BEA WebLogic Server™

WebLogic jCOM プログラマーズ ガイド

BEA WebLogic Server バージョン 7.0
マニュアルの日付 : 2002 年 6 月
改訂 : 2002 年 6 月 28 日

著作権

限定的権利条項

商標または登録商標

WebLogic jCOM プログラマーズ ガイド

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

目次

このマニュアルの内容

1. WebLogic jCOM の概要

2. COM クライアント アプリケーションから WebLogic Server への呼び出し

4. jCOM ツールの詳細

5. アップグレードに関する考慮事項

このマニュアルの内容

このマニュアルでは、WebLogic Server の機能である jCOM の使用方法について説明します。jCOM を使用することにより、2 つのアプリケーションを記述することができます。WebLogic Server 上にある Java オブジェクトのメソッドの作成および呼び出しを行う Common Object Model (COM) クライアント アプリケーションと、WebLogic Server の外部に配置された COM オブジェクト上のメソッドの作成と呼び出しを行う WebLogic Server アプリケーションです。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic jCOM の概要」では、WebLogic jCOM の機能とアーキテクチャについて概説し、アプリケーション開発の計画に役立つ情報を提供します。
- 第 2 章「COM クライアント アプリケーションから WebLogic Server への呼び出し」では、COM クライアントから WebLogic Server 上のメソッドを呼び出すために WebLogic jCOM を使用する方法について説明します。
- 第 3 章「WebLogic Server から COM アプリケーションへの呼び出し」では、WebLogic Server から COM オブジェクト上のメソッドを呼び出すために WebLogic jCOM を使用する方法について説明します。
- 第 4 章「jCOM ツールの詳細」では、jCOM アプリケーションで使用するツールの詳細について説明します。
- 第 5 章「アップグレードに関する考慮事項」では、WebLogic jCOM 6.1 から WebLogic jCOM 7.0 へアップグレードする方法について説明します。

対象読者

このマニュアルは、WebLogic Server と相互運用する COM アプリケーションを構築するアプリケーション開発者を対象としています。このマニュアルは、Web テクノロジ、Common Object Model、Distributed Common Object Model、オブジェクト指向プログラミング手法、および Java プログラミング言語に読者が精通していることを前提として書かれています。

e-docs Web サイト

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

このマニュアルの印刷方法

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

このマニュアルの PDF 版は、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。

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　WebLogic jCOM の概要

以下の節では、WebLogic jCOM の概要を説明します。

WebLogic jCOM とは

WebLogic jCOM は、WebLogic Server でデプロイされる Java/J2EE オブジェクトと、Microsoft Office 製品ファミリ、Visual Basic オブジェクトおよび C++ オブジェクト、その他のコンポーネント オブジェクト モデル / 分散コンポーネント オブジェクト モデル (COM/DCOM 準拠) 環境で使用できる Microsoft ActiveX コンポーネントとの間で双方向アクセスを可能にするソフトウェア ブリッジです。

通常 Microsoft のアプリケーションとの通信には、Web サービスを使用する方法が推奨されます。この種の通信を利用するため、従来の COM アプリケーションから .NET へ移行する計画を立てることをお薦めします。jCOM は、Java - COM 間の統合を必要とする暫定的なソリューションへの移行経路として提供され、小規模なプロジェクトやブリッジ ソリューションに適しています。

市販されている他の Java - COM ブリッジとは異なり、jCOM は WebLogic Server において Java サイドで動作するように特別に設計されています。jCOM を任意の Java 仮想マシン (JVM) と COM オブジェクトとの通信に使用することはできま

せん。なお、jCOM は WebLogic Server スレッドを直接使用するため、サービスを COM オブジェクトに対して公開するための非常に堅牢な手段となります。

注意： WebLogic jCOM 7.0 は、WebLogic Server の一部として動作します。以前のバージョンでは、スタンドアロンのソフトウェア ブリッジとして動作していました。WebLogic jCOM をスタンドアロンで実行しないでください。

WebLogic jCOM では、オブジェクト タイプの違いは意識されなくなります。つまり、COM クライアントには WebLogic Server オブジェクトが COM オブジェクトのように見え、WebLogic Server アプリケーションには COM コンポーネントが Java オブジェクトのように見えます。

WebLogic jCOM は以下のようにして双方向を実現します。

- Microsoft COM クライアントが、それ自体 COM コンポーネントであるかのように WebLogic Server 内のオブジェクトにアクセスできます。

および

- WebLogic Server 内のアプリケーションが、それ自体 Java オブジェクトであるかのように COM コンポーネントにアクセスできます。

用語に関する重要な注意

このプログラミング ガイドでは、アクセス方向によってアプリケーションを次の 2 つのタイプに分けています。

- アプリケーション内で COM クライアントが WebLogic Server オブジェクトにアクセスする場合は、「COM-to-WLS」アプリケーション。
- アプリケーション内で WebLogic Server が COM オブジェクトにアクセスする場合は、「WLS-to-COM」アプリケーション。

jCOM アーキテクチャ

WebLogic jCOM は、Distributed Computing Environment Remote Procedure Call (DCE RPC) 上の COM/DCOM 分散コンポーネント インフラストラクチャと Java Remote Method Protocol/Internet Inter-ORB Protocol (JRMP/IIOP) 上の Remote Method Invocation (RMI) 分散コンポーネント インフラストラクチャの両方を実装する実行時コンポーネントを提供します。これにより、反対側のオブジェクトも、各環境のネイティブ オブジェクトのように見えます。

また、WebLogic jCOM は、両タイプのインタフェース間で変換を行う自動ツールも提供します。これは、前述のプロトコルで双方が通信するために必要な COM/DCOM プロキシと RMI スタブを自動的に構築します。

WebLogic jCOM は、DCOM 技術と RMI 技術の間で必要な変換をすべて行い、RMI クライアントとして WebLogic Server に接続します。それから、WebLogic Server にデプロイされているエンタープライズ Java Bean (EJB) に、通常の EJB クライアントからの要求と同じように、要求を伝えます。

同様に、WebLogic Server にデプロイされているコンポーネントが DCOM オブジェクトによって提供されるサービスを要求すると、その要求は jCOM コンポーネントによって、WebLogic Server により発行される通常の RMI クライアント要求から DCOM 準拠要求に変換され、DCOM 環境内の該当するオブジェクトへ伝えられます。

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。

jCOM を使用する理由

WebLogic jCOM を使用する主な理由は以下のとおりです。

- 多様なハードウェアおよびソフトウェア プラットフォームにわたる分散型アプリケーション間の相互運用性を実現するため。

- Microsoft の開発ツールや開発スタッフの訓練に多額の投資をしてきたことから、Java クライアント ソフトウェアを記述せずにクライアント アプリケーションを WebLogic Server 上のビジネス ロジックへアクセスさせたいと考える人を援助するため。
- 完全に統合されたアプリケーションの構築と既存コンポーネントの再利用のために、COM/DCOM と Java 環境の両方で利用できるスキルを活用したいと考える e- ビジネス アプリケーション構築者のニーズに応えるため。各環境の詳細は、別の環境に慣れた開発者にまったく意識させないようにすることができます。

WebLogic jCOM は、異種の環境およびアプリケーションの透過的な相互運用というソフトウェア業界の動向に従っています。

WebLogic jCOM の機能

WebLogic jCOM サブシステムの主要な機能は以下のとおりです。

- 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、オブジェクト タイプのレイト 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。
- 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。
- 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。1-10 ページの「DCOM モードとネイティブ モード」を参照してください。
- WebLogic jCOM は、イベント処理をサポートしています。たとえば、標準 COM イベント メカニズムを使用して Visual Basic から Java イベントにアクセスすることや、Java オブジェクトが COM コンポーネント イベントをサブスクライブすることができます。

WebLogic jCOM のサンプル

WebLogic Server には、RPC スタイル Web サービスおよびメッセージスタイル Web サービスを作成する両方の例と、Web サービスを呼び出す Java および Microsoft VisualBasic クライアント アプリケーションの両方の例があります。

WebLogic Server には、次の各 jCOM サンプルが付属しています。

- ゼロ クライアント インストールを使用して、WebLogic Server 上にデプロイされている EJB に Excel Visual Basic アプリケーション (VBA) クライアントからアクセスする方法を示すサンプル。

  ゼロ クライアント インストールでは、Windows クライアント マシンに WebLogic jCOM ソフトウェアが必要ありません。ただし、これを実現するには、サーブレットから「オブジェクト参照モニカ」を取り出して COM クライアント コードに配置する必要があります。

- アーリー バインディングを使用して、WebLogic Server 上にデプロイされている EJB に Excel VBA クライアントからアクセスする方法を示すサンプル。

- レイト バインディングを使用して、WebLogic Server 上にデプロイされている EJB に Excel VBA クライアントからアクセスする方法を示すサンプル。

WL_HOME\samples\server\src\examples\jcom ディレクトリにサンプルがあります。WL_HOME は、WebLogic Platform の最上位のインストール ディレクトリです。

サンプルを作成して実行する方法の詳細については、ブラウザで Web ページ *WL_HOME*\samples\server\src\examples\jcom\package_summary.html を呼び出してください。

WebLogic jCOM アプリケーションのプランニング

jCOM アプリケーションの設計と構築の前に、いくつか重要な決定をする必要があります。決定すべきことは、次のとおりです。

- アプリケーションにゼロ クライアント アーキテクチャを採用するかどうか (COM-to-WLS のみ)
- アーリー バインディングとレイト バインディングのどちらを採用するか (COM-to-WLS のみ)
- ネイティブ モードと DCOM モードのどちらで jCOM アプリケーションを実行するか (COM-to-WLS と WLS-to-COM の両方)

この節では、これらを決定する上で役立つ情報を提供します。

ゼロ クライアントのデプロイメント

jCOM ゼロ クライアント デプロイメントは実装が簡単です。クライアント マシンには WebLogic jCOM 固有のソフトウェアは必要ありません。

オブジェクト参照モニカ (`objref`) モニカ文字列を使用して、WebLogic Server の位置を COM クライアントにコード化します。`objref` モニカを生成します。このモニカによって、WebLogic Server の IP アドレスとポートがエンコーディングされます。COM クライアント コードのモニカ文字列はプログラムで取得するか、コピーして貼り付けるか、あるいは WebLogic Server サーブレットから取得します。サーバ接続が確立されると、COM 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。

