、「page」です。
- type: 変数の型を識別します。型指定については、JmlString、JmlBoolean、JmlNumber または JmlFPNumber を参照してください。デフォルトは、「string」です。
- cookie（必須）: この変数の設定に使用する値を持つ Cookie の名前を指定します。Cookie が存在する場合は、この宣言によって生じた変数かどうかに関係なく、その変数値は必ず更新されます。Cookie が存在しない場合、変数値はそのままです。

例 次の例では、JmlString 型の user という名前のリクエスト変数を、user という名前の Cookie の値に設定します。

```
<jml:useCookie id = "user" type = "string" cookie = "user" scope = "request" />
```

これは、次の例と同じです。

```
<jsp:useBean id = "user" class = "oracle.jsp.jml.JmlString" scope = "request" />
<%
     Cookies [] cookies = request.getCookies();
     for (int i = 0; i < cookies.length; i++) {
             if (cookies[i].getName().equals("user")) {
                     user.setValue(cookies[i].getValue());
                     break;
             }
     }
%>
```

JML remove タグ

オブジェクト（通常は Bean）をそのスコープから削除します。

構文

```
<jml:remove id = "beanInstanceName"
          [ scope = "page" | "request" | "session" | "application" ] />
```

属性

- id（必須）: 削除対象の Bean の名前を指定します。
- scope: 削除対象の Bean のスコープを指定します。指定しない場合、スコープは、1）page、2）request、3）session、4）application の順序で検索されます。id に一致する名前を持つ最初のオブジェクトが削除されます。

例 次の例では、セッションの user オブジェクトを削除します。

```
<jml:remove id = "user" scope = "session" />
```

これは、次の例と同じです。

```
<% session.removeValue("user"); %>
```

ロジックおよびフロー制御タグの説明

次の各項では、ロジックとフロー制御に使用する次の JML タグについて説明します。

- JML if タグ
- JML choose...when...[otherwise] タグ
- JML for タグ
- JML foreach タグ
- JML return タグ
- JML flush タグ

これらのタグは、Java の経験が十分でない開発者を対象としており、反復ループや条件分岐などの Java ロジックおよびフロー制御構文のかわりに使用できます。

JML if タグ

単一条件文を評価します。条件が true の場合は、`if` タグのボディが実行されます。

構文

```
<jml:if condition = "<%= jspExpression %>" >
     ...body of jml:if tag (executed if the condition is true)...
</jml:if>
```

属性

- `condition`（必須）: 評価対象の条件式を指定します。

例 次の E-Commerce の例では、ユーザーのショッピング・カート情報を表示します。このコードは、現在の T シャツの注文を保持している変数が空かどうかをチェックします。空でない場合、ユーザーが注文したサイズが表示されます。`currTS` が、`JmlString` 型であるとします。

```
<jml:if condition = "<%= !currTS.isEmpty() %>" >
     <S>(size: <%= currTS.getValue().toUpperCase() %>)</S> 
</jml:if>
```

JML choose...when...[otherwise] タグ

`choose` タグは、`when` タグおよび `otherwise` タグと関連付けて、複数の条件文を提供します。

`choose` タグのボディには、1 つ以上の `when` タグを含めます。それぞれの `when` タグが 1 つの条件を表します。最初の `when` 条件が true の場合は、`when` タグのボディが実行されます。実行されるのは、最大 1 つの `when` ボディです。

`when` 条件がすべて false で、オプションの `otherwise` タグが指定されている場合は、`otherwise` タグのボディが実行されます。

構文

```
<jml:choose>
    <jml:when condition = "<%= jspExpression %>" >
        ...body of 1st jml:when tag (executed if the condition is true)...
    </jml:when>
    ...
    [...optional additional when tags...]
    [ <jml:otherwise>
        ...body of jml:otherwise tag (executed if all when conditions false)...
    </jml:otherwise> ]
</jml:choose>
```

属性 when タグは次の属性を使用します。

- condition（必須）：評価対象の条件式を指定します。

choose タグと otherwise タグには属性がありません。

例 次の E-Commerce の例では、ユーザーのショッピング・カート情報を表示します。このコードは、注文があったかどうかをチェックします。注文があった場合は、現在の注文が表示されます。そうでない場合は、購入するかどうかをユーザーに再度尋ねます。この例では、現在の注文を表示するコードが省略されています。orderedItem が、JmlBoolean 型であると想定します。

```
<jml:choose>
     <jml:when condition = "<%= orderedItem.getValue() %>"  >
          You have changed your order:
             -- output the current order --
     </jml:when>
     <jml:otherwise>
          Are you sure we can't interest you in something, cheapskate?
     </jml:otherwise>
</jml:choose>
```

JML for タグ

Java の for ループと同様に、ループを使用した反復機能を提供します。

id 属性は、java.lang.Integer 型のローカル・ループ変数です。範囲は、from 属性の値から開始し、値は、ループのボディの実行を終了するたびに増分され、to 属性の値を超えるまで増分されます。

範囲を超えると、制御は for 終了タグの次の最初の文に移動します。

> **注意：** 降順の範囲はサポートされていません。これは、from 値は to 値より小さいか等しい必要があるためです。

構文

```
<jml:for id = "loopVariable"
         from = "<%= jspExpression %>"
         to = "<%= jspExpression %>" >
     ...body of jml:for tag (executed once at each value of range, inclusive)...
</jml:for>
```

属性

- id（必須）：ループ変数の名前であり、範囲内の現行の値が保持されます。内容は、java.lang.Integer 値であり、タグ・ボディ内でのみ使用できます。
- from（必須）：範囲の開始を指定します。この属性は、Java int 値を評価する必要がある式です。
- to（必須）：範囲の終了を指定します。この属性は、Java int 値を評価する必要がある式です。

例 次の例では、「Hello World」を下位のヘッダー（H1、H2、H3、H4、H5）に段階的に繰り返し出力します。

```
<jml:for id="i" from="<%= 1 %>" to="<%= 5 %>" >
     <H<%=i%>>
            Hello World!
     </H<%=i%>>
</jml:for>
```

JML foreach タグ

同種の値セット間を反復する機能を提供します。タグ・ボディはセット内の要素ごとに 1 回実行されます。セットが空の場合、ボディは実行されません。

id 属性は、現行のセット要素の値が含まれたローカル・ループ変数です。その型は、type 属性に指定されています。指定する型は、そのセット要素の型と（使用可能な場合）一致している必要があります。

このタグは、現在次の型のデータ構造の反復をサポートしています。

- Java 配列
- java.util.Enumeration
- java.util.Vector

構文

```
<jml:foreach id = "loopVariable"
             in = "<%= jspExpression %>"
             limit = "<%= jspExpression %>"
             type = "package.class" >
...body of jml:foreach tag (executes once for each element in data structure)...
</jml:foreach>
```

属性

- id（必須）: ループ変数の名前です。ループ変数には、反復の各ステップでの現行の要素の値が保持されます。この属性は、タグ・ボディ内でのみ使用できます。その型は、type 属性に指定されている型と同じです。
- in（必須）: 反復対象の一連の値を含む Java 配列、Enumeration オブジェクトまたは Vector オブジェクトを評価する JSP 式を指定します。
- limit（必須）: 反復の最大数を定義する Java int 値を、セット内の要素数に関係なく評価する JSP 式を指定します。
- type（必須）: ループ変数の型を指定します。指定する型は、そのセット要素の型と（使用可能な場合）一致している必要があります。

例 次の例は、リクエスト・パラメータ間を反復します。

```
<jml:foreach id="name" in="<%= request.getParameterNames() %>" type="java.lang.String" >
    Parameter: <%= name %>
    Value: <%= request.getParameter(name) %> <br>
</jml:foreach>
```

このかわりに、複数の値を持つパラメータを処理する場合は、次のようになります。

```
<jml:foreach id="name" in="<%= request.getParameterNames() %>" type="java.lang.String" >
    Parameter: <%= name %>
    Value: <jml:foreach id="val" in="<%=request.getParameterValues(name)%>"
                        type="java.lang.String" >
                <%= val %> :
           </jml:foreach>
    <br>
</jml:foreach>
```

JML return タグ

このタグに到達すると、後続の処理を行わずに実行がページから戻されます。

構文

```
<jml:return />
```

属性

なし

例 次の例では、タイマーが期限切れになると、ページの処理なしに実行が戻されます。

```
<jml:if condition="<%= timer.isExpired() %>" >
     You did not complete in time!
     <jml:return />
</jml:if>
```

JML flush タグ

ページ・バッファの現行のコンテンツをクライアントに書き戻します。ただし、このタグの適用はそのページがバッファされている場合のみです。そうでない場合、効果はありません。

構文

```
<jml:flush />
```

属性

なし

例 次の例は、現行のページ・コンテンツを、コストのかかる操作の前にフラッシュします。

```
<jml:flush />
<% myBean.expensiveOperation(out); %>
```

4

データ・アクセス JavaBeans とデータ・アクセス・タグ

この章では、OC4J が提供する、サーブレットおよび JSP ページからデータベースへのアクセスに使用する JavaBeans とタグについて説明します。

この章には、次の項目が含まれます。

データ・アクセスに関する JavaBeans

OC4J 製品には、データベースへのアクセスに使用できる一連の JavaBeans が含まれています。次の各項で Bean について説明します。

- データ・アクセス JavaBeans の概要
- データ・ソースと接続プーリング用のデータ・アクセスのサポート
- データ・アクセス JavaBean の説明

注意： ここで説明する JavaBeans は、4-11 ページの「データ・アクセス用 SQL タグ」で説明されているタグで使用します。一般的に、これらの Bean とタグは、適切な JDBC ドライバ・クラスがあることを前提として、Oracle 以外のデータベースで使用できます。ただし、後述されている機能の多くは、注意にあるように Oracle 固有の機能です。

データ・アクセス JavaBeans の概要

OC4J には、データベースにアクセスするための一連のカスタム JavaBeans が用意されています。`oracle.jsp.dbutil` パッケージには、次の Bean が含まれています。

- `ConnBean` は、データベース接続をオープンします。この Bean は、データ・ソースおよび接続プーリングもサポートします。関連情報は、4-3 ページの「データ・ソースと接続プーリング用のデータ・アクセスのサポート」を参照してください。
- `ConnCacheBean` では、データベース接続に Oracle JDBC の接続キャッシング実装が使用されます。この Bean には、JDBC 2.0 が必要です。
- `DBBean` は、データベースの問合せを実行します。それ自体に接続機能がありますが、データ・ソースはサポートしません。
- `CursorBean` は、問合せ（`UPDATE`、`INSERT` および `DELETE` 文）およびストアド・プロシージャ・コールに対する一般的な DML サポートを提供します。

この項の説明は、Oracle JDBC の操作知識があることを前提にしています。必要に応じて、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

データ・アクセス JavaBeans を使用するには、`ojsputil.jar` ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J のインストール時にインストールされます。XML 関連のメソッドおよび機能については、Oracle Application Server で提供される `xsu12.jar`（JDK 1.2.x 以上）のファイルも必要です。

また、Oracle Database および JDK 1.4 の場合は、`ojdbc14.jar` などの適切な JDBC ドライバ・クラスがインストール済で、クラスパスに存在している必要があります。

データ・ソースと接続プーリング用のデータ・アクセスのサポート

データ・アクセス JavaBeans は、データ・アクセス・タグ・ライブラリと同様に、接続プロパティを指定するためのデータ・ソースの使用をサポートしています。接続プーリングのサポート手段も実装されます。この機能によって、Oracle 接続オブジェクトおよび OC4J 接続オブジェクトの両方がサポートされます。

JSP ページでデータ・ソースを使用するには、データ・ソース、その JNDI 名およびその接続プロパティとプーリング・プロパティを定義する必要があります。OC4J では、`data-sources.xml` ファイルの `<data-source>` 要素でこれらを定義します。次に例を示します。

```
<data-source
   class="oracle.jdbc.pool.OracleDataSource"
   name="jdbc/ejbpool/OracleDS"
   location="jdbc/ConnectionDS"
   ejb-location="jdbc/ejbpool/OracleDS"
   url="jdbc:oracle:thin:@myhost:1521/myservice"
   username="scott"
   password="tiger"
   min-connections="3"
   max-connections="50"
   wait-timeout="10"
   inactivity-timeout="30" />
```

列挙データ・ソースの JNDI 参照では、`ejb-location` JNDI 名のみの使用をお薦めします。データ・ソースの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

データ・アクセス JavaBean の説明

次の各項では、データ・アクセス JavaBeans（`ConnBean`、`ConnCacheBean`、`DBBean` および `CursorBean`）の属性とメソッドについて、データ・ソースを使用する例を含めて説明します。

- データベース接続用 ConnBean
- 接続キャッシング用 ConnCacheBean
- 問合せ専用 DBBean
- DML とストアド・プロシージャ用 CursorBean
- 例 : データ・ソースを利用した ConnBean と CursorBean の使用

データベース接続用 ConnBean

`oracle.jsp.dbutil.ConnBean` を使用して、単純なデータベース接続（プーリングやキャッシングを使用しない接続）を確立します。

注意 : データ・ソースを必要としない問合せのみの場合は、それ自体に接続機能がある `DBBean` を使用するほうが簡単です。

`ConnBean` には、次のプロパティがあります。データ・ソースを使用する場合、`user`、`password` および `URL` の各プロパティは不要です。

- `dataSource`: データ・ソース位置の JNDI 名

  これは、データ・ソースをサポートする環境に対してのみ有効です。OC4J でデータ・ソースを設定する方法の詳細は、4-3 ページの「データ・ソースと接続プーリング用のデータ・アクセスのサポート」を参照してください。

- `user`: データベース・スキーマのユーザー ID
- `password`: ユーザー・パスワード

- URL: データベース接続文字列
- stmtCacheSize: Oracle JDBC の文のキャッシング用キャッシュ・サイズ

  stmtCacheSize を設定すると、Oracle JDBC の文のキャッシングを有効にできます。
- executeBatch: Oracle JDBC のバッチ更新用バッチ・サイズ

  executeBatch を設定すると、Oracle JDBC のバッチ更新を有効にできます。
- preFetch: Oracle JDBC の行のプリフェッチでプリフェッチする文の数

  preFetch を設定すると、Oracle JDBC の行のプリフェッチを有効にできます。
- commitOnClose: 接続のクローズ時に commit を実行するかどうかを「true」または「false」で指定します。

  commitOnClose の値は、接続のクローズ時に commit を自動的に実行するかどうかを示します。「true」に設定すると commit が実行され、「false」に設定すると rollback が実行されます。Oracle9*i*AS リリース 2 より前では、常に commit が自動的に実行されましたが、現行のリリースのデフォルトは自動 rollback です。commitOnClose プロパティには、移行を簡素化するための下位互換性があります。

  アプリケーション全体の commit-on-close がアプリケーションの web.xml ファイルに設定されている場合がありますが、ConnBean プロパティの設定は、その設定に自動的に依存しているわけではありません。JSP ページで dbOpen タグのかわりに ConnBean を使用する場合は、commit-on-close コンテキスト・パラメータの値を取得し、ConnBean インスタンスの commitOnClose 値に明示的に設定してください。次に、commit-on-close コンテキスト・パラメータを設定する web.xml エントリのサンプルを示します。

```
<context-param>
   <param-name>commit-on-close</param-name>
   <param-value>true</param-value>
</context-param>
```

  注意： 文のキャッシング、バッチ更新および行のプリフェッチの詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

ConnBean には、次のようなプロパティの setter メソッドと getter メソッドが用意されています。

- void setDataSource(String)
- String getDataSource()
- void setUser(String)
- String getUser()
- void setPassword(String)
- String getPassword()
- void setURL(String)
- String getURL()
- void setStmtCacheSize(int)
- int getStmtCacheSize()
- void setExecuteBatch(int)
- int getExecuteBatch()
- void setPreFetch(int)
- int getPreFetch()

- `void setCommitOnClose(String)`
- `String getCommitOnClose()`

> **注意：** JSP ページで使用する JavaBean と同様に、`ConnBean` プロパティは、setter メソッドを直接使用しないで、`jsp:setProperty` 操作で設定できます。

次のメソッドを使用して、接続のオープンとクローズ、または接続の状態を確認します。

- `void connect()`

  `ConnBean` プロパティの設定を使用して、データベース接続を確立します。

- `void close()`

  接続とオープン中のカーソルをクローズします。

- `boolean isConnectionClosed()`

  接続がクローズされているかどうかを判断します。

カーソルをオープンし、`CursorBean` オブジェクトを戻すには、次のメソッドを使用します。

- `CursorBean getCursorBean(int, String)`

または

- `CursorBean getCursorBean(int)`

  次の内容を入力します。

  - JDBC 文のタイプを指定するための `Statement` オブジェクト用の `CursorBean.PLAIN_STMT`、`PreparedStatement` オブジェクト用の `CursorBean.PREP_STMT` または `CallableStatement` オブジェクト用の `CursorBean.CALL_STMT` のいずれかの `int` 型定数
  - 実行する SQL 操作を指定する文字列（オプション）

    または、文を実行する `CursorBean` メソッド・コールに SQL 操作を指定することもできます。

`CursorBean` 機能の詳細は、4-8 ページの「DML とストアド・プロシージャ用 CursorBean」を参照してください。

接続キャッシング用 ConnCacheBean

データベース接続には、`oracle.jsp.dbutil.ConnCacheBean` を使用して、Oracle JDBC の接続キャッシング機能を使用し、JDBC 2.0 の接続プーリングを使用します。接続キャッシングの詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

> **注意：** データ・ソースや単純な接続オブジェクトを使用するには、かわりに `ConnBean` を使用します。

`ConnCacheBean` には、次のプロパティがあります。

- `user`: データベース・スキーマのユーザー ID
- `password`: ユーザー・パスワード
- `URL`: データベース接続文字列
- `maxLimit`: このキャッシュによる許容最大接続数
- `minLimit`: このキャッシュに存在する最小接続数

  使用接続数がこの最小接続数に満たない場合は、キャッシュのアイドル・プールにも接続があります。

- stmtCacheSize: Oracle JDBC の文のキャッシング用キャッシュ・サイズ

  stmtCacheSize を設定すると、Oracle JDBC の文のキャッシング機能を有効にできます。Oracle JDBC の文のキャッシング機能と制限の詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

- cacheScheme: キャッシュのタイプ

  これは、次のいずれかの int 型定数によって示されます。

  - DYNAMIC_SCHEME: 最大制限を超えて新規のプール接続を作成できます。ただし、各接続は、用意した論理的な接続インスタンスが使用されなくなり次第、自動的にクローズおよび解放されます。
  - FIXED_WAIT_SCHEME: 最大制限に達すると、新規接続は、既存の接続オブジェクトが解放されるまで待機します。
  - FIXED_RETURN_NULL_SCHEME: 最大制限に達すると、接続オブジェクトが解放されるまで、新規接続は失敗します（null が戻ります）。

ConnCacheBean クラスには、プロパティを取得および設定する次のメソッドが組み込まれています。

- void setUser(String)
- String getUser()
- void setPassword(String)
- String getPassword()
- void setURL(String)
- String getURL()
- void setMaxLimit(int)
- int getMaxLimit()
- void setMinLimit(int)
- int getMinLimit()
- void setStmtCacheSize(int)
- int getStmtCacheSize()
- void setCacheScheme(int)

  ConnCacheBean.DYNAMIC_SCHEME、ConnCacheBean.FIXED_WAIT_SCHEME または ConnCacheBean.FIXED_RETURN_NULL_SCHEME を指定します。

- int getCacheScheme()

  ConnCacheBean.DYNAMIC_SCHEME、ConnCacheBean.FIXED_WAIT_SCHEME または ConnCacheBean.FIXED_RETURN_NULL_SCHEME を戻します。

ConnCacheBean クラスは、oracle.jdbc.pool.OracleDataSource クラスから、プロパティおよび関連する getter メソッドと setter メソッドも継承します。これによって、databaseName、dataSourceName、description、networkProtocol、portNumber、serverName および driverType の各プロパティの getter メソッドと setter メソッドが提供されます。これらのプロパティおよびその getter メソッドと setter メソッドの詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

注意： JSP ページで使用する JavaBean と同様に、ConnCacheBean プロパティは、setter メソッドを直接使用しないで、jsp:setProperty 操作で設定できます。

次のメソッドを使用して、接続をオープンおよびクローズします。

- `Connection getConnection()`

  ConnCacheBean プロパティの設定を使用して、接続のキャッシュから接続を取得します。

- `void close()`

  接続とオープン中のカーソルをすべてクローズします。

ConnCacheBean クラスは、Oracle JDBC のバッチ更新および行のプリフェッチを直接サポートしていませんが、`getConnection()` メソッドで取得する `Connection` オブジェクトの `setDefaultExecuteBatch(int)` メソッドおよび `setDefaultRowPrefetch(int)` メソッドをコールすることで、これらの機能を有効にできます。`Connection` オブジェクトから作成した JDBC の文オブジェクトの `setExecuteBatch(int)` メソッドおよび `setRowPrefetch(int)` メソッドを使用することもできます。（バッチ更新は、プリコンパイルされた SQL 文でのみサポートされます。）これらの機能の詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

> **注意：** ConnBean とは異なり、ConnCacheBean を使用する場合は、標準の `Connection` オブジェクトの機能を使用して、文オブジェクトを作成および実行します。

問合せ専用 DBBean

問合せのみを実行するには、`oracle.jsp.dbutil.DBBean` を使用します。

> **注意：**
>
> - DBBean 自体にも接続機能がありますが、データ・ソースはサポートしません。データ・ソースが必要な場合は、ConnBean を使用します。
> - その他の DML 操作（`UPDATE`、`INSERT`、`DELETE` またはストアド・プロシージャ・コール）については、CursorBean を使用します。

DBBean には、次のプロパティがあります。

- user: データベース・スキーマのユーザー ID
- password: ユーザー・パスワード
- URL: データベース接続文字列

DBBean には、次のようなプロパティの setter メソッドと getter メソッドが用意されています。

- `void setUser(String)`
- `String getUser()`
- `void setPassword(String)`
- `String getPassword()`
- `void setURL(String)`
- `String getURL()`

> **注意：** JSP ページで使用する JavaBean と同様に、DBBean プロパティは、setter メソッドを直接使用しないで、`jsp:setProperty` 文で設定できます。

次のメソッドを使用して、接続をオープンおよびクローズします。

- `void connect()`

  DBBean プロパティの設定を使用して、データベース接続を確立します。

- `void close()`

  接続とオープン中のカーソルをクローズします。

次のいずれかのメソッドを使用して、問合せを実行します。

- `String getResultAsHTMLTable(String)`

  SELECT 文が含まれた文字列を入力します。このメソッドは、HTML 表として結果セットを出力するために必要な HTML で文字列を戻します。SQL 列名（または別名）は、表の列ヘッダーに使用されます。

- `String getResultAsXMLString(String)`

  SELECT 文が含まれた文字列を入力します。このメソッドは、XML タグとして SQL 列名（または別名）を使用して、結果セットを XML 文字列として戻します。

DML とストアド・プロシージャ用 CursorBean

単純な接続での SELECT、UPDATE、INSERT または DELETE の各操作またはストアド・プロシージャ・コールには、oracle.jsp.dbutil.CursorBean を使用します。接続には、事前定義の ConnBean オブジェクトが使用されます。

SQL 操作は、ConnBean オブジェクトの getCursorBean() コールに指定するか、または後述のように CursorBean オブジェクトの create()、execute() または executeQuery() メソッドのいずれかをコールすることによって指定できます。

CursorBean は、スクロール可能で更新可能なカーソル、バッチ更新、行のプリフェッチおよび問合せタイムアウト制限をサポートしています。Oracle JDBC のこれらの機能の詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

注意： 接続キャッシングを使用するには、ConnCacheBean と標準の Connection オブジェクトの機能を使用します。CursorBean は使用しないでください。

CursorBean には、次のプロパティがあります。

- executeBatch: Oracle JDBC のバッチ更新用バッチ・サイズ

  このプロパティを設定すると、Oracle JDBC のバッチ更新を有効にできます。

- preFetch: Oracle JDBC の行のプリフェッチでプリフェッチする文の数

  このプロパティを設定すると、Oracle JDBC の行のプリフェッチを有効にできます。

- queryTimeout: タイムアウトの発行まで、ドライバが文の実行を待機する秒数

- resultSetType: 結果セットのスクロール可能性

  これは、次のいずれかの int 型定数によって示されます。

  - TYPE_FORWARD_ONLY（デフォルト）: （next() メソッドを使用して）前方にしかスクロールできず、位置を設定できない結果セットに対して使用します。
  - TYPE_SCROLL_INSENSITIVE: 前後方向へのスクロールおよび位置の設定ができ、基礎となるデータの変更には対応していない結果セットに対して使用します。
  - TYPE_SCROLL_SENSITIVE: 前後方向へのスクロールおよび位置の設定ができ、基礎となるデータの変更に対応している結果セットに対して使用します。

- resultSetConcurrency: 結果セットの更新可能性

  これは、次のいずれかの int 型定数によって示されます。

  - CONCUR_READ_ONLY（デフォルト）: 読取り専用（更新不可）の結果セットに対して使用します。
  - CONCUR_UPDATABLE: 更新可能な結果セットに対して使用します。

必要に応じて、これらのプロパティを次のメソッドで設定し、Oracle JDBC の機能を有効にできます。

- void setExecuteBatch(int)
- int getExecuteBatch()
- void setPreFetch(int)
- int getPreFetch()
- void setQueryTimeout(int)
- int getQueryTimeout()
- void setResultSetConcurrency(int)

  CursorBean.CONCUR_READ_ONLY または CursorBean.CONCUR_UPDATABLE を指定します。

- int getResultSetConcurrency()

  CursorBean.CONCUR_READ_ONLY または CursorBean.CONCUR_UPDATABLE を戻します。

- void setResultSetType(int)

  CursorBean.TYPE_FORWARD_ONLY、CursorBean.TYPE_SCROLL_INSENSITIVE または CursorBean.TYPE_SCROLL_SENSITIVE を指定します。

- int getResultSetType()

  CursorBean.TYPE_FORWARD_ONLY、CursorBean.TYPE_SCROLL_INSENSITIVE または CursorBean.TYPE_SCROLL_SENSITIVE を戻します。

注意： JSP ページで使用する JavaBean と同様に、CursorBean プロパティは、setter メソッドを直接使用しないで、jsp:setProperty 操作で設定できます。

CursorBean インスタンスを jsp:useBean 文で定義した後に問合せを実行するには、CursorBean メソッドを使用し、次の 2 つのいずれかの方法でカーソルを作成できます。カーソルを作成し、個別のステップで接続を指定するには、次のメソッドを使用します。

- void create()
- void setConnBean(ConnBean)

または、次のメソッドを使用してプロセスを単一のステップに組み込むこともできます。

- void create(ConnBean)

ConnBean オブジェクトの設定は、4-3 ページの「データベース接続用 ConnBean」を参照してください。

次のメソッドを使用して、問合せを指定および実行します（JDBC の単純な Statement オブジェクトがバックグラウンドで使用されます）。

- ResultSet executeQuery(String)

  SELECT 文が含まれた文字列を入力します。

結果セットを HTML 表または XML 文字列としてフォーマットする場合は、executeQuery() のかわりに、次のいずれかのメソッドを使用します。

- String getResultAsHTMLTable(String)

  結果セットに HTML 表を作成するために、HTML 文の文字列を戻します。SELECT 文が含まれた文字列を指定します。

- String getResultAsXMLString(String)

  結果セットのデータを XML 文字列で戻します。SELECT 文が含まれた文字列を指定します。

CursorBean インスタンスを jsp:useBean 操作で定義した後に UPDATE 文、INSERT 文または DELETE 文を実行するには、CursorBean のメソッドを使用し、次の 2 つのいずれかの方法でカーソルを作成できます。次のメソッドを使用して、カーソルを作成（文のタイプを整数で、SQL 文を文字列で指定）し、接続を指定します。

- void create(int, String)
- void setConnBean(ConnBean)

または、次のメソッドを使用してプロセスを単一のステップに組み込むこともできます。

- void create(ConnBean, int, String)

ConnBean オブジェクトの設定は、4-3 ページの「データベース接続用 ConnBean」を参照してください。

int 型の入力には、Statement オブジェクト用の CursorBean.PLAIN_STMT、PreparedStatement オブジェクト用の CursorBean.PREP_STMT または CallableStatement オブジェクト用の CursorBean.CALL_STMT のいずれかの定数を使用して、JDBC 文のタイプを指定します。String への入力は、SQL 文を指定します。

次のメソッドを使用して、INSERT 文、UPDATE 文または DELETE 文を実行します。boolean 戻り値は無視してもかまいません。

- boolean execute()

また、バッチ更新の場合は、次のメソッドを使用して影響を受けた行数を戻します。

- int executeUpdate()

注意： SQL 操作は、文の作成時または実行時のいずれか（両方ではなく）に指定してください。execute() メソッドおよび executeUpdate() メソッドには、SQL 操作を指定するための文字列を必要に応じて指定できます。この指定は、ConnBean の getCursorBean() メソッドおよび create() メソッドについても同様です。

さらに、CursorBean は、コール可能文に対する registerOutParameter()、プリコンパイルされた SQL 文とコール可能文に対する set*XXX*()、および結果セットとコール可能文に対する get*XXX*() など、Oracle JDBC の機能をサポートします。

次のメソッドを使用して、データベース・カーソルをクローズします。

- void close()

例 : データ・ソースを利用した ConnBean と CursorBean の使用

次のサンプルは、ConnBean とデータ・ソースを使用して接続をオープンし、CursorBean を使用して問合せを実行する JSP ページを示しています。

```
<%@ page import="java.sql.*, oracle.jsp.dbutil.*" %>
<jsp:useBean id="cbean" class="oracle.jsp.dbutil.ConnBean" scope="session">
   <jsp:setProperty name="cbean" property="dataSource"
                    value="<%=request.getParameter("datasource")%>"/>
</jsp:useBean>
<% try {
      cbean.connect();
      String sql="SELECT ename, sal FROM scott.emp ORDER BY ename";
      CursorBean cb = cbean.getCursorBean (CursorBean.PREP_STMT, sql);
      out.println(cb.getResultAsHTMLTable());
      cb.close();
      cbean.close();
    } catch (SQLException e) {
      out.println("<P>" + "There was an error doing the query:");
      out.println("<PRE>" + e + "</PRE>¥n<P>"); }
%>
```

データ・アクセス用 SQL タグ

OC4J には、データベースにアクセスする SQL コマンドを実行するために、JSP ページで使用できる一連のタグがあります。次の各項では各タグについて説明します。

- データ・アクセス・タグの概要
- データ・アクセス・タグの説明

注意 : OC4J が提供する SQL カスタム・タグ・ライブラリは、JavaServer Pages 標準タグ・ライブラリ（JSTL）より前に開発され、JSTL と重複する機能が含まれています。標準に準拠するために、原則として、JSTL の使用をお薦めします。1-18 ページの「JavaServer Pages 標準タグ・ライブラリのサポート」を参照してください。

ただし、既存のライブラリもサポート対象です。JSTL で使用できないカスタム・ライブラリの機能のうち、有効と判断される機能は、必要に応じて、JSTL 標準に採用される予定です。

データ・アクセス・タグの概要

OC4J には、SQL 機能に関するカスタム・タグ・ライブラリが用意されています。次のタグで構成されます。

- dbOpen: データベース接続をオープンします。このタグは、データ・ソースおよび接続プーリングもサポートします。関連情報は、4-3 ページの「データ・ソースと接続プーリング用のデータ・アクセスのサポート」を参照してください。
- dbClose: データベース接続をクローズします。
- dbQuery: 問合せを実行します。
- dbCloseQuery: 問合せのカーソルをクローズします。
- dbNextRow: 結果セットの行を処理します。
- dbExecute: SQL 文（DML または DDL）を実行します。
- dbSetParam: パラメータを設定して、dbQuery タグまたは dbExecute タグにバインドします。
- dbSetCookie: Cookie を設定します。

SQL タグを使用する場合は、次の要件に注意してください。

- JDK 1.4 では `ojdbc14.jar` など、適切な JDBC ドライバ・ファイルがインストール済で、クラスパスに存在している必要があります。
- `ojsputil.jar` ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J のインストール時に予約済のタグ・ライブラリ・ディレクトリにインストールされます。
- タグ・ライブラリ・ディスクリプタ・ファイル `sqltaglib.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`sqltaglib.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/sqltaglib.tld
  ```

タグ・ライブラリ・ディスクリプタ・ファイル、`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリおよび `uri` 値の内容などを含む JSP タグ・ライブラリの使用方法に関する一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意 :

- データ・アクセス・タグは、4-2 ページの「データ・アクセスに関する JavaBeans」で説明されている Bean を使用します。一般的に、これらの Bean とタグは、適切な JDBC ドライバ・クラスがあることを前提として、Oracle 以外のデータベースで使用できます。ただし、後述されている機能の多くは、注意にあるように Oracle 固有の機能です。
- データ・アクセス・タグを使用するアプリケーションでは、`dbSetParam` タグを使用して、SQL 文のテキストの全文ではなくパラメータ値のみを指定することを考慮してください。これによって、ユーザーが必要な値以外の追加の SQL を入力する SQL ポイゾニングの可能性を回避できます。

データ・アクセス・タグの説明

次の各項では、データ・アクセス・タグの詳細な構文について説明し、データ・ソースでの `dbOpen` タグと `dbQuery` タグの使用例を紹介します。

注意：

- このタグ構文では、接頭辞「sql:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

SQL dbOpen タグ

dbOpen タグを使用し、dbQuery や dbExecute などのタグを使用した後続の SQL 操作のために、データベース接続をオープンします。オープンするには、データ・ソースの場所を指定（この場合、接続キャッシュがサポートされます）するか、ユーザー、パスワードおよび URL を個別に指定します。OC4J でデータ・ソースを設定する方法の詳細は、4-3 ページの「データ・ソースと接続プーリング用のデータ・アクセスのサポート」を参照してください。

実装には、oracle.jsp.dbutil.ConnBean インスタンスを使用します。単純な接続で、接続のキャッシュを使用しない場合は、必要に応じて、stmtCacheSize、preFetch および batchSize などの ConnBean のプロパティを設定し、Oracle JDBC の機能を有効にできます。詳細は、4-3 ページの「データベース接続用 ConnBean」を参照してください。

接続するための ConnBean オブジェクトは、dbOpen タグのタグ補足情報クラスのインスタンスに作成されます。標準の JSP タグ・ライブラリのフレームワークとタグ補足情報クラスの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

構文

```
<sql:dbOpen
   [ connId = "connection_id" ]
   [ scope = "page" | "request" | "scope" | "application" ]
   [ dataSource = "JNDI_name" ]
   [ user = "username"
     password = "password"
     URL = "databaseURL" ]
   [ commitOnClose = "true" | "false" ] >

   ...

</sql:dbOpen>
```

この接続を使用して実行するネストされたコードは、dbOpen の開始タグと終了タグ間のタグ・ボディに挿入できます。

注意： dataSource 属性を設定するか、user、password および URL の各属性を設定する必要があります。必要に応じて、データ・ソースを使用して URL を指定し、次に dbOpen タグの user 属性と password 属性を個別に使用できます。

データ・ソースを接続のキャッシュに使用すると、最初のキャッシュの使用で、データ・ソースが初期化されます。dbOpen タグの user 属性と password 属性を使用してユーザーとパスワードを指定すると、そのユーザーとパスワードに関するキャッシュが初期化されます。後続のキャッシュは、同じユーザーとパスワードに対して使用されます。

属性

- connId: 必要に応じてこの属性を使用し、接続の ID 名を指定します。この ID は、dbQuery や dbExecute など、後続のタグ内で参照できます。dbQuery タグおよび dbExecute タグを dbOpen タグ内にネストすることもできます。この接続 ID は、接続をクローズするときに dbClose タグ内で参照することもできます。

  接続 ID は、dbQuery タグまたは dbExecute タグを dbOpen タグ内にネストする場合にも指定できます。この場合、接続は接続 ID を使用して検出されます。scope 属性によって、同じ接続 ID を使用しながら異なるスコープを持つ接続を、複数設定できます。

  接続 ID を指定すると、dbClose タグで明示的にクローズするまで、接続はクローズされません。接続 ID がない場合、接続は、dbOpen 終了タグの検出時に自動的にクローズされます。

- scope（必ず connId とともに使用）: 接続インスタンスのスコープを指定します。デフォルトは、page スコープです。

  dbOpen タグにスコープの設定を指定した場合は、同じ接続 ID を使用している他のタグ（dbQuery、dbExecute または dbClose）にも同じスコープの設定を指定する必要があります。

- dataSource（user、password および URL の各属性を設定しない場合は必須）: 必要に応じてこの属性を使用し、データベース接続のためのデータ・ソースの JNDI 名を指定します。最初に、OC4J の data-sources.xml ファイルにデータ・ソースを設定します。（詳細は、4-3 ページの「データ・ソースと接続プーリング用のデータ・アクセスのサポート」を参照してください。）dataSource の設定は、data-sources.xml の <data-source> 要素にある location 名、ejb-location 名または pooled-location 名に対応している必要があります。

  データ・ソースには URL 設定の指定が必要ですが、ユーザーとパスワードのペアの指定は不要です。かわりに、dbOpen タグの user 属性と password 属性を使用できます。

  この属性は、OC4J 環境でのみサポートされます。

  > **注意：** 列挙データ・ソースの JNDI 参照では、ejb-location JNDI 名のみの使用をお薦めします。データ・ソースの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

- user（ユーザーとパスワードのペアがデータ・ソースを介して指定されていない場合は必須）: データベース接続用のユーザー名です。

  データ・ソースと user 属性の両方を介してユーザー名が指定されている場合は、user 属性が優先されます。データ・ソースが、異なるユーザー名を使用した既存の論理接続のあるプール接続である場合は、競合が発生する可能性がありますので、このような重複は避けてください。

- password（ユーザーとパスワードのペアがデータ・ソースを介して指定されていない場合は必須）: データベース接続用のユーザー・パスワードです。

  パスワードは JSP ページにハードコードしないでください。セキュリティ上問題となります。かわりに、次のようにして request オブジェクトからパスワードおよび他のパラメータを取得できます。

  ```
  <sql:dbOpen connId="conn1" user='<%=request.getParameter("user")%>'
              password='<%=request.getParameter("password")%>' URL="url" />
  ```

  user 属性と同様に、パスワードをデータ・ソースと password 属性の両方に指定した場合は、password 属性が優先されます。

- URL（データ・ソースが指定されていない場合は必須）: データベース接続用の URL です。データ・ソースを介して URL が提供されている場合、dbOpen タグの URL 属性は無視されます。

- commitOnClose: 「true」に設定すると、接続がクローズまたはスコープ外になったときに自動的に SQL commit が実行されます。デフォルトの「false」に設定すると、自動的に SQL rollback が実行されます。

  便宜上、アプリケーション全体で自動 commit または自動 rollback の動作を指定する場合は、次のように、アプリケーションの web.xml ファイルにパラメータ名 commit-on-close を設定します。

```
<context-param>
   <param-name>commit-on-close</param-name>
   <param-value>true</param-value>
</context-param>
```

  dbOpen タグの commitOnClose 設定は、web.xml ファイルの commit-on-close 設定よりも優先されます。

> **注意：** 以前のリリースでは、この動作は、接続のクローズ時に、常に自動的にコミットされます。commitOnClose 属性には、移行を簡素化するための下位互換性があります。

SQL dbClose タグ

dbClose タグを使用して、dbOpen タグに指定したオプションの connId パラメータに関連付けられている接続をクローズします。dbOpen タグで connId が使用されていない場合、接続は、dbOpen 終了タグに達したときに自動的にクローズされます。この場合、dbClose タグは不要です。

構文

```
<sql:dbClose connId = "connection_id"
           [ scope = "page" | "request" | "scope" | "application" ] />
```

属性

- connId（必須）: クローズされる接続の ID で、接続をオープンした dbOpen タグに指定した ID です。
- scope: 接続インスタンスのスコープです。デフォルトは「page」です。ただし、dbOpen タグに page 以外のスコープを指定した場合は、dbClose タグにも同じスコープを指定する必要があります。

SQL dbQuery タグ

dbQuery タグを使用して問合せを実行し、JDBC 結果セット、HTML 表、XML 文字列または XML DOM オブジェクトのいずれかで、結果を出力します。dbQuery 開始タグと終了タグの間のタグ・ボディに、SELECT 文（1 文のみ）を指定します。

このタグは、カーソルに oracle.jsp.dbutil.CursorBean オブジェクトを使用するため、必要に応じて、結果セットのタイプ、結果セットの並行性、バッチ・サイズおよびプリフェッチ・サイズなどのプロパティを設定できます。CursorBean 機能の詳細は、4-8 ページの「DML とストアド・プロシージャ用 CursorBean」を参照してください。

XML で使用する場合、このタグは XML プロデューサとして動作します。詳細は、5-2 ページの「XML プロデューサと XML コンシューマ」を参照してください。5-7 ページの「transform タグと dbQuery タグの使用例」も参照してください。

構文

```
<sql:dbQuery
          [ queryId = "query_id" ]
          [ connId = "connection_id" ]
          [ scope = "page" | "request" | "scope" | "application" ]
          [ output = "HTML" | "XML" | "JDBC" ]
          [ maxRows = "number" ]
          [ skipRows = "number" ]
          [ bindParams = "value" ]
          [ toXMLObjName = "objectname" ] >

   ...SELECT statement (one only)...

 </sql:dbQuery>
```

重要:

- SELECT 文をセミコロンで終了しないでください。現時点では構文エラーになります。
- dbQuery タグは、現時点では LOB 列をサポートしません。

属性

- queryId: カーソルの ID 名を指定します。dbNextRow タグを使用して結果を処理する場合、この属性は必須です。

  queryId パラメータを指定すると、dbCloseQuery タグで明示的にクローズするまで、カーソルはクローズされません。問合せ ID がない場合、カーソルは、dbQuery 終了タグの検出時に自動的にクローズされます。この属性はリクエスト時属性ではないため、JSP 式の値は指定できません。

- connId: データベース接続用の ID で、接続をオープンした dbOpen タグの connId 設定に従います。dbQuery タグに connId を指定しない場合は、そのタグを dbOpen タグのボディにネストし、dbOpen タグでオープンした接続を使用する必要があります。この属性は、リクエスト時属性ではありません。

- scope: 接続インスタンスのスコープです。デフォルトは「page」です。ただし、関連する dbOpen タグに page 以外のスコープを指定した場合は、dbQuery タグにも同じスコープを指定する必要があります。この属性は、リクエスト時属性ではありません。

- output: 任意の出力フォーマットです。次のいずれかを指定します。

  - HTML を指定すると、結果セットは HTML 表で出力されます（デフォルト）。
  - XML を指定すると、結果セットは XML 文字列または XML DOM オブジェクト（オブジェクト名が toXMLObjName 属性に指定されている場合）で出力されます。
  - JDBC を指定すると、結果セットは JDBC ResultSet オブジェクトで出力されます。このオブジェクトは、行間を反復する dbNextRow タグを使用して処理できます。

- maxRows: 表示データの最大行数です。デフォルトはすべての行です。

- skipRows: 問合せ結果を表示する前に、結果内でスキップするデータ行の数です。デフォルトは 0 です。

- bindParams: パラメータを問合せにバインドします。次は、従業員番号の入力プロンプトをユーザーに表示するアプリケーションの例です。ここでは、指定された値を問合せの empno フィールドにバインドする bindParams を使用しています。

  ```
  <sql:dbQuery connId="con1" bindParams="empno">
       select * from EMP where empno=?
  </sql:dbQuery>
  ```

dbSetParam タグで設定したパラメータ値を bindParams 属性を介してバインドすることもできます。4-19 ページの「SQL dbSetParam タグ」を参照してください。

- toXMLObjName: 結果を XML DOM オブジェクトで出力する場合に、XML オブジェクト名を指定します。これを使用するには、output も「XML」に設定する必要があります。

SQL dbCloseQuery タグ

dbCloseQuery タグを使用して、dbQuery タグに指定したオプションの queryId パラメータに関連付けられているカーソルをクローズします。dbQuery タグで queryId が指定されていない場合、カーソルは、dbQuery 終了タグに達したときに自動的にクローズされます。この場合、dbCloseQuery タグは不要です。

構文

```
<sql:dbCloseQuery queryId = "query_id" />
```

属性

- queryId（必須）: クローズするカーソルの ID で、カーソルをオープンした dbQuery タグに指定した ID です。

SQL dbNextRow タグ

dbNextRow タグを使用して、dbQuery タグで取得し、指定した queryId に関連付けられている結果セットの各行を処理します。dbNextRow 開始タグと終了タグの間のタグ・ボディに、処理コードを指定します。このボディは、結果セットの各行に対して実行されます。

dbNextRow タグを使用するには、dbQuery タグで、output を「JDBC」に設定し、queryId を、参照する dbNextRow タグに指定する必要があります。

結果セット・オブジェクトは、dbQuery タグのタグ補足情報クラスのインスタンスに作成されます。標準の JSP タグ・ライブラリのフレームワークとタグ補足情報クラスの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

構文

```
<sql:dbNextRow queryId = "query_id" >
...Row processing...
</sql:dbNextRow >
```

属性

- queryId（必須）: 処理する結果が含まれているカーソルの ID で、カーソルをオープンした dbQuery タグに指定した ID です。

例 次の例では、dbOpen タグ、dbQuery タグおよび dbNextRow タグを組み合せて使用しています。

```
<sql:dbOpen connId="con1" URL="jdbc:oracle:thin:@myhost:1521/myservice"
            user="scott" password="tiger">
</sql:dbOpen>
<sql:dbQuery connId="con1" output="jdbc" queryId="myquery">
             select * from EMP
</sql:dbQuery>
<sql:dbNextRow queryId="myquery">
             <%= myquery.getString(1) %>
</sql:dbNextRow>
<sql:dbCloseQuery queryId="myquery" />
<sql:dbClose connId="con1" />
```

SQL dbExecute タグ

dbExecute タグを使用して、単一の DML 文または DDL 文を実行します。dbExecute 開始タグと終了タグの間のタグ・ボディに、文を指定します。

このタグでは、カーソルに oracle.jsp.dbutil.CursorBean オブジェクトが使用されます。CursorBean 機能の詳細は、4-8 ページの「DML とストアド・プロシージャ用 CursorBean」を参照してください。

構文

```
<sql:dbExecute
          [ connId = "connection_id" ]
          [ scope = "page" | "request" | "scope" | "application" ]
          [ output = "yes" | "no" ]
          [ bindParams = "value" ] >

   ...DML or DDL statement (one only)...

</sql:dbExecute >
```

重要 :

- DML 文や DDL 文をセミコロンで終了しないでください。現時点では構文エラーになります。
- dbExecute タグは、現時点では LOB 列をサポートしません。

属性

- connId: データベース接続の ID で、接続をオープンした dbOpen タグの connId 設定に従います。dbExecute タグに connId を指定しない場合は、そのタグを dbOpen タグのボディにネストし、dbOpen タグでオープンした接続を使用する必要があります。
- scope: 接続インスタンスのスコープです。デフォルトは「page」です。ただし、dbOpen タグに page 以外のスコープを指定した場合は、dbExecute タグにも同じスコープを指定する必要があります。
- output: output="yes" のとき、DML 文の場合は、HTML 文字列「*number* 行影響を受けました。」がブラウザに出力され、操作によって影響を受けたデータベースの行数がユーザーに通知されます。DDL 文の場合は、文の実行ステータスが出力されます。デフォルトは「no」です。
- bindParams: パラメータを SQL 文にバインドします。次は、従業員番号の入力プロンプトをユーザーに表示するアプリケーションの例です。ここでは、指定された値を DELETE 文の empno フィールドにバインドする bindParams を使用しています。

  ```
  <sql:dbExecute connId="con1" bindParams="empno">
       delete from EMP where empno=?
  </sql:dbExecute>
  ```

  dbSetParam タグで設定したパラメータ値を bindParams 属性を介してバインドすることもできます。次項の「SQL dbSetParam タグ」を参照してください。

SQL dbSetParam タグ

このタグを使用すると、問合せにバインド（dbQuery タグを使用）またはその他の SQL 操作にバインド（dbExecute タグを使用）するパラメータ値を設定できます。

> **注意：** データ・アクセス・タグを使用するアプリケーションでは、dbSetParam タグを使用して、SQL 文のテキストの全文ではなくパラメータ値のみを指定することを考慮してください。これによって、ユーザーが必要な値以外の追加の SQL コードを入力する SQL ポイゾニングの可能性を回避できます。

構文

```
<sql:dbSetParam name = "param_name"
                value = "param_value"
              [ scope = "page" | "request" | "scope" | "application" ] />
```

属性

- name（必須）: 設定するパラメータの名前です。
- value（必須）: パラメータの値です。
- scope: バインド・パラメータのスコープです。デフォルトは、page スコープです。

例 次の例では、dbSetParam タグを使用して、id2 という名前のパラメータの値を設定します。この値は、dbExecute タグで SQL 文にバインドされます。

```
<sql:dbSetParam name="id2" value='<%=request.getParameter("id")%>'
                scope="session" />
Result:
   <HR>
   <sql:dbOpen dataSource="<%= dataSrcStr %>" >
        <sql:dbExecute output="yes" bindParams="id2 name job sal">
             insert into emp(empno, ename, deptno, job, sal)
                  values (?, ?, 20, ?, ?)
        </sql:dbExecute>
    </sql:dbOpen>
    <HR>
```

SQL dbSetCookie タグ

このタグを使用すると、Cookie を設定できます。dbSetCookie タグは、標準の javax.servlet.http.Cookie クラスの機能をラップします。

構文

```
<sql:dbSetCookie name = "cookie_name"
               [ value = "cookie_value" ]
               [ domain = "domain_name" ]
               [ comment = "comment" ]
               [ maxAge = "age" ]
               [ version = "protocol_version" ]
               [ secure = "true" | "false" ]
               [ path = "path" ] />
```

属性

- name（必須）: Cookie の名前です。
- value: Cookie の値です。Cookie では null 値が許可されるため、この属性は必須ではありません。
- domain: Cookie のドメイン名です。ドメイン名のフォームは、RFC 2019 の仕様に従います。
- comment: Cookie の目的を説明するコメントです。
- maxAge: Cookie の許容最大時間（秒単位）です。「-1」に設定すると、Cookie はブラウザが停止するまで保持されます。
- version: Cookie が準拠する HTTP プロトコルのバージョンです。
- secure: Cookie が HTTPS などの保護プロトコルを使用して送信されるかどうかをブラウザに通知します。
- path: Cookie のファイル・システム・パス（クライアントが Cookie を戻す場所）を指定します。

例

```
<sql:dbSetCookie name="cId" value='<%=request.getParameter("id")%>'
                 maxAge='800000' />
```

例 : データ・ソースを利用した dbOpen と dbQuery の使用

次のサンプルは、データ・ソースで dbOpen タグを使用して接続をオープンしてから、dbQuery タグを使用して問合せを実行する JSP ページを示しています。

```
<%@ taglib uri="
http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/
sqltaglib.tld" prefix="sql" %>
<HTML>
<BODY>
   <sql:dbOpen dataSource='<%=request.getParameter("datasource") %>'
               connId="con1">
   </sql:dbOpen>
   <sql:dbQuery connId="con1">
      SELECT * FROM emp ORDER BY ename
   </sql:dbQuery>
   <sql:dbClose connId="con1" />
</BODY>
</HTML>
```

5

XML と XSL に関するタグのサポート

この章では、OC4J が提供する、XML データおよび XSL 変換に使用できるタグについて説明します。また、他の OC4J タグでの XML の追加機能のサマリーを示します。これらのタグは、JSP 仕様に従って実装されます。

この章には、次の項目が含まれます。

注意： JSP ページでの XML 関連機能の追加情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

XML をサポートする Oracle タグの概要

次の各項では、OC4J が提供する XML 機能を備えたタグの概要を説明します。これらのタグは、XML DOM オブジェクトを入力として取得できるタグ、XML DOM オブジェクトを出力として生成できるタグ、指定したスタイルシートに基づいて XML 文書を変換するタグ、およびデータを入力ストリームから XML DOM オブジェクトに解析するタグです。

注意： OC4J が提供する XML カスタム・タグ・ライブラリは、JavaServer Pages 標準タグ・ライブラリ（JSTL）より前に開発され、JSTL と重複する機能が含まれています。標準に準拠するために、原則として、JSTL の使用をお薦めします。1-18 ページの「JavaServer Pages 標準タグ・ライブラリのサポート」を参照してください。

ただし、既存のライブラリもサポート対象です。JSTL で使用できないカスタム・ライブラリの機能のうち、有効と判断される機能は、必要に応じて、JSTL 標準に採用される予定です。

XML プロデューサと XML コンシューマ

XML 関連操作は、次のいずれか（または両方）に分類できます。

- XML プロデューサ。XML オブジェクトを出力します。
- XML コンシューマ。XML オブジェクトを入力として取得します。

同様に、XML 関連タグも、XML プロデューサまたは XML コンシューマ（あるいはその両方）に分類できます。XML プロデューサは、XML オブジェクトを XML コンシューマに明示的にまたは暗黙的に渡すことができます。暗黙的に渡すことを「匿名の受渡し」と呼びます。

XML 関連タグ間での明示的な受渡しの場合、プロデューサ・タグでは `toXMLObjName` 属性が、コンシューマ・タグでは `fromXMLObjName` 属性が使用されます。この受渡し処理は、バックグラウンドで、JSP の標準 `pageContext` オブジェクトの `getAttribute()` メソッドと `setAttribute()` メソッドを使用して実行されます。次は、明示的な受渡しの使用例です。

```
<sql:dbQuery output="XML" toXMLObjName="foo" ... >
   ...SQL query...
</sql:dbQuery>
...
<ojsp:cacheXMLObj fromXMLObjName="foo" ... />
```

XML 関連タグ間での暗黙的な受渡しの場合、`toXMLObjName` 属性と `fromXMLObjName` 属性は使用しないでください。この受渡し処理は、タグ・ハンドラ間での直接的な相互作用によって実行されます。この場合、通常タグはネストされています。次は、暗黙的な受渡しの使用例です。

```
<ojsp:cacheXMLObj ... >
   <sql:dbQuery output="XML" >
      ...SQL query...
   </sql:dbQuery>
</ojsp:cacheXMLObj>
```

この例では、`dbQuery` タグで作成された XML は、`pageContext` オブジェクトに格納されず、`cacheXMLObj` タグに直接渡されます。

暗黙的な受渡しでは、タグがコンシューマとして機能できるように、タグ・ハンドラは、OC4J `ImplicitXMLObjConsumer` インタフェースを次のように実装します。

```
interface ImplicitXMLObjConsumer
{
   void setImplicitFromXMLObj();
}
```

XML 機能を持つ OC4J タグのサマリー

OC4J が提供するタグ・ライブラリに関して、表 5-1 に、XML プロデューサまたはコンシューマとして機能できるタグのサマリーを示します。

表 5-1 XML 機能を持つ OC4J タグ

タグ	ライブラリ	プロデューサ / コンシューマ	関連属性	タグ情報
transform / styleSheet	XML	両方	fromXMLObjName toXMLObjName	5-4 ページの「XML/XSL データ変換用の XML transform タグと styleSheet タグ」
parsexml	XML	プロデューサ	toXMLObjName	5-5 ページの「入力ストリームからの変換を行う XML parsexml タグ」
cacheXMLObj	Web Object Cache	両方	fromXMLObjName toXMLObjName	7-19 ページの「Web Object Cache の cacheXMLObj タグ」
dbQuery	SQL	プロデューサ	toXMLObjName	4-15 ページの「SQL dbQuery タグ」
invoke	Web サービス	プロデューサ	toXMLObjName	11-12 ページの「Web サービスの invoke タグ」

注意：

- XML `transform` タグと `styleSheet` タグは同じ機能を持ち、同一の結果を生成します。
- `cacheXMLObj` タグを XML タグ・ライブラリ・ディスクリプタ・ファイル（`xml.tld`）と Web Object Cache タグ・ライブラリ・ディスクリプタ・ファイル（`jwcache.tld`）の両方に定義すると便利です。

XML ユーティリティ・タグ

次の各項では、OC4J が提供する XML ユーティリティ・タグについて説明します。

- XML ユーティリティ・タグの説明
- XML ユーティリティ・タグの例

XML ユーティリティ・タグ・ライブラリを使用する場合は、次の要件に注意してください。

- XML タグ・ライブラリを使用するには、`ojsputil.jar`、ファイル `xmlparserv2.jar` ファイルおよび `xsu12.jar` ファイルがインストール済で、クラスパスに存在している必要があります。これらのファイルは、OC4J に含まれています。`ojsputil.jar` ファイルは、予約済のタグ・ライブラリ・ディレクトリにあります。
- タグ・ライブラリ・ディスクリプタ・ファイル `xml.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`xml.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「xml:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を `taglib` ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

XML ユーティリティ・タグの説明

次の各項では XML ユーティリティ・タグについて説明します。

- XML/XSL データ変換用の XML transform タグと styleSheet タグ
- 入力ストリームからの変換を行う XML parsexml タグ

重要： タグの属性はリクエスト時属性です。つまり、特に注記がないかぎり、タグ属性は JSP 式を入力として取得できます。

XML/XSL データ変換用の XML transform タグと styleSheet タグ

XML と XSL を動的 JSP ページに頻繁に使用する場合は、結果をクライアントに戻す前に、サーバーで XSL 変換を行う必要があります。この処理を簡素化するために、Oracle では、XML ライブラリに 2 つの同義のタグを用意しています。結果を HTTP クライアントに直接出力することも、指定した XML DOM オブジェクトに出力することもできます。この項の説明と例のように、`transform` タグまたは `styleSheet` タグのいずれかを使用します。この 2 つのタグによって得られる効果は同じです。

それぞれのタグは、XML プロデューサと XML コンシューマの両方の役割を果たします。次のいずれかを入力として取得できます。

- XML DOM オブジェクト。
- XML コードを生成する JSP コマンドと静的テキストを含むタグ・ボディ。

これらのタグは、次のいずれか、または両方に出力できます。ただし、いずれの場合も適用するスタイルシートを指定する必要があります。

- XML DOM オブジェクト。
- ブラウザへの出力 Writer。この場合は、指定したスタイルシートが適用されます。

入力にタグ・ボディを使用すると、タグは開始タグから終了タグまでにあるすべてのものに適用されます。1 つのページ内に複数の XSL 変換ブロックを設定できます。この場合、各ブロックはそのブロック自体の `transform` タグまたは `styleSheet` タグでバインドされ、適切なスタイルシートへの独自の `href` ポインタを指定します。

構文

```
<xml:transform href="xslRef"
               [ fromXMLObjName = "objectname" ]
               [ toXMLObjName = "objectname" ]
               [ toWriter = "true" | "false" ] >

   [...body...]

</xml:transform >
```

または

```
<xml:styleSheet href="xslRef"
               [ fromXMLObjName = "objectname" ]
               [ toXMLObjName = "objectname" ]
```

```
            [ toWriter = "true" | "false" ] >

   [...body...]

</xml:styleSheet >
```

属性

- href（必須）: XML データ変換に使用する XSL スタイルシートを指定します。この属性は、XML オブジェクト（書式設定なしで変換できる）への出力またはブラウザへの出力に関係なく、必須です。

  href 属性については、次の点に注意してください。

  - この属性は、静的 XSL のスタイルシートまたは動的に生成されたスタイルシートのいずれにも適用できます。たとえば、スタイルシートを生成する JSP ページまたはサーブレットに適用できます。
  - この属性は、完全修飾の URL（http://*host*:*port*/*path*）、アプリケーション相対 URL の JSP 参照（「/」で開始する）、およびページ相対 URL の JSP 参照（「/」で開始しない）の場合があります。アプリケーション相対 URL パスとページ相対 URL パスの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。
  - この属性の値は、静的な Java 文字列リテラルの場合や、標準 JSP のリクエスト時の式によって動的に指定された値の場合があります。

- fromXMLObjName: タグ・ボディからではなく、DOM オブジェクトから入力する場合に、入力 XML DOM オブジェクトを指定します。タグ・ボディと fromXMLObjName 指定の両方がある場合は、fromXMLObjName が優先されます。
- toXMLObjName: HTTP クライアントへの出力用の JSP Writer オブジェクトのかわりに（またはこのオブジェクトに加えて）、DOM オブジェクトを出力先とする場合に、出力 XML DOM オブジェクトの名前を指定します。この属性は、暗黙的 XML コンシューマが存在する場合は不要です。たとえば、タグ内に transform タグまたは styleSheet タグがネストされている場合です。
- toWriter: この属性は「true」または「false」のいずれかで、出力先が、HTTP クライアントへの出力用の JSP Writer オブジェクトであるかどうかを示します。これは、DOM オブジェクトへの出力のかわりに（または追加して）指定できます。デフォルトでは、下位互換性を維持するため、「true」に設定されます。（Oracle9*i*AS リリース 2 より前のリリースでは、この属性が唯一の出力の選択肢で、toXMLObjName 属性はありませんでした。）

入力ストリームからの変換を行う XML parsexml タグ

XML タグ・ライブラリには、XML プロデューサ・ユーティリティ・タグ parsexml が含まれています。このタグは、入力ストリームから XML DOM オブジェクトに変換します。このタグによって、指定したリソースまたはタグ・ボディから入力を取得できます。

構文

```
<xml:parsexml
            [ resource = "xmlresource" ]
            [ toXMLObjName = "objectname" ]
            [ validateResource = "dtd_path" ]
            [ root = "dtd_root_element" ] >

   [...body...]

