**CXF-JAXRS-JAXWS预研**

# 目录

[目录 2](#_Toc364088132)

[版本修订 4](#_Toc364088133)

[1 术语 5](#_Toc364088134)

[2 目的 5](#_Toc364088135)

[3 内容 5](#_Toc364088136)

[4 Web Service 6](#_Toc364088137)

[4.1 定义 6](#_Toc364088138)

[4.2 Web Serivce体系结构 6](#_Toc364088139)

[5 Rest 6](#_Toc364088140)

[5.1 简介 6](#_Toc364088141)

[5.2 原则 7](#_Toc364088142)

[5.3 各编程语言对Rest的支持 7](#_Toc364088143)

[6 比较基于SOAP的服务和REST 8](#_Toc364088144)

[7 Java对Web Service的支持 8](#_Toc364088145)

[8 JAX-WS 9](#_Toc364088146)

[8.1 简介 9](#_Toc364088147)

[8.2 JAX-WS发展历史 9](#_Toc364088148)

[8.3 JAX-WS的两种开发方式 9](#_Toc364088149)

[8.4 使用JAX-WS开发Web Service 10](#_Toc364088150)

[8.5 JAX-WS常用注解 11](#_Toc364088151)

[8.5.1 @WebService 11](#_Toc364088152)

[8.5.2 @WebMethod 11](#_Toc364088153)

[8.5.3 @WebParam 11](#_Toc364088154)

[9 JAX-RS 11](#_Toc364088155)

[9.1 简介 11](#_Toc364088156)

[9.2 Resource类和Resource方法 12](#_Toc364088157)

[9.3 表示一个资源类(资源类建模) 12](#_Toc364088158)

[9.4 JAX-RS常用注解 14](#_Toc364088159)

[9.4.1 @Path 15](#_Toc364088160)

[9.4.2 @POST、@GET、@PUT、@Delete 16](#_Toc364088161)

[说明： 17](#_Toc364088162)

[9.4.3 @Consumes 18](#_Toc364088163)

[9.4.4 @Produces 18](#_Toc364088164)

[9.4.5 @QueryParam 19](#_Toc364088165)

[9.4.6 @PathParam 19](#_Toc364088166)

[9.5 数据返回格式的选择 20](#_Toc364088167)

[9.6 代码示例 20](#_Toc364088168)

[10 JAXM&SAAJ 21](#_Toc364088169)

[11 JAX-RS 与 JAX-WS 的比较 21](#_Toc364088170)

[12 JAXB 22](#_Toc364088171)

[12.1 简介 22](#_Toc364088172)

[12.2 JAXB常用注解 22](#_Toc364088173)

[12.2.1 模板 22](#_Toc364088174)

[13 Apache CXF 22](#_Toc364088175)

[13.1 简介 22](#_Toc364088176)

[13.2 功能特性 23](#_Toc364088177)

[14 集成CXF(Rest支持)需要哪bundle支持 23](#_Toc364088178)

[15 CXF-JAXRS-Blueprint 24](#_Toc364088179)

[15.1 Restful服务的支持 24](#_Toc364088180)

[15.2 JAX-RS : Services Configuration 25](#_Toc364088181)

[16 附录 25](#_Toc364088182)

[16.1 HTTP 25](#_Toc364088183)

[16.1.1 请求头 25](#_Toc364088184)

[16.1.2 状态码 25](#_Toc364088185)

[16.2 WSDL 26](#_Toc364088186)

[16.3 WSDL的主要元素 26](#_Toc364088187)

[16.3.1 portType 27](#_Toc364088188)

[16.3.2 types 27](#_Toc364088189)

[16.3.3 binding 27](#_Toc364088190)

[16.3.4 message 27](#_Toc364088191)

[16.3.5 port 27](#_Toc364088192)

[16.3.6 service 27](#_Toc364088193)

[16.4 web service其它技术 27](#_Toc364088194)

# 版本修订

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 描述 | 修订人 |
| V1.0 | 2012-7-31 | 创建 | 陈绪绍 |
| V1.1 | 2012-9-3 | 完善restful部分 | 陈绪绍 |
| V1.2 | 2012-9-5 | 增加附录部分 | 陈绪绍 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 术语

* HTML：超文本标记语言
* HTTP：超文本转移协议
* JSON：JavaScript 对象符号、一种数据
* MIME：多用途网络邮件扩展
* POJO：普通 Java 对象
* REST：具象状态传输
* URI：统一资源标识符
* URL：统一资源定位符
* XML：可扩展标记语言
* Ajax 全称为“Asynchronous JavaScript and XML”（异步 JavaScript 和 XML）
* REST：具象状态传输
* XML：可扩展标记语言
* CRUD：创建、检索、更新、删除

# 目的

为了增强物流平台与其它系统的交互能力，在物流平台引入Web Service服务组件。

# 内容

1. Web Services、Rest
2. JAX-RS、JAX-WS
3. Apache CXF
4. 调研过程中所涉及的其它技术也有涉及，如SOAP、JAXB、HTTP等

本预研文档，是一个迭代完善的过程。在后续的技术预研和实践当中，逐步添加与完善相关章节的内容。

# Web Services

## 定义

根据W3C的定义，Web服务（Web Services）应当是一个软件系统，用以支持网络间不同机器的互动操作。网络服务通常是许多应用程序接口（API）所组成的，它们通过网络例如国际互联网（Internet）的远程服务器端，执行客户所提交服务的请求。通俗的说WEB服务就是远程计算机上开放的应用程序接口，此API可以是java编写的，也可以是.net编写的；应用可能部署在windows环境的，也可能部署在linux环境，因此WEB服务是跨语言和跨平台的,除了Web Services外常见的远程调用技术还有RMI、CORBA等。常见的WEB服务如：股票信息查询、天气信息查询、淘宝开放API、校内网开发API、中国移动短信发送、电子支付接口等。

Web Services 发展至今已有两种形式：REST 和 SOAP。REST Web Services 基于 HTTP 协议，SOAP Web Services 支持多种传输协议：HTTP、SMTP、MIME 等等。

