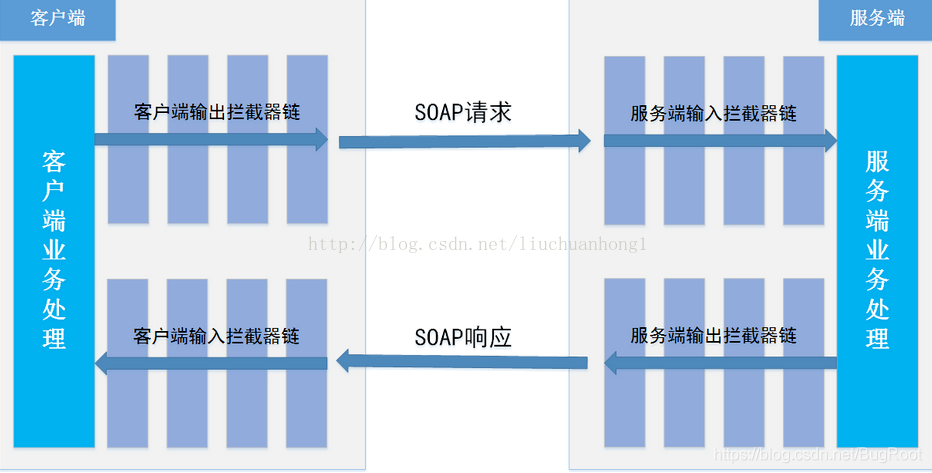
CXF的demo

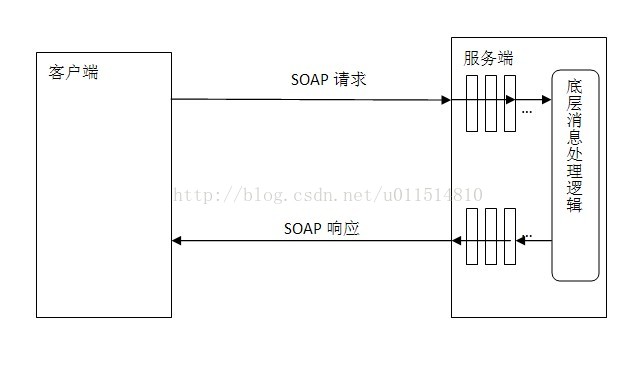
# webservice介绍

Web Service是一个平台独立的，低耦合的，自包含的、基于可编程的web的应用程序，可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的交互操作的应用程序。 [1]

Web Service技术， 能使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件， 就可相互交换数据或集成。依据Web Service规范实施的应用之间， 无论它们所使用的语言、 平台或内部协议是什么， 都可以相互交换数据。Web Service是自描述、 自包含的可用网络模块， 可以执行具体的业务功能。Web Service也很容易部署， 因为它们基于一些常规的产业标准以及已有的一些技术，诸如标准通用标记语言下的子集XML、HTTP。Web Service减少了应用接口的花费。Web Service为整个企业甚至多个组织之间的业务流程的集成提供了一个通用机制。– 来自百度百科

# webservice数据流图





参考：<https://blog.csdn.net/BugRoot/article/details/116782909>

# CXF拦截器

参考链接：<https://blog.csdn.net/weixin_34210740/article/details/85931841>

CXF拦截器是功能的主要实现单元，也是主要的扩展点，可以在不对核心模块进行修改的情况下，动态添加功能。 当服务被调用时，会经过多个拦截器链（Interceptor Chain）处理，拦截器链在服务输入（IN）或输出（OUT）阶段实现附加功能，拦截器可以在客户端加入，也可以在服务端加入。

## 拦截器链的阶段

拦截器链有多个阶段，每个阶段都有多个拦截器。拦截器在拦截器链的哪个阶段起作用，可以在拦截器的构造函数中声明。

输入拦截器链有如下几个阶段，这些阶段按照在拦截器链中的先后顺序排列。

| **阶段名称** | **阶段功能描写叙述** |
| --- | --- |
| RECEIVE | Transport level processing(接收阶段，传输层处理) |
| (PRE/USER/POST)\_STREAM | Stream level processing/transformations(流处理/转换阶段) |
| READ | This is where header reading typically occurs(SOAPHeader读取) |
| (PRE/USER/POST)\_PROTOCOL | Protocol processing, such as JAX-WS SOAP handlers(协议处理阶段，比如JAX-WS的Handler处理) |
| UNMARSHAL | Unmarshalling of the request(SOAP请求解码阶段) |
| (PRE/USER/POST)\_LOGICAL | Processing of the umarshalled request(SOAP请求解码处理阶段) |
| PRE\_INVOKE | Pre invocation actions(调用业务处理之前进入该阶段) |
| INVOKE | Invocation of the service(调用业务阶段) |
| POST\_INVOKE | Invocation of the outgoing chain if there is one(提交业务处理结果，并触发输入连接器) |

