



BEA WebLogic Server™

WebLogic Server 口 ギング サービスの使 い方

著作権

Copyright © 2002, BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社（以下、「BEA」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii)、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA、Jolt、Tuxedo、および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder、BEA Campaign Manager for WebLogic、BEA eLink、BEA Manager、BEA WebLogic Commerce Server、BEA WebLogic Enterprise、BEA WebLogic Enterprise Platform、BEA WebLogic Express、BEA WebLogic Integration、BEA WebLogic Personalization Server、BEA WebLogic Platform、BEA WebLogic Portal、BEA WebLogic Server、BEA WebLogic Workshop および How Business Becomes E-Business は、BEA Systems, Inc の商標です。

その他の商標はすべて、関係各社がその権利を有します。

WebLogic Server ログイン サービスの使い方

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	2002年6月28日	BEA WebLogic Server バージョン 7.0

目次

このマニュアルの内容

対象読者.....	v
e-docs Web サイト.....	v
このマニュアルの印刷方法.....	vi
関連情報.....	vi
サポート情報.....	vi
表記規則.....	vii

1. WebLogic ロギング サービスの概要

2. WebLogic Server ログへのメッセージの書き込み

I18N メッセージ カタログ フレームワーク 使い方: 主要な手順.....	2-1
手順 1: メッセージ カタログの作成.....	2-2
手順 2: メッセージ カタログのコンパイル.....	2-4
手順 3: コンパイルされたメッセージ カタログからのメッセージの使用 2-5	
NonCatalogLogger API の使用.....	2-6
GenericServlet の使用.....	2-11
リモート アプリケーションからのメッセージの書き込み.....	2-11
リモート JVM からファイルへのメッセージの書き込み.....	2-12
デバッグ メッセージの書き込み.....	2-12

3. WebLogic Server ログの表示

4. WebLogic Server ログからのメッセージのリسن

手順 1: 通知リスナの作成.....	4-2
WebLogic Server JVM 内で実行されるアプリケーション用の通知リスナ の作成.....	4-3
リモート アプリケーション用の通知リスナの作成.....	4-5
手順 2: 通知リスナの登録.....	4-7
addNotificationListener API の使用.....	4-8
通知リスナの登録例.....	4-9

手順 3 : 通知フィルタの作成と登録.....	4-12
フィルタの作成と登録.....	4-12
WebLogicLogNotification オブジェクト	4-13
通知フィルタの例	4-15

このマニュアルの内容

このマニュアルでは、アプリケーションが、BEA WebLogic Server™ のログファイルにメッセージを書き込んだり、WebLogic Server がブロードキャストしたログメッセージをリスンしたりする方法について説明します。また、WebLogic Server Administration Console を使用してログメッセージを表示する方法についても概説します。

マニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic Server ログへのメッセージの書き込み」
- 第 2 章「WebLogic Server ログの表示」
- 第 3 章「WebLogic Server ログからのメッセージのリスン」

対象読者

このマニュアルは、WebLogic Server 上で動作する Web アプリケーションまたはその他の Java 2 Platform, Enterprise Edition (J2EE) コンポーネントを構築するアプリケーション開発者を対象としています。このマニュアルは、Web テクノロジ、オブジェクト指向プログラミング手法、および Java プログラミング言語に読者が精通していることを前提として書かれています。

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。

関連情報

BEA の Web サイトでは、WebLogic Server の全マニュアルを提供しています。特に、『管理者ガイド』の「ログ メッセージを使用した WebLogic Server の管理」では、WebLogic Server で生成されたログ ファイルをコンフィグレーションする方法について説明しています。また、『インターナショナルライゼーション ガイド』では、アプリケーションで使用できるメッセージ カタログの設定方法について説明しています。

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
太字	用語集で定義されている用語を示す。
[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。

表記法	適用
等幅テキスト	<p>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。</p> <p>例：</p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
太字の等幅テキスト	<p>コード内の重要な箇所を示す。</p> <p>例：</p> <pre>void commit ()</pre>
斜体の等幅テキスト	<p>コード内の変数を示す。</p> <p>例：</p> <pre>String <i>expr</i></pre>
すべて大文字のテキスト	<p>デバイス名、環境変数、および論理演算子を示す。</p> <p>例：</p> <pre>LPT1 SIGNON OR</pre>
{ }	<p>構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。</p>
[]	<p>構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。</p> <p>例：</p> <pre>buildobjclient [-v] [-o name] [-f file-list]...[-l file-list]...</pre>

表記法	適用
	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	<p>コマンドラインで以下のいずれかを示す。</p> <ul style="list-style-type: none"> ■ 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。 <p>実際には、この省略符号は入力しない。</p> <p>例：</p> <pre>buildobjclient [-v] [-o name] [-f file-list]...[-l file-list]...</pre>
.	<p>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。</p> <p>実際には、この省略符号は入力しない。</p>



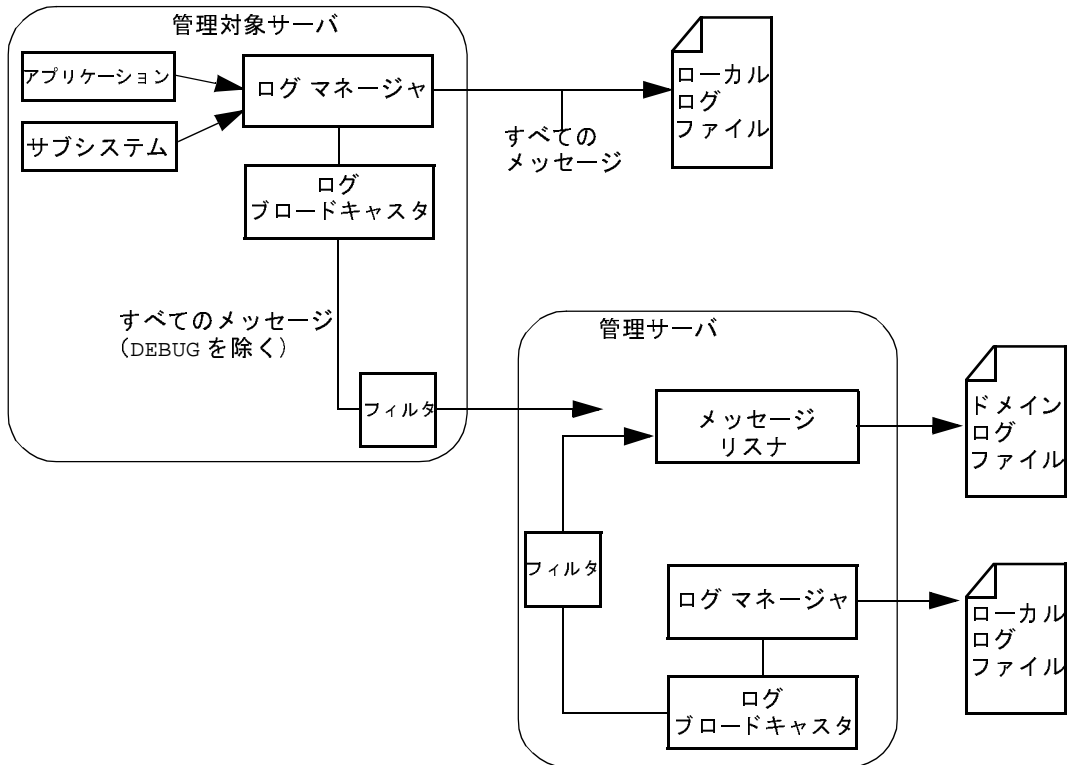
1 WebLogic ロギング サービスの概要

WebLogic Server には、ログ メッセージを記録、参照、およびリスンするための機能が用意されています。WebLogic Server サブシステムはこれらのサービスを利用して、新しいアプリケーションのデプロイメントや 1 つまたは複数のサブシステムのエラーなどのイベントに関する情報を提供します。また、アプリケーションはこれらのサービスを利用してそのステータスを送信し、特定のイベントに応答することができます。たとえば、WebLogic ロギング サービスを利用すれば、どのユーザが特定のアプリケーション コンポーネントを呼び出すかを記録したり、エラー状態をレポートしたり、アプリケーションをプロダクション環境にリリースする前にデバッグしたりできます。また、アプリケーションが特定のサブシステムからのログ メッセージをリスンして適切に応答するようコンフィグレーションすることもできます。