</xml:parsexml >
```

属性

- resource: タグ・ボディからではなく XML リソースから入力する場合に、XML リソースを指定します。たとえば、次のように指定します。

  ```
  resource="/dir1/hello.xml"
  ```

  タグ・ボディと指定したリソースの両方がある場合は、リソースが優先されます。
- toXMLObjName: 出力先の XML DOM オブジェクトの名前を指定します。この属性は、暗黙的 XML コンシューマが存在する場合は不要です。たとえば、タグ内に parsexml タグがネストされている場合です。
- validateResource: XML 妥当性チェックに対して、適切な DTD へのパスを指定できます。あるいは、DTD を XML リソースに埋め込むことができます。この属性は、リクエスト時属性ではありません。
- root: DTD のルート要素を指定して妥当性チェックを行います。この属性は、リクエスト時属性ではありません。root 要素を指定せずに validateResource を指定した場合、デフォルトのルートは、DTD のトップレベルになります。

XML ユーティリティ・タグの例

次の各項では XML ユーティリティ・タグの使用例を紹介します。

- transform タグの使用例
- transform タグと dbQuery タグの使用例
- transform タグと parsexml タグの使用例

transform タグの使用例

この項では、XSL スタイルシートのサンプルと、そのスタイルシートで transform タグを使用して出力をフィルタ処理する JSP ページのサンプルを示します。これは非常に単純化した例です。ページ内の XML は静的なものです。より現実的な例では、JSP ページを使用して、変換の実行前に XML のすべてまたは一部を動的に生成します。

サンプル・スタイルシート : hello.xsl

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

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

  <xsl:template match="page">
   <html>
    <head>
     <title>
      <xsl:value-of select="title"/>
     </title>
    </head>
    <body bgcolor="#ffffff">
     <xsl:apply-templates/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="title">
   <h1 align="center">
    <xsl:apply-templates/>
   </h1>
  </xsl:template>

  <xsl:template match="paragraph">
   <p align="center">
    <i>
```

```
    <xsl:apply-templates/>
   </i>
  </p>
 </xsl:template>

</xsl:stylesheet>
```

サンプル JSP ページ : hello.jsp

```
<%@ page session = "false" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld"
           prefix="xml" %>

<xml:transform href="style/hello.xsl" >

<page>
 <title>Hello</title>
 <content>
  <paragraph>This is my first XML/XSL file!</paragraph>
 </content>
</page>

</xml:transform>
```

この例では、図 5-1 で示すような出力が生成されます。

図 5-1 サンプル XSL 出力

transform タグと dbQuery タグの使用例

この例では、dbQuery タグから結果セットを戻し、transform タグを使用して、XSL スタイルシート rowset.xsl で問合せ結果をフィルタ処理します（コードを次に示します）。また、dbOpen タグを使用して接続をオープンします。接続文字列は、request オブジェクトまたは useDataSource.jsp ページのいずれかから取得されます（コードを次に示します）。dbOpen タグから transform タグへのデータの受渡しは暗黙的に実行されます。関連情報は、4-15 ページの「SQL dbQuery タグ」および 4-13 ページの「SQL dbOpen タグ」を参照してください。

JSP ページ

```
<%@ page import="oracle.sql.*, oracle.jdbc.driver.*, oracle.jdbc.*, java.sql.*" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld"
           prefix="xml" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/sqltaglib.tld"
           prefix="sql" %>

<%
   String dataSrcStr=request.getParameter("dataSrcStr");
   if (dataSrcStr==null) {
     dataSrcStr=(String)session.getValue("dataSrcStr");
   } else {
     session.putValue("dataSrcStr",dataSrcStr);
   }
   if (dataSrcStr==null) { %>
<jsp:forward page="../../sql/useDataSource.jsp" />
<%
   }

%>
<h3>Transform DBQuery Tag Example</h3>
<xml:transform href="style/rowset.xsl" >
    <sql:dbOpen connId="conn1" dataSource="<%= dataSrcStr %>" />
    <sql:dbQuery connId="conn1" output="xml" queryId="myquery" >
       select ENAME, EMPNO from EMP order by ename
    </sql:dbQuery>
    <sql:dbCloseQuery queryId="myquery" />
    <sql:dbClose connId="conn1" />
</xml:transform>
```

> **注意：** この例の dbOpen タグの場合は、データ・ソースで URL、ユーザー名およびパスワードが指定されているものとします。

rowset.xsl

```
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="ROWSET">
 <html><body>
 <h1>A Simple XML/XSL Transformation</h1>
 <table border="2">
<xsl:for-each select="ROW">
  <tr>
    <td><xsl:value-of select="@num"/></td>
    <td><xsl:value-of select="ENAME"/></td>
    <td><xsl:value-of select="EMPNO"/></td>
  </tr>
</xsl:for-each>
</table>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

useDataSource.jsp

```
<body bgcolor="#FFFFFF">
<font size=+0>
<B>Please enter a suitable JDBC connection string, before you try the above demo</B>
<pre>
     To use a data source that you have set up in data-sources.xml, enter the
     data source string below. Once you have set the data source string it
     will remain in effect until the session times out.
</pre>
<%
   String dataSrcStr;
   dataSrcStr=request.getParameter("dataSrcStr");
   if (dataSrcStr==null) {
     dataSrcStr=(String)session.getValue("dataSrcStr");
   }
   if (dataSrcStr==null) {
     dataSrcStr="jdbc/OracleCoreDS";  // default data source string
   }

   session.putValue("dataSrcStr",dataSrcStr);
%>
<FORM METHOD=get ACTION="<%= request.getParameter("nextaction") %>" >
<INPUT TYPE="text" NAME="dataSrcStr" SIZE=40 value="<%=dataSrcStr%>" >
<INPUT TYPE="submit" VALUE="Change Data Source String" >
</FORM>
</font>
```

transform タグと parsexml タグの使用例

この項では、XSL スタイルシートの email.xsl を使用して、parsexml タグから出力を取得し、その出力を transform タグによってフィルタ処理する例を 2 つ示します。いずれの場合も、データは、指定したリソースの XML ファイルから parsexml タグ・ハンドラによって収集された後、toxml1 XML オブジェクトを介して parsexml タグから transform タグに明示的に渡されます。

最初の例では、XML リソースの email.xml と DTD の email.dtd を使用します。root 属性は指定されていません。したがって、妥当性チェックは、トップレベル要素の <email> から行われます。

第 2 の例では、XML リソース emailWithDtd.xml を使用します。このファイルには、DTD が埋め込まれています。root 属性によって、妥当性チェックが要素 <email> から行われることが明示的に指定されています。

email.xml、email.dtd、emailWithDtd.xml および email.xsl の各ファイルも次に示します。

transform と parsexml の使用例 1

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld"
           prefix="xml" %>
<h3>XML Parsing Tag Email Example</h3>
<xml:transform fromXMLObjName="toxml1" href="style/email.xsl">
   <xml:parsexml resource="style/email.xml" validateResource="style/email.dtd"
                 toXMLObjName="toxml1">
   </xml:parsexml>
</xml:transform>
```

transform と parsexml の使用例 2

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld"
           prefix="xml" %>
<h3>XML Parsing Tag Email Example</h3>
<xml:transform fromXMLObjName="toxml1" href="style/email.xsl">
   <xml:parsexml resource="style/emailWithDtd.xml" root="email"
                 toXMLObjName="toxml1">
   </xml:parsexml>
</xml:transform>
```

email.xml

```
<email>
<recipient>Manager</recipient>
<copyto>jsp_dev</copyto>
<subject>XML Bug fixed</subject>
<bugno>BUG 1109876!</bugno>
<body>for reuse tag and checked in the latest version!</body>
<sender>Developer</sender>
</email>
```

email.dtd

```
<!ELEMENT email (recipient,copyto,subject,bugno,body,sender)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copyto (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT bugno (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
```

emailWithDtd.xml

```
<!DOCTYPE email [
<!ELEMENT email (recipient,copyto,subject,bugno,body,sender)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copyto (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT bugno (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT sender (#PCDATA)>]>
<email>
<recipient>Manager</recipient>
<copyto>jsp_dev</copyto>
<subject>XML Bug fixed</subject>
<bugno>BUG 1109876!</bugno>
<body>for reuse tag and checked in the latest version!</body>
<sender>Developer</sender>
</email>
```

email.xsl

```
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="email">
 <html><body>
  To: <xsl:value-of select="recipient"/>
  CC: <xsl:value-of select="copyto"/>
  Subject:   <xsl:value-of select="subject"/> ...
  <xsl:value-of select="body"/> !!
  Thanks  <xsl:value-of select="sender"/>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

6

Edge Side Includes 用の JESI タグ

この章では、OC4J が提供する Edge Side Includes for Java（JESI）タグ・ライブラリについて説明します。これらのタグは、Oracle Application Server Web Cache で使用可能な Edge Side Includes（ESI）フレームワークの上で稼働して、JSP アプリケーションに ESI キャッシング機能を提供します。

この章には、次の項目が含まれます。

- Edge Side Includes テクノロジと処理の概要
- JESI 機能の概要
- Oracle JESI タグの説明
- JESI タグの処理と JESI と ESI 間での変換

OracleAS Web Cache、Oracle Application Server Java Object Cache および OC4J Web Object Cache の説明を含む Web キャッシングの概要は、1-13 ページの「Web アプリケーションに対する Oracle キャッシング・サポートのサマリー」を参照してください。

> **注意：** JESI 仕様はまだ確定されていません。最新の作業バージョンに準拠するように多大な労力が傾注されていますが、OC4J 10.1.2 の実装が JESI 仕様の最終バージョンに全面的に準拠するという保証はありません。

Edge Side Includes テクノロジと処理の概要

JESI タグは、JSP ページの動的コンテンツをキャッシュ可能なコンポーネントに分割します。その基礎となるのが Edge Side Includes のアーキテクチャとマークアップ言語です。

JESI タグの使用は、特定の ESI プロセッサやキャッシング・システムに依存しているわけではありませんが、通常、Oracle ユーザーは、OracleAS Web Cache とその ESI プロセッサを使用しています。

次の各項では、Oracle JESI タグの基礎となっている基本テクノロジの一部に関するバックグラウンド情報について説明します。

- Edge Side Includes テクノロジ
- Oracle Application Server Web Cache と ESI プロセッサ

この項で説明するのは、ESI アーキテクチャと言語の簡単な概要のみです。ESI テクノロジに関する追加情報は、次の Web サイトを参照してください。

```
http://www.esi.org
```

Edge Side Includes テクノロジ

この項では、ESI テクノロジの特性と ESI のサロゲートの概念を紹介します。

ESI の概要

Edge Side Includes とは、XML スタイルのマークアップ言語のことです。この言語によって、Web サーバーから離れたネットワークの「edge（端点）」で動的コンテンツのアセンブリを実行できます。また、この言語は、Web キャッシュやコンテンツ配信ネットワーク（CDN）など、現在使用可能なツールを活用して、ユーザーのパフォーマンスを向上させるように設計されています。

ESI には、中間段階の処理を促進することによって、Web サーバーとアプリケーション・サーバーの負荷を軽減する手段が含まれています。サロゲートまたはリバース・プロキシと呼ばれるこの手段は、ESI 言語を理解し、Web サーバーにかわって動作します。ESI コンテンツの目的は、Web サーバーを離れてから、ユーザーのブラウザに表示されるまでの間の処理を行うことにあります。サロゲートは、HTTP ヘッダーを介してコマンド実行されます。したがって、サロゲートを ESI プロセッサと呼ぶことができます。また、Web キャッシュの機能の一部に含めることもできます。

ESI は、Web ページの各動的部分を個別にキャッシュし、適切に切り離して取得できるパーシャル・ページのキャッシング方法論に近いと言えます。

ESI マークアップ・タグを使用すると、開発者は集計 Web ページを定義できます。また、HTTP クライアントで表示するために、必要に応じて、ESI プロセッサで取得およびアセンブルするキャッシュ可能コンポーネントを定義できます。集計ページは、ユーザーが指定した URL に関連付けられているリソースですが、これはアセンブリ用のコンテナと考えることができます。これには、ESI タグを使用して指定された取得用とアセンブリ用の命令が含まれます。

重要： JESI タグを使用しているページでは、ESI タグを直接使用しないでください。

サロゲートについて

サロゲートは、ページ・コンテンツを所有する Web サーバーのかわりに動作するため、コンテンツ所有者は、サロゲートの動作を完全に制御できます。これにより、サロゲートを使用しない場合に比較して、大幅なパフォーマンスの向上が可能になります。

サロゲートでのキャッシング処理は、HTTP でのキャッシング処理に類似しています。どちらも、類似した鮮明な妥当性チェック機能を基盤として使用しています。ただし、サロゲートには、その上に制御機能も備わっています。

ESI の主要機能

ESI 言語のバージョン 1.0 には、次の主要な機能が含まれています。

- インクルード

  ESI プロセッサは、ネットワークから取得された動的コンテンツのフラグメントを、集計ページにアセンブルして、ユーザーに出力します。各フラグメントには、独自のメタデータがあり、そのフラグメントのキャッシング動作を制御できます。次の図 6-1 を参照してください。

- 変数のサポート

  ESI は、HTTP のリクエスト属性に基づく変数の使用をサポートしています。ESI 文では、処理時に変数を使用することも、その変数を処理済のマークアップに直接出力することもできます。

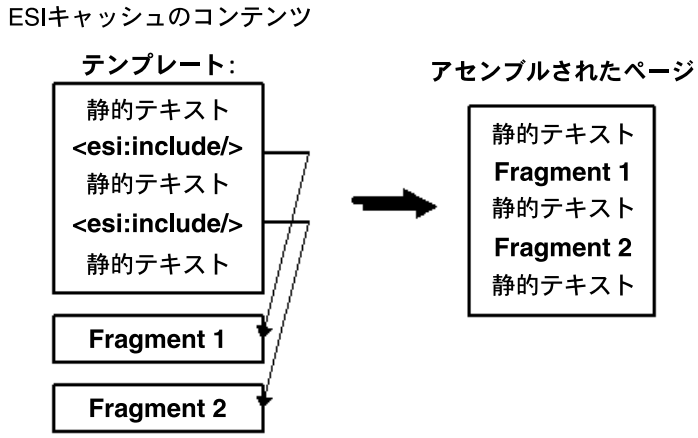
- 条件付き処理

  ESI では、ページの処理方法の判断に、条件付きロジックのブール比較を使用できます。

- エラー処理と代替処理

  一部の ESI タグは、デフォルト・リソースまたは代替リソース（あるいはその両方）の指定をサポートしています。代替リソースには、プライマリ・リソースが検出できなかった場合の代替 Web ページなどがあります。

図 6-1 ESI include モデル



Oracle Application Server Web Cache と ESI プロセッサ

この項では、OracleAS Web Cache とその ESI プロセッサを紹介します。詳細は、『Oracle Application Server Web Cache 管理者ガイド』を参照してください。

Oracle Application Server Web Cache の概要

Oracle では、Web サイトのパフォーマンスの問題を抱える E-Business を支援するために、OracleAS Web Cache を提供しています。この製品は、コンテンツ対応のサーバー・アクセラレータ、つまりリバース・プロキシ・サーバーであり、Oracle Application Server 上で稼働する Web サイトのパフォーマンス、スケーラビリティおよび可用性を向上させます。

OracleAS Web Cache では、頻繁にアクセスされる URL のページをメモリーに格納することによって、Web アプリケーション・サーバー上でこれらの URL に対するリクエストを繰り返して処理する必要がなくなります。静的ドキュメントのみを処理するレガシー・プロキシ・サーバーとは異なり、OracleAS Web Cache は、1 つ以上の Web アプリケーション・サーバーからの静的コンテンツと動的に生成されたコンテンツの両方をキャッシュします。キャッシュ・ヒットの数が増えるため、レガシー・プロキシよりもパフォーマンスが強化され、アプリケーション・サーバーへの負荷が減少します。

概念的には、OracleAS Web Cache は、Web アプリケーション・サーバーの手前に位置しており、Web サーバーのコンテンツをキャッシュし、そのコンテンツをリクエストしている Web ブラウザに送信します。Web サイトへのアクセス時に、Web ブラウザは、HTTP プロトコルまたは HTTPS プロトコルのリクエストを OracleAS Web Cache に送信します。すると、OracleAS Web Cache が Web アプリケーション・サーバーに対する仮想サーバーとして動作します。リクエストされたコンテンツがすでに期限切れの場合、無効になった場合またはアクセスできない場合、OracleAS Web Cache は、その Web アプリケーション・サーバーから新しいコンテンツを取得します。

Oracle Application Server Web Cache の使用ステップ

次に、ブラウザと OracleAS Web Cache との典型的な対話のステップを示します。

1. ブラウザが、企業の Web サイトにリクエストを送信します。
2. このリクエストの結果、ドメイン・ネーム・システム（DNS）に対して、Web サイトの IP アドレスのリクエストが生成されます。
3. DNS は、OracleAS Web Cache の IP アドレスを戻します。
4. ブラウザは、Web ページのリクエストを OracleAS Web Cache に送信します。
5. リクエストされたコンテンツがキャッシュにある場合、OracleAS Web Cache は、そのコンテンツをブラウザに直接送信します。これは、キャッシュ・ヒットと呼ばれます。
6. リクエストされたコンテンツが OracleAS Web Cache に存在しない場合、またはそのコンテンツが古くなったか無効になっている場合、Web Cache は、そのリクエストを Web アプリケーション・サーバーに渡します。これは、キャッシュ・ミスと呼ばれます。
7. Web アプリケーション・サーバーは、OracleAS Web Cache を介してコンテンツを送信します。
8. OracleAS Web Cache は、そのコンテンツをクライアントに送信し、そのページのコピーをキャッシュに作成します。

注意： キャッシュに格納されたページは、無効になったか古くなった後、ある時点で削除されます。

Oracle Application Server Web Cache の ESI プロセッサ

OracleAS Web Cache には、キャッシングにおける Edge Side Includes マークアップ言語の使用をサポートする ESI プロセッサが組み込まれています。（詳細は、6-2 ページの「Edge Side Includes テクノロジ」を参照してください。）

OracleAS Web Cache 環境の Web 開発者は、アプリケーションで ESI 言語を直接使用できます。ただし、JSP 開発者の場合は、様々な理由から、ESI 言語に対する便利な JSP インタフェースとして提供されている JESI タグ・ライブラリを使用します。次項の「JESI タグのメリット」を参照してください。

JESI 機能の概要

次の各項では、JESI 機能と Oracle による実装について説明します。

次の Web サイトから JESI の提案仕様にアクセスできます。

```
http://www.esi.org
```

JESI タグのメリット

OC4J では、JESI タグ・ライブラリを、ESI タグと Web キャッシングに関する Edge Side Includes 機能への便利なインタフェースとして提供しています。開発者は、Web アプリケーションで ESI タグを直接使用できますが、JSP ページでは JESI タグを使用したほうが便利です。次に、ESI タグを直接使用せずに、JESI タグを使用する主なメリットについて説明します。

- 標準 JSP フレームワークと便利な機能
  - JESI タグによって、JSP プログラミングの使い慣れた便利な機能を使用できます。たとえば、完全な URL やファイル・パスではなく、ページ相対位置やアプリケーション相対位置によってインクルード・ページを参照できます。
  - JESI タグの属性に動的な値を渡すことができます。
  - JESI タグを他の JSP タグ・ライブラリからのタグと併用できます。
- JESI ショート・カット構文

  JESI タグは、メタデータ情報（キャッシュ内のページの期限切れなど）の指定、必要に応じたページの明示的な無効化、および Cookie 情報を使用したページのパーソナライズなどに使用する便利な構文とタグ属性をサポートしています。
- アプリケーション・レベルの構成ファイル

  JESI タグ・ライブラリでは、アプリケーション・レベルの構成ファイルを使用して、特定の環境に適したデプロイ時パラメータとアプリケーションのデフォルト設定を簡単に指定できます。このようにして、多様なニーズを持つ様々な環境にデプロイし、アプリケーション・コードを変更せずに、適切なデフォルトを設定できます。たとえば、このような構成ファイルを使用して、無効化リクエストに対して、キャッシュ・サーバーの URL、ユーザー名およびパスワードを事前に設定できます。

Oracle による JESI タグ実装の概要

Oracle が JESI を実装するレイヤーは、標準 ESI フレームワークの最上部です。Java Community Process（JCP）組織の支援による（OC4J 10.1.2 実装時点で）保留中の JESI 標準 JSR-128 にも準拠しています。JCP 組織と JSR-128 の状況の詳細は、次の URL を参照してください。

```
http://www.jcp.org
```

JESI タグ・ライブラリは、標準実装であるため、次の点に注意してください。

- JESI タグを使用するアプリケーションは、OC4J JSP コンテナに依存しません。任意の標準 JSP コンテナに移植可能です。（OC4J 以外では、JSR-128 とともに提供されるリファレンス実装、または該当する場合は使用中の JSP コンテナとともに提供される JESI 実装を使用できます。）
- このマニュアルでは、特に OracleAS Web Cache とその ESI プロセッサについて説明していますが、JESI タグ・ライブラリは、特定のキャッシング環境に依存せず、ESI 1.0 仕様に準拠したすべての ESI プロセッサで使用できます。

Oracle JESI タグ・ライブラリは、次のタグをサポートします。

- ページ・コンテンツの動的キャッシングに使用する JESI `control`、JESI `include`、JESI `param`、JESI `template`、JESI `fragment` および JESI `codeblock`
- 必要に応じて、キャッシュ内のオブジェクトの明示的な無効化に使用する JESI `invalidate`（およびサブタグ）
- Cookie を介してページのカスタマイズに使用する JESI `personalize`

JSP 開発者は、対応する ESI タグ（`esi:include` など）ではなく、これらのタグ（JESI `include` など）を使用します。このタグの有用性と利便性については、6-5 ページの「JESI タグのメリット」で説明しています。

注意： Oracle JESI タグ・ライブラリは、JSP カスタム・タグ・ライブラリの標準全般に従って実装されます。標準 JavaServer Pages タグ・ライブラリのフレームワークの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

JESI の使用モデル

JESI タグを使用して、集計ページとそのキャッシュ可能コンポーネントを定義する方法には、次の 2 つのモデルがあります。

- control/include モデル
- template/fragment モデル

この項では、これら 2 つのモデルについて説明し、最後に JESI `include` タグに関する特記事項を説明します。

control/include モデル

JESI タグを control/include 方式で使用する方法は、モジュール化された方法です。この方法では、通常、ほとんど（またはすべて）のキャッシュ可能コンテンツをインクルード・ページとして集計ページに表示します。この方法は、新規ページを開発する場合に特に便利です。このモデルは、次のように使用します。

- JESI `control` タグをトップレベルのページで使用し、必要に応じて、インクルード・コンテンツ以外のコンテンツにキャッシング・パラメータを設定します。
- JESI `include` タグを使用して、動的コンテンツをインクルードします。
- JESI `control` タグを各インクルード・ページ内で使用し、必要に応じて、これらのページにキャッシング・パラメータを設定します。

各インクルード・ファイルは、個別のキャッシュ可能オブジェクトです（ただし、タグ設定によってキャッシングを無効にできます）。また、トップレベルのページのコンテンツもすべて個別オブジェクトです。

状況によっては、両タグともオプションになります。ページには、JESI `control` タグを、JESI `include` タグなしで指定できます。実際に、これが既存のページを JESI 用に変換する簡単な方法です。また、JESI `include` タグを使用しているページに、必ずしも JESI `control` タグを指定する必要はありません。このタグ指定に関係なく、ESI プロセッサには JESI `include` タグの有無が適切に通知されます。また、インクルード・ページに JESI `control` タグを指定する必要はありません。

トップレベルのページまたはインクルード・ページがキャッシュできるかどうかは、次の要素によって決まります。

- JESI `control` タグがある場合、キャッシュ可能かどうかは属性設定またはデフォルトの属性値（使用可能な場合）によって決まります。
- JESI `control` タグがない場合、キャッシュ可能かどうかは、ESI プロセッサの設定によって決まります。
- トップレベルのページの JESI `control` タグは、インクルード・ページには影響を与えません。

タグ構文と例は、次の各項を参照してください。

- 6-13 ページの「JESI control タグ」
- 6-14 ページの「JESI include タグ」
- 6-16 ページの「JESI param タグ」
- 6-17 ページの「例 : control/include モデル」

template/fragment モデル

template/fragment 方式では、コンテンツは単一ページに含まれ、必要に応じてページを独立したキャッシュ可能なフラグメントに分割します。このモデルは、既存のページを JESI 用に変換し、特定の部分を独立したキャッシュ可能コンポーネントにする場合に特に便利です。このモデルは、次のように使用します。

- JESI `template` タグを使用して、すべての参照可能なコンテンツの集計を囲みます。このタグはフラグメント以外のコンテンツにキャッシング・パラメータを設定します。`template` タグ以外には、参照可能なコンテンツが存在しないようにする必要があります。
- `template` 開始タグと終了タグとの間に JESI `fragment` タグを必要に応じて使用して、集計内のフラグメントを個別にキャッシュするように定義します。
- JESI `include` タグも、テンプレート・レベルまたはフラグメント・レベルのいずれかで必要に応じて使用します。
- template タグ内で `codeblock` タグをフラグメントの外で使用して、特定のコード・ブロックの条件付き実行を指定します。

JESI `template` タグと JESI `fragment` タグは、必ず同時に使用します。ページに個別のフラグメントが不要な場合は、JESI `template` タグではなく、JESI `control` タグを使用します。

各フラグメントは個別のキャッシュ可能オブジェクトです。フラグメント以外のテンプレート・レベルのコンテンツもすべて個別のキャッシュ可能オブジェクトです。JESI `include` タグを介してインクルードされるページも、個別のキャッシュ可能オブジェクトです。キャッシュ可能かどうかは、次のように決定されます。

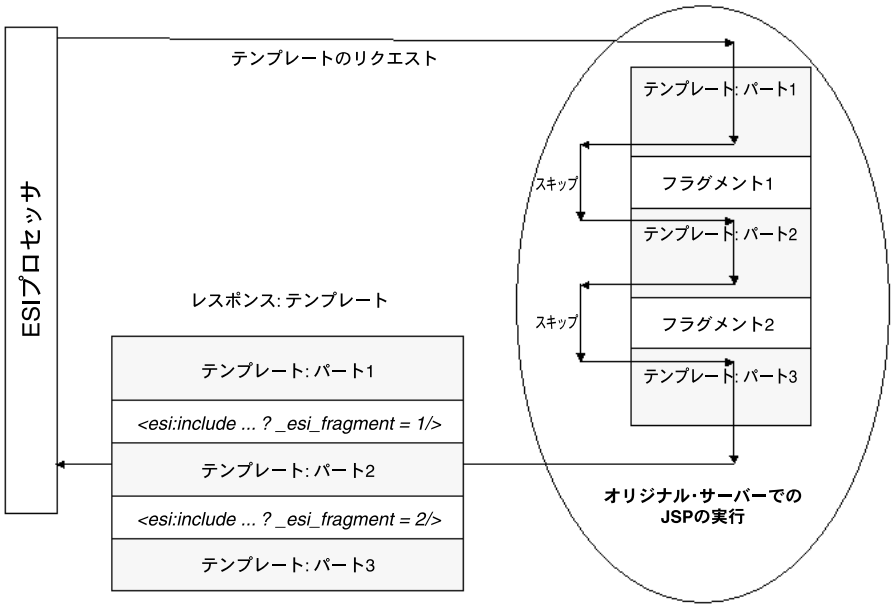
- テンプレート（フラグメント以外のコンテンツ）がキャッシュ可能かどうかは、JESI `template` タグ属性設定またはデフォルトの属性値（使用可能な場合）によって決まります。
- また、フラグメントがキャッシュ可能かどうかは、JESI `fragment` タグの属性設定またはデフォルトの属性値（使用可能な場合）によって決まります。
- インクルード・ページがキャッシュ可能かどうかは、次の要素によって決まります。
  - JESI `control` タグがある場合、キャッシュ可能かどうかは属性設定またはデフォルトの属性値（使用可能な場合）によって決まります。
  - JESI `control` タグがない場合、キャッシュ可能かどうかは、ESI プロセッサの設定によって決まります。

注意： template/fragment モデルは、処理済の JESI `control` タグを含むレスポンスに使用できます。たとえば、集計レスポンスの条件付き生成に任意の代替ページ・セットからのレスポンスをインクルードできる場合には、この方法が必須です。この場合、JSP コンテナは `control` タグの属性設定を使用し、`template` タグの属性設定を無視しますが、必要な場合には `fragment` タグを適切に囲む `template` タグが存在することに注意してください。template/fragment モデルでは常にこのように設定されるため、`template` タグ以外にキャッシュ可能コンテンツが存在しないようにする必要があります。

テンプレートとフラグメントは、独立したキャッシュ可能オブジェクトであるため、ESI プロセッサで期限切れになる時点が異なる場合があります。キャッシュ・ミスの発生時、または期限切れオブジェクトがリクエストされた時、ESI プロセッサは、オリジナル・サーバー（Oracle Application Server の場合は OC4J）に新しいコピーをリクエストします。

リクエストされたオブジェクトが JESI テンプレートの場合、JSP コンテナはフラグメント以外のページのコードを実行します。JSP トランスレータは、生成した出力に、フラグメントをインクルードする場所を指定する ESI マークアップを配置します。この時点では、JESI フラグメントに格納されているコードは実行されません。次の図に、これを示します。

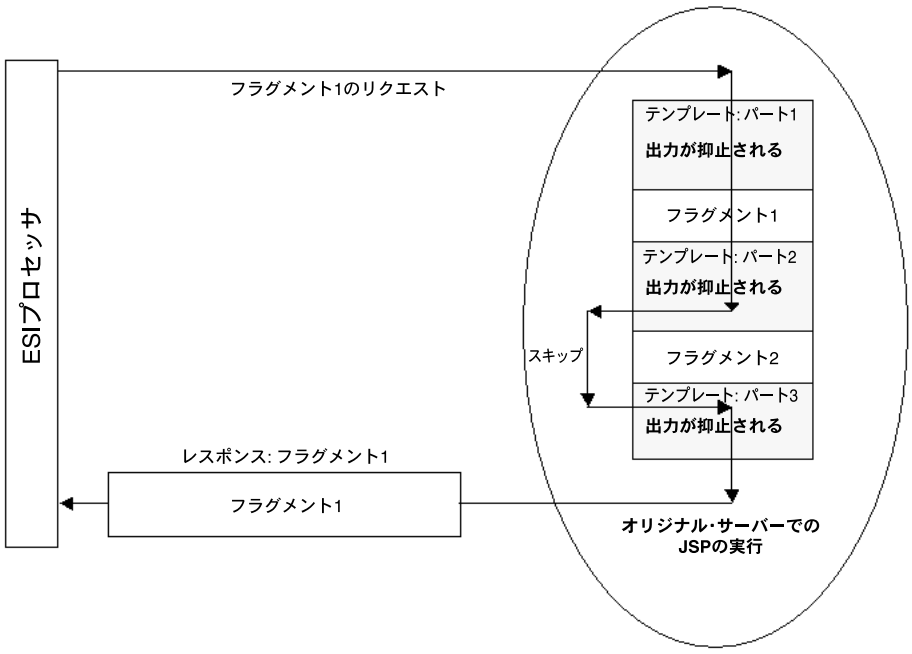
図 6-2 JESI テンプレート・リクエスト



フラグメントが期限切れになると、ESI プロセッサはオリジナル・サーバーに対して、その特定のフラグメントをリクエストします。フラグメントを実行するために、OC4J JSP コンテナは、テンプレート・コード（フラグメント以外のコード）に加えて、リクエスト対象のフラグメントのコードを実行します。テンプレート・コードの実行により、フラグメントは変数の宣言や初期化など、特定の副次効果に依存可能になります。

フラグメント・コードの出力はレスポンスで戻され、テンプレート・コードの出力は破棄されます。レスポンスを受信すると、ESI プロセッサはフラグメントの更新済コピーをキャッシュします。次の図に、これを示します。

図 6-3 JESI フラグメント・リクエスト



注意： コストのかかるテンプレート・コードの実行が不必要に反復されないように、コードは JESI codeblock タグ内の適切な位置に配置してください。各 codeblock タグは、中のコードを実行する必要のあるタイミング（テンプレートのリクエスト時、フラグメントのリクエスト時または常時）に従って構成します。

テンプレートとフラグメントのコード位置と期限切れのポリシーを選択する際には、この動作に注意してください。特に、テンプレート・コードは更新リクエストのたびに実行されるため、コストのかかるコードを配置する位置に注意します。コストのかかる計算は、毎回実行が必要な場合または codeblock タグ内に適切に配置されている場合を除いて、テンプレート・レベルに配置しないでください。それ以外の場合、コストのかかる計算は、できるだけ期限切れ期間が長いフラグメントに配置します。

図 6-4 に、codeblock タグの使用例を示します。この例では、コード・ブロックはフラグメントのリクエスト時にのみ実行されます。この図では、リクエストはテンプレートに関するものであるため、コード・ブロックは実行されません。

図 6-5 に、codeblock タグのもう 1 つの使用例を示します。この例でも、コード・ブロックはフラグメントのリクエスト時にのみ実行されます。ただし、この図では、リクエストはフラグメントに関するものであるため、コード・ブロックが実行されます。

図 6-4 テンプレート・リクエストによる JESI codeblock フラグメントの実行

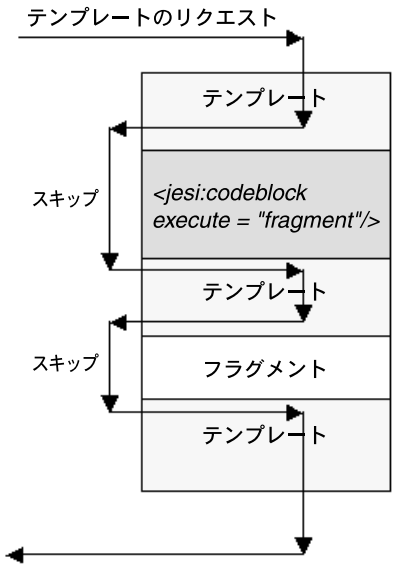
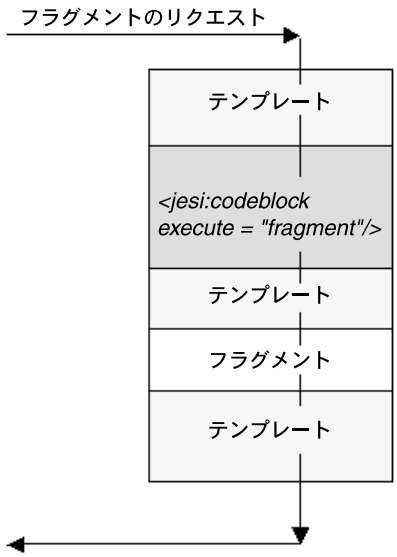


図 6-5 フラグメント・リクエストによる JESI codeblock フラグメントの実行



同一リクエスト時に、2 つのフラグメントは実行されないことにも注意してください。たとえば、スクリプトレット変数の値を 1 つのフラグメントで宣言または設定して、別のフラグメントで依存させることは避けてください。1 つの変数が 2 つ以上のフラグメントで必要な場合は、テンプレート・コード（可能であれば codeblock タグ内）で宣言および設定してください。同様に、1 つのフラグメント内にリクエスト属性またはセッション属性を設定し、それを別のフラグメントに読み取ることは避けてください。このような、グローバルなページ・ロジックは、テンプレート・レベルに配置してください。

最後に、ページの様々なフラグメントがリフレッシュされる時点は、無効化メッセージと期限切れの設定によって異なることに注意してください。通常、適切にチューニングされたアプリケーションでは、ほとんどのフラグメントは ESI キャッシュから提供され、再生成が必要になることはまれです。

注意： JESI タグが JESI ルールに準拠し、相互に適切にネストしていれば、JESI `template`、JESI `fragment` および JESI `include` タグ内で `jsp:include` タグを使用できます。たとえば、ページに JESI `template` タグを使用し、`template` タグ内で `jsp:include` タグを使用し、JESI `fragment` タグをインクルード・ページに使用できます。また、上位レベルに他の `template` タグがない場合や、レスポンス・バッファへの全出力が `template` タグ内にある場合は、インクルード・ページ内で JESI `template` タグを使用できます。

タグ構文と例は、次の各項を参照してください。

- 6-19 ページの「JESI template タグ」
- 6-20 ページの「JESI fragment タグ」
- 6-14 ページの「JESI include タグ」
- 6-21 ページの「JESI codeblock タグ」
- 6-22 ページの「例 : template/fragment モデル」

JESI と JSP Includes に関する注意

control/include モデルと template/fragment モデルのいずれを使用する場合も、JESI `include` 文については、次の点に注意してください。

- ネストされた JESI インクルードは、独自の JESI `include` 文を含むページをインクルードした JESI `include` 文、または JESI `fragment` 文によって定義されたフラグメント内の JESI `include` 文として、サポートされています。

  たとえば、2 番目の場合、ESI プロセッサは次のステップを実行します。

  1. 集計ページのコンテンツをリクエストします。
  2. キャッシュ内でフラグメントのコンテンツを検索するか（使用可能な場合）、またはリクエストします。
  3. キャッシュ内でインクルード・ページのコンテンツを検索するか（使用可能な場合）、またはリクエストします。

- JESI `include` タグと標準 `jsp:include` タグは概念的には似ていますが、キャッシング用に JSP ページを変換するときに、JESI `include` タグを `jsp:include` タグの代替として使用できない場合があります。ESI プロセッサでは、別々の HTTP リクエストを使用するため、HTTP のリクエストまたはレスポンス・オブジェクトを、あるページと JESI `include` タグでインクルードされたページとの間で渡すことができません。インクルード・ページのコードが集計ページのリクエストまたはレスポンス・オブジェクトにアクセスする必要がある場合は、JESI template/fragment モデルを使用して、JESI `include` タグではなく、JESI `fragment` タグ（集計ページの JESI `template` タグ内）にコードを配置することを検討する必要があります。

キャッシュ内のオブジェクトの無効化

データベース内の関連データへの変更など、外部の状況によっては、キャッシュ内のオブジェクトを明示的に無効化する必要があります。また、あるページの実行によって、別のページに対応しているキャッシュ内のオブジェクトのデータが無効化される場合もあります。

このため、JESI には、JESI `invalidate` タグと関連サブタグが用意されています。これらのタグを使用すると、次の適切な組合せに基づいてページを無効化できます。

- 完全な URI または URI 接頭辞
- Cookie の名前 / 値ペア（オプション）
- HTTP/1.1 のリクエスト・ヘッダーの名前 / 値ペア（オプション）

無効化メッセージは XML ベースのフォーマットで、無効化する対象の URL を指定します。このメッセージは、JESI `invalidate` タグの実行時に JSP コンテナで起動され、`POST` メソッドによって HTTP を介してキャッシュ・サーバーに転送されます。次に、キャッシュ・サーバーが応答し、無効化レスポンスが HTTP を介して返信されます。

タグ構文と例は、6-24 ページの「キャッシュ内のオブジェクトの無効化に使用するタグとサブタグの説明」を参照してください。

キャッシュ内のページのパーソナライズ

動的 Web ページには、個別のユーザーにあわせてカスタマイズされた情報が頻繁に表示されます。たとえば、「ようこそ」のページには、ユーザー名や特別な挨拶文、あるいはユーザーが所有している株式の現在の相場などが表示されます。

このようなカスタマイズされた出力の場合、Web ページは、JESI `personalize` タグによって提供される Cookie 情報に依存します。このタグは、Cookie 置換を実行する必要があることを ESI プロセッサに通知します。このタグがない場合は、Web ページを複数のユーザーが ESI レベルで共有できません。

タグ構文と例は、6-30 ページの「ページのパーソナライズに使用するタグの説明」を参照してください。

JESI フォールバックの実行

JESI タグを使用するページに使用可能な ESI プロセッサがない場合（OracleAS Web Cache がインストールされていないシステムや Web Cache またはその ESI プロセッサが障害を起こしているシステムなど）は、OC4J の JSP コンテナがそのページを適切にアセンブルします。実質的には、最も重要な機能を引き継いで提供し、ページを適切に実行します。キャッシングは発生せず、JESI タグ属性値のエラー・チェックも実行されません。

これらの状況では、JSP コンテナは特定の JESI タグを次のように処理します。

- JESI `control` タグを無視します。
- JESI `include` タグを `jsp:include` タグであるかのように実行し、関連する JESI `param` タグを `jsp:param` タグであるかのように実行します。JESI `include` タグ内でネストしているスクリプトレット・コードも実行されることに注意してください。

  注意： この場合、JESI `include` 機能とは異なり、インクルード・ページ用の独立したレスポンス・オブジェクトは存在しません。

- JESI `template` および `fragment` タグが適切にネストしているかどうかをチェックしますが、それ以外はこれらのタグを無視して、単一のリクエストですべてのタグ・ボディを実行します。
- JESI `codeblock` タグ内のコードを無条件で実行します。
- JESI `invalidation` タグとすべてのサブタグを無視します。
- JESI `personalize` タグについては、Cookie が存在していた場合は、レスポンス・ボディに Cookie 値を挿入します。以前に Cookie が存在せず、`personalize` タグにデフォルト値が指定されている場合、JSP コンテナはレスポンス・ボディにデフォルト値を挿入します。以前に Cookie が存在せず、デフォルト値が指定されていない場合、`personalize` タグは効果がありません。

Oracle JESI タグの説明

次の各項では、OC4J が提供する JESI タグの構文と属性を説明し、使用例を示します。

- 動的キャッシング用タグの説明
- キャッシュ内のオブジェクトの無効化に使用するタグとサブタグの説明
- ページのパーソナライズに使用するタグの説明

JESI タグ・ライブラリを使用する場合は、次の要件に注意してください。

- 標準 JavaServer Pages タグ・ライブラリの実装である Oracle JESI タグ・ライブラリは、OC4J が提供する `ojsputil.jar` ファイルに含まれています。このファイルは、予約済のタグ・ライブラリ・ディレクトリにあります。このファイルがインストール済で、クラスパスに存在していることを確認してください。
- タグ・ライブラリ・ディスクリプタ・ファイル `jesitaglib.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` ファイルに配置されます。`jesitaglib.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意 :

- このタグ構文では、接頭辞「jesi:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を `taglib` ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

動的キャッシング用タグの説明

次の各項では、動的キャッシング用の JESI タグの使用方法、その構文と属性を説明し、例を示します。

control/include モデルと template/fragment モデルの概要は、6-6 ページの「JESI の使用モデル」を参照してください。

JESI control タグ

JESI `control` タグは、control/include 使用モデルで JSP ページのキャッシュ動作を制御します。JESI `control` タグは、トップレベルのページまたは任意のインクルード・ページで使用できます。ただし、必須ではありません。control/include モデルでは、JESI `control` タグがないページがキャッシュ可能かどうかは、ESI プロセッサの設定によって決まります。（詳細は、6-6 ページの「JESI の使用モデル」を参照してください。）

JESI `control` タグによる操作では、HTTP レスポンス・ヘッダーが設定されるため、このタグはできるだけページの先頭付近で他の JESI タグやバッファ・フラッシュより前に置く必要があります。

次の点に注意してください。

- JESI `control` タグの属性はすべてオプションです。何も設定せずにタグを使用すると、デフォルトではレスポンスのキャッシュは 24 時間で期限切れになるように設定され、期限切れのオブジェクトは即時に削除されます。
- キャッシング動作を ESI プロセッサの構成によって決める必要がある場合は、問題のページに JESI `control` タグを使用しません。
- 集計ページ（JESI `include` タグを含むページ）の JESI `control` タグは、インクルード・ページには影響しません。必要に応じて、各インクルード・ページで JESI `control` タグを使用してください。
- JSP コンテナが単一レスポンスの生成中に複数の JESI `control` タグを検出すると、最初のタグのみが処理されます。残りのタグは無視されます。JESI `include` タグを介してインクルードされるページ（独自の JESI `control` タグを持つページなど）は、独立したレスポンスとなることに注意してください。
- JSP コンテナが JESI `template` タグを検出しており、同じレスポンスの生成中に JESI `control` タグを検出すると、`control` タグは無視されます。

- JESI `control` タグを含むページがリクエスト・パラメータに依存する場合は、リクエストの問合せ文字列に応じてページの異なるバージョンをキャッシュする必要があるかどうかを検討してください。異なるリクエスト・パラメータ値が多すぎるため、キャッシュ内のページのバージョンが多くなりすぎることが予想される場合は、そのページをまったくキャッシュしない選択もできます（つまり、`cache="no"` を設定します）。

構文

```
<jesi:control
            [ expiration = "value" ]
            [ maxRemovalDelay = "value" ]
            [ cache = "yes" | "no" | "no-remote" ]
            [ control = "uninterpreted_string" ] />
```

属性

- `expiration`: キャッシュ内のオブジェクトの存続期間を秒単位で指定します。デフォルトは 86400（24 時間）です。
- `maxRemovalDelay`: 期限が切れたキャッシュ内のオブジェクトを、ESI プロセッサで引き続き格納できる最大時間（秒単位）を指定します。デフォルトは 0（ゼロ）で、即時に削除されます。
- `cache`: タグに対応するレスポンスがキャッシュ可能かどうかを指定します。「`yes`」（デフォルト）に設定すると、キャッシュ可能になります。キャッシングを無効化するには、`cache` を「`no`」に設定します。また、リモート ESI プロセッサやコンテンツ配信ネットワーク上ではなく、最も近いキャッシュへのキャッシングのみを有効にするには、「`no-remote`」に設定します。

  ページをキャッシュ不可にするのは、たとえば、JESI `include` タグを `copyParam="yes"` に設定して使用している場合です。次の「JESI include タグ」を参照してください。
- `control`: この属性の値は、JESI `control` タグの処理中に作成された `Surrogate-Control` レスポンス・ヘッダーを変更せずに追加されます。OracleAS Web Cache の ESI プロセッサはこの属性を使用しませんが、アプリケーションに別の ESI プロセッサを使用して、ヘッダーで追加の独自情報を渡す必要がある場合に便利です。

注意：

- `control` 属性名を `control` タグ名と混同しないでください。
- `Surrogate-Control` ヘッダーについては、6-31 ページの「JESI タグの処理と JESI と ESI 間での変換」を参照してください。

JESI include タグ

JESI `include` タグは、標準の `jsp:include` タグと同様に、インクルード・ページの出力を集計ページの出力に動的に挿入します。そのために、このタグは ESI プロセッサにインクルード・ページの処理とアセンブルを指示します。各インクルード・ページは、独立したキャッシュ可能オブジェクト（設定によっては、キャッシュ不可オブジェクト）です。

このタグは、次の使用例にある control/include モデルまたは template/fragment モデルのいずれでも使用できます。

- JESI `control` タグなしで、または JESI `template` タグと `fragment` タグなしで独立して使用
- control/include モデルで JESI `control` タグの後に使用
- template/fragment モデルで、JESI `fragment` タグ内または JESI `template` タグ内（フラグメント以外）のいずれかで使用

（詳細は、6-6 ページの「JESI の使用モデル」を参照してください。）

また、JESI include をネストできます。この場合、JESI include タグを介してインクルードされるページに JESI include タグを使用する方法と、標準 jsp:include タグを介してインクルードされるページに JESI include タグを使用する方法があります。

インクルード・ページがキャッシュ可能かどうかは、次の要素によって決まります。

- JESI control タグがある場合、キャッシュ可能かどうかは属性設定またはデフォルトの属性値（使用可能な場合）によって決まります。
- JESI control タグがない場合、キャッシュ可能かどうかは、ESI プロセッサの設定によって決まります。

構文

```
<jesi:include page = "uri"
            [ alt = "alternate_uri" ]
            [ ignoreError = "true" | "false" ]
            [ flush = "true" | "false" ]
            [ copyParam = "true" | "false" ] >

...optional jesi:param tags, related scriptlets...

</jesi:include>
```

注意：

- 標準の jsp:include タグおよびオプションの jsp:param サブタグと同様に、JESI include タグ内でネストした JESI param タグを使用して、集計ページ（JESI include タグを含むページ）に送られる新規パラメータを指定できます。タグ構文については、6-16 ページの「JESI param タグ」を参照してください。また、JESI include タグのボディには、追加したパラメータの評価に使用するスクリプトレット・コードを挿入できます。ただし、通常、スクリプトレット・コードの出力と JESI include タグのボディからの出力は破棄されます。
- JESI include タグの動作が jsp:include タグとは異なる場合があります。これは、JESI include タグによってインクルード・ページに独立したリクエスト・オブジェクトとレスポンス・オブジェクトが使用されるためです。たとえば、集計ページでリクエスト属性が設定され、インクルード・ページではこの属性をリクエスト・オブジェクトから読み取る場合、JESI include タグは適していません。
- copyparam 形式の copyParam 属性は廃止予定ですが、下位互換性のために使用できます。copyparam から copyParam への変更は、JESI の提案仕様に準拠するために行われました。copyparam はいずれサポート外になると思われます。

属性

- page（必須）：ページ相対位置またはアプリケーション相対位置のいずれかを使用して、インクルード対象の JSP ページの URI を指定します。（ページ相対位置とアプリケーション相対位置の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。）完全な「http://...」または「https://...」の URL もサポートされています。

  URI では、必要に応じて、インクルード・ページに渡す問合せパラメータと値を追加指定できますが、それよりも JESI param サブタグを使用することをお薦めします。6-16 ページの「JESI param タグ」を参照してください。
- alt: page 属性に指定されたページが見つからない場合に、インクルード対象となる代替ページの URI を指定します。構文は、page 属性の構文と同じです。

- ignoreError: 「true」に設定すると、インクルード・ページ（page 属性および alt 属性で指定したページ）にアクセスできない場合でも、集計ページのインクルード処理を続行できます。デフォルトは「false」です。
- flush: この属性は無視されます。ただし、jsp:include 構文からの移行を容易にするためにサポートされています。
- copyParam: インクルード・ページがリクエスト・パラメータを使用する場合は、この属性を「true」に設定すると、集計ページの HTTP リクエスト文字列からインクルード・ページにパラメータとその値をコピーできます。デフォルト値は「false」です。

  リクエスト・パラメータがインクルード・ページに必要で、copyParam="true" の場合は、集計ページをキャッシュしない（JESI control、JESI template または JESI fragment のタグに cache="no" を設定）か、複数バージョンの集計ページをキャッシュするかのいずれかを、パラメータ設定に従って実行する必要があります。

  たとえば、次のような使用例は避けてください。

```
<jesi:control cache="yes"/>
...
<jesi:include page="arf.jsp" copyParam="true" />
```

  これは、この集計ページのコピーがキャッシュから提供され、この後続リクエストのパラメータが当初リクエストのパラメータと異なる場合、そのページがサーバー上で実行されないか、または新規パラメータが arf.jsp に適切にコピーされないためです。この結果、クライアントには不適切なパラメータから生成された arf.jsp が提供されることになります。

  ただし、この使用例は、次のような特定の状況では、問題となりません。

  - arf.jsp ページでリクエスト・パラメータを使用しない状況。
  - 適切なバージョンの集計ページと arf.jsp が、URL パラメータに基づいて ESI プロセッサでキャッシュされる状況。詳細は、『Oracle Application Server Web Cache 管理者ガイド』を参照してください。

JESI param タグ

JESI param タグは、JESI include タグのオプションのサブタグです。これらのタグは、標準の jsp:include および jsp:param タグの場合と同様に連動します。

1 つ以上の JESI param サブタグを使用して、JESI include タグのターゲット・ページに追加の問合せパラメータを渡すことができます。この方が、JESI include タグの page URI にパラメータを指定するよりも簡単です。両方のメカニズムを使用すると、param タグのパラメータは include タグの page URI のパラメータの後に追加されます。include タグの copyParam="yes" 設定を介して元のリクエストからコピーされたパラメータは、JESI param タグのパラメータの後に追加されます。

サンプルについては、6-19 ページの「例 5: param タグを使用した control/include」を参照してください。

注意： パラメータ名と値は、集計ページ（JESI include および param タグを含むページ）の生成時に評価されることに注意してください。その後、集計ページが ESI プロセッサでキャッシュされると、インクルード・ページに渡されたパラメータ名と値は、集計ページが再生成されるまでそのまま残ります。これは、copyParam="true" 設定を介してリクエストからコピーされるリクエスト・パラメータの処理に類似しています。

構文

```
<jesi:include page = "uri" ... >
   <jesi:param name="param_name"
         value="param_value" />
   ...
</jesi:include>
```

属性

- name（必須）：パラメータの名前を指定します。
- value（必須）：パラメータの値を指定します。

例 : control/include モデル

この項では、control/include モデルでの JESI タグの使用例を示します。

例 1: control/include 次の例では、デフォルトのキャッシュ設定を使用します。JESI control タグは不要です。JESI include タグは、代替ファイルを指定しません。したがって、「ファイルが見つかりません。」というエラーが発生すると処理は停止します。flush 属性は、使用できますが、無視されます。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<html>
<body>
<jesi:include page="stocks.jsp" flush="true" />
<p>
<hr>
<jesi:include page="/weather.jsp" flush="true" />
<p>
<hr>
<jesi:include page="../sales.jsp" flush="true" />
</body>
</html>
```

例 2: control/include この例では、JESI control タグを使用し、maxRemovalDelay および expiration に対して、デフォルト以外のキャッシュ設定を指定します。さらに、ページのキャッシングを明示的に有効にします。ただし、デフォルトですでに有効になっています。最初の JESI include タグは、ESI プロセッサが order.jsp を取得できない場合の代替ページを指定し、どのページも取得できない場合でも、処理を続行する必要があることを指定します。第 2 の JESI include タグは、代替ページを指定しません。したがって、ページを取得できない場合、処理は停止します。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>

<jesi:control maxRemovalDelay="1000" expiration="300" cache="yes"/>
<jesi:include page="order.jsp" alt="alt.jsp" ignoreError="true"/>
<jesi:include page="commit.jsp" />
```

例 3: control/include この例は、条件付きの出力を含む集計ページの例です。Cookie は、カスタマの ID を表します。Cookie が見つからない場合は、一般的な製品情報を含む一般的な「ようこそ」ページが表示されます。Cookie が見つかった場合は、ユーザー・プロファイルに基づいて製品リストが表示されます。このリストは、JESI include 文を使用してページに表示されます。

JESI control タグは、maxRemovalDelay および expiration に対してデフォルト以外の値も設定し、そのページのキャッシングを明示的に有効化します。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>

<jesi:control maxRemovalDelay="1000" expiration="300" cache="yes"/>
<%
  String customerId=CookieUtil.getCookieValue(request,"customerid");
  if (customerId==null) {
    // some unknown customer
%>
    <jesi:include page="genericwelcome.jsp" />
<%
  }
  else {
    // a known customer; trying to retrieve recommended products from profiling
    String recommendedProductsDescPages[]=
               ProfileUtil.getRecommendedProductsDescURL(customerId);
    for (int i=0; i < recommendedProductsDescPages.length; i++) {
%>
    <jesi:include page="<%=recommendedProductsDescPages[i]%>" />
<%
    }
  }
%>
```

例 4: control/include 次に、リクエスト・パラメータを含む JESI include 文の使用例を示します。メイン・ページには、次の URL からアクセスすると想定します。

```
http://host:port/application1/main.jsp?p2=abc
```

メイン・ページは、パラメータ設定 p2=abc を取得します。ページは次のとおりです。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<html>
<jesi:control cache="no" />
<jesi:include page="a.jsp?p1=v1" />
<h3>hello ...</h3>
<jesi:include page="b.jsp" />
<h3>world ...</h3>
<jesi:include page="c.jsp?p1=v2" copyParam="true" />
</html>
```

a.jsp ページは、パラメータ設定 p1=v1 を取得します。c.jsp ページは、設定 p1=v2 に加えて設定 p2=abc を（メイン・ページの URL に copyParam および p2 を設定した結果として）取得します。

さらに、トップレベルのページは、cache="no" の設定によって、キャッシュ不可です。実際には、このように集計ページがキャッシュ不可の場合のみ、JESI include タグで copyParam 設定を使用することに注意してください。これはリクエスト属性がリクエストのたびに変わるためです。また、この cache="no" 設定は、インクルード・ページには効果がないことにも注意してください。インクルード・ページは、デフォルトでキャッシュ可能な状態です。つまり、それぞれのインクルード・ページ自体に、cache="no" 設定を含む JESI control タグがないかぎり、キャッシュ可能です。

例 5: param タグを使用した control/include 次に、インクルード・ページ・リクエストに新規パラメータとしてランタイム値を追加する JESI `param` タグの使用例を示します。メイン・ページには、パラメータ設定 `p1=v1` を使用して次のような URL からアクセスすると想定します。

```
http://host:port/application/main.jsp?p1=v1
```

ページは次のとおりです。

```
<jesi:control cache="yes" />
<jesi:include page="a.jsp" >
   <% String v2 = null;
      if(request.getParameter("p1").equals("v1")
          v2 = "v1 set";
      else
          v2 = "v2 unset";
   %>
   <jesi:param name="p2" value="<%=v2%>" />
</jesi:include>
```

JESI template タグ

template/fragment の使用モデルで、テンプレート・コンテンツ（フラグメント以外）のキャッシング動作を指定するには、JESI `template` タグを使用します。（詳細は、6-6 ページの「JESI の使用モデル」を参照してください。）対応する HTTP ヘッダーは、ESI 仕様に基づいて設定されます。ここでは、フラグメント以外のコンテンツはテンプレート・コンテンツと呼ばれる独立したキャッシュ可能オブジェクトで、JESI `fragment` タグで設定される各フラグメントのコンテンツも独立したキャッシュ可能オブジェクトです。

> **重要：** すべてのレスポンスの出力は、`template` の開始タグと終了タグの間で生成される必要があります。JESI `template` 開始タグは、できるだけページの先頭に配置してください。ページ内のすべての JESI `fragment` タグやバッファ・フラッシュより前に指定する必要があります。また、テキスト、HTML マークアップ、改行または空白など、他の参照可能な出力のコンテンツより前に指定する必要があります。JESI `template` 終了タグは、できるだけページの末尾で、ページ内の他のすべての JESI `fragment` タグや他の参照可能な出力のコンテンツの後に配置してください。

JESI `template` タグは常に JESI `fragment` タグとともに使用してください。個別のフラグメントが不要な場合は、JESI `template` タグではなく、JESI `control` タグを使用します。

次の点に注意してください。

- JESI `template` タグの属性はすべてオプションです。何も設定せずにタグを使用すると、デフォルトではレスポンスのキャッシュは 24 時間で期限切れになるように設定され、期限切れのオブジェクトは即時に削除されます。
- template/fragment モデルでは、キャッシュ可能かどうかの決定を ESI プロセッサに移譲できません。JESI `template` タグを使用する必要があります。また、テンプレート・コンテンツがキャッシュ可能かどうかは、`template` タグ属性設定またはデフォルト値（使用可能な場合）によって決まります。同様に、各フラグメントは JESI `fragment` タグを使用して設定する必要があり、各フラグメントがキャッシュ可能かどうかは `fragment` タグ属性設定またはデフォルト値によって決まります。
- 1 つの JSP ページに複数の JESI `template` タグを使用しないでください。また、追加の JESI `template` タグを、`jsp:include` 機能によって同じレスポンス・オブジェクトにインクルードされているページに使用しないでください。いずれの場合も、例外が発生します。
- 上位レベルのページに `template` タグがなく、この項で説明する他の関連する制限に従っていれば、標準 `jsp:include` タグを介してインクルードされるページに JESI `template` タグを配置できます。

- JSP コンテナが JESI control タグを検出しており、同じレスポンスの生成中に JESI template タグを検出すると、template タグの属性はすべて無視され、キャッシングは control タグに従って実行されます。

  注意： この場合、template タグが完全に無視されるわけではありません。JESI control タグも含むページに template/fragment モデルを使用すると（この状況は、template タグを含むページが通常は条件付きで動的にインクルードされる場合に発生することがあります）、JSP コンテナは必要に応じて fragment タグを適切に囲む template タグが存在することを認識します。

- JESI template タグのキャッシュ可能かどうかの設定は、各フラグメントには影響を与えません。フラグメントには独自の設定（またはデフォルト値）が必要です。
- リクエスト・パラメータがフラグメントで必要な場合は、それを囲んでいるテンプレート・コンテンツをキャッシュしない（JESI template タグに cache="no" を設定）か、テンプレート・コンテンツをキャッシュするかのいずれかを、パラメータ値に従って実行する必要があります。異なるバージョンのフラグメントも、パラメータ値に従ってキャッシュする必要があります。

  フラグメントのバックグラウンドでは、JESI include タグによってインクルードされたページと同様に、追加リクエストが関係しています。リクエスト・パラメータ（設定されている場合）は、常にテンプレートからフラグメントに渡されます。これは、copyParam="true" が設定された JESI include タグの機能と同じです。（この種の問題については、6-14 ページの「JESI include タグ」でも説明しています。）

JESI template タグには、JESI control タグの場合と同じ使用方法で同じ属性を指定します。

構文

```
<jesi:template
             [ expiration = "value" ]
             [ maxRemovalDelay = "value" ]
             [ cache = "yes" | "no" | "no-remote" ]
             [ control = "uninterpreted_string" ] >

...page content, jesi:fragment tags, optional jesi:include tags, optional
jesi:codeblock tags..

</jesi:template>
```

属性

属性の説明は、6-13 ページの「JESI control タグ」を参照してください。

JESI fragment タグ

template/fragment モデルでは、JESI template タグ内の JESI template の開始タグと終了タグの間で、1 つ以上の JESI fragment タグを使用します。（詳細は、6-6 ページの「JESI の使用モデル」を参照してください。）JESI fragment タグはそれぞれ、必要に応じて、JSP ページの個別のフラグメントのキャッシング動作を定義します。各フラグメントは独立したキャッシュ可能オブジェクトです。

特定のフラグメントが ESI 機能によって集計レスポンスにインクルードするためにリクエストされると、ESI プロセッサはそのフラグメントのみを取得します。

JESI fragment タグには、JESI control タグおよび JESI template タグの場合と同じ使用方法で同じ属性を指定します。

次の点に注意してください。

- JESI `fragment` タグごとに、独自のキャッシング指示を ESI プロセッサに指定します。キャッシュ可能かどうかは、指定した属性設定またはデフォルト値（使用可能な場合）によって決まります。JESI `template` タグの設定は、フラグメントには影響を与えません。
- JESI `fragment` タグを別の JESI `fragment` タグにネストすることはできません。
- control/include モデルとは異なり、template/fragment モデルでは、`template` タグと `fragment` タグ（使用可能な場合）が必要な場合に、キャッシング指示を ESI プロセッサに移譲できません。キャッシュ可能かどうかは、常に `template` または `fragment` タグ属性設定かデフォルト値によって決まります。
- この項で説明する制限に従っていれば、標準 `jsp:include` タグを介してインクルードされるページに JESI `fragment` タグを配置できます。JESI `fragment` タグを囲む JESI `template` タグは、同じインクルード・ページ、または `jsp:include` 文を含むページなど、上位レベルのページに配置できます。

構文

```
<jesi:fragment
              [ expiration = "value" ]
              [ maxRemovalDelay = "value" ]
              [ cache = "yes" | "no" | "no-remote" ]
              [ control = "uninterpreted_string" ] >

...JSP code fragment...

</jesi:fragment>
```

属性

属性の説明は、6-13 ページの「JESI control タグ」を参照してください。

JESI codeblock タグ

template/fragment モデルでは、必要に応じて 1 つ以上の JESI `codeblock` タグをテンプレート・コード内（フラグメント以外）で使用して、特定のコード・ブロックの条件付き実行を指定できます。各 `codeblock` タグは 1 つのコード・ブロックを囲み、その実行のタイミングを次のように指定します。

- テンプレートがリクエストされたときのみ

または

- 1 つ（任意）のフラグメントがリクエストされたときのみ

または

- 常時（テンプレートとフラグメントのどちらがリクエストされるかに関係なく）

このタグを使用しないと、リクエストごと（各テンプレート・リクエストと任意のフラグメントに対する各リクエスト）にすべてのテンプレート・コードが実行されますが、フラグメント・リクエストの場合はテンプレート出力が破棄されます。

フラグメントがリクエストされるたびにテンプレートを実行することが重要ですが（フラグメントが変数宣言や初期化のようなテンプレート・コードの副次効果に依存できるようにするため）、フラグメントに重要でないコード・ブロックも存在する場合があります。この種のコード・ブロックを `codeblock` タグ内に配置し、テンプレートがリクエストされた場合にのみ実行するように指定できます。

または、潜在的にすべてのフラグメントに不可欠であっても、テンプレート自体には不可欠でないテンプレート・コード・ブロックが存在する場合もあります。この種のコード・ブロックを `codeblock` タグ内に配置し、任意のフラグメントがリクエストされた場合にのみ実行するように指定できます。

注意： JESI codeblock タグ内では、参照可能な出力を生成しないことをお薦めします。これは、テンプレート・リクエストとフラグメント・リクエストの実行の違いによる予期しない動作を避けるためです。execute="template"（または"always"）の場合にテンプレートがリクエストされると、意図したとおりにコードが実行されてコンテンツが出力されます。ただし、execute="fragment"（または"always"）の場合にフラグメントがリクエストされると、コードは実行されますが、テンプレートの出力全体が非表示になります（フラグメントがリクエストされた場合は常に非表示です）。6-7 ページの「template/fragment モデル」の図 6-3 を参照してください。

構文

```
<jesi:template ... >
...
   <jesi:codeblock execute = "template" | "fragment" | "always" >
      ...request-dependent JSP content...
   </jesi:codeblock>
...
</jesi:template>
```

属性

- execute（必須）：値「template」を指定すると、テンプレートがリクエストされた場合にのみコード・ブロックが実行されます。値「fragment」を指定すると、任意のフラグメントがリクエストされた場合にのみコード・ブロックが実行されます。「always」を設定すると、コード・ブロックはページ・リクエストごとに実行されます。これは、codeblock タグを使用しない場合と同等です。

例 : template/fragment モデル

この項では、template/fragment モデルでの JESI タグの使用例を示します。

例 1: template/fragment この例は、JESI template タグと JESI fragment タグの一般的な使用例です。タグには expiration 属性のみが設定されているため、他のすべての設定はデフォルトに従います。cache 属性はデフォルトで「yes」に設定されるため、テンプレートと 3 つのフラグメントすべてがキャッシュされます。

テンプレート・コンテンツ（フラグメント以外）には、JESI template タグに基づいて、3600 秒の期限切れ時間が使用されています。この期限切れの設定は、フラグメント以外のすべての HTML ブロックに適用されます。JSP コード・ブロック #1 は期限切れ設定 60 で、JSP コード・ブロック #2 はデフォルトの期限切れ設定で、JSP コード・ブロック #3 は期限切れ設定 600 でそれぞれキャッシュされます。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<jesi:template expiration="3600">
...HTML block #1...
   <jesi:fragment expiration="60">
   ...JSP code block #1...
   </jesi:fragment>
...HTML block #2...
   <jesi:fragment>
   ...JSP code block #2...
   </jesi:fragment>
...HTML block #3...
  <jesi:fragment expiration="600">
   ...JSP code block #3...
   </jesi:fragment>
...HTML block #4...
</jesi:template>
```

例 2: template/fragment この例では、フラグメント内で JESI include タグを使用します。次に、このページのキャッシュ可能オブジェクトを示します。

- 各インクルード・ページ
- 各フラグメント（インクルード・ページ以外）
- HTML ブロックの集計（テンプレート・レベルにあるフラグメント以外のすべてのブロック）

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<jesi:template expiration="3600">
...HTML block #1...
   <jesi:fragment expiration="60">
   ...JSP code block #1...
   <jesi:include page="stocks.jsp" />
   </jesi:fragment>
...HTML block #2...
   <jesi:fragment>
   ...JSP code block #2...
   <jesi:include page="/weather.jsp" />
   </jesi:fragment>
...HTML block #3...
   <jesi:fragment expiration="600">
   ...JSP code block #3...
   <jesi:include page="../sales.jsp" />
   </jesi:fragment>
...HTML block #4...
</jesi:template>
```

例 3: codeblock を使用した template/fragment この例では、template/fragment モデルでの codeblock タグの使用方法を示す概念的な例を示します。この例では、パフォーマンスを改善するために、データベースへの接続コードがコード・ブロック内に配置されているため、不必要に再実行されることはありません。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<jesi:template>
Welcome to the Frequent Flyer Home page!
<jesi:codeblock execute="fragment" >
  /* Open a database connection and store it in the variable dbConn. */
</jesi:codeblock>
BEST DEALS
<jesi:fragment expiration="600" maxRemovalDelay="180">
...in Air Travel
/* Select the three cheapest USA domestic round-trip fares, using the database
   connection stored in dbConn. */
</jesi:fragment>

<jesi:fragment expiration="600" maxRemovalDelay="180">
...in Accommodations
/* select the three best hotel deals, using the database connection stored in dbConn.
*/
</jesi:fragment>

Click here to access your current Mileage account <...>
</jesi:template>
```

キャッシュ内のオブジェクトの無効化に使用するタグとサブタグの説明

ESI プロセッサでキャッシュ内のオブジェクトを明示的に無効化するには、JESI `invalidate` タグと、必要に応じて、次のサブタグを使用します。

- JESI `object`
- JESI `cookie`（JESI `object` のサブタグ）
- JESI `header`（JESI `object` のサブタグ）

次の各項では、これらのタグの構文、JESI 構成ファイル（無効化対象となるユーザー名、パスワードおよびログイン先 URL の指定に使用可能）について説明し、例を示します。

概要は、6-11 ページの「キャッシュ内のオブジェクトの無効化」を参照してください。

JESI invalidate タグ

JESI `invalidate` タグをそのサブタグ JESI `object` とともに使用して、1 つ以上のキャッシュ内のオブジェクトを明示的に無効化します。

サブタグは、次のように使用します。

- URI または URI 接頭辞に基づいて、無効化対象を指定するには、必須の JESI `object` サブタグを使用します。
- 必要に応じて、JESI `object` タグの JESI `cookie` サブタグまたは JESI `header` サブタグ（あるいはその両方）を使用し、Cookie または HTTP ヘッダー情報に基づいて、無効化対象の追加基準を指定します。

構文

```
<jesi:invalidate
             [ url = "url"
               username = "user_name"
               password = "password" ]
             [ config = "configfilename" ]
             [ output = "browser" ] >
```

必須サブタグ（6-27 ページの「JESI object サブタグ」を参照）

```
             <jesi:object ... >
```

JESI `object` のオプションのサブタグ（6-28 ページの「JESI cookie サブタグ」を参照）

```
                  <jesi:cookie ... />
```

JESI `object` のオプションのサブタグ（6-29 ページの「JESI header サブタグ」を参照）

```
                  <jesi:header ... />

             </jesi:object>

</jesi:invalidate>
```

ユーザー、パスワードおよび URL のすべてを、それぞれ個別の属性を使用して指定するか、または config 属性で参照するかデフォルトの場所にある構成ファイルにすべてを指定します。デフォルトの場所は /WEB-INF/jesi.xml、または下位互換性のために維持されている /WEB-INF/config.xml です。構成ファイルの詳細は、6-26 ページの「JESI 構成ファイル」を参照してください。構成ファイルと属性設定を使用してユーザー名、パスワードおよび URL を指定すると、属性設定が優先されます。

OC4J の jazn-data.xml ファイル内で OracleAS Web Cache の「invalidator」アカウント用に <user> 要素を指定すると、パスワードをクリア・テキストで指定するかわりに、password 属性に特殊構文を使用して jazn-data.xml 内の情報を参照できます。パスワードは、jazn-data.xml では不明瞭化された形式で指定されます。次の username および password 属性の説明を参照してください。jazn-data.xml ファイルの詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

注意： invalidate タグ内で複数の object タグを使用することは可能です。

属性

- url: キャッシュ・サーバーの URL を指定します。この属性を省略すると、ユーザー名、パスワードとともに URL を JESI 構成ファイルに指定する必要があります。
- username: 無効化のためにキャッシュ・サーバーにログインするユーザー名を指定します。OracleAS Web Cache では、通常、「invalidator」ユーザー名が必要です。この属性を省略すると、パスワード、URL とともにユーザー名を JESI 構成ファイルに指定する必要があります。

  OC4J の jazn-data.xml ファイルに OracleAS Web Cache の「invalidator」アカウント用の <user> 要素が含まれている場合は、そのアカウント名を username 値に次のように使用できます。

  ```
  username="invalidator"
  ```

- password: 無効化を実行するためにキャッシュ・サーバーにログインするためのパスワードを指定します。この属性を省略すると、ユーザー名、URL とともにパスワードを JESI 構成ファイルに指定する必要があります。

  OC4J の jazn-data.xml ファイルに OracleAS Web Cache の「invalidator」アカウント用の <user> 要素が含まれている場合は、次のように特殊な右矢印構文でダッシュ（-）と右山カッコ（>）に続けて invalidator アカウント名を指定して、そのファイルから不明瞭化されていないパスワードを取得できます。

