xiEclipse，tomcat安装：E:\02 工作库\00 共通\04 工具

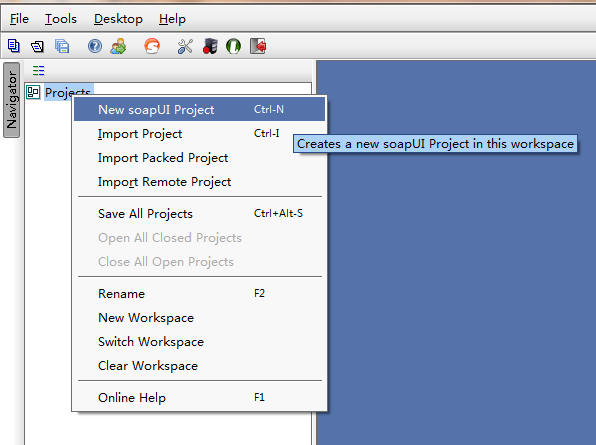
导入finict框架：E:\02 工作库\00 共通\03 开发框架\fihict

部署到tomcat.,访问地址：<http://localhost:8080/fihict_example/helloWebservice?wsdl>

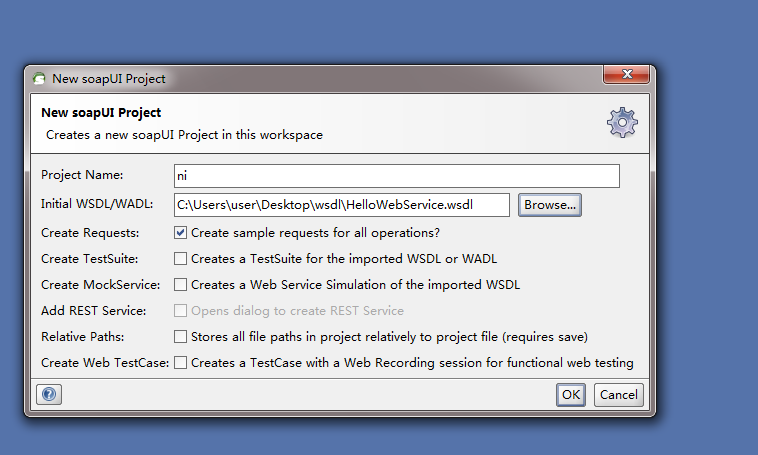
右键查看源文件保存了wsdl文件。

现在用SoapUI调用这个webService:

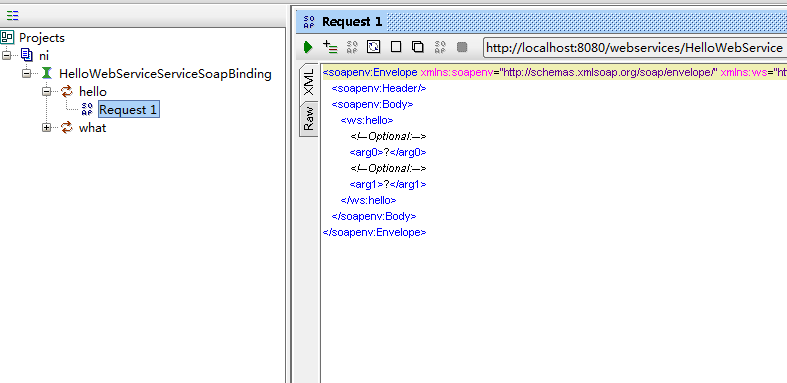
新建一个项目



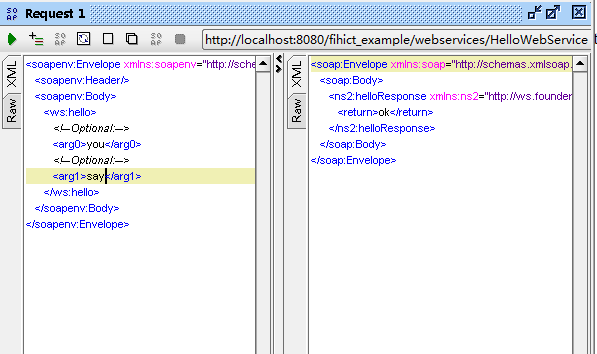
在Project Name中可以输入一个名称（注意不可以是中文），然后在Initial WSDL/WADL:中导入我们已经保存的WSDL格式的文件



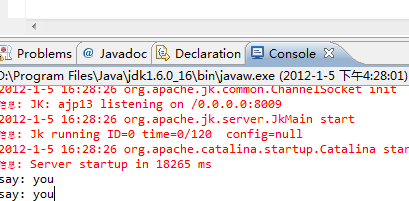
然后双击打开Request



在<arg0></arg0>和<arg1></arg1>中可以随意输入东西，点击左上角的绿色三角行就会出现此图

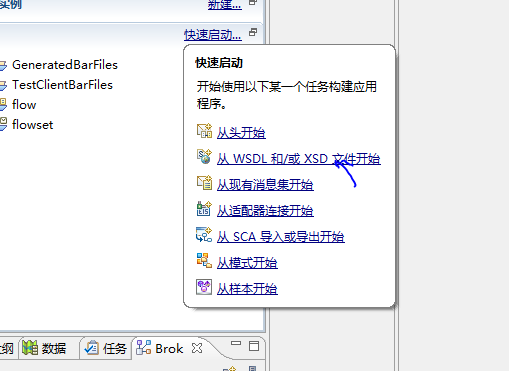


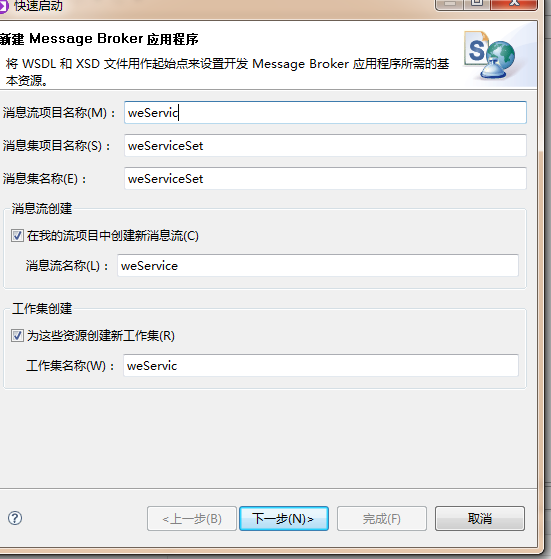
而在eclipse中会在控制台中输出



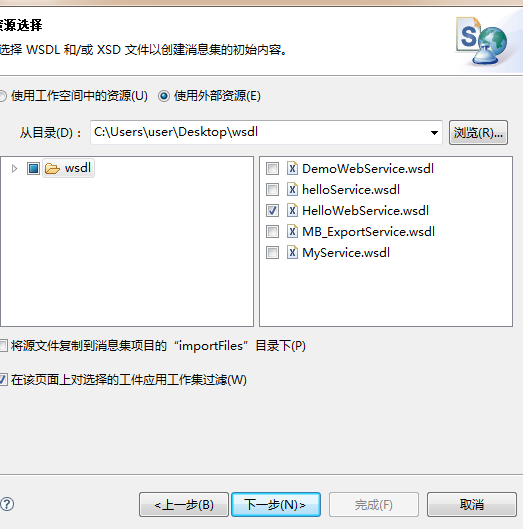
调用成功。

1. 对外发布一个webService:

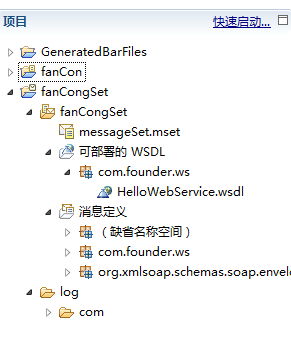




注：消息集，消息流名称不能有空格，且不能为中文。



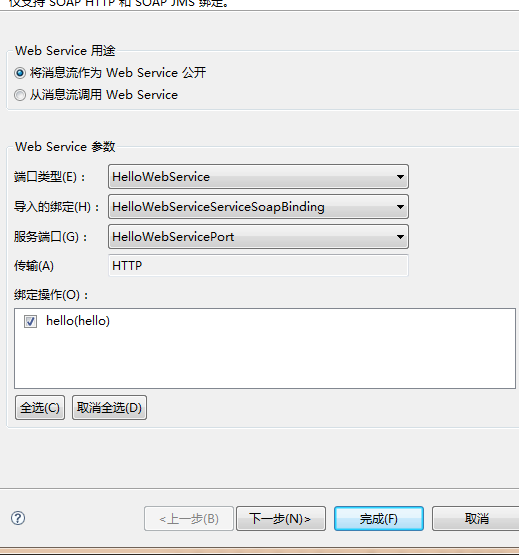
使用外部资源导入我们刚才保存的那个wsdl文件。



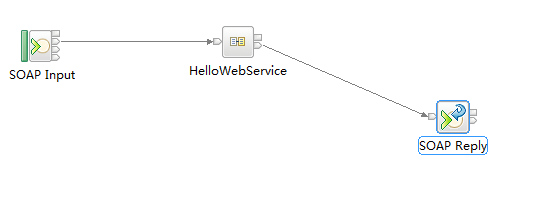
完成之后看到会有两个工程：在可部署的WSDL下将导入的wsdl拖入到画布中：

第一个默认选项为对外发布一个webservice

第二个为调用外布webService.



完成之后会看到以下三个节点。连接起来。



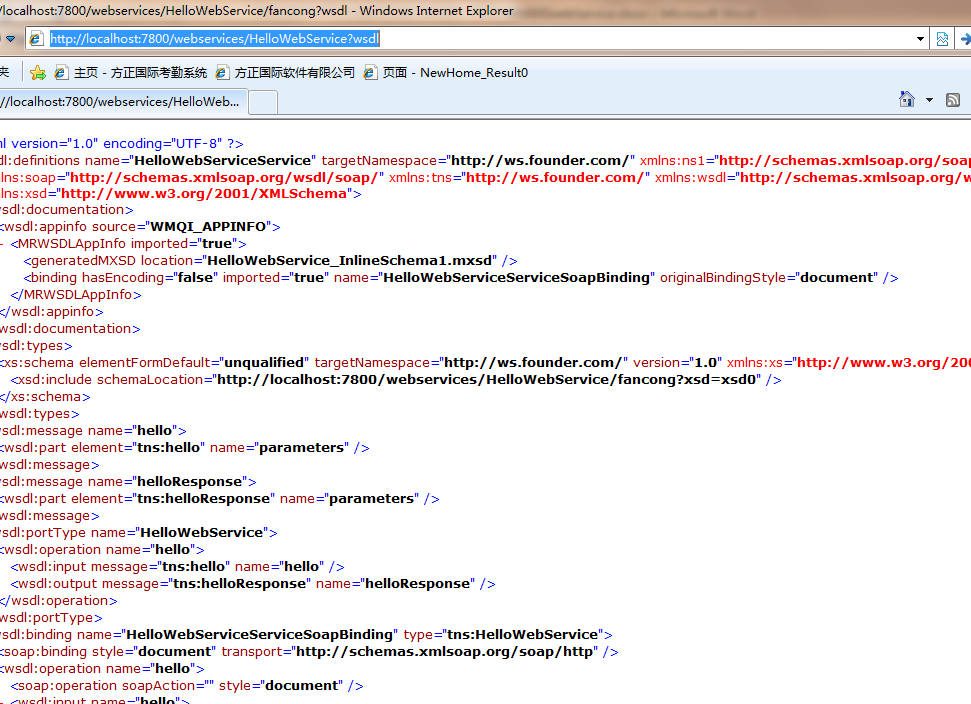
在SOAP Input 节点属性Http Transport中如下属性框勾选。这样我们的wsdl文件才可以发布出去被访问到。



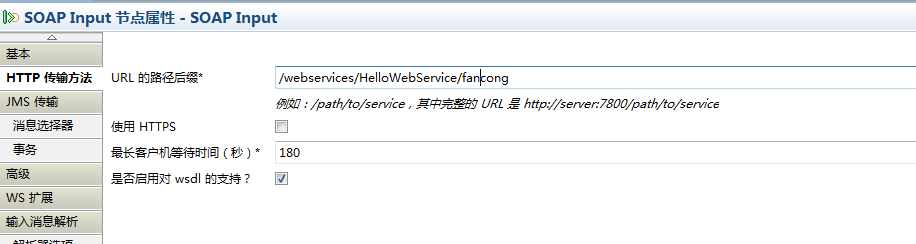
这时我们就可以将消息部署发布出去，打开浏览器就可访问了。

在SOAP Input 节点属性Http Transport中将地址复制下

MB默认端口7800输入访问地址就可看到发布出去wsdl文件。



也可在这个地方上改为我们想要的地址：

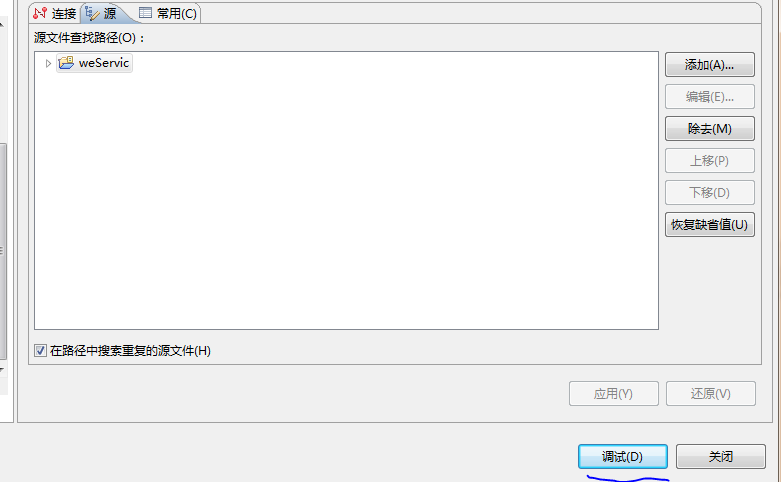




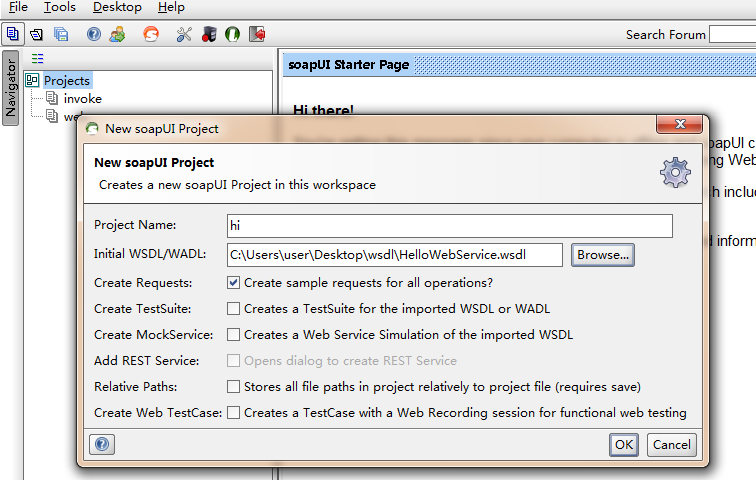
测试：

先将代理上部署的webService停掉。（不行，在将代理停掉重启）

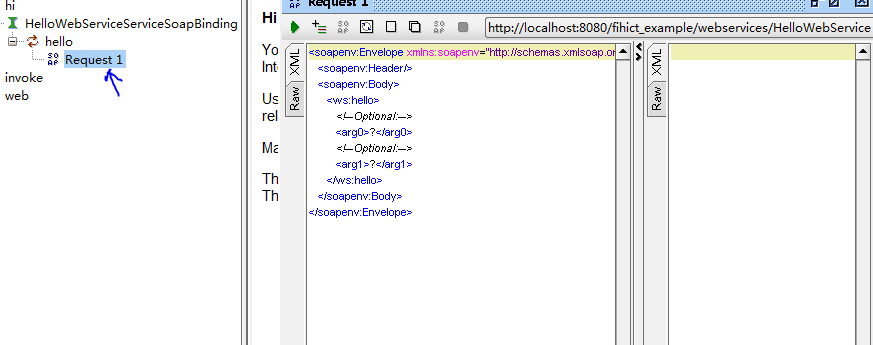
将消息配置到debug里：



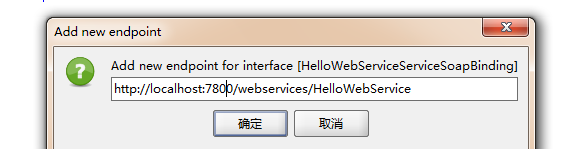
打开SoapUI.

