BEA WebLogic Server™

WebLogic JMX Service プログラマーズ ガイド

BEA WebLogic Server バージョン 7.0
改訂 : 2004 年 3 月 18 日

WebLogic JMX Service プログラマーズ ガイド

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	2004 年 3 月 18 日	BEA WebLogic Server バージョン 7.0

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

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、WebLogic Server のドメイン、クラスタ、およびサーバ インスタンスをコンフィグレーションおよびモニタする方法について説明します。

マニュアルの内容は以下のとおりです。

- 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、WebLogic Server MBean、MBean ホーム インタフェース、分散管理アーキテクチャについて概説します。
- 第 2 章「WebLogic Server MBean へのアクセス」では、WebLogic Server MBean のインタフェースにアクセスする方法について説明します。
- 第 3 章「コンフィグレーション情報のアクセスと変更」では、WebLogic Server リソースのコンフィグレーションを検索および変更する方法についての例を示します。
- 第 4 章「実行時の情報へのアクセス」では、WebLogic Server ドメインやサーバ インスタンスに関する実行時情報を検索および変更する方法についての例を示します。
- 第 5 章「WebLogic Server MBean 通知およびモニタの使い方」では、WebLogic Server の MBean 属性値の変化を観察し、それに応答する方法について説明します。

注意： WebLogic Security サービスでは、WebLogic Server のセキュリティを管理する MBean および MBean を新しく生成するためのツールを提供します。これらの MBean はセキュリティ MBean と呼ばれ、その利用モデルはこのマニュアルで説明するモデルとは異なります。セキュリティ MBean の詳細については、『WebLogic Security サービスの開発』を参照してください。

対象読者

このマニュアルは、BEA WebLogic Server の機能を使用するカスタム アプリケーションを作成して、アプリケーションやサーバ インスタンスをモニタおよびコンフィグレーションすること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。Java Management Extensions (JMX) に必ずしも精通している必要はありません。

このマニュアルでは WebLogic Server が提供する管理対象 Bean (MBean) へのアクセスおよび使用方法について説明しますが、ユーザ独自の MBean を作成する方法については説明しません。WebLogic Server が提供する MBean とは別の MBean を作成および使用する方法については、JMX 1.0 仕様 (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能) を参照してください。

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	プレースホルダを示す。 例: `String CustomerName;`

大文字の等幅テキスト	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]`

表記法	適用

|	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

WebLogic Server は、オープンで拡張可能な管理サービスを提供するために、Sun Microsystems, Inc. の Java Management Extensions (JMX) 1.0 仕様を実装しています。便利な独自のメソッド群とその他の拡張を追加することで、WebLogic Server 分散環境を活用しています。

すべての WebLogic Server リソースはこれらの JMX ベースのサービスによって管理され、WebLogic Server の内部で実行されるサードパーティのサービスとアプリケーションも同様に管理できます。これらの JMX サービスを使用して独自の管理ユーティリティを構築することで、WebLogic Server のリソースやアプリケーションを管理できます。

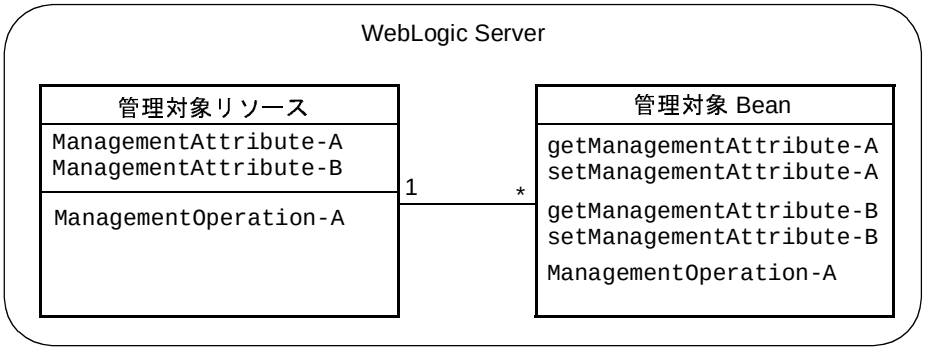
以下の節では、WebLogic Server の JMX サービスについて概説します。

JMX 1.0 仕様に目を通すには、http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からアーカイブをダウンロードします。ダウンロードしたアーカイブに、API のドキュメントが格納されています。

WebLogic Server の管理対象リソースと MBean

WebLogic Server 内部のサブシステム (JMS プロバイダや JDBC コンテナなど) およびそれらが管理する項目 (JMS サーバや JDBC 接続プールなど) を、**WebLogic Server 管理対象リソース**と呼びます。各管理対象リソースには、管理用にコンフィグレーションおよびモニタできる一連の属性が組み込まれています。たとえば、各 JDBC 接続プールには、その名前、ドライバ名、初期容量、およびキャッシュ サイズを定義する属性が組み込まれています。一部の管理対象リソースは、管理用に使用できるメソッド (操作) を備えています。WebLogic JMX サービスは、1 つまたは複数の管理対象 Bean (MBean) を通じてこれらの管理属性および操作を公開します。**MBean** は JMX 仕様に基づいて開発される具体的な Java クラスです。MBean は、管理対象リソースの各管理属性に対するゲッター操作とセッター操作、リソースごとに使用できる追加の管理操作を備えることができます。詳細については、図 1-1 を参照してください。

図 1-1 管理対象リソースと管理対象 Bean



管理対象リソースのコンフィグレーションのために属性および操作をエクスポーズする WebLogic Server MBean のことは**コンフィグレーション MBean** と呼び、管理対象リソースの実行時状態に関する情報を提供する MBean のことは**実行時 MBean** と呼びます。コンフィグレーション MBean と実行時 MBean とでは分散および管理の方法が異なるため、WebLogic Server ドメイン内のリソースをコンフィグレーションしたり、リソースの実行時状態に関するデータを表示したりする機能が異なります。

以下の節では、WebLogic Server における MBean の分散および管理の方法について説明します。

WebLogic Server ドメインの基本構成

WebLogic Server の管理**ドメイン**は、WebLogic Server リソースの論理的に関連したグループです。ドメインには、**管理サーバ**と呼ばれる特殊な WebLogic Server インスタンスが含まれます。管理サーバでは、ドメイン内のすべてのリソースを一元的にコンフィグレーションおよび管理します。通常は、**管理対象サーバ**と呼ばれる WebLogic Server インスタンスも含めてドメインをコンフィグレーションします。開発したアプリケーションや EJB などのリソースは管理対象サーバにデプロイし、管理サーバはコンフィグレーションや管理の目的にのみ使用します。

複数の管理対象サーバを使用すると重要なアプリケーションでロード バランシングとフェイルオーバを利用でき、1 つの管理サーバは管理対象サーバ インスタンスの管理を容易にします。ドメインの詳細については、『管理者ガイド』の「システム管理の概要」を参照してください。

管理対象リソースをコンフィグレーションするための MBean

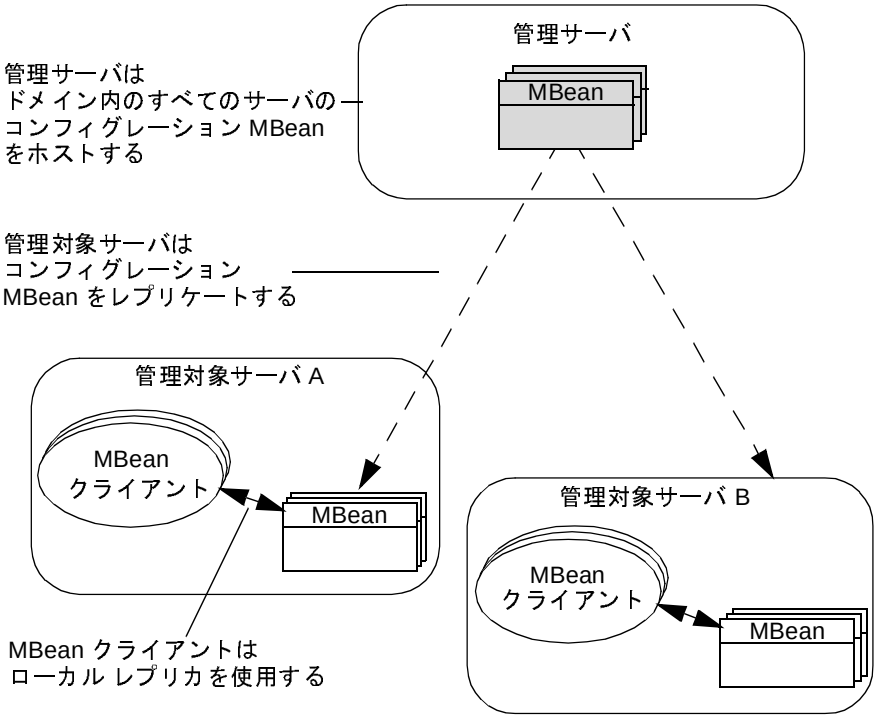
管理責任を管理サーバに集中させる WebLogic Server モデルをサポートするために、管理サーバはドメイン内のすべてのサーバ インスタンスのすべての管理対象リソースのコンフィグレーション MBean をホストします。さらに、管理サーバはサーバ インスタンスを停止して再起動するときに利用できるようにコンフィグレーション データの変更を保存します。

WebLogic Server リソースのコンフィグレーションを変更するには、管理サーバ上のコンフィグレーション MBean の値を変更します。

コンフィグレーション MBean のローカル レプリカ

パフォーマンスを向上させたり、クラスタ化機能をサポートするために、各管理対象サーバはドメイン内のすべてのコンフィグレーション MBean のローカル レプリカを作成します。MBean と対話する WebLogic Server サブシステムおよびアプリケーションは、管理サーバに対してリモート呼び出しを実行する代わりに、ローカル サーバ上のレプリカを使用します (図 1-2 を参照)。

図 1-2 MBean のレプリケーション



管理サーバ上のコンフィグレーション MBean は**管理 MBean** と呼ばれ、管理対象サーバ上のレプリカは**ローカル コンフィグレーション MBean** と呼ばれます。

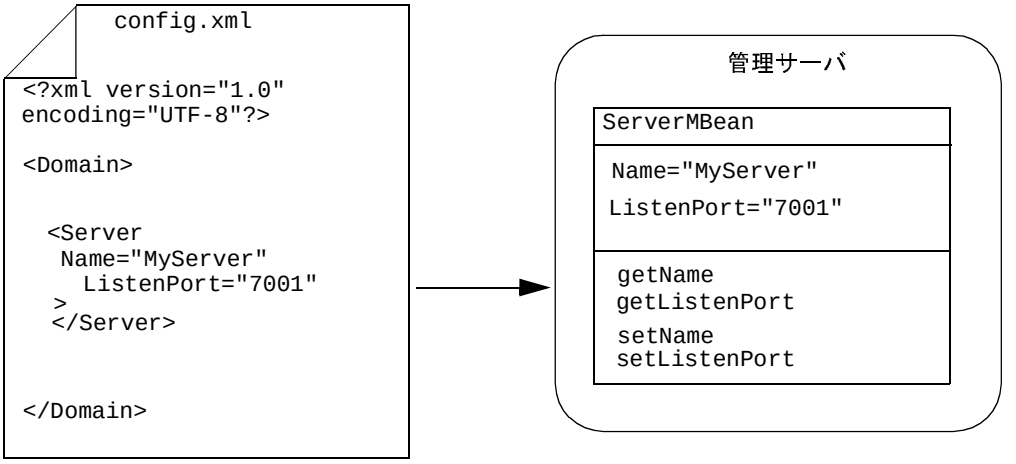
注意： 管理サーバは、管理 MBean だけでなく、ローカル コンフィグレーション MBean もホストします。ローカル コンフィグレーション MBean は、管理サーバのサブシステム、および管理サーバにデプロイされたアプリケーションによって使用されます。

コンフィグレーション MBean のライフサイクル

この節では、管理 MBean とローカル コンフィグレーション MBean がどのように初期化されるのか、コンフィグレーション データの変更がどのように WebLogic Server システム全体に伝播されるのか、および属性値をどのように変更すればサーバ インスタンスの再起動時に利用可能になるのかを説明します。

1. コンフィグレーション MBean のライフサイクルは、管理サーバを起動したときに開始されます。管理サーバは、その起動時にドメインの config.xml ファイルのデータを使用してドメインのすべての管理 MBean を初期化します (図 1-3 を参照)。

図 1-3 コンフィグレーション MBean の初期化

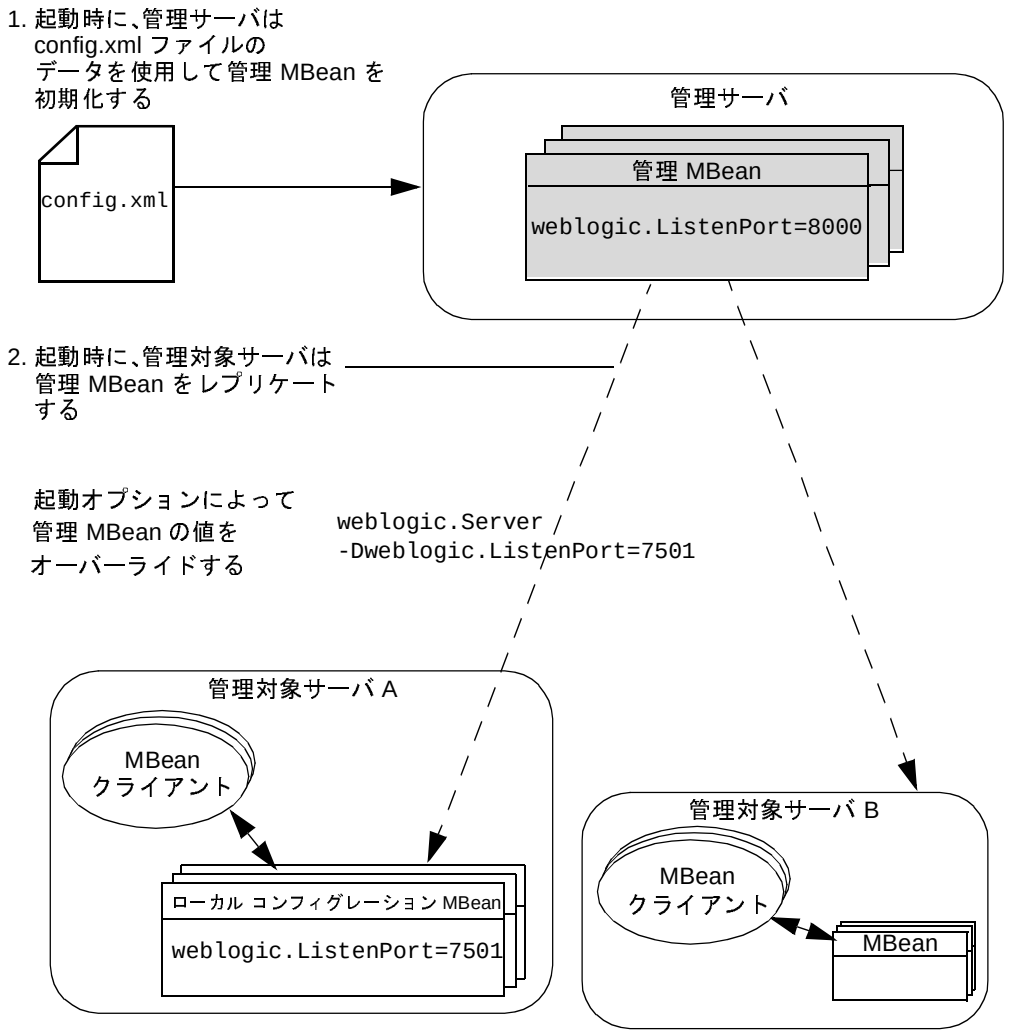


管理サーバは、その起動時にのみ config.xml ファイルからデータを読み出します。

2. 管理対象サーバは、起動時に管理サーバに接続して自身のコンフィグレーション データを取得します。デフォルトでは、管理対象サーバはドメイン内のリソースをコンフィグレーションする管理 MBean のレプリカを作成します。しかし、サーバの起動コマンドで引数を使用して、管理 MBean の値をオーバーライドできます。

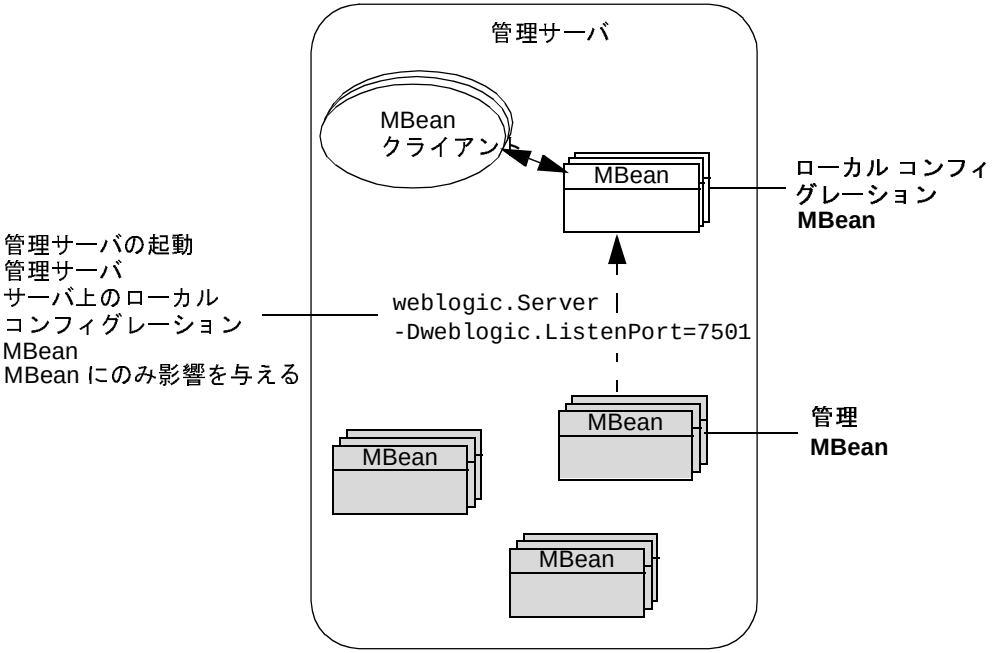
   たとえば、管理対象サーバ A について、config.xml ファイルにそのリスン ポートが 8000 であると指定されているとします。この場合、weblogic.Server コマンドを使用して管理対象サーバ A を起動したときに起動オプションとして -Dweblogic.ListenPort=7501 を指定すると、現在のサーバ セッションのリスン ポートが変更されます。管理対象サーバは管理 MBean のレプリカを作成しますが、リスン ポートの値として 7501 を代わりに使用します。管理対象サーバ A を再起動すると、このサーバは再び config.xml ファイルの値 8000 を使用します (図 1-4 を参照)。

図 1-4　管理 MBean の値のオーバーライド



管理サーバの起動時に `config.xml` の値をオーバーライドするために使用した起動コマンドの引数は、管理サーバ上のローカル コンフィグレーション MBean の値にのみ影響します。コマンド引数は管理 MBean の値には影響を与えず、したがって以降のサーバ セッションにも影響を与えません (図 1-5 を参照)。

図 1-5 管理サーバの値のオーバーライド



3. 管理 MBean の値を変更し、対応する管理対象サーバが稼働している場合、管理サーバはその変更をローカル コンフィグレーション MBean に伝播します。属性によっては、基になるリソースは再起動するまで新しい値を受け付けることができない場合があります。WebLogic Server Javadoc では、管理対象リソースが現在のセッションで属性の新しい値を受け付けることができるかどうかが示されています。管理対象リソースが新しい値を受け付けることができる場合でも、コンフィグレーションの変更をチェックする頻度によっては、そのリソースが更新値を即座に使用しない場合もあります。

   注意： 変更するのは管理 MBean の属性値のみとし、ローカル コンフィグレーション MBean の属性値は変更しないようにしてください。他の管理対象サーバのデータをレプリケートする際、管理対象サーバは管理 MBean に格納された値を使用します。管理 MBean とローカル コンフィグレーション MBean の値が異なると、通信上の問題が発生するおそれがあります。

4. 管理サーバは、管理 MBean が変更されたかどうかを定期的にチェックし、変更があった場合はそれらを `config.xml` に書き込みます。また、管理サーバの停止時や、WebLogic Server ユーティリティ (Administration Console や `weblogic.Admin` など) による MBean 属性の変更時にも、変更が `config.xml` ファイルに書き込まれます。
5. ローカル コンフィグレーション MBean は、管理対象サーバの停止時に破棄されます。管理 MBean は、管理サーバの停止時に破棄されます。

管理対象サーバ独立のための MBean のレプリケーション

管理対象サーバ独立 (MSI) は、管理サーバが利用できない場合に管理対象サーバを起動できる機能です。管理対象サーバで MSI がコンフィグレーションされている場合、その管理対象サーバには、そのローカル コンフィグレーション MBean に加え、ドメインのすべての管理 MBean のコピーが格納されます。

管理対象サーバ上の管理 MBean とは対話しないでください。これらはドメインの前回の正常なコンフィグレーションを反映しており、管理対象サーバを MSI モードで起動するためだけに使用します。管理対象サーバで管理 MBean を変更すると、管理対象サーバのコンフィグレーションが管理サーバと一致しなくなり、予測できない結果になることがあります。また、管理対象サーバは、他の管理対象サーバ上の管理 MBean を認識しません。

MSI の詳細については、『WebLogic Server ドメイン管理』の「管理サーバにアクセスできない場合の管理対象サーバの起動」を参照してください。

コンフィグレーション MBean API のドキュメント

コンフィグレーション MBean のドキュメントを参照するには、次の手順に従います。

1. WebLogic Server Javadoc を開きます。
2. Web ブラウザの左上のペインで、`weblogic.management.configuration` をクリックします。

   左下のペインに、パッケージのリンクが表示されます。
3. 左下のペインで、`weblogic.management.configuration` を再びクリックします。

