

## **Oracle® Application Server Containers for J2EE**

JavaServer Pages 開発者ガイド

10g (9.0.4)

**部品番号 : B12331-01**

2004 年 1 月

Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド, 10g (9.0.4)

部品番号 : B12331-01

原本名 : Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide, 10g (9.0.4)

原本部品番号 : B10320-01

原本著者 : Brian Wright

原本協力者 : Michael Freedman, Julie Basu, Alex Yiu, Sunil Kunisetty, Gael Stevens, Sumathi Gopalakrishnan, Ping Guo, Olga Peschansky, John O'Duinn, YaQing Wang, Song Lin, Helen Zhao, Hal Hildebrand, Jasen Minton, Ashok Banerjee, Matthieu Devin, Jose Alberto Fernandez, Jerry Schwarz, Clement Lai, Shinji Yoshida, Kenneth Tang, Robert Pang, Kannan Muthukkaruppan, Ralph Gordon, Shiva Prasad, Sharon Malek, Jeremy Litz, Kuassi Mensah, Susan Kraft, Sheryl Maring, Ellen Barnes, Angie Long, Sanjay Singh

Copyright © 2000, 2003 Oracle Corporation. All rights reserved.

#### 制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社にある権利が記されています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

#### U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation, and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかるとして使用する場合、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性 (redundancy)、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle は Oracle Corporation およびその関連会社の登録商標です。その他の名称は、Oracle Corporation または各社が所有する商標または登録商標です。

---

---

# 目次

|   |      |
|---|------|
| はじめに .....  | ix   |
| 対象読者 .....  | x    |
| このマニュアルの構成 .....                                  | xi   |
| 関連ドキュメント .....                                    | xii  |
| 表記規則 .....  | xv   |
| <br>  |      |
| <b>1 JSP 全体の概要</b>                                |      |
| <b>JavaServer Pages の概要</b> .....                 | 1-2  |
| JSP ページの例 .....                                   | 1-2  |
| JSP コーディングとサーブレット・コーディングの使いやすさの比較 .....           | 1-4  |
| ビジネス・ロジックとページ・プレゼンテーションの分離 : JavaBeans のコール ..... | 1-5  |
| JSP ページと代替マークアップ言語 .....                          | 1-6  |
| <b>JSP 構文の要素の概要</b> .....                         | 1-6  |
| ディレクティブ .....                                     | 1-7  |
| スクリプト要素 .....                                     | 1-9  |
| JSP オブジェクトとスコープ .....                             | 1-11 |
| 標準アクション : JSP タグ .....                            | 1-15 |
| 文字列値から Bean プロパティへの変換 .....                       | 1-21 |
| カスタム・タグ・ライブラリ .....                               | 1-23 |
| <b>JSP の実行</b> .....                              | 1-24 |
| JSP コンテナの概要 .....                                 | 1-24 |
| JSP の実行モデル .....                                  | 1-25 |
| JSP ページとオンデマンド変換 .....                            | 1-26 |
| JSP ページのリクエスト .....                               | 1-27 |

## 2 Oracle JSP 実装の概要

|  |      |
|--|------|
| Oracle Application Server と JSP サポートの概要 .....  | 2-2  |
| Oracle Application Server の概要 .....            | 2-2  |
| Oracle HTTP Server と mod_oc4j .....            | 2-3  |
| OC4J の概要 .....                                 | 2-4  |
| OC4J の JSP 実装の概要 .....                         | 2-8  |
| Oracle JDeveloper の JSP サポート .....             | 2-12 |
| Oracle の付加価値機能の概要 .....                        | 2-13 |
| OC4J が提供するタグ・ライブラリとユーティリティの概要 .....            | 2-13 |
| Oracle 固有の機能の概要 .....                          | 2-14 |
| キャッシング・サポートに関するタグと API の概要 .....               | 2-16 |
| JavaServer Pages 標準タグ・ライブラリ (JSTL) のサポート ..... | 2-16 |

## 3 スタート・ガイド

|  |      |
|--|------|
| 初期段階での考慮事項 .....                                 | 3-2  |
| アプリケーション・ルート機能 .....                             | 3-2  |
| クラスパスの機能 .....                                   | 3-4  |
| 実行時の再変換または再ロード .....                             | 3-5  |
| JSP のコンパイルに関する考慮事項 .....                         | 3-6  |
| JSP のセキュリティに関する考慮事項 .....                        | 3-7  |
| JSP のパフォーマンスに関する考慮事項 .....                       | 3-7  |
| デフォルト・パッケージのインポート .....                          | 3-9  |
| JSP ファイルのネーミング規則 .....                           | 3-10 |
| JDK 1.4 に関する考慮事項: パッケージに含まれないクラスを起動できない .....    | 3-11 |
| OC4J が提供する主なサポート・ファイル .....                      | 3-12 |
| OC4J での JSP 構成 .....                             | 3-13 |
| JSP コンテナの設定 .....                                | 3-13 |
| JSP 構成パラメータ .....                                | 3-14 |
| JSP の OC4J 構成パラメータ .....                         | 3-28 |
| 主な OC4J 構成ファイル .....                             | 3-31 |
| Oracle Enterprise Manager での JSP の構成 .....       | 3-32 |
| Application Server Control の「JSP プロパティ」ページ ..... | 3-33 |
| 「JSP プロパティ」ページでサポートされる構成パラメータ .....              | 3-34 |
| 「JSP プロパティ」ページでサポートされていない構成パラメータ .....           | 3-35 |

## 4 プログラミングに関する基本的な考慮事項

|   |      |
|---|------|
| JSP とサーブレット間の相互作用 .....                           | 4-2  |
| JSP ページからのサーブレットの起動 .....                         | 4-2  |
| JSP ページから起動したサーブレットへのデータの受渡し .....                | 4-3  |
| サーブレットからの JSP ページの起動 .....                        | 4-3  |
| JSP ページとサーブレット間でのデータの受渡し .....                    | 4-4  |
| JSP とサーブレット間の相互作用のサンプル .....                      | 4-5  |
| JSP データ・アクセスに関するサポートと機能 .....                     | 4-6  |
| データ・アクセスに対する JSP サポートの概要 .....                    | 4-6  |
| JDBC を使用した JSP データ・アクセスのサンプル .....                | 4-7  |
| JDBC パフォーマンス強化機能の使用 .....                         | 4-10 |
| JSP ページからの EJB コール .....                          | 4-13 |
| Oracle SQLJ の JSP サポート .....                      | 4-14 |
| OracleXMLQuery クラス .....                          | 4-18 |
| JSP のリソース管理 .....                                 | 4-18 |
| 標準セッションのリソース管理 : HttpSessionBindingListener ..... | 4-18 |
| リソース管理のための Oracle の付加価値機能の概要 .....                | 4-23 |
| 実行時エラーの処理 .....                                   | 4-24 |
| サーブレットと JSP の実行時エラー処理機能 .....                     | 4-24 |
| JSP エラー・ページの例 .....                               | 4-25 |

## 5 JSP XML サポート

|  |      |
|--|------|
| JSP XML 文書と JSP XML ビュー : 概要と比較 .....      | 5-2  |
| JSP XML 文書の詳細 .....                        | 5-4  |
| JSP XML 構文のサマリー表 .....                     | 5-5  |
| JSP XML ルート要素と JSP XML 名前空間 .....          | 5-7  |
| JSP XML ディレクティブ要素 .....                    | 5-8  |
| JSP XML の宣言要素、式要素およびスクリプトレット要素 .....       | 5-9  |
| JSP XML の標準アクションとカスタム・アクションの要素 .....       | 5-10 |
| JSP XML のテキスト要素とその他の要素 .....               | 5-10 |
| 比較サンプル : 従来の JSP ページと JSP XML 文書との比較 ..... | 5-11 |
| JSP XML ビューの詳細 .....                       | 5-14 |
| JSP ページから XML ビューへの変換 .....                | 5-14 |
| 妥当性チェックにおけるエラー・レポートの jsp:id 属性 .....       | 5-15 |
| 例 : 従来の JSP ページから XML ビューへの変換 .....        | 5-16 |

## 6 プログラミングに関するその他の考慮事項

|  |      |
|--|------|
| 一般的な JSP プログラミングの方針 .....                | 6-2  |
| JavaBeans とスクリプトレットの比較 .....             | 6-2  |
| 静的なインクルードと動的なインクルードの比較 .....             | 6-2  |
| JSP タグ・ライブラリの作成と使用を考慮する時期 .....          | 6-5  |
| JSP プログラミングのその他のヒント .....                | 6-6  |
| 直接起動から JSP ページを除外する方法 .....              | 6-6  |
| 集中チェッカ・ページの使用 .....                      | 6-7  |
| 大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処 ..... | 6-9  |
| メソッド変数宣言とメンバー変数宣言の比較 .....               | 6-10 |
| page ディレクティブの特性 .....                    | 6-12 |
| JSP での空白の保持とバイナリ・データの使用 .....            | 6-15 |

## 7 JSP の変換とデプロイ

|  |      |
|--|------|
| JSP トランスレータの機能 .....                     | 7-2  |
| 生成されるコードの機能 .....                        | 7-2  |
| 出力名に関する一般規則 .....                        | 7-3  |
| 生成されるパッケージとクラスの名前 .....                  | 7-4  |
| 生成されるファイルとその格納場所 .....                   | 7-6  |
| Oracle JSP のグローバル・インクルード .....           | 7-8  |
| ojspc 事前変換ユーティリティ .....                  | 7-12 |
| 基本的な ojspc 機能の概要 .....                   | 7-12 |
| ojspc バッチ事前変換の概要 .....                   | 7-14 |
| ojspc のオプションのサマリー表 .....                 | 7-16 |
| ojspc のコマンドライン構文 .....                   | 7-19 |
| ojspc のオプションの説明 .....                    | 7-20 |
| ojspc の出力ファイル、場所および関連オプションのサマリー .....    | 7-32 |
| JSP デプロイに関する考慮事項 .....                   | 7-34 |
| EAR/WAR デプロイの概要 .....                    | 7-34 |
| Oracle JDeveloper によるアプリケーションのデプロイ ..... | 7-36 |
| JSP の事前変換 .....                          | 7-37 |
| バイナリ・ファイルのみのデプロイ .....                   | 7-40 |

## 8 JSP タグ・ライブラリ

|   |      |
|---|------|
| タグ・ライブラリのフレームワークの概要 .....                           | 8-2  |
| カスタム・タグ・ライブラリの実装の概要 .....                           | 8-2  |
| JSP 仕様 1.1 と 1.2 の間のタグ・ライブラリ変更の概要 .....             | 8-4  |
| <b>タグ・ライブラリ・ディスクリプタ</b> .....                       | 8-7  |
| TLD の妥当性チェックおよび機能の概要 .....                          | 8-7  |
| tag 要素の使用 .....                                     | 8-9  |
| その他の主な要素とそのサブ要素 : validator および listener .....      | 8-13 |
| <b>タグ・ライブラリと TLD の設定およびアクセス</b> .....               | 8-13 |
| 概要 : taglib ディレクティブによるタグ・ライブラリの指定 .....             | 8-13 |
| 物理的な場所によるタグ・ライブラリの指定 .....                          | 8-15 |
| 単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス .....     | 8-16 |
| web.xml のタグ・ライブラリへの使用 .....                         | 8-18 |
| タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング ..... | 8-19 |
| 例 : 単一の JAR ファイルに複数のタグ・ライブラリと TLD がある場合 .....       | 8-23 |
| <b>タグ・ハンドラ</b> .....                                | 8-26 |
| タグ・ハンドラの概要 .....                                    | 8-26 |
| 属性の処理および文字列の変換 .....                                | 8-27 |
| カスタム・タグの処理 (タグ・ボディを使用する場合と使用しない場合) .....            | 8-28 |
| ボディ処理用の整定数の要約 .....                                 | 8-30 |
| 反復のない単純タグ・ハンドラ .....                                | 8-31 |
| 反復がある単純タグ・ハンドラ .....                                | 8-32 |
| ボディ・コンテンツにアクセスするタグ・ハンドラ .....                       | 8-34 |
| TryCatchFinally インタフェース .....                       | 8-36 |
| 外部タグ・ハンドラ・インスタンスへのアクセス .....                        | 8-37 |
| <b>OC4J の JSP タグ・ハンドラの機能</b> .....                  | 8-38 |
| 実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化 .....             | 8-38 |
| タグ・ハンドラのコード生成 .....                                 | 8-41 |
| <b>スクリプト変数、宣言およびタグ補足情報クラス</b> .....                 | 8-41 |
| スクリプト変数の使用 .....                                    | 8-41 |
| スクリプト変数のスコープ .....                                  | 8-42 |
| TLD の variable 要素を使用した変数宣言 .....                    | 8-43 |
| タグ補足情報クラスを使用した変数宣言 .....                            | 8-44 |

|  |      |
|--|------|
| 妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス .....           | 8-46 |
| TLD の validator 要素 .....                     | 8-47 |
| 主要な TLV 関連クラスおよび validation() メソッド .....     | 8-48 |
| TLV の処理 .....                                | 8-49 |
| 妥当性チェック機能 .....                              | 8-49 |
| タグ・ライブラリのイベント・リスナー .....                     | 8-50 |
| TLD の listener 要素 .....                      | 8-51 |
| タグ・ライブラリのイベント・リスナーのアクティブ化 .....              | 8-51 |
| イベント・リスナー情報を取得するための TLD へのアクセス .....         | 8-52 |
| カスタム・タグの完全な例 .....                           | 8-52 |
| 例: IterationTag インタフェースの使用 .....             | 8-52 |
| 例: IterationTag インタフェースおよびタグ補足情報クラスの使用 ..... | 8-56 |
| コンパイル時のタグ .....                              | 8-61 |
| コンパイル時と実行時の機能に関する考慮事項 .....                  | 8-61 |
| JML ライブラリの JSP 実行時バージョンとコンパイル時バージョン .....    | 8-61 |

## 9 JSP グローバリゼーション・サポート

|  |     |
|--|-----|
| コンテンツ・タイプの設定 .....                               | 9-2 |
| page ディレクティブでのコンテンツ・タイプの設定 .....                 | 9-2 |
| コンテンツ・タイプの動的な設定 .....                            | 9-5 |
| JSP ライター・オブジェクトのキャラクタ・セットに対する Oracle の拡張機能 ..... | 9-6 |
| マルチバイト・パラメータ・エンコードに対する JSP サポート .....            | 9-7 |
| 標準の setCharacterEncoding() メソッド .....            | 9-7 |
| 従来のサーブレット環境に対する Oracle の拡張機能の概要 .....            | 9-8 |

## A サーブレットと JSP の技術的なバックグラウンド

|                                       |     |
|---------------------------------------|-----|
| サーブレットに関するバックグラウンド .....              | A-2 |
| サーブレット・テクノロジーの概要 .....                | A-2 |
| サーブレットのインタフェース .....                  | A-3 |
| サーブレット・コンテナ .....                     | A-4 |
| サーブレット・セッション .....                    | A-5 |
| サーブレット・コンテキスト .....                   | A-7 |
| イベント・リスナーによるアプリケーションのライフ・サイクル管理 ..... | A-8 |
| サーブレットの起動 .....                       | A-8 |



|                            |      |
|----------------------------|------|
| Web アプリケーション階層 .....       | A-9  |
| 標準の JSP インタフェースとメソッド ..... | A-11 |

## B Apache JServ 環境

|  |      |
|--|------|
| JServ 環境での作業開始 .....                           | B-2  |
| Apache JServ Web Server のクラスパスへのファイルの追加 .....  | B-2  |
| JServ の JSP ファイル名拡張子のマッピング .....               | B-3  |
| JServ の JSP 構成パラメータ .....                      | B-4  |
| JServ での JSP パラメータの設定 .....                    | B-15 |
| JServ での ojspc の使用 .....                       | B-16 |
| JServ 環境に関する考慮事項 .....                         | B-17 |
| mod_jserv Apache Mod .....                     | B-17 |
| JServ でのアプリケーション・ルート・サポートに関する JSP コンテナ機能 ..... | B-18 |
| JServ でのアプリケーションとセッションのフレームワークの概要 .....        | B-18 |
| JServ における JSP とサーブレット間でのセッション共有 .....         | B-19 |
| JServ での動的なインクルードと転送 .....                     | B-19 |
| JServ ディレクトリの別名変換 .....                        | B-22 |
| JServ での JSP のセキュリティに関する考慮事項 .....             | B-24 |
| JServ でのマルチバイト・パラメータ・エンコード .....               | B-24 |
| JServ に対する JSP アプリケーションとセッションのサポート .....       | B-32 |
| globals.jsa 機能の概要 .....                        | B-32 |
| globals.jsa の構文とセマンティクスの概要 .....               | B-34 |
| globals.jsa のイベント・ハンドラ .....                   | B-36 |
| グローバルな宣言とディレクティブ .....                         | B-41 |
| globals.jsa からの移行 .....                        | B-43 |

## C サード・パーティ・ライセンス

|                                   |     |
|-----------------------------------|-----|
| Apache HTTP Server .....          | C-2 |
| The Apache Software License ..... | C-2 |
| Apache JServ .....                | C-3 |
| Apache JServ Public License ..... | C-3 |

## 索引



---

---

# はじめに

このマニュアルでは、Sun 社が先導する業界グループが提供する、JavaServer Pages (JSP) テクノロジーの Oracle 実装について説明します。標準機能の概要を説明しますが、主として Oracle 実装の詳細と付加価値機能を中心に説明します。

JavaServer Pages テクノロジーは、標準の 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) です。

Oracle Application Server 10g (9.0.4) における OC4J の JSP コンテナは、Sun 社の JSP 仕様 1.2 の完全な実装です。

この章には、次の各項が含まれます。

- [対象読者](#)
- [このマニュアルの構成](#)
- [関連ドキュメント](#)
- [表記規則](#)

## 対象読者

このマニュアルは、JavaServer Pages テクノロジに基づく Web アプリケーション作成に関心がある開発者を対象としています。Web およびサーブレット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
- 一般的なサーブレット・テクノロジ（技術的な説明は、付録 A にあります。）
- Web サーバーおよびサーブレット環境の構成方法
- HTML
- Java
- Oracle JDBC（Oracle Database にアクセスする JSP アプリケーション用）

標準の JSP のテクノロジと構文については、第 1 章で概要を説明していますが、詳細な説明は含まれていません。標準の JSP 機能の追加情報は、Sun 社の JSP 仕様などの参考資料を参照してください。

JSP 仕様 1.2 はサーブレット 2.3 環境に依存しているため、このマニュアルでは、この環境を中心に説明します（JSP 1.1 に対する下位互換性についても説明しています）。OC4J JSP コンテナは、以前のサーブレット環境用の特別な機能を備えています。これらの機能については、サーブレット 2.0 環境（特に、Oracle Application Server に搭載されている Apache JServ）に関連しているため、付録 B で特に説明しています。

OC4J 製品に搭載されているタグ・ライブラリおよびユーティリティについては、『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』を参照してください。

OC4J の JSP ページで作業を開始する際の入門的な情報は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

# このマニュアルの構成

このマニュアルは次の章で構成されています。

## 第 1 章「JSP 全体の概要」

この章では、標準の JSP 1.2 テクノロジーについて説明します。詳細な説明は含まれていません。

## 第 2 章「Oracle JSP 実装の概要」

この章では、OC4J が提供する JSP 実装の概要を説明し、移植可能な機能と Oracle 固有の付加価値機能についても説明します。

## 第 3 章「スタート・ガイド」

この章では、OC4J JSP コンテナ、OC4J Web サーバー構成および JSP 構成に必要なファイルについて説明します。

## 第 4 章「プログラミングに関する基本的な考慮事項」

この章では、JSP のプログラミングに関する基本的な考慮事項、JSP とサーブレット間の相互作用およびデータベース・アクセスについて、例を示して説明します。

## 第 5 章「JSP XML サポート」

この章では、JavaServer Pages による XML のサポートについて、JSP 仕様 1.2 で追加された機能を中心に説明します。JSP XML 構文および JSP XML ビューについても説明します。

## 第 6 章「プログラミングに関するその他の考慮事項」

この章では、開発者が考慮する必要がある、プログラミング、構成および実行時に関する一般的な問題について説明します。OC4J 環境固有の考慮事項についても説明します。

## 第 7 章「JSP の変換とデプロイ」

この章では、OC4J の JSP トランスレータ、および Oracle ojspc 事前変換ユーティリティの機能について説明し、デプロイに関する一般のおよび OC4J 固有の考慮事項について説明します。

## 第 8 章「JSP タグ・ライブラリ」

この章では、カスタム・タグ・ライブラリの標準的な JSP 1.2 フレームワークについて説明します。また、タグ・ライブラリ・サポートに関する OC4J の拡張機能、およびベンダー固有のコンパイル時タグについても説明します。

## 第 9 章「JSP グローバリゼーション・サポート」

この章では、グローバリゼーション・サポートの機能について説明します。

## 付録 A 「サーブレットと JSP の技術的なバックグラウンド」

この付録では、サーブレット・テクノロジーのバックグラウンドについて簡単に説明し、変換済ページ用の標準 JSP インタフェースについて説明します。

## 付録 B 「Apache JServ 環境」

この付録では、JServ サーブレット 2.0 環境について詳しく説明します。デプロイ、構成および特別なプログラミングに関する考慮事項についても説明します。

## 付録 C 「サード・パーティ・ライセンス」

この付録では、Oracle Application Server に搭載され、このマニュアルで説明されているサード・パーティ製品に対するサード・パーティ・ライセンスについて説明します。

# 関連ドキュメント

詳細は、次の Oracle リソースを参照してください。

Oracle Java Platform グループからは、OC4J に関する次の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 Stand Alone User's Guide』

ユーザーズ・ガイドのこのバージョンは特に、OC4J のスタンドアロン・バージョン用に書かれたもので、OTN-J からスタンドアロン・バージョンをダウンロードした場合に入手できます。スタンドアロン OC4J は、通常は本番環境ではなく、開発環境で使用されます。

- 『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、基本的なサーブレットの開発、JDBC と EJB の使用、アプリケーションの構築とデプロイ、サーブレットと Web サイトの構成など、OC4J でのサーブレットとサーブレット・コンテナの使用に関する情報を、サーブレット開発者に提供します。開発用のスタンドアロン環境および本番用の Oracle Application Server における OC4J について考慮しています。

- 『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』

このマニュアルは、タグ・ライブラリ、JavaBeans および OC4J が提供する他の Java ユティリティに関する概念的な情報、詳細な構文および使用に関する情報を提供します。他の Oracle 製品グループのタグ・ライブラリのサマリーもあります。

- 『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 が提供する標準的な 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（『Oracle Application Server 10g セキュリティ・ガイド』と混同しないでください）では、OC4J に固有のセキュリティ機能と実装について説明しています。JAAS（Java Authentication and Authorization Service）および他の Java セキュリティ・テクノロジーの使用に関する情報も記載されています。
- 『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。

- 『Oracle9i Java Developer's Guide』
- 『Oracle9i Java スタアド・プロシージャ開発者ガイド』
- 『Oracle9i JDBC 開発者ガイドおよびリファレンス』
- 『Oracle9i SQLJ 開発者ガイドおよびリファレンス』
- 『Oracle9i 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 10g 管理者ガイド』
- 『Oracle Application Server 10g セキュリティ・ガイド』
- 『Oracle Application Server 10g 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 10g グローバリゼーション・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 10g アップグレード 10g (9.0.4)』

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-J（Oracle Technology Network Japan）上の Oracle JDeveloper マニュアル  
<http://otn.oracle.co.jp/products/jdev/index.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 リファレンス』
- 『Oracle9i アプリケーション開発者ガイド - 基礎編』
- 『Oracle9i 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』
- 『Oracle9i SQL リファレンス』
- 『Oracle9i Net Services 管理者ガイド』
- 『Oracle Advanced Security 管理者ガイド』
- 『Oracle9i データベース・リファレンス』

リリース・ノート、インストレーション関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J (Oracle Technology Network Japan) に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

<http://otn.oracle.co.jp/membership/>

OTN-J のユーザー名とパスワードを取得済の場合は、次の OTN-J Web サイトのドキュメントのセクションに直接接続できます。

<http://otn.oracle.co.jp/document/>

Java サーブレットおよび JavaServer Pages に関する次の OTN-J Web サイト：

<http://otn.oracle.co.jp/tech/java/servlets/index.html>

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.html>
- Java サーブレット (最新仕様を含む) に関する Web サイト：  
<http://java.sun.com/products/servlet/index.html>
- JavaServer Pages の jsp-interest ディスカッション・フォーラム  
サブスクライブするには、メッセージ本文に次の行を記入して、[listserv@java.sun.com](mailto: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勧めします。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  |
|---------------|---|
| ...           | 文またはコマンド内の水平の省略記号は、例に直接関連しない文またはコマンドの一部が省略されていることを示します。   |
| <b>太字</b>     | 太字は、リンクやクリックするボタンなど、GUI コンポーネントを示します。   |
| 固定幅フォント       | テキスト内の固定幅文字は、実行可能ファイル、ファイル名、ディレクトリ名、Java クラス名、Java メソッド名、変数名、その他のプログラム要素 (JSP タグまたは属性、XML 要素または属性など)、またはデータベースの SQL コマンドや要素 (スキーマ名、表名、列名など) などの項目を表します。 |
| 固定幅フォントのイタリック | 固定幅フォントのイタリックは、プレースホルダまたは変数を示します。   |
| [ ]           | 大カッコは、カッコ内の項目を任意に選択することを表します。   |
|               | 縦線は、複数の選択項目の区切りに使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。   |



---

---

# JSP 全体の概要

この章では、JavaServer Pages (JSP) テクノロジーの標準機能、および JSP 実行モデルについて説明します。一般的な情報の詳細は、Sun 社の JSP 仕様を参照してください。

JSP の機能は、サーブレットの機能に基づいています。詳細は、Sun 社の Java サーブレット仕様も参照してください。

Oracle Application Server Containers for J2EE (OC4J) での JSP 実装の概要については、第 2 章「Oracle JSP 実装の概要」を参照してください。また、標準のサーブレットと JSP テクノロジーに関連するバックグラウンドについては、付録 A「サーブレットと JSP の技術的なバックグラウンド」で説明しています。

この章には、次の各項が含まれます。

- JavaServer Pages の概要
- JSP 構文の要素の概要
- JSP の実行

---

---

**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。この章にリストされていたアプリケーションは、OC4J デモにあります。OC4J デモは、OTN-J (Oracle Technology Network Japan) の次の場所から入手できます (OTN-J に登録する必要があります)。無償で登録できます)。

[http://otn.oracle.co.jp/sample\\_code/index.html](http://otn.oracle.co.jp/sample_code/index.html)

---

---

## JavaServer Pages の概要

JavaServer Pages は、Sun 社が提供するテクノロジーで、Web アプリケーション（Web サーバー上で動作するアプリケーション）が出力したページに、動的なコンテンツを生成するための便利な方法です。

Java サブレットのテクノロジーと密接に関連しているこのテクノロジーによって、開発者は、Java コードおよび外部 Java コンポーネントのコールを、Web ページの HTML コード（または XML などのマークアップ・コード）内に含めることができます。JavaServer Pages (JSP) テクノロジーは、JavaBeans および Enterprise JavaBeans (EJB) による、ビジネス・ロジックおよび動的な機能のためのフロントエンドとして機能します。

JSP コードは、Web ページ内の Web スクリプト・コード（JavaScript など）とは異なりません。通常の HTML ページに含めることができる内容は、JSP ページにも含めることができます。

データベース・アプリケーションの典型的な使用例では、JSP ページは、JavaBean、Enterprise JavaBean などのコンポーネントをコールし、Bean は通常、JDBC（場合によっては SQLJ）を使用して直接または間接的にデータベースにアクセスします。

JSP ページは、実行前に Java サブレットに変換され、他のサブレットと同様に、HTTP リクエストを処理してレスポンスを生成します。JSP テクノロジーによって、サブレットのコード作成がより便利になります。通常、変換は要求に応じて実行されますが、事前に実行される場合もあります。

さらに、JSP ページは、サブレットと完全に相互運用できます。つまり、JSP ページは、サブレットの出力をインクルードしたり、サブレットに出力を転送できます。一方、サブレットは、JSP ページの出力をインクルードしたり、JSP ページに出力を転送できません。

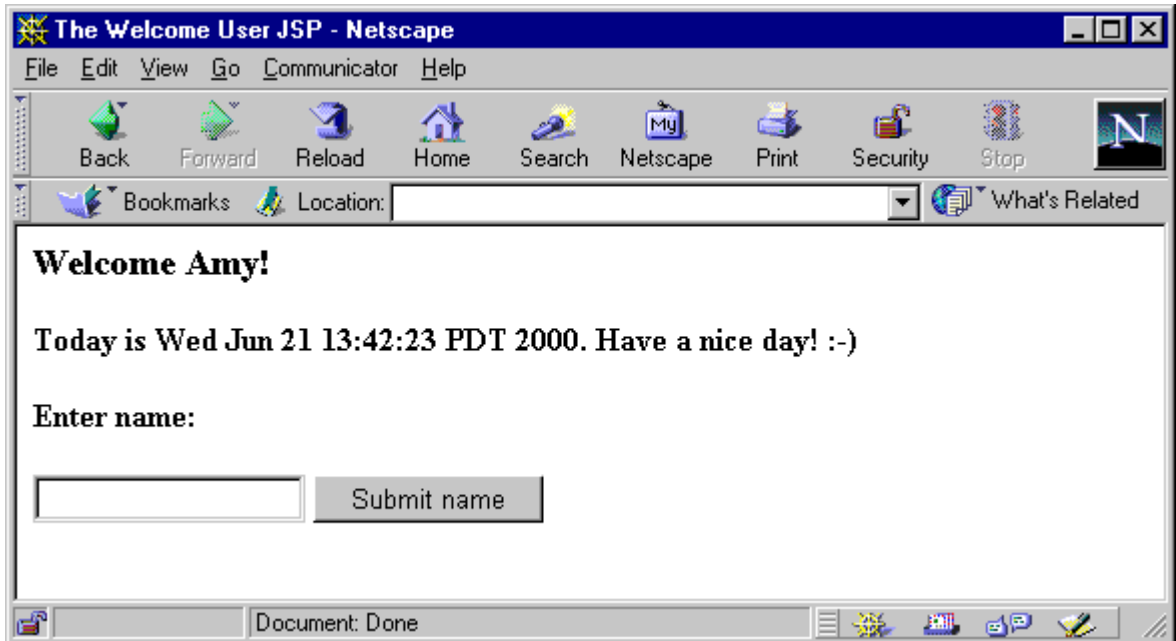
## JSP ページの例

次に、単純な JSP ページの例を示します。ここで使用されている JSP 構文の要素については、1-6 ページの「[JSP 構文の要素の概要](#)」を参照してください。

```
<HTML>
<HEAD><TITLE>The Welcome User JSP</TITLE></HEAD>
<BODY>
<% String user=request.getParameter("user"); %>
<H3>Welcome <%= (user==null) ? "" : user %>!</H3>
<P><B> Today is <%= new java.util.Date() %>. Have a nice day! :-)</B></P>
<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

前述の例に示すように、従来の JSP ページ内の Java の要素は、`<%` や `%>` などのタグで始まります (5-4 ページの「[JSP XML 文書の詳細](#)」で説明されているように、JSP XML 構文の場合は異なります)。この例の Java コードでは、HTTP リクエスト・オブジェクトからユーザー名を取得してユーザー名を出力し、現在の日付を取得します。

ユーザーが「Amy」という名前を入力すると、この JSP ページでは、次の出力が作成されます。



## JSP コーディングとサーブレット・コーディングの使いやすさの比較

Java コードと Java コールを HTML ページ内で組み合わせると、Java コードをサーブレットで直接使用するより使いやすくなります。JSP 構文を使用すると、動的 Web ページを簡単な方法でコーディングできるため、通常は、Java サーブレット構文より大幅に少ないコード量で済みます。次に、サーブレット・コードと JSP コードを比較した例を示します。

### サーブレット・コード

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Hello extends HttpServlet
{
    public void doGet(HttpServletRequest rq, HttpServletResponse rsp)
    {
        rsp.setContentType("text/html");
        try {
            PrintWriter out = rsp.getWriter();
            out.println("<HTML>");
            out.println("<HEAD><TITLE>Welcome</TITLE></HEAD>");
            out.println("<BODY>");
            out.println("<H3>Welcome!</H3>");
            out.println("<P>Today is "+new java.util.Date()+".</P>");
            out.println("</BODY>");
            out.println("</HTML>");
        } catch (IOException ioe)
        {
            // (error processing)
        }
    }
}
```

標準の `HttpServlet` 抽象クラス、`HttpServletRequest` インタフェースおよび `HttpServletResponse` インタフェースに関する情報は、A-3 ページの「[サーブレットのインタフェース](#)」を参照してください。

### JSP コード

```
<HTML>
<HEAD><TITLE>Welcome</TITLE></HEAD>
<BODY>
<H3>Welcome!</H3>
<P>Today is <%= new java.util.Date() %>.</P>
</BODY>
</HTML>
```

JSP 構文がより簡略であることがわかります。JSP 構文によって、パッケージのインポートや `try...catch` ブロックなど、Java のオーバーヘッドが軽減されます。

---

**注意：** OC4J 9.0.3 実装では、JSP ページにデフォルトでインポートされるパッケージのリストが変更されました。デフォルトのリストは、JSP 仕様に従って削減されました。詳細は、3-9 ページの「[デフォルト・パッケージのインポート](#)」を参照してください。したがって、Oracle9iAS リリース 2 (9.0.3) 以上の場合、前述の JSP の例には `java.io` パッケージをインポートするための構成設定が必要です。

---

さらに、JSP トランスレータは、.java 出力ファイル内で、大量のサーブレット・コーディングのオーバーヘッドを自動的に処理します。たとえば、標準の `javax.servlet.jsp.HttpJspPage` インタフェースを直接または間接的に実装し (A-11 ページの「[標準の JSP インタフェースとメソッド](#)」を参照)、コードを追加して HTTP セッションを取得します。

また、JSP ページの HTML は、サーブレット・コードのように Java の出力文に埋め込まれていないため、HTML オーサリング・ツールを使用して JSP ページを作成できます。

## ビジネス・ロジックとページ・プレゼンテーションの分離 : JavaBeans のコール

JSP テクノロジーによって、静的なページ・プレゼンテーションを決定する HTML コードの開発と、ビジネス・ロジックを処理して動的コンテンツを表示する Java コードの開発を分離できます。したがって、HTML の知識はあるが Java には不慣れなプレゼンテーションとレイアウトの担当と、Java の知識はあるが HTML には不慣れなコード担当との間で、メンテナンス作業を簡単に分割できます。

典型的な JSP ページでは、ほとんどの Java コードとビジネス・ロジックは、JSP ページに埋め込まれているコードではなく、JSP ページから起動する JavaBeans または Enterprise JavaBeans 内にあります。

JSP テクノロジーには、JavaBeans クラスのインスタンスを定義および作成するため、次の構文が用意されています。

```
<jsp:useBean id="pageBean" class="mybeans.NameBean" scope="page" />
```

この例では、`mybeans.NameBean` クラスのインスタンス `pageBean` を作成します。scope パラメータについては、この章で後述します。

この Bean インスタンスは、このページの後の方で、次の例のように使用できます。

```
Hello <%= pageBean.getNewName() %> !
```

この例では、`pageBean` の `newName` 属性が、たとえば、名前「Julie」(ユーザー入力などで設定) の場合は、「Hello Julie !」が出力されます。

ビジネス・ロジックをページ・プレゼンテーションから分離すると、ビジネス・ロジックと動的コンテンツを担当する Java のエキスパート (NameBean クラスのコードを所有およびメンテナンスする担当者) と、アプリケーション・ユーザーが参照する Web ページの静的なプレゼンテーションとレイアウトを担当する HTML のエキスパート (JSP ページの .jsp ファイルのコードを所有およびメンテナンスする担当者) の間で、作業を分担できます。

JavaBeans で使用するタグ (たとえば、JavaBean インスタンスを宣言するための useBean、Bean プロパティにアクセスするための getProperty と setProperty) の詳細は、1-15 ページの「標準アクション:JSP タグ」で説明します。

## JSP ページと代替マークアップ言語

JavaServer Pages テクノロジは通常、動的 HTML の出力に使用されますが、JSP 仕様では、構造化されたテキスト・ベースのドキュメント出力に関する追加の型もサポートしています。JSP トランスレータは、JSP 要素の外にあるテキストは処理しないため、Web ページに適切なテキストは、通常 JSP ページにも適切です。

JSP ページは、HTTP リクエストから情報を取得し、データベース・サーバーから (たとえば、SQL データベース問合せを使用して) 情報にアクセスします。この情報を、必要に応じて結合および処理し、動的コンテンツを持つ HTTP レスポンスに取り込みます。コンテンツは、HTML、DHTML、XHTML、XML などにフォーマットできます。

XML の JSP サポートの詳細は、第 5 章「JSP XML サポート」、および『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』を参照してください。

## JSP 構文の要素の概要

1-2 ページの「JSP ページの例」では、JSP 構文の単純な例を示しました。ここでは、トップレベルの構文カテゴリを説明します。

- **ディレクティブ**: JSP ページ全体に関する情報を伝達します。
- **スクリプト要素**: 宣言、式、スクリプトレット、コメントなどの Java コーディング要素です。
- **オブジェクトとスコープ**: JSP オブジェクトは、明示的または暗黙的に作成でき、指定されたスコープ内 (JSP ページやセッション内など) でアクセスできます。
- **操作**: オブジェクトを作成したり、JSP レスポンスの出力ストリームに影響を与えます。

この項では、基本的な構文とコード例も含めて、各カテゴリについて説明します。Bean プロパティの変換およびカスタム・タグ・ライブラリ (カスタム・アクションに使用されます) の概要についても説明します。詳細は、Sun 社の JSP 仕様を参照してください。



---

---

**注意：** この項では、従来の JSP 構文について説明します。JSP XML 構文および JSP XML 文書の詳細は、第 5 章「JSP XML サポート」を参照してください。

---

---

## ディレクティブ

ディレクティブは、JSP ページ全体に関する指示を JSP コンテナに提供します。この情報は、ページの変換または実行で使用されます。基本的な構文は、次のとおりです。

```
<%@ directive attribute1="value1" attribute2="value2"... %>
```

JSP 仕様では、次のディレクティブをサポートします。

- page
- include
- taglib

**page ディレクティブ** ページに依存する属性を指定します。たとえば、スクリプト言語、コンテンツ・タイプ、文字エンコード、拡張するクラス、インポートするパッケージ、使用するエラー・ページ、JSP ページの出力バッファ・サイズ、バッファが満杯になったときにバッファを自動的にフラッシュするかどうかなどがあります。例：

```
<%@ page language="java" import="packages.mypackage" errorPage="boof.jsp" %>
```

また、自動フラッシュを有効にし、JSP ページの出力バッファ・サイズを 20KB に設定する場合は、次のようにします。

```
<%@ page autoFlush="true" buffer="20kb" %>
```

次に、ページのバッファを無効にする例を示します。

```
<%@ page buffer="none" %>
```

---

---

### 注意：

- デフォルトのバッファ・サイズは 8KB です。
  - `buffer="none"` の場合、`autoFlush="true"` の設定は無効です。
  - エラー・ページを使用する JSP ページでは、バッファを有効にする必要があります。エラー・ページへの転送が行われる（ブラウザには出力されません）と、バッファはクリアされます。
  - Oracle JSP の実装では、デフォルト言語は「java」に設定されています。ただし、デフォルト言語は、明示的に設定することをお勧めします。
  - `page` ディレクティブの属性を使用して、JSP ページおよびレスポンス・オブジェクトのコンテンツ・タイプとキャラクタ・セットを設定する方法は、9-2 ページの「[page ディレクティブでのコンテンツ・タイプの設定](#)」を参照してください。
- 
- 

**include ディレクティブ** 変換時に JSP ページに挿入されるテキストまたはコードを含むリソースを指定します。例：

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

リソースへのページ相対パスまたはコンテキスト相対パスのいずれかを指定します。ページ相対パスおよびコンテキスト相対パスの詳細は、1-27 ページの「[JSP ページのリクエスト](#)」を参照してください。

---

---

### 注意：

- `include` ディレクティブは「静的なインクルード」と呼ばれ、この章で後述する `jsp:include` 操作に本質は類似していますが、`jsp:include` は変換時ではなくリクエスト時に効果があります。6-2 ページの「[静的なインクルードと動的なインクルードの比較](#)」を参照。
  - `include` ディレクティブは、同じサーブレット・コンテキスト（アプリケーション）内のファイル間でのみ使用できます。
  - インクルード・ファイルのネーミング規則の詳細は、3-10 ページの「[JSP ファイルのネーミング規則](#)」を参照してください。
- 
-

**taglib ディレクティブ** このディレクティブを使用して、JSP ページで使用するカスタム JSP タグのライブラリを指定します。ベンダーは、独自のタグ・セットを使用して、JSP 機能を拡張できます。このディレクティブには、タグ・ライブラリ・ディスクリプタへのポインタ、およびこのライブラリからタグの使用方法を識別するための接頭辞が含まれます。例：

```
<%@ taglib uri="/oracustomtags" prefix="oracust" %>
```

ライブラリ内のいずれかのタグをページの後のほうで使用するとき、常に `oracust` 接頭辞を使用します。このライブラリには、タグの `dbaseAccess` が含まれていると仮定します。

```
<oracust:dbaseAccess ... >
...
</oracust:dbaseAccess>
```

JSP タグ・ライブラリとタグ・ライブラリ・ディスクリプタの概要は、1-23 ページの「[カスタム・タグ・ライブラリ](#)」で説明します。詳細は、[第 8 章「JSP タグ・ライブラリ」](#)を参照してください。

## スクリプト要素

JSP スクリプト要素には、JSP ページに表示可能な Java コードの次のカテゴリが含まれます。

- 宣言
- 式
- スクリプトレット
- コメント

**宣言** JSP ページで使用するメソッドまたはメンバー変数を宣言する文です。

JSP 宣言は、`<%!...%>` 宣言タグ内で標準の Java 構文を使用し、メンバー変数またはメソッドを宣言します。これによって、生成されたサーブレット・コードで、対応する宣言が行われます。例：

```
<%! double f1=0.0; %>
```

この例では、メンバー変数の `f1` を宣言します。JSP トランスレータで生成されたサーブレット・クラス・コードで、`f1` はクラスのトップレベルで宣言されます。

---

---

**注意：** メンバー変数とは対照的に、メソッド変数は、後述するように、JSP スクリプトレット内で宣言されます。メンバー変数とメソッド変数の比較は、6-10 ページの「[メソッド変数宣言とメンバー変数宣言の比較](#)」を参照してください。

---

---

**式** 評価され、必要に応じて文字列値に変換され、ページ内の検出された場所に表示される Java 式です。

JSP 式は、セミコロンで終了せず、`<%=...%>` タグ内に含まれます。例：

```
<P><B> Today is <%= new java.util.Date() %>. Have a nice day! </B></P>
```

---

---

**注意：** `jsp:setProperty` 文などに含まれるリクエスト時属性の JSP 式は、文字列値に変換する必要はありません。

---

---

**スクリプトレット** ページのマークアップ言語内に記述される、Java コードの断片です。

スクリプトレットまたはコードの一部分は、1 行または複数行の Java コードで構成されている場合があります。このスクリプトレットを JSP ページの HTML コード内で使用すると、条件ブランチやループなどを設定できます。

JSP スクリプトレットは `<%=...%>` スクリプトレット・タグ内に含まれ、通常の Java 構文を使用します。

例 1:

```
<% if (pageBean.getNewName().equals("")) { %>
    I don't know you.
<% } else { %>
    Hello <%= pageBean.getNewName() %>.
<% } %>
```

1 行で構成された 3 つの JSP スクリプトレットが、2 行の HTML コードと交互に記述され、その 1 行にはセミコロンが不要な JSP 式が含まれています。JSP 構文によって、HTML コードは、`if` ブランチと `else` ブランチ（スクリプトレットに設定された Java の中カッコ内）内で条件付きで実行されるコードになります。

前述の例では、JavaBean インスタンスの `pageBean` を使用しています。

例 2:

```
<% if (pageBean.getNewName().equals("")) { %>
    I don't know you.
    <% empmgr.unknownemployee();
} else { %>
    Hello <%= pageBean.getNewName() %>.
    <% empmgr.knownemployee();
} %>
```

この例では、Java コードがスクリプトレットに追加されています。ここでは、JavaBean インスタンスの `pageBean` を使用し、すでにインスタンス化されているオブジェクトの `empMgr` には、既知または未知の従業員に対して適切な機能を実行するメソッドが設定されていることが前提になっています。

---

---

**注意：** メンバー変数とは対照的に、メソッド変数の宣言には、次の例のように JSP スクリプトレットを使用します。

```
<% double f2=0.0; %>
```

このスクリプトレットは、メソッド変数の `f2` を宣言します。JSP トランスレータで生成されたサブレット・クラス・コードでは、`f2` は、サブレットのサービス・メソッド内で変数として宣言されます。

メンバー変数は、前述のように、JSP 宣言で宣言されます。

メソッド変数とメンバー変数の比較は、6-10 ページの「[メソッド変数宣言とメンバー変数宣言の比較](#)」を参照してください。

---

---

**コメント** Java コード内に埋め込まれたコメントと同様に、JSP コード内に埋め込まれた、開発者のコメントです。

コメントは、`<%--...-->` 構文内に含まれます。例：

```
<%-- Execute the following branch if no user name is entered. -->
```

HTML のコメントとは異なり、JSP のコメントは、ユーザーがブラウザでページのソースを表示しても参照できません。

## JSP オブジェクトとスコープ

このマニュアルの JSP オブジェクトという用語は、JSP ページで宣言されるか、または JSP ページからアクセス可能な Java クラス・インスタンスを指します。JSP オブジェクトは、次のいずれかです。

- **明示的：**明示的なオブジェクトは、JSP ページのコード内で宣言および作成され、選択した `scope` の設定に従って、その JSP ページおよびその他のページにアクセスできます。

または

- **暗黙的：**暗黙的なオブジェクトは、基礎となる JSP 機能によって作成され、特定のオブジェクト型に固有の `scope` 設定に従って、JSP ページの Java スクリプトレットまたは式からアクセスできます。

この項には、次の項目が含まれます。

- [明示的なオブジェクト](#)
- [暗黙的なオブジェクト](#)
- [暗黙的なオブジェクトの使用](#)
- [オブジェクトのスコープ](#)

### 明示的なオブジェクト

通常、明示的なオブジェクトとは、`jsp:useBean` 操作文で宣言および作成された **JavaBean** インスタンスです。`jsp:useBean` 文などの操作文は、1-15 ページの「[標準アクション: JSP タグ](#)」で説明しています。ここでは例を示します。

```
<jsp:useBean id="pageBean" class="mybeans.NameBean" scope="page" />
```

この文は、`mybeans` パッケージにある `NameBean` クラスの `pageBean` インスタンスを定義します。`scope` パラメータについては、1-14 ページの「[オブジェクトのスコープ](#)」を参照してください。

Java クラス・インスタンスを Java プログラム内で作成するように、オブジェクトは、Java スクリプトレットまたは宣言内でも作成できます。

### 暗黙的なオブジェクト

JSP テクノロジーによって、JSP ページで一連の暗黙的なオブジェクトが使用可能になります。これらのオブジェクトは、JSP コンテナによって自動的に作成される Java オブジェクトで、基礎となるサーブレット環境との相互作用を可能にします。

次に、使用可能な暗黙的なオブジェクトを示します。これらのオブジェクトで使用可能なメソッドについては、次のサイト（サーブレット 2.2 クラス用とサーブレット 2.3 クラス用）にある Sun 社の指定クラスとインタフェースに関する Javadoc を参照してください。

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

<http://java.sun.com/products/servlet/2.3/javadoc/index.html>

- `page`

このオブジェクトは JSP ページ実装クラスのインスタンスで、ページの変換時に作成されます。ページ実装クラスは、インタフェース `javax.servlet.jsp.HttpJspPage` を実装します。`page` オブジェクトは、JSP ページ内では `this` と同じ意味であることに注意してください。
- `request`

このオブジェクトは、HTTP リクエストを表し、`javax.servlet.ServletRequest` インタフェースを拡張する `javax.servlet.http.HttpServletRequest` インタフェースを実装するクラスのインスタンスです。

- **response**

このオブジェクトは、HTTP レスポンスを表し、`javax.servlet.ServletResponse` インタフェースを拡張する `javax.servlet.http.HttpServletResponse` インタフェースを実装するクラスのインスタンスです。

特定のリクエストに対する `response` オブジェクトと `request` オブジェクトは、相互に関連があります。
- **pageContext**

このオブジェクトは、JSP ページのページ・コンテキストを表し、JSP ページ・インスタンスのすべての `page` スコープ・オブジェクトの格納およびアクセスで使用されます。`pageContext` オブジェクトは、`javax.servlet.jsp.PageContext` クラスのインスタンスです。

`pageContext` オブジェクトには `page` スコープがあり、これによって、関連付けられた JSP ページ・インスタンスにのみアクセスできます。
- **session**

このオブジェクトは、HTTP セッションを表す、`javax.servlet.http.HttpSession` インタフェースを実装するクラスのインスタンスです。
- **application**

このオブジェクトは、Web アプリケーションのサーブレット・コンテキストを表し、`javax.servlet.ServletContext` インタフェースを実装するクラスのインスタンスです。

`application` オブジェクトは、単一の JVM 内でアプリケーションのインスタンスの一部として実行している JSP ページ・インスタンスからアクセスできます（プログラマは、JVM の使用に関して、サーバーのアーキテクチャの知識が必要です）。
- **out**

コンテンツを JSP ページ・インスタンスの出力ストリームに書き込むためのオブジェクトです。このオブジェクトは、`java.io.Writer` クラスを拡張する `javax.servlet.jsp.JspWriter` クラスのインスタンスです。

`out` オブジェクトは、特定のリクエストについて、`response` オブジェクトに関連付けられます。
- **config**

このオブジェクトは、JSP ページのサーブレット構成を表し、`javax.servlet.ServletConfig` インタフェースを実装するクラスのインスタンスです。一般的に、サーブレット・コンテナは、`ServletConfig` インスタンスを使用して、初期化中のサーブレットに情報を提供します。この情報の一部が、適切な `ServletContext` インスタンスです。

- `exception` (JSP エラー・ページのみ)

この暗黙的なオブジェクトは、JSP エラー・ページ (別の JSP ページから例外がスローされた場合に処理を転送するページ) にも適用されます。page ディレクティブの `isErrorPage` 属性が `true` に設定されている必要があります。

暗黙的な `exception` オブジェクトは、別の JSP ページからスローされ、結果的に現在のエラー・ページを起動したキャッチされなかった例外を表す `java.lang.Exception` インスタンスです。

`exception` オブジェクトには、例外の発生時に処理が転送された JSP エラー・ページ・インスタンスからのみアクセスできます。JSP エラー処理の例と `exception` オブジェクトの使用方法は、4-24 ページの「[実行時エラーの処理](#)」を参照してください。

### 暗黙的なオブジェクトの使用

前項で説明した暗黙的なオブジェクトは、いずれも便利なオブジェクトです。次の例では、`request` オブジェクトを使用して HTTP リクエストから `username` パラメータの値を取得し、表示します。

```
<H3> Welcome <%= request.getParameter("username") %> ! </H3>
```

`request` オブジェクトは、他の暗黙的なオブジェクトと同様に、自動的に使用可能になり、明示的にインスタンス化されません。

### オブジェクトのスコープ

JSP ページ内のオブジェクトは、明示的か暗黙的かに関係なく、特定のスコープ内でアクセス可能です。明示的なオブジェクト (`jsp:useBean` 操作で作成された `JavaBean` インスタンスなど) の場合は、1-12 ページの「[明示的なオブジェクト](#)」の例に示すように、次の構文を使用して明示的にスコープを設定できます。

```
scope="scopevalue"
```

スコープには、次の 4 種類があります。

- `scope="page"` (デフォルトのスコープ) : オブジェクトには、そのオブジェクトが作成された JSP ページ内からのみアクセスできます。page スコープのオブジェクトは、暗黙的な `pageContext` オブジェクトに格納されます。page スコープは、ページの実行が停止すると終了します。

JSP ページの実行中に、ユーザーがページをリフレッシュすると、すべての page スコープのオブジェクトについて新規インスタンスが作成されます。

- `scope="request"`: オブジェクトには、そのオブジェクトを作成した JSP ページと同じ HTTP リクエスト・サービスを提供している JSP ページからアクセスできます。request スコープのオブジェクトは、暗黙的な `request` オブジェクトに格納されます。request スコープは、HTTP リクエストが完了すると終了します。



- `scope="session"`: オブジェクトには、そのオブジェクトを作成した JSP ページと同じ HTTP セッションを共有する JSP ページからアクセスできます。`session` スコープのオブジェクトは、暗黙的な `session` オブジェクトに格納されます。`session` スコープは、HTTP セッションがタイムアウトになるか、または無効になると終了します。
- `scope="application"`: オブジェクトには、単一の Java Virtual Machine 上で、そのオブジェクトを作成した JSP ページと同じ Web アプリケーションで使用される JSP ページからアクセスできます。これは、Java の静的変数の概念と同じです。`application` スコープのオブジェクトは、暗黙的な `application` サブレット・コンテキスト・オブジェクトに格納されます。`application` スコープは、アプリケーション自体が終了するか、JSP コンテナまたはサブレット・コンテナが停止すると終了します。

次のように、狭いスコープから広いスコープへ 4 つのスコープに区分けされます。

page < request < session < application

1 つのアプリケーション内にある異なるページ間でオブジェクトを共有する場合、たとえば、あるページから別のページに実行を転送したり、あるページのコンテンツを別のページにインクルードする場合は、共有するオブジェクトに対して `page` スコープを使用できません。この場合は、ページごとに関連付けられた個別のオブジェクト・インスタンスを使用します。ページ間でのオブジェクト共有に使用できる最も狭いスコープは、`request` です (ページをインクルードしたり転送する方法については、次の「標準アクション: JSP タグ」を参照)。

---

**注意:** `request`、`session` および `application` スコープはサブレットにも適用されます。

---

## 標準アクション: JSP タグ

JSP 操作の要素により、JSP ページの実行中に発生する操作 (Java オブジェクトをインスタンス化し、ページに対して使用可能にするなど) が行われます。次のような操作が含まれます。

- JavaBean インスタンスを作成し、そのプロパティにアクセスする操作
- 別の HTML ページ、JSP ページまたはサブレットに実行を転送する操作
- 外部リソースを JSP ページにインクルードする操作

標準アクションについては、JSP 仕様に一連のタグが定義されています。JSP ページのコードは、この章で前述したディレクティブとスクリプト要素を使用して作成できますが、ここで説明する標準的なタグを使用すると、機能性や利便性が向上します。

JSP の標準アクションに対する一般的なタグの構文は、次のとおりです。

```
<jsp:tag attr1="value1" attr2="value2" ... attrN="valueN">
...body...
</jsp:tag>
```

ボディがない場合は、次のとおりです。

```
<jsp:tag attr1="value1", ..., attrN="valueN" />
```

JSP 仕様には、次の標準アクション・タグが含まれます。次に、各タグについて簡単に説明します。

- `jsp:usebean`
- `jsp:setProperty`
- `jsp:getProperty`
- `jsp:param`
- `jsp:include`
- `jsp:forward`
- `jsp:plugin`

**jsp:useBean タグ** `jsp:useBean` タグは、Java 型のインスタンス（通常は `JavaBean` クラス）にアクセスし、Java 型のインスタンスを作成します。また、インスタンスを指定の名前または ID に関連付けます。インスタンスは、指定したスコープのスクリプト変数として、その ID を介して使用できます。スクリプト変数の概要は、1-23 ページの「[カスタム・タグ・ライブラリ](#)」を参照してください。スコープの詳細は、1-11 ページの「[JSP オブジェクトとスコープ](#)」を参照してください。

主な属性は、`class`、`type`、`id` および `scope` です（使用頻度は多くありませんが、後述の `beanName` 属性もあります）。

`id` 属性を使用して、インスタンス名を指定します。JSP コンテナは最初に、指定したスコープ内で、指定した型の指定の ID によってオブジェクトを検索します。オブジェクトが見つからない場合、コンテナはそのオブジェクトの作成を試みます。

`class` 属性は、JSP コンテナが必要に応じてインスタンス化できるクラスを指定するために使用します。クラスは、抽象クラスにはできません。また、引数のないコンストラクタが必要です。`type` 属性は、JSP コンテナがインスタンス化できない型（インタフェース、抽象クラス、あるいは引数のないコンストラクタを持たないクラスのうちのいずれか）を指定するために使用します。インスタンスがすでに存在する場合やインスタンス化可能なクラスのインスタンスが型に割り当てられる場合は、`type` を使用します。次に、典型的な 3 種類の使用例を示します。

- ターゲット・スコープにすでに存在しているインスタンスを指定するには、`type` と `id` を使用します。
- クラスのインスタンス名を指定するには、`class` と `id` を使用します。インスタンスは、ターゲット・スコープにすでに存在しているインスタンスか、JSP コンテナによって新規作成されるインスタンスのいずれかです。

- インスタンス化するクラス、およびインスタンスを割り当てる型を指定するには、`class`、`type` および `id` を使用します。この場合、クラスは、型に対して正式に割り当て可能であることが必要です。

`scope` 属性を使用して、インスタンスの範囲を指定します。ページ・コンテキスト・オブジェクトに関連付けるインスタンスの場合は `page`、HTTP リクエスト・オブジェクトに関連付けるインスタンスの場合は `request`、HTTP セッション・オブジェクトに関連付けるインスタンスの場合は `session`、サーブレット・コンテキストに関連付けるインスタンスの場合は `application` です。

`class` 属性のかわりに、`beanName` 属性を使用することもできます。この場合は、クラス名のかわりにシリアライズ可能なリソースを指定するオプションがあります。`beanName` 属性を使用すると、JSP コンテナは `java.beans.Beans` クラスの `instantiate()` メソッドを使用してインスタンスを作成します。

次の例では、型 `MyIntfc` の `request` スコープ・インスタンス `reqobj` を使用しています。`MyIntfc` はインタフェースであり、直接インスタンス化できないため、`reqobj` がすでに存在している必要があります。

```
<jsp:useBean id="reqobj" type="mypkg.MyIntfc" scope="request" />
```

次の例では、クラス `PageBean` の `page` スコープ・インスタンス `pageobj` を使用しています（必要に応じて、このインスタンスを最初に作成します）。

```
<jsp:useBean id="pageobj" class="mybeans.PageBean" scope="page" />
```

次の例では、クラス `SessionBean` のインスタンスを作成し、そのインスタンスを型 `MyIntfc` の変数 `sessobj` に割り当てています。

```
<jsp:useBean id="sessobj" class="mybeans.SessionBean"
             type="mypkg.MyIntfc" scope="session" />
```

**jsp:setProperty タグ** `jsp:setProperty` タグは、1 つ以上の Bean プロパティを設定します。Bean は、`jsp:useBean` タグにすでに指定されている必要があります。指定のプロパティに値を直接指定したり、指定のプロパティの値に関連の HTTP リクエストのパラメータから取得したり、HTTP リクエストのパラメータから一連のプロパティと値を繰り返し取得できます。

次の例では、`pageBean` インスタンス（前述の `jsp:useBean` の例で定義済）の `user` プロパティの値を「Smith」に設定します。

```
<jsp:setProperty name="pageBean" property="user" value="Smith" />
```

次の例では、HTTP リクエストの `username` というパラメータの値セットに従って、`pageBean` インスタンスの `user` プロパティを設定します。

```
<jsp:setProperty name="pageBean" property="user" param="username" />
```

Bean のプロパティとリクエスト・パラメータが同じ名前 (user) の場合は、次のようにプロパティを設定できます。

```
<jsp:setProperty name="pageBean" property="user" />
```

次の例では、HTTP リクエストのパラメータで、Bean のプロパティ名とリクエストのパラメータ名を繰り返し一致させ、対応するリクエストのパラメータ値に従って Bean のプロパティ値を設定します。

```
<jsp:setProperty name="pageBean" property="*" />
```

jsp:setProperty タグを使用すると、文字列以外のプロパティの値を指定する場合でも、文字列入力を使用できます。これは、バックグラウンドで変換が行われるためです。1-21 ページの「[文字列値から Bean プロパティへの変換](#)」を参照。

---

**重要：** property="\*" の場合は、次の点に注意してください。

- エラーが発生した場合に反復を継続するように指定するには、setProperty\_onerr\_continue 構成パラメータを true に設定します。このパラメータの詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。(以前のリリースでは、デフォルトで反復が継続されました。OC4J 9.0.4 実装では、エラーが発生するとデフォルトで反復が停止されます。)
- JSP 仕様では、プロパティの設定順序を規定していません。順序を指定し、JSP ページを移植可能にする場合は、各プロパティに対して異なる jsp:setProperty 文を使用する必要があります。また、jsp:setProperty 文を個別に使用する場合、JSP トランスレータは、対応する setXXX() メソッドを直接生成できます。この場合は、変換中にのみイントロスペクションが発生します。実行時は、Bean のイントロスペクションが発生しないので、コストがかかりません。

---

**jsp:getProperty タグ** jsp:getProperty タグは、Bean のプロパティ値を読み取って Java 文字列に変換し、暗黙的な out オブジェクトに配置します。これによって、文字列値を出力として表示できます。Bean は、jsp:useBean タグにすでに指定されている必要があります。文字列に変換する場合、プリミティブ型は直接変換され、オブジェクト型は、java.lang.Object クラスに指定されている toString() メソッドを使用して変換されます。

次の例では、pageBean Bean の user プロパティの値を、out オブジェクトに出力します。

```
<jsp:getProperty name="pageBean" property="user" />
```

**jsp:param タグ** jsp:param タグは、jsp:include タグ、jsp:forward タグおよび jsp:plugin タグ (後述) とともに使用できます。

`jsp:param` タグを `jsp:forward` タグおよび `jsp:include` タグとともに使用すると、HTTP request オブジェクトのパラメータ値に対して名前 / 値ペアが必要に応じて提供されます。この操作で指定した新規のパラメータと値は、request オブジェクトに追加され、古い値よりも優先されます。

次の例では、request オブジェクトの `username` パラメータに、Smith という値を設定します。

```
<jsp:param name="username" value="Smith" />
```

**jsp:include タグ** `jsp:include` タグは、リクエスト時にページが表示されると、静的または動的な追加リソースをページに挿入します。リソースは、相対 URL（ページ相対またはアプリケーション相対）を使用して指定します。例：

```
<jsp:include page="/templates/userinfopage.jsp" flush="true" />
```

`flush` 属性を「true」に設定すると、`jsp:include` 操作の実行時に、バッファがブラウザにフラッシュされます。JSP 仕様および OC4J JSP コンテナは、「true」または「false」のいずれかの設定をサポートします。デフォルトは「false」です（JSP 仕様 1.1 でサポートされる設定は「true」のみで、`flush` は必須属性です）。

次の例に示すように、`jsp:param` タグを持つ操作ボディも指定できます。

```
<jsp:include page="/templates/userinfopage.jsp" flush="true" >
  <jsp:param name="username" value="Smith" />
  <jsp:param name="userempno" value="9876" />
</jsp:include>
```

次の構文は、前述の例の代替として機能します。

```
<jsp:include page="/templates/userinfopage.jsp?username=Smith&userempno=9876" flush="true" />
```

---

---

### 注意：

- 動的なインクルードと呼ばれる `jsp:include` タグは、この章で前述した `include` ディレクティブと特性は類似していますが、変換時ではなくリクエスト時に効果があります。6-2 ページの「[静的なインクルードと動的なインクルードの比較](#)」を参照。
- `jsp:include` タグは、同じサーブレット・コンテキスト（アプリケーション）内のページ間でのみ使用できます。

---

---

**jsp:forward タグ** `jsp:forward` タグは、現行のページの実行を事実上終了し、その出力を破棄し、新規ページ（HTML ページ、JSP ページまたはサーブレットのいずれか）をディスパッチします。

`jsp:forward` タグを使用するには、JSP ページをバッファする必要があります。page ディレクティブに `buffer="none"` は設定できません。この操作では、バッファがクリアされ、コンテンツはブラウザに出力されません。

`jsp:include` の場合と同様、次の 2 番目の例に示すように、`jsp:param` タグを持つ操作ボディも指定できます。

```
<jsp:forward page="/templates/userinfopage.jsp" />
```

または

```
<jsp:forward page="/templates/userinfopage.jsp" >
  <jsp:param name="username" value="Smith" />
  <jsp:param name="userempno" value="9876" />
</jsp:forward>
```

---

### 注意：

- この `jsp:forward` の例と前述の `jsp:include` の例の違いは、`jsp:include` の例では、現行のページの出力内に `userinfopage.jsp` を挿入したのに対して、`jsp:forward` の例では、現行のページの実行を停止して `userinfopage.jsp` を表示していることです。
  - `jsp:forward` タグは、同じサーブレット・コンテキスト内のページ間でのみ使用できます。
  - `jsp:forward` タグによって、元の `request` オブジェクトは、ターゲット・ページに転送されます。`request` オブジェクトを転送しない場合は、標準の `javax.servlet.http.HttpServletResponse` インタフェースに指定されている `sendRedirect(String)` メソッドを使用できます。これによって、一時的なリダイレクト・レスポンスが、指定したリダイレクト場所の URL を使用して、クライアントに送信されます。相対 URL は開発者が指定できます。この相対 URL は、サーブレット・コンテナによって絶対 URL に変換されます。
-

**jsp:plugin タグ** `jsp:plugin` タグによって、指定のアプレットまたは `JavaBean` がクライアント・ブラウザで実行されます。必要に応じて、`Java` プラグイン・ソフトウェアのダウンロードが先行して実行されます。

`jsp:plugin` 属性を使用して、実行するアプレットやコード・ベースなどの構成情報を指定します。JSP コンテナは、ダウンロード用のデフォルト URL を提供できますが、`nspluginurl="url"` 属性 (Netscape ブラウザの場合) または `iepluginurl="url"` 属性 (Internet Explorer ブラウザの場合) を指定することもできます。

`jsp:params` の開始タグと終了タグの間にネストされている `jsp:param` タグを使用して、アプレットまたは `JavaBean` にパラメータを指定します (`jsp:params` の開始タグと終了タグは、`jsp:include` 操作または `jsp:forward` 操作で `jsp:param` を使用する場合は不要です)。

プラグインが実行できない場合は、`jsp:fallback` の開始タグと終了タグを使用して、実行する代替テキストに定めます。

次の例は、Sun 社 JSP 仕様 1.2 からの抜粋で、アプレットのプラグインの使用方法を示しています。

```
<jsp:plugin type=applet code="Molecule.class" codebase="/html" >
  <jsp:params>
    <jsp:param name="molecule" value="molecules/benzene.mol" />
  </jsp:params>
  <jsp:fallback>
    <p> Unable to start the plugin. </p>
  </jsp:fallback>
</jsp:plugin>
```

ARCHIVE、HEIGHT、NAME、TITLE、WIDTH など、その他のパラメータも `jsp:plugin` タグ内で使用できます。これらのパラメータの使用方法は、一般的な HTML 仕様に従います。

## 文字列値から Bean プロパティへの変換

JSP ページで `jsp:useBean` タグを介して `JavaBean` を使用し、次に `jsp:setProperty` タグを使用して `Bean` プロパティを設定する場合は、バックグラウンドで変換が行われるため、文字列以外のプロパティの値を指定する場合でも、文字列入力を使用できます。変換には、2つの手順があります。次の各項で説明します。

- 一般的なプロパティ変換
- プロパティ・エディタによるプロパティの型変換

## 一般的なプロパティ変換

関連するプロパティ・エディタがない Bean プロパティについて、文字列値を使用してプロパティを設定する場合の変換方法を表 1-1 に示します。

表 1-1 属性の変換方法

| プロパティの型             | 変換  |
|---------------------|---|
| Boolean または boolean | Boolean クラスの <code>valueOf (String)</code> メソッドに基づきます。                                  |
| Byte または byte       | Byte クラスの <code>valueOf (String)</code> メソッドに基づきます。                                     |
| Character または char  | String クラスの <code>charAt (0)</code> メソッド (索引値として 0 を入力) に基づきます。                         |
| Double または double   | Double クラスの <code>valueOf (String)</code> メソッドに基づきます。                                   |
| Integer または int     | Integer クラスの <code>valueOf (String)</code> メソッドに基づきます。                                  |
| Float または float     | Float クラスの <code>valueOf (String)</code> メソッドに基づきます。                                    |
| Long または long       | Long クラスの <code>valueOf (String)</code> メソッドに基づきます。                                     |
| Short または short     | Short クラスの <code>valueOf (String)</code> メソッドに基づきます。                                    |
| Object              | String のコンストラクタがコールされる場合と同じように、リテラルの文字列を入力します。<br>Object インスタンスとして String インスタンスが戻されます。 |

## プロパティ・エディタによるプロパティの型変換

Bean プロパティには、関連するプロパティ・エディタを指定できます。このエディタは、`java.beans.PropertyEditor` インタフェースを実装するクラスです。このようなクラスは、プロパティの編集に使用する GUI をサポートします。一般的に、標準の Java 型には標準のプロパティ・エディタがあり、ユーザー定義型にはユーザー定義のプロパティ・エディタがあります。ただし、OC4J の JSP 実装で検索されるのは、ユーザー定義のプロパティ・エディタのみです。`sun.beans.editors` パッケージのデフォルトのプロパティ・エディタは考慮されません。

プロパティ・エディタの詳細、およびプロパティ・エディタを型に関連付ける方法については、Sun 社の JavaBeans API 仕様を参照してください。

プロパティ・エディタが関連付けられているプロパティを、JavaBeans 仕様で指定されているように設定するには、以前と同様に文字列値を使用できます。この場合は、入力された文字列を適切な型の変換する際に、`PropertyEditor` インタフェースで指定されている `setAsText (String text)` メソッドが使用されます (`setAsText ()` メソッドが `IllegalArgumentException` をスローした場合、変換は失敗します)。



## カスタム・タグ・ライブラリ

JSP 仕様では、前述の標準的な JSP タグ以外に、ベンダーが独自のタグ・ライブラリを定義できます。また、ベンダーは、顧客が独自のタグ・ライブラリを定義できるフレームワークを実装することもできます。

タグ・ライブラリは、カスタム・タグのコレクションを定義し、JSP のサブ言語とみなすことができます。開発者は、JSP ページを手動でコーディングするときにタグ・ライブラリを直接使用できますが、Java 開発ツールで自動的に使用することもできます。標準のタグ・ライブラリは、異なる JSP コンテナ実装間で移植可能であることが必要です。

タグ・ライブラリは、1-7 ページの「[ディレクティブ](#)」で説明した `taglib` ディレクティブを使用して、JSP ページにインポートします。

JSP タグ・ライブラリに対する標準的な JavaServer Pages のサポートに関する主な概念には、次の項目が含まれます。

- タグ・ライブラリ・ディスクリプタ

タグ・ライブラリ・ディスクリプタ (TLD) は、タグ・ライブラリとライブラリの各タグに関する情報が含まれた XML 文書です。TLD のファイル名には、拡張子 `.tld` が付きます。

- タグ・ハンドラ

カスタム・タグの動作を決めるタグ・ハンドラは、標準の `javax.servlet.jsp.tagext` パッケージで `Tag`、`IterationTag` または `BodyTag` のいずれかのインタフェースを実装する Java クラスのインスタンスです。実装するインタフェースは、タグにボディがあるかどうか、およびタグ・ハンドラがボディ・コンテンツへアクセスする必要があるかどうかによって決まります。

- スクリプト変数

カスタム・タグ操作によって、タグ自体、またはスクリプトレットなどのスクリプト要素で使用できるサーバー・サイド・オブジェクトを作成できます。このためには、スクリプト変数の作成または更新が必要です。

カスタム・タグによって定義されるスクリプト変数の詳細は、TLD または (パッケージ `javax.servlet.jsp.tagext` 内の) `TagExtraInfo` 抽象クラスのサブクラスで指定されます。このマニュアルでは、`TagExtraInfo` のサブクラスをタグ補足情報クラスと呼びます。JSP コンテナは、このクラスのインスタンスを変換時に使用します。

- タグ・ライブラリ・バリデータ

タグ・ライブラリ・バリデータ・クラスには、タグ・ライブラリを使用する JSP ページの妥当性チェックを、指定の制約に従って行うためのロジックがあります。

- イベント・リスナー

サーブレット 2.3 のイベント・リスナーは、タグ・ライブラリと併用できます。この機能は、アプリケーションの `web.xml` ファイルでリスナーを宣言する便利な代替方法です。

- タグ・ライブラリの web.xml の使用

Sun 社の Java サブレット仕様では、サブレットの標準デプロイメント・ディスクリプタである web.xml ファイルについて説明しています。JSP アプリケーションは、このファイルを使用して、JSP タグ・ライブラリ・ディスクリプタの場所を指定します。

JSP タグ・ライブラリの場合、web.xml ファイルには、taglib 要素、および taglib-uri と taglib-location の 2 つのサブ要素を含めることができます。

これらの項目については、[第 8 章「JSP タグ・ライブラリ」](#)を参照してください。詳細は、Sun 社の JSP 仕様を参照してください。

OC4J が提供するタグ・ライブラリの詳細は、『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』を参照してください。

## JSP の実行

この項では、オンデマンド変換（JSP ページの初回実行時）、JSP コンテナのロール、サブレット・コンテナおよびエラー処理など、JSP ページの実行方法の概要を説明します。

---

---

**注意：** JSP コンテナという用語は、Sun 社の JSP 仕様 1.1 で最初に使用されたもので、1.1 より前の仕様では JSP エンジンと呼ばれていました。したがって、この 2 つの用語は同じ意味です。

---

---

## JSP コンテナの概要

JSP コンテナは、JSP ページの変換、実行および処理を行い、JSP ページにリクエストを渡すエンティティです。

JSP コンテナの構成は、厳密には各実装によって異なりますが、サブレットまたは複数サブレットの集合で構成されています。したがって、JSP コンテナは、サブレット・コンテナによって実行されます。サブレット・コンテナの概要は、[A-4 ページの「サブレット・コンテナ」](#)を参照してください。

JSP コンテナは、Web サーバーが Java で記述されている場合、Web サーバーに取り込むことができます。そうでない場合は、関連付けた Web サーバーで使用できます。

## JSP の実行モデル

JSP ページの実行モデルには、次の 2 種類があります。

- ほとんどの実装およびほとんどの状態で、JSP コンテナは、ページを実行する直前に、オンデマンドで、つまり、ユーザーがリクエストしたときにページを変換します。
- ただし、開発者がページを事前に変換し、そのページを作業サーブレットとしてデプロイする場合があります。コマンドライン・ツールを使用してページを変換、ロードおよび公開し、ページを実行可能にできます。変換は、クライアントまたはサーバーのいずれかで実行できます。ユーザーが JSP ページをリクエストすると、ページは直接実行されるため、変換は必要ありません。

### オンデマンド変換モデル

JSP ページはオンデマンド変換で実行するのが一般的です。JSP コンテナが取り込まれた Web サーバーから JSP ページがリクエストされると、フロントエンド・サーブレットがインスタンス化されて起動します (Web サーバーが正しく構成されていることが前提です)。このサーブレットは、JSP コンテナのフロントエンドとみなすことができます。OC4J では、`oracle.jsp.runtimev2.JspServlet` です。

`JspServlet` は、必要な場合 (変換したクラスが存在しない場合、または JSP ページ・ソースが更新されている場合)、JSP ページの検索、変換およびコンパイルを行い、そのページの実行をトリガーします。

Web サーバーは、ファイル名の拡張子 `*.jsp` (URL 内) を `JspServlet` にマッピングするために、正しく構成されている必要があります。この構成は、OC4J のインストール時に自動的に処理されます。詳細は、3-13 ページの「[JSP コンテナの設定](#)」で説明します。

### 事前変換モデル

通常のオンデマンド変換とは別の方法として、開発者が JSP ページを事前に変換してからデプロイする場合があります。この事前変換には、次のようなメリットがあります。

- 実行時の変換が不要なため、ユーザーは、初めて JSP ページをリクエストする際に時間を節約できます。
- 専用ソフトウェアであるか、またはセキュリティ上の理由でコードを公開しないようにするために、バイナリ・ファイルのみをデプロイする場合に役立ちます。

詳細は、7-37 ページの「[JSP の事前変換](#)」および 7-40 ページの「[バイナリ・ファイルのみのデプロイ](#)」を参照してください。

Oracle には、JSP ページを事前に変換するための `ojspc` コマンドライン・ユーティリティが用意されています。このユーティリティには、出力ファイル用の適切なベース・ディレクトリを、アプリケーションのデプロイ方法に応じて設定できるオプションがあります。`ojspc` ユーティリティについては、7-12 ページの「[ojspc 事前変換ユーティリティ](#)」で説明します。

## JSP ページとオンデマンド変換

典型的なオンデマンド変換では、JSP ページは、通常、次の手順で実行されます。

1. ユーザーは、ファイル名が `.jsp` で終わる URL により、JSP ページをリクエストします。
2. Web サーバーのサーブレット・コンテナは、URL 内のファイル名の拡張子 `.jsp` を認識すると、すぐに JSP コンテナを起動します。
3. JSP コンテナは、JSP ページが初めてリクエストされると、その JSP ページを検索して変換します。変換処理では、`.java` ファイルにサーブレット・コードが作成され、その `.java` ファイルがコンパイルされて、サーブレットの `.class` ファイルが作成されません。

JSP トランスレータによって生成されたサーブレット・クラスは、`javax.servlet.jsp.HttpJspPage` インタフェースを実装するクラス (JSP コンテナによって提供されます) を拡張します (A-11 ページの「標準の JSP インタフェースとメソッド」で説明します)。このサーブレット・クラスは、「ページ実装クラス」と呼ばれます。このマニュアルでは、ページ実装クラスのインスタンスを「JSP ページ・インスタンス」と呼びます。

JSP ページをサーブレットに変換すると、標準的なサーブレット・プログラミングのオーバーヘッド (`HttpJspPage` インタフェースの実装、サービス・メソッドのコード生成など) が、生成されたサーブレット・コードに自動的に取り込まれます。

4. JSP コンテナは、ページ実装クラスのインスタンス化と実行をトリガーします。

次に、JSP ページ・インスタンスは、HTTP リクエストを処理して HTTP レスポンスを生成し、そのレスポンスをクライアントに返信します。

---

---

**注意：** 前述の手順では、概略を説明しています。前述のように、各ベンダーは JSP コンテナの実装方法を決定しますが、JSP コンテナはサーブレットまたは複数サーブレットの集合で構成されます。たとえば、サーブレットには、JSP ページを検索するフロントエンド・サーブレット、変換とコンパイルを処理する変換サーブレット、各ページ実装クラスによって拡張されるラッパー・サーブレット・クラス (変換されたページは実際には純粋なサーブレットではなく、サーブレット・コンテナで直接実行できないため) などがあります。サーブレット・コンテナは、これらのコンポーネントを実行するために必要です。

---

---

## JSP ページのリクエスト

JSP ページは、URL にアクセスして直接的にリクエストしたり、別の Web ページやサーブレットを使用して間接的にリクエストできます。

### JSP ページの直接的なリクエスト

サーブレットや HTML ページと同様に、ユーザーは、URL により JSP ページを直接リクエストできます。たとえば、次のように、myapp ディレクトリに HelloWorld JSP ページがあり、myapp は、Web サーバーの myapproot コンテキスト・パスにマッピングされていると仮定します。

```
myapp/dir1/HelloWorld.jsp
```

たとえば、次の URL にアクセスしてこのページをリクエストできます。

```
http://host:port/myapproot/dir1/HelloWorld.jsp
```