ゼロ クライアント デプロイメントのメリットとデメリット

次の表に、ゼロ クライアント実装のメリットとデメリットを示します。

メリット	デメリット
クライアント マシン レジストリに WebLogic jCOM 固有のソフトウェアをロードする必要がない。	WebLogic Server マシンの *`WL_HOME`*`\bin` ディレクトリから jCOM 固有のツールをいくつかコピーする必要がある。
レイト バインディングを使用するので (1-8 ページの「アーリー バインディングとレイト バインディング」を参照)、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る。	WebLogic Server の位置とポート番号を COM クライアントにコード化する必要があるため、サーバ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しなければならない。

メリット	デメリット
	アプリケーションでアーリー バインディングの利点が得られなくなる (1-8 ページの「アーリー バインディングとレイト バインディング」を参照)。

WebLogic jCOM デプロイメントで多数の COM クライアント マシンが必要な場合は、ゼロ クライアント モデルのプログラミング モデルが適しています。

ゼロ クライアントのサンプル

ゼロ クライアント実装の例として、インストールされている WebLogic Server の *WL_HOME*\samples\server\src\examples\jcom\zeroclient ディレクトリにあるサンプルを参照してください。

アーリー バインディングとレイト バインディング

バインディングでは、ルーチンまたはモジュールのシンボリック アドレスを物理的アドレスに置換します。アーリー バインディングとレイト バインディングは、ともに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します。

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。

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。

各バインディング モデルのメリットとデメリット

次の表に、アーリー バインディング モデルのメリットとデメリットを示します。

アーリー バインディングのメリット	アーリー バインディングのデメリット
■ レイト バインド実装よりも信頼性が高い。 ■ コンパイル時の型チェックによりデバッグが容易である。 ■ アプリケーションのエンド ユーザが型ライブラリを参照できる。 ■ レイト バインド実装に比べ、実行時のトランザクション パフォーマンスが向上する。	■ 型ライブラリとラッパーを生成する必要があるため、実装が複雑になる。 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 サイドで必要となる。クライアントとサーバ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らない。 ■ レイト バインド アクセスが備えている柔軟性に欠ける。これは、Java コンポーネントに変更を加えた場合にラッパーと型ライブラリを再生成する必要があるため。 ■ レイト バインド実装に比べ、実行時の初期化が低速になる。

次の表に、レイト バインディング モデルのメリットとデメリットを示します。

レイト バインディングのメリット	レイト バインディングのデメリット
■ 実装が容易である。 ■ 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る。 ■ アーリー バインド実装に比べ、実行時の初期化が高速になる。	■ 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る。 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るまで分からない。 ■ アーリー バインド実装に比べ、実行時のトランザクション パフォーマンスが劣る。

アーリー バインディングとレイト バインディングのサンプル

アーリー バインディング実装の例として、インストールされている WebLogic Server の *WL_HOME*\samples\server\src\examples\jcom\earlybound ディレクトリにあるアーリー バインドのサンプルを参照してください。

レイト バインディング実装の例として、インストールされている WebLogic Server の *WL_HOME*\samples\server\src\examples\jcom\latebound ディレクトリにあるレイト バインドのサンプルを参照してください。

DCOM モードとネイティブ 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。WebLogic jCOM アプリケーションが DCOM モードで動作している場合、COM クライアントは WebLogic Server と DCOM プロトコルで通信します。

ネイティブ モードの場合、COM クライアントは WebLogic Server にネイティブ呼び出しを行い (COM-to-WLS)、WebLogic Server は COM アプリケーションにネイティブ呼び出しを行います。

ネイティブ モードではローカルのオペレーティング システムと CPU に対して個別にコンパイルおよび最適化された、ネイティブ コードの動的にロードされるライブラリ (DLL) が使用されるので、COM-to-WLS アプリケーションでも WLS-to-COM アプリケーションでも、ネイティブ モードを使用するとパフォーマンスが向上します。

また、ネイティブ モードで動作している COM-to-WLS アプリケーションは、COM クライアントと WebLogic Server との通信に WebLogic の T3/IIOP プロトコルを使用します。これにより、以下の利点が得られます。

- ネットワークの呼び出しが減るので、DCOM 呼び出しを使用した場合に比べてパフォーマンスが向上する。

  たとえば、COM アプリケーションで、WebLogic Server への呼び出しによって値が返される 100 のデータ要素を含むベクトルを作成するとします。これを DCOM モードで行うと、サーバに対する往復のネットワーク呼び出しが

100 回必要になります。ネイティブ モードでは、1 回の往復の呼び出しで済みます。

- WebLogic Server のフェイルオーバ機能とロード バランシング機能を利用できる。

ただし、どちらのタイプのアプリケーションでも、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 アクセスの実装では、すべての COM クライアント マシンに WebLogic Server をインストールする必要があります。ただし、COM アプリケーションを実行するマシンごとに別個の WebLogic Server ライセンスが必要というわけではありません。

また、WLS-to-COM アプリケーションの場合、ネイティブ モードで実行するには、WebLogic Server が Windows マシンで動作している必要があります。

ネイティブ モードのメリットとデメリット

次の表に、ネイティブ モード実装のメリットとデメリットを示します。

メリット	デメリット
COM-to-WLS アプリケーションでは、呼び出しがネットワーク上で行われないので、DCOM モードの場合に比べてパフォーマンスが向上する。	COM-to-WLS アプリケーションでも WLS-to-COM アプリケーションでも、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 モードの実装では WebLogic Server をすべての COM マシンにインストールする必要がある。
COM-to-WLS アプリケーションでは、WebLogic Server のロード バランシング機能とフェイルオーバ機能を利用できる。	WLS-to-COM アプリケーションの場合、ネイティブ モードで実行するには、WebLogic Server が Windows マシンで動作している必要がある。

メリット	デメリット
WLS-to-COM アプリケーションの場合、COM オブジェクトを WebLogic Server と同じマシンにインストールすると、WebLogic Server でネットワーク呼び出しが不要になるので、パフォーマンスが向上する。	

2 COM クライアント アプリケーションから WebLogic Server への呼び出し

この章では、COM クライアントから WebLogic Server 上のメソッドを呼び出すために WebLogic jCOM を使用する方法について説明します。

ネイティブ モードの特別な要件

COM-to-WLS アプリケーションをネイティブ モードで実行するためには、WebLogic Server を COM 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。ただし、COM クライアントを実行するマシンごとに別個の WebLogic Server ライセンスが必要というわけではありません。

ネイティブ モードの詳細については、2-15 ページの「ネイティブ モードでの COM-to-WLS アプリケーションの実行」を参照してください。

COM クライアントから WebLogic Server を呼び出す主な手順

この節では、COM クライアントから WebLogic Server への呼び出しを行うための主な手順を簡単に説明します。詳細については、以降の節で説明します。

WebLogic Server 側

1. アーリー バインディングを使用している場合、`java2com` ツールを実行して Java ラッパー クラスと Interface Definition Library (IDL) ファイルを生成し、これらのファイルをコンパイルします。2-3 ページの「Java ラッパーおよび IDL ファイルの生成 - アーリー バインディングのみ」を参照してください。
2. サーバを起動します。2-5 ページの「サーバの起動」を参照してください。
3. サーバ リスン ポート上で COM 呼び出しを有効化します。2-6 ページの「サーバ リスン ポート上での COM 呼び出しの有効化」を参照してください。
4. COM クライアントにサーバ クラスへのアクセスを許可します。2-6 ページの「アクセス制御のコンフィグレーション」を参照してください。
5. 他の関連するコンソール プロパティをコンフィグレーションします。2-8 ページの「その他のコンソール プロパティのコンフィグレーション」を参照してください。
6. 静的コンソール プロパティの値を変更した場合、新しい値を有効にするためにサーバを再起動します。

COM クライアント側

1. jCOM ツール ファイルおよび WebLogic Server クラス ファイル (ネイティブ モードの場合のみ) をインストールします。2-8 ページの「必要なファイルのインストール」を参照してください。
2. ゼロ クライアント インストールの場合
   - オブジェクト参照モニカ (ORM) を WebLogic Server ORM サーブレットからプログラムとして取得するか、アプリケーションに貼りつけることで取得します。2-9 ページの「WebLogic Server サーブレットからのオブ

ジェクト参照モニカの取得 - ゼロ クライアントのみ」を参照してください。

3. アーリー バインディングを使用している場合
   - WebLogic Server マシン上で生成された IDL ファイルを取得して、型ライブラリにコンパイルします。
   - 型ライブラリと、サービスする WebLogic Server を登録します。

   上記の手順については、2-10 ページの「Java ラッパーおよび IDL ファイルの生成 - アーリー バインディングのみ」を参照してください。
4. レジストリに WebLogic Server JVM を登録します。ネイティブ モードで WebLogic Server と通信する場合、ここで設定します。2-12 ページの「クライアント マシン レジストリへの WebLogic Server JVM の登録」を参照してください。
5. COM クライアント アプリケーションをコード化します。2-14 ページの「COM クライアント アプリケーションのコード化」を参照してください。
6. COM クライアントを起動します。2-15 ページの「COM クライアントの起動」を参照してください。

WebLogic Server の準備

以下の節では、COM クライアントが WebLogic Server オブジェクト上のメソッドを呼び出せるように、WebLogic Server を準備する方法について説明します。

Java ラッパーおよび IDL ファイルの生成 - アーリー バインディングのみ

1. JDK ライブラリへのパスと `weblogic.jar` へのパスを `CLASSPATH` に追加します。たとえば、次のように指定します。

   ```
   set CLASSPATH=%JAVA_HOME%\lib\tools.jar;
   %WL_HOME%\server\lib\weblogic.jar;%CLASSPATH%
   ```

JAVA_HOME は、JDK がインストールされているルート フォルダ (通常 c:\bea\jdk131) で、WL_HOME は、WebLogic Platform ソフトウェアがインストールされているルート ディレクトリ (通常 c:\bea\weblogic700) です。

