1. 用JAX-WS在Tomcat中发布WebService

JDK中已经内置了Webservice发布，不过要用Tomcat等Web服务器发布WebService，还需要用第三方Webservice框架。Axis2和CXF是目前最流行的Webservice框架，这两个框架各有优点，不过都属于重量级框架。

JAX-WS RI是JAX WebService参考实现。相对于Axis2和CXF，JAX-WS RI是一个轻量级的框架。虽然是个轻量级框架，JAX-WS RI也提供了在Web服务器中发布Webservice的功能。官网地址https://jax-ws.java.net/。下面用JAX-WS RI在Tomcat中发布WebService。

博客中的实例代码
http://download.csdn.net/detail/accountwcx/8922191

服务端

新建一个Maven Web项目，在项目中添加JAX-WS RI引用，pom.xml配置文件如下

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.rvho</groupId>
 <artifactId>jaxwsserver</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>
 <properties>
  <!--  -->
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <!--  -->
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <!--  -->
  <maven.compiler.encoding>UTF-8</maven.compiler.encoding>
 </properties>
 <dependencies>
  <!-- JAXWS-RI -->
  <dependency>
```

```
   <groupId>com.sun.xml.ws</groupId>
   <artifactId>jaxws-rt</artifactId>
   <version>2.2.10</version>
  </dependency>
 </dependencies>
</project>
```

创建服务接口

```
package com.rvho.server.ws;
import java.util.Date;
import javax.jws.WebService;
/**
 * WebService
 */
@WebService(name = "HelloWS", targetNamespace = "http://www.tmp.com/ws/hello")
public interface HelloWService {
 /**
  *
  *
  * @return
  */
 String index();
 /**
  *
  *
  * @param x
  * @param y
  * @return
  */
 Integer add(Integer x, Integer y);
 /**
  *
  *
  * @return
  */
 Date now();

 /**
  *
  * @param name
  * @param age
  * @return
  */
 PersonEntity getPerson(String name, Integer age);
}
```

创建服务接口实现类(SEI)

```
package com.rvho.server.ws.impl;
import java.util.Date;
import javax.jws.WebService;
import com.rvho.server.entity.PersonEntity;
import com.rvho.server.ws.HelloWService;
@WebService(
  endpointInterface = "com.rvho.server.ws.HelloWService",
  portName = "HelloWSPort",
  serviceName = "HelloWSService",
  targetNamespace = "http://www.tmp.com/ws/hello")
public class HelloWServiceImpl implements HelloWService {
 public String index() {
  return "hello";
 }
 public Integer add(Integer x, Integer y) {
  return x + y;
 }
 public Date now() {
  return new Date();
 }

 public PersonEntity getPerson(String name, Integer age) {
  PersonEntity person = new PersonEntity();
  person.setAge(age);
  person.setName(name);

  return person;
 }
}
```

服务中用到的复杂类型PersonEntity

```
package com.rvho.server.entity;
import java.io.Serializable;
public class PersonEntity implements Serializable {
 private static final long serialVersionUID = -7211227324542440039L;

 private String name;
 private Integer age;

 public String getName() {
  return name;
 }
```

```
 public void setName(String name) {
  this.name = name;
 }
 public Integer getAge() {
  return age;
 }
 public void setAge(Integer age) {
  this.age = age;
 }
}
```

在WEB-INF中创建WebService配置文件sun-jaxws.xml，配置文件中一个WebService对应一个Endpoint。

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
 <!-- http:///services/hello -->
 <endpoint name="hello" implementation="com.rvho.server.ws.impl.HelloWServiceImpl" url-pattern="/services/hello" />
</endpoints>
```

在web.xml中添加WSServlet，如果Web项目使用Servlet 3.0则不需要以下配置。

```
<!-- Servlet 3.0 -->
<servlet>
 <servlet-name>jaxws</servlet-name>
 <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
 <servlet-name>jaxws</servlet-name>
 <url-pattern>/services</url-pattern>
</servlet-mapping>
```

发布服务后，在浏览器中输入http://<网站路径>/services/hello可以看到如下页面

http://blog.csdn.net/

客户端

在JDK的bin文件夹中，提供了一个根据wsdl生成java类的工具wsimport.exe。

```
: wsimport [options] <WSDL_URI>
 [options] :
  -b <path>                  jaxws/jaxb
                            ( <path>  -b)
  -B<jaxbOption>             JAXB
  -catalog <file>
                             TR9401, XCatalog  OASIS XML
  -d <directory>
  -encoding <encoding>
  -extension                 -



  -help
  -httpproxy:<host>:<port>   HTTP  ( 8080)
  -keep
  -p <pkg>
  -quiet                     wsimport
  -s <directory>
  -target <version>          JAXWS
                             2.2,  2.0, 2.1  2.2
                            , 2.0  JAXWS 2.0
  -verbose
  -version
  -wsdllocation <location>  @WebServiceClient.wsdlLocation
  -clientjar <jarfile>       Artifact  jar
                             Web  WSDL
  -generateJWS               JWS
  -implDestDir <directory>   JWS
  -implServiceName <name>    JWS
  -implPortName <name>       JWS
:
  -XadditionalHeaders
                                   Java
  -Xauthfile                       :
                                   http://username:password@example.org/stock?wsdl
  -Xdebug
  -Xno-addressing-databinding       W3C EndpointReferenceType  Java
  -Xnocompile                       Java
  -XdisableAuthenticator            JAX-WS RI ,
```

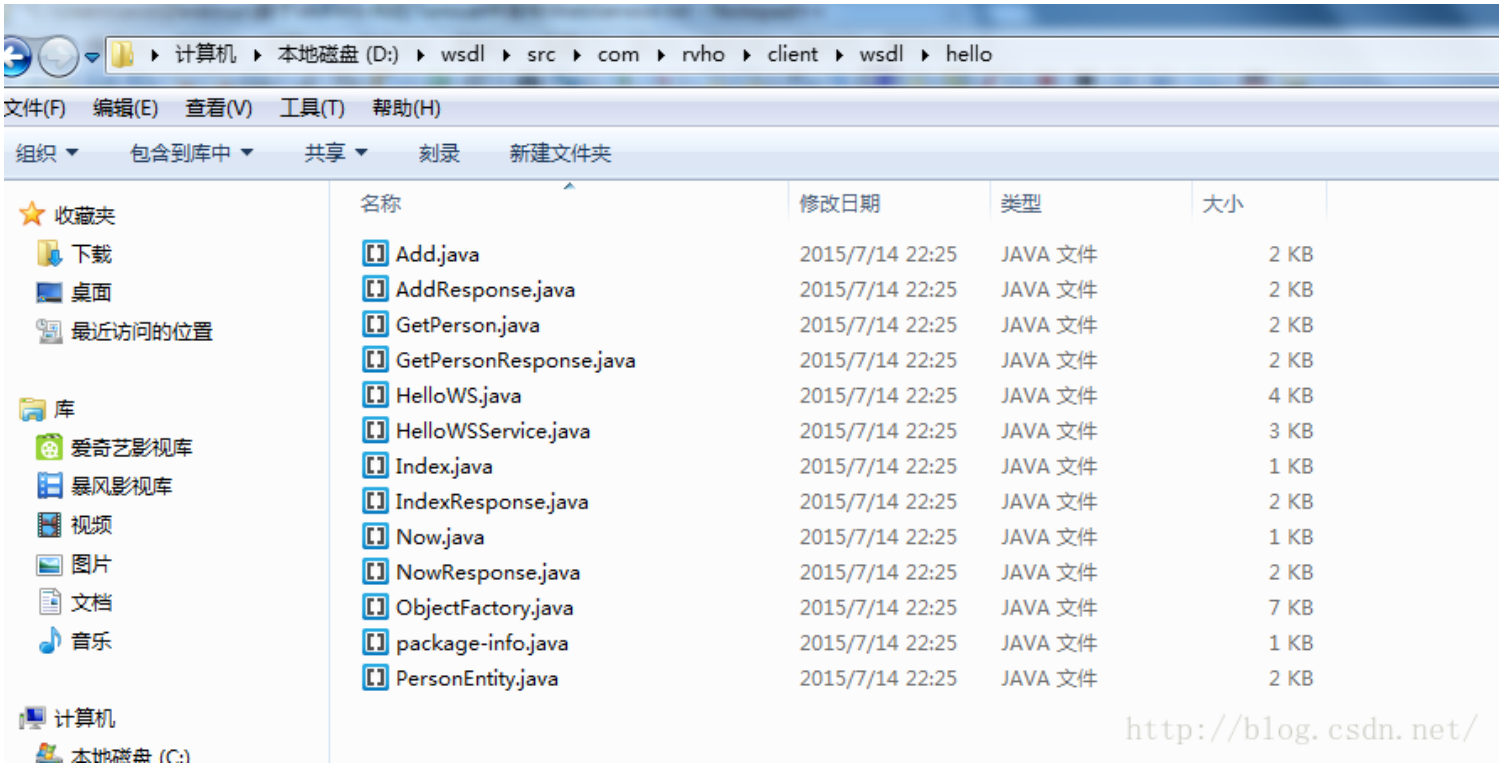
```
                                    -Xauthfile  ()
  -XdisableSSLHostnameVerification  wsdl  SSL

:
  wsimport stock.wsdl -b stock.xml -b stock.xjb
  wsimport -d generated http://example.org/stock?wsdl
```

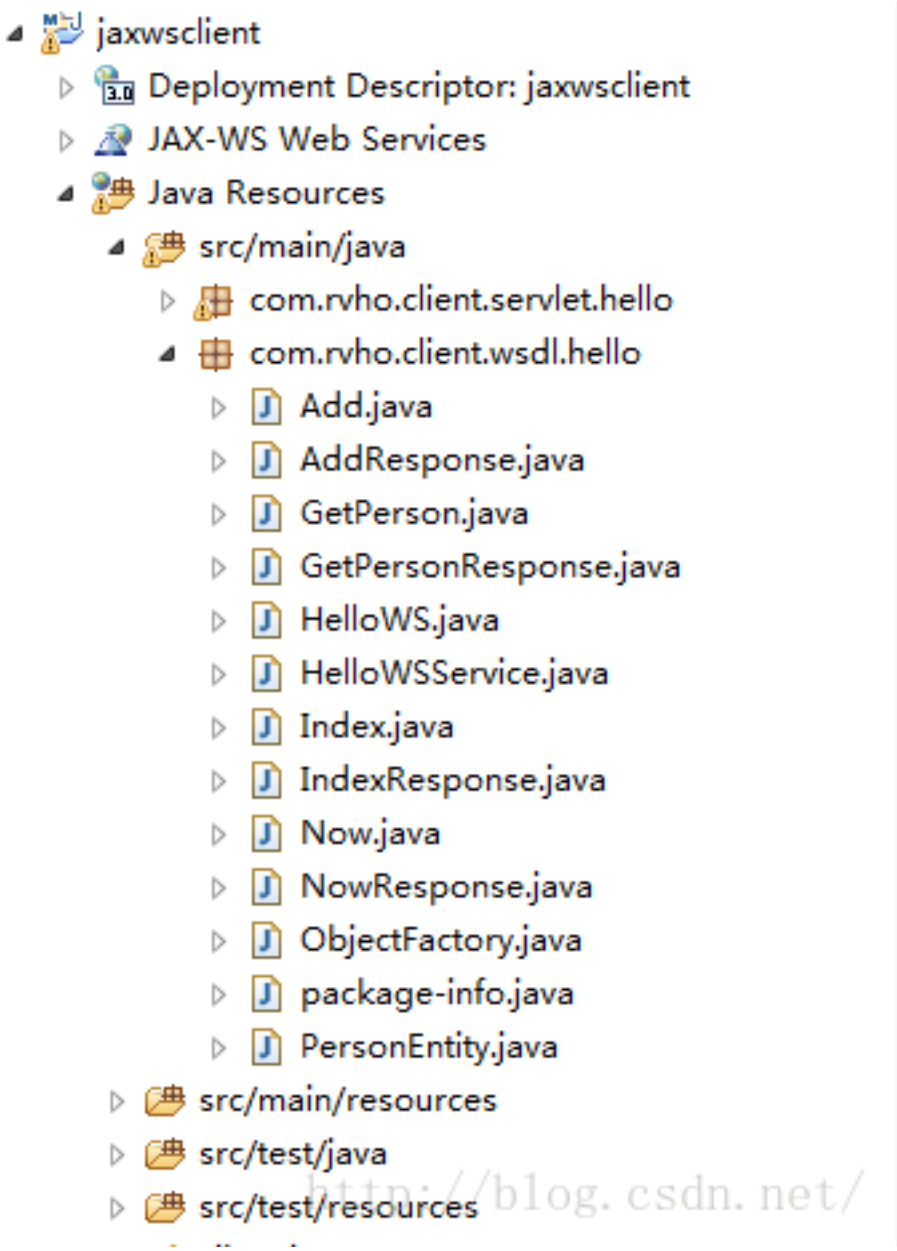
输入以下命令，即可生成Java类

```
D:\Program Files\Java\jdk1.8.0_25\bin>wsimport.exe -encoding utf-8 -p
com.rvho.client.wsdl.hello -d d:\wsdl\compile -s d:\wsdl\src
http://localhost:8014/jaxwsserver/services/hello?wsdl
```

最后生成的客户端Java类



把生成的Java类添加到客户端相应的Package下



在客户端调用服务

```
package com.rvho.client.wsdl.hello;
import java.net.URL;
public class Client {
 public static void main(String[] args) throws Exception {

  URL wsdlUrl = new URL("http://localhost:8014/jaxwsserver/services/hello?wsdl");
  HelloWSService helloWSS = new HelloWSService(wsdlUrl);
  HelloWS helloWS = helloWSS.getHelloWSPort();

  Integer x = 3;
  Integer y = 5;
  Integer add = helloWS.add(x, y);
```

```
  System.out.println("add");
  System.out.println("3 + 5 = " + add);
  System.out.println("");

  String name = "";
  Integer age = 19;
  PersonEntity person = helloWS.getPerson(name, age);
  System.out.println("getPerson");
  System.out.println("name = " + person.getName() + " age = " + person.getAge());
  System.out.println("");
 }
}
```

注意

JAXWS-RI在Tomcat 8中部署，调用服务时会有如下错误

```
: onComplete() failed for listener of type
[org.apache.catalina.core.AsyncListenerWrapper]
java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or
any of the dispatch() methods has been called
 at org.apache.catalina.core.AsyncContextImpl.getRequest(AsyncContextImpl.java:225)
 at
com.sun.xml.ws.transport.http.servlet.WSAsyncListener$1.onComplete(WSAsyncListener.java
:69)
 at
org.apache.catalina.core.AsyncListenerWrapper.fireOnComplete(AsyncListenerWrapper.java:
35)
 at org.apache.catalina.core.AsyncContextImpl.fireOnComplete(AsyncContextImpl.java:99)
 at org.apache.coyote.AsyncStateMachine.asyncPostProcess(AsyncStateMachine.java:208)
 at org.apache.coyote.AbstractProcessor.asyncPostProcess(AbstractProcessor.java:173)
 at
org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.j
ava:662)
 at
org.apache.coyote.http11.Http11NioProtocol$Http11ConnectionHandler.process(Http11NioPro
tocol.java:222)
 at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1566)
 at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.run(NioEndpoint.java:1523)
 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
 at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
 at java.lang.Thread.run(Thread.java:745)
```

2. JAX-WS HandlerChain使用详解

JAX-WS的Handler和Servlet的Filter相似，可以对所有WebServicer进行拦截，在Handler中可以记录日志、权限控制、对请求的SOAP消息进行加密，解密等。JAX-WS提供两个Handler接口，LogicalHandler和SOAPHandler。LogicalHandler处理的是Message Payload，只能够访问消息单元中的SOAP消息体。SOAPHandler处理的是整个SOAP消息（包含SOAP header和SOAP body)，可以访问整个SOAP消息。

