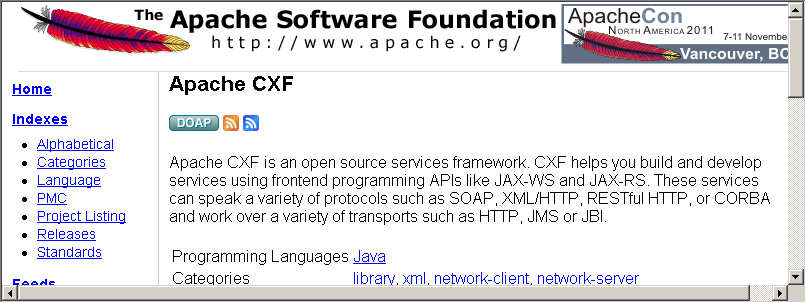
# 什么是cxf

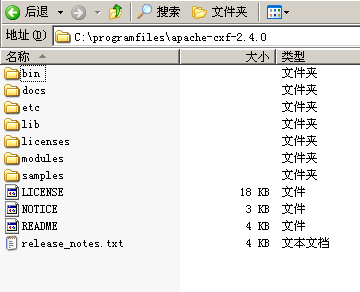
Apache CXF = Celtix + Xfire

支持多种协议：

* 1. SOAP1.1,1,2
  2. HTTP
  3. CORBA（Common Object Request Broker Architecture公共对象请求代理体系结构,早期语言使用的WS。C,c++,C#）
  4. 并可以与Spring进行快速无缝的整合
  5. 灵活的部署：可以运行有Tomcat,Jboss,Jetty(内置),IBMWS,BeaWS上面。



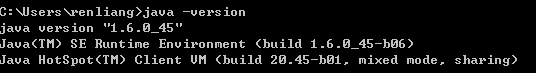




bin（目录）   
     bin 目录中是 CXF 框架中所提供的代码生成、校验、管理控制台工具（可执行命令）   
  
**docs（目录）**   
   CXF 所有类（class）对应的 API 文档，为开发者使用 CXF 完成应用开发提供应有的帮助。   
  
**etc（目录）**   
    包含一个基本的 Service 暴露所需要的 web.xml 文件，及其它的配置文件。   
  
**lib（目录）**   
    lib 目录中包含 CXF 及其运行时所需要的和可选的第三方支持类包（.jar 文件），可以根据不同项目所需的 CXF 特性选择所需要的支持类包。如果不想一一去区分的话，可   
以直接在 Web 项目中包含所有的 CXF 及其运行时所需要的第三方支持类包（.jar 文件）即可。   
其中 cxf-2.0.2-incubator.jar 是 CXF 框架的二进制包文件，包含了全部的模块（modules），cxf-manifest-incubator.jar 是列表清单文件 manifest jar 。   
  
**以下的 jar 包是所有 CXF 项目所必需的：**   
cxf.jar   
commons-logging.jar   
geronimo-activation.jar (Or the Sun equivalent)   
geronimo-annotation.jar (Or the Sun equivalent)   
geronimo-javamail.jar (Or the Sun equivalent)   
neethi.jar   
jaxb-api.jar   
jaxb-impl.jar   
stax-api.jar   
XmlSchema.jar   
wstx-asl.jar   
xml-resolver.jar   
  
**对于 Java2WSDL 和 WSDL2Java，除了必需的之外，还需要再增加如下 jar 包：**   
jaxb-xjc.jar   
veliocity.jar   
velocity-dep.jar   
  
**为了支持 JAX-WS ，除了必需的之外，还需要再增加如下 jar 包：**   
jaxws-api.jar   
saaj-api.jar   
saaj-impl.jar   
asm.jar （可选的，但是可以提升包装类型的性能）   
  
**为了支持 XML 配置，除了必需的之外，还需要再增加如下 jar 包：**aopalliance.jar   
spring-beans.jar   
spring-context.jar   
spring-core.jar   
spring.web.jar   
  
**为了独立的 HTTP 服务支持，除了必需的之外，还需要再增加如下 jar 包：**geronimo-servlet.jar   
jetty.jar   
jetty-sslengine.jar   
jetty-util.jar   
sl4j.jar & sl4j-jdk14.jar （可选的，但是可以提升日志 logging）   
  
