JNDI迷宮を進む

（落ち着いて）beanのホーム・インターフェースの見つけ方を学ぶ

Daniel Would (WOULDD@uk.ibm.com)
CICS System Test
IBM

2003年 11月 18日

単一マシンのプログラミングから、EJB技術と分散コンピューティングという、より野蛮な世界に移ったJava開発者は困難に陥りがちです。JNDIの迷路をうまく通過できるようなコードを書くのは難しく、マシンや設定が複数になると何かがおかしくなる可能性も大きくなります。この記事ではEJB開発者であるDaniel Wouldが、JNDI名前空間の中で公開されているEJBコンポーネントにうまくたどり着くためにはクライアント・コードをどう書けばよいかを解説し、その作業を楽にする各種のプログラミング・オプションを説明します。また読者のアプリケーションでユーティリティ・クラスとして使えるコードもいくつかお見せします。

読者がクライアント・アプリケーションでEJBコンポーネントを見えるようにするのに問題を経験したことがなく、異なった条件でJavaプラットフォームをインストールした環境や全く違ったマシン上でクライアントやbeanを実行するのが複雑だと思ったこともないのであれば、この記事は面白くもないでしょう。ただし、もし読者がまだ始めたばかりであれば、そして実際に構成されたシステム上で何かをした途端におかしな、よく分からないエラー・メッセージが次々に出てくるようなら、このまま読み続けてください。

EJBエラー？　あわてずに！

あなたはエンタープライズJavaBeansを解説したお気に入りのJavaの本で、必要な節は読み終わりました。そしてHelloWorld beanも理解し、推奨の展開手順に従って公開しました。今度はこの傑作を呼び出すクライアントを書かなければなりません。そこでリスト1に示すようなものができあがります。

リスト1. beanを呼び出す、非常に単純なクライアント

```
InitialContext ic = new InitialContext();
Object or = ic.lookup("ejb/HelloWorldHome");
if (or != null) {
  // Narrow the return object to the Home class type
    HelloWorldHome home =
      (HelloWorldHome)PortableRemoteObject.narrow(or,
        HelloWorldHome.class);
  // Create an EJB object instance using the home interface.
    HelloWorld hw = home.create();
  // Invoke the method
    System.out.Println(hw.hello());
}
```

Javaをインストールした一番手近にあるもの、つまりアプリケーション・サーバーが使用しているマシン上でこのクライアントをコマンドラインで実行します。全て完璧に動作！成功に気を良くして、クライアントを2台目のマシンで実行します。今度はとんでもないメッセージが出ます。最初はおそらく`java.lang.NoClassDefFoundError: javax/ejb/EJBObject`で、その後に別の`NoClassDefFoundError`が山のように続いているでしょう。必要なスタブやタイのあるJARファイルを渡し忘れたり、EJBに関連したいろいろなものが無かったり、足りなかったりしているためです。それでも最初の面白い行（`InitialContext ic = new InitialContext();`）の所まではクライアントが動きました。この行にまで来た時に見える例外は（まず必ず例外が見えると思いますが）、選んだ特定のコンテキスト・プロバイダーによって異なります。

コンテキスト？　なんですかそれは？　用語を整理しましょう

先に進む前に、いくつか用語を定義しておくことにしましょう。コンピューターの世界はおかしな用語や流行語、略語などが散らかっていますが、Java技術も例外ではありません。（それとも`JavaIsNoException`でしょうか。）上で説明したような問題に突き当たると、読者としてはこの分野で使われている用語に震え上がってしまうかも知れません。そこで、この記事で飛び交う用語を説明しておきましょう。用語をきちんとしておくのは良いことです。

名前空間、コンテキスト、初期コンテキスト、そしてサブコンテキスト
: これらの用語は全て位置に関するものです。つまりクライアントの視点から見た、EJBコンポーネントが存在する概念的な位置です。例えとして名前空間を町と考えてみてください。町にある店はEJBホーム・インターフェース（これについてはすぐに説明します）で表現されます。コンテキストは町の中での位置です。初期コンテキストは最初に始める所、ちょうど町に入る道のようなものです。そしてサブコンテキストは通りの名前です。