右ペインに、パッケージの要約が表示されます (図 1-6 を参照)。

図 1-6 configuration パッケージの Javadoc

4. インタフェース名をクリックすると、その API ドキュメントが表示されます。

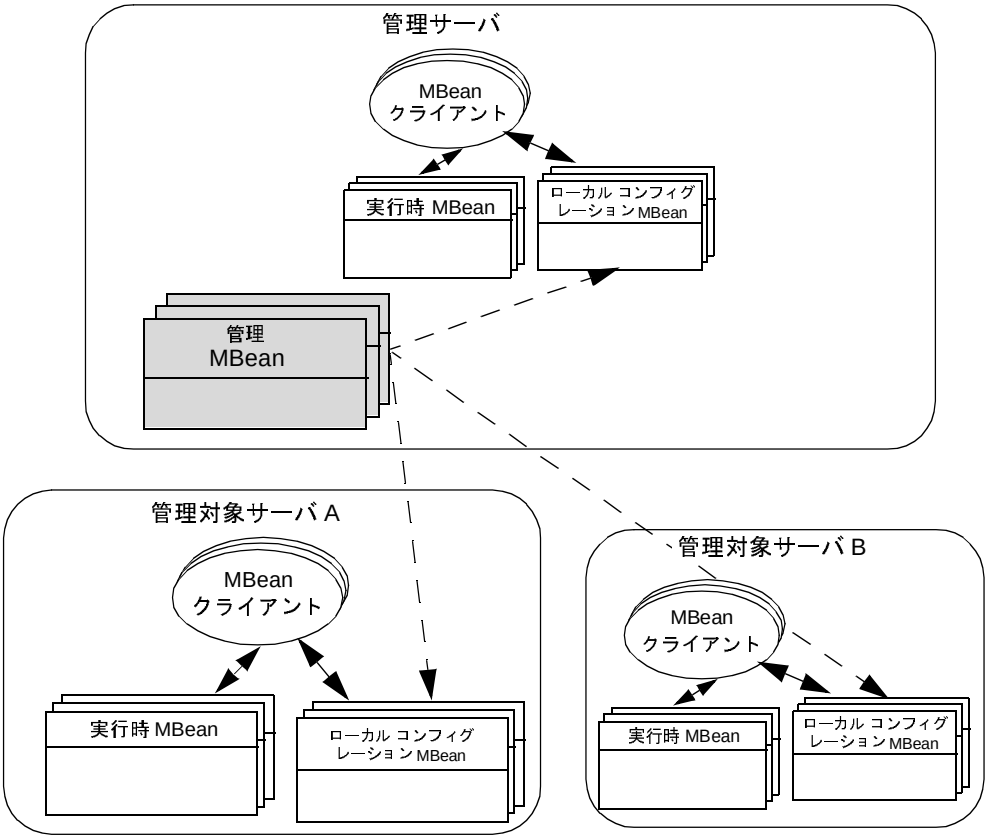
管理対象リソースの実行時の状態を参照するための MBean

WebLogic Server 管理対象リソースは、1 つまたは複数の実行時 MBean を通じてその実行時の状態に関するパフォーマンス メトリックなどの情報を提供します。実行時 MBean はコンフィグレーション MBean のようにレプリケートされず、基になる管理対象リソースと同じサーバ インスタンス上だけに存在します。

実行時 MBean は一時的なデータだけを保持し、データを `config.xml` ファイルに保存しません。サーバ インスタンスを停止すると、実行時 MBean から得られる実行時の統計とメトリックはすべて破棄されます。

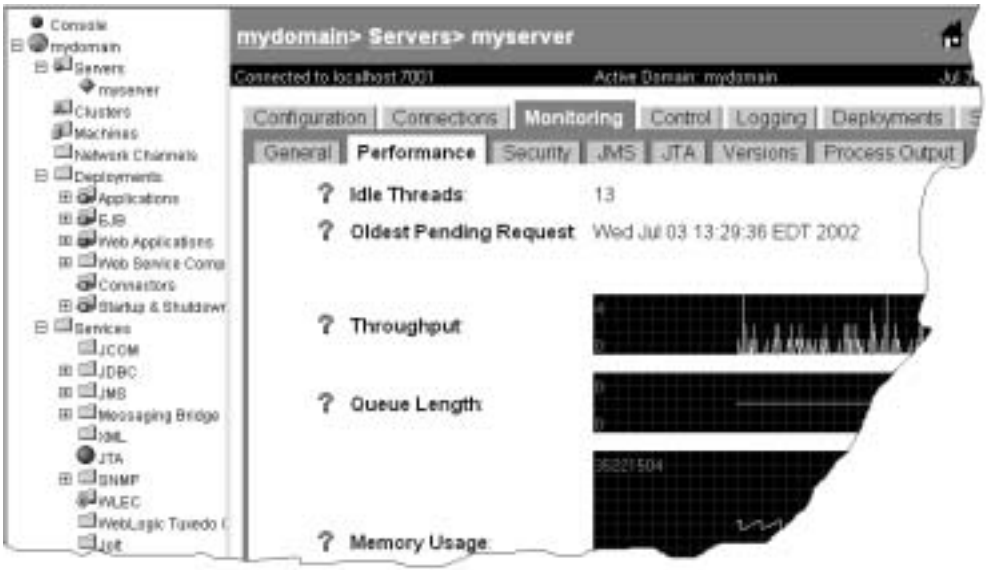
次の図に (図 1-7)、実行時 MBean、管理 MBean、およびローカル コンフィグレーション MBean がドメイン内でどのように配布されるかを示します。

図 1-7 MBean の配布



これらの値は、Administration Console、`weblogic.Admin` ユーティリティ、または MBean API を使用して参照できます (図 1-8 を参照)。

図 1-8 Administration Console からの実行時メトリックの参照



また、これらのインタフェースを使用すると、実行時の値を変更することもできます。たとえば、`weblogic.management.runtime.DeployerRuntimeMBean` を使用すると、デプロイされているモジュールの実行時の状態を変更することによってそのモジュールをアクティブ化および非アクティブ化できます。

実行時 MBean API のドキュメント

実行時 MBean のドキュメントを参照するには、次の手順に従います。

1. WebLogic Server Javadoc を開きます。
2. Web ブラウザの左上のペインで、`weblogic.management.runtime` をクリックします。

   左下のペインに、パッケージのリンクが表示されます。
3. 左下のペインで、`weblogic.management.runtime` を再びクリックします。

   右ペインに、パッケージの要約が表示されます (図 1-9 を参照)。

図 1-9　runtime パッケージの Javadoc

4. インタフェース名をクリックすると、その API ドキュメントが表示されます。

セキュリティ MBean

WebLogic Security サービスでは、WebLogic Server のセキュリティを管理する MBean および MBean を新しく生成するためのツールを提供します。これらの MBean はセキュリティ MBean と呼ばれ、その利用モデルはこのマニュアルで説明するモデルとは異なります。セキュリティ MBean の詳細については、『WebLogic Security サービスの開発』を参照してください。

WebLogic Server 以外の MBean

WebLogic Server には数百もの MBean が用意されており、それらの多くは、EJB、Web アプリケーション、およびその他のデプロイ可能な J2EE モジュールのコンフィグレーションとモニタに使用されます。アプリケーションまたはサービスをコンフィグレーションするために追加の MBean を使用する場合、独自の MBean を作成することができます。

作成する MBean は、JMX 仕様 (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能) によって定義されているすべての JMX 1.0 機能を活用できます。

ただし、WebLogic Server の JMX 拡張を使用できるのは、WebLogic Server に用意されている MBean だけです。たとえば、アプリケーション用に作成した独自の MBean は、そのデータを `config.xml` ファイルに保存できず、次節 (「MBean サーバ および MBeanHome インタフェース」) で説明する型保障インタフェースを使用できません。

MBean サーバ および MBeanHome インタフェース

WebLogic Server インスタンスの内部では、MBean を登録し、MBean へのアクセスを提供する実際の処理は MBean サーバ サブシステムに委託されます。管理対象サーバ上の MBean サーバは、現在の管理対象サーバ上のローカル コンフィグレーション MBean と実行時 MBean だけを登録し、それらへのアクセス提供します。管理サーバ上の MBean サーバは、ドメインの管理 MBean に加え、管理サーバ上のローカル コンフィグレーション MBean と実行時 MBean を登録し、それらへのアクセスを提供します。

注意 : 管理対象サーバの独立 (MSI) がコンフィグレーションされている管理対象サーバでは、MBean サーバ は、管理サーバが利用できない場合にサーバが起動に使用する管理 MBean のレプリカも登録します。それらの管理 MBean のレプリカとは対話しないでください。詳細については、1-9 ページの「管理対象サーバ独立のための MBean のレプリケーション」を参照してください。

MBean サーバ サブシステムにアクセスするには、`weblogic.management.MBeanHome` インタフェースを使用します。`MBeanHome` からは、以下のインタフェースを使用して MBean サーバとその MBean と対話できます (図 1-10 を参照)。

- `javax.management.MBeanServer`。MBean と対話するための標準 JMX インタフェース。このインタフェースを使用すると、MBean サーバに登録されている MBean をルックアップし、MBean で使用できる操作セットを判別し、各操作が返すデータ型を調べることができます。`MBeanServer` インタフェースを介して MBean 操作を呼び出す場合は、標準 JMX メソッドを使用する必要があります。次に例を示します。
  - `MBeanHome.getMBeanServer().getAttribute(`*`MBeanObjectName`*`, `*`attributeName`*`)`
  - `MBeanHome.getMBeanServer().setAttribute(`*`MBeanObjectName`*`, `*`attributeName`*`)`
  - `MBeanHome.getMBeanServer().invoke(`*`MBeanObjectName`*`, `*`operationName`*`, `*`params`*`, `*`signature`*`)`

  `MBeanServer` API の詳細なリストについては、http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロードできる JMX 1.0 API のドキュメントを参照してください。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

  `MBeanServer` インタフェースは、ユーザが作成および登録した MBean (WebLogic 以外の MBean) と対話する唯一の方法です。

- `javax.management.MBeanServer` インタフェースおよび `java.rmi.Remote` インタフェースを拡張した `weblogic.management.RemoteMBeanServer` インタフェース。リモート JVM から WebLogic Server MBean へのアクセスに標準 JMX 技術を使用する場合や、リモート JVM から WebLogic 以外の MBean と対話する場合は、`RemoteMBeanServer` インタフェースを使用します。

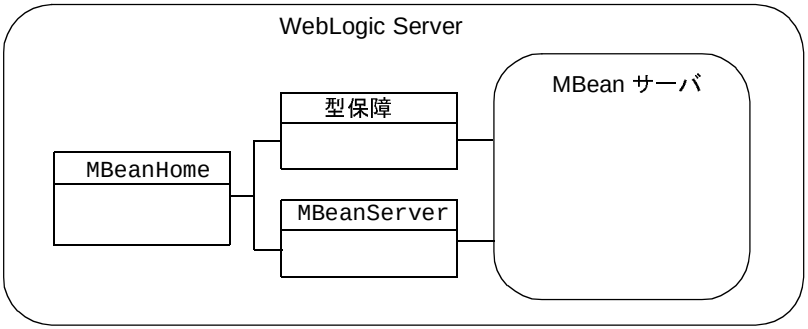
- MBean のメソッドをあたかも直接呼び出せるかのように見せる WebLogic Server 型保証インタフェース。このインタフェースを使用すると、MBean サーバに登録されている MBean をルックアップし、その MBean の取得、設定などの操作を呼び出すことができます。次に例を示します。

  ```
  wlMBean = MBeanHome.getMBean(WebLogicObjectName)
  wlMBean.get Attribute
  ```

`wlMBean.set`*`Attribute`*
`wlMBean.`*`operationName`*

型保証インタフェースは、`java.rmi.Remote` インタフェースを拡張したものですので、リモート JVM からの WebLogic Server MBean へのアクセスに使用できます。

図 1-10 MBean サーバとそれらのインタフェース



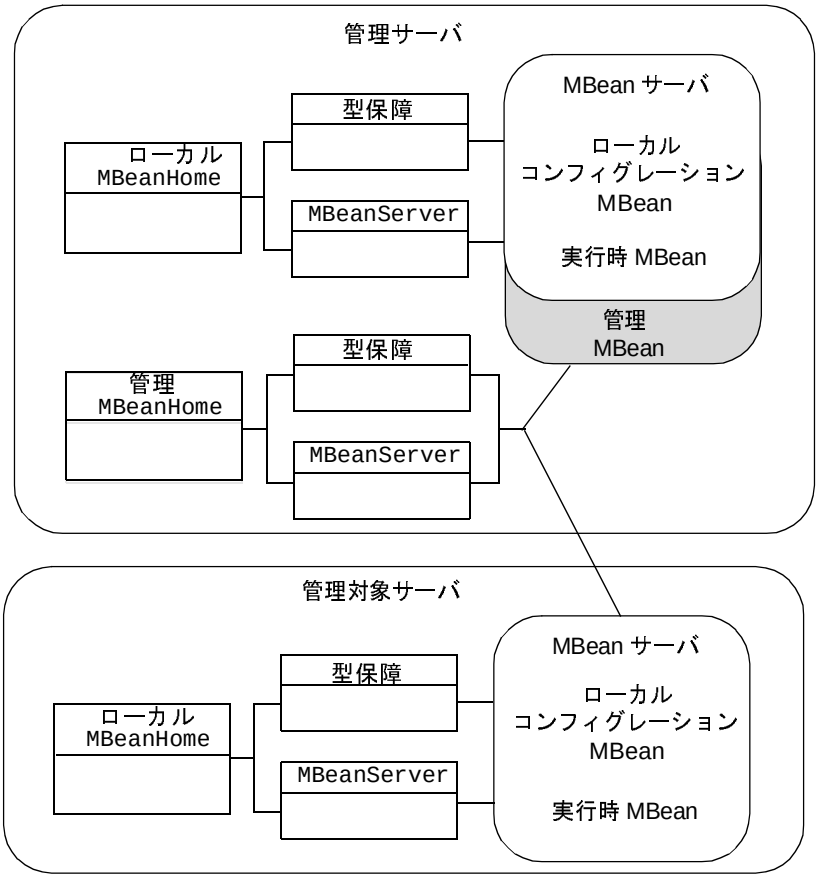
ローカル MBeanHome および管理 MBeanHome

WebLogic Server のすべてのインスタンスは、その MBean サーバにホストされている MBean にアクセスするための**ローカル `MBeanHome`** インタフェースを提供します。

管理対象サーバおよび管理サーバにおいて、ローカル `MBeanHome` インタフェースは現在のサーバについてのみ実行時 MBean へのアクセスを実現するとともに、ドメイン内のすべてのローカル コンフィグレーション MBean へのアクセスも実現します。

管理サーバは、`MBeanHome` インタフェースのもう 1 つのインスタンスを提供します。この**管理 `MBeanHome`** は、管理 MBean に加え、ドメイン内のすべてのサーバ インスタンス上の MBean へのアクセスを提供します。管理 `MBeanHome` は、RMI を使用して管理対象サーバ上の MBean と対話します。このため、ネットワーク リソースの使用量が増加し、ローカル `MBeanServer` または `MBeanHome` インタフェースを使用するときより時間がかかる場合があります (図 1-11 を参照)。

図 1-11　ローカルおよび管理 MBeanHome インタフェース



ローカル MBeanHome と管理 MBeanHome は同じインタフェース クラスの 2 つのインスタンスであるため、この 2 種類の MBeanHome の API は MBeanHome インスタンスの名前とアクセス可能な MBean のセットが異なるだけです。

通知とモニタ

管理ニーズに応じて、MBean API を使用して要求時のみに MBean 属性を参照することも、WebLogic Server 通知およびモニタ機能を使用して MBean 属性の変更時にレポート (JMX 通知) を自動的にブロードキャストすることもできます。

これらの機能を使用するには、次の手順に従います。

- JMX リスナを作成します。JMX リスナは、指定した MBean のすべての属性変更をリスンおよびレポートします。たとえば、リスナを別個のロジックと一緒に使用して、デプロイされているコンポーネントのコンフィグレーションをユーザが変更したときにシステム管理者に電子メールが送信されるようにできます。リスナの使い方については、第 5 章「WebLogic Server MBean 通知およびモニタの使い方」を参照してください。
- JMX モニタを作成します。JMX モニタは、設定したパラメータ セットから外れた MBean 属性の変更だけをリスンおよびレポートします。たとえば、モニタを別個のロジックと一緒に使用して、空いているスレッド プール数が所定の上限を超えたときにシステム管理者に電子メールが送信されるようにできます。 詳細については、第 5 章「WebLogic Server MBean 通知およびモニタの使い方」を参照してください。

Administration Console と weblogic.Admin ユーティリティ

WebLogic Server Administration Console と `weblogic.Admin` ユーティリティは、WebLogic Server JMX サービスを使用した管理ユーティリティの例です。JMX アプリケーションを開発する前に WebLogic Server 管理サービスに精通するため、これらのインタフェースを使用できます。

Administration Console

Administration Console は、WebLogic Server JMX API を呼び出すサーブレットを備えた Web アプリケーションです。Administration Console に表示されるほとんどの値は、管理 MBean と実行時 MBean の属性です。Administration Console はローカル コンフィグレーション MBean を読み書きしないので、サーバ インスタンスで現在使用されていない値をレポートする可能性があります。たとえば、`weblogic.Server` 起動オプションを使用してコンフィグレーション済みリスンポートをオーバーライドした場合、Administration Console はオーバーライドした値ではなく `config.xml` ファイル中の値をレポートします。

Administration Console がどの MBean 属性をレポートしているかを確認するには、最上部のバナーにある疑問符記号のアイコンをクリックします。Administration Console のフィールドに関連付けられた MBean クラスおよび属性を表示するには、ヘルプ ウィンドウの [属性] リンクをクリックします。

Administration Console のフィールドの横に表示される注意アイコン (黄色い三角形に感嘆符のアイコン) は、属性が動的ではないことを示します。このような属性を変更した場合、基になる管理対象リソースはサーバを再起動するまで新しい値を使用できません。

Administration Console からの動的な値を変更すると、それに対応する管理 MBean が更新されます。この変更がどのようにローカル コンフィグレーション MBean に伝播するかについては、1-5 ページの「コンフィグレーション MBean のライフサイクル」を参照してください。

weblogic.Admin ユーティリティ

`weblogic.Admin` ユーティリティには、管理 MBean とコンフィグレーション MBean の作成、値の取得と設定、操作の呼び出し、およびインスタンスの削除を行うコマンドが用意されています。また、実行時 MBean の値の取得と操作の呼び出しを行うコマンドも用意されています。WebLogic Server 管理サービスとプログラム的に対話する JMX アプリケーションを作成する代わりに、このユーティリティを使用するシェル スクリプトを作成することもできますが、JMX アプリケーションのパフォーマンスの方が、コマンドライン ユーティリティを呼び出すシェル スクリプトより優れています。

weblogic.Admin ユーティリティでは、JMX コードを記述する前に、MBean のオブジェクト名を確認したり、コマンドラインから属性を取得および設定したりすることもできます。このマニュアルの以降の章では、JMX 開発の過程で weblogic.Admin ユーティリティを使用する例を紹介します。

詳細については、『管理者ガイド』の「MBean 管理コマンド リファレンス」を参照してください。

2 WebLogic Server MBean へのアクセス

すべての JMX タスク (MBean 属性の表示と変更、通知の使用、および変更のモニタ) では、同じプロセスで MBean にアクセスします。

以下の節では、WebLogic Server MBean にアクセスする方法について説明します。

MBean へのアクセス : 主な手順

WebLogic Server で MBean にアクセスするための主な手順は次のとおりです。

1. `weblogic.management.MBeanHome` インタフェースを使用して MBean サーバにアクセスします。2-4 ページの「MBeanHome インタフェースへのアクセス」を参照してください。
2. 以下のいずれかのインタフェースを使用して、MBean の操作の取得、ルックアップ、および呼び出しを行います。

- WebLogic Server に付属の型保障インタフェース。このインタフェース (JMX に対する WebLogic Server の拡張) では、WebLogic Server で提供される MBean でのみ操作の取得と呼び出しを行えます。2-10 ページの「型保障インタフェースを使用した MBean へのアクセス」を参照してください。
- 標準の JMX `javax.management.MBeanServer` インタフェース。このインタフェースでは、WebLogic Server MBean または独自に作成した MBean で操作の取得および呼び出しを行えます。2-18 ページの「MBeanServer インタフェースを使用した MBean へのアクセス」を参照してください。
- `javax.management.MBeanServer` インタフェースおよび java.rmi.Remote インタフェースを拡張した weblogic.management.RemoteMBeanServer インタフェース。

ほとんどの場合では、これらのインタフェースを使用して MBean のリストを取得してから、そのリストをフィルタ処理して特定の MBean で操作の取得と呼び出しを行います。しかし、MBean の `WebLogicObjectName` が分かっている場合は、名前によって直接 MBean を取得できます。

使用するインタフェースの決定

MBean にアクセスするときには、どのインタフェースを使用するかに関して 2 つの選択を行う必要があります。

- MBean サーバへのアクセスに、ローカル サーバ インスタンス上の `MBeanHome` インタフェースまたは管理 `MBeanHome` インタフェースのどちらを使用するか。選択した `MBeanHome` インタフェースによって、アクセス可能な MBean が決まります。

  次の表に、ローカルの `MBeanHome` インタフェースと管理 `MBeanHome` インタフェースのどちらを使用するかを決定するための考慮事項を示します。

表 2-1 ローカル MBeanHome と管理 MBeanHome のどちらを使用するか

アプリケーションで管理する要素	取得する MBeanHome インタフェース
ローカル コンフィグレーション MBean または実行時 MBean	管理 MBeanHome またはローカル MBeanHome 管理 MBeanHome は、ドメイン内のすべてのサーバのすべての MBean にアクセスするのに便利な単一のインタフェースを提供する。このインタフェースは、複数のサーバ インスタンスから MBean を取得し、そのリストから特定のサーバ インスタンスの MBean を繰り返し検索する場合などに使用する。 ローカル MBeanHome は、現在のサーバについてのみ実行時 MBean へのアクセスを実現するとともに、ドメイン内のすべてのローカル コンフィグレーション MBean へのアクセスも実現する。このインタフェースでは、クライアントがサーバ インスタンスへの接続を直接確立する必要があるため、MBean へのアクセスに使用するネットワーク ホップの数も少なくて済む。 ローカル MBeanHome を使用する際には通常、最上位 MBean の 1 つを取得し、それを使用して MBean 階層を辿る。2-15 ページの「ローカルのコンフィグレーション MBean と実行時 MBean の階層を辿る」を参照。
管理 MBean	管理 MBeanHome

