**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 | 2013-9-4 | 创建 | 陈绪绍 |
| V |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 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 Service、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 Service的表现形式

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。

Rest更轻量级。

# Rest

## 简介

REST 是英文 Representational State Transfer 的缩写，这个术语由 Roy Thomas Fielding 博士在他的论文《Architectural Styles and the Design of Network-based Software Architectures》中提出。从这篇论文的标题可以看出：REST 是一种基于网络的软件架构风格。

REST 架构是针对传统 Web 应用提出的一种改进，是一种新型的分布式软件设计架构。对于异构系统如何进行整合的问题，目前主流做法都集中在使用 SOAP、WSDL 和 WS-\* 规范的 Web Services。而REST架构实际上也是解决异构系统整合问题的一种新思路。

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

如果开发者在开发过程中能坚持 REST 原则，将可以得到一个使用了优质 Web 架构的系统，从而为系统提供更好的可伸缩性，并降低开发难度。关于 REST 架构的主要原则如下：

* 网络上的所有事物都可被抽象为资源（Resource）。
* 每个资源都有一个唯一的资源标识符（Resource Identifier，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”）。
* 同一资源具有多种表现形式(ATOM、XML、JSON)。
* 使用标准方法操作资源(HTTP 的 POST，GET，PUT， DELETE等方法)
* 通过缓存来提高性能。
* 对资源的各种操作不会改变资源标识符。
* 所有的操作都是无状态的（Stateless，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）。

## Rest催生出了新的开发架构

## 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。

## 问题

为何说基于 SOAP 的服务是重量级的服务，Rest是轻量级的？

## 概念总结

Restful Web服务是一种基于 REST 和 HTTP 协议的轻量级Web 服务，它把Web应用系统中的一切都当作是资源，它利用标准的HTTP请求方法（GET、POST、PUT 和 DELETE等），以URL的形式访问(功能调用)Web资源。

# Java对Web Service的支持

JAVA中共有三种Web Service规范，分别JAX-WS(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。

使用最多的是JAX-WS和JAX-RS。

# 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 |

# 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。

支持接口和普通Java类。

## 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：

|  |  |
| --- | --- |
| @Path("/helloworld")  **public** **class** HelloWorld {  @GET  @Path("/echo")  **public** String Echo(){  **return** "hello world";  }  } | @Path("/")  **public** **class** HelloWorld {  @GET  @Path("/sayHi/{name}")  **public** String sayHi(@PathParam("name") String name) {  **return** "Hello," + name + "!";  }  ….  } |

**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 (XML)

MediaType.APPLICATION\_JSON (JSON)

MediaType.APPLICATION\_FORM\_URLENCODED (表单)

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

### @FormParam

When the input request body is of the type "application/x-www-form-urlencoded", a.k.a. an HTML Form, you can inject individual form parameters from the request body into method parameter values.

|  |
| --- |
| <form method="POST" action="/resources/service">  First name:  <input type="text" name="firstname">  <br>  Last name:  <input type="text" name="lastname">  </form> |

If you post through that form, this is what the service might look like:

|  |
| --- |
| @Path("/")  public class NameRegistry {  @Path("/resources/service")  @POST  public void addName(@FormParam("firstname") String first, @FormParam("lastname") String last) {...} |

### @DefaultValue

@DefaultValue is a parameter annotation that can be combined with any of the other @\*Param annotations to define a default value when the HTTP request item does not exist.

|  |
| --- |
| @GET  public String getBooks(@QueryParam("num") @DefaultValue("10") int num) {...} |

## 数据返回格式的选择

1. XML

* 序列化耗时
* 返回的数据量点大

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。

## JAX-RS注解示例

### @GET、@POST、@PUT、@DELETE、@PATH

|  |
| --- |
| @Path("/library")  public class Library {  @GET  @Path("/books")  public String getBooks() {...}  @GET  @Path("/book/{isbn}")  public String getBook(@PathParm("isbn") String isbn) {  // search my database and get a string representation and return it  }  @PUT  @Path("/book/{isbn}")  public void addBook(@PathParam("isbn") String id, @QueryParam("name") String name) {...}  @DELETE  @Path("/book/{id}")  public void removeBook(@PathParam("id") String id ){...}  } |

### @QueryParam

|  |
| --- |
| @GET  @Produces("application/json")  @Path("json/companyList")  public CompanyList getJSON(@QueryParam("start") int start, @QueryParam("limit") int limit) {    CompanyList list = new CompanyList(companyService.listCompanies(start, limit));    return list;  } |

### @PathParam

|  |
| --- |
| @Path("/library")  public class Library {  @GET  @Path("/book/{isbn}")  public String getBook(@PathParm("isbn") String id) {  // search my database and get a string representation and return it  }  } |

# 物流平台Rest开发约定与规范

## 顶级资源只使用”/”路径

|  |
| --- |
| @Path("/")  **public** **interface** UserService {  @GET  @Path("getUser/{id}/")  @Produces({"application/json"})  User getUser(@PathParam("id") String id);  …….  } |

## 一般只使用@GET(查询)、@POST(创建、插入)、@PUT(更新)、@DELETE(删除)

# 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

Wsdl文档示例

### portType

### types

### binding

### message

### port

### service

## web service其它技术

JAXR

SAAJ

StAX

SEI Service Endpoint Interface

SIB

SOAP

UDDI

XML

Schema