BEA WebLogic Server

BEA WebLogic Express™

WebLogic JSP Tag Extensions

プログラマーズ ガイド

BEA WebLogic Server 6.1
マニュアルの日付：2001 **年** 12 **月** 19 **日**

著作権

限定的権利条項

商標または登録商標

WebLogic JSP Tag Extensions プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
	2001 年 12 月 19 日	WebLogic Server バージョン 6.1

目次

5. 管理とコンフィグレーション

索引

このマニュアルの内容

このマニュアルでは、カスタム JavaServer Pages（JSP）タグと JSP タグ ライブラリを記述およびデプロイする方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「JSP タグ拡張のプログラミングの概要」では、JSP タグの機能とデプロイメントについて概説します。
- 第 2 章「カスタム JSP タグを作成する主な手順」では、カスタム JSP タグの作成および使用に必要な手順を示します。
- 第 3 章「タグ ライブラリ記述子の作成」では、タグ ライブラリ記述子（TLD）ファイルを作成する方法について説明します。
- 第 4 章「タグ ハンドラの実装」では、拡張タグの機能を実装する Java クラスを記述する方法について説明します。
- 第 5 章「管理とコンフィグレーション」では、JSP タグ拡張を使用するための管理タスクとコンフィグレーション タスクについて簡単に説明します。

対象読者

このマニュアルは、Sun Microsystems の Java 2 Platform, Enterprise Edition（J2EE）を使用して e- コマース アプリケーションを構築するアプリケーション開発者を対象としています。Web 技術、オブジェクト指向プログラミング技術、および Java プログラミング言語に読者が精通していることを前提として書かれてます。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

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

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

関連情報

- Sun Microsystems の JSP 1.1 仕様
- 『WebLogic JSP プログラマーズ ガイド』
- 『Web アプリケーションのアセンブルとコンフィグレーション』

サポート情報

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

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

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

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

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

表記法	適用

[ ]	構文の中で任意指定の項目を示す。
	例 :
	`java utils.MulticastTest -n name -a address`
	`[-p portnumber] [-t timeout] [-s send]`

|	構文の中で相互に排他的な選択肢を区切る。
	例 :
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる ■ 任意指定の引数が省略されている ■ パラメータや値などの情報を追加入力できる
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 JSP タグ拡張のプログラミングの概要

JSP 1.1 仕様には、JavaServer Pages（JSP）ページでカスタム タグを作成および使用するための機能が導入されました。カスタム タグは、Web ページ作成者が使いやすく管理しやすいように、Web ページのプレゼンテーションからビジネス ロジックの複雑さを分離するための優れた方法です。JSP ページでカスタム JSP タグ拡張を使用すると、動的コンテンツを生成できます。また、さまざまな Web 開発ツールを使用して、プレゼンテーションを作成できるようになります。

WebLogic Server は、JSP 1.1 仕様に定義されているタグ拡張メカニズムを完全にサポートしています。

以下の節では、JSP タグ拡張の概要を示します。

- カスタム タグ機能の概要
- JSP でのカスタム タグの使い方
- タグ ライブラリの参照

カスタム タグ機能の概要

カスタム JSP タグを作成するには、タグ ハンドラと呼ばれる Java クラスを作成します。タグ ハンドラは、以下のいずれかの方法で作成します。

- タグのライフ サイクルの間に呼び出されるメソッドを定義する `Tag` または `BodyTag` インタフェースを実装します。
- `Tag` または `BodyTag` インタフェースを実装する抽象基本クラスを拡張します。

抽象基本クラスを拡張すると、タグ ハンドラ クラスがそのインタフェース内のすべてのメソッドを実装する必要がなくなります。また、他の便利な機能を使用できるようになります。`TagSupport` クラスと `BodyTagSupport` クラスが `Tag` または `BodyTag` インタフェースを実装し、API に組み込まれます。

1 つまたは複数の JSP タグをタグ ライブラリに登録することができます。タグ ライブラリは、タグ ライブラリ記述子（TLD）ファイルによって定義されます。TLD は各タグの構文を記述し、その機能を実行する Java クラスにそのタグを関連付けます。

JSP でのカスタム タグの使い方

カスタム タグは、以下のタスクを実行できます。

- 出力を生成します。タグの出力はその親スコープに送られます。スコープは以下のいずれかです。
  - そのタグが JSP ページに直接含まれる場合には、その JSP ページの出力が親スコープとなります。
  - そのタグが別の親タグ内でネストされている場合、その出力はその親タグの評価済み本体の一部になります。
- JSP ページの中でスクリプト変数として参照および使用できる新しいオブジェクトを定義します。タグは、固定名前付きスクリプト変数を導入するか、`id` 属性を備えた動的名前付きスクリプト変数を定義できます。
- 一定の条件が満たされるまで、そのタグ本体のコンテンツの処理を繰り返します。これは、出力を反復的に生成するか、またはサーバサイド アクションを繰り返し呼び出すために使用します。
- その JSP ページの残りの部分を要求の一部として処理するか、それともスキップするかを決定します。

カスタム タグのフォーマット

カスタム タグのフォーマットは、空タグと呼ばれる空のものか、または本体付きタグと呼ばれる本体を含むものです。どちらのタイプのタグも、そのタグを実装する Java クラスに渡される複数の属性を受け付けます。詳細については、4-6 ページの「タグ本体内の例外処理」を参照してください。

空タグの形式は次のとおりです。

```
<mytaglib:newtag attr1="aaa" attr2="bbb" ... />
```

本体付きタグの形式は次のとおりです。

```
<mytaglib:newtag attr1="aaa" attr2="bbb" ... >
  本体
</mytaglib:newtag>
```