t

创建一个新的工程导入刚才的wsdl文件，我们现在需要选择创建一个调用者的。点OK，可看到自动生成一个请求消息，打开后是一个xml结构的消息

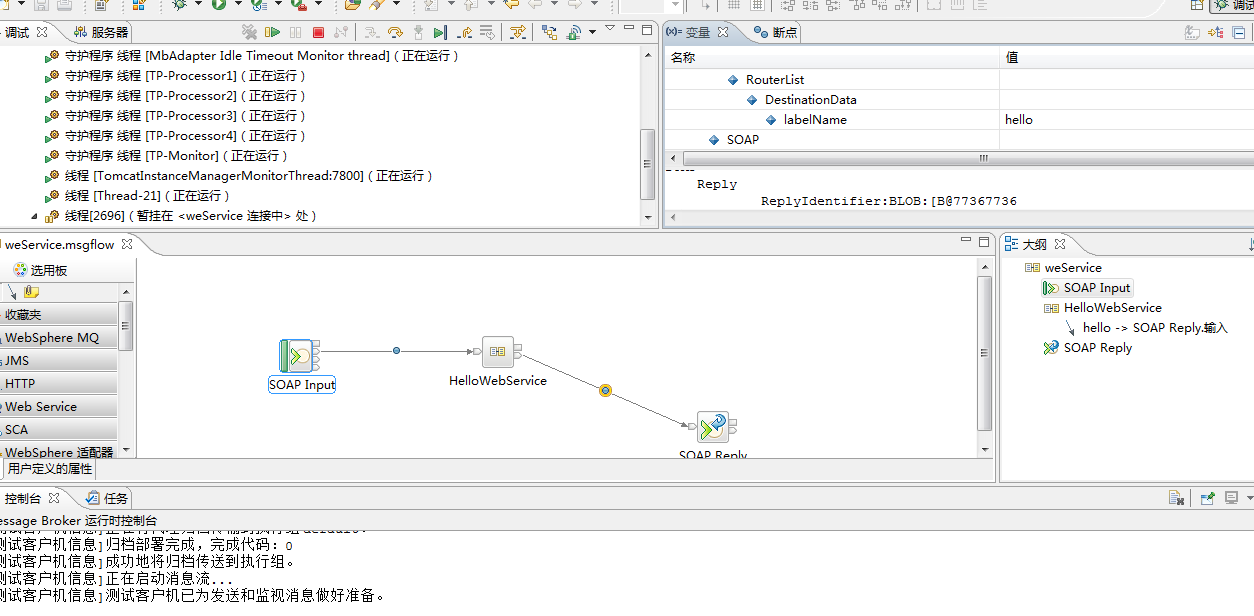


然后将wsdl刚发布出去的地址 copy下赋盖到现在地址上

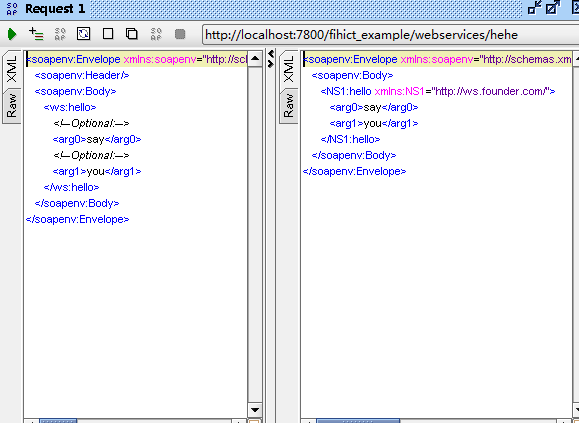


可在参数中输入参数：点右上角绿三角的测试

调试中进入程序：



测试成功：

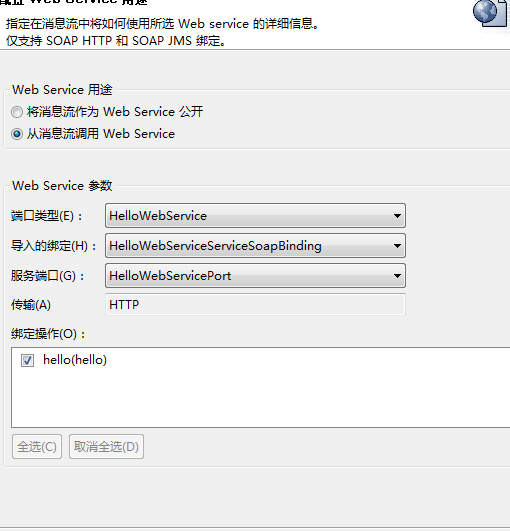


1. 调用外布的webService:

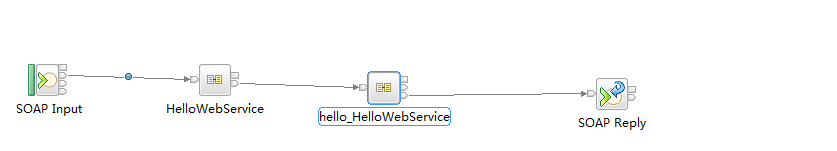
同样在快速启动中从wsdl文件中导入我们保存好的wsdl文件。

这次将导入进来的wsdl文件拖到画布中时，我们要选择第二个：

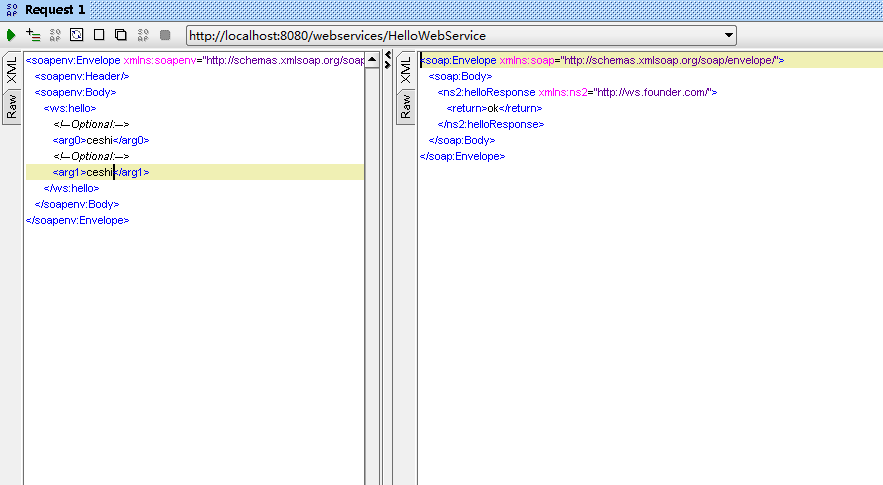
调用外布的webService.



完成之后将这四个节点连接起来。如下图：

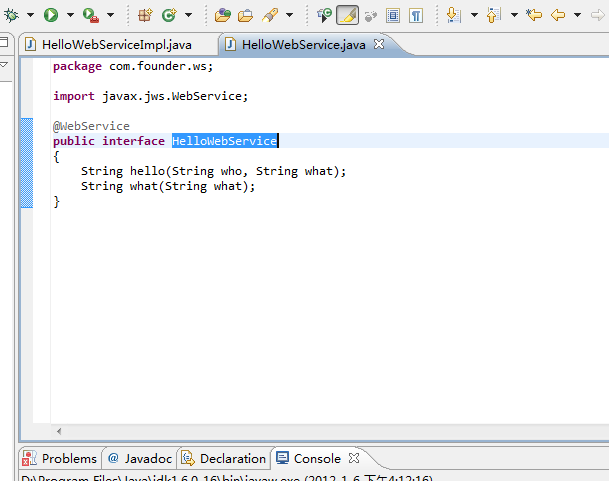


然后在soapUI中测试，点击左上角的绿色三角按钮就可出现如下图

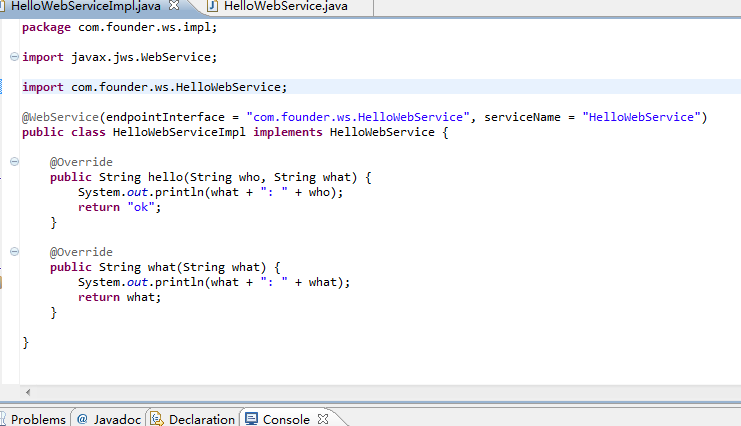


第二个例子 第一个方法的参数的返回值作为第二个方法的参数

在eclipse当中添加一个方法，先定义一个接口



实现这个方法



然后开启tomcat在IE中访问<http://localhost:8080/webservices/HelloWebService?wsdl>会出现这两个方法“hello”和”what”,然后另存为“.wsdl”格式



在WebSphere Message Broker Toolkit 7.0工具中创建一个项目



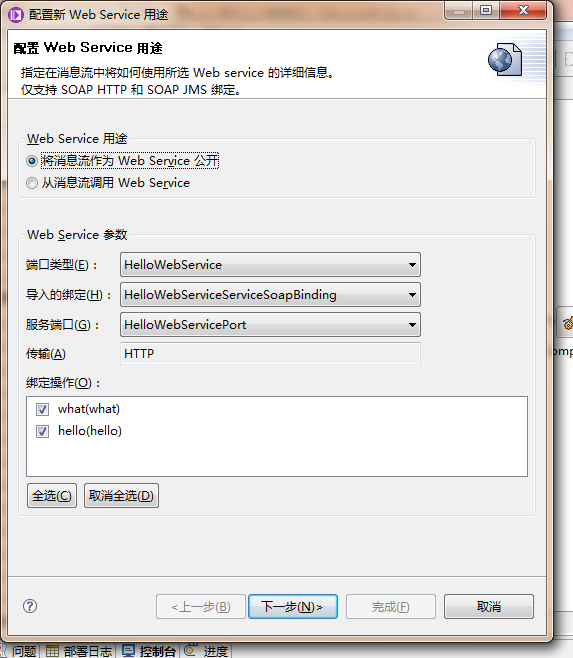
然后建立这个项目其中在创建项目时不可以出现中文



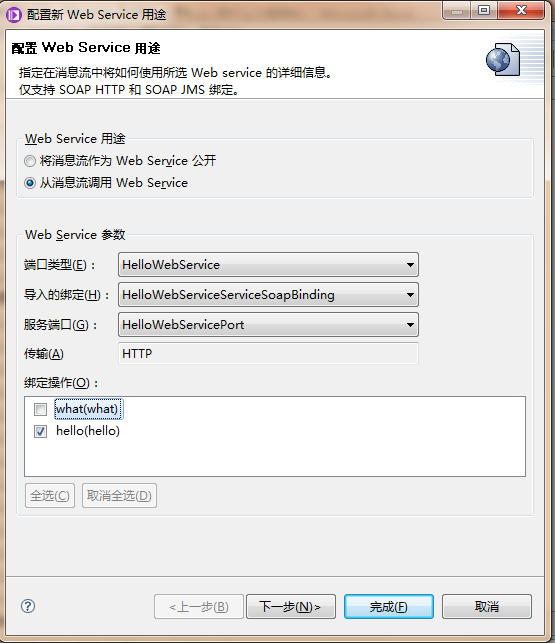
然后把HelloWebService.wsdl拖到fan.msgflow中



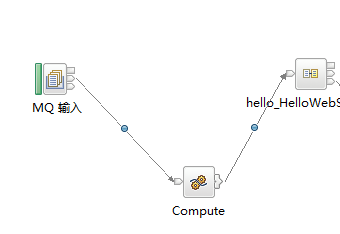
拖带fan.msgflow中后会出现



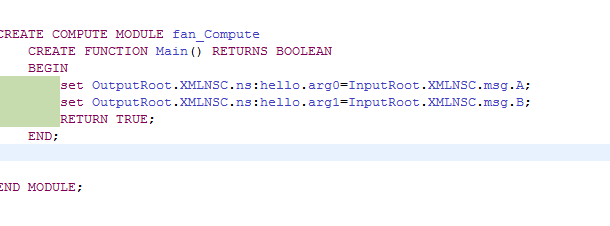
然后选择从消息流调用web Service 并且绑定一个方法



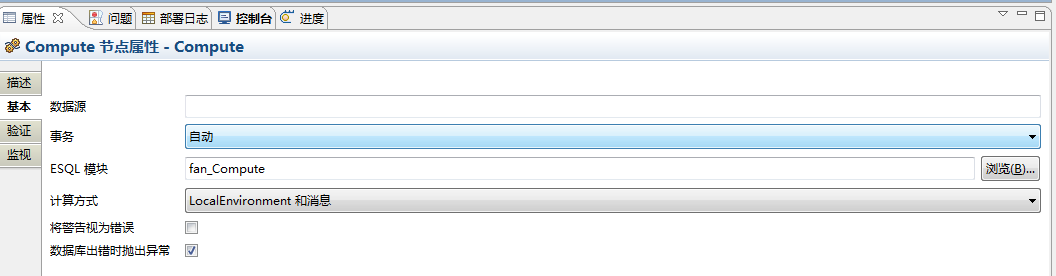
然后从空间中拖出一个MQinput和一个Conpute，然后和Hello\_helloWebService三个节点连接起来



