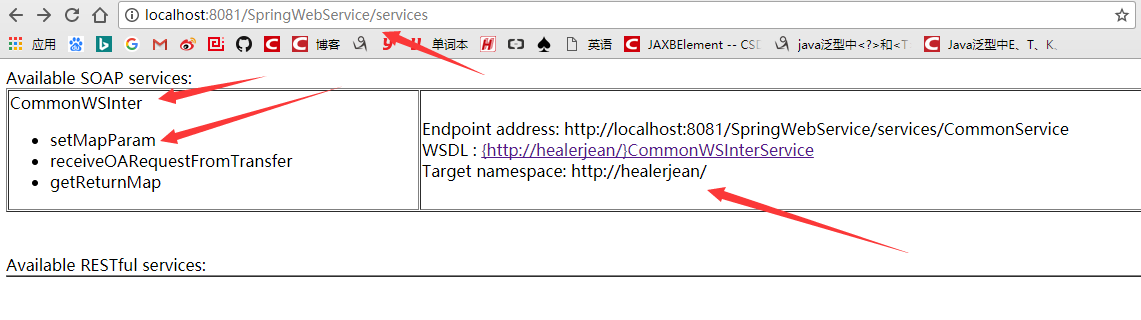
此文章已于 1:17:27 2017/11/18 发布到 HealerJean梦想博客

利用反射调用webservice

类别 webservice ; 反射

# 1、建立一个项目，专门用来发布服务，tomcat端口为8081，服务如下

## 1、webservice发布在tomcat上

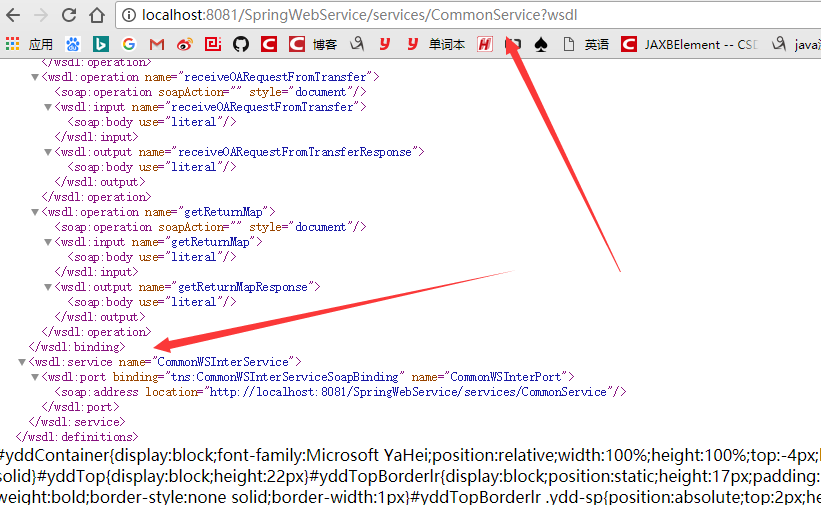


## 2、点开链接wsdl

<http://localhost:8081/SpringWebService/services/CommonService?wsdl>

### 1、注意下面的serviceName 为CommonWSInterService

### 2、portName 为CommonWSInterPort



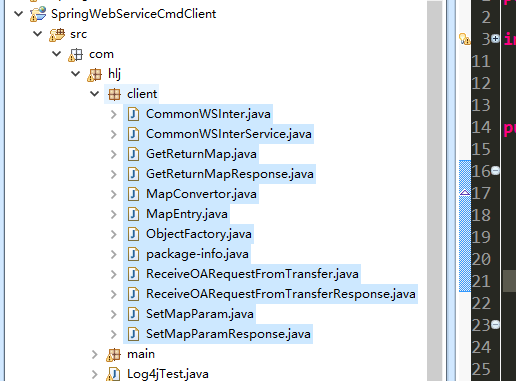
### 3、发布的代码为

|  |
| --- |
| @WebService(targetNamespace="http://healerjean/")  **public** **interface** CommonWSInter {  **public** String **setMapParam**(@XmlJavaTypeAdapter(MapAdapter.**class**)Map<String ,Object> map) ;      } |

### 4、实现的类为，最后返回Success

|  |
| --- |
| **public** **class** CommonWSImpl **implements** CommonWSInter{    /\*\*  \* 测试传入的参数为map类型  \*/  @Override  **public** String **setMapParam**(Map<String, Object> map) {    System.out.println(map.toString());    **return** "Success";  }    } |

