BEA WebLogic Server™

WebLogic RMI over IIOP
プログラマーズ ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2003 年 4 月 29 日

著作権

限定的権利条項

商標または登録商標

WebLogic RMI over IIOP 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
	2003 年 4 月 29 日	BEA WebLogic Server バージョン 6.1

目次

このマニュアルの内容

1. WebLogic RMI over IIOP の使い方

このマニュアルの内容

このマニュアルでは、Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) について解説し、さまざまなクライアントの種類に応じた RMI over IIOP アプリケーションの作成方法について説明します。BEA WebLogic Server 環境で IIOP を使用して RMI リモート オブジェクトにアクセスする機能をクライアントに提供することにより、RMI プログラミング モデルを拡張する方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic RMI over IIOP の使い方」では、RMI over IIOP の概要、RMI over IIOP アプリケーションの開発方法、および WebLogic Server のコンフィグレーション方法について説明します。これらのタスクを説明するコードも示されています。

対象読者

このマニュアルは主に、Internet Inter-ORB Protocol（IIOP）を使用して Remote Method Invocation（RMI）リモート オブジェクトにアクセスする機能をクライアントに提供することに関心があるアプリケーション開発者を対象としています。この機能により、RMI と Common Object Request Broker Architecture (CORBA) の相互運用性が高まります。このマニュアルは、WebLogic Server プラットフォーム、CORBA、および Java プログラミング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] をクリックするか、または「e-docs」という製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/）を直接表示してください。

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 ファイルずつ印刷できます。

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

Adobe Acrobat Reader がない場合は、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

以下の WebLogic Server のマニュアルには、RMI over IIOP の使い方に関する情報が含まれています。

RMI over IIOP の一般情報については、以下のソースを参照してください。

- OMG Web サイト（http://www.omg.org/）
- Sun Microsystems, Inc. の Java サイト（http://java.sun.com/）

CORBA と分散オブジェクト コンピューティング、トランザクション処理、および Java の詳細については、http://edocs.beasys.co.jp/e-docs/ の参考文献を参照してください。

サポート情報

BEA WebLogic Server のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。寄せられた意見については、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。

電子メールのメッセージには、BEA WebLogic Server 6.1 リリースのドキュメントをご使用の旨をお書き添えください。

本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（**www.bea.com**）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	用語集で定義されている用語を示す。
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

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:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な箇所を示す。 例: `void` **`commit`** `( )`
斜体の等幅テキスト	コード内の変数を示す。 例: `String` *`expr`*

表記法	適用
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	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。 実際には、この省略符号は入力しない。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
. . .	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。 実際には、この省略符号は入力しない。

1 WebLogic RMI over IIOP の使い方

以下の節では、RMI over IIOP の機能について説明します。

- 概要
- RMI over IIOP の概要
- RMI-IIOP プログラミング モデル
- RMI クライアントを使用した RMI over IIOP
- IDL クライアントを使用した RMI over IIOP
- Tuxedo と Tuxedo クライアントを使用した RMI-IIOP
- WebLogic Server の RMI-IIOP 用コンフィグレーション
- 考慮事項
- コード例
- その他の情報源

概要

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。これによって、RMI リモート オブジェクトが Common Object Request Broker Architecture（CORBA）クライアントの新しいクラスに公開されます。CORBA クライアントは、C++ を含むさまざまな言語で記述でき、リモート オブジェクトとの対話には Interface-Definition-Language（IDL）を使用します。RMI-IIOP の WebLogic Server 6.1 実装はかなり改良されており、以下の機能を備えています。

- 標準 IIOP プロトコルを使用して Java RMI クライアントに接続できます。
- CORBA/IDL クライアント（C++ で記述されたものも含む）に接続できます。
- Tuxedo サーバと相互運用できます。
- WebLogic Server にホストされている EJB に、さまざまなクライアントを接続できます。

開発者の間では、CORBA/IDL クライアント（以降では「IDL クライアント」と呼びます）から J2EE サービスにアクセスできるようにすることに対して強い要望があります。RMI は EJB に対して有効な技術なので、RMI over IIOP を使用すれば、さまざまなクライアントをサポートする機能が拡張されます。ただし、Java と CORBA は全く異なるオブジェクト モデルに基づいています。このため、この 2 つのプログラミング パラダイムに基づいて作成されたオブジェクト間でのデータ共有は、最近までは、Remote と CORBA のプリミティブ データ型だけに限られていました。CORBA の構造体も Java のオブジェクトも、異なるオブジェクト間で簡単にやり取りすることはできませんでした。その結果、Objects-by-Value 仕様が Object Management Group（OMG）によって作成されました。この仕様では、Java オブジェクト モデルを CORBA/IDL プログラミング モデルにエクスポートでき、Java、CORBA の 2 つのモデル間での複合データ型の交換が可能になります。Objects-by-Value 仕様が正しく実装されている CORBA ORB を使用すれば、WebLogic Server でも Objects-by-Value をサポートできます。

標準 IIOP プロトコルを活用して、RMI-IIOP を Java/RMI クライアントで使用することも可能です。JDK のリリース 1.3.1 ではこの機能がかなり拡充され、Java-to-Java の環境においても RMI-IIOP で問題なく WebLogic Server 6.1 を使用できます。

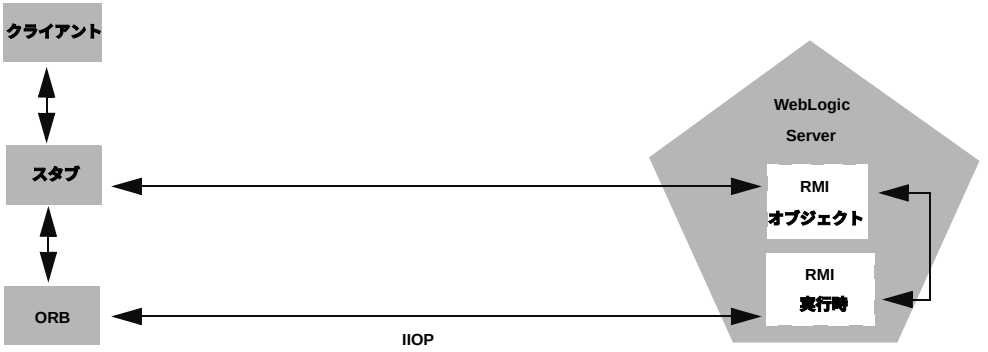
また、WebLogic Server 6.1 は Weblogic Tuxedo Connector の実装も備えています。この基盤技術を利用すると、Tuxedo サーバとの相互運用が可能になります。この機能を使用すると、Tuxedo を ORB として活用したり、従来の Tuxedo システムと統合したりできます。

このマニュアルでは、さまざまなクライアントの向けの RMI over IIOP アプリケーションを作成する方法について説明します。独自の RMI-IIOP アプリケーションをどのように開発するかは、どのサービスやクライアントを統合するかによって異なります。詳細については、以降の節を参照してください。

RMI over IIOP の概要

RMI over IIOP は、RMI プログラミング モデルによるアプリケーションです。このプログラミングでは、JNDI および RMI タイプのシステムが使用されます。WebLogic RMI に関するより一般的な情報については『WebLogic RMI プログラマーズ ガイド』を、JNDI については『WebLogic JNDI プログラマーズ ガイド』を参照してください。これらの技術はどちらも RMI-IIOP には不可欠です。RMI-IIOP アプリケーションの構築を始める前に、これらの技術の一般的な概念について理解しておくことをお勧めします。

図 1-1 RMI オブジェクトの関係



RMI-IIOP プログラミング モデル

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。IIOP は、異種分散システムの相互運用が容易になるように設計された堅牢なプロトコルで、多くのベンダによってサポートされています。