- MBean の操作のアクセスおよび呼び出しに、WebLogic Server 型保障インタフェース、標準の JMX MBeanServer インタフェース、または WebLogic RemoteMBeanServer インタフェースのうちどれを使用するか。

  次の表に、型保障インタフェースと MBeanServer インタフェースのどちらを使用するかを決定するための考慮事項を示します。

表 2-2 型保障インタフェースと MBeanServer インタフェースのどちらを使用するか

アプリケーションの動作	使用するインタフェース
WebLogic Server MBean のみと対話する	WebLogic Server 型保障インタフェース
WebLogic Server 以外の J2EE プラットフォームで動作しなければならない場合がある	`MBeanServer` 別の JVM で動作している MBean にクライアントからアクセスするには `RemoteMBeanServer` を使用する。クライアント コードは他の J2EE サーバでも使用できる。ただし、`RemoteMBeanServer` の代わりに、標準の `MBeanServer` インタフェースを拡張した他のインタフェースを使用する必要がある。
WebLogic Server MBean 以外の MBean と対話する	`MBeanServer` 別の JVM で動作している MBean にクライアントからアクセスするには `RemoteMBeanServer` を使用する。

MBeanHome インタフェースへのアクセス

ローカル `MBeanHome` インタフェースまたは管理 `MBeanHome` インタフェースを取得する最も単純な方法は、WebLogic Server の `Helper` クラスを使用することです。標準的な J2EE の手法の方がやりやすい場合は、Java Naming and Directory Interface (JNDI) を使用して `MBeanHome` を取得できます。

ヘルパー API を使用した MBeanHome インタフェースの取得

WebLogic Server には、MBeanHome インタフェースの取得プロセスを簡素化する weblogic.management.Helper クラスが用意されています。

Helper API を使用するには、以下の情報を収集します。

- MBean の操作を呼び出すパーミッションを持つ WebLogic Server ユーザのユーザ名とパスワード。詳細については、『管理者ガイド』の「システム管理操作の保護」を参照してください。
- ローカルの MBeanHome インタフェースにアクセスする場合、対象サーバの名前 (ドメイン コンフィグレーションに定義されている名前) と URL
- 管理 MBeanHome にアクセスする場合、管理サーバの URL

上の情報を収集したら、以下のいずれかの API を使用します。

- ローカルの MBeanHome を取得する場合
  ```
  Helper.getMBeanHome(java.lang.String user, java.lang.String
  password, java.lang.String serverURL, java.lang.String
  serverName)
  ```
- 管理 MBeanHome を取得する場合
  ```
  Helper.getAdminMBeanHome(java.lang.String user,
  java.lang.String password, java.lang.String adminServerURL)
  ```

Helper API の詳細については、WebLogic Server Javadoc を参照してください。

例：ローカル MBeanHome インタフェースの取得

次の例 (コード リスト 2-1) は、Helper API を使用して MS1 というサーバのローカル MBeanHome インタフェースを取得するクラスです。

コード リスト 2-1 ローカル MBeanHome インタフェースの取得

```
import weblogic.management.Helper;
import weblogic.management.MBeanHome;
```

```
public class UseHelper {
    public static void main(String[] args) {
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        String msName = "MS1";
        MBeanHome localHome = null;

        try {
            localHome = (MBeanHome)Helper.getMBeanHome(username, password, url,
                        msName);
            System.out.println("Local MBeanHome for" + localHome +
                        " found using the Helper class");
        } catch (IllegalArgumentException iae) {
            System.out.println("Illegal Argument Exception: " + iae);
        }
    }
}
```

JNDI を使用した MBeanHome インタフェースの取得

Helper API では MBeanHome インタフェースを取得する簡単な方法が提供されますが、JNDI を使用して MBeanHome を取得する標準的なアプローチの方がもっとわかりやすいかもしれません。管理対象サーバの JNDI ツリーから、サーバのローカル MBeanHome インタフェースにアクセスできます。管理サーバの JNDI ツリーからは、ドメイン内のすべてのサーバ インスタンスのローカル MBeanHome インタフェースに加えて管理 MBeanHome にもアクセスできます。

JNDI を使用して MBeanHome インタフェースを取得するには、次の手順に従います。

1. weblogic.jndi.Environment オブジェクトを構築し、Environment メソッドを使用してオブジェクトをコンフィグレーションします。

   a. setSecurityPrincipal メソッドと setSecurityCredentials メソッドを使用してユーザ資格を指定します。

      指定したユーザ資格に、MBeanHome インタフェースを使用して要求を実行するパーミッションが付与されているかどうかが検証されます。詳細に

ついては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

b. アプリケーションと MBeanHome インタフェースが異なる JVM で実行されている場合は、Environment.setProviderUrl メソッドを使用して MBeanHome インタフェースをホストするサーバ インスタンスを指定します。URL には、サーバのリスン アドレスと、サーバが管理要求をリスンするポートを指定する必要があります。

管理 MBeanHome を取得する場合は、setProviderUrl に管理サーバを指定する必要があります。

c. getInitialContext メソッドを使用して javax.naming.Context オブジェクトを初期化します。

たとえば、次のコードでは、WLServerHost というホスト コンピュータで動作するサーバ インスタンスへの初期コンテキストを設定し、ドメイン全体のデフォルトの管理ポートを使用して管理要求を受信しています。

```
Environment env = new Environment();
    env.setProviderUrl("t3://WLServerHost:9002");
    env.setSecurityPrincipal("weblogic");
    env.setSecurityCredentials("weblogic");
    Context ctx = env.getInitialContext();
```

weblogic.jndi.Environment の詳細については、WebLogic Server Javadoc を参照してください。

2. javax.naming.Context メソッドを使用して、現在のコンテキストの MBeanHome インタフェースをルックアップして取得します。

ローカル MBeanHome インタフェースと管理 MBeanHome のどちらを取得するかに応じて、以下の API のいずれかを使用します。

- 現在のコンテキストのローカル MBeanHome を取得するには次の API を使用する。
  javax.naming.Context.lookup(MBeanHome.LOCAL_JNDI_NAME)
- 現在のコンテキストが管理サーバの場合は、次の API を使用してドメイン内の任意のサーバ インスタンスのローカル MBeanHome を取得する。
  javax.naming.Context.lookup("weblogic.management.home.*relevantServerName*")

  relevantServerName は、ドメイン コンフィグレーションに定義されているサーバの名前です。

- 現在のコンテキストが管理サーバの場合は、次の API を使用して管理 MBeanHome を取得する。
  `javax.naming.Context.lookup(MBeanHome.ADMIN_JNDI_NAME)`

  管理 MBeanHome インタフェースは、ドメイン内のすべてのローカル コンフィグレーション MBean、管理 MBean、および実行時 MBean へのアクセスを提供します。

  `javax.naming.Context.lookup(String name)` の詳細については、Sun Javadoc を参照してください。

以降の節では、MBeanHome インタフェースを取得する例を示します。

- 例 : 外部クライアントからの管理 MBeanHome の取得
- 例 : 内部クライアントからのローカル MBeanHome の取得

例：外部クライアントからの管理 MBeanHome の取得

次の例 (コード リスト 2-2) は、異なる JVM で実行されているアプリケーションから管理 MBeanHome インタフェースをルックアップする方法を示したものです。この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 2-2 外部クライアントからの管理 MBeanHome の取得

```
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 class RetrieveMBeanHome{

    public static void main(String[] args) {
        MBeanHome home = null;
        // ドメイン変数
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
```

```
        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // 管理 MBeanHome インタフェースを取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home + " from the
                               Admin server");

        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }
    }
}
```

例：内部クライアントからのローカル MBeanHome の取得

管理サーバ (または管理する 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単です。コード リスト 2-3 は、WebLogic Server インスタンスと同じ JVM で動作している JMX アプリケーションが、t3://localhost:7001 でリスンするサーバ インスタンスのローカル MBeanHome をルックアップする方法を示しています。

コード リスト 2-3 内部クライアントからのローカル MBeanHome の取得

```
import javax.naming.Context;
import javax.management.ObjectName;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.jndi.Environment;

public class serverInfo {
    public static void main(String[] args) {

        MBeanHome home = null;
        // ドメイン変数
        String url = "t3://localhost:7001";
```

```
String username = "weblogic";
String password = "weblogic";

try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);

    // 初期コンテキストを設定
    Context ctx = env.getInitialContext();

     // サーバ固有の MBeanHome インタフェースを取得
    home = (MBeanHome)ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
    System.out.println("Got the Server-specific MBeanHome: " + home);
```

型保障インタフェースを使用した MBean へのアクセス

いったん MBeanHome インタフェースを取得したら、MBean の型保障インタフェースを取得する MBeanHome インタフェース内のメソッドを使用して MBean にアクセスするのがもっとも簡単です。

この型保障インタフェースは、WebLogic Server が提供する MBean でのみ使用できます。独自に作成した MBean タイプに基づく MBean には使用できません。

全 MBean のリストの取得

MBeanHome.getAllMBeans メソッドを使用すると、取得する MBeanHome インタフェースのスコープに含まれる MBean のオブジェクト名をルックアップできます。たとえば、管理 MBeanHome を取得する場合、getAllMBeans() を使用するとドメイン内のすべての MBean のリストが返されます。ローカル MBeanHome インタフェースを取得して getAllMBeans() を使用すると、現在のサーバのみの実行時 MBean とドメイン内のすべてのローカル コンフィグレーション MBean のリストが返されます。

コード リスト 2-4 のクラス例 :

1. JNDI API を使用して、管理 MBeanHome インタフェースを取得します。
2. MBeanHome.getAllMBeans メソッドを使用して、ドメイン内のすべての MBean を取得します。
3. MBean のリストを Set オブジェクトに割り当て、Set インタフェースおよび Iterator インタフェースを使用してリスト内を検索します。
4. WebLogicMBean.getObjectName メソッドを使用して、各 MBean の WebLogicObjectName を取得します。
5. WebLogicObjectName.getName メソッドおよび getType メソッドを使用して、WebLogicObjectName の Name および Type の値を取得します。

この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 2-4 ドメイン内のすべての MBean の取得

```
import javax.naming.Context;
import java.util.Set;
import java.util.Iterator;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;

public class ListAllMBeans{
    public static void main(String args[]) {
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        try {
            //JNDI を使用して MBeanHome を取得
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            MBeanHome home = (MBeanHome)ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
```

```
            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());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

MBeanHome.getAllMBeans メソッドの詳細については、WebLogic Server Javadoc を参照してください。

タイプおよびリストからの選択による MBean の取得

MBeanHome のスコープに含まれるすべての MBean のリストを取得する代わりに、特定のタイプと一致する MBean のリストを取得できます。Type は、MBean が管理するリソースのタイプと、MBean が管理 MBean であるか、ローカル コンフィグレーション MBean であるか、または実行時 MBean であるかを示します。MBean のタイプの詳細については、2-21 ページの「WebLogic Server MBean の WebLogicObjectName の使用」を参照してください。

コード リスト 2-5 のクラス例 :

1. JNDI を使用して、管理 MBeanHome インタフェースを取得します。
2. MBeanHome.getMBeansByType メソッドを使用して、ドメイン内のすべての ServerRuntime MBean のリストを取得します。
3. MBean のリストを Set オブジェクトに割り当て、Set インタフェースおよび Iterator インタフェースを使用してリスト内を検索します。

4. ServerRuntime.getName メソッドを使用して、各 ServerRuntime MBean の名前を取得します。ServerRuntime MBean の名前は、サーバ インスタンスの名前に対応します。
5. サーバ Server1 の ServerRuntime MBean が見つかると、標準出力にメッセージが出力されます。

この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 2-5 MBean のリストからのタイプによる選択

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.ObjectName;

import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.jndi.Environment;

public class serverRuntimeInfo {

    public static void main(String[] args) {

        MBeanHome home = null;

        // ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        //JNDI を使用して管理 MBeanHome を取得
        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
```

```
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //管理 MBeanHome インタフェースを取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }

        //getMBeansByType メソッドを使用してドメイン内のすべての ServerRuntime MBean
        //を取得
        try {
            mbeanSet = home.getMBeansByType("ServerRuntime");

            //取得結果をサーバ名と比較し、一致する
            //ServerRuntime MBean を検索
            mbeanIterator = mbeanSet.iterator();
            while(mbeanIterator.hasNext()) {
                serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
                //serverRuntime.getName を使用して Server1 の ServerRuntime
                //MBean を検索
                if(serverRuntime.getName().equals(serverName)) {
                    System.out.println("Got the serverRuntimembean: " +
                    serverRuntime + " for: " + serverName);
                 }
            }
         } catch (Exception e) {
               System.out.println("Exception caught:" +e);
         }
    }
}
```

`MBeanHome.getMBeansByType` メソッドの詳細については、WebLogic Server Javadoc を参照してください。

ローカルのコンフィグレーション MBean と実行時 MBean の階層を辿る

WebLogic Server の MBean は、それらが関連付けられているリソースを反映した階層の中に存在します。たとえば、各サーバ インスタンスには複数の実行キューを含めることができ、WebLogic Server では各 `ExecuteQueueMBean` が `ServerMBean` の子になることでこの関係が表されます。

MBean の階層を辿ることが、ローカル コンフィグレーション MBean と実行時 MBean を取得する最も簡単な方法です。管理 MBean を取得する必要がある場合、または管理 `MBeanHome` を使用して MBean を取得する必要がある場合は、まずタイプで MBean を取得し、それからそのリストをフィルタ処理することをお勧めします。2-12 ページの「タイプおよびリストからの選択による MBean の取得」を参照してください。

コンフィグレーション MBean 階層のルートは `DomainMBean` です。このルートの下には、以下のような MBean があります。

- `ClusterMBean`
- `ServerMBean`
- `ApplicationMBean`
- `RealmMBean`
- JDBC と JMS のコンフィグレーション MBean

実行時階層のルートは `ServerRuntimeMBean` です。このルートの直下には、以下のような MBean があります。

- `ClusterRuntimeMBean`
- `ApplicationRuntimeMBean`
- JDBC と JMS の実行時 MBean

親 MBean では通常、その子を取得するためのメソッドが提供されます。たとえば `ServerMBean.getExecuteQueues` は、そのサーバでコンフィグレーションされているすべての `ExecuteQueueMBean` を返します。

ローカルのコンフィグレーション MBean と実行時 MBean の階層を辿るには、次の手順を行います。

1. JMX アプリケーションから、ローカルの MBeanHome インタフェースを取得します。
2. ローカルの MBeanHome インタフェースから、以下のメソッドのいずれかを呼び出して最上位 MBean の 1 つを取得します。
   - `getConfigurationMBean (java.lang.String name, java.lang.String type)`

     MBeanHome.getConfigurationMBean の Javadoc を参照してください。
   - `getRuntimeMBean (java.lang.String name, java.lang.String type)`

     MBeanHome.getRuntimeMBean の Javadoc を参照してください。

   これらのメソッドでは、DomainMBean または ServerRuntimeMBean 直下の MBean のみ取得できます。これらのメソッドでは、MBean 階層の最初のレベルより下の MBean は返されません。
3. 取得した MBean から、MBean の子を取得するメソッドを呼び出します。

   親 MBean で子 MBean を取得するためのメソッドが提供されない場合は、`getMBeanByType()` を使用し、結果を繰り返し処理して基準と一致する MBean を見つけます。ローカル コンフィグレーション MBean を取得する場合は、必ず MBean タイプの値に `Config` を付け足してください。2-12 ページの「タイプおよびリストからの選択による MBean の取得」を参照してください。

注意： ローカル コンフィグレーション MBean は、値の読み込みのみを目的として取得することをお勧めします。ローカル コンフィグレーション MBean の属性値を変更しないでください。他の管理対象サーバのデータをレプリケートする際、管理サーバは管理 MBean に格納された値を使用します。管理 MBean とローカル コンフィグレーション MBean の値が異なると、通信上の問題が発生するおそれがあります。

コード リスト 2-6 は、ManagedServer1 という名前のサーバ インスタンスにあるすべてのローカル コンフィグレーション ExecuteQueueMBean を取得する例を示しています。

コード リスト 2-6 ローカル コンフィグレーション ExecuteQueueMBean の取得

```
import javax.naming.Context;
import javax.management.ObjectName;
```

```
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ConfigurationMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.configuration.ExecuteQueueMBean;

import weblogic.jndi.Environment;

public class serverConfigInfo {
    public static void main(String[] args) {
        MBeanHome home = null;
        ServerMBean servercfg = null;
        ExecuteQueueMBean[] xqueues = null;
        ExecuteQueueMBean xqueue = null;

        // ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "ManagedServer1";
        String username = "weblogic";
        String password = "weblogic";

        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);

            // 初期コンテキストを設定
            Context ctx = env.getInitialContext();

            // サーバ固有の MBeanHome インタフェースを取得
            home = (MBeanHome)ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
            System.out.println("Got the Server-specific MBeanHome: " + home);

           // ローカル コンフィグレーション ServerMBean を取得
            servercfg = (ServerMBean)home.getConfigurationMBean(serverName,
                                                    "ServerConfig");
           System.out.println("Got the Server Config MBean: " + servercfg);

           // サーバ インスタンスでコンフィグレーションされたすべての
           //ExecuteQueue MBean を取得
           xqueues = servercfg.getExecuteQueues();

           // 結果を繰り返し処理する
           for (int i=0; i < xqueues.length; i++){
                xqueue = xqueues[i];
                System.out.println("Execute queue name: " +
                    xqueue.DEFAULT_QUEUE_NAME);
              System.out.println("Thread count:" + xqueue.getThreadCount());
```

```
            }
        } catch (Exception e) {
             System.out.println("Exception caught:" +e);
         }
    }
}
```

サーバ コンフィグレーション MBean を取得するためにどのサーバ インスタンスでも実行できる汎用 JMX コードを作成する場合は、次の手順に従います。

1. ローカル `MBeanHome` インタフェースから、`getMBeansByType` メソッドを使用してサーバの `ServerRuntimeMBean` を取得します。
   `serverRuntime = MBeanHome.getMBeansByType(ServerRuntime)`

   ローカル `MBeanHome` インタフェースは現在のサーバ インスタンスに固有の実行時 MBean にしかアクセスできないので、`getMBeansByType(ServerRuntime)` は現在のサーバの `ServerRuntimeMBean` のみを返します。

2. `ServerRuntimeMBean` の `getName` メソッドを使用して、サーバの名前を取得します。
   `serverName = serverRuntime.getName()`

3. `MBeanHome.getConfigurationMBean` を呼び出すときにサーバ名を使用します。
   `MBeanHome.getConfigurationMBean(serverName,"ServerConfig")`

詳細については、4-1 ページの「例：アクティブなドメインとサーバの判別」を参照してください。

MBeanServer インタフェースを使用した MBean へのアクセス

MBean と対話する標準的な JMX の手法では、`javax.management.MBeanServer` インタフェースを使用して、MBean サーバに登録されている MBean をルックアップします。その後に、`MBeanServer` インタフェースを使用して、MBean 属性を取得または設定するか、MBean の操作を呼

び出します。MBeanServer のメソッドの詳細なリストについては、http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロードできる JMX 1.0 API のドキュメントを参照してください。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

WebLogic Server における JMX 実装では、MBeanHome インタフェースを使用して MBeanServer インタフェースをルックアップします。

コード リスト 2-7 のクラス例 :

1. JNDI を使用して、管理 MBeanHome インタフェースを取得します。この例では管理 MBean を取得しているため、管理 MBeanHome インタフェースを使用する必要があります。
2. 管理 MBeanHome インタフェースを使用して、MBeanServer インタフェースを取得します。
3. MBeanServer.queryNames メソッドを使用して、ドメイン内の JDBCConnectionPoolMBean のすべてのインスタンスをルックアップします。queryNames メソッド シグネチャでは、文字列 "examples:Type=JDBCConnectionPool,*" を Object としてキャストするための例が必要になります。
4. MBean のリストを Set オブジェクトに割り当て、Set インタフェースおよび Iterator インタフェースを使用してリスト内を検索します。

この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 2-7 MBeanServer インタフェースの使用

```
import java.util.Iterator;
import java.util.Set;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import javax.management.MBeanServer;
import javax.management.ObjectName;
import javax.management.QueryExp;
import weblogic.jndi.Environment;
```

```
import weblogic.management.MBeanHome;
import weblogic.management.RemoteMBeanServer;

public class ListJDBCInfo {

    public static void main(String[] args) {
        QueryExp query = null;
        MBeanHome home = null;
        RemoteMBeanServer homeServer = null;

        //ドメイン変数
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        //初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //管理 MBeanHome インタフェースを取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home + " from the
                               Admin server");

            //MBeanServer インタフェースを取得
            homeServer = home.getMBeanServer();

            //オブジェクト名に「JDBCConnectionPool」の含まれる
            //MBean のリストを取得
            Set JDBCMBeans = homeServer.queryNames(new
                    ObjectName("mydomain:Type=JDBCConnectionPool,*"), query);
            //「query」は JMX javax.managementQueryExp を実装する
            //オブジェクト

            for (Iterator itr = JDBCMBeans.iterator(); itr.hasNext(); ) {
                ObjectName mbean = (ObjectName)itr.next();
                System.out.println("Matches to the MBean query:" + mbean);
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

WebLogic Server MBean の WebLogicObjectName の使用

WebLogic Server MBean をインスタンス化すると、`weblogic.management.WebLogicObjectName` の規約に準拠した名前で登録されます。MBean の `WebLogicObjectName` がわかっている場合は、`MBeanHome` インタフェースを取得した後に名前で直接 MBean を取得できます。

MBean の `WebLogicObjectName` は、次の規約に基づいて、すべてのドメインにわたって固有となる識別名を特定の MBean に与えます。

domain:Name=*name*,Type=*type*[,Location=*serverName*]
[,*TypeOfParentMBean=NameOfParentMBean*][,*TypeOfParentMBean1=NameOfParentMBean1*]...

attribute=value の組み合わせの順序はそれほど重要ではありませんが、名前は必ず *domain*: から始める必要があります。また、MBean で複数の親 MBean を表現することもできます。

たとえば、次の例は `MyServer` というサーバ インスタンスにデプロイされているアプリケーション内の EJB の `EJBComponentRuntime` MBean の `WebLogicObjectName` です。この名前の属性と値の組み合わせのうち、1 番目と 4 番目 (`ApplicationRuntime=MyServer_MyEAR` と `ServerRuntime=MyServer`) はこの EJB の親 MBean を表します。