输出拦截器链有如下几个阶段，这些阶段按照在拦截器链中的先后顺序排列。

| **阶段名称** | **阶段功能描写叙述** |
| --- | --- |
| SETUP | Any set up for the following phases(设置阶段) |
| (PRE/USER/POST)\_LOGICAL | Processing of objects about to marshalled |
| PREPARE\_SEND | Opening of the connection(消息发送准备阶段。在该阶段创建Connection) |
| PRE\_STREAM | 流准备阶段 |
| PRE\_PROTOCOL | Misc protocol actions(协议准备阶段) |
| WRITE | Writing of the protocol message, such as the SOAP Envelope.(写消息阶段) |
| MARSHAL | Marshalling of the objects |
| (USER/POST)\_PROTOCOL | Processing of the protocol message |
| (USER/POST)\_STREAM | Processing of the byte level message(字节处理阶段，在该阶段把消息转为字节) |
| SEND | 消息发送 |

在输出拦截器链的SEND阶段后，还会触发以\_ENDING结尾阶段，这些ENDING阶段与以上阶段相应，主要用于清理或者关闭资源。ENDING阶段触发的顺序例如以下：

1. SEND\_ENDING
2. POST\_STREAM\_ENDING
3. USER\_STREAM\_ENDING
4. POST\_PROTOCOL\_ENDING
5. USER\_PROTOCOL\_ENDING
6. MARSHAL\_ENDING
7. WRITE\_ENDING
8. PRE\_PROTOCOL\_ENDING
9. PRE\_STREAM\_ENDING
10. PREPARE\_SEND\_ENDING
11. POST\_LOGICAL\_ENDING
12. USER\_LOGICAL\_ENDING
13. PRE\_LOGICAL\_ENDING
14. SETUP\_ENDING

在CXF中，所有对消息的处理都是通过各种拦截器实现。CXF已经实现了多种拦截器，如操纵消息头、执行认证检查、验证消息数据、日志记录、消息压缩等，有些拦截器在发布服务、访问服务时已经默认添加到拦截器链。

CXF默认输入拦截器链，假设没有加入额外的拦截器。CXF输入会顺序经过下面拦截器：

| **拦截器名称** | **拦截器功能** |
| --- | --- |
| AttachmentInInterceptor | Parse the mime headers for mime boundaries, finds the “root” part and resets the input stream to it, and stores the other parts in a collection of Attachments |
| StaxInInterceptor | Creates an XMLStreamReader from the transport InputStream on the Message |
| ReadHeadersInterceptor | Parses the SOAP headers and stores them on the Message |
| SoapActionInInterceptor | Parses “soapaction” header and looks up the operation if a unique operation can be found for that action. |
| MustUnderstandInterceptor | Checks the MustUnderstand headers, its applicability and process it, if required |
| SOAPHandlerInterceptor | SOAP Handler as per JAX-WS |
| LogicalHandlerInInterceptor | Logical Handler as per JAX-WS |
| CheckFaultInterceptor | Checks for fault, if present aborts interceptor chain and invokes fault handler chain |
| URIMappingInterceptor | Can handle HTTP GET, extracts operation info and sets the same in the Message |
| DocLiteralnInterceptor | Examines the first element in the SOAP body to determine the appropriate Operation (if soapAction did not find one) and calls the Databinding to read in the data. |
| SoapHeaderInterceptor | Perform databinding of the SOAP headers for headers that are mapped to parameters |
| WrapperClassInInterceptor | For wrapped doc/lit, the DocLiteralInInterceptor probably read in a single JAXB bean. This interceptor pulls the individual parts out of that bean to construct the Object[] needed to invoke the service. |
| SwAInInterceptor | For Soap w/ Attachments, finds the appropriate attachments and assigns them to the correct spot in the parameter list. |
| HolderInInterceptor | For OUT and IN/OUT parameters, JAX-WS needs to create Holder objects. This interceptor creates the Holders and puts them in the parameter list. |
| ServiceInvokerInInterceptor | Actually invokes the service. |

