BEA WebLogic jCOM

ユーザーズ ガイド

WebLogic jCOM バージョン 6.1
マニュアルの日付：2001 年 11 月 1 日

著作権

限定的権利条項

商標または登録商標

目次

このマニュアルの内容

1. BEA WebLogic jCOM の概要

2. WebLogic jCOM プログラミング モデル

3. プログラミング

4. アプリケーションのデプロイ

5. WebLogic jCOM ツール

このマニュアルの内容

このマニュアルでは、BEA WebLogic jCOM のブリッジ機能およびアーキテクチャについて説明します。このマニュアルでは、WebLogic jCOM ブリッジの一方向、つまり COM クライアントから Java オブジェクトへの呼び出し（COM から Java）を中心に説明します。Java クライアントから COM コンポーネントへのアクセスについての詳細は、『WebLogic jCOM リファレンス』を参照してください。

このマニュアルの構成は次のとおりです。

- 第 1 章「BEA WebLogic jCOM の概要」では、WebLogic jCOM とそのアーキテクチャについて概説します。
- 第 2 章「WebLogic jCOM プログラミング モデル」では、WebLogic jCOM を使用して COM と Java 間の通信を実装するいくつかの方法について説明します。
- 第 3 章「プログラミング」では、WebLogic jCOM を使用して COM クライアントから Java コンポーネントにアクセスするためにプログラマが行う必要があることについて説明します。
- 第 4 章「アプリケーションのデプロイ」では、アプリケーションの実行前に行っておく必要がある手順について説明します。
- 第 5 章「WebLogic jCOM ツール」では、よく使用する WebLogic jCOM ツールについて説明します。

対象読者

このマニュアルは、WebLogic Server で WebLogic jCOM 機能を使用するシステム構築者、アプリケーション開発者、および管理者を対象としています。

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（http://www.beasys.co.jp/index.html）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）から無料で入手できます。

関連情報

BEA の Web サイトでは、WebLogic jCOM のすべてのドキュメントを提供しています。http://edocs.beasys.co.jp/e-docs/wls61 を参照してください。

サポート情報

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

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。本バージョンの BEA WebLogic jCOM について不明な点がある場合、または BEA WebLogic jCOM のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	同時に押すキーを示す。

表記法	適用
斜体	強調または本のタイトルを示す。

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

斜体の等幅テキスト	コード内の変数を示す。 例 : `String CustomerName;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。
[ ]	構文内の任意指定の項目を示す。 例 : `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例 : `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`

表記法	適用
. . .	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 BEA WebLogic jCOM の概要

以下の節では、WebLogic jCOM を使用して WebLogic Server が COM クライアントにアクセスできるようにする方法について説明します。

- BEA WebLogic jCOM とは
- Java と COM の衝突
- 衝突に対する WebLogic jCOM のソリューション
- WebLogic jCOM の動作
- WebLogic jCOM の機能

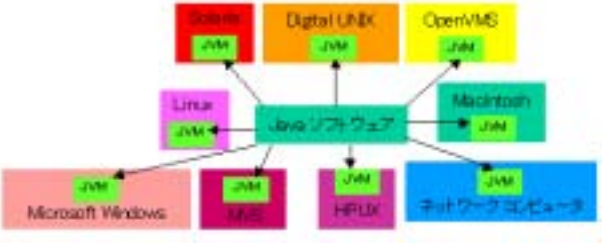
BEA WebLogic jCOM とは

BEA WebLogic jCOM は、双方向型の COM-Java ブリッジ ツールです。WebLogic jCOM を使用すると、あたかも Java オブジェクトであるかのように Component Object Model（COM）コンポーネントにアクセスし、あたかも COM コンポーネントであるかのように pure Java オブジェクトにアクセスできます。

Java と COM の衝突

Java のサポート

多くの企業が Java を採用している理由の 1 つは、Java プログラムが標準 Java 仮想マシン（JVM）をサポートする任意のプラットフォームで実行できるため、ソフトウェアの開発コストとデプロイメント コストを大幅に削減できることです。



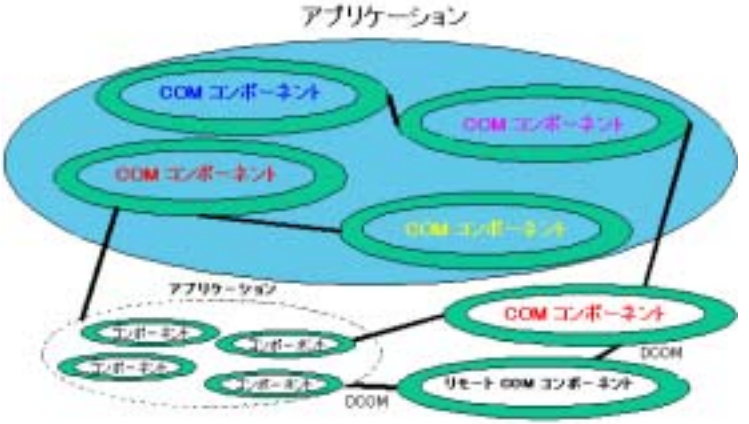
「純粋な」Java ソフトウェアを開発するという原則は非常に重要です。多くのソフトウェア開発者は、特定のプラットフォームにしばられるソフトウェアを使用することを極端に嫌います。

COM のサポート

ソフトウェア コンポーネントはオブジェクト指向ソフトウェア開発の自然な進歩の結果であり、アプリケーションのパーツを独立したコンポーネントに分離することを可能にします。このようなコンポーネントは、複数のアプリケーション間で共有できます。また、コンポーネントには厳密に定義されたインタフェースを介してのみアクセスできるので、それらのコンポーネントの実装を変更しても、それらを使用するアプリケーションに影響を与えることがありません。

Microsoft の広く使用されている Component Object Model（COM）は、コンポーネント統合のためのバイナリ標準を定義したものです。COM を使用すると、Visual C++ で作成されたアプリケーションから Visual BASIC などで作成された COM コンポーネントにアクセスできます。

これまでに非常に多くの COM コンポーネントが作成されており、それらを購入することができます。実際、Microsoft Windows で開発された最近のソフトウェアのほとんどは、コンポーネントを使用して作成されています。



Microsoft Windows 上で行う現在のソフトウェア設計では、事実上、アプリケーションを COM コンポーネント ベースのアプローチで設計することが求められます。そのメリットの 1 つは、Distributed COM（DCOM）を使用することによって、同じホスト上のみならずリモート ホストからもアプリケーションの機能の一部にアクセスできることです。

Java と COM の衝突

どこでも動作する Java ソフトウェアを作成したいという要望と COM ソフトウェア コンポーネントを再利用しなければならないという要件の間で衝突が発生する場合があります。

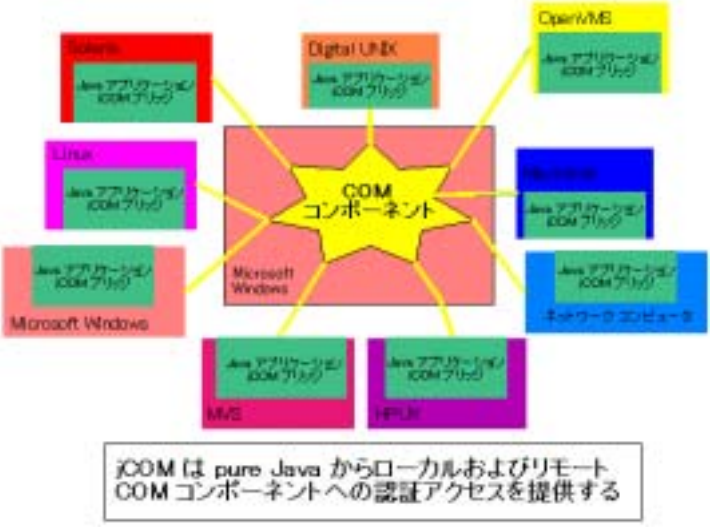
こうした衝突は、対立する 2 つの陣営が存在する企業において表面化します。一方の陣営は、pure Java ソフトウェアを特定のプラットフォームに限定することを極端に嫌う Java 開発者であり、他方の陣営は、COM コンポーネントの利用および再利用を主張する Windows 開発者です。

しかしながら、標準 JVM で動作する、pure Java で開発された新しいアプリケーションから既存のアプリケーションにアクセスしなければならないケースも多々あります。こうした既存のアプリケーションの多くは、Microsoft Windows 環境

で開発されています。また、これらのアプリケーションは厳密に設計されているので、それらは COM コンポーネントを公開することによって自身の機能を他のアプリケーションに公開します。

衝突に対する WebLogic jCOM のソリューション

BEA WebLogic jCOM は、「一度書けばどこでも動作する」という Java のメリットと COM コンポーネントの再利用によるメリットをつなぐブリッジを提供することで、このような衝突を解決します。



内部的には、WebLogic jCOM は、Microsoft が COM コンポーネントのアクセス用に定義した標準のプラットフォーム独立メカニズムを使用します。

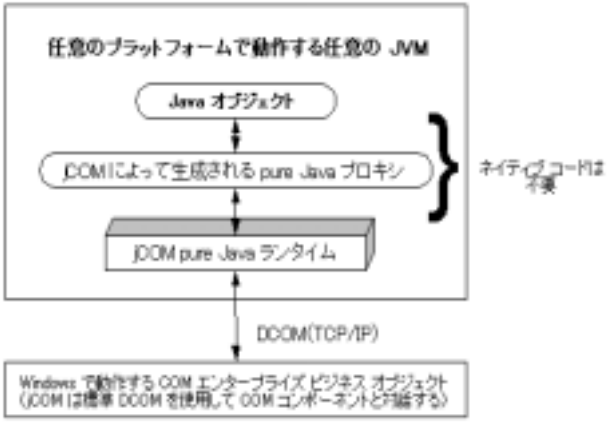
WebLogic jCOM の動作

WebLogic jCOM の Java-COM ブリッジ機能を使用すると、任意のオペレーティング システム環境で動作する Java オブジェクトから COM に、またはその反対にアクセスできます。