```
mydomain:ApplicationRuntime=MyServer_MyEAR,Location=MyServer,Name
=MyServer_MyEAR_SessionEJB,ServerRuntime=MyServer,Type=EJBCompone
ntRuntime
```

次の表では、この名前の各コンポーネントについて説明します。

表 2-3 WebLogic Server MBean の命名規約

コンポーネント	指定されるもの
domain	WebLogic Server 管理ドメインの名前。

表 2-3 WebLogic Server MBean の命名規約

コンポーネント	指定されるもの
Name=*name*	関連するリソースを作成したときに指定した文字列。たとえば、JDBC 接続プールを作成する場合、そのプールの名前 (MyPool1 など) を指定する必要がある。MyPool1 を表す JDBCConnectionPoolMBean は、その JMX オブジェクト名に Name=MyPool1 を使用する。 WebLogicObjectName.getName メソッドは、特定の MBean に対するこの値を返す。 MBean を作成したら、この Name コンポーネントの値として、ドメイン内の他のどの MBean とも異なるユニークな値を指定する必要がある。
Type=*type*	MBean がそのインスタンスとなっているインタフェース クラスを表す。すべての WebLogic Server MBean は、weblogic.management.configuration パッケージまたは weblogic.management.runtime パッケージで定義されているいずれかのインタフェース クラスのインスタンスである。コンフィグレーション MBean については、Type はインスタンスが管理 MBean であるのかそれともローカル コンフィグレーション MBean であるのかも示す。すべての WebLogic Server MBean インタフェース クラスの詳細なリストについては、weblogic.management.configuration パッケージまたは weblogic.management.runtime パッケージの WebLogic Server Javadoc を参照。 Type コンポーネントに指定する値を決めるには、次の手順に従う。 1. MBean のインタフェース クラスを見つけて、クラス名から MBean サフィックスを削除する。たとえば、weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean のインスタンスである MBean の場合は、JDBCConnectionPoolRuntime を使用する。 2. ローカル コンフィグレーション MBean の場合は、名前に Config を付加する。たとえば、weblogic.management.configuration.JDBCConnectionPoolMBean インタフェース クラスのインスタンスであるローカル コンフィグレーション MBean の場合は、JDBCConnectionPoolConfig を使用する。それに対応する管理 MBean インスタンスでは、JDBCConnectionPool を使用する。

表 2-3 WebLogic Server MBean の命名規約

コンポーネント	指定されるもの
`Location=servername`	すべての実行時 MBean とローカル コンフィグレーション MBean には `Location` コンポーネントがある。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指定する。管理 MBean にこのコンポーネントはない。 たとえば、`myserver` という名前のサーバで動作する `ServletRuntime` MBean では、`WebLogicObjectName` に次のコンポーネントが含まれる。 `mydomain:Name=myServlet,Type=ServletRuntime,Location=myserver` `WebLogicObjectName.getLocation` メソッドは、特定の MBean に対するこの値を返す。

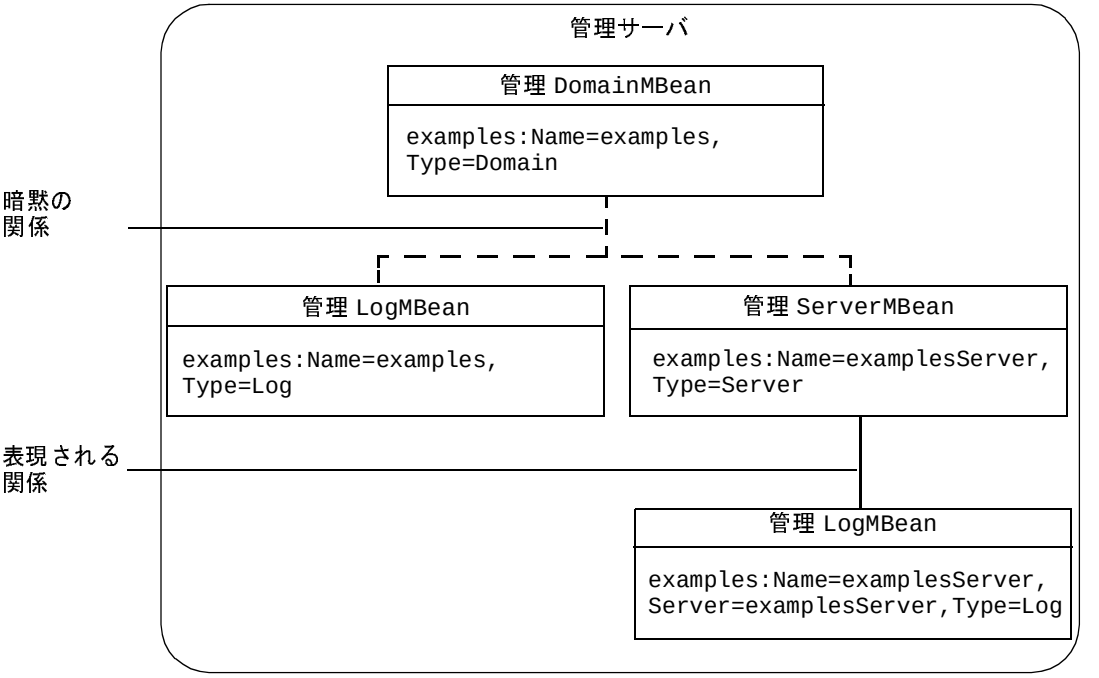
表 2-3 WebLogic Server MBean の命名規約

コンポーネント	指定されるもの

TypeOfParentMBean=NameOfParentMBean	親 MBean と子の関係を持つ実行時 MBean、ローカル コンフィグレーション MBean、または管理 MBean が、オブジェクト名でこの特別な属性を使用して関係を識別する。
	
	注意： DomainMBean を除き、すべての MBean はドメインの DomainMBean の直接的または間接的な子である。この親子関係はすべての MBean に当てはまるので、WebLogicObjectName では表現されない。
	
	たとえば、LogMBean のインスタンスは、ドメイン全体のログ ファイルをコンフィグレーションするためにドメインによって使用される。また、各 WebLogic Server インスタンスは、サーバ固有のログ ファイルをコンフィグレーションするために LogMBean の固有のインスタンスを保持する。ドメインで使用される LogMBean は子の関係を表現しないが、サーバ インスタンスで使用される LogMBean はサーバの ServerMBean との子の関係を表現する (図 2-1 を参照)。
	
	examplesServer がそのログ ファイルを管理するために使用する管理 LogMBean の名前を表現するには、次の名前を使用する。
	
	`examples:Name=examplesServer,Server=examplesServer, Type=Log`
	
	examplesServer がそのログ ファイルを管理するために使用するローカル コンフィグレーション LogMBean の名前を表現するには、次の名前を使用する。
	
	`examples:Location=examplesServer,Name=examplesServer, ServerConfig=examplesServer,Type=LogConfig`
	
	規約では、WebLogic Server の子 MBean は Name コンポーネントに親 MBean と同じ値を使用する。たとえば、examplesServer Server MBean の子である LogMBean はその WebLogicObjectName で Name=examplesServer を使用する。親 MBean に同じタイプの複数の子がある場合、WebLogic Server ではこの規約に従うことができない。
	
	MBean の WebLogicObjectName が親子関係を表現するかどうかを判断するには、WebLogicObjectName.getParent メソッドまたは weblogic.Admin GET コマンドを使用する。

図 2-1 では、LogMBean のインスタンスの 1 つは DomainMBean の子で、これを使用してドメイン全体のログ ファイルを管理しています。LogMBean のもう 1 つのインスタンスはサーバ インスタンスの ServerMBean の子で、これを使用してサーバ固有のログ ファイルを管理しています。WebLogicObjectName の *TypeOfParentMBean=NameOfParentMBean* コンポーネントにより、MBean インスタンスのアプリケーション内でのあいまいさが排除されています。

図 2-1 LogMBean インスタンスの親子関係



weblogic.Admin を使用した WebLogicObjectName の検索

MBean の WebLogicObjectName に指定する値がわからない場合は、weblogic.Admin ユーティリティを使用して WebLogicObjectName を検索できます。このユーティリティでは、アクティブなサーバ インスタンス上の WebLogic Server MBean についてのみ情報が返されます。

たとえば、examples ドメインにある LogMBean の管理インスタンスの WebLogicObjectName を検索するには、examplesServer 管理サーバで次のコマンドを入力します。管理サーバのリスン ポートは 8001 で、weblogic は MBean の属性を表示するパーミッションを持つユーザの名前とパスワードです。

```
java weblogic.Admin -url localhost:8001 -username weblogic
 -password weblogic GET -pretty -type Log
```

このコマンドでは、コード リスト 2-8 の出力が返されます。出力を見ると、このコマンドでは管理サーバ上の Log タイプの 2 つの MBean が返されることがわかります。最初の MBean (examples:Name=examplesServer,Server=examplesServer,Type=Log) には、examplesServer の ServerMBean と子の関係があります。この関係は、その MBean がサーバ固有のログ ファイルをコンフィグレーションする LogMBean であることを示します。2 番目の MBean (examples:Name=examples,Type=Log) には子の関係はありません。つまり、ドメイン全体のログ ファイルをコンフィグレーションする MBean であるということです。

-pretty を指定すると、各 MBean の属性と値が weblogic.Admin ユーティリティによって別の行に配置されます。この引数を指定しない場合は、それぞれの属性と値の組み合わせが中括弧 ({}) で囲まれ、すべてが 1 行に出力されます。

コード リスト 2-8 weblogic.Admin の出力

```
---------------------------
MBeanName:
"examples:Name=examplesServer,Server=examplesServer,Type=Log"
        CachingDisabled: true
        FileCount: 7
        FileMinSize: 500
```

```
        FileName: examplesServer\examplesServer.log
        FileTimeSpan: 24
        Name: examplesServer
        Notes:
        NumberOfFilesLimited: false
        ObjectName: examplesServer
        Parent: examplesServer
        Registered: false
        RotationTime: 00:00
        RotationType: none
        Type: Log

---------------------------
MBeanName: "examples:Name=examples,Type=Log"
        CachingDisabled: true
        FileCount: 7
        FileMinSize: 500
        FileName: ./logs/wl-domain.log
        FileTimeSpan: 24
        Name:examples
        Notes:
        NumberOfFilesLimited: false
        ObjectName:examples
        Parent:examples
        Registered: false
        RotationTime: 00:00
        RotationType: none
        Type: Log
```

LogMBean のローカル コンフィグレーション MBean インスタンスを表示するには、type 引数の値に Config を付加します。

```
java weblogic.Admin -url localhost:8001 -username weblogic
-password weblogic GET -pretty -type LogConfig
```

このコマンドでは、コード リスト 2-9 の出力が返されます。出力を見ると、ローカル コンフィグレーション MBean の WebLogicObjectName に Location コンポーネントが含まれていることがわかります。

コード リスト 2-9　ローカル コンフィグレーション MBean

```
---------------------------
MBeanName:
"examples:Location=examplesServer,Name=examplesServer,ServerConfi
g=examplesServer,Type=LogConfig"
        CachingDisabled: true
        FileCount: 7
        FileMinSize: 500
        FileName: examplesServer\examplesServer.log
```

```
        FileTimeSpan: 24
        Name: examplesServer
        Notes:
        NumberOfFilesLimited: false
        ObjectName: examplesServer
        Registered: false
        RotationTime: 00:00
        RotationType: none
        Type: LogConfig

---------------------------
MBeanName:
"examples:Location=examplesServer,Name=examples,Type=LogConfig"
        CachingDisabled: true
        FileCount: 7
        FileMinSize: 500
        FileName: ./logs/wl-domain.log
        FileTimeSpan: 24
        Name:examples
        Notes:
        NumberOfFilesLimited: false
        ObjectName:examples
        Registered: false
        RotationTime: 00:00
        RotationType: none
        Type: LogConfig
```

3 コンフィグレーション情報のアクセスと変更

管理サーバ上のコンフィグレーション MBean (管理 MBean) では、ドメイン内のすべての WebLogic Server インスタンスの管理対象リソースをコンフィグレーションします。パフォーマンスを向上させるには、各サーバ インスタンスにおいて管理 MBean のローカル レプリカを作成して使用します。このローカル レプリカをローカル コンフィグレーション MBean と呼びます。

注意： ローカル コンフィグレーション MBean の値を表示することはできますが、ローカル コンフィグレーション MBean の属性値を変更することはお勧めできません。属性値を変更する場合は、管理 MBean でのみ変更するようにしてください。ドメインのコンフィグレーション データをレプリケートする際、管理対象サーバは管理 MBean に格納された値を使用します。管理 MBean とローカル コンフィグレーション MBean の値が異なると、通信上の問題が発生するおそれがあります。

以下の節では、`weblogic.Admin` ユーティリティ、JMX `MBeanServer` API、および WebLogic Server 型保障インタフェースを使用して WebLogic Server リソースのコンフィグレーションをプログラム的に表示および変更する例を紹介します。

例 : weblogic.Admin を使用した標準出力のメッセージ レベルの表示

この例では、weblogic.Admin ユーティリティを使用して管理対象サーバに接続し、その StdoutSeverityLevel 属性の値をルックアップします。この属性は、サーバの ServerMBean 属性に属し、どの重大度のメッセージを標準出力に出力するかを判断するしきい値を示します。

値の変更は管理 MBean でのみ行うことをお勧めしますが、値を表示するにはローカル コンフィグレーション MBean の値をルックアップするほうがよい場合もあります。たとえば、管理サーバが停止していると、管理 MBean にはアクセスできなくなってしまいます。

このコマンド例は次のように機能します。

1. -url 引数を使用して、myHost というホストで稼動し、ポート 8001 をリスンする管理対象サーバに接続します。
2. -username 引数および -password 引数を使用して、MBean 属性を表示するためのパーミッションを持つユーザの資格を指定します。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。
3. GET コマンドを使用して、ローカル コンフィグレーション MBean を取得します。

   このコマンドでは、ローカル コンフィグレーション MBean を指定するために、ServerMBean インタフェース名から MBean を削除して Config を追加します。ServerMBean のローカル コンフィグレーション インスタンスの -type 値が ServerConfig であるのに対し、対応する管理 MBean インスタンスの -type 値が Server であることに注意してください。詳細については、2-21 ページの表 2-3「WebLogic Server MBean の命名規約」の Type の説明を参照してください。

コード リスト 3-1　メッセージ レベルのコンフィグレーション

```
java weblogic.Admin -url myHost:8001 -username weblogic -password weblogic
GET -pretty -type ServerConfig
```

```
--------------------------
MBeanName:
"examples:Location=examplesServer,Name=examplesServer,Type=ServerConfig"
        AcceptBacklog: 50
        AdministrationPort: 0
...

        StdoutDebugEnabled: false
        StdoutEnabled: true
        StdoutFormat: standard
        StdoutLogStack: true
        StdoutSeverityLevel: 16
```

例：標準出力のメッセージ レベルのコンフィグレーション

この例のクラスでは、`weblogic.management.configuration.ServerMBean` の `StdoutSeverityLevel` 属性の値を変更して、`examplesServer` という名前のサーバ インスタンスが標準出力に送信するメッセージのレベルを変更します。

この例では、コンフィグレーション値を変更する必要があるため、管理 MBean の値を変更しています。また、変更した内容は、WebLogic 管理サービスを使用して管理対象サーバに伝播しています。

このクラス例は次のように機能します。

1. JNDI を使用して、管理サーバの管理 `MBeanHome` インタフェースをルックアップします。
2. `MBeanHome.getMBean(String name, String type)` API を使用して、Server1 というサーバ インスタンスの `ServerMBean` 管理 MBean の型保障インタフェースを取得します。
3. 型保障インタフェースを使用して `ServerMBean.setStdoutSeverityLevel` メソッドを呼び出し、重大度を 64 に設定します。

この例で、`weblogic` は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 3-2　標準出力の重大度のコンフィグレーション

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.MBeanServer;
import javax.management.Attribute;
import java.lang.Object;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.configuration.ServerMBean;

public class ChangeStandardOut1 {

    public static void main(String[] args) {
        MBeanHome home = null;
        ServerMBean server = null;
        // ドメイン変数
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        String serverName = "Server1";

        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            // MBeanHome.getMBean(name, type) を使用して、ServerMBean の
            // 型保障インタフェースを取得
            server = (ServerMBean)home.getMBean(serverName,"Server");

            // ServerMBean.setStdoutSeverityLevel を使用
            server.setStdoutSeverityLevel(64);

            //  操作が成功したことをフィードバック
            System.out.println("Changed standard out severity level to: " +
                               server.getStdoutSeverityLevel());
        } catch (Exception e) {
            System.out.println("Caught exception:" +e);
        }
```

```
    }
}
```

4　実行時の情報へのアクセス

WebLogic Server には、管理対象リソースの実行時の状態に関する情報を提供する MBean が数多く用意されています。この実行時データを参照および変更するアプリケーションを作成するには、まず `MBeanServer` インタフェースまたは WebLogic Server 型保障インタフェースを使用して実行時 MBean を取得する必要があります。その後に、`weblogic.management.runtime` パッケージの API を使用して実行時データを参照または変更します。API ドキュメントの表示については、1-12 ページの「実行時 MBean API のドキュメント」を参照してください。

以下の節では、WebLogic Server ドメインとサーバ インスタンスに関する実行時情報を取得および修正する例を示します。

例 : アクティブなドメインとサーバの判別

`MBeanHome` インタフェースには、現在アクティブなドメインの名前およびサーバ インスタンスの名前を判別するために使用できる API があります。この情報は、引数としてドメイン名やサーバ名をとる他の API で使用できます。

コード リスト 4-1 のクラス例では以下の処理を実行します。

1. 管理 `MBeanHome` インタフェースを取得します。

   この例では管理 `MBeanHome` を使用していますが、ローカル `MBeanHome` インタフェースを使用することもできます。ローカル `MBeanHome` インタフェースを使用する場合は、現在のドメインの名前とサーバ インスタンスしか取得で

きません。管理 MBeanHome インタフェースを使用すると、ドメイン内で現在アクティブになっている**すべての**サーバの名前を取得できます。

2. MBeanHome.getActiveDomain().getName() を使用して、ドメインの名前を取得します。
3. getMBeansByType メソッドを使用して、ドメイン内のすべての ServerRuntime MBean のセットを取得します。
4. 取得したセットを検索し、ServerRuntimeMBean インスタンスの名前と WebLogic Server インスタンスの名前を比較します。インスタンスがアクティブであれば、そのサーバの名前を出力します。
5. インスタンスがアクティブであれば、そのサーバの名前を出力します。

次の例で、weblogic は MBean の属性を参照および変更するパーミッションを持つユーザのユーザ名とパスワードです。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-1　アクティブなドメインとサーバの判別

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class getActiveDomainAndServers {
    public static void main(String[] args) {
        MBeanHome home = null;

        //管理サーバの url
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
```

```
            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }

        //アクティブなドメインの名前を取得
        try {
            System.out.println("Active Domain: " +
                    home.getActiveDomain().getName() );
        } catch (Exception e) {
            System.out.println("Exception:" +e);
        }

        // ドメイン内のサーバの名前を取得
        System.out.println("Active Servers: ");
        Set mbeanSet = home.getMBeansByType("ServerRuntime");
        Iterator mbeanIterator = mbeanSet.iterator();
        while(mbeanIterator.hasNext()) {
            ServerRuntimeMBean serverRuntime =
                (ServerRuntimeMBean)mbeanIterator.next();
            // アクティブなサーバの名前を出力
          if(serverRuntime.getState().equalsIgnoreCase("RUNNING")) {
                System.out.println("Name: " + serverRuntime.getName());
                System.out.println("ListenAddress: " +
                          serverRuntime.getListenAddress());
                 System.out.println("ListenPort: " +
                          serverRuntime.getListenPort());
                 //count++;
            }
        }

        System.out.println("Number of servers active in the domain: " +
                           mbeanSet.size());
    }
}
```

weblogic.Admin を使用したアクティブなドメインおよびサーバの判別

コード リスト 4-1 のコード例をコンパイルおよび実行してアクティブなドメインとサーバを判別することも可能ですが、weblogic.Admin ユーティリティを使用すると、Java クラスをコンパイルしなくても同様のタスクを実行できます。

次のコマンドでは、現在アクティブなドメインの名前が返されます。AdminServer はドメインの管理サーバ、MyHost は管理サーバのホスト コンピュータ、weblogic は MBean の属性を参照するパーミッションを持つユーザの名前とパスワードです。

```
java weblogic.Admin -url MyHost:8001 -username weblogic -password
weblogic GET -type DomainRuntime -property Name
```

コマンドの出力には、DomainRuntimeMBean の WebLogicObjectName およびその Name 属性の値が含まれます。

