Android + Ksoap2 + axis2 + tomcat8.x + mysql

1.预备知识

**1.1 关于tomcat和apache**

　　Apache是web服务器（静态解析，如HTML），tomcat是java应用服务器（动态解析，如JSP、PHP）

　　Tomcat只是一个servlet(jsp也翻译成servlet)容器，可以认为是apache的扩展，但是可以独立于apache运行

**联系**

　　1）Apache是普通服务器，本身只支持html即普通网页，可以通过插件支持php，还可以与Tomcat连通(Apache单向连接Tomcat，就是说通过Apache可以访问Tomcat资源，反之不然)。

　　2）Apache只支持静态网页，但像asp、jsp、php、cgi等动态网页就需要Tomcat来处理。

　　3）Apache和Tomcat整合使用：

　　　　如果客户端请求的是静态页面，则只需要Apache服务器响应请求；

　　　　如果客户端请求动态页面，则是Tomcat服务器响应请求，将解析的JSP等网页代码解析后回传给Apache服务器，再经Apache返回给浏览器端。

　　　　这是因为jsp是服务器端解释代码的，Tomcat只做动态代码解析，Apache回传解析好的静态代码，Apache+Tomcat这样整合就可以减少Tomcat的服务开销。

　　4）Apache和Tomcat是独立的，在同一台服务器上可以集成。

**区别**

　　Apache是有C语言实现的，支持各种特性和模块从而来扩展核心功能；Tomcat是Java编写的，更好的支持Servlet和JSP。

　　1、Apache是Web服务器，Web服务器传送(serves)页面使浏览器可以浏览，Web服务器专门处理HTTP请求(request)，但是应用程序服务器是通过很多协议来为应用程序提供 (serves)商业逻辑(business logic)。

　　Tomcat是运行在Apache上的应用服务器，应用程序服务器提供的是客户端应用程序可以调用(call)的方法 (methods)。它只是一个servlet(jsp也翻译成servlet)容器，可以认为是Apache的扩展，但是可以独立于apache运行。

　　2、Apache是普通服务器，本身只支持html静态普通网页。不过可以通过插件支持PHP，还可以与Tomcat连通(单向Apache连接Tomcat,就是说通过Apache可以访问Tomcat资源，反之不然)，Tomcat是jsp/servlet容器，同时也支持HTML、JSP、ASP、PHP、CGI等，其中CGI需要一些手动调试，不过很容易的。

　　3、Apache侧重于http server，Tomcat侧重于servlet引擎，如果以standalone方式运行，功能上Tomcat与apache等效支持JSP，但对静态网页不太理想。

　　4、Apache可以运行一年不重启，稳定性非常好，而Tomcat则不见得。

　　5、首选web服务器是Apache，但Apache解析不了的jsp、servlet才用tomcat。

　　6、Apache是很最开始的页面解析服务，tomcat是后研发出来的，从本质上来说tomcat的功能完全可以替代Apache，但Apache毕竟是tomcat的前辈级人物，并且市场上也有不少人还在用Apache，所以Apache还会继续存在，不会被取代，apache不能解析java的东西，但解析html速度快。

**两者例子**：

　　Apache是一辆车，上面可以装一些东西如html等，但是不能装水，要装水（如servlet jsp）必须要有容器（桶），而这个桶也可以不放在卡车上，那这个桶就是TOMCAT。

**两者整合**：

　　Apache是一个web服务器环境程序，启用他可以作为web服务器使用，不过只支持静态网页，不支持动态网页，如asp、jsp、php、cgi

　　如果要在Apache环境下运行jsp就需要一个解释器来执行jsp网页，而这个jsp解释器就是Tomcat

　　那为什么还要JDK呢？因为jsp需要连接数据库的话就要jdk来提供连接数据库的驱程，所以要运行jsp的web服务器平台就需要APACHE+TOMCAT+JDK

**整合的好处**：

　　如果客户端请求的是静态页面，则只需要Apache服务器响应请求

　　如果客户端请求动态页面，则是Tomcat服务器响应请求

　　因为jsp是服务器端解释代码的，这样整合就可以减少Tomcat的服务开销

从中可以看出，实现一个服务器，最基本的功能就是要能够用FTP或者HTTP等协议，实现对请求的响应并执行任务把结果返回给请求者。他的内部就是一个公司的加工生产部门（负责处理业务逻辑），外部有天线就是一个公司的营销订货部门（负责收发数据）。请求越多，任务就越多，工厂也必然很多。因此，一个服务器也必然是多线程甚至是分布式。