# 2、生产客户端



# 3、设计WsDestEO，用来存储远程的webservice信息。制造这个list，这样选择pkWsDest，就可以选择要调用哪个服务

|  |
| --- |
| **public** **class** WsDestEO {  **private** String pkWsDest;  **private** String destName;  **private** String isActive;  **private** String wsdlAddr;  **private** String namespace;  **private** String serviceName;  **private** String portName;  **private** String objFacy;  **private** String userCode;  **private** String password;  **private** String memo;  **private** Date ts;  **public** String **getPkWsDest**() {  **return** pkWsDest;  } |

# 4、启动spring时候，加载上面的WsDestEO，这样只要spring容器一直存在，就能够使用上面的对象，也就是远程服务的参数。

## 1、创建接口IWsService

|  |
| --- |
| **public** **interface** IWsService {  **public** WsDestInfo **getWsDestInfoByKey**(String key);    } |

## 2、实现接口WsServiceImp利用@PostConstructs，使spring启动即可加载（非常重要），使用list加载WsDestEO，这样可以加载多个，下面用来2个，第一个是真实使用的，第二个是为了试验效果

|  |
| --- |
| @Service  **public** **class** WsServiceImp **implements** IWsService {    **private List<WsDestInfo> wsDestInfos = new ArrayList<WsDestInfo>();**  /\*\*  \* 初始化  \* **@author** :HealerJean:  \* **@date** :2017年11月17日 下午4:44:48  \* **@Description**:  \* void  \*/  @SuppressWarnings("null")  @PostConstruct  **public** **void** **init**() {    List<WsDestEO> wsDestEOs = **new** ArrayList<WsDestEO>();  WsDestEO wsDestEOOne = **new** WsDestEO(  "trustee\_healerjean", // pkWsDest  "传输系统职业年金发送服务",  "Y", //选择是否可用，Y表示可用 "http://localhost:8081/SpringWebService/services/CommonService?wsdl", // wsdlAddr  "http://healerjean/", // namespace  "CommonWSInterService", // serviceName  "CommonWSInterPort", // portName  "com.hlj.client.ObjectFactory", // objFacy  "dicp", // userCode 这里用不到，随便写的，可以没有  "dicp", // password 这里用不到，随便写的，可以没有  " ", // memo 这里用不到，随便写的，可以没有  **new** Date());  WsDestEO wsDestEOTwo = **new** WsDestEO(  "trustee\_healerjean2",  "传输系统职业年金发送服务",  "N", "http://localhost:8080/SpringWebService/services/CommonService?wsdl",  "http://healerjean/",  "BusinessReceiveServices",  "BusinessReceiveServicesPort",  "com.reflect.client.ObjectFactory",  "dicp",  "dicp",  " ",  **new** Date());  wsDestEOs.add(wsDestEOOne);  **for** (WsDestEO wsDestEO : wsDestEOs) {  **try** {  WsDestInfo wsDestInfo = **new** WsDestInfo();  wsDestInfo.setWsDestEO(wsDestEO);  wsDestInfo.init(); // 初始化Dispatch对象  wsDestInfos.add(wsDestInfo);  } **catch** (Exception e) {  e.printStackTrace();  // 打印后吃掉异常，保证程序正常启动  }  }  }    @Override  **public** WsDestInfo **getWsDestInfoByKey**(String key) {  **for** (WsDestInfo wsDestInfo : wsDestInfos) {  **if** (wsDestInfo.getWsDestEO().getPkWsDest().equals(key)) {  **return** wsDestInfo;  }  }  **return** **null**;  }    } |
|  |