```
{MBeanName="myDomain:Location=AdminServer,Name=myDomain,ServerRun
time=AdminServer,Type=DomainRuntime"{Name=myDomain}}
```

例：WebLogic Server インスタンスの実行時の状態の参照および変更

weblogic.management.runtime.ServerRuntimeMBean インタフェースは、WebLogic Server インスタンスの実行時の情報を提供します。たとえば、サーバが使用しているリスン ポートとアドレスを示します。また、このインタフェースにはサーバを正常にまたは強制的に停止する操作が組み込まれています。

この節では、ServerRuntimeMBean を検索し、その MBean を使用してサーバ インスタンスを正常に停止する例を示します。正常な停止を開始すると、サーバはサブシステムに作業中のすべての要求を完了するように指示します。サブシステムが作業を完了した後、サーバは停止します。

各例には、ServerRuntimeMBean を取得するための異なる方法を示してあります。

- 4-5 ページの「ローカル MBeanHome と getRuntimeMBean() の使用」
- 4-9 ページの「管理 MBeanHome と getMBean() の使用」
- 4-7 ページの「管理 MBeanHome と getMBeansByType() の使用」
- 4-11 ページの「MBeanServer インタフェースの使用」

weblogic.Admin ユーティリティでは、実行時 MBean の属性値を変更できません。

ローカル MBeanHome と getRuntimeMBean() の使用

各 WebLogic Server インスタンスは、独自の MBeanHome インタフェースをホストします。このインタフェースは、そのサーバ インスタンス上のローカル コンフィグレーション MBean と実行時 MBean へのアクセスを提供します。管理 MBeanHome インタフェースを使用する場合とは異なり、ローカル MBeanHome を使用すると、フィルタ処理によって現在のサーバに適用される実行時 MBean を見つけ出す手間が省けます。また、管理サーバを経由して要求をルーティングするのではなく、直接ローカル サーバに接続しているので、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。

MBeanHome インタフェースには、現在の WebLogic Server に存在する最上位の実行時 MBean だけを返す getRuntimeMBean() メソッドがあります (2-15 ページの「ローカルのコンフィグレーション MBean と実行時 MBean の階層を辿る」を参照)。MBeanHome.getRuntimeMBean() メソッドを管理サーバ上で呼び出した場合、管理サーバを管理およびモニタする実行時 MBean だけが返されます。

コード リスト 4-2 のクラス例では以下の処理を実行します。

1. t3://ServerHost:7001 で要求をリスンするサーバ インスタンスに接続するための情報をもとに javax.naming.Context オブジェクトをコンフィグレーションします。
2. Context.lookup メソッドを使用して、サーバ インスタンスのローカル MBeanHome インタフェースを取得します。

   MBeanHome.LOCAL_JNDI_NAME フィールドに、現在のサーバのローカル MBeanHome の JNDI 名が返されます。
3. MBeanHome.getRuntimeMBean(String name,String type) メソッドを使用して、現在のサーバ インスタンスの ServerRuntimeMBean を取得します。
4. ServerRuntimeMBean メソッドを呼び出して、サーバの状態を取得および変更します。

次の例で、weblogic は MBean の属性を参照および修正するパーミッションを持つユーザのユーザ名とパスワードで、Server1 は状態を参照および変更する必要のある WebLogic Server インスタンスの名前です。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-2　ローカル MBeanHome と getRuntimeMBean() の使用

```
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class serverRuntime1 {

    public static void main(String[] args) {
        MBeanHome home = null;

        // ドメイン変数
        String url = "t3://ServerHost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        ServerRuntimeMBean serverRuntime = null;

        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

           // ローカル MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
            System.out.println("Got the MBeanHome: " + home + " for server: " +
                                                              serverName);
        } catch (Exception e) {
            System.out.println("Caught exception:" +e);
        }

        // ここで、getRuntimeMBean メソッドを使用してサーバ インスタンスの
       //ServerRuntimeMbean にアクセスする

       try {
           serverRuntime =
```

```
                    (ServerRuntimeMBean)home.getRuntimeMBean
                    (serverName,"ServerRuntime");
            System.out.println("Got serverRuntimeMBean: " + serverRuntime);

            //ServerRuntimeMBean を使用して状態を取得および変更する
            System.out.println("Current state: " + serverRuntime.getState() );
        } catch (javax.management.InstanceNotFoundException e) {
            System.out.println("Caught exception:" +e);
        }

        try{
            serverRuntime.shutdown();
            System.out.println("Current state: " + serverRuntime.getState() );
        } catch (weblogic.server.ServerLifecycleException e) {
            System.out.println("Caught exception:" +e);
        }
    }
}
```

管理 MBeanHome と getMBeansByType() の使用

4-2 ページのコード リスト 4-1「アクティブなドメインとサーバの判別」の例と同じように、この節のクラス例では管理 `MBeanHome` インタフェースを使用して `ServerRuntime` MBean を取得します。管理 `MBeanHome` はドメイン内のすべての MBean への単一のアクセス ポイントとなりますが、これを使用するには、取得する MBean の `WebLogicObjectName` を構築するか、またはフィルタ処理を行って特定のサーバに存在する MBean を見つけ出す必要があります。

コード リスト 4-3 のクラス例では以下の処理を実行します。

1. 管理 `MBeanHome` インタフェースを取得します。
2. `MBeanHome.getMBeansByType` メソッドを使用して、ドメイン内のすべての `ServerRuntime` MBean のセットを取得します。
3. MBean のリストを `Set` オブジェクトに割り当て、`Set` インタフェースおよび `Iterator` インタフェースを使用してリスト内を検索します。
4. `ServerRuntimeMBean.getName` メソッドを使用して、MBean の `WebLogicObjectName` の `Name` コンポーネントを取得し、`Name` の値を別の値と比較します。

5. 特定のサーバ インスタンスの ServerRuntimeMBean が見つかると、ServerRuntimeMBean.getState メソッドを使用してサーバの現在の状態を返します。

6. ServerRuntimeMBean.shutdown() メソッドを呼び出して、正常な停止を開始します。

次の例で、weblogic はサーバの状態を変更するパーミッションを持つユーザのユーザ名とパスワードで、Server1 は状態を参照および変更する必要のある WebLogic Server インスタンスの名前です。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-3　管理 MBeanHome と getMBeansByType() の使用

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class serverRuntimeInfo {
    public static void main(String[] args) {
        MBeanHome home = null;

        //ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        //初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
```

```
        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }

        //getMBeansByType メソッドを使用して
        //ServerRuntime MBean を取得する
        try {
            mbeanSet = home.getMBeansByType("ServerRuntime");
            mbeanIterator = mbeanSet.iterator();
            // 各 ServerRutime MBean 内のサーバの名前を serverName
            // で指定された値と比較する
            while(mbeanIterator.hasNext()) {
                serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
                if(serverRuntime.getName().equals(serverName)) {
                    System.out.println("Found the serverRuntimembean: " +
                                       serverRuntime + " for: " + serverName);
                    System.out.println("Current state: " +
                                                 serverRuntime.getState() );
                    System.out.println("Stopping the server ...");
                    serverRuntime.shutdown();
                    System.out.println("Current state: " +
                                        serverRuntime.getState() );
                }
            }
        } catch (Exception e) {
            System.out.println("Caught exception:" +e);
        }
    }
}
```

管理 MBeanHome と getMBean() の使用

すべての MBean のリストを取得し、そのリストをフィルタ処理して特定のサーバの `ServerRuntimeMBean` を見つけ出す代わりに、この例では、MBean 命名規約を使用して `Server1` というサーバ インスタンス上の `ServerRuntimeMBean` の `WebLogicObjectName` を構築します。`WebLogicObjectName` の構築については、2-21 ページの「WebLogic Server MBean の WebLogicObjectName の使用」を参照してください。

正しいオブジェクト名を確実に指定するには、weblogic.Admin GET コマンドを使用します。たとえば次のコマンドでは、MyHost というホスト コンピュータで実行されるサーバ インスタンスの ServerRuntimeMBean のオブジェクト名と属性のリストが返されます。

```
java weblogic.Admin -url http://MyHost:7001 -username weblogic
 -password weblogic GET -pretty -type ServerRuntime
```

weblogic.Admin ユーティリティを使用した MBean 情報の検索については、『管理者ガイド』の「MBean 管理コマンド リファレンス」を参照してください。

コード リスト 4-4 で、weblogic は MBean の属性を参照および修正するパーミッションを持つユーザのユーザ名とパスワード、Server1 は状態を参照および変更する必要のある WebLogic Server インスタンスの名前、mihirDomain は WebLogic Server 管理ドメインの名前です。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-4　管理 MBeanHome と getMBean() の使用

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo2 {
    public static void main(String[] args) {
        MBeanHome home = null;
        // ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        String domain = "examples";
        ServerRuntimeMBean serverRuntime = null;

        // 初期コンテキストを設定
        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 Admin MBeanHome from the Admin server: "
                               + home);
        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }

        try {
            WebLogicObjectName objName = new WebLogicObjectName(serverName,
                          "ServerRuntime",home.getDomainName(),serverName);
           System.out.println("Created WebLogicObjectName: " + objName);
            serverRuntime = (ServerRuntimeMBean)home.getMBean(objName);
            System.out.println("Got the serverRuntime using the adminHome: " +
                               serverRuntime );
            System.out.println("Current state: " + serverRuntime.getState() );
            System.out.println("Stopping the server ...");

            // 状態を SHUTDOWN に変更
            serverRuntime.shutdown();
            System.out.println("Current state: " + serverRuntime.getState() );
        } catch(Exception e) {
            System.out.println("Exception:" +e);
        }
    }
}
```

MBeanServer インタフェースの使用

この節の例は、標準的な JMX の手法で MBean と対話します。管理 `MBeanHome` インタフェースを使用して `javax.management.MBeanServer` インタフェースを取得し、`MBeanServer` を使用して `Server1` というサーバ インスタンスの `ServerRuntimeMBean` の `ListenPort` 属性の値を取得します。

次の例で、`weblogic` は MBean の属性を参照および修正するパーミッションを持つユーザのユーザ名とパスワードで、`mihirDomain` は WebLogic Server 管理ドメインの名前です。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-5　管理 MBeanHome と getMBean() の使用

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;
import javax.management.MBeanServer;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo3 {
    public static void main(String[] args) {
        MBeanHome home = null;

        //ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        Object attributeValue = null;
        MBeanServer homeServer = null;

        //初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got Admin MBeanHome from the Admin server: " +
                                                                    home);
        } catch (Exception e) {
            System.out.println("Exception caught:" +e);
        }
        try {
            // MBean オブジェクト名を作成
            WebLogicObjectName objName = new WebLogicObjectName(serverName,
                        "ServerRuntime",home.getDomainName(),serverName);
            System.out.println("Created WebLogicObjectName: " + objName);

            //MBeanServer インタフェースを取得
            homeServer = home.getMBeanServer();
```

```
            //ServerRuntimeMBean の ListenPort 属性を取得
            attributeValue = homeServer.getAttribute(objName, "ListenPort");
            System.out.println("ListenPort for " + serverName + " is:" +
                                                                attributeValue);
        } catch(Exception e) {
            System.out.println("Exception:" +e);
        }
    }
}
```

例：クラスタに関する実行時の情報の参照

この節の例では、クラスタ内で現在実行されている WebLogic Server インスタンスの数と名前を取得します。この例では、`weblogic.management.runtime.ClusterRuntimeMBean` を使用します。この MBean は、1 つの管理対象サーバから見た WebLogic クラスタのメンバーの情報を提供します。

`ClusterRuntimeMBean` のインスタンスは管理対象サーバだけがホストします。また、`ClusterRuntimeMBean` のインスタンスはクラスタに参加しているアクティブな管理対象サーバから取得できます。

クラスタにあるアクティブな管理対象サーバから `ClusterRuntimeMBean` を取得するために、この例では次のことを行います。

1. 管理 `MBeanHome` を取得します。このインタフェースは管理サーバ上で実行され、ドメイン内のすべての `ClusterRuntimeMBean` へのアクセスを提供します。
2. すべての `ClusterRuntimeMBeans` を取得し、それらが対象クラスタに属しているかどうかを調べます。
3. 対象クラスタ内の管理対象サーバの `ClusterRuntimeMBean` を 1 つ見つけます。
4. 管理対象サーバの `ClusterRuntimeMBean` の API を使用して、クラスタ内のアクティブ サーバの数と名前を調べます。

この例で、weblogic は MBean の属性を参照および変更するパーミッションを持つユーザのユーザ名とパスワードです。MBean を変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 4-6　クラスタ内で実行中のサーバのリストの取得

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.runtime.ClusterRuntimeMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.MBeanHome;

public class getRunningServersInCluster {
    public static void main(String[] args) {
        MBeanHome home = null;

        // ドメイン変数
        String url = "t3://localhost:7001"; //url of the Administration Server
        /* ドメインに複数のクラスタが存在する場合、クラスタ内の
         * すべてのサーバのリストを定義する。このリストとドメイン内のサーバを比較して
         * 対象クラスタに属するサーバを特定する
         */
        String server1 = "cs1"; // クラスタ内のサーバの名前
        String server2 = "cs2"; // クラスタ内のサーバの名前
        String username = "weblogic";
        String password = "weblogic";
        ClusterRuntimeMBean clusterRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;
        String name = "";
        String[] aliveServerArray = null;

        // 初期コンテキストを設定
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
```

```
            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            // ドメイン内の ClusterRuntime MBean のリストを取得
            mbeanSet = home.getMBeansByType("ClusterRuntime");
            mbeanIterator = mbeanSet.iterator();
            while(mbeanIterator.hasNext()) {

                // リストから 1 つの ClusterRuntime MBean を取得
                clusterRuntime = (ClusterRuntimeMBean)mbeanIterator.next();
                // ClusterRuntime MBean の名前を取得
                name = clusterRuntime.getName();
                // 現在の ClusterRuntimeMBean が対象クラスタ内のサーバに
                // 属しているかどうかを特定
                if(name.equals(server1) || name.equals(server2) ) {
                    // 現在の ClusterRuntimeMBean を使用してクラスタ内の
                    // サーバの数を取得
                    System.out.println("\nNumber of active servers in the
                         cluster: " + clusterRuntime.getAliveServerCount());
                    // クラスタ内のサーバの名前を取得
                    aliveServerArray = clusterRuntime.getServerNames();
                    break;
                }
            }
        } catch (Exception e) {
             System.out.println("Caught exception:" +e);
        }
        if(aliveServerArray == null) {
            System.out.println("\nThere are no running servers in the cluster");
            System.exit(1);
        }

        System.out.println("\nThe running servers in the cluster are: ");
        for (int i=0; i < aliveServerArray.length; i++) {
            System.out.println("server " + i + " : " + aliveServerArray[i]);
        }
    }
}
```

EJB の実行時情報の表示

WebLogic Server では、サーバ インスタンスにデプロイ した EJB ごとに、weblogic.management.runtime パッケージから MBean タイプがインスタンス化されます (表 4-1 を参照)。weblogic.management.runtime パッケージに含まれる MBean の詳細については、WebLogic Server Javadoc を参照してください。

表 4-1 EJB の実行時情報を提供する MBean

MBean タイプ	説明
EJBComponentRuntimeMBean	EJB モジュール用に収集されるすべての実行時情報の最上位インタフェース。
StatefulEJBRuntimeMBean	ステートフル セッション Bean の場合にのみインスタンス化される。 ステートフル セッション Bean 用に収集された EJB 実行時情報にアクセスするためのメソッドを含む。
StatelessEJBRuntimeMBean	ステートレス セッション Bean の場合にのみインスタンス化される。 ステートレス セッション Bean 用に収集された EJB 実行時情報にアクセスするためのメソッドを含む。
MessageDrivenEJBRuntimeMBean	メッセージ駆動型 Bean の場合にのみインスタンス化される。 メッセージ駆動型 Bean 用に収集された EJB 実行時情報にアクセスするためのメソッドを含む。
EntityEJBRuntimeMBean	エンティティ Bean 用に収集された EJB 実行時情報にアクセスするためのメソッドを含む。
EJBCacheRuntimeMBean	EJB 用に収集されたキャッシュ実行時情報にアクセスするためのメソッドを含む。
EJBLockingRuntimeMBean	EJB 用に収集されたロック マネージャ実行時情報にアクセスするためのメソッドを含む。
EJBTransactionRuntimeMBean	EJB 用に収集されたトランザクション実行時情報にアクセスするためのメソッドを含む。

表 4-1 EJB の実行時情報を提供する MBean

MBean タイプ	説明
EJBPoolRuntimeMBean	ステートレス セッション Bean の場合にのみインスタンス化される。 ステートレス セッション Bean 用に収集されたフリー プール実行時情報にアクセスするためのメソッドを含む。 WebLogic Server では、フリー プールを使用してステートレス セッション EJB のパフォーマンスとスループットを高める。フリー プールには、非バインド ステートレス セッション EJB が格納される。非バインド EJB インスタンスはステートレス セッション EJB クラスのインスタンスで、メソッド呼び出しを処理しない。

WebLogic Server には、追加の抽象インタフェースとして EJBRuntimeMBean が用意されています。このインタフェースには、その他の EJB 実行時 MBean で使用するメソッドが含まれます。

EJB 実行時 MBean は階層内でインスタンス化されます。次に例を示します。

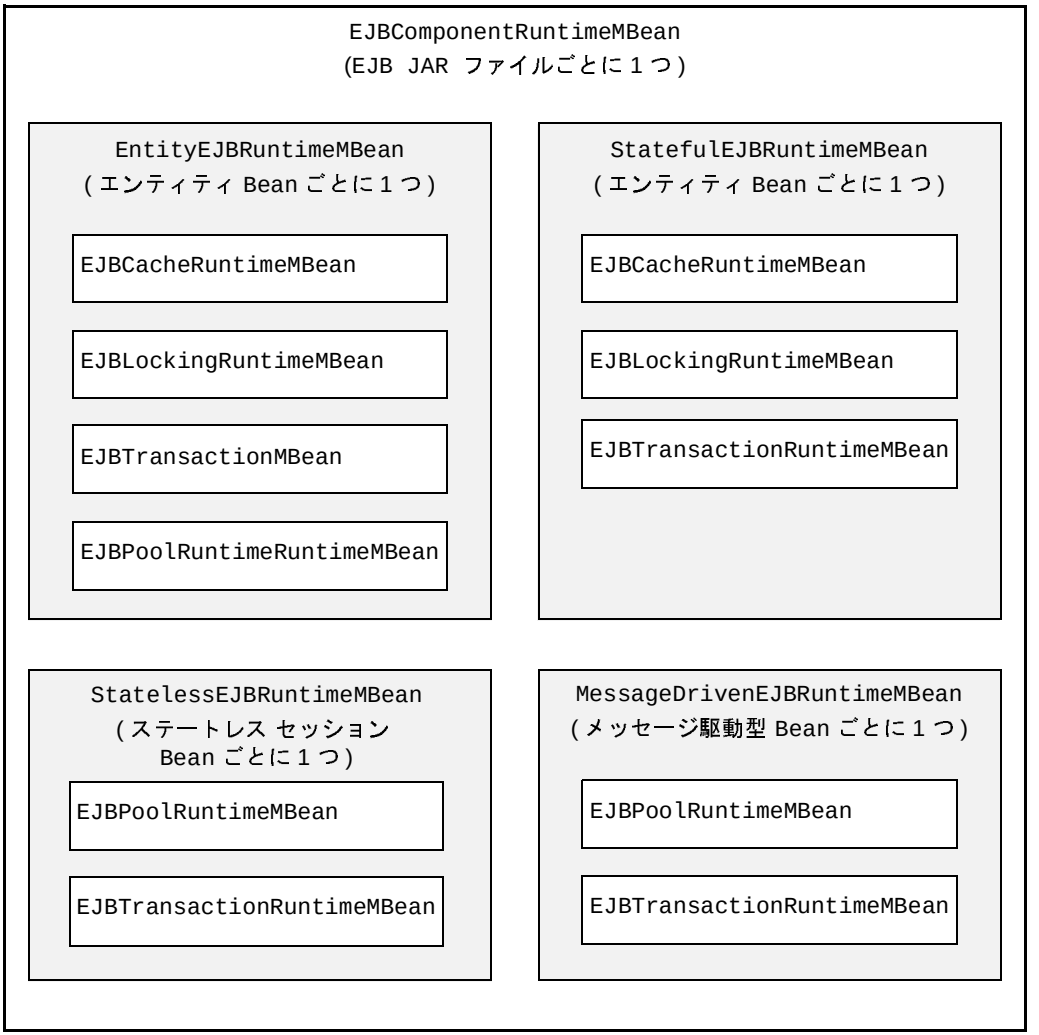
- 各 EJB jar ファイルは、EJBComponentRuntimeMBean のインスタンスを使用して実行時データをエクスポーズする。
- jar 内の各エンティティ Bean は、EntityEJBRuntimeMBean のインスタンスを使用して実行時データをエクスポーズする。
- 各 EntityEJBRuntimeMBean は、最大で 4 つの MBean の親になることができる。

  EntityEJBRuntimeMBean は、常に EJBCacheRuntimeMBean、EJBTransactionRuntimeMBean、および EJBPoolRuntimeMBean の親になります。4 つ目の子 MBean である EJBLockingRuntimeMBean は、エンティティ Bean が排他的同時方式 (weblogic-ejb-jar.xml デプロイメント記述子でコンフィグレーション) を使用する場合にのみ作成されます。

取得する実行時データのタイプによっては、データのコンテキストを提供するために親 MBean の名前も取得する必要があります。たとえば、`EJBTransactionRuntimeMBean.TransactionsRolledBackTotalCount` の値を取得する場合は、その値をどのエンティティ Bean から取得したかを識別するため、その親の名前 `EJBEntityRuntimeMBean` も取得します。

この階層関係を図 4-1 に示します。

図 4-1EJB 実行時 MBean の階層関係



例 : すべてのステートフル EJB およびステートレス EJB の実行時情報の取得

コード リスト 4-7 の例では、ドメイン内にデプロイされたすべての EJB の実行時情報を取得するために以下の処理を実行します。

1. 管理サーバに接続し、管理 `MBeanHome` インタフェースを取得します。

   特定のサーバ インスタンスにデプロイされた EJB のみの実行情報を取得する場合は、特定のサーバ インスタンスに接続してローカル `MBeanHome` インタフェースを取得します。詳細については、2-9 ページの「例 : 内部クライアントからのローカル MBeanHome の取得」を参照してください。

2. フリー プール内でアイドル状態の Bean インスタンスの数を表示するため、以下の処理を実行します。

   a. `MBeanHome.getMBeansByType` を呼び出してすべての `StatelessEJBRuntime` MBean を取得します。

   b. ステートレス EJB ごとに `displayEJBInfo` メソッド (このクラス内で定義) を呼び出します。このメソッドは以下の処理を実行します。

   - `StatelessEJBRuntimeMBean.getEJBName` メソッド (すべての EJB 実行時 MBean は `EJBRuntimeMBean` から継承) を呼び出して、MBean の名前を取得する。
   - MBean の階層を上がって、親 EJB コンポーネントおよびアプリケーションの名前を取得する。

     すべての EJB は、J2EE モジュールとして機能する EJB コンポーネント内にパッケージ化されます。EJB コンポーネントは、エンタープライズ アプリケーションを使用してパッケージ化できます。

   c. `StatelessEJBRuntime.getPoolRuntime` メソッドを呼び出して、ステートレス EJB に関連付けられている `EJBPoolRuntimeMBean` を取得します。

   d. `EJBPoolRuntimeMBean.getIdleBeansCount` メソッドを呼び出します。

3. ドメイン内の各ステートフル EJB がロールバックされたトランザクションの割合を調べるため、以下の処理を実行します。

a. MBeanHome.getMBeansByType を呼び出してすべての StatefulEJBRuntime MBean を取得します。

b. displayEJBInfo メソッド (このクラス内で定義) を呼び出します。

c. EJBRuntime.getTransactionRuntime メソッドを呼び出して、ステートフル EJB に関連付けられている EJBTransactionRuntimeMBean を取得します。

d. EJBTransactionRuntimeMBean.getTransactionsRolledBackTotalCount メソッドおよび getTransactionsCommittedTotalCount メソッドを呼び出します。

e. コミットされたトランザクションの数をロールバックされたトランザクションの数で割って、ロールバックされたトランザクションの割合を算出します。

コード リスト 4-7 EJB の実行時情報の表示

