WebService

1. 复习准备

1.1. Schema 约束

几个重要知识：

1. **namespace**

   相当于 Schema 文档的 id,它的值必须是唯一的

2. **targetNamespace 属性**

   用来指定 schema 文档的 namespace 值

3. **xmlns 属性**

   引入某个命名空间

4. **schemaLocation 属性**

   指定引入的命名空间的 schema 文件的位置

1. 在Schema规范中，xml文件中的所有标签和属性都需要有schema文件来定义(约束)

2. 如何引入约束?

   xmlns属性来指定：它的值为一个schema文件的namespace值

3. 每个schmema文件都必须有一个唯一标识，平常一般取名为id，但在schema中以namespace(命名空间)来表达

也就是每个Schema文件都有一个唯一的namespace值

4. schema文件的namespace值如何指定?

targetNamespace属性来指定: 它的值是一个url格式的文本(路径不一定真实存在, 但必须唯一)

5. 如果引入的schema约束不是w3c组织定义, 那么在引入后还需要指定schema文件的位置

6. 如何来指定schema文件的位置?

schemaLocation属性来指定: 它的值由两部分组成: namcespace+path

7. 如果引入了N个约束, 那反必须给n-1个取别名,

在使用到某个取了别名的 schema 文档的标签或属性时,必须通过别名来引导

1.2. HTTP 协议

几个重要知识:

1. 请求的组成：

2. 响应的组成

3. 请求的过程

2. 提出 2 个问题

问题一：

1. 它们公司服务器的数据库中都保存了天气预报数据吗?
2. 如果没有, 那数据都存在哪了呢?
3. 这些网站是如何得到这些数据的呢?

问题二:

上证指数	2155.81	-42.71	-1.94%
深证成指	8439.47	-174.24	-2.02%
中小板综	6014.63	-69.14	-1.14%
沪深300	2384.44	-44.59	-1.84%
沪市B股	247.05	-8.15	-3.19%
深市B股	821.20	-2.72	-0.33%
沪市基金	3973.09	-46.70	-1.16%
深市基金	4938.12	-49.99	-1.00%

各个门户网站显示的股票行情信息数据又是怎么来的呢?

