※注：本資料は、JRcServer Standard版に対してのものであり、
現在（2005/08/30）時点のバージョンは a020 でPrototype版
なので、内容として、大きく異なります。

2005/08/30

JRcServerの利点と欠点

既存のEJBの問題点を解決する、新しい規格の
分散環境プロジェクト

Copyright (c) 2005 masahito suzuki, Inc. All Rights Reserved

１．ＪＲｃＳｅｒｖｅｒとは、何なのか？

JRcServer[Java-RemoteCommand-Server]は、既存のＥＪＢの問題点を解決する、新しい規格の分散環境プロジェクトである。しかし、これだけでは、「JRcServerとは、何なのか？」とか、「比較対照である、ＥＪＢにどのような問題点があるのか？」と、思われる方も多いと思うので、これらの内容を順次説明する。

まず始めに、既存のＥＪＢの問題点を挙げる。

現在の分散環境の主流は、EJB[Enterprise-Java-Bean]であり、多くのエンタープライズ・システムで採用されている。しかし、ＥＪＢは、多くの問題点があり、大規模システムなどの、ＥＪＢを利用しないと、開発が困難な場合以外は、あまり採用されていない。

それは、既存のＥＪＢは、「重量級のコンテナ」であることから、コンポーネント開発は、思ったよりも困難で、面倒であることが、１つの問題点となっている。
もう１つの問題点として、ＥＪＢコンテナと、それを呼び出す側が、同一マシン上の場合と、リモート間の場合では、呼び出し側のプログラム実装が変わってしまうのである。そして、リモート上にＥＪＢコンテナを配置する場合、呼び出し側は、ホスト名（IPアドレス）＋ポート番号を設定して、ＥＪＢコンテナの位置を指定しなければならない。
最後の問題点として、ＥＪＢコンテナを利用する場合、同じＪ２ＥＥのＷｅｂコンテナ（Ｓｅｒｖｌｅｔや、ＪＳＰ）から利用する事を前提としているため、プログラム言語は、JAVA言語を想定している。そのため、ＥＪＢコンテナは、他のプログラム言語からは、単純には利用できないので、再利用率が低いと言える（ＣＯＲＢＡや、ブリッジからだと、他言語から利用できなくもないが、これはこれで、面倒な作業のため、「単純」ではない）。

しかし、ＥＪＢはこのような問題点があるにも関わらず、エンタープライズ・システムに採用されるのは、その規模のプロジェクトに対して、構築に欠かせない機能（JTA、JMSなど）がある事と、処理を分散できることで、システムに耐久力を持つことが出来る事と、ベンダやオープンソースなど、多くの製品があることから、業界標準として、確立しているからである。だが、冒頭の「あまり採用されない」とあるように、小～中規模のＷｅｂアプリケーションでは、積極的にＥＪＢを使うことは、ほとんどない。

このように、既存のＥＪＢには、多くの問題点があるが、特に「重量級のコンテナ」が問題であることから、小～中規模のＷｅｂアプリケーションからは、ほとんど利用されていない。しかし、本来分散環境の利点である、「拡張性」と「再利用」は、別段エンタープライズ・システムにのみ、有効と言うわけではなく、むしろサーバを持つシステムにとっては、非常に重要な要素であるので、利用できるものなら、利用したい。

次に今回のプロジェクト「JRcServer」は、これら既存のＥＪＢの問題点を、どのように解決しているかを説明する。

JRcServerは、「軽量コンテナ」として、分散環境を提供し、上記の「拡張性」の問題点や、「再利用」の範囲を解決することで、下記の３点により、効率の良い、分散環境を提供していく。

・初心者にやさしい分散環境を提供し、コンポーネントの開発や、**※１**ＪＲｃコンテナの呼び出しを、楽で、簡単にする。
・プログラムの実装においては、「ネットワーク」や、「**※１**ＪＲｃコンテナの位置や状況」を意識させないようにすることで、拡張性を向上させる。
・他のプログラム言語から、**※１**ＪＲｃコンテナの呼び出せることを、ＪＲｃＳｅｒｖｅｒのテーマとすることで、コンポーネントの再利用率を向上させる（ＯＤＢＣのように、ＯＳやプログラム言語を意識しなくても、利用できるようにする）。

しかし、言葉で「簡単」とか「拡張性がある」とか、「再利用率が向上」とか、言っても、文面で説明するだけでは、理解することは難しいと思う。
そこで、本資料は、JRcServerを、既存のＥＪＢと、コーディング例から比較することで、これらの利点と欠点を列挙し、具体的に、「JRcServerとは、何なのか？」の問いに、答えていく。

