**笔记**

# 安装Eclipse

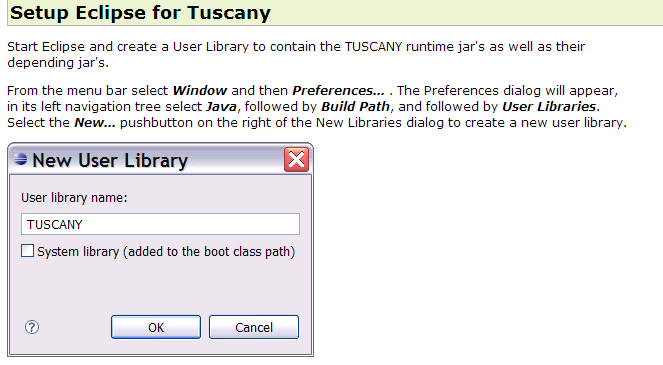
根据书《Open SOA》采用软件eclipse-java-ganymede-SR2-win32（3.4版本），要实现SCA的设计，需要安装SOA插件，可参照第“9”个问题，进行安装，之后就可以SCA设计了。

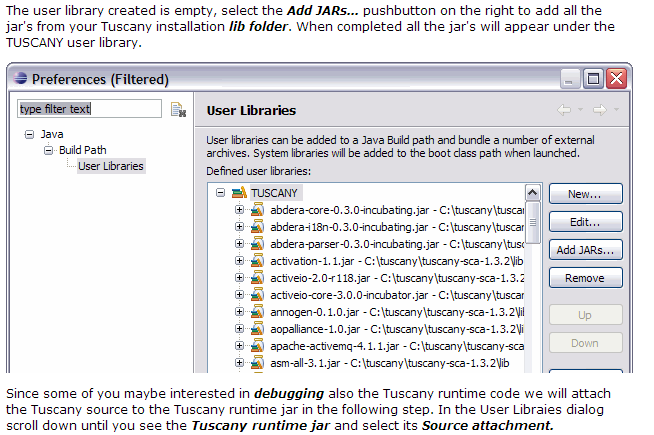
# SCA （apache-tuscany-sca-1.4）

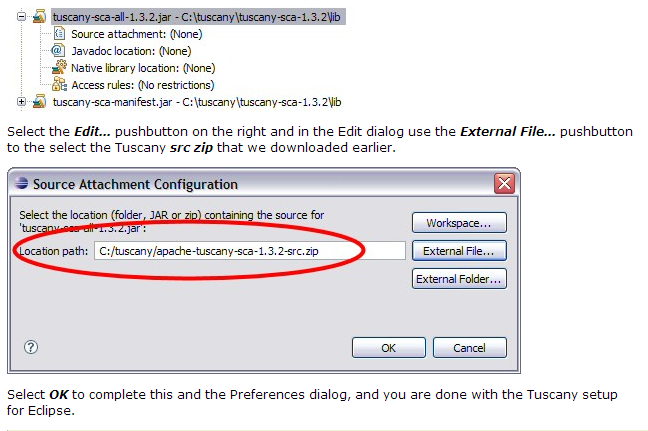
环境：JDK1.7（JDK1.8缺少某些包，例：java.util.Map下没有Entry<String, Item>）

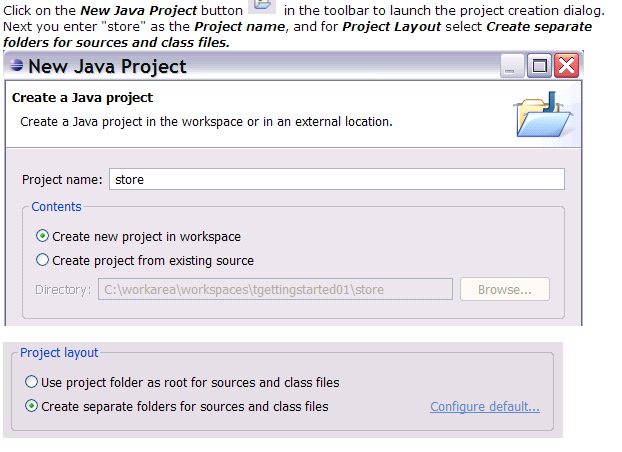
## 将apache-tuscany-sca-1.4包引入Eclipse中有两种方法：

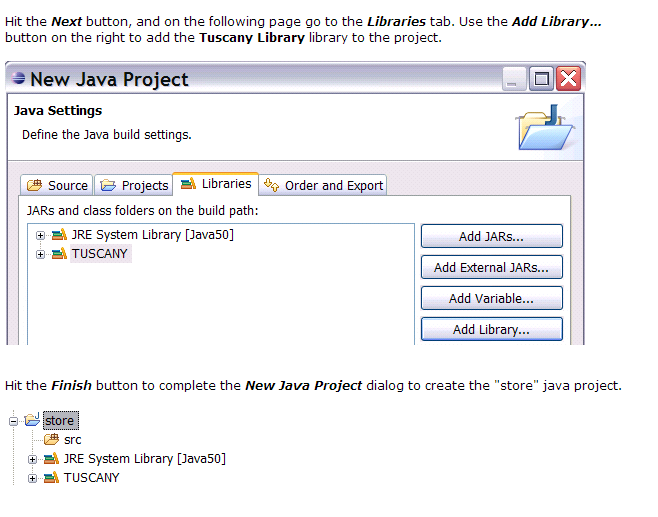
第一、（http://tuscany.apache.org/getting-started-with-tuscany.html）



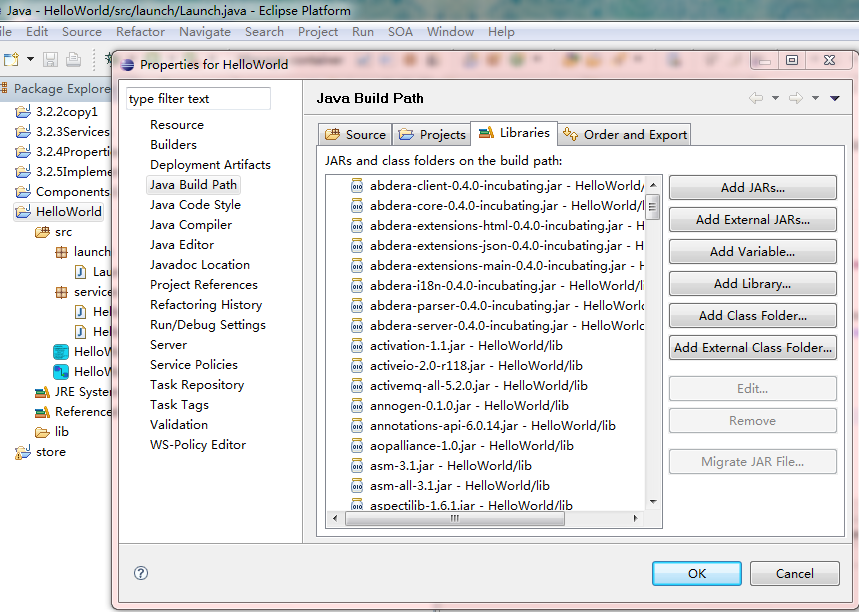






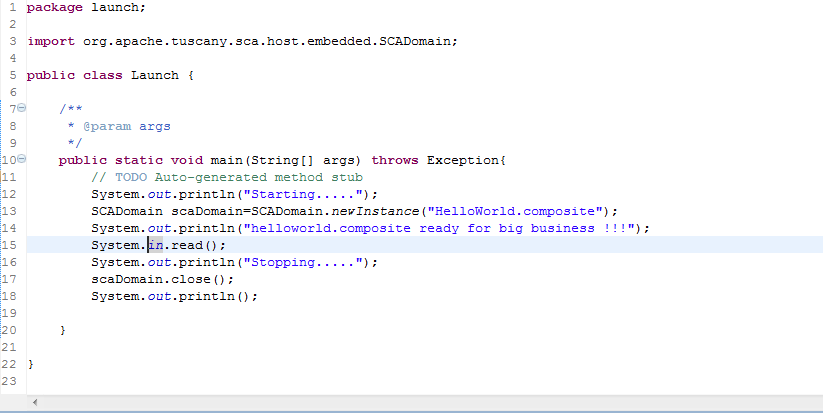


第二、在创建工程后，将包中的“lib”文件夹复制到与“src”同级（这样就不用管包路径的问题）。右键工程名，选【Build path】【configure build path】，在【Java Build Path】中的【Libraries】可以添加包。（当然也可在此处删掉不用的包）



