BEA WebLogic Server™

WebLogic Server における Web サーバ プラグインの使い方

BEA WebLogic Server バージョン 7.0
マニュアルの改訂 : 2003 年 7 月 21 日

著作権

限定的権利条項

商標または登録商標

WebLogic Server における Web サーバ プラグインの使い方

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2003 年 7 月 21 日	BEA WebLogic Server バージョン 7.0

目次

3. Microsoft Internet Information Server (IIS) プラグインのインストールとコンフィグレーション

このマニュアルの内容

このマニュアルでは、サードパーティ製管理サーバにリクエストをプロキシするためのプラグインの使い方について説明します。構成は次のとおりです。

- 第 1 章「WebLogic Server における Web サーバ プラグイン使用の概要」では、WebLogic Server で利用可能なプラグインについて説明します。
- 第 2 章「Apache HTTP Server プラグインのインストールとコンフィグレーション」では、WebLogic Server Apache プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 3 章「Microsoft Internet Information Server (IIS) プラグインのインストールとコンフィグレーション」では、Microsoft Internet Information Server 用の WebLogic Server プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 4 章「Netscape Enterprise Server (NES) プラグインのインストールとコンフィグレーション」では、Netscape Enterprise Server (プロキシ) プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 5 章「Web サーバ プラグインのパラメータ」では、Web サーバ プラグインのパラメータについて説明します。
- 付録 A「別の Web サーバへのリクエストのプロキシ」では、HTTP リクエストを他の Web サーバに転送するプロキシとして、WebLogic Server を機能させる方法について説明します。

対象読者

このマニュアルは、WebLogic Server プラットフォームとその各種サブシステムを管理するシステム管理者を主な対象としています。

e-docs Web サイト

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

このマニュアルの印刷方法

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

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

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

サポート情報

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

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

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

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

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

斜体の等幅テキスト	コード内の変数を示す。 例： `String CustomerName;`

表記法	適用
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例： LPT1 BEA_HOME OR
{ }	構文の中で複数の選択肢を示す。
[ ]	構文の中で任意指定の項目を示す。 例： `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例： `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 WebLogic Server における Web サーバ プラグイン使用の概要

以下の節では、WebLogic Server で使用するために BEA Systems が提供するプラグインについて説明します。

プラグインとは

プラグインとは、小さなソフトウェア プログラムのことで、開発者はこれを使って WebLogic Server 実装を拡張します。プラグインを使うことにより、WebLogic Server は異なる Web サーバ上にデプロイされているアプリケーションと通信することができます。その結果、WebLogic Server では WebLogic Server の動的な機能を必要とするリクエストを処理できます。それらのリクエストは、通常は、動的 HTML または JSP (JavaServer Pages) で最高のサービスを提供できるリクエストです。

WebLogic Server 付属のプラグイン

WebLogic Server 6.1 SP6、7.0 SP5、および 8.1 SP2 以降において、WebLogic Server プラグインは WebLogic Server の任意のバージョン (5.1 を含む) にプロキシすることが確認されています。

WebLogic Server には、以下の Web サーバ用のプラグインがあります。

- Apache HTTP Server

- Microsoft Internet Information Server
- Netscape Enterprise Server

2 Apache HTTP Server プラグインのインストールとコンフィグレーション

以下の節では、Apache HTTP Server プラグインをインストールしてコンフィグレーションする方法を説明します。

Apache HTTP Server プラグインの概要

Apache HTTP Server プラグインを使用すると、Apache HTTP サーバから WebLogic Server へリクエストをプロキシできます。このプラグインは、WebLogic Server の動的な機能を必要とするリクエストを WebLogic Server が処理できるようにすることによって Apache を拡張します。

このプラグインは、Apache サーバが静的ページを提供している環境で使用されることを想定しています。ドキュメント ツリーの他の部分 (HTTP サーブレットや JavaServer Pages によって最も適切な状態で生成される動的ページ) は、別のプロセス (おそらく別のホスト) で動作している WebLogic Server に委託されます。それでも、エンド ユーザ (ブラウザ) では、WebLogic Server に委託される HTTP リクエストは同じソースから来ているものと認識されます。

HTTP トンネリング (企業のファイアウォールを経由した HTTP リクエストおよび応答のアクセスを可能にする技術) も、このプラグインを通じて機能でき、ブラウザ以外のクライアントが WebLogic Server サービスにアクセスすることを可能にします。

Apache HTTP Server プラグインは、Apache HTTP サーバ内で Apache モジュールとして機能します。Apache モジュールは起動時に Apache サーバによってロードされ、特定の HTTP リクエストがそこに委託されます。Apache モジュールは、HTTP サーブレットと似ていますが、プラットフォームにネイティブなコードで記述されています。

Apache バージョン 1.3.x の制限事項

Apache HTTP Server のバージョン 1.3.x では、以降のバージョンにはない WebLogic Server プラグインに関する制限事項があります。