2. java2com ツールで java ラッパー ファイルと IDL ファイルを生成します。

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

java2com GUI が表示されます。

3. 次のように入力します。

[Java Classes & Interfaces] : jCOMHelper examples.ejb20.basic.containerManaged.AccountHome [ここに変換するラッパー クラスをリストします。]
[Name of generated IDL File] : IDL ファイルの名前
[Output Directory] : ドライブ文字とルート ディレクトリ *\TLB*

TLB は OLE 型ライブラリです。

java2com ツールは、指定されたクラスとメソッド パラメータで使用される他のすべてのクラスを参照します。この作業は再帰的に行われます。ここでは、複数のクラスまたはインタフェースをスペースで区切って指定できます。

抽象クラスではなく、パラメータ コンストラクタを持たない Java のパブリック クラスはすべて、COM クラスとしてアクセス可能になります。他のパブリック クラス、およびすべてのパブリック インタフェースは、COM インタフェースとしてアクセス可能になります。

この時点で [Generate] ボタンをクリックしてラッパーと IDL を作成する場合、生成されたラッパーと IDL をコンパイルしようとすると、エラーが発生します。これは、java2com ツールのデフォルトで特定のクラスが省略されているためです。コンパイル中に生成されたエラーを参照することにより、どのクラスで問題が発生しているか判別できます。

この問題を修正するには、java2com ツールで [Names] ボタンをクリックして、必要なクラス ファイルへのあらゆる参照を削除します。この例では、次の参照を削除する必要があります。

```
*.toString->''''
class java.lang.Class->''''
```

4. これらの参照が削除されると、ラッパーと IDL を生成できます。java2com GUI で [Generate] をクリックします。

java2com ツールでは、Java オブジェクトへのアクセスに使用される DCOM マーシャリング コードを含む Java クラスが生成されます。これらの生成されたクラスは、WebLogic jCOM ランタイムによって背後で使用されます。必要なのはこれらのクラスをコンパイルすることだけです。また、これらのクラスが CLASSPATH に含まれていることを確認してください。

サーバの起動

WebLogic Server を起動します。「WebLogic Server の起動と停止」を参照してください。

サーバ リスン ポート上での COM 呼び出しの有効化

WebLogic Server をインストールすると自動的に jCOM がインストールされますが、jCOM を有効にする必要があります。これによって、リスン ポート上で COM の呼び出しをリスンするようにサーバに知らせます。

1. WebLogic Server Administration Console を起動します。
2. 左ペインで、サーバの名前をクリックします。
3. 右ペインの [接続] タブをクリックした後で [jCOM] タブをクリックして、[jCOM properties] 画面を表示します。
4. [COM を有効化] ボックスをチェックします。
5. サーバを再起動して設定を有効にします。コンソールで jCOM プロパティを設定する予定がある場合、サーバを起動する前に設定してください (2-8 ページの「その他のコンソール プロパティのコンフィグレーション」を参照)。

アクセス制御のコンフィグレーション

COM クライアント アプリケーションがアクセスする必要のあるクラスに、COM クライアント ユーザ アクセスを許可します。特定のアプリケーションによって、どのクラスを公開するかが決定されます。

ここでは、このリリースに付属するゼロ クライアントの例を示します (*`WL_HOME`*`/samples/server/src/examples/jcom/zeroclient` を参照してください)。

ゼロ クライアントの場合、COM クライアントは次の 3 つのクラスにアクセスする必要があります。

- `java.util.Collection`
- `java.util.Iterator`
- `ejb20.basic.beanManaged`

java.util.Collection および java.util.Iterator へのアクセスの許可

1. WebLogic Server Administration Console の左ペインで [サービス] ノードをクリックしてから、その下の [JCOM] ノードをクリックします。
2. 右ペインで、次のように入力します。

   ```
   java.util.*
   ```

3. [ポリシーを定義] をクリックします。
4. [ポリシー条件] ボックスで、[呼び出し側をメンバとするグループは] をダブルクリックします。
5. [グループ名の入力] フィールドに、アクセスを許可するユーザ グループの名前を入力します。

注意： ゼロ クライアントの例では、「everyone」にアクセスが許可されていますが、アクセスを許可する際には、できるだけ制約を設けることをお勧めします。

6. [追加] をクリックします。
7. [OK] をクリックします。
8. ウィンドウの右下にある [適用] をクリックします。

ejb20.basic.beanManaged へのアクセスの許可

ゼロ クライアントの例では、`ejb20.basic.beanManaged` クラスへのアクセスも必要です。

これを実現するには、手順 3 の「java.util.*」を「ejb20.basic.beanManaged.*」に置き換えて、「java.util.Collection および java.util.Iterator へのアクセスの許可」の手順を繰り返します。

最後にアスタリスクが付いているため、実際には `ejb20.basic.beanManaged` パッケージ全体にアクセスを許可していることに注意してください。

クラスへのアクセス許可および取消しに関する詳細については、Console オンライン ヘルプ の 7.x セキュリティに関する節を Web ブラウザで参照してください。

その他のコンソール プロパティのコンフィグレーション

その他の必要な jCOM コンソール プロパティをコンフィグレーションします。詳細については、jCOM プロパティに関する Console オンライン ヘルプを参照してください。

これらのプロパティのいずれかを有効にするためにサーバを起動する必要がある場合は、ここで再起動します。

COM クライアントの準備

以下の節では、WebLogic Server オブジェクト上のメソッドを呼び出せるように COM クライアントを準備する方法について説明します。

必要なファイルのインストール

WebLogic Server オブジェクト上のメソッドを呼び出すために、クライアント マシンにインストールしておく必要のあるファイルが多数あります。以下に示すように、このうち一部のファイルは、ネイティブ モードでメソッドを呼び出す場合にのみ必要です。

jCOM ツール ファイル

jCOM ツールを実行するには、5 つのファイルと 3 つのフォルダ (すべてのサブフォルダとファイルを含む) が必要です。WebLogic Server をインストールしたマシン上の *WL_HOME*\server\bin ディレクトリに、ファイルがあります。ファイルは次のとおりです。

- JintMk.dll
- ntvinv.dll
- regjvm.exe
- regjvmcmd.exe
- regtlb.exe
- regjvm (すべてのサブフォルダとファイルを含む)
- regjvmcmd (すべてのサブフォルダとファイルを含む)
- regtlb (すべてのサブフォルダとファイルを含む)

jCOM ツールの詳細については、第 4 章「jCOM ツールの詳細」を参照してください。

WebLogic Server クラス ファイル - ネイティブ モードのみ

ネイティブ モードでアプリケーションを実行するには、COM クライアント マシンが特定の WebLogic Server クラス ファイルにアクセスできる必要があります。これらのファイルを取得するには、各 COM クライアント マシンに WebLogic Server をインストールします。ただし、COM クライアントを実行するマシンごとに別個の WebLogic Server ライセンスが必要というわけではありません。

WebLogic Server サーブレットからのオブジェクト参照モニカの取得 - ゼロ クライアントのみ

オブジェクト参照モニカ (ORM) は、WebLogic Server から取得できます。COM クライアント アプリケーションからモニカを使用できるので、regjvmcmd を実行する必要がなくなります。新規サーバを作成しても、サーバのホストとポートが同じである限りモニカは有効なままです。

COM クライアント コードに対して ORM を取得する方法が 2 つあります。

- WebLogic Server 上で実行しているサーブレットを通じて、ORM を取得します。WebLogic Server 上で Web ブラウザを開き、http://[*wlshost*]:[*wlsport*]/bea_wls_internal/com へ行きます。

ここで *wlshost* は WebLogic Server マシンを、*wlsport* はサーバのポート番号を示します。

- WebLogic Server マシンの完全な名前または TCP/IP アドレスと、ポート番号をパラメータとして指定して、`com.bea.jcom.GetJvmMoniker` Java クラスを実行します。

  ```
  java com.bea.jcom.GetJvmMoniker [wlshost] [wlsport]
  ```

  objref モニカを示し、その使い方を説明する長いメッセージが表示されます。また、表示されるテキストは自動的にクリップボードにコピーされるので、ソースに直接貼り付けできます。指定したマシンとポートで WebLogic Server にアクセスするために、返された objref モニカを使用できます。

Java ラッパーおよび IDL ファイルの生成 - アーリー バインディングのみ

クライアント側に割り当てられたラッパー ファイルおよびインタフェース定義言語 (IDL) ファイルの生成を実行します。

1. IDL をクライアント マシンにコピーします。

   WebLogic Server 上で `java2com` ツールの実行 (2-3 ページの「WebLogic Server の準備」を参照) に成功した場合、IDL ファイルがサーバ マシン上に作成されています。この IDL ファイルをクライアント マシンにコピーして、この COM アプリケーションの `\TLB` サブディレクトリに配置します。

注意： `java2com` ツールがサンプルの `\TLB` サブディレクトリに出力しているはずなので、同じマシン上でクライアントとサーバを実行している場合、この手順は必要ありません。

2. IDL ファイルをコンパイルして、型ライブラリを作成します。

   ```
   midl containerManagedTLB.idl
   ```

   このコマンドで Microsoft IDL コンパイラ `MIDL.EXE` を呼び出して、コンパイルを実行します。コンパイルの結果、`containerManagedTLB.tlb` という型ライブラリが作成されます。

3. 型ライブラリを登録し、サービスする JVM を設定します。

```
regtlb /unregisterall
regtlb containerManagedTLB.tlb registered_jvm
```

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 行目の行は、新しくコンパイルされた型ライブラリを登録します。

regtlb に渡される 2 番目のパラメータ registered_jvm は重要です。これは、型ライブラリにリンクされる JVM の名前を指定します。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。

WebLogic Server JVM は、regjvm ツールを通じてクライアント マシンのレジストリに登録されます。詳細については、2-12 ページの「クライアント マシン レジストリへの WebLogic Server JVM の登録」を参照してください。

ラッパー ファイルに関する注意

- 一般にラッパー ファイルはサーバ上に配置し、コンパイルしておく必要があります。IDL ファイルはクライアント上に配置し、コンパイルしておく必要があります。サーバとクライアントを別々のマシン上で実行しているときに、クライアント側でラッパーと IDL を作成した場合、コンパイルしたばかりのラッパー ファイルをサーバに配布する必要があります。サーバ側でラッパーと IDL を作成した場合、IDL ファイルがコンパイルされて型ライブラリが作成されるクライアントに IDL ファイルを移動させる必要があります。