RMI-IIOP を使用する場合、アプリケーションの開発時に従うべき 2 つの基本的なプログラミング モデル、RMI クライアントを使用した RMI-IIOP と IDL クライアントを使用した RMI-IIOP があります。これらのモデルの機能や概念はある程度共通しています。たとえば、どちらのモデルでも Object Request Broker（ORB）および Internet InterORB Protocol（IIOP）を使用します。使用する技術もよく似ています。ただし、異種システム間での相互運用が可能な環境を作成するためのアプローチにははっきりとした違いがあります。IIOP は分散アプリケーション用の転送プロトコルとして使用できます。この場合、インタフェースとして IDL と Java RMI のどちらで記述されたものを使用するかを選択する必要があります。プログラミングをする際には、どちらのインタフェースを使用するかを決めなければなりません。2 つを混在させることはできません。

RMI クライアントを使用した RMI over IIOP では、RMI の機能を IIOP プロトコルと組み合わせることで、完全に Java プログラミング言語で作業することを可能にしています。RMI クライアントを使用した RMI-IIOP は Java-to-Java モデルで、ORB は通常、クライアントで動作する JDK の一部となっています。RMI-IIOP では、オブジェクトは参照としても値としても渡すことができます。

IDL クライアントを使用した RMI over IIOP では、Object Request Broker（ORB）を使用するとともに、IDL と呼ばれる相互運用可能な言語を作成するコンパイラを使用します。CORBA のプログラマは、CORBA オブジェクトの定義、実装、および Java プログラミング言語からのアクセスに、CORBA インタフェース定義言語（IDL : Interface Definition Language）のインタフェースを使用できます。

詳細については、以下の OMG 仕様を参照してください。

- Java Language Mapping to OMG IDL 仕様
- CORBA/IIOP 2.4.2 仕様

BEA Tuxedo クライアントまたはサービスを WebLogic Server に統合する場合には、これら 2 つのプログラミング モデルに加えもう 1 つオプションがあります。WebLogic Server に付属する Weblogic Tuxedo Connector を使用すると、Tuxedo と WebLogic の統合がより簡単でより強力になります。Weblogic Tuxedo Connector には、その基盤となるメカニズムとして RMI-IIOP が使用されています。

RMI プログラミング モデルの選択

アプリケーションの分散環境をどのように作成するかを定義するには、さまざまな要因を考慮する必要があります。現時点では、RMI-IIOP を採用するためのモデルはさまざまで、かなりの混乱が見られます。これは、それらのモデルが同じ機能や標準を備えており、どのモデルに従うべきかの見通しがたてにくいことが原因です。この状況を整理するため、最近ではそれらのモデルが以下のように分類されています。

- RMI クライアントを使用した RMI-IIOP
- IDL クライアントを使用した RMI-IIOP
- Tuxedo クライアントを使用した RMI-IIOP

以降の節では、各プログラミング モデルの利点を概説します。まずは、基本的な RMI モデルの説明から始めます。

RMI（Remote Method Invocation）は、分散コンピューティングの Java-to-Java モデルです。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。これは、Java-to-Java パラダイムには理想的です。すべての RMI-IIOP モデルは RMI に基づいていますが、IIOP を使用しない基本的な RMI モデルに従う場合は、Java 以外の言語で記述されたクライアントを統合することはできません。詳細については、『WebLogic RMI プログラマーズ ガイド』を参照してください。

RMI クライアントを使用した RMI-IIOP は、Java および J2EE プログラミング モデルを指向する人向けで、RMI の機能が IIOP プロトコルに結合されています。アプリケーションが Java で開発されており、IIOP の利点を活用したいと考えているのであれば、RMI クライアントを使用した RMI-IIOP モデルをお勧めします。RMI-IIOP を使用すると、Java ユーザは RMI インタフェースをプログラミングし、基盤となる転送プロトコルとして IIOP を使用できます。RMI クライアントは、J2EE または J2SE コンテナによってホストされた RMI-IIOP 対応 ORB を実行します。ほとんどの場合、JDK は 1.3 以降です。WebLogic 固有のクラスは不要であり、必要なものはこのシナリオで自動的にダウンロードされます。クライアントの分散を最小にするにはよい方法です。通常の WebLogic RMI で使用する独自の t3 プロトコルを使用する必要はありません。RMI を使用した RMI-IIOP は、エンタープライズ JavaBean に接続するには特に便利です。詳細については、「RMI クライアントを使用した RMI over IIOP」を参照してください。

IDL を使用した RMI-IIOP の場合は、CORBA プログラミング モデルに従う必要があり、Java 以外のクライアントとの相互運用を可能にします。CORBA アプリケーションが既に存在する場合は、IDL クライアントを使用した RMI-IIOP モデルに従ってプログラミングする必要があります。基本的に、IDL インタフェースは Java から生成します。クライアント コードと WebLogic Server との通信は、これらの IDL インタフェースを介して行われます。これが基本的な CORBA プログラミングです。詳細については、「IDL クライアントを使用した RMI over IIOP」を参照してください。

WebLogic を既存の Tuxedo システムに統合する場合は、Weblogic Tuxedo Connector を活用することをお勧めします。Weblogic Tuxedo Connector は、Tuxedo で開発済みのクライアントやサービスを、RMI-IIOP を使用して WebLogic Server と相互運用できるようにします。詳細については、「Tuxedo と Tuxedo クライアントを使用した RMI-IIOP」を参照してください。

クライアント	クライアントの言語	プロコトル	定義	メリット
RMI	Java	t3	JavaSoft RMI 仕様に準拠したクライアント。	高速でスケーラビリティが高い。最適化された WebLogic t3 プロトコルの使用によりパフォーマンスを向上。
RMI クライアントを使用した RMI over IIOP	Java	IIOP	Objects-by-Value に対する CORBA 2.4.2 仕様のサポートを利用する RMI クライアント。この Java クライアントは、標準 RMI/JNDI モデルで開発される。	Internet-Inter-ORB-Protocol による RMI。RMI-IIOP 標準を使用。クライアントに WebLogic クラスは不要。
IDL クライアントを使用した RMI-IIOP	C++、C、Smalltalk、COBOL（OMG IDL からのマッピングがあるすべての言語）	IIOP	CORBA 2.4.2 ORB を使用する CORBA クライアント。注意：ネームスペース衝突が起こるため、Java CORBA クライアントは RMI over IIOP 仕様ではサポートされない。	WebLogic、および C++、COBOL などで記述されたクライアントとの相互運用性。

クライアント	クライアントの言語	プロコトル	定義	メリット
Tuxedo クライアントを使用した RMI-IIOP	C++、C、COBOL （Tuxedo が提供する OMG IDL からのマッピングがあるすべての言語）	TGIIOP	Tuxedo 8.0 以降で開発された Tuxedo サーバ。	WebLogic Server アプリケーションと Tuxedo サービス間の相互運用性。

RMI クライアントを使用した RMI over IIOP

RMI over IIOP アプリケーションを開発するには、以下の手順を実行する必要があります。

1. リモート インタフェースと実装クラスの開発を行い、Java コンパイラでコンパイルします。
2. `-iiop` オプションを使用して IIOP クラスの生成を行います。WebLogic RMI コンパイラで作成された IIOP スタブは、JDK 1.3 ORB で使用されることを前提としています。他の ORB を使用する場合、それぞれの ORB ベンダのマニュアルを参照して、これらのスタブが適切かどうかを判断してください。-iiopDirectory オプションを使用して、IIOP クラスを生成するディレクトリを指定します。
3. RMI クライアントの開発を行い、言語固有のコンパイラでコンパイルします。

RMI アプリケーションの開発手順の詳細については、『WebLogic RMI プログラマーズ ガイド』を参照してください。

リモート インタフェースと実装クラスの開発

RMI オブジェクトを開発するには、java.rmi.Remote を拡張するインタフェースでオブジェクトのパブリック メソッドを定義する必要があります。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。samples\examples\iiop\rmi\server\wls に格納されている Ping の例を示します。

```
public interface Pinger extends java.rmi.Remote {
public void ping() throws java.rmi.RemoteException;
public void pingRemote() throws java.rmi.RemoteException;
public void pingCallback(Pinger toPing) throws
java.rmi.RemoteException;
{
```