  ```
  password="->invalidator"
  ```

- config: アプリケーション相対位置またはページ相対位置を使用して、JESI 構成ファイルを指定します。この構成ファイルを使用すると、対応するタグ属性を使用せずに、キャッシュ・サーバーの URL、無効化の基準となるユーザー名およびパスワードを指定できます。次の点に注意してください。
  - 構成ファイルを config 属性に指定するかわりに、デフォルトの場所にある構成ファイルを使用できます。6-26 ページの「JESI 構成ファイル」を参照してください。
  - タグ属性で username、password および url をすべて指定すると、構成ファイルは参照されません。
- output: 必要に応じて、キャッシュ・サーバーから無効化レスポンスを受信する出力デバイスを設定します。現在サポートされている設定は、ユーザーの Web ブラウザにメッセージを表示するために Web キャッシュ・レスポンスを HTML 形式でラップする「browser」のみです。このパラメータを設定しないと、無効化レスポンスは表示されません。

JESI 構成ファイル

JESI の提案仕様では、構成ファイルの使用がサポートされます。現在、構成ファイルを使用できるのは、無効化対象のユーザー名、パスワードおよび URL を指定する場合のみです。（または、各 JESI `invalidate` タグの属性でユーザー名、パスワードおよび URL を指定できます。6-24 ページの「JESI invalidate タグ」を参照してください。）

JESI 構成ファイルには、トップレベルの `<jesi-config>` 要素、その下の `<invalidation>` サブ要素、および `<invalidation>` 要素の下位の `<username>`、`<password>` および `<url>` サブ要素が必要です。

注意： 下位互換性を維持するために、現在は `<jesi-config>` および `<invalidation>` のかわりに廃止予定の要素 `<ojsp-config>` および `<web-cache>` をそれぞれ使用できます。新規要素は、JESI の提案仕様に準拠することを目的としています。`<ojsp-config>` および `<web-cache>` は、将来のリリースでサポート外になると思われます。

現行の実装では、2 つのデフォルト・ファイルが可能です。または、ファイルをアプリケーションの任意の場所に配置し、`invalidate` タグの `config` 属性で（アプリケーション相対位置またはページ相対位置を使用して）ファイル名と場所を指定できます。

優先デフォルト・ファイルは、JESI の提案仕様に準拠する `/WEB-INF/jesi.xml` です。下位互換性を維持するために、以前のデフォルト・ファイル `/WEB-INF/config.xml` もサポートされます。

無効化対象のユーザー名、パスワードおよび URL の取得には、次の優先順位が使用されます。

1. JESI `invalidate` タグに `username`、`password` および `url` 属性設定が（3 つすべて）指定されている場合は、その値が使用されます。

   注意： `invalidate` タグに 3 つの属性すべてではなく 1 つまたは 2 つを指定すると、例外が発生します。また、1 つ以上の属性値に空の文字列または null を指定した場合も、例外が発生します。

   ---

2. `invalidate` タグに `username`、`password` および `url` を指定せずに `config` 属性に構成ファイルを指定すると、指定した構成ファイルからの値が使用されます。
3. `invalidate` タグに `username`、`password`、`url` および `config` を指定しないと、JSP コンテナはデフォルトの構成ファイルを使用します。最初に、コンテナは `/WEB-INF/jesi.xml` ファイルを検索し、見つかった場合はそのファイルからの設定を使用します。ファイルが見つからない場合は `/WEB-INF/config.xml` を検索し、見つかった場合はそのファイルからの設定を使用します。

   注意： 次の場合には、`invalidate` タグにユーザー名、パスワードおよび URL を指定しないと、例外が発生します。

   - いずれかの時点で、3 つの属性すべてが指定されていない構成ファイルが見つかった場合
   - 構成ファイルが見つからない場合

OC4J の `jazn-data.xml` ファイルに OracleAS Web Cache の「invalidator」アカウント用の `<user>` 要素が含まれている場合は、特殊な右矢印構文でダッシュ（-）と右山カッコ（>）に続けて invalidator アカウント名を指定して、JESI 構成ファイルにそのアカウント名を使用し、`jazn-data.xml` ファイルからパスワードを取得できます。次の「例 2: jazn-data.xml からパスワードを取得する構成ファイル」を参照してください。

例 1: クリア・テキストによるパスワードを含む構成ファイル 次の例に、URL とログイン情報の設定用に、url、username および password 属性のかわりに使用する構成ファイルを示します。

```
<?xml version="1.0" ?>
<jesi-config>
  <invalidation>
      <url>http://yourhost.yourcompany.com:4001</url>
      <username>invalidator</username>
      <password>invpwd</password>
  </invalidation>
</jesi-config>
```

例 2: jazn-data.xml からパスワードを取得する構成ファイル 次の例では、クリア・テキストを使用してパスワードを指定するかわりに、特殊な「->」構文を使用して jazn-data.xml ファイルから不明瞭化されていないパスワードを取得します。この例では、jazn-data.xml に OracleAS Web Cache の「invalidator」アカウント用の <user> 要素が含まれていることを前提としています。

```
<?xml version="1.0" ?>
<jesi-config>
  <invalidation>
      <url>http://yourhost.yourcompany.com:4001</url>
      <username>invalidator</username>
      <password>->invalidator</password>
  </invalidation>
</jesi-config>
```

JESI object サブタグ

完全な URI または URI 接頭辞に基づいて、無効化するキャッシュ内のオブジェクトを指定するには、JESI invalidate タグの必須サブタグ JESI object を使用します。必要に応じて、JESI cookie サブタグまたは JESI header サブタグ（あるいはその両方）を使用し、Cookie または HTTP ヘッダー情報に基づいて無効化の追加基準を指定します。

uri 属性の設定に、完全な URI または URI 接頭辞のいずれかを指定します。このフィールドが完全な URI または接頭辞として解析されるかどうかは、prefix 属性の設定によって決まります。

構文

```
<jesi:object uri = "uri_or_uriprefix"
           [ maxRemovalDelay = "value" ]
           [ prefix = "yes" | "no" ] >
```

オプションのサブタグ（次の「JESI cookie サブタグ」を参照）

```
                  <jesi:cookie ... />
```

オプションのサブタグ（6-29 ページの「JESI header サブタグ」を参照）

```
                  <jesi:header ... />

</jesi:object>
```

どちらのサブタグも使用しない場合の構文を次に示します。

```
<jesi:object uri = "uri_or_uriprefix"
           [ maxRemovalDelay = "value" ]
           [ prefix = "yes" | "no"] />
```

注意 :

- invalidate タグ内で複数の object タグを使用することは可能です。
- object タグ内で複数の cookie タグまたは header タグを使用することは可能です。

属性

- uri（必須）: 無効化するキャッシュ内のオブジェクトに対応するページの完全な URI（prefix="no" の場合）、または位置に従って無効化される複数ページのオブジェクトを指定する URI 接頭辞（prefix="yes" の場合）のいずれかを指定します。

  接頭辞を指定すると、そのディレクトリに属しているページのキャッシュ内のオブジェクトがすべて無効化されます。たとえば、「/abc/def」という接頭辞を設定した場合は、対応するディレクトリとサブディレクトリにあるすべてのページについて、キャッシュ内のオブジェクトが無効化されます。
- prefix: uri 属性を URI 接頭辞としてのみ解析する場合は、「yes」に設定します。uri 属性を完全な URI として解析する場合は、デフォルト設定の「no」を使用します。
- maxRemovalDelay: キャッシュ内のオブジェクトが無効化されてから削除されるまで、つまり ESI プロセッサで提供できなくなるまでの最大遅延秒数を指定します。この遅延のデフォルトは 0（ゼロ）で、即時に削除されます。

JESI cookie サブタグ

無効化の追加基準として Cookie 情報を使用する場合は、JESI object タグ（JESI invalidate タグのサブタグ）の JESI cookie サブタグを 1 つ以上使用します。この Cookie 情報は、JESI object タグでの URI または URI 接頭辞の設定に加えて使用されます。JESI header タグに加えて使用される場合もあります。cookie タグは、複数バージョンがキャッシュされているオブジェクトを Cookie 情報に基づいて無効化する場合に役立ちます。

cookie タグには、ボディはありません。

構文

```
<jesi:cookie name = "cookie_name"
           [ value = "cookie_value" ] />
```

注意 :

- object タグ内で複数の cookie タグを使用することは可能です。
- 他のほとんどの JESI タグ属性とは異なり、value 属性には null 値または空の文字列値を指定できます。

属性

- name（必須）: Cookie の名前です。
- value: Cookie の値です。

cookie サブタグの使用箇所ごとに、無効化するオブジェクトのリクエスト URL には、name 属性設定（および指定されている場合は value 属性設定）と一致する Cookie が含まれている必要があります。

JESI header サブタグ

無効化の追加基準として HTTP/1.1 ヘッダー情報を使用する場合は、JESI object タグ（JESI invalidate タグのサブタグ）の JESI header サブタグを 1 つ以上使用します。このヘッダー情報は、JESI object タグでの URI または URI 接頭辞の設定に加えて使用されます。JESI cookie タグに加えて使用される場合もあります。header タグは、複数バージョンがキャッシュされているオブジェクトをヘッダー情報に基づいて無効化する場合に役立ちます。

header タグには、ボディはありません。

構文

```
<jesi:header name = "header_name"
             value = "header_value" />
```

> **注意：** object タグ内で複数の header タグを使用することは可能です。

属性

- name（必須）: HTTP/1.1 ヘッダーの名前です。
- value（必須）: HTTP/1.1 ヘッダーの値です。

header サブタグの使用箇所ごとに、無効化するオブジェクトのリクエスト URL には、name および value 属性設定と一致するヘッダーが含まれている必要があります。

例 : ページの無効化

この項では、JESI invalidate タグ、そのサブタグ JESI object および JESI object タグの JESI cookie サブタグを使用したページの無効化の例を示します。

例 1: ページの無効化 この例では、完全な URI によって指定された、ESI プロセッサ内のシングル・オブジェクトを無効化します。（デフォルトでは、object タグの uri 属性は、URI 接頭辞ではなく、完全な URI を指定します。）JESI invalidate タグは、キャッシュ・サーバーの URL および無効化アカウントのユーザー名とパスワードも指定します。さらに、キャッシュ・サーバーからの無効化レスポンスをユーザーのブラウザに表示することを指定します。

```
...
<jesi:invalidate url="http://yourhost.yourcompany.com:4001"
                 username="invalidator" password="invpwd"
                 output="browser">
     <jesi:object uri="/images/logo.gif"/>
</jesi:invalidate>
...
```

例 2: ページの無効化 この例は、直前の「例 1: ページの無効化」と同じです。ただし、構成ファイルを使用してキャッシュ・サーバー URL とログイン情報を指定します。

```
...
<jesi:invalidate config="/myconfig.xml" output="browser">
     <jesi:object uri="/images/logo.gif"/>
</jesi:invalidate>
...
```

JESI invalidate タグは、構成ファイルのアプリケーション相対位置を指定します。たとえば、myconfig.xml には、次のようなコンテンツがあります。

```
<?xml version="1.0" ?>
<jesi-config>
  <invalidation>
      <url>http://yourhost.yourcompany.com:4001</url>
      <username>invalidator</username>
      <password>invpwd</password>
  </invalidation>
</jesi-config>
```

例 3: ページの無効化 この例では、URI 接頭辞「/」に従って、ESI プロセッサ内の全オブジェクトを無効化します。この例では、無効化レスポンスのブラウザでの表示は指定していません。したがって、このメッセージはまったく表示されません。

```
...
<jesi:invalidate url="http://yourhost.yourcompany.com:4001"
                 username="invalidator" password="invpwd">
     <jesi:object uri="/" prefix="yes"/>
</jesi:invalidate>
...
```

例 4: ページの無効化 この例では、シングル・オブジェクトを無効化します。ただし、このオブジェクトは最大 30 分（1800 秒）間は有効状態を維持できます。

```
...
<jesi:invalidate url="http://yourhost.yourcompany.com:4001"
                 username="invalidator" password="invpwd">
     <jesi:object uri="/images/logo.gif" maxRemovalDelay="1800"/>
</jesi:invalidate>
...
```

例 5: ページの無効化 この例では、6-29 ページの「例 1: ページの無効化」と同じオブジェクトに無効化を指定します。ただし、リクエスト URL に値 `customer` を持つ `user_type` という名前の Cookie がある場合のみ、そのオブジェクトを無効化することを指定します。

```
...
<jesi:invalidate url="http://yourhost.yourcompany.com:4001"
                 username="invalidator" password="invpwd">
     <jesi:object uri="/images/logo.gif">
          <jesi:cookie name="user_type" value="customer"/>
     </jesi:object>
</jesi:invalidate>
...
```

ページのパーソナライズに使用するタグの説明

同じキャッシュ内のページが複数のユーザー間で共有されている状態で、ページのカスタマイズを許可するには、そのページによる Cookie 情報とセッション情報への依存性を ESI プロセッサに通知する必要があります。たとえば、Cookie 値の置換は、Web サーバーではなく、ESI プロセッサで発生します。

JESI personalize タグ

JESI `personalize` タグは、現行リクエストからの Cookie 値を置換してからキャッシュ内のページを提供するように ESI プロセッサに指示して、ページのカスタマイズを許可します。

このタグには、Cookie 名と値を含む ESI プレースホルダをレスポンス・ボディに挿入する効果があります。`name` 属性に指定した Cookie がリクエスト内で見つからず、null 以外の値が指定されている場合は、その値が使用されます。リクエスト内で Cookie が見つからないか、null 値が指定されていても、`default` 属性に値が指定されている場合は、新規 Cookie が作成され、`default` の値が使用されます。以前に Cookie 値が存在せず、`default` 値が指定されていない場合、このタグは効果がありません。

`personalize` タグには、ボディはありません。

構文

```
<jesi:personalize name = "cookie_name"
                [ default = "default_value" ] />
```

注意 :

- 「value」形式の default 属性は廃止予定ですが、下位互換性のために使用できます。value から default への変更は、JESI の提案仕様に準拠するために行われました。value は将来のリリースではサポート外になると思われます。
- OC4J では、ESI 仕様に準拠するために、指定された default（または value）設定は一重引用符で自動的に囲まれます。OC4J 9.0.4 より前の実装では、設定の一部として一重引用符を含める必要がありました。

属性

- name（必須）: ページのパーソナライズに使用する値を持つ Cookie の名前を指定します。
- default: Cookie が見つからない場合、または null 値が指定されている場合のオプションのデフォルト値です。

例 : ページのパーソナライズ

次に、JESI personalize タグの使用例を示します。

```
<jesi:personalize name="user_id" default="guest" />
```

生成された ESI タグによって、ESI プロセッサは必要な情報を検出できます。この場合、プロセッサは、user_id という名前の Cookie を検索し、その値を取得します。この Cookie が見つからない場合は、デフォルト値の「guest」を使用します。

この Cookie 値の置換を ESI プロセッサで処理することによって、ESI プロセッサはアプリケーション・サーバーの関与なしで、キャッシュ内の単一コピーからカスタマイズされた複数のページを提供できます。

JESI タグの処理と JESI と ESI 間での変換

JESI タグ・ハンドラ・クラスは、OC4J の JESI タグ・ライブラリの一部として提供され、JSP 機能から ESI 機能へのブリッジの役を果たします。タグ・ハンドラは、JESI タグからの ESI タグの生成、無効化に対する HTTP リクエストの生成、HTTP レスポンス・ヘッダーの設定などを行います。ただし、この変換は、JESI タグと ESI タグの間または JESI タグ属性と ESI タグ属性の間の単純な 1 対 1 のマッピングとはかぎりません。

例 : JESI と ESI 間でのインクルード・ページの変換

JESI と ESI 間での変換の例として、次の JSP コードを示します。

```
<p>BEGIN</p>
<jesi:control cache="no"/>
<jesi:include page="stocks.jsp" flush="true" />
<p>
<hr>
<jesi:include page="/weather.jsp" copyParam="true" flush="true" />
<p>
<hr>
<jesi:include page="../sales.jsp?tax=local" copyParam="true" flush="true" />
<p>END</p>
```

この JSP コードは、次の URL を含むページの一部と想定します。

```
http://host:port/application1/top.jsp
```

さらに次のリクエストを想定します。

```
http://host:port/application1/top.jsp?city=Washington_DC
```

この場合、JESI include タグ・ハンドラは、次のレスポンスのように、ESI マークアップを生成します。

レスポンス・ヘッダーは、次のとおりです。

```
Surrogate-Control: content="ESI/1.0",max-age=86400+0,no-store
```

レスポンス・ボディは、次のとおりです。

```
<p>BEGIN</p>
<esi:include src="/application1/stocks.jsp"/>

<p>
<hr>
<esi:include src="/weather.jsp?city=Washington_DC"/>

<p>
<hr>
<esi:include src="/sales.jsp?tax=local&city=Washington_DC"/>

<p>END</p>
```

このレスポンスは、クライアントに配信される前に、ESI プロセッサによって読み取られます。Surrogate-Control ヘッダーは、ESI プロセッサに対して、レスポンス・ボディに ESI マークアップが含まれていることを警告します。したがって、キャッシング機能は、ESI タグのレスポンス・ボディ内を検索します。また、Surrogate-Control ヘッダーは cache="no" 属性設定に従ってキャッシュ・ディレクティブを no-store に設定します。この場合、期限切れ間隔と最大遅延間隔は影響しません。

3 つの esi:include タグのそれぞれに対して、ESI プロセッサは、指定された URL に追加リクエストを作成します。各レスポンスは、トップレベルのページにインクルードされ、そのページがアセンブルされた後でのみ、クライアントに配信されます。クライアントが受信するレスポンスは 1 つですが、キャッシュでは、そのレスポンスを取得するために最初に 4 つのリクエストが作成されます。この操作は、オーバーヘッドが大きいように見えますが、weather.jsp などのように、他の多数のリクエストによって同じインクルード・ページが使用される場合は、全体の効率が向上します。これらのページは、ESI プロセッサ上で個別にキャッシュされるため、これらのページに対するリクエストは不要です。

例 : JESI と ESI 間でのテンプレートとフラグメントの変換

従業員が企業イントラネット・サイトに接続する場合を想定します。すべてのレスポンスに存在する少数の機能を除いて、そのページのコンテンツは動的です。特に、株式チャートとその企業に関する最新のビジネス見出しを表示するフッターは常に存在しています。このビジネス見出しは外部のビジネス・ニューズ・サイトから取得されます。戻すページのすべてに同じ情報が含まれている必要があり、取得にはコストがかかるため、このフッターは ESI プロセッサでキャッシュするほうが効率的です。

ページ・レスポンスの残りの部分は動的で、毎回わずかに異なる方法で株式のフラグメントが取り込まれます。ページの再書込みを回避するために、フッターに JESI フラグメントのマークを、それを囲んでいるページに JESI テンプレートのマークを付けることができます。

チャリティ・キャンペーンの期間中、主催者が目標金額と現在の寄付金額を示す棒グラフを、すべての企業ページの一部として表示すると想定します。この情報は、特別のデータベース表に格納されており、毎日 2 回更新されています。この棒グラフは、JESI フラグメントの追加として適切な候補です。JESI template タグをページの最上部に追加し、JESI fragment タグを使用して、個別エンティティとしてキャッシュするフラグメントを囲みます。

企業ページへの URL は、次の URL と想定します。

```
http://www.bigcorp.com/employee_page.jsp
```

さらに、そのページを次のように変更したと想定します。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jesitaglib.tld"
           prefix="jesi" %>
<jesi:template cache="no" >

   <p>BEGIN</p>
   ... some dynamic page content...
   <jesi:fragment>
   This_is_the_body_of_Charity_Chart
   </jesi:fragment>
   ... some more dynamic content...
   <jesi:fragment>
   This_is_the_body_of_Business_Footer
   </jesi:fragment>
</jesi:template>
<p>END</p>
```

ページのリクエスト時に、次の HTTP レスポンスが生成されます。

レスポンス・ヘッダーは、次のとおりです。

```
Surrogate-Control: content="ESI/1.0",max-age=86400+0,no-store
```

レスポンス・ボディは、次のとおりです。

```
<p>BEGIN</p>
... some dynamic page content...
<esi:include src="/employee_page.jsp?__esi_fragment=1"/>
... some more dynamic content...
<esi:include src="/employee_page.jsp?__esi_fragment=2"/>
<p>END</p>
```

6-31 ページの「例 : JESI と ESI 間でのインクルード・ページの変換」に示した JESI `include` の例と同様に、`Surrogate-Control` レスポンス・ヘッダーは ESI プロセッサに対して警告を通知します。`no-store` ディレクティブが生成された理由は、JESI `template` タグに `cache="no"` の設定があるためです。

ESI プロセッサは、2 つのリクエストを追加作成し、2 つのフラグメントをフェッチしてキャッシュします。その後で、合成されたページが従業員に戻されます。従業員がそのページで再度作業をすると、動的コンテンツが新規に生成されます。ただし、チャートとフッターはキャッシュから提供されます。

注意 : `Surrogate-Control` ヘッダーは、ESI プロセッサによって使用され、クライアントへの最終レスポンスには表示されません。

7

Web Object Cache のタグと API

この章では、OC4J が提供するアプリケーション・レベルのキャッシング機能である Web Object Cache について説明します。Java で作成された Web アプリケーションでは、Web Object Cache を Oracle Application Server Web Cache とともに使用するとスピードとスケーラビリティが増大します。

この章には、次の項目が含まれます。

- Web Object Cache の概要
- Web Object Cache の主要機能
- ポリシー仕様の属性と使用
- Web Object Cache タグの説明
- Web Object Cache API の説明
- キャッシュ・ポリシー・ディスクリプタ
- キャッシュ・リポジトリ・ディスクリプタ
- バックエンド・リポジトリの構成

OracleAS Web Cache および Oracle Application Server Java Object Cache の説明を含む Web キャッシングの概要は、1-13 ページの「Web アプリケーションに対する Oracle キャッシング・サポートのサマリー」を参照してください。

Web Object Cache の概要

OC4J Web Object Cache は、Java で記述された Web アプリケーションが、JSP やサーブレットなどの動的 Web ページが生成した部分的な結果と中間結果を取得、格納、再利用、後処理およびメンテナンスできるようにする機能です。プログラミング・インタフェースについては、タグ・ライブラリと Java API を備えています。

Web Object Cache は Java レベルで機能し、JSP アプリケーションとサーブレット・アプリケーションの HTTP 環境と緊密に統合されています。キャッシュ内のオブジェクトは、HTML または XML のフラグメント、XML DOM オブジェクトまたは Java のシリアライズ可能オブジェクトで構成できます。

Web Object Cache のプログラミング・インタフェースを使用すると、Web ページをページ・ブロックに分割できます。ページ・ブロックでは、キャッシング制御の精度を向上させるために、個別のキャッシュ・オブジェクトを定義します。（ここでは、ブロックとオブジェクトという用語は、同義に使用されています。）これによって、アプリケーション自体が、実行時にキャッシュ・エンティティの存続時間とその他の動作を個別に管理できます。アプリケーション開発者は、自分のアプリケーションの Web ページのライフ・サイクル・パターンを最もよく理解しています。したがって、ページをキャッシュ・ブロックに分割する方法を決定する適任者です。キャッシュ内のオブジェクトのメンテナンス・ポリシーは、外部ファイルに宣言して指定するか、キャッシュ・ポリシー・ディスクリプタに指定するか、またはアプリケーション自体にプログラムで指定できます。

次の各項では、Web Object Cache の概要を説明します。

Web Object Cache の利点

注意： Web Object Cache は特定の使用例で有用であり、OracleAS Web Cache などの他のキャッシング機能の必要性を否定するものではありません。Web Object Cache の概要、OracleAS Web Cache や Oracle Application Server Java Object Cache との関連、およびそれぞれどのような場合に適しているかの説明は、1-13 ページの「Web アプリケーションに対する Oracle キャッシング・サポートのサマリー」を参照してください。

Web Object Cache を使用することによって、データベースへの問合せやその結果の書式化または変換など、コストのかかる中間処理を伴う動的アプリケーションでの、ページ・ブロックや Java オブジェクトの構築に要する時間が大幅に削減されます。後続の問合せでは、このキャッシュから情報を取得します。このため、問合せと書式化を繰り返して行う必要がありません。

さらに、開発者は、API コールやカスタム JSP タグを使用し、プログラムによってキャッシュを厳密に制御できます。これには、キャッシュ・エントリの作成時期、名前の指定、期限切れ時期、参照できるユーザーと参照対象のキャッシュ・データ、結果をユーザーに提示する前にキャッシュ・データに適用できる操作などの制御が含まれます。

一部の Web アプリケーションでは、対象データの特質と使用方法に応じて Web Object Cache を使用することで、多くの利点を得ることができます。たとえば、カタログやディレクトリのブラウズ、遅延株式相場およびパーソナライズ・ポータルなどのアプリケーションにとっては、特に大きな利点となります。ただし、リアルタイムの株取引や株式相場などのアプリケーションにとっては、利点となりません。これは、データの更新があまりにも頻繁に行われるため、キャッシング操作のオーバーヘッドによって利点が損われてしまうためです。（ただし、このような状況でも、OracleAS Web Cache が役立つ可能性があります。これはオーバーヘッドが比較的少ないためです。）

通常、Web Object Cache は、次の場合に最も効果を発揮します。

- XSLT または XML DOM 操作など、キャッシュ内のデータ・オブジェクトに対して特別な後処理を行う場合。
- キャッシュ内の XML データや Java オブジェクトの再利用、SMTP、JMS、AQ または SOAP によるデータの送信など、HTTP 以外の状況でデータを共有する場合。
- 存続期間の長いデータを永続的に格納するための、ファイル・システムまたはデータベースへのキャッシュ内のデータの格納など、特別な格納のニーズに対応する場合。
- Web ベースのグループウェア・アプリケーションなど、様々なデータ項目への様々なアクセス権を様々なユーザーに許可する、アプリケーション固有の認可が必要な場合。

  アプリケーションには独自の認可スキームを設定できます。Web Object Cache は、Java の認可ロジック内に埋め込まれています。

Web Object Cache は、JSP ページで使用すると特に便利です。JSP コードの生成によって、開発作業を大幅に軽減できます。

Web Object Cache のコンポーネント

Web Object Cache は、次の 2 つの主要コンポーネントで構成されています。

- キャッシュ・リポジトリ
- キャッシュ・プログラミング・インタフェース

この項では、Web Object Cache のデフォルトのキャッシュ・リポジトリである Oracle Application Server Java Object Cache の概要についても説明します。

キャッシュ・リポジトリ

キャッシュ・リポジトリは、データの格納、データ配分およびキャッシュの期限切れに関連したコンポーネントです。プログラム可能な Web キャッシュ（Web Object Cache など）には、層やプラットフォームに応じて、複数のリポジトリ実装が可能です。たとえば、ファイル・システムを中間層の 2 次記憶域として使用し、データベース表をデータベース層の 1 次記憶域として使用できます。

Web Object Cache では、Oracle Application Server Java Object Cache をデフォルトのリポジトリとして使用しています。Java Object Cache は、アプリケーションで使用するために設計された汎用の Java キャッシング・サービスおよび API で、オブジェクトには名前でアクセスできます。

Java Object Cache は、強力で柔軟性の高いプログラミング機能です。キャッシュできるオブジェクトの種類やオブジェクトの生成元に制限はありません。各オブジェクトの管理は簡単にカスタマイズできます。各オブジェクトには、次のような一連の属性があります。

- オブジェクトのキャッシュへのロード方法
- オブジェクトの格納場所（メモリー内、ディスク上、またはその両方）
- オブジェクトの存続期間（TTL とも呼ばれます）
- オブジェクトが無効化された場合の通知先

オブジェクトはグループ単位または個別に無効化できます。

Java Object Cache の詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

注意： Java Object Cache またはファイル・システムを Web Object Cache のバックエンド・リポジトリとして構成する方法の詳細は、7-42 ページの「バックエンド・リポジトリの構成」を参照してください。

キャッシュ・プログラミング・インタフェース

フロントエンド・キャッシング・インタフェースは、JSP ページおよびサーブレットを介して HTTP 処理を実行し、キャッシュ・ポリシー（キャッシュの動作を決定するルールと仕様）に関連するセマンティクスを指示するために使用します。

OC4J Web Object Cache のプログラミング・インタフェースは、次のように分割できます。

- Web Object Cache API

  これは、サーブレットと JSP ページ間の共通レイヤーで、HTTP セマンティクスとキャッシュ・ポリシーを処理します。このレイヤーは、キャッシュ・リポジトリと通信します。

- Web Object Cache タグ・ライブラリ

  これは、JSP のカスタム・タグ機能を使用した、Web Object Cache API 用の便利なラッパーです。キャッシングの制御には、JSP ページでカスタム・タグを使用します。この場合、API は基礎となるタグ・ハンドラ・クラスを介してコールされます。

この章では、これらのプログラミング・インタフェースとキャッシュ・リポジトリとの相互作用について説明します。キャッシュ・タグについては、7-15 ページの「Web Object Cache タグの説明」で説明します。基礎となるキャッシュ・ポリシー API については、7-27 ページの「Web Object Cache API の説明」で説明します。サーブレットでは、基礎となる API を使用し、JSP ページでは通常、より便利なタグを使用します。

キャッシュ・ポリシーとスコープ

キャッシュ・ポリシーとは、キャッシュの詳細とその動作を決定する一連の仕様です。次の項目が含まれます。

- キャッシュ・スコープ
- キャッシュ・ブロックのネーミング規則
- データの期限切れルール
- キャッシュ・リポジトリ名

キャッシュ・ポリシーの仕様（7-8 ページの「ポリシー仕様の属性と使用」を参照）は、次のいずれかを使用して設定できます。

- キャッシュ・タグ属性（JSP ページ用）

  7-15 ページの「Web Object Cache タグの説明」を参照してください。

- キャッシュ・ポリシー・メソッド（サーブレット用）

  7-27 ページの「Web Object Cache API の説明」を参照してください。

- 外部キャッシュ・ポリシーのディスクリプタ・ファイル（JSP ページまたはサーブレット用）

  7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。

キャッシュ・ポリシー・オブジェクト（`oracle.jsp.jwcache.CachePolicy` クラスのインスタンス）は、これらの入力に基づいたポリシー設定を使用して作成されます。期限切れポリシーは、キャッシュ・ポリシーの一部であるため、各 `CachePolicy` オブジェクトには、`oracle.jsp.jwcache.ExpirationPolicy` クラスのインスタンスである属性が含まれます。

キャッシュ・データは、session scope（現行の HTTP セッションに対してのみ使用可能な場合）または application scope（アプリケーションの全ユーザーに対して使用可能な場合）のいずれかになります。

たとえば、口座の残高をキャッシュするオンライン銀行取引のアプリケーションを考えてみます。この情報に関心があるのは、現行のユーザーのみです。したがって、`session` スコープが適切です。

一方、全ユーザーに対して同じ汎用製品を推奨する「ようこそ」ページを持つオンライン・ストアについて考えてみます。この場合のページには `application` スコープを持つキャッシュの使用が適切です。

Web Object Cache の主要機能

次の各項では、Web Object Cache の主要機能について説明します。

キャッシュ・ブロックのネーミング : 暗黙的なネーミングと明示的なネーミングの比較

キャッシュ・ブロックは、キャッシュ・ブロック名に関連付けられています。キャッシュ・ブロック名は、キャッシング・ポリシーによって暗黙的（通常はこれを推奨）に、またはアプリケーション・コードによって明示的に決定できます。ページ取得時には、対象のページ・フラグメントの再生成を回避するために、キャッシュ・ブロック名をルックアップします。

暗黙的なネーミングの場合は、次の 2 つを入力します。

- キャッシュ・ポリシー

  キャッシュ・ポリシー API レイヤーが、ネーミング・ロジックを実行します。

- HTTP リクエスト・オブジェクト

  キャッシング・ロジックは、標準の Java Servlet API から対応するセマンティクスを借用します。

ほとんどの場合、暗黙的なネーミングで指定された名前には、十分な情報が含まれています。これは、通常、Web アプリケーションに対するすべての入力（アプリケーションで生成する内容を決める入力）が HTTP リクエストに含まれているためです。

ただし、明示的なネーミングのほうが適切な場合があります。たとえば、ユーザー・グループが同じデータを共有する場合です。この場合は、関連する識別情報がユーザーの HTTP リクエストから直接使用できない可能性があるため、暗黙的なキャッシュ名は役に立ちません。かわりに、そのグループを識別するキャッシュ名を明示的に生成するコードを作成します。名前生成ロジックには、アプリケーション内に存在する他の状態ではなく、リクエスト・パラメータのみを入力として使用することをお薦めします。これによってセマンティクスをたどり、コードをデバッグすることが容易になります。

次に明示的なネーミングの例を示します。`cache` タグでは、`someMethod()` をコールする JSP 式を含む `name` 属性によって、キャッシュ・ブロック名が設定されます。

```
<ojsp:cache policy="/WEB-INF/policy1.cpd"
            name="<%= someObj.someMethod() %>" >
...static text...
<% // dynamic content ... %>
</ojsp:cache>
```

次の例では、`name` 属性が `cache` タグ内に存在しないため、キャッシュ・ブロック名は HTTP リクエストとキャッシュ・ポリシーに基づいて暗黙的に決定されます。

```
<ojsp:cache policy="/WEB-INF/policy2.cpd" >
...static text...
<% // dynamic content ... %>
</ojsp:cache>
```

詳細は、7-11 ページの「キャッシュ・ブロックのネーミングと autoType 属性の詳細」を参照してください。

注意 : キャッシュ・ブロックは、ネストできます。この場合、内部キャッシュ・ブロックのロジックは、外部ブロックのコンテンツの再生成が必要な場合のみ実行されます。

クローン化可能なキャッシュ・オブジェクト

OC4J Web Object Cache には、oracle.jsp.jwcache.CloneableCacheObj インタフェースが備わっています。このインタフェースは、クローン化可能にする必要があるシリアライズ可能なキャッシュ・オブジェクトに実装できます。シリアライズされずに、キャッシュされた変更可能なオブジェクトに対して、クローニングは、キャッシュ・オブジェクトの完全な階層コピーを作成するのに有効です。この項では、クローニングの有用性について説明します。最初に必要なバックグラウンド情報を説明します。

メモリー指向リポジトリと 2 次記憶装置リポジトリとの比較

Web Object Cache のバックエンドとして使用できるリポジトリには、次の 2 つのカテゴリがあります。

- 2 次記憶装置キャッシュ・リポジトリ（ファイル・システム・リポジトリなど）
- メモリー指向キャッシュ・リポジトリ（Web Object Cache のデフォルト・リポジトリである Oracle Application Server Java Object Cache など）

2 次記憶装置リポジトリでは、キャッシュ操作時に Java のシリアライズが必要です。キャッシュへの格納時、オブジェクトはリポジトリにシリアライズされます。キャッシュからの取得時、そのオブジェクトはメモリーにデシリアライズされます。したがって、シリアライズとデシリアライズ処理の結果、キャッシュ・オブジェクトの完全な個別コピーが各キャッシュ操作時に自動的に作成されます。

これは、キャッシュ・オブジェクトをメモリー指向リポジトリとの間で格納または取得する場合には、当てはまりません。メモリー指向リポジトリでは、ユーザー・アプリケーション内の同一オブジェクトがキャッシュに格納されます。あるいは、キャッシュ内の同一オブジェクトがユーザーのために取得されます。デフォルトでは、コピーは作成されません。複数の取得が行われる場合は、すべての取得で同じオブジェクトが共有されます。

キャッシュ・オブジェクトをクローニングするメリット

多くのアプリケーションでは、異なる取得ごとに、1 つのキャッシュ・オブジェクトの異なるコピーを使用する場合があります。これには 2 つの主要な理由があります。

- 同一キャッシュ・オブジェクトを複数の取得間で共有する場合は、ある場所でデータに変更を加えると、別の場所で取得および使用する値に対して意図せずに影響を与える可能性があります。
- 同一キャッシュ・オブジェクトを複数の取得間で共有する場合は、複数の Java スレッドが同じオブジェクトに同時にアクセスする可能性があります。その結果、元のオブジェクト設計がスレッド・セーフでなかった場合は、スレッド・セーフティの問題が発生します。たとえば、オブジェクトが最初からページ・スコープまたはリクエスト・スコープ専用に設計されている場合、各オブジェクトに対するスレッドは 1 つのみのはずです。このスレッド動作の前提条件に違反することになります。

この問題を回避するために、汎用の Java のシリアライズ可能データをメモリー指向リポジトリとの間で格納または取得するときは、完全な階層コピーを使用してください。「完全な階層」が意味することは、オブジェクトが参照する直接的なメンバーのみではなく、参照するすべての間接的な変数をコピーすることにあります。たとえば、オブジェクト xyz にメンバー変数として java.util.Vector インスタンスがあるとします。完全な階層コピーのクローニングでは、Vector インスタンス自体の他に、Vector インスタンスが参照するすべての変更可能なオブジェクトや要素のコピーが含まれます。

CloneableCacheObject インタフェースの使用

CloneableCacheObject インタフェースとその cloneCacheObj() メソッドをキャッシュ・オブジェクトに実装した場合、Web Object Cache は、そのキャッシュ・オブジェクトがメモリー指向キャッシュ・リポジトリとの間で格納または取得されると、cloneCacheObj() を自動的にコールし、各キャッシュ・オブジェクトの完全な階層コピーを作成します。

キャッシュ・ブロックのランタイム機能

実行時に Web Object Cache キャッシュ・タグが検出されると、タグ・ハンドラは、対応するキャッシュ・オブジェクトが存在するかどうか、そのオブジェクトが最近作成されたもので再利用可能であるかどうかをチェックします。再利用可能な場合、タグ・ボディ内のコードは実行されずに、そのキャッシュ・オブジェクトが再利用されます。ただし、そのキャッシュ・オブジェクトが存在しない場合や古すぎる場合は、タグ・ボディのコードが実行され、新規オブジェクト（ページ・フラグメント、XML DOM オブジェクトまたは Java のシリアライズ化可能オブジェクト）が生成されます。次に、この新しく生成されたオブジェクトは、特別なバッファ書込みまたはオブジェクトの受渡しによって取得され、キャッシュに格納されます。

コンテンツ生成に複雑なデータベース問合せなどが含まれ、コストがかかる場合、およびキャッシュの存続時間が適切であるためにキャッシュ内のデータが再利用可能な場合は、Web Object Cache によって、時間とシステム・リソースを大幅に節約できます。また、アプリケーションのスピードとスループットが大幅に改善されます。

データの無効化と期限切れ

キャッシュ・ブロックは、指定した継続時間後または指定時刻に期限切れとなるように設定できます。またはメソッド・コールやタグの起動によって明示的に無効化できます。

キャッシュ・ブロックの期限切れ

キャッシュ・ブロックは主に準静的なフラグメント情報で構成されているため、Oracle の実装には厳密に一貫性のある期限切れモデルは不要です。通常、一貫性の低いモデルでも許容できる結果が得られ、同期によるオーバーヘッドが減少します。

Web Object Cache ブロックのデータの期限切れには、次の 2 つのカテゴリがあります。

- 継続時間（TTL）: データは、指定した時間キャッシュ内に存在した後、期限切れとなります。
- 固定日 / 時 : 毎日指定した時間または毎週指定した曜日など、設定した時間に定期的に期限切れとなります。

期限切れの詳細は、`oracle.jsp.jwcache.ExpirationPolicy` クラスのインスタンスにある属性設定によって決定されます。この `ExpirationPolicy` オブジェクトは、キャッシュ・ブロックに関連付けられた `CachePolicy` オブジェクトの属性です。7-13 ページの「期限切れポリシーの属性」を参照してください。

JSP ページでは、`ExpirationPolicy` 属性は Web Object Cache キャッシュ・タグの属性を使用して設定できます。サーブレットでは、`ExpirationPolicy` オブジェクトのメソッドを直接使用できます。（詳細は、7-33 ページの「ExpirationPolicy メソッド」を参照してください。）あるいは、キャッシュ・ポリシー・ディスクリプタを使用して、`ExpirationPolicy` 属性を設定できます。（詳細は、7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。）

キャッシュ・ブロックの無効化

期限切れに基づいてキャッシュを無効にするかわりに、次の方法を使用してキャッシュを明示的に無効化できます。

- `invalidateCache` タグを使用します。7-24 ページの「Web Object Cache の invalidateCache タグ」を参照してください。
- 1 つ以上のキャッシュ・ブロックを明示的に無効化するには、`CachePolicy` インスタンスのオーバーロードされた `invalidateCache()` メソッド、`invalidateCacheLike()` メソッドまたは `invalidateCacheOtherPathLike()` メソッドを使用します。7-29 ページの「CachePolicy メソッド」を参照してください。

ポリシー仕様の属性と使用

この項では、キャッシュ・ポリシーの属性、特に CachePolicy クラスと ExpirationPolicy クラスの属性について説明します。これらの属性は、JSP ページではカスタム・タグを使用して設定でき、サーブレットでは付属の Java API を使用して直接設定できます。キャッシュ・ポリシー・ディスクリプタ・ファイルを使用しても設定できます。

キャッシュ・ポリシーの属性

キャッシュ・ポリシーについては、7-4 ページの「キャッシュ・ポリシーとスコープ」で説明しています。このポリシーは、キャッシュ・ブロックの動作を決定する詳細な項目で構成されています。後続の項で説明するように、キャッシュ・ポリシーの属性は複数の方法で設定できます。

- JSP ページでは、カスタム・タグを使用します。

  7-15 ページの「Web Object Cache タグの説明」を参照してください。

- サーブレットでは、メソッドのコールを使用します。

  7-29 ページの「CachePolicy メソッド」を参照してください。

- キャッシュ・ポリシー・ディスクリプタ・ファイルを使用します。

  7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。

キャッシュ・ポリシー設定を指定すると、結果的にキャッシュ・ポリシー・オブジェクトが作成されます。このオブジェクトには、期限切れポリシー・オブジェクトが属性の 1 つとして含まれます。次の短縮コードは CachePolicy クラス（oracle.jsp.jwcache パッケージ内）のコードで、キャッシュ・ポリシー属性の名前を示しています。ただし、このコードは説明用のコードです。

```
class CachePolicy
{
   boolean ignoreCache;
   int scope;
   int autoType;
   String selectedParameters[];
   String selectedCookies[];
   Date reusableTimeStamp;
   long reusableDeltaTime;
   ExpirationPolicy expirationPolicy;
   String cacheRepositoryName;
   boolean reportException;
}
```

注意： 後述する整定数に対する名前は、サーブレットで使用される名前です。Web Object Cache タグには様々な名前を使用できます。7-16 ページの「Web Object Cache の cache タグ」を参照してください。

キャッシュ・ポリシー属性の説明

表 7-1 で、キャッシュ・ポリシー・オブジェクトの属性について説明します。

表 7-1 キャッシュ・ポリシー属性の説明

属性	型	説明
ignoreCache	boolean	開発時専用の属性です。コードを頻繁に変更する場合は、この属性を `true` に設定してキャッシュを無効にします。その結果、通常、変更前に生成された結果が戻ることはありません。 デフォルト : `false`
scope	int	キャッシュのスコープを指定します。現行の HTTP セッションのみにアクセスできるキャッシュ・ブロックには、整定数 `SCOPE_SESSION` を、アプリケーションの全 HTTP セッションにアクセスできるキャッシュ・ブロックには、`SCOPE_APP` を使用します。 デフォルト : `SCOPE_APP`
autoType	int	キャッシュ・ブロックのネーミングを明示的に行うか暗黙的に行うかを指定し、暗黙的なネーミングの場合は HTTP リクエストのプロパティをキャッシュ・ブロックのネーミングで使用する方法も指定します。この名前は、後続リクエストでそのキャッシュが再利用される時点を決定するときに必要です。7-11 ページの「キャッシュ・ブロックのネーミングと autoType 属性の詳細」を参照してください。 デフォルト : 暗黙的。URI と全パラメータおよび選択した Cookie に従います（`TYPE_URI_ALLPARAM`）。
selectedParameters[]	String []	キャッシュ・ブロックのネーミングで使用するために選択したリクエスト・パラメータの名前です。`autoType` とともに使用します。7-11 ページの「キャッシュ・ブロックのネーミングと autoType 属性の詳細」を参照してください。 デフォルト : `null`
selectedCookies[]	String[]	キャッシュ・ブロックのネーミングで使用するために選択した Cookie の名前です。`autoType` とともに使用します。7-11 ページの「キャッシュ・ブロックのネーミングと autoType 属性の詳細」を参照してください。 デフォルト : `null`

表 7-1 キャッシュ・ポリシー属性の説明（続き）

属性	型	説明
reusableTimeStamp	java.util.Date	キャッシュの使用可能性に対する絶対制限時間。この制限時間より前に作成されたすべてのキャッシュ・ブロックは、再利用されません。かわりに、データが再生成されますが、キャッシュ・ブロックの変更はありません。7-12 ページの「reusableTimeStamp と reusableDeltaTime の詳細」を参照してください。 reusableTimeStamp に関しては、次の点に注意してください。 ■ この属性は、1970 年 1 月 1 日午前 0 時から任意の絶対制限時間までをミリ秒で、または java.util.Date インスタンスとして表すことができます。cache タグを使用すると他の便利なフォーマットが利用可能です。（詳細は、7-15 ページの「Web Object Cache タグの説明」を参照してください。） ■ この属性は、reusableDeltaTime よりも優先されます。 ■ この属性の値が整定数 REUSABLE_ALWAYS または文字列定数 REUSABLE_IGNORED として設定されている場合、キャッシュ・エントリは、キャッシュ内にあるかぎり常に再利用できます。 ■ この属性は、XML キャッシュ・ポリシー・ディスクリプタ・ファイルでは使用できません。 デフォルト : 常に再利用可能
reusableDeltaTime	long	キャッシュの使用可能性に対する相対制限時間。キャッシュ・ブロックの作成時間と現在の時間の差が reusableDeltaTime より大きい場合、キャッシュ・ブロックは再利用されません。かわりに、データが再生成されますが、キャッシュ・ブロックの変更はありません。7-12 ページの「reusableTimeStamp と reusableDeltaTime の詳細」を参照してください。 reusableDeltaTime に関しては、次の点に注意してください。 ■ この属性は秒単位で指定します。 ■ reusableTimeStamp 属性によってオーバーライドされます。 ■ この属性の値が整定数 REUSABLE_ALWAYS または文字列定数 REUSABLE_IGNORED として設定されている場合、キャッシュ・エントリは、キャッシュ内にあるかぎり常に再利用できます。 デフォルト : 常に再利用可能
expirationPolicy	ExpirationPolicy	期限切れポリシー・オブジェクト（oracle.jsp.jwcache.ExpirationPolicy のインスタンス）。リポジトリによってキャッシュ・ブロックが記憶域から削除される状況を指定します。 デフォルト : デフォルトの期限切れポリシー・オブジェクト 期限切れポリシー・オブジェクト、パラメータおよびデフォルトの詳細は、7-13 ページの「期限切れポリシーの属性」を参照してください。

表 7-1 キャッシュ・ポリシー属性の説明（続き）

属性	型	説明
cacheRepositoryName	String	キャッシュ・リポジトリの名前。各キャッシュ・ポリシーは、独自のリポジトリを使用できます。 キャッシュ・リポジトリの構成は、`/WEB-INF/wcache.xml` ファイルに定義されています。 デフォルト：「DefaultCacheRepository」
reportException	boolean	この属性を `false` に設定すると、ほとんどのキャッシュ操作障害がサイレント状態になり、例外がブラウザにレポートされなくなります。 デフォルト：`true`

キャッシュ・ブロックのネーミングと autoType 属性の詳細

7-5 ページの「キャッシュ・ブロックのネーミング：暗黙的なネーミングと明示的なネーミングの比較」で説明したように、キャッシュ・ブロックの名前は暗黙的（自動ネーミングとも呼ばれます）または明示的（ユーザー・ネーミングとも呼ばれます）に指定できます。

具体的には、キャッシュ・ブロックのネーミングには、6 つの方法があります。最初の方法は、明示的なネーミングです。この方法は、`TYPE_USERSPECIFIED`（整定数）の `autoType` 設定によって指定します。

他の 5 つの方法は、暗黙的なネーミングのバリエーションです。

- リクエスト URI のみを名前に使用する暗黙的なネーミング

  `TYPE_URI_ONLY` の `autoType` 設定によって指定します。

- 次の内容に基づく暗黙的なネーミング

  リクエスト URI + 問合せ文字列 + 選択した Cookie

  `TYPE_URI_QUERYSTR` の `autoType` 設定によって指定します。`selectedCookies[]` 属性に Cookie を指定します。

- 次の内容に基づく暗黙的なネーミング

  リクエスト URI + すべてのパラメータ + 選択した Cookie（**デフォルト**）

  `TYPE_URI_ALLPARAM` の `autoType` 設定によって指定します。`selectedCookies[]` 属性に Cookie を指定します。

- 次の内容に基づく暗黙的なネーミング

  リクエスト URI + 選択したパラメータ + 選択した Cookie

  `TYPE_URI_SELECTEDPARAM` の `autoType` 設定によって指定します。パラメータを `selectedParameters[]` 属性に、Cookie を `selectedCookies[]` 属性に指定します。

- 次の内容に基づく暗黙的なネーミング

  リクエスト URI + 除外したパラメータ以外の全パラメータ + 選択した Cookie

  `TYPE_URI_EXCLUDEDPARAM` の `autoType` 設定によって指定します。Cookie を `selectedCookies[]` 属性に、除外したパラメータを `selectedParameters[]` 属性に指定します。

たとえば、各ユーザー用にパーソナライズした挨拶文を含む JSP ページ `welcome.jsp` を開発したと仮定します。パーソナライズされた挨拶文を含むデータは、そのページにある唯一のキャッシュ・ブロックです。さらに、「リクエスト URI + 選択したパラメータ + 選択した Cookie」に基づくネーミングを指定したと仮定します。この場合、キャッシュ・ブロックのネーミングに選択したパラメータは、`user` のみで、Cookie は選択していません。

このページが次のようにリクエストされたと仮定します。

```
http://host:port/a.jsp?user=Amy
```

この場合、a.jsp?user=Amy がキャッシュ・ブロックの名前になります。

さらに、このページがその後、別のユーザーによって、次のようにリクエストされたと仮定します。

```
http://host:port/a.jsp?user=Brian
```

この場合、Amy キャッシュは再利用されません。これは、user の値が異なるためです。かわりに、新規キャッシュ・ブロックが、a.jsp?user=Brian という名前で作成されます。

その後、最初のユーザーが次のようにリクエストしたと仮定します。

```
http://host:port/a.jsp?mypar=3&user=Amy
```

ユーザーが再度 Amy であるため、このリクエストでは最初のキャッシュが再利用され、Amy のカスタマイズ情報が再生成されることなく表示されます。mypar パラメータは、キャッシング機能とは関係ありません。これは、おそらく mypar の値はキャッシュ可能なページ出力には関係ないと判断され、キャッシュ・ポリシー・オブジェクトの selectedParameters[] リストにこのパラメータが含まれていないためです。

さらに次の後続リクエストを想定します。

```
http://host:port/a.jsp?yourpar=4&user=Brian&hello=true&foo=barfly
```

ユーザーが再度 Brian であるため、このリクエストでは第 2 のキャッシュが再利用され、Brian のカスタマイズ情報が再生成されることなく表示されます。yourpar、hello および foo の各パラメータは、キャッシング機能には無関係です。これは、キャッシュ・ポリシー・オブジェクトの selectedParameters[] リストにこれらのパラメータが含まれていないためです。

reusableTimeStamp と reusableDeltaTime の詳細

再利用可能の概念は、TTL の概念とは異なり、より細かい制御を目的としていることに注意してください。TTL は、キャッシュの一般的な存続期間を制御します。詳細は、「期限切れポリシーの属性」を参照してください。通常、キャッシュ内のデータの使用を適切に制限するには TTL のみ必要です。

再利用可能性に関する属性には、reusableTimeStamp と reusableDeltaTime があります。これらは、より特定した使用を目的としており、キャッシュ内のデータの期限切れまたは無効化には影響を与えません。たとえば、Web レポートの更新に対する要件がユーザーによって異なる状況を考えてみます。多くのユーザーが過去の任意の時間に作成されたレポートを受け入れることができる状況で、すべてのユーザーが同じバージョンを見て内容を比較しようと考えているとします。この場合、適切な TTL 値は、「1 日」です。

また、データの時間によって影響を受ける小グループの特権ユーザーがいると仮定します。このユーザー・グループには、1 時間以内の情報が必要であるとします。

この場合、TTL は、すべてのユーザーに対して「1 日」に設定されていますが、特権ユーザーに対しては「1 時間」の reusableDeltaTime 設定が可能です。この設定によって、データが 1 時間を経過した場合、このキャッシュは特権ユーザーに対しては使用されなくなります。ただし、reusableTimeStamp と reusableDeltaTime によりキャッシュが期限切れになったり、その他の影響を受けることはありません。キャッシュ内のデータは、特権ユーザー以外のユーザーに対しては、TTL に従ってそのまま使用できます。

特権ユーザー・グループに対する、reusableTimeStamp と reusableDeltaTime への値の設定は、アプリケーション・ロジックに依存します。

期限切れポリシーの属性

期限切れポリシーの概要は、7-7 ページの「データの無効化と期限切れ」で説明しています。期限切れポリシーには、キャッシュ・ブロックの期限切れ時点、そのデータが使用不可になる時点およびデータの再生成が必要な時点を決定する詳細が含まれています。（ほとんどの説明で、期限切れポリシーはキャッシュ・ポリシーの一部として考えることができます。）
ExpirationPolicy 属性は、CachePolicy 属性と同様に、次のいずれかの方法で設定できます。

- JSP ページでは、カスタム・タグを使用します。

  7-15 ページの「Web Object Cache タグの説明」を参照してください。

- サーブレットでは、メソッドのコールを使用します。

  7-33 ページの「ExpirationPolicy メソッド」を参照してください。

- キャッシュ・ポリシー・ディスクリプタ・ファイルを使用します。

  7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。

次の短縮コードは、ExpirationPolicy クラス（oracle.jsp.jwcache パッケージ内）のコードで、期限切れポリシー属性の名前を示しています。ただし、このコードは説明用のコードです。

```
class ExpirationPolicy
{
   int expirationType;
   long TTL;
   long timeInaDay;
   int dayInaWeek;
   int dayInaMonth;
   boolean writeThrough;
}
```

表 7-2 で、期限切れポリシー・オブジェクトの属性について説明します。

注意： 後述する整定数に対する名前は、サーブレットで使用される名前です。Web Object Cache タグには様々な名前を使用できます。7-16 ページの「Web Object Cache の cache タグ」を参照してください。

表 7-2 期限切れポリシー属性の説明

属性	型	説明
expirationType	int	これは期限切れポリシーの種類で、次のいずれかです（TYPE_*XXX* の値は整定数です）。 ■ TTL。TTL 属性に従って、一定時間後に期限切れになります。TYPE_TTL の expirationType 設定で指定します。 ■ DAILY。timeInaDay 属性に従って、1 日の指定した時間に期限切れになります。TYPE_DAILY の expirationType 設定で指定します。 ■ WEEKLY。dayInaWeek 属性と timeInaDay 属性に従って、指定した曜日の指定した時間に期限切れになります。TYPE_WEEKLY の expirationType 設定で指定します。 ■ MONTHLY。dayInaMonth 属性と timeInaDay 属性に従って、指定した日付の指定した時間に期限切れになります。TYPE_MONTHLY の expirationType 設定で指定します。 デフォルト：TTL

表 7-2 期限切れポリシー属性の説明（続き）

属性	型	説明
TTL	long	TTL。キャッシュ・ブロックが有効である期間を秒単位で表します。値は、正数である必要があります。 デフォルト : 300（5 分間）
timeInaDay	long	日単位、週単位または月単位の期限切れの時間。午前 0 時を基点に秒単位で表します。つまり、0（ゼロ）は 00:00:00（午前 0 時）を、86399 は 23:59:59 を表します。 デフォルト : 300（00:05:00）、`expirationType=TYPE_TTL` の場合は無視されます。
dayInaWeek	int	週単位の期限切れの曜日。`timeInaDay` で指定した時間に期限切れとなります。可能な値 : `WEEKLY_SUNDAY`、`WEEKLY_MONDAY`、`WEEKLY_TUESDAY`、`WEEKLY_WEDNESDAY`、`WEEKLY_THURSDAY`、`WEEKLY_FRIDAY` または `WEEKLY_SATURDAY`（整定数） デフォルト : WEDNESDAY、`expirationType=TYPE_WEEKLY` 以外の場合は無視されます。
dayInaMonth	int	月単位の期限切れの日付。たとえば、毎月 10 日の場合は、10 を指定します。`timeInaDay` で指定した時間に期限切れとなります。最大設定数は、そのキャッシュ・ブロックが作成された月の日数です。たとえば、キャッシュ・ブロックが作成されたのが 6 月の場合、`dayInaMonth` の設定値が 31 であっても、有効な値は 30 です。 デフォルト : 10、`expirationType=TYPE_MONTHLY` 以外の場合は無視されます。
writeThrough	boolean	キャッシュ・リポジトリでキャッシュ・エントリを即時書込みキャッシュとして処理するかどうかを指定するフラグ。指定すると、エントリはファイル・システムやデータベースなどの 2 次記憶装置に即時に書き込まれます。即時書込みモードでは、これを `true` に設定します。即時書込みキャッシュは、サーバーの再起動や電源障害の後も残ります。 `false` に設定すると、キャッシュ・エントリは、遅延書込みキャッシュとして処理されます。この設定は、存続時間が 5 分や 10 分と短く、再生成にもコストがかからないキャッシュに適切です。 キャッシュ・リポジトリの中には、即時書込みモードをサポートしていないリポジトリもあれば、必ず即時書込みモードを使用するリポジトリもあります。 デフォルト : `true`

Web Object Cache タグの説明

OC4J が提供するカスタム・タグを使用して、JSP ページから、キャッシュ・ポリシーの設定、期限切れポリシーの設定および明示的な無効化などを指定できます。次の各項では各タグについて説明します。

Web Object Cache タグ・ライブラリを使用する場合は、次の要件に注意してください。

- Web Object Cache クラスは、OC4J が提供する `ojsputil.jar` ファイル内にあります。このファイルは、予約済のタグ・ライブラリ・ディレクトリにあります。このファイルがインストール済で、クラスパスに存在していることを確認してください。
- Oracle Application Server Java Object Cache をバックエンド・リポジトリとして使用するには、`cache.jar` ファイルがインストール済で、クラスパスに存在している必要があります。このファイルも OC4J に同梱されています。OC4J 10.1.2 実装では、`cache.jar` は `oc4j.jar` のマニフェスト・クラスパスに示されます。Web Object Cache タグ・ライブラリが OC4J によってロードされる場合、ユーザー側操作は不要です。
- タグ・ライブラリ・ディスクリプタ・ファイル `jwcache.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`jwcache.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jwcache.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「ojsp:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を `taglib` ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

キャッシュ・タグの説明

この項では、次のタグについて説明します。

- `cache`

  一般的な文字ベースのキャッシング（HTML または XML のフラグメント）に使用します。

- `cacheXMLObj`

  XML オブジェクトのキャッシングに使用します。このオブジェクトのパラメータは、`cache` タグ・パラメータのスーパーセットを構成します。Web Object Cache は XML 文書の後処理時に特に便利なため、`cacheXMLObj` タグは、`cache` タグより頻繁に使用される可能性があります。

- `useCacheObj`

  Java のシリアライズ可能オブジェクトの一般的なキャッシングに使用します。一部のセマンティクスと構文は、標準の `jsp:useBean` タグに従ってパターン化されています。

- `cacheInclude`

  `cache` タグの機能と標準の `jsp:include` タグの機能を結合します。

この項では、キャッシュ・タグ内でのコードの条件付き実行によって発生する可能性がある問題、およびこの問題の回避策として、キャッシュ・ブロックを個別 JSP ページに分割し、必要に応じて cacheInclude タグを使用してページを適切に結合する方法についても説明します。

Web Object Cache の cache タグ

この項では、cache タグの構文と属性について説明します。このタグを使用すると、XML オブジェクトや Java のシリアライズ可能オブジェクトのキャッシングとは異なり、JSP アプリケーションで一般的なキャッシングを設定できます。

注意： XML オブジェクトのキャッシングには、かわりに cacheXMLObj タグを使用します。Java のシリアライズ可能オブジェクトのキャッシングには、useCacheObj タグを使用します。これらのタグは、この項で説明する cache タグのすべての属性をサポートしています。7-19 ページの「Web Object Cache の cacheXMLObj タグ」および 7-21 ページの「Web Object Cache の useCacheObj タグ」を参照してください。

構文

```
<ojsp:cache
   [ policy = "filename" ]
   [ ignoreCache = "true" | "false" ]
   [ invalidateCache = "true" | "false" ]
   [ scope = "application" | "session" ]
   [ autoType = "user" | "URI" | "URI_query" | "URI_allParam" |
                "URI_selectedParam" | "URI_excludedParam" ]
   [ selectedParam = "space-delimited_string_of_parameter_names" ]
   [ selectedCookies = "space-delimited_string_of_cookie_names" ]
   [ reusableTimeStamp = "yyyy.mm.dd hh:mm:ss z" |
                         "yyyy.mm.dd hh:mm:ss" | "yyyy.mm.dd"| "ignored" ]
   [ reusableDeltaTime = "number" | "ignored" ]
   [ name = "blockname" ]
   [ expirationType = "TTL" | "daily" | "weekly" | "monthly" ]
   [ TTL = "number" ]
   [ timeInaDay = "number" ]
   [ dayInaWeek = "Sunday" | "Monday" | "Tuesday" | "Wednesday" |
                  "Thursday" | "Friday" | "Saturday" ]
   [ dayInaMonth = "number" ]
   [ writeThrough = "true" | "false" ]
   [ printCacheBlockInfo = "true" | "false" ]
   [ printCachePolicy = "true" | "false" ]
   [ cacheRepositoryName = "name" ]
   [ reportException = "true" | "false" ] >

...Code for cache block...

</ojsp:cache>
```

注意： 主なデフォルト値は、TTL が 300 秒、dayInaMonth が 10（月の 10 日目）、キャッシュ・リポジトリ名が DefaultCacheRepository です。

属性

cache タグのパラメータのほとんどは、この章の前半で説明した CachePolicy クラスまたは ExpirationPolicy クラスの属性に対応しています（以降に参照先を示しています）。

- policy: 必要に応じて、キャッシュ・ポリシー・ディスクリプタを指定します。このディスクリプタの設定がキャッシュ・ポリシーの定義に使用されます。キャッシュ・ポリシー・ディスクリプタは、個別のキャッシュ・タグ属性の設定のかわりに使用できます。または、タグ属性の設定によってオーバーライドできるデフォルト値を設定できます。

  JSP アプリケーション相対 URL 構文に従ってディスクリプタ・ファイル名を指定します。アプリケーション相対 URL 構文の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

  次に、キャッシュ・ポリシー・ディスクリプタの簡単な例を示します。

```
<!--
test-policy.cpd
-->

<cachePolicy scope="application">
   <expirationPolicy expirationType="TTL" TTL="25" timeInaDay="00:10:00"
                     writeThrough="true" />
</cachePolicy>
```

  詳細は、7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。

- ignoreCache: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- invalidateCache: 最初に無効化するキャッシュ・ブロックに対応しているキャッシュ・ブロック（同じ名前を持つ既存の任意のキャッシュ・ブロック）に対して、このフラグを有効にします。これは、暗黙的なキャッシュ・ブロックのネーミングを使用している場合は特に便利です。ただし、明示的な名前にも使用できます。その場合は、cache タグの name 属性にキャッシュ・ブロック名を指定します。デフォルト設定は「false」です。

  注意： この属性を汎用的な invalidateCache タグと混同しないでください。7-24 ページの「Web Object Cache の invalidateCache タグ」を参照してください。invalidateCache 属性は、個別のキャッシュ・ブロックの無効化に使用する特殊なまたは拡張された属性です。

- scope: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- autoType: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。タグ属性の設定値とクラス属性値（整定数）の対応関係は、次のとおりです。
  - user 設定は、TYPE_USERSPECIFIED と等価です。
  - URI は、TYPE_URI_ONLY と等価です。
  - URI_query は、TYPE_URI_QUERYSTR と等価です。
  - URI_allParam は、TYPE_URI_ALLPARAM と等価です。
  - URI_selectedParam は、TYPE_URI_SELECTEDPARAM と等価です。
  - URI_excludedParam は、TYPE_URI_EXCLUDEDPARAM と等価です。
- selectedParam: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- selectedCookies: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- reusableTimeStamp: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- reusableDeltaTime: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- name: 明示的なキャッシュ・ブロックのネーミングを使用する場合は、name パラメータを使用してブロック名を指定します。

- expirationType: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- TTL: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- timeInaDay: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- dayInaWeek: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- dayInaMonth: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- writeThrough: 7-13 ページの「期限切れポリシーの属性」を参照してください。
- printCacheBlockInfo（デバッグ用）: このパラメータを有効にすると、キャッシュ・ブロックの内部キャッシュ名、作成時間および期限切れ時期が HTML/XML コメント構成メンバー内に出力されます。デフォルト設定は「false」です。
- printCachePolicy（デバッグ用）: このパラメータを有効にすると、このキャッシュ・ブロックのすべてのキャッシュ・ポリシー属性の値が HTML/XML コメント構成メンバー内に出力されます。デフォルト設定は「false」です。
- cacheRepositoryName: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- reportException: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

属性の使用方法

- name 属性は、autoType が user に設定されている場合のみ必要です。
- selectedParam 属性は、autoType が URI_selectedParam または URI_excludedParam に設定されている場合のみ必要です。
- selectedCookies 属性は、autoType が user または URI に設定されている場合は必要ありません。
- timeInaDay 属性は、expirationType が TTL に設定されている場合は必要ありません。
- dayInaWeek 属性は、expirationType が weekly に設定されている場合のみ必要です。
- dayInaMonth 属性は、expirationType が monthly に設定されている場合のみ必要です。

例 : cache タグ

次の例では、cache タグを使用して、一連の項目を表示およびキャッシュします。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jwcache.tld"
           prefix="ojsp" %>
<title>listitem.jsp</title>
<%
   String itemid=request.getParameter("itemid");
   if (itemid==null) {
     out.println("Please select a category from the above drop down box.");
     return;
   }
%>
<% long l1=(new java.util.Date()).getTime(); %>
<ojsp:cache autoType="URI_selectedParam" selectedParam="itemid"
  printCacheBlockInfo="true" printCachePolicy="true"
  policy="/WEB-INF/test-policy.cpd"
>
   Item List: <b><%= itemid %></b><br>
   Time: <%= new java.util.Date() %>
   <br>
   <jsp:useBean class="java.util.Hashtable" id="table" scope="application" />
   <hr>
   <%
     Vector list=(Vector) table.get(itemid);
     if (list==null) {
```

```
      out.println("No such item!");
    }
    else {
      for (int i=0; i<list.size(); i++) {
  %>
  <%= list.elementAt(i) %><br>
  <%
      }
    }
  %>
  timestamp:<%= new java.util.Date() %>
  <br>
</ojsp:cache>
<% long l2=(new java.util.Date()).getTime(); %>
Time for general cache operation:<%= l2-l1 %>
<br>
```

Web Object Cache の cacheXMLObj タグ

通常、XML DOM オブジェクトをキャッシュする場合は、cache タグではなく、cacheXMLObj を使用します。

cacheXMLObj タグは、7-16 ページの「Web Object Cache の cache タグ」で説明した cache タグのすべての属性、およびこの項で説明する属性をサポートしています。

構文（cache タグの構文に追加）

```
<ojsp:cacheXMLObj
   ...
   [ fromXMLObjName = "objectname" ]
   [ toXMLObjName = "objectname" ]
   [ toWriter = "true" | "false" ] >

[...Code for cache block...]

