BEA WebLogic Server™

WebLogic JMX Service
プログラマーズ ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

WebLogic JMX Service プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1 サービス パック 3

目次

このマニュアルの内容

1. WebLogic JMX サービスの概要

2. WebLogic Server MBean へのアクセス

3. MBean 通知の使い方

4. WebLogic Server MBean のモニタ

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ の Management API を使用して、アプリケーションをサポートするために WebLogic Server を強化する方法について説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic JMX サービスの概要」では、WebLogic Server 管理インタフェースについて説明し、MBean、管理ドメイン、およびサーバ コンフィグレーションについて概説します。
- 第 2 章「WebLogic Server MBean へのアクセス」では、クライアント アプリケーションから WebLogic Server MBean にアクセスして使用する方法について説明します。
- 第 3 章「MBean 通知の使い方」では、クライアント アプリケーションで MBean 通知をリスンおよび応答する方法について説明します。
- 第 4 章「WebLogic Server MBean のモニタ」では、モニタ用 MBean から MBean の属性をモニタする方法について説明します。

対象読者

このマニュアルは、BEA WebLogic Server のコア テクノロジを使用するカスタム アプリケーションの作成に関心がある独立ソフトウェア ベンダ（ISV）とその他の開発者を対象としています。BEA WebLogic Server プラットフォームと 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のメイン トピックを一度に 1 つずつ印刷できます。

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

Adobe Acrobat Reader は、Adobe の Web サイト（http://www.adobe.co.jp）から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。以下の BEA WebLogic Server のマニュアルには、WebLogic Server を拡張する方法を理解するための関連情報が含まれています。

- BEA WebLogic Server のマニュアル（オンラインで入手可能）
  - 管理者ガイド
  - プログラミング ガイド
  - WebLogic Server API

- Sun Microsystems, Inc. の Java サイト（http://java.sun.com/）

BEA WebLogic Server と Java の詳細については、
http://edocs.beasys.co.jp/e-docs/ の参考文献を参照してください。

サポート情報

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

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成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〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

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

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

表記法	適用
[ ]	構文内の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 JMX サービスの概要

以下の節では、WebLogic Server の JMX 管理フレームワークについて概説します。

概要

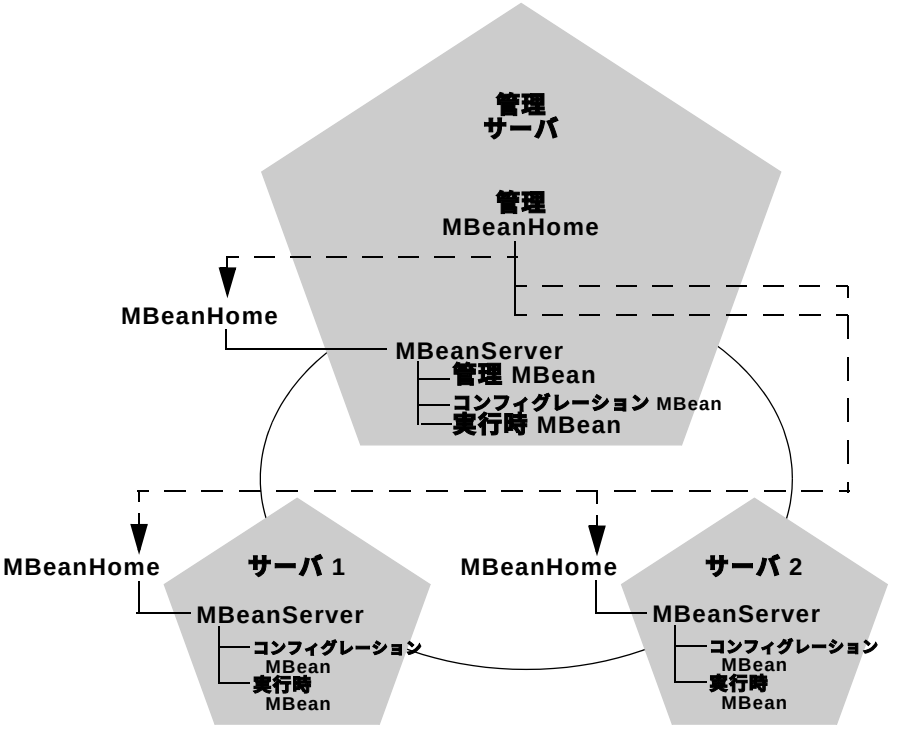
WebLogic Server 管理アーキテクチャは、Sun Microsystems の Java Management Extension（JMX）仕様をベースとしています。BEA では、WebLogic Server で使用できる数多くの API およびリソースをあらかじめ用意しており、それらのリソースをモニタおよび管理するための JMX に準拠した Administration Console を提供しています。

すべての WebLogic Server 管理機能には、Management Bean（MBean）を使用してアクセスできます。MBean は WebLogic Server のドメイン コンフィグレーションまたは実行時の状態からその値を取得します。MBean を使用すると、開発者は JMX 標準 API を通じて WebLogic Server に関するすべてのコンフィグレーションおよびモニタ情報にプログラムでアクセスできます。

このガイドでは、WebLogic Server の JMX 実装について概説します。その結果、WebLogic Server の、JMX で管理できるリソースをモニタおよび管理するアプリケーションや管理フレームワークを開発できるようになります。

WebLogic Server の管理システム

WebLogic Server の管理システムでは、JMX 1.0 仕様で示されている必須のコンポーネントがすべて実装されています。WebLogic Server インストールには複数のサーバを含めることができるので、JMX コンポーネントは必ずインストール全体に分散されます。次の図は、典型的な WebLogic Server インストールにおける JMX コンポーネントを表しています。



以降の節では、各 JMX コンポーネントについて概説します。

管理対象リソース

管理対象リソースには、WebLogic Server インスタンスでホストされる API、サービス、アプリケーションなどがあります。それらの API、サービス、アプリケーションは、JMX による管理用にあらかじめ用意されているものです。各管理対象リソースには、リソースのモニタまたは変更に使用される 1 つまたは複数の MBean があります。

WebLogic に用意されている API には、EJB、JDBC、JMS、JTA、XML などがあります。スタートアップ クラス、シャットダウン クラス、セキュリティ レルムなどの WebLogic Server サービスもあらかじめ用意されています。さらに、WebLogic Server では Web アプリケーションとそのコンポーネントに対しても JMX による管理が提供されているので、JMX 仕様を使用してアプリケーションのデプロイメント パラメータを変更したり、そのデプロイメント ステータスをモニタしたりできます。

MBean

MBean（管理対象 Bean）は、管理対象リソースを表す JMX の構成概念です。WebLogic Server の各管理対象リソース（API、サービス、アプリケーション コンポーネント）では、1 つまたは複数の MBean を使用して、リソースをモニタまたは変更するためのインタフェースが用意されます。

WebLogic Server MBean には、JMX 仕様で定義されている、以下のような標準の処理ツールがすべて用意されています。

- MBean をインスタンス化するためのコンストラクタ
- MBean の属性を表示、設定および取得するためのメソッド
- MBean 固有の追加操作を実行するためのメソッド
- MBean イベントをブロードキャストするための通知