**1.2关于webservice**

**1.2.1------webservice**

Web Service技术， 能使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件， 就可相互交换数据或集成。依据Web Service规范实施的应用之间， 无论它们所使用的语言、 平台或内部协议是什么， 都可以相互交换数据。

Webservice基于SOAP即简单对象访问协议(Simple Object Access Protocol)，它是用于交换XML（标准通用标记语言下的一个子集）编码信息的轻量级协议。

SOAP 通讯协议使用 HTTP 来发送XML 格式的信息。

说简单点就是客户端（android PC桌面应用程序或通过浏览器即C/S或者B/S）和后台服务端（数据库、服务端各种资源等）进行交互的中介，他对外提供很多服务接口（就是可以调用的方法函数、类等）。屏蔽了客户端和后台的直接交互，提高安全性，而且平台无关、语言无关，什么平台（PC android ）上的什么语言（java C C# Html）都可以调用这些服务。而这些服务放在哪里呢，就放在一个webapp里面（webservice可以算作是webapp的一部分）或者直接单独作为一个提供webservice的webapp部署到tomcat或者apache中。

如果说apache或者tomcat是一个公司（含有生产部门和营销部门），那么webservice就像是一个这个公司的服务部门（公司对外或者对内提供什么服务）。

**1.2.2-----HTTP、TCP/IP、SOAP**

网络由下往上分为物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。IP协议对应于网络层，TCP协议对应于传输层，而HTTP协议对应于应用层，三者从本质上来说没有可比性，socket则是对TCP/IP（Transmission Control Protocol / Internet Protocol）协议的封装和应用（程序员层面上）。也可以说，TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据。

顺便说说socket，实际上socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议。实际上，Socket跟TCP/IP协议没有必然的联系。Socket编程接口在设计的时候，就希望也能适应其他的网络协议。所以说，Socket的出现只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，从而形成了我们知道的一些最基本的函数接口，比如create、listen、connect、accept、send、read和write等等。事实上socket是一种方法，就是如何封装TCP/IP的方法，java中已经实现了对TCP/IP的封装，并且有socket这个类，所以可以直接用。

HTTP(Hypertext Transfer Protocol超文本转移协议)是利用TCP在两台电脑(通常是Web服务器和客户端)之间传输信息的协议。客户端使用Web浏览器发起HTTP请求给Web服务器，Web服务器发送被请求的信息给客户端。可以近似的认为他是专门为web设计的，尤其是web浏览器的出现。

SOAP（Simple Object Access Protocol ）简单对象访问协议是在分散或分布式的环境中交换信息的简单的协议，是一个基于XML的协议，它包括四个部分：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=RPC+HTTP+XML

SOAP简单的理解，就是这样的一个开放协议SOAP=RPC+HTTP+XML：采用HTTP作为底层通讯协议；RPC作为一致性的调用途径，ＸＭＬ作为数据传送的格式，允许服务提供者和服务客户经过防火墙在INTERNET进行通讯交互。RPC的描叙可能不大准确，因为SOAP一开始构思就是要实现平台与环境的无关性和独立性，每一个通过网络的远程调用都可以通过SOAP封装起来。SOAP 使用 HTTP 传送 XML，尽管HTTP 不是有效率的通讯协议，而且 XML 还需要额外的文件解析（parse），两者使得交易的速度大大低于其它方案。但是XML 是一个开放、健全、有语义的讯息机制，而 HTTP 是一个广泛又能避免许多关于防火墙的问题，从而使SOAP得到了广泛的应用。但是如果效率对你来说很重要，那么你应该多考虑其它的方式，而不要用 SOAP。

**1.2.2-----web、internet、networks**

## *What is The Internet?*

The [Internet](http://www.webopedia.com/TERM/I/Internet.html) is a massive [network](http://www.webopedia.com/TERM/N/network.html) of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet. Information that travels over the Internet does so via a variety of languages known as [protocols](http://www.webopedia.com/TERM/P/protocol.html).