ホーム・インターフェースとリモート・インターフェース
: エンタープライズJavaBeanコンポーネントには3つの部分があります。第１はbeanコード自体です。次はホーム・インターフェースで、これによって望むタイプのEJB beanを作るためのメソッドを定義します。ホーム・インターフェースは名前空間で公開されます。ホーム・インターフェースがあると、`Create()`を呼んでアプリケーション・サーバーからリモート・インターフェースを取得することができます。リモート・インターフェースを取得できると、実際のEJBコードを構成するメソッドを呼ぶことができます。 こうした用語を町の例えに合わせるにはどうすべきでしょう。目指す町に着き、目指す住所が分かったら、店の中に入るか呼び鈴を鳴らします（`Create()`を呼びます）。この手順はどの店に行く場合も同じですが、店からの反応はその店がどういうサービスを提供するか、例えば肉屋なのか、パン屋なのか、ロウソク屋なのかによって異なります。こうした反応がリモート・インターフェース

です。人はみな異なり、異なる事を要求されます。適切な依頼をするには（つまり適切なメソッドを呼ぶには）、話しかけている相手のタイプ（つまりbean）を知っていなければなりません。肉屋にパンを注文してはいけないのです。

CosNaming、LDAPそしてJNDI

Java Naming and Directory Interface(JNDI)は名前空間とどう相互動作すべきかを指示する標準インターフェースです。LDAPとCosNamingは、ここではJNDI名前空間のタイプです。町の例えを拡張して言うと、JNDIは町のひな形、CosNamingとLDAPは具体的な町になります。どの町も機能は似ていますが配置にはそれぞれ違いがあります。

プロパティは地図になる

ではこうした要素を使って、リモートのマシンからEJBコンポーネントにあるメソッドをうまく呼ぶにはどうしたら良いかを見てみましょう。クライアント・プログラムが、注意深く作られたEJBコンポーネントにうまく接続できるためには、いくつか必要なものがあります。第１にクライアント・コードに対する全JARファイル、それにJ2EE.jarのような汎用のEJB関連JARファイル、beanの展開中に生成されたスタブやタイなどです。こうしたファイルでクライアントを最初の初期コンテキストまで持って行くのです。

クライアントが必要とする情報として次に来るのは一部のプロパティの値です。最初に`java.naming.factory.initial`に何か値が必要です。このプロパティは初期コンテキスト・ファクトリを提供するクラスを指します。このプロパティの典型的な値は（この例でも使うのですが）`com.sun.jndi.cosnaming.CNCtxFactory`です。このクラスは`rt.jar`にありますが、これはベースJVMの一部だということです。CosNamingネームサーバーがこのファクトリを使うのですが、JVMにもLDAPファクトリがあります。後で見るように、アプリケーション・サーバーにはそれぞれ独自の初期コンテキスト・ファクトリがあるのです。

このクラスはネームサーバーURLの詳細とポート番号と共に、名前空間との相互動作に使う`InitialContext`クラスの生成に使われます。ただしプロバイダー名が無い場合には単純にポート900（またはコンテキスト・ファクトリのデフォルト・ポート番号ならどれでも）のlocalhostに接続します。リモート・サーバーに接続するにはプロパティ`java.naming.provider.url`に値が必要です。

これが新人プログラマーにとってこんなにややこしいのは、アプリケーション・サーバーのローカルで実行する限りは、だいたいどれも動作するのです。そこの環境が全て面倒を見てくれて、`InitialContext`を要求すれば、環境が適切な初期コンテキストを答えてくれるのです。ところが一旦クライアントを別のマシンに移すと、何から何まで自分でしなければなりません。どのJARファイルをコピーするのか、どんな設定をするのかを全部自分で知らなければなりません。私が知っている人の中には、ただクライアントを適切に動作させるだけのためにアプリケーション・サーバーのJARファイルの一式全部を別のマシンにコピーせざるを得なかった人がいるのです！

デフォルトとして`InitialContext`ファクトリは`jndi.properties`で定義されており、このファクトリ・クラスはサーバーURLとポート番号のデフォルトを持っています。このファイルはクラスパス（通常はローカル・ディレクトリを意味します）または、クラスパスにあるどれかJARの中にあります。アプリケーション・サーバーによっては違ったJARファイルにデフォルトがあるかも知れません。WebSphere Application Serverではデフォルト・コピーを`namingclient.jar`に保存しています。自分独自のデフォルトを規定するには、クラスパスの最初にあるファイルのコピーを編集

します。これがプロパティを設定する一つの方法です。コマンドラインからの設定やコードで行われる設定が無い場合には、クライアントは`jndi.properties`にある値を使います。単純な設定にはこれでも良いかも知れませんが、複数のサーバーや名前空間を相手にする場合には、クライアント毎に設定を行うハメになります。