ユーザーが初めて HelloWorld.jsp をリクエストすると、JSP コンテナは、このページの変換と実行をトリガーします。後続のリクエストでは、JSP コンテナはページの実行のみトリガーし、変換手順は不要となります。

---

---

**注意：** 一般的なサーブレットと JSP の起動の詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

---

---

### JSP ページの間接的なリクエスト

サーブレットと同様に、JSP ページも、通常の HTML ページからリンクしたり、別の JSP ページやサーブレットから参照して、間接的に実行できます。

ある JSP ページを別の JSP ページにある JSP 文から起動する場合のパスは、アプリケーション・ルートに対して相対的なパス（コンテキスト相対パスまたはアプリケーション相対パスと呼ばれます）、または起動ページに対して相対的なパス（ページ相対パスと呼ばれます）のいずれかになります。ページ相対パスと異なり、アプリケーション相対パスは、「/」で始まります。

通常、これらのパスは、URL リンクまたは HTML リンクで使用するパスと同じではありません。前項の例の場合、次のように、HTML リンクで使用するパスは、直接的な URL リクエストで使用するパスと同じです。

```
<a href="/myapp/dir1/HelloWorld.jsp" /a>
```

次に、JSP 文のアプリケーション相対パスを示します。

```
<jsp:include page="/dir1/HelloWorld.jsp" flush="true" />
```

次に、同じディレクトリ内の JSP ページから HelloWorld.jsp を起動するページ相対パスを示します。

```
<jsp:forward page="HelloWorld.jsp" />
```

(jsp:include 文および jsp:forward 文については、1-15 ページの「[標準アクション: JSP タグ](#)」を参照してください。)

---

## Oracle JSP 実装の概要

Oracle Application Server の Oracle Application Server Containers for J2EE (OC4J) が提供する JSP コンテナは、JSP 仕様 1.2 の完全な実装です。この機能は、サーブレット 2.3 の機能に基づいています。また、OC4J サーブレット・コンテナは、サーブレット仕様 2.3 の完全な実装です。

この章では、Oracle Application Server、OC4J、OC4J JSP の実装と機能、および提供されているカスタム・タグ・ライブラリとユーティリティ (『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』を参照) の概要を説明します。

この章には、次の項目が含まれます。

- [Oracle Application Server と JSP サポートの概要](#)
- [Oracle JDeveloper の JSP サポート](#)
- [Oracle の付加価値機能の概要](#)

## Oracle Application Server と JSP サポートの概要

次の各項では、Oracle Application Server、その J2EE 環境、JSP 実装および Web サーバーの概要を説明します。

- [Oracle Application Server の概要](#)
- [Oracle HTTP Server と mod\\_oc4j](#)
- [OC4J の概要](#)
- [OC4J の JSP 実装の概要](#)

---

**注意：** 以前のリリースの Oracle Application Server ユーザーは、現行リリースに移行する際の問題点について、『Oracle Application Server 10g アップグレード 10g (9.0.4)』を参照してください。

---

## Oracle Application Server の概要

Oracle Application Server は、スケーラブルでセキュアな中間層のアプリケーション・サーバーです。このサーバーは、Web コンテンツの配信、Web アプリケーションのホスト管理、およびバックオフィス・アプリケーションへの接続を行い、クライアント・ブラウザによる、これらのサービスへのアクセスを可能にします。ユーザーは、インターネットまたは企業のイントラネットやエクストラネット上で、情報にアクセスしてビジネス分析を実行し、ビジネス・アプリケーションを実行できます。主な機能には、ビジネス・インテリジェンス、E-Business 統合、J2EE Web サービス、パフォーマンスとキャッシング、ポータル、ワイヤレス・サービスおよび管理とセキュリティがあります。パフォーマンス、スケーラビリティおよび信頼性については、クラスタリング機能とロード・バランシング機能もあります。

このような広範囲なコンテンツとサービスを提供するために、Oracle Application Server には、多くのコンポーネントが組み込まれています。たとえば、Oracle HTTP Server、Oracle Application Server Web Cache、Oracle Application Server Web Services、Oracle Application Server Portal、Oracle Application Server Wireless、Oracle Application Server Forms Services と Oracle Application Server Reports Services (Oracle Forms ベースのアプリケーションとレポート生成のサポート用) および Oracle Application Server Personalization が組み込まれています。また、Enterprise JavaBeans、ストアド・プロシージャおよび Oracle Business Components for Java をサポートするビジネス・ロジックの実行時環境が組み込まれています。

Java 環境に対して、Oracle Application Server は、J2EE 1.3 に準拠したコンテナとサービスのセットである、Oracle Application Server Containers for J2EE (OC4J) を提供します。これには、JSP コンテナ (このマニュアルで説明します)、サーブレット・コンテナおよび EJB コンテナが含まれています。



管理については、HTML ベースの Oracle Enterprise Manager を使用して、Oracle Application Server と OC4J を完全に管理および構成できます。この管理と構成には、クラスタリング、構成およびデプロイの管理に対する完全なサポートが含まれます。

---

---

**注意：**

- Oracle Application Server は、現在、Apache JServ サブレット環境（付録 B 「Apache JServ 環境」を参照）も提供しています。
  - 便宜上、移行中の一時的なモードとして、必要に応じて JServ および OC4J で実行中のページとともに、JServ と OC4J を使用できます。この構成の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。
- 
- 

## Oracle HTTP Server と mod\_oc4j

Apache Web サーバーによる Oracle HTTP Server は、特に本番環境では、Web アプリケーションの HTTP エントリ・ポイントとして Oracle Application Server に組み込まれています。デフォルトでは、すべての OC4J プロセスに対するフロントエンドになります。したがって、クライアントのリクエストは、最初に Oracle HTTP Server を通ります。

Oracle HTTP Server を使用すると、動的なコンテンツは、Apache Software Foundation または Oracle のいずれかが提供する各種の Apache mod コンポーネントを通じて配信されます。通常、静的なコンテンツはファイル・システムから配信され、この方法が効率的です。Apache mod は、通常は C コードのモジュールで、Apache アドレス空間で実行され、リクエストを特定 mod に固有のプロセッサに渡します。mod ソフトウェアは、特定のプロセッサで使用するために作成されたものです。

Oracle Application Server は、Oracle HTTP Server と OC4J との間の通信に使用する mod\_oc4j Apache mod を提供します。この mod は、リクエストを Oracle HTTP Server から OC4J プロセスにルーティングし、レスポンスを OC4J プロセスから Web クライアントに転送します。

通信は、Apache JServ プロトコル (AJP) を通じて行われます。AJP は、バイナリ形式の使用やメッセージ・ヘッダーの効率的な処理など、高速通信を可能にする様々な機能を備えているため、HTTP より優先して採用されました。

mod\_oc4j では、次の機能が提供されます。

- 多数のバックエンド OC4J クラスタ間でのロード・バランシング機能
- ステートフル・サブレットのステートレス・セッション・ルーティング  
このルーティングは、Cookie を拡張使用して実行されます。ルーティング情報が Cookie 自体に維持されるため、ステートフル・サブレットは常に同じ OC4J JVM にルーティングされます。

- 高可用性

mod\_oc4j モジュールは、必要に応じて、OC4J インスタンスを自動的に再起動できません。

---

---

**注意：**

- Oracle Application Server には、JServ サブレット環境用に、Apache の mod\_jserv も含まれています。この機能については、[付録 B 「Apache JServ 環境」](#) で説明します。Oracle Application Server が提供する他の Apache mod コンポーネントは、JSP アプリケーションには関係ありません。
  - Oracle HTTP Server は、OC4J スタンドアロン環境には関係ありません（通常は、開発時のみ使用されます）。2-7 ページの「[スタンドアロン OC4J](#)」を参照。
- 
- 

## OC4J の概要

次の各項では、Oracle Application Server の J2EE コンポーネントである OC4J の機能の概要について説明します。

- [OC4J の一般的な機能](#)
- [OC4J のサービス](#)
- [OC4J のコンテナ](#)
- [スタンドアロン OC4J](#)

### OC4J の一般的な機能

OC4J は、高パフォーマンスの、J2EE に準拠したコンテナとサービスのセットで、スケーラブルで信頼性の高いサーバー・インフラストラクチャを提供します。Oracle Application Server 10g (9.0.4) では、OC4J は J2EE 仕様 1.3 に準拠しています。

開発者の便宜のために、OC4J は Oracle JDeveloper および他の開発ツールと統合されたため、開発プロセスでは Oracle Application Server から切り離して、スタンドアロン・モードで実行できます。

開発ツールで作成した Java アプリケーションは、OC4J にデプロイできます。OC4J は、標準の EAR または WAR デプロイをサポートします。OC4J にデプロイされたアプリケーションは、標準的な Java のプロファイル機能とデバッグ機能を使用してデバッグできます。

セキュリティのために、OC4J は Secure Socket Layer (SSL) および HTTPS 機能をサポートしています。

---

---

**注意：** 各 OC4J インスタンスは、単一の Java Virtual Machine (JVM) で実行されます。

---

---

## OC4J のサービス

OC4J では、次の Java および J2EE サービスをサポートしています。

- J2EE Connector Architecture (JCA) : JCA は、J2EE プラットフォームを異機種のエンタープライズ情報システム (ERP システム、メインフレーム・トランザクション処理、データベース・システム、レガシー・アプリケーションなど) に接続するための標準アーキテクチャを定義します。
- Java Transaction API (JTA) と 2 フェーズ・コミット : JTA によって、複数リソースに対する同時更新が 1 つの調整されたトランザクションとして処理できます。
- Java Message Service (JMS) 統合 : この統合によって、Oracle の JMS 実装と他の JMS プロバイダの JMS 実装との間で、互換性を維持できます。
- Java Naming and Directory Interface (JNDI) : JNDI は、ルックアップのために名前をリソースに関連付けます。
- Java Authentication and Authorization Service (JAAS) : JAAS の Oracle 実装と Java2 セキュリティ・モデルは、セキュアなアプリケーションの開発とデプロイ、および詳細な認証とアクセス制御に対する完全なサポートを提供します。
- JDBC データ・ソース : データベースに接続するための標準機能です。

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。

## OC4J のコンテナ

OC4J 9.0.4 実装は、次の J2EE コンテナを提供します。

- JSP コンテナ。Sun 社の JSP 仕様 1.2 に準拠しています。  
JSP バンドルは、Web サービス、キャッシング機能、SQL アクセス、ファイル・アクセスなどの機能を実装するタグ・ライブラリも提供します。OC4J が提供する JSP コンテナの概要は、2-8 ページの「[OC4J の JSP 実装の概要](#)」を参照してください。
- サーブレット・コンテナ。サーブレット仕様 2.3 に準拠しています (主な機能については後述)。
- EJB コンテナ。EJB 仕様 2.0 に準拠しています (主な機能については後述)。

OC4J のコンテナには、実行時のパフォーマンス・データを提供するダイナミック・モニタリング・サービス (Dynamic Monitoring Service: DMS) をサポートする手段が用意されています。このデータは、Enterprise Manager を使用して表示できます。

---

---

**注意：** JSP 仕様 1.2 準拠をサポートするには、サーブレット 2.3 に準拠している必要があります。

---

---

**サーブレット・コンテナの主な機能** OC4J のサーブレット・コンテナは、次の主な機能をサポートします。

- **SSL と HTTPS:** Oracle Application Server では、OC4J は、セキュアな AJP を使用して、Oracle HTTP Server と OC4J 間の SSL (Secure Socket Layer) 通信をサポートします。またスタンドアロンでは、OC4J は HTTPS を使用して、クライアントと OC4J 間の SSL 通信を直接サポートします。
- **SSO と Oracle Internet Directory の統合 :** Oracle JAAS 実装によって行われます。
- **ステートフル・フェイルオーバーとクラスタ・デプロイ :** 配布可能なアプリケーションの場合、フェイルオーバーのときに状態が失われないように、セッション状態は代替 OC4J サーバーに複製されます。
- **サーブレットのフィルタ処理 :** HTTP リクエストまたはレスポンスのコンテンツを変換し、ヘッダー情報を変更できます。
- **アプリケーション・レベルとセッション・レベルのイベント・リスナー :** この機能によって、サーブレット・コンテキストと HTTP セッション・オブジェクト間の相互作用をより強力に制御できるため、アプリケーションが使用するリソースを効率的に管理できます。

詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

**EJB コンテナの主な機能** OC4J の EJB コンテナは、次の機能をサポートします。

- **Session Bean:** Session Bean は、タスク指向のリクエストで使用します。Session Bean は、ステートレスまたはステートフルとして定義できます。ステートレス Session Bean は、コール間の状態情報を維持できません。一方、ステートフル Session Bean は、コール間の状態を維持できます。
- **Entity Bean:** Entity Bean はデータを表します。Entity Bean は、コンテナを使用してデータを永続的に維持できます (コンテナ管理の永続性 (CMP) と呼ばれます)。また、Bean 実装を使用してデータを管理できます (Bean 管理の永続性 (BMP) と呼ばれます)。
- **Message-Driven Bean (MDB) :** Message-Driven Bean は、キューまたはトピックから JMS メッセージを受信するために使用します。次に、他の EJB を起動して、受信した JMS メッセージを処理できます。

OC4J の EJB サポートには、次の機能も含まれます。

- Session Bean と Entity Bean のクラスタリング
- 複数のクライアントからの同時アクセスをサポートする、拡張された Entity Bean の並行性モデル
- Entity Bean 用に拡張されたロック・モデル（コミット時ロック・モード / 即時ロック・モード / 読取り専用モード）
- 業界標準に準拠したインフラストラクチャを提供して、長時間実行するビジネス・トランザクションを調整する Active Components for Java (AC4J)

詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

## スタンドアロン OC4J

本番環境では通常、OC4J は完全な Oracle Application Server 環境で使用されます。この環境には、Oracle HTTP Server (2-3 ページの「[Oracle HTTP Server と mod\\_oc4j](#)」を参照)、OracleAS Web Cache および Enterprise Manager が含まれます。

開発環境では、OC4J はスタンドアロン・コンポーネントとしても使用できます。この場合は、OTN-J (Oracle Technology Network Japan) (<http://otn.oracle.co.jp/products/oc4j/index.html>) から OC4J\_extended.zip をダウンロードします。

スタンドアロン OC4J を使用する場合は、ポート 8888 を介して独自の HTTP Web リスナーを使用できます。スタンドアロン OC4J の詳細は、『Oracle Application Server Containers for J2EE Stand Alone User's Guide』(OC4J\_extended.zip とともにダウンロード可能) および『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

---

---

**注意：** スタンドアロン OC4J を使用するには、Sun 社の JDK (サポートされているバージョン) をインストールする必要があります。JDK は、スタンドアロン OC4J 製品には同梱されていません。

---

---

## OC4J の JSP 実装の概要

Oracle Application Server 10g (9.0.4) の JSP コンテナは、JSP 仕様 1.2 に準拠しています。

通常、JSP 1.2 環境では、OC4J サブレット・コンテナなどのサブレット 2.3 環境が必要です。ただし、OC4J JSP 実装は、Apache JServ、サブレット 2.0 環境での実行もサポートします。このサポートを可能にするために、OC4J JSP コンテナは、2.0 の仕様を超える必須のサブレット機能をエミュレートします。

ただし、通常は様々な理由から、OC4J サブレット 2.3 環境を使用することをお勧めしません。

これらの機能については、次の各項で説明します。

- [JSP コンテナの履歴と統合](#)
- [JSP フロントエンド・サブレットと構成](#)
- [JSP 1.2 に関する OC4J JSP の機能](#)
- [OC4J の構成可能な JSP 拡張機能](#)
- [サブレット環境間の移植性](#)

### JSP コンテナの履歴と統合

OC4J を含む最初のリリースである Oracle9iAS リリース 1.0.2.2 には、1) オラクル社が開発した「OracleJSP」というコンテナ、2) Ironflare AB 社からライセンスを受けた「Orion JSP コンテナ」というコンテナ、の 2 種類の JSP コンテナがありました。

OracleJSP コンテナには、いくつかのメリットがあり、グローバリゼーションなどをサポートするための便利な付加価値機能と拡張機能が含まれていました。Orion コンテナにも、処理速度が速いなどのメリットがありましたが、デメリットもありました。JSP リファレンス実装 (Tomcat) と比較して標準動作が安定せず、国際化およびグローバリゼーションに対するサポートも十分ではありませんでした。

Oracle9iAS リリース 2 (9.0.2) で初めて、OracleJSP コンテナと Orion コンテナが、このマニュアルで「OC4J JSP コンテナ」と呼ばれる単一の JSP コンテナに統合されました。このコンテナは、前の 2 つのバージョンの長所を備え、OC4J サブレット・コンテナ内でサブレットとして効率的に実行され、他の OC4J のコンテナとも統合されます。統合されたコンテナは、主に OracleJSP トランスレータと Orion コンテナ・ランタイムで構成され、簡素化されたディスパッチャと OC4J コア・ランタイム・クラスとともに実行されます。

## JSP フロントエンド・サーブレットと構成

OC4J JSP コンテナは、フロントエンド・サーブレットの `oracle.jsp.runtime.v2.JspServlet` を使用します。3-13 ページの「[OC4J での JSP 構成](#)」を参照。

OC4J 以外の環境（Apache JServ など）では、以前のフロントエンド・サーブレットの `oracle.jsp.JspServlet` を使用します。B-2 ページの「[JServ 環境での作業開始](#)」を参照。

## JSP 1.2 に関する OC4J JSP の機能

OC4J 9.0.4 の実装では、OC4J JSP コンテナは、JSP 仕様 1.2 に完全に準拠しています。この仕様で追加された機能のほとんどは、カスタム・タグ・ライブラリの領域にあります。

- タグ・ライブラリ機能：
  - ボディ・コンテンツ・オブジェクトをメンテナンスしたり、アクセスすることなく、タグ・ボディを反復できるタグ・ハンドラ・インタフェースがあります。
  - タグ・ライブラリ・バリデータ・クラスを作成し、タグ・ライブラリに関連付けることができます。バリデータ・インスタンスは、ライブラリを使用する JSP ページをチェックして、必要な制約に一致していることを確認します。
  - サーブレットのイベント・リスナーを宣言できるのは、便宜上、`web.xml` ファイル内ではなく、タグ・ライブラリ・ディスクリプタ内です。これによって、タグ・ライブラリの使用に関連付けられているアプリケーションとセッションのリソースを簡単に管理できます。
  - 複数のタグ・ライブラリとそれらの TLD は、単一の JAR ファイルにパッケージングできます。

タグ・ライブラリの機能の詳細は、[第 8 章「JSP タグ・ライブラリ」](#)を参照してください。詳細なサマリーは、[8-4 ページの「JSP 仕様 1.1 と 1.2 の間のタグ・ライブラリ変更の概要」](#)を参照してください。

- XML の機能：
  - 以前の OC4J JSP コンテナは、標準の XML 代替構文をサポートしていましたが、現在の JSP 仕様に従って、新しいテクノロジーに置換されています。
  - OC4J JSP コンテナは、変換されたすべてのページに対して XML ビューを生成します。この XML ビューは、ページを記述する XML 文書へのマッピングです。このビューは、タグ・ライブラリ・バリデータ・クラスで使用できます。

これらの機能の詳細は、[第 5 章「JSP XML サポート」](#)を参照してください。

- 文字エンコード機能：

OC4J JSP 実装では、page ディレクティブの `pageEncoding` 属性をサポートしています。このサポートによって、(`contentType` 属性で指定される) レスポンスの文字エンコードとは異なるページ・ソースに対して文字エンコードを指定できます。

9-2 ページの「[コンテンツ・タイプの設定](#)」を参照。

## OC4J の構成可能な JSP 拡張機能

JSP 仕様 1.2 準拠に加え、Oracle Application Server 10g (9.0.4) の OC4J JSP コンテナには、注目すべき次の機能があります。

2-13 ページの「[Oracle の付加価値機能の概要](#)」も参照してください。

OC4J 9.0.3 の実装以降に、次の機能がサポートされています。

- 単一の JSP 変換単位内で同じディレクティブ属性の設定が重複している場合に、JSP 変換エラーを回避するためのモード切替え (OC4J または JServ)

JSP 仕様では、単一の JSP 変換単位内で同じディレクティブ属性の設定が重複している場合 (`page` ディレクティブの `import` 属性を除く) に、変換エラーが指示されます。たとえば、ページとインクルード・ファイルの両方で、ある属性に同じ値を設定している場合 (`language="java"` など) は、このエラーは不適切です。

`forgive_dup_dir_attr` パラメータについては、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

- `web.xml` ファイルと TLD の XML 妥当性チェック用の個別のモード切替え (OC4J または JServ)

`web.xml` の妥当性チェックは、デフォルトでは無効ですが、有効にできます。TLD の妥当性チェックは、デフォルトでは有効ですが、無効にできます。

`xml_validate` パラメータと `no_tld_xml_validate` パラメータについては、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

- 追加インポート用のモード・フラグ (OC4J または JServ)

このフラグを使用して、JSP のデフォルト設定よりも多い Java パッケージを自動的にインポートします。

`extra_imports` パラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

- タグ・ライブラリを共有するための予約済の場所 (OC4J または JServ)

複数の Web アプリケーションで共有するタグ・ライブラリ JAR ファイルを配置するディレクトリを指定できます。

`well_known_taglib_loc` パラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。



- 構成可能な JSP タイムアウト

JSP ページに対してタイムアウト値を指定できます。この値を経過した後に再度リクエストされなかった場合、そのページはメモリーから削除されます。jsp-timeout パラメータについては、3-28 ページの「[JSP 関連の OC4J 構成パラメータに関する説明](#)」を参照してください。

OC4J 9.0.2 の実装以降に、次の機能がサポートされています。

- ページの自動再変換と自動再ロードのためのモード切替え (OC4J のみ)

選択できるモードは、1) 自動再ロードまたは JSP ページの自動再変換を行わずに JSP ページを実行する、2) ページ実装クラス (JavaBeans クラスまたは他の依存クラスを除く) を自動的に再ロードする、3) 変更された JSP ページを自動的に再変換する、のいずれかです。

main\_mode パラメータについては、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

- タグ・ハンドラ・インスタンスのプーリング (OC4J または JServ)

タグ・ハンドラの作成とガベージ・コレクションの時間を節約するために、タグ・ハンドラ・インスタンスのプーリングを必要に応じて有効にできます。タグ・ハンドラ・インスタンスは、application スコープにプールされます。ページごとに、または同じページのセクションごとに、異なる設定を使用できます。8-38 ページの「[実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)」を参照。

- NULL 出力に対する出力モード (OC4J のみ)

JSP ページからの NULL 出力に対して、デフォルトの NULL 文字列のかわりに空の文字列を出力できます。

jsp-print-null パラメータについては、3-28 ページの「[JSP 関連の OC4J 構成パラメータに関する説明](#)」を参照してください。

- シングル・スレッド・モデルの JSP インスタンスのプーリング (OC4J のみ)

シングル・スレッド (スレッド・セーフでない) の JSP ページの場合、ページ・インスタンスはプールされます。この機能には切替えがありません。常に有効です。

## サーブレット環境間の移植性

JSP コンテナは、OC4J のコンポーネントとして提供されていますが、他の環境への移植が可能です。OC4J JSP コンテナ自体が必須のサーブレット機能をエミュレートするため、この移植性は以前のサーブレット環境 (特に Apache JServ サーブレット 2.0 環境) に拡張されます (通常、JSP 仕様 1.2 準拠をサポートするにはサーブレット 2.3 環境が必要です)。

サーブレット仕様 2.0 では、アプリケーションごとにサーブレット・コンテキストが提供されるのではなく、Java Virtual Machine ごとに 1 つのサーブレット・コンテキストが提供されるように制限されていました。OC4J JSP サーブレットのエミュレーション機能によって、サーブレット 2.0 環境でも完全なアプリケーション・フレームワークを実現でき、アプリ

セッションに個別のサーブレット・コンテキスト・オブジェクトと HTTP セッション・オブジェクトが提供されます。

この拡張機能によって、OC4J JSP コンテナは、基礎となるサーブレット環境に制限されることはありません。

## Oracle JDeveloper の JSP サポート

現在の Visual Java プログラミング・ツールは、JSP コーディングをサポートしています。特に、Oracle JDeveloper は、JSP 開発をサポートし、次の機能を備えています。

- OC4J JSP コンテナの統合。アプリケーション開発サイクル（JSP ページの編集、デバッグおよび実行）を完全にサポートします。
- デプロイされた JSP ページのデバッグ。
- JSP 要素ウィザード。事前定義済 Web Bean をページに追加する便利な方法です。
- カスタム JavaBeans の取込みのサポート。
- JSP アプリケーションのデプロイ・オプション。JDeveloper Business Components for Java(BC4J) に依存しています

JSP デプロイのサポートの詳細は、7-36 ページの「[Oracle JDeveloper によるアプリケーションのデプロイ](#)」を参照してください。

デバッグでは、JDeveloper によって JSP ページ内にブレーク・ポイントを設定し、JSP ページから JavaBeans へのコールを追跡できます。これは、手動によるデバッグと比較して大変便利な方法です。たとえば、JSP ページに出力文を追加し、状態をレスポンス・ストリーム（ブラウザ表示用）やサーバー・ログ（暗黙的な application オブジェクトの log() メソッドを使用）に出力できます。

JDeveloper については、JDeveloper のオンライン・ヘルプ、または次の OTN-J (Oracle Technology Network Japan) のサイトを参照してください。

<http://otn.oracle.co.jp/products/jdev/index.html>

(この Web サイトを参照するには、OTN-J に登録する必要があります。無償で登録できます。) JDeveloper が提供する JSP タグ・ライブラリの概要は、『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』を参照してください。

---

---

**注意：** 他の主要な IDE ベンダーが、OC4J とシームレスに統合できるプラグイン・モジュールを構築しています。これにより、開発者は、IDE 内から直接、OC4J で実行中の J2EE アプリケーションをビルド、デプロイおよびデバッグできます。詳細は、次の Web サイトを参照してください。

[http://otn.oracle.com/products/ias/9ias\\_partners.html](http://otn.oracle.com/products/ias/9ias_partners.html)

---

---

## Oracle の付加価値機能の概要

JSP ページに関する OC4J の付加価値機能は、次の 3 つのカテゴリにグループ化できます。

- カスタム・タグ・ライブラリ、カスタム JavaBeans またはカスタム・クラスの実装によって提供される機能。通常、他の JSP 環境に移植できます。
- Oracle 固有の機能。
- キャッシング・テクノロジーをサポートする機能。

次の各項では、これらのカテゴリの機能の概要、および JavaServer Pages 標準タグ・ライブラリ (JSTL) に対する Oracle サポートの概要について説明します。JSTL サポートの詳細は、『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』を参照してください。

## OC4J が提供するタグ・ライブラリとユーティリティの概要

この項では、標準準拠のカスタム・タグ・ライブラリ、カスタム JavaBeans およびその他のクラスを介して実装される、OC4J JSP 拡張機能の概要について説明します。これらの機能の詳細は、『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』を参照してください。

- スコープを指定できる JavaBeans として実装された拡張型。
- イベント処理用の `JspScopeListener`。
- XML と XSL の統合。
- データ・アクセス用タグ・ライブラリ (SQL タグとも呼ばれる) と JavaBeans。
- JSP Markup Language (JML) のカスタム・タグ・ライブラリ。JSP 開発に必要な Java に関する知識レベルが軽減されます。
- Oracle Personalization のタグ・ライブラリ。
- Web サービス用タグ・ライブラリ。
- タグ・ライブラリと JavaBeans。ファイルのアップロード、ファイルのダウンロード、アプリケーション内からの電子メール送信に利用します。
- EJB タグ・ライブラリ
- その他のユーティリティ・タグ (指定したロケールに適した日付や通貨を表示するタグなど)。

---

**注意：** キャッシングに使用するその他のタグ・ライブラリの概要は、2-16 ページの「キャッシング・サポートに関するタグと API の概要」を参照してください。

---

## Oracle 固有の機能の概要

この項では、OC4J JSP コンテナがサポートする Oracle 固有のプログラミング拡張機能の概要を説明します。

- SQLJ のサポート。SQLJ は、SQL 文を Java コードに直接埋め込むための標準構文です。
- グローバル・インクルード。1 つまたは複数のファイルを、複数のページに静的に、かつ自動的にインクルードする機能です。
- パフォーマンス測定に対するダイナミック・モニタリング・サービス (DMS) のサポート。
- サブレット 2.0 環境用の拡張アプリケーション・フレームワークとグローバリゼーション・サポート。

---

---

**重要：** SQLJ がサポートされるのは、現行リリースまでです。

---

---

### SQLJ サポート

動的なサーバー・ページには、一般的に、データベースから抽出されたデータが含まれません。JSP 開発者は通常、標準の Java Database Connectivity (JDBC) API またはデータベース用 JavaBeans のカスタム・セットを使用します。

SQLJ は、静的な SQL の命令を Java コードに直接埋め込むための標準構文です。これによって、データベース・アクセスのプログラミングが大幅に簡素化されます。OC4J JSP コンテナは、JSP スクリプトレットで SQLJ プログラミングをサポートします。

SQLJ 文は、`#sql` トークンで示されます。JSP ソース・コード・ファイルのファイル名拡張子 `.sqljsp` を使用するか、または `page` ディレクティブに `language="sqlj"` を指定することで、JSP トランスレータをトリガーして Oracle SQLJ トランスレータを起動できます。

詳細は、4-14 ページの「[Oracle SQLJ の JSP サポート](#)」を参照してください。

### グローバル・インクルード

OC4J JSP コンテナは、グローバル・インクルードと呼ばれる機能を提供しています。この機能によって、仮想の JSP `include` ディレクティブを使用して、指定のディレクトリ内またはディレクトリ下の JSP ページに静的にインクルードする 1 つ以上のファイルを指定できます。変換時に、JSP コンテナは、インクルード・ファイルとディレクトリをページに指定する構成ファイル `/WEB-INF/ojsp-global-include.xml` を検索します。

この拡張機能は、以前の Oracle JSP リリースで `globals.jsa` または `translate_params` 機能を使用していたアプリケーションを移行する場合に、特に便利です。詳細は、7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照してください。

## ダイナミック・モニタリング・サービスのサポート

DMS によって、OC4J も含めた多数の Oracle Application Server コンポーネントにパフォーマンス監視機能が追加されます。DMS の目的は、組込みのパフォーマンス測定値を介して、実行時の動作に関する情報を提供することにあります。これによって、ユーザーは、パフォーマンスに関する問題を診断、分析およびデバッグできます。DMS は、この情報を、デプロイ中も含めていつでも使用できるパッケージで提供します。データは HTTP を通じて公開され、ブラウザで表示できます。

OC4J JSP コンテナは DMS 機能をサポートし、関連する統計値を計算し、スパイ・サーブレットなどの DMS サブレットに情報を提供して、エージェントを監視します。統計には、次の内容（必要に応じて、平均、最大および最小を使用）が含まれます。時間はミリ秒単位で示されます。

- HTTP リクエストの処理時間
- JSP サービス・メソッドの処理時間
- 作成済または使用可能な JSP インスタンスの数
- アクティブな JSP インスタンスの数

(JSP インスタンスの数は、page ディレクティブの `isThreadSafe` が `false` に設定されたシングル・スレッド状態の場合のみカウントされます。)

これらのサブレットの標準構成は、OC4J の `application.xml` および `default-web-site.xml` 構成ファイルにあります。Enterprise Manager を使用して DMS にアクセスし、DMS 情報を表示して、必要に応じて DMS 構成を変更します。

JSP メトリックの正確な定義と、JSP メトリックを表示および分析する詳細な手順については、『Oracle Application Server 10g パフォーマンス・ガイド』を参照してください。

## 拡張されたサーブレット 2.0 のサポート

OC4J は、サーブレット 2.0 JServ 環境用の特別な機能をサポートします。できるだけ早く OC4J サブレット 2.3 環境に移行することをお勧めしますが、しばらくの間は、次の点に注意してください。

- サーブレット 2.0 環境用の拡張されたアプリケーション・フレームワーク  
B-32 ページの「[JServ に対する JSP アプリケーションとセッションのサポート](#)」を参照。
- サーブレット 2.0 環境用の拡張されたグローバリゼーション・サポート  
B-24 ページの「[JServ でのマルチバイト・パラメータ・エンコード](#)」を参照。

参照先の各項には、OC4J に移行するための情報が含まれています。

## キャッシング・サポートに関するタグと API の概要

Web でのパフォーマンスの問題に対応するために、E-Business では、コスト効率の高いテクノロジーとサービスを導入して、インターネット・サイトのパフォーマンスを改善する必要があります。Webキャッシングは、この問題を解決するための主要なテクノロジーで、静的および動的な Web コンテンツをキャッシングします。Webキャッシングの利点には、パフォーマンス、スケーラビリティ、高可用性、コスト削減およびネットワークの通信量の軽減があります。

OC4J は、Webキャッシング・テクノロジーに対して、次のサポートを提供します。

- Edge Side Includes (ESI) 用の JESI タグ・ライブラリ。XML スタイルのマークアップ言語で、動的なコンテンツを Web サーバーから独立してアセンブリできます。

OracleAS Web Cache は、ESI エンジンを用意しています。

- Web Object Cache 用のタグ・ライブラリとサーブレット API。アプリケーション・レベルのキャッシュで、Java Web アプリケーション内に埋め込まれて維持されます。

Web Object Cache は、Oracle Application Server の Java Object Cache をデフォルト・リポジトリとして使用します。

これらの機能の詳細は、『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』を参照してください。

## JavaServer Pages 標準タグ・ライブラリ (JSTL) のサポート

Oracle Application Server 10g (9.0.4) では、OC4J JSP 製品は、Sun 社の JSTL 仕様 1.0 で指定されているとおり、JavaServer Pages 標準タグ・ライブラリ (JSTL) をサポートしています。

JSTL は、Java などのスクリプト言語に不慣れな JSP ページの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、スクリプトレットが使用されていました。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。

JSTL の主な機能は、次のとおりです。

- JSTL の式言語 (Expression Language: 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、リクエスト時の属性とスクリプトレットが不要になります。

- 式言語サポート、条件付きロジックとフローの制御、イテレータ操作および URL ベース・リソースへのアクセス用のコア・タグ
- XML 処理、フロー制御および XSLT 変換用のタグ
- データベース・アクセス用の SQL タグ
- I18N 対応の国際化および書式設定用のタグ

「I18N」は、国際化標準を指します。

タグのサポートは、前述の機能に基づいて 4 つの JSTL サブライブラリに編成されています。

JSTL サポートの詳細は、『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』を参照してください。JSTL の詳細は、次の Web サイト上の仕様を参照してください。

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

---

---

**注意：** OC4J が提供する 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 JavaServer Pages 標準タグ・ライブラリ (JSTL) 以前のライブラリで、重複した機能があります。標準に準拠するように操作を進めるには、カスタム・ライブラリではなく、原則として JSTL を使用することをお薦めします。ただし、Oracle では既存のタグも継続してサポートされています。JSTL ではまだ使用できないカスタム・ライブラリの機能のうち、有用と考えられる機能については、必要に応じて JSTL 標準に採用される予定です。

---

---





---

---

## スタート・ガイド

この章では、JSP 環境での基本事項である、主なサポート・ファイル、主な OC4J 構成ファイルおよび JSP コンテナの構成について説明します。また、アプリケーション・ルート機能、クラスパス機能、セキュリティ上の問題およびファイルのネーミング規則など、初期段階での考慮事項についても説明します。

作業の開始に当たっては、次の内容をシステム上で実行できることが前提となります。

- Java の実行
- Java コンパイラの実行（通常は、標準の javac）
- HTTP サーブレットの実行

この章には、次の項目が含まれます。

- [初期段階での考慮事項](#)
- [OC4J が提供する主なサポート・ファイル](#)
- [OC4J での JSP 構成](#)
- [主な OC4J 構成ファイル](#)
- [Oracle Enterprise Manager での JSP の構成](#)

---

---

**注意：** JSP ページは、HTTP 1.0 以上をサポートする標準ブラウザを使用して実行します。JSP ページ内のすべての Java コードは Web サーバーで実行されるため、ユーザーの Web ブラウザの JDK または Java 環境は関係ありません。

---

---

## 初期段階での考慮事項

次の各項では、JSP ページのコーディングまたは使用を開始する前に考慮しておく必要がある事項について説明します。

- アプリケーション・ルート機能
- クラスパスの機能
- 実行時の再変換または再ロード
- JSP のコンパイルに関する考慮事項
- JSP のセキュリティに関する考慮事項
- JSP のパフォーマンスに関する考慮事項
- デフォルト・パッケージのインポート
- JSP ファイルのネーミング規則
- JDK 1.4 に関する考慮事項: パッケージに含まれないクラスを起動できない

## アプリケーション・ルート機能

サーブレット仕様 (サーブレット 2.2 以降) に従って、各 Web アプリケーションには独自のサーブレット・コンテキストがあります。各サーブレット・コンテキストは、サーバーのファイル・システム内でディレクトリ・パスに関連付けられています。このディレクトリ・パスは、Web アプリケーションのモジュール用のベース・パスです。このベース・パスがアプリケーション・ルートです。各 Web アプリケーションには、独自のアプリケーション・ルートがあります。標準のサーブレット環境の Web アプリケーションの場合、サーブレット、JSP ページ、および HTML ファイルなどの静的なファイルは、すべてこのアプリケーション・ルートに基づいています (これに対して、サーブレット 2.0 環境では、サーブレットと JSP ページのアプリケーション・ルートと、静的なファイルのドキュメント・ルートは異なります)。

サーブレット URL には、次の汎用的なフォームがあります。

```
http://host:port/contextpath/servletpath
```

サーブレット・コンテキストが作成されると、アプリケーション・ルートと、URL のコンテキスト・パス部分との間にマッピングが指定されます。サーブレット・パスは、アプリケーションの `web.xml` ファイルに定義されます。`web.xml` 内の `<servlet>` 要素によって、サーブレット・クラスがサーブレット名に関連付けられます。`web.xml` 内の `<servlet-mapping>` 要素によって、URL パターンが指定のサーブレットに関連付けられます。サーブレットが実行されると、サーブレット・コンテナは、指定された URL パターンを既知のサーブレット・パスと比較し、一致したサーブレット・パスを選択します。詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

たとえば、アプリケーション・ルートが `/home/dir/mybankapp/mybankwebapp` のアプリケーションが、コンテキスト・パス `/mybank` にマッピングされると仮定します。さらに、このアプリケーションには、サーブレット・パスが `loginservlet` のサーブレットが含まれると仮定します。このサーブレットは、次のように起動できます。

```
http://host:port/mybank/loginservlet
```

アプリケーション・ルートのディレクトリ名はユーザーには表示されません。

このアプリケーションの HTML ページについてこの例を使用すると、次の URL によって、`/home/dir/mybankapp/mybankwebapp/dir1/abc.html` ファイルがポイントされます。

```
http://host:port/mybank/dir1/abc.html
```

各サーブレット環境には、デフォルトのサーブレット・コンテキストもあります。このコンテキストのコンテキスト・パスは、デフォルトのサーブレット・コンテキストのアプリケーション・ルートにマッピングされている「/」のみです。たとえば、デフォルトのコンテキストのアプリケーション・ルートが `/home/dir/defaultapp/defaultwebapp` で、サーブレット・パスが `myservlet` のサーブレットでデフォルトのコンテキストを使用すると仮定します。その場合の URL は、次のとおりです。

```
http://host:port/myservlet
```

デフォルトのコンテキストは、URL に指定されているコンテキスト・パスと一致しない場合にも使用されます。

HTML ファイルについてこの例を使用すると、次の URL によって `/home/dir/defaultapp/defaultwebapp/dir2/def.html` ファイルがポイントされます。

```
http://host:port/dir2/def.html
```

## クラスパスの機能

JSP コンテナは、Web サーバー上の標準の場所を使用して、変換済 JSP ページおよび必須クラス (JavaBeans など) 用の .class ファイルと .jar ファイルを検索します。コンテナは、Web サーバーのクラスパス構成を使用せずに標準の場所でファイルを検索します。

従属クラスの場所は次のとおりです。これらの場所はアプリケーション・ルートと相対的な関係があります。

```
/WEB-INF/classes/...  
/WEB-INF/lib
```

JSP ページ実装クラス (変換済ページ) の場所は、次のとおりです。

```
.../_pages/...
```

/WEB-INF/classes ディレクトリは、Java の各 .class ファイル用です。これらのクラスは、Java パッケージのネーミング規則に従って、classes ディレクトリ下のサブディレクトリに格納する必要があります。たとえば、oracle.jsp.sample.lottery パッケージに含まれるように定義した、LottoBean と呼ばれる JavaBean があると仮定します。JSP コンテナは、アプリケーション・ルートに対して相対的な次の場所で、LottoBean.class を検索します。

```
/WEB-INF/classes/oracle/jsp/sample/lottery/LottoBean.class
```

lib ディレクトリは、JAR (.jar) ファイル用です。Java パッケージ構造は JAR ファイル構造に指定されているため、すべての JAR ファイルは、サブディレクトリ内ではなく、lib ディレクトリ内に直接格納されます。たとえば、LottoBean.class は、アプリケーション・ルートに対して相対的な次の場所にある lottery.jar に格納されます。

```
/WEB-INF/lib/lottery.jar
```

\_pages ディレクトリは、OC4J の J2EE ホーム・ディレクトリの下にあり、jsp-cache-directory 構成パラメータの値によって決まります。詳細は、7-7 ページの「[JSP トランスレータの出力ファイルの格納場所](#)」を参照してください。

---

---

**重要：** \_pages ディレクトリのデフォルトの場所など、実装に関する詳細は、今後のリリースで変更される場合があります。

---

---

## 実行時の再変換または再ロード

実行時、JSP ページの再変換または JSP ページ実装クラスのリロードは、JSP `main_mode` 構成パラメータによって制御されます。設定できる値は、変更された JSP ページを再変換する `recompile` (デフォルト)、JSP コンテナによって生成されて変更されたクラス (ページ実装クラスなど) を再ロードする `reload`、または本番環境で最適なパフォーマンスを実現するためにタイムスタンプをチェックせずに実行する `justrun` です。追加情報は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

---

---

### 注意：

- ここで説明する内容は、事前の変換には関係ありません。
  - OC4J JSP コンテナには、独自のクラス・ローダーはありません。
  - メモリー内の値を使用して、クラス・ファイルの最終変更時間を保持するため、ページ実装クラス・ファイルをファイル・システムから削除しても、関連する JSP ページ・ソースは自動的に再変換されません。
  - ページ実装クラス・ファイルは、メモリー・キャッシュが失われると再生成されます。この再生成は、サーバーの再起動後やこのアプリケーション内の別のページが再変換された後に、リクエストがこのページにダイレクトされるたびに発生します。
  - OC4J では、静的にインクルードされたページ (つまり、`include` ディレクティブを使用してインクルードされたページ) が更新された場合、インクルード先のページは、次の起動時に自動的に再変換されます (これは、JServ では実行されません)。
- 
- 

サブレット・レイヤーでのクラス・ローダーの動作の詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

## JSP のコンパイルに関する考慮事項

Java コンパイルには、OC4J と同じプロセス内で実行するインプロセス・コンパイルと、別のプロセスで実行するアウトプロセス・コンパイルがあります。

デフォルトでは、OC4J 全体でアウトプロセス・コンパイルを使用し、コンパイラは個別の実行可能ファイルとして起動します。デフォルトのコンパイラ実行可能ファイルは Sun 社の JDK で提供される `javac` ですが、別のコンパイラを使用するように OC4J を構成することもできます。スタンドアロン OC4J で別のコンパイラを使用するには、`<java-compiler>` 要素を任意のオプション設定で OC4J の `server.xml` ファイルに追加します。Oracle Application Server 環境では、Oracle Enterprise Manager を使用して構成を変更します。

ただし、JSP のパフォーマンスを改善するため、OC4J JSP コンテナではデフォルトでインプロセス・コンパイルを使用します。この場合、Sun 社の JDK の `tools.jar` ファイルがインストール済で、クラスパスにあることが前提です。インプロセス・コンパイルを使用すると、コンパイラのクラスが直接起動します。`tools.jar` がクラスパスにあることを確認するには、`server.xml` ファイルの `<library>` 要素を使用します。

また、関連する JSP 構成パラメータとして、`use_old_compiler` パラメータと `javaccmd` パラメータがあります。

- `use_old_compiler` を `false` に設定すると、JSP コンテナに対して残りの OC4J と同じコンパイラを使用するように強制できます。コンパイルには、`javac` を使用するアウトプロセス・コンパイル（デフォルト）と、`server.xml` ファイルの `<java-compiler>` 要素に基づくコンパイルがあります。`tools.jar` がクラスパスにある場合、`use_old_compiler` フラグはデフォルトで `true` に設定されます。これによって、`javaccmd` が設定されている場合を除いて、インプロセス・コンパイルが使用されます。
- アウトプロセス・コンパイラを使用する場合、残りの OC4J が使用するコンパイラとは異なるコンパイラを使用するには、`use_old_compiler` を `true` に設定し、`javaccmd` パラメータを使用して任意のコンパイラを指定します（`use_old_compiler` が `false` に設定されている場合、`javaccmd` パラメータは無視されます）。

---

---

### 注意：

- `tools.jar` がクラスパスにない場合、`use_old_compiler` は強制的に `false` に設定されます。
  - `use_old_compiler` パラメータおよび `javaccmd` パラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。
  - `server.xml` ファイルの要素の詳細は、『[Oracle Application Server Containers for J2EE ユーザーズ・ガイド](#)』を参照してください。
- 
-

## JSP のセキュリティに関する考慮事項

アプリケーションのセキュリティに関しては、次の点に注意してください。

- 存在しない JSP ファイルがリクエストされたときにファイルの物理的なパスを表示しない場合は、`debug_mode` パラメータがデフォルトの `false` に設定されていることを確認してください。このパラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。
- JServ 環境の場合は、さらに考慮事項があります。B-24 ページの「[JServ での JSP のセキュリティに関する考慮事項](#)」を参照。

## JSP のパフォーマンスに関する考慮事項

次の各項では、パフォーマンスの最適化と監視に関する JSP、OC4J および Oracle Application Server の機能について説明します。

- [最適化に関するプログラム上の考慮事項](#)
- [構成の最適化](#)
- [事前変換の ojspc ユーティリティ](#)
- [OC4J と Oracle Application Server のその他のパフォーマンス機能](#)

### 最適化に関するプログラム上の考慮事項

JSP ページを作成する際は、次の点を考慮してください。

- JSP ページのバッファを無効にします。デフォルトでは、JSP ページはページ・バッファと呼ばれるメモリー領域を使用します。このバッファ（デフォルトは 8 KB）は、動的なグローバリゼーション・サポートのコンテンツ・タイプの設定、転送またはエラー・ページをページで使用する場合に必要です。このような機能をページで使用しない場合は、`page` ディレクティブのバッファを無効にできます。

```
<%@ page buffer="none" %>
```

これによって、メモリーの使用量が減少し、バッファをコピーする出力手順が不要になるため、パフォーマンスが改善されます。

- HTTP セッション・オブジェクトが不要な場合は、使用しないでください。JSP ページで HTTP セッションが不要な場合（基本的に、セッション属性の格納または取得が不要な場合）は、セッションを使用しないように指定できます。`page` ディレクティブで、次のように指定します。

```
<%@ page session="false" %>
```

これによって、セッションの作成または取得のオーバーヘッドが減少するため、ページのパフォーマンスが改善されます。

デフォルトでは、サーブレットはセッションを使用しませんが、JSP ページはセッションを使用します。バックグラウンド情報は、A-5 ページの「[サーブレット・セッション](#)」を参照してください。

- OracleAS Web Cache や Java Object Cache など、一般的な Oracle Application Server のキャッシング機能以外に、JSP ページに固有で、OC4J が提供するカスタム・タグ・ライブラリから使用可能なキャッシング機能があります。JESI タグ・ライブラリおよび Web Object Cache の概要は、2-16 ページの「[キャッシング・サポートに関するタグと API の概要](#)」を参照してください。追加情報は、『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』を参照してください。

### 構成の最適化

パフォーマンスに影響する、JSP および OC4J の構成パラメータが多数存在します。次に示します。

- 本番環境では JSP ページが変更されないため、タイムスタンプをチェックしないように JSP コンテナを構成する必要があります（ページに再変換が必要かどうかを確認する場合は、タイムスタンプをチェックします）。Oracle JSP 構成パラメータ `main_mode` を値 `justrun` に設定することで、この指定が可能です。このパラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。
- 各 JSP ページ内でタグ・ハンドラ・インスタンスを再利用するように指定することで、タグ・ライブラリのパフォーマンスを改善できます。特に、多数のカスタム・タグがある JSP ページについて最適な結果を得るためには、タグ・ハンドラ再利用のロジックおよびパターンを、実行時ではなく変換時に決定するように指定します。これは、Oracle JSP 構成パラメータ `tags_reuse_default` で指定できます。8-38 ページの「[実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)」を参照。
- Oracle JSP の追加構成パラメータの多くがパフォーマンスに影響し、パフォーマンスが向上または低下します。`check_page_scope`、`precompile_check`、`reduce_tag_code` および `static_text_in_chars` の詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。
- OC4J 9.0.4 以上の実装には、タグ・ライブラリをより効率的に使用するための機能が追加されました。主な機能は、タグ・ライブラリ・ディスクリプタ (TLD) の永続的なキャッシングです。この機能は、OC4J 構成パラメータ `jsp-cache-tlds` で有効にできます。8-19 ページの「[タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング](#)」を参照。
- OC4J 構成パラメータ `simple-jsp-mapping` および `enable-jsp-dispatcher-shortcut` は、パフォーマンスに大きく影響します。これらのパラメータの詳細は、3-28 ページの「[JSP 関連の OC4J 構成パラメータに関する説明](#)」を参照してください。



## 事前変換の ojspc ユーティリティ

JSP ページを事前変換するときには、ojspc ユーティリティの使用を検討してください。JSP ページにはユーザーが最初にアクセスするため、このユーティリティを使用すると、ページを変換するときのパフォーマンスの損失が発生しません。事前変換の利点の詳細は、7-37 ページの「[JSP の事前変換](#)」を参照してください。ユーティリティ自体の詳細は、7-12 ページの「[ojspc 事前変換ユーティリティ](#)」を参照してください。

## OC4J と Oracle Application Server のその他のパフォーマンス機能

次に示す、パフォーマンスの最適化と監視に関する OC4J および Oracle Application Server の機能に注意してください。

- デフォルト以外の文字セットを使用する JspWriter 出力オブジェクトの OC4J JSP コードが最適化されました。
- データ・アクセス (SQL) タグ・ライブラリなど、OC4J が提供するタグ・ライブラリは、データベース接続などのために、追加の Oracle リソース・プーリングやリソース・クリーン・アップ機能を利用するように最適化されました。
- Oracle Application Server ダイナミック・モニタリング・サービスを使用して、パフォーマンスを追跡できます。2-15 ページの「[ダイナミック・モニタリング・サービスのサポート](#)」を参照。
- パフォーマンスおよび堅牢さに関する一般的な OC4J または Oracle Application Server の機能の詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server 10g パフォーマンス・ガイド』を参照してください。

## デフォルト・パッケージのインポート

Oracle9iAS リリース 2 (9.0.3) から、OC4J JSP コンテナは、JSP 仕様に基づいて次のパッケージを JSP ページにデフォルトでインポートします。page ディレクティブの import 設定は不要です。

```
javax.servlet.*
javax.servlet.http.*
javax.servlet.jsp.*
```

以前のリリースでは、次のパッケージもデフォルトでインポートされました。

```
java.io.*
java.util.*
java.lang.reflect.*
java.beans.*
```

使用する未修飾のクラス名と、インポートしたパッケージ内の同じ名前のクラスとの間で発生する競合を最小化するために、インポートするデフォルトのパッケージ・リストが縮小されました。ただし、このために、以前のバージョンの OC4J で使用していたアプリケーションで移行上の問題が発生する可能性があります。このようなアプリケーションは、正常にコンパイルされない場合があります。

デフォルト・リストのパッケージより多くのパッケージをインポートする必要がある場合は、次の 2 つの方法があります。

- 追加のパッケージ名または完全修飾したクラス名を、1 つ以上の page ディレクティブの `import` 設定に指定します。詳細は、1-7 ページの「[ディレクティブ](#)」の項の page ディレクティブについての説明、および 6-14 ページの「[累積される page ディレクティブの import 設定](#)」を参照してください。  
複数ページの場合、デフォルト・リストより多いパッケージのインポートは、グローバル・インクルード機能を使用して実行できます。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。
- 追加のパッケージ名または完全修飾されたクラス名は、JSP の `extra_imports` 構成パラメータまたは事前変換用の `ojspc -extraImports` オプションを使用して指定します。構文は、OC4J 構成パラメータ設定、JServ 構成パラメータ設定および `ojspc` オプション設定でそれぞれ異なります。必要に応じて次の項を参照してください。
  - 3-18 ページ「[JSP 構成パラメータの説明](#)」
  - 7-20 ページ「[ojspc のオプションの説明](#)」
  - B-10 ページ「[JServ の構成パラメータの説明](#)」

## JSP ファイルのネーミング規則

サーブレット仕様では、JSP ページのファイル名に拡張子 `.jsp` が必要です。ただし、サーブレット仕様 2.3 では、個別に変換可能な完全なページと、個別に変換できないページ・セグメント（`include` ディレクティブを介して移入したファイルなど）の違いは識別されません。

JSP 仕様 1.2 では、次のことをお勧めします。

- トップレベルのページ、動的にインクルードされるページおよび転送されるページ（独自に変換可能なページ）には、`.jsp` 拡張子を使用します。
- `include` ディレクティブを介して導入したページ・セグメント（独自に変換できないファイル）には、`.jsp` を使用しないでください。このようなファイルに対する拡張子の指定はありませんが、`.jspx`、`.jspxf` または `.jsf` の使用をお勧めします。

## JDK 1.4 に関する考慮事項：パッケージに含まれないクラスを起動できない

Sun 社の JDK 1.4 環境 (Oracle Application Server 10g (9.0.4) に付属) に移行する場合、移行に関する考慮事項の中で、サーブレットおよび JSP の開発者にとって特に重要なものがあります。

Sun 社が述べているように、コンパイラでは、不特定の名前空間から型をインポートするインポート文は拒否されます。これは、JDK の以前のバージョンに関するセキュリティ上の問題とあいまい性に対処するための措置でした。基本的に、これはパッケージに含まれていないクラス (クラスのメソッド) を起動できないことを示します。パッケージに含まれていないクラスを起動しようとする、コンパイル時に致命的エラーが発生します。

これは特に、JSP ページから JavaBeans を起動する JSP 開発者に影響します。このような Bean は、パッケージの外部にあることが多いためです (JSP 仕様 2.0 では、新しいコンパイラの要件を満たすために、Bean はパッケージ内に存在することが必要です)。パッケージの外部にある JavaBeans が起動されると、OC4J 9.0.3 / JDK 1.3.1 環境で作成および実行された JSP アプリケーションは、OC4J 9.0.4 / JDK 1.4 環境では動作しなくなります。

すべての JavaBean と起動されたその他のクラスがパッケージ内に含まれるようにアプリケーションを更新するまで、この問題を回避するには JDK 1.3.1 環境に戻るといった選択肢があります。

---

---

### 注意：

- `javac -source` コンパイラ・オプションは、JDK 1.4 コンパイラによって JDK 1.3.1 コードが透過的に処理されることを目的としていますが、このオプションは、パッケージに含まれないクラスの問題には対処していません。
  - OC4J では、JDK 1.3.1 と JDK 1.4 コンパイラのみがサポートおよび認証されています。`server.xml` ファイルに `<java-compiler>` 要素を追加することで、別のコンパイラを指定できます。これがパッケージに含まれないクラスの問題の対処方法になることもありますが、OC4J とともに使用する場合、Oracle では他のコンパイラは認証またはサポートされていません。(また、Oracle Application Server 環境で `server.xml` ファイルを直接更新しないでください。Oracle Enterprise Manager を使用してください。)
- 
- 

パッケージに含まれないクラスの問題および JDK 1.4 の互換性に関するその他の問題は、次の Web サイトを参照してください。

<http://java.sun.com/j2se/1.4/compatibility.html>

特に、「Java 2 プラットフォーム、Standard Edition、v1.4.0 と v1.3 の非互換性」のリンクをクリックしてください。

## OC4J が提供する主なサポート・ファイル

この項では、JSP コンテナまたは JSP アプリケーションで使用する JAR ファイルと ZIP ファイルの概要を説明します。これらのファイルは、システム上で OC4J のクラスパスにインストールされます。

- `ojjsp.jar`: JSP コンテナ用のクラス。
- `ojsputil.jar`: OC4J が提供するタグ・ライブラリとユーティリティ用のクラス。
- `xmlparserv2.jar`: XML 解析用。web.xml デプロイメント・ディスクリプタ・ファイル、タグ・ライブラリ・ディスクリプタ、および XML 関連タグ機能が必要です。
- `xsul2.jar` / `xsul11.jar`: クライアント上の XML 機能用（それぞれ、JDK 1.2.x 以上用、1.1.x 用）。
- `ojdbc14.jar` / `classes12.jar` / `classes111.jar`: Oracle JDBC Drivers 用（それぞれ、JDK 1.4 用、1.2 以上用、1.1 用）。
- `translator.jar`: Oracle SQLJ トランスレータ用。
- `runtime12.jar` / `runtime12ee.jar` / `runtime11.jar` / `runtime.jar` / `runtime-nonoracle.jar`: Oracle SQLJ Runtime 用（それぞれ、Oracle9i 以上の JDBC の JDK 1.2.x 以上用、Oracle9i 以上の JDBC の JDK 1.2.x 以上の Enterprise Edition 用、Oracle9i 以上の JDBC の JDK 1.1.x 用、任意の Oracle JDBC バージョンの JDK 1.1.x 以上用、Oracle 以外の JDBC ドライバの JDK 環境用）。
- `jndi.jar`: リソース（JDBC データ・ソース、Enterprise JavaBeans など）をルックアップするための JNDI サービス用。
- `jta.jar`: Java Transaction API 用。

特定のタグ・ライブラリなど、特定の領域に関連したファイルもあります。これには、次のファイルが含まれます。

- `mail.jar`: アプリケーション内での電子メール機能用（標準の `javax.mail` パッケージ）。
- `activation.jar`: 電子メール機能用の Java アクティブ化ファイル。
- `cache.jar`: Oracle Application Server の Java Object Cache 用（OC4J の Web Object Cache 用のデフォルトのバックエンド・リポジトリ）。

## OC4J での JSP 構成

次の項では、JSP 環境の構成に関する項目について説明します。

- [JSP コンテナの設定](#)
- [JSP 構成パラメータ](#)
- [JSP の OC4J 構成パラメータ](#)

---

---

### 注意：

- OC4J の構成ファイルと構成パラメータ、および手動によるそれらの更新方法の説明では、通常、スタンドアロン OC4J 環境を想定しています。これは開発時の典型的な環境です。本番デプロイなどの、Oracle Application Server 環境における Oracle Enterprise Manager を使用した JSP 構成の詳細は、3-32 ページの「[Oracle Enterprise Manager での JSP の構成](#)」を参照してください。
  - JServ などの OC4J 以外の環境では、`oracle.jsp.runtimev2.JspServlet` バージョンではなく、従来の `oracle.jsp.JspServlet` フロントエンド・サーブレットを使用してください。B-2 ページの「[JServ 環境での作業開始](#)」を参照。
- 
- 

## JSP コンテナの設定

JSP コンテナは、OC4J に適切に事前構成されています。次の設定は、OC4J の `global-web-application.xml` ファイルの設定を示します。このファイルは、JSP フロントエンド・サーブレットの名前をマッピングし、JSP ページの適切なファイル名拡張子をマッピングします。

```
<orion-web-app ... >
...
<web-app>
...
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>oracle.jsp.runtimev2.JspServlet</servlet-class>
  ...
  init_params
  ...
</servlet>
...
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>/* .jsp</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>/*.JSP</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>/*.sqljsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>/*.SQLJSP</url-pattern>
</servlet-mapping>

...
</web-app>
...
</orion-web-app>
```

global-web-application.xml ファイルの詳細は、『Oracle Application Server Containers for J2EE サブプレット開発者ガイド』を参照してください。

## JSP 構成パラメータ

OC4J の JSP フロントエンド・サブプレットの `oracle.jsp.runtimev2.JspServlet` は、JSP 操作を制御する多数の構成パラメータをサポートします。この項では、これらのパラメータについて説明します。次の項目では、パラメータのサマリー表、各パラメータの詳細な説明、およびこれらのパラメータを OC4J の `global-web-application.xml` ファイルや `orion-web.xml` ファイルに設定する方法について説明します。

- [JSP 構成パラメータのサマリー表](#)
- [JSP 構成パラメータの説明](#)
- [OC4J での JSP 構成パラメータの設定](#)

### JSP 構成パラメータのサマリー表

表 3-1 は、`JspServlet` でサポートする構成パラメータのサマリーです。この表では、各パラメータについて、事前に変換するページに対する同等の `ojspc` 変換オプション、およびパラメータの使用は実行時かコンパイル時か、について示します。

---

---

**注意：** `ojspc` オプションに関する情報は、7-20 ページの「[ojspc のオプションの説明](#)」を参照してください。

---

---

表 3-1 JSP 構成パラメータ (OC4J 環境)

| パラメータ                | 関連の<br>ojspc オプション | 説明   | デフォルト     | 実行時/<br>コンパイル<br>時 |
|----------------------|--------------------|--|-----------|--------------------|
| check_page_scope     | (なし)               | このブール値を true に設定すると、JspScopeListener による page スコープ・チェックが有効になります (OC4J のみ)。   | false     | 実行時                |
| debug_mode           | (なし)               | このブール値を true に設定すると、実行時に例外が発生した場合、スタック・トレースが出力されます。  | false     | 実行時                |
| emit_debuginfo       | -debug             | このブール値を true に設定すると、デバッグ用 (開発用) に、元の .jsp ファイルに対する行マップが生成されます。   | false     | コンパイル時             |
| external_resource    | -extres            | このブール値を true に設定すると、変換時に、ページのすべての静的なコンテンツが別の Java リソース・ファイルに格納されます。  | false     | コンパイル時             |
| extra_imports        | -extraImports      | このパラメータを使用すると、JSP のデフォルトより多いインポートが追加されます。  | null      | コンパイル時             |
| forgive_dup_dir_attr | -forgiveDupDirAttr | このブール値を true に設定すると、単一の JSP 変換単位内で同じディレクティブ属性に重複する設定がある場合でも、JSP 1.2 の変換エラーを回避します。  | false     | コンパイル時             |
| javaccmd             | -noCompile         | javac コマンドライン、または別の Java コンパイラを指定する場合に使用します。このオプションを使用すると、コンパイラは OC4J とは別のプロセスで実行されます。<br>use_old_compiler が false に設定されている場合、javaccmd パラメータは無視されます。 | null      | コンパイル時             |
| main_mode            | (なし)               | 変更が行われた場合に、JSP 生成クラスを自動的に再ロードするか、または JSP ページを自動的に再変換するかを決定します。設定できる値は、justrun、reload および recompile です。   | recompile | 実行時                |

表 3-1 JSP 構成パラメータ (OC4J 環境) (続き)

| パラメータ                      | 関連の<br>ojspc オプション    | 説明   | デフォルト | 実行時/<br>コンパイル<br>時 |
|----------------------------|-----------------------|--|-------|--------------------|
| no_tld_xml_validate        | -noTldXmlValidate     | このブール値を true に設定すると、TLD の XML 妥当性チェックは実行されません。デフォルトでは、TLD の妥当性チェックが実行されません。  | false | コンパイル時             |
| old_include_from_top       | -oldIncludeFromTop    | このブール値を true に設定すると、Oracle9iAS リリース 2 より前のリリースの動作との下位互換性を維持するために、ネストされた include ディレクティブ内のページの場所がトップレベルのページに関連づけられます。 | false | コンパイル時             |
| precompile_check           | (なし)                  | このブール値を true に設定すると、標準の jsp_precompile 設定に対して HTTP リクエストがチェックされます。   | false | 実行時                |
| reduce_tag_code            | -reduceTagCode        | このブール値を true に設定すると、カスタム・タグを使用するために生成されるコードのサイズがさらに縮小します。  | false | コンパイル時             |
| req_time_introspection     | -reqTimeIntrospection | このブール値を true に設定すると、コンパイル時にイントロスペクションが実行できない場合、リクエスト時に <b>JavaBean</b> のイントロスペクションが有効になります。                          | false | コンパイル時             |
| setproperty_onerr_continue | (なし)                  | このブール値を true に設定すると、property="*" の場合、jsp:setProperty の使用時にエラーが発生したとき、リクエスト・パラメータの反復と対応する Bean プロパティの設定が継続されます。       | false | 実行時                |



表 3-1 JSP 構成パラメータ (OC4J 環境) (続き)

| パラメータ                | 関連の<br>ojspc オプション | 説明   | デフォルト                                  | 実行時/<br>コンパイル<br>時 |
|----------------------|--------------------|--|--|--------------------|
| sqljcmd              | -S                 | このパラメータは、SQLJ コマンドラインを指定する場合、または別のSQLJ トランスレータを指定する場合に使用し、コマンドラインを必要に応じて設定します (開発用)。別のトランスレータを指定すると、そのトランスレータは OC4J とは別のプロセスで実行されます。Oracle Application Server が提供する Oracle SQLJ バージョンを使用するには、デフォルトのオプション設定で null 設定を使用します。 | null                                   | コンパイル時             |
| static_text_in_chars | -staticTextInChars | このブール値を true に設定すると、JSP トランスレータは、JSP ページの静的なテキストをバイトではなく文字として生成します。  | false                                  | コンパイル時             |
| tags_reuse_default   | -tagReuse          | JSP タグ・ハンドラの再利用モードを指定します。実行時モデルには runtime を、コンパイル時モデルには compiletime または compiletime_with_release を、タグ・ハンドラの再利用を禁止するには none を指定します。   | runtime                                | いずれか一方             |
| use_old_compiler     | (なし)               | このブール値を false に設定すると、JSP コンテナに対して残りの OC4J と同じコンパイラを使用するように強制されます。この値に設定しない場合、OC4J はデフォルトでインプロセス・コンパイル (または、該当する場合は javaccmd 設定に基づいて実行するコンパイル) を使用します。  | tools.jar<br>がクラスパスにある場合は true に設定します。 | コンパイル時             |

表 3-1 JSP 構成パラメータ (OC4J 環境) (続き)

| パラメータ                 | 関連の<br>ojspc オプション | 説明  | デフォルト              | 実行時/<br>コンパイル<br>時 |
|-----------------------|--------------------|---|--------------------|--------------------|
| well_known_taglib_loc | (なし)               | TLD キャッシングが有効になっていない場合は、複数の Web アプリケーションで共有するタグ・ライブラリ JAR ファイルを配置するディレクトリを指定します。デフォルトの位置は、 <code>ORACLE_HOME</code> ディレクトリの下<br><code>j2ee/home/jsp/lib/taglib</code> です。 | (「説明」の列を参照)        | コンパイル時             |
| xml_validate          | -xmlValidate       | このブール値を <code>true</code> に設定すると、 <code>web.xml</code> ファイルの XML 妥当性チェックが実行されます。デフォルトでは、 <code>web.xml</code> の妥当性チェックは実行されません。   | <code>false</code> | コンパイル時             |

## JSP 構成パラメータの説明

この項では、OC4J の JSP 構成パラメータについて詳細に説明します。

### `check_page_scope` (ブール値、デフォルト: `false`)

OC4J 環境で、このパラメータを `true` に設定すると、`JspScopeListener` ユーティリティによる Oracle 固有の `page` スコープ・チェックが有効になります。パフォーマンス上の理由から、デフォルトは `false` です。

このパラメータは、OC4J 以外の環境には関係ありません。JServ の場合、Oracle 固有の `page` スコープ・チェックは常に有効です。その他の環境では、Oracle 固有の実装は使用されません。`JspScopeListener` による `page` スコープ機能には `checkPageScope` カスタム・タグを使用する必要があります。`JspScopeListener` ユーティリティの詳細は、『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』を参照してください。

### `debug_mode` (ブール値、デフォルト: `false`)

`true` を使用すると、実行時に例外が発生した場合、スタック・トレースが出力されます。`false` を設定すると、この機能は無効になります。

---

**重要:** `debug_mode` が `false` に設定されていて、ファイルが見つからない場合、見つからないファイルのフルパスは表示されません。このことは、存在しない JSP ファイルがリクエストされたときにファイルの物理的なパスを表示しない場合、セキュリティ上の重要な考慮事項です。

---

**emit\_debuginfo** (ブール値、デフォルト: false)

開発時にこのフラグを true に設定すると、JSP トランスレータは、デバッグ用に、元の .jsp ファイルに対する行マップを生成します。false に設定すると、行は、生成されたページ実装クラスである .java ファイルに対してマッピングされます。

---

---

**注意:**

- emit\_debuginfo は、Oracle JDeveloper によって使用可能になります。
  - 事前に変換するページの場合は、ojspc -debug オプションを使用します。
- 
- 

**external\_resource** (ブール値、デフォルト: false)

このフラグを true に設定すると、JSP トランスレータは、ページの静的なコンテンツを生成されたページ実装クラスのサービス・メソッドではなく Java リソース・ファイルに格納します。

リソース・ファイル名は、JSP ページ名に拡張子 .res を付けた名前になります。Oracle Application Server 10g では、たとえば、MyPage.jsp という名前のファイルを変換すると、通常の実出力以外に MyPage.res というファイルが作成されます (厳密な実装は、今後のリリースで変更される可能性があります)。

トランスレータは、リソース・ファイルを、生成されたクラス・ファイルとして同じディレクトリに格納します。

1 ページに大量の静的なコンテンツが存在する場合は、この技法によって変換速度が向上し、ページの実行速度も向上します。詳細は、6-9 ページの「[大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処](#)」を参照してください。

---

---

**注意:** 事前に変換するページの場合は、ojspc -extres オプションを使用します。

---

---

**extra\_imports** (インポート・リスト、デフォルト: null)

3-9 ページの「[デフォルト・パッケージのインポート](#)」に説明されているように、各 JSP ページにインポートされるパッケージのデフォルト・リストは、OC4J 9.0.3 の実装前のリストより小規模です。このデフォルト・リストは、JSP の仕様に基づいています。ただし、extra\_imports 構成パラメータを使用して、追加インポート用のパッケージ名または完全修飾されたクラス名を指定することによって、コードの更新を回避できます。一般的な構文は、3-27 ページの「[OC4J での JSP 構成パラメータの設定](#)」を参照してください。名前には、

カンマまたは空白のデリミタが使用できます。たとえば、次のサンプルは、いずれも問題ありません。

```
<init-param>
  <param-name>extra_imports</param-name>
  <param-value>java.util.* java.beans.*</param-value>
</init-param>
```

または

```
<init-param>
  <param-name>extra_imports</param-name>
  <param-value>java.util.*,java.beans.*</param-value>
</init-param>
```

---

---

**注意：**

- 事前に変換するページの場合は、`ojspc -extraImports` オプションを使用します。
  - `extra_imports` のかわりに、グローバル・インクルードを使用することもできます。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。
- 
- 

**forgive\_dup\_dir\_attr** (ブール値、デフォルト: `false`)

このブール値を `true` に設定すると、単一の JSP 変換単位 (JSP ページおよび `include` ディレクティブを使用してインクルードされたページ) 内で同じディレクティブ属性に重複する設定がある場合でも、JSP 1.2 以上の環境での変換エラーを回避します。

JSP 仕様では、複数のディレクティブ属性 (`page` ディレクティブの `import` 属性を除く) が単一の JSP 変換単位内で 2 回以上設定されていないことを、JSP コンテナが確認する必要があります。詳細は、6-12 ページの「[page ディレクティブ属性の重複設定の禁止](#)」を参照してください。

JSP 仕様 1.1 では、このような制限はありませんでした。OC4J では、下位互換性のために `forgive_dup_dir_attr` パラメータを提供しています。

---

---

**注意：** 事前に変換するページの場合は、`ojspc -forgiveDupDirAttr` オプションを使用します。

---

---

**javaccmd** (コンパイラの実行可能ファイルとオプション、デフォルト:null)

`use_old_compiler` を `true` に設定すると、`javaccmd` を使用して (通常は開発過程で)、JSP 変換時に使用する Java コンパイラのコマンドラインを指定できます。これは、特定の `javac` 設定または別の Java コンパイラ (コマンドラインを必要に応じて設定します) を指定する場合に役立ちます。実行可能ファイルのフルパスを指定するか、または実行可能ファイルのみ指定すると、JSP コンテナはシステム・パス内を検索できます。

たとえば、`javaccmd` を値 `javac -verbose` に設定すると、コンパイラは冗長モードで実行されます。

次の点に注意してください。

- `use_old_compiler` が `false` に設定されている場合、`javaccmd` は無視されます。
- `javaccmd` を使用すると、コンパイラは OC4J とは別のプロセスで実行されます。

関連情報は、3-6 ページの「[JSP のコンパイルに関する考慮事項](#)」を参照してください。

---

---

#### 注意：

- 指定した Java コンパイラはクラスパスにインストールされ、フロントエンド・ユーティリティ (該当する場合) はシステム・パスにインストールされている必要があります。
- 事前に変換するページの場合は、`ojspc -noCompile` オプションによって同様の機能を使用できます。その場合は、`javac` によるコンパイルは実行されないため、任意のコンパイラを使用して、変換されたクラスを手動でコンパイルできます。

---

---

**main\_mode** (再ロードまたは再変換のモード、デフォルト:recompile)

JSP コンテナの操作モードを指示するフラグです。特に、JSP ページの自動再変換、および変更された JSP 生成 Java クラスのリロードを指示します。

次の設定がサポートされています。

- `justrun`: 実行時ディスパッチャは、タイムスタンプ・チェックを実行しません。したがって、JSP ページの再変換または JSP 生成 Java クラスのリロードはありません。コード変更がないデプロイ環境では、このモードが最も効率的です。
- `reload`: ディスパッチャは、ページ実装クラスなど、JSP トランスレータによって生成されたクラスのタイムスタンプをチェックして、最後のロード以降に変更または再デプロイされたクラスを再ロードします。これは、開発環境から本番環境に、ページ・ソースではなくコンパイル済クラスをデプロイまたは再デプロイする場合などに便利です。
- `recompile` (デフォルト) : ディスパッチャは、JSP ページのタイムスタンプをチェックし、ロード以降に変更されている場合はそのページを再変換および再ロードし、`reload` 機能を実行します。

**no\_tld\_xml\_validate** (ブール値、デフォルト: false)

true に設定すると、アプリケーションのタグ・ライブラリ・ディスクリプタ (TLD) の XML 妥当性チェックが無効になります。デフォルトでは、TLD の妥当性チェックが実行されます。

関連情報は、8-7 ページの「[TLD の妥当性チェックおよび機能の概要](#)」を参照してください。

---

---

**注意：** 事前に変換するページの場合は、`ojspc -noTldXmlValidate` オプションを使用します。

---

---

**old\_include\_from\_top** (ブール値、デフォルト: false)

このオプションは、include ディレクティブの機能について、Oracle9iAS リリース 2 より前の Oracle JSP バージョンとの下位互換性を維持するために使用します。このパラメータを true に設定すると、ネストされた include ディレクティブ内のページの場所がトップレベルのページに対して相対的になります。false に設定すると、ページの場所が 1 つ上位の親ページに対して相対的になります。これは JSP 仕様に準拠しています。

---

---

**注意：** 事前に変換するページの場合は、`ojspc -oldIncludeFromTop` オプションを使用します。

---

---

**precompile\_check** (ブール値、デフォルト: false)

true に設定すると、標準の `jsp_precompile` 設定に対して HTTP リクエストがチェックされます。`precompile_check` が true で、リクエストによって `jsp_precompile` が使用可能な場合、JSP ページは実行されず、事前変換されるのみです。`precompile_check` を false に設定すると、パフォーマンスが改善され、リクエストの `jsp_precompile` の設定値は無視されます。

`jsp_precompile` の詳細は、7-39 ページの「[実行を伴わない標準の JSP の事前変換](#)」、および Sun 社の JSP 仕様を参照してください。

**reduce\_tag\_code** (ブール値、デフォルト: false)

Oracle JSP の実装によって、カスタム・タグを使用するために生成されるコードのサイズは縮小されますが、`reduce_tag_code` を true に設定すると、さらにサイズが縮小されます。ただし、タグ・ハンドラの再利用に関しては、パフォーマンスに影響を与える場合があります。8-41 ページの「[タグ・ハンドラのコード生成](#)」を参照。

---

---

**注意：** 事前に変換するページの場合は、`ojspc -reduceTagCode` オプションを使用します。

---

---

**req\_time\_introspection** (ブール値、デフォルト: false)

trueに設定すると、コンパイル時にイントロスペクションが実行できない場合、リクエスト時にJavaBeanのイントロスペクションが有効になります。ただし、有効なコンパイル時のイントロスペクションが正常終了した場合、このフラグの設定に関係なく、リクエスト時のイントロスペクションは実行されません。

リクエスト時のイントロスペクションの使用例として、タグ・ハンドラによって、タグ補足情報クラスのVariableInfoにある一般的なjava.lang.Objectインスタンスが変換時とコンパイル時に戻されますが、実際には、特定のオブジェクトがリクエスト時(実行時)に生成される場合を想定します。この場合、req\_time\_introspectionが有効になると、JSPコンテナはリクエスト時までイントロスペクションを遅延します(VariableInfoの使用方法は、8-41ページの「スクリプト変数、宣言およびタグ補足情報クラス」を参照してください)。

このフラグには、if..then..elseループの別の分岐などで、Beanを2回宣言できるという効果もあります。次の例を考えてみます。req\_time\_introspectionのデフォルト値falseを使用すると、このコードにより解析例外が発生します。true値を使用すると、コードはエラーなしで機能します。

```
<% if (cond) { %>
    <jsp:useBean id="foo" class="pkgA.Foo1" />
<% } else { %>
    <jsp:useBean id="foo" class="pkgA.Foo2" />
<% } %>
```

---

**注意:** 事前に変換するページの場合は、ojspc  
-reqTimeIntrospection オプションを使用します。

---

**setproperty\_onerr\_continue** (ブール値、デフォルト: false)

このブール値をtrueに設定すると、property="\*"の場合、jsp:setProperty文の使用時にエラーが発生したとき、リクエスト・パラメータの反復と対応するBeanプロパティの設定が継続されます。

以前のリリースでは、どのような場合でもOC4J JSPコンテナは継続されました。OC4J 9.0.4実装では、JSPリファレンス実装との互換性を改善するため、エラーが発生するとデフォルトでOC4J JSPコンテナは反復を停止します。setproperty\_onerr\_continueフラグによって、下位互換性が維持されます。

関連情報は、1-15ページの「標準アクション: JSPタグ」の項にあるjsp:setPropertyの説明を参照してください。

**sqljcmd** (SQLJ トランスレータの実行可能ファイルとオプション、デフォルト: null)

このパラメータは、次のいずれかの場合の開発過程で役立ちます。

- 1つ以上の SQLJ コマンドライン・オプションを設定する場合
- 異なる SQLJ トランスレータ、または OC4J が提供するバージョンとは異なるバージョンを使用する場合
- OC4J とは別のプロセスで SQLJ を実行する場合

このオプションを使用すると、SQLJ トランスレータは OC4J とは別のプロセスで実行されません。

実行可能ファイルのフルパスを指定するか、または実行可能ファイルのみ指定すると、JSP コンテナはシステム・パス内を検索できます。

たとえば、ユーザー `scott/tiger` でオンライン・セマンティクス・チェックを行う SQLJ を実行し、ISO 標準の SQLJ コードを生成するには、`sqljcmd` を次の値に設定します。

```
sqljcmd=sqlj -user=scott/tiger -codegen=iso
```