## *What is The Web (World Wide Web)?*

The [World Wide Web](http://www.webopedia.com/TERM/W/World_Wide_Web.html), or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. Web services, which use HTTP to allow applications to communicate in order to exchange business logic, use the the Web to share information. The Web also utilizes[browsers](http://www.webopedia.com/TERM/B/browser.html), such as [Internet Explorer](http://www.webopedia.com/TERM/I/Internet_Explorer.html) or [Firefox](http://www.webopedia.com/TERM/F/Firefox.html), to access Web documents called [Web pages](http://www.webopedia.com/TERM/W/web_page.html) that are linked to each other via [hyperlinks](http://www.webopedia.com/TERM/H/hyperlink.html). Web documents also contain graphics, sounds, text and video.

The Web is just one of the ways that information can be disseminated over the Internet. The Internet, not the Web, is also used for [e-mail](http://www.webopedia.com/TERM/E/e_mail.html), which relies on [SMTP](http://www.webopedia.com/TERM/S/SMTP.html), [Usenet](http://www.webopedia.com/TERM/U/USENET.html) news groups, [instant messaging](http://www.webopedia.com/TERM/I/instant_messaging.html) and [FTP](http://www.webopedia.com/TERM/F/FTP.html). So the Web is just a portion of the Internet, albeit a large portion, but the two terms are not synonymous and should not be confused.

**Internet算是物理上的一个概念，实现了计算机在在物理层面的相互连接，从而组成互联网，而web万维网，才是我们所说的平时上网的概念。**

**1.3关于AXIS2**

Web Service是现在最适合实现SOAP的技术，而Axis2是实现Web Service的一种技术框架（架构）axis2也可以自己启动一个服务独立运行（官网上axis2.bin版本），也可以作为一个应用程序部署到应用服务器中（官网上axis2.war版本）

我们的系统使用的axis2.war就相当一个可以发布的webapplication（可以将axis2.war直接拷贝到tomcat8.x的wepapps目录中进行部署），然后通过axis2来发布webservice。

注意：axis、xfire、cfx都是编写webservice的框架。

一个webapplication的目录结构，最主要的就是：

/webapplication web应用的根目录，所有的JSP和HTML文件都存放于此目录下

/webapplication/WEB-INF 存放web应用的发布描述文件web.xml

/webapplication/WEB-INF/classes 存放各种class文件，Servlet类文件也放于此目录

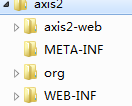
/ webapplication /WEB-INF/lib 存放web应用所需的各种JAR文件，比如可以存放JDBC驱动程序的JAR文件

/ webapplication /otherfoldersOrfiles

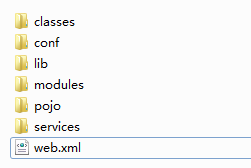
WEB-INF目录里的东西都是对客户端的封闭的，客户端无法通过URL直接访问，是服务端例如tomcat才能访问的。

和WEB-INF同级的目录才是客户端可通过URL访问的，如jsp文件，图片资源等

axis2.war（可以看做一个特殊的webapplication，基于axis2的webservice）的目录结构如下：



WEB-INF子目录



axis2-web中存放的是各种资源，包括图片文件jsp页面等；

WEB-INF是一个重要的目录，以后发布服务都在pojo（不用配置，直接将写好的webservice二进制文件即\*.class文件放入其中即可）或者services（需要service.xml配置文件，里面一般放的是自定义的打包好的aar或jar格式的webservice文件）目录中

**1.4关于Ksoap**

通过WebService可以将不同操作系统平台，不同语言、不同技术整合到一起。PC版本的WebService客户端类库非常丰富，例如，Axis2、CXF等，但这些类库对于Android系统过于庞大，也未必很容易移植到 Android系统上。因此，这些开发包并不在我们考虑的范围内。适合手机的WebService客户端类库也有一些。本例使用了比较常用的 KSOAP2。

在Android SDK中并没有提供调用WebService的库，因此，需要使用第三方类库（KSOAP2）来调用WebService。

**1.5关于XML vs JSON**

“JSON”的全称是“JavaScript Object Notation”，即JavaScript对象符号，是一种轻量级的数据交换格式，易于阅读和编写，同时也易于机器解析和生成。

XMl 全称“xtensible Markup Language可扩展标记语言，标准通用标记语言的子集，一种用于标记电子文件使其具有结构性的标记语言。它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。 它非常适合万维网传输，提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

