物联网中间件

第六章 Web Services ——— WSDL、JAX-WS

参考材料

- WSDL
  - 课本10.3
  - https://www.yiibai.com/wsdl/wsdl_introduction.html
  - https://www.runoob.com/wsdl/wsdl-tutorial.html

- JAX-WS
  - 课本10.4

WSDL

- WSDL（网络服务描述语言，Web Services Description Language）是一门基于 XML 的语言，用于描述 Web Services 以及如何对它们进行访问。WSDL由Microsoft和IBM联合开发。

什么是 WSDL?

- WSDL 指网络服务描述语言
- WSDL 使用 XML 编写
- WSDL 是一种 XML 文档
- WSDL定义描述了如何访问Web服务以及它将执行的操作。
- WSDL发音为'wiz-dull'，拼写为'W-S-D-L'

WSDL用法

- WSDL通常与SOAP和XML Schema结合使用，以通过Internet提供Web服务。
- 连接到Web服务的客户端程序可以读取WSDL以确定服务器上可用的功能。
- 使用的任何特殊数据类型都以XML Schema的形式嵌入到WSDL文件中。
- 然后，客户端可以使用SOAP实际调用WSDL中列出的函数。

WSDL版本

- WSDL 1.1 - 于2001年3月15日发布，WSDL 1.1的规范可通过访问网址：
  - https://www.w3.org/TR/2001/NOTE-wsdl-20010315 了解。

- WSDL 2.0 - 于2007年6月26日发布，WSDL 2.0的规范可通过访问网址：
  - http://www.w3.org/TR/wsdl20-primer/ 了解。

- 许多编程API和测试工具广泛支持WSDL 1.1。 但WSDL 2.0仍然没有得到很好的支持，即使WSDL 2.0规范从2007年就已经开始引入。
- 以下是使用WSDL 1.1和WSDL 2.0测试的编程API和测试工具的列表：

API/工具	版本	年份	WSDL 1.1	WSDL 2.0
SoapUI	5.2.0	2018	Yes	No
PHP SOAP扩展	7.0.2	2015	Yes	No
Perl SOAP::Lite	1.27	2017	Yes	No
Apache Axis2/Java	1.7.8	2018	Yes	Yes

WSDL 文档

- WSDL 文档仅仅是一个简单的 XML 文档。
- 它包含一系列描述某个 web service 的定义。

WSDL <definitions>元素

- WSDL <definitions>元素必须是所有WSDL文档的根元素，它定义了Web服务的名称。

```
<definitions name="HelloService"
             targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
             xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ..........................................
</definitions>
```

- 指定此文档名为：HelloService。
- 指定targetNamespace属性，targetNamespace是XML Schema的约定，它使WSDL文档能够引用自身。在此示例中，我们指定了一个： http://www.examples.com/wsdl/HelloService.wsdl 的targetNamespace指定默认命名空间：xmlns=http://schemas.xmlsoap.org/wsdl/ 。 因此，假定所有没有名称空间前缀的元素(如message或portType)都是默认WSDL名称空间的一部分。
- 指定在整个文档的其余部分中使用的其它命名空间。

WSDL 文档结构

元素	定义
<portType>	web service 执行的操作
<message>	web service 使用的消息
<types>	web service 使用的数据类型
<binding>	web service 使用的通信协议

```
<definitions>

    <types>
        data type definitions.........
    </types>

    <message>
        definition of the data being communicated....
    </message>

    <portType>
        set of operations......
    </portType>

    <binding>
        protocol and data format specification....
    </binding>

</definitions>
```

WSDL types

- *<types>* 元素定义 web service 使用的数据类型。

- 为了最大程度的平台中立性，WSDL 使用 XML Schema 语法来定义数据类型。
- 如果服务仅使用XML Schema内置的简单类型(如字符串和整数)，则不需要types元素。

- 数据类型解决了识别数据类型以及要与Web服务一起使用的格式的问题。 类型信息在发送方和接收方之间共享。

WSDL 消息

- WSDL <message>元素描述了Web服务生产者和消费者之间交换的数据。

- 每个Web服务都有两条消息：输入和输出。
- 输入描述Web服务的参数
- 输出描述Web服务的返回数据。
- 每条消息包含零个或多个<part>参数，每个参数对应一个Web服务函数的参数。
- 每个<part>参数与<types>元素中定义的具体类型相关联。

WSDL 消息

```
<message name = "SayHelloRequest">
    <part name = "firstName" type = "xsd:string"/>
</message>

<message name = "SayHelloResponse">
<part name = "greeting" type = "xsd:string"/>
</message>
```

WSDL 消息

- 这里定义了两个消息元素。 第一个表示请求消息SayHelloRequest，第二个表示响应消息SayHelloResponse。
- 这些消息中都包含一个<part>元素。
- 对于请求，<part>指定函数参数; 在这个示例中，指定一个firstName参数。

- 对于响应<part>指定函数返回值; 在这个示例中，指定一个问候语(greeting)返回值。

WSDL 端口

- *<portType>* 元素是最重要的 WSDL 元素。
- 它描述一个 web service可被执行的操作，以及相关的消息。
- 可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）。 该接口有点类似Java的接口，都是定义了一个抽象类型和方法，没有定义实现。