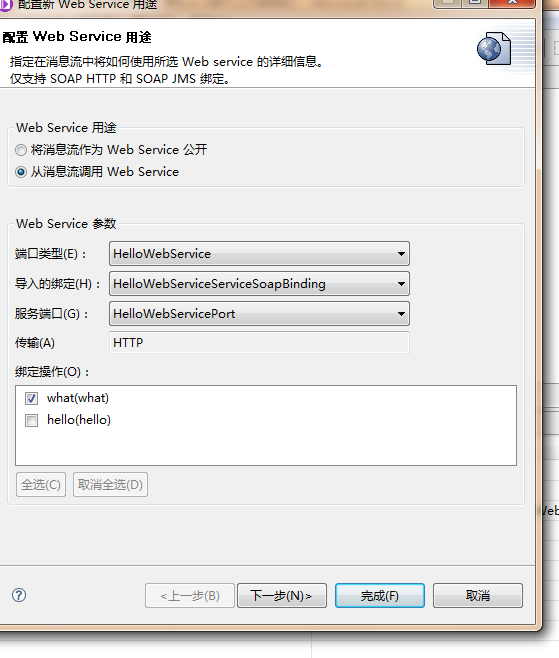
然后双击打开Compute在里面编码转换代码



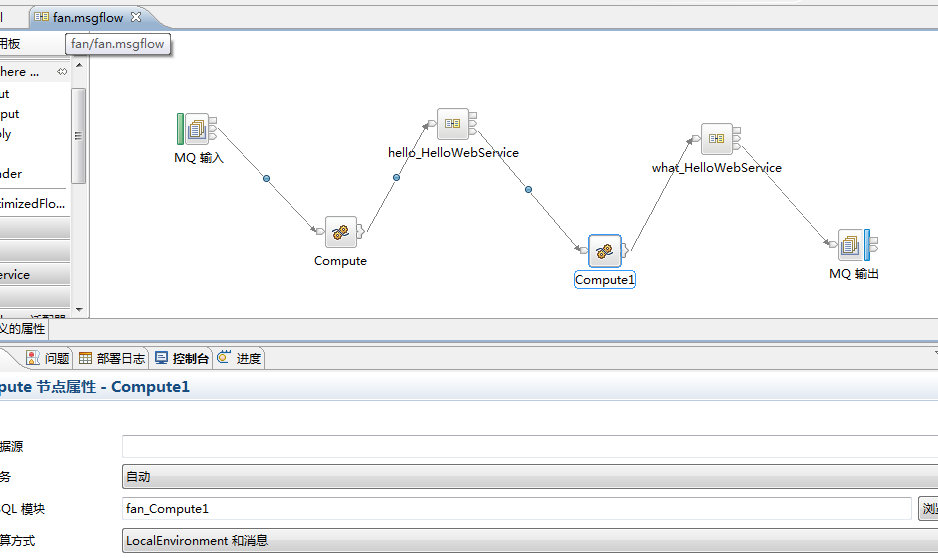
在Compute属性中把计算方式选为LocalEnvironment和消息



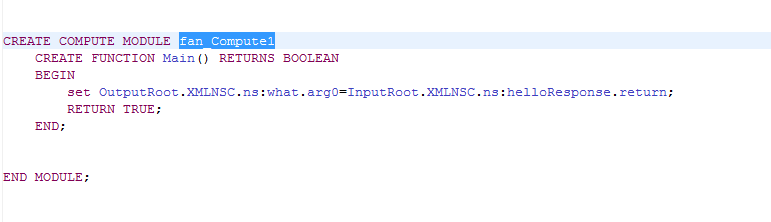
然后在把HelloWebService.wsdl拖进选择what方法名称



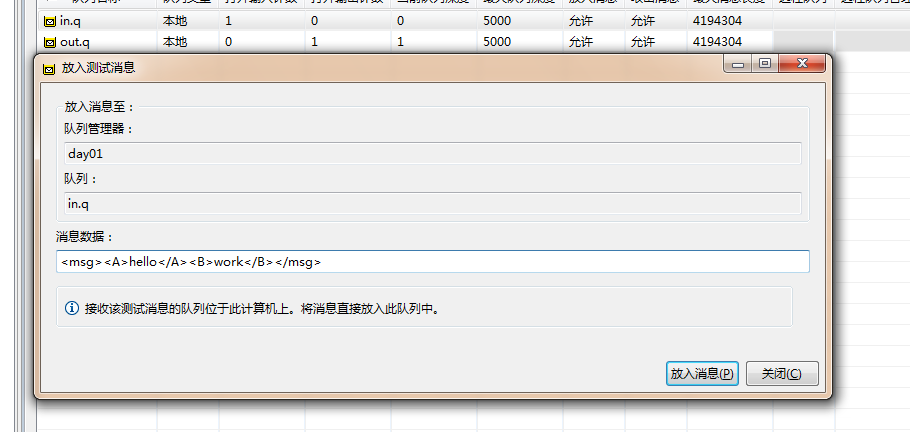
然后在添加一个Compute和MQOutput将这些节点连接起来（同样要把Compute属性选择上）



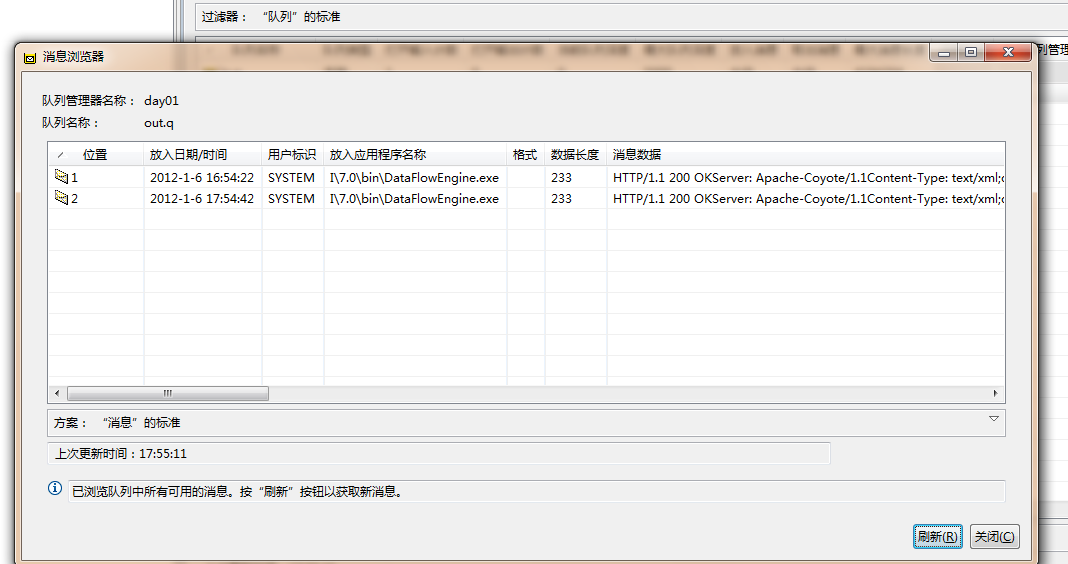
然后双击打开Compute编写代码



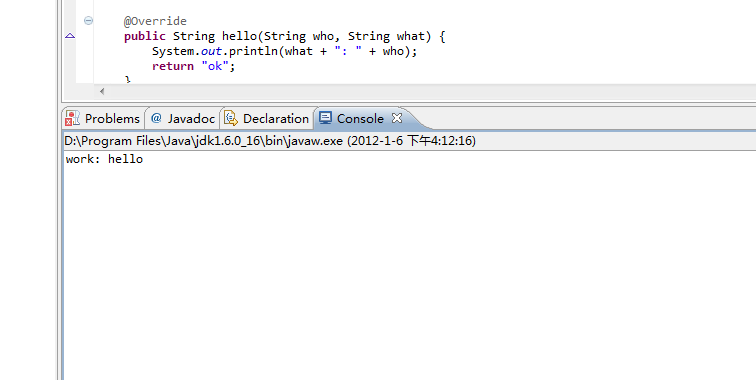
然后保存并且启动代理，可以在MQ 资源管理器中进行测试，在in.q中输入<msg><A>hello</A><B>work</B></msg>，同时也可以添加断点进行跟踪查看



然后在out.q中可以查看测试数据

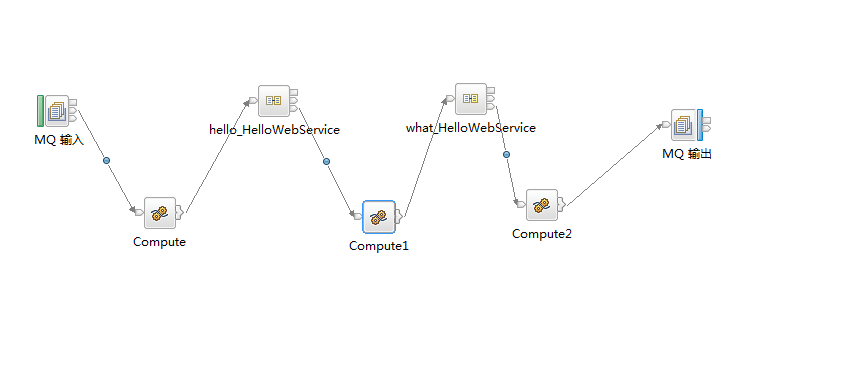


同时在eclipse中可以查看到输出的数据

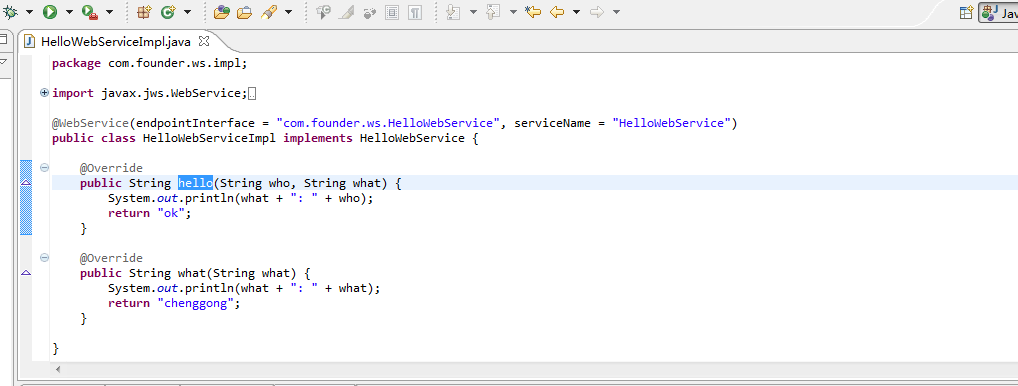


第三个例子 第一个返回值+第二个返回值，输出

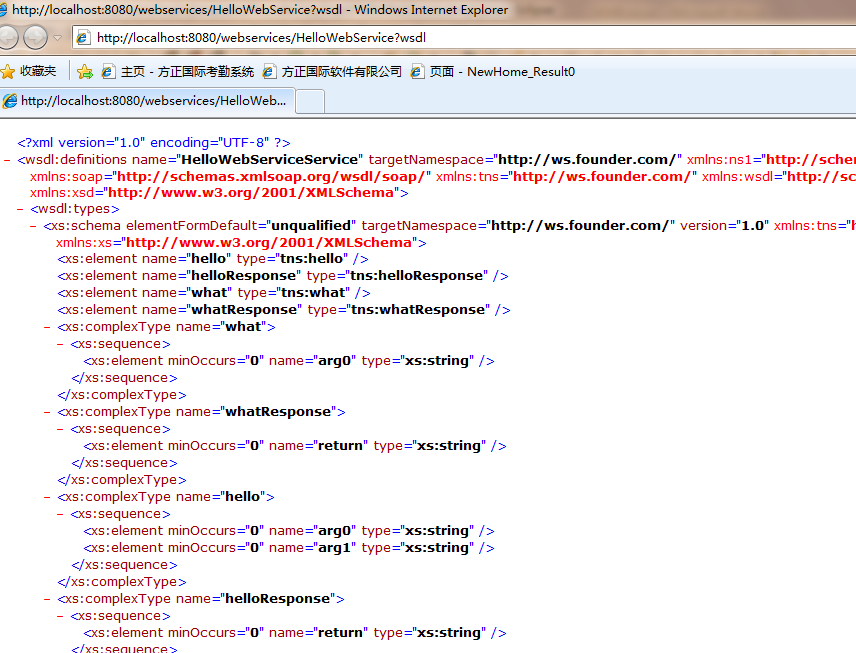
1. 创建一个新的项目
2. 把wsdl文件拖入选项版中，并且要拖入两次（一个是hello方法一个是what方法）选择不同的方法名称。添加一个MQINPUT、一个MQOUTPUT和三个Compute,并且连接起来



在eclipse中编写两个方法



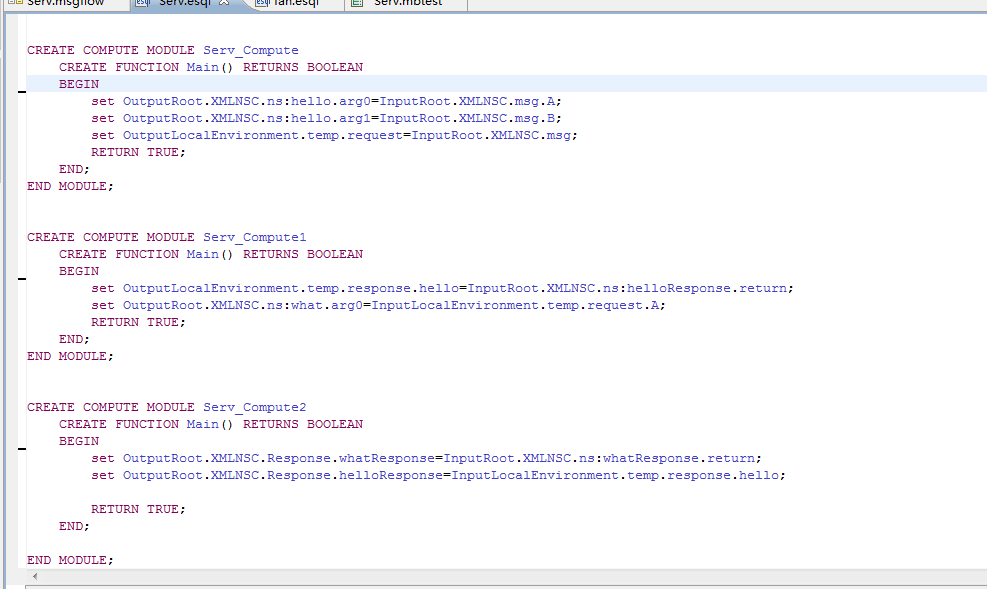
启动tomact，可以通过<http://localhost:8080/webservices/HelloWebService?wsdl>可以访问