CXF默认输出拦截器链，假设没有加入额外的拦截器，CXF输入会顺序经过下面拦截器：

| **拦截器名称** | **拦截器功能** |
| --- | --- |
| HolderOutInterceptor | For OUT and IN/OUT params, pulls the values out of the JAX-WS Holder objects (created in HolderInInterceptor) and adds them to the param list for the out message. |
| SwAOutInterceptor | For OUT parts that are Soap attachments, pulls them from the list and holds them for later. |
| WrapperClassOutInterceptor | For doc/lit wrapped, takes the remaining parts and creates a wrapper JAXB bean to represent the whole message. |
| SoapHeaderOutFilterInterceptor | Removes inbound marked headers |
| SoapActionOutInterceptor | Sets the SOAP Action |
| MessageSenderInterceptor | Calls back to the Destination object to have it setup the output streams, headers, etc… to prepare the outgoing transport. |
| SoapPreProtocolOutInterceptor | This interceptor is responsible for setting up the SOAP version and header, so that this is available to any pre-protocol interceptors that require these to be available. |
| AttachmentOutInterceptor | If this service uses attachments (either SwA or if MTOM is enabled), it sets up the Attachment marshallers and the mime stuff that is needed. |
| StaxOutInterceptor | Creates an XMLStreamWriter from the OutputStream on the Message. |
| SoapHandlerInterceptor | JAX-WS SOAPHandler |
| SoapOutInterceptor | Writes start element for soap:envelope and complete elements for other header blocks in the message. Adds start element for soap:body too. |
| LogicalHandlerOutInterceptor | JAX-WS Logical handler stuff |
| WrapperOutInterceptor | If wrapped doc/lit and not using a wrapper bean or if RPC lit, outputs the wrapper element to the stream. |
| BareOutInterceptor | Uses the databinding to write the params out. |
| SoapOutInterceptor$SoapOutEndingInterceptor | Closes the soap:body and soap:envelope |
| StaxOutInterceptor$StaxOutEndingInterceptor | Flushes the stax stream. |
| MessageSenderInt$MessageSenderEnding | Closes the exchange, lets the transport know everything is done and should be flushed to the client. |

## 拦截器分类

1）按位置分：服务器拦截器、客户端拦截器

2）按消息分：入拦截器、处拦截器

3）按定义者分：系统拦截器、自定义拦截器

## 拦截器API

Interceptor（拦截器接口）

AbstractPhaseInterceptor(自定义拦截器从此继承)

LoggingInInterceptor(系统日志入拦截器类)

LoggingOutInterceptor(系统日志出拦截器类)

## 日志拦截器

CXF已经内置了一些拦截器，这些拦截器大部分默认添加到拦截器链中，有些需要手动添加，如CXF提供的日志拦截器：输入日志拦截器LoggingInInterceptor和输出日志拦截器LoggingOutInterceptor，可以用在服务端也可以用在客户端，用来在测试或调试的时候输出服务端、客户端请求和接收到的信息。