WSDL 端口

- 一个portType中可以定义多个operation，一个operation可以看作是一个方法

- < operation >元素组合了多个消息(<message>)元素，以形成完整的单向或往返操作。

- 例如，< operation >可以将一个请求和一个响应消息组合成单个请求/响应操作。 这在SOAP服务中最常用。

WSDL 端口

```
<portType name = "Hello_PortType">
    <operation name = "sayHello">
        <input message = "tns:SayHelloRequest"/>
        <output message = "tns:SayHelloResponse"/>
    </operation>
</portType>
```

- portType元素定义了一个名称为sayHello的操作。
- 该操作由单个输入消息SayHelloRequest和一个输出消息SayHelloResponse组成。

WSDL 实例

```
<message name = "SayHelloRequest">
    <part name = "firstName" type = "xsd:string"/>
</message>

<message name = "SayHelloResponse">
<part name = "greeting" type = "xsd:string"/>
</message>

<portType name = "Hello_PortType">
<operation name = "sayHello">
    <input message = "tns:SayHelloRequest"/>
    <output message = "tns:SayHelloResponse"/>
</operation>
</portType>
```

- 在这个例子中，*<portType>* 元素把 "Hello_PortType" 定义为某个*端口*的名称，把 "sayHello" 定义为某个*操作*的名称。
- 操作 " sayHello " 拥有一个名为 " sayHelloRequest" 的*输入消息*，以及一个名为 " sayHelloResponse" 的*输出消息*。
- *<message>* 元素可定义每个消息的*参数*，以及相关联的数据类型
- 对比传统的编程， Hello_PortType 是一个函数库，而 " sayHello " 是带有输入参数 " sayHelloRequest " 和返回参数 sayHelloResponse 的一个函数。

WSDL操作类型

类型	定义
One-way	此操作可接受消息，但不会返回响应。
Request-response	此操作可接受一个请求并会返回一个响应
Solicit-response	此操作可发送一个请求，并会等待一个响应。
Notification	此操作可发送一条消息，但不会等待响应。

One-Way 操作

```
<message name="newTermValues">
    <part name="term" type="xs:string"/>
    <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
<operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
</operation>
</portType >
```

- 在这个例子中，端口 "glossaryTerms" 定义了一个名为 "setTerm" 的 one-way 操作。
- 这个 "setTerm" 操作可接受新术语表项目消息的输入，这些消息使用一条名为 "newTermValues" 的消息，此消息带有输入参数 "term" 和 "value"。不过，没有为这个操作定义任何输出。

Request-Response 操作

```
<message name="getTermRequest">
    <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
<part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
<operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
</operation>
</portType>
```

- 在这个例子中，端口 "glossaryTerms" 定义了一个名为 "getTerm" 的 request-response 操作。
- "getTerm" 操作会请求一个名为 "getTermRequest" 的输入消息，此消息带有一个名为 "term" 的参数，并将返回一个名为 "getTermResponse" 的输出消息，此消息带有一个名为 "value" 的参数。

- 要封装错误，还可以指定可选的fault元素

Solicit-response 询问 - 响应

- 该服务发送消息并接收响应。 因此，操作有一个output元素，后跟一个input元素。 要封装错误，还可以指定可选的fault元素。

```
<wsdl:portType .... > *
<wsdl:operation name = "nmtoken" parameterOrder = "nmtokens">
<wsdl:output name = "nmtoken"  message = "qname"/>
<wsdl:input name = "nmtoken"  message = "qname"/>
<wsdl:fault name = "nmtoken" message = "qname"/>*
</wsdl:operation>
        </wsdl:portType >
```

Notification

- 该服务发送一条消息。 因此，操作具有单个output元素。

```
<wsdl:portType .... > *
<wsdl:operation name = "nmtoken">
<wsdl:output name = "nmtoken"  message = "qname"/>
</wsdl:operation>
        </wsdl:portType>
```

WSDL Bindings

- binding元素将一个抽象portType映射到一组具体协议(SOAP和HTTP)、消息传递样式、编码样式。
- *<binding>* 元素为每个端口定义消息格式和协议细节。
- 绑定可以通过多种传输方式提供，包括HTTP GET，HTTP POST或SOAP。
- 绑定提供了有关用于传输portType操作的协议的具体信息。
- 绑定提供服务所在的信息。
- 对于SOAP协议，绑定是使用<soap:binding>，表示传输是基于HTTP协议的SOAP消息。

WSDL Bindings

- 绑定元素有两个属性：name和type属性。

```
<binding name = "Hello_Binding" type =
"tns:Hello_PortType">
```

- 在上面示例代码中，name属性定义绑定的名称，type属性指向绑定的端口，在本例中为Hello_PortType端口。

soap:binding

- soap:binding 元素表示将通过SOAP提供绑定。
- style属性指示SOAP消息格式的整体样式。 style的值rpc指定RPC格式。
- transport属性指示SOAP消息的传输。
  - http://schemas.xmlsoap.org/soap/http值表示SOAP HTTP传输
  - http://schemas.xmlsoap.org/soap/smtp 表示SOAP SMTP传输。

WSDL Bindings

- soap:operation
- 此元素指示特定操作与特定SOAP实现的绑定。
- soapAction 用来定义消息请求的地址。也就是消息发送到哪个操作或者服务方法

