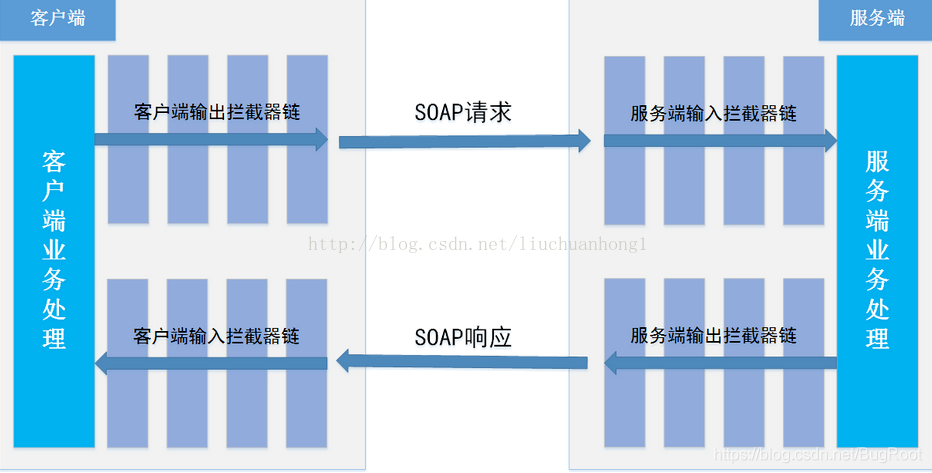
CXF的demo

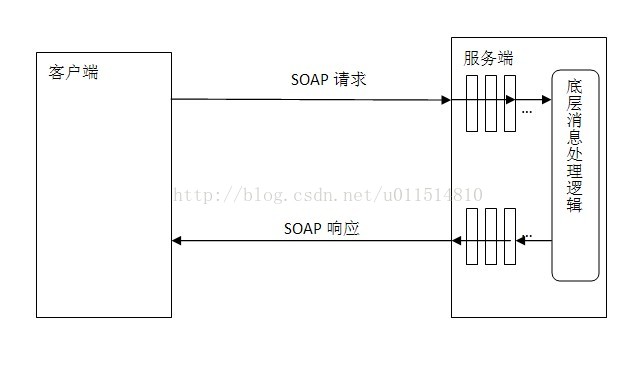
# webservice介绍

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/weibin_6388/article/details/82786841>

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。

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。

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。

## 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>

# JaxWsServerFactoryBean实现基本的webservice

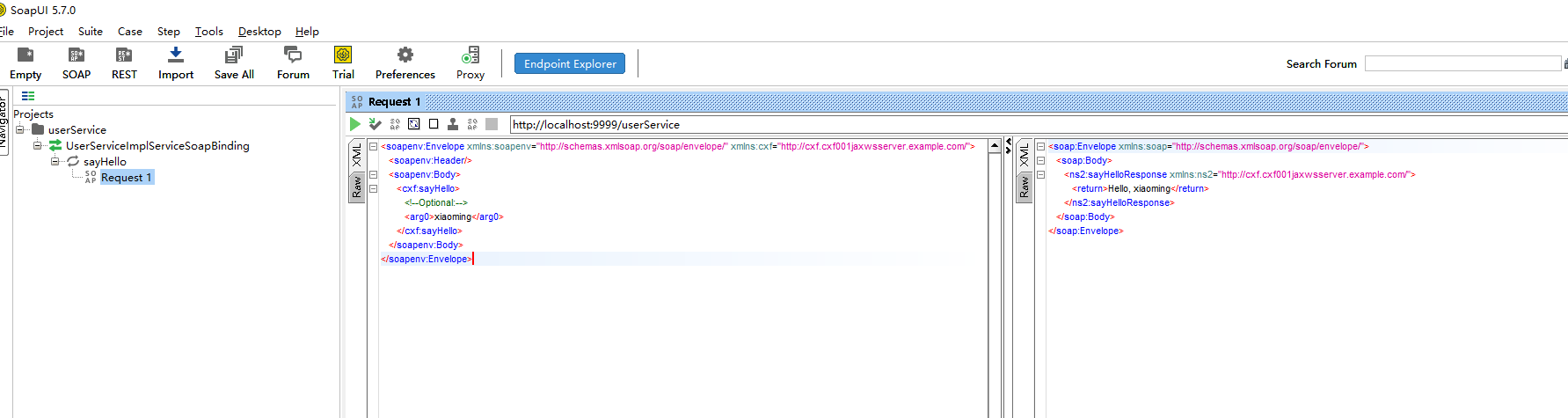
## 服务端实现

*/\*\*  
 \* 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();