```
import java.util.Iterator;
import java.util.Set;
import javax.management.InstanceNotFoundException;
import javax.naming.Context;
import javax.naming.InitialContext;

import weblogic.management.MBeanHome;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ApplicationMBean;
import weblogic.management.configuration.EJBComponentMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.EJBComponentRuntimeMBean;
import weblogic.management.runtime.EJBPoolRuntimeMBean;
import weblogic.management.runtime.EJBRuntimeMBean;
import weblogic.management.runtime.EJBTransactionRuntimeMBean;
import weblogic.management.runtime.StatelessEJBRuntimeMBean;
import weblogic.jndi.Environment;

public final class EJBMonitor {
    private String url = "t3://localhost:7001";
    private String user = "weblogic";
    private String password = "weblogic";
    private MBeanHome   mBeanHome; // admin

    public EJBMonitor() throws Exception {
        Environment env = new Environment();
        env.setProviderUrl(url);
```

```
    env.setSecurityPrincipal(user);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();
    mBeanHome = (MBeanHome)ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
}

public void displayStatelessEJBPoolIdleCount()
throws Exception
{
    int idleCount = 0;
    String type = "StatelessEJBRuntime";
    Set beans = mBeanHome.getMBeansByType(type);
    System.out.println("Printing Stateless Session pool idle count:");
    for(Iterator it=beans.iterator();it.hasNext();) {
        StatelessEJBRuntimeMBean rt = (StatelessEJBRuntimeMBean)it.next();
        displayEJBInfo(rt);
        EJBPoolRuntimeMBean pool = rt.getPoolRuntime();
        idleCount = pool.getIdleBeansCount();
     }
     System.out.println("Pool Idle Bean Count: "+ idleCount +"\n");
}

public void displayStatefulEJBTransactionRollbackPercentages()
throws Exception
{
   String type = "StatefulEJBRuntime";
   Set beans = mBeanHome.getMBeansByType(type);
   System.out.println("Printing Stateful transaction rollback
          percentages:");
   for(Iterator it=beans.iterator();it.hasNext();) {
       EJBRuntimeMBean rt = (EJBRuntimeMBean)it.next();
       displayEJBInfo(rt);
       EJBTransactionRuntimeMBean trans = rt.getTransactionRuntime();
       String rollbackPercentage = "0";
       long rollbackCount = trans.getTransactionsRolledBackTotalCount();
       if(rollbackCount > 0) {
           long totalTransactions = rollbackCount +
                   trans.getTransactionsCommittedTotalCount();
           rollbackPercentage =
                   ""+(float)rollbackCount/totalTransactions*100;
        }

        System.out.println("Transaction rollback percentage: "+
           rollbackPercentage +"\n");
    }
}

private void displayEJBInfo(EJBRuntimeMBean rt) throws Exception {
    System.out.println("EJB Name: "+rt.getEJBName());
    EJBComponentRuntimeMBean compRTMBean =
```

```
            EJBComponentMBean compMBean = compRTMBean.getEJBComponent();
        ApplicationMBean appMBean = (ApplicationMBean)compMBean.getParent();
        System.out.println("Application Name: "+appMBean.getName());
        System.out.println("Component Name: "+compMBean.getName());
        WebLogicObjectName objName = rt.getObjectName();
        System.out.println("Server Name: "+objName.getLocation());
    }

    public static void main(String[] argv) throws Exception {
        EJBMonitor m = new EJBMonitor();
        m.displayStatelessEJBPoolIdleCount();
        m.displayStatefulEJBTransactionRollbackPercentages();
    }
}
```

5 WebLogic Server MBean 通知およびモニタの使い方

コンフィグレーション情報と実行時情報の変更をレポートするために、すべての WebLogic Server MBean は JMX 通知を送信します。**通知** は、基になるリソースで発生した状態の変更または他の特定の状態を説明する JMX オブジェクトです。

これらの通知をリスンするには、**リスナ**という Java クラスを作成できます。たとえば、アプリケーションのデプロイ、アンデプロイ、再デプロイ時に通知を受信するリスナをアプリケーションに組み込むことができます。

以下の節では、通知とリスナの使い方について説明します。

通知のブロードキャストと受信

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おり、発生したイベントのタイプに応じて異なるタイプの通知オブジェクトを送信できます。たとえば、MBean はその属性値が変更されると通知を送信します。

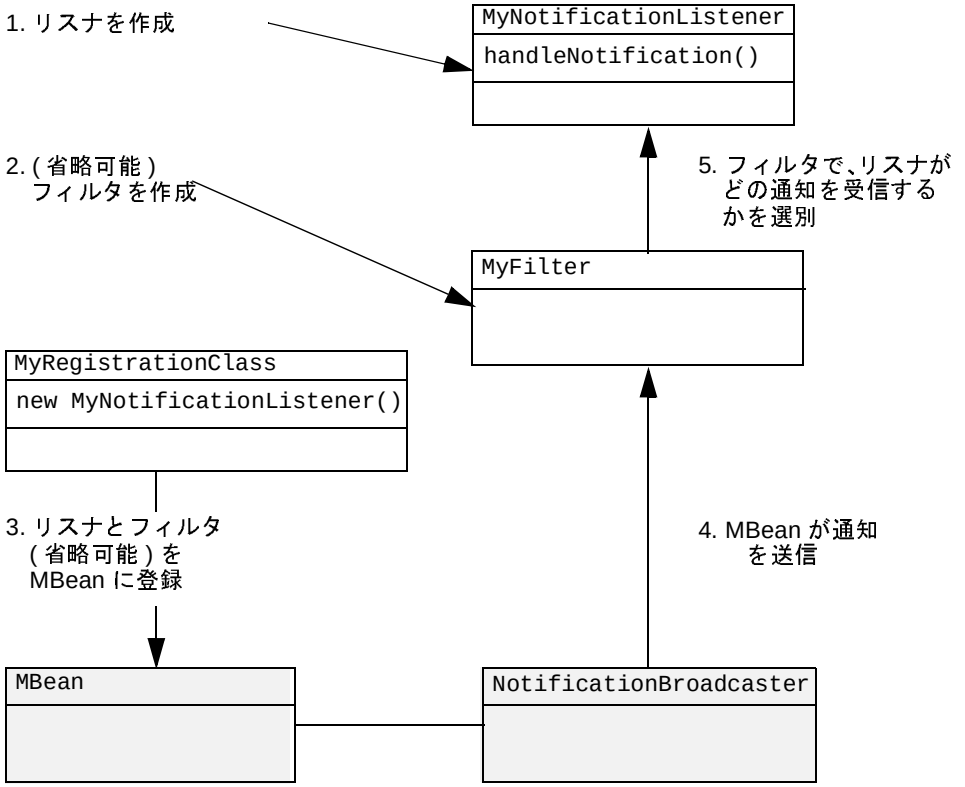
こうした通知をリスンするには、`javax.management.NotificationListener` を実装するリスナ クラスを作成します。

リスナは、デフォルトでは MBean から送信されるすべての通知を受信しますが、特定の通知のみを受信するように設定するのが一般的です。たとえば、`LogBroadCasterRuntime` MBean は WebLogic Server インスタンスがログ メッセージを生成するたびに通知を送信しますが、通常は特定のログ メッセージ (たとえば特定の重大度のメッセージ) のみをリスンします。リスナが受信する通知を限定するには通知フィルタを作成します。

リスナとフィルタ (省略可能) を作成したら、それらのクラスを通知の送信元となる MBean に登録します。

図 5-1 に、MBean がブロードキャストした通知のサブセットのみを受信する `NotificationListener` の基本的な仕組みを示します。

図 5-1 MBean からの通知の受信



JMX 通知とその仕組みの詳しい説明が必要な場合は、http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html から JMX 1.0 仕様をダウンロードしてください。

MBean 内での変更のモニタ

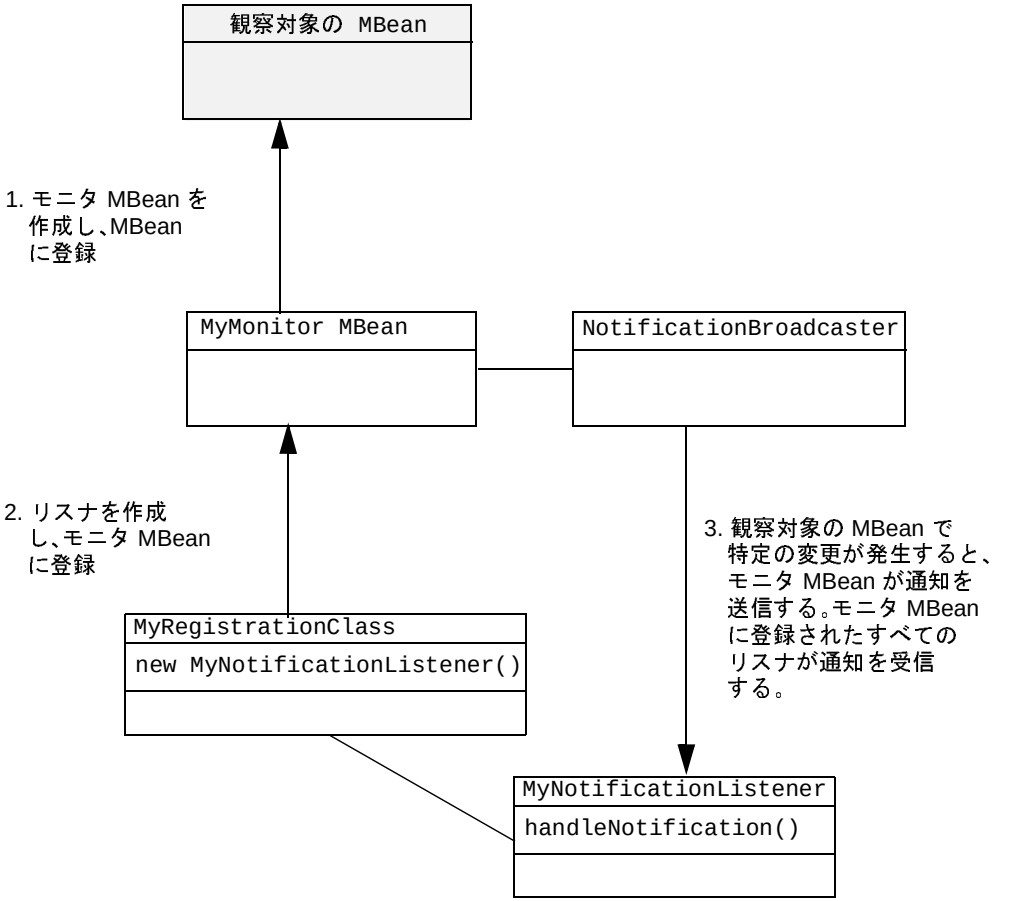
WebLogic Server には、MBean を定期的に観察し、特定の MBean 属性が所定のしきい値を超えたときにだけ JMX 通知を送信するようにコンフィグレーションできる**モニタ MBean** のセットが用意されています。モニタ MBean では、MBean の属性の値、または数値属性の 2 つの連続する値の差異 (オプション) を観察できます。モニタ MBean が観察した値を **派生ゲージ**と呼びます。

派生ゲージが一連の条件を満たす場合、モニタ MBean は特定の通知タイプを送信します。また、属性値のモニタ中に一定のエラー状態が発生した場合にも通知を送信できます。

モニタ MBean を使用するには、モニタをコンフィグレーションして WebLogic Server MBean に登録します。次に、リスナ クラスを作成して**モニタ MBean** に登録します。モニタ MBean は特定のタイプの通知のみを送信するため、モニタ MBean からの通知をリスンする場合はフィルタを使用しないのが一般的です。

図 5-2 に、WebLogic Server MBean に登録されたモニタ MBean の基本的な仕組みを示します。モニタ MBean に登録された `NotificationListener` は、モニタ MBean 内の条件が満たされたときに通知を受信します。

図 5-2 モニタ MBean



ベスト プラクティス：直接的なリスンとモニタの比較

WebLogic Server では、MBean 内での変更に関する通知を 2 つの方法で送信できます。1 つはリスナを作成して MBean に直接登録する方法 (図 5-1 を参照)、もう 1 つは MBean を定期的に観察するモニタ MBean をコンフィグレーションし、属性値が所定の条件を満たしたときに通知を送信する方法 (図 5-2 を参照) です。どちらの方法を選ぶかは、通知を受信する際の要件の複雑さによって決まります。

要件が単純な場合は、リスナを MBean に直接登録する方法をお勧めします。`NotificationListener` インタフェースおよび `NotificationFilter` インタフェースをリスナやフィルタに実装することで、値をしきい値などの値と比較することも可能です。ただし、通知内のデータを評価して応答するには、独自のコードを作成する必要があります。リスナを MBean に直接登録する利点は、通知が MBean からリスナに送信されるため、変更がほぼ即座に通知される点です。

モニタ MBean は、通知の要件が非常に複雑な場合や、MBean 属性の単一の変更とは直接関係のない複数の変更をまとめてモニタしたい場合に使用します。モニタ MBean には、非常に特殊な状況下でデータを比較したり通知を送信したりするためのさまざまなツールが用意されています。ただし、モニタは MBean を定期的にポーリングして属性値の変更を観察するため、指定したポーリング間隔でしか変更を通知できません。

ベスト プラクティス：よくモニタする属性

表 5-3 に、WebLogic Server のパフォーマンスの概況を表す属性を示します。これらの属性は、リスナを作成して MBean に直接登録する方法でも、モニタ MBean をコンフィグレーションする方法でもモニタできます。

リスナを作成して登録する場合、およびモニタ MBean をコンフィグレーションする場合は、モニタする属性を含む MBean の `WebLogicObjectName` を指定する必要があります。詳細については、5-15 ページの「通知リスナおよびフィルタの登録」および 5-28 ページの「モニタとリスナのインスタンス化」を参照してください。

表 5-3 の情報は、各 MBean の `WebLogicObjectName` の構築に使用します。表中の *`domain`* は WebLogic Server ドメインの名前、*`server`* はモニタする MBean をホストする WebLogic Server インスタンスの名前です。

表 5-3 よくモニタする WebLogic Server 属性

MBean と属性名	説明
MBean タイプ : `ServerRuntime` 属性名 : `State` この MBean の `WebLogicObjectName`: *`domain`*`:Location=`*`server`*`,Name=`*`server`*`, Type=ServerRuntime` 例 : `examples:Location=ExamplesServer,Name=ExamplesServer,Type=ServerRuntime`	サーバの状態が `Initializing`、`Suspended`、`Running`、または `ShuttingDown` のいずれにあるかを示す。
MBean タイプ : `ServerRuntime` 属性名 : `OpenSocketsCurrentCount` この MBean の `WebLogicObjectName`: 上の行を参照	これら 2 つの属性を組み合わせることで、サーバのリスン ポートにおける現在のアクティビティを、ポートにバックログできる要求の総数と比較できる。 これらの属性は、以下のように別々の MBean に格納されていることに注意。 ■ `OpenSocketsCurrentCount` は `ServerRuntime` MBean に格納 ■ `AcceptBacklog` は `Server` コンフィグレーション MBean に格納
MBean タイプ : `Server` 属性名 : `AcceptBacklog` この MBean の `WebLogicObjectName`: *`domain`*`:Name=`*`server`*`,Type=Server` 例 : `examples:Name=ExamplesServer,Type=Server`	

表 5-3 よくモニタする WebLogic Server 属性

MBean と属性名	説明
MBean タイプ：`ExecuteQueueRuntime` 属性名：`ExecuteThreadCurrentIdleCount` この MBean の `WebLogicObjectName`: *`domain`*`:Location=`*`server`*`,Name=default,ServerRuntime=`*`server`*`,Type=ExecuteQueueRuntime` 例： `examples:Location=ExamplesServer,Name=default,ServerRuntime=ExamplesServer,Type=ExecuteQueueRuntime`	サーバのデフォルトの実行キュー内において、メモリ空間を占有しているのにデータの処理には使用されていないスレッドの数を表示する。 サーバインスタンス上に複数の実行キューを作成して、重要度の高いアプリケーションのパフォーマンスを最適化できる。ただし、`default` 実行キューはデフォルトで使用可能。詳細については、「実行キューによるスレッド使用の制御」を参照。
MBean タイプ：`ExecuteQueueRuntime` 属性名：`PendingRequestCurrentCount` この MBean の `WebLogicObjectName`: 上の行を参照	サーバのデフォルトの実行キューで待機している要求の数を表示する。
MBean タイプ：`JVMRuntime` 属性名：`HeapSizeCurrent` この MBean の `WebLogicObjectName`: *`domain`*`:Location=`*`server`*`,Name=`*`server`*`,ServerRuntime=`*`server`*`,Type=JVMRuntime` 例： `examples:Location=ExamplesServer,Name=ExamplesServer,ServerRuntime=ExamplesServer,Type=JVMRuntime`	サーバの JVM ヒープにおいて現時点で使用可能なメモリの量 (バイト) を表示する。 詳細については、「Java 仮想マシン (JVM) のチューニング」を参照。

表 5-3 よくモニタする WebLogic Server 属性

MBean と属性名	説明
MBean タイプ : `JDBCConnectonPoolRuntime` 属性名 : `ActiveConnectionsCurrentCount` この MBean の `WebLogicObjectName`: *`domain`*`:Location=`*`server`*`,Name=`*`poolName`*`, ServerRuntime=`*`server`*`, Type=JDBCConnectionPoolRuntime` *`poolName`* は、接続プールの作成時に指定した名前。 例 : `examples:Location=ExamplesServer,Name=MyPool-PointBase,ServerRuntime=ExamplesServer,Type=JDBCConnectionPoolRuntime`	JDBC 接続プールにおいて現時点でアクティブな接続の数を表示する。 詳細については、「WebLogic Server のチューニング」を参照。
MBean タイプ : `JDBCConnectonPoolRuntime` 属性名 : `ConnectionsHighCount` この MBean の `WebLogicObjectName`: 上の行を参照	JDBC 接続プール内のアクティブな接続の最大数。接続プールがインスタンス化されるたびにゼロからカウントされる。

WebLogic Server MBean からの通知のリスン : 主な手順

WebLogic Server MBean から直接送信される通知をリスンするには、次の手順に従います。

1. どのタイプの通知をリスンするかを決めます。5-9 ページの「WebLogic Server 通知タイプ」を参照してください。
2. アプリケーションにリスナ クラスを作成します。5-10 ページの「通知リスナの作成」を参照してください。

3. 必要に応じてフィルタ クラスを作成し、リスナが MBean から受信する通知のタイプを指定します。5-13 ページの「通知フィルタの作成」を参照してください。
4. 通知の送信元となる MBean にリスナとフィルタを登録するための追加のクラスを作成します。5-15 ページの「通知リスナおよびフィルタの登録」を参照してください。

WebLogic Server 通知タイプ

WebLogic Server MBean には `javax.management.NotificationBroadcaster` インタフェースが実装されており、発生したイベントのタイプに応じて異なるタイプの通知オブジェクトを送信できます。

- MBean の属性値が変更されると、`javax.management.AttributeChangeNotification` オブジェクトが送信される。
- WebLogic Server リソースでログ メッセージが生成されると、サーバの `LogBroadcasterRuntimeMBean` から `weblogic.management.WebLogicLogNotification` タイプの通知が送信される。`WebLogicLogNotification` の詳細については、WebLogic Server Javadoc を参照してください。
- MBean が登録されるか、登録解除されると、WebLogic Server JMX サービスから `javax.management.MBeanServerNotification` タイプの通知が送信される。
- MBean 属性が配列である場合、MBean の `add`*`AttributeName`* メソッドを呼び出して配列に要素を追加すると、MBean から `weblogic.management.AttributeAddNotification` オブジェクトが送信される。`add`*`AttributeName`* メソッドをエクスポーズする MBean には、`weblogic.management.configuration.XMLRegistryMBean` などがあります。詳細については、WebLogic Server Javadoc を参照してください。
- MBean 属性が配列である場合、MBean の `remove`*`AttributeName`* メソッドを呼び出して配列から要素を削除すると、MBean から `weblogic.management.AttributeRemoveNotification` オブジェクトが送信される。

javax.management 通知タイプの詳細については、JMX 1.0 API のドキュメントを参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

weblogic.management 通知タイプの詳細については、AttributeAddNotification と AttributeRemoveNotification の Javadoc を参照してください。

通知リスナの作成

通知リスナを作成するには、次の手順に従います。

1. 次の**いずれか**を実装するクラスを作成します。
   - 同じ JVM で WebLogic Server として動作するクライアントの場合は、javax.management.NotificationListener を実装するクラス
   - リモート JVM で動作するクライアントの場合は、weblogic.management.RemoteNotificationListener を実装するクラス

     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。
2. クラスに、次の**いずれか**を追加します。
   - 同じ JVM で WebLogic Server として動作するクライアントの場合は、NotificationListener.handleNotification(Notification notification, java.lang.Object handback) メソッドを追加
   - リモート JVM で動作するクライアントの場合は、RemoteNotificationListener.handleNotification(Notification notification, java.lang.Object handback) メソッドを追加

   注意 : 通知ブロードキャスタのブロッキングを避けるため、このメソッドの実装はできる限り早く返す必要があります。
3. リスナが受信した通知オブジェクトからデータを取得するには、handleNotification メソッド内で通知オブジェクトに対して javax.management.Notification メソッドを呼び出します。

たとえば、通知に関連付けられたタイム スタンプを取得するには、`notification.getTimeStamp()` を呼び出します。

すべての通知タイプは `javax.management.Notification` の拡張であるため、以下の `Notification` メソッドはすべての通知に使用できます。

- `getMessage()`
- `getSequenceNumber()`
- `getTimeStamp()`
- `getType()`
- `getUserData()`

`Notification` メソッドの詳細については、JMX 1.0 API ドキュメント (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能) の `javax.management.Notification` Javadoc を参照してください。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

4. ほとんどの通知タイプには、その通知固有のデータを取得するための追加のメソッドが用意されています。たとえば、`WebLogicLogNotification` は、WebLogic Server ログ メッセージの特定の属性を取得するためのメソッドとして、ログ メッセージの重大度を取得する `getSeverity()` などのメソッドを備えています。

   通知タイプに固有の (つまり、標準の `javax.management.Notification` メソッドでは取得できない) データを取得するには、次の手順に従います。

   a. `handleNotification` メソッドに、通知をフィルタして特定のタイプの通知のみを選択するためのロジックを追加します。

   b. 通知タイプごとに用意されているメソッドを呼び出して、通知オブジェクトからデータを抽出します。

      次に例を示します。

