



# BEA WebLogic Server™

## WebLogic RMI プログ ラマーズ ガイド

BEA WebLogic Server バージョン 7.0  
マニュアルの日付 : 2002 年 6 月  
改訂 : 2004 年 9 月 8 日

## 著作権

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

## 限定的権利条項

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

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

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

## 商標または登録商標

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

その他の商標はすべて、関係各社がその権利を有します。

## WebLogic RMI プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002年6月28日	BEA WebLogic Server バージョン 7.0

---

# 目次

## このマニュアルの内容

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

## 1. WebLogic RMI について

WebLogic RMI とは.....	1-1
WebLogic RMI の機能.....	1-2

## 2. WebLogic RMI の機能とガイドライン

WebLogic RMI フレームワーク.....	2-1
WebLogic RMI コンパイラ.....	2-2
スタブとスケルトンから動的プロキシとバイトコードへの移行.....	2-3
WebLogic RMI コンパイラのオプション.....	2-3
クラスタ内のレプリケーションされないスタブ.....	2-6
WebLogic RMI コンパイラのその他の機能.....	2-6
RMI の動的プロキシ.....	2-6
WebLogic RMI コンパイラとプロキシの使い方.....	2-7
ホット コード生成.....	2-8
RMI と T3 プロトコル.....	2-8

## 3. WebLogic RMI の実装

WebLogic RMI API の概要.....	3-1
WebLogic RMI の実装の手順.....	3-2
リモートで呼び出すことができるクラスを作成する.....	3-3
手順 1. リモートインタフェースを作成する.....	3-3
手順 2. リモートインタフェースを実装する.....	3-4
手順 3. Java クラスをコンパイルする.....	3-6

---

手順 4. 実装クラスを <b>RMI</b> コンパイラでコンパイルする .....	3-6
手順 5. リモート メソッドを呼び出すコードを記述する .....	3-7
完全なコード例 .....	3-8

---

# このマニュアルの内容

このマニュアルでは、Sun Microsystems による JavaSoft Remote Method Invocation (RMI) の BEA WebLogic Server™ RMI 実装について説明します。BEA の実装は WebLogic RMI と呼ばれます。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic RMI について」では、WebLogic RMI の機能とアーキテクチャの概要について説明します。
- 第 2 章「WebLogic RMI の機能とガイドライン」では、WebLogic Server の RMI のプログラミングで使用する機能について説明します。
- 第 3 章「WebLogic RMI の実装」では、WebLogic RMI の一部として添付されているパッケージについて説明し、WebLogic RMI の実装方法を示します (パブリック API には、WebLogic で実装された RMI 基本クラス、レジストリ、およびサーバのパッケージが含まれています)。

## 対象読者

このマニュアルは、Remote Method Invocation (RMI) 機能を使用して e- コマースアプリケーションを構築するアプリケーション開発者を対象としています。Web テクノロジ、オブジェクト指向プログラミング手法、および Java プログラミング言語に読者が精通していることを前提として書かれています。

## e-docs Web サイト

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

---

# このマニュアルの印刷方法

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

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

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

## 関連情報

BEA の Web サイトでは、WebLogic Server の全マニュアルを提供しています。このマニュアル以外に、『WebLogic RMI over IIOP プログラマーズ ガイド』を参照する場合があります。WebLogic RMI over IIOP を使用すると、クライアントは、Internet Inter-ORB Protocol (IIOP) を介して RMI リモート オブジェクトにアクセスできるため、RMI プログラミング モデルが拡張されます。

## 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](mailto: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、WebLogic Server の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。

---

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](http://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。
<i>斜体</i>	強調または書籍のタイトルを示す。

表記法	適用
等幅テキスト	<p>コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。</p> <p>例：</p> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
斜体の等幅テキスト	<p>コード内の変数を示す。</p> <p>例：</p> <pre>String <i>CustomerName</i>;</pre>
すべて大文字のテキスト	<p>デバイス名、環境変数、および論理演算子を示す。</p> <p>例：</p> <pre>LPT1 BEA_HOME OR</pre>
{ }	<p>構文の中で複数の選択肢を示す。</p>
[ ]	<p>構文の中で任意指定の項目を示す。</p> <p>例：</p> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>
	<p>構文の中で相互に排他的な選択肢を区切る。</p> <p>例：</p> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>

---

表記法	適用
...	コマンドラインで以下のいずれかを示す。 <ul style="list-style-type: none"><li>■ 引数を複数回繰り返すことができる。</li><li>■ 任意指定の引数が省略されている。</li><li>■ パラメータや値などの情報を追加入力できる。</li></ul>
.	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 RMI について