RMI オブジェクトでは、interfaceNameImpl というクラスにインタフェースを実装します。このクラスには、記述済みのリモート 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。すべてのコード生成はこのクラスファイルに依存します。その実装クラスを JNDI ツリーにバインドすると、クライアントで使用できるようになります。実装クラスは通常、WebLogic スタートアップ クラスとしてコンフィグレーションされ、JNDI ツリーにオブジェクトをバインドする main メソッドを含んでいます。以下は、先ほどの Ping の例をもとに開発した実装クラスからの抜粋です。

```
public static void main(String args[]) throws Exception {
  if (args.length > 0)
  remoteDomain = args[0];

  Pinger obj = new PingImpl();
  Context initialNamingContext = new InitialContext();
  initialNamingContext.rebind(NAME,obj);
  System.out.println("PingImpl created and bound to "+ NAME);
```

リモート インタフェースと実装クラスが開発できたら、Java コンパイラでコンパイルします。RMI-IIOP アプリケーションでのこれらのクラスの開発は、通常の RMI での開発と同じです。RMI オブジェクトの開発の詳細については、「WebLogic RMI API」を参照してください。

IIOP クラスの生成

実装クラスに対して -iiop オプションを使用して WebLogic RMI コンパイラを実行し、必要な IIOP スタブとスケルトンを生成します。スタブは、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 スケルトンに転送します。サーバサイド スケルトンは、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。-iiop オプションを使用して WebLogic RMI コンパイラを実行するには、次のコマンド パターンを使用します。

```
$ java weblogic.rmic -iiop nameOfImplementationClass
```

Pinger の例では、*nameOfImplementationClass* の部分が examples.iiop.rmi.server.wls.PingerImpl になります。-iiopDirectory オプションを使用すると、これらの IIOP クラスをどこに記述するかを指定できます。生成されるファイルは、*nameOfInterface_Stub.class* および *nameOfInterface_Skel.class* です。作成される 4 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、およびそのスタブとスケルトンは、オブジェクトの main() メソッドのネーミング方式で使用した 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。

RMI クライアントの開発

RMI over IIOP クライアントは、初期コンテキストを作成してオブジェクトをルックアップすることで、リモート オブジェクトにアクセスします。次に、このオブジェクトは適切な型にキャストされます。RMI over IIOP RMI クライアントが通常の RMI クライアントと異なるのは、初期コンテキストを取得する際にプロトコルとして IIOP が定義されるという点です。このため、ルックアップとキャストは、javax.rmi.PortableRemoteObject.narrow() メソッドと組み合わせて行われます。

たとえば、RMI クライアント ステートレス セッション Bean サンプル（配布キットに含まれている examples.iiop.ejb.stateless.rmiclient パッケージ）では、RMI クライアントが初期コンテキストを作成し、EJBean ホームの

ルックアップを行い、EJBean への参照を取得して、EJBean のメソッドを呼び出します。この例を IIOP で動作させるには、クライアントで以下の手順を実行します。

- 初期コンテキストを取得します。
- `javax.rmi.PortableRemoteObject.narrow()` メソッドと組み合わせてルックアップを実行するように、クライアントのコードを修正します。

初期コンテキストの取得では、JNDI コンテキスト ファクトリを定義する際に以下の 2 つの選択肢があります。

- `weblogic.jndi.WLInitialContextFactory`
- `com.sun.jndi.cosnaming.CNCtxFactory`

これらのいずれかを使用して、新しい `InitialContext()` のパラメータとして提供する「`Context.INITIAL_CONTEXT_FACTORY`」プロパティの値を設定できます。Sun バージョンを使用している場合は、Sun JNDI クライアント、つまり J2SE 1.3 の Sun RMI-IIOP ORB 実装を使用することになります。このことは、クライアントでの WebLogic クラスの使用を最小限に抑える場合には重要です。ただし、WebLogic の RMI-IIOP 実装を十分に活用するには、`weblogic.jndi.WLInitialContextFactory` メソッドを使用することをお勧めします。

Sun JNDI クライアントおよび Sun ORB を使用する場合、いくつか注意すべき点があります。JNDI クライアントでは、ネームスペースからリモート オブジェクト参照を読み込む機能がサポートされていますが、シリアライズされた汎用 Java オブジェクトの読み込みはサポートされていません。つまり、ネームスペースから EJBHome などを読み込むことはできますが、DataSource オブジェクトを読み込むことはできません。このコンフィグレーションでは、クライアントが開始したトランザクション（JTA API）もサポートされません。また、セキュリティもサポートされていません。ステートレス セッション Bean の RMI クライアントの例では、次のコードで初期コンテキストを取得します。

コード リスト 1-1 InitialContext を取得するコード

```
* JDK1.3 クライアントでは、次のように Properties オブジェクトを使用すると
* 機能する
*/

  private Context getInitialContext() throws NamingException {
```

```
try {
  // InitialContext を取得
  Properties h = new Properties();
  h.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.cosnaming.CNCtxFactory");
  h.put(Context.PROVIDER_URL, url);
  return new InitialContext(h);
} catch (NamingException ne) {
  log("We were unable to get a connection to the WebLogic server at
  "+url);
  log("Please make sure that the server is running.");
  throw ne;
  }

/**
* Java2 バージョンを使用して InitialContext を取得する場合。
* このバージョンは、jndi.properties ファイルがアプリケーションのクラスパス
* に存在するかどうかに依存する。
* 詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jndi/jndi.html
* を参照
*/

private static Context getInitialContext()
  throws NamingException

{
  return new InitialContext();
}
```

初期コンテキストを取得した後は、オブジェクトを特定のクラス型にキャストする際に必ず javax.rmi.PortableRemoteObject.narrow() を使用します。たとえば、EJBean ホームを探し、その結果を TraderHome オブジェクトに渡すクライアント コードは、次のように javax.rmi.PortableRemoteObject.narrow() を使用するように修正します。

コード リスト 1-2 ルックアップを実行するコード

```
/**
 * RMI/IIOP クライアントはこの narrow 関数を使用する
 */
private Object narrow(Object ref, Class c) {
  return PortableRemoteObject.narrow(ref, c);
}

/**
 * JNDI ツリーで EJB ホームをルックアップ
 */
private TraderHome lookupHome()
  throws NamingException
{
```

```
  // JNDI を使用して Bean ホームをルックアップ
  Context ctx = getInitialContext();

      try {
       Object home = ctx.lookup(JNDI_NAME);
       return (TraderHome) narrow(home, TraderHome.class);
}      catch (NamingException ne) {
       log("The client was unable to lookup the EJBHome.  Please
       make sure ");
       log("that you have deployed the ejb with the JNDI name
       "+JNDI_NAME+" on the WebLogic server at "+url);
       throw ne;
  }
}

/**
 * JDK1.3 クライアントでは、次のように Properties オブジェクトを使用すると
 * 機能する
 */
private Context getInitialContext() throws NamingException {

      try {
       // InitialContext を取得
       Properties h = new Properties();
       h.put(Context.INITIAL_CONTEXT_FACTORY,
       "com.sun.jndi.cosnaming.CNCtxFactory");
       h.put(Context.PROVIDER_URL, url);
       return new InitialContext(h);
  }    catch (NamingException ne) {
       log("We were unable to get a connection to the WebLogic
       server at "+url);
       log("Please make sure that the server is running.");
       throw ne;
  }
}
```

---

urlでは、コマンドラインの引数として渡されたプロトコル、ホスト名、WebLogic Server 用のリスニング ポートを定義し、それらがコマンドライン引数として渡されます。

```
public static void main(String[] args) throws Exception {

  log("\nBeginning statelessSession.Client...\n");

  String url       = "iiop://localhost:7001";
```

このクライアントが IIOP 経由で接続できるようにするには、次のようなコマンドでクライアントを実行します。

```
$ java -Djava.security.manager -Djava.security.policy=java.policy
  examples.iiop.ejb.stateless.rmiclient.Client
  iiop://localhost:7001
```

