## Why EJB?

* 最重要的理由：分布式！分布式对象之间互相协作，完成特定的业务功能。分布式对象之间应实现分布透明性（或位置透明性）。即在客户端代码中无需指定分布式对象的位置（通过配置来解决）。
* 分布式对象之间的事务支持（RMI不支持事务）
* 应用程序需要支持不同的客户端，只需要少量的代码，就可以让远程客户端访问到企业Bean。
* 如果你的应用需要进行分布式部署、如果你的分布式对象需要支持事务、如果你的关键业务逻辑需要能够支持多种不同类型的客户端，那么EJB将会是一个很好的选择。

## 什么叫会话（Session）？

* 即从客户端获得EJB对象开始，然后调用EJB的方法（可以多次），直到客户端生命周期结束，或客户端释放了EJB对象为止，称为一次会话
* 随着会话的终止，EJB对象也有可能会被EJB容器销毁

### 什么叫有状态的会话Bean？（Stateful Session Bean）

* 即EJB能够为同一个客户端在多次请求（方法调用）之间保持状态信息。
* 比如：基于某种原因（可能是因为系统要支持多种不同类型的客户端），购物篮功能作为系统的核心业务，需要由EJB对象来担当（而不是由HTTP SESSION对象）。那么EJB必须能够区分不同的客户端，并分别为不同的客户端保持与其对应的状态信息。
* 即，从某个客户端的角度来看，似乎EJB对象正被它独占了一样，不会因为有任何其它的客户端同时对同一个EJB访问而影响其最终的计算结果。

### 什么叫无状态的会话Bean？(Stateless Session Bean)?

* 并非是说EJB不能存在状态，而是说EJB容器不会对EJB的状态做管理
* 容器会使用实例池的方式，甚至单例的方式来实现无状态的Session Bean。
* 因为EJB容器不会对Stateless Session Bean的状态进行管理，所以它的性能要比Stateful Session Bean的好

由Spring管理的业务逻辑类，是无状态的（Singleton）

|  |
| --- |
| <bean id="aclManager" class="com.bjsxt.oa.manager.impl.AclManagerImpl">  <property name="sessionFactory" ref="sessionFactory"></property>  </bean>    <bean id="moduleManager" class="com.bjsxt.oa.manager.impl.ModuleManagerImpl">  <property name="sessionFactory" ref="sessionFactory"></property>  </bean> |

Struts2中的Action，是有状态的（prototype）

|  |
| --- |
| <bean id="companyCustomerAction" class="com.bjsxt.crm.web.struts2.actions.CompanyCustomerAction" scope="prototype">  <property name="customerManager" ref="customerManager"></property>  <property name="dataDictionaryManager" ref="dataDictionaryManager"></property>  </bean>  <bean id="personalCustomerAction" class="com.bjsxt.crm.web.struts2.actions.PersonalCustomerAction" scope="prototype">  <property name="customerManager" ref="customerManager"></property>  <property name="dataDictionaryManager" ref="dataDictionaryManager"></property>  </bean> |