## 创建SCA工程

1. 首先将apache-tuscany-sca-1.4的内部包引用到工程中。
2. 在src下创建“服务包”（package），再在其下一级创建“Java接口”（@Remotable，可进行远程调用）；
3. 然后，创建实现该接口的类；
4. 右击【src】，选【new】【File】（名字为“XXX.composite”），或者【other】中选【SCA tools】。在该文件中编写XML文件，引入“服务”和“组件”。可以右击【XXX.composite】，选择【initialize……..】看到可视化的组件连接。
5. 编写完毕后，创建一个启动服务器的静态main类，可命名为“launch”，调用SCADomain类。



1. 最后启动即可。

## 服务的不同方式

1. 直接Web Servise

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"  xmlns:s="http://HelloWorld"  targetNamespace="http://HelloWorld"  name="HelloWorld">    <service name="HelloWorld"  promote="HelloWorld">  <interface.java  interface="services.HelloWorld" />  <binding.ws uri="http://localhost:8080/HelloWorld" />  </service>    <component name="HelloWorld">  <implementation.java  class="services.HelloWorldImpl" />  </component>    </composite> |

1. HTTP（Java Script）

采用JavaScript实现可视化的网页发布服务，如“store”工程：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"  xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"  xmlns:s="http://store"  targetNamespace="http://store"  name="store">    <component name="store">  <t:implementation.widget location="ufservices/store.html" />  <service name="Widget">  <t:binding.http uri="http://localhost:8080/store" />  </service>  <reference name="catalog" target="Catalog">  <t:binding.jsonrpc />  </reference>  <reference name="shoppingCart" target="ShoppingCart/Cart">  <t:binding.atom />  </reference>  <reference name="shoppingTotal" target="ShoppingCart/Total">  <t:binding.jsonrpc />  </reference>  </component>  <component name="Catalog">  <implementation.java class="services.CatalogImpl" />  <property name="currencyCode">USD</property>  <service name="Catalog">  <t:binding.jsonrpc uri="http://localhost:8080/Catalog"/>  </service>  <reference name="currencyConverter" target="CurrencyConverter" />  </component>  <component name="ShoppingCart">  <implementation.java class="services.ShoppingCartImpl" />  <service name="Cart">  <t:binding.atom uri="http://localhost:8080/ShoppingCart/Cart" />  </service>  <service name="Total">  <t:binding.jsonrpc uri="http://localhost:8080/Total"/>  </service>  </component>  <component name="CurrencyConverter">  <implementation.java class="services.CurrencyConverterImpl" />  </component>  </composite> |

1. JAVA客户端

# JDBC(MySQL)

先将myaql-connector-java-5.1.34-bin.jar（在安装的文件夹中）引入到工程中。

1. 加载驱动程序

|  |
| --- |
| Class.forName("com.mysql.jdbc.Driver"); |

1. 建立连接

|  |
| --- |
| Connection connection  =DriverManager.getConnection(dbURL,dbUser,dbPwd); |

其中，dbURL="jdbc:mysql://localhost/test";dbUser="root";dbPwd="1234";

1. 创建语句

|  |
| --- |
| Statement statement=connection.createStatement(); |

1. 执行语句

|  |
| --- |
| statement.executeUpdate("insert into Student(studentName,idNumber,score) values('"+studentName+"','"+idNumber+"',"+score+");"); |

查询：ResultSet resultset=statement.executeQuery("select \* from Student where idNumber"+"='"+idNumber+"';");

1. 处理结果ResultSet

|  |
| --- |
| **while**(resultset.next()){  System.out.println(resultset.getString(1)+"\t"+resultset.getString(2)+"\t"+resultset.getString(3));  } |

最后，

|  |
| --- |
| connection.close(); |

这种是直接采用JAVA语言执行SQL语句，不方便，一般会中间会采用Hibernate。

## 查询数据库

在上边，查询语句后，设想将数据库中得到的数据进行对象化，如：

|  |
| --- |
| stuQuery.setstudentName(resultset.getString(1));  stuQuery.setidNumber(resultset.getString(2));  stuQuery.setscore(resultset.getDouble(3)); |

但总是报错，换成输出字符串就好了。不知道为什么。

**解决方法：为使其输出并List化**

采用getMetaData()函数，将数据保存在一个String数组中（当然可以是二维数组）

|  |
| --- |
| ResultSetMetaData md=resultset.getMetaData();//得到的是数据库元数据的信息  resultset.getRow();//得到查询结果的条目数  String[] stuQuery=**new** String [md.getColumnCount()];  **while**(resultset.next()){  //System.out.println(resultset.getString(1)+"\t"+resultset.getString(2)+"\t"+resultset.getString(3));\  //System.out.println(md.getColumnCount());  **for**(**int** i=0;i<md.getColumnCount();i++){  stuQuery[i]=resultset.getString(i+1);  }  } |

# 服务的引用是通过setter方法注入的

在组件之间进行引用时，必须将被引用者通过setter方法注入，完成实例化。如：在composite文件中：

|  |
| --- |
| <reference name="createStudent" target="CreateStudent"/>  <reference name="queryStudent" target="QueryStudent"/> |

对应于引用者的具体实现为：

|  |
| --- |
| **private** CreateStudent createstudent;  **private** QueryStudent querystudent;  **实例化：**  @Reference  **public** **void** setCreateStudent(  CreateStudent createstudent) {  **this**.createstudent = createstudent;  }  @Reference  **public** **void** setQueryStudent(  QueryStudent querystudent) {  **this**.querystudent=querystudent;  } |

并且还得分别写setter函数，可能在引用的时候，会用到对应对象的setter函数。

3、当一个服务有多个方法，其中有两个方法要求传入的参数一样时，在运行时，由于网址是一样的，怎么区分不同的请求呢？

# 关于数组初始化问题

在Java中，定义数组时，最好进行初始化操作，以免返回null而出错。如：

|  |
| --- |
| String[] stuQuery=**new** String [md.getColumnCount()]; |

其中字符串数组中都为null。

当：

|  |
| --- |
| String[] stuQuery=null； |

返回值得时候，可能就会出现异常。

# 关于SCA中上层composite引用下层composite

## SCA上层composite调用下层

其中除了具体内容要按照下边去写，还要**注意**：**定义composite 中属性targetNamespace上下层必须一致。**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"  xmlns:s="http://SchoolSystem"  targetNamespace="http://SchoolSystem"  name="SchoolSystemComposite">    <service name="SchoolSystem"  promote="SchoolSystem">  <interface.java  interface="schoolSystem.SchoolSystem" />  <binding.ws uri="http://localhost:8086/SchoolSystem" />  </service>    <component name="SchoolSystem">  <implementation.java class="schoolSystem.SchoolSystemImpl" />  <reference name="studentSystem" target="StudentSystem"/>  </component>    <component name="StudentSystem">  <implementation.composite name="s:StudentSystemComposite"/>  </component>  </composite> |

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"  xmlns:s="http://SchoolSystem"  targetNamespace="http://SchoolSystem"  name="StudentSystemComposite">    <service name="StudentSystem"  promote="StudentSystem">  <interface.java  interface="studentSystem.StudentSystem" />  </service>    <component name="StudentSystem">  <implementation.java class="studentSystem.StudentSystemImpl" />  <reference name="createStudent" target="CreateStudent"/>  <reference name="queryStudent" target="QueryStudent"/>  </component>    <component name="CreateStudent">  <implementation.java class="studentSystem.CreateStudentImpl"/>  </component>    <component name="QueryStudent">  <implementation.java class="studentSystem.QueryStudentImpl"/>  </component>    </composite> |

## 一个数据库中两个表问题

当一个数据库中有两个表时，并让其关联起来。注意两表的**主键不能一样**。

1. 当同时插入两个表时，要分别建立连接，如StudentSystemSQLs中。
2. 查询两个表的数据：

|  |
| --- |
| ResultSet resultset=statement.executeQuery("select \* from Student as a,StudentInfo as b where a.IdNumber=b.IdNumber"  +" and a.IdNumber='"+idNumber+"';"); |