## Web Services如何工作？

基础的 Web Services 平台是 XML + HTTP。

HTTP 协议是最常用的因特网协议。

XML 提供了一种可用于不同的平台和编程语言之间的语言。

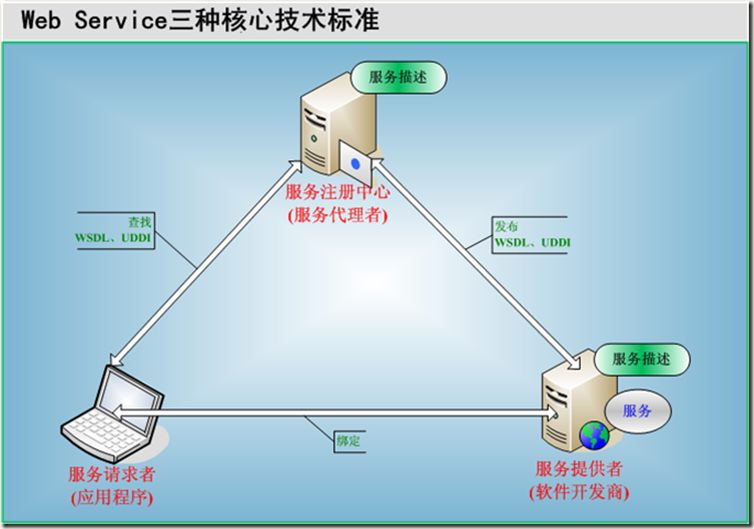
## Web Services平台的元素

SOAP (简易对象访问协议)

UDDI (通用描述、发现及整合)

WSDL (Web services 描述语言)

## Web Serivces体系结构

[](http://images.cnblogs.com/cnblogs_com/QinBaoBei/WindowsLiveWriter/WebServices_C6DA/%E5%9B%BE%E7%89%871_2.png)

# Rest

## 简介

REST的全称为Representational State Transfer，翻译为中文是：表述性状态转移。它不是一种框架，也不是一种规范，而是一种网络应用程序的设计风格和开发方式。用来降低开发的复杂性，提高系统的可伸缩性。

REST了早由Roy Fielding提出。Roy Fielding在他的博士论文《 Architectural Styles and the Design of Network-based Software Architectures 》中正式提出 REST 术语。REST 是一种架构风格（或模式），不是一种具体的标准或架构；REST 不是一个协议，它是对 Web 体系结构设计原则的一种描述，是一种对于服务器的更加有效的抽象方式，是一种基于资源的服务访问架构风格。REST 描述了如何设计和开发分布式 Web 应用系统，它将服务器抽象为一组离散资源的集合，资源又是一个抽象的概念，而不是代表某个具体的东西。从 REST 的概念上来看，所有能够被抽象成资源的东西都可以被指定一个 URL，而开发人员所需要做的工作就是如何能把用户需求抽象为资源，以及如何抽象的精确。因为对资源抽象的越为精确，对 REST 的应用来说就越好。

相较于基于SOAP和WSDL的Web服务，REST 模式提供了更为简洁的实现方案。目前，越来越多的 Web 服务开始采用 REST 风格设计和实现，真实世界中比较著名的 REST 服务包括：Google AJAX 搜索 API、Amazon Simple Storage Service (Amazon S3)等。

## 原则

为了降低开发的复杂度和提高系统的可伸缩性，开发人员在设计应用程序时应该注意坚持 REST 的四个基本设计原则：

1. 可寻址的资源

所有的“事物”对应唯一的资源标识，并且通过标识可以获取该“事物”。

1. 使用 HTTP 的标准方法

开发人员显式地使用 HTTP 的 POST，GET，PUT， DELETE 方法，对系统资源进行创建、读取、更新和删除的操作。

1. 资源多重表述

用户可以根据需求，自由选择 REST API 请求和返回结果的格式，比如是选择 ATOM 格式，XML 格式还是 JSON 格式。这样就需要开发人员在设计 REST API 的时候，提供完备的内容协商能力。

1. 无状态通信

服务端不保存客户端的信息，从而提高系统的容错性和伸缩性。

## 各编程语言对Rest的支持

当前各编程语言对Rest都提供了支持。如：

* Java(JAX-RS、Jersey/RESTEasy/Restlet、Spring MVC 3/Struts 2)
* Ruby(Ruby on Rails)
* PHP(CakePHP)
* Python(Django)
* JavaScript(jQuery/ExtJS/Dojo/Prototype)
* C#(ASP.NET MVC)

# 比较基于SOAP的服务和REST

现在开放式 Web 平台的 API 主要是通过 REST 和基于 SOAP 的服务来实现。

传统的 Web 服务通过简单对象访问协议 (SOAP) 进行消息的交换。它是一种用于一单向通信的消息格式，将消息组合成 XML 文档。描述了消息的传输，主要是通过 HTTP 协议。它定义一组 RPC 调用与 SOAP 消息互相转换的契约，将 RPC 调用封装为 SOAP 消息进行传输，并在服务器端反向转换为服务器端 RPC 调用，最终结果再以类似机制返回给客户端。

基于 SOAP 的服务是重量级的服务，因为它有严格的约束和标准，开发人员需要深入了解基于 SOAP 的 web 服务中用到的关键技术：XML，WSDL、SOAP 及 UDDI，这样就缺少了开发的灵活性。

相比基于 XML 技术的其它臃肿的 web 服务而言，REST 显得更加简洁，更轻量级。增删查改是应用软件里面最常见的操作，而在 HTTP 方法中，正好有其对应的方法实现，所以可以有效的降低复杂度。同时也能够满足异构平台之间的交互。

当前，基于SOAP的服务和REST二者都有一定的应用场景，互联网上的很多应用，同时提供基于SOAP的服务和REST。

# Java对Web Services的支持

JAVA中共有三种Web Service规范，分别JAX-WS(最早是JAX-RPC)、JAX-RS、JAXM&SAAJ。

其中有两种SOAP Web Service规范：JAX-WS和SAAJ。JAX-RS是Rest服务定义。

使用较多的是JAX-WS和JAX-RS。

# JAX-WS

JAX-WS的细节内容可参考Java官方页面：

http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html

## 简介

JAX-WS(Java API for XML Web Services)是一组专门用于实现XML Web Services 的Java API。JAX-WS通常可简写为JWS， JDK 1.6自带JAX-WS 版本为2.1。不过，JAX-WS 只提供Web Services 的基础功能，所以如果你希望实现 Web Services 的复杂功能，比如 WS-Security，WS-Policy，WS-RM、服务拦截等，那就需要切换到Apache CXF 、Metro 、Axis2等Java Web Services构建工具。