タグ本体には、多くの JSP 構文、およびネストされた本体を持つ別のカスタム JSP タグを入れることができます。タグは、お互いの内部で任意のレベルにネストできます。次に例を示します。

```
<mytaglib:tagA>
   <h2> これは tagA の本体です </h2>
   このテキストはこれまでに <mytaglib:counter /> 回現れました !
  <p>
  <mytaglib:repeater repeat=4>
     <p>Hello World!
  </mytaglib:repeater>
</mytaglib:tagA>
```

上の例では、3 つのカスタム タグを使用して、本体付きタグの中にタグをネストする方法が示されています。これらのタグは、以下のように機能します。

- 本体付きタグ <mytaglib:tagA> は、その評価済み本体からの HTML 出力しか認識しません。つまり、ネストされた JSP タグの <mytaglib:counter> と <mytaglib:repeater> がまず評価され、それらの出力が <mytaglib:tagA> タグの評価済み本体の一部になります。
- 本体付きタグの本体がまず JSP として評価され、ネストされた本体付きタグも含め、その中に入っているすべてのタグが変換され、さらにそれらの本体が再帰的に評価されます。評価済み本体の結果が本体付きタグの出力として直接使用されるか、または本体付きタグがその評価済み本体のコンテンツに基づいて出力を決定します。
- 本体付きタグの JSP から生成される出力は、通常の HTML として取り扱われます。つまり、出力はさらに JSP として解釈されることはありません。

サンプル シナリオ

以下のシナリオに、カスタム タグを使用して何ができるかを示します。

- 空タグは、その属性に基づいてサーバサイドの作業を実行できます。そのタグが実行するアクションによって、ページの残りの部分を解釈するか、それともリダイレクトなどの別のアクションを実行するかが決められます。この機能は、ユーザがページにアクセスする前にログインしたかどうかをチェックし、ログインしていなければログイン ページにリダイレクトする場合に役立ちます。
- 空タグは、その属性に基づいてページにコンテンツを挿入できます。このようなタグを使用すると、ページのヒット数を数える単純なカウンタや、その他のテンプレート ベースの挿入を実装できます。
- 空タグは、その属性に基づいて、ページの残りの部分で使用可能なサーバサイド オブジェクトを定義できます。このタグを使用すると、同じ JSP ページ内の他の場所でデータのクエリを受けることができる EJB への参照を作成することができます。
- 本体付きタグは、ブラウザに送信される HTML ページの一部となる前にその出力を処理し、その出力を評価して、ブラウザに送信する HTML を決定できます。この機能は、「引用 HTML」、つまり再フォーマットされたコンテンツを生成するためか、または SQL クエリなどの他の関数に渡すパラメータとして使用できます（SQL クエリの場合、タグの出力はフォーマットされた結果セットとなる）。
- 本体付きタグは、特定の条件が満たされるまで、その本体を繰り返し処理できます。

タグ ライブラリの参照

JSP タグ ライブラリは、タグ ライブラリ記述子 (`tld`) によって定義されます。JSP ページからカスタム タグ ライブラリを使用するには、以下のように `<%@ taglib %>` ディレクティブを使用してそのタグを参照します。次に例を示します。

```
<%@ taglib uri="myTLD" prefix="mytaglib" %>
```

uri

JSP エンジンは、Web アプリケーション デプロイメント記述子（web.xml）の <taglib-uri> 要素で定義し uri とこの uri 属性を照合することによって、タグ ライブラリ記述子ファイルを見つけようとします。たとえば、上の taglib ディレクティブの myTLD は、次のように Web アプリケーション デプロイメント記述子のタグ ライブラリ記述子（library.tld）を参照します。

```
<taglib>
  <taglib-uri>myTLD</taglib-uri>
  <taglib-location>library.tld</taglib-location>
</taglib>
```

prefix

prefix 属性は、タグ ライブラリにラベルを割り当てます。カスタム JSP タグを使用してページを作成する場合は、このラベルを使って関連するタグ ライブラリを参照します。たとえば、上の例のライブラリ（mytaglib）が newtag という新しいタグを定義した場合、このタグを JSP ページで使用するには、以下のように記述します。

```
<mytaglib:newtag>
```

詳細については、3-1 ページの「タグ ライブラリ記述子の作成」を参照してください。

2 カスタム JSP タグを作成する主な手順

カスタム JSP タグを作成および使用するには、以下の手順を実行します。

1. タグ ハンドラ クラスを記述します。JSP でカスタム タグを使用する場合、このクラスはそのタグの機能を実行します。タグ ハンドラ クラスは、`javax.servlet.jsp.tagtext.BodyTag` または `javax.servlet.jsp.tagtext.Tag` という 2 つのインタフェースのいずれかを実装します。タグ ハンドラ クラスは、タグ ライブラリの一部として実装されます。詳細については、4-1 ページの「タグ ハンドラの実装」を参照してください。
2. JSP ソース内で、JSP `<taglib>` ディレクティブを使用してタグ ライブラリを参照します。タグ ライブラリは、JSP タグを集めたものです。このディレクティブを JSP ソースの先頭に挿入します。詳細については、1-4 ページの「タグ ライブラリの参照」を参照してください。
3. タグ ライブラリ記述子（TLD）を記述します。TLD は、タグ ライブラリを定義し、タグ ハンドラ クラス名、属性、タグに関するその他の情報など、各タグに関する追加情報を提供します。詳細については、3-1 ページの「タグ ライブラリ記述子の作成」を参照してください。
4. タグ ライブラリ記述子を、Web アプリケーション デプロイメント記述子（web.xml）で参照します。詳細については、「Web アプリケーションのデプロイメント記述子の記述」を参照してください。
5. JSP でカスタム タグを使用します。詳細については、1-2 ページの「JSP でのカスタム タグの使い方」を参照してください。