- ラッパー ファイルと IDL ファイルは、*java2com ツール*を 1 回実行して作成する必要があります。サーバ上とクライアント上の両方で java2com ツールを別々に実行しようとすると、作成するラッパー ファイルと IDL ファイルが通信できなくなります。IDL とラッパーは識別のためにユニークなスタンプを持ちます。ラッパーは java2com ツールの共通呼び出しから作成される IDL ファイルとのみ通信できます。IDL ファイルも通信できるのは、同じ条件で作成されたラッパーだけです。その結果、java2com ツールを 1 回実行する必要があり、作成されるファイルは後で分散されます。Java ソース コードに間違いがある場合や変更を加えるために、java2com ツールを再度実行する必要がある場合は、すべてのラッパー ファイル、IDL ファイル、および TLB ファイルを削除し、手順をすべてやり直す必要があります。

- java2com ツールを使用して非推奨のメソッドを含む (または参照する) クラスのラッパーを作成する場合、コンパイル時に非推奨の警告が表示されま

す。これらの警告は無視できます。WebLogic jCOM は、メソッドを COM からアクセス可能にします。

- 生成されたラッパー クラスは、CLASSPATH に含める必要があります。単に EJBjar に含めることはできません。

クライアント マシン レジストリへの WebLogic Server JVM の登録

サーバ名を Windows レジストリに追加し、その名前を TCP/IP アドレスおよび WebLogic が受信する COM 要求をリスンするクライアント サーバ間の通信ポートに関連付けることで、ローカル Java Vertual Machine に登録します。デフォルトで、これは localhost:7001 です。

1. `regjvm` GUI ツールを呼び出すと、次の画面が表示されます。

2. WebLogic Server がローカルホスト以外で実行しており、7001 以外のポートをリスンしている場合、ホスト名 (または IP アドレス) およびポート番号を入力します。

   `regjvm` のコマンドライン バージョンを使用することもできます。

   ```
   regjvmcmd servername localhost[7001]
   ```

JVM の登録解除

regjvm (または regjvmcmd) 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て新しいエントリを作成しなければなりません。

regjvm ツール ウィンドウで JVM を登録解除するには、登録を解除する JVM を選択して、[Delete] を選択します。

別の方法として、コマンドライン ツール regjvmcmd から JVM の登録を解除します。

```
regjvmcmd /unregister servername
```

ネイティブ モードを選択している場合

COM クライアントをネイティブ モードで実行している場合、regjvm ウィンドウで [Native mode] または [Native mode in process] ラジオボタンをクリックするか、または /native パラメータを使用して regjvmcmd を呼び出します。この手順に関する詳細については、2-15 ページの「ネイティブ モードでの COM-to-WLS アプリケーションの実行」を参照してください。

COM クライアント アプリケーションのコード化

これで、WebLogic Server オブジェクト上のメソッドを呼び出すことができます。これを自然にコード化する方法は、レイト バインディングを選択するか、アーリー バインディングを選択するかによって異なります。

この節で記載するコードは、この製品に付属するアーリー バインドのサンプルとレイト バインドのサンプルからの抜粋です。サンプル コードは *SAMPLES_HOME*\server\src\examples\jcom から入手できます。

レイト バインド アプリケーション

このレイト バインドのサンプルである Visual Basic アプリケーションからの抜粋では、Account EJB のホーム インタフェース mobjHome の COM バージョンの宣言に注意してください。この COM オブジェクトは、サーバ側にある AccountHome インタフェースのインスタンスにリンクされます。

```
Dim mobjHome As Object

Private Sub Form_Load()

'Handle errors

On Error GoTo ErrOut '

Bind the EJB AccountHome object via JNDI

Set mobjHome =
CreateObject("examplesServer:jndi:ejb20-containerManaged-AccountHome")
```

レイト バインド クライアントの確認済みの問題と回避策

オーバーロードされていながらパラメータの数が同じメソッドを処理する場合、WebLogic jCOM には問題があります。オーバーロードされたメソッド内のパラメータ数が異なる場合、そのような問題はありません。

パラメータ数が同じ場合には、呼び出しに失敗します。

残念ながら、メソッド InitialContext.lookup はオーバーロードされています。

```
public Object lookup(String)
public Object lookup(javax.naming.Name)
```

ルックアップを実行するには、オブジェクトを作成するために特別な JNDI モニカを使用する必要があります。

```
Set o = CreateObject("servername:jndi:objectname")
```

アーリー バインド アプリケーション

アーリー バインド コードの最も明らかな特徴は、オブジェクトとして宣言される変数の数が少ないことです。以前生成した型ライブラリを使用することにより、ここでオブジェクトを宣言できます。

「Java ラッパーおよび IDL ファイルの生成 - アーリー バインディングのみ」で生成した型ライブラリを使用してオブジェクトを宣言します。この Visual Basic コードからの抜粋では、IDL ファイルは `containerManagedTLB` と呼ばれ、EJB は `ExamplesEjb20BasicContainerManagedAccountHome` と呼ばれます。

```
Dim objNarrow As New containerManagedTLB.JCOMHelper
```

ここで、オブジェクトのメソッドを呼び出すことができます。

```
Set mobjHome = objNarrow.narrow(objTemp,
"examples.ejb20.basic.containerManaged.AccountHome")
```

COM クライアントの起動

COM クライアント アプリケーションを起動します。

ネイティブ モードでの COM-to-WLS アプリケーションの実行

COM-to-WLS アプリケーションでは、ネイティブ モードに「イン プロセス」と「アウト オブ プロセス」の区別があります。

- アウト オブ プロセス。JVM は独自のプロセス内で作成され、COM プロセスと WebLogic Server JVM プロセスの間でプロセス間通信が発生します。
- イン プロセス。WebLogic Server JVM 全体が COM プロセスに取り込まれます。つまり、COM クライアントのアドレス空間にロードされます。WebLogic Server クライアントサイド クラスはこの JVM 内に配置されています。

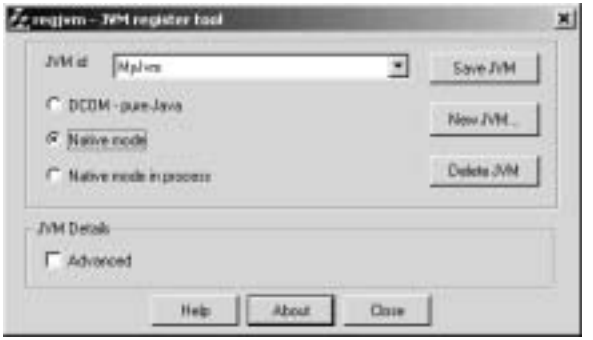
アプリケーションで使用するプロセスを指定するには、`regjvm` GUI ツール インタフェースで [Native mode in process] または [Native mode] ラジオ ボタンを選択します。

アウト オブ プロセスで JVM を実行するネイティブ モード

JVM をアウト オブ プロセスで実行する (ただし、ネイティブ コードを使用して COM クライアントに中の Java オブジェクトへのアクセスを許可する) 場合、次の手順に従います。

1. `regjvm` GUI ツールを呼び出して、JVM をネイティブとして登録します。`regjvm` では、さまざまなレジストリ エントリが設定され、WebLogic jCOM の COM と WLS 間のメカニズムが容易になります。

注意 : JVM を登録する場合には、[JVM id] フィールドにサーバの名前を指定する必要があります。たとえば、`exampleServer` で JCOM ネイティブ モードが有効になっているときに、`regjvm` を使用して登録する場合には、[JVM id] ボックスに `exampleServer` と入力します。



2. JVM が実行されていない場合は、[Advanced] ラジオ ボタンをクリックして、[Launch Command] フィールドに JVM のパスを入力します。

`regjvm` ツールの詳細については、第 4 章「jCOM ツールの詳細」を参照してください。

3. アプリケーション コードの `main` セクションに次のコードを挿入して、WebLogic jCOM ランタイムに JVM が呼び出しを受け取る準備ができていることを知らせます。

```
com.bea.jcom.Jvm.register("MyJvm");
public class MyJvm {

public static void main(String[] args) throws Exception {
// "firstjvm" の名前で JVM を登録
```

```
com.bea.jcom.Jvm.register("firstjvm");


Thread.sleep(6000000); // 1 時間の休止
}
```

4. これで、Visual Basic からレイト バインディングを使用して、JVM にロードできる Java クラスのインスタンスをインスタンス化できます。

   ```
   Set acctEJB =
   CreateObject("firstjvm.jndi.ejb20.beanManaged.AccountHome")
   ```

5. JVM を指定したら、標準 WebLogic jCOM `regtlb` コマンドを使用して Java オブジェクトにアーリー バインド アクセスします (`regtlb` はパラメータとして型ライブラリの名前と JVM 名を取り、その型ライブラリに定義されているすべての COM オブジェクトをその JVM に存在するものとして登録します)。

   また、独自のインスタンシエータ (`com.bea.jcom.Jvm.register(...)` への追加パラメータ) を JVM に関連付けることによって、COM クライアントの代わりに Java オブジェクトのインスタンス化を制御できます。これは、オブジェクト ファクトリの一種です。

インプロセスで JVM を実行するネイティブ モード

このテクニックを使用すると、実際に JVM を COM クライアントのアドレス空間にロードできます。

ここで再び `regjvm` コマンドを使用します。ただし、今度は追加パラメータを指定します。

注意 : JVM を登録する場合には、[JVM id] フィールドにサーバの名前を指定する必要があります。たとえば、`exampleServer` で JCOM ネイティブ モードが有効になっているときに、`regjvm` を使用して登録する場合には、[JVM id] ボックスに `exampleServer` と入力します。

最も単純な例は、Visual Basic を使用して Java オブジェクトにレイト バインド アクセスすることです。まず、JVM を登録します。Sun の JDK 1.3.1 (`c:\bea\jdk131` にインストールされている) を使用し、WebLogic Server が

`c:\bea\weblogic700\server\lib\weblogic.jar` にインストールされており、Java クラスが `c:\pure` に格納されている場合は、以下の `regjvm` ツール画面のように設定します。