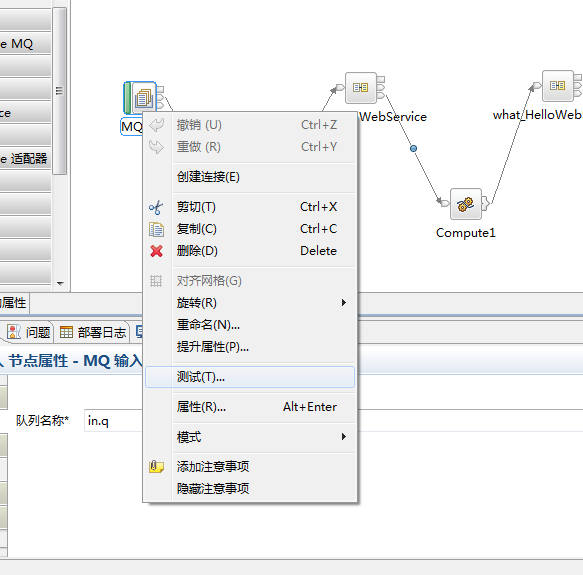
在input和output中记得修改队列名称，在input中要在“输入消息解析”中选择



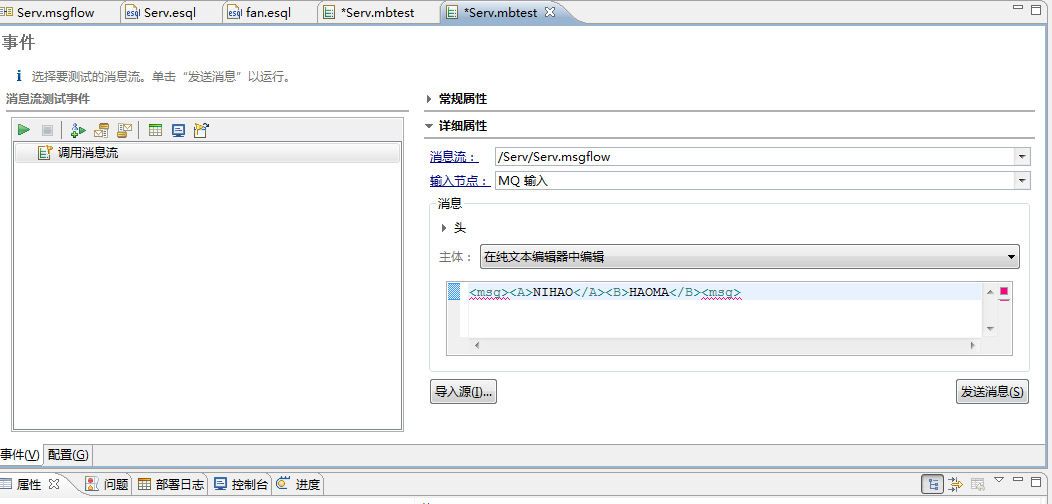
在Compute中要通过编码来控制输出结果



然后可以通过测试进行判断是否成功

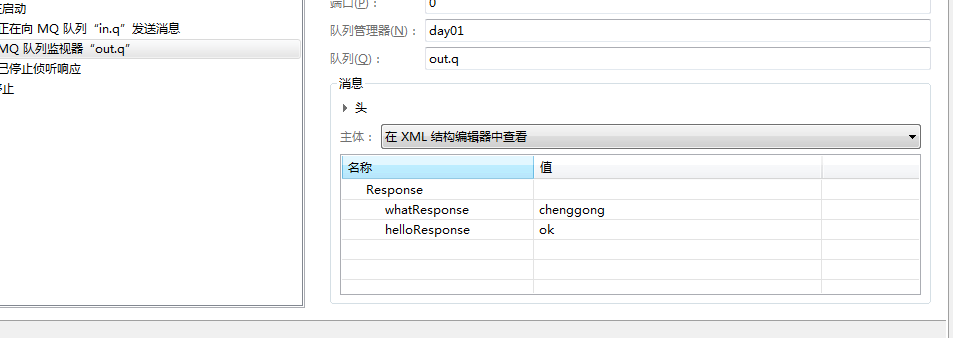


在测试消息中可以输入<msg><A>NIHAO</A><B>HAOMA</B><msg>进行测试



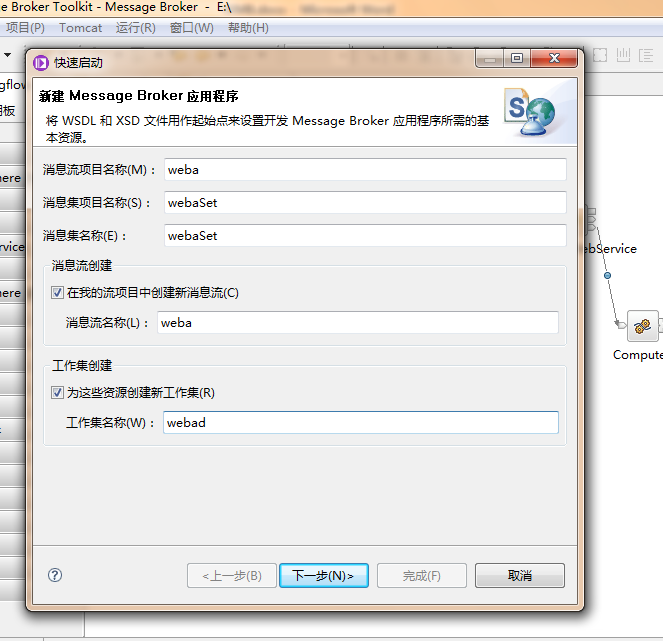
在测试结果中看到helloResponse ok 就表示成

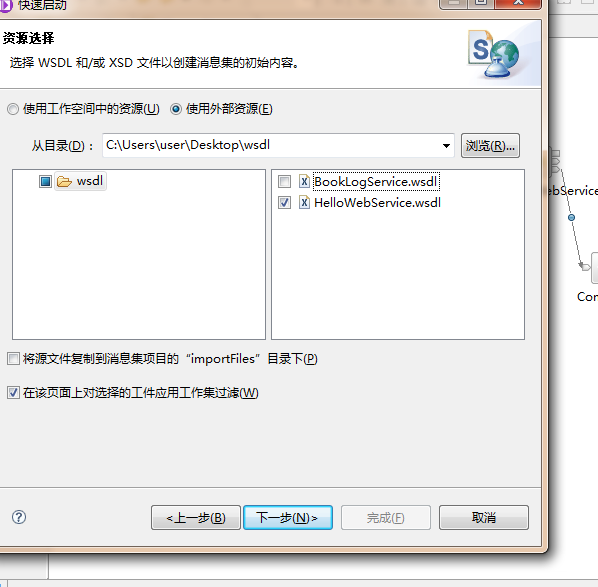
whatResponse chenggong 功了

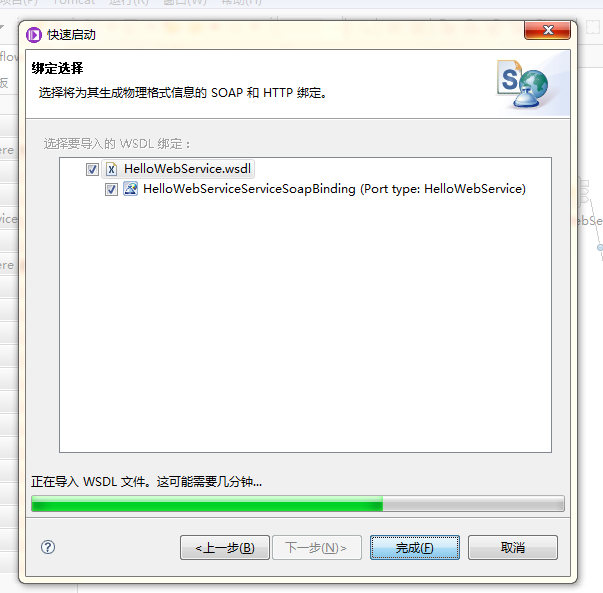


例子：通过Input节点传入4个值，用其中两个值作为参数调取webService方法的返回值，返回结果在加上4个值。一共返回5个结果

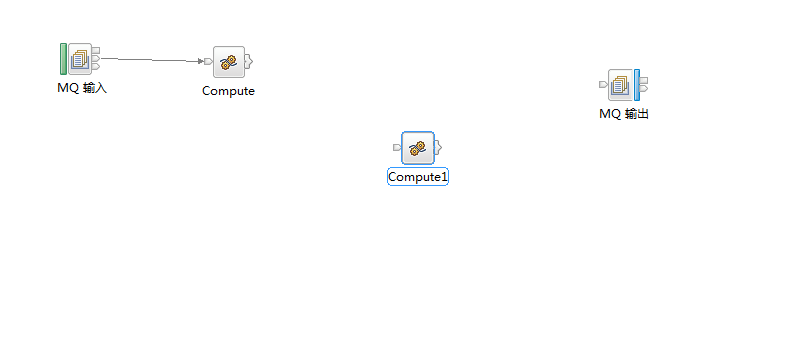
新建一个项目



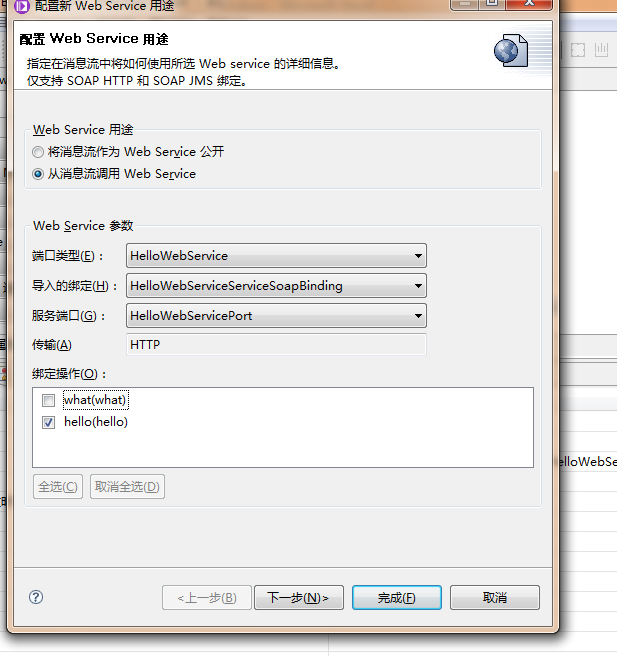




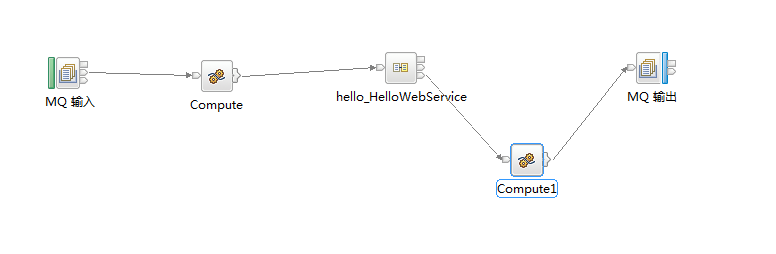
先添加一个Input、Output和两个Compute节点



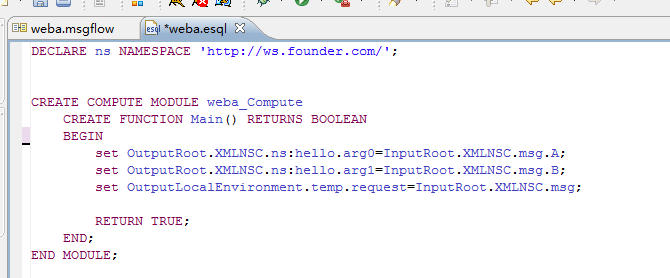
创建一个调用web Service的节点



将其5个节点连接起来，记得修改各个节点中的属性



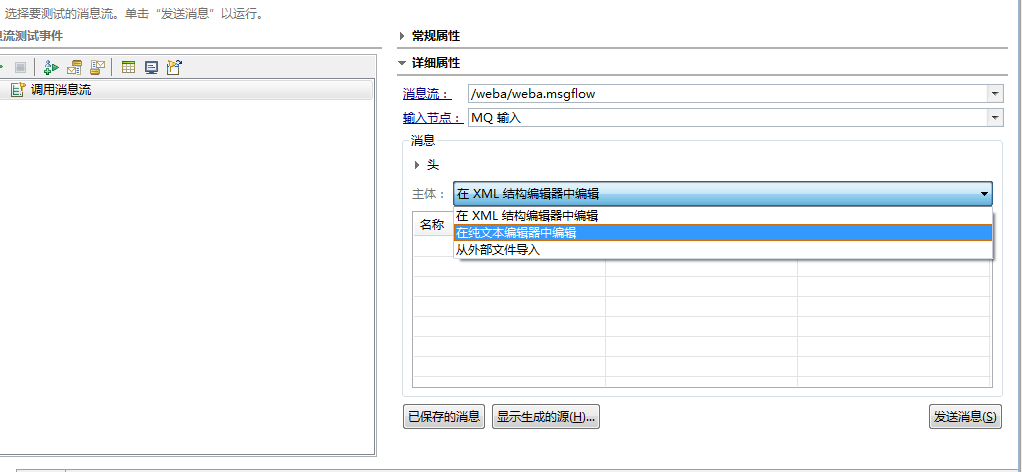
用Compute 1节点分解参数让A和B调去webService的方法



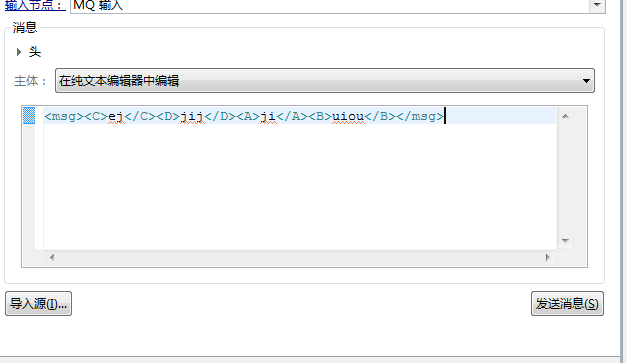
然后在用Compute1节点读取webSerivce返回的值和输入的4个值



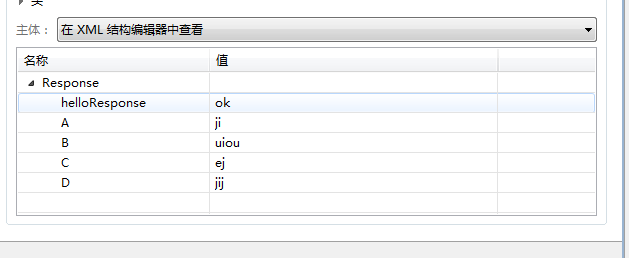
测试，在Input节点右键可见“测试”