3　タグ ライブラリ記述子の作成

以下の節では、タグ ライブラリ記述子（TLD）ファイルを作成する方法について説明します。

- タグ ライブラリ記述子の概要
- タグ ライブラリ記述子の記述
- タグ ライブラリ記述子のサンプル

タグ ライブラリ記述子の概要

タグ ライブラリを使用すると、開発者は関連する機能を持つタグを 1 つにまとめることができます。タグ ライブラリには、タグ拡張を記述し、それらを Java クラスに関連付けるタグ ライブラリ記述子（TLD）ファイルが必要となります。TLD は、WebLogic Server とオーサリング ツールが拡張機能に関する情報を取得するために使用されます。TLD ファイルは、XML 表記法で記述します。

タグ ライブラリ記述子の構文は文書型定義（DTD）に指定されており、http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd で入手できます。

タグ ライブラリ記述子の記述

タグ ライブラリ記述子ファイル内の要素は、この DTD に定義されている順序で並べます。この順序付けは、以下に示す手順で使用されます。TLD 要素がこの順序どおりに並べられていない場合、XML パーサは例外を送出します。

TLD の本体は、<taglib> ... </taglib> 要素の中にネストされた要素を含んでいます。これらのネストされた要素についても次の手順で説明します。このドキュメントでは、見やすくするためにネストされた要素はその親要素からインデントされていますが、TLD ではインデント処理は必要ありません。

3-7 ページの「タグ ライブラリ記述子のサンプル」の例では、code という新しいタグを宣言します。このタグの機能は、Java クラスの weblogic.taglib.quote.CodeTag によって実装されます。

タグ ライブラリ記述子を作成するには、次の手順に従います。

1. テキスト ファイルを作成して、適切な名前と拡張子 .tld を付け、JSP を持つ Web アプリケーションの WEB-INF ディレクトリに置きます。WEB-INF ディレクトリの内容は非公開で、WebLogic Server によって HTTP を介して提供されません。
2. 次のヘッダを追加します。

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.
 //DTD JSP Tag Library 1.1//EN" "web-jsptaglib_1_1.dtd">
```

3. TLD のコンテンツを、<taglib> 要素の中に埋めこみます。コンテンツには、タグ ライブラリに関する情報を含む要素と、各タグを定義する要素が含まれます。次に例を示します。

```
<taglib>
  ... taglib 記述子の本体 ...
</taglib>
```

4. タグ ライブラリを指定します。

   `<tlib-version>version_number</tlib-version>`
   （必須）タグ ライブラリのバージョン番号。

   `<jsp-version>version_number</jsp-version>`
   （必須）このタグ ライブラリが機能するために必要な JSP 仕様のバージョン（番号）を記述します。デフォルトは 1.2 です。

   `<shortname>TagLibraryName</shortname>`
   （必須）タグ ライブラリに短縮名を割り当てます。この要素は、WebLogic Server では使用されません。

   `<uri>unique_string</uri>`
   （必須）このバージョンのタグ ライブラリをユニークに識別するパブリックな URI を定義します。

`<displayname>display_name</displayname>`

（必須）ツールによって表示される短縮名を指定します。

`<smallicon>icon.jpg</smallicon>`

（省略可能）小さいアイコン（16x16）の画像が含まれるファイルの名前を指定します。ファイル名はタグ ライブラリ内の相対パスです。画像は JPEG または GIF 形式で、ファイル名はそれぞれ「.jpg」または「.gif」というサフィックスで終わる必要があります。アイコンはツールによって使用されます。

`<largeicon>icon.jpg</uri>`

（省略可能）大きいアイコン（32x32）の画像が含まれるファイルの名前を指定します。ファイル名はタグ ライブラリ内の相対パスです。画像は JPEG または GIF 形式で、ファイル名はそれぞれ「.jpg」または「.gif」というサフィックスで終わる必要があります。アイコンはツールによって使用されます。

`<description>...text...</description>`

（必須）タグ ライブラリについて説明する任意のテキスト文字列を定義します。

`<validator>unique_string</validator>`

（省略可能）このタグでスクリプト変数の情報を定義します。1 つまたは複数の変数のサブ要素を持つべきタグに null 以外のオブジェクトを返させるための変換時エラーを提供する TagExtraInfo クラスです。

`<listener>unique_string</listener>`

（省略可能）自動的にインスタンス化および登録される任意のイベント リスナ オブジェクトを定義します。

5. タグ ライブラリ バリデータを定義します（省略可能）。

`<validator>`

バリデータの最上位要素。

`<validator-class>my.validator</validator-class>`

（必須）検証を実行する Java クラス。

`<init-param>`

（省略可能）バリデータ クラスの初期化パラメータを定義します。

<param-name>param</param-name>
このパラメータの名前を定義します。

<param-value>value</param-value>
このパラメータの値を定義します。

6. タグを定義します。

タグ ライブラリに新しいタグを定義するには、それぞれ別個の <tag> 要素を使用します。<tag> 要素には、以下のネスト タグを埋め込むことができます。

<name>tag_name</name>
（必須）タグの名前を指定します。JSP ファイルのタグを参照するときに、次のように「:」という記号の後に使用します。

<mytaglib:tag_name>

詳細については、1-2 ページの「JSP でのカスタム タグの使い方」を参照してください。

<tagclass>package.class.name</tagclass>
（必須）このタグの機能を実装するタグ ハンドラ クラスを宣言します。そのクラスの完全修飾パッケージ名を指定します。

クラス ファイルは、WEB-INF\classes ディレクトリの下に、パッケージ名を反映したディレクトリ構造に従って配置します。タグ ライブラリ jar ファイルにクラスをパッケージすることもできます。詳細については、5-2 ページの「JSP タグ ライブラリを JAR ファイルとしてデプロイする」を参照してください。

<teiclass>package.class.name</teiclass>
（省略可能）このタグによって導入されるスクリプト変数を記述する TagExtraInfo のサブクラスを宣言します。タグが新しいスクリプト変数を定義しない場合、この要素は使われません。そのクラスの完全修飾パッケージ名を指定します。このクラスに含まれるタグの属性を検証できます。

クラス ファイルは、Web アプリケーションの WEB-INF\classes ディレクトリの下に、パッケージ名を反映したディレクトリ構造に従って配置します。タグ ライブラリ jar ファイルにクラスをパッケージすることもできます。詳細については、5-2 ページの「JSP タグ ライブラリを JAR ファイルとしてデプロイする」を参照してください。

`<bodycontent>tagdependent | JSP | empty</bodycontent>`

（省略可能）タグ本体のコンテンツを定義します。

`empty` を指定すると、タグは JSP ページ内で空タグ フォーマットで使用されます。たとえば、`<taglib:tagname/>` というように使用されます。

JSP を指定すると、タグのコンテンツは JSP として解釈され、タグは本体付きフォーマットで使用しなければなりません。次に例を示します。

`<taglib:tagname>...</taglib:tagname>.`

`tagdependent` を指定すると、タグは本体のコンテンツを JSP 以外のもの（たとえば SQL 文）であると解釈します。

`<bodycontent>` 要素が定義されていない場合、デフォルト値は `JSP` となります。

`<attribute>`

（省略可能）JSP ページ内のタグ要素に指定できる属性の名前を定義します。次に例を示します。