※1：「付属1：JRcＳｅｒｖｅｒ構成図」の、<u>JRcServer</u> 部分が <u>JRcコンテナ</u> に該当する

２．ＪＲｃＳｅｒｖｅｒと既存のＥＪＢとの比較

２．１．コンポーネント作成から登録までの作業工程を比較してみる

まず、始めに分散環境をＥＪＢで、作成／登録する作業の流れについて、簡単ではあるが説明する。

A. **リモート・インターフェイスを作成。**
B. **Home インターフェイスを作成。**
C. **Enterprise Beanの実装クラスを作成。**
D. ***デプロイメント・ディスクリプタを作成（DeployToolを利用）。***
E. A～Cまでをコンパイルし、AからDまでをｅｊｂ－ jar化させる。
F. Ｅで作成したｅｊｂ－ jarファイルをＥＪＢコンテナにデプロイメント(対象ディレクトリにセット)。

注：今回のＥＪＢの作成手段は、Ｊ２ＥＥの「DeployTool」を用いて作成していく

このうち、A～Cまでは、コンポーネントを作る要素となる、Ｊａｖａプログラムで、Dはデプロイ先を定義（DeployToolを利用）する内容となる。そして、ＥとＦによって、コンポーネントをコンパイルし、その内容をＥＪＢコンテナにデプロイメントさせている。

次に、同じような分散環境をJRcServerで、作成／登録する作業の流れについて、簡単ではあるが説明する。

ａ デプロイ先のディレクトリ(例 ${JRCSERVER_HOME}/deploy)を指定。
(これは１度設定した場合は、必要ない)
ｂ **JRcServerコンポーネントを作成。**
ｃ. ｂをコンパイルし、ＪAR化
ｄ ｃで作成したＪＡＲファイルを特定のディレクトリに設定(例 ${JRCSERVER_HOME}/deploy)。
ｅ ***ｄで登録したコンポーネントに名前を付ける。***

このうち、ｂがコンポーネントをつくる要素となる、Ｊａｖａプログラムで、ｅが、JRcServerに対象コンポーネントを定義する内容となる。そして、ｃとｄによって、コンポーネントをコンパイルし、その内容をＪＲｃコンテナにデプロイメントさせている。

これまでの中で、２つの共通工程として同じものは、以下の通りである。

・コンパイルし、ＪARファイルに変換。(厳密には少し違うが、行っている作業工程は同じであるため、同一とする)
・対象のＪARファイルをディレクトリにセット。

次に、２つの工程が違うものは、以下の通りである。

・JRcServerは、コンポーネントを示すＪＡＶＡファイルは１つ。
>> EJBは、コンポーネントを示すＪａｖａファイルは３つ。

・JRcServerは、ＪARファイルのコンポーネントに、コマンド名を付与する必要がある。
>> EJBは、デプロイメント・ディスクリプタ(DeployToolから登録)によって、デプロイ先が定義されている。

このように、ＪＲｃＳｅｒｖｅｒは、ＥＪＢとは違い、１つのＪＡＶＡファイルでコンポーネントを表現できる。この部分は、ＥＪＢとしては、「C. Enterprise Beanの実装クラスを作成」にあたる部分であり、ＡやＢの余分なインターフェイスを定義しなくても良い。

しかし、ＥＪＢとは違い、ＪＲｃＳｅｒｖｅｒは、JRcCNS[JRc-Component-Naming-Service]に対して、登録コンポーネントに、名前を付与する必要がある。ＥＪＢでは、この工程を「D. デプロイメント・ディスクリプタ」により、ＤｅｐｌｏｙToolを使って、コンポーネント定義を行っているが、JRcＳｅｒｖｅｒの場合は、ＪRcＣｌｉｅｎｔから、ＲＯＯＴユーザでコマンドを使って、登録しなければならない。

これら作業工程の内容をまとめると、コンポーネントの作成～登録までを比較した場合、JRcServerには、以下の利点と欠点がある。

[利点] **コンポーネントを構成するＪＡＶＡプログラムは１つのファイルでよい。**

[欠点] **ＪＲｃコンテナにコンポーネントを登録する場合、ＪＲｃＣＮＳに対して、ＪＲｃＣｌientからＲＯＯＴユーザで、登録する必要がある。**ただし、ＥＪＢコンテナにコンポーネントを登録するためにＤｅｐｌoyToolを使う必要があることから、小さな「欠点」であると言える。

このように、「利点」としては、ＥＪＢよりも、ＪＲｃＳｅrverで、コンポーネントを生成したほうが、１つのＪＡＶＡファイルで作成できるので、「軽量である」と言える。

