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 7.0
改訂 : 2004 年 8 月 16 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
なし	2004 年 8 月 16 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 vi
このマニュアルの印刷方法 vi
関連情報 vi
サポート情報 vii
表記規則 viii

1. RMI over IIOP の概要

RMI および RMI over IIOP とは 1-1
WebLogic RMI-IIOP の概要 1-2
RMI (Java) クライアント使用型 RMI-IIOP のサポート 1-3
Tuxedo クライアント使用型 RMI-IIOP のサポート 1-3
CORBA/IDL クライアント使用型 RMI-IIOP のサポート 1-4
プロトコルの互換性 1-4
サーバ間の相互運用性 1-4
クライアントとサーバ間の相互運用性 1-6

2. RMI over IIOP プログラミング モデル

RMI-IIOP プログラミング モデルの概要 2-1
IIOP を使用しない RMI アプリケーション 2-3
RMI (Java) クライアント使用型 RMI-IIOP アプリケーション 2-4
RMI (Java) クライアント使用型 RMI-IIOP を適用すべき状況 2-4
RMI クライアント使用型 RMI-IIOP アプリケーションの開発 2-5
WebLogic RMI-IIOP RMI クライアント使用型 RMI-IIOP アプリケーション 2-10
CORBA/IDL クライアント使用型 RMI-IIOP アプリケーション 2-11
CORBA/IDL クライアントの取り扱い 2-11
Java-to-IDL マッピング 2-12
Objects-by-Value 2-13
CORBA/IDL クライアント使用型 RMI-IIOP アプリケーションの開発 .

2-14
Tuxedo 8.1 ORB 用 WebLogic C++ クライアントの開発 2-17
いつ WebLogic C++ クライアントを使用するのか.....2-18
WebLogic C++ クライアントの仕組み.....2-18
WebLogic C++ クライアントの開発 2-19
WebLogic C++ クライアントの制限 2-20
WebLogic C++ クライアントのコード サンプル 2-20
WebLogic Tuxedo Connector を使用した RMI-IIOP アプリケーション 2-20
いつ WebLogic Tuxedo Connector を使用するのか.....2-20
WebLogic Tuxedo Connector の仕組み.....2-21
WebLogic Tuxedo Connector のコード サンプル.....2-21
RMI-IIOP での EJB の使用.....2-21
コード例.....2-23
RMI-IIOP と RMI オブジェクトのライフサイクル.....2-28

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

コンフィグレーションの概要.....3-1
RMI over IIOP と SSL の併用 3-3
RMI-IIOP と SSL および Java クライアントの併用.....3-3
BEA Tuxedo クライアントでの SSL プロトコルの使用.....3-4
委託を通じた CORBA クライアントからオブジェクトへのアクセス 3-4
委託の概要.....3-5
委託の例.....3-6
ハードウェア ロード バランサと RMI over IIOP の併用.....3-8
WebLogic RMI-IIOP の制約事項 3-8
サーバで RMI-IIOP を使用する際の制約.....3-9
クライアントで RMI-IIOP を使用する際の制約 3-9
Java IDL クライアントの開発上の制約.....3-10
オブジェクトの値渡しに関する制約.....3-10
RMI-IIOP サンプル コード パッケージ.....3-11
その他の情報源 3-12

このマニュアルの内容

このマニュアルでは、RMI (Remote Method Invocation) over IIOP (Internet Inter-ORB Protocol) について解説し、さまざまなクライアント タイプに応じた 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。RMI-IIOP では、BEA WebLogic Server 環境で Java クライアントから Java リモート オブジェクトにも CORBA リモート オブジェクトにもアクセスできるようにすることで、RMI プログラミング モデルをどう拡張しているのかを説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「RMI over IIOP の概要」では、RMI および RMI over IIOP とはどのようなものなのかを明らかにし、WebLogic Server RMI-IIOP 実装の概要を示します。
- 第 2 章「RMI over IIOP プログラミング モデル」では、さまざまなクライアント タイプに応じた RMI-IIOP アプリケーションの開発方法について説明します。
- 第 3 章「WebLogic Server の RMI-IIOP 用コンフィグレーション」では、WebLogic Server を用いた RMI-IIOP アプリケーションのサポートに関係する概念、問題点、および手続きについて説明します。

対象読者

このマニュアルは、IIOP (Internet Inter-ORB Protocol) を用いてクライアントから RMI (Remote Method Invocation) リモート オブジェクトにアクセスできるようにしようとお考えのアプリケーション開発者を対象としています。このマニュアルは、BEA 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、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。

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

表記法	適用
[ ]	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]`
|	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 RMI over IIOP の概要

以下の各節では、RMI over IIOP を概観します。

- RMI および RMI over IIOP とは
- WebLogic RMI-IIOP の概要
- プロトコルの互換性

RMI および RMI over IIOP とは

RMI-IIOP を理解するには、まず RMI を実務レベルで理解している必要があります。RMI (Remote Method Invocation) は、Java での分散オブジェクト コンピューティングの標準規格です。RMI を使用すると、アプリケーション側では、ネットワーク内の別の場所に存在するオブジェクトへの参照を取得したあと、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複数の Java 仮想マシン上でどのように動作するかを規定するものです。RMI は Java で記述されており、Java プログラム専用のものです。

RMI over IIOP は、IIOP プロトコルを介して動作するように RMI を拡張したものです。この拡張には役に立つメリットが 2 つあります。まず、Java-to-Java パラダイムでは、これによって、標準 IIOP (Internet Interop-Orb-Protocol) に対応したプログラムを作成できるようになります。一方、Java 単独の環境以外で作業している場合には、この拡張によって、Java プログラムから CORBA (Common Object Request Broker Architecture) クライアントとやり取りし、CORBA オブジェクトを実行できるようになります。CORBA クライアントは、さまざまな言語 (C++ を含む) で記述することができ、インタフェース定義言語 (IDL : Interface-Definition-Language) を用いてリモート オブジェクトとやり取りすることができます。

WebLogic RMI-IIOP の概要

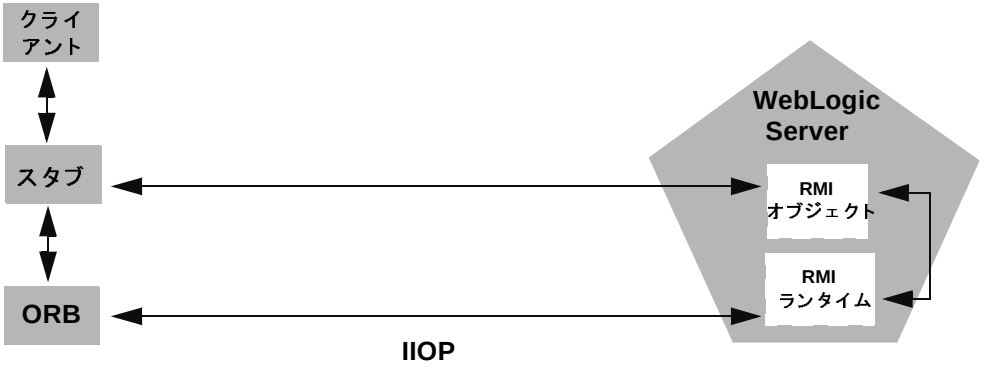
RMI over IIOP は RMI プログラミング モデルに基づいたものであり、また、それほどではないにせよ、JNDI (Java Naming and Directory Interface) にも基づいています。WebLogic RMI と JNDI の詳細については、『WebLogic RMI プログラマーズ ガイド』の「WebLogic RMI の実装」と『WebLogic JNDI プログラマーズ ガイド』を参照してください。これらの技術はどちらも RMI-IIOP にとってきわめて重要なものなので、RMI-IIOP アプリケーションの構築を始める前に、これらの技術の一般的な概念についてよく理解しておくことを強くお勧めします。