以下の節では、WebLogic RMI の機能について概説します。

- WebLogic RMI とは
- WebLogic RMI の機能

## WebLogic RMI とは

Remote Method Invocation (RMI) は、Java を使用した分散オブジェクト コンピューティングのための標準仕様です。RMI を使用すると、アプリケーションはネットワーク内の別の場所に存在するオブジェクトを参照し、そのオブジェクトのメソッドを、そのオブジェクトがあたかもクライアントの仮想マシンにローカルに存在するかのように呼び出すことができます。RMI は、分散 Java アプリケーションがどのように複数の Java 仮想マシン (JVM) で動作するかを指定するものです。

WebLogic は、JavaSoft RMI 仕様を実装しています。WebLogic RMI は、標準ベースの分散コンピューティングを実現します。WebLogic Server を使用すると、高速かつ信頼性の高い大規模なネットワーク コンピューティングが可能になります。さらに WebLogic RMI を使用すると、製品、サービス、リソースをネットワーク内のどこにでも配置しつつ、プログラマとエンド ユーザにはそれらをローカル環境の一部であるかのように見せることができます。

WebLogic RMI は直線的なスケーラビリティをサポートするだけでなく、コンフィグレーションされた数のサーバ スレッドに実行要求を分割できます。複数のサーバ スレッドの使用により、WebLogic Server はレイテンシと空いたプロセスを活用できます。

WebLogic RMI は、標準に完全に準拠しています。RMI の他の実装を使用する場合は、`import` 文を変更するだけでプログラムを変換できます。JavaSoft の RMI の参照実装と WebLogic の RMI 製品には相違がありますが、これらの相違は開発者からはまったく見えません。

また、WebLogic RMI は WebLogic Java Naming and Directory Interface (JNDI) と完全に統合されています。WebLogic RMI では JNDI API またはレジストリ インタフェースを使用して、アプリケーションを意味のあるネーム スペースに分けることができます。

このマニュアルは WebLogic RMI の使い方について説明したのですが、リモート オブジェクトや分散アプリケーションの記述方法についての初心者向け チュートリアルではありません。RMI について知りたい場合は、JavaSoft Web サイトの「RMI tutorial」を参照してください。

# WebLogic RMI の機能

JavaSoft の RMI 参照実装と同じように、WebLogic RMI は複数の Java 仮想マシン (JVM) での透過的なリモート呼び出しを提供しています。RMI 仕様書に書かれているリモート インタフェースおよび実装は、変更を加えずに WebLogic RMI で使用できます。

次の表に、RMI の WebLogic 実装の重要な機能を示します。

表 1-1 WebLogic RMI のパフォーマンス

機能	WebLogic RMI
全体的なパフォーマンス	WebLogic Server フレームワークとの統合により、WebLogic RMI のパフォーマンスが向上する。WebLogic Server は、通信の基本サポート、スレッドとソケットの管理、効率的なガベージコレクション、およびサーバ関連サポートを提供する。

表 1-1 WebLogic RMI のパフォーマンス (続き)

機能	WebLogic RMI
スケーラビリティ	直線的なスケーラビリティをサポートする。JavaSort RMI に比べてスケーラビリティが極めて高い。PC クラスの比較的小規模な単一プロセッサのサーバであっても、1000 を超える RMI クライアントを同時にサポートできる (サーバの負荷とメソッド呼び出しの複雑さによって異なる)。
スレッドとソケットの管理	WebLogic RMI のクライアントからネットワークへのトラフィック用に、非同期、双方向の単一接続が使用される。この接続で、WebLogic JDBC 要求などのサービスもサポートできる。
シリアライゼーション	高性能なシリアライズにより、リモート クラスを 1 回しか使用しない場合でも大幅なパフォーマンス向上が実現される。
共存オブジェクトの解決	同じ場所にあるオブジェクトがリモートとして定義されている場合のパフォーマンスの低下がない。同じ場所にある「リモート」オブジェクトへの参照は、実装オブジェクトへの直接参照として解決される。
サービスをサポートするプロセス	WebLogic RMI レジストリは RMI レジストリプロセスに取って代わるものである。WebLogic RMI は WebLogic Server 内で稼動する。他のプロセスを追加する必要はない。

表 1-2 WebLogic RMI の使いやすさ

機能	WebLogic RMI
rmic	スタブとスケルトンは実行時に WebLogic RMI によって動的に生成される。このため、クラスタ対応クライアントまたは Internet Inter-ORB Protocol (IIOP) クライアントを除いて、明示的に weblogic.rmic を実行する必要はない。

表 1-2 WebLogic RMI の使いやすさ (続き)