3. 关于 Web Service 的几个问题

3.1. Web service 是什么？

1. 基于 Web 的服务：服务器端整出一些资源让客户端应用访问（获取数据）

2. 一个跨语言、跨平台的规范（抽象）

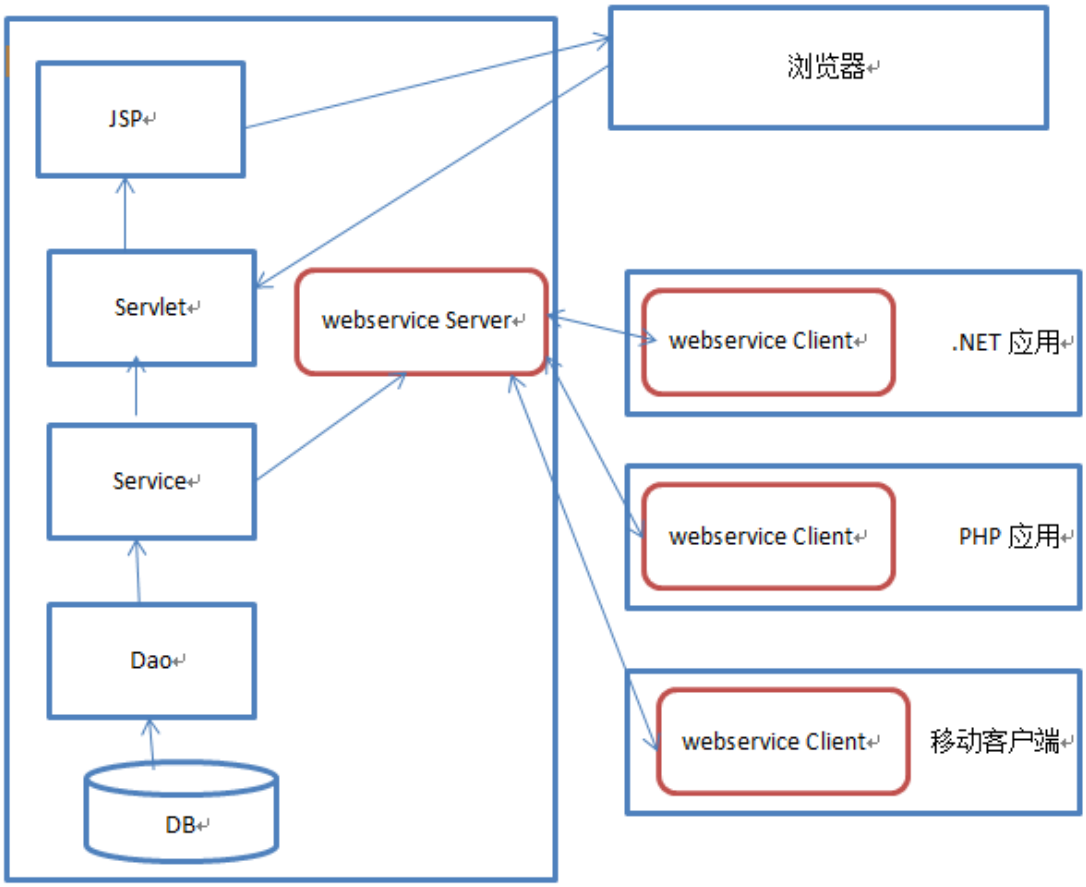
3. 多个跨平台、跨语言的应用间通信整合的方案（实际）

以各个网站显示天气预报功能为例:

气象中心的管理系统将收集的天气信息并将数据暴露出来(通过 WebService Server)，而各大站点的应用就去调用它们得到天气信息并以不同的样式去展示(WebService Client).

网站提供了天气预报的服务，但其实它们什么也没有做，**只是简单了调用了一下气象中心服务器上的一段代码而已**。

Web 应用(Java)



3.2. 为什么要用 Web service？

web service 能解决：

跨平台调用

跨语言调用

远程调用

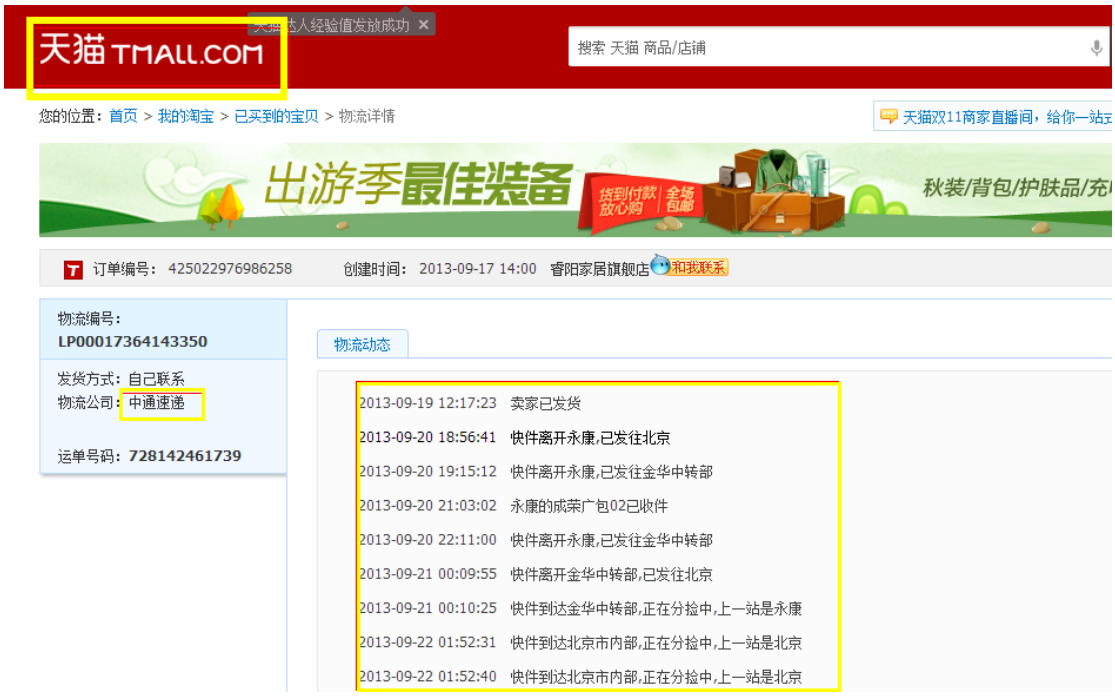
3.3. 什么时候使用 web Service?