输入<msg><C>ej</C><D>jij</D><A>ji</A><B>uiou</B></msg>

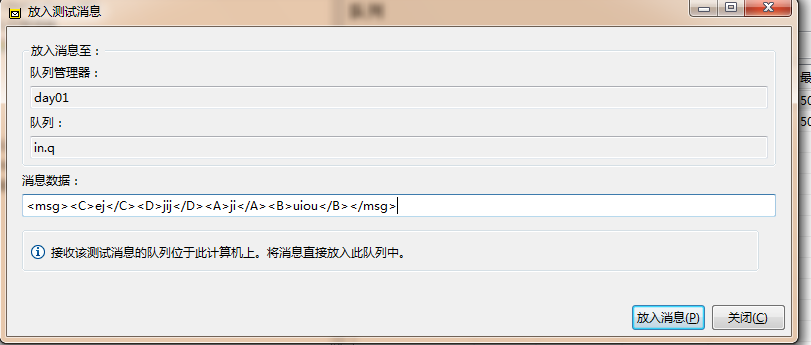


点击发送消息，结果为

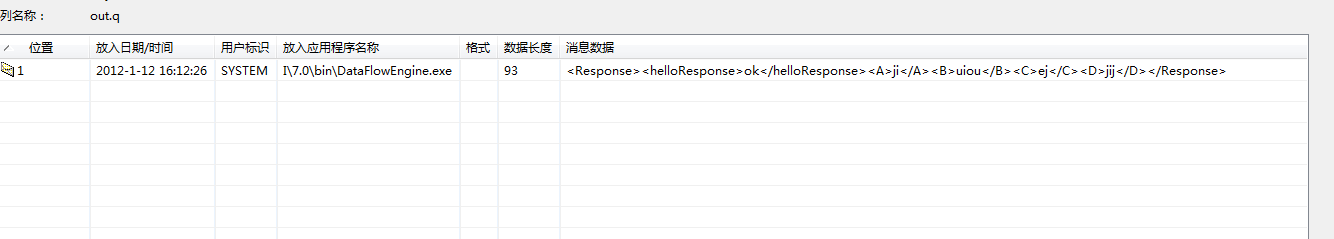


用MQ测试为

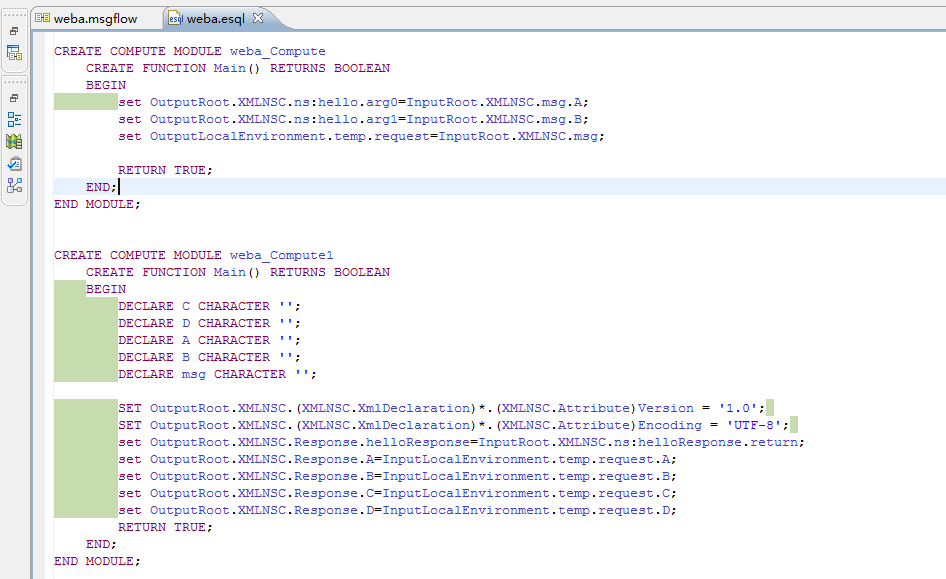
在in.q中放入消息



然后在out.q中浏览消息为



在输入的时候添加上<?xml version="1.0" encoding="UTF-8"?>让其输出

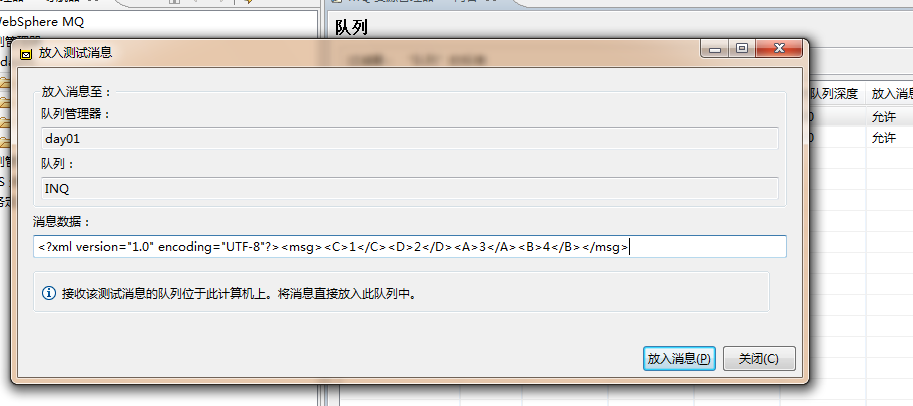


主要是这两句代码

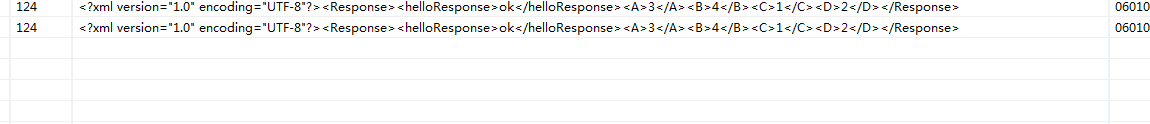
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)\*.(XMLNSC.Attribute)Version = '1.0';

SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)\*.(XMLNSC.Attribute)Encoding = 'UTF-8';

放入<?xml version="1.0" encoding="UTF-8"?><msg><C>1</C><D>2</D><A>3</A><B>4</B></msg>消息



输出消息为：



通过java代码测试主要为

String s="<msg><C>1</C><D>2</D><A>3</A><B>4</B></msg> <?xml version=\"1.0\" encoding=\"UTF-8\">";//定义字符创

msg.writeString(s);//输入字符串

在MQ\_Put\_Sample中添加，点击右键测试

**package** cn.ibm.com;

**import** com.ibm.mq.MQException;

**import** com.ibm.mq.MQMessage;

**import** com.ibm.mq.MQPutMessageOptions;

**import** com.ibm.mq.MQQueue;

**import** com.ibm.mq.MQQueueManager;

**import** com.ibm.mq.constants.MQConstants;

**import** cn.ibm.com.Message;

/\*\*

\* Simple example program

\*/

**public** **class** MQ\_Put\_Sample {

**private** **static** **final** String *qManager* = "day01";

**private** **static** **final** String *qName* = "INQ";

**public** **static** **void** main(String args[]) {

**try** {

MQQueueManager qMgr = **new** MQQueueManager(*qManager*);

**int** openOptions = MQConstants.*MQOO\_INPUT\_AS\_Q\_DEF*

| MQConstants.*MQOO\_OUTPUT*;

MQQueue queue = qMgr.accessQueue(*qName*, openOptions);

MQMessage msg = **new** MQMessage();

//String a= "<?xml version=\"1.0\" encoding=\"UTF-8\">";

String s="<msg><C>1</C><D>2</D><A>3</A><B>4</B></msg> <?xml version=\"1.0\" encoding=\"UTF-8\">";//定义字符创

msg.writeString(s);//输入字符串

MQPutMessageOptions pmo = **new** MQPutMessageOptions();

System.*out*.println("Sending a message...");

queue.put(msg, pmo);

queue.close();

System.*out*.println("Disconnecting from the Queue Manager");

qMgr.disconnect();

System.*out*.println("Done!");

} **catch** (MQException ex) {

System.*out*

.println("A WebSphere MQ Error occured : Completion Code "

+ ex.completionCode + " Reason Code "

+ ex.reasonCode);

ex.printStackTrace();

**for** (Throwable t = ex.getCause(); t != **null**; t = t.getCause()) {

System.*out*.println("... Caused by ");

t.printStackTrace();

}

} **catch** (java.io.IOException ex) {

System.*out*

.println("An IOException occured whilst writing to the message buffer: "

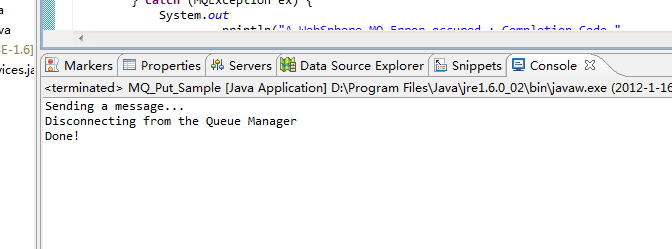
+ ex);

}

**return**;

}

}



在MQ\_Get\_Sample中用String msgText = rcvMessage.readLine();//输出值接收消息

**package** cn.ibm.com;

**import** com.ibm.mq.MQException;

**import** com.ibm.mq.MQGetMessageOptions;

**import** com.ibm.mq.MQMessage;

**import** com.ibm.mq.MQQueue;

**import** com.ibm.mq.MQQueueManager;

**import** com.ibm.mq.constants.MQConstants;

**public** **class** MQ\_Get\_Sample {

**private** **static** **final** String *qManager* = "day01";

**private** **static** **final** String *qName* = "OUTQ";

// MQGetMessageOptions gmo = new MQGetMessageOptions();

// gmo.matchOptions = MQC.MQMO\_NONE;

// gmo.options = MQC.MQGMO\_NO\_WAIT | MQC.MQGMO\_BROWSE\_NEXT;

// queue = mqm.accessQueue(qname, MQC.MQOO\_BROWSE | MQC.MQOO\_INQUIRE);

**public** **static** **void** main(String args[]) **throws** Exception {

**try** {

MQQueueManager qMgr = **new** MQQueueManager(*qManager*);

**int** openOptions = MQConstants.*MQOO\_INPUT\_AS\_Q\_DEF*

| MQConstants.*MQOO\_OUTPUT*;

MQQueue queue = qMgr.accessQueue(*qName*, openOptions);

MQMessage rcvMessage = **new** MQMessage();

MQGetMessageOptions gmo = **new** MQGetMessageOptions();

queue.get(rcvMessage, gmo);

String msgText = rcvMessage.readLine();//输出值

System.*out*.println("The message is: " + msgText);

queue.close();

qMgr.disconnect();

} **catch** (MQException ex) {

System.*out*

.println("A WebSphere MQ Error occured : Completion Code "

+ ex.completionCode + " Reason Code "

+ ex.reasonCode);

ex.printStackTrace();

**for** (Throwable t = ex.getCause(); t != **null**; t = t.getCause()) {

System.*out*.println("... Caused by ");

t.printStackTrace();

}

} **catch** (java.io.IOException ex) {

System.*out*

.println("An IOException occured whilst writing to the message buffer: "

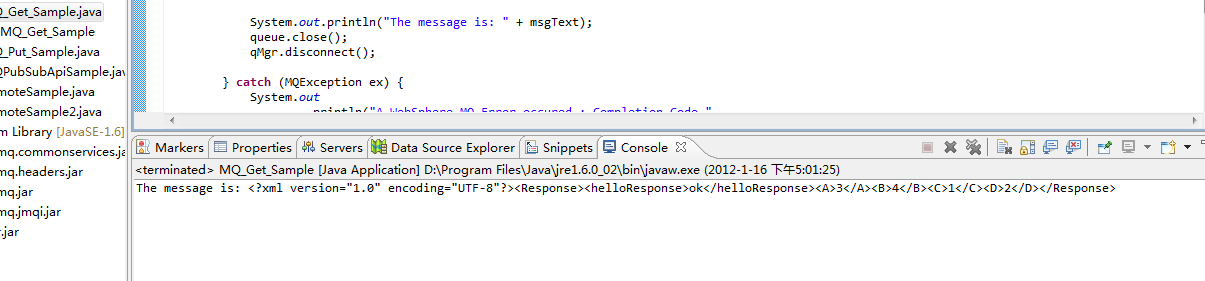
+ ex);

}

**return**;

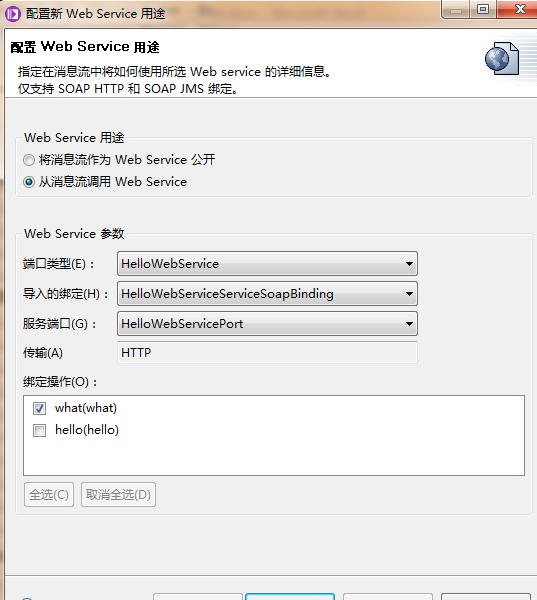
}

}

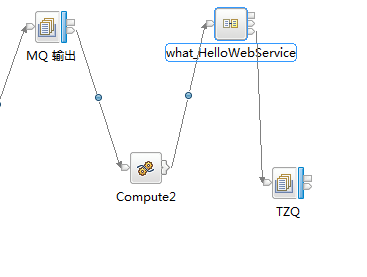


当在OUTPUT中有消息的时候，提示一下

首先建立一个webService

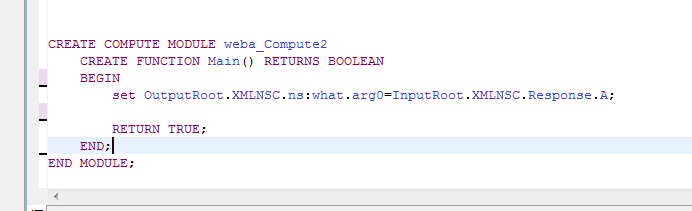


然后添加一个Compute节点让OUTPUT、Compute和刚刚新添加的what\_HelloWebService接口连接起来，在让what\_HelloWebService和TZQ节点连接起来