クライアントの RMI インタフェースをナロー変換するには、サーバから RMI インタフェース用の適切なスタブが提供される必要があります。このクラスのロードは、JDK ネットワーク クラスローダの使用を前提としており、デフォルトでは有効にはなっていません。これを有効にするには、適切な Java ポリシー ファイルを使用して、クライアントのセキュリティ マネージャを設定する必要があります。`java.policy` ファイルの例を示します。

```
grant {
// 一時的にパーミッションを付与する
permission java.security.AllPermission;
}
```

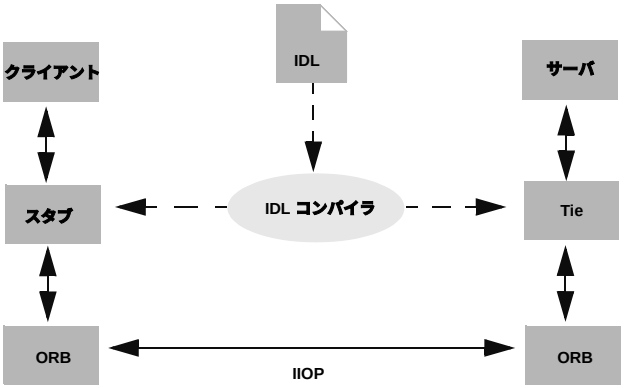
クライアントのセキュリティ マネージャを設定するには次のようにします。

```
java -Djava.security.manager -Djava.security.policy==java.policy
myclient
```

IDL クライアントを使用した RMI over IIOP

CORBA では、リモート オブジェクトへのインタフェースは、プラットフォームに依存しないインタフェース定義言語（IDL）で記述されます。IDL を特定の言語にマッピングするには、IDL コンパイラで IDL をコンパイルします IDL コンパイラによって、スタブやスケルトンなど、多くのクラスが生成されます。これらのクラスは、クライアントやサーバで、リモート オブジェクトへの参照の取得や、リクエストの転送、受信した呼び出しのマーシャリングなどに使用します。IDL クライアントを使用する場合でも、以降の節で説明するように、Java リモート インタフェースと実装クラスを使用することをお勧めします。IDL で記述したコードを逆マッピングして Java コードを作成することも可能ですが、これは難しく、バグの多いエンタープライズになります。Java リモート インタフェースと実装ファイルを使用して IDL を生成すれば、WebLogic クライアントおよび CORBA クライアントとの相互運用性も確保できます。

図 1-2 IDL クライアント（CORBA オブジェクト）の関係

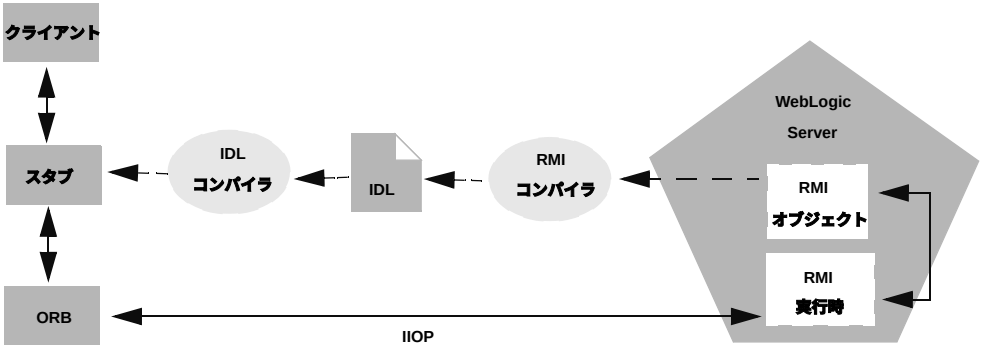


Java と IDL のマッピング

WebLogic RMI over IIOP では、リモート オブジェクトへのインタフェースは、`java.rmi.Remote` を拡張した Java リモート インタフェースに記述されます。Java-to-IDL マッピング仕様は、IDL が Java リモート インタフェースからどのように作成されるのかを定義しています。WebLogic RMI over IIOP 実装では、実装クラスは、WebLogic RMI コンパイラ、または WebLogic EJB コンパイラに `-idl` オプションを付けて実行されます。これにより、Java-to-IDL マッピング仕様を使って、リモート インタフェースの IDL に相当する部分が作成されます。次に、IDL コンパイラでこの IDL がコンパイルされ、CORBA クライアントに必要なクラスが生成されます。

クライアントは、リモート オブジェクトへの参照を取得し、スタブを介してメソッド呼び出しを転送します。WebLogic Server では、受信した IIOP リクエストを解析して RMI 実行時にそのリクエストを直接ディスパッチする `CosNaming` サービスを実装しています。

図 1-3 WebLogic RMI over IIOP オブジェクトの関係



Objects-by-Value

Objects-by-Value 仕様では、2 つのプログラミング言語間で複合データ型をやり取りすることができます。IDL クライアントで Objects-by-Value をサポートするには、Objects-by-Value をサポートする Object Request Broker（ORB）と組み合わせてそのクライアントを開発する必要があります現在のところ、Objects-by-Value を正確にサポートしている ORB はあまりありません。RMI over IIOP アプリケーションを開発する際には、IDL クライアントで Objects-by-Value をサポートするかどうかを考慮し、それに従って RMI インタフェースを設計する必要があります。つまり、Objects-by-Value をサポートしない IDL クライアントのみをアプリケーションでサポートする場合は、RMI インタフェースがプリミティブ データ型だけを渡すように制限する必要があります詳細については、「valuetype」を参照してください。

IDL を使用した RMI over IIOP アプリケーションの開発

IDL を使用して RMI over IIOP アプリケーションを開発するには、以下の手順を実行する必要があります。

1. リモート インタフェースと実装クラスの開発を行い、Java コンパイラでコンパイルします。

2. WebLogic RMI コンパイラまたは WebLogic EJB コンパイラで IDL ファイルの生成を行います。
3. IDL コンパイラで IDL ファイルのコンパイルを行い、その結果として生成されたクラスを C++ などの言語固有のコンパイラでコンパイルします。
4. IDL クライアントの開発を行い、言語固有のコンパイラでコンパイルします。

注意： 最初に、リモート インタフェースと実装クラスを開発します。CORBA プログラマの場合は IDL でコーディングする方が簡単かもしれませんが、IDL から Java へのマッピングがあまりうまく行かないため問題が発生するおそれがあります。IDL から Java を生成すること（「逆マッピング」と呼ばれます）はお勧めできません。

リモート インタフェースと実装クラスの開発

RMI オブジェクトを開発するには、`java.rmi.Remote` を拡張するインタフェースでオブジェクトのパブリック メソッドを定義する必要があります。

RMI オブジェクトでは、*`interfaceName`*`Impl` というクラスにインタフェースを実装できます。その実装クラスを JNDI ツリーにバインドすると、クライアントで使用できるようになります。実装クラスは通常、WebLogic スタートアップ クラスとしてコンフィグレーションされ、JNDI ツリーにオブジェクトをバインドする `main` メソッドを含んでいます。リモート インタフェースと実装クラスの開発は、RMI、IDL、および Tuxedo のすべてのクライアントで同じです。この最初の手順についての詳細は、「リモート インタフェースと実装クラスの開発」を参照してください。RMI オブジェクトの開発の詳細については、「WebLogic RMI API」を参照してください。

非 OBV クライアントのサポートに関する考慮事項

クライアント ORB が Objects-by-Value をサポートしない場合、RMI インタフェースを制限して、他のインタフェースか CORBA プリミティブ データ型のみを渡すようにする必要があります。次の表に、Objects-by-Value のサポートについてテスト済みの ORB を示します

表 1-1

ベンダ	バージョン	Objects-by-Value
Inprise	VisiBroker 3.3、3.4	サポートされていない
Inprise	VisiBroker 4.x	サポートされている
JavaSoft	JDK 1.2	サポートされていない
JavaSoft	RMI over IIOP 参照実装	サポートされていない
Iona	Orbix 2000	サポートされている（ただし、この実装ではいくつかの問題が認識されている）