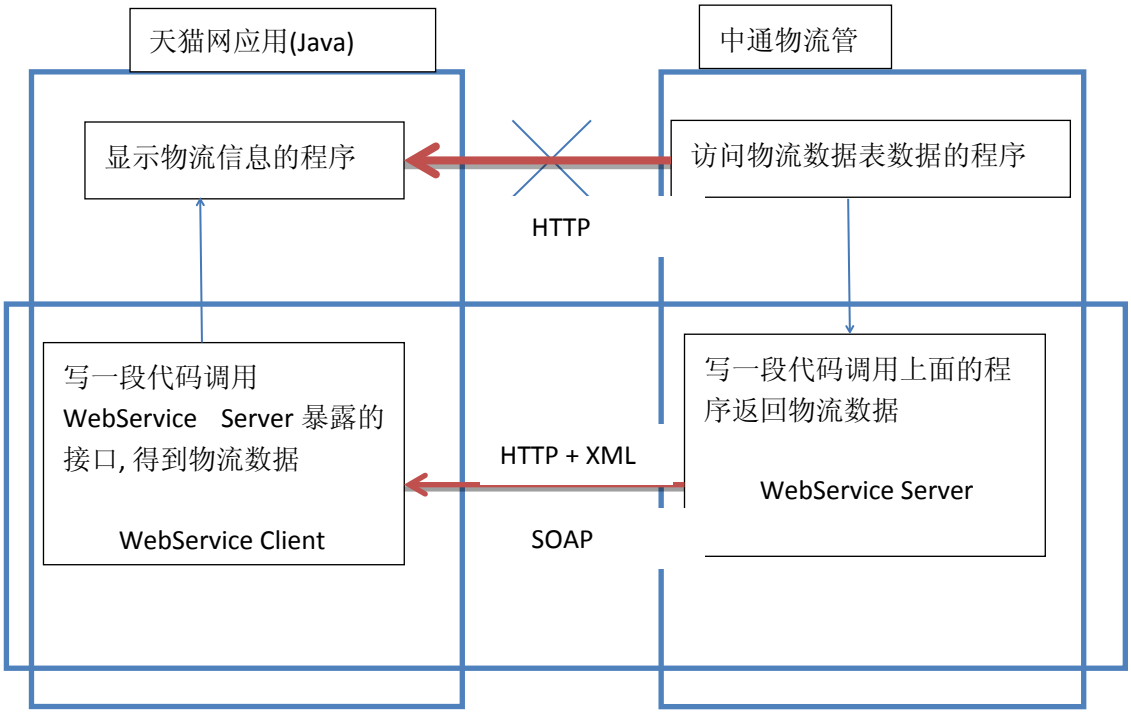
1. 同一家公司的新旧应用之间

2. 不同公司的应用之间

   分析业务需求：天猫网与中通物流系统如何交互？

3. 一些提供数据的内容聚合应用：天气预报、股票行情





4. Web Service 中的几个重要术语

4.1. WSDL： web service definition language

直译 : WebService 定义语言

1. 对应一种类型的文件**.wsdl**

2. **定义了 web service 的服务器端与客户端应用交互传递请求和响应数据的格式和方式**

3. 一个 web service 对应一个**唯一的 wsdl 文档**

4.2. SOAP：simple object access protocal

直译: 简单对象访问协议

1. 是一种简单的、基于 **HTTP 和XML的协议**, 用于在 WEB 上交换**结构化的数据**

2. soap 消息：**请求消息**和**响应消息**

3. **http+xml 片断**

4.3. SEI：WebService EndPoint Interface

直译: **web service 的终端接口，**

1. 就是 WebService 服务器端用来处理请求的接口

4.4. CXF：Celtix + XFire

一个 apache 的用于开发 webservice 服务器端和客户端的框架

5. 开发 webservice

5.1. 概述

- 开发手段：
  - 使用 JDK 开发(1.6 及以上版本)
  - 使用 CXF 框架开发(工作中)
- 组成：
  - 服务器端
  - 客户端

5.2. 使用 JDK 开发 WebService

1).开发服务器端

- Web Service 编码：
  - @WebService(SEI 和 SEI 的实现类)

- @WebMethod(SEI 中的所有方法)

- **发布 Web Service：**
  - Endpoint(终端, 发布 webservice)