## WEB容器是靠什么知道请求是否是由同一个session来发出？

靠jsessionid

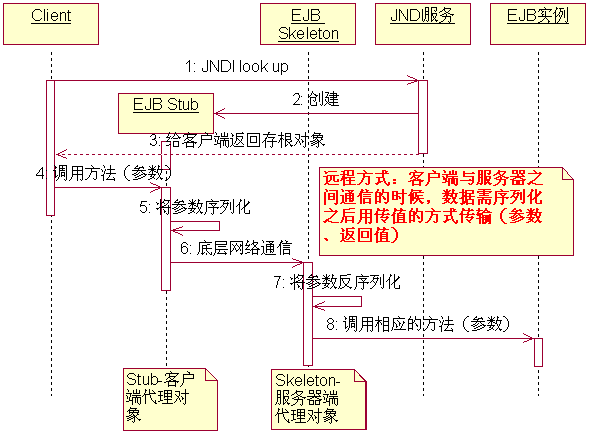
## 定义客户端访问接口

* 客户端仅通过业务逻辑Bean所定义的接口来访问业务逻辑（跟具体实现无关）。
* 当在开发企业Bean之前，首先要做的第一个决定是：EJB要支持什么样的客户端？因为不同的客户端，其访问方式也将不同。
  + 远程客户端
  + 本地客户端
  + WebService客户端
* 关于JVM进程的问题：
  + 假设在同一台机器上，安装了一个JVM，一个JBOSS应用服务器
  + 1、启动JBOSS应用服务器（启动JBOSS当然要用到JVM进程）
  + 2、另外编写一个包含main方法的JAVA类，并编译以及运行（要运行这个程序，当然也要用到JVM进程）
  + 问题：在运行这个JAVA类的时刻，这台机器上有多少个JVM进程？

### 远程客户端（Remote Client）

* 一个远程客户端具有如下特性：
  + 它可以运行在跟企业Bean不同的机器或不同的JVM进程上
  + 它可以是web组件、应用客户端、或其它的企业Bean
  + 对远程客户端来说，企业Bean的位置是透明的
  + 为了创建一个能够被远程客户端访问的企业Bean，你必须用@Remote注解来定义这些企业Bean。

### 本地客户端(Local Client)

* 一个本地客户端具有如下特性：
  + **它必须与企业Bean运行在同一个JVM中**
  + 它可能是一个web组件或别的企业Bean
  + 为了创建一个能够被本地访问的企业Bean，你必须用@Local注解来定义这些企业Bean。
  + 一个企业Bean接口可以同时被定义为@Remote和@Local

### 远程访问(调用)方式(Remote Access)

### 本地访问方式(Local Access)

## Web service客户端

Web service客户端可以访问**无状态会话Bean**的接口，只有在业务逻辑方法被标识为@WebMethod的时候，web service客户端才可以访问到

## 方法的参数和访问方式

不同的访问方式（Remote、Local、web service），会影响到Bean方法的参数及其返回值。

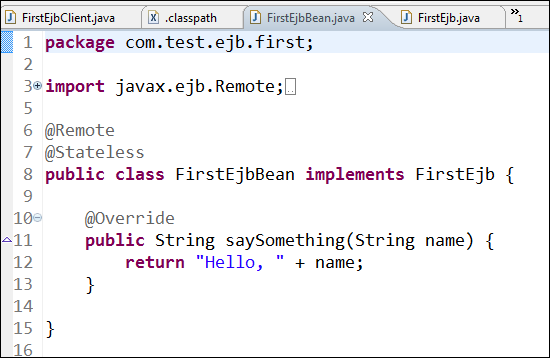
***独立性***

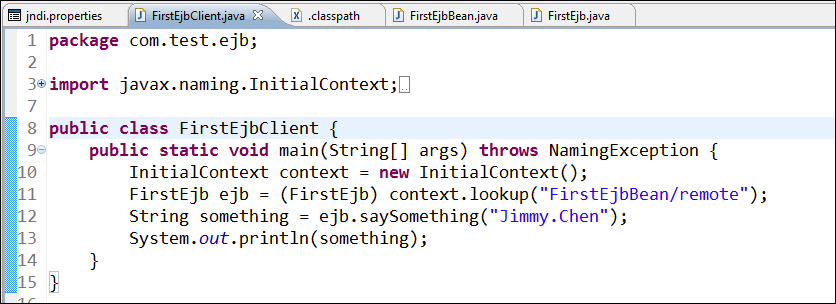
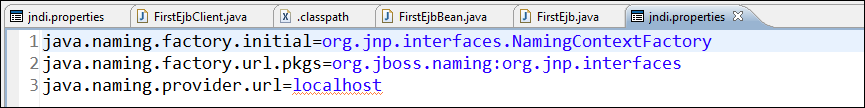
如果是远程调用，客户端操纵的Bean的参数，其实是一份参数值的拷贝。因此，对参数的修改，不会影响到Bean。但是对于本地调用来说，客户端操纵的Bean的参数，就是一个引用，它对参数的修改，将会影响到Bean。所以，**不管在哪种情况下，请避免修改参数的值**！