- soap:body
- 此元素用于指定输入和输出消息的详细信息

```
<binding name = "Hello_Binding" type = "tns:Hello_PortType">
    <soap:binding style = "rpc" transport = "http://schemas.xmlsoap.org/soap/http"/>
    <operation name = "sayHello">
        <soap:operation soapAction = " http://www.examples.com/SayHello "/>

        <input>
            <soap:body
                    encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                    namespace = "urn:examples:helloservice" use = "encoded"/>
        </input>

        <output>
            <soap:body
                    encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                    namespace = "urn:examples:helloservice" use = "encoded"/>
        </output>
    </operation>
</binding>
```

- <operation > 属性可以包含 < input > ，< output > 和 < fault > 的元素，它们都对应于PortType栏中的相同元素。
- 只有 < input > ，< output > 元素在上例中提供。这三个元素中的每一个可有一个可选 的"name"属性。
- 在本例的 < input > 元素中有一个 < soap:body > 元素，它指定了哪些信息被写进 SOAP消息的信息体中。该元素有以下属性：

- Use　用于制定数据是“encoded”还 是“literal”。
- Namespace　每个SOAP消息体可以有其自己的namespace来防止命名冲突。
- EncodingStyle　对SOAP编码，它应该有以下URI值：
"http://schemas.xmlsoap.org/soap/encoding"

WSDL <ports>元素

- WSDL <ports>元素通过为绑定指定单个地址来定义单个端点。
- 这是指定端口的语法 –

```
<wsdl:port name = "nmtoken" binding = "qname"> *
<-- extensibility element (1) -->
</wsdl:port>
```

- port元素有两个属性：name和binding。
- name属性在封闭的WSDL文档中定义的所有端口中提供唯一名称
- binding属性是指使用WSDL定义的链接规则进行绑定。

WSDL <service>元素

- WSDL <service>元素定义Web服务支持的端口。 对于每个支持的操作，都有一个<port>元素。 service元素是端口的集合。

- Web服务客户端可以从服务元素中学习以下内容 -
- 在哪里访问该服务?
- 通过哪个端口访问Web服务?
- 如何定义通信消息?

```
<service name = "Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding = "tns:Hello_Binding" name = "Hello_Port">
        <soap:address
                location = "http://www.examples.com/SayHello/">
    </port>
</service>
```

- <soap:address>用于指定端口的地址信息。
- 端口不得指定多个地址。
- 端口不得指定除地址信息之外的任何绑定信息。

```
<binding name =" Hello_Binding" type = "tns:Hello_PortType">
    <soap:binding style = "rpc"
                  transport = "http://schemas.xmlsoap.org/soap/http"/>
    <operation name = "sayHello">
        <soap:operation soapAction = "sayHello"/>

        <input>
            <soap:body
                    encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                    namespace = "urn:examples:helloservice" use = "encoded"/>
        </input>

        <output>
            <soap:body
                    encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                    namespace = "urn:examples:helloservice" use = "encoded"/>
        </output>
    </operation>
</binding>
```

WSDL例子

- HelloService.wsdl

WSDL 样式（SOAP 样式）

- WSDL SOAP 绑定可以是 RPC 样式的绑定，也可以是文档样式的绑定。同样，SOAP 绑定可以有编码的用法，也可以有文字的用法。这给我们提供了四种样式/用法模型：
- RPC/编码
- RPC/文字
- 文档/编码
- 文档/文字
- 文档/文字包装的样式

RPC/编码

- Java 方法
- public void myMethod(int x);

- **用于 myMethod 的 RPC/编码的 WSDL**

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
</message>
<message name="empty"/>
<portType name="PT">
<operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
</operation>
</portType>
<binding .../>
```

用于 myMethod 的 RPC/编码的 SOAP 消息

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
        </myMethod>
    </soap:body>
</soap:envelope>
```

优点

- WSDL 基本达到了尽可能地简单易懂的要求。
- 操作名出现在消息中，这样接收者就可以很轻松地把消息发送到方法的实现。

缺点

- 类型编码信息（比如 xsi:type="xsd:int" ）通常就是降低吞吐量性能的开销。

- 您不能简单地检验此消息的有效性，因为只有

<x xsi:type="xsd:int">5</x> 行包含在 Schema 中定义的内容；其余的 soap:body 内容都来自 WSDL 定义。

RPC/文字

- 用于 myMethod 的 RPC/文字的 WSDL

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
</message>
<message name="empty"/>
<portType name="PT">
<operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
</operation>
</portType>
<binding .../>
```

RPC/文字的 SOAP 消息

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
        </myMethod>
    </soap:body>
</soap:envelope>
```

优点

- WSDL 还是基本达到了尽可能地简单易懂的要求。
- 操作名仍然出现在消息中。
- 去掉了类型编码。

缺点

- 您仍然不能简单地检验此消息的有效性，因为只有

  <x >5</x> 行包含在 Schema 中定义的内容；其余的 soap:body 内容都来自 WSDL 定义。

文档/文字

- 用于 myMethod 的文档/文字的 WSDL

```
<types>
    <schema>
        <element name="xElement" type="xsd:int"/>
    </schema>
</types>
<message name="myMethodRequest">
<part name="x"
      element="xElement"/>
</message>
<message name="empty"/>
<portType name="PT">
<operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
</operation>
</portType>
<binding .../>
```