</ojsp:cacheXMLObj>
```

注意：

- このタグは、必要に応じて、ボディなしの単一タグにすることもできます。その場合、かわりに fromXMLObjName 属性を入力に使用できます。

```
<ojsp:cacheXMLObj ... fromXMLObjName="..." ... />
```

- また、このタグを XML タグ・ライブラリにコピーし、xml.tld タグ・ライブラリ・ディスクリプタ・ファイルに定義すると便利です。
- このタグは、XML プロデューサと XML コンシューマの両方の役割を果たします。XML オブジェクトが暗黙的に渡されている場合、fromXMLObjName と toXMLObjName は使用しないでください。（詳細は、5-2 ページの「XML プロデューサと XML コンシューマ」を参照してください。）

属性（cache タグの属性に追加）

- fromXMLObjName: 明示的な受渡しの場合は、（pageContext オブジェクトから）キャッシュに渡す XML 入力オブジェクトの名前を指定します。
- toXMLObjName: 明示的な受渡しの場合は、キャッシュから（pageContext オブジェクトに）渡す XML 出力オブジェクトの名前を指定します。
- toWriter: XML オブジェクトを JSP ライターに書き込み、ユーザーのブラウザに直接出力する場合は、「true」に設定します。デフォルト値は「false」です。

注意： cacheXMLObj タグは、OC4J が提供する複数のカスタム・タグの 1 つで、XML に関連したタグです。これらのタグは、場合により（あるいは常に）、XML オブジェクトを入力として取得したり、出力として作成します。この他に、この種のタグには、SQL ライブラリの dbQuery タグ（問合せ結果を XML DOM オブジェクトとして出力できます）および XML ライブラリの transform タグと styleSheet タグ（XML オブジェクトを入力として取得し、XSLT 変換を使用して別の XML オブジェクトまたは JSP Writer を出力として作成できます）があります。これらのタグは、XML データの明示的な受渡し用に fromXMLObjName 属性と toXMLObjName 属性を持つことで一致しています。詳細は、5-2 ページの「XML プロデューサと XML コンシューマ」を参照してください。

例 : cacheXMLObj タグ

この例では、Web Object Cache タグ、JESI タグおよび XML と SQL のタグ・ライブラリのタグが使用されています。（JESI タグについては、6-12 ページの「Oracle JESI タグの説明」を参照してください。XML transform タグについては、5-3 ページの「XML ユーティリティ・タグ」を参照してください。SQL タグについては、4-11 ページの「データ・アクセス用 SQL タグ」を参照してください。）

SQL dbOpen タグと SQL dbQuery タグは、データベースに接続して、問合せを実行します。cacheXMLObj タグは、問合せにより生成される XML DOM オブジェクトをキャッシュします。後続の実行（異なるスタイルシートによる出力など）では、問合せの再実行は不要です。これは、Web Object Cache から DOM オブジェクトを取得できるためです。XML transform タグは、変数を使用して指定した XML スタイルシートに基づいて問合せ結果を出力します。JESI fragment タグによって、キャッシュ対象の HTML 出力が囲まれますが、これにはアプリケーション・レベルのキャッシングは不要です。JESI template タグは、cache="no" 設定を使用して、フラグメント以外のキャッシングを無効にします。

```
<jesi:template cache="no">
<% String userStyleLoc="style/rowset.xsl"; %>
<h3>Transform DBQuery Tag Example</h3>
<h4>Current Time=<%= new java.util.Date() %></h4>
   <jesi:fragment expiration="60">
   <!-- You can cache HTML in OracleAS Web Cache with JESI
     or you can cache it in Oracle Web Object Cache -->
   <h4>Cached Time=<%= new java.util.Date() %></h4>
   <sql:dbOpen connId="conn1" dataSource="<%= dataSrcStr %>" />
   <xml:transform href="<%= userStyleLoc %>" >
   <%-- The XML DOM object is produced by dbQuery
     And, the DOM object is cached in Oracle Web Object Cache.
     XSLT is performed on the cached object. --%>
     <ojsp:cacheXMLObj TTL="60" toWriter="false">
       <sql:dbQuery connId="conn1" output="xml" queryId="myquery" >
          select ENAME, EMPNO from EMP
       </sql:dbQuery>
     </ojsp:cacheXMLObj>
   </xml:transform>
   <sql:dbCloseQuery queryId="myquery" />
   <sql:dbClose connId="con1" />
   </jesi:fragment>
</jesi:template>
```

Web Object Cache の useCacheObj タグ

シリアライズ可能なすべての Java オブジェクトをキャッシュするには、useCacheObj タグを使用します。

useCacheObj タグは、7-16 ページの「Web Object Cache の cache タグ」で説明した cache タグのすべての属性、およびこの項で説明する属性をサポートしています。

構文（cache タグの構文に追加）

```
<ojsp:useCacheObj
   ...
     type="classname"
     id = "instancename"
   [ cacheScope = "application" | "session" ] >

...Code for cache block...

</ojsp:useCacheObj>
```

> **注意：** id 属性と type 属性は、リクエスト時属性でないため、JSP の実行時の式を使用して設定することはできません。

属性（cache タグの属性に追加）

- type（必須）：キャッシュする Java オブジェクトのクラス名を指定します。
- id（必須）：キャッシュする Java オブジェクトのインスタンス名を指定します。
- cacheScope: この属性の使用方法は、cache タグと cacheXMLObj タグの scope 属性と同じです。7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

この項で説明する type 属性と id 属性は、jsp:useBean タグの type（または class）属性と id 属性と同じように使用されます。

例 : useCacheObj タグ

```
<ojsp:useCacheObj id="a2" policy="/WEB-INF/test-policy.cpd"
  type="examples.RStrArray" >
<%
   // create a temp writeable array
   WStrArray tmpa2=new WStrArray(3);
   tmpa2.setStr(2,request.getParameter("testing4"));
   tmpa2.setStr(1,"def");
   tmpa2.setStr(0, (new java.util.Date()).toString() );
   // create a readonly copy for the cache
   a2=new RStrArray(tmpa2);
   // storing the a2 into pagecontext
   // so useCacheObj tag can pick it up
   pageContext.setAttribute("a2",a2);
%>
</ojsp:useCacheObj>
```

Cache タグ内でのコードの条件付き実行

cache タグ（cache、cacheXMLObj または useCacheObj）内のコードは条件付きで実行されます。特に次のような場合が該当します。

- キャッシュ・タグ内のすべてのコードが実行されるのは、関連付けられたキャッシュ・ブロックが再利用されない場合のみです。

  次に例を示します。

  ```
  <% String str=null; %>
  <% ojsp:useCacheObj ... >
     <% str = "abc"; //...more Java code...%>
  </ojsp:useCacheObj>
  <% out.print(str.length()); // May cause null pointer exception
  ```

  このキャッシュが使用可能で再利用される場合、文字列 str を適切に初期化するコードは実行されません。

- キャッシュ・タグ内にメソッド・ベースの変数宣言を配置した場合、その変数をタグ外で使用することはできません。

  次に例を示します。

  ```
  <ojsp:useCacheObj ... >
     <% String str = "abc"; //...more Java code...%>
  </ojsp:useCacheObj>
  <% // String str will not be available here %>
  ```

cache タグ（cacheXMLObj や useCacheObj ではなく）を使用している場合は、キャッシュ・ブロックを個別の JSP ページに分割することによって、この種の状況に陥る可能性が少なくなります。この場合、各キャッシュ・ブロックは、独自の URI で表します。必要に応じて動的インクルード機能を使用すると、これらのページを結合できます。

この操作をさらに簡単にするために、Oracle は cacheInclude タグを用意しています。次項の「Web Object Cache の cacheInclude タグ」で説明します。

Web Object Cache の cacheInclude タグ

cacheInclude タグは、cache タグ（ただし、cacheXMLObj タグや useCacheObj タグを除く）の機能と標準の jsp:include タグの機能を結合します。

キャッシュ・ブロックの個別ページへの配置および cacheInclude タグの使用は、モジュール性や透明性、さらに前述の「Cache タグ内でのコードの条件付き実行」で説明した問題などを考慮すると、多くのメリットがあります。

ただし、次の制限事項に注意してください。

- cacheInclude タグには、ランタイムの JSP 式は使用できません。
- キャッシュ・ブロックには、暗黙的なキャッシュ・ブロックのネーミングを使用する必要があります。
- jsp:include タグとは異なり flush パラメータはありません。

これらの制限が問題になる場合は、cache タグと jsp:include タグを個別に使用します。

cacheInclude タグと JESI include タグの間の重要な違いにも注意してください。（このタグの詳細は、6-14 ページの「JESI include タグ」を参照してください。）OracleAS Web Cache は、Web Object Cache とは異なるキャッシング・レイヤーにあるため、JESI include タグのインクルード先ページとインクルード・ページは、同じリクエスト・オブジェクトを共有できません。cacheInclude タグにはこのような制限はありません。インクルード先ページとインクルード・ページは、同じリクエスト・オブジェクトを共有します。その結果、Bean と request スコープの属性をこの 2 つのページ間で相互に渡すことができます。

構文

```
<ojsp:cacheInclude
     policy = "filename"
     page = "URI"
   [ printCacheBlockInfo = "true" | "false" ]
   [ reportException = "true" | "false" ] >

...Code for cache block...

</ojsp:cacheInclude>
```

注意： cacheInclude タグの場合は、policy と page がリクエスト時属性でないため、JSP の式を使用してその値を決定することはできません。（cache タグ、cacheXMLObj タグおよび useCacheObj タグの場合、policy は、リクエスト時属性です。）

属性

- policy（必須）：キャッシュ・ポリシー・ディスクリプタ・ファイルを使用して、キャッシュ・ポリシーの設定を指定する必要があります。個別のパラメータ設定はサポートされていません。
- page（必須）：この page 属性を使用して、動的にインクルードするページの URI を指定します。標準の jsp:include タグと同じです。
- printCacheBlockInfo（デバッグ用）：7-16 ページの「Web Object Cache の cache タグ」を参照してください。
- reportException: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

属性の使用方法

次の cacheInclude タグの使用方法について考えてみます。

```
<ojsp:cacheInclude page="anotherPage.jsp" policy="foo.cpd" >
```

これは、次の例と同じです。

```
<ojsp:cache policy="foo.cpd" >
   <% pageContext.include("anotherPage.jsp"); %>
</ojsp:cache>
```

これは、次の例とも同じです。

```
<jsp:include page="anotherPage.jsp" flush="true" />
```

anotherPage.jsp は、次のように構成されているとします。

```
<ojsp:cache policy="foo.cpd" >
...anotherPage.jsp contents...
</ojsp:cache>
```

キャッシュ無効化タグの説明

この項では、invalidateCache タグの使用方法を説明します。

Web Object Cache の invalidateCache タグ

キャッシュ・ブロックをプログラム・ロジックを使用して明示的に無効化する場合は、invalidateCache タグを使用できます。この項では、このタグの構文と属性について説明します。

注意：

- invalidateCache タグは、新規の Cookie を受け入れません。したがって、使用できるのは、現行の HTTP リクエストの既存の Cookie のみです。新規 Cookie の入力の詳細は、7-29 ページの「CachePolicy メソッド」を参照してください。
- invalidateCache タグをキャッシュ・タグの invalidateCache 属性と混同しないでください。この属性には、既存のキャッシュ・オブジェクトを無効化するという制限があります。

構文

```
<ojsp:invalidateCache
   [ policy = "filename" ]
   [ ignoreCache = "true" | "false" ]
   [ scope = "application" | "session" ]
   [ autoType = "user" | "URI" | "URI_query" | "URI_allParam" |
                "URI_selectedParam" | "URI_excludedParam" ]
   [ selectedParam = "space-delimited_string_of_parameter_names" ]
   [ selectedCookies = "space-delimited_string_of_cookie_names" ]
   [ name = "blockname" ]
   [ invalidateNameLike = "true" | "false" ]
   [ page = "URI" ]
   [ autoInvalidateLevel = "application" | "page" | "param" | "cookie" ]
   [ cacheRepositoryName = "name" ]
   [ reportException = "true" | "false" ] />
```

注意： autoInvalidateLevel のデフォルト値は、ページ URI の仕様によって決まります。7-25 ページの「page と autoInvalidateLevel の使用」を参照してください。

属性

invalidateCache タグのほとんどのパラメータは、cache タグおよび cacheXMLObj タグにも存在し、同じように使用されます。詳細は、この章で前述しています（以降に参照先を示しています）。

- policy: 7-16 ページの「Web Object Cache の cache タグ」を参照してください。
- ignoreCache: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- scope: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- autoType: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。タグ属性の設定値とクラス属性値（整定数）の対応関係は、次のとおりです。
  - user 設定は、TYPE_USERSPECIFIED と等価です。
  - URI は、TYPE_URI_ONLY と等価です。
  - URI_query は、TYPE_URI_QUERYSTR と等価です。
  - URI_allParam は、TYPE_URI_ALLPARAM と等価です。

  - URI_selectedParam は、TYPE_URI_SELECTEDPARAM と等価です。
  - URI_excludedParam は、TYPE_URI_EXCLUDEDPARAM と等価です。
- selectedParam: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- selectedCookies: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- name: 明示的なキャッシュ・ブロックのネーミングによって名前を指定された 1 つ以上のキャッシュ・ブロックを無効化するには、後述の「name と invalidateNameLike の使用」の指示に従って、この属性を invalidateNameLike とともに使用します。
- invalidateNameLike: 明示的なキャッシュ・ブロックのネーミングによって名前を指定された 1 つ以上のキャッシュ・ブロックを無効化するには、後述の「name と invalidateNameLike の使用」の指示に従って、この属性を name とともに使用します。デフォルト設定は「false」です。
- page: ページ相対 URI またはアプリケーション相対 URI を指定します。暗黙的なキャッシュ・ブロックのネーミングによって名前を指定された 1 つ以上のキャッシュ・ブロックを無効化するには、後述の「page と autoInvalidateLevel の使用」の指示に従って、この属性を autoInvalidateLevel とともに使用します。
- autoInvalidateLevel: 暗黙的なキャッシュ・ブロックのネーミングによって名前を指定された 1 つ以上のキャッシュ・ブロックを無効化するには、後述の「page と autoInvalidateLevel の使用」の指示に従って、この属性を page とともに使用します。
- cacheRepositoryName: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。
- reportException: 7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

name と invalidateNameLike の使用 明示的なキャッシュ・ブロックのネーミングによって名前を指定した 1 つ以上のキャッシュ・ブロックを無効化するには、name 属性と invalidateNameLike 属性を次のように併用します。

- invalidateNameLike="false" の場合は、name パラメータを使用して、無効化する単一キャッシュ・ブロックの名前を指定します。
- invalidateNameLike="true" の場合で、基礎となるキャッシュ・リポジトリがワイルド・カード文字をサポートしている場合は、ワイルド・カード文字「*」を name パラメータに使用すると、その基準に適合した名前を持つ複数のキャッシュ・ブロックを無効化できます。（Oracle Application Server Java Object Cache では、現在ワイルド・カード文字をサポートしていません。）

page と autoInvalidateLevel の使用 暗黙的なキャッシュ・ブロックのネーミングによって名前を指定した 1 つ以上のキャッシュ・ブロックを無効化するには、page 属性と autoInvalidateLevel 属性を併用します。

page 属性を使用して、Web ページの適切な URI を指定します。暗黙的なネーミングでは、キャッシュ・ブロック名は、Web ページの URI に基づいています。

autoInvalidateLevel を使用して無効化のスコープ（application スコープ、page スコープ、parameter スコープまたは cookie スコープ）を次のように指定します。

- autoInvalidateLevel="application" の場合は、そのページが属しているアプリケーションに関連付けられているすべてのキャッシュ・ブロックが無効化されます。

  たとえば、アプリケーションが /mycontext コンテキスト・パスの下にあり、autoInvalidateLevel="application" の場合は、http://*host*:*port*/mycontext の下にあるすべてのページのキャッシュ・エントリが無効化されます。

  次に対応する使用例を示します。

```
<ojsp:invalidateCache page="/" autoInvalidateLevel="application" />
```

- autoInvalidateLevel="page" の場合は、そのページに関連付けられているすべてのキャッシュ・ブロックのエントリが無効化されます。次に例を示します。

  http://*host*:*port*/mycontext/mypage01.jsp?foo=bar

  このリクエストで autoInvalidate="page" の場合、mypage01.jsp のキャッシュ・エントリは、関連付けられているリクエスト・パラメータと Cookie に関係なく、すべて無効化されます。この中には、次の例のように関連付けられたキャッシュ・ブロックも含まれます。

  http://*host*:*port*/mycontext/mypage01.jsp?p1=v1

  次に対応する使用例を示します。

  ```
  <ojsp:invalidateCache page="/mypage01.jsp" autoInvalidateLevel="page" />
  ```

- autoInvalidateLevel="param" の場合、同一の選択済パラメータ名とその値を含むページのキャッシュ・エントリが、関連付けられている Cookie に関係なく、すべて無効化されます。

  次の例を考えてみます。

  ```
  <ojsp:invalidateCache policy="/WEB-INF/c1.cpd"
                        page="/mypage01.jsp?foo=bar"
                        autoInvalidateLevel="param" />
  ```

  この場合、次の例のように関連付けられているキャッシュ・ブロックは、無効化されません。

  http://*host*:*port*/mycontext/mypage01.jsp?foo=bar2

  ただし、次の例のように関連付けられているキャッシュ・ブロックは、関連付けられている Cookie に関係なく、無効化されます。

  http://*host*:*port*/mycontext/mypage01.jsp?foo=bar

  続いて、次の例を考えてみます。

  http://*host*:*port*/mycontext/mypage01.jsp?foo=bar&p1=v1

  このリクエストに関連付けられたキャッシュ・ブロックは、c1.cpd に foo HTTP リクエスト・パラメータのみが選択され、キャッシュ・ブロックが同一のキャッシュ・ポリシー c1.cpd に格納されている場合、無効化されます。ただし、このキャッシュ・ブロックが c1.cpd に格納されていない場合、または c1.cpd に p1 パラメータも選択されている場合は、無効化されません。

- autoInvalidateLevel="cookie" の場合、無効化されるのは、同一ページ、同一の選択済パラメータと値、および同一 Cookie に関連付けられているキャッシュ・エントリのみです。

注意： ページ URI に疑問符が含まれている場合、デフォルトの autoInvalidateLevel は、param です。疑問符が含まれていない場合、デフォルトは page です。

例 : キャッシュ無効化タグの使用

この項では、キャッシュの無効化の簡単な例を示します。

例 : invalidateCache タグ

次のページでは、以前にキャッシュされた項目リストに 1 つの項目を追加し、次にそのキャッシュを無効化します。このリストは、後で新規項目によって再度キャッシュされると想定しています。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jwcache.tld"
           prefix="ojsp" %>
<title>added.jsp</title>
<jsp:useBean class="java.util.Hashtable" id="table" scope="application" />
<%
   String itemid=request.getParameter("itemid");
   String addItem=request.getParameter("addItem");
   Vector list=(Vector) table.get(itemid);
   if (list==null) {
     list=new Vector();
     table.put(itemid,list);
   }
   list.addElement(addItem);
%>
<b><%= addItem %></b> was added into category <b><%= itemid %></b>.<br>
<% String viewPage="listitem.jsp?itemid="+itemid; %>
<% long l1=(new java.util.Date()).getTime(); %>
<ojsp:invalidateCache page="<%= viewPage %>" autoInvalidateLevel="param"
  policy="/WEB-INF/test-policy.cpd"
    />
<% long l2=(new java.util.Date()).getTime(); %>
Existing cache entry has been invalidated. <br>
Invalidation took <%= l2-l1 %> milliseconds.
<br>
<jsp:include page="<%= viewPage %>" flush="true" />
<br>
<a href="seeitems.jsp" >Select items</a>
or
<a href="additem.html" >Add items</a>
<br>
```

Web Object Cache API の説明

サーブレットから、CachePolicy メソッドを使用して、キャッシュ・ポリシー設定を変更したり、キャッシュ・ブロックを無効化できます。また、ExpirationPolicy メソッドを使用すると、期限切れ設定を変更できます。このためには、キャッシュ・ポリシー・オブジェクトの作成とその期限切れポリシー・オブジェクト属性の取得が必要です。これは JSP キャッシュ・タグ・ハンドラによって自動的に行われます。

次の各項では API について説明します。

- キャッシュ・ポリシー・オブジェクトの作成
- CachePolicy メソッド
- 期限切れポリシー・オブジェクトの取得
- ExpirationPolicy メソッド
- CacheBlock メソッド
- タグ・コードと API コードの比較

ここで説明する Web Object Cache クラスは、oracle.jsp.jwcache パッケージに含まれていて、OC4J に同梱の ojsputil.jar ファイル内にあります。このファイルがインストール済で、クラスパスに存在していることを確認してください。また、Oracle Application Server Java Object Cache をバックエンド・リポジトリとして使用するには、cache.jar ファイルがインストール済で、クラスパスに存在している必要があります。このファイルも OC4J に同梱されています。

この項で説明するクラス、インタフェースおよびメソッドの詳細は、OC4J に付属の Javadoc を参照してください。

キャッシュ・ポリシー・オブジェクトの作成

CachePolicy オブジェクトの作成には、次の 2 つのアプローチがあります。

- CacheClientUtil クラスの静的 lookupPolicy() メソッドを使用します。
- パブリック CachePolicy コンストラクタの 1 つを使用します。

> **注意：** キャッシュ・ポリシー・オブジェクトは、データベース接続やカーソルなどのリソース・オブジェクトではありません。したがって、ライフ・サイクル管理やリソース管理を考慮せずに操作できます。

lookupPolicy() メソッドの使用

通常、CachePolicy オブジェクトを作成する最も簡単な方法は、CacheClientUtil クラスの静的 lookupPolicy() メソッドを使用する方法です。次に例を示します。

```
CachePolicy cachePolicyObject = oracle.jsp.jwcache.CacheClientUtil.lookupPolicy
                                (servletConfig, request, "/WEB-INF/foo.cpd");
```

サーブレット構成オブジェクト（javax.servlet.ServletConfig インスタンス）、リクエスト・オブジェクト（javax.servlet.http.HttpServletRequest インスタンス）および XML キャッシュ・ポリシー・ディスクリプタ・ファイルの URI パス（アプリケーションのルートに相対）を入力します。

次に、キャッシュ・ポリシー・ディスクリプタ・ファイルの簡単な例を示します。

```
<!--
test-policy.cpd
-->

<cachePolicy scope="application">
   <expirationPolicy expirationType="TTL" TTL="25" timeInaDay="00:10:00"
                     writeThrough="true" />
</cachePolicy>
```

詳細は、7-39 ページの「キャッシュ・ポリシー・ディスクリプタ」を参照してください。

CachePolicy コンストラクタの使用

CachePolicy クラスには、3 つのパブリック・コンストラクタがあります。サーブレット構成オブジェクトのみを必要とする単純コンストラクタ、別の CachePolicy オブジェクトをコピーする「copy」コンストラクタおよび特定のサーブレット構成オブジェクトを含む「copy」コンストラクタです。

```
public CachePolicy(javax.servlet.ServletConfig config)

public CachePolicy(CachePolicy cPolicy)

public CachePolicy(javax.servlet.ServletConfig config,
                   CachePolicy cPolicy)
```

CachePolicy メソッド

CachePolicy オブジェクトでは、複数のユーティリティ・メソッドおよび主要属性の getter メソッドと setter メソッドを使用できます。

CachePolicy メソッド・シグネチャと共通のパラメータ

次の短縮コードには、CachePolicy オブジェクトで使用可能な主要メソッドのシグネチャが含まれています。ただし、このコードは説明用のコードです。

関連属性の説明は、7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

```
class CachePolicy
{
   boolean isRecent(CacheBlock block);
   void putCache(Object data, HttpServletRequest req, SectionId sectionId);
   void putCache(Object data, HttpServletRequest req, String specifiedName);
   void putAutoCacheForOtherPath(Object data, HttpServletRequest req,
                       String otherPath, StringSectionid sectionId);
   void putAutoCacheForOtherPath(Object data, HttpServletRequest req,
        String otherPath, Cookie[] newCookies, StringSectionid sectionId);
   CacheBlock getCache(HttpServletRequest req, SectionId sectionId);
   CacheBlock getCache(HttpServletRequest req, String specifiedName);
   CacheBlock getAutoCacheForOtherPath(HttpServletRequest req,
                         String otherPath, StringSectionId sectionId);
   CacheBlock getAutoCacheForOtherPath(HttpServletRequest req,
        String otherPath, Cookie[] newCookies, StringSectionId sectionId);
   void invalidateCache(HttpServletRequest req, SectionId sectionId);
   void invalidateCache(HttpServletRequest req, String specifiedName);
   void invalidateCacheLike(HttpServletRequest req, String specifiedName);
   void invalidateCacheLike(HttpServletRequest req, int autoInvalidateLevel);
   void invalidateCacheLike(HttpServletRequest req, String specifiedName,
                           int autoInvalidateLevel);
   void invalidateCacheOtherPathLike(HttpServletRequest req, String otherPath);
   void invalidateCacheOtherPathLike(HttpServletRequest req, String otherPath,
                         Cookie[] newCookies, int autoInvalidateLevel);
   Date getCurrentTime();
}
```

これらのメソッドでは、次の複数の共通パラメータが使用されています。

- req、javax.servlet.http.HttpServletRequest インスタンス

  これは、現行の HTTP リクエスト・オブジェクトです。

- newCookies、javax.servlet.http.Cookie[] 配列

  これは、新規 Cookie の配列です。新規 Cookie を渡すと、それらの Cookie は、otherPath パラメータを使用するキャッシュ操作（putAutoCacheForOtherPath() メソッドなど）で使用されます。その場合は、キャッシュ・ポリシーがいくつかの Cookie を選択し、無効化が Cookie レベルで行われることが前提となります。新規 Cookie を渡さない場合は、現行の HTTP リクエストの Cookie がかわりに使用されます。

- specifiedName、Java 文字列

  明示的なキャッシュ・ブロックのネーミングでは、これはブロック名を指します。つまり、新規キャッシュ・ブロックを作成している場合は任意のキャッシュ・ブロック名、既存のキャッシュ・ブロックを取得している場合はその既存キャッシュ・ブロックの名前です。

- sectionId、oracle.jsp.jwcache.SectionId インスタンス（具体的には、StringSectionId または NumberSectionId インスタンス）

  暗黙的なキャッシュ・ブロックのネーミングの場合、これはキャッシュ・ブロックの追跡に使用されるカウンタです。JSP ページでは、JSP キャッシュ・タグ・ハンドラによって使用、増分およびメンテナンスされます。JSP pageContext オブジェクトに格納されます。

SectionId は、2 つのクラス（StringSectionId と NumberSectionId）によって実装されるインタフェースです。StringSectionId をメソッド・シグネチャに指定する場合は、そのクラスのインスタンスを使用する必要があります。SectionId を指定する場合は、どちらのクラスのインスタンスも使用できます。ただし、通常は、StringSectionId を使用してください。NumberSectionId クラスは、主として JSP タグ・ハンドラが使用する目的で作成されています。

サーブレットでは、セクション ID インスタンスを手動で作成する必要があります。StringSectionId インスタンスの使用については、7-37 ページの「サーブレット・ページ : DemoCacheServlet.java」で詳しく説明します。

注意： StringSectionId インスタンスを構成する場合は、文字列をアルファベット（数値ではなく）文字で開始する必要があります。

- otherPath、Java 文字列

  格納、取得または無効化の対象となる関連付けられたキャッシュ・ブロックを持つ別の JSP ページの URI です。

- autoInvalidateLevel、整数

  暗黙的なキャッシュ・ブロックのネーミングで使用すると、無効化レベル（application、page、parameter または cookie）を指定できます。CachePolicy 整定数の AUTO_INVALIDATE_APP_LEVEL、AUTO_INVALIDATE_PAGE_LEVEL、AUTO_INVALIDATE_PARAM_LEVEL または AUTO_INVALIDATE_COOKIE_LEVEL を使用します。

CachePolicy メソッドの説明

CachePolicy メソッドは、次のように機能します。

- isRecent()

  このメソッドは、指定したキャッシュ・ブロックのタイムスタンプをチェックし、その内容が再利用可能か判断します。判断材料として現行の時間とキャッシュ・ポリシーの reusableTimeStamp 属性と reusableDeltaTime 属性に値を使用します。

- putCache(data)

  このメソッドを使用してオブジェクトをキャッシュ・リポジトリに配置します。data パラメータは、これ以上の変更が不要である、キャッシュ対象のシリアライズ可能な Java オブジェクトです。JSP ページでは、JSP cache タグ・ハンドラが putCache() をコールして、BodyContent インスタンスをキャッシュします。cacheXMLObj タグ・ハンドラは、このメソッドをコールして、XML DOM オブジェクトをキャッシュします。サーブレットまたは useCacheObj タグでは、キャッシュのターゲット・オブジェクトは、任意の Java のシリアライズ可能オブジェクトとなります。

  HTTP リクエスト・オブジェクトとキャッシュ・ブロック名（明示的なネーミングの場合）、またはセクション ID（暗黙的なネーミングの場合）も指定する必要があります。

注意： putCache() メソッドは、キャッシュ・ポリシーの ignoreCache 属性が「true」の場合、何も実行しません。

- putAutoCacheForOtherPath(...)

  指定した文字列ベースのセクション ID とページ・パスに基づき、（指定した Cookie も必要に応じて使用して）指定のオブジェクトをキャッシュ・リポジトリに配置します。HttpServletRequest オブジェクトも入力する必要があります。キャッシュ・ポリシーには、明示的なネーミングは使用できません（つまり、autoType=TYPE_USERSPECIFIED を指定できません）。

- `getCache(...)`

  このメソッドを使用して、リポジトリからキャッシュ内の項目を、`CacheBlock` インスタンスの形式で取得します。キャッシュ・ブロック名（明示的なネーミングの場合）またはセクション ID（暗黙的なネーミングの場合）を指定できます。HTTP リクエスト・オブジェクトも入力する必要があります。

  注意： `getCache()` メソッドは、キャッシュ・ポリシーの `ignoreCache` 属性が `true` の場合、何も実行しません。

- `getAutoCacheForOtherPath(...)`

  指定した文字列ベースのセクション ID とページ・パスに基づき、（指定した Cookie も必要に応じて使用して）キャッシュ内の項目をリポジトリから取得します。`HttpServletRequest` オブジェクトも入力する必要があります。キャッシュ・ポリシーに明示的なネーミングを使用すると、例外が発生します。（つまり、`autoType=TYPE_USERSPECIFIED` は指定できません。）

- `invalidateCache(...)`

  このメソッドを使用して、単一のキャッシュ・ブロックを無効化します。無効化は、HTTP リクエスト・オブジェクトと指定したキャッシュ・ブロック名（明示的なネーミングの場合）またはセクション ID（暗黙的なネーミングの場合）に基づいて行われます。

- `invalidateCacheLike(...)`

  このメソッドを使用して、複数のキャッシュ・ブロックを無効化します。明示的なキャッシュ・ブロックのネーミングを使用し、キャッシュ・リポジトリがワイルド・カードのネーミングをサポートしている場合は、`specifiedName` パラメータをワイルド・カード文字「*」付きで入力できます。Oracle Application Server Java Object Cache では、現在ワイルド・カード文字をサポートしていません。

  暗黙的なキャッシュ・ブロックのネーミングを使用する場合は、`autoInvalidateLevel` パラメータを指定して、`HttpServletRequest` オブジェクトとオプションの `specifiedName` パラメータを組み合せ、無効化するキャッシュ・ブロックを決定する必要があります。`autoInvalidateLevel` パラメータには、7-24 ページの「Web Object Cache の invalidateCache タグ」で説明したように、JSP `invalidateCache` タグと同じ機能があります（つまり、`invalidateCache` タグの `page` パラメータからの情報ではなく、リクエスト・オブジェクトからの情報を使用します）。

- `invalidateCacheOtherPathLike(...)`

  このメソッドを使用して、`otherPath` パラメータに指定した URI に関連付けられているキャッシュ・ブロックを無効化します。リクエスト・オブジェクトと URI のみを取得するシグネチャでは、`autoInvalidateLevel` パラメータが、URI に基づいて自動的に設定されます。URI に疑問符（?）がある場合は `param` レベル、それ以外の場合は `page` レベルに設定されます。

  このメソッドの詳細シグネチャによって、`autoInvalidateLevel` の設定と無効化に使用する Cookie を明示的に制御できます。

- `getCurrentTime()`

  このキャッシュ・ポリシーに指定されている基礎となるキャッシュ・リポジトリの現行の時間値を `java.util.Date` インスタンスとして取得します。

CachePolicy の getter メソッドと setter メソッド

次のメソッドを使用すると、CachePolicy オブジェクトの属性を取得または変更できます。これらの属性の説明は、7-8 ページの「キャッシュ・ポリシーの属性」を参照してください。

- `boolean getIgnoreCache()`
- `void setIgnoreCache(boolean ignoreCache)`
- `void setIgnoreCache(String ignoreCacheStr)`
- `int getScope()`
- `void setScope(int scope)`

  scope の値には、整定数の SCOPE_APP と SCOPE_SESSION を使用します。
- `int getAutoType()`
- `void setAutoType(int autoType)`

  autoType の値には、整定数の TYPE_USERSPECIFIED、TYPE_URI_ONLY、TYPE_URI_QUERYSTR、TYPE_URI_ALLPARAM、TYPE_URI_SELECTEDPARAM および TYPE_URI_EXCLUDEDPARAM を使用します。
- `String[] getSelectedParam()`
- `void setSelectedParam(String[] selectedParameters)`
- `void setSelectedParam(String selectedParamStr)`
- `String[] getSelectedCookies()`
- `void setSelectedCookies(String[] selectedCookies)`
- `void setSelectedCookies(String selectedCookiesStr)`
- `Date getReusableTimeStamp()`
- `void setReusableTimeStamp(Date reusableTimeStamp)`
- `void setReusableTimeStamp(long reusableTimeStamp)`

  reusableTimeStamp の値で、整定数の REUSABLE_ALWAYS は、そのキャッシュが常に再利用可能であることを示します。
- `long getReusableDeltaTime()`
- `void setReusableDeltaTime(long reusableDeltaTime)`

  reusableDeltaTime の値で、整定数 REUSABLE_ALWAYS は、そのキャッシュが常に再利用可能であることを示します。
- `ExpirationPolicy getExpirationPolicy()`
- `void setExpirationPolicy(ExpirationPolicy expirationPolicy)`
- `String getCacheRepositoryName()`
- `void setCacheRepositoryName(String repoName)`
- `boolean getReportException()`
- `void setReportException (boolean reportException)`
- `void setReportException (String reportExceptionStr)`

次のメソッドも使用可能です。ただし、主に Web Object Cache タグ・ハンドラによって使用されます。

- `void setScope(String scopeStr)`

  `scope` の値には、文字列定数の `SCOPE_APP_STR` と `SCOPE_SESSION_STR` があります。

- `void setAutoType(String autoTypeStr)`
- `void setReusableTimeStamp(String reusableTimeStampStr)`

  `reusableTimeStamp` の値で、文字列定数 `REUSABLE_IGNORED` は、そのキャッシュが常に再利用可能であることを示します。

- `void setReusableDeltaTime(String reusableDeltaTimeStr)`

  `reusableDeltaTime` の値で、文字列定数 `REUSABLE_IGNORED` は、そのキャッシュが常に再利用可能であることを示します。

期限切れポリシー・オブジェクトの取得

各 `CachePolicy` オブジェクトには、`ExpirationPolicy` 属性があります。キャッシュ・ブロックに期限切れポリシーを設定する場合は、`CachePolicy` オブジェクトの `getExpirationPolicy()` メソッドを使用できます。次に例を示します。

```
CachePolicy cachePolicyObj = CacheClientUtil.lookupPolicy
                             (config, request, "/WEB-INF/mypolicy.cpd");
ExpirationPolicy expPolicyObj = cachePolicyObj.getExpirationPolicy();
```

ExpirationPolicy メソッド

`ExpirationPolicy` クラスの属性には、次の getter メソッドと setter メソッドがあります。これらの属性の説明は、7-13 ページの「期限切れポリシーの属性」を参照してください。

- `int getExpirationType()`
- `void setExpirationType(int expirationType)`
- `void setExpirationType(String expirationTypeStr)`
- `long getTTL()`
- `void setTTL(long ttl)`
- `long getTimeInaDay()`
- `void setTimeInaDay(long timeInaDay)`
- `void setTimeInaDay(String timeInaDayStr)`
- `int getDayInaWeek()`
- `void setDayInaWeek(int dayInaWeek)`
- `void setDayInaWeek(String dayInaWeekStr)`
- `int getDayInaMonth()`
- `void setDayInaMonth(int dayInaMonth)`
- `boolean getWriteThrough()`
- `void setWriteThrough(boolean writeThrough)`
- `void setWriteThrough(String writeThroughStr)`

さらに、ExpirationPolicy クラスには、次のユーティリティ・メソッドがあります。

- long getExpirationTime(long createTime)

  たとえば、キャッシュ・ブロックの作成時間が 1970 年 1 月 1 日午前 0 時を基点としたミリ秒で表される場合、このメソッドが計算して戻す期限切れ時間も、1970 年 1 月 1 日午前 0 時を基点としたミリ秒で表されます。つまり、期限切れのタイムスタンプは、期限切れポリシーに従います。

ExpirationPolicy クラスは、expirationType 属性に対して、次の整定数を定義します。

- TYPE_TTL
- TYPE_DAILY
- TYPE_WEEKLY
- TYPE_MONTHLY

次の整定数は、dayInaWeek 属性に対して定義されます。

- WEEKLY_SUNDAY
- WEEKLY_MONDAY
- WEEKLY_TUESDAY
- WEEKLY_WEDNESDAY
- WEEKLY_THURSDAY
- WEEKLY_FRIDAY
- WEEKLY_SATURDAY

CacheBlock メソッド

CachePolicy オブジェクトの getCache() メソッドを使用すると、関連する CacheBlock オブジェクトを取得できます。詳細は、7-29 ページの「CachePolicy メソッド」と 7-37 ページの「サーブレット・ページ : DemoCacheServlet.java」で説明します。

次の短縮コードは、oracle.jsp.jwcache.CacheBlock クラスの主要メソッドを示しています。ただし、このコードは説明用のコードです。

```
class CacheBlock
{
   long getCreationTime();
   long getExpirationTime();
   Serializable getData();
}
```

次に、これらのメソッドを簡単に説明します。

- getCreationTime(): キャッシュ・ブロックの作成時間を示すタイムスタンプを戻します。
- getExpirationTime(): キャッシュ・ブロックの期限切れ時間を示すタイムスタンプを戻します。
- getData(): キャッシュ・ブロック・データを戻します。

注意 : 作成時間と期限切れ時間は、1970 年 1 月 1 日午前 0 時を基点とするミリ秒で表されます。

タグ・コードと API コードの比較

次の例では、2 つのキャッシュ・フラグメントからタイムスタンプ出力をキャッシュし、表示するアプリケーションに対する 3 つのアプローチのコードを示します。

- 最初のアプローチ tagcode.jsp は、Oracle Web Object Cache のタグを使用する簡単な JSP ページです。
- 第 2 のアプローチ servletcode.jsp は、Web Object Cache タグを使用するかわりに Java スクリプトレット内で Web Object Cache API を使用するより複雑な JSP ページです。
- 第 3 のアプローチ DemoCacheServlet.java は、サーブレット内で Web Object Cache API を使用します。

3 つのコード・サンプルの後に、キャッシュ・ポリシー・ディスクリプタ test-policy.cpd を示します。

各アプローチで、アプリケーションは、表示する 2 つのフラグメントをキャッシュします。再ロードを何回行っても、フラグメントに表示される時間は、キャッシュ内のフラグメントが期限切れになるまで変わりません。最初のフラグメントは、キャッシュ・ポリシー・ディスクリプタ（test-policy.cpd）から 25 秒の TTL 値を取得しているため、期限切れまで 25 秒かかります。第 2 のフラグメントはキャッシュ・ポリシー・ディスクリプタの TTL 値を、ページ・コードに直接設定されている値でオーバーライドしているため、期限切れまで 15 秒かかります。

サンプル・アプリケーションの出力は、次のようになります。

```
fragment#1  (expires in 25 seconds based on TTL value test-policy)
Sun May 27 15:20:46 PDT 2001

fragment#2 (expires in 15 seconds because TTL overrides test-policy value)
Sun May 27 15:20:46 PDT 2001
```

単純な JSP ページ : tagcode.jsp

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jwcache.tld"
           prefix="ojsp" %>
<title>tagcode.jsp</title>
<pre>
tagcode.jsp
<ojsp:cache policy="/WEB-INF/test-policy.cpd" >
  fragment#1  (expires in 25 seconds based on TTL value test-policy)
  <%= new java.util.Date() %>
</ojsp:cache>
<ojsp:cache policy="/WEB-INF/test-policy.cpd" TTL="15" >
  fragment#2 (expires in 15 seconds because TTL overrides test-policy value)
  <%= new java.util.Date() %>
</ojsp:cache>
</pre>
```

スクリプトレット JSP ページ : servletcode.jsp

コード・ノートは、次項の「サーブレット・ページ : DemoCacheServlet.java」のサーブレット・バージョンと同じです。

```
<%@ page import="oracle.jsp.jwcache.*,java.io.*" %>
<title>servletcode.jsp</title>
<pre>
servletcode.jsp
<%
  CachePolicy cachePolicyObj = CacheClientUtil.lookupPolicy(config,request,
                               "/WEB-INF/test-policy.cpd" );  // Note A
  StringSectionId sectionId=new StringSectionId("s1");   // Note B
  CacheBlock cacheBlockObj=null;

  cacheBlockObj = cachePolicyObj.getCache(request,sectionId); // Note C
```

```
  if (!cachePolicyObj.isRecent(cacheBlockObj)) {  // Note D
    CharArrayWriter newOut=new CharArrayWriter();
    PrintWriter pw=new PrintWriter(newOut);

    // actual logic within a cache block
    pw.println
          ("fragment#1 (expires in 25 seconds based on TTL value test-policy)");
    pw.println(new java.util.Date());
    // which generates content into the "out" object

    if (cacheBlockObj == null) {   // Note E
      cachePolicyObj.putCache(newOut.toCharArray(),request,sectionId);
      // Note F
    }

    out.write(newOut.toCharArray());
    // writing out newly created data back to the original writer
  }
  else {
    out.write((char[])cacheBlockObj.getData());
    // writing the existing cached data to the writer
  }

  sectionId=new StringSectionId("s2");
  long timeToLive = 15;   // now set TTL to 15 on this block
  ExpirationPolicy expirationPolicy = cachePolicyObj.getExpirationPolicy();
  expirationPolicy.setTTL(timeToLive);
  cachePolicyObj.setExpirationPolicy(expirationPolicy);
  cacheBlockObj = cachePolicyObj.getCache(request,sectionId);
  if (!cachePolicyObj.isRecent(cacheBlockObj)) {
    CharArrayWriter newOut=new CharArrayWriter();
    PrintWriter pw=new PrintWriter(newOut);

    // actual logic within a cache block
    pw.println
 ("fragment#2 (expires in 15 seconds because TTL overrides test-policy value)");
    pw.println(new java.util.Date());
    // which generates content into the "out" object

    if (cacheBlockObj == null) {
      cachePolicyObj.putCache(newOut.toCharArray(),request,sectionId);
    }

    out.write(newOut.toCharArray());
    // writing out newly created data back to the original writer
  }
  else {
    out.write((char[])cacheBlockObj.getData());
    // writing the existing cached data to the writer
  }

%>
</pre>
```

サーブレット・ページ : DemoCacheServlet.java

コード・ノートは、コードの末尾に説明してあります。

```
package demoPkg;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

import java.io.PrintWriter;
import java.io.CharArrayWriter;

import oracle.jsp.jwcache.CachePolicy;
import oracle.jsp.jwcache.ExpirationPolicy;
import oracle.jsp.jwcache.StringSectionId;
import oracle.jsp.jwcache.CacheBlock;
import oracle.jsp.jwcache.CacheClientUtil;

public class DemoCacheServlet extends HttpServlet{

  public void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    // standard writer object from servlet engine
    PrintWriter out=response.getWriter();
    ServletConfig config=getServletConfig();

    try {
      CachePolicy cachePolicyObj = CacheClientUtil.lookupPolicy(config,request,
        "/WEB-INF/test-policy.cpd" );  // Note A
      StringSectionId sectionId=new StringSectionId("s1");   // Note B
      CacheBlock cacheBlockObj=null;

      cacheBlockObj = cachePolicyObj.getCache(request,sectionId); // Note C
      if (!cachePolicyObj.isRecent(cacheBlockObj)) {  // Note D
        CharArrayWriter newOut=new CharArrayWriter();
        PrintWriter pw=new PrintWriter(newOut);

        // actual logic within a cache block
        pw.println("fragment#1");
        pw.println(new java.util.Date());
        // which generates content into the "out" object

        if (cacheBlockObj == null) {   // Note E
          cachePolicyObj.putCache(newOut.toCharArray(),request,sectionId);
          // Note F
        }

        out.write(newOut.toCharArray());
        // writing out newly created data back to the original writer
      }
      else {
        out.write((char[])cacheBlockObj.getData());
        // writing the existing cached data to the writer
      }

      sectionId=new StringSectionId("s2");
      long timeToLive = 15;   // now set TTL to 15 on this block
      ExpirationPolicy expirationPolicy = cachePolicyObj.getExpirationPolicy();
      expirationPolicy.setTTL(timeToLive);
      cachePolicyObj.setExpirationPolicy(expirationPolicy);
      cacheBlockObj = cachePolicyObj.getCache(request,sectionId);
      if (!cachePolicyObj.isRecent(cacheBlockObj)) {
```

```
        CharArrayWriter newOut=new CharArrayWriter();
        PrintWriter pw=new PrintWriter(newOut);

        // actual logic within a cache block
        pw.println("fragment#2");
        pw.println(new java.util.Date());
        // which generates content into the "out" object

        if (cacheBlockObj == null) {
          cachePolicyObj.putCache(newOut.toCharArray(),request,sectionId);
        }

        out.write(newOut.toCharArray());
        // writing out newly created data back to the original writer
      }
      else {
        out.write((char[])cacheBlockObj.getData());
        // writing the existing cached data to the writer
      }

    } catch (Throwable th) {
      // your exception handling code here
      th.printStackTrace(out);
    }
  }
}
```

コード・ノート 前述の例の主要機能について説明します。

- キャッシュ・ポリシー・オブジェクトは `lookupPolicy()` コール（Note A）で作成され、属性はキャッシュ・ポリシー・ディスクリプタ `test-policy.cpd` に基づいて設定されます。
- 各キャッシュ・ブロックのセクション ID は、暗黙的なキャッシュ・ブロックのネーミングで必要な場合に作成されます（Note B）。セッション ID の詳細は、7-29 ページの「CachePolicy メソッド」を参照してください。
- キャッシュ・ブロックは、キャッシュ・ポリシー・オブジェクトの `getCache()` メソッドを使用してリポジトリから取得され（Note C）、`putCache()` メソッドを使用してリポジトリに配置されます。いずれの場合もセクション ID に基づいて行われます。
- `isRecent()` コールは、そのキャッシュ・ブロックが再利用可能かどうかを判断します（Note D）。使用できる場合、そのキャッシュ内のデータは、キャッシュ・ブロックの `getData()` メソッドを使用して取得されます。（詳細は、7-34 ページの「CacheBlock メソッド」を参照してください。）使用できない場合は、出力をバッファして、キャッシュ・リポジトリに保存しなおすために、特別な `PrintWriter` オブジェクトが作成されます。キャッシュ・ブロック・オブジェクトが見つからない場合（null の場合、Note E）は、キャッシュ・ポリシー・オブジェクトの `putCache()` メソッドがコールされ、新規キャッシュ・ブロックが作成されます（Note F）。

キャッシュ・ポリシー・ディスクリプタ : test-policy.cpd

このキャッシュ・ポリシー・ディスクリプタは、サンプル・アプリケーションへの 3 つのアプローチ（`tagcode.jsp`、`servletcode.jsp` および `DemoCacheServlet.java`）のすべてで使用されます。

```
<!--
test-policy.cpd
-->

<cachePolicy scope="application">
   <expirationPolicy expirationType="TTL" TTL="25" timeInaDay="00:10:00"
                     writeThrough="true" />
</cachePolicy>
```

キャッシュ・ポリシー・ディスクリプタ

XML スタイルのキャッシュ・ポリシー・ディスクリプタを使用することによって、CachePolicy オブジェクトと ExpirationPolicy オブジェクトの属性設定を指定できます。使用する JSP ページまたはサーブレットで、cache タグ、cacheXMLObj タグ、useCacheObj タグ、cacheInclude タグまたは invalidateCache タグの policy 属性を使用してキャッシュ・ポリシー・ディスクリプタを指定します。

次の各項では、キャッシュ・ポリシー・ディスクリプタ DTD、サンプル・キャッシュ・ポリシー・ディスクリプタおよびキャッシュ・ポリシー・ディスクリプタのロードとリフレッシュに関する情報を示します。

キャッシュ・ポリシー・ディスクリプタ DTD

この項では、Web Object Cache のキャッシュ・ポリシー・ディスクリプタ DTD、cachepolicy.dtd を示します。

```
<!--
cachepolicy.dtd
-->
<!--
This DTD is used to validate any (Oracle programmable web)
cache policy descriptors (for example, "/WEB-INF/foo.cpd").
-->

<!--
The cachePolicy element is the root element of cache policy descriptors.
configuration descriptor.
-->

<!ELEMENT cachePolicy (
   selectedParam*, selectedCookie*,
   reusableTimeStamp?, reusableDeltaTime?,
   cacheRepositoryName?, expirationPolicy? ) >

<!ATTLIST cachePolicy ignoreCache (true | false) "false" >
<!ATTLIST cachePolicy scope (application | session) "application" >
<!ATTLIST cachePolicy autoType
   (user | URI | URI_query |
    URI_allParam | URI_selectedParam | URI_excludedParam )
    "URI_allParam" >
<!ATTLIST cachePolicy reportException (true | false) "true" >

<!ELEMENT selectedParam (#PCDATA) >
<!ELEMENT selectedCookie (#PCDATA) >
<!ELEMENT reusableTimeStamp (#PCDATA) >
<!ELEMENT reusableDeltaTime (#PCDATA) >
<!ELEMENT cacheRepositoryName (#PCDATA) >

<!ELEMENT expirationPolicy EMPTY >

<!ATTLIST expirationPolicy expirationType (TTL | daily | weekly | monthly)
         "TTL" >
<!ATTLIST expirationPolicy TTL CDATA "300" >
<!ATTLIST expirationPolicy timeInaDay CDATA #IMPLIED >
<!ATTLIST expirationPolicy dayInaWeek
   (Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday)
   "Wednesday" >
<!ATTLIST expirationPolicy dayInaMonth CDATA "10" >
<!ATTLIST expirationPolicy writeThrough (true | false) "true" >
```

サンプル・キャッシュ・ポリシー・ディスクリプタ

この項では、TTL 属性と timeInaDay 属性を設定する簡単なキャッシュ・ポリシー・ディスクリプタの例を示します。

```
<!--
test-policy.cpd
-->

<cachePolicy scope="application">
   <expirationPolicy expirationType="TTL" TTL="25" timeInaDay="00:10:00"
                     writeThrough="true" />
</cachePolicy>
```

キャッシュ・ポリシー・ディスクリプタのロードとリフレッシュ

XML キャッシュ・ポリシー・ディスクリプタ・ファイルから CachePolicy オブジェクトを作成するには、CacheClientUtil クラスの静的 lookupPolicy() メソッドのコールが必要です。JSP ページの場合、これは自動的に処理されます。サーブレットの場合は、コードに lookupPolicy() コールをインクルードする必要があります。7-37 ページの「サーブレット・ページ : DemoCacheServlet.java」を参照してください。

キャッシング・ポリシーが以前にロードされていない場合、lookupPolicy() の起動により、XML ディスクリプタが解析され、それを使用して新規 CachePolicy オブジェクトとその ExpirationPolicy 属性が構成されます。lookupPolicy() メソッドの詳細は、7-28 ページの「キャッシュ・ポリシー・オブジェクトの作成」を参照してください。

CachePolicy オブジェクトは、アプリケーションに関連付けられている ServletContext オブジェクトの下に間接的に格納されます。同一のキャッシング・ポリシーを再度リクエストすると、格納されているポリシー・オブジェクトが、ディスクリプタを再読取りまたは再解析することなく戻されます。セキュリティ上の理由に加えて、パフォーマンス上の理由から、キャッシュ・ポリシー・ディスクリプタ・ファイルはほとんど変更されないため、OC4J には、ディスクリプタの自動再ロード機能は備わっていません。その結果、キャッシュ・ポリシー・オブジェクトは、アクセス速度を上げるため、中間層の JVM に格納されます。

CachePolicy オブジェクトは、サーブレット・コンテキストが破棄されるまで、または CacheClientUtil クラスの静的 refreshPolicy() メソッドがコールされるまで有効です。このメソッドには、lookupPolicy() メソッドと同じコール順序があります。たとえば、次のように指定します。

```
oracle.jsp.jwcache.CacheClientUtil.refreshPolicy
                    (servletConfig, request, "/WEB-INF/foo.cpd");
```

キャッシング・ポリシーを変更およびリフレッシュする場合、アクティブなキャッシュ・ブロックは影響を受けません。

キャッシュ・リポジトリ・ディスクリプタ

XML スタイルのキャッシュ・リポジトリ・ディスクリプタを使用して、Web Object Cache のバックエンド・キャッシュ・リポジトリとして使用するリポジトリとその構成方法を指定します。次の各項では、キャッシュ・リポジトリ・ディスクリプタの DTD とサンプル・キャッシュ・リポジトリ・ディスクリプタを示します。

- キャッシュ・リポジトリ・ディスクリプタ DTD
- サンプル・キャッシュ・リポジトリ・ディスクリプタ

注意 : デフォルトでは、Web Object Cache は Oracle Application Server Java Object Cache をキャッシュ・リポジトリとして使用します。

キャッシュ・リポジトリ・ディスクリプタ DTD

この項では、Web Object Cache のキャッシュ・リポジトリ・ディスクリプタ DTD、wcache.dtd を示します。

```
<!--
Copyright 2000 Oracle Corporation
wcache.dtd
-->
<!--
This DTD is used to validate "/WEB-INF/wcache.xml", which is used to hold
web cache repositories configuration information for
Oracle programmable web caching components.
-->

<!--
The wcache-config element is the root element of web cache repositories
configuration descriptor.
-->

<!ELEMENT wcache-config (cache-repository*)>

<!ELEMENT cache-repository (cache-repository-name,cache-repository-class,init-param*)>

<!ELEMENT cache-repository-name (#PCDATA)>
<!ELEMENT cache-repository-class (#PCDATA)>

<!ELEMENT init-param (param-name,param-value)>
<!ELEMENT param-name (#PCDATA)>
<!ELEMENT param-value (#PCDATA)>
```

サンプル・キャッシュ・リポジトリ・ディスクリプタ

この項では、OC4J が提供するキャッシュ・リポジトリ・ディスクリプタを示します。

> **注意：** DTD には、reporoot が含まれていません。これは、ファイル・システムのキャッシュ実装にのみ使用する特殊なパラメータです。

```
<wcache-config>

<cache-repository>
   <cache-repository-name>DefaultCacheRepository</cache-repository-name>
   <cache-repository-class>
      oracle.jsp.jwcache.repository.impl.OCSRepoImpl
   </cache-repository-class>
</cache-repository>

<cache-repository>
   <cache-repository-name>SimpleFSRepo</cache-repository-name>
   <cache-repository-class>
      oracle.jsp.jwcache.repository.impl.SimpleFSRepositoryImpl
   </cache-repository-class>
   <init-param>
      <param-name>reporoot</param-name>
      <param-value>/tmp/reporoot</param-value>
   </init-param>
</cache-repository>

</wcache-config>
```

バックエンド・リポジトリの構成

この項では、Oracle Application Server Java Object Cache またはファイル・システムを OC4J Web Object Cache のバックエンド・リポジトリとして構成する方法について説明します。

Oracle Application Server Java Object Cache の構成上の注意

OC4J の server.xml ファイルでは、<javacache-config> 要素で Java Object Cache 構成ファイルが指定されている必要があります。これは <application-server> 要素のサブ要素です。デフォルトのエントリを次に示します。

```
<application-server ... >
...
   <javacache-config path="../../../javacache/admin/javacache.xml" />
...
</application-server>
```

このデフォルト・エントリで、デフォルト構成ファイルのディレクトリ位置（server.xml の格納場所）を想定すると、デフォルトで OC4J インスタンスは *ORACLE_HOME*/javacache/admin ディレクトリにある同じ Java Object Cache 構成ファイル javacache.xml を共有します。

次に Java Object Cache 構成ファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-configuration
 xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
   <logging>
     <location>javacache.log</location>
     <level>ERROR</level>
   </logging>
   <communication>
     <isDistributed>true</isDistributed>
     <coordinator discovery-port="7000"/>
   </communication>
   <persistence>
     <location>diskcache</location>
     <disksize>32</disksize>
   </persistence>
   <max-objects>1000</max-objects>
   <max-size>48</max-size>
   <clean-interval>30</clean-interval>
</cache-configuration>
```

Java Object Cache、その構成および javacache.xml ファイルの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。server.xml ファイルの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

ファイル・システム・キャッシュの構成上の注意

ファイル・システムをバックエンド・リポジトリとして使用する場合は、キャッシュ・リポジトリ・ディスクリプタ（wcache.xml）を編集して reporoot を設定し、ファイル・システム・キャッシュのルート・ディレクトリを指定します。このファイルは、OC4J サンプルがインストールされている /WEB-INF ディレクトリにあります。reporoot 値を設定するキャッシュ・リポジトリ・ディスクリプタの詳細と例は、7-40 ページの「キャッシュ・リポジトリ・ディスクリプタ」を参照してください。

次に、UNIX システムの例を示します。

```
<init-param>
   <param-name>reporoot</param-name>
   <param-value>/mydir/repositoryroot</param-value>
</init-param>
```

次に、Windows システムの例を示します。

```
<init-param>
   <param-name>reporoot</param-name>
   <param-value>c:¥mydir¥repositoryroot</param-value>
</init-param>
```

8

ファイル・アクセスとメール用の Bean とタグ

ファイルへのアクセス（アップロードとダウンロード）および電子メール用の OC4J タグと JavaBeans について説明します。電子メール用のタグと JavaBeans によって、添付にファイル・アクセス機能が使用されます。この章の構成は、次のとおりです。

ファイル・アクセス JavaBeans とファイル・アクセス・タグ

OC4J には、JSP ページとサーブレットに便利なファイルのアップロード機能とダウンロード機能を追加する、標準準拠のタグ・ライブラリと JavaBeans が用意されています。ファイルは、ファイル・システムやデータベースとの間でアップロードしたりダウンロードできます。

次の各項では、ファイル・アクセス・タグとファイル・アクセス Bean について説明します。

OC4J のファイル・アクセス機能の概要

開発者は、カスタム・タグまたは JavaBeans のいずれかを必要に応じて使用し、ユーザーによるファイルのアップロードまたはダウンロードを可能にするアプリケーションをプログラミングできます。いずれの場合も、アプリケーションは、アップロードの場合はクライアント・システム上のファイルの元の場所またはダウンロードの場合はクライアント・システム上のダウンロード先の場所を、ユーザーがブラウザを介して指定するようにプログラミングされます。アップロード用の JSP ページについて、OC4J には、ファイルの元の場所を指定する際に使用するフォームを作成する convenience タグの `httpUploadForm` が用意されています。

接続先のファイル・システムまたはデータベースの場所の指定を含め、アップロードの処理には、`HttpUploadBean` JavaBean または `httpUpload` タグを使用します。ソース・ファイル・システムまたはデータベースの場所の指定を含め、ダウンロードの処理には、`HttpDownloadBean` または `httpDownload` タグを使用します。Bean は、`HttpFileAccessBean` を拡張します。ただし、これは、公開使用を目的としていません。すべての Bean は、`oracle.jsp.webutil.fileaccess` パッケージにあります。

ファイル・アップロードの概要

ユーザーがアップロードするファイルの場所を JSP ページに指定するためのフォームは、`httpUploadForm` タグを使用して作成できます。このタグによって、ユーザーはアップロードするファイルを選択し、必要な複数パートの HTTP リクエストを作成できます。標準の HTML フォームを使用しても、リクエストを作成できます。

`HttpUploadBean` JavaBean または `httpUpload` タグを使用して、複数の部分に分割されたフォームでエンコードされたデータ・ストリームを受信および処理し、ファイル・システムまたはデータベースのいずれかの適切な場所にそのファイルを書き込みます。ターゲット・ファイルまたはデータベース行がすでに存在する場合、前のデータを上書きするかどうかを決定できる機能があります。

注意： アップロードするファイルの最大サイズは、2GB です。

ファイル・システムの接続先 接続先がファイル・システム内の場合は、ベース・ディレクトリを指定するプロパティ・ファイルを用意する必要があります。プロパティ・ファイルは、`fileaccess.properties` という名前でアプリケーションの `/WEB-INF` ディレクトリに配置し、さらに、ディレクトリへの絶対パスを示す `fileaccess.basedir` エントリが存在している必要があります。次に例を示します。

```
fileaccess.basedir=/tmp
```

注意： Windows システムでも、ディレクトリ・パスには円記号ではなくスラッシュを使用する必要があります。

```
fileaccess.basedir=c:/tmp
```

さらに、OC4J がインストールされているドライブと同じドライブ（この場合は `c:`）を指定する必要があります。

ベース・ディレクトリの下には、サブディレクトリ（各認可ユーザーのサブディレクトリなど）を必要に応じて置く必要があります。ベース・ディレクトリの下の接続先サブディレクトリは、アップロード Bean またはアップロード・タグの属性を使用して指定する必要があります。すべてのディレクトリとサブディレクトリが存在し、書込み可能な状態である必要があります。OC4J の機能を使用して、作成したり書込み可能にすることはできません。

データベースの接続先 接続先がデータベース内の場合は、デフォルトの表 `fileaccess` を必要に応じて使用できます。この表は、付属の `fileaccess.sql` スクリプトを使用して作成します。あるいは、必要な列型が含まれた他の既存の表を利用できます。いずれの場合も、データベースへの接続を `oracle.jsp.dbutil.ConnBean` または標準の `java.sql.Connection` のインスタンスとして指定する必要があります。`ConnBean` インスタンスを明示的に指定するか、または JSP ページで `dbOpen` タグ内に `httpUpload` タグをネストした結果として暗黙的に提供することもできます。（`ConnBean` JavaBean と `dbOpen` タグの詳細は、第 4 章「データ・アクセス JavaBeans とデータ・アクセス・タグ」を参照してください。）

注意： 現在、`java.sql.Connection` 型がサポートされるのはファイル・アクセス Bean の場合のみで、タグの場合はサポートされません。

さらに、アップロード Bean またはアップロード・タグの属性を使用して接続先を指定する必要があります。接続先は、データベース表の接頭辞列に配置される Java 文字列の値です。接頭辞は、ファイル・システム・パスと同じです。

ファイル・データは、BLOB または CLOB としてデータベースに書き込まれます。どちらにするかは、アップロード Bean またはアップロード・タグ属性を使用して指定できます。

デフォルトの `fileaccess` 表を使用しない場合は、アップロード Bean またはアップロード・タグの属性を使用して、データベース表の名前およびファイル・データ、ファイル接頭辞およびファイル名を格納する列の名前を指定する必要があります。それ以外の表を使用する場合は、次のように、`fileaccess` のパターンに準拠する必要があります。

- ファイル名を保持する列と接頭辞を保持する列で構成された一意の連結キーが必要です。
- ファイル・データに対する BLOB または CLOB の列が必要です。
- ファイル・データ列以外のすべての列で null データが使用可能である必要があります。

注意： `ConnBean` インスタンスを使用する場合は、接続を起動する `jsp:useBean` タグに指定したスコープの最後で、接続が自動的にクローズされます。`Connection` インスタンスのような機能はありません。

アップロードに関するセキュリティ上の考慮事項 データベースへのアップロードの場合、データベース表には、指定のファイルに対する特定の認可ユーザーを示す列がありません。したがって、すべてのユーザーは、ファイルの接頭辞を知らなくても、他のユーザーがアップロードしたファイルを警告なしに参照できます。これを防止するために、各接頭辞に適切なユーザー名を付加できます。

ファイル・ダウンロードの概要

HttpDownloadBean JavaBean または httpDownload タグは、次のように使用します。

- ユーザーが、ファイルの検索で一致させるファイル・システムのソース・ディレクトリまたはデータベースの接頭辞を指定できるようにします。

  次の点に注意してください。

  - データベースからのダウンロードでは、接頭辞の一致に大 / 小文字区別が適用されます。
  - ファイル・システムからのダウンロードでは、大 / 小文字区別のある UNIX などのオペレーティング・システムの場合はソース・ディレクトリの一致に大 / 小文字区別が適用されます。
  - 現在、ファイル名の指定は、部分名または完全名のいずれもサポートされていません。

- ダウンロード可能なファイルのリストを取得して表示します。

  使用可能なファイルのリストが表示された後、ユーザーはそのリストから一度に 1 つのファイルをダウンロードできます。

再帰的なダウンロードを行うかどうかを指定する機能もあります。このダウンロードでは、サブディレクトリ内のファイルまたはデータベース接頭辞の追加情報を含むファイルも、ダウンロード可能になります。データベースのダウンロードの場合の接頭辞はファイル・システム・パスと同じで、ファイルを階層にグループ化するために使用できます。データベースからの再帰的なダウンロードの例として、接頭辞に /user を指定したと仮定します。再帰的なダウンロードでは、「/user」で始まる接頭辞を持つファイルとの一致を検出します。「/user/bill」や「/user/mary」、「/user1」、「/user2」、「/user1/tom」および「/user2/susan」などがその例です。

ファイル・システムからのファイルのダウンロードでは、8-2 ページの「ファイル・アップロードの概要」に説明されている機能を利用します。つまり、fileaccess.properties ファイルを使用してベース・ディレクトリを指定し、ダウンロード Bean またはダウンロード・タグ内の属性を使用してファイル・パスの残りの部分を指定します。

データベースからのファイルのダウンロードでは、データベースへのアップロードと同様に、oracle.jsp.dbutil.ConnBean または java.sql.Connection のインスタンスを指定する必要があります。さらに、デフォルトの fileaccess 表（付属の fileaccess.sql スクリプトを使用して作成可能）を使用していない場合は、データベースの表や列に関する必要な情報をすべて指定する必要があります。この情報は、ダウンロード Bean またはダウンロード・タグの属性を使用して指定します。

ファイルの実際のダウンロードは、OC4J が提供する DownloadServlet によって実行されます。ダウンロード・タグの使用時に、タグ属性を使用してこのサーブレットのパスを指定します。ファイル・システム・ソースの場合は、このサーブレットへのハイパーリンクが自動的に作成されます。このため、ユーザーは各ファイルのリンクを選択して、そのファイルをダウンロードできます。データベース・ソースの場合、サーブレットはファイル・コンテンツを構成する選択済の CLOB または BLOB データをフェッチします。（詳細は、8-12 ページの「ダウンロード・サーブレット」を参照してください。）

ダウンロードに関するセキュリティ上の考慮事項 ダウンロードの場合は、ユーザーがソース（サーバー・サイド）・ファイル・システムまたはデータベースの内容を参照できる機能に制限を設けることを検討してください。警告がない場合は、次のような例が発生する可能性があります。

- ファイル・システムのダウンロードの場合、ソース値「*」（ユーザー入力で指定される可能性がある）によって、ベース・ディレクトリの下にあるすべてのディレクトリがダウンロードの対象になり、ユーザーが選択できるように、すべてのファイルの名前が表示されます。
- データベースからの再帰的なダウンロードの場合は、source 文字列（ユーザー入力で指定される可能性がある）で始まる接頭辞を持つすべてのファイルがダウンロードの対象となり、ファイル名がすべて表示されます。「*」のソースは、すべての接頭辞に一致します。

これが心配な場合は、次のような保護手段が考えられます。

- ファイル・システムからのダウンロード時に、source 値「*」を受け入れない方法。
- データベースからの再帰的なダウンロードを許可しない方法。
- 部分ディレクトリ・パスまたは接頭辞文字列に、ユーザー名などの source 値を自動的に付加して、ユーザーがアクセスできる領域を制限する方法。

