## 1、发布一个简单的终端

### 1、接口类HelloWS

|  |
| --- |
| @WebService  **public** **interface** HelloWS {  @WebMethod  **public** String sayHello(String name);  } |

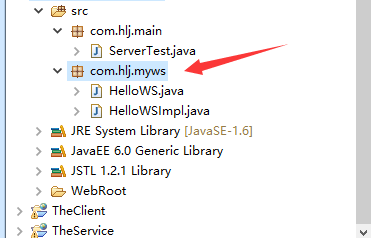
### 2、实现类

|  |
| --- |
| @WebService  **public** **class** HelloWSImpl **implements** HelloWS {  @Override  **public** String sayHello(String name) {  System.*out*.println("server sayHello()"+name);  **return** "Hello " +name;  }  } |

### 3、发布终端

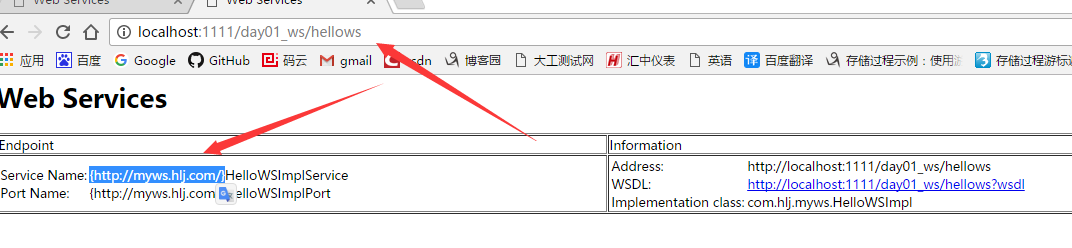
|  |
| --- |
| **public** **class** ServerTest {  **public** **static** **void** main(String[] args) {  String address = "http://localhost:1111/day01\_ws/hellows";    Endpoint.*publish*(address , **new** HelloWSImpl());  System.*out*.println("发布webservice成功!");  }  } |

### 4、项目结构如下



### 5、发布终端，会发现类似于targetNamespace的东西产生了，它其实就是我们的项目中接口所在包的名字

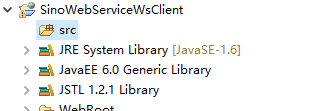
|  |
| --- |
| **public** **class** ServerTest {  **public** **static** **void** main(String[] args) {  String address = "http://localhost:1111/day01\_ws/hellows";    Endpoint.*publish*(address , **new** HelloWSImpl());  System.*out*.println("发布webservice成功!");  }  } |



|  |  |
| --- | --- |
| Service Name: | {http://myws.hlj.com/}HelloWSImplService |
| Port Name: | {http://myws.hlj.com/}HelloWSImplPort |

# 2、根据1中建立一个客户端web项目

## 1、项目建立好后

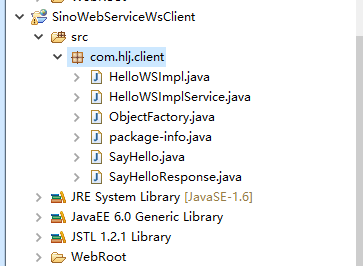


## 2、利用命令生成客户端代码，并建立包com.hlj.client

wsimport -s E:\\WorkSpace\\JspWorkSpace\\MyeclipseWebService\\SinoWebServiceWsClient\\src -p com.hlj.client -keep http://localhost:1111/day01\_ws/hellows?wsdl