各 WebLogic Server 管理ドメインは WebLogic Server の複数のインスタンスを同時に実行できるので、ロギング サービスは複数のサーバ インスタンスで生成されるメッセージを収集して、単一の、ドメイン全体のメッセージ ログにまとめます。このドメイン全体のメッセージ ログを使用すると、ドメイン全体のステータスを確認できます。

ドメインのステータスを把握できるようにするために、各 WebLogic Server インスタンスは組み込み Java Management Extensions (JMX) 機能を使用してメッセージをブロードキャストします。これらのメッセージは、ログ ファイルに格納されます。ブロードキャストには、サブシステムとアプリケーションが生成するすべてのメッセージが含まれます (アプリケーションによって生成される特別なデバッグ メッセージを除く)。管理サーバは、これらの通知をリスンし、それらのサブセットをドメイン全体のログ ファイルに書き込みます (図 1-1 を参照)。

図 1-1 WebLogic Server ログイン サービス



以降の章では、アプリケーションがメッセージを記録およびリスンする仕組みと、それらのメッセージを **WebLogic Server Administration Console** で参照する方法について説明します。

2 WebLogic Server ログへのメッセージの書き込み

以下の節では、ログメッセージを WebLogic Server ログ ファイルに書き込むことによってアプリケーションの管理を効率化する方法について説明します。

- 2-1 ページの「I18N メッセージ カタログ フレームワーク 使い方: 主要な手順」
- 2-6 ページの「NonCatalogLogger API の使用」
- 2-11 ページの「GenericServlet の使用」

また、この章には以下の節も含まれています。

- 2-11 ページの「リモートアプリケーションからのメッセージの書き込み」
- 2-12 ページの「デバッグ メッセージの書き込み」

I18N メッセージ カタログ フレームワーク 使い方: 主要な手順

インターナショナルライゼーション (I18N) メッセージ カタログ フレームワークは、アプリケーションが WebLogic Server ログに独自のメッセージセットを送るために使用する一連のユーティリティと API を提供します。このフレームワークは、ログメッセージをローカライズする必要のあるアプリケーションにとって理想的ですが、ローカライズの不要なアプリケーションに対しても、状態の通信や出力のためのフレキシブルで優れた一連のツールを提供します。

I18N メッセージ カタログ フレームワークを使用してログメッセージを書き込むには、以下のタスクを行います。

- 手順 1: メッセージ カタログの作成

- 手順 2: メッセージカタログのコンパイル
- 手順 3: コンパイルされたメッセージカタログからのメッセージの使用

手順 1: メッセージカタログの作成

メッセージカタログは、一連のテキストメッセージを含んだ XML ファイルです。通常、アプリケーションはデフォルトのメッセージセットを含むメッセージカタログ 1 つを使用し、ローカライズされたメッセージを含む追加カタログを任意で使用します。

適切にフォーマットされたメッセージカタログを作成および編集するには、**WebLogic Server** と一緒にインストールされるグラフィカルユーザインタフェース、**WebLogic** メッセージエディタユーティリティを使用します。

メッセージエディタにアクセスするには、**WebLogic Server** ホストから次の手順を実行します。

1. `WL_HOME\server\bin\setWLSEnv.cmd` (UNIX では `setWLSEnv.sh`) を入力してクラスパスを設定します。`WL_HOME` は **WebLogic Server** をインストールしたディレクトリです。
2. 次のコマンドを入力します。`java weblogic.MsgEditor`
3. 新しいカタログを作成するには、[ファイル | New Catalog] を選択します。

メッセージエディタの使い方については、以下を参照してください。

- 『インターナショナルライゼーションガイド』の「BEA WebLogic Server メッセージエディタの使い方」
- 『インターナショナルライゼーションガイド』の「BEA WebLogic Server でのメッセージカタログの使い方」

メッセージエディタでの作業を保存すると、**WebLogic Server** は、メッセージカタログを構成する XML ファイルを作成します。コードリスト 2-1 に、3 つのメッセージを定義するメッセージカタログサンプルを示します。このサンプルは、**WebLogic Server** サンプルと一緒に次の場所にインストールされています。`WL_HOME\samples\server\src\examples\i18n\msgcat\UserServerSCEexample.xml`.

コード リスト 2-1 メッセージ カタログの例

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls700/msgcat.dtd">
<message_catalog
  i18n_package="examples.i18n.logging.startup"
  l10n_package="examples.i18n.logging.startup"
  subsystem="UserStartupClass"
  version="1.0"
  baseid="909050"
  endid="909059"
>
  <logmessage
    messageid="909050"
    datelastchanged="973906351125"
    datehash="-854388901"
    severity="info"
    method="logInitialMessage(int money 0)"
    stacktrace="false"
  >
    <messagebody>
      This message displays a number as currency:
      {0,number,currency}.
    </messagebody>
    <messagedetail>
      Just an example.
    </messagedetail>
    <cause>
    </cause>
    <action>
    </action>
  </logmessage>
  <logmessage
    messageid="909051"
    datelastchanged="973906462765"
    datehash="-1800319350"
    severity="warning"
    method="logStringAndPrevCallCount(String str0, int num1)"
    stacktrace="false"
  >
    <messagebody>
      This message displays the string "{0}". There
      {1,choice,0#were no previous calls|1#was one previous

```

```
call|2#were {1,number} previous calls} to the logger method
for this message.
</messagebody>

    <messagedetail>
    </messagedetail>

    <cause>
    </cause>

    <action>
    </action>

</logmessage>

<logmessage
  messageid="909052"
  datelastchanged="973906532006"
  datehash="-160371672"
  severity="error"
  method="logFinalMessage()"
  stacktrace="false"
>

  <messagebody>
  This is not really an error, the example has finished
  normally.
  </messagebody>