用于 myMethod 的文档/文字的 SOAP 消息

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
    </soap:body>
</soap:envelope>
```

优点

- 没有编码信息
- 您可以在最后用任何 XML 检验器检验此消息的有效性。soap:body （ <xElement>5</xElement> ）中每项内容都定义在 Schema 中

缺点

- WSDL 变得有些复杂。不过，这是一个非常小的缺点，因为 WSDL 并没有打算由人来读取。

- SOAP 消息中缺少操作名。而如果没有操作名，服务器把请求发送具体的实现就可能比较困难，并且有时变成不可能完成的任务

文档/文字包装的样式

- 用于 myMethod 的文档/文字包装的 WSDL

```
<types>
    <schema>
        <element name="myMethod"/>
        <complexType>
            <sequence>
                <element name="x" type="xsd:int"/>
            </sequence>
        </complexType>
    </element>
</schema>
        </types>
<message name="myMethodRequest">
<part name="
        parameters" element="
        myMethod"/>
</message>
<message name="empty"/>
<portType name="PT">
<operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
</operation>
</portType>
```

用于 myMethod 的文档/文字包装的 SOAP 消息

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
        </myMethod>
    </soap:body>
</soap:envelope>
```

- SOAP 消息看起来非常类似于 RPC/文字的 SOAP 消息。您可能会说，它看起来与 RPC/文字的 SOAP 消息是完全一样的，不过，这两种消息之间存在着微妙的区别。在 RPC/文字的 SOAP 消息中，<soap:body> 的 <myMethod> 子句是操作的名称。在文档/文字包装的 SOAP 消息中， <myMethod> 子句是单个输入消息的组成部分参数的元素的名称。因此，包装的样式具有这样的一个特征，输入元素的名称与操作的名称是相同的。此样式是把操作名放入 SOAP 消息的一种巧妙方式。

文档/文字包装的样式的特征

- 输入消息只有一个组成部分。
- 该部分就是一个元素。
- 该元素有与操作相同的名称。
- 该元素的复杂类型没有属性。

优点

- 没有编码信息。
- 出现在 soap:body 中的每项内容都是由 Schema 定义的，所以您现在可以很容易地检验此消息的有效性。
- 方法名又出现在 SOAP 消息中。

缺点

- WSDL 甚至更复杂，但是这仍然是一个非常小的缺点。

根据以下代码判断WSDL样式：

```
SOAP Message Itself:
<soap: Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Body>
<myHelloWorld>
    <i>7</i>
</myHelloWorld>
</soap:Body>
        </soap: Envelope>

        Part of a WSDL Document:
<message name="myHelloWorldRequest">
<part name="i" type="xsd:int" />
        </message>
```

- A）RPC/编码
- B）RPC/文字
- C）文档/编码
- D）文档/文字
- E）文档/文字包装的样式

Web服务器(Web Server)

- Web服务器可以解析(handles)HTTP协议。
- 当Web服务器接收到一个HTTP请求(request)，会返回一个HTTP响应 (response) ,例如送回一个HTML页面。
- 当一个请求(request)被送到Web服务器里来时，它只单纯的把请求(request)传递给可以很好的处理请求 (request)的程序
- Web服务器仅仅提供一个可以执行服务器端(server-side)程序和返回(程序所产生的)响应(response)的环境，而不会超出职能范围。

JAX-WS

- JAX-WS(Java API for XML Web Services)规范是一组XML web services的JAVA API，JAX-WS允许开发者可以选择RPC-oriented或者message-oriented 来实现自己的web services。
- 在服务器端，用户只需要通过Java语言定义远程调用所需要实现的接口SEI（service endpoint interface），并提供相关的实现，通过调用JAX-WS的服务发布接口就可以将其发布为WebService接口。
- 在客户端，用户可以通过JAX-WS的API创建一个代理（用本地对象来替代远程的服务）来实现对于远程服务器端的调用。

JAX-WS由来

- Web 服务已经出现很久了。
- 首先是 SOAP，但 SOAP 仅描述消息的情况
- 然后是 WSDL，WSDL 并不会告诉您如何使用 Java编写 Web 服务
- 在这种情况下，JAX-RPC 1.0 应运而生。
- 经过数月使用之后，编写此规范的 Java Community Process (JCP) 人员认识到需要对其进行一些调整，调整的结果就是 JAX-RPC 1.1。

JAX-WS由来

- JAX-RPC 1.1使用大约一年之后，JCP 人员希望构建一个更好的版本：JAX-RPC 2.0。
- JAX-RPC 2.0其主要目标是与行业方向保持一致，但行业中不仅只使用 RPC Web 服务，还使用面向消息的 Web 服务。因此从名称中去掉了“RPC”，取而代之的是“WS”（当然表示的是 Web 服务）。因此 JAX-RPC 1.1 的后续版本是 JAX-WS 2.0——Java API for XML-based Web services。

Java 注解（Annotation）

```
package myHelloWorld;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{
    @WebMethod String HelloWorld(String name);
}
```

Java 注解（Annotation）

- 注解（Annotation），也叫元数据。一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。

- Java代码中使用注释是为了提升代码的可读性，也就是说，注释是给人看的（对于编译器来说没有意义）。注解可以看做是注释的“强力升级版"，它可以向编译器、虚拟机等解释说明一些事情（也就是说它对编译器等工具也是“可读”的）。