2). 开发客户端

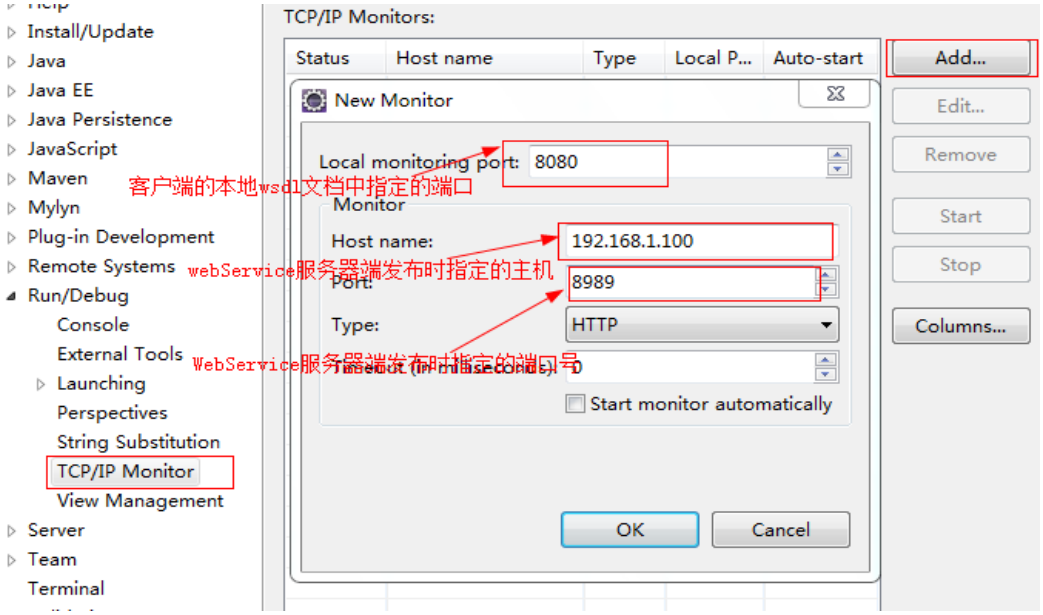
- **使用 eclipse 提供的 web service 浏览器访问**
  - 查看对应的 wsdl 文档：.....?wsdl (一般浏览器)
  - 请求 webService 并查看请求和响应消息(**webservice 浏览器**)

- **创建客户端应用编码方式访问**
  - 借助 jdk 的 wsimort.exe 工具生成客户端代码：

    wsimport -keep url //url 为 wsdl 文件的路径
  - 借助生成的代码编写请求代码

5.3. 监听请求: 使用 Eclipse 的 TCP/IP 工具

1. 将服务器端的 WSDL 文档保存到客户端本地
2. 修改文档: 将端口号从 8989 改为 8080
3. 根据本地的 wsdl 文档生成客户端代码, 并编写客户端的调用代码
4. 配置 eclipse 的 TCP/IP,启动监听



5. 执行客户端代码发送 WebService 请求

5.4. 调用免费的 web service(天气预报)

1. **Google”免费 WebService”, 找到提供天气预报 Webservice 的网络地址**
   - http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx
2. **使用 eclipse 的 web service 浏览器访问**
3. **客户端编码方式访问**
   - 借助命令工具自动生成客户端代码
   - 借助生成的代码编写请求代码

说明: 直接生成客户端代码会抛异常, 无法生成客户端代码, 解决办法:

1. 将对应的 wsdl 文档保存到本地
2. 修改 wsdl 文档的部分内容:

   将 `<s:element ref="s:schema" /><s:any />` 替换成 `<s:any minOccurs="2" maxOccurs="2"/>`

备注: 这个是 Java 调用 net 的 webservice 都有的问题

5.5. 使用 CXF 开发 web service

加入 cxf 的 jar 包即可，其它不需要改动

5.6. WebService 请求深入分析

1). 分析 WebService 的 WSDL 文档结构

1.1). 实例截图

1.2). 文档结构

```
<definitions>
        <types>
```

```
            <schema>
                <element>
        </types>
        <message>
            <part>
        </message>
        <portType>
            <operation>
                <input>
                <output>
        </portType>
        <binding>
            <operation>
                <input>
                <output>
        </binding>
        <service>
            <port>
                <address>
        </service>
</definitions>
```