```
<taglib:mytag myAttribute="myAttributeValue">
```

タグに指定できる属性を定義するには、それぞれ別個の `<attribute>` 要素を使用します。タグ属性を使用すると、JSP 作成者はタグの動作を変更できます。

```
<name>myAttribute</name>
```

`<required>true | false</required>`

（省略可能）この属性が JSP ページの中でオプションとして使用されるかどうかを定義します。

ここで定義しない場合、デフォルトは `false`、つまりその属性は省略可能となります。

`true` が指定され、その属性が JSP ページの中で使用されない場合、変換時エラーが発生します。

`<rtexprvalue>true | false</rtexprvalue>`

（省略可能）この属性の値としてスクリプトレット式を指定して、要求時にそれを動的に計算できるようにするかどうかを定義します。

この要素を指定しない場合、デフォルトとして `false` が使用されます。

```
</attribute>
```

7. スクリプト変数を定義します（省略可能）。

```
<tag>
```
要素内にスクリプト変数を定義できます。

```
<variable>
```
変数の宣言の最上位要素。

```
<name-given>someName</name-given>
```
変数の名前を定義します。使用する属性から名前を定義できます。

```
<name-from-attribute>attrName</name-from-attribute>
```
attrName の値を使用して変数の名前を指定します。

```
<variable-class>some.java.type</variable-class>
```
この変数の Java タイプ。

```
<declare>true</declare>
```
（省略可能）true に設定する場合、変数が定義されることを示します。

```
<scope>AT_BEGIN</scope>
```
スクリプト変数のスコープ。有効なオプションは以下のとおりです。

`NESTED`（変数はタグ本体の内部でのみ使用できます）

`AT_BEGIN`（変数は本体を実行する直前に定義さます）

`AT_END`（変数は本体を実行した直後に定義さます）

```
</variable>
```

タグ ライブラリ記述子のサンプル

以下にタグ ライブラリ記述子のサンプルを示します。

コード リスト 3-1 タグ ライブラリ記述子（tld）のサンプル

```
<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.1</jsp-version>
  <shortname>quote</shortname>
  <description>
      このタグ ライブラリには、HTML のコンテンツをフォーマットするのに有効な
      タグ拡張がいくつか含まれています。
  </description>

  <tag>
    <name>code</name>
    <tagclass>weblogic.taglib.quote.CodeTag</tagclass>
    <bodycontent>tagdependent</bodycontent>
    <attribute>
      <name>fontAttributes</name>
    </attribute>
    <attribute>
      <name>commentColor</name>
    </attribute>
    <attribute>
      <name>quoteColor</name>
    </attribute>
  </tag>