- 除了向编译器等传递一些信息，我们也可以使用注解生成代码。比如我们可以使用注解来描述我们的意图，然后让注解解析工具来解析注解，以此来生成一些”模板化“的代码。比如Hibernate、Spring等框架大量使用了注解，来避免一些重复的工作。

定义Server端的WebService服务接口

```
package myHelloWorld;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{
    @WebMethod String HelloWorld(String name);
}
```

- 类定义上，加了一个“@WebService”的annotation，这是定义JAX-WS定义WebService的关键，这个annotation用来告诉java解析器你希望把这个接口中的方法发布成一些WebService的服务。

- @SOAPBinding标注表示此Web服务使用SOAP协议，style = Style.RPC 表明使用RPC/文字模式

- **@WebMethod，**该注解用于用@WebService注解的类或接口的方法上，表示要发布的方法

有了接口定义，下面给出一个实现

```
package myHelloWorld;
import javax.jws.WebService;
//Service Implementation
@WebService(endpointInterface = "myHelloWorld.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
    @Override
    public String HelloWorld(String name) {
        System.out.println(name+" says hello");
        return "Hello World " + name;
    }
}
```

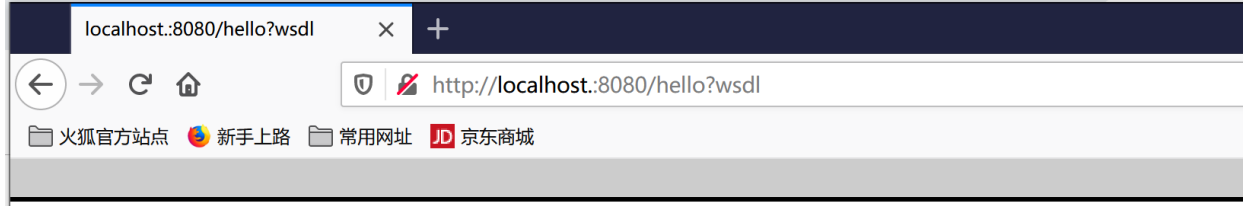
- 这里WebService annotation里加了一个参数"endpointInterface"，这个参数用来指定这个WebService的抽象服务接口
- @Override - 检查该方法是否是重写方法。如果发现其父类，或者是引用的接口中并没有该方法时，会报编译错误。
- @Override 帮助自己**检查**是否正确的复写了父类中已有的方法。编译器可以给你**验证**@Override下面的方法名是否是你父类中所有的，如果没有就算参数不匹配便报错。
- 当你如果没写@Override，而你下面的方法名又写错了，这时你的编译器是可以编译通过的，因为编译器以为这个方法是你的子类中自己增加的方法。

发布服务

```
package myHelloWorld;
import javax.xml.ws.Endpoint;
import myHelloWorld.HelloWorldImpl;
//Endpoint publisher
public class HelloWorldPublisher{
    public static void main(String[] args) {
        Endpoint.publish("http://localhost.:8080/hello", new HelloWorldImpl());
    }
}
```

- 这里publish方法需要两个参数:
- address：服务对外暴露的用于调用服务的地址
- implementor：服务的实现对象启动这个Server类，就可以访问服务了。

- 要测试服务有没有启动，可以输入 http://localhost.:8080/hello?wsdl ，如果一切正常，就可以看到一个wsdl定义内容，表示服务已经成功启动。