## 3、刷新项目

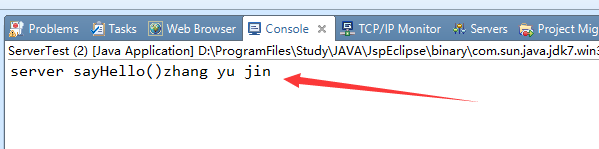


## 4、客户端建立测试类调用接口HelloWSImpl,也可以是HelloWS接口

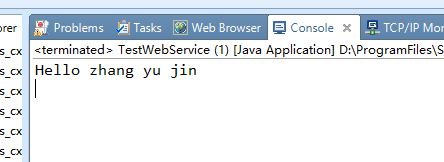
|  |
| --- |
| **public** **class** TestWebService {    **public** **static** **void** main(String[] args) {  HelloWSImpl hello = **new** HelloWSImplService().getHelloWSImplPort();    String name = hello.sayHello("zhang yu jin ");  System.*out*.println(name);  }  } |

## 5、控制台打印如下

### 服务的控制台

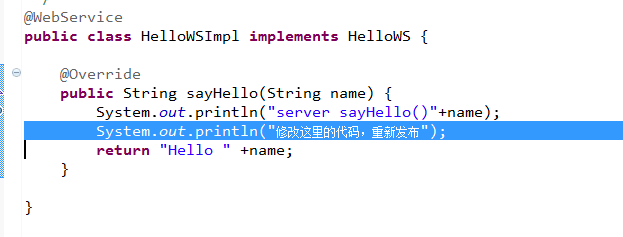


### 客户端控制台



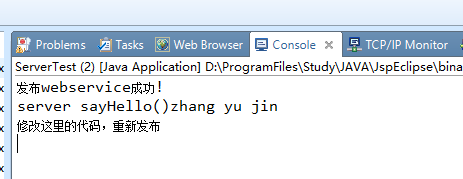
## 6、直接修改服务的接口中的实现代码重新发布，但是在客户端不重新获取wsdl文件，成功