关于xml和json的比较，网上有很多。<http://blog.csdn.net/slin000/article/details/2441436>

<http://blog.csdn.net/sanpintian/article/details/7347711>

<http://blog.csdn.net/xiazdong/article/details/7059573>

XML和JSON都使用结构化方法来标记数据，下面来做一个简单的比较。

用XML表示中国部分省市数据如下：

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

<country><name>中国</name>

<province><name>黑龙江</name><cities><city>哈尔滨</city><city>大庆</city></cities></province>

<province><name>广东</name><cities><city>广州</city><city>深圳</city><city>珠海</city></cities></province>

<province><name>台湾</name><cities><city>台北</city><city>高雄</city></cities></province>

<province><name>新疆</name><cities><city>乌鲁木齐</city></cities></province>

用JSON表示如下：

　　{name:"中国",</country>

　　province:[{name:"黑龙江",cities:{city:["哈尔滨","大庆"]}},

　　 {name:"广东",cities:{city:["广州","深圳","珠海"]}},

{name:"台湾",cities:{city:["台北","高雄"]}},

{name:"新疆",cities:{city:["乌鲁木齐"]}}]}

　　编码的可读性，xml有明显的优势，毕竟人类的语言更贴近这样的说明结构。json读起来更像一个数据块，读起来就比较费解了。不过，我们读起来费解的语言，恰恰是适合机器阅读，所以通过json的索引.province[0].name就能够读取“黑龙江”这个值。

编码的手写难度来说，xml还是舒服一些，好读当然就好写。不过写出来的字符JSON就明显少很多。去掉空白制表以及换行的话，JSON就是密密麻麻的有用数据，而xml却包含很多重复的标记字符。

注意理解，json （JavaScript Object Notation）和xml（Extensible Markup Language）只是用来传输数据的一种格式，在网络上传输数据基本上最基础的就是字节流，而字符串最容易转换编码成字节流，所以json和xml两种格式实际上都可以看做是一种具有特殊规范的字符串。

要实现网络传输数据，除了那些html 等格式的文本以及文本字符串，还有json、xml。基于各种http、ftp、tcp等协议把要传输的数据（对象、基本数据等等）包装成xml或者json格式，最后再转换成字节流传输或在接收端对xml和json进行解析。

我们用的axis2框架，默认情况下是用xml来传输数据（这个是axis2帮你做好的），如果要想传输json格式数据，需要自己把数据拼接成json格式（其实就是一个符合特定规范的字符串）或者借助第三方类库例如json-lib等对数据进行封装即可。说的具体点就是你的webservice方法里面要传入或返回的参数（复杂数据类型，比如对象）都先封装成字符串（String），不过这个String是一个具有json格式的string。 然后再将json字符串解析成你要的对象或者其他数据类型。其实就是多了一步对数据的封装和解析成json。

使用的Ksoap2使用自己的一个高度抽象类SoapObject来进行数据传输及解析，这个类据说类似于JSONObject。SoapObject完成了对xml的解析和封装。

As for the data returned is a simple data, such as a String or Int value, the returned data the only one SoapObject object, this object can get the value returned

If the return is a complex data container, then get is a tree structure, in which to have multiple SoapObject object, specific operations, see the following code:

if (envelope.getResponse() != null) {

SoapObject result = (SoapObject) envelope.bodyIn;

// result.getProperty("name");// name,?

int count = result.getPropertyCount(); //SoapObject

for(int i=0;i<count;i++){ System.out.println("time",((SoapObject)result.getProperty(i)).getProperty("date"));

System.out.println("title",((SoapObject)result.getProperty(i)).getProperty("theme"));

System.out.println("info",((SoapObject)result.getProperty(i)).getProperty("content"));

}

}

2.开发环境搭建

**2.1搭建webservice开发环境**

1下载Mysql

2下载tomcat8.x并配置环境变量

CATALINA\_HOME: E:\apache-tomcat-8.0.5

Path: %CATALINA\_HOME%\bin;

ClassPath: %CATALINA\_HOME%\lib\servlet-api.jar;

3下载axis2并配置环境变量

Apache Axis2 下载页面：[http://axis.apache.org/axis2/java/core/download.cgi](http://axis.apache.org/axis2/java/core/download.cgi" \t "_blank) (当前最新版本1.6.2)