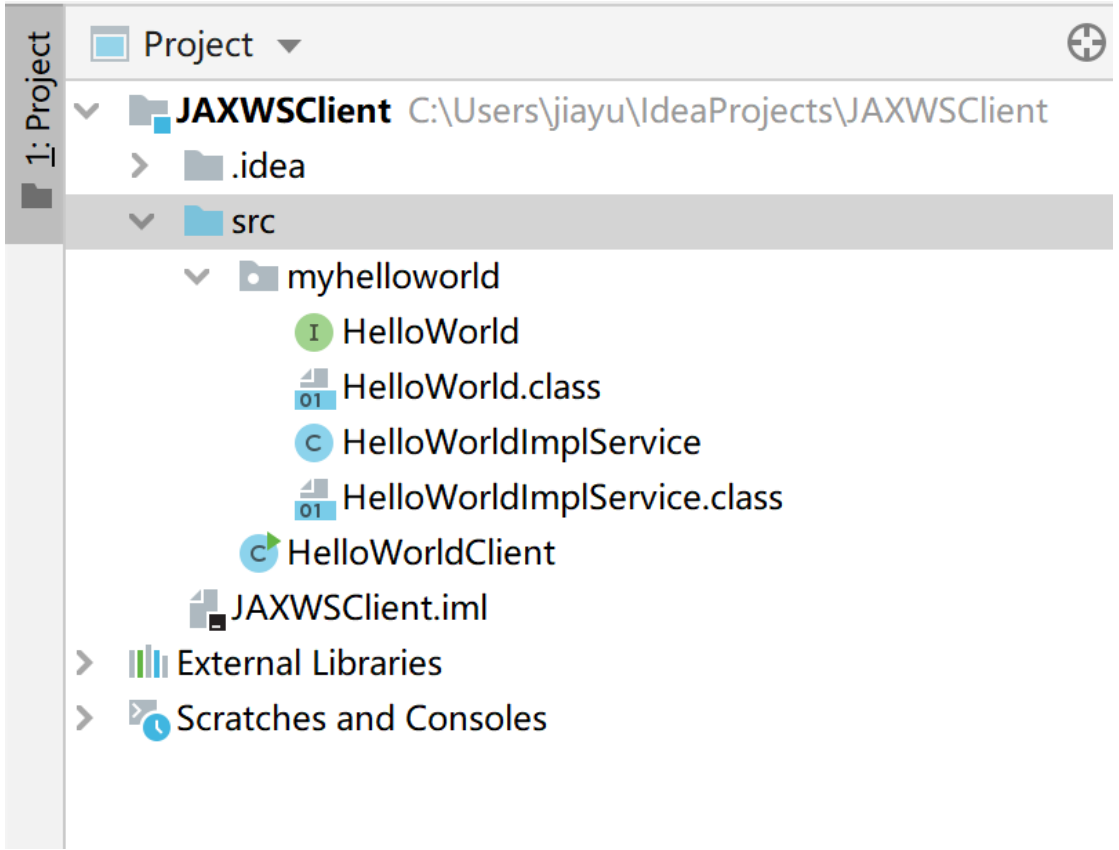

```
<!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
 -->
<!--
    Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
 -->
<definitions targetNamespace="http://myHelloWorld/" name="HelloWorldImplService">
  <types/>
  <message name="HelloWorld">
    <part name="arg0" type="xsd:string"/>
  </message>
  <message name="HelloWorldResponse">
    <part name="return" type="xsd:string"/>
  </message>
  <portType name="HelloWorld">
    <operation name="HelloWorld">
      <input message="tns:HelloWorld"/>
      <output message="tns:HelloWorldResponse"/>
    </operation>
  </portType>
  <binding name="HelloWorldImplPortBinding" type="tns:HelloWorld">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="HelloWorld">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal" namespace="http://myHelloWorld/"/>
      </input>
      <output>
        <soap:body use="literal" namespace="http://myHelloWorld/"/>
      </output>
    </operation>
  </binding>
  <service name="HelloWorldImplService">
    <port name="HelloWorldImplPort" binding="tns:HelloWorldImplPortBinding">
      <soap:address location="http://localhost.:8080/hello"/>
    </port>
  </service>
</definitions>
```

wsimport工具的使用

- JDK中提供了一个 wsimport 的工具，路径为 “JDK_PATH/bin”，可以相当方便的用来创建相应wsdl的Service类文件和Port类文件。

- -d:生成客户端执行类的class文件
- -s:生成客户端执行类的源文件
- -keep:表示导出webservice的class文件时是否也导出源代码java文件
- 在客户端应用程序的文件夹下执行命令wsimport -keep http://localhost.:8080/hello?wsdl -d

根据这个wsdl生成几个相应的类文件，



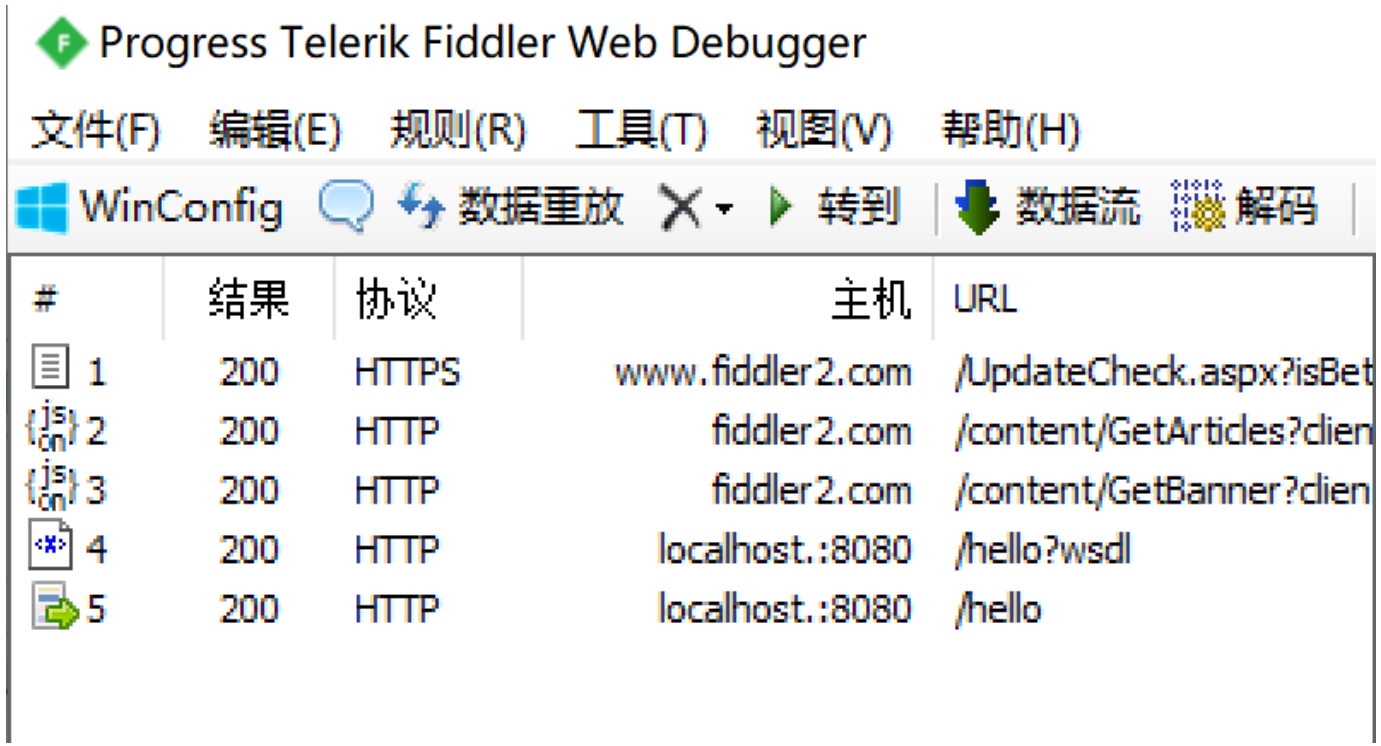
- 其中主要的是一个Service类 HelloWorldImplService.java 和 一个Interface接口 HelloWorld

