# （这里说的是客户端，服务端当然可以map，因为本来就是自己的代码）

# 解释：webservice cxf是不提供map解析的，所以我们这里需要建立一个map的操作类适配器MapAdapter和一个转化实体类MapConvertor

## 1、建立MapConvertor，观察下面有一个MapEntry的内部实体类，cmd客户端的时候会把这两个类分开，这个MapEntry其实相当于map，可以观察下面的key和values

|  |
| --- |
| **public** **class** MapConvertor {  **private** List<MapEntry> list = **new** ArrayList<MapEntry>();  **public** **void** addMapEntry(String fieldName, Object fieldValue) {  MapEntry MapEntry = **new** MapEntry();  MapEntry.setKey(fieldName);  MapEntry.setValue(fieldValue);  list.add(MapEntry);  }  **public** List<MapEntry> getList() {  **return** list;  }  **public** **void** setList(List<MapEntry> list) {  **this**.list = list;  }  **public** **static** **class** MapEntry {  **private** String key;  **private** Object value;  **public** String getKey() {  **return** key;  }  **public** **void** setKey(String key) {  **this**.key = key;  }  **public** Object getValue() {  **return** value;  }  **public** **void** setValue(Object value) {  **this**.value = value;  }  }  } |

## 2、MapAdapter开始转化map和MapConvetor

|  |
| --- |
| **public** **class** MapAdapter **extends** XmlAdapter<MapConvertor, Map<String, Object>> {  /\*\*  \* unmarshal：反序列化，将 Data 对象转为 Map 对象。  \*/  @Override  **public** Map<String, Object> unmarshal(MapConvertor data) **throws** Exception {  Map<String, Object> map = **new** HashMap<String, Object>();  **for** (MapConvertor.MapEntry entry : data.getList()) {  map.put(entry.getKey(), entry.getValue());  }  **return** map;  }    /\*\*  \* marshal：序列化，将 Map 对象转为 Data 对象。  \*/  @Override  **public** MapConvertor marshal(Map<String, Object> map) **throws** Exception {  MapConvertor data = **new** MapConvertor();  **for** (Map.Entry<String, Object> entry : map.entrySet()) {  data.addMapEntry(entry.getKey(), entry.getValue());  }  **return** data;  } |

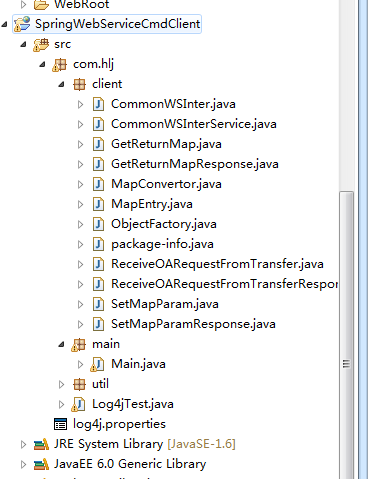
## 3、webservice引入上面的转化操作，这里在map的参数前面放入了注解和MapAdapter

|  |
| --- |
| @WebService(targetNamespace="http://healerjean/")  **public** **interface** CommonWSInter {    **public** String receiveOARequestFromTransfer(@WebParam(name = "name") String name);    //测试传入的参数为map类型  **public** String setMapParam(@XmlJavaTypeAdapter(MapAdapter.**class**)Map<String ,Object> map) ;    //测试取得webservice结果为Map  **public** @XmlJavaTypeAdapter(MapAdapter.**class**)Map<String ,Object> getReturnMap(String name);    } |

## 4、接口的实现类impl

|  |
| --- |
| **public** **class** CommonWSImpl **implements** CommonWSInter{  @Override  **public** String receiveOARequestFromTransfer(String name) {  System.*out*.println("receiveOARequestFromTransfer 方法名字打印");    **return** "receiveOARequestFromTransfer name = "+name;  }  /\*\*  \* 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";  }  /\*\*  \* 测试取得webservice结果为Map  \*/  @Override  **public** Map<String, Object> getReturnMap(String name) {  System.*out*.println("正在被调用getMap方法");  Map<String, Object> map = **new** HashMap<String, Object>();  map.put("name",name);  map.put("age", 23);  **return** map;  } |