機能	WebLogic RMI
簡単に使用できる拡張機能	リモート インタフェースとコード生成のための、簡単に使用できる拡張機能を備えている。たとえば、インタフェースの各メソッドは、 <code>throws</code> ブロック内で <code>java.rmi.RemoteException</code> を宣言する必要がない。アプリケーションが送出する例外は、そのアプリケーションに固有なものでもよく、 <code>RuntimeException</code> を拡張することもできる。
プロキシ	リモート オブジェクトのクライアントが使用するクラス。RMI の場合はスケルトン クラスとスタブ クラスが使用される。スタブ クラスは、クライアントの Java 仮想マシン (JVM) で呼び出されるインスタンス。スケルトン クラスはリモート JVM に存在し、リモート JVM 上で呼び出されたメソッドと引数のマーシャリングを解除し、リモート オブジェクトのインスタンスのメソッドを呼び出した後、結果をマーシャリングしてクライアントに返す。
セキュリティ マネージャ	セキュリティ マネージャは不要。WebLogic Server で提供されるすべての WebLogic RMI サービスは、セキュア ソケット レイヤ (SSL) やアクセス制御リスト (ACL) などのより高度なセキュリティ オプションを備えている。RMI コードを WebLogic RMI コードに変換するときに、 <code>setSecurityManager()</code> への呼び出しをコメントアウトできる。

表 1-2 WebLogic RMI の使いやすさ (続き)

機能	WebLogic RMI
継承	UnicastRemoteObject を拡張する必要がないため、論理オブジェクト階層が保持される。リモート クラスは、 <code>rmi.server</code> パッケージ実装を継承するために <code>UnicastRemoteObject</code> を継承する必要がない。リモート クラスにアプリケーション階層内のクラスを継承させつつ、 <code>rmi.server</code> パッケージの動作を保持できる。
ツールと管理	RMI レジストリをホストする <code>WebLogic Server</code> は、分散アプリケーションを開発およびデプロイメントするのに十分なツールを備えた環境を提供する。

表 1-3 WebLogic RMI のネーミングとルックアップ

機能	WebLogic RMI
ネーミング	WebLogic JNDI と完全に統合されている。JNDI API またはレジストリ インタフェースを使用して、アプリケーションを意味のあるネーム スペースに分けることができる。JNDI を使用すると、 <code>Lightweight Directory Access Protocol (LDAP)</code> や <code>Novell NDS</code> のようなエンタープライズネーミング サービスを介して RMI オブジェクトをパブリック化できる。
ルックアップ	URL では、標準の <code>rmi://</code> 方式、 <code>https://</code> 、 <code>iiop://</code> 、または <code>http://</code> を使用する。 <code>http://</code> は、WebLogic RMI の HTTP リクエストをトンネリングし、ファイアウォールを通る場合でも WebLogic RMI のリモート呼び出しを使用可能にする。

表 1-3 WebLogic RMI のネーミングとルックアップ (続き)

機能	WebLogic RMI
クライアントサイドの呼び出し	<p>クライアントからサーバ、クライアントからクライアント、またはサーバからクライアントへの呼び出しをサポートする。クライアントとサーバが最適化済み、多重化、非同期、双方向の接続で接続されている適切な WebLogic Server 環境で動作する。このため、クライアント アプリケーションはレジストリを介してオブジェクトを発行でき、他のクライアントまたはサーバは、クライアント常駐オブジェクトをサーバ常駐オブジェクトとして使用できる。</p>

## 2 WebLogic RMI の機能とガイドライン

以下の節では、WebLogic Server で使用するための RMI のプログラミングで用いる WebLogic RMI の機能について説明します。

- WebLogic RMI フレームワーク
- WebLogic RMI コンパイラ
- RMI の動的プロキシ
- ホット コード生成
- RMI と T3 プロトコル

### WebLogic RMI フレームワーク

WebLogic RMI はクライアントとサーバのフレームワークに分けられています。実行時のクライアントはサーバのソケットを持たないため、接続をリスンしていません。クライアントはサーバを介して接続を取得します。サーバだけがクライアントのソケットを認識します。このため、クライアントにあるリモートオブジェクトをホストする場合、クライアントは WebLogic Server に接続していなければなりません。WebLogic Server はクライアントへのリクエストを処理して、クライアントへ情報を渡します。つまり、クライアントサイドの RMI オブジェクトには、クラスタ内であっても単一の WebLogic Server を介してのみアクセスできます。クライアントサイドの RMI オブジェクトが JNDI ネーミング サービスにバインドされている場合、そのオブジェクトへのアクセスは、そのバインドを実行したサーバにアクセスできる場合に限り可能です。

# WebLogic RMI コンパイラ

WebLogic RMI コンパイラ (`weblogic.rmic`) は、リモート オブジェクトを生成しコンパイルするためのコマンド ライン ユーティリティです。`weblogic.rmic` では、クライアントサイドでカスタム リモート オブジェクト インタフェースのための動的プロキシを生成し、サーバサイド オブジェクトにホット コード生成を提供します。

動的プロキシ クラスは、クライアントに渡されるシリアライズ可能クラスです。クライアントは、**WebLogic RMI** レジストリでクラスをルックアップすることによって、そのクラスのプロキシを取得します。クライアントは、あたかもローカルなクラスであるかのようにプロキシのメソッドを呼び出します。プロキシは、要求をシリアライズして **WebLogic Server** に送ります。