Apache Axis2 Binary Distribution(1.6.2)：[http://mirror.bjtu.edu.cn/apache//axis/axis2/java/core/1.6.2/axis2-1.6.2-bin.zip](http://mirror.bjtu.edu.cn/apache/axis/axis2/java/core/1.6.2/axis2-1.6.2-bin.zip" \t "_blank)

WAR Distribution：[http://mirror.bjtu.edu.cn/apache//axis/axis2/java/core/1.6.2/axis2-1.6.2-war.zip](http://mirror.bjtu.edu.cn/apache/axis/axis2/java/core/1.6.2/axis2-1.6.2-war.zip" \t "_blank)

Eclipse 插件：

Service Archive Wizard - Eclipse Plug-in(用来将服务代码打包成后缀名为.aar文件的插件)：

<http://www.apache.org/dyn/mirrors/mirrors.cgi/axis/axis2/java/core/1.6.2/axis2-eclipse-service-plugin-1.6.2.zip>

Code Generator Wizard - Eclipse Plug-in(用来将服务代码生成wsdl文件以及解析将wsdl文件生成客户端代码的插件)：

<http://www.apache.org/dyn/mirrors/mirrors.cgi/axis/axis2/java/core/1.6.2/axis2-eclipse-codegen-plugin-1.6.2.zip>

AXIS2\_HOME: C:\Program Files\axis2-1.6.2

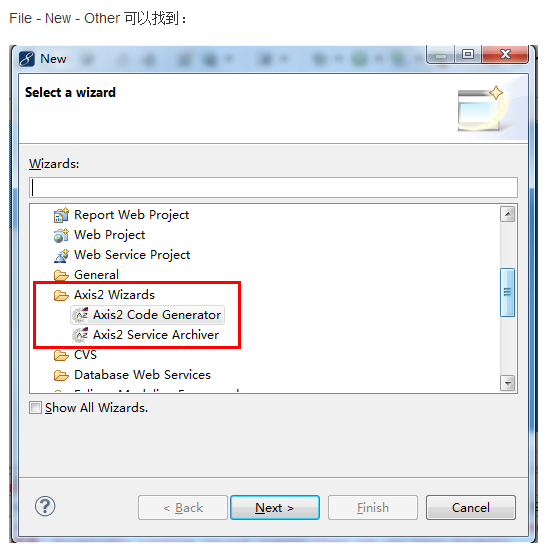
Path:

%AXIS2\_HOME%\bin;

3下载eclipse-jee-luna-R-win32

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/lunar>

4用Eclipse-jee-luna，配置axis2插件。将两个插件解压后放到 %Eclipse安装根目录%\dropins目录下面，启动Eclipse（此步骤可以不做，因为可以不用插件开发）



5最后在Eclipse中Window🡪Preference🡪Java🡪BuildPath—>UserLibraries里面新建一个axis2类库（以后用）,add external jars把C:\Program Files\axis2-1.6.2\lib里面的jar文件都加进来

在Window🡪Preference🡪Server中加入tomcat

**2.2搭建Android开发环境**

1下载ADT adt-bundle-windows-x86-20140702.zip

<http://developer.android.com/sdk/index.html>

2解压后配置环境变量

SDK\_HOME: E:\AndroidDevelop\adt-bundle-windows-x86-20140702\sdk

Path:

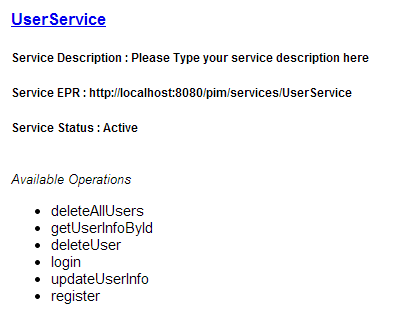
%SDK\_HOME%\tools;%SDK\_HOME%\platform-tools;

3.开发笔记

这里有webservice入门教程

<http://www.blogjava.net/nokiaguy/archive/2009/01/02/249556.html>

这里是webservice发布后通过浏览器访问的界面，里面有服务的描述



这是通过 <http://localhost:8080/pim/services/UserService?wsdl> 访问的xml文档。里面试对可以调用的方法的描述，包括命名空间、方法参数等。