# 回调

**回调**：双向通信，异步通信。当服务器较长时间去响应请求时，可以采用异步操作，客户端继续执行下面程序。当响应完成后，再回调给客户端。

## 正常步骤：

1. 创建**回调接口**，定义执行**回调时调用的操作**，如将从数据库中取出的数据返回。
2. 创建服务端的**服务接口和实现类**，增加**回调注解**。表明该组件支持回调。
3. 创建或修改客户端类，**实现第一步中定义的回调接口**，**添加到第二步中创建的服务的引用**，修改业务逻辑以使用服务端组件的功能。
4. **修改composite文件**以配置装配集。

如：

1、**回调接口**

|  |
| --- |
| @Remotable  **public** **interface** EventNotificationCallback {  **void** success (**boolean** status);  } |

*注意：回调接口必须声明@Remotable，否则会出现错误。*

2、**服务端支持回调的服务接口**

|  |
| --- |
| @Remotable  @Callback(EventNotificationCallback.**class**)//指定回调，表明该组件支持回调  **public** **interface** EventNotificationComponent {  @OneWay //标记notify操作为非阻塞，告知客户端继续处理其他程序  **public** **void** notify(TicketDO ticket);  } |

3、**服务端支持回调的服务接口的实现**

|  |
| --- |
| @Service(EventNotificationComponent.**class**)  **public** **class** EventNotificationComponentImpl **implements** EventNotificationComponent{  @Callback  **protected** EventNotificationCallback callback;//注入回调类    **public** **void** notify(TicketDO ticket) {  // **TODO** Auto-generated method stub  **try**{  Thread.*sleep*(40000); //休眠40秒，仅为模拟服务  }**catch**(InterruptedException e){  e.printStackTrace();  }  System.*out*.println("hello");  callback.success(**true**); //执行回调操作，具体实现在上一级  }    } |

**实现回调接口**

|  |
| --- |
| **public** **class** CreateTicketComponentImpl **implements**  CreateTicketComponent, EventNotificationCallback{  @Reference  **protected** SessionManager sessionManager;//没有用到setter方法    **private** EventNotificationComponent eventNotifier;//用setter方法，和上者效果相同;该组件支持回调    **public** **int** create(TicketDO ticket) {  //Random r = new Random();  //return r.nextInt(300000);  System.*out*.println("Mr/Mrs "+ticket.getCustomerName()+",your ticket problem has got!");  eventNotifier.notify(ticket);//跳过此步后，继续执行，等其异步返回  **return** sessionManager.getSessionId();    }  **public** **void** success(**boolean** status) { //回调方法实现  // **TODO** Auto-generated method stub  System.*out*.println("Notify results: "+status);  }  @Reference  **public** **void** setEventNotifier(EventNotificationComponent eventNotifier){  **this**.eventNotifier=eventNotifier;  }  } |

4、**配置composite文件**

|  |
| --- |
| <component name="CreateTicketComponent">  <implementation.java  class="service.CreateTicketComponentImpl"/>  <reference name="sessionManager" target="SessionManagerComponent"/>    <reference name="eventNotifier" target="EventNotificationComponent">  <interface.java interface="callBack.EventNotificationComponent" callbackInterface="callBack.EventNotificationCallback"/>  </reference>  </component> |

1. **服务端支持回调的接口必须在composite文件中声明为组件，服务对其进行调用。**
2. **2中服务接口的配置中，**

|  |
| --- |
| @OneWay //标记notify操作为非阻塞，告知客户端继续处理其他程序 |

其为单向服务调用（其他服务调用模式可参见SWIM书），当使用该模式时，函数必须为void，不能又返回值。

当采用@Callback回调模式时，异步回调效果并不明显。

## 问题与解决

第三步，客户端实现回调接口时，将接口EventNotificationCallback添加进后，

|  |
| --- |
| **public** **class** CreateTicketComponentImpl **implements**  CreateTicketComponent, EventNotificationCallback{ |

出现了“警告”

|  |
| --- |
| 警告: Component reference target not found, it might be a remote service running elsewhere in the SCA Domain: Composite = {http://TicketProblem}ProblemManagementComposite Service = CreateTicketComponent |

运行调用时，出现错误：

|  |
| --- |
| 严重: org.osoa.sca.ServiceUnavailableException: Unable to create SCA binding invoker for local target ProblemTicketComponent reference createTicket (bindingURI=CreateTicketComponent operation=create) |

**解决：**将EventNotificationCallback从程序中去掉，即

|  |
| --- |
| **public** **class** CreateTicketComponentImpl **implements**  CreateTicketComponent{ |

然后运行调用**，就成功了！**

**存在疑问：**没有具体声明该类实现回调接口，为什么仍然必须实现会调接口的方法？并且最后还调用成功了？

**猜测**：是不是程序中用到下面程序，调用了支持回调的组件，从而自动默认该类实现回调接口。

|  |
| --- |
| **private** EventNotificationComponent eventNotifier;//用setter方法，和上者效果相同;该组件支持回调 |

**注意：***由于本人是将回调接口与客户端放在不同的包中，后来证明将回调接口与客户端放到一个包中也不行，会出现同样的问题*

## 关于引用setter的问题

如6.2中所示，当组件注入引用时，要用setter函数将服务引用到组件中。但有两种方式：

1. 将引用定义为private类型时

|  |
| --- |
| **private** EventNotificationComponent eventNotifier; |

后边要加setXXX对应函数：

|  |
| --- |
| @Reference  **public** **void** setEventNotifier(EventNotificationComponent eventNotifier){  **this**.eventNotifier=eventNotifier;  } |

1. 将引用定义为protected类型时

|  |
| --- |
| @Reference  **protected** SessionManager sessionManager; |

就不用再添加对应的setXXX函数了。

# Composite问题

## composite文件中引用名“name”必须与Java中引用的变量保持一致

组件之间的引用必须保持：composite文件中引用名“name”必须与Java中引用的变量保持一致，否则会报错。

|  |
| --- |
| <component name="ProblemTicketComponent">  <implementation.java  class="service.ProblemTicketComponentImpl"/>  <reference name="createTicket" target="CreateTicketComponent"/>  <reference name="emailService" target="EmailServiceComponent"/>  </component> |

|  |
| --- |
| **private** CreateTicketComponent createTicket;    **private** EmailServiceComponent emailService; |

当

|  |
| --- |
| <reference name="emailService" target="EmailServiceComponent"/> |

改为：

|  |
| --- |
| <reference name="EmailService" target="EmailServiceComponent"/> |

时，出现以下错误：

|  |
| --- |
| 严重: Reference not found for component reference: Component = ProblemTicketComponent Reference = EmailService  二月 06, 2015 4:59:13 下午 org.apache.tuscany.sca.assembly.builder.impl.CompositeBindingURIBuilderImpl  严重: Reference not found for component reference: Component = ProblemTicketComponent Reference = EmailService  二月 06, 2015 4:59:13 下午 org.apache.tuscany.sca.assembly.builder.impl.ComponentReferenceWireBuilderImpl  警告: No targets for reference: Composite = {http://TicketProblem}ProblemManagementComposite Reference = emailService  Exception in thread "main" org.osoa.sca.ServiceRuntimeException: Reference not found for component reference: Component = ProblemTicketComponent Reference = EmailService |

## 引用的定义通过promote属性注入到实现类的对应成员变量中

如书中3.2.6节描述那样，reference元素可以从component元素定义中移到组件级别（即移除component成为独立的组件），**@promote属性可以表明使用该引用的组件**，其结构为：**<component-name>/<reference-name>**。如：

|  |
| --- |
| <component name="SystemErrorComponent">  <implementation.java  class="issueManagement.SystemErrorComponentImpl" />  </component>  **<reference** name="ProblemTicket"  **promote="SystemErrorComponent/problemTicket">**  <binding.ws uri="http://localhost:8085/ProblemTicketService" />  **</reference>** |

# 脚本语言支持（Ruby）

问题：除了没有加属性，服务发布成功，但调用服务的时候出现异常，还没找到解决方法。

|  |
| --- |
| 严重: org.jruby.exceptions.RaiseException  org.osoa.sca.ServiceRuntimeException: org.jruby.exceptions.RaiseException |