ホット コード生成では、クライアント上の動的プロキシからの要求を処理する、サーバサイドのクラスのバイト コードを作成します。動的に生成されたバイト コードは、クライアント要求をデシリアライズし、実装クラスに対して実行します。次に結果をシリアライズしてクライアントのプロキシへ送り返します。クラスの実装は、**Weblogic Server** の **WebLogic RMI** レジストリ内の名前に関連付けられます。

RMI オブジェクトがデプロイされている場合、`weblogic.rmic` を実行すると、プロキシ クラスが自動的に作成され、実行時にホット コード生成機能によってバイトコードが動的に生成されます。`weblogic.rmic` の使い方については、「手順 4. 実装クラスを RMI コンパイラでコンパイルする」を参照してください。`weblogic.rmic` のコマンドライン オプションの一覧については、「**WebLogic RMI** コンパイラのオプション」を参照してください。

**注意：** クラスタ対応クライアントまたは IIOP クライアントの場合は、明示的に `weblogic.rmic` を実行するだけです (**WebLogic RMI over IIOP** を使用すると、クライアントは、**Internet Inter-ORB Protocol (IIOP)** を介して **RMI** リモート オブジェクトにアクセスできるため、**RMI** プログラミング モデルが拡張されます)。**RMI over IIOP** の使用方法の詳細については、『**WebLogic RMI over IIOP プログラマーズ ガイド**』を参照してください。

## スタブとスケルトンから動的プロキシとバイトコードへの移行

WebLogic Server 6.1 より前のバージョンでは、`weblogic.rmic` を実行すると、クライアントにはスタブが、サーバサイドにはスケルトンコードが生成されました。今回のバージョンでは、クライアントサイドで生成されていたスタブは動的プロキシに、サーバサイドのスケルトンはバイトコードに置き換えられています。これにより、クラスの生成が不要になりました。

バージョン 6.1 より前の WebLogic RMI オブジェクトをバージョン 6.1 以降の WebLogic Server で実行できるようにするには、それらのオブジェクトに対して `weblogic.rmic` を再度実行します。これにより、必要なプロキシおよびバイトコードが生成され、デプロイ済みの RMI オブジェクトが有効になります。動的プロキシの詳細については、「RMI の動的プロキシ」を参照してください。

リモート オブジェクトが EJB の場合は、`weblogic.ejbrc` を再度実行すると、バージョン 6.1 より前の WebLogic Server オブジェクトがバージョン 6.1 以降で動作するようになります。`weblogic.ejbrc` の使用方法の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。

リモート オブジェクトに対して、`-oneway`、`-clusterable`、`-stickToFirstServer` のうちの 1 つまたは複数のパラメータを使用して `weblogic.rmic` を再度実行するか、または `weblogic.ejbrc` を再度実行すると、そのオブジェクトのデプロイメント記述子が生成されます。

## WebLogic RMI コンパイラのオプション

WebLogic RMI コンパイラは、Java コンパイラがサポートしているオプションをすべて受け入れます。たとえば、コマンドラインのコンパイラ オプションに `-d \classes examples.hello.HelloImpl` を追加できます。ほかにも、Java コンパイラがサポートしているすべてのオプションを使用でき、これらのオプションは直接 Java コンパイラに渡されます。

次の表に、`java weblogic.rmic` オプションを示します。これらのオプションは、`java weblogic.rmic` の後、リモート クラス名の前に入力します。

表 2-1 WebLogic RMI コンパイラのオプション

オプション	説明
-help	オプションの説明を表示する。
-version	バージョン情報を出力する。
-dispatchPolicy <queueName>	サービスが <b>WebLogic Server</b> の実行スレッドを取得するために使う、コンフィグレーション済みの実行キューを指定する。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。
-idl	リモート インタフェース用の <b>IDL</b> を生成する。
-idlOverwrite	既存の <b>IDL</b> ファイルを上書きする。
-idlVerbose	<b>IDL</b> 情報についての冗長な情報を表示する。
-idlStrict	<b>OMG</b> 規格に従って <b>IDL</b> を生成する。
-idlNoFactories	値型 (value type) のファクトリ メソッドを生成しない。
-idlDirectory <idlDirectory>	<b>IDL</b> ファイルを作成するディレクトリを指定する (デフォルトはカレント ディレクトリ)。
-clusterable	そのサービスをクラスタ化可能 ( <b>WebLogic Server</b> クラスタ内の複数のサーバがホストできる) として指定する。各ホスティング オブジェクト、またはレプリカは、共通名でネーミング サービスにバインドされる。そのサービス スタブがネーミング サービスから取り出される場合、それはレプリカのリストを保持するレプリカ対応参照を含み、レプリカ間のロード バランシングやフェイルオーバーを行う。
-loadAlgorithm <algorithm>	-clusterable と組み合わせた場合だけ使用可能。ロード バランシングおよびフェイルオーバーに使用する特定のアルゴリズムのサービスを指定する (デフォルトは <code>weblogic.cluster.loadAlgorithm</code> )。ラウンドロビン、ランダム、重みベースの中から 1 つ選択できる。