***粗粒度的数据访问***

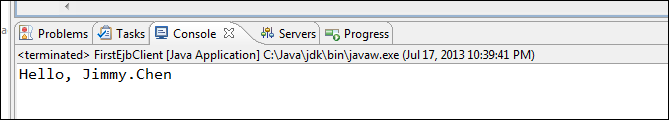
因为远程调用的速度比较慢，所以在设计的时候，请尽量使用粗粒度的接口设计。即尽量减少方法的调用，并尽可能在一次方法调用中传输完毕所需要的数据！

# 第一个EJB实例(Hello world)

1. 新建一个**EJB工程**，并创建接口
2. 实现这个接口，并标记为Remote和Stateless
3. 将工程1部署到JBOSS中。
4. 新建一个**java工程**，并将上面工程的FirstEjb接口导出为jar包（只需要导出这个接口就可以了），并引用到这个java工程中。
5. 将JBOSS中的client目录下的所有jar包引用到工程中。
6. 创建一个包含main方法的类，并调用该EJB
7. 在java工程的类路径下创建文件jndi.properties

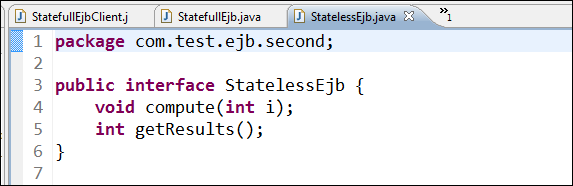


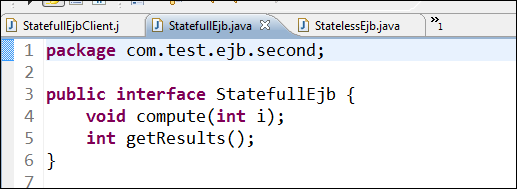
成功输出：



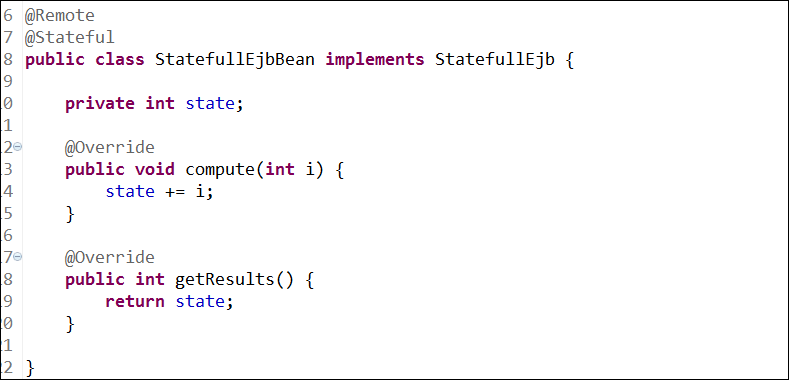
# 第二个EJB实例（The differences between stateful EJB and stateless EJB）