## 自定义拦截器样例

### 用户名密码拦截器

<https://blog.csdn.net/weixin_43468680/article/details/102660545>

## 原理介绍（重要）

cxf架构（官方文档）

链接：<https://cxf.apache.org/docs/cxf-architecture.html>

# CXF JAX-WS开发

## JaxWsServerFactoryBean实现基本的webservice

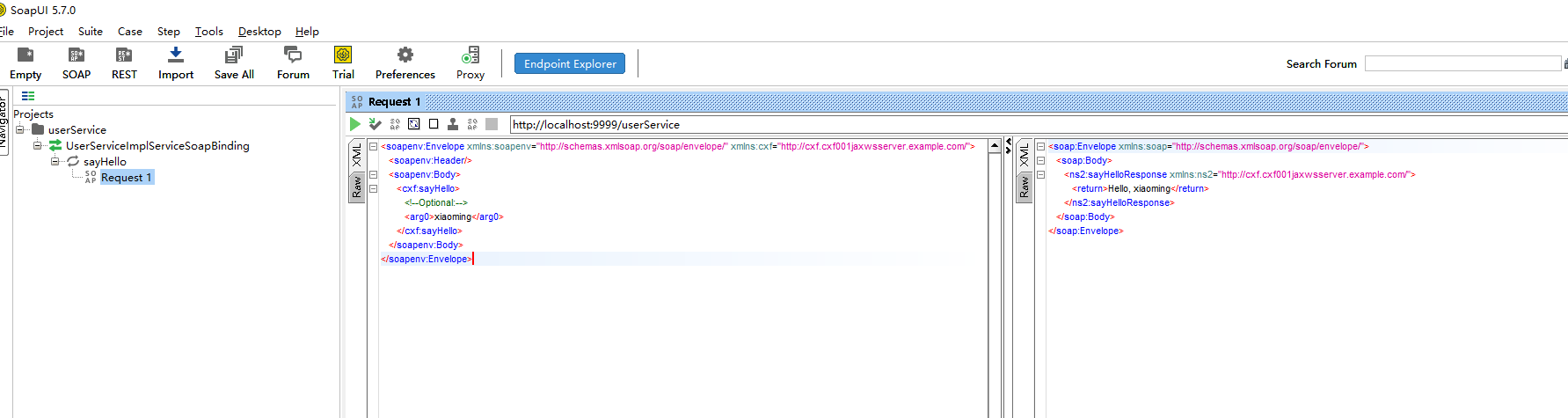
### 服务端实现

*/\*\*  
 \* https://cxf.apache.org/javadoc/latest/org/apache/cxf/jaxws/JaxWsServerFactoryBean.html  
 \*/*static void startWsService(){  
 *//发布的地址和服务名* JaxWsServerFactoryBean jaxWsServerFactoryBean = new JaxWsServerFactoryBean();  
 jaxWsServerFactoryBean.setAddress("http://localhost:9999/userService");  
 jaxWsServerFactoryBean.setServiceClass(UserServiceImpl.class);  
  
  
 Server server = jaxWsServerFactoryBean.create();  
 server.start();  
 *log*.info(">>>>>>>>开始服务>>>>>>>>");  
}

### 客户端实现

@Test  
public void test(){  
 JaxWsProxyFactoryBean jaxWsProxyFactoryBean = new JaxWsProxyFactoryBean();  
 jaxWsProxyFactoryBean.setServiceClass(IUserService.class);  
 jaxWsProxyFactoryBean.setAddress("http://localhost:9999/userService?wsdl");  
  
 IUserService userService = (IUserService) jaxWsProxyFactoryBean.create();  
 String result = userService.sayHello("小明");  
 System.*out*.println(result);  
}

### SOAPUI工具



## JaxWsServerFactoryBean实现webservice并添加自定义拦截器

<https://www.likecs.com/show-323928.html>

### 服务端实现

static void startWsService(){  
 *//发布的地址和服务名* JaxWsServerFactoryBean jaxWsServerFactoryBean = new JaxWsServerFactoryBean();  
 jaxWsServerFactoryBean.setAddress("http://localhost:9999/userService");  
 jaxWsServerFactoryBean.setServiceClass(UserServiceImpl.class);  
  
 jaxWsServerFactoryBean.getInInterceptors().add(new LoggingInInterceptor());  
 jaxWsServerFactoryBean.getOutFaultInterceptors().add(new LoggingOutInterceptor());  
  
 jaxWsServerFactoryBean.getInInterceptors().add(new MessageInterceptor(Phase.*RECEIVE*));  
 jaxWsServerFactoryBean.getOutInterceptors().add(new MessageInterceptor(Phase.*SEND*));  
  
 Server server = jaxWsServerFactoryBean.create();  
 server.start();  
 *log*.info(">>>>>>>>开始服务>>>>>>>>");  
}

### 自定义消息拦截器

import lombok.extern.slf4j.Slf4j;  
import org.apache.cxf.interceptor.Fault;  
import org.apache.cxf.message.Message;  
import org.apache.cxf.phase.AbstractPhaseInterceptor;  
  
@Slf4j  
public class MessageInterceptor extends AbstractPhaseInterceptor<Message> {  
  
 public MessageInterceptor(String phase) {  
 super(phase);  
 }  
  
 @Override  
 public void handleMessage(Message message) throws Fault {  
 *log*.info("############handleMessage##########");  
  
 if (message.getDestination() != null) {  
 *log*.info(message.getId() + "#" + message.getDestination().getMessageObserver());  
 }  
 if (message.getExchange() != null) {  
 *log*.info(message.getExchange().getInMessage() + "#" + message.getExchange().getInFaultMessage());  
 *log*.info(message.getExchange().getOutMessage() + "#" + message.getExchange().getOutFaultMessage());  
 }  
 }  
}

### 客户端实现

@Test  
public void test(){  
 JaxWsProxyFactoryBean jaxWsProxyFactoryBean = new JaxWsProxyFactoryBean();  
 jaxWsProxyFactoryBean.setServiceClass(IUserService.class);  
 jaxWsProxyFactoryBean.setAddress("http://localhost:9999/userService?wsdl");  
  
 jaxWsProxyFactoryBean.getInInterceptors().add(new LoggingInInterceptor());  
 jaxWsProxyFactoryBean.getOutFaultInterceptors().add(new LoggingOutInterceptor());  
  
 jaxWsProxyFactoryBean.getInInterceptors().add(new MessageInterceptor(Phase.*RECEIVE*));  
 jaxWsProxyFactoryBean.getOutInterceptors().add(new MessageInterceptor(Phase.*SEND*));  
  