WebLogic Server 7.0 での RMI-IIOP 実装の機能は以下のとおりです。

- 標準 IIOP プロトコルを使用して Java RMI クライアントを WebLogic Server に接続できる。
- CORBA/IDL クライアント (C++ で記述されたものを含む) を WebLogic Server に接続できる。
- WebLogic Server と Tuxedo クライアントの間で相互運用できる。
- WebLogic Server をホストとする EJB に、さまざまなクライアントを接続できる。

このマニュアルでは、RMI および RMI-IIOP を用いるさまざまなクライアントタイプに応じたアプリケーションの作成方法について説明します。RMI-IIOP アプリケーションをどのように開発するかは、どのようなサービスやクライアントを統合するかによって決まります。

以下の図は、IIOP を用いた RMI オブジェクト関係を示しています。



RMI (Java) クライアント使用型 RMI-IIOP のサポート

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ができます。JDK のリリース 1.3.1 ではこの機能が強化され、Java-to-Java の環境において RMI-IIOP で手軽に WebLogic Server 7.0 を使用できます。また、完全にクラスタ化可能な RMI-IIOP アプリケーションを開発できる、「ファット」RMI-IIOP RMI クライアントも導入されました。この新しい WebLogic RMI-IIOP RMI クライアントを使用するには、クライアント サイドの CLASSPATH に `weblogic.jar` (*`WL_HOME`*`/server/lib` にある) を含め、`-Dweblogic.system.iiop.enableClient=true` コマンド ライン オプションを使用してクライアントを起動する必要があります。

Tuxedo クライアント使用型 RMI-IIOP のサポート

WebLogic Server 7.0 は WebLogic Tuxedo Connector の実装も備えています。これは、Tuxedo サーバとの相互運用を可能にする基盤技術です。WebLogic Tuxedo Connector を使用すると、Tuxedo を ORB として活用したり、WebLogic Server

上で開発したアプリケーションと従来の Tuxedo システムを統合することができます。詳細については、「WebLogic Tuxedo Connector」ページを参照してください。

CORBA/IDL クライアント使用型 RMI-IIOP のサポート

開発者にとっては、CORBA/IDL クライアントから J2EE サービスにアクセスできることが必要です。しかし、Java と CORBA は非常に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。この制限を解消するために、Object Management Group (OMG) によって Objects-by-Value 仕様が策定されました。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。

プロトコルの互換性

WebLogic Server 7.0 と WebLogic Server 6.x および 8.1 の相互運用性は、以下のシナリオでサポートされています。

- サーバ間の相互運用性
- クライアントとサーバ間の相互運用性

サーバ間の相互運用性

次の表は、2 つの WebLogic Server インスタンス間で相互運用を実現するために利用できるオプションを示しています。

表 1-1 WebLogic Server 間の相互運用性

このサーバへ このサーバから	WebLogic Server 6.0	WebLogic Server 6.1 SP2 以降のサービス パック	WebLogic Server 7.0	WebLogic Server 8.1
WebLogic Server 6.0	RMI/T3 HTTP	HTTP	HTTP Web サービス[1]	HTTP Web サービス[2]
WebLogic Server 6.1 SP2 以降のサービス パック	HTTP	RMI/T3 RMI/IIOP[3] HTTP Web サービス	RMI/T3 RMI/IIOP[4] HTTP Web サービス	RMI/T3[5] RMI/IIOP[6] HTTP Web サービス[7]
WebLogic Server 7.0	HTTP	RMI/T3 RMI/IIOP[8] HTTP	RMI/T3 RMI/IIOP[9] HTTP Web サービス	RMI/T3 RMI/IIOP[10] HTTP Web サービス[11]
WebLogic Server 8.1	HTTP	RMI/T3 RMI/IIOP[12] HTTP	RMI/T3 RMI/IIOP[13] HTTP Web サービス[14]	RMI/T3 RMI/IIOP HTTP Web サービス
Sun JDK ORB クライアント[15]	RMI/IIOP[16]	RMI/IIOP[17]	RMI/IIOP[18]	RMI/IIOP[19]

1. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
2. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
3. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。
4. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。
5. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。マーシャリング中の例外に関する確認済みの問題があります。
6. トランザクションの伝播なし。例外マーシャリングの確認済みの問題。
7. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。

8. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。
9. クラスタ化された URL のサポートはありません。
10. クラスタ化された URL のサポートはありません。
11. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
12. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。マーシャリング中の例外に関する確認済みの問題があります。
13. クラスタ化された URL のサポートはありません。トランザクションの伝播はありません。
14. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
15. このオプションでは、WebLogic Server でホストされているアプリケーション内から JDK ORB へのダイレクトな呼び出しが行われます。
16. JDK 1.3.x のみ。クラスタ化なし。トランザクションの伝播なし。
17. JDK 1.3.x のみ。クラスタ化なし。トランザクションの伝播なし。
18. JDK 1.3.x または 1.4.1。クラスタ化なし。トランザクションの伝播なし。
19. JDK 1.3.x または 1.4.1。クラスタ化なし。トランザクションの伝播なし。

クライアントとサーバ間の相互運用性

次の表は、スタンドアロンの Java クライアント アプリケーションと WebLogic Server インスタンスの間で相互運用性を実現するために利用できるオプションを示しています。

表 1-2 クライアントとサーバ間の相互運用性

このサーバへ このクライアント (スタンドアロン) から	WebLogic Server 6.0	WebLogic Server 6.1	WebLogic Server 7.0	WebLogic Server 8.1
WebLogic Server 6.0	RMI HTTP	HTTP	HTTP Web サービス[1]	HTTP Web サービス[2]
WebLogic Server 6.1	HTTP	RMI/T3 HTTP Web サービス	RMI/T3 HTTP Web サービス[3]	RMI/T3[4] HTTP Web サービス[5]

このサーバへ このクライアント (スタンドアロン) から	WebLogic Server 6.0	WebLogic Server 6.1	WebLogic Server 7.0	WebLogic Server 8.1
WebLogic Server 7.0	HTTP	RMI/T3 RMI/IIOP[6] HTTP	RMI/T3 RMI/IIOP[7] HTTP Web サービス	RMI/T3 RMI/IIOP[8] HTTP Web サービス[9]
WebLogic Server 8.1	HTTP	RMI/T3 RMI/IIOP[10] HTTP	RMI/T3 RMI/IIOP[11] HTTP Web サービス[12]	RMI/T3 RMI/IIOP HTTP Web サービス
Sun JDK ORB クライアント[13]	RMI/IIOP[14]	RMI/IIOP[15]	RMI/IIOP[16]	RMI/IIOP[17]

1. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
2. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
3. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
4. 6.1 SP4 より前のリリースで例外マーシャリングに確認済みの問題があります。
5. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
6. クラスタおよびフェイルオーバのサポートはありません。トランザクションの伝播はありません。
7. クラスタおよびフェイルオーバのサポートはありません。
8. クラスタおよびフェイルオーバのサポートはありません。
9. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
10. クラスタおよびフェイルオーバのサポートはありません。トランザクションの伝播はありません。マーシャリング中の例外に関する確認済みの問題があります。
11. クラスタおよびフェイルオーバのサポートはありません。トランザクションの伝播はありません。マーシャリング中の例外に関する確認済みの問題があります。
12. 「このサーバへ」のバージョンで生成されたポータブル クライアント スタブを使用する必要があります。
13. このオプションでは、クライアント アプリケーション内から JDK ORB へのダイレクトな呼び出しが行われます。
14. JDK 1.3.x のみ。クラスタ化なし。トランザクションの伝播なし。

15. JDK 1.3.x のみ。クラスタ化なし。トランザクションの伝播なし。
16. JDK 1.3.x または 1.4.1。クラスタ化なし。トランザクションの伝播なし。
17. JDK 1.3.x または 1.4.1。クラスタ化なし。トランザクションの伝播なし。