JMX 仕様に合わせて、すべての WebLogic MBean は標準的な MBean として実装され、その属性と処理は関連付けられたインタフェースで直接指定されます。WebLogic Server では、MBean のいくつかのタイプが定義されており、

WebLogic Server の管理システム内でのそれぞれの機能が指定されています。特定の MBean タイプの詳細については、「WebLogic Server MBean」を参照してください。

MBeanServer

JMX 仕様で説明されているように、MBeanServer は管理フレームワーク内の MBean にアクセスするための原則的なエージェントです。MBeanServer は、MBean のレジストリとして機能します。MBeanServer を使用すると、管理アプリケーションで MBean をルックアップし、MBean の属性およびメソッドを判別し、MBean 通知をリスンできます。

WebLogic 管理ドメイン内の各サーバには、独自の MBeanServer があります。各 MBeanServer（管理サーバの MBeanServer を除く）には、ローカルの WebLogic Server インスタンスに適用される MBean しか登録されません。たとえば、ある管理対象 WebLogic Server インスタンスの MBeanServer を使用するアプリケーションでは、その特定のサーバにデプロイされた Web アプリケーションをモニタできますが、そのドメイン内の別のサーバにデプロイされた Web アプリケーションはモニタできません。

管理サーバは、それ自体が WebLogic Server インスタンスなので、同様に MBeanServer を保持しています。管理サーバの MBeanServer は、そのサーバ独自のコンフィグレーション MBean と実行時 MBean だけでなく、ドメイン全体の管理 MBean のホストとなる点でユニークなものです。

MBeanHome

JMX 1.0 仕様では、MBeanServer インタフェースを MBeanServer の JVM の外部にある管理クライアントで使用できるようにするためのガイドラインが示されていません。WebLogic Server バージョン 6.1 では、`MBeanHome` インタフェースを通じて、MBeanServer インタフェースを、任意のクライアント（ローカルのクライアントまたはサーバの JVM の外部にあるクライアント）で使用できるようになっています。

MBeanHome は、WebLogic Server MBean へのアクセスに使用できる MBeanServer インタフェースのラッパーに過ぎません。ほとんどの場合、アプリケーションでは MBeanServer の代わりに MBeanHome を使用してサーバ リソースを管理できます。MBeanServer の MBeanHome を取得する単純な JNDI ルックアップを使用することで、すべてのクライアントから管理機能にアクセスできます。

MBeanHome インタフェースでは、WebLogic Server MBean の属性にアクセスするための、タイプ分けされたインタフェースが提供されているので、MBeanHome のほうが、一般的に MBeanServer より使いやすくなっています。たとえば、アプリケーションが MBeanHome から serverMBean を取得したら、serverMBean.getListenPort() を呼び出して、サーバのリスン ポートの int 値を返すことができます。MBeanServer を使用して同様の処理を実行するには、アプリケーションでまず、serverMBean の JMX オブジェクト名を取得し、その ListenPort 属性を要求する必要があります。属性自体が汎用オブジェクトとして返されるので、アプリケーションではその特定の属性を int 値にキャストする必要があります。

MBeanHome は、WebLogic Server MBean にアクセスする場合にのみ使用できます。MBeanHome を使用して、ユーザ定義 MBean を取得することはできません。登録済みのユーザ MBean にアクセスする必要のある純粋な JMX アプリケーションでは、まず MBeanHome をルックアップし、getMBeanServer() を呼び出すことによって、MBeanServer インタフェースを取得および使用できます。

先ほどの図に示されているように、管理ドメイン内の各 WebLogic Server インスタンスには MBeanServer と、対応する MBeanHome があります。これらの MBeanHome インタフェースを使用すると、個々の WebLogic Server のコンフィグレーション MBean と実行時 MBean をアプリケーションで処理できます。

管理サーバには、標準の MBeanHome インタフェース（サーバ インスタンスのローカルのコンフィグレーション MBean と実行時 MBean 用）だけでなく、以下で説明するような、ドメイン全体の MBeanHome インタフェースも提供されます。

管理 MBeanHome

WebLogic Server の管理システムでは、管理ドメイン内の全サーバ インスタンスの WebLogic MBean すべてにアクセスできる、ドメイン全体の MBeanHome インタフェースが利用されます。管理 MBeanHome には、管理サーバおよびすべての管理対象サーバのコンフィグレーション MBean と実行時 MBean に加えて、ドメイン全般の管理 MBean が含まれます。

ドメイン全体の MBeanHome インタフェースには、関連付けられた MBeanServer がありませんが、サーバ固有の MBeanHome と同じように操作できます。使用可能な MBean のリストだけが異なります。管理 MBeanHome は、先ほどの図に点線で示されているように、個々のサーバの MBeanHome インタフェースを使用して、別のサーバの MBean にアクセスします。

アプリケーションは、WebLogic 管理サーバを通じて、ドメイン全体の MBeanHome インタフェースを取得します。アプリケーションでは、ドメイン全体の MBeanHome を取得してから、ドメイン内の使用可能な MBean のリストをフィルタ処理することで、ドメイン全体の管理 MBean や個々のサーバの MBean を処理できます。「WebLogic Server MBean へのアクセス」では、MBeanHome インタフェースをプログラムで取得する方法や、アプリケーションで取得する必要のある MBeanHome インタフェースに関する情報について説明しています。

WebLogic Server MBean

WebLogic Server では、以下の 3 つの MBean タイプが定義されています。

- 管理 MBean。config.xml から読み込まれる、ドメイン全体のコンフィグレーション パラメータを表します。
- コンフィグレーション MBean。管理 MBean のサーバごとのコピーです。サーバではこれを使用してコンフィグレーションが行われます。
- 実行時 MBean。さまざまな WebLogic Server コンポーネントおよびサブシステムの実行時の状態を表します。

以降の節では、各 MBean のタイプについて説明します。

管理 MBean

管理 MBean は、WebLogic Server 管理ドメイン全体のコンフィグレーション済みプロパティを表します。ドメインの管理サーバを起動すると、ドメインの config.xml ファイルで指定されている要素および属性を使用して、サーバによって管理 MBean が作成されます。

すべての管理 MBean は、管理サーバが起動すると自動的に登録されます。まだ実行されていない管理対象サーバや、まだ管理サーバに接続されていない管理対象サーバの管理 MBean も登録されます。

JMX 管理アプリケーションでは、管理 MBean の属性を変更することで、間接的に管理ドメインの `config.xml` ファイルを変更できます。管理 MBean サーバでは 5 分ごとに管理 MBean が変更されているかどうかがチェックされ、必要に応じて変更が `config.xml` に書き込まれます。

管理 MBean に対する変更は、管理サーバの停止時や、WebLogic Server ユーティリティ（Administration Console、`weblogic.Admin`、`weblogic.Deploy` など）による MBean 属性の変更時にも `config.xml` ファイルに書き込まれます。