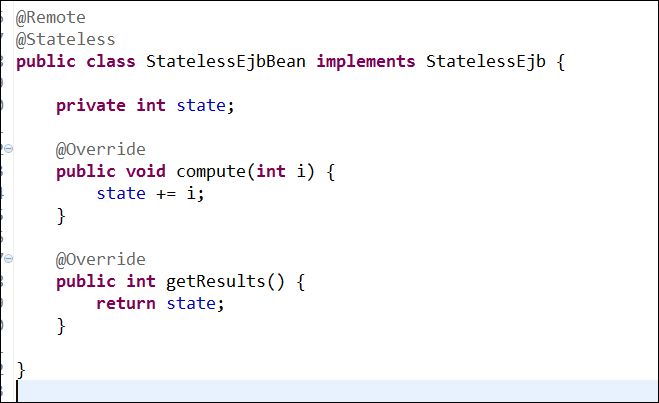
1. 创建EJB接口，方法完全一样



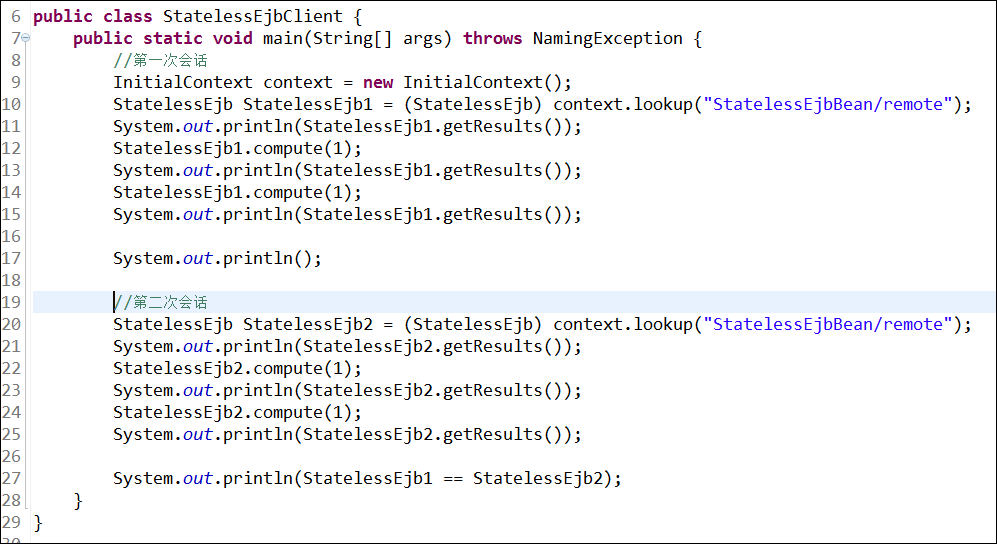


1. 创建实现类，并标识为Remote和Stateful/Stateless

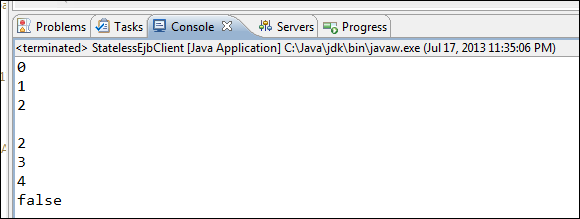




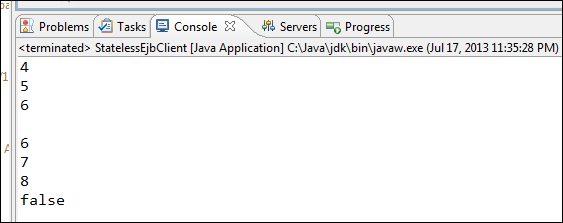
1. 在客户端工程（JAVA工程）中创建这Stateless EJB的客户端代码