注册Handler的方式有下面几种：

使用HandlerResolver（客户端比较方便）

使用HandlerChain注解和配置文件

从WSDL生成

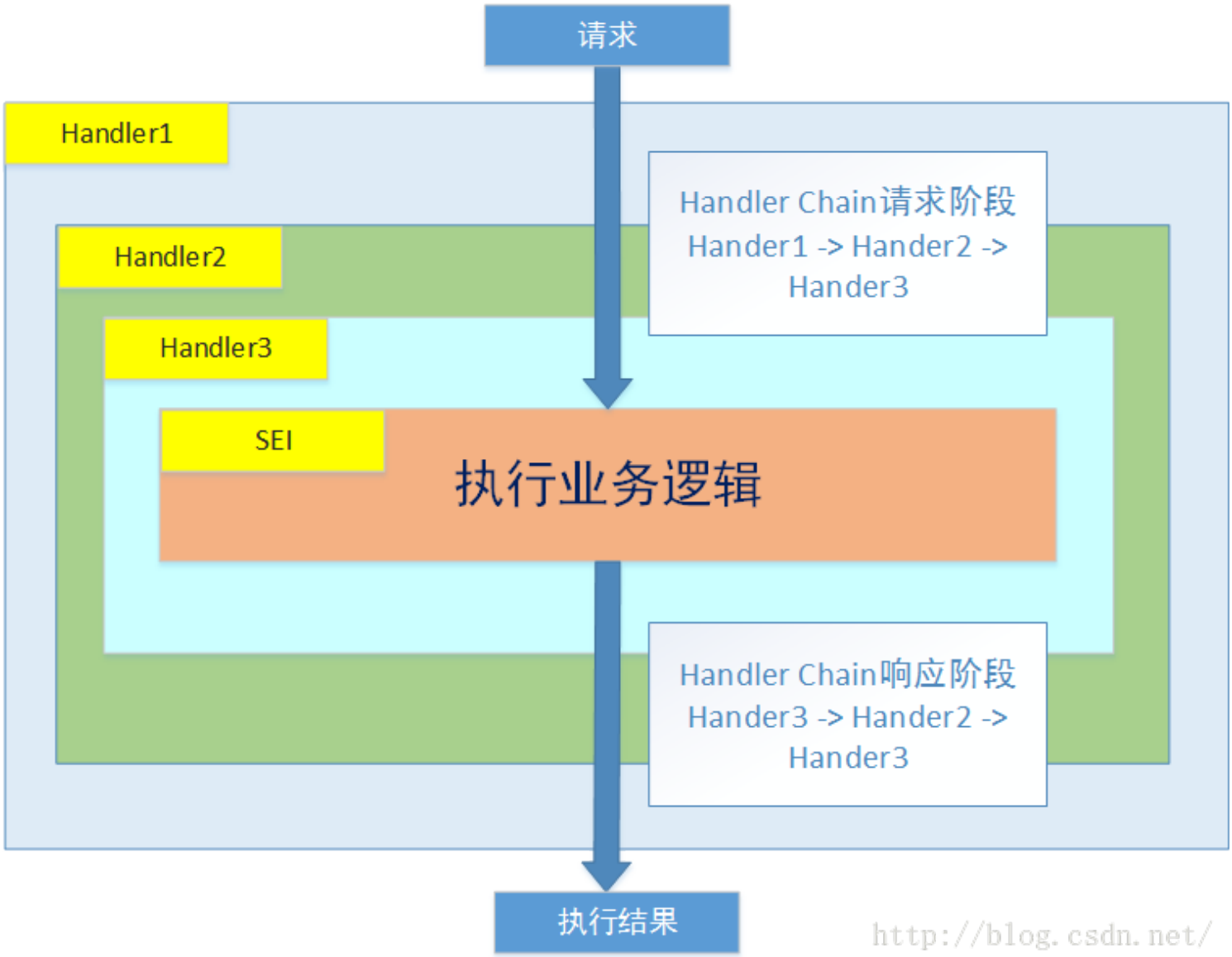
使用Custom Binding声明HandlerChain

实例代码http://download.csdn.net/detail/accountwcx/8922191

JAX-WS中WebService执行顺序如图所示

下面用SOAPHandler实现在WebService服务端记录请求内容和响应内容。

```
import java.io.IOException;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPException;
```



http://blog.csdn.net/

```
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
/**
 * SOAP
 * @author accountwcx@qq.com
 *
 */
public class LoggerHandler implements SOAPHandler<SOAPMessageContext> {
 @Override
 public void close(MessageContext context) {
 }
 @Override
 public boolean handleFault(SOAPMessageContext context) {
  return true;
 }
 @Override
 public boolean handleMessage(SOAPMessageContext context) {
  //
```

```
  Boolean output = (Boolean) context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
  System.out.println(output ? "SOAP" : "SOAP");

  SOAPMessage message = context.getMessage();

  try {
   message.writeTo(System.out);
  } catch (SOAPException e) {
   e.printStackTrace();
  } catch (IOException e) {
   e.printStackTrace();
  }

  System.out.println("");
  return true;
 }
 @Override
 public Set<QName> getHeaders() {
  return null;
 }
}
```

在classpath下建handler-chain.xml配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <javaee:handler-chain>
  <javaee:handler>
   <javaee:handler-class>com.rvho.server.ws.handler.LoggerHandler</javaee:handler-class>
  </javaee:handler>
 </javaee:handler-chain>
</javaee:handler-chains>
```

在服务实现类上添加HandlerChain配置

```
package com.rvho.server.ws.impl;
import java.util.Date;
import javax.jws.HandlerChain;
import javax.jws.WebService;
import com.rvho.server.ws.HelloWService;
@WebService(
 endpointInterface = "com.rvho.server.ws.HelloWService",
 portName = "HelloWSPort",
```

```
 serviceName = "HelloWSService",
 targetNamespace = "http://www.tmp.com/ws/hello"
)
@HandlerChain(file="handler-chain.xml") //Handler
public class HelloWServiceImpl implements HelloWService {
 public String index() {
  return "hello";
 }
 public Integer add(Integer x, Integer y) {
  return x + y;
 }
 public Date now() {
  return new Date();
 }
}
```

服务实现接口

```
package com.rvho.server.ws;
import java.util.Date;
import javax.jws.WebService;
/**
 * WebService
 */
@WebService(
 name = "HelloWS",
 targetNamespace = "http://www.tmp.com/ws/hello"
)
public interface HelloWService {
 /**
  *
  *
  * @return
  */
 String index();
 /**
  *
  *
  * @param x
  * @param y
  * @return
  */
 Integer add(Integer x, Integer y);
 /**
  *
  *
  * @return
```

```
  */
 Date now();
}
```

客户端发起index请求，服务端的记录

```
SOAP
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header/>
 <S:Body>
  <ns2:index xmlns:ns2="http://www.tmp.com/ws/hello" />
 </S:Body>
</S:Envelope>
SOAP
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header/>
 <S:Body>
  <ns2:indexResponse xmlns:ns2="http://www.tmp.com/ws/hello">
   <return>hello</return>
  </ns2:indexResponse>
 </S:Body>
</S:Envelope>
```

3. JAX-WS使用Handler实现简单的WebService权限验证

WebService如果涉及到安全保密或者使用权限的时候，WS-Security通常是最优选择。WS-Security (Web服务安全) 包含了关于如何在WebService消息上保证完整性和机密性的规约，如何将签名和加密头加入SOAP消息。不过WS-Security也有一些性能上的损耗，在信息保密要求不是很高的情况下，可以通过在SOAPHeader中添加简单的校验信息实现。

具体思路是客户端调用需要认证的服务时，在SOAPHeader中添加授权信息（如用户名、密码或者序列号等）。服务端收到请求，在SOAPHeader中校验授权信息，校验通过则执行请求，校验不通过则返回错误提示。

实例代码
http://download.csdn.net/detail/accountwcx/8922191

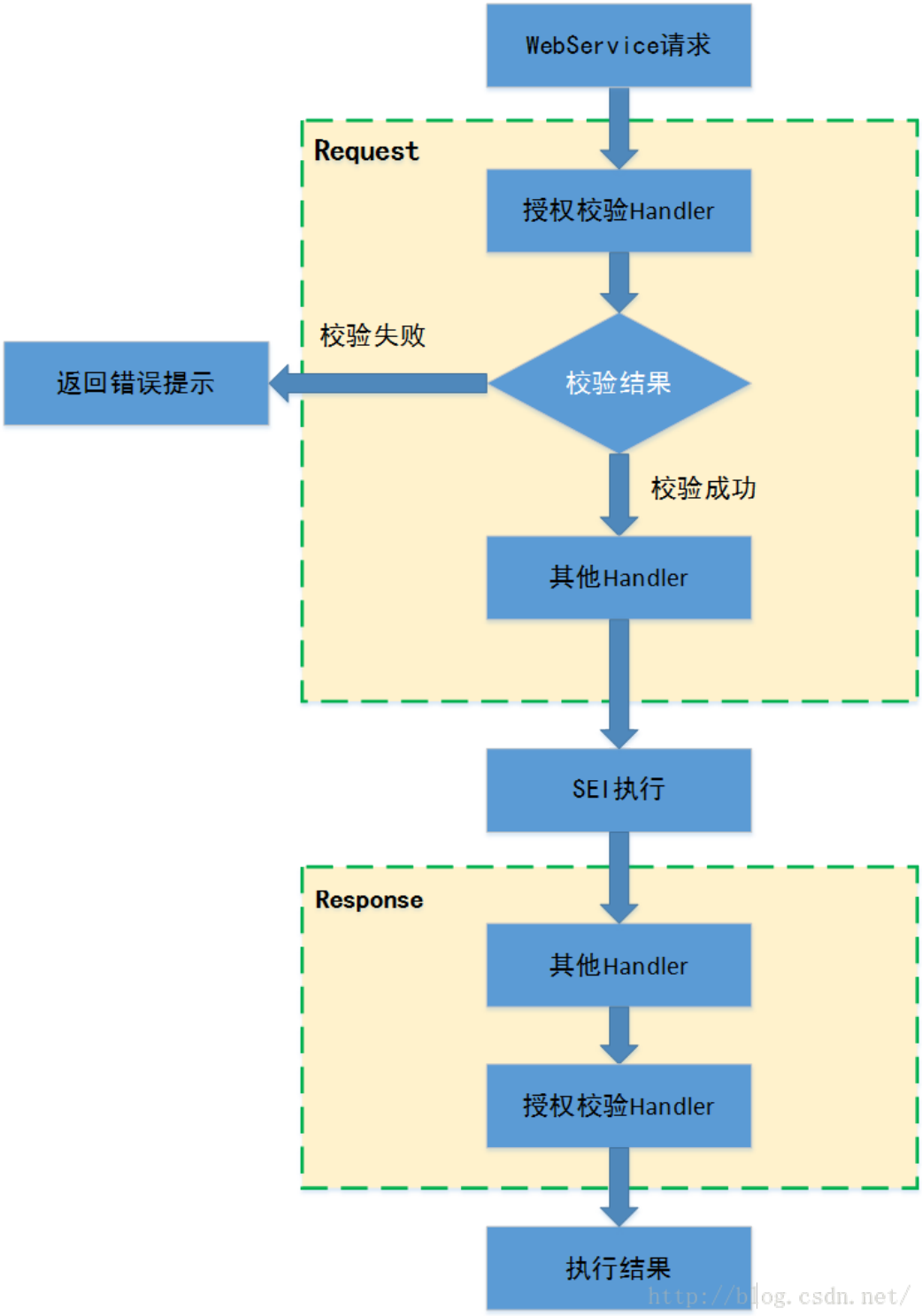
客户端发起请求在SOAPHeader中添加的授权数据格式如下

```
<auth xmlns="http://www.tmp.com/auth">
 <name>admin</name>
 <password>admin</password>
</auth>
```

服务端

服务端授权校验Handler

```
import java.util.Iterator;
```




```
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
```

```
/**
 * Handler
 * @author accountwcx@qq.com
 *
 */
public class ValidateAuthHandler implements SOAPHandler<SOAPMessageContext> {
 @Override
 public void close(MessageContext context) {
 }
 @Override
 public boolean handleFault(SOAPMessageContext context) {
  return true;
 }
 @Override
 public boolean handleMessage(SOAPMessageContext context) {
  //
  Boolean output = (Boolean) context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

  boolean result = false;

  SOAPMessage message = context.getMessage();

  //
  if(!output){
   result = validate(message);

   if(!result){
    validateFail(message);
   }
  }

  System.out.println(output ? "" : "");
  try {
   message.writeTo(System.out);
  } catch (Exception e) {
   e.printStackTrace();
  }

  System.out.println("\r\n");
  return result;
 }

 /**
  * SOAPBodySOAPFault
  * @param message
  */
 private void validateFail(SOAPMessage message) {
  try {
   SOAPEnvelope envelop = message.getSOAPPart().getEnvelope();
   envelop.getHeader().detachNode();
```

```
   envelop.addHeader();
   envelop.getBody().detachNode();
   SOAPBody body = envelop.addBody();
   SOAPFault fault = body.getFault();
   if (fault == null) {
    fault = body.addFault();
   }
   fault.setFaultString("");
   message.saveChanges();
  } catch (SOAPException e) {
   e.printStackTrace();
  }
 }

 /**
  *
  * @param message
  * @return truefalse
  */
 private boolean validate(SOAPMessage message){
  boolean result = false;

  try {
   SOAPEnvelope envelop = message.getSOAPPart().getEnvelope();
   SOAPHeader header = envelop.getHeader();

   if(header != null){
    Iterator iterator = header.getChildElements(new QName("http://www.tmp.com/auth",
"auth"));
    SOAPElement auth = null;

    if(iterator.hasNext()){
     //auth
     auth = (SOAPElement)iterator.next();

     //name
     Iterator it = auth.getChildElements(new QName("http://www.tmp.com/auth", "name"));
     SOAPElement name = null;
     if(it.hasNext()){
      name = (SOAPElement)it.next();
     }

     //password
     it = auth.getChildElements(new QName("http://www.tmp.com/auth", "password"));
     SOAPElement password = null;
     if(it.hasNext()){
      password = (SOAPElement)it.next();
     }

     //namepassword
```

```
     if(name != null && password != null && "admin".equals(name.getValue()) &&
"admin".equals(password.getValue())){
      result = true;
     }
    }
   }

  } catch (SOAPException e) {
   e.printStackTrace();
  }

  return result;
 }
 @Override
 public Set<QName> getHeaders() {
  return null;
 }
}
```

客户端

客户端添加授权Handler

```
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
/**
 *
 * @author accountwcx@qq.com
 *
 */
public class AddAuthHandler implements SOAPHandler<SOAPMessageContext> {
 @Override
 public boolean handleMessage(SOAPMessageContext context) {
  //
  Boolean output = (Boolean) context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
  SOAPMessage message = context.getMessage();
```

```
  if (output) {
   try {
    SOAPHeader header = message.getSOAPHeader();
    if (header == null) {
     header = message.getSOAPPart().getEnvelope().addHeader();
    }
    SOAPElement auth = header.addChildElement(new QName("http://www.tmp.com/auth",
"auth"));

    SOAPElement name = auth.addChildElement("name");
    name.addTextNode("admin");
    SOAPElement password = auth.addChildElement("password");
    password.addTextNode("admin");

    message.saveChanges();
   } catch (SOAPException e) {
    e.printStackTrace();
   }
  }