  <messagedetail>
  </messagedetail>

  <cause>
  </cause>

  <action>
  </action>

</logmessage>
</message_catalog>
```

手順 2 : メッセージ カタログのコンパイル

メッセージカタログを作成したら、次のユーティリティを使用して XML ファイルから Java クラス ファイルを生成します。

- `i18ngen` ユーティリティは、メッセージカタログを検証し、クラスパスにコンパイルして配置される **Java** ファイルを生成します。各 **Java** クラスには、**XML** ファイルのメッセージに対応するメソッドが含まれます
- `l10ngen` ユーティリティは、ロケール固有のカタログを検証し、カタログで定義されたさまざまな異なるロケールで使用される追加のプロパティ ファイルを作成します。

Java クラス ファイルを生成およびコンパイルするには、次の手順に従います。

1. `WL_HOME\server\bin\setWLSEnv.cmd` (UNIX では `setWLSEnv.sh`) を使用してクラスパスを設定します。`WL_HOME` は **WebLogic Server** をインストールしたディレクトリです。
2. 以下のいずれかのコマンドを入力します。
 - `java weblogic.i18ngen [options] files`
 - `java weblogic.i18ngen.tools.l10ngen [options] filelist`

これらのコマンドにより、**Java** ソース ファイルが生成されます。パッケージ名は、**XML** 入力ファイルの `i18n_package` および `l10n_package` 属性によって指定されます。

3. ソース ファイルをコンパイルしてクラスパスに追加します。

`i18ngen` コマンドの詳細については、『**インターナショナルライゼーション ガイド**』の「**BEA WebLogic Server のインターナショナルライゼーション ユーティリティの使い方**」を参照してください。

手順 3：コンパイルされたメッセージ カタログからのメッセージの使用

`i18ngen` によって生成されたクラスは、**WebLogic Server** ログにメッセージを送信するためのインタフェースを提供します。これらのクラスの中では、各ログメッセージはアプリケーションが呼び出す 1 つのメソッドによって表されます。

たとえば、`UserServerSCEExample.xml` という名のメッセージカタログ (コードリスト 2-1 を参照) の場合、`i18ngen` ユーティリティは `examples.i18n.logging.startup.UserServerSCEExampleLogger` という名前

のクラスを生成します。アプリケーションで `logFinalMessage` メッセージを書き込む場合、`UserServerSCExampleLogger.logFinalMessage()` メソッドを呼び出します。コードリスト 2-2 に、このメソッドを呼び出す JSP を示します。

コードリスト 2-2 メッセージ カタログを使用する JSP の例

```
<html>
Order complete. Thanks for your order!

<%@ page
import="examples.il8n.logging.message.UserServerSVExampleLogger"
%>

<%
    UserServerSVExampleLogger.logFinalMessage();
%>

</body>
</html>
```

NonCatalogLogger API の使用

I18N メッセージ カタログ フレームワークの使用に加え、アプリケーションでは `weblogic.logging.NonCatalogLogger` API を使用して **WebLogic Server** ログにメッセージを送信できます。カタログからメッセージを呼び出す代わりに `NonCatalogLogger` を使用することで、メッセージテキストをアプリケーション コードに直接配置します。アプリケーションをインターナショナル化する必要がある場合は、メッセージ ロギングの唯一の手段としてこの機能を使用することはお勧めしません。

また、`NonCatalogLogger` は (**WebLogic Server JVM** 内で実行されているのではなく) 独自の **JVM** で実行されているクライアント コードによっても使用されません。詳細については、2-11 ページの「リモート アプリケーションからのメッセージの書き込み」を参照してください。

WebLogic Server JVM の内部で実行されるアプリケーションで `NonCatalogLogger` を使用するには、以下を行うためのコードをアプリケーションに追加します。

1. `weblogic.logging.NonCatalogLogger` インタフェースをインポートします。
2. 以下のコンストラクタを使用して `NonCatalogLogger` オブジェクトをインスタンス化します。

```
NonCatalogLogger(java.lang.String myApplication)
```

`myApplication` は、アプリケーションから **WebLogic Server** ログに送信されるメッセージを識別するために、ユーザが指定する名前です。

3. いずれかの `NonCatalogLogger` メソッドを呼び出します。

正常な処理を報告するには、以下のメソッドを使用します。

- `info(java.lang.String msg)`
- `info(java.lang.String msg, java.lang.Throwable t)`

サーバ/アプリケーションの正常な処理に影響しない要注意の処理、イベント、またはコンフィグレーションを報告するには、以下のメソッドを使用します。

- `warning(java.lang.String msg)`
- `warning(java.lang.String msg, java.lang.Throwable t)`

システム/アプリケーションが割り込みやサービスの停止をせずに対処できるエラーを報告するには、以下のメソッドを使用します。

- `error(java.lang.String msg)`
- `error(java.lang.String msg, java.lang.Throwable t)`

処理の詳細またはアプリケーションの状態を示すには、以下のメソッドを使用します。これらのデバッグメッセージは、ドメイン ログには転送されません。この重大度を使用する場合、アプリケーション用の「デバッグモード」を作成することをお勧めします。次に、デバッグモードで実行されるようコンフィグレーションされている場合にのみデバッグメッセージを出力するようアプリケーションをコンフィグレーションします。デバッグメッセージの使い方については、2-12 ページの「デバッグメッセージの書き込み」を参照してください。

- `debug(java.lang.String msg)`
- `debug(java.lang.String msg, java.lang.Throwable t)`

`Throwable` 引数を取るすべてのメソッドは、エラー ログにスタック トレースを出力する可能性があります。`NonCatalogLogger API` の詳細については、`weblogic.logging.NonCatalogLogger Javadoc` を参照してください。

2 WebLogic Server ログへのメッセージの書き込み

コードリスト 2-3 に、NonCatalogLogger API を使用してさまざまな重大度のメッセージを WebLogic Server ログに書き込むサーブレットを示します。

コード リスト 2-3 NonCatalogLogger メッセージの例

```
import java.io.PrintWriter;
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.logging.NonCatalogLogger;

public class MyServlet extends HttpServlet {

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

        PrintWriter out = response.getWriter();
        NonCatalogLogger myLogger = null;

        try {

            out.println("Testing NonCatalogLogger. See WLS Server log for output
                message.");

            // NonCatalogLogger インスタンスを作成する。このインスタンスからの
            // すべてのメッセージには <MyApplication> 文字列が含まれる
            myLogger = new NonCatalogLogger("MyApplication");

            // アプリケーションが起動したことを示す INFO メッセージを出力する
            myLogger.info("Application started.");

            // 例外メッセージの例を示すために、コードの次の行では
            // 意図的に初期コンテキストを設定している。デフォルトの
            // ポート番号 (7001) を使用するサーバでこのサーブレットを実行すると、
            // サーブレットは例外を送出する
            Environment env = new Environment();
            env.setProviderUrl("t3://localhost:8000");

            Context ctx = env.getInitialContext();

        }
    }
}
```

```
catch (Exception e){
    out.println("Can't set initial context: " + e.getMessage());
}
// スタック トレースを含む WARNING メッセージを出力する
mylogger.warning("Can't establish connections. ", e);
}
}
}
```

上記の例で示したサーブレットを 8000 以外のリスン ポートを指定するサーバ上で実行すると、以下のメッセージが **WebLogic Server** ログ ファイルに出力されます。メッセージは山括弧 (< >) で囲まれた一連の文字列またはフィールドで構成されています。

コード リスト 2-4 NonCatalogLogger Output

```
####<Jun 26, 2002 12:04:21 PM EDT> <Info> <MyApplication> <peach> <examplesServer>
<ExecuteThread: '10' for queue: 'default'> <kernel identity> <> <000000>
<Application started.>

####<Jun 26, 2002 12:04:23 PM EDT> <Warning> <MyApplication> <peach>
<examplesServer> <ExecuteThread: '10' for queue: 'default'> <kernel identity> <>
<000000> <Can't establish connections. >

javax.naming.CommunicationException. Root exception is
java.net.ConnectException: t3://localhost:8000: Destination unreachable; nested
exception is:

...

```

2 WebLogic Server ログへのメッセージの書き込み

表 2-1 に、NonCatalogLogger ログ メッセージに含まれるすべてのフィールドを示します。

表 2-1 NonCatalogLogger ログ メッセージのフォーマット

フィールド	説明
Localized Timestamp	メッセージが生成された日付と時刻。年、月、日、時、分、および秒が記載される。例: <Jun 26, 2002 12:04:21 PM EDT>
Severity	次の重大度のいずれか。メッセージの生成に使用したメソッドのタイプに対応する。 Info、Warning、Error、Debug
Subsystem	メッセージのソースを示す。NonCatalogLogger コンストラクタで指定する文字列。
MachineName	アプリケーションが実行される JVM をホストするコンピュータの名前。
ServerName	アプリケーションが実行される WebLogic Server インスタンスの名前。
ThreadId	現在のプロセスが使用している実行スレッドを示す。WebLogic Server で複数の実行キューを使用すると、アプリケーションから実行スレッドへのアクセスを微調整できる（従って、パフォーマンスを最適化できる）。詳細については、『パフォーマンスチューニングガイド』の「実行キューによるスレッド使用の制御」を参照。
User Id	エラーが報告されたときに実行されていたシステムのユーザ。
TransactionId	トランザクションのコンテキストでロギングされたメッセージにのみ示される。
Message Id	メッセージの 6 桁の識別子。NonCatalogLogger メッセージの場合、メッセージ ID は常に 000000。
Message Text	NonCatalogLogger メソッドで指定するテキスト。
ExceptionName	メッセージに例外がロギングされた場合、このフィールドには例外の名前が示される。

GenericServlet の使用

`javax.servlet.GenericServlet` サープレット仕様は、サープレットが `WebLogic Server` ログにシンプルなメッセージを書き込むために使用できる以下の API を提供します。

- `log(java.lang.String msg)`
- `log(java.lang.String msg, java.lang.Throwable t)`

これらの API の使い方については、`javax.servlet.GenericServlet` の J2EE Javadoc

(<http://java.sun.com/products/servlet/2.3/javadoc/javax/servlet/GenericServlet.html>) を参照してください。

JSP は `GenericServlet` から拡張されることはなく、これらの API を使用できません。JSP でログ ファイルにメッセージを送信する場合は、I18N メッセージカタログ サービスまたは `NonCatalogLogger` API を使用してください。

リモート アプリケーションからのメッセージの書き込み

アプリケーションが `WebLogic Server` から離れた JVM で実行される場合、メッセージ カタログおよび `NonCatalogLogger` は使用できませんが、メッセージは `WebLogic Server` ログには書き込まれません。代わりに、アプリケーションのメッセージはリモート JVM の標準出力に書き込まれます。

`WebLogic` ロギング サービスで、リモート JVM マシンが保持するログ ファイルにメッセージを送る場合は、リモート JVM を起動するコマンドで次の引数を指定します。

```
-Dweblogic.log.FileName=logfilename
```

`logfilename` は、リモート ログ ファイルの名前です。

メッセージカタログのサブセットと NonCatalogLogger メッセージをリモート JVM ログ ファイルと標準出力に送る場合は、さらに次の起動引数を指定します。