オプション	説明
-callRouter <callRouterClass>	-clusterable と組み合わせた場合だけ使用可能。 ルーティング メソッド呼び出し用に使われるクラスを指定する。このクラスは <code>weblogic.rmi.cluster.CallRouter</code> を実装する必要がある。指定した場合、各メソッド呼び出しの前にそのクラスのインスタンスが呼び出され、そのメソッド パラメータに基づいてルーティングするサーバを指定できる。サーバ名または <code>null</code> が返される。 <code>null</code> は現在のロード アルゴリズムが使用されることを示す。
-stickToFirstServer	-clusterable と組み合わせた場合だけ使用可能。 セッション維持型ロード バランシングを有効にする。最初の要求をサービスするために選ばれたサーバが、後続のすべての要求にも使用される。
-methodsAreIdempotent	-clusterable と組み合わせた場合だけ使用可能。 このクラスのメソッドが多重呼び出し不変であることを示す。これにより、リモート メソッドが呼び出される前に起きた障害かどうか保証できなくても、通信障害があれば、スタブはその復旧を試みることができるようになる。デフォルトでは（このオプションが使われなければ）、スタブはリモート メソッドが呼び出された前に起きたことが保証されている障害に関してだけ再試行する。
-replicaListRefreshInterval <seconds>	-clusterable と組み合わせた場合だけ使用可能。 クラスタのレプリカ リストをリフレッシュする試みの最小間隔を指定する（デフォルトは 180 秒）。
-iiop	サーバから IIOP スタブを生成する。
-iiopDirectory	IIOP プロキシ クラスが作成されるディレクトリを指定する。
-commentary	注釈を出力する。
-nomanglednames	コンパイル時にリモート クラスに固有のプロキシが作成される。

オプション	説明
-keepgenerated	WebLogic RMI コンパイラを実行するときに、生成したスタブ クラス ファイルとスケルトン クラス ファイルのソースを保持できる。

## クラスタ内のレプリケーションされないスタブ

weblogic.rmic を使って、レプリケートされないスタブをクラスタ内に生成することもできます。このようなスタブは、「固定」サービスと呼ばれています。これらのスタブは登録されたホストからのみ使用可能であり、透過的なフェイルオーバーやロード バランシングは提供しません。固定サービスはレプリケートされたクラスタ全体の JNDI ツリーにバインドされるので、クラスタ全体で使用可能になります。固定サービスをホストする各サーバが故障しても、クライアントは別のサーバにフェイルオーバーすることはできません。

## WebLogic RMI コンパイラのその他の機能

WebLogic RMI コンパイラには、他にも以下のような機能があります。

- リモート メソッドのシグネチャは `RemoteException` を送出する必要がありません。
- リモートの例外は `RuntimeException` にマッピングできます。
- リモート クラスは、非リモート インタフェースも実装できます。

## RMI の動的プロキシ

動的プロキシまたはプロキシは、リモート オブジェクトのクライアントが使用するクラスです。このクラスは、作成されるときに、実行時に指定されたインタフェースのリストを実装します。

RMI では、動的に生成されたバイトコードとプロキシクラスが使用されます。プロキシクラスは、クライアントの Java 仮想マシン (JVM) で呼び出されるインスタンスです。プロキシクラスは、呼び出されたメソッド名とその引数をマーシャリングして、リモートの JVM に転送します。リモート呼び出しが終了して返された後に、プロキシクラスはクライアント上でその結果のマーシャリングを解除します。生成されたバイトコードはリモート JVM に存在し、リモート JVM 上で呼び出されたメソッドと引数のマーシャリングを解除し、リモートオブジェクトのインスタンスのメソッドを呼び出した後、結果をマーシャリングしてクライアントに戻します。

## WebLogic RMI コンパイラとプロキシの使い方

WebLogic RMI コンパイラは、デフォルトの動作により、リモート インタフェース用のプロキシと、そのプロキシを共有するリモート クラス用のプロキシを作成します。プロキシは、リモート オブジェクトのクライアントが使用するクラスです。RMI では、動的に生成されたバイトコードとプロキシクラスが使用されます。

たとえば、WebLogic RMI コンパイラでは、`example.hello.HelloImpl` と `counter.example.CiaoImpl` は、一対のプロキシクラスとバイトコード、つまりリモート オブジェクト (このサンプルでは `example.hello.Hello`) によって実装されたリモート インタフェースに適合するプロキシで表わされます。

リモート オブジェクトが複数のインタフェースを実装する場合、プロキシの名前とパッケージは 1 組のインタフェースをエンコードすることによって決定されます。WebLogic RMI コンパイラの `-nomanglednames` というオプションを使って、デフォルトの動作をオーバーライドできます。このオプションを使用すると、コンパイル時にリモート クラスに固有のプロキシが作成されます。クラス固有のプロキシが検出された場合は、そのプロキシはインタフェース固有のプロキシに優先します。