输出为：

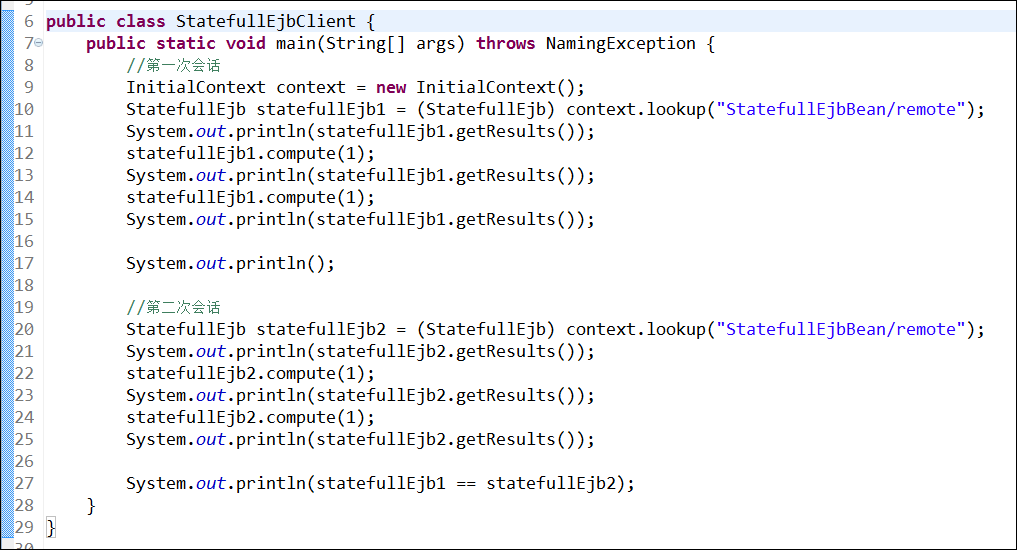


再次运行的结果如下：

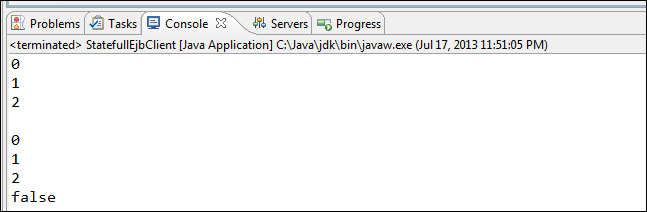


从上面的输出可以看出，服务器端会在无论是多次会话间，还是不同的JVM进程请求之间都保留信Sateless的EJB的状态，但即使是这样，客户端lookup出来的EJB对象都是不同的对象，因为它们其实是EJB的代理对象。

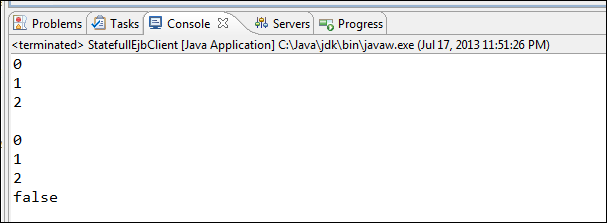
1. 创建Stateful EJB的客户端代码(与stateless的一致)



运行输出为：



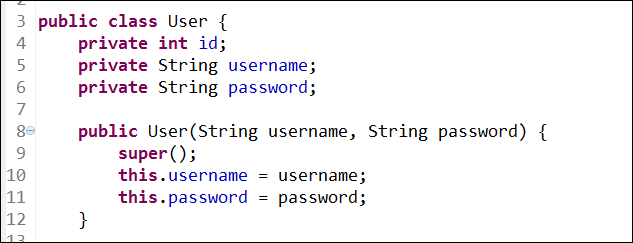
再次运行的输出依然为：



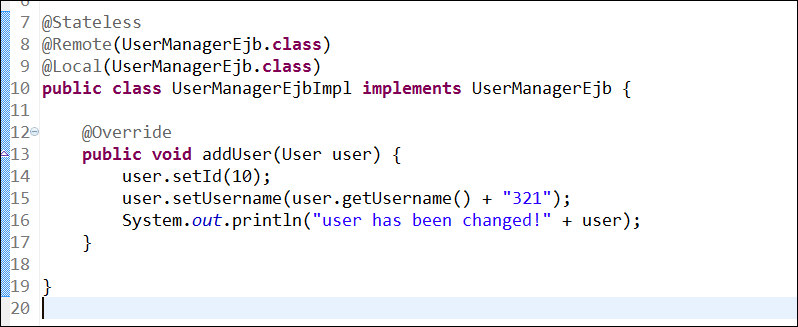
从上面的输出可以看出，有状态的EJB会被服务器保留当前这次会话的信息，而在多次会话或者多次不同的请求之前，它们的状态不会保留。