技术细节

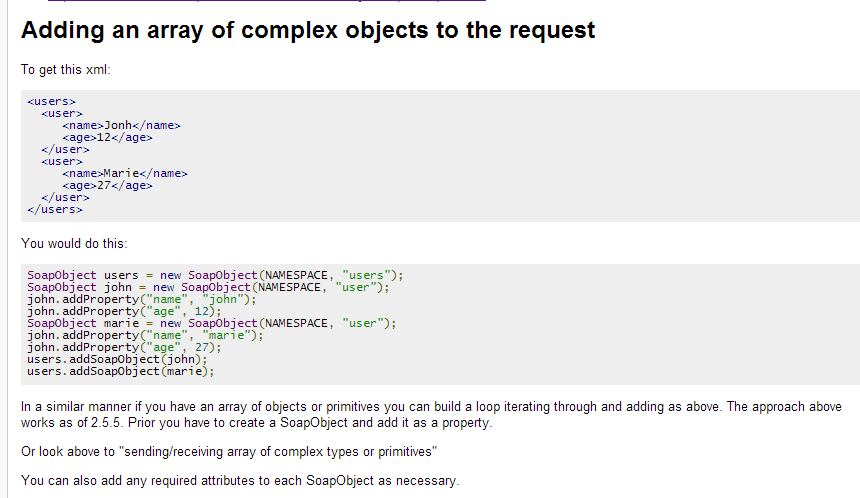
WebService类中包含byte[]类型参数的方法在wsdl2java生成的stub类中对应的数据类型不再是byte[]类型，而是javax.activation.DataHandler。DataHandler类是专门用来映射WebService二进制类型的。

在WebService类中除了可以使用byte[]作为传输二进制的数据类型外，也可以使用javax.activation.DataHandler作为数据类型。 不管是使用byte[]，还是使用javax.activation.DataHandler作为WebService方法的数据类型，使用wsdl2java命令生成的stub类中相应方法的类型都是javax.activation.DataHandler。而象使用.net、delphi生成的stub类的相应方法类型都是byte[]。这是由于javax.activation.DataHandler类是Java特有的，对于其他语言和技术来说，并不认识javax.activation.DataHandler类，因此，也只有使用最原始的byte[]了。

Ksoap（毕竟是轻量级的）不如axis2那么强大，axis2的wsdl2java和java2wsdl工具能够直接生成客户端可以使用的stub类，而且支持各种数据类型（比如自定义的对象数组array of objects，服务端的写法就是随便使用自定义的对象数组）的直接传输，但是Ksoap2不能生成stub类，最蛋疼的就是不支持自定义对象数组等复杂数据类型（比如对象里面还有对象）的直接传输，需要自己序列化并反序列化。

Ksoap2中的这句代码

SoapObject so = (SoapObject)envelope.bodyin;实际上是将调用webservice返回的xml数据文件解析成了SoapObject对象，而至于SoapObject对象，这个对象高度抽象，而且是递归定义



注意编写的代码的编码格式，在windows下面用记事本编写，不用管格式默认utf-8.（不要用notepad++），也可在eclipse里面编写

注意使用wsdl2java命令生成客户端代码的格式样例：**wsdl2java –uri（不是url）**[**http://localhost:8080/pim/services/UserService?wsdl**](http://localhost:8080/pim/services/UserService?wsdl) **-p client –s -o D:\stub**

**p:指定包名 s:生成同步代码 a生成异步代码 o:指定存根路径（注意写一个完整的路径）**

**webservice中的会话管理**

使用单服务会话管理，要在stub类的构造方法（含参数最多的）中加入\_serviceClient.getOptions().setManageSession(true);语句

还要注意服务端的services.xml中配置scope属性值transportsession

使用跨服务会话管理（多个服务之间会话），不需要改变stub类，只需要在services.xml中配置scope属性值application

Request Session Scope

Request session scope is the default session scope in Axis2. When we deploy a service without knowing anything about session management, then our service will be deployed in request session scope; this is much like using no session management at all. The lifetime of this session is limited to method invocation lifetime, or the request processing time.

Once we deploy a service in request session scope, for each individual invocation a new service implementation class will be created. Let's say we have deployed a service called "Foo" in request scope. If a client invokes the service 10 times, there will be 10 instances of the service implementation class.