さらに、WebLogic RMI のプロキシクラスでは、プロキシは `final` ではありません。同じ場所に配置されたリモート オブジェクトへの参照は、プロキシではなくオブジェクトそのものへの参照です。

# ホット コード 生成

`rmic` を実行すると、**WebLogic Server** のホット コード生成機能により、メモリ内にサーバクラス用のバイトコードが自動生成されます。バイトコードは、リモート オブジェクトの必要に応じて、動的に生成されます。現在のバージョンの **WebLogic Server** では、`weblogic.rmic` を実行しても、オブジェクトのスケルトンクラスは生成されません。

# RMI と T3 プロトコル

**WebLogic Server** の RMI 通信では T3 プロトコルが使用されます。このプロトコルは、**WebLogic Server** と他の Java プログラム (クライアントやその他の **WebLogic Server** を含む) との間のデータ転送用に最適化されたプロトコルです。サーバ インスタンスは、接続先の各 Java 仮想マシン (JVM) を追跡し、JVM のすべてのトラフィックを転送するための T3 接続を作成します。

たとえば、Java クライアントが **WebLogic Server** 上のエンタープライズ Bean および JDBC 接続プールにアクセスすると、1 つのネットワーク接続が **WebLogic Server** の JVM とクライアントの JVM との間に確立されます。T3 プロトコルは 1 つの接続上のパケットを見えない形で多重化するため、EJB および JDBC のサービスでは、専用のネットワーク接続を単独で使用しているかのように記述することができます。

2 つのサーバ インスタンスや 1 つのサーバ インスタンスと 1 つの Java クライアントなど、T3 接続された任意の 2 つの Java プログラムは、定期的にポイントツーポイントの「ハートビート」を使用して、使用可能な状態であることを通知および判定します。各エンド ポイントは定期的にハートビートをピアに送るとともに、ピアから継続してハートビートを受け取ることでピアが使用可能な状態であると判定します。

サーバ インスタンスがハートビートを送る頻度は、ハートビート間隔 (デフォルトでは 60 秒) で決まります。

ピアからのハートビートがある回数届かなないと、サーバ インスタンスはピアが使用不可の状態であると判定します。この回数は、ハートビート周期 (デフォルトは 4) によって決まります。

したがって、各サーバ インスタンスは、ピアをアクセス不能であると判定するまでに、ピアからメッセージ (ハートビートまたは他の通信) のない状態で最大 240 秒 (4 分) 待ちます。

タイムアウトのデフォルト値は変更しないことをお勧めします。



## 3 WebLogic RMI の実装

以下の節では、WebLogic RMI API について説明します。

- WebLogic RMI API の概要
- WebLogic RMI の実装の手順

### WebLogic RMI API の概要

WebLogic RMI の一部として、いくつかのパッケージが WebLogic Server に付属しています。パブリック API には以下のものが含まれています。

- RMI 基本クラスの WebLogic 実装
- レジストリ
- サーバのパッケージ
- WebLogic RMI コンパイラ
- パブリック API に含まれないサポート クラス

既に RMI クラスを記述している場合は、リモート インタフェースとそれを拡張するクラスで `import` 文を変えるだけで WebLogic RMI をインストールできます。クライアント アプリケーションにリモート呼び出しを追加するには、レジストリ内でオブジェクトを名前でもックアップします。

すべてのリモート オブジェクトの基本単位は `java.rmi.Remote` インタフェースで、これにはメソッドが含まれていません。この「タグ付け」インタフェースを拡張し（リモート クラスを識別するタグとして機能するように）、リモート オブジェクトの構造を作成するメソッド スタブを使って、使用するリモート インタ

フェースを作成します。次に、リモート クラスを使ってリモート インタフェースを実装します。この実装はレジストリ内の名前にバインドされ、クライアントやサーバはそこからオブジェクトをルックアップしてリモートで使用できます。

JavaSoft の RMI の参照実装の場合と同様に、`java.rmi.Naming` クラスは重要なクラスです。このクラスには、レジストリ内のリモート オブジェクトに名前をバインド、アンバインド、リバインドするメソッドが含まれています。また、クライアントからレジストリ内の名前付きリモート オブジェクトにアクセスするための `lookup()` メソッドも含まれています。

さらに、WebLogic JNDI はネーミング サービスとルックアップ サービスを提供します。WebLogic RMI は、JNDI によるネーミングとルックアップもサポートしています。

WebLogic RMI 例外は `java.rmi` 例外と同じ機能を備えているので、既存のインタフェースと実装で例外の処理方法を変える必要がありません。

## WebLogic RMI の実装の手順

以下の節では WebLogic Server RMI の実装方法について説明します。

- リモートで呼び出すことができるクラスを作成する
  - 手順 1. リモート インタフェースを作成する
  - 手順 2. リモート インタフェースを実装する
  - 手順 3. Java クラスをコンパイルする
  - 手順 4. 実装クラスを RMI コンパイラでコンパイルする
  - 手順 5. リモート メソッドを呼び出すコードを記述する
- 完全なコード例