SOAP Request Envelope:

```
xmlns:xsd="http://www.w3.org
xmlns:xsi="http://www.w3.org
- <soapenv:Body>
  - <q0:sayHello>
      <arg0>Jack</arg0>
    </q0:sayHello>
  </soapenv:Body>
</soapenv:Envelope>
```

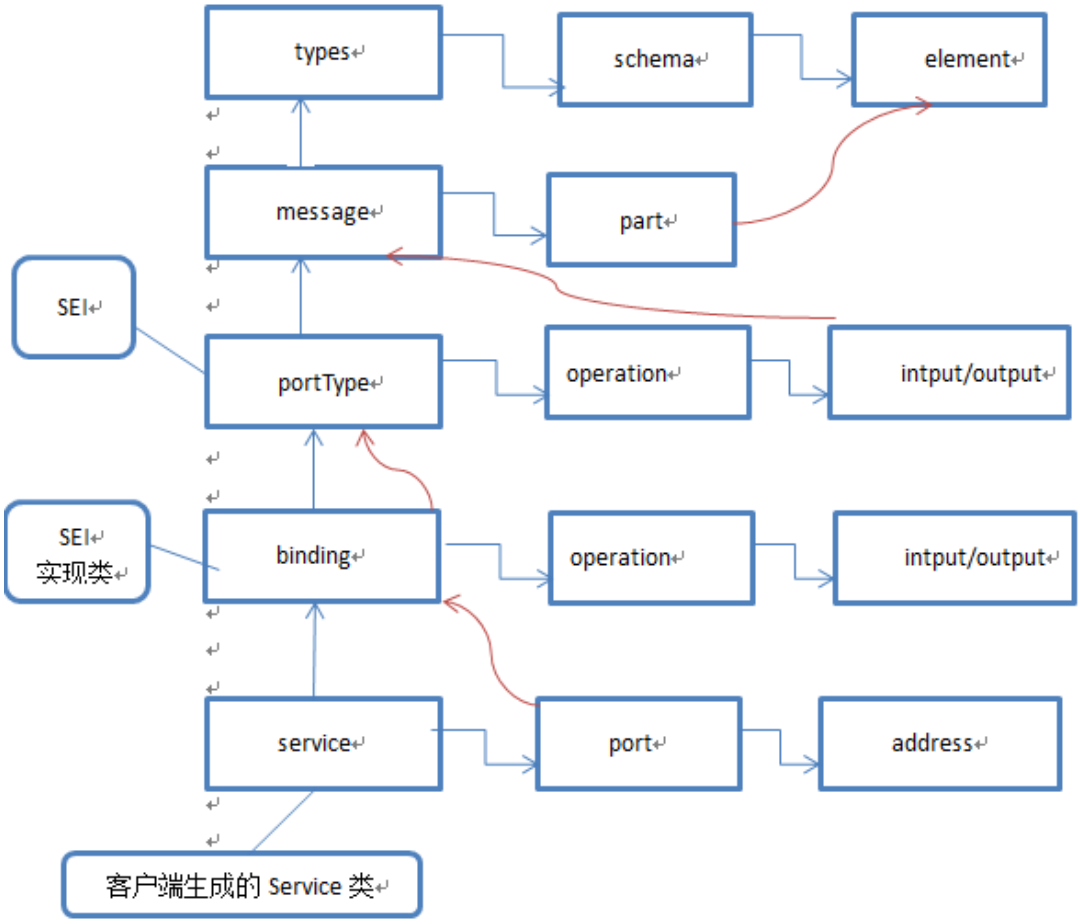
SOAP Response Envelope:

```
<S:Envelope xmlns:S="http://schemas.xmlso
- <S:Body>
  - <ns2:sayHelloResponse xmlns:ns2="http://
      <return>hello Jack</return>
    </ns2:sayHelloResponse>
  </S:Body>
</S:Envelope>
```

```
@WebService
public interface HelloWS {

    @WebMethod
    public String sayHello(String name);
}
```

1.3). 文档结构图



- **types** - 数据类型(标签)定义的容器，里面使用 schema 定义了一些标签结构供 message 引用

- **message** - 通信消息的数据结构的抽象类型化定义。引用 types 中定义的标签
- **operation** - 对服务中所支持的操作的抽象描述，一个 operation 描述了一个访问入口的请求消息与响应消息对。
- **portType** - 对于某个访问入口点类型所支持的操作的抽象集合，这些操作可以由一个或多个服务访问点来支持。
- **binding** - 特定端口类型的具体协议和数据格式规范的绑定。
- **service**- 相关服务访问点的集合
- **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。