COM 開発者は、Java オブジェクトにコールバックを行うことができます。WebLogic jCOM は、Java オブジェクトをリモート対応にします。これにより、そのすべてのパブリック メソッドとメンバー変数に COM からアクセスできます。

Java プログラマには、WebLogic jCOM によって COM コンポーネントが Java オブジェクトのように見えるので、COM プロパティ、メソッド、およびイベントが Java プロパティ、メソッド、およびイベントとして示されます。

図 1-1 WebLogic jCOM は任意のプラットフォーム上の任意の Java 仮想マシンで動作し、ネイティブ コードを必要としない



WebLogic jCOM の pure Java ランタイムは、Distributed COM layered over Remote Procedure Calls（RPC）を使用して COM と対話します。これ自身は、TCP/IP 上にレイヤされます。このため、最低レベルでは WebLogic jCOM は完全に標準の Java ネットワーキング クラスを使用します。

WebLogic jCOM の機能

WebLogic jCOM の主要な機能は以下のとおりです。

- WebLogic jCOM は、COM コンポーネントと Java を結ぶ双方向ブリッジを提供します。COM クライアントからエンタープライズ Java Bean（EJB）と Java オブジェクトにアクセスでき、Java クライアントから COM コンポーネントにアクセスできます。
- WebLogic jCOM はクライアントによってアクセスされるデータ型の存在を隠すので、最も適切な Java オブジェクトと COM コンポーネントが動的にマップされます。
- WebLogic jCOM は、レイト バインディングとアーリー バインディングの両方をサポートしています。
- すべての JVM に対応しています。
- JVM が動作するマシンには、WebLogic jCOM ランタイム（つまり jCOM.jar のみ）しか必要ありません。jCOM.jar ファイルは pure Java であり、ネイティブ コードは不要です。
- COM コンポーネントをホストするマシン上にネイティブ コードは必要ありません。内部的には、WebLogic jCOM は Windows DCOM ネットワーク プロトコルを使用してローカルおよびリモート COM コンポーネントと pure Java 環境間の通信を実現します。ただし、WebLogic jCOM は、Windows プラットフォームで動作するときのパフォーマンスを最大限に高めるオプションの「ネイティブ モード」をサポートしています。
- WebLogic jCOM Java ランタイム（jcom.jar）のサイズは約 595K です。
- WebLogic jCOM は、イベント処理をサポートしています。たとえば、WebLogic jCOM では、標準 COM イベント メカニズムを使用して VB から Java イベントにアクセスし、Java オブジェクトは COM コンポーネント イベントをサブスクライブできます。
- WebLogic jCOM では、認証をまったく使用しないか、または接続レベル認証を使用して Java から COM コンポーネントにアクセスできます。

2 WebLogic jCOM プログラミング モデル

以下の節では、WebLogic jCOM を使用して Java と COM 間の通信を実装するいくつかの方法、および各実装の長所と短所について説明します。

- WebLogic jCOM コンポーネント
- 用語の定義
- WebLogic jCOM プログラミング モデル

WebLogic jCOM コンポーネント

BEA WebLogic jCOM を使用すると、Microsoft Windows 上で動作する COM クライアントから任意のオペレーティング システム上で動作する Java オブジェクト（Java コンポーネントなど）にアクセスできます。

つまり、WebLogic jCOM は、Java COM と Java を結ぶブリッジを作成するための実行時環境と必要なコンポーネントを提供するキットです。これから見ていくすべての例では、このブリッジは WebLogic Server マシン上に存在します。

`jCOM.jar` ランタイムは、ブリッジをコンパイルおよび実行するために必要です。ブリッジ自体はユーザによって作成される Java プロセスで、クライアントとサーバ間での COM と Java 間の通信を処理するためにバックグラウンドで動作します。以下の例に示す `JCOMBridge.java` ブリッジは、それらの例のためのブリッジとしてだけでなく、独自のブリッジを作成するための土台としても役立ちます。

実行時ファイルの他にも、WebLogic jCOM にはクライアントおよびサーバ環境をコンフィグレーションするためのツールとコンポーネントが用意されています。

Java 仮想マシンは、常に VB プロセスへの独立したプロセスとして（場合によっては別のマシン上で）実行されることに注意してください。WebLogic jCOM は、VB ウィンドウ内部に JavaBean を配置するために使用できません。これらは独立したプロセスに属するからです。しかし、Visual Basic プログラミングからパブリック Java クラスのすべてのパブリック メソッドおよびフィールドにアクセスできます。

COM と Java 間の通信は、以下のプログラミング モデルを使用して実現できます。

- DCOM ゼロ クライアント プログラミング モデル
- DCOM レイト バインド プログラミング モデル
- DCOM アーリー バインド プログラミング モデル
- DCOM レイト バインド カプセル化プログラミング モデル
- ネイティブ レイト バインド プログラミング モデル
- ネイティブ アーリー バインド プログラミング モデル

用語の定義

DCOM モード

DCOM（Distributed Component Object Model）モードは、Component Object Model（COM）を使用して異なるコンピュータ上のオブジェクト間の通信をサポートします。

ネイティブ モード

ネイティブ モードは、ネイティブ コード（DLL）を使用します。これらの DLL はローカルのオペレーティング システムと CPU 向けにコンパイルおよび最適化されているので、パフォーマンスが向上します。

ゼロ クライアント インストール

ゼロ クライアント インストールとは、Windows クライアント マシンにソフトウェアをインストールする必要がないことを意味します。デプロイメントのオーバーヘッドは存在しません。

レイト バインディング

レイト バインディングは、別のアプリケーションのオブジェクトへのアクセスを提供する手段です。レイト バインド アクセスでは、アクセスするオブジェクトに関する情報はコンパイル時に提供されず、これらのオブジェクトは実行時に動的に評価されます。このため、アクセスするメソッドとプロパティが実際に存在するかどうかは、プログラムを実行してみて初めて分かります。

アーリー バインディング

アーリー バインディングは、別のアプリケーションのオブジェクトへのアクセスを提供する手段です。アーリー バインド アクセスでは、アクセスするオブジェクトに関する情報はプログラムのコンパイル中に提供され、それらのオブジェクトはコンパイル時に評価されます。この方法では、サーバ アプリケーションは型ライブラリを提供し、クライアント アプリケーションはクライアント システムにロードするためにそのライブラリを識別する必要があります。

WebLogic jCOM プログラミング モデル

COM と Java 間の通信は、DCOM モードかネイティブ モードを使用して実装できます。どちらの方法でも、プログラミング モデルを選択できます。

すべての実装では、COM クライアントは Microsoft Windows プラットフォームで動作します。Java コンポーネントは、Java™ 2 Platform, Enterprise Edition 1.3 および WebLogic Server 6.1 上で動作します。クライアントとサーバ / ブリッジは、1 つのシステム実装の同じシステムで動作します。

DCOM ゼロ クライアント プログラミング モデル

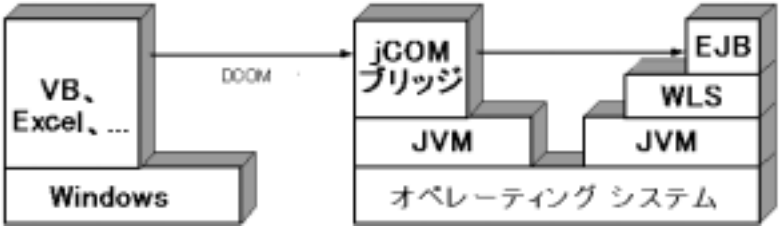
DCOM ゼロ クライアント インストールは実装が簡単です。クライアント サイドには WebLogic jCOM 固有のソフトウェアは必要ありません。

ゼロ クライアント インストールではレイト バインディングが使用されるので、Java コンポーネントの変更に関してレイト バインディングと同じ柔軟性が得られます。ただし、この実装ではブリッジの位置とポート番号を COM クライアントにハード コード化する必要があります。このため、ブリッジの位置が変更された場合、ソース コード内の参照を再生成して変更する必要があります。

DCOM ゼロ クライアント インストールは、DCOM アーリー バインディング実装に比べてエラーが発生しやすくなります。

DCOM レイト バインド実装に比べ、DCOM ゼロ クライアント インストールでは実行時の初期化パフォーマンスが向上します。

図 2-1 一般的な DCOM ゼロ クライアント実装の実行時インストール



クライアントが Java コンポーネントにアクセスするには、以下のことを行います。

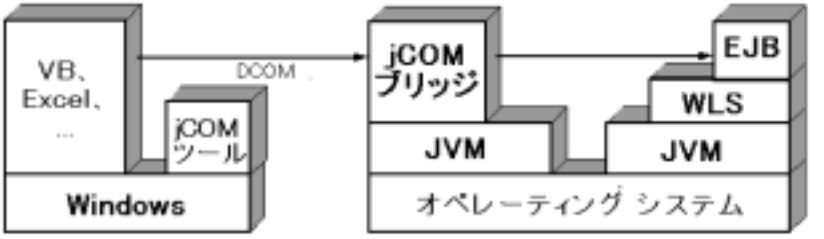
1. `objref` モニカ文字列を使用して、WebLogic jCOM ブリッジの位置をクライアントにハード コード化します。`objref` モニカを生成します。このモニカによって、WebLogic jCOM ブリッジの IP アドレスとポートがエンコーディングされます。ブリッジ接続が確立されると、クライアントは COM オブジェクトを Java コンポーネント内のインタフェースにリンクできます。
2. この COM オブジェクトへのクライアント内の参照は、あたかも COM メソッドであるかのように Java メソッドを使用します。