しかし、「欠点」として、ＪＲｃコンテナにコンポーネントを登録するには、ＥＪＢコンテナに登録する方法とは違い、JRcＣlient経由で、登録する必要がある。そして、この作業は、新規のコンポーネントを作成する度に、毎回発生するため、ＥＪＢと比べて、欠点であるといえる。しかし、ＥＪＢコンテナに、デプロイするために、ＤeplyToolを用いる必要があることから、それほど大きな差はないと思っている。また、これらを登録する時に、登録用シェルなどを作る事で、ＪＲｃコンテナに対して、自動化できるため、面倒な登録作業は、ある程度回避できる。

また、ＪＲｃＣＮＳに対して、コンポーネント名を登録しなくとも、コンポーネント自体は、呼び出すことが出来る。

[呼び出し例] “servername@rcmd com.JRcServer.sys.rcmd. RemoteLoginID ”

ここの[servername]の部分には、対象のJRcServer名をセットする。

しかし、上記の呼び出し方は、直接Javaオブジェクトを呼び出しているため、見た目にも美しくなく、使い勝手が悪い。また、デフォルトでは[rcmd]はＲＯＯＴユーザでしか呼び出せないので、セッションなどを利用することはできず、またＲＯＯＴユーザを汎用的に利用することは、危険であるので、通常運用は厳禁である。

２．２．コンポーネント作成から登録までのプログラムサンプルから比較してみる

２．２．１．ＥＪＢの分散環境のプログラムサンプルを見る

始めに、JRcServerの比較対照となる、ＥＪＢをコーディングレベルでみて見る。

EJB(SessionBean)を構成するための、プログラムや外部定義ファイルは、最低でも、以下の内容が必要となる。

A. Remoteインターフェイス。
B. Homeインターフェイス。
C. Beanクラス。
D. デプロイメントディスクリプタ定義。
E. 呼び出し側のプログラム。

AからＣまでは、ＥＪＢを構成するプログラムで、Dは、デプロイ条件を持つ外部定義ファイルで、これはＤeployToolを使った場合、自動的に作られる。
そして、Eは、ＥＪＢを利用した、呼び出し側のプログラムである。

初めに、Remoteインターフェイスのプログラムを説明する。

EJBHelloWorld.java

```
01:package test ;
02:import javax.ejb.* ;
03:import javax.rmi.* ;
04:
05:public interface EJBHelloWorld extends EJBObject
06:{
07:	public String getHello() throws RemoteException
08:}
```

このインターフェイスは、EJBが実行するオリジナルのメソッドを列挙するためのもので、ここでは、getHello メソッドが該当する。

つぎに、Ｈｏｍｅインターフェイスのプログラムを説明する。

EJBHelloWorldHome.java

```
01:package test ;
02:import javax.ejb.* ;
03:import javax.rmi.* ;
04:
05:public interface EJBHelloWorldHome extends EJBHome
06:{
07:        public EJBHelloWorld create() throws RemoteException,CreateException
08:}
```

このインターフェイスは、Ｒｅｍｏ teインターフェイス(EJBHelloworld)を返すための、create メソッドを持っている。

最後に、Ｂｅａｎクラスのプログラムを説明する。

EJBHelloWorldBean.java

```
01:package test ;
02:import javax.ejb.* ;
03:import javax.rmi.* ;
04:
05:public interface EJBHelloWorldBean implements SessionBean
06:{
07:        public String getHello() throws RemoteException{
08:                return “hello” ;
09:        }
10:        public void setSessionContext(SessionContext session ){}
11:        public void ejbCreate(){}
12:        public void ejbRemove(){}
13:        public void ejbActivate(){}
14:        public void ejbPassivate(){}
15:}
```

このオブジェクトは、Ｒemo teインターフェイスで定義した、オリジナルメソッドを実装する必要があり、ここでは、getHello メソッドが該当する。

このように、ＥＪＢは１つの分散コンポーネントをつくる場合、最低３つのプログラムを用意する必要がある。

つぎに、これら３つのプログラムから、ＥＪＢ－ＪＡＲを作成する必要がある。作成方法として、今回はＪ２ＥＥに付属する、DeployToolを利用する。これを利用することにより、別途デプロイメントディスクリプタを作成する必要がなくなり、ウインドウから、ウイザード形式で、ＥＪＢ－ＪＡＲを作れるので、便利である。また、このＥＪＢに対するＪＮＤＩ名は、「EJBHello」とする。
これらの工程で、分散コンポーネントが作成され、ＥＪＢコンテナに、デプロイメントされる。

次に、上記の内容で作成した、ＥＪＢを呼び出すプログラムを作る。呼び出す方式としては、プロセス間通信による、呼び出し方法と、リモート上のマシンから、分散コンポーネントを呼び出す、２つの方法がある。