キープアライブ接続はサポートされない

Apache HTTP Server プラグインのバージョン 1.3.x は、リクエストごとにソケットを作成し、応答を読み込んでからソケットを閉じます。Apache HTTP サーバは多重処理されるため、WebLogic Server と Apache HTTP Server プラグインの間では接続プールとキープアライブ接続はサポートされていません。

矛盾した状態

Apache HTTP Server はマルチプロセス アーキテクチャであり、プラグインの状態は複数の子プロセス間で同期されません。次のような問題が発生する可能性があります。

- クラスタ化された環境では、DynamicServerList がすべてのプラグイン プロセスで最新の状態になっていないために、アクセスできない WebLogic Server インスタンスにプラグインがリクエストをディスパッチする場合があります。
- クラスタ化されていない環境では、WebLogic Server インスタンスの再起動後に作成されたセッションの維持が失われる場合があります。一部のプラグイン プロセスには再起動されたサーバの新しい JVMID がなく、そのサーバを不明な JVMID として扱うためです。

このような矛盾を一時的に修正するには、Apache サーバを再起動するか HUP 信号 (kill -HUP) を送信して、すべてのプラグイン プロセスを更新します。

この問題を回避するには、Apache 2.0.x にアップグレードし、httpd.conf で MaxSpareServers=1 を設定して、マルチスレッドおよびシングルプロセス モデルを使用するように Apache をコンフィグレーションします。httpd.conf の編集の詳細については、2-10 ページの「Apache HTTP Server プラグインのコンフィグレーション」を参照してください。

Apache バージョン 2.0 のキープアライブ接続

Apache HTTP Server プラグインのバージョン 2.0 は、WebLogic Server との接続の再利用可能なプールを使用してパフォーマンスを向上させます。このプラグインは、同じクライアントからの後続リクエストにプール内の同じ接続を再利用することで、WebLogic Server との間で HTTP 1.1 キープアライブ接続を実装します。接続が 30 秒 (またはユーザ定義の時間) を超えて非アクティブな場合、その接続は閉じて、プールに返されます。この機能は、必要に応じて無効にできます。詳細については、「KeepAliveEnabled」を参照してください。

リクエストのプロキシ

このプラグインは、指定されたコンフィグレーションに基づいてリクエストを WebLogic Server にプロキシします。リクエストは、リクエストの URL (または URL の一部) に基づいてプロキシできます。この方法は、パスに基づくプロキシです。要求されたファイルの MIME タイプに基づいてプロキシすることもできます。あるいは、前述の方法を組み合わせて利用することもできます。リクエストが両方の基準に一致する場合、そのリクエストはパスを基準にプロキシされ

ます。リクエストの種類ごとに、プラグインの補足的な動作を定義する追加パラメータを指定することもできます。詳細については、2-10 ページの「Apache HTTP Server プラグインのコンフィグレーション」を参照してください。

動作確認

Apache HTTP Server プラグインは、Linux、Solaris、AIX、Windows および HPUX11 の各プラットフォームでサポートされています。プラグインはすべてのリリースのすべてのオペレーティング システムでサポートされているわけではありません。Apache の特定のバージョンのプラットフォーム サポートについては、「WebLogic Platform サポート対象のコンフィグレーション」を参照してください。

Apache HTTP Server プラグインのインストール

Apache HTTP Server プラグインは、Apache HTTP サーバで Apache モジュールとしてインストールします。モジュールは、動的共有オブジェクト (DSO) または静的リンク モジュールとしてインストールされます。静的リンク モジュールとしてのインストールは、Apache バージョン 1.3.x でのみ可能です。この節では、DSO および静的リンク モジュールについてそれぞれの場合のインストール手順を紹介します。

Apache HTTP Server プラグインの動的共有オブジェクトとしてのインストール

Apache HTTP Server プラグインを動的共有オブジェクトとしてインストールするには、次の操作を行います。

1. 使用しているプラットフォーム用の共有オブジェクト ファイルを見つけます。

   Apache プラグインは、Solaris、Linux、Windows、および HPUX11 プラットフォームでの使用を考慮して共有オブジェクト (.so) として配布されます。各共有オブジェクト ファイルは、プラットフォーム、クライアントと Apache の間での SSL の使用と不使用、および SSL 暗号化の強度 (通常または 128 ビット) に応じて別々のバージョンとして配布されます。128 ビット バージョンは、128 ビット バージョンの WebLogic Server をインストールする場合のみインストールされます。共有オブジェクト ファイルは、WebLogic Server の以下のディレクトリに配置されています。WL_HOME は WebLogic プラットフォームの最上位のインストール ディレクトリであり、Server ディレクトリには WebLogic Server のインストール ファイルが格納されます。

   Solaris

   ```
   WL_HOME/Server/lib/solaris
   ```

   Linux

   ```
   WL_HOME/Server/lib/linux
   ```

   Windows（Apache 2.0 のみ）

   ```
   WL_HOME\Server\bin\mod_wl_20.so
   ```

   HPUX11

   ```
   WL_HOME/Server/lib/hpux11
   ```

   警告： Apache 2.0.x サーバを HP-UX11 で実行する場合は、Apache サーバをビルドする前に下で示されている環境変数を設定します。HP-UX ではリンクされたライブラリがロードされる順序に問題があるので、ビルドの前にロード順が環境変数として事前設定されていないとコア ダンプが生じることがあります。以下の環境変数を設定します。