JAX-WS的定义和实现主要位于javax.jws.\*包中。

更多JAX-WS内容可参考Oracle官方网站如下页面：

<http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html>

## JAX-WS发展历史

Web Servic出现以来。首先是 SOAP，但 SOAP 仅描述消息的情况，然后是 WSDL，WSDL 并不会告诉您如何使用 Java™ 编写 Web 服务。在这种情况下，JAX-RPC 1.0 应运而生。经过数月使用之后，编写此规范的 Java Community Process (JCP) 人员认识到需要对其进行一些调整，调整的结果就是 JAX-RPC 1.1。该规范使用大约一年之后，JCP 人员希望构建一个更好的版本：JAX-RPC 2.0。其主要目标是与行业方向保持一致，但行业中不仅只使用 RPC Web 服务，还使用面向消息的 Web 服务。因此从名称中去掉了“RPC”，取而代之的是“WS”（当然表示的是 Web 服务）。因此 JAX-RPC 1.1 的后续版本是 JAX-WS 2.0——Java API for XML-based Web services。

## JAX-WS的两种开发方式

* 契约优先(WSDL First)：从WSDL契约着手，生成Java类来实现服务。
* 代码优先(JAVA First)：从Java类着手，使用Annotation来生成 WSDL 文件和Java 接口。

契约优先WSDL 方法需要对用于定义消息格式的 WSDL 和 XML 模式定义（XSD）有良好的理解。如果您Web服务相当陌生，建议从代码优先方法着手。

## 使用JAX-WS开发Web Services

JDK1.6新增支持webService。即JAX-WS。

|  |
| --- |
| package net.chenxs;  import javax.jws.WebMethod;  import javax.jws.WebService;  import javax.jws.soap.SOAPBinding;  import javax.xml.ws.Endpoint;  @WebService(serviceName = "MyService", targetNamespace = "http://www.chenxs.net")  @SOAPBinding(style = SOAPBinding.Style.RPC)  public class MyService {  @WebMethod  public String sayHello(String name) {  System.out.println("hello " + name);  return "hello " + name;  }  public static void main(String[] args) {  　　　　//发布Web Service  Endpoint.publish("http://localhost:9000/myservice?wsdl ", new MyService());  }  } |

在浏览器中访问如下URL“http://localhost:9000/myservice?wsdl”，若能看到如下WSDL文件内容。则表示Web Service发布成功。



## JAX-WS常用注解

### @WebService

 @WebService 注解将一个POJO变成完整的Web服务端点。JAX-WS 假设在带 @WebService 注解的类中所有公共方法都应该转换为 Web 服务操作，并以方法名作为操作名。有如下几个属性可以配置。

* endpointInterface：指向一个定义此WebService抽象定义接口的完整类路径
* name：WebService名；默认的port名为"实现类名+Port"，binding名为"实现类名+PortBinding",通过指定name的值来替换实现类名。
* portName：指定port名，可以完成替换默认port名，或由上面的"name"指定的port名。
* targetNamespace：指定targetNamespace值，默认的值为 "http://包名/"，可以通过此变量指定一个自定义的targetNamespace值。（注：如果分别定义和接口和实现，则他们有各自的targetNamespace）
* serviceName：指定service名
* wsdlLocation：指向一个预定义的wsdl的文件，替代自动生成的wsdl文件

如：

|  |
| --- |
| package com.ibm.samples.helloworld;  import javax.jws.WebService;  @WebService(name = "HelloWorld", targetNamespace = "urn:samples.ibm.com/HelloWorld")  public interface HelloWorld {  ...  } |

### @WebMethod

@WebMethod有如下几个属性可以设置：

* action：指定此方法对应的action
* exclude：false --表示此方法包含在web服务中；true表示排除此方法
* operationName：指定方法对应的operation的名字。

如：

|  |
| --- |
| @WebMethod(exclude=true)  public String sayHi(@WebParam(name="name") String name); |

### @WebParam

@WebParam给参数命名，提高可代码可读性。此项可选。

## SEI与SIB

* SEI(Service Endpoint Interface ) 服务端点接口(Web服务的接口)。
* SIB(Service Implementation Bean) 服务实现Bean(Web服务的实现)。

基于SOAP的Java Web服务尽管可以只由单独的Java类来实现，但是遵从最佳实践原则，最好使用“接口+实现”的方式来实现。首先应该定义一个Java接口类，在此接口内声明Web服务需要完成或实现的业务方法。通常这类接口被称为"SEI"，即服务端点接口（Service Endpoint Interface）。而与之对应的实现类被称为"SIB"，即服务实现Bean（Service Implementation Bean）。SIB可以是一个POJO（Pure old java object or plain ordinary java object）或一个无状态会话EJB（Enterprise Java Bean）。

## 发布Web Services

使用JAX-WS的API发布Web Serivces相当简单。如下：

|  |
| --- |
| import javax.xml.ws.Endpoint;  public class Server {  public static void main(String[] args) {  Endpoint.publish("http://localhost:8088/calculator",new HelloWorldImpl());  }  } |

这里publish方法需要两个参数：

* address：服务对外暴露的用于调用服务的地址
* implementor：服务的实现对象

## 测试Web Services是否发布成功

启动这个Server类，就可以访问服务了。要测试服务有没有启动，可以输入ttp://localhost:8088/helloworld?wsdl，如果一切正常，就可以看到一个wsdl定义内容，表示服务已经成功启动。

## 调用Web Services

|  |
| --- |
| **public** **class** Client {  /\*\*  \* **@param** args  \* **@throws** Exception  \*/  **public** **static** **void** main(String[] args) **throws** Exception {  URL url = **new** URL("http://localhost:8181/helloworld?wsdl");  QName qname =**new** QName("http://samples.ws.wms.csi.com/", "HelloWorldImplService");  Service service = Service.*create*(url,qname);    HelloWorld hw = service.getPort(HelloWorld.**class**);  hw.sayHello("chenxushao");  }  } |