ゼロ クライアント インストール実装の例については、インストールした WebLogic jCOM の samples\examples ディレクトリにある Zero Client Installation サンプルを参照してください。

DCOM レイト バインド プログラミング モデル

DCOM レイト バインド アクセスは実装が簡単です。また、オブジェクト参照が実行時にのみ評価されるので実装に柔軟性があります。しかし、コンパイル時に型チェックを実行できないので、エラーが発生しやすくなります。

図 2-2 一般的な DCOM レイト バインド実装の実行時インストール



クライアントが Java コンポーネントにアクセスするために、以下のことが行われます。

1. クライアント ソース コードで、COM オブジェクトを Java コンポーネント内のインタフェースにリンクします。クライアントがインタフェースにアクセスするには、WebLogic jCOM ツールを使用して JVM を Windows レジストリに登録し、それを WebLogic jCOM ブリッジの IP アドレスとポートに関連付ける必要があります。
2. この COM オブジェクトへのクライアント内の参照は、あたかも COM メソッドであるかのように Java メソッドを使用します。

DCOM レイト バインド実装の例については、インストールされている WebLogic jCOM の samples\examples ディレクトリにある Late Bound Implementation サンプルを参照してください。

DCOM アーリー バインド プログラミング モデル

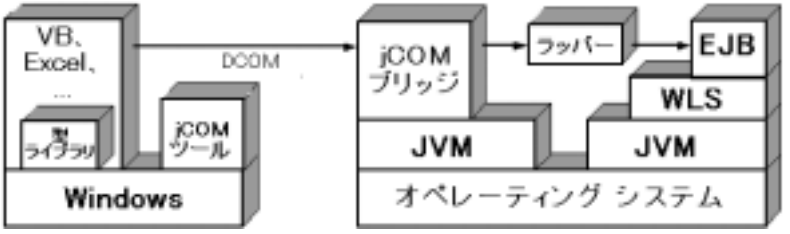
DCOM アーリー バインド アクセスは、実装が複雑です。これは、型ライブラリとラッパーを生成する必要があるからです。型ライブラリはクライアント サイドで、ラッパーはサーバ サイドで必要となります。クライアントとユーザが別個のマシンで動作している場合、型ライブラリとラッパーは同じマシン上で生成して、必要とされているシステムにコピーしなければなりません。

アーリー バインド アクセスは、レイト バインド アクセスが備えている柔軟性に欠けます。Java コンポーネントに変更を加えた場合、ラッパーと型ライブラリを再生成する必要があるからです。これに代わるハイブリッドな方法については、レイト バインド カプセル化プログラミング モデルを参照してください。

アーリー バインド アクセスでは、信頼性が向上します。コンパイル時の型チェックによりデバッグが容易になり、ユーザは型ライブラリを参照することができます。

DCOM レイト バインド実装に比べ、DCOM アーリー バインド実装では実行時のトランザクション パフォーマンスが向上しますが、実行時の初期化は低速化します。

図 2-3 一般的な DCOM アーリー バインド実装の実行時インストール



クライアントが Java コンポーネントにアクセスするには、以下のことを行います。

1. クライアント ソース コードで、生成した型ライブラリを使用して COM オブジェクトを宣言します。クライアントがインタフェースにアクセスするには、WebLogic jCOM ツールを使用して JVM を Windows レジストリに登録し、それを WebLogic jCOM ブリッジの IP アドレスとポートに関連付ける必要があります。型ライブラリがクライアント サイドで登録され、登録された JVM 名にリンクされます。

2. この COM オブジェクトへのクライアント内の参照は、あたかも COM メソッドであるかのように Java メソッドを使用します。

DCOM アーリー バインド実装の例については、インストールされている WebLogic jCOM の `samples\examples` ディレクトリにある Early Bound Implementation サンプルを参照してください。

DCOM レイト バインド カプセル化プログラミング モデル

アーリー バインディングを使用済みで、Java コンポーネント インタフェースを変更するか、または単に将来の可能性に備えたい場合は、レイト バインド カプセル化を採用することをお勧めします。Java コンポーネントを変更すると、ラッパーの再コンパイルに加え、クライアント サイドで新しい型ライブラリを作成する必要があります。ラッパーと型ライブラリは相互に依存し合っており、バージョン固有のものだからです。こうした状況は、Java コンポーネント インタフェースの変更が特定のクライアント プログラムと無関係な場合にも起こり得ます。このため、この Java コンポーネントにアクセスするすべてのクライアント システムに新しい型ライブラリを再配布しなければならない場合もあります。

レイト バインド カプセル化方法を使用すると、アーリー バインド プログラミングの主要なメリットはそのままに、ラッパーと型ライブラリを必要としない柔軟なレイト バインド設計を実装できます。

次に説明する基本手順は Visual Basic クライアント向けのものですが、どのようなプログラミング言語にも適用することができます。

1. 型ライブラリを作成し、1 つまたは複数の VB クラスの内部で使用するオブジェクトのインタフェースを実装します。
2. このクラスは、オブジェクトのレイト バインド バージョンと通信するためのラッパーとして機能できます。
3. インタフェースが確立されたら、型ライブラリを削除します（これが存在する限り、実装はアーリー バインドのまま変わらない）。

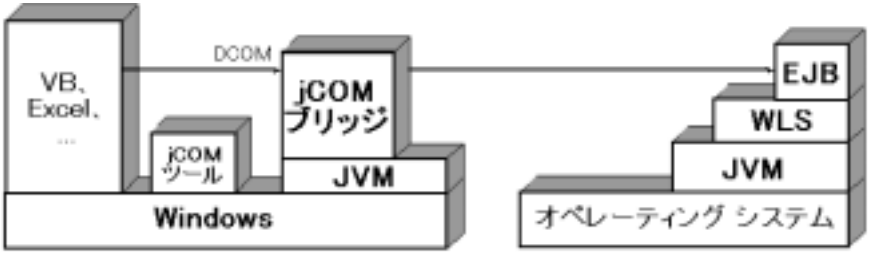
ネイティブ レイト バインド プログラミング モデル

ネイティブ レイト バインド アクセスは実装が簡単です。また、オブジェクト参照が実行時にのみ評価されるので実装に柔軟性があります。しかし、コンパイル時に型チェックを実行できないので、エラーが発生しやすくなります。

作成されているネイティブ ライブラリは Windows 専用なので、ネイティブ レイト バインド アクセスでは WebLogic jCOM ブリッジをクライアント マシンにインストールする必要があります。

DCOM レイト バインド実装に比べ、ネイティブ レイト バインド実装では実行時の初期化パフォーマンスが大幅に向上します。

図 2-4 一般的なネイティブ レイト バインド実装の実行時インストール



ネイティブ モードの実装の詳細については、『WebLogic jCOM リファレンス』の「ネイティブ モード」を参照してください。

ネイティブ アーリー バインド プログラミング モデル

ネイティブ アーリー バインド アクセスは、実装が複雑です。これは、型ライブラリとラッパーを生成する必要があるからです。型ライブラリはクライアント サイドで、ラッパーはサーバ サイドで必要となります。クライアントとユーザが別個のマシンで動作している場合、型ライブラリとラッパーは同じマシン上で生成して、必要とされているシステムにコピーしなければなりません。

アーリー バインド アクセスは、レイト バインド アクセスが備えている柔軟性に欠けます。Java コンポーネントに変更を加えた場合、ラッパーと型ライブラリを再生成する必要があるからです。

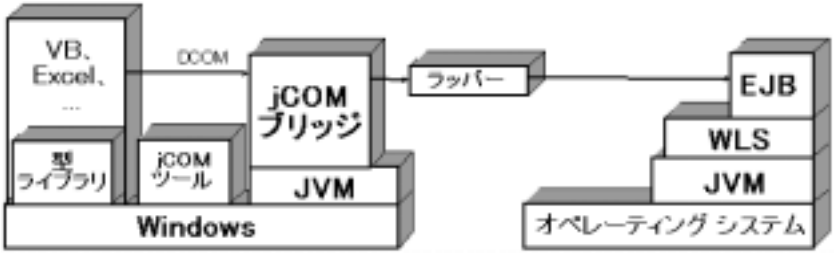
アーリー バインド アクセスでは、信頼性が向上します。コンパイル時の型チェックによりデバッグが容易になり、ユーザは型ライブラリを参照することができます。

作成されているネイティブ ライブラリは Windows 専用なので、ネイティブ アーリー バインド アクセスでは WebLogic jCOM ブリッジをクライアント マシンにインストールする必要があります。

DCOM アーリー バインド実装に比べ、ネイティブ レイト バインド実装では実行時の初期化パフォーマンスが大幅に向上します。

ネイティブ レイト バインド実装に比べ、ネイティブ アーリー バインド実装では実行時の初期化が低速になりますが、トランザクション パフォーマンスが若干向上します。

図 2-5　一般的なネイティブ アーリー バインド実装の実行時インストール



ネイティブ モードの実装の詳細については、『WebLogic jCOM リファレンス』の「ネイティブ モード」を参照してください。

3　プログラミング

WebLogic jCOM は、Windows DCOM ネットワーク プロトコルを使用してローカルおよびリモート COM コンポーネントと pure Java 環境間の通信を実現します。以下の節では、WebLogic jCOM を使用して COM クライアントから Java コンポーネントにアクセスするためにプログラマが行う必要があることについて説明します。

