The JFox Project

J2EE Application Server Implementation Project

JFox User Guide
V2.0

(HDR-100001-JFoxUserGuide-P-FR-1.0)

Close Time: 2004-09-13

Specification Lead

Yang Yong

Expert Group

JFox PMC

Comments, please email to: webmaster@jfox.cn

The JFox Project

J2EE Application Server Implementation Project

历史记录

日期	版本	描述	作者
2004-07-24	V0.7	**文档结构建立**	Peter Cheng
2004-08-26	V1.0	**文章结构整理**	Yang Yong, Peter Cheng，
2004-9-14	V2.0	2.0 **修正**	YangYong

The JFox Project

J2EE Application Server Implementation Project

目录

The JFox Project

J2EE Application Server Implementation Project

1.概览

JFox(http://www.huihoo.org/JFox) 是基于 J2EE 的应用服务器，遵循 ejb2.0 规范，为了简化 EJB 的开发，提高开发和部署的效率，在表达形式上也做了一些改进，我们的目标是在规范和开发效率之间找到一个平衡，既尽可能遵循规范，又尽可能改变 ejb 开发缓慢、布署麻烦的现状，切实提供一个快速的 J2EE 中间件平台。

2.产品特性

1. 符合 ejb2.0 SessionBean 规范
   暂不支持 EntityBean，推荐采用 SessionBean + DAO/OR 来进行数据持久化
2. 采用增强的 IOC 内核，并结合 JMX 的优势，支持 Web 组建管理
3. 完全支持 JTA 1.01b 规范，支持两阶段提交(2pc)
4. 支持多种数据源，MySQL、SQLServer、Oracle、DB2
5. Remote 和 Local 调用自动切换，同一个应用服务器上的 ejb 之间的调用自动使用 Local 调用，无需实现 Local 接口
6. 采用动态代理调用框架，无需 ejb 预编译
7. 自动发布，拷贝即可发布
   有新的 EJB 要发布的时候，只要将该 ejb 的 jar 包拷贝到%JFOX_HOME%/deploy 目录下即可，自动发布器会即时将该 ejb 布署到应用服务器中
8. 使用 Meta 作为发布描述文件
   采用 Meta 方式进行智能化布署，使得布署不再需要繁琐的步骤和不同工种之间的协作，布署的描述由程序员通过编程完成。
9. 协议后决的调用方式
   协议后决的调用方式使得在调用过程可以动态改变调用协议，通过 jndi.lookup 得到的 Home 存根是一个与协议无关的存根，当通过该 Home 存根进行调用的时候，才会选择一种合适的协议，当前对于同 JVM 之间的调用自动选择为 Local 调用，对远程调用，则自动选择 JRMP 协议，也可以通过 Home.setProtocol 来强行制定同 JVM 采用 JRMP 协议进行调用
10. 采用优化的调用链模型，提高服务端执行速度
11. 通过 XML－RPC 支持 Web Service
12. 集成 Jetty Web Container
13. 支持 EAR 部署

3.系统安装

1. 安装 JDK，建议 1.4 以上版本，并设置%JAVA_HOME%环境变量；
2. 从 http://sourceforge.net/projects/JFox 下载 JFox 最新版本，解压至某个目录，假定为%JFOX_HOME%
3. 在%JFOX_HOME%/bin 下，运行 startup.bat 启动 JFox
4. 连接 http://localhost:8080，可以访问 JFox 的 Web 界面，在这里可以执行 JFox 提供的例子
5. 连接 http://localhost:8082，可以看到正在运行的系统组件，并可以进行管理
6. 连接 http://localhost:8080/jpetstore，可以运行 PetStore 的演示

4.应用开发

4.1 第一个 EJB

假定我们要开发一个叫做 Hello 的 Stateless Session Bean，步骤如下，具体代码参考 %JFOX_HOME%/TestEJB

1. 定义 Remote 接口，TestStateless.java
2. 定义 Home 接口，TestStatelessHome.java
3. 实现 Hello EJB，HelloBean.java
4. 制作 ejb-jar.xml 文件
5. 可选择实现一个 Meta 类，名称必须为 TestStatelessMeta.java
6. 使用 jar 打包，拷贝至%JFOX_HOME%deploy 下，发布完成