## 使用wsimport生成Web Services客户端

wsimport -keep -p com.wsclient http://localhost:8888/ns?wsdl

-keep 选项指示保留生成的文件，-p 选项指定需要在其中生成构件的包名，

http://localhost:8888/ns?wsdl 指定 WSDL 文件的位置。



## 使用Apache TCPMon查看SOAP消息

 由于所有的消息都是通过类的方法被Java在后台封装了，所以一大不利处就是没法查看具体发送的SOAP消息和返回的结果。

 所以需要一个工具用来查看与特定的Server和Port上消息的交互。有一大堆工具可以做这个，推荐Apache TCPMon。

## MTOM与Web Services

我们先来了解默认情况下 SOAP 是如何传输数据的：

在 SOAP 消息中所有的二进制数据都必须以编码之后的形态存在于 XML 文件中（为避免字符冲突）。正常文本 XML 使用 Base64 对二进制数据进行编码，这就要求每三个字节对应四个字符，从而使得数据的大小增加三分之一。如果我们需要传送 10M 的文件，编码之后文件大小就 13M。这种情况下，JAVA 引入了 MTOM（消息传输优化机制）消息编码。MTOM 就是针对 SOAP 消息传输的基础上提出的改进办法。对于大量数据的传递，不会进行 Base64 编码，而是直接以附件的二进制原始数据的形式封装在 SOAP 消息的 MIME 部分，进行传输。使用 MTOM 的目的在于优化对较大的二进制负载的传输。对于较小的二进制负载来说，使用 MTOM 发送 SOAP 消息会产生显著的开销，但是，当这些负载增大到几千个字节时，该开销会变得微不足道。

在服务器端增加upload() 和download() 两个方法，并通过添加标注 @MTOM，在服务器端开启 MTOM 消息传输功能，同时用datahandler类型封装传输文件。并使用 @XmlMimeType("application/octet-stream") 标注 datahandler，以表示这是一个附件类型的二进制数据。

## JAX-WS异步调用

## JAX-WS验证

## 集成CXF(JAX-WS支持)需要哪bundle支持

集成Apache CXF JAX-WS支持涉及了大约30多个bundle，开发时统一提供给大家。

## CXF-JAX-WS-Blueprint

|  |
| --- |
| <!-- START SNIPPET: beans -->  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"  xmlns:cxf="http://cxf.apache.org/blueprint/core"  xsi:schemaLocation="  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd  http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd">  <cxf:bus>  <cxf:features>  <cxf:logging/>  </cxf:features>  </cxf:bus>  <jaxws:endpoint id="helloWorld"  implementor="org.apache.servicemix.examples.cxf.HelloWorldImpl"  address="/HelloWorld">  <jaxws:outInterceptors>  <bean class="org.apache.servicemix.examples.cxf.EnableCORSInterceptor"/>  </jaxws:outInterceptors>  </jaxws:endpoint>  </blueprint>  <!-- END SNIPPET: beans --> |

# JAX-RS

## 简介

Java EE6引入了对JSR-311的支持。JSR-311（JAX-RS：Java API for RESTful Web Services）旨在定义一个统一的规范，使得 Java 程序员可以使用一套固定的接口来开发 REST 应用，避免了依赖于第三方框架。同时，JAX-RS使用POJO编程模型和基于标注的配置，并集成JAXB，从而可以有效缩短 REST 应用的开发周期。

JAX-RS定义的API位于javax.ws.rs包中。

JAX-RS的具体实现由第三方提供，例如Sun的参考实现 Jersey、Apache 的 CXF以及JBoss的RESTEasy等。

更多JAX-RS详细内容可参考Oracle官方网站如下页面：

<http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>

## Resource类和Resource方法

Web 资源作为一个 Resource 类来实现，对资源的请求由 Resource 方法来处理。Resource 类或 Resource 方法被打上了 Path 标注，Path 标注的值是一个相对的 URI 路径，用于对资源进行定位，路径中可以包含任意的正则表达式以匹配资源。和大多数 JAX-RS 标注一样，Path 标注是可继承的，子类或实现类可以继承超类或接口中的 Path 标注。

Resource 类是 POJO，使用 JAX-RS 标注来实现相应的 Web 资源。Resource 类分为根 Resource 类和子 Resource 类，区别在于子 Resource 类没有打在类上的 Path 标注。Resource 类的实例方法打上了 Path 标注，则为 Resource 方法或子Resource 定位器，区别在于子 Resource 定位器上没有任何 @GET、@POST、@PUT、@DELETE 或者自定义的 @HttpMethod。

## 表示一个资源类(资源类建模)

每个资源必须被建模成一个具体结构，比如 XML 和 JSON。该资源的所有属性和值将使用相应 XML 属性名和值映射。以本文为例，让我们考虑将 “Employee” 作为一个资源，每个 Employee 拥有惟一的属性，比如员工 ID，员工名字、员工地址、员工邮箱、以及员工电话号码

终端用户总是以 XML 或 JSON 方式来表示一个资源。当终端用户需要创建或更新资源的一个新实例时，它们传递该资源相应的 XML 或 JSON。资源类中的 Wink 注释通常将 XML 或 JSON 编组到一个 JAVA 对象中。定义一个实体对象 (Employee) 实例的好处是可以在终端编组 XML 或 JSON，您拥有一个已经保存到数据存储中的实体对象。

资源可以表示一个可支持数据检索和操作的可用组件。资源类可以定义帮助实现事务逻辑的资源方法。如下显示如何为 Employee 资源创建一个资源类。更详细内容可参考JAXB章节内容。

