BEA WebLogic jCOM

リファレンス

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

4. ガベージ コレクションと参照カウント

5. ネイティブ モード

6. セキュリティ / 認証

7. サポートされる COM データ型とそれらに相当する Java

データ型

8. スレッディング

9. Java からの COM オブジェクトの使い方

10. jCOM.jar の Java プロパティ

11. トラブルシューティング

1 com2java ツール

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。

以下の節では、次のことを見ていきます。

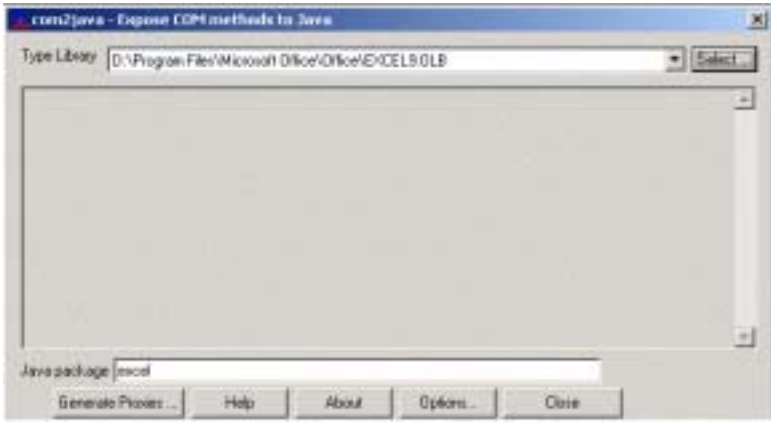
- com2java の使い方
- com2java によって生成されるファイル

com2java の使い方

com2java ツールは、WebLogic jCOM のインストール ディレクトリの *bin* サブディレクトリに存在します。たとえば、WebLogic jCOM を *c:\bea\wlserver6.1\jcom* にインストールした場合、com2java は *c:\bea\wlserver6.1\jcom\bin\com2java.exe* に存在します。

以下に説明する GUI インタフェースの他に、**com2javacmd** ツールによってコマンド ライン インタフェースも提供されています。詳細については、「com2javacmd /?」と入力します。

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 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入力します。

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

オプション ボタンをクリックすると、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。特定の生成処理でのみオプションが必要な場合、プロキシの生成後にオプションを必ずリセットする必要があります。オプションは以下のとおりです。

[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）を付けます。

[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りません。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します。メソッド名を変更したくない場合は、[`Options`] ダイアログ ボックスの [`Lowercase method names`] チェックボックスをオフにします。

[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ます。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ます。

[Generate retry code on '0x80010001 - Call was rejected by callee']

COM サーバがビジーの場合、このエラーを受け取る場合があります。このオプションを選択すると、このエラーを受け取るたびにコードが再試行します。

[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 パラメータを持ちます。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です。

このオプションは、実際に渡されるものを変更しません。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します。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」を指定します。これは、2D 配列を受け渡すときに役立ちます。COM IDL では、次元の数は SAFEARRAY に対して指定されません。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します。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ます。

[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ります。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ます。

[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ます。

[Generate depreciated constructors]

生成されるプロキシには、現在は廃止されているコンストラクタがいくつか含まれます。これらのコンストラクタを 1 つも生成したくない場合は、このオプションを選択します。

[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します。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ります。

[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ません。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依然としてアクセスできます。生成されるコメントを参照してください。

[Generate Java AWT classes]

.Java クラスを GUI クラスとして生成します。ActiveX コントロールを Java フレームに埋め込むために使用します。

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 によって生成されるファイル

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（ソース インタフェース ― イベントを含む）
- COM クラス

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を理解するには、アクセスする COM オブジェクトに関連するドキュメントを参照する必要があります。

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

生成されたファイルを元の API に関連付けるのが難しい場合があります。このような場合は、BEA にお問い合わせください。別の COM クライアント（アーリーバインド VB、VC++ など）から正常に動作するコードを提供していただければ、同じことを Java から実行できるようにします。

列挙値

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 COM 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。

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 ファイルとして使用していることに気付くと、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）。

このため、各イベント インタフェースについて、WebLogic jCOM はアダプタ クラスを生成します。

まとめ - イベント

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。

2 WebLogic jCOM および COM イベント

COM クラスがイベントを生成できる（ソース インタフェースを持つ）場合、WebLogic jCOM の com2java ツールは、Java オブジェクトが標準 Java セマンティクスを使用してそれらのイベントをサブスクライブするためのコードを生成します。生成されるファイルについては、com2java ドキュメントの関連セクションを参照してください。

要約すると、COM クラスが同じ型ライブラリに定義されているソース インタフェースを持つ場合、WebLogic jCOM の com2java ツールは以下のことを行います。

- COM クラスにアクセスするために生成する対応 Java クラスに *addXYZListener* メソッドと *removeXYZListener* メソッドを生成します。
- イベント インタフェースを *java.util.EventListener* から派生させます。
- イベント インタフェースのメソッドごとに、*java.util.EventObject* から派生するクラスを作成します。これには元のすべてのメソッド パラメータが含まれ、そのインスタンスはそのメソッドの唯一のパラメータとして渡されます。
- イベント インタフェースを実装するアダプタ クラスを生成し、インタフェースの各メソッドの空のデフォルト実装を提供します。

Java プログラマからは、それ以外のものは何も存在しません。標準 Java イベント テクニックを使用します。つまり、イベント リスナ インタフェースを実装し、*addXYZListener* メソッドを使用してイベントをサブスクライブします。

3 例外

WebLogic jCOM は、COM の例外と Java の例外を透過的にマップします。以下の節では、次のことを見ていきます。

- COM コンポーネントで発生する例外
- Java オブジェクトの例外
- 例外インターセプト メカニズム

COM コンポーネントで発生する例外

Java クライアントが COM コンポーネントのメソッドを呼び出した場合、そのコンポーネントが例外を生成する場合があります。

COM 例外には、エラー番号、ソース、および説明が関連付けられています。

COM コンポーネント内での例外の発生

Visual BASIC では、例外は次のように発生します。

```
Public Sub errorMethod()
       Err.Raise vbObjectError + 1051, "My program", "This is the
description"
End Sub
```

Visual C++ では、例外は次のように発生します。

```
AfxThrowOleDispatchException(0, "I am sorry Dave, I can't do
that...");
```

Java での例外の捕捉

COM コンポーネントが例外を送出すると、com.bea.jcom.AutomationException のインスタンスが Java クライアントに送出されます。このクラスに関連する javadoc ドキュメントは次のとおりです。

AutomationException クラスは、メソッドを公開し、送出された COM 例外のエラー番号、ソース、および説明を取得します。

```
import com.bea.jcom.AuthInfo;
import java.net.UnknownHostException;
import java.io.IOException;
public class Example {

       public static void main(java.lang.String[] args) throws
IOException, UnknownHostException {

              vbexcep.Class1 c1 = null;

              try {
              c1 = new vbexcep.Class1();
              c1.errorMethod();
              } catch(com.bea.jcom.AutomationException ae) {
              System.out.println("Caught: " + ae);
              System.out.println("Source: " + ae.getSource());
              System.out.println("Description: " +
ae.getDescription());
              System.out.println("Code: " + ae.getCode());
              } finally {
              com.bea.jcom.Cleaner.releaseAll();
              }
       }
}
```

この例の実行結果は次のとおりです。

AutomationException に代わる IOException の使用

AutomationException クラスを使用しない場合、代わりに java.io.IOException を捕捉できます。AutomationException は IOException から派生するからです。これにより、エラー コード、説明、およびソースにアクセスできないことを犠牲にして、COM 関連クラスに対するコードの依存性を減らすことができます。

Java オブジェクトで発生する例外

COM コンポーネントは、WebLogic jCOM を使用して Java オブジェクトのメソッドを呼び出します。

このようなメソッドが例外を生成した場合、WebLogic jCOM はその例外を捕捉し、それを適切な COM 例外に変換します。

たとえば、次の Java コードを見てみます。「method1」メソッドは特に具体的なものではありません。

```
import java.io.*;
public class Simple {
public static void main(String[] args) throws Exception {

com.bea.jcom.Jvm.register("firstjvm");
Thread.sleep(6000000); // 1 時間の休止
}

public void method1() throws java.io.IOException {
throw new java.io.IOException("A deliberate exception");
}
}
}
```

Visual Basic クライアント コードは次のとおりです。

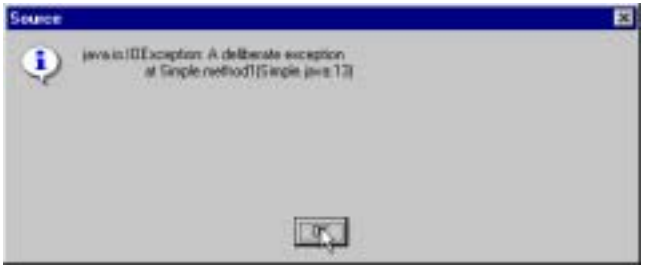
```
Public Sub method1(ByVal p1 As Object)
Set p1 = GetObject("firstjvm:Simple")
On Error GoTo ErrorHandler
p1.method1

ErrorHandler: ' Error-handling routine.
MsgBox Err.Source, vbInformation, "Source"
MsgBox Err.Description, vbInformation, "Description"
MsgBox Str(Err.Number), vbInformation, "Code"

End Sub
```