適切な SQLJ ライブラリがクラスパスにあり、フロントエンド・ユーティリティ (この例の `sqlj` など) がシステム・パスにあることが必要です。Oracle SQLJ の場合は、`translator.jar` ファイルおよび適切な SQLJ Runtime の JAR ファイルがクラスパスにあることが必要です。3-12 ページの「[OC4J が提供する主なサポート・ファイル](#)」を参照。

---

---

**注意：**

- 事前に変換するページの場合は、`ojspc -S` オプションによって関連する機能を使用できます。
  - SQLJ がサポートされるのは、現行リリースまでです。
- 
- 

**static\_text\_in\_chars** (ブール値、デフォルト: false)

true に設定すると、JSP トランスレータは、JSP ページの静的なテキストをバイトではなく文字として生成します。次の例に示すように、実行時に文字エンコードをアプリケーションで動的に変更する必要がある場合は、このフラグを有効にします。

```
<% response.setContentType("text/html; charset=UTF-8"); %>
```

(関連情報は、9-5 ページの「[コンテンツ・タイプの動的な設定](#)」を参照してください。)

デフォルト設定の false を使用すると、静的なテキスト・ブロックの出力で、パフォーマンスが改善されます。



---

---

**注意：** 事前に変換するページの場合は、`ojspc -staticTextInChars` オプションを使用します。

---

---

#### **tags\_reuse\_default** (タグ・ハンドラの再利用のモード、デフォルト:runtime)

このパラメータを使用して、タグ・ハンドラの再利用 (タグ・ハンドラ・インスタンスのプーリング) のモードを指定します。

- タグ・ハンドラの再利用を無効にするには、`none` に設定します。この設定は、特定の JSP ページで、JSP ページ・コンテキスト属性 `oracle.jsp.tags.reuse` を `true` の値に設定するとオーバーライドできます。
- タグ・ハンドラ再利用の実行時モデルを有効にするには、デフォルトの `runtime` に設定します。この設定は、特定の JSP ページで、JSP ページ・コンテキスト属性 `oracle.jsp.tags.reuse` を `false` の値に設定するとオーバーライドできます。
- タグ・ハンドラ再利用のコンパイル時モデルをその基本モードで有効にするには、`completetime` に設定します。
- タグ・ハンドラ再利用のコンパイル時モデルを、その「解放」モードで有効にするには、`completetime_with_release` に設定します。この場合、タグ・ハンドラの `release()` メソッドは、指定したページで指定したタグ・ハンドラが使用されている間にコールされます。

---

---

#### **注意：**

- 下位互換性のために、`runtime` と等価の `true` 設定、`none` と等価の `false` 設定もサポートされます。
  - 事前に変換するページの場合は、`ojspc -tagReuse` オプションを使用します。
- 
- 

タグ・ハンドラの再利用に関する詳細は、8-38 ページの「[実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)」を参照してください。

#### **use\_old\_compiler** (プール値、デフォルト:tools.jar がクラスパスにある場合はtrue)

`use_old_compiler` を `false` に設定すると、JSP コンテナに対して残りの OC4J と同じコンパイラを使用するように強制できます。コンパイルには、`javac` を使用するアウトプロセス・コンパイル (デフォルト) と、`server.xml` ファイルの `<java-compiler>` 要素に基づくコンパイルがあります。`tools.jar` がクラスパスにある場合、`use_old_compiler` フラグはデフォルトで `true` に設定されます。これによって、`javaccmd` が設定されている場合を除いて、インプロセス・コンパイルが使用されます (`tools.jar` がクラスパスにあることを確認するには、`server.xml` ファイルの `<library>` 要素を使用します)。

関連情報は、3-6 ページの「[JSP のコンパイルに関する考慮事項](#)」を参照してください。

---

---

**注意：**

- `tools.jar` がクラスパスにない場合、`use_old_compiler` は強制的に `false` に設定されます。
  - アウトプロセス・コンパイラを使用する場合、残りの OC4J が使用するコンパイラとは異なるコンパイラを使用するには、`use_old_compiler` を `true` に設定し、`javaccmd` パラメータを使用して任意のコンパイラを指定します。
- 
- 

**well\_known\_taglib\_loc** (共有タグ・ライブラリの位置、デフォルト:説明を参照)

永続的な TLD キャッシングが有効になっていない場合は、`well_known_taglib_loc` を使用して、複数の Web アプリケーション間で共有するタグ・ライブラリ JAR ファイルを配置する予約済の場所として、1つのディレクトリを指定できます。重要な関連情報は、8-20 ページの「[TLD キャッシングと予約済のタグ・ライブラリの場所](#)」を参照してください。

相対ディレクトリの位置を指定します。これは、`ORACLE_HOME` が定義されている場合は `ORACLE_HOME` の下、`ORACLE_HOME` が定義されていない場合は、(OC4J プロセスが開始された) カレント・ディレクトリの下になります。デフォルト値は、次のとおりです。

- `ORACLE_HOME` が定義されている場合は、  
`ORACLE_HOME/j2ee/home/jsp/lib/taglib/`

または

- `ORACLE_HOME` が定義されていない場合は、`./jsp/lib/taglib`

**xml\_validate** (ブール値、デフォルト: `false`)

`true` に設定して、アプリケーション `web.xml` ファイルの XML 妥当性チェックを有効にします。Tomcat リファレンス実装では XML 妥当性チェックを実行しないため、デフォルトでは、`xml_validate` は `false` です。

---

---

**注意：** 事前に変換するページの場合は、`ojspc -xmlValidate` オプションを使用します。

---

---

## OC4JでのJSP構成パラメータの設定

スタンドアロンOC4J開発環境では、JSP構成パラメータをJSPフロントエンド・サーブレットの<servlet>要素内で、global-web-application.xml、web.xmlまたはorion-web.xmlに直接設定できます。3-13ページの「[JSPコンテナの設定](#)」に示したglobal-web-application.xml部分では、設定がinit\_paramsプレースホルダの位置に入ります。

---

---

**注意：** Oracle Application Serverの本番環境では、Enterprise Managerを使用して構成します。Enterprise ManagerでApplication Server Control Webモジュールの「[拡張プロパティ](#)」ページを使用すると、global-web-application.xmlまたはorion-web.xmlファイルを更新できます。このApplication Server Controlの詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

---

---

次の例は、JSPフロントエンド・サーブレットの<servlet>要素とサブ要素の設定のリストです。このサンプルでは、precompile\_checkフラグを有効にし、タイムスタンプをチェックせずに実行するためにmain\_modeフラグを設定して、Javaコンパイラを冗長モードで実行します。

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>oracle.jsp.runtimev2.JspServlet</servlet-class>
  <init-param>
    <param-name>precompile_check</param-name>
    <param-value>>true</param-value>
  </init-param>
  <init-param>
    <param-name>main_mode</param-name>
    <param-value>justrun</param-value>
  </init-param>
  <init-param>
    <param-name>javaccmd</param-name>
    <param-value>javac -verbose</param-value>
  </init-param>
</servlet>
```

global-web-application.xmlファイルの設定は、特定アプリケーションのweb.xmlファイルの設定でオーバーライドできます。また、orion-web.xmlファイルの設定を使用すると、web.xmlファイルの設定に関するデブロイ固有のオーバーライドが可能です。global-web-application.xmlおよびorion-web.xmlの詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

## JSP の OC4J 構成パラメータ

JSP コンテナの JspServlet フロントエンド・サーブレットに対するパラメータとは別に、JSP ページに影響を与える OC4J 構成パラメータもあります。この項では、OC4J の `global-web-application.xml` ファイルまたは `orion-web.xml` ファイルに指定するルート `<orion-web-app>` 要素の JSP 関連属性について説明します。これらのファイルの詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

### JSP 関連の OC4J 構成パラメータに関する説明

OC4J の `global-web-application.xml` ファイルまたは `orion-web.xml` ファイルに指定する次の `<orion-web-app>` 属性は、JSP のパフォーマンスと機能に影響を与えます。

- `jsp-print-null`: このフラグを `false` に設定すると、JSP ページからの NULL 出力に対して、NULL 文字列ではなく空の文字列が出力されます。デフォルトは `true` です。
- `jsp-timeout`: 整数値（秒単位）を指定します。この値を経過した後にリクエストされなかった場合、その JSP ページはメモリーから削除されます。これによって、コール頻度の低いページに割り当てられているリソースが解放されます。デフォルト値は 0（ゼロ）で、タイムアウトはありません。
- `jsp-cache-directory`: JSP キャッシュ・ディレクトリは、JSP トランスレータから出力されたファイルのベース・ディレクトリとして使用されます（7-7 ページの「[JSP トランスレータの出力ファイルの格納場所](#)」を参照）。また、アプリケーション・レベルの TLD キャッシングのベース・ディレクトリとしても使用されます（8-21 ページの「[TLD キャッシュ機能とファイル](#)」を参照）。デフォルト値は `./persistence` で、アプリケーションのデプロイ・ディレクトリに関連します。
- `jsp-cache-tlds`: 永続的な TLD キャッシングが有効かどうかを示します。TLD キャッシングは、予約済のタグ・ライブラリの場所にある TLD についてはグローバル・レベル、`/WEB-INF` ディレクトリの下に TLD についてはアプリケーション・レベルの両方で実装されます。デフォルト設定は `true` です。予約済の場所は、`jsp-taglib-locations` 属性によります。関連情報は、8-20 ページの「[TLD キャッシングと予約済のタグ・ライブラリの場所](#)」を参照してください。
- `jsp-taglib-locations`: `jsp-cache-tlds` 属性を介して永続的な TLD キャッシングが有効になっている場合は、`jsp-taglib-locations` を使用して、セミコロンで区切られた 1 つ以上のディレクトリのリストを、予約済の場所として使用するよう指定します。タグ・ライブラリ JAR ファイルをこれらの場所に格納して、複数の Web アプリケーションで共有したり、TLD キャッシングに使用できます。重要な関連情報は、8-20 ページの「[TLD キャッシングと予約済のタグ・ライブラリの場所](#)」を参照してください。

絶対ディレクトリ・パスまたは相対ディレクトリ・パスの組合せを指定できます。相対ディレクトリは、`ORACLE_HOME` が定義されている場合は `ORACLE_HOME` の下、`ORACLE_HOME` が定義されていない場合は、OC4J プロセスが開始されたカレント・ディレクトリの下になります。デフォルト値は、次のとおりです。

- `ORACLE_HOME` が定義されている場合は、  
`ORACLE_HOME/j2ee/home/jsp/lib/taglib/`
- または
- `ORACLE_HOME` が定義されていない場合は、`./jsp/lib/taglib`

---

**重要：** `orion-web.xml` ではなく、`global-web-application.xml` のみで `jsp-taglib-locations` 属性を使用します。

---

- `simple-jsp-mapping:*.jsp` ファイル拡張子が、アプリケーションに影響する Web ディスクリプタの `<servlet>` 要素 (`global-web-application.xml`、`web.xml` および `orion-web.xml`) の `oracle.jsp.runtimev2.JspServlet` フロントエンド JSP サブレットのみにマッピングされる場合は、`true` に設定します。これにより、JSP ページのパフォーマンスが改善されます。デフォルト設定は `false` です。
- `enable-jsp-dispatcher-shortcut`: デフォルトで `true` 設定の場合、特に `simple-jsp-mapping` 属性の `true` 設定とともに使用すると、OC4J JSP コンテナによってパフォーマンスが大幅に改善されます。これは特に、多数の `jsp:include` 文がある JSP ページに当てはまります。ただし、`true` 設定の使用は、`web.xml` の `<jsp-file>` 要素を使用して JSP ファイルを定義する場合、次の例に示すように、そのファイルに対応する `<url-pattern>` 仕様があることを前提とします。

```
<servlet>
  <servlet-name>foo</servlet-name>
  <jsp-file>bar.jsp</jsp-file>
</servlet>
...
<servlet-mapping>
  <servlet-name>foo</servlet-name>
  <url-pattern>/mypath</url-pattern>
</servlet-mapping>
```

対応する `<url-pattern>` を設定せずに `<jsp-file>` を使用する場合 (典型的な使用例ではありません) は、`enable-jsp-dispatcher-shortcut="false"` を設定します。

---

---

**注意：** <orion-web-app> 要素の autoreload-jsp-pages 属性および autoreload-jsp-beans 属性は、Oracle Application Server 10g (9.0.4) の OC4J JSP コンテナではサポートされていません。autoreload-jsp-pages と同等の機能を持つ JSP の main\_mode 構成パラメータを使用できます (3-18 ページの「[JSP 構成パラメータの説明](#)」を参照)。

---

---

## JSP 関連の OC4J 構成パラメータの設定

OC4J インスタンス内のすべてのアプリケーションに適用する構成値を設定するには、OC4J の global-web-application.xml ファイルの <orion-web-app> 要素を使用します。特定のアプリケーション・デプロイに対して構成値を設定するには、デプロイ固有の orion-web.xml ファイルの <orion-web-app> 要素を使用して、global-web-application.xml の設定をオーバーライドします。

次に例を示します。

```
<orion-web-app ... jsp-print-null="false" ... >
...
</orion-web-app>
```

<orion-web-app> 要素には多くの属性とサブ要素があることに注意してください。詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

---

---

**注意：** OC4J スタンドアロン環境を使用している場合のみ、これらのファイルを直接更新します。Oracle Application Server 環境では、Enterprise Manager を使用して構成します。Oracle Application Server 10g (9.0.4) において、Enterprise Manager の Application Server Control の「JSP プロパティ」ページでは JSP の <orion-web-app> 属性はサポートされていませんが、Application Server Control Web モジュールの「拡張プロパティ」ページを介して設定できます。このページの詳細は、『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

---

---

## 主な OC4J 構成ファイル

OC4J 環境には、次の主な構成ファイルがあります。

すべての OC4J アプリケーション用のグローバル・ファイルは、OC4J 構成ファイルのディレクトリにあります。

- `server.xml`: 各 J2EE アプリケーションの `<application>` サブ要素を持つ、全体的な `<application-server>` 要素が含まれます。各 `<application>` サブ要素は、アプリケーションの名前、および EAR デプロイ・ファイルの名前と場所を指定します。  
`<application-server>` 要素は、EAR ファイルがデプロイ用に格納および抽出される一般的なアプリケーション・ソース・ディレクトリ、および OC4J 固有の構成ファイルが生成されるアプリケーション・デプロイ・ディレクトリを指定します。さらに、デフォルトの Web サイト用の `<web-site>` 要素があり、サーバーに追加する Web サイトごとに、この `<web-site>` 要素を追加できます。
- `default-web-site.xml` (または、スタンドアロン OC4J 用の `http-web-site.xml`、あるいは該当する他の Web サイトの XML ファイル) : デフォルトの Web サイトの各 Web アプリケーションで使用する `<web-app>` 要素が含まれ、アプリケーション名を Web アプリケーション名にマッピングします。Web アプリケーション名は、WAR デプロイ・ファイル名に対応しています。追加の Web サイトの XML ファイルには、`server.xml` ファイル内の追加の Web サイトに指定した内容と同じ機能があります。
- `global-web-application.xml`: OC4J Web アプリケーション用のグローバル構成ファイルです。デフォルトの構成を確立し、JSP フロントエンド・サーブレットの `JspServlet` の設定と構成が含まれます。
- `application.xml`: OC4J デフォルト・アプリケーションのアプリケーション・ディスクリプタです。デフォルトでは、他の OC4J アプリケーションの親アプリケーションです。標準の J2EE アプリケーション・レベルの `application.xml` ファイルと混同しないでください。OC4J デフォルト・アプリケーションのアプリケーション・ディスクリプタは OC4J 固有で、`orion-application.dtd` によって管理されます。
- `data-sources.xml`: データベース接続用のデータ・ソースを指定します。

(Oracle Application Server の OC4J ディレクトリ・パスは構成可能です。スタンドアロン OC4J の場合、構成ファイルのディレクトリは、デフォルトで `j2ee/home/config` です。)

グローバルな `application.xml` ファイル以外に、各アプリケーション用の標準の `application.xml` ファイルと `orion-application.xml` ファイル (オプション) があります。これらのファイルは、アプリケーションの EAR ファイルにあります。

また、アプリケーションの EAR ファイル内にある WAR ファイルには、標準の `web.xml` ファイルと `orion-web.xml` ファイル（オプション）があります。これらのファイルは、アプリケーション固有およびデプロイ固有の構成を設定するために使用し、`global-web-application.xml` の設定をオーバーライドしたり、必要に応じて追加の設定を提供します。`global-web-application.xml` ファイルと `orion-web.xml` ファイルは、同じ要素（`web.xml` ファイルでサポートされている要素のスーパーセット）をサポートします。

`orion-application.xml` ファイルと `orion-web.xml` ファイルがアーカイブ・ファイルにない場合、これらのファイルは、`global-web-application.xml` ファイルの設定に従って、最初のデプロイ時に生成されます。

追加情報は、7-34 ページの「[EAR/WAR デプロイの概要](#)」を参照してください。これらのファイルの使用方法については、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server Containers for J2EE サブレット開発者ガイド』を参照してください。

## Oracle Enterprise Manager での JSP の構成

Oracle Application Server 環境では、本番デプロイ用の構成など、Enterprise Manager を使用して OC4J 構成します。これには、OC4J JSP コンテナ用のフロントエンド JSP サブレットの構成も含まれます。

Oracle Enterprise Manager Application Server Control は、Oracle Application Server インスタンス用の管理ツールです。この管理ツールを使用して、リアルタイムのパフォーマンスを監視し、Oracle Application Server のコンポーネントやインスタンスを管理および構成できます。これには、OC4J のインスタンスも含まれます。特に、Application Server Control には、「JSP プロパティ」ページが含まれています。Application Server Control は Oracle Application Server とともにインストールされます。ias\_admin ユーザーでログインします。



## Application Server Control の「JSP プロパティ」ページ

次の図は、OC4J インスタンス用の Application Server Control の「JSP プロパティ」ページ（一部）を示しています。

### JSP Properties: jsp

Refreshed at Wednesday, July 10, 2002 7:41:52 PM PDT 

#### Oracle JSP Container Properties

The following properties may be used to configure the Oracle JSP Container.

|                                      |                                    |                    |  |
|--------------------------------------|------------------------------------|--------------------|--|
| Debug Mode                           | <input type="button" value="No"/>  | Emit Debug Info    | <input type="button" value="No"/>            |
| External Resource for Static Content | <input type="button" value="Yes"/> | When a JSP Changes | <input type="button" value="Recompile JSP"/> |
| Generate Static Text as Bytes        | <input type="button" value="Yes"/> | Precompile Check   | <input type="button" value="No"/>            |
| Tags Reuse Default                   | <input type="button" value="Yes"/> | Validate XML       | <input type="button" value="No"/>            |
| Reduce Code Size for Custom Tags     | <input type="button" value="No"/>  |                    |  |
| SQLJ Command                         | <input type="text"/>               |                    |  |
| Alternate Java Compiler              | <input type="text"/>               |                    |  |

Revert

Apply

Enterprise Manager の Application Server Control を介して Oracle Application Server インスタンスに初めてアクセスすると、Oracle Application Server インスタンス・ホームページが表示されます。「JSP プロパティ」ページにドリルダウンするには、次の手順を実行します。

1. Oracle Application Server インスタンス・ホームページで、OC4J インスタンスの名前を「システム・コンポーネント」表で選択します。その OC4J インスタンスの OC4J ホームページが表示されます。
2. OC4J ホームページで、「管理」をクリックします。OC4J 管理ページが表示されます。
3. OC4J 管理ページで、「インスタンス・プロパティ」の下にある「JSP コンテナのプロパティ」をクリックします。「JSP プロパティ」ページが表示されます。

Enterprise Manager の使用については、次のマニュアルを参照してください。Web アプリケーションのデプロイと構成の概要は、『Oracle Application Server Containers for J2EE サブプレット開発者ガイド』を参照してください。OC4J 関連のデプロイと構成については、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。Enterprise Manager を使用した Oracle Application Server 環境の管理についての一般情報は、『Oracle Application Server 10g 管理者ガイド』を参照してください。

## 「JSP プロパティ」 ページでサポートされる構成パラメータ

表 3-2 は、Enterprise Manager の Application Server Control の「JSP プロパティ」 ページに表示される JSP コンテナのプロパティと、3-14 ページの「JSP 構成パラメータ」で説明されている JSP コンテナのフロントエンド・サーブレットの構成パラメータとの対応関係を示しています。設定の意味は、該当する項を参照してください。

設定できる値のうち、太字はデフォルト値です。Application Server Control のデフォルトは本番環境用のものであり、開発環境におけるデフォルトとは必ずしも同じではありません。

**表 3-2 Application Server Control のプロパティ、JSP パラメータ**

| Application Server Control の JSP コンテナのプロパティ | 設定できる値                                  | JSP 構成パラメータ          | 設定できる値                                |
|---|---|----------------------|---------------------------------------|
| デバッグ・モード                                    | No<br>Yes                               | debug_mode           | false<br>true                         |
| 静的コンテンツの外部リソース                              | No<br>Yes                               | external_resource    | false<br>true                         |
| 静的テキストをバイトとして生成                             | No<br>Yes                               | static_text_in_chars | false<br>true                         |
| デフォルトでタグを再利用                                | No<br>Yes                               | tags_reuse_default   | None<br><b>runtime</b>                |
| カスタム・タグのコード・サイズの削減                          | No<br>Yes                               | reduce_tag_code      | false<br>true                         |
| デバッグ情報の表示                                   | No<br>Yes                               | emit_debuginfo       | false<br>true                         |
| JSP ページの変更時                                 | <b>JSP の再コンパイル</b><br>クラスのリロード<br>実行しない | main_mode            | <b>recompile</b><br>reload<br>justrun |
| プリコンパイル・チェック                                | No<br>Yes                               | precompile_check     | false<br>true                         |
| XML の検証                                     | No<br>Yes                               | xml_validate         | false<br>true                         |
| SQLJ コマンド                                   | コマンド文字列<br>(デフォルトは<br>NULL)             | sqljcmd              | コマンド文字列<br>(デフォルトは<br>NULL)           |
| 代替 Java コンパイラ                               | コマンド文字列<br>(デフォルトは<br>NULL)             | javaccmd             | コマンド文字列<br>(デフォルトは<br>NULL)           |

---

---

**注意:**

- Oracle Application Server 10g (9.0.4) では、Application Server Control がサポートするのは、コンパイル時ではなく実行時のタグ・ハンドラの再利用のみです。つまり、`completetime` または `completetime_with_release` の `tags_reuse_default` 設定は、Application Server Control では直接サポートされていません。
  - Application Server Control の JSP コンテナのプロパティ「静的テキストをバイトとして生成」は、JSP 構成パラメータ `static_text_in_chars` に対応しますが、両者の方向性は反対です。そのため、それぞれのデフォルト値「Yes」と「false」は等価になります。
- 
- 

## 「JSP プロパティ」ページでサポートされていない構成パラメータ

Oracle Application Server 10g (9.0.4) では、次の構成パラメータは Application Server Control の「JSP プロパティ」ページでサポートされていません。

- JSP フロントエンド・サーブレット・パラメータ: `check_page_scope`、`extra_imports`、`forgive_dup_dir_attr`、`no_tld_xml_validate`、`old_include_from_top`、`req_time_introspection` および `well_known_taglib_loc`
- `global-web-application.xml` または `orion-web.xml` 内の `<orion-web-app>` 要素の JSP 関連属性: `jsp-print-null` と `jsp-timeout`、`jsp-cache-directory`、`jsp-cache-tlds`、`jsp-taglib-locations`、`simple-jsp-mapping` および `enable-jsp-dispatcher-shortcut`

かわりに、これらのパラメータは、`orion-web.xml` または他の適切な XML ファイル (`web.xml` や `global-web-application.xml` など) で更新する必要があります。`orion-web.xml` または `global-web-application.xml` は、Application Server Control Web モジュールの「拡張プロパティ」ページで編集します。詳細は『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。関連情報は、3-27 ページの「[OC4J での JSP 構成パラメータの設定](#)」および 3-30 ページの「[JSP 関連の OC4J 構成パラメータの設定](#)」を参照してください。



---

## プログラミングに関する基本的な考慮事項

この章では、JSP ページのプログラミングに関する基本的な考慮事項について、例を示して説明します。JSP とサーブレット間の相互作用およびデータベース・アクセスについても説明します。

この章には、次の項目が含まれます。

- JSP とサーブレット間の相互作用
- JSP データ・アクセスに関するサポートと機能
- JSP のリソース管理
- 実行時エラーの処理

## JSP とサーブレット間の相互作用

JSP ページのコーディングは多くの点で便利ですが、サーブレットのコールが必要な場合があります。その一例は、6-17 ページの「[JSP ページでバイナリ・データを回避する理由](#)」で説明するように、バイナリ・データを出力する場合です。

このため、サーブレットと JSP ページ間での往復が、1 つのアプリケーション内で必要になる場合があります。次の各項で、その方法を説明します。

- [JSP ページからのサーブレットの起動](#)
- [JSP ページから起動したサーブレットへのデータの受渡し](#)
- [サーブレットからの JSP ページの起動](#)
- [JSP ページとサーブレット間でのデータの受渡し](#)
- [JSP とサーブレット間の相互作用のサンプル](#)

---

---

**重要：** ここでの説明は、OC4J (サーブレット 2.3) など、サーブレット 2.2 以上の環境を前提にしています。Apache JServ 環境とその他のサーブレット 2.0 環境に関する考慮事項については、このマニュアルの該当する項を参照してください。

---

---

### JSP ページからのサーブレットの起動

ある JSP ページから別の JSP ページを起動する場合と同様に、`jsp:include` 操作タグと `jsp:forward` 操作タグを使用して、JSP ページからサーブレットを起動できます (1-15 ページの「[標準アクション:JSP タグ](#)」を参照)。次に例を示します。

```
<jsp:include page="/servlet/MyServlet" flush="true" />
```

ページの実行中にこの文が出現すると、ページ・バッファがブラウザに出力され、サーブレットが実行されます。サーブレットの実行が終了すると、制御が JSP ページに戻されて、ページの実行が続行されます。この機能は、JSP ページ間における `jsp:include` 操作タグと同じ機能です。

また、JSP ページ間における `jsp:forward` 操作タグと同様に、次の文は、ページ・バッファをクリアし、JSP ページの実行を終了して、サーブレットを実行します。

```
<jsp:forward page="/servlet/MyServlet" />
```

---

---

**重要：** JServ またはサーブレット 2.0 環境では、サーブレットへのインクルードまたは転送はできません。JSP ラッパー・ページを書き込む必要があります。詳細は、B-19 ページの「[JServ での動的なインクルードと転送](#)」を参照してください。

---

---

## JSP ページから起動したサーブレットへのデータの受渡し

JSP ページからサーブレットに動的にインクルードまたは転送を行う場合は、`jsp:param` タグを使用して、サーブレットにデータを渡すことができます（別の JSP ページへのインクルードまたは転送でも同様です）。

`jsp:param` タグは、`jsp:include` タグまたは `jsp:forward` タグ内で使用できます。次に例を示します。

```
<jsp:include page="/servlet/MyServlet" flush="true" >
  <jsp:param name="username" value="Smith" />
  <jsp:param name="userempno" value="9876" />
</jsp:include>
```

`jsp:param` タグの詳細は、1-15 ページの「[標準アクション: JSP タグ](#)」を参照してください。

適切なスコープの `JavaBean`、または `HTTP` リクエスト・オブジェクトの属性を使用して、JSP ページとサーブレットとの間でデータの受渡しを行うこともできます。リクエスト・オブジェクトの属性の使用方法は、4-4 ページの「[JSP ページとサーブレット間でのデータの受渡し](#)」で説明します。

## サーブレットからの JSP ページの起動

標準の `javax.servlet.RequestDispatcher` インタフェースの機能を使用すると、サーブレットから JSP ページを起動できます。この機能を使用するには、次の手順に従ってコードを作成します。

1. サーブレット・インスタンスからサーブレット・コンテキスト・インスタンスを取得します。

```
ServletContext sc = this.getServletContext();
```

2. サーブレット・コンテキスト・インスタンスからリクエスト・ディスパッチャを取得し、ターゲットの JSP ページのページ相対パスまたはアプリケーション相対パスを `getRequestDispatcher()` メソッドへの入力として指定します。

```
RequestDispatcher rd = sc.getRequestDispatcher("/jsp/mypage.jsp");
```

この手順の実行前か実行中に、`HTTP` リクエスト・オブジェクトの属性を必要に応じて使用して、JSP ページにデータを受け渡すことができます。詳細は、次項の「[JSP ページとサーブレット間でのデータの受渡し](#)」を参照してください。

3. リクエスト・ディスパッチャの `include()` メソッドまたは `forward()` メソッドを起動し、HTTP リクエスト・オブジェクトとレスポンス・オブジェクトを引数として指定します。例:

```
rd.include(request, response);
```

または

```
rd.forward(request, response);
```

これらのメソッドの機能は、`jsp:include` タグおよび `jsp:forward` タグの機能と同じです。`include()` メソッドは一時的に制御を移すのみで、後で、起動したサーブレットに実行の制御が戻されます。

`forward()` メソッドは、出力バッファをクリアすることに注意してください。

---

---

**注意：** リクエスト・オブジェクトとレスポンス・オブジェクトは、標準のサーブレット機能 (`javax.servlet.http.HttpServlet` クラスに指定されている `doGet()` メソッドなど) を使用して、事前に取得されています。

---

---

## JSP ページとサーブレット間でのデータの受渡し

前の項の「サーブレットからの JSP ページの起動」で説明したように、リクエスト・ディスパッチャを使用してサーブレットから JSP ページを起動するとき、HTTP リクエスト・オブジェクトを必要に応じて使用して、データを受け渡すことができます。この操作は、次のいずれかの方法で実行できます。

- `name=value` のペアを持つ「?」構文を使用してリクエスト・ディスパッチャを取得すると、問合せ文字列を URL に追加できます。例:

```
RequestDispatcher rd =  
    sc.getRequestDispatcher("/jsp/mypage.jsp?username=Smith");
```

ターゲットの JSP ページ (またはサーブレット) では、暗黙的な `request` オブジェクトの `getParameter()` メソッドを使用すると、パラメータ・セットの値を取得できません。

- HTTP リクエスト・オブジェクトの `setAttribute()` メソッドを使用できます。例:

```
request.setAttribute("username", "Smith");  
RequestDispatcher rd = sc.getRequestDispatcher("/jsp/mypage.jsp");
```

ターゲットの JSP ページまたはサーブレットでは、暗黙的な `request` オブジェクトの `setAttribute()` メソッドを使用すると、パラメータ・セットの値を取得できます。



---

---

**注意：** `jsp:param` タグのかわりに、この項で説明する機能を使用して、JSP ページからサーブレットにデータを受け渡すことができます。

---

---

## JSP とサーブレット間の相互作用のサンプル

この項では、前の各項で説明した機能を使用した JSP ページとサーブレットのサンプルを示します。JSP ページの `Jsp2Servlet.jsp` には、サーブレットの `MyServlet` がインクルードされ、このサーブレットには別の JSP ページの `welcome.jsp` がインクルードされます。

### Jsp2Servlet.jsp のコード

```
<HTML>
<HEAD> <TITLE> JSP Calling Servlet Demo </TITLE> </HEAD>
<BODY>

<!-- Forward processing to a servlet -->
<% request.setAttribute("empid", "1234"); %>
<jsp:include page="/servlet/MyServlet?user=Smith" flush="true"/>

</BODY>
</HTML>
```

### MyServlet.java のコード

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.PrintWriter;
import java.io.IOException;

public class MyServlet extends HttpServlet {

    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException {
        PrintWriter out= response.getWriter();
        out.println("<B><BR>User:" + request.getParameter("user"));
        out.println
            ("", Employee number:" + request.getAttribute("empid") + "</B>");
        this.getServletContext().getRequestDispatcher
            ("/jsp/welcome.jsp").include(request, response);
    }
}
```

### welcome.jsp のコード

```
<HTML>
<HEAD> <TITLE> The Welcome JSP </TITLE> </HEAD>
<BODY>

<H3> Welcome! </H3>
<P><B> Today is <%= new java.util.Date() %>. Have a nice day! </B></P>
</BODY>
</HTML>
```

## JSP データ・アクセスに関するサポートと機能

次の各項では、データへのアクセス時に考慮する必要がある OC4J JSP と Oracle の機能について説明します。

- [データ・アクセスに対する JSP サポートの概要](#)
- [JDBC を使用した JSP データ・アクセスのサンプル](#)
- [JDBC パフォーマンス強化機能の使用](#)
- [JSP ページからの EJB コール](#)
- [Oracle SQLJ の JSP サポート](#)
- [OracleXMLQuery クラス](#)

### データ・アクセスに対する JSP サポートの概要

JDBC API は単なる一連の Java インタフェースであるため、JavaServer Pages のテクノロジーは、JSP スクリプトレット内でのこの API の使用を直接サポートします。

Oracle JDBC には、いくつかのドライバが用意されています。1) JDBC OCI ドライバは、Oracle クライアントのインストールで使用します。2) 100% Java の JDBC Thin ドライバは、基本的にすべてのクライアント側（アプレットも含む）で使用できます。3) JDBC サーバー側 Thin ドライバは、Oracle Database インスタンスに別の Oracle Database インスタンス内からアクセスするために使用します。4) JDBC サーバー側内部ドライバは、Java コードが（Java スタッド・プロセスなどから）実行されているデータベースにアクセスするために使用します。このマニュアルでは、JDBC に関する基本的な知識があることを前提としています。『Oracle9i 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。

OC4J JSP コンテナは、SQLJ（Java に埋め込まれた SQL）とともに、EJB コールもサポートします。

さらに、JavaServer Pages 標準タグ・ライブラリ（JSTL）の SQL タグ、および OC4J で提供される JavaBeans およびカスタム SQL タグがあります。詳細は、『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』を参照してください。

## JDBC を使用した JSP データ・アクセスのサンプル

次の例では、ユーザーが HTML フォームを使用して入力（ボックスに入力し「Ask Oracle」ボタンをクリック）した検索条件から、動的に問合せを作成します。指定の問合せを実行するには、JSP 宣言で定義した `runQuery()` というメソッドで、JDBC コードを使用します。また、出力を作成するには、JSP 宣言で `formatResult()` メソッドを定義します。`runQuery()` メソッドでは、`scott` スキーマとパスワード `tiger` を使用します

HTML の `INPUT` タグによって、フォームに入力された文字列は `cond` と命名されます。したがって、`cond` は、この HTTP リクエストの暗黙的な `request` オブジェクトの `getParameter()` メソッドに対する入力パラメータであり、`runQuery()` メソッドに対する入力パラメータでもあります（このメソッドによって、`cond` 文字列が問合せの `WHERE` 句に配置されます）。

---



---

### 注意：

- この例では、`<%!...%>` 宣言構文を使用して `runQuery()` メソッドが定義されていますが、かわりに、`<%...%>` スクリプトレット構文を使用することもできます。
  - この例では、JDBC OCI ドライバを使用するため、Oracle クライアントのインストールが必要です。このサンプルを実行する場合は、適切な JDBC ドライバと接続文字列を使用してください。
- 
- 

```
<%@ page language="java" import="java.sql.*" %>
<HTML>
<HEAD> <TITLE> The JDBCQuery JSP </TITLE> </HEAD>
<BODY BGCOLOR="white">
<% String searchCondition = request.getParameter("cond");
   if (searchCondition != null) { %>
       <H3> Search results for <I> <%= searchCondition %> </I> </H3>
       <B> <%= runQuery(searchCondition) %> </B> <HR><BR>
<% } %>
<B>Enter a search condition:</B>
<FORM METHOD="get">
<INPUT TYPE="text" NAME="cond" SIZE=30>
<INPUT TYPE="submit" VALUE="Ask Oracle">
</FORM>
</BODY>
</HTML>
<!-- Declare and define the runQuery() method. --%>
<%! private String runQuery(String cond) throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rset = null;
    try {
```

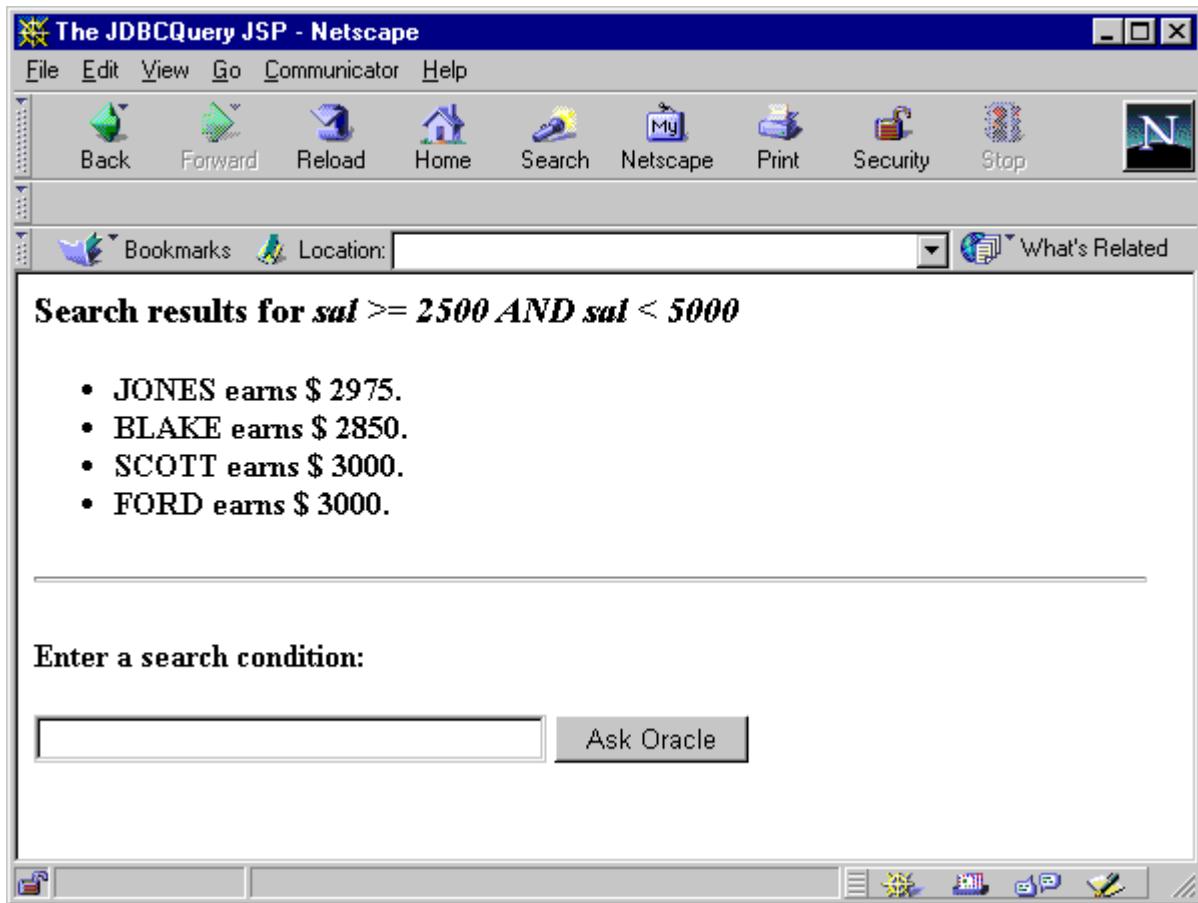
```
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection("jdbc:oracle:oci:@",
                                          "scott", "tiger");

        stmt = conn.createStatement();
        // dynamic query
        rset = stmt.executeQuery ("SELECT ename, sal FROM scott.emp "+
                                  (cond.equals("") ? "" : "WHERE " + cond ));
        return (formatResult(rset));
    } catch (SQLException e) {
        return ("<P> SQL error: <PRE> " + e + " </PRE> </P>¥n");
    } finally {
        if (rset!= null) rset.close();
        if (stmt!= null) stmt.close();
        if (conn!= null) conn.close();
    }
}

private String formatResult(ResultSet rset) throws SQLException {
    StringBuffer sb = new StringBuffer();
    if (!rset.next())
        sb.append("<P> No matching rows.<P>¥n");
    else {
        sb.append("<UL>");
        do {
            sb.append("<LI>" + rset.getString(1) +
                      " earns $" + rset.getInt(2) + ".</LI>¥n");
        } while (rset.next());
        sb.append("</UL>");
    }
    return sb.toString();
}
%>
```

次の図は、次の入力に対するサンプル出力を示しています。

```
sal >= 2500 AND sal < 5000
```



## JDBC パフォーマンス強化機能の使用

OC4J の JSP アプリケーションでは、次のパフォーマンス強化機能を使用できます。これらの機能は、Oracle JDBC 拡張機能によってサポートされます。

- データベース接続のキャッシング
- JDBC 文のキャッシング
- 更新文のバッチ実行
- 問合せ時の行のプリフェッチ
- 行セットのキャッシング

これらのパフォーマンス機能のほとんどは、(DBBean ではなく) データ・アクセス用 JavaBeans の Oracle ConnBean と ConnCacheBean によってサポートされます。これらの Bean については、『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』を参照してください。

### データベース接続のキャッシング

新規のデータベース接続の作成はコストがかかる操作であるため、できるだけ回避する必要があります。かわりに、データベース接続のキャッシュを使用します。JSP アプリケーションは、物理的な接続の既存のプールから論理的な接続を取得し、終了時にその接続をプールに戻すことができます。

接続プールは、4 つの JSP スコープ (application、session、page または request) のいずれかに作成できます。可能な最大のスコープを使用すると最も効率的で、Web サーバーで許可されている場合は application スコープ、それ以外の場合は session スコープを使用します。

JDBC 2.0 の標準拡張機能で指定されている標準接続プーリングに基づいた Oracle JDBC 接続キャッシング・スキームは、OC4J が提供するデータ・アクセス用 JavaBean の ConnCacheBean に実装されています。データ・アクセス用 JavaBean の ConnBean がサポートする標準のデータソース接続プーリング機能も使用できます。これらの Bean については、『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』を参照してください。

Oracle JDBC の OracleConnectionCacheImpl クラスを、JavaBean と同様に、直接使用することもできます。次に例を示します (すべての OracleConnectionCacheImpl 機能は、ConnCacheBean から使用可能です)。

```
<jsp:useBean id="occi" class="oracle.jdbc.pool.OracleConnectionCacheImpl"
            scope="session" />
```

ConnCacheBean と同じプロパティを OracleConnectionCacheImpl で使用できます。このプロパティは、`jsp:setProperty` タグを使用して設定するか、またはクラスの setter メソッドを使用して直接設定できます。

Oracle JDBC の接続キャッシング・スキームと OracleConnectionCacheImpl クラスの詳細は、『Oracle9i 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。

## JDBC 文のキャッシング

Oracle JDBC の拡張機能である文のキャッシング機能は、単一の物理的な接続内（ループ内、または繰り返しコールされるメソッド内）で繰り返し使用される実行文をキャッシングして、パフォーマンスを向上させます。文をキャッシュすると、その文を実行するたびに文を再解析したり、文オブジェクトを再作成したり、パラメータ・サイズ定義を再計算する必要はありません。

Oracle JDBC の文のキャッシング・スキームは、OC4J が提供するデータ・アクセス用 JavaBeans の ConnBean と ConnCacheBean に実装されています。これらの Bean には、それぞれ `stmtCacheSize` プロパティがあり、`jsp:setProperty` タグまたは Bean の `setStmtCacheSize()` メソッドを使用して設定できます。Bean の詳細は、『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』を参照してください。

文のキャッシング機能は、Oracle JDBC の OracleConnection クラスおよび OracleConnectionCacheImpl クラスからも直接使用できます。Oracle JDBC の文のキャッシング・スキーム、および OracleConnection クラスと OracleConnectionCacheImpl クラスの詳細は、『Oracle9i 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。

---

**重要：** 文は、単一の物理的な接続内でのみキャッシュできます。接続キャッシュに対して文のキャッシング機能を有効にすると、プールされた単一の接続オブジェクトから複数の論理接続オブジェクト間で文をキャッシュできます。ただし、プールされた複数の接続オブジェクト間ではキャッシュできません。

---

## バッチ更新

Oracle JDBC のバッチ更新機能は、バッチの値（制限）を、プリコンパイルされた SQL 文の各オブジェクトに関連付けます。execute メソッドがコールされるたびにプリコンパイルされた SQL 文を実行する JDBC ドライバのかわりにバッチ更新を使用すると、ドライバは、その文を累積した実行リクエストのバッチに追加します。バッチの値に達すると、ドライバは、実行のためにすべての操作をデータベースに渡します。たとえば、バッチの値が 10 の場合は、10 個の操作の各バッチがデータベースに送信され、1 回の送信ですべての操作が処理されます。

OC4J は、データ・アクセス用 JavaBean の ConnBean の executeBatch プロパティを使用して、Oracle JDBC のバッチ更新を直接サポートします。このプロパティは、

`jsp:setProperty` タグまたは Bean の setter メソッドを使用して設定できます。かわりに `ConnCacheBean` を使用している場合は、接続内または作成する文オブジェクト内で Oracle JDBC 機能を使用して、バッチ更新を有効にできます。これらの Bean については、『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』を参照してください。

Oracle JDBC のバッチ更新の詳細は、『Oracle9i 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。

### 行のプリフェッチ

問合せ結果セットを移入する場合、Oracle JDBC の行のプリフェッチ機能によって、データベースへのトリップごとにクライアントにプリフェッチする行数を決定できます。これによって、サーバーへのラウンドトリップ回数が減少します。

OC4J は、データ・アクセス用 JavaBean の `ConnBean` の `preFetch` プロパティを使用して、Oracle JDBC の行のプリフェッチを直接サポートします。このプロパティは、`jsp:setProperty` タグまたは Bean の setter メソッドを使用して設定できます。かわりに `ConnCacheBean` を使用している場合は、接続内または作成する文オブジェクト内で Oracle JDBC 機能を使用して、行のプリフェッチを有効にできます。これらの Bean については、『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』を参照してください。

Oracle JDBC の行のプリフェッチについては、『Oracle9i 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。

### 行セットのキャッシング

キャッシュ内の行セットは、切断された、シリアライズ可能でスクロール可能なコンテナを、取得したデータに提供します。この機能は、頻繁に変更しない小規模なデータ・セットに対して有効です。特に、クライアントが頻繁かつ継続的に情報へのアクセスを要求する場合に便利です。これに対して、標準の結果セットを使用する場合は、基礎になる接続とリソースを保持する必要があります。ただし、キャッシュ内の行セットが大規模な場合は、アプリケーション・サーバーのメモリーを大量に消費することに注意してください。

Oracle Database の Oracle JDBC 実装では、キャッシュ内の行セットを提供します。Oracle JDBC ドライバを使用している場合は、次の例に示すように、JSP ページ内でコードを使用して、キャッシュ内の行セットを作成および移入します。

```
CachedRowSet crs = new CachedRowSet();  
crs.populate(rset); // rset is a previously created JDBC ResultSet object.
```

行セットが移入されると、元の結果セットの取得時に使用した、接続と文オブジェクトがクローズされます。

Oracle JDBC のキャッシュ内の行セットについては、『Oracle9i 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。



## JSP ページからの EJB コール

JSP ページは、EJB をコールして追加処理またはデータ・アクセスを実行できます。典型的なアプリケーション設計では、クライアント・リクエストの初期処理を行うために JavaServer Pages をフロントエンドとして使用し、Enterprise JavaBeans をコールして、データ・ソースに対する読取りや書込みなどの作業を実行します。次の各項では、EJB の使用方法の概要について説明します。

- EJB の構成とデプロイの概要
- EJB コールのコード手順とアプローチ
- OC4J の EJB タグ・ライブラリの使用

### EJB の構成とデプロイの概要

JSP ページから EJB をコールするための構成およびデプロイ手順は、サーブレットから EJB をコールする手順と似ています。この手順に関しては、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。手順は次のとおりです。

- JSP ページからコールする EJB ごとに、アプリケーションの web.xml ファイルに <ejb-ref> 要素を定義します。
- EJB の型を指定するための適切なサブ要素 (<session> や <entity> など) を持つ <enterprise-beans> 要素が含まれた、ejb-jar.xml デプロイメント・ディスクリプタを作成します。サブ要素内で、コールする各 EJB の名前、クラス名などの詳細を指定します。
- ejb-jar.xml ファイルを EJB アーカイブにパッケージングします。デプロイ要件は、サーブレットの要件とほとんど同じです。

### EJB コールのコード手順とアプローチ

JSP ページで EJB を起動する主な手順は、次のとおりです。

1. Bean ホームとリモート・インタフェース用の EJB パッケージを、EJB コールを行う各 JSP ページにインポートします。page ディレクティブを使用します。
2. JNDI を使用して、EJB ホーム・インタフェースをルックアップします。
3. ホームから EJB リモート・オブジェクトを作成します。
4. リモート・オブジェクトでビジネス・メソッドを起動します。

JSP ページでは、ほとんどのサーブレット・コードをスクリプトレットのフォームで使用できるため、サーブレットで使用するコードと同じコードをスクリプトレットで使用すると、EJB を JSP ページから簡単にコールできます。この方法は、手順 2～4 を実行する 1 つの方法です。

または、OC4J が提供する EJB タグ・ライブラリのタグを使用することもできます。詳細は、次項の「[OC4J の EJB タグ・ライブラリの使用](#)」を参照してください。タグによって、コーディングが簡素化されます。基本的に、タグを使用すると、JSP ページで共通に使用する通常の JavaBeans と同様に、Enterprise JavaBeans を処理できます。

### OC4J の EJB タグ・ライブラリの使用

前項の「[EJB コールのコード手順とアプローチ](#)」を参照してください。前項で説明したように、page ディレクティブで適切なパッケージをインポートします。次の手順に従って、OC4J の EJB タグを使用します。

- taglib ディレクティブを使用して、使用するタグの接頭辞とタグ・ライブラリ・ディレクトリ（TLD）を指定します。
- 手順 2 では、EJB の useHome タグを使用します。
- 手順 3 では、EJB の useBean タグ内で、EJB の createBean タグを使用できます。
- 手順 4 では、EJB の iterate タグによって、ビジネス・メソッドを EJB オブジェクトのコレクションの各メンバーに適用できます。通常は find メソッドによって戻されます。

EJB タグ・ライブラリと詳細なタグの構文については、『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』を参照してください。

タグ・ライブラリを使用する場合とスクリプトレット・コードを使用する場合のデプロイ要件は同じです。他のタグ・ライブラリと同様に、TLD とライブラリ・サポート・クラス（タグ・ハンドラ・クラスとタグ補足情報クラス）は、アプリケーションに対してアクセス可能であることが必要です。

## Oracle SQLJ の JSP サポート

---

---

**重要：** SQLJ がサポートされるのは、現行リリースまでです。

---

---

SQLJ は、静的な SQL の命令を Java コードに直接埋め込むための標準構文です。これによって、データベース・アクセスのプログラミングが大幅に簡素化されます。OC4J JSP コンテナは Oracle SQLJ 実装をサポートしているため、JSP 文で SQLJ 構文を使用できます。SQLJ 文は、#sql トークンで示されます。Oracle SQLJ のデータベース・アクセスは、通常、Oracle JDBC ドライバを介して行われます。

Oracle SQLJ のプログラミング機能、構文およびコマンドライン・オプションに関する一般情報は、『Oracle9i SQLJ 開発者ガイドおよびリファレンス』を参照してください。

## SQLJ JSP コード例

次に、JSP ページの SQLJ のサンプルを示します。page ディレクティブによって、通常 SQLJ に必要なクラスがインポートされます。

```
<%@ page language="sqlj"
    import="sqlj.runtime.ref.DefaultContext,oracle.sqlj.runtime.Oracle" %>
<HTML>
<HEAD> <TITLE> The SQLJQuery JSP </TITLE> </HEAD>
<BODY BGCOLOR="white">
<% String empno = request.getParameter("empno");
if (empno != null) { %>
<H3> Employee # <%=empno %> Details: </H3>
<%= runQuery(empno) %>
<HR><BR>
<% } %>
<B>Enter an employee number:</B>
<FORM METHOD="get">
<INPUT TYPE="text" NAME="empno" SIZE=10>
<INPUT TYPE="submit" VALUE="Ask Oracle">
</FORM>
</BODY>
</HTML>
<%!
```

```
private String runQuery(String empno) throws java.sql.SQLException {
    DefaultContext dctx = null;
    String ename = null; double sal = 0.0; String hireDate = null;
    StringBuffer sb = new StringBuffer();
    try {
        dctx = Oracle.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");
        #sql [dctx] {
            SELECT ename, sal, TO_CHAR(hiredate,'DD-MON-YYYY')
            INTO :ename, :sal, :hireDate
            FROM scott.emp WHERE UPPER(empno) = UPPER(:empno) };
        sb.append("<BLOCKQUOTE><BIG><B><PRE>¥n");
        sb.append("Name : " + ename + "¥n");
        sb.append("Salary : " + sal + "¥n");
        sb.append("Date hired : " + hireDate);
        sb.append("</PRE></B></BIG></BLOCKQUOTE>");
    } catch (java.sql.SQLException e) {
        sb.append("<P> SQL error: <PRE> " + e + " </PRE> </P>¥n");
    } finally {
        if (dctx!= null) dctx.close();
    }
}
```

```

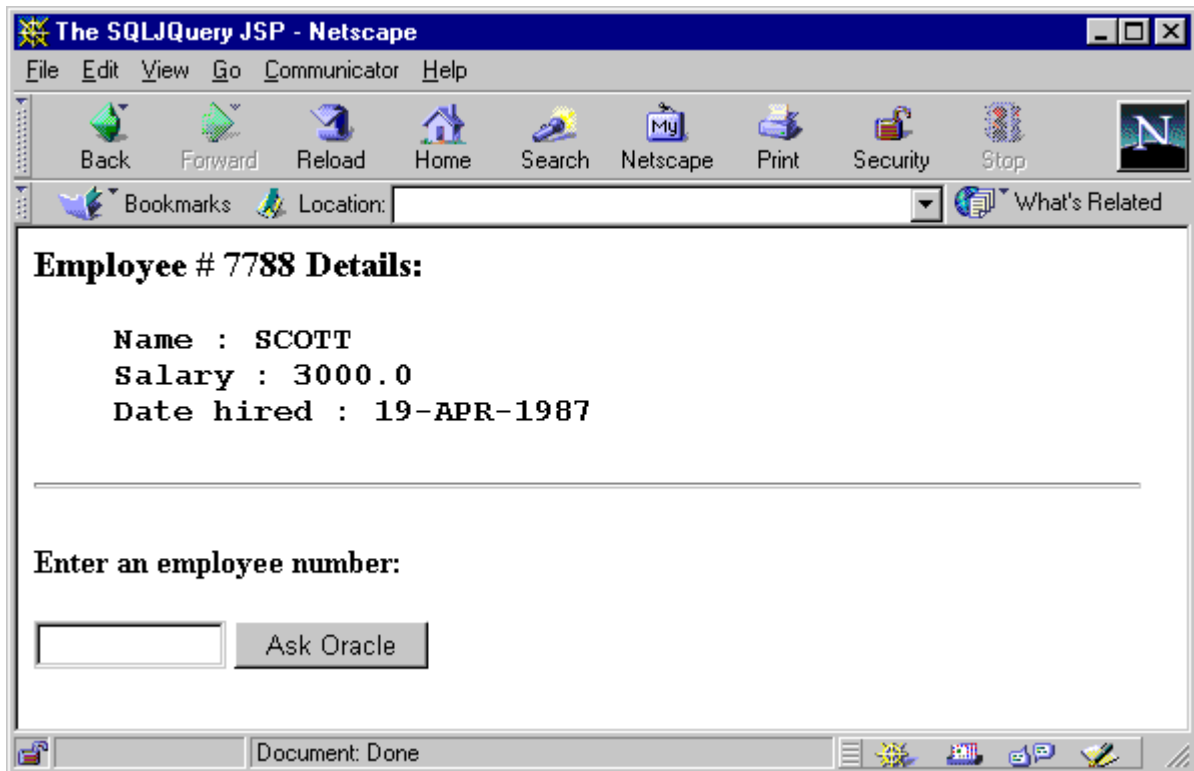
        return sb.toString();
    }

    %>

```

この例では、JDBC OCI ドライバを使用するため、Oracle クライアントのインストールが必要です。接続の取得に使用する Oracle クラスは、Oracle SQLJ 実装で提供されます。

従業員番号 7788 を入力すると、次のよう出力されます。



---

---

**注意：**

- JSP ページを同じ JVM 内で繰り返して起動する場合は、デフォルトの接続コンテキストではなく、例で使用している `dctx` などの明示的な接続コンテキストを常に使用することをお勧めします (`dctx` はローカルのメソッド変数です)。
  - Oracle SQLJ 実装を使用する場合、OC4J JSP コンテナには SQLJ リリース 8.1.6.1 以上が必要です。
- 
- 

## SQLJ トランスレータのトリガー

次のいずれかの方法で、OC4J の JSP トランスレータをトリガーして Oracle SQLJ トランスレータを起動します。

- ファイル名拡張子 `.sqljsp` を JSP ソース・ファイルに対して使用する方法  
または
- `language="sqlj"` を page ディレクティブに指定する方法

いずれの場合も、JSP トランスレータは、`.java` ファイルではなく、`.sqlj` ファイルを生成します。次に、Oracle SQLJ トランスレータが起動し、この `.sqlj` ファイルを `.java` ファイルに変換します。

SQLJ を使用すると、追加の出力ファイルが生成されます。詳細は、7-6 ページの「[生成されるファイルとその格納場所](#)」を参照してください。

---

---

**重要：**

- SQLJ を使用するには、環境に適した SQLJ JAR ファイルをインストールし、クラスパスに追加する必要があります。3-12 ページの「[OC4J が提供する主なサポート・ファイル](#)」を参照。
  - 同じアプリケーション内では、同じベース・ファイル名を `.jsp` ファイルと `.sqljsp` ファイルに対して使用しないでください。これは、同じ名前を使用すると、クラス名と `.java` ファイル名が重複して生成されるためです。
- 
-

## Oracle SQLJ オプションの設定

SQLJ の JSP ページを実行または事前変換するときには、任意の Oracle SQLJ オプション設定を指定できます。次のように、オンデマンド変換や事前変換の場合にも指定できます。

- オンデマンド変換の場合は、JSP の `sqljcmd` 構成パラメータを使用します。このパラメータによって、特定の SQLJ トランスレータ実行可能ファイルを指定でき、さらに SQLJ コマンドライン・オプションも設定できます。

`sqljcmd` の詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

- `ojspc` 事前変換ツールを使用して事前変換を行う場合は、`ojspc -s` オプションを使用します。このオプションによって、SQLJ コマンドライン・オプションを設定できます。

詳細は、7-19 ページの「[ojspc のコマンドライン構文](#)」および 7-20 ページの「[ojspc のオプションの説明](#)」を参照してください。

## OracleXMLQuery クラス

`oracle.xml.sql.query.OracleXMLQuery` クラスは、データベース問合せで使用する XML 機能に対する Oracle Database XML SQL Utility の一部です。このクラスには `xsu12.jar` (JDK 1.1.x の場合は `xsu11.jar`) ファイルが必要です。このファイルは、OC4J が提供する一部のカスタム・タグと JavaBeans の XML 機能にも必要です。このファイルは、Oracle Database および Oracle Application Server で提供されます。

OracleXMLQuery クラス、および XML SQL Utility の他の機能については、『Oracle XML Developer’s Kit プログラマーズ・ガイド』を参照してください。

## JSP のリソース管理

次の各項では、リソース管理に関する標準機能と Oracle の付加価値機能について説明します。

- [標準セッションのリソース管理 : HttpSessionBindingListener](#)
- [リソース管理のための Oracle の付加価値機能の概要](#)

### 標準セッションのリソース管理 : HttpSessionBindingListener

JSP ページでは、実行中に取得したリソース (JDBC 接続、文、結果セット・オブジェクトなど) の適切な管理が必要です。標準の `javax.servlet.http` パッケージには、`session` スコープのリソースを管理するために、`HttpSessionBindingListener` インタフェースと `HttpSessionBindingEvent` クラスが用意されています。たとえば、`session` スコープの問合せ Bean はこの機能を使用して、Bean がインスタンス化されるときにデータベース・カーソルを取得し、HTTP セッションが終了するときにそのカーソルをクローズできます (4-7 ページの「[JDBC を使用した JSP データ・アクセスのサンプル](#)」の例では、問合せごとに接続をオープンしてクローズするため、オーバーヘッドが増大します)。

この項では、`HttpSessionBindingListener` の `valueBound()` メソッドと `valueUnbound()` メソッドの使用方法について説明します。

---

---

**注意：** Bean インスタンスは、HTTP セッション・オブジェクトのイベント通知リストに登録する必要がありますが、`jsp:useBean` 文によって自動的に登録されます。

---

---

## valueBound() メソッドと valueUnbound() メソッド

`HttpSessionBindingListener` インタフェースを実装するオブジェクトは、`valueBound()` メソッドと `valueUnbound()` メソッドを実装できます。いずれのメソッドも、`HttpSessionBindingEvent` インスタンスを入力として取得します。これらのメソッドは、サーブレット・コンテナによってコールされます。`valueBound()` メソッドは、オブジェクトがセッションに格納されるときにコールされ、`valueUnbound()` メソッドは、オブジェクトがセッションから削除されるか、あるいはセッションがタイムアウトまたは無効になるとコールされます。開発者は通常、オブジェクトが保持するリソースを解放するために `valueUnbound()` メソッドを使用します（後述の例では、データベース接続の解放で使用します）。

次の「[JDBCQueryBean JavaBean コード](#)」の項では、`HttpSessionBindingListener` を実装する `JavaBean` のサンプルとその `Bean` をコールする JSP ページのサンプルを示します。

## JDBCQueryBean JavaBean コード

次に、`JDBCQueryBean` のサンプル・コードを示します。これは、`HttpSessionBindingListener` インタフェースを実装する `JavaBean` です。ここでは、データベース接続用に `JDBC OCI ドライバ` を使用しています。このサンプル・コードを実行する場合は、適切な `JDBC ドライバ` と接続文字列を使用してください。

`JDBCQueryBean` は、HTML リクエストから検索条件を取得し（[4-21 ページの「UseJDBCQueryBean JSP ページ」](#)を参照）、その検索条件に基づいて動的な問合せを実行し、結果を出力します。

このクラスは、セッション終了時にデータベース接続をクローズする `valueUnbound()` メソッド（`HttpSessionBindingListener` インタフェースで指定されます）も実装します。

```
package mybeans;

import java.sql.*;
import javax.servlet.http.*;

public class JDBCQueryBean implements HttpSessionBindingListener
{
    String searchCond = "";
    String result = null;
```

```
public void JDBCQueryBean() {
}

public synchronized String getResult() {
    if (result != null) return result;
    else return runQuery();
}

public synchronized void setSearchCond(String cond) {
    result = null;
    this.searchCond = cond;
}

private Connection conn = null;

private String runQuery() {
    StringBuffer sb = new StringBuffer();
    Statement stmt = null;
    ResultSet rset = null;
    try {
        if (conn == null) {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:oci8:@",
                                             "scott", "tiger");
        }

        stmt = conn.createStatement();
        rset = stmt.executeQuery ("SELECT ename, sal FROM scott.emp "+
                                 (searchCond.equals("") ? "" : "WHERE " + searchCond ));
        result = formatResult(rset);
        return result;

    } catch (SQLException e) {
        return ("<P> SQL error: <PRE> " + e + " </PRE> </P>¥n");
    }
    finally {
        try {
            if (rset != null) rset.close();
            if (stmt != null) stmt.close();
        }
        catch (SQLException ignored) {}
    }
}

private String formatResult(ResultSet rset) throws SQLException {
    StringBuffer sb = new StringBuffer();
```



```

    if (!rset.next())
        sb.append("<P> No matching rows.<P>¥n");
    else {
        sb.append("<UL><B>");
        do { sb.append("<LI>" + rset.getString(1) +
            " earns $" + rset.getInt(2) + "</LI>¥n");
        } while (rset.next());
        sb.append("</B></UL>");
    }
    return sb.toString();
}

public void valueBound(HttpSessionBindingEvent event) {
    // do nothing -- the session-scope bean is already bound
}

public synchronized void valueUnbound(HttpSessionBindingEvent event) {
    try {
        if (conn != null) conn.close();
    }
    catch (SQLException ignored) {}
}
}

```

---

**注意：** 前述のコードは、サンプル用です。大規模な Web アプリケーションでデータベース接続のプーリングを処理する方法としては、必ずしもお薦めできません。

---

## UseJDBCQueryBean JSP ページ

次の JSP ページでは、前の「[JDBCQueryBean JavaBean コード](#)」の項で定義した JDBCQueryBean JavaBean を使用し、session スコープを使用して Bean を起動します。ここでは、JDBCQueryBean を使用して、ユーザーが入力した検索条件と一致する従業員名を表示します。

JDBCQueryBean は、この JSP ページの `jsp:setProperty` タグから検索条件を取得します。このタグは、ユーザーが HTML フォームを使用して入力した `searchCond` リクエスト・パラメータの値に従って、Bean の `searchCond` プロパティを設定します。HTML の INPUT タグによって、フォームに入力された検索条件は `searchCond` と命名されます。

```

<jsp:useBean id="queryBean" class="mybeans.JDBCQueryBean" scope="session" />
<jsp:setProperty name="queryBean" property="searchCond" />

```

```

<HTML>
<HEAD> <TITLE> The UseJDBCQueryBean JSP </TITLE> </HEAD>

```

```
<BODY BGCOLOR="white">

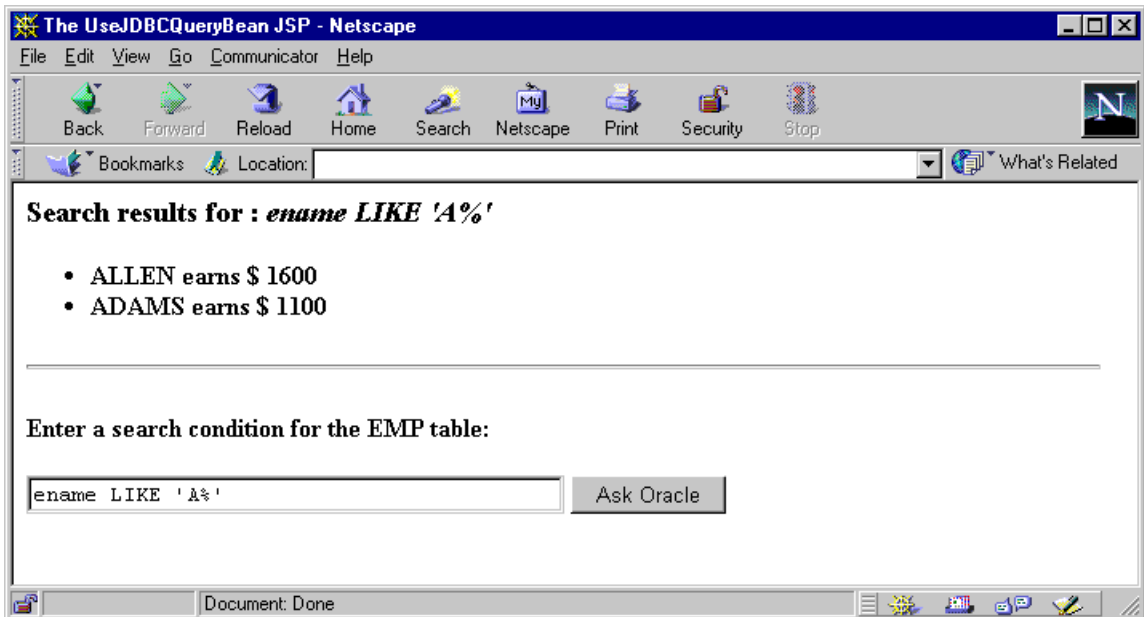
<% String searchCondition = request.getParameter("searchCond");
   if (searchCondition != null) { %>
       <H3> Search results for : <I> <%= searchCondition %> </I> </H3>
       <%= queryBean.getResult() %>
       <HR><BR>
   <% } %>

<B>Enter a search condition for the EMP table:</B>

<FORM METHOD="get">
<INPUT TYPE="text" NAME="searchCond" VALUE="ename LIKE 'A%' " SIZE="40">
<INPUT TYPE="submit" VALUE="Ask Oracle">
</FORM>

</BODY>
</HTML>
```

次に、このページの入力と出力のサンプルを示します。



## HttpSessionBindingListener のメリット

前述の例では、HttpSessionBindingListener 機能のかわりに、JavaBean の finalize() メソッドで接続をクローズできます。finalize() メソッドは、セッションのクローズ後に Bean のガベージ・コレクションが実行されるとコールされます。ただし、HttpSessionBindingListener インタフェースの動作は、finalize() メソッドよりも確実に予測できます。ガベージ・コレクションの頻度は、アプリケーションのメモリー消費パターンによって異なります。これに対して、HttpSessionBindingListener インタフェースの valueUnbound() メソッドは、セッションの停止時に確実にコールされます。

## リソース管理のための Oracle の付加価値機能の概要

OC4J などのサーブレット 2.3 環境では、OC4J JSP は JspScopeListener インタフェースを提供し、application スコープ、session スコープ、request スコープまたは page スコープのリソースを管理します。

この機能は、サーブレットと JSP 標準に従って、page、request、session または application スコープのオブジェクトをサポートします。他のスコープとともに session スコープをサポートするクラスを作成するために、クラスに JspScopeListener インタフェースと HttpSessionBindingListener インタフェースを実装して、両方のインタフェースを統合できます。OC4J 環境または JServ 環境での page スコープの場合は、Oracle 固有の実行時実装を使用することもできます。

HttpSessionBindingListener の構成と統合方法については、『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』を参照してください。

## 実行時エラーの処理

JSP ページを実行してクライアント・リクエストを処理している間に、ページの内側または外側（コールされた `JavaBean` 内など）で実行時エラーが発生する場合があります。この項ではエラー処理機能について説明し、基本的な例を示します。

## サーブレットと JSP の実行時エラー処理機能

この項では、サーブレット 2.3 および JSP 1.2 の実行時例外の処理機能について説明します。JSP エラー・ページを使用する場合についても説明します。

### 一般的なサーブレット実行時エラー処理機能

JSP ページの実行中に発生した実行時エラーは、標準の Java 例外処理機能によって、次のいずれかの方法で処理されます。

- 標準の Java 例外処理コードを使用して、JSP ページ内の Java スクリプトレットで例外を取得して処理できます。
- JSP ページで例外が取得されない場合、リクエストおよび取得されなかった例外 (`java.lang.Throwable` インスタンス) は、エラー・リソースに転送されます。JSP エラーは、この方法で処理することをお勧めします。この場合、エラーの情報を持つ例外インスタンスは、名前に `javax.servlet.jsp.jspException` を使用し、`setAttribute()` コールを介して、`request` オブジェクトに格納されます。

エラー・リソースの URL は、元の JSP ページで `page` ディレクティブの `errorPage` 属性を設定すると指定できます (`page` ディレクティブなどの JSP ディレクティブの概要は、1-7 ページの「[ディレクティブ](#)」を参照してください)。

サーブレット 2.2 以上の環境では、次の命令を使用して、`web.xml` デプロイメント・ディスクリプタにデフォルトのエラー・ページを指定することもできます。

```
<error-page>
  <error-code>404</error-code>
  <location>/error404.html</location>
</error-page>
```

デフォルトのエラー・リソースの詳細は、Sun 社の Java サーブレット仕様 2.3 を参照してください。

## JSP エラー・ページ

オプションとして、別の JSP ページを、元の JSP ページからの実行時例外のエラー・リソースとして使用する方法があります。JSP エラー・ページには、`isErrorPage="true"` に設定する `page` ディレクティブが必要です。この方法で定義されたエラー・ページは、`web.xml` ファイルで宣言されているエラー・ページよりも優先されます。

エラーの情報を持つ `java.lang.Throwable` インスタンスは、エラー・ページで JSP の暗黙的な `exception` オブジェクトを介してアクセス可能です。このオブジェクトにアクセスできるのはエラー・ページのみです。`exception` オブジェクトなど、JSP の暗黙的なオブジェクトの詳細は、1-12 ページの「[暗黙的なオブジェクト](#)」を参照してください。

元の JSP ページに `autoFlush="true"` (デフォルト設定) の `page` ディレクティブがあり、そのページの `JspWriter` オブジェクトのコンテンツがレスポンスの出力ストリームにすでにフラッシュされている場合、取得されなかった例外をエラー・ページに転送しようとしても、レスポンスをクリアできない可能性があることに注意してください。一部のレスポンスは、ブラウザがすでに受信している可能性があります。

エラー・ページの使用例は、次の「[JSP エラー・ページの例](#)」の項を参照してください。

## JSP エラー・ページの例

次の `nullpointer.jsp` の例では、エラーを生成し、エラー・ページの `myerror.jsp` を使用して、暗黙的な `exception` オブジェクトのコンテンツを出力します。

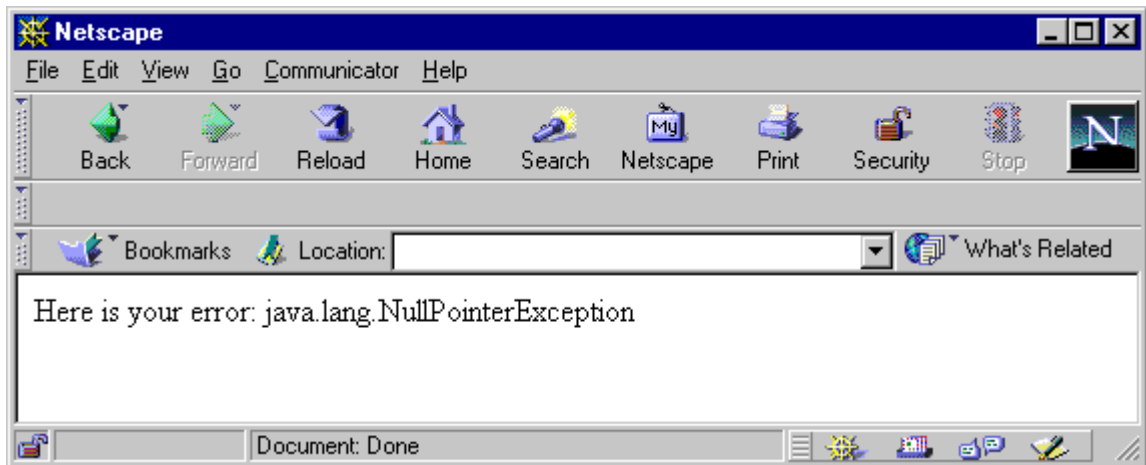
### `nullpointer.jsp` のコード

```
<HTML>
<BODY>
<% page errorPage="myerror.jsp" %>
Null pointer is generated below:
<%
    String s=null;
    s.length();
%>
</BODY>
</HTML>
```

### `myerror.jsp` のコード

```
<HTML>
<BODY>
<%@ page isErrorPage="true" %>
Here is your error:
<%= exception %>
</BODY>
</HTML>
```

この例では、次の内容が出力されます。



---

**注意：** 処理がエラー・ページに転送された場合、`nullpointer.jsp` の「Null pointer is generated below:」の行は出力されません。これは、`jsp:include` 機能と `jsp:forward` 機能の違いを示しています。  
`jsp:forward` の場合は、転送先ページの出力によって、転送元ページの出力が置換されます。

---

---

## JSP XML サポート

JSP 仕様 1.2 で導入された XML に対する追加サポートによって、JavaServer Pages は、XML 文書の作成に効果的なモデルとしてより一層重要視されています。これらの拡張機能によって、JSP テクノロジは XML テクノロジをさらに補完し、XML ツールへのアクセスがより容易になりました。JSP XML サポートによるもう 1 つのメリットは、ページの妥当性チェックがさらに強力で包括的になることです。

この章では、JavaServer Pages による XML のサポートについて説明します。JSP の構文要素に対応する XML スタイルのサポート、および JSP ページの XML ビューの概要についても説明します。これらの機能は JSP 仕様 1.2 に追加されたものです。JSP 仕様 1.1 では、JSP XML 構文に対するサポートがオプションで組み込まれ、構文が定義されていました。

この章には、次の各項が含まれます。

- [JSP XML 文書と JSP XML ビュー: 概要と比較](#)
- [JSP XML 文書の詳細](#)
- [JSP XML ビューの詳細](#)

カスタム・タグを介して OC4J で提供される XML と XSL の JSP サポートの詳細は、『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』を参照してください。

XML に関する一般情報は、次の Web サイトで XML 仕様を参照してください。

<http://www.w3.org/XML/>

## JSP XML 文書と JSP XML ビュー：概要と比較

<%@ page...>ディレクティブ、<%@ include...>ディレクティブ、スクリプトレット用の <%...%>、宣言用の <%!...%>、式用の <%=...%> など、従来の JSP 構成メンバーは、XML 文書内では構文として有効ではありません。この問題は、XML と互換性のある同等の構文を定義することで、JSP 仕様 1.1 で対処されました。ただし、JSP 仕様 1.1 では、JSP コンテナによるこの構文のサポートはオプションでした。

JSP 仕様 1.2 以上では、XML 互換の JSP 構文、追加機能および必要なサポートが、JSP に準拠しているコンテナによってさらに完全にサポートされます。

---

**注意：** Oracle9iAS リリース 2 (9.0.3) より前のリリースでは、OC4J JSP コンテナは、JSP 仕様 1.1 のオプションの XML 代替構文をサポートしていません。この XML 代替構文の実装は、JSP コンテナによって、現在の JSP 仕様で規定されている完全な XML サポートに置換されています。JSP 1.1 の構文自体は変更されていませんが、この章で説明しているように、現在の JSP XML サポートには様々な機能が追加されています。

JSP 仕様 1.1 では、ページ内で従来の構文と XML 代替構文を混在させることができませんでした。これは、現在の仕様には当てはまりません。

---

JSP XML 文書 (JSP 仕様では JSP 文書と呼ばれます) という用語は、この XML 互換の構文を使用する JSP ページを指します。構文には、たとえば、JSP のディレクティブ、宣言、式およびスクリプトレットにかかわって機能するルート要素と複数の要素が含まれています (標準的なタグ操作およびカスタム・タグ操作は、すでに XML 規則に従っています)。詳細は、5-4 ページの「[JSP XML 文書の詳細](#)」を参照してください。

JSP XML 文書は、純粋な XML 構文内で適切に構成されており、名前空間を識別します。JSP XML 文書では、JSP XML コア構文および使用するカスタム・タグ・ライブラリの構文を指定するために、XML 名前空間が使用されます。対照的に、従来の JSP ページは通常、XML 文書ではありません。

JSP XML 文書のファイル名拡張子は、従来の JSP ページと同じ .jsp です。ただし、そのルート要素 (<jsp:root>) のために、JSP コンテナでは XML 文書として認識されます。また、JSP XML 文書に対するセマンティック・モデルも従来のページと同様です。JSP XML 文書は、対応する構文を使用した従来のページと同じ一連の操作と結果を指示します。空白の処理は XSLT 規則に従います。JSP XML 文書のノードが識別されると、空白のみのテキスト・ノードは、テンプレート・データ用の <jsp:text> 要素内である場合を除き、文書から削除されます。<jsp:text> 要素の内容は、そのまま保持されます。

---

**注意：** テンプレート・データは、JSP トランスレータでは解析できないテキストで構成されています。

---



JSP 1.2 環境での JSP XML 文書は、JSP コンテナによって直接処理できます。JSP XML 文書は、XML 開発ツールまたは他の XML ツールで使用することもできます。このようなツールが一般化して普及するにつれて、JSP XML 文書の重要性が増大します。

JSP 仕様での XML サポートのもう 1 つの主な機能は、JSP XML ビューです。仕様の中で、JSP XML ビューは「XML 構文または従来の構文で記述された JSP ページと、それを説明する XML 文書との間のマッピング」として定義されています。JSP XML ビューは、変換時に JSP コンテナによって生成されます。