### 1、服务中添加代码

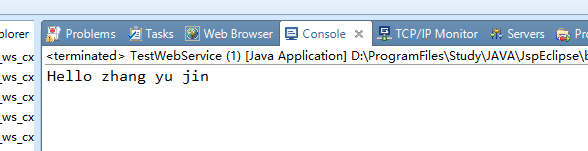


### 2、测试客户端重新调用，控制台如下

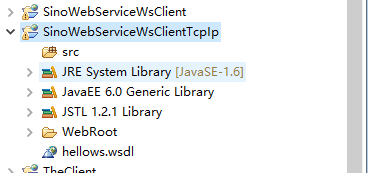
#### 1、服务端控制台



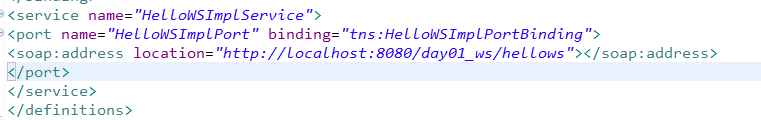
#### 2、客户端控制台



# 3、配置eclipse用来，监听请求，与真实代码无关，（重新建立一个项目）我这里的如下



## 1、复制wsdl文件，并修改改为8080（一共两处，修改最下面的）（这里的ip因为是本机测试所以才是localhost，实际项目中为其他的ip）

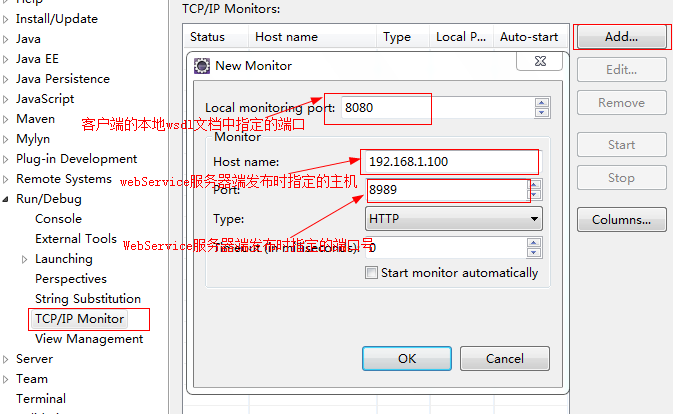


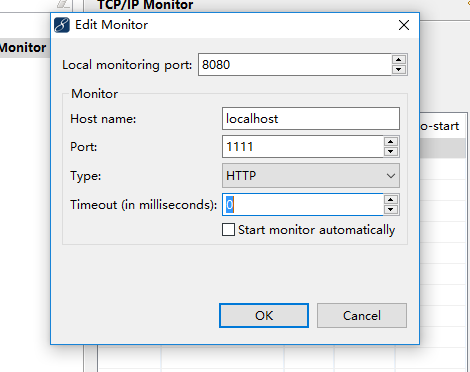
<http://localhost:1111/day01_ws/hellows?wsdl>

## 2、cmd窗口中拖动这个wsdl文件创建



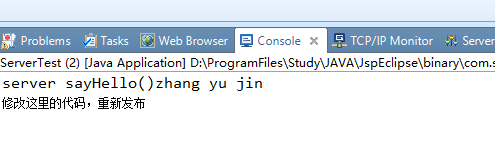
## 3、因为我们上面修改了端口，肯定是不行的，所以这里配置tcp/ip，端口转发器，用来监听这个端口，启动这个tcp/ip