  System.out.println(output ? "" : "");
  try {
   message.writeTo(System.out);
  } catch (Exception e) {
   e.printStackTrace();
  }

  System.out.println("\r\n");
  return true;
 }
 @Override
 public boolean handleFault(SOAPMessageContext context) {
  return true;
 }
 @Override
 public void close(MessageContext context) {
 }
 @Override
 public Set<QName> getHeaders() {
  return null;
 }
}
```

客户端Handler配置文件handler-chain.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"
```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <javaee:handler-chain>
  <javaee:handler>
   <javaee:handler-class>com.rvho.client.handler.AddAuthHandler</javaee:handler-class>
  </javaee:handler>
 </javaee:handler-chain>
</javaee:handler-chains>
```

客户端的Service中添加Handler配置文件

```
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.9-b130926.1035
 * Generated source version: 2.2
 *
 */
@WebServiceClient(name = "HelloWSService", targetNamespace =
"http://www.tmp.com/ws/hello", wsdlLocation =
"http://localhost:8014/jaxwsserver/services/hello?wsdl")
@HandlerChain(file="handler-chain.xml")
public class HelloWSService
    extends Service
{
    private final static URL HELLOWSSERVICE_WSDL_LOCATION;
    private final static WebServiceException HELLOWSSERVICE_EXCEPTION;
    private final static QName HELLOWSSERVICE_QNAME = new
QName("http://www.tmp.com/ws/hello", "HelloWSService");
    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:8014/jaxwsserver/services/hello?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        HELLOWSSERVICE_WSDL_LOCATION = url;
        HELLOWSSERVICE_EXCEPTION = e;
    }
    public HelloWSService() {
        super(__getWsdlLocation(), HELLOWSSERVICE_QNAME);
    }
    public HelloWSService(WebServiceFeature... features) {
        super(__getWsdlLocation(), HELLOWSSERVICE_QNAME, features);
    }
    public HelloWSService(URL wsdlLocation) {
        super(wsdlLocation, HELLOWSSERVICE_QNAME);
    }
```

```
    public HelloWSService(URL wsdlLocation, WebServiceFeature... features) {
        super(wsdlLocation, HELLOWSSERVICE_QNAME, features);
    }
    public HelloWSService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }
    public HelloWSService(URL wsdlLocation, QName serviceName, WebServiceFeature...
features) {
        super(wsdlLocation, serviceName, features);
    }
    /**
     *
     * @return
     *     returns HelloWS
     */
    @WebEndpoint(name = "HelloWSPort")
    public HelloWS getHelloWSPort() {
        return super.getPort(new QName("http://www.tmp.com/ws/hello", "HelloWSPort"),
HelloWS.class);
    }
    /**
     *
     * @param features
     *     A list of {@link javax.xml.ws.WebServiceFeature} to configure on the proxy.
Supported features not in the <code>features</code> parameter will have their default
values.
     * @return
     *     returns HelloWS
     */
    @WebEndpoint(name = "HelloWSPort")
    public HelloWS getHelloWSPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://www.tmp.com/ws/hello", "HelloWSPort"),
HelloWS.class, features);
    }
    private static URL __getWsdlLocation() {
        if (HELLOWSSERVICE_EXCEPTION!= null) {
            throw HELLOWSSERVICE_EXCEPTION;
        }
        return HELLOWSSERVICE_WSDL_LOCATION;
    }
}
```

客户端发起index请求

```
URL wsdlUrl = new URL("http://localhost:7180/jaxwsserver/services/hello?wsdl");
HelloWSService helloWSS = new HelloWSService(wsdlUrl);
HelloWS helloWS = helloWSS.getHelloWSPort();
String index = helloWS.index();
```

客户端发起正确授权的请求以及服务器的响应

```
<!--  -->
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <auth xmlns="http://www.tmp.com/auth">
   <name>admin</name>
   <password>admin</password>
  </auth>
 </SOAP-ENV:Header>
 <S:Body>
  <ns2:index xmlns:ns2="http://www.tmp.com/ws/hello" />
 </S:Body>
</S:Envelope>
<!--  -->
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header/>
 <S:Body>
  <ns2:indexResponse xmlns:ns2="http://www.tmp.com/ws/hello">
   <return>hello</return>
  </ns2:indexResponse>
 </S:Body>
</S:Envelope>
```

客户端发起错误授权的请求以及服务器的响应

```
<!--  -->
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <auth xmlns="http://www.tmp.com/auth">
   <name></name>
   <password></password>
  </auth>
 </SOAP-ENV:Header>
 <S:Body>
  <ns2:index xmlns:ns2="http://www.tmp.com/ws/hello" />
```

```
 </S:Body>
</S:Envelope>
<!--  -->
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Header/>
 <S:Body>
  <S:Fault>
   <faultcode>S:Server</faultcode>
   <faultstring></faultstring>
  </S:Fault>
 </S:Body>
</S:Envelope>
```

HandlerReolver代替Handler配置文件

handler-chain配置文件对所有的请求都添加授权验证信息，有些时候不是所有的请求都需要添加授权验证，HandlerResolver提供了在编程时添加Handler的方法，可以用HandlerResolver给需要授权的接口添加Handler。

```
URL wsdlUrl = new URL("http://localhost:7180/jaxwsserver/services/hello?wsdl");
HelloWSService helloWSS = new HelloWSService(wsdlUrl);
//HandlerResolverHandler
helloWSS.setHandlerResolver(new HandlerResolver(){
 @Override
 @SuppressWarnings("rawtypes")
 public List<Handler> getHandlerChain(PortInfo portInfo) {
  List<Handler> handlerChain = new ArrayList<Handler>();
  handlerChain.add(new AddAuthHandler());
  return handlerChain;
 }

});
HelloWS helloWS = helloWSS.getHelloWSPort();
//index
String index = helloWS.index();
```

4. CXF实战(一)

Apache CXF提供了用于方便地构建和开发WebService的可靠基础架构。它允许创建高性能和可扩展的服务，可以部署在Tomcat和基于Spring的轻量级容器中，也可以部署在更高级的服务器上，例如Jboss、WebSphere或WebLogic。
CXF提供了以下功能：
WebService服务标准支持：
Java API for XML Web Services (JAX-WS)
SOAP
WebService描述语言（Web Services Description Language， WSDL）
消息传输优化机制（Message Transmission Optimization Mechanism，MTOM）
WS-Basic Profile
WS-Addressing
WS-Policy
WS-ReliableMessaging
WS-Security
前端建模：CXF允许使用不同的前端API来创建Service。如CXF允许使用简单的工厂Bean并通过JAX-WS实现来创建WebService，允许创建动态WebService客户端。
工具支持：CXF提供了在Java Bean、WebService和WSDL之间进行转换的工具，提供了对Maven和Ant集成的支持，并无缝地支持Spring集成。
RESTful支持：CXF支持Restful，并支持Java平台的JAX-RS实现。
对不同传输和绑定的支持：CXF支持不同数据类型的传输，除了支持SOAP和HTTP协议绑定外，还支持JAXB和AEGIS绑定。
对非XML绑定的支持：CXF支持非XML绑定，如JSON、CORBA、JBI和SCA等。
Code First和Xml First：CXF支持使用Code First或者Xml First的方式创建WebService。
下面用CXF创建独立发布的WebService。
服务端
在Eclipse中创建maven项目，配置文件pom.xml中引入CXF，具体配置如下

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfstandalone</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <!--  -->
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <!--  -->
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <!--  -->
```

```
        <maven.compiler.encoding>UTF-8</maven.compiler.encoding>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
    </properties>
    <dependencies>
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <!-- CXFWeb -->
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!-- End CXF -->
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <!-- sourcetargetjdk1.8 -->
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

服务接口HelloWS，该接口发布welcome方法。

```
package com.rvho.cxfstandalone.ws;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService(
        name = "HelloWS",
        targetNamespace = "http://www.tmp.com/services/hello"
    )
public interface HelloWS {
    @WebMethod
```

```
    String welcome(@WebParam(name = "name") String name);
}
```

服务接口HelloWS实现类HelloWSImpl。

```
package com.rvho.cxfstandalone.ws.impl;
import javax.jws.WebService;
import com.rvho.cxfstandalone.ws.HelloWS;
@WebService(
        endpointInterface = "com.rvho.cxfserver.ws.HelloWS",
        portName = "HelloWSPort",
        serviceName = "HelloWSService",
        targetNamespace = "http://www.tmp.com/services/hello"
    )
public class HelloWSImpl implements HelloWS {
    @Override
    public String welcome(String name) {
        return "Welcome " + name;
    }
}
```

发布服务

```
package com.rvho.cxfstandalone;
import org.apache.cxf.jaxws.JaxWsServerFactoryBean;
import com.rvho.cxfstandalone.ws.HelloWS;
import com.rvho.cxfstandalone.ws.impl.HelloWSImpl;
public class CxfServer {
    public static void main(String[] args) {
        JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();
        factory.setServiceClass(HelloWS.class);
        //
        factory.setAddress("http://localhost:8999/services/hello");
        factory.setServiceBean(new HelloWSImpl());
        factory.create();
    }
}
```

在浏览器中输入服务发布地址http://localhost:8999/services/hello?wsdl如果看到如下内容，表示服务发布成功。

客户端

用wsimport或者wsdl2java把服务端发布的服务生成java对象。

wsimport是jdk提供的wsdl转java对象工具，在jdk的bin目录下可以找到。用法是

```
"D:\Program Files\Java\jdk1.8.0_25\bin\wsimport.exe" -encoding utf-8 -d
D:\dev\wsdl\cxfserver\wsimport\compile -s D:\dev\wsdl\cxfserver\wsimport\src -p
com.rvho.cxfclient.wsdl.wsimport.hello http://localhost:8999/services/hello?wsdl
```

wsdl2java是cxf提供的wsdl转java对象工具，在cxf的bin目录下可以找到。用法是

```
"D:\dev\library\java\cxf\apache-cxf-3.1.0\bin\wsdl2java" -p
com.rvho.cxfclient.wsdl.cxf.hello -d D:\dev\wsdl\cxfserver\cxf\src
http://localhost:8999/services/hello?wsdl
```

上面命令的路径是我本机路径，有可能不一样。

客户端可以通过jaxws调用，也可以通过cxf调用。

```
//jaxws
```


```
//URL
URL wsdlUrl = new URL("http://localhost:8999/services/hello?wsdl");
HelloWSService helloWSS = new HelloWSService(wsdlUrl);
HelloWS helloWS = helloWSS.getHelloWSPort();
String welcome = helloWS.welcome("accountwcx@qq.com");
System.out.println(welcome);
/*
    cxf
    cxf
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxws</artifactId>
        <version>3.1.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-transports-http</artifactId>
        <version>3.1.1</version>
    </dependency>
*/
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(HelloWS.class);
factory.setAddress("http://localhost:8999/services/hello");
HelloWS helloWS = factory.create(HelloWS.class);
String welcome = helloWS.welcome("accountwcx@qq.com");
System.out.println(welcome);
```

```
$(function () {
   $('pre.prettyprint code').each(function () {
      var lines = $(this).text().split('\n').length;
      var $numbering = $('').addClass('pre-numbering').hide();
      $(this).addClass('has-numbering').parent().append($numbering);
      for (i = 1; i <= lines; i++) {
         $numbering.append($('').text(i));
      };
      $numbering.fadeIn(1700);
   });
});
```

5. CXF实战之在Tomcat中发布Web Service(二)

服务接口及实现类请参考WebService框架CXF实战(一)
创建Maven Web项目，在pom.xml中添加CXF和Spring Web的引用，由于CXFServlet需要Spring Web的支持。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfserver</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
    </properties>
    <dependencies>
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!-- End CXF -->
        <!-- CXFServletSpring Web -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>4.1.7.RELEASE</version>
        </dependency>
    </dependencies>
</project>
```

在WEB-INF下创建cxf-servlet.xml配置文件。

```
<?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:soap="http://cxf.apache.org/bindings/soap"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/bindings/soap
http://cxf.apache.org/schemas/configuration/soap.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <jaxws:server id="helloWSServer" serviceClass="com.rvho.cxfserver.ws.HelloWS"
address="/hello">
        <jaxws:serviceBean>
            <bean class="com.rvho.cxfserver.ws.impl.HelloWSImpl" />
        </jaxws:serviceBean>
    </jaxws:server>
</beans>
```

在WEB-INF/web.xml中添加CXFServlet配置，CXFServlet匹配/services路径下的所有请求。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>cxfserver</display-name>
    <!-- CXF Servlet -->
    <servlet>
        <servlet-name>cxfservlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxfservlet</servlet-name>
        <!-- /services -->
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <!-- End CXF Servlet -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

启动Tomcat后，在浏览器中输入http://<网站路径>/cxfserver/services即可看到如下效果，由于这里配置CXFServlet的路径是/services，如果配置其他路径，服务的请求路径也不一样，不过大体上是http://<网站路径>/cxfserver/

```
$(function () {
  $('pre.prettyprint code').each(function () {
    var lines = $(this).text().split('\n').length;
    var $numbering = $('').addClass('pre-numbering').hide();
    $(this).addClass('has-numbering').parent().append($numbering);
    for (i = 1; i <= lines; i++) {
      $numbering.append($('').text(i));
    };
    $numbering.fadeIn(1700);
  });
});
```

6. CXF实战之集成Spring(三)

CXF原生支持Spring，可以和Spring无缝集成。WebService框架CXF实战一在Tomcat中发布WebService(二)通过Spring Web实现CXFServlet。下面将Spring和CXF集成在一起，CXF发布的WebService可以调用Spring的Bean。

创建Maven Web项目，在pom.xml中添加CXF和Spring的引用，由于该Web项目中不涉及数据库，没有添加Spring JDBC、Spring ORM等数据库相关模块。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfserver</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
        <!-- Spring -->
        <spring.version>4.1.7.RELEASE</spring.version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
```

```
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- End Spring -->
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!-- End CXF -->
    </dependencies>
</project>
```

在项目中添加Dao接口及实现类，WebService调用Dao层数据，项目结构如图所示。

HelloDao接口，提供welcome方法。

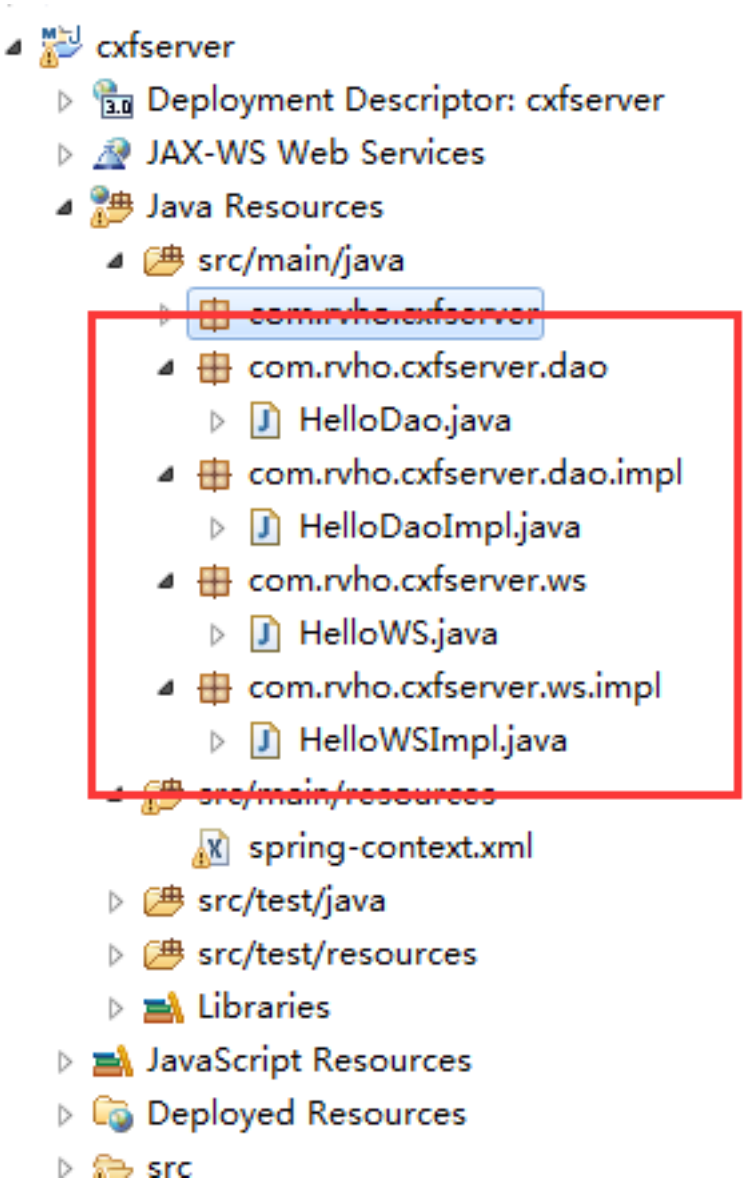
```
package com.rvho.cxfserver.dao;
public interface HelloDao {
    String welcome(String name);
}
```

HelloDao接口实现类HelloDaoImpl