Visual BASIC コードは、単にエラー ハンドラを確立し、Java オブジェクトの呼び出しを行います。この例の Java オブジェクトは、Visual BASIC エラー ハンドラによって捕捉される例外を故意に生成します。

このエラー ハンドラは、エラー情報を示す一連のメッセージ ボックスを表示します。

ソース

WebLogic jCOM は、エラーを生成したメソッドのスタック トレースを自動的に [*Source*] に入力します。



説明

WebLogic jCOM は、*Exception.getMessage()* から返された文字列を自動的に [*Description*] に入力します。

WebLogic jCOM の内部メソッドを示すスタック トレースは削除してあります。

コード

WebLogic jCOM は、自動的にコードを 0x80020009 に設定します。これは「Exception occurred.」用の COM です。

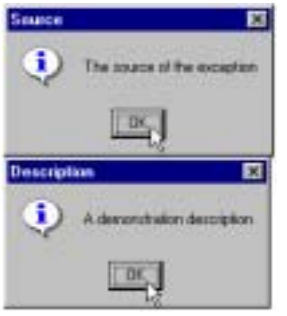
独自のソース / 説明 / コードを指定したい場合

WebLogic jCOM のデフォルトをそのまま使用せずに、エラー情報を明示的に設定したい場合は、com.bea.jcom.AutomationException のインスタンスを送出できます。

上の例の Java コードを次のように変更したとします。

```
public void method1() throws com.bea.jcom.AutomationException {
long code = 0x80020009;
String source = "The source of the exception";
String description = "A demonstration description";
throw new com.bea.jcom.AutomationException(code, source,
description);
}
```

この場合、以下のメッセージ ボックスが表示され、明示的に設定した情報が示されます。



例外インターセプト メカニズムの使い方

また、例外が COM クライアントに返されるときに呼び出されるコードを提供するためのフックも用意されています。これを使用すると、例外情報を変更したり、COM クライアントが後で取得できるよう例外オブジェクトを保存したりできます。

COM-to-Java から呼び出されたときに生成された例外をインターセプトするには、次のようにインターセプタを一度登録します。

```
com.bea.jcom.ExceptionInterceptor interceptor = new
YourInterceptor();
com.bea.jcom.AutomationException.setExceptionInterceptor
(interceptor);
```

次のように、ExceptionInterceptor インタフェースを実装するクラスを作成します。

```
class YourInterceptor implements com.bea.jcom.ExceptionInterceptor
{

       public com.bea.jcom.AutomationException
handleException(Throwable t) {

    ...

  }

}
```

デフォルト エラーが COM クライアントに返されるようにする場合、ハンドラで null を返します。それ以外の場合、次のような特定のエラーを生成できます。

```
public com.bea.jcom.AutomationException handleException(Throwable
t) {

       System.out.println("Intercepting: " + t);
       long code = 0x80020009;
       String source = "The source of the exception";
       String description = "A demonstration description";
       return new com.bea.jcom.AutomationException(code, source,
description);
}
```

4 ガベージ コレクションと参照カウント

Java クライアントから参照される COM オブジェクト

Java 仮想マシンは、COM オブジェクトの Java 参照がこれ以上アクセスされない場合、その参照に対してガベージ コレクションを実行します。

こうした参照にガベージ コレクションを行うと、WebLogic jCOM はその DCOM オブジェクトの詳細を、解放される DCOM オブジェクト参照の内部リストに追加します。WebLogic jCOM デーモン スレッドは、ガベージ コレクションによって 10 秒ごとにこれらの DCOM オブジェクト参照を解放します。

オブジェクト参照を手動で**明示的に解放**する場合は、*com.bea.jcom.Cleaner.release(...)* メソッドを呼び出して、そのオブジェクト参照をパラメータとして渡します。

JVM がシャットダウンするときに、*com.bea.jcom.Cleaner.releaseAll()* を呼び出す必要があります。これにより、ガベージ コレクションでまだ解放されていない COM オブジェクト参照がすべて解放されます。このメソッドを呼び出したら、WebLogic jCOM を介してアクセスした COM オブジェクトを使用することができなくなります。

DCOM モードで動作している場合、WebLogic jCOM ランタイムは DCOM プロトコルに従って DCOM ping メッセージを送信して、クライアントがまだ応答していることを COM サーバに知らせます。

COM クライアントから参照される Java オブジェクト

COM クライアントが Java オブジェクトの参照を保持している場合、WebLogic jCOM ランタイムは COM クライアントに代わってその Java オブジェクトの参照を JVM 内部に保持します。また、Java オブジェクトにエクスポートされた COM 参照の数のカウントを保持し、COM 参照カウントがゼロになるとその参照を解放します。DCOM モードで動作している場合、WebLogic jCOM は、COM クライアントがまだ応答していることを知らせる DCOM ping メッセージを受信します。この ping メッセージが（DCOM 仕様に従って）6 分間受信されない場合、WebLogic jCOM ランタイムは ping されないすべての Java オブジェクトを解放します。

Java オブジェクトが COM クライアントによって参照されなくなったときに通知を受け取るようにしたい場合、次のメソッドを呼び出して、`com.bea.jcom.Unreferenced` インタフェースを実装する Java クラスのインスタンスの参照を渡します。

```
com.bea.jcom.Cleaner.addUnreferencedListener(com.bea.jcom.Unrefer
enced listener)
```

`Unreferenced` インタフェースは次のようになります。

```
public interface Unreferenced {
public void objectUnreferenced(Object o);
}
```

通知が必要でなくなったときは、次のメソッドを呼び出します。

```
public static void removeUnreferencedListener(Unreferenced
listener)
```

次に小さい例を示します。

```
public class MyJvm {

public static void main(String[] args) throws Exception {
com.bea.jcom.Jvm.register("firstjvm");

MyUnreferencedListener l = new MyUnreferencedListener();
com.bea.jcom.Cleaner.addUnreferencedListener(l);

Thread.sleep(6000000); // 1 時間の休止
```

```
com.bea.jcom.Cleaner.removeUnreferencedListener(l);
}
}

class MyUnreferencedListener implements com.bea.jcom.Unreferenced
{
public void objectUnreferenced(Object o) {
System.out.println("** Object no longer referenced: " + o);
}
}
```

5　ネイティブ モード

Java-COM ブリッジは、ネットワーク ベースの DCOM をサポートして Java オブジェクトが COM オブジェクトと対話できるようにするだけでなく、代わりの手段としてブリッジを実行するためのネイティブ コード（DLL）も提供します。

Java コードは、ネイティブ モードと DCOM のどちらを使用する場合も変化しません。

注意 : デフォルトによって、WebLogic jCOM は DCOM を使用します。ネイティブ モードは明示的に有効にする必要があります。

以下の節では、次のことを見ていきます。

- サポートの対象
- ネイティブ モードでの Java クライアントの実行
- ネイティブ モードでの COM クライアントの実行（JVM アウトオブプロセス）
- ネイティブ モードでの COM クライアントの実行（JVM インプロセス）
- サポートされない機能と確認済みの問題

サポートの対象

IDispatch と vtable

ネイティブ モードでは、IDispatch と Custom（vtable）メソッド呼び出しの両方をどちらの方向でも行うことができます。COM インタフェースは、デュアルである必要がありません（IUnknown から直接派生できます）。

インプロセスとアウトオブプロセス

WebLogic jCOM のネイティブ モードは、以下のものサポートします。

- アウトオブプロセス（リモートを含む）COM コンポーネントと対話する Java クライアント。
- インプロセス COM コンポーネントと対話する Java クライアント。この場合、COM コンポーネントの DLL は JVM のプロセスにインプロセスでロードされます。
- JVM が（同じマシン上で）独立したプロセスで動作している Java オブジェクトと対話する COM クライアント。
- JVM が COM クライアントのアドレス空間にインプロセスでロードされる Java オブジェクトと対話する COM クライアント。

すべての JVM

WebLogic jCOM は、任意のプラットフォームで動作する任意の JVM と一緒に使用できます。

Microsoft Transaction Server / COM+

Java オブジェクトを MTS（インプロセス）に適切にロードし、VB ベース クライアントからその MTS 上のメソッドを呼び出すことに成功しました。

Java オブジェクトは標準 IObjectControl COM インタフェース（com2java で生成された通常の Java インタフェースとして認識する）を実装し、MTS は通常のメソッド（activate、canBePooled、など）を呼び出すことができました。

最後に、WebLogic jCOM ランタイムで特殊なフックを使用して、Java オブジェクトから IObjectContext MTS/COM+ インタフェースにアクセスし、いくつかの属性をテストしました。トランザクションで動作するときなどに、正確に検出が行われました。