JSP XML 文書の場合、JSP XML ビューはページ・ソースとほぼ同じです。相違点は、XML ビューが `include` ディレクティブに従って拡張される点です。XML ビューをサポートする JSP コンテナについての他の（オプションの）相違点は、改善されたエラー・レポートに関する ID 属性がすべての XML 要素に追加される点です。

従来の JSP ページの場合、JSP コンテナは、一連の変換を実行してページから XML ビューを作成します。詳細は、5-14 ページの「[JSP XML ビューの詳細](#)」を参照してください。

JSP XML ビューの主要機能は、ページの妥当性チェックです。JSP 仕様 1.2 以降、すべてのタグ・ライブラリがその TLD 内に `<validator>` 要素を保持し、妥当性チェックを実行できるクラスを指定できます。このようなクラスは、タグ・ライブラリ・バリデータ (TLV) ・クラスと呼ばれます。TLV クラスの目的は、タグ・ライブラリを使用する JSP ページが、実装した必要な制約に準拠しているかどうか検証することです。バリデータ・クラスは、その妥当性チェックのソースとして JSP XML ビューを使用します。

つまり、JSP XML 構文を必要に応じて使用して、XML 互換の JSP ページを作成できます。対照的に、JSP XML ビューは JSP コンテナの機能で、ページの妥当性チェックに使用されません。

## JSP XML 文書の詳細

ここでは、JSP XML 文書の構文について、より詳しく説明します。全体の解説は、Sun 社の JSP 仕様を参照してください。

---

---

**重要：** JSP の従来の構文と JSP XML 構文を 1 つのファイルに混在させることはできません。ただし、`include` ディレクティブを使用すると、1 つの変換単位で両方の構文を使用できます。たとえば、従来の JSP ページに JSP XML 文書を含めることができます。

---

---

---

---

**注意：** JSP XML 文書では、`DOCTYPE` 文を使用しません。

---

---

JSP XML 構文には、次の要素を含めることができます。

- ルート要素 `<jsp:root ... >`: JSP XML コア構文用の名前空間仕様、および使用するカスタム・タグ・ライブラリ用の名前空間仕様が含まれます。
- JSP ディレクティブ要素: `page` ディレクティブおよび `include` ディレクティブ用

---

---

**注意：** ルート要素の `xmlns` 属性を使用した個別の機能は、`taglib` ディレクティブを使用した場合と同じです。詳細は、5-7 ページの「[JSP XML ルート要素と JSP XML 名前空間](#)」を参照してください。

---

---

- JSP の宣言要素
- JSP の式要素
- JSP のスクリプトレット要素
- JSP の標準アクションの要素
- JSP のカスタム・アクションの要素
- テキスト要素 `<jsp:text ... >`: テンプレート（静的）データ用
- テンプレート・データに関するその他の XML 要素（必要な場合）

次の項目では、これらのタイプの各要素について説明し、従来の JSP ページと対応する JSP XML 文書との比較例を示します。

## JSP XML 構文のサマリー表

表 5-1 では、JSP XML 構文を要約し、従来の JSP 構文と比較しています（該当する場合）。

**表 5-1 JSP XML 構文と従来の JSP 構文との比較**

| JSP XML 構文  | 対応する従来の JSP 構文                                 |
|---|--|
| ルート要素<br><pre>&lt;jsp:root   xmlns:jsp=...   xmlns:xxx =...   ...   version=... /&gt;</pre> <p>ルート要素は、標準的な JSP XML 名前空間、カスタム・タグ・ライブラリの XML 名前空間、および JSP のバージョン番号（必須）を示します。5-7 ページの「<a href="#">JSP XML ルート要素と JSP XML 名前空間</a>」を参照。</p> | タグ・ライブラリの xmlns 設定は、JSP の taglib ディレクティブと等価です。 |
| JSP の page ディレクティブ要素<br><pre>&lt;jsp:directive.page ... /&gt;</pre> <p>5-8 ページの「<a href="#">JSP XML ディレクティブ要素</a>」を参照。</p>  | <pre>&lt;%@ page ... %&gt;</pre>               |
| JSP の include ディレクティブ要素<br><pre>&lt;jsp:directive.include ... /&gt;</pre> <p>5-8 ページの「<a href="#">JSP XML ディレクティブ要素</a>」を参照。</p>  | <pre>&lt;%@ include ... %&gt;</pre>            |
| JSP の宣言要素<br><pre>&lt;jsp:declaration&gt;   declaration &lt;/jsp:declaration&gt;</pre> <p>5-9 ページの「<a href="#">JSP XML の宣言要素、式要素およびスクリプトレット要素</a>」を参照。</p>  | <pre>&lt;%! declaration %&gt;</pre>            |

表 5-1 JSP XML 構文と従来の JSP 構文との比較 (続き)

| JSP XML 構文   | 対応する従来の JSP 構文  |
|--|---|
| <p>JSP の式要素</p> <pre>&lt;jsp:expression&gt;   expression &lt;/jsp:expression&gt;</pre> <p>5-9 ページの「<a href="#">JSP XML の宣言要素、式要素およびスクリプトレット要素</a>」を参照。</p>         | <pre>&lt;%= expression %&gt;</pre>                            |
| <p>JSP のスクリプトレット要素</p> <pre>&lt;jsp:scriptlet&gt;   code fragment &lt;/jsp:scriptlet&gt;</pre> <p>5-9 ページの「<a href="#">JSP XML の宣言要素、式要素およびスクリプトレット要素</a>」を参照。</p> | <pre>&lt;% code fragment %&gt;</pre>                          |
| <p>JSP の標準アクション (jsp:include や jsp:forward など)</p> <p>5-10 ページの「<a href="#">JSP XML の標準アクションとカスタム・アクションの要素</a>」を参照。</p>  | <p>JSP の標準アクション</p> <p>従来の標準アクションの構文は、すでに XML 互換です。</p>       |
| <p>JSP のカスタム・アクション (カスタム・タグ)</p> <p>5-10 ページの「<a href="#">JSP XML の標準アクションとカスタム・アクションの要素</a>」を参照。</p>  | <p>JSP のカスタム・アクション</p> <p>従来のカスタム・アクションの構文は、すでに XML 互換です。</p> |
| <p>標準アクションまたはカスタム・アクション内のリクエスト時属性の JSP 式</p> <pre>&lt;foo:bar attr="%=expr%" /&gt;</pre> <p>5-10 ページの「<a href="#">JSP XML の標準アクションとカスタム・アクションの要素</a>」を参照。</p>      | <pre>&lt;foo:bar attr="&lt;%=expr%" /&gt;</pre>               |

表 5-1 JSP XML 構文と従来の JSP 構文との比較 (続き)

| JSP XML 構文  | 対応する従来の JSP 構文 |
|---|----------------|
| テキスト要素<br><pre>&lt;jsp:text&gt; ... &lt;/jsp:text&gt;</pre>                   | テンプレート・データ     |
| <p>これはテンプレート・データ用です。5-10 ページの「<a href="#">JSP XML のテキスト要素とその他の要素</a>」を参照。</p> |                |
| <p>その他の XML 要素。&lt;jsp:text&gt; 要素がある場所で使用されている可能性があります。</p>                  | テンプレート・データ     |
| <p>5-10 ページの「<a href="#">JSP XML のテキスト要素とその他の要素</a>」を参照。</p>                  |                |

## JSP XML ルート要素と JSP XML 名前空間

<jsp:root> 要素には、3つの主要な機能があります。

- 文書を JSP XML 文書として確立し、JSP コンテナに対して適切な処理を指示します。
- xmlns 属性の設定を使用して、JSP XML コア構文とカスタム・タグ・ライブラリに対して、必要な XML 名前空間を示します。
- JSP のバージョン番号 (必須) を指定します。

常に 1 つの xmlns 属性があり、JSP XML コア構文の名前空間を識別します。

```
xmlns:jsp="http://java.sun.com/JSP/Page"
```

この xmlns:jsp 設定によって、JSP 仕様に定義されている標準的な要素が使用できます。

また、使用する各カスタム・タグ・ライブラリ用の xmlns 属性を含めて、タグ・ライブラリの接頭辞と名前領域を指定する必要があります。つまり、タグ使用の妥当性チェックに使用するための対応する TLD をポイントする必要があります。これらの xmlns 設定は、従来の JSP ページの taglib ディレクティブと等価です。

TLD をポイントするには、URN または URI のいずれかを使用できます。Sun 社の JSP 仕様 1.2 には、タグ・ライブラリ接頭辞 (eg と temp) について、次の例が記載されています。

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
          xmlns:eg="http://java.apache.org/tomcat/examples-taglib"
          xmlns:temp="urn:jsptld:/WEB-INF/tlds/my.tld"
          version="1.2"
>
```

```
...body of document...
```

```
</jsp:root>
```

URN は、アプリケーション相対パスを示し、`urn:jsptld:path` の形式であることが必要です。このパスは `taglib` ディレクティブの `uri` 属性と同じ方法で指定されます。8-13 ページの「[概要: taglib ディレクティブによるタグ・ライブラリの指定](#)」を参照。

URI は完全な URL の場合があります。そうでない場合は、`web.xml` ファイルの `<taglib>` 要素、または TLD の `<uri>` 要素のマッピングに従います。8-18 ページの「[web.xml のタグ・ライブラリへの使用](#)」および 8-16 ページの「[単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス](#)」を参照。

例の中の `version` 属性にも注意してください。これは必須属性で、ページが使用する JSP のバージョンを指定します (1.2 以上)。

## JSP XML ディレクティブ要素

`page` ディレクティブと `include` ディレクティブに対応する JSP XML 要素があります (`taglib` ディレクティブは、前述の「[JSP XML ルート要素と JSP XML 名前空間](#)」で説明されているように、`<jsp:root>` 要素の `xmlns` 設定で置換されます)。

`page` ディレクティブまたは `include` ディレクティブを、対応する JSP XML 要素に変換するのは簡単な作業です。次に例を示します。

**例: page ディレクティブ** 次に `page` ディレクティブの例を示します。

```
<%@ page language="sqlj"
    import="sqlj.runtime.ref.DefaultContext,oracle.sqlj.runtime.Oracle" %>
```

これは、次の JSP XML 要素に対応します。

```
<jsp:directive.page language="sqlj"
    import="sqlj.runtime.ref.DefaultContext,oracle.sqlj.runtime.Oracle" />
```

**例: include ディレクティブ** 次に `include` ディレクティブの例を示します。

```
<%@ include file="/jsp/userinfo.jsp" %>
```

これは、次の JSP XML 要素に対応します。

```
<jsp:directive.include file="/jsp/userinfo.jsp" />
```

---

---

**注意:** ページの XML ビューには、`include` 要素がありません。これは、静的にインクルードされたセグメントはビューに直接コピーされるためです。

---

---

## JSP XML の宣言要素、式要素およびスクリプトレット要素

JSP の宣言、式およびスクリプトレットに対応する JSP XML 要素があります。

これらの構成に対応する JSP XML 要素に変換するのは簡単な作業です。次に例を示します。

**例：JSP の宣言** 次に JSP の宣言例を示します。

```
<%! public String func(int myint) { if (myint<10) return("..."); } %>
```

これは、次の JSP XML 要素に対応します。

```
<jsp:declaration>
  <![CDATA[ public String func(int myint) { if (myint<10) return("..."); } ]]>
</jsp:declaration>
```

XML の CDATA (文字データ) の指示が使用されているのは、宣言に「<」文字が含まれているためです。この文字は XML パーサーに対して特別な意味を持ちます (JSP XML ページの作成に XML エディタを使用すると、この文字は自動的に処理されます)。「<」の代わりに「&lt;」エスケープ文字を使用して、次のように記述することもできます。

```
<jsp:declaration>
  public String func(int myint) { if (myint &lt; 10) return("..."); }
</jsp:declaration>
```

**例：JSP の式** 次に JSP の式の例を示します。

```
<%= (user==null) ? "" : user %>
```

これは、次の JSP XML 要素に対応します。

```
<jsp:expression> (user==null) ? "" : user </jsp:expression>
```

**例：JSP のスクリプトレット** 次に JSP のスクリプトレットの例を示します。

```
<% if (pageBean.getNewName().equals("")) { %>
  ...
```

これは、次の JSP XML 要素に対応します。

```
<jsp:scriptlet> if (pageBean.getNewName().equals("")) { </jsp:scriptlet>
  ...
```

## JSP XML の標準アクションとカスタム・アクションの要素

JSP の標準アクション (`jsp:include`、`jsp:forward` および `jsp:useBean` など) とカスタム・アクションに対する従来の構文は、すでに XML 互換です。ただし、JSP XML 構文で標準アクションやカスタム・アクションを実行する場合は、次の点に注意してください。

- リクエスト時の式の値を受け入れる属性を持つ標準アクションやカスタム・アクションの要素は、次の構文を使用してその値を取得できます。

```
"%=expression%"
```

この構文の前後には、「<」および「>」の山カッコがなく、`expression` の前後の空白は不要であることに注意してください。XML 文書の場合と同様に、規定の引用符の後にある `expression` の値は、JSP のリクエスト時の式の場合と同じです。

- 引用符は、XML 仕様に従う必要があります。
- テンプレート・データは、`<jsp:text>` 要素または選択した XML 要素（標準でもカスタムでもない要素）を使用して導入できます。詳細は、次項の「[JSP XML のテキスト要素とその他の要素](#)」を参照してください。

## JSP XML のテキスト要素とその他の要素

`<jsp:text>` 要素は、JSP XML 文書のテンプレート・データを示します。

```
<jsp:text>
  ...template data...
</jsp:text>
```

`<jsp:text>` 要素を検出した JSP コンテナは、その内容を現行の JSP の `out` オブジェクトに渡します (XSLT の `<xsl:text>` 要素の処理とほぼ同じです)。

JSP 仕様では、`<jsp:text>` 要素が使用されている場所で、テンプレート・データ用に任意の要素 (標準アクションの要素やカスタム・アクションの要素以外) を使用できます。これら任意の要素は、現行の JSP の `out` オブジェクトに送信されるコンテンツを使用し、`<jsp:text>` 要素と同じ方法で処理されます。

次に、Sun 社の JSP 仕様 1.2 での例を示します。

**例：その他の JSP XML 要素** 次のような JSP XML 文書のソース・テキストについて考えてみます。

```
<hello><jsp:scriptlet>int i=3;</jsp:scriptlet>
<hi>
<jsp:text> hi you all
</jsp:text><jsp:expression>i</jsp:expression>
</hi>
</hello>
```



このソース・テキストによって、次の内容が JSP コンテナから出力されます。

```
<hello> <hi> hi you all  
3 </hi></hello>
```

空白の扱いに注意してください。

## 比較サンプル: 従来の JSP ページと JSP XML 文書との比較

ここでは、2つのバージョン（従来の構文と XML 構文）の JSP ページを示します。

このサンプルのデプロイと実行の詳細は、次の Web サイトを参照してください。

<http://otn.oracle.co.jp/products/oc4j/index.html>

（この Web サイトを参照するには、OTN-J に登録する必要があります。無償で登録できません。）

**従来の JSP ページのサンプル** 従来の構文によるサンプル・ページを示します。

```
<%@ page session = "false" %>  
  
<jsp:useBean id = "picker" class = "oracle.jsp.sample.lottery.LottoPicker"  
            scope = "page" />  
  
<% picker.setIdentity(request.getRemoteAddr() ); %>  
  
<HTML>  
<HEAD>  
  <TITLE>Lotto Number Generator</TITLE>  
</HEAD>  
  
<BODY BACKGROUND="images/cream.jpg" BGCOLOR="#FFFFFF">  
  
<H1 ALIGN="CENTER"></H1>  
  
<BR>  
  
<H1 ALIGN="CENTER">Your Specially Picked</H1>  
<P ALIGN="CENTER"><IMG SRC="images/winningnumbers.gif" WIDTH="450" HEIGHT="69"  
ALIGN="BOTTOM"  
BORDER="0"></P>  
<P>  
  
<P ALIGN="CENTER">  
<TABLE ALIGN="CENTER" BORDER="0" CELLPADDING="0" CELLSPACING="0">  
<TR>  
<%
```

```
        int [] picks = picker.getPicks();
        for (int i = 0; i < picks.length; i++) {
%>
            <TD>
                <IMG SRC="images/ball<%= picks[i] %>.gif" WIDTH="68" HEIGHT="76"
                    ALIGN="BOTTOM" BORDER="0">
            </TD>

        <%
        }
%>
</TR>
</TABLE>

</P>

<P ALIGN="CENTER"><BR>
<BR>
<IMG SRC="images/playrespon.gif" WIDTH="120" HEIGHT="73" ALIGN="BOTTOM" BORDER="0">

</BODY>
</HTML>
```

**JSP XML 文書のサンプル** 同じページをXML構文で示します。

```
<jsp:root
    xmlns:jsp="http://java.sun.com/JSP/Page"
    version="1.2">

    <jsp:directive.page session = "false" contentType="text/html"/>

    <jsp:useBean id = "picker" class = "oracle.jsp.sample.lottery.LottoPicker"
        scope = "page" />
    <jsp:scriptlet>picker.setIdentity(request.getRemoteAddr() ); </jsp:scriptlet>
    <jsp:text><![CDATA[<HTML>

    <HEAD>
        <TITLE>Lotto Number Generator</TITLE>
    </HEAD>

    <BODY BACKGROUND='../basic/lottery/images/cream.jpg' BGCOLOR='#FFFFFF'>

    <H1 ALIGN='CENTER' ></H1>

    <BR>

    <H1 ALIGN='CENTER' >Your Specially Picked</H1>
```

```
<P ALIGN='CENTER'><IMG SRC='../basic/lottery/images/winningnumbers.gif'
    WIDTH='450' HEIGHT='69' ALIGN='BOTTOM' BORDER='0'></P>

<P ALIGN='CENTER'>
<TABLE ALIGN='CENTER' BORDER='0' CELLPADDING='0' CELLSPACING='0'>
<TR>]]</jsp:text>
<jsp:scriptlet>
    int [] picks = picker.getPicks();
    for (int i = 0; i &lt; picks.length; i++)
    {
</jsp:scriptlet>
<jsp:text><![CDATA[<TD>
    <IMG SRC='../basic/lottery/images/ball]]>
</jsp:text>
<jsp:expression>picks[i]</jsp:expression>
<jsp:text>
    <![CDATA[.gif' WIDTH='68' HEIGHT='76' ALIGN='BOTTOM' BORDER='0'>
</TD>]]></jsp:text>
<jsp:scriptlet>
    }
</jsp:scriptlet>
<jsp:text><![CDATA[</TR>
</TABLE>
</P>
<P ALIGN='CENTER'><BR>
<BR>
<IMG SRC='../basic/lottery/images/playrespon.gif' WIDTH='120' HEIGHT='73'
ALIGN='BOTTOM' BORDER='0'>
</BODY>
</HTML>]]></jsp:text>
</jsp:root>
```

## JSP XML ビューの詳細

JSP 1.2 を使用してコンパイルを行うコンテナで JSP ページを変換すると、XML ビューと呼ばれる、解析結果の XML バージョンが作成されます。JSP 仕様では、XML ビューを、JSP ページ（従来のページまたは JSP XML 文書のいずれか）から、そのページについての記述がある XML 文書へのマッピングとして定義しています。XML ビューは、タグ・ライブラリ・バリデータ・クラスによるページの妥当性チェックに使用できます（8-46 ページの「[妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス](#)」を参照）。ページの XML ビューは、JSP XML 構文を使用してユーザー自身が記述した場合のページに類似していますが、主な相違点がいくつかあります。これについて簡単に説明します。

この項には、次の項目が含まれます。

- [JSP ページから XML ビューへの変換](#)
- [妥当性チェックにおけるエラー・レポートの jsp:id 属性](#)
- [例: 従来の JSP ページから XML ビューへの変換](#)

詳細は、Sun 社の JSP 仕様を参照してください。

## JSP ページから XML ビューへの変換

JSP ページを変換する場合、JSP コンテナは、XML ビューを作成する際に従来の JSP ページと JSP XML 文書の両方に対して、次の変換を実行します。

- コンテナは、XML ビューを拡張して、include ディレクティブを使用して移入したファイルをインクルードします。
- オプションの jsp:id 属性（改善されたエラー・レポート用）をサポートしている JSP コンテナは、その属性をページ内の各 XML 要素に挿入します。5-15 ページの「[妥当性チェックにおけるエラー・レポートの jsp:id 属性](#)」を参照。

JSP XML 文書の場合は、これらの点が、XML ビューと元のページとの主な差異の原因となります。

JSP コンテナは、従来の JSP ページに対して、次の追加変換を実行します。

- JSP XML 構文の場合は標準の xmlns 属性設定で、JSP バージョンの場合は version 属性で、<jsp:root> 要素を追加します。5-7 ページの「[JSP XML ルート要素と JSP XML 名前空間](#)」を参照。
- 各 taglib ディレクティブを <jsp:root> 要素の追加の xmlns 属性に変換します。5-7 ページの「[JSP XML ルート要素と JSP XML 名前空間](#)」を参照。
- 各 page ディレクティブを JSP XML 構文の対応する要素に変換します。5-8 ページの「[JSP XML ディレクティブ要素](#)」を参照。
- 宣言、式、スクリプトレットを、それぞれ JSP XML 構文の対応する要素に変換します。5-9 ページの「[JSP XML の宣言要素、式要素およびスクリプトレット要素](#)」を参照。

- リクエスト時の式を XML 構文に変換します。5-10 ページの「[JSP XML の標準アクションとカスタム・アクションの要素](#)」を参照。
- テンプレート・データの `<jsp:text>` 要素を作成します。5-10 ページの「[JSP XML のテキスト要素とその他の要素](#)」を参照。
- JSP の引用符を XML の引用符に変換します。
- JSP のコメント (`<%-- comment --%>`) を無視します。これらのコメントは XML ビューに表示されません。

---

---

**注意：**

- XML ビューに DOCTYPE 文はありません。
  - 5-10 ページの「[JSP XML のテキスト要素とその他の要素](#)」に説明されているように、その他の XML 要素は XML ビューに表示されません。テンプレート・データで使用されるのは、`<jsp:text>` 要素のみです。
- 
- 

## 妥当性チェックにおけるエラー・レポートの `jsp:id` 属性

JSP 仕様には、JSP コンテナが XML ビューの各 XML 要素に追加できるオプションの `jsp:id` 属性についての説明があります。コンテナは、この機能をサポートして JSP 1.2 に準拠する必要はありませんが、OC4J JSP コンテナはこの機能をサポートしています。

`jsp:id` 属性がある場合は、ページの妥当性チェック時にタグ・ライブラリ・バリデータ・クラスによって使用されます。この属性の目的は、改善されたエラー・レポートを提供することです。これは、(JSP コンテナによる `jsp:id` サポートの実装方法に従って) エラーの発生場所を開発者に正確に示すのに役立ちます。

`jsp:id` 属性の値は、それぞれの値または ID が、XML ビューのすべての要素間で一意となるように、コンテナによって生成される必要があります。

タグ・ライブラリ・バリデータ・オブジェクトは、戻された `ValidationMessage` オブジェクトで、これらの ID を使用できます (TLV クラスに関するバックグラウンド情報は、8-46 ページの「[妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス](#)」を参照してください)。

OC4J の JSP 実装では、ID を持つ `ValidationMessage` オブジェクトが戻されると、各 ID は、一致している要素のタグ名とソース位置を反映するために変換されます。

## 例 : 従来の JSP ページから XML ビューへの変換

この例では、従来のページ・ソースを示し、次に、OC4J の JSP トランスレータが生成するページの XML ビューを示します。このコードでは、Oracle JSP のバージョン番号と構成パラメータの値が表示されています。

**従来の JSP ページ** 従来の JSP ページを示します。

```
<HTML>
  <HEAD>
    <TITLE>JSP Information </TITLE>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
    JSP Version:<BR>
      <%= application.getAttribute("oracle.jsp.versionNumber") %>
    <BR>
    JSP Init Parameters:<BR>
    <%
      for (Enumeration paraNames = config.getInitParameterNames();
        paraNames.hasMoreElements() ;) {
        String paraName = (String)paraNames.nextElement();
      %>
    <%=paraName%> = <%=config.getInitParameter(paraName)%>
    <BR>
    <%
      }
    %>
  </BODY>
</HTML>
```

**JSP ページの XML ビュー** 対応する XML ビューを示します。

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" jsp:id="0" version="1.2">
  <jsp:text jsp:id="1"><![CDATA[ <HTML>
    <HEAD>
      <TITLE>JSP Information </TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
      JSP Version:<BR>]]></jsp:text>
  <jsp:expression jsp:id="2">
    <![CDATA[ application.getAttribute("oracle.jsp.versionNumber") ]]>
  </jsp:expression>
  <jsp:text jsp:id="3"><![CDATA[
    <BR>
      JSP Init Parameters:<BR>
    ]]>
  </jsp:text>
  <jsp:scriptlet jsp:id="4"><![CDATA[
```

```
        for (Enumeration paraNames = config.getInitParameterNames();
            paraNames.hasMoreElements() ;) {
            String paraName = (String)paraNames.nextElement();
        ]]></jsp:scriptlet>
<jsp:text jsp:id="5"><![CDATA[
    ]]></jsp:text>
<jsp:expression jsp:id="6"><![CDATA[paraName]]></jsp:expression>
<jsp:text jsp:id="7"><![CDATA[ = ]]></jsp:text>
<jsp:expression jsp:id="8">
    <![CDATA[config.getInitParameter(paraName)]]>
</jsp:expression>
<jsp:text jsp:id="9"><![CDATA[
    <BR>
    ]]></jsp:text>
<jsp:scriptlet jsp:id="10"><![CDATA[
    }
    ]]></jsp:scriptlet>
<jsp:text jsp:id="11"><![CDATA[
    </BODY>
</HTML>

]]></jsp:text>
</jsp:root>
```





# 6

---

## プログラミングに関するその他の考慮事項

この章では、JSP アプリケーションの開発に使用する、様々なプログラミングの方針とヒントについて説明します。この章には、次の項目が含まれます。

- 一般的な JSP プログラミングの方針
- JSP プログラミングのその他のヒント

## 一般的な JSP プログラミングの方針

この項では、特定のターゲット環境に関係なく、JSP ページのプログラミング時に考慮する必要がある事項について説明します。この項には、次の項目が含まれます。

- [JavaBeans とスクリプトレットの比較](#)
- [静的なインクルードと動的なインクルードの比較](#)
- [JSP タグ・ライブラリの作成と使用を考慮する時期](#)

---

---

**注意：** この項で説明する内容以外に、JSP 変換およびデプロイに関連する事項と動作についても理解が必要です。第 7 章「[JSP の変換とデプロイ](#)」を参照。

---

---

## JavaBeans とスクリプトレットの比較

1-5 ページの「[ビジネス・ロジックとページ・プレゼンテーションの分離 : JavaBeans のコール](#)」では、JavaServer Pages テクノロジーの主なメリットについて説明しています。つまり、ビジネス・ロジックを含み、動的なコンテンツを決定する Java コードは、リクエスト処理、プレゼンテーション・ロジックおよび静的なコンテンツを含む HTML コードから分離できます。この分離によって、HTML のエキスパートはプレゼンテーションに集中でき、Java のエキスパートは、JSP ページからコールされる JavaBeans のビジネス・ロジックに集中できます。

標準の JSP ページに含まれるのは、通常、リクエスト処理やプレゼンテーション用の Java 機能に関する簡単な Java コードのみです。4-7 ページの「[JDBC を使用した JSP データ・アクセスのサンプル](#)」に示したサンプル・ページは説明に役立つ例ですが、理想的な設計ではありません。たとえば、このサンプルの `runQuery()` メソッドでのデータ・アクセスは、JavaBean で実行する方が適切です。ただし、出力をフォーマットする `formatResult()` メソッドは、JSP ページの方が適切です。

## 静的なインクルードと動的なインクルードの比較

`include` ディレクティブ (1-7 ページの「[ディレクティブ](#)」を参照) は、変換時にインクルード・ページのコピーを作成し、それを JSP ページ (インクルード先ページ) にコピーします。この機能は、静的なインクルード (または変換時インクルード) と呼ばれ、次の構文を使用します。

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

`jsp:include` タグ (1-15 ページの「標準アクション:JSP タグ」を参照) は、実行時に、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します。この機能は、動的なインクルード (または実行時インクルード) と呼ばれ、次の構文を使用します。

```
<jsp:include page="/jsp/userinfopage.jsp" flush="true" />
```

C 構文の知識がある方にとって、静的なインクルードは `#include` 文に相当します。動的なインクルードは、ファンクション・コールと同じです。いずれのインクルードも便利ですが、異なる目的で使用されます。

---

---

**注意：** 静的なインクルードと動的なインクルードは、同じサブレット・コンテキスト内のページ間でのみ使用できます。

---

---

### 静的なインクルードのロジック手法

静的なインクルードの場合は、インクルード先の JSP ページで生成されるコードのサイズが大きくなります。これは、変換時に `include` ディレクティブの時点で、インクルード・ページのテキストが、インクルード先ページに物理的にコピーされるためです。1 つのページが複数回、インクルード先ページにインクルードされると、複数のコピーが作成されます。

静的にインクルードされた JSP ページは、独立した変換可能なエンティティである必要はありません。このページは、インクルード先ページにコピーされるテキストのみで構成されます。インクルード・テキストがコピーされたインクルード先ページは、変換可能であることが必要です。インクルード先ページは、インクルード・ページがコピーされる前に変換可能である必要はありません。静的にインクルードされた一連のページは、それ自体では独立できないフラグメントとなる可能性があります。

### 動的なインクルードのロジック手法

動的なインクルードでは、リクエスト・ディスパッチャなどへのメソッド・コールが増加しますが、インクルード先ページで生成されたコードのサイズが大幅に増加することはありません。動的なインクルードによって、実行時の処理は、インクルード先ページからインクルード・ページに切り替えられます。これは、インクルード・ページのテキストをインクルード先ページに物理的にコピーする処理とは逆になります。

動的なインクルードの場合は、リクエスト・ディスパッチャへのコールを追加するため、処理のオーバーヘッドが増加します。

動的にインクルードされたページは、それ自体で変換および実行できる独立したエンティティであることが必要です。インクルード先ページも同様に、動的なインクルードなしに変換および実行できる独立したエンティティであることが必要です。

## 動的なインクルードと静的なインクルードのメリット、デメリットおよび代表的な使用例

静的なインクルードはページのサイズに影響を与え、動的なインクルードは処理のオーバーヘッドに影響を与えます。静的なインクルードの場合は、動的なインクルードに必要なリクエスト・ディスパッチャのオーバーヘッドを回避できますが、大規模なファイルの場合に問題が生じる可能性があります。（生成されたページ実装クラスのサービス・メソッドのサイズは、64KB に制限されています。詳細は、6-9 ページの「[大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処](#)」を参照してください。）

また、静的なインクルードの過度の使用は、JSP ページのデバッグを困難にする可能性があるため、プログラムの実行をトレースすることがさらに困難になります。静的にインクルードしたページ間でのわかりにくい相互依存は避けてください。

静的なインクルードは通常、そのコンテンツが複数の JSP ページで繰り返し使用される、小規模なファイルをインクルードするために使用します。例：

- ログまたは著作権メッセージをアプリケーションの各ページの最上部や最下部に静的にインクルードする場合。
- 複数のページに必要な宣言やディレクティブ（Java クラスのインポートなど）を含むページを静的にインクルードする場合。
- アプリケーションの各ページから、集中ステータス・チェッカ・ページを静的にインクルードする場合（6-7 ページの「[集中チェッカ・ページの使用](#)」を参照）。

動的なインクルードは、モジュール方式のプログラミングに役立ちます。1つのページを、独立して実行したり、別のページの出力の一部を生成するために使用できます。動的にインクルードされたページは、インクルード先ページのサイズを増やさずに、複数のインクルード先ページで再利用できます。

---

---

**注意：** OC4J には、1つのファイルを複数のページに静的にインクルードする便利な方法として、グローバル・インクルードが用意されています。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。

---

---

## JSP タグ・ライブラリの作成と使用を考慮する時期

開発チームがカスタム・タグの作成と使用を検討することが必要な場合があります。特に、次の場合には必要です。

- カスタム・タグを作成しないと、プレゼンテーションと出力フォーマットに関する大量の Java ロジックをインクルードする必要が生じる場合。
- 便利な JSP プログラミング・アクセスを機能に提供するには、カスタム・タグを作成しないと Java API の使用が必要となる場合。
- JSP 出力の特殊な操作またはリダイレクションが必要な場合。

### Java 構文の置換

JSP 開発者に Java プログラミングの経験がない場合、ページで Java ロジック（たとえば、JSP 出力のプレゼンテーションとフォーマットを指示するロジック）をコーディングするのは困難な場合があります。

このような場合に、JSP タグ・ライブラリが役立ちます。出力を生成するこのようなロジックが多数の JSP ページで必要な場合、Java ロジックを置換するタグ・ライブラリは、JSP 開発者にとって非常に便利です。

このようなタグ・ライブラリの例として、OC4J でサポートされている JavaServer Pages 標準タグ・ライブラリ (JSTL) と、OC4J が提供する JSP Markup Language (JML) タグ・ライブラリの 2 つがあります。これらのライブラリの詳細は、『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』を参照してください。

### API 機能に対する便利な JSP プログラミング・アクセスの提供

タグ・ライブラリの提供によって、Web アプリケーションのプログラマは、サーブレットまたは JSP スクリプトレットから製品の機能や拡張機能を使用するために、Java API に依存する必要がなくなります。タグ・ライブラリを使用すると、プログラマのタスクが大幅に軽減され、タグ・ハンドラによって適切な API コールが自動的に処理されます。

たとえば、OC4J では、電子メールとファイル・アクセス機能用に、タグと JavaBeans が提供されています。また、OC4J の Web Object Cache には、タグ・ライブラリと Java API が提供されています。同様に、Oracle Application Server Personalization は Java API を提供していますが、OC4J でも、Personalization のアプリケーションをプログラミングする場合に、かわりに使用できるタグ・ライブラリを提供しています。

### JSP 出力の操作またはリダイレクト

カスタム・タグを使用するもう 1 つの状況は、レスポンス出力に関する特殊な実行時処理が必要な場合です。必要な機能を使用するには、処理手順を追加したり、出力をブラウザ以外の場所にリダイレクトする必要がある場合があります。

たとえば、ブラウザではなくログ・ファイルにリダイレクトする出力テキストの周囲に配置するカスタム・タグを作成する場合、コードは次のようになります。この例の `cust` はタグ・ライブラリの接頭辞で、`log` はライブラリのタグの 1 つです。

```
<cust:log>
  Today is <%= new java.util.Date() %>
  Text to log.
  More text to log.
  Still more text to log.
</cust:log>
```

## JSP プログラミングのその他のヒント

前述した一般的なプログラミングの方針以外に、次の項で説明するように、プログラミングに関して考慮する必要がある様々なヒントがあります。

- [直接起動から JSP ページを除外する方法](#)
- [集中チェック・ページの使用](#)
- [大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処](#)
- [メソッド変数宣言とメンバー変数宣言の比較](#)
- [page ディレクティブの特性](#)
- [JSP での空白の保持とバイナリ・データの使用](#)

### 直接起動から JSP ページを除外する方法

一部の JSP ページをアプリケーションからのみアクセス可能にし、ユーザーが直接起動できないようにする場合があります。特に、Model-View-Controller (MVC) などのアーキテクチャで必要になる場合があります。

たとえば、フロントエンドまたは表示ページが `index.jsp` であるとします。ユーザーは、このページに直接アクセスする URL リクエストを介してアプリケーションを起動します。ここで、`index.jsp` には、2 番目のページの `included.jsp` が含まれ、3 番目のページの `forwarded.jsp` に転送されるとします。さらに、ユーザーが URL リクエストを介してこれらのページを直接起動できないようにする必要があります。

これを行うには、`included.jsp` と `forwarded.jsp` をアプリケーションの `/WEB-INF` ディレクトリに格納します。これらのページをこのディレクトリに格納すると、URL リクエストを介して直接起動できなくなります。直接起動しようとすると、ブラウザでエラー・レポートが生成されます。

`index.jsp` ページの文は次のようになります。

```
<jsp:include page="WEB-INF/included.jsp"/>
...
<jsp:forward page="WEB-INF/forwarded.jsp"/>
```

アプリケーション構造は次のようになります。構造には、サーブレット、JavaBeans または他のクラス用の標準の `classes` ディレクトリ、および JAR ファイル用の標準の `lib` ディレクトリが含まれます。

```
index.jsp
WEB-INF/
  web.xml
  included.jsp
  forwarded.jsp
classes/
lib/
```

## 集中チェッカ・ページの使用

JSP アプリケーションの全般的な管理や監視に、アプリケーションの各ページからインクルードする集中チェッカ・ページを使用すると便利です。集中チェッカ・ページは、各ページの実行中に次のタスクを実行できます。

- セッション・ステータスのチェック
- ログイン・ステータスのチェック (Cookie をチェックして有効なログインが行われたかどうかを確認するなど)
- 使用状況プロファイルのチェック (マウス・クリックやページへの接続など、対象イベントを記録するためのロギング機能を実装している場合)

この他にも多くの使用方法があります。

たとえば、`HttpSessionBindingListener` インタフェースを実装したセッション・チェッカ・クラスの `MySessionChecker` があります (4-18 ページの「[標準セッションのリソース管理 : HttpSessionBindingListener](#)」を参照)。

```
public class MySessionChecker implements HttpSessionBindingListener
{
    ...

    valueBound(HttpSessionBindingEvent event)
    {...}

    valueUnbound(HttpSessionBindingEvent event)
    {...}

    ...
}
```

たとえば、次のような内容を含むチェッカ・ページ、`centralcheck.jsp` を作成できます。

```
<jsp:useBean id="sessioncheck" class="MySessionChecker" scope="session" />
```

`centralcheck.jsp` が含まれるページでは、セッション終了時に `sessioncheck` がスコープ外になると同時に、サーブレット・コンテナが、`MySessionChecker` クラスに実装されている `valueUnbound()` メソッドをコールします。これは、セッション・リソースを管理するために実行されます。`centralcheck.jsp` は、アプリケーションの各 JSP ページの最後に含めることができます。

---

---

### 注意：

- OC4J には、1つのファイルを複数のページに静的にインクルードする便利な方法として、グローバル・インクルードが用意されています。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。
  - この種の機能について、サーブレット・フィルタの使用を検討することもできます。サーブレット・フィルタについては、『[Oracle Application Server Containers for J2EE サーブレット開発者ガイド](#)』を参照してください。
- 
-



## 大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処

JSP ページに大量の静的なコンテンツ（実行時に変更されるコンテンツのない大量の HTML コード）が含まれると、変換と実行の速度が低下する場合があります。

2つの対処方法があり、いずれの対処方法でも変換速度は向上します。

- 静的な HTML コードを別のファイルに格納し、`jsp:include` タグを使用して、実行時にその出力を JSP ページ出力にインクルードします。`jsp:include` タグの詳細は、1-15 ページの「[標準アクション: JSP タグ](#)」を参照してください。

---

**重要：** 静的な `include` ディレクティブは有効ではありません。このディレクティブを使用すると、インクルード・ファイルが変換時にインクルードされ、そのコードが実際にインクルード先ページにコピーされます。これでは、問題の解決になりません。

---

- 静的な HTML コードを Java リソース・ファイルに格納します。

`external_resource` 構成パラメータを有効にすると、JSP トランスレータがこの操作を実行します。このパラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。

事前に変換する場合は、`ojspc` ツールの `-extres` オプションが同じ機能を提供します。

---

**注意：** 静的な HTML コードをリソース・ファイルに格納すると、前述の `jsp:include` による対処よりもメモリーのフットプリントが大きくなる場合があります。これは、クラスがロードされるたびに、ページ実装クラスはリソース・ファイルをロードする必要があるためです。

---

大量の静的なコンテンツを含む JSP ページ、または大量のタグ・ライブラリを使用する JSP ページで起こりうるもう 1 つの問題は、ほとんど（全部ではない場合）の JVM では、単一のメソッド内のコード・サイズが 64 KB に制限されていることです。`javac` によるコンパイルは可能ですが、JVM では実行できません。実質的に JSP ページのソース・ファイル全体から生成された Java コードは、ページ実装クラスのサービス・メソッドに追加されるため、この問題は、JSP トランスレータの実装によっては、JSP ページに関する問題になる可能性があります。Java コードは静的な HTML をブラウザに出力するために生成され、スクリプトレットからの Java コードは直接コピーされます。

同様に、JSP ページの Java スクリプトレットのサイズが大きいため、サービス・メソッドでサイズ制限の問題が発生する場合があります。ページ内の Java コードが問題の原因の場合は、コードを `JavaBeans` に移動する必要があります。

大量のタグ・ライブラリの使用によって JSP ページのサイズが制限される場合、ページを複数のページに分割し、必要に応じて `jsp:include` タグを使用するという解決策が一般的です。

### メソッド変数宣言とメンバー変数宣言の比較

1-9 ページの「[スクリプト要素](#)」で、メンバー変数の宣言には JSP の `<%! ... %>` 宣言を使用し、メソッド変数は `<% ... %>` スクリプトレットで宣言する必要があることを説明しました。

変数の使用方法に応じて、適切な機能を使用して宣言するように注意してください。

- `<%! ... %>` JSP 宣言構文内で宣言される変数は、JSP トランスレータが生成したページ実装クラス内のクラス・レベルで宣言されます。この場合、オブジェクト・インスタンスを宣言すると、そのオブジェクトは複数のリクエストから同時にアクセスできません。したがって、`page` ディレクティブ内で `isThreadSafe="false"` が宣言されていないかぎり、オブジェクトはスレッド・セーフであることが必要です。
- `<% ... %>` JSP スクリプトレット構文内で宣言される変数は、ページ実装クラスのサービシ・メソッドに対してローカルです。メソッドがコールされるたびに、変数またはオブジェクトのインスタンスが個別に作成されるため、スレッド・セーフティは必要ありません。

次に `decltest.jsp` の例を示します。

```
<HTML>
<BODY>
<% double f2=0.0; %>
<%! double f1=0.0; %>
Variable declaration test.
</BODY>
</HTML>
```

この場合、ページ実装クラスのコードは次のようになります。

```
package ...;
import ...;

public class decltest extends ... {
    ...

    // ** Begin Declarations
    double f1=0.0;           // *** f1 declaration is generated here ***
    // ** End Declarations
```

```
public void _jspService
    (HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    ...

    try {
        out.println( "<HTML>");
        out.println( "<BODY>");
        double f2=0.0;    // *** f2 declaration is generated here ***
        out.println( "");
        out.println( "");
        out.println( "Variable declaration test.");
        out.println( "</BODY>");
        out.println( "</HTML>");
        out.flush();
    }
    catch( Exception e) {
        try {
            if (out != null) out.clear();
        }
        catch( Exception clearException) {
        }
    }
    finally {
        if (out != null) out.close();
    }
}
}
```

---

---

**注意：** これは、概念を説明するためのコードです。クラスの大部分は簡素化のために削除されているため、JSP トランスレータで生成されるページ実装クラスの実際のコードとは異なります。

---

---

## page ディレクティブの特性

この項では、次の page ディレクティブの特性について説明します。

- page ディレクティブは静的で、変換時に効果があります。パラメータ設定が実行時に評価されるようには指定できません。
- JSP 仕様 1.2 以上では、ディレクティブ属性の設定が重複することは禁止されています。この禁止事項は page ディレクティブに特に関係していますが、page ディレクティブの import 属性にはこの制限は当てはまりません。
- page ディレクティブの Java import 設定は、JSP ページまたは変換単位内に累積されません。

### 静的な page ディレクティブ

page ディレクティブは静的で、変換時に解析されます。動的な設定を実行時に解析するようには指定できません。次に例を示します。

**例 1** 次の page ディレクティブは有効です。

```
<%@ page contentType="text/html; charset=EUCJIS" %>
```

**例 2** 次の page ディレクティブは無効で、エラーが発生します（この例では EUCJIS がハードコードされていますが、実行時に動的に決定されるすべてのキャラクタ・セットに当てはまります）。

```
<% String s="EUCJIS"; %>  
<%@ page contentType="text/html; charset=<%=s%>" %>
```

一部の page ディレクティブ設定には、代替策があります。例 2 の場合は、コンテンツ・タイプを動的に設定できる `setContentType()` メソッドを使用できます（9-5 ページの「[コンテンツ・タイプの動的な設定](#)」を参照）。

### page ディレクティブ属性の重複設定の禁止

JSP 仕様では、複数のディレクティブ属性（page ディレクティブの import 属性を除く）が単一の JSP 変換単位（JSP ページおよび include ディレクティブを使用してインクルードされたページ）内で 2 回以上設定されていないことを、JSP コンテナが確認する必要があります。JSP 1.2 の場合、これは page ディレクティブのみに適用されますが、将来の JSP バージョンでは、他の関連ディレクティブが追加される可能性があります。

ディレクティブ属性の重複設定が許可されている JSP 1.1 標準に対する下位互換性のために、OC4J では `forgive_dup_dir_attr` 構成パラメータが提供されています。このパラメータの詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。たとえば、以前にコーディングしたページには、page ディレクティブの language 属性がすべて java に設定されているセグメントが複数含まれている可能性があります。

属性の重複設定については、次の点に注意してください。

- JSP 仕様では、異なる属性が設定されているかぎり、複数の page ディレクティブが許可されます。

次の場合は許可されます。

```
<%@ page buffer="none" %>
<%@ page session="true" %>
```

または

```
-----
<%@ page buffer="10kb" %>
<%@ include file="b.jsp" %>
-----
```

```
b.jsp
<%@ page session="false" %>
-----
```

次の場合は許可されません。

```
<%@ page buffer="none" %>
<%@ page buffer="10kb" %>
```

または

```
<%@ page buffer="none" buffer="10kb" %>
```

または

```
-----
<%@ page buffer="10kb" %>
<%@ include file="b.jsp" %>
-----
```

```
b.jsp
<%@ page buffer="3kb" %>
-----
```

- 変換単位は、JSP ページおよび include ディレクティブを使用してインクルードされたページで構成されていますが、jsp:include タグを使用してインクルードされたページは含まれていません。jsp:include タグを使用してインクルードされたページは、変換時に静的にインクルードされるのではなく、実行時に動的にインクルードされません。詳細は、6-2 ページの「静的なインクルードと動的なインクルードの比較」を参照してください。

したがって、次の場合は許可されます。

```
-----  
<%@ page buffer="10kb" %>  
<jsp:include page="b.jsp" />  
-----  
b.jsp  
<%@ page buffer="3kb" %>  
-----
```

- 前述のように、`page` ディレクティブの `import` 属性には、属性の重複設定に対する制限は当てはまりません。次項の「[累積される page ディレクティブの import 設定](#)」を参照してください。

### 累積される page ディレクティブの import 設定

`page` ディレクティブの `import` 属性には、JSP 1.2 のディレクティブ属性の重複に関する制限は当てはまりません。JSP ページまたは変換単位（JSP ページおよび `include` ディレクティブを使用してインクルードされたページ）では、`page` ディレクティブの `Java import` 設定が累積されます。

1 つの JSP ページまたは変換単位内では、次の 2 つの例は等価です。

```
<%@ page language="java" %>  
<%@ page import="sqlj.runtime.ref.DefaultContext, java.sql.*" %>
```

または

```
<%@ page language="java" %>  
<%@ page import="sqlj.runtime.ref.DefaultContext" %>  
<%@ page import="java.sql.*" %>
```

最初の `page` ディレクティブの `import` 設定の後は、2 番目の `page` ディレクティブの `import` 設定が、インポートするクラスまたはパッケージのセットに追加されます。インポートするクラスまたはパッケージは置換されません。

## JSP での空白の保持とバイナリ・データの使用

JSP コンテナは通常、ソース・コードの空白（改行を含む）をブラウザへの出力内に保持します。開発者が意図しない空白が挿入される場合があるため、一般的に、JSP テクノロジーは、バイナリ・データの生成には適していません。

### 空白の例

次の 2 つの JSP ページでは、ソース・コード内での改行の使用に応じて、異なる HTML 出力が作成されます。

#### 例 1 - 改行なし

次の JSP ページでは、Date() コールと getParameter() コールの後に改行がありません。(Date() コールで始まる 3 行目と 4 行目は、実際には 1 行のコードが折り返されています。)

nowhit.jsp:

```
<HTML>
<BODY>
<%= new java.util.Date() %> <% String user=request.getParameter("user"); %> <%=
(user==null) ? "" : user %>
<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

このコードによるブラウザへの HTML 出力は、次のとおりです。日付の後に空白行はありません。

```
<HTML>
<BODY>
Tue May 30 20:07:04 PDT 2000
<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

## 例 2 – 改行あり

次の JSP ページでは、Date () コールと getParameter () コールの後に改行があります。

whitespace.jsp:

```
<HTML>
<BODY>
<%= new java.util.Date() %>
<% String user=request.getParameter("user"); %>
<%= (user==null) ? "" : user %>
<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

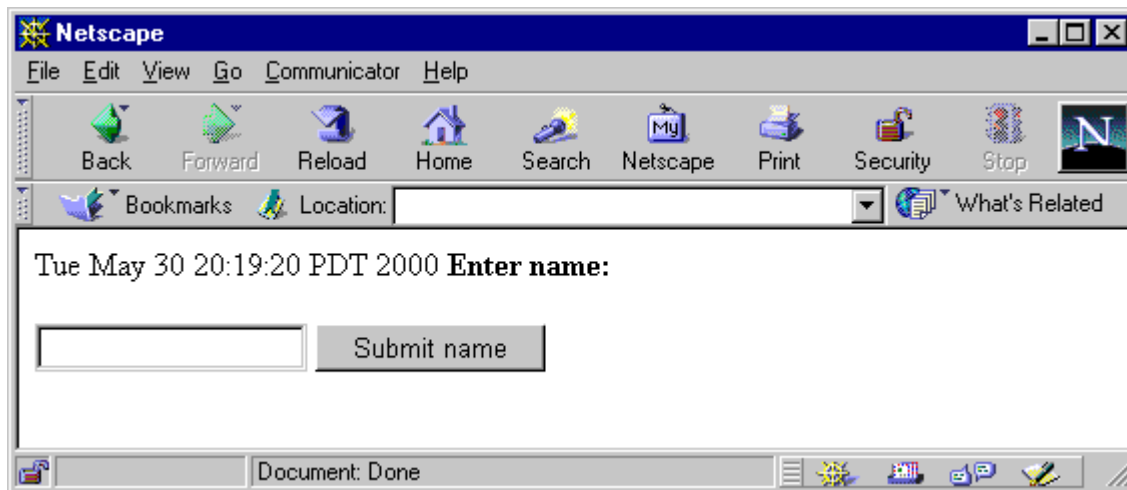
このコードによるブラウザへの HTML 出力は、次のとおりです。

```
<HTML>
<BODY>
Tue May 30 20:19:20 PDT 2000

<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

日付と「Enter name:」行との間に、2 行の空白行があることに注意してください。この 2 つの例では、次の図に示すように、ブラウザに同じ表示を作成するため、その違いは大きくありません。ただし、この例は、空白の保持に関する一般的な問題点を示しています。





## JSP ページでバイナリ・データを回避する理由

次の理由から、JSP ページはバイナリ・データの生成には適していません。通常は、かわりにサーブレットを使用します。

- JSP 実装は、バイナリ・データを処理するよう設計されていません。つまり、`JspWriter` クラスには実際のバイトを書き込むためのメソッドがありません。
- 実行中、JSP コンテナは空白を保持しています。空白は不要な場合があります。このため、JSP ページは、ブラウザへのバイナリ・データ出力（.gif ファイルなど）の生成など、空白が重要な場合には適していません。

次に一般的な例を示します。

...

```
<% response.getOutputStream().write(...binary data...) %>
<% response.getOutputStream().write(...more binary data...) %>
```

この場合、ブラウザは、出力バッファのバッファリングに応じて、バイナリ・データの途中または最後にある不要な改行文字を受け取ります。コードの行間で改行を使用しないことによってこの問題は回避できますが、この方法は望ましいプログラミング・スタイルではありません。

---

**注意：** 前述の例は説明のみを目的としているため、将来の Oracle JSP バージョンや他の JSP コンテナに移植できない場合があります。

---

JSP テクノロジは、動的なテキスト・コンテンツのプログラミングの簡素化を目的としているため、JSP ページでバイナリ・データの生成を試行すると、JSP テクノロジの特性を失うことになります。

---

## JSP の変換とデプロイ

この章では、OC4J の JSP トランスレータの操作、`ojspc` ユーティリティと事前変換が有効な状況、および JSP デプロイに関する考慮事項を説明します。

この章には、次の項目が含まれます。

- [JSP トランスレータの機能](#)
- [ojspc 事前変換ユーティリティ](#)
- [JSP デプロイに関する考慮事項](#)

## JSP トランスレータの機能

JSP トランスレータは、JSP ページ実装クラスの標準 Java コードを生成します。このクラスは、基本的に、JSP 機能が追加されたサーブレット・クラスです。

次の各項では、JSP トランスレータの全般的な機能について、Oracle Application Server の OC4J などでのオンデマンド変換の動作を中心に説明します。

- 生成されるコードの機能
- 出力名に関する一般規則
- 生成されるパッケージとクラスの名前
- 生成されるファイルとその格納場所
- Oracle JSP のグローバル・インクルード

---

---

**重要：** この項で説明するパッケージとクラスのネーミング、ファイルとディレクトリのネーミング、出力ファイルの格納場所、および生成されるコードに関する実装の詳細は、説明を目的としています。正確な詳細は、リリースごとに変更される場合があります。

---

---

## 生成されるコードの機能

この項では、JSP ソース（通常は .jsp ファイルと .sqljsp ファイル）の変換時に、JSP トランスレータによって生成されるページ実装クラス・コードの一般的な機能について説明します。

### ページ実装クラス・コードの機能

JSP トランスレータは、ページ実装クラスのサーブレット・コードを生成するとき、標準のプログラミング・オーバーヘッドの一部を自動的に処理します。オンデマンド変換モデルと事前変換モデルの両方について、生成されたコードには、自動的に次の機能が含まれます。

- `javax.servlet.jsp.HttpJspPage` インタフェースを実装した JSP トランスレータから提供されるラッパー・クラスを拡張します。これによって、より汎用的な `javax.servlet.jsp.JspPage` インタフェースが拡張され、さらに `javax.servlet.Servlet` インタフェースが拡張されます。
- `HttpJspPage` インタフェースで指定される `_jspService()` メソッドを実装します。このメソッドは、サービス・メソッドと呼ばれ、ページ実装クラスの中心的なメソッドです。Java スクリプトレットからのコード、式、および JSP ページ内の JSP タグは、このメソッド実装に組み込まれます。
- JSP ソース・コードの `page` ディレクティブで `session="false"` を特に設定しないかぎり、HTTP セッションをリクエストするコードが含まれます。

JSP とサーブレットの主なクラスとインタフェースの概要は、付録 A 「サーブレットと JSP の技術的なバックグラウンド」を参照してください。

## 静的なテキスト用のメンバー変数

ページ実装クラスのサービス・メソッドの `_jspService()` には、JSP ページの静的なテキストを出力するための出力文 (`out.print()`) または暗黙的な `out` オブジェクトにある同等のコール) が含まれます。JSP トランスレータは、静的なテキスト自体をページ実装クラス内の一連のメンバー変数に格納します。サービス・メソッドの `out.print()` 文は、テキストを出力するために、これらのメンバー変数の属性を参照します。

---

---

### 注意：

- OC4J の JSP トランスレータは、静的なテキストを Java リソース・ファイルに必要なに応じて格納できます。これは、大量の静的なテキストを含むページの場合にメリットがあります。6-9 ページの「[大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処](#)」を参照。この機能は、オンデマンド変換の場合は JSP `external_resource` 構成パラメータ、事前変換の場合は `ojspc-extres` フラグを介してリクエストできます。
  - OC4J 9.0.4 の実装以前は、静的なテキストはページ実装クラスのメンバー変数に格納されるのではなく、ページ実装クラス内のインナー・クラスに格納されていました（そのため、余分な `.class` ファイルがありました）。
- 
- 

## 出力名に関する一般規則

JSP トランスレータは、一貫した一連の規則に従って、出力クラス、パッケージ、ファイルおよびディレクトリのネーミングを行います。ただし、この一連の規則と他の実装の詳細は、リリースによって変更される場合があります。

変更されない規則の 1 つは、JSP ページのベース名に特殊文字が含まれていないかぎり、ベース名がそのまま出力クラス名とファイル名に含まれることです。たとえば、`MyPage123.jsp` を変換すると、文字列「`MyPage123`」は常に、ページ実装クラス名、Java ソース・ファイル名およびクラス・ファイル名の一部になります。

Oracle Application Server 10g (9.0.4) では、ベース名の前にアンダースコア (`_`) が付きます。`MyPage123.jsp` を変換すると、ページ実装クラスの `_MyPage123` がソース・ファイルの `_MyPage123.java` に作成され、`_MyPage123.class` にコンパイルされます。

同様に、Java パッケージ名の作成でパス名が使用されている場合は、そのパスの各コンポーネントの前にアンダースコアが付きます。たとえば、`/jspdir/myapp/MyPage123.jsp` を変換すると、`_MyPage123` クラスは次のパッケージ内に含まれます。

```
_jspdir._myapp
```

パッケージ名は、出力の .java ファイルと .class ファイルのディレクトリの作成で使用されるため、出力ディレクトリ名にもアンダースコアが付きます。たとえば、webapp/test というディレクトリにある JSP ページを変換すると、JSP トランスレータは、デフォルトで、ページ実装クラス・ソースに対して webappdeployment/\_pages/\_test というディレクトリを作成します。7-6 ページの「[生成されるファイルとその格納場所](#)」で説明するように、すべての出力ディレクトリは、標準の \_pages ディレクトリの下に作成されます。

JSP ページ名またはパス名に特殊文字を使用している場合、JSP トランスレータは、出力クラス名、パッケージ名およびファイル名に不正な Java 特殊文字が含まれていないことを確認します。たとえば、My-name\_foo12.jsp を変換すると、\_My\_2d\_name\_\_foo12 という名前のクラスがソース・ファイルの \_My\_2d\_name\_\_foo12.java に作成されます。ハイフンは、英数字の文字列に変換されます（「foo12」の前には追加のアンダースコアも挿入されます）。この場合、JSP ページ名の英数字部分のみが、出力のクラス名とファイル名にそのまま含まれます。この例では、「My」、「name」または「foo12」が含まれます。

これらの規則の例は、この章で後述します。

## 生成されるパッケージとクラスの名前

JSP 仕様には、JSP テキストを解析して変換するための統一プロセスが定義されていますが、生成されるクラスの命名方法は説明されていません。クラスの命名方法は、各 JSP 実装によって異なります。

この項では、変換時にコードを生成するとき、OC4J の JSP トランスレータが行うパッケージ名とクラス名の作成方法について説明します。

---

---

**注意：** OC4J の JSP トランスレータが出力クラス、パッケージおよびファイルのネーミングで使用する一般規則については、7-3 ページの「[出力名に関する一般規則](#)」を参照してください。

---

---

## パッケージのネーミング

オンデマンド変換の場合は、ユーザーが JSP ページのリクエスト時に指定する URL パス（具体的には、ドキュメント・ルートまたはアプリケーション・ルートに対して相対的なパス）によって、生成されるページ実装クラスのパッケージ名が決定します。URL パス内の各ディレクトリは、パッケージ階層のレベルを表します。

ただし、生成されるパッケージ名は、URL 内の文字の大小に関係なく、常に小文字になります。

次の URL の例を考えてみます。

```
http://host:port/HR/expenses/login.jsp
```

この結果、現行の OC4J の JSP 実装における、生成されたコードのパッケージ仕様は次のとおりです。

```
package _hr._expenses;
```

(実装の詳細は、今後のリリースで変更される場合があります。)

JSP ページがアプリケーション・ルート・ディレクトリにある場合、パッケージ名は生成されません。この場合の URL は、次のとおりです。

```
http://host:port/login.jsp
```

## クラスのネーミング

.jsp ファイル (または .sqljsp ファイル) のベース名によって、生成されるコード内のクラス名が決定します。

次の URL の例を考えてみます。

```
http://host:port/HR/expenses/UserLogin.jsp
```

現行の OC4J の JSP 実装における、生成されたコードのクラス名は次のとおりです。

```
public class _UserLogin extends ...
```

(実装の詳細は、今後のリリースで変更される場合があります。)

ユーザーが URL に指定する文字は、実際の .jsp ファイル名または .sqljsp ファイル名と大 / 小文字が一致する必要があります。たとえば、実際のファイル名が UserLogin.jsp または userlogin.jsp である場合、エンド・ユーザーは、それぞれの名前をそのまま指定できますが、実際のファイル名が UserLogin.jsp の場合、userlogin.jsp は指定できません。

現在のトランスレータは、ファイル名の大 / 小文字に従ってクラス名の大 / 小文字を区別しています。例：

- ファイル名 Userlogin.jsp は、クラスでは \_Userlogin になります。
- ファイル名 Userlogin.jsp は、クラスでは \_Userlogin になります。
- ファイル名 userlogin.jsp は、クラスでは \_userlogin になります。

クラス名の大 / 小文字を区別する場合は、それに従って .jsp ファイル名または .sqljsp ファイル名を付ける必要があります。ただし、ページ実装クラスはエンド・ユーザーに表示されないため、通常は問題になりません。

## 生成されるファイルとその格納場所

オンデマンド変換の場合、この項では、JSP トランスレータで生成されるファイルとその格納場所について説明します。(事前変換の場合は、ojspc によるファイルの格納方法が異なり、固有の関連オプションがあります。詳細は、7-32 ページの「[ojspc の出力ファイル、場所および関連オプションのサマリー](#)」を参照してください。)

JSP 構成パラメータの詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

---

---

**注意：** 出力クラス、パッケージおよびファイルのネーミングで使用される一般規則については、7-3 ページの「[出力名に関する一般規則](#)」を参照してください。

---

---

### JSP トランスレータで生成されるファイル

この項では、通常の JSP ページ (.jsp ファイル) と SQLJ JSP ページ (.sqljsp ファイル、または page ディレクティブに language="sqlj" があるファイル) の 2 つのページについて、JSP トランスレータで生成されるファイルを列挙して説明します。ファイル名の例では、変換されるファイルの名前として、Foo.jsp または Foo.sqljsp を使用します。

#### ソース・ファイル

- ページが SQLJ JSP ページの場合、.sqlj ファイル (\_Foo.sqlj など) は、OC4J の JSP トランスレータで作成されます。
- .java ファイル (\_Foo.java など) は、ページ実装クラスに対して作成されます。このファイルは、JSP トランスレータによって .jsp ファイルから直接作成されるか、ページが SQLJ JSP ページの場合は、SQLJ トランスレータによって .sqlj ファイルから作成されます。デフォルトでは、現在インストールされている Oracle SQLJ トランスレータが使用されますが、sqljcmd JSP 構成パラメータを使用して別のトランスレータを指定できます。

#### バイナリ・ファイル

- ISO コードの生成を伴う SQLJ JSP ページの場合は、SQLJ プロファイルの SQLJ 変換時に 1 つ以上のバイナリ・ファイルが作成されます。デフォルトでは、これらのバイナリ・ファイルは、.ser Java リソース・ファイルですが、sqljcmd 構成パラメータを介して SQLJ -ser2class フラグを有効にすると、.class ファイルになります。リソース・ファイルまたは .class ファイルの名前の一部には、「Foo」が使用されます。(Oracle 固有のコードの場合、デフォルトの SQLJ コード生成は -codegen=oracle です。Oracle 固有のコードの場合、プロファイルは作成されません。ISO コードの場合は -codegen=iso を使用します。)
- .class ファイルは、Java コンパイラによって、ページ実装クラスに対して作成されます。デフォルトでの Java コンパイラは JDK javac ですが、JSP javaccmd 構成パラメータを使用して別のコンパイラを指定できます。



- `.res` Java リソース・ファイル (`_Foo.res` など) は、`external_resource` JSP 構成パラメータが有効な場合、静的なページのコンテンツに対して必要に応じて作成されず。

---

**注意：** ページ実装クラスに対して生成されるファイルの正確な名前は、今後のリリースで変更される場合がありますが、一般的なフォームは同じです。名前には、常にベース名 (この例の場合は「`Foo`」) が含まれますが、詳細は変更される場合があります。

---

## JSP トランスレータの出力ファイルの格納場所

JSP トランスレータは、生成された出力ファイルを `_pages` ディレクトリに格納します。このディレクトリは、JSP キャッシュ・ディレクトリの下に作成されます。JSP キャッシュ・ディレクトリは、`global-web-application.xml` ファイルまたはアプリケーションの `orion-web.xml` ファイルの `<orion-web-app>` 要素の `jsp-cache-directory` 属性に指定されます。`jsp-cache-directory` のデフォルトの `./persistence` 値を想定している場合は、一般的に、次の場所が基本になります。

```
ORACLE_HOME/j2ee/home/app-deployment/app-name/web-app-name/persistence/_pages/...
```

スタンドアロン OC4J では、次の場所が OC4J がインストールされた場所に対して相対的になります。

```
j2ee/home/app-deployment/app-name/web-app-name/persistence/_pages/...
```

次に、構成ファイルについて説明します。構成ファイルの関連情報は、3-31 ページの「[主な OC4J 構成ファイル](#)」を参照してください。

- `app-deployment` ディレクトリは OC4J のデプロイ・ディレクトリで、OC4J の `server.xml` ファイルに指定されています (スタンドアロン OC4J では、一般的には、`application-deployments` ディレクトリです)。
- `app-name` はアプリケーション名で、`server.xml` の `<application>` 要素に従います。
- `web-app-name` は対応する Web アプリケーション名で、OC4J Web サイトの XML ファイル (通常、Oracle Application Server では `default-web-site.xml` ファイル、スタンドアロン OC4J では `http-web-site.xml` ファイル) 内の `<web-app>` 要素のアプリケーション名にマッピングされます。

`_pages` ディレクトリ下のパスは、アプリケーション・ルート・ディレクトリ下の `.jsp` ファイルのパスによって決まります。

たとえば、スタンドアロン OC4J で、スタンドアロン OC4J のデフォルトの Web アプリケーション・ディレクトリの下に `examples/jsp` サブディレクトリにある `welcome.jsp` ページについて考えてみます。このページへのパスは次のようになり、OC4J がインストールされた場所に対して相対的です。

```
j2ee/home/default-web-app/examples/jsp/welcome.jsp
```

デフォルトのアプリケーション・デプロイ・ディレクトリの場合、JSP トランスレータは、出力ファイル (`_welcome.java` および `_welcome.class`) を次のディレクトリに格納します。

```
j2ee/home/application-deployments/default/defaultWebApp/persistence/_pages/_examples/_jsp
```

`.jsp` ソース・ファイルは、アプリケーション・ルート・ディレクトリ下の `examples/jsp` サブディレクトリに格納されているため、JSP トランスレータは、パッケージ名として `_examples._jsp` を生成し、出力ファイルを `_pages` ディレクトリ下の `_examples/_jsp` サブディレクトリに格納します。

スタンドアロン OC4J では、次の点に注意してください。

- `application-deployments` ディレクトリは、OC4J のデフォルトのデプロイ・ディレクトリです。
- `default` は OC4J のデフォルトのアプリケーション名で、`defaultWebApp` はデフォルトの Web アプリケーション名です。両方とも、`default-web-app` ディレクトリに格納された JSP ページで使用されます。

---

**重要：** 生成された出力ファイルの格納場所と出力ファイル名での「\_」の使用など、実装の詳細は、今後のリリースで変更される場合があります。

---

## Oracle JSP のグローバル・インクルード

OC4J JSP コンテナでは、グローバル・インクルードと呼ばれる機能が提供されています。この機能によって、仮想の JSP `include` ディレクティブを使用して、指定のディレクトリ内またはディレクトリ下の JSP ページに静的にインクルードする 1 つ以上のファイルを指定できます。変換時に、JSP コンテナは、インクルード・ファイルとディレクトリをページに指定する構成ファイル `/WEB-INF/ojsp-global-include.xml` を検索します。

この拡張機能は、以前の Oracle JSP リリースで `globals.jsa` または `translate_params` 機能を使用していたアプリケーションを移行する場合に、特に便利です。

グローバルにインクルードされたファイルは、次の場合などに使用できます。

- グローバルな Bean 宣言（以前は、`globals.jsa` でサポートされていました。）
- 共通のページ・ヘッダーまたはフッター
- `translate_params` と同等の機能を持つコード（一般的に、JServ 環境用）

## ojjsp-global-include.xml ファイル

ojjsp-global-include.xml ファイルは、インクルードするファイルの名前、ファイルのインクルード先 (JSP ページの最上部または最下部)、およびグローバル・インクルードを適用する JSP ページの場所を指定します。この項では、ojjsp-global-include.xml の要素について説明します。

### <ojjsp-global-include>

ojjsp-global-include.xml ファイルのルート要素です。属性はありません。

<ojjsp-global-include> のサブ要素は次のとおりです。

<include>

### <include ... >

<ojjsp-global-include> の <include> サブ要素を使用して、インクルードするファイル、およびインクルード先が JSP ページの最上部かまたは最下部かを指定します。

<include> のサブ要素は次のとおりです。

<into>

<include> の属性は次のとおりです。

- **file** – 「/header.html」や「/WEB-INF/globalbeandclarations.jsph」などのインクルードするファイルを指定します。ファイル名の設定は、必ずスラッシュ (/) で始まります。つまり、ファイル名はページ相対ではなく、アプリケーション相対である必要があります。
- **position** – ファイルのインクルード先が JSP ページの最上部かまたは最下部かを指定します。サポートされている値は、「top」(デフォルト) と「bottom」です。

### <into ... >

<include> のこのサブ要素を使用して、指定ファイルのインクルード先 JSP ページの場所 (ディレクトリ、場合によってはサブディレクトリ) を指定します。この要素にサブ要素はありません。

<into> の属性は次のとおりです。

- **directory** – ディレクトリを指定します。このディレクトリ内 (必要に応じてそのサブディレクトリ内) にある JSP ページは、<include> 要素の **file** 属性に指定されているファイルを、静的にインクルードします。directory の設定値は、「/dir1」のように、スラッシュ (/) で始まる必要があります。また、「/dir1/」のように、設定値のディレクトリ名の後にスラッシュを付けることができます。付けない場合は、変換時に内部で追加されます。

- `subdir` – `directory` のすべてのサブディレクトリ内にある JSP ページに、ファイルの静的なインクルードが必要かどうかを指定します。サポートされている値は、「true」（デフォルト）と「false」です。

## グローバル・インクルードの例

この項では、グローバル・インクルードの例を示します。

**例：ヘッダー/フッター** 次の `ojjsp-global-include.xml` ファイルの例を考えてみます。

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE ojjsp-global-include SYSTEM 'ojjsp-global-include.dtd'>

<ojjsp-global-include>
  <include file="/header.html">
    <into directory="/dir1" />
  </include>
  <include file="/footer1.html" position="bottom">
    <into directory="/dir1" subdir="false" />
    <into directory="/dir1/part1/" subdir="false" />
  </include>
  <include file="/footer2.html" position="bottom">
    <into directory="/dir1/part2/" subdir="false" />
  </include>
</ojjsp-global-include>
```

この例には、次の 3 つの目的があります。

- `header.html` ファイルは、`dir1` ディレクトリ内またはその下にある JSP ページの最上部にインクルードされます。結果は、このディレクトリ内またはその下にある各 `.jsp` ファイルのページの最上部に次の `include` ディレクティブがある場合と同じになります。

```
<% include file="/header.html" %>
```

- `footer1.html` ファイルは、`dir1` ディレクトリまたはその `part1` サブディレクトリにある JSP ページの最下部にインクルードされます。結果は、これらのディレクトリ内にある各 `.jsp` ファイルのページの最下部に次の `include` ディレクティブがある場合と同じになります。

```
<% include file="/footer1.html" %>
```

- `footer2.html` ファイルは、`dir1` ディレクトリの `part2` サブディレクトリにある JSP ページの最下部にインクルードされます。結果は、このディレクトリ内にある各 `.jsp` ファイルのページの最下部に次の `include` ディレクティブが含まれている場合と同じになります。

```
<% include file="/footer2.html" %>
```

---

---

**注意:** 複数のヘッダー・ファイルまたはフッター・ファイルを単一の JSP ページにインクルードする場合の順序は、`ojjsp-global-include.xml` ファイルの `<include>` 要素の順序に従います。

---

---

**例: `translate_params` と同等のコード** 次の `ojjsp-global-include.xml` ファイルの例を考えてみます。

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE ojjsp-global-include SYSTEM 'ojjsp-global-include.dtd'>

<ojjsp-global-include>
  <include file="/WEB-INF/nls/params.jsf">
    <into directory="/" />
  </include>
</ojjsp-global-include>
```

`params.jsf` に、次のコードが含まれていると仮定します。

```
<% request.setCharacterEncoding(response.getCharacterEncoding()); %>
```

`params.jsf` ファイル（基本的に、`setCharacterEncoding()` メソッド・コール）は、アプリケーション・ルート・ディレクトリ内またはその下にある JSP ページの最上部にインクルードされます。つまり、このファイルは、アプリケーション内の JSP ページにインクルードされます。結果は、このディレクトリ内またはその下にある各 `.jsp` ファイルのページの最上部に次の `include` ディレクティブがある場合と同じになります。

```
<%@ include file="/WEB-INF/nls/parms.jsf" %>
```

B-31 ページの「[translate\\_params からの移行](#)」も参照してください。

## ojspc 事前変換ユーティリティ

ojspc ユーティリティは、JSP ページの事前変換用に OC4J から提供されています。事前変換の考慮点は、7-37 ページの「[JSP の事前変換](#)」および 7-40 ページの「[バイナリ・ファイルのみのデプロイ](#)」を参照してください。

次の各項で、ojspc 機能について説明します。

- [基本的な ojspc 機能の概要](#)
- [ojspc バッチ事前変換の概要](#)
- [ojspc のオプションのサマリー表](#)
- [ojspc のコマンドライン構文](#)
- [ojspc のオプションの説明](#)
- [ojspc の出力ファイル、場所および関連オプションのサマリー](#)

---

---

### 重要：

- JSP ページを事前変換する場合は、異なるオペレーティング・システムや JDK により問題が発生するなど、開発システムと実行時システム間の本質的な違いに注意してください。
  - ojspc を使用するには、JDK の `tools.jar` ファイルがクラスパスにある必要があります。これは、Oracle Application Server のインストール時に自動的に処理されます。
- 
- 

## 基本的な ojspc 機能の概要

単純な JSP (SQLJ JSP ではなく) ページの場合、ojspc のデフォルトの機能は次のとおりです。

- JSP ファイル (通常は `.jsp`) を、引数として直接取得するか、または引数として取得されたアーカイブ・ファイルから取得します。
- JSP トランスレータを起動して JSP ファイルを Java ページ実装クラス・コードに変換し、`.java` ファイルを作成します。
- Java コンパイラを起動して `.java` ファイルをコンパイルし、ページ実装クラスに対する `.class` ファイルを作成します。

次に、SQLJ JSP ページでのデフォルトの ojspc 機能を説明します。

- SQLJ JSP ファイル (.sqljsp ファイル、または page ディレクティブで language="sqlj" に指定されている JSP ファイル) を、引数として直接取得するか、または引数として取得されたアーカイブ・ファイルから取得します。
- JSP トランスレータを起動し、SQLJ JSP ページをページ実装クラスに対する .sqlj ファイルに変換します。
- Oracle SQLJ トランスレータを起動し、.sqlj ファイルを変換します。これによって、ページ実装クラスに対する .java ファイルが作成されます。また、ISO コード生成では、変換によって、SQLJ プロファイル・ファイル (デフォルトでは .ser Java リソース・ファイル) が作成されます。

SQLJ プロファイルと Oracle 固有のコード生成については、『Oracle9i SQLJ 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：** Oracle 固有のコード (プロファイルなし) の場合、SQLJ コード生成のデフォルト設定は -codegen=oracle です。ISO コード生成の場合は -codegen=iso を指定します。

---

---

- Java コンパイラを起動して .java ファイルをコンパイルし、ページ実装クラスに対する .class ファイルを作成します。

特定の状況 (この章で後述する -extres オプションの説明を参照) では、ojspc オプションによって、JSP トランスレータは、このコンテンツをページ実装クラスに格納せずに、静的なページのコンテンツに対する .res Java リソース・ファイルを作成します。

ojspc は JSP トランスレータを起動するため、ojspc の出力規則は、一般的なトランスレータの規則と同じです (該当する場合)。生成されたコードの機能、出力名の一般規則、生成されたパッケージとクラスの名前、生成されたファイルとその格納場所など、JSP トランスレータの出力に関する一般情報は、7-2 ページの「[JSP トランスレータの機能](#)」を参照してください。

---

---

**注意：** ojspc コマンドライン・ツールは、oracle.jsp.tool.Jspc クラスを起動するフロントエンド・ユーティリティです。

---

---

## ojspc バッチ事前変換の概要

Oracle9iAS リリース 2 (9.0.3) より前のリリースでは、ojspc で変換できるのは、JSP ファイル（または SQLJ JSP ファイル）のみでした。現在では、バッチ事前変換にアーカイブ・ファイル（JAR、WAR、EAR または ZIP の各ファイル）も使用できるようになりました。

---

---

**注意：** ojspc ユーティリティは、ファイルがアーカイブ・ファイルであるかどうか判断する際に、ファイル名拡張子を使用していません。ファイルの内部構造を検査してアーカイブ・ファイルであるかどうかを判断します。

---

---

ojspc のコマンドラインにアーカイブ・ファイル名が含まれている場合、ojspc は、デフォルトで次の手順を実行します。

1. アーカイブ・ファイルを開きます。
2. アーカイブ・ファイル内の .jsp、.sqljsp、.sqlj および .java ファイルをすべて変換およびコンパイルします。
3. 作成された .class ファイルと Java リソース・ファイルを、アーカイブ・ファイル内のネストされた JAR ファイルに追加し、処理中に作成された .java ファイルと .sqlj ファイルを破棄します。ネストされた JAR ファイル内の .class ファイルとリソース・ファイルには、ディレクトリ・パスがあるため、抽出時は、元の JSP ファイルが抽出後に変換された場合と同じディレクトリに配置されます。

デフォルトでは、元のアーカイブ・ファイルが一時記憶領域に抽出されます。次に、一時的なアーカイブ・ファイルが作成され、元のアーカイブ・ファイルのコンテンツがこの一時ファイルにコピーされます。さらに、事前変換で出力された .class ファイルとリソース・ファイルが一時ファイル内のネストされた JAR ファイルに追加され、元のアーカイブ・ファイルが削除されます。この一時ファイルには元のファイル名が割り当てられます。変換されたページが正常にコンパイルされるように、元のアーカイブ・ファイルはすべて抽出されます。新しい出力ファイル名を指定することもできます。その場合は、元のアーカイブ・ファイルが保持されます。

ネストされた JAR ファイルは、作成されたアーカイブ・ファイルの `_pages` ディレクトリに格納されます。ファイル名には、作成されたアーカイブ・ファイルのベース名（元のアーカイブ・ファイル名、または ojspc -output オプションに基づいた名前）が含まれ、.jar 拡張子が付きます。OC4J 9.0.4 実装では、たとえば、myarch.war のアーカイブ出力ファイル名の場合、ネストされた JAR ファイル名は `__oracle_jsp_myarch.jar` になります。実装の詳細は、今後のリリースで変更される可能性があります。アーカイブ・ファイルのベース名は、ネストされた JAR ファイル名に常に含まれます。

ネストされた JAR ファイル内のファイル・パスは、適切かつ予期されるとおりに、Java パッケージ名と、JSP include 文と forward 文の指定したファイル・パスに基づきます。