**为了支持 Aegis ，除了必需的之外，还需要再增加如下 jar 包：**   
jaxen.jar   
jdom.jar   
stax-utils.jar   
  
**为了支持 WS-Security ，除了必需的之外，还需要再增加如下 jar 包：**bcprov-jdk14.jar   
wss4j.jar   
xalan.jar   
xmlsec.jar   
  
**为了支持 HTTP Binding ，除了必需的之外，还需要再增加如下 jar 包：**jra.jar   
jettison.jar （仅为 JSON 服务所需的）   
  
**licenses（目录）**   
列表了引用第三方 jar 包的相关许可协议。   
  
**modules（目录）**   
modules 目录中包含了 CXF 框架根据不同特性分开进行编译的二进制包文件。发布基于 CXF 框架的 Web 项目时，可以选择使用该目录下的所有 .jar 文件，也可以选择 lib 目 录中的 cxf-2.0.2-incubator.jar 文件。   
  
**samples（目录）**   
samples 目录中包含了所有随 CXF 二进制包发布的示例，包含这些示例的源代码和相关 Web 应用配置文件，可以方便地用 Ant 来编译运行测试这些示例，来了解 CXF 的开发和   
  
使用的方法。可以通过 samples 目录和它各个子目录下的 README.txt 的文件来详细了解示例的编译与运行的步骤。

# 环境搭建

* 1. JAVA\_HOME 需要jdk的支持



* 1. CXF\_HOME

1. 
   1. ANT\_HOME

C:\Users\renliang\AppData\Roaming\Tencent\Users\491793587\QQ\WinTemp\RichOle\_OC1`$RL_73NZS~@4P06DN4.jpg

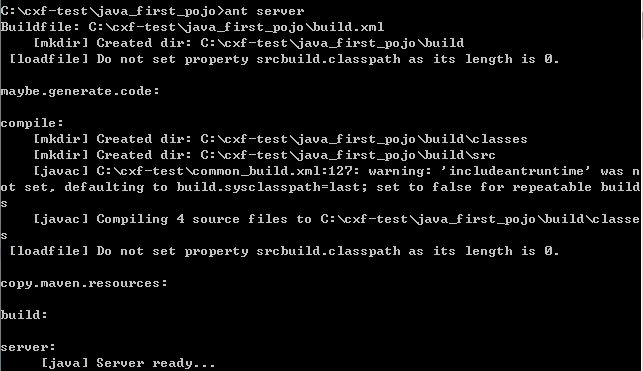
* 1. CATALINA\_HOME

1. 
   1. Path = %JAVA\_HOME%\bin;%CXF\_HOME%\bin;%CATALINA\_HOME%\bin;%ANT\_HOME%\bin
   2. CLASSPATH=.;%CXF\_HOME%\lib\cxf-manifest.jar;.\build\classes