</taglib>
```

4　タグ ハンドラの実装

以下の節では、拡張タグの機能を実装する Java クラスを記述する方法について説明します。

- タグ ハンドラの API
- タグ ハンドラのライフ サイクル
- タグ本体の反復処理
- タグ本体内の例外処理
- タグ属性の使い方
- 新しいスクリプト変数の定義
- 協調的ネスト タグの記述
- タグ ライブラリ バリデータの使用

タグ ハンドラの API

JSP 1.1 API には、カスタム タグ ハンドラの作成に使用するクラスとインタフェースのセットが定義されています。`javax.servlet.jsp.tagext` API のドキュメントについては、http://java.sun.com/j2ee/j2sdkee/techdocs/api/index.html を参照してください。

タグ ハンドラは、以下の 2 つのインタフェースのうちの 1 つを実装しなければなりません。

`Tag`

カスタム タグが本体のないタグ（空タグ）の場合、`javax.servlet.jsp.tagext.Tag` インタフェースを実装します。この API は、コンビニエンス クラス `TagSupport` も提供します。このクラス

は、Tag インタフェースを実装し、このインタフェース内で定義されているメソッド用のデフォルトの空メソッドを提供します。

BodyTag
: カスタム タグが本体を使用する必要がある場合、javax.servlet.jsp.tagext.BodyTag インタフェースを実装します。この API は、コンビニエンス クラス BodyTagSupport も提供します。このクラスは、BodyTag インタフェースを実装し、このインタフェース内で定義されているメソッド用のデフォルトの空メソッドを提供します。BodyTag は Tag を拡張したものであり、インタフェース メソッドのスーパーセットです。

タグ ハンドラのライフ サイクル

Tag インタフェースまたは BodyTag インタフェースのどちらかから継承され、そのタグ ハンドラ クラスによって実装されたメソッドは、JSP ページの処理中の特定の時点で JSP エンジンによって呼び出されます。これらのメソッドは、タグのライフ サイクルにおけるさまざまなポイントを表し、以下の順序で実行されます。

1. JSP エンジンが JSP ページの中でタグを見つけると、新しいタグ ハンドラが初期化されます。javax.servlet.jsp.tagext.Tag インタフェースの setPageContext() メソッドと setParent() メソッドが呼び出されて、そのタグ ハンドラの環境コンテキストが設定されます。タグ開発者は、基本クラス TagSupport または BodyTagSupport を拡張する場合、これらのメソッドを実装する必要はありません。
2. タグ属性ごとに setXXXX() という JavaBean のようなメソッドが呼び出されます。詳細については、4-6 ページの「タグ本体内の例外処理」を参照してください。
3. doStartTag() メソッドが呼び出されます。タグ ハンドラでこのメソッドを定義すると、タグ ハンドラを初期化したり、データベースなどの必要なあらゆるリソースへの接続を開くことができます。

   doStartTag() メソッドの終わりには、タグ本体を評価すべきかどうかを、以下の値定数のうちの 1 つをタグ ハンドラ クラスから返すことで指定できます。

SKIP_BODY

タグの本体をスキップするよう JSP エンジンに指示します。タグが本体のないタグ（空タグ）である場合、この値を返します。タグのライフ サイクルのうち本体に関係のある部分はスキップされ、次に呼びされるメソッドは doEndTag() となります。

EVAL_BODY_INCLUDE

タグ本体のコンテンツを評価して組み込むよう JSP エンジンに指示します。タグのライフ サイクルのうち本体に関係のある部分はスキップされ、次に呼びされるメソッドは doEndTag() となります。

この値を返すことができるのは、Tag インタフェースを実装するタグの場合だけです。この値を使用すると、本体を組み込むかどうかは決定できるが、そのコンテンツには関知しないタグを記述できます。タグが BodyTag インタフェースを実装する（または BodyTagSuport クラスを拡張する）場合、この値を返すことはできません。

EVAL_BODY_TAG

タグ本体を評価してから doInitBody() メソッドを呼び出すよう JSP エンジンに指示します。この値は、タグが BodyTag インタフェースを実装する（または BodyTagSuport クラスを拡張する）場合にだけ返すことができます。

4. setBodyContent() メソッドが呼び出されます。この時点では、タグからの出力は BodyContent という特殊な JspWriter にリダイレクトされ、クライアントには送られません。本体を評価して得られるすべてのコンテンツは BodyContent バッファに追加されます。このメソッドにより、タグ ハンドラは BodyContent バッファへの参照を格納して doAfterBody() メソッドで評価後処理のために使えるようにすることができます。

   タグが出力を JSP ページ（ネスト タグの場合はその親スコープ）に渡す場合、タグはタグ ライフ サイクルのこの時点から doEndTag() メソッドが終わるまでに、そのタグの出力を親スコープの JspWriter に明示的に書き出さなければなりません。タグ ハンドラは、getEnclosingWriter() メソッドを使用して、親スコープの出力にアクセスできます。

   コンビニエンス クラスの BodyTagSupport を使用している場合、このメソッドを実装する必要はありません。これは、タグが BodyContent への参

照を保持し、getBodyContent() メソッドを介してその参照を使用できるようにするからです。

5. doInitBody() メソッドが呼び出されます。このメソッドを使用すると、タグ本体が初めて評価される直前に何らかの処理を実行できます。ここでは、スクリプト変数を設定したり、タグ本体の前の BodyContent に何らかのコンテンツを挿入したりできます。ここで付加したコンテンツは、JSP ページのタグ本体のコンテンツとは異なり、JSP として評価されることはありません。

   このメソッドで実行する処理と doStartTag() メソッドの終わりに実行する処理との大きな違いは（EVAL_BODY_TAG を返そうとしていることが分かった場合）、このメソッドでは、タグの出力のスコープはネストされていて、JSP ページ（または親タグ）には直接向けられないということです。すべての出力は、BodyContent という特殊な JspWriter に書き込まれます。