2 RMI over IIOP プログラミング モデル

以下の各節では、さまざまなプログラミング モデルを用いた RMI-IIOP アプリケーションの開発方法について説明します。

- RMI-IIOP プログラミング モデルの概要
- RMI (Java) クライアント使用型 RMI-IIOP アプリケーション
- CORBA/IDL クライアント使用型 RMI-IIOP アプリケーション
- Tuxedo 8.1 ORB 用 WebLogic C++ クライアントの開発
- WebLogic Tuxedo Connector を使用した RMI-IIOP アプリケーション
- RMI-IIOP での EJB の使用
- コード例
- RMI-IIOP と RMI オブジェクトのライフサイクル

RMI-IIOP プログラミング モデルの概要

IIOP は、異機種分散システム間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 に関連する基本的なプログラミング モデルには、RMI クライアントを使用した RMI-IIOP (RMI クライアント使用型 RMI-IIOP) と IDL クライアントを使用した RMI-IIOP (IDL クライアント使用型 RMI-IIOP) の 2 種類があります。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。ただし、これら 2 つのモデルには、異機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としては、はっきりとした違いがあります。IIOP は単に、インタフェースが IDL か Java RMI のどちらかで記述された分散ア

プリケーションの転送プロトコルにすぎません。プログラムを作成する際には、IDL インタフェースと RMI インタフェースのどちら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。

分散アプリケーション環境をどのように作成するかは、いくつかの要因で決まります。RMI-IIOP を採用するためのモデルにはさまざまなものがあり、それらが提供する機能や標準の多くが共通しているので、どのモデルに従えばよいか見通しが立てにくいのが現状です。そこで、各モデルのコンポーネントとメリットを以下の表にまとめてみました。ここでは、利用可能なプログラミング モデル間の相違点をはっきりさせるために、IIOP を使用しない単純な RMI モデルも一緒に示してあります。

表 2-1 RMI プログラミング モデル

クライアント	クライアントの使用言語	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。Java プログラム専用	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 over IIOP RMI クライアント	Java	IIOP	CORBA 2.3 仕様による Objects-by-Value のサポートを利用した 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。IIOP 標準を使用。クライアントに WebLogic クラスは不要。

クライアント	クライアントの使用言語	プロコトル	定義	メリット
RMI over IIOP RMI クライアント	Java	IIOP	CORBA 2.3 仕様による Objects-by-Value のサポートを利用した 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。IIOP 標準を使用。完全にクラスタ化可能だが、クライアントが参照する `weblogic.jar` が必要。
RMI-IIOP CORBA/ IDL クライアント	C++、C、Smalltalk、COBOL (OMG IDL からマッピング可能なあらゆる言語)	IIOP	CORBA 2.3 ORB を使用した 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 Server と、C++、COBOL などで記述されたクライアントとの相互運用性。
RMI-IIOP Tuxedo クライアント	C++、C、COBOL、Java (Tuxedo で OMG IDL にマッピング可能なあらゆる言語)	TGIOP (Tuxedo-General-Inter-Orb-Protocol)	Tuxedo 8.0 ローリング パッチ 15 以降で開発された Tuxedo クライアント	WebLogic Server アプリケーションと Tuxedo クライアント / サービスとの相互運用性

IIOP を使用しない RMI アプリケーション

RMI 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、ネットワーク内の別の場所に存在するオブジェクトへの参照をアプリケーション側で取得することができます。RMI-IIOP モデルはすべて RMI に基づいていますが、IIOP を使用しない基本的な RMI モデルに従う場合には、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。また、独自プロト

コルである T3 を使用して、クライアント側に WebLogic クラスを用意することもあります。RMI アプリケーションの開発については、『WebLogic RMI プログラマーズ ガイド』の「WebLogic RMI の実装」を参照してください。

RMI (Java) クライアント使用型 RMI-IIOP アプリケーション

RMI クライアントを使用した RMI over IIOP (RMI クライアント使用型 RMI over IIOP) では、RMI の機能と標準 IIOP プロトコルが一体化されており、完全に Java プログラミング言語だけで作業できるようになっています。RMI クライアント使用型 RMI-IIOP は Java-to-Java モデルであり、その場合 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。

RMI (Java) クライアント使用型 RMI-IIOP を適用すべき状況

RMI クライアント使用型 RMI-IIOP は J2EE プログラミング モデルを指向しており、RMI の機能と IIOP プロトコルが一体化されています。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 を使用する場合、Java ユーザは RMI インタフェースのプログラムを作成したあと、基礎となる転送メカニズムとして IIOP を使用することができます。RMI クライアントは、J2EE または J2SE コンテナ (ほとんどの場合、JDK 1.3 以降) をホストとする RMI-IIOP 対応 ORB を実行します。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。これは、配布すべきクライアントの量を最小限に抑えるよい方法です。また、通常の WebLogic RMI で用いられる独自の t3 プロトコルを使用する必要もありません。使用するのは、独自仕様でない業界標準に基づいた IIOP なのです。

RMI クライアント使用型 RMI-IIOP アプリケーションの開発

RMI クライアント使用型 RMI-IIOP を用いてアプリケーションを開発するには、以下の手順に従います。

1. リモート オブジェクトのパブリック メソッドを、`java.rmi.Remote` を拡張するインタフェースに定義します。

   このリモート インタフェースには、コードをあまり記述する必要が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。たとえば、インストール済み WebLogic Server の *`SAMPLES_HOME`*`/server/src/examples/iiop/rmi/server/wls` に格納されている Ping サンプルを以下に示します。

   ```
   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;
   }
   ```

2. `interfaceNameImpl` というクラスにインタフェースを実装し、それを JNDI ツリー内にバインドしてクライアントから利用できるようにします。

   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。通常は、実装クラスを WebLogic 起動クラスとしてコンフィグレーションし、そのオブジェクトを JNDI ツリー内にバインドする main メソッドを組み込みます。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);
   ```

3. リモート インタフェースと実装クラスを 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 の概要」を参照してください。

4. 実装クラスに対して WebLogic RMI または EJB コンパイラを実行して、必要な IIOP スタブを生成します。以下のように、IIOP スタブの生成に `-iiop` オプションを使用する必要はもはやないことに注意してください。

   ```
   $ java weblogic.rmic nameOfImplementationClass
   ```

   Pinger サンプルの場合には、*`nameOfImplementationClass`* の部分が `examples.iiop.rmi.server.wls.PingerImpl` になります。

   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。今度は、サーバサイド スケルトンが、その呼び出しを実際のリモート オブジェクト実装に転送します。なお、WebLogic RMI コンパイラで作成される IIOP スタブは、JDK 1.3.1_01 以降の ORB で使用するためのものである点に注意してください。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。

5. ここまでで作成したファイル、すなわち、リモート インタフェース、それを実装するクラス、およびスタブが WebLogic Server の CLASSPATH に含まれていることを確かめます。

6. 初期コンテキストを取得します。

   RMI クライアントは、初期コンテキストを作成しオブジェクトをルックアップする (次のステップを参照) 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。

   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`

   パラメータとして新しい `InitialContext()` に渡す「`Context.INITIAL_CONTEXT_FACTORY`」プロパティの値を設定する際には、これらのクラスのいずれかを使用します。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 を使用する場合には、Sun 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はサポートされているものの、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ないことを承知しておいてください。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。

`InitialContext` の取得 :

```
* JDK1.3 クライアントでは、Properties オブジェクトを
* 次のように使用できる
   */
  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/wls/docs70/jndi/jndi.html
* を参照のこと
```

```
private static Context getInitialContext()
  throws NamingException

{
  return new InitialContext();
}
```

7. 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。

   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 クライアントのステートレス セッション Bean サンプル (配布キットに付属している examples.iiop.ejb.stateless.rmiclient パッケージ) では、RMI クライアントは初期コンテキストを作成し、EJBean ホームをルックアップし、EJBean への参照を取得し、そしてその EJBean のメソッドを呼び出します。

   通常ならオブジェクトを特定のクラス タイプへキャストするような状況ではすべて、javax.rmi.PortableRemoteObject.narrow() メソッドを使用する必要があります。CORBA クライアントからは、リモート インタフェースを実装しないオブジェクトが返される可能性があります。そこで、リモート インタフェースを実装するようにオブジェクトを変換するために ORB から narrow メソッドが提供されるのです。たとえば、EJBean ホームのルックアップとその Home オブジェクトへのキャストを扱うクライアント コードは、以下に示すように、javax.rmi.PortableRemoteObject.narrow() を使用するように修正する必要があります。

   ルックアップの実行 :