```
-Dweblogic.StdoutEnabled=true
```

```
-Dweblogic.StdoutDebugEnabled=boolean
```

```
-Dweblogic.StdoutSeverityLevel = [ 64 | 32 | 16 | 8 | 4 | 2 | 1 ]
```

boolean は true か false のいずれかで、StdoutSeverityLevel の数値は以下の重大度に対応します。

INFO(64)、WARNING(32)、ERROR(16)、NOTICE(8)、CRITICAL(4)、
ALERT(2)、および EMERGENCY(1)

リモート JVM からファイルへのメッセージの書き込み

リモート JVM は、自身のステート情報を送信するための独自のメッセージセットを生成できます。たとえば、ガベージコレクションに関するメッセージを生成するよう JVM をコンフィグレーションできます。デフォルトでは、JVM はこれらのメッセージを標準出力に送ります。これらのメッセージは JVM のログ ファイルにリダイレクトできませんが、別のファイルに保存できます。詳細については、『管理者ガイド』の「System.out および System.err のファイルへのリダイレクト」を参照してください。

デバッグ メッセージの書き込み

アプリケーションの開発中、そのアプリケーションの低レベル アクティビティを示すメッセージを作成および使用できると便利です。DEBUG 重大度を使用すると、これらの低レベル メッセージを分類できます。アプリケーションが生成するすべての DEBUG メッセージは WebLogic Server ログ ファイルに送られます。重大度に基づいてログ メッセージを動的に除外できるサードパーティ ログギング サービスの Log4j とは異なり、WebLogic Server ログにはアプリケーションが生成するすべてのレベルのメッセージが含まれます。

WebLogic Server が DEBUG メッセージを標準出力に送るようコンフィグレーションできません。詳細については、**Administration Console** オンラインヘルプの「一般的なログ ファイル設定の指定」を参照してください。

DEBUG 重大度を使用する場合、アプリケーション用の「デバッグ モード」を作成することをお勧めします。たとえば、アプリケーションがブール値を格納するオブジェクトを作成できるとします。デバッグ モードを有効または無効にするには、ブール値を切り替えます。次に、DEBUG メッセージごとに、アプリケーションのデバッグ モードが有効化されている場合にのみメッセージを出力するラッパーを作成します。

次に例を示します。

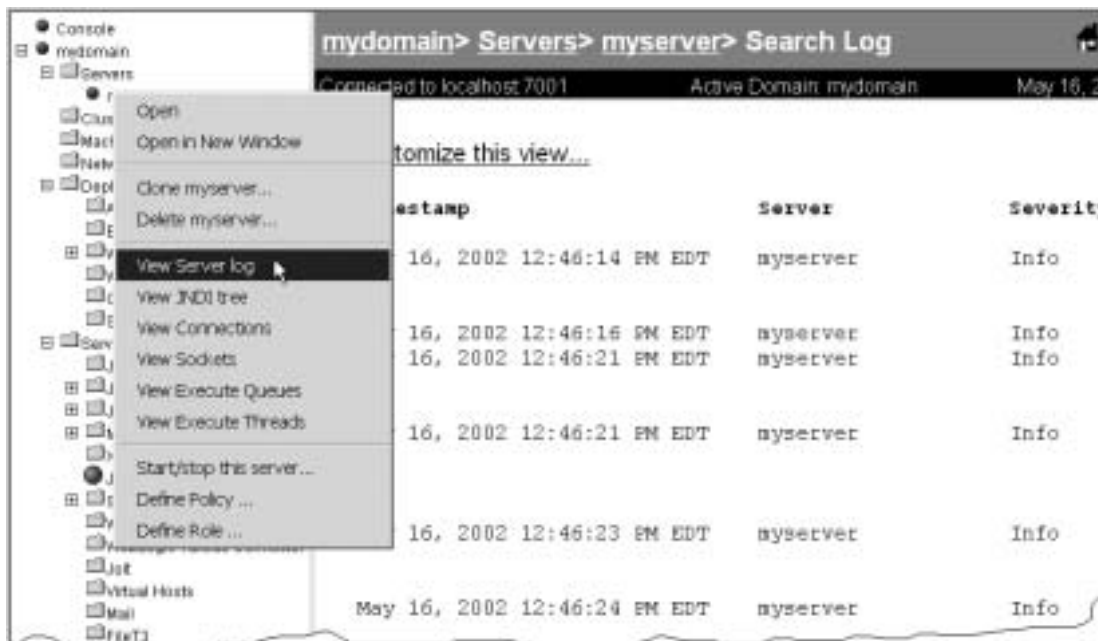
```
private static boolean debug = Boolean.getBoolean("my.debug.enabled");
if (debug) {
    mylogger.debug("Something debuggy happened");
}
```

この種のラッパーは、メッセージカタログ フレームワークを使用するメッセージと NonCatalogLogger API を使用するメッセージの両方に対して使用できます。

3 WebLogic Server ログの表示

Administration Console には、ローカルサーバログとドメイン全体のメッセージログ用のログビューアがそれぞれ用意されています（機能はほぼ同じです）。ログビューアは、メッセージ内のフィールドに基づいてメッセージを検索できます。たとえば、重大度、発生時間、ユーザ ID、サブシステム、あるいは短い説明に基づいてメッセージを表示できます。また、記録どおりにメッセージを表示したり、過去のログメッセージを検索することもできます。詳細については、図 3-1 を参照してください。

図 3-1 ログビューア



3 WebLogic Server ログの表示

Administration Console からのメッセージの表示に加え、どのメッセージを標準出力に送るかも指定できます。デフォルトでは、ERROR 以上のメッセージだけが標準出力に送られます。

メッセージ ログの表示、コンフィグレーション、および検索については、以下のトピックを参照してください。

- Administration Console オンライン ヘルプの「サーバのログの表示」
- Administration Console オンライン ヘルプの「一般的なログ ファイル設定の指定」
- Administration Console オンライン ヘルプの「サーバ ログ ファイルのデバッグ情報のコンフィグレーション」
- Administration Console オンライン ヘルプの「ドメイン ログの表示」
- 『管理者ガイド』の「ログ メッセージを使用した WebLogic Server の管理」

4 WebLogic Server ログからのメッセージのリスン

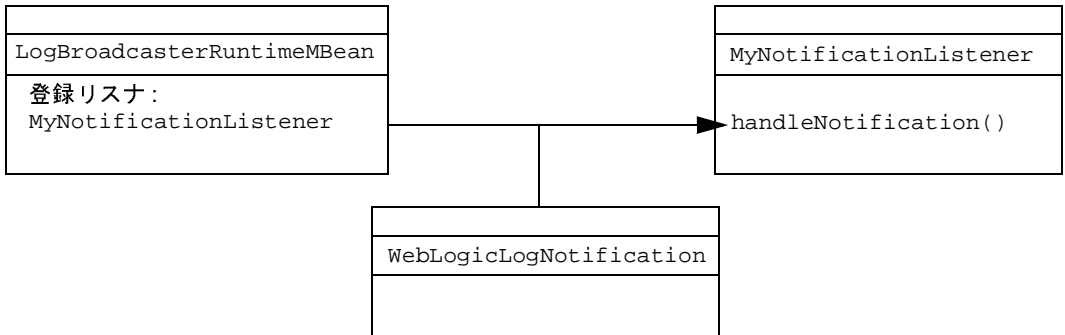
各 WebLogic Server は、そのログ メッセージを JMX 通知フォームでブロードキャストします。ブロードキャストには、WebLogic Server インスタンス、そのサブシステム、およびアプリケーションが WebLogic Server ログに書き込むすべてのメッセージ（DEBUG 重大度のメッセージを除く）が含まれます。管理サーバは、これらの通知をリスンし、それらのサブセットをドメイン全体のログ ファイルに書き込みます（1-6 ページの図 1-1 「WebLogic Server ロギング サービス」を参照）。

アプリケーションは、WebLogic Server インスタンスからブロードキャストされるログ メッセージをリスンできます。たとえば、特定のサブシステムのエラーを知らせるログ メッセージをリスンできます。この場合、アプリケーションは以下のようなアクションを実行できます。

- WebLogic 管理者にログ メッセージを電子メールで送信する
- 自身またはそのサブシステムを終了または再起動する

これらの通知をリスンするには、通知リスナを作成し、そのリスナを WebLogic Server ブロードキャスト MBean の LogBroadcasterRuntimeMBean に登録します。通知リスナは、JMX NotificationListener インタフェースの実装です。LogBroadcasterRuntimeMBean は、通知を発するとき登録済みリスナの handleNotification メソッドを使用して WebLogicLogNotification オブジェクトを受け渡します。詳細については、図 4-1 を参照してください。

図 4-1 WebLogic ブロードキャストとリスナ



WebLogicLogNotification オブジェクトの詳細については、4-13 ページの「WebLogicLogNotification オブジェクト」を参照してください。

アプリケーションが WebLogic Server ログからの通知をリسنできるようにするには、以下のタスクを実行します。

- 手順 1: 通知リスナの作成
- 手順 2: 通知リスナの登録
- 手順 3: 通知フィルタの作成と登録

注意: アプリケーションが WebLogic Server JVM の外部で実行される場合、そのアプリケーションは WebLogic Server ログ通知をリسنできませんが、WebLogic ロギング サービスを利用してメッセージをブロードキャストすることはできません。

手順 1 : 通知リスナの作成

通知リスナを作成するための手順は、アプリケーションが WebLogic Server JVM 内で実行されているかどうかによって異なります。

この節では、以下の項目について説明します。

- WebLogic Server JVM 内で実行されるアプリケーション用の通知リスナの作成
- リモート アプリケーション用の通知リスナの作成

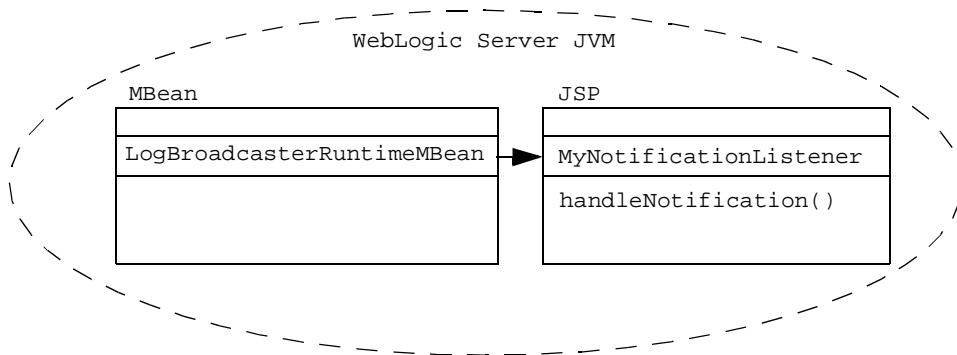
WebLogic Server JVM 内で実行されるアプリケーション用の通知リスナの作成

アプリケーションが WebLogic Server JVM 内で実行される場合、次の手順に従います。

1. `javax.management.Notification.*` インタフェースをインポートします。**WebLogic Server** はすでにこれらのインタフェースを持ち、クラスパス上にあることを要求しているため、クラスに入れる必要があるのはインポート文のみです。
2. `NotificationListener` を実装するクラスを作成します。実装には `NotificationListener.handleNotification()` メソッドを含める必要があります。
`NotificationListener` の詳細については、`javax.management.Notification` Javadoc (<http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>) を参照してください。

図 4-2 に、JSP が WebLogic Server JVM 内で実行されているシステムを示します。JSP は、`LogBroadcasterRuntimeMBean` からの通知をリスンします。

図 4-2 ローカル JSP 用のリスナ



コード リスト 4-1 に、ローカル クライアントの通知リスナの例を示します。このリスナは、WebLogicLogNotification ゲッター メソッドを使用して、受信したすべてのメッセージを出力します。詳細については、4-13 ページの「WebLogicLogNotification オブジェクト」を参照してください。

コード リスト 4-1 ローカル クライアント用の通知リスナの例

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;

...

public class MyNotificationListener implements
    NotificationListener {

    ...

    public void handleNotification(Notification notification, Object obj) {
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
        System.out.println("WebLogicLogNotification");
        System.out.println(" type = " + wln.getType());
        System.out.println(" message id = " + wln.getMessageId());
        System.out.println(" server name = " + wln.getServername());
        System.out.println(" timestamp = " + wln.getTimeStamp());
        System.out.println(" message = " + wln.getMessage() + "\n");
    }
}
```

