# WebService

## Web service简介

不是一种框架、更不是一种技术；

而是一种跨平台、跨语言的规范。

## JavaWeb和WebService的区别

web程序客户端是用浏览器进行请求，发送http协议请求，相对webservice更轻量级。而一般的web项目，或者讲网站，主要是以网页的形式，直接呈现给客户。web 采用 REST 架构，相对于 webservice 来说，极大的减轻开发人员的负担；

web service可以提供wsdl地址给其他人调用，并且可以实现服务共享，对分布式部署很有用，而一般的web项目只是针对项目内部，对外部的支持不是很好。webservice是soap协议，比较重量级；webservice允许不同平台之间的程序以一种标准格式进行数据交换,不是把数据直接呈现给用户，而是需要程序去调用其内含的方法或接口，调用程序接收到返回数据后再根据自己的需要进行处理或呈现。你可以理解成一个远程的API服务。所以一般它都是面向有一些编程经验的客户了。

## Web service应用场景

为了解决不同平台、不同语言所编写的应用之间如何调用。

如：Linux平台下用java写的程序🡪调用win平台下c#写的应用

## 相关框架简介

分别为Axis，**axis2**，Xfire，**CXF**以及JWS(也就是前面所述的JAX-WS，这是Java6发布所提供的对Web Service服务的一种实现。)

前面几项都为开源项目，而其中又以axis2与cxf所最为常用，Axis与XFire已随着技术不断的更替慢慢落幕，而目前也只有axis2和cxf官方有更新，Axis与XFire都已不再更新。

注：

Axis(apache) 🡪 Axis2(apache)

XFile 🡪 XFire + Celtrix = CXF

XFile : 本身就是一个Service框架

Celtrix：ESB框架

**ESB简介**

SOA（面向服务的架构）

要点：Service1 ，Service2 ，Service3—所有组件都是“即插即用”的。

IBM提倡的SOA架构，系统以“组装电脑”的方式来开发软件。

SOA两个要素：

1.各种提供服务的组件。（Web Service）

2.企业服务总线（Enterprise Service Bus ，ESB）

## Web Service实际中用途

**1.同一个公司的新、旧系统的整合。**

如：Linux上的java应用，去调用win平台的C#应用

**2.不同公司的业务整合。**

如：淘宝订单中显示的物流信息；

这个信息是物流公司提供的接口，淘宝去调用这个接口，把物流展示出来。以此把两个公司的系统整合在一起。

解决了不同公司系统平台不同，语言不同的问题。

**3.内容聚合。**

如：一个应用提供股票行情，外汇行情，天气预报，编程语言排行等等

这个内容聚合的应用，需要调用大量不同的平台。

……

## Web service开发流程

### 环境变量的配置

CXF\_HOME D:\ apache-cxf-2.7.18

CLASSPATH中添加路径D:\ apache-cxf-2.7.18\lib  
PATH中添加 D:\ apache-cxf-2.7.18\bin

### Cxf开发web service服务端

\*\*每个webService组件需要两个部分：接口和实现类。\*\*

（1）开发一个web service业务接口，该接口要用@WebServicex修饰。

（2）开发一个Web Service实现类，该接口也要用@WebServicex修饰。

其中要指定两个参数：

endpointInterface ：实现的接口名称

serviceName ：自己命名的service名字

（3）发布WebService

### Cxf开发web service客户端

（1）调用cxf调用的wsdl2java工具，根据wsdl文档生成相应的java代码。

任何语言实现了web service，都需要提供并暴露wsdl文档

$>wsdl2java http://127.0.0.1:9999/webservice?wsdl

（2）找到wsdl2java所生成类的中，一个继承了Service的类。

该类的实例可以当成工厂来使用。

（3）调用该类的getxxxxPort()方法。

## Web service的三个技术

1. wsdl web service define language web-service定义的语言
2. soap simple object access protocal-简单对象访问协议
3. uddi Universal Description, Discovery and Integration-描述、发现以及整合