```
package com.rvho.cxfserver.dao.impl;
import org.springframework.stereotype.Repository;
import com.rvho.cxfserver.dao.HelloDao;
@Repository("helloDao")
public class HelloDaoImpl implements HelloDao {
    @Override
    public String welcome(String name) {
        return "CXF" + name;
    }
}
```

HelloWS接口，由于集成了Spring，该接口既是WebService，也是Spring Service。

```
package com.rvho.cxfserver.ws;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService(
        name = "HelloWS",
        targetNamespace = "http://www.tmp.com/services/hello"
    )
public interface HelloWS {
```




```
    @WebMethod
    String welcome(@WebParam(name = "name") String name);
}
```

HelloWS接口实现类HelloWSImpl，实现类调用HelloDaoImpl的方法。

```
package com.rvho.cxfserver.ws.impl;
import javax.annotation.Resource;
import javax.jws.WebService;
import org.springframework.stereotype.Service;
import com.rvho.cxfserver.dao.HelloDao;
import com.rvho.cxfserver.ws.HelloWS;
@WebService(
        endpointInterface = "com.rvho.cxfserver.ws.HelloWS",
        portName = "HelloWSPort",
        serviceName = "HelloWSService",
        targetNamespace = "http://www.tmp.com/services/hello"
    )
@Service("helloWS")
public class HelloWSImpl implements HelloWS {
    @Resource
    private HelloDao helloDao;
```

```
    @Override
    public String welcome(String name) {
        return helloDao.welcome(name);
    }
}
```

Spring上下文配置

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
    <!-- <context:annotation-config /> -->
    <!-- SpringRepositoryServiceController -->
    <context:component-scan base-package="com.rvho" />
</beans>
```

在WEB-INF下创建cxf-servlet.xml，内容如下

```
<?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:soap="http://cxf.apache.org/bindings/soap"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/bindings/soap
http://cxf.apache.org/schemas/configuration/soap.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <!-- helloWS -->
    <jaxws:endpoint id="helloWSEndpoint" implementor="#helloWS"
address="/hello"></jaxws:endpoint>
</beans>
```

在web.xml中添加Spring上下文Listener以及CXFServlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
```

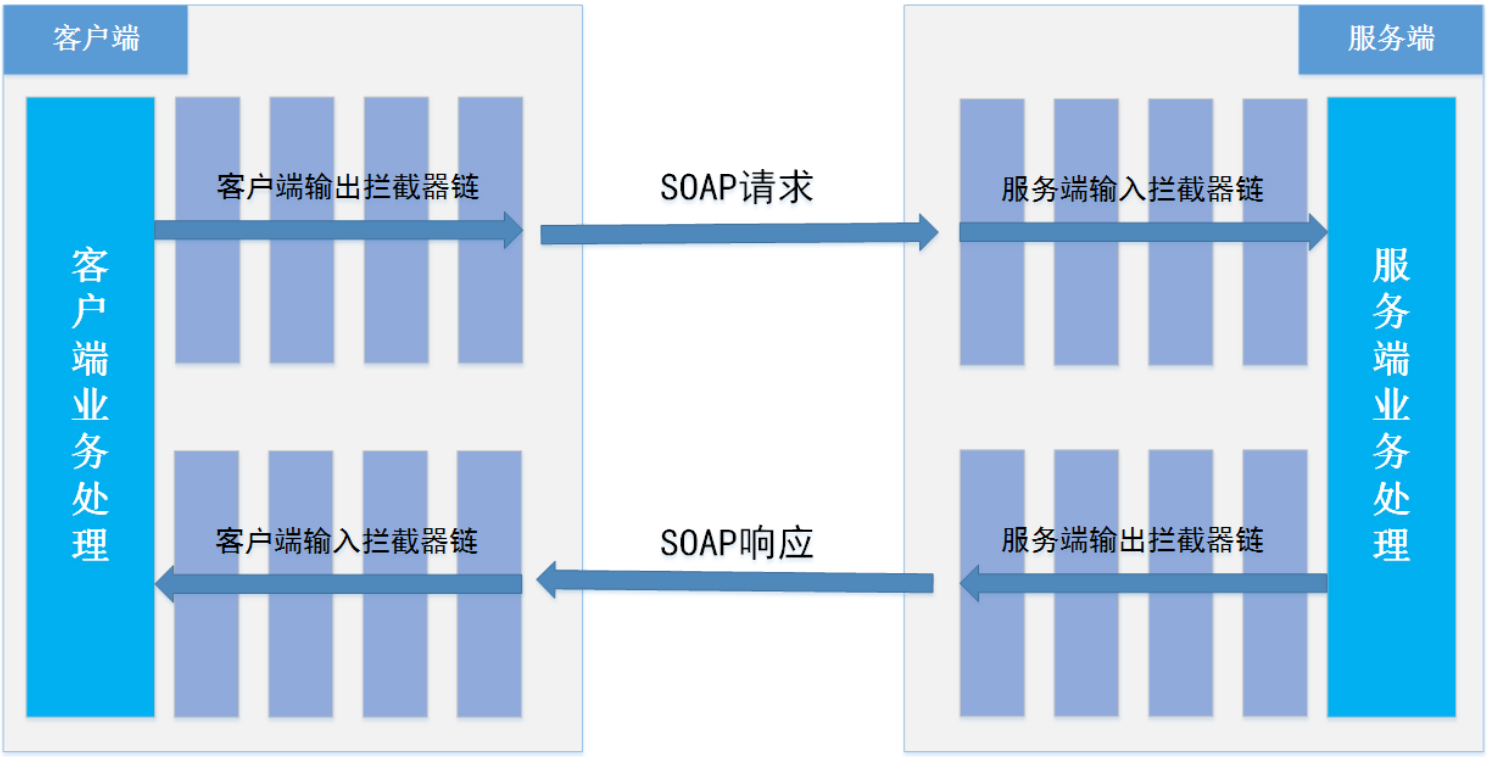
```
    <display-name>cxfserver</display-name>
    <!-- Spring -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-context.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <!-- End Spring -->
    <!-- CXF Servlet -->
    <servlet>
        <servlet-name>cxfservlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxfservlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <!-- End CXF Servlet -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

代码下载地址

```
$(function () {
    $('pre.prettyprint code').each(function () {
        var lines = $(this).text().split('\n').length;
        var $numbering = $('').addClass('pre-numbering').hide();
        $(this).addClass('has-numbering').parent().append($numbering);
        for (i = 1; i <= lines; i++) {
            $numbering.append($('').text(i));
        };
        $numbering.fadeIn(1700);
    });
});
```

7. CXF实战之拦截器Interceptor(四)

拦截器（Interceptor）是CXF功能最主要的扩展点，可以在不对核心模块进行修改的情况下，动态添加很多功能。拦截器和JAX-WS Handler、Filter的功能类似，当服务被调用时，就会创建一个拦截器链（Interceptor Chain），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，也可以在服务端添加。当客户端发起一个WebService请求时，在客户端会创建输出拦截器链，服务端接收到客户端的后，会创建输入拦截器链。当服务端返回响应消息时，响应消息会经过服务端的输出拦截链，客户端接收到服务端的响应时，会创建输入拦截器链，响应消息先经过输入拦截器链处理。拦截器在服务端和客户端的作用如图所示。



拦截器链阶段
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。
阶段名称
阶段功能描述
RECEIVE
Transport level processing(接收阶段，传输层处理)
(PRE/USER/POST)_STREAM
Stream level processing/transformations(流处理/转换阶段)
READ
This is where header reading typically occurs(SOAPHeader读取)
(PRE/USER/POST)_PROTOCOL
Protocol processing, such as JAX-WS SOAP handlers(协议处理阶段，例如JAX-WS的Handler处理)

UNMARSHAL
Unmarshalling of the request(SOAP请求解码阶段)
(PRE/USER/POST)_LOGICAL
Processing of the umarshalled request(SOAP请求解码处理阶段)
PRE_INVOKE
Pre invocation actions(调用业务处理之前进入该阶段)
INVOKE
Invocation of the service(调用业务阶段)
POST_INVOKE
Invocation of the outgoing chain if there is one(提交业务处理结果，并触发输入连接器)

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。

阶段名称
阶段功能描述
SETUP
Any set up for the following phases(设置阶段)
(PRE/USER/POST)_LOGICAL
Processing of objects about to marshalled
PREPARE_SEND
Opening of the connection(消息发送准备阶段，在该阶段创建Connection)
PRE_STREAM
流准备阶段
PRE_PROTOCOL
Misc protocol actions(协议准备阶段)
WRITE
Writing of the protocol message, such as the SOAP Envelope.(写消息阶段)
MARSHAL
Marshalling of the objects
(USER/POST)_PROTOCOL
Processing of the protocol message
(USER/POST)_STREAM
Processing of the byte level message(字节处理阶段，在该阶段把消息转为字节)
SEND
消息发送

在输出拦截器链的SEND阶段后，还会触发以_ENDING结尾阶段，这些ENDING阶段与以上阶段对应，主要用于清理或者关闭资源。ENDING阶段触发的顺序如下：

SEND_ENDING
POST_STREAM_ENDING
USER_STREAM_ENDING
POST_PROTOCOL_ENDING
USER_PROTOCOL_ENDING
MARSHAL_ENDING

WRITE_ENDING
PRE_PROTOCOL_ENDING
PRE_STREAM_ENDING
PREPARE_SEND_ENDING
POST_LOGICAL_ENDING
USER_LOGICAL_ENDING
PRE_LOGICAL_ENDING
SETUP_ENDING

CXF默认拦截器链

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。

CXF默认输入拦截器链，如果没有添加额外的拦截器，CXF输入会顺序经过以下拦截器：

拦截器名称

拦截器功能

AttachmentInInterceptor

Parse the mime headers for mime boundaries, finds the “ root ” part and resets the input stream to it, and stores the other parts in a collection of Attachments

StaxInInterceptor

Creates an XMLStreamReader from the transport InputStream on the Message

ReadHeadersInterceptor

Parses the SOAP headers and stores them on the Message

SoapActionInInterceptor

Parses “ soapaction ” header and looks up the operation if a unique operation can be found for that action.

MustUnderstandInterceptor

Checks the MustUnderstand headers, its applicability and process it, if required

SOAPHandlerInterceptor

SOAP Handler as per JAX-WS

LogicalHandlerInInterceptor

Logical Handler as per JAX-WS

CheckFaultInterceptor

Checks for fault, if present aborts interceptor chain and invokes fault handler chain

URIMappingInterceptor

Can handle HTTP GET, extracts operation info and sets the same in the Message

DocLiteralnInterceptor

Examines the first element in the SOAP body to determine the appropriate Operation (if soapAction did not find one) and calls the Databinding to read in the data.

SoapHeaderInterceptor

Perform databinding of the SOAP headers for headers that are mapped to parameters

WrapperClassInInterceptor

For wrapped doc/lit, the DocLiteralInInterceptor probably read in a single JAXB bean. This interceptor pulls the

individual parts out of that bean to construct the Object[] needed to invoke the service.

SwAInInterceptor

For Soap w/ Attachments, finds the appropriate attachments and assigns them to the correct spot in the parameter list.

HolderInInterceptor

For OUT and IN/OUT parameters, JAX-WS needs to create Holder objects. This interceptor creates the Holders and puts them in the parameter list.

ServiceInvokerInInterceptor

Actually invokes the service.

CXF默认输出拦截器链，如果没有添加额外的拦截器，CXF输入会顺序经过以下拦截器：

拦截器名称

拦截器功能

HolderOutInterceptor

For OUT and IN/OUT params, pulls the values out of the JAX-WS Holder objects (created in HolderInInterceptor) and adds them to the param list for the out message.

SwAOutInterceptor

For OUT parts that are Soap attachments, pulls them from the list and holds them for later.

WrapperClassOutInterceptor

For doc/lit wrapped, takes the remaining parts and creates a wrapper JAXB bean to represent the whole message.

SoapHeaderOutFilterInterceptor

Removes inbound marked headers

SoapActionOutInterceptor

Sets the SOAP Action

MessageSenderInterceptor

Calls back to the Destination object to have it setup the output streams, headers, etc… to prepare the outgoing transport.

SoapPreProtocolOutInterceptor

This interceptor is responsible for setting up the SOAP version and header, so that this is available to any pre-protocol interceptors that require these to be available.

AttachmentOutInterceptor

If this service uses attachments (either SwA or if MTOM is enabled), it sets up the Attachment marshallers and the mime stuff that is needed.

StaxOutInterceptor

Creates an XMLStreamWriter from the OutputStream on the Message.

SoapHandlerInterceptor

JAX-WS SOAPHandler

SoapOutInterceptor

Writes start element for soap:envelope and complete elements for other header blocks in the message. Adds start element for soap:body too.