こうしたプロパティの値は、使いたい名前空間という点から見た時にどう違うのでしょう。先に説明しましたが、JNDI名前空間にはちょっと違った2つの種類、CosNamingとLDAPがあり、それぞれには関連のトランスポート、IIOPとLDAPがあります。LDAP名前空間はLDAPのトランスポートを使い（`ldap://myldapnameserver`のようなURLで接続します）、CosNamingはIIOPのトランスポートを使います（`iiop://mycosnamingserver`のようなURLで接続します）。CosNamingのデフォルトのポート番号は900で、LDAPのデフォルトは389です。ただし、名前空間のサーバー実装によっては異なる場合もあります。

コマンドラインからの設定プロパティ

コマンドラインからプロパティを設定するにはどうすべきかを見てみましょう。ホームで遊んでみたければ、JDKインストールの中にある`bin`フォルダーに行ってみてください。このフォルダーの中に`tnameserv.exe`（Windowsの場合）または単に`tnameserv`（UNIXベースのシステムの場合）という名前のプログラムがあります。このプログラムを実行するとサンプルのCosNamingネームサーバーがポート900で起動します。

そろそろCosNaming名前空間を見る事ができるユーティリティを用意すべきでしょう。私は個人的には開発環境としてEclipseを使います。私が使ったJNDIブラウザ・プラグインへのリンクは参考文献に置いておきます。理論的には読者のマシンのポート900で名前空間ブラウザを指す事ができるはずで、ごくつまらない、空の名前空間が見えるはずです（ただしアプリケーション・サーバーによっては、デフォルトで名前空間にいろいろなものを入れている場合もあります）。私たちの名前空間を活気づけるために、中に何かを入れるための簡単なプログラムを書いてみましょう（リスト2）。

リスト2. cosNaming名前空間との相互動作の簡単な例

```
package example.publisher;

import javax.naming.InitialContext;

public class Publish {

    public static void main(String[] args) {
        //
        //This example creates a subcontext in a namespace
        //
        try{
            InitialContext ic = new InitialContext();
            ic.createSubcontext("Test");
        }catch(Exception e){
            System.out.println(e);
            e.printStackTrace();

        }
    }
}
```

このアプリケーションでは、適切な初期コンテキストを取得するために必要な全てのプロパティは得られるものと単純に想定しています。これでコマンドラインから実行できますが、実行する時には（URLを環境に合わせ）そうしたプロパティを与えます。

```
java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
     -Djava.naming.provider.url=iiop://mymachine:900
       example.publisher.Publish
```

全てうまく行っているので、私たちのクライアントが（例としてのネームサーバーの）コンテキストを検出し、`Test`というサブコンテキストを生成します。これは名前空間ブラウザを使って確認する事ができます。

今度は同じコマンドラインを使って（当然URLは調整した上で）、1台のマシンでネームサーバーを実行し、リスト2のアプリケーションを別のマシンで実行してみてください。問題なく動作するはずです。（バインドされているものを変更するためにサンプルを手直ししたり、もっと極端に、サブコンテキストを生成する代わりに削除したりしてみてください。２回目に実行したときには動作する事を確認できるはずです。）

アプリケーション内でプロパティを設定する

こうしたオプションをコマンドラインでは設定したくないとしたらどうでしょう？これには代替手段があり、こうしたプロパティをプログラムの内部で設定できるのです。これは`java`コマンドに特別なオプションを与える必要がない事を意味します。必要なプロパティを明示的に設定するようにリスト2のコードを更新すると、リスト3のようになります。

リスト3. cosNaming名前空間との相互動作の簡単な例　プロパティをアプリケーション・コード内で設定する

```
package example.publisher;

import javax.naming.InitialContext;

public class Publish {

    public static void main(String[] args) {
        //
        //This example creates a subcontext in a namespace
        //
        try{
            Properties prop = new Properties();
            prop.setProperty("java.naming.factory.initial",
              "com.sun.jndi.cosnaming.CNCtxFactory");
            prop.setProperty("java.naming.provider.url",
              "iiop://mymachine:900");
            InitialContext ic = new InitialContext(prop);
            ic.createSubcontext("Test");
        }catch(Exception e){
            System.out.println(e);
            e.printStackTrace();

        }
    }
}
```