# リモートで呼び出すことができるクラスを作成する

独自の WebLogic RMI クラスを簡単な手順で記述できます。次に、その単純な例を示します。

## 手順 1. リモート インタフェースを作成する

リモートで呼び出せるすべてのクラスは、リモート インタフェースを実装します。Java コードのテキスト エディタを使用し、以下のガイドラインに従ってリモート インタフェースを記述します。

- リモート インタフェースは、`java.rmi.Remote` インタフェースを拡張しなければなりません。これにはメソッド シグネチャが含まれていません。インタフェースを実装する各リモート クラスで実装されるメソッド シグネチャを含めます。インタフェースの作成方法については、Sun Microsystems JavaSoft チュートリアル「[Creating Interfaces](#)」を参照してください。
- リモート インタフェースはパブリックでなければなりません。パブリックでない場合、クライアントはリモート インタフェースを実装するリモート オブジェクトをロードしようとするエラーを受け取ります。
- JavaSoft の RMI とは異なり、インタフェース内の各メソッドがその `throws` ブロックで `java.rmi.RemoteException` を宣言する必要はありません。アプリケーションが送出する例外は、そのアプリケーションに固有なものでもよく、`RuntimeException` を拡張することも可能です。WebLogic RMI は、サブクラス `java.rmi.RemoteException` を作成するため、既存の RMI クラスがある場合は、例外処理を変更する必要がありません。
- リモート インタフェースにはコードがあまり記述されていない場合もあります。必要なのは、リモート クラスで実装するメソッドに対するメソッド シグネチャだけです。

以下の例に、メソッド シグネチャ `sayHello()` が含まれたリモート インタフェースを示します。

```
package examples.rmi.multihello;
import java.rmi.*;
public interface Hello extends java.rmi.Remote {
    String sayHello() throws RemoteException;
```

```
}
```

JavaSoft の RMI では、リモート インタフェースを実装するすべてのクラスには、コンパイル済みのプロキシが存在しなければなりません。WebLogic RMI は、柔軟性の高い実行時のコード生成をサポートします。つまり、WebLogic RMI は、型さえ正しければ、そのほかはインタフェースを実装するクラスに依存しない動的プロキシと動的に生成されるバイトコードもサポートします。クラスが 1 つのリモート インタフェースを実装する場合、コンパイラが生成したプロキシとバイトコードは、リモート インタフェースと同じ名前になります。クラスが複数のリモート インタフェースを実装する場合、コード生成の結果できるプロキシとバイトコードの名前は、コンパイラが使う名前の区切り方によって変わります。

## 手順 2. リモート インタフェースを実装する

Java コードのテキスト エディタを使用して、リモートで呼び出されるクラスを記述します。このクラスは、手順 1 で記述したリモート インタフェースを実装する必要があります。これは、インタフェースに含まれるメソッド シグネチャを実装したことを意味します。現在のリリースでは、WebLogic RMI で行われるコード生成はこのクラスファイルに依存します。

WebLogic RMI では、クラスは JavaSoft RMI で必須の `UnicastRemoteObject` を拡張する必要はありません (`UnicastRemoteObject` を拡張することはできますが、必須ではありません)。このため、アプリケーションに適用しやすいクラス階層を維持できます。

クラスは、複数のリモート インタフェースを実装できます。また、クラスにはリモート インタフェースにないメソッドも定義できますが、このようなメソッドはリモートで呼び出せません。

次の例では、複数の `HelloImpls` を作成するクラスを実装し、それぞれをレジストリ内の固有の名前にバインドします。メソッド `sayHello()` は、ユーザに「Hello」とあいさつし、リモートで呼び出されたオブジェクトを識別します。

```
package examples.rmi.multihello;
import java.rmi.*;
public class HelloImpl implements Hello {
    private String name;
```

```
public HelloImpl(String s) throws RemoteException {
    name = s;
}

public String sayHello() throws RemoteException {
    return "Hello!From " + name;
}
```

次に、リモート オブジェクトのインスタンスを作成する main() メソッドを記述し、それを名前 (オブジェクトの実装を指し示す URL) にバインドすることによって **WebLogic RMI** のレジストリに登録します。リモートでオブジェクトを使用するためにプロキシを取得しようとするクライアントは、オブジェクトを名前で見つけ出すことができます。