IDL ファイルの生成

実装クラスを開発してコンパイルしたら、`-idl` オプションを付けて WebLogic RMI コンパイラまたは WebLogic EJB コンパイラを実行し、IDL ファイルを生成する必要があります。必要なスタブ クラスは、IDL ファイルのコンパイル時に生成されます。これらのコンパイラに関する一般的な情報については、「WebLogic RMI API」と『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。「Java Language Mapping to OMG IDL Specification」に記載されている Java IDL 仕様も参照してください。

以下のコンパイラ オプションは、RMI over IIOP に固有なものです。

オプション	機能
`-idl`	コンパイルされている実装クラスのリモート インタフェース用の IDL ファイルを作成する。

オプション	機能
-idlDirectory	生成された IDL の保存先ディレクトリを指定する。
-idlFactories	valuetype のファクトリ メソッドを生成する。クライアント ORB が factory valuetype をサポートしていない場合に役立つ。
-idlNoValueTypes	valuetype の IDL が生成されないようにする。
-idlOverwrite	同名の IDL ファイルが存在する場合には、コンパイラによって上書きされる。
-idlStrict	Objects-By-Value 仕様に厳密に従った IDL を生成する（ejbc では使用できない）。
-idlVerbose	IDL 情報についての冗長な情報を表示する。
-idlVisibroker	Visibroker 4.1 C++ とある程度の互換性がある IDL を生成する。

オプションの適用例を、次の RMI コンパイラの実行例で示します。

```
> java weblogic.rmic -idl -idlDirectory /IDL rmi_iiop.HelloImpl
```

コンパイラは、実装クラスのパッケージに従って、IDL ファイルを、idlDirectoy のサブディレクトリ内に生成します。たとえば上記のコマンドでは、Hello.idl ファイルが \IDL\rmi_iiop ディレクトリに生成されます。idlDirectory オプションを指定しなかった場合、IDL ファイルは、スタブやスケルトン クラスの生成先を基にした場所に生成されます。

IDL ファイルのコンパイル

IDL ファイルがあれば、これを基に IDL クライアントで必要なスタブ クラスを作成し、リモート クラスと通信できます。ORB ベンダによって IDL コンパイラが提供されます。

WebLogic コンパイラで生成された IDL ファイルには、#include orb.idl ディレクティブが含まれています。この IDL ファイルは各 ORB ベンダから提供されます。orb.idl ファイルは、WebLogic 配布キットの \lib ディレクトリにあります。このファイルは、JDK の ORB で使用されることを前提にしています。

IDL クライアントの開発

IDL クライアントは、純粋な CORBA クライアントで、WebLogic クラスはまったく必要としません。ORB ベンダによっては、リモート クラスへの参照を解決し、ナロー変換して、取得する場合のために、追加のクラスが生成されることがあります。VisiBroker 4.1 ORB 向けに開発された次のクライアントの例では、クライアントはネーミング コンテキストを初期化し、リモート オブジェクトへの参照を取得し、そのリモート オブジェクトに対するメソッドを呼び出します。

コード リスト 1-3 RMI-IIOP の例の C++ クライアントから抜粋したコード

```
// 文字列からオブジェクト
CORBA::Object_ptr o;

cout << "Getting name service reference" << endl;
if (argc >= 2 && strncmp (argv[1], "IOR", 3) == 0)
  o = orb->string_to_object(argv[1]);
else
  o = orb->resolve_initial_references("NameService");

// ネーミング コンテキストを取得
cout << "Narrowing to a naming context" << endl;
CosNaming::NamingContext_var context =
CosNaming::NamingContext::_narrow(o);
CosNaming::Name name;
name.length(1);
name[0].id = CORBA::string_dup("Pinger_iiop");
name[0].kind = CORBA::string_dup("");

// 解決して RMI オブジェクトにナロー変換
cout << "Resolving the naming context" << endl;
CORBA::Object_var object = context->resolve(name);

cout << "Narrowing to the Ping Server" << endl;
::examples::iiop::rmi::server::wls::Pinger_var ping =
  ::examples::iiop::rmi::server::wls::Pinger::_narrow(object);

// それを ping
cout << "Ping (local) ..." << endl;
ping->ping();
```

```
}
```

ネーミング コンテキストを取得する前に、標準のオブジェクト URL（CORBA/IIOP 2.4.2 仕様、13.6.7）を使用して初期参照が解決されている点に注目してください。サーバでのルックアップが、COS ネーミング サービス API を実装する JNDI のラッパーによって解決されています。

このネーミング サービスは、WebLogic Server アプリケーションが論理名を使ってオブジェクト参照を公開するのを可能にしています。これにより、IDL クライアント アプリケーションは、CORBA ネーム サービスに WebLogic Server の JNDI ツリー内のルックアップを要求することでオブジェクトを見つけることができるようになります。CORBA ネーム サービスは以下を提供します。

- Object Management Group（OMG）Interoperable Name Service（INS）仕様の実装
- オブジェクト参照をネーミングの階層構造（この場合は JNDI）にマッピングするためのアプリケーション プログラミング インタフェース（API）
- バインドを表示したり、ネーミング コンテキスト オブジェクトやアプリケーション オブジェクトをネームスペースにバインドしたりアンバインドしたりするためのコマンド

上記の例では、`Client.exe -ORBInitRef NameService=iioploc://localhost:7001/NameService` を使用してクライアントを実行します。

注意： ネーミング コンテキストは、CORBA オブジェクトを WebLogic IOR にナロー変換することでも取得できます。WebLogic Server に付属の `host2ior` ユーティリティを使用し、次のコマンドを実行して WebLogic Server IOR をコンソールに出力できます（このユーティリティは非推奨となったため、プロダクション システムでは使用されません）。

```
$  java utils.host2ior hostName port
```

`hostName` は WebLogic Server が稼働するマシンのホスト名、`port` は WebLogic Server が接続をリスンするポートです。

Tuxedo と Tuxedo クライアントを使用した RMI-IIOP

WebLogic Server には、RMI-IIOP を使って WebLogic Server アプリケーションと Tuxedo サービスの相互運用を可能にする機能が備わっています。EJB や WebLogic のその他のアプリケーションを、Tuxedo クライアントから呼び出す機能も備えています。

インストール環境の `samples/examples/iiop` ディレクトリにある RMI-IIOP のサンプルには、Tuxedo サーバおよび Tuxedo クライアントと共に動作するよう WebLogic Server をコンフィグレーションおよびセットアップする方法のサンプルが含まれています。

WebLogic-Tuxedo Connector

WebLogic-Tuxedo Connector を使用すると、WebLogic Server アプリケーションと Tuxedo サービスの相互運用が実現されます。このコネクタでは、XML コンフィグレーション ファイルを使用することで、Tuxedo サービスを呼び出すように WebLogic Server をコンフィグレーションできます。また、サービス要求に応じて、Tuxedo から WebLogic Server エンタープライズ JavaBean（EJB）やその他のアプリケーションを呼び出すこともできます。Tuxedo で開発したアプリケーションを WebLogic Server に移行する場合や、従来の Tuxedo システムを新しい WebLogic 環境に統合する場合、WebLogic Tuxedo Connector を使用すれば、スケーラビリティと信頼性の高い Tuxedo の CORBA 環境を有効に活用できます。以下では、WebLogic Tuxedo Connector についての情報を提供し、Tuxedo での CORBA アプリケーションの構築について説明します。

- WebLogic-Tuxedo Connector の各種ガイド
- Tuxedo に関する「CORBA topics」

BEA WebLogic C++ クライアント

WebLogic Server 6.1 SP3 は、Tuxedo 8.0 C++ Client ORB と相互運用できます。このクライアントは、値によるオブジェクトと CORBA Interoperable Naming Service (INS) をサポートします。Tuxedo リリース 8.0 RP 56 以上が必要です。Tuxedo C++ Client ORB を入手する方法については、BEA のサービス担当者までお問い合わせください。

Tuxedo C++ Client ORB と共に WebLogic C++ クライアントを使用する方法については、以下のドキュメントを参照してください。

- Tuxedo Corba クライアント アプリケーションの作成方法の概要については、「BEA Tuxedo Corba クライアント アプリケーションの開発」を参照してください。
- C++ IDL コンパイラの使い方については、「OMG IDL 構文と C++ IDL コンパイラ」を参照してください。
- Interoperable Naming Service を使用して NameService などの初期オブジェクトに対するオブジェクト参照を取得する方法については、「インターオペラブル ネーミング サービス ブートストラップ処理メカニズム」を参照してください。