- サーバサイドのプログラミング要件
- クライアント サイドのプログラミング要件
- Java オブジェクトのインスタンス化のインターセプト
- コンストラクタを使用した COM からの Java オブジェクトのインスタンス化
- シングルトン Java オブジェクトの実装

サーバサイドのプログラミング要件

サーバ サイドのプログラミング要件は、ほとんどのプログラミング モデルの場合、コンフィグレーションとブリッジのコンパイルおよびアクティブ化に限られます。アーリー バインド プログラミング モデルまたはレイト バインド カプセル化プログラミング モデルを実装する場合、ラッパーと型ライブラリも生成する必要があります。

ブリッジ ファイルの `JCOMBridge.java` は、WebLogic jCOM に用意されています。このファイルは前述のすべての例で使用されており、独自のブリッジ ファイルを作成するための土台としても使用できます。

ブリッジをサーバ サイドに構築するには、実装するプログラミング モデルに関係なく、以下のことを行う必要があります。

1. WebLogic Server を起動します。
2. PATH 環境変数をインストールされている WebLogic Server と JDK 1.3 のルート ディレクトリに設定します（setEnv ファイルを実行する）。
3. ブリッジ ファイルの次の行を編集して、WebLogic Server のリスン ポート（デフォルトは 7001）を正確に指定します。

   ```
   env.put(Context.PROVIDER_URL, "t3://localhost:7001")
   ```

4. 構築スクリプトの build.xml を実行して、WebLogic jCOM ブリッジ ファイルをコンパイルします。そのためには、次のコマンドを実行します。

   ```
   ant
   ```

5. PATH 環境変数をインストールされている WebLogic jCOM のルート ディレクトリに設定することによって、WebLogic jCOM 環境をコンフィグレーションします。
6. ブリッジ ファイル JCOMBridge.java を実行する次の文を使用して、サーバ サイドのブリッジを起動します。

   ```
   java -classpath %CLASSPATH% -DJCOM_DCOM_PORT=7050 JCOMBridge
   ```

   ここで 7050 は、WebLogic jCOM ブリッジがクライアントとサーバ間の通信をリスンする必要があるポートです。

   ブリッジ ファイル、JCOMBridge.java は、WLS TCP/IP アドレスとリスン ポートを定義し、JVM 名を JNDI に登録します。

アーリー バインディングまたはレイト バインド カプセル化を実装する場合は、さらに次の手順を行う必要があります。

1. java2com ツールの実行およびブリッジとラッパーのコンパイルに必要な生成環境を設定します。CLASSPATH には、jCOM.jar のパスとアクセスする Java クラスのパスが含まれている必要があります。
2. java2com ツールを使用して、Java コンポーネントの Java ラッパーと IDL ファイルを生成します。

   ```
   java com.bea.java2com.Main
   ```

この行を実行すると、java2com ウィンドウが開きます。[Java Classes & Interfaces] フィールドに、使用するクラス（ブリッジ クラスを含む）の名前を入力する必要があります。

3. 生成されたクラスと JCOMBridge.class ファイルが CLASSPATH に含まれていることを確認し、Java ラッパー ファイルをコンパイルします。

   ```
   javac Output Directory\*.java
   ```

クライアント サイドのプログラミング要件

クライアント システムで必要なプログラミング手順については、以下のプログラミング モデルを実装する方法を見ていきます。

- DCOM ゼロ クライアント プログラミング モデル
- DCOM レイト バインド プログラミング モデル
- DCOM アーリー バインド プログラミング モデル
- DCOM レイト バインド カプセル化プログラミング モデル
- ネイティブ モード プログラミング モデル

DCOM ゼロ クライアント プログラミング モデル

DCOM ゼロ クライアント プログラミング モデルの実装に必要な基本的なクライアント サイト プログラミング手順は次のとおりです（VB クライアントが WebLogic Server 上の EJB にアクセスする場合）。

1. Java クラス `com.bea.jcom.GetJvmMoniker` を使用して、ブリッジの場所のコード化された参照（objref）を生成します。パラメータとして、サーバ マシンのフルネームまたは TCP/IP アドレス、およびブリッジにアクセスできるポートを指定します。次に例を示します。

   ```
   java com.bea.jcom.GetJvmMoniker mymachine.mycompany.com 7050
   ```

   または

   ```
   java com.bea.jcom.GetJvmMoniker localhost 7050
   ```

2. クライアント ソース コードで、生成された objref を含む次の文を使用してブリッジにアクセスします。

   ```
   Set objBridge = GetObject("objref:generatedobjref")
   ```

3. これに続き、JNDI を使用してサーバからすべてのオブジェクトを要求できます。次に例を示します。

   ```
   Set objHome = objBridge.get("JVMName:jndi name of ejb")
   ```

DCOM レイト バインド プログラミング モデル

DCOM レイト バインド プログラミング モデルの実装に必要な基本的なクライアント サイト プログラミング手順は次のとおりです（VB クライアントが WebLogic Server 上の EJB にアクセスする場合）。

1. VB クライアントのソース コードで、COM オブジェクトを EJB のインタフェースにリンクします。この VB クライアントのソース コードでは、EJB のホーム インタフェース、`objHome` の COM バージョンの宣言に注意してください。COM オブジェクトは、サーバ サイドにある EJB のホーム インタフェースのインスタンスにリンクされます。

   ```
   Dim objHome As Object

   Private Sub Form_Load()

   'Handle errors

   On Error GoTo ErrOut

   'Bind the EJB's HomeInterface object via JNDI

   Set objHome = GetObject("JVMName:jndi name of ejb")
   ```

   `GetObject` は、WebLogic Server 上の JNDI ルックアップを通してオブジェクトを取得します。JVM（「*JVMName*」）は、手順 3. の説明のとおりレジストリに登録する必要があります。

2. このオブジェクトの以後の参照は COM オブジェクトを指しているように見えますが、実際はあたかも COM メソッドであるかのように Java メソッドを使用します。

3. クライアント システムで、regjvm ツールを使用してローカル Java 仮想マシンを登録します。そのためには、そのマシン名を Windows レジストリに追加し、TCP/IP アドレスと、WebLogic jCOM が WebLogic jCOM 要求をリスンするクライアントとサーバ間の通信ポートに関連付けます。次に例を示します。

   ```
   regjvmcmd JVMName localhost[7050]
   ```

DCOM アーリー バインド プログラミング モデル

DCOM アーリー バインド プログラミング モデルの実装に必要な基本的なクライアント サイト プログラミング手順は次のとおりです（VB クライアントが WebLogic Server 上の EJB にアクセスする場合）。

1. 生成された IDL（「サーバサイドのプログラミング要件」を参照）をクライアント システムにコピーします。

2. Microsoft IDL コンパイラ、midl.exe を使用して、IDL ファイルを型ライブラリにコンパイルします。

   ```
   midl generatedIDLFileName.idl
   ```

   コンパイルの結果、名前が同じで拡張子が .tlb の型ライブラリが生成されます。

3. 型ライブラリを登録し、サービスする JVM を設定します。次に例を示します。

   ```
   regtlb /unregisterall
   regtlb generatedIDLFileName.tlb JVMName
   ```

   上の最初の行は、登録済みの型ライブラリ バージョンの登録を解除するために regtlb.exe を呼び出します。2 番目の行は、新しくコンパイルされた型ライブラリを登録し、その型ライブラリにリンクされる JVM の名前（"*JVMName*"）を指定します。WebLogic jCOM ランタイムでは、型ライブラリで定義されたオブジェクト呼び出しを適切なラッパー クラスにリンクするためにこの情報が必要となります。

4. これで、クライアントは型ライブラリにアクセスできます。VB プロジェクトをロードし、[Projects] メニューの [Reference] を選択します。スクロールして型ライブラリを見つけ、そのチェック ボックスをオンにします。[OK] をクリックします。

5. 型ライブラリを使用することにより、オブジェクトは「As Object」として宣言されません。

   ```
   Dim objCOM As generatedIDLFileName.generated class name
   ```

   たとえば、完全修飾 Java クラスが *examples.ejb.basic.containerManaged.AccountHome* の場合、生成されるクラス名は *ExampleEjbBasicContainerManagedAccountHome* となります。

6. オブジェクトのメソッドとプロパティに関する情報にアクセスするには、次の行も必要です。

   ```
   Dim objTemp As Object

   Dim objBridge As New generatedIDLFileName.COMtoWebLogic

   Set objTemp = GetObject("JVMName:jndi name of ejb")

   Set objHome = objBridge.narrow(objTemp,"fully qualified java
   class")
   ```

   objTemp オブジェクトはレイト バインド モデルを使用して EJB オブジェクトの参照を取得することに注意してください。レイト バインド オブジェクトはブリッジの「narrow」メソッドに受け渡され、次にアーリー バインド オブジェクトが与えられます。

7. クライアント システムで、regjvm ツールを使用してローカル Java 仮想マシンを登録します。そのためには、そのマシン名を Windows レジストリに追加し、TCP/IP アドレスと、WebLogic jCOM が WebLogic jCOM 要求をリスンするクライアントとサーバ間の通信ポートに関連付けます。

   ```
   regjvmcmd JVMName localhost[7050]
   ```

DCOM レイト バインド カプセル化プログラミング モデル

レイト バインド カプセル化を使用すると、アーリー バインド プログラミングの主要なメリットはそのままに、ラッパーと型ライブラリを必要としない柔軟なレイト バインド モデルを実装できます。

たとえば、Visual Basic クライアントが EJB にアクセスする場合は、次のことを行う必要があります。

1. 生成された IDL（「サーバサイドのプログラミング要件」を参照）をクライアント システムにコピーします。
2. Microsoft IDL コンパイラ、midl.exe を使用して、IDL ファイルを型ライブラリにコンパイルします。