リモート アプリケーション用の通知リスナの作成

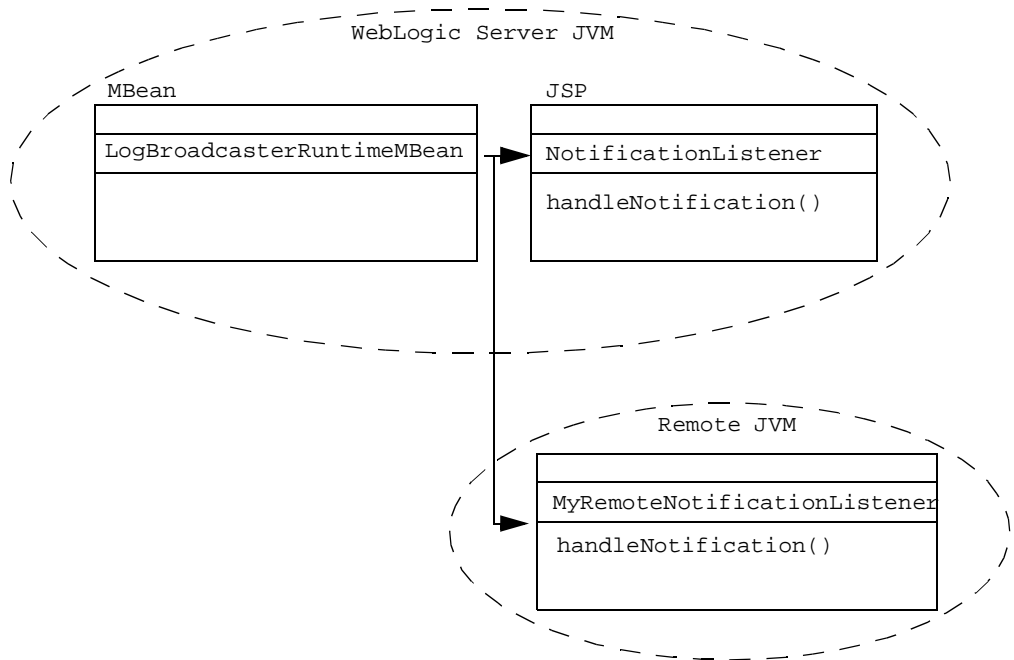
アプリケーションが WebLogic Server JVM の外部で実行される場合、次の手順に従います。

1. `WL_HOME/server/lib/weblogic_sp.jar` と `WL_HOME/server/lib/weblogic.jar` がアプリケーションのクラスパスにあることを確認します。
2. `javax.management.Notification.*` インタフェースをインポートします。
3. `weblogic.management.RemoteNotificationListener` を実装するクラスを作成します。`RemoteNotificationListener MBean` は、`javax.management.NotificationListener` および `java.rmi` を拡張することによって、RMI を介してリモート アプリケーションで通知を使用できるようにします。

実装には `RemoteNotificationListener.handleNotification()` メソッドを含める必要があります。詳細については、`weblogic.management.RemoteNotificationListener Javadoc` を参照してください。

図 4-3 に、JSP が WebLogic Server JVM 内で実行され、アプリケーションがリモート JVM で実行されるシステムを示します。メッセージをリスンするために、JSP は `NotificationListener` を実装し、リモート アプリケーションは `RemoteNotificationListener` を実装します。

図 4-3 ローカル JSP とリモート アプリケーション



コード リスト 4-2 に、リモート クライアントの通知リスナの例を示します。

コード リスト 4-2 リモート クライアント用の通知リスナの例

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;

import weblogic.management.RemoteNotificationListener;
import weblogic.management.logging.WebLogicLogNotification;

...

public class MyRemoteNotificationListener implements
    RemoteNotificationListener {

...

```

```
public void handleNotification(Notification notification, Object obj) {  
    WebLogicLogNotification wln = (WebLogicLogNotification)notification;  
}
```

手順 2 : 通知リスナの登録

通知リスナを実装したら、そのリスナを **WebLogic Server** インスタンスの `LogBroadcasterRuntimeMBean` に登録する必要があります。各 **WebLogic Server** は独自のメッセージをブロードキャストするため、各 **WebLogic Server** で通知リスナを登録する必要があります。

この節では、リスナを登録するために使用するコードについて説明します。ここに示すコード部分は、クライアントアプリケーションの起動時、**WebLogic Server** の起動時、またはアプリケーションで通知を受け取る場合に実行するクラスに追加できます。

WebLogic Server インスタンスの `LogBroadcasterRuntimeMBean` に登録するには、コードで次の処理を行う必要があります。

1. 以下のインタフェースをインポートします。

```
javax.naming.Context  
javax.naming.InitialContext  
javax.naming.AuthenticationException  
javax.naming.CommunicationException  
javax.naming.NamingException  
weblogic.jndi.Environment  
weblogic.management.MBeanHome
```

2. `MBeanHome` から `MBeanServer` を取得します。詳細については、『**WebLogic JMX Service プログラマーズ ガイド**』の「**WebLogic Server MBean へのアクセス**」を参照してください。
3. `MBeanServer` の `addNotificationListener()` メソッドを使用して、通知リスナを `LogBroadcasterRuntimeMBean` に登録します。

addNotificationListener API の使用

addNotificationListener API の構文は次のとおりです。

```
MBeanServer.addNotificationListener(ObjectName name,  
    NotificationListener listener,  
    NotificationFilter filter,  
    java.lang.Object handback)
```

以下の値を指定します。

- name は、WebLogic Server インスタンスの LogBroadcasterRuntimeMBean のオブジェクト名です。このオブジェクト名は、以下のいずれかを行うことによって取得できます。
 - インスタンス weblogic.management.WebLogicObjectName を作成する。詳細については、WebLogicObjectName Javadoc を参照してください。
 - 実行時に weblogic.management.runtime.LogBroadcasterRuntimeMBean をロックアップして .getObjectname() を呼び出す。詳細については、LogBroadcasterRuntimeMBean Javadoc を参照してください。
 - weblogic.Admin GET コマンドを使用する。詳細については、『管理者ガイド』の「GET コマンド」を参照してください。
- listener は、4-2 ページの「手順 1: 通知リスナの作成」で作成した通知リスナのインスタンスです。
- filter は、フィルタ オブジェクトです。フィルタが null の場合、通知の処理前にフィルタ処理は実行されません。フィルタ オブジェクトの作成および登録については、4-12 ページの「手順 3: 通知フィルタの作成と登録」で説明します。
- handback は、通知のブロードキャスト時にリスナに送信されるコンテキストです。

addNotificationListener API の詳細については、javax.management.MBeanServer の Javadoc (<http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>) を参照してください。

通知リスナの登録例

以下に、「手順 1：通知リスナの作成」で定義したリスナの登録例を示します。
コード リスト 4-3 とコード リスト 4-4 の例は、以下のことを行います。

1. `weblogic.management.Helper` API を使用して、`peach` というサーバ用のサーバ固有の `MBeanHome` インタフェースを取得します。`MBeanHome` インタフェースの取得については、『[WebLogic JMX Service プログラマーズ ガイド](#)』の「[WebLogic Server MBean へのアクセス](#)」を参照してください。
2. `MBeanHome` インタフェースを使用して、対応する `MBeanServer` インタフェースを取得します。
3. `LogBroadcasterRuntimeMBean` オブジェクト名を取得するための別のメソッドを使用します。
4. 「手順 1：通知リスナの作成」で定義したリスナ オブジェクトをインスタンス化します。
5. リスナ オブジェクトを `LogBroadcasterRuntimeMBean` に登録します。

コード リスト 4-3 では、`WebLogicObjectName` を使用して `LogBroadcasterRuntimeMBean` オブジェクト名を指定しています。

コード リスト 4-3 `WebLogicObjectName` の使用

```
public void find(String host,
                int port,
                String username,
                String password){

    String url = "t3://" + host +
                ":" + port;