JVM 名、CLASSPATH、および JVM bin ディレクトリ パスを指定します。

これで、Visual Basic から GetObject メソッドを呼び出すことができるようになります。

```
MessageBox GetObject("MyJVM.jndi.ejb20.beanManaged.AccountHome")
```

`regjvm` ツールの詳細については、第 4 章「jCOM ツールの詳細」を参照してください。

3 WebLogic Server から COM アプリケーションへの呼び出し

以下の節では、WLS-to-COM アプリケーション、すなわち、WebLogic jCOM を使用して WebLogic Server から COM オブジェクト上にメソッドを呼び出すアプリケーションを準備およびデプロイする方法を説明します。

- ネイティブ モードの特別な要件
- WebLogic Server から COM アプリケーションを呼び出す主な手順
- COM アプリケーションの準備
- WebLogic Server の準備

ネイティブ モードの特別な要件

ネイティブ モードを使用する WLS-to-COM アプリケーションには、特別な要件が 2 つあります。

- COM アプリケーションをネイティブ モードで実行するには、WebLogic Server が COM アプリケーション マシンにインストールされている必要があります。ただし、COM アプリケーション マシンごとに別個の WebLogic Server ライセンスが必要というわけではありません。
- ネイティブ モードで実行するには、WebLogic Server が Windows マシンで動作している必要があります。

WebLogic Server から COM アプリケーションを呼び出す主な手順

この節では、WebLogic Server から COM アプリケーションへ呼び出す主な手順を簡単に説明します。詳細については、以降の節で説明します。

COM 側

1. COM アプリケーションをコード化します。3-3 ページの「COM アプリケーションのコード化」を参照してください。
2. `com2java` ツールで COM オブジェクトから Java クラスを生成します。3-3 ページの「com2java GUI ツールでの Java クラスの生成」を参照してください。
3. WebLogic Server 用にクラスをパッケージ化します。3-4 ページの「WebLogic Server 用の Java クラスのパッケージ化」を参照してください。
4. COM アプリケーションを起動します。3-4 ページの「COM アプリケーションの起動」を参照してください。

WebLogic Server 側

1. サーバを起動します。3-4 ページの「サーバの起動」を参照してください。
2. WebLogic Server Administration Console で、必要に応じて COM を有効にしコンフィグレーションします。3-5 ページの「サーバ リスン ポート上での COM 呼び出しの有効化」を参照してください。
3. 他の Java オブジェクトと同じように COM オブジェクトを使用します。

COM アプリケーションの準備

以下の節では、WebLogic Server がオブジェクトでメソッドを呼び出せるよう COM クライアントを準備する方法について説明します。

COM アプリケーションのコード化

- 必要に応じて COM アプリケーションをコード化します。

com2java GUI ツールでの Java クラスの生成

COM 型ライブラリに対して com2java GUI ツールを実行すると、COM 型ライブラリ内のクラスとインタフェースに対応する Java クラス ファイルの集合が生成されます。

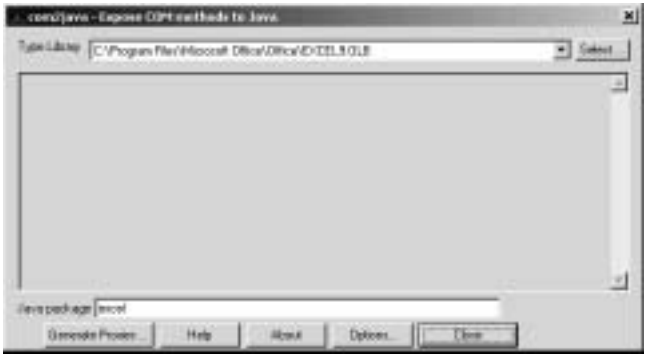
以降で、GUI ツールでの Java クラスの生成を示します。WebLogic jCOM ツール全般の詳細については、第 4 章「jCOM ツールの詳細」を参照してください。

1. com2java GUI ツールを実行するには、次の手順に従います。

   a. *WEBLOGIC_HOME*/server/bin ディレクトリへ移動します (あるいは、このディレクトリを CLASSPATH に追加します)。

   b. COM マシンでコマンド シェルを開き、com2java.exe ファイルを起動します。

   ```
   > com2java
   ```



2. 一番上のフィールドで適切な型ライブラリを選択し、[Java package] テキスト ボックスに、生成されたファイルを含むパッケージの名前を入力します。com2java ツールでは、特定の型ライブラリに対して指定した直前のパッケージ名を記憶しています。

3. [Generate Proxies] をクリックして、Java クラス ファイルを生成します。

WebLogic Server 用の Java クラスのパッケージ化

COM オブジェクトを EJB から呼び出す場合は、WebLogic Server が検出できるように、`com2java` で生成するクラス ファイルを EJB `.jar` にパッケージ化する必要があります。生成したファイルは、特定のパッケージに格納できます。たとえば、Excel 型ライブラリのすべてのファイルを *excel* という Java パッケージに格納できます。

EJB `.jar` ファイルのパッケージ化の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server コンテナ用の EJB のパッケージ化」の章を参照してください。

COM アプリケーションの起動

Java クラス ファイルを生成してそれを適切にパッケージ化したら、あとは COM アプリケーションを起動すれば、WebLogic Server に公開する COM オブジェクトをインスタンス化および実行できます。

WebLogic Server の準備

以下の節では、COM アプリケーションのオブジェクトにメソッドを呼び出せるよう WebLogic Server を準備する方法を説明します。

サーバの起動

WebLogic Server を起動します。「WebLogic Server の起動と停止」を参照してください。

サーバ リスン ポート上での COM 呼び出しの有効化

WebLogic Server をインストールすると自動的に jCOM がインストールされますが、jCOM を有効にする必要があります。これによって、リスン ポート上で COM の呼び出しをリスンするようにサーバに知らせます。

1. Administration Console を起動します。
2. 左ペインで、サーバの名前をクリックします。
3. 右ペインの [接続] タブをクリックした後で [jCOM] タブをクリックして、[jCOM properties] 画面を表示します。
4. [COM を有効化] ボックスをチェックします。
5. サーバを再起動して設定を有効にします。コンソールを介して [ネイティブ モードを有効化] またはその他の jCOM プロパティを設定する場合は、サーバを再起動する前に行ってください。

ネイティブ モードの有効化 (必要な場合)

WebLogic Server と COM アプリケーションをネイティブ モードで通信させる場合は、WebLogic Server コンソールを介してすぐに有効化します。ネイティブ モードを使用するかどうかを判断する際は、第 1 章「WebLogic jCOM の概要」の「DCOM モードとネイティブ モード」を参照してください。

ネイティブ モードを有効化するには、以下の手順に従います。

1. Administration Console を起動します。
2. 左ペインで、サーバの名前をクリックします。
3. 右ペインの [接続] タブをクリックした後で [jCOM] タブをクリックして、[jCOM properties] 画面を表示します。
4. [ネイティブ モードを有効化] ボックスをチェックします。

その他の関連するコンソール プロパティのコンフィグレーション

その他の必要な jCOM コンソール プロパティをコンフィグレーションします。詳細については、jCOM プロパティに関する Console オンライン ヘルプを参照してください。

jCOM コンフィグレーションを変更した場合は、サーバを再起動する必要があります。Administration Console からも、変更を有効にするためにサーバを再起動する必要のあることが示されます。

COM オブジェクトを呼び出すためのサーバ コードの準備

COM オブジェクトを呼び出すためにサーバ コードを準備します。

com2java により生成された Java クラスの使い方

`com2java` ツールは、型ライブラリ内で COM クラスを発見するたびにその COM クラスにアクセスするための Java クラスを生成します。生成された Java クラスは、複数のコンストラクタを持ちます。

- デフォルト コンストラクタは、認証なしでローカル ホスト上に COM クラスのインスタンスを作成します。
- 2 番目のコンストラクタは、認証なしで特定のホスト上に COM クラスのインスタンスを作成します。
- 3 番目のコンストラクタは、特定の認証でローカル ホスト上に COM クラスのインスタンスを作成します。

- 4 番目のコンストラクタは、特定の認証で特定のホスト上に COM クラスのインスタンスを作成します。
- 最後のコンストラクタは、返されたオブジェクト参照をラップするために使用できます。これは COM クラスのインスタンスを参照します。

以下に DataLabelProxy クラスから生成されたコンストラクタのサンプルを示します。

```
public DataLabelProxy() {}


        public DataLabelProxy(Object obj) throws java.io.IOException
{
                super(obj, DataLabel.IID);

        }

        protected DataLabelProxy(Object obj, String iid) throws
        java.io.IOException

        {

        super(obj, iid);

        }

        public DataLabelProxy(String CLSID, String host, boolean
        deferred) throws java.net.UnknownHostException,
        java.io.IOException{ super(CLSID, DataLabel.IID, host,
null);

        }

        protected DataLabelProxy(String CLSID, String iid, String
host,
        AuthInfo authInfo) throws java.io.IOException { super(CLSID,
        iid, host, authInfo);

        }
```

COM インタフェースから com2java により生成された Java インタフェースの使い方

COM インタフェースのメソッドは、特定のインタフェースを介してオブジェクトの参照を返すことができます。

たとえば、Excel 型ライブラリ (Excel8.olb) は、_Application COM インタフェースを定義し、そのメソッド Add は COM IDL で次のように定義されます。

```
[id(0x0000023c), propget, helpcontext(0x0001023c)]
HRESULT Workbooks([out, retval] Workbooks** RHS);
```

このメソッドは、Workbooks COM インタフェースを実装するオブジェクトの参照を返します。Workbooks インタフェースは _Application インタフェースと同じ型ライブラリに定義されているので、com2java ツールは作成する _Application Java インタフェースに次のメソッドを生成します。

```
/** * getWorkbooks.
*
* @return Workbooks への参照
* @exception java.io.IOException 通信に問題がある場合
* @exception com.bea.jcom.AutomationException リモート サーバが例外を送出する場合 */
public Workbooks getWorkbooks () throws java.io.IOException,
com.bea.jcom.AutomationException;
```