---

---

**注意：** OC4J 9.0.4 以上の実装では、ネストされたアーカイブ・ファイルのサポートは、EAR ファイル内の WAR ファイルに限定されています。この状況では、EAR ファイルのコンテンツが抽出された後、WAR ファイルのコンテンツが抽出および処理され、必要に応じて更新されます。その後、EAR ファイルの他のコンテンツが処理され、必要に応じて EAR ファイルが更新されます。

---

---

ojspc は、次の設定によって他の機能を提供しています。

- `-batchMask` オプションを使用すると、事前変換およびコンパイル用のファイル名拡張子を指定できます。指定した拡張子は、デフォルト (`*.jsp`、`*.sqljsp`、`*.sqlj` および `*.java`) のかわりに使用されます。
- `-output` オプションを使用すると、新しいアーカイブ・ファイル名を指定できます。デフォルトでは、ojspc は元のアーカイブ・ファイルを更新して、出力された `.class` ファイルとリソース・ファイルを追加します (場合によっては、`-deleteSource` オプションに基づいて、処理されたソース・ファイルを削除します)。元のアーカイブ・ファイルが変更されていないことを確認する場合は、`-output` オプションを有効にします。この場合、元のアーカイブ・ファイルのコンテンツがすべて、指定した出力アーカイブ・ファイルにコピーされ、元のファイルではなく、指定した出力アーカイブ・ファイルが更新されます。元のアーカイブ・ファイルは変更されません。デプロイには、元のファイルではなく新しいファイルを使用します。
- 出力されるアーカイブ・ファイルに処理されたソース・ファイルを含めない場合は、`-deleteSource` オプションを使用します。`-output` オプションも使用しない場合、処理が終了すると処理されたソース・ファイルがすべて元のアーカイブ・ファイルから削除されます。`-batchMask` オプションを使用しない場合、これは、すべての `.jsp`、`.sqljsp`、`.sqlj` および `.java` ファイルを構成します。それ以外の場合、これは、`-batchMask` 設定に指定されたすべてのファイルを構成します。

これらのオプションの使用例は、7-20 ページの「[ojspc のオプションの説明](#)」を参照してください。

## ojspc のオプションのサマリー表

表 7-1 に、ojspc 事前変換ユーティリティでサポートされているオプションを要約します。オプションの詳細は、7-20 ページの「[ojspc のオプションの説明](#)」で説明します。

表の 2 列目には、OC4J などオンデマンド変換環境での類似または関連した JSP 構成パラメータを示します。

---



---

**注意：** JServ 環境では、ojspc コマンドのかわりに、ojspc\_jserv コマンドを使用します。B-16 ページの「[JServ での ojspc の使用](#)」を参照。  
-staticTextInChars オプションは JServ には関係ないため、ojspc\_jserv ではサポートされていません。

---



---

表 7-1 ojspc 事前変換ユーティリティのオプション

| オプション         | 関連の JSP 構成パラメータ | 説明  | デフォルト                                    |
|---------------|-----------------|---|--|
| -addclasspath | (なし)            | javac のための追加クラスパス・エントリを指定します。   | 空 (追加パス・エントリなし)                          |
| -appRoot      | (なし)            | アプリケーションに対して相対的な、ページからの静的な include ディレクティブに対するアプリケーション・ルート・ディレクトリを指定します。                | カレント・ディレクトリ                              |
| -batchMask    | (なし)            | バッチ事前変換の場合、処理に使用するファイル・マスクを必要に応じて指定します。   | *.jsp、<br>*.sqljsp、<br>*.sqlj、<br>*.java |
| -dir (or -d)  | (なし)            | ojspc が、生成されたバイナリ・ファイル (.class とリソース) を格納する場所を指定します。バッチ事前変換には、このオプションを使用しないでください。       | カレント・ディレクトリ                              |
| -debug        | emit_debuginfo  | このフラグを使用すると、ojspc は、デバッグ用に、元の .jsp ファイルに対する行マップを生成します。                                  | false                                    |
| -deleteSource | (なし)            | バッチ事前変換の場合、このフラグを使用すると、出力されるアーカイブ・ファイルから処理されたソース・ファイルが削除されます (つまり、アーカイブ・ファイルにコピーされません)。 | false                                    |
| -extend       | (なし)            | 生成されたページ実装クラスが拡張するクラスを指定します。バッチ事前変換には、このオプションを使用しないでください。                               | 空  |

表 7-1 ojspc 事前変換ユーティリティのオプション (続き)

| オプション              | 関連の JSP 構成<br>パラメータ  | 説明  | デフォルト                        |
|--------------------|----------------------|---|------------------------------|
| -extraImports      | extra_imports        | このオプションを使用して、JSP のデフォルトより多いインポートを追加します。   | 空                            |
| -extres            | external_resource    | このフラグ使用すると、ojspc は JSP ファイルから静的なテキスト用の外部リソース・ファイルを生成します。  | false                        |
| -forgiveDupDirAttr | forgive_dup_dir_attr | このフラグを使用して、単一の JSP 変換単位内で同じディレクティブ属性に重複する設定がある場合に、JSP 1.2 の変換エラーを回避します。   | false                        |
| -help (or -h)      | (なし)                 | このフラグを使用すると、ojspc は使用情報を表示します。  | false                        |
| -implement         | (なし)                 | 生成されたページ実装クラスが実装するインタフェースを指定します。バッチ事前変換には、このオプションを使用しないでください。   | 空                            |
| -noCompile         | javaccmd             | このフラグを使用すると、ojspc は生成されたページ実装クラスをコンパイルしません。   | false                        |
| -noTldXmlValidate  | no_tld_xml_validate  | このフラグを使用して、TLD の XML 妥当性チェックを無効にします。デフォルトでは、TLD の妥当性チェックが実行されます。  | false                        |
| -oldIncludeFromTop | old_include_from_top | このフラグを使用して、Oracle9iAS リリース 2 より前の Oracle JSP の動作との下位互換性を維持するために、ネストされた include ディレクティブにあるページの場所がトップレベルのページに対して相対的であることを指定します。 | false                        |
| -output            | (なし)                 | バッチ事前変換の場合に、出力アーカイブ・ファイル名を必要に応じて指定します。  | 元のアーカイブ・ファイル                 |
| -packageName       | (なし)                 | 生成されたページ実装クラスのパッケージ名を指定します。   | 空 (パッケージ名は、.jsp ファイルの場所に基づく) |
| -reduceTagCode     | reduce_tag_code      | このフラグを使用して、カスタム・タグを使用するために生成されたコードのサイズをさらに縮小します。  | false                        |

表 7-1 ojspc 事前変換ユーティリティのオプション (続き)

| オプション                 | 関連の JSP 構成<br>パラメータ    | 説明   | デフォルト       |
|-----------------------|------------------------|--|-------------|
| -reqTimeIntrospection | req_time_introspection | コンパイル時にイントロスペクションが実行できない場合は、このフラグを有効にして、JavaBean のイントロスペクションをリクエスト時に許可します。   | false       |
| -S-<sqlj_option>      | sqljcmd                | Oracle SQLJ オプション (SQLJ JSP ページ用) の前にある -s 接頭辞を使用します。  | 空           |
| -srcdir               | (なし)                   | ojspc が、生成されたソース・ファイル (.java と .sqlj) を格納する場所を指定します。バッチ事前変換には、このオプションを使用しないでください。  | カレント・ディレクトリ |
| -staticTextInChars    | static_text_in_chars   | このフラグを使用すると、JSP トランスレータは、JSP ページの静的なテキストをバイトではなく文字として生成します。  | false       |
| -tagReuse             | tags_reuse_default     | JSP タグ・ハンドラの再利用モードを指定します。実行時モデルには runtime を、コンパイル時モデルには compiletime_with_release または compiletime を、タグ・ハンドラの再利用を禁止するには none を指定します。 | runtime     |
| -verbose              | (なし)                   | このフラグを使用すると、ojspc は実行時の状態情報を出力します。   | false       |
| -version              | (なし)                   | このフラグを使用すると、ojspc は、JSP のバージョン番号を表示します。  | false       |
| -xmlValidate          | xml_validate           | このフラグを使用して、web.xml ファイルの XML 妥当性チェックを要求します。デフォルトでは、web.xml の妥当性チェックは実行されません。   | false       |

## ojspc のコマンドライン構文

次に、一般的な ojspc コマンドライン構文 (% はシステム・プロンプトを表します) を示します。

```
% ojspc [option_settings] file_list
```

ファイル・リストには JSP ファイル (SQLJ JSP ファイルを含む) と他のソース・ファイル (.sqlj および .java)、またはアーカイブ・ファイル (JAR、WAR、EAR または ZIP ファイル) を含めることができます。

構文に関する注意は、次のとおりです。

- 複数の JSP ファイルの変換では、すべてのファイルで同じキャラクタ・セット (デフォルト設定、または page ディレクティブの設定のいずれか) を使用する必要があります。
- ファイル・リストのファイル名は、空白で区切ります。
- オプション・リスト内のオプション名とオプション値の間のデリミタには空白を使用します。
- オプション名は大 / 小文字を区別しませんが、オプション値 (パッケージ名、ディレクトリ名、クラス名、インタフェース名など) は、通常、大 / 小文字を区別します。
- コマンドラインにオプション名を入力して、デフォルトでは無効のブール型のオプション (フラグ) を有効にします。たとえば、`-extres true` ではなく、`-extres` と入力します。
- 実際は、オプション設定とファイル名は任意の順序に入れ替えることができます。

次に 2 つの例を示します。

```
% ojspc -dir /myapp/mybindir -srcdir /myapp/mysrcdir -extres MyPage.sqljsp MyPage2.jsp
```

```
% ojspc -deleteSource myapp.war
```

## ojspc のオプションの説明

この項では、ojspc のオプションについて詳細に説明します。

**-addclasspath** (完全修飾パス、ojspc のデフォルト:空)

生成されたページ実装クラス・ソースのコンパイル時に使用する、javac のための追加クラスパス・エントリを指定します。指定しない場合、javac はシステムのクラスパスのみを使用します。

---

---

**注意:** -addclasspath 設定は、SQLJ トランスレータでも SQLJ JSP ページに対して使用します。

---

---

**-appRoot** (完全修飾パス、ojspc のデフォルト:カレント・ディレクトリ)

アプリケーション・ルート・ディレクトリを指定します。デフォルトは、ojspc を実行したときのカレント・ディレクトリです。

指定したアプリケーション・ルート・ディレクトリ・パスは、次の場合に使用されます。

- 変換するページの静的な include ディレクティブで使用

指定したディレクトリ・パスは、変換するページの include ディレクティブ内でアプリケーション相対 (コンテキスト相対) パスの前に付加されます。

- ページ実装クラスのパッケージを決定するときに使用

パッケージのディレクトリは、変換されるファイルの場所に基づいて、アプリケーション・ルート・ディレクトリに対して相対的に決定されます。このパッケージによって、出力ファイルの格納場所も決定します (7-32 ページの「[ojspc の出力ファイル、場所および関連オプションのサマリー](#)」を参照)。

このオプションは、たとえば、ojspc を他のディレクトリから実行した場合に、インクルード・ファイルを検出できないときに必要になります。

次に例を示します。

- 次のファイルを変換すると仮定します。

```
/abc/def/ghi/test.jsp
```

- 次のように、カレント・ディレクトリの /abc から ojspc を実行します (% は UNIX プロンプトを表します)。

```
% cd /abc
% ojspc def/ghi/test.jsp
```

- test.jsp ページには次の include ディレクティブがあります。

```
<%@ include file="/test2.jsp" %>
```

- 次のように、test2.jsp ページは /abc ディレクトリ内にあります。

```
/abc/test2.jsp
```

この例では、デフォルトのアプリケーション・ルートの設定がカレント・ディレクトリの /abc ディレクトリであるため、-appRoot の設定は不要です。include ディレクティブでは、アプリケーションに対して相対的な /test2.jsp 構文（「/」で始まります）を使用するため、インクルード・ページの場所は、/abc/test2.jsp になります。

この場合のパッケージは \_def.\_ghi です。このパッケージは、ojspc を実行したときのカレント・ディレクトリに対して相対的な test.jsp の場所に基づいて決定されます（カレント・ディレクトリはデフォルトのアプリケーション・ルートです）。これに従って、出力ファイルが格納されます。

ただし、他のディレクトリ（/home/mydir など）から ojspc を実行する場合は、次の例に示すように、-appRoot の設定が必要です。

```
% cd /home/mydir
% ojspc -appRoot /abc /abc/def/ghi/test.jsp
```

パッケージは \_def.\_ghi のままですが、ディレクトリは、指定のアプリケーション・ルート・ディレクトリに対して相対的な test.jsp の場所に基づいて決定されます。

---

**注意：** 一般的に、指定のアプリケーション・ルート・ディレクトリは、変換される JSP ページが格納されるディレクトリの親ディレクトリになります。

---

**-batchMask**（アーカイブのバッチ処理を行うファイル・マスク、ojspc のデフォルト：説明を参照）

バッチ事前変換の場合、このオプションを使用して、アーカイブ・ファイルで処理されるソース・ファイルを指定できます。デフォルトでは、.jsp、.sqljsp、.sqlj および .java ファイルがすべて処理されます。-batchMask オプションによって指定されたファイル・マスクは、これらのデフォルトに追加するのではなく、デフォルトのかわりに使用されます。

ファイル・マスクのリストの前後は引用符で囲み、リスト内ではデリミタにカンマまたはセミコロンを使用します。ファイル・マスクの前後にある空白は無視されます。マスクにディレクトリを含めることができます。

OC4J 9.0.4 以上の実装では、-batchMask 実装で、標準のワイルドカード・パターン・マッチングが完全にサポートされています。

デフォルト設定の場合、次の 2 つの例と同じです。

```
% ojspc myapp.war
% ojspc -batchMask "*.jsp,*.sqljsp,*.sqlj,*.java" myapp.war
```

この次の例では、.java および .sqlj ファイルの処理が削除されていますが、.jspf および .jspx ファイルの処理が追加されています。

```
% ojspc -batchMask "*.jspx,*.jspx,*.jsp,*.sqljsp" myapp.war
```

次の例では、.sqljsp、.sqlj または .java ファイルは処理されていません。名前が「abc」で始まり、アーカイブ・ファイルの上位レベルのサブディレクトリにある .jsp ファイルのみが処理されます。

```
% ojspc -batchMask "*/abc*.jsp" myapp.zip
```

次の例は、前述の例と同じですが、アーカイブ・ファイルの上位レベルにあり、名前が「abc」で始まる .jsp ファイルも処理されます。

```
% ojspc -batchMask "abc*.jsp, */abc*.jsp" myapp.jar
```

最後の例では、ファイル a.jspc と、「My」で始まり、mydir/subdir のサブディレクトリであるディレクトリにあり、パターン「t?st」（2 番目の文字が任意の文字、「test」、「tast」、「tust」など）と一致する .jsp ファイルが明確に処理されます。

```
% ojspc -batchMask "mydir/subdir/t?st/My*.jsp, a.jspc" myapp.ear
```

---

---

**重要：** このオプションで指定するファイル・マスクでは大 / 小文字を区別しません。

---

---

**-debug** (ブール値、ojspc のデフォルト: false)

このフラグを有効にすると、ojspc は、デバッグ用に元の JSP ファイルに対する行マップを生成します。無効にすると、行マップは、生成されたページ実装クラスに対して行われません。

このフラグは、Oracle JDeveloper を使用する場合などにソース・レベルの JSP デバッグで役立ちます。

---

---

**注意：** オンデマンド変換の場合は、JSP の emit\_debuginfo 構成パラメータが同じ機能を提供します。

---

---



**-deleteSource** (ブール値、ojspc のデフォルト: false)

バッチ事前変換で、出力されるアーカイブ・ファイルに、処理されたソース・ファイルを含めない場合は、このフラグを有効にします。デフォルトでは、.jsp、.sqljsp、.sqlj および .java ファイルが対象となります。それ以外の場合は、-batchMask オプションのファイル・マスクと一致するファイルのみが対象となります。生成された .java および .sqlj ファイルも、通常どおり破棄されます。

-output オプションを使用しない場合、元のアーカイブ・ファイルのコンテンツが上書きされ、処理後に処理されたソース・ファイルが削除されます。-output オプションを使用すると、処理されたソース・ファイルは、指定の出力アーカイブ・ファイルにはコピーされません (元のアーカイブ・ファイルは変更されません)。

---



---

### 注意:

- 名前がデフォルトのファイル拡張子と一致しない (-batchMask オプションを使用しない場合) ファイル、または -batchMask オプションを使用して指定した名前マスクと一致しないファイルは、-deleteSource オプションでは破棄されません。これらのファイルは、必要に応じて、出力されたアーカイブ・ファイルから削除する必要があります。これは特に、静的にインクルードされたソース・ファイルの場合に当てはまります。静的にインクルードされたソース・ファイルは、独自に変換できないため、.jsp 拡張子、またはファイルを独自に変換しようとする他の拡張子を使用できません。
  - JSP ソース・ファイルをデプロイしない場合に、-deleteSource を使用するには、ターゲットの JSP 実行時環境を、使用可能なソース・ファイルがない場合でも正しく動作するように構成する必要があります。7-40 ページの「[バイナリ・ファイルのみを実行するための OC4J JSP コンテナの構成](#)」を参照。
- 
- 

**-dir** (完全修飾パス、ojspc のデフォルト: カレント・ディレクトリ)

ojspc が生成されたバイナリ・ファイル (.class ファイルと Java リソース・ファイル) を格納するベース・ディレクトリを指定します (-extres オプションによって静的なコンテンツに対して作成された .res ファイルは、Java リソース・ファイルです。SQLJ トランスレータが、SQLJ JSP ページに対して SQLJ の ISO コード生成で作成した .ser プロファイル・ファイルも同様です)。ショートカットとして、-d も使用できます。

指定されたパスは、ファイルのシステム・パスとして (アプリケーション相対パスまたはページ相対パスではなく) 取得されるため、ディレクトリはすでに存在している必要があります。

指定されたディレクトリ下のサブディレクトリは、パッケージに応じて、必要な場合に自動的に作成されます。詳細は、7-32 ページの「[ojspc の出力ファイル、場所および関連オプションのサマリー](#)」を参照してください。

デフォルトでは、カレント・ディレクトリ (ojspc を実行したときのカレント・ディレクトリ) を使用します。

このオプションを使用して、生成されたバイナリ・ファイルを未使用のディレクトリに格納することをお勧めします。これによって、作成されたファイルを簡単に識別できます。

---

---

**注意：**

- バッチ事前変換には、`-dir` を使用しないでください。
- ディレクトリ名に空白を使用できる Windows などの環境では、ディレクトリ名を引用符で囲んでください。

---

---

**-extend** (完全修飾 Java クラス名、ojspc のデフォルト: 空)

生成されたページ実装クラスが拡張する Java クラスを指定します。

---

---

**注意：** バッチ事前変換には、`-extend` を使用しないでください。

---

---

**-extraImports** (インポート・リスト、ojspc のデフォルト: 空)

3-9 ページの「[デフォルト・パッケージのインポート](#)」に説明されているように、Oracle9iAS リリース 2 (9.0.3) より前の場合に比べて、OC4J JSP コンテナには、各 JSP ページにインポートされるパッケージの小規模なデフォルト・リストがあります。このデフォルト・リストは、JSP の仕様にに基づいています。ただし、`-extraImports` オプションを使用して、追加インポート用のパッケージ名または完全修飾されたクラス名を指定することによって、コードの更新を回避できます。名前を指定する場合は、次の例のように引用符で囲み、カンマまたはセミコロンで区切る必要があります。

```
% ojspc -extraImports "java.util.*,java.io.*" foo.jsp
```

---

---

**注意：**

- 引用符で囲まれたパッケージ名またはクラス名の前後にある空白は無視されます。
- オンデマンド変換の場合は、JSP の `extra_imports` 構成パラメータが同じ機能を提供します。
- `-extraImports` のかわりに、グローバル・インクルードを使用することもできます。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。

**-extres** (ブール値、ojspc のデフォルト: false)

このフラグを有効にすると、ojspc は、ページの静的なコンテンツを、生成されたページ実装クラスのサービス・メソッドではなく、Java リソース・ファイルに格納します。

リソース・ファイル名は、JSP ページ名に基づいて付けられます。現在の OC4J の JSP 実装では、コア名が JSP 名と同じになりますが (JSP 名に特殊文字が含まれない場合)、接頭辞のアンダースコア ( `_` ) と接尾辞の `.res` が名前に付けられます。たとえば、`MyPage.jsp` を変換すると、通常の実出力以外に、`_MyPage.res` が作成されます。ただし、名前の生成に関する正確な実装は、今後のリリースで変更される場合があります。

リソース・ファイルは、出力された `.class` ファイルと同じディレクトリに格納されます。

1 ページに大量の静的なコンテンツが存在する場合は、この技法によって変換速度が向上し、ページの実行速度も向上します。詳細は、6-9 ページの「[大量の静的なコンテンツまたは重要なタグ・ライブラリの使用に対する対処](#)」を参照してください。

---

---

**注意：** オンデマンド変換の場合は、JSP の `external_resource` 構成パラメータが同じ機能を提供します。

---

---

**-forgiveDupDirAttr** (ブール値、ojspc のデフォルト: false)

このフラグを有効にすると、単一の JSP 変換単位 (JSP ページおよび `include` ディレクティブを使用してインクルードされたページ) で同じディレクティブ属性に対して重複する設定がある場合でも、JSP 1.2 以上での変換エラーを回避します。

JSP 仕様では、複数のディレクティブ属性 (`page` ディレクティブの `import` 属性を除く) が単一の JSP 変換単位内で 2 回以上設定されていないことを、JSP コンテナが確認する必要があります。詳細は、6-12 ページの「[page ディレクティブ属性の重複設定の禁止](#)」を参照してください。

JSP 仕様 1.1 では、このような制限はありませんでした。OC4J では、下位互換性のために `-forgiveDupDirAttr` オプションを提供しています。

---

---

**注意：** オンデマンド変換の場合は、JSP の `forgive_dup_dir_attr` 構成パラメータが同じ機能を提供します。

---

---

**-help** (ブール値、ojspc のデフォルト: false)

このオプションを使用すると、ojspc は、使用情報を表示してから終了します。簡単な方法として、`-h` も使用できます。

**-implement** (完全修飾 Java インタフェース名、ojspc のデフォルト: 空)

生成されたページ実装クラスが実装する Java インタフェースを指定します。

---

---

**注意：** バッチ事前変換には、`-implement` を使用しないでください。

---

---

**-noCompile** (ブール値、ojspc のデフォルト : `false`)

このフラグを有効にすると、ojspc は、生成されたページ実装クラスの Java ソースをコンパイルしません。このオプションは、なんらかの理由で、後で代替の Java コンパイラを使用してコンパイルする場合などに使用します。

---

---

**注意：**

- オンデマンド変換の場合は、JSP の `javaccmd` 構成パラメータが関連機能を提供します。この機能によって、完全な Java コンパイラのコマンドラインを指定できます。必要に応じて代替コンパイラを使用できます。
  - SQLJ JSP ページの場合は、`-noCompile` を有効にすると、SQLJ 変換は実行されますが、Java コンパイルは実行されません。
- 
- 

**-noTldXmlValidate** (ブール値、ojspc のデフォルト : `false`)

アプリケーションのタグ・ライブラリ・ディスクリプタ (TLD) に対する XML 妥当性チェックを実行しない場合は、このフラグを有効にします。デフォルトでは、TLD の妥当性チェックが実行されます。

関連情報は、8-7 ページの「[TLD の妥当性チェックおよび機能の概要](#)」を参照してください。

---

---

**注意：** オンデマンド変換の場合は、JSP の `no_tld_xml_validate` 構成パラメータが同じ機能を提供します。

---

---

**-oldIncludeFromTop** (ブール値、ojspc のデフォルト : `false`)

このオプションは、`include` ディレクティブの機能について、Oracle9iAS リリース 2 より前の Oracle JSP バージョンとの下位互換性を維持するために使用します。このフラグを有効にすると、ネストされた `include` ディレクティブ内のページの場所がトップレベルのページに対して相対的になります。それ以外の場合は、1 つ上位の親ページに対して相対的になります。これは JSP 仕様に準拠しています。

---

---

**注意：** オンデマンド変換の場合は、JSP の `old_include_from_top` 構成パラメータが同じ機能を提供します。

---

---

**-output** (アーカイブ・ファイル名、ojspc のデフォルト: なし)

バッチ事前変換の場合、元のアーカイブ・ファイルを更新するのではなく、出力用に新しいアーカイブ・ファイルを指定するには、`-output` オプションを使用します。この場合は、元のアーカイブ・ファイルのすべてのコンテンツが、指定したアーカイブ・ファイルにコピーされます。次に、事前変換で出力された `.class` ファイルおよびリソース・ファイルが、指定したファイル内のネストされた JAR ファイルに格納されます (`-deleteSource` が有効になっている場合、指定したファイルからソース・ファイルが削除されます)。元のアーカイブ・ファイルは変更されません。デプロイには、元のファイルではなく新しいファイルを使用します (ネストされた JAR ファイルの詳細は、7-14 ページの「[ojspc バッチ事前変換の概要](#)」を参照)。

`-output` オプションを指定しないと、元のアーカイブ・ファイルが更新され、新しいアーカイブ・ファイルは作成されません。

次に、`-output` の使用例を示します。

```
% ojspc -output myappout.war myapp.war
```

**-packageName** (完全修飾パッケージ名、ojspc のデフォルト: `.jsp` ファイルの場所に基づいて生成)

Java のドット構文を使用して生成されたページ実装クラスのパッケージ名を指定します。

このオプションを設定しない場合のパッケージ名は、`ojspc` を実行したときのカレント・ディレクトリに対して相対的な `.jsp` ファイルの場所に基づいて決定されます。

たとえば、`/myapproot` ディレクトリから `ojspc` を実行し、`.jsp` ファイルが `/myapproot/src/jspsrc` ディレクトリに存在する場合について考えます (% は UNIX プロンプトを表します)。

```
% cd /myapproot
% ojspc -packageName myroot.mypackage src/jspsrc/Foo.jsp
```

この結果、パッケージ名として `myroot.mypackage` が使用されます。

この例で `-packageName` オプションが使用されなかった場合、JSP トランスレータは (現在の実装内で) デフォルトの `_src._jspsrc` をパッケージ名として使用します (実装の詳細は、今後のリリースで変更される場合があります)。

**-reduceTagCode** (ブール値、ojspc のデフォルト: `false`)

Oracle JSP を実装すると、カスタム・タグを使用するために生成されるコードのサイズが縮小されますが、このフラグを有効にすると、さらにサイズが縮小されます。ただし、タグ・ハンドラの再利用に関しては、パフォーマンスに影響を与える場合があります。8-41 ページの「[タグ・ハンドラのコード生成](#)」を参照。

---

---

**注意：** オンデマンド変換の場合は、JSP の `reduce_tag_code` 構成パラメータが同じ機能を提供します。

---

---

**-reqTimeIntrospection** (ブール値、ojspc のデフォルト: false)

このフラグを有効にすると、コンパイル時にイントロスペクションが実行できなかった場合、リクエスト時に **JavaBean** のイントロスペクションを実行できます。ただし、有効なコンパイル時のイントロスペクションが正常終了した場合、このフラグの設定に関係なく、リクエスト時のイントロスペクションは実行されません。

リクエスト時のイントロスペクションの使用例として、タグ・ハンドラによって、タグ補足情報クラスの `VariableInfo` インスタンスにある一般的な `java.lang.Object` インスタンスが変換時とコンパイル時に戻される一方で、実際には、特定のオブジェクトがリクエスト時 (実行時) に生成される場合を想定します。この場合、`-reqTimeIntrospection` が有効になっていると、JSP コンテナはリクエスト時までイントロスペクションを遅延します (`VariableInfo` の使用法は、8-41 ページの「[スクリプト変数、宣言およびタグ補足情報クラス](#)」を参照してください)。

このフラグには、`if..then..else` ループの別の分岐などで、**Bean** を 2 回宣言できるという効果もあります。次の例を考えてみます。`-reqTimeIntrospection` が有効になっていないと、このコードにより、解析例外が発生します。有効になっている場合、コードはエラーなしで機能します。

```
<% if (cond) { %>
    <jsp:useBean id="foo" class="pkgA.Foo1" />
<% } else { %>
    <jsp:useBean id="foo" class="pkgA.Foo2" />
<% } %>
```

---

---

**注意：** オンデマンド変換の場合は、JSP の `req_time_introspection` 構成パラメータが同じ機能を提供します。

---

---

**-S-<sqlj\_option> <value>** ( SQLJ オプション設定の前にある -S、ojspc のデフォルト: 空)

SQLJ JSP ページの場合は、ojspc `-S` オプションを使用して、**Oracle SQLJ** オプションを SQLJ トランスレータに渡します。複数の `-S` を使用できます。それぞれ 1 つの SQLJ オプション設定を指定します。

SQLJ トランスレータを直接実行する場合と異なり、**SQLJ** オプションとその値は空白で区切ります (他の ojspc オプションとの一貫性を維持するため)。

次に例を示します (% は UNIX プロンプトを表します)。

```
% ojspc -S-codegen iso -dir /myapproot/mybindir MyPage.jsp
```

このコマンドは、デフォルトの Oracle 固有のコードではなく ISO 標準コードを生成するように、SQLJ に指示しています。

次に、別の例を示します。

```
% ojspc -S-codegen iso -S-ser2class true -dir /myapproot/mybindir MyPage.jsp
```

このコマンドも、再度 ISO 標準コードを生成するように SQLJ に指示しています。また、プロファイルを .class ファイルに変換するために、-ser2class オプションを有効にしています。

---

---

**注意：** 前述の例で示すように、-s オプションを設定して SQLJ のプール型オプションを有効にするときは、明示的に true に設定する必要があります。これは、明示的に true を設定しない ojspc のプール型オプションと対照的です。

---

---

特定の Oracle SQLJ オプションについて、次の内容に注意してください。

- SQLJ -encoding オプションを使用しないでください。かわりに、JSP ページの page ディレクティブにある contentType 属性または pageEncoding 属性を使用してください。
- ojspc -addclasspath オプションを使用する場合は、SQLJ -classpath オプションを使用しないでください。
- ojspc -noCompile オプションを使用する場合は、SQLJ -compile オプションを使用しないでください。
- ojspc -dir オプションを使用する場合は、SQLJ -d オプションを使用しないでください。
- ojspc -srcdir オプションを使用する場合は、SQLJ -dir オプションを使用しないでください。

Oracle SQLJ トランスレータのオプションについては、『Oracle9i SQLJ 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：** オンデマンド変換の場合は、JSP の sqljcmd 構成パラメータが関連機能を提供します。この機能によって、必要なオプション設定で完全な SQLJ コマンドラインを入力できます。必要な場合は代替の SQLJ コンパイラを使用できます。

---

---

**-srcdir** (完全修飾パス、ojspc のデフォルト:カレント・ディレクトリ)

ojspc が生成されたソース・ファイル (.sqlj ファイル (SQLJ JSP ページ用) と .java ファイル) を格納するベース・ディレクトリを指定します。

指定されたパスは、アプリケーション相対パスまたはページ相対パスではなく、ファイルのシステム・パスとして取得されます。ディレクトリはすでに存在している必要があります。

指定されたディレクトリ下のサブディレクトリは、パッケージに応じて、必要な場合に自動的に作成されます。詳細は、7-32 ページの「[ojspc の出力ファイル、場所および関連オプションのサマリー](#)」を参照してください。

デフォルトでは、カレント・ディレクトリ (ojspc を実行したときのカレント・ディレクトリ) を使用します。

このオプションを使用して、生成されたソース・ファイルを未使用のディレクトリに格納することをお勧めします。これによって、作成されたファイルを簡単に識別できます。

---

---

**注意:**

- バッチ事前変換には、-srcdir を使用しないでください。
  - ディレクトリ名に空白を使用できる Windows などの環境では、ディレクトリ名を引用符で囲んでください。
- 
- 

**-staticTextInChars** (ブール値、ojspc のデフォルト: false)

このフラグを有効にすると、JSP トランスレータは、JSP ページの静的なテキストをバイトではなく文字として生成します。デフォルト設定の false では、静的なテキスト・ブロックの出力で、パフォーマンスが改善されます。

次の例に示すように、実行時に文字エンコーディングをアプリケーションで動的に変更する必要がある場合は、このフラグを有効にします。

```
<% response.setContentType("text/html; charset=UTF-8"); %>
```

---

---

**注意:** オンデマンド変換の場合は、JSP の `static_text_in_chars` 構成パラメータが同じ機能を提供します。

---

---

**-tagReuse** (タグ・ハンドラの再利用のモード、ojspc のデフォルト: runtime)

このオプションを使用して、タグ・ハンドラの再利用 (タグ・ハンドラ・インスタンスのプーリング) のモードを指定します。

- タグ・ハンドラの再利用を無効にするには、none に設定します。この設定は、特定の JSP ページで、JSP ページ・コンテキスト属性 `oracle.jsp.tags.reuse` を true の値に設定するとオーバーライドできます。



- タグ・ハンドラ再利用の実行時モデルを有効にするには、デフォルトの runtime に設定します。この設定は、特定の JSP ページで、JSP ページ・コンテキスト属性 `oracle.jsp.tags.reuse` を `false` の値に設定するとオーバーライドできます。
- タグ・ハンドラ再利用のコンパイル時モデルをその基本モードで有効にするには、`completetime` に設定します。
- タグ・ハンドラ再利用のコンパイル時モデルを、その「解放」モードで有効にするには、`completetime_with_release` に設定します。この場合、タグ・ハンドラの `release()` メソッドは、指定したページで指定したタグ・ハンドラが使用されている間にコールされます。

---

---

**注意：**

- 下位互換性のために、runtime と等価の `true` 設定、none と等価の `false` 設定もサポートされます。
  - オンデマンド変換の場合は、JSP の `tags_reuse_default` 構成パラメータが同じ機能を提供します。
- 
- 

タグ・ハンドラの再利用に関する詳細は、8-38 ページの「[実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)」を参照してください。

**-verbose** (ブール値、ojspc のデフォルト: `false`)

このフラグを有効にすると、ojspc は、実行時の変換手順をレポートします。

次の例は、`myerror.jsp` の変換に対する `-verbose` の出力を示します。(この例では、ojspc は `myerror.jsp` が存在するディレクトリから実行されます。% は UNIX プロンプトを表します。)

```
% ojspc -verbose myerror.jsp
Translating file: myerror.jsp
1 JSP files translated successfully.
Compiling Java file: ./_myerror.java
```

**-version** (ブール値、ojspc のデフォルト: `false`)

このオプションを使用すると、ojspc は、JSP のバージョン番号を表示してから終了します。

**-xmlValidate** (ブール値、ojspc のデフォルト: `false`)

アプリケーション `web.xml` ファイルに対して XML 妥当性チェックを実行する場合は、このフラグを有効にします。Tomcat JSP リファレンス実装では XML 妥当性チェックを実行しないため、このフラグはデフォルトでは無効になっています。

---

---

**注意：** オンデマンド変換の場合は、JSP の `xml_validate` 構成パラメータが同じ機能を提供します。

---

---

## ojspc の出力ファイル、場所および関連オプションのサマリー

ojspc は、デフォルトでは、JSP トランスレータがオンデマンド変換で生成したのと同じファイル・セットを生成し、ojspc を実行したときのカレント・ディレクトリ内またはその下にそのファイル・セットを格納します（ここでは、バッチ事前変換の場合は考慮していません）。

次のファイルが生成されます。

- SQLJ JSP ページの場合は `.sqlj` ソース・ファイル（バッチ事前変換の場合はコンパイル後に破棄されます。）
- `.java` ソース・ファイル（バッチ事前変換の場合はコンパイル後に破棄されます。）
- ページ実装クラスに対する `.class` ファイル
- ISO コード生成を使用する SQLJ JSP ページの場合は Java リソース・ファイル（`.ser`）、またはオプションで SQLJ プロファイル用の `.class` ファイル
- 必要に応じて、ページの静的なテキストの Java リソース・ファイル（`.res`）

JSP トランスレータが生成するファイルの詳細は、7-6 ページの「[生成されるファイルとその格納場所](#)」を参照してください。

7-20 ページの「[ojspc のオプションの説明](#)」で説明した一般的に使用されるオプション（ファイルの生成と格納に関連する ojspc のオプション）のサマリーは、次のとおりです。

- `-appRoot` は、アプリケーション・ルート・ディレクトリを指定します。
- `-srcdir` は、ソース・ファイルを指定の場所に格納します（バッチ事前変換の場合は関係ありません）。
- `-dir` は、バイナリ・ファイル（`.class` ファイルと Java リソース・ファイル）を指定の場所に格納します（バッチ事前変換の場合は関係ありません）。
- `-noCompile` は、生成されたページ実装クラスのソースをコンパイルしません。

その結果、`.class` ファイルは作成されません。SQLJ JSP ページの場合、変換済の `.java` ファイルは作成されますが、コンパイルはされません。

- `-extres` は、静的なテキストを Java リソース・ファイルに格納します。
- `-S-ser2class`（SQLJ JSP ページ用のみ、および ISO 標準 SQLJ コード生成用のみの、SQLJ の `-ser2class` オプション）は、`.ser` Java リソース・ファイルではなく、`.class` ファイル内に SQLJ プロファイルを生成します。

出力ファイルの格納に関してバッチ事前変換を考慮しない場合、カレント・ディレクトリ (-dir オプションと -srcdir オプションで指定するディレクトリ) の下のディレクトリ構造は、パッケージに基づきます。パッケージのディレクトリは、変換されるファイルの場所に基づいて、アプリケーション・ルート・ディレクトリに対して相対的に決定されます。つまり、カレント・ディレクトリまたは -appRoot で指定されたディレクトリのいずれかです。

たとえば、次の ojspc を実行すると仮定します (% は UNIX プロンプトを表します)。

```
% cd /abc
% ojspc def/ghi/test.jsp
```

パッケージは `_def._ghi` で、出力ファイルは `/abc/_def/_ghi` ディレクトリに格納されます。このディレクトリの `_def/_ghi` サブディレクトリ構造は、プロセスの一部として作成されます。

-dir オプションと -srcdir オプションを使用して別の出力場所を指定すると、指定したディレクトリの下に `_def/_ghi` サブディレクトリ構造が作成されます。

次のように、別のディレクトリから ojspc を実行すると仮定します。

```
% cd /home/mydir
% ojspc -appRoot /abc /abc/def/ghi/test.jsp
```

パッケージは `_def._ghi` のままですが、ディレクトリは、指定のアプリケーション・ルートに対して相対する `test.jsp` の場所に基づいて決定されます。出力ファイルは、`/home/mydir/_def/_ghi` ディレクトリまたは -dir オプションと -srcdir オプションで指定したディレクトリ下の `_def/_ghi` サブディレクトリに格納されます。いずれの場合も、`_def/_ghi` サブディレクトリ構造は、プロセスの一部として作成されます。

## JSP デプロイに関する考慮事項

次の各項では、デプロイに関する一般的な考慮事項とシナリオについて説明します（ほとんどの内容は、ターゲット環境から独立しています）。

- [EAR/WAR デプロイの概要](#)
- [Oracle JDeveloper によるアプリケーションのデプロイ](#)
- [JSP の事前変換](#)
- [バイナリ・ファイルのみのデプロイ](#)

## EAR/WAR デプロイの概要

この項では、OC4J のデプロイ機能と標準の WAR デプロイ機能の概要を説明します。

Oracle Application Server 環境での OC4J に対するデプロイの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

### OC4J のデプロイ機能

OC4J では、標準の EAR (Enterprise Archive) ファイルを使用して、各アプリケーションをデプロイします。OC4J の `server.xml` ファイルの `<application>` 要素を使用して、アプリケーションの名前、および EAR ファイルの名前と場所を指定します。このファイルは、OC4J 構成ファイルのディレクトリにあります。

---

---

**注意：** Oracle Application Server のディレクトリ・パスは構成可能です。スタンドアロン OC4J の場合、構成ファイルのディレクトリは、デフォルトで `j2ee/home/config` です。

---

---

Oracle Application Server 環境では、Enterprise Manager を使用してデプロイおよび構成を行い、`server.xml` や他の構成ファイルを直接更新しないでください。詳細は、『Oracle Application Server 10g 管理者ガイド』および『Oracle Enterprise Manager 概要』を参照してください。

スタンドアロン OC4J 開発環境では、OC4J はデプロイ用に `admin.jar` ツールをサポートしています。このツールは、ツールに指定した設定値に基づいて、`server.xml`、`http-web-site.xml` および他の構成ファイルを変更します。構成ファイルは、手動で変更することもできます（通常はお勧めしません）。Oracle Application Server で Enterprise Manager を使用せずに構成ファイルを変更した場合は、`updateConfig` コマンドを使用して `dcmctl` ツールを実行し、Oracle Application Server Distributed Configuration Management (DCM) に変更内容を通知する必要があります（OC4J が Oracle Application Server とは別に実行されるスタンドアロン OC4J モードでは、この処理は適用されません）。

`dcmctl` ツールの詳細は、『Oracle Application Server 10g 管理者ガイド』を参照してください。

EAR ファイルには、次のファイルが含まれます。

- 標準の `application.xml` 構成ファイル。/`META-INF` にあります。
- (オプション) `orion-application.xml` 構成ファイル。/`META-INF` にあります。
- 標準の WAR (Web Archive) ファイル。

WAR ファイルには、次のファイルが含まれます。

- 標準の `web.xml` 構成ファイル。/`WEB-INF` にあります。  
特定のアプリケーションの `web.xml` ファイルでは、各構成パラメータに関するグローバル設定値、または JSP サブレット (デフォルトでは `oracle.jsp.runtime.v2.JspServlet`) の定義に関するグローバル設定値をオーバーライドできます。各アプリケーションでは、JSP サブレットの独自のインスタンスを使用します。
- (オプション) `orion-web.xml` 構成ファイル。/`WEB-INF` にあります。
- アプリケーション (サブレットや JavaBeans など) の実行に必要なクラス。  
`WEB-INF/classes` ディレクトリの下および `WEB-INF/lib` ディレクトリの JAR ファイルにあります。
- JSP ページと静的な HTML ファイル。

Oracle Application Server へのデプロイの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。スタンドアロン環境へのデプロイおよび `admin.jar` の詳細は、『Oracle Application Server Containers for J2EE Stand Alone User's Guide』を参照してください。また、OC4J の重要な構成ファイルのサマリーは、3-31 ページの「[主な OC4J 構成ファイル](#)」を参照してください。

## 標準の WAR デプロイ

JSP 仕様 (JSP1.1 以降) では、サブレット仕様 (サブレット 2.2 以降) に従って、JavaServer Pages を含む Web アプリケーションのパッケージングとデプロイがサポートされています。

典型的な実装では、WAR 機能を使用して JSP ページをデプロイし、JAR ユーティリティを使用して WAR ファイルを作成できます。JSP ページはソース・フォームで配布でき、必要なサポート・クラスと静的な HTML ファイルとともにデプロイされます。

サブレット仕様に従い、Web アプリケーションには、デプロイメント・ディスクリプタの `web.xml` ファイルが含まれています。このファイルには、JSP ページとアプリケーションのその他のコンポーネントに関する情報が含まれています。`web.xml` ファイルは、WAR ファイルに格納される必要があります。

また、サブレット仕様では、`web.xml` デプロイメント・ディスクリプタの XML DTD を定義し、サブレット・コンテナで Web アプリケーションをデプロイして、デプロイメント・ディスクリプタに準拠させる正確な方法を指定しています。

これらのロジック手法による WAR ファイルは、Web アプリケーションを標準のサーブレット環境に開発者が意図したとおりにデプロイする最も有効な方法です。

web.xml デプロイメント・ディスクリプタのデプロイ構成には、サーブレット・パスと、起動する JSP ページとサーブレットとの間のマッピングが含まれます。セッションのタイムアウト値、ファイル名拡張子の MIME タイプへのマッピング、エラー・コードの JSP エラー・ページへのマッピングなど、web.xml には多くの追加機能も指定できます。

標準の WAR デプロイの詳細は、Sun 社の Java サーブレット仕様を参照してください。

---

**注意：** OC4J では、通常、WAR ファイルを EAR ファイル内にデプロイします。WAR ファイルを直接デプロイすると、OC4J はその WAR ファイルを透過的に EAR ファイルにラップします。

---

## Oracle JDeveloper によるアプリケーションのデプロイ

Oracle JDeveloper は、単純なアーカイブ、J2EE アプリケーション (EAR ファイル)、J2EE EJB モジュール (EJB JAR ファイル)、J2EE Web モジュール (WAR ファイル)、J2EE クライアント・モジュール (クライアント JAR ファイル)、JSP 1.2 のタグ・ライブラリ (タグ・ライブラリ JAR ファイル)、ビジネス・コンポーネントの EJB Session Bean プロファイル、ビジネス・コンポーネントのアーカイブ・プロファイルなど、多くのタイプのデプロイ・プロファイルをサポートします。

Oracle JDeveloper を使用して Business Components for Java (BC4J) の Web アプリケーションを作成すると、J2EE Web モジュールのデプロイ・アーカイブが作成され、BC4J と Web アプリケーションの両方のファイルが含まれます。

JDeveloper のデプロイ・ウィザードを使用して、ビジネス・コンポーネントを J2EE Web モジュールとしてデプロイするために必要なすべてのコードを作成します。JSP クライアントは通常、J2EE Web モジュール構成内の BC4J アプリケーションにアクセスします。また、JSP クライアントは、データ・タグ、データ Web Bean、または UIX タグを使用して、ビジネス・コンポーネントにアクセスできます (BC4J と UIX タグ・ライブラリの概要については、『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』を参照してください)。

J2EE Web モジュールは WAR ファイルとしてパッケージングされ、1 つ以上の Web コンポーネント (サーブレットと JSP ページ) およびデプロイメント・ディスクリプタ・ファイルの web.xml が含まれます。

JDeveloper によって、Web コンポーネントと web.xml ファイルを含むデプロイ・プロファイルを作成し、1 つの標準の J2EE EAR ファイルにパッケージングしてデプロイできます。JDeveloper は、作成された EAR ファイルを取得し、それを Oracle Application Server の 1 つ以上のインスタンスにデプロイします。

JDeveloper については、JDeveloper のオンライン・ヘルプ、または次の OTN-J (Oracle Technology Network Japan) のサイトを参照してください。

<http://otn.oracle.co.jp/products/jdev/index.html>

## JSP の事前変換

JSP ページは一般的にオンデマンドで使用され、起動時に変換されます。変換の順序は、ユーザーには表示されません。他に、JSP ページを事前変換する方法もあります。この方法には少なくとも次の2つのメリットがあります。

- ページの最初の起動時に、オーバーヘッドを軽減します。
- ユーザーのかわりに、開発者またはデプロイヤが変換またはコンパイルのエラーを確認できます。

また、7-40 ページの「[バイナリ・ファイルのみのデプロイ](#)」で説明されているように、ページを事前変換するとバイナリ・ファイルのみをデプロイできます。

OC4J ユーザーは、Oracle の `ojspc` ユーティリティを使用し、事前変換用の個別ファイル、またはバッチ事前変換用のアーカイブ・ファイル (JAR、WAR、EAR または ZIP) のいずれかを指定できます。標準の `jsp_precompile` 機能もあります。この項には、次の項目が含まれます。

- [ojspc を使用したページ事前変換方法](#)
- [ojspc を使用したバッチ事前変換](#)
- [実行を伴わない標準の JSP の事前変換](#)

このユーティリティの詳細は、7-12 ページの「[ojspc 事前変換ユーティリティ](#)」も参照してください。

### ojspc を使用したページ事前変換方法

`ojspc` を使用して事前変換を実行する場合は、`-dir` オプションを使用して、生成されるバイナリ・ファイルを格納する適切な出力ベース・ディレクトリを設定します。ここでは、バッチ事前変換については考慮しません。

7-7 ページの「[JSP トランスレータの出力ファイルの格納場所](#)」の例について考えてみます。この例では、JSP ページは次に示すように、スタンドアロン OC4J のデフォルトの Web アプリケーション・ディレクトリ下の `examples/jsp` サブディレクトリに格納されています。

```
j2ee/home/default-web-app/examples/jsp/welcome.jsp
```

ユーザーは、次の URL を使用してこのページを起動します。

```
http://host:port/examples/jsp/welcome.jsp
```

(これは一般的な例です。コンテキスト・パスを使用する OC4J の構成には該当しません。)

このページのオンデマンド変換の場合、例に示すように、JSP トランスレータは、生成されるバイナリ・ファイルを格納する次のベース・ディレクトリをデフォルトで使用します。

```
j2ee/home/application-deployments/default/defaultWebApp/persistence/_pages/_examples/_jsp
```

事前変換を実行する場合は、カレント・ディレクトリをアプリケーション・ルート・ディレクトリに設定し、ojspc で `_pages` ディレクトリを出力ベース・ディレクトリに設定します。この設定によって、適切なパッケージ名とファイル階層が作成されます。引き続きコード例を示します（% は UNIX プロンプト、`OC4J_HOME` は `OC4J` がインストールされているディレクトリを表し、ojspc コマンドは 2 行目に折り返されています）。

```
% cd OC4J_HOME/j2ee/home/default-web-app
% ojspc examples/jsp/welcome.jsp
-dir OC4J_HOME/j2ee/home/application-deployments/default/defaultWebApp/persistence/_pages
```

（ここでは、適切な `OC4J_HOME` ディレクトリが指定されていることが前提とされています。）ojspc コマンドは、`examples/jsp/welcome.jsp` を変換し、ベース出力ディレクトリとして `_pages` ディレクトリを指定します。

前述の URL は `examples/jsp/welcome.jsp` のアプリケーション相対パスを指定しているため、JSP コンテナは、実行時に、`_pages` ディレクトリ下の `_examples/_jsp` サブディレクトリ内でバイナリ・ファイルを検索します。このサブディレクトリは、前述の例に示すように、ojspc が実行されると自動的に作成されます。

ソース・ファイルが事前変換後に変更されていない場合、または JSP の `main_mode` フラグが `justrun` に設定されている場合、JSP コンテナは実行時に事前変換されたバイナリ・ファイルを検出するため、変換を実行する必要はありません。

---

**注意：** `OC4J` の JSP 実装の詳細（出力ディレクトリ名でのアンダースコア（`_`）の使用など）は、リリースごとに変更される場合があります。このマニュアルは、特に `Oracle Application Server 10g (9.0.4)` に適用されます。

---

### ojspc を使用したバッチ事前変換

`OC4J 9.0.3` 以上の実装には、アーカイブ・ファイル（JAR、WAR、EAR または ZIP ファイル）内の JSP ファイルをバッチ事前変換するための `ojspc` 機能があります。`ojspc` コマンドラインでアーカイブ・ファイルを指定すると、デフォルトで、コンテンツ内のすべての `.jsp`、`.sqljsp`、`.sqlj` および `.java` ファイルが事前変換およびコンパイルされます。また、必要に応じて、このアーカイブ・ファイルは、変換によって出力される `.class` ファイルと Java リソース・ファイル（生成された `.java` ファイルと `.sqlj` ファイルを除く）を含めるために更新されます。その後、結果のアーカイブ・ファイルをデプロイします。

この基本機能の他に、主な `ojspc` のオプションを次のように使用できます。

- `-batchMask` オプションを使用すると、事前変換およびコンパイル用のファイル名拡張子を指定できます。指定した拡張子は、デフォルト（`*.jsp`、`*.sqljsp`、`*.sqlj` および `*.java`）のかわりに使用されます。



- `-output` オプションを使用すると、新しいアーカイブ・ファイル名を指定できます。デフォルトでは、`ojspc` は元のアーカイブ・ファイルを更新して、出力された `.class` ファイルとリソース・ファイルを追加します（場合によっては、`-deleteSource` オプションに基づいて、処理されたソース・ファイルを削除します）。元のアーカイブ・ファイルが変更されていないことを確認する場合は、`-output` オプションを有効にします。この場合、元のアーカイブ・ファイルのコンテンツがすべて、指定したアーカイブ・ファイルにコピーされ、元のファイルではなく、指定したファイルが更新されます。元のアーカイブ・ファイルは変更されません。デプロイには、元のファイルではなく新しいファイルを使用します。
- 出力されるアーカイブ・ファイルに処理されたソース・ファイルを含めない場合は、`-deleteSource` オプションを使用します。`-output` オプションも使用しない場合、処理が終了すると処理されたソース・ファイルがすべて元のアーカイブ・ファイルから削除されます。`-batchMask` オプションを使用しない場合、これは、すべての `*.jsp`、`*.sqljsp`、`*.sqlj` および `*.java` ファイルを構成します。それ以外の場合、これは、`-batchMask` 設定に指定されたすべてのファイルを構成します。

## 実行を伴わない標準の JSP の事前変換

オンデマンド変換でページを通常の方法で起動するとき、実行を伴わない JSP の事前変換を指定することもできます。次の手順で指定します。

1. JSP の `precompile_check` 構成パラメータを有効にします（3-14 ページの「[JSP 構成パラメータ](#)」を参照）。
2. 標準の `jsp_precompile` リクエスト・パラメータは、ブラウザから JSP ページを起動するときに有効にします。

次に、`jsp_precompile` の使用例を示します。

```
http://host:port/foo.jsp?jsp_precompile=true
```

または

```
http://host:port/foo.jsp?jsp_precompile
```

(「`=true`」はオプションです。)

この操作モードの詳細は、Sun 社の JSP 仕様を参照してください。

## バイナリ・ファイルのみのデプロイ

所有権またはセキュリティ上の理由から JSP ソースの公開を避けるには、ページを事前に変換し、変換およびコンパイルされたバイナリ・ファイルのみデプロイします。事前に変換されたページは、オンデマンド変換での前回の実行から、または `ojspc` を使用して、標準の J2EE 環境にデプロイできます。次の 2 つの手順が必要です。

1. バイナリ・ファイルは、適切にアーカイブおよびデプロイする必要があります。
2. ターゲット環境で、JSP ソースが使用できない場合は、ページを実行するために JSP コンテナが正しく構成されている必要があります。

### バイナリ・ファイルのアーカイブとデプロイ

バイナリ・ファイルは、適切な階層に作成してアーカイブする必要があります。

- `ojspc` による事前変換の場合は、最初に、カレント・ディレクトリをアプリケーション・ルート・ディレクトリに設定する必要があります。`ojspc` の実行後、アーカイブ用のベース・ディレクトリとして `ojspc` の出力ディレクトリを使用して、出力ファイルをアーカイブします。このユーティリティの一般情報は、7-12 ページの「[ojspc 事前変換ユーティリティ](#)」を参照してください。
- オンデマンド変換の環境で、前回の実行時に作成されたバイナリ・ファイルをアーカイブする場合は、出力ディレクトリ構造（通常は、`_pages` ディレクトリ下）をアーカイブします。

ターゲット環境では、アーカイブ JAR ファイルを `/WEB-INF/lib` ディレクトリに格納します。または、適切なディレクトリ（通常は、`_pages` ディレクトリ）下にあるアーカイブ済ディレクトリ構造をリストアします。

### バイナリ・ファイルのみを実行するための OC4J JSP コンテナの構成

バイナリ・ファイルを OC4J 環境にデプロイした場合は、JSP 構成パラメータ `main_mode` の値を `justrun` または `reload` に設定し、元のソースなしで JSP ページを実行します。

この設定を行わない場合、JSP トランスレータは、常に JSP ソース・ファイルを検索して、ページ実装の `.class` ファイルより後に変更されたかどうかを確認し、ソース・ファイルが見つからない場合は「ファイルが見つかりません」というエラー・メッセージを表示して終了します。

`main_mode` が適切に設定されている場合、ユーザーは、ソース・ファイルが存在していた場合に使用される URL を使用して、ページを起動できます。

OC4J 環境での構成パラメータの設定方法は、3-27 ページの「[OC4J での JSP 構成パラメータの設定](#)」を参照してください。

---

## JSP タグ・ライブラリ

この章では、カスタム・タグ・ライブラリ、およびベンダーが独自のライブラリの提供に使用できる基本的なフレームワークについて説明します。また、Oracle の拡張機能、および標準の実行時タグとベンダー固有のコンパイル時タグとの比較についても説明します。この章には、次の各項が含まれます。

- タグ・ライブラリのフレームワークの概要
- タグ・ライブラリ・ディスクリプタ
- タグ・ライブラリと TLD の設定およびアクセス
- タグ・ハンドラ
- OC4J の JSP タグ・ハンドラの機能
- スクリプト変数、宣言およびタグ補足情報クラス
- 妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス
- タグ・ライブラリのイベント・リスナー
- カスタム・タグの完全な例
- コンパイル時のタグ

この章では、標準タグ・ライブラリの機能の概要について説明します。詳細は、Sun 社の JSP 仕様を参照してください。OC4J が提供するタグ・ライブラリについては、『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』を参照してください。

カスタム・タグの多くの構文は、XML 規則に従っています。XML に関する一般情報は、次の Web サイトで仕様を参照してください。

<http://www.w3.org/XML/>

## タグ・ライブラリのフレームワークの概要

JavaServer Pages テクノロジーによって、ベンダーはカスタムの JSP タグ・ライブラリを作成できます。タグ・ライブラリは、カスタム・アクションのコレクションを定義します。タグは、開発者が JSP ページを手動でコーディングするときに直接使用されるか、Java 開発ツールで自動的に使用されます。

この項では、JSP タグ・ライブラリのフレームワークの概要を説明し、JSP 仕様 1.2 で追加されたタグ・ライブラリの機能のサマリーを示します。

タグ・ライブラリと標準の JavaServer Pages タグ・ライブラリのフレームワークに関するさらに詳しい情報は、次のリソースを参照してください。

- Sun 社の JSP 仕様
- 次の Web サイトにある、`javax.servlet.jsp.tagext` パッケージに関する Sun 社の Javadoc

[http://java.sun.com/j2ee/sdk\\_1.3/techdocs/api/javax/servlet/jsp/tagext/package-summary.html](http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/jsp/tagext/package-summary.html)

## カスタム・タグ・ライブラリの実装の概要

カスタム・タグ・ライブラリは、次の基本的なフォームの `taglib` ディレクティブを使用して、JSP ページにアクセスできます。

```
<%@ taglib uri="URI" prefix="prefix" %>
```

タグ・ライブラリの実装と使用方法では、次の事項に注意してください。

- ライブラリのタグは、タグ・ライブラリ・ディスクリプタ (TLD) に定義されています。詳細は、8-7 ページの「[タグ・ライブラリ・ディスクリプタ](#)」で説明します。
- `taglib` ディレクティブ内の URI は、TLD へのポインタです。詳細は、8-13 ページの「[概要: taglib ディレクティブによるタグ・ライブラリの指定](#)」で説明します。URI のショートカットを使用できます。詳細は、8-18 ページの「[web.xml のタグ・ライブラリへの使用](#)」で説明します。
- `taglib` ディレクティブの接頭辞は、JSP ページでライブラリからタグを選択するときに使用する文字列です。

次のように、`taglib` ディレクティブに接頭辞 `oracust` が指定されていると仮定します。

```
<%@ taglib uri="URI" prefix="oracust" %>
```

さらに、ライブラリには `mytag` というタグがあると仮定します。この `mytag` は、次のように使用できます。

```
<oracust:mytag attr1="...", attr2="..." />
```

接頭辞 `oracust` を使用することで、`mytag` が TLD に定義されており、そのファイルは、前述の `taglib` ディレクティブに指定されている URI で検出できることが、JSP トランスレータに通知されます。

- TLD 内のタグのエントリは、属性 (`mytag` では使用) とその名前をタグで使用するかどうかなど、タグの使用方法に関する仕様を提供します。
- タグのセマンティクス (タグを使用した結果、発生する操作) は、タグ・ハンドラ・クラスに定義されています。詳細は、8-26 ページの「[タグ・ハンドラ](#)」を参照してください。タグごとに独自のタグ・ハンドラ・クラスがあり、そのクラス名は TLD に指定されています。
- タグ属性は、標準的な Java 型またはオブジェクト型 (一般的な `java.lang.Object` またはユーザー定義型のいずれか) になります。

通常、標準的な Java 型の属性を文字列値として設定します。適切な変換が自動的に行われます。

また、文字列値を指定して Object 型の属性を設定できます。文字列は Object インスタンスに変換され、タグ・ハンドラ・インスタンス内の対応する setter メソッドに渡されます。この機能は JSP 仕様に準拠しています。

ユーザー定義型の属性は、その型のインスタンスを戻すリクエスト時の式を使用して設定する必要があります。

- TLD は、タグにボディが使用されているどうかを示します。

次の例では、ボディを持たないタグが使用されています。

```
<oracust:mytag attr1="...", attr2="..." />
```

これに対して、次の例では、ボディを持つタグが使用されています。

```
<oracust:mytag attr1="...", attr2="..." >
    ...body...
</oracust:mytag>
```

- カスタム・タグ操作によって、タグ自体またはスクリプトレットなどの JSP スクリプト要素で使用できる 1 つ以上のサーバー・サイド・オブジェクトを作成できます。このオブジェクトは、スクリプト変数と呼ばれます。

スクリプト変数は、TLD 内の `<variable>` 要素またはタグ補足情報クラスを使用して宣言できます。詳細は、8-41 ページの「[スクリプト変数、宣言およびタグ補足情報クラス](#)」を参照してください。

タグによって、次の例に示すように、`myobj` オブジェクトを作成する構文を持つスクリプト変数を作成して使用できます。

```
<oracust:mytag id="myobj" attr1="...", attr2="..." />
```

- TLD では、必要に応じて、タグ・ライブラリで使用するタグ・ライブラリ・バリデータ・クラスを宣言できます。このクラスには、タグ・ライブラリを使用する JSP ページの妥当性を、指定の制約に従ってチェックするためのロジックがあります。8-46 ページの「[妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス](#)」を参照。
- TLD では、必要に応じて、タグ・ライブラリで使用する 1 つ以上のイベント・リスナーを宣言できます。この機能は、アプリケーションの `web.xml` ファイルでリスナーを宣言する便利な代替方法です。8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照。
- ネストされたタグの処理または状態管理が必要な場合、ネストされたタグのタグ・ハンドラは、外部タグのタグ・ハンドラにアクセスできます。8-37 ページの「[外部タグ・ハンドラ・インスタンスへのアクセス](#)」を参照。

この章の後の項では、これらの項目について詳しく説明します。

---

---

**注意：** OC4J JSP コンテナでは、JServ 環境におけるタグ・ライブラリ機能（タグ・ライブラリのイベント・リスナー以外）もサポートしていません。

---

---

## JSP 仕様 1.1 と 1.2 の間のタグ・ライブラリ変更の概要

JSP 仕様 1.2 では、タグ・ライブラリにおける次の部分のサポートを改善する機能が追加されました。

- タグ・ライブラリ・ディスクリプタ機能  
機能の概要は、次項の「[JSP 仕様 1.1 と 1.2 の間の TLD 変更の概要](#)」で説明します。TLD 機能の詳細は、8-7 ページの「[タグ・ライブラリ・ディスクリプタ](#)」を参照してください。
- 単一の JAR ファイルにおける複数のタグ・ライブラリとその TLD のサポート  
JSP 仕様 1.1 では、単一の JAR ファイルに複数の TLD をパッケージングすることはできませんが、JSP 仕様 1.2 ではサポートされています。8-26 ページの「[タグ・ハンドラ](#)」を参照。
- タグ・ハンドラ機能  
機能の概要は、8-6 ページの「[JSP 仕様 1.1 と 1.2 の間のタグ・ハンドラ変更の概要](#)」を参照してください。タグ・ハンドラ機能の詳細は、8-26 ページの「[タグ・ハンドラ](#)」で説明します。
- タグ・ライブラリ・バリデータ  
この機能は、JSP 仕様 1.2 で追加されました。8-46 ページの「[妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス](#)」を参照。

- タグ・ライブラリのイベント・リスナー  
この機能も、JSP 仕様 1.2 で追加されました。8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照。
- Object 型のタグ属性のサポート  
JSP 仕様 1.2 では、`java.lang.Object` 型のタグ属性のサポートが追加されました。OC4J JSP コンテナではこの機能をサポートします。詳細は、前項の「[カスタム・タグ・ライブラリの実装の概要](#)」を参照してください。

---

**重要：** Oracle Application Server 10g (9.0.4) では、デフォルトで、OC4J JSP コンテナは JSP 1.2 ではなく JSP 1.1 のタグ構文を使用するように設定されています。次の各項で説明する JSP 1.2 の機能を使用するには、8-7 ページの「[TLD の妥当性チェックおよび機能の概要](#)」で説明するように、JSP 1.2 の TLD DTD を指定します。

---

## JSP 仕様 1.1 と 1.2 の間の TLD 変更の概要

次に、JSP 仕様 1.2 で導入された TLD 構文および機能の概要を説明します。これらの変更は、Oracle9iAS リリース 2 (9.0.3) 以降に使用可能になりました。これらの機能については、8-7 ページの「[タグ・ライブラリ・ディスクリプタ](#)」を参照してください。

- `<validator>` 要素とそのサブ要素。タグ・ライブラリのタグ・ライブラリ・バリデータ・クラスを宣言できます。
- `<listener>` 要素とそのサブ要素。タグ・ライブラリのイベント・リスナーを宣言できます。
- `<tag>` 要素の `<variable>` サブ要素とそのサブ要素。TLD を使用してスクリプト変数を直接宣言できます。
- `<tag>` 要素の `<attribute>` サブ要素内の `<type>` サブ要素。属性のデータ型を記述します。
- `<display-name>` 要素、`<large-icon>` 要素、`<small-icon>` 要素および `<tag>` 要素内の同じ名前のサブ要素。オーサリング・ツールで使用します。
- JSP 仕様 1.1 から名前が変更された要素は、次のとおりです。
  - `<info>` 要素、および `<tag>` 要素内の同じ名前のサブ要素は、`<description>` に名前が変更されました。
  - `<tlibversion>` 要素は、`<tlib-version>` に変更されました。
  - `<jspversion>` 要素は、`<jsp-version>` に変更されました。
  - `<shortname>` 要素は、`<short-name>` に変更されました。
  - `<tag>` 要素内の `<tagclass>`、`<teiclass>` および `<bodycontent>` サブ要素は、`<tag-class>`、`<tei-class>` および `<body-content>` に変更されました。

---

---

**注意：**

- OC4J JSP コンテナでは、TLD の XML 妥当性チェックと web.xml ファイルの妥当性チェックを個別に実行できます。デフォルトでは、TLD の妥当性チェックが有効で、web.xml の妥当性チェックは無効になっています (no\_tld\_xml\_validate パラメータおよび xml\_validate パラメータについては、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください)。Oracle9iAS リリース 2 (9.0.2) 以前では、TLD および web.xml はいずれも xml\_validate パラメータを使用して妥当性チェックが実行されていました。デフォルトでは、妥当性チェックは無効でした。
- OC4J には、サンプルの XSL テンプレートが用意されています。このテンプレートを標準的な XSLT プログラム (oraxsl など) で使用すると、JSP 1.1 準拠の TLD を JSP 1.2 準拠の TLD に変換できます。このテンプレートは、OC4J デモ・ディレクトリ構造の misc ディレクトリに格納されています。次の場所からデモをダウンロードできます。

[http://otn.oracle.co.jp/sample\\_code/index.html](http://otn.oracle.co.jp/sample_code/index.html)

---

---

## JSP 仕様 1.1 と 1.2 の間のタグ・ハンドラ変更の概要

JSP 仕様 1.1 には、タグ・ハンドラで実装できるインタフェースとして、ボディを持たないタグに使用する Tag、およびボディを持つタグに使用する BodyTag の 2 つが記載されています。JSP 仕様 1.2 では、IterationTag インタフェースが追加されました。このインタフェースは、タグ・ボディの反復が必要なタグで、ボディ・コンテンツ・オブジェクトを通じてタグ・ボディ・コンテンツにアクセスする必要はないタグに使用します。

IterationTag は、Tag を拡張し、BodyTag によって拡張されます。

JSP 仕様 1.2 では、int 定数の EVAL\_BODY\_TAG (処理対象のタグ・ボディがあることを示します) が使用できなくなり、EVAL\_BODY\_AGAIN および EVAL\_BODY\_BUFFERED に置換されています。EVAL\_BODY\_AGAIN は、タグ・ボディを反復するタグで使用され、反復の継続を指定します。EVAL\_BODY\_BUFFERED は、ボディ・コンテンツへのアクセスが必要なタグで使用され、BodyContent オブジェクトの作成を指示します。

また、JSP 仕様 1.2 では、TryCatchFinally インタフェースが追加されました。このインタフェースをタグ・ハンドラで実装すると、例外の発生時におけるデータ整合性とリソース管理が改善されます。

JSP 1.2 における変更は、Oracle9iAS リリース 2 (9.0.3) 以降に有効になりました。これらの新機能については、8-26 ページの「[タグ・ハンドラ](#)」を参照してください。



## タグ・ライブラリ・ディスクリプタ

タグ・ライブラリ・ディスクリプタ (TLD) は、タグ・ライブラリとその個々のタグを定義する XML スタイルの文書です。TLD のファイル名には、拡張子 `.tld` が付きます。

JSP コンテナは、TLD を使用して、ライブラリのタグを検出したときに実行する操作を確認します。JSP ページの `taglib` ディレクティブによって、TLD の格納場所が JSP コンテナに通知されます (8-13 ページの「[概要: taglib ディレクティブによるタグ・ライブラリの指定](#)」を参照)。

次の各項では、TLD の構文と使用方法の概要、および一般情報について説明します。関連項目の詳細は、該当する項を参照してください。

- [TLD の妥当性チェックおよび機能の概要](#)
- [tag 要素の使用](#)
- [その他の主要要素とそのサブ要素: validator および listener](#)

詳細は、Sun 社の JSP 仕様を参照してください。

サンプルの TLD は、8-56 ページの「[例: IterationTag インタフェースおよびタグ補足情報クラスの使用](#)」を参照してください。

---

**注意:** デフォルトでは、OC4J JSP コンテナは、TLD の XML 妥当性チェックを実行します。これを無効にするには、JSP 構成パラメータ `no_tld_xml_validate` を `true` に設定します。詳細は、3-18 ページの「[JSP 構成パラメータの説明](#)」を参照してください。事前変換には、`ojspc -noTldXmlValidate` オプションを使用します (7-20 ページの「[ojspc のオプションの説明](#)」を参照)。

---

## TLD の妥当性チェックおよび機能の概要

OC4J JSP コンテナでは、TLD 妥当性チェックが無効になっていないかぎり、TLD の DOCTYPE 宣言を使用して、妥当性チェックを実行する TLD DTD のバージョンを判断します。Oracle Application Server 10g (9.0.4) のデフォルトでは、JSP コンテナは JSP 1.1 の TLD DTD であるとみなします。JSP 1.2 の TLD DTD を使用するには、システム ID (DTD の場所) として次を指定します。

```
http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd
```

次に例を示します。

```
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
```

TLD 妥当性チェックが有効な場合、XML パーサーは、JSP 1.2 の前述の DOCTYPE 宣言を使用して適切な DTD を参照できる必要があります。JSP の `no_tld_xml_validate` パラメータがデフォルトの `false` に設定されている場合、または事前変換で `ojspc -noTldXmlValidation` フラグが使用されていない場合に、TLD 妥当性チェックが有効になります。

---

**注意：** JSP 仕様に従って、絶対 URL を使用してシステム ID を指定します。TLD で、絶対 URL を持つパブリックな外部の DOCTYPE 宣言を使用しない場合、Oracle Application Server 10g (9.0.4) のデフォルトには JSP 1.1 の TLD DTD が使用されるとみなします。

---

TLD は、各タグの定義とともに、タグ・ライブラリ全体の定義も提供します。タグごとに、タグの名前、その属性（ある場合）、スクリプト変数（ある場合）、およびタグのセマンティクスを処理するクラスの名前を定義します。8-9 ページの「[tag 要素の使用](#)」を参照。

ライブラリ全体の TLD 定義には、タグ・ライブラリ・バリデータ・クラスおよびイベント・リスナーを含めることができます。8-13 ページの「[その他の主な要素とそのサブ要素：validator および listener](#)」を参照。

TLD は、ライブラリ全体の次の定義も提供します。

---

**注意：** `<tag>` 要素、`<validator>` 要素、`<listener>` 要素および次に示す要素は、TLD の `<taglib>` ルート要素内のトップレベルのサブ要素です。

---

- 必須の `<tlib-version>` 要素は、タグ・ライブラリのバージョン番号（任意のバージョン番号）を指定します。
- 必須の `<jsp-version>` 要素は、タグ・ライブラリが準拠する JSP バージョン（1.2 など）を指定します。
- `<uri>` 要素は、タグ・ライブラリを一意に識別する文字列値を指定できます。この要素は、複数のタグ・ライブラリとその TLD が単一の JAR ファイルにパッケージングされている場合に、特に役立ちます。8-16 ページの「[単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス](#)」を参照。
- 必須の `<short-name>` 要素は、オーサリング・ツールで使用できるように、ライブラリのわかりやすいデフォルト名を指定します。また、短縮名は、`taglib` ディレクティブで使用するために、ライブラリのタグの接頭辞としても使用できます。
- 使用可能な追加の要素（通常はオーサリング・ツールで使用します）として、タグ・ライブラリの表示名を指定するための `<display-name>` 要素、および大きなアイコン、小さなアイコンまたはその両方を指定するためにファイル名（.jpg または .gif）に対

して指定する `<large-icon>` および `<small-icon>` 要素があります。アイコンのファイルの場所は、TLD に対して相対的になります。

- `<description>` 要素は、タグ・ライブラリの説明を提供します。

---

---

**注意：** JSP 1.2 の TLD DTD には、いくつかの記述要素が追加されています。`<taglib>` ルート要素の直下には `<description>` 要素があり、`<tag>`、`<variable>` および `<attribute>` 要素の下にも `<description>` サブ要素があります。`<tag>` 要素の下には、`<example>` サブ要素もあります。これらのサブ要素によって、タグ・ライブラリを使用する開発者に対して情報を提供できます。特に、TLD は、XSLT スタイルシートなどを使用して、記述要素内の資料から開発者向けの文書を提供するように処理できます。この情報は、ツール（Oracle JDeveloper など）のヘルプ・ウィンドウに表示できます。

---

---

## tag 要素の使用

タグ・ライブラリの各タグは、TLD の `<taglib>` ルート要素内の `<tag>` 要素に指定します。TLD には、1 つ以上の `<tag>` 要素が必要です。この項では、その使用方法とサブ要素について説明します。

### tag 要素のサブ要素

`<tag>` 要素のサブ要素によって、次のようにタグを定義します。

- 必須の `<name>` サブ要素は、タグの名前を指定します。
- 必須の `<tag-class>` サブ要素は、対応するタグ・ハンドラ・クラスの名前を指定します。タグ・ハンドラ・クラスについては、8-26 ページの「[タグ・ハンドラ](#)」を参照してください。
- `<body-content>` サブ要素は、タグ・ボディ（ある場合）の処理方法を示します。例と関連情報は、8-11 ページの「[tag 要素および body-content サブ要素の使用例](#)」を参照してください。
- 各 `<variable>` サブ要素（ある場合）およびその下のサブ要素は、スクリプト変数を定義します。スクリプト変数については、8-41 ページの「[スクリプト変数、宣言およびタグ補足情報クラス](#)」を参照してください。`<variable>` 要素は、状況が比較的複雑ではなく、スクリプト変数のロジックでタグ補足情報クラスを必要としない場合に使用します。変数名を指定するには、名前を直接指定する場合は `<name-given>` サブ要素を使用し、変数名を指定するタグ属性の名前を指定する場合は `<name-from-attribute>` サブ要素を使用します。その他に、変数のクラスを指定する `<variable-class>` サブ要素、変数のスコープを指定する `<scope>` サブ要素、および変数が新規に定義されるものであるかどうかを指定する `<declare>` サブ要素があります。詳細は、8-43 ページの「[TLD の variable 要素を使用した変数宣言](#)」を参照してください。`<variable>` 内には、オプションの `<description>` 要素もあります。

- 各 <tei-class> サブ要素（ある場合）は、スクリプト変数を定義するタグ補足情報クラスの名前を指定します。これは、<variable> 要素のみでは変数を宣言できない場合に使用します。詳細は、8-44 ページの「タグ補足情報クラスを使用した変数宣言」を参照してください。
- 各 <attribute> サブ要素（ある場合）およびその下のサブ要素は、カスタム・タグの使用時に指定可能なパラメータであるタグの属性に関する情報を提供します。<attribute> のサブ要素には、属性名を指定する <name> 要素、属性値の Java 型を示す <type> 要素（オプション）、属性が必須かどうかを指定する <required> 要素（デフォルト値は false）、および属性の値として実行時の式を受け入れるかどうかを指定する <rtexprvalue> 要素（デフォルト値は false）があります。例と関連情報は後述します。<attribute> 内には、オプションの <description> 要素もあります。

---

**注意：** Oracle Application Server 10g (9.0.4) では、OC4J JSP コンテナは <type> 要素を無視します。この要素は、TLD を調べる際に参照用としてのみ使用されます。次の点にも注意してください。

- <rtexprvalue> が false に指定されている場合、リテラル属性値に対する <type> の値（ある場合）は常に `java.lang.String` になります。
- <rtexprvalue> が true に指定されている場合は、このタグ属性に対応するタグ・ハンドラ・プロパティの型によって、指定する <type> の値（ある場合）が決まります。

- 
- タグ・ライブラリ全体の場合と同様に、各タグには、オーサリング・ツールで使用する <display-name>、<large-icon> および <small-icon> サブ要素を指定できます。
  - <description> サブ要素は、タグの説明を提供します。
  - <example> サブ要素は、タグの使用例を提供します。

---

**注意：**

- カスタム・タグ名には、XML 仕様に従う、NMTOKEN 型である必要があります。たとえば、名前の先頭を数値にすることはできません。
  - 属性名は XML 属性のネーミング規則に準拠し、タグ・ハンドラ・クラス内の setter メソッドは JavaBeans 仕様に準拠する必要があります。
-

## tag 要素および body-content サブ要素の使用例

次に、myaction というタグのサンプルの TLD エントリを示します。

```
<tag>
  <name>myaction</name>
  <tag-class>examples.MyactionTag</tag-class>
  <tei-class>examples.MyactionTagExtraInfo</tei-class>
  <body-content>JSP</body-content>
  <attribute>
    <name>attr1</name>
    <required>true</required>
  </attribute>
  <attribute>
    <name>attr2</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
```

このエントリでは、タグ・ハンドラ・クラスは MyactionTag で、タグ補足情報クラスは MyactionTagExtraInfo です。属性 attr1 は必須です。属性 attr2 はオプションで、実行時の式を値として指定できます。

<body-content> 要素は、タグ・ボディ（ある場合）の処理方法を示します。次の3つの選択肢があります。

- 値 empty は、タグでボディを使用しないことを示します。この場合、タグ・ボディが存在すると、OC4J JSP トランスレータは例外を戻します。
- 値 JSP（デフォルト）は、JSP ソース・コードの変換時に、タグ・ボディを処理する必要があることを示します。
- 値 tagdependent は、タグ・ボディを変換する必要がないことを示します。ボディ内のテキストは、テンプレート・データとして処理されます。

次に例を示します。

```
<foo:bar>
  <%=blah%>
</foo:bar>
```

bar タグに <body-content> の値 JSP が設定されている場合は、ボディが JSP トランスレータで処理されて、式が評価されます。<body-content> の値が tagdependent の場合、JSP トランスレータはボディを処理しません。この場合、「<」、「%」、「=」および「>」は特別な意味を持ちません。これらの文字は、ボディの残りの部分とともにリテラル文字として処理され、タグ・ハンドラに直接渡される JSP out オブジェクトに含まれます。