在这些步骤中，前 4 步都和经典的 ejb 开发相同，第五步，是在标准允许的情况下，对 EJB 部署的扩展，但是与一般的采用一个特殊的 xml 文件（如：jboss 采用 jboss-ejb-jar.xml）来描述不同，JFox 采用一个 Meta 类来描述 JFox 特定的部署信息，这样而简化 EJB 的开发，Meta 类可以通过继承 org.jfox.ejb.meta.EJBMetaSupport 来实现，如果没有提供 Meta 类，JFox 采用默认的特定部署信息，比如：JNDI Name 为 ejb/EJB_NAME，远程调用协议为 JRMP，本地调用采用 LOCAL 协议。

在这个例子中，我们先忽略 Meta 类，采用默认的配置，Meta 类的编写在后面的章节中详细描述。

按照 EJB 的规范，编写好 ejb-jar.xml，下面的部分从 %JFOX_HOME%/TestEJB/META-INF/ejb-jar.xml 文件中截取。

```
<?xml version="1.0"?>
```

```
<session>
    <display-name>Test Stateless SessionBean</display-name>
    <ejb-name>TestStatelessEJB</ejb-name>
    <home>org.jfox.examples.ejb.stateless.TestStatelessHome</home>
    <remote>org.jfox.examples.ejb.stateless.TestStateless</remote>
    <ejb-class>org.jfox.examples.ejb.stateless.TestStatelessBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
</session>
```

完成之后，编译，并打成一个 jar **包，并将** ejb-jar.xml **放置到** jar **包中的** META-INF **下，这样** ejb **组件就完成了，将包拷贝到**%JFOX_HOME%deploy **下即会自动发布，**JNDI Name **为** ejb/TestStatelessEJB**，**TestStatelessEJB **即是** ejb-jar.xml **中指定的该** ejb **的名称，这个名称必须是唯一的。**

TestStatelessEJB **的客户端的详细代码可以看** %JFOX_HOME%/TestEJB/src/org/jfox/examples/ejb/stateless/TestMain.java**，下面是通过** JNDI **获得生成** Hello **对象的代码，注意的还是** JNDI **的名称** ejb/ TestStatelessEJB

```
Context ctx = new InitialContext();
Object obj = ctx.lookup("/ejb/ TestStatelessEJB");
home = (HelloHome)javax.rmi.PortableRemoteObject.narrow(obj,HelloHome.class);
hello = home.create();
```

4.2 EJBMeta **的定制**

JFox **采用** EJBMeta **类来描述** JFox **特有的部署信息，而不是常见的一个扩展的** xml **来描述，这个类是可选的，如果采用默认的部署描述，可以不提供该类，如果要提供** Meta **类，该类的名称必须为**EJB_CLASS_NAMEMeta**。** EJBMeta **要完成最重要的** 3 **个描述信息是：远程调用协议、本地调用协议、**EJB **的** JNDI **名称。默认情况（没有提供）下，远程调用协议为** JRMP**，本地调用协议为** LOCAL**，**JNDI Name **为** ejb/EJB_NAME**，**EJB_NAME**是在** ejb-jar.xml **中指定的该** ejb **的名称。如果你希望远程调用采用其他的协议，比如：**JRMP_SSL**，**SOAP **等，或者本地调用不希望采用** LOCAL **协议，而采用** JRMP **等远程协议，或者希望采用指定的** JNDI **名称，则需要提供** Meta **类。**

该类的内容可以参考：

%JFOX_HOME%/TestEJB/src/org/jfox/examples/ejb/stateless/TestStatlessMeta.java**，下面是节选的内容：**

```
public class TestStatelessMeta extends EJBMetaSupport {

    public TestStatelessMeta(EJBDescriptor ejbDescriptor) {
        super(ejbDescriptor);
```

```
        }

        public Protocol getRemoteProtocol() {
            return Protocol.JRMP_SSL;
        }

        public Protocol getLocalProtocol() {
            return super.getLocalProtocol();
        }

        public String getJndiName() {
            return super.getJndiName();
        }

    }
```