生成された _ApplicationProxy Java クラスのメソッドの実装が見えるようになります。

```
/**
* getWorkbooks.
*
* @return Workbooks への参照
* @exception java.io.IOException 通信に問題が
ある場合
* @exception com.bea.jcom.AutomationException リモート サーバが
例外を送出する場合
*/
public Workbooks getWorkbooks () throws java.io.IOException,
```

```
com.bea.jcom.AutomationException{ com.bea.jcom.MarshalStream
marshalStream = newMarshalStream("getWorkbooks");
marshalStream = invoke("getWorkbooks", 52, marshalStream);
Object res = marshalStream.readDISPATCH("return value");
Workbooks returnValue = res == null ? null :new
WorkbooksProxy(res);
checkException(marshalStream,
marshalStream.readERROR("HRESULT"));
return returnValue;
}
```

見て分かるとおり、getWorkbooks メソッドは生成された WorkbooksProxy Java クラスを内部的に利用します。前述のとおり、com2java ツールは、Workbooks 戻り値型を持つメソッドを生成します。Workbooks インタフェースは _Application と同じ型ライブラリに定義されているからです。

Workbooks インタフェースが異なる型ライブラリに定義されている場合、WebLogic jCOM は次のコードを生成します。

```
/**
* getWorkbooks.
*
* @return Workbooks への参照
* @exception java.io.IOException 通信に問題が
ある場合
* @exception com.bea.jcom.AutomationException リモート サーバが
例外を送出する場合
*/
public Object getWorkbooks () throws java.io.IOException,
com.bea.jcom.AutomationException;
```

この場合、生成されたプロキシ クラスを明示的に使用して返された Workbooks にアクセスする必要があります。

```
Object wbksObj = app.getWorkbooks();
Workbooks workbooks = new WorkbooksProxy(wbObj);
```

4　jCOM ツールの詳細

以下の節では、jCOM アプリケーションで使用するツールについて詳しく説明します。

- com2java
- java2com
- regjvm
- regjvmcmd
- regtlb

com2java

WebLogic jCOM の `com2java` ツールは、型ライブラリから情報を読み込み、その型ライブラリに定義されている COM クラスおよびインタフェースにアクセスするための Java ファイルを生成します。

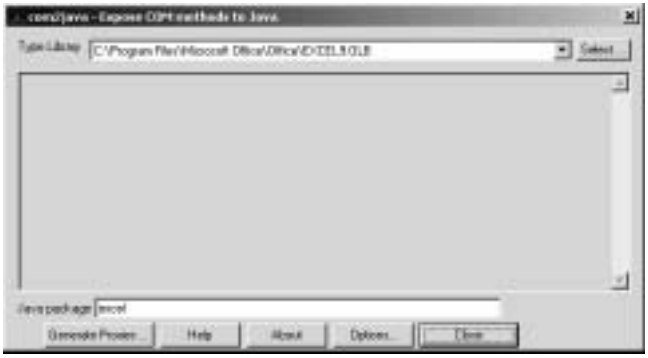
型ライブラリには、COM クラス、インタフェース、およびその他のコンストラクトが登録されています。通常、これらは Visual C++ や Visual BASIC などの開発ツールによって生成されます。

このため、一部の型ライブラリは簡単に識別可能です。拡張子 `olb` または `tlb` で終わるファイルは、確実に型ライブラリです。しかし、少々ややこしいのは、型ライブラリは実行ファイルなどの他のファイルに格納される場合があることです。Visual BASIC は、生成した実行ファイルに型ライブラリを格納します。

com2java の使い方

com2java は、コマンド シェルで入力するか、そのアイコンをダブルクリックして起動します。

com2java を起動すると、次のダイアログが表示されます。



型ライブラリの選択

ツールで処理する型ライブラリを選択するには、[Select ...] ボタンをクリックします。

型ライブラリは、COM コンポーネントを含んでいる実行ファイルやダイナミック リンク ライブラリ (DLL) ファイルなどの内部に隠されている場合もあります。

com2java ツールは、以前にオープンしてそのプロキシを生成した型ライブラリのリストを記憶しています。

Java パッケージ名の指定

com2java ツールは、型ライブラリの COM クラスおよびインタフェースに対応する Java ソース ファイルを生成します。生成したファイルは、特定のパッケージに格納できます。たとえば、Excel 型ライブラリのすべてのファイルを excel という Java パッケージに格納できます。

[Java package] テキスト ボックスに、生成されたファイルを格納するパッケージの名前を指定します。

com2java ツールは、特定の型ライブラリに対して指定した直前のパッケージ名を記憶しています。

オプション

[Options...] ボタンをクリックすると、以下で説明する com2java 用のオプションを選択するためのダイアログ ボックスが表示されますこれらのオプションは、com2java のセッション間で自動的に保存されます。特定のプロキシの生成処理でのみオプションが必要な場合は、プロキシの生成後にオプションをリセットします。

オプション	説明
[Clash Prefix]	型ライブラリに定義されている COM インタフェースのメソッドが既に Java で使用されているメソッド (getClass() メソッドなど) と衝突する場合、com2java は生成したメソッドの名前の先頭に zz_ という文字列 (デフォルト) を付ける。
[Lower case method names]	Java メソッド名の命名規則では、メソッド名は小文字で開始しなければならない。com2java ツールはデフォルトでこの規則を採用し、それに応じてメソッド名を変更する。com2java でこの規則を無視する場合は、[Options] ダイアログ ボックスの [Lower case method names] チェックボックスをオフにする。

オプション	説明
[Only generate IDispatch]	WebLogic jCOM は、IDispatch および vtable アクセスを使用した COM オブジェクトの呼び出しをサポートしている。このオプションを選択すると、すべての呼び出しは IDispatch インタフェースを使用して行われる。
[Generate retry code on '0x80010001 - Call was rejected by callee']	COM サーバがビジーの場合、エラー コードを受け取る場合がある。このオプションを選択すると、このエラー コードを受け取るたびにコードが再試行される。
[Generate Arrays as Objects]	SAFEARRAY のパラメータは、生成される java.lang.Object 型の対応する Java パラメータを持っている。これは、Variant の外部の 2 次元配列を Java から COM オブジェクトへ、または COM オブジェクトから受け渡す場合に必要。 このオプションは、実際に渡されるものを変更しない。それは依然として配列であり、生成された Java インタフェースに存在する。生成されたメソッド プロトタイプで配列型を生成せずに、「Object」を指定する。これは、2D 配列を受け渡すときに役立つ。COM IDL では、次元の数は SAFEARRAY に対して指定されない。また、[Generate Arrays as Objects] オプションをチェックしなかった場合、WebLogic jCOM は単一要素配列を受け渡したと仮定して対応するプロトタイプを生成する。 このオプションを選択し、`com2java` で「String[]」(たとえば) の代わりに「Object」を生成することにより、自由に 2D 文字列配列を受け渡すことができる。
[Prompt for names for imported tlbs]	型ライブラリは、別の型ライブラリをインポートする場合がある。インポートされた型ライブラリのプロキシも生成する場合、このオプションを選択するとこれらのプロキシのパッケージ名を指定できる。
[Don't generate dispinterfaces]	このオプションを選択すると、ディスパッチインタフェースとして定義されているインタフェースのプロキシの生成が禁止される。

オプション	説明
[Generate deprecated constructors]	生成されるプロキシには、現在は非推奨となっているコンストラクタがいくつか含まれる。これらのコンストラクタを生成したくない場合は、このオプションを選択する。
[Don't rename methods with same names]	COM クラスで名前の衝突が検出された場合、com2java はそのメソッドの 1 つの名前を自動的に変更する。このオプションを選択すると、名前の自動変更が無効になる。
[Ignore conflicting interfaces]	COM クラスが同名のメソッドを定義する複数のインタフェースを実装する場合、このオプションを選択すると、対応する Java クラスは余計なインタフェースを実装しない。生成される getAsXXX メソッドを使用すれば、これらのインタフェースに引き続きアクセスできる。生成されるコメントを参照のこと。
[Generate Java Abstract Window Toolkit (AWT) classes]	Java クラスを GUI クラスとして生成する。ActiveX コントロールを Java フレームに埋め込むために使用する。

プロキシの生成

[Generate Proxies] ボタンをクリックすると、com2java ツールによって生成される Java ファイルを格納するディレクトリを選択できます。

ディレクトリを選択すると、com2java は型ライブラリを解析し、対応するファイルを指定したディレクトリに出力します。既にディレクトリに Java ソースファイルが存在する場合、WebLogic jCOM は警告メッセージを発行し、操作をキャンセルできるようにします。

com2java により生成されたファイル

com2java ツールは、型ライブラリに含まれる以下の 3 種類のコンストラクトを扱います。

- 列挙値
- COM インタフェース
- COM クラス

これについてはこの節で説明します。

生成された Java ファイルを使用して COM オブジェクトを操作する方法については、アクセスする COM オブジェクトに関するドキュメントを参照してください。

たとえば、`com2java` を Excel 型ライブラリに対して実行する場合、生成される Java ファイルは Microsoft Excel COM API に対応しています。このため、Microsoft Excel のプログラミング ドキュメントを参照する必要があります。たとえば、Excel 2000 COM API については下記を参照します。

http://msdn.microsoft.com/library/default.asp?URL=/library/officedev/off2000/xltocobjectmodelapplication.htm

列挙値

Java では、列挙値は `java.util.Enumeration` で表されるリストです。型ライブラリに列挙値が含まれている場合、WebLogic jCOM はその列挙中の要素ごとの定数定義を含んだ Java インタフェースを生成します。

COM インタフェース

WebLogic jCOM は、2 種類のインタフェースを処理します。1 つはディスパッチ インタフェースです。このメソッドには COM IDispatch メカニズムを使用しないとアクセスできません。もう 1 つはデュアル インタフェースです。このメソッドは直接呼び出すことができます (vtbl アクセス)。

型ライブラリに定義されている COM インタフェースごとに、`com2java` ツールは Java インタフェースと Java クラスの 2 つの Java ファイルを生成します。

生成される Java インタフェースの名前は、COM インタフェースの名前と同じです。たとえば、COM インタフェース名が `IMyInterface` の場合、`com2java` ツールは `IMyInterface` という Java インタフェースを `IMyInterface.java` ファイルに生成します。