これでこのプログラムは長々としたコマンドライン設定が必要なくなりました。ただし、この方法で書かれたアプリケーションは設定にハードコード化されていることに注意してください。

beanへの道を探る

ここまではリモート名前空間に接続されて何かをしたという前提で、いくつか基本的な例を見てきました。もちろん何かをしたといっても、ごくつまらない、サブコンテキストを生成するという事だけです。実際にはこのツールを使えば、何でも生成したり公開したりすることができるはずです。本当にしたいのはオブジェクトを参照することでしょう。この節ではCosNaming名前空間で公開されている、HelloWorld beanへのHomeインターフェースを取得します。その次にLDAP名前空間ではどうするかを見ていきます。

議論を進めるために、HelloWorld beanは展開してあり、そのホーム・インターフェース`HelloWorldHome`が`example/HelloWorldHome`で公開されているとしましょう。（これを試してみたいけれども自分でHelloWorld beanを生成したくないのであれば、あらかじめパッケージされたbeanのJARファイルと、それに伴うクライアント用ファイルをダウンロードするためのリンクが参考文献にあります。）

コンテキストのヒント

名前空間URLフォーマットでは、初期コンテキストを設定してデフォルトよりもツリーのずっと上の方で開始するようにする事ができます。例えば、プロバイダーURLに私たちの例の`iiop://mymachine:900/example`を使えば、`example/HelloWorldHome`ではなく`HelloWorldHome`を参照しさえすれば良いのです。つまり初期コンテキストは`example`内部にあるのです。これは名前空間の同じ構造の下でいくつかの参照をするときには便利です。こうすることで、設定を変更した場合にも、変更しなければならないコード部分はプロバイダーURLだけですむのです。

この前の節では、ネームサーバーに接続するという大仕事をしました。今度はEJBコンポーネントを参照する事だけです。それには文字列を参照メソッドに渡す必要がありますが、これは言ってみれば`InitialContext`（町の中での出発点）から`HomeInterface`（家はまたは店）に行く道順を表します。簡単そうですね。ところが、あなたが選んだ特定のコンテキスト・ファクトリが影響し始めるのは正にここなのです。WebSphereのようなアプリケーション・サーバーについてくるファクトリ・クラスでは、自分が必ずしも名前空間のルートに置かれるとは限らないのです。ですから`HomeInterface`の変化を参照するために必要な文字列は、`InitialContext`が自分を町のどこに置くかによって変わるのです。それだけではなく、（コンテキスト・ファクトリによって）ローカル・サーバーでは、リモート・サーバーとは異なる開始ルートに置かれてしまうかも知れないのです。

この理由から、使うべき参照文字列はリスト3のようにハードコード化するのは避け、コマンドラインまたはプロパティ・ファイルから渡すようにした方が良いと思います。複数のステップがある構成では特に言える事です。例えば、EJBコンポーネントを呼ぶクライアントがあるかも知れません。そのbeanが今度は2番目のEJBコンポーネントを呼ぶ必要があり、そのEJBコンポーネントは別のサーバーにあるかも知れないのです！そういう場合には、各ステップを通してプロパティを渡す必要があります。こうすることで、試しては修正する簡単な参照機構ができ、その上最終的なアプリケーションが、新しいネームサーバーへの展開のように比較的小さな変更に柔軟に対応できるようになります。では参照アプリケーションの例を見てみましょう。リスト4ではプロパティはプログラム的に設定されますが、コマンドラインの値を元にしています。その結果、見ていただくと分かると思いますが、コマンドラインの見え方が前の例とはちょっと違っています。

リスト4. ホーム・インターフェースを参照する

```
package example.lookup;
```

```
import java.util.Properties;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import example.HelloWorld;
import example.HelloWorldBean;
import example.HelloWorldHome;
import javax.naming.InitialContext;

public class Lookup {

    public static void main(String[] args) {
        //
        //This example looks up the home interface of an EJB to a namespace
        //
        try{
            Properties prop = new Properties();
            prop.setProperty("java.naming.factory.initial",args[0]);
            prop.setProperty("java.naming.provider.url",args[1]);
            InitialContext ic = new InitialContext(prop);
            Object or = ic.lookup(args[2]);
            if (or != null) {
                // Narrow the return object to the Home class type
                HelloWorldHome home =
                  (HelloWorldHome)PortableRemoteObject.narrow(or,
                      HelloWorldHome.class);
                // Create an EJB object instance using the home interface.
                HelloWorld hw = home.create();
                // Invoke the method
                System.out.Println(hw.hello());
            }
        }catch(Exception e){
            System.out.println(e);
            e.printStackTrace();

        }
    }
}
```

このプログラムは3つのパラメーターで呼び出されます。使用するコンテキスト・ファクトリとプロバイダーURL、参照すべき名前を含む文字列、の3つです。初めの２つはもう分かっています。3番目は何でしょう？