# 产品部署问题

1、按照源代码的步骤，将war文件放到webapps的路径下，未能成功在Tomcat中运行服务。*并且Tomcat与J2EE结合的比较好*。

2、但实现了3.2.6节的“使用Web服务绑定来指定远程引用”，其实质是一样的，只是调用的服务运行的环境不同而已。*注意：被调用的服务要发布出来。*

3、由代码的具体实现，如：

|  |
| --- |
| **import** org.osoa.sca.annotations.Reference;  **import** service.ProblemTicketComponent;  **import** service.TicketDO;  **public** **class** SystemErrorComponentImpl **implements** SystemErrorComponent{  @Reference  **protected** ProblemTicketComponent problemTicket;  **public** **int** systemProblem(String system, String title, String problem) {  // **TODO** Auto-generated method stub  System.*out*.println("\*\*\*SystemErrorComponentImpl\*\*\*");  **int** rval=0;  TicketDO ticket=**new** TicketDO();  ticket.setSubject(title);  ticket.setProblemDesc(problem);  ticket.setSource(system);    rval=problemTicket.createTicket(ticket);  **return** rval;  }  } |

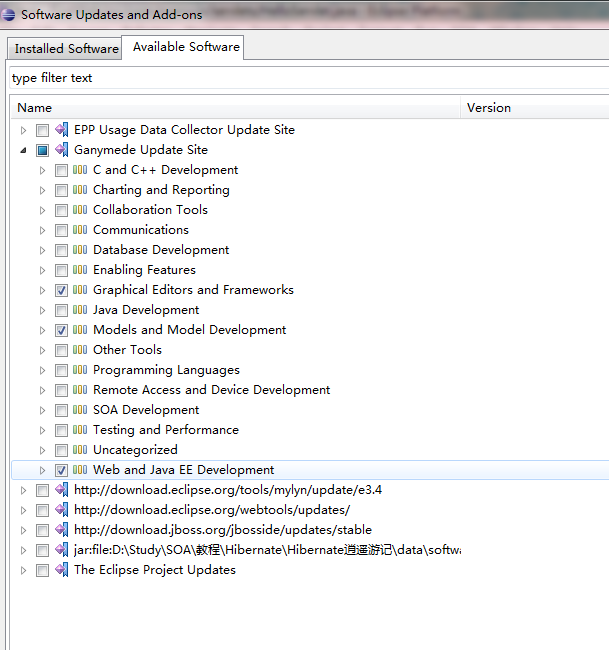
*有代码可以看出，当要调用远程的服务时，必须将远程服务本地化，因为程序中引用了远程服务的类，如程序中阴影部分。*

# Eclipse安装Tomcat和开发web项目

## Eclipse安装Tomcat插件

1、**Tomcat的版本要和Eclipse支持Tomcat的版本一致或兼容**，如本人用的Eclipse为eclipse-java-ganymede-SR2-win32（3.4版本）支持到Tomcat6。经试验，只有将Tomcat6.xx安装才可以，本人采用：apache-tomcat-6.0.43-windows-x86。安装成功，并且启动也成功。（*具体配置可查询网页*）

2、由于本人用的是eclipse-java-ganymede，要开发Web 项目，还需要安装 J2EE 和 Web Development 插件，如下图中，将勾选的进行安装，既可以。



*参考网站：http://www.ibm.com/developerworks/cn/opensource/os-eclipse-tomcat/*

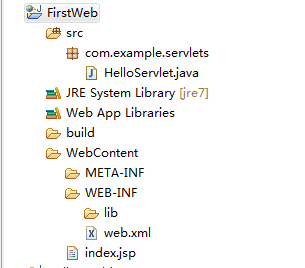
## 创建Web项目问题与解决

按照网站上的说明新建web项目，实现了JSP。

**问题**：项目添加 Java servlets 并加以部署时，出现下面库类找不到的情况。

|  |
| --- |
| **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse; |

**解决：**经查询，需要将Tomcat中jsp-api.jar和servlet-api.jar两包放到项目的lib文件中。



**问题：**将Web项目添加到服务器上，并启动服务器，当在浏览器中输入<http://localhost:8080/FirstWeb/index.jsp>时，出现以下错误：

|  |
| --- |
| 严重: Servlet.service() for servlet jsp threw exception  java.lang.NullPointerException  at org.apache.jsp.index\_jsp.\_jspInit(index\_jsp.java:22)  at org.apache.jasper.runtime.HttpJspBase.init(HttpJspBase.java:52)  at org.apache.jasper.servlet.JspServletWrapper.getServlet(JspServletWrapper.java:164)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

**解决：**将lib文件中的jsp-api.jar包去掉，再启动，运行成功了。

**总结**：Web项目开发JSP，可以直接添加到服务器，启动即可。而Servlet要将servlet-api.jar添加到lib文件夹中。（*其实JSP运行的时候，是先转化为Servlet，然后再运行*）输入<http://localhost:8080/FirstWeb/HelloServlet>可以访问Servlet程序。

# SDO问题

## SDO处理复杂的XML结构

之前SOAP使用的结构都是扁平的，或者说所使用的数据类（如TickectDO类）是一个扁平的数据结构，用于表示传入的问题报告单。但当SOAP为更加反映真实世界场景的复杂XML数据结构使，就没有那么简单了（如书上108页的示例）。

|  |
| --- |
| <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:iss="http://chapter4.book.opensoa/issue">  <soapenv:Header/>  <soapenv:Body>  <iss:createTicket>  <iss:Problem severity="low" category="systems" subcategory="os">  <iss:Header>  <iss:From>user</iss:From>  <iss:Subject>test subject</iss:Subject>  <iss:To>test to</iss:To>  <iss:CreationDate>2008-02-17T00:00:00.000-07:00</iss:CreationDate>  </iss:Header>  <iss:Description>test description</iss:Description>  </iss:Problem>  </iss:createTicket>  </soapenv:Body>  </soapenv:Envelope> |

而不是简单的：

|  |
| --- |
| <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:chap="http://chapter441.book.opensoa/">  <soapenv:Header/>  <soapenv:Body>  <chap:systemProblem>  <arg0>test arg1</arg0>  <arg1>test arg2</arg1>  <arg2>test arg3</arg2>  </chap:systemProblem>  </soapenv:Body>  </soapenv:Envelope> |

若知道对应的WSDL文件或者对应的XSD文件，可以通过SDO来实现服务：

1. 采用Apache-Tuscany-sdo包中XSD2JavaGenerator将WSDL文件的XML结构定义生产JAVA类
2. 创建新的服务接口，以及使用新的SDO生成类的服务实现

*注：事先得知道XML对应的XSD或WSDL文件*

## 服务发布过程和调用远程WEB服务

服务——>WSDL文件——>（经SoapUI）客户端的XML（可以作为用户接口）

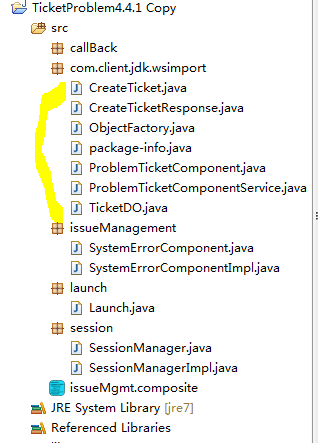
调用远程WEB服务，也要将WSDL文件转化为Java类，即进行本地化操作。

# 远程调用，分布式（或产品）部署

## 远程调用Web Services步骤（以TicketProblem4.4.1 Copy为例）

1. 获取远程服务的WSDL地址（或文件），如：<http://192.168.2.12:8085/ProblemTicketService?wsdl>
2. 用JDK（1.6版本以上）自带的wsimport.exe工具，将WSDL2Java，代码如下：wsimport -s E: -p com.client.jdk.wsimport <http://192.168.2.12:8085/ProblemTicketService?wsdl>（也可以是WSDL文件所在的文件夹中。其中，“-s E:”表示生成的Java类的输出在E盘，具体文件夹为“com/client/jdk/wsimport”）
3. Xjc命令可以将XSD等文件转为Java，如x**jc** **–d**  java类的存放路径 **–p** 类的包名  xsd文件名。