   ```
   export EXTRA_LDFLAGS="-lstd -lstream -lCsup -lm -lcl -ldld
   -lpthread"
   ```

   `configure`、`make`、および `make install` のステップに進みます。

   ```
   ./configure --prefix=$INSTALLATION_DIRECTORY --enable-so
   --with-mpm=worker

   make

   make install
   ```

Apache サーバのビルドとコンフィグレーションの詳細については、Apache HTTP Server のマニュアルを参照してください。

次の表を基準に適切な共有オブジェクトを選択してください。

Apache のバージョン	通常強度の暗号化	128 ビットの暗号化
標準の Apache バージョン 1.x	`mod_wl.so`	`mod_wl128.so`
Apache w/ SSL/EAPI バージョン 1.x （Stronghold、modssl など）	`mod_wl_ssl.so`	`mod_wl128_ssl.so`
Apache + Raven バージョン 1.x Raven で配布される重要なパッチによってプラグインが標準の共有オブジェクトと互換性がなくなるので必要となる。	`mod_wl_ssl_raven.so`	`mod_wl128_ssl_raven.so`
標準の Apache バージョン 2.x	`mod_wl_20.so`	`mod_wl28_20.so`

Apache 2.0.39 用のプラグインは WebLogic Server 配布キットには同梱されていません。下記の場所からダウンロードできます。

http://dev2dev.bea.com/codelibrary/code/apache.jsp

2. 共有オブジェクトを有効にします。

Apache HTTP Server プラグインは、Apache 動的共有オブジェクト (DSO) として Apache HTTP サーバにインストールされます。Apache の DSO サポートは、`mod_wl.so` のロード前に有効にする必要のある `mod_so.c` というモ

ジュールに基づいています。提供されるスクリプトを使用して Apache をインストールした場合、mod_so.c は既に有効になっているはずです。
mod_so.c が有効であることを確認するには、以下のいずれかのコマンドを実行します。

APACHE_HOME\bin\httpd -l (Apache 1.x の場合)

APACHE_HOME\bin\Apache -l (Apache 2.x の場合)

APACHE_HOME は、Apache HTTP サーバのインストール ディレクトリです。

このコマンドでは、有効なすべてのモジュールのリストが表示されます。mod_so.c がリストにない場合は、ソース コードから Apache HTTP サーバを構築して、以下のオプションがコンフィグレーションされているようにします。

```
...
--enable-module=so
--enable-rule=SHARED_CORE
...
```

3. apxs (APache eXtenSion、http://httpd.apache.org/docs/programs/apxs.html の「Manual Page: apxs」を参照) というサポート プログラムを使用して Apache 1.x サーバに Apache HTTP Server プラグインをインストールします 次のコマンドを発行すると、

   ```
   $ apxs mod_so.c
   ```

   Apache ソース ツリーの外で動的共有オブジェクトベースのモジュールがビルドされ、httpd.conf ファイルに次の行が追加されます。

   ```
   AddModule mod_so.c
   ```

   Apache 2.x の場合は、apxs を実行するのではなく、mod_wl_20.so ファイルを *APACHE_HOME*\modules ディレクトリにコピーします。詳細については、「Apache HTTP サーバ バージョン 2.0 ドキュメント」を参照してください。

4. Apache サーバに対する WebLogic Server 拡張機能の weblogic_module をアクティブにします。
   - Apache 1.x の場合は、WebLogic Server でコマンド シェルを使用して、使用しているプラットフォーム用の共有オブジェクトがあるディレクトリに移動し、次のコマンドを発行して weblogic_module をアクティブにします。Perl がインストールされていないと、この Perl スクリプトは実行できません。

     ```
     perl APACHE_HOME\bin\apxs –i –a –n weblogic mod_wl.so
     ```

このコマンドでは、mod_wl.so ファイルが *APACHE_HOME*\libexec ディレクトリにコピーされます。さらに、httpd.conf ファイルに weblogic_module に関する指示が 2 行追加され、このモジュールがアクティブ化されます。Apache 1.x サーバの *APACHE_HOME*/conf/httpd.conf ファイルに以下の行が追加されていることを確認してください。

```
LoadModule weblogic_module    libexec/mod_wl.so
AddModule mod_weblogic.c
```

- Apache 2.x の場合は、手作業で次の行を *APACHE_HOME*/conf/httpd.conf ファイルに追加します。

```
LoadModule weblogic_module     modules/mod_wl_20.so
```

5. 2-10 ページの「Apache HTTP Server プラグインのコンフィグレーション」の説明に従って、Apache httpd.conf コンフィグレーション ファイルで追加パラメータをコンフィグレーションします。httpd.conf ファイルでは、Apache HTTP Server プラグインの動作をカスタマイズできます。