JSP XML 文書の場合は、考慮事項が他にもあります。XML 文書は XML パーサーによって解析されるため、値 `tagdependent` の実装は不適切です。この値は、基本的に JSP XML 文書では意味を持ちません。

その理由の 1 つは、XML には、CDATA トークンを使用してボディ・コンテンツをエスケープする便利な機能があることです。さらに、`tagdependent` を実装した場合と同様にコンテンツを直接渡すことは、実際に適切でない場合があります。SQL 問合せでタグを使用する例について考えてみます。従来の構文は次のとおりです。

```
<foosql:query ... >
  select ... where salary > 1000
</foosql:query>
```

この構文と、次の JSP XML 構文を比較してみます。

```
<foosql:query ... >
  <![CDATA[select ... where salary > 1000]]>
</foosql:query>
```

従来の構文では、`<body-content>` の値に `tagdependent` を使用すると、問合せ文が JSP out オブジェクトに直接渡されるため、正しい結果が得られます。

XML 構文では、CDATA トークン（またはエスケープ文字の「`&gt;`」）が必要です。これを使用しないと、文字「`>`」は XML パーサーに対して特別な意味を持つためです。

この例では、`tagdependent` を実装すると、ボディ全体が out オブジェクトに渡されます。

```
<![CDATA[select ... where salary > 1000]]>
```

しかし、実際に渡す必要がある情報は、次に示す SQL 問合せのみです。

```
select ... where salary > 1000
```

この問合せを実行するには、`<body-content>` の値に JSP を使用してボディを処理し、XML パーサーに対して CDATA トークンを使用します。これは、`tagdependent` を実装した場合よりも適切な動作です。

JSP XML 構文の詳細は、5-4 ページの「[JSP XML 文書の詳細](#)」を参照してください。

## その他の主要要素とそのサブ要素 : validator および listener

JSP 仕様 1.2 では、TLD の <validator> 要素および <listener> 要素が追加されました。

<validator> 要素とそのサブ要素は、タグ・ライブラリを使用する JSP ページの妥当性をチェックするタグ・ライブラリ・バリデータ (TLV) ・クラスに関する情報を指定します。

<validator> 要素には、<validator-class>、<description> および <init-param> の 3 つのサブ要素があります。<init-param> サブ要素の機能は、web.xml ファイルの <servlet> 要素内の <init-param> サブ要素と同じです。このサブ要素には、各パラメータを指定する <param-name> および <param-value> サブ要素があります。詳細は、8-46 ページの「[妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス](#)」を参照してください。

<listener> 要素とそのサブ要素の <listener-class> は、タグ・ライブラリで使用するイベント・リスナーを指定します (ライブラリで使用するリソース・プールの作成および破壊など)。詳細は、8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照してください。

## タグ・ライブラリと TLD の設定およびアクセス

次の各項では、タグ・ライブラリと TLD のパッケージング、格納およびアクセスについて説明します。

- [概要 : taglib ディレクティブによるタグ・ライブラリの指定](#)
- [物理的な場所によるタグ・ライブラリの指定](#)
- [単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス](#)
- [web.xml のタグ・ライブラリへの使用](#)
- [タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング](#)
- [例 : 単一の JAR ファイルに複数のタグ・ライブラリと TLD がある場合](#)

### 概要 : taglib ディレクティブによるタグ・ライブラリの指定

この項では、taglib ディレクティブの使用方法を要約し、JSP 仕様 1.1 の機能と、JSP 仕様 1.2 以降の機能を比較します。

次の基本的なフォームの taglib ディレクティブを使用して、カスタム・ライブラリを JSP ページにインポートします。

```
<%@ taglib uri="URI" prefix="prefix" %>
```

prefix 設定は、ライブラリのタグの使用時期を規定する文字列を指定します。たとえば、接頭辞が oracust に指定されているライブラリに mytag がある場合は、次のように mytag を使用します。

```
<oracust:mytag attr1="..." attr2="..." >
...
</oracust:mytag>
```

---

**注意：** 接頭辞は、XML 名前空間仕様のネーミング規則に準拠する必要があります。

---

JSP 仕様 1.1 では、次の例に示すように、uri 設定によって、ファイルの場所を直接またはショートカット URI を使用して示すことができると指示されていました。

- 任意のタグ・ライブラリを定義する TLD の WAR ファイル構造内における物理的な場所を示すことができます。
- 任意のタグ・ライブラリのコンポーネントおよび TLD が含まれる JAR ファイルの物理的な場所を示すことができます。JSP 仕様 1.1 では、JAR ファイル内に含まれるのは、1 つのタグ・ライブラリと 1 つの TLD のみです。

詳細は、8-15 ページの「[物理的な場所によるタグ・ライブラリの指定](#)」を参照してください。

JSP 仕様 1.2 以上では、同様に uri 設定によって、TLD の物理的な場所、または 1 つのタグ・ライブラリとその TLD が含まれる JAR ファイルの場所を示すことができますが、次のように使用することもできます。

- <uri> 要素値と一致する値を JAR ファイル内の TLD の 1 つに指定することで、単一の JAR ファイルにパッケージングされた複数のタグ・ライブラリの 1 つを指定できます。この場合、uri 設定は、物理的な場所へのポインタではなく、一意のキーとして使用されます。

JSP 1.1 と同様に、ショートカット URI も使用できます。

詳細は、8-16 ページの「[単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス](#)」を参照してください。ショートカット URI については、8-18 ページの「[web.xml のタグ・ライブラリへの使用](#)」を参照してください。



## 物理的な場所によるタグ・ライブラリの指定

JSP 仕様 1.1 で最初に定義されているように、JSP ページの `taglib` ディレクティブでは、特定のタグ・ライブラリを定義する TLD の名前、および WAR ファイル構造における物理的な場所を完全に指定できます。次に例を示します。

```
<%@ taglib uri="/WEB-INF/oracustomtags/tlds/mytld.tld" prefix="oracust" %>
```

アプリケーションに対して相対的な場所として指定します（この例に示すように、「/」で始まります）。アプリケーションに対して相対的な構文については、1-27 ページの「[JSP ページのリクエスト](#)」を参照してください。

TLD は、`/WEB-INF` ディレクトリまたはサブディレクトリ内に含まれている必要があることに注意してください。

かわりに、JSP 仕様 1.1 以降で定義されているように、`taglib` ディレクティブは、TLD ではなく JAR ファイルの名前、およびアプリケーション相対の物理的な場所を指定できます。JAR ファイルには、単一のタグ・ライブラリおよびそれを定義する TLD が含まれます。この場合、JSP 仕様 1.1 では、TLD は JAR ファイル内に次のように格納され、名前が付けられる必要がありました。

```
META-INF/taglib.tld
```

JSP 仕様 1.1 では、JAR ファイルは、`/WEB-INF/lib` ディレクトリ内に格納する必要もありました。

次に、タグ・ライブラリ JAR ファイルを指定する `taglib` ディレクティブの例を示します。

```
<%@ taglib uri="/WEB-INF/lib/mytaglib.jar" prefix="oracust" %>
```

次項の「[単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス](#)」も参照してください。JSP 仕様 1.2 で導入された機能について説明しています。

---

---

**注意：** この項で説明している例では、`taglib` ディレクティブは、`web.xml` ファイルの設定に従って、完全な URI の値に対応するショートカット URI を指定できます。8-18 ページの「[web.xml のタグ・ライブラリへの使用](#)」を参照。

---

---

## 単一の JAR ファイル内での複数のタグ・ライブラリのパッケージングおよびアクセス

前項の「物理的な場所によるタグ・ライブラリの指定」では、JSP 1.1 で `taglib` ディレクティブを使用して、TLD を物理的な場所によって指定したり、単一のタグ・ライブラリとその TLD を含む JAR ファイルを指定する方法を説明しました。

これに加えて、JSP 仕様 1.2 では、単一の JAR ファイル内に、複数のタグ・ライブラリおよびタグ・ライブラリを定義する TLD をパッケージングできるようになりました。JAR ファイル内では、TLD は `/META-INF` ディレクトリまたはサブディレクトリ内に格納されている必要があります。

単一の JAR ファイル内に 1 つの TLD がパッケージングされている場合は `/META-INF/taglib.tld` として格納できます (JSP 1.2 では必須要件ではありません) が、単一の JAR ファイル内に複数の TLD がある場合は、一意の名前またはサブディレクトリを使用する必要があります。たとえば、単一の JAR ファイル内に 3 つの TLD をパッケージングするには、次の 2 つの方法があります。

```
META-INF/abctags.tld
META-INF/deftags.tld
META-INF/ghitags.tld
```

または

```
META-INF/abc/taglib.tld
META-INF/def/taglib.tld
META-INF/ghi/taglib.tld
```

各 TLD には、`<taglib>` 要素内に `<uri>` 要素があります。この機能は、次のように使用します。

- `<uri>` 要素に指定する値は、対応するタグ・ライブラリを使用する JSP ページ内で、`taglib` ディレクティブの `uri` 設定と一致する値であることが必要です。
- 予期しない結果を回避するために、`<uri>` の各値は、サーバー上のすべての TLD 内にある `<uri>` の値の中で一意であることが必要です。

`<uri>` 要素には任意の値を設定できます。この値はキーとしてのみ使用され、物理的な場所を示しません。ただし、次の例に示すように、値は、表記規則に従って物理的な場所を示す場合と同じ形式で設定します。

```
<uri>http://www.mycompany.com/j2ee/jsp/tld/myproduct/mytags.tld</uri>
```

`<uri>` の値は、XML 名前空間の規則に従って設定する必要があります。

複数の TLD が含まれた JAR ファイルは、`/WEB-INF/lib` ディレクトリ内、または OC4J の予約済みのタグ・ライブラリの場所 (8-19 ページの「[タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング](#)」を参照) に格納する必要があります。JSP

コンテナは、変換時に、この2つの場所で JAR ファイルを検索し、各 JAR ファイルで TLD を検索し、各 TLD にアクセスして <uri> 要素を検索します。

---

---

**注意：**

- <uri> 要素と対応する taglib ディレクティブでは、ショートカット URI 設定を指定できます。これは、web.xml ファイルの設定に対応します。詳細は、8-18 ページの「[web.xml のタグ・ライブラリへの使用](#)」で説明します。
  - JSP 1.2 準拠のコンテナ（OC4J JSP コンテナなど）は、JSP 1.2 の TLD と同様に、JSP 1.1 の TLD についても複数の TLD のパッケージング機能をサポートしています。
- 
- 

**例：単一の JAR ファイルに複数のタグ・ライブラリがある場合の URI 設定** 次の TLD が含まれている JAR ファイルの myapptags.jar について考えてみます。

```
META-INF/mytaglib1.tld
META-INF/mytaglib2.tld
```

mytaglib1.tld で次のように指定するとします。

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>shorty</short-name>
  <uri>http://www.foo.com/jsp/mytaglib1</uri>
  <description>example TLD</description>
  <tag>
    <name>mytag1</name>
    ...
  </tag>
  ...
</taglib>
```

mytag1、または mytaglib1.tld で定義された別のタグを使用する場合、JSP ページの taglib ディレクティブは次のようになります。

```
<%@ taglib uri="http://www.foo.com/jsp/mytaglib1" prefix="myprefix1" %>
```

この例（単一の JAR ファイルに複数のタグ・ライブラリがある場合）では、URI の値はキーワードとしてのみ使用されています。任意の値を設定できます。

詳細な例は、8-23 ページの「[例：単一の JAR ファイルに複数のタグ・ライブラリと TLD がある場合](#)」を参照してください。

## web.xml のタグ・ライブラリへの使用

Sun 社の Java サーブレット仕様では、サーブレットの標準デプロイメント・ディスクリプタである web.xml ファイルについて説明しています。JSP ページでは、このファイルを使用して、JSP TLD の場所または URI 識別子を指定できます。

JSP タグ・ライブラリの場合、web.xml ファイルには、<taglib> 要素と次の 2 つのサブ要素を含めることができます。

- <taglib-uri>
- <taglib-location>

各 TLD、または単一のタグ・ライブラリとその TLD が含まれる JAR ファイルを使用する場合、<taglib-location> サブ要素は、TLD またはタグ・ライブラリ JAR ファイルの、アプリケーション相対の物理的な場所（「/」で始まります）を示します。関連情報は、8-15 ページの「[物理的な場所によるタグ・ライブラリの指定](#)」を参照してください。

複数のタグ・ライブラリとその TLD が含まれる JAR ファイルを使用する場合、<taglib-location> サブ要素は、タグ・ライブラリの一意の識別子を示します。この場合、<taglib-location> 値は実際には場所ではなくキーを示し、任意のタグ・ライブラリの TLD にある <uri> 値に対応します。関連情報は、8-16 ページの「[単一の JAR ファイル内の複数のタグ・ライブラリのパッケージングおよびアクセス](#)」を参照してください。

<taglib-uri> サブ要素は、JSP ページの taglib ディレクティブで使用するショートカット URI を示します。この URI は、付随する <taglib-location> サブ要素に指定されている物理的な場所または URI 識別子にマッピングされます。

次に、TLD のサンプルの web.xml エントリを示します。

```
<taglib>
  <taglib-uri>/oracustomtags</taglib-uri>
  <taglib-location>/WEB-INF/oracustomtags/tlds/mytld.tld</taglib-location>
</taglib>
```

このエントリによって、/oracustomtags は、JSP ページの taglib ディレクティブの /WEB-INF/oracustomtags/tlds/mytld.tld と同等になります。

この例の場合、JSP ページの次のディレクティブによって、JSP コンテナは、web.xml ファイルの /oracustomtags URI を検出し、TLD (mytld.tld) の名前と場所を検出します。

```
<%@ taglib uri="/oracustomtags" prefix="oracust" %>
```

この文によって、このカスタム・タグ・ライブラリの任意のタグを JSP ページで使用できます。

web.xml デプロイメント・ディスクリプタの詳細は、Sun 社の Java サーブレット仕様、および Sun 社の JSP 仕様を参照してください。

---

---

**重要：** 一般的に、TLD が JSP の予約済タグ・ライブラリの場所に格納され、`<listener>` 要素が設定されている場合は、`web.xml` の `<taglib>` 要素を使用する必要があります。これは、リスナーをアクティブ化するために TLD を検出してアクセスする唯一の方法です。ただし、永続的な TLD キャッシングを使用する場合には当てはまりません。8-19 ページの「[タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング](#)」および 8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照。

---

---

## タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング

JSP 仕様で規定されている標準的な JSP の予約済 URI 機能の拡張機能として、OC4J JSP コンテナは、予約済のタグ・ライブラリの場所と呼ばれる 1 つ以上のディレクトリの使用をサポートします。このディレクトリには、複数の Web アプリケーション間で共有するタグ・ライブラリ JAR ファイルを格納できます。

OC4J 9.0.4 以上の実装には、TLD の永続的なキャッシング機能もあり、予約済のタグ・ライブラリの場所における TLD のグローバル・キャッシュ、および TLD を使用するアプリケーションに対するアプリケーション・レベルでのキャッシュを使用できます。

TLD キャッシングの使用により、アプリケーションの起動時や JSP ページ変換時のパフォーマンス速度が向上します。ただし、次の状況では、通常は TLD キャッシングをオフにしてください。

- アプリケーションでタグ・ライブラリが使用されていない。

または

- JSP ページの事前変換が完了しており、タグ・ライブラリのイベント・リスナーに `<listener>` 要素を使用している TLD がない（8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照）。

次の各項では、追加情報について説明します。

- [TLD キャッシングと予約済のタグ・ライブラリの場所](#)
- [TLD キャッシュ機能とファイル](#)

## TLD キャッシングと予約済のタグ・ライブラリの場所

TLD キャッシングは、<orion-web-app> 要素の jsp-cache-tlds 属性を介して有効または無効にします。グローバル・レベルの場合は、global-web-application.xml ファイルのこの属性を使用し、アプリケーション・レベルの場合は、アプリケーションの orion-web.xml ファイルのこの属性を使用します。

デフォルトでは、TLD キャッシングは、global-web-application.xml のデフォルト設定 jsp-cache-tlds="true" を介して、グローバル・レベルで有効になっています。これは、各アプリケーションの orion-web.xml ファイルのデフォルト設定でもありますが、orion-web.xml の jsp-cache-tlds="false" の設定を使用して、特定のアプリケーションの TLD キャッシングを無効にできます。これにより、グローバル設定がオーバーライドされます。

global-web-application.xml の false 設定を使用して、グローバル・レベルで TLD キャッシングを無効にしてから、orion-web.xml の true 設定を使用して、特定のアプリケーションの TLD キャッシングをオプションで有効にすることもできます。

---

---

**注意：** デフォルトでは、orion-web.xml は、global-web-application.xml から jsp-cache-tlds の設定を継承します。

---

---

TLD キャッシングが有効になっている場合は、global-web-application.xml の <orion-web-app> 要素の jsp-taglib-locations 属性で、セミコロンをデリミタとして使用したディレクトリ・パスのリストを使用して、1 つ以上の予約済のタグ・ライブラリの場所を指定できます。この属性の追加情報は、3-28 ページの「[JSP の OC4J 構成パラメータ](#)」参照してください。

---

---

**重要：** orion-web.xml ではなく、global-web-application.xml のみで jsp-taglib-locations 属性を使用します。

---

---

キャッシングが無効になっている場合、TLD キャッシングの可用性より前に存在した機能を使用して、予約済のタグ・ライブラリの場所は、単一のディレクトリに制限されます。この場合、予約済の場所は、well\_known\_taglib\_loc JSP 構成パラメータによって決定されます。このパラメータの追加情報は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

Oracle Application Server 環境では、デフォルトの予約済の場所は、ORACLE\_HOME/j2ee/home/jsp/lib/taglib (ORACLE\_HOME が定義されていることが前提) です。

---

---

**重要:**

- 予約済の場所にあるファイルを選択するアプリケーションの場合、`jsp-taglib-locations` で指定されたディレクトリ、または `well_known_taglib_loc` で指定されたディレクトリを、構成ファイル・ディレクトリ（スタンドアロン OC4J のデフォルトでは `j2ee/home/config`）にある OC4J `application.xml` グローバル・ファイルの `<library>` 要素の `path` 属性設定に追加する必要があります。`application.xml` 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。
  - TLD が、予約済の場所とアプリケーションの `/WEB-INF` ディレクトリ下の両方にある場合、`/WEB-INF` のコピーが優先され、使用されます。
  - URI 値が同じ TLD が、`/WEB-INF` ディレクトリと、`/WEB-INF/lib` ディレクトリの JAR ファイルにもある場合、どちらが使用されるかは確定できません。このような状況は避けてください。
- 
- 

## TLD キャッシュ機能とファイル

TLD キャッシングを使用するアプリケーションの場合、グローバル・レベルまたはアプリケーション・レベルで有効になっているかどうかに関係なく、2つのレベルのキャッシングがあり、各レベルのキャッシングには2つの機能があります。

キャッシング・レベル:

- TLD にはグローバル・キャッシュがあります。TLD は、予約済のタグ・ライブラリの場所の JAR ファイルにあります。
- アプリケーションの `/WEB-INF` ディレクトリには、TLD のアプリケーション・レベルのキャッシュがあります。

アプリケーション・レベルでは、タグ・ライブラリ JAR ファイルには TLD が含まれており、`/WEB-INF/lib` ディレクトリ内に格納する必要があります。個別の TLD は、`/WEB-INF` 内のディレクトリに直接、または任意のサブディレクトリに格納できますが、できれば、`/WEB-INF/lib` または `/WEB-INF/classes` には格納しないでください。

各レベルのキャッシング機能:

- 関係する場所のリソース情報を含むファイル。グローバル・キャッシュの場合は予約済の場所、アプリケーション・レベル・キャッシュの場合は `/WEB-INF` または `/WEB-INF/lib` です。この機能によって、JAR ファイルを複数回スキャンする必要はありません。JAR ファイルには、2つのタイプのエントリがあります。

- リソースごとのタイムスタンプを含む全リソース（タグ・ライブラリ JAR ファイル）のリスト。これにより、リソースの変更が検出されます。また、各リソースに TLD が含まれているかどうか、true または false でわかります。
  - TLD のリスト。リストの各エントリは、TLD 名、TLD URI 値（ある場合）およびライブラリ・リスナー（ある場合）で構成されます（8-50 ページの「[タグ・ライブラリのイベント・リスナー](#)」を参照）。
- 各 TLD のシリアライズ化された DOM 表現。この機能によって、TLD を複数回解析する必要はありません。

グローバル・キャッシュは、構成ディレクトリと並列の、`tldcache` というディレクトリに常に格納されます。`tldcache` ディレクトリには、次のものが含まれます。

- `_GlobalTldCache` ファイル。前述のように、予約済の場所に関するリソース情報が含まれています。
- 予約済の場所にある TLD の DOM 表現。予約済の場所の JAR ファイルに格納されている TLD ごとに、TLD の名前に基づいたファイル名とともに、JAR ファイルの名前から従って、サブディレクトリ内に DOM 表現が格納されます。たとえば、予約済の場所の `ojsputil.jar` に `email.tld` がある場合、DOM 表現は次のファイル（ディレクトリ `ojsputil_jar` のファイル名 `email`）に格納されます。

```
ORACLE_HOME/j2ee/home/jsp/lib/taglib/persistence/ojsputil_jar/email
```

これは、Oracle Application Server 環境で、`ORACLE_HOME` が定義されている場合です。スタンドアロン OC4J では、`j2ee` ディレクトリは、OC4J がインストールされた場所に対する相対パスになります。

アプリケーション・レベルのキャッシュは、`global-web-application.xml` または `orion-web.xml` の `jsp-cache-directory` 設定で指定されたディレクトリに格納されます（`jsp-cache-directory` の詳細は、3-28 ページの「[JSP の OC4J 構成パラメータ](#)」を参照してください）。このディレクトリには、次のものが含まれます。

- `_TldCache` ファイル。前述のように、`/WEB-INF` ディレクトリの TLD に関するリソース情報が含まれています。`/WEB-INF/lib` の JAR ファイル内、または `/WEB-INF` や任意のサブディレクトリに個別に格納されますが、できれば、`/WEB-INF/lib` または `/WEB-INF/classes` は使用しないでください。
- `/WEB-INF` の TLD の DOM 表現。`/WEB-INF/lib` ディレクトリの JAR ファイルに格納されている TLD の場合、DOM 表現は、グローバル・キャッシュについて説明したのと同じスキーム・タイプで、`jsp-cache-directory` で指定されたディレクトリのサブディレクトリに格納されます。`/WEB-INF` の個別の TLD の場合、DOM 表現は、`jsp-cache-directory` の場所に直接格納されます。



---

---

**注意：**

- グローバル・レベルでの TLD の変更は、OC4J を再起動した後にのみ反映されます。
  - アプリケーション・レベルでの TLD の変更は、スタンドアロン OC4J 環境ではただちに反映されますが、Oracle Application Server 環境では、アプリケーションを再起動した後にのみ反映されます。
  - OC4J の冗長レベルを大きくすると、TLD キャッシュの構成、および重複している TLD URI に関する情報を確認できます。レベル 4 では一部の情報が提供され、レベル 5 では追加情報が提供されます。Oracle Enterprise Manager を使用して冗長レベルを設定できます。デフォルトのレベルは 3 です。
- 
- 

**例：単一の JAR ファイルに複数のタグ・ライブラリと TLD がある場合**

この項では、タグ・ライブラリのパッケージングの例を示します。ここでは、複数のタグ・ライブラリが単一の JAR ファイルにパッケージングされています。JAR ファイルには、タグ・ハンドラ・クラス、タグ・ライブラリ・バリデータ・クラス、および複数のライブラリの TLD が含まれています。次に、JAR ファイルの内容と構造を示します。

```
examples/BasicTagParent.class
examples/ExampleLoopTag.class
examples/BasicTagChild.class
examples/BasicTagTLV.class
examples/TagElemFilter.class
examples/XMLViewTagTLV.class
examples/TagFilter.class
examples/XMLViewTag.class
META-INF/xmlview.tld
META-INF/exampletag.tld
META-INF/basic.tld
META-INF/MANIFEST.MF
```

## 複数のライブラリに対する主要な TLD エントリの例

この項では、TLD の <uri> 要素を示します。

basic.tld ファイルには次の内容が含まれます。

```
<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>basic</short-name>
  <uri>http://xmlns.oracle.com/j2ee/jsp/tld/demos/basic.tld</uri>

  ...

</taglib>
```

exampletag.tld ファイルには次の内容が含まれます。

```
<taglib xmlns="http://java.sun.com/JSP/TagLibraryDescriptor">

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>example</short-name>
  <uri>http://xmlns.oracle.com/j2ee/jsp/tld/demos/exampletag.tld</uri>
  ...
</taglib>
```

xmlview.tld ファイルには次の内容が含まれます。

```
<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>demo</short-name>
  <uri>http://xmlns.oracle.com/j2ee/jsp/tld/demos/xmlview.tld</uri>

  ...

</taglib>
```

## 複数のライブラリに対する web.xml ファイルの主要なエントリの例

この項では、web.xml デプロイメント・ディスクリプタの <taglib> 要素を示します。この要素は、URI のすべての値（前項で説明した TLD の <uri> 要素）を、ライブラリにアクセスする JSP ページで使用するショートカット URI 値にマッピングします。

```
...
<taglib>
  <taglib-uri>/oraloop</taglib-uri>
  <taglib-location>http://xmlns.oracle.com/j2ee/jsp/tld/demos/exampletag.tld
</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/orabasic</taglib-uri>
  <taglib-location>http://xmlns.oracle.com/j2ee/jsp/tld/demos/basic.tld
</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/oraxmlview</taglib-uri>
  <taglib-location>http://xmlns.oracle.com/j2ee/jsp/tld/demos/xmlview.tld
</taglib-location>
</taglib>
...
```

## 複数のライブラリに対する JSP ページの taglib ディレクティブの例

この項では、適切な taglib ディレクティブを示します。このディレクティブは、前項でリストされている web.xml 要素に定義されたショートカット URI 値を参照します。

ページ basic1.jsp には、次のディレクティブが含まれます。

```
<%@ taglib prefix="basic" uri="/orabasic" %>
```

ページ exampletag.jsp には、次のディレクティブが含まれます。

```
<%@ taglib prefix="example" uri="/oraloop" %>
```

ページ xmlview.jsp には、次のディレクティブが含まれます。

```
<%@ taglib prefix="demo" uri="/oraxmlview" %>
```

## タグ・ハンドラ

次の各項では、カスタム・タグを使用することにより発生する操作のセマンティクスを定義するタグ・ハンドラについて説明します。

- [タグ・ハンドラの概要](#)
- [属性の処理および文字列の変換](#)
- [カスタム・タグの処理（タグ・ボディを使用する場合と使用しない場合）](#)
- [ボディ処理用の整定数の要約](#)
- [反復のない単純タグ・ハンドラ](#)
- [反復がある単純タグ・ハンドラ](#)
- [ボディ・コンテンツにアクセスするタグ・ハンドラ](#)
- [TryCatchFinally インタフェース](#)
- [外部タグ・ハンドラ・インスタンスへのアクセス](#)

### タグ・ハンドラの概要

タグ・ハンドラは、標準の `javax.servlet.jsp.tagext.Tag` インタフェースを直接的または間接的に実装する Java クラスのインスタンスです。タグ・ボディがあるかどうか、およびボディの処理方法に応じて、タグ・ハンドラは `javax.servlet.jsp.tagext` パッケージの次のいずれかのインタフェースを実装します。

- **Tag:** このインタフェースは、すべてのタグを処理するための基本メソッドを定義しますが、タグ・ボディの処理は含まれません。
- **IterationTag:** このインタフェースは、Tag を拡張し、タグ・ボディを反復するために使用されます。
- **BodyTag:** このインタフェースは、IterationTag を拡張し、タグ・ボディ・コンテンツにアクセスするために使用されます。

タグ・ハンドラ・クラスは、これらのインタフェースのいずれかを直接実装したり、いずれかのインタフェースを実装するクラス（Sun 社が提供するサポート・クラスの 1 つなど）を拡張できます。

カスタム・タグごとに独自のハンドラ・クラスがあります。タグ・ハンドラ・クラス名は、表記規則によって決まります。たとえば、abc というタグは、AbcTag という名前になります。

タグ・ライブラリの TLD によって、ライブラリ内の各タグのタグ・ハンドラ・クラス名が指定されます。8-7 ページの「[タグ・ライブラリ・ディスクリプタ](#)」を参照。

タグ・ハンドラ・インスタンスは、通常、JSP ページ実装インスタンスによって引数が 0 (ゼロ) のコンストラクタを使用して作成され、リクエスト時に使用されるサーバー・サイド・オブジェクトです。タグ・ハンドラには、JSP コンテナで設定されたプロパティがあります。このプロパティには、カスタム・タグを使用する JSP ページのページ・コンテキスト・オブジェクト、およびこのタグを外部タグ内にネストして使用する場合は、タグ・ハンドラの親オブジェクトも含まれます。タグ・ハンドラは、必要に応じて、パラメータの受渡し、タグ・ボディの評価、および JSP ページ内の他のオブジェクト (他のタグ・ハンドラを含む) へのアクセスをサポートしています。

タグ・ハンドラ・クラスのサンプル・コードは、8-56 ページの「例: IterationTag インタフェースおよびタグ補足情報クラスの使用」を参照してください。

---

---

**注意:** JSP 仕様では、1 つの JSP ページ内で同じカスタム・タグを繰り返して使用する場合に、同じタグ・ハンドラ・インスタンスを使用するか、または別のインスタンスを使用するかは指示されていません。これは、JSP ベンダーが決定します。Oracle 実装については、8-38 ページの「OC4J の JSP タグ・ハンドラの機能」を参照してください。

---

---

## 属性の処理および文字列の変換

タグ・ハンドラ・クラスには、カスタム・タグの各属性について基礎となるプロパティがあります。このプロパティは、setter メソッドがある点などで、JavaBean のプロパティに類似しています。

タグ属性を設定するには、次の 2 つの方法があります。

- 属性がリクエスト時以外の属性の場合は、次のように、文字列リテラル値を使用して設定します。

```
nrtattr="string"
```

リクエスト時以外の属性の場合、基礎となるタグ・ハンドラ・プロパティが String 型でないと、JSP コンテナは文字列値を適切な型の値に変換しようとします。

タグ属性は Bean のプロパティに似ているため、その処理 (文字列値から適切な型への変換など) も Bean のプロパティの処理に似ています。1-21 ページの「文字列値から Bean プロパティへの変換」を参照。

- 属性がリクエスト時の属性の場合は、次のように、リクエスト時の式を使用して設定します。

```
rtattr="<%=expression%>"
```

リクエスト時の属性の場合、変換は行われません。リクエスト時の式は、属性およびそれに対応するタグ・ハンドラ・プロパティ (任意のプロパティ型) に割り当てることができます。この操作は、ユーザー定義型のタグ属性などに適用できます。

## カスタム・タグの処理（タグ・ボディを使用する場合と使用しない場合）

カスタム・タグは、標準の JSP タグと同様に、ボディを使用する場合と使用しない場合があります。カスタム・タグでは、ボディを使用する場合でも、タグ・ハンドラによるボディ・コンテンツへのアクセスを必要としない場合があります。

次の 4 つの場合があります。

### 1. ボディがない場合。

この場合、開始タグと終了タグは使用せず、単一のタグのみが使用されます。次に、一般的な例を示します。

```
<oracust:mytag attr1="...", attr2="..." />
```

このコードと次のコードは同等で、いずれも使用可能です。

```
<oracust:mytag attr1="...", attr2="..." ></oracust:abcdef>
```

この場合、タグ・ハンドラは、Tag インタフェースを実装する必要があります。

TLD でこのタグの `<body-content>` は、`empty` に設定する必要があります。

### 2. タグ・ハンドラによるボディ・コンテンツへのアクセスが不要で、1 回のみ実行されるボディを使用する場合。

この場合も、開始タグと終了タグの間に文のボディがありますが、タグ・ハンドラはボディを処理しません。ボディの文は、通常の JSP 処理に対してのみ渡されます。次に、一般的な例を示します。

```
<foo:if cond="<%= ... %>" >  
...body executed if cond is true, but body content not accessed by tag  
handler...  
</foo:if>
```

この場合、タグ・ハンドラは、Tag インタフェースを実装する必要があります。

TLD でこのタグの `<body-content>` は、ボディ・コンテンツを変換する場合は JSP (デフォルト)、テンプレート・データとして処理する場合は `tagdependent` に設定する必要があります。

### 3. タグ・ハンドラによるボディ・コンテンツへのアクセスが不要で、反復して実行されるボディを使用する場合。

タグ・ボディが反復して処理されることを除いて、2 番目の場合と同じです。

```
<foo:myiteratetag ... >  
...body executed multiple times, according to attribute or other settings, but  
body content not accessed by tag handler...  
</foo:myiteratetag>
```

この場合、タグ・ハンドラは、IterationTag インタフェースを実装する必要があります。

TLD でのこのタグの <body-content> は、ボディ・コンテンツを変換する場合は JSP (デフォルト)、テンプレート・データとして処理する場合は tagdependent に設定する必要があります。

4. タグ・ハンドラによる処理が必要なボディを使用する場合。

この場合も、開始タグと終了タグの間に文のボディがありますが、タグ・ハンドラはそのボディ・コンテンツにアクセスする必要があります。

```
<oracust:mybodytag attr1="...", attr2="..." >  
...body accessed and processed by tag handler...  
</oracust:mybodytag>
```

この場合、タグ・ハンドラは、BodyTag インタフェースを実装する必要があります。

TLD でのこのタグの <body-content> は、ボディ・コンテンツを変換する場合は JSP (デフォルト)、テンプレート・データとして処理する場合は tagdependent に設定する必要があります。

---

---

**注意：**

- 1 番目 (ボディがない場合) の操作は、**empty** アクションと呼ばれます。2～4 番目 (ボディがある場合) の操作は、**non-empty** アクションと呼ばれます。
  - 1～3 番目 (タグ・ハンドラによるボディ・コンテンツの処理が不要) の場合、ハンドラは単純タグ・ハンドラと呼ばれます。
  - <body-content> 要素の追加情報は、8-9 ページの「[tag 要素の使用](#)」を参照してください。
- 
-

## ボディ処理用の整定数の要約

次の各項で説明するタグ・ハンドラ・インタフェースは、状況に応じて適切な int 定数を戻すために実装する必要があるメソッドを指定します。

`doStartTag()` メソッドは、`Tag` インタフェースに定義され、`IterationTag` インタフェースおよび `BodyTag` インタフェースによって継承されます。このメソッドの可能な戻り値は、次のとおりです。

- `SKIP_BODY`: ボディがない場合、またはボディの評価をスキップする場合に、この値を使用します。
- `EVAL_BODY_INCLUDE`: ボディを評価して、現行の `JSP out` オブジェクトに渡す場合に、この値を使用します。ボディ・コンテンツの特別な処理は行われず、ボディ・コンテンツ・オブジェクトは作成されません。
- `EVAL_BODY_BUFFERED` (`BodyTag` クラスの場合のみ) : タグ・ボディのコンテンツについて `BodyContent` オブジェクトを作成し、コンテンツの評価と実行で使用する場合に、この値を使用します。
- `EVAL_BODY_TAG`: **この値は使用できなくなりました** (以前は、タグ・ハンドラによる特別な処理が必要なボディを使用する場合にこの値が使用されました)。  
`EVAL_BODY_AGAIN` または `EVAL_BODY_BUFFERED` を使用してください。両方とも、`EVAL_BODY_TAG` と同じ int 値を持ちます。

`doAfterBody()` メソッドは、`IterationTag` インタフェースに定義され、`BodyTag` インタフェースによって継承されます。このメソッドの可能な戻り値は、次のとおりです。

- `SKIP_BODY`: ボディの評価をスキップする場合、またはボディの反復を停止する場合に、この値を使用します。
- `EVAL_BODY_AGAIN`: ボディの反復を継続する場合に、この値を使用します。

`doEndTag()` メソッドは、`Tag` インタフェースに定義され、`IterationTag` インタフェースおよび `BodyTag` インタフェースによって継承されます。このメソッドの可能な戻り値は、次のとおりです。

- `SKIP_PAGE`: タグの後のページの残りをスキップする場合に、この値を使用します。この値によって、リクエストが完了します。
- `EVAL_PAGE`: タグの後のページの残りを評価する場合に、この値を使用します。



## 反復のない単純タグ・ハンドラ

ボディを使用しないカスタム・タグ、またはタグ・ハンドラによるアクセスや特別な処理を必要としないコンテンツを持つボディを使用するカスタム・タグの場合、タグ・ハンドラは単純タグ・ハンドラと呼ばれます。タグ・ハンドラ・クラスは、次の標準インタフェースを実装できます。

- `javax.servlet.jsp.tagext.Tag`

ただし、反復するタグ・ボディを使用する場合、タグ・ハンドラは、かわりに `IterationTag` インタフェースを実装する必要があります。8-32 ページの「[反復がある単純タグ・ハンドラ](#)」を参照。

標準の `javax.servlet.jsp.tagext.TagSupport` クラスは、`Tag` インタフェース以外に `IterationTag` インタフェースも実装します。このため、タグ・ボディを反復しないタグでの `TagSupport` クラスの使用は効率的ではありません。このことは、JSP 1.1 で `TagSupport` を拡張するタグ・ハンドラを作成している場合、JSP 1.1 環境から JSP 1.2 環境にコードを移行するときに特に考慮する必要があります。ボディの反復が不要な単純タグ・ハンドラでは、最初から `Tag` インタフェースを実装することをお勧めします。

`Tag` インタフェースは、次の主要な機能を実行するメソッドを定義します。

- JSP ページ・コンテキスト・オブジェクトの設定 (`pageContext` プロパティ)。
- 親タグ・ハンドラ (該当する場合で、最も近くにある、引用符で囲まれたタグ) の設定または取得 (`parent` プロパティ)。
- タグ属性の設定。
- `doStartTag()` メソッドの戻り値に応じて、タグ・ボディを条件付きで処理 (詳細は後述します)。
- `doEndTag()` メソッドの戻り値に応じて、タグの後の JSP ページの残りを条件付きで処理 (詳細は後述します)。
- 状態情報の解放。

詳細は、次のサイトにある、Sun 社の `Tag` インタフェースに関する Javadoc を参照してください。

[http://java.sun.com/j2ee/sdk\\_1.3/techdocs/api/javax/servlet/jsp/tagext/Tag.html](http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/jsp/tagext/Tag.html)

特に、`Tag` インタフェースは、次の重要なメソッドを指定します。

- `doStartTag()`
- `doEndTag()`

タグ開発者は、必要に応じて、これらのメソッドのコードをタグ・ハンドラ・クラスに用意します。これらのメソッドは、開始タグと終了タグがそれぞれ検出されると実行されます。JSP トランスレータによって生成された JSP ページの実装クラスには、これらのメソッドへの適切なコールが含まれます。

操作処理（操作タグで実行する処理）は、`doStartTag()` メソッドに実装します。`doEndTag()` メソッドは、適切な後処理を実装します。ボディを使用しないタグの場合は、基本的に、これら 2 つのメソッドが実行されても何も処理されません。

また、Tag インタフェースは、`pageContext` プロパティと `parent` プロパティの `getter` メソッドと `setter` メソッドも指定します。JSP ページ実装インスタンスは、`doStartTag()` メソッドと `doEndTag()` メソッドを起動する前に、`setPageContext()` メソッドと `setParent()` メソッドを起動します。

`doStartTag()` メソッドによって、`int` 値が戻されます。Tag インタフェースを実装しているタグ・ハンドラ・クラスの場合、この値は次のいずれかになります。

- `SKIP_BODY`: ボディ（ある場合）を評価しません。このハンドラに関連付けられたタグについて、TLD で `<body-content>` 設定が `empty` に指定されている場合は、このオプションのみ使用できます。
- `EVAL_BODY_INCLUDE`: ボディを評価して、現行の JSP out オブジェクトに渡します。

`doEndTag()` メソッドによって、`int` の値として次のいずれかが戻されます。

- `SKIP_PAGE`: タグの後のページの残りをスキップします。リクエストが別のページ（現行ページが転送されるかインクルードされたページ）から実行されている場合は、現行ページの残りの評価のみスキップされます。
- `EVAL_PAGE`: タグの後のページの残りを評価します。

## 反復がある単純タグ・ハンドラ

タグ・ハンドラによるアクセスおよび特別な処理を必要としないボディを使用する一方、反復などにより評価を繰り返す必要があるカスタム・タグの場合、タグ・ハンドラ・クラスは、次の標準インタフェースを実装できます。

- `javax.servlet.jsp.tagext.IterationTag`

`IterationTag` インタフェースは Tag インタフェースを拡張します。`IterationTag` インタフェースを実装するクラスも、単純タグ・ハンドラと呼ばれます。

次の標準サポート・クラスは、`IterationTag` インタフェースおよび `java.io.Serializable` インタフェースを実装し、ベース・クラスとして使用できます。

- `javax.servlet.jsp.tagext.TagSupport`

Tag インタフェースおよび `IterationTag` インタフェースからの適切なメソッドの実装に加えて、`TagSupport` クラスには便利なメソッドの `findAncestorWithClass()` が含まれています。このメソッドは、Tag インタフェースに定義された `getParent()` メソッドをコールします。

---

---

**注意：** タグ・ハンドラがボディの反復をサポートする必要がない場合、TagSupport クラスを拡張することはお薦めできません。TagSupport は IterationTag インタフェースを実装するため、必要としないループ・ロジックが含まれることとなります。このため、通常は効率が低下し、メソッドが Java のサイズ制限の 64KB を超える可能性が大きくなります。

---

---

IterationTag インタフェースは、Tag インタフェースから基本的なタグ処理機能 (doStartTag() メソッド、doEndTag() メソッドなど) を継承します。8-31 ページの「[反復のない単純タグ・ハンドラ](#)」を参照。

IterationTag インタフェースは、次の重要なメソッドも定義します。

■ doAfterBody()

このメソッドは、各タグ・ボディの評価後にコールされ、ボディを再評価する必要があるかどうかを確認します。このメソッドは、次のいずれかの int 値を戻します。

- SKIP\_BODY: 反復を停止し、タグ・ボディを再評価しません。かわりに、doEndTag() をコールします。SKIP\_BODY 設定は、ボディが最初に評価されない場合にも使用されます。また、このハンドラに関連付けられたタグについて、TLD で <body-content> 設定が empty に指定されている場合は、このオプションのみ使用できます。
- EVAL\_BODY\_AGAIN: 反復を継続し、タグ・ボディを再評価します。ボディの評価後、doAfterBody() メソッドが再度コールされます。

---

---

**注意：**

- JSP 仕様 1.1 では、doAfterBody() メソッドは BodyTag インタフェースに定義されていました。JSP 仕様 1.2 以上では、このメソッド定義を IterationTag インタフェースに移行することによって、単純反復タグ・ハンドラは BodyContent オブジェクトを維持するオーバーヘッドを回避できます。
  - IterationTag の詳細な使用例は、8-52 ページの「[例: IterationTag インタフェースの使用](#)」を参照してください。
- 
-

## ボディ・コンテンツにアクセスするタグ・ハンドラ

タグ・ハンドラによるアクセスを必要とするボディ・コンテンツを使用するカスタム・タグの場合、タグ・ハンドラ・クラスは、次の標準インタフェースを実装できます。

- `javax.servlet.jsp.tagext.BodyTag`

次の標準サポート・クラスは、`BodyTag` インタフェースおよび `java.io.Serializable` インタフェースを実装し、ベース・クラスとして使用できます。

- `javax.servlet.jsp.tagext.BodyTagSupport`

このクラスは、`Tag`、`IterationTag` および `BodyTag` インタフェースから適切なメソッドを実装します。

---

---

**注意：** タグ・ハンドラが実際にボディ・コンテンツにアクセスする必要がない場合は、`BodyTag` インタフェース（または `BodyTagSupport` クラス）を使用する必要はありません。使用すると、`BodyContent` オブジェクトを作成して維持するための不要なオーバーヘッドが発生することになります。ボディの反復が必要かどうかに応じて、`Tag` インタフェースまたは `IterationTag` インタフェース（または `TagSupport` クラス）を使用してください。

---

---

### BodyTag の機能

`BodyTag` インタフェースは、`Tag` インタフェースから基本的なタグ処理機能（`doStartTag()` メソッド、`doEndTag()` メソッド、およびその定義済の戻り値など）を継承します。このインタフェースは、`IterationTag` インタフェースからも機能（`doAfterBody()` メソッドおよびその定義済の戻り値など）を継承します。8-31 ページの「[反復のない単純タグ・ハンドラ](#)」および 8-32 ページの「[反復がある単純タグ・ハンドラ](#)」を参照。

`BodyTag` インタフェースには、継承した機能の他に、タグ・ボディから実行結果を取得する機能が追加されます。タグ・ボディの評価は、`javax.servlet.jsp.tagext.BodyContent` クラスのインスタンスにカプセル化されています。このインスタンスは、必要に応じて、ページ実装オブジェクトによって作成されず。8-36 ページの「[BodyContent オブジェクト](#)」を参照。

`Tag` インタフェースと同様に、`BodyTag` インタフェースに指定された `doStartTag()` メソッドは、`int` の戻り値として `SKIP_BODY` および `EVAL_BODY_INCLUDE` をサポートします。`BodyTag` インタフェースの場合、このメソッドは、`int` の戻り値として `EVAL_BODY_BUFFERED` もサポートします。次に、それぞれの値の概要を説明します。

- `SKIP_BODY`: ボディを評価しません。
- `EVAL_BODY_INCLUDE`: ボディを評価し、ボディ・コンテンツをタグ・ハンドラに対して使用可能にせず、JSP out オブジェクトに渡します。これは、基本的に `IterationTag` インタフェースを実装するタグ・ハンドラを使用した場合の `EVAL_BODY_INCLUDE` の動作と同じです。
- `EVAL_BODY_BUFFERED`: タグ・ボディ・コンテンツを処理するための `BodyContent` オブジェクトを作成します。

`BodyTag` インタフェースには、次のメソッドの定義も追加されます。

- `setBodyContent()`: タグ・ハンドラの `bodyContent` プロパティ (`BodyContent` インスタンス) を設定します。
- `doInitBody()`: タグ・ボディの評価を準備します。

`doStartTag()` メソッドで `EVAL_BODY_BUFFERED` が戻されると、JSP ページ実装インスタンスは次の手順を順番に実行します。

1. `BodyContent` インスタンスを作成します。
2. タグ・ハンドラの `setBodyContent()` メソッドをコールし、`BodyContent` インスタンスをタグ・ハンドラに渡します。
3. タグ・ハンドラの `doInitBody()` メソッドをコールし、`BodyContent` インスタンスに関連する初期化 (必要な場合) を実行します。

これらの手順は、タグ・ボディが評価される前に実行されます。ボディの評価中、JSP out オブジェクトは `BodyContent` オブジェクトにバインドされます。

各ボディの評価後に、`IterationTag` インタフェースを実装しているタグ・ハンドラに対して、ページ実装インスタンスはタグ・ハンドラの `doAfterBody()` メソッドをコールします。可能な戻り値は次のとおりです。

- `SKIP_BODY`: 反復を停止し、タグ・ボディを再評価しません。かわりに、`doEndTag()` をコールします。JSP out オブジェクトは、ページ・コンテキストからリストアされません。
- `EVAL_BODY_AGAIN`: 反復を継続し、タグ・ボディを再評価します。ボディは、評価時に現行の JSP out オブジェクトに渡されます。ボディの評価後、`doAfterBody()` メソッドが再度コールされます。

ボディの評価が完了しても多数の反復が必要な場合、ページ実装インスタンスは、タグ・ハンドラの `doEndTag()` メソッドを起動します。

## BodyContent オブジェクト

BodyTag インタフェースを実装しているタグ・ハンドラでは、`javax.servlet.jsp.tagext.BodyContent` クラスのインスタンスを通じてタグ・ボディの評価結果にアクセスできます。このクラスは、`javax.servlet.jsp.JspWriter` クラスを拡張します。

BodyContent インスタンスは、JSP ページ・コンテキストの `pushBody()` メソッドを使用して作成されます。

BodyContent クラスには、継承した `JspWriter` 機能の他に、次の処理を実行するメソッドも追加されます。

- コンテンツを `java.io.Reader` オブジェクトとして戻します (`getReader()` メソッド)。
- コンテンツを `java.io.Writer` オブジェクトに書き込みます (`writeOut()` メソッド)。
- コンテンツを `String` オブジェクトに変換します (`getString()` メソッド)。
- コンテンツをクリアします (`clearBody()` メソッド)。

BodyContent オブジェクトの一般的な使用方法は、次のとおりです。

- コンテンツを `String` インスタンスに変換し、その文字列を操作の値として使用します。
- コンテンツを JSP out オブジェクトに書き込みます。このオブジェクトは、開始タグが検出された時点でアクティブになっています。

## TryCatchFinally インタフェース

タグの処理中に例外が発生した場合のデータ整合性とリソース管理のため、JSP 仕様 1.2 では、`javax.servlet.jsp.tagext.TryCatchFinally` インタフェースが導入されました。エラーの処理が必要なタグの場合や、リソースの適切な解放が必要な場合は、このインタフェースをタグ・ハンドラに実装すると特に役立ちます。

TryCatchFinally インタフェースは、次のメソッドを指定します。

- `void doCatch(java.lang.Throwable throw)`

このメソッドは、タグ・ボディの評価中、あるいは、`doStartTag()`、`doEndTag()`、`doAfterBody()` または `doInitBody()` メソッドのコール中に `Throwable` エラーが発生した場合に、タグ・ハンドラで起動できます。検出された `Throwable` オブジェクトは、`doCatch()` メソッドで入力として取得されます。`setter` メソッドのコール中に `Throwable` エラーが発生した場合、このメソッドは起動されません。

`doCatch()` メソッドは、例外（元の `Throwable` 例外または新規の例外）をスローして、エラー連鎖に伝播できます。

- `void doFinally()`

このメソッドは、`doCatch()` メソッドで説明した `Throwable` エラーが発生したかどうかに関係なく、起動されます。ただし、`setter` メソッドのコール中に `Throwable` エラーが発生した場合、このメソッドは起動されません。

`doFinally()` メソッドは、例外をスローしません。

次に、Sun 社の JSP 仕様 1.2 から抜粋した、一般的な `TryCatchFinally` の起動の例を示します。

```
h = get a Tag(); // get a tag handler, perhaps from pool

h.setPageContext(pc); // initialize as desired
h.setParent(null);
h.setFoo("foo");

// tag invocation protocol; see Tag.java
try {
    h.doStartTag()...
    ....
    h.doEndTag()...
} catch (Throwable t) {
    /* React to exceptional condition; invoked if exception occurs between
       doStartTag() and doEndTag(). */
    h.doCatch(t);
} finally {
    // restore data invariants and release pre-invocation resources
    h.doFinally();
    /* doFinally() is almost always called, unless Throwable error occurs
       during setter method, or Java thread terminates. */
}
```

## 外部タグ・ハンドラ・インスタンスへのアクセス

ネストされたカスタム・タグを使用する場合、ネストされたタグのタグ・ハンドラ・インスタンスは、外部タグのタグ・ハンドラ・インスタンスにアクセスできます。これは、ネストされたタグによる処理と状態管理に役立ちます。

この機能は、`javax.servlet.jsp.tagext.TagSupport` クラスの静的な `findAncestorWithClass()` メソッドを介してサポートされます。外部タグ・ハンドラ・インスタンスの名前が JSP ページ・コンテキストに指定されていない場合でも、その外部タグ・ハンドラ・インスタンスは、指定したタグ・ハンドラ・クラスのインスタンスの最も近くにある、引用符で囲まれたインスタンスであるため、アクセス可能です。

次の JSP コードの例を考えてみます。

```
<foo:bar1 attr="abc" >
  <foo:bar2 />
</foo:bar1>
```

bar2 タグ・ハンドラ・クラス（表記規則に従って、クラス名は Bar2Tag になります）のコード内に、次の文を含めることができます。

```
Tag bar1tag = TagSupport.findAncestorWithClass(this, Bar1Tag.class);
```

findAncestorWithClass() メソッドへの入力項目は、次のとおりです。

- findAncestorWithClass() がコールされたクラス・ハンドラ・インスタンスである this オブジェクト（この例では、Bar2Tag インスタンス）
- java.lang.Class インスタンスとして、bar1 タグ・ハンドラ・クラス（この例では、Bar1Tag とします）の名前

findAncestorWithClass() メソッドは、適切なタグ・ハンドラ・クラスのインスタンスを戻します。この例では、Bar1Tag を javax.servlet.jsp.tagext.Tag インスタンスとして戻します。

これは、Bar2Tag インスタンスで bar1 タグ属性の値が必要な場合、または Bar1Tag インスタンスについてのメソッドのコールが必要な場合に、Bar2Tag が外部の Bar1Tag インスタンスにアクセスするのに役立ちます。

## OC4J の JSP タグ・ハンドラの機能

この項では、タグ・ハンドラのプーリング、および生成コードのサイズ縮小に関する OC4J の JSP 拡張機能について説明します。この項には、次の項目が含まれます。

- [実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)
- [タグ・ハンドラのコード生成](#)

### 実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化

パフォーマンスを改善するために、各 JSP ページ内でタグ・ハンドラ・インスタンスを再利用するように指定できます。これは、タグ・ハンドラ・インスタンスのプーリングとも呼ばれます。Oracle Application Server 10g (9.0.4) では、次の 2 つのモデルがあります。

- **実行時モデル**: タグ・ハンドラ再利用のロジックとパターンは、実行時（JSP ページの実行時）に決定されます。タグ・ハンドラは、application スcope内で再利用されません。
- **コンパイル時モデル**: タグ・ハンドラ再利用のロジックとパターンは、コンパイル時（JSP ページの変換時）に決定されます。この方法は、同じページ内に多数のタグがある（たとえば、数百のタグがある）アプリケーションの場合、パフォーマンスを効率的に改善できます。



JSP 構成パラメータの `tags_reuse_default` は、いずれのモデルでも使用できます。このパラメータとその設定方法の詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

## タグ・ハンドラ再利用に関する重要事項

タグ・ハンドラ再利用に関しては、次の事項に注意してください。

- 現在の実装では、`tags_reuse_default` のデフォルト設定は `runtime` で、実行時モデルが使用されます。
- この設定を実行時モデル (`tags_reuse_default` の値は `runtime`) からコンパイル時モデル (`tags_reuse_default` の値は `completetime` または `completetime_with_release`) に変更した場合、またはコンパイル時モデルから実行時モデルに変更した場合は、JSP ページを再変換する必要があります。
- JSP コンテナは、JServ 環境でのタグ・ハンドラ再利用もサポートします。この環境では、`tags_reuse_default` のデフォルト設定は `none` で、タグ・ハンドラは再利用されません。
- 指定したタグ・ハンドラ・インスタンスによって、一度に 1 つのリクエストのみ処理されます。

## タグ・ハンドラ再利用の実行時モデルの有効化または無効化

実行時モデルは、次のいずれかの方法で有効化できます。

- `tags_reuse_default` のデフォルト値の `runtime` を使用します (下位互換性を維持するために、`runtime` と等価の `true` の設定もサポートされています)。

または

- `tags_reuse_default` の値が `none` の場合は、JSP ページ・コンテキストの `oracle.jsp.tags.reuse` 属性を `true` に設定することによって、特定の JSP ページでこの値をオーバーライドできます。例：

```
pageContext.setAttribute("oracle.jsp.tags.reuse", new Boolean(true));
```

実行時モデルは、次のいずれかの方法で無効化できます。

- `tags_reuse_default` の値を `none` に設定します。この設定によって、コンパイル時モデルも無効化されます (下位互換性を維持するために、`none` と等価の `false` の設定もサポートされています)。

または

- `tags_reuse_default` の値が `runtime` の場合は、JSP ページ・コンテキストの `oracle.jsp.tags.reuse` 属性を `false` に設定することによって、特定の JSP ページでこの値をオーバーライドできます。例：

```
pageContext.setAttribute("oracle.jsp.tags.reuse", new Boolean(false));
```

---

---

**注意：**

- タグ・ハンドラ再利用の設定をコンパイル時モデルから実行時モデルに変更した場合は、JSP ページを再変換する必要があります。
  - ページごと、または同じページのセクションごとに、異なる `oracle.jsp.tags.reuse` 設定を使用できます。
  - `tags_reuse_default` の設定が `completetime` または `completetime_with_release` の場合、`oracle.jsp.tags.reuse` 属性は無視されます。
- 
- 

**タグ・ハンドラ再利用のコンパイル時モデルの有効化または無効化**

次のいずれかの方法で、タグ・ハンドラ再利用をコンパイル時モデルに変更できます。

- 構成パラメータの `tags_reuse_default` を `completetime` に設定します。

または

- 構成パラメータの `tags_reuse_default` を `completetime_with_release` に設定します。

`completetime_with_release` に設定すると、同じページ内で同じタグ・ハンドラが使用されるたびに、タグ・ハンドラの `release()` メソッドがコールされます。このメソッドは、タグ・ハンドラ実装の詳細に従って、状態情報を解放します。たとえば、タグが使用されるたびに状態情報を解放するようにタグ・ハンドラがコーディングされている場合は、`completetime_with_release` 設定が適切です。タグ・ハンドラの実装、および使用するコンパイル時設定に関する詳細が不明な場合は、それぞれの値を試してください。

コンパイル時モデルを無効化するには、`tags_reuse_default` の値を `none` に設定します。この設定によって、実行時モデルも無効化されます。

---

---

**注意：**

- 実行時モデルからコンパイル時モデルに変更した場合は、JSP ページを再変換する必要があります。
  - `tags_reuse_default` の設定が `completetime` または `completetime_with_release` の場合、ページ・コンテキストの `oracle.jsp.tags.reuse` 属性は無視されます。
- 
-

## タグ・ハンドラのコード生成

Oracle JSP 実装では、カスタム・タグを使用するための生成コードのサイズが縮小されています。また、JSP 構成フラグの `reduce_tag_code` を `true` に設定すると、さらにサイズを縮小できます。

ただし、このフラグを有効にすると、タグ・ハンドラを最大限に再利用するコード生成パターンになりません。8-38 ページの「[実行時またはコンパイル時のタグ・ハンドラ再利用の有効化または無効化](#)」で説明したように、`tags_reuse_default` を `true` に設定するとパフォーマンスを改善できますが、`reduce_tag_code` も `true` に設定すると、その効果は最大限になりません。

これらのパラメータとその設定方法の詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

## スクリプト変数、宣言およびタグ補足情報クラス

カスタム・タグ操作によって、タグ自体またはスクリプトレットや他のタグなどのスクリプト要素で使用できる 1 つ以上のサーバー・サイド・オブジェクト（スクリプト変数と呼ばれます）を作成できます。スクリプト変数は、基本的にはタグ・ライブラリの TLD に `<variable>` 要素を使用して定義し、スクリプト変数のロジックが複雑な場合は、タグ補足情報クラスを使用して定義できます。

この項には、次の項目が含まれます。

- [スクリプト変数の使用](#)
- [スクリプト変数のスコープ](#)
- [TLD の `variable` 要素を使用した変数宣言](#)
- [タグ補足情報クラスを使用した変数宣言](#)

### スクリプト変数の使用

カスタム・タグに明示的に定義されたオブジェクトは、そのオブジェクトの ID をハンドルとして使用して、JSP ページ・コンテキストから他の操作で参照できます。次に例を示します。

```
<oracust:foo id="myobj" attr1="..." attr2="..." />
```

この文によって、`myobj` オブジェクトは、`myobj` の宣言されたスコープに基づいて、ページ内のスクリプト要素で使用可能になります（8-42 ページの「[スクリプト変数のスコープ](#)」を参照）。`id` 属性は変換時の属性です。変数は次のいずれかの方法で指定できます。

- 変数用の <variable> 要素を TLD に指定し、変数の名前と型および追加情報を指定します。8-43 ページの「[TLD の variable 要素を使用した変数宣言](#)」を参照。
- タグ補足情報クラスを作成して、変数の名前と型、追加情報および関連ロジックを指定します。タグ補足情報クラスの名前は、TLD の <tei-class> 要素に指定します。8-44 ページの「[タグ補足情報クラスを使用した変数宣言](#)」を参照。

通常は、<variable> 機能を使用すると便利です。

JSP コンテナによって、myobj がページ・コンテキストに入力されます。これによって、他のタグまたはスクリプト変数は、次の構文を使用して、後でこのオブジェクトを取得できます。

```
<oracust:bar ref="myobj" />
```

この myobj オブジェクトは、foo タグと bar タグのタグ・ハンドラ・インスタンスに渡されます。ここで必要なのは、オブジェクトの名前 (myobj) のみです。

---

---

**注意：** id と ref はサンプルの属性名であるため、これらの属性名に対する事前定義の特別なセマンティクスはありません。属性名を定義し、ページ・コンテキスト内のオブジェクトを作成および取得するのは、タグ・ハンドラが行います。

---

---

## スクリプト変数のスコープ

変数を作成するタグの <variable> 要素またはタグ補足情報クラスに、スクリプト変数のスコープを指定します。スコープには、次の int 定数のいずれかを指定できます。

- NESTED: スクリプト変数を定義する操作の開始タグと終了タグの間で使用可能なスクリプト変数の場合は、この設定を使用します。
- AT\_BEGIN: 開始タグからそのページの最後まで使用可能なスクリプト変数の場合は、この設定を使用します。
- AT\_END: 終了タグからそのページの最後まで使用可能なスクリプト変数の場合は、この設定を使用します。

## TLD の variable 要素を使用した変数宣言

JSP 仕様 1.1 では、カスタム・タグでスクリプト変数を使用するには、タグ補足情報 (TEI) クラスを作成する必要がありました。8-44 ページの「[タグ補足情報クラスを使用した変数宣言](#)」を参照。JSP 仕様 1.2 では、より簡単な方法が導入され、関連付けられたタグが定義されている TLD で `<variable>` 要素を使用できます。変数に関連するロジックが単純で、TEI クラスを使用する必要がない場合は、この方法で十分です。

`<variable>` 要素は、変数を使用するタグを定義する `<tag>` 要素内のサブ要素です。

変数の名前は、次のいずれかの方法で指定できます。

- `<variable>` 要素内の `<name-given>` サブ要素を使用して、変数の名前を直接指定します。

または

- `<variable>` 要素内の `<name-from-attribute>` サブ要素を使用して、タグ属性を指定します (タグ属性の値によって、変換時に変数の名前が指定されます)。

`<name-given>` および `<name-from-attribute>` 以外に、`<variable>` 要素には次のサブ要素があります。

- `<variable-class>` 要素は、変数のクラスを指定します。デフォルトは `java.lang.String` です。
- `<declare>` 要素は、その変数が新規に宣言される変数かどうかを指定します。この場合、JSP トランスレータが変数を宣言します。デフォルトは `true` です。 `false` に設定すると、変数は、標準的な機能 (`jsp:useBean` 操作、JSP スクリプトレット、JSP 宣言またはカスタム・アクションなど) を使用して、JSP ページですでに宣言されているとみなされます。
- `<scope>` 要素は、変数のスコープとして、`NESTED`、`AT_BEGIN` または `AT_END` のいずれかを指定します。これらの値の説明は、8-42 ページの「[スクリプト変数のスコープ](#)」を参照してください。デフォルトは `NESTED` です。

次の例では、タグ `myaction` に対して 2 つのスクリプト変数を宣言しています。`<tag>` 要素内で、変数の説明に直接関係ない部分は省略されていることに注意してください。

```
<tag>
  <name>myaction</name>
  ...
  <attribute>
    <name>attr2</name>
    <required>true</required>
  </attribute>
  <variable>
    <name-given>foo_given</name-given>
    <declare>false</declare>
    <scope>AT_BEGIN</scope>
  </variable>
```

```
<variable>
  <name-from-attribute>attr2</name-from-attribute>
  <variable-class>java.lang.Integer</variable-class>
</variable>
</tag>
```

最初の変数の名前は、foo\_given にハードコードされています。デフォルトで、この変数の型は String になります。新規に宣言される変数ではなく、すでに宣言されている変数とみなされるため、そのスコープは開始タグからページの最後までとなります。

2 番目の変数の名前は、必須の attr2 属性の設定に基づきます。この変数の型は Integer です。デフォルトでは、この変数は新規に宣言され、そのスコープは NESTED (myaction の開始タグと終了タグの間) になります。

関連する TLD 構文の詳細は、8-7 ページの「[タグ・ライブラリ・ディスクリプタ](#)」を参照してください。

## タグ補足情報クラスを使用した変数宣言

複雑な関連ロジックを持つスクリプト変数の場合、変数を宣言するために TLD 内で <variable> 要素を使用するだけでは不十分な場合があります。この場合は、スクリプト変数に関する詳細を javax.servlet.jsp.tagext.TagExtraInfo 抽象クラスのサブクラスに指定できます。このマニュアルでは、このようなサブクラスをタグ補足情報クラスと呼びます。タグ補足情報クラスは、タグ属性の追加の妥当性チェックをサポートし、スクリプト変数に関する追加情報を JSP 実行時機能に提供します。

JSP コンテナは、タグ補足情報インスタンスを変換時に使用します。指定のタグのスクリプト変数で使用するタグ補足情報クラスは、TLD で指定します。次に、<tei-class> 要素の使用例を示します。

```
<tag>
  <name>loop</name>
  <tag-class>examples.ExampleLoopTag</tag-class>
  <tei-class>examples.ExampleLoopTagTEI</tei-class>
  <body-content>JSP</body-content>
  <description>for loop</description>
  <attribute>
    ...
  </attribute>
  ...
</tag>
```

次に、関連クラスを示します。これらのクラスも javax.servlet.jsp.tagext パッケージに含まれています。

- **TagData**: このクラスのインスタンスには、タグ・インスタンスの変換時の属性値情報が含まれます。
- **VariableInfo**: このクラスの各インスタンスには、実行時にタグによって宣言、作成または変更されたスクリプト変数の情報が含まれます。
- **TagInfo**: このクラスのインスタンスには、関連するタグに関する情報が含まれます。クラスは TLD からインスタンス化され、変換時のみ使用できます。TagInfo には、`getTagName()`、`getTagClassName()`、`getBodyContent()`、`getDisplayName()` および `getInfoString()` などのメソッドがあります。

詳細は、次の場所を参照してください。

[http://java.sun.com/j2ee/sdk\\_1.3/techdocs/api/javax/servlet/jsp/tagext/package-summary.html](http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/jsp/tagext/package-summary.html)

---

---

**注意:** tag-extra-info 実装で TagInfo インスタンスを使用するのは、一般的ではありません。ただし、たとえば、複数のタグ・ライブラリと TLD に 1 つのタグ補足情報クラスをマッピングする場合には便利です。

---

---

TagExtraInfo クラスには、次のメソッドが関連しています。

- `boolean isValid(TagData data)`  
JSP トランスレータは、このメソッドをコールして、変換時にタグ属性の妥当性チェックを実行し、TagData インスタンスに渡します。
- `VariableInfo[] getVariableInfo(TagData data)`  
JSP トランスレータは、変換時にこのメソッドをコールして、TagData インスタンスに渡します。このメソッドは、VariableInfo インスタンスの配列を戻します (タグで作成されたスクリプト変数ごとに 1 つのインスタンスが含まれます)。
- `void setTagInfo(TagInfo info)`  
このメソッドをコールすると、TagInfo インスタンスがタグ補足情報クラスの属性として設定されます。このメソッドは通常、JSP コンテナによってコールされます。
- `TagInfo getTagInfo()`  
このメソッドを使用して、タグ補足情報クラスの TagInfo 属性を取得します。TagInfo 属性は以前に設定されていることが前提です。

タグ補足情報クラスは、スクリプト変数に関する次の情報を使用して、各 VariableInfo インスタンスを構成します。

- 変数の名前
- 変数の Java 型 (プリミティブ型以外)

- 新規に宣言された変数かどうかを示すブール値（この場合、JSP トランスレータが変数を宣言します）
- 変数のスコープ

---

---

**重要：** OC4J 9.0.4 実装では、`getVariableInfo()` メソッドは、スクリプト変数の Java 型について、完全修飾クラス名 (FQCN) または部分修飾クラス名 (PQCN) のいずれかを戻すことができます。以前のリリースでは FQCN が必須でしたが、パッケージ間でクラス名が重複するのを避けるため、現行のリリースでも FQCN の使用をお勧めします。プリミティブ型はサポートされていません。

---

---

タグ補足情報クラスのサンプル・コードは、8-58 ページの「[タグ補足情報クラスのサンプル : ExampleLoopTagTEI.java](#)」を参照してください。

## 妥当性チェックおよびタグ・ライブラリ・バリデータ・クラス

JSP 仕様 1.2 では、必要に応じてバリデータ・クラスを各タグ・ライブラリに関連付ける機能が導入されました。このクラスは、タグ・ライブラリ・バリデータ (TLV) ・クラスと呼ばれます。TLV クラスの目的は、タグ・ライブラリを使用する任意の JSP ページの妥当性チェックを実行することです。TLV クラスの実装によって、ページが指定の制約に準拠していることを検証します。TLV クラスでは、関連するタグ・ライブラリの使用方法に関してのみ制約事項をチェックするのが一般的ですが、チェック内容に制限はありません。TLV クラスでは JSP ページのあらゆる側面をチェックできます。

タグ・ライブラリ・バリデータ・クラスは、`javax.servlet.jsp.tagext.TagLibraryValidator` クラスのサブクラスであることが必要です。

次の各項では、タグ・ライブラリの妥当性チェックと TLV クラスについて説明します。

- [TLD の validator 要素](#)
- [主要な TLV 関連クラスおよび validation\(\) メソッド](#)
- [TLV の処理](#)
- [妥当性チェック機能](#)



## TLD の validator 要素

タグ・ライブラリに対して TLV クラスを指定するには、TLD で `<validator>` 要素を使用します。`<validator>` 要素には、次のサブ要素があります。

- `<validator-class>` サブ要素は、TLV クラスの名前を指定します。
- `<description>` サブ要素は、TLV クラスに関する説明を提供します。
- `<init-param>` サブ要素とそのサブ要素 (`<param-name>` および `<param-value>`) は、TLV クラスの初期化パラメータの設定に使用できます。これは、アプリケーションのデプロイメント・ディスクリプタ (`web.xml`) 内にある `<servlet>` 要素の `<init-param>` サブ要素の動作に類似しています。`<init-param>` 要素の下には、オプションの `<description>` サブ要素もあります。

次に、Sun 社の JSTL 仕様 1.0 から抜粋した、`<validator>` 要素の例を示します。

**例 1** TLV クラス (`ScriptFreeTLV`) の例を示します。このクラスは、初期化パラメータの設定に従って、JSP 宣言、JSP スクリプトレット、JSP 式、および実行時の式を使用禁止にできます。この例では、JSP 式および実行時の式が使用可能になり、JSP 宣言または JSP スクリプトレットが使用禁止になります。

```
<validator>
  <validator-class>
    javax.servlet.jsp.jstl.tlv.ScriptFreeTLV
  </validator-class>
  <init-param>
    <param-name>allowDeclarations</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>allowScriptlets</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>allowExpressions</param-name>
    <param-value>>true</param-value>
  </init-param>
  <init-param>
    <param-name>allowRTEExpressions</param-name>
    <param-value>>true</param-value>
  </init-param>
</validator>
```

**例 2** TLV クラス (PermittedTagLibsTLV) の例を示します。このクラスは、初期化パラメータで指定したタグ・ライブラリのみ使用可能にします。TLV クラスが関連付けられたタグ・ライブラリは、暗黙的に使用可能になります。さらに、TLV クラスを使用すると、初期化パラメータ設定でリストに指定したライブラリを使用可能にできます。リスト内のエントリは空白で区切られます。この例では、JSTL ライブラリの core、xml、fmt および sql のみ使用可能にします。

```
<validator>
  <validator-class>
    javax.servlet.jsp.jstl.tlv.PermittedTaglibsTLV
  </validator-class>
  <init-param>
    <param-name>permittedTaglibs</param-name>
    <param-value>
      http://java.sun.com/jstl/core
      http://java.sun.com/jstl/xml
      http://java.sun.com/jstl/fmt
      http://java.sun.com/jstl/sql
    </param-value>
  </init-param>
</validator>
```

## 主要な TLV 関連クラスおよび validation() メソッド

前述したように、TLV クラスは javax.servlet.jsp.tagext.TagLibraryValidator クラスのサブクラスです。

次の関連クラスも javax.servlet.jsp.tagext パッケージに含まれています。

- PageData: このクラスのインスタンスは、JSP トランスレータによって生成され、変換されるページの XML ビューに対応する情報が含まれます。
- ValidationMessage: このクラスのインスタンスには、TLV インスタンスからのエラー・メッセージが含まれます。エラー・メッセージは、TLV の validate() メソッドによって戻されます。

TLV クラスの主要なメソッドは、次のとおりです。

- ValidationMessage[] validate  
(String prefix, String uri, PageData page)

JSP コンテナは、<validator> 要素が指定された TLD をポイントする taglib ディレクティブを検出するたびに、このメソッドをコールします。このメソッドは、入力として、タグ・ライブラリの接頭辞、TLD の URI、およびページの PageData オブジェクト (XML ビュー) を取得します。妥当性のチェック中にエラーが発生すると、validate() メソッドは妥当性チェック・メッセージの配列を戻します。jsp:id 属性 (オプション) は OC4J JSP コンテナでサポートされているため、jsp:id の値は妥当性チェック・メッセージに含まれます。

詳細は、次項の「[TLV の処理](#)」を参照してください。

## TLV の処理

変換時に JSP ページで `taglib` ディレクティブが検出されるたびに、JSP コンテナは、関連する TLD を検索して、TLV クラスを指定している `<validator>` 要素を調べます。該当する TLD が見つかり、コンテナでは、変換時に次の手順を実行します。ここで説明するクラスとメソッドに関するバックグラウンド情報は、前項の「[主要な TLV 関連クラスおよび validation\(\) メソッド](#)」を参照してください。

1. `<validator>` 要素の `<init-param>` サブ要素に指定された初期化パラメータ設定を使用して、TLV クラスをインスタンス化します。
2. TLV インスタンスに対して、JSP ページの XML ビューを公開します（5-14 ページの「[JSP XML ビューの詳細](#)」を参照）。
3. TLV インスタンスの `validate()` メソッドをコールして、JSP ページの妥当性チェックを実行します（次項の「[妥当性チェック機能](#)」を参照）。エラーが発生した場合、このメソッドは、`ValidationMessage` インスタンスの配列を戻します。エラーがない場合、このメソッドは、`null` または空の `ValidationMessage[]` 配列を戻すことができます。

---

**注意：** OC4J JSP コンテナは、必要に応じて、妥当性チェック・エラーのレポート機能を改善するために、JSP 1.2 機能の `jsp:id` 属性を実装できません。詳細は、5-15 ページの「[妥当性チェックにおけるエラー・レポートの `jsp:id` 属性](#)」を参照してください。

---

4. このライブラリ（TLV クラスに関連付けられたライブラリ）に属するカスタム・タグが検出されるたびに、タグ補足情報クラスがチェックされます。指定されている場合は、JSP コンテナによってインスタンス化され、`isValid()` メソッドがコールされてタグの属性の妥当性チェックが実行されます。`isValid()` メソッドは、妥当性チェックが正常に実行されると `true` を返し、失敗すると `false` を返します。

## 妥当性チェック機能

通常、JSP ページの XML ビューの妥当性を DTD と照合してチェックすることはできません。JSP ページの XML ビューには DOCTYPE 文が含まれていません。名前空間を識別する様々な機能を、妥当性チェックで使用できます。その中の 1 つが、W3C の XML Schema 言語です。詳細は、W3C の次の Web サイトを参照してください。

<http://www.w3.org/XML/>

JSP ページで特定の要素セットのみが使用されていること、または使用されていないことを単に確認する場合などは、基本的な機能で十分な場合があります。

## タグ・ライブラリのイベント・リスナー

サーブレット仕様では、次のタイプのイベント・リスナーの使用方法を説明しています。

- サーブレット・コンテキスト・リスナー。  
javax.servlet.ServletContextListener インタフェースを実装します。
- サーブレット・コンテキスト属性リスナー。  
javax.servlet.ServletContextAttributeListener インタフェースを実装します。
- HTTP セッション・リスナー。javax.servlet.http.HttpSessionListener インタフェースを実装します。
- HTTP セッション属性リスナー。  
javax.servlet.http.HttpSessionAttributeListener インタフェースを実装します。

サーブレット 2.3 の機能では、アプリケーションの web.xml ファイルにイベント・リスナーを指定できます。これによって、イベント・リスナーがサーブレット・コンテナに登録され、関連する状態変更が通知されます。たとえば、サーブレット・コンテキスト・リスナーには、アプリケーションの起動や停止など、アプリケーションの ServletContext オブジェクトにおける変更が通知されます。イベント・リスナーの追加情報は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

JSP 仕様 1.2 では、タグ・ライブラリのパッケージングとデプロイを容易にするため、TLD での <listener> 要素のサポートが導入されました。web.xml ファイルにイベント・リスナーを指定するかわりに、この要素を使用してイベント・リスナーを指定できます。次の各項では、次の機能について説明します。

- [TLD の listener 要素](#)
- [タグ・ライブラリのイベント・リスナーのアクティブ化](#)
- [イベント・リスナー情報を取得するための TLD へのアクセス](#)

## TLD の listener 要素

TLD では、各 <listener> 要素は、<taglib> ルート要素の下のトップレベルの要素となります。<listener-class> 要素は、<listener> 要素の必須のサブ要素で、インスタンス化するリスナー・クラスを指定します。このクラスは、ServletContextListener、ServletContextAttributeListener、HttpSessionListener または HttpSessionAttributeListener インタフェースを実装します。

次に例を示します。

```
<taglib>
...
  <listener>
    <listener-class>mypkg.MyServletContextListener</listener-class>
  </listener>
...
</taglib>
```

## タグ・ライブラリのイベント・リスナーのアクティブ化

アプリケーションが起動すると、サーブレット・コンテナは、JSP コンテナをコールして、次の処理を実行します。

1. TLD を検索してアクセスします。
2. TLD を読み取り、その <listener> 要素を検索します。
3. リスナーをインスタンス化して登録します。

これは、特定のタグ・ライブラリの使用に関連付けられたアプリケーション・レベルおよびセッション・レベルのリソースを管理できる便利な方法です。この機能は、基本的に、web.xml ファイルに指定されたサーブレット・コンテキスト・リスナーの機能と同じです。

---

---

### 注意：

- TLD に指定されたイベント・リスナーの場合、リスナーが登録される順序は定義されていませんが、アプリケーションの起動前、および web.xml ファイルに指定されているリスナーの後に、すべてのリスナーが登録されます。
  - TLD が WAR ファイル構造内に存在する場合は、その構造内でリスナーが検索され、関連するタグ・ライブラリが実際にはアプリケーションで使用されていない場合でも、すべてのリスナーが登録されます。
- 
-

## イベント・リスナー情報を取得するための TLD へのアクセス

JSP コンテナが TLD にアクセスして `<listener>` 要素を検索できるように、標準的な方法を使用する必要があります。TLD の場所、アクセス可能性およびパッケージングに関する一般情報は、8-13 ページの「[タグ・ライブラリと TLD の設定およびアクセス](#)」を参照してください。OC4J の予約済タグ・ライブラリの場所に関する説明が記載されています。

さらに、一般的に、TLD が予約済のタグ・ライブラリ・ディレクトリにあり、TLD に指定されたリスナーをアプリケーションでアクティブ化する場合は、アプリケーションの `web.xml` ファイルの `<taglib>` 要素にタグ・ライブラリを指定する必要があります。この手順を実行しないと、共有ディレクトリ内にある TLD にアクセスできないため、`<listener>` 要素を検索できません。これは、共有ライブラリ内のタグ・ライブラリを使用しないアプリケーションで、パフォーマンスの低下を防ぐためです。ただし、永続的な TLD キャッシングを使用している場合（8-19 ページの「[タグ・ライブラリを共有するための Oracle の拡張機能と永続的な TLD キャッシング](#)」を参照）、`web.xml` の `<taglib>` 要素は必要ありません。

## カスタム・タグの完全な例

次の各項では、カスタム・タグの使用の完全な例として、JSP ページ、タグ・ハンドラ・クラスおよびタグ・ライブラリ・ディスクリプタの各サンプルを示します。

- 例 : [IterationTag インタフェースの使用](#)
- 例 : [IterationTag インタフェースおよびタグ補足情報クラスの使用](#)

---

---

**注意：** ここで示す例は説明を目的としているため、必ずしも現実的で効率的な使用方法を示しているわけではありません。

---

---

### 例 : IterationTag インタフェースの使用

このサンプルでは、`myIterator` というカスタム・タグを使用して、コレクション内の現在の項目をスクリプト変数として使用可能にする方法を示します。スクリプト変数は、TLD の `<variable>` 要素を使用して定義します。

解凍やデプロイなど、この例に関する詳細情報は、次の OTN-J の Web サイトを参照してください。

<http://otn.oracle.co.jp/products/oc4j/index.html>

(この Web サイトを参照するには登録が必要ですが、無償で登録可能です。)

## JSP ページのサンプル : exampleiterator.jsp

次の JSP ページでは myIterator タグを使用しています。