ファイル・アップロードおよびダウンロード用 JavaBean とクラスの説明

この項では、OC4J が提供するファイル・アップロードおよびダウンロード用 JavaBeans（それぞれ、HttpUploadBean と HttpDownloadBean）の属性とメソッドについて説明します。

OC4J が提供する、ファイルのダウンロードを実際に実行する DownloadServlet、およびファイルのアップロードとダウンロードに関連する例外でファイル・アクセス JavaBeans が使用する FileAccessException クラスについても簡単に説明します。

JavaBean の仕様に準拠して、ファイル・アップロードおよびダウンロード用 JavaBeans には、引数なしのコンストラクタがあります。

注意： ファイル・アップロードおよびダウンロード用 JavaBeans を使用するには、ojsputil.jar ファイルがインストール済で、クラスパスに存在していることを確認します。このファイルは、OC4J に含まれています。

HttpUploadBean

oracle.jsp.webutil.fileaccess.HttpUploadBean JavaBean には、アップロードに使用する情報を指定するための setter メソッドが多数あります。また対応するほとんどの getter メソッドも含まれています。必須属性と適切な属性をすべて設定した後、upload() メソッドを使用してアップロードを実行します。アップロードされたファイルの名前を表示するメソッドもあります。そのため通常は、情報を含むメッセージをブラウザに提供できます。

HttpUploadBean は、HttpDownloadBean と同様に、HttpFileAccessBean を拡張します。ただし、これ自体は、公開使用を目的としていません。

関連情報は、8-2 ページの「ファイル・アップロードの概要」を参照してください。

必須属性のサマリー

次に、HttpUploadBean の必須属性のサマリーを示します。

- 常に必須：destination
- データベースへのアップロードに必須：destinationType、connection
- デフォルトの fileaccess 表以外のデータベース表へのアップロードに必須：table、prefixColumn、fileNameColumn、dataColumn
- ファイル・データに CLOB 列を使用しているデータベース表へのアップロードに必須：fileType

さらに、ファイル・システムへのアップロードでは、setBaseDir() メソッドをコールして、サーブレット・コンテキストと HTTP リクエスト・オブジェクトを指定する必要があります。その結果、Bean はベース・ディレクトリを指定する fileaccess.properties ファイルを検出できます。

メソッド

次に、HttpUploadBean のパブリック・メソッドを説明します。

注意： HttpUploadBean の属性と setter メソッドの多くは、HttpDownloadBean の場合と同じです。

- `void upload(javax.servlet.http.HttpServletRequest req)`
  `throws FileAccessException`

  必須の Bean 属性と適切な Bean 属性をすべて設定した後、このメソッドを使用してアップロードします。req パラメータは、複数の部分に分割されたフォームでエンコードされたファイルを含む HTTP リクエスト・インスタンスです。JSP ページの場合は、暗黙的な request オブジェクトを使用します。

- `void setBaseDir(javax.servlet.ServletContext sc,`
  `javax.servlet.http.HttpServletRequest req)`
  `throws FileAccessException`

  ファイル・システムへのアップロードでは、このメソッドを使用してベース・ディレクトリとして何を使用するかを決定します。この情報は、アプリケーションの /WEB-INF ディレクトリにある fileaccess.properties ファイルから取得されます。このファイルは、サーブレット・コンテキストの入力パラメータを使用して検出されます。baseDir 設定と destination 設定の組合せによって、アップロード・ディレクトリへの絶対パスが指定されます。

  req パラメータは、ベース・ディレクトリ情報のリクエストに使用するサーブレット・リクエスト・インスタンスです。JSP ページの場合は、暗黙的な request オブジェクトを使用します。

  このメソッドは、データベースのアップロードには関係ありません。

- `void setDestination(String destination)`

  このメソッドは常に必須です。

  ファイル・システムへのアップロードでは、destination とベース・ディレクトリの組合せで、アップロード・ディレクトリへの絶対パスが指定されます。

  データベースへのアップロードでは、destination がファイル接頭辞として使用されます。（ベース・ディレクトリはありません。）この接頭辞はファイル・システム・パスと同じで、ファイルを階層にグループ化するために使用できます。接続先文字列には、「.」や「/」などのセパレータ文字を含めることができます。

  注意： 通常、destination 値の少なくとも一部は、ユーザー入力に基づいています。

- `void setDestinationType(String destinationType)`
  `throws FileAccessException`

- `void setDestinationType(int destinationType)`
  `throws FileAccessException`

  オーバーロードされた setDestinationType() メソッドを使用して、アップロードの対象がファイル・システムであるかデータベースであるかを指定します。

  データベースへのアップロードでは、destinationType を、文字列「database」、定義済の String 定数 FileAccessUtil.DATABASE、int 型の値 1、または定義済の int 型の定数 FileAccessUtil.LOCATION_TYPE_DATABASE のいずれかに設定します。

  ファイル・システムへのアップロードはデフォルトです。ただし、これを明示的に指定する場合は、destinationType を、文字列「filesystem」、定義済の String 定数 FileAccessUtil.FILESYSTEM、int 型の値 0、または定義済の int 型の定数 FileAccessUtil.LOCATION_TYPE_FILESYSTEM のいずれかに設定します。

  FileAccessUtil は、oracle.jsp.webutil.fileaccess パッケージ内にあります。

- `String getDestinationType()`

  接続先情報を取得します。文字列を返す getter メソッドのみ提供されています。

- `void setOverwrite(String overwrite)`
  `throws FileAccessException`

- `void setOverwrite(boolean overwrite)`

  オーバーロードされた `setOverwrite()` メソッドを使用して、既存ファイルの上書きまたは同じファイル名と接頭辞を持つ行の更新を行います。このメソッドは、ファイル・システムとデータベースのアップロード両方に関係しています。

  上書きはデフォルトで有効化されていますが、文字列「`true`」またはブール値 `true` の `overwrite` 設定で、明示的に有効化できます。上書きの無効化は、文字列「`false`」またはブール値 `false` の設定で行います。文字列の設定には、大 / 小文字区別がありません。ここで示した以外の設定は、受け入れられません。

- `void setFileType(String fileType)`
  `throws FileAccessException`

- `void setFileType(int fileType) throws FileAccessException`

  データベースへのアップロードでは、オーバーロードされた `setFileType()` メソッドを使用して、データの格納を、バイナリ・データ用の BLOB（デフォルト）で行うか、文字データ用の CLOB で行うかを指定します。CLOB の場合は、`fileType` を、文字列「`character`」、定義済の `String` 定数 `FileAccessUtil.CHARACTER_FILE`、または `int` 型の値 1 のいずれかに設定します。BLOB を明示的に指定するには、`fileType` を、文字列「`binary`」、定義済の `String` 定数 `FileAccessUtil.BINARY_FILE`、または `int` 型の値 0 のいずれかに設定します。文字列の設定には、大 / 小文字区別がありません。ここで示した以外の設定は、受け入れられません。

  `FileAccessUtil` は、`oracle.jsp.webutil.fileaccess` パッケージ内にあります。

- `String getFileType()`

  ファイル・タイプ情報を取得します。文字列を返す getter メソッドのみ提供されています。

- `void setTable(String tableName)`

  デフォルトの `fileaccess` 表以外のデータベース表へのアップロードでは、このメソッドを使用して表名を指定します。

- `String getTable()`

  表名を取得します。

- `void setPrefixColumn(String prefixColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表へのアップロードでは、このメソッドを使用してファイル接頭辞を格納する列の名前を指定します。（`fileaccess` では、この列名は、`fileprefix` です。）`destination` 値がこの列に書き込まれます。

- `String getPrefixColumn()`

  ファイル接頭辞を格納する列名を取得します。

- `void setFileNameColumn(String fileNameColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表へのアップロードでは、このメソッドを使用してファイル名を格納する列の名前を指定します。（`fileaccess` では、この列名は、`filename` です。）ファイル名には、ファイル名の拡張子が含まれます。

- `String getFileNameColumn()`

  ファイル名を格納する列名を取得します。

- `void setDataColumn(String dataColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表へのアップロードでは、このメソッドを使用してファイル・コンテンツを格納する BLOB 列または CLOB 列の名前を指定します。（`fileaccess` では、この列名は、`data` です。）

- `String getDataColumn()`

  ファイル・コンテンツを格納する列名を取得します。

- `void setConnection(ConnBean conn)`

- `void setConnection(java.sql.Connection conn)`

  データベース表（デフォルトの表またはそれ以外の表）へのアップロードでは、このオーバーロードされたメソッドを使用してデータベース接続を指定します。`oracle.jsp.dbutil.ConnBean` または標準の `java.sql.Connection` 型のいずれかのインスタンスを指定できます。`ConnBean` JavaBean の詳細は、4-3 ページの「データベース接続用 ConnBean」を参照してください。

  `Connection` インスタンスを使用する場合は、明示的なオープンとクローズが必要です。`ConnBean` インスタンスの場合は、自動的に処理されます。

- `java.util.Enumeration getFileNames()`

  このメソッドは、アップロードされたファイルの名前を格納する `Enumeration` インスタンスを戻します。（この機能は、`httpUpload` タグでは使用できません。）

例： この例では、単純な HTML フォームを使用してファイル・システムにアップロードするファイルを指定し、次に、`HttpUploadBean` を採用している JSP ページを使用してアップロードを行います。

次に示す HTML フォームでは、その操作用に `beanUploadExample.jsp` を指定し、複数の部分に分割されたアップロード・ストリームを生成します。

> **注意：** ファイル・システムへのアップロード用にベース・ディレクトリを適切に設定してください。8-2 ページの「ファイル・システムの接続先」を参照してください。

```
<html><body>
<form action="beanUploadExample.jsp" ENCTYPE="multipart/form-data" method=POST>
<br> File to upload: <INPUT TYPE="FILE" NAME="File" SIZE="50" MAXLENGTH="120" >
<br><INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Send"> </form>
</body></html>
```

次に `beanUploadExample.jsp` ページを示します。

```
<%@ page language="java"
         import="java.util.*, oracle.jsp.webutil.fileaccess.*" %>
<html><body>
<% String userdir = "fileaccess"; %>   // user's part of the upload directory
<jsp:useBean id="upbean"
   class="oracle.jsp.webutil.fileaccess.HttpUploadBean" >
   <jsp:setProperty name="upbean" property="destination"
                    value="<%= userdir %>" />
</jsp:useBean>
<%   upbean.setBaseDir(application, request);
         upbean.upload(request);
         Enumeration fileNames = upbean.getFileNames();
         while (fileNames.hasMoreElements()) { %>
              <br><%= (String)fileNames.nextElement() %>
  <% } %>
<br>Done!
</body></html>
```

HttpDownloadBean

oracle.jsp.webutil.fileaccess.HttpDownloadBean JavaBean には、ダウンロードに使用する情報を指定するための setter メソッドが多数あります。また対応するほとんどの getter メソッドも含まれています。必須属性と適切な属性をすべて設定した後、listFiles() メソッドを使用して、ダウンロードするファイルのリストを表示します。実際のダウンロードでは、OC4J が提供する DownloadServlet を使用して一度に 1 つのファイルがダウンロードされます。8-12 ページの「ダウンロード・サーブレット」を参照してください。

> **注意：** DownloadServlet の URL をアプリケーション・コード内に構成する必要があります。

HttpDownloadBean は、HttpUploadBean と同様に、HttpFileAccessBean を拡張します。ただし、これ自体は、公開使用を目的としていません。

関連情報は、8-2 ページの「ファイル・アップロードの概要」を参照してください。

必須属性のサマリー

次に、HttpDownloadBean の必須属性のサマリーを示します。

- 常に必須：source
- データベースへのアップロードに必須：sourceType、connection
- デフォルトの fileaccess 表以外のデータベース表からのダウンロードに必須：table、prefixColumn、fileNameColumn、dataColumn
- ファイル・データに CLOB 列を使用しているデータベース表からのダウンロードに必須：fileType

さらに、ファイル・システムからのダウンロードでは、setBaseDir() メソッドをコールして、サーブレット・コンテキストと HTTP リクエスト・オブジェクトを指定する必要があります。その結果、Bean はベース・ディレクトリを指定する fileaccess.properties ファイルを検出できます。

メソッド

次に、HttpDownloadBean のパブリック・メソッドを説明します。

> **注意：** HttpDownloadBean の属性と setter メソッドの多くは、HttpUploadBean の場合と同じです。

- ```
  void listFiles(javax.servlet.http.HttpServletRequest req)
                                  throws FileAccessException
  ```

  必須の Bean 属性と適切な Bean 属性をすべて設定した後、このメソッドを使用してダウンロードするファイルのリストを表示します。これらのファイルは、ソース・ディレクトリにあるファイルか、ソース・データベース接頭辞に一致するファイルです。req パラメータは、HTTP レスポンス・インスタンスです。JSP ページの場合は、暗黙的な request オブジェクトを使用します。

  ファイル・リストから選択する場合は、DownloadServlet への HREF リンクを作成し、そのリンクを各ファイルとファイル接頭辞に渡します。これにより、ユーザーは、ダウンロードする各ファイルのリンクを選択できます。

  > **注意：** listFiles() メソッドは、ファイル名をメモリーおよび JSP ページまたはサーブレットに書き込みます。後でそのファイル名に再度アクセスする場合は、getFileNames() メソッドを使用して、ファイル名をメモリーから読み取ります。
  ```

- `java.util.Enumeration getFileNames()`

  このメソッドは、ダウンロードするファイルの名前を格納する Enumeration インスタンスを戻します。listFiles() メソッドはすでにコールされている必要があります。listFiles() メソッドは、ファイル名をメモリーおよび JSP ページまたはサーブレットに書き込みます。getFileNames() メソッドはそのファイル名をメモリーから読み取ります。

- ```
  void setBaseDir(javax.servlet.ServletContext sc,
                  javax.servlet.http.HttpServletRequest req)
                                      throws FileAccessException
  ```

  ファイル・システムからのダウンロードでは、このメソッドを使用してベース・ディレクトリとして何を使用するかを決定します。この情報は、アプリケーションの /WEB-INF ディレクトリにある fileaccess.properties ファイルから取得されます。このファイルは、サーブレット・コンテキストの入力パラメータを使用して検出されます。baseDir 設定と source 設定の組合せによって、ファイルのダウンロード元のディレクトリへの絶対パスが指定されます。

  sc パラメータは、アプリケーションのサーブレット・コンテキスト・インスタンスです。JSP ページの場合は、暗黙的な application オブジェクトを使用します。

  req パラメータは、ベース・ディレクトリ情報のリクエストに使用する HTTP リクエスト・インスタンス用です。JSP ページの場合は、暗黙的な request オブジェクトを使用します。

  ベース・ディレクトリは、データベースからのダウンロードには関係ありません。

- `void setSource(String source)`

  このメソッドは常に必須です。

  ファイル・システムからのダウンロードでは、source とベース・ディレクトリの組合せによって、ファイルのダウンロード元のディレクトリへの絶対パスが指定されます。source が「*」に設定されている場合、ベース・ディレクトリの下にあるすべてのディレクトリがダウンロードの対象となります。

  データベースからのダウンロードでは、source がファイル接頭辞として使用されます。（ベース・ディレクトリは関係ありません。）この接頭辞はファイル・システム・パスと同じで、ファイルを階層にグループ化するために使用できます。再帰的なダウンロードが有効な場合（setRecurse() メソッドを使用）は、「%」が source 値に付加され、問合せの WHERE 句に適切な LIKE 句が含まれます。したがって、source 値と部分的に一致する接頭辞を持つすべてのファイルがダウンロードの対象となります。データベース表のすべての行が一致する必要がある場合は、source を「*」に設定します。

  注意： 通常、source 値の少なくとも一部は、ユーザー入力に基づいています。

- ```
  void setSourceType(String sourceType)
                              throws FileAccessException
  ```

- ```
  void setSourceType(int sourceType)
                            throws FileAccessException
  ```

  オーバーロードされた setSourceType() メソッドを使用して、ダウンロード元がファイル・システムであるかデータベースであるかを指定します。

  データベースからのダウンロードでは、sourceType を、文字列「database」、定義済の String 定数 FileAccessUtil.DATABASE、int 型の値 1、または定義済の int 型の定数 FileAccessUtil.LOCATION_TYPE_DATABASE のいずれかに設定します。

  ファイル・システムからのダウンロードはデフォルトです。ただし、これを明示的に指定する場合は、sourceType を、文字列「filesystem」、定義済の String 定数 FileAccessUtil.FILESYSTEM、int 型の値 0、または定義済の int 型の定数 FileAccessUtil.LOCATION_TYPE_FILESYSTEM のいずれかに設定します。

  FileAccessUtil は、oracle.jsp.webutil.fileaccess パッケージ内にあります。
  ```

- `String getSourceType()`

  ソース・タイプ情報を取得します。文字列を返す getter メソッドのみ提供されています。

- `void setRecurse(String recurse) throws FileAccessException`

- `void setRecurse(boolean recurse)`

  オーバーロードされた `setRecurse()` メソッドを使用して、再帰的なダウンロード機能を有効または無効にします。この場合、ファイル・システムのサブディレクトリ内のファイルやデータベース接頭辞の追加情報を含むファイルも、ダウンロードの対象としてリストされます。データベースからのこの機能の例として、`source` が「`/user`」に設定されていると仮定します。再帰機能では、「`/user/bill`」や「`/user/mary`」、および「`/user1`」、「`/user2`」、「`/user1/tom`」、「`/user2/susan`」などの接頭辞を持つファイルに対しても一致を検出します。

  再帰機能はデフォルトで有効化されていますが、文字列「`true`」またはブール値 `true` の `recurse` 設定で、明示的に有効化できます。再帰機能の無効化は、文字列「`false`」またはブール値 `false` の設定で行います。文字列の設定には、大 / 小文字区別がありません。ここで示した以外の設定は、受け入れられません。

- `void setFileType(String fileType)`
  `throws FileAccessException`

- `void setFileType(int fileType) throws FileAccessException`

  データベースからのダウンロードでは、オーバーロードされた `setFileType()` メソッドを使用して、データの格納を、バイナリ・データ用の BLOB（デフォルト）で行うか、文字データ用の CLOB で行うかを指定します。CLOB の場合は、`fileType` を、文字列「`character`」、定義済の `String` 定数 `FileAccessUtil.CHARACTER_FILE`、または `int` 型の値 1 のいずれかに設定します。BLOB を明示的に指定するには、`fileType` を、文字列「`binary`」、定義済の `String` 定数 `FileAccessUtil.BINARY_FILE`、または `int` 型の値 0 のいずれかに設定します。文字列の設定には、大 / 小文字区別がありません。ここで示した以外の設定は、受け入れられません。

  `FileAccessUtil` は、`oracle.jsp.webutil.fileaccess` パッケージ内にあります。

- `String getFileType()`

  ファイル・タイプ情報を取得します。文字列を返す getter メソッドのみ提供されています。

- `void setTable(String tableName)`

  デフォルトの `fileaccess` 表以外のデータベース表からのダウンロードでは、このメソッドを使用して表名を指定します。

- `String getTable()`

  表名を取得します。

- `void setPrefixColumn(String prefixColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表からのダウンロードでは、このメソッドを使用してファイル接頭辞を格納する列の名前を指定します。（`fileaccess` では、この列名は、`fileprefix` です。）

- `String getPrefixColumn()`

  ファイル接頭辞を格納する列名を取得します。

- `void setFileNameColumn(String fileNameColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表からのダウンロードでは、このメソッドを使用してファイル名を格納する列の名前を指定します。（`fileaccess` では、この列名は、`filename` です。）ファイル名には、ファイル名の拡張子が含まれます。

- `String getFileNameColumn()`

  ファイル名を格納する列名を取得します。

- `void setDataColumn(String dataColumnName)`

  デフォルトの `fileaccess` 表以外のデータベース表からのダウンロードでは、このメソッドを使用してファイル・コンテンツを保持する BLOB 列または CLOB 列の名前を指定します。（`fileaccess` では、この列名は、`data` です。）

- `String getDataColumn()`

  ファイル・コンテンツを格納する列名を取得します。

- `void setConnection(ConnBean conn)`

- `void setConnection(java.sql.Connection conn)`

  データベース表（デフォルトの表やそれ以外を含む）からのダウンロードでは、このメソッドを使用してデータベース接続を指定します。`oracle.jsp.dbutil.ConnBean` または標準の `java.sql.Connection` 型のいずれかのインスタンスを指定できます。`ConnBean` JavaBean の詳細は、4-3 ページの「データベース接続用 ConnBean」を参照してください。

  `Connection` インスタンスを使用する場合は、明示的なオープンとクローズが必要です。`ConnBean` インスタンスの場合は、自動的に処理されます。

例 次に、ファイル・システムからのダウンロードに `HttpDownloadBean` を使用している JSP ページの例を示します。このページでは、ダウンロード・サーブレットの URL を構成する必要があります。

```
<%@ page language="java" import="java.util.*, oracle.jsp.webutil.fileaccess.*" %>
<html><body>
<% String servletPath = "/servlet/download/";  // path to the download servlet
   String userDir = "fileaccess/";   // user part of  download directory
%>
<jsp:useBean id="dbean"
   class="oracle.jsp.webutil.access.HttpDownloadBean" >
   <jsp:setProperty name="dbean" property="source" value='<%=userDir %>' />
</jsp:useBean>
<%    dbean.setBaseDir(application, request);
      dbean.listFiles(request); %>
The following files were found:
<%   Enumeration fileNames = dbean.getFileNames();
         while (fileNames.hasMoreElements()) {
                String name = (String)fileNames.nextElement();  %>
               <br><a href="<%= servletPath + name %>" > <%= name %></a>
<%    } %>
<br>Done!
</body></html>
```

ダウンロード・サーブレット

`HttpDownloadBean` タグまたは `httpDownload` タグを介してダウンロード機能を使用するには、Web サーバーで使用できる `oracle.jsp.webutil.fileaccess.DownloadServlet` クラスが必要です。Web サーバーでのクラスのマッピングは、サーブレット・パス設定にも反映する必要があります。`httpDownload` タグを使用する場合は、`servletPath` 属性を使用して反映し、`HttpDownloadBean` を使用する場合は、アプリケーション・コード内に反映します。

FileAccessException クラス

`oracle.jsp.webutil.fileaccess.FileAccessException` クラスは、OC4J が提供する便利なクラスで、ファイル・アクセスに関する例外を処理します。このクラスは、標準の `java.sql.SQLException` クラスと `java.io.IOException` クラスの機能をラップします。また、SQL と I/O の例外処理に加えて、ファイル・アクセス Bean からの例外も処理します。

ファイル・アップロードおよびダウンロード用タグの説明

OC4J には、ファイル・アップロード用の httpUpload タグがあります。このタグでは、HttpUploadBean を使用します。httpUploadForm タグは、ユーザーがアップロードするファイルを指定するフォームのプログラミングで使用すると便利です。このフォームのコードは手動でも作成できます。

OC4J には、ファイルのダウンロード用にカスタム httpDownload タグが用意されています。このタグでは、HttpDownloadBean を使用します。この項では、これらのタグとその属性について説明します。

ファイルのアップロードおよびダウンロード用タグを使用する場合は、次の要件に注意してください。

- ojsputil.jar ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J に同梱されていて、予約済のタグ・ライブラリ・ディレクトリにあります。
- タグ・ライブラリ・ディスクリプタ・ファイル fileaccess.tld が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な taglib ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は ojsputil.jar に配置されます。fileaccess.tld の uri 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/fileaccess.tld
  ```

taglib ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび uri 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「fileaccess:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

---

httpUploadForm タグ

httpUploadForm タグを使用すると、複数の部分に分割されてエンコードされたフォーム・データを使用して、アプリケーションにフォームを簡単に作成できます。このフォームによって、ユーザーはアップロードするファイルを指定できます。

構文

```
<fileaccess:httpUploadForm formsAction = "action"
                           [ maxFiles = "max_number" ]
                           [ fileNameSize = "file_input_box_num_chars" ]
                           [ maxFileNameSize = "max_file_name_num_chars" ]
                           [ includeNumbers = "true" | "false" ]
                           [ submitButtonText = "button_label_text" ] />
```

注意： httpUploadForm タグは、必要に応じてボディを使用できます。たとえば、ボディがユーザー・プロンプトで構成されている場合があります。

属性

- formsAction（必須）: フォームの送信後に実行される操作を示します。たとえば、formsAction を、HttpUploadBean または httpUpload タグを使用する JSP ページの名前にすることもできます。
- maxFiles: フォームに表示する入力行の数を指定します。デフォルトは 1 です。
- fileNameSize: ファイル名の入力ボックス（複数可）の文字幅を指定します。デフォルトは 20 文字です。
- maxFileNameSize: ファイル名の最大許容文字数を指定します。デフォルトは 80 文字です。
- includeNumbers: ファイル名の入力ボックスに番号を付ける場合は、「true」に設定します。デフォルト設定は「false」です。
- submitButtonText: フォームの送信ボタン上に表示するテキストを指定します。デフォルトは「Send」です。

httpUpload タグ

このタグは、HttpUploadBean JavaBean の機能をラップし、その属性を対応させます。関連情報は、8-2 ページの「ファイル・アップロードの概要」および 8-5 ページの「HttpUploadBean」を参照してください。

構文

```
<fileaccess:httpUpload destination = "dir_path_or_prefix"
                [ destinationType = "filesystem" | "database" ]
                [ connId = "id" ]
                [ scope = "request" | "page" | "session" | "applicaton" ]
                [ overwrite = "true" | "false" ]
                [ fileType = "character" | "binary" ]
                [ table = "table_name" ]
                [ prefixColumn = "column_name" ]
                [ fileNameColumn = "column_name" ]
                [ dataColumn = "column_name" ] />
```

注意： ファイル・システムへのアップロードでは、ベース・ディレクトリは、タグ・ハンドラによって JSP ページ・コンテキストから自動的に取得できます。

属性

- destination（必須）: ファイル・システムへのアップロードでは、ファイルのアップロード先のディレクトリ・パスを示します。このパスは /WEB-INF/fileaccess.properties ファイルに指定されているベース・ディレクトリの下にあります。データベースへのアップロードでは、destination がファイル接頭辞を示します。これは、概念上、ファイル・システム・パスと同じです。

  注意： 通常、destination 値の少なくとも一部は、ユーザー入力に基づいています。

- destinationType: データベースへのアップロードでは、「database」に設定します。デフォルトは、ファイル・システムへのアップロードですが、明示的に「filesystem」に設定することもできます。この値には、大 / 小文字区別がありません。
- connId: データベースへのアップロードでは、使用するデータベース接続の ConnBean 接続 ID を指定します。または、httpUpload タグを dbOpen タグ内で使用して、dbOpen 接続を暗黙的に使用します。OC4J が提供する ConnBean JavaBean と dbOpen タグの詳細は、第 4 章「データ・アクセス JavaBeans とデータ・アクセス・タグ」を参照してください。

- scope: データベースへのアップロードでは、接続に使用する ConnBean インスタンスのスコープを指定します。ここで指定するスコープ設定は、dbOpen タグの場合と同様に、ConnBean インスタンスの作成時のスコープ設定と一致している必要があります。httpUpload タグが dbOpen タグ内にネストされている場合は、connId または scope の指定は不要です。この場合、情報は dbOpen タグから取得されます。それ以外の場合、デフォルトのスコープ設定は「page」です。
- overwrite: アップロードするファイルと同じパスと名前を持つ既存ファイルを上書きしない場合、またはデータベースのアップロードで、同じファイル名と接頭辞を持つ行を更新しない場合は、「false」に設定します。この場合、ファイルがすでに存在していると、エラーが生成されます。デフォルトでは、overwrite は「true」に設定され、httpUpload はファイルを上書きします。
- fileType: データベースへのアップロードでは、文字データの場合は「character」に設定します。この文字データは CLOB に書き込まれます。デフォルトの設定はバイナリ・データの「binary」です。このバイナリ・データは BLOB に書き込まれます。
- table: デフォルトの fileaccess 表以外のデータベース表へのアップロードでは、この属性を使用して表名を指定します。
- prefixColumn: デフォルトの fileaccess 表以外のデータベース表へのアップロードでは、この属性を使用してファイル接頭辞を含む列の名前を指定します。この列には、destination 値が書き込まれます。
- fileNameColumn: デフォルトの fileaccess 表以外のデータベース表へのアップロードでは、この属性を使用してファイル名を含む列の名前を指定します。
- dataColumn: デフォルトの fileaccess 表以外のデータベース表へのアップロードでは、この属性を使用してファイル・コンテンツを含む列の名前を指定します。

例 次に、httpUploadForm タグを使用して、アップロードするファイルを指定する HTML フォームを作成するページの例を示します。httpUploadForm タグは、そのフォーム操作として、httpUploadExample.jsp を指定します。httpUploadExample.jsp ページでは、httpUpload タグを使用して、データベース内のデフォルトの fileaccess 表にアップロードします。

次に、HTML フォームのページを示します。

```
<%@ page language="java" import="java.io.*"  %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/fileaccess.tld"
           prefix="upload" %>
<html> <body>
<fileaccess:httpUploadForm
      formsAction="httpUploadExample.jsp"
      maxFiles='<%= request.getParameter("MaxFiles") %>'
      includeNumbers="true"  fileNameSize="50"  maxFileNameSize="120" >
          <br> File:
</fileaccess:httpUploadForm>
</body> </html>
```

次は、httpUploadExample.jsp ページです。httpUpload タグは、dbOpen タグ内に存在することによって、データベース接続を取得している点に注意してください。必要な場合は、接続の取得に useDataSource.jsp が使用されている点にも注意してください。5-9 ページの「useDataSource.jsp」を参照してください。

```
<%@ page language="java" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/fileaccess.tld"
           prefix="upload" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/sqltaglib.tld"
           prefix="sql" %>
<% String dataSrcStr=request.getParameter("dataSrcStr"); // get conn string
   if (dataSrcStr==null) {  dataSrcStr=(String)session.getValue("dataSrcStr"); }
   else {  session.putValue("dataSrcStr",dataSrcStr); }
   if (dataSrcStr==null) { %>
       <jsp:forward page="useDataSource.jsp" />
```

```
<% } %>
<html><body>
<sql:dbOpen dataSource="<%= dataSrcStr %>" >
   <fileaccess:httpUpload  destinationType = "database"
                           destination="tagexample" />
</sql:dbOpen>
Done! </body></html>
```

注意： この例の dbOpen タグの場合は、データ・ソースで URL、ユーザー名およびパスワードが指定されているものとします。

httpDownload タグ

このタグは、HttpDownloadBean JavaBean の機能をラップし、その属性を対応させます。関連情報は、8-4 ページの「ファイル・ダウンロードの概要」および 8-9 ページの「HttpDownloadBean」を参照してください。

構文

```
<fileaccess:httpDownload servletPath = "path"
                  source = "dir_path_or_prefix"
                [ sourceType = "filesystem" | "database" ]
                [ connId = "id" ]
                [ scope = "request" | "page" | "session" | "applicaton" ]
                [ recurse = "true" | "false" ]
                [ fileType = "character" | "binary" ]
                [ table = "table_name" ]
                [ prefixColumn = "column_name" ]
                [ fileNameColumn = "column_name" ]
                [ dataColumn = "column_name" ] />
```

注意：

- httpDownload タグは、必要に応じてボディを使用できます。たとえば、ボディがユーザー・プロンプトで構成されている場合があります。
- ファイル・システムからのダウンロードでは、ベース・ディレクトリは、タグ・ハンドラによって JSP ページ・コンテキストから自動的に取得できます。

属性

- servletPath（必須）：各ファイルのダウンロードを実際に実行する Oracle DownloadServlet へのパスです。たとえば、DownloadServlet がすでにアプリケーション app にインストールされ、その名前 download にマップされている場合は、「/app/download/」を先頭のスラッシュと末尾のスラッシュ付きで、servletPath 設定として使用します。httpDownload タグ・ハンドラは、このパスを DownloadServlet への URL の構成に使用します。

  このサーブレットの詳細は、8-12 ページの「ダウンロード・サーブレット」を参照してください。

- source（必須）：ファイル・システムからのダウンロードでは、ファイルの取得元のディレクトリ・パスを示します。このパスは /WEB-INF/fileaccess.properties ファイルに指定されているベース・ディレクトリの下にあります。値「*」を指定すると、ベース・ディレクトリの下の全ディレクトリが使用可能になります。

  データベースからのダウンロードでは、この属性は、ファイル接頭辞を示します。これは、概念上、ファイル・システム・パスと同じです。再帰的なダウンロードが有効な場合（recurse 属性を使用）は、「%」が source 値に付加され、問合せの WHERE 句に適切な LIKE 句が含まれます。したがって、source 値と部分的に一致する接頭辞を持つすべてのファイルがダウンロードの対象となります。データベース表のすべての行が一致する必要がある場合は、source を「*」に設定します。

注意： 通常、source 値の少なくとも一部は、ユーザー入力に基づいています。

- sourceType: データベースからのダウンロードでは、「database」に設定します。デフォルトは、ファイル・システムからのダウンロードですが、明示的に「filesystem」に設定することもできます。
- connId: データベースからのダウンロードでは、使用するデータベース接続の ConnBean 接続 ID を指定します。または、httpDownload タグを dbOpen タグ内で使用して、dbOpen 接続を暗黙的に使用します。OC4J が提供する ConnBean JavaBean と dbOpen タグの詳細は、第 4 章「データ・アクセス JavaBeans とデータ・アクセス・タグ」を参照してください。
- scope: データベースからのダウンロードでは、接続に使用する ConnBean インスタンスのスコープを指定します。ここで指定するスコープ設定は、dbOpen タグの場合と同様に、ConnBean インスタンスの作成時のスコープ設定と一致している必要があります。httpDownload タグが dbOpen タグ内にネストされている場合は、connId または scope の指定は不要です。この場合、情報は dbOpen タグから取得されます。それ以外の場合、デフォルトのスコープ設定は「page」です。
- recurse: 再帰的なダウンロード機能を使用しない場合は、「false」に設定します。再帰的なダウンロードでは、ファイル・システムのサブディレクトリ内のファイルまたはデータベース接頭辞の追加情報を含むファイルも、ダウンロード可能としてリストされます。データベースからのこの機能の例として、source が「/user」に設定されていると仮定します。再帰機能では、「/user/bill」や「/user/mary」、および「/user1」、「/user2」、「/user1/tom」、「/user2/susan」などの接頭辞を持つファイルに対しても一致を検出します。デフォルト・モードは再帰的なダウンロードですが、「true」に設定すると、明示的に有効化できます。
- fileType: データベースからのダウンロードでは、文字データの場合は「character」に設定します。この文字データは CLOB から取得されます。デフォルトの設定はバイナリ・データの「binary」です。このバイナリ・データは BLOB から取得されます。
- table: デフォルトの fileaccess 表以外のデータベース表からのダウンロードでは、この属性を使用して表名を指定します。
- prefixColumn: デフォルトの fileaccess 表以外のデータベース表からのダウンロードでは、この属性を使用してファイル接頭辞を含む列の名前を指定します。これは、source 値が格納される場所です。
- fileNameColumn: デフォルトの fileaccess 表以外のデータベース表からのダウンロードでは、この属性を使用してファイル名を含む列の名前を指定します。ファイル名には、ファイル名の拡張子が含まれます。
- dataColumn: デフォルトの fileaccess 表以外のデータベース表からのダウンロードでは、この属性を使用してファイル・コンテンツを含む列の名前を指定します。

例 次に、httpDownload タグを使用してデータベースのデフォルトの fileaccess 表からダウンロードする JSP ページの例を示します。タグ・ボディのコンテンツ（
:）は、ダウンロード可能なファイルのリストにある各ファイル名の前に出力されます。DownloadServlet サーブレットのパスを httpDownload タグに指定する必要があることに注意してください。タグ・ハンドラは、このパスを使用して、ダウンロードを実際に実行する DownloadServlet への URL を構成します。

```
<%@ page language="java" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/fileaccess.tld"
           prefix="download" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/sqltaglib.tld"
           prefix="sql" %>
<% String dataSrcStr=request.getParameter("dataSrcStr");
   if (dataSrcStr==null) { dataSrcStr=(String)session.getValue("dataSrcStr");}
   else { session.putValue("dataSrcStr",dataSrcStr);}
   if (dataSrcStr==null) { %>
      <jsp:forward page="useDataSource.jsp" />
<% } %>
<html> <body>
<% String servletPath = "/servlet/download/"; %>
<sql:dbOpen dataSource="<%= dataSrcStr %>" >
<fileaccess:httpDownload sourceType = "database"
            source="tagexample" servletPath = '<%= servletPath %>' >
   <br>:
</fileaccess:httpDownload>
</sql:dbOpen>
<br>Done!
</body> </html>
```

注意： この例の dbOpen タグの場合は、データ・ソースで URI、ユーザー名およびパスワードが指定されているものとします。

メール JavaBean とメール・タグ

多くの場合、Web サイトの状態またはユーザー操作に基づいて、電子メール・メッセージを Web アプリケーションから送信できると便利です。Sun 社では、このために、プラットフォームやプロトコルに依存しないフレームワークの仕様を定めました。これは、JavaMail API と呼ばれ、javax.mail パッケージとサブパッケージを使用します。

利便性向上のために、Oracle には、サーブレットや JSP ページを介して電子メール機能を使用できるように、この JavaMail API に基づいた JavaBean と JSP カスタム・タグが用意されています。Bean とタグは、OC4J が提供する他の JavaBeans やカスタム・タグと同様、JSP 標準とサーブレット標準に基づいて実装されます。

次の各項では、メール JavaBean とメール・タグについて説明します。

- メール JavaBean とメール・タグについての一般的な考慮事項
- メール添付
- SendMailBean の説明
- sendMail タグの説明

JavaMail API の詳細は、次の Sun 社の Web サイトを参照してください。

http://java.sun.com/products/javamail/1.2/docs/javadocs/index.html

メール JavaBean とメール・タグについての一般的な考慮事項

メール JavaBean（SendMailBean）またはメール・タグ（sendMail）を使用するときは、次の点に注意してください。

- メール機能を使用するには、mail.jar ファイル（JavaMail パッケージを含む）と JavaBeans Activation Framework（JAF）の jaf.jar ファイルが、クラスパスに存在している必要があります。これらのファイルは、OC4J に含まれています。
- 添付のサポートを有効化するには、適切に設定されたファイル sendmail.properties が、アプリケーションの /WEB-INF ディレクトリに存在する必要があります。次項の「添付の有効化」を参照してください。
- 電子メール・メッセージのサイズに特別の制限はありません。ただし、JVM、システム・メモリーまたはメール・サーバーには制限があります。
- デフォルトのメール・セッションの設定は、特定の Web サーバーに固有の設定です。メール Bean とメール・タグの現行の実装では、デフォルトのメール・セッションを自動的に使用することはサポートされていません。かわりに、独自のコードを作成してデフォルトのメール・セッション（プラットフォームにある場合）を取得し、メール Bean またはメール・タグで使用可能にできます。

メール添付

メール Bean とメール・タグによって、電子メール・メッセージとともに添付を送信できます。（このサポートは、OC4J 9.0.3 で実装されています。）操作モードは次の 3 つです。

- 添付のサポートなし。
- OC4J サーバー・マシン上にある 1 つ以上のファイルの添付をサポート。サーバー・サイド添付と呼ばれます。
- クライアント・マシン上にある 1 つのファイルの添付をサポート。クライアント・サイド添付と呼ばれます。

クライアント・サイド添付の場合、ファイルは、処理の一部として、サーバー・マシンに自動的にアップロードされます。複数のクライアント・サイド添付はサポートされません。

添付の有効化

添付が有効かどうか、および添付の種類は、アプリケーションの /WEB-INF ディレクトリにある sendmail.properties ファイルによって決まります。次の内容を含むファイルでは、添付が無効化されます。

```
## email attachment permissions
sendmail.attachment=none
```

このファイルを /WEB-INF に作成し、メール添付を使用する OC4J インスタンス用に適切に更新する必要があります。

1 つのアプリケーションでサポートできるのは、サーバー・サイド添付かクライアント・サイド添付のいずれかです。両方はサポートできません。

サーバー・サイド添付を有効化するには、設定を server に変更します。

```
sendmail.attachment=server
```

クライアント・サイド添付を有効化するには、設定を client に変更します。

```
sendmail.attachment=client
```

複数設定するとエラーになります。

注意： sendmail.properties ファイルが存在しない場合は、sendmail.properties が none に設定された場合と同様に処理されます。この場合、メール添付は無効化されます。

添付の送信

メール・タグでサーバー・サイド添付が有効化されている場合、1 つ以上のサーバー・サイド・ファイルをメッセージに添付するには、`serverAttachment` タグ属性を使用します。クライアント・サイド添付が有効化されている場合、クライアント・サイド・ファイル（最大 1 ファイル）をメッセージに添付するには、`clientAttachment` タグ属性を使用します。8-24 ページの「sendMail タグの説明」を参照してください。1 つのアプリケーションでサポートできるのは、2 つの添付モードのいずれか 1 つです。両方はサポートできません。

サーバー添付モードとクライアント添付モードの両方のメール Bean には、添付ファイルの名前を指定または取得するメソッドが含まれています。`setServerAttachment()`、`getServerAttachment()`、`setClientAttachment()` および `getClientAttachment()` の詳細は、8-21 ページの「SendMailBean メソッドの説明」を参照してください。

メール・タグまたはメール Bean では、サーバー・サイド・ファイルのリストは、カンマ区切りまたはセミコロン区切りで指定できます。ただし、スペース区切りは、ファイル名にスペースが許可されているオペレーティング・システムの場合でも使用できません。

添付の使用方法

メール添付に関する次の使用方法は、メール・タグとメール Bean の両方に適用されます。

- クライアント・サイド・ファイル添付の場合、ファイル・アクセス `httpUpload` タグは、バックグラウンドで使用されます。ファイルは OC4J サーバー・マシン上の一時的な場所にアップロードされ、メッセージが送信された後は削除されます。`httpUpload` タグの制限または要件は、すべてクライアント・サイドのメール添付にも適用されます。8-13 ページの「ファイル・アップロードおよびダウンロード用タグの説明」を参照してください。
- 電子メール・サーバーの多くには、サイズ制限があります。通常は、1 つの添付につき約 4MB に制限されています。メール・タグまたはメール Bean に対する唯一の制限は、サーバー・マシンのディスク制限またはメモリー制限によって決まります。
- 添付に問題が発生すると、電子メール・メッセージは終了されます。
- サーバー添付モードおよびクライアント添付モードのいずれの場合も、パス名はメール受信者には公開されません。公開されるのは、ファイル名のみです。
- サーバー・サイド添付の場合は、複数の同名ファイル（ただし、パスは異なるファイル）の添付がサポートされます。競合を回避するためにファイル名が変更される可能性も含め、これが受信者側でどのように処理されるかは、使用しているメール・クライアントによって異なります。同様に、いずれの添付モードでも、前のメッセージの添付と同名の添付がある場合、メール・クライアントはファイル名を変更する可能性があります。これらはすべて、OC4J のメール添付機能では制御できません。
- ファイル名にワイルド・カード文字は使用できません。

SendMailBean の説明

OC4J が提供する `oracle.jsp.webutil.email.SendMailBean` JavaBean は、サーブレットから JSP アプリケーションまでの電子メール機能をサポートします。これを JSP ページで使用するには、標準の `jsp:useBean` タグを使用してインスタンス化できます。（ただし、JSP アプリケーションの場合は、通常、`SendMailBean` ではなく `sendMail` タグを使用します。8-24 ページの「sendMail タグの説明」を参照してください。）

SendMailBean の要件

`SendMailBean` を使用するには、`ojsputil.jar`、`mail.jar` および `activation.jar` の各ファイルがインストール済で、クラスパスに存在していることを確認します。これらのファイルは、OC4J に含まれています。

`SendMailBean` をコードで使用する場合は、次の内容を指定する必要があります。

- メッセージ送信者

  送信者の指定には、`setSender()` メソッドを使用します。

- メッセージのプライマリ受信者（複数可）

  プライマリ受信者（複数可）の指定には、setRecipient() メソッドを使用します。

- 直接的または間接的で、有効な JavaMail セッション・オブジェクト（javax.mail.Session）

  JavaMail セッションは、次の 3 つの方法で設定されます。

  – setHost() メソッドを使用してホスト・システムを指定します。この場合、JavaMail セッション・オブジェクトは自動的に作成されます。

  – setMailSession() メソッドを使用して、JavaMail セッション・オブジェクトを直接指定します。

  – JSP アプリケーションの場合は、setSession() メソッドを使用して、JSP ページ・コンテキストにある「セッション文字列 javax.mail.Session オブジェクト」のペアでアクセスできる既存の JavaMail セッション・オブジェクトの名前を指定します。この場合は、sendMessage() メソッドをコールして電子メール・メッセージを送信するときに、ページ・コンテキスト・インスタンスを入力パラメータとして指定する必要があります。

その他の SendMailBean の属性は、すべてオプションです。

SendMailBean メソッドの説明

この項では、メール・メッセージの送信、メール・セッションのクローズおよび Bean 属性の設定と取得に使用する SendMailBean のメソッドを示し、内容を説明します。

注意： JavaBean の仕様に準拠して、SendMailBean には、引数なしのコンストラクタがあります。

次に、SendMailBean のパブリック・メソッドを示します。

- void sendMessage()
- void sendMessage(javax.servlet.jsp.PageContext)

  電子メール・メッセージの送信には、sendMessage() メソッドを使用します。

  setSession() メソッドを使用して JavaMail セッションを設定する場合は、sendMessage(PageContext) シグネチャを使用して、指定のメール・セッション・インスタンスを保持しているページ・コンテキスト・インスタンスを指定する必要があります。

  setMailSession() メソッドまたは setHost() メソッドを使用して JavaMail セッションを設定する場合は、sendMessage() メソッドの使用時にページ・コンテキストを指定する必要はありません。

  ただし、ページ・コンテキスト・インスタンスの指定は、「text」コンテンツ・タイプの電子メール・メッセージのキャラクタ・セットの決定に関連する可能性があることに注意してください。sendMessage() メソッドの起動時にページ・コンテキストを指定しない場合、デフォルトのキャラクタ・セットは、ISO-8859-1 になります。ページ・コンテキストを指定した場合、デフォルトのキャラクタ・セットは、ページ・コンテキストの response オブジェクトのキャラクタ・セットになります。コンテンツ・タイプとキャラクタ・セットは、setContentType() メソッドを使用して直接指定することもできます。

- void close()

  JavaMail セッション・インスタンスのリソースを SendMailBean インスタンスから解放する場合は、このメソッドを使用します。このメソッドによって、実際にセッションがクローズされることはありません。

- void setBcc(String s)

  メッセージのブラインド・コピーを受信するには、スペースまたはカンマで区切った ID（電子メール・アドレスまたは別名）のリストを指定します。これらの ID は、メッセージの「Cc」フィールドには表示されません。

- `String getBcc()`

  メッセージのブラインド・コピーを受信するには、ID リストを取得します。

- `void setCc(String s)`

  メッセージのコピーを受信するには、ID（電子メール・アドレスまたは別名）のスペースまたはカンマ区切りのリストを指定します。これらの ID は、メッセージの「Cc」フィールドに表示されます。

- `String getCc()`

  メッセージのコピーを受信するには、ID リストを取得します。

- `void setContent(String s)`

  電子メール・メッセージのコンテンツを指定します。

- `String getContent()`

  電子メール・メッセージのコンテンツを取得します。

- `void setContentEncoding(String s)`

  電子メール・メッセージのコンテンツ・エンコーディングを指定します。base64 エンコーディングには「`base64`」または「`B`」を、quoted-printable エンコーディングには「`quoted-printable`」または「`Q`」を、7 ビット・エンコーディングには「`7bit`」を、8 ビット・エンコーディングには「`8bit`」をそれぞれ指定します。これらのコンテンツ・エンコーディングは、JavaMail 標準と RFC 2047 標準に含まれています。エントリには、大 / 小文字区別がありません。

  デフォルトのコンテンツ・エンコーディングの設定は、「`null`」です。この場合、メッセージとヘッダーのエンコーディングは、コンテンツによって決まります。エンコードされる文字の大部分が ASCII の場合は、quoted-printable エンコーディングが使用されます。それ以外の場合は、`base64` エンコーディングが使用されます。

- `String getContentEncoding()`

  メッセージのコンテンツ・エンコーディングを取得します。

- `void setContentType(String s)`

  MIME タイプおよび必要に応じてメッセージのキャラクタ・セットを、次の例のように指定します。

  ```
  setContentType("text/html");

  setContentType("text/html; charset=US-ASCII");
  ```

  デフォルトの MIME タイプの設定は、「`text/plain`」です。ただし、この MIME タイプまたは他の「`text/xxxx`」MIME タイプの設定を明示的に指定しない場合、キャラクタ・セットを指定できません。

  デフォルトのキャラクタ・セットは、`sendMessage()` メソッドをコールして電子メール・メッセージを送信するときに、JSP ページ・コンテキスト・インスタンスを指定するかどうかによって決まります。ページ・コンテキストを指定しない場合、デフォルトのキャラクタ・セットは、`ISO-8859-1` になります。ページ・コンテキストを指定した場合、デフォルトのキャラクタ・セットは、ページ・コンテキストの `response` オブジェクトのキャラクタ・セットになります。

- `String getContentType()`

  メッセージの MIME タイプ、および適用可能な場合は、文字エンコーディングを取得します。

- `void setHost(String s)`

  JavaMail セッションの設定方法の 1 つは、メール・サーバー・ホスト名を指定する方法です。この指定で、`SendMailBean` はセッションを自動的に取得します。このためには、`setHost()` メソッドを使用して、「`gmail.oraclecorp.com`」などのメール・ホスト名を指定します。

  JavaMail セッションの設定の概要は、8-20 ページの「SendMailBean の要件」を参照してください。

- `String getHost()`

  指定したメール・サーバーのホスト名を取得します。

- `void setMailSession(javax.mail.Session sessobj)`

  JavaMail セッションの設定方法の 1 つは、セッション・オブジェクトを直接指定することです。このためには、`setMailSession()` メソッドを使用して、`javax.mail.Session` インスタンスを指定します。

  JavaMail セッションの設定の概要は、8-20 ページの「SendMailBean の要件」を参照してください。

- `javax.mail.Session getMailSession()`

  このメソッドは、以前に設定した JavaMail セッションを戻します。

- `void setRecipient(String s)`

  メッセージのプライマリ受信者の ID（電子メール・アドレスまたは別名）のスペースまたはカンマ区切りのリストを指定します。これらの ID は、メッセージの「To」フィールドに表示されます。最低 1 名の受信者を指定する必要があります。

- `String getRecipient()`

  メッセージのプライマリ受信者の ID リストを取得します。

- `void setSender(String s)`

  メッセージ送信者の ID（電子メール・アドレスまたは別名）を指定します。この ID は、メッセージの「From」フィールドに表示されます。送信者は必ず指定する必要があります。

- `String getSender()`

  メッセージ送信者の ID を取得します。

- `void setSession(String s)`

  JavaMail セッションの設定方法の 1 つは、JSP ページ・コンテキスト・オブジェクトにすでに存在している `javax.mail.Session` インスタンスの名前を指定する方法です。このためには、`setSession()` メソッドを使用して、そのセッション・インスタンスの名前を指定します。

  この場合は、`sendMessage()` メソッドを使用して電子メール・メッセージを送信するとき、`javax.servlet.jsp.PageContext` インスタンスを入力として指定する必要があります。

  JavaMail セッションの設定の概要は、8-20 ページの「SendMailBean の要件」を参照してください。

- `String getSession()`

  セッション・インスタンスの名前を取得します。

- `void setSubject(String s)`

  メッセージ件名の行を指定します。

- `String getSubject()`

  メッセージ件名の行を取得します。

- `void setServerAttachment(String s)`

  電子メール・メッセージに添付するサーバー・サイド・ファイルに関するカンマまたはセミコロン区切りのファイル名（パスを含む）リストを指定します。ファイルは、OC4J サーバー・マシン上にある必要があります。サーバー・サイド添付は、`sendmail.properties` ファイルで有効化する必要があります。

- `String getServerAttachment()`

  メッセージに添付するサーバー・サイド・ファイルのファイル名リストを取得します。このリストは、ユーザー確認ページなどを表示する場合に便利です。

- `void setClientAttachment(String s)`

  電子メール・メッセージに添付するクライアント・サイド・ファイル（最大 1 ファイル）のパスとファイル名を指定します。ファイルは、ユーザーのクライアント・マシン上にある必要があります。クライアント・サイド添付は、`sendmail.properties` ファイルで有効化する必要があります。

- `String getClientAttachment()`

  メッセージに添付するクライアント・サイド・ファイルの名前を取得します。この名前は、ユーザー確認ページなどを表示する場合に便利です。

注意： メール添付の関連情報は、8-19 ページの「メール添付」を参照してください。1 つのアプリケーションで使用できるのは、サーバー・サイド添付かクライアント・サイド添付のいずれかです。両方は使用できません。

---

sendMail タグの説明

JSP 開発者が便利なように、OC4J には、JSP ページに電子メール機能を設定する `sendMail` タグが用意されています。次の各項では各タグについて説明します。

- sendMail タグ構文
- sendMail タグ属性の説明
- sendMail タグに関するサンプル・アプリケーション

`sendMail` タグを使用する場合は、次の要件に注意してください。

- `ojsputil.jar`、`mail.jar` および `activation.jar` の各ファイルがインストール済で、クラスパスに存在していることを確認してください。これらのファイルは OC4J に含まれています。`ojsputil.jar` は予約済のタグ・ライブラリ・ディレクトリにあります。
- 現在の実装では、`sendMail` タグには独自のタグ・ライブラリ・ディスクリプタ・ファイル `email.tld` が含まれています。このファイルは、アプリケーションで使用可能である必要があります。また、タグを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`email.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/email.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

sendMail タグ構文

sendMail タグには、次の構文があります。

```
<mail:sendMail host = "SMTP_host_name" | session = "JavaMail_session_name"
               sender = "sender_address"
               recipient = "primary_recipient_IDs"
             [ cc = "cc_recipient_IDs" ]
             [ bcc = "bcc_recipient_IDs" ]
             [ subject = "subject_line" ]
             [ contentType = "MIME_type; [charset=charset]" ]
             [ contentEncoding = "B"|"base64"|"Q"|"quoted-printable"|
                                 "7bit"|"8bit" ]
             [ serverAttachment = "server_file_list" |
               clientAttachment = "client_file" ] >
...
E-mail body
...
</mail:sendMail>
```

sendMail タグの使用方法 sendMail タグを使用する場合は、次の点に注意してください。

- sender 属性と recipient 属性は必須です。また、host 属性または session 属性のいずれかが必須です。
- 複数の受信者、cc の宛先または bcc の宛先は、スペースまたはカンマで区切ります。
- serverAttachment を使用するには、サーバー・サイド添付が sendmail.properties ファイルで有効化されていることが前提です。同様に、clientAttachment を使用するには、クライアント・サイド添付が sendmail.properties ファイルで有効化されていることが前提です。1 つのアプリケーションで有効化できるモードは 1 つのみです。8-19 ページの「添付の有効化」を参照してください。
- serverAttachment 設定のファイル名は、カンマまたはセミコロンで区切ります。スペースで区切ることはできません。
- 電子メールのボディには、JSP 構文を含めることができます。この構文は、JSP トランスレータによって処理されます。
- タグが使用する属性は、通常、ユーザーがフォーム・フィールドに入力します。すべての属性が、リクエスト時の式を受け入れます。
- このタグ構文では、接頭辞「mail:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

sendMail タグ属性の説明

sendMail タグは、次の属性をサポートします。

- host（session が指定されていない場合は必須）：「gmail.oraclecorp.com」などの適切なメール・ホスト名です。メール・メッセージに対する JavaMail セッション・オブジェクトの作成時に使用されます。JavaMail セッションは、session 属性を使用して決定することもできます。
- session（host が指定されていない場合は必須）：JSP ページ・コンテキストから取得できる既存の JavaMail セッション・オブジェクトの名前です。JavaMail セッションは、host 属性を使用して決定することもできます。
- sender（必須）：メッセージ送信者の ID（電子メール・アドレスまたは別名）です。この ID は、メッセージの「From」フィールドに表示されます。
- recipient（必須）：メッセージのプライマリ受信者の ID をスペースまたはカンマで区切ったリストです。これらの ID は、メッセージの「To」フィールドに表示されます。

- cc: メッセージのコピーを受信する ID のスペースまたはカンマ区切りのリストです。これらの ID は、メッセージの「Cc」フィールドに表示されます。
- bcc: メッセージのブラインド・コピーを受信する ID のスペースまたはカンマ区切りのリストです。これらの ID は、メッセージの「Cc」フィールドには表示されません。
- subject: メッセージ件名の行です。
- contentType: メッセージの MIME タイプです。キャラクタ・セットも必要に応じて指定できます。次に例を示します。

  ```
  contentType="text/html"

  contentType="text/html; charset=US-ASCII"
  ```

  デフォルトの MIME タイプの設定は、「text/plain」です。ただし、この MIME タイプまたは他の text/*xxxx* MIME タイプを明示的に指定しない場合、キャラクタ・セットを指定できません。

  デフォルトのキャラクタ・セットは、JSP ページ・コンテキストの response オブジェクトのキャラクタ・セットです。

- contentEncoding: base64 エンコードには「B」または「base64」を、quoted-printable エンコードには「Q」または「quoted-printable」を、7 ビット・エンコードには「7bit」を、8 ビット・エンコードには「8bit」をそれぞれ指定します。これらは、標準の JavaMail エンコーディングおよび RFC 2047 エンコーディングです。エントリには、大 / 小文字区別がありません。

  デフォルトのコンテンツ・エンコーディングの設定は、「null」です。この場合、メッセージとヘッダーのエンコーディングは、コンテンツによって決まります。エンコードされる文字の大部分が ASCII の場合は、quoted-printable エンコードが使用されます。それ以外の場合は、base64 エンコードが使用されます。

- serverAttachment: 電子メール・メッセージに添付するサーバー・サイド・ファイルのカンマまたはセミコロン区切りのリストです。サーバー・サイド添付は、sendmail.properties ファイルで有効化する必要があります。

  次に例を示します。

  ```
  serverAttachment="/tmp/confirm.pdf,/home/schedule.doc"
  ```

- clientAttachment: 電子メール・メッセージに添付するクライアント・サイド・ファイル（最大 1 ファイル）の名前です。クライアント・サイド添付は、sendmail.properties ファイルで有効化する必要があります。

  次に例を示します。

  ```
  clientAttachment="c:¥finance¥budget02.xls"
  ```

注意： 電子メール添付の関連情報は、8-19 ページの「メール添付」を参照してください。1 つのアプリケーションで使用できるのは、サーバー・サイド添付かクライアント・サイド添付のいずれかです。両方は使用できません。

---

sendMail タグに関するサンプル・アプリケーション

次に、sendMail タグを使用したサンプル・アプリケーションを示します。添付はありません。ページでの最初の実行サイクルでは、ユーザーが送信者（またはその他）を指定する前に、ユーザー入力のための HTML フォームが表示されます。ページでの次の実行サイクルでは、ユーザーが入力を送信した後に、sendMail タグが実行されます。このページでは、スローされた例外を表示するためのエラー・ページ error.jsp（後述）も使用されます。

```
<%@ page language="java" errorPage="error.jsp" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/email.tld"
           prefix="mail" %>
<%
if (request.getParameter("sender")==null) {
%>
<HTML>
<HEAD><TITLE>SendMail Sample</TITLE></HEAD>
<FORM METHOD=post>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="20%">
<TR><TD>Host:</TD><TD><INPUT TYPE="text" name="host" ></TD></TR>
<TR><TD>From:</TD><TD><INPUT TYPE="text" name="sender" ></TD></TR>
<TR><TD>To:</TD><TD><INPUT TYPE="text" name="recipient"  ></TD></TR>
<TR><TD>Cc:</TD><TD><INPUT TYPE="text" name="cc" ></TD></TR>
<TR><TD>Bcc:</TD><TD><INPUT TYPE="text" name="bcc" ></TD></TR>
<TR><TD>Subject:</TD><TD><INPUT TYPE="text" name="subject"
VALUE="Hi"></TD></TR>
</TABLE><br>
<TEXTAREA name="body" ROWS=4 COLS=30>"How are you!"</TEXTAREA><br><br>
<INPUT TYPE="submit" value="Send">
</FORM>
<%
}
else{
%>
<BODY BGCOLOR="#FFFFFF">
<P>Result:
       <HR>
       <mail:sendMail host='<%=request.getParameter("host")%>'
             sender='<%=request.getParameter("sender")%>'
             recipient='<%=request.getParameter("recipient")%>'
             cc='<%=request.getParameter("cc")%>'
             bcc='<%=request.getParameter("bcc")%>'
             subject='<%=request.getParameter("subject")%>'>
             <%=request.getParameter("body")%>
       </mail:sendMail>
Sent out Successfully!
       <HR>
</BODY>
<%
}
%>
</HTML>
```

次にエラー・ページ error.jsp を示します。

```
<%@ page language="java" isErrorPage="true"%>
<HTML>
Error: <%= exception.getMessage() %>
</HTML>
```

このアプリケーションを実行すると、図 8-1 に示すデフォルトの画面が最初に表示されます。

図 8-1 デフォルトの入力テキストを含んだメッセージ

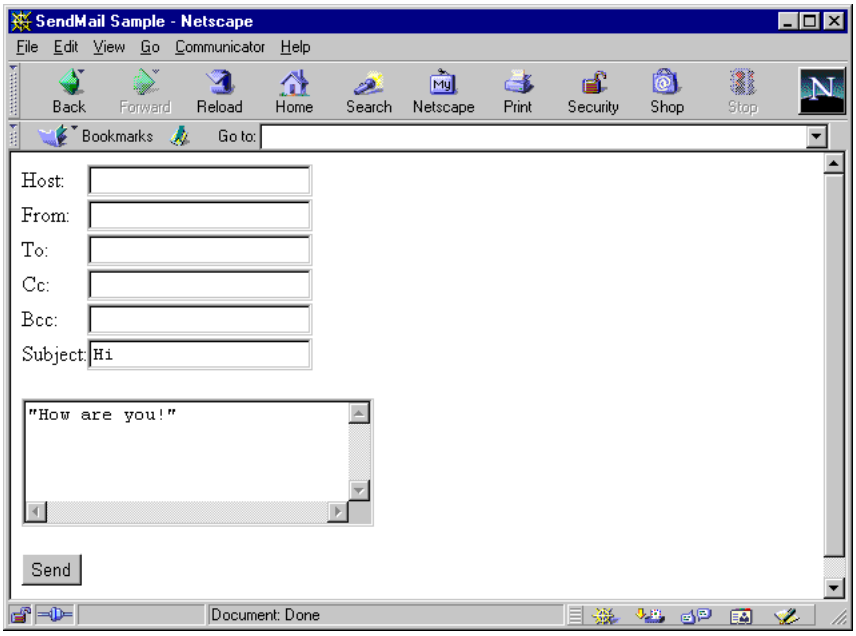
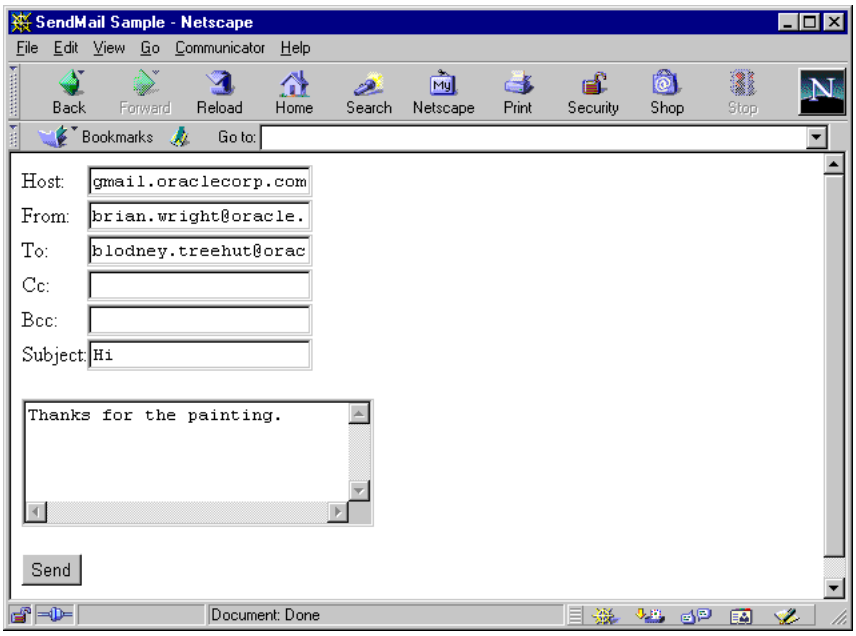


図 8-2 は、ホスト gmail.oraclecorp.com を介して brian.wright@oracle.com から blodney.treehut@oracle.com に送信されるメッセージのユーザー入力を示しています。

図 8-2 ユーザー入力を含んだメッセージ



9

JSP のユーティリティとユーティリティ・タグ

この章では、JSP ページで使用できる各種の OC4J ユーティリティ機能について説明します。この章の構成は次のとおりです。

- JspScopeListener による JSP のイベント処理
- EJB タグ
- 一般的なユーティリティ・タグ

JspScopeListener による JSP のイベント処理

> **重要：** JspScopeListener は OC4J 10.1.2 実装で廃止され、それ以降の実装でサポート外になる予定です。ほとんどの機能は、J2EE 1.4 以上の標準機能に含まれています。

標準サーブレットと JSP テクノロジでサポートされるのは、セッション・ベースのイベントのみです。Oracle では、このサポートを、oracle.jsp.event パッケージの JspScopeListener インタフェースと JspScopeEvent クラスを使用して、ページ・ベース、リクエスト・ベースおよびアプリケーション・ベースのイベントにまで拡張します。

次の各項では、JspScopeListener 機能について説明して例を示します。

- JspScopeListener の一般的な使用
- OC4J および他のサーブレット 2.3 環境での JspScopeListener の使用
- JspScopeListener の使用例

JspScopeListener の一般的な使用

アプリケーション内の Java オブジェクトの場合は、JspScopeListener インタフェースを適切なクラスに実装してから、そのクラスのオブジェクトを jsp:useBean などのタグを使用して、JSP スコープに連結します。

スコープの最後に到達すると、JspScopeListener を実装したオプションとそのスコープに連結されたオブジェクトが通知されます。JSP コンテナでは、JspScopeEvent インスタンスを JspScopeListener インタフェースに指定されている outOfScope() メソッドを使用して、該当するオブジェクトに送信することによって、この通知を実行します。

このイベント・リスナーの機能は、page スコープや request スコープに関するオブジェクト・リソースを、エラー状態に関係なく常に解放しておこうとする開発者にとっては、大きな利点となります。この機能によって、開発者は、自分のページ実装を Java の try/catch/finally ブロックで囲む必要がなくなります。

JspScopeEvent オブジェクトのプロパティには、次の内容が含まれています。

- 終了するスコープ（int 型の定数 PAGE_SCOPE、REQUEST_SCOPE、SESSION_SCOPE または APPLICATION_SCOPE のいずれかで表します。）

  このスコープは次の JspScopeEvent メソッドで取得できます。

  ```
  public int getScope()
  ```

- このスコープでのオブジェクトに対するリポジトリとなるコンテナ・オブジェクト（暗黙的なオブジェクトの page、request、session または application のいずれか）

  これは関連スコープを管理するオブジェクトです。このオブジェクトは次の JspScopeEvent メソッドで取得できます。

  ```
  public java.lang.Object getContainer()
  ```

- 通知に関連するオブジェクト名

  これは JspScopeListener を実装するクラスのインスタンス名です。このクラスのインスタンスは page、request、session または application オブジェクト（使用可能な場合）の属性であるため、このインスタンス名は属性名です。この名前は次の JspScopeEvent メソッドで取得できます。