它的内容很简单，可以直接从 EJBMetaSupport **继承，然后覆盖必要的方法。**

要注意的是，getRemoteProtocol() **方法不能返回** Protocol.LOCAL **协议，理由是很明显的，**LOCAL **协议只能用在同一虚拟机中对** EJB **的调用。**

4.3 事务处理

为了保证 JFox **的服务端执行效率和业务安全，**JFox **实现了一个性能优异的事务处理器，通过和** jboss **进行对比测试，** JFox tm **提供了更好的性能，详情参见：** %JFOX_HOME%/docs/JFoxtm-test.doc

我们推荐你使用容器管理的事务，JFox **支持** EJB **规范定义的所有事务属性。**

4.4 数据源

JFox **数据源支持所有提供** XA **规范驱动程序的数据库，现在绝大多数数据库都已提供** XA **驱动，包括** Microsoft SQL Server**，**SQL Server **的** jdbc **驱动可以在** Microsoft **的网站上下载到。**

要添加一个数据源，可以在%JFOX_HOME%/modules/jdbc/META-INF/module.xml **中增加一个** component**，下面的内容定义了一个不支持** XA **事务以及一个支持** XA **事务的数据源**

```
<?xml version="1.0" encoding="gb2312" ?>
<module>
    <description>JDBC Module</description>
     <priority>1</priority>
```

```
<component>
    <description>Pool DataSource Service(No Transaction)</description>

    <implementation>org.jfox.jdbc.datasource.PoolDataSourceService</implementation>
    <factor>default</factor>

    <singleton>true</singleton>
    <type-singleton>false</type-singleton>
    <constructor>
        <param>  <!-- datasource name-->
            <type>java.lang.String</type>
            <value>TestMysqlDataSource</value>
        </param>
        <param>  <!-- driver class name -->
            <type>java.lang.String</type>
            <value>com.mysql.jdbc.Driver</value>
        </param>
        <param>  <!-- url -->
            <type>java.lang.String</type>
            <value>jdbc:mysql://localhost/test</value>
        </param>
        <param>  <!-- user -->
            <type>java.lang.String</type>
            <value>root</value>
        </param>
        <param>  <!-- password -->
            <type>java.lang.String</type>
            <value></value>
        </param>
    </constructor>
</component>

<component>
    <description>Pool XADataSource Service(Transaction)</description>

    <implementation>org.jfox.jdbc.xa.TxDataSourceService</implementation>
    <factor>default</factor>

    <singleton>true</singleton>
    <type-singleton>false</type-singleton>
    <constructor>
```

```
            <param>  <!-- datasource name-->
                <type>java.lang.String</type>
                <value>TestXAMysqlDataSource</value>
             </param>
            <param>  <!-- url -->
                <type>java.lang.String</type>
                <value>jdbc:mysql://localhost/test</value>
             </param>
            <param>  <!-- user -->
                <type>java.lang.String</type>
                <value>root</value>
             </param>
            <param>  <!-- password -->
                <type>java.lang.String</type>
                <value></value>
             </param>
        </constructor>
    </component>
```

</module>**这将创立一个名为** TestMysqlDataSource **的** Mysql **数据源和一个** TestXAMysqlDataSource **的** XA Mysql **数据源，数据源的名称不仅用来唯一的标志一个数据源，也将用在** jndi **的名称，**%DS_NAME%**的数据源的** jndi **名称为**/%DS_NAME%**，比如上面配置的** DataSource **将可以通过** ctx.lookup("/TestMysqlDataSource") **和** ctx.lookup("/TestXAMysqlDataSource")**得到。**

对于不同的数据源，URL **的写法不一样，如下：**

mysql: jdbc:mysql://localhost/test

Oracle: jdbc:oracle:thin:@localhost:1521:yang

DB2: jdbc:db2://localhost;databaseName=SAMPLE

MSSQL Server: (**必须设置** selectMethod =cursor)

jdbc:microsoft:sqlserver://localhost:1433; selectMethod =cursor;databaseName=db

注意：要使** MSSQL Server **支持** XA JDBC **驱动，需要在服务器上运行一段存储过程，具体操作请从** Microsoft **网站下载** MSSQL Server JDBC **驱动后阅读其文档。