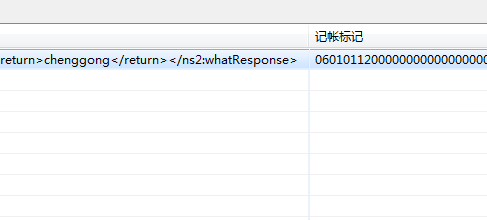


当OUPUT中有消息了，让其中一个值做为参数传入到what\_HelloWebService方法中，让TZQ输出

Compute2节点代码



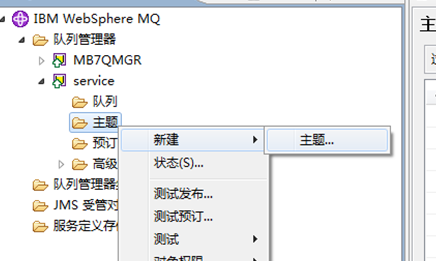
在MQ队列中建立一个TZQ，其输出结果为

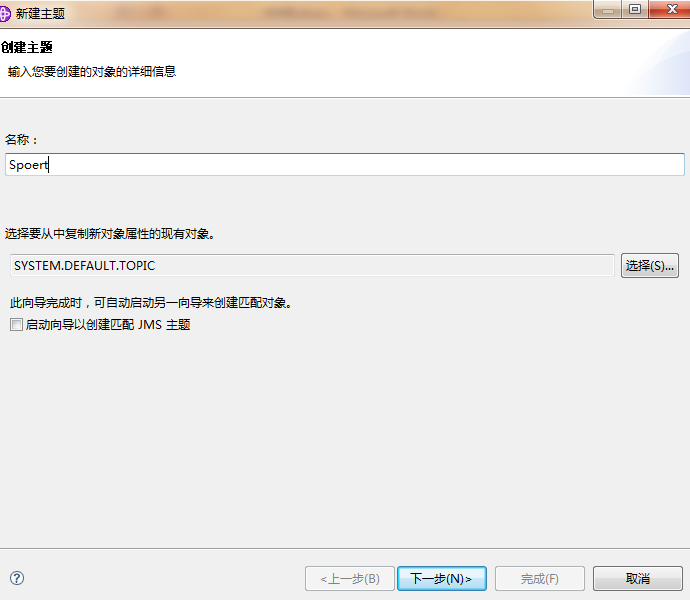


WMB订阅与发布

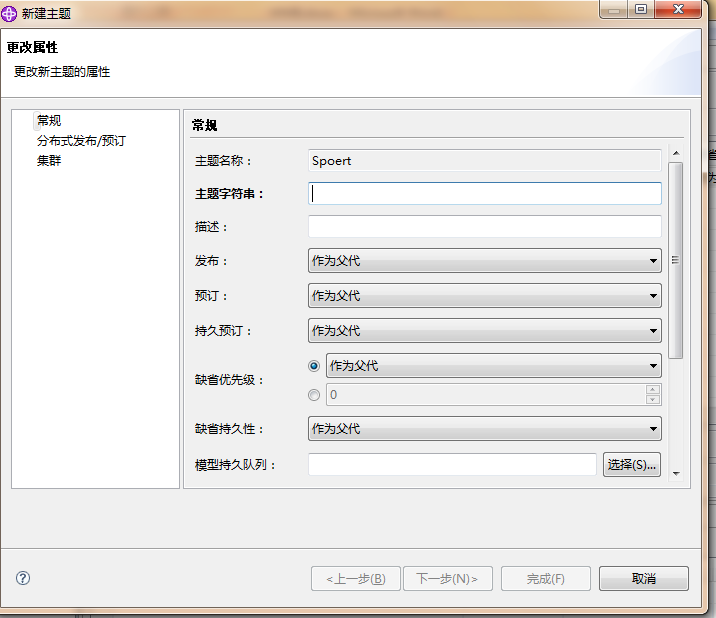
1. 使用MQ订阅与发布

首先在MQ队列管理其中建立一个主题，右键新建一个主题主题我们可以随便定义，例如：Spoert

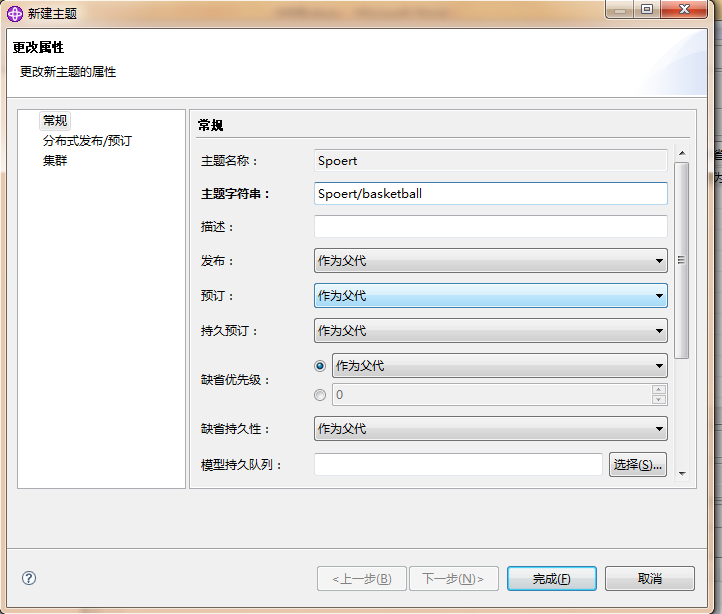




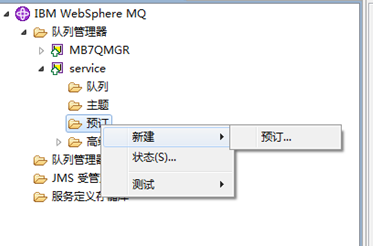
然后点击下一步

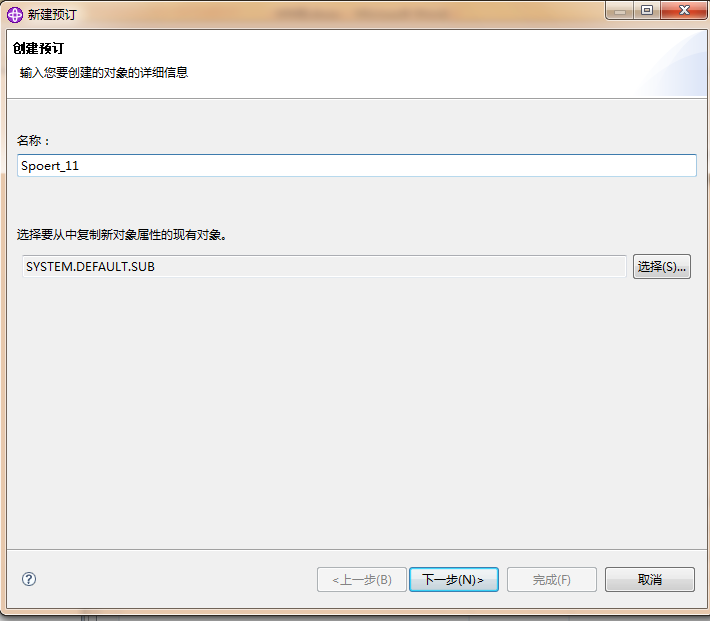


然后在“主题字符串”中添加Spoert/basketball，Spoert和basketball必须用/分割，然后点击完成

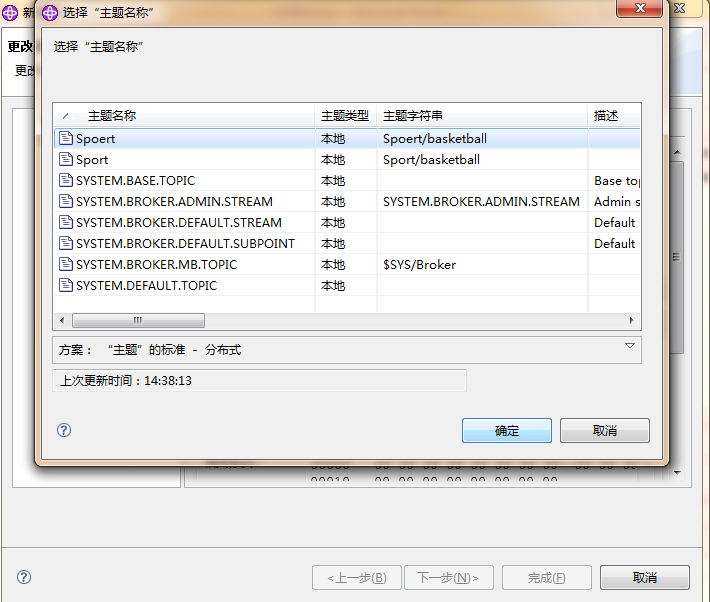


然后在预定主题中新建一个，可以随意起一个名字。例如：Spoert\_11

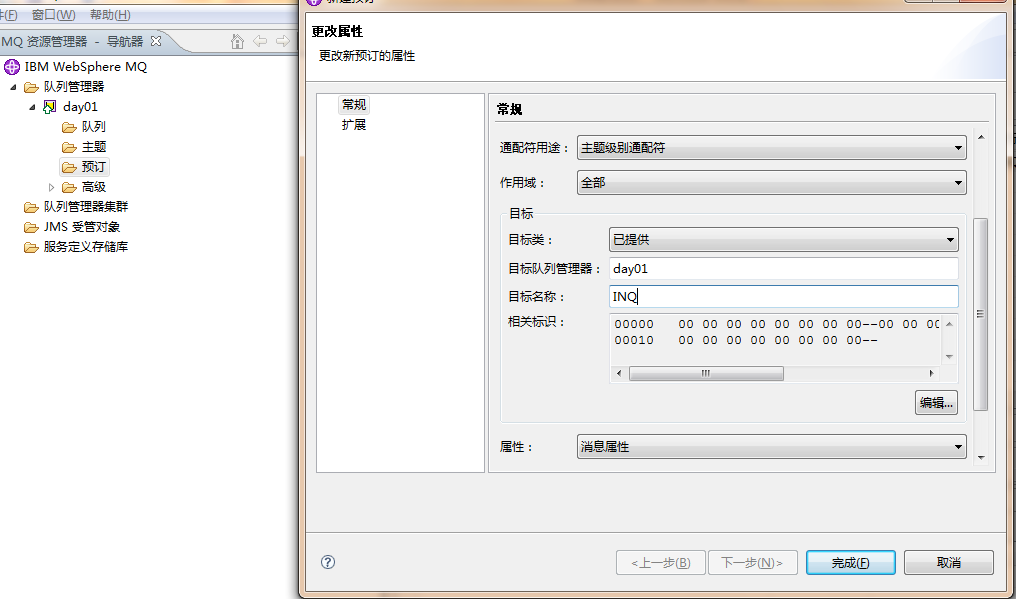




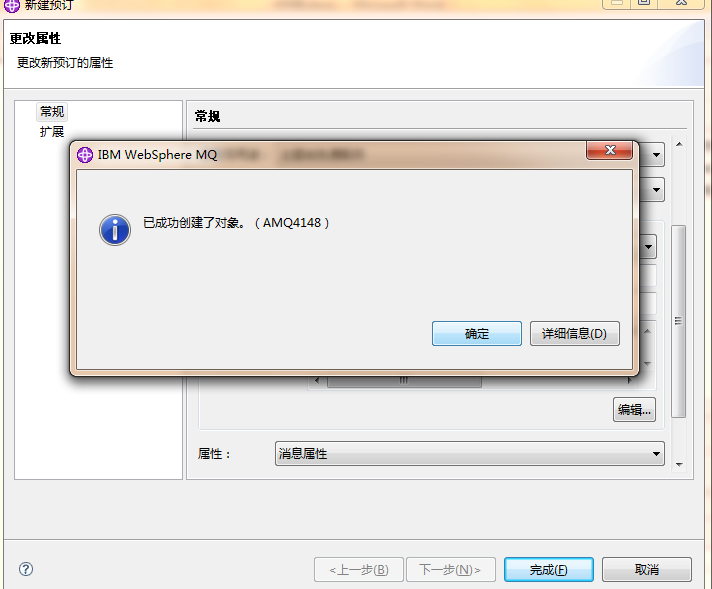
然后点击下一步，在主题名称中选择刚才建立的主题



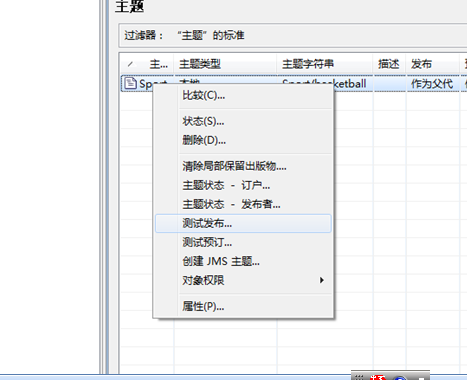
然后在目标队列管理器中填写，队列管理器和目标名称

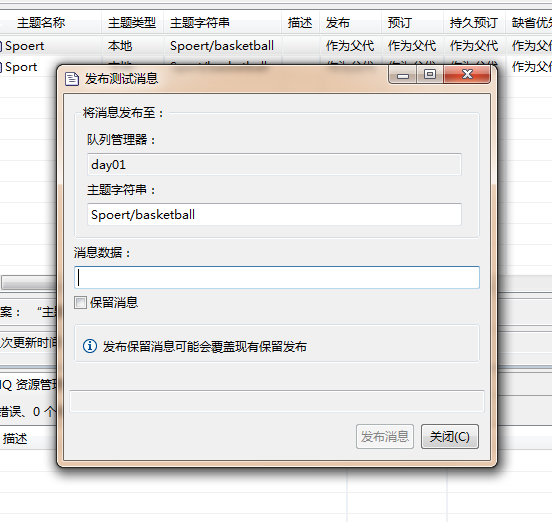


然后点击完成

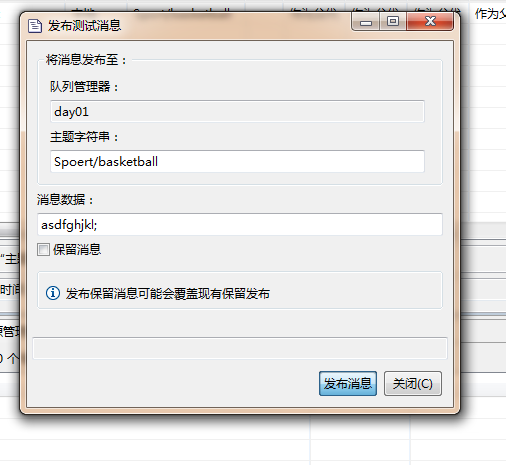


然后在主题中点击右键测试发布

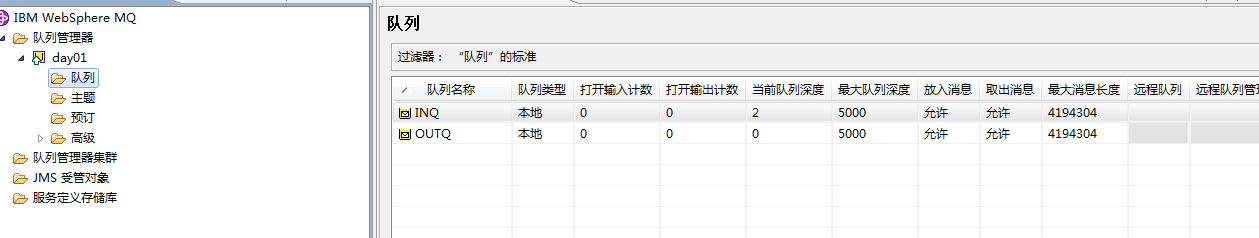




在消息数据中随便填写数据，然后发布出去



然后在队列中查看消息

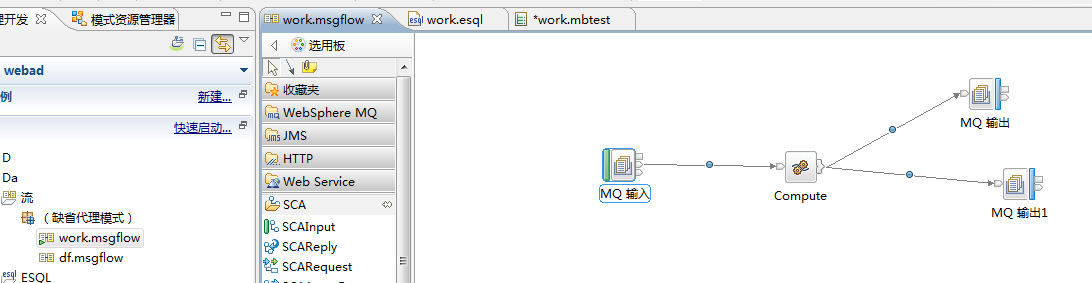


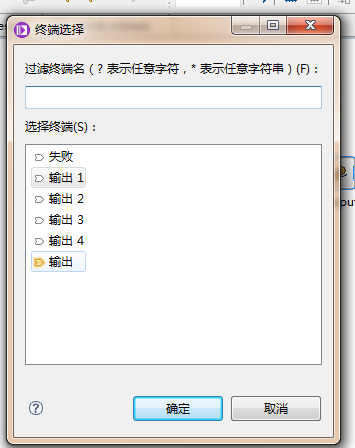
在INQ中浏览消息



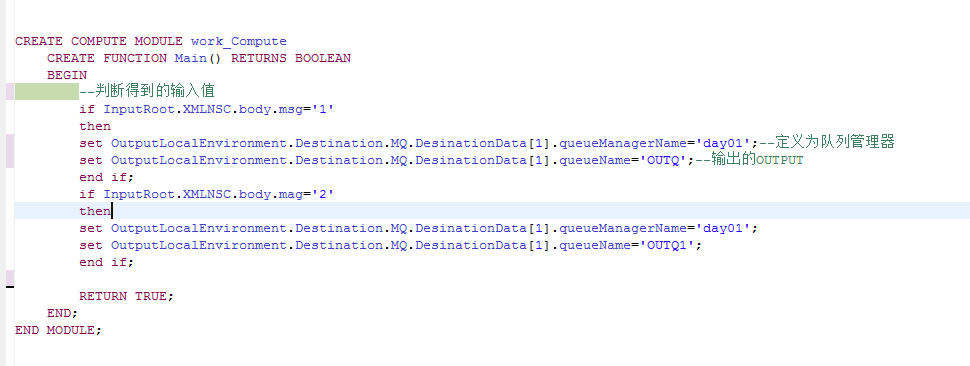
一个INPUT通过Compute节点对应多个OUTPUT

先建立一个项目流并在页面上添加一个INPUT、Compute和多个OUTPUT,在Compute节点中通过一个节点对应多个OUTPUT

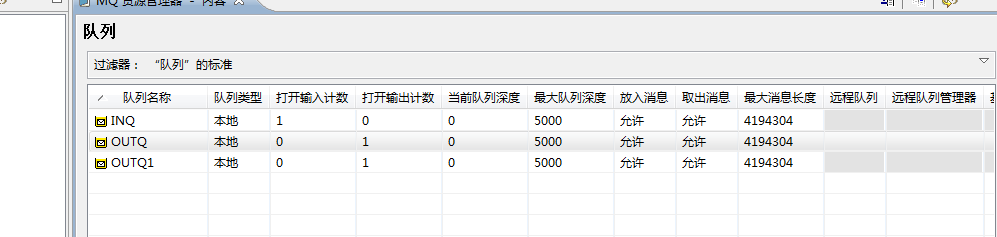




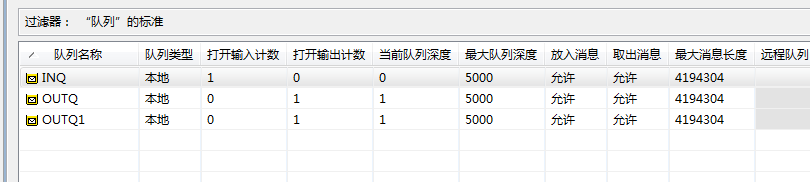
在ESQL中编写代码如：



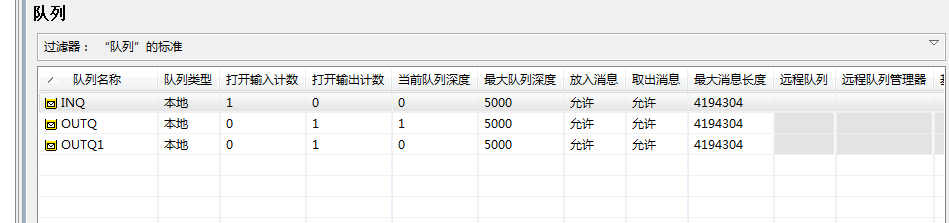
在资源管理器中建立一个INPUT和多个OUT



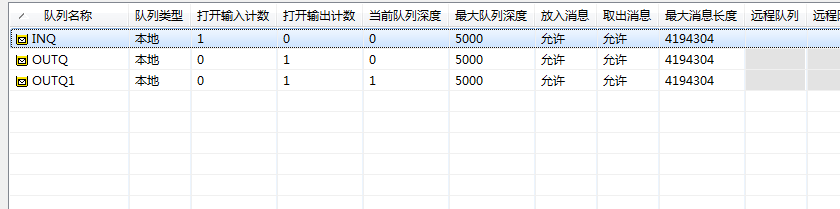
在INQ中输入<body><msg>1</msg><mag>2</mag></body>,输出结果为OUTQ和OUTQ1中都有结果



在INQ中输入<body><msg>1</msg></body>，可以在OUTQ中看到有结果

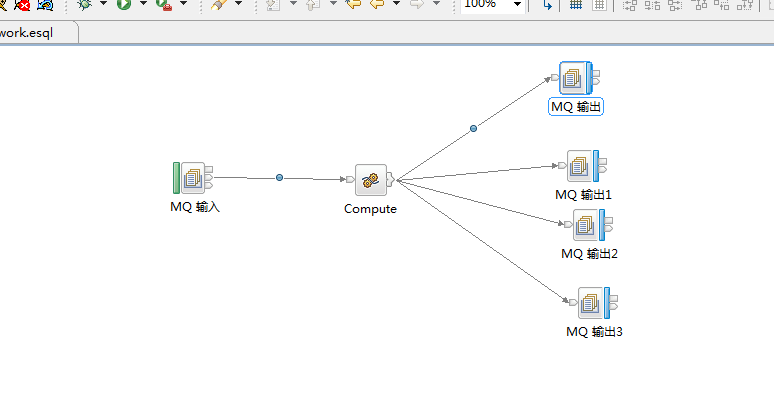