**注意**：由其对远程服务本地化时，本地化的类与WSDL有关，最好将WSDL文件放到工程中后，再转化，便于移植。

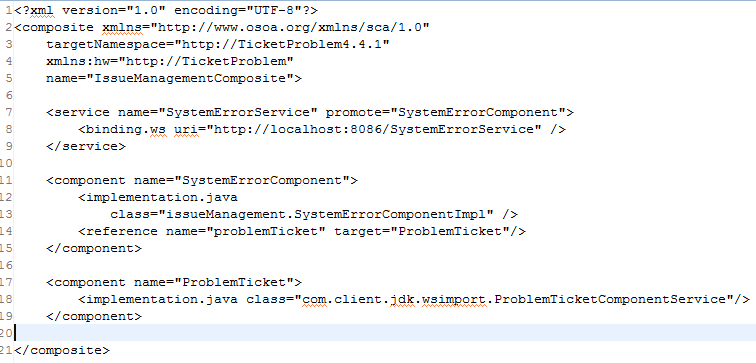


1. 将生成的Java类放入工程中，视为本地类直接调用
2. 具体的调用过程（Java类）和相关配置（.composite文件）如下图：

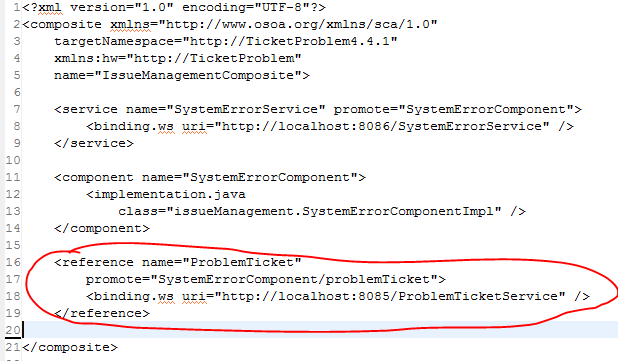
**Java类调用过程（只用到服务和对应组件的方法）：**



**.composite文件：**



而不能为（因为下图中的调用是对远程进行的调用，而上述方法已转为本地）：



（在接口类中声明为@Remotable的接口，是对远程调用来说的，而不是本地）

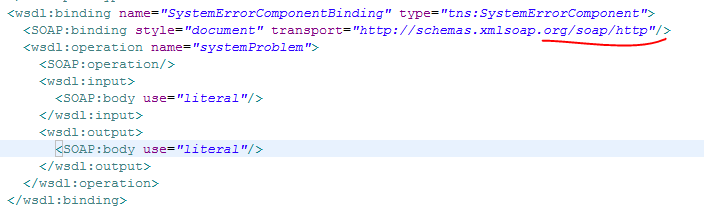
可参考: <http://www.shuyangyang.com.cn/jishuliangongfang/Javabiancheng/2014-05-05/211.html>

<http://blog.csdn.net/yaerfeng/article/details/8366212>

注：用axis2自带的WSDL2Java.bat生成的Java类在SCA里边会报错，不适用。（然后，去查找XSD2JavaGenerator→《Tuscany SCA in Action》中查到jdk有自带的wsimport.exe工具进行转换→调用）

# SCA组件绑定JMS协议（SOAP over JMS）

## 创建JMS服务步骤（以TicketProblem4.4.1 JMS为例）

1. 将原有组件装配以Web Services的形式及SOAP over HTTP进行发布（即一般的WEB服务），得到WSDL文件。然后将WSDL文件放入工程中。
2. 对WSDL文件进行修改，使其支持JMS协议。原始的WSDL文件只有下图中支持http的协议：

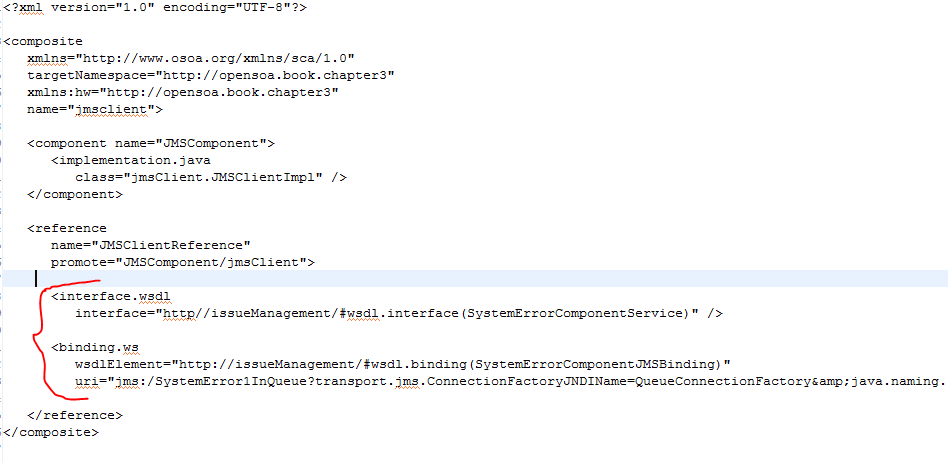
其他不做任何改变，只需要增加支持JMS协议的部分。如下图：



1. 配置.composite文件，修改服务提供端（即步骤1中发布服务的一端）的绑定，使其符合JMS协议，如下图（issueMgmt.composite文件）：其中，详细绑定信息为（对其结构介绍见14.2）：

|  |
| --- |
| <binding.ws wsdlElement="http://issueManagement/#wsdl.binding(SystemErrorComponentJMSBinding)" uri="jms:/SystemError1InQueue?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&amp;java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&amp;java.naming.provider.url=tcp://localhost:61616?jms.watchTopicAdvisories=false" /> |

1. 配置JMS Client的.composite文件，如下图（jmsclient.composite文件）：

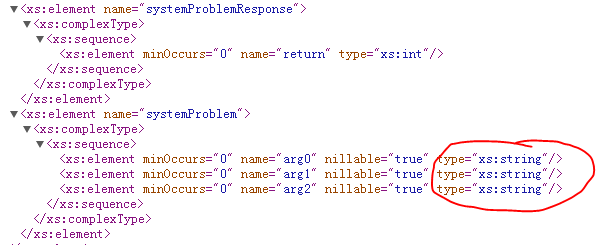


其中，详细绑定信息为：

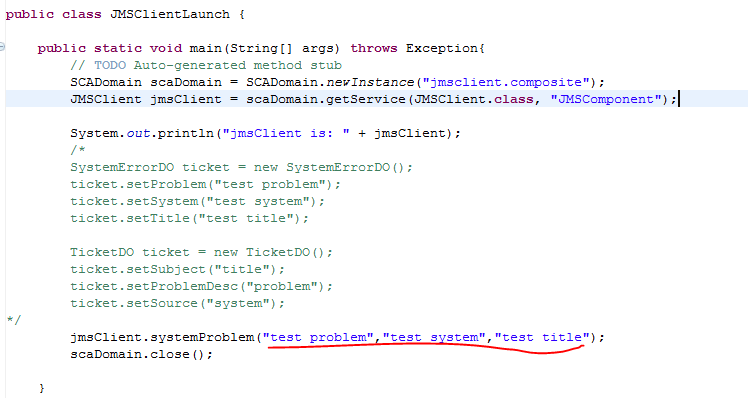
|  |
| --- |
| <interface.wsdl interface="http//issueManagement/#wsdl.interface(SystemErrorComponentService)" />  <binding.ws wsdlElement="http://issueManagement/#wsdl.binding(SystemErrorComponentJMSBinding)"  uri="jms:/SystemError1InQueue?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&amp;java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&amp;java.naming.provider.url=tcp://localhost:61616?jms.watchTopicAdvisories=false" /> |

1. 编写JMS Client，此处用于测试SCA组件的JMS绑定。

注意：由JMS Client传入的数据参数必须与WSDL文件中的请求变量的类型及个数**保持一致**。

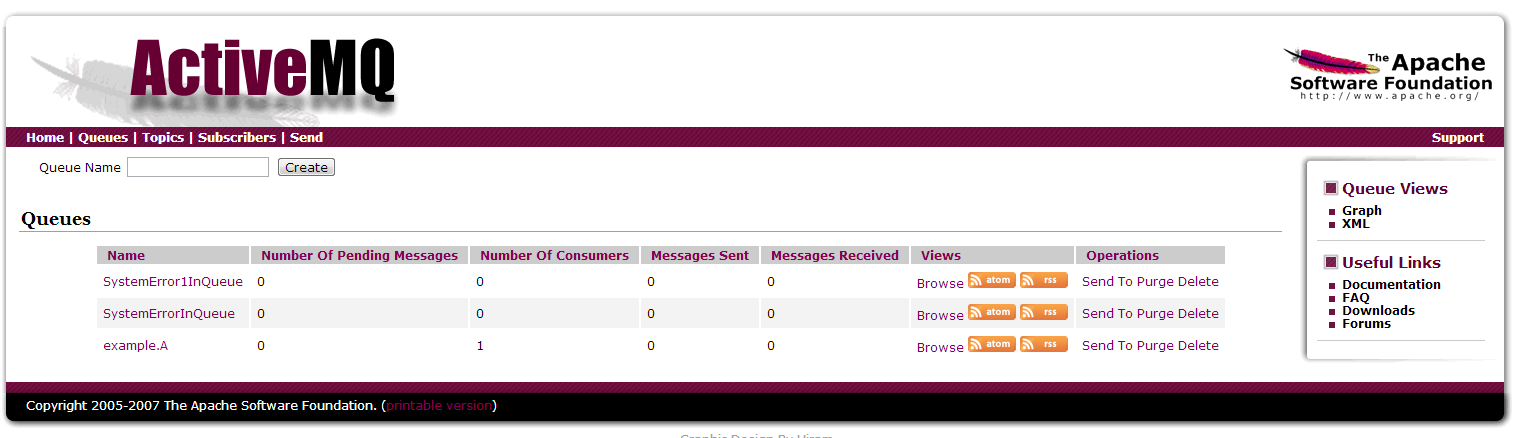


如本例中传入的是3个String类型的数据，所以客户端必须传入3个String类型的数据。如下图：



1. 下载ActiveMQ，本例中下载的为apache-activemq-5.2.0-bin，运行bin文件夹中的activemq.bat，启动消息队列MQ，在<http://localhost:8161/admin>中