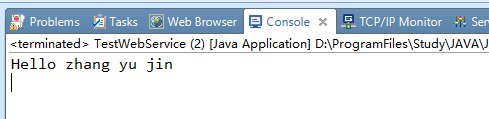


## 4、测试，客户端开始运行

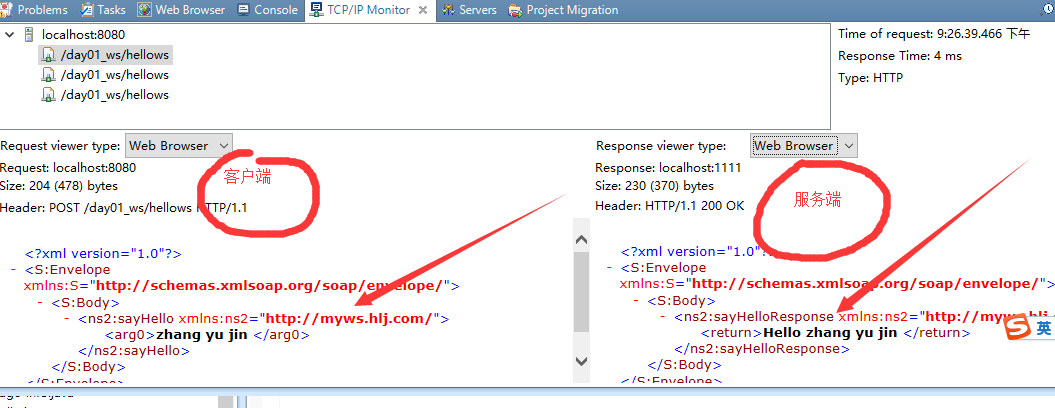
### 1、服务端控制台



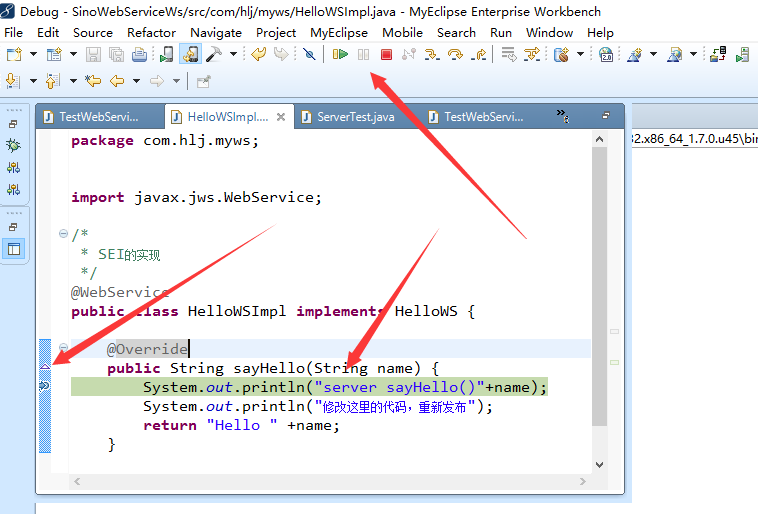
### 2、客户端控制台



### 3、tcp/ip控制台，可以观察到客户端和服务端的东西



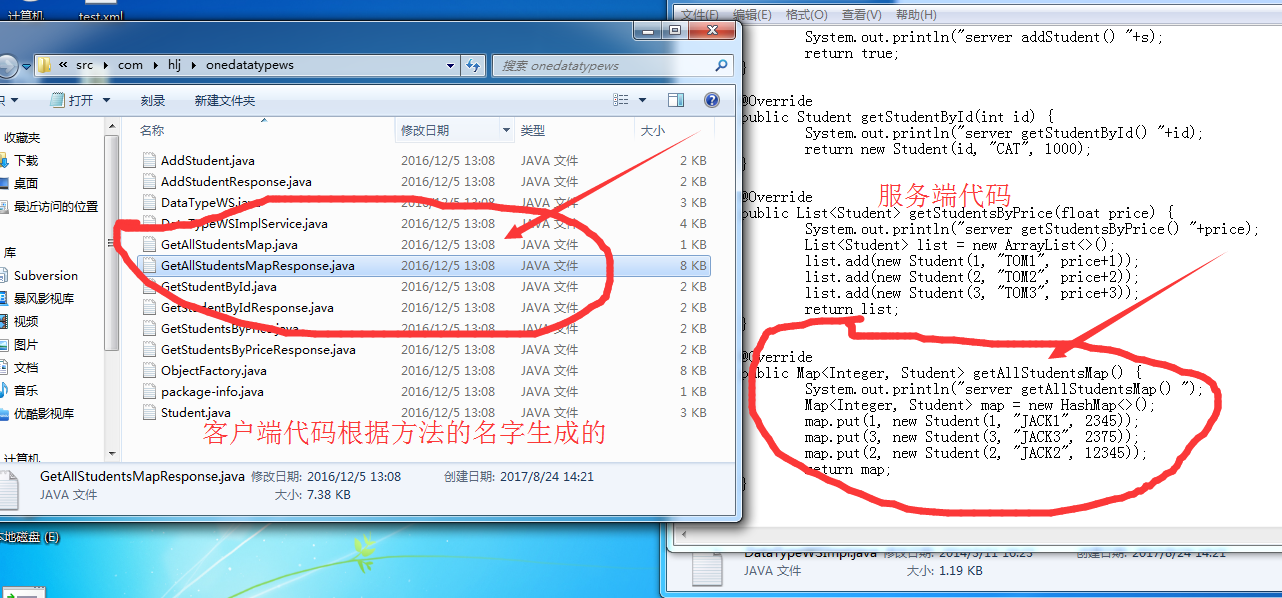
# 4、debug项目，只要服务端debug，客户端调用的时候就可以debug调试了。



# 5、参数和结果类型

## 1、接口函数返回值为map类型

### 1、本地客户端根据wsdl生成代码之后会有一个Return内置类，内置于根据这个函数的名字所产生的类。



### 2、服务端接口函数为

|  |
| --- |
| @Override  public Map<Integer, Student> getAllStudentsMap() {  System.out.println("server getAllStudentsMap() ");  Map<Integer, Student> map = new HashMap<>();  map.put(1, new Student(1, "JACK1", 2345));  map.put(3, new Student(3, "JACK3", 2375));  map.put(2, new Student(2, "JACK2", 12345));  return map;  } |