6. 以下のいずれかのコマンドで、*APACHE_HOME*\conf\httpd.conf ファイルの構文を検証します。

   APACHE_HOME\bin\apachect1 configtest (Apache 1.x の場合)

   APACHE_HOME\bin\Apache -t (Apache 2.x の場合)

   このコマンドの出力では、httpd.conf ファイルのエラーが示されます。

注意： Apache -t は、Windows 上の Apache 2.x では有効なコマンドです。しかし、HP-UX 上の Apache 2.x の場合、Apache コマンド ファイルではなく apachectl を使用するので、Apache -t は有効なコマンドではありません。

7. Weblogic Server を再起動します。

8. Apache HTTP サーバを起動 (コンフィグレーションを変更した場合は再起動) します。

9. ブラウザを開き、Apache サーバの URL + 「/weblogic/」を設定して Apache プラグインをテストします。この設定では、WebLogic Server でデフォルト Web アプリケーションとして定義されている、デフォルトの WebLogic Server HTML ページ、ウェルカム ファイル、またはデフォルト サーブレットが開くはずです。次に例を示します。

   http://myApacheserver.com/weblogic/

Apache HTTP Server プラグインの静的リンク モジュールとしてのインストール

Apache HTTP Server プラグインを静的リンク モジュールとしてインストールするには、次の操作を行います。

1. 使用しているプラットフォーム用のリンク ライブラリ ファイルを見つけます。

   各ライブラリ ファイルは、プラットフォームおよび SSL 暗号化の強度 (通常または 128 ビット) に応じて別々のバージョンとして配布されます。128 ビット バージョンは、128 ビット バージョンの WebLogic Server をインストールする場合のみインストールされます。ライブラリ ファイルは、WebLogic Server の以下のディレクトリに配置されています。

   Solaris
   ```
   WL_HOME/Server/lib/solaris
   ```
   Linux
   ```
   WL_HOME/Server/lib/linux
   ```
   HPUX11
   ```
   WL_HOME/Server/lib/hpux11
   ```

   次の表を基準に適切な共有オブジェクトを選択してください。

Apache のバージョン	通常強度の暗号化	128 ビットの暗号化
標準の Apache バージョン 1.3.x	`libweblogic.a`	`libweblogic128.a`

   Gnu C Compiler (gcc) を使用する場合は、gcc 2.95.x が推奨バージョンです。

2. 次のコマンドで Apache プラグイン配布キットを復元します。

   ```
   tar -xvf apache_1.3.x.tar
   ```

3. 復元した配布キットの中で `src/modules` ディレクトリに移動します。

4. `weblogic` というディレクトリを作成します。

5. `Makefile.libdir`、`Makefile.tmpl` を WebLogic Server の `lib` ディレクトリから `src\modules\weblogic` にコピーします。
6. `libweblogic.a` (128 ビットのセキュリティを使用する場合は `libweblogic128.a`) を静的リンク ライブラリの格納されている同じディレクトリ (手順 1. を参照) から `src\modules\weblogic` にコピーします。
7. 通常の強度の暗号化を使用する場合は、Apache 1.3 のホーム ディレクトリから次のコマンドを実行します。

   ```
   configure --activate-module=src\modules\weblogic\libweblogic.a
   ```

8. 128 ビットの暗号化を使用する場合は、Apache 1.3 のホーム ディレクトリから次のコマンドを (1 行で) 実行します。

   ```
   configure--activate-module=
    src\modules\weblogic\libweblogic128.a
   ```

9. 次のコマンドを実行します。

   ```
   make
   ```

10. 次のコマンドを実行します。

    ```
    make install
    ```

11. 「Apache HTTP Server プラグインの動的共有オブジェクトとしてのインストール」の手順 4 から 8 を行います。

Apache HTTP Server プラグインのコンフィグレーション

Apache HTTP サーバにプラグインをインストールした後は、`httpd.conf` ファイルを編集して Apache プラグインをコンフィグレーションします。`httpd.conf` ファイルを編集して、プラグイン用のネイティブ ライブラリを Apache モジュールとしてロードしなければならないことを Apache Web サーバに通知し、どのリクエストをそのモジュールで処理しなければならないかを記述します。

httpd.conf ファイルの編集

Apache HTTP サーバの `httpd.conf` ファイルを編集し、Apache HTTP Server プラグインをコンフィグレーションします。

1. `httpd.conf` ファイルを開きます。このファイルは、*`APACHE_HOME`*`\conf\httpd.conf` にあります (`APACHE_HOME` は Apache HTTP サーバのルート ディレクトリ)。
2. Apache 1.x の場合は、`apxs` ユーティリティの実行で以下の 2 行が `httpd.conf` ファイルに追加されていることを確認します。

   ```
   LoadModule weblogic_module    libexec\mod_wl.so
   AddModule mod_weblogic.c
   ```