ネイティブ モードでの Java クライアントの実行

Java クライアントがネイティブ モードで COM オブジェクトにアクセスするためには、COM オブジェクトの型ライブラリに対して com2java ツールを実行し、WebLogic jCOM を DCOM モードで使用するときのように Java プロキシを生成します。

JVM を実行する場合、JCOM_NATIVE_MODE プロパティを定義してネイティブ モードを有効にします。
java -DJCOM_NATIVE_MODE JCOMBridge

COM オブジェクトのコンストラクタにホスト名を渡して、リモート COM コンポーネントを作成します。

WebLogic jCOM ランタイムは、最初にフラグを CLSCTX_ALL に設定して CoCreateInstanceEx を試みます。これに失敗した場合、フラグを CLSCTX_SERVER に設定して再試行します。

スレッド モデル

デフォルトでは、WebLogic jCOM ランタイムは COINIT_MULTITHREADED フラグを使用して COM を初期化します。異なるフラグで COM を初期化する場合は、JCOM_COINIT_VALUE プロパティを設定します。次に例を示します。

```
java -DJCOM_NATIVE_MODE -DJCOM_COINIT_VALUE=2 JCOMBridge
```

ネイティブ モードを使用して、DLL にホストされている COM コンポーネントと対話しようとしたときに **Class Not Registered というメッセージを取得した場合は、**JCOM_COINIT_VALUE を上のとおり 2 に設定してください。

WebLogic jCOM ランタイムは、すべての COM オブジェクト参照をグローバル インタフェース テーブル（GIT）という特別な COM テーブルに配置します。Java から COM に呼び出しが行われるたびに、オブジェクト参照が GIT から取得されます。これにより、呼び出しが現在のスレッドから行われるようになります。

ただし、一部の COM オブジェクト参照（MSHTML 内の Frames コレクションなど）は GIT に置かれないため、WebLogic jCOM は直接的なポインタ参照を格納します。この場合、作成するスレッドがそのオブジェクト上で呼び出しを行うようにする必要があります。この状況は非常にまれであり、何が起こったのかを知らせるメッセージが WebLogic jCOM ログに格納されます（ロギングが有効になっている場合）。

ネイティブ モードでの COM クライアントの実行（JVM アウトオブプロセス）

この節を読む前に、標準 DCOM Java の例（VB と Java 間のアーリー バインディングとレイト バインディング）を通読（およびできれば実行）してください。

JVM をプロセス外部で実行する（しかし COM クライアントがネイティブ モードを使用して Java オブジェクトにアクセスする）場合、「regjvm」コマンドを使用してネイティブとしてそれを登録する必要があります。regjvm コマンドはさまざまなレジストリ エントリを設定して WebLogic jCOM の COM と Java 間のメカニズムを容易にします。

main で、*com.bea.jcom.Jvm.register("MyJvm")* を呼び出すことによって 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。

次に、JVM を起動します。

```
java -DJCOM_NATIVE_MODE YourMain
```

これで、VB 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 aHashtable = GetObject("MyJvm:java.util.Hashtable")
```

「MyJvm」は、JVM を識別する任意の文字列です。

これは、JVM が既に実行されている場合にのみ機能します。regjvm コマンドの追加パラメータは、実行されていない JVM を起動するためのコマンドを指定できます。

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)。

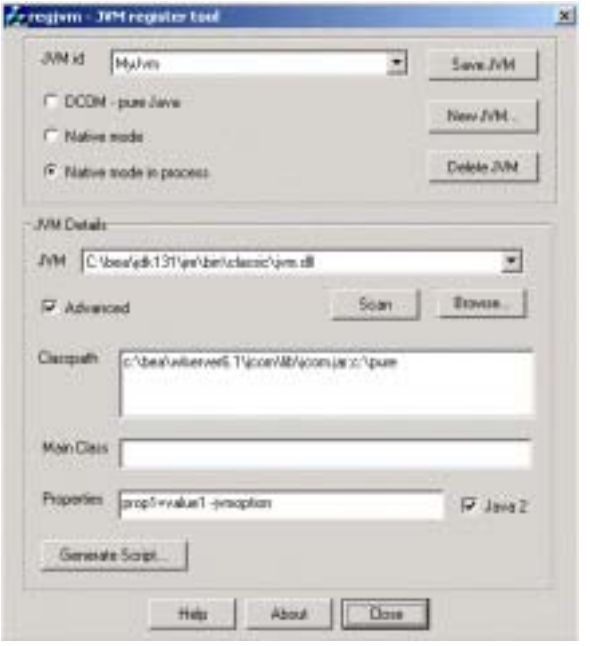
また、独自のインスタンシエータ（com.bea.jcom.Jvm.register(...) への追加パラメータ）を JVM に関連付けることによって、COM クライアントの代わりに Java オブジェクトのインスタンス化を制御できます。これはオブジェクト ファクトリの一種で、WebLogic jCOM ドキュメントの標準 COM->EJB サンプルのほとんどで使用されます。

ネイティブ モードでの COM クライアントの実行（JVM インプロセス）

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。

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（`d:\bea\jdk131` にインストールされている）を使用し、WebLogic jCOM が `d:\bea\wlserver6.1\jcom` 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、VB から次のことを行うことができます。

```
MessageBox GetObject("MyJvm:java.util.Hashtable")
```

JVM 用のプロパティを指定する場合、regjvm コマンド ラインの最後に `name=value`（スペースで区切る）という形式で追加します（`-Dname=value` は使用しないでください）。たとえば、問題が発生した場合、`JCOM_LOG_LEVEL=3` と `JCOM_LOG_FILE=c:\temp\jcom.log` を「regjvm」コマンドの最後に追加してロギングを有効にします。上で説明したメカニズム（com.bea.jcom.Log.logImmediately(...) の呼び出し）は、Java クラスに実行の機会があった場合にのみ有効です。問題はその前に存在する場合があります。

E_NOMONIKER エラーを取得した場合、WebLogic jCOM Moniker（jintmk.dll）のロギングを有効にし、必要な JVM クラス、WebLogic jCOM ランタイム（jcom.jar）、および必要なクラスを使用できるようにするためのクラスパスを調べてください。

サポートされない機能と確認済みの問題

次の機能は、ネイティブ モードでサポートされていないか、またはバグです。これらを重要事項として追加または解決する場合は、BEA にお問い合わせください。

1. パラメータとしての構造

6　セキュリティ / 認証

この章は、以下の 4 つの節に分かれています。

- 最初の節では、WebLogic Server によってホストされている EJB を呼び出すクライアントを JNDI 認証を利用して認証する方法について説明します。
- 2 番目の節では、COM コンポーネントのメソッドを呼び出す Java クライアントを認証する方法について説明します。
- 3 番目の節では、Java コードを呼び出す COM クライアントのアイデンティティを識別する方法について説明します。
- 4 番目の節では、pure Java クライアントからの NT ドメイン / ユーザ / パスワードを認証する方法について説明します。
- 最後の節では、新しい接続が COM クライアントから来たときに通知されるようにする方法について見ていきます。

WebLogic Server での JNDI を使用した認証

クライアント アプリケーションがセキュリティ資格（ユーザ名とパスワードなど）を提供し、JNDI 認証を使用して WLS でアクセスを認証するためにそれらが使用される場合、次のことを行う必要があります。

1. jCOM ブリッジで JNDI 認証を有効にします。

   jCOM ブリッジは、COM クライアント（Excel など）が WebLogic Server でホストされる EJB にアクセスすることを許可します。JNDI 認証を提供するには、`InitialContext` を呼び出す前に適切なプロパティを追加する必要があります。

```
Context.SECURITY_AUTHENTICATION
Context.SECURITY_PRINCIPAL
Context.SECURITY_CREDENTIALS
```

WebLogic jCOM サンプルに付属の jCOM ブリッジ（*c:\bea\wlserver6.1\jcom\samples\JCOMBridge.java* など）を見てください。上の 3 行のコードのコメントを `login` メソッドで解除します。

2. クライアント アプリケーションでユーザ名とパスワードを使用または取得します。

   VBA の場合は次のようにします。

```
'jCOM ブリッジにアクセスする (jCOM ファイルにはアクセスしない )
Set objBridge = GetObject("objref:...:")

'JNDI 認証用のユーザ名とパスワードを設定する
Dim bridge As Object
Set bridge = objBridge.get("ejb:JCOMBridge")
bridge.login "newUsername", "newPassword"