选Queues选项，即得到下图：



其中，可以管理队列。

1. 在eclipse中运行两端的程序，可在上图中的管理平台上看到消息队列的个数及发生和消耗的状态。至此，SCA的JMS绑定完成。

## SCA绑定问题及Interface格式问题

SCA中常见的绑定方式（<http://blog.csdn.net/whuqin/article/details/5944169> ）：

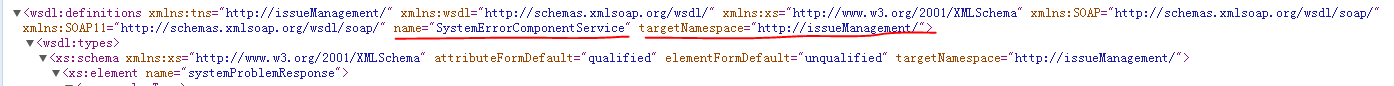
|  |
| --- |
| <binding.sca/>  <!--其中wsdlElement的值有四种格式:  1.服务,<wsdl-namespace-uri>#wsdl.service(<service-name>)  2.wsdl 1.1端口,<wsdl-namespace-uri>#wsdl.port(<service-name>/<port-name>)  3.wsdl 2.0端点,<wsdl-namespace-uri>#wsdl.endpoint(<service-name>/<endpoint-name>)  4.绑定,<wsdl-namespace-uri>#wsdl.binding(<binding-name>)  -->  <binding.ws wsdlElement=""/>  <binding.jms/> |

### 第14.1节中服务的和客户端的JMS配置如下：

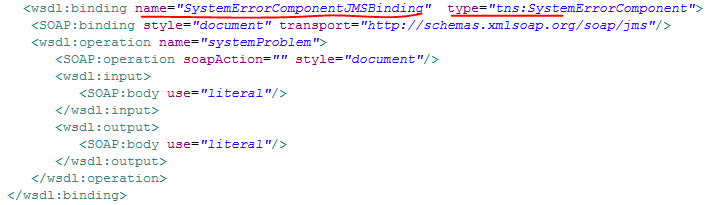
1、服务端的配置

|  |
| --- |
| <binding.ws wsdlElement="http://issueManagement/#wsdl.binding(SystemErrorComponentJMSBinding)" uri="jms:/SystemError1InQueue?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&amp;java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&amp;java.naming.provider.url=tcp://localhost:61616?jms.watchTopicAdvisories=false" /> |

1. “http://issueManagement/”为WSDL文件中的targetNamespace，如下图中



1. “SystemErrorComponentJMSBinding”为WSDL中**JMS的绑定名**



1. “SystemError1InQueue”为自由定义的队列名。
2. “jms.watchTopicAdvisories=false”，《 OpenSOA 》书中没有加这一条，会遇到错误：

|  |
| --- |
| Error sending JMS message to destination : temp-queue://ID:lemovo-PC-14006-1427879481364-0:0:1 javax.jms.InvalidDestinationException: Cannot publish to a deleted Destination: temp-queue://ID:lemovo-PC-14006-1427879481364-0:0:1 |

经过查找，在网页<https://issues.apache.org/jira/browse/AMQ-3336>中找到相关信息，原因为：

|  |
| --- |
| the failover transport will recreate temp destinations but an activemq connection tracks a temp destination deleted advisory.  When the broker is shutdown via stop, the temp dest is deleted as part of the connection close and the advisory fires so the connection sees the destination as deleted.  You need to configure the activemq connection to not track temp advisories: jms.watchTopicAdvisories=false as a url param on the client brokerURL, or via the corresponding attribute on the connection or connection factory. |

从而加上了“jms.watchTopicAdvisories=false”。

1. JMS客户端配置

|  |
| --- |
| <interface.wsdl interface="http//issueManagement/#wsdl.interface(SystemErrorComponentService)" />  <binding.ws wsdlElement="http://issueManagement/#wsdl.binding(SystemErrorComponentJMSBinding)"  uri="jms:/SystemError1InQueue?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&amp;java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&amp;java.naming.provider.url=tcp://localhost:61616?jms.watchTopicAdvisories=false" /> |

其中，相同部分与上述相同。但此处采用接口格式为：

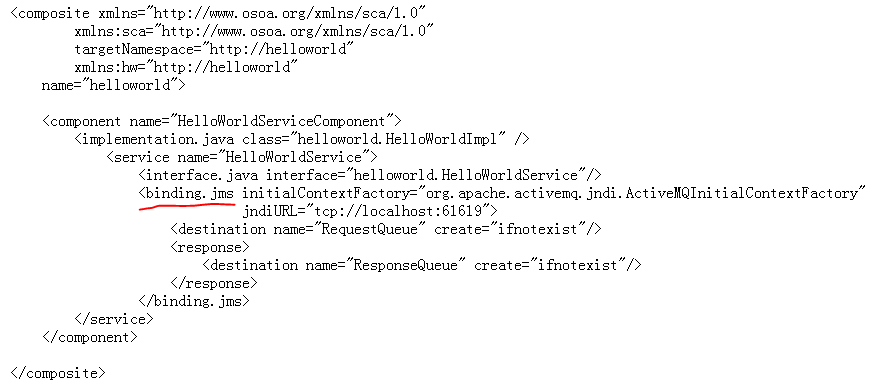
<WSDL-namespace-URI>#wsdl.interface(<**portType**或Interface-name>)

### binding.jms问题

Apache Tuscany支持两种JMS绑定方法。第一种，使用SOAP over JMS，即通过JMS消息实现SOAP，也就是本例中使用的。另一种使用binding.jms类型。

前者已经实现，而后者可能由于eclipse插件安装有问题，而没有binding.jms的类型。

实例可参见apache-tuscany-sca-1.4包中samples文件夹中的hello-world工程。

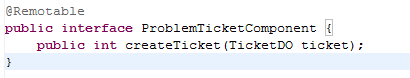


但将工程添加到eclipse中后，出现“The import org.junit cannot be resolved”的错误。解决方法，其中，要安装Junit4：



### 附加：服务接口暴露决定WSDL文件中XSD格式

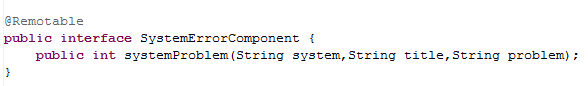
**当接口为**：



生成对应的WSDL文件中，请求变量就是“ticketDO”类型，如图：



当接口为：

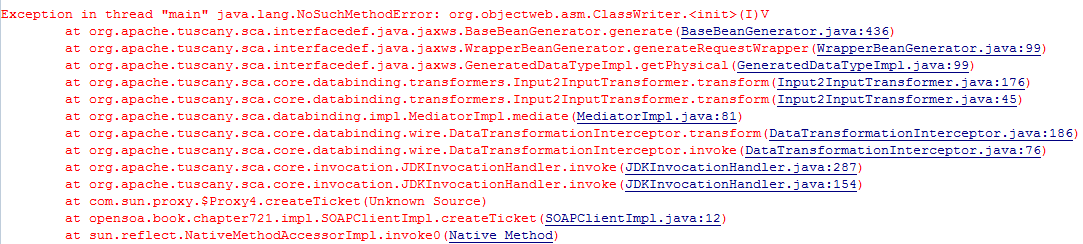


生成的WSDL文件，就是14.1节步骤5的情况。从而，影响JMSClient的编写。

# 第7.2.1节 使用SCA客户端组件实现服务集成

## Exception in thread "main" java.lang.NoSuchMethodError

在eclipse中正常创建jBPM工程时，运行客户端访问第三方的Web服务和SCA服务时，总会遇到异常如下：



即：

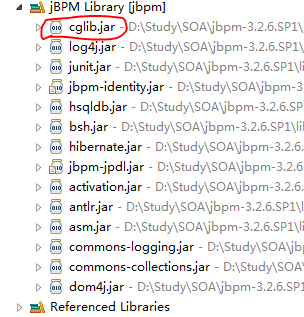
Exception in thread "main" java.lang.NoSuchMethodError: org.objectweb.asm.ClassWriter.<init>(I)V