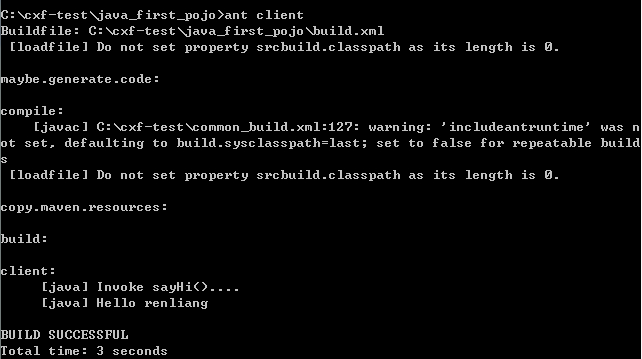
# 3.运行cxf例子

1.拷贝例子中的common\_build.xml和java\_first\_pojo到没有中文目录的下

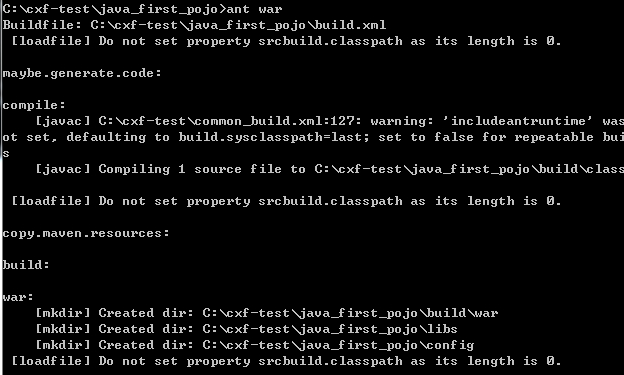
2. 执行ant server



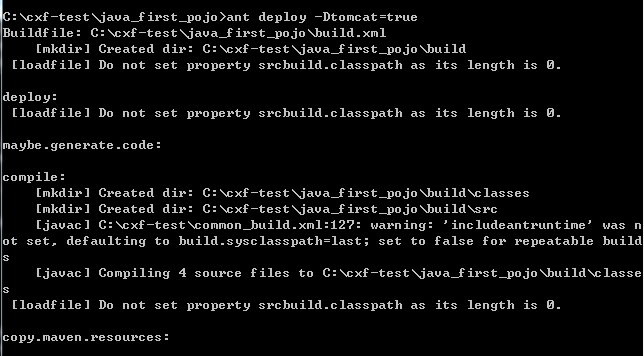
3.执行ant client



1. 执行ant war



5. 执行ant deploy –Dtomcat=true将项目发布到tomcat的服务器上。

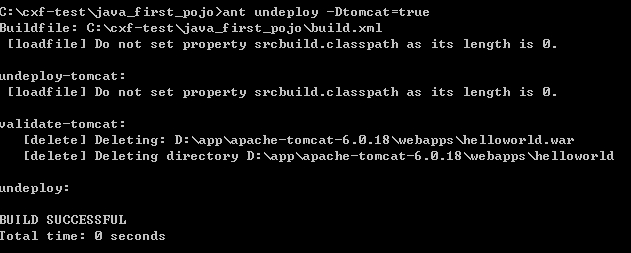


发布到tomcat中：访问<http://localhost:8080/helloworld/services/hello_world?wsdl>

服务路径由cxf-servlet.xml来配置

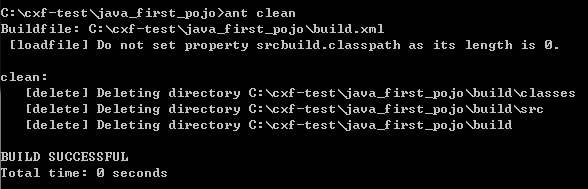
6.卸载部署包

停止tomcat后卸载应用：ant undeploy –Dtomcat=true



7.清理构建目录

ant clean



# 实现cxf第一个示例

1. 创建java项目
2. 引入所有依赖包
3. 创建服务类

用两个不同的类发布应用：

ServerFactoryBean(不需要使用@webservice) 生成的文档不规范，不建议使用

JaxWsServerFactoryBean(建议使用此类，需要使用@webservice) 生成的文档不规范，可以发布SOAP1.1，SOAP1.2的协议，当cxf的服务类中没有方法时也可以发布成功，不报错。如果使用SOAP1.2需要用@bindType注解指定

当使用SOAP1.2时wsimport命令失效，需要使用cxf的wsdl2java

建议：发布服务的时候使用SOAP1.2，客户端调用的时候使用SOAP1.1

@WebService

@BindingType(value=javax.xml.ws.soap.SOAPBinding.*SOAP11HTTP\_BINDING*)

**public** **class** HelloService {

**public** **static** **void** main(String[] args) {

//创建服务工厂对象

//ServerFactoryBean sfb = **new** ServerFactoryBean();

JaxWsServerFactoryBean sfb = **new** JaxWsServerFactoryBean ();

//设置服务地址

sfb.setAddress("http://127.0.0.1:7777/hello");

//设置服务类

sfb.setServiceClass(HelloService.**class**);

//设置服务对象的实例

sfb.setServiceBean(**new** HelloService());

//发布服务

sfb.create();

}

**public** String sayHello(String name){

**return** name+ " hello";

}

}

1. 通过wsimport（SOAP1.1）生成客户端代码
2. 调用webservice

# wsdl2java

在cxf中，也提供了一个用于生成客户端调用代码的工具。它的功能就如同wsimport一样。

先让我们了解一下cxf的wsdl2java工具，可以生成一堆客户端调用的代码。既可以生成SOAP1.1也可以生成SOAP1.2

此工具位于cxf\_home/bin目录下。参数与wsimport有所不同。

它包含以下参数：

* 1. －d参数，指定代码生成的目录。
  2. －p参数，指定生成的新的包结构。

需要说明的是，由于wsdl2java是根据**jdk1.7**生成的本地代码，所以，需要对生成的代码做一点点修改。

在命令行执行：

wsdl2java –d **.** <http://127.0.0.1:6666/helloworld?wsdl>

C:\Users\renliang\AppData\Roaming\Tencent\Users\491793587\QQ\WinTemp\RichOle\CM9K2_{4B5F{2J8D0DML4KA.jpg