コンフィグレーション MBean

管理 MBean が `config.xml` 要素の永続的な値を表すのに対して、コンフィグレーション MBean は同じ要素の「アクティブな」値を表します。これは、WebLogic Server サブシステムがサーバの有効期間中に処理に使用する、コンフィグレーション属性のアクティブな値（コンフィグレーション MBean）です。

WebLogic Server を起動すると、そのコンフィグレーション MBean の大部分は、サーバの管理 MBean から管理サーバに登録されているとおりに派生されます。たとえば、管理対象サーバは管理サーバに接続して、そのすべてのコンフィグレーション MBean を、関連付けられた管理 MBean から派生させることができます。

ただし、WebLogic Server の起動時にコマンドライン オプションを使用して、`config.xml` プロパティをオーバーライドすることもできます。この場合、サーバのコンフィグレーション MBean 属性は、管理 MBean の値ではなく、オーバーライド値から派生されます。その後、オーバーライド値のない属性が、管理サーバに登録されている管理 MBean から派生されます。

JMX アプリケーションでは、コンフィグレーション MBean を変更して、アクティブな WebLogic Server インスタンスのコンフィグレーションを一時的に変更できます。ただし、コンフィグレーション MBean に対する変更は、サーバが再起動したり、停止したりすると失われます。サーバのコンフィグレーションに対して永続的な変更を行うには、アプリケーションで、対応するリソースの管理 MBean を変更する必要があります。この変更は、自動的に `config.xml` ファイ

ルに書き込まれます。管理 MBean に対する変更は、対応するコンフィグレーション MBean にも影響します。その結果、WebLogic Server サブシステムでは、新しくコンフィグレーションされた属性値が使用されます。

実行時 MBean

実行時 MBean は、それが表す基盤リソースまたはサブシステムの実行時の一時的な状態を表します。実行時 MBean は、属性値が派生されたり、オーバーライドされたりせず、指定された時間のサーバ リソースの状態を表すという点で、管理 MBean やコンフィグレーション MBean とは異なります。

たとえば、実行時 MBean は、WebLogic Server で現在利用可能なソケットの数や、サーバの現在の状態（実行中、サスペンド中、停止されようとしている、など）を表す場合に使用されます。

アプリケーションでは、実行時 MBean を使用して、Web アプリケーションなどの管理対象リソースのリソース使用状況をモニタし、潜在的なパフォーマンスの問題点を診断できます。

MBean の命名規約

すべての WebLogic Server MBean には、名前、タイプ、ドメインがあります。これらの属性は MBean の JMX オブジェクト名に反映されます。オブジェクト名は、指定した MBean に対する、すべてのドメインにわたってユニークな識別子であり、次のような構造を持っています。

domain name:Name=*name*,Type=*type*[,*attr*=*value*]...

Name は、MBean の指定したドメインとタイプに対するユニークな識別子です。

Type は、MBean が表す管理対象リソースのタイプを示します。リソース タイプの例として、Server、WebComponent、または JDBCConnectionPoolRuntime などがあります。Type は、以下に示す標準的なサフィックスを付加することで、管理、コンフィグレーション、実行時の各 MBean の識別にも使用されます。

- ＜サフィックスなし＞は、管理 MBean を示します。
- Config は、コンフィグレーション MBean を示します。

- Runtime は、実行時 MBean を示します。

たとえば、Type の値が JDBCConnectionPool の MBean ならば、

- 管理 MBean の場合は JDBCConnectionPool となります。
- コンフィグレーション MBean の場合は JDBCConnectionPoolConfig となります。
- 実行時 MBean の場合は JDBCConnectionPoolRuntime となります。

「MBean」サフィックスは、MBean の基本タイプを取得するために MBean インタフェース名から削除されています。JDBCConnectionPool の MBean の場合、実際の MBean インタフェース名は JDBCConnectionPoolMBean です。

特定のタイプの MBean には、JMX オブジェクト名に追加属性があります。すべての実行時 MBean とコンフィグレーション MBean には Location コンポーネントがあります。このコンポーネントは、MBean が格納されているサーバの名前を値として使用します。For example:

```
mydomain:Name=myServlet,Type=ServletRuntime,Location=myserver
```

親 MBean と子の関係を持つすべての MBean は、そのオブジェクト名に関係を識別するための追加属性を持ちます。属性の形式は、次のようになります。

TypeOfParentMBean=NameOfParentMBean

次の例では、Server は親 MBean のタイプ、myserver は親 MBean の名前を表します。

```
mydomain:Name=mylog,Type=Log,Server=myserver
```

パッケージの命名規約

管理 MBean およびコンフィグレーション MBean のすべてのインタフェース タイプは、weblogic.management.configuration API に含まれています。

実行時 MBean のすべてのインタフェース タイプは、weblogic.management.runtime API に含まれています。

エージェントレベルのインタフェース（MBeanHome インタフェース、RemoteMBeanServer インタフェースなど）は、weblogic.management API に含まれています。

WebLogic Server MBean のクイック リファレンス

WebLogic Server には、サーバ システムおよびアプリケーションのコンフィグレーションに使用される数多くの MBean が用意されています。多くの場合、関連する MBean には、1 つまたは複数の関連する MBean を取得するためのゲッターを提供する「親」MBean を通じて、簡単にアクセスできます。この節では、JMX プログラミングに役立つ WebLogic Server MBean の主要なカテゴリの概要を説明すると共に、そのクイック リファレンスを提供します。

注意： WebLogic Server Management API は、Javadoc でオンライン ドキュメント化されています。WebLogic Server のプログラミング ガイドには、WebLogic Server MBean によってモデル化されるプログラミング API およびサービスに関する詳細な情報が記載されています。

ドメイン MBean

`DomainMBean` は、管理ドメイン全体をあらわす、高レベルの WebLogic Server MBean です。`DomainMBean` を取得したら、そのゲッター メソッドを使用して、次の図で示されているような、ドメインのログ、セキュリティ、SNMP、JTA コンフィグレーションを表す MBean を取得できます。同様に、`SecurityMBean` や `SNMPAgentMBean` などの MBean は、そのコンフィグレーションの一部を指定する MBean にアクセスするためのゲッターを提供します。

対象 MBean

対象 MBean は、管理ドメイン内にアプリケーションおよびリソースをデプロイするときに選択できるオブジェクトを表します。これらのオブジェクトには、WebLogic Server インスタンスや WebLogic Server クラスタを表す MBean があります。

すべての対象 MBean では、`weblogic.management.configuration.TargetMBean` インタフェースが実装されています。つまり、クラスタ、サーバの両方を、接続プールなどのリソースや、アプリケーション コンポーネントのデプロイの対象として選択できます。

サーバ MBean およびカーネル MBean

ServerMBean は、KernelMBean インタフェースを拡張したものであり、管理ドメイン内の特定の WebLogic Server インスタンスを表します。ServerMBean を取得するアプリケーションでは、関連付けられたサーバおよび WebLogic Server カーネルのコンフィグレーションを指定する、子 MBean に簡単にアクセスできます。

次の図は、KernelMBean インタフェースおよび ServerMBean インタフェースのゲッター メソッドを使用して取得できる、子 MBean を表しています。