まず初めに、プロセス間通信による、呼び出しプログラムから、説明する。

TestEJBClient.java

```
01:import javax.naming.* ;
02:import javax.rmi.* ;
03:
04:public class TestEJBClient
05:{
06:	public static void main( String[] args ){
07:		try{
08:			Context ctx = new InitialContext() ;
09:			EJBHelloWorldHome home = ( EJBHelloWorldHome ) PortableRemoteObject.narrow(ctc.lookup(“EJBHello”),
				EJBHelloWorldHome.class ) ;
10:			EJBHelloworld hello = home.create() ;
11:			System.out.println( “exec  =  ” + hello.getHello() ) ;
12:		}catch( Exception e ){
13:			e.printStackTrace() ;
14:		}
15:	}
16:}
```

このソースコード[TestEJBClient.java]をコンパイルして、実行すると、以下の結果が、テキストコンソールに表示される。

exec = hello

このように、普通にEJB を呼び出す場合、通常のプログラムとは違い、JNDI を意識する必要があることで、面倒な手続きを、プログラムに実装しなければならない。そして、このように、多くの手続きがあることから、仮に「初めて EJB を使ってみよう！」と言う人達が、EJBのプログラム実装を体験した時に、「なんだか良く分からない？」とか、「面倒くさい！！」と思ってしまい、全くメリットを感じない場合が多い。

つぎに、リモート上から、上記と同じように getHello メソッドを実施するプログラムを、説明する。

TestRMEJBClient.java

```
01:import javax.naming.* ;
02:import javax.rmi.* ;
03:
04:public class TestRMEJBClient
05:{
06:        public static void main( String[] args ){
07:                try{
08:                        Properties env = new Properties() ;
09:                        env.put( Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.cosnaming.CNCtxFactory" ) ;
10:                        env.put( Context.PROVIDER_URL, "iiop://IP-ADDRESS:PORT" ) ;
11:                        Context ctx = new InitialContext( env ) ;
12:                        EJBHelloWorldHome home = ( EJBHelloWorldHome ) PortableRemoteObject.narrow(ctc.lookup("EJBHello"),
                                EJBHelloWorldHome.class ) ;
13:                        EJBHelloworld hello = home.create() ;
14:                        System.out.println( "exec  =  " + hello.getHello() ) ;
15:                }catch( Exception e ){
16:                        e.printStackTrace() ;
14:                }
15:        }
```

このソースコード[TestRMEJBClient.java]をコンパイルして、実行すると、以下の結果が、テキストコンソールに表示される。

exec = hello

基本的には、[TestEJBClient.java]で呼び出す場合と同様なのだが、呼び出し元がリモート上にあるため、８から１１行までが、作業として増える。

このように、リモート間でＥＪＢを利用する場合、８から１１行までの分、実装しなければならなくなり、仮に、システムを構築した当初は、ＥＪＢと呼び出し側との関係が、プロセス間通信であったとして、その後の状況から、拡張する必要が発生し、これらの関係をリモート間に変更した場合、呼び出し側のプログラムを全て再実装＋再テストしなければいけなくなり、拡張性が悪い。
そして、１０行目にあるように、IP アドレス(またはドメイン名)＋ポート番号で、リモート先を指定しなければならず、リモート間で、ＥＪＢを使う場合は、接続先を意識しなければならず、移植性が損なわれてしまう。

このように、EJB を利用する場合、「拡張性」や「移植性」の悪さが目立つ。ただし、ＥＪＢコンテナを呼び出す時の通信プロトコルが、ＯＲＢであることから、Ｓｏｃｋｅｔなどの通信を意識したプログラム実装とは違って、通常のメソッドと同じように、プログラム実装ができる。それに JNDI は、様々なネーミングサービスに対応していることから、ＤＮＳを選択することで、インターネット上からも利用できる。

まとめると、EJB をコーディングレベルから見た場合、以下の利点と、欠点がある。

[利点]

・ORB であることから、通常のメソッドと同じように、プログラム実装ができる。
・JNDI は、様々なネーミングサービスに対応しているため、ＤＮＳを選択することで、インターネット上からも利用できる。
・BEA や IBM などベンダ提供のサーバや、JBOSS などのオープンソースが、EJB に対応しており、幅広い選択肢があり、
そして何より、実績がある。

[欠点]