在INQ中输入<body><mag>2</mag></body>，可以在OUTQ1中看到有结果

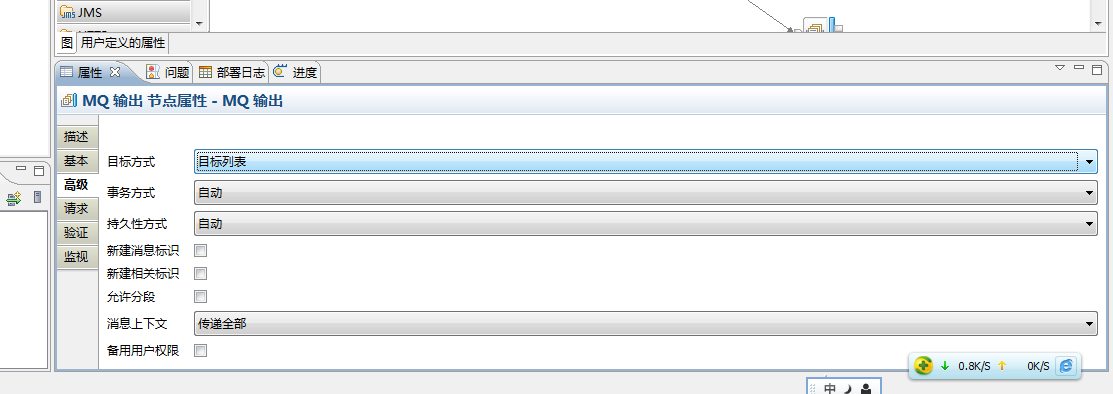


一个INPUT通过Compute节点对应4个OUTPUT

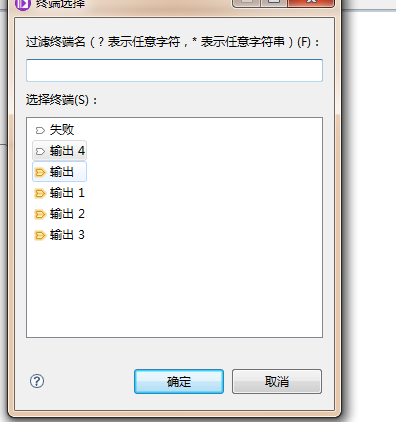
先建立一个项目流并在页面上添加一个INPUT、Compute和多个OUTPUT,在Compute节点中通过一个节点对应多个OUTPUT



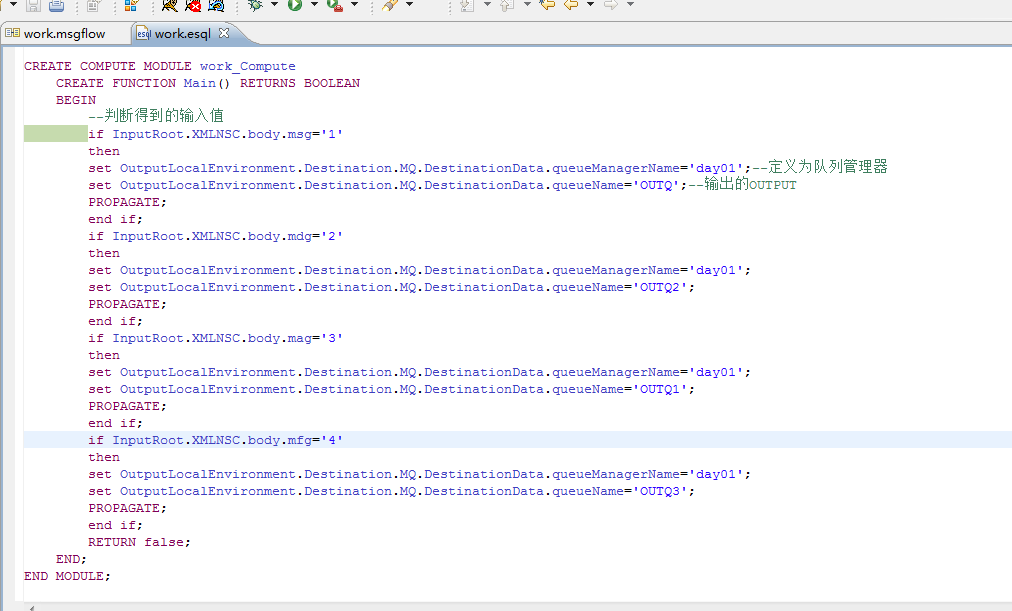
在OUT中配置中，不写队列名称，只在高级中把目标方式选择为“目标列表”



在Compute节点中一个输出对应一个OUT



在ESQL中编写的代码

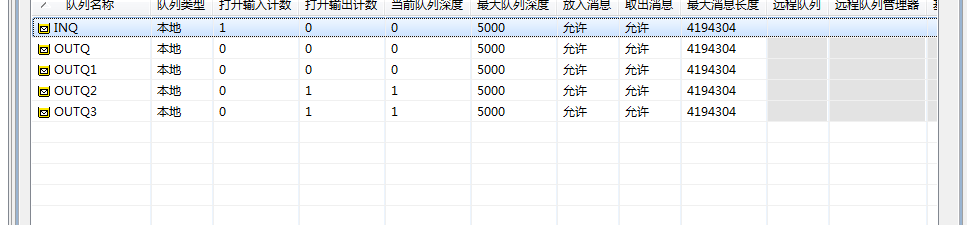


测试，当输入

<body><msg>1</msg><mag>3</mag></body>，显示结果为



如果输入<body><mdg>2</mdg><mfg>4</mfg></body>，显示结果为



定时取数据

放入数据的代码为MQ\_Put\_Sample

**package** cn.ibm.com;

**import** com.ibm.mq.MQException;

**import** com.ibm.mq.MQMessage;

**import** com.ibm.mq.MQPutMessageOptions;

**import** com.ibm.mq.MQQueue;

**import** com.ibm.mq.MQQueueManager;

**import** com.ibm.mq.constants.MQConstants;

/\*\*

\* Simple example program

\*/

**public** **class** MQ\_Put\_Sample {

**private** **static** **final** String *qManager* = "day01";

**private** **static** **final** String *qName* = "INQ";

**try** {

MQEnvironment.*hostname* = "172.18.73.29";

MQEnvironment.*channel* = "CLIENT.QM\_SRV";

MQEnvironment.*port* = 1415;

MQEnvironment.*CCSID* = 1208;

MQEnvironment.*properties*.put(~~MQC~~.*TRANSPORT\_PROPERTY*, ~~MQC~~.*TRANSPORT\_MQSERIES*);

MQQueueManager qMgr = **new** MQQueueManager(*qManager*);

**int** openOptions = MQConstants.*MQOO\_INPUT\_AS\_Q\_DEF*

| MQConstants.*MQOO\_OUTPUT* | MQConstants.*MQOO\_INQUIRE*;

MQQueue queue = qMgr.accessQueue(*qName*, openOptions,**null**, **null**,

**null**);

MQMessage msg = **new** MQMessage();

String s="<msg><C>1</C><D>2</D><A>3</A><B>4</B></msg>";

msg.writeString(s);