6. doAfterBody() メソッドが呼び出されます。このメソッドは、タグの本体が評価され BodyContent バッファに追加された後に呼び出されます。タグ ハンドラは、評価済みタグ本体に基づいて何らかの処理を行うためにこのメソッドを実装する必要があります。ハンドラがコンビニエンス クラスの BodyTagSupport を拡張する場合、getBodyContent() メソッドを使用して評価済みタグ本体にアクセスできます。単に BodyTag インタフェースを実装するだけであれば、setBodyContent() メソッドを定義して、そこに BodyContent インスタンスへの参照を格納しておく必要があります。

   doAfterBody() メソッドの終わりには、タグのライフ サイクルを、前と同じように以下の値定数のうちの 1 つを返すことで決定できます。

   SKIP_BODY

   処理を続行し、本体を再び評価しないよう JSP エンジンに指示します。タグのライフ サイクルは doEndTag() メソッドに進みます。

   EVAL_BODY_TAG

   本体を再び評価するよう JSP エンジンに指示します。評価済み本体が BodyContent に追加され、doAfterBody() メソッドが再び呼び出されます。

   この時点で、タグ ハンドラが親スコープに出力を書き出すよう指定できます。親スコープへのライタを取得するには、BodyTagSupport.getPreviousOut() メソッドまたは

BodyContent.getEnclosingWriter() メソッドを使用します。どちらのメソッドでも、同じ親ライタが取得されます。

タグ ハンドラは評価済み本体のコンテンツを親スコープに書き出すこともあれば、その評価済み本体をさらに処理して他の何らかの出力を書き出すこともあります。本体の反復処理ごとに BodyContent が既存の BodyContent に追加されるので、SKIP_BODY を返すよう決定した場合、反復処理された本体のコンテンツ全体だけを書き出します。そのようにしないと、後続の各反復処理のコンテンツが何度も出力に現れることになります。

7. pageContext 内の out ライタが親の JspWriter に復元されます。このオブジェクトは実際にはスタックであり、pushBody() メソッドと popBody() メソッドを使用して pageContext 上の JSP エンジンによって処理されます。ただし、タグ ハンドラの中でこれらのメソッドを使用してこのスタックを処理しようとしてはいけません。

8. doEndTag() メソッドが呼び出されます。タグ ハンドラはこのメソッドを実装して、タグ処理後のサーバサイド作業を実行し、出力を親スコープ JspWriter に書き出し、データベース接続などのリソースをクローズできます。

   タグ ハンドラは、doEndTag() メソッド内で pageContext.getOut() を実行して得られる JspWriter を使用して、親スコープに出力を直接書き出します。なお、pageContext.out は、前の手順で popBody() が pageContext 上で呼び出されたときに親ライタに復元されています。

   doEngTag() メソッドから以下の値のうちの 1 つを返すことで、JSP ページの残りの部分の評価フローを制御できます。

   EVAL_PAGE

   JSP ページの残りの部分の処理を続行するよう JSP エンジンに指示します。

   SKIP_PAGE

   JSP ページの残りの部分をスキップするよう JSP エンジンに指示します。

9. release() メソッドが呼び出されます。この呼び出しは、タグ ハンドラ インスタンスが逆参照され、ガベージ コレクション用に使用できるようになる直前に行われます。

タグ本体の反復処理

javax.servlet.jsp.tagext.IterationTag インタフェースを実装するタグでは、タグの本体を条件付きで再評価できる doAfterBody() というメソッドを使用できます。doAfterBody() が IterationTag.EVAL_BODY_AGAIN を返す場合は、本体が再評価されます。doAfterBody() が Tag.SKIP_BODY を返す場合は、本体はスキップされて doEndTag() メソッドが呼び出されます。詳細については、このインタフェースの J2EE Javadoc を参照してください（Sun Microsystems の Javadoc は、http://java.sun.com/products/jsp/download.html でダウンロードできます）。

注意： IterationTag インタフェースは、Sun Microsystems の JSP 1.2 仕様の新機能です。バージョン 1.2 は仕様の最終草案として提案されたもので、変更されることがあります。アプリケーションで JSP 1.2 の機能を使用する場合は、この仕様が最終的なものではなく、将来変更される可能性があることに注意してください。

タグ本体内の例外処理

javax.servlet.jsp.tagext.TryCatchFinally インタフェースの doCatch() および doFinally() メソッドを実装すると、タグ内から送出される例外を捕捉できます。詳細については、このインタフェースの J2EE Javadoc を参照してください。（Sun Microsystems の Javadoc は、http://java.sun.com/products/jsp/download.html でダウンロードできます）。

注意： TryCatchFinally インタフェースは、Sun Microsystems の JSP 1.2 仕様の新機能です。バージョン 1.2 は仕様の最終草案として提案されたもので、変更されることがあります。アプリケーションで JSP 1.2 の機能を使用する場合は、この仕様が最終的なものではなく、将来変更される可能性があることに注意してください。

タグ属性の使い方

カスタム タグでは、JSP ページから指定できる属性を何個でも定義できます。これらの属性は、タグ ハンドラに情報を渡してその動作をカスタマイズするために使用できます。

各属性名は、TLD の中で <attribute> 要素を使用して宣言します。この要素は、属性の名前とその他の属性プロパティを宣言します。

JavaBean 規約と同じように、タグ ハンドラは属性名に基づいてセッター メソッドとゲッター メソッドを実装しなければなりません。たとえば、foo という属性を宣言する場合、タグ ハンドラは以下のパブリック メソッドを定義しなければなりません。