LogicalHandlerOutInterceptor

JAX-WS Logical handler stuff
WrapperOutInterceptor
If wrapped doc/lit and not using a wrapper bean or if RPC lit, outputs the wrapper element to the stream.
BareOutInterceptor
Uses the databinding to write the params out.
SoapOutInterceptor$SoapOutEndingInterceptor
Closes the soap:body and soap:envelope
StaxOutInterceptor$StaxOutEndingInterceptor
Flushes the stax stream.
MessageSenderInt$MessageSenderEnding
Closes the exchange, lets the transport know everything is done and should be flushed to the client.

```
$(function () {
  $('pre.prettyprint code').each(function () {
    var lines = $(this).text().split('\n').length;
    var $numbering = $('').addClass('pre-numbering').hide();
    $(this).addClass('has-numbering').parent().append($numbering);
    for (i = 1; i <= lines; i++) {
      $numbering.append($('').text(i));
    };
    $numbering.fadeIn(1700);
  });
});
```

8. CXF实战之自定义拦截器(五)

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，有些拦截器也可以手动添加，如手动添加CXF提供的日志拦截器。也可以自定义拦截器，CXF中实现自定义拦截器很简单，只要继承AbstractPhaseInterceptor或者AbstractPhaseInterceptor的子类（如AbstractSoapInterceptor）即可。

自定义权限认证拦截器

权限认证拦截器处理SOAPHeader中的认证信息，客户端在发起请求时在SOAPHeader中添加认证信息，服务端在接收到请求后，校验认证信息，校验通过则继续执行，校验不通过则返回错误。

```
<!--  -->
<auth xmlns="http://www.tmp.com/auth">
    <name>admin</name>
    <password>admin</password>
</auth>
```

客户端添加授权拦截器

```
import java.util.List;
import javax.xml.namespace.QName;
import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.headers.Header;
import org.apache.cxf.helpers.DOMUtils;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
/**
 *
 *
 * @author accountwcx@qq.com
 *
 */
public class AuthAddInterceptor extends AbstractPhaseInterceptor<SoapMessage> {
    public AuthAddInterceptor(){
        //
        super(Phase.PREPARE_SEND);
    }
    @Override
    public void handleMessage(SoapMessage message) throws Fault {
        List<Header> headers = message.getHeaders();
        Document doc = DOMUtils.createDocument();
        //Element auth = doc.createElement("auth");
        Element auth = doc.createElementNS("http://www.tmp.com/auth", "auth");
        Element name = doc.createElement("name");
        name.setTextContent("admin");
        Element password = doc.createElement("password");
        password.setTextContent("admin");
```

```
        auth.appendChild(name);
        auth.appendChild(password);
        headers.add(new Header(new QName(""), auth));
    }
}
```

服务端授权验证拦截器

```
import java.util.List;
import javax.xml.namespace.QName;
import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.headers.Header;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
/**
 *
 *
 * @author accountwcx@qq.com
 *
 */
public class AuthValidateInterceptor extends AbstractPhaseInterceptor<SoapMessage> {
    public AuthValidateInterceptor(){
        super(Phase.PRE_INVOKE);
    }
    @Override
    public void handleMessage(SoapMessage message) throws Fault {
        List<Header> headers = message.getHeaders();
        if(headers == null || headers.size() < 1) {
            throw new Fault(new Exception(""));
        }
        Element auth = null;
        //
        for(Header header : headers){
            QName qname = header.getName();
            String ns = qname.getNamespaceURI();
            String tagName = qname.getLocalPart();
            if(ns != null && ns.equals("http://www.tmp.com/auth") && tagName != null &&
tagName.equals("auth")){
                auth = (Element)header.getObject();
                break;
            }
        }
        //
        if(auth == null){
            throw new Fault(new Exception(""));
        }
        NodeList nameList = auth.getElementsByTagName("name");
        NodeList pwdList = auth.getElementsByTagName("password");
        if(nameList.getLength() != 1 || pwdList.getLength() != 1){
```

```
            throw new Fault(new Exception(""));
        }
        String name = nameList.item(0).getTextContent();
        String password = pwdList.item(0).getTextContent();
        if(!"admin".equals(name) || !"admin".equals(password)){
            throw new Fault(new Exception(""));
        }
    }
}
```

服务端拦截器配置

```
<?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:soap="http://cxf.apache.org/bindings/soap"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/bindings/soap
http://cxf.apache.org/schemas/configuration/soap.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <jaxws:endpoint id="helloWSEndpoint" implementor="#helloWS" address="/hello">
        <jaxws:inInterceptors>
            <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"></bean>
            <bean
class="com.rvho.cxfserver.interceptor.AuthValidateInterceptor"></bean>
        </jaxws:inInterceptors>
        <jaxws:outInterceptors>
            <bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"></bean>
        </jaxws:outInterceptors>
    </jaxws:endpoint>
</beans>
```

客户端请求

```
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(HelloWS.class);
factory.setAddress("http://localhost:8280/cxfserver/services/hello");
factory.getInInterceptors().add(new org.apache.cxf.interceptor.LoggingInInterceptor());
//
factory.getOutInterceptors().add(new
com.rvho.cxfclient.interceptor.AuthAddInterceptor());
factory.getOutInterceptors().add(new
org.apache.cxf.interceptor.LoggingOutInterceptor());
HelloWS helloWS = factory.create(HelloWS.class);
String welcome = helloWS.welcome("accountwcx@qq.com");
```

CXF日志拦截器

CXF提供了输入日志拦截器LoggingInInterceptor和输出日志拦截器LoggingOutInterceptor，日志拦截器可以用在服务端也可以用在客户端。在测试或者调试的时候，可以用日志拦截器输出服务端、客户端请求和接收到的信息。

服务端日志内容

```
 30, 2015 10:51:37  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Inbound Message
----------------------------
ID: 1
Address: http://localhost:8280/cxfserver/services/hello
Encoding: UTF-8
Http-Method: POST
Content-Type: text/xml; charset=UTF-8
Headers: {Accept=[*/*], cache-control=[no-cache], connection=[keep-alive], Content-
Length=[212], content-type=[text/xml; charset=UTF-8], host=[localhost:8280],
pragma=[no-cache], SOAPAction=[""], user-agent=[Apache CXF 3.1.1]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:welcome
xmlns:ns2="http://www.tmp.com/services/hello"><name>accountwcx@qq.com</name></ns2:welco
me></soap:Body></soap:Envelope>
--------------------------------------
 30, 2015 10:51:37  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Outbound Message
---------------------------
ID: 1
Response-Code: 200
Encoding: UTF-8
Content-Type: text/xml
Headers: {}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:welcomeResponse
xmlns:ns2="http://www.tmp.com/services/hello"><return>
CXFaccountwcx@qq.com</return></ns2:welcomeResponse></soap:Body></soap:Envelope>
--------------------------------------
```

客户端日志内容

```
 30, 2015 10:51:37  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Outbound Message
---------------------------
ID: 1
Address: http://localhost:8280/cxfserver/services/hello
Encoding: UTF-8
Http-Method: POST
Content-Type: text/xml
Headers: {Accept=[*/*], SOAPAction=[""]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:welcome
xmlns:ns2="http://www.tmp.com/services/hello"><name>accountwcx@qq.com</name></ns2:welco
me></soap:Body></soap:Envelope>
--------------------------------------
 30, 2015 10:51:37  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Inbound Message
---------------------------
ID: 1
```

```
Response-Code: 200
Encoding: UTF-8
Content-Type: text/xml;charset=UTF-8
Headers: {content-type=[text/xml;charset=UTF-8], Date=[Thu, 30 Jul 2015 02:51:37 GMT],
Server=[Apache-Coyote/1.1], transfer-encoding=[chunked]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:welcomeResponse
xmlns:ns2="http://www.tmp.com/services/hello"><return>
CXFaccountwcx@qq.com</return></ns2:welcomeResponse></soap:Body></soap:Envelope>
-------------------------------------
CXFaccountwcx@qq.com
```

```
$(function () {
    $('pre.prettyprint code').each(function () {
        var lines = $(this).text().split('\n').length;
        var $numbering = $('').addClass('pre-numbering').hide();
        $(this).addClass('has-numbering').parent().append($numbering);
        for (i = 1; i <= lines; i++) {
            $numbering.append($('').text(i));
        };
        $numbering.fadeIn(1700);
    });
});
```

9. CXF实战之传输文件(六)

CXF的文件传输通过MTOM实现。MTOM（SOAP Message Transmission Optimization Mechanism）SOAP消息传输优化机制，可以在SOAP消息中发送二进制数据。MTOM允许将消息中包含的大型数据元素外部化，并将其作为无任何特殊编码的二进制数据随消息一起传送。相对于把二进制转为base64进行传输，MTOM具有更高的传输效率。

文件传输包装类

CXF文件传输DataHandler只有二进制数据，没有文件名、文件类型和文件大小等，需要额外的传输参数。通常自定义文件传输包装类来传输二进制和额外参数。

```
package com.rvho.cxfserver;
import javax.activation.DataHandler;
import javax.xml.bind.annotation.XmlMimeType;
import javax.xml.bind.annotation.XmlType;
/**
 * CXF CXFDataHandler
 *
 * @author accountwcx@qq.com
 */
@XmlType(name = "CxfFileWrapper")
public class CxfFileWrapper {
    //
    private String fileName;
    //
    private String fileExtension;
    //
    private DataHandler file;
    public String getFileName() {
        return fileName;
    }
    public void setFileName(String fileName) {
        this.fileName = fileName;
    }
    public String getFileExtension() {
        return fileExtension;
    }
    public void setFileExtension(String fileExtension) {
        this.fileExtension = fileExtension;
    }
    //
    @XmlMimeType("application/octet-stream")
    public DataHandler getFile() {
        return file;
    }
    public void setFile(DataHandler file) {
        this.file = file;
    }
```

```
}
```

服务端

文件传输服务接口

```
package com.rvho.cxfserver.ws;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import com.rvho.cxfserver.CxfFileWrapper;
@WebService(name = "FileWS", targetNamespace = "http://www.tmp.com/services/file")
public interface FileWS {
    /**
     *
     * @param file
     * @return truefalse
     */
    @WebMethod
    boolean upload(@WebParam(name = "file") CxfFileWrapper file);
    /**
     *
     * @return
     */
    @WebMethod
    CxfFileWrapper download();
}
```

文件传输服务实现类

```
package com.rvho.cxfserver.ws.impl;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.jws.WebService;
import org.springframework.stereotype.Service;
import com.rvho.cxfserver.CxfFileWrapper;
import com.rvho.cxfserver.ws.FileWS;
@WebService(endpointInterface = "com.rvho.cxfserver.ws.FileWS", portName =
"FileWSPort", serviceName = "FileWSService", targetNamespace =
"http://www.tmp.com/services/file")
@Service("fileWS")
public class FileWSImpl implements FileWS {
    @Override
    public boolean upload(CxfFileWrapper file){
        boolean result = true;
        OutputStream os = null;
        InputStream is = null;
```

```
    BufferedOutputStream bos = null;
    try {
        is = file.getFile().getInputStream();
        //
        File dest = new File("d:\\dev\\tmp\\upload\\" + file.getFileName());
        os = new FileOutputStream(dest);
        bos = new BufferedOutputStream(os);
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = is.read(buffer)) != -1) {
            bos.write(buffer, 0, len);
        }
        bos.flush();
    } catch (Exception e) {
        e.printStackTrace();
        result = false;
    } finally {
        if(bos != null){
            try{
                bos.close();
            }catch(Exception e){
            }
        }
        if(os != null){
            try{
                os.close();
            }catch(Exception e){
            }
        }
        if(is != null){
            try{
                is.close();
            }catch(Exception e){
            }
        }
    }
    return result;
}
@Override
public CxfFileWrapper download() {
    //
    //String filePath = "D:\\dev\\tmp\\upload\\.txt";
    String filePath = "E:\\TDDOWNLOAD\\Xme_5.0.517.exe";
    CxfFileWrapper fileWrapper = new CxfFileWrapper();
    fileWrapper.setFileName("Xme_5.0.517.exe");
    fileWrapper.setFileExtension("exe");
    DataSource source = new FileDataSource(new File(filePath));
    fileWrapper.setFile(new DataHandler(source));
    return fileWrapper;
}
```

```
}
```

服务端配置文件，需要在Endpoint配置中开启MTOM。

```
<jaxws:endpoint id="fileWSEndpoint" implementor="#fileWS" address="/file">
    <jaxws:inInterceptors>
        <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"></bean>
    </jaxws:inInterceptors>
    <jaxws:outInterceptors>
        <bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"></bean>
    </jaxws:outInterceptors>
    <jaxws:properties>
        <!-- MTOM -->
        <entry key="mtom_enabled" value="true"></entry>
    </jaxws:properties>
</jaxws:endpoint>
```

客户端

客户端下载文件

```
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(FileWS.class);
factory.setAddress("http://localhost:8280/cxfserver/services/file");
FileWS fileWS = factory.create(FileWS.class);
CxfFileWrapper fileWrapper = fileWS.download();
OutputStream os = null;
InputStream is = null;
BufferedOutputStream bos = null;
try {
    is = fileWrapper.getFile().getInputStream();
    //
    File dest = new File("d:\\dev\\tmp\\download\\" + fileWrapper.getFileName());
    os = new FileOutputStream(dest);
    bos = new BufferedOutputStream(os);
    byte[] buffer = new byte[1024];
    int len = 0;
    while ((len = is.read(buffer)) != -1) {
        bos.write(buffer, 0, len);
    }
    bos.flush();
    System.out.println("");
} catch (IOException e) {
    e.printStackTrace();
}finally{
    if(bos != null){
        try{
            bos.close();
        }catch(Exception e){
        }
    }
    if(os != null){
        try{
            os.close();
```

```
        }catch(Exception e){
        }
    }
    if(is != null){
        try{
            is.close();
        }catch(Exception e){
        }
    }
}
```

客户端上传文件

```
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(FileWS.class);
factory.setAddress("http://localhost:8280/cxfserver/services/file");
factory.getInInterceptors().add(new org.apache.cxf.interceptor.LoggingInInterceptor());
factory.getOutInterceptors().add(new
com.rvho.cxfclient.interceptor.AuthAddInterceptor());
factory.getOutInterceptors().add(new
org.apache.cxf.interceptor.LoggingOutInterceptor());
FileWS fileWS = factory.create(FileWS.class);
CxfFileWrapper fileWrapper = new CxfFileWrapper();
fileWrapper.setFileName("Xme_5.0.517.exe");
fileWrapper.setFileExtension("exe");
String filePath = "E:\\TDDOWNLOAD\\Xme_5.0.517.exe";
//String filePath = "e:\\temp\\.zip";
DataSource source = new FileDataSource(new File(filePath));
fileWrapper.setFile(new DataHandler(source));
boolean success = fileWS.upload(fileWrapper);
System.out.println(success ? "" : "");
```

注意事项

CXF不能传输大文件，否则会提示内存溢出。

```
信息: Creating Service {http://www.tmp.com/services/file}FileWSService from class com.rvho.cxfclient.wsdl.wsimport.file.FileWS
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at com.sun.xml.bind.v2.util.ByteArrayOutputStreamEx.readFrom(ByteArrayOutputStreamEx.java:75)
    at com.sun.xml.bind.v2.runtime.unmarshaller.Base64Data.get(Base64Data.java:196)
    at com.sun.xml.bind.v2.runtime.unmarshaller.Base64Data.writeTo(Base64Data.java:312)
    at com.sun.xml.bind.v2.runtime.output.UTF8XmlOutput.text(UTF8XmlOutput.java:312)
    at com.sun.xml.bind.v2.runtime.XMLSerializer.leafElement(XMLSerializer.java:356)
    at com.sun.xml.bind.v2.model.impl.RuntimeBuiltinLeafInfoImpl$PcdataImpl.writeLeafElement(RuntimeBuiltinLeafInfoImpl.java:191)
    at com.sun.xml.bind.v2.runtime.MimeTypedTransducer.writeLeafElement(MimeTypedTransducer.java:96)
    at com.sun.xml.bind.v2.runtime.reflect.TransducedAccessor$CompositeTransducedAccessorImpl.writeLeafElement(TransducedAccessor.java:254)
    at com.sun.xml.bind.v2.runtime.property.SingleElementLeafProperty.serializeBody(SingleElementLeafProperty.java:130)
    at com.sun.xml.bind.v2.runtime.ClassBeanInfoImpl.serializeBody(ClassBeanInfoImpl.java:360)
    at com.sun.xml.bind.v2.runtime.XMLSerializer.childAsXsiType(XMLSerializer.java:696)
    at com.sun.xml.bind.v2.runtime.property.SingleElementNodeProperty.serializeBody(SingleElementNodeProperty.java:158)
    at com.sun.xml.bind.v2.runtime.ClassBeanInfoImpl.serializeBody(ClassBeanInfoImpl.java:360)
    at com.sun.xml.bind.v2.runtime.XMLSerializer.childAsXsiType(XMLSerializer.java:696)
    at com.sun.xml.bind.v2.runtime.property.SingleElementNodeProperty.serializeBody(SingleElementNodeProperty.java:158)
    at com.sun.xml.bind.v2.runtime.ElementBeanInfoImpl$1.serializeBody(ElementBeanInfoImpl.java:160)
    at com.sun.xml.bind.v2.runtime.ElementBeanInfoImpl$1.serializeBody(ElementBeanInfoImpl.java:130)
    at com.sun.xml.bind.v2.runtime.ElementBeanInfoImpl.serializeBody(ElementBeanInfoImpl.java:332)
    at com.sun.xml.bind.v2.runtime.ElementBeanInfoImpl.serializeRoot(ElementBeanInfoImpl.java:339)
    at com.sun.xml.bind.v2.runtime.ElementBeanInfoImpl.serializeRoot(ElementBeanInfoImpl.java:75)
    at com.sun.xml.bind.v2.runtime.XMLSerializer.childAsRoot(XMLSerializer.java:494)
    at com.sun.xml.bind.v2.runtime.MarshallerImpl.write(MarshallerImpl.java:323)
    at com.sun.xml.bind.v2.runtime.MarshallerImpl.marshal(MarshallerImpl.java:251)
```