WebLogic Server の RMI-IIOP 用コンフィグレーション

CORBA クライアントからクライアント ID を伝播するための標準がないため、IIOP で接続しているクライアントの ID はデフォルトで「guest」になります。次の例に示すように、ユーザおよびパスワードを `config.xml` ファイルで設定して、IIOP で接続するすべてのクライアントに対して単一の ID を確立できます。

```
<Server
Name="myserver"
NativeIOEnabled="true"
DefaultIIOPUser="Bob"
DefaultIIOPPassword="Gumby1234"
ListenPort="7001">
```

また、config.xml では IIOPEnabled 属性も設定できます。デフォルト値は「true」です。IIOP のサポートを無効にする場合のみ 「false」に設定してください。すべてのリモート オブジェクトが JNDI ツリーにバインドされて、クライアントからアクセスできるようになることが保証されている限り、RMI over IIOP を使用するために、WebLogic Server を特別にコンフィグレーションする必要はありません。RMI オブジェクトは通常、スタートアップ クラスによって JNDI ツリーにバインドされます。EJBean ホームは、デプロイメント時に JNDI ツリーにバインドされます。WebLogic Server は、JNDI ツリーのルックアップ呼び出しをすべて委託することにより、CosNaming サービスを実装します。

WebLogic Server 6.1 では、RMI-IIOP の corbaname および corbaloc JNDI 参照がサポートされています。CORBA/IIOP 2.4.2 仕様を参照してください。したがって、ejb-jar.xml に次のように追加することができます。

```
<ejb-reference-description>
<ejb-ref-name>WLS</ejb-ref-name>
<jndi-name>corbaname:iiop:1.2@localhost:7001#ejb/foo</jndi-name>
</ejb-reference-description>
```

reference-description スタンザは、ejb-jar.xml で定義されたリソース参照を、WebLogic Server で使用可能な実際のリソースの JNDI 名にマップします。ejb-ref-name ではリソース参照名を指定します。このリソース参照は、EJB プロバイダが ejb-jar.xml デプロイメント ファイル内に配置する参照です。jndi-name では、WebLogic Server で使用可能な実際のリソース ファクトリの JNDI 名を指定します。

iiop:1.2 は、<jndi-name> セクションに含まれています。WebLogic Server 6.1 には、GIOP 1.2 の実装が含まれています。GIOP では、相互運用している ORB 間で交換されるメッセージの形式を指定します。これにより、数多くの ORB およびアプリケーション サーバとの相互運用性が確保されます。GIOP のバージョンは、corbaname 参照または corbaloc 参照のバージョン番号によって制御できます。

考慮事項

RMI-IIOP と RMI オブジェクトのライフサイクル

WebLogic Server のデフォルトのガベージ コレクション機能は、使用も参照もされていないサーバ オブジェクトのガベージ コレクションを行います。これにより、使われていない多数のオブジェクトのためにメモリが不足する危険が減ります。しかし、RMI-IIOP では、クライアントがリモート オブジェクトに対する参照を保持していても、6 分程度そのオブジェクトを呼び出していない場合、このポリシーのために `NoSuchObjectException` エラーが発生する可能性があります。EJB の場合、または一般に JNDI などを通してサーバ インスタンスが参照する RMI オブジェクトの場合は、このような例外は発生しません。

RMI-IIOP に対する J2SE の仕様では、RMI-IIOP の下での RMI オブジェクトのライフサイクルを管理するには、Distributed Garbage Collection (DGC) ではなく、`javax.rmi.PortableRemoteObject` に対する `exportObject()` メソッドと `unexportObject()` メソッドを使うように規定されています。ただし、`exportObject()` と `unexportObject()` は WebLogic Server のデフォルトのガベージ コレクション ポリシーには影響を及ぼさないことに注意してください。デフォルトのガベージ コレクション ポリシーを変更する必要がある場合は、BEA のテクニカル サポートに問い合わせてください。

クライアントで RMI-IIOP を使用する際の制約

JDK 1.3（およびその他のバージョン）は RMI-IIOP には準拠していません。クライアントで RMI-IIOP を使用する場合は、JDK に関して以下の点に注意してください。

- クライアントによって境界が定められるトランザクションはサポートされません。
- IOR では GIOP 1.0 メッセージおよび GIOP 1.1 プロファイルが送信されます。
- EJB 2.0 の相互運用性に必要な部分（GIOP 1.2、コードセット ネゴシエーション、UTF-16）はサポートされません。

- 区切られたメソッド名の扱いにバグがあります。
- チェックされていない例外が正しくアンマーシャリングされません。
- valuetype のエンコーディングに関連する細かなバグがあります。
- クラスタ (フェイルオーバとロード バランシング) はサポートしてません。

これらの多くは、両方でサポートすることが不可能なものです。WebLogic では、できる限り仕様に準拠しているオプションがサポートされています。

Java と IDL クライアント モデル

RMI-IIOP を使用する予定がある場合は、RMI クライアント モデルを使用して Java クライアントを開発することをお勧めします。Java IDL クライアントを開発する場合、さまざまな問題に直面するおそれがあります。クラスパスの問題だけでなく、名前の衝突、サーバサイドとクライアントサイドでのクラスの区別など、さまざまな問題が発生することが予想されます。RMI オブジェクトと IDL クライアントは異なるタイプのシステムを持っているので、サーバサイドのインタフェースを定義するクラスは、クライアントサイドのインタフェースを定義するクラスとは全く異なるものになります。

RMI-IIOP での EJB の使用

異種サーバ環境において EJB の相互運用性を確保するため、RMI over IIOP を使用してエンタープライズ JavaBean を実装する場合、CORBA に対する EJB アーキテクチャの標準マッピングを使用することで以下が可能となります。

- ORB を使用する Java RMI クライアントが、WebLogic Server over IIOP 上のエンタープライズ Bean にアクセスできます。
- Java 以外のプラットフォームの CORBA クライアントが、WebLogic Server 上のエンタープライズ Bean オブジェクトにアクセスできます。

WebLogic RMI over IIOP は、EJB から IDL クライアントへの接続を可能にするためのフレームワークです。ただし、現時点では、EJB-to-IDL の実装に必要なユーザ ID をどのように渡すかについて標準的な方法がありません。また、クラ

イアントからのトランザクションの伝播についても懸案事項になっています。RMI over IIOP では、CORBA/IDL クライアントは EJBean にアクセスできますが、以下のサービスはまだ利用できません。

- EJB トランザクション サービス
- EJB セキュリティ サービス

WebLogic Server アプリケーションから CORBA クライアントにマッピングを派生する場合、マッピング情報のソースは Java ソース ファイルに定義された EJB クラスです。WebLogic Server には、必要な IDL ファイルを生成するための `weblogic.ejbc` ユーティリティが用意されています。これらのファイルは、CORBA ビューを、対象となる EJB のステートと動作で表します。`weblogic.ejbc` ユーティリティを使用すると、以下を実行できます。

- EJB クラス、インタフェース、およびデプロイメント記述子ファイルを JAR ファイルに配置する。
- EJB 用の WebLogic Server コンテナ クラスを生成する。
- RMI コンパイラを使用して各 EJB コンテナ クラスを実行し、スタブとスケルトンを作成する。
- これらのクラスへの CORBA インタフェースを記述する CORBA IDL ファイルのディレクトリ ツリーを生成する。

`weblogic.ejbc` ユーティリティでは、さまざまなコマンド修飾子がサポートされています。「IDL ファイルの生成」の表を参照してください。

その結果として生成されたファイルはコンパイラで処理されます。`idlSources` ディレクトリからソース ファイルが読み込まれ、CORBA C++ のスタブ ファイルとスケルトン ファイルが生成されます。これらのファイルは、valuetype を除くすべての CORBA データ型に対して有効です（詳細については以下の節を参照してください）。生成された IDL ファイルは、`idlSources` ディレクトリに配置されます。Java から IDL への処理にはさまざまな問題があります。Java Language Mapping to OMG IDL 仕様を参照してください。