```
public void setFoo(String f);
public String getFoo();
```

属性名の先頭の文字は、プレフィックスの set または get の後では大文字になることに注意してください。

JSP エンジンは、タグ ハンドラが初期化されてから doStartTag() メソッドが呼び出されるまでの間に、各属性のセッター メソッドを呼び出します。一般に、タグ ハンドラの他のメソッドからアクセスできるメンバー変数に属性値を格納するには、セッター メソッドを実装しなければなりません。

新しいスクリプト変数の定義

タグ ハンドラは、さまざまなスコープで JSP ページから参照できる新しいスクリプト変数を使用できます。スクリプト変数は、それらの定義済みスコープの内部で暗黙的なオブジェクトのように使用できます。

javax.servlet.jsp.tagext.TagExtraInfo を拡張する Java クラスを識別するための新しいスクリプト変数は、<teiclass> 要素を使用して定義します。次にその例を示します。

```
<teiclass>weblogic.taglib.session.ListTagExtraInfo</teiclass>
```

次に、ExtraTagInfo クラスを作成します。次に例を示します。

```
package weblogic.taglib.session;
import javax.servlet.jsp.tagext.*;

public class ListTagExtraInfo extends TagExtraInfo {

  public VariableInfo[] getVariableInfo(TagData data) {
    return new VariableInfo[] {
           new VariableInfo("username",
                            "String",
                            true,
                            VariableInfo.NESTED),
           new VariableInfo("dob",
                            "java.util.Date",
                            true,
                            VariableInfo.NESTED)
         };
  }
}
```

上の例では、VariableInfo 要素の配列を返す getVariableInfo() という 1 つのメソッドが定義されています。各要素は、新しいスクリプト変数を定義します。上記のサンプルでは、java.lang.String 型の username と java.util.Date 型の dob という 2 つのスクリプト変数が定義されています。

VariableInfo() のコンストラクタは、以下の 4 つの引数を取ります。

- 新しい変数の名前を定義する String
- その変数の Java データ型を定義する String。java.lang パッケージ以外のパッケージに含まれる型の完全パッケージ名を指定します。
- その変数を使う前にインスタンス化しなければならないかどうかを宣言する boolean。タグ ハンドラが Java 以外の言語で書かれている場合を除き、この変数は「true」に設定します。
- その変数のスコープを宣言する int。以下に示す VariableInfo に定義されている静的フィールドを使用します。

  VariableInfo.NESTED
  : タグの開始タグと終了タグの間でだけ使用できます。

  VariableInfo.AT_BEGIN
  : 開始タグとページの最後の間で使用できます。

  VariableInfo.AT_END
  : 終了タグとページの最後の間で使用できます。

タグ ハンドラは、ページのコンテキストを介してスクリプト変数の値を初期化しなければなりません。たとえば、上で定義したスクリプト変数の値を初期化するには、doStartTag() メソッドの中で以下の Java ソースを使用します。

```
pageContext.setAttribute("name", nameStr);
pageContext.setAttribute("dob", bday);
```

ここで、最初のパラメータはスクリプト変数の名前を指定し、2 番目のパラメータは代入される値を示します。なお、ここでは Java 変数の nameStr は String 型で、bday は java.util.Date 型です。

また、TagExtraInfo クラスを使用して作成された変数にアクセスするには、useBean を使用して作成された JavaBean にアクセスするときと同じようにそれを参照します。

動的名前付きスクリプト変数

タグの属性から、新しいスクリプト変数の名前を定義できます。このように定義することで、1 つのスクリプト変数を定義するタグの複数のインスタンスを同じスコープで使用しつつ、タグのスクリプト変数名の衝突を避けることができます。TagExtraInfo を拡張するクラスからこれを実行するには、getVariableInfo() メソッドに渡される TagData からスクリプト変数の名前を取得しなければなりません。

TagData からは、getAttributeString() メソッドを使用して、スクリプト変数の名前を指定する属性の値を検索できます。さらに、id 属性の値を返す getId() メソッドも存在します。これは、JSP タグからもたらされる新しい暗黙的オブジェクトに名前を付けるのによく使用されます。

タグ ライブラリ記述子の変数の定義

タグ ライブラリ記述子の変数を定義できます。詳細については、3-6 ページの手順 7. を参照してください。

協調的ネスト タグの記述

ネストされているタグが、その親タグに定義されているプロパティを暗黙的に使用するよう設計できます。たとえば、コード例の「SQL Query」（WebLogic Server の samples\examples\jsp\tagext\sql ディレクトリを参照）では、<sql:query> タグが <sql:connection> タグの内部にネストされています。query タグは親スコープの connection タグを検索して、そのタグによって確立された JDBC 接続を使用します。

親スコープのタグを見つけるために、ネスト タグは TagSupport クラスの 静的メソッドである findAncestorWithClass() を使用します。次に、QueryTag のサンプルから抜粋したコードを示します。

```
try {
  ConnectionTag connTag = (ConnectionTag)
    findAncestorWithClass(this,
      Class.forName("weblogic.taglib.sql.ConnectionTag"));
  } catch(ClassNotFoundException cnfe) {
    throw new JspException("Query tag connection "+
                           "attribute not nested "+
                           "within connection tag");
}
```

この例では、与えられたクラスに一致するタグ ハンドラ クラスを持つ最も近い親タグ クラスが返されます。直系の親タグがこのタイプでなければ、さらにその親が調べられます。一致するタグが見つかるまでこの処理が繰り返され、それでも見つからない場合は ClassNotFoundException が送出されます。

カスタム タグでこの機能を使用すれば、JSP ページでのタグの構文と使い方を簡素化できます。

タグ ライブラリ バリデータの使用

注意： Java タグ ライブラリ バリデータは、Sun Microsystems の JSP 1.2 仕様の新機能です。バージョン 1.2 は仕様の最終草案として提案されたもので、変更されることがあります。アプリケーションで JSP 1.2 の機能を使用する場合は、この仕様が最終的なものではなく、将来変更される可能性があることに注意してください。

タグ ライブラリ バリデータはユーザが記述する Java クラスです。これを使用すると JSP ページ上でカスタム検証を実行できます。バリデータ クラスは入力ストリームとして JSP ページ全体を取得します。バリデータ クラスに記述した条件に基づいてページを検証できます。バリデータの一般的な使い方としては、バリデータ クラス内で XML パーサを使用して、文書型定義（DTD）に対してページを検証する場合が挙げられます。バリデータ クラスはページの変換時（JSP がサーブレットに変換されるとき）に呼び出され、ページが検証されると `null` 文字列を返します。または、検証が失敗するとエラー情報を含む文字列を返します。

タグ ライブラリ バリデータを実装するには、次の手順に従います。

1. バリデータ クラスを記述します。バリデータ クラスは `javax.servlet.jsp.tagext.TagLibraryValidator` クラスを拡張します。
2. タグ ライブラリ記述子内のバリデータを参照します。次に例を示します。