 IUserService userService = (IUserService) jaxWsProxyFactoryBean.create();  
 String result = userService.sayHello("小明");  
 System.*out*.println(result);  
}

## JAXRSServerFactoryBean实现

未经过验证

### 服务端实现

JAXRSServerFactoryBean jrf = new JAXRSServerFactoryBean();

jrf.setResourceClasses(RestServiceImpl.class);

jrf.setResourceProvider(RestServiceImpl.class, new SingletonResourceProvider(new RestServiceImpl()));

jrf.setAddress(url);

jrf.getInInterceptors().add(new LoggingInInterceptor());

jrf.getOutInterceptors().add(new LoggingOutInterceptor());

jrf.create();

### 客户端实现

JAXRSClientFactoryBean factory = new JAXRSClientFactoryBean();

factory.setServiceClass(RestService.class);

factory.setAddress(url);

factory.getInInterceptors().add(new LoggingInInterceptor());

factory.getOutInterceptors().add(new LoggingOutInterceptor());

RestService ser = factory.create(RestService.class);

ser.get();

# CXF JAX-RS开发

## 使用CXF开发JAX-RS类型的WebService

参考链接：<http://t.zoukankan.com/EzraOholiabXue-p-8066157.html>

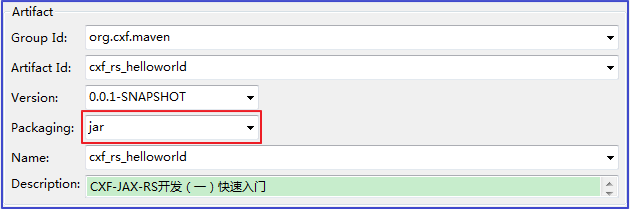
参考链接2：<http://t.zoukankan.com/laoyeye-p-6519846.html> （也不错）

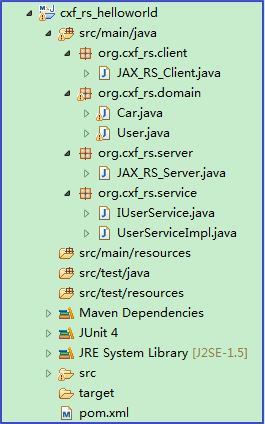
一、简介

　　资源驱动。基于HTTP协议[按照标准指定URL，就可以访问数据]以XML|JSON格式传输数据。

二、quickstart

　　1、创建maven project[Packaging:jar]





　　2、导入依赖

<!-- CXF的rs开发 -->

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-frontend-jaxrs</artifactId>

<version>3.0.1</version>

</dependency>

<!-- 内置jetty的web服务器 -->

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-transports-http-jetty</artifactId>

<version>3.0.1</version>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

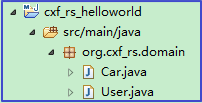
<artifactId>slf4j-log4j12</artifactId>

<version>1.7.12</version>

</dependency>

　　3、搭建服务端

　　　　3.1、导入实体bean



[参考CXF-JAX-WS开发（一）入门案例，2.4.1、导入实体bean目录下的实体类Car.java和User.java](http://www.cnblogs.com/EzraOholiabXue/p/8006117.html)

　　　　3.2、构建服务bean

　　　　　　3.2.1、IUserService.java

package org.cxf\_rs.service;

import java.util.List;

import org.cxf\_rs.domain.User;

public interface IUserService {

//增

public void saveUser(User user);

//删

public void deleteUser(Integer id);

//改

public void updateUser(User user);

//查-查询所有

public List<User> findAllUsers();

//查-根据id查询

public User finUserById(Integer id);

}

　　　　　　3.2.2、UserServiceImpl.java

package org.cxf\_rs.service;

import java.util.ArrayList;

import java.util.List;

import org.cxf\_rs.domain.Car;

import org.cxf\_rs.domain.User;

public class UserServiceImpl implements IUserService {

//增

public void saveUser(User user) {

System.out.println("save user:" + user);

}

//删

public void deleteUser(Integer id) {

System.out.println("delete user id :" + id);

}

//改

public void updateUser(User user) {

System.out.println("update user:" + user);

}

//查-查询所有

public List<User> findAllUsers() {

List<User> users = new ArrayList<User>();

User user1 = new User();

user1.setId(1);

user1.setUsername("小明");

user1.setCity("北京");

List<Car> carList1 = new ArrayList<Car>();

Car car1 = new Car();

car1.setId(101);

car1.setCarName("保时捷");

car1.setPrice(1000000d);

carList1.add(car1);