`com2java` が生成するもう 1 つのファイルは Java クラスです。このファイルには、インタフェースを実装する COM オブジェクトにアクセスするためのコードと、インタフェースを実装する Java クラスのメソッドを COM オブジェクトが呼び出すためのコードが含まれます。生成される Java クラスの名前は、インタフェースの名前に「Proxy」を付加したものです。前項の例では、WebLogic jCOM は *IMyInterfaceProxy* という Java クラスを *IMyInterfaceProxy.java* というファイルに生成します。

COM インタフェースのメソッドごとに、WebLogic jCOM は Java インタフェース内に対応するメソッドを生成します。インタフェース内に生成される一部の定数は、生成されるコメントが示すように無視して構いません。それらについて知る必要も、使用する必要もありません。

WebLogic jCOM は、インタフェースとそのメソッドを記述した型ライブラリからコメントを取り出し、それらを生成された javadoc コメントで使用します。

COM クラス

COM クラスは、1 つまたは複数の COM インタフェースを実装します。これは、Java クラスが 1 つまたは複数の Java インタフェースを実装できるのと同じです。

型ライブラリの COM クラスごとに、`com2java` ツールは、その COM クラスと同名の対応する Java クラスを生成します。WebLogic jCOM は、複数のインタフェースを実装するクラスもサポートしています。

WebLogic jCOM が生成する Java クラスを使用すると、対応する COM クラスにアクセスできます。

特殊なケース -- ソース インタフェース (イベント)

COM クラスは、インタフェースをソース インタフェースとして指定できます。このため、そのインタフェースを実装する COM クラスのインスタンスは、そのインタフェースに定義されているイベントをサブスクライブできます。サブスクライブしたオブジェクトに対してインタフェース内のメソッドが呼び出されます。

注意 : `com2java` ツールが型ライブラリ内のインタフェースをイベント インタフェースとして扱うには、そのインタフェースをソース インタフェースとして使用する COM クラスが少なくとも 1 つは型ライブラリに存在しなければなりません。

COM イベントは接続ポイントとソース インタフェースを使用して動作しますが、Java のイベント メカニズムはそれとは異なります。`com2java` ツールは、COM メカニズムを Java プログラムから完全に隠し、標準 Java テクニックを利用してイベントを提供します。

具体的には、`com2java` は COM クラスにアクセスするために生成した Java クラスに 2 つのメソッドを追加します。

`com2java` ツールは、クラスがインタフェースをソース インタフェースとして使用していることを検出すると、そのインタフェース用の特殊なコードを生成します。com2java ツールは、Java イベントの命名規則に従って、インタフェースを `java.util.EventListener` Java インタフェースから派生させます。

もう 1 つの Java イベントの規則は、インタフェースの各メソッドは単一のパラメータを持つ必要があるということです。これは、`java.util.EventObject` Java クラスから派生したクラスのインスタンスです。

最後の Java イベント関連の規則の 1 つは、アダプタ クラスの使用です。このクラスは、イベント インタフェースを実装し、インタフェースのメソッド用の空のデフォルト実装を提供します。イベントにサブスクライブされるクラスを作成する開発者は、インタフェースのすべてのメソッドを実装する必要がありません (特に大規模なインタフェースを使用するときに面倒になります)。

イベント インタフェースごとに、WebLogic jCOM はアダプタ クラスを生成します。

java2com

java2com は、どのプラットフォームでも実行できます。WebLogic jCOM ランタイム weblogic.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、Java クラスを (Java 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。

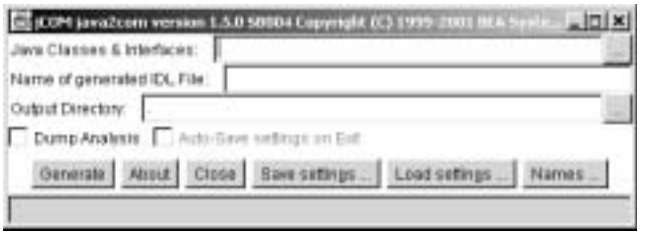
- COM インタフェース定義言語 (IDL) ファイル。
- pure Java DCOM マーシャリング コード (ラッパー)。これは、vtable (レイトバインド) アクセスを使用して COM から Java オブジェクトへのアクセスを容易にするために WebLogic jCOM ランタイムによって使用されます。

これらのファイルを生成した後、Microsoft の MIDL ツールを使用して IDL ファイルをコンパイルします。

IDL ファイルとラッパーを生成するには、最初に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
```

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)。

フィールド	説明
[Java Classes and 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する。これらは、CLASSPATH でアクセス可能でなければならない。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する。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する。 クラス名はスペースで区切って入力する。[...] 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る。
[Name of Generated IDL File]	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。myjvm と指定した場合は、myjvm.idl が生成される。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る。
[Output Directory]	java2com が生成したファイルを出力するディレクトリ。デフォルトはカレント ディレクトリ (「.」)。
[Dump Analysis]	java2com が発見したクラスをそのまま表示する。
[Save Settings] と [Load Settings]	[Save Settings] ボタンをクリックすると、現在の java2com 設定が保存される。この操作を行ってから、[Generate] をクリックする。 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する。存在する場合、そのファイルから設定を自動的にロードする。

フィールド	説明
[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る。 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る。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 名を指定できる。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たない。 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る。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 名にマップする。[Add this Member Name map] をクリックすると、選択したメンバーを指定した COM 名にマップできる。

フィールド	説明
[Generate] ボタン	このボタンをクリックすると、ラッパーおよび IDL ファイルが生成される。 `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る。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` となる。 `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る。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` となる。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` を生成する。 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) を持つ。 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する。このツールは Visual C++ に付属しており、Microsoft Web サイトからダウンロードできる。次のコマンド、 `midl procdServ.idl` は、`prodServ.tlb` という型ライブラリを生成する。このライブラリは、4-23 ページの「regtlb」で説明するとおりに登録する必要がある。

regjvm

WebLogic jCOM を使用して 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るようにするには、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て新しいエントリを登録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 *`WL_HOME`*`\server\bin` ディレクトリに格納されています)。

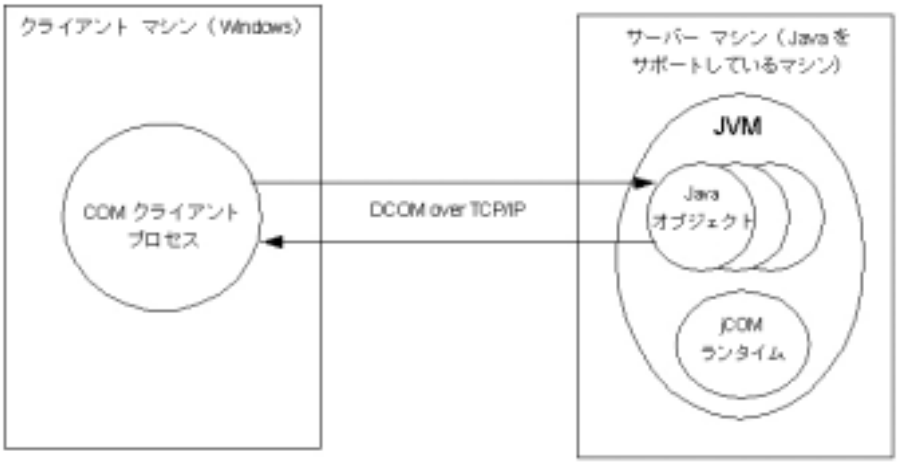
JVM モード

COM クライアントから JVM には、以下のモードでアクセスできます。

- DCOM モード
- ネイティブ モード (アウト オブ プロセス)
- ネイティブ モード (イン プロセス)

DCOM モード

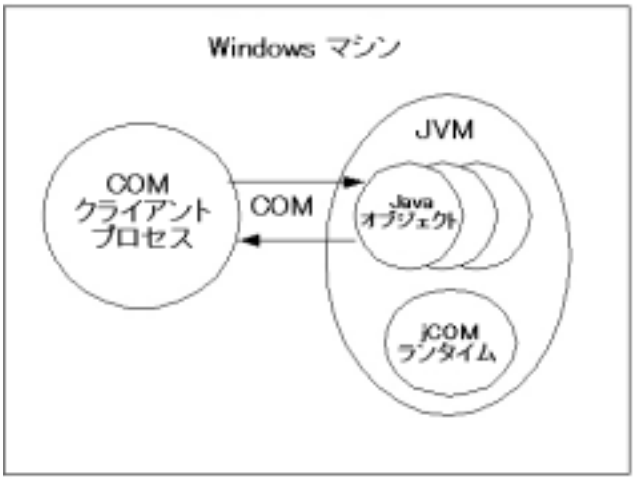
DCOM モードでは、Java サーバ サイド上にネイティブ コードは不要です。このため、Java コードは Java 仮想マシンがインストールされている Unix マシンや他のマシン上に配置できます。Windows クライアント マシン上で JVM を登録する場合、サーバのホスト マシン名 (ローカル コンポーネント用のローカルホストの場合もあります) とポート番号を定義します。



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。

ネイティブ モード アウト オブ プロセス

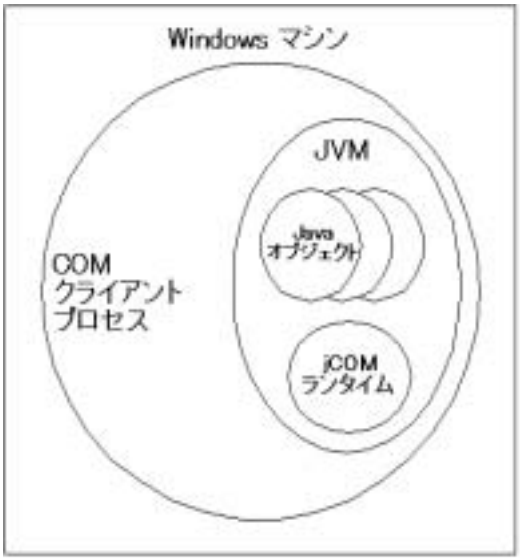
ネイティブ モードは、現在ローカル マシン上でのみ動作します。JVM 名以外にパラメータは必要ありません。



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。

ネイティブ モード イン プロセス

イン プロセスのネイティブ モードを使用すると、Java オブジェクトを COM クライアントと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。

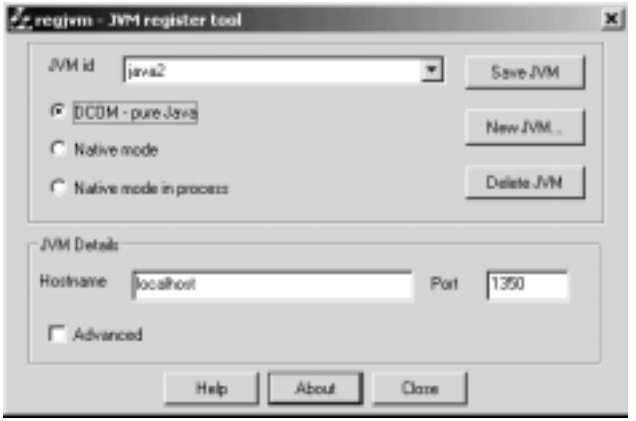
regjvm GUI ツールのユーザ インタフェース

`regjvm` ツールを実行すると、以下のダイアログ ボックスが表示されます。

- 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。JVM は変更、追加、または削除できます。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 モードを選択します。その際、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。

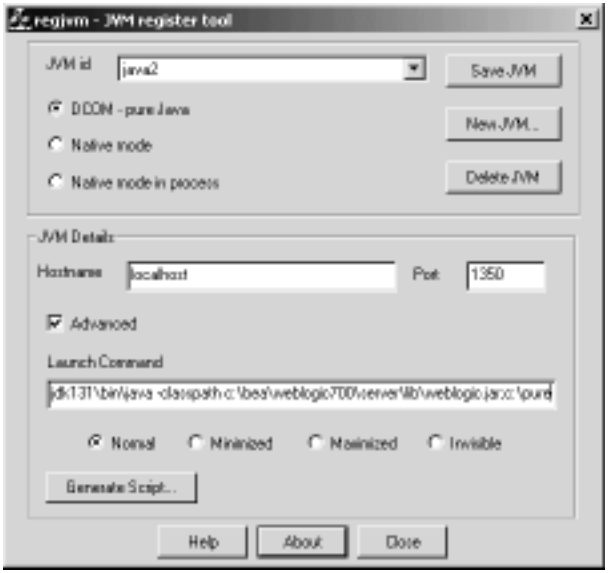
  これらのオプションについては、以下の節で説明します。

regjvm GUI ツールの DCOM モードオプション



標準オプション

- [JVM id](必須) - JVM を指定する必要があります。[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。
- [Hostname] - JVM が存在するホスト名または IP アドレスです。
- [Port] - JVM との通信を開始するためのポート番号です。



詳細オプション

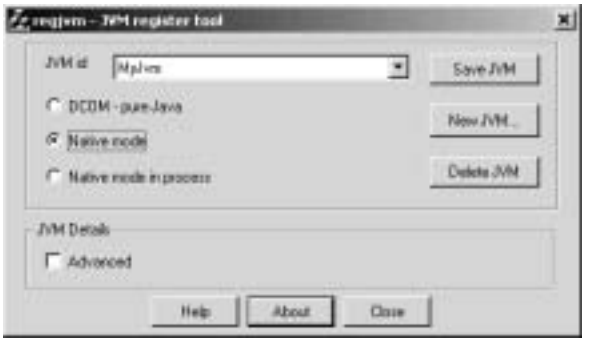
- [Launch command](必須) - JVM が自動的に起動する場合に使用されるコマンドです。通常、このコマンドは次のようになります。