# 第三个EJB例子（远程/本地调用时的方法参数传递方式的区别）

建立User.java



创建EJB的接口并实现它：

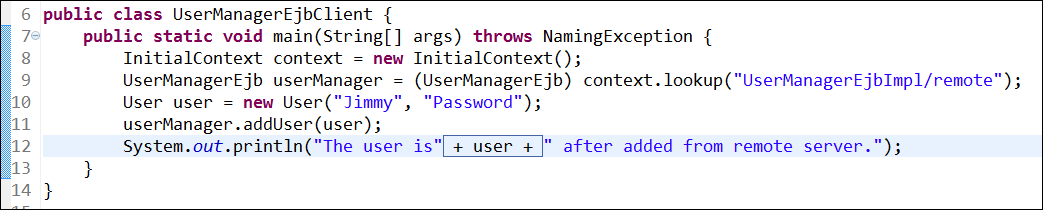


注：

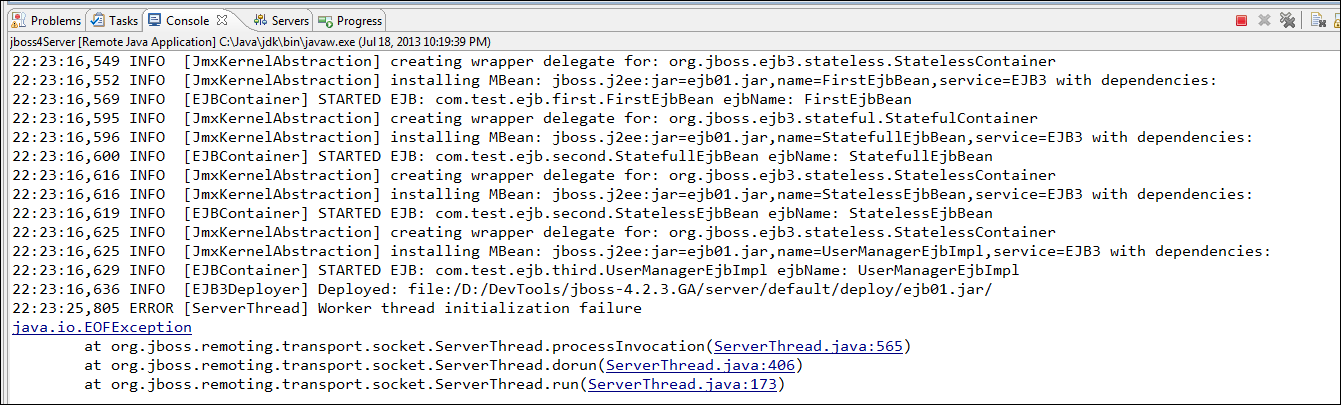
1. 把该EJB**同时标记为remote和local**，我们将会同时用远程的本地的方式来调用它来看它们之间的方法参数传递方式的区别
2. 在addUser方法里手动设置ID为10，模拟ID自动递增程序，并只修改它的username
3. 上面的Remote和Local注解传递了接口的参数，这是因为JBOSS4中，如果同一个EJB被同时标识为远程和本地EJB，那么就必须这样来写，否则会报以下错误：