Car car2 = new Car();

car2.setId(102);

car2.setCarName("宝马");

car2.setPrice(400000d);

carList1.add(car2);

user1.setCars(carList1);

users.add(user1);

User user2 = new User();

user2.setId(2);

user2.setUsername("小丽");

user2.setCity("上海");

users.add(user2);

return users;

}

//查-根据id查询

public User finUserById(Integer id) {

if (id == 1) {

User user1 = new User();

user1.setId(1);

user1.setUsername("小明");

user1.setCity("北京");

return user1;

}

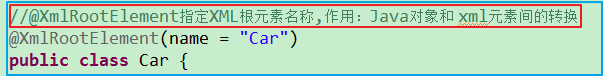
return null;

}

}

　　　　3.3、添加注解

　　　　　　3.3.1、domain[加@XmlRootElement注解]



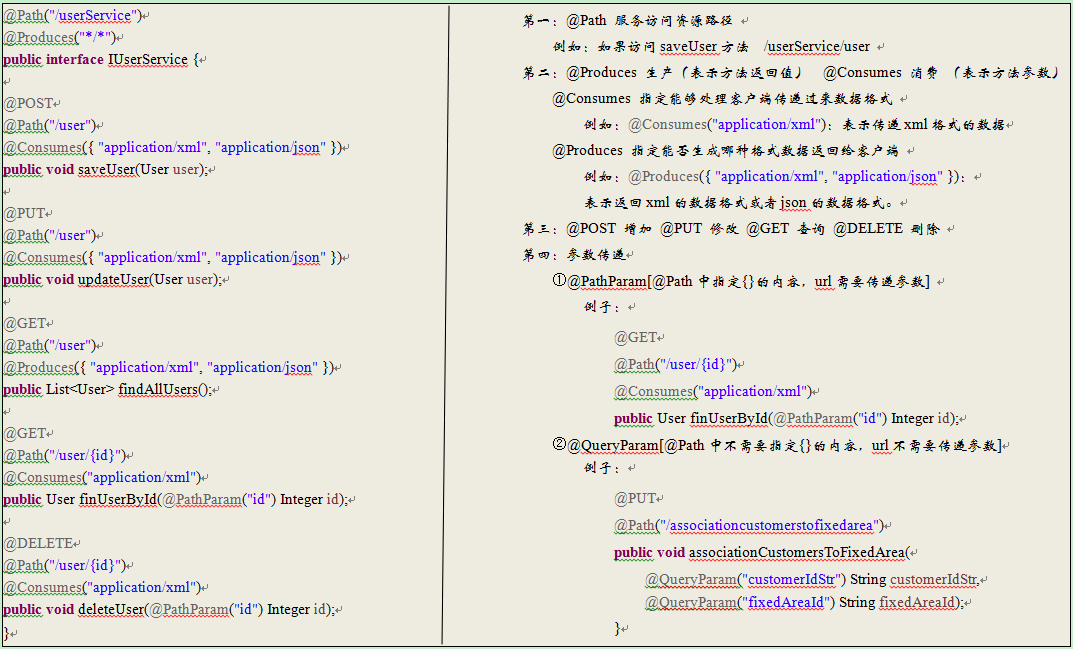
https://images2017.cnblogs.com/blog/1111077/201712/1111077-20171219163100225-732375545.png

　　　　　　未加的话报如下错误：



　　　　　　3.3.2、service

　　　　　　　　①常用注解



　　　　　　　　②IUserService.java带注解

package org.cxf\_rs.service;

import java.util.List;

import javax.ws.rs.Consumes;

import javax.ws.rs.DELETE;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.PUT;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import org.cxf\_rs.domain.User;

@Path("/userService")

@Produces("\*/\*")

public interface IUserService {

//增

@POST

@Path("/user")

@Consumes({ "application/xml", "application/json" })

public void saveUser(User user);

//删

@DELETE

@Path("/user/{id}")

@Consumes("application/xml")

public void deleteUser(@PathParam("id") Integer id);

//改

@PUT

@Path("/user")

@Consumes({ "application/xml", "application/json" })

public void updateUser(User user);

//查-查询所有

@GET

@Path("/user")

@Produces({ "application/xml", "application/json" })

public List<User> findAllUsers();

//查-根据id查询

@GET

@Path("/user/{id}")

@Consumes("application/xml")

@Produces({ "application/xml", "application/json" })

public User finUserById(@PathParam("id") Integer id);