  ```
  c:\bea\jdk131\bin\java -classpath
  c:\bea\weblogic700\server\lib\weblogic.jar;c:\pure MyMainClass
  ```

  `weblogic.jar` および該当する `jdk` ファイルが CLASSPATH にあることが重要です。

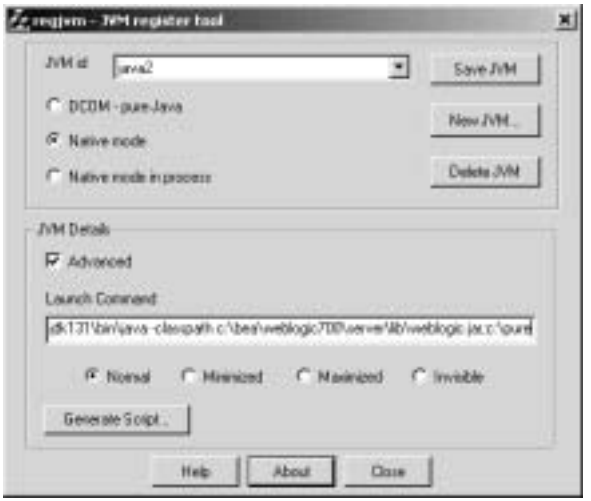
- [Generate Script...](省略可能) - JVM の設定を選択するレジストリ スクリプトを生成できます。

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

- [JVM id](必須) - JVM を指定する必要があります。[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

詳細オプションは、DCOM モードの詳細オプションと同じです。4-18 ページの「regjvm GUI ツールの DCOM モードオプション」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 id](必須) - JVM を指定する必要があります。[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。次の構文が必要です。prop1=value1 prop2=value2...
- [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 モードの場合と同じです。4-18 ページの「regjvm GUI ツールの DCOM モードオプション」を参照してください。

regjvmcmd

regjvmcmd は、4-13 ページの「regjvm」で説明した 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これを実行します。

```
regjvmcmd
```

最も単純な形式の regjvmcmd 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 に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 WebLogic Server の起動時に指定する TCP/IP ポートです。

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、まず次のコマンドを使用してその登録を解除する必要があります。

```
regjvmcmd /unregister JvmId
```

regtlb

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必要のある 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。

5　アップグレードに関する考慮事項

以下の節では、WebLogic jCOM 6.1 から WebLogic jCOM 7.0 へのアップグレードについて説明します。

- jCOM 7.0 の実装のメリット
- COM コードへの変更
- セキュリティの変更
- コンフィグレーションの変更

jCOM 7.0 の実装のメリット

WebLogic jCOM 7.0 は、次の理由から、WebLogic jCOM 6.1 に比べて実装がきわめて簡単になりました。

- ブリッジの書き込みやインストールはもう必要ありません。jCOM ランタイムは WebLogic Server に組み込まれています。WebLogic Server をインストールすると、jCOM 機能は自動的にインストールされます。
- COM マシンで必要なソフトウェアは、WebLogic Server インストール ディレクトリから `.dll` ファイルと `.exe` ファイルをコピーして取得します。
- jCOM が自動的に有効になります。つまり、WebLogic Server がリスン ポート上で COM 呼び出しをリスンするよう自動的にコンフィグレーションされます。
- jCOM プロパティは、WebLogic Server の Administration Console だけでコンフィグレーションできるようになりました。

COM コードへの変更

7.0 にアップグレードすると、COM アプリケーション コードに次の影響があります。

- ゼロ クライアント アプリケーションを実行している場合は、WebLogic Server で実行中のサーブレットからプログラムによってオブジェクト参照モニカ (ORM) を取得できます。また、従来のように `com.bea.jcom.GetJvmMoniker` を実行して取得することもできます。

  ORM をサーブレットから取得するには、WebLogic Server で Web ブラウザを開いて http://[wlshost]:[wlsport]/bea_wls_internal/com に移動します。

- COM コードから、個別の jCOM ブリッジへの参照をすべてパージします。

セキュリティの変更

以前は jCOM 固有のソフトウェアで処理されていたセキュリティが、WebLogic Server のロールおよびポリシーのセキュリティ メカニズムによって実装されるようになります。つまり、COM クライアントが WebLogic Server オブジェクトにアクセスできるようにするには、COM クライアントが使用できるようにそれらのオブジェクトをエクスポートする必要があります。この処理は、WebLogic Server Administration Console で行います。

詳細については、第 2 章「COM クライアント アプリケーションから WebLogic Server への呼び出し」の「アクセス制御のコンフィグレーション」を参照してください。

コンフィグレーションの変更

プロパティはコマンド ラインではなくコンソールでコンフィグレーションされるようになり、多くのプロパティが廃止されました。次の表は、6.1 のプロパティと 7.0 のプロパティの関係を示しています。

6.1 のプロパティ	7.0 での処理
ENABLE_TCP_NODELAY	必要ない。
JCOM_DCOM_PORT	必要ない。ポート WebLogic Server に対する新しいポートのデフォルトは、通常、7001 でリスンする。
JCOM_COINIT_VALUE	WebLogic Server Administration Console の [アパートメント スレッド] プロパティでコンフィグレーションする。
JCOM_INCOMING_CONNECTION_TIMEOUT	WebLogic Server Administration Console の [COM メッセージ タイムアウト] プロパティでコンフィグレーションする。
JCOM_OUTGOING_CONNECTION_TIMEOUT	指定されたミリ秒の間使用されていない送信時接続 (WebLogic jCOM ランタイムによって開始された接続) が切断される。コンフィグレーションするには、コマンドライン (たとえば WebLogic Server の起動スクリプトの Java オプション) に「JCOM_OUTGOING_CONNECTION_TIMEOUT = *[number of milliseconds]*」パラメータを追加する。
com.bea.jcom.server	WebLogic Server のリスン ポートを使用する。

6.1 のプロパティ	7.0 での処理
JCOM_MAX_REQUEST_HANDLERS	jCOM スレッディングは WebLogic Server スレッド プールと統合されたので、この設定値は WebLogic Server 用にコンフィグレーションされるスレッドの数に一致する。
JCOM_NATIVE_MODE	WebLogic Server Administration Console の [ネイティブ モードを有効化] プロパティでコンフィグレーションする。
JCOM_NOGIT	必要ない。
JCOM_NTAUTH_HOST	WebLogic Server Administration Console の [NTAuth Host] プロパティでコンフィグレーションする。
JCOM_LOCAL_PORT_START	必要ない。この範囲に対しては WebLogic Server のリスン ポートを使用する。
JCOM_LOCAL_PORT_END	必要ない。この範囲に対しては WebLogic Server のリスン ポートを使用する。
JCOM_PROXY_PACKAGE	必要ない。
JCOM_SKIP_CLOSE	必要ない。WebLogic Server は [COM メッセージ タイムアウト] プロパティの値に基づいて接続をクローズする。
JCOM_WS_NAME	必要ない。WebLogic jCOM は、`CreateObject` 文で起動されたサーバ インスタンスの名前を使用する。