```
if(notification instanceof MonitorNotification) {
  MonitorNotification monitorNotification =
(MonitorNotification)
                                                  notific
ation;
  System.out.println("This notification is a
MonitorNotification");
  System.out.println("Observed Attribute: " +
                        monitorNotification.getObservedAttrib
```

```
ute() );
}
```

NotificationListener クラスを作成する際は、上記の手順に加えて以下を考慮に入れてください。

- 通知フィルタを作成して使用しないと、リスナは登録された MBean からのあらゆる通知タイプの通知をすべて受信する。

  MBean が送信する可能性のある通知を 1 つのリスナで受信するのではなく、複数のフィルタとリスナを組み合わせて受信することをお勧めします。複数のリスナを使用すると JVM の初期化に時間がかかりますが、コードの保守が容易になるというメリットがあります。

- WebLogic Server 環境にモニタしたい MBean タイプのインスタンスが複数含まれている場合は、1 つの通知リスナを作成し、モニタする MBean インスタンスに応じて複数の登録クラスを作成できる。

  たとえば、WebLogic Server ドメインに 3 つの JDBC 接続プールが含まれている場合であれば、AttributeChangeNotifications をリスンするリスナクラスを 1 つ作成します。次に、JDBCConnectionPoolRuntime MBean の特定のインスタンスにリスナを登録するための登録クラスを 3 つ作成します。

- handleNotification メソッド シグネチャには handback オブジェクト用の引数が含まれるが、リスナは handback オブジェクトからデータを取得したり、handback オブジェクトを操作したりする必要はない。handback オブジェクトは、MBean エミッタに関する情報をリスナに関連付けるために使用するオブジェクトです。

次の例ではリモート リスナを作成します。このリスナは、AttributeChangeNotification オブジェクトを受信し、変更された値の属性名、変更前の値、および変更後の値を AttributeChangeNotification メソッドを使用して取得します。

コード リスト 5-1 通知リスナ

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import weblogic.management.RemoteNotificationListener;
import javax.management.AttributeChangeNotification;

public class MyListener implements RemoteNotificationListener {
```

```
    public  void handleNotification(Notification notification, Object obj) {

        if(notification instanceof AttributeChangeNotification) {
            AttributeChangeNotification attributeChange =
                     (AttributeChangeNotification) notification;
            System.out.println("This notification is an
                      AttributeChangeNotification");
            System.out.println("Observed Attribute: " +
                                      attributeChange.getAttributeName() );
            System.out.println("Old Value: " + attributeChange.getOldValue() );
            System.out.println("New Value: " + attributeChange.getNewValue() );
        }
    }
}
```

通知フィルタの作成

フィルタを作成して登録するには、次の手順に従います。

1. `javax.management.NotificationFilter` を実装するシリアライズ可能なクラスを作成します。

   必要に応じて、通知をフィルタするためのユーティリティ メソッドを提供する `javax.management.NotificationFilterSupport` クラスをインポートします。これらのメソッドの詳細については、JMX 1.0 API ドキュメントを参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

   フィルタがシリアライズ可能でなければならないのは、リモート通知リスナで使用する場合のみです。RMI で使用するクラスは、リモート JVM で分解および再構築する必要があるため、シリアライズ可能でなければなりません。

2. `isNotificationEnabled(Notification notification)` メソッドを使用して、シリアライズ可能なオブジェクトが一連の条件が満たされたときに true 値を返すかどうかを指定します。

   ブール値として true が返されると、フィルタは登録されているリスナに通知を転送します。

3. (省略可能) 通知からデータを取得し、これに基づいてアクションを実行するコードを含めることもできます。たとえば、フィルタで `javax.management.AttributeChangeNotification` メソッドを使用して特定の属性の変更後の値を取得し、値が指定したしきい値を超えていたら、JavaMail API を使用して管理者に電子メールを送信できます。

コード リスト 5-2 では、`AttributeChangeNotification` タイプの通知のみを転送する `NotificationFilter` の例を示します。

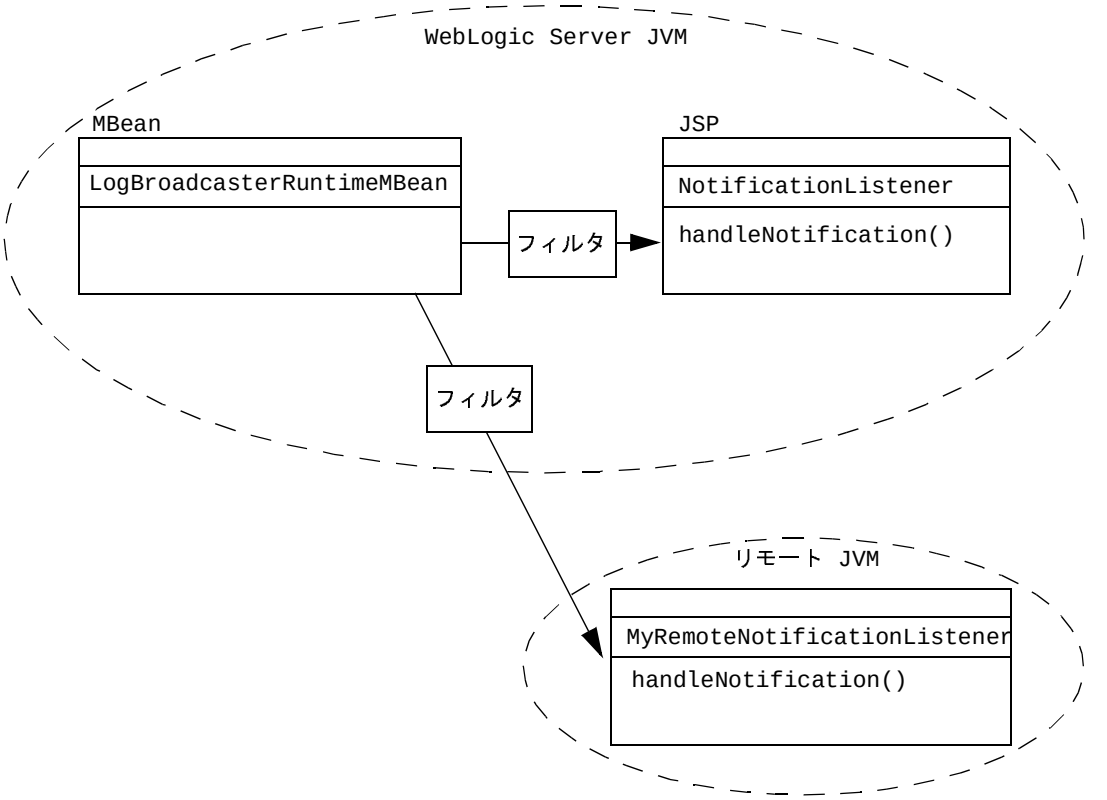
コード リスト 5-2 通知フィルタの例

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.AttributeChangeNotification;

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

    public boolean isNotificationEnabled(Notification notification) {
        if (!(notification instanceof AttributeChangeNotification)) {
            return false;
        }
        AttributeChangeNotification acn =
                    (AttributeChangeNotification)notification;
        acn.getAttributeName().equals("ActiveConnectionsHighCount"); {
            return true;
        }
    }
}
```

フィルタ クラスのサーバ クラスパスへの追加

リモート JVM で動作するリスナのフィルタを作成したら、通知をリスンするサーバ インスタンスのクラスパスにフィルタのクラスを追加できます。リスナはリモート JVM で動作しますが、フィルタのクラスをサーバのクラスパスに追加することで、フィルタとリスナの間でのシリアライズされたデータの伝達を最小限に抑えることができます。詳細については、図 5-4 を参照してください。

図 5-4 WebLogic Server で実行できるフィルタ



通知リスナおよびフィルタの登録

通知リスナ クラスとフィルタ クラス (省略可能) を実装したら、リスナとフィルタを MBean インスタンスに登録するための追加のクラスを作成します。登録クラスは、モニタする MBean インスタンスごとに 1 つずつ作成する必要があります。

通知リスナとフィルタを登録するには、次の手順に従います。

1. `MBeanHome` インタフェースを取得し、`MBeanHome` を使用して `MBeanServer` インタフェースを取得するクラスを作成します。

リスナとフィルタを管理 MBean に登録する場合は、管理サーバにしかない管理 MBeanHome を取得する必要があります。ローカル コンフィグレーション MBean または実行時 MBean に登録する場合は、その MBean をホストするサーバ インスタンスのローカル MBeanHome を取得する必要があります。

2. 作成したリスナ クラスとフィルタ クラスをインスタンス化します。
3. 登録先となる MBean の WebLogicObjectName を構築します。
4. MBeanServer インタフェースの addNotificationListener() メソッドに WebLogicObjectName、リスナ クラス、およびフィルタ クラスを渡して、リスナとフィルタを登録します。

   図 5-1 では、MBean の addNotificationListener() メソッドを呼び出してリスナとフィルタを直接 MBean に登録する場合を示しましたが、実際の場面では MBeanServer インタフェースの addNotificationListener() メソッドを使用することをお勧めします。これにより、登録のためだけに特定の MBean をルックアップする必要がなくなります。

次の例には、リモート JVM で動作する登録クラスを示します。同じ JVM で WebLogic Server インスタンスとして動作するクラスであれば、MBeanHome インタフェースを取得するコードはもっと単純になります。詳細については、2-4 ページの「MBeanHome インタフェースへのアクセス」を参照してください。

次のクラス例では、コード リスト 5-1 のリスナと コード リスト 5-2 のフィルタを、Server1 というサーバ インスタンスの Server 管理 MBean に登録します。この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

この登録クラスの例には、通知を取得するまでクラスをアクティブに保つためのコードも含まれています。クラスを呼び出してアクティブに保つのはより大きなアプリケーションの役割であり、リスナ クラスはこうした大きなアプリケーションのコンテキストで動作するため、このコードは通常は必要ありません。このコードが例に含まれているのは、簡単にコンパイルしてその動作を確認できるようにするためです。

コード リスト 5-3 管理 MBean へのリスナの登録

```
import java.util.Set;
import java.util.Iterator;
```

```
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.ObjectName;
import javax.management.Notification;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.configuration.ServerMBean;

public class listener {

    public static void main(String[] args) {

        MBeanHome home = null;
        RemoteMBeanServer rmbs = null;

        //ドメイン変数
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        //MBeanHome を使用して MBeanServer を取得
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // 管理 MBeanHome を取得
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
            rmbs = home.getMBeanServer();
        } catch (Exception e) {
            System.out.println("Caught exception:" +e);
        }

        try {
            // リスナ クラスをインスタンス化
            MyListener listener = new MyListener();
            MyFilter filter = new MyFilter();

            // リスンする MBean の WebLogicObjectName
            // を構築
            WebLogicObjectName mbeanName = new WebLogicObjectName(serverName,
```

```
                                "Server",home.getDomainName());
            System.out.println("Created WebLogicObjectName: " + mbeanName);

            //MBean の名前とリスナ クラスを MBeanServer
            // の addNotificationListener メソッドに渡す
            rmbs.addNotificationListener(mbeanName, listener, filter, null);
            System.out.println("\n[myListener]: Listener registered ...");

            // リモート クライアントをアクティブに保つ
            System.out.println("pausing...........");
            System.in.read();
        } catch(Exception e) {
            System.out.println("Exception:" +e);
        }
    }
}
```

コンフィグレーション監査メッセージのリスン：主な手順

デフォルトでは、管理サーバはユーザがコンフィグレーションを変更したとき、またはドメイン内のいずれかのリソースで管理操作を実行したときにログ メッセージを送出します。たとえば、ユーザがドメイン内の管理対象サーバで SSL を無効にすると、管理サーバはログ メッセージを送出します。そのようなメッセージは、ドメインのコンフィグレーションで行われた変更の証跡となります (コンフィグレーション監査)。『管理者ガイド』の「コンフィグレーション監査」を参照してください。

コンフィグレーション監査メッセージに応答する JMX リスナおよびフィルタを作成して使用するには、次の手順を行います。

1. WebLogic Server のログ メッセージから情報を抽出する通知リスナを作成およびコンパイルします。

   5-19 ページの「コンフィグレーション監査メッセージの通知リスナ」を参照してください。

2. コンフィグレーション監査メッセージのみを選択する通知フィルタを作成およびコンパイルします。

5-20 ページの「コンフィグレーション監査メッセージの通知フィルタ」を参照してください。

3. 管理サーバの LogBroadcasterRuntime MBean にリスナとフィルタを登録するクラスを作成およびコンパイルします。これは、ログ メッセージを JMX 通知としてブロードキャストするのに WebLogic Server インスタンスで使用される MBean です。

   5-21 ページの「コンフィグレーション監査メッセージの登録クラス」を参照してください。

4. 管理サーバのクラスパスに通知フィルタを追加します。

   通知リスナが管理サーバの JVM 内で動作する場合 (たとえば起動クラスとして実行される場合) は、通知リスナと登録クラスも管理サーバのクラスパスに追加します。

5. 登録クラスを呼び出すか、それを管理サーバの起動クラスとしてコンフィグレーションします。

   『管理者ガイド』の「起動クラスと停止クラスのコンフィグレーション」を参照してください。

コンフィグレーション監査メッセージの通知リスナ

コード リスト 5-1 の通知リスナと同様に、コード リスト 5-4 の通知リスナも RemoteNotificationListener とその handleNotification メソッドを実装します。

コンフィグレーション監査メッセージはすべて WebLogicLogNotification 型なので、コード リスト 5-4 のリスナは WebLogicLogNotification インタフェースをインポートし、そのメソッドを使用して各コンフィグレーション監査メッセージ内の情報を取得します。

コード リスト 5-4 コンフィグレーション監査メッセージの通知リスナ

```
import javax.management.Notification;
import javax.management.NotificationListener;
import weblogic.management.RemoteNotificationListener;
import weblogic.management.logging.WebLogicLogNotification;

public class ConfigAuditListener implements RemoteNotificationListener {
    public  void handleNotification(Notification notification, Object obj) {
```

```
            WebLogicLogNotification changeNotification =
                    (WebLogicLogNotification) notification;
            System.out.println("A user has attempted to change the configuration
                     of a WebLogic Server domain.");
            System.out.println("Admin Server Name: " +
                     changeNotification.getServername() );
            System.out.println("Time of attempted change:" +
                     changeNotification.getTimeStamp() );
            System.out.println("Message details:" +
                     changeNotification.getMessage() );
            System.out.println("Message ID string:" +
                     changeNotification.getMessageId() );
    }
}
```

コンフィグレーション監査メッセージの通知フィルタ

通知フィルタがない場合、コード リスト 5-4 のリスナは管理サーバがブロードキャストしたすべてのメッセージでサーバ名、タイムスタンプ、およびメッセージ テキストを出力します。

リソースが修正されたことを示すコンフィグレーション監査メッセージだけを転送するために、コード リスト 5-5 のフィルタは `WebLogicLogNotification.getMessageId` メソッドを使用して、受信したすべてのログ通知のメッセージ ID を取得します。

リソースが変更されたことを示すコンフィグレーション監査メッセージは、メッセージ ID `159904` で識別されます (『管理者ガイド』の「コンフィグレーション監査」を参照)。送信されてくるログ通知のメッセージ ID 値がコンフィグレーション監査メッセージの ID と一致する場合、フィルタは `true` として評価し、そのメッセージを登録リスナに転送します。

コード リスト 5-5 コンフィグレーション監査メッセージの通知フィルタ

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

public class ConfigAuditFilter implements NotificationFilter ,
                 java.io.Serializable{
    int configChangedId = 159904;
```

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

        }

        WebLogicLogNotification wln =
                   (WebLogicLogNotification)notification;
        int messageId = wln.getMessageId();
        if (configChangedId == messageId) {
                return true;

        } else {
          return false;
        }
    }
}
```

コンフィグレーション監査メッセージの登録クラス

コード リスト 5-6 のクラスは、通知リスナと通知フィルタを管理サーバの LogBroadcasterRuntime MBean に登録します。この MBean は WebLogic Server の各インスタンスのシングルトンであり、常に TheLogBroadcaster という名前です。

コード リスト 5-6 コンフィグレーション監査メッセージの登録クラス

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.ObjectName;
import javax.management.Notification;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.configuration.ServerMBean;

public class ListenRegistration {
    public static void main(String[] args) {
        MBeanHome home = null;
        RemoteMBeanServer rmbs = null;
```

```
// ドメイン変数
String url = "t3://localhost:7001";
String serverName = "examplesServer";
String username = "weblogic";
String password = "weblogic";

//MBeanHome を使用して MBeanServer を取得
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    //管理 MBeanHome インタフェースを取得
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got the Admin MBeanHome: " + home );
    rmbs = home.getMBeanServer();

} catch (Exception e) {
    System.out.println("Caught exception:" +e);
}

try {
    //リスナ クラスをインスタンス化
    ConfigAuditListener listener = new ConfigAuditListener();
    ConfigAuditFilter filter = new ConfigAuditFilter();

    //リスンする必要のある MBean の WebLogicObjectName
    //を作成

    WebLogicObjectName mbeanName = new WebLogicObjectName(
              "TheLogBroadcaster",
              "LogBroadcasterRuntime",
              home.getDomainName(),
              serverName );
    System.out.println("Created WebLogicObjectName: " + mbeanName);

    //MBean の名前とリスナ クラスを MBeanServer の
    //addNotificationListener メソッドに渡す
    rmbs.addNotificationListener(mbeanName, listener, filter, null);
    System.out.println("\n[myListener]: Listener registered ...");

    //リモート クライアントをアクティブに保つ
    System.out.println("pausing.....");
    System.in.read();
} catch(Exception e) {
    System.out.println("Exception:" +e);
```

```
        }
    }
}
```

モニタ MBean を使用した変更の観察：主な手順

モニタ MBean をコンフィグレーションおよび使用するには、次の手順に従います。

1. 観察するデータの型に対応するモニタ MBean タイプを選択します。5-23 ページの「モニタ MBean タイプの選択」を参照してください。
2. モニタ MBean からの通知をリスンできるリスナ クラスを作成します。5-26 ページの「モニタ MBean 用の通知リスナの作成」を参照してください。
3. モニタ MBean のコンフィグレーション、そのモニタ MBean へのリスナ クラスの登録、および観察対象 MBean へのモニタ MBean の登録を行うクラスを作成します。5-28 ページの「モニタとリスナのインスタンス化」

モニタ MBean タイプの選択

WebLogic Server には、特定のデータ型の変更を観察する専用の モニタ MBean が用意されています。MBean が返す属性値のオブジェクト タイプと一致するモニタ MBean タイプをコンフィグレーションしてインスタンス化する必要があります。たとえば、`StringMonitor` タイプに基づくモニタ MBean は、実際の属性値が (`instanceof` 演算子で定義されているように) `String` インスタンスである限り、`Object` として宣言されている属性を観察できます。

モニタ タイプを選択するには次の手順に従います。

1. 以下のいずれかを行って、観察する MBean 属性によって返されるオブジェクトの型を調べます。
   - WebLogic Server Javadoc を参照する。

- `weblogic.Admin GET` コマンドを使用する。このコマンドでは、指定する MBean に関する情報を取得できます。詳細については、『管理者ガイド』の「MBean 管理コマンド リファレンス」を参照してください。
- モニタする MBean に対して `javap` コマンドを使用する。`javap` コマンドは、クラス ファイルを逆アセンブルする Java ユーティリティです。

2. 次の表から、モニタ タイプを選択します。

表 5-1 モニタ MBean と観察対象オブジェクトのタイプ

モニタ MBean のタイプ	観察するオブジェクト タイプ
`CounterMonitor`	`Integer`
`GaugeMonitor`	整数または浮動小数点 (`Byte`、`Integer`、`Short`、`Long`、`Float`、`Double`)
`StringMonitor`	`String`

モニタ タイプの詳細については、JMX 1.0 仕様を参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

モニタの通知タイプ

それぞれのタイプのモニタ MBean は、特定のタイプの `javax.management.monitor.MonitorNotification` 通知を送信します。`MonitorNotification.getType()` メソッドを使用すると、あらゆる通知のタイプを判別できます。