# 5、设计发送的sender（非常重要）

## 1、getWsDestInfo通过pkWsDest可以找到WsDestInfo，也就是服务信息

## 2、WsInvokeResult，为invoke反射之后，生成的，这里的代码会在send之后触发，具体看后文，WsDestInfo

## 3、接口里面的参数SetMapParam为发布服务里面的方法的名字，这里客户端import时候将它制造为一个类了，下面QName为方法名字，首字母要小写哦

**4、WsResponse为反射调用远程服务的返回值里面的**setReturnObject**为**setMapParamResponse，**客户端的方法返回值**

|  |
| --- |
| /\*\*  \* **@author** : HealerJean  \* **@date** ： 2017年11月17日 上午9:26:02  \* **@Description**: 渠道调用受托的服务。用来交给受托进行代理人发来数据的业务校验  \*/  @Component  **public** **class** SetMapParamSender **extends** WsSender<SetMapParam>{    @Override  **public** WsDestInfo **getWsDestInfo**() {  **return** wsService.getWsDestInfoByKey("trustee\_healerjean");  }  @Override  **public** QName **getOperationQName**() {  **return** **new** QName(getWsDestInfo().getWsDestEO().getNamespace(), "setMapParam");  }      @Override  **protected** WsResponse **handleInvokeResult**(WsInvokeResult wsInvokeResult) {  WsResponse response = **new** WsResponse();  **if** (YesOrNo.YES.getCode().equals(wsInvokeResult.getInvokeResult())) {  SetMapParamResponse setMapParamResponse = (SetMapParamResponse) wsInvokeResult.getReturnObject();  /\*  String result = "01".equals(setMapParamResponse.getReturn()) ? YesOrNo.YES.getCode() : YesOrNo.NO.getCode();  String respInfo = YesOrNo.YES.getCode().equals(result) ? "处理成功" : "处理失败：" + receiveDataOrFeedbackFromInteResponse.getReturn().getMsg();  response.setRespResult(result);  response.setRespInfo(respInfo);\*/  response.setReturnObject(setMapParamResponse);  } **else** {  response.setRespResult(YesOrNo.NO.getCode());  response.setRespInfo("调用失败：" + wsInvokeResult.getInvokeInfo());  }  **return** response;  }  } |

# 5、（重中之重）4中抽象父类send方法就在这里面，invoke也在，并且有返回值，

|  |
| --- |
| **public** **abstract** **class** WsSender<**T**> {    @Autowired  **protected** IWsService wsService;  @Autowired  **private** WsInvoker invoker;  **private** String senderName;    **public** **WsSender**() {  String simpleName = getClass().getSimpleName();  senderName = simpleName.substring(0, 1).toLowerCase() + simpleName.substring(1);  }    /\*\*  \* 默认为短类名首字母小写  \* **@return**  \*/  **protected** String **getSenderName**() {  **return** senderName;  }    **public** **abstract** WsDestInfo **getWsDestInfo**();    **public** **abstract** QName **getOperationQName**();    /\*\*  \* 处理调用结果  \* **@param** wsInvokeResult 调用结果  \* **@return**  \*/  **protected** WsResponse **handleInvokeResult**(WsInvokeResult wsInvokeResult) {  // 默认情况下，简单返回调用结果  WsResponse response = **new** WsResponse();  response.setRespResult(wsInvokeResult.getInvokeResult());  response.setRespInfo(wsInvokeResult.getInvokeInfo());  response.setRespClob(wsInvokeResult.getInvokeClob());  response.setReturnObject(wsInvokeResult.getReturnObject());  **return** response;  }    @SuppressWarnings("unchecked")  **protected** JAXBElement<**T**> **buildParam**(**T** param) {  // 兼容重复的情况  **if** (param **instanceof** JAXBElement<?>) {  **return** (JAXBElement<**T**>) param;  }  JAXBElement<**T**> element = **new** JAXBElement<**T**>(  getOperationQName(),  (Class<**T**>) param.getClass(),  param);  **return** element;  }    **public** WsResponse **send**(SendReasonVO reason, **T** param) {  JAXBElement<**T**> element = buildParam(param);  **return** sendIndependenceTrasaction(reason, element);  }    **public** WsResponse **send**(SendReasonVO reason, JAXBElement<?> param) {  **return** sendIndependenceTrasaction(reason, param);  }  **public** **void** **sendAsyn**(**final** SendReasonVO reason, **T** param) {  **final** JAXBElement<**T**> element = buildParam(param);  // 此处简单启动异步线程处理，后期需要统一异步方式  Runnable runnable = **new** Runnable() {  @Override  **public** **void** **run**() {  sendIndependenceTrasaction(reason, element);  }  };  Thread thread = **new** Thread(runnable);  thread.start();  }    **private** WsResponse **sendIndependenceTrasaction**(**final** SendReasonVO reason, **final** JAXBElement<?> param) {  **return** execute(reason, param, getWsDestInfo());  }    **private** WsResponse **execute**(SendReasonVO reason, JAXBElement<?> param, WsDestInfo destInfo) {      String invokeResult = **null**;  String invokeInfo = **null**;  String invokeClob = **null**;  Object returnObj = **null**;  String respStatus = "";  **try** {  Object obj = invoker.invoke(param, destInfo);  **if** (obj **instanceof** JAXBElement) {  returnObj = ((JAXBElement<?>) obj).getValue();  } **else** {  returnObj = obj;  }  respStatus = RespStatus.Normal.getCode();  invokeResult = YesOrNo.YES.getCode();  invokeInfo = "调用成功";  // 正常响应报文，此报文不包含SOAP信封信息  invokeClob = destInfo.marshal(obj);  } **catch** (Exception e) {  e.printStackTrace();  invokeResult = YesOrNo.NO.getCode();  respStatus = RespStatus.Exception.getCode();  }      WsInvokeResult wsInvokeResult = **new** WsInvokeResult();  wsInvokeResult.setInvokeResult(invokeResult);  wsInvokeResult.setInvokeInfo(invokeInfo);  wsInvokeResult.setInvokeClob(invokeClob);  wsInvokeResult.setReturnObject(returnObj);      // 处理调用结果  WsResponse response = handleInvokeResult(wsInvokeResult);    **return** response;  }  } |