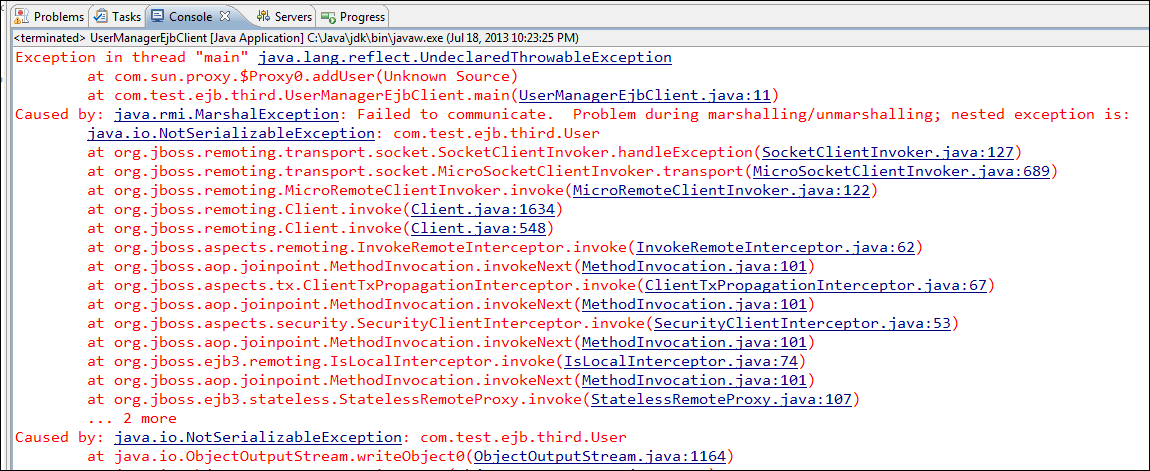
**javax.ejb.EJBException: Local and Remote Interfaces cannot have duplicate interface for bean XXXXX**

创建客户端调用程序：



这时会在服务器端和客户端同时报错：

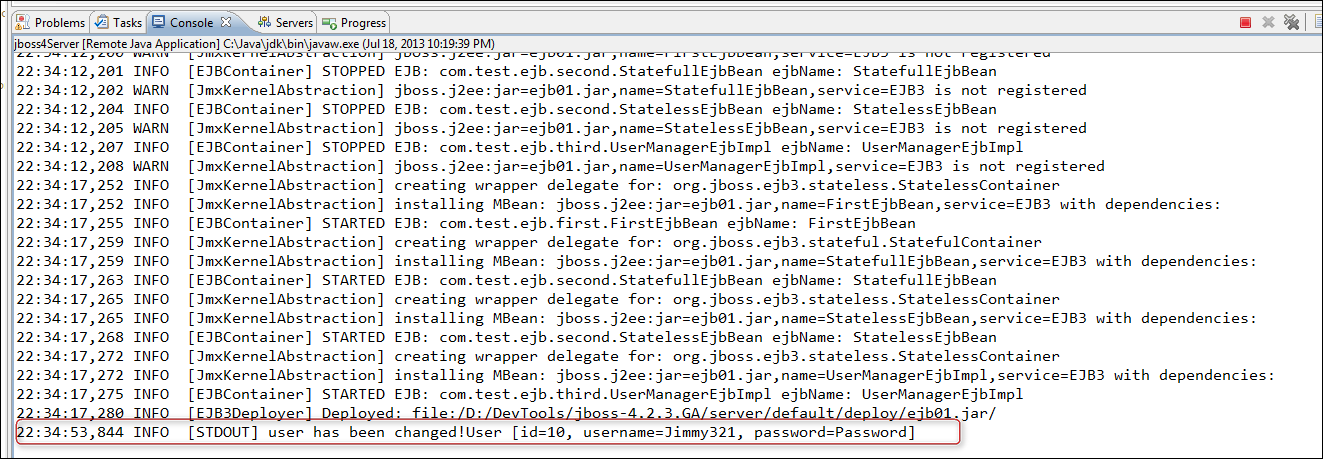




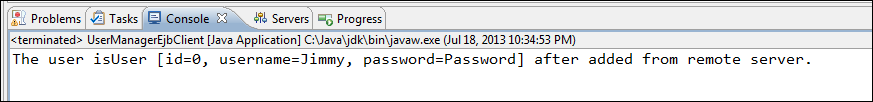
从这个错误可以看出，remote调用EJB时的方法参数类型如果是复杂类型，那么必须实现Serializable接口。