・EJB を使いこなすためには、J2EE 自体を、ある程度理解しなければならず、そのことが、EJB を学ぶ人達に対して、
敷居が高いものとなっている。
・EJBは「重量級のコンテナ」であることから、コンポーネントの実装や、呼び出しに多くの手続きを要する。
・プロセス間通信で利用するのか、リモート上で利用するのかを考慮しなければならず、その状況に応じて、ＥＪＢコンテナを
呼び出す側は、実装方法が変わってしまう。このため、これらの関係を変更する場合は、呼び出し側の実装個所を
全て変更しなければならず、拡張性が悪い。
・EJBコンテナをリモート上で利用する場合、IP アドレス(ドメイン名)＋ポート番号で、接続先を決めるので、移植性が
損なわれてしまう。
・ＥＪＢコンテナを利用する場合、同じＪ２ＥＥのＷｅｂコンテナ（Servletや、JSP）から利用する事を前提としているため、
プログラム言語は、JAVA言語を想定している。そのため、ＥＪＢコンテナは、他のプログラム言語からは、単純には
利用できないので、再利用率が低いと言える（ＣＯＲＢＡや、ブリッジからだと、他言語から利用できなくもないが、
これはこれで、面倒な作業のため、「単純」ではない）。

２．２．２．ＪＲｃＳｅｒｖｅｒの分散環境のプログラムサンプルを見る

次に、JRcServerを利用する場合の内容を、ソースレベルで説明する。

JRcServerを使って、分散環境を実現させるためには、最低でも、以下の内容が必要である。

ａ 分散コンポーネントとなる、プログラム。
b. 呼び出し側のプログラム。

上記の内容のａは、サーバ側で動作する、分散コンポーネント用のプログラムである。
そして、ｂ．は、その分散環境を利用するクライアント側のプログラムである。

始めに、サーバ側で動作する分散コンポーネント用のプログラムを説明する。

CmpHelloWorld.java

```
01:package test ;
02:import com.JRcServer.sys.rcmd.* ;
03:import com.JRcServer.exception.* ;
04:
05:public class CmpHelloWorld extends JRemoteComponent
06:{
07:	public String execution( String[] args ) throws ExecutionException{
08:		return ( args == null ) ? “hello” : (“hello=” + args.length ) ;
09:	}
10:}
```

JRcServerのコンポーネントを作成する場合、[com.JRcServer.sys.rcmd.JRemoteComponent]を継承し、[execution]メソッドに、処理内容を記述するだけで、コンポーネントが作成出来る。また、この[execution]メソッドは、通常のＪａｖａアプリケーションで初めに実行されるmain メソッドに、良く似た形式を採用していることから、Ｊａｖａプログラム経験者であれば、すぐにでも、コンポーネント製作に取り掛かることが出来る。

上記で作成したプログラム [CmpHelloWorld.java]を、通常のJavaプログラムと同じように、コンパイル後、ＪＡR化することで、完了となる。
このサンプルでは、JAR名を[CmpHW.jar]とする。

その前に、ここからはJRcServerを起動しなくてはいけない。起動方法は、${JRCSERVER_HOME}/bin/jrcsv.cmd を実行し、暫くすれば、起動完了となる。

次に、ＪＲｃコンテナに対して、デプロイメントしなければいけないのだが、その前に、もしデプロイ先のディレクトリが決まっていない場合は、それを設定してやる必要がある。デプロイ先のディレクトリの登録方法として、Windows の場合、コマンドプロンプトから、ＪＲｃClientDriverを使って登録する。

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:>cd %JRCSERVER_HOME%
c:\JRcServer>mkdir deploy
c:\JRcServer>%JRCSERVER_HOME%\bin\jrccl.cmd user passwd
JRcClientDriver version XXX YYYY/MM/DD.
OK.
>servername@dyadd $(JRCSERVER_HOME)/deploy
>quit
```

※ここの user,passwd,servername は、それぞれ環境に合わせて設定しなければならない(以降同様の条件となる)。

上記のマーキング部分の入力情報により、コンポーネントのデプロイ先が $(JRCSERVER_HOME)/deploy」として、定義された。
（また、このコマンドdyadd は、ROOTユーザでしか実行できないので、ROOTユーザでログインしなければならない。）

あとは、作成コンポーネントJARファイル[CmpHW.jar]を、このディレクトリにセットしてやれば、後はJRcServerが自動で、デプロイしてくれる。

次に、このデプロイした、対象コンポーネントに対して、コマンド名を付与しなければならないのだが、これも、上記と同様に、JRcClientDriverを利用して行う。

基本的には先ほどと同じ手順で、起動する。(これも、ROOTユーザでログインしなければならない。)

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:>%JRCSERVER_HOME%\bin\jrccl.cmd user passwd
JRcClientDriver version XXX YYYY/MM/DD.
OK.
>servername@rcadd hello “rcmd test.CmpHelloWorld”
>quit
```