クラスタ MBean

アプリケーションでは、Web アプリケーションや WebLogic Server リソースのデプロイ時に、コンフィグレーション済みの WebLogic Server クラスタを対象として選択することもできます。`ClusterMBean` は、主にゲッター メソッドおよびセッター メソッドを使用して、ロード バランシング アルゴリズム、マルチキャスト メッセージ プロパティなどのクラスタ プロパティをコンフィグレーションします。また、`ClusterMBean` には、クラスタのメンバーである、すべての `ServerMBean` を返すゲッター メソッドもあります。

デプロイ可能なユニット MBean

数多くの WebLogic Server MBean で `DeploymentMBean` インタフェースが実装されています。`DeploymentMBean` は、ドメイン内のサーバまたはクラスタにデプロイできる Web アプリケーション、Web アプリケーション コンポーネント、または WebLogic Server リソースの任意のタイプを表します。

デプロイ可能なユニットの操作に関心がある場合は、まず weblogic.management.configuration.DeploymentMBean に慣れておいてください。このインタフェースでは、対象を取得または追加したり、デプロイメント順を設定したりする場合に使用する基本的なメソッドが提供されています。

`DeploymentMBean` の基本的な操作に慣れたら、このインタフェースを実装する個々のデプロイ可能なユニット MBean を参照してください。デプロイ可能なアプリケーション コンポーネントを表す MBean は、以下のとおりです。

- ComponentMBean
- ConnectorComponentMBean
- EJBComponentMBean
- ShutdownClassMBean
- StartupClassMBean
- VirtualHostMBean
- WebAppComponentMBean
- WebDeploymentMBean

- WebServerMBean

デプロイ可能な WebLogic Server リソースを表す MBean は、以下のとおりです。

- JDBCConnectionPoolMBean
- JDBCDataSourceMBean
- JDBCMultiPoolMBean
- JDBCTxDataSourceMBean
- JMSConnectionFactoryMBean
- JMSServerMBean
- MessagingBridgeMBean
- RMCFactoryMBean
- WLECConnectionPoolMBean

2 WebLogic Server MBean へのアクセス

以下の節では、クライアント アプリケーションまたは管理フレームワークから WebLogic Server MBean にアクセスする方法について説明します。

概要

「WebLogic Server の管理システム」で説明したように、クライアントは 2 つの主要なエージェントレベル インタフェース、`MBeanServer` および `MBeanHome` を使用して MBean にアクセスできます。

WebLogic Server の JVM の内部または外部のクライアントから MBean にアクセスするための簡単で、タイプ分けされたインタフェースとして、`MBeanHome` が提供されています。アプリケーションで、MBean への完全に JMX に準拠したアクセスが必要な場合は、`MBeanHome` を通じて `MBeanServer` インタフェースを取得することもできます。

以降の節では、`MBeanHome` を取得し、`MBeanHome` インタフェースを通じて WebLogic Server MBean にアクセスする基本的な手順について説明します。MBeanServer インタフェースを通じた MBean へのアクセスに関する詳細については、JMX 仕様を参照してください。

WebLogic Server MBean に対するクライアント インタフェースの選択

ドメイン内の各サーバには、そのサーバのコンフィグレーション MBean および実行時 MBean のホストとなる MBeanHome（および対応する MBeanServer）が存在します。さらに管理サーバには、ドメイン全体のすべての MBean へのアクセスを提供する、管理 `MBeanHome` があります。アプリケーションで使用するインタフェースの選択は、以下の要因によって決まります。

- アプリケーションを完全な JMX 準拠にする必要があるかどうか。
- アプリケーションでユーザ定義 MBean にアクセスする必要があるかどうか。
- アプリケーションで管理 MBean を処理するかどうか。
- アプリケーションで 1 つの WebLogic Server を管理するか、または複数の WebLogic Server を管理するか。

MBeanHome と MBeanServer

`MBeanHome` インタフェースを使用すると、WebLogic Server MBean に簡単にアクセスできます。ただし、`MBeanServer` インタフェースに登録されている可能性のあるユーザ定義 MBean にはアクセスできません。アプリケーションでユーザ定義 MBean にアクセスする必要がある場合には、必ず、`MBeanServer` インタフェースを使用してアクセスしてください。

アプリケーションを完全に JMX 仕様に準拠させる必要がある場合には、`javax.management.MBeanServer` インタフェースの使用を選択することもできます。ただし、`MBeanHome` インタフェースでは、WebLogic MBean にアクセスするための、タイプ分けされたインタフェースが提供されているので、このインタフェースのほうが `MBeanServer` より一般的に使いやすくなっています。

注意： この章のすべての例で、`MBeanHome` が MBean にアクセスするための主要な方法として使用されています。`MBeanServer` の使用については、JMX 仕様を参照してください。

サーバ MBeanHome と管理 MBeanHome

アプリケーションでは、アクセスする MBean に応じて、個々のサーバの MBeanHome インタフェースや管理 MBeanHome インタフェースを使用できます。

アプリケーションで管理 MBean を管理する必要がある場合には、必ず、管理サーバのドメイン全体の MBeanHome インタフェースを使用してください。管理対象サーバの MBeanHome インタフェースを通じては、管理 MBean にアクセスできないからです。

アプリケーションでドメイン内の複数の WebLogic Server インスタンスを管理する場合には、ドメイン全体の MBeanHome インタフェースを使用したほうが望ましいこともあります。ドメイン全体のインタフェースを使用して、JMX オブジェクト名をフィルタ処理することによって、管理ドメイン内の任意の WebLogic Server から MBean にアクセスできます。

アプリケーションでドメイン内の WebLogic Server インスタンスを 1 つだけ管理する場合には、ドメイン全体の MBeanHome ではなく、そのサーバのローカルの MBeanHome インタフェースを取得したほうがよいこともあります。ローカルのインタフェースを使用することで、そのサーバに適用される MBean を識別するために MBean をフィルタ処理する手間を省くことができます。また、管理対象サーバ自体に直接、接続しているので、ローカルのインタフェースではより少ないネットワーク ホップで MBean にアクセスできます。

JNDI を使用した MBeanHome の取得

サーバの MBeanHome は、MbeanHome.LOCAL_JNDI_NAME 定数を使用して、関連サーバの JNDI ツリーから取得できます。

ドメイン全体の管理 MBeanHome は、管理サーバの JNDI ツリーで MBeanHome.ADMIN_JNDI_NAME により発行されます。

また、この管理サーバは、その JNDI ツリー上のドメイン内のサーバごとに MBeanHome を発行します。この MBeanHome は管理サーバの JNDI ツリーから取得でき、MbeanHome.JNDI_NAME+"."+*relevantServerName* 定数を使用してアクセスできます。

個々のサーバの MBeanHome の javax.management.MBeanServer は、その MBeanHome の getMBeanServer() メソッドを呼び出すことによって取得できます。ドメイン全体の MBeanHome には、対応する javax.management.MBeanServer はありません。管理 MbeanHome の getMBeanServer() メソッドを呼び出すと、管理サーバの MBeanServer が返されます。