  ```
  public String getName()
  ```

- JSP の暗黙的な application オブジェクト

  このオブジェクトは次の JspScopeEvent メソッドで取得できます。

  ```
  public ServletContext getApplication()
  ```

JspScopeEvent クラスには次のコンストラクタがあります。

```
public JspScopeEvent (ServletContext sc, Object container, String name,
                      int scope)
```

OC4J および他のサーブレット 2.3 環境での JspScopeListener の使用

JspScopeListener は、様々な機能を使用して異なるスコープをサポートします。ただし、すべてはサーブレットと JSP の標準に基づいて実装されます。

OC4J 環境でページを実行する場合は、便宜上、page スコープに対する OC4J 固有の実行時実装もあります。

これらの機能については次の項を参照してください。

- JspScopeListener の要件
- ページ・スコープをサポートする実行時実装およびタグの実装
- リクエスト・スコープをサポートするサーブレット・フィルタの実装
- アプリケーション・スコープをサポートするリスナー・クラスの実装
- セッション・スコープをサポートする HttpSessionBindingListener との統合

JspScopeListener の要件

JspScopeListener の実装には、次の内容が必要です。

- oracle.jsp.event.JspScopeListener インタフェースと JspScopeEvent クラス、および oracle.jsp.event.impl パッケージのクラス。これらは、すべて ojsp.jar ファイルに含まれています。
- サーブレット 2.3 以上の環境（OC4J など）

ページ・スコープをサポートする実行時実装およびタグの実装

OC4J 環境では、Oracle 固有の実行時実装によって、page スコープ機能がサポートされます。この機能を有効化するには、JSP check_page_scope 構成パラメータを true に設定します。デフォルトは、パフォーマンス上の理由から、false です。

他の環境への移植性については、カスタム・タグ checkPageScope を使用して、page スコープをサポートする実装もあります。適切なコードを checkPageScope の開始タグと終了タグの間に配置します。このタグは、属性なしで、次のように定義します。

```
<!-- The checkPageScope tag -->
<tag>
   <name>checkPageScope</name>
   <tagclass>oracle.jsp.jml.tagext.CheckPageScopeListenerTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>
      To provide the notification logic for any
      JspScopeListener stored in page scope.
      This tag is not needed on OC4J.
   </info>
</tag>
```

次にこのタグの使用例を示します。

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/jml.tld"
           prefix="jml" %>
<jml:checkPageScope>
pagescope.jsp
<jsp:useBean id="tb" class="testpkg.TestData" />
<%
   /* testpkg.TestData implements oracle.jsp.event.JspScopeListener.
      checkPageScope tag will provide the notification logic for any
      JspScopeListener stored in page scope.
      This tag is not needed on OC4J.
   */
   // some more JSP / code here ...
%>
<%= new java.util.Date() %>
</jml:checkPageScope>
```

注意： checkPageScope タグは、現在 Oracle JML タグ・ライブラリに含まれています。このタグ・ライブラリは ojsputil.jar ファイルにあり、jml.tld タグ・ライブラリ・ディスクリプタ・ファイルが必要です。適切な taglib ディレクティブは、前述の例を参照してください。関連情報は、3-2 ページの「JSP Markup Language（JML）タグ・ライブラリの概要」を参照してください。

リクエスト・スコープをサポートするサーブレット・フィルタの実装

request スコープのオブジェクトは、サーブレット・フィルタによってサポートされています。フィルタ処理は、指定した URL パターンに一致するすべてのサーブレットに適用されます。

リクエスト・スコープ・オブジェクトのイベント処理をサポートするには、次のようなエントリをアプリケーションの web.xml ファイルに追加するか、必要に応じて orion-web.xml または global-web-application.xml ファイルに追加します。JspScopeListener 機能を適切に操作するには、この設定を他のすべての filter 設定の後に指定する必要があります。

```
<filter>
   <filter-name>Request Filter</filter-name>
   <filter-class>oracle.jsp.event.impl.RequestScopeFilter</filter-class>
</filter>
<!-- Define filter mappings for the defined filters -->
<filter-mapping>
   <filter-name>Request Filter</filter-name>
   <url-pattern>/jsp/*</url-pattern>
</filter-mapping>
```

注意： この特殊な例では、フィルタが処理する URL パターンは「/jsp/*」です。ユーザーは、「/*.jsp」や「/*」などの他のパターンを選択できます。

アプリケーション・スコープをサポートするリスナー・クラスの実装

application スコープを含むオブジェクトは、Servlet 仕様に従って、サーブレット・コンテキスト・リスナー実装クラスによってサポートされます。

アプリケーション・スコープ・オブジェクトのイベント処理をサポートするには、次のようなエントリをアプリケーションの web.xml ファイルに追加します。JspScopeListener 機能を適切に操作するには、この設定を他のすべての listener 設定の後に指定する必要があります。

```
<listener>
   <listener-class>oracle.jsp.event.impl.AppScopeListener</listener-class>
</listener>
```

アプリケーション・スコープ・オブジェクトの場合は、アプリケーションとサーブレット・コンテキストの終了時の通知に加えて、属性がサーブレット・コンテキストで置換されたとき、またはサーブレット・コンテキストから削除されたときも、通知が行われます。たとえば、アプリケーション・スコープ・オブジェクトのリスナー outOfScope() メソッドが次の状況でコールされたとします。この場合のサーブレット・コンテキスト・オブジェクトは、ctx です。

```
ctx.setAttribute("name", "Smith");
...
ctx.setAttribute("name, "Jones");
```

または

```
ctx.setAttribute("name", "Smith");
...
ctx.removeAttribute("name");
```

> **注意：** この機能は、Oracle9*i*AS リリース 2 より前では使用できませんでした。

セッション・スコープをサポートする HttpSessionBindingListener との統合

セッション・スコープ・オブジェクトの場合は、JspScopeListener インタフェースと標準の javax.servlet.http.HttpSessionBindingListener インタフェースの両方を実装するクラスを作成できます。この機能によって、他のスコープでも、このクラスのインスタンスをサポートする柔軟性が得られます。ただし、インスタンスを session スコープ以外で使用しない場合、JspScopeListener の実装は不要です。

統合の使用例では、HttpSessionBindingListener インタフェースに指定されている valueUnbound() メソッドが、JspScopeListener インタフェースに指定されている outOfScope() メソッドをコールする必要があります。

次に、基本的な例を示します。

```
import oracle.jsp.event.impl.*;
import javax.servlet.*;
import javax.servlet.http.*;

class SampleObj implements HttpSessionBindingListener,JspScopeListener
{
   public void valueBound(HttpSessionBindingEvent e)
   {
      System.out.println("The object implements the JspScopeListener also");
   }

   public void valueUnBound(HttpSessionBindingEvent e)
   {
      try
      {
         outOfScope(new JspScopeEvent(null,(Object)e.getSession(),
```

```
            e.getName(),javax.servlet.jsp.PageContext.SESSION_SCOPE));
      } catch (Throwable e) {}
   ...........
   }
   public void outOfScope(JspScopeEvent e)
   {...}
}
```

JspScopeListener の使用例

この項では、JspScopeListener の使用方法について 2 つの例を示します。最初は JSP ページと付属の JavaBean の例、次にサーブレットの例を示します。

例 : JspScopeListener を使用した JSP ページ

この例は、JspScopeListener インタフェースを実装する JavaBean の ScopeDispatcher と、リクエスト・スコープ機能とアプリケーション・スコープ機能に ScopeDispatcher インスタンスを使用する JSP ページで構成されています。

bookcatalog.jsp bookcatalog.jsp ページによって、ユーザーはカタログで書籍を検索したり、新しい書籍のエントリを挿入することができます。このカタログは、最初にローカル・ファイル・ストリームから読み取られたハッシュテーブルに保存されています。

リクエストの終了時に、新しい書籍が発行されている場合、1）その書籍はアプリケーション・レベルの catalog ハッシュテーブルに入力され、2）書籍の数が増加します。

アプリケーションの実行の終了時に、catalog ハッシュテーブルは、ローカル・ファイル・ストリームに戻され、新たに挿入された書籍数が表示されます。問合せ結果は、書籍検索が実行された場合に表示されます。

```
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%! static int newbookCount = 0; %>
<%! static Hashtable catalog; %>
<%! boolean bookAdded = false; %>
<html>
<head>
<title> BookStore Price catalog </title>
</head>
<body bgcolor="white">
<font size=5 color="red">
<table color="#FFFFCC" width="100%" border="1" cellspacing="0" cellpadding="0" >
<tr>
<td>
<form action="bookcatalog.jsp">
<b> BookName </b>
<input type="text" name="bookname">
<input type="submit" value="Get the Price">
</form>
</td>
<td>
<form action="bookcatalog.jsp">
<b>BookName</b>
<input type="text" name="new_book">
<br>
<b>Price</b>
<input type="text" name="price">
<input type="submit" value="Add to Catalog">
</form>
</td>
</tr>
</table>
```

```
<%
  String bookname = request.getParameter("bookname");
  catalog = (Hashtable) application.getAttribute("pricelist");
  if (catalog == null)
  {
   try{
    ObjectInputStream oin = new ObjectInputStream
                            (new FileInputStream("bookcatalog.out"));
    Object obj =  oin.readObject();
    catalog = (Hashtable) obj;
    oin.close();
   }
  catch(Exception e) {
   catalog = new Hashtable();}
  application.setAttribute("pricelist",catalog);
 }
  if (bookname != null)
  {
    String price = (String) catalog.get(bookname.trim());
    if (price != null)
    {
     out.println("<h2>Book : " +bookname+ "</h2>");
     out.println("<h2>Price: "+price +"</h2>");
    }
    else
      out.println("<h2> Sorry, the  Book : " + bookname + " is not available in
                   the catalog</h2>");
  }
%>

<%-- declare the event dispatchers --%>
<jsp:useBean id = "requestDispatcher"
             class = "oracle.jsp.sample.event.ScopeDispatcher"
             scope = "request" >
   <jsp:setProperty name = "requestDispatcher" property = "page"
                    value = "<%= this %>" />
   <jsp:setProperty name = "requestDispatcher" property = "methodName"
                    value = "request_OnEnd" />
</jsp:useBean>

<jsp:useBean id = "appDispatcher"
             class = "oracle.jsp.sample.event.ScopeDispatcher"
             scope = "application" >
   <jsp:setProperty name = "appDispatcher" property = "page"
                    value = "<%= this %>" />
   <jsp:setProperty name = "appDispatcher" property = "methodName"
                    value = "application_OnEnd" />
</jsp:useBean>
<%!
   // request_OnEnd Event Handler
   public void request_OnEnd(HttpServletRequest request) {
      // acquire beans
               String newbook  = request.getParameter("new_book");
               bookAdded = false;
               if ((newbook != null) && (!newbook.equals("")))
               {
                  catalog.put(newbook,request.getParameter("price"));
                  newbookCount++;
                  bookAdded = true;
               }
   }
%>
```

```
<%!
  public void application_OnEnd(ServletContext application)
  {
    try
    {
      ObjectOutputStream os = new ObjectOutputStream(
                                  new FileOutputStream("bookcatalog.out"));
      os.writeObject(catalog);
      os.flush();
      os.close();
    }
    catch (Exception e)
    {}
  }
%>

<%
if (bookAdded)
   out.println("<h2> The New book is been added in the catalog </h2>");
%>
<%-- Page implementation goes here --%>
<h2> Total number of books added is <%= newbookCount %></h2>
</font>
</body>
</html>
```

ScopeDispatcher.java

```
package oracle.jsp.sample.event;
import java.lang.reflect.*;
import oracle.jsp.event.*;

public class ScopeDispatcher extends Object implements JspScopeListener {
    private Object page;
    private String methodName;
    private Method method;

    public ScopeDispatcher() {
    }

    public Object getPage() {
        return page;
    }

    public void setPage(Object page) {
        this.page = page;
    }

    public String getMethodName() {
        return methodName;
    }

    public void setMethodName(String m) throws NoSuchMethodException,
                                               ClassNotFoundException  {
        method = verifyMethod(m);
        methodName = m;
    }

    public void outOfScope(JspScopeEvent ae) {
        int scope = ae.getScope();

        if ((scope == javax.servlet.jsp.PageContext.REQUEST_SCOPE  ||
             scope == javax.servlet.jsp.PageContext.APPLICATION_SCOPE)
```

```
             && method != null) {
            try {
                Object args[] = {ae.getContainer()};
                method.invoke(page, args);
            } catch (Exception e) {
                // catch all and continue
            }
        }
    }

    private Method verifyMethod(String m) throws NoSuchMethodException,
                                                 ClassNotFoundException {
        if (page == null) throw new NoSuchMethodException(
                                  "A page hasn't been set yet.");

        // Don't know whether this is a request or page handler so try one then
       // the other
        Class c = page.getClass();
        Class pTypes[] = {Class.forName("javax.servlet.ServletContext")};

        try {
            return c.getDeclaredMethod(m, pTypes);
        } catch (NoSuchMethodException nsme) {
            // fall through and try the request signature
        }

        pTypes[0] = Class.forName("javax.servlet.http.HttpServletRequest");
        return c.getDeclaredMethod(m, pTypes);
    }
}
```

例 : JspScopeListener を使用したサーブレット

この項では、リクエスト・スコープ・オブジェクトの JspScopeListener 機能を使用するサンプル・サーブレットを説明します。ネストしたクラス DBScopeObj によって、JspScopeListener インタフェースが実装されます。

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.jsp.event.*;
import oracle.jsp.event.impl.*;

public class RequestScopeServlet extends HttpServlet {

    PrintWriter out;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        out         = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title> RequestScopeServlet! </title>");
        out.println("</head>");
        response.setContentType("text/html");
        DBScopeObj aobj = new DBScopeObj();
        request.setAttribute("dbcon",aobj);
        request.setAttribute("name","scott");
        request.setAttribute("company","oracle");
        request.setAttribute("city","sanmateo");
```

```
        Enumeration en = request.getAttributeNames();
        out.println("<BR> Request Attributes : <BR> <BR>");
        while (en.hasMoreElements()) {
            String key = (String)en.nextElement();
            Object value = request.getAttribute(key);
            out.println(key + "   : " + value+"<BR>");
         }
        out.println("</body>");
        out.println("</html>");
    }

    class DBScopeObj implements JspScopeListener
    {
       public void initDBConnection()
       {
            //  can create a minimum number of predefined
            // DBConnections
       }

       DBScopeObj()
       {
           // if DBconnection is available in the connection
           // pool then pickup from the pool and give the handle.
       }

       public void outOfScope(JspScopeEvent e)
       {
        ServletContext ctx = e.getApplication();
        out.println
            ("<BR>**************************************************");
        out.println("<BR> JspScopeEvent <BR>");
        out.println("<BLINK>");
        out.println
            ("<BR> In outOfScope method for the Request Attribute <BR>");
        out.println("Name =  " +e.getName() + "<BR>");
        out.println("</BLINK>");
        out.println
            ("**************************************************<BR>");
        // logging in the context also

        ctx.log("**************************************************");
        ctx.log(" JspScopeEvent ");
        ctx.log(" In outOfScope method for the Request Attribute ");
        ctx.log("Name =  " +e.getName());
        ctx.log("**************************************************");
        returnDBConnection();
       }

       public void returnDBConnection()

       {
           //Can return the handle to the connection pool
       }
    }
}
```

EJB タグ

OC4J には、JSP ページでの Enterprise JavaBeans の使用を簡素化するカスタム・タグ・ライブラリが用意されています。このライブラリには、ホーム・インスタンスの作成、EJB インスタンスの作成および EJB のコレクション間の反復などに使用するタグが含まれています。

OC4J EJB タグの機能は、J2EE 仕様に従っています。このタグを使用すると、`web.xml` ファイルの構成情報を使用して、EJB を名前でインスタンス化できます。タグの中には、通常の JavaBean の起動に使用する `jsp:useBean` タグの機能に似た機能を持つ `useBean` タグがあります。

次の各項では、各タグについて説明し、例を示します。

EJB タグの構成

使用する各 EJB について、アプリケーションの `web.xml` ファイルにある `<ejb-ref>` 要素を、次の例のように使用します。

```
<ejb-ref>
   <ejb-ref-name>ejb/DemoSession</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>ejbdemo.DemoSessionHome</home>
   <remote>ejbdemo.DemoSession</remote>
</ejb-ref>
```

`<ejb-ref>` 要素とそのサブ要素、またはローカル・インタフェースを使用するための `<ejb-local-ref>` は、Servlet 仕様に準拠して使用されます。概要は、次のとおりです。

- `<ejb-ref-name>` サブ要素は、J2EE アプリケーションの他のコンポーネントがこのコンポーネントへのアクセスに使用できる参照名を指定します。たとえば、この名前を場所を示す値で使用できます。
- `<ejb-ref-type>` サブ要素は、EJB のカテゴリを指定します。
- `<home>` サブ要素は、EJB ホーム・インタフェースのパッケージと型を指定します。または、EJB ローカル・インタフェースの場合は `<local-home>` サブ要素を使用します。
- `<remote>` サブ要素は、EJB リモート・インタフェースのパッケージと型を指定します。または、EJB ローカル・インタフェースの場合は `<local>` サブ要素を使用します。

これらの値は、EJB タグの属性値に反映されます。

EJB の開発と構成の追加情報は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

EJB タグの説明

この項では、OC4J EJB タグの構文と属性を説明します。次の要件に注意してください。

- ojsputil.jar ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J に同梱されていて、予約済のタグ・ライブラリ・ディレクトリにあります。
- タグ・ライブラリ・ディスクリプタ・ファイル ejbtaglib.tld が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な taglib ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は ojsputil.jar に配置されます。ejbtaglib.tld の uri 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/ejbtaglib.tld
  ```

taglib ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび uri 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「ejb:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

次の各項では、EJB タグの詳細を説明します。

EJB インスタンスを最初に作成するときは、useHome タグを使用してホーム・インタフェース・インスタンスを作成する必要があります。その後、適宜、次のインスタンスを作成します。

- 単一 ELB インスタンスを作成する場合：useBean タグと、useBean タグの value 属性またはネストした createBean タグ
- EJB インスタンスのコレクションを作成し、そのインスタンス間（主として Entity Bean の場合）を反復する場合：iterate タグ

EJB インスタンスは、作成された後、適切なスコープ・オブジェクトに入れられます。その後でこのインスタンスにアクセスするために必要なのは、useBean タグのみです。

EJB useHome タグ

useHome タグは、EJB のホーム・インタフェースを参照して、インスタンスを作成します。

構文

```
<ejb:useHome id = "home_instance_name"
             type = "home_interface_type"
             location = "home_lookup_name"
           [ local = "true" | "false" ] />
```

このタグは、ボディを使用しません。

属性

- id（必須）: ホーム・インタフェース・インスタンスの名前を指定します。これは、local 属性の設定に応じて、ローカル・ホーム・インタフェースまたはリモート・ホーム・インタフェースのいずれかです。このインスタンスには、開始タグからページの終わりまでアクセスできます。
- type（必須）: ホーム・インタフェースの名前（Java の型）です。
- location（必須）: アプリケーション内の任意の EJB のホーム・インタフェースをルックアップするために使用する JNDI の名前です。
- local: ローカル・ホーム・インタフェースを使用するには「true」に設定します。デフォルト値は「false」で、リモート・ホーム・インタフェースが使用されます。useHome タグで local="true" に設定する場合は、useBean タグでも同じように設定する必要があります。

例

```
<ejb:useHome id="aomHome" type="com.acme.atm.ejb.AccountOwnerManagerHome"
             location="java:comp/env/ejb/accountOwnerManager" />
```

EJB useBean タグ

EJB useBean タグは、EJB をインスタンス化して使用するためのタグです。id、type および scope の各属性は、通常の JavaBean をインスタンス化する標準の jsp:useBean タグの場合と同じように使用します。

EJB を最初にインスタンス化する場合は、次の 2 つの機能のいずれかを使用できます。

- value 属性

または

- ネストした EJB createBean タグ

ネストした createBean タグを使用している場合、EJB インスタンスは、親 useBean タグの value 属性に暗黙的に戻されます。一度 EJB がインスタンス化されると、value 属性とネストした createBean タグは、同じ EJB インスタンスを使用する後続の useBean タグには不要になります。

注意： EJB インスタンスのコレクションの使用方法については、9-14 ページの「EJB iterate タグ」を参照してください。

構文

```
<ejb:useBean id = "EJB_instance_name"
             type = "EJB_class_name"
           [ value = "<%=Object%>" ]
           [ scope = "page" | "request" | "session" | "application" ]
           [ local = "true" | "false" ] >

... nested createBean tag for first instantiation, if no value attribute ...

</ejb:useBean>
```

属性

- id（必須）: EJB のインスタンス名を指定します。
- type（必須）: EJB のクラス名を指定します。
- value: EJB を最初にインスタンス化するときに、ネストした createBean タグを使用しない場合は、value 属性を使用して EJBObject インスタンスを戻し、ナローイングを行います。これは、EJB をインスタンス化する機能です。
- scope: EJB インスタンスのスコープを指定します。デフォルトのスコープ設定は「page」です。
- local: ローカル・ホーム・インタフェースを使用するには「true」に設定します。デフォルト値は「false」で、リモート・ホーム・インタフェースが使用されます。useBean タグで local="true" に設定する場合は、useHome タグでも同じように設定する必要があります。

> **注意：** 配布可能アプリケーションで scope="session" に設定されている場合は、local="true" を使用できません。

例 次に、すでにインスタンス化されている EJB の使用例を示します。

```
<ejb:useBean id="bean" type="com.acme.MyBean" scope="session" />
```

EJB createBean タグ

EJB を最初にインスタンス化するときに、EJB useBean タグの value 属性を使用しない場合は、EJB createBean タグを useBean タグ内にネストし、EJB インスタンスの作成作業を実行する必要があります。これが、EJBObject インスタンスになります。このインスタンスは、親 useBean タグの value 属性に暗黙的に戻されます。

構文

```
<ejb:createBean instance = "<%=Object%>" />
```

このタグは、ボディを使用しません。

属性

- instance（必須）:EJB（作成された EJBObject インスタンス）を戻します。

例 この createBean タグ内で、EJB ホーム・インタフェース・インスタンスの create() メソッドが、EJB のインスタンスを作成します。

```
<ejb:useBean id="bean" type="com.acme.MyBean" scope="session">
     <ejb:createBean instance="<%=home.create()%>" />
</ejb:useBean>
```

EJB iterate タグ

このタグを使用して EJB インスタンスのコレクション間を反復します。このタグは主として Entity Bean に使用されます。これは、Entity Bean の標準の finder メソッドがコレクションを戻すためです。

開始タグでは、finder メソッドの結果を介してコレクションがホーム・インタフェースから取得されます。タグ・ボディでは、必要に応じてコレクションが反復されます。

> **注意：** 単一 EJB インスタンスの使用方法については、9-13 ページの「EJB useBean タグ」を参照してください。

構文

```
<ejb:iterate id = "EJB_instance_name"
             type = "EJB_class_name"
             collection = "<%=Collection%>"
           [ max = "<%=Integer%>" ] >

... body ...

</ejb:iterate>
```

ボディは、コレクション内の各 EJB について一度評価されます。

属性

- id（必須）: イテレータ変数、つまり各反復の EJB インスタンス名です。
- type（必須）: EJB のクラス名です。
- collection（必須）: EJB コレクションを戻します。
- max: 反復する Bean の最大数を必要に応じて指定します。

例

```
<ejb:iterate id="account" type="com.acme.atm.ejb.Account"
             collection="<%=accountManager.getOwnerAccounts()%>"
             max="100">
     <jsp:getProperty name="account" property="id" />
</ejb:iterate>
```

EJB タグの例

この項では、EJB タグの使用例を示します。1 つは Session Bean の使用例、もう 1 つは Entity Bean の使用例です。

EJB タグの Session Bean の例

この例は、アプリケーションの web.xml ファイルの次の構成に依存します。

```
<ejb-ref>
   <ejb-ref-name>ejb/DemoSession</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>ejbdemo.DemoSessionHome</home>
   <remote>ejbdemo.DemoSession</remote>
</ejb-ref>
```

サンプル・コードは、次のとおりです。

```
<%@ page import="ejbdemo.*" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/ejbtaglib.tld"
           prefix="ejb" %>
<html>
<head> <title>Use EJB from JSP</title> </head>
<body>

<ejb:useHome id="home" type="ejbdemo.DemoSessionHome"
             location="java:comp/env/ejb/DemoSession" />
<ejb:useBean id="demo" type="ejbdemo.DemoSession" scope="session" >
    <ejb:createBean instance="<%=home.create()%>" />
</ejb:useBean>
<heading2>      Enterprise Java Bean:  </heading2>
 <p><b> My name is "<%=demo.getName()%>". </b></p>
</body>
</html>
```

このサンプル・コードによって、次の内容が実行されます。

- EJB ホーム・インタフェースの home インスタンスを作成します。useHome タグの type 値は、web.xml ファイルにある <ejb-ref> 要素の <home> サブ要素の値と一致しています。また、useHome タグの location 値は、<ejb-ref> 要素の <ejb-ref-name> サブ要素の値を反映しています。
- home.create() メソッドを使用して、EJB の demo インスタンスを作成します。useBean タグの type 値は、web.xml ファイルにある <ejb-ref> 要素の <remote> サブ要素の値と一致しています。
- demo.getName() メソッドを使用してユーザー名を出力します。

EJB タグの Entity Bean の例

この例は、アプリケーションの web.xml ファイルの次の構成に依存します。

```
<ejb-ref>
   <ejb-ref-name>ejb/DemoEntity</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>ejbdemo.DemoEntityHome</home>
   <remote>ejbdemo.DemoEntity</remote>
</ejb-ref>
```

サンプル・コードは、次のとおりです。

```
<%@ page import="ejbdemo.*" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/ejbtaglib.tld"
           prefix="ejb" %>
<html>
<head> <title>Iterate over EJBs from JSP</title> </head>
<body>

<ejb:useHome id="home" type="ejbdemo.DemoEntityHome"
             location="java:comp/env/ejb/DemoEntity" />
<% int i=0; %>
<ejb:iterate id="demo" type="ejbdemo.DemoEntity"
             collection="<%=home.findAll()%>" max="3" >
<li> <heading2> Bean #<%=++i%>:  </heading2>
 <b> My name is "<%=demo.getName()+"_"+ demo.getId()%>". </b> </li>
</ejb:iterate>
</body>
</html>
```

このサンプル・コードによって、次の内容が実行されます。

- EJB ホーム・インタフェースの home インスタンスを作成します。useHome タグの type 値は、web.xml ファイルにある <ejb-ref> 要素の <home> サブ要素の値と一致しています。また、useHome タグの location 値は、<ejb-ref> 要素の <ejb-ref-name> サブ要素の値を反映しています。
- home.findAll() メソッドを使用して EJB のコレクションを戻します。iterate タグの type 値は、web.xml ファイルにある <ejb-ref> 要素の <remote> サブ要素の値と一致しています。
- コレクション間を反復します。現行のインスタンスの demo を必ず使用します。また、demo.getName() メソッドと demo.getId() メソッドを使用して各 EJB の情報を出力します。

一般的なユーティリティ・タグ

OC4J には、多様な操作を実行するその他のユーティリティ・タグが多数あります。次の各項ではタグの詳細を説明します。

- Display タグ
- その他のユーティリティ・タグ

ユーティリティ・タグを使用する場合は、次の要件に注意してください。

- `ojsputil.jar` ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J に同梱されていて、予約済のタグ・ライブラリ・ディレクトリにあります。
- タグ・ライブラリ・ディスクリプタ・ファイル `utiltaglib.tld` が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な `taglib` ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は `ojsputil.jar` に配置されます。`utiltaglib.tld` の `uri` 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/utiltaglib.tld
  ```

`taglib` ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび `uri` 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「util:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を `taglib` ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

Display タグ

次の各項では、display タグの構文と属性について説明します。

- displayCurrency ユーティリティ・タグ
- displayDate ユーティリティ・タグ
- displayNumber ユーティリティ・タグ

displayCurrency ユーティリティ・タグ

指定した金額を、ロケールに適切な通貨として書式化して表示します。ロケールを指定しないと、`request` オブジェクトがロケールのかわりに検索されます。オブジェクトが検出されない場合は、システムのデフォルトのロケールが使用されます。

構文

```
<util:displayCurrency amount = "<%=Double%>"
                      [ locale = "<%=Locale%>" ] />
```

このタグは、ボディを使用しません。

属性

- `amount`（必須）：書式化する金額を指定します。
- `locale`: 必要に応じて、ロケールを `java.util.Locale` インスタンスとして指定します。

例

```
<util:displayCurrency amount="<%=account.getBalance()%>"
                      locale="<%=account.getLocale()%>" />
```

displayDate ユーティリティ・タグ

指定した日付をロケールに適した書式で表示します。ロケールを指定しない場合は、システムのデフォルトのロケールが使用されます。

構文

```
<util:displayDate date = "<%=Date%>"
                [ locale = "<%=Locale%>" ] />
```

このタグは、ボディを使用しません。

属性

- date（必須）: 書式化する日付を java.util.Date インスタンスとして指定します。
- locale: 必要に応じて、ロケールを java.util.Locale インスタンスとして指定します。

例

```
<util:displayDate date="<%=account.getDate()%>"
                  locale="<%=account.getLocale()%>" />
```

displayNumber ユーティリティ・タグ

指定した数値をロケールに適した書式および必要に応じて指定した書式で表示します。ロケールを指定しない場合は、システムのデフォルトのロケールが使用されます。

構文

```
<util:displayNumber number = "<%=Double%>"
                  [ locale = "<%=Locale%>" ]
                  [ format = "<%=Format%>" ] />
```

このタグは、ボディを使用しません。

属性

- number（必須）: 書式化する数値を指定します。
- locale: 必要に応じて、ロケールを java.util.Locale インスタンスとして指定します。
- format: 必要に応じて、書式を java.text.Format インスタンスとして指定します。

例

```
<util:displayNumber number="<%=shoe.getSize()%>" />
```

その他のユーティリティ・タグ

次の各項では、一般的なユーティリティ・タグの構文と属性について説明します。

iterate ユーティリティ・タグ

このタグを使用してコレクション間を反復します。開始タグでコレクションを取得し、ボディでそれを反復します。

構文

```
<util:iterate id = "instance_name"
              type = "class_name"
              collection = "<%=Collection%>"
            [ max = "<%=Integer%>" ] >

... body ...

</util:iterate>
```

ボディは、コレクション内の各要素について一度評価されます。

属性

- id（必須）: イテレータ変数、つまり各反復のインスタンス名です。
- type（必須）: クラス名です。つまり、コレクションはこの型の一連のインスタンスです。
- collection（必須）: コレクション自体です。
- max: 反復する要素の最大数を必要に応じて指定します。

例

```
<util:iterate id="contact" type="com.acme.connections.Contact"
              collection="<%=company.getContacts()%>" >
     <jsp:getProperty name="contact" property="name"/>
</util:iterate>
```

ifInRole ユーティリティ・タグ

指定したアプリケーションのロールにユーザーが存在するかどうかに基づいて、タグ・ボディを評価し、JSP ページのボディにインクルードします。タグ・ハンドラは、request オブジェクトの isUserInRole() メソッドを実行します。

ロールの概念は Servlet 仕様に基づいています。ロールは、アプリケーションの web.xml ファイルの <role> 要素に定義します。

構文

```
<util:ifInRole role = "<%=String%>"
             [ include = "true" | "false" ] >

   ... body to include ...

</util:ifInRole>
```

属性

- role（必須）: ユーザーが、この指定したロールに存在するかどうかをチェックします。
- include: 「true」（デフォルト）に設定すると、ユーザーがロールに存在する場合にのみ、そのボディを含めます。「false」に設定すると、ユーザーがロールに存在しない場合にのみ、そのボディを含めます。

例

```
<util:ifInRole role="users" include="true">
     Logged in as <%=request.getRemoteUser()%><br>
     <form action="logout.jsp">
     <input type="submit" value="Log out"><br>
     </form>
</util:ifInRole>
<util:ifInRole role="users" include="false">
     <form method="POST">
     Username: <input name="j_username" type="text"><br>
     Password: <input name="j_password" type="password"><br>
     <input type="submit" value="Log in">
     </form>
</util:ifInRole>
```

lastModified ユーティリティ・タグ

現行のファイルの最終変更日付を、ロケールに適切な書式で表示します。ロケールを指定しないと、request オブジェクトがロケールのかわりに検索されます。オブジェクトが検出されない場合は、システムのデフォルトのロケールが使用されます。

構文

```
<util:lastModified
             [ locale = "<%=Locale%>" ] />
```

このタグは、ボディを使用しません。

属性

- locale: 必要に応じて、ロケールを java.util.Locale インスタンスとして指定します。

例

```
<util:lastModified />
```

10

パーソナライズ・タグ

この章では、Oracle Application Server Personalization で使用する OC4J に付属のタグ・ライブラリについて説明します。このライブラリを使用するには、OracleAS Personalization 製品が適切にインストールされていることが前提です。

この章には、次の項目が含まれます。

OracleAS Personalization の詳細は、『Oracle Application Server Personalization 管理者ガイド』および『Oracle Application Server Personalization プログラマーズ・ガイド』を参照してください。

パーソナライズの概要

この項では、パーソナライズについて紹介します。まず、一般的な概念、続いて、特に Oracle による実装の概要について説明します。

パーソナライズの一般的な概要

この概要では、パーソナライズの一般的な概念について紹介し、概念が混同されることがある、パーソナライズとカスタマイズの違いについて説明します。

パーソナライズの概念

パーソナライズは、行動、購入、レーティングおよびデモグラフィック・データに基づいて、アプリケーション・ユーザー向けにリコメンデーションを作成する機能です。リコメンデーションは、ユーザーのアプリケーション・セッション中にリアルタイムで作成されます。ユーザーの動作はデータベース・リポジトリ内のプロファイルに保存され、将来のユーザーの動作を予測するためのモデルの構築に使用されます。

将来のユーザー・セッションでは、これらのモデルを使用して、購入する製品またはサービス、あるいはアクセスする Web サイトなど、類似ユーザー（または単一セッション内では、同じユーザー）の動作および希望を予測します。ユーザーは、これらの予測に基づいたリコメンデーションを受け取ります。

OracleAS Personalization タグ・ライブラリは、パーソナライズの 2 つの主要機能を公開します。

- ユーザー・プロファイルに収集される過去のユーザーの動作に基づいて、最も関連する配信コンテンツを選択します。
- このパーソナライズされたコンテンツをアプリケーションの出力または Web ページに柔軟な方法で埋め込みます。

典型的なパーソナライズ・スキーマでは、次のいずれか、またはすべてが考慮されます。

- ユーザーの Web サーフィンのパターン
- 過去におけるユーザーの購入アクティビティ
- 過去におけるユーザーの項目のレーティング
- ユーザーが示す興味の性質および程度（「好む」と「買う」の違い、など）の予測
- 年齢、性別および収入など、ユーザーのデモグラフィック

注意： パーソナライズの概念は、Web サイトおよび Web アプリケーションに制限されません。パーソナライズは、CRM アプリケーションなど、適切なデータがあり、パーソナライズのリコメンデーションが必要とされるアプリケーションで使用できます。ただし、このドキュメントでは、Web アプリケーションに焦点を当てます。

パーソナライズとカスタマイズの比較

Oracle で実装され、この章で説明しているパーソナライズとは、コンテンツが自動的および暗黙的に選択される、一連の複雑で動的な機能です。通常、「パーソナライズ」と呼ばれるが、実際には単に「カスタマイズ」である、より単純で静的な Web サイトの機能とは混同しないでください。

多くのサイトが、たとえば、地域の天気、関心のある株価情報、好きなスポーツといった項目をユーザーに示し、選択した項目に基づいて出力を表示するなどのカスタマイズを提供しています。これは、確かに配信されるコンテンツをパーソナライズしていますが、プロセスは静的で、明示的なユーザーの関与を必要とします。ユーザーが、他の項目を選択することで明示的に変更する機会が与えられるまで、コンテンツの焦点は変わりません。

パーソナライズは、ユーザーの直接的なリクエストなしに、ユーザーのコンテンツを自動的に選択します。コンテンツの選択プロセスは表示されません。さらに、動作を観察することで、システムがユーザーの傾向をより理解するようになると、将来の動作および関心を予測するうえで正確性が増します。

Oracle Application Server Personalization の概要

OracleAS Personalization は、Oracle Database のデータ・マイニング・アルゴリズムを使用して、ユーザーに最も関連するコンテンツを選択します。通常、過去および現在におけるユーザーの動作に関する大量のデータを使用して、OracleAS Personalization リコメンデーション・エンジン（10-4 ページの「リコメンデーション・エンジンの概要」で定義）によって、リコメンデーションが計算されます。このアプローチは、常識的な経験則に依存し、システム内でルールを手動定義する必要がある他のアプローチよりも優れています。

OracleAS Personalization を使用するアプリケーションは、OracleAS Personalization 自体がターゲット・データを提供して、データ収集を制御します。このプロセスにより、アプリケーションが、最低限の有用性しかないデータを大量に収集するのを避けることができます。

OracleAS Personalization タグ・ライブラリにより、幅広い層の JSP 開発者が、この機能を HTML、XML または JavaScript ページで使用できるようになります。タグ・インタフェースのレイヤーは、リコメンデーション・エンジンの低レベルの Java API の最上部に位置します。

リコメンデーションの基礎

OracleAS Personalization 環境のチューニングおよび構成に応じて、リコメンデーションは、たとえば次のような 1 つ以上の要素に基づきます。

- デモグラフィックに基づく類似ユーザーの過去における動作
- ユーザーのプロファイルまたはデモグラフィックを考慮せずに、項目 1、2 および 3 を見たユーザーが、項目 5 および 6 にも関心があるだろうという一般的な傾向が確立されるなど、同じ関心を示した過去のユーザーの動作
- 初めてのユーザー、または匿名のビジターに対してさえもユーザー固有のパーソナライズを可能にし、現在のアクセスの目的を高度にチューニングする、同じユーザーの現在のセッションにおける以前の動作
- 現在のプロモーション、今週の目玉商品などに基づく、ホット・ピックスのリコメンデーション（ユーザー ID を示すものもあれば、示さないものもある）

主要コンポーネント

OracleAS Personalization には、次の主要コンポーネントが含まれています。

- マイニング・テーブル・リポジトリ（MTR）
- マイニング・オブジェクト・リポジトリ（MOR）
- 1 つ以上のリコメンデーション・エンジンで構成されるリコメンデーション・エンジン・ファーム
- リコメンデーション・エンジン Java API

これらについては、次の各項で紹介します。

マイニング・テーブル・リポジトリの概要

OracleAS Personalization マイニング・テーブル・リポジトリ（MTR）には、データ・マイニングに使用されるスキーマおよびデータが含まれています。これは、次のデータを含むデータベース表およびビューのセットです。

- 以前のユーザー動作の記録
- 他の場所で収集され、リポジトリにインポートされたデータ
- ユーザーのデモグラフィック

これらの要素は、ともに、将来のユーザーのプリファレンスを予測するためのモデルの構築に使用されます。

モデルの概要

基本的に、モデルはユーザー・データから演繹された規則の集合です。規則の簡素化された例は、「55 歳以上の女性で、収入が $150,000 から $200,000 の間で、最近スキューバ・ダイビング用のタンクとマスクを購入した人は、フィンとウェット・スーツも購入するであろう」です。

OracleAS Personalization では、モデルは、マイニング・テーブル・リポジトリから収集された記録済の事実に基づいて開発されます。モデル内の規則は、特定の種別のユーザーに典型的であると推測される行動に関する常識的な仮定ではなく、使用可能なデータから厳密に演繹されます。特定のユーザーの特性が、最も使用可能なモデルの規則にどの程度近いかによって、生成されるリコメンデーションが正しいか、または適切であるかどうかが決まります。

マイニング・オブジェクト・リポジトリの概要

OracleAS Personalization マイニング・オブジェクト・リポジトリ（MOR）は、OracleAS Personalization データ・マイニング・スキーマで定義されているとおりに、マイニング・メタ・データおよびマイニング・モデルの結果を保持するデータベース・スキーマです。マイニング・オブジェクト・リポジトリは、データ・マイニング・システムへのログイン、ログオフおよび OracleAS Personalization イベントのスケジュールの焦点として機能します。モデルは、OracleAS Personalization データ・マイニング・アルゴリズムに基づいて、マイニング・テーブル・リポジトリから構築されます。

関連するアルゴリズムをチューニングして、データの異なる特性の比重を変更することにより、同じデータから異なるモデルを構築できます。したがって、ある状況において、マイニング・オブジェクト・リポジトリ内に複数のモデルが存在することがあっても、特定の時間には、1 つのモデルのみがリコメンデーション・エンジンにデプロイされます。

リコメンデーション・エンジンの概要

OracleAS Personalization リコメンデーション・エンジン（RE）は、デプロイ中に OracleAS Personalization モデルをダウンロードし、リコメンデーションのリクエストを処理する際に、適切なユーザー・プロファイル・データをマイニング・テーブル・リポジトリからフェッチする Oracle Database スキーマです。各エンジンは、次のようなアクティビティを実行します。

- モデル・データのロードと保持
- リコメンデーション・リクエストの処理
- ユーザー・プロファイル・データの収集

リコメンデーション・エンジンは、リコメンデーション・リクエストを実行時に処理し、パーソナライズ・リコメンデーションを生成します。また、セッション中にユーザーのプロファイル・データを収集して、Web サイトの現在におけるユーザーの動作を追跡します。この追跡機能により、匿名ユーザーについても登録済のユーザーと同様に、セッション固有のパーソナライズが可能になります。

リコメンデーション・エンジンの移入には、モデルの構築、およびリコメンデーション・エンジン・スキーマへのデプロイが含まれます。これらのステップは、バックグラウンドで行われます。特定のリコメンデーションは、このスキーマ内の PL/SQL ストアド・プロシージャによって計算されます。

リコメンデーション・エンジン・ファームの概要

リコメンデーション・エンジンは、リコメンデーション・エンジン・ファームに含まれる必要があります。ファーム内のすべてのエンジンは、同じモデルでロードされ、同様に使用できます。ファームは 1 つのエンジンのみで構成できますが、ロード・バランシングおよびフェイルオーバーに対応するため、ファーム内に複数のエンジンを含めることをお薦めします。望ましい効果を得るため、これらのエンジンは、異なる物理的システム上の別のデータベース内に置かれます。

リコメンデーション・エンジン API の概念および機能の概要

OracleAS Personalization には、リコメンデーション・エンジンとともに使用する Java API が用意されています。API は、主に、特定のユーザーに対する適切な項目のリコメンデーションのリクエストに使用されます。API は、リコメンデーション・エンジン・データベース・スキーマのストアド・プロシージャへのクライアント・インタフェースとして基本的に機能します。リコメンデーションは、JDBC の接続プーリングを使用して、ストアド・プロシージャへの JDBC コールにより計算されます。

また API は、ユーザー・プロファイル・データを収集するための、データ・コレクション・キャッシュと呼ばれる短期的な記憶域を提供します。これらのデータは、リコメンデーション・エンジン表を介してマイニング・テーブル・リポジトリに周期的にフラッシュされます。ユーザー・データを収集した直後にリコメンデーション・エンジンに書き込まず、この方法でデータをキャッシュすることで、必要な JDBC コールの数を最小限に抑えることができます。ただし、リコメンデーションがリクエストされるたびに、同期 JDBC コールが生成されることに注意してください。リコメンデーション・リクエストの結果は、一意でパーソナライズされているという性質上、キャッシュされません。

JSP プログラマ向けに、リコメンデーション・エンジン API の機能は、OracleAS Personalization タグ・ライブラリの機能にラップされているため、このドキュメントでは、API の詳細は説明していません。このタグ・ライブラリには、便利な自動化機能が含まれていますが、API を直接使用する場合には明示的に管理する必要があります。

次の項では、OracleAS Personalization リコメンデーション・エンジンの概念および機能の概要を説明します。

ビジターとカスタマの比較

リコメンデーション・エンジンには、2 つの種別のユーザーがあります。

- ビジター : 認識されず、デモグラフィック・プロファイル、または過去の動作、プリファレンスおよび操作の格納済履歴を持たない匿名ユーザーです。
- カスタマ : 登録済であるため認識され、また、正確なリコメンデーションの生成に使用されるデモグラフィック・プロファイルおよび動作の格納済履歴を持つユーザーです。

注意 : セッション中に、匿名のビジターを登録済ユーザーに変換できます。10-22 ページの「setVisitorToCustomer パーソナライズ・タグ」を参照してください。

項目、リコメンデーション、分類およびカテゴリ

OracleAS Personalization では、項目は、単一の項または最小の情報単位を示す一般的な概念です。次に例を示します。

- 製品
- 購入するサービス
- ユーザーによって選択される URL
- ユーザーの性別または年齢などのデモグラフィック・データ

項目は、複数の方法で使用されます。

- ユーザー・データの記録のため、項目記録タグに渡されます。この場合、データ項目と呼ばれることがあります。
- 提案として戻されます。この場合、リコメンデーションと呼ばれます。リコメンデーションとして戻される各項目には、レーティングまたはランキングのいずれかである予測値も含まれます。これらのタグのサマリーを、10-7 ページの「レーティングとランキング」に示します。
- アプリケーションによるリコメンデーションのリクエスト時に、入力として渡されます。これは、過去の項目に基づいて項目をリコメンデーションする抱合せ販売の場合、または、特定の一連の項目のレーティングあるいはランキングのために行われます。

インベントリ・システム内の各項目は、すべて、分類に属する必要があります。OracleAS Personalization では、分類は項目の構造的構成です。通常、項目の構成は、ツリーまたはツリー集合のような階層構造で、より広範なグループから先端の各項目へと枝分かれします。分類に含める項目は排他的ではありません。同じ項目を複数の分類に含めることができます。分類は、long 整数である分類 ID で示されます。

アプリケーション・ホストである Web サイトまたはカテゴリは、異なるクライアント・カタログまたは Web サイトに異なる分類 ID を使用することにより、クライアント・データ・セットを区別できます。それぞれのケースで適切な分類が使用されるように、適切な処理を使用してユーザーの種別が識別されます。たとえば、`www.oracle.com` のカスタマが、DBA または Web 開発者であることを示したとします。これによって、このユーザーの将来のアクセスをパーソナライズするのに使用される分類が決定されます。たとえば、プロモーション・キャンペーン、バナーおよび本やトレーニングの販売などのオファーが、Web 生産性ツールの分類またはデータベース管理ツールの分類から取り出されます。

分類内の各項目は、カテゴリにグループ化できます。分類の構造では、カテゴリは、関連する項目のグループで構成される中間ノードです。ただし、ある項目を複数のカテゴリ内に含めることができる点に注意してください。たとえば、『イングリッシュ・ペイシェント』という映画は、「小説の映画化」、「アカデミー賞受賞」、「外国」、「ドラマ」などのカテゴリに属すると考えられます。

レーティング値自体のパラメータも必要であるレーティング項目以外、項目は一般的に、`type` パラメータおよび `ID` パラメータによって一意に識別されます。アプリケーションが、各項目のタイプおよび ID を識別するなんらかのインベントリ・システムに依存できることが前提です。タイプは、「靴」または「スポーツ・イベント」などです。ID は識別番号です。単一の分類内で、2 つの項目に同じ ID を付けることはできません。

パーソナライズ・フィルタ処理設定の一部では、リコメンデーションが、特定の映画のタイトルなどの項目ではなく、「ドラマ」などのカテゴリを示す場合があることに注意してください。この場合は、リコメンデーションの項目タイプは、Category です。10-17 ページの「リコメンデーションのフィルタ処理」も参照してください。

OracleAS Personalization タグ・ライブラリには、JSP ページでの項目およびリコメンデーションの使用を簡素化するのに便利な、`oracle.jsp.webutil.personalization.Item` という public クラスが含まれています。このクラスを使用して、タイプ、ID および予測値にアクセスします。詳細は、10-39 ページの「項目クラスの説明」を参照してください。

レーティングとランキング

リコメンデーションとして戻された項目には、次のような予測値が含まれます。

- レーティング項目の場合は、各項目の予測値がレーティングです。これにより、ユーザーの関心度が予測されます。
- 購入項目またはナビゲーション項目の場合、予測値は、ユーザーの関心度について見積られた確率に基づいて、戻された項目間で関連するランキングを示します。

ナビゲーション項目は、ページの表示、リンクの選択、またはボタンのクリックなど、Web アプリケーションが「ヒット」として考慮する可能性のある動作を示します。

レーティングについて レーティングは、事前定義のスケールでカスタマのプリファレンスを数量で示す方法です。たとえば、映画の場合、ユーザーが好きな映画には 5 つの星、つまり 5.0 のレーティングを付けるという 5 段階方式を採用する場合があります。将来のセッションにおいて、OracleAS Personalization は、このユーザー、および類似した興味や背景を持つユーザーにとって、この映画が高い関心を引くであろうと予測します。大好きというわけではないが、普通程度に好むというユーザーの場合、この映画には 3 つ半の星、または 3.5 のレーティングが付くかもしれません。

ユーザーが、Web サイトで対話的に項目にレーティングを付けると、絶対的なレーティング値が記録されます。できるだけ望ましい粒度が得られるように、レーティングは浮動小数点の数値です。

OracleAS Personalization アルゴリズムによると、リコメンデーション・エンジン API によって（または、JSP ページの場合はリコメンデーション・タグによって）戻されるレーティングが予測値です。

OracleAS Personalization レーティング・システムでは、前例の 0.0 から 5.0 のように、境界は構成可能です。これは、マイニング・テーブル・リポジトリの `MTR.MTR_BIN_BOUNDARY` 表で指定されます。

ランキングについて ランキングは、項目のグループ間で、項目の関連ランクを示す整数です。項目は、購入される（購入する商品の場合）または選択される（アクセスする URL リンクの場合）と予想される確率に基づいてソートされます。確率は、データ・マイニング・モデルおよびカスタマのプロファイル・データを使用して計算されます。

たとえば、項目 A、項目 B および項目 C という 3 つの項目がリコメンデーションとして戻されたと仮定します。ユーザーが関心を持つ確率が、A が 0.9、B が 0.55、C が 0.83 の場合、A には 1、B には 2、C には 3 のランキングが付きます。

項目のランキングは、相対的で動的です。相対的である理由は、ランキングは、相互に比較される項目に対してのみ意味があり、特定の順序でソートされるからです。動的である理由は、同じ項目のランキングが別のカスタマまたは異なる項目に対してランキングされるときには変わる可能性があるからです。

ステートフルとステートレスなリコメンデーション・エンジン・セッションの比較

Web アプリケーションは、ステートフルまたはステートレスのいずれかです。つまり、アプリケーションは、リクエスト間でユーザー・セッションおよびユーザー固有の情報をサーバー上で保持する場合もあれば、しない場合もあります。リコメンデーション・エンジン API およびタグ・ライブラリは、両方の状況を処理するように設計されています。リクエスト間でユーザー情報を保持することには明白な利点がありますが、スループット向上のため、ステートレス・アプリケーションに依存している大規模なサイトもあります。

ただし、リコメンデーション・エンジンは、Web アプリケーションのセッション動作に関係なく、リコメンデーション・エンジン・データベース・スキーマ内におけるユーザー・セッションを常に追跡します。

リコメンデーション・エンジンは、ユーザー ID によってユーザー・セッションを追跡します。したがって、匿名ビジターに一時ユーザー ID を割り当てる場合、注意が必要です。すべての匿名ビジターに同一の ID が使用され、全員の動作が追跡された場合、これらのビジターから収集されたデータは単一のリコメンデーション・エンジン・セッションから取得されたとみなされるため、いずれか 1 人の匿名ビジターの動作がその他のビジターに対するリコメンデーションに影響を与えることになります。この問題は、リコメンデーション・エンジン内で一意の一時 ID を各匿名ビジターに割り当てることで回避できます。

注意： リコメンデーション・エンジン API を直接使用するのに比べ、タグ・ライブラリの 1 つの利点は、ステートレス・アプリケーション内のリコメンデーション・エンジン・セッションの追跡が自動管理される点です。API を直接使用する場合、このマッピングはユーザー自身が設定する必要があります。

ただし、タグ・ライブラリを使用するリコメンデーション・エンジン・セッションの追跡には、Cookie をサポートおよび受け入れるためのクライアント（おそらくブラウザ）が必要である点に注意してください。これが常に保証されない場合、ステートフルなアプリケーションとして宣言する必要があります。

リコメンデーションのリクエスト

リコメンデーション・エンジン・セッションが確立され、データが移入された後は、アプリケーションは、そこからリコメンデーションをリクエストできます。OracleAS Personalization は、適切なリコメンデーションをコール側アプリケーションに戻し、その後、アプリケーションはユーザーに何をどのように渡すか決定します。

JSP ページでは、アプリケーションは、複数のリコメンデーション・タグの 1 つを使用してリコメンデーションをリクエストできます。リコメンデーション・エンジンは、チューニングおよびフィルタ処理設定に準拠し、ユーザー・データに基づいて一連の提案された項目を戻します。OracleAS Personalization タグ・ライブラリを使用する場合、タグ属性または構成ファイルを使用してチューニングおよびフィルタ処理設定を指定できます。

JDBC コールにより、一連のリコメンデーションが、リコメンデーション・エンジン・データベース・スキーマ内で生成されます。コールに要する時間は、場合により異なります。これは、基準、処理が必要なデータ・レコードの数、規則表のサイズおよびユーザー・プロファイル・データのサイズなどの要素、ならびにリコメンデーション・リクエストの仕様によって決まります。リコメンデーションは、パーソナライズ・モデルに基づいて選択され、アプリケーションが接続されているリコメンデーション・エンジンにデプロイされます。OracleAS Personalization タグを使用するとき、`startRESession` タグの属性を使用して、使用するリコメンデーション・エンジンを指定します。

抱合せ販売のリコメンデーションの場合は、アプリケーションは、過去にユーザーが関心を持った 1 つ以上の購入項目またはナビゲーション項目を入力として渡す必要があります。抱合せ販売のリコメンデーションは、渡される 1 つ以上の項目に基づき、また、おそらく過去または現在のユーザー・データにも基づきます。

リコメンデーション項目は配列で戻され、各リコメンデーションの予測値（10-7 ページの「レーティングとランキング」で説明するように、レーティングまたはランキングのいずれか）および配列全体のインタレスト・ディメンションが含まれます。リコメンデーションとして戻される項目について、インタレスト・ディメンションは、項目がユーザーにとってどのような関心の対象となるか（購入項目、ナビゲーション項目またはレーティング項目）を示します。

リコメンデーション・エンジン API により、リコメンデーションは戻される前に分類に基づいてフィルタ処理されます。

パーソナライズ・タグ機能の概要

次の項では、OracleAS Personalization タグ・ライブラリの機能の概要を説明します。各タグの説明および構文は、10-19 ページの「パーソナライズ・タグおよびクラスの説明」を参照してください。

- リコメンデーション・エンジン・セッションの管理
- パーソナライズ・タグでの項目の使用
- 項目記録タグの使用モード
- リコメンデーションおよび評価タグに対するチューニング、フィルタ処理、ソートの使用

注意： OracleAS Personalization タグ・ライブラリは、HTML を唯一の出力形式として想定しません。XML および JavaScript などの他の形式もサポートされます。

リコメンデーション・エンジン・セッションの管理

リコメンデーション・エンジン・セッションの作成操作および閉じる操作は、`startRESession` タグおよび `endRESession` タグにより処理されます。OracleAS Personalization を使用する JSP ページの場合は、少なくとも 1 つの `startRESession` タグが実行され、このタグが特定のリコメンデーション・エンジン・セッションについて検出される最初の OracleAS Personalization タグであるかどうか確認する必要があります。

OracleAS Personalization タグ・ライブラリは、HTTP セッション・オブジェクトによって状態情報を保持するステートフル・アプリケーションと、保持しないステートレス・アプリケーションのいずれかをサポートできます。`startRESession` の `session` 属性を使用して、使用するモードを指定できます。タグ・ライブラリで HTTP セッション・オブジェクトが使用されるようにする場合は、「`true`」設定、HTTP セッションでいずれのタグも使用しない場合は、「`false`」設定を使用します。

`startRESession` タグの `session` 属性を「`true`」に設定すると、`session` 設定を JSP page ディレクティブで「`true`」に設定した場合と同様の結果が得られます。これらの違いは、`startRESession` タグで属性を「`true`」に設定した場合、このタグを含むページ全体のみでなく、同じリコメンデーション・エンジン・セッション内で実行するパーソナライズ・タグを含んだその他のページも影響を受ける点です。

`startRESession` タグが実行された後、パーソナライズ・タグは、Web クライアントとリコメンデーション・エンジン・データベースとの関係を維持するため、必要に応じて、後続のパーソナライズ・タグを同じユーザーに適用できます。

リコメンデーション・エンジン・セッションの開始

`startRESession` タグには、リコメンデーション・エンジン名、およびタグ属性設定や `personalization.xml` 構成ファイル設定を組み合せた情報を指定します。

リコメンデーション・エンジン・セッションが同じ Web クライアントに対して以前に開始されたが、その後 `endRESession` タグが実行されていない場合、`startRESession` タグによって操作が実行されません。これは、JSP ページが実行される順序について柔軟性が得られるため便利です。アプリケーションの複数のページで、`startRESession` タグを配置しても問題はありません。

このタグの詳細は、10-20 ページの「startRESession パーソナライズ・タグ」を参照してください。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。

ステートフル・アプリケーションの使用

HTTP セッションを使用するステートフル・アプリケーションの場合、セッション情報は、JSP の暗黙的な session オブジェクトである、標準の HttpSession インスタンスで保持されます。

startRESession タグが検出されると、session 属性が「true」（デフォルト）に設定されている場合、セッション・オブジェクトが存在しなければ自動的に作成されます。

ステートレス・アプリケーションの使用

ステートレス・アプリケーションの場合、タグ・ライブラリは、Cookie を使用して内部セッションを追跡します。したがって、ステートレス・アプリケーションを使用する場合は、パーソナライズ・タグは、クライアント・ブラウザが Cookie を受け入れるときにのみ機能する点に注意してください。ブラウザが Cookie を拒否するか、または機能不足のために Cookie を受け入れられない場合は、ステートフル機能が必要です（startRESession タグを session="true" に設定）。

リコメンデーション・エンジン・セッションの終了

ステートフル・アプリケーションが、特定のリコメンデーション・エンジン・セッションを必要としなくなったときは、endRESession タグを使用できます。startRESession タグを使用し、endRESession タグを繰返し実行すると、いずれの操作も行われなくなるため、このタグをアプリケーションの複数のページに配置しても問題はありません。

endRESession タグは、ステートレス・アプリケーションには何の影響もありません。

ステートフル・アプリケーションでの endRESession タグの使用はオプションの場合もありますが、次の状況では必要です。

- アプリケーションが、同じブラウザまたは同じ HTTP セッションから別のリコメンデーション・エンジン・ユーザー ID を使用して新規のリコメンデーション・エンジン・セッションを後で開始する予定の場合
- 同じブラウザまたは同じ HTTP セッションから異なるリコメンデーション・エンジンに接続する場合

これらの場合、endRESession タグは、次の startRESession タグの前に実行される必要があります。

また、HTTP セッションが終了するかなり前に OracleAS Personalization タグを使用してアプリケーションを停止し、リコメンデーション・エンジンのリソースを解放する場合にも、endRESession タグの使用をお薦めします。

このタグの詳細は、10-22 ページの「endRESession パーソナライズ・タグ」を参照してください。

注意： endRESession がステートフル・アプリケーションで使用されない場合、HTTP セッションがスコープ外になると、基礎となるリコメンデーション・エンジン・セッションは自動的に終了します。ステートレス・アプリケーションでは、基礎となるリコメンデーション・エンジン・セッションをタイム・アウトさせることができます。

パーソナライズ・タグでの項目の使用

OracleAS Personalization タグ・ライブラリには、項目の操作用に多数のタグが用意されています。ユーザーの動作情報を記録するタグ、以前に記録されたユーザーの動作情報を削除するタグ、リコメンデーションとして項目を出力するタグ、およびレーティングまたはランキングのいずれかに評価される特定の項目セットの入力用タグがあります。

次の項では、タグ・ライブラリの使用方法の概要について説明します。

項目の記録および削除タグの概要

次のタグは、リコメンデーション・エンジン・セッション・キャッシュにデータ項目を記録するか、または、セッション内で以前に記録された項目を削除するためのものです。

- `recordNavigation` および `removeNavigationRecord`
- `recordPurchase` および `removePurchaseRecord`
- `recordRating` および `removeRatingRecord`
- `recordDemographic` および `removeDemographicRecord`

注意： セッション中、記録済の項目は、リコメンデーション・エンジンに周期的にフラッシュされます。フラッシュ後も項目を削除できますが、データベースのラウンドトリップが必要です。`REFlushInterval` の関連情報は、10-20 ページの「startRESession パーソナライズ・タグ」を参照してください。

購入、ナビゲーションまたはレーティング項目を記録あるいは削除するには、タイプと ID（レーティング項目の場合は値）を指定するか、項目配列と索引を配列内に指定することにより、記録あるいは削除する項目を特定する必要があります。詳細は、10-13 ページの「入力項目の仕様」を参照してください。現在のユーザーに暗黙的に適用されるデモグラフィック項目を記録または削除するには、`AGE` などのデモグラフィックのタイプ、および「44」などの値を指定する必要があります。詳細は、10-15 ページの「デモグラフィック項目」を参照してください。

通常、`removeXXXRecord` タグを使用する必要はほとんどありません。`recordXXX` タグを受信ページ内に配置する場合、`removePurchaseRecord` または `removeNavigationRecord` タグを使用する必要はありません。`removeRatingRecord` および `removeDemographicRecord` タグは、最初の入力が記録された後で、ユーザーが考えを変えるような状況にのみ使用する必要があります。関連情報は、10-15 ページの「項目記録タグの使用モード」を参照してください。

タグ情報の詳細は、10-33 ページの「項目記録および削除タグの説明」を参照してください。

リコメンデーションおよび評価タグの概要

次のタグは、リコメンデーションとして項目の配列を戻します。

- selectFromHotPicks
- getRecommendations
- getCrossSellRecommendations
- evaluateItems

このドキュメントでリコメンデーション・タグと呼ばれている selectFromHotPicks、getRecommendations および getCrossSellRecommendations タグの場合、項目の配列は、分類全体から、または分類内のホット・ピックス・グループから戻される一連のリコメンデーションです。getCrossSellRecommendations タグは、リコメンデーション（抱合せ販売ともいう）の基礎となる、一連の購入項目またはナビゲーション項目も入力として取得する必要があります。

ホット・ピックスはプロモーション項目、または特別に選択された他の項目グループです。選択する項目は、タグの属性により指定できます。ホット・ピックスの詳細は、『Oracle Application Server Personalization 管理者ガイド』を参照してください。

evaluateItems の場合、評価する特定の項目セットを入力する必要があります。同じ項目の一部またはすべてが戻され（戻されない場合もある）、インタレスト・ディメンションに基づいてレーティングまたはランキングが決定されます。バックグラウンド情報は、10-7 ページの「レーティングとランキング」を参照してください。

getRecommendations および evaluateItems タグの場合、結果は、特定のユーザーに基づきます。ユーザー ID は、startRESession タグにより指定され、すべての後続のパーソナライズ・タグに暗黙的に適用されます。getCrossSellRecommendations タグは、一連の入力項目によって決まります。

リコメンデーション・タグの詳細 次は、各リコメンデーション・タグの追加情報です。タグの詳細説明は、10-23 ページの「リコメンデーションおよび評価タグの説明」を参照してください。

- selectFromHotPicks: 一連のホット・ピックス・グループの項目が戻されます。hotPicksGroups 属性を使用して、ホット・ピックス・グループを指定します。ユーザーによって結果が変わらないため、これはある意味では OracleAS Personalization タグ・ライブラリ内の「パーソナライズでない」タグです。ただし、パーソナライズ・アプリケーションで、初めてのビジターや、特定の地域または特定の嗜好を持つグループなどにプロモーションを表示するときに役立つ場合があります。
- getRecommendations: ユーザーに基づいて項目が戻されますが、fromHotPicksGroups 属性により指定される一連のホット・ピックス・グループの項目である必要があることを指定することもできます。
- getCrossSellRecommendations: 入力項目に基づいて項目が戻されます。また、戻される項目が fromHotPicksGroups 属性により指定される一連のホット・ピックス・グループの項目である必要があることを指定することもできます。入力項目は、1 人のユーザーが以前に関心を持っていたものが想定されます。このタグの機能は、「過去にユーザーがこの入力項目を購入またはナビゲートしたと仮定すると、将来、このユーザーが最も関心を持つ可能性のある追加項目は何か」という質問に答えることです。この答えは、インタレスト・ディメンションに基づいて、購入またはナビゲートする追加項目になります。

入力項目 入力として項目を取得するタグである、getCrossSellRecommendations および evaluateItems タグの場合、1 つ以上のネストされた forItem タグを使用して希望の項目を指定することや、タグ属性を使用して項目の配列全体を入力することができます。入力項目の詳細は、10-13 ページの「入力項目の仕様」を参照してください。

出力項目 evaluateItems および getCrossSellRecommendations タグの場合、項目の出力配列用のタグ補足情報（TEI）変数の名前を指定するのに必要なタグ属性があります。getRecommendations および selectFromHotPicks タグの場合、この属性はオプションです。また、この項目は、getRecommendations または selectFromHotPicks タグ内にネストされているいずれの getNextItem タグにも順次、使用できます。

リコメンデーション・タグの場合、maxQuantity 属性を使用して、出力項目の最大数を指定できます。戻される項目の実際の数を決定するには、戻される項目に対する TEI 配列変数の length 属性を使用します。配列サイズ用に別の TEI 変数は提供されません。TEI 変数の詳細は、次の項の「戻される項目に対するタグ補足情報スクリプト変数の使用」を参照してください。

戻される項目に対するタグ補足情報スクリプト変数の使用

項目の配列を戻す各タグについて、次の配列タイプのスクリプト変数の使用を可能にするタグ補足情報（TEI）クラスがあります。