3. Apache 2.x の場合は、次の行を httpd.conf ファイルに追加します。

   ```
   LoadModule weblogic_module   modules\mod_wl_20.so
   ```

4. 以下のいずれかを定義する `IfModule` ブロックを追加します。

   クラスタ化されていない WebLogic Server の場合

   WebLogicHost および WebLogicPort パラメータ

   WebLogic Server のクラスタの場合

   WebLogicCluster パラメータ

   次に例を示します。

   ```
   <IfModule mod_weblogic.c>
     WebLogicHost myweblogic.server.com
     WebLogicPort 7001
   </IfModule>
   ```

5. MIME タイプを基準にリクエストをプロキシする場合は、`MatchExpression` 行も `IfModule` ブロックに追加します。リクエストは、MIME タイプに加えて、または MIME タイプではなくパスを基準にしてもプロキシできます。パスを基準としたプロキシは、MIME タイプを基準としたプロキシに優先します。パスによるプロキシをコンフィグレーションするには、手順 6. を参照してください。

   たとえば、クラスタ化されていない WebLogic Server に対する次の `IfModule` ブロックでは、MIME タイプが `.jsp` であるすべてのファイルがプロキシされます。

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

次のように、複数の MatchExpressions を使用することもできます。

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

MIME タイプを基準に WebLogic Server のクラスタにリクエストをプロキシする場合は、WebLogicHost および WebLogicPort パラメータの代わりに WebLogicCluster パラメータを使用します。次に例を示します。

```
<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

6. パスを基準にリクエストをプロキシする場合は、Location ブロックおよび SetHandler 文を使用します。SetHandler は、Apache HTTP Server プラグイン モジュールのハンドラを指定します。たとえば、次の Location ブロックでは、URL に /weblogic の含まれるすべてのリクエストがプロキシされます。

```
<Location /weblogic>
                SetHandler weblogic-handler
</Location>
```

7. Apache HTTP Server プラグインの追加パラメータを定義します。

   Apache HTTP Server プラグインは、5-2 ページの「Web サーバ プラグインの一般的なパラメータ」にリストされているパラメータを認識します。Apache HTTP Server プラグインの動作を修正するには、以下のいずれかの場所でそれらのパラメータを定義します。

   - Location ブロック (パスを基準にしたプロキシに適用されるパラメータの場合)
   - IfModule ブロック (MIME タイプを基準にしたプロキシに適用されるパラメータの場合)

httpd.conf ファイルの編集に代わる別の方法

- 2-11 ページの「httpd.conf ファイルの編集」の方法とは別に、IfModule ブロックにインクルードされる weblogic.conf という独立したファイルでパラメータを定義することもできます。このインクルード ファイルを使用すると、コンフィグレーションをモジュール化できます。次に例を示します。

  ```
  <IfModule mod_weblogic.c>
    # パラメータを定義する WebLogic 用コンフィグレーション ファイル
    Include conf/weblogic.conf
  </IfModule>
  ```

  注意： インクルード ファイルでのパラメータの定義は、Apache HTTP Server プラグインと WebLogic Server の間で SSL を使用する場合はサポートされません。

- パラメータはそれぞれ独立した行に入力します。パラメータとその値の間に「=」を挿入しないでください。次に例を示します。

  ```
  PARAM_1 value1
  PARAM_2 value2
  PARAM_3 value3
  ```

- IfModule ブロックの MatchExpression で指定された MIME タイプと Location ブロックで指定されたパスの両方にリクエストが一致する場合は、Location ブロックで指定された動作が優先します。

- CookieName パラメータは、IfModule ブロックで定義する必要があります。

- <location> ブロックに加えて <files> ブロックもリクエストの照合に使用し、仮想ホストで Stronghold SSL (Apache ベースの商用 Web サーバ) を使用している場合、MatchExpression は無視され、<files> ブロックと <location> ブロックで定義されているルールがリクエストに適用されます。Stronghold を使用していない場合は、<location> ブロックのルールで <files> ブロックのルールが打ち消されます。

- <VirtualHost> ブロックを使用する場合は、各仮想ホストのすべてのコンフィグレーション パラメータ (MatchExpression など) をその <VirtualHost> ブロックに配置する必要があります。

- <files> ブロックではなく MatchExpression 文を使用することをお勧めします。

Apache HTTP Server httpd.conf ファイルのテンプレート

この節では、サンプルの `httpd.conf` ファイルを紹介します。このサンプルをテンプレートとして使用し、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。Apache HTTP サーバでは大文字と小文字は区別されません。また、`LoadModule` および `AddModule` 行は、`apxs` ユーティリティによって自動的に追加されます。

```
##########################################################

APACHE-HOME/conf/httpd.conf file
##########################################################

LoadModule weblogic_module    libexec/mod_wl.so

AddModule mod_weblogic.c

<Location /weblogic>
                SetHandler weblogic-handler
  PathTrim /weblogic
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<Location /servletimages>
                SetHandler weblogic-handler
  PathTrim /something
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

httpd.conf コンフィグレーション ファイルのサンプル

httpd.conf ファイルの location ブロックでパラメータを定義する代わりに、必要に応じて、httpd.conf ファイルの IfModule によってロードされる weblogic.conf ファイルを使用できます。次の例をテンプレートとして使用して、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。