例 : 外部クライアントからの MBeanHome のルックアップ

次の例は、異なる JVM で実行中のアプリケーションで、管理サーバの MBeanHome インタフェースをルックアップする方法を示しています。

注意 : この例では、非推奨の MBeanHome.JNDI_NAME 定数を使用して、特定のサーバの MBeanHome を取得しています。アプリケーションでは、この代わりに「JNDI を使用した MBeanHome の取得」に記載されている JNDI 名を使用してください。

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
...
public void findExternal(String host,
                         int    port,
                         String password) {
        String url = "t3://" + host +
                     ":" + port;
```

```
    String username = "system";
    try {
        Environment env = new Environment();
        env.setProviderUrl(url);
        env.setSecurityPrincipal(username);
        env.setSecurityCredentials(password);
        ctx = env.getInitialContext();
        home = (MBeanHome)ctx.lookup(MBeanHome.JNDI_NAME + "." +
                                     SERVER_NAME);
        System.out.println(SERVER_NAME +
                           " MBeanHome found externally");
        ctx.close();
    } catch (AuthenticationException ae) {
        System.out.println("Authentication Exception: " + ae);
    } catch (CommunicationException ce) {
        System.out.println("Communication Exception: " + ce);
    } catch (NamingException ne) {
        System.out.println("Naming Exception: " + ne);
    }
}
```

例 : 内部クライアントからの MBeanHome のルックアップ

管理サーバ（または、モニタする WebLogic Server インスタンス）と同じ JVM にクライアント アプリケーションが存在する場合、MBeanHome の JNDI ルックアップはより簡素なものになります。次の例は、管理サーバと同じ JVM で実行中の JSP で、MBeanHome をルックアップする方法を示しています。

注意 : この例では、非推奨の MBeanHome.JNDI_NAME 定数を使用して、特定のサーバの MBeanHome を取得しています。アプリケーションでは、この代わりに「JNDI を使用した MBeanHome の取得」に記載されている JNDI 名を使用してください。

```
...
public void findInternal() {
        Environment env = new Environment();


        try {
            ctx = env.getInitialContext();
            home = (MBeanHome)ctx.lookup(MBeanHome.JNDI_NAME + "." +
                                         SERVER_NAME);
            System.out.println(SERVER_NAME +
                               " MBeanHome found internally");
            ctx.close();
        } catch (NamingException ne) {
            System.out.println("Naming Exception: " + ne);
        }
    }
```

例 :MBeanHome からの MBeanServer の取得

アプリケーションで、MBeanServer インタフェースと直接対話する必要がある場合は、MBeanHome を使用すると、それに関連付けられた MBeanServer を簡単に取得できます。

注意 : この例では、非推奨の MBeanHome.ADMIN_JNDI_NAME 定数を使用して、管理サーバの MBeanHome を取得しています。アプリケーションでは、この代わりに「JNDI を使用した MBeanHome の取得」に記載されている JNDI 名を使用してください。

```
...
home = (MBeanHome)ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
RemoteMBeanServer homeServer = (RemoteMBeanServer)home.getMBeanServer();
...
```

ヘルパー クラスを使用した MBeanHome インタフェースの取得

WebLogic Server バージョン 6.1 では、内部クライアントの MBeanHome インタフェースの取得プロセスを簡素化する weblogic.management.Helper クラスが提供されています。Helper クラスは、サーバまたは管理 MBeanHome を取得するメソッドを提供します。

たとえば、Helper クラスを使用して、管理サーバとローカル サーバの両方の MBeanHome を取得するには、次のように記述します。

```
public void find(String host,
                 int port,
                 String password) {
        String url = "t3://" + host +
                 ":" + port;
```

```
        try {
            localHome = (MBeanHome)Helper.getMBeanHome("system",
                                                       password,
                                                       url,
                                                       SERVER_NAME);
            adminHome = (MBeanHome)Helper.getAdminMBeanHome("system",
                                                            password,
                                                            url);
            System.out.println("Local and Admin Homes " +
                               "found using the Helper class");
        } catch (IllegalArgumentException iae) {
            System.out.println("Illegal Argument Exception: " + iae);
        }
    }
```

MBeanHome からの MBean へのアクセス

MBeanHome を取得したら、javax.management.MBeanHome で説明されているメソッドを使用して個々の MBean をルックアップできます。たとえば、管理 MBeanHome のすべての MBean をルックアップし、その JMX オブジェクト名を出力するには、次のように記述します。

```
public void displayMBeans() {
        Set allMBeans = home.getAllMBeans();
        System.out.println("Size: " + allMBeans.size());
        for (Iterator itr = allMBeans.iterator(); itr.hasNext(); ) {
            WebLogicMBean mbean = (WebLogicMBean)itr.next();
            WebLogicObjectName objectName = mbean.getObjectName();
```

```
            System.out.println(objectName.getName() +
                               " is a(n) " +
                               mbean.getType());
        }
    }
```

MBeanHome.getMBean() メソッドを使用することで、個々の MBean にアクセスできます。getMBean() には、いくつかの異なるメソッド シグネチャがあります。最も単純なシグネチャは、デフォルト ドメインで指定された名前とタイプを持つ WebLogicMBean を返します。

MBeanHome では、特定の WebLogic Server MBean タイプを取得するための、追加のゲッター メソッドが提供されています。たとえば、現在のドメインから Server のコンフィグレーション MBean を取得するには、getConfigurationMBean() メソッドを使用できます。

```
String myBeanType = "ServerConfig";
ConfigurationMBean myServerMBean =
        home.getConfigurationMBean(SERVER_NAME, myBeanType);
```

実行時 MBean を取得するには、getRuntimeMBean() メソッドを使用します。**実行時 MBean** とは、WeLogic Server とアプリケーションのコンポーネントに関する実行時情報を与えるローカル MBean です。他の MBeanHome メソッドとは異なり、getRuntimeMBean() は、現在の WebLogic Server 上にある実行時 MBean だけを返します。管理サーバ上の MBeanHome.getRuntimeMBean() を呼び出しても、管理対象サーバから実行時 MBean を返すことはありません。たとえば、以下のようなコードになっている箇所では、現在の WebLogic Server から JDBCConnectionPoolRuntime MBean を返します。

```
String poolName = "requestConnectionPool";
JDBCConnectionPoolRuntimeMBean runtimeMBean =
        (JDBCConnectionPoolRuntimeMBean)home.getRuntimeMBean(poolName,
        "JDBCConnectionPoolRuntime");
```

MBeanHome で使用できる各ゲッター メソッドについては、weblogic.management.MBeanHome の Javadoc を参照してください。

カスタム MBean の MBeanServer への登録

WebLogic Server 管理サービスは JMX を使用して実装されているので、独自の MBean を作成し、インストールされている WebLogic Server の MBeanServer に登録することもできます。登録すると、WebLogic Server の MBeanServer 実装を利用して独自の MBean のホストにし、内部および外部クライアントで独自の MBean を使用できるようになります。

すべてのカスタム MBean は、登録され、JMX に準拠した MBeanServer インタフェースを使用してアクセスされなければなりません。カスタム MBean に対しては、MBeanHome インタフェースを使用できません。MBeanHome は、WebLogic Server MBean のみをクライアントで使用できるようにします。さらに、weblogic.Admin などの BEA ユーティリティを使用してカスタム MBean にアクセスすることもできません。

次の例は、基本的な MBean 実装と、MBean を管理サーバの MBeanServer に登録するクライアント アプリケーションを示しています。ただし、この例では、JMX 仕様で説明されているすべての要件（MBean の例外処理など）が示されているわけではありません。独自のカスタム MBean の実装に関する詳細については、JMX 仕様を参照してください。

カスタム MBean の例

この例では、カスタム MBean は、1 つのメソッド実装のみを必要とする省略されたインタフェースで構成されています。