|  |
| --- |
| package rest.resource;  import javax.xml.bind.annotation.XmlAccessType;  import javax.xml.bind.annotation.XmlAccessorType;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;  import javax.xml.bind.annotation.XmlTransient;  import openJpa.model.\_Employee;  @XmlAccessorType(XmlAccessType.PROPERTY)  @XmlRootElement(name="employee")  public class Employee {  private \_Employee modelEmployee;  public Employee() {  super();  this.modelEmployee = new \_Employee();  }    public Employee(\_Employee modelEmployee) {  this.modelEmployee = modelEmployee;  }  @XmlTransient  public \_Employee getEmployee() {  return this.modelEmployee;  }  public void setEmployee(\_Employee modelEmployee)  {  this.modelEmployee = modelEmployee;  }    @XmlElement(name = "employeeName")  public String getEmployeeName() {  return modelEmployee.getEmployeeName();  }    public void setEmployeeName(String employeeName) {  modelEmployee.setEmployeeName(employeeName);    }  @XmlElement(name = "address")  public String getAddress() {  return modelEmployee.getAddress();  }  public void setAddress(String address) {  modelEmployee.setAddress(address);  }    @XmlElement(name = "email")  public String getEmail() {  return modelEmployee.getEmail();  }    public void setEmail(String email) {  modelEmployee.setEmail(email);  }    @XmlElement(name = "telephone")  public int getTelephone() {  return modelEmployee.getTelephone();  }    public void setTelephone(int telephone) {  modelEmployee.setTelephone(telephone);  }    @XmlElement(name = "employeeId")  public int getEmployeeId() {  return modelEmployee.getEmployeeId();  }    public void setEmployeeId(int employeeId) {  modelEmployee.setEmployeeId(employeeId);  }    @XmlElement(name = "version")  public int getVersion() {  return modelEmployee.getVersion();  }    public void setVersion(int version) {  modelEmployee.setVersion(version);  }  } |

## JAX-RS常用注解

JAX-RS提供了一些注解将一个资源类，一个POJO Java类，封装为Web资源。

JAX-RS 利用 Java 注解的威力，使用注解来执行诸如以下的操作：

1. 将 HTTP 方法和 URIs 绑定到 Java 类的方法
2. 将来自 URI 或 HTTP 头部的元素作为方法参数注入
3. 在 HTTP 消息体和 Java 类型之间来回转换
4. 将 URI 模式绑定到 Java 类和方法 — @Path 注释
5. 将 HTTP 操作绑定到 Java 方法 — @GET、 @POST、@PUT 和 @DELETE 注释

主要的标注有：

* @Path，标注资源类或方法的相对路径
* @GET，@PUT，@POST，@DELETE，标注方法是用的 HTTP 请求的类型，用于对资源进行检索、更新、创建和删除的操作。
* @Produces，标注返回的 MIME 媒体类型
* @Consumes，标注可接受请求的 MIME 媒体类型
* @PathParam，@QueryParam，@HeaderParam，@CookieParam，@MatrixParam，@FormParam, 分别标注方法的参数来自于 HTTP 请求的不同位置，例如 @PathParam 来自于 URL 的路径，@QueryParam 来自于 URL 的查询参 数，@HeaderParam 来自于 HTTP 请求的头信息，@CookieParam 来自于 HTTP 请求的 Cookie。

### @Path

含义：用来映射URI，为资源类以及资源类中包含的方法提供访问路径。value 值可以包含文本字符、变量或具有定制正则表达式的变量。

作用位置：类或方法

示例：

|  |
| --- |
| **package net.chenxs;**  **import** javax.ws.rs.GET;  **import** javax.ws.rs.Path;  @Path("/helloworld")  **public** **class** HelloWorld {  @GET  @Path("/echo")  **public** String Echo(){  **return** "hello world";  }  } |

**说明：**使用@PATH标识HelloWorld类，HelloWorld类被当作是一个 JAX-RS 资源（或服务），正如其类定义前面的 @Path("/helloworld ") 注释所示。字符串 "/helloworld " 是该资源的相对根 URI。JAX-RS 将匹配相对 URI "/helloworld " 的 HTTP 请求路由到 HelloWorld 类的方法。

### **@POST、@GET、@PUT、@Delete**

JAX-RS 定义了@POST、@GET、@PUT和@DELETE，分别对应4种 HTTP 方法，用于对资源进行创建、检索、更新和删除的操作(CRUD)。

当Web Service获得客户端发出的对与某个网络资源的HTTP(POST、GET、PUT、DELETE等)操作时，服务器会调用被上述注解所修饰方法来处理相应的HTTP请求。当然，被调用的资源类方法首先得满足URI。

**POST**

含义：用于在服务器上创建资源

作用位置：方法

示例：

|  |
| --- |
| @Path("/")  public class BookkeepingService {  @Path("/account/")  @POST  @Consumes("application/json")  public Response createAccount(Account account) {  ......  }  } |

URL示例：<http://www.example.com/account/>

说明：如果使用 POST 方法请求资源”/account”，则 createAccount 方法将被调用，JSON 格式的请求体被自动映射为实体参数account。

**GET**

含义：用于在服务器上检索资源。

作用位置：方法

示例：

|  |
| --- |
| @Path("/")  public class BookkeepingService {  @Path("/person/{id}/accounts/")  @GET  @Produces("application/json")  public Account[] readAccountsByPerson(@PathParam("id")  int id) {  ......  }  } |

说明：

URL示例：<http://www.example.com/persion/123/accounts>

**PUT**

含义：用于更新服务器上的资源。

作用位置：方法

示例：

|  |
| --- |
| @Path("/")  public class BookkeepingService {  @Path("/account/")  @PUT  @Consumes("application/json")  public Response updateAccount(Account account) {  ......  }  } |

说明：如果使用 PUT 方法请求资源”/account”，则 updateAccount 方法将被调用，JSON格式的请求体被自动映射为实体参数account。

URL示例：<http://www.example.com/account>

**DELETE**

含义：用于删除服务器上的资源

作用位置：方法

示例：

|  |
| --- |
| @Path("/")  public class BookkeepingService {  @Path("/account/{id:\\d+}/")  @DELETE  public Response deleteAccount(@PathParam("id")  int id) {  ......  }  } |

说明：如果使用 DELETE 方法请求资源”/account/323”，则 deleteAccount 方法将被调用，方法参数 id 被赋值为 323。

URL示例：http://www.example.com/account/123

### @Consumes

含义：用来表示资源类方法能够处理的 MIME 的媒体类型。

作用位置：类或方法，大多数情况下用在方法中

常用值：

MediaType.APPLICATION\_XML

MediaType.APPLICATION\_JSON

MediaType.APPLICATION\_FORM\_URLENCODED

示例：

|  |
| --- |
| @Path("/person/")  @POST  @Consumes("application/json")  public Response createPerson(Person person) {  ......  } |