$(function () {

```
  $('pre.prettyprint code').each(function () {
    var lines = $(this).text().split('\n').length;
    var $numbering = $('').addClass('pre-numbering').hide();
    $(this).addClass('has-numbering').parent().append($numbering);
    for (i = 1; i <= lines; i++) {
      $numbering.append($('').text(i));
    };
    $numbering.fadeIn(1700);
  });
});
```

10. CXF实战之RESTFul服务(七)

JAX-RS概述

JAX-RS是Java提供用于开发RESTful Web服务基于注解(annotation)的API。JAX-RS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，避免了依赖第三方框架。同时JAX-RS使用POJO编程模型和基于注解的配置并集成JAXB，可以有效缩短REST应用的开发周期。

JAX-RS只定义RESTful API，具体实现由第三方提供，如Jersey、Apache CXF等。

JAX-RS包含近五十多个接口、注解和抽象类：

javax.ws.rs包含用于创建RESTful服务资源的高层次（High-level）接 口和注解。

javax.ws.rs.core包含用于创建RESTful服务资源的低层次（Low-level）接口和注解。

javax.ws.rs.ext包含用于扩展JAX-RS API支持类型的APIs。

JAX-RS常用注解：

@Path：标注资源类或方法的相对路径。

@GET、@PUT、@POST、@DELETE：标注方法的HTTP请求类型。

@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：标注方法的参数来自于HTTP请求的位置。@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。

服务端

下面用CXF发布一个图书馆RESTFul服务，实现书籍的查询、添加、删除和修改。CXF发布RESTFul服务需要引入cxf-rt-frontend-jaxrs，pom.xml配置如下。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfrestserver</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
    </properties>
    <dependencies>
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
```

```
            <version>${cxf.version}</version>
        </dependency>
        <!-- tomcatservletcxf-rt-transports-http-jetty -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!-- End CXF -->
    </dependencies>
</project>
```

书籍XML格式

```
<Book>
    <author>·</author>
    <id>10</id>
    <name></name>
    <price>3.0</price>
</Book>
```

书籍实体类

```
package com.rvho.rest.server;
import javax.xml.bind.annotation.XmlRootElement;
/**
 *
 * @author accountwcx@qq.com
 */
@XmlRootElement(name = "Book")
public class Book {
    //id
    private String id;
    //
    private String name;
    //
    private String author;
    //
    private Double price;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public Double getPrice() {
        return price;
```

```
    }
    public void setPrice(Double price) {
        this.price = price;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
}
```

图书馆服务类

```
package com.rvho.rest.server;
import java.util.HashMap;
import java.util.Map;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
/**
 *
 * @author accountwcx@qq.com
 *
 */
@Path("/library")
@Produces("text/xml")
public class LibraryService {
    private Map<String, Book> books = new HashMap<String, Book>();
    public LibraryService(){
        init();
    }
    /**
     *
     * @param id
     * @return
     */
    @GET
    @Path("/books/{id}/")
    public Book getBook(@PathParam("id") String id){
        return books.get(id);
    }
    /**
     *
     * @param book
     * @return
```

```
 */
@PUT
@Path("/books/")
public Response updateBook(Book book){
    Response r;
    if(book == null){
        r = Response.noContent().build();
        return r;
    }
    String id = book.getId();
    Book b = books.get(id);
    if(b != null){
        books.put(id, book);
        r = Response.ok(true, MediaType.TEXT_PLAIN).build();
    }else{
        r = Response.notModified().build();
    }
    return r;
}
/**
 *
 * @param book
 * @return
 */
@POST
@Path("/books/")
public Response addBook(Book book){
    Response r;
    if(book == null){
        r = Response.notModified().build();
    }else{
        books.put(book.getId(), book);
        r = Response.ok(true, MediaType.TEXT_PLAIN).build();
    }
    return r;
}
/**
 *
 * @param book
 * @return
 */
@DELETE
@Path("/books/{id}/")
public Response deleteBook(@PathParam("id") String id){
    Response r;
    Book book = books.get(id);
    if(book == null){
        r = Response.notModified("id").build();
    }else{
        books.remove(id);
```

```
            r = Response.ok(book, MediaType.APPLICATION_XML).build();
        }
        return r;
    }
    /**
     *
     */
    private void init(){
        Book book = null;
        book = new Book();
        book.setAuthor("CSDN");
        book.setId("blog");
        book.setName("");
        book.setPrice(3.0);
        books.put(book.getId(), book);
        book = new Book();
        book.setAuthor("CSDN");
        book.setId("app");
        book.setName("CSDN");
        book.setPrice(30.0);
        books.put(book.getId(), book);
        book = new Book();
        book.setAuthor("CSDN");
        book.setId("resource");
        book.setName("CSDN");
        book.setPrice(5.0);
        books.put(book.getId(), book);
        book = new Book();
        book.setAuthor("CSDN");
        book.setId("rs");
        book.setName("JAX-RS");
        book.setPrice(15.0);
        books.put(book.getId(), book);
    }
}
```

发布服务

```
JAXRSServerFactoryBean sf = new JAXRSServerFactoryBean();
sf.setResourceClasses(LibraryService.class);
sf.setResourceProvider(LibraryService.class, new SingletonResourceProvider(new
LibraryService()));
sf.setAddress("http://localhost:8280/rest/");
sf.create();
```

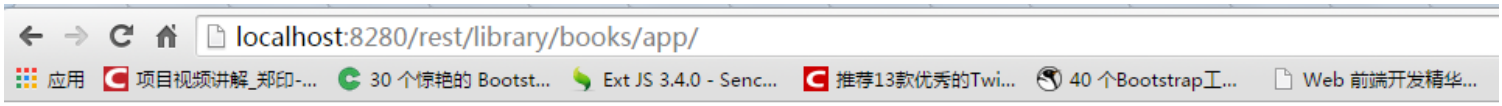
客户端

查询书籍信息，在浏览器中输入http://localhost:8280/rest/library/books/app/

下面用Fiddler模拟POST、DELETE和PUT请求。

添加书籍

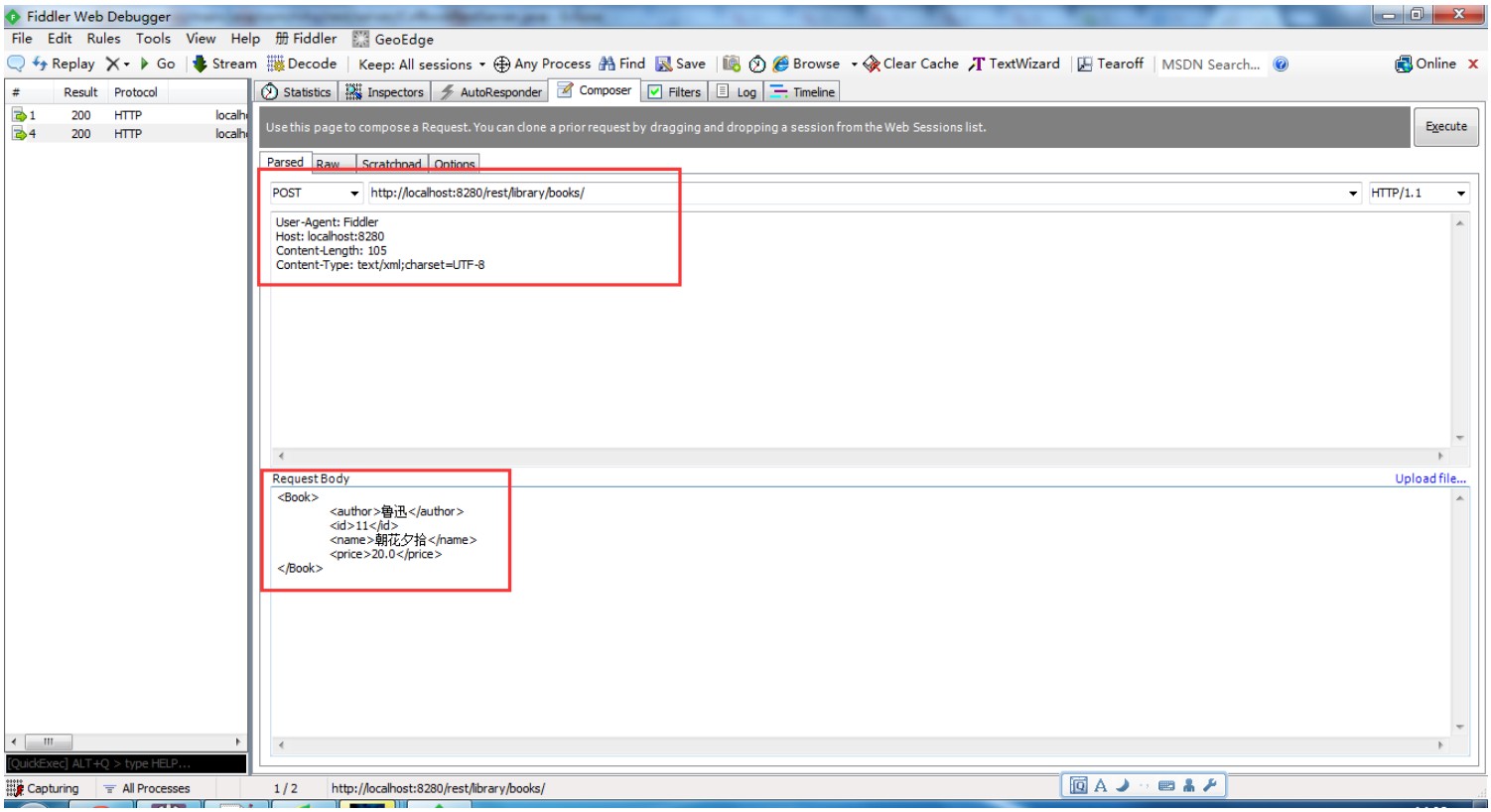
```
<Book>
    <author></author>
    <id>11</id>
```



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Customer>
    <author>CSDN</author>
    <id>app</id>
    <name>如何下载CSDN移动客户端</name>
    <price>30.0</price>
  </Customer>
```

```
    <name></name>
    <price>20.0</price>
</Book>
```



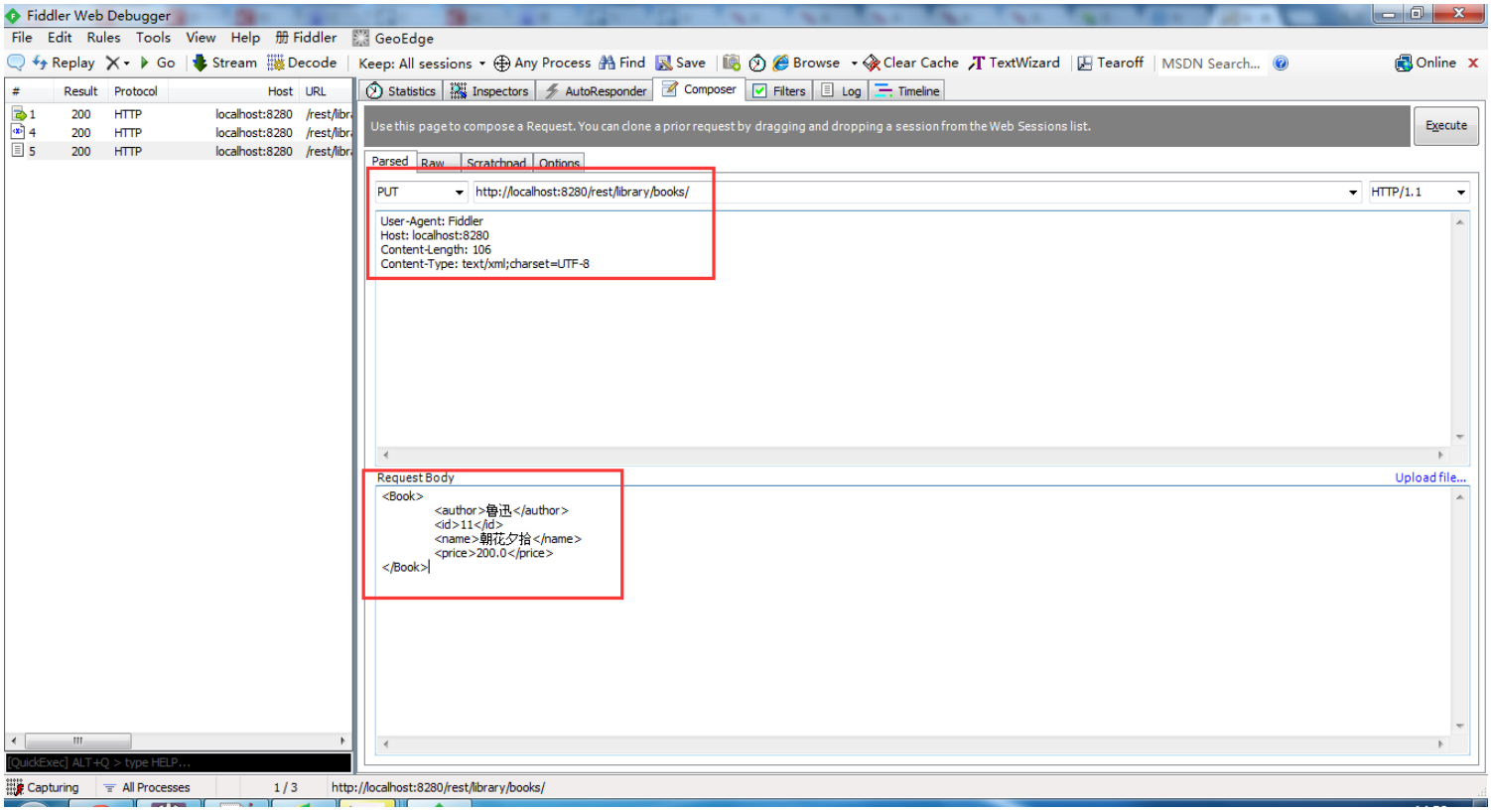
添加返回结果

```
HTTP/1.1 200 OK
Date: Fri, 31 Jul 2015 06:50:30 GMT
Content-Type: text/plain
Date: Fri, 31 Jul 2015 06:50:30 GMT
Content-Length: 4
Server: Jetty(9.2.10.v20150310)
true
```