WebLogic クラスタを使用した例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる (WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く )

<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
  MatchExpression *.jsp
</IfModule>
#######################################################
```

複数の WebLogic クラスタを使用した例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる (WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く )

<IfModule mod_weblogic.c>
  MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=7001|Debug=ON
  MatchExpression *.html WebLogicCluster=myHost1:7282,myHost2:7283|ErrorPage=
    http://www.xyz.com/error.html
</IfModule>
```

WebLogic クラスタを使用しない例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる (WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く )

<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

IP ベースの仮想ホスティングのコンフィグレーション例

```
NameVirtualHost 172.17.8.1
<VirtualHost goldengate.domain1.com>
WebLogicCluster tehama1:4736,tehama2:4736,tehama:4736
PathTrim /x1
ConnectTimeoutSecs 30
</VirtualHost>
<VirtualHost goldengate.domain2.com>
WeblogicCluster green1:4736,green2:4736,green3:4736
PathTrim /y1
ConnectTimeoutSecs 20
</VirtualHost>
```

単一 IP アドレスによる名前ベースの仮想ホスティングのコンフィグレーション例

```
<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
         <Location / >
                SetHandler weblogic-handler
               WebLogicCluster 162.99.55.71:7001,162.99.55.72:7001
               Idempotent ON
                Debug ON
                DebugConfigInfo ON
        </Location>
</VirtualHost>
```

```
<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
       <Location / >
                SetHandler weblogic-handler
                WebLogicHost russell
                WebLogicPort 7001
                Debug ON
                DebugConfigInfo ON
        </Location>
</VirtualHost>
```

Apache プラグインでの SSL の使用

セキュア ソケット レイヤ (SSL) プロトコルを使用すると、Apache HTTP Server プラグインと WebLogic Server の間で接続を保護できます。SSL プロトコルは、Apache HTTP Server プラグインと WebLogic Server の間でやり取りされるデータに機密性と整合性をもたらします。

Apache HTTP Server プラグインは、(通常はブラウザによって) HTTP リクエストで指定される転送プロトコル (`http` または `https`) では、SSL プロトコルを使用して Apache HTTP Server プラグインと WebLogic Server の間の接続が保護されるかどうかを判断しません。

HTTP クライアントと Apache HTTP サーバ間では相互 SSL を使用できますが、Apache HTTP サーバと WebLogic Server 間では一方向の SSL が使用されます。

Apache と HTTP クライアント間の相互 SSL の実装

1. クライアント証明書を要求するよう Apache HTTP サーバをコンフィグレーションします。証明書は、次のいずれかのリクエスト属性として格納されます。
   - `javax.net.ssl.peer_certificates`
     `weblogic.security.X509Certificate` 証明書を返します。
   - `java.security.cert.X509Certificate`
     `java.security.cert.X509` 証明書を返します。

2. 次のようにリクエスト属性を読み出して、証明書にアクセスします。

   ```
   request.getAttribute("javax.net.ssl.peer_certificates");
   ```

3. WebLogic Server で、`weblogic.security.acl.certAuthenticator.authenticate()` メソッドを使用してユーザを認証します。

Apache HTTP Server プラグインと WebLogic Server 間の SSL のコンフィグレーション

Apache HTTP Server プラグインと WebLogic Server の間で SSL プロトコルを使用するには、次の手順を行います。

1. SSL 向けに WebLogic Server をコンフィグレーションします。詳細については、「SSL プロトコルのコンフィグレーション」を参照してください。
2. WebLogic Server の SSL リスン ポートをコンフィグレーションします。詳細については、「SSL プロトコルのコンフィグレーション」を参照してください。
3. Apache サーバで、`httpd.conf` ファイルの `WebLogicPort` パラメータを、手順 2. でコンフィグレーションしたリスン ポートに設定します。
4. Apache サーバで、`httpd.conf` ファイルの `SecureProxy` パラメータを `ON` に設定します。
5. SSL 接続の情報を定義する追加パラメータを `httpd.conf` ファイルで設定します。パラメータの詳細なリストについては、5-15 ページの「Web サーバ プラグインの SSL パラメータ」を参照してください。

WL-Proxy-Client-Cert ヘッダの信頼の指定

プラグインは `WL-Proxy-Client-Cert` ヘッダでユーザの ID 証明書をエンコードし、そのヘッダを WebLogic Server インスタンスに渡すことができます (「別の Web サーバへのリクエストのプロキシ」を参照)。WebLogic Server インスタンスは、それがセキュアなソース (プラグイン) から来ているものと信頼し、ヘッダの証明書情報を使用してユーザを認証します。以前の WebLogic Server のデ