```
public interface MyCustomMBean {
     int getMyAttribute();
}
```

クライアント アプリケーションの例

クライアント アプリケーションでは、以下のアクションが実行されます。

- WebLogic Server ヘルパー クラスを使用した、管理サーバの MBeanHome の取得
- MBeanHome を使用した、関連付けられた MBeanServer インタフェースの取得
- カスタム MBean の MBeanServer への登録
- カスタム MBean 属性の値の取得
- MBean の MBeanServer からの登録解除

MBeanHome および MBeanServer の取得など、上記のアクションの多くについては、この節で既に説明されています。登録および属性の呼び出しだけが MBean によって異なります。これらの呼び出しは MBeanServer インタフェースに直接作用し、完全に JMX に準拠しているからです。詳細な情報は、Java コメントに記載されています。

```
import weblogic.management.MBeanHome;
import weblogic.management.Helper;
import weblogic.management.RemoteMBeanServer;
import javax.management.*;
import MyCustomMBean;

// クライアント クラスで MyCustomMBean が実装され、main 関数によって
// MBeanHome と MBeanServer の取得、MBean の登録、属性値へのアクセス、
// およびそれ自体の登録解除が行われる

public class MyCustom implements MyCustomMBean, java.io.Serializable {

  public static void main(String[] args)
```

```
// 実際の JMX クライアントでは、アプリケーション内の別の場所で例外が適切に処理される
// わかりやすくするために、この例では例外の送出だけを行う
throws Exception {

  // ヘルパー クラスを使用して、管理サーバの MBeanHome を
  // 取得する
  MBeanHome mbh = Helper.getMBeanHome("system",
  "system_password","t3://localhost:7001","examplesserver");

  // MBeanHome を使用して、MBeanServer インタフェースを取得する
  RemoteMBeanServer mbs = mbh.getMBeanServer();

  // MBeanServer インタフェースに対して JMX 呼び出しを使用するには、
  // ObjectName を使用する必要がある
  ObjectName mbo = new ObjectName("user_Domain:Name=x");

  // カスタム MBean の MBeanServer への登録を試みる
  try {
    mbs.registerMBean((Object)new MyCustom(),mbo);
  } catch(InstanceAlreadyExistsException i) {
    System.out.println("MBean ("+mbo+") allready exists");
  }

  // MyAttribute の値を取得し、出力する
  System.out.println("Value of MyAttribute of ("+mbo+")from MBeanServer = "+
  mbs.getAttribute(mbo,"MyAttribute"));
```