修改书籍

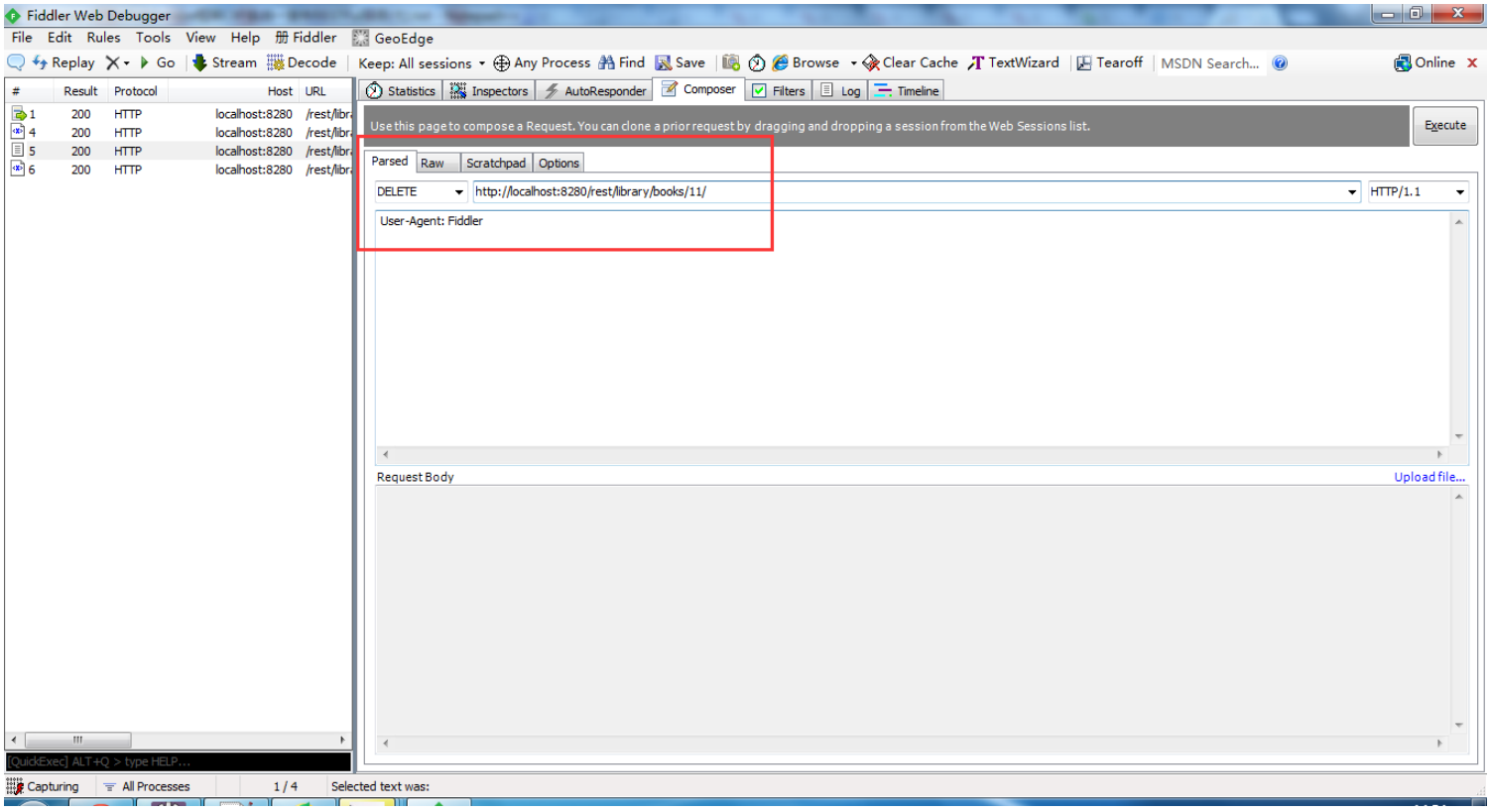
```
<Book>
    <author></author>
    <id>11</id>
    <name></name>
    <price>200.0</price>
</Book>
```

返回结果




```
HTTP/1.1 200 OK
Date: Fri, 31 Jul 2015 06:52:24 GMT
Content-Type: text/plain
Date: Fri, 31 Jul 2015 06:52:24 GMT
Content-Length: 4
Server: Jetty(9.2.10.v20150310)
true
```

删除书籍



返回结果

```
HTTP/1.1 200 OK
Date: Fri, 31 Jul 2015 06:54:36 GMT
Content-Type: application/xml
Date: Fri, 31 Jul 2015 06:54:36 GMT
Content-Length: 147
Server: Jetty(9.2.10.v20150310)
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><Book><author>
</author><id>11</id><name></name><price>200.0</price></Book>
```

Tomcat中发布RESTful

CXF在Tomcat中发布服务，需要Spring-Web支持，pom.xml配置如下。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfrestserver</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
        <!-- Spring -->
        <spring.version>4.1.7.RELEASE</spring.version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- End Spring -->
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!-- End CXF -->
    </dependencies>
</project>
```

在web.xml中添加CXFServlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>cxfrestserver</display-name>
    <!-- Spring -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-context.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

```
    </listener>
    <!-- End Spring -->
    <!-- CXF Servlet -->
    <servlet>
        <servlet-name>cxfservlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxfservlet</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
    <!-- End CXF Servlet -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

在WEB-INF文件夹下创建cxf-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxrs="http://cxf.apache.org/jaxrs"
    xsi:schemaLocation="
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
        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">
    <bean id="libraryServiceBean" class="com.rvho.rest.server.LibraryService"></bean>
    <jaxrs:server id="libraryServer" address="/">
        <jaxrs:serviceBeans>
            <ref bean="libraryServiceBean"/>
        </jaxrs:serviceBeans>
    </jaxrs:server>
</beans>
```

Spring上下文配置

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
    http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
    <!-- <context:annotation-config /> -->
    <!-- SpringRepositoryServiceController -->
    <context:component-scan base-package="com.rvho" />
</beans>
```

启动Tomcat，在浏览器中输入http://<地址>/cxfrestserver/rest即可看到服务


```
$(function () {
  $('pre.prettyprint code').each(function () {
    var lines = $(this).text().split('\n').length;
    var $numbering = $('').addClass('pre-numbering').hide();
    $(this).addClass('has-numbering').parent().append($numbering);
    for (i = 1; i <= lines; i++) {
      $numbering.append($('').text(i));
    };
    $numbering.fadeIn(1700);
  });
});
```

11. CXF实战之WS-Security(八)

Web-Security概述

WS-Security(Web服务安全)是一种提供在Web Service上应用安全的方法的网络传输协议，协议包含了关于如何在Web Service消息上保证完整性和机密性的规约。WS-Security描述了如何将签名和加密头加入SOAP消息。除此以外，还描述了如何在消息中加入安全令牌，包括二进制安全令牌，如X.509认证证书和Kerberos门票(ticket)。WS-Security将安全特性放入SOAP消息头中在应用层处理，这样协议保证了端到端的安全。

CXF中使用Web-Security

在CXF中实现服务端或者客户端的WS-Security，需要设置WSS4J拦截器。WS-Security规范支持多种令牌方式，UserNameToken头是其中一种方式。UserNameToken头是一种把用户米和密码或者密码摘要传到另一个端点的标准方式。

服务端

在配置文件pom.xml中添加WS-Security引用。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rvho</groupId>
    <artifactId>cxfservers</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <!-- CXF -->
        <cxf.version>3.1.1</cxf.version>
        <!-- Spring -->
        <spring.version>4.1.7.RELEASE</spring.version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- End Spring -->
        <!-- CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-ws-security</artifactId>
            <version>${cxf.version}</version>
        </dependency>
        <!--tomcatjetty-->
        <!--
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>${cxf.version}</version> </dependency>
        -->
        <!-- End CXF -->
    </dependencies>
</project>
```

服务端回调函数ServerPasswordCallbackHandler，校验客户端请求是否合法，合法就放行，否则拒绝

执行任何操作。

```
package com.rvho.cxfserver.callback;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.wss4j.common.ext.WSPasswordCallback;
public class ServerPasswordCallbackHandler implements CallbackHandler {
    @Override
    public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
        if (pc.getIdentifier().equals("admin")) {
            //
            //
            //org.apache.ws.security.WSSecurityException
            pc.setPassword("123");
        }
    }
}
```

服务端通过输入拦截器WSS4JInInterceptor实现WS-Security的校验，如果服务端集成了Spring，WSS4JInInterceptor配置如下。

```
<?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:soap="http://cxf.apache.org/bindings/soap"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/bindings/soap
http://cxf.apache.org/schemas/configuration/soap.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <!--  -->
    <bean id="serverPasswordCallback"
class="com.rvho.cxfserver.callback.ServerPasswordCallbackHandler"></bean>
    <jaxws:endpoint id="helloWSEndpoint" implementor="#helloWS" address="/hello">
        <jaxws:inInterceptors>
            <!-- WS-Security -->
            <bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
                <!--  -->
                <constructor-arg>
                    <map>
                        <entry key="action" value="UsernameToken"/>
                        <!-- PasswordText -->
                        <entry key="passwordType" value="PasswordText"/>
                        <entry key="passwordCallbackRef">
                            <!--  -->
                            <ref bean="serverPasswordCallback"/>
```

```
                    </entry>
                </map>
            </constructor-arg>
        </bean>
        <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"></bean>
    </jaxws:inInterceptors>
    <jaxws:outInterceptors>
        <bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"></bean>
    </jaxws:outInterceptors>
</jaxws:endpoint>
</beans>
```

如果服务端通过代码方式发布服务，可以用API添加。

```
JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();
factory.setServiceClass(HelloWS.class);
factory.setAddress("http://localhost:8280/cxfservers/services/hello");
factory.setServiceBean(new HelloWSImpl());
//WS-Security
Map<String, Object> inProps = new HashMap<String, Object>();
inProps.put(WSHandlerConstants.ACTION, WSHandlerConstants.USERNAME_TOKEN);
inProps.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_TEXT);
inProps.put(WSHandlerConstants.PW_CALLBACK_CLASS,
ServerPasswordCallbackHandler.class.getName());
factory.getInInterceptors().add(new WSS4JInInterceptor(inProps));
factory.getInInterceptors().add(new LoggingInInterceptor());
factory.create();
```

客户端

客户端回调函数ServerPasswordCallbackHandler，回调函数在发请求时添加密码。

```
package com.rvho.cxfclient.callback;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.wss4j.common.ext.WSPasswordCallback;
public class ClientPasswordCallbackHandler implements CallbackHandler {
    @Override
    public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
        pc.setPassword("123");
    }
}
```

客户端请求

```
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(HelloWS.class);
factory.setAddress("http://localhost:8280/cxfservers/services/hello");
factory.getInInterceptors().add(new org.apache.cxf.interceptor.LoggingInInterceptor());
//WS-Security
Map<String, Object> outProps = new HashMap<String, Object>();
outProps.put(WSHandlerConstants.ACTION, WSHandlerConstants.USERNAME_TOKEN);
```

```
//
outProps.put(WSHandlerConstants.USER, "admin");
outProps.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_TEXT);
outProps.put(WSHandlerConstants.PW_CALLBACK_CLASS,
ClientPasswordCallbackHandler.class.getName());
factory.getOutInterceptors().add(new WSS4JOutInterceptor(outProps));
factory.getOutInterceptors().add(new
com.rvho.cxfclient.interceptor.AuthAddInterceptor());
factory.getOutInterceptors().add(new
org.apache.cxf.interceptor.LoggingOutInterceptor());
HelloWS helloWS = factory.create(HelloWS.class);
String welcome = helloWS.welcome("accountwcx@qq.com");
System.out.println(welcome);
```

客户端和服务端数据格式

```
 03, 2015 2:31:51  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Outbound Message
---------------------------
ID: 1
Address: http://localhost:8280/cxfservers/services/hello
Encoding: UTF-8
Http-Method: POST
Content-Type: text/xml
Headers: {Accept=[*/*], SOAPAction=[""]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Header><wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" soap:mustUnderstand="1"><wsse:UsernameToken wsu:Id="UsernameToken-
581191d1-7dcb-479b-a739-
0ebb063d740f"><wsse:Username>admin</wsse:Username><wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">123</wsse:Password></wsse:UsernameToken></wsse:Security><auth
xmlns="http://www.tmp.com/auth"><name>admin</name><password>admin</password></auth></so
ap:Header><soap:Body><ns2:welcome
xmlns:ns2="http://www.tmp.com/services/hello"><name>accountwcx@qq.com</name></ns2:welco
me></soap:Body></soap:Envelope>
--------------------------------------
 03, 2015 2:31:51  org.apache.cxf.services.HelloWSService.HelloWSPort.HelloWS
: Inbound Message
----------------------------
ID: 1
Address: http://localhost:8280/cxfservers/services/hello
Encoding: UTF-8
Http-Method: POST
Content-Type: text/xml; charset=UTF-8
Headers: {Accept=[*/*], Cache-Control=[no-cache], connection=[keep-alive], Content-
Length=[847], content-type=[text/xml; charset=UTF-8], Host=[localhost:8280],
Pragma=[no-cache], SOAPAction=[""], User-Agent=[Apache CXF 3.1.1]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Header><wsse:Security
```