   ```
   midl generatedIDLFileName.idl
   ```

   コンパイルの結果、名前が同じで拡張子が .tlb の型ライブラリが生成されます。
3. 型ライブラリを登録し、サービスする JVM を設定します。次に例を示します。

   ```
   regtlb /unregisterall
   regtlb generatedIDLFileName.tlb JVMName
   ```

   上の最初の行は、登録済みの型ライブラリ バージョンの登録を解除するために regtlb.exe を呼び出します。2 番目の行は、新しくコンパイルされた型ライブラリを登録し、その型ライブラリにリンクされる JVM の名前（"*JVMName*"）を指定します。WebLogic jCOM ランタイムでは、型ライブラリで定義されたオブジェクト呼び出しを適切なラッパー クラスにリンクするためにこの情報が必要となります。
4. [Project ｜ References] ダイアログを使用して、Visual Basic プロジェクトからこの型ライブラリを参照します。
5. からのクラス モジュールをプロジェクトに追加し、そのソース ウィンドウを開きます。
6. EJB の参照に使用する型「Object」のモジュール レベル変数を追加します。

7. クラス モジュールのソース ウィンドウの上部にある [object] および [procedure] プルダウン メニューを使用して、`Class_Initialize` メソッドを追加し、このメソッドの中に、モジュール レベル オブジェクト変数に EJB の参照を割り当てるために必要なソース コードを挿入します。必要な初期化ソースについては、「DCOM レイト バインド プログラミング モデル」を参照してください。

8. クラス ソースの上部で `Implements` キーワードを使用して、アクセスする必要があるオブジェクトの参照を実装します。EJB オブジェクトを参照するには、型ライブラリの名前の後にドットを付けます（`Implements` *`generatedIDLFileName.generated class name`*）。

   たとえば、完全修飾 Java クラスが *`examples.ejb.basic.containerManaged.AccountHome`* の場合、*`generated class name`* は `ExampleEjbBasicContainerManagedAccountHome` となります。

9. [object] および [procedure] プルダウン メニューを使用して、アクセスする必要があるすべての EJB オブジェクトからメソッドとプロパティを選択します。これにより、選択したメソッドとプロパティ用のスケルトン ソース コードが生成されるはずです。

10. これらのメソッドおよびプロパティ宣言内で、作成したモジュール レベル オブジェクトを介して EJB のメソッドとプロパティにアクセスするレイト バインド コードを挿入します。

11. 「Implements」エントリをクラス ソースの冒頭から削除し、[Project | References] ダイアログから型ライブラリを削除します。クラスは、型ライブラリ（そしてサーバ上のラッパー）に依存せずに EJB にアクセスできるようになります。

これが済んだら、作成したクラスをインスタンス化し、あたかもアーリー バインドであるかのようにすべての EJB 機能にアクセスできます。VP クライアント ソースに採用されたメソッドとプロパティのインタフェースが静的である限り、EJB に対するどのような変更も VB プロジェクトに影響を与えません。

ネイティブ モード プログラミング モデル

ネイティブ モードでは、COM クライアントはクライアントと同じマシン上で動作している Java オブジェクトにアクセスします。WebLogic jCOM は、ネイティブ コードを使用してこの対話を容易にします。ネイティブ モードの詳細については、『WebLogic jCOM リファレンス』の「ネイティブ モード」を参照してください。

Java オブジェクトのインスタンス化のインターセプト

Java オブジェクトのインスタンス化を制御する場合、`com.bea.jcom.Instanciator` インタフェースを実装するクラスを作成します。このインタフェースは、次のような 1 つのメソッドを持ちます。

```
public Object instanciate(String javaClass) throws
com.bea.jcom.AutomationException;
```

Jvm.register(...) を呼び出すときに参照を 2 番目のパラメータとしてインスタンシエータに渡します。

```
com.bea.jcom.Jvm.register("MyJvm", myInstanciator);
```

WebLogic jCOM によって使用されるデフォルトのインスタンシエータは次のとおりです。

```
public final class DefaultInstanciator implements
com.bea.jcom.Instanciator {
public Object instanciate(String javaClass)
throws com.bea.jcom.AutomationException {
try {
return Class.forName(javaClass).newInstance();
} catch(Exception e) {
e.printStackTrace();
throw new AutomationException(e);
}
}
}
```

たとえば、これは VB と EJB のブリッジです（Sun の JNDI Tutorial を基礎にしている）。

```
import javax.naming.*;
import java.util.Hashtable;
import com.bea.jcom.*;

public class VBtoEJB {
public static void main(String[] args) throws Exception {
Jvm.register("ejb", new EjbInstanciator());
Thread.sleep(10000000);
}
}

class EjbInstanciator implements Instanciator {
Context ctx;

EjbInstanciator() throws NamingException {
Hashtable env = new Hashtable(11);
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://... TBS ...");
ctx = new InitialContext(env);
}

public Object instanciate(String javaClass) throws
AutomationException {
try {
try {
return Class.forName(javaClass).newInstance();
} catch(Exception e) {}
return ctx.lookup(javaClass);
} catch (Throwable t) {
t.printStackTrace();
throw new AutomationException(new Exception("Unexpected: " + t)); }
}
}
```

上のコードをコンパイルしたら、次のようにそれをマシン（development.company.com）上で実行します。

```
java -DJCOM_DCOM_PORT=4321 VBtoEJB
```

次に、Windows マシン上で次のように WebLogic jCOM regjvmcmd コマンドを使用します。

```
regjvmcmd ejb development.company.com[4321]
```

次に、VB から以下を使用します。

```
Set myEjb = GetObject("ejb:cn=ObjectName")
MsgBox myEjb.someProperty
myEjb.myMethod "a parameter"
```

コンストラクタを使用した COM からの Java オブジェクトのインスタンス化

COM にはコンストラクタの概念はありません。1 つの方法は、デフォルト コンストラクタを定義して、適切なパラメータを取る静的メンバーを定義し、オブジェクトをインスタンス化して返すことです。

```
public class MyClass {
       public MyClass() {}
       public MyClass(String p1, int p2, double p3) {
       ...
       }

public static MyClass createMyClass(String p1, int p2, double p3) {
       return new MyClass(p1, p2, p3);
}
}
```

もう 1 つの方法は、WebLogic jCOM のインスタンス化インターセプト機能を使用することです。JVM を登録するときに、オブジェクトの参照を受け渡すことができます。このオブジェクトのクラスは、`GetObject("MyJvm:MyClass")` を使用するときに呼び出される特別な Java jCOM インタフェースを実装します。コロンの後はすべて渡すので、実際には `GetObject("MyJvm:MyClass(1, 2, three, 4.0")` を行って、インターセプタで渡される文字列を解析し、適切なコンストラクタを呼び出します。

シングルトン Java オブジェクトの実装

COM 用語では、オブジェクトのインスタンスが常に 1 つだけ存在する場合、そのオブジェクトはシングルトンです。CreateInstance を呼び出すたびに、同じオブジェクトの参照を取得します。このため、すべてのクライアントが同じインスタンスにアクセスすることが保証されます。

Java オブジェクトのインスタンス化を制御することで、COM クライアントからアクセスできるシングルトン Java オブジェクトを実装できます。次に、`mySingletonClass` というクラスのイニシエータの例を示します。

```
import java.util.*;
import com.bea.jcom.*;

public class COMtoJava {
public static void main(String[] args) throws Exception {
try {
Jvm.register("MyJvmId", new
SingletonInstanciator("MySingletonClass"));

while (true) { // 永遠に待機
Thread.sleep(100000);
}
} catch (Exception e) {
System.out.println(e.getMessage());
e.printStackTrace();
}
}
}

class SingletonInstanciator implements Instanciator {
String singletonClassname;
static Object singletonObject = null;

SingletonInstanciator(String singletonClassname) {
try {
this.singletonClassname = singletonClassname;
if (singletonObject == null) {
System.out.println("SingletonInstanciator: creating the singleton
[" + singletonClassname + "]");
// シングルトンを初期化
Class classObject = Class.forName(singletonClassname);
singletonObject = classObject.newInstance();
}
} catch (Exception e) {
System.out.println(e.getMessage());
e.printStackTrace();
}
}

public Object instanciate(String javaClass) throws
AutomationException {
try {
System.out.println("instanciate for " + javaClass);

// 要求がシングルトンの作成の場合、既存のインスタンスを返す
if (javaClass.equals(singletonClassname)) {
return singletonObject;
} else {
Class classObject = Class.forName(javaClass);
return classObject.newInstance();
}
}
catch (Exception e)
{
System.out.println("Failed to instanciate class " + javaClass);
System.out.println(e.getMessage());
e.printStackTrace();
```

```
System.out.println("Throwing exception back to caller.");
throw new AutomationException(e);
}
}
}
```

MySingletonClass 実装のサンプルを以下に示します。

```
public class MySingletonClass {
public MySingletonClass() {
System.out.println("MySingletonClass constructor called.");
}

public int Method1(int val) {
return val + 1;
}
}
```

上の両方をコンパイルしたら、次のように COMtoJava をマシン（development.company.com）上で実行します。

```
java -DJCOM_DCOM_PORT=4321 COMtoJava
```

次に、Windows マシン上で次のように WebLogic jCOM regjvmcmd コマンドを使用します。

```
regjvmcmd MyJvmId development.company.com[4321]
```

次に、VB から以下を使用します。

```
Set objMySingleton1 = GetObject("MyJvmId:mySingletonClass")
Set objMySingleton2 = GetObject("MyJvmId:mySingletonClass")
MsgBox objMySingleton1 & objMySingleton2
```