### Wsdl

WSDL 是基于 XML 的用于描述 Web Services 以及如何访问 Web Services 的语言。

WSDL 指网络服务描述语言 (Web Services Description Language)。

#### 标签介绍

<portType>：端口定义了指向某个 web service 的连接点。

通常是一个函数库，包含多个operation操作。

<message>：函数中的参数。

<types>：使用的数据类型。

<binding>：web service使用的通信协议。一般有http等。

注：<http://www.w3school.com.cn/wsdl/wsdl_binding.asp>

#### Wsdl文档作用

通俗的说，wsdl文档描述了web service的3个方面：

1. what：该webservice包含什么操作

-wsdl:operation name=xxx

1. how：该webservice这个操作该怎么调用
2. where：该webservice的服务地址

总之，只要知道该web service的wsdl文档，接下来就可以调用web service服务。

#### 两个组成部分

Web service接口

1. types（标准的schema）
2. 2n个message
3. PortType –n个operation

Web service实现

1. binding元素 -n个更详细的operation
2. service –指定web service 的服务器地址

### Soap

SOAP 是基于 XML 的简易协议，可使应用程序在 HTTP 之上进行信息交换。

SOAP 是用于访问网络服务的协议。

SOAP 是一种用于发送消息的格式

SOAP=XML + HTTP

几个标签：

<soap:Envelope> 必须的

<soap:Header> 不是必须的，由程序员控制添加

<soap:Body> 必须的。可以有两种情况，一种交互正确时，将由Fault子元素；

一种是请求不存在，由wsdl控制。

### uddi

UDDI 是一种目录服务，通过它，企业可注册并搜索 Web services。

## Web service的一次调用

一次web Service的调用，其实不是方法调用，而是发送soap消息。（即xml文档片段）

xml里的两个属性：

targetNamespace –相当于java中的package

wsdl:import 导入其他包

xmlns—相当于java语言中的import，xmlns导入的是targetNamespace

xml中导包的语法

import:u java.until.\*;

import:s java.sql.\*

u:Date d = new u:Date();

### 对于sayHi操作

传入的消息是

<sayHi>

<arg0>字符串</arg0>

</sayHi>

传出的消息时

<sayHiResponse>

<return>字符串</return>

</sayHiresponse>

### 对于getCatByUser操作

传入的消息是

<getCatByUser>

<arg0>

<address>字符串</address>

<id>int</id>

<name>字符串</name>

<pass>字符串</pass>

</arg0>

</getCatByUser>

传出的消息是

<getCatByUserResponse>

<return>

<color>string</color>

<id>int</id>

<name>string</name>

</return>

</getCatByUserResponse>

## 调用web service的本质

1. 客户端把调用方法参数，转换xml文档片段（soap）。

---必须符合wsdl规范。

1. 通过客户端，把xml文档片段传给服务器。
2. 服务器接收到文档片段。
3. 服务器解析xml文档片段，提取其中的数据，并转换为调用web service所需的参数。
4. 服务器端执行方法。
5. 把执行方法得到的返回值，再次转换为xml文档片段（soap）。

---必须符合wsdl规范。

1. 通过网络把xml文档片段穿给客户端。
2. 客户端接收xml文档片段，并把数据转换为web service的返回值。

本质上看，要一个语言支持web service，唯一的要求是支持xml的解析、生成和网络传输。

## 系统无法处理的类型

在cxf开发中，如果遇到系统无法处理的类型，就需要自行处理。

处理思路是：提供一个转换器，该转换器负责把cxf搞不定的类型，

转换为搞得定的类型。

1. 使用 @XmlJavaTypeAdapter修饰cxf无法处理的。
2. 实现自己的转换器。实现转换器时候，需要开发一个cxf可以搞定的类型。

## web service进行权限控制

1. 服务器端input消息必须携带用户名、密码信息，或者不正确，拒绝调用。--如果不用 cxf框架，soap消息的生成，解析都需要程序员负责。无论添加用户名，密码信息，还是提取用户名，密码，都要由程序员完成。比较麻烦。
2. 如果用cxf框架，soap消息的生成，解析都是由cxf框架完成的。为了让程序员能访问，并修改cxf框架所生成的soap消息，cxf提供了拦截器。

有两种拦截器：in拦截器，out拦截器。

### 服务器端添加拦截器

1. 获取Endpoint的publish方法返回值。
2. 调用该返回值的getInInterceptor,getOutInterceptor方法来获取拦截器列表，

然后添加拦截器。

#### 对于say操作：

**传入消息**

<?xml version="1.0" ?>

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">

<S:Body>

<ns2:sayHi xmlns:ns2="http://webService.liangjt.com/">

<arg0>zhangsan</arg0>

</ns2:sayHi>

</S:Body>

</S:Envelope>

**传出消息**

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<ns2:sayHiResponse xmlns:ns2="http://webService.liangjt.com/">