有了这些类以后，就可以很简单的实现Client端

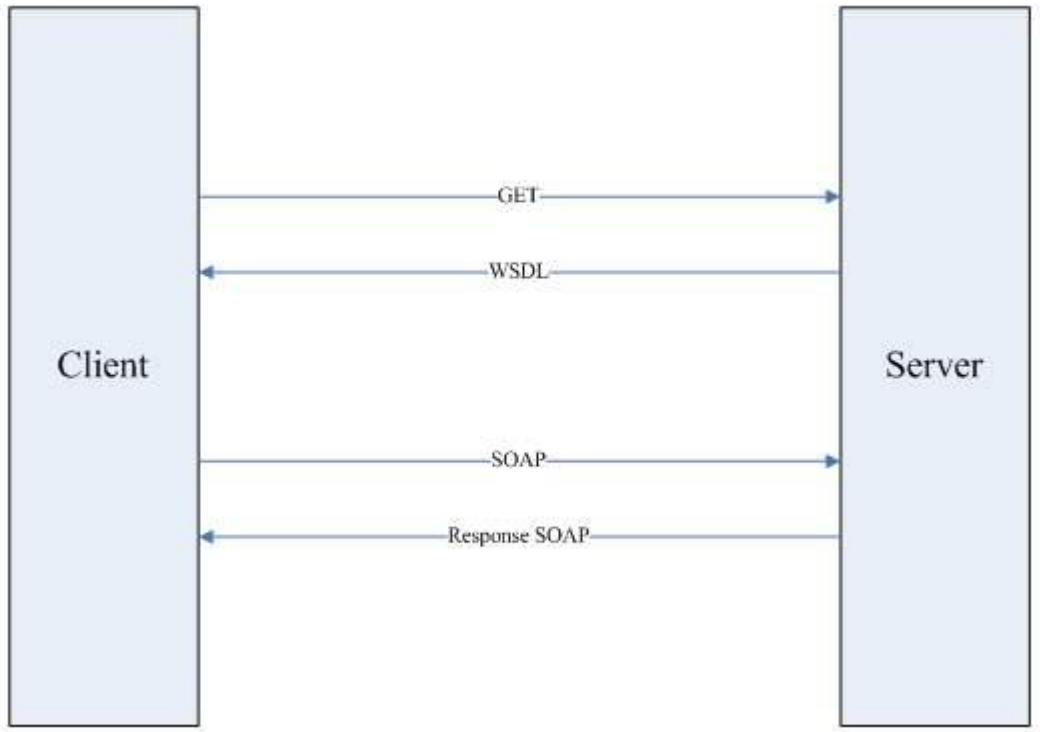
```
import myhelloworld.HelloWorld;
import myhelloworld.HelloWorldImplService;
public class HelloWorldClient {
    public static void main(String[] args) {
        System.setProperty("http.proxyHost", "127.0.0.1");
        System.setProperty("http.proxyPort", "8888");
        HelloWorldImplService myHelloWorld = new HelloWorldImplService();
        HelloWorld myinterface = myHelloWorld.getHelloWorldImplPort();
//Note the format of the operation call "helloWorld".
//This matches the format in the wsimport-generated HelloWorld.java file.
        String response = myinterface.helloWorld("AHU");
        System.out.println(response);
    }
}
```

使用Fiddler抓取SOAP通信数据包

- 注意：默认的时候Fiddler是不能嗅探到localhost的网站。 在localhost后面加个点号，Fiddler就能嗅探到。
- 这个项目是作为 webservice 的客户端，来调用 webservice 服务端的，即客户端调用服务端的模式，所以直接使用 fiddler 是抓不到包的。
- 我们要在 java 的方法中加入代理设置：
- System.setProperty("http.proxyHost", "127.0.0.1");
- System.setProperty("http.proxyPort", "8888");
- 这样，所有的方法调用都会通过 fiddler 软件，这样我们就可以看到抓到的数据包


Progress Telerik Fiddler Web Debugger
文件(F) 编辑(E) 规则(R) 工具(T) 视图(V) 帮助(H)
WinConfig 数据重放 转到 数据流 解码
结果 协议 主机 URL
1 200 HTTPS www.fiddler2.com /UpdateCheck.aspx?isBet
2 200 HTTP fiddler2.com /content/GetArticles?clien
3 200 HTTP fiddler2.com /content/GetBanner?clien
4 200 HTTP localhost.:8080 /hello?wsdl
5 200 HTTP localhost.:8080 /hello