フォルト動作では、WL-Proxy-Client-Cert ヘッダを常に信頼していました。WebLogic Server 6.1 SP2 からは、WL-Proxy-Client-Cert ヘッダの信頼を明示的に定義する必要があります。新しいパラメータ clientCertProxy を使用すると、証明書ヘッダを信頼するかどうか指定することができます。セキュリティの強化には、WebLogic Server へのすべての接続を制限する接続フィルタを使用します。そうすることで、プラグインが動作しているマシンからの接続のみを WebLogic Server で受け容れることができます。

clientCertProxy パラメータは、HTTPClusterServlet および Web アプリケーションに追加されています。

HTTPClusterServlet では、このパラメータを次のように web.xml ファイルに追加します。

```
<context-param>
        <param-name>clientCertProxy</param-name>
        <param-value>true</param-value>
</context-param>
```

Web アプリケーションでは、このパラメータを次のように web.xml ファイルに追加します。

```
ServletRequestImpl context-param
<context-param>
        <param-name>weblogic.httpd.clientCertProxy</param-name>
        <param-value>true</param-value>
</context-param>
```

このパラメータは、次のようにクラスタで使用することもできます。

```
<Cluster ClusterAddress="127.0.0.1" Name="MyCluster"
        ClientCertProxyHeader="true"/>
```

SSL-Apache コンフィグレーションに関する問題

SSL を使用するように Apache プラグインをコンフィグレーションする際には、以下の確認済みの問題に注意してください。

- PathTrim パラメータは、<Location> タグの内部でコンフィグレーションする必要があります。

次のコンフィグレーションは正しくありません。

```
<Location /weblogic>
                SetHandler weblogic-handler
</Location>

<IfModule mod_weblogic.c>
 WebLogicHost localhost
                WebLogicPort 7001
  PathTrim /weblogic
</IfModule>
```

次のコンフィグレーションは適切です。

```
<Location /weblogic>
                SetHandler weblogic-handler
  PathTrim /weblogic
</Location>
```

- Sunfreeware.com のコンパイル済みの OpenSSL を使用する場合、プラグインが WebLogic Server のバックエンド インスタンスに接続しようとすると、フェイルオーバが適切に動作しないことがあります。そのような障害が発生した場合は、以下のコンフィグレーション設定を使用して、OpenSSL、`modssl` および Apache を再ビルドしてください。
  - OpenSSL をビルドする場合

    ```
    ./Configure solaris-sparcv8-gcc -fexceptions

    --prefix=/home/egross/solaris/ssl shared

    make

    make install
    ```

  - modssl および Apache をビルドする場合

    ```
    cd ..

    cd mod_ssl-2.8.12-1.3.27

    export LD_LIBRARY_PATH=/home/egross/solaris/ssl/lib

     ./configure "--with-apache=../apache_1.3.27"

    "--with-ssl=/home/egross/solaris/ssl"
    "--prefix=/usr/local/apache_so"

    "--enable-rule=SHARED_CORE" "--enable-shared=ssl"
    "--enable-module=so" "$@"

    cd ../apache_1.3.27

    make
    ```

```
make install
```

- Include ディレクティブは Apache SSL では機能しません。パラメータはすべて httpd.conf ファイルで直接コンフィグレーションする必要があります。SSL を使用する際には次のコンフィグレーションは使用しないでください。

```
<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  Include weblogic.conf
</IfModule>
```

  - WebLogic Server Apache プラグインの現在の実装では、ApacheSSL で複数の証明書ファイルを使用することはできません。

接続エラーとクラスタのフェイルオーバ

WebLogic Server に接続するときに、Apache HTTP Server プラグインは複数のコンフィグレーション パラメータを使用して WebLogic Server ホストへの接続の待ち時間と、接続確立後の応答の待ち時間を判断します。接続できないか、応答がない場合、このプラグインはクラスタ内の別の WebLogic Server インスタンスに接続してリクエストを送信しようとします。接続が失敗するか、クラスタ内のどの WebLogic Server からも応答がない場合は、エラー メッセージが送信されます。

2-24 ページの「接続のフェイルオーバ」図 2-1 は、プラグインがどのようにフェイルオーバを処理するのかを示しています。

接続失敗の考えられる原因

接続要求に WebLogic Server ホストが応答できない場合は、ホスト マシンの問題やネットワークの問題など、サーバに障害があることが考えられます。

すべての WebLogic Server インスタンスが応答できない場合は、WebLogic Server が動作していないことや、サーバのハング、データベースの問題など、アプリケーションに障害があることが考えられます。

クラスタ化されていない単一 WebLogic Server でのフェイルオーバ

WebLogic Server のインスタンスが 1 つだけ動作している場合、プラグインは `WebLogicHost` パラメータで定義されているサーバにのみ接続しようとします。その試行が失敗すると、`HTTP 503` エラー メッセージが返されます。プラグインは、`ConnectTimeoutSecs` を超えるまでその同じ WebLogic Server インスタンスへの接続を試み続けます。