If we want to specify the scope explicitly, we can still do that by adding scope attribute to service element in services.xml as follows:

<service name="Foo" scope="request"> </service>

To get a better understanding of request scope, create a service using the following service class, deploy it and invoke:

public class MyService implements Lifecycle { public void init(ServiceContext context) throws AxisFault { System.out.println("I'm inside init method "); } public void destroy(ServiceContext context) { System.out.println("I'm inside destroy method"); } public String foo(String foo) { return foo; } }

*Note:* To invoke the service, we simply type the following in the browser:

http://localhost:8080/axis2/services/MyService/foo?foo=foo

Then we will see the following in the server console:

I'm inside init method

*If we continue to do so, we will see this statement printed every time we invoke the service.*

It should be noted here that even if we deploy a service in a request scope, there are many ways of keeping our service as a state full service. One approach is to store the state in Axis2 global run time (ConfigurationContext) and retrieve whenever necessary.

SOAP Session Scope

The purpose of SOAP session is to have a transport-independent way of managing sessions between two SOAP nodes, the client and the server. Here Axis2 uses SOAP headers in order to manage the session. SOAP session scope has a slightly longer lifetime than request session scope; deploying a service in SOAP session is required to change services.xml as well. Managing SOAP session requires both the client and service to be aware of the sessions. This means that the client has to send the session-related data if he wants to access the same session, while the service has to validate the user using session-related data.

In order to manage SOAP session, a client has to send additional reference parameters in the SOAP header, which is named serviceGroupId (this will be sent to the client when he invokes a service that deploys in SOAP session for the first time). In the meantime, SOAP session provides a way to manage sessions across single service invocations, as well as for multiple services within a service group. As long as we are in the same SOAP session, we can manage service-related data in ServiceContext; if we want to share data across other services in the group, we can use ServiceGroupContext to store the session-related data.

When a client tries to access a service that we have deployed in SOAP session for the first time, Axis2 will generate serviceGroupId and send that to the client as a reference parameter in wsa:ReplyTo, as shown below. However, it should be mentioned that the server has to have WS-addressing support for SOAP session support to be available to both the client and the server.

<wsa:ReplyTo> <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address> <wsa:ReferenceParameters> <axis2:ServiceGroupId xmlns:axis2="http://ws.apache.org/namespaces/axis2">urn:uuid:65E9C56F702A398A8B11513011677354</axis2:ServiceGroupId> </wsa:ReferenceParameters> </wsa:ReplyTo>

If a client wants to live in the same session, then he has to copy that reference parameter and send it back to the server when he invokes the service in the second instance. As long as a client sends the valid serviceGroupId, he can use the same session and the service can maintain the session-related data. Unlike request session, SOAP session has a default timeout period; if the client does not touch the service for a period of 30 seconds, the session will expire. In this situation, a client that sends the old serviceGroupId will get an AxisFault. We can change the default timeout period by changing the server's axis2.xml as follows:

<parameter name="ConfigContextTimeoutInterval">30000</parameter>

*By changing the parameter value, we can obtain the desired timeout interval.*

As mentioned earlier, deploying a service in SOAP session requires us to change services.xml as follows:

<service name="MyService" scope="soapsession"> </service>

It should be noted here that, although we are required to copy the references parameter, once we use the Axis2 client we can configure it to copy the parameter automatically.

To get an idea about SOAP session management, let's write the following service class and deploy it. The service class adds the current passed value to the previous value and sends the result, so we should see the incremental values changing. First, write the service and deploy it in SOAP session, completing the necessary changes to services.xml

public class MyService { public int add(int value) { MessageContext messageContext = MessageContext.getCurrentMessageContext(); ServiceContext sc = messageContext.getServiceContext(); Object previousValue = sc.getProperty("VALUE"); int previousIntValue = 0; if (previousValue != null) { previousIntValue = Integer.parseInt((String)previousValue); } int currentValue = previousIntValue + value; sc.setProperty("VALUE","" + currentValue); return currentValue; } }

Now let's use the following code to invoke the service. As you can see, we have engaged the addressing module, but we have not done anything to manage the sessions.