次に、HelloImpl クラス用の main() メソッドの例を示します。この例では、**WebLogic Server** レジストリに HelloRemoteWorld という名前で HelloImpl オブジェクトに登録します。

```
public static void main(String[] argv) {
    // 以下のコードは WebLogic RMI では不要
    // System.setSecurityManager(new RmiSecurityManager());
    // しかし、このコードを含める場合には、以下のように
    // 条件文にする必要がある
    // if (System.getSecurityManager() == null)
    // System.setSecurityManager(new RmiSecurityManager());
    int i = 0;
    try {
        for (i = 0; i < 10; i++) {
            HelloImpl obj = new HelloImpl("MultiHelloServer" + i);
            Context.rebind("//localhost/MultiHelloServer" + i, obj);
            System.out.println("MultiHelloServer" + i + " created.");
        }
        System.out.println("Created and registered " + i +
            " MultiHelloImpls.");
    }
}
```

```
catch (Exception e) {  
    System.out.println("HelloImpl error:" + e.getMessage());  
    e.printStackTrace();  
}  
}
```

WebLogic RMI では、アプリケーションにセキュリティを統合するためにセキュリティ マネージャを設定する必要はありません。セキュリティは、WebLogic の SSL と ACL のサポートによって処理されます。必要な場合は独自のセキュリティ マネージャを使うこともできますが、それを WebLogic Server にインストールしないでください。

## 手順 3. Java クラスをコンパイルする

javac または他の Java コンパイラを使用して .java ファイルをコンパイルし、リモート インタフェースとそれを実装するクラス用の .class ファイルを作成します。

## 手順 4. 実装クラスを RMI コンパイラでコンパイルする

リモート クラスに対して WebLogic RMI コンパイラ (`weblogic.rmic`) を実行し、動的プロキシとバイトコードを動的に生成します。プロキシは、リモート オブジェクト用のクライアントサイド プロキシで、個々の WebLogic RMI 呼び出しを対応するサーバサイド バイトコードに転送します。サーバサイド バイトコードは、その呼び出しを実際のリモート オブジェクトの実装に転送します。`weblogic.rmic` を実行するには、次のコマンド パターンを使用します。

```
$ java weblogic.rmic nameOfRemoteClass
```

`nameOfRemoteClass` は、リモート インタフェースを実装するクラスの完全なパッケージ名です。上で使用した例では、このコマンドは次のようになります。

```
$ java weblogic.rmic examples.rmi.hello>HelloImpl
```

生成したプロキシとバイトコードのソースを保持する場合は、`weblogic.rmic` を実行するときに `-keepgenerated` フラグを設定します。使用可能なコマンドライン オプションの一覧については、「WebLogic RMI コンパイラのオプション」を参照してください。

`weblogic.rmic` を実行すると、プロキシとバイトコードが動的に生成されます。プロキシは、`nameOfInterface_Proxy.class` というクラスになります。作成される 3 つのファイル、つまりリモート インタフェース、リモート インタフェースを実装するクラス、およびプロキシは、`weblogic.rmic` によって生成されるバイトコードとともに、オブジェクトの `main()` メソッドのネーミング方式で利用した URL を持つ **WebLogic Server** の `CLASSPATH` の該当ディレクトリに入れる必要があります。

## 手順 5. リモート メソッドを呼び出すコードを記述する

リモート クラス、リモート クラスが実装するインタフェース、およびそのプロキシとバイトコードをコンパイルして **WebLogic Server** にインストールしたら、**Java** コードのテキスト エディタを使用し、**WebLogic** クライアント アプリケーションにコードを追加してリモート クラスのメソッドを呼び出すことができます。

通常、そのためにはコードを 1 行だけ記述します。つまり、リモート オブジェクトへの参照を取得します。これは、`Naming.lookup()` メソッドで行います。次に、上の例で作成したオブジェクトを使用する短い **WebLogic** クライアント アプリケーションの例を示します。

```
package mypackage.myclient;
import java.rmi.*;

public class HelloWorld throws Exception {

    // WebLogic のレジストリ内の
    // リモート オブジェクトをルックアップする
    Hello hi = (Hello)Naming.lookup("HelloRemoteWorld");
    // メソッドをリモートで呼び出す
    String message = hi.sayHello();
    System.out.println(message);
}
```

この例では、クライアントとして **Java** アプリケーションを使用しています。

## 完全なコード例

次に、Hello インタフェースの完全なコードを示します。

```
package examples.rmi.hello;
import java.rmi.*;

public interface Hello extends java.rmi.Remote {

    String sayHello() throws RemoteException;

}
```

次に、このインタフェースを実装する HelloImpl クラスの完全なコードを示します。

```
package examples.rmi.hello;

import java.rmi.*;

public class HelloImpl
    // 以下はWebLogic RMI では不要
    // extends UnicastRemoteObject
    implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello() throws RemoteException {
        return "Hello Remote World!!";
    }
}
```

```
public static void main(String[] argv) {
    try {
        HelloImpl obj = new HelloImpl();
        Naming.bind("HelloRemoteWorld", obj);
    }
    catch (Exception e) {
        System.out.println("HelloImpl error:" + e.getMessage());
        e.printStackTrace();
    }
}
```