```
/**
 * 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、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";
```

8. 以下のようなコマンドでクライアントを実行することで、IIOP を通じてクライアントをサーバに接続します。

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

9. 以下のようにして、クライアント側にセキュリティ マネージャを設定します。

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

   クライアント側の RMI インタフェースをナロー変換するには、サーバからそのインタフェースに適した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、クライアントにセキュリティ マネージャを設定する必要があります。Java セキュリティの詳細については、Sun のサイト (`http://java.sun.com/security/index.html`) を参照してください。なお、`java.policy` ファイルの例を以下に示します。

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

WebLogic RMI-IIOP RMI クライアント使用型 RMI-IIOP アプリケーション

WebLogic Server 7.0 では、完全にクラスタ化可能な RMI-IIOP アプリケーションを開発できる、「ファット」RMI-IIOP RMI クライアントを使用できます。この新しい WebLogic RMI-IIOP RMI クライアントを使用するには、クライアント サイドの CLASSPATH に `weblogic.jar` (*`WL_HOME`*`/server/lib` にある) を含め、`-D weblogic.system.iiop.enableClient=true` コマンド ライン オプションを使用して WebLogic を起動する必要があります。この操作を行わない場合は、このクライアントの開発手順は、「RMI (Java) クライアント使用型 RMI-IIOP アプリケーション」で説明した手順と同じになります。

CORBA/IDL クライアント使用型 RMI-IIOP アプリケーション

CORBA/IDL クライアントを使用した RMI over IIOP (CORBA/IDL クライアント使用型 RMI over IIOP) には、ORB (Object Request Broker) と共に、IDL と呼ばれる相互運用のための言語を作成するコンパイラが必要になります。C、C++、COBOL などは、ORB で IDL にコンパイル可能な言語の例です。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。

CORBA/IDL クライアント使用型 RMI-IIOP を用いると、Java 以外のクライアントと Java オブジェクトとの相互運用が可能になります。CORBA アプリケーションがすでに存在する場合には、CORBA/IDL クライアント使用型 RMI-IIOP モデルに従ってプログラムを作成しなければなりません。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。

以下の各節では、CORBA/IDL クライアント使用型 RMI-IIOP アプリケーションを開発するためのガイドラインを少し示します。

詳細については、Object Management Group (OMG) による以下の仕様を参照してください。

- 詳細については、Java Language Mapping to OMG IDL 仕様を参照してください。
- CORBA/IIOP 2.4.2 仕様

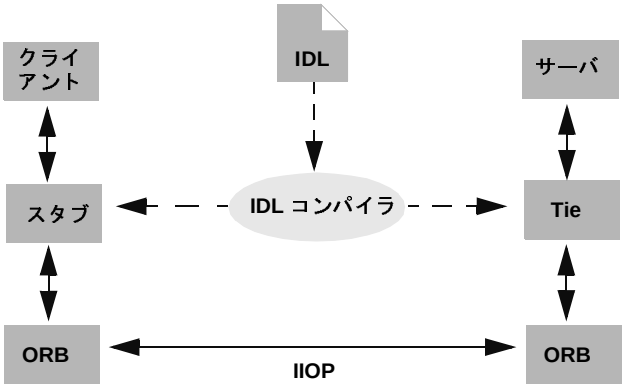
CORBA/IDL クライアントの取り扱い

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といった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、リクエストの転送、および受信した呼び出しのマーシャリングに使用されます。IDL クライアントを使用する場合でも、以降の各節で示すように、プログラミングを行うに当たっては、まず Java リモート インタフェースおよび実装クラスの作成から始め、そのあと IDL を生成して、WebLogic クライアントおよび CORBA クライアントとの相互運用性を実現することを強くお勧めします。IDL でコードを記述したあと、その逆マッピングによって Java コードを作成することも可能ですが、それは難しく、多数のバグを発生させることになるので、WebLogic ではお勧めしません。

IDL と RMI-IIOP モデルの関係を以下の図に示します。

図 2-1 IDL クライアント (CORBA オブジェクト) の関係



Java-to-IDL マッピング

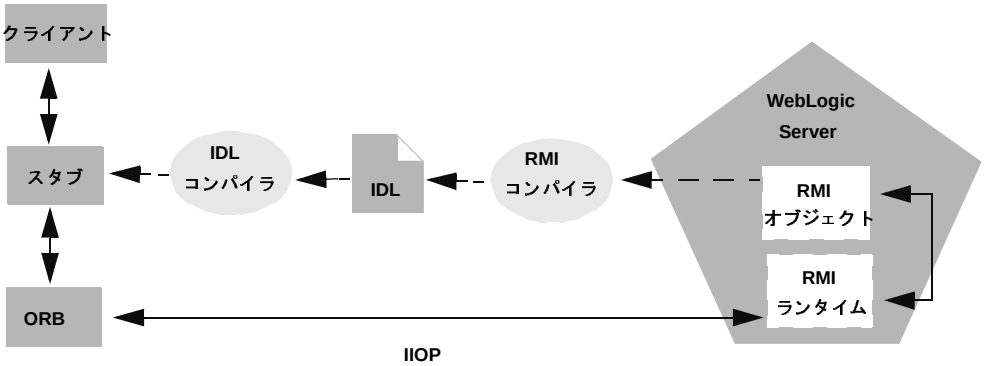
WebLogic RMI では、リモート オブジェクトへのインタフェースは、`java.rmi.Remote` を拡張した Java リモート インタフェースに記述されます。Java-to-IDL マッピング仕様は、IDL が Java リモート インタフェースからどのように作成されるのかを定義しています。WebLogic RMI over IIOP 実装では、`-idl` オプションを付けて WebLogic RMI コンパイラまたは WebLogic EJB コン

パイラで実装クラスを実行します。このプロセスによって、リモート インタフェースの IDL 相当部分が作成されます。そのあと、この IDL を IDL コンパイラでコンパイルして、CORBA クライアントに必要なクラスを生成します。

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、着信した IIOP リクエストを解析し RMI 実行時環境に直接ディスパッチする `CosNaming` サービスを実装しています。

このプロセスを以下の図に示します。

図 2-2 WebLogic RMI over IIOP オブジェクトの関係



Objects-by-Value

Objects-by-Value 仕様により、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できるようになります。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 ORB (Object Request Broker) 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 は比較的少数です。

IDL を使用する 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検討し、それに応じて RMI インタフェースを設計する必要があります。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。Objects-by-Value のサポート状況に関して BEA Systems で検証した ORB とその結果を以下の一覧表に示します。

表 2-2 主な ORB とその Objects-by-Value のサポート状況

ベンダ	バージョン	Objects-by-Value
BEA	Tuxedo 8.1 C++ クライアント ORB	サポートしている
Borland	VisiBroker 3.3、3.4	サポートしていない
Borland	VisiBroker 4.x, 5.x	サポートしている
Iona	Orbix 2000	サポートしている (ただし、この実装ではいくつかの問題が認識されている)

Objects-by-Value の詳細については、3-10 ページの「オブジェクトの値渡しに関する制約」を参照してください。

CORBA/IDL クライアント使用型 RMI-IIOP アプリケーションの開発

CORBA/IDL を使用した RMI over IIOP アプリケーションを開発するには、以下の手順に従います。

1. 2-5 ページの「RMI クライアント使用型 RMI-IIOP アプリケーションの開発」の手順 1 ～ 3 を実行します。
2. `-idl` オプションを付けて WebLogic RMI コンパイラまたは WebLogic EJB コンパイラを実行することで、IDL ファイルを生成します。

   IDL ファイルをコンパイルすると、必要なスタブ クラスが生成されます。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 の実装」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。さらに、Java IDL 仕様については、Java Language Mapping to OMG IDL 仕様を参照してください。

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

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。
`-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` 値タイプをサポートしていない場合に役立つ。
`-idlNoValueTypes`	値タイプに応じた 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 仕様に厳密に準拠した IDL を生成する (ejbc では使用できない)。
`-idlVerbose`	IDL 生成についての詳細な情報を表示する。
`-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、上記のコマンドの場合には、`Hello.idl` ファイルが `/IDL/rmi_iiop` ディレクトリに生成されます。`idlDirectory` オプションが使用されない場合には、IDL ファイルは、スタブ クラスやスケルトン クラスの生成先からの相対パスに生成されます。

3. IDL ファイルをコンパイルして、IDL クライアントでリモート クラスと通信するのに必要なスタブ クラスを作成します。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、WebLogic Server に付属している JDK に含まれている ORB だけで使用するためのものです。

4. IDL クライアントを開発します。

   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。

   RMI-IIOP サンプルの C++ クライアントから抜粋したコード