    // サーバの MBeanHome インタフェースを取得
    try {
        serverSpecificHome = (MBeanHome)Helper.getMBeanHome(username,
                                                            password,
                                                            url,
                                                            peach);
    } catch (IllegalArgumentException iae) {
        System.out.println("Illegal Argument Exception: " + iae);
    }
}
```

4 WebLogic Server ログからのメッセージのリسن

```
//MBeanHome を使用してサーバの MBeanServer インタフェースを取得
MBeanServer mServer = serverSpecificHome.getMBeanServer();

//サーバの LogBroadcasterRuntimeMBean の WebLogicObjectName を作成
WebLogicObjectName logBCName = new WebLogicObjectName("WebLogicLogBroadcaster",
    "LogBroadcasterRuntime",
    myDomain,
    myServer);

//リスナ オブジェクトをインスタンス化
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();

//リスナを登録
mServer.addNotificationListener( logBCName,
    myListener,
    null,
    null);
}
```

コード リスト 4-4 では、MBeanHome.getMBeanByClass を使用して LogBroadcasterRuntimeMBean オブジェクト名を検索しています。

コード リスト 4-4 getObjectNames() の使用

```
public void find(String host,
    int port,
    String username
    String password){

    String url = "t3://" + host +
        ":" + port;

//サーバの MBeanHome インタフェースを取得
    try {
        serverSpecificHome = (MBeanHome)Helper.getMBeanHome(username,
            password,
            url,
            peach);
    } catch (IllegalArgumentException iae) {
        System.out.println("Illegal Argument Exception: " + iae);
    }
}
```

```
//MBeanHome を使用してサーバの MBeanServer インタフェースを取得
MBeanServer mServer = serverSpecificHome.getMBeanServer();

//getMBeanByClass を使用してオブジェクトを検索
LogBroadcasterRuntimeMBean logBCOname = (LogBroadcasterRuntimeMBean)
    home.getMBeanByClass(Class.forName
        ("weblogic.management.runtime.LogBroadcasterRuntimeMBean")
    );

// リスナ オブジェクトをインスタンス化
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();

// リスナを登録
mServer.addNotificationListener( logBCOname,
    myListener,
    null,
    null);
}
```

コード リスト 4-5 は、weblogic.Admin GET を使用して LogBroadcasterRuntimeMBean オブジェクト名を検索したことを前提としています。また、この例には weblogic.Admin GET が返すオブジェクト名のフォーマットも示されています。

コード リスト 4-5 weblogic.Admin GET の使用

```
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();
MBeanServer mServer = home.getMBeanServer();

ObjectName logBCOname = new
ObjectName("mydomain:Location=myserver,Name=TheLogBroadcaster,Type=LogBroadcasterRuntime");

mServer.addNotificationListener( logBCOname,
    myListener,
    null,
    null);
```

手順 3 : 通知フィルタの作成と登録

デフォルトでは、前節で登録した通知フィルタは `LogBroadcasterRuntimeMBean` からのすべての通知をリスンしてアプリケーションに送信します。登録フィルタを作成すると、アプリケーションに関連する通知だけを送信するよう `LogBroadcasterRuntimeMBean` をコンフィグレーションできます。フィルタでは、ユーザが作成した条件と通知が一致するかどうかをチェックされます。`true` と評価された場合にのみ、`LogBroadcasterRuntimeMBean` は通知を送信します。

この節では、以下の項目について説明します。

- フィルタの作成と登録
- `WebLogicLogNotification` オブジェクト
- 通知フィルタの例

フィルタの作成と登録

フィルタを作成するには、次の手順に従います。

1. 以下のインタフェースをインポートします。

```
import javax.management.Notification
import javax.management.NotificationFilter
import javax.management.Notification.*
```
2. 以下のことを行うシリアライズ可能なオブジェクトをクラスを作成します。
 - a. `javax.management.NotificationFilter` を実装する。
 - b. 通知の文字列を検索する。

`WebLogicLogNotification` オブジェクトとしてキャストされた通知を検索するために、`WebLogicLogNotification` ゲッター メソッドを使用できます。たとえば、ゲッター メソッドを使用すると、メッセージ タイムスタンプ、重大度、ユーザ ID、メッセージを生成したサブシステムの名

前、メッセージテキストなどを取得できます。詳細については、「WebLogicLogNotification オブジェクト」を参照してください。

- c. ブールを使用して、シリアライズ可能なオブジェクトが `true` 値を返すかどうかを示す。
 - d. (省略可能) ブールの `true` に応じてアクションを実行するコードを挿入する。たとえば、メッセージの重大度が `WARNING` 以上の場合にフィルタが `JavaMail API` を使用して管理者に電子メールを送信することができます。
3. `addNotificationListener API` を使用してフィルタを登録します。詳細については、4-8 ページの「`addNotificationListener API` の使用」を参照してください。

WebLogicLogNotification オブジェクト

WebLogic Server が生成するすべてのメッセージは、`weblogic.management.logging.WebLogicLogNotification` オブジェクトとして機能します。WebLogicLogNotification オブジェクトには以下のフィールドで構成されます。

- タイプ — JMX 仕様で必要とされたとおりに通知を識別する。このフィールドの形式は次のようになります。

```
weblogic.logMessage.subSystem.messageID
```

ここで、`subSystem` はログ メッセージを発行したサブシステムまたはアプリケーションで、`messageID` は WebLogic Server 内部のメッセージ ID です。

注意： NonCatalogLogger メッセージの場合、メッセージ ID は常に 000000 です。

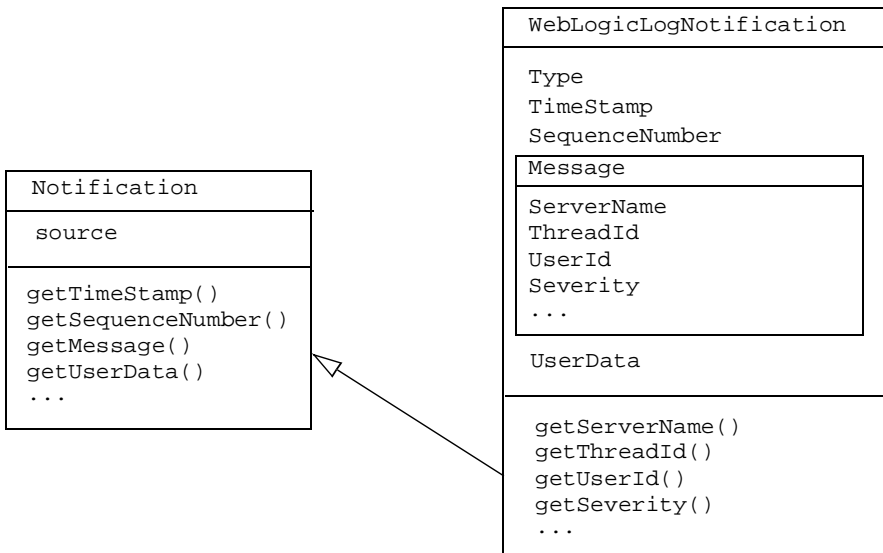
- タイムスタンプ — この通知の元となるログ メッセージがサーバによって生成された時間。
- 連続番号
- メッセージ — ログ メッセージ。
- ユーザ データ — 現在はこのフィールドは使用されていません。

4 WebLogic Server ログからのメッセージのリスン

WebLogicLogNotification は、`javax.management.Notification` からゲッター メソッド継承し、ログ メッセージ内のフィールドごとに 1つのゲッター メソッドを提供します (図 4-4 を参照)。

これらのゲッター メソッドを使用すると、WebLogicLogNotification 内の情報を検索または出力できます。詳細については、`weblogic.management.logging.WebLogicLogNotification` Javadoc を参照してください。

図 4-4 WebLogicLogNotification ゲッター メソッド



通知フィルタの例

コード リスト 4-6 に、WebLogicLogNotification.getType メソッドを使用する NotificationFilter の例を示します。

コード リスト 4-6 通知フィルタの例

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.Notification.*;
import weblogic.management.logging.WebLogicLogNotification;

....

public class MyLogNotificationFilter implements NotificationFilter,
    java.io.Serializable {

public MyLogNotificationFilter() {
    subsystem = "";
}

public boolean isNotificationEnabled(Notification notification) {
    if (!(notification instanceof WebLogicLogNotification)) {
        return false;
    }

    WebLogicLogNotification wln = (WebLogicLogNotification)notification;

    if (subsystem == null ||
        subsystem.equals("")) {
        return true;
    }

    StringTokenizer tokens = new StringTokenizer(wln.getType(), ".");
    tokens.nextToken();
    tokens.nextToken();
    return (tokens.nextToken().equals(subsystem));
}

public void setSubsystemFilter(String newSubsystem) {
    subsystem = newSubsystem;
}

}
```