2). 测试 CXF 支持的数据类型

1. 基本类型
   - int,float,boolean 等

2. 引用类型

- String
- 集合：数 1 组，List, Set, Map
- 自定义类型 Student

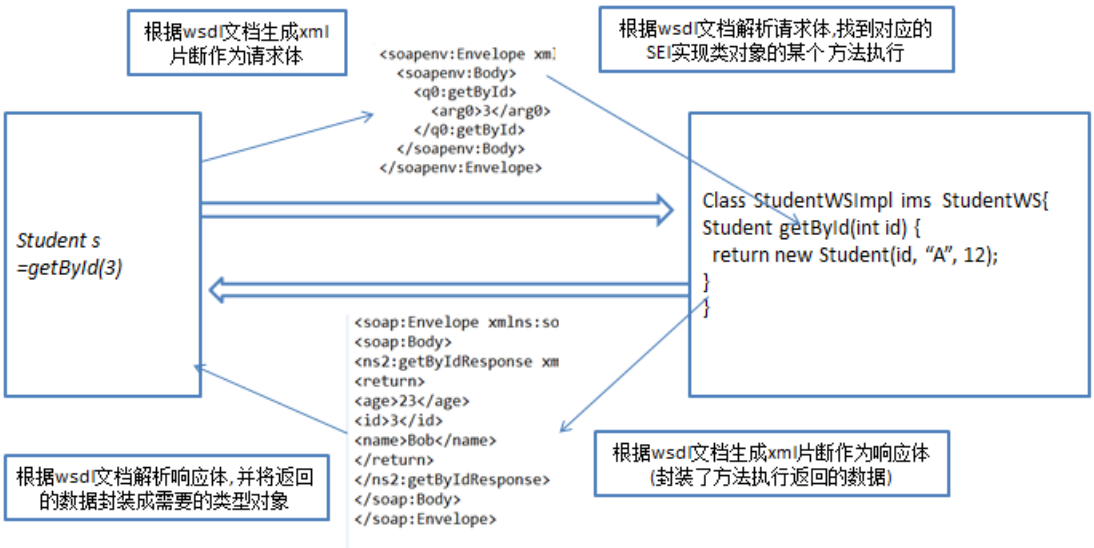
3). 一次 Web service 请求的流程

一次 web service 请求的本质:

1) 浏览器向服务器端发送了一个 soap 消息(http 请求+xml 片断)

2) 服务器端处理完请求后, 向客户端返回一个 soap 消息

那么它的流程是怎样的呢？



5.7. CXF 框架的深入使用

1).CXF 的拦截器

1.1) 理解

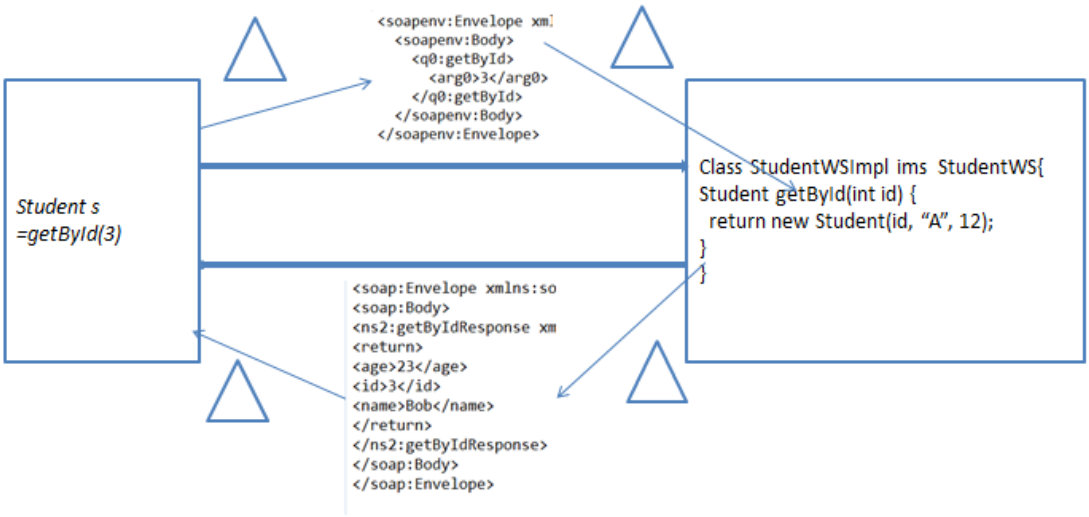
- **为什么设计拦截器？**
  1. 为了在 webservice 请求过程中,能动态操作请求和响应数据, CXF 设计了拦截器.
- 拦截器分类：
  1. 按所处的位置分：服务器端拦截器，客户端拦截器
  2. 按消息的方向分：入拦截器，出拦截器
  3. 按定义者分：系统拦截器，自定义拦截器
- **拦截器 API**