```
oracle.jsp.webutil.personalization.Item[]
```

項目の配列は、この変数で戻されます。これらの各タグには、変数名の指定に使用する storeResultsIn 属性が含まれます。アプリケーションで配列をループして、HTML 表などにすべての項目を表示できます。戻される項目数を決定するには、配列の length 属性を使用します。

selectFromHotPicks、getRecommendations および getCrossSellRecommendations タグは、NAVIGATION、PURCHASING または RATING のいずれかである、配列内の項目のインタレスト・ディメンションを示す TEI String 変数も戻します。インタレスト・ディメンションに変数名を指定するには、storeInterestDimensionIn タグ属性を使用します。

> **注意：** タグ補足情報クラスおよびスクリプト変数の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

入力項目の仕様

入力項目を必要とする 2 つの一般的な状況があります。

- 入力項目を getCrossSellRecommendations または evaluateItems タグに指定する場合

  getCrossSellRecommendations または evaluateItems タグに加えて、1 つ以上のネストされた forItem タグを使用します。forItem タグは、希望の入力項目を選択するのに使用します。

- 項目をリコメンデーション・エンジン・セッションに記録する、または同様の構文を使用して、以前に記録された項目を削除する場合

  これには、record*XXX* または remove*XXX*Record タグを使用します。

次の一般的な方法で、項目を指定できます。

1. 希望の項目にそれぞれタイプおよび ID を指定し、レーティング項目の場合は、レーティング値も指定します。デモグラフィック項目の場合は、タイプおよび値を指定します。
2. 希望の各項目について項目配列を指定し、この配列内に索引を指定します。
3. 項目の配列全体を指定します（record*XXX* および remove*XXX*Record タグの場合は関係ありません）。

使用例 2 と 3 の場合の詳細は、次の項の「項目配列の入力」を参照してください。

次のように、1 つ以上の項目を getCrossSellRecommendations または evaluateItems タグに入力できます。

- 各 forItem タグの type および ID 属性を使用して、1 つ以上の forItem タグをタグの内部にネストし、希望の項目を指定します（前述の使用例 1）。

または

- 各 forItem タグの itemList 属性を使用して、1 つ以上の forItem タグをタグの内部にネストして項目配列を指定し、index 属性を使用して、配列の要素を指定します（使用例 2）。

または

- タグの inputItemList 属性により、Item[] 配列を指定します（使用例 3）。配列全体が入力として取得されます。

これらのプロシージャを 1 つ以上同時に使用できます。getCrossSellRecommendations および evaluateItems タグは、複数のソースから入力を取得できます。

次のように、record*XXX* または remove*XXX*Record タグに項目を指定できます。

- タグの type および ID 属性、また、recordRating あるいは removeRatingRecord には value 属性を使用して、項目を指定します（前述の使用例 1）。recordDemographic または removeDemographicRecord の場合は、type および value 属性を使用します。

または

- タグの itemList 属性を使用して、項目配列を指定し、このタグの index 属性を使用して、希望の配列の要素を指定します（使用例 2）。

項目配列の入力

Item[] オブジェクトの配列をタグに入力する場合、JSP 式を使用して配列を指定する必要があります。これは、次のすべてのタグに適用されます。

- inputItemList 属性を使用して、配列全体を入力する場合は、getCrossSellRecommendations または evaluateItems
- itemList および index 属性を使用して、配列を入力し、使用する 1 つの要素を指定する場合は、forItem（getCrossSellRecommendations または evaluateItems 内）、recordPurchase、recordNavigation、recordRating、removePurchaseRecord、removeNavigationRecord あるいは removeRatingRecord

次の方法で配列を指定できます。

- スクリプトレット内で作成し、JSP 式を使用して指定します。

```
<% Item[] myList = newItem[] {newItem("shoes", 1)}; %>
<op:evaluateItems inputItemList="<%=myList %> .../>
```

- リコメンデーション・タグからの出力を含む TEI 変数を使用して指定します。

```
<op:getRecommendations storeResultsIn="myRecs" .../>
<!-- First tag is closed, but TEI variable is still in scope.
Later use it in second tag. -->
<op:getCrossSellRecommendations inputItemList="<%=myRecs %>" />
```

注意： ここで示すタグの構文の詳細は、10-23 ページの「リコメンデーションおよび評価タグの説明」を参照してください。

デモグラフィック項目

性別および年齢など、ユーザーに関するバックグラウンド情報で構成されるデモグラフィック・データ項目は、recordDemographic および removeDemographicRecord タグでのみ使用されます。購入、ナビゲーションまたはレーティング情報を含まないため、リコメンデーション・タグあるいは getCrossSellRecommendations や evaluateItems タグへの入力からは戻されません。

購入項目ならびにナビゲーション項目の場合は type および ID によって識別されますが、デモグラフィック項目は、type ならびに value によって識別されます。これらは、recordDemographic および removeDemographicRecord タグのただ 2 つの属性です。次に示す複数のタイプが事前定義されており、マイニング・テーブル・リポジトリの MTR.MTR_CUSTOMERS 表に列として存在します。

- GENDER
- AGE
- MARITAL_STATUS
- PERSONAL_INCOME
- HOUSEHOLD_INCOME
- IS_HEAD_OF_HOUSEHOLD
- HOUSEHOLD_SIZE
- RENT_OWN_INDICATOR

ATTRIBUTE1 から ATTRIBUTE50 まで、カスタマイズ可能な列が 50 あります。カスタム・タイプを使用するには、次を実行する必要があります。

1. ATTRIBUTEx 列を既存のエンタープライズ・データベースにマップし、属性を定義します。
2. MTR.MTR_BIN_BOUNDARIES 表で、対応する値の境界を定義します。

項目記録タグの使用モード

現在、項目記録タグの操作モードとして受信モードのみ使用できます。このモードでは、購入する項目やナビゲート先の URL などをユーザーが選択すると、受信ページと呼ばれる送信先のページに、この項目を記録する record*XXX* タグが含まれます。

一般的な例として、getRecommendations タグを使用するページで、順に表示するリコメンデーションのリストが生成されると仮定します。各リコメンデーション項目には、ユーザーが詳細情報を取得する場合に選択する「**Details**」リンク、およびユーザーが項目を購入する場合に選択する「**Purchase**」リンクが含まれます。「**Details**」を選択してユーザーがアクセスするページに recordNavigation タグ、「**Purchase**」を選択してユーザーがアクセスするページ（たとえば、購入確認ページ）に recordPurchase タグを配置できます。いずれの場合も、この項目に一意で指定されているタイプおよび ID は、受信ページ側ですでに認識されていると予想できます。

同様に、ユーザーがデモグラフィック情報を入力する JSP ページに recordDemographic タグを配置することもあります。たとえば、ユーザーが配偶者の有無、年齢、個人収入を入力できるページがあるとします。44 歳の独身で年収が $50,000 のユーザーがこの情報を入力すると、HTML フォームのバックグラウンドのターゲット・アクションとして、このプロファイルに合せて宣伝ページが作成されます。このページには、MARITAL_STATUS、AGE および PERSONAL_INCOME タイプの recordDemographic タグが含まれます。単一のページに複数の recordDemographic タグを使用できます。

通常、購入項目およびナビゲーション項目の場合は type と ID など、適切な属性を指定することにより、項目を識別します。または、以前に作成した項目リスト、およびそのリストに索引値を使用して、項目を選択します。アプリケーションは、session または request オブジェクト内に項目リスト配列オブジェクトをコピーし、また、索引もパラメータとして受信ページに渡します。受信ページでは、項目リストが session または request オブジェクトから取得され、索引とともに record*XXX* タグに渡されます。このアプローチでは、送信ページは、受信ページを起動する前に 1 つ以上の索引を収集し、同じ項目リストから多数の項目を同時に記録できるという、少なくとも 1 つの利点があります。

リコメンデーションおよび評価タグに対するチューニング、フィルタ処理、ソートの使用

前に要約したように、selectFromHotPicks、getRecommendations、getCrossSellRecommendations および evaluateItems タグは、すべて項目の配列を戻します。次の項では、戻されるリコメンデーションを詳細に作成するのに使用するチューニングおよびフィルタ処理設定について説明します。また、リコメンデーションのソート設定に関する情報もあります。ただし、項目の出力は常に一連の項目の入力からなので、フィルタ処理設定は、evaluateItems タグには適用されません。

チューニング設定

複数のチューニング設定によって、戻されるリコメンデーション内でリコメンデーション・エンジンによって使用されるいくつかの条件および論理が決定されます。各設定には値が必要であり、ここで説明する方法の 1 つで決定されます。

これらの設定は、表 10-1 のサマリーで示すように、selectFromHotPicks、getRecommendations、getCrossSellRecommendations および evaluateItems タグの tuning*XXX* 属性により指定できます。または、tuningName 属性を使用して、アプリケーション・レベルの personalization.xml ファイル（優先）か、サーバー全体の personalization.xml ファイルのいずれかの中の、指定した <Tuning> 要素から設定を取得できます。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。

属性設定がなく、<Tuning> 要素もない場合、次のステップの順にデフォルト値が選択されます。

1. アプリケーション・レベルの personalization.xml ファイル内の <DefaultTuning> 要素に準拠します。
2. サーバー全体の personalization.xml ファイル内の <DefaultTuning> 要素に準拠します。
3. 次のハードコードされた設定に準拠します。

```
tuningDataSource="ALL"
tuningInterestDimension="NAVIGATION"
tuningPersonalizationIndex="MEDIUM"
tuningProfileDataBalance="BALANCED"
tuningProfileUsage="INCLUDE"
```

注意： ハードコードされているデフォルトを使用するには、tuning*XXX* 属性設定のいずれも使用しないでください。一部のチューニング設定がタグ内で定義されている場合、ハードコードされた値はいずれも使用されません。この場合、タグ属性内または personalization.xml ファイル内にいずれの設定も見つからなければ例外が発生します。

表 10-1 リコメンデーションのリクエスト設定のチューニング

属性	説明	設定
tuningDataSource	リコメンデーションの作成で考慮される過去のユーザー・データを指定します。（この種類のデータ・ソースを J2EE プラットフォーム・モデルのデータ・ソース概念と混同しないでください。）	ALL、NAVIGATION、PURCHASE、RATING、DEMOGRAPHIC
tuningInterestDimension	戻されるリコメンデーションの種類を指定します。	RATING、PURCHASING、NAVIGATION、
tuningPersonalizationIndex	リコメンデーション・アルゴリズムの一般性またはパーソナライズの程度を選択します。	LOW、MEDIUM、HIGH
tuningProfileDataBalance	リコメンデーションを作成する際に、履歴データ、現在のセッション・データ、または両方のどれに重点を置くかを選択します。	HISTORY、CURRENT、BALANCED
tuningProfileUsage	ユーザーのデモグラフィック・プロファイルのデータを使用するかどうかを選択します。	INCLUDE、EXCLUDE

詳細は、『Oracle Application Server Personalization 管理者ガイド』を参照してください。

リコメンデーションのフィルタ処理

チューニング設定に加え、リコメンデーション・リクエストに指定可能なフィルタ処理設定があります。各設定には値が必要であり、ここで説明する方法の 1 つで決定されます。

これらの設定は、getRecommendations、getCrossSellRecommendations および selectFromHotPicks タグの filtering*XXX* 属性により指定できます。（フィルタ処理は、evaluateItems タグには関係ありません。）または、filteringName 属性を使用して、アプリケーション・レベルの personalization.xml ファイル（優先）か、サーバー全体の personalization.xml ファイルのいずれかの中の、指定した <Filtering> 要素から設定を取得できます。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。

属性設定も <Filtering> 要素も存在しない場合、デフォルト値はアプリケーション・レベルの personalization.xml ファイル（優先）か、サーバー全体の personalization.xml ファイルのいずれかの中の、<DefaultFiltering> 要素から選択されます。

これらは、フィルタ処理用パラメータです。

- filteringTaxonomyID: OracleAS Personalization 環境において、整数が項目分類の ID である場合、これは、整数を示す Java 文字列です。
- filteringMethod: これは、ALL_ITEMS、INCLUDE_ITEMS、EXCLUDE_ITEMS、SUBTREE_ITEMS、ALL_CATEGORIES、INCLUDE_CATEGORIES、EXCLUDE_CATEGORIES、SUBTREE_CATEGORIES および CATEGORY_LEVEL のいずれかです。表 10-2 に意味のサマリーを示します。常に、これらのメソッドは、filteringTaxonomyID 値により指定される分類に適用されます。

  getCrossSellRecommendations タグの場合は、ALL_ITEMS、INCLUDE_ITEMS、EXCLUDE_ITEMS および SUBTREE_ITEMS 設定のみがサポートされています。
- filteringCategories: これは、各 ID の後ろを単一のプラス記号（+）で区切った整数 ID の Java 文字列で、特定の分類内にある既存の項目カテゴリを識別します。カテゴリは、マイニング・テーブル・リポジトリの MTR.MTR_CATEGORY 表で指定されます。

注意： filteringMethod が ALL_ITEMS または ALL_CATEGORIES のときは、filteringCategories 設定を指定しないでください。

表 10-2 リコメンデーションのリクエスト用のフィルタ処理メソッド

フィルタ処理メソッド	説明
ALL_ITEMS	分類内のすべての項目をリコメンデーションします。
INCLUDE_ITEMS	filteringCategories で指定されるカテゴリに属する項目をリコメンデーションします。
EXCLUDE_ITEMS	filteringCategories で指定されるカテゴリに属さない分類の項目をリコメンデーションします。
SUBTREE_ITEMS	filteringCategories で指定されるカテゴリのサブツリーに属する項目をリコメンデーションします。
ALL_CATEGORIES	分類内のすべてのカテゴリをリコメンデーションします。
INCLUDE_CATEGORIES	filteringCategories で指定されるカテゴリをリコメンデーションします。
EXCLUDE_CATEGORIES	filteringCategories で指定されない分類のカテゴリをリコメンデーションします。
SUBTREE_CATEGORIES	filteringCategories で指定されるカテゴリのサブツリーからのカテゴリをリコメンデーションします。
CATEGORY_LEVEL	filteringCategories で指定されるカテゴリと同じレベルのカテゴリをリコメンデーションします。

すべての *XXX*_CATEGORIES 設定について、リコメンデーションは、映画のタイトルなどの特定の項目ではなく「ドラマ」など、カテゴリの形式で戻されます。この場合、項目タイプは Category で、先に、マイニング・テーブル・リポジトリでカテゴリが定義されている必要があります。

詳細は、『Oracle Application Server Personalization プログラマーズ・ガイド』を参照してください。

ソート順序

レーティングまたはランキングのいずれかの各項目の prediction フィールドに従って、戻された項目をソートできます。このフィールドを使用する方法の詳細は、10-7 ページの「レーティングとランキング」を参照してください。

selectFromHotPicks、getRecommendations、getCrossSellRecommendations または evaluateItems タグの sortOrder 属性を使用して、ASCEND、DESCEND あるいは NONE（デフォルト）のソート順序を指定します。昇順は、最も一致するものを最初に表示し、降順はその逆です。5 段階にランキングされた項目の昇順は、1 が最高なので、1、2、3、4、5 の順になります。また、5 段階にレーティングされた項目の昇順は、最も大きな数字が最高のレーティングを意味するので、4.5、3.9、2.5、2.2、1.8 の順になります。

パーソナライズ・タグおよびクラスの説明

次の項では、OracleAS Personalization タグおよび Item public クラスの構文の詳細ならびに使用例を示した後、最後にタグの制限事項について説明します。

OracleAS Personalization タグ・ライブラリを使用する場合は、次の要件に注意してください。

- ojsputil.jar ファイルがインストール済で、クラスパスに存在していることを確認してください。このファイルは、OC4J に同梱されていて、予約済のタグ・ライブラリ・ディレクトリにあります。
- oreapi-rt.jar ファイルにあるリコメンデーション・エンジン API のクラスが必要となります。Business Intelligence オプションを指定して Oracle Application Server をインストールした場合、このファイルは、[SRCHOME]/dmt/jlib ディレクトリ内にインストールされます。このファイルを、アプリケーションによるアクセスが可能な場所にコピーします。
- タグ・ライブラリ・ディスクリプタ・ファイル personalization.tld が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な taglib ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は ojsputil.jar に配置されます。personalization.tld の uri 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/personalization.tld
  ```

taglib ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび uri 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意：

- このタグ構文では、接頭辞「op:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

ここで説明しているタグ属性の中には、使用する場合に OracleAS Personalization およびリコメンデーション・エンジンの実装に関する一般的な知識を必要とするものがあります。詳細は、『Oracle Application Server Personalization 管理者ガイド』または『Oracle Application Server Personalization プログラマーズ・ガイド』を参照してください。

セッション管理タグの説明

次の項では、リコメンデーション・エンジン・セッションの開始、終了および管理用のタグについて説明します。

startRESession パーソナライズ・タグ

この項では、リコメンデーション・エンジン・セッションの開始に使用する startRESession タグの構文および属性について説明します。関連情報は、10-9 ページの「リコメンデーション・エンジン・セッションの管理」を参照してください。

startRESession タグは、同じリコメンデーション・エンジン・セッション内で実行するその他の OracleAS Personalization タグの前に実行される必要があります。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:startRESession REName = "recommendation_engine_connection_name"
                  [ REURL = "rec_engine_database_connection_URL" ]
                  [ RESchema = "rec_engine_schema_name" ]
                  [ REPassword = "rec_engine_schema_password" ]
                  [ RECacheSize = "kilobytes_of_cache" ]
                  [ REFlushInterval = "milliseconds_to_flush" ]
                  [ session = "true" | "false" ]
                  [ userType = "visitor" | "customer" ]
                  [ UserID = "user_ID_for_site_login" ]
                  [ storeUserIDIn = "variable_name" ]
                  [ disableRecording = "true" | "false" ] />
```

startRESession タグには、ボディはありません。

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- REName（必須）：リコメンデーション・エンジン・ファーム内のリコメンデーション・エンジンの接続名を指定するのに使用します。一部の状況においては、「属性の使用方法」で説明しているように、設定を取得するために personalization.xml 内の <RE> 要素の名前も一致させる必要があります。関連情報は、10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」を参照してください。
- REURL: リコメンデーション・エンジン・データベースの JDBC 接続文字列です。
- RESchema: リコメンデーション・エンジン・データベース・スキーマの名前です。
- REPassword: RESchema の名前に対応するパスワードです。

  OC4J の jazn-data.xml ファイルに、リコメンデーション・エンジン・データベース内の、たとえば scott というアカウント用の <user> 要素が含まれている場合は、次のように特殊な右矢印構文でダッシュ（-）と右山カッコ（>）に続けてこのアカウント名を指定することにより、そのファイルから不明瞭化されていないパスワードを取得できます。

  ```
  password="->scott"
  ```

- RECacheSize: リコメンデーション・セッション・キャッシュのサイズを KB で指定するのに使用します。デフォルトは 3234KB です。これは、「属性の使用方法」で説明しているように、REFlushInterval に合せて調整する必要があります。
- REFlushInterval: リコメンデーション・エンジン・セッション・キャッシュ内のデータがリコメンデーション・エンジン・スキーマにフラッシュされる頻度を指定するのに使用します。単位はミリ秒で、デフォルトは 60000（1 分）です。これは、「属性の使用方法」で説明しているように、RECacheSize に合せて調整する必要があります。
- session: 「true」設定（デフォルト）を指定して HTTP セッション・オブジェクトを使用し、OracleAS Personalization JSP ページがステートフル形式で動作するように指定します。かわりに Cookie を使用し、ページがステートレス形式で動作するように指定する場合は、「false」設定を使用します。
- userType: Web サイト・ユーザーが匿名のビジター（デフォルト）または登録済のカスタマのいずれであるかを示します。

- userID: Web サイト・ユーザーのユーザー名です。匿名のビジターなど、名前がない場合、タグ・ハンドラにより ID が自動的に生成されます。
- storeUserIDIn: 後で使用するために、userID 値を格納する場合、storeUserIDIn によって、格納先の TEI String 変数の名前を指定できます。この属性は、自動生成されるユーザー ID の場合に便利です。
- disableRecording: 「true」設定を使用して、record*XXX* タグの任意のアクションを無効にします。これにより、たとえば、Web サイトで、ユーザーが自身のアクティビティを記録しないように指定するなどの操作が可能になります。また、ピーク時間中のサイトのパフォーマンスを向上させます。たとえば、現在のユーザー ID を基にして、この属性をリクエスト時に設定できます。これによって、JSP コードを変更することなく、適切なユーザーのみ、または適切な時間に限って記録を無効に設定できます。デフォルト設定は「false」です。

属性の使用方法

- startRESession タグが機能するには、REName 属性が必要で、さらに、タグ属性または personalization.xml ファイルの 1 つを使用して、REURL、RESchema および REPassword を定義する必要があります。(10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。)
- REName は、リコメンデーション・エンジン・ファーム内のリコメンデーション・エンジンの接続名を指定します。効率を上げるため、複数のユーザー・セッションで同じ接続をできるだけ共有します。これには、同じ接続を使用するときは常に同じ REName を使用します。リコメンデーション・エンジンの接続は、確立されると、REName 値をキーとして使用してキャッシュされます。

  REURL、RESchema または REPassword は、最初に接続を確立する startRESession タグの属性によって設定されず、これら 3 つの設定は、すべて、Name 属性が startRESession タグの REName 値と一致する <RE> 要素を含んだ personalization.xml ファイルから取得される必要があります。このとき、デフォルト値以外の値を使用する場合、RECacheSize および REFlushInterval も <RE> 要素で設定する必要があります。この例では、アプリケーション全体の personalization.xml が最初に検索され、サーバー全体の personalization.xml は、アプリケーション全体のファイルに REName 値を名前として持つ <RE> 要素が含まれない場合にのみ検索されます。

  注意： REName が既存の接続名と一致するとき、REURL、RESchema、REPassword、RECacheSize および REFlushInterval は、余分なため無視されます。

- REName 属性を <RE> 要素とともに使用して、ファーム内のリコメンデーション・エンジン間のロード・バランシングを簡易化できます。各 <RE> 要素は、ファーム内の異なるリコメンデーション・エンジンを指します。JSP ページは、複数のロード・バランシング機能に基づいて、異なる startRESession タグの REName 属性に別の値を割り当てることで、ファーム内の異なるリコメンデーション・エンジン間を移動できます。
- デフォルト値が RECacheSize および REFlushInterval に指定されていますが、これらは、作業開始のためだけに用意されたものです。アプリケーションの実行に慣れたら、Web サイトの状況に応じて、これらの値をチューニングできます。RECacheSize および REFlushInterval の設定は、相互に関連しており、また、ユーザー操作の結果としてリコメンデーション・エンジン・セッション・キャッシュに項目が追加される速度の見積りによって異なります。デフォルトのキャッシュ・サイズは最大 3234KB で、これは約 4800 項目の格納に十分なサイズです。デフォルトのフラッシュ間隔は 60 秒（60000 ミリ秒）で、毎秒 80 項目のキャッシュ受信率が得られます。フラッシュ間隔を 120 秒に延長すると、新しい項目を毎秒 40 個しか追加できません。これに対し、フラッシュ間隔を 30 秒に短縮すると、キャッシュ受信率は毎秒 160 項目になります。ただし、フラッシュ間隔を短縮する不利な点は、フラッシュされた後に項目を削除すると（remove*XXX*Record タグを使用して）、データベースのラウンドトリップが必要となることです。

  同じ JVM 内の同じリコメンデーション・エンジンの接続を共有するすべてのセッションは、同じセッション・キャッシュも共有する点に注意してください。キャッシュの受信率は、このようなすべてのセッションの累積です。

例 次の例は、リコメンデーション・エンジン・データベースの URL およびユーザー名の scott を確立し、OC4J jazn-data.xml ファイルから scott アカウントの不明瞭化されていないパスワードを取得する startRESession タグを示します。これは、jazn-data.xml に scott のエントリがあることを前提とします。

```
<op:startRESession REName = "RE1"
                   REURL = "@jdbc:oracle:thin:@sid"
                   RESchema = "scott"
                   REPassword = "->scott" />
```

endRESession パーソナライズ・タグ

このタグを使用して、ステートフル・アプリケーションのリコメンデーション・エンジン・セッションを明示的に終了します。通常、これはオプションですが、一部の状況では必要とされます。（詳細は、10-10 ページの「リコメンデーション・エンジン・セッションの終了」を参照してください。）アプリケーション・ロジックで、リコメンデーション・エンジン・セッションを今後必要としないと判断される場合も、ステートフル・アプリケーションでこのタグを使用することをお薦めします。これにより、不必要なリソースが解放されます。

endRESession を使用しない状況では、次の動作に注意してください。

- startRESession タグの session 属性を「true」に設定してリコメンデーション・エンジン・セッションを開始した場合、このリコメンデーション・エンジン・セッションは、HTTP セッションの終了時に暗黙的に終了します。
- session を「false」に設定してリコメンデーション・エンジン・セッションを開始した場合は、リコメンデーション・エンジン・セッションは、十分な時間、非アクティブとなった後、タイムアウトされます。タイムアウトの間隔は、リコメンデーション・エンジン・スキーマの構成パラメータとして指定されます。endRESession タグの影響はありません。

構文

```
<op:endRESession />
```

endRESession タグには、属性もボディもありません。

setVisitorToCustomer パーソナライズ・タグ

匿名のビジターが、登録済のカスタマ・アカウントを作成する場合に、このタグを使用します。このタグを実行するとき、既存のリコメンデーション・エンジン・セッションがビジターのセッションからカスタマのセッションに変換されます。このセッションで以前に収集されたデータは保存されます。このタグは、実際には新規のカスタマを作成せず、また、新規ログインを実行しません。継続中のリコメンデーション・エンジン・セッションを変換するのみです。

customerID は、リクエスト時の属性で、アプリケーションによって提供される必要があります。

構文

```
<op:setVisitorToCustomer customerID = "<%=registered_customer_name%>" />
```

setVisitorToCustomer タグには、ボディはありません。

属性

- customerID（必須）：アプリケーションによって、新規登録されたカスタマに ID が提供されます。

リコメンデーションおよび評価タグの説明

次の項で、リコメンデーション・タグ、評価タグおよび関連サブタグの詳細を説明します。

次のタグが対象です。

- getRecommendations パーソナライズ・タグ
- getCrossSellRecommendations パーソナライズ・タグ
- selectFromHotPicks パーソナライズ・タグ
- evaluateItems パーソナライズ・タグ
- forItem パーソナライズ・タグ
- getNextItem パーソナライズ・タグ

10-12 ページの「リコメンデーションおよび評価タグの概要」も参照してください。

getRecommendations パーソナライズ・タグ

このタグを使用して、購入、ナビゲーションまたはレーティングに関する一連のリコメンデーションをリクエストします。特定の分類からの項目が、指定したチューニングおよびフィルタ処理を使用して考慮されます。リコメンデーションは、次のタイプの配列で戻されます。

```
oracle.jsp.webutil.personalization.Item[]
```

getCrossSellRecommendations および evaluateItems などの他のタグでは、リコメンデーションの基礎として使用するために項目を入力する必要があるのに対し、getRecommendations タグではその必要はありません。リコメンデーションは、特定の項目ではなく、ユーザーの ID およびプロファイル（ユーザー・セッションおよび履歴データ）に基づきます。

生成されるリコメンデーションは、オプションで、タグの storeResultsIn 属性で変数名を指定して、Item[] タイプの TEI 変数内に格納できます。リコメンデーションは、getRecommendations タグ内でも暗黙的に取得可能です。オプションで、希望する項目の処理に getNextItem タグがネストされたタグ・ボディを使用できます。10-31 ページの「getNextItem パーソナライズ・タグ」を参照してください。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:getRecommendations
  [ from = "top" | "bottom" ]
  [ fromHotPicksGroups = "string_of_Hot_Picks_group_numbers" ]
  [ storeResultsIn = "TEI_variable_name" ]
  [ storeInterestDimensionIn = "TEI_variable_name" ]
  [ maxQuantity = "integer_value" ]
  [ tuningName = "name_from_config_file_Tuning_element" ]
  [ tuningDataSource = "ALL"|"NAVIGATION"|"PURCHASE"|"RATING"|"DEMOGRAPHIC" ]
  [ tuningInterestDimension = "NAVIGATION"|"PURCHASING"|"RATING" ]
  [ tuningPersonalizationIndex = "LOW"|"MEDIUM"|"HIGH" ]
  [ tuningProfileDataBalance = "HISTORY"|"CURRENT"|"BALANCED" ]
  [ tuningProfileUsage = "INCLUDE"|"EXCLUDE" ]
  [ filteringName = "name_from_config_file_Filtering_element" ]
  [ filteringTaxonomyID = "integer_value" ]
  [ filteringMethod = "ALL_ITEMS"|"EXCLUDE_ITEMS"|"INCLUDE_ITEMS"|
            "SUBTREE_ITEMS"|"ALL_CATEGORIES"|"INCLUDE_CATEGORIES"|
            "EXCLUDE_CATEGORIES"|"SUBTREE_CATEGORIES"|CATEGORY_LEVEL" ]
  [ filteringCategories = "string_of_integers" ]
  [ sortOrder = "ASCEND"|"DESCEND"|"NONE" ] >

...

</op:getRecommendations>
```

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- from: 項目をその分類全体から選択する場合に使用します。デフォルトで通常の設定である top 設定は、最も希望に合う項目を N 個以下（N は表示するリコメンデーションの最大数）表示します（maxQuantity）。bottom 設定は、最も希望に合わない項目を N 個以下表示します。これは、たとえば、製品管理部門が最もカスタマの人気が少ない項目を把握する場合などに便利です。
- fromHotPicksGroups: 項目を 1 つ以上のホット・ピックス・グループから選択する場合に使用します。アプリケーションは、startRESession タグで指定されたものと同じリコメンデーション・エンジンから一連のホット・ピックス・グループの ID 番号を決定する必要があります。fromHotPicksGroups 属性で、「10+20+30」というように、グループの ID 番号をプラス記号（+）で区切って 1 つの文字列にリストしてください。
- storeResultsIn: オプションで、生成されるリコメンデーションが格納される Item[] タイプの TEI 変数の名前を指定します。（これは、getCrossSellRecommendations の属性には必要ですが、getRecommendations には必要ありません。）変数名を指定すると、この変数のスコープは AT_BEGIN となり、開始タグからページの終わりまで使用できるようになります。値は、JSP 式ではなく、変数名であることに注意してください。変換には変数名を指定する必要があり、これは、リクエスト時属性ではありません。
- storeInterestDimensionIn: オプションで、NAVIGATION、PURCHASING または RATING のいずれかのインタレスト・ディメンションを格納するための TEI 文字列変数の名前を指定します。比較には、Item クラス定義済定数である INT_DIM_NAVIGATION、INT_DIM_PURCHASING または INT_DIM_RATING を使用します。変数名を指定すると、この変数のスコープは AT_BEGIN となり、開始タグからページの終わりまで使用できるようになります。変換には変数名を指定する必要があり、これは、リクエスト時属性ではありません。戻される値は、タグで使用される tuningInterestDimension 設定と同じになります。
- maxQuantity: 戻されるリコメンデーションの最大数を指定するのに使用します。アプリケーションの personalization.xml ファイル、またはサーバー全体の personalizaton.xml ファイルの <RecommendationSettings> 要素で一般的なデフォルト設定が指定されている場合、これはオプションです。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。
- tuningName: チューニング設定が取得されるように、personalization.xml の <Tuning> 要素の名前を指定するのに使用します。または、個別の tuning*XXX* 属性を使用します。
- tuningDataSource: 10-16 ページの「チューニング設定」を参照してください。
- tuningInterestDimension: 10-16 ページの「チューニング設定」を参照してください。
- tuningPersonalizationIndex: 10-16 ページの「チューニング設定」を参照してください。
- tuningProfileDataBalance: 10-16 ページの「チューニング設定」を参照してください。
- tuningProfileUsage: 10-16 ページの「チューニング設定」を参照してください。
- filteringName: フィルタ処理設定が取得されるように、personalization.xml の <Filtering> 要素の名前を指定するのに使用します。または、個別の filtering*XXX* 属性を使用します。
- filteringTaxonomyID: 10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。
- filteringMethod: 10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。
- filteringCategories: 10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。文字列内の整数は、「101+200+35」というように、プラス記号（+）で区切ります。

- sortOrder: 項目のソートを昇順（ASCEND、最も一致するものを一番上に表示）または降順（DESCEND）に指定するのに使用します。デフォルトは、いずれでもなく（NONE）、ソート要件はありません。詳細は、10-18 ページの「ソート順序」を参照してください。

属性の使用方法

- from または fromHotPicksGroups のいずれかを指定する必要があります。
- storeResultsIn 属性、または getNextItem タグがネストされたタグ・ボディのいずれか、あるいは両方（オプション）を使用して出力項目にアクセスします。
- personalization.xml 内の <Tuning> 要素の名前に対応する tuningName を指定するか、または、tuning*XXX* 属性を使用して個別のチューニング設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-16 ページの「チューニング設定」を参照してください。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。
- personalization.xml 内の <Filtering> 要素の名前に対応する filteringName を指定するか、または、filtering*XXX* 属性を使用して個別のフィルタ処理設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。
- filteringMethods が、ALL_ITEMS または ALL_CATEGORIES に設定されていないかぎり、filteringCategories 設定が必要です。これらの設定は、タグ属性または personalization.xml を使用して実行できます。
- マイニング・テーブル・リポジトリで定義されているように、特定の項目ではなく、*XXX*_CATEGORIES フィルタ処理メソッドによりカテゴリが戻されます。

例 次は、getRecommendations タグの基本的な使用例です。storeResultsIn 属性は、受信および表示する結果の Item[] 配列を定義します。

```
<op:getRecommendations storeResultsIn="myRecs">
<% for(int i = 0; i< myRecs.length; i++) {
   Render(myRecs(i).getType(),myRecs(i).getID());
} %>
</op:getRecommendations>
```

ネストされた getNextItem タグを使用する getRecommendations タグの例について、10-31 ページの「getNextItem パーソナライズ・タグ」も参照してください。

getCrossSellRecommendations パーソナライズ・タグ

getRecommendations タグと同様、getCrossSellRecommendations タグは、購入、ナビゲーションまたはレーティングに対する一連のリコメンデーションを Item[] タイプの配列で戻します。特定の分類からの項目が、指定したチューニングおよびフィルタ処理を使用して考慮されます。

ただし、getCrossSellRecommendations を使用するには、生成されるリコメンデーションの基礎として使用される、ユーザーが過去に関心を示した一連の購入項目またはナビゲーション項目を入力する必要があります。項目は、すべて同じ分類に属する必要があります。

項目は、指定した項目配列、または forItem タグがネストされたボディ・タグを使用して入力できます。詳細は、10-13 ページの「入力項目の仕様」を参照してください。10-30 ページの「forItem パーソナライズ・タグ」も参照してください。

getCrossSellRecommendations タグからのリコメンデーションは、タグの storeResultsIn 属性で指定される変数名を持つ、Item[] タイプの TEI 変数内に格納されます。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:getCrossSellRecommendations
    storeResultsIn = "TEI_variable_name"
  [ storeInterestDimensionIn = "TEI_variable_name" ]
  [ fromHotPicksGroups = "string_of_Hot_Picks_group_numbers" ]
  [ inputItemList = "item_array_expression" ]
  [ maxQuantity = "integer_value" ]
  [ tuningName = "name_from_config_file_Tuning_element" ]
  [ tuningDataSource = "ALL"|"NAVIGATION"|"PURCHASE"|"RATING"|"DEMOGRAPHIC" ]
  [ tuningInterestDimension = "NAVIGATION"|"PURCHASING"|"RATING" ]
  [ tuningPersonalizationIndex = "LOW"|"MEDIUM"|"HIGH" ]
  [ tuningProfileDataBalance = "HISTORY"|"CURRENT"|"BALANCED" ]
  [ tuningProfileUsage = "INCLUDE"|"EXCLUDE" ]
  [ filteringName = "name_from_config_file_Filtering_element" ]
  [ filteringTaxonomyID = "integer_value" ]
  [ filteringMethod = "ALL_ITEMS"|"EXCLUDE_ITEMS"|"INCLUDE_ITEMS"|
                      "SUBTREE_ITEMS" ]
  [ filteringCategories = "string_of_integers" ]
  [ sortOrder = "ASCEND"|"DESCEND"|"NONE" ] >

...

</op:getCrossSellRecommendations>
```

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- `inputItemList`: `Item[]` によって入力項目を指定する場合は、配列を戻す JSP 式とともにこの属性を使用します。式内の項目配列は、前のリコメンデーション・タグから取得されます。詳細は、10-14 ページの「項目配列の入力」を参照してください。

10-23 ページの「getRecommendations パーソナライズ・タグ」で説明しているように、「属性の使用方法」で説明されている制限事項以外、また `getCrossSellRecommendations` タグには `storeResultsIn` 属性が必要な点以外は、`getRecommendations` タグには `getCrossSellRecommendations` タグのその他すべての属性が使用されます。

チューニング、フィルタ処理およびソートの詳細は、10-16 ページの「チューニング設定」、10-17 ページの「リコメンデーションのフィルタ処理」および 10-18 ページの「ソート順序」を参照してください。

属性の使用方法

- 入力項目には、`inputItemList` 属性、または `forItem` タグがネストされたボディ、あるいはその両方（オプション）が必要です。両方の機能を使用する場合は、最初に `forItem` タグが実行され、示された項目が項目リスト内に配置されます。次に、`inputItemList` エントリが考慮され、リストに追加されます。
- `getRecommendations` タグとは異なり、`storeResultsIn` は、`getCrossSellRecommendations` タグの必須属性です。生成されるリコメンデーションを格納するため、`Item[]` タイプの TEI 変数の名前を指定する必要があります。
- `personalization.xml` 内の `<Tuning>` 要素の名前に対応する `tuningName` を指定するか、または、`tuning`*XXX* 属性を使用して個別のチューニング設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-16 ページの「チューニング設定」を参照してください。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。
- `tuningInterestDimension` 設定が `tuningDataSource` 設定と異なる場合、OracleAS Personalization の規則の設定によって、いずれのリコメンデーションも取得されない場合があります。
- `personalization.xml` 内の `<Filtering>` 要素の名前に対応する `filteringName` を指定するか、または、`filtering`*XXX* 属性を使用して個別のフィルタ処理設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。

- filteringMethods が、ALL_ITEMS に設定されていないかぎり、filteringCategories 設定が必要です。これらの設定は、タグ属性または personalization.xml を使用して実行できます。
- getCrossSellRecommendations タグでは、カテゴリ・ベースのフィルタ処理は使用できないため、ALL_ITEMS、INCLUDE_ITEMS、EXCLUDE_ITEMS および SUBTREE_ITEMS という、一部のフィルタ処理メソッドのみがサポートされています。

例 次の例では、getCrossSellRecommendations タグを使用して、過去において特定の DVD をレンタルまたは購入したユーザーに、他の DVD を提案します。

```
<% long[] ids = ApplicationPackage.getUserHistory("Smith01");
   Item[] DVDs = new Item[ids.length];
   for(int i=0; i<ids.length; i++) {
      DVDs[i] = new Item("DVD", ids[i] );
   }
   pageContext.setAttribute("pastInterest", DVDs);
%>
<op: getCrossSellRecommendations inputItemList="pastInterest"
                                 storeResultsIn="moreDVDs"
                                 maxQuantity = "4"
                                 sortOrder="ASCEND" />
<!-- display 4 best cross-sell items -->
<h1> You will also enjoy these titles! </h1>

ApplicationSupport.displayItem(moreDVDs[1].getType(), moreDVDs[1].getID() );
ApplicationSupport.displayItem(moreDVDs[2].getType(), moreDVDs[2].getID() );
ApplicationSupport.displayItem(moreDVDs[3].getType(), moreDVDs[3].getID() );
ApplicationSupport.displayItem(moreDVDs[4].getType(), moreDVDs[4].getID() );
```

ネストされた forItem タグを使用する getCrossSellRecommendations タグの例について、10-30 ページの「forItem パーソナライズ・タグ」も参照してください。

selectFromHotPicks パーソナライズ・タグ

このタグを使用して、分類全体からではなく、また、ユーザー・プロファイルを考慮せずに、一連のホット・ピックス・グループからのリコメンデーションをリクエストします。指定したグループの項目に対し、チューニングおよびフィルタ処理も適用できます。

selectFromHotPicks は、ユーザーの ID およびプロファイルを考慮しないことを除き、fromHotPicksGroups 設定が指定された getRecommendations タグと基本的には同様に機能します。このタグの詳細は、10-23 ページの「getRecommendations パーソナライズ・タグ」を参照してください。

オプションで、タグの storeResultsIn 属性で変数名を指定して、生成されるリコメンデーションを Item[] タイプの TEI 変数内に格納できます。リコメンデーションは、selectFromHotPicks タグ内でも暗黙的に取得可能です。オプションで、希望する項目の処理に getNextItem タグがネストされたタグ・ボディを使用できます。10-31 ページの「getNextItem パーソナライズ・タグ」を参照してください。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:selectFromHotPicks
    hotPicksGroups = "string_of_Hot_Picks_group_numbers"
  [ storeResultsIn = "TEI_variable_name" ]
  [ storeInterestDimensionIn = "TEI_variable_name" ]
  [ maxQuantity = "integer_value" ]
  [ tuningName = "name_from_config_file_Tuning_element" ]
  [ tuningDataSource = "ALL"|"NAVIGATION"|"PURCHASE"|"RATING"|"DEMOGRAPHIC" ]
  [ tuningInterestDimension = "NAVIGATION"|"PURCHASING"|"RATING" ]
  [ tuningPersonalizationIndex = "LOW"|"MEDIUM"|"HIGH" ]
  [ tuningProfileDataBalance = "HISTORY"|"CURRENT"|"BALANCED" ]
  [ tuningProfileUsage = "INCLUDE"|"EXCLUDE" ]
  [ filteringName = "name_from_config_file_Filtering_element" ]
  [ filteringTaxonomyID = "integer_value" ]
  [ filteringMethod = "ALL_ITEMS"|"EXCLUDE_ITEMS"|"INCLUDE_ITEMS"|
           "SUBTREE_ITEMS"|"ALL_CATEGORIES"|"INCLUDE_CATEGORIES"|
           "EXCLUDE_CATEGORIES"|"SUBTREE_CATEGORIES"|CATEGORY_LEVEL" ]
  [ filteringCategories = "string_of_integers" ]
  [ sortOrder = "ASCEND"|"DESCEND"|"NONE" ] >

...

</op:selectFromHotPicks>
```

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- hotPicksGroups（必須）：リコメンデーションの選択元のホット・ピックス・グループを 1 つ以上指定するには、これを使用する必要があります。アプリケーションは、startRESession タグで指定されたものと同じリコメンデーション・エンジンから 1 つ以上のホット・ピックス・グループの ID 番号を決定する必要があります。hotPicksGroups 属性で、「1+20+35」というように、グループの ID 番号をプラス記号（+）で区切って 1 つの文字列にリストしてください。

10-23 ページの「getRecommendations パーソナライズ・タグ」で説明しているように、getRecommendations タグには、「属性の使用方法」で説明する制限事項以外、その他の属性をすべて使用します。

チューニング、フィルタ処理およびソートの詳細は、10-16 ページの「チューニング設定」、10-17 ページの「リコメンデーションのフィルタ処理」および 10-18 ページの「ソート順序」を参照してください。

ネストされた getNextItem タグを使用する selectFromHotPicks タグの例については、10-31 ページの「getNextItem パーソナライズ・タグ」を参照してください。

属性の使用方法

- hotPicksGroups 属性は、getRecommendations タグの fromHotPicksGroups 属性と同等ですが、hotPicksGroups は必須です。
- storeResultsIn 属性、または getNextItem タグがネストされたタグ・ボディのいずれか、あるいは両方（オプション）を使用して出力項目にアクセスします。
- personalization.xml 内の <Tuning> 要素の名前に対応する tuningName を指定するか、または、tuning*XXX* 属性を使用して個別のチューニング設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、「チューニング設定」の 10-16 ページを参照してください。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。
- personalization.xml 内の <Filtering> 要素の名前に対応する filteringName を指定するか、または、filtering*XXX* 属性を使用して個別のフィルタ処理設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。

- filteringMethods が、ALL_ITEMS または ALL_CATEGORIES に設定されていないかぎり、filteringCategories 設定が必要です。これらの設定は、タグ属性または personalization.xml を使用して実行できます。
- マイニング・テーブル・リポジトリで定義されているように、特定の項目ではなく、*XXX*_CATEGORIES フィルタ処理メソッドによりカテゴリが戻されます。

evaluateItems パーソナライズ・タグ

evaluateItems タグを使用して、タグに入力される一連の項目を評価します。項目は、すべて同じ分類に属する必要があります。PURCHASING または NAVIGATION のインタレスト・ディメンションの場合、項目にはランキングが付けられます。RATING のインタレスト・ディメンションの場合、項目にはレーティングが付けられます。評価された項目のサブセット（tuningDataSource 設定の影響によって、項目を含まない場合から、すべての項目を含む場合まで）が、Item[] タイプの TEI 配列変数で戻されます。storeResultsIn 属性で変数名を指定する必要があります。配列内の各項目で、prediction 属性にランキングまたはレーティング値が含まれます。

項目のレーティングおよびランキングに関するバックグラウンド情報の詳細は、10-7 ページの「レーティングとランキング」を参照してください。

項目は、指定した項目配列、または forItem タグがネストされたボディ・タグを使用して入力できます。詳細は、10-13 ページの「入力項目の仕様」を参照してください。10-30 ページの「forItem パーソナライズ・タグ」も参照してください。

> **注意：** 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:evaluateItems
    storeResultsIn = "TEI_variable_name"
    taxonomyID = "integer_value"
  [ inputItemList = "item_array_expression" ]
  [ tuningName = "name_from_config_file_Tuning_element" ]
  [ tuningDataSource = "ALL"|"NAVIGATION"|"PURCHASE"|"RATING"|"DEMOGRAPHIC" ]
  [ tuningInterestDimension = "NAVIGATION"|"PURCHASING"|"RATING" ]
  [ tuningPersonalizationIndex = "LOW"|"MEDIUM"|"HIGH" ]
  [ tuningProfileDataBalance = "HISTORY"|"CURRENT"|"BALANCED" ]
  [ tuningProfileUsage = "INCLUDE"|"EXCLUDE" ]
  [ sortOrder = "ASCEND"|"DESCEND"|"NONE" ] >

...

</op:evaluateItems>
```

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- taxonomyID（必須）：項目の取得元である分類の ID を指定する整数です。
- inputItemList: Item[] によって入力項目を指定する場合は、配列を戻す JSP 式とともにこの属性を使用します。式内の項目配列は、前のリコメンデーション・タグから取得されます。詳細は、10-14 ページの「項目配列の入力」を参照してください。

10-23 ページの「getRecommendations パーソナライズ・タグ」で説明しているように、「属性の使用方法」で説明されている制限事項以外、また evaluateItems タグには storeResultsIn 属性が必要な点以外は、getRecommendations タグには evaluateItems のその他すべての属性を使用します。

チューニングおよびソートの詳細は、10-16 ページの「チューニング設定」および10-18 ページの「ソート順序」を参照してください。

属性の使用方法

- 入力項目には、inputItemList 属性、または forItem タグがネストされたボディ、あるいはその両方（オプション）が必要です。両方の機能を使用する場合は、最初に forItem タグが実行され、示された項目が項目リスト内に配置されます。次に、inputItemList エントリが考慮され、リストに追加されます。
- getRecommendations タグとは異なり、storeResultsIn は、evaluateItems タグの必須属性です。レーティングされた項目を格納するため、Item[] タイプの TEI 変数の名前を指定する必要があります。
- personalization.xml 内の <Tuning> 要素の名前に対応する tuningName を指定するか、または、tuning*XXX* 属性を使用して個別のチューニング設定を指定します。どちらも指定しない場合、デフォルト値を選択する方法の詳細は、10-16 ページの「チューニング設定」を参照してください。10-41 ページの「パーソナライズ・タグ・ライブラリ構成ファイル」も参照してください。
- レーティングされる項目は、単に入力される項目なので、evaluateItems タグには、いずれのフィルタ処理属性も存在しません。したがって、別の属性である taxonomyID を使用して分類を指定する必要があります。

例 この例では、販売項目を入力として取得し、evaluateItems タグを使用してユーザーの関心度の高い順に並べ、最も高いものを表示します。

```
<% Item[] saleItems = ApplicationSupport.getSaleItems(); %>
<!-- Choose the sale items of greatest interest to this user -->
<op:evaluateItems storeResultsIn="bestItems"  taxonomyID="1"
                  inputItemList="<%=saleItems%>" />

<% ApplicationSupport.displayItem(bestItems(1)); %>
```

forItem パーソナライズ・タグ

このタグを使用して、getCrossSellRecommendations タグまたは evaluateItems タグの入力となる各項目を指定します。

forItem タグの使用方法の概要は、10-13 ページの「入力項目の仕様」を参照してください。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:forItem
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ]
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ] />
```

forItem タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- index と itemList の両方を使用します。

または

- type と ID の両方を使用します。

属性

- itemList: Item[] 配列を戻す JSP 式を使用します。式内の項目配列は、前のリコメンデーション・タグから取得されます。この属性は、希望する要素配列を指定する index とともに使用します。type および ID を使用する場合は、この属性を使用しないでください。詳細は、10-14 ページの「項目配列の入力」を参照してください。
- index: 希望する項目配列の要素の索引番号を指定するのに使用します。itemList 属性で項目配列を指定します。type および ID を使用する場合は、この属性を使用しないでください。
- type: たとえば「靴」など、項目タイプを指定します。この属性は、ID とともに使用します。index および itemList を使用する場合は、この属性を使用しないでください。
- ID: 指定したタイプの各項目に一意の識別番号です。この属性は、type とともに使用します。index および itemList を使用する場合は、この属性を使用しないでください。

例 次の例は、指定された複数の靴の購入項目を抱合せ販売のリコメンデーション用の入力として使用し、生成されたリコメンデーションを表示します。

```
<op: getCrossSellRecommendations storeResultsIn="shoeItems" >
   <op:forItem type="shoes" ID="20" />
   <op:forItem type="shoes" ID="26" />
   <op:forItem type="shoes" ID="45" />
   <op:forItem type="shoes" ID="93" />
   <op:forItem type="shoes" ID="101" />
</op:getCrossSellRecommendations>
<p> Based on past shoe purchases, here are the shoes we recommend! </p>
<%= ApplicationSupport.displayItemArray(shoeItems) %>
```

getNextItem パーソナライズ・タグ

オプションで、getRecommendations 内または selectFromHotPicks タグ・ボディにネストされた getNextItem タグを使用して、出力タグによって戻されるリコメンデーションにアクセスできます。（あるいは、getRecommendations または selectFromHotPicks タグの storeResultsIn 属性を使用してもこの項目にアクセスできます。）

getNextItem タグが初めて実行されると最初の項目にアクセスし、その後、項目配列内をとおして getNextItem が引き続き 1 つずつ実行され、次の項目が取得されます。項目配列の最後に達すると、タグは各タグ属性に null 値を与えます。

タグ属性を使用して、次の項目のタイプおよび ID、または Item インスタンス自体のいずれかを格納します。

次の点に注意してください。

- storeResultsIn 属性を使用して、getRecommendations または selectFromHotPicks タグからの明示的な項目配列を使用しても、getNextItem タグの使用は制限されません。storeResultsIn を使用してアクセス可能な項目配列は、getNextItem タグを使用した処理には影響されません。
- 他の getRecommendations 内にネストされた 1 つ以上の getRecommendations タグ、または、他の selectFromHotPicks タグ内の 1 つ以上の selectFromHotPicks タグを使用する場合、これらのタグのうち 1 つのみが、ネストされた getNextItem タグを使用して、生成されるタグに暗黙的にアクセスできます。ネスト・チェーン内の他のタグは、storeResultsIn 属性を使用する必要があります。selectFromHotPicks タグ内の getRecommendations タグ、または getRecommendations タグ内の selectFromHotPicks タグについては、そのような制限事項は存在しません。

構文

```
<op:getNextItem
          [ storeTypeIn = "TEI_variable_for_item_type" ]
          [ storeIDIn = "TEI_variable_for_item_ID" ]
          [ storeItemIn = "TEI_variable_for_Item_instance" ] />
```

getNextItem タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- storeTypeIn と storeIDIn の両方を使用します。

または

- storeItemIn を使用します。

属性 属性の説明に続く、「属性の使用方法」も参照してください。

- storeTypeIn: TEI String 変数名を指定して、次の項目のタイプを格納します。storeItemIn を使用する場合以外は、これを storeIDIn とともに使用します。
- storeIDIn: TEI String 変数名を指定して、次の項目の ID を格納します。storeItemIn を使用する場合以外は、これを storeTypeIn とともに使用します。
- storeItemIn: Item タイプの TEI 変数名を指定して、次の項目を格納します。storeTypeIn および storeIDIn を使用する場合は、この属性を使用しないでください。

属性の使用方法

すべての TEI 変数は AT_END のスコープで、終了タグから JSP ページの終わりまで使用できます。TEI 変数はすべて、ページの前半でスクリプトレット・コードによって宣言され、getNextItem タグのスコープで参照可能である必要があります。他のパーソナライズ・タグ内の TEI 変数とは異なり、これらの変数は、JSP コンテナによっては宣言されません。

例 次の例は、getRecommendations タグ内のループで使用されている getNextItem タグを示しています。このループは、getNextItem が null を戻したときに終了します。

```
<op:getRecommendations from="top"
                       tuningName="BalancedTuning"
                       filteringName="GeneralFiltering" >
<p> Top Picks selected especially for you: </p>
        <% String type=null;
           String ID=null;
           while(true) { %>
             <op:getNextItem storeTypeIn="type" storeIDIn="ID" />
             <% if(type==null) break;%>
             <li> type: <%=type%> ID: <%=ID%> </li>
        <% } %>
</op:getRecommendations>
```

次の例では、selectFromHotPicks タグ内のループの getNextItem を示しています。

```
<op:selectFromHotPicks hotPicksGroups="1+5"
                       tuningName="HotPicksTuning"
                       filteringName="GeneralFiltering" >
<p> We know you enjoy Horror and Musical movies. Look what we have on
sale this week! </p>
        <% Item item=null;
           while(true) { %>
             <op:getNextItem storeItemIn="item" />
             <% if(item==null) break;%>
             <li> <%= ApplicationSupport.displayItem(item) %> </li>
        <% } %>

</op:selectFromHotPicks>
```

項目記録および削除タグの説明

次の項では、record*XXX* および removeXXXRecord タグの詳細を説明します。項目をリコメンデーション・エンジン・セッションのキャッシュに記録するには、適切な record*XXX* タグを使用します。セッション内で以前に記録された項目を削除する場合は、対応する removeXXXRecord タグを使用します。キャッシュ内の項目は、周期的にリコメンデーション・エンジン・セッションにフラッシュされます。フラッシュ後に項目を削除した場合は、データベースのラウンドトリップが必要となります。

- recordNavigation パーソナライズ・タグ
- recordPurchase パーソナライズ・タグ
- recordRating パーソナライズ・タグ
- recordDemographic パーソナライズ・タグ
- removeNavigationRecord パーソナライズ・タグ
- removePurchaseRecord パーソナライズ・タグ
- removeRatingRecord パーソナライズ・タグ
- removeDemographicRecord パーソナライズ・タグ

10-11 ページの「項目の記録および削除タグの概要」も参照してください。

recordNavigation パーソナライズ・タグ

このタグを使用して、ナビゲーション項目をリコメンデーション・エンジン・セッション内に記録します。これは、ユーザーがこの項目にナビゲートすることで関心を示したことを記録します。たとえば、ユーザーが興味のあるアイコンを見て、アイコンの横の**「Tell Me More」**ボタンをクリックする場合などです。ナビゲーション項目の削除用タグの詳細は、10-36 ページの「removeNavigationRecord パーソナライズ・タグ」を参照してください。

startRESession タグの disableRecording 属性を「true」に設定することにより、recordNavigation タグのアクションを無効にできます。詳細は、10-20 ページの「startRESession パーソナライズ・タグ」を参照してください。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:recordNavigation
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

recordNavigation タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

- type: たとえば「靴」など、項目タイプを指定します。この属性は、ID とともに使用します。index および itemList を使用する場合は、この属性を使用しないでください。
- ID: 指定したタイプの各項目に一意の識別番号です。この属性は、type とともに使用します。index および itemList を使用する場合は、この属性を使用しないでください。
- itemList: Item[] 配列を戻す JSP 式を使用します。式内の項目配列は、前のリコメンデーション・タグから取得されます。この属性は、希望する要素配列を指定する index とともに使用します。type および ID を使用する場合は、この属性を使用しないでください。詳細は、10-14 ページの「項目配列の入力」を参照してください。
- index: 希望する項目配列の要素の索引番号を指定するのに使用します。itemList 属性で項目配列を指定します。type および ID を使用する場合は、この属性を使用しないでください。

recordPurchase パーソナライズ・タグ

このタグを使用して、購入項目をリコメンデーション・エンジン・セッション内に記録します。これは、ユーザーが行った購入を記録します。購入項目の削除用タグの詳細は、10-37 ページの「removePurchaseRecord パーソナライズ・タグ」を参照してください。

startRESession タグの disableRecording 属性を「true」に設定することにより、recordPurchase タグのアクションを無効にできます。詳細は、10-20 ページの「startRESession パーソナライズ・タグ」を参照してください。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:recordPurchase
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

recordPurchase タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

属性は、recordNavigation タグの場合と同じです。10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

例 次の 2 つの JSP ページからの抽出部分を検討します。

ページ 1:

```
<%@ page session="true" %>
<op:getRecommendations storeResultsIn "myRecs" />
...display recommendations...
<% session.setAttribute("recommendationList", myRecs); %>
```

ページ 2:

```
<%@ page session="true" %>
<op:recordPurchase itemList="<%=session.getAttribute(¥"recommendationList¥") %>"
                   index="<%=request.getParameter(¥"index¥" %>" />
```

ページ 1 は、リコメンデーションのリストを取得し、各項目を「**Buy**」リンク付きで表示します。項目配列は、使用する後続ページの session オブジェクト内に格納されます。

ページ 2 は、ユーザーが特定のリコメンデーションを購入するため、リンクを選択したときに実行されます。項目リストがセッション属性から取得され、選択された項目の索引がリクエスト・パラメータから取得されます。ページ 2 の例には、「買い物カゴ」があります。

recordRating パーソナライズ・タグ

このタグを使用して、レーティング項目をリコメンデーション・エンジン・セッション内に記録します。これは、この項目に対するユーザーのレーティングに基づきます。レーティング項目の削除用タグの詳細は、10-38 ページの「removeRatingRecord パーソナライズ・タグ」を参照してください。

これは、recordNavigation および recordPurchase とは異なり、レーティング値も指定する必要があります。

> **注意：** 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:recordRating value = "rating_value"
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

recordRating タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

いずれの場合も value 属性が必要です。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

- value（必須）：ユーザーのレーティング値を示す文字列です。整数または浮動小数点の数値を入力できます。数値は、マイニング・テーブル・リポジトリの MTR.MTR_BIN_BOUNDARIES 表の境界に基づいた適切なレーティング範囲内に存在する必要があります。

他の属性は、recordNavigation タグの場合と同じです。10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

recordDemographic パーソナライズ・タグ

このタグを使用して、デモグラフィック項目をリコメンデーション・エンジン・セッション内に記録します。デモグラフィック項目は、特定のユーザーに関する個人的な情報で構成されます。デモグラフィック項目の削除用タグの詳細は、10-38 ページの「removeDemographicRecord パーソナライズ・タグ」を参照してください。

このタグは、他の `recordXXX` タグとは異なり、`type` と `value` の 2 つの属性のみを持ちます。`type` 属性は、`AGE` など、この項目に含まれる情報の種類を示します。`value` 属性には、`44` など、対応する値が含まれます。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:recordDemographic
        type = "GENDER"|"AGE"|"MARITAL_STATUS"|"PERSONAL_INCOME"|
               "HOUSEHOLD_INCOME"|"IS_HEAD_OF_HOUSEHOLD"|"HOUSEHOLD_SIZE"|
               "RENT_OWN_INDICATOR"|"ATTRIBUTE1"|...|"ATTRIBUTE50"
        value = "item_value" />
```

`recordDemographic` タグには、ボディはありません。

属性

- `type`（必須）：サポートされているデモグラフィック・タイプの 1 つを指定します。複数の名前付きタイプに加え、`ATTRIBUTE1`、`ATTRIBUTE2`、`...`、`ATTRIBUTE50` というように、カスタマイズ可能な 50 のタイプがあります。詳細は、10-15 ページの「デモグラフィック項目」を参照してください。
- `value`（必須）：`GENDER` 項目に対して `MALE` または `FEMALE` など、適切な値を指定します。

removeNavigationRecord パーソナライズ・タグ

このタグを使用して、以前にこのセッションでリコメンデーション・エンジン・セッション内に記録されたナビゲーション項目を削除します。ナビゲーション項目の記録用タグの詳細は、10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

項目を削除するには、項目が記録されたものと同じリコメンデーション・エンジン・セッション中に、`removeNavigationRecord` タグを使用する必要があります。セッション・キャッシュは、セッション中、リコメンデーション・エンジン・データベース・スキーマに周期的にフラッシュされます。フラッシュ後に項目を削除する場合は、タグの削除を実行するのにデータベースのラウンド・トリップが必要となります。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:removeNavigationRecord
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

removeNavigationRecord タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

属性は、recordNavigation タグの場合と同じです。10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

removePurchaseRecord パーソナライズ・タグ

このタグを使用して、以前にこのセッションでリコメンデーション・エンジン・セッション内に記録された購入項目を削除します。購入項目の記録用タグの詳細は、10-34 ページの「recordPurchase パーソナライズ・タグ」を参照してください。

項目を削除するには、項目が記録されたものと同じリコメンデーション・エンジン・セッション中に、removePurchaseRecord タグを使用する必要があります。セッション・キャッシュは、セッション中、リコメンデーション・エンジン・データベース・スキーマに周期的にフラッシュされます。フラッシュ後に項目を削除する場合は、タグの削除を実行するのにデータベースのラウンド・トリップが必要となります。

> **注意：** 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:removePurchaseRecord
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

removePurchaseRecord タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

属性は、recordNavigation タグの場合と同じです。10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

removeRatingRecord パーソナライズ・タグ

このタグを使用して、以前にこのセッションでリコメンデーション・エンジン・セッション内に記録されたレーティング項目を削除します。レーティング項目の記録用タグの詳細は、10-35 ページの「recordRating パーソナライズ・タグ」を参照してください。

これは、removeNavigationRecord および removePurchaseRecord とは異なり、レーティング値も指定する必要があります。

項目を削除するには、項目が記録されたものと同じリコメンデーション・エンジン・セッション中に、removeRatingRecord タグを使用する必要があります。セッション・キャッシュは、セッション中、リコメンデーション・エンジン・データベース・スキーマに周期的にフラッシュされます。フラッシュ後に項目を削除する場合は、タグの削除を実行するのにデータベースのラウンド・トリップが必要となります。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:removeRatingRecord value = "rating_value"
          [ type = "type_of_item" ]
          [ ID = "item_ID_number" ]
          [ itemList = "item_array_expression" ]
          [ index = "index_into_item_array" ] />
```

removeRatingRecord タグには、ボディはありません。

このタグには、2 つの使用モードがあります。

- type と ID の両方を使用します。

または

- index と itemList の両方を使用します。

いずれの場合も value 属性が必要です。

関連情報は、10-13 ページの「入力項目の仕様」を参照してください。

属性

- value（必須）: 以前に記録されたユーザーのレーティング値を示す文字列です。

他の属性は、recordNavigation タグの場合と同じです。10-33 ページの「recordNavigation パーソナライズ・タグ」を参照してください。

removeDemographicRecord パーソナライズ・タグ

このタグを使用して、以前にこのセッションでリコメンデーション・エンジン・セッション内に記録されたデモグラフィック項目を削除します。デモグラフィック項目の記録用タグの詳細は、10-36 ページの「recordDemographic パーソナライズ・タグ」を参照してください。

このタグは、他の removeXXXRecord タグとは異なり、type と value の 2 つの属性のみを持ちます。type 属性は、AGE など、この項目に含まれる情報の種類を示します。value 属性には、44 など、対応する値が含まれます。

項目を削除するには、項目が記録されたものと同じリコメンデーション・エンジン・セッション中に、removeDemographicRecord タグを使用する必要があります。セッション・キャッシュは、セッション中、リコメンデーション・エンジン・データベース・スキーマに周期的にフラッシュされます。フラッシュ後に項目を削除する場合は、タグの削除を実行するのにデータベースのラウンド・トリップが必要となります。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。

構文

```
<op:removeDemographicRecord
        type = "GENDER"|"AGE"|"MARITAL_STATUS"|"PERSONAL_INCOME"|
               "HOUSEHOLD_INCOME"|"IS_HEAD_OF_HOUSEHOLD"|"HOUSEHOLD_SIZE"|
               "RENT_OWN_INDICATOR"|"ATTRIBUTE1"|...|"ATTRIBUTE50"
        value = "item_value" />
```

removeDemographicRecord タグには、ボディはありません。

属性

属性は、recordDemographic タグの場合と同じです。10-36 ページの「recordDemographic パーソナライズ・タグ」を参照してください。

項目クラスの説明

OracleAS Personalization タグ・ライブラリには、JSP ページでの項目、カテゴリおよびリコメンデーションの使用を簡易化する、次のような便利なラッパー・クラスが用意されています。

oracle.jsp.webutil.personalization.Item

必要に応じて、タグ・ハンドラは Item インスタンスを作成します。Item インスタンスを必ず使用し、場合によっては直接作成する、2 つの特定の使用例があります。

- リコメンデーション項目からタイプ、ID および予測値を取得する場合

  購入項目またはナビゲーション項目については、予測値はランキングです。レーティング項目については、予測値はランキングです。

- getCrossSellRecommendations および evaluateItems タグの入力項目リスト用にインスタンスを手動で作成する場合

最初の使用例では、Item クラスにより次の getter メソッドが提供されます。

- String getType()

  たとえば「靴」などの項目タイプを戻します。デモグラフィック項目の場合はサポートされているデモグラフィック・タイプの 1 つを戻します。CATEGORY の値は、リコメンデーションされる全体のカテゴリを示します。

- long getID()

  項目の ID 番号を戻します。

- float getPrediction()

  レーティング項目の場合はレーティング、購入項目またはナビゲーション項目の場合はランキングを戻します。ランキングは常に整数ですが、レーティングは浮動小数点の数値もありうるため、この属性は浮動小数点の数値である必要があります。

2 番目の使用例では、次の setter メソッドが提供されます。

- void setType(java.lang.String)

  項目タイプを設定します。

- void setID(long)

  項目の ID 番号を設定します。

項目をカテゴリとして定義し、また、カテゴリとして定義済の場合は判定するメソッドもあります。

- void setCategory()

  項目タイプを CATEGORY に設定します。

- boolean isCategory()

  項目タイプが CATEGORY の場合に true を戻します。

Item クラスにより、次のパブリック・コンストラクタが提供されます。

- Item()
- Item(String type, long ID)
- Item(String type, String ID)

type 属性は文字列である必要がありますが、ID 属性は、文字列または long 値のいずれでも指定できます。

Item クラスは、インタレスト・ディメンションの場合、次の String 定数値も定義します。次の値を使用して、リコメンデーション・タグの storeInterestDimensionIn 属性で戻される値と比較します。

- INT_DIM_NAVIGATION: 関心度の高いナビゲーション項目としてリコメンデーションされる項目を示します。
- INT_DIM_PURCHASING: 関心度の高い購入項目としてリコメンデーションされる項目を示します。
- INT_DIM_RATING: 関心度の高いレーティング項目としてリコメンデーションされる項目を示します。

パーソナライズ・タグの制限事項

OracleAS Personalization タグの属性設定に関して、次の制限事項に注意してください。

- startRESession タグには、次の制限事項があります。
  - RERame 属性の最大許容文字数は 12 です。
  - REURL 属性の最大許容文字数は 256 です。
  - RESchema 属性の最大許容文字数は 30 です。
  - REPassword 属性の最大許容文字数は 30 です。
  - userID 属性の最大許容文字数は 32 です。

  personalization.xml ファイルでは使用されない userID を除き、personalization.xml ファイル内の <RE> 要素の対応する属性にも同じ制限事項が適用されます。
- タグに渡される、またはタグから戻される Item 要素の数は 1024 以下です。この値は、タグ間で渡される 1 つの Item[] 配列の最大サイズであるだけでなく、項目リストと 1 つ以上の forItem タグの両方からタグが入力を受信する場合の合計最大値になります。
- リコメンデーション・タグの getRecommendations、getCrossSellRecommendations および selectFromHotPicks の場合は、最大 1024 のホット・ピックス・グループを指定できます。これは、getRecommendations および getCrossSellRecommendations タグの fromHotPicksGroups 属性、ならびに、fromHotPicksGroups タグの hotPicksGroups 属性に適用されます。
- また、リコメンデーション・タグの場合、filteringCategories 属性で最大 256 のカテゴリを指定できます。

  同等に、personalization.xml ファイルの <Filtering> 要素内には、256 の <Category> サブ要素を含めることができます。
- recordDemographic、removeDemographicRecord、recordRating および removeRatingRecord タグの value 属性の最大長は、60 文字です。