```
    // カスタム MBean が登録解除される
      mbs.unregisterMBean(mbo);
  }

  // この例のクライアントでは、MyCustomMBean インタフェースの、
  // 出力を行うメソッドが 1 つ実装されている
  public int getMyAttribute() {
    System.out.println("getMyAttribute invoked.");
    return 999;
  }
}
```

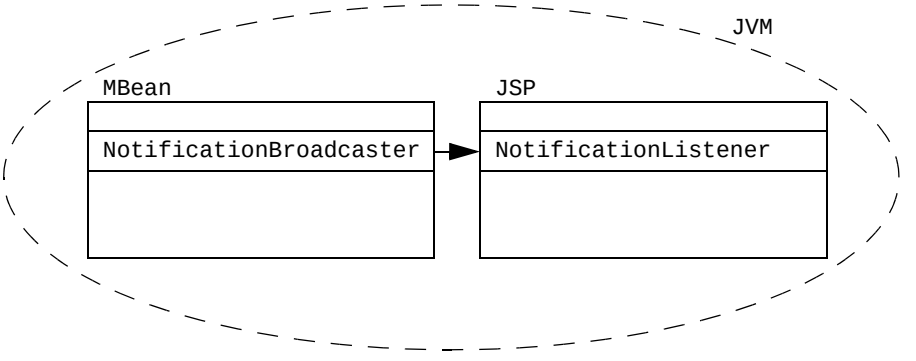
3 MBean 通知の使い方

以下の節では、WebLogic Server MBean からブロードキャストされる、さまざまな通知の使い方について概説します。

概要

すべての WebLogic Server MBean には `javax.management.NotificationBroadcaster` インタフェースが実装されているため、WebLogic Server MBean は標準の JMX 通知タイプを送信できます。

MBean 通知を監視するには、`NotificationListener` インタフェースをクライアント アプリケーションに実装し、受信する通知の送信元になる MBean にリスナ クラスを登録します。次の図は、JSP またはサーブレットを使用して通知をモニタする基本的なシステムを示しています。



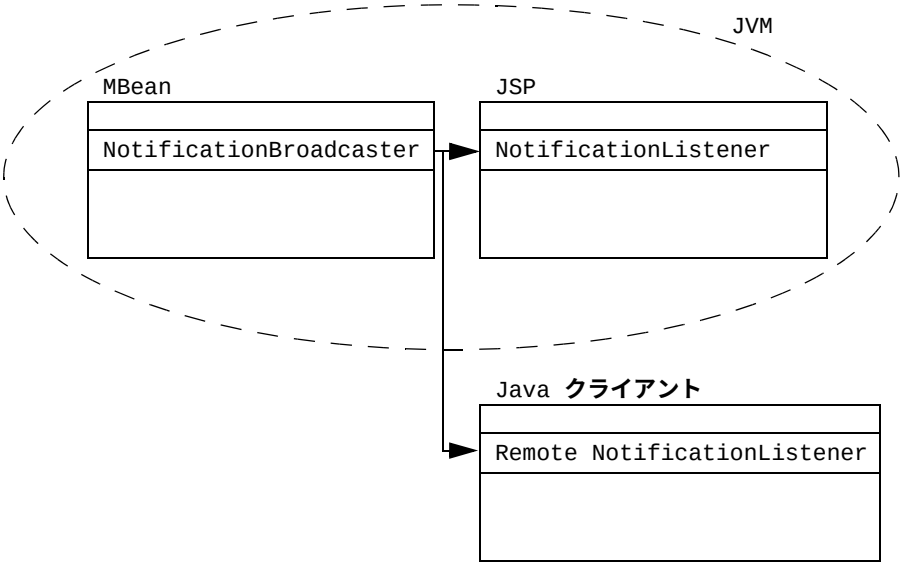
必要に応じて、リスナ クラスには NotificationFilter クラスを登録できます。これにより、リスナが受信する通知に関して詳細な指定を行うことができます。

注意： JMX 通知とその機能の詳細については、Sun Microsystems J2EE JMX 仕様を参照してください。

外部クライアントでの通知の使用

JMX 1.0 仕様では、ブロードキャストを行っている MBean の JVM の外部にあるクライアントで通知を使用できるようにする方法については定義されていません。WebLogic Server バージョン 6.1 では、weblogic.management.RemoteNotificationListener インタフェースを通じて、外部でも通知を使用できるようになっています。

RemoteNotificationListener は、javax.management.NotificationListener および java.rmi.Remote を拡張したものであり、RMI を通じて外部クライアントで MBean 通知を使用できるようにします。次の図に示すように、WebLogic MBean 通知を受信するために、リモートの Java クライアントには、NotificationListener ではなく、RemoteNotificationListener が実装されます。



リモートの Java クライアント リスナの登録は、標準の JMX `addNotificationListener()` メソッドを使用して行われます。

MBean 通知の概要

WebLogic Server 通知では、JMX 1.0 仕様で示されている、標準の通知クラスが使用されます。さらに、WebLogic Server では、WebLogic Server MBean ログ通知を処理するための、追加の通知クラスおよび通知ヘルパー クラスが用意されています。以降の節では、WebLogic Server の JMX アプリケーションで使用できる通知タイプおよび通知クラスについて概説します。

基本的な JMX 通知

すべての WebLogic Server MBean には、`NotificationBroadcaster` インタフェースが実装されているため、WebLogic Server MBean は JMX 1.0 仕様で示されている通知タイプを生成できます通知タイプは以下のとおりです。

- `javax.management.AttributeChangeNotification`。MBean の属性値が変更された場合の通知に使用します。
- `javax.management.MbeanServerNotification`。MBeanServer に委託された通知に使用します。

また、特定の WebLogic Server MBean では、「add」メソッドおよび「remove」メソッドを保持する属性用に通知タイプがさらに 2 種類サポートされています。

- `weblogic.management.AttributeAddNotification` は、属性の `add`*`AttributeName`* メソッドが呼び出されるときにブロードキャストされます。
- `weblogic.management.AttributeRemoveNotification` は、属性の `remove`*`AttributeName`* メソッドが呼び出されるときにブロードキャストされます。

WebLogic Server ログ通知

WebLogic Server には、ログ メッセージのブロードキャストを処理する `LogBroadcasterRuntime` MBean が用意されています。ログ通知をリスンする必要のあるクライアント アプリケーションでは、通知リスナを `LogBroadcasterRuntime` MBean に簡単に登録できます。

WebLogic Server ログ メッセージを表す通知には、以下のような多くの情報が含まれています。

- ログ メッセージを発行したマシンの名前
- ログ メッセージを発行した WebLogic Server の名前
- ログ メッセージの ID 番号

こうした WebLogic Server ログ情報を JMX アプリケーションで抽出し、使用できるよう、`WebLogicLogNotification` ラッパー クラスが用意されています。`WebLogicLogNotification` では、メッセージに関連付けられたトランザクション ID、ユーザ ID、およびバージョン番号を取得するメソッドだけでなく、ログ メッセージの部分を抽出する、単純なゲッター メソッドも提供されています。

「WebLogic Server ログ通知の処理」では、クラスおよびインタフェースがサポートされているログ通知の使い方の詳細について説明しています。

基本的な JMX 通知の使い方

外部クライアント アプリケーションで WebLogic MBean 通知を受信するには、以下の処理を行う必要があります。

1. `RemoteNotificationListener` インタフェースを実装します。
2. 受信する通知の送信元になる MBean へリスナ クラスを登録します。

以降の節では、これらの処理の基本手順について説明します。

通知リスナの作成

通知リスナ クラスは、1 つまたは複数の MBean によってブロードキャストされる JMX 通知を処理します。JMX アプリケーションが、ブロードキャストを行っている MBean の JVM の外部にある場合は、リスナ クラスに `weblogic.management.RemoteNotificationListener` を実装して、通知の受信時にアクションを実行する `handleNotification()` クラスを提供する必要があります。次に実装例を示します。

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;
...
public class WebLogicLogNotificationListener implements
        RemoteNotificationListener {
...
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");
    }
```

通知リスナの MBean への登録

すべての WebLogic Server MBean は通知をブロードキャストします。このため、使用可能な任意の MBean に `NotificationListener` を登録できます。`NotificationListener` は、その MBean の `addNotificationListener()` メソッドを呼び出すことによって登録できます。

ただし、ほとんどの場合では、MBeanServer の `addNotificationListener()` メソッドを使用して、リスナを登録することが望ましいと言えます。`javax.management.MBeanServer` インタフェースを使用することで、登録を行うためだけに特定の MBean をルックアップする手間を省くことができます。たとえば、「通知リスナの作成」で定義されたリスナは次のように登録されます。

```
rmbs = home.getMBeanServer();
oname = new WebLogicObjectName("TheLogBroadcaster",
        "LogBroadcasterRuntime",
        DOMAIN_NAME,
        SERVER_NAME);
rmbs.addNotificationListener(oname,
        listener,
        null,
        null);
```

WebLogic Server ログ通知の処理

上記の例で示されているように、ログ メッセージを受信するために、クライアント アプリケーションでは標準の JMX API を使用して、通知リスナを WebLogic Server `LogBroadcasterRuntimeMBean` に登録できます。`LogBroadcasterRuntimeMBean` は、サーバによって生成されるログ メッセージ用の通知の生成を行います。

LogBroadcasterRuntimeMBean によってブロードキャストされるすべての通知のタイプは、WebLogicLogNotification です。TheLogBroadcaster という LogBroadcasterRuntimeMBean が、サーバごとに 1 つだけ存在します。

LogBroadcasterRuntimeMBean には、「MBeanHome からの MBean へのアクセス」で説明されているメカニズムを使用してアクセスできます。

WebLogicLogNotification の内容

WebLogic Server ログ メッセージのすべての JMX 通知には、以下のフィールドがあります。

- タイプ ― ログ通知のマップ先のタイプ フィールド。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 WebLogic Server サブシステムを示し、*messageID* は WebLogic Server 内部のメッセージ ID を示します。
- 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 メッセージの実際の本文。
- 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。

すべての通知のタイプは、WebLogicLogNotification です。このヘルパー クラスでは、ログ メッセージの個々のフィールドすべてに対するゲッター メソッドが提供されています。WebLogicLogNotification クラスを使用すると、クライアント アプリケーションで重要度、ユーザ ID、サブシステムなどのフィールドに基づいて、ログ通知を簡単にフィルタ処理できます。

次に示す NotificationFilter の例では、WebLogicLogNotification クラスを使用して、特定のメッセージ ID（111000）のメッセージだけが選択され、通知として送信されます。

```
import javax.management.Notification;

import javax.management.NotificationFilter;
```

```
import javax.management.Notification.*;
....
public class WebLogicLogNotificationFilter implements NotificationFilter,
                                                    java.io.Serializable {
public WebLogicLogNotificationFilter() {
        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;
    }
}
```