このように、マーキング部分の入力情報の[rcadd]コマンドを使って、JRcCNS に対して登録することで、上記コマンド名 [hello]が、対象コンポーネント名に割り当てられる。

ここで、名前を付与するときの先頭にセットしている[rcmd]とは、リモートコマンド実行を意味する予約コマンドで、このコマンドが対象コンポーネントを実行する役割をもっており、通 常は上記のように設定することで、[hello]と言うものに対して、実行コマンドとすることが出来る。そのため、この名前付けをうまく利用して１つのコンポーネントに対して、別の価値を求める複数のコマンドを作成することが出来る。
例えば、以下のように、第一引数を固定とするコマンドを作ることも出来る。

servername@rcadd hello2 "hello world?? "

このようにすれば、[hello2]は、[hello world??] コマンドを呼び出した場合と、同じ意味になる。また、スペースを入れると、区切り条件になるので、スペースを含む文字列を１つの文字列として、認識させたい場合は、コーテーション('や ")で囲う必要がある。

これで、ＪＲｃコンテナに対して、test.CmpHelloWorld が登録された。そこで、正しく登録が行われているか確かめるため、以下のことを実行する。確認方法として、ＪＲｃＣlientＤriverを使って、登録コマンド[hello]を直接実行してみる。（この場合は、別にＲＯＯＴユーザでなくとも、かまわない。）

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:\JRcServer>%JRCSERVER_HOME%\bin\jrccl.cmd user passwd
JRcClientDriver version XXX YYYY/MM/DD.
OK.
>servername@hello
hello
>servername@hello a b c
hello=3
```

上記のように、マーキング部分の入力情報の実行結果として、登録しているコンポーネントの実行結果が返されている。今回登録した実行コンポーネントは、引数が存在しない場合は、[hello]と返し、存在する場合は、[hello=引数の数]を返すので、登録されていることが、確認できる。

今度は、JRcClientを用いて、プログラム内から、コンポーネント実行を行う。

といっても、先ほど行った、JRcClientDriverから、[hello]コマンドを実行したことと、同じような感じで、プログラム実装するので、それほど難しいものではない。また、一見すると、ＪＤＢＣのような感じで、プログラム実装を行える。

まずは、下記のソースコードを見てほしい。

TestJRcClient.java