}

　　4、发布服务

package org.cxf\_rs.server;

import org.apache.cxf.interceptor.LoggingInInterceptor;

import org.apache.cxf.interceptor.LoggingOutInterceptor;

import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;

import org.cxf\_rs.domain.Car;

import org.cxf\_rs.domain.User;

import org.cxf\_rs.service.IUserService;

import org.cxf\_rs.service.UserServiceImpl;

public class JAX\_RS\_Server {

public static void main(String[] args) {

// 创建业务接口实现类对象

IUserService userService = new UserServiceImpl();

// 服务器FactoryBean创建服务

JAXRSServerFactoryBean restServer = new JAXRSServerFactoryBean();

restServer.setResourceClasses(User.class, Car.class);

restServer.setServiceBean(userService);

restServer.setAddress("http://localhost:9999/");

// 打印日志

restServer.getInInterceptors().add(new LoggingInInterceptor());

restServer.getOutInterceptors().add(new LoggingOutInterceptor());

// 发布服务

restServer.create();

System.out.println("服务器启动成功！");

}

}

　　5、客户端测试

　　　　5.1、浏览器测试

　　　　　　5.1.1、查询所有

　　　　　　　　查询所有：<http://localhost:9999/userService/user>



　　　　　　5.1.2、根据id查询

　　　　　　　　根据id查询：<http://localhost:9999/userService/user/1>



　　　　5.2、WebClient 工具类[CXF 自带的工具包]

　　　　　　5.2.1、导入依赖

<!-- 导入CXF-Client依赖 -->

<dependency>

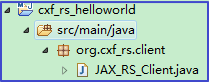
<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-rs-client</artifactId>

<version>3.0.1</version>

</dependency>

　　　　　　5.2.2、目录结构



　　　　　　5.2.3、JAX\_RS\_Client.java

　　　　　　　　① 测试新增

// 测试新增

@Test

public void test\_save(){

/\*\*

\* create ：建立与调用服务资源路径连接

\* path ：访问服务器的路径--->@Path

\* type ：发送给服务器数据格式--->@Consumes

\* accept ：接收服务器传输数据格式--->@Produces

\*/

User user = new User();

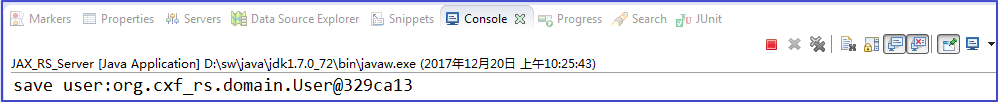
user.setId(3);

user.setUsername("小美");

user.setCity("深圳");

WebClient.create("http://localhost:9999/").path("userService/user").type(MediaType.APPLICATION\_JSON).post(user);

}



　　　　　　　　②测试删除

// 测试删除

@Test

public void test\_delete(){

/\*\*

\* create ：建立与调用服务资源路径连接

\* path ：访问服务器的路径--->@Path

\* type ：发送给服务器数据格式--->@Consumes

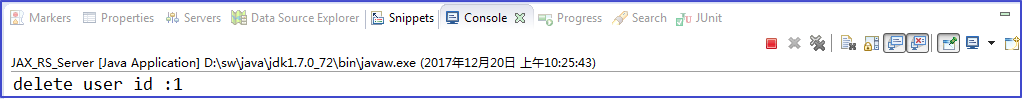
\* accept ：接收服务器传输数据格式--->@Produces

\*/

// 删除

WebClient.create("http://localhost:9999/").path("userService/user/1").type(MediaType.APPLICATION\_XML).delete();

}



　　　　　　　　③测试修改

// 测试修改

@Test

public void test\_update(){

/\*\*

\* create ：建立与调用服务资源路径连接

\* path ：访问服务器的路径--->@Path

\* type ：发送给服务器数据格式--->@Consumes

\* accept ：接收服务器传输数据格式--->@Produces

\*/

User user = new User();

user.setId(3);

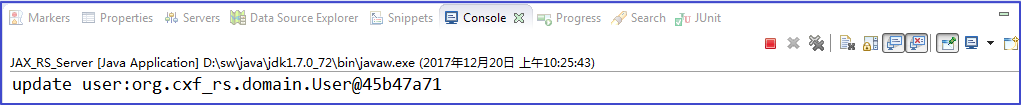
user.setUsername("小美");

user.setCity("深圳");

// 修改

WebClient.create("http://localhost:9999/").path("userService/user").type(MediaType.APPLICATION\_JSON).put(user);