### 3、客户端调用方法以及结果的处理

|  |
| --- |
| public class ClientTest2 {  public static void main(String[] args) {  DataTypeWSImplService factorty = new DataTypeWSImplService();  DataTypeWS dataTypeWS = factorty.getDataTypeWSImplPort();  Student s = new Student();  s.setId(12);  s.setName("张宇晋");  s.setPrice(23);    boolean success = dataTypeWS.addStudent(s);  //System.out.println("client "+success);    List<Student> list = dataTypeWS.getStudentsByPrice(34);  System.out.println(list);    Return r = dataTypeWS.getAllStudentsMap();  List<Entry> entrys = r.getEntry();  for(Entry entry : entrys) {  Integer id = entry.getKey();  Student student = entry.getValue();  System.out.println(id+"\_"+student);  }  }  } |

# 6、spring 搭建webservice（包括接口中添加的拦截器）（这个没有实战）

## 1、服务端配置cxf（项目tomcat启动即可成功发布）

|  |
| --- |
| <!-- 添加终端 -->  <jaxws:endpoint  id=*"orderWS"*  implementor=*"com.atguigu.day02\_ws\_cxf\_spring.ws.OrderWSImpl"*  address=*"/orderws"*> <!-- 虚拟地址 和id名字不一定一样 -->  <!-- <jaxws:inInterceptors>  <bean class="com.atguigu.day01\_ws.interceptor.CheckUserInterceptor"></bean>  </jaxws:inInterceptors> -->  </jaxws:endpoint> |

## 2、客户端配置cxf

|  |
| --- |
| <jaxws:client id=*"orderClient"*  serviceClass= *"com.atguigu.day02\_ws\_cxf\_spring.ws.OrderWS"*  address= *"http://localhost/day02\_ws\_cxf\_spring/orderws"*>      <jaxws:outInterceptors>  <bean class=*"org.apache.cxf.interceptor.LoggingOutInterceptor"*/>  <bean class=*"com.atguigu.day01\_ws\_cxf\_client.interceptor.AddUserInterceptor"*>  <constructor-arg name=*"name"* value=*"xfzhang"*/>  <constructor-arg name=*"password"* value=*"123456"*/>  </bean>  </jaxws:outInterceptors>  </jaxws:client> |

## 3、客户端调用接口（客户端配置的id就是接口的名字）

|  |
| --- |
| **public** **class** ClientTest {  **public** **static** **void** main(String[] args) {  ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext(**new** String[] {"client-beans.xml"});  OrderWS orderWS = (OrderWS) context.getBean("orderClient");  Order order = orderWS.getOrderById(24);  System.*out*.println(order);  }  } |

# 7、全局拦截器和接口私有的拦截器

## 1、中科软项目中已经有了全局拦截器，不再讲，这里是添加的接口私有的拦截器

### 1、服务端

|  |
| --- |
| <jaxws:server id=*"commonWS"* serviceClass=*"com.hlj.webservice.CommonWSInter"*  address=*"/CommonService"*> 解释下图  <jaxws:serviceBean>  <ref bean=*"CommonWSImpl"* /> <!-- 引入接口实现类 -->  </jaxws:serviceBean>  <jaxws:inInterceptors>  <bean class="com.atguigu.day01\_ws.interceptor.CheckUserInterceptor"></bean>  </jaxws:inInterceptors>  </jaxws:server>  <!-- 接口实现类 -->  <bean id=*"CommonWSImpl"* class=*"com.hlj.webservice.CommonWSImpl"* /> |

### 2、客户端property 添加即可

|  |
| --- |
| <bean id=*"commonClientFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"* value=*"com.hlj.webservice.CommonWSInter"* />  <property name=*"address"* value=*"http://localhost:8080/SpringProject/services/CommonService"* />  <!--<property name="address" value="http://10.3.181.48:8080/dsp\_oa/services/CommonService" /> -->  <property name=*"outInterceptors"*>    </property>  </bean> |