4.5 传输协议

JFox **目前支持** LOCAL、JRMP、JRMP_SSL、SOAP **协议，可以见** org.jfox.ejb.meta.Protocal JFox **使用了协议后决的调用方式，在获得** EJB Home **的时候，并没有给接下来的调用绑定协议，直到** create EJB **对象的时候，才绑定一个确切的协议，我们叫这个为协议后决的调用方式，这种调用方式可以让我们自由的选择和切换** ejb **调用的协议。默认情况下，远程客户**

调用应用服务器中的 EJB 组件，将使用 JRMP 协议，而同一个应用服务器中间的 EJB 之间的调用，将使用之间的 Java 调用方式（LOCAL 调用），提高执行速度。但是在某些特定的情况下，比如希望 EJB 之间的调用也采用 JRMP 进行远程调用（虽然这种情况很少），或者希望采用 JRMP_SSL 来实现对 EJB 的安全调用，或者希望采用 SOAP 调用协议来访问防火墙之后的应用，这个时候，就需要强行转换调用协议。

转换调用协议有两种方法：

1．**通过定义 Meta 类来改变 ejb 的默认访问协议，这在上面的 EJBMeta 一节以及讲到。**

2. **在程序中动态的切换协议，看下面的代码节选，完整的代码见：**

%JFOX_HOME%/TestEJB/src/org/jfox/examples/ejb/protocol/TestProtocolBean.java

```
public class TestProtocolBean implements SessionBean {
 public  TestProtocolBean() {
  }

public void setSessionContext(SessionContext sessionContext) throws EJBException {
  }

 public  void  ejbRemove() throws EJBException {
  }

public void ejbActivate() throws EJBException {
  }

public void ejbPassivate() throws EJBException {
  }

public void ejbCreate() throws CreateException {
  }

 public String invokeByJRMP() {
       try {
           TestStatelessHome  home  =  getTestStatelessOnePhase();
            ((ExtendedEJBHome)  home).useProtocol(Protocol.JRMP);
           TestStateless  statelessEjb  =  home.create();
           String  voice  =  statelessEjb.getVoice();
            return voice;

        }
       catch(Exception  e)  {
            e.printStackTrace();
           throw  new  EJBException(e);
        }
```

```
    }

    public String invokeByJRMP_SSL() {
        try {
            TestStatelessHome home = getTestStatelessOnePhase();
             ((ExtendedEJBHome) home).useProtocol(Protocol.JRMP_SSL);
            TestStateless statelessEjb = home.create();
            String voice = statelessEjb.getVoice();
             return voice;

        }
        catch(Exception e) {
             e.printStackTrace();
            throw new EJBException(e);
        }

    }

    public String invokeBySOAP() {
        try {
            TestStatelessHome home = getTestStatelessTwoPhase();
             ((ExtendedEJBHome) home).useProtocol(Protocol.SOAP);
            TestStateless statelessEjb = home.create();
            String voice = statelessEjb.getVoice();
             return voice;

        }
        catch(Exception e) {
             e.printStackTrace();
            throw new EJBException(e);
        }

    }

    public String invokeByLOCAL() {
        try {
            TestStatelessHome home = getTestStatelessTwoPhase();
             ((ExtendedEJBHome) home).useProtocol(Protocol.LOCAL);
            TestStateless statelessEjb = home.create();
            String voice = statelessEjb.getVoice();
             return voice;
```

```
            }
            catch(Exception e) {
                e.printStackTrace();
                throw new EJBException(e);
            }

        }

        private static TestStatelessHome getTestStatelessOnePhase() throws Exception {
            Context ctx = new InitialContext();
            Object home = ctx.lookup("java:comp/env/ejb/TestStatelessEJB");
            TestStatelessHome statelessHome = (TestStatelessHome)
        javax.rmi.PortableRemoteObject.narrow(home, TestStatelessHome.class);
            return statelessHome;
        }

        private static TestStatelessHome getTestStatelessTwoPhase() throws Exception {
            Context initCtx = new InitialContext();
            Context ctx = (Context) initCtx.lookup("java:comp/env");
            Object home = ctx.lookup("ejb/TestStatelessEJB");
            TestStatelessHome statelessHome = (TestStatelessHome)
        javax.rmi.PortableRemoteObject.narrow(home, TestStatelessHome.class);
            return statelessHome;
        }
    }
```