ネームサーバーとしてまだ`tnameserv`を使っているのであれば、おそらくbeanを直接`/example/HelloWorldHome`に公開したでしょう。その場合には、単純に`/example/HelloWorldHome`を3番目のパラメーターとして渡せば参照はうまく行くはずです。ところが使っているネームサーバーがもっと複雑な名前空間を使っている場合には、使用している展開ツールが強制する追加レベルがあるかも知れません。例えばWebSphereはJavaBeanコンポーネントをデフォルトで`ejb/`に展開しますが、これは名前空間のルートではありません。WebSphereのコンテキスト・ファクトリを使った場合にのみ、文字列`/ejb/example/HelloWorldHome`で渡す事で名前空間の適切な場所に置かれるのです。アプリケーション・サーバーが提供するものとは異なるコンテキスト・ファクトリを使った場合（標準のJavaをインストールしただけのマシンでクライアントを実行する場合にはあり得ます）にはもっと問題が悪化します。ただし、アプリケーション・サーバーのネームサーバーに関する資料にはEJBコンポーネントの参照をする際には名前空間のどこで開始するかを説明してあるはずです。資料にある例を見てからブラウザで名前空間を見て、クライアントの`InitialContext`がbeanをどこに置くかを確定してください。名前空間はループしている場合がよくあります。ですから枝をたどって行けば無限にたどり着くことができます。これはつまり、

大部分の開始コンテキストから、望む家にたどり着くための道を見つける事ができるということを意味しているのです。

いずれにせよ、リスト4のアプリケーションに適切なパラメーターを渡すコマンドラインは次のようなものです。

```
java Lookup com.sun.jndi.cosnaming.CNCtxFactory
  iiop://mymachine:900 example/HelloWorldHome
```

CosNamingでは名前空間サブコンテキストは、標準的なURLのようにスラッシュ（ `/` ）で区切られます。LDAPの構文は異なります。これを次に見ていきます。

LDAPを導入する

今度はLDAPに行きましょう。ここではLDAPはもう一つのJNDI名前空間ですが、その構造はCosNaming名前空間の構造とは異なった形で表現されます。LDAPはコンテキスト・ファクトリも違うものを要求します。ただしこの点は、いずれにせよコマンドラインで適切なファクトリを指定するので問題にはなりません。（ ここでの例ではベースJVMの一部にあるものを使いますが、ここでもアプリケーション・サーバーによっては独自のファクトリがある場合もある事に注意してください。 ）また、LDAPは異なるネームサーバーへのポインターも要求するのですが、幸いこれもコマンドラインで指定できるのです。もちろんホーム・インターフェースの位置を表す文字列は異なりますが、どう思いますか？そう、これもコマンドラインで指定するのです。こうした、コマンドライン・コールを使う利点が分かるでしょう。必要なのはテストプログラムの呼び方を変えるだけで、理論的にはどんなJNDIネームサーバーにも到達でき、CosNamingからLDAPへ切り替えさえ問題なく、しかも何のコード変更も必要なくできてしまうのです。ただし、それは理論的な話であって、実際にはパラメーターを適切にする事が秘訣なのです。

一部のネームサーバーは名前空間の一部を保護します。つまり許可された部分しか公開できないということです。実行中のLDAPサーバーがあり、その詳細が次のようであるとしましょう。

- URL:`ldap://mymachine:1389`
- BaseDN:`c=myldap`

LDAP名前空間でのツリーの構造は一般的にこのようになります。

```
ibm-wsnName=MyServer,ibm-wsnName=HelloWorldHome
```

ところがこの文字列を私たちのプログラムに渡すときには逆にする必要があるのです（ 私に理由を聞かないでください ）。ですから私たちが使う文字列はこのようになります。

```
ibm-wsnName=HelloWorldHome,ibm-wsnName=MyServer,
```

BaseDNは名前空間のどこで開始したいかを表します。対象のLDAPネームサーバーがどう構成されているかにより、いろいろあり得ます。この例では真っ直ぐに`c=myldap`のルートに行きます。名前空間の中のツリーにジャンプしたいときには、その点にまでジャンプする代わりに、BaseDNとして`ibm-wsnTree=myTree,c=myldap`を規定します。

ですからプログラムに渡すコマンドライン・パラメーターは次のようになります。

```
java Lookup com.sun.jndi.ldap.LdapCtxFactory ldap://mymachine:1389/c=myldap/
     ibm-wsnName=HelloWorldHome,ibm-wsnName=MyServer
```