经过查找资料，发现由于引用包的问题：



1. 正常创建jBPM工程时

软件会自动将jBPM包自动加入到工程中，如下图：



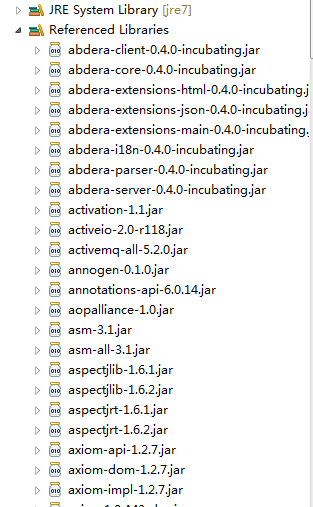
项目在运行的时候会出现上述异常，由于红圈中cglib.jar包，其版本与软件版本不匹配而出现问题。

1. 将jBPM包手动引入项目中时

当手动将jBPM包引入到工程中时，就不会出现上述异常，网上资料说明要将apache-tuscany-sca-1.4中cglib-nodep-2.2.jar去掉，由于版本不匹配，将jBPM包中cglib.jar和cglib-nodep-2.2.jar都去掉后（或者不去掉，由于它们在2.1\_3版本后边，而直接运行2.1\_3版本），程序运行正常。

**推断原因**：当自动创建jBPM工程时，引入上图中的包，在项目运行时，会优先调用“jBPM Library[jbpm]“中的cglib.jar，而出现异常。

在手动引入时，会按下图中包进行检索，优先使用排在前列的包，从而不会出错。



参考: <http://zisetoufa.iteye.com/blog/1552985>

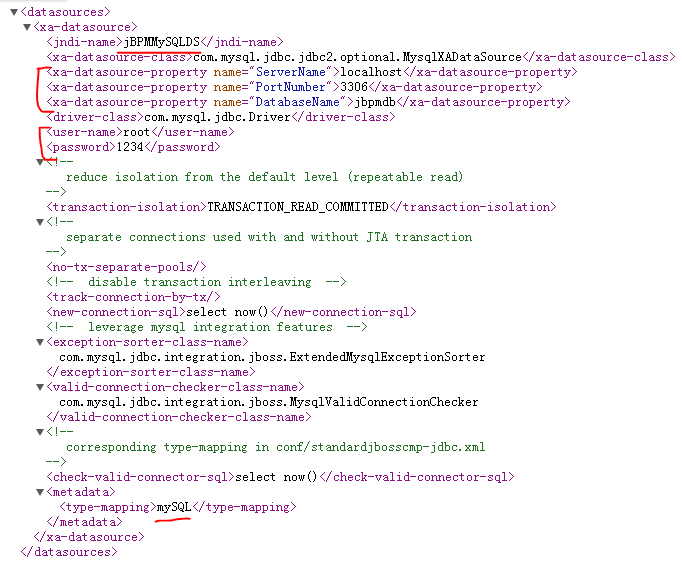
<http://blog.csdn.net/liu372267001/article/details/6393496>

## 15.2 JBPM流程部署步骤

1. 创建process project. 并完成相关流程定义xml和java类编写

2. 构建jBPM数据库（默认为Hypersonic数据库，现改为MySQL）

1. 复制MySQL的JDBC驱动到D:\Study\SOA\jboss-4.2.3.GA\server\default\lib下。
2. 将MySQL数据库对应的示例文件jbpm-mysql-ds.xml（地址为：D:\Study\SOA\jboss-4.2.3.GA\docs\examples\jbpm）复制到D:\Study\SOA\jboss-4.2.3.GA\server\default\deploy下。
3. 修改jbpm-mysql-ds.xml以对应部署环境中的MySQL数据库配置，如下图。

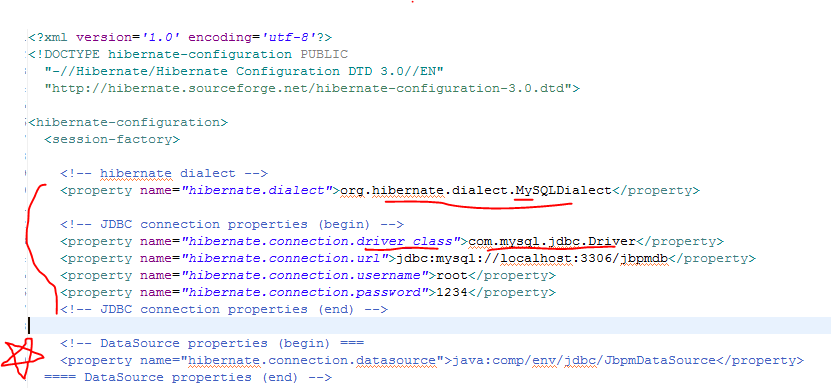


1. 采用MySQL数据库作为jBPM的数据源时，首先要为jBPM创建一个数据库模式。在MySQL数据库里创建一个新的schema，name为jbpmDB的数据库；将D:\Study\SOA\jbpm-3.2.6.SP1\database下的jbpm.jpdl.mysql.sql导入到jbpmDB中。
2. 在jbpmDB数据库的jbpm\_id\_user中加入用户，如下图

|  |
| --- |
| insert into jbpm\_id\_user(ID\_, CLASS\_, NAME\_, EMAIL\_, PASSWORD\_)  values ('1','U','user','ssd@126.com','user'); |

一般初始用户四个，user，manager，shipper，admin；密码和用户名相同。

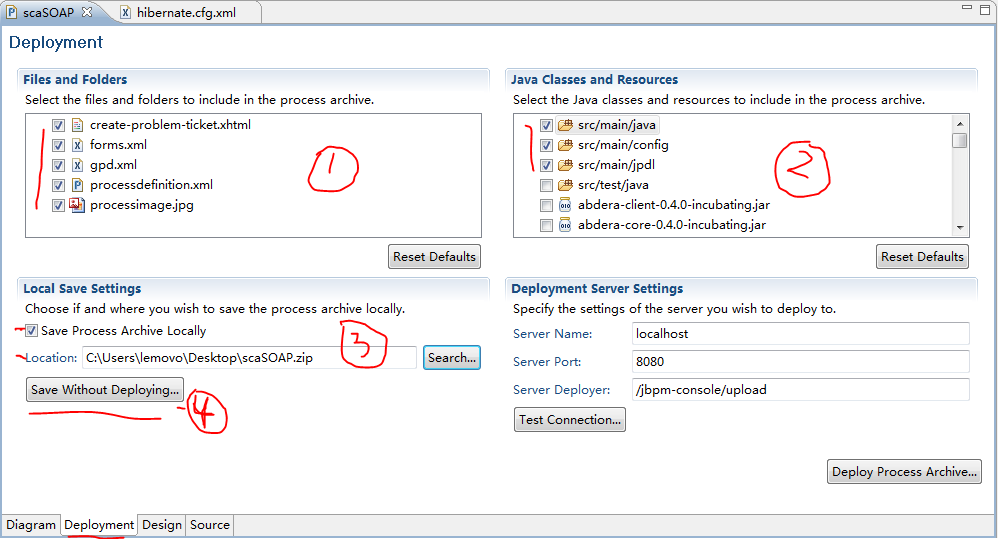
1. 配置hibernate.cxf.xml，如下



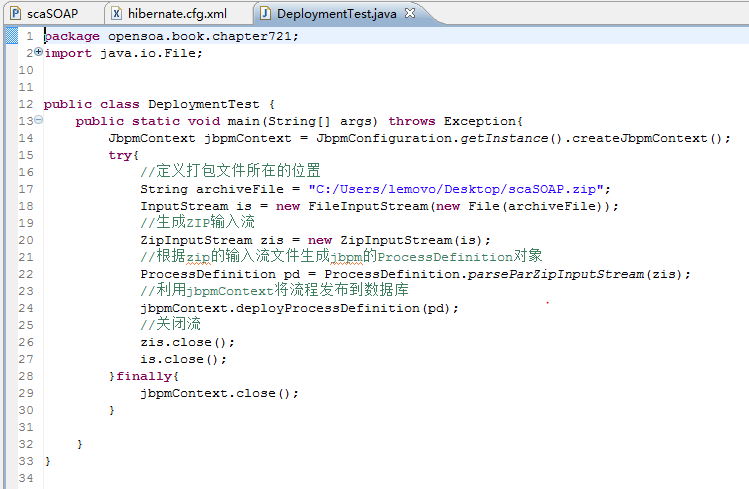
注意图中未被注释掉的部分，需要配置正确，特别第一二行，别遗漏。

另外，在一些书中，五角星部分会取消掉注释，用jndi数据源的形式，但是出现错误，后参考部分资料，直接使用jdbc连接MySQL数据库即可，即五角星部分还是需要注释掉。