索引

A

addNotificationListener メソッド 4-8
Administration Console 3-1
ALERT 重大度 2-12

C

Console、Administration 3-1
CRITICAL 重大度 2-12

E

EMERGENCY 重大度 2-12
ERROR 重大度 2-12
ExceptionName メッセージ フィールド
2-10

G

GenericServlet 2-11

H

handleNotification メソッド
ローカルアプリケーション用 4-3
定義 4-1

I

INFO 重大度 2-12

J

Java クラス ファイル 2-4
Java クラス ファイルのコンパイル 2-5
Java 仮想マシン。「リモート JVM」を参照
Java パッケージ名 2-5

JMX 1-5

JSP

NotificationListeners 4-3
GenericServlet 2-11
NotificationListeners 4-5
メッセージ カタログ 2-6

L

LocalizedTimestamp メッセージ フィー
ルド 2-10
Log4j 2-12
LogBroadcasterRuntimeMBean オブジェ
クト
オブジェクト名の取得 4-8
定義 4-1
LogBroadCasterRuntimeMBean オブジェ
クトのオブジェクト名 4-8-4-11

M

MachineName メッセージ フィールド
取得 4-13
定義 2-10
Message メッセージ フィールド 2-10, 4-13
MessageId メッセージ フィールド
取得 4-13
定義 2-10
millisecondsFromEpoch メッセージ
フィールド 2-10

N

NonCatalogLogger オブジェクト
メッセージ ID 4-13
API 2-6
勧告 2-6

メッセージフォーマット 2-10

例 2-8

NOTICE 重大度 2-12

R

RemoteNotificationListener オブジェクト 4-5

RMI 4-5

S

SequenceNumber メッセージフィールド

取得 4-13

定義 4-13

ServerName メッセージフィールド

取得 4-13

定義 2-10

Severity メッセージフィールド

取得 4-13

定義 2-10

Subsystem メッセージフィールド 2-10

T

ThreadId メッセージフィールド

取得 4-13

定義 2-10

Throwable メッセージフィールド 4-13

TimeStamp メッセージフィールド

取得 4-13

定義 4-13

TransactionId メッセージフィールド

取得 4-13

定義 2-10

Type メッセージフィールド

取得 4-13

定義 4-13

U

UserData メッセージフィールド 4-13

UserId メッセージフィールド 2-10

W

WARNING 重大度 2-12

weblogic.MsgEditor コマンド 2-2

WebLogicLogNotification オブジェクト

ゲッター メソッド 4-14

検索 4-13

使用例 4-4

WebLogicLogNotification オブジェクト
のゲッター メソッド 4-14

X

XML 2-2, 2-4

い

印刷、製品のマニュアル vi

インターナショナルライゼーション、勧告
2-1, 2-6

インタフェース、インポート

ツウチフィルタヨウ 4-12

通知リスナ 4-3-??

NonCatalogLogger API 用 2-7

通知リスナ ??-4-7

か

カスタマ サポート情報 vi

カタログ、メッセージ 2-2-2-5

ガベージコレクション 2-12

環境、設定 2-2

管理サーバ 1-5, 4-1

き

起動引数 2-11

く

クライアント JVM。「リモート JVM」を
参照

クライアントアプリケーション。「リモート
アプリケーション」を参照

クラスパス 2-2, 4-5
クラス ファイル 2-4

さ

サーブレット 2-11
サポート
 技術情報 vi
サンプル
 リモート アプリケーション用の通知
 リスナ 4-6
 ローカル アプリケーション用の通知
 リスナ 4-4
通知フィルタ 4-15
通知リスナの登録 4-9
NonCatalogLogger メッセージ 2-8
メッセージ カタログ 2-3
メッセージ カタログの使用 2-6

し

重大度
 数値 2-12
 定義 2-12
 メッセージの除外に使用 2-12

つ

通知
 ブロードキャストのデフォルト 4-12
 「メッセージ」を参照
 定義 4-1
通知フィルタ
 addNotificationListener メソッド
 での指定 4-8
 作成と登録 4-12
 定義 4-12
 例 4-15
通知リスナ
 デフォルト動作 4-12
 リモート アプリケーション用、例 4-6
 ローカル アプリケーション用、例 4-4
 登録例 4-9

起動 4-7
定義 4-1
登録 4-7
通知リスナの起動 4-7

て

デバッグ メッセージ 1-5, 2-12, 2-13, 4-1
電子メール 4-1, 4-13

は

パス、設定。「クラスパス」を参照 2-2
パッケージ名 2-5

ひ

標準出力 2-11, 3-2

ふ

ブロードキャスト
 WebLogic Server から 4-1
 リモート アプリケーションから 4-2
 WebLogic Server から 4-1, 4-7

ま

マニュアル、入手先 v

め

メッセージ エディタ GUI 2-2
メッセージ カタログ 2-2-2-5
メッセージの出力 4-4
メッセージのフォーマット。「メッセージ
 フォーマット」を参照
メッセージ ログの表示 3-1
メッセージ。ログ メッセージを参照

り

リモート JVM

- NonCatalogLogger メッセージ 2-6
 - 起動引数 2-11
 - 通知リスナ 4-5
 - 標準出力への書き込み 2-11
 - ログ ファイル 2-11
- リモート JVM の起動オプション 2-11
- リモート JVM の起動パラメータ 2-11
- リモート JVM の起動引数 2-11
- リモート アプリケーション 2-11
- リモート アプリケーションのメッセージの場所 2-11

- ログ メッセージ フォーマット
 - NonCatalogLogger 2-10
 - メッセージ カタログ 2-2

ろ

- ローカライゼーション

 - 勧告 2-1

 - プロパティ ファイル 2-5

- ローカライゼーション用プロパティ ファイル 2-5

- ログ メッセージ

- ログ メッセージの除外 2-12

- ログ ビューア 3-1

- ログ ファイル

 - ドメイン用 4-1

 - リモート JVM 2-11

 - リモート アプリケーション 2-11

- ログ メッセージ

 - ログ ビューアでの検索 3-1

 - 「通知」を参照

 - 出力 4-4

 - 表示 3-1

 - サーブレットから 2-11

 - デバッグの除外 2-12

- ログ メッセージ ID

 - NonCatalogLogger メッセージ用 4-13

 - 例 2-3

- ログ メッセージ テキスト

 - NonCatalogLogger 2-6, 2-10

 - 検索 4-13

 - フィルタ処理 4-12

 - メッセージ カタログ 2-2

 - 例 2-4