```
01:import com.JRcServer.lib.* ;
02:
03:public class TestJRcClient
04:{
05:        public static void main( String[] args ){
06:                JRcConnection conn = null ;
07:                String ret = null
08:
09:                try{
10:                        conn = JRcClientDriver.get( “user”,”passwd” ) ;
11:                        ret = conn.execution( “servername@hello” ) ;
12:                        System.out.println( “[CMD]:\”hello\”  “ + ret ) ;
13:                        ret = conn.execution( “servername@hello a b c” ) ;
14:                        System.out.println( “[CMD]:\”hello a b c\”  “ + ret ) ;
15:                }catch( Exception e ){
16:                        e.printStackTrace() ;
17:                }
18:        }
19:}
```

このソースコード[TestJRcClient.java]をコンパイルして、実行すると、以下の結果が、テキストコンソールに表示される。

[CMD]: "hello " hello
[CMD]: "hello a b c " hello=3

このように、[TestJRcClient.java]プログラムコードを見れば分かるように、JRcClientは、JDBCのStatementと良く似た利用方法であることから、JDBCを使った事がある人であれば、違和感なく、JRcコンテナから、対象コンポーネントを実行するプログラムを実装することができる。

まとめると、JRcServerをコーディングレベルから見た場合、以下の利点と、欠点がある。

[利点]

・プログラムの実装においては、「ネットワーク」を意識しなくてもよい。何故なら、「付録１：JRcServer構成図」の JRcGateway[Client]が、グループ内に存在するJRcServerの名前解決を自動で行ってくれるからである。
・JRcコンポーネントを呼び出す場合、プロセス間通信であるか、リモートマシン上であるかを、考慮しなくても良い。何故なら、「付録１：JRcServer構成図」のJRcClientは、接続先のJRcGateway[Client]の位置を、外部定義で管理しているからである。
・分散コンポーネントが、main メソッドに似ているものを採用しているので、Javaプログラム経験者であれば、容易に実装することができる。
・JRcClientは、JDBCのような使い方ができるので、JDBCを利用したことのある、JAVAプログラマであれば、容易に実装することができる。
・コンポーネントの実行方法が「コマンドライン」形式であるため、テキストコンソール(Windows では、コマンドプロンプト)からも、プログラム内からも、差がなく実行できる(つまり、CORBAや、ブリッジなど、別途細工をする必要がない。)
・通常、ORB形式のものは、専用コンパイラ（ここでは、DeployTool が該当する）から、分散環境の手続きを示すプログラムを、自動生成しなければならないが、JRcServerは、このようなことは必要とせず、普通のプログラムと同じ工程で、実行形式を作成できる。

[欠点]

・コンポーネントに渡される引数は、文字列なので、数値などで利用する場合、変換をする必要がある。
・コンポーネントからの戻り値は、文字列であるため、数値などで利用する場合、変換する必要がある。
・新しいコンポーネントをＪＲｃコンテナに登録するには、ＪＲｃＣｌｉｅｎｔ経由で、ＪＲｃＣＮＳに登録する作業工程が、必要となる。
・マルチキャストを使っているので、JRcServerを直接インターネット上から、利用することは出来ない。
(ただし、『付録１：JRcServer構成図」の JRcClientとJRcGateway[Ｃｌｉｅｎｔ]の通信プロトコルは、ＴＣＰ／IPなので、
これをインターネット上から、利用することは可能である。)