```
// 文字列をオブジェクトに変換
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 RMI オブジェクトにナロー変換する
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);

// ping を呼び出す
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 仕様の 13.6.7 節を参照) を使用して初期参照が解決されている点に注意してください。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 サービスを利用することで、WebLogic Server アプリケーションは、論理名を使ってオブジェクト参照をアドバタイズできるようになります。CORBA ネーム サービスから提供されるものは以下のとおりです。

- Object Management Group (OMG) の Interoperable Name Service (INS) 仕様の実装
- オブジェクト参照を階層型ネーミング構造 (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

5. IDL クライアント アプリケーションでは、WebLogic Server の JNDI ツリー内の名前をルックアップするように CORBA ネーミング サービスに依頼することで、オブジェクトを見つけることができます。上記の例では、以下のコマンドを使ってクライアントを実行します。

   ```
   Client.exe -ORBInitRef
   NameService=iioploc://localhost:7001/NameService.
   ```

Tuxedo 8.1 ORB 用 WebLogic C++ クライアントの開発

WebLogic C++ クライアントは、Tuxedo 8.1 C++ クライアント ORB を使用して、WebLogic Server で動作している EJB の IIOP リクエストを生成します。このクライアントは、Objects-by-Value および CORBA Interoperable Naming Service (INS) をサポートします。

いつ WebLogic C++ クライアントを使用するのか

以下の状況では、WebLogic C++ クライアントの使用を検討してください。

- サードパーティ製品を使用しないことで開発プロセスを簡略化する
- 既存の C++ クライアントを拡張または修正可能にするクライアントサイド ソリューションを提供する

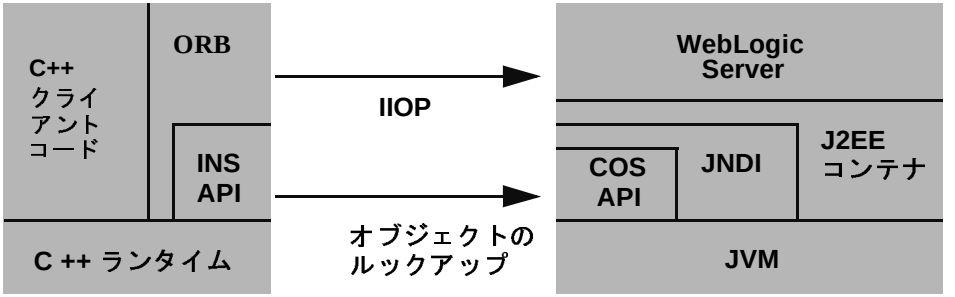
Tuxedo C++ クライアント ORB は Tuxedo 8.1 以上に同梱されていますが、WebLogic C++ クライアントは Tuxedo のライセンスがなくても開発できます。Tuxedo の無償開発版は、BEA ダウンロード センターから入手できます。

WebLogic C++ クライアントの仕組み

WebLogic C++ クライアントは、次のモデルを使用してクライアント リクエストを処理します。

- WebLogic C++ クライアントのコードが、WebLogic Server のサービスを要求する。
  - Tuxedo ORB が、IIOP リクエストを生成します。
  - ORB オブジェクトは初めにインスタンス化され、Object-by-Value データ型をサポートします。
- クライアントは、CORBA Interoperable Name Service (INS) を使用して、JNDI ネーミング サービスにバインドされた EJB オブジェクトをルックアップする。INS を使用して NameService などの初期オブジェクトへの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。

図 2-3 WebLogic C++ クライアントと WebLogic Server の相互運用性



WebLogic C++ クライアントの開発

C++ クライアントを開発するには、次の手順に従います。

1. `-idl` オプションを使って `ejbc` コンパイラを使用して、C++ クライアントと相互運用する EJB をコンパイルします。これで、EJB の IDL スクリプトが生成されます。
2. C++ IDL コンパイラを使用して IDL スクリプトをコンパイルし、CORBA クライアント スタブ、サーバ スケルトン、およびヘッダ ファイルを生成します。C++ IDL コンパイラの使用方法については、「OMG IDL 構文と C++ IDL コンパイラ」を参照してください。
3. EJB がサーバ側の実装を表すので、サーバ スケルトンを破棄します。
4. EJB を実装する C++ クライアントを CORBA オブジェクトとして作成します。CORBA クライアント アプリケーションの作成方法に関する一般的な情報については、『CORBA クライアント・アプリケーションの開発方法』を参照してください。
5. Tuxedo `buildobjclient` コマンドを使用して クライアントをビルドします。

WebLogic C++ クライアントの制限

WebLogic C++ クライアントには、以下の制限があります。

- セキュリティは WebLogic Server セキュリティ サービスを通じて提供する
- サーバ側のトランザクション境界設定のみ提供する

WebLogic C++ クライアントのコード サンプル

WebLogic Server 製品では、WebLogic C++ クライアントのサンプルが提供されます。そのサンプルは、`SAMPLES_HOME\server\examples\src\examples\iiop\ejb` ディレクトリにあります。各サンプルの説明と、ビルド、コンフィグレーション、および実行の手順については、`package-summary.html` を参照してください。コード例は、修正して再利用できます。

WebLogic Tuxedo Connector を使用した RMI-IIOP アプリケーション

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。

いつ WebLogic Tuxedo Connector を使用するのか

Tuxedo でアプリケーションを開発して WebLogic Server に移行する場合、または旧来の Tuxedo システムを新しい WebLogic 環境に統合しようとしている場合には、WebLogic Tuxedo Connector の使用を検討してください。WebLogic Tuxedo Connector を使用すると、Tuxedo の拡張性と信頼性が高い CORBA 環境を利用することができます。

WebLogic Tuxedo Connector の仕組み

コネクタでは、XML コンフィグレーション ファイルを使用して、WebLogic Server からの Tuxedo サービスの呼び出しを可能にしています。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。

以下のマニュアルではそれぞれ、Weblogic Tuxedo Connector と、Tuxedo での CORBA アプリケーションの構築について説明しています。

- 「WebLogic Tuxedo Connector」ページ
- Tuxedo に関する「CORBA のトピック」

WebLogic Tuxedo Connector のコード サンプル

WebLogic Server 製品では、WebLogic Tuxedo Connector IIOP のサンプルが提供されます。そのサンプルは、`SAMPLES_HOME\server\examples\src\examples\iiop\ejb` ディレクトリにあります。各サンプルの説明と、ビルド、コンフィグレーション、および実行の手順については、`package-summary.html` を参照してください。コード例は、修正して再利用できます。

RMI-IIOP での EJB の使用

RMI over IIOP を使用するエンタープライズ JavaBean を実装することで、異機種サーバ環境における以下のような EJB 相互運用性を実現することができます。

- 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 にアクセスできる。
- Java 以外のプラットフォームの CORBA/IDL クライアントから、WebLogic Server 上のいかなるエンタープライズ Bean オブジェクトにもアクセスできる。

CORBA/IDL クライアントを使用する場合には、Java ソース ファイルに定義されている EJB クラスがマッピング情報の源になります。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 の状態と動作で表します。`weblogic.ejbc` ユーティリティの用途は以下のとおりです。