ExtendedEJBHome.useProtocol() **支持的有效参数在** Protocol **类中定义，现在有** JRMP **和** LOCAL，JRMP_SSL，SOAP **协议不区分大小写。因为** JFox **已经自动选择了最佳的调用协议，且强行转换调用协议超出了** EJB **规范的内容，所以我们希望您一般情况下不要这样做。**

4.6 Web Container

JFox **集成** Jetty **做为** Web Container，Jetty **是知名的、开放的、被广泛使用的** Web **服务器。** JFox **支持** web.xml **的** ejb-ref resource-ref resource-env-ref env-entry **等环境引用功能。**

5.应用部署

JFox 支持动态部署 EAR、JAR、WAR 模块，部署目录为%JFOX_HOME%/deploy，只要将上述三类型的模块拷贝至该目录，JFox 会自动发现并部署。

deploy 目录下已经有几个默认的 ear 包和 war 供用户参考,其中 xmlrpc.war 提供对 SOAP 协议的支持，如果用户需要使用 SOAP 协议，请不要删除该包。

5.1 EJB JAR 部署

JAR 包的部署由 JFox 新生一个 classloader 直接装载。

5.2 WAR 部署

JFox 首先将 war 包解压至于%JFOX_HOME/temp/webapps，然后交给 jetty 部署该目录，JFox classloader 是该 web module 的 parent classloader。

注意： *1.如果你的 Servlet 采用默认的访问路径，那么该 Servlet 包必须由 jetty 的 classloader 来加载，也就是应该放到 WAR 的 lib 或者 classes 目录下，不能放到 JFox classloader 的 class path 中。*

2.%JFOX_HOME%/webapps 目录用来以目录结构直接部署，在启动 JFox 是，WebContainer 将加载该目录下的 web 应用，该目录下的 war 文件并不会自动部署

5.3 EAR 部署

EAR 的部署涵盖的 JAR 部署和 WAR 部署。

JFox 首先该 ear 包解压到%JFOX_HOME/temp/ear_file_name下，然后重新组织目录结构，将 ejb jar 包都拷贝到解压后目录的 apps 下，war 包解压到 webapps 目录下，公共的 lib(由 ear 包中的 MANIFEST.MF 的 CLASS-PATH 来指定)移动到 lib 目录下，这样重新组织目录结构是为了让用户很方面就可以看到部署了哪些 ejb jar,web war 和使用了哪些 lib。然后 JFox 会生成了一个 classloader，这些 classloader 装载 lib 下面的类，然后状态 ejb jar，最后 jetty 使用该 classloader 作为 parent classloader 新生一个 web classloader 来部署 webapps 下面的 web 模块,这样 web 中的 servlet 应用和 ejb 可以共享 lib 下的库。部署器也会分析 web 模块的 web.xml 文件，绑定其中定义的 resource-ref resource-env-ref ejb-ref env-entry 等资源引用信息。

更多的内容可以参考 %JFOX_HOME/examples 的代码，也可以参考 %JFOX_HOME/deploy 下的包。

6.运行环境

本章将详细介绍 JFox 系统的运行环境。主要包含硬件主机的配置和支撑软件的安装需

求。

- **硬件配置**

  服务器：PC Server **或** SUN Ultra 10

  CPU：PIII 500 **或与其相当的处理器**

  内存：256M **以上**

  硬盘：1GB **以上**

- **支撑软件**

  操作系统：Linux, Windows 98, 2000, XP, Solaris8 或 Solaris2.6

6.参考

1. http://www.gnu.org
2. http://www.eclipse.org
3. http://www.opensource.org
4. http://www.onjava.com
5. http://www.apache.org
6. http://www.objectweb.org
7. http://www.jcp.org