WebLogic Server エラー メッセージ用の通知リスナの例

クライアント アプリケーションでは、ログ メッセージを通知として受信し、以下のようなアクションを実行する、さまざまなカスタム `NotificationListener` を作成できます。

- WebLogic 管理者に重要なログ メッセージを電子メールで送信する
- ログ メッセージをデータストアに追加する

この通知リスナの基本的なフォームは、「通知リスナの作成」で示されている例とは若干異なります。その例で出力されるメッセージを、通知に応答してアクションを実行する、必要な JDBC 呼び出しまたはページング処理で置き換えてください。

4 WebLogic Server MBean のモニタ

以下の節では、WebLogic Server MBean 属性をモニタする方法について概説します。

概要

WebLogic Server クライアントでは、モニタを設定して 1 つまたは複数の MBean 属性をモニタできます。JMX 仕様には、パッケージ `javax.management.monitor` のさまざまなタイプのモニタが定義されています。標準の JMX モニタは以下のとおりです。

- `CounterMonitor`。整数の属性を監視します。
- `GaugeMonitor`。整数または浮動小数点の属性を監視します。
- `StringMonitor`。文字列の属性を監視します。

JMX モニタは、多くの場合、特定の条件がモニタで満たされると、それを示すために、通知をブロードキャストします。このため、システムのモニタには通常、通知リスナや通知フィルタなど、標準の通知の構成概念が含まれており、それらはモニタに登録されています。

モニタの設定

ここでは、JMX 通知を受信するためのカウンタ モニタの設定方法の例を示します。この例でもモニタの通知を監視するのに通知リスナが使用されているので、情報の一部は「MBean 通知の使い方」にある例から作成されています。

通知リスナの作成

この例ではまず、CounterListener という通知リスナが作成されています。この通知リスナは JMX モニタから送信される通知を受信します。

```
import java.rmi.Remote;
import javax.management.Notification;
import javax.management.monitor.MonitorNotification;
import weblogic.management.RemoteNotificationListener;

public class CounterListener implements RemoteNotificationListener  {
    String message;
    public void handleNotification(Notification notification ,Object obj) {
        System.out.println("\njavax.management.Notification");
        System.out.println("            type = " +
                           notification.getType());
        System.out.println("  sequenceNumber = " +
                           notification.getSequenceNumber());
        System.out.println("          source = " +
                           notification.getSource());
        System.out.println("       timestamp = " +
                           notification.getTimeStamp() + "\n");
```

```
        if(notification instanceof MonitorNotification) {
            MonitorNotification monitorNotification =
                                 (MonitorNotification) notification;
           System.out.println("\njavax.management.monitor.MonitorNotification");
            System.out.println("  observed attr = " +
                               monitorNotification.getObservedAttribute() );
            System.out.println("   observed obj =" +
                               monitorNotification.getObservedObject() );
            System.out.println("  trigger value =" +
                               monitorNotification.getTrigger() + "\n");
            message = "Mbean: " + monitorNotification.getObservedAttribute() +
                      "\n" +
                      "Attribute: " + monitorNotification.getObservedObject() +
                      "\n" +
                      "Trigger Value: " + monitorNotification.getTrigger();
        }
    }
}
```

リスナおよびモニタのインスタンス化

次の例のモニタ クラスでは、リスナおよびモニタ オブジェクトの両方がインスタンス化され、`ServerSecurityRuntime.InvalidLoginAttemptsTotalCount` 属性を監視するモニタが登録されています。この属性は、サーバへのログインの失敗数を示します。

無効なログインの試行回数がしきい値を超えた場合、通知リスナ `CounterListener.handleNotification()` によって `handleNotification` メソッドが呼び出されます。

モニタ コードの例は次のとおりです。

```
import java.rmi.RemoteException;
import java.util.Set;
import java.util.Iterator;
import javax.management.*;
import javax.management.AttributeChangeNotification;
import javax.management.AttributeChangeNotificationFilter;
import javax.management.monitor.CounterMonitor;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.*;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.monitor.*;
import weblogic.management.runtime.ServerRuntimeMBean;

public class InvalidLoginMonitor {
    public static void main (String args[]) {
        // パスワードの引数があることを確認する
        if (args.length != 3) {
            System.out.println("Usage: java InvalidLoginMonitor " +
                               "<system password> " +
                               "<domain name> " +
                               "<server name>");
            return;
        }
        String url      = "t3://localhost:7001";
        String username = "system";
```

```
String password = args[0];
MBeanHome home  = null;
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got the MBeanHome " + home);
    RemoteMBeanServer rmbs = home.getMBeanServer();
    CounterMonitor monitor = new CounterMonitor();
    CounterListener listener = new CounterListener();
    WebLogicObjectName monitorObjectName =
           new WebLogicObjectName("MyCounter",
                                  "CounterMonitor",
                                  args[1],
                                  args[2]);
    WebLogicObjectName securityRtObjectName =
           new WebLogicObjectName("myserver",
                                  "ServerSecurityRuntime",
                                  args[1],
                                  args[2]);
    Long t = new Long(2);
    Long offset = new Long(0);
    monitor.setThreshold((Number)t);
    monitor.setNotify(true);
```

```
            monitor.setOffset((Number)offset);
            monitor.setObservedAttribute("InvalidLoginAttemptsTotalCount");
            monitor.setObservedObject(securityRtObjectName);
            monitor.addNotificationListener(listener, null, null);
            monitor.preRegister(rmbs, monitorObjectName);
            monitor.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

モニタ シナリオの例

この節では、パフォーマンスやリソース使用状況を監視するためにモニタする可能性のある、一部の典型的な MBean 属性について概説します。個々の MBean の属性またはメソッドに関する詳細については、適切な MBean の API ドキュメントを参照してください。

JDBC のモニタ

JDBCConnectionPoolRuntime MBean には、デプロイ済みの JDBC 接続プールへの接続状況を示すいくつかの属性があります。アプリケーションでは、これらの属性をモニタして、接続のリークだけでなく、接続の遅延および失敗を監視できます。次の表は、JDBC のモニタで通常使用される、これらの MBean 属性の概要を示しています。

JDBCConnectionPoolRuntime MBean の属性	アプリケーションによる一般的なモニタ
LeakedConnectionCount	リークされた接続の総数があらかじめ指定されたしきい値に達すると、リスナに通知する。リークされた接続は、チェックアウト済みの接続だが、close() 呼び出しを通じて接続プールに返されない。リークされた接続はその後の接続要求の遂行に使用できないので、その総数をモニタすることは重要である。
ActiveConnectionsCurrentCount	指定された JDBC 接続プールに対する、現在のアクティブな接続数があらかじめ指定されたしきい値に達すると、リスナに通知する。
ConnectionDelayTime	接続プールに接続する平均時間があらかじめ指定されたしきい値を超えると、リスナに通知する。
FailuresToReconnect	接続プールがそのデータストアへの再接続に失敗すると、リスナに通知する。アプリケーションでは、この属性がインクリメントしたり、しきい値に達したりすると、許容できる中断時間のレベルに応じてリスナに通知できる。