## 4、本地形成客户端

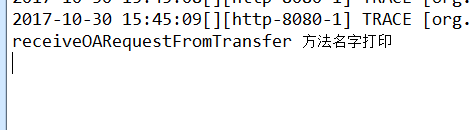


## 5、开始调用

### 1、先测试是否能够接通，所以先测试一个string的

|  |
| --- |
| /\*\*  \* 1、String  \*  \*/  **private** **static** **void** setAndgetStringWs() {  CommonWSInter commonWSInter =**new** CommonWSInterService().getCommonWSInterPort();    /\*\*  \* 调用传入string 参数  \*/  String string = commonWSInter.receiveOARequestFromTransfer("HealerJean");  System.*out*.println(string);    } |

### 2、测试成功

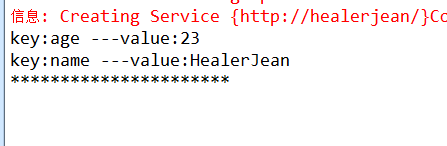


## 6、测试结果为map类型

### 1、结果是MapConvertor类型的，所以打印的时候，需要将数据取出来，利用mapConvertor.getList()

|  |
| --- |
| /\*\*  \* 2、  \* 取得map返回结果  \*/  **private** **static** **void** getMapWs() {  CommonWSInter commonWSInter =**new** CommonWSInterService().getCommonWSInterPort();  MapConvertor mapConvertor = commonWSInter.getReturnMap("HealerJean") ;  **for**(MapEntry mapEntry :mapConvertor.getList()){  System.*out*.println(  "key:"+mapEntry.getKey()+  " ---value:"+mapEntry.getValue());  }  } |

### 2、测试成功，客户端控制台



## 7、测试传入参数为map类型，这里使用到了反射，因为我们在客户端形成的代码中MapConvertor并没有可以构造里面的参数List<MapEntry>，所以只能构造给list

### 1、客户端MapConvertor, 并没有可以构造里面的参数List<MapEntry>，所以只能构造给list

|  |
| --- |
| @XmlAccessorType(XmlAccessType.*FIELD*)  @XmlType(name = "mapConvertor", propOrder = {  "list"  })  **public** **class** MapConvertor {  @XmlElement(nillable = **true**)  **protected** List<MapEntry> list;  \*/  **public** List<MapEntry> getList() {  **if** (list == **null**) {  list = **new** ArrayList<MapEntry>();  }  **return** **this**.list;  }  } |

### 2、反射实现参数的传递MapConvertor,其实最重要的就是里面的mapEntries，因为它就是相当于map的集合

|  |
| --- |
| /\*\*  \* 3、  \* map作为参数参入  \* 利用反射实现  \*/  **private** **static** **void** setMapParamWs() **throws** InstantiationException,  IllegalAccessException, SecurityException, NoSuchFieldException {  CommonWSInter commonWSInter =**new** CommonWSInterService().getCommonWSInterPort();  List<MapEntry> mapEntries = **new** ArrayList<MapEntry>();  MapEntry mapEntry = **new** MapEntry();  mapEntry.setKey("nameField");  mapEntry.setValue("HealerJean");    mapEntries.add(mapEntry);  Class mapConvertorClass = MapConvertor.**class**;  MapConvertor mapConvertorFinal =(MapConvertor) mapConvertorClass.newInstance();    Field list = mapConvertorClass.getDeclaredField("list");  list.setAccessible(**true**);  list.set(mapConvertorFinal, mapEntries);      // 不需要反射，这样就可以发送成功。  //mapConvertorFinal.getList().add(mapEntry);    String reString = commonWSInter.setMapParam(mapConvertorFinal);    System.*out*.println("客户端调用 setMapParam "+reString);  } |

### 3、测试成功，服务端控制台