说明：

### @Produces

含义：用来表示资源类方法能够返回的MIME的媒体类型。

作用位置：类或方法，大多数情况下用在方法中

常用值：

MediaType.APPLICATION\_XML

MediaType.APPLICATION\_JSON

MediaType.TEXT\_HTML

示例：

|  |
| --- |
| // 获取一个用户信息  　@GET  　 @Path("{userid}/")  　@Produces({"application/json",”application/xml”})  public JSONObject getUser(@PathParam("userid") String userid)  throws JSONException, Exception {  Person personEntity = mf.findPerson(userid);  if (personEntity==null) throw new Exception ("没有找到用户");  return new JSONObject()  .put("username", personEntity.getUserName())  .put("password", personEntity.getPassword())  } |

说明：

URL访问示例：

http://localhost:9000/users/getUser/111?\_type=xml

http://localhost:9000/users/getUser/111?\_type=json

### @QueryParam

@QueryParam 指定来自于URL的查询参数

这个标注是给通过?的方式传参获得参数值的,如:

GET /employees?maxyear=2009&minyear=1999

|  |
| --- |
| @Path(/employees/")  @GET  public Response getEmployees(  @DefaultValue("2002") @QueryParam("minyear") int minyear,  @DefaultValue("2010") @QueryParam("maxyear") int maxyear)  {...} |

### @PathParam

含义：该标注是用来获取映射路径上的变量值供方法使用

示例：

|  |
| --- |
| @GET  @Path("/book/{isbn}")  public String getBook(@PathParam("isbn") String id) {  　　　……  }  } |

说明：

@PathParam("isbn ") 注解匹配前面的 @Path({"isbn"}) 注解。这个注解从 URI 模式 /books/{isbn} 提取 {isbn}，并将其注入 getBook方法的 id 参数的值。

访问URL：

GET http://myhost.com/services/book/333

## 数据返回格式的选择

1. XML

* 序列化耗时
* 返回的数据量点大

很想说的是twitter stream api抛弃了xml

1. JSON

* 简单清晰
* 数据包小

Taobao，amazon都返回“JSON和XML”2种格式

还有其它一些格式，Atom、RSS等。此处不详述。

## 代码示例

|  |
| --- |
| @GET  @Produces({MediaType.APPLICATION\_JSON})  @Path("/book/{id}")  public Book getBook(@PathParam("id") String id){  return new Book("huhu","huhu");  }  @PUT  @Path("/book/{name}")  public void updateBook(@PathParam("name") PathSegment book){  Iterator<Book> it = books.iterator();  String name = String.valueOf(book.getMatrixParameters().get("name"));  String content = String.valueOf(book.getMatrixParameters().get("content"));  while(it.hasNext()){  Book booktmp = it.next();  if(name.equals(booktmp.getName())){  booktmp.setContent(content);  break;  }  }  } |

# JAXM&SAAJ

JAXM(JAVA API For XML Message) 主要定义了包含了发送和接收消息所需的API，相当于Web服务的服务器端，其API位于javax.messaging.\*包，它是JAVA EE的可选包，因此SAAJ(SOAP With Attachment API For Java，JSR 67)是与JAXM搭配用的API，为构建SOAP包和解析SOAP包提供了重要的支持，支持附件传输，它在服务器端、客户端都需要使用。这里还要提到的是SAAJ规范，其API位于javax.xml.soap.\*包。JAXM&SAAJ与JAX-WS都是基于SOAP的Web服务，相比之下JAXM&SAAJ暴漏了SOAP更多的底层细节，编码比较麻烦，而JAX-WS更加抽象，隐藏了更多的细节，更加面向对象，实现起来你基本上不需要关心SOAP的任何细节。那么如果你想控制SOAP消息的更多细节，可以使用JAXM&SAAJ。

# JAX-RS 与 JAX-WS 的比较

|  |  |  |
| --- | --- | --- |
|  | JAX-RS (JSR 311) | JAX-WS (JSR 224) |
| 适用范围 | 适用于简单的远程数据访问 | 适用于复杂的数据交互模式 |
| 消息传输 | 请求和响应通过 HTTP 消息正文来传输 | 请求和响应被放在 SOAP 消息，作为传输层（HTTP, JMS, SMTP 等）消息正文进行传输 |
| 客户端调用 | 只需要一个 URL就能调用 Web 服务 | 需要编写标准的客户端代码，对 Web Services进行访问 |
| 安全性 | 简单的认证和授权机制 | 丰富的安全策略，可实现各种安全需求 |
| 访问终端 | 桌面浏览器或移动终端浏览器 | 桌面、浏览器 |

# JAXB

## 简介

JAXB (Java Architecture for XML Binding)是一种特殊的序列化/反序列化工具。它可以使XML数据以Java Objects的形式直接应用于Java程序之中，使Java Objects与XML数据之间的转换成为可能。这类工具还有XStream等。

在JAXB中将Java Objects到XML数据的转换称为marshal；XML数据到Java Objects的转换称为unmarshal。需要用到的主要类有JAXBContext、Marshaller、Unmarshaller。

## JAXB常用注解

### 模板

|  |
| --- |
| @XmlRootElement(name="product")  @XmlAccessorType(XmlAccessType.FIELD)  public class Product {  @XmlElement(required=true)  protected int id;  @XmlElement(required=true)  protected String name;  @XmlElement(required=true)  protected String description;  @XmlElement(required=true)  protected int price;    public Product() {}  } |

# Apache CXF

## 简介

Apache CXF = Celtix[一款开源ESB产品] + XFire，Apache CXF 的前身叫 Apache CeltiXfire，现在已经正式更名为Apache CXF 了，以下简称为CXF。CXF继承了Celtix和XFire两大开源项目的精华，提供了对JAX-WS全面的支持，并且提供了多种 Binding 、DataBinding、Transport以及各种Format的支持，并且可以根据实际项目的需要，采用代码优先（Code First）或者WSDL优先（WSDL First来轻松地实现Web Services的发布和使用。

Apache CXF是一个开源的 Web Service框架，CXF帮助您利用Frontend编程 API来构建和开发Services，像JAX-WS。这些Services可以支持多种协议，比如：SOAP、XML/HTTP、RESTful HTTP或者CORBA ，并且可以在多种传输协议上运行，比如：HTTP、JMS或JBI，CXF大大简化了Services的创建和发布，同时它继承了XFire传统，一样可以天然地和 Spring 进行无缝集成。在OSGi平台，它也支持blueprint的配置方式发布服务。