# 6、创建测试的SetMapParamService，

## 1、下面有个MapConver的转化，比较恶心，为什么，因为这种方法产生的客户端没有适配器Adapter，而输入的map是客户端的com.hlj.client.MapConvertor，所以需要转化

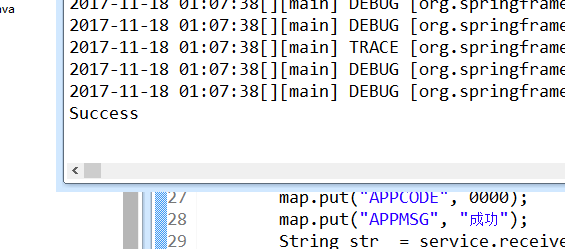
|  |
| --- |
| @Component  **public** **class** SetMapParamService {      @Autowired  **public** SetMapParamSender setMapParamSender;  /\*\*  \*  \* **@author** :HealerJean:  \* **@date** :2017年11月17日 上午9:47:18  \* **@Description**: 调用受托的服务，让受托进行代理人发来数据的业务校验  \* **@param** map  \* **@return**  \* Result  \*/  **public** String **receiveOrBackSender**(Map<String, Object> map){  com.hlj.client.MapConvertor clientConvertor = **new** com.hlj.client.MapConvertor();  com.reflect.mapconver.MapAdapter adapter = **new** com.reflect.mapconver.MapAdapter();  **try** {  com.reflect.mapconver.MapConvertor marshaled = adapter.marshal(map);  List<com.reflect.mapconver.MapConvertor.MapEntry> entries = marshaled.getList();  **for** (com.reflect.mapconver.MapConvertor.MapEntry mapEntry : entries) {  com.hlj.client.MapEntry entry = **new** com.hlj.client.MapEntry ();  **if** (mapEntry.getKey().toUpperCase().equals("INSTITUTIONID")  || mapEntry.getKey().toUpperCase().equals("BUSINESSCODE")  || mapEntry.getKey().toUpperCase().equals("APPSERIONO")  || mapEntry.getKey().toUpperCase().equals("APPCODE")  || mapEntry.getKey().toUpperCase().equals("APPCODE")) {  // entry之间值交换  entry.setKey(mapEntry.getKey());  entry.setValue(mapEntry.getValue());  clientConvertor.getList().add(entry);  }  }  } **catch** (Exception e) {  e.printStackTrace();  // 组装发送参数  }  /\*\*  \* 发送的日志记录 Reason 用来存储报文的主键相关，我这里用不着  \*/  SendReasonVO reason = **new** SendReasonVO();    SetMapParam setMapParam = **new** SetMapParam();  setMapParam.setArg0(clientConvertor);    WsResponse response = setMapParamSender.send(reason, setMapParam);  SetMapParamResponse setMapParamResponse = (SetMapParamResponse) response.getReturnObject();  **return** setMapParamResponse.getReturn();  }  } |