'JNDI を介して EJB AccountHome オブジェクトをバインド
Set mobjHome = objBridge.get("ejb:beanManaged.AccountHome")
```

   WebLogic Server での JNDI 認証については、次の URL を参照してください。
   http://edocs.beasys.co.jp/e-docs/wls61/security/prog.html#1024165

COM コンポーネントにアクセスする Java クライアントの認証

WebLogic jCOM では、認証をまったく使用しないか、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。

Windows での Java コードの実行

Windows で実行し、WebLogic jCOM で現在のアイデンティティを自動的に取得する場合、WebLogic jCOM *bin* ディレクトリを PATH 環境変数に追加します。

Windows 以外のプラットフォームでの Java コードの実行

Windows で実行しない場合、または WebLogic jCOM でネイティブ コードを使用して現在のアイデンティティを取得しない場合、プログラムの開始時に AuthInfo.setDefault(...) を呼び出して、COM コンポーネントの作成および使用時にプロセス ワイド ベースで使用する認証を設定します。

このプロセスワイドのデフォルトは、AuthInfo.setThreadDefault(...) を使用して上書きできます。これは、**現在のスレッド**に使用される認証を確立します。現在のスレッド用の認証を削除するには、AuthInfo.setThreadDefault(null) を呼び出します。

AuthInfo.setDefault(...) を呼び出して JVM ワイドベースで使用される認証を確立することをお勧めします。これにより、WebLogic jCOM デーモン スレッドは認証された通信を実行できるようになります（ガベージ コレクションされていない COM オブジェクト参照を解放する場合など）。

WebLogic jCOM は、現時点では**認証**のみをサポートしており、**暗号化**はサポートしていません。暗号化を追加したい場合は、BEA にお問い合わせください。

認証アクセスの実行の検証

Windows NT では、認証アクセスが実行されていることを認証できます。

`User Manager for Domains` ツールを使用します。そのためには、[スタート｜プログラミング｜管理ツール｜ドメイン ユーザ マネージャ] を選択し、次に [原則｜監査] を選択して [監査の原則] ダイアログを表示します。

ここから、ログインとログオフの監査を有効にできます。

監査を有効にしたら、イベント ビューアを起動します。そのためには、[スタート｜プログラミング｜管理ツール｜イベント ビューア] を選択し、[ログ｜セキュリティ] を選択してセキュリティ ログを表示します。

WebLogic jCOM は、実行されているホスト名を送信します。このホスト名には (jCOM) という文字列が付けられ、ワークステーション名としてログに記録されます。

Java コードを呼び出す COM クライアントのアイデンティティの識別

COM クライアントが WebLogic jCOM の DCOM エンジンを介して Java オブジェクトのメソッドを呼び出す場合、次のものを呼び出すことができます。

- com.bea.jcom.AuthInfo.getCallerDomain() を呼び出して、呼び出し側の NT ドメインを検索します（確認可能な場合）。これは文字列です。
- com.bea.jcom.AuthInfo.getCallerUser() を呼び出して、呼び出し側の NT ドメインを検索します（確認可能な場合）。これは文字列です。
- com.bea.jcom.isCallerAuthenticated() を呼び出して、WebLogic jCOM が NT Challenge-Response プロトコルを使用して呼び出し側のアイデンティティを検証できたかどうかを調べます。これを行うためには、JCOM_NTAUTH_HOST プロパティを、ユーザを認証できる NT マシンの IP 名に設定する必要があります。

pure Java ソフトウェアからの NT ドメイン / ユーザ / パスワードの認証

UNIX マシン（または任意の場所）で動作する Java プログラムからのドメイン / ユーザ / パスワードを認証するには、静的な **com.bea.jcom.NTLMAuthenticate.validate(...)** メソッドを使用します。

これはこのメソッドに関連する Javadoc です。

```
public static void validate(String pdcTcpHost,
String domain,
String user,
String password) throws IOException
```

NT ドメイン / ユーザ / パスワードを認証します。Java をサポートする任意のマシンで動作し、ネイティブ コードを必要としません（jcom.jar ランタイムのみ）。パスワードはネットワーク上で転送されません（WebLogic jCOM が NT Challenge-Response メカニズムを実装します）。ドメイン / ユーザ / パスワードが有効な場合は単にこのメソッドが返され、それ以外の場合はセキュリティ例外が送出されます。

パラメータ :
pdcTcpHost ― WebLogic jCOM が認証を実行できる NT マシンの IP 名
domain ― ユーザの NT ドメイン名
user ― ユーザの NT ユーザ名
password ― ユーザのパスワード

例外 : SecurityException
ドメイン / ユーザ / パスワードが正確でない場合

例外 : IOException
認証の対象となる NT マシンとの対話に問題が存在した場合

注意 : このメソッドは WebLogic jCOM pure Java-COM ブリッジとはまったく関係がなく、WebLogic jCOM を使用して Java から COM オブジェクトに（またはその反対に）アクセスするときにこのメソッドを呼び出す必要はありません。

DCOM エンジンの一部として pure Java で NT Challenge-Response メカニズムを実装しているので、このメソッドを公開するのはささいなことですが、役に立つ場合もあります。

COM クライアントからの新しい接続のリスン

このメカニズムは、ネイティブ モードで実行中のときには実装されません。

WebLogic jCOM の ConnectionListener メカニズムにより、COM クライアントからの新しい COM 接続がオープンまたはクローズするときに通知が行われるように要求し、受け取った接続を拒否できます。

com.bea.jcom.ConnectionListener インタフェースを実装するクラスを作成し、com.bea.jcom.Cleaner.addConnectionListener(...) を呼び出すことによってそのクラスのインスタンスを登録します。

7 サポートされる COM データ型とそれらに相当する Java データ型

WebLogic jCOM は、すべての COM Automation 型をサポートしています。BEA は、新しい Java クラスを導入せずに COM 型にアクセスすることを方針としました。このため、COM Variant は Object（java.lang.Long など）にマップされ、配列を含む Variant は Java 配列にマップされます。たとえば、使用する必要がある com.bea.jcom.Variant クラスは見つかりません。

WebLogic jCOM は、Java コレクション型に Visual Basic Collection としてアクセスすることをサポートしています。

この章では、以下のことを説明します。

- Java オブジェクトへの Visual Basic Collection としてのアクセス
- COM IDL 型の Java、および VC++ へのマップ
- 異なる型を含む COM Variant の Java へのマップ
- COM から Java へのレイトバインド アクセス中に使用される型変換
- Java から COM VariantEnums へのアクセス

Java オブジェクトへの Visual Basic Collection としてのアクセス

Visual Basic には、*Collection* というクラスがあらかじめ用意されています。WebLogic jCOM は、java.util.Vector および java.util.List のインスタンスに VB Collection としてアクセスすることをサポートしています。

Visual BASIC コードの例