}



　　　　　　　　④测试查询所有

// 测试查询所有

@Test

public void test\_findAll() {

/\*\*

\* create ：建立与调用服务资源路径连接

\* path ：访问服务器的路径--->@Path

\* type ：发送给服务器数据格式--->@Consumes

\* accept ：接收服务器传输数据格式--->@Produces

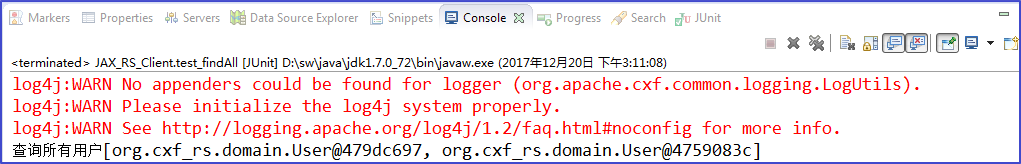
\*/

// 查询所有用户

Collection<? extends User> users = WebClient.create("http://localhost:9999/userService/user").accept(MediaType.APPLICATION\_XML).getCollection(User.class);

System.out.println("查询所有用户" + users);

}



　　　　　　　　⑤测试根据id查询

// 测试根据id查询

@Test

public void test\_findById() {

/\*\*

\* create ：建立与调用服务资源路径连接

\* path ：访问服务器的路径--->@Path

\* type ：发送给服务器数据格式--->@Consumes

\* accept ：接收服务器传输数据格式--->@Produces

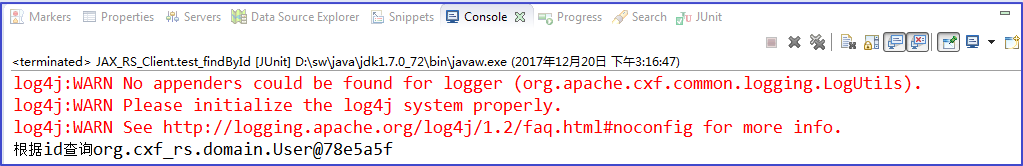
\*/

WebClient client = WebClient.create("http://localhost:9999/");

User responseUser = client.path("userService/user/1").accept(MediaType.APPLICATION\_XML).get(User.class);

System.out.println("根据id查询" + responseUser);

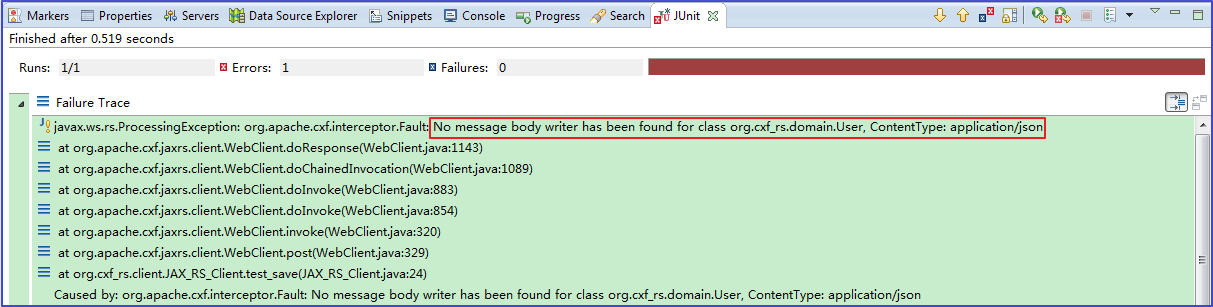
}



　　　　　　5.2.4、测试新增报异常

　　　　　　　　①异常描述

　　　　　　　　　　Caused by: javax.ws.rs.ProcessingException: No message body writer has been found for class cn.itcast.cxf.domain.User, ContentType: application/json



　　　　　　　　②出现原因

　　　　　　　　　　测试时传输JSON格式的数据

　　　　　　　　③解决方案

　　　　　　　　　引入json转换器的依赖

<!-- CXF的json转换器，拓展json -->

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-rs-extension-providers</artifactId>

<version>3.0.1</version>

</dependency>

<!-- 转换json的工具包，被cxf-rt-rs-extension-providers包依赖 -->

<dependency>

<groupId>org.codehaus.jettison</groupId>

<artifactId>jettison</artifactId>

<version>1.3.7</version>

</dependency>

# 杂项

## Jersey (JAX-RS) 教程

<https://rumenz.com/java-topic/jersey-jax-rs-tutorials/index.html>

## CXF对AJAX跨域的支持

AJAX跨域问题解决通常有两种方案，分别是JSONP和CORS，CXF分别对其都有很好的支持。

### JSONP

添加JSONP依赖

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-rs-extension-providers</artifactId>

<version>${cxf.version}</version>

</dependency>

### CORS

添加CORS依赖

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-rs-security-cors</artifactId>

<version>${cxf.version}</version>

</dependency>