  Interceptor(拦截器接口)

  AbstractPhaseInterceptor(自定义拦截器从此继承)

  LoggingInInterceptor(系统日志入拦截器类)

  LoggingOutInterceptor(系统日志出拦截器类)



1.2) 编码实现拦截器

- 使用日志拦截器，实现日志记录
  - LoggingInInterceptor
  - LoggingOutInterceptor
- 使用自定义拦截器，实现用户名与密码的检验
  - 服务器端的 in 拦截器
  - 客户端的 out 拦截器
  - xfzhang/123456

2). 用 CXF 编写基于 spring 的 web service

2.1). 编码实现

1. **Server 端**
   - 创建 spring 的配置文件 beans.xml,在其中配置 SEI
   - 在 web.xml 中，配置上 CXF 的一些核心组件
2. **Client 端**
   - 生成客户端代码
   - 创建客户端的 spring 配置文件 beans-client.xml,并配置
   - 编写测试类请求 web service

2.2). 添加自定义拦截器

1. **Server 端**
   - 在 beans.xml 中，在 endpoint 中配置上入拦截器
2. **Client 端**
   - 通过 Client 对象设置出拦截器

5.8. 其它调用 WebService 的方式

1). Ajax 调用 webService

```
var data = '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/enve
//1. 创建xmlHttpRequest对象
var request = getRequest();
//2. 设置回调函数
request.onreadystatechange = function() {
    if (request.readyState == 4 && request.status == 200) {
        var dom = request.responseXML;
        var returnEle = dom.getElementsByTagName("return")[0];
        var id = returnEle.getElementsByTagName("id")[0].firstChild.data;
        alert(id);
    }
};
//3. 打开连接
request.open("POST", "http://localhost/day02_ws_cxf_spring2/orderws");
//4. 设置一请求头
request.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
//5. 发送请求
request.send(data); //将data作为请求体发送过去
```

跨域请求问题:

1. 什么是跨域请求?

1. sina.com--=->baidu.com/xxx.jsp

2. localhost----→192.168.42.165

2. 解决 ajax 跨域请求 webservice 的问题?

在客户端应用中使用 java 编码去请求 webservice，在页面中去请求自己的后台

2). Jquery 调用 WebService

```
var data = '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">'
        +'<soap:Header><atguigu><name>xfzhang</name><pwd>123456</pwd></atguigu>'
        +'</soap:Header><soap:Body><ns2:getOrder xmlns:ns2="http://bean.day02_ws_c>
        +'<arg0>123</arg0></ns2:getOrder></soap:Body></soap:Envelope>';
$.post(
    "http://localhost/day02_ws_cxf_spring2/orderws",
    data,
    function(msg) {
        alert($(msg).find("id").text());
        alert($(msg).find("no").text());
        alert($(msg).find("price").text());
    },
    "xml"
);
```

3). HttpURLConnection 调用 WebService

```
String data = "<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/e
String path = "http://192.168.1.103:9090/day02_ws_cxf/dataTypeWs";
URL url = new URL(path);
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("POST");
connection.setDoOutput(true);
connection.setDoInput(true);
connection.setRequestProperty("Content-Type", "text/xml;charset=utf-8");

OutputStream os = connection.getOutputStream();
os.write(data.getBytes("utf-8"));
os.flush();
int responseCode = connection.getResponseCode();
if(responseCode==200) {
    InputStream is = connection.getInputStream();
    ServletOutputStream outputStream = response.getOutputStream();
    byte[] buffer = new byte[1024];
    int len = 0;
    while((len=is.read(buffer))>0) {
        outputStream.write(buffer, 0, len);
    }
```