MQPutMessageOptions pmo = **new** MQPutMessageOptions();

System.*out*.println("Sending a message...");

queue.put(msg, pmo);

queue.close();

System.*out*.println("Disconnecting from the Queue Manager");

qMgr.disconnect();

System.*out*.println("Done!");

} **catch** (MQException ex) {

System.*out*

.println("A WebSphere MQ Error occured : Completion Code "

+ ex.completionCode + " Reason Code "

+ ex.reasonCode);

ex.printStackTrace();

**for** (Throwable t = ex.getCause(); t != **null**; t = t.getCause()) {

System.*out*.println("... Caused by ");

t.printStackTrace();

}

} **catch** (java.io.IOException ex) {

System.*out*

.println("An IOException occured whilst writing to the message buffer: "

+ ex);

}

**return**;

}

}

在原有weba基础上进行添加，在java承训中的MQ\_Client中添加一个方法

方法名为：Timera

**package** cn.ibm.com;

**import** com.ibm.mq.~~MQC~~;

**import** com.ibm.mq.MQException;

**import** com.ibm.mq.MQGetMessageOptions;

**import** com.ibm.mq.MQMessage;

**import** com.ibm.mq.MQQueue;

**import** com.ibm.mq.MQQueueManager;

**import** com.ibm.mq.constants.MQConstants;

**public** **class** Timera **extends** java.util.TimerTask {

**private** **static** **final** String *qManager* = "day01";

**private** **static** **final** String *qName* = "OUTQ";

// MQGetMessageOptions gmo = new MQGetMessageOptions();

// gmo.matchOptions = MQC.MQMO\_NONE;

// gmo.options = MQC.MQGMO\_NO\_WAIT | MQC.MQGMO\_BROWSE\_NEXT;

// queue = mqm.accessQueue(qname, MQC.MQOO\_BROWSE | MQC.MQOO\_INQUIRE);

@Override

**public** **void** run() {

**try** {

Hashtable ht = **new** Hashtable();

ht.put(~~MQC~~.*PORT\_PROPERTY*, 1415);

ht.put(~~MQC~~.*CCSID\_PROPERTY*, 1208);

ht.put(~~MQC~~.*HOST\_NAME\_PROPERTY*, "172.18.73.29");

ht.put(~~MQC~~.*CHANNEL\_PROPERTY*, "CLIENT.QM\_SRV");

ht.put(~~MQC~~.*TRANSPORT\_PROPERTY*, ~~MQC~~.*TRANSPORT\_MQSERIES*);

MQQueueManager qMgr = **new** MQQueueManager(*qManager*);

**int** openOptions = MQConstants.*MQOO\_INPUT\_AS\_Q\_DEF*

| MQConstants.*MQOO\_OUTPUT*;

MQQueue queue = qMgr.accessQueue(*qName*, openOptions);

MQMessage rcvMessage = **new** MQMessage();

MQGetMessageOptions gmo = **new** MQGetMessageOptions();

queue.get(rcvMessage, gmo);

String msgText = rcvMessage.readLine();//输出值

System.*out*.println("The message is: " + msgText);

queue.close();

qMgr.disconnect();

} **catch** (MQException ex) {

System.*out*

.println("A WebSphere MQ Error occured : Completion Code "

+ ex.completionCode + " Reason Code "

+ ex.reasonCode);

ex.printStackTrace();

**for** (Throwable t = ex.getCause(); t != **null**; t = t.getCause()) {

System.*out*.println("... Caused by ");

t.printStackTrace();

}

} **catch** (java.io.IOException ex) {

System.*out*

.println("An IOException occured whilst writing to the message buffer: "

+ ex);

}

**return**;

}

}

然后在MQ\_Get\_Sample设置间隔多少秒和等待多长时间

**package** cn.ibm.com;

**import** java.io.IOException;

**import** java.util.Timer;

**public** **class** MQ\_Get\_Sample {

**public** **static** **void** main(String args[]) **throws** Exception {

Timer timer = **new** Timer();

timer.schedule(**new** Timera(), 30000,60000);//运行30秒，间隔1分钟

/\* while(true){

try{

int ch=System.in.read();

if(ch-'c'==0){

timer.cancel();

}

}catch(IOException e){

e.printStackTrace();

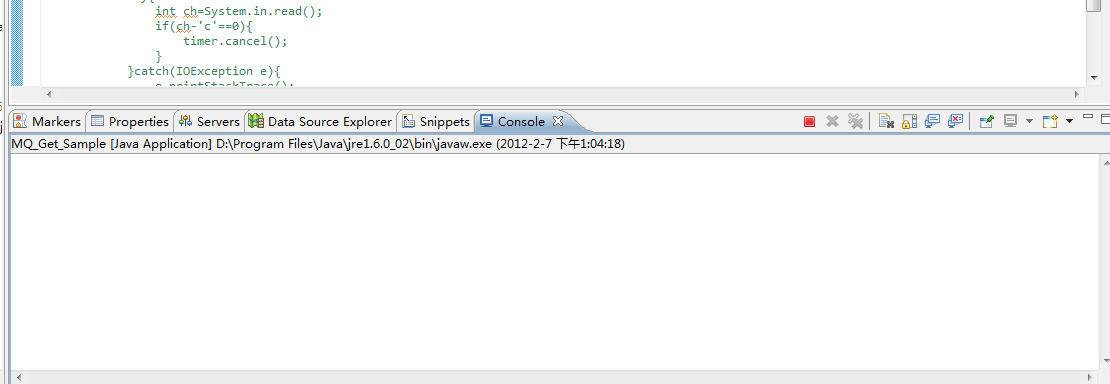
}

}\*/

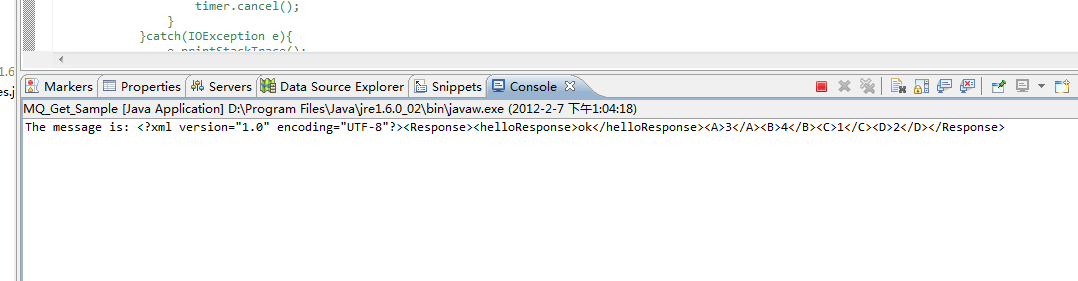
}

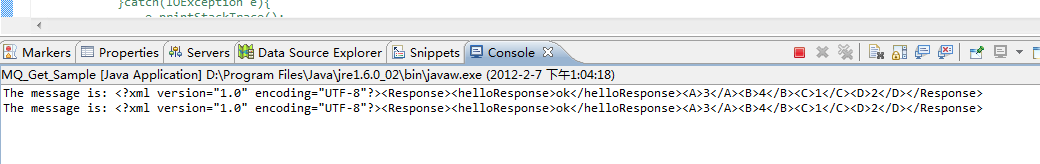
}

运行MQ\_Get\_Sample类中的方法



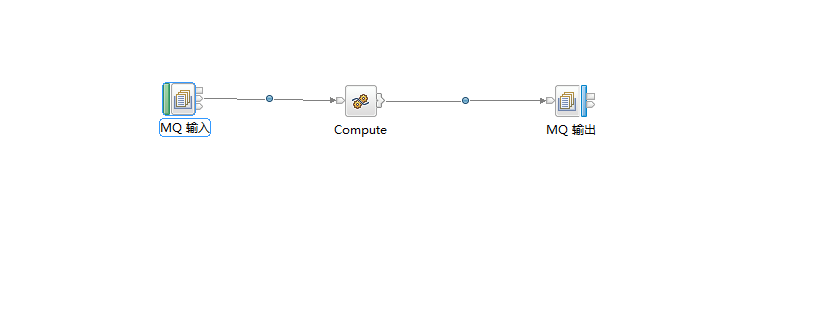
在INQ中输入<?xml version="1.0" encoding="UTF-8"?><msg><C>1</C><D>2</D><A>3</A><B>4</B></msg>，然后MQ\_Get\_Sample中会得到。



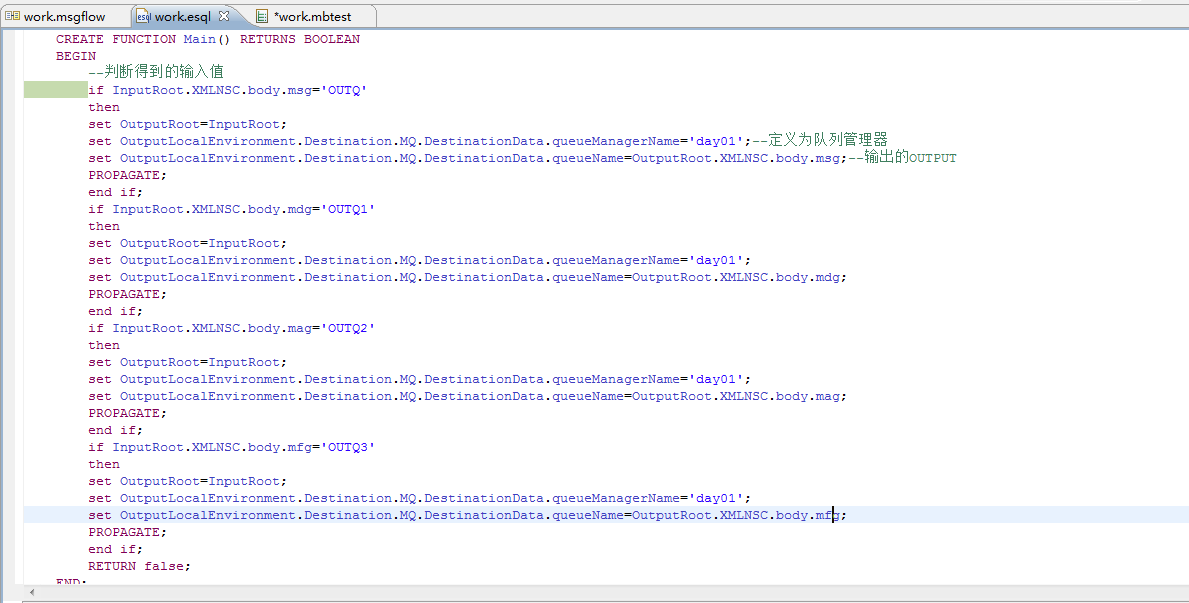


当传入数据后动态接收到OUTPUT中

在原有基础上修改ESQL语句，流程图



修改的ESQL语句



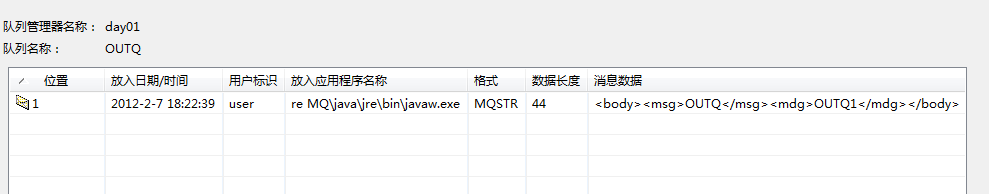
测试，输入单个<body><msg>OUTQ</msg></body>

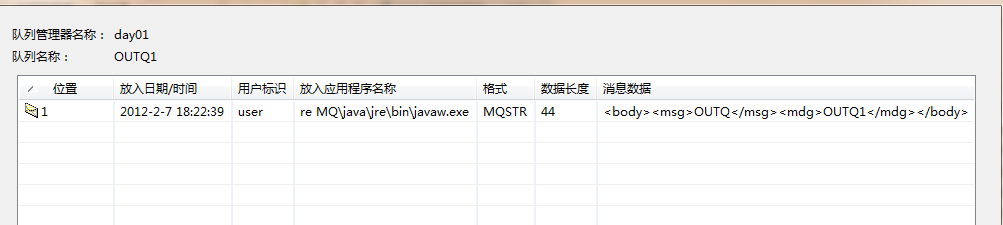




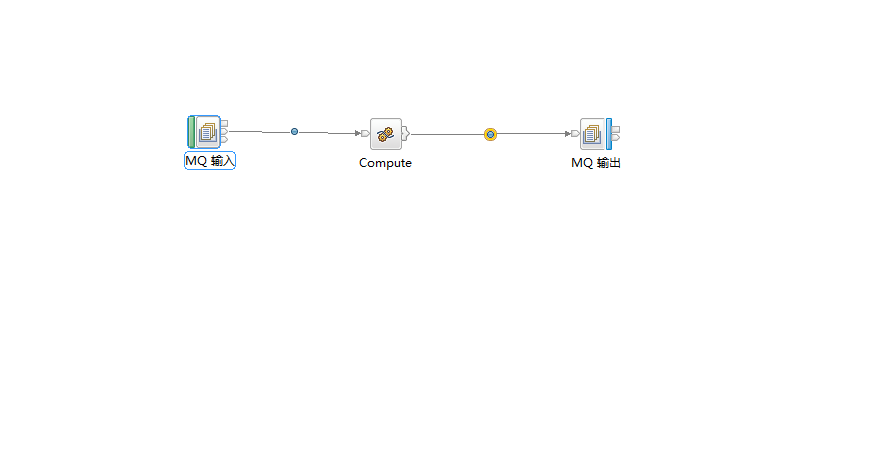
输入多个参数<body><msg>OUTQ</msg><mdg>OUTQ1</mdg></body>







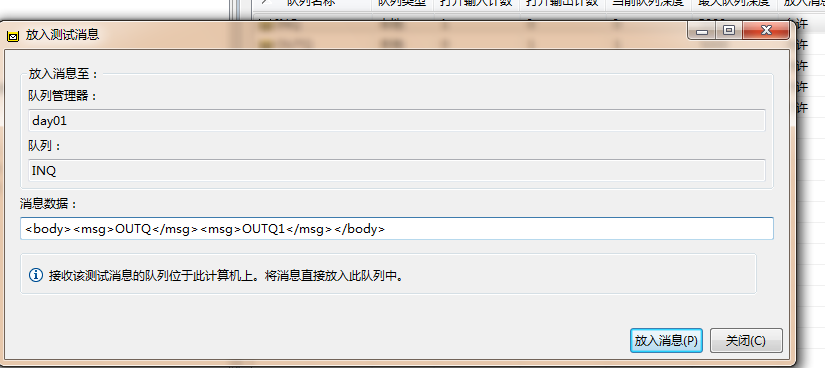
通过while循环进行遍历输出



ESQL代码的编写



测试，在INQ中输入<body><msg>OUTQ</msg><msg>OUTQ1</msg></body>



输出效果为：