## 2、检查用户的拦截器

package com.atguigu.day01\_ws.interceptor;  
  
import javax.xml.namespace.QName;  
  
import org.apache.cxf.binding.soap.SoapMessage;  
import org.apache.cxf.headers.Header;  
import org.apache.cxf.interceptor.Fault;  
import org.apache.cxf.phase.AbstractPhaseInterceptor;  
import org.apache.cxf.phase.Phase;  
import org.w3c.dom.Element;  
/\*\*  
 \* 查检用户的拦截器  
 \*/  
public class CheckUserInterceptor extends AbstractPhaseInterceptor<SoapMessage> {  
  
 public CheckUserInterceptor() {  
 super(Phase.PRE\_PROTOCOL);  
 System.out.println("CheckUserInterceptor()");  
 }  
 */\*  
 <Envelope>  
 <head>  
 <atguigu>  
 <name>xfzhang</name>  
 <password>123456</password>  
 </atguigu>  
 <atguigu2>  
 <name>xfzhang</name>  
 <password>123456</password>  
 </atguigu2>  
 <head>  
 <Body>  
 <sayHello>  
 <arg0>BOB</arg0>  
 <sayHello>  
 </Body>  
 </Envelope>  
 \*/*

@Override  
 public void handleMessage(SoapMessage message) throws Fault {  
 Header header = message.getHeader(new QName("atguigu"));  
 if(header!=null) {  
 Element atguiguEle = (Element) header.getObject();  
 String name = atguiguEle.getElementsByTagName("name").item(0).getTextContent();  
 String password = atguiguEle.getElementsByTagName("password").item(0).getTextContent();  
 if("xfzhang".equals(name) && "123456".equals(password)) {  
 System.out.println("Server 通过拦截器....");  
 return;  
 }  
 }  
 //不能通过  
 System.out.println("Server 没有通过拦截器....");  
 throw new Fault(new RuntimeException("请求需要一个正确的用户名和密码!"));  
 }  
  
}

## 3、添加用户拦截器

package com.atguigu.day01\_ws\_cxf\_client.interceptor;  
  
import java.util.List;  
  
import javax.xml.namespace.QName;  
  
import org.apache.cxf.binding.soap.SoapMessage;  
import org.apache.cxf.headers.Header;  
import org.apache.cxf.interceptor.Fault;  
import org.apache.cxf.phase.AbstractPhaseInterceptor;  
import org.apache.cxf.phase.Phase;  
import org.apache.xml.utils.DOMHelper;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
  
public class AddUserInterceptor extends AbstractPhaseInterceptor<SoapMessage> {  
  
 private String name;  
 private String password;  
   
   
   
 public AddUserInterceptor(String name, String password) {  
 super(Phase.PRE\_PROTOCOL);//准备协议化时拦截  
 this.name = name;  
 this.password = password;  
 System.out.println("AddUserInterceptor()....");  
 }  
  
 */\*  
 <Envelope>  
 <head>  
 <atguigu>  
 <name>xfzhang</name>  
 <password>123456</password>  
 </atguigu>  
 <atguigu2>  
 <name>xfzhang</name>  
 <password>123456</password>  
 </atguigu2>  
 <head>  
 <Body>  
 <sayHello>  
 <arg0>BOB</arg0>  
 <sayHello>  
 </Body>  
 </Envelope>  
 \*/* @SuppressWarnings("deprecation")  
 @Override  
 public void handleMessage(SoapMessage msg) throws Fault {  
 List<Header> headers = msg.getHeaders();  
   
 */\*  
 <atguigu>  
 <name>xfzhang</name>  
 <password>123456</password>  
 </atguigu>  
 \*/* Document document = DOMHelper.createDocument();  
 Element rootEle = document.createElement("atguigu");  
 Element nameELe = document.createElement("name");  
 nameELe.setTextContent(name);  
 rootEle.appendChild(nameELe);  
 Element passwordELe = document.createElement("password");  
 passwordELe.setTextContent(password);  
 rootEle.appendChild(passwordELe);  
   
 headers.add(new Header(new QName("atguigu"), rootEle));  
 System.out.println("client handleMessage()....");  
 }  
  
}