動的サーバ リスト

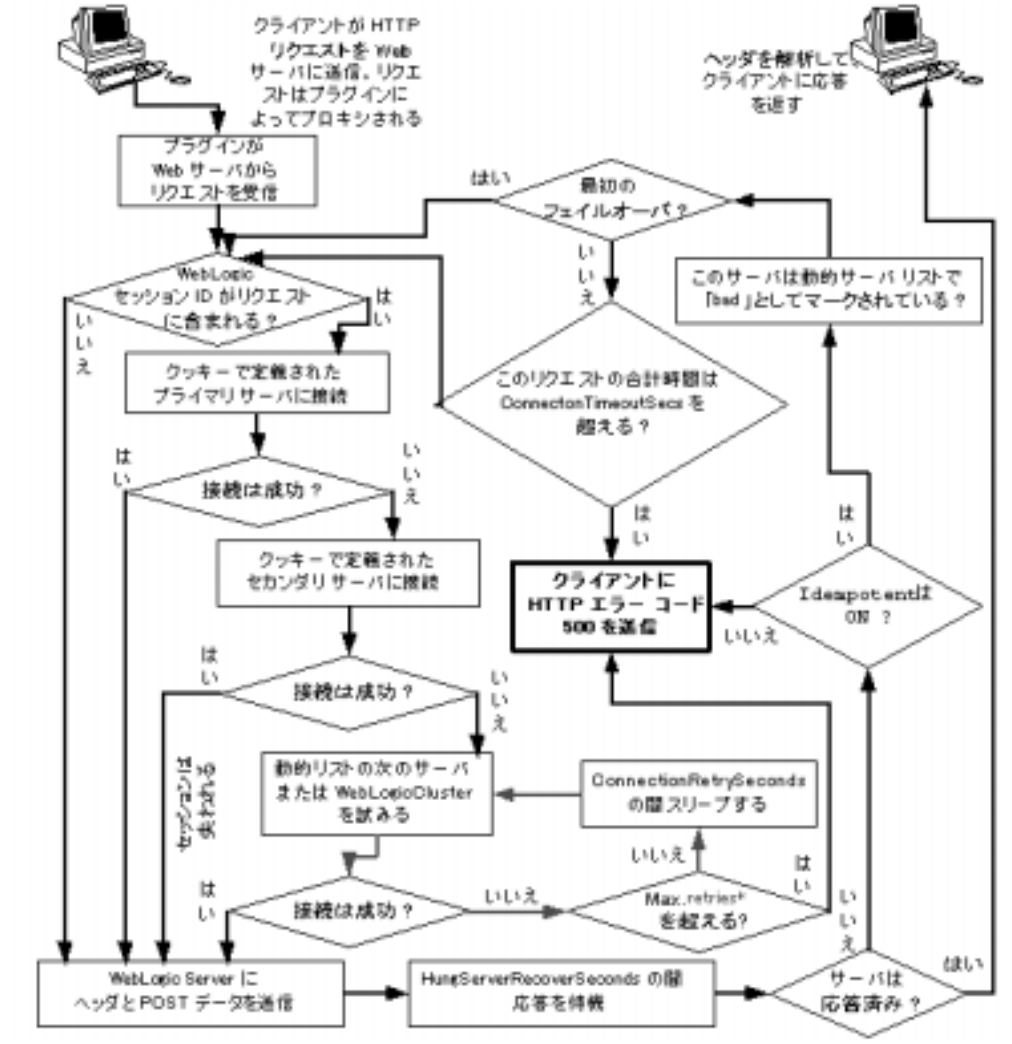
`WebLogicCluster` パラメータで WebLogic Server のリストを指定すると、プラグインではクラスタ メンバー間でのロード バランシングの起点としてそのリストが使用されます。最初のリクエストがそれらのサーバの 1 つに転送された後に、クラスタ内のサーバの更新されたリストを格納する動的サーバ リストが返されます。更新されたリストはクラスタ内の新しいサーバを追加し、すでにクラスタから外れているか、リクエストに応答できなかったサーバを削除します。このリストは、クラスタで変更が行われたときに HTTP 応答によって自動的に更新されます。

フェイルオーバ、クッキー、および HTTP セッション

リクエストがクッキー、POST データ、または URL エンコーディングを通じてセッション情報を格納している場合、そのセッション ID にはセッションが最初に確立された特定のサーバ (プライマリ サーバ) への参照と元のセッションがレプリケートされる追加サーバ (セカンダリ サーバ) への参照が含まれています。クッキーが含まれているリクエストは、プライマリ サーバに接続しようとします。その試行が失敗すると、リクエストはセカンダリ サーバに転送されます。プライマリ サーバとセカンダリ サーバが両方とも失敗すると、セッションが失われて、プラグインは動的クラスタ リストの別のサーバにあらためて接続しようとします。詳細については、2-24 ページの「接続のフェイルオーバ」図 2-1 を参照してください。

注意： POST データが 64K を超える場合、プラグインは、セッション ID を取得するための POST データの解析を行いません。したがって、セッション ID を POST データに格納した場合、プラグインはリクエストを正しいプライマリまたはセカンダリ サーバにルーティングできないので、セッション データが失われる可能性があります。

図 2-1 接続のフェイルオーバ



* 赤いループで許可される再試行の限度は、次の式で計算されます。
`ConnectTimeoutSecs ÷ ConnectRetrySecs`