出る可能性のあるエラー・メッセージ

いろいろ注意したとしても、EJBコンポーネントを展開する際には否応なく各種の例外メッセージとお馴染みになるでしょう。頻繁に見る事になるエラー・メッセージと、それが出る理由のいくつかは次のようなものです。

- `CORBA.OBJECT_NOT_EXIST`: EJBを見る場所を間違えています
- `CORBA.MARSHALL _something`: CORBA marshall 例外は頻繁に起こります。詳細はそれぞれ異なりますが、基本的には同じことを意味します。何かデータを取得したが、そのデータはクライアントが期待したものではない、ということです。参照を間違えたのかも知れないし、実際に展開されているEJBコンポーネント・クラスのバージョンと、クライアントが知っているバージョンとが違っているのかも知れません。または何らかの問題があり、クライアントORBはサーバーORBが送信したものを単に理解できないだけなのかも知れません。
- `javax.naming.NoInitialContextException`: あれ！コンテキスト・ファクトリまたはプロバイダーURLを指定し損なったか、単にネームサーバーが動作していないだけなのかも知れません。

ここでLDAPコンテキスト・ファクトリを指定し、次に、開始したい点でLDAPサーバーの名前を渡します。その次に、参照したいEJBコンポーネントへの予約パスを渡します。そしてこのコマンドラインを使って、CosNamingの例で使ったのと同じコード（リスト4）を呼び出します。

もちろん、この記事で使ったコードでヘルパー・クラスのメソッドを作れないという理由は何もありません。（ヘルパー・クラスは3つのパラメーターを取り込み、参照を制限しようと試みる前に`ic.lookup(args[2])`コールで返されるObject orを返すものです。）そうすれば、どこで参照をしたい場合でも単にこのヘルパー・クラスを使い、現在の状況を表す適切なパラメーターを渡し、必要なオブジェクト参照を戻して本当のクラスにまで制限するようにできます。（**注意**：私はこうしたクラスのパフォーマンスを保証するわけではありませんし、こうしたコード断片は「現状のままの状態」で提供します。私もIBMもなんら保証してはいませんのでご注意下さい。）もちろん、反映（reflection）を使って完全に汎用の方法をとる事もできますが、ここで説明したよりもずっと複雑ですし、この記事の範囲外になります。

終了する前に検討すべき事が一つあります。リスト3と4で使った技法を組み合わせたクライアントを書く事もできるのです。そういうクライアントであればコマンドラインで値が与えられたかどうかをチェックし、与えられていればその値を設定します。与えられていなければハードコード化された値を使うのです。こうすればプログラムのデフォルト値は意味のあるものなり、必要あればコマンドラインのオプションで上書きする事もできるのです。そのために必要な調整はごく僅かですみます。

無事に到着

まとめです。複数のEJB参照やアプリケーションがどんなシステムでも動作するためには、次の４点を理解して実行する事が必要です。

- どの段階においても、次の段階の全スタブやタイはクラスパスになければなりません。ローカル環境が、そのクラスがどのように見えるかを知らない限り、使用するオブジェクトを制限する事ができないのです。

- 各段階には`J2EE.jar`のような汎用のEJB関連JARファイルが必要です。
- コンテキスト・ファクトリのタイプやネームサーバー名、JNDI参照文字列をパラメーターとして渡します。こうすることで容易に変更に対応できるようになります。
- 自分の名前空間を知っている必要があります。JNDI参照文字列は名前空間での開始位置からオブジェクトが保存される場所まで、明示できている必要があります。ただし、同じ場所から開始するとは限らない事に注意してください！ツールを使って名前空間をブラウズし、ローカル参照、リモート参照を開始する場所を見つける必要があります。

同じマシンですべて実行するようなコードを書いてきた開発者にとっては、リモートの名前空間をたどって行くのには戸惑うでしょう。この記事でご紹介したヒントやコードが、分散EJBアプリケーションを立ち上げて動くようにするのに役に立つと思います。JNDI名前空間に熟達したら、developerWorks にあるBrett McLaughlinによるEJBベスト・プラクティスのシリーズ（参考文献）をよく読んでください。コード最適化のための非常に良いヒントがあります。

著者について

Daniel Would

Daniel Wouldは2001年7月にIBMに入社しました。CICS システム・テスターとしての現在の仕事に移る前にはCTGシステム・テスターとして１年間働いてきました。IBMではJava技術やEJBコンポーネントを中心とした仕事をしています。連絡先はwouldd@uk.ibm.comです。