```
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" soap:mustUnderstand="1"><wsse:UsernameToken wsu:Id="UsernameToken-
581191d1-7dcb-479b-a739-
0ebb063d740f"><wsse:Username>admin</wsse:Username><wsse:Password
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">123</wsse:Password></wsse:UsernameToken></wsse:Security><auth
xmlns="http://www.tmp.com/auth"><name>admin</name><password>admin</password></auth></so
ap:Header><soap:Body><ns2:welcome
xmlns:ns2="http://www.tmp.com/services/hello"><name>accountwcx@qq.com</name></ns2:welco
me></soap:Body></soap:Envelope>
--------------------------------------
```

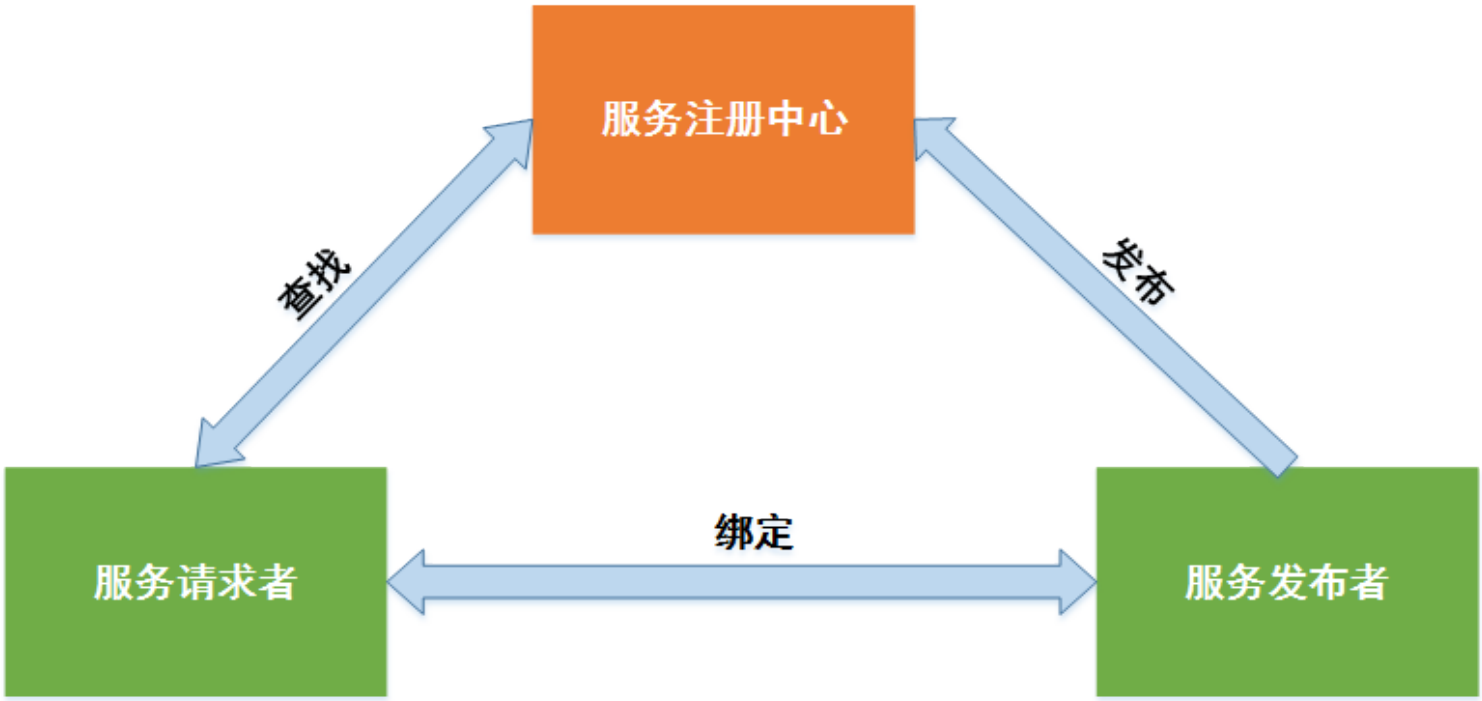
```
$(function () {
    $('pre.prettyprint code').each(function () {
        var lines = $(this).text().split('\n').length;
        var $numbering = $('').addClass('pre-numbering').hide();
        $(this).addClass('has-numbering').parent().append($numbering);
        for (i = 1; i <= lines; i++) {
            $numbering.append($('').text(i));
        };
        $numbering.fadeIn(1700);
    });
});
```

12. Web Service相关规范

Web Service概述

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，用于开发分布式的互操作的应用程序。

在Web Service的体系架构中有三个角色：服务提供者(Service Provider)，也叫服务生产者；服务请求者(Service Requester)，也叫服务消费者；服务注册中心(Service Register)，也叫服务代理，服务提供者在这里发布服务，服务请求者在这里查找服务，获取服务的绑定信息。三个角色之间的关系如图所示。



角色间主要有三个操作：

发布(Publish)，服务提供者把服务按照规范格式发布到服务注册中心；

查找(Find)，服务请求者根据服务注册中心提供的规范接口发出查找请求，获取绑定服务所需的相关信息。

绑定(Bind)，服务请求者根据服务绑定信息对自己的系统进行配置，从而可以调用服务提供者提供的服务。

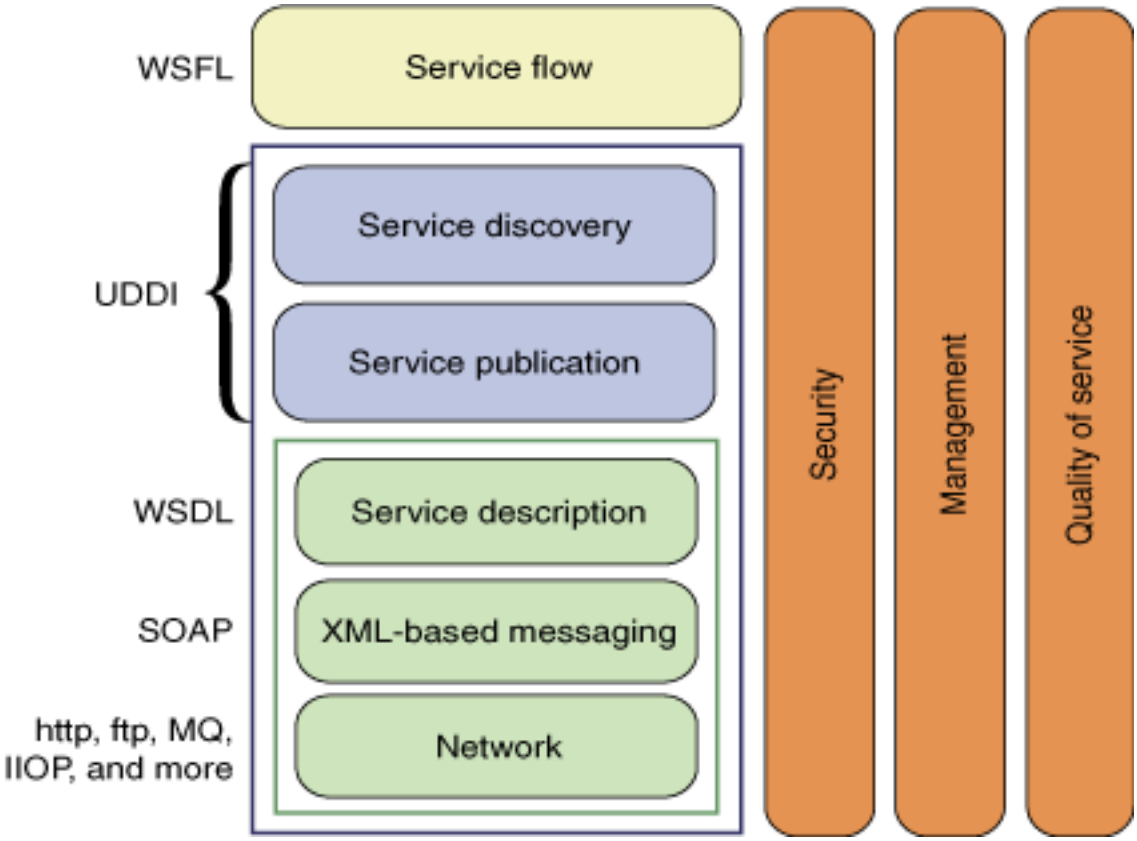
Web Service规范了体系中的各种关键技术，包括服务描述、发布、发现以及消息传输等，参考模型如图所示。

规范

Web Service的规范包括基本规范(WSDL、SOAP、UDDI)以及扩展规范WS-*(WS-Security、WS-Policy、WS-Addressing、WS-Trust等)。

WSDL

WSDL(Web Service Description Language/Web服务描述语言)是用XML文档来描述Web服务的标准，通过WSDL可描述Web服务的三个基本属性：



服务能做什么(服务所提供的操作/方法)
如何访问服务(和服务交互的数据格式以及协议)
服务位于何处(与协议相关的服务地址，如URL)
WSDL文档以端口集合的形式来描述Web服务，WSDL服务描述包含对一组操作和消息的一个抽象定义、绑定到这些操作和消息的一个具体协议、以及一个网络端点规范。WSDL文档被分为服务接口(Service Interface/抽象)定义和服务实现(Service Implementation/具体实现)定义，WSDL基本结构如图所示。
WSDL文档中主要元素：
抽象定义
Types：定义Web服务使用的所有数据类型集合，可被元素的各消息部件所引用。
Messages：通信消息数据结构的抽象类型化定义。使用Types所定义的类型来定义整个消息的数据结构，包括函数参数或文档描述。
PortTypes：引用消息部分中消息定义来描述函数签名(操作名、输入参数、输出参数)。
具体定义
Operation：对服务中操作的抽象描述。一个Operation描述了一个访问入口的请求/响应消息对。
Bindings：portType部分的操作在此绑定实现，包含了如何将抽象接口的元素转变为具体表示的细节。
Port：定义为协议/数据格式绑定与具体Web访问地址组合的单个服务访问点。
Services：确定每个绑定的端口地址。
portType、message和type描述了Web服务是什么，binding描述了如何使用Web服务，port和service描述了Web服务的位置。
SOAP
SOAP(Simple Object Access Protocol/简单文件传输协议)是一个轻量的、简单的、基于XML的协议。

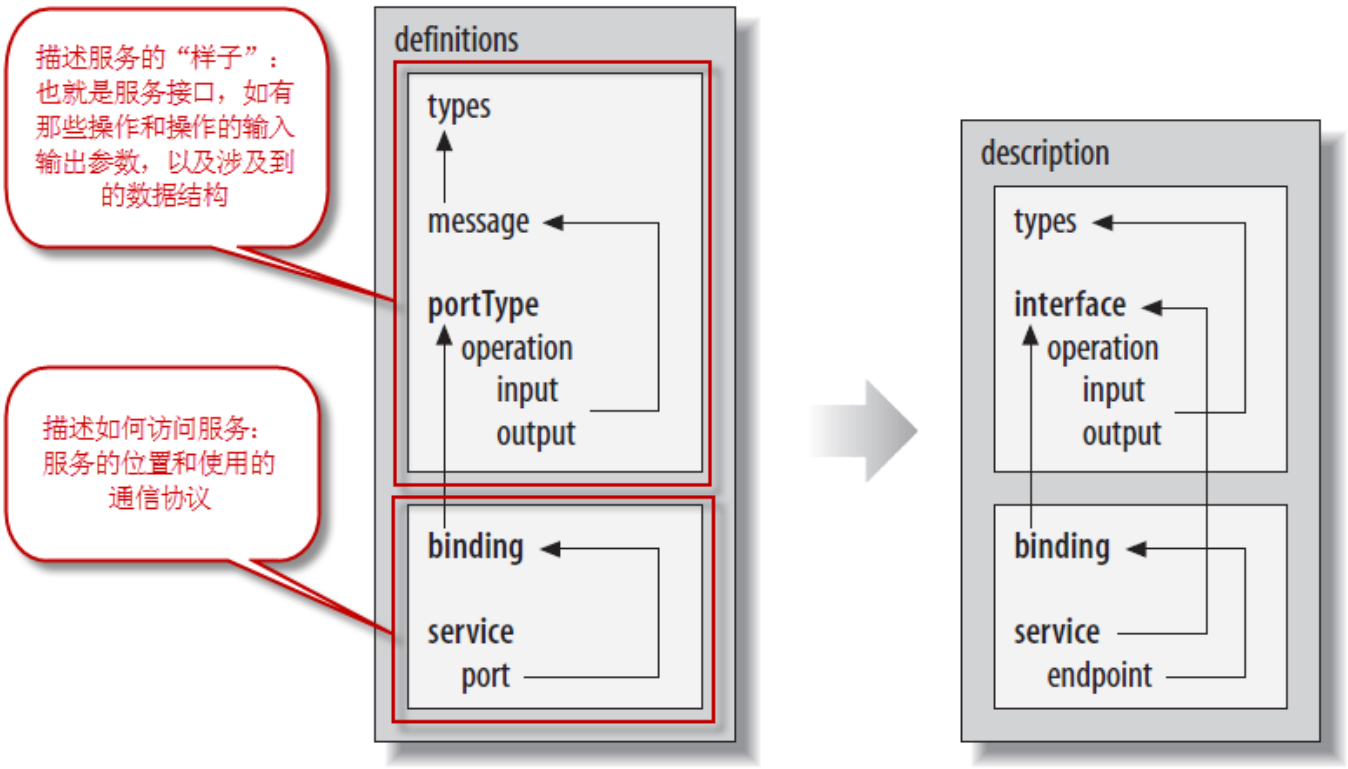


FIGURE 16-1. General structure of WSDL 1.1 and 2.0 files

SOAP包括四个部分：
SOAP封装(envelop)，封装定义了一个描述消息中的内容是什么，是谁发送的，谁应当接受并处理它以及如何处理它们的框架。
SOAP编码规则（encoding rules），用于表示应用程序需要使用的数据类型的实例。
SOAP RPC表示(RPC representation)，表示远程过程调用和应答的协定。
SOAP绑定（binding），使用底层协议交换信息。
SOAP的基本结构包含以下元素：
Envelope（必须）标识文档为SOAP消息，是SOAP的根元素。
Header（可选）包含头部信息，如果SOAP消息有Header，则Header必须是Envelope的第一个子元素。
Body（必须）包含所有的调用和响应信息。
Fault（可选）提供错误消息。
SOAP格式

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <auth xmlns="http://www.tmp.com/auth">
            <name>admin</name>
            <password>admin</password>
        </auth>
    </soap:Header>
    <soap:Body>
        <ns2:welcome xmlns:ns2="http://www.tmp.com/services/hello">
            <name>accountwcx@qq.com</name>
```

```
        </ns2:welcome>
    </soap:Body>
</soap:Envelope>
```

UDDI

UDDI(Universal Description, Discovery and Integration/通用描述、发现和集成)主要提供基于Web服务的注册和发现机制，为Web服务提供三个重要的技术支持：

标准、透明、专门描述Web服务的机制。

调用Web服务的机制。

可以访问的Web服务注册中心。

WS-Policy

WS-Policy(Web Services Policy Framework/Web服务策略框架)提供了一种灵活、可扩展的语法，用XML表示Web Services系统中实体的能力、要求和一般特性。WS-Policy将策略定义为一组策略替换选项，其中每个策略替换选项又是一组策略断言。WS-Policy将策略断言分为两类，一类指传统的要求和功能，这些要求和功能最终将出现在网络中(如身份验证方案、传输协议选择等)。另一类策略断言并不直接表现在网络中，但却对正确地选择和使用服务至关重要。

WS-Addressing

WS-Addressing(Web服务寻址)为Web Services提供一种与传输层无关的、传送寻址信息的机制。WS-Addressing主要由两部分组成：传送Web Services端点的引用数据结构，以及一套能够在特定的消息上关联寻址信息的消息寻址属性。

WS-ReliableMessaging

WS-ReliableMessaging定义了一种协议和一套机制，使Web Services开发人员能够确保在两个端点之间可靠的传递消息，该规范具有各种传递保证和健壮性特征。

WS-Security

WS-Security以现有的密码学以及XML加密和签名行业标准为基础，为Web Services应用程序提供了一组全面的安全特性，可以通过WS-Policy和WS-SecurityPolicy来指定特定应用程序可以使用哪些特性，从而允许服务客户机自行配置以访问服务。

WS-Security覆盖以下三方面内容：

描述通过消息完整性、消息机密性和SOAP消息传递的增强质量。

提供关联安全性令牌和消息的通用机制。

描述如何对二进制安全性令牌编码。

WS-Security提供三种主要机制：

安全性令牌传播

消息完整性

消息机密性

WS-SecurityPolicy

WS-SecurityPolicy以WS-Policy为基础，采用声明方式来定义可用于指定安全相关策略的XML元素，声明允许Web Services明确地指定其策略。

JAX-WS

JAX-WS(Java API for XML-Based Web Services)是一组XML Web Services的Java API，用于简化使用Java发布Web Services和生成Web Services客户端。JAX-WS提供了完整的Web Services堆栈，可减少开发和部署Web Services的工作量。JAX-WS允许开发者选择RPC-oriented或者message-oriented来实现自己的Web Services。JAX-WS支持WS-I Basic Profile 1.1，还包括Java Architecture for XML

Binding(JAXB)和SOAP with Attachments API for Java(SAAJ)。
JAX-WS 2.0的主要特性：
新的API和新的编程模型。新的API主要包含在javax.xml.ws包中，包括服务端和客户端的一些核心类。
新的编程模型包括增强的Handler Framework、异步调用和Provider/Dispatch动态编程模型。
使用注解来描述Web Services，不再依赖Web Services描述文件。
通过JAXB 2.0完成XML数据和Java对象的绑定。
MTOM/XOP(SOAP Message Transmission Optimization Mechanism/XML Binary Optimzied)和swaRef(SOAP Attachment Refenreces)解决SOAP消息传输二进制附件的问题，同时优化消息传输。
对SOAP 1.2提供支持。
JAX-WS开发Web Services有契约优先方法与代码优先方法，契约优先是指从WSDL生成Java类来实现Web服务。代码优先是指从Java类着手，使用注解来生成WSDL文件。

```
$(function () {
  $('pre.prettyprint code').each(function () {
    var lines = $(this).text().split('\n').length;
    var $numbering = $('').addClass('pre-numbering').hide();
    $(this).addClass('has-numbering').parent().append($numbering);
    for (i = 1; i <= lines; i++) {
      $numbering.append($('').text(i));
    };
    $numbering.fadeIn(1700);
  });
});
```