```
<%@ page contentType="text/html; charset=windows-1252"%>
<HTML>
<HEAD>
<TITLE>
JSP 1.2 IterationTag Sample
</TITLE>
</HEAD>
<%@ taglib uri="/WEB-INF/exampleiterator.tld" prefix="it"%>
<BODY>

<% java.util.Vector vector = new java.util.Vector();
   vector.addElement("One");
   vector.addElement("Two");
   vector.addElement("Three");
   vector.addElement("Four");
   vector.addElement("Five");
%>
   Collection to Iterate over is <%=vector%> ..... <p>

   <B>Iterating ...</B><br>
   <it:myIterator collection="<%= vector%>" >
       Item <B><%= item%></B><br>
   </it:myIterator>
</p>
</BODY>
</HTML>
```

## タグ・ハンドラ・クラスのサンプル : MyIteratorTag.java

このサンプルで示すタグ・ハンドラ・クラス MyIteratorTag では、doStartTag() メソッドにより、コレクションが NULL かどうかチェックされます。NULL でない場合、このメソッドはコレクション・オブジェクトを取得します。イテレータに 1 つ以上の要素が含まれている場合、doStartTag() メソッドは、コレクション内の最初の項目を page スコープのオブジェクトとして使用可能にし、EVAL\_BODY\_INCLUDE を戻します。これによって、タグ・ボディのコンテンツがレスポンス・オブジェクトに追加され、doAfterBody() メソッドがコールされることを JSP コンテナに警告します。

このクラスは、IterationTag インタフェースを実装する、タグ・ハンドラ・サポート・クラスの TagSupport を拡張します。

```

package oracle.taglib;

import java.util.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

/**
 * MyIteratorTag extends TagSupport. The TagSupport class in JSP 1.2 implements the
 * IterationTag
 */

public class MyIteratorTag extends TagSupport
{
    private Iterator iterator;
    private Collection _collection;

    public void setCollection(Collection collection)
    {
        this._collection = collection;
    }

    public int doStartTag() throws JspTagException
    {
        if (_collection == null)
        {
            throw new JspTagException("No collection with name "
                + _collection
                + " found");
        }

        iterator = _collection.iterator();
        if (iterator.hasNext())
        {
            pageContext.setAttribute("item", iterator.next());
            return EVAL_BODY_INCLUDE;
        }
        else
        {
            return SKIP_BODY;
        }
    }

    public int doAfterBody()
    {
        if (iterator.hasNext())
        {

```



```

        pageContext.setAttribute("item", iterator.next());
        return EVAL_BODY_AGAIN;
    }
    else
    {
        return SKIP_BODY;
    }
}
}

```

### タグ・ライブラリ・ディスクリプタのサンプル : exampleiterator.tld

次に、myIterator というタグを定義する TLD のサンプルを示します。このサンプルでは、TLD で <variable> 要素を使用してスクリプト変数を直接定義できる、JSP 1.2 の機能を利用しています。この TLD は、java.lang.Object 型のスクリプト変数 item を定義します (JSP 1.1 環境では、この操作を行うためにタグ補足情報クラスを使用する必要があります)。変数は、新規に宣言されます。

myIterator タグには、コレクションを指定するための collection 属性があります。この属性は必須で、実行時の式として設定できます。このタグでは、<body-content> の値として JSP が指定されています。これは、JSP トランスレータがボディ・コードを処理および変換する必要があることを示します。

JSP 1.2 の構文の場合は、JSP 1.2 のタグ・ライブラリの DTD パスを指定してください。

```

<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>iterate</short-name>
  <description>This tag lib implements new JSP 1.2 IterationTag
    interface</description>
  <tag>
    <name>myIterator</name>
    <tag-class>oracle.taglib.MyIteratorTag</tag-class>
    <body-content>JSP</body-content>
    <attribute>
      <name>collection</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <variable>
      <name-given>item</name-given>

```

```
<variable-class>java.lang.Object</variable-class>
<declare>true</declare>
<!-- default scope: nested -->
<description>Scripting Variable item</description>
</variable>
</tag>
</taglib>
```

## 例 : IterationTag インタフェースおよびタグ補足情報クラスの使用

この項では、loop というカスタム・タグの定義と使用の完全な例を示します。このカスタム・タグを使用して、タグ・ボディを指定の回数だけ反復します。このカスタム・タグでは、タグ補足情報クラスを使用してスクリプト変数を定義します。

この例には、次の内容が含まれます。

- タグを使用するページの JSP ソース・コード
- タグ・ハンドラ・クラスのソース・コード
- タグ補足情報クラスのソース・コード
- TLD

---

---

**注意：** ここで示すサンプル・コードでは、oracle.jsp.jml パッケージ内の拡張データ型を使用しています。詳細は、『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』を参照してください。

---

---

### JSP ページのサンプル : exampletag.jsp

次に、exampletag.jsp という JSP ページのサンプルを示します。このサンプルでは、loop タグを使用し、外部ループは 5 回、内部ループは 3 回実行するように指定します。

```
<%@ taglib uri="/WEB-INF/exampletag.tld" prefix="foo" %>
<% int num=5; %>
<br>
<pre>
<foo:loop index="i" count="<%=num%>">
body1here: i expr: <%=i%>
    i property: <jsp:getProperty name="i" property="value" />
  <foo:loop index="j" count="3">
body2here: j expr: <%=j%>
    i property: <jsp:getProperty name="i" property="value" />
    j property: <jsp:getProperty name="j" property="value" />
  </foo:loop>
</foo:loop>
</pre>
```

## タグ・ハンドラ・クラスのサンプル : ExampleLoopTag.java

この項では、ExampleLoopTag というタグ・ハンドラ・クラスのソース・コードを示します。次の点に注意してください。

- このタグ・ハンドラ・クラスは、IterationTag インタフェースを実装する標準の TagSupport クラスを拡張します。
- doStartTag() メソッドは、整数の EVAL\_BODY\_INCLUDE を戻します。したがって、タグ・ボディ（ここではループ）が処理されます。
- ループ内を通過するたびに、doAfterBody() メソッドはカウンタを1ずつ増分します。このメソッドは、反復回数が残っている場合は EVAL\_BODY\_AGAIN を返し、最後の反復を終了した場合は SKIP\_BODY を戻します。
- このクラスでは、doEndTag() メソッドが定義されません。TagSupport の基礎となる実装が使用されます。

次に、コード例を示します。

```
package examples;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.util.Hashtable;
import java.io.Writer;
import java.io.IOException;
import oracle.jsp.jml.JmlNumber;

public class ExampleLoopTag
    extends TagSupport
{
    String index;
    int count;
    int i;
    JmlNumber ib;

    public ExampleLoopTag() {
        resetAttr();
    }

    public void release() {
        resetAttr();
    }

    private void resetAttr() {
        index=null;
        count=0;
        i=0;
    }
}
```

```
        ib=null;
    }

    public void setIndex(String index)
    {
        this.index=index;
    }
    public void setCount(String count)
    {
        this.count=Integer.parseInt(count);
    }

    public int doStartTag() throws JspException {
        ib=new JmlNumber();
        pageContext.setAttribute(index, ib);
        i++;
        ib.setValue(i);
        return EVAL_BODY_INCLUDE;
    }

    public int doAfterBody() throws JspException {
        if (i >= count) {
            return SKIP_BODY;
        } else
            pageContext.setAttribute(index, ib);
        i++;
        ib.setValue(i);
        return EVAL_BODY_AGAIN;
    }
}
```

### タグ補足情報クラスのサンプル : ExampleLoopTagTEI.java

この項では、loop タグで使用するスクリプト変数を示すタグ補足情報クラスのソース・コードを示します。

VariableInfo インスタンスは、変数に次の内容を指定するように構成されます。

- 変数名は、index 属性に基づいています。
- 変数のタイプは oracle.jsp.jml.JmlNumber で、完全修飾クラス名として指定する必要があります。
- 変数は、JSP トランスレータによって新規に宣言されます。
- 変数のスコープは、NESTED です。

さらに、タグ補足情報クラスには、count 属性が有効かどうかを判断する isValid() メソッドがあります。属性の値は、整数であることが必要です。

```
package examples;

import javax.servlet.jsp.tagext.*;

public class ExampleLoopTagTEI extends TagExtraInfo {
    public VariableInfo[] getVariableInfo(TagData data) {
        return new VariableInfo[]
        {
            new VariableInfo(data.getAttributeString("index"),
                "oracle.jsp.jml.JmlNumber",
                true,
                VariableInfo.NESTED)
        };
    }

    public boolean isValid(TagData data)
    {
        String countStr=data.getAttributeString("count");
        if (countStr!=null) // for request-time case
        {
            try {
                int count=Integer.parseInt(countStr);
            }
            catch (NumberFormatException e)
            {
                return false;
            }
        }
        return true;
    }
}
```

### タグ・ライブラリ・ディスクリプタのサンプル : `exampletag.tld`

この項では、タグ・ライブラリの TLD を示します。次の例のライブラリは、`loop` という 1 つのタグのみで構成されています。

この TLD では、JSP 1.2 の構文に準拠し、`loop` タグに関して次のように指定します。

- タグ・ハンドラ・クラスは、`examples.ExampleLoopTag` です。
- タグ補足情報クラスは、`examples.ExampleLoopTagTEI` です。
- `body-content` の値には JSP を指定します。これは、JSP トランスレータがボディ・コードを処理および変換する必要があることを示します。
- `index` 属性と `count` 属性があり、両方とも必須です。`count` 属性には、リクエスト時の JSP 式を設定できます。

次に、TLD の例を示します。

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <description>
    A simple tab library for the examples
  </description>
  <!-- example tag -->
  <!-- for loop -->
  <tag>
    <name>loop</name>
    <tag-class>examples.ExampleLoopTag</tag-class>
    <tei-class>examples.ExampleLoopTagTEI</tei-class>
    <body-content>JSP</body-content>
    <description>for loop</description>
    <attribute>
      <name>index</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>count</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

## コンパイル時のタグ

JSP 仕様で説明されているように、標準のタグ・ライブラリでは、実行時サポート機能が使用されます。通常、タグ・ライブラリは移植可能で、特定の JSP コンテナを必要としません。

ただし、ベンダーは、JSP トランスレータのベンダー固有の機能を使用して、カスタム・タグをサポートすることもできます。このようなタグは、他のコンテナに移植できません。

通常は、実行時機能を使用する標準の移植可能なタグの開発をお勧めしますが、この項で後述するように、コンパイル時機能を使用するタグが適切な場合があります。

## コンパイル時と実行時の機能に関する考慮事項

JSP 仕様には、カスタム・タグ・ライブラリの実行時サポート機能についての説明があります。XML スタイルの TLD を使用してタグを指定するこの機能については、この章で前述しています。このモデルに準拠したタグ・ライブラリを作成および使用すると、通常、そのライブラリは、標準の JSP 環境に移植できます。

ただし、コンパイル時の実装を考慮する理由があります。

- コンパイル時の実装によって、効率的なコードを作成できます。
- コンパイル時の実装によって、開発者は変換時とコンパイル時にエラーを発見できるため、ユーザーの実行時にエラーが発生しません。

## JML ライブラリの JSP 実行時バージョンとコンパイル時バージョン

OC4J には、JSP Markup Language (JML) ライブラリと呼ばれる移植可能なタグ・ライブラリが用意されています。このライブラリでは、標準の JSP 1.2 の実行時機能が使用されません。

ただし、JML タグは、コンパイル時機能でもサポートされています。これは、実行時機能が導入された JSP 仕様 1.1 より前に、JSP 実装でタグが最初に導入されたためです。コンパイル時タグは、下位互換性を維持するために現在もサポートされています。

コンパイル時の実装の一般的なメリットとデメリットは、Oracle JML タグ・ライブラリにも適用されます。コンパイル時の JML 実装を使用する方がメリットが多い場合もあります。また、この実装にはいくつかの追加タグがあり、サポートされている追加の式の構文もあります。

JML ライブラリの実行時バージョンとコンパイル時バージョンに関しては、『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』を参照してください。





---

---

## JSP グローバリゼーション・サポート

OC4J JSP コンテナは、JSP 仕様に従って、標準のグローバリゼーション・サポート (National Language Support (NLS) と呼ばれます) を提供し、マルチバイト・パラメータ・エンコードをサポートしないサーブレット環境の拡張サポートも提供します。

ローカライズされたコンテンツの標準の Java サポートでは、テキストの内部表現を統一するため、Unicode を使用する必要があります。Unicode は、別のキャラクタ・セットに変換するためのベース・キャラクタ・セットとして使用されます。(Unicode のバージョンは、JDK のバージョンによって異なります。Sun 社の Javadoc で `java.lang.Character` クラスの Unicode のバージョンを確認できます。)

この章では、グローバリゼーションと国際化に対する JSP サポートについて、重要な事項を説明します。この章には、次の項目が含まれます。

- [コンテンツ・タイプの設定](#)
- [マルチバイト・パラメータ・エンコードに対する JSP サポート](#)

---

---

**注意：** Oracle Application Server のグローバリゼーション・サポートの詳細は、『Oracle Application Server 10g グローバリゼーション・ガイド』を参照してください。

---

---

## コンテンツ・タイプの設定

次の各項では、コンテンツ・タイプを JSP ページに静的または動的に指定する標準的な方法について説明します。また、IANA (Internet Assigned Numbers Authority) 以外のキャラクタ・セットを JSP ライター・オブジェクトに対して指定できる Oracle の拡張メソッドについても説明します。

- [page ディレクティブでのコンテンツ・タイプの設定](#)
- [コンテンツ・タイプの動的な設定](#)
- [JSP ライター・オブジェクトのキャラクタ・セットに対する Oracle の拡張機能](#)

### page ディレクティブでのコンテンツ・タイプの設定

page ディレクティブには、JSP のページ・ソース (変換時) またはレスポンス (実行時) の文字エンコードに影響を与える 2 つの属性 (`pageEncoding` と `contentType`) があります。`contentType` 属性は、MIME タイプのレスポンスにも影響を与えます。各属性の機能は、次のとおりです。

- `contentType` を使用すると、ページ・ソースとレスポンス、および MIME タイプのレスポンスの文字エンコードを設定できます。
- `pageEncoding` を使用すると、ページ・ソースの文字エンコードを設定できます。JSP 仕様 1.2 で追加されたこの属性によって、レスポンスの文字エンコードとは異なるページ・ソースの文字エンコードを設定できます。ただし、キャラクタ・セットを指定する `contentType` 属性がない場合、この設定はレスポンスの文字エンコードに対するデフォルトになります。

`contentType` と `pageEncoding` の関連の詳細は、この項で後述します。

`contentType` には、次の構文を使用します。

```
contentType="TYPE; charset=character_set"
```

または、デフォルトのキャラクタ・セットを使用する場合は、次のように、MIME タイプを設定します。

```
contentType="TYPE"
```

`pageEncoding` には、次の構文を使用します。

```
pageEncoding="character_set"
```

すべての属性を設定するには、次の構文を使用します。

```
<%@ page ... contentType="TYPE; charset=character_set"  
    pageEncoding="character_set" ... %>
```

TYPE は IANA の MIME タイプで、character\_set は IANA のキャラクタ・セットです。contentType 属性を使用してキャラクタ・セットを指定する場合、セミコロンの後の空白はオプションです。

次に、contentType と pageEncoding の設定例を示します。

```
<%@ page language="java" contentType="text/html" %>
```

または

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>
```

または

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="US-ASCII" %>
```

page ディレクティブ設定がない場合のデフォルト設定は、次のとおりです。

- 従来の JSP ページに対するデフォルトの MIME タイプは text/html で、JSP XML 文書に対するデフォルトは text/xml です。
- ページ・ソースの文字エンコード (変換時) のデフォルトは、従来の JSP ページの場合は ISO-8859-1 (Latin-1 と呼ばれます)、JSP XML 文書の場合は UTF-8 または UTF-16 です。
- レスポンスの文字エンコードのデフォルトは、従来の JSP ページの場合は ISO-8859-1、JSP XML 文書の場合は UTF-8 または UTF-16 です。

UTF-8 と UTF-16 のいずれに設定するかは、<http://www.w3.org/TR/REC-xml.html> のサイトにある XML 仕様の「Autodetection of Character Encodings」に従います。

ただし、文字エンコードに関しては、pageEncoding と contentType には関連がありません。次の表を参照してください。

|                          | contentType エンコードが指定されている場合   | contentType エンコードが指定されていない場合   |
|--------------------------|---|--|
| pageEncoding が指定されている場合  | ページ・ソースのエンコードは pageEncoding に従います。<br>レスポンスのエンコードは contentType に従います。 | ページ・ソースのエンコードは pageEncoding に従います。<br>レスポンスのエンコードは pageEncoding に従います。 |
| pageEncoding が指定されていない場合 | ページ・ソースのエンコードは contentType に従います。<br>レスポンスのエンコードは contentType に従います。  | ページ・ソースのエンコードはデフォルトに従います。<br>レスポンスのエンコードはデフォルトに従います。                   |

使用方法については、次の重要な点に注意してください。

- `contentType` または `pageEncoding` を設定する `page` ディレクティブは、JSP ページ内のできるかぎり前の位置にある必要があります。
- ページが JSP XML 文書である場合、`pageEncoding` 設定は無視されます。かわりに、JSP コンテナは、その文書の XML エンコード宣言を使用します。次に例を示します。

```
<?xml version="1.0" encoding="EUC-JP" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="1.2">
<jsp:directive.page contentType="text/html;charset=Shift_Jis" />
<jsp:directive.page pageEncoding="UTF-8" />
...
```

有効なページ・エンコードは EUC-JP であり、UTF-8 ではありません。

- `pageEncoding` は、バイト・シーケンスがターゲット・キャラクタ・セット内の正当な文字を表しているページに対してのみ使用してください。
- `contentType` は、バイト・シーケンスがターゲット・キャラクタ・セット内の正当な文字を表しているページまたはレスポンス出力に対してのみ使用してください。
- たとえば、`contentType` で指定されているレスポンス出力のターゲット・キャラクタ・セットは、ページ・ソースのキャラクタ・セットに対するスーパーセットである必要があります。たとえば、UTF-8 は Big5 のスーパーセットですが、ISO-8859-1 はスーパーセットではありません。
- `page` ディレクティブのパラメータは静的です。レスポンスに対して異なるキャラクタ・セット仕様が必要であることが実行時に判明した場合、ページでは次のいずれかの方法で対処します。
  - サーブレットのレスポンス・オブジェクト API を使用して、実行中にコンテンツ・タイプを設定します。詳細は、9-5 ページの「[コンテンツ・タイプの動的な設定](#)」を参照してください。

または

- リクエストを別の JSP ページまたはサーブレットに転送します。
- JSP XML 文書ではなく、ISO-8859-1 以外のキャラクタ・セットで記述された従来の JSP ページ・ソースの場合は、適切なキャラクタ・セットを、`contentType` 属性または `pageEncoding` 属性を使用して `page` ディレクティブに設定する必要があります。JSP コンテナは変換時に設定を認識している必要があるため、ページ・エンコードのキャラクタ・セットは動的に設定できません。
- このマニュアルでは、わかりやすくするために、ページ・テキスト、リクエスト・パラメータおよびレスポンス・パラメータに、すべて同じエンコードを使用する一般的な場合を前提にしています。ただし、技術的には異なるエンコードも可能です。リクエスト・パラメータのエンコードは、ブラウザで制御されます。ただし、Netscape ブラウザと Internet Explorer ブラウザは、レスポンス・パラメータに指定した設定に従います。

IANA は、次のサイトで MIME タイプのレジストリを維持します。

<ftp://www.isi.edu/in-notes/iana/assignments/media-types/media-types>

IANA は、次のサイトで文字エンコードのレジストリを維持します。リストに「preferred MIME name」（推奨 MIME 名）と示されている場合は、それを使用します。

<http://www.iana.org/assignments/character-sets>

9-6 ページの「[JSP ライター・オブジェクトのキャラクタ・セットに対する Oracle の拡張機能](#)」で説明されている Oracle の追加拡張機能を除き、IANA リストのキャラクタ・セットのみを使用してください。

## コンテンツ・タイプの動的な設定

HTTP レスポンス用の適切なコンテンツ・タイプが実行時まで判明しない場合は、JSP ページでコンテンツ・タイプを動的に設定できます。標準の `javax.servlet.ServletResponse` インタフェースは、このために次のメソッドを指定します。

```
void setContentType(java.lang.String contentType)
```

---

---

**重要：** OC4J 環境でコンテンツ・タイプの動的設定を使用するには、JSP の `static_text_in_chars` 構成パラメータを有効にする必要があります。詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

---

---

JSP ページの暗黙的な `response` オブジェクトは

`javax.servlet.http.HttpServletResponse` インスタンスです。ここでは、`HttpServletResponse` インタフェースによって `ServletResponse` インタフェースが拡張されます。

page ディレクティブの `contentType` 設定のように、`setContentType()` メソッドの入力には、MIME タイプのみ、またはキャラクタ・セットと MIME タイプの両方を含めることができます。例：

```
response.setContentType("text/html; charset=UTF-8");
```

または

```
response.setContentType("text/html");
```

page ディレクティブと同様に、デフォルトの MIME タイプは、従来の JSP ページの場合は `text/html`、JSP XML 文書の場合は `text/xml` で、デフォルトの文字エンコードは ISO-8859-1 です。

JspWriter オブジェクトへの出力を記述する前に、ページのできるかぎり前の位置にコンテンツ・タイプを設定します。

setContentTypes() メソッドは、変換時に、JSP ページのテキストの解析に影響を与えません。変換時に特定のキャラクタ・セットが必要な場合は、page ディレクティブに指定する必要があります。詳細は、9-2 ページの「[page ディレクティブでのコンテンツ・タイプの設定](#)」を参照してください。

---

**注意：** OC4J などのサーブレット 2.2 以上の環境では、response オブジェクトには setLocale() メソッドがあります。このメソッドは、java.util.Locale オブジェクトを入力として取得し、指定のロケールに基づいてキャラクタ・セットを設定します。たとえば、次のメソッドのコールを実行すると、キャラクタ・セットは Shift\_JIS に設定されます。

```
response.setLocale(new Locale("ja", "JP"));
```

キャラクタ・セットの動的仕様の場合は、setContentTypes() または setLocale() への最新のコールが優先されます。

---

## JSP ライター・オブジェクトのキャラクタ・セットに対する Oracle の拡張機能

標準的な使用方法では、response オブジェクトのコンテンツ・タイプのキャラクタ・セットは、page ディレクティブの contentType パラメータまたは response.setContentTypes() メソッドで決定され、自動的に JSP ライター・オブジェクトのキャラクタ・セットになります。JSP ライター・オブジェクトは、javax.servlet.jsp.JspWriter インスタンスです。

ただし、一部のキャラクタ・セットは、IANA で認識されていないため、標準のコンテンツ・タイプ設定で使用できません。このため、OC4J には、oracle.jsp.util.PublicUtil クラスの静的な setWriterEncoding() メソッドが用意されています。

```
static void setWriterEncoding(JspWriter out, String encoding)
```

このメソッドによって、JSP ライターのキャラクタ・セットを直接指定し、response オブジェクトのキャラクタ・セットをオーバーライドできます。次の例では、コンテンツ・タイプのキャラクタ・セットとして Big5 を使用しますが、JSP ライターのキャラクタ・セットとして、Big5 の非 IANA 香港語キャラクタ・セットである MS950 を指定します。

```
<%@ page contentType="text/html; charset=Big5" %>  
<% oracle.jsp.util.PublicUtil.setWriterEncoding(out, "MS950"); %>
```

---

---

**注意:** `setWriterEncoding()` メソッドは、JSP ページ内のできるかぎり前の位置で使用してください。

---

---

## マルチバイト・パラメータ・エンコードに対する JSP サポート

サーブレット仕様では、`javax.servlet.ServletRequest` インタフェースに `setCharacterEncoding()` メソッドがあります。このメソッドは、サーブレット・コンテナのデフォルトのエンコードが、マルチバイトのリクエスト・パラメータと Bean プロパティの設定 (Java コード内の `getParameter()` コールや JSP コードに Bean プロパティを設定する `jsp:setProperty` タグなど) に対して適切でない場合に役に立ちます。

`setCharacterEncoding()` メソッドと Oracle の同等の拡張機能は、特に次の場合に、パラメータの名前と値に影響を与えます。

- リクエスト・オブジェクト `getParameter()` メソッドの出力
- リクエスト・オブジェクト `getParameterValues()` メソッドの出力
- リクエスト・オブジェクト `getParameterNames()` メソッドの出力
- Bean プロパティ値に対する `jsp:setProperty` の設定

この項には、次の項目が含まれます。

- [標準の `setCharacterEncoding\(\)` メソッド](#)
- [従来のサーブレット環境に対する Oracle の拡張機能の概要](#)

### 標準の `setCharacterEncoding()` メソッド

サーブレット仕様 2.3 以上では、`setCharacterEncoding()` メソッドは、HTTP リクエストの読取りに使用するデフォルト以外の文字エンコードを指定する標準機能として、`javax.servlet.ServletRequest` インタフェースに指定されています。このメソッドのシグネチャは、次のとおりです。

```
void setCharacterEncoding(java.lang.String enc)
    throws java.io.UnsupportedEncodingException
```

`enc` パラメータは、任意の文字エンコードの名前を指定する文字列で、デフォルトの文字エンコードをオーバーライドします。このメソッドは、リクエスト・パラメータを読み取る前、または `getReader()` メソッド (`ServletRequest` インタフェースに指定) を介して入力を読み取る前にコールしてください。

対応する次の `getter` メソッドもあります。

```
String getCharacterEncoding()
```

## 従来のサーブレット環境に対する Oracle の拡張機能の概要

サーブレット 2.3 より前の環境では、`setCharacterEncoding()` メソッドは使用できません。このような環境（特に、JServ サーブレット 2.0 環境）に対して、Oracle では次の 2 つの機能を用意しています。

- 静的な `oracle.jsp.util.PublicUtil.setRequestCharacterEncoding()` メソッド（推奨）
- `translate_params` 構成パラメータ（または同等のコード）

これらの機能の詳細は、B-24 ページの「[JServ でのマルチバイト・パラメータ・エンコード](#)」を参照してください。



---

---

# サーブレットと JSP の技術的な バックグラウンド

この付録では、サーブレットと JSP に関する技術的なバックグラウンドについて説明します。このマニュアルは、サーブレット・テクノロジーの知識があるユーザーを対象にしていますが、ここで説明する内容は、その知識を新たにするのに役立ちます。

また、生成した JSP ページ実装クラスによって自動的に実装される、標準の JSP インタフェースについても簡単に説明します。ただし、ほとんどのユーザーにはこの情報は必要ありません。

この付録には、次の項目が含まれます。

- [サーブレットに関するバックグラウンド](#)
- [Web アプリケーション階層](#)
- [標準の JSP インタフェースとメソッド](#)

---

---

**注意：** サーブレットと OC4J のサーブレット・コンテナの詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

---

---

## サーブレットに関するバックグラウンド

JSP ページは Java サーブレットに変換されるため、ここでは、サーブレット・テクノロジーについて簡単に説明します。ここで説明する概念の詳細は、Sun 社の Java サーブレット仕様を参照してください。

この項で説明するメソッドの詳細は、Sun 社の次のサイトで Javadoc を参照してください (サーブレット 2.2 とサーブレット 2.3)。

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

<http://java.sun.com/products/servlet/2.3/javadoc/index.html>

## サーブレット・テクノロジーの概要

近年、サーブレット・テクノロジーは、動的な HTML ページを介して Web サーバーの機能を拡張する強力な方法として急速に発展しました。サーブレットとは、Web サーバーで実行する Java プログラムです。これとは対照的に、アプレットは、クライアント・ブラウザで実行する Java プログラムです。サーブレットは、ブラウザから HTTP リクエストを取得して動的なコンテンツを生成し (データベースを問い合わせなどの方法で)、HTTP レスポンスをブラウザに戻します。

サーブレットが開発される前は、CGI (Common Gateway Interface) テクノロジーが動的なコンテンツに使用されていました。この CGI プログラムは、Perl などの言語で記述され、Web サーバーを介して Web アプリケーションによってコールされていました。しかし、CGI は、アーキテクチャとスケーラビリティに制限があったため、理想的なテクノロジーではありませんでした。

サーブレット・テクノロジーは、スケーラビリティの向上に加えて、Java のメリットであるオブジェクト指向、プラットフォームからの独立性、セキュリティおよび堅牢さも備えています。サーブレットでは、すべての標準 Java API を使用できます。これには、データベース・プログラマにとって特に重要な、Java データベースに接続するための JDBC API も含まれます。

Java の分野では、サーバー集中型アプリケーション (データベースにアクセスするアプリケーションなど) に関して、サーブレット・テクノロジーは、アプレット・テクノロジーよりも多くのメリットを提供します。このメリットの 1 つは、サーブレットが、一般的に多くのリソースを備えた堅牢なマシンであるサーバーで実行されるため、クライアントのリソースの使用を最小限に抑えることができることです。これに対して、アプレットはクライアントのブラウザにダウンロードされ、クライアントで実行されます。もう 1 つのメリットは、データに直接アクセスできることです。サーブレットが実行される Web サーバーは、アクセスするデータと同じネットワーク・ファイアウォール側に存在します。クライアント・マシンで実行されるアプレットは、ファイアウォールの外側に存在するため、アプレットのダウンロード元ではないサーバーにアプレットがアクセスするには、特別な方法 (署名付きアプレットなど) が必要です。

## サーブレットのインタフェース

Java サーブレットは、標準の `javax.servlet.Servlet` インタフェースを実装するように定義されています。このインタフェースは、サーブレットの初期化、リクエストの処理、サーブレットの構成などの基本情報の取得、およびサーブレット・インスタンスの終了を行うメソッドを指定します。

Web アプリケーションの場合、`Servlet` インタフェースは、標準の `javax.servlet.http.HttpServlet` 抽象クラスを拡張することによって実装できます。`HttpServlet` クラスには、次のメソッドが含まれます。

- `init(...)` および `destroy(...)`: それぞれ、サーブレットの初期化メソッドおよび終了メソッド。
- `doGet(...)`: HTTP GET リクエスト用メソッド。
- `doPost(...)`: HTTP POST リクエスト用メソッド。
- `doPut(...)`: HTTP PUT リクエスト用メソッド。
- `doDelete(...)`: HTTP DELETE リクエスト用メソッド。
- `service(...)`: HTTP リクエストを受信するためのメソッド。デフォルトでは、リクエストを適切な `doXXX()` メソッドにディスパッチします。
- `getServletInfo(...)`: サーブレット自体の情報を提供するため、サーブレットで使用するメソッド。

`HttpServlet` を拡張するサーブレット・クラスは、これらのメソッドの中から必要なメソッドを実装する必要があります。各メソッドは、標準の `javax.servlet.http.HttpServletRequest` インスタンスおよび標準の `javax.servlet.http.HttpServletResponse` インスタンスを入力として取得します。

`HttpServletRequest` インスタンスは、HTTP リクエストに関する情報（リクエスト・パラメータの名前と値、リクエストを送信したリモート・ホストの名前、リクエストを受信したサーバーの名前など）をサーブレットに提供します。`HttpServletResponse` インスタンスは、コンテンツ長と MIME タイプの指定、出力ストリームの提供など、レスポンスを送信するときの HTTP 固有の機能を提供します。

## サーブレット・コンテナ

サーブレット・コンテナはサーブレット・エンジンとも呼ばれ、サーブレットを実行および管理します。通常、サーブレット・コンテナは Java で記述され、Web サーバーに組み込まれます (Web サーバーが Java で記述されている場合)。そうでない場合は、Web サーバーに関連付けられ、その Web サーバーで使用されます。

サーブレットがコールされると (サーブレットが URL によって指定された場合など)、Web サーバーは HTTP リクエストをサーブレット・コンテナに渡します。次に、コンテナは、そのリクエストをサーブレットに渡します。サーブレットの管理を実行する間に、単純なコンテナは、次の処理を実行します。

- サーブレットのインスタンスを作成し、`init()` メソッドをコールしてそのインスタンスを初期化します。
- サーブレットの `service()` メソッドをコールします。
- 必要に応じて、サーブレットの `destroy()` メソッドをコールし、インスタンスを破棄します。これによって、そのインスタンスはガベージ・コレクションの対象になります。

パフォーマンス上の理由から、サーブレット・コンテナは通常、サーブレット・インスタンスをタスクの完了ごとに破棄せず、メモリー内に保持して再利用します。インスタンスを破棄するのは、Web サーバーの停止など、頻度の低いイベントの場合のみです。

サーブレットの実行中に別のサーブレット・リクエストが発生した場合のサーブレット・コンテナの動作は、そのサーブレットがシングル・スレッド・モデルを使用しているか、マルチスレッド・モデルを使用しているかによって異なります。シングル・スレッドの場合、サーブレット・コンテナは、同時にコールされた複数の `service()` メソッドが、単一のサーブレット・インスタンスにディスパッチされないようにします。この場合は、複数のサーブレット・インスタンスを個別に起動します。マルチスレッド・モデルの場合、コンテナは、単一のサーブレット・インスタンスに対して複数の `service()` メソッドを同時にコールできます (コールごとに個別のスレッドを使用します)。ただし、サーブレット開発者は同期を管理する責任があります。

## サーブレット・セッション

サーブレットは、HTTP セッションを使用して、HTTP リクエストを行ったユーザーを追跡します。これによって、単一ユーザーからの複数のリクエストをステートフルな方法で管理できます。サーブレット・セッション・トラッキングは、CGI など以前のテクノロジーの HTTP セッション・トラッキングと類似しています。

### HttpSession インタフェース

標準のサーブレット API では、各ユーザーは、標準の `javax.servlet.http.HttpSession` インタフェースを実装するクラスのインスタンスで表されます。サーブレットは、セッションに関する情報をこの `HttpSession` オブジェクトに設定し（スコープは必ずアプリケーション・レベルを設定）、ここから情報を取得できます。

サーブレットは、`HttpServletRequest` オブジェクト（HTTP リクエストを表します）の `getSession()` メソッドを使用して、ユーザー用の `HttpSession` オブジェクトを取得または作成します。このメソッドは、セッション・オブジェクトがない場合、ブール型の引数を取得して、ユーザー用に新規のセッション・オブジェクトを作成する必要があるかどうかを指定します。

`HttpSession` インタフェースは、次のメソッドを指定して、セッション情報の取得および設定を行います。

- `public void setAttribute(String name, Object value)`  
このメソッドは、指定のオブジェクトを指定の名前でセッションにバインドします。
- `public Object getAttribute(String name)`  
このメソッドは、指定の名前でセッションにバインドされたオブジェクトを取得します。一致するオブジェクトがない場合は、`null` を取得します。

---

**注意：** 以前のサーブレット実装では、`setAttribute()` および `getAttribute()` のかわりに、同じシグネチャを持つ `putValue()` および `getValue()` が使用されます。

---

サーブレット・コンテナとサーブレット自体の実装に応じて、セッションは、設定された時間の経過後に自動的に終了するか、サーブレットによって明示的に無効にすることができます。サーブレットは、`HttpSession` インタフェースで指定した次のメソッドを使用して、セッションのライフ・サイクルを管理できます。

- `public boolean invalidate()`  
このメソッドは、セッションを即時に無効にして、オブジェクトをセッションからバインド解除します。

- `public boolean setMaxInactiveInterval(int interval)`  
このメソッドは、タイムアウト時間を秒単位の整数で設定します。
- `public boolean isNew()`  
このメソッドは、セッションを作成したリクエスト内では `true` を戻し、そうでない場合は `false` を戻します。
- `public boolean getCreationTime()`  
このメソッドは、セッション・オブジェクトの作成時間を、1970年1月1日午前0時以降の経過時間としてミリ秒単位で戻します。
- `public boolean getLastAccessedTime()`  
このメソッドは、クライアントによる最新のリクエストの時間を、1970年1月1日午前0時以降の経過時間としてミリ秒単位で戻します。

### セッション・トラッキング

`HttpSession` インタフェースは、セッションの追跡に関するいくつかの機能をサポートします。各機能には、セッション ID を割り当てる方法が含まれます。セッション ID は、サーブレット・コンテナが割り当て、使用する中間ハンドルです。同じユーザーによる複数のセッションでは、必要に応じて、同じセッション ID を共有できます。

次のセッション・トラッキング機能がサポートされています。

- **Cookie**  
HTTP リクエストごとに、サーブレット・コンテナは **Cookie** をクライアントに送信し、クライアントはその **Cookie** をサーバーに戻します。これによって、**Cookie** が示すセッション ID にリクエストが関連付けられます。この機能は最も頻繁に使用され、サーブレット仕様 2.2 以上に準拠するサーブレット・コンテナでサポートされます。
- **URL 書換え**  
サーブレット・コンテナは、次の例に示すようにセッション ID を URL パスに追加します。  
`http://host:port/myapp/index.html?jsessionid=6789`  
この機能は、クライアントが **Cookie** を受け取らない場合に最も頻繁に使用されます。

## サーブレット・コンテキスト

サーブレット・コンテキストは、単一の JVM 内にある、1 つの Web アプリケーションの全インスタンス（つまり、Web アプリケーションの一部であるサーブレットと JSP ページの全インスタンス）に関する状態情報を維持するために使用します。これは、サーバー上の単一のクライアントに関する状態情報をセッションで維持する方法に類似していますが、サーブレット・コンテキストは、単一ユーザーに対して固有ではなく、複数のクライアントを処理できます。通常は、指定の JVM 内で実行される Web アプリケーションごとに、1 つのサーブレット・コンテキストが存在します。サーブレット・コンテキストは、アプリケーション・コンテナと考えることができます。

サーブレット・コンテキストは、標準の `javax.servlet.ServletContext` インタフェースを実装するクラスのインスタンスです。このようなクラスは、複数のサーブレットをサポートする Web サーバーに提供されます。

`ServletContext` オブジェクトによって、サーブレット環境に関する情報（サーバー名など）が提供され、単一の JVM 内では、グループ内の複数サーブレット間でリソースを共有できます（複数の同時 JVM をサポートするサーブレット・コンテナの場合は、リソース共有の実装が異なります）。

サーブレット・コンテキストは、アプリケーションを実行するユーザーのセッション・オブジェクトを維持し、そのアプリケーションの実行インスタンスにスコープを提供します。この機能によって、各アプリケーションが固有のクラス・ローダーからロードされ、その実行時オブジェクトが、他のアプリケーションの実行時オブジェクトと区別されます。特に、`HttpSession` がアプリケーションのユーザーごとに固有であるのと同様に、`ServletContext` オブジェクトはアプリケーションに対して固有です。

サーブレット仕様 2.2 以上では、ほとんどの実装で、単一のホスト内で複数のサーブレット・コンテキストを提供できます。これによって、各 Web アプリケーションは、独自のサーブレット・コンテキストを使用できます（以前の実装では、通常、特定のホストに対して 1 つのサーブレット・コンテキストのみ提供されていました）。

`ServletContext` インタフェースは、サーブレットが、そのサーブレットを実行しているサーブレット・コンテナと通信できるメソッドを指定します。これは、サーブレットがアプリケーション・レベルの環境と状態情報を取得できる方法の 1 つです。

---

---

**注意：** 以前のバージョンのサーブレット仕様では、サーブレット・コンテキストの概念が十分に定義されていませんでした。バージョン 2.1 (b) 以上では概念がより明確になり、HTTP セッション・オブジェクトが複数のサーブレット・コンテキスト・オブジェクト間で存在できないように指定されました。

---

---

## イベント・リスナーによるアプリケーションのライフ・サイクル管理

サーブレット仕様 2.2 で、標準の Java イベント・リスナー機能による限定的なアプリケーションのライフ・サイクル管理が初めて提供されました。HTTP セッション・オブジェクトでは、セッション・オブジェクト内のオブジェクトの追加または削除を認識するためにイベント・リスナーを使用できます。一般的に、セッション・オブジェクト内のオブジェクトを削除するのは、そのセッションが無効になった場合であるため、この機能によって、開発者はセッション・ベースでリソースを管理できます。同様に、イベント・リスナー機能によって、ページ・ベースおよびリクエスト・ベースのリソースも管理できます。

サーブレット仕様 2.3 では、イベント・リスナーに対する追加サポートが提供され、サーブレット・コンテキスト・ライフサイクル、サーブレット・コンテキスト属性、HTTP セッション・ライフサイクルおよび HTTP セッション属性の変更が通知されるイベント・リスナーに対して実装可能なインタフェースを定義しています。詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

## サーブレットの起動

HTML ページと同様に、サーブレットも URL を使用して直接起動できます。サーブレットは、Web サーバー実装でサーブレットが URL にマッピングされた方法に従って起動されます。次のマッピング方法があります。

- 特定の URL を特定のサーブレット・クラスにマッピングできます。
- ディレクトリ全体をマッピングできるため、ディレクトリ内の任意のクラスをサーブレットとして実行します。たとえば、`/servlet` というディレクトリをマッピングすると、`/servlet/servlet_name` の形式の URL でサーブレットを実行できます。
- ファイル名拡張子をマッピングできるため、ファイル（ファイル名にその拡張子を含む）を指定した URL でサーブレットを実行できます。

マッピング方法は、Web サーバー構成の一部として指定されます。OC4J では、`global-web-application.xml` ファイルの設定によってマッピング方法が決まります。

JSP ページと同様に、サーブレットも、`jsp:include` タグや `jsp:forward` タグを使用して間接的に起動できます。4-2 ページの「[JSP ページからのサーブレットの起動](#)」を参照。



## Web アプリケーション階層

Web アプリケーション（サーブレットと JSP ページの組合せで構成される）に関連するエンティティは、単純な階層構造ではありませんが、次の順序の階層と考えることができます。

### 1. サーブレット・オブジェクト（ページ実装オブジェクトを含む）

実行アプリケーションの各サーブレットと各 JSP ページ実装には、1 つのサーブレット・オブジェクトがあります（使用する実行モデルがシングル・スレッドかマルチスレッドかによって、複数のオブジェクトが存在する場合があります）。サーブレット・オブジェクトは、クライアントからのリクエスト・オブジェクトを処理し、レスポンス・オブジェクトをそのクライアントに戻します。JSP ページは、サーブレット・コードと同様に、レスポンス・オブジェクトの作成方法を指定します。

1 つのページまたはサーブレットが別のページまたはサーブレットにインクルードまたは転送された場合など、特定の状況での複数のサーブレット・オブジェクトは、1 つのリクエスト・オブジェクト内に存在しているサーブレット・オブジェクトと考えることができます。

通常、ユーザーは、セッション中に複数のサーブレット・オブジェクトにアクセスし、そのサーブレット・オブジェクトはセッション・オブジェクトに関連付けられています。

サーブレット・オブジェクトは、ページ実装オブジェクトと同様に、標準の `javax.servlet.Servlet` インタフェースを間接的に実装します。Web アプリケーション内のサーブレットの場合は、標準の `javax.servlet.http.HttpServlet` 抽象クラスをサブクラス化することで、このインタフェースを実装します。JSP ページ実装クラスの場合は、標準の `javax.servlet.jsp.HttpJspPage` インタフェースを実装することで、このインタフェースを実装します。

### 2. リクエスト・オブジェクトとレスポンス・オブジェクト

これらのオブジェクトは、ユーザーによるアプリケーションの実行によって生成された個別の HTTP リクエストと HTTP レスポンスを表します。

ユーザーは通常、セッション中に、複数のリクエストを生成し、複数のレスポンスを受け取ります。リクエスト・オブジェクトとレスポンス・オブジェクトは、セッション内に含まれているのではなく、セッションに関連付けられています。

クライアントから送信されたリクエストは、URL の仮想パスに従って、適切なサーブレット・コンテキスト・オブジェクト（クライアントが使用するアプリケーションに関連付けられたオブジェクト）にマッピングされます。仮想パスには、アプリケーションのルート・パスが含まれます。

リクエスト・オブジェクトは、標準の `javax.servlet.http.HttpServletRequest` インタフェースを実装します。

レスポンス・オブジェクトは、標準の `javax.servlet.http.HttpServletResponse` インタフェースを実装します。

### 3. セッション・オブジェクト

セッション・オブジェクトには、指定のセッションのユーザーに関する情報が格納され、複数のページ・リクエスト間で単一ユーザーを識別する方法を提供します。ユーザーごとに1つのセッション・オブジェクトが存在します。

任意の時点で、1つのサーブレットまたは JSP ページに対して複数のユーザーが存在できます。各ユーザーは、固有のセッション・オブジェクトで表されます。ただし、すべてのセッション・オブジェクトは、アプリケーション全体に対応するサーブレット・コンテキストによって維持されます。実際には、各セッション・オブジェクトは、共通のサーブレット・コンテキストに関連付けられた Web アプリケーションのインスタンスを表すと考えることができます。

通常、セッション・オブジェクトは、複数のリクエスト・オブジェクト、レスポンス・オブジェクト、およびページまたはサーブレット・オブジェクトを順に使用し、他のセッションは同じオブジェクトを使用しません。ただし、このセッション・オブジェクトは、これらのオブジェクトを実際には含んでいません。

特定ユーザーのセッションのライフ・サイクルは、そのユーザーの最初のリクエストから始まります。そのユーザー・セッションが終了すると（ユーザーがアプリケーションを終了したり、タイムアウトが発生した場合など）、ライフ・サイクルが終了します。

HTTP セッション・オブジェクトは、`javax.servlet.http.HttpSession` インタフェースを実装します。

---

---

**注意：** バージョン 2.1 (b) より前のサーブレット仕様では、1つのセッション・オブジェクトが複数のサーブレット・コンテキスト・オブジェクト間で存在することが可能でした。

---

---

### 4. サーブレット・コンテキスト・オブジェクト

サーブレット・コンテキスト・オブジェクトは、サーバー内の特定のパスに関連付けられます。このパスは、サーブレット・コンテキストに関連付けられたアプリケーションのモジュールに対するベース・パスで、アプリケーション・ルートと呼ばれます。

特定の JVM 内では、1つのアプリケーションの全セッションに対して、1つのサーブレット・コンテキスト・オブジェクトが存在し、アプリケーションを形成するサーブレットと JSP ページにサーバーから情報を提供します。また、他のアプリケーションから独立したセキュアな環境内では、サーブレット・コンテキスト・オブジェクトによって、複数のアプリケーション・セッションでデータを共有できます。

サーブレット・コンテナは、標準の `javax.servlet.ServletContext` インタフェースを実装するクラスを提供し、ユーザーがアプリケーションを最初にリクエストしたときにこのクラスをインスタンス化して、この `ServletContext` オブジェクトにアプリケーションの場所を示すパス情報を提供します。

サーブレット・コンテキスト・オブジェクトには、通常、アプリケーションを同時に使用する複数のユーザーを表すためのセッション・オブジェクトのプールがあります。

サーブレット・コンテキストのライフ・サイクルは、対応するアプリケーションに対する（いずれかのユーザーからの）最初のリクエストから始まります。ライフ・サイクルが終了するのは、サーバーが停止または終了した場合のみです。

サーブレット・コンテキストの概要は、A-7 ページの「[サーブレット・コンテキスト](#)」を参照してください。

#### 5. サーブレット構成オブジェクト

サーブレット・コンテナは、サーブレットが初期化される時、サーブレット構成オブジェクトを使用して情報をサーブレットに渡します。Servlet インタフェースの `init()` メソッドは、サーブレット構成オブジェクトを入力として取得します。

サーブレット・コンテナは、標準の `javax.servlet.ServletConfig` インタフェースを実装するクラスを提供し、必要に応じてそのクラスをインスタンス化します。サーブレット・コンテキスト・オブジェクト（サーブレット・コンテナによってインスタンス化されます）は、サーブレット構成オブジェクトに含まれます。

## 標準の JSP インタフェースとメソッド

次の 2 つの標準インタフェースは、両方とも `javax.servlet.jsp` パッケージ内に存在し、JSP トランスレータによって生成されたコード内に実装するために使用できます。

- `JspPage`
- `HttpJspPage`

`JspPage` は、使用するプロトコルを特定しない汎用的なインタフェースです。このインタフェースは、`javax.servlet.Servlet` インタフェースを拡張します。

`HttpJspPage` は、HTTP プロトコルを使用する JSP ページ用のインタフェースです。このインタフェースは `JspPage` を拡張し、通常、JSP トランスレータによって生成されたサーブレット・クラスによって、直接かつ自動的に実装されます。

`JspPage` は、生成されたクラスのインスタンスを初期化したり終了するときに使用する、次のメソッドを指定します。

- `jspInit()`
- `jspDestroy()`

特別な初期化機能または終了機能が必要な場合は、次の例に示すように、JSP 宣言を指定して、関連のメソッドをオーバーライドする必要があります。

```
<%! void jspInit()  
    {  
        ...your implementation code...  
    }  
%>
```

HttpJspPage によって、次のメソッドの指定が追加されます。

- `_jspService()`

このメソッドのコードは、通常、トランスレータによって自動的に生成され、次の内容が含まれます。

- JSP ページのスクリプトレットからのコード
- JSP ディレクティブからのコード
- ページの静的なコンテンツ

(JSP ディレクティブは、スクリプトレット用の Java 言語の指定、パッケージのインポートの提供など、ページに関する情報を提供します。詳細は、1-7 ページの「[ディレクティブ](#)」を参照してください。)

Servlet メソッドと同様に、`_jspService()` メソッドは、`HttpServletRequest` インスタンスと `HttpServletResponse` インスタンスを入力として取得します。

JspPage インタフェースと HttpJspPage インタフェースは、Servlet インタフェースから次のメソッドを継承します。

- `init()`
- `destroy()`
- `service()`
- `getServletConfig()`
- `getServletInfo()`

Servlet インタフェースとその主要なメソッドの詳細は、A-3 ページの「[サーブレットのインタフェース](#)」を参照してください。

---

---

## Apache JServ 環境

Oracle Application Server で提供される主要な Web アプリケーション環境は、Oracle Application Server Containers for J2EE (OC4J) です。さらに、Oracle Application Server 10g (9.0.4) では、Apache JServ サブレット環境も用意されています（実際、JServ は、アプリケーション・サーバーの以前のリリースでも主要なサブレット環境でした）。

JServ 環境を下位互換性を維持するために使用していた場合は、サブレット 2.0 環境の場合と同様に、サブレットと JSP の使用に関する特別な考慮事項があります。この付録では、それらの考慮事項について次の各項で説明します。

- [JServ 環境での作業開始](#)
- [JServ 環境に関する考慮事項](#)
- [JServ に対する JSP アプリケーションとセッションのサポート](#)

---

---

**重要：** Oracle Application Server 10g (9.0.4) 以降では、JServ 環境は用意もサポートもされなくなります。JServ 環境を使用していたアプリケーションは、OC4J に移行する必要があります。これは Oracle の `globals.jsa` および `translate_params` 機能を使用していたアプリケーションに特に関係します。これらの機能からの移行の詳細は、B-43 ページの「[globals.jsa からの移行](#)」および B-31 ページの「[translate\\_params からの移行](#)」を参照してください。

---

---

## JServ 環境での作業開始

次の各項では、JSP ページを実行する JServ の構成方法について説明します。

- [Apache JServ Web Server のクラスパスへのファイルの追加](#)
- [JServ の JSP ファイル名拡張子のマッピング](#)
- [JServ の JSP 構成パラメータ](#)
- [JServ での JSP パラメータの設定](#)

### Apache JServ Web Server のクラスパスへのファイルの追加

ファイルを JServ 環境の Web サーバーのクラスパスに追加するには、JServ conf ディレクトリにある `jserv.properties` ファイルに、適切な `wrapper.classpath` コマンドを挿入します。そのクラスパスには、すでに `jsdk.jar` が存在する必要があります。Sun 社の JSDK 2.0 で提供されるこのファイルは、JServ に必要な `javax.servlet.*` パッケージのサーブレット 2.0 バージョンを提供します。また、JDK 環境用のファイルがクラスパスにすでに存在している必要があります。

次の例 (UNIX ディレクトリ・パスを使用する場合) には、JSP、JDBC および SQLJ のファイルが含まれています。`ORACLE_HOME` を、使用する Oracle ホームのパスに書き換えてください。

---

---

**注意：** ここに示すファイルの場所は例のため、変更される場合があります。インストールした場所を確認してください。

---

---

```
# servlet 2.0 APIs (required by JServ, from Sun JSDK 2.0):
wrapper.classpath=ORACLE_HOME/Apache/Jsdk/lib/jsdk.jar
#
# servlet 2.2 APIs (required and provided by OC4J):
wrapper.classpath=ORACLE_HOME/ojsp/lib/servlet.jar
# JSP packages:
wrapper.classpath=ORACLE_HOME/j2ee/home/lib/ojsp.jar
wrapper.classpath=ORACLE_HOME/j2ee/home/jsp/lib/taglib/ojsputil.jar
# XML parser (used for servlet 2.2 web deployment descriptor):
wrapper.classpath=ORACLE_HOME/lib/xmlparserv2.jar
# JDBC libraries for Oracle Database access (JDK 1.2.x environment):
wrapper.classpath=ORACLE_HOME/jdbc/lib/classes12.zip
# SQLJ translator (optional):
wrapper.classpath=ORACLE_HOME/sqlj/lib/translator.zip
# SQLJ runtime (optional) (for JDK 1.2.x enterprise edition):
wrapper.classpath=ORACLE_HOME/sqlj/lib/runtime12.zip
```

---

**重要:** `servlet.jar` (javax.servlet.\* パッケージのサーブレット 2.2 以上のバージョン用に OC4J で提供) が JServ 環境のクラスパスに存在している場合は、`jsdk.jar` を先行させる必要があります。

---

次の `useBean` コマンドの例を考えてみます。

```
<jsp:useBean id="queryBean" class="mybeans.JDBCQueryBean" scope="session" />
```

次の `wrapper.classpath` コマンドを、`jserv.properties` ファイルに追加できます (この例は、Windows NT 環境の場合です)。

```
wrapper.classpath=D:¥Apache¥Apache1.3.9¥beans¥
```

この結果、`JDBCQueryBean.class` は、次の場所に配置されます。

```
D:¥Apache¥Apache1.3.9¥beans¥mybeans¥JDBCQueryBean.class
```

## JServ の JSP ファイル名拡張子のマッピング

JServ 環境で、各 JSP ファイル名拡張子を `oracle.jsp.JspServlet` (JServ の JSP フロントエンド・サーブレット) にマッピングするには、`jserv.conf` ファイルまたは `mod_jserv.conf` ファイルにある `ApJServAction` コマンドが必要です。これらの構成ファイルは、JServ conf ディレクトリにあります。

(従来のバージョンでは、Apache conf ディレクトリの `httpd.conf` ファイルを更新する必要がありますがありました。現行のバージョンでは、`jserv.conf` ファイルまたは `mod_jserv.conf` ファイルは、実行時に `httpd.conf` にインクルードされます。どちらのファイルがインクルードされているかは、`httpd.conf` ファイルを確認してください。)

次に例を示します (UNIX 構文を使用する場合)。

```
# Map file name extensions (.sqljsp and .SQLJSP are optional).
ApJServAction .jsp /servlets/oracle.jsp.JspServlet
ApJServAction .JSP /servlets/oracle.jsp.JspServlet
ApJServAction .sqljsp /servlets/oracle.jsp.JspServlet
ApJServAction .SQLJSP /servlets/oracle.jsp.JspServlet
```

`oracle.jsp.JspServlet` に対するこのコマンドで使用するパスは、ファイル・システムにあるリテラルなディレクトリ・パスではありません。指定するパスは、JServ サーブレット構成 (サーブレット・ゾーンのマウント方法、ゾーン・プロパティ・ファイルの名前、サーブレットのリポジトリとして指定されたファイル・システム・ディレクトリ) によって異なります。「サーブレット・ゾーン」は JServ の用語で、その概念は「サーブレット・コンテキスト」に類似しています。詳細は、JServ のマニュアルを参照してください。

## JServ の JSP 構成パラメータ

この項では、Oracle JspServlet が JServ 環境用にサポートする構成パラメータについて説明します。

OC4J の JSP 構成パラメータの詳細は、3-14 ページの「[JSP 構成パラメータ](#)」を参照してください。

### JServ の構成パラメータのサマリー表

[表 B-1](#) は、元の Oracle JSP フロントエンド・サーブレットの `oracle.jsp.JspServlet` がサポートする構成パラメータのサマリーです。これは、JServ 環境で使用するフロントエンドです (OC4J では、フロントエンド・サーブレットの `oracle.jsp.runtimev2.JspServlet` が使用されます)。表には、各パラメータについて次の項目が記載されています。

- OC4J の構成パラメータとの対応 (ある場合)
- 事前変換に使用する同等または関連の `ojspc` オプション
- オプションの簡単な説明
- デフォルト値
- 使用時期がコンパイル時か実行時か

---

---

#### 注意：

- 同等の `ojspc` オプションに関する情報は、7-12 ページの「[ojspc 事前変換ユーティリティ](#)」を参照してください。
  - OC4J が提供する `main_mode`、`precompile_check` および `static_text_in_chars` の各パラメータは、JServ 環境では使用できません。ただし、JServ の場合、デフォルトでは、静的なテキストは文字として出力されます。
- 
-



表 B-1 JSP 構成パラメータ (JServ 環境)

| パラメータ             | OC4J の<br>構成パラメータ<br>との関連            | 関連の<br>ojspc オプション                        | 説明   | デフォルト                | 実行時/<br>コンパイル<br>時 |
|-------------------|--------------------------------------|---|--|----------------------|--------------------|
| alias_translation | OC4J では該当<br>なし。                     | (なし)                                      | このブール値を true<br>に設定して、JSP ペー<br>ジ参照のためのディ<br>レクトリの別名変換<br>における JServ 制限に<br>対処します。   | false                | いずれか<br>一方         |
| bypass_source     | main_mode フ<br>ラグに移行、<br>justrun 設定。 | (なし)                                      | このブール値を true<br>に設定すると、JSP コ<br>ンテナは、.jsp ソー<br>スの FileNotFoundException<br>例外を無視します。<br>ソースが使用できな<br>い場合は、事前変換<br>およびコンパイルさ<br>れたコードを使用し<br>ます。 | false                | 実行時                |
| classpath         | OC4J では該当<br>なし。                     | -addclasspath (関連し<br>ていますが、機能は異<br>なります) | これは、JSP クラスを<br>ロードするためのク<br>ラスパスの追加エン<br>トリです。  | null<br>(追加パス<br>なし) | いずれか<br>一方         |
| debug_mode        | 同一                                   | (なし)                                      | このブール値を true<br>に設定すると、JSP コ<br>ンテナは、実行時に<br>例外が発生した場合、<br>スタック・トレース<br>を出力します。  | false                | 実行時                |
| developer_mode    | main_mode フ<br>ラグに移行。                | (なし)                                      | このブール値を<br>false に設定すると、<br>ページのリクエスト<br>時に、ページの再変<br>換とクラスの再ロー<br>ドが必要かどうかを<br>判断するためのタイ<br>ムスタンプのチェッ<br>クが行われません。                              | true                 | 実行時                |

表 B-1 JSP 構成パラメータ (JServ 環境) (続き)

| パラメータ                     | OC4J の<br>構成パラメータ<br>との関連 | 関連の<br>ojspc オプション | 説明  | デフォルト        | 実行時/<br>コンパイル<br>時 |
|---------------------------|---------------------------|--------------------|---|--------------|--------------------|
| emit_debuginfo            | 同一                        | -debug             | このブール値を true に設定すると、デバッグ用に、元の .jsp ファイルに対する行マップが生成されます。                             | false        | コンパイル時             |
| external_resource         | 同一                        | -extres            | このブール値を true に設定すると、JSP トランスレータによって、ページのすべての静的なコンテンツが、変換時に別の Java リソース・ファイルに格納されます。 | false        | コンパイル時             |
| external_resource_timeout | jsp-timeout の機能に相当。       | (なし)               | 外部リソースを解放するタイムアウト値を設定します (external_resource が有効な場合に関係します)。                          | 0 (タイムアウトなし) | 実行時                |
| extra_imports             | 同一                        | -extraImports      | このパラメータを使用すると、JSP のデフォルトより多いインポートが追加されます。   | null         | コンパイル時             |
| forgive_dup_dir_attr      | 同一                        | -forgiveDupDirAttr | このブール値を true に設定すると、単一の JSP 変換単位内で同じディレクティブ属性に重複する設定がある場合でも、JSP 1.2 の変換エラーを回避します。   | false        | コンパイル時             |
| javaccmd                  | 同一                        | -noCompile         | javac コマンドラインまたは別の Java コンパイラを指定する場合に使用します。このオプションを使用すると、コンパイラは別のプロセスで実行されます。       | null         | コンパイル時             |

表 B-1 JSP 構成パラメータ (JServ 環境) (続き)

| パラメータ                  | OC4J の<br>構成パラメータ<br>との関連 | 関連の<br>ojspc オプション    | 説明   | デフォルト | 実行時/<br>コンパイル<br>時 |
|------------------------|---------------------------|-----------------------|--|-------|--------------------|
| no_tld_xml_validate    | 同一                        | -noTldXmlValidate     | このブール値を true に設定すると、TLD の XML 妥当性チェックが実行されません。デフォルトでは、TLD の妥当性チェックが実行されます。   | false | コンパイル時             |
| old_include_from_top   | 同一                        | -oldIncludeFromTop    | このブール値を true に設定すると、Oracle9iAS リリース 2 より前の Oracle JSP の動作との下位互換性を維持するために、ネストされた include ディレクティブ内のページの場所がトップレベルのページと関連付けられます。 | false | コンパイル時             |
| reduce_tag_code        | 同一                        | -reduceTagCode        | このブール値を true に設定すると、カスタム・タグを使用するために生成されるコードのサイズがさらに縮小します。  | false | コンパイル時             |
| req_time_introspection | 同一                        | -reqTimeIntrospection | このブール値を true に設定すると、コンパイル時にイントロスペクションが実行できない場合、リクエスト時に JavaBean のイントロスペクションが有効になります。   | false | コンパイル時             |

表 B-1 JSP 構成パラメータ (JServ 環境) (続き)

| パラメータ                                       | OC4J の<br>構成パラメータ<br>との関連 | 関連の<br>ojspc オプション | 説明   | デフォルト | 実行時/<br>コンパイル<br>時 |
|---|---------------------------|--------------------|--|-------|--------------------|
| send_error                                  | OC4J では該当<br>なし。          | (なし)               | このブール値を true に設定すると、ファイルが検出されない場合は標準の「404」エラーが、コンパイル障害の場合は「500」エラーが出力されます (カスタマイズされたメッセージの出力ではありません)。          | false | 実行時                |
| session_sharing<br>(globals.jsa とともに<br>使用) | OC4J では該当<br>なし。          | (なし)               | globals.jsa を使用するアプリケーションの場合、このブール値を true に設定すると、JSP セッション・データが、基礎となるサーブレット・セッションに伝播されます。                      | true  | 実行時                |
| setProperty_onerr_continue                  | 同一                        | (なし)               | このブール値を true に設定すると、property="*" の場合、jsp:setProperty の使用時にエラーが発生したとき、リクエスト・パラメータの反復と対応する Bean プロパティの設定が継続されます。 | false | 実行時                |

表 B-1 JSP 構成パラメータ (JServ 環境) (続き)

| パラメータ              | OC4J の<br>構成パラメータ<br>との関連 | 関連の<br>ojspc オプション | 説明  | デフォルト | 実行時/<br>コンパイル<br>時 |
|--------------------|---------------------------|--------------------|---|-------|--------------------|
| sqljcmd            | 同一                        | -S                 | SQLJ のコマンドラインである sqlj オプションまたは代替の SQLJ トランスレータを別の JVM で実行するためのパラメータです。null に設定すると、OC4J が提供する Oracle SQLJ パッケージが、デフォルトのオプション設定で使用されます。 | null  | コンパイル時             |
| tags_reuse_default | 同一                        | (なし)               | このブール値は、JSP タグ・ハンドラのプーリングのデフォルト設定を指定しません (true はデフォルトで有効、または false はデフォルトで無効になります)。このデフォルト設定は、特定の JSP ページについてはオーバーライドできます。            | false | 実行時                |
| translate_params   | OC4J では該当なし。              | (なし)               | このブール値を true に設定すると、マルチバイト・エンコードを実行しないサーブレット・コンテナがオーバーライドされます。  | false | 実行時                |
| unsafe_reload      | OC4J では該当なし。              | (なし)               | このブール値を true に設定すると、JSP ページの再変換と再ロード時に、アプリケーションとセッションが再起動されません。   | false | 実行時                |

表 B-1 JSP 構成パラメータ (JServ 環境) (続き)

| パラメータ                 | OC4J の構成パラメータとの関連 | 関連の ojspc オプション | 説明   | デフォルト    | 実行時 / コンパイル時 |
|-----------------------|-------------------|-----------------|--|----------|--------------|
| well_known_taglib_loc | 同一                | (なし)            | 複数の Web アプリケーションで共有するタグ・ライブラリ JAR ファイルを配置するディレクトリを指定します。 | 後述の注意を参照 | コンパイル時       |
| xml_validate          | 同一                | -xmlValidate    | このブール値を true に設定すると、web.xml ファイルで XML 妥当性チェックが実行されます。    | false    | コンパイル時       |

**注意：** ORACLE\_HOME ディレクトリの下の次の場所が well\_known\_taglib\_loc のデフォルト値です。

```
j2ee/home/jsp/lib/taglib/
```

## JServ の構成パラメータの説明

この項では、JServ 環境での構成パラメータについて説明します。

### alias\_translation (ブール値、デフォルト: false)

このパラメータによって、Oracle JSP コンテナは、JServ でディレクトリの別名変換を処理するときの制限に対処できます。現在の制限については、B-22 ページの「[JServ ディレクトリの別名変換](#)」を参照してください。

ディレクトリの別名変換用のコマンドである httpd.conf が JServ サーブレット環境で正しく動作するためには、次の例に示すように、alias\_translation を true に設定します。

```
Alias /icons/ "/apache/apache139/icons/"
```

### bypass\_source (ブール値、デフォルト: false)

通常、JSP ページのリクエスト時に、Oracle JSP コンテナは、ページ実装クラスが見つかったとしても、対応する .jsp ソース・ファイルが見つからない場合、FileNotFoundException 例外をスローします。これは、JSP コンテナが、デフォルトで、ページ実装クラスの生成以降にページ・ソースが変更されたかどうかをチェックするためです。

このパラメータを true に設定すると、JSP コンテナは、ページ・ソースが見つからない場合でも、ページ実装クラスを処理および実行します。

`bypass_source` が有効な場合、コンテナは、再変換でソースが使用可能かどうか、そして必要かどうかをチェックします。ソースが必要かどうかを判断する要因の1つは、`developer_mode` パラメータの設定です。

---

---

**注意:** `bypass_source` オプションは、生成済みのクラスのみが存在し、ソースが存在しないデプロイ環境で役立ちます (関連する説明は、7-40 ページの「[バイナリ・ファイルのみのデプロイ](#)」を参照してください)。

---

---

**classpath** (完全修飾パス、デフォルト: null)

JSP ページの変換時、コンパイル時または実行時に使用するクラスパス・エントリを、JSP のデフォルトのクラスパスに追加します。

一般的に、Oracle JSP コンテナは、独自のクラスパス (この `classpath` パラメータによるエントリも含む)、システムのクラスパス、Web サーバーのクラスパス、ページ・リポジトリ、および JSP アプリケーションのルート・ディレクトリに対して相対的に事前定義された場所から、クラスをロードします。

`classpath` 設定に指定されたパスを使用してロードされるクラスは、システム・クラス・ローダーではなく、JSP クラス・ローダーによってロードされます。JSP 実行時に、JSP クラス・ローダーによってロードされたクラスは、システム・クラス・ローダーまたは他のクラス・ローダーによってロードされたクラスにアクセスできません (また、アクセスされません)。

---

---

**注意:**

- 実行時のクラス自動再ロードは、JSP クラスパス内のクラスにのみ適用されます。これには、`classpath` パラメータで指定したパスも含まれます。
  - ページの事前変換時には、`ojspc -addclasspath` オプションによって、様々な関連機能が提供されます。7-20 ページの「[ojspc のオプションの説明](#)」を参照。
- 
- 

**debug\_mode** (ブール値、デフォルト: false)

このフラグの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**developer\_mode** (ブール値、デフォルト: true)

このフラグを `false` に設定すると、Oracle JSP コンテナは、ページのリクエスト時に、ページ実装クラスのタイムスタンプと、`.jsp` ソース・ファイルのタイムスタンプを比較しません。`developer_mode` を `true` に設定すると、コンテナは、毎回、ページ実装クラスの生成以降にソースが変更されたかどうかをチェックします。変更されている場合、JSP ト

ランスレータはページを再変換します。developer\_mode を false に設定すると、JSP コンテナは、ページまたはアプリケーションに対する最初のリクエスト時のみチェックします。後続のリクエストでは、生成されたページ実装クラスを再実行するのみです。

このフラグは、JavaBeans、および JSP ページでコールされる他のサポート・クラスの動的なクラス再ロードにも影響を与えます。developer\_mode を true に設定すると、JSP コンテナは、このようなクラスが、JSP クラス・ローダーによるロード以降に変更されたかどうかをチェックします。

通常、developer\_mode は false に設定することをお勧めします。特に、コードの変更がなくパフォーマンスが重要な問題である本番環境では、この設定をお勧めします。

**emit\_debuginfo** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**external\_resource** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**external\_resource\_timeout** (int 型、デフォルト: 0 (タイムアウトなし))

external\_resource が有効な場合は、external\_resource\_timeout を使用して、JSP ページ・リソースのタイムアウト値 (秒単位) を指定できます。設定した時間内に JSP ページへのアクセスがなかった場合は、そのページの外部リソースが解放されます。外部リソースは、次回、ページがリクエストされたときに再ロードされます。

---

---

**注意:** OC4J では、jsp-timeout フラグによって、外部リソースがページ全体とともに解放されます。これは、global-web-application.xml ファイルまたは orion-web.xml ファイル内の <orion-web-app> 要素の属性です。3-28 ページの「[JSP の OC4J 構成パラメータ](#)」を参照。

---

---

**extra\_imports** (インポート・リスト、デフォルト: null)

3-9 ページの「[デフォルト・パッケージのインポート](#)」に説明されているように、Oracle JSP コンテナでは、各 JSP ページにインポートされるパッケージのデフォルト・リストが、Oracle9iAS リリース 2 (9.0.3) より前のリストと比べて小規模です。このデフォルト・リストは、JSP の仕様にに基づいています。ただし、extra\_imports 構成パラメータを使用して、追加インポート用のパッケージ名または完全修飾されたクラス名を指定することによって、コードの更新を回避できます。一般的な構文は、B-15 ページの「[JServ での JSP パラメータの設定](#)」を参照してください。次の例のように、名前のリストには引用符と空白のデリミタを使用する必要があります。



```
servlet.oracle.jsp.JspServlet.initArgs=extra_imports='java.util.* java.io.*'
```

**forgive\_dup\_dir\_attr** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**javaccmd** (コンパイラの実行可能ファイルとオプション、デフォルト: null)

JServ 環境では関連する `use_old_compiler` パラメータがないことを除いて、このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**no\_tld\_xml\_validate** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**old\_include\_from\_top** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**reduce\_tag\_code** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**req\_time\_introspection** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**send\_error** (ブール値、デフォルト: false)

このフラグを `true` に設定すると、Oracle JSP コンテナは、ファイルが見つからない場合は汎用の「404」メッセージを、コンパイル・エラーの場合は汎用の「500」メッセージを出力します。

これは、詳細な情報（見つからないファイルの名前など）を提供する、カスタマイズされたメッセージの出力とは対照的です。JServ などの一部の環境では、「404」または「500」メッセージの出力を設定すると、カスタマイズされたメッセージは出力できません。

**session\_sharing** (ブール値、デフォルト: true) (globals.jsa とともに使用)

サーブレット 2.0 環境で、アプリケーションに対して globals.jsa を使用すると、各 JSP ページでは固有の JSP セッション・ラッパーが使用されます。このラッパーは、サーブレット・コンテナによって提供される単一の全体的なサーブレット・セッション・オブジェクトに含まれています。

この状況で、session\_sharing パラメータを true (デフォルト) に設定すると、JSP セッション・データは、基礎となるサーブレット・セッションに伝播されます。これによって、アプリケーション内のサーブレットは、アプリケーション内の JSP ページのセッション・データにアクセスできます。

session\_sharing を false に設定すると (ほとんどの JSP 実装で標準的な動作が並行して実行されます)、JSP セッション・データはサーブレット・セッションに伝播されません。その結果、アプリケーション・サーブレットは、JSP セッション・データにアクセスできません。

**setProperty\_onerr\_continue** (ブール値、デフォルト: false)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**sqljcmd** (SQLJ トランスレータの実行可能ファイルとオプション、デフォルト: null)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**tags\_reuse\_default** (ブール値、デフォルト: false)

このパラメータの使用方法は JServ と OC4J で同じですが、JServ でのデフォルトは false です。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**translate\_params** (ブール値、デフォルト: false)

---

**注意:** translate\_params パラメータではなく、PublicUtil.setReqCharacterEncoding() メソッドを使用することをお勧めします。B-24 ページの「[setReqCharacterEncoding\(\) メソッド](#)」を参照。

---

このフラグを true に設定すると、マルチバイト (グローバル化・サポート) のリクエスト・パラメータまたは Bean プロパティ設定をエンコードしないサーブレット・コンテナがオーバーライドされます。この設定の場合、Oracle JSP コンテナは、リクエスト・パラメータと Bean プロパティ設定をエンコードします。false に設定すると、サーブレット・コンテナから未変更のパラメータが戻されます。

`translate_params` の機能と使用方法、およびこのパラメータを使用しない状況の詳細は、B-24 ページの「[JServ でのマルチバイト・パラメータ・エンコード](#)」を参照してください。

**unsafe\_reload** (ブール値、デフォルト: `false`)

デフォルトでは、Oracle JSP コンテナは、JSP ページが動的に再変換および再ロード (`.jsp` ソース・ファイルのタイムスタンプが、対応するページ実装クラスのタイムスタンプより新しい場合に実行されます) されるたびに、アプリケーションとセッションを再起動します。

このパラメータを `true` に設定すると、JSP コンテナは、動的な再変換および再ロードの後、アプリケーションを再起動しません。これによって、既存のセッションが無効になるのを防ぎます。デプロイ環境では、`true` 設定が適切です。開発環境では、`false` (デフォルト) 設定が適切です。

指定の JSP ページで `developer_mode` が `false` に設定されている場合 (この場合、JSP コンテナは、最初のリクエスト後に再変換を実行しません)、そのページに対する最初のリクエスト以降、このパラメータの影響はありません。

**well\_known\_taglib\_loc** (ディレクトリ・パス、デフォルト: `j2ee/home/jsp/lib/taglib/`)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

**xml\_validate** (ブール値、デフォルト: `false`)

このパラメータの使用方法は、JServ と OC4J で同じです。3-18 ページの「[JSP 構成パラメータの説明](#)」を参照。

## JServ での JSP パラメータの設定

JServ 環境の各 Web アプリケーションには、ゾーン・プロパティ・ファイルと呼ばれる独自のプロパティ・ファイルがあります。Apache 用語のゾーンは、基本的にサブレット・コンテキストと同じ意味です。

ゾーン・プロパティ・ファイルの名前は、ゾーンのマウント方法によって異なります (ゾーンとマウント方法については、[Apache JServ のマニュアル](#)を参照)。

JServ で JSP 構成パラメータを設定するには、アプリケーションのゾーン・プロパティ・ファイルに `JspServlet initArgs` プロパティを設定します。次に例を示します (UNIX 構文を使用する場合)。

```
servlet.oracle.jsp.JspServlet.initArgs=developer_mode=false,  
sqljcmd=sqlj -user=scott/tiger -ser2class=true,classpath=/mydir/myapp.jar
```

(これは、1 行が折り返されています。)

サブレット・パスの `servlet.oracle.jsp.JspServlet` も、ゾーンのマウント方法によって異なります。このパスは、リテラルなディレクトリ・パスではありません。

次の点に注意してください。

- 複数の `initArgs` コマンドを使用すると、結果が蓄積されるため、前のコマンドの結果がオーバーライドされます。たとえば、2つのコマンドを次の順序で使用すると仮定します。

```
servlet.oracle.jsp.JspServlet.initArgs=foo1=val1,foo2=val2
servlet.oracle.jsp.JspServlet.initArgs=foo1=val3
```

この2つのコマンドの組合せは、次の1つのコマンドと同じです。