ServiceClient sc = new ServiceClient(); sc.engageModule("addressing"); Options opts = new Options(); opts.setTo(new EndpointReference( "http://127.0.0.1:8080/axis2/services/MyService")); opts.setAction("urn:add"); sc.setOptions(opts); OMElement ele = sc.sendReceive(createPayLoad(10)); System.out.println(ele.getFirstElement().getText()); ele = sc.sendReceive(createPayLoad(10)); System.out.println(ele.getFirstElement().getText());

**createPayLoad method is shown below:**

public static OMElement createPayLoad(int intValue) { OMFactory fac = OMAbstractFactory.getOMFactory(); OMNamespace omNs = fac.createOMNamespace( "http://ws.apache.org/axis2", "ns1"); OMElement method = fac.createOMElement("add", omNs); OMElement value = fac.createOMElement("args", omNs); value.setText("" + intValue); method.addChild(value); return method; }

Once we run the code, we will see the following output in the client side:

10

10

That means that although we have invoked the service twice, we have gotten the same output. Now let's change our client code a bit and see what results we get. Just add the following line of code and run the client again.

opts.setManageSession(true); sc.setOptions(opts);

Now you should see the following output in the client side.

10

20

So that simply tells us that we have invoked the service in a session-aware manner.

*It should be mentioned that you will need to engage the addressing module in both the client side and the server side in order to get SOAP session working.*

Transport Session Scope

In the case of Transport session, Axis2 uses transport-related session management techniques to manage sessions. For example, in the case of HTTP, HTTP cookies are used to manage session. The lifetime of the session is controlled by the transport, not by Axis2; Axis2 simply stores service context and ServiceGroupContext in the transport session object so that service can access those contexts as long as session lives.

One of the key advantages of Transport session over other sessions is that we can talk to multiple service groups within one transport session. In SOAP session, we don’t have a way to communicate between two service groups, but with the transport session we have that capability. In this case, the number of service instances created depends on the number of transport sessions created.

Deploying a service in transport session requires us to change services.xml as follows:

<service name="MyService" scope="transportsession"> </service>

Now, let's change our previous sample to have the scope as transport and redeploy the service. Let's also try to invoke the service in the following ways:

***Option 1 : Using the browser.***

http://localhost:8000/axis2/services/MyService/add?value=10

If we keep on typing that, then we will get the output as **10 , 20 , 30** etc.

***Option 2 : Using the service client. To use the service client, we must first set the session management flag to true so that it will send back the transport cookie. So if we run the following code, we will see this outcome:***

ServiceClient sc = new ServiceClient(); Options opts = new Options(); opts.setTo(new EndpointReference( "http://127.0.0.1:8000/axis2/services/MyService")); opts.setAction("urn:add"); opts.setManageSession(true); sc.setOptions(opts); OMElement ele = sc.sendReceive(createPayLoad(10)); System.out.println(ele.getFirstElement().getText()); ele = sc.sendReceive(createPayLoad(10)); System.out.println(ele.getFirstElement().getText());

10

20

This simply tells us that we have invoked the service deployed in the transport session in a session-aware manner. In this case we do not need to have addressing module.

Application scope

Application scope has the longest lifetime compared to others; the lifetime of the application session is equal to the lifetime of the system. If we deploy a service in application scope there will be only one instance of that service, and there will be only one service context for that service. In the Axis2 world, if we consider the memory footprint and if we do not want to manage session, then a good option is to deploy the service in application scope.

When we deploy a service in application scope, a client does not need to send any additional data to use the same session.

To deploy a service in application scope we need to change axis2.xml as shown below:

<service name="foo" scope="application"> </service>

Managing Sessions Using ServiceClient

As we know by now, managing sessions in the client side involves bit of a work. In both SOAP session and Transport session, a client has to send the session-related data if he wants to live in the same session. He might be able to do this for SOAP session by copying the required reference parameters. However, with Transport session, a user cannot obtain access to transport to copy and send cookies.

To make the user experience easier, Axis2 has the built-in capability of managing sessions within the client session by just setting a flag (which we have used already). Then, depending on the service side session, it will send the corresponding data as long as we use the same service client.

If we want to live in the same session then we can create a service client as shown below, then reuse the created service client object to invoke the service.

Options options = new Options(); **options.setManageSession(true);** ServiceClient sender = new ServiceClient(); sender.setOptions(options);

Once we create ServiceCient as shown above, if the service is deployed in SOAP session it will copy the serviceGroupId and send that from the second invocation onwards. If the server sends the session id, such as http cookies, it will copy that to ServiceContext (in the client side) and send it back to server when the client invokes the service for the second time.

Conclusion

Stateless nature is one of the main characteristics of Web services, but that is a limitation for advanced web services developers. Developing an enterprise level application using Web services is not easy unless we have a session management layer. Axis2 has four levels of sessions to address enterprise Web service development issues.