5.9. 通过注解修改 wsdl 文档

1). JDK 中的相关注解

1.1). @WebService

- 作用在具体类上。而不是接口。
- 一个类只有添加了此注解才可以通过 Endpoint 发布为一个 web 服务。
- 一个添加了此注解的类，必须要至少包含一个实例方法。静态方法和 final 方法不能被发布为服务方法。
- WebService 注解包含以下参数：

可选元素摘要	
String	**endpointInterface** 定义服务抽象 Web Service 协定的服务端点接口的完整名称。
String	**name** Web Service 的名称。
String	**portName** Web Service 的端口名称。
String	**serviceName** Web Service 的服务名称。
String	**targetNamespace** 如果 @WebService.targetNamespace 注释是关于某一服务端点
String	**wsdlLocation** 描述服务的预定义 WSDL 的位置。

1.2). @WebMethod

- 此注解用在方法上，用于修改对外暴露的方法。

```
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface WebMethod
```

定制一个公开为 Web Service 操作的方法。关联方法必须是公共方法且其参数可以返回

可选元素摘要

String	**action** 此操作的动作。
boolean	**exclude** 将某一方法标记为不作为一个 web 方法公开。
String	**operationName** 与此方法匹配的 wsdl:operation 的名称。

1.3). @WebResult

用于定制返回值到 WSDL 的映射

```
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface WebResult
```

定制返回值到 WSDL 部分和 XML 元素的映射关系。

可选元素摘要	
boolean	**header** 如果为 true，则结果是从消息头而不是消息正文获取的。
String	**name** 返回值的名称。
String	**partName** 表示此返回值的 wsdl:part 的名称。
String	**targetNamespace** 返回值的 XML 名称空间。

1.4). @WebParam

用于定义 WSDL 中的参数映射

```
@Retention(value=RUNTIME)
@Target(value=PARAMETER)
public @interface WebParam
```

定制单个参数到 Web Service 消息部分和 XML 元素的映射关系。

可选元素摘要

boolean	**header** 如果为 true，则参数是从消息头而不是消息正文获取的。
WebParam.Mode	**mode** 参数的流向（IN、OUT 或 INOUT 之一）。
String	**name** 参数名称。
String	**partName** 表示此参数的 wsdl:part 的名称。
String	**targetNamespace** 参数的 XML 名称空间。

1.5). @XmlElement

用于定义实体类的属性到 WSDL 中的映射(get/set 方法上)

可选元素摘要

String	**defaultValue** 此元素的默认值。
String	**name** XML 模式元素的名称。
String	**namespace** XML 模式元素的 XML 目标名称空间。
boolean	**nillable** 将元素声明自定义为 nillable。
boolean	**required** 自定义所需的元素声明。
Class	**type** 正被引用的 Java 类。

```
<wsdl:portType name="OrderWSImpl">
- <wsdl:operation name="getOrder">
    <wsdl:input message="tns:getOrder" name="getOrder" />
    <wsdl:output message="tns:getOrderResponse" name="getOrderResponse" />
  </wsdl:operation>
</wsdl:portType>
```

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="
 xmlns:tns="http://bean.day02_ws_cxf_spring.atguigu.com/" xmlns:soap="http:/
 xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="OrderWSImplService"
 targetNamespace="http://bean.day02_ws_cxf_spring.atguigu.com/">
```

```
▼<wsdl:service name="OrderWSImplService">
  ▼<wsdl:port binding="tns:OrderWSImplServiceSoapBinding" name="OrderWSImplPort">
     <soap:address location="http://localhost/day02_ws_cxf_spring2/orderws"/>
   </wsdl:port>
 </wsdl:service>
```

```
▼<xs:complexType name="getOrder">
  ▼<xs:sequence>
     <xs:element minOccurs="0" name="arg0" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
```

```
<xs:complexType name="getOrderResponse">
 ▼<xs:sequence>
    <xs:element minOccurs="0" name="return" type="tns:order"/>
  </xs:sequence>
</xs:complexType>

<xs:sequence>
  <xs:element minOccurs="0" name="_id" type="xs:string"/>
  <xs:element minOccurs="0" name="no" type="xs:string"/>
  <xs:element name="price" type="xs:double"/>
</xs:sequence>
```

2). 说明

即使是没有修改源代码，只修改了注解，客户端的代码也必须要重新生成, 否则调用将会失败。