官网：http://cxf.apache.org/

## 功能特性

CXF 包含了大量的功能特性，但是主要集中在以下几个方面：

* 支持 Web Services 标准：CXF 支持多种 Web Services 标准，包含 SOAP、Basic Profile、WS-Addressing、WS-Policy、WS-ReliableMessaging和 WS-Security。
* Frontends：CXF 支持多种“Frontend”编程模型，CXF 实现了 JAX-WS API （遵循 JAX-WS 2.0 TCK 版本），它也包含一个“simple frontend”允许客户端和EndPoint的创建，而不需要 Annotation 注解。CXF 既支持 WSDL 优先开发，也支持从 Java 的代码优先开发模式。
* 容易使用： CXF 设计得更加直观与容易使用。有大量简单的 API 用来快速地构建代码优先的 Services，各种 Maven 的插件也使集成更加容易，支持 JAX-WS API ，支持 Spring 2.0 更加简化的 XML 配置方式，等等。
* 支持二进制和遗留协议：CXF 的设计是一种可插拨的架构，既可以支持 XML ，也可以支持非 XML 的类型绑定，比如：JSON 和 CORBA。

# 集成CXF(Rest支持)需要哪bundle支持

集成Apache CXF Rest支持涉及了大约30多个bundle，开发时统一提供给大家。

# CXF-JAXRS-Blueprint

目前CXF能和Spring等流行框架进行集成，相关流程框架也提供了CXF的配置支持，在OSGi领域，CXF也支持blueprint。

## Restful服务的支持

编写相应的服务类。如：

|  |
| --- |
| package com.chinasofti.samples.rs;  import java.util.List;  import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import javax.ws.rs.core.MediaType;  @Path("myService")  public interface MyService {  @GET  @Path("sayHello/{name}")  @Produces(MediaType.TEXT\_PLAIN)  String sayHello(@PathParam("name") String name);    @GET  @Path("getuser/{id}/")  @Produces({"application/json","application/xml"})  User getUser(@PathParam("id") String id);  @GET  @Path("getAllUsers")  @Produces({"application/json","application/xml"})  List<User> getUsers();  } |

## JAX-RS : Services Configuration

|  |
| --- |
| <blueprint xmlns="<http://www.osgi.org/xmlns/blueprint/v1.0.0>"             xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"             xmlns:jaxws="<http://cxf.apache.org/blueprint/jaxws>"             xmlns:jaxrs="<http://cxf.apache.org/blueprint/jaxrs>"             xmlns:cxf="<http://cxf.apache.org/blueprint/core>"             xsi:schemaLocation="  <http://www.osgi.org/xmlns/blueprint/v1.0.0><http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd>  <http://cxf.apache.org/blueprint/jaxrs><http://cxf.apache.org/schemas/blueprint/jaxrs.xsd>  <http://cxf.apache.org/blueprint/core><http://cxf.apache.org/schemas/blueprint/core.xsd>             ">       <cxf:bus>          <cxf:features>              <cxf:logging/>          </cxf:features>      </cxf:bus>        <jaxrs:server id="customerService" address="/customers">          <jaxrs:serviceBeans>             <ref component-id="serviceBean" />          </jaxrs:serviceBeans>       </jaxrs:server>        <bean id="serviceBean" class="service.CustomerService"/>  </blueprint> |

访问地址

<http://localhost:8080/cxf/myrest/myService/getAllUsers?_type=xml>

<http://localhost:8080/cxf/myrest/myService/getAllUsers?_type=json>

# 附录

## HTTP

### 请求头

### 状态码

* 状态码(3位数字)分类
  + 1xx:**通知**——仅在与HTTP服务器沟通时使用
  + 2xx:**成功**——成功收到、理解和接受动作
    - 200(“OK”)、201(“Created”)、204(“No Content”)
  + 3xx:**重定向**——为完成请求，必须进一步采取措施
    - 301(“Moved Permanently”)、303(“See Other”)、304(“Not Modified”)、307(“Temporary Redirect”)
  + 4xx:**客户端错误**——请求包含错误的语法或不能完成
    - 400(“Bad Request”)、401(“Unauthorized”)、403(“Forbidden”)、404(“Not Found”)、405(“Method Not Allowed”)、406(“Not Acceptable”) 、409(“Conflict”)、410(“Gone”)
  + 5xx:**服务器端错误**——服务器不能完成明显合理的请求
    - 500(“Internal Server Error”)、503(“Service Unavailable”)

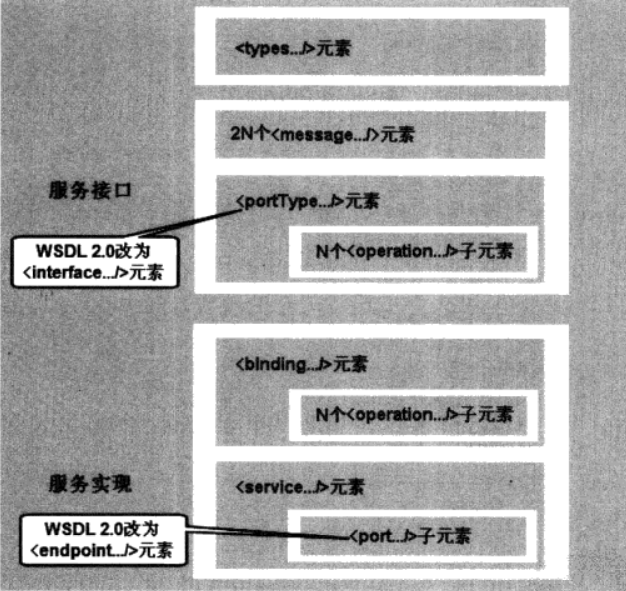
## WSDL

WSDL(Web Services Description Language)指网络服务描述语言，用XML描述Web Services、描述web services提供什么服务和调用这些服务需要什么参数及调用的返回值,以及调用这些服务的地址等信息。

### WSDL的主要元素

一个WSDL文档的主要结构是类似这样的：

|  |
| --- |
| <definitions>  <types>  definition of types........  </types>  <message>  definition of a message....  </message>  <portType>  definition of a port.......  </portType>  <binding>  definition of a binding....  </binding>  </definitions> |