```
Private Sub Form_Load()
Dim c As Collection
Set c = GetObject("firstjvm:java.util.LinkedList")
c.Add "hello" ' List is now: "hello"
c.Add Now, , "hello" ' List is now: , "hello"
c.Add "Goodbye", , , "hello" ' List is now: now, "hello", "Goodbye"
c.Add "Before", , 3 ' List is now: now, "hello", "Before", "Goodbye"
c.Add "After", , , 4 ' List is now: now, "hello", "Before",
"Goodbye", "After"
For Each e In c
MsgBox e
Next
c.Remove 2 ' List is now: now, "Before", "Goodbye", "After"
c.Remove "Goodbye" ' List is now: now, "Before", "After"
MsgBox c.Item(1)
MsgBox c.Count
End Sub
```

VB	java.util.Vector へのマップ	注意
Collection.Add(*item*)	aVector.addElement(*item*)	
Collection.Add(*item*, , *before*)	aVector.insertElementAt(*before* - 1, *item*)	*before* が数字の場合。
Collection.Add(*item*, , *before*)	aVector.insertElementAt(aVector.indexOf(*before*), *item*)	*before* が数字でない場合。
Collection.Add(*item*, , , *after*)	aVector.insertElementAt(*after*, *item*)	*after* が数字でない場合。
Collection.Add(*item*, , , *after*)	aVector.insertElementAt(aVector.indexOf(*after*) + 1, *item*)	*after* が数字の場合。
Collection.Count	aVector.size()	
Collection.Item(*index*)	aVector.elementAt(*index* - 1)	*index* は数字でなければならない。
Collection.Remove(*index*)	aVector.removeElementAt(*index* - 1)	*index* が数字の場合。

Collection.Remove(*index*)	aVector.removeElement(*index*)	*index* が数字でない場合。
aCollection 中の各 x	x は aVector.elements() 列挙内の各オブジェクトの値を取る。	

VB	java.util.List へのマップ	注意
Collection.Add(*item*)	aList.target.add(*item*)	
Collection.Add(*item*, , *before*)	aList.add(*before* - 1, *item*)	*before* が数字の場合。
Collection.Add(*item*, , *before*)	aList.add(aVector.indexOf(*before*), *item*)	*before* が数字でない場合。
Collection.Add(*item*, , , *after*)	aList.add(*after*, *item*)	*after* が数字でない場合。
Collection.Add(*item*, , , *after*)	aList.add(aVector.indexOf(*after*) + 1, *item*)	*after* が数字の場合。
Collection.Count	aList.size()	
Collection.Item(*index*)	aList.get(*index* - 1)	*index* は数字でなければならない。
Collection.Remove(*index*)	aList.remove(*index* - 1)	*index* が数字の場合。
Collection.Remove(*index*)	aList.remove(*index*)	*index* が数字でない場合。
aCollection 中の各 x	x は aList.listIterator() イテレータ内の各オブジェクトの値を取る。	

COM IDL 型の Java、VB、および VC++ へのマップ

次の表に、各 IDL 型と、Java、Visual BASIC、および Visual C++ からのその使い方を示します。

表 7-1

IDL	Java	Visual BASIC	Visual C++
[in] VARIANT_BOOL	**boolean**	ByVal boolean	VARIANT_BOOL
[in, out] または [out] VARIANT_BOOL*	**boolean[]** 単一要素配列	boolean	VARIANT_BOOL*
[in] unsigned char	**byte**	ByVal byte	unsigned char
[in, out] または [out] unsigned char*	**byte[]** 単一要素配列	byte	unsigned char*
[in] double	**double**	ByVal Double	double
[in, out] または [out] double*	**double[]** 単一要素配列	double	double*
[in] float	**float**	ByVal Single	float
[in, out] または [out] float*	**float[]** 単一要素配列	Single	float*
[in] long	**int** IDL long は 32 ビット	ByVal Long>	long>
[in, out] または [out] long*	**int[]** 単一要素配列	Long>	long*
[in] short	**short**	ByVal Integer>	short>
[in, out] または [out] short*	**short[]** 単一要素配列	Integer	short*
[in] BSTR	**java.lang.String**	ByVal Single	BSTR
[in, out] または [out] BSTR*	**java.lang.String[]** 単一要素配列	String	BSTR*

表 7-1 （続き）

IDL	Java	Visual BASIC	Visual C++
[in] CY	**long** CY は固定小数点、64 ビット	ByVal Currency	CURRENCY
[in, out] または [out] CY*	**long[]** 単一要素配列	Currency	CURRENCY*
[in] DATE	**java.util.Date**	ByVal Date	DATE
[in, out] または [out] DATE*	**java.util.Date[]**	Date	DATE*
[in] IDispatch*	**java.lang.Object**	ByVal Object	IDispatch*
[in, out] または [out] IDispatch**	**java.lang.Object[]** 単一要素配列	Object	IDispatch**
[in] ISomeInterface*	**ISomeInterface** 生成される Java インタフェース	ByVal ISomeInterface	ISomeInterface*
[in, out] または [out] ISomeInterface**	**ISomeInterface[]** 単一要素配列	ISomeInterface	ISomeInterface**
[in] SomeClass*	**SomeClass** 生成される Java クラス	ByVal SomeClass	SomeClass*
[in, out] または [out] SomeClass**	**SomeClass[]** 単一要素配列	SomeClass	SomeClass**
[in] IUnknown*	**java.lang.Object**	特定のクラス / インタフェース	IUnknown*
[in] Variant	**java.lang.Object**	ByVal Variant	VARIANT
[in, out] または [out] Variant*	**java.lang.Object[]** 単一要素配列	Variant	VARIANT*
[in] SAFEARRAY(unsigned char)	**byte[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(unsigned char)*	**byte[][]**	VB ではサポートされない	SAFEARRAY**

表 7-1 （続き）

IDL	Java	Visual BASIC	Visual C++
[in] SAFEARRAY(VARIANT_BOOL)	**boolean[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(VARIANT_BOOL)*	**boolean[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(short)	**short[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(short)*	**short[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(long)	**int[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(long)*	**int[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(float)	**float[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(float)*	**float[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(double)	**double[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(double)*	**double[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(CURRENCY)	**long[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(CURRENCY)*	**long[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(DATE)	**java.util.Date[]**	VB ではサポートされない	SAFEARRAY*

表 7-1 （続き）

IDL	Java	Visual BASIC	Visual C++
[in, out] または [out] SAFEARRAY(DATE)*	**java.util.Date[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(BSTR)	**String[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(BSTR)*	**String[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(LPDISPATCH)	**Object[]**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(LPDISPATCH)*	**Object[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(VARIANT)	**Object**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(VARIANT)*	**Object[][]**	VB ではサポートされない	SAFEARRAY**
[in] SAFEARRAY(LPUNKNOWN)	**Object**	VB ではサポートされない	SAFEARRAY*
[in, out] または [out] SAFEARRAY(LPUNKNOWN)*	**Object[][]**	VB ではサポートされない	SAFEARRAY**

異なる型を含む COM Variant の Java へのマップ

次の表に、Variant と Java 型のマップを示します。

以下を含む Variant	Java 型へのマップ
VARIANT_BOOL	**java.lang.Boolean**
VARIANT_BOOL*	**java.lang.Boolean[]** 単一要素配列
unsigned char	**java.lang.Byte**
unsigned char*	**java.lang.Byte[]** 単一要素配列
double	**java.lang.Double**
double*	**java.lang.Double[]** 単一要素配列
float	**java.lang.Float**
float*	**java.lang.Float[]** 単一要素配列
long	**java.lang.Integer** IDL long は 32 ビット
long*	**java.lang.Integer[]** 単一要素配列
short	**java.lang.Short**
short*	**java.lang.Short[]** 単一要素配列
BSTR	**java.lang.String**
BSTR*	**java.lang.String[]** 単一要素配列
CY	**java.lang.Long**
CY*	**java.lang.Long[]** 単一要素配列
Decimal	**java.math.BigDecimal**

以下を含む Variant	Java 型へのマップ
Decimal*	**java.math.BigDecimal[]** 単一要素配列
DATE	**java.util.Date**
DATE*	**java.util.Date[]**
SCODE	**java.lang.Long**
SCODE*	**java.lang.Long[]** 単一要素配列
IDispatch*	**java.lang.Object**
IUnknown*	**java.lang.Object**
VARIANT_BOOL の単一次元配列	**boolean[]**
unsigned char の 1 次元配列	**byte[]**
double の 1 次元配列	**double[]**
float の 1 次元配列	**float[]**
long の 1 次元配列	**int[]** IDL long は 32 ビット </TD
short の 1 次元配列	**short[]**
BSTR の 1 次元配列	**java.lang.String[]**
CY の 1 次元配列	**long[]**
DATE の 1 次元配列	**java.util.Date[]**
SCODE の 1 次元配列	**long[]**
IDispatch* の 1 次元配列	**java.lang.Object[]**
IUnknown* の 1 次元配列	**java.lang.Object[]**
Variant の 1 次元配列	**java.lang.Object[]**
Variant の 2 次元配列	**java.lang.Object[][]**

com.bea.jcom.EmptyVariant.TYPE と com.bea.jcom.NullVariant.TYPE を受け渡すことで、VT_EMPTY または VT_NULL 型の Variant を受け渡すことができます。

COM から Java へのレイトバインド アクセス中に使用される型変換

以下の表に、左欄の VB 型の VB 値「vb」を上欄の Java 型に変換するために使用される Java 式を示します。***** が付けられた変換は、*Type Mismatch Error* を生成する場合があります。

Java boolean、byte、short、および int

Java（右） VB（下）	boolean	byte	short	int
Boolean	自然	vb ? 1 : 0	vb ? 1 : 0	vb ? 1 : 0
Byte（0..255）	vb != 0	new Integer(vb).byteValue()	new Integer(vb).shortValue()	new Integer(vb).intValue()
Integer（16 ビット）	vb != 0	new Short(vb).byteValue()	自然	new Short(vb).intValue()
Long（32 ビット）	vb != 0	new Integer(vb).byteValue()	new Integer(vb).shortValue()	自然
Single	vb != 0.0F	new Float(vb).byteValue()	new Float(vb).shortValue()	new Float(vb).intValue()
Double	vb != 0.0	new Double(vb).byteValue()	new Double(vb).shortValue()	new Double(vb).intValue()

Java（右） VB（下）	boolean	byte	short	int
Currency（64 ビット）	vb != 0	new Long(vb).byteValue()	new Long(vb).shortValue()	new Long(vb).intValue()
String	new Boolean(vb).booleanValue()	new Long(vb).byteValue()	new Long(vb).shortValue()	new Long(vb).intValue()*
Date	型不一致エラー	型不一致エラー	型不一致エラー	型不一致エラー
Object	型不一致エラー	型不一致エラー	型不一致エラー	型不一致エラー
Variants	内容に応じて上記のとおり			

Java long、char、float、および double

VB/Java	long	char	float	double
Boolean	vb ? 1 : 0	vb ?(char)1 :(char)0	vb ?1.0F :0.0F	vb ? 1.0 : 0.0
Byte（0..255）	new Integer(vb).longValue()	(char)(new Integer(vb).intValue())	new Integer(vb).floatValue()	new Integer(vb).doubleValue()
Integer（16 ビット）	new Short(vb).longValue()	(char)(new Short(vb).intValue())	new Short(vb).floatValue()	new Short(vb).doubleValue()
Long (32 bit)	new Integer(vb).longValue()	(char)(new Integer(vb).byteValue())	new Integer(vb).floatValue()	new Integer(vb).doubleValue()
Single	new Float(vb).longValue()	(char)(new Float(vb).intValue())	自然	new Float(vb).doubleValue()
Double	new Double(vb).longValue()	(char)(new Double(vb).intValue())	new Double(vb).floatValue()	自然
Currency（64 ビット）	自然	(char)(new Long(vb).intValue())	new Long(vb).floatValue()	new Long(vb).doubleValue()

VB/Java	long	char	float	double
String	new Long(vb).longValue()	(char)(new Long(vb).intValue())	new Long(vb).floatValue()	new Long(vb).doubleValue()
Date	型不一致エラー	型不一致エラー	型不一致エラー	型不一致エラー
Object	エラー	エラー	エラー	エラー
Variants	内容に応じて上記のとおり			

Java String、Date、および Object

VB/Java	String	Date	Object
Boolean	vb + ""	エラー	new Boolean(vb)
Byte（0..255）	vb + ""	エラー	new Byte(vb)
Integer（16 ビット）	vb + ""	エラー	new Short(vb)
Long(32 bit)	vb + ""	エラー	new Integer(vb)
Single	vb + ""	エラー	new Float(vb)
Double	vb + ""	エラー	new Double(vb)
Currency	vb + ""	エラー	new Long(vb)
String	自然	new Date(vb)	vb (as String_
Date	vb.toString()	自然	vb (as Date)
Object	vb.toString	エラー	vb
Variant	内容に応じて上記のとおり		

配列

Java メソッドが配列のパラメータを持つ場合、そのメソッドに VB パラメータを値で渡すときにエラーが生成されます。値を参照で渡し、型が正確に一致する場合（下表を参照）、Java メソッドは単一要素配列を受け取ります。この単一要素を修正して、対応する VB パラメータを変更できます。配列を渡し、配列の内容の型が正確に一致する場合、Java メソッドは配列を受け取ります。この配列の要素を修正して対応する VB パラメータを変更できます。

Java 型	参照渡しの VB の一致
boolean[]	参照によって渡される Boolean
byte[]	参照によって渡される Byte
short[]	参照によって渡される Short
int[]	参照によって渡される Long
long[]	参照によって渡される Currency
float[]	参照によって渡される Single
double[]	参照によって渡される Double
String[]	参照によって渡される String
java.util.Date[]	参照によって渡される Date

配列

Java 型	一致する VB 型のパラメータ
boolean[]	Dim anArray(n) as Boolean
byte[]	Dim anArray(n) as Byte
short[]	Dim anArray(n) as Short
int[]	Dim anArray(n) as Long

Java 型	一致する VB 型のパラメータ
long[]	Dim anArray(n) as Currency
float[]	Dim anArray(n) as Single
double[]	Dim anArray(n) as Double
String[]	Dim anArray(n) as String
java.util.Date[]	Dim anArray(n) as Date
Object[]	Dim anArray(n) as String
Object[]	Dim anArray(n) as Date
Object[]	Dim anArray(n) as Object
Object[]	Dim anArray(n) as Variant

Java から COM VariantEnums へのアクセス

一部の COM メソッドは、COM VariantEnumeration 型を返します。WebLogic jCOM の java2com ツールは、返された型を標準 Java java.lang.Enumeration に自動的に変換します。しかし、完全な一致は存在しません。これは、COM Enumeration には 「hasMoreElements()」呼び出しに相当するものがないからです。このため、「NoSuchElementException」を取得するまで「nextElement」を繰り返す必要があります。

ここで、回避策があります。**JCOM_PREFETCH_ENUMS** Java プロパティを設定した場合、WebLogic jCOM は背後にある次の要素をプリフェッチします。このため、「hasMoreElements()」メソッドを適切に実行できます。

8　スレッディング

Java から COM へ

WebLogic jCOM は、Java から COM オブジェクトへの複数の同時要求を処理します。

WebLogic jCOM ランタイム DCOM エンジンを使用すると、複数の同時要求を同じ COM オブジェクト、および異なる COM オブジェクトに対して行うことができます。また、WebLogic jCOM は DCOM と対話し、DCOM は DCE RPC の上層にあるので、WebLogic jCOM は大規模なメソッド呼び出しを複数の RPC PDU に分割し、分割された応答を再び結合します。

リモート COM オブジェクトが複数の同時要求を適切に処理するかどうかは、すべてそのスレッディング モデルと実装によって決まります。

COM から Java へ

WebLogic jCOM は、COM オブジェクトからの要求を処理するためにスレッドのプールを保持します。デフォルトでは、WebLogic jCOM は最大 20 のスレッドで要求を処理しますが、次のように、**JCOM_MAX_REQUEST_HANDLERS** プロパティを自由に設定することで、最大スレッド数を変更できます。