次の表では、モニタ MBean が送信する通知のタイプについて説明します。

表 5-2 モニタ MBean と MonitorNotification のタイプ

モニタ MBean のタイプ	送信する MonitorNotification タイプ
CounterMonitor	カウンタ モニタは、カウンタの値がしきい値 (比較レベル) に到達したか、またはそれを超えた場合に jmx.monitor.counter.threshold を送信する。
GaugeMonitor	■ 観察対象の属性値が**増加**し、高しきい値と等しくなるかそれを超えた場合、モニタは通知タイプ jmx.monitor.gauge.high を送信する。この属性値が低しきい値と等しくなるかそれを下回った場合を除き、以後高しきい値に達しても通知は行われない。 ■ 観察対象の属性値が**減少**し、低しきい値と等しくなるかそれを下回った場合、モニタは通知タイプ jmx.monitor.gauge.low を送信する。この属性値が高しきい値と等しくなるかそれを超えた場合を除き、以後低しきい値に達しても通知は行われない。
StringMonitor	■ 観察対象の属性値が比較文字列値と**一致**した場合、モニタは通知タイプ jmx.monitor.string.matches を送信する。この属性値が比較文字列値と異なる場合を除き、以後この文字列が比較文字列と一致しても通知は行われない。 ■ この属性値が比較文字列値と**異なる**場合、モニタは通知タイプ jmx.monitor.string.differs を送信する。この属性値が比較文字列値と一致する場合を除き、以後この文字列が比較文字列と異なっても通知は行われない。

エラー通知タイプ

すべてのモニタは、エラー状態を示す以下の通知タイプを送信できます。

- `jmx.monitor.error.mbean`。この通知タイプは、観察対象 MBean が MBean サーバに登録されていないことを示します。この通知には、観察対象のオブジェクト名が示されます。
- `jmx.monitor.error.attribute`。この通知タイプは、観察対象の属性が観察対象オブジェクトに存在しないことを示します。この通知には、観察対象のオブジェクト名と属性名が示されます。
- `jmx.monitor.error.type`。この通知タイプは、観察対象の属性値のオブジェクト インスタンスが `null` か、またはモニタに対応していないタイプであることを示します。この通知には、観察対象のオブジェクト名と属性名が示されます。
- `jmx.monitor.error.runtime`。この通知タイプには、観察対象の属性値の取得中に (上記以外の理由で) 送出された例外が含まれます。

また、カウンタ モニタとゲージ モニタは、以下の状態のときに `jmx.monitor.error.threshold` 通知タイプも送信できます。

- カウンタ モニタの場合、しきい値、オフセット、または係数のタイプが観察対象のカウンタ属性と同じでない場合
- ゲージ モニタの場合、低しきい値または高しきい値が観察対象のゲージ属性と同じでない場合

モニタ MBean 用の通知リスナの作成

他の MBean と同じように、モニタ MBean は `javax.management.NotificationBroadcaster` を実装することによって通知を送信します。モニタ MBean からの通知のリスナを作成するには、次のようなクラスを作成します。

1. `NotificationBroadcaster` または `weblogic.management.RemoteNotificationListener` を実装する

2. `NotificationListener.handleNotification()` または `RemoteNotificationListener.handleNotification()` メソッドを組み込む

同じ通知リスナを `LogBroadcasterMBean` のインスタンス、モニタ MBean、または他の MBean に登録できます。

次の例では、WebLogic Server JVM の外部の JVM で実行されるアプリケーション用のリスナ オブジェクトを作成します。この例には、モニタ MBean から通知を受信したときに追加のメッセージを出力するロジックが含まれています。このロジックを修正して、モニタ通知のタイプに応じてリスナの応答を変化させることも可能です。モニタ通知のタイプについては、5-24 ページの「モニタの通知タイプ」を参照してください。

コード リスト 5-7 モニタ通知のリスナ

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

import weblogic.management.RemoteNotificationListener;
import weblogic.management.MBeanHome;

public class CounterListener implements RemoteNotificationListener {

    public void handleNotification(Notification notification ,Object obj) {
        System.out.println("\n\n Notification Received ...");
        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("This notification is a MonitorNotification");
            System.out.println("Observed Attribute: " +
                    monitorNotification.getObservedAttribute() );
            System.out.println("Observed Object: " +
                    monitorNotification.getObservedObject() );
            System.out.println("Trigger value: " +
                    monitorNotification.getTrigger() );
        }
```

```
    }
}
```

モニタとリスナのインスタンス化

モニタ MBean を管理対象 MBean に登録する手順は、モニタ MBean を単一のサーバ インスタンスに登録するか、ドメイン内の複数のサーバ インスタンスに登録するかによって異なります。

モニタ MBean を単一のサーバ インスタンスに登録するには、次の手順に従います。

1. モニタ MBean をインスタンス化してコンフィグレーションします。
2. 観察対象の MBean をホストするサーバ インスタンスの `MBeanHome` インタフェースを取得します。
3. モニタ MBean を観察対象 MBean に登録します。

モニタ MBean を複数のサーバ インスタンスに登録するには、次の手順に従います。

1. モニタ MBean をインスタンス化してコンフィグレーションします。
2. 観察対象 MBean のインスタンスをホストする**各サーバ インスタンス**の `MBeanHome` インタフェースを取得します。
3. 各サーバ インスタンスについて、モニタ MBean を観察対象 MBean に登録します。

以降の節では、これらの手順の例を示します。

- 5-29 ページの「例 : 単一のサーバにおける MBean のモニタ」
- 5-32 ページの「例 : 複数のサーバにおける MBean のインスタンスのモニタ」

例：単一のサーバにおける MBean のモニタ

次の例では、ExecuteQueRuntimeMBean の ServicedRequestTotalCount 属性用のモニタを作成します。この属性は、対応する実行キューによって処理された要求の数を返します。WebLogic Server は、実行キューを使用して、重要度の高いアプリケーションのパフォーマンスを最適化します。詳細については、「実行キューによるスレッド使用の制御」を参照してください。

コード リスト 5-8 の例では、単一のサーバ インスタンスの ExecuteQueRuntimeMBean 用のカウンタ モニタを次の手順で作成します。

1. javax.management.monitor.CounterMonitor オブジェクトをインスタンス化します。
2. 次の手順に従って、モニタ オブジェクトをコンフィグレーションします。
   a. **モニタ オブジェクト** の JMX オブジェクト名を変数に割り当てます。

      コード リスト 5-8 では WebLogicObjectName() を使用していますが、モニタ オブジェクトに対して javax.management.ObjectName を使用することもできます。オブジェクト名は WebLogic Server ドメイン全体にわたって固有にし、次の JMX 命名規約に従う必要があります。

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

   b. WebLogicObjectName() を使用して **観察対象 MBean** の JMX オブジェクト名を変数に割り当てます。

      観察対象 MBean が WebLogic Server MBean の場合、javax.management.ObjectName の代わりに WebLogicObjectName() を使用する必要があります。また、MBeanHome.getMBeansByType() または他の WebLogic Server API を使用して観察対象 MBean オブジェクトの名前を取得できます。MBean のさまざまな取得方法の例については、2-1 ページの「WebLogic Server MBean へのアクセス」を参照してください。

   c. モニタのしきい値パラメータの値を設定します。使用可能なパラメータ セットは、CounterMonitor、GaugeMonitor、StringMonitor のうちのどれをインスタンス化するかによって異なります。

   d. モニタの API を使用してモニタ オブジェクトをコンフィグレーションします。

モニタをコンフィグレーションするために渡すパラメータについては、以下を参照してください。

- 5-36 ページの「CounterMonitor オブジェクトのコンフィグレーション」
- 5-38 ページの「GaugeMonitor オブジェクトのコンフィグレーション」
- 5-39 ページの「StringMonitor オブジェクトのコンフィグレーション」.

3. 5-26 ページの「モニタ MBean 用の通知リスナの作成」で作成したリスナ オブジェクトをインスタンス化します。
4. モニタの `addNotificationListener()` メソッドを使用してリスナ オブジェクトを登録します。
5. (この手順は、モニタ クラスが WebLogic Server JVM の外部の JVM で実行される場合だけに必要) 次の手順に従って、リモート JVM にある MBean サーバの参照をあらかじめ登録します。
   a. 管理 `MBeanHome` インタフェースを使用して、`MBeanServer` インタフェースを取得します。
   b. モニタの `preRegister()` メソッドを使用します。
6. モニタの `start()` メソッドを使用してモニタを起動します。

この例で、`weblogic` は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『管理者ガイド』の「システム管理操作の保護」を参照してください。

コード リスト 5-8 モニタとリスナのインスタンス化

```
import javax.management.monitor.CounterMonitor;
import javax.management.ObjectName;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.configuration.ServerMBean;

public class clientMonitor {

    // WebLogic ドメインの名前。ここを、インストール環境に //
    // 合わせて特定のドメイン名に変更する                    //
    private static String weblogicDomain = "mydomain";
```

```
// WebLogic サーバの名前。ここを、インストール環境に //
// 合わせて特定のサーバ名に変更する                    //
private static String weblogicServer = "myserver";

public static void main (String Args[]) {

    try {
        //CounterMonitor をインスタンス化
        CounterMonitor monitor = new CounterMonitor();

        // CounterMonitor オブジェクトの objectName を構築
        WebLogicObjectName monitorObjectName = new
                 WebLogicObjectName("MyCounter",
                 "CounterMonitor",weblogicDomain);

        // 親 MBean の objectName を構築
        WebLogicObjectName pObjectName = new
                  WebLogicObjectName(weblogicServer,
                  "ServerRuntime",weblogicDomain);

        // 観察対象 MBean の objectName を構築
                  WebLogicObjectName qObjectName = new
                  WebLogicObjectName("default",
                  "ExecuteQueueRuntime",weblogicDomain,
                  weblogicServer, pObjectName);

        // CounterMonitor オブジェクトをコンフィグレーションするときに使用する
        // 変数を定義
        Integer threshold = new Integer(10);
        Integer offset = new Integer(1);

        //CounterMonitor API を使用してモニタ オブジェクトをコンフィグレーション
        monitor.setThreshold(threshold);
        monitor.setNotify(true);
        monitor.setOffset(offset);
        monitor.setObservedObject(qObjectName);
        monitor.setObservedAttribute("ServicedRequestTotalCount");

        // リスナをインスタンス化してモニタに登録
        CounterListener listener = new CounterListener();
        monitor.addNotificationListener(listener, null, null);

        // 管理 MBeanHome API を使用して MBeanServer インタフェースを取得
        // これはモニタをクライアント側から登録する場合に          // 必要
        String url = "t3://localhost:7001"; //URL of the Admin Server
        String username = "weblogic";
        String password = "weblogic";
        MBeanHome home = null;
        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);
            RemoteMBeanServer rmbs = home.getMBeanServer();
            monitor.preRegister(rmbs, monitorObjectName);

            //モニタを起動
            monitor.start();
        }catch (Exception e) { e.printStackTrace(); }
    }
}
```

例：複数のサーバにおける MBean のインスタンスのモニタ

WebLogic Server ドメインでは、サーバ インスタンスごとに複数の MBean インスタンスが保持されています。たとえば、各サーバ インスタンスは独自の `ServerRuntimeMBean`、`LogMBean`、および `ExecuteQueueRuntimeMBean` をホストします。

一部の MBean は、サーバ インスタンスが特定のサービスをホストする場合にのみインスタンス化されます。たとえば、Java Messaging Service (JMS) を使用する場合は、JMS 送り先として定義された各サーバ インスタンスが独自の `JMSDestinationRuntimeMBean` をホストします。JMS 送り先の詳細については、『WebLogic JMS プログラマーズ ガイド』の「分散送り先の使用」を参照してください。

コード リスト 5-9 の例では、ドメイン内の各サーバ インスタンスで `JMSDestinationRuntimeMBean` を次の手順でモニタします。

1. ドメインの管理 `MBeanHome` を取得します。
2. `MBeanHome.getMBeansByType` を呼び出して、ドメイン内の `JMSDestinationRuntimeMBean` のすべてのインスタンスを取得します。
3. 各 `JMSDestinationRuntimeMBean` について、`GaugeMonitor` オブジェクトが次の手順でインスタンス化およびコンフィグレーションされます。
   a. `javax.management.monitor.GaugeMonitor` のデフォルト コンストラクタを使用して `GaugeMonitor` オブジェクトをインスタンス化します。

b. GaugeMonitor オブジェクトをコンフィグレーションします。

GaugeMonitor.setObservedObject メソッドに値を提供するため、JMSDestinationRuntimeMBean が WebLogicMBean としてキャストされ、つづいて WebLogicMBean.getObjectName が呼び出されます。.

4. 各 GaugeMonitor オブジェクトについて GaugeMonitor.addNotificationListener メソッドが呼び出され、通知リスナがインスタンス化されてモニタに登録されます。

通知リスナの例については、5-26 ページの「モニタ MBean 用の通知リスナの作成」を参照してください。

5. 各 GaugeMonitor オブジェクトについて、ホスト サーバの JVM 内の MBean サーバへの参照が次の手順で登録されます。

a. Context.lookup(MBeanHome.JNDI_NAME.*serverName*) メソッドを呼び出し、管理 MBeanHome インタフェースを使用して MBeanServer インタフェースを取得します。

lookup メソッドに *serverName* 値を提供するため、JMSDestinationRuntimeMBean の WebLogicMBean キャストが参照され、その WebLogicMBean.getObjectName().getLocation() メソッドが呼び出されます。

b. モニタの preRegister() メソッドを呼び出します。

6. モニタの start() メソッドを使用してモニタを起動します。

7. クラスをアクティブに保つためのコードを挿入します。クラスを呼び出してアクティブに保つのはより大きなアプリケーションの役割であり、モニタはこうした大きなアプリケーションのコンテキストで動作するため、このコードは通常は必要ありません。このコードが例に含まれているのは、簡単にコンパイルしてその動作を確認できるようにするためです。

8. モニタを停止するコードを挿入します。これにより、JVM によってモニタに割り当てられたスレッドも終了します。

この例で、weblogic は MBean 属性を表示および変更するパーミッションを持つユーザです。MBean を表示および変更するパーミッションについては、『WebLogic リソースのセキュリティ』の「セキュリティ ロール」を参照してください。

コード リスト 5-9 複数のサーバ インスタンスにおけるゲージ モニタのインスタンス化

```
import java.util.Set;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import javax.naming.Context;

import javax.management.monitor.GaugeMonitor;
import javax.management.ObjectName;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.runtime.JMSDestinationRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class GaugeMonitorClient {

   public static void main (String Args[]) throws Exception {
      // 管理サーバの url
      String url = "t3://localhost:7001";
      String username = "weblogic";
      String password = "weblogic";
      String domain = "examples";

      try {
         // 管理 MBeanHome を取得
         Environment env = new Environment();
         env.setProviderUrl(url);
         env.setSecurityPrincipal(username);
         env.setSecurityCredentials(password);
         Context ctx = env.getInitialContext();
         MBeanHome home = (MBeanHome)
             ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

         // ドメイン内の JMSDestinationRuntimeMBean インスタンスを
         // すべて取得
         Set mbeanSet =
             home.getMBeansByType("JMSDestinationRuntime");

         System.out.println("Retrieved the following mbeans");
         Iterator iter = mbeanSet.iterator();
         while (iter.hasNext()){
            WebLogicMBean bean = (WebLogicMBean) iter.next();
            System.out.println("Name = "+bean.getName());
            System.out.println("WebLogicObjectName =
```

```
            "+bean.getObjectName()+"\n");
      }

      List list = Collections.synchronizedList(new ArrayList());
      Iterator it = mbeanSet.iterator();
      int i = 0;
      while (it.hasNext()) {
         // ゲージ モニタをインスタンス化
         GaugeMonitor monitor = new GaugeMonitor();
         // ゲージ モニタをコンフィグレーション
         monitor.setThresholds(new Long("30"), new Long("4"));
         monitor.setNotifyHigh(true);
         monitor.setNotifyLow(true);
         WebLogicMBean bean = (WebLogicMBean) it.next();
         ObjectName myON = bean.getObjectName();
         monitor.setObservedObject(myON);
         monitor.setObservedAttribute("MessagesCurrentCount");

         // 通知リスナをインスタンス化して登録
         MyNotificationListener listener = new MyNotificationListener();
         monitor.addNotificationListener(listener,null,null);
         // モニタをあらかじめ登録して起動
         MBeanHome localhome = (MBeanHome)
             ctx.lookup(MBeanHome.JNDI_NAME
             +"."+bean.getObjectName().getLocation());
         RemoteMBeanServer rmbs = localhome.getMBeanServer();
         WebLogicObjectName monitorObjectName = new WebLogicObjectName
             ("myGaugeMonitor" + (++i), "GaugeMonitor", domain,
             bean.getObjectName().getLocation());
         monitor.preRegister(rmbs, monitorObjectName);
         monitor.start();
         System.out.println("Monitor waiting on event notification.");
         list.add(monitor);
         myON = null;
      }

      //モニタをアクティブに保つ
      System.out.println("pausing..........");
      System.in.read();

      //各モニタを停止
      Iterator deregisterList = list.iterator();
      while (deregisterList.hasNext()) {
         GaugeMonitor gauge = (GaugeMonitor) deregisterList.next();
         System.out.println("deregistering...");
         gauge.preDeregister();
      }
      return;
   }
```

```
      catch (Exception e){
         e.printStackTrace();
      }
   }
}
```

CounterMonitor オブジェクトのコンフィグレーション

CounterMonitor オブジェクトは、整数で表される MBean 属性の変更を観察します。次に、CounterMonitor インスタンスの一般的なコンフィグレーションを行うために使用する一連の CounterMonitor 操作を示します。

- 観察対象の属性がしきい値を超えたときに通知を送信する。

  ```
  setThreshold(int threshold);
  setNotify(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- 観察対象の属性がしきい値を超えたときに通知を送信する。次にオフセット値分だけしきい値を増加させます。観察対象の属性が新しいしきい値を超えるたびに、しきい値はオフセット値だけ増加します。たとえば、Threshold を 1000 に、Offset を 2000 に設定した場合、観察対象の属性が 1000 を超えると、CounterMonitor オブジェクトは通知を送信して、しきい値を 3000 に増やします。観察対象の属性が 3000 を超えると、CounterMonitor オブジェクトは通知を送信して、しきい値を再び 5000 に増やします。

  ```
  setThreshold(int threshold);
  setNotify(true);
  setOffset(int offset);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- 観察対象の属性がしきい値を超えたときに通知を送信し、次にオフセット値分だけしきい値を増加させる。setModulus メソッドで指定された値にしきい値が達すると、しきい値はモニタの setThreshold メソッドの最後の呼び出しで指定されたオフセットの適用前の値に戻ります。たとえば、元の Threshold が 1000 に、Modulus が 5000 に設定されている場合、

Threshold が 5000 を超えると、モニタは通知を送信して、Threshold を 1000 に戻します。

```
setThreshold(int threshold);
setNotify(true);
setOffset(int offset);
setModulus(int modulus);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- 連続する 2 つの観察値の差異がしきい値を超えたときに通知を送信する。たとえば、Threshold が 20 で、モニタが属性値 2 を観察したとします。この場合、次の観察値が 22 を超えると、モニタは通知を送信します。しかし、次の観察で値が 10 であり、その次の観察で 25 である場合、2 つの連続する観察値の変化はいずれも 20 以下なので、モニタは通知を送信しません。

  ```
  setThreshold(int threshold);
  setNotify(true);
  setDifferenceMode(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- 連続する 2 つの観察値の差異がしきい値を超えたときに通知を送信して、しきい値をオフセット値だけ増加させる。setModulus メソッドで指定された値にしきい値が達すると、しきい値はモニタの setThreshold メソッドの最後の呼び出しで指定されたオフセットの適用前の値に戻ります。

  ```
  setThreshold(int threshold);
  setNotify(true);
  setOffset(int offset);
  setModulus(int modulus);
  setDifferenceMode(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

CounterMonitor インスタンスで可能なすべてのコンフィグレーションを確認するには、JMX 1.0 API のドキュメントを参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

GaugeMonitor オブジェクトのコンフィグレーション

GaugeMonitor オブジェクトは、整数または浮動小数点で表される MBean 属性の変更を観察します。次に、GaugeMonitor インスタンスの一般的なコンフィグレーションを行うために使用する一連の GaugeMonitor 操作を示します。

- 観察対象の属性が高しきい値を超えたときに通知を送信する。

```
setHighThreshold(int Highthreshold);
setNotifyHigh(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- 観察対象の属性が高しきい値または低しきい値の範囲を出たときに通知を送信する。

```
setThresholds(int Highthreshold, Lowthreshold);
setNotifyHigh(true);
setNotifyLow(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- 連続する 2 つの観察値の差異が高しきい値または低しきい値の範囲を出たときに通知を送信する。

```
setThresholds(int Highthreshold, Lowthreshold);
setNotifyHigh(true);
setNotifyLow(true);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

GaugeMonitor はオフセットまたは係数をサポートしていません。

GaugeMonitor インスタンスで可能なすべてのコンフィグレーションを確認するには、JMX 1.0 API のドキュメントを参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

StringMonitor オブジェクトのコンフィグレーション

StringMonitor オブジェクトは、文字列で表される MBean 属性の変更を観察します。次に、StringMonitor インスタンスの一般的なコンフィグレーションを行うために使用する一連の StringMonitor 操作を示します。

- 観察対象の属性が StringToCompare に指定されている文字列と**一致した**ときに通知を送信する。

```
setStringToCompare(String);
setNotifyMatch(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- 観察対象の属性が StringToCompare に指定されている文字列と**異なる**ときに通知を送信する。

```
setStringToCompare(String);
setNotifyDiffer(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

StringMonitor インスタンスで可能なすべてのコンフィグレーションを確認するには、JMX 1.0 API のドキュメントを参照してください (http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html からダウンロード可能)。ダウンロードしたアーカイブに、API ドキュメントが格納されています。

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の詳細については、該当の MBean の WebLogic Server Javadoc 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。

表 5-3 JDBC のモニタ属性

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。

索引

R

S

W

あ

い

え

お

か

く

け

こ

さ

し

ろ