これにより、同じオブジェクトの 2 つの参照が作成されます。

4　アプリケーションのデプロイ

以下の節では、アプリケーションの実行前に行っておく必要がある手順について説明します。

- デプロイメント オプション
- アプリケーションのデプロイ

デプロイメント オプション

WebLogic jCOM を使用して COM クライアントから Java オブジェクトにアクセスする場合、使用する実装に応じて以下の 5 つの異なるデプロイメント シナリオが存在します。

- DCOM ゼロ クライアント実装のデプロイ
- DCOM レイト バインド実装のデプロイ
- DCOM アーリー バインド実装のデプロイ
- DCOM レイト バインドカプセル化実装のデプロイ
- ネイティブ モード実装のデプロイ

また、デプロイメントは、次の中で複数の WebLogic Server を使用することによって影響を受ける場合があります。

- サーバのクラスタ化

使用する実装に適したインストールを選択した場合、コンフィグレーション、デプロイメント、および実行時に必要なすべての WebLogic jCOM ファイルは既に適切な場所に格納されています。ユーザが作成したファイルは、適切な場所に手動でコピーする必要があります。

WebLogic jCOM を使用して Java クライアントから COM オブジェクトにアクセスするときのデプロイメントの詳細については、『WebLogic jCOM リファレンス』を参照してください。

アプリケーションのデプロイ

DCOM ゼロ クライアント実装のデプロイ

DCOM ゼロ クライアント実装では、クライアント システム上での実装は必要ありません。ただし、クライアントから WebLogic jCOM ブリッジには、サーバの場所（IP とポート）のハードコード化された参照を介してアクセスすることに注意してください。この参照は、クライアント ソース コードに存在します。サーバの場所が変更された場合、COM クライアントがこのサーバと通信するためには、この `objref` を再登録してクライアントのソース コードに挿入する必要があります。

サーバ インストール プロセス（通常 server install）は、必要な WebLogic jCOM コンポーネントを実行時にサーバにインストールします。

- WebLogic jCOM bridge: `jcom.jar`

EJB が WebLogic Server にデプロイされ、サーバがアクティブ化されたら、コンパイルされたブリッジをアクティブ化できます。サーバの TCP/IP アドレスとリスン ポートに対する変更は、必ずブリッジファイル自体に反映されなければなりません。

`objref` を生成し、ブリッジをアクティブ化する方法については、「プログラミング」の章のゼロ クライアント インストールの例を参照してください。

DCOM レイト バインド実装のデプロイ

DCOM レイト バインド実装の場合、クライアント インストール プロセス（通常クライアント インストール）が必要なコンポーネントを実行時にサーバにインストールします。

- WebLogic jCOM Moniker : `JintMk.dll`

クライアントのコンフィグレーションに必要なものの他に、次のものが必要です。

- `regjvm`

アプリケーションの実行前に、`regjvm` または `regjvmcmd` を使用して JVM を登録しておく必要があります。`regjvm` と `regjvmcmd` の詳細については、「WebLogic jCOM ツール」を参照してください。

注意： サーバの場所を変更した場合、JVM を再登録する必要があります。これを行う前に、古いエントリの登録を解除する必要があります。`regjvmcmd` ツールは古いエントリを同名の新しいエントリで上書きしないからです。古いエントリの登録を解除するには、コマンド ライン ツールの `regjvmcmd` を使用するか、または GUI ツール `regjvm`（どちらも `jCOM\bin` ディレクトリに存在する）を使用します。

サーバ インストール プロセス（通常 server install）は、必要な WebLogic jCOM ブリッジ ファイルを実行時にサーバにインストールします。

- WebLogic jCOM bridge: `jcom.jar`

EJB が WebLogic Server にデプロイされ、サーバがアクティブ化されたら、コンパイルされたブリッジをアクティブ化できます。サーバの TCP/IP アドレスとリスン ポートに対する変更は、必ずブリッジファイル自体に反映されなければなりません。

JVM を登録してブリッジをアクティブ化する方法については、「プログラミング」の章のレイト バインド実装の例を参照してください。

DCOM アーリー バインド実装のデプロイ

DCOM アーリー バインド実装の場合、クライアント インストール プロセス（通常 client install）が必要な jCOM ツールを実行時にサーバにインストールします。

- WebLogic jCOM Moniker : `JintMk.dll`

クライアントのコンフィグレーションに必要なものの他に、次のものが必要です。

- `regtlb`
- `regjvm`

`java2com` ツールによって生成された型ライブラリが実行時にクライアント システムに存在し、`CLASSPATH` 環境変数がその型ライブラリの格納先のディレクトリを指していることを確認する必要があります。型ライブラリは、WebLogic jCOM ブリッジから取得したリモート オブジェクトのアーリー バインディングで必要です。型ライブラリをクライアントに登録するには、`regtlb` ツールを使用します。`java2com` および `regtlb` ツールの詳細については、「WebLogic jCOM ツール」を参照してください。

アプリケーションの実行前に、`regjvm` または `regjvmcmd` を使用して JVM を登録しておく必要があります。`regjvm` と `regjvmcmd` の詳細については、「WebLogic jCOM ツール」を参照してください。

注意 : サーバの場所を変更した場合、JVM を再登録する必要があります。これを行う前に、古いエントリの登録を解除する必要があります。`regjvmcmd` ツールは古いエントリを同名の新しいエントリで上書きしないからです。古いエントリの登録を解除するには、コマンド ライン ツールの `regjvmcmd` を使用するか、または GUI ツール `regjvm`（どちらも `jCOM\bin` ディレクトリに存在する）を使用します。

サーバ インストール プロセス（通常 server install）は、必要な WebLogic jCOM ブリッジ ファイルを実行時にサーバにインストールします。

- WebLogic jCOM bridge: `jcom.jar`

`java2com` ツールによって生成されたラッパー クラスが実行時にサーバ システムに存在し、`CLASSPATH` 環境変数がそれらの格納先ディレクトリを指していることを確認する必要があります。ラッパー クラスは、カプセル化するために作成した Java オブジェクトとアーリー バインド通信を可能にします。

EJB が WebLogic Server にデプロイされ、サーバがアクティブ化されたら、コンパイルされたブリッジをアクティブ化できます。サーバの TCP/IP アドレスとリスン ポートに対する変更は、必ずブリッジファイル自体に反映されなければなりません。

型ライブラリと JVM の登録、およびブリッジのアクティブ化についての詳細は、「プログラミング」の章のアーリー バインド実装の例を参照してください。

DCOM レイト バインドカプセル化実装のデプロイ

レイト バインド カプセル化は開発時にはアーリー バインディングを、実行時にはレイト バインディングを使用するので、レイト バインドカプセル化実装のデプロイはレイト バインド実装のデプロイとほぼ同じです。レイト バインディングが実行時に実装されるようにするには、開発時に参照される型ライブラリにクライアントからアクセスできないようにする必要があります。これにより、サーバ サイドのラッパーにもアクセスできなくなります。

DCOM レイト バインド実装の場合、クライアント インストール プロセス（通常 client install）が必要な WebLogic jCOM ツールを実行時にサーバにインストールします。

- WebLogic jCOM Moniker : `JintMk.dll`

クライアントのコンフィグレーションに必要なものの他に、次のものが必要です。

- `regjvm`

開発時に参照される型ライブラリが実行時にクライアントから見えないようにする必要があります。そのようにしない限り、実装はアーリー バインドのままです。

アプリケーションの実行前に、`regjvm` または `regjvmcmd` を使用して JVM を登録しておく必要があります。`regjvm` と `regjvmcmd` の詳細については、「WebLogic jCOM ツール」を参照してください。

注意 : サーバの場所を変更した場合、JVM を再登録する必要があります。これを行う前に、古いエントリの登録を解除する必要があります。`regjvmcmd` ツールは古いエントリを同名の新しいエントリで上書きしな

いからです。古いエントリの登録を解除するには、コマンド ライン ツールの `regjvmcmd` を使用するか、または GUI ツール `regjvm`（どちらも `jCOM\bin` ディレクトリに存在する）を使用します。

サーバ インストール プロセス（通常 server install）は、必要な WebLogic jCOM ブリッジ ファイルを実行時にサーバにインストールします。

- WebLogic jCOM bridge: `jcom.jar`

EJB が WebLogic Server にデプロイされ、サーバがアクティブ化されたら、コンパイルされたブリッジをアクティブ化できます。サーバの TCP/IP アドレスとリスン ポートに対する変更は、必ずブリッジファイル自体に反映されなければなりません。

ネイティブ モード実装のデプロイ

ネイティブ モードでは、COM クライアントはクライアントと同じマシン上で動作している Java オブジェクトにアクセスします。WebLogic jCOM は、ネイティブ コードを使用してこの対話を容易にします。ネイティブ モードの詳細については、『WebLogic jCOM リファレンス』の「ネイティブ モード」を参照してください。

サーバのクラスタ化

サーバ クラスタ内で複数の WLS マシンを使用する場合、デプロイメント方法に影響が及ぶ場合があります。フェイルオーバー機能を保持するには、すべての WebLogic jCOM ファイルをクライアント システムにデプロイすること（上記のネイティブ モード デプロイメントとほぼ同じ）をお勧めします。クラスタ内の 1 台のマシンだけに WebLogic jCOM ブリッジを配置した場合、システムは障害の影響を受けやすくなります。ブリッジ マシンに障害が発生した場合、すべての WebLogic jCOM 機能を失うことになります。WebLogic jCOM ブリッジをクライアント システムに配置するとこの危険を回避され、クラスタはフェイルオーバ機能を通常どおり使用できます。WebLogic jCOM の将来のリリースは、WebLogic Server と緊密に統合される予定です。これにより、クラスタ化フェイルオーバ機能を自動的に保持できるようになります。