次の例（配布キットの *`WL_HOME`*`\examples\iiop\ejb\entity\server\wls`）では、作成済みの Bean から IDL を生成する方法を示します。

```
> java weblogic.ejbc -compiler javac -keepgenerated
-idl -idlDirectory idlSources
-iiop build\std_ejb_iiop.jar
%APPLICATIONS%\ejb_iiop.jar
```

その後、EJB インタフェースおよびクライアント アプリケーションをコンパイルしています（この例では CLIENT_CLASSES ターゲット変数を使用しています）。

```
> javac -d %CLIENT_CLASSES% Trader.java TraderHome.java
TradeResult.java Client.java
```

つづいて、`weblogic.ejbc` を使用して作成した IDL ファイルに対して IDL コンパイラを実行し、C++ のソース ファイルを作成しています。

```
>%IDL2CPP% idlSources\examples\rmi_iiop\ejb\Trader.idl
. . .
>%IDL2CPP% idlSources\javax\ejb\RemoveException.idl
```

これで、C++ クライアントをコンパイルできます。

valuetype

参照ではなく値でオブジェクトを渡す場合、valuetype を使用する必要があります（詳細については、http://cgi.omg.org/cgi-bin/doc?formal/99-10-07 にある CORBA 仕様の第 5 章を参照してください）。valuetype は、定義または参照されるプラットフォームごとに実装する必要があります。以下では、WebLogic Server 上のエンティティ Bean にアクセスする特殊な C++ クライアントを参照して、複雑な valuetype を渡すことの難しさについて説明します（`examples\iiop\ejb\entity\server\wls` ディレクトリおよび `examples\iiop\ejb\entity\cppclient` ディレクトリを参照してください）。

Java プログラマが直面する問題の 1 つは、常に可視とは限らない派生データ型の使用についての問題です。たとえば、EJB ファインダにアクセスする場合、Java プログラマは Collection または Enumeration を見ますが、その基盤となる実装には注意を払いません。これは、JDK 実行時によってネットワーク経由でクラスロードされるためです。しかし、C++ および CORBA のプログラマはネットワークを介して送られてくる型を知っている必要があります。型を知っていることで、その型用の valuetype ファクトリを登録でき、ORB によるアンマーシャリングが可能となります。

`\iiop\ejb\entity\cppclient` のサンプルにある valuetype の例は、EJBObjectEnum および Vector です。定義された EJB インタフェースで `ejbc` を実行するだけでは、これらの定義はインタフェースには表示されず生成もされません。このため、`ejbc` では、特にりモート インタフェース用に IDL を生成する目的で、りモート インタフェースではない Java クラスも使用できます。valuetype ファクトリを登録する方法については、`\iiop\ejb\entity\cppclient` の例を参照してください。

シリアライズ可能で `writeObject()` を定義する Java 型は、IDL 内のカスタム valuetype にマップされます。valuetype を手作業でアンマーシャルするには、C++ コードを記述する必要があります。その例としては、`examples\iiop\ejb\entity\tuxclient\ArrayList_i.cpp` を参照してください。

注意： Tuxedo を使用している場合は、IDL コンパイラに対して `-i` 修飾子を指定して、*`FileName`*`_i.h` および *`FileName`*`_i.cpp` という名前の実装ファイルを作成できます。たとえば、次の構文では、実装ファイルとして `TradeResult_i.h` および `TradeResult_i.cpp` が作成されます。

```
idl -IidlSources -i
idlSources\examples\rmi_iiop\ejb\rmi_iiop\TradeResult.idl
```

その結果として生成されたソース ファイルによって、valuetype にアプリケーション定義オペレーションの実装が提供されます。実装ファイルは、CORBA クライアント アプリケーションに含まれます。

SSL を使用した RMI over IIOP

SSL プロトコルを使用すると、RMI または EJB リモート オブジェクトへの IIOP 接続を保護できます。SSL プロトコルは、認証を通じて接続を保護し、オブジェクト間のデータ交換を暗号化します。WebLogic Server では、以下の方法で SSL を使用した RMI over IIOP を使用できます。

- CORBA/IDL クライアント Object Request Broker（ORB）による方法
- Java クライアントを使用した方法

どちらの場合でも、SSL プロトコルを使用するよう、WebLogic Server をコンフィグレーションする必要があります。詳細については、「SSL プロトコルのコンフィグレーション」を参照してください。

CORBA/IDL クライアント ORB による SSL を使用した RMI over IIOP を使用するには、次の手順を行います。

1. SSL プロトコルを使用するよう、CORBA クライアント ORB をコンフィグレーションします。SSL プロトコルのコンフィグレーションの詳細については、クライアント ORB の製品マニュアルを参照してください。

2. host2ior ユーティリティを使用して、WebLogic Server IOR をコンソールに出力します。host2ior ユーティリティによって、SSL 接続用と非 SSL 用の 2 種類の IOR が出力されます。
3. SSL IOR は、WebLogic Server JNDI ツリーにアクセスする CosNaming サービスへの初期参照を取得するときに使用します。

Java クライアントを使用した SSL を使用した RMI over IIOP を使用するには、次の手順を行います。

1. コールバックを使用する場合は、Java クライアントのプライベート キーとデジタル証明書を取得します。
2. -d オプションを付けて ejbc コンパイラを実行します。
3. RMI クライアントを起動する場合は、下記のコマンド オプションを使用します。マシン名、通常のポート、および SSL ポートを指定する必要があります。また、ORB 独自のクラスをラップし、セキュアな接続を処理する JDK の問題を解決する weblogic.corba.orb.ssl.ORB クラスを使用しなければなりません。

```
java -Dweblogic.security.SSL.ignoreHostnameVerification=true \
-Dweblogic.SSL.ListenPorts=localhost:7701:7702 \
-Dorg.omg.CORBA.ORBClass=weblogic.corba.orb.ssl.ORB \
weblogic.rmiiiop.HelloJDKClient iiop://localhost:7702

*

* または、サーバからクライアントへの接続に cert チェーンを使用する

*

*java -Dweblogic.corba.orb.ssl.certs=myserver/democert.pem
-Dweblogic.corba.orb.ssl.key=myserver/demokey.pem
-Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.corba.orb.ssl.ListenPorts=localhost:7701:7702
-Dorg.omg.CORBA.ORBClass=weblogic.corba.orb.ssl.ORB
-Djava.security.manager -Djava.security.policy==java.policy
-ms32m -mx32m weblogic.rmiiop.HelloJDKClient port=7702


-Dssl.certs=directory location of digital certificate for Java
client
-Dssl.key=directory location of private key for Java client
```

Java クライアントでは、CLASSPATH に含まれる SSL プロトコル用に WebLogic Server が使用するクラスが必要になります。

着信接続の場合（コールバックのための WebLogic Server から Java クライアントへの接続）、Java クライアントのデジタル証明書とプライベート キーをコマンドラインで指定する必要があります。`ssl.certs` および `ssl.key` というコマンドライン オプションを使用して、情報を入力します。

クライアント証明書の使い方

クライアント ORB で SSL をサポートするように設定すると、RMI over IIOP および SSL と共にクライアント証明書を使用することにより、さらに高いレベルのセキュリティを実現できます。クライアント証明書の使用を選択するかどうかによって動作は異なります。

クライアント ORB は WebLogic Server の信頼された認証局を認識する必要があり、WebLogic Server は ORB の信頼された認証局を認識する必要があります。WebLogic Server に ORB の認証局を認識させるには、クライアント ORB の信頼された認証局を WebLogic Server にコピーします。

1. java `utils.der2pem` を使用して、認証局を変換します。
2. `ca.pem` ファイルを、新しい `ca_new.pem` ファイルにコピーします。
3. クライアント ORB の信頼された `ca.pem` ファイルを、新しい `ca_new.pem` ファイルの最後に追加します。
4. Console で、信頼された CA ファイルのファイル名を `ca_new.pem` に変更します。