# 7、执行测试

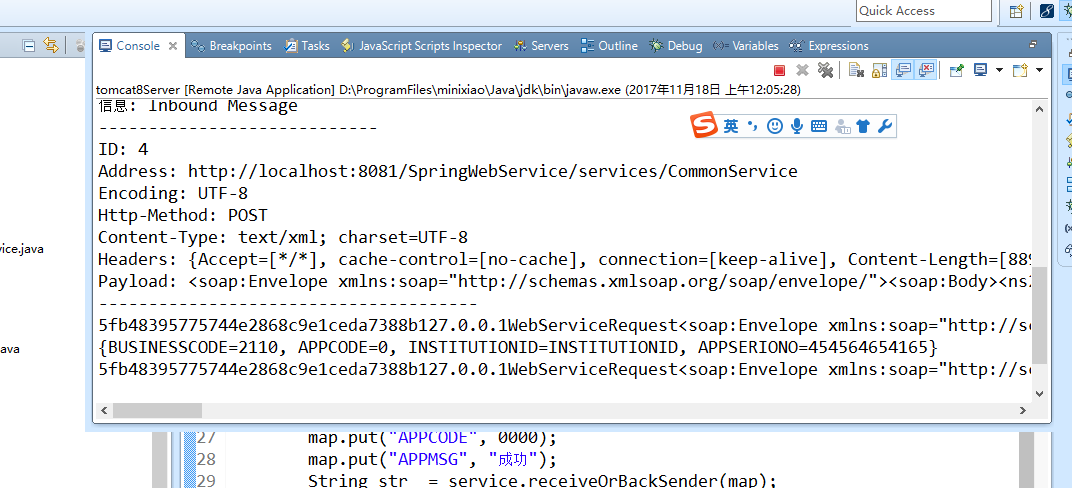
|  |
| --- |
| **public** **class** TestMain {    **public** **static** **void** **main**(String[] args) {  ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext(  **new** String[] { "applicationContext.xml" });    SetMapParamService service=(SetMapParamService)context.getBean(SetMapParamService.**class**);      Map<String, Object> map = **new** HashMap<String, Object>();  map.put("INSTITUTIONID", "INSTITUTIONID");  map.put("BUSINESSCODE", "2110");  map.put("APPSERIONO", "454564654165");  map.put("APPCODE", 0000);  map.put("APPMSG", "成功");  String str = service.receiveOrBackSender(map);  System.out.println(str);  }  } |

# 8、测试成功

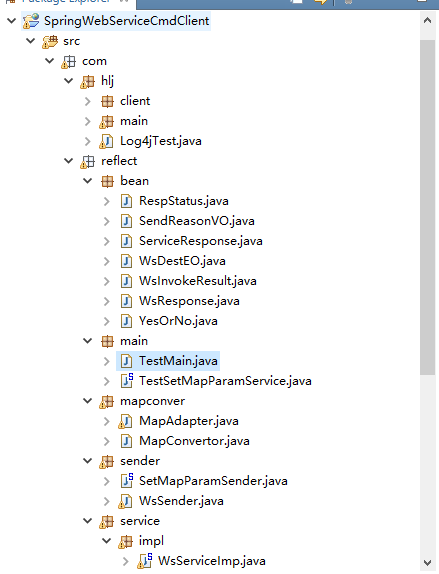
## 1、客户端



## 2、服务端



# 8、项目中间有改动的地方，请观察代码



# 8、WsDestInfo

|  |
| --- |
| import com.reflect.bean.WsDestEO;  public class WsDestInfo {  private WsDestEO wsDestEO;  private JAXBContext context;  private Dispatch<Object> dispatch;    public void init() {  try {  QName serviceQname = new QName(wsDestEO.getNamespace(), wsDestEO.getServiceName());  QName portQname = new QName(wsDestEO.getNamespace(), wsDestEO.getPortName());  context = JAXBContext.newInstance(Class.forName(wsDestEO.getObjFacy()));  URL url = new URL(wsDestEO.getWsdlAddr());  Service create = Service.create(url, serviceQname);  dispatch = create.createDispatch(portQname, context, Mode.PAYLOAD);  } catch (Exception e) {  e.printStackTrace();  }  }    public Marshaller getMarshaller() {  try {  Marshaller marshaller = context.createMarshaller();  marshaller.setProperty(Marshaller.JAXB\_FRAGMENT, true); // 省略xml头信息  marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, false); // 格式化生成的xml串  return marshaller;  } catch (JAXBException e) {  e.printStackTrace();  }  return null;  }    public Unmarshaller getUnmarshaller() {  try {  return context.createUnmarshaller();  } catch (JAXBException e) {  e.printStackTrace();  }  return null;  }    public Dispatch<Object> getDispatch() {  return dispatch;  }    public String marshal(Object obj) {  StringWriter writer = new StringWriter();  try {  getMarshaller().marshal(obj, writer);  return writer.toString();  } catch (JAXBException e) {  throw new RuntimeException(e);  }    }    public JAXBElement<?> unmarshal(Reader reader) {  try {  return (JAXBElement<?>) getUnmarshaller().unmarshal(reader);  } catch (JAXBException e) {  throw new RuntimeException(e);  }  }  public WsDestEO getWsDestEO() {  return wsDestEO;  }  public void setWsDestEO(WsDestEO wsDestEO) {  this.wsDestEO = wsDestEO;  }    } |