修改User类，这时就可以看到正确的输出了：

服务器端的输出为：

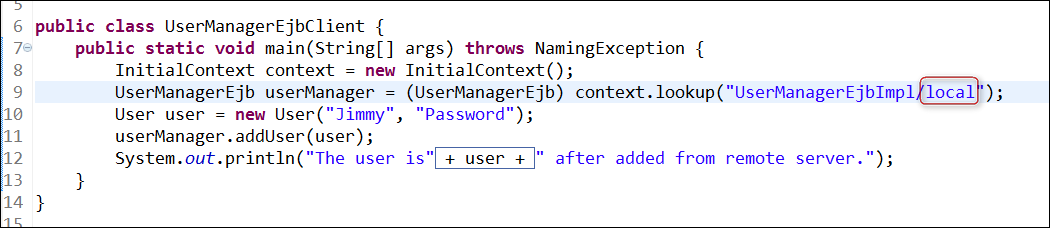


客户端为：

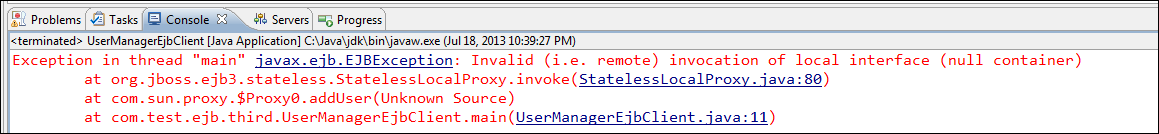


从上面的输出可以看出，remote方式的调用时，对于方法上的参数传递方式为值传递，不会对客户端的参数产生影响（调用后的user对象id和username都没有改变）。

将客户端的访问程序改为本地调用的方式：

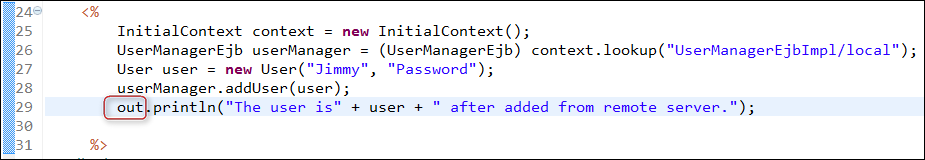


这时会在客户端得出如下错误(服务器端没有错误)

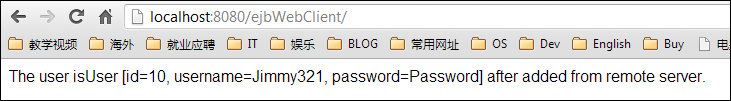


原因为对于本地调用方式，必须保证是在同一个JVM中发生，但在我们的程序里，很明显这是两个不同的JVM，所以会报错。

我们可以使用WEB工程并把它部署在与EJB在同一个容器里来做这个实验：



Copy上面客户端的代码到JSP中，导入相应的包，并将System.out改为out.可得到以下输出：



可以看出，在EJB本地调用中，方法参数的传递类型为地址传递，从而可以将修改后的参数值反馈到本地调用的客户端程序中。并且对于本地调用，并不要求参数的类型必须实现Serializable接口，如果不实现这个接口，上面也会照样做以上的输出。

注：对于这个web工程，我们不需要提供jndi.properties文件，因为这是一个本地调用，这些信息在JBOSS都存在。另外不需要再将jboss/client下的jar包加载到web工程中，这些也是在JBOSS中存在的东西。其实也不可以需要将ejb工程引用到web工程，因为你在部署web工程的时候也把EJB的工程部署上去了，并且他们都是在同一个JVM下。之所以加上是为了在编写代码的时候避免编译错误。

# 将EJB发布为WebService

只有stateless的EJB才可以被发布为WebService

WebService的参数和返回值等不是通过序列化来传输的，而是将对象的内容以XML的形式输出，到server端时再将它读出来的，所以用于WebService的对象可以不需要序列化

# Trouble shooting

## 启动时报JVMJ9VM039I的错误

在websphere的JDK配置时添加-Xshareclasses:none参数但貌似还是没有用。

Clear websphere在myeclipse中的配置，再重新设置后无缘无故的就好了。