- EJB のクラス、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 ファイルに格納する。
- 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。「CORBA/IDL クライアント使用型 RMI-IIOP アプリケーションの開発」を参照してください。

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。コンパイラはその際に、`idlSources` ディレクトリからソース ファイルを読み込み、CORBA C++ のスタブ ファイルとスケルトン ファイルを生成します。値タイプ以外のすべての CORBA データ型に対しては、生成されるこれらのファイルで十分です (詳細については、「WebLogic RMI-IIOP の制約事項」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。なお、Java-to-IDL マッピング処理には、さまざまな問題が潜んでいます。詳細については、`Java Language Mapping to OMG IDL` 仕様を参照してください。また、Sun からも優れたガイド『Enterprise JavaBeansTM Components and CORBA Clients: A Developer Guide』が公開されています。

以下の例では、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
build\std_ejb_iiop.jar
%APPLICATIONS%\ejb_iiop.jar
```

次に、以下のようにして、EJB インタフェースとクライアント アプリケーションをコンパイルします (この例では CLIENT_CLASSES および APPLICATIONS 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
```

次に、先ほど weblogic.ejbc を使用して作成した IDL ファイルに対して、以下のように IDL コンパイラを実行し、C++ のソース ファイルを作成します。

```
>%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することができます。

RMI-IIOP での EJB の使用方法の詳細については、WebLogic Server に付属している RMI-IIOP サンプルを参照してください。このサンプルは、インストール済み環境の *SAMPLES_HOME*/server/src/examples/iiop ディレクトリに入っています。

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、サンプルの説明を参照してください。特に WebLogic Tuxedo Connector に関するサンプルについては、/wlserver7.0/samples/examples/wtc ディレクトリを参照してください。

次の表に、WebLogic Server 7.0 で提供される RMI-IIOP のサンプルに関する情報を示します。

図 2-4 WebLogic Server 7.0 の IIOP サンプル

サンプル	ORB/ プロトコル	必須条件
`iiop.ejb.entity.cppclient` WebLogic Server のエンティティ セッション Bean を呼び出す C++ クライアントを提供する。	■ Borland Visibroker 4.1 ■ Borland Visibroker 5.0	■ Borland Visibroker 4.1 の場合 : GIOP 1.0 プロトコルを使用する。ユーザは `DefaultGIOPMinorVersion` 属性を追加し、`config.xml` のサーバ Mbean でその値を「1」に設定する必要がある。 ■ Borland Visibroker 5.0 の場合 : `config.xml` のサーバ Mbean でデフォルト ネイティブ コードセットとして `utf-16/iso-8859-1` を指定する。 ■ Borland Visibroker 5.0 の場合 : GIOP 1.2 を使用する。GIOP のバージョンが含まれる完全な corbaloc url を使用する (`Client -ORBInitRef NameService=corbaloc:iiop:1.2@localhost:7001/NameService` など)。
`iiop.ejb.entity.tuxclient` 複雑な valuetype を使用して WebLogic Server のエンティティ セッション Bean を呼び出す Tuxedo クライアントを提供する。	BEA IIOP	Tuxedo 8.x。Tuxedo のライセンスは必要ない。
`iiop.ejb.entity.server.wls` C++ クライアントまたは Tuxedo クライアントとエンティティ Bean の接続を例示する。	該当なし	

サンプル	ORB/ プロトコル	必須条件
`iiop.ejb.stateless.cppclient` WebLogic Server のステートレス セッション Bean を呼び出す C++ CORBA クライアントを提供する。また、WebLogic Tuxedo Connector を使用して Tuxedo サーバへの送信 RMI-IIOP 呼び出しを行う方法も例示する。	■ Borland Visibroker 4.1 ■ Borland Visibroker 5.0	■ Borland Visibroker 4.1 の場合：GIOP 1.0 プロトコルを使用する。ユーザは `DefaultGIOPMinorVersion` 属性を追加し、`config.xml` のサーバ Mbean でその値を「1」に設定する必要がある。 ■ Borland Visibroker 5.0 の場合：`config.xml` のサーバ Mbean でデフォルト ネイティブ コードセットとして `utf-16/iso-8859-1` を指定する。 ■ Borland Visibroker 5.0 の場合：GIOP 1.2 を使用する。GIOP のバージョンが含まれる完全な corbaloc url を使用する (`Client -ORBInitRef NameService=corbaloc:iiop:1.2@localhost:7001/NameService` など)。
`iiop.ejb.stateless.rmiclient` WebLogic Server のステートレス セッション Bean を呼び出す RMI Java クライアントを提供する。また、WebLogic Tuxedo Connector を使用して Tuxedo サーバへの送信 RMI-IIOP 呼び出しを行う方法も例示する。	JDK 1.3.1	JDK 1.3.1 では、セキュリティ ポリシー ファイルがないとサーバにアクセスできない。
`iiop.ejb.stateless.sectuxclient` WebLogic のステートレス セッション Bean を呼び出すセキュアな Tuxedo クライアントを例示する。	BEA IIOP	Tuxedo 8.x。Tuxedo のライセンスは必要ない。

サンプル	ORB/ プロトコル	必須条件
`iiop.ejb.stateless.server.tux` Tuxedo サーバを通じてさまざまなクライアント アプリケーションからステートレス セッション Bean を呼び出す方法を示す。Tuxedo クライアントとの連携で、WebLogic Tuxedo Connector を使用したサーバ間の接続も例示する。	Tuxedo TGIOP	■ Tuxedo 8.x ■ WebLogic Tuxedo Connector を使用する場合は Tuxedo のライセンス ■ サーバ間接続を実現する WebLogic Tuxedo Connector。「RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用」を参照。
`iiop.ejb.stateless.server.wls` さまざまなクライアントを使用して、WebLogic Server で直接、または Tuxedo サーバを通じて間接的にステートレス EJB を呼び出す方法を例示する。	該当なし	
`iiop.ejb.stateless.tuxclient` WebLogic Server で直接ステートレス セッション Bean を呼び出すか、Tuxedo サーバを通じて WebLogic の同じステートレス セッション Bean を呼び出す Tuxedo クライアントを提供する。また、WebLogic Tuxedo Connector を使用して Tuxedo サーバから WebLogic Server への送信 RMI-IIOP 呼び出しを行う方法も例示する。	BEA IIOP	Tuxedo 8.x。Tuxedo のライセンスは必要ない。

サンプル	ORB/ プロトコル	必須条件
`iiop.rmi.cppclient` Tuxedo サーバまたは WebLogic Server のいずれかを呼び出す C++ クライアントを提供する。WebLogic Tuxedo Connector を使用したサーバ間接続も例示します。	■ Borland Visibroker 4.1 ■ Borland Visibroker 5.0 ■ Orbix 2000	■ Borland Visibroker 4.1 の場合：GIOP 1.0 プロトコルを使用する。ユーザは `DefaultGIOPMinorVersion` 属性を追加し、`config.xml` のサーバ Mbean でその値を「1」に設定する必要がある。 ■ Borland Visibroker 5.0 の場合：`config.xml` のサーバ Mbean でデフォルト ネイティブ コードセットとして `utf-16/iso-8859-1` を指定する。 ■ Borland Visibroker 5.0 の場合：GIOP 1.2 を使用する。GIOP のバージョンが含まれる完全な corbaloc url を使用する (`Client -ORBInitRef NameService=corbaloc:iiop:1.2@localhost:7001/NameService` など)。
`iiop.rmi.rmiclient` WebLogic Server との接続を例示する RMI クライアントを提供する。また、WebLogic Tuxedo Connector を使用して WebLogic Server から Tuxedo サーバへの送信呼び出しを行う方法も例示する。	該当なし	サーバにアクセスするにはセキュリティ ポリシー ファイルが必要。

サンプル	ORB/ プロトコル	必須条件
`iiop.rmi.server.tux` Tuxedo サーバを介したさまざまなクライアント アプリケーションからの接続を例示する。Tuxedo クライアントとの連携で、WebLogic Tuxedo Connector を使用したサーバ間の接続も例示する。	Tuxedo TGIOP	■ Tuxedo 8.x ■ WebLogic Tuxedo Connector を使用する場合は Tuxedo のライセンス ■ サーバ間接続を実現する WebLogic Tuxedo Connector。「RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用」を参照。
`iiop.rmi.server.wls` 単純な Ping アプリケーションを使用したさまざまなクライアント、Tuxedo、および WebLogic Server 間の接続を例示する。	該当なし	
`iiop.rmi.tuxclient` Tuxedo サーバとの接続を例示する Tuxedo クライアントを提供する。	BEA IIOP	Tuxedo 8.x。Tuxedo のライセンスは必要ない。

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 ガベージ コレクションによって、未使用および未参照のサーバ オブジェクトのガベージ コレクションが行われます。これにより、多数の未使用オブジェクトを実行することが原因でメモリが不足するおそれが小さくなります。このポリシーでは、クライアントがリモート オブジェクトへの参照を保持しているものの、そのオブジェクトを約 6 分間呼び出してしない場合に、RMI-IIOP で `NoSuchObjectException` エラーが発生することがあります。こうした例外は EJB、つまり JNDI などを経由してサーバ インスタンスによって参照される RMI オブジェクトの場合には発生しません。

RMI-IIOP の J2SE 仕様では、分散ガベージ コレクション (DGC) ではなく、`javax.rmi.PortableRemoteObject` に対して `exportObject()` および `unexportObject()` メソッドを使用して、RMI-IIOP 環境で RMI オブジェクトのライフサイクルを管理することを求めています。ただし、`exportObject()` および `unexportObject()` は、WebLogic Server のデフォルト ガベージ コレクション ポリシーには影響はありません。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場合は、BEA テクニカル サポートにお問い合わせください。

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

以下の各節では、WebLogic Server の RMI-IIOP 用コンフィグレーションに関する概念と手続きについて説明します。

- コンフィグレーションの概要 e
- RMI over IIOP と SSL の併用
- 委託を通じた CORBA クライアントからオブジェクトへのアクセス
- ハードウェア ロード バランサと RMI over IIOP の併用
- WebLogic RMI-IIOP の制約事項
- RMI-IIOP サンプル コード パッケージ
- その他の情報源

コンフィグレーションの概要

CORBA クライアントからクライアント ID を伝達するための十分な標準規格が存在しないため、IIOP を通じて WebLogic Server に接続するすべてのクライアントの ID は、デフォルトで「guest」になります。以下の例に示すように、`config.xml` ファイルにユーザ名とパスワードを設定することで、IIOP を通じて WebLogic Server の個々のインスタンスに接続するすべてのクライアント用に単一の ID をセットアップすることができます。

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

また、config.xml には IIOPEnabled 属性を設定することもできます。これのデフォルト値は "true" で、IIOP のサポートを無効にする場合にかぎり、これを "false" に設定します。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、起動クラスによって 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 Service サービスを実装します。

WebLogic Server 7.0 では、corbaname および corbaloc の各 RMI-IIOP JNDI 参照をサポートしています。CORBA/IIOP 2.4.2 仕様を参照してください。これらの参照の特徴の 1 つは、ある WebLogic Server をホストとする EJB などのオブジェクトを他のアプリケーション サーバから IIOP を通じて利用できるようにすることが可能になるということです。そのため、たとえば、ejb-jar.xml に以下を追加することもできます。