・現在の JRcServerには、実績がない(JRcServerStandard版は、完成していないから)。

２．２．３．ＪＲｃＳｅｒｖｅｒとＥＪＢをコーディングレベルから見て比較する。

これまでに、JRcServerと、ＥＪＢを、コーディングレベルからみた内容により、利点や欠点を表してきた。今回は、その内容を比較して、既存のＥＪＢよりも、JRcServerが如何に、簡単で、便利になったかを示す。

初めに、既存のＥＪBの欠点を、JRcServerがどのように克服しているかを説明する。

[EJB]

・EJB を使いこなすためには、J2EE 自体を、ある程度理解しなければならず、そのことが、EJB を学ぶ人達に対して、敷居が高いものとなっている。

[JRcServer]

・分散コンポーネントが、main メソッドに似ているものを採用しているので、Javaプログラム経験者であれば、容易に実装することができる。

[EJB]

・EJBは「重量級のコンテナ」であることから、コンポーネントの実装や、呼び出しに多くの手続きを要する。

[JRcServer]

・JRcClientは、JDBCのような使い方ができるので、ＪＤＢＣを利用したことのある、ＪＡＶＡプログラマであれば、容易に実装することができる。

[EJB]

・プロセス間通信で利用するのか、リモート上で利用するのかを考慮しなければならず、その状況に応じて、ＥＪＢコンテナを呼び出す側は、実装方法が変わってしまう。このため、これらの関係を変更する場合は、呼び出し側の実装個所を全て変更しなければならず、拡張性が悪い。

[JRcServer]

・ＪＲｃコンポーネントを呼び出す場合、プロセス間通信であるか、リモートマシン上であるかを、考慮しなくても良い。何故なら、「付録１：JRcServer構成図」の JRcClientは、接続先の JRcGateway[Client]の位置を、外部定義で管理しているからである。

[EJB]

・EJBコンテナをリモート上で利用する場合、IP アドレス(ドメイン名)＋ポート番号で、接続先を決めるので、移植性が損なわれてしまう。

[JRcServer]

・プログラムの実装においては、「ネットワーク」を意識しなくてもよい。何故なら、「付録１：JRcServer構成図」のJRcGateway[Client]が、グループ内に存在するJRcServerの名前解決を自動で行ってくれるからである。

[EJB]

・ＥＪＢコンテナを利用する場合、同じＪ２ＥＥのＷｅｂコンテナ（Servletや、JSP）から利用する事を前提としているため、プログラム言語は、JAVA言語を想定している。そのため、ＥＪＢコンテナは、他のプログラム言語からは、単純には利用できないので、再利用率が低いと言える（CORBAや、ブリッジからだと、他言語から利用できなくもないが、これはこれで、面倒な作業のため、「単純」ではない）。

[JRcServer]

・コンポーネントの実行方法が「コマンドライン」形式であるため、テキストコンソール(Windows では、コマンドプロンプト)からも、プログラム内からも、差がなく実行できる(つまり、CORBAや、ブリッジなど、別途細工をする必要がない。)

次に、既存のＥＪＢに対して、ＪＲｃＳｅrverが、利点を求めた結果、出てきた問題は以下の通りである。

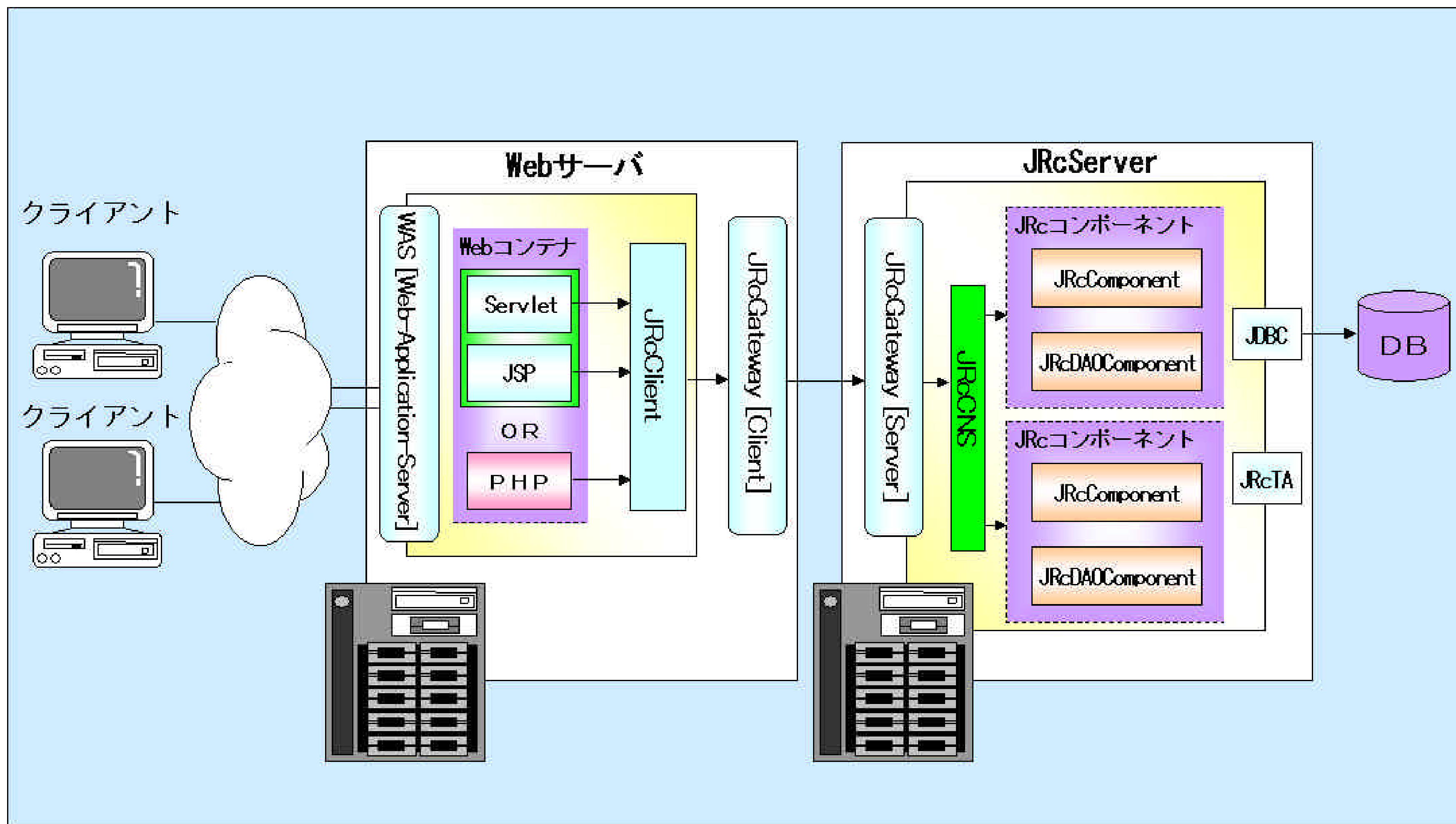
・コンポーネントに渡される引数は、文字列なので、数値などで利用する場合、変換をする必要がある。
・コンポーネントからの戻り値は、文字列であるため、数値などで利用する場合、変換する必要がある。
・新しいコンポーネントをＪＲｃコンテナに登録するには、ＪＲｃClient経由で、ＪＲｃＣＮＳに登録する作業工程が、必要となる。
・マルチキャストを使っているので、JRcServerを直接インターネット上から、利用することは出来ない。
(ただし、「付録１：JRcServer構成図」のJRcClientとJRcGateway[Client]の通信プロトコルは、ＴＣＰ／IPなので、これをインターネット上から、利用することは可能である。)

３．最後に

この文章で記述されている内容は、Ｓtandard版での、内容であることから、現在[2005/08/30]のＰrototype版とは、全然違ったものであると言えます。そのため、現在のバージョン　a020 版は、この内容と違うことから「コマンドライン形式の**分散環境だけ**を試したい」と思う人は、このバージョンで体験してください。

また、Ｓtandard版以降の展望は、簡単ではありますが、『付録２．ロードマップ』を参照してください。

付録１．ＪＲｃＳｅｒｖｅｒ構成図



付録２．ロードマップ