```
<validator>
  <validator-class>
     myapp.tools.MyValidator
  </validator-class>
</validator>
```

3. （省略可能）初期化パラメータを定義します。バリデータ クラスで初期化パラメータを取得および使用できます。次に例を示します。

```
<validator>
  <validator-class>
     myapp.tools.MyValidator
  </validator-class>
  <init-param>
     <param-name>myInitParam</param-name>
     <param-value>foo</param-value>
  </init-param>
</validator>
```

4. Web アプリケーションの `WEB-INF\classes` ディレクトリに、バリデータ クラスをパッケージ化します。タグ ライブラリ jar ファイルにクラスをパッケージ化することもできます。詳細については、5-2 ページの「JSP タグ ライブラリを JAR ファイルとしてデプロイする」を参照してください。

5　管理とコンフィグレーション

以下の節では、JSP タグ拡張を使用するための管理タスクとコンフィグレーション タスクの概要を示します。

- JSP タグ ライブラリのコンフィグレーション
- JSP タグ ライブラリを JAR ファイルとしてデプロイする

JSP タグ ライブラリのコンフィグレーション

ここでは、JSP タグ ライブラリをコンフィグレーションおよびデプロイする手順について説明します。タグ ライブラリは、jar ファイルとしてデプロイできます（5-2 ページの「JSP タグ ライブラリを JAR ファイルとしてデプロイする」を参照）。

1. タグ ライブラリ記述子（TLD）を作成します。

   詳細については、3-1 ページの「タグ ライブラリ記述子の作成」を参照してください。

2. この TLD を Web アプリケーション デプロイメント記述子の web.xml で参照します。次に例を示します。

```
<taglib>
  <taglib-uri>myTLD</taglib-uri>
  <taglib-location>WEB-INF/library.tld</taglib-location>
</taglib>
```

この例では、タグ ライブラリ記述子は `library.tld` というファイルです。Web アプリケーションのルートに対して相対的な `tld` の場所を常に指定します。

Web アプリケーション デプロイメント記述子の編集の詳細については、「Taglib 要素」を参照してください。

3. タグ ライブラリ記述子ファイルを、Web アプリケーションの `WEB-INF` ディレクトリに配置します。
4. JSP ページでタグ ライブラリを参照します。

   JSP で、JSP ディレクティブを使用してタグ ライブラリを参照します。次に例を示します。

   ```
   <%@ taglib uri="myTLD" prefix="mytaglib" %>
   ```

   WebLogic JSP の詳細については、『WebLogic JSP プログラマーズ ガイド』を参照してください。
5. タグのタグ ハンドラ Java クラス ファイルを、Web アプリケーションの `WEB-INF\classes` ディレクトリに配置します。
6. Web アプリケーションを WebLogic Server にデプロイします。詳細については、「Web アプリケーションのデプロイメント」を参照してください。

JSP タグ ライブラリを JAR ファイルとしてデプロイする

上に示した手順に加えて、JSP タグ ライブラリを `jar` ファイルとしてデプロイすることもできます。

1. TLD（タグ ライブラリ記述子）ファイルを `taglib.tld` という名前で作成します。

   詳細については、3-1 ページの「タグ ライブラリ記述子の作成」を参照してください。

2. タグ ライブラリで使用されるコンパイル済み Java タグ ハンドラ クラスのファイルを格納するディレクトリを作成します。
3. このディレクトリのサブディレクトリを、`META-INF` という名前で作成します。
4. 手順 1. で作成した `taglib.tld` ファイルを、手順 3. で作成した `META-INF` ディレクトリにコピーします。
5. 手順 2. で作成したディレクトリから次のコマンドを実行して、コンパイルされた Java クラス ファイルを `jar` ファイルにアーカイブします。

   ```
   jar cv0f myTagLibrary.jar
   ```

   （`myTagLibrary.jar` はユーザが指定する名前です）
6. `jar` ファイルを、タグ ライブラリを使用する Web アプリケーションの `WEB-INF\lib` ディレクトリにコピーします。
7. このタグ ライブラリ記述子を Web アプリケーション デプロイメント記述子の `web.xml` で参照します。次に例を示します。

   ```
   <taglib>
     <taglib-uri>myjar.tld</taglib-uri>
     <taglib-location>
        /WEB-INF/lib/myTagLibrary.jar
     </taglib-location>
   </taglib>
   ```

   詳細については、「Web アプリケーションのデプロイメント記述子の記述」を参照してください。
8. JSP でタグ ライブラリを参照します。次に例を示します。

   ```
   <%@ taglib uri="myjar.tld" prefix="wl" %>
   ```

索引