```
<ejb-reference-description>
<ejb-ref-name>WLS</ejb-ref-name>
<jndi-name>corbaname:iiop:1.2@localhost:7001#ejb/j2ee/interop/foo
</jndi-name>
</ejb-reference-description>
```

reference-description スタンザは、ejb-jar.xml に定義されているリソース参照を WebLogic Server 内に用意されている実際のリソースの JNDI 名にマップするものです。ejb-ref-name ではリソース参照名を指定します。これは、EJB プロバイダによって ejb-jar.xml デプロイメント記述子ファイル内に記載される参照です。jndi-name では、WebLogic Server に用意されている実際のリソース 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 7.0 には、GIOP (General-Inter-Orb-Protocol) 1.2 の実装が用意されています。GIOP は、相互運用する ORB 間で交換されるメッセージのフォーマットを規定するものです。これによって、他の多数の ORB やアプリケーション サーバとの相互運用が可能になります。GIOP のバージョンは、corbaname または corbaloc 参照内のバージョン番号で指定することができます。

RMI over IIOP と SSL の併用

セキュア ソケット レイヤ (SSL) プロトコルを使用すれば、RMI または EJB リモート オブジェクトへの IIOP 接続を保護することができます。SSL プロトコルでは、認証を通じて接続の安全性を確保し、オブジェクト間で交換されるデータを暗号化します。WebLogic Server では、以下の方法で RMI over IIOP と SSL を併用することができます。

- CORBA/IDL クライアント ORB (Object Request Broker) を用いる
- Java クライアントを用いる
- BEA Tuxedo を用いる

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

RMI-IIOP と SSL および Java クライアントの併用

1. コールバックを使用する場合には、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。詳細については、「SSL のコンフィグレーション」を参照してください。
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、以下のように、`weblogic.corba.orb.ssl.ORB` クラスを使用する必要もあります。このクラスは、ORB 自身のクラスをラップし、JDK でセキュアな接続を処理する際の問題を解決するものです。

```
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、クライアント接続に対して証明書チェーンを使用するには ：

*

```
*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=Java` クライアントのデジタル証明書のディレクトリ位置
`-Dssl.key=Java` クライアントのプライベート キーのディレクトリ位置

4. **WebLogic Server で使用される SSL プロトコル用のクラスを Java クライアントの CLASSPATH に追加します。**

着信接続の場合 (WebLogic Server から Java クライアントへのコールバック用の接続) には、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します。`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、この情報を入力します。詳細については、「SSL のコンフィグレーション」を参照してください。

BEA Tuxedo クライアントでの SSL プロトコルの使用

SSL プロトコルを使用して BEA Tuxedo クライアントと WebLogic Server 間の通信を保護する方法については、「BEA Tuxedo クライアントおよび WebLogic Server での SSL プロトコルの使用」を参照してください。

委託を通じた CORBA クライアントからオブジェクトへのアクセス

WebLogic Server には、CORBA クライアントから RMI リモート オブジェクトにアクセスできるようにするサービスが用意されています。また、それ以外の方法として、WebLogic Server に CORBA ORB (Object Request Broker) をホストし、

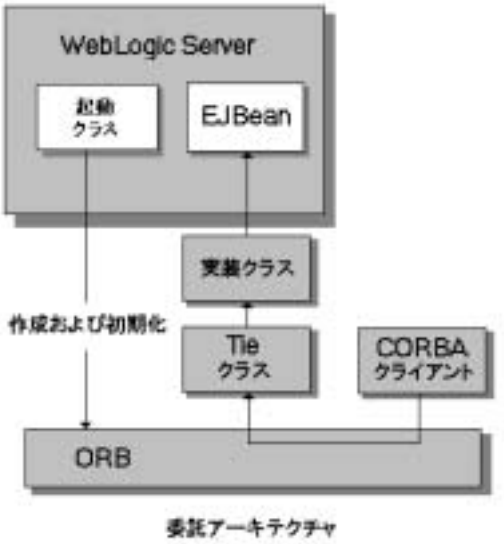
発着信メッセージをその ORB に委託することで、サーバ内でバインド可能なあらゆるオブジェクトを CORBA クライアントから間接的に呼び出せるようにすることもできます。

委託の概要

WebLogic Server をホストとするオブジェクトに CORBA 呼び出しを委託するには、いくつかのオブジェクトが連携してそれを実行する必要があります。それらのオブジェクトを作成する主な手順を以下に示します。

1. WebLogic Server を実行している JVM と共存するように ORB を作成し初期化する起動クラスを作成します。
2. その ORB から着信するメッセージを受け付けるオブジェクトを作成するための IDL (インタフェース定義言語) を作成します。
3. その IDL をコンパイルします。これによって多数のクラスが生成されますが、そのうちの 1 つが Tie クラスです。Tie クラスは、着信呼び出しを処理するためにサーバサイドで用いられ、それらの呼び出しをしかるべき実装クラスにディスパッチします。その実装クラスは、CORBA クライアントに代わって、サーバの接続、適切なオブジェクトのルックアップ、およびそのオブジェクトに対するメソッドの呼び出しを行います。

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。起動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。



委託の例

以下のサンプル コードでは、サーバへの接続、JNDI ツリー内の Foo オブジェクトのルックアップ、および bar メソッドの呼び出しを行う実装クラスを作成しています。このオブジェクトはまた、以下の処理による CORBA 環境の初期化を行う起動クラスでもあります。

- ORB の作成
- Tie オブジェクトの作成
- 実装クラスと Tie オブジェクトの関連付け
- ORB への Tie オブジェクトの登録
- ORB のネーミング サービス内への Tie オブジェクトのバインド