<return>zhangsan，您好。现在时间是：Wed Jun 07 00:03:50 CST 2017</return>

</ns2:sayHiResponse>

</soap:Body></soap:Envelope>

默认情况下，soap消息中，不会显示Header信息；如果想添加账号密码，需要自己去实现。

#### 对于getAllCat操作

**传入消息：**

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">

<S:Body>

<ns2:getAllCat xmlns:ns2="http://webService.liangjt.com/">

</ns2:getAllCat>

</S:Body>

</S:Envelope>

**传出消息：**

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<ns2:getAllCatResponse xmlns:ns2="http://webService.liangjt.com/">

<return>

<entries><key>第2只</key><value><color>蓝色2</color><id>2</id><name>机器猫2</name></value></entries>

<entries><key>第1只</key><value><color>蓝色1</color><id>1</id><name>机器猫1</name></value></entries>

<entries><key>第4只</key><value><color>蓝色12</color><id>12</id><name>机器猫12</name></value></entries>

<entries><key>第3只</key><value><color>蓝色3</color><id>3</id><name>机器猫3</name></value></entries>

<entries><key>第6只</key><value><color>蓝色32</color><id>32</id><name>机器猫32</name></value></entries>

<entries><key>第5只</key><value><color>蓝色22</color><id>22</id><name>机器猫22</name></value></entries>

</return>

</ns2:getAllCatResponse>

</soap:Body>

</soap:Envelope>

### 客户端添加拦截器

1. 调用ClientProxy的getClient方法，调用该方法以远程Web Service的代理为参数。
2. 调用该返回值的getInInterceptor,getOutInterceptor方法来获取拦截器列表，然后添加拦截器。

### 自定义拦截器

需要实现Interceptor接口。实际上，我们一般会继承AbstractPhaseInterceptor类。

服务器端：拦截的IN消息，检查In消息中是否包含授权信息；

客户端：拦截的是Out消息，在发送服务器端的soap中添加授权信息；

这个是发送的soap片段：

<soap:Header>

<authHeader>

<userId>1111</userId>

<userPass>1111</userPass>

</authHeader>

</soap:Header>

代码部分，放在auth\_ws\_server,auth\_ws\_client中。

## Cxf与Spring的整合（开放接口给别人）

可以在传统的JavaEE应用基础添加一层webService层。

我们的javaEE应用就可以对外暴露webService，这样就可以允许任何平台，任何语言调用javaEE应用。

在传统sshm项目基础上，添加webService应用

### 整合步骤

1. 添加cxf核心jar包
2. 在web.xml中配置cxf的核心控制器：

（CXFServlet：因为版本不同，这个org.apache.cxf.transport.servlet.CXFServlet.class会在两个不同的jar包中。

低版本： ctx-3.16.jar；高版本： cxf-rt-transports-http-3.1.6.jar）

1. 在spring配置文件中导入cxf提供的xchema ，xml配置文件
2. 在spinrg配置文件中使用jaxws:endpoint暴露
3. 如果要添加拦截器，在jaxws:endpoint中添加inInterceptors，outInterceptors子元素。

注：代码放在cxf\_spring,call\_cxf\_spring中。

### 添加拦截器

在jaxws:endpoint中添加inInterceptors，outInterceptors子元素。

## Cxf与Spring的整合（别人提供接口）

采用：spirng+structs+cxf

1. 让Action依赖远程webService接口
2. 复制jar包
3. 在spring配置文件中导入cxf提供的xchema ，xml配置文件
4. 在spring配置文件中使用jaxws:client配置远程webService代理
5. 如果要添加拦截器，在jaxws:client中添加inInterceptors，outInterceptors子元素。

代码放入：spring\_cxf\_client作为客户端，auth\_ws\_server作为服务端

(注：运行会有空指针；原因是spring注入service没有成功；可能是spring struts不对，先不搞这个问题了。目的已经达到了。)

## 在javeEE项目上添加webService服务（不修改之前代码）

步骤：

1. 导入jar包
2. 在spring配置文件中导入cxf提供的xchema ，xml配置文件
3. 定义webService的接口，接口中定义方法（@webService）
4. 定义实现类（@webService）
5. ApplicationContext.xml中配置jaxws:endpoint元素，引入业务层。
6. Filter不拦截webservice的服务路径