* **Definitions**

是WSDL文档的根元素，definitions还声明各命名空间。

* **types**

数据类型定义的容器，它使用某种类型系统(一般地使用XML Schema中的类型系统)。

* **message**

<message> 元素定义一个操作的数据元素。

每个消息均由一个或多个部件组成。可以把这些部件比作传统编程语言中一个函数调用的参数。

* **portType**

<portType…/>元素用于描述一个Web Service所有可以被调用的操作，以及与该操作相关的消息。每个<portType…/>元素可以包含N个<operation…/>子元素，每个<operation…/>子元素定义一个允许被远程调用的操作。可以将该元素当作传统编程语言中的一个类，而每个<operation…/>子元素则可当成传统编程语言中的一个函数或方法。

* **binding**
* **service**

## SOAP

### 基本概念

SOAP是简单的对象访问协议（Simple Object Access Protocol）的简称。SOAP协议以XML作为数据传输格式，底层采用HTTP、SMTP等协议作为传输协议。不管是XML文档，还是HTTP、SMTP等协议，它们都是平台无关的，因此SOAP可以非常方便地解决异构系统相互通信的问题。SOAP是Web Service的根基。

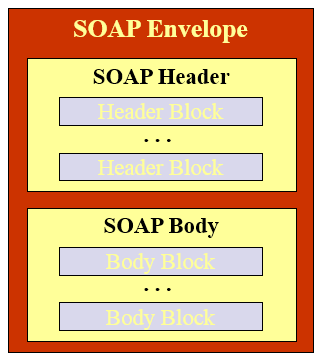
### SOAP消息结构

一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

* 必需的 Envelope 元素，SOAP消息的根元素
* 可选的 Header 元素，包含头部信息，可包含有关 SOAP 消息的应用程序专用信息（比如认证、支付等）。如果 Header 元素被提供，则它必须是 Envelope 元素的第一个子元素。
* 必需的 Body 元素，包含所有的调用和响应信息
* 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息，。在一条SOAP消息中，Fault元素只能出现一次。

|  |
| --- |
| <?xml version="1.0"?>  <soap:Envelope  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">  <soap:Header>  ...  ...  </soap:Header>  <soap:Body>  ...  ...  <soap:Fault>  ...  ...  </soap:Fault>  </soap:Body>  </soap:Envelope> |

如下图：



### SOAP消息示例

1. SOAP请求

|  |
| --- |
| <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  <soap:Body>  <ns2:sayHi xmlns:ns2="http://samples.ws.wms.csi.com/">  <name>chenxushao</name>  </ns2:sayHi>  </soap:Body></soap:Envelope> |

1. SOAP响应

|  |
| --- |
| <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  <soap:Body>  <ns2:sayHiResponse xmlns:ns2="http://samples.ws.wms.csi.com/">  <return>chenxushao,welcome!</return>  </ns2:sayHiResponse>  </soap:Body></soap:Envelope> |

更细节的SOAP介绍，可参考如下页面：

http://www.w3school.com.cn/soap/index.asp

## UDDI

UDDI 是一种目录服务，企业可以使用它对 Web services 进行注册和搜索。

UDDI，英文为 "Universal Description, Discovery and Integration"，可译为“通用描述、发现与集成服务”。

* 什么是 UDDI？

UDDI 是一个独立于平台的框架，用于通过使用 Internet 来描述服务，发现企业，并对企业服务进行集成。

UDDI 指的是通用描述、发现与集成服务

UDDI 是一种用于存储有关 web services 的信息的目录。

UDDI 是一种由 WSDL 描述的 web services 界面的目录。

UDDI 经由 SOAP 进行通信

UDDI 被构建入了微软的 .NET 平台

* UDDI 基于什么？

UDDI 使用 W3C 和 IETF\* 的因特网标准，比如 XML、HTTP 和 DNS 协议。

UDDI 使用 WSDL 来描述到达 web services 的界面

此外，通过采用 SOAP，还可以实现跨平台的编程特性，大家知道，SOAP 是 XML 的协议通信规范，可在 W3C 的网站找到相关的信息。

\*注释：IETF - Internet Engineering Task Force

* UDDI 的好处

任何规模的行业或企业都能得益于 UDDI。

在 UDDI 之前，还不存在一种 Internet 标准，可以供企业为它们的企业和伙伴提供有关其产品和服务的信息。也不存在一种方法，来集成到彼此的系统和进程中。

* UDDI 规范帮助我们解决的问题：

使得在成百万当前在线的企业中发现正确的企业成为可能

定义一旦首选的企业被发现后如何启动商业

扩展新客户并增加对目前客户的访问

扩展销售并延伸市场范围

满足用户驱动的需要，为在全球 Internet 经济中快速合作的促进来清除障碍

* UDDI 如何被使用

假如行业发布了一个用于航班比率检测和预订的 UDDI 标准，航空公司就可以把它们的服务注册到一个 UDDI 目录中。然后旅行社就能够搜索这个 UDDI 目录以找到航空公司预订界面。当此界面被找到后，旅行社就能够立即与此服务进行通信，这样由于它使用了一套定义良好的预订界面。

* 谁在支持 UDDI？

UDDI 是一个跨行业的研究项目，由所有主要的平台和软件提供商驱动，比如：Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, 以及 Sun, 它既是一个市场经营者的团体，也是一个电子商务的领导者。

已有数百家公司参与了这个 UDDI 团体。

* UDDI现状

UDDI的前景是非常美好的，它希望企业之间通过UDDI提供的公共目录服务，动态发现并调用互联网上提供的服务。但这基于一个假设：所有企业都愿意动态地建立和其它企业之间的业务关系，即使以前这二者并无合作关系。这个假设在现实生活中却遇到了极大的困难，因此UDDI并没有获得真正的成功。2005.12，IBM、Microsoft和SAP停止了面向公众开放的UDDI注册中心。

由于UDDI实际上并没有赢得广泛的支持，然而Web Service的注册和发现依然是非常重要的，因此新的标准和产品正在不断出现。

由于UDDI在实际应用中的制度，因此不同企业之间通过Web Service进行交互的可能性依然不大。但在某个企业内部的不同部门之间，不同平台的异构系统之间，却可以广泛地使用Web Service相互通信。这很容易理解：虽然外部的UDDI注册中心没有得到广泛支持，但在企业内部，所有部门、所有开发者可以非常方便地了解对方所能提供的Web Service，因此可以非常方便地调用它们。