通信流程图示



客户端发送的SOAP消息

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:HelloWorld xmlns:ns2="http://myHelloWorld/">
            <arg0>AHU</arg0>
        </ns2:HelloWorld>
    </S:Body>
</S:Envelope>
```

服务器回复的SOAP消息

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>

        <ns2:HelloWorldResponse xmlns:ns2="http://myHelloWorld/">
            <return>Hello World AHU</return>
        </ns2:HelloWorldResponse>
    </S:Body>
</S:Envelope>
```

Dynamic Proxy Client

- 不需要使用wsimport 命令
- 代理对象运行时才生成
- 但是需要提供Service Endpoint Interface 文件经编译过后的.class 文件 （SEI：服务器端的接口定义）

```java
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import myHelloWorld.HelloWorld;
/* This example does not use wsimport. Instead, it
creates a service instance manually. */
public class HelloWorldClient{
    public static void main(String[] args) throws Exception {
        URL location_of_wsdl = new URL("http://localhost.:8080/hello?wsdl");
        QName name_of_service = new QName("http://myHelloWorld/",
                "HelloWorldImplService");
        Service service = Service.create(location_of_wsdl, name_of_service);
        HelloWorld hello = service.getPort(HelloWorld.class);
        String response = hello.HelloWorld("Dynamic Proxy Client");
        System.out.println(response);
    }
}
```

- wsdl文件路径：需要读取服务端提供的wsdl定义文件

- 要调用的Service的Qname：一个wsdl中可能定义了多个Service，所以需要指定要调用的service名

- QName(String namespaceURI, String localPart)
- 指定名称空间 URI 和本地部分的 QName 构造方法。

- 创建Service：有了wsdl的service的qname以后，就可以创建对应的service对象了
- 取得相应的Port：有了Service，然后就可以取得Service中的某个Port
- 调用方法：最后相应的信息都取到以后，就可以调用希望的方法了

使用互联网上Web Services 提供商提供的服务

- http://www.webxml.com.cn/zh_cn/web_services.aspx

- https://www.programmableweb.com

实验演示

答疑

- 据以下服务器端的部分WSDL文档（注：……代表省略部分内容），以及客户端服务器端与客户端通信的SOAP消息，回答下列问题

1.WSDL以及SOAP的定义以及作用。

2.简述该WEB Service的名称、地址以及大致功能，服务器接受的参数的值以及数据类型，服务器的返回值以及数据类型

```
<definitions ……>
<message name="Country"> <part name="arg0" type="xsd:string"/></message>
<message name="COVIDCases">
<part name="return" type="xsd:integer"/> </message>
    <portType name=" COVIDAPI">
        <operation name="GetCOVIDCases">
            <input message="tns:Country "/>
            <output message="tns:COVIDCases "/></operation></portType>
……
<service name=" COVIDCasesImplService">
       <portname=" COVIDCasesPort" binding="tns:COVIDCasesPortBinding">
       <soap:address location="http://localhost:8080/covid"/> </port>
</service>
</definitions>
```

- <S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
          <ns2:Country xmlns:ns2="http://COVIDCases/">
              <arg0>USA</arg0>
          </ns2:Country>
      </S:Body>
  </S:Envelope>
- <S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:COVIDCases xmlns:ns2="http://COVIDCases/">
              <return> 4657693</return>
          </ns2:COVIDCases>
      </S:Body>
  </S:Envelope>

有关SOAP描述错误的是（）

- A）SOAP 即 Simple Object Access Protocol 也就是简单对象访问协议。
- B）SOAP 是用于在应用程序之间进行通信的一种通信协议。
- C）SOAP 基于JSON 和 HTTP，其通过JSON 来实现消息描述，然后再通过 HTTP 实现消息传输。
- D）SOAP 协议的一个重要特点是它独立于底层传输机制，Web 服务应用程序可以根据需要选择自己的数据传输协议， 可以在发送消息时来确定相应传输机制。

下列哪些元素不属于SOAP消息

- A）Envelope 元素
- B） Header 元素
- C）Body 元素
- D）Author元素
- E） Fault 元素

有关JSON描述错误的是（）

- A) JSON 指的是 JavaScript 对象表示法
- B）JSON 是存储和交换文本信息的语法。类似 XML。
- C）JSON 比 XML体积稍大，但是更快，更易解析。
- D）JSON 是轻量级的文本数据交换格式

下面哪一个是JSON数据？

- A）　{name:"xiaoming",age,"student"}
- B）　{"name":"xiaoming","age":"student"}
- C）　{"xiaoming","student"}
- D）　["xiaoming","student"]

有关WSDL描述错误的是（）

- A）WSDL 指网络服务描述语言，用于描述网络服务
- B）WSDL 使用 XML 编写
- C）是一种 XML 文档
- D）WSDL 不能用于定位网络服务

下列哪些元素不属于WSDL命名空间（xmlns=http://schemas.xmlsoap.org/wsdl/）

- A）<definitions>
- B）<types>
- C）<message>
- D）<portType>
- E）<function>
- F）<binding>