```
java -DJCOM_MAX_REQUEST_HANDLERS=50 YourMainProgram
```

WebLogic jCOM は、必要に応じてこうしたスレッドを作成します。つまり、初めから 20 のスレッドが即座に作成されるわけではありません。最初のスレッドは、最初のリモート要求が届いたときに作成されます。2 番目の要求が届き、最初のスレッドが要求を処理している場合、WebLogic jCOM は 2 番目のスレッドを作成して 2 番目の要求を処理します。このようにして、最大 20 のスレッド（デフォルト）が作成されます。

WebLogic jCOM のスレッド割り当てメカニズムを確認したい場合、内部ロギングを有効に（JCOM_LOG_LEVEL プロパティを最大値の 3 に設定）して WebLogic jCOM を実行します。次の Visual BASIC コードでは、Java オブジェクトへのコールバックが行われます（p1 は Java オブジェクト）。

```
Public Sub method1(ByVal p1 As Object)
p1.aMethod
End Sub
```

以下は、何が起こったかを示すログの抜粋です。*IDispatch::GetIDsOfNames* は、リフレクションによって WebLogic jCOM で内部的に処理されます。

```
13:05:20+: ObjectExporter0 received an unfragmented request with
call ID 1
13:05:20+: Maximum number of request handler threads is set to 20
13:05:20+: There are 0 request handler threads, of which 0 are
currently busy.
13:05:20+: Creating a new request handler thread.
13:05:20 : IDispatch::GetIDsOfNames request on ExcepDemo@1f230b for
AMETHOD. Returning memid 9
13:05:20+: ObjectExporter0 sending 44 bytes
13:05:20+: ObjectExporter0 read 192 bytes
13:05:20+: ObjectExporter0 received an unfragmented request with
call ID 2
13:05:20+: There are 1 request handler threads, of which 0 are
currently busy.
13:05:20 : IDispatch::Invoke request received for public void
ExcepDemo.aMethod() on ExcepDemo@1f230b
```

Java オブジェクト内の同じメソッドが COM オブジェクトによって同時に複数呼び出されないようにする場合は、標準 Java *synchronized* キーワードを使用します。これにより、別のオブジェクトが既にメソッドを呼び出している場合、同じメソッドの呼び出しがブロックされます。

9 Java からの COM オブジェクトの使い方

この章では、COM オブジェクトのインスタンスを作成し、WebLogic jCOM を使ってそれを Java から使用する方法について説明します。また、COM オブジェクトへのメソッド呼び出しから返される COM オブジェクトの新しい参照についても説明します。

- com2java によって COM クラスから生成される Java クラス
- com2java によって COM インタフェースから生成される Java インタフェースおよびクラス

この章を読む前に、com2java ツールのマニュアルを通読しておいてください。

com2java によって COM クラスから生成される Java クラス

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 クラスが生成されます。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。

デフォルト コンストラクタ

デフォルト コンストラクタを使用すると、デフォルト認証を使用するか（設定されている場合）、認証を使用せずに（デフォルトが設定されていない場合）ローカル マシン上に COM オブジェクトのインスタンスを作成できます。

2 番目のコンストラクタ (String host)

2 番目のコンストラクタを使用すると、デフォルト認証を使用するか（設定されている場合）、認証を使用せずに（デフォルトが設定されていない場合）特定のマシン上に COM オブジェクトのインスタンスを作成できます。

3 番目のコンストラクタ (AuthInfo authInfo)

3 番目のコンストラクタを使用すると、特定の認証を使用してローカル ホスト上に COM オブジェクトのインスタンスを作成できます。

4 番目のコンストラクタ (String host, AuthInfo authInfo)

4 番目のコンストラクタを使用すると、特定の認証を使用して特定のホスト上に COM オブジェクトのインスタンスを作成できます。

最後のコンストラクタ (Object objRef)

最後のコンストラクタは、実際に新しい COM クラスのインスタンスを作成しません。その代わり、このコンストラクタを使用すると、（メソッド呼び出しから、またはプロパティもしくはイベントを介して）別の COM クラスから返された COM クラスの参照にアクセスできます。

メソッドまたはプロパティが COM クラスの参照を返した場合、com2java ツールは適切な Java クラスを返すメソッド シグネチャを自動的に生成します（返された COM クラスが同じ型ライブラリに定義されている場合）。

com2java によって COM インタフェースから生成される Java インタフェースおよびクラス

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 return value. An reference to a Workbooks
* @exception java.io.IOException If there are communications
problems.
* @exception com.bea.jcom.AutomationException If the remote server
throws an exception.
*/
```

```
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 return value. An reference to a Workbooks
* @exception java.io.IOException If there are communications
problems.
* @exception com.bea.jcom.AutomationException If the remote
server throws an exception.
*/
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、この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 return value. An reference to a Workbooks
* @exception java.io.IOException If there are communications
problems.
* @exception com.bea.jcom.AutomationException If the remote server
throws an exception.
*/
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);
```