   認証チェーン ファイルは `ca.pem` のままになります。

注意： クライアント ORB に WebLogic Server の信頼された認証局を認識させる方法については、ORB 製品のマニュアルを参照してください。

`weblogic.security.acl.CertAuthenticator` を実装して、Console でこのクラスを登録します。この処理方法の例については、WebLogic Server 配布キットの `examples.security.cert` を参照してください。

Administration Console を使用して、クライアント証明書の提示を強制するオプションを設定します。

1. [サーバ] ノードをクリックして、右ペインで必要なサーバを選択します。
2. [SSL] タブをクリックして、[クライアント認証を強制する] ボックスを選択します。

3. [適用] をクリックします。

RMI over IIOP と SSL を使用すると、次のように動作します。

クライアントが証明書の提示を強制される場合

- クライアント ORB は SSL を使用して IOR を呼び出します。
- サーバサイドでは、認証局クラスが呼び出されて、ユーザを認可するかどうかを決定します。

  認証局をコンフィグレーションするには、[SSL] タブを選択して、証明書の有効性の判定に使用する認証局を指定します。認証局クラスは証明書にアクセスして、証明書にあるフィールドを使用してユーザを識別します。

  WL_HOME`samples\examples\security\cert\` の例には、証明書の電子メール アドレスをレルム内のユーザにマップする、非常に単純な認証局クラスが含まれています。
- 認証局がコンフィグレーションされていない場合、または認証局が `null` を返す場合は、パーミッション例外はクライアントに返されず、メソッドは実行されません。

クライアントが証明書の提示を強制されない場合

- クライアント ORB は SSL を使用して IOR を呼び出します。
- サーバサイドでは、デフォルト IIOP ユーザが呼び出されます。

  [SSL] タブでこのユーザ オプションを設定するには、[プロトコル] タブをクリックして、[デフォルト IIOP ユーザ] ボックスをチェックします。

委託による CORBA クライアントから WebLogic Server オブジェクトへのアクセス

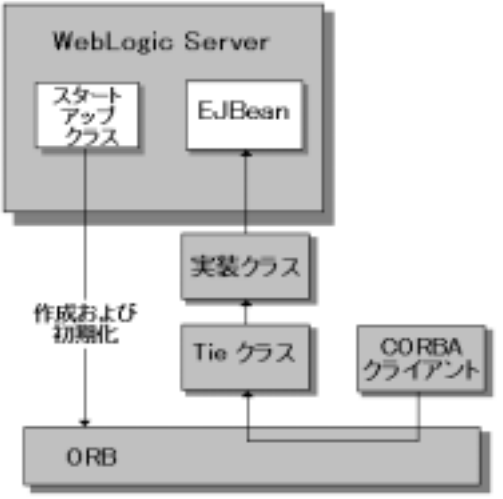
WebLogic Server は、CORBA クライアントから RMI リモート オブジェクトへのアクセスを可能にするサービスを提供します。しかし、別の方法として、WebLogic Server 内で CORBA ORB（Object Request Broker）をホストして送受信メッセージを委託することで、サーバ内でバインド可能なオブジェクトを CORBA クライアントから間接的に呼び出せるようにする方法もあります。ここでは、この方法の概要を説明します。

概要

WebLogic Server によってホストされたオブジェクトに CORBA 呼び出しを委託するためには、相互に連携しなければならないオブジェクトがいくつかあります。まず、WebLogic Server を実行している JVM と共存する ORB が必要です。これを実現するためには、ORB を作成して初期化するスタートアップ クラスを作成します。また、ORB から受信したメッセージを受け付けるオブジェクトも必要です。このオブジェクトを作成するには、IDL（インタフェース定義言語）を作成しなければなりません。IDL をコンパイルするといくつかのクラスが生成されますが、その 1 つが Tie クラスです。Tie クラスは、受信した呼び出しを処理して適切な実装クラスにディスパッチするためにサーバサイドで使用します。実装クラスはサーバへの接続を担い、適切なオブジェクトをルックアップして CORBA クライアントの代わりにオブジェクトのメソッドを呼び出します。

次の図には、サーバに接続して EJBean 上で動作する実装クラスに、EJBean の呼び出しを委託する CORBA クライアントを示します。同様のアーキテクチャを使用すると、これとは逆の状況でも機能します。スタートアップ クラスを使用すると、ORB を起動し、対象となる CORBA 実装オブジェクトへの参照を取得

できます。このクラスは、JNDI ツリー内の別の WebLogic オブジェクトで使用できるようにして、CORBA オブジェクトへの適切な呼び出しを委託することも可能です。



委託アーキテクチャ

コード例

次のコード例では、サーバに接続し、JNDI ツリー内で `Foo` オブジェクトをルックアップして `bar` メソッドを呼び出す実装クラスを作成しています。このオブジェクトは、以下の作業によって CORBA 環境を初期化するためのスタートアップ クラスでもあります。

- ORB を作成する
- Tie オブジェクトを作成する
- 実装クラスを Tie オブジェクトに関連付ける
- Tie オブジェクトを ORB に登録する
- Tie オブジェクトを ORB のネーミング サービス内でバインドする

スタートアップ クラスを実装する方法の詳細については、「WebLogic Server の起動と停止」を参照してください。

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.rmi.*;
import javax.naming.*;
import weblogic.jndi.Environment;

public class FooImpl implements Foo

{
  public FooImpl() throws RemoteException {
    super();

  }

  public void bar() throws RemoteException, NamingException {
    // 呼び出しを委託するためインスタンスをルックアップして呼び出す
    weblogic.jndi.Environment env = new Environment();
    Context ctx = env.getInitialContext();
    Foo delegate = (Foo)ctx.lookup("Foo");
    delegate.bar();
    System.out.println("delegate Foo.bar called!");

  }

  public static void main(String args[]) {
        try {
      FooImpl foo = new FooImpl();

      // ORB を作成および初期化
      ORB orb = ORB.init(args, null);

      // Tie を作成および ORB に登録
      _FooImpl_Tie fooTie = new _FooImpl_Tie();
      fooTie.setTarget(foo);
      orb.connect(fooTie);

      // ネーミング コンテキストを取得
      org.omg.CORBA.Object o = \
      orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(o);

          // ネーミングでオブジェクト参照をバインドする

      NameComponent nc = new NameComponent("Foo", "");
      NameComponent path[] = {nc};
      ncRef.rebind(path, fooTie);

      System.out.println("FooImpl created and bound in the ORB
      registry.");

    }
    catch (Exception e) {
```

```
            System.out.println("FooImpl.main: an exception occurred:");
            e.printStackTrace();
        }
    }
}
```

コード例

examples.iiop パッケージは、数多くのクライアント とアプリケーションとの接続を示したもので、*WL_HOME*\samples\examples\iiop ディレクトリにあります。EJB を RMI-IIOP で使用し、C++ クライアントに接続して Tuxedo サーバとの相互運用性を設定する例も用意されています。詳細については、例の説明を参照してください。WebLogic Tuxedo Connector に関する例は、\wlserver6.1\samples\examples\wtc ディレクトリにあります。

その他の情報源

WebLogic RMI over IIOP は、RMI の完全に実装することを目的としています。使用しているバージョンに適用される補足事項については、『リリース ノート』を参照してください。

- 『WebLogic JNDI プログラミング ガイド』
- 『WebLogic RMI プログラマーズ ガイド』
- Java Remote Method Invocation（RMI）のホームページ
- Sun の RMI 仕様
- Sun の RMI チュートリアル
  http://java.sun.com/j2se/1.3/docs/guide/rmi/getstart.doc.html
  http://java.sun.com/j2se/1.3/docs/guide/rmi/rmisocketfactory.doc.html
  http://java.sun.com/j2se/1.3/docs/guide/rmi/activation.html
- Sun の RMI over IIOP ドキュメント

- OMG のホームページ
- CORBA Language Mapping 仕様
- 「CORBA Technology and the Java Platform」
- Sun の Java IDL ページ
- Objects-by-Value 仕様