5 WebLogic jCOM ツール

以下の節では、よく使用する WebLogic jCOM ツールについて説明します。

- regjvm GUI ツール
- regjvmcmd コマンド ライン ツール
- java2com ツール
- regtlb ツール

regjvm GUI ツール

WebLogic jCOM を使用すると、COM をサポートする言語で、あたかも COM オブジェクトであるかのように Java オブジェクトにアクセスできます。

そのためには、Java オブジェクトが実行される JVM の参照を（COM クライアント マシン上に）登録する必要があります。`regjvm` ツールを使用すると、マシン上ですべての JVM 参照を作成および管理できます。

注意： `regjvm` ツールでは、古いエントリと同名の新しいエントリが入力されても、古いエントリは上書きされません。このため、通信するマシンのホスト名またはポートを変更する必要がある場合は、古いエントリの登録を解除しなければなりません。そのためには、コマンド ライン ツールの `regjvmcmd.exe` か、または GUI ツールの `regjvm.exe` を使用します（どちらも `jCOM\bin` ディレクトリに格納されています）。

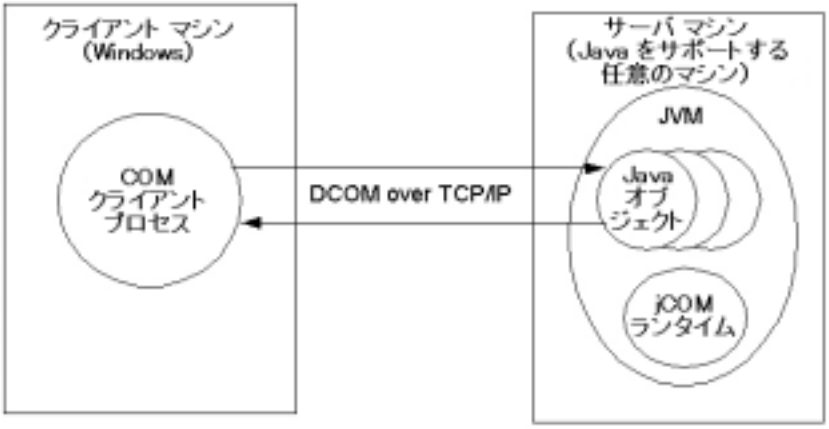
JVM のモード

COM クライアントから JVM には、以下のモードを使用してアクセスできます。

- DCOM モード
- ネイティブ モード（プロセスの外部）
- ネイティブ モード（プロセスの内部）

DCOM モード

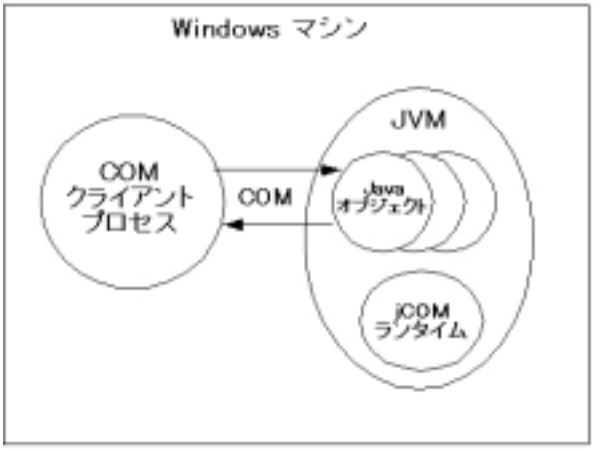
DCOM モードでは、Java サーバ サイド上にネイティブ コードは不要です。このため、Java コードは Java 仮想マシンがインストールされている Unix マシンや他のマシン上に配置できます。Windows クライアント マシン上で JVM を登録する場合、サーバのホスト マシン名（ローカル コンポーネント用のローカルホストの場合もあります）とポート番号を定義します。



JVM 内の Java コードは、com.bea.jcom.Jvm.register("<jvm id>") を呼び出す必要があります。ここで <jvm id> は、regjvm で定義した JVM の ID です。また JVM は、指定した <jvm id> 用に regjvm ツールで定義したポートに設定された JCOM_DCOM_PORT プロパティで開始されなければなりません。

ネイティブ モード（プロセスの外部）

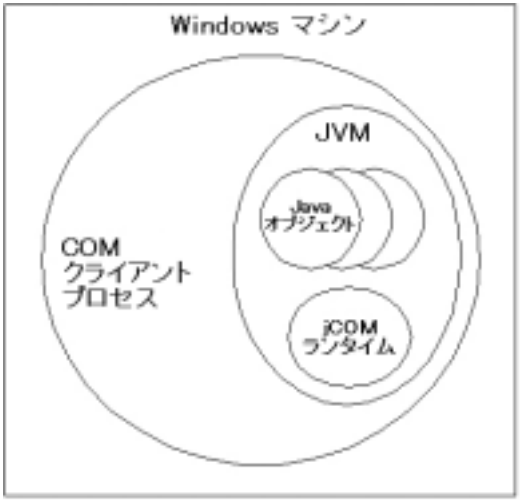
ネイティブ モードは、現在ローカル マシン上でのみ動作します。JVM 名以外にパラメータは必要ありません。



JVM 内の Java コードは、com.bea.jcom.Jvm.register("<jvm id>") を呼び出す必要があります。ここで <jvm id> は、regjvm で定義した JVM の ID です。また JVM は、設定された JCOM_NATIVE_MODE プロパティで開始されなければなりません。

ネイティブ モード（プロセスの内部）

プロセス内部のネイティブ モードを使用すると、Java オブジェクトを COM クライアント同じプロセスに実際にロードできます。もちろん、どちらのオブジェクトも同じマシン上に配置しなければなりません。



JVM は、com.bea.jcom.Jvm.register() を呼び出すか、またはクライアントの追加プロセスとして開始する必要があります。

regjvm GUI ツールのインタフェース

`regjvm` ツールを実行すると、ダイアログが表示されます。このダイアログは、以下の 2 つの部分に分かれています。

- 上部は、現在のマシン上のすべての JVM を選択および管理するための部分です。ここでは、JVM の追加、変更、または削除を行うことができます。異なる JVM に切り替える場合は、現在選択されている JVM に対する変更を保存する必要があります。また、JVM モードの選択もここで行います。この場合、必要な情報がダイアログの下部に表示されます。
- ダイアログの下部には、それぞれの JVM で必要な詳細が、JVM のモードに従って表示されます。JVM の詳細の他に、各 JVM モードの詳細オプションを表示するためのチェックボックスも存在します。

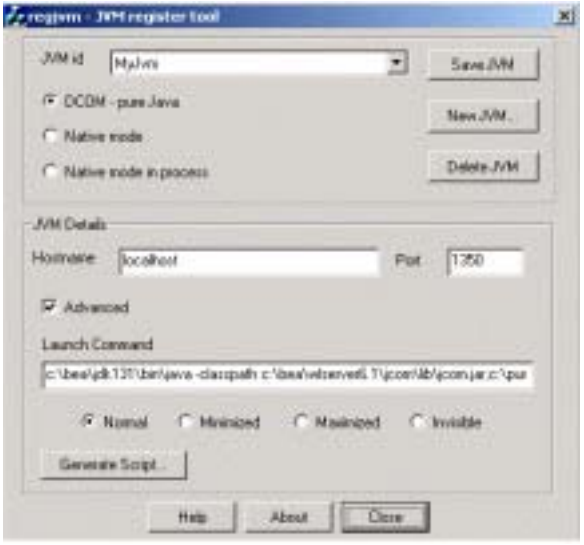
  これらのオプションについては、以下の節で説明します。

regjvm GUI ツールの DCOM モード オプション

標準オプション

- **[Hostname]**（必須）― JVM が存在する IP 名と IP アドレスです。
- **[Port]**（必須）― JVM との通信を開始するためのポートです。

詳細オプション



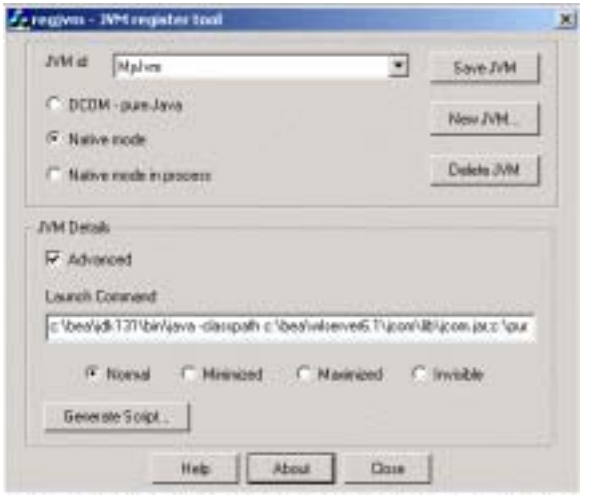
- **[Launch command]**（省略可能）— JVM が自動的に起動する場合に使用されるコマンドです。通常、このコマンドは次のようになります。c:\bea\jdk131\bin\java -classpath c:\bea\wlserver6.1\jcom\lib\jcom.jar; c:\pure MyMainClass.
- **起動オプション**（省略可能）— サーバ コンポーネントの初期ウィンドウ状態を指定できます。
- **[Generate Script...]**（省略可能）— JVM の設定を選択するレジストリ スクリプトを生成できます。