パーソナライズ・タグ・ライブラリ構成ファイル

OracleAS Personalization タグ・ライブラリでは、グローバルおよびデフォルトのタグ属性設定を指定するための personalization.xml という構成ファイルの使用がサポートされています。次の項では、personalization.xml ファイルおよびサポートされている要素について説明します。

personalization.xml ファイル

OracleAS Personalization タグ・ライブラリでは、personalization.xml という構成ファイルがサポートされています。これらのファイルは、オプションのタグ属性にデフォルト設定を指定する場合や、デフォルトおよび名前付きのチューニング設定やフィルタ処理設定を指定する場合に便利です。特にチューニングおよびフィルタ処理設定は、作業が複雑で、複数のタグまたは複数のページに設定する必要があり不便なため、personalization.xml を使用するのが有益です。

特定のアプリケーションに関連する 2 つの personalization.xml ファイルが存在します。

- /WEB-INF/personalization.xml

  このファイルは特定のアプリケーション専用で、アプリケーション全体の設定またはデフォルトに使用します。

- *ORACLE_HOME*/j2ee/home/config/personalization.xml

  これは、サーバー全体の構成ファイルです。タグ属性、または特定のアプリケーションの personalization.xml ファイルで必要な設定が見つからないに場合に使用します。

personalization.xml の要素の説明

この項では、OracleAS Personalization タグ・ライブラリでサポートされている personalization.xml 要素の XML DTD について説明します。これらの要素は、トップレベルの <personalization-config> 要素内に存在します。

パーソナライズ・タグは、personalization.xml ファイルをこの DTD に対して検証します。

注意： 10-40 ページの「パーソナライズ・タグの制限事項」も参照してください。これらの制限事項の一部は、personalization.xml 要素およびタグ属性設定に適用されます。

RecommendationSettings 要素

この要素を使用して、maxQuantity（戻されるリコメンデーションの最大数を指定）、getRecommendations、getCrossSellRecommendations および selectFromHotPicks タグのデフォルト値を設定します。

maxQuantity 設定は、正の整数で示される文字数である必要があります。

定義

```
<!ELEMENT RecommendationSettings EMPTY>
    <!ATTLIST RecommendationSetting  maxQuantity  CDATA #REQUIRED>
```

RE 要素

この要素を使用して、リコメンデーション・エンジンの接続名を指定し、接続します。属性の詳細は、10-20 ページの「startRESession パーソナライズ・タグ」を参照してください。

定義

```
<!ELEMENT RE EMPTY>
   <!ATTLIST RE Name CDATA #REQUIRED>
   <!ATTLIST RE URL CDATA #REQUIRED>
   <!ATTLIST RE Schema CDATA #REQUIRED>
   <!ATTLIST RE Password CDATA #REQUIRED>
   <!ATTLIST RE CacheSize CDATA #REQUIRED>
   <!ATTLIST RE FlushInterval CDATA #REQUIRED>
```

`startRESession` タグの `REName` 属性内の `Name` 属性を参照できます。

OC4J の `jazn-data.xml` ファイルに、リコメンデーション・エンジン・データベース内のアカウント（たとえば、`scott`）用の `<user>` 要素が含まれている場合は、次のように特殊な右矢印構文でダッシュ（`-`）と右山カッコ（`>`）に続けてこのアカウント名を指定することにより、そのファイルから不明瞭化されていないパスワードを取得できます。

```
Password="->scott"
```

Tuning 要素

この要素を使用して、名前付きチューニング設定を定義します。属性の詳細は、10-16 ページの「チューニング設定」を参照してください。

定義

```
<!ELEMENT Tuning EMPTY>
   <!ATTLIST Tuning Name CDATA #REQUIRED>
   <!ATTLIST Tuning  DataSource
             (NAVIGATION|PURCHASING|RATING|DEMOGRAPHIC|ALL) "ALL" >
   <!ATTLIST Tuning InterestDimension (NAVIGATION|PURCHASING|RATING)
             #REQUIRED >
   <!ATTLIST Tuning PersonalizationIndex (LOW|MEDIUM|HIGH) #REQUIRED >
   <!ATTLIST Tuning  ProfileDataBalance (HISTORY|CURRENT|BALANCED)
             #REQUIRED >
   <!ATTLIST Tuning ProfileUsage (INCLUDE|EXCLUDE) "INCLUDE" >
```

`Name` 属性は必須で、また、この一連のチューニング設定に一意の名前を指定して、この名前がリコメンデーション・タグの `tuningName` 属性で参照可能になるようにする必要があります。

リコメンデーション・リクエストのチューニング設定を完全に定義するには、`INCLUDE` のデフォルト値を持つ `ProfileUsage` 以外、他の属性も必須です。詳細は、『Oracle Application Server Personalization プログラマーズ・ガイド』を参照してください。

DefaultTuning 要素

個別のチューニング・タグ属性および `tuningName` タグ属性が（`personalization.xml` 内の対応する `<Tuning>` 要素とともに）ない場合、チューニング設定にはこの要素を使用します。

属性の意味は、前述の `<Tuning>` 要素と同じです。

定義

```
<!ELEMENT DefaultTuning EMPTY>
   <!ATTLIST DefaultTuning  DataSource
             (NAVIGATION|PURCHASING|RATING|DEMOGRAPHIC|ALL) "ALL" >
   <!ATTLIST DefaultTuning InterestDimension (NAVIGATION|PURCHASING|RATING)
             #REQUIRED >
   <!ATTLIST DefaultTuning PersonalizationIndex (LOW|MEDIUM|HIGH)
             #REQUIRED >
   <!ATTLIST DefaultTuning ProfileDataBalance (HISTORY|CURRENT|BALANCED)
             #REQUIRED >
   <ATTLIST! DefaultTuning ProfileUsage (INCLUDE|EXCLUDE) "INCLUDE" >
```

Filtering 要素および Category 要素

これらの要素を使用して、名前付きフィルタ処理設定を定義します。属性の詳細は、10-17 ページの「リコメンデーションのフィルタ処理」を参照してください。

フィルタ処理の Name 属性を使用して、パーソナライズ・タグから参照される一意の名前を指定します。

AllItems および AllCategories サブ要素以外のフィルタ処理サブ要素内に、1 つ以上の <Category> 要素をネストする必要があります。<Category> 要素のコンテンツは、long 整数で示される文字列である必要があります。

定義

```
<!ELEMENT Filtering   (ExcludeItems|IncludeItems|ExcludeCategories|
                       IncludeCategories|CategoryLevel|SubTreeItems|
                       SubTreeCategories|AllItems|AllCategories) >
   <!ATTLIST Filtering Name CDATA #REQUIRED>
   <!ATTLIST Filtering TaxonomyID CDATA #REQUIRED>

<!ELEMENT Category (#PCDATA) >
<!ELEMENT ExcludeItems ( Category+ ) >
<!ELEMENT IncludeItems ( Category+ ) >
<!ELEMENT ExcludeCategories ( Category+ ) >
<!ELEMENT IncludeCategories ( Category+ ) >
<!ELEMENT CategoryLevel ( Category+ ) >
<!ELEMENT SubTreeItems ( Category+ ) >
<!ELEMENT SubTreeCategories ( Category+ ) >
<!ELEMENT AllItems EMPTY >
<!ELEMENT AllCategories EMPTY >
```

DefaultFiltering 要素

個別のフィルタ処理タグ属性および filteringName タグ属性が（personalization.xml 内の対応する <Filtering> 要素とともに）ない場合、フィルタ処理設定にはこの要素を使用します。

定義

```
<!ELEMENT DefaultFiltering (ExcludeItems|IncludeItems|ExcludeCategories|
                            IncludeCategories|CategoryLevel|SubTreeItems
                            SubTreeCategories|AllItems|AllCategories) >
   <!ATTLIST DefaultFiltering TaxonomyID CDATA #REQUIRED>

<!ELEMENT Category (#PCDATA) >
<!ELEMENT ExcludeItems ( Category+ ) >
<!ELEMENT IncludeItems ( Category+ ) >
<!ELEMENT ExcludeCategories ( Category+ ) >
<!ELEMENT IncludeCategories ( Category+ ) >
<!ELEMENT CategoryLevel ( Category+ ) >
<!ELEMENT SubTreeItems ( Category+ ) >
<!ELEMENT SubTreeCategories ( Category+ ) >
<!ELEMENT AllItems EMPTY >
<!ELEMENT AllCategories EMPTY >
```

サンプル personalization.xml ファイル

```
<?xml version="1.0" ?>
<personalization-config>
    <description> Sample personalization config file </description>
   <RecommendationSettings maxQuantity="5" />
   <RE Name="RE1" URL="jdbc:oracle:thin:@sid" Schema="RESCHEMA"
       Password="secret" CacheSize="2999" FlushInterval="30000" />
   <RE Name="RE2" URL="jdbc:oracle:oci:@acme" Schema="RE2-schema"
       Password="RE2-pwd" CacheSize="5555" FlushInterval="100000" />
   <Tuning Name = "tuning1" DataSource="ALL"
           InterestDimension="NAVIGATION"
           PersonalizationIndex="HIGH" ProfileDataBalance="BALANCED"
           ProfileUsage="INCLUDE" />
   <DefaultTuning DataSource="PURCHASING" InterestDimension="RATING"
           PersonalizationIndex="MEDIUM" ProfileDataBalance="CURRENT"
           ProfileUsage="EXCLUDE" />
   <Filtering Name = "filter1" TaxonomyID="25" >
      <CategoryLevel>
         <Category>10</Category>
         <Category>11</Category>
         <Category>15</Category>
      </CategoryLevel>
   </Filtering>
   <DefaultFiltering  TaxonomyID="1" >
      <AllItems/>
   </DefaultFiltering>
</personalization-config>
```

または、RESCHEMA および RE2-schema アカウントに jazn-data.xml エントリがある場合、personalization.xml 内でクリア・テキストで指定するかわりに、次のようにそのエントリから不明瞭化されていないパスワードを取得できます。

```
   <RE Name="RE1" URL="jdbc:oracle:thin:@sid" Schema="RESCHEMA"
       Password="->RESCHEMA" CacheSize="2999" FlushInterval="30000" />
   <RE Name="RE2" URL="jdbc:oracle:oci:@acme" Schema="RE2-schema"
       Password="->RE2-schema" CacheSize="5555" FlushInterval="100000" />
```

11

Web サービス・タグ

OC4J で提供される Oracle のタグ・ライブラリを使用して、開発者は JSP ページを作成し、Web サービスのクライアント・プログラムとして使用できます。この章では、タグ・ライブラリについて説明します。次の項目が含まれます。

- Web サービスの概要
- OC4J Web サービス・タグ

この章では、読者に Web サービス、Simple Object Access Protocol（SOAP）および Web Services Description Language（WSDL）の知識があることを前提としていますが、これらの概要についても説明します。さらに、関連する World Wide Web Consortium（W3C）仕様など、追加ドキュメントの参照先も示します。

OC4J Web サービス・タグ・ライブラリは、Oracle Application Server Web Services に基づいています。詳細は、『Oracle Application Server Web Services 開発者ガイド』を参照してください。

Web サービスの概要

次の各項では、Web サービスの概念について大まかに説明します。

Web サービスの一般的な概要

Web サービスとは、コンピューティング・プラットフォームに関係なくクライアントがインターネット上で起動できる一連のプロシージャまたはアクションのことです。Web サービスは、SOAP、WSDL および UDDI（すべて後述します）など、広範囲に採用されている一連の標準に従って分散コンピューティング環境で緩やかに結合されたコンポーネントで構成されます。たとえば、「ワールド・カップ・サッカー」というサービスは、得点、スケジュールおよび順位表を取得するアクションで構成されます。

Web サービスには、次の機能が必要です。

- 機能、入力属性および出力属性など、サービス自体を記述できる必要があります。Web サービスでは、XML スタイルの WSDL 文書を使用してサービスを記述します。11-3 ページの「Web Services Description Language の主要な要素の概要」を参照してください。
- クライアント・アプリケーションがアクセスできるように、サービスは常に使用可能な状態である必要があります。このための標準的な方法は、Universal Description, Discovery, and Integration（UDDI）ディレクトリに登録することです。パブリック UDDI ディレクトリは、複数のビジネス・グループやユーザー（あるいはインターネット上の任意のユーザー）を集計するのに使用できます。一方、プライベート UDDI ディレクトリは、特定のビジネスまたはグループ内でのみ使用できます。
- クライアント・アプリケーションは、Web サービスを検索して調査した後、標準プロトコルを使用してサービスを起動できる必要があります。Web サービスの主要なプロトコルは、Simple Object Access Protocol（SOAP）です。SOAP では、Web サービスはサーバー側の SOAP サーバーのバックグラウンドに位置し、クライアント・アプリケーションはクライアント側の SOAP サーバーを通過します。データの交換は「SOAP エンベロープ化」され、ファイアウォールを経由してアクセスできます。この SOAP を使用したデータ交換は、概念的には Remote Method Invocation（RMI）によるデータ交換に類似しています。ただし、RMI による交換では、ファイアウォールを通過できません。SOAP の概要は、次の「SOAP の概要と関連機能」を参照してください。
- 起動された Web サービスは、レスポンスを戻し、リクエストされた結果をクライアント・アプリケーションに提供する必要があります。これは、同じ標準プロトコル（SOAP など）によって実行されます。

Web サービス、特に OracleAS Web Services の詳細は、『Oracle Application Server Web Services 開発者ガイド』を参照してください。

関連する仕様（W3C SOAP 仕様、W3C WSDL 仕様および UDDI 仕様）はそれぞれ、次の Web サイトを参照してください。

```
http://www.w3.org/TR/SOAP

http://www.w3.org/TR/wsdl

http://www.uddi.org/specification.html
```

SOAP の概要と関連機能

この項では、SOAP の概要について説明します。詳細は、W3C Simple Object Access Protocol（SOAP）1.1 仕様を参照してください。

SOAP は XML ベースの軽量のプロトコルで、型指定されたデータや構造化データをインターネット上またはその他の分散環境上で交換するために使用されます。これ以外の機能として、SOAP は、リモート・プロシージャ・コール（RPC）およびメッセージ指向のデータ交換をサポートします。

メッセージ指向の実装では、データ交換は、モジュール化パッケージングとエンコーディング・モデルを使用して行われます。メッセージは、操作に関連付けられた入力データ部分と出力データ部分が指定された WSDL コンポーネントです。詳細は、11-4 ページの「Web サービス・メッセージと XML Schema 定義の概要」を参照してください。

RPC はソケットの代替で、プロシージャ・コール・レベルでの通信インタフェースを備えています。RPC はローカル・プロシージャのコールに似ていますが、実際には、コールの引数はパッケージ化されてリモート・ターゲットに送信されます。RPC 機能では、リクエスト / レスポンスの方法論を利用しています。この場合、エンド・ポイントでは、プロシージャ指向のメッセージを受信し、対応するレスポンスを返信します。

SOAP を RPC とともに使用する場合は、プロトコル・バインディングに依存しません。HTTP がプロトコル・バインディングの場合、HTTP リクエストは RPC コールに対応し、HTTP レスポンスは RPC レスポンスに対応しています。

SOAP の主な特長は、次のとおりです。

- SOAP エンベロープ構成メンバー : エンベロープは、SOAP ヘッダーと SOAP ボディを囲み、メッセージの内容、必須かどうか、および処理の責任者などを示します。
- SOAP エンコード・ルール : アプリケーションで使用されているデータ型のインスタンスの交換に使用するシリアライズ機能を定義します。
- SOAP RPC 表現 : RPC コールとレスポンスを表現する表記規則を指定します。

Web Services Description Language の主要な要素の概要

Web サービスは、XML ベースの Web Services Description Language を使用して WSDL（`.wsdl`）文書に記述されます。

次に、WSDL の主要な用語を示します。

- 操作 : サービスが実行する特定のアクション。ワールド・カップ・サービスの例では、「得点の取得」、「スケジュールの取得」および「順位表の取得」がこれに該当します。
- メッセージ : 操作に対する入力および出力対象のデータを指定する抽象的な定義。
- ポート・タイプ : サービスがサポートする操作の抽象的な定義。
- バインド : サービスがサポートする 1 つ以上の操作に対するプロトコルとデータ・フォーマットの仕様。バインディング機能は、データ・エンコーディング、メッセージ・プロトコルおよび通信プロトコルなどの Web サービスの一般的または抽象的な定義を具体的な実装にマップします。
- ポート : 単一のエンド・ポイント。つまり、バインドとネットワーク・アドレスの組合せです。実質的には、ポートは、ポート・タイプによって記述される機能を具体的に表したものです。SOAP ベースの実装では、ポートは SOAP の位置です。

  より正確に言うと、Web サービスは、単なる抽象的なアクションや操作のコレクションではなく、関連するポート、つまりエンド・ポイントのコレクションです。

WSDL の仕様では、WSDL 文書の一般的な構造の概要について説明しています。次の主要な要素が含まれています。詳細は、W3C Web Services Description Language（WSDL）1.1 仕様を参照してください。

- `<types>` 要素には、1 つ以上の `<schema>` サブ要素を使用して、サービスの操作で使用されるメッセージで交換されるデータの説明を含めます。

- <message> 要素には、操作の入力または出力として送信されるデータの抽象的な定義を含めます。
- <portType> 要素には、1 つ以上の <operation> サブ要素を使用して、Web サービス操作の抽象的な定義を含めます。<operation> 要素では、操作の入力に使用されるメッセージと出力に使用されるメッセージを指定します。
- <binding> 要素は、<operation> サブ要素を使用して、各操作を特定のプロトコル、および使用するデータ・フォーマットにバインドします。
- <service> 要素は、Web サービスのポート、つまりエンド・ポイントを定義します。<service> 要素内には、1 つ以上の <port> サブ要素があり、各 <port> 要素によって、バインディングはエンド・ポイントを定義するアドレスに結合されます。

Web サービス・メッセージと XML Schema 定義の概要

メッセージは、Web サービスの操作、つまりメソッドで使用されるパラメータを定義します。メッセージは、1 つ以上のパートで構成される通信対象データの型指定された定義です。各パートは、「購入発注書」パートや「請求書」パートなどの論理エンティティに相当します。各パートには、関連付けられたデータ項目に対する型指定があります。

OracleAS Web Services などの SOAP ベースの実装では、メッセージで使用するデータ型は、XML Schema Definition（XSD）言語を使用して定義します。この言語は、事前定義の単純型とユーザー定義の複合型をサポートしています。

XSD を使用した実装の場合、メッセージを定義する構文は、次のとおりです。

```
<message name="nmtoken">
   <part name="nmtoken" [type="qname"] [element="qname"] />
</message>
```

この構文では、element 属性は、XSD 構文を使用して定義された XSD 複合型の場所を参照します。type 属性は XSD 単純型を、「nmtoken」は標準 XML の名前トークンを、「qname」は標準 XML の修飾名をそれぞれ示しています。メッセージ、および各メッセージのパートは 0（ゼロ）以上です。

SOAP エンコーディング・スタイルの encoded では、使用できるのは単純型のみであるため、element 属性は使用されません。エンコーディング・スタイルが literal の場合は、単純型も複合型も設定できます。したがって、<part> 要素には、type 属性か element 属性のいずれかを使用できます。ただし、両方は使用できません。

次に、後述の「例 : WSDL 定義」から抜粋したメッセージ定義の例を示します。

```
<message name="GetLastTradePriceInput">
   <part name="body" element="xsd1:TradePriceRequest"/>
</message>
```

GetLastTradePriceInput は、メッセージ名です。その名前が示すとおり、これは入力メッセージです。この場合、element 属性は、複合型の TradePriceRequest が定義されている名前空間を参照します。次に、その定義例（後述の「例 : WSDL 定義」の一部）を示します。

```
<element name="TradePriceRequest">
   <complexType>
      <all>
          <element name="tickerSymbol" type="string"/>
          <element name="companyName" type="string"/>
      </all>
   </complexType>
</element>
```

XML Schema の入門書は、次の W3C のサイトから入手できます。

http://www.w3.org/TR/xmlschema-0/

Web サービスの例

この例では、Web サービスの WSDL 定義を示し、HTTP リクエストと HTTP レスポンスにそれぞれ埋め込まれた入力メッセージと出力メッセージの例を示します。

例 : WSDL 定義

W3C Web Services Description Language（WSDL）1.1 仕様には、次の WSDL 文書の例が記載されています。この例では、入力として株価銘柄記号を受け取り、出力として現在の株価を戻す、株式相場サービスを定義しています。SOAP エンコーディング・スタイルの `literal` を使用しているため、複合型が使用できます（実際に使用されています）。

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
         <element name="TradePriceRequest">
            <complexType>
               <all>
                  <element name="tickerSymbol" type="string"/>
               </all>
            </complexType>
         </element>
         <element name="TradePrice">
            <complexType>
               <all>
                  <element name="price" type="float"/>
               </all>
            </complexType>
         </element>
      </schema>
   </types>

   <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>

   <message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
   </message>

   <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
         <input message="tns:GetLastTradePriceInput"/>
         <output message="tns:GetLastTradePriceOutput"/>
      </operation>
   </portType>

   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
      <soap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
         <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
            <input>
               <soap:body use="literal"/>
            </input>
```

```
            <output>
               <soap:body use="literal"/>
            </output>
         </operation>
      </binding>

      <service name="StockQuoteService">
         <documentation>My first service</documentation>
            <port name="StockQuotePort" binding="tns:StockQuoteBinding">
               <soap:address location="http://example.com/stockquote"/>
             </port>
      </service>

</definitions>
```

WSDL 定義では、最初に入力メッセージ GetLastTradePriceInput および出力メッセージ GetLastTradePriceOutput を指定し、次にこれらを操作 GetLastTradePrice に結合した後、その操作のバインドとポートを定義します。

注意 :

- この例では、データ交換に対する XML Schema 定義も含め、Web サービス定義のあらゆる側面が同一文書内で設定されています。別の方法として、たとえば stockquote.xsd は、この文書内の名前空間ではなく、個別の XSD 文書にすることもできます。この方法については、W3C WSDL 仕様で説明しています。ただし、OC4J Web サービス・タグ・ライブラリでは、<import> 要素を使用して他の WSDL 文書をインポートする WSDL 文書はサポートしていないことに注意してください。
- この例では、ドキュメント・スタイルのバインディングを使用しています。OC4J 10.1.2 の Web サービス・タグ・ライブラリ実装では、RPC スタイルとドキュメント・スタイルのバインディングがサポートされます。ドキュメント・スタイルの場合、出力レスポンス・オブジェクトは XMLElement 型の XML ドキュメントです。RPC スタイルの場合、出力オブジェクトは様々な型になります。

例 : HTTP リクエストとレスポンスに埋め込まれた SOAP メッセージ

前述の例で定義されている Web サービスに対応して、この項ではメッセージの外観、つまり HTTP リクエストに埋め込まれ SOAP エンベロープ化された入力メッセージと HTTP レスポンスに埋め込まれ SOAP エンベロープ化された出力メッセージを示します。次の例は、W3C Web Services Description Language（WSDL）1.1 仕様からの抜粋です。

次にリクエストを示します。

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "SOAP_URI"

   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <m:GetLastTradePrice xmlns:m="xmlns_URI">
            <m:tickerSymbol>DIS</m:tickerSymbol>
         </m:GetLastTradePrice>
      </soapenv:Body>
   </soapenv:Envelope>
```

この例の *xmlns_URI* は、GetLastTradePrice 操作とそのメッセージが定義されている名前空間を識別するために使用される URI 値です（定義例については、前述の「例 : WSDL 定義」にある WSDL 文書を参照してください）。また、tickerSymbol も定義されています。リクエストは、Walt Disney 社の株価です。*SOAP_URI* は、SOAP の HTTP バインドに対する SOAP アクションの HTTP ヘッダーを示す URI です。

次にレスポンスを示します。

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <m:GetLastTradePriceResponse xmlns:m="Some_URI">
            <m:price>34.5</m:price>
         </m:GetLastTradePriceResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

操作 *Xxxx* に対するレスポンスは、慣例的に *Xxxx*Response と呼ばれます。*Some_URI* は、GetLastTradePriceResponse 操作が定義されている名前空間を識別するために使用される URI 値です。

OC4J Web サービス・タグ

次の各項では、Web サービス・タグ・ライブラリの概要と詳細、およびタグ・ライブラリ実装のベースである OracleAS Web Services の概要について説明します。

- OracleAS Web Services の概要とタグ・ライブラリの実装
- Web サービス・タグの機能の概要
- Web サービス・タグの説明
- Web サービス・タグの例

OracleAS Web Services の概要とタグ・ライブラリの実装

OC4J が提供する Web サービス・タグ・ライブラリを使用すると、開発者は、Web サービスのクライアント・アプリケーション用 JSP ページを簡単に作成できます。実装には、SOAP ベースの機能を使用します。クライアント・アプリケーションでは、WSDL 文書にアクセスした後、WSDL 情報を使用して Web サービスの操作にアクセスします。

また、このタグ・ライブラリでは、Oracle による動的起動 API の実装を使用します。この実装の説明は、『Oracle Application Server Web Services 開発者ガイド』に記載されています。クライアント・アプリケーションで実行時に WSDL 文書を取得する場合は、動的起動 API を使用して、WSDL 文書に記述されている SOAP 操作を起動します。タグ・ハンドラでは、Web サービスを起動する SOAP リクエストの送信時と SOAP レスポンスの処理時に API を使用します。

Oracle の動的起動 API は、oracle.j2ee.ws.client および oracle.j2ee.ws.client.wsdl パッケージのクラスとインタフェースで構成されています。

oracle.j2ee.ws.client パッケージには、次の内容が含まれています。

- WebServiceProxyFactory: WSDL 文書を（文書が含まれている Java 入力ストリームまたは文書の URL を使用して）指定することで、WebServiceProxyFactory インスタンスは、WSDL 文書に指定されているサービス名とそのポート名の 1 つを使用して、WebServiceProxy インスタンス（WebServiceProxy インタフェースを実装するクラスのインスタンス）を作成できます。

- WebServiceProxy: このインタフェースを使用して、WSDL 文書に定義されているサービスを表します。各 WebServiceProxy インスタンスは、WSDL 文書の位置に基づいています。また、必要に応じて、使用するサービスとポートを識別する追加の修飾子に基づきます。WebServiceProxy クラスは、WSDL 文書で公開された全操作の構文とシグネチャも含め、WSDL ポート・タイプを決定するメソッドと定義済の操作を起動するメソッドを公開します。
- WebServiceMethod: このインタフェースを使用して、Web サービスのメソッド、つまり操作を起動します。

oracle.j2ee.ws.client.wsdl パッケージには、次の内容が含まれています。

- Operation: WSDL 操作を表します。
- Message: 操作の入力または出力に使用するメッセージを表します。
- Part: メッセージ・パートを表します。
- Input: 入力メッセージを表します。
- Output: 出力メッセージを表します。

注意： 動的起動 API は、*ORACLE_HOME*/lib ディレクトリの dsv2.jar にパッケージ化されています。また、SOAP 実装の場合は、*ORACLE_HOME*/soap ディレクトリ内に soap.jar が必要です。

Web サービス・タグの機能の概要

この項では、OC4J Web サービス・タグ・ライブラリとその機能の概要について説明します。このタグ・ライブラリは、次の機能をサポートします。

- Web サービスへのバインド
- SOAP リクエストと SOAP レスポンスを使用した Web サービス操作の使用
- 入力および出力用メッセージ・パートの定義
- SOAP/XML データ型の Java 型へのマッピング
- クライアント・アプリケーションで使用するカスタム・プロパティの設定

また、タグ・ライブラリは、次の名前空間を持つバージョンの W3C XML Schema を使用した WSDL 文書で定義された起動操作をサポートしています。

```
http://www.w3.org/2001/XMLSchema
```

Web サービス・タグ・ライブラリには、webservice タグ（必要に応じて map タグと property タグをネストできます）および invoke タグ（必要に応じて part タグをネストできます）が含まれています。これらのタグは、次のように使用します。

- webservice: Web サービス・プロキシを作成します。このタグには、WSDL 文書の URL が必要です。したがって、次の組合せのいずれかを使用します。
  - バインドと SOAP 位置（UDDI レジストリで識別される WSDL 文書に役立ちます）。
  - サービス名とポート（タグ属性または WSDL 文書の最初のサービスとその最初のポートを使用して指定します）。
- map: このタグが指定されている場合、Web サービス・プロキシは、このタグを使用して SOAP マッピング・レジストリにエントリを追加します。このレジストリは、ローカルの SOAP/XML 型を Java 型にマップするレジストリです。任意の数の map タグを 1 つの webservice タグ内にネストできます。つまり、任意の型マッピングごとに 1 つのタグをネストできます。
- property: 必要に応じて、このタグを使用して、Web サービスのクライアント・アプリケーションで使用できるいくつかのカスタム・プロパティのいずれかを定義します。property タグはすべて、webservice タグ内にネストする必要があります。プロパティには、親の Web サービスと同じスコープが設定されます。

- invoke: このタグを使用して Web サービスの操作を起動します。invoke タグが Web サービス・プロキシにアクセスするには、webservice タグ内にネストされているか、あるいはスクリプト変数を使用する必要があります。
- part: 操作に入力メッセージ・パートが含まれている場合は、invoke タグ内で part タグをネストして使用し、メッセージ・パートを定義します。パートごとに 1 つの part タグを使用します。

注意 :

- タグ・ライブラリは、WSDL 文書内での <import> 要素を使用した他の WSDL 文書のインポートはサポートしていません。
- カスタム HTTP バインディングまたはカスタム MIME バインディングなどのカスタムのバインディングは、サポートされません。

OC4J Web サービス・タグ・ライブラリの実装は、OracleAS Web Services の実装に基づいているため、OracleAS Web Services のその他の制限はタグ・ライブラリにも適用されます。

Web サービス・タグの説明

次の各項では、JavaServer Pages タグ・ライブラリの標準準拠の実装である OC4J Web サービス・タグの詳細および構文表記について説明します。

- Web サービスの webservice タグ
- Web サービスの map タグ
- Web サービスの property タグ
- Web サービスの invoke タグ
- Web サービスの part タグ

Web サービス・タグ・ライブラリを使用する場合は、次の要件に注意してください。

- Web サービス・タグ・ライブラリは、ojsputil.jar ファイルに含まれています。このファイルは、OC4J に同梱されていて、予約済のタグ・ライブラリ・ディレクトリにあります。このファイルがインストール済で、クラスパスに存在していることを確認してください。
- タグ・ライブラリ・ディスクリプタ・ファイル wstaglib.tld が、アプリケーションで使用可能である必要があります。また、ライブラリを使用する JSP ページには、適切な taglib ディレクティブが存在する必要があります。Oracle Application Server のインストール時、TLD は ojsputil.jar に配置されます。wstaglib.tld の uri 値は次のとおりです。

  ```
  http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/wstaglib.tld
  ```

taglib ディレクティブ、予約済のタグ・ライブラリ・ディレクトリ、TLD ファイルおよび uri 値の内容の詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

この項で説明するタグの使用例は、11-14 ページの「Web サービス・タグの例」を参照してください。

注意 :

- このタグ構文では、接頭辞「ws:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

Web サービスの webservice タグ

このタグを使用して、Web サービス・プロキシ、つまり oracle.j2ee.ws.client.WebServiceProxy インタフェースを実装するクラスのインスタンスを作成します。このタグには、WSDL 文書の URL が必要です。次のように、バインディングと SOAP 位置、またはサービス名とポートのいずれかを使用します。

1. タグ属性にバインディングと SOAP 位置が指定されている場合、タグ・ハンドラはそれらを使用してプロキシを作成します。この場合、サービス名とポートのタグ属性は無視されます。
2. バインディングと SOAP 位置が指定されていない場合、タグ・ハンドラはサービス名とポートを次のように使用します。
   a. タグ属性によってサービス名とポートが指定されている場合、タグ・ハンドラはそれを使用してプロキシを作成します。
   b. サービス名とポートが指定されていない場合、タグ・ハンドラは、WSDL 文書の最初のサービスとそのサービスの最初のポートを使用します。

UDDI レジストリを介して WSDL 文書にアクセスする Web サービスの場合は、バインディングと SOAP 位置を使用すると非常に便利です。その場合は、UDDI 問合せを使用してバインディングと位置を決定し、リクエスト時の式を使用してタグに設定できます。

作成された Web サービス・プロキシは、ネストされた任意の map タグを使用して、エントリを SOAP マッピング・レジストリに追加します。次項の「Web サービスの map タグ」を参照してください。

構文

```
<ws:webservice wsdlUrl = "WSDL_URL_of_Web_service"
               [ id = "variable_name_for_Web_service_proxy" ]
               [ scope = "page" | "request" | "session" | "application" ]
               [ binding = "SOAP_binding_information" ]
               [ soapLocation = "SOAP_endpoint_URL" ]
               [ service = "service_name_in_WSDL" ]
               [ port = "port_name_for_service" ] >

...body / nested tags...

</ws:webservice>
```

注意： scope 属性は、リクエスト時の式を取得できません。

属性

- wsdlUrl（必須）: 該当する Web サービスの WSDL にアクセスできる URL を指定します。
- id: webservice タグ内にネストされていない invoke タグでアクセスする Web サービスの場合は、id 属性を使用して、WebServiceProxy スクリプト変数の名前を指定します。その結果、invoke タグでこの変数を参照できます。指定する名前は、有効な Java 識別子である必要があります。id 属性を使用する場合、指定した変数は、スコープ AT_END（webservice 終了タグから JSP ページの終わりまで使用可能）で自動的に宣言されます。
- scope: 必要に応じて、webservice タグのスコープを指定します。デフォルト設定は「page」です。
- binding: 前述の使用例 1 の場合は、binding 属性を使用して、SOAP 位置（エンド・ポイントの URL）の SOAP バインド情報を指定し、SOAP 位置は soapLocation 属性で指定します。これらの属性は一緒に使用する必要があります。WSDL 文書で定義済のバインド情報によって、特定のポート・タイプによって定義された操作とメッセージに対する具体的なプロトコルとデータ・フォーマットの仕様が指定されます。

- soapLocation: 前述の使用例 1 の場合は、soapLocation を使用して、WSDL 文書で定義済の SOAP 位置（エンド・ポイントの URL）を指定します。この場合、binding 属性で指定したバインド情報が適用されます。これらの属性は一緒に使用する必要があります。
- service: 前述の使用例 2 の場合は、service 属性を使用して WSDL 文書で定義済のサービスの名前を指定します。この属性は、port 属性とともに使用する必要があります。ただし、binding と soapLocation を使用すると、いずれも無視されます。
- port: 前述の使用例 2 の場合は、port 属性を使用してサービスのポートを指定します。サービスは、service 属性によって指定されます。これらの属性は一緒に使用する必要があります。Web サービス・プロキシは指定されたポートを使用します。ポート・アドレスは、WSDL 文書内の対応する <service> 要素で指定されます。binding 属性と soapLocation 属性を使用する場合、service 属性と port 属性は無視されます。

Web サービスの map タグ

相互運用性を維持するには、WSDL 定義の SOAP/XML データ型を Java クライアント・アプリケーションの JSP ページで使用される Java 型にマップするマッピング機能が必要です。これは、OracleAS Web Services の SOAP マッピング・レジストリによって可能になります。

任意の数の map タグを webservice タグ内にネストすると、Web サービス・プロキシはエントリをレジストリに追加できます。該当する型マッピングごとに 1 つの map タグを使用します。

レジストリは、org.apache.soap.util.xml パッケージの XMLJavaMappingRegistry クラスのインスタンスです。WebServiceProxy インスタンスには、レジストリにアクセスする getXMLMappingRegistry() メソッドがあります。

map タグには、型マッピングに役立つエンコーディング・スタイル、シリアライズ・ユーティリティ（クラス名）、デシリアライズ・ユーティリティ（クラス名）および名前空間 URI などを指定する属性が含まれています。Web サービス・タグ・ライブラリは、シリアライズ・ユーティリティとデシリアライズ・ユーティリティのカスタム・クラスをサポートしているため、これらを独自に作成することも可能です。

重要： map タグの使用時は、このタグを webservice タグ内にネストする必要があります。

構文

```
<ws:map localName = "local_name_of_SOAPXML_type"
        namespaceUri = "URI_of_namespace_for_SOAPXML_type"
        javaType = "Java_type_to_map"
        encodingStyle = "URL_of_SOAP_encoding_style"
        java2xmlClassName = "Java_to_XML_serializer"
        xml2javaClassName = "XML_to_Java_deserializer" />
```

属性

- localName（必須）: SOAP/XML データ型のローカル名（SOAPStruct など）を指定します。
- namespaceUri（必須）: SOAP/XML データ型の名前空間に対して有効な URI を指定します。次に例を示します。

  ```
  http://soapinterop.org/xsd
  ```

- javaType（必須）: SOAP/XML 型にマップする Java 型を指定します。指定する型は、正式にマップ可能な型である必要があります。
- encodingStyle（必須）: SOAP のエンコード・スタイルに対して有効な URI を指定します。次に例を示します。

  ```
  http://schemas.xmlsoap.org/soap/encoding
  ```

- java2xmlClassName（必須）: Java から XML に変換するデータをシリアライズする機能を備えたクラス名を指定します。カスタムのクラスでも構いません。次に例を示します。

  ```
  org.apache.soap.encoding.soapenc.BeanSerializer
  ```

- xml2javaClassName（必須）: XML から Java に変換するデータをデシリアライズする機能を備えたクラス名を指定します。カスタムのクラスでも構いません。次に例を示します。

  ```
  org.apache.soap.encoding.soapenc.BeanSerializer
  ```

Web サービスの property タグ

必要に応じて、このタグを使用して名前 / 値ペアを指定し、Web サービスのクライアント・アプリケーションで使用できる複数のカスタム・プロパティを定義します。たとえば、ネットワーク・ファイアウォールを経由してアクセスするためにプロキシが必要な場合は、property タグを使用して、HTTP プロキシのホストとポートを指定できます。次のプロパティがサポートされます。

- http.proxyHost: HTTP プロキシ・サーバーのホスト名を指定します。
- http.proxyPort: HTTP プロキシ・サーバーのポート番号を指定します。
- javax.net.ssl.KeyStore: Oracle セキュリティ Wallet ファイルのフルパスを指定します。

> **重要：** property タグの使用時は、このタグを webservice タグ内にネストする必要があります。プロパティには、親の Web サービスと同じスコープを設定します。

構文

```
<ws:property name="http.proxyHost" | "http.proxyPort" | "javax.net.ssl.KeyStore"
             value = "property_value" />
```

属性

- property（必須）: 設定するプロパティを指定します。タグ構文のリストにある、サポート対象プロパティの 1 つを指定する必要があります。
- value（必須）: プロパティの値（ホスト名、ポート番号または Oracle Wallet ファイルのフルパス）を指定します。

Web サービスの invoke タグ

このタグを使用して Web サービスの操作を起動します。タグ・ハンドラは、リモートの Web サービス操作をコールし、SOAP リクエストに入力メッセージを渡して、SOAP レスポンスを待機します。操作を指定するだけでなく、戻されるレスポンスを格納するオブジェクトのオブジェクト ID を指定する必要があります。タグ・ハンドラでは、操作名を使用して、WSDL 文書内でその操作を検索します。

invoke タグは、次のいずれかの方法で Web サービス・プロキシにアクセスできます。

- invoke タグを、プロキシを設定する webservice タグ内にネストします。
- invoke タグに webservice 属性を指定し、webservice タグの id 属性を使用して作成された WebProxyService スクリプト変数にアクセスします。

オーバーロードされた操作（異なる I/O メッセージを使用した同名の 2 つの操作）がある場合、invoke タグには、それぞれの操作に対する入力および出力メッセージ名を指定する属性がいくつか含まれます。この場合、RPC スタイルのバインディングでは、指定した入力メッセージ名と出力メッセージ名を使用して、操作の RPC シグネチャが構成されます。それ以外の場合、RPC シグネチャは、WSDL 文書に従ったデフォルトとなります。

出力メッセージに複数のパートが含まれている場合、戻される結果は、メッセージ・パート（すべて単一の SOAP レスポンス内のパート）の配列になります。

invoke タグは XML プロデューサとして動作でき、toXMLObjName 属性を使用して XML 出力オブジェクトの明示的な受渡しをサポートします。これは、Web Object Cache タグや XML transform タグなど、他の種類のタグに invoke タグをネストする場合に便利です。また、XML 出力オブジェクトを JSP ページの JspWriter オブジェクトに書き込んで、ユーザーのブラウザに直接出力できます。この機能は、xmlToWriter 属性を使用して有効化します。

注意：

- SOAP レスポンスを待機すると、機能がブロックされます。
- 出力結果オブジェクトのスコープ（id 属性で識別）は、Web サービスの webservice タグに定義されているプロキシ・オブジェクトのスコープと同じです。webservice タグ内でネストした invoke タグの場合、このスコープは webservice 開始タグから webservice 終了タグまでです。ただし、id オブジェクトには JSP ページのコンテキスト・オブジェクトの findAttribute() メソッドを使用して、webservice タグ以外からもアクセスできます。

構文

```
<ws:invoke id = "variable_name_for_output_result"
           operation = "operation_to_invoke"
         [ webservice = "variable_name_of_Web_service_proxy" ]
         [ inputMsgName = "name_of_input_message" ]
         [ outputMsgName = "name_of_output_message" ]
         [ xmlToWriter = "true" | "false" ]
         [ toXMLObjName = "objectname" ] >

...body / nested tags...

</ws:invoke>
```

属性

- id（必須）: 出力結果オブジェクトのスクリプト変数名を指定します。指定する名前は、有効な Java 識別子である必要があります。id オブジェクトのスコープについては、前述のタグ構文の前の「注意」を参照してください。
- operation（必須）: 実行する操作（WSDL 文書の操作）を指定します。
- webservice: 起動するサービスに対応している WebServiceProxy スクリプト変数の名前を指定する場合は、この属性を使用します。invoke タグが、任意のサービスにアクセスする webservice タグ内にネストされている場合、この属性は不要です。
- inputMsgName: 必要に応じて、操作に対する入力メッセージ名（WSDL 文書の wsdl:input タグの名前）を指定します。これが必要なのは、オーバーロードされた操作（異なるメッセージ名を使用した複数の同名の操作）がある場合のみです。
- outputMsgName: 必要に応じて、操作に対する出力メッセージ名（WSDL 文書の wsdl:output タグの名前）を指定します。これが必要なのは、オーバーロードされた操作（異なるメッセージ名を使用した複数の同名の操作）がある場合のみです。
- xmlToWriter: ドキュメント・スタイル Web サービスの場合に、出力が XML オブジェクトで、XML を JSP ページの JspWriter 出力オブジェクトに出力するには、この属性を「true」に設定します。デフォルト設定は「false」です。
- toXMLObjName: ドキュメント・スタイル Web サービスの場合に、出力を XML オブジェクトで明示的に渡すには、この属性を使用してオブジェクト名を指定します。

Web サービスの part タグ

操作の実行に入力メッセージ・パートの値が必要な場合は、入力パートごとに 1 つの part タグを invoke タグ内でネストさせて使用します。

パートの値の指定方法は、RPC スタイルとドキュメント・スタイルのうち、どちらの Web サービスを使用しているかに応じて異なります。RPC スタイルの場合は、value 属性を使用する必要があります。ドキュメント・スタイルの場合は、タグ・ボディに XML リクエスト要素を使用して値を渡す方法があります。

注意 : タグ・ボディと value 属性の両方を使用すると、タグ・ボディは無視されます。

構文

```
<ws:part name = "part_name"
       [ value = "part_value" ] >

...optional body, with request element, for document-style...

</ws:part>
```

属性

- name（必須）: 入力パートの名前（有効な Java 識別子）を指定します。
- value: 入力パートの値を指定します。これは、RPC スタイルの Web サービスには必須です。ドキュメント・スタイルの Web サービスの場合は、このかわりにタグ・ボディを使用できます。

Web サービス・タグの例

この項では、Web サービス・タグ・ライブラリで使用するテンプレート、RPC スタイル Web サービスを起動するサンプル JSP ページ、およびドキュメント・スタイル Web サービスを起動するサンプル・ページを示します。

Web サービスの例 : テンプレートの使用方法

```
<HTML>
<HEAD>
<TITLE>Title</TITLE>
</HEAD>
<BODY>
<H2>This is sample HTML text.</H2>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/wstaglibrary.tld"
           prefix="ws" %>
<ws:webservice id="myws"
               wsdlUrl="wsdlurl"
               {
                  binding="" soapLocation="" | service="" port=""
               }
               {
                  scope="page | request | session | application"
               }
                >
   <ws:property name="property" value="string"/>

   <ws:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           localname="SOAPStruct"
           namespaceUri="http://soapinterop.org/xsd"
           javaType="MySoapStructBean"
           java2xmlClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
```

```
            xml2javaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
    />

</ws:webservice>

<ws:invoke id="result" webservice="myws" operation="add" inputMsgName=""
            outputMsgName="">
   <ws:part name="part_name" value="{string | <%= expression %>}"/>
</ws:invoke>

<% =result %>
</BODY>
</HTML>
```

Web サービスの例 : RPC スタイル Web サービスのサンプル JSP ページ

```
<%@ page contentType="text/html"%>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/wstaglib.tld"
            prefix="ws" %>
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; ">
</HEAD>
<BODY>
<%
 String itemID = request.getParameter("itemID");
%>
<ws:webservice id="ebay"
    wsdlUrl="http://www.xmethods.net/sd/2001/EBayWatcherService.wsdl"
    binding="eBayWatcherBinding"
    soapLocation="http://services.xmethods.net:80/soap/servlet/rpcrouter"
    scope="page">
  <ws:property name="http.proxyHost" value="www-proxy.us.oracle.com"/>
  <ws:property name="http.proxyPort" value="80"/>
</ws:webservice>
<ws:invoke id="price" webservice="ebay" operation="getCurrentPrice">
  <ws:part name="auction_id" value="<%=itemID%>"/>
</ws:invoke>
<B>
Action price for eBay Item # <%=itemID%> is :
</B>
<P>
$<%= price%>
@
<%= new java.util.Date()%>
</P>
</BODY>
</HTML>
```

Web サービスの例 : ドキュメント・スタイル Web サービスのサンプル JSP ページ

```
<%@ page contentType="text/xml;"%>
<%@ page  import= oracle.xml.parser.v2.XMLElement;"%>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/wstaglib.tld"
           prefix="ws" %>
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xml.tld"
           prefix="xml" %>
<ws:webservice id="bookService"
        wsdlUrl="http://hosting.msugs.ch/cheeso9/books/books.asmx?WSDL"
        binding="LookyBookServiceSoap"
        soapLocation ="http://hosting.msugs.ch/cheeso9/books/books.asmx"
        scope="session">
</ws:webservice>
<ws:invoke id="bookResult"
        operation="GetInfo"
        webservice="bookService">
    <ws:part name="parameters">
        <GetInfo xmlns="http://dinoch.dyndns.org/webservices/">
          <ISBN>SomeISBNNumber</ISBN>
        </GetInfo>
    </ws:part>
</ws:invoke>
<%
    XMLNode resultNode = (XMLNode) bookResult;
    resultNode.Error! Bookmark not defined.(new java.io.PrintWriter(out));
%>
</BODY>
</HTML>
```

A

JML コンパイル時構文とタグ

重要： JML タグ・ライブラリは OC4J 10.1.2 実装で廃止され、それ以降の実装でサポート外になる予定です。かわりに、標準の JSTL タグ・ライブラリを使用してください。

JSP タグ・ライブラリ・フレームワークは、JSP 1.1 仕様に組み込まれました。JSP 仕様 1.1 が実装される前の Oracle JSP リリースでは、Oracle 固有のトランスレータ拡張機能としてのみ JML タグをサポートできました。このマニュアルでは、これを「コンパイル時タグ・サポート」と呼びます。

OC4J の JSP リリースでは、継続してコンパイル時 JML 実装をサポートしています。ただし、通常は可能なかぎり、標準準拠の実行時実装を使用することをお薦めします。実行時実装については、第 3 章「JSP Markup Language タグ」で説明しています。

この付録では、実行時実装とは共通する点のないコンパイル時実装の機能について説明します。次の項目が含まれます。

- JML コンパイル時構文のサポート
- JML コンパイル時タグのサポート

コンパイル時実装を使用するメリットについての一般的な説明は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

JML コンパイル時構文のサポート

次の各項では、Oracle 固有の Bean 参照構文と Bean 式構文について説明します。これらの構文は、コンパイル時 JML 実装によってサポートされ、タグの属性値を指定するために使用されます。

- JML Bean の参照と式、コンパイル時実装
- JML 式による属性設定

この機能には、OC4J JSP トランスレータが必要です。この機能は他の JSP 環境には移植できません。

JML Bean の参照と式、コンパイル時実装

Bean 参照とは、JavaBean インスタンスへの参照を指します。この参照によって Bean のプロパティまたはメソッドのいずれかにアクセスします。この参照には、Bean 自体が別の Bean のプロパティである場合の、Bean のプロパティまたはメソッドへの参照が含まれます。

この参照は煩雑になります。それは、標準の JavaBeans 構文では、直接参照ではなく、アクセッサ・メソッドをコールしてプロパティにアクセスする必要があるためです。次の直接参照の例を考えてみます。

```
a.b.c.d.doIt()
```

この例は、標準の JavaBeans 構文では、次のように表します。

```
a.getB().getC().getD().doIt()
```

ただし、Oracle のコンパイル時 JML 実装には、簡略化された構文が用意されています。この構文については、次の各項で説明します。

JML Bean 参照

コンパイル時 JML 実装がサポートする Oracle 固有の構文を使用すると、Bean 参照をダイレクト・ドット（.）表記法を使用して表記できます。標準の Bean のプロパティ・アクセッサ・メソッド構文も有効です。

次の標準の JavaBean 参照を考えてみます。

```
customer.getName()
```

JML Bean 参照構文では、次のいずれかの方法でこれを表すことができます。

```
customer.getName()
```

または

```
customer.name
```

JavaBeans は、デフォルトのプロパティを必要に応じて持つことができます。参照が明示的に記述されていない場合には、このプロパティが参照されます。JML Bean 参照では、デフォルトのプロパティ名を省略できます。前述の例で、name がデフォルトのプロパティの場合は、次のすべてが有効な JML Bean 参照になります。

```
customer.getName()
```

または

```
customer.name
```

または

```
customer
```

ほとんどの JavaBeans では、デフォルトのプロパティを定義しません。デフォルトのプロパティを定義する JavaBeans の中で、最も重要なのは、第 2 章「拡張型用の JavaBeans」で説明した JML データ型 JavaBeans です。

JML 式

コンパイル時 JML 実装がサポートする JML 式の構文は、標準の JSP 式構文のスーパーセットです。前項で説明した JML Bean 参照構文もサポートされます。

JML 式に配置される JML Bean 参照は、次の構文で囲む必要があります。

`$[`*`JML_bean_reference`*`]`

JML 式による属性設定

3-3 ページの「JSP Markup Language（JML）タグの説明」では、タグ属性の標準準拠の構文について説明しています。その項で説明しているように、属性は、実行時またはコンパイル時の JML 実装のいずれにも設定できます。また Oracle 以外の JSP 環境にも設定できます。

ただし、Oracle 固有のコンパイル時実装のみを使用する場合は、前項の「JML Bean の参照と式、コンパイル時実装」で説明しているように、JML Bean 参照と JML 式の構文を使用して、属性を設定できます。次の要件に注意してください。

- 第 3 章で説明されている、文字列リテラルまたは式を受け入れる属性については、標準の JSP `<%=...%>` 構文内の `$[...]` 構文で JML 式を使用できます。

  JML `useVariable` タグの使用例を考えてみます。実行時実装には、次のような構文を使用します。

  ```
  <jml:useVariable id = "isValidUser" type = "boolean"
                   value = "<%= dbConn.isValid() %>" scope = "session" />
  ```

  コンパイル時実装には、このかわりに次のような構文も使用できます（`value` 属性は、文字列リテラルまたは式のいずれかです）。

  ```
  <jml:useVariable id = "isValidUser" type = "boolean"
                   value = "<%= $[dbConn.valid] %>" scope = "session" />
  ```

- 第 3 章で説明されている、式のみを受け入れる属性については、`<%=...%>` 構文にネストせずに、その `$[...]` 構文で JML 式を使用できます。

  JML `choose...when` タグの使用例を考えてみます。実行時実装には、次のような構文を使用します（`orderedItem` は `JmlBoolean` インスタンスとします）。

  ```
  <jml:choose>
       <jml:when condition = "<%= orderedItem.getValue() %>"  >
                 You have changed your order:
               -- outputs the current order --
       </jml:when>
       <jml:otherwise>
                 Are you sure we can't interest you in something?
       </jml:otherwise>
  </jml:choose>
  ```

  コンパイル時実装には、このかわりに次のような構文も使用できます（`condition` 属性は、式のみです）。

  ```
  <jml:choose>
       <jml:when condition = "$[orderedItem]"  >
                 You have changed your order:
               -- outputs the current order --
       </jml:when>
       <jml:otherwise>
                 Are you sure we can't interest you in something?
       </jml:otherwise>
  </jml:choose>
  ```

JML コンパイル時タグのサポート

この項では、次の項目を説明します。

- コンパイル時 JML サポートで使用する必要がある taglib ディレクティブのドキュメント。
- すべてのコンパイル時タグのサマリー。実行時実装で非サポートのタグの説明もあります。
- 実行時実装では非サポートで、コンパイル時実装でサポートされるタグの説明。

実行時実装でもサポートされているタグについては、3-3 ページの「JSP Markup Language（JML）タグの説明」で説明しています。

注意： 多くの場合、実行時実装で非サポートの JML タグには、それに相当する標準の JSP タグがあります。ただし、一部のコンパイル時タグは、サポートされていません。これは、これらのタグに、現行の JSP 仕様に準拠する場合に実装が困難な機能が含まれているためです。

コンパイル時 JML サポートに使用する taglib ディレクティブ

Oracle コンパイル時 JML サポートの実装では、カスタム・クラスの `OpenJspRegisterLib` を使用して、JML タグ・サポートを実装します。

コンパイル時実装で JML タグを使用する JSP ページでは、標準 JSP のタグ・ライブラリを使用する場合のように TLD ファイルを指定するのではなく、`taglib` ディレクティブで、このクラスの完全修飾名を指定する必要があります。

```
<%@ taglib uri="oracle.jsp.parse.OpenJspRegisterLib" prefix="jml" %>
```

JML 実行時実装での `taglib` ディレクティブの使用方法の詳細は、3-2 ページの「JSP Markup Language（JML）タグ・ライブラリの概要」を参照してください。

JML タグのサマリー、コンパイル時と実行時の比較

ほとんどの JML タグは、実行時モデルとコンパイル時モデルの両方で使用できます。ただし、次の各表のサマリーに示すような例外があります。

表 A-1 Bean バインディング・タグ : コンパイル時モデルと実行時モデルの比較

タグ	Oracle コンパイル時実装でのサポートの有無	Oracle 実行時実装でのサポートの有無
useBean	有	無、`jsp:useBean` を使用
useVariable	有	有
useForm	有	有
useCookie	有	有
remove	有	有

表 A-2 Bean 操作タグ : コンパイル時モデルと実行時モデルの比較

タグ	Oracle コンパイル時実装でのサポートの有無	Oracle 実行時実装でのサポートの有無
getProperty	有	無、`jsp:getProperty` を使用
setProperty	有	無、`jsp:setProperty` を使用
set	有	無
call	有	無
lock	有	無

表 A-3 フロー制御タグ : コンパイル時モデルと実行時モデルの比較

タグ	Oracle コンパイル時実装でのサポートの有無	Oracle 実行時実装でのサポートの有無
if	有	有
choose	有	有
for	有	有
foreach	有、type 属性はオプション	有、type 属性は必須
return	有	有
flush	有	有
include	有	無、jsp:include を使用
forward	有	無、jsp:forward を使用

表 A-4 XML タグ : コンパイル時モデルと実行時モデルの比較

タグ	Oracle コンパイル時実装でのサポートの有無	Oracle 実行時実装でのサポートの有無
transform	廃止予定	有
styleSheet	廃止予定	有

表 A-5 ユーティリティ・タグ : コンパイル時モデルと実行時モデルの比較

タグ	Oracle コンパイル時実装でのサポートの有無	Oracle 実行時実装でのサポートの有無
print	有、二重引用符で文字列リテラルを指定	無、JSP 式を使用
plugin	有	無、jsp:plugin を使用

注意 : Oracle9*i*AS リリース 2（9.0.3）からは、transform タグと styleSheet タグをコンパイル時実装に使用することはお薦めしません。

追加 JML タグの説明、コンパイル時実装

次の各項では、JML コンパイル時実装ではサポートされ、JML 実行時実装ではサポートされない JML タグの詳細を説明します。実行時実装でサポートされているタグについては、3-3 ページの「JSP Markup Language（JML）タグの説明」で説明しています。

注意：

- このタグ構文では、接頭辞「jml:」が使用されます。慣例的にこのように表記しますが、必須ではありません。任意の接頭辞を taglib ディレクティブに指定できます。
- このマニュアルのタグ構文規則の詳細は、1-2 ページの「タグ構文の表記と意味」を参照してください。

JML useBean タグ

このタグは、ページで使用するオブジェクトを宣言します。以前にインスタンス化したオブジェクトが存在する場合は、指定したスコープでそのオブジェクトを名前で検索します。オブジェクトが存在しない場合、このタグは適切なクラスの新規インスタンスを作成し、指定したスコープに名前で連結します。

構文とセマンティクスは、標準の jsp:useBean タグの場合と同じです。ただし、JSP 式が jsp:useBean の使用で有効な場合は、JML useBean の使用では JML 式または JSP 式のいずれかが有効です。

jsp:useBean タグの概要は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

構文

```
<jml:useBean id = "beanInstanceName"
[ scope ="page" | "request" | "session" | "application" ]
  class ="package.class" |
  type = "package.class" |
  class ="package.class" type = "package.class" |
  beanName = "package.class" | "<%= jmlExpression %>" type = "package.class"  />
```

setProperty タグなど、追加でタグをネストし、</jml:useBean> 終了タグで終了することもできます。

属性

id の指定に加えて、class または type（あるいは class および type）または beanName を指定する必要があります。

jsp:useBean 属性とその構文の詳細は、Sun 社の JSP 仕様を参照してください。

例

```
<jml:useBean id = "isValidUser" class = "oracle.jsp.jml.JmlBoolean" scope = "session" />
```

JML getProperty タグ

このタグは、機能的には標準の jsp:getProperty タグと同一です。このタグは、Bean プロパティの値をレスポンスに出力します。

getProperty の使用方法の一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』または Sun 社の JSP 仕様を参照してください。

構文

```
<jml:getProperty name = "beanInstanceName"
                 property = "propertyName" />
```

属性

- name（必須）: 取得するプロパティを持つ Bean の名前です。
- property（必須）: 取得するプロパティの名前です。

例 次の例では、salary プロパティの現行の値を出力します。salary は JmlNumber 型です。

```
<jml:getProperty name="salary" property="value" />
```

これは、次の例と同じです。

```
<%= salary.getValue() %>
```

JML setProperty タグ

このタグは、標準の jsp:setProperty タグがサポートする機能をサポートします。さらに JML 式のサポート機能が追加されています。特に、JML Bean 参照を使用できます。

setProperty の使用方法の一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』または Sun 社の JSP 仕様を参照してください。

構文

```
<jml:setProperty name = "beanInstanceName"
                 property = " * " |
                 property = "propertyName" [ param = "parameterName" ] |
                 property = "propertyName"
                [ value = "stringLiteral" | "<%= jmlExpression %>" ] />
```

属性

- name（必須）: 設定するプロパティを持つ Bean の名前です。
- property（必須）: 設定するプロパティの名前です。
- value: リクエスト・パラメータを使用せずに、値を直接設定できるオプションのパラメータです。JML setProperty タグは、値の指定で、標準の JSP 式に加えて、JML 式もサポートしています。

例 次の例では、salary を 6 パーセント増額して更新します。salary は JmlNumber 型です。

```
<jml:setProperty name="salary" property="value" value="<%= $[salary] * 1.06 %>" />
```

これは、次の例と同じです。

```
<% salary.setValue(salary.getValue() * 1.06); %>
```

JML set タグ

このタグには、Bean プロパティの設定に、setProperty タグの構文よりも便利な構文を使用する別の設定方法が用意されています。

構文

```
<jml:set name = "beanInstanceName.propertyName"
         value = "stringLiteral" | "<%= jmlExpression %>" />
```

属性

- name（必須）: 設定する Bean プロパティへの直接参照（JML Bean 参照）です。
- value（必須）: 新規プロパティの値です。この値は、文字列リテラル、JML 式または標準の JSP 式で表します。

例 次の各例では、salary を 6 パーセント増額して更新します。salary は JmlNumber 型です。

```
<jml:set name="salary.value" value="<%= salary.getValue() * 1.06 %>" />
```

または

```
<jml:set name="salary.value" value="<%= $[salary.value] * 1.06 %>" />
```

または

```
<jml:set name="salary" value="<%= $[salary] * 1.06 %>" />
```

これらは、次の例と同じです。

```
<% salary.setValue(salary.getValue() * 1.06); %>
```

JML call タグ

このタグには、何も戻さない Bean メソッドを起動する機能が備わっています。

構文

```
<jml:call method = "beanInstanceName.methodName(parameters)" />
```

属性

- method（必須）: スクリプトレットに書き込むメソッドのコールです。ただし、その文の *beanInstanceName.methodName* 部分が、JML 式の $[...] 構文で囲まれている場合は、JML Bean 参照として書き込むことができます。

例 次の例では、クライアントを別のページにリダイレクトします。

```
<jml:call name='response.sendRedirect("http://www.oracle.com/")' />
```

これは、次の例と同じです。

```
<% response.sendRedirect("http://www.oracle.com/"); %>
```

JML lock タグ

このタグによって、タグ・ボディ内で使用するコードの名前付きオブジェクトへの同期アクセスを制御できます。

通常、JSP 開発者は、並行性の問題を心配する必要はありません。ただし、アプリケーション・スコープ・オブジェクトは、アプリケーションを稼働している全ユーザー間で共有されるため、重要なデータへのアクセスは制御および調整する必要があります。

JML lock タグを使用すると、異なるユーザーによる同時更新を防止できます。

構文

```
<jml:lock name = "beanInstanceName" >
   ...body...
</jml:lock>
```

属性

- name（必須）: lock タグ・ボディでのコードの実行時に、ロックが必要なオブジェクトの名前です。

例 次の例の pageCount は、アプリケーション・スコープの JmlNumber 値です。変数は、このコードが現行の値を取得して新規の値を設定するまでの間、別のユーザーによる値の更新を防止するためにロックされます。

```
<jml:lock name="pageCount" >
    <jml:set name="pageCount.value" value="<%= pageCount.getValue() + 1 %>" />
</jml:lock>
```

これは、次の例と同じです。

```
<% synchronized(pageCount)
   {
      pageCount.setValue(pageCount.getValue() + 1);
   }
%>
```

JML include タグ

このタグは、インクルード先ページ（include を起動するページ）に応じて、別の JSP ページ、サーブレットまたは HTML ページの出力をインクルードします。このタグには、標準の jsp:include タグと同じ機能があります。ただし、このタグでは、page 属性を JML 式として表すこともできます。

include の使用方法の一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』または Sun 社の JSP 仕様を参照してください。

構文

```
<jml:include page = "relativeURL" | "<%= jmlExpression %>"
             flush = "true" | "false" />
```

属性

include 属性とその使用方法の一般情報は、Sun 社の JSP 仕様を参照してください。

例 次の例では、table.jsp の出力、つまり問合せ文字列とリクエスト属性のデータに基づいて、HTML 表をレンダリングするプレゼンテーション・コンポーネントの出力をインクルードします。

```
<jml:include page="table.jsp?maxRows=10" flush="true" />
```

JML forward タグ

このタグは、リクエストを別の JSP ページ、サーブレットまたは HTML ページに転送します。このタグには、標準の jsp:forward タグと同じ機能があります。ただし、このタグでは、page 属性を JML 式として表すこともできます。

forward の使用方法の一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

構文

```
<jml:forward page = "relativeURL" | "<%= jmlExpression %>" />
```

属性

forward 属性の一般情報は、Sun 社の JSP 仕様を参照してください。

例

```
<jml:forward page="altpage.jsp" />
```

JML print タグ

このタグには、<%= *expr* %> 形式の標準の JSP 式と基本的に同じ機能があります。指定した JML 式または文字列リテラルを評価し、結果をレスポンスに出力します。このタグでは、JML 式を <%= ...%> 構文で囲む必要はありません。ただし、文字列リテラルは、二重引用符で囲みます。

構文

```
<jml:print eval = '"stringLiteral"' | "jmlExpression" />
```

属性

- eval（必須）: 評価および出力する文字列または式を指定します。

例 次の例のいずれも、JmlNumber 型の salary の現行の値を出力します。

```
<jml:print eval="$[salary]" />
```

または

```
<jml:print eval="salary.getValue()" />
```

次の例では、文字列リテラルを出力します。

```
<jml:print eval='"Your string here"' />
```

JML plugin タグ

このタグは、機能的には標準の jsp:plugin タグと同一です。

plugin の使用方法の一般情報は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』または Sun 社の JSP 仕様を参照してください。

B

サード・パーティ・ライセンス

この付録には、Oracle Application Server に付属するすべてのサード・パーティ製品のサード・パーティ・ライセンスが含まれます。この付録には次の項目が含まれています。

Apache HTTP Server

Apache のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Apache ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または Apache から提供されません。

Apache ソフトウェア・ライセンス

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
```

```
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
```

Jaxen

Oracle はサード・パーティ・ライセンスの本文を表示することが求められていますが、サード・パーティ・プログラムは Oracle ライセンスの対象となり、Oracle はサード・パーティのテクノロジに対する保証および技術サポートを提供しないものとします。

このプログラムには、Jaxen のサード・パーティ・コードが組み込まれています。Jaxen のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Jaxen ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Jaxen ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または Jaxen から提供されません。

Jaxen ソフトウェア・ライセンス

Copyright (C) 2000-2002 bob mcwhirter & James Strachan. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "Jaxen" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jaxen.org.
4. Products derived from this software may not be called "Jaxen", nor may "Jaxen" appear in their name, without prior written permission from the Jaxen Project Management (pm@jaxen.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgment equivalent to the following: "This product includes software developed by the Jaxen Project (http://www.jaxen.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jaxen.org/.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE Jaxen AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Jaxen Project and was originally created by bob mcwhirter and James Strachan . For more information on the Jaxen Project, please see http://www.jaxen.org/.

SAXPath

Oracle はサード・パーティ・ライセンスの本文を表示することが求められていますが、サード・パーティ・プログラムは Oracle ライセンスの対象となり、Oracle はサード・パーティのテクノロジに対する保証および技術サポートを提供しないものとします。

このプログラムには、SAXPath のサード・パーティ・コードが組み込まれています。SAXPath のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（SAXPath ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、SAXPath ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または SAXPath から提供されません。

SAXPath ソフトウェア・ライセンス

Copyright (C) 2000-2002 werken digital. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "SAXPath" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@saxpath.org.
4. Products derived from this software may not be called "SAXPath", nor may "SAXPath" appear in their name, without prior written permission from the SAXPath Project Management (pm@saxpath.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgment equivalent to the following: "This product includes software developed by the SAXPath Project (http://www.saxpath.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.saxpath.org/.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SAXPath AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the SAXPath Project and was originally created by bob mcwhirter and James Strachan . For more information on the SAXPath Project, please see http://www.saxpath.org/. */

索引

I

J

L

M

O

P

R

S

T

U

W

X

あ

い

か

き

く

こ

さ

し

せ

そ

た

ち

て

な

は

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

れ