10 jCOM.jar の Java プロパティ

次の表に、jcom.jar に対して設定可能なプロパティを示します。

jCOM プロパティ	説明
ENABLE_TCP_NODELAY	TCP/IP 接続は DCOM モードで動作するときに TCP_NODELAY を設定する。実際に行われる処理については、標準 Java API マニュアルを参照。
JCOM_DCOM_PORT	jCOM が DCOM 要求を受信するために使用する TCP/IP ポート（実際には 2 つのポートを使用）。
JCOM_COINIT_VALUE	WebLogic jCOM が CoInitializeEx を使って新しいスレッド用の COM を初期化するときに使用される COM モードを設定する。
JCOM_INCOMING_CONNECTION_TIMEOUT	指定されたミリ秒の間使用されていない受信時接続が切断される。
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。

jCOM プロパティ	説明
com.bea.jcom.server	一般に、DCOM モードで動作中、WebLogic jCOM ランタイムは COM に渡す Java オブジェクトの DCOM オブジェクト参照にローカルマシンの IP アドレスをすべて組み込む。このプロパティを設定すると、それらの IP アドレスを特定のアドレスに限定できる（RMI_LOCAL_HOST も使用するが、com.bea.jcom. サーバはこれを上書きする）。
JCOM_MAX_REQUEST_HANDLERS	WebLogic jCOM マニュアルのマルチスレッディングの節を参照。
JCOM_NATIVE_MODE	WebLogic jCOM ランタイムに、ネイティブ モードで動作するよう指示する（このプロパティに関連する値は無視される）。

jCOM プロパティ	説明
JCOM_NOGIT	WebLogic jCOM ランタイムに、ネイティブ モードで動作中に COM グローバルインタフェース テーブルを使用しないよう指示する。このプロパティを設定すると、WebLogic jCOM は GIT クッキーの代わりにオブジェクト参照ポインタを直接格納する。このため、スレッド間でオブジェクト参照を受け渡す際に問題が発生する場合がある。
JCOM_NTAUTH_HOST	このプロパティを、WebLogic jCOM が DCOM 呼び出しを認証するために対話する NT サーバ マシンの名前に設定する。WebLogic jCOM リファレンスのセキュリティの節を参照。
JCOM_LOCAL_PORT_START	WebLogic jCOM ランタイムに、COM サーバへの DCOM 接続をオープンするときに使用するローカル ポートを指示する。指定されたポートが既に使用されている場合、JCOM_LOCAL_PORT_END に達するまで次のポートの使用を試みる。

jCOM プロパティ	説明
JCOM_LOCAL_PORT_END	WebLogic jCOM ランタイムに、COM サーバへの DCOM 接続をオープンするときに使用するローカル ポートの上限を指示する（JCOM_LOCAL_PORT_START を参照）。
JCOM_PROXY_PACKAGE	java2com によって生成されるプロキシが（ここで説明されていないメカニズムによって）特定のパッケージに置かれたときに使用される。
JCOM_SKIP_CLOSE	WebLogic jCOM は、DCOM モードで動作中にピアがソケットをクローズできるようにする。
JCOM_WS_NAME	AuthInfo.setDefault(...) または AuthInfo.setThreadDefault(...) を使用して、クライアントが DCOM モードで動作中に NT マシンがクライアントのワークステーション名として認識する名前を設定する。

11　トラブルシューティング

以下の節では、WebLogic jCOM の一般的な問題と、それらを解決するための有益なヒントについて説明します。

- WebLogic jCOM のパフォーマンスの向上
- DOS エラー
- Java エラー
- Visual BASIC エラー
- Visual C++ エラー
- 特定の環境で動作しているときの DCOM のセキュリティ問題
- java2com によって生成される IID*.java ラッパーがコンパイルされない場合
- 掲載されていないエラーが発生した場合

WebLogic jCOM のパフォーマンスの向上

ロギング短縮ランタイムの使用

パフォーマンスを向上する最も簡単な方法は、ロギング短縮 WebLogic jCOM ランタイムを使用することです。*jcom.jar* を CLASSPATH に指定する代わりに、同じディレクトリにある *jcom_reduced_logging.jar* を使用します。

このランタイムを使用すると、WebLogic jCOM ログ ファイルの生成を禁止して問題を解決でき、環境によってはパフォーマンスが著しく向上します。

ネイティブ モードでの実行

WebLogic jCOM の DCOM モード（デフォルト）で実行し、JVM が MS Windows で動作している場合、ネイティブ モードに切り替えることを検討します。これにより、WebLogic jCOM ランタイムは DCOM の代わりにネイティブ コードを使用して Java と COM 間で対話を行います。コードを変更する必要はまったくありません。ネイティブ モードへの切り替えは、ランタイム プロパティを定義することで行います。

詳細については、WebLogic jCOM リファレンスのネイティブ モードの節を参照してください。

DOS エラー

発生したエラー

- **DOS: The *java* command is not recognised**

この場合、PATH 環境変数に JDK *bin* ディレクトリが指定されていません。

発生したエラー

- **DOS (while compiling using the *java* command): java.lang.OutOfMemoryError**

この場合、コンパイラにより多くのメモリを使用するよう指示します。以下を使用してコンパイルします。
java -Jmx64m -J-ms64m YourProgram.java

発生したエラー

- **DOS: The name specified is not recognized as an internal or external command, operable program or batch file.**

regtlb または **regjvm** を使おうとした場合。

この場合、PATH 環境変数に WebLogic jCOM の *bin* ディレクトリが指定されていません。

Java エラー

発生したエラー

- **java Can't find class com/bea/java2com/Main**
- **java Package com.bea.jcom not found in import.**
- **java java.lang.NoClassDefFoundError: com/bea/jintact/Helper**

この場合、CLASSPATH 環境変数に WebLogic jCOM ランタイム（*jcom.jar*）が指定されていません。

発生したエラー

- **java AutomationException: 0x80080005 - Server execution failed. Note that Windows 95 does not support automatic launch of a server, it must be running already**

Windows NT で実行している場合、NT イベント ログ（[**スタート｜プログラム｜管理ツール｜イベント ビューア**] の [**ログ｜システム**]）をチェックします。これにより、起動時の障害の理由を発見できる場合があります。

このエラーの一般的な原因は、サーバのレジストリ内の PATH が不正確であることです。

また、いくつかの NT のバグによってこのエラーが発生する場合もあります。http://support.microsoft.com/support/kb/articles/q185/1/26.asp と http://support.microsoft.com/support/kb/articles/q158/5/08.asp を参照してください。

発生したエラー

- **java AutomationException: 0x80010001 - Call was rejected by callee. in 'Invoke'**

このエラーは、Excel の実行中に発生する場合があります。これは、Excel の制限であると見なしています。Microsoft サポートの回答の 1 つは次のとおりです。「このエラーは通常サーバ アプリケーションがビジー状態でクライアントに応答できないときに発生します」。

com2java ツールの [Options] ダイアログ ボックスには、このエラーの発生時に WebLogic jCOM コードが再試行するオプションが用意されています。このオプションを選択して、プロキシを再生成します。

発生したエラー

- **java AutomationException: 0x80020003 - Member not found in 'IDispatch::invoke'**

「ocxhost」を使用しており、ActiveX コントロールをロードしようとしている場合、指定している prog ID が不正確か、またはコントロールが正確にマシンに登録されていません。

「checkconfig」コマンドを使用して、正確な prog ID を見つけます。

```
checkconfig /typelib  config.log
```

生成された config.log ファイルを参照すると、使用する prog ID を確認できます。

発生したエラー

- **java.lang.ClassNotFoundException**

CLASSPATH が正確に定義されているかどうかを確認します。クラスを異なるクラス ローダにロードするサーブレット環境またはその他の環境を使用する場合、生成されるプロキシと jcom.jar ファイルが同じクラス ローダによってロードされることを確認してください。

発生したエラー

- **java.lang.UnsatisfiedLinkError: getNegociateMessage**

jcom.jar ファイルがシステム クラス ローダによってロードされていることをチェックします。これは、ネイティブ認証を使おうとしているが、システム クラス ローダによってロードされるクラスだけが JNI を使用できるので、JNI が適切に動作しないことを意味しています。システム クラス ローダを使用するには、jcom.jar を CLASSPATH 環境変数にシステム レベルで指定します。前述のトラブルシューティングも参照してください。

発生したエラー

- **java AutomationException: some other error code -**

エラー コードに関連するエラー メッセージが存在しない場合、AutomationException のケースでは、Microsoft Developer Network マニュアルでエラー コードを参照できます。また、Visual C++ がインストールされている場合は、winerr.h ファイルまたはエラー ルックアップ ユーティリティが役立ちます。