regjvm GUI ツールのネイティブ モード オプション

標準オプション

このモードの標準オプションは存在しません。

詳細オプション



- **[Launch command]**（省略可能）― DCOM モードを参照してください。

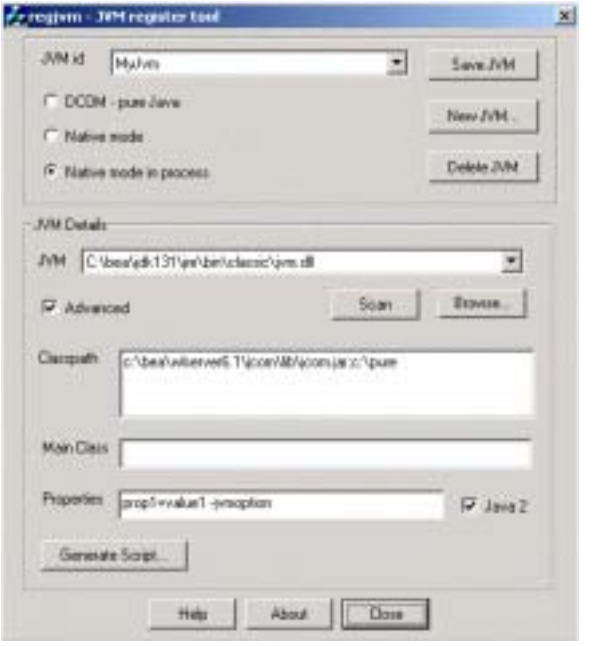
- **起動オプション**（省略可能）― DCOM モードを参照してください。
- **[Generate Script...]**（省略可能）― DCOM モードを参照してください。

regjvm GUI ツールのプロセス内ネイティブ モード オプション

標準オプション

- **[JVM]**（必須）― JVM を指定します。[Browse] ボタンをクリックすると、独自の JVM を選択できます。[Scan] ボタンをクリックすると、ローカル マシンから JVM を探して（数分かかる場合があります）、それらを選択用のリストボックスに挿入します。

詳細オプション



- **[Classpath]**（省略可能）― JVM の CLASSPATH です。空白のままにすると、実行時の CLASSPATH 環境変数が使用されます。それ以外の場合、内容が CLASSPATH 環境変数に追加されます。
- **[Main class]**（省略可能）― 呼び出す Main メソッドを含むクラスの名前です。
- **[Properties]**（省略可能）― 設定する必要があるプロパティです。prop1=value1 prop2=value2 という構文にする必要があります。
- **[Java 2]**（省略可能）― プロパティを設定した場合、Java 2（JDK 1.2.x、1.3.x）を使用するときはこれをオンにし、1.1.x を使用するときはオフにする必要があります。
- **[Generate Script...]**（省略可能）― DCOM モードを参照してください。

regjvmcmd コマンド ライン ツール

regjvmcmd は、前述の GUI ツール、regjvm のコマンド ライン バージョンです。このパラメータのまとめを参照するには、パラメータを指定せずに regjvmcmd を実行します。

最も単純な形式では、以下のものを指定します。

- jvm ID（*com.bea.jcom.Jvm.register("JvmId")* で使用する名前に対応）。
- JVM アクセスするためのバインディング。形式は *hostname[port]* で、hostname は JVM を実行するマシン名、port は JCOM_DCOM_PORT プロパティを選択することによって JVM の起動時に指定する TCP/IP ポートです（*java -DJCOM_DCOM_PORT=1234 MyMainClass* など）。

JVM を登録する必要がないか、またはその登録を変更する場合、まず *regjvmcmd/unregister JvmId* を使用してその登録を解除する必要があります。

java2com ツール

java2com ツールは、Java クラスを（Java の reflection メカニズムを使用して）解析し、以下のものを出力します。

- COM インタフェース定義言語（IDL）ファイル。
- pure Java DCOM マーシャリング コード（ラッパー）。これは、vtable（レイト バインド）アクセスを使用して COM から Java オブジェクトへのアクセスを容易にするために WebLogic jCOM ランタイムによって使用されます。

IDL ファイルは、Microsoft's MIDL ツールでコンパイルする必要があります。

IDL ファイルとラッパーを生成するには、次のコマンドを使用して java2com ツールを起動します。

```
java com.bea.java2com.Main
```

このツールは、どのようなプラットフォームでも実行できます。WebLogic jCOM ランタイム jCOM.jar が CLASSPATH 環境変数に含まれていることを確認してください。

java2com ツールは、次のダイアログ ボックスを表示します。

このダイアログ ボックスには、以下のフィールドが存在します（コンフィグレーションの変更は、ダイアログ ボックスの終了時に自動的に保存されます）。

1. [Java Classes & Interfaces]

   java2com で解析する「ルート」Java クラスとインタフェースが存在します。これらは、CLASSPATH でアクセス可能でなければなりません。WebLogic jCOM はこれらのクラスを解析し、COM IDL 定義と、COM から Java クラスにアクセスするための Java DCOM マーシャリング コードを生成します。次に、そのクラスのパラメータまたはフィールドで使用されるクラスまたはインタフェースに対して同じ解析を繰り返し実行し、同様にアクセスできるすべての Java クラスとインタフェースを解析します。

   クラス名はスペースで区切って入力します。[...] ボタンをクリックすると、クラスのリストを表示して、そのリストから追加または削除を行うためのダイアログが表示されます。

2. [Name of Generated IDL File]

   これは、生成される COM インタフェース定義言語（IDL）ファイルの名前です。myjvm と指定した場合は、myjvm.idl が生成されます。この名前は、Microsoft の MIDL コンパイラを使用して myjvm.idl をコンパイルするときに生成される型ライブラリの名前にも使用されます。

3. [Output Directory]

   java2com が生成したファイルを出力するディレクトリです。デフォルトはカレント ディレクトリ（「.」）です。

4. [Dump Analysis]

   java2com が発見したクラスをそのまま表示します。

5. [Save Settings] と [Load Settings]

   [Save Settings] ボタンをクリックすると、現在の java2com 設定が保存されます。

   java2com は、起動時にカレント ディレクトリに java2com.ser 設定ファイルが存在するかどうかをチェックします。存在する場合、そのファイルから設定を自動的にロードします。

6. [Names...]

   [Names...] ボタンをクリックすると、次のダイアログ ボックスが表示されます。

   クラス / インタフェース名ドロップダウン リストから「*」を選択すると、メンバー（フィールドまたはクラス）名の名前を入力するためのテキスト ボックスが表示されます。生成するクラスまたはインタフェースでそのメンバー名が見つかったときに使用される対応 COM 名を指定できます。この名前を空白のままにした場合、Java メンバーは COM インタフェースで生成される対応メンバーを持ちません。

   クラス / インタフェース名ドロップダウン リストから特定の COM クラス名またはインタフェースを選択すると、そのクラスまたはインタフェースのメ

ンバーのセットがその下に表示されます。使用する COM 名を指定し、[Add this Class Name map] をクリックすることによって、選択したクラス / インタフェースを指定した COM 名にマップします。[Add this Class Name map] をクリックすると、選択したメンバーを指定した COM 名にマップできます。

7. [Generate] ボタン

   ラッパーおよび IDL ファイルを生成します。

`java2com` が発見するパブリック Java インタフェースごとに、対応する COM インタフェース定義が作成されます。Java インタフェース名が com.bea.finance.Bankable の場合、生成される COM インタフェースの名前は、[Names ...] ダイアログで異なる名前を指定しない限り、ComBeaFinanceBankable となります。

`java2com` が発見するパブリック Java クラスごとに、対応する COM インタフェース定義が作成されます。Java クラス名が *`com.bea.finance.Account`* の場合、生成される COM インタフェースの名前は、[Names ...] ダイアログで異なる名前を指定しない限り、*`IComBeaFinanceAccount`* となります。また、Java クラスがパブリック デフォルト コンストラクタを持つ場合、java2com は、[Names ...] ダイアログで異なる名前を指定しない限り、COM クラス ComBeaFinanceAccount を生成します。

Java クラスが Java イベントを生成できる場合、生成される COM クラスは Java クラスによってサポートされるイベントに対応するソース インタフェース（COM イベント）を持ちます。

生成された IDL ファイルは、Microsoft の MIDL ツールを使用してコンパイルします。このツールは Visual C++ に付属しており、MS Web サイトからダウンロードできます。次のコマンド、

```
midl prodServ.idl
```

は、`prodServ.tlb` という型ライブラリを生成します。このライブラリは、以下の節で説明するとおり登録できます。

regtlb ツール

WebLogic jCOM の regtlb ツールは、COM のアーリー バインディング メカニズムを使用して Java オブジェクトにアクセスする COM Windows クライアントに型ライブラリを登録します。regtlb は、2 つのパラメータを取ります。最初のパラメータは、登録する型ライブラリ ファイルの名前です。2 番目は、型ライブラリに記述されている COM クラスが存在する JVM の ID です。

WebLogic jCOM java2com ツールで生成された IDL ファイルから型ライブラリが生成された場合、regtlb コマンドは、型ライブラリ内の各 COM クラスに対応する Java クラス名を自動的に調べます。型ライブラリの COM クラス記述の形式は次のとおりです。

```
Java class java.util.Observable (via jCOM))
```

java2com で生成された IDL ファイルから型ライブラリが生成されなかった場合、各 COM クラスに対してインスタンス化する Java クラスの名前を指定する必要があります。

このため、誰かが Atldll.Apple のインスタンスを作成しようとした場合、WebLogic jCOM は JVM MyJvm 内の com.bea.MyAppleClass をインスタンス化します。MyAppleClass クラスは、COM クラス Atldll.Apple によって実装される atldll.tlb から WebLogic jCOM の java2com ツールで生成された Java インタフェースを実装します。