3. 流程部署。一般有两种方法，其一是利用jBPM的流程设计器结合可以发布流程的Servlet实现；第二种方法是将流程文件从设计器里导出为一个zip包文件，然后利用应用程序将其发布到数据库中。下图既是第二种方法。

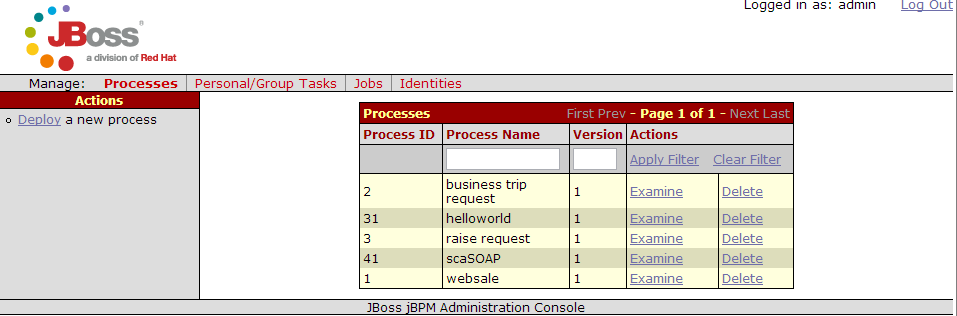


4. 发布流程模板的java类

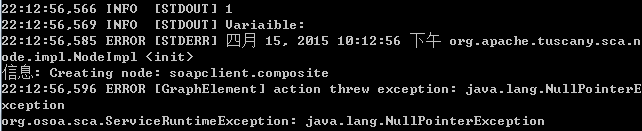


5. 启动jboss，在D:\Study\SOA\jboss-4.2.3.GA\bin通过run.bat启动。

浏览器中打开<http://localhost:8080/jbpm-console>，便进入jboss的web控制台。



可以在web控制台中对流程进行状态监控和测试。但是7.2.1中scaSOAP这个例子在web控制台中进行测试时出现错误，尚未解决，错误如下：



由于该例的action调用的是外部sca的web服务，所以，猜测之一是sca相关包的原因。

但是在eclipse内部用SCAClientSOAPTest.java进行测试，流程是正常的。

(流程部署用到的包和文件在文件夹《7.2.1节部署流程时用到的文件》中，其中注意包org是来自apache-tuscany-sca-1.4中的包tuscany-sca-all-1.4.jar。将其加入Zip文件中后，解决了一些问题，但最终还是因为composite文件的问题，导致失败)

## 15.3 db4o面向对象数据库

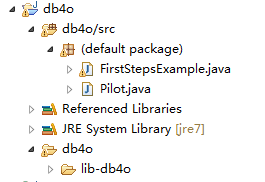
1. 解压db4o-8.0-java.zip，放到英文目录下即可。

2. 设置classpath，将D:\Study\SOA\db4o-8.0-java\db4o-8.0\lib\db4o-8.0.249.16098-all-java5.jar加入classpath中，记得最后别遗漏;号。

3. eclipse中安装插件，在software update对话框中AddSite，选择local按钮，再选择D:\Study\SOA\db4o-8.0-java\db4o-8.0\ome\ObjectManagerEnterprise-Java-8.0.0，然后更新安装即可。

4. db4o的具体使用见D:\Study\SOA\db4o-8.0-java\db4o-8.0\readme.html。

5. db4o在java类中相关基本使用方法，见eclipse中，在项目工程中使用db4o时，记得引入db4o的lib。



# 航班服务项目

## 服务暴露

创建方法不变，建立了3个基本的服务（原子服务）：BasicFlightInfo、FlightRoute、WeatherWebService(远程调用)

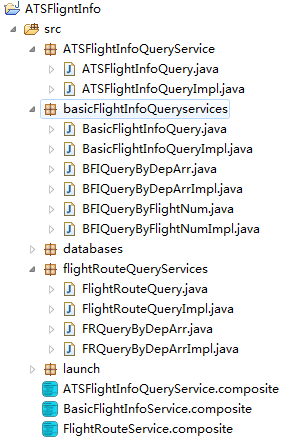
**WSDL文件与Java工程间的对应关系：**

1. WSDL文件中的**name**与服务**暴露出去的接口**一致，在其后边添加后缀**Service**；
2. **targetNamespace**与接口所在的**包名**一致；
3. element中的请求与响应，与接口中包含的方法对应；

## 服务组件嵌套、重用

为更好的实现服务组件的重用，即对基本服务修改最少进行组合。

1. 在Java中，工程调用工程不太适合实现，除非被调用的方法都改为static，不适合。
2. 可以将原子服务中核心的类拷贝到组合服务中，然后只需调用各自接口即可，最后在composite文件中进行配置，如：



1. 配置composite文件时，注意所有composite文件的targetNamespace必须相同，这样才能调用成功。
2. 数组转为List时，采用Arrays.*asList*（）函数，如：

|  |
| --- |
| List<String> FRInfolist = Arrays.*asList*(FRInfo); |

返回的为ArrayList类型的链表，但注意该ArrayList类为**Arrays的内部类**，与java.util中的不同。其不包含添加和删除方法，详见源代码。

*参见*：<http://blog.csdn.net/dreamxiang68/article/details/7377363>

## 回调

1、**回调模型：**



客户端请求完后，服务器处理请求，客户端继续执行其他操作，等到服务器回调时，客户端再执行。实现异步通信。

在具体实现时，要有一个回调接口接收服务端的调用，如FlightInfoCallBack工程中的FlightInfoQueryCallBack为回调接口；WeatherWeb作为服务端执行回调的接口，WeatherWebImpl为其实现。

其中，WeatherWeb在composite文件中必须声明为组件；单向服务调用模式的接口函数必须为void，@Callback异步效果不明显；

2、**预想在回调接口将查到的数据返回**

构造了一个成员变量：

|  |
| --- |
| **private** List<String> *flightinfoquery*=**new** ArrayList<String>(); |

但是返回数值总是没有天气信息，原因：程序中两个函数中的该变量属于不同的对象，故出现此问题。在前边加上static就解决了，改变量为类的全局变量。

**注意：**java中，**成员变量问题**

|  |
| --- |
| public static final String APPNAME= "xyz"; // global constant   public static String currentUser = "abc";  // global variable |

如果不用static修饰，变量为实例变量，与对象共存亡；用static修饰，变量为类属性，与类共存亡，即其可以被所有对象访问。

**成员变量小结：**

变量的划分: 所有变量可以分成成员变量和局部变量，成员变量又可以分成实例属性(不以static修饰)和类属性(以static修饰)，而且成员变量要在类里面定义，而局部变量则分为形参(方法签名中定义的变量)，方法局部变量(在方法内定义)和代码块局部变量(在代码块内定义(例如for循环里面的变量))，并且局部变量定义必须在方法里面！

成员变量: 成员变量被分为**类属性**和**实例属性**两种，定义一个属性时不使用static修饰的就是实例属性，使用static修饰的就是类属性。其中类属性可以理解为类成员变量，他作为类的一个成员，与类共存亡；实例属性则可理解为实例成员变量，它作为实例的一个成员，与实例共存亡。类属性也可以让该类的实例来访问(实例.类属性)，其实这个只是一个假象,其实是类.类属性这样来...

## JMS