Visual BASIC エラー

発生したエラー

- **VB: Run-time error '-2147221020 (800401e3)': Automation Error Invalid Syntax**

このエラーは、以下を使用して Java オブジェクトをインスタンス化しようとしたときに発生します。

```
Set o = GetObject("SomeJvmId:SomeJavaClass")
```

ここで *SomeJvmId* は、*regjvm* コマンドでマシンに登録されていません。

発生したエラー

- **VB: Run-time error '429': ActiveX component can't create object**

このエラーが発生するのは、JVM が動作していないか、正確に登録されていないか、または Java クラスをインスタンス化できなかった場合です。以下を使用してロギングを有効にして JVM を起動します。

```
java -DJCOM_DCOM_PORT=??? -DJCOM_LOG_LEVEL=3 YourMainClassName
```

VB クライアントを実行しているときに、WebLogic jCOM ランタイムが VB クライアントから接続を受け取るかどうかを確認します。接続を受け取らない場合、以下のいずれかです。

com.bea.jcom.Jvm.register("jvmId") を呼び出すときに使用される jvmId が間違っています。

JVM の起動時に指定されるポートが間違っているか、既に使用中です。ロギングを有効にした場合、WebLogic jCOM ランタイムが使用するポートを確認できます。
929376053810 ：OXID Resolver started. Listening on port 1111
JVM の登録時に間違いを犯しています。つまり、VB クライアント マシン上で *regjvm jvmId host[port]* を呼び出したときに JvmID、ホスト、ポートのいずれかを間違って指定しています。
VB 環境は非常に古いので、GetObject への「MyJvm:MyClass」パラメータ フォーマットをサポートしていません。これを解決するには、WebLogic jCOM の「regprogid」コマンドを使用して progid を JVM/ クラスにマップします。
regprogid My.ProgId "MyJvm:MyClass"
次に、コードで GetObject("My.ProgId") を使用します。

発生したエラー

- **VB: Run-time error '-2147483644 (8000004)': Automation Error No Such interface supported**

このエラーは、COM オブジェクトにアーリー バインド アクセスし、そのオブジェクトのメソッドを呼び出すときに発生します。java2com ツールで生成された Java ラッパー ファイルをコンパイルしていないか、またはそれらが CLASSPATH に存在しないため WebLogic jCOM ランタイム（jcom.jar）によってロードできません。

JDK1.2 を使用している場合、WebLogic jCOM ランタイムを「ext」ディレクトリに置かないでください。このランタイムがそのディレクトリに存在する場合、CLASSPATH に指定されているクラス（ラッパー クラスなど）をロードできません。代わりに、このランタイムを CLASSPATH に置きます。

WebLogic jCOM ロギングを 3（完全）に設定すると、これを確認できます。
java -DJCOM_LOG_LEVEL=3 Main

Visual C++ エラー

発生したエラー

- **VC++: 0x80040154 (Class not registered)**

VB エラー 429 の場合と同じ手順に従います。

発生したエラー

- **VC++: 0x80000004**

VB エラー 8000004 の場合と同じ手順に従います。

特定の環境で動作しているときの DCOM のセキュリティ問題

Java アプリケーションを特定の環境（サーブレット /IDE）で実行する場合、コマンド ラインから正常に実行できるにもかかわらず、DCOM セキュリティ エラーが発生する場合があります。

その原因は以下のいずれかです。

1. Java コードが異なるユーザとして実行されています。ネイティブ認証は Java コードを実行するユーザを選択しましたが、そのユーザは十分な DCOM アクセス権を持っていません。
2. jcom\bin ディレクトリがシステム レベルで PATH に存在しないか、PATH の変更を有効にするためにサーブレット /IDE 環境を再起動していません。
3. WebLogic jCOM ランタイムがシステム クラス ローダによってロードされていません。ネイティブ認証は JNI を使用し、JNI 呼び出しはシステム クラス ローダによってロードされたクラスから行われなければなりません。一部の環境では、異なるクラス ローダによってロードされるクラスパスを指定できます。こうしたクラスパスにある Java ファイルは JNI を使用できません。システム クラス ローダが使用されるようにする安全な方法は、jcom.jar をクラスパス環境変数に指定するか、java.exe の実行時に

-cp/-classpath/-Djava.class.path オプションを使用することです。また、WebLogic jCOM プロキシ ファイルと jcom.jar ランタイムを同じクラス ローダによってロードする必要があることも考慮する必要があります。

前述のとおり WebLogic jCOM ロギングを有効にした場合、問題 2 および 3 ではネイティブ認証 DLL が適切にロードまたは実行できないことが分かります。また、WebLogic jCOM ロギングは、jcom.jar が実行されているクラス ローダを示します。ブートストラップがシステム クラス ローダです。

もう 1 つの解決策は、com.bea.jcom.AuthInfo.setDefault(...) を使用することです。これは pure java 認証を使用し（JNI は不要）、Windows 以外のプラットフォーム上でも動作します。

セキュリティ問題の詳細については、この章の他の項目、およびこのマニュアルのセキュリティの章を参照してください。

java2com によって生成される IID*.java ラッパーがコンパイルされない場合

次のようなエラーが発生する場合があります。

```
tlb\remote\IID6cce3097_00e5_1000_500a_bf09cf1e0000Wrapper.java:2:
IID6cce3097_00e5_1000_500a_bf09cf1e0000Wrapper should

be declared abstract; it does not define getHomeInterfaceClass()
in IID6cce3097_00e5_1000_500a_bf09cf1e0000Wrapper

public class IID6cce3097_00e5_1000_500a_bf09cf1e0000Wrapper
extends com.bea.jcom.Dispatch implements javax.ejb.EJB

MetaData    {
```

または

```
D:\pure\JCOM\TLB\Bean>javac
IID828d8ad4_00e5_1000_502a_bf09cf1e0000Wrapper.java

IID828d8ad4_00e5_1000_502a_bf09cf1e0000Wrapper.java:2:
IID828d8ad4_00e5_1000_502a_bf09cf1e0000Wrapper should be declared

abstract; it does not define toString(boolean) in
IID828d8ad4_00e5_1000_502a_bf09cf1e0000Wrapper
```

```
public class IID828d8ad4_00e5_1000_502a_bf09cf1e0000Wrapper
extends com.bea.jcom.Dispatch implements java.security

.Certificate    {
```

デフォルトでは、java2com ツールは特定のクラスとメソッドを無視して、生成されるラッパーの数を減らそうとします。具体的には、java.lang.Class などのクラス（前者のケース）、または toString() のようなメソッド（後者のケース）が無視されることがあります。実際に必要なときにどのクラスまたはメソッドが無視されるのかを特定するには、発生したエラーを調べる必要があります。

型ライブラリの登録を解除し（登録済みの場合）、生成されたラッパーを削除し、java2com を再実行し、[Names...] ボタンをクリックして、必要なクラスまたはメソッドの "" へのマッピングを削除します。

掲載されていないエラーが発生した場合

support@bea.com まで電子メールでお問い合わせください。その際、以下の情報を明記してください。

- 使用している WebLogic jCOM のバージョン
- 使用している JDK のバージョン
- 使用しているプラットフォーム（インストールされているサービス リリース / オプション パックを含む）
- Java ソフトウェアをアプレット / サーブレット / アプリケーション サーバで実行している場合は、バージョンの詳細
- *checkconfig config.log* によって生成された config.log
- 以下の説明に従って有効にしたロギングによって生成された jcom.log

特定のコンフィグレーション問題は、jcom\bin ディレクトリにある checkconfig ツールを実行することによって識別できます。詳しい使い方については、`checkconfig /?` を実行してください。

取得したエラーの性質によっては、WebLogic jCOM で生成できる詳細なエラー情報が役立ちます。

WebLogic jCOM Java ランタイムでロギング情報を *jcom.log* ファイルに生成するには、com.bea.jcom.Log.logImmediately 呼び出しを追加します。これは、Java コードの中で一度だけ呼び出す必要があります。これを追加する最も簡単な方法は、（Java から COM へ、または COM から Java への呼び出し時に）使用する最初の Java クラスの静的ブロックにその呼び出しを追加します。

```
public class YourClass {
  static {
    com.bea.jcom.Log.logImmediately(3, "c:\\jcom.log");
  }
  ...
}
```

アプレットを使用する場合、次のようなロギングを Java コンソールに生成できます。

```
static {
  com.bea.jcom.Log.logImmediately(3, System.err);
}
```

3 はログ レベルで、冗長を意味します。2 番目のパラメータとして任意のファイル名を指定できます。

WebLogic jCOM Moniker（jintmk.dll --「接着剤」）の内部でロギングを有効にするには、レジストリ エディタを使用して、`HKEY_LOCAL_MACHINE\SOFTWARE\Linar\JintMkr` の下に値を作成します。この値は <:fc 11>logFile<:/fc> という名前にし、`c:\temp\jcommk.log` などのファイル名に設定します。

問題が発生した場合は、上記の両方のログをお送りください。