```
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 {
    // 呼び出しの委託先となる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 を作成し初期化する
      ORB orb = ORB.init(args, null);

      // Tie オブジェクトを作成し、ORB に登録する
      _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する
      org.omg.CORBA.Object o = \
      orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(o);

      // オブジェクト参照をネーミング サービスにバインドする

      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();

    }

  }

}
```

起動クラスの実装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。

ハードウェア ロード バランサと RMI over IIOP の併用

注意： この機能は、ハードウェア ロード バランサを使用してブートストラップを行っている場合にのみ正常に動作します。

サービス パック 5 以降の WebLogic Server 7.0 BEA ORB を拡張すると、ブートストラップ時の再接続の強制によるハードウェア ロード バランシングをサポートできます。これにより、ハードウェア ロード バランサによる接続試行のバランシングが可能になります。

ほとんどの場合、いったん接続が確立されると、元の接続を使用して次の NameService ルックアップが実行されます。しかし、この機能はハードウェア ロード バランサにエンド ポイントの再ネゴシエーションを強制するため、既存の接続で処理中のリクエストはすべて破棄されます。

`-Dweblogic.system.iiop.reconnectOnBootstrap` システム プロパティを使用すると、BEA ORB の接続動作を設定できます。有効な値は、次のとおりです。

- true— エンド ポイントの再ネゴシエーションを強制する。
- false— デフォルト値。

ハードウェア ロード バランサを必要とする環境では、このプロパティを true に設定する必要があります。

WebLogic RMI-IIOP の制約事項

以下の各節では、WebLogic RMI-IIOP に関するさまざまな問題点の概要を説明します。

サーバで RMI-IIOP を使用する際の制約

サーバで RMI-IIOP を使用する場合は、以下の制約に注意してください。

- IIOP プロトコルを介して動作する RMI オブジェクトに対するクラスタリングのサポートは、サーバサイド オブジェクトに限定される
- クラスタ化された URL はサポートされない
- IIOP を介して動作しているクラスタ化されたオブジェクトに対するロードバランシングとフェイルオーバは、それらのオブジェクトが WebLogic Server 実行時環境で動作している場合にのみサポートされる

クライアントで RMI-IIOP を使用する際の制約

WebLogic Server と一緒に使用する JDK は、バージョン 1.3.1_01 またはそれ以降でなければなりません。それ以前のバージョンは RMI-IIOP に準拠していません。これら旧バージョンの JDK には以下の問題点があることに注意してください。

- GIOP 1.0 メッセージと GIOP 1.1 プロファイルが IOR で送信される
- EJB 2.0 相互運用性を実現するのに必要なコンポーネント (GIOP 1.2、コードセット ネゴシエーション、UTF-16) をサポートしていない
- 変形メソッド名の取り扱いにバグがある
- 未検査の例外が正しくアンマーシャリングされない
- 値タイプのエンコーディングに関してわずかなバグがある

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。選択肢がいくつかある場合には、WebLogic では仕様に準拠する方をサポートしています。

Java IDL クライアントの開発上の制約

RMI-IIOP を使用する予定であれば、RMI クライアント モデルを用いて Java クライアントを開発することを強くお勧めします。Java IDL クライアントを開発する場合には、名前の衝突やクラスパスの問題が発生するおそれがあり、サーバサイドとクライアントサイドのクラスを分離しておく必要があります。RMI オブジェクトと IDL クライアントのタイプ システムはそれぞれ異なるので、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非常に異なるものになります。

オブジェクトの値渡しに関する制約

オブジェクトを値で渡すには、値タイプを使用する必要があります（詳細については、CORBA/IIOP 2.4.2 仕様の第 5 章を参照してください）。値タイプ は、それが定義または参照されるプラットフォームごとに実装されます。この節では、WebLogic Server 上のエンティティ Bean にアクセスする C++ クライアントのケースを具体的に取り上げながら、複合的な値タイプを渡す際の問題点について説明します
(*SAMPLES_HOME*/server/src/examples/iiop/ejb/entity/server/wls ディレクトリと *SAMPLES_HOME*/server/src/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関することです。たとえば、EJB ファインダにアクセスする際に、Java プログラマは Collection や Enumeration を目にすることになりますが、その基礎となる実装には注意を払いません。これは、JDK ランタイムがネットワークを通じてこれらのクラスをロードするためです。しかし、C++ CORBA プログラマであれば、ネットワークを通じて送られてくるデータ型を把握している必要があります。そうすれば、それに応じた値タイプ ファクトリを登録でき、また ORB でそのデータ型をアンマーシャリングできるようになります。

SAMPLES_HOME/server/src/examples/iiop/ejb/entity/cppclient サンプルにある EJBObjectEnum と Vector がこの例です。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**生成されません**。インタフェースにそれらが現れないからです。このため、ejbc には、リモート インタフェース以外の Java クラスも指定できるようになっています (これは特に、それらのイ

ンタフェースの IDL を生成するのが目的です)。なお、値タイプ ファクトリを登録する方法については、`/iiop/ejb/entity/cppclient` サンプルを参照してください。

シリアライズ可能でありながら `writeObject()` を定義する Java 型は、IDL で記述されるカスタム値タイプにマップされます。値タイプを手動でアンマーシャリングする C++ コードを書く必要があります。その方法の例としては、*`SAMPLES_HOME`*`/server/src/examples/iiop/ejb/enity/tuxclient/ArrayList_i.cpp` を参照してください。

注意： Tuxedo を使用する場合には、IDL コンパイラに `-i` 修飾子を指定することで、*`FileName`*`_i.h` および *`FileName`*`_i.cpp` という実装ファイルを作成させることができます。たとえば、次の構文の場合、`TradeResult_i.h` および `TradeResult_i.cpp` という実装ファイルが作成されます。

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

結果として生成されるソース ファイルには、値タイプに対するアプリケーション定義の操作が実装されています。実装ファイルは、CORBA クライアント アプリケーションに組み込まれます。

RMI-IIOP サンプル コード パッケージ

`examples.iiop` パッケージは、`SAMPLES_HOME/server/src/samples/examples/iiop` ディレクトリにあり、その中には、さまざまなクライアントとアプリケーションとの接続性について説明するサンプルが用意されています。これらのサンプルでは、RMI-IIOP での EJB の使用、C++ クライアントへの接続、および Tuxedo サーバとの相互運用性のセットアップに関する例を示しています。詳細については、サンプルの説明を参照してください。特に WebLogic Tuxedo Connector に関するサンプルについては、`/wlserver6.1/samples/examples/wtc` ディレクトリを参照してください。

その他の情報源

WebLogic RMI-IIOP は、RMI の完全な実装を意図したものです。ご使用のバージョンに該当しそうなその他の考慮事項については、『リリース ノート』を参照してください。

- 『WebLogic JNDI プログラマーズ ガイド』
- 『WebLogic RMI プログラマーズ ガイド』
- 「Java Remote Method Invocation (RMI)」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 に関するドキュメント
- OMG 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