```
servlet.oracle.jsp.JspServlet.initArgs=foo1=val3,foo2=val2
```

最初の2つのコマンドでは、`val3` 値によって `foo1` の `val1` 値がオーバーライドされますが、`foo2` 設定には影響ありません。

- `initArgs` パラメータはカンマをデリミタとして使用するため、パラメータ設定値にカンマは使用できません。ただし、空白と特殊文字（「=」など）は問題ありません。

## JServ での ojspc の使用

JServ 環境で `ojspc` を使用するには、別のコマンドが必要です。`ojspc` ではなく `ojspc_jserv` を使用します。これは、同じクラスである `oracle.jsp.tool.Jspc` を実行しますが、JServ に適切なクラスパスで設定されます。

JServ 環境では、静的なテキストがデフォルトの文字として出力されるため、`ojspc -staticTextInChars` オプションの効果はありません。このオプションは使用禁止にできません。

---

---

**重要：** `ojspc_jserv` を使用するには、`ojspc` と同様に、Sun 社の JDK (バージョン 1.1.8 以上) を使用する必要があり、クラスパスには `tools.jar` (JDK 2.0 以上の場合) または `classes.zip` (JDK 1.1.8 の場合) が存在している必要があります。

---

---

## JServ 環境に関する考慮事項

JServ 環境はサーブレット 2.0 環境であるため、この環境で JSP ページを実行する場合は、特に考慮する必要がある事項があります。サーブレット 2.2 以上の環境で使用できる重要な機能の一部は、サーブレット仕様 2.0 ではサポートされていません。

JSP ページに対して JServ 環境を構成する方法は、B-2 ページの「[JServ 環境での作業開始](#)」を参照してください。

次の各項では、JServ 環境に関する特別な考慮事項について説明します。

- [mod\\_jserv Apache Mod](#)
- [JServ でのアプリケーション・ルート・サポートに関する JSP コンテナ機能](#)
- [JServ でのアプリケーションとセッションのフレームワークの概要](#)
- [JServ における JSP とサーブレット間でのセッション共有](#)
- [JServ での動的なインクルードと転送](#)
- [JServ ディレクトリの別名変換](#)
- [JServ での JSP のセキュリティに関する考慮事項](#)
- [JServ でのマルチバイト・パラメータ・エンコード](#)

### mod\_jserv Apache Mod

Apache が提供する mod\_jserv コンポーネントは、HTTP リクエストを、中間層の JVM の JServ サーブレット・コンテナで実行される JSP ページまたはサーブレットに委任します。中間層環境での物理的なホストは、バック・エンドの Oracle Database インスタンスと同じ場合とそうでない場合があります。

mod\_jserv と中間層の JVM との間の通信には、TCP/IP 上の独自の Apache JServ プロトコル (AJP) が使用されます。mod\_jserv コンポーネントは、ロード・バランシングのために、リクエストを 1 つのプール内にある複数の JVM に委任できます。

mod\_jserv の構成情報は、Apache のマニュアルを参照してください。このマニュアルは、Oracle Application Server に同梱されています。

## JServ でのアプリケーション・ルート・サポートに関する JSP コンテナ機能

JServ などのサーブレット 2.0 環境は、単一のアプリケーション環境であるため、アプリケーション・ルート概念がありません。Web サーバーのドキュメント・ルートが実質的なアプリケーション・ルートです。デフォルトでは、Oracle Application Server の JServ 環境で実行中の JSP ページとサーブレットは、JServ が提供する Apache mod\_jserv モジュールを介してルーティングされ、Apache JServ のドキュメント・ルートを使用します。これは、通常は、.../htdocs というディレクトリです。さらに、httpd.conf 構成ファイルの alias 設定を使用して、仮想のドキュメント・ルートを指定することもできます。

ただし、Oracle JSP コンテナは、JServ 環境のドキュメント・ルートとアプリケーション・ルートに関する追加の機能を提供します。Oracle JSP の `globals.jsa` 機能を使用すると、ドキュメント・ルート下のディレクトリが、指定のアプリケーションのアプリケーション・ルートとして機能するように指定できます。この指定を行うには、`globals.jsa` ファイルをマーカーとして任意のディレクトリに格納します。詳細は、B-32 ページの「[globals.jsa 機能の概要](#)」を参照してください。

アプリケーション・ルート・ディレクトリは、次のいずれかの場所に格納できます。検索される順に示します。

1. アプリケーションのマッピング先 Web サーバー・ディレクトリ
2. Web サーバーのドキュメント・ルート・ディレクトリ
3. `globals.jsa` ファイルが格納されているディレクトリ

## JServ でのアプリケーションとセッションのフレームワークの概要

サーブレット仕様 2.0 では Web アプリケーション概念が十分に定義されていなかったため、JServ がサポートするサーブレット・コンテナごとのサーブレット・コンテキストは 1 つのみです。さらに、サーブレット・コンテナごとのセッション・オブジェクトも 1 つのみです。

ただし、Oracle JSP コンテナでは、JServ 環境で使用する特別なアプリケーション・フレームワークをサポートしています。このサポートを実現するために、アプリケーション・マーカーとして使用できる `globals.jsa` ファイルが用意されています。このファイルによって、アプリケーションごとに固有のサーブレット・コンテキストとセッション・オブジェクトを使用できます。

詳細は、B-33 ページの「[globals.jsa による固有のアプリケーションとセッション](#)」を参照してください。

## JServ における JSP とサーブレット間でのセッション共有

JServ 環境において、JSP ページとサーブレット間で HTTP セッション情報を共有するには、JServ 環境で Oracle JSP コンテナのフロントエンドとして動作するサーブレットの `oracle.jsp.JspServlet` が、JSP ページとセッションを共有するサーブレット（1 つまたは複数）と同じゾーン内に存在するように環境を構成する必要があります。詳細は、Apache のマニュアルを参照してください。

ゾーン設定が適切であることを検証するため、一部のブラウザでは、Cookie に対する警告を使用できます。Apache 環境では、Cookie 名にゾーン名が含まれます。

さらに、JServ などのサーブレット 2.0 環境で、アプリケーションに対して `globals.jsa` を使用すると、各 JSP ページでは固有の JSP セッション・ラッパーが使用されます。このラッパーは、サーブレット・コンテナによって提供される単一の全体的なサーブレット・セッション・オブジェクトに含まれています。

この状況で、JSP `session_sharing` 構成パラメータを `true`（デフォルト）に設定すると、JSP セッション・データは、基礎となるサーブレット・セッションに伝播されます。これによって、アプリケーション内のサーブレットは、アプリケーション内の JSP ページのセッション・データにアクセスできます。

`session_sharing` を `false` に設定すると（ほとんどの JSP 実装で標準的な動作が並行して実行されます）、JSP セッション・データはサーブレット・セッションに伝播されません。その結果、アプリケーション・サーブレットは、JSP セッション・データにアクセスできません。

関連情報は、次の各項も参照してください。

- B-32 ページ「[JServ に対する JSP アプリケーションとセッションのサポート](#)」
- B-4 ページ「[JServ の JSP 構成パラメータ](#)」
- B-15 ページ「[JServ での JSP パラメータの設定](#)」

## JServ での動的なインクルードと転送

JSP の動的なインクルード（`jsp:include` タグを使用）と転送（`jsp:forward` タグを使用）は、リクエスト・ディスパッチャ機能に依存しています。この機能は、サーブレット 2.2 以上の環境で提供されていますが、サーブレット 2.0 環境にはありません。

このため、Oracle JSP コンテナには、JSP ページから他の JSP ページへ、または JSP ページから JServ などのサーブレット 2.0 環境にある静的な HTML ファイルへの動的なインクルードと転送を可能にする拡張機能が用意されています。

ただし、サーブレット 2.0 環境用のこの機能では、サーブレットへの動的なインクルードと転送はできません。これは、サーブレットの実行が、JSP コンテナではなく JServ などのサーブレット・コンテナによって制御されるためです。

JServ のサーブレットに対してインクルードまたは転送を行う場合は、サーブレットに対するラッパーとして動作する JSP ページを作成できます。

次に、サーブレットとそのサーブレットに対するラッパーとして動作する JSP ページの例を示します。JServ 環境では、JSP ラッパー・ページにインクルードまたは転送することで、実質的にサーブレットへのインクルードまたは転送を行うことができます。

**サーブレットのコード** 次のサーブレット (TestServlet) にインクルードまたは転送すると仮定します。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet extends HttpServlet {

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        System.out.println("initialized");
    }

    public void destroy()
    {
        System.out.println("destroyed");
    }

    public void service
        (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><BODY>");
        out.println("TestServlet Testing");
        out.println("<H3>The local time is: "+ new java.util.Date());
        out.println("</BODY></HTML>");
    }
}
```



**JSP ラッパー・ページのコード** 前述のサーブレットに対して、次の JSP ラッパー (wrapper.jsp) を作成できます。

```
<!-- wrapper.jsp--wraps TestServlet for JSP include/forward -->
<%@ page isThreadSafe="true" import="TestServlet" %>
<%!
    TestServlet s=null;
    public void jspInit() {
        s=new TestServlet();
        try {
            s.init(this.getServletConfig());
        } catch (ServletException se)
        {
            s=null;
        }
    }
    public void jspDestroy() {
        s.destroy();
    }
%>
<% s.service(request,response); %>
```

サーブレット 2.0 環境で wrapper.jsp にインクルードまたは転送することは、サーブレット 2.2 以上の環境で TestServlet に直接インクルードまたは転送する場合と同じ結果になります。

---

---

**注意：**

- JSP ラッパー・ページで、isThreadSafe を true に設定するか、または false に設定するかは、元のサーブレットがスレッド・セーフかどうかによって決まります。
  - このような状況では、JSP ラッパー・ページを使用するかわりに、HTTP クライアント・コードを元の JSP ページ (jsp:include または jsp:forward 操作を実行する JSP ページ) に追加することもできます。標準の java.net.URL クラスのインスタンスを使用すると、元の JSP ページからサーブレットへの HTTP リクエストを作成できます。この場合、セッション・データまたはセキュリティ接続情報は共有できないことに注意してください。
- 
-

## JServ ディレクトリの別名変換

Apache JServ では、ディレクトリの別名変換をサポートしているため、`httpd.conf` 構成ファイルの `Alias` コマンドを使用して、仮想ディレクトリを作成できます。これによって、Web ドキュメントを、デフォルトのドキュメント・ルート・ディレクトリ以外の場所に格納できます。

次のサンプルの `httpd.conf` エントリについて考えてみます。

```
Alias /icons/ "/apache/apache139/icons/"
```

このコマンドによって、`icons` が `/apache/apache139/icons/` パスに対する別名として使用可能になります。この方法によって、たとえば、`/apache/apache139/icons/art.gif` というファイルには、次の URL でアクセスできます。

```
http://host:port/icons/art.gif
```

ただし、現在のところ、この機能はサブレットと JSP ページに対して正しく動作していません。これは、別名のディレクトリ下にあるファイルを処理するとき、Apache JServ の `getRealPath()` メソッドが正しい値を戻さないためです。

Oracle JSP コンテナは、Apache 固有の構成パラメータである `alias_translation` をサポートしています。このパラメータを `true` に設定すると、この制限に対処できます。デフォルト設定は `false` です。

また、`alias_translation` を `true` に設定すると、別名ディレクトリがアプリケーション・ルートになることに注意してください。したがって、ターゲット・ファイル名が「/」で始まる `jsp:include` タグまたは `jsp:forward` タグでは、必要なターゲット・ファイルの場所が別名ディレクトリに対して相対的に指定されます。

次の例では、`/private/foo` の下にあるすべての JSP ファイルと HTML ファイルが、実質的に、アプリケーション `/mytest` の下にあります。

```
Alias /mytest/ "/private/foo/"
```

JSP ページが次の場所にあると仮定します。

```
/private/foo/xxx.jsp
```

次の例では、`xxx.jsp` が別名ディレクトリ `/private/foo` (実質的なアプリケーション・ルート) のすぐ下にあるため、`jsp:include` タグは有効です。

```
<jsp:include page="/xxx.jsp" flush="true" />
```

他のアプリケーション内または一般的なドキュメント・ルート内の JSP ページは、`/mytest` アプリケーション内の JSP ページまたは HTML ファイルに転送またはインクルードできません。ページまたは HTML ファイルの転送または挿入は、サブレット仕様に従って、同じアプリケーション内でのみ可能です。

---

---

**注意:**

- 暗黙的なアプリケーションが、Web サーバーのドキュメント・ルートおよび各別名ルートに対して作成されます。
  - JServ での JSP 構成パラメータの設定方法は、B-15 ページの「[JServ での JSP パラメータの設定](#)」を参照してください。
- 
- 

また、2 つの別名が同じディレクトリ・パスで開始すると、問題となることに注意してください。次の 2 つの別名の例を考えてみます。

```
Alias /foo/bar1 "/path/to/my/dir/x/bar1"  
Alias /foo/bar2 "/path/to/my/dir/y/bar2"
```

/foo/bar1/bar1.jsp に対する最初のリクエストは機能しますが、/foo/bar2/bar2.jsp に対する後続のリクエストは、bar2.jsp の /path/to/my/dir/x を適切に検索しないため、FileNotFound 例外で障害が発生します。これは、JServ の getRealPath() 実装に伴う制限のためで、不適切な情報が戻ります。この場合は、次の対処方法があります。

- 実際の下位ディレクトリを使用して、別名を 1 つのみ指定します。

```
Alias /foo "/path/to/my/dir"
```

ここでは、bar1 および bar2 のディレクトリが、それぞれ /path/to/my/dir/bar1 および /path/to/my/dir/bar2 として物理的に存在するので、問題はありません。

または

- 複数の別名を指定します。ただし、物理的なディレクトリに別名ディレクトリと同じ名前がないように設定します。

```
Alias /foo/bar1 "/path/to/my/dir/x_bar1"  
Alias /foo/bar2 "/path/to/my/dir/y_bar2"
```

bar1 のかわりに x\_bar1 が、bar2 のかわりに y\_bar2 が使用されていることに注意してください。前述の問題のある例では、最初の別名に bar1 (ディレクトリ名と同じ)、2 番目の別名に bar2 (ディレクトリ名と同じ) を使用していました。

## JServ での JSP のセキュリティに関する考慮事項

JServ 環境では、セキュリティに関する次の考慮事項に注意してください。

- `_pages` ディレクトリへのアクセスは制限することをお勧めします。デフォルトの `_pages` ディレクトリへのアクセスは、デフォルトではすでに制限されています。さらに、別名を使用している場合は、各別名の下にある `_pages` ディレクトリへのアクセスも制限してください。`_pages` ディレクトリに関する一般情報は、7-6 ページの「[生成されるファイルとその格納場所](#)」を参照してください。
- `globals.jsa` ファイルが使用されている場所へのアクセスも制限することをお勧めします。アクセスは、デフォルトではすでに制限されています。`globals.jsa` の詳細は、B-32 ページの「[JServ に対する JSP アプリケーションとセッションのサポート](#)」を参照してください。

## JServ でのマルチバイト・パラメータ・エンコード

この項では、Java コードでの `getParameter()` コール、JSP コードで Bean プロパティを設定する `jsp:setProperty` タグなど、JServ などのサーブレット 2.0 環境での、マルチバイトのリクエスト・パラメータと Bean プロパティの設定をサポートする Oracle の拡張機能について説明します。次の 2 つの機能が用意されています。

- 静的な `oracle.jsp.util.PublicUtil.setReqCharacterEncoding()` メソッド (推奨)
- `translate_params` 構成パラメータ (または同等のコード)

`translate_params` の説明の後に、OC4J 環境への移行時にこのパラメータも移行する方法について説明します。

マルチバイト・パラメータ・エンコードの一般情報は、9-7 ページの「[マルチバイト・パラメータ・エンコードに対する JSP サポート](#)」を参照してください。

### setReqCharacterEncoding() メソッド

サーブレット 2.3 より前の環境用に、`setReqCharacterEncoding()` メソッドが用意されています。このメソッドは、サーブレット・コンテナのデフォルトのエンコードが適切でない場合に役立ちます。このメソッドを使用して、Java コードでの `getParameter()` コール、JSP コードで Bean プロパティを設定する `jsp:setProperty` タグなど、マルチバイトのリクエスト・パラメータおよび Bean プロパティ設定のエンコードを指定します。デフォルトのエンコードが適切な場合はこのメソッドを使用する必要はなく、使用すると、アプリケーションでパフォーマンスのオーバーヘッドが生じる場合があります。

`setReqCharacterEncoding()` メソッドは、`oracle.jsp.util` パッケージの `PublicUtil` クラスにある静的なメソッドで、シグネチャは次のとおりです。

```
public static void setReqCharacterEncoding
    (HttpServletRequest req, String encoding)
    throws java.io.UnsupportedEncodingException
```

このメソッドは、特に、次のパラメータ名と値に影響を与えます。

- リクエスト・オブジェクト `getParameter()` メソッドの出力
- リクエスト・オブジェクト `getParameterValues()` メソッドの出力
- リクエスト・オブジェクト `getParameterNames()` メソッドの出力
- Bean プロパティ値に対する `jsp:setProperty` の設定

メソッドの起動時に、次のように、リクエスト・オブジェクトと任意のエンコードを指定する文字列を入力します。

```
oracle.jsp.util.PublicUtil.setReqCharacterEncoding(request, "EUC-JP");
```

---

---

**注意：** `setReqCharacterEncoding()` メソッドには、サーブレット 2.3 API の `request.setCharacterEncoding()` メソッドとの上位互換性があります。

---

---

## JSP の `translate_params` 構成パラメータ

このブール・フラグを `true` に設定すると、マルチバイト（グローバリゼーション・サポート）のリクエスト・パラメータまたは Bean プロパティ設定をエンコードしないサーブレット・コンテナがオーバーライドされます。デフォルト設定は `false` です。`true` に設定した場合、Oracle JSP コンテナは、リクエスト・パラメータと Bean プロパティ設定をデコードおよびエンコードします。`false` に設定すると、サーブレット・コンテナから未変更のパラメータが戻されます。

次の状況では、`translate_params` を有効にしないでください。

- サーブレット・コンテナがマルチバイト・パラメータ・エンコードを適切に処理している場合  
この状況で、`translate_params` を `true` に設定すると、正しい結果が得られない場合があります。JServ 1.1 では、マルチバイト・パラメータ・エンコードが適切に処理されないことが判明しています。
- リクエスト・パラメータで、JSP の `page` ディレクティブまたは `setContentType()` メソッドでレスポンス用に指定したエンコードとは異なるエンコードを使用する場合
- `translate_params` の内容と同等の代替機能を持つコードが、すでに JSP ページに存在している場合

B-26 ページの「`translate_params` 構成パラメータと同等のコード」を参照。

## マルチバイト以外のサブレット・コンテナをオーバーライドする場合の `translate_params` の影響

`translate_params` を `true` に設定すると、マルチバイトのリクエスト・パラメータと Bean プロパティ設定をデコードおよびエンコードできないサブレット・コンテナの不完全な機能がオーバーライドされます (JSP 構成パラメータの設定方法は、B-15 ページの「JServ での JSP パラメータの設定」を参照してください)。

このフラグを有効にすると、JSP コンテナは、`response.getCharacterEncoding()` メソッドで示された `response` オブジェクトのキャラクタ・セットに基づいて、リクエスト・パラメータと Bean プロパティの設定をエンコードします。

## `translate_params` 構成パラメータと同等のコード

`translate_params` 構成パラメータを使用できない、または使用しない場合があります。そのような場合は、JSP ページのスク립トレット・コードを使用して、同等の機能を実装すると便利です。例:

```
<%@ page contentType="text/html; charset=EUC-JP" %>
...
String paramName="XXYYZZ";           // where XXYYZZ is a multibyte string
paramName =
    new String(paramName.getBytes(response.getCharacterEncoding()), "ISO8859_1");
String paramValue = request.getParameter(paramName);
paramValue= new String(paramValue.getBytes("ISO8859_1"), "EUC-JP");
...
```

このコードでは、次の処理を実行します。

- 検索するパラメータ名として、`XXYYZZ` を設定します (`XX`、`YY` および `ZZ` は、3つの日本語文字と仮定します)。
- パラメータ名をサブレット・コンテナのキャラクタ・セットである `ISO-8859-1` にエンコードします。これによって、サブレット・コンテナはパラメータ名を解析できません (最初に、リクエスト・オブジェクトの文字エンコードを使用して、パラメータ名のバイト配列が作成されます)。
- パラメータ名との一致を検索して、リクエスト・オブジェクトからパラメータ値を取得します (リクエスト・オブジェクト内のパラメータ名も `ISO-8859-1` エンコードのため、一致を検出できます)。
- パラメータ値を `EUC-JP` にエンコードし、さらに処理を進めるか、またはブラウザに出力します。

次の2つの項では、有効な `translate_params` に基づくグローバル化のサンプル、および同等のコードが含まれているために `translate_params` の設定に基づかないグローバル化のサンプルを示します。

## translate\_params に基づくグローバル化のサンプル

次のサンプルでは、日本語のユーザー名を受け取り、その名前をブラウザに正確に出力します。マルチバイトのリクエスト・パラメータをエンコードできないサーブレット環境の場合、このサンプルは、JSP 構成パラメータ `translate_params` の `true` の設定に従います。

XXYY はパラメータ名（日本語のユーザー名）で、AABB はデフォルト値（日本語）と仮定します。

(`translate_params` と同等の機能があるため、`translate_params` の設定に従わないコードのサンプルは、次項を参照してください。)

```
<%@ page contentType="text/html; charset=EUC-JP" %>

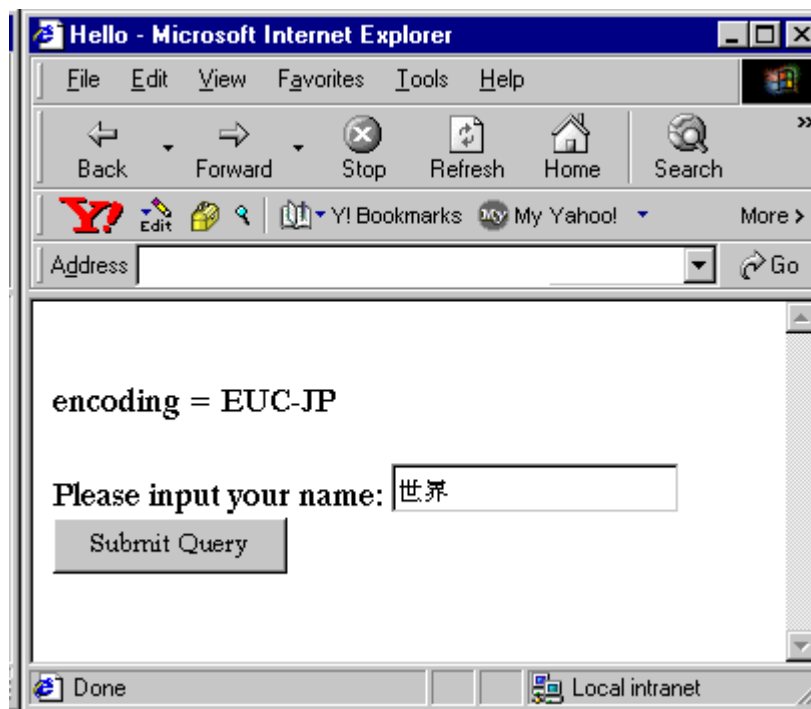
<HTML>
<HEAD>
<TITLE>Hello</TITLE></HEAD>
<BODY>
<%
    //charset is as specified in page directive (EUC-JP)
    String charset = response.getCharacterEncoding();
%>
    <BR> encoding = <%= charset %> <BR>

<%

String paramValue = request.getParameter("XXYY");

if (paramValue == null || paramValue.length() == 0) { %>
    <FORM METHOD="GET">
    Please input your name: <INPUT TYPE="TEXT" NAME="XXYY" value="AABB" size=20> <BR>
    <INPUT TYPE="SUBMIT">
    </FORM>
<% }
else
{ %>
    <H1> Hello, <%= paramValue %> </H1>
<% } %>
</BODY>
</HTML>
```

次のように入力します。



次のように出力されます。





### translate\_params に基づかないグローバル化のサンプル

次のサンプルでは、前述のサンプルと同様に、日本語のユーザー名を受け取り、その名前をブラウザに正確に出力します。ただし、このサンプルでは、`translate_params` と同等の機能を持つコードがあるため、`translate_params` の設定には従っていません。

---

**重要：** `translate_params` と同等の機能を持つコードを使用する場合は、`translate_params` フラグを有効にしないでください。このフラグを有効にすると、正しい結果が得られない場合があります。

---

XXYY はパラメータ名（日本語のユーザー名）で、AABB はデフォルト値（日本語）と仮定します。

このサンプル内の主なコードの説明は、B-26 ページの「[translate\\_params 構成パラメータと同等のコード](#)」を参照してください。

```
<%@ page contentType="text/html; charset=EUC-JP" %>

<HTML>
<HEAD>
<TITLE>Hello</TITLE></HEAD>
<BODY>
<%
    //charset is as specified in page directive (EUC-JP)
    String charset = response.getCharacterEncoding();
%>
    <BR> encoding = <%= charset %> <BR>
<%
String paramName = "XXYY";

paramName = new String(paramName.getBytes(charset), "ISO8859_1");

String paramValue = request.getParameter(paramName);

if (paramValue == null || paramValue.length() == 0) { %>
    <FORM METHOD="GET">
    Please input your name: <INPUT TYPE="TEXT" NAME="XXYY" value="AABB" size=20> <BR>
    <INPUT TYPE="SUBMIT">
    </FORM>
<% }
else
{
    paramValue= new String(paramValue.getBytes("ISO8859_1"), "EUC-JP"); %>
    <H1> Hello, <%= paramValue %> </H1>
<% } %>
</BODY>
</HTML>
```

## translate\_params からの移行

JServ から OC4J に移行する場合、OC4J JSP コンテナのグローバル・インクルード機能（7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照）は、グローバル化に translate\_params を使用していたアプリケーションの移行に役立ちます。

この場合、グローバルにインクルードされたファイルを、次のいずれかのスクリプトレットで構成すると、translate\_params と同等の機能をアーカイブできます。

- リクエストのキャラクタ・セットをハードコードします。

```
<% request.setCharacterEncoding("desired_charset"); %>
```

または

- JSP の page ディレクティブに指定されているレスポンスのキャラクタ・セットを、リクエストのキャラクタ・セットとして使用します。

```
<% request.setCharacterEncoding(response.getCharacterEncoding()); %>
```

または

- Java ロジックによって動的に判断されたレスポンスのキャラクタ・セットを、リクエストのキャラクタ・セットとして使用します。

```
<% String yourCharSet = yourLogicToDetermineCharset();
   response.setContentType("text/html; charset="+yourCharSet);
   request.setCharacterEncoding(response.getCharcterEncoding());
   // NOTE: The relative ordering of response.setContentType()
   // and request.setCharacterEncoding() is important.
   %>
```

## JServ に対する JSP アプリケーションとセッションのサポート

Oracle JSP コンテナでは、JSP 仕様をサーブレット 2.0 環境で実装するための機能として、`globals.jsa` ファイルをサポートしています。Web アプリケーションとサーブレット・コンテキストは、サーブレット仕様 2.0 では完全に定義されていません。

次の各項では、`globals.jsa` 機能について説明し、OC4J 環境に移行するときの `globals.jsa` からの移行についても説明します。

- [globals.jsa 機能の概要](#)
- [globals.jsa の構文とセマンティクスの概要](#)
- [globals.jsa のイベント・ハンドラ](#)
- [グローバルな宣言とディレクティブ](#)
- [globals.jsa からの移行](#)

---

---

### 重要：

- `globals.jsa` 機能は、OC4J JSP コンテナではサポートされていません。JServ でのみ使用できます。
  - セキュリティ上の理由から、`globals.jsa` ファイルへのアクセスは制限することをお勧めします。デフォルトではすでに制限されています。
  - `globals.jsa` ファイルの名前には、すべて小文字を使用してください。大 / 小文字の混在は、大 / 小文字を区別しない環境では問題ありませんが、区別する環境にページをポートした場合に発生した問題の診断が困難になります。
- 
- 

### globals.jsa 機能の概要

単一の Java Virtual Machine 内では、アプリケーションごとに（つまり、サーブレット・コンテキストごとに）1 つの `globals.jsa` ファイルを使用できます。このファイルは、次の領域で Web アプリケーションの概念をサポートします。

- アプリケーションのデプロイ—このファイルのアプリケーション位置マーカーとしてのロールによってアプリケーション・ルートを定義します。
- 固有のアプリケーションとセッション—このファイルを使用することによって、固有のサーブレット・コンテキストとセッション・オブジェクトを各アプリケーションに提供します。
- アプリケーションのライフ・サイクル管理—セッションとアプリケーションの開始イベントと終了イベントを使用します。

`globals.jsa` ファイルには、グローバルな Java 宣言のための手段およびアプリケーションのすべての JSP ページにわたる JSP ディレクティブも用意されています。

## globals.jsa によるアプリケーションのデプロイ

サーブレットが組み込まれていない JSP アプリケーションをデプロイするには、ディレクトリ構造を Web サーバーにコピーし、`globals.jsa` という名前のファイルを作成してアプリケーション・ルート・ディレクトリに格納します。

`globals.jsa` ファイルは、サイズを 0 (ゼロ) にできます。Oracle JSP コンテナはこのファイルを検索し、このファイルが格納されているディレクトリ (URL 仮想パスからマッピングされます) をアプリケーションのルート・ディレクトリとして定義します。

JSP コンテナは、JSP アプリケーション・リソースのデフォルトの場所も定義します。たとえば、アプリケーションに対して相対的な `/WEB-INF/classes` ディレクトリと `/WEB-INF/lib` ディレクトリにあるアプリケーションの Bean とクラスは、JSP クラス・ローダーによって自動的にロードされるため、特別な構成は必要ありません。

---

**注意：** サーブレットが組み込まれているアプリケーションの場合、特に、サーブレット仕様 2.2 より前のサーブレット環境では、サーブレットをデプロイする場合と同様に手動による構成が必要です。

---

## globals.jsa による固有のアプリケーションとセッション

サーブレット仕様 2.0 には、Web アプリケーションの概念およびサーブレット・コンテキストとアプリケーションとの間の関連が明確に定義されていませんでした (これより後のサーブレット仕様では定義されています)。JServ などのサーブレット 2.0 環境では、JVM ごとに、サーブレット・コンテキスト・オブジェクトは 1 つのみです。サーブレット 2.0 環境では、セッション・オブジェクトも 1 つのみです。

`globals.jsa` ファイルは、特にサーブレット 2.0 環境で使用するために、1 つの Web サーバー内で複数のアプリケーションとセッションをサポートします。

アプリケーションごとに固有のサーブレット・コンテキスト・オブジェクトを使用できない場合、アプリケーションに `globals.jsa` ファイルが存在することによって、Oracle JSP コンテナは、固有の `ServletContext` オブジェクトをそのアプリケーションに提供できません。

さらに、1 つまたは複数のサーブレット・コンテキストを持つ 1 つのセッション・オブジェクトのみが存在している場合は、`globals.jsa` ファイルの存在によって、JSP コンテナは、`HttpSession` プロキシ・オブジェクトをアプリケーションに提供できます。これによって、他のアプリケーションとのセッション変数名の衝突を回避できます。ただし、他のアプリケーションによるアプリケーション・データの参照や変更は防止できません。これは、`HttpSession` オブジェクトが、その一部の機能を基礎となるサーブレット・セッション環境に依存する必要があるためです。

## globals.jsa によるアプリケーションとセッションのライフ・サイクル管理

アプリケーションは、重要な状態遷移の発生を認識する必要があります。たとえば、アプリケーションは、HTTP セッションの開始時にリソースを取得して終了時に開放する場合があります。あるいは、アプリケーション自体の開始時または終了時に、永続データをリストアまたは保存します。

globals.jsa ファイルは、次の 4 つのイベントでこの機能をサポートします。

- session\_OnStart
- session\_OnEnd
- application\_OnStart
- application\_OnEnd

サーバーのレスポンスが必要なこれらのイベントに対するイベント・ハンドラを globals.jsa ファイルに記述できます。

session\_OnStart イベントと session\_OnEnd イベントは、HTTP セッションの開始時と終了時にそれぞれトリガーされます。

application\_OnStart イベントは、単一の JVM 内でアプリケーションに対する最初のリクエストが発生すると、そのアプリケーションに対してトリガーされます。

application\_OnEnd イベントは、Oracle JSP コンテナがアプリケーションをアンロードするとトリガーされます。

詳細は、B-36 ページの「globals.jsa のイベント・ハンドラ」を参照してください。

## globals.jsa の構文とセマンティクスの概要

この項では、globals.jsa ファイルの一般的な構文とセマンティクスの概要を説明します。

globals.jsa ファイル内の各イベント・ブロック (session\_OnStart ブロック、session\_OnEnd ブロック、application\_OnStart ブロックまたは application\_OnEnd ブロック) には、イベント開始タグ、イベント終了タグ、およびイベント・ハンドラ・コードが含まれたボディ (開始タグと終了タグの間のすべてのコード) があります。

次に、このパターンの例を示します。

```
<event:session_OnStart>
  <% This scriptlet contains the implementation of the event handler %>
</event:session_OnStart>
```

イベント・ブロックのボディには、有効な JSP タグ (標準タグやカスタム・タグ・ライブラリに定義されているタグ) を含めることができます。

ただし、イベント・ブロック内の JSP タグの範囲は、そのブロックのみに制限されません。たとえば、あるイベント・ブロック内の `jsp:useBean` タグで宣言された Bean は、その Bean を使用する別のイベント・ブロックで再度宣言する必要があります。この制限の回避に、`globals.jsa` のグローバル宣言機能を使用できます。B-41 ページの「[グローバルな宣言とディレクティブ](#)」を参照。

4つのイベント・ハンドラの詳細は、B-36 ページの「[globals.jsa のイベント・ハンドラ](#)」を参照してください。

---

**重要：** 通常の JSP ページで使用する静的なテキストは、`session_OnStart` ブロックにのみ存在できます。`session_OnEnd`、`application_OnStart` および `application_OnEnd` の各イベント・ブロックに含めることができるのは、Java スクリプトレットのみです。

---

JSP の暗黙的なオブジェクトは、次のように、`globals.jsa` のイベント・ブロックで使用できます。

- `application_OnStart` ブロックは、`application` オブジェクトにアクセスできます。
- `application_OnEnd` ブロックは、`application` オブジェクトにアクセスできます。
- `session_OnStart` ブロックは、`application`、`session`、`request`、`response`、`page` および `out` の各オブジェクトにアクセスできます。
- `session_OnEnd` ブロックは、`application` オブジェクトと `session` オブジェクトにアクセスできます。

**完全な `globals.jsa` ファイルの例** この例では、4つのイベント・ハンドラをすべて使用する完全な `globals.jsa` ファイルを示します。

```
<event:application_OnStart>

  <%-- Initializes counts to zero --%>
  <jsp:useBean id="pageCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <jsp:useBean id="sessionCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <jsp:useBean id="activeSessions" class="oracle.jsp.jml.JmlNumber" scope = "application" />

</event:application_OnStart>

<event:application_OnEnd>

  <%-- Acquire beans --%>
  <jsp:useBean id="pageCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <jsp:useBean id="sessionCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <% application.log("The number of page hits were: " + pageCount.getValue() ); %>
  <% application.log("The number of client sessions were: " + sessionCount.getValue() ); %>

</event:application_OnEnd>
```

```
</event:application_OnEnd>

<event:session_OnStart>

  <%-- Acquire beans --%>
  <jsp:useBean id="sessionCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <jsp:useBean id="activeSessions" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <%
    sessionCount.setValue(sessionCount.getValue() + 1);
    activeSessions.setValue(activeSessions.getValue() + 1);
  %>
  <br>
  Starting session #: <%=sessionCount.getValue() %> <br>
  There are currently <b> <%= activeSessions.getValue() %> </b> active sessions <p>

</event:session_OnStart>

<event:session_OnEnd>

  <%-- Acquire beans --%>
  <jsp:useBean id="activeSessions" class="oracle.jsp.jml.JmlNumber" scope = "application" />
  <%
    activeSessions.setValue(activeSessions.getValue() - 1);
  %>

</event:session_OnEnd>
```

## globals.jsa のイベント・ハンドラ

この項では、globals.jsa の4つのイベント・ハンドラについて詳細を説明します。

### application\_OnStart イベント・ハンドラ

application\_OnStart ブロックの一般的な構文は、次のとおりです。

```
<event:application_OnStart>
  <% This scriptlet contains the implementation of the event handler %>
</event:application_OnStart>
```

application\_OnStart イベント・ハンドラのボディは、Oracle JSP コンテナが最初の JSP ページをアプリケーションにロードすると実行されます。この状態は、通常、クライアントによる最初の HTTP リクエストが、アプリケーション内の任意のページに対して行われると発生します。アプリケーションは、このイベントを使用して、アプリケーション全体のリソース（データベース接続プールや永続リポジトリからアプリケーション・オブジェクトに読み取られたデータなど）を初期化します。



イベント・ハンドラに含めることができるのは、JSP タグ（カスタム・タグを含む）のみです。静的なテキストを含めることはできません。

このイベント・ハンドラで発生し、イベント・ハンドラ・コードで処理されないエラーは、JSP コンテナが自動的に捕え、アプリケーションのサーブレット・コンテキストを使用してログします。イベント処理は、エラーが発生しなかった場合と同じように続行されます。

**例 : application\_OnStart** この例では、特定のユーザーについて生成された宝くじ抽選番号が、1 日間キャッシュされています。ユーザーは、抽選番号を再リクエストすると、同じ番号セットを取得します。キャッシュは 1 日 1 回リサイクルされ、各ユーザーは新しい抽選番号を取得します。宝くじアプリケーションが正常に機能するためには、アプリケーションの停止時にキャッシュを永続的に保持し、アプリケーションが再びアクティブ化したときにそのキャッシュをリフレッシュする必要があります。

application\_OnStart イベント・ハンドラは、キャッシュを lotto.che ファイルから読み取ります。

```
<event:application_OnStart>

<%
    Calendar today = Calendar.getInstance();
    application.setAttribute("today", today);
    try {
        FileInputStream fis = new FileInputStream
            (application.getRealPath("/") + File.separator + "lotto.che");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Calendar cacheDay = (Calendar) ois.readObject();
        if (cacheDay.get(Calendar.DAY_OF_YEAR) == today.get(Calendar.DAY_OF_YEAR)) {
            cachedNumbers = (Hashtable) ois.readObject();
            application.setAttribute("cachedNumbers", cachedNumbers);
        }
        ois.close();
    } catch (Exception theE) {
        // catch all -- can't use persistent data
    }
%>

</event:application_OnStart>
```

## application\_OnEnd イベント・ハンドラ

application\_OnEnd ブロックの一般的な構文は、次のとおりです。

```
<event:application_OnEnd>
  <% This scriptlet contains the implementation of the event handler %>
</event:application_OnEnd>
```

application\_OnEnd イベント・ハンドラのボディは、Oracle JSP コンテナが JSP アプリケーションをアンロードすると実行されます。アンロードが発生するのは、以前にロードされたページがオンデマンドの動的な再変換後に再ロードされたとき (JSP の `unsafe_reload` 構成パラメータが有効でない場合)、または、基礎となるサーブレット・コンテナによって `destroy()` メソッドがコールされ、JSP コンテナ (それ自体がサーブレット) が終了するときです。アプリケーションは、application\_OnEnd イベントを使用して、アプリケーション・レベルのリソースをクリーンアップしたり、アプリケーションの状態を永続的なストアに書き込みます。

イベント・ハンドラに含めることができるのは、JSP タグ (カスタム・タグを含む) のみです。静的なテキストを含めることはできません。

このイベント・ハンドラで発生し、イベント・ハンドラ・コードで処理されないエラーは、JSP コンテナが自動的に捕え、アプリケーションのサーブレット・コンテキストを使用してログします。イベント処理は、エラーが発生しなかった場合と同じように続行されます。

**例 : application\_OnEnd** このイベント・ハンドラでは、アプリケーションの終了前に、キャッシュが `lotto.che` ファイルに書き込まれます。

```
<event:application_OnEnd>

<%
Calendar now = Calendar.getInstance();
Calendar today = (Calendar) application.getAttribute("today");
if (cachedNumbers.isEmpty() ||
    now.get(Calendar.DAY_OF_YEAR) > today.get(Calendar.DAY_OF_YEAR)) {
    File f = new File(application.getRealPath("/") + File.separator + "lotto.che");
    if (f.exists()) f.delete();
    return;
}

try {
    FileOutputStream fos = new FileOutputStream
        (application.getRealPath("/") + File.separator + "lotto.che");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(today);
    oos.writeObject(cachedNumbers);
    oos.close();
} catch (Exception theE) {
    // catch all -- can't use persistent data
```

```
}  
%>  
  
</event:application_OnEnd>
```

## session\_OnStart イベント・ハンドラ

session\_OnStart ブロックの一般的な構文は、次のとおりです。

```
<event:session_OnStart>  
  <% This scriptlet contains the implementation of the event handler %>  
  Optional static text...  
</event:session_OnStart>
```

session\_OnStart イベント・ハンドラのボディは、Oracle JSP コンテナが JSP ページのリクエストに応じて新規セッションを作成すると実行されます。この状態は、アプリケーション内のセッション対応 JSP ページが最初のリクエストを受け取ると、クライアントごとに発生します。

アプリケーションは、このイベントを次の目的で使用できます。

- 特定のクライアントに関連するリソースを初期化するため
- アプリケーション内でのクライアントの開始位置を制御するため

暗黙的な out オブジェクトは session\_OnStart で使用可能なため、JSP タグ以外に静的なテキストを含めることができるのは、この globals.jsa イベント・ハンドラのみです。

session\_OnStart イベント・ハンドラは、JSP ページのコードの実行前にコールされます。その結果、session\_OnStart の出力は、そのページの出力より先行します。

session\_OnStart イベント・ハンドラおよびイベントをトリガーした JSP ページは、同じ out ストリームを共有します。このストリームのバッファ・サイズは、JSP ページのバッファ・サイズによって制御されます。session\_OnStart イベント・ハンドラは、ブラウザへのストリームを自動的にフラッシュしません。ストリームは、一般的な JSP ルールに従ってフラッシュされます。session\_OnStart イベントをトリガーする JSP ページには、ヘッダーも書き込むことができます。

このイベント・ハンドラで発生し、イベント・ハンドラ・コードで処理されないエラーは、JSP コンテナが自動的に捕え、アプリケーションのサーブレット・コンテキストを使用してログします。イベント処理は、エラーが発生しなかった場合と同じように続行されます。

**例 : session\_OnStart** 次の例では、新規セッションが、必ずアプリケーションの最初のページ (index.jsp) で起動します。

```
<event:session_OnStart>

    <% if (!page.equals("index.jsp")) { %>
        <jsp:forward page="index.jsp" />
    <% } %>

</event:session_OnStart>
```

## session\_OnEnd イベント・ハンドラ

session\_OnEnd ブロックの一般的な構文は、次のとおりです。

```
<event:session_OnEnd>
    <% This scriptlet contains the implementation of the event handler %>
</event:session_OnEnd>
```

session\_OnEnd イベント・ハンドラのボディは、Oracle JSP コンテナが既存のセッションを無効にすると実行されます。これは、次のいずれかの状況で発生します。

- アプリケーションが、session.invalidate() メソッドをコールしてセッションを無効にした場合
- セッションがサーバーで期限切れ (タイムアウト) となった場合

アプリケーションは、このイベントを使用して、クライアントのリソースを解放します。

イベント・ハンドラに含めることができるのは、JSP タグ (タグ・ライブラリのタグを含む) のみです。静的なテキストを含めることはできません。

このイベント・ハンドラで発生し、イベント・ハンドラ・コードで処理されないエラーは、JSP コンテナが自動的に捕え、アプリケーションのサーブレット・コンテキストを使用してログします。イベント処理は、エラーが発生しなかった場合と同じように続行されます。

**例 : session\_OnEnd** この例では、セッションの終了時に、アクティブなセッションの件数が減分されます。

```
<event:session_OnEnd>

    <!-- Acquire beans -->
    <jsp:useBean id="activeSessions" class="oracle.jsp.jml.JmlNumber" scope = "application" />

    <%
        activeSessions.setValue(activeSessions.getValue() - 1);
    %>

</event:session_OnEnd>
```

## グローバルな宣言とディレクティブ

globals.jsa ファイルは、イベント・ハンドラの保持以外に、JSP アプリケーションに対するディレクティブとオブジェクトのグローバルな宣言にも使用できます。JSP ディレクティブ、JSP 宣言、JSP コメント、および scope パラメータ (jsp:useBean など) を指定した JSP タグをインクルードできます。

この項には、次の項目が含まれます。

- [グローバルな JSP ディレクティブ](#)
- [globals.jsa での宣言](#)
- [グローバルな JavaBeans](#)
- [globals.jsa の構造](#)
- [グローバルな宣言とディレクティブの例](#)

### グローバルな JSP ディレクティブ

globals.jsa 内で使用するディレクティブには、次の 2 つの目的があります。

- globals.jsa ファイル自体の処理に必要な情報を宣言します。
- 後続ページのデフォルト値を確立します。

globals.jsa ファイル内のディレクティブは、アプリケーション内の全 JSP ページの暗黙的なディレクティブになります。ただし、特定のページについては、globals.jsa ディレクティブを上書きできます。

globals.jsa ディレクティブは、1 つの JSP ページ内で、属性ごとに上書きされます。たとえば、globals.jsa ファイルに、次のディレクティブがあると仮定します。

```
<%@ page import="java.util.*" bufferSize="10kb" %>
```

さらに、JSP ページには、次のディレクティブがあると仮定します。

```
<%@page bufferSize="20kb" %>
```

このページは、次のディレクティブがあるページと同じになります。

```
<%@ page import="java.util.*" bufferSize="20kb" %>
```

### globals.jsa での宣言

任意のイベント間で共有するメソッドまたはデータ・メンバーを globals.jsa ファイルで宣言する場合は、globals.jsa ファイル内で JSP の `<%!... %>` 宣言を使用します。

アプリケーション内の JSP ページはこれらの宣言にアクセスできないため、アプリケーション・ライブラリの実装にはこの機能を使用できません。宣言は、イベント・ハンドラ間で共有する共通機能について、globals.jsa ファイルでサポートされます。

## グローバルな JavaBeans

globals.jsa ファイル内で宣言する最も一般的な要素は、グローバル・オブジェクトです。globals.jsa ファイルで宣言するオブジェクトは、globals.jsa イベント・ハンドラおよびアプリケーション内の全 JSP ページの暗黙的なオブジェクト環境の一部になります。

globals.jsa ファイルで (jsp:useBean タグなどを使用して) 宣言するオブジェクトは、アプリケーションの個別の JSP ページで再度宣言する必要はありません。

グローバル・オブジェクトは、scope パラメータを指定する JSP タグまたは拡張機能 (標準の jsp:useBean タグや JML の useVariable タグなど) を使用して宣言できます。グローバルに宣言したオブジェクトのスコープは、session または application のいずれかになります (page または request スコープは指定できません)。

ネストされたタグはサポートされます。したがって、jsp:setProperty タグは、jsp:useBean タグ内でネストできます (jsp:setProperty を jsp:useBean タグの外側で使用すると、変換エラーが発生します)。

## globals.jsa の構造

グローバル・オブジェクトを globals.jsa イベント・ハンドラで使用する場合は、宣言する位置が重要です。特定のイベント・ハンドラの前に宣言されるオブジェクトのみが、暗黙的なオブジェクトとして、そのイベント・ハンドラに追加されます。このため、開発者は、globals.jsa ファイルを次の順序で作成することをお勧めします。

1. グローバル・ディレクティブ
2. グローバル・オブジェクト
3. イベント・ハンドラ
4. 宣言

## グローバルな宣言とディレクティブの例

次の globals.jsa ファイルのサンプルでは、次の処理を実行します。

- globals.jsa ファイルおよび後続の全ページ用の JML タグ・ライブラリ (この場合は、コンパイル時実装) を定義します。globals.jsa ファイルの taglib ディレクティブをインクルードすることで、アプリケーションの個別の JSP ページにディレクティブをインクルードする必要がなくなります。
- 全ページで使用する 3 つのアプリケーション変数を、jsp:useBean 文で宣言します。

```
<%-- Directives at the top --%>

    <%@ taglib uri="oracle.jsp.parse.OpenJspRegisterLib" prefix="jml" %>

<%-- Declare global objects here --%>

    <%-- Initializes counts to zero --%>
    <jsp:useBean id="pageCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
    <jsp:useBean id="sessionCount" class="oracle.jsp.jml.JmlNumber" scope = "application" />
    <jsp:useBean id="activeSessions" class="oracle.jsp.jml.JmlNumber" scope = "application" />

<%-- Application lifecycle event handlers go here --%>

    <event:application_OnStart>
        <% This scriptlet contains the implementation of the event handler %>
    </event:application_OnStart>

    <event:application_OnEnd>
        <% This scriptlet contains the implementation of the event handler %>
    </event:application_OnEnd>

    <event:session_OnStart>
        <% This scriptlet contains the implementation of the event handler %>
    </event:session_OnStart>

    <event:session_OnEnd>
        <% This scriptlet contains the implementation of the event handler %>
    </event:session_OnEnd>

<%-- Declarations used by the event handlers go here --%>
```

## globals.jsa からの移行

globals.jsa を使用するアプリケーションを使用して、JServ から OC4J に移行する場合、次の方針を使用できます。

- globals.jsa をアプリケーション・マーカースとして使用するかわりに、標準の WAR パッケージングを使用してアプリケーション構造を示します。
- globals.jsa のセッション開始、セッション終了、アプリケーション開始およびアプリケーション終了の各イベントを使用するかわりに、標準のサーブレット 2.3 のリスナー機能を使用します。たとえば、標準の `javax.servlet.ServletContextListener` インタフェースおよび `javax.servlet.http.HttpSessionListener` インタフェースを使用すると、同等の機能が提供されます。

- `globals.jsa` をグローバル変数宣言で使用するかわりに、単一のソース・ファイル内で宣言し、Oracle JSP コンテナのグローバル・インクルード機能を使用します。7-8 ページの「[Oracle JSP のグローバル・インクルード](#)」を参照。

---

---

**注意：** Oracle Application Server 10g (9.0.4) の JServ 環境では、以前の JSP フロントエンド・サーブレットの `oracle.jsp.JspServlet` が使用されます。OC4J フロントエンド・サーブレットの `oracle.jsp.runtimev2.JspServlet` は、`globals.jsa` をサポートしていません。

---

---



---

---

## 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。この章の内容は、次のとおりです。

- [Apache HTTP Server](#)
- [Apache JServ](#)

## 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。

## The Apache Software License

```
/* =====  
 * 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.
*/

```

## Apache JServ

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 JServ Public License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistribution 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.

- All advertising materials mentioning features or use of this software must display the following acknowledgment:

**This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (<http://java.apache.org/>).**

- The names "Apache JServ", "Apache JServ Servlet Engine" and "Java Apache Project" must not be used to endorse or promote products derived from this software without prior written permission.
- Products derived from this software may not be called "Apache JServ" nor may "Apache" nor "Apache JServ" appear in their names without prior written permission of the Java Apache Project.
- Redistribution of any form whatsoever must retain the following acknowledgment:

**This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (<http://java.apache.org/>).**

THIS SOFTWARE IS PROVIDED BY THE JAVA APACHE PROJECT "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 JAVA APACHE PROJECT 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.

## 記号

\_jspService() メソッド, A-12

## A

activation.jar、電子メール用の Java アクティブ化ファイル, 3-12  
addclasspath、ojspc のオプション, 7-20  
Apache JServ, 「JServ」を参照  
Application Server Control  
「JSP プロパティ」ページ, 3-33  
概要, 3-32  
サポートされていない JSP パラメータ, 3-35  
サポートされている JSP パラメータ, 3-34  
application\_OnEnd タグ、globals.jsa, B-38  
application\_OnStart タグ、globals.jsa, B-36  
application.xml、OC4J 構成ファイル, 3-31  
application オブジェクト (暗黙的), 1-13  
application スコープ (JSP オブジェクト), 1-15  
appRoot、ojspc のオプション, 7-20  
autoreload-jsp-pages、autoreload-jsp-beans (非サポート), 3-30

## B

batchMask、ojspc のオプション, 7-21  
bypass\_source 構成パラメータ (JServ), B-10

## C

cache.jar、Java Object Cache 用, 3-12  
check\_page\_scope 構成パラメータ, 3-18  
classesXX.zip、JDBC 用, 3-12

classpath  
classpath 構成パラメータ (JServ), B-11  
config オブジェクト (暗黙的), 1-13  
Cookie, A-6

## D

d、ojspc のオプション (バイナリ出力ディレクトリ), 7-23  
data-sources.xml、OC4J 構成ファイル, 3-31  
debug\_mode 構成パラメータ, 3-18  
default-web-site.xml、OC4J 構成ファイル, 3-31  
deleteSource、ojspc のオプション, 7-23  
developer\_mode 構成パラメータ (JServ), B-11  
DMS サポート, 2-15

## E

EAR ファイル, 3-31, 7-34  
EJB  
JSP ページからのコール, 4-13  
OC4J の EJB タグ・ライブラリの使用, 4-14  
emit\_debuginfo 構成パラメータ, 3-19  
empty アクション (タグ・ライブラリ), 8-29  
enable-jsp-dispatcher-shortcuts フラグ, 3-29  
Enterprise Manager  
「JSP プロパティ」ページ, 3-33  
サポートされていない JSP パラメータ, 3-35  
サポートされている JSP パラメータ, 3-34  
exception オブジェクト (暗黙的), 1-14  
extend、ojspc のオプション, 7-24  
external\_resource\_timeout 構成パラメータ (JServ), B-12  
external\_resource 構成パラメータ, 3-19  
extra\_imports 構成パラメータ, 3-19

extraImports、ojspc のオプション, 7-24  
extres、ojspc のオプション, 7-25

## F

fallback タグ (plugin タグとともに使用), 1-21  
Feiner, Amy (Welcome), 1-3  
forgive\_dup\_dir\_attr 構成パラメータ, 3-20  
forgiveDupDirAttr、ojspc のオプション, 7-25  
forward タグ, 1-19

## G

getProperty タグ, 1-18  
globals.jsa  
アプリケーション・イベント, B-36  
アプリケーションとセッションのライフ・サイクル, B-34  
アプリケーションのデプロイ, B-33  
移行, B-43  
イベント処理, B-36  
機能の概要, B-32  
グローバル宣言, B-41  
グローバルな JavaBeans, B-42  
グローバルな JSP ディレクティブ, B-41  
構文とセマンティクスの概要, B-34  
固有のアプリケーションとセッション, B-33  
サーブレット 2.0 に対する拡張サポート, B-32  
セッション・イベント, B-39  
ファイルの内容、構造, B-42  
例、宣言とディレクティブ, B-42  
globals.jsa によるセッションのサポート (JServ), B-33  
global-web-application.xml、OC4J 構成ファイル, 3-31

## H

help、ojspc のオプション, 7-25  
HttpJspPage インタフェース, A-11  
HttpSessionBindingListener, 4-18  
HttpSession インタフェース, A-5

## I

id 属性 (XML ビュー), 5-15  
implement、ojspc のオプション, 7-25  
include タグ, 1-19  
include ディレクティブ, 1-8

## J

JavaBeans  
useBean タグを使用, 1-16  
グローバルな JavaBeans、globals.jsa, B-42  
スクリプトレットとの比較, 6-2  
ビジネス・ロジックの分離に使用, 1-5  
javaccmd 構成パラメータ, 3-21  
JDeveloper  
JSP サポート, 2-12  
JSP ページのデプロイで使用, 7-36  
JDK 1.4 に関する考慮事項, 3-11  
jndi.jar、データ・ソースおよび EJB 用, 3-12  
JServ  
JServ での ojspc の使用, B-16  
JSP アプリケーション・フレームワーク, B-18  
JSP とサーブレット間でのセッション共有の概要, B-19  
JSP の動的なインクルードのサポート, B-19  
mod\_jserv モジュール, B-17  
Oracle Application Server で使用, B-1  
エラー処理、send\_error 構成パラメータ, B-13  
クラスパス構成, B-2  
構成パラメータ, B-4  
構成パラメータの設定, B-15  
構成、ファイル名拡張子のマッピング, B-3  
セッション共有、session\_sharing 構成パラメータ, B-14  
特別な考慮事項の概要, B-17  
別名変換, B-22  
JServ でのアプリケーション・フレームワーク, B-18  
JSP fallback タグ (plugin タグとともに使用), 1-21  
JSP forward タグ, 1-19  
JSP getProperty タグ, 1-18  
jsp id 属性 (XML ビュー), 5-15  
JSP include タグ, 1-19  
JSP param タグ, 1-18  
JSP plugin タグ, 1-21  
JSP setProperty タグ, 1-17  
JSP useBean タグ  
構文, 1-16  
JSP XML 構文、「XML 構文」を参照  
JSP XML ビュー、「XML ビュー」を参照  
JSP XML 文書, 5-2  
jsp-cache-directory 設定, 3-28, 7-7  
jsp-cache-tlds フラグ, 3-28, 8-20  
JspPage インタフェース, A-11

jsp-print-null フラグ, 3-28  
jspService() メソッド, A-12  
jsp-taglib-locations 設定, 3-28, 8-20  
jsp-timeout フラグ, 3-28  
JspWriter オブジェクト, 1-13  
JSP からのサーブレットの起動、サーブレットからの  
JSP の起動, 4-2  
JSP からのサーブレットのコール、サーブレットから  
の JSP のコール, 4-2  
JSP とサーブレット間の相互作用  
JSP からのサーブレットの起動, 4-2  
サーブレットからの JSP の起動、リクエスト・ディ  
スパッチャ, 4-3  
サンプル・コード, 4-5  
データの受渡し、JSP からサーブレット, 4-3  
データの受渡し、サーブレットから JSP, 4-4  
JSP トランスレータ、「トランスレータ」を参照  
JSP のテキスト要素 (XML 構文), 5-10  
JSP のルート要素 (XML 構文), 5-7  
JSP ページ (MVC アーキテクチャなど) を除外, 6-6  
JSP ページの JDBC  
使用サンプル, 4-7  
パフォーマンス強化, 4-10  
JSP ページの実行, 1-24  
JSP ページの実行モデル, 1-25  
JSP ページのリクエスト, 1-27  
JSP ページを使用したサーブレットのラップ, B-20  
JSP をサポートする Oracle プラットフォーム  
JDeveloper, 2-12  
Oracle Application Server, 2-2  
JSTL、サポートの概要, 2-16  
jta.jar、Java Transaction API 用, 3-12

## M

---

mail.jar、アプリケーションからの電子メール用, 3-12  
mod、Apache, 2-3  
Model-View-Controller、JSP ページを除外, 6-6  
MVC アーキテクチャ、JSP ページを除外, 6-6

## N

---

National Language Support、「グローバル化・サ  
ポート」を参照  
NLS、「グローバル化・サポート」を参照  
no\_tld\_xml\_validate 構成パラメータ, 3-22  
noCompile、ojspc のオプション, 7-26

non-empty アクション (タグ・ライブラリ), 8-29  
noTldXmlValidate、ojspc のオプション, 7-26  
NULL 出力フラグ, 3-28  
NULL データ、出力モード, 3-28

## O

---

OC4J  
JSP 実装の概要, 2-8  
概要, 2-4  
スタンドアロン, 2-7  
OC4J のスタンドアロン・バージョン, 2-7  
ojspc 事前変換ツール  
JServ での使用, B-16  
オプションのサマリー表, 7-16  
オプションの説明, 7-20  
概要, 7-12  
基本的な機能の概要, 7-12  
コマンドライン構文, 7-19  
出力ファイル、場所、関連オプション, 7-32  
バッチ事前変換の概要, 7-14  
バッチ事前変換のために使用, 7-38  
ページ事前変換のために使用, 7-37  
ojjsp.jar、JSP コンテナ用, 3-12  
ojsputil.jar、JSP タグ・ライブラリおよびユーティリ  
ティ用, 3-12  
old\_include\_from\_top 構成パラメータ, 3-22  
oldIncludeFromTop、ojspc のオプション, 7-26  
Oracle Application Server  
JServ の使用, B-1  
JSP サポート, 2-2  
概要, 2-2  
Oracle Enterprise Manager、「Enterprise Manager」を  
参照  
Oracle HTTP Server  
mod\_jserv の使用, B-17  
概要、Apache mod の使用, 2-3  
output、ojspc のオプション, 7-27  
out オブジェクト (暗黙的), 1-13

## P

---

packageName、ojspc のオプション, 7-27  
pageContext オブジェクト (暗黙的), 1-13  
page オブジェクト (暗黙的), 1-12  
page スコープ (JSP オブジェクト), 1-14

page ディレクティブ  
概要, 1-7  
グローバル化・サポートのための  
contentype 設定, 9-2  
特性, 6-12  
param タグ, 1-18  
plugin タグ, 1-21  
precompile\_check 構成パラメータ, 3-22

## R

---

reduce\_tag\_code 構成パラメータ, 3-22  
reduceTagCode、ojspc のオプション, 7-27  
req\_time\_introspection 構成パラメータ, 3-23  
reqTimeIntrospection、ojspc のオプション, 7-28  
RequestDispatcher インタフェース, 4-3  
request オブジェクト  
JSP の暗黙的な request オブジェクト, 1-12  
request スコープ (JSP オブジェクト), 1-14  
response オブジェクト  
JSP の暗黙的な response オブジェクト, 1-13  
runtimeXX.zip、SQLJ 用, 3-12

## S

---

S、ojspc のオプション (SQLJ オプション用), 7-28  
send\_error 構成パラメータ (JServ), B-13  
server.xml、OC4J 構成ファイル, 3-31  
session\_OnEnd タグ、globals.jsa, B-40  
session\_OnStart タグ、globals.jsa, B-39  
session\_sharing 構成パラメータ (JServ), B-14  
session オブジェクト  
JSP の暗黙的な session オブジェクト, 1-13  
session スコープ (JSP オブジェクト), 1-15  
setCharacterEncoding() メソッド, 9-7  
setContentype() メソッド、グローバル化・サポート, 9-5  
setProperty\_onerr\_continue 構成パラメータ, 3-23  
setProperty タグ, 1-17  
setReqCharacterEncoding() メソッド、マルチバイト・パラメータ・エンコード (JServ), B-24  
setWriterEncoding() メソッド、グローバル化・サポート, 9-6  
simple-jsp-mapping フラグ, 3-29  
SQLJ  
JSP コード例, 4-15  
JSP サポート, 4-14

Oracle SQLJ オプションの設定, 4-18  
sqljcmd 構成パラメータ, 3-24  
sqljsp ファイル, 4-17  
SQLJ オプション用の ojpspc の S オプション, 7-28  
SQLJ トランスレータのトリガー, 4-17  
sqljcmd 構成パラメータ, 3-24  
SQLJ 用の sqljsp ファイル, 4-17  
srcdir、ojspc のオプション, 7-30  
static\_text\_in\_chars 構成パラメータ, 3-24  
staticTextInChars、ojspc のオプション, 7-30

## T

---

taglib ディレクティブ  
一般的な使用, 8-13  
構文, 1-9  
tagReuse、ojspc のオプション, 7-30  
tags\_reuse\_default 構成パラメータ, 3-25  
TLD、「タグ・ライブラリ・ディスクリプタ」を参照  
TLD の永続的なキャッシング, 8-19  
translate\_params 構成パラメータ (JServ)  
translate\_params に基づかないグローバル化・サポートのサンプル, B-29  
translate\_params に基づくグローバル化・サポートのサンプル, B-27  
移行, B-31  
一般情報, B-14, B-25  
同等のコード, B-26  
マルチバイト以外のサブレット・コンテナをオーバーライドする場合の影響, B-26  
translator.zip、SQLJ 用, 3-12

## U

---

unsafe\_reload 構成パラメータ (JServ), B-15  
URL 書換え, A-6  
use\_old\_compiler 構成パラメータ, 3-25  
useBean タグ, 1-16

## V

---

variable 要素 (タグ・ライブラリ), 8-43  
verbose、ojspc のオプション, 7-31  
version、ojspc のオプション, 7-31



## W

---

WAR デプロイ, 7-34  
WAR ファイル, 3-31, 7-34  
web.xml、タグ・ライブラリへの使用, 8-18  
Web アプリケーション階層, A-9

## X

---

xml\_validate 構成パラメータ, 3-26  
xmlparserv2.jar、XML 妥当性チェック, 3-12  
xmlValidate、ojspc のオプション, 7-31  
XML 構文  
  JSP XML 構文のサマリー表, 5-5  
  カスタム・アクションの要素, 5-10  
  サンプル、従来の構文と XML 構文の比較, 5-11  
  式要素, 5-9  
  スクリプトレット要素, 5-9  
  宣言要素, 5-9  
  ディレクティブ要素, 5-8  
  テキスト要素とその他の要素, 5-10  
  標準アクションの要素, 5-10  
  ルート要素とタグ・ライブラリ名前空間, 5-7  
XML サポート  
  JSP XML 構文, 5-4  
  JSP XML 文書, 5-2  
  JSP XML 文書と JSP XML ビュー、概要, 5-2  
  XML 妥当性チェック構成パラメータ, 3-26  
  XML 妥当性チェック用の ojspc のオプション, 7-31  
  XML ビュー, 5-14  
XML ビュー  
  JSP ページから XML ビューへの変換, 5-14  
  妥当性チェックの jsp id 属性, 5-15  
  変換サンプル, 5-16  
xsu12.jar または xsu11.jar、XML 用, 3-12

## あ

---

アプリケーション・イベント  
  globals.jsa の使用, B-36  
  サブレット・アプリケーションのライフ・サイクル, A-8  
アプリケーション階層, A-9  
アプリケーション相対パス, 1-27

アプリケーションのサポート  
  globals.jsa の使用, B-33  
  サブレット・アプリケーションのライフ・サイクル, A-8  
アプリケーション・ルート機能, 3-2  
暗黙的な JSP オブジェクト  
  暗黙的なオブジェクトの使用, 1-14  
  概要, 1-12

## い

---

移行  
  globals.jsa から, B-43  
  translate\_params から, B-31  
イベント処理  
  globals.jsa の使用, B-36  
  HttpSessionBindingListener の使用, 4-18  
  サブレット・アプリケーションのライフ・サイクル, A-8  
インポート、デフォルト・パッケージ, 3-9

## え

---

エラー処理 (実行時), 4-24

## お

---

オブジェクトとスコープ (JSP オブジェクト), 1-11  
親プロパティ (タグ・ハンドラ), 8-31  
オンデマンド変換 (実行時), 1-25, 1-26

## か

---

外部リソース・ファイル  
  external\_resource パラメータの使用, 3-19  
  ojspc の extres オプションの使用, 7-25  
  静的なテキスト, 6-9  
拡張機能  
  DMS サポート, 2-15  
  Oracle 固有の拡張機能の概要, 2-14  
  SQLJ サポートの概要, 2-14  
  移植可能な拡張機能の概要, 2-13  
  キャッシング・サポートの概要, 2-16  
  グローバル・インクルードの概要, 2-14  
  プログラムによる拡張機能の概要, 2-13  
  カスタム・タグ、「タグ・ライブラリ」を参照

## き

---

キャッシング・サポート、概要、2-16  
行セットのキャッシング、4-12  
行のプリフェッチ、4-12

## く

---

クラスのネーミング、トランスレータ、7-5  
クラスパス  
  JSP クラスパス機能、3-4  
クラスパス構成 (JServ)、B-2  
グローバル化・サポート  
  JSP ライターのキャラクタ・セット、9-6  
  translate\_params に基づかないサンプル、B-29  
  translate\_params に基づくサンプル、B-27  
  概要、9-1  
  コンテンツ・タイプの設定 (静的)、9-2  
  コンテンツ・タイプの設定 (動的)、9-5  
  マルチバイト・パラメータ・エンコード、9-7  
グローバル・インクルード (Oracle の拡張機能)  
  translate\_params からの移行時に使用、B-31  
  一般的な使用、7-8

## こ

---

構成  
  Enterprise Manager での JSP の構成、3-32  
  JServ の JSP 構成パラメータ、B-4  
  JSP 関連の OC4J 構成パラメータ、3-28  
  JSP 関連の OC4J 構成パラメータの設定、3-30  
  JSP 構成パラメータ、3-14  
  JSP 構成パラメータの設定、3-27  
  JSP コンテナの設定、3-13  
  主な JAR ファイルと ZIP ファイル、3-12  
  主な OC4J 構成ファイル、3-31  
  パラメータの設定、JServ、B-15  
  ファイル名拡張子のマッピング、JServ、B-3  
構文 (概要)、1-6  
コード、トランスレータで生成、7-2  
コメント (JSP コード内)、1-11  
コンテキスト相対パス、1-27  
コンテキスト・パス、3-2  
コンテナ  
  JSP コンテナ、1-24  
  サーブレット・コンテナ、A-4

コンテンツ・タイプの設定  
  静的 (page ディレクティブ)、9-2  
  動的 (setContentype メソッド)、9-5  
コンパイル  
  javaccmd 構成パラメータ、3-21  
  ojspc の noCompile オプション、7-26  
  use\_old\_compiler 構成パラメータ、3-25  
  インプロセスとアウトプロセスの比較、3-6  
  デフォルト設定、関連オプション、3-6

## か

---

サービス・メソッド、JSP、A-12  
サーブレット  
  JSP ページを使用したサーブレットのラップ、B-20  
  アプリケーションのライフ・サイクル管理、A-8  
  技術的なバックグラウンド、A-2  
  サーブレット・オブジェクト、A-9  
  サーブレット構成オブジェクト、A-11  
  サーブレット・コンテキスト、A-7  
  サーブレット・コンテキスト・オブジェクト、A-10  
  サーブレット・コンテナ、A-4  
  サーブレット・セッション、A-5  
  サーブレット・テクノロジーの概要、A-2  
  サーブレットのインタフェース、A-3  
  サーブレットの起動、A-8  
  セッション・オブジェクト、A-10  
  セッション共有、JSP、JServ、B-19  
  リクエスト・オブジェクトとレスポンス・オブジェクト、A-9  
サーブレット 2.0 環境  
  globals.jsa を使用した追加サポート、B-32  
  アプリケーション・ルート機能に関する JSP コンテナ機能、B-18  
サーブレット・コンテキスト  
  概要、A-7  
  サーブレット・コンテキスト・オブジェクト、A-10  
サーブレット・コンテナ、A-4  
サーブレット・セッション  
  HttpSession インタフェース、A-5  
  セッション・トラッキング、A-6  
サーブレットと JSP 間の相互作用  
  JSP からのサーブレットの起動、4-2  
  サーブレットからの JSP の起動、リクエスト・ディスプレイパッチャ、4-3  
  サンプル・コード、4-5

- データの受渡し、JSP からサーブレット, 4-3
- データの受渡し、サーブレットから JSP, 4-4
- サーブレット・パス, 3-2
- 最適化、「パフォーマンス」を参照
- サンプル・アプリケーション
  - HttpSessionBindingListener のサンプル, 4-19
  - IterationTag の定義と使用, 8-52
  - JSP とサーブレット間の相互作用, 4-5
  - SQLJ の例, 4-15
  - XML ビューへの変換, 5-16
  - カスタム・タグの定義と使用, 8-56
  - グローバル化、translate\_params に基づかない, B-29
  - グローバル化、translate\_params に基づく, B-27
- 従来の構文と XML 構文の比較, 5-11
- デモの入手先、OTN, 1-1

## し

---

- 式
  - XML 式要素, 5-9
  - 式の構文, 1-10
- 事前変換
  - ojspc ユーティリティ, 7-12
  - 実行を伴わない、一般的, 7-39
- 実行時の再変換または再ロード, 3-5
- 出力ファイル
  - ojspc の d オプション (バイナリ・ファイルの場所), 7-23
  - ojspc の srcdir オプション (ソース・ファイルの場所), 7-30
  - 格納場所, 7-7
  - トランスレータで生成, 7-6
  - 場所および関連オプション、ojspc, 7-32
- 出力名、規則, 7-3
- ショートカット URI (タグ・ライブラリ), 8-18

## す

---

- スクリプト変数 (タグ・ライブラリ)
  - TEI クラスを使用した宣言, 8-44
  - TLD を使用した宣言, 8-43
  - 使用, 8-41
  - スコープ, 8-42

- スクリプト要素
  - 概要, 1-9
  - コメント, 1-11
  - 式, 1-10
  - スクリプトレット, 1-10
  - 宣言, 1-9
- スクリプトレット
  - JavaBeans との比較, 6-2
  - XML スクリプトレット要素, 5-9
  - スクリプトレットの構文, 1-10
- スコープ (JSP オブジェクト), 1-14

## せ

---

- 生成されるコード、トランスレータ, 7-2
- 生成される出力名、トランスレータ, 7-3
- 静的なインクルード
  - ディレクティブ, 1-8
  - 動的なインクルードとの比較, 6-2
  - ロジック手法, 6-3
- 静的なテキスト
  - external\_resource パラメータ, 3-19
  - 外部リソース、ojspc の extres オプション, 7-25
  - 外部リソース・ファイル, 6-9
  - 大量の静的なコンテンツに対する対処, 6-9
  - メンバー変数内, 7-3
- セキュリティ
  - JServ での考慮事項, B-24
  - 一般的な考慮事項, 3-7
- セッション・イベント
  - globals.jsa の使用, B-39
  - HttpSessionBindingListener の使用, 4-18
- セッション・オブジェクト
  - 概要, A-10
- セッション共有、概要、JSP とサーブレット間、JServ, B-19
- セッション・トラッキング, A-6
- 接続のキャッシング、概要, 4-10
- 宣言
  - XML 宣言要素, 5-9
  - グローバル宣言、globals.jsa, B-41
  - メソッド変数とメンバー変数の比較, 6-10
  - メンバー変数, 1-9

## そ

---

相互作用、JSP とサーブレット間、4-2

操作タグ

forward タグ、1-19

getProperty タグ、1-18

include タグ、1-19

JSP XML ページ、5-10

param タグ、1-18

plugin タグ、1-21

setProperty タグ、1-17

useBean タグ、1-16

標準アクションの概要、1-15

ソース・ファイルの場所、ojspc の srcdir オプション、7-30

## た

---

ダイナミック・モニタリング・サービス、「DMS」を参照

タイムアウト設定

JServ、B-12

OC4J、3-28

タグ・ハンドラ (タグ・ライブラリ)

empty アクション、8-29

JSP 1.1 と 1.2 の間の変更、8-6

non-empty アクション、8-29

OC4J のタグ・ハンドラ・インスタンスの再利用 / プーリング、8-38

OC4J のタグ・ハンドラのコード生成、8-41

外部タグ・ハンドラへのアクセス、8-37

概要、8-26

タグ・ハンドラ・クラスのサンプル、8-53、8-57

単純タグ・ハンドラ、ボディの反復あり、8-32

単純タグ・ハンドラ、ボディの反復なし、8-31

ボディ・コンテンツへのアクセス、8-34

ボディ処理用の定数、8-30

ボディの処理、8-28

タグ補足情報クラス (タグ・ライブラリ)

一般的使用、getVariableInfo() メソッド、8-44

タグ補足情報クラスのサンプル、8-58

タグ・ライブラリ

IterationTag、完全な例、8-52

JAR ファイル内の単一のタグ・ライブラリ、8-15

taglib ディレクティブ、8-13

TLD の永続的なキャッシング、8-19

web.xml の使用、8-18

アプリケーション間で共有、8-19

機能の概要、1-23

実行時とコンパイル時の実装の比較、8-61

スクリプト変数、8-41

タグ・ハンドラ、8-26

タグ・ライブラリ・ディスクリプタ、8-7

タグ・ライブラリ名前空間 (XML 構文)、5-7

タグ・ライブラリのリスナー、8-50

タグ・ライブラリ・バリデータ・クラス、8-46

単一の JAR ファイル内での複数のタグ・ライブラリ、8-16

定義と使用、完全な例、8-56

名前空間、XML サポート、5-7

標準の実装の概要、8-2

標準フレームワーク、8-2

方針、作成時期、6-5

予約済の場所、8-19

タグ・ライブラリ・ディスクリプタ

JAR ファイル内の単一の TLD の指定、8-15

JSP 1.1 と 1.2 の間の変更、8-5

listener 要素とサブ要素、8-13

taglib ディレクティブ、8-13

tag 要素とサブ要素、8-9

TLD 妥当性チェック構成パラメータ、3-22

TLD 妥当性チェック用の ojspc のオプション、7-26

validator 要素とサブ要素、8-13

web.xml にショートカット URI を定義、8-18

永続的なキャッシング、8-19

各 TLD の指定、8-15

機能の概要、8-7

サンプル・ファイル、8-55、8-59

単一の JAR ファイル内に複数のタグ・ライブラリがある場合の TLD の指定、8-16

タグ・ライブラリ・バリデータ・クラス、8-46

妥当性チェック、タグ・ライブラリ、8-46

単純タグ・ハンドラ (タグ・ライブラリ)

ボディの反復あり、8-32

ボディの反復なし、8-31

## ち

---

チェッカ・ページ、6-7

## て

---

### ディレクティブ

forgive\_dup\_dir\_attr 構成パラメータ, 3-20

include ディレクティブ, 1-8

ojspc forgiveDupDirAttr オプション, 7-25

page ディレクティブ, 1-7

taglib ディレクティブ, 1-9

XML ディレクティブ要素, 5-8

概要, 1-7

グローバルなディレクティブ、globals.jsa, B-41

ディレクトリの別名変換、「別名変換」を参照

データ・アクセス機能, 4-6

テキスト要素 (XML 構文), 5-10

### デバッグ

debug、ojspc のオプション, 7-22

debug\_mode 構成パラメータ, 3-18

emit\_debuginfo 構成パラメータ, 3-19

JDeveloper を使用, 2-12

デプロイ、一般的な考慮事項

JDeveloper によるページのデプロイ, 7-36

WAR デプロイ, 7-34

概要, 7-34

実行を伴わない一般的な事前変換, 7-39

バイナリ・ファイルのみのデプロイ, 7-40

バッチ事前変換のための ojspc, 7-38

ページ事前変換のための ojspc, 7-37

デモの入手先、OTN, 1-1

テンプレート・データ, 5-2

## と

---

### 動的なインクルード

JServ に対する特別なサポート, B-19

静的なインクルードとの比較, 6-2

操作タグ, 1-19

大量の静的なコンテンツ, 6-9

ロジック手法, 6-3

動的な転送、JServ に対する特別なサポート, B-19

### トランスレータ

Oracle JSP のグローバル・インクルード, 7-8

出力ファイルの格納場所, 7-7

生成されるクラス名, 7-5

生成されるコードの機能, 7-2

生成される名前、一般規則, 7-3

生成されるパッケージ名, 7-4

生成されるファイル, 7-6

生成されるメンバー変数、静的なテキスト, 7-3

## な

---

名前空間 (XML 構文), 5-7

## ね

---

ネーミング規則、JSP ファイル, 3-10

## は

---

バイナリ・データ、JSP で回避する理由, 6-17

バイナリ・ファイルのデプロイ, 7-40

バイナリ・ファイルの場所、ojspc の d オプション,  
7-23

パッケージのインポート、デフォルト, 3-9

パッケージのネーミング

ojspc の packageName オプション, 7-27

トランスレータ, 7-4

バッチ更新、「バッチの更新」を参照

バッチ事前変換

ojspc -batchMask オプション, 7-21

ojspc -deleteSource オプション, 7-23

ojspc -output オプション, 7-27

ojspc バッチ機能の概要, 7-14

バッチの更新, 4-11

パフォーマンス

OC4J および Oracle Application Server の機能, 3-9

構成に関する考慮事項, 3-8

事前変換の使用, 3-9

プログラム上の考慮事項, 3-7

## ひ

---

### ヒント

JavaBeans とスクリプトレットの比較, 6-2

JSP での空白の保持, 6-15

JSP によるバイナリ・データの回避, 6-17

page ディレクティブの特性, 6-12

サーブレット・ラッパーとしての JSP ページ, B-20

静的なインクルードと動的なインクルードの比較,  
6-2

対処、大量の静的なコンテンツ, 6-9

タグ・ライブラリの作成時期, 6-5

チェック・ページの使用, 6-7  
メソッド変数宣言とメンバー変数宣言の比較, 6-10

## ふ

---

ファイル  
主な JAR ファイルと ZIP ファイル, 3-12  
格納場所、トランスレータの出力, 7-7  
トランスレータで生成, 7-6  
場所、ojspc の d オプション, 7-23  
場所、ojspc の srcdir オプション, 7-30  
ファイルのネーミング規則、JSP ファイル, 3-10  
プリフェッチ行、「行のプリフェッチ」を参照  
プログラミングに関する考慮事項  
一般的な方針, 6-2  
その他のヒント, 6-6  
文のキャッシング, 4-11

## へ

---

ページ実装クラス  
概要, 1-26  
生成されるコード, 7-2  
ページ相対パス, 1-27  
別名変換、JServ  
alias\_translation 構成パラメータ, B-10  
概要, B-22  
変換、オンデマンド (実行時), 1-26

## ま

---

マルチバイト・パラメータ・エンコード  
JServ 環境, B-24  
一般 / 標準, 9-7

## め

---

明示的な JSP オブジェクト, 1-12  
メソッド変数宣言, 6-10  
メンバー変数宣言, 6-10

## よ

---

予約済の場所 (タグ・ライブラリ), 8-19

## り

---

リクエスト・オブジェクト  
概要, A-9  
リクエスト・ディスパッチャ (JSP とサーブレット間  
の相互作用), 4-3  
リスナー、タグ・ライブラリ, 8-50  
リソース管理  
JSP 拡張機能の概要, 4-23  
標準セッションの管理, 4-18

## る

---

ルート要素 (XML 構文), 5-7

## れ

---

レスポンス・オブジェクト  
概要, A-9