SOAP1.2消息

POST /WebServices/MobileCodeWS.asmx HTTP/1.1

Host: webservice.webxml.com.cn

Content-Type: application/soap+xml; charset=utf-8

Content-Length: length

<?xml version="1.0" encoding="utf-8"?>

<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">

<soap12:Body>

<getMobileCodeInfo xmlns="http://WebXml.com.cn/">

<mobileCode>string</mobileCode>

<userID>string</userID>

</getMobileCodeInfo>

</soap12:Body>

</soap12:Envelope>

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: length

<?xml version="1.0" encoding="utf-8"?>

<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">

<soap12:Body>

<getMobileCodeInfoResponse xmlns="http://WebXml.com.cn/">

<getMobileCodeInfoResult>string</getMobileCodeInfoResult>

</getMobileCodeInfoResponse>

</soap12:Body>

</soap12:Envelope>

# 7.使用JaxWsServerFactoryBean调用接口服务类

1.创建服务接口

@WebService//注意注解加在接口上

**public** **interface** HI {

**public** String sayHi(String name);

}

2.创建接口实现类

**public** **class** HIImpl **implements** HI {

@Override

**public** String sayHi(String name) {

// **TODO** Auto-generated method stub

**return** name + " hello";

}

}

3.创建发布服务类

**public** **static** **void** main(String[] args) {

JaxWsServerFactoryBean jf = **new** JaxWsServerFactoryBean();

jf.setAddress("http://127.0.0.1:5555/hi");

jf.setServiceClass(HI.**class**);

jf.setServiceBean(**new** HIImpl());

jf.create();

}

4. 生成客户端代码

C:\Users\renliang\AppData\Roaming\Tencent\Users\491793587\QQ\WinTemp\RichOle\92W1SYQ}B%7SS(XNHRD_]V3.jpg

5.调用客户端

**public** **class** Client {

**public** **static** **void** main(String[] args) {

HIService hs = **new** HIService();

HI serviceClass = hs.getHIPort();

System.*out*.println(serviceClass.sayHi("张三"));

}

}

# 8.拦截输入输出消息

LoggingInInterceptor – 信息输入时的拦截器 –请求

LoggingOutInterceptor –信息输出时的拦截器-响应

**public** **class** InterServer {

**public** **static** **void** main(String[] args) {

JaxWsServerFactoryBean jf = **new** JaxWsServerFactoryBean();

jf.getInInterceptors().add(**new** LoggingInInterceptor());

jf.getOutInterceptors().add(**new** LoggingOutInterceptor());

jf.setAddress("http://127.0.0.1:5555/hi");

jf.setServiceClass(HI.**class**);

jf.setServiceBean(**new** HIImpl());

jf.create();

}

}

# 9.在web项目中创建类的cxf服务

1.创建web项目

2.导入所有包

3.创建服务类,必须指定注解@webService

4.配置web.xml,

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>cxf</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>cxf</servlet-name>

<url-pattern>/services/\*</url-pattern>

</servlet-mapping>

<session-config>

<session-timeout>60</session-timeout>

</session-config>

</web-app>

1. 创建cxf的核心配置文件cxf-servlet.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:jaxws=*"http://cxf.apache.org/jaxws"*

xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"* xmlns:cxf=*"http://cxf.apache.org/core"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd*

*http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd*

*http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd"*>

<!-- 引入CXF Bean定义如下,早期的版本中使用 -->

<import resource=*"classpath:META-INF/cxf/cxf.xml"* />

<import resource=*"classpath:META-INF/cxf/cxf-extension-soap.xml"* />

<import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"* />

<jaxws:endpoint id=*"hello"* address=*"/hello"* implementor=*"com.rl.cxf.web.server.HelloService"*>

<jaxws:outInterceptors>

<bean class=*"org.apache.cxf.interceptor.LoggingInInterceptor"*></bean>

</jaxws:outInterceptors>

<jaxws:inInterceptors>

<bean class=*"org.apache.cxf.interceptor.LoggingOutInterceptor"*></bean>

</jaxws:inInterceptors>

</jaxws:endpoint>

</beans>

# 10修改cxf-servlet.xml的位置和文件名

1.第一种方式

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>cxf</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

<init-param>

<param-name>config-location</param-name>

<param-value>classpath:cxf.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>cxf</servlet-name>

<url-pattern>/services/\*</url-pattern>

</servlet-mapping>

</web-app>

2.第二种方式

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:cxf.xml</param-value>

</context-param>

<servlet>

<servlet-name>cxf</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>cxf</servlet-name>

<url-pattern>/services/\*</url-pattern>

</servlet-mapping>

</web-app>

# 11在web项目中创建接口的cxf服务

1. 创建服务接口在接口上加@webservice
2. 创建服务接口的实现类
3. 在web.xml中配置CXFServlet
4. 配置cxf-servlet.xml

<jaxws:server id=*"bye"* address=*"/bye"* serviceClass=*"com.rl.cxf.web.inter.ByeInter"*>

<jaxws:serviceBean>

<bean class=*"com.rl.cxf.web.inter.ByeInterImpl"*></bean>

</jaxws:serviceBean>

<jaxws:outInterceptors>

<bean class=*"org.apache.cxf.interceptor.LoggingInInterceptor"*></bean>

</jaxws:outInterceptors>

<jaxws:inInterceptors>

<bean class=*"org.apache.cxf.interceptor.LoggingOutInterceptor"*></bean>

</jaxws:inInterceptors>

</jaxws:server>

1. 使用wsdl2java生成客户端代码
2. 调用webservice

# 12使用jquery调用cxf

$(**function**(){

$("#mybutton").click(**function**(){

**var** data = '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:q0="http://server.web.cxf.rl.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">'

+'<soapenv:Body>'

+'<q0:sayHello>'

+' <arg0>sss</arg0>'

+' </q0:sayHello>'

+'</soapenv:Body>'

+'</soapenv:Envelope>';

$.ajax({

url:'http://localhost:8080/cxf-web-server/services/hello',

type:'post',

dataType:'xml',

contentType:'text/xml;charset=UTF-8',

data:data,

success:**function**(responseText){

alert($(responseText).find('return').text());

},

error:**function**(){

alert("error");

}

})

})

})

# 13.jquery调用cxf的ws小练习

实现添加人，和查询人。

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

<script type="text/javascript" src="js/jquery-1.6.2.js"></script>

<script type="text/javascript">

$(function(){

//保存用户

$("#save").click(function(){

var name = $("#name").val();

var age = $("#age").val();

var gender = $("#gender").val();

$.ajax({

url:"http://localhost:8080/cxf\_web\_server/services/person",

type:"POST",

dataType:"xml",

contentType:"text/xml;charset=UTF-8",

data:'<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:q0="http://server.itcast.cn/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">'

+'<soapenv:Body>'

+'<q0:addPerson>'

+'<arg0>'

+'<age>'+age+'</age> '

+'<gender>'+gender+'</gender> '

+'<name>'+name+'</name> '

+'</arg0>'

+'</q0:addPerson>'

+'</soapenv:Body>'

+'</soapenv:Envelope>',

success:function(data){

alert('保存成功');

},

error:function(){

alert("system error");

}

});

});

//查询用户

$("#query").click(function(){

$.ajax({

url:"http://localhost:8080/cxf\_web\_server/services/person",

type:"POST",

dataType:"xml",

contentType:"text/xml;charset=UTF-8",

data:'<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:q0="http://server.itcast.cn/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">'

+' <soapenv:Body>'

+' <q0:queryPersonAll /> '

+' </soapenv:Body>'

+' </soapenv:Envelope>',

success:function(data){

var doc = $(data).find('return');

$("#mydiv").empty();

var result = '';

doc.each(function(){

var name = $(this).find('name').text();

var age = $(this).find('age').text();

var gender = $(this).find('gender').text();

result = result + name + " "+ age + " " + gender + "<br>";

});

$("#mydiv").append(result);

},

error:function(){

alert("system error");

}

});

});

});

</script>

</head>

<body>

姓名：<input id="name" type="text"><br>

年龄：<input id="age" type="text"><br>

性别：<input id="gender" type="text"><br>

<input id="save" type="button" value="保存"><input id="query" type="button" value="查询"><br>

<div id="mydiv" style="width:300px;height:300px; border:1px solid;">

</div>

</body>

</html>