**一、WSSecurity简述**

安全的Web服务是Web服务成功的必要保证。但大家都知道，Web服务使用XML来进行数据交换，而XML在默认情况下是明文编码的；同时，大部分Web服务使用HTTP协议作为传输协议，同样，HTTP也是使用明文方式来传输数据的。这就造成了在不加密的传输协议上传输不加密的信息，从而使信息传输的保密性受到威胁。作为企业级的应用，以上的方式不能满足安全性基本要求：

²  数据在internet上传输的时侯是不应该被第三方能够看到的；

²  双方必须能够验定彼此间的来源；

²  双方必须能够确定被传送的数据没有被在中途中遭到黑客的修改。

通过使用SSL协议我们可以解决第一个问题即："不应该被第三方看到"；使用数字签名和数字证书可以解决后面的两个问题。当使用数字证书方法时，Web 服务请求者必须有一个由可信认证中心签署的数字证书。请求者使用这个证书来表明它们的身份，并对 SOAP 消息进行数字签名。对方系统接收到消息后，就可对消息做时间戳记并进行日志记录。此时，数字签名会得到验证。验证过程将确保消息来自发送方，并且还要验证消息内容在传输过程中没有被篡改。

IBM、Microsoft 和 Verisign 于2002年十二月份联合发布了一个关于 Web 服务安全性（Web Services Security，WS-Security）的规范，该规范描述如何向 SOAP 消息附加签名和加密报头；另外，它还描述如何向消息附加安全性令牌（包括二进制安全性令牌，如 X.509 证书），提供了一套帮助 Web 服务开发者保护 SOAP 消息交换的机制。

根据应用的对安全要求的级别不同，可以采用不同的方式来实现安全性，以下是目前最常用的一些实现方式（从低到高排列）：

²  J2EE Web应用默认的访问控制（数据是明文的）；

²  使用axis的Handler进行访问控制（数据是明文的）；

²  使用Servlet过滤器（Filter）进行访问控制（数据是明文的）；

²  使用SSL/HTTPS协议来传输（加密的数据传输协议）；

²  使用WS-Security规范对信息进行加密与身份认证（数据被加密传输）。

前三种方式对于安全级别要求不高的应用是可行的，它能够使用Web应用访问认证机制来进行权限验证，从而保护对资源的访问。但需要注意的是，虽然它们进行了身份验证，但信息的传递还是以明文的方式进行的，不能保证信息在传输过程中不被窃取。SSL是一个安全的传输协议，使用它传输Web服务能保证信息不被第三方窃取。但它有个缺点就是对系统资源消耗大。采用最后一种方式，信息被签名后再加密，然后把加密后的信息网络上传播，这样，即使第三方获得加密后的传输信息，也不能解密。对于安全级别要求高的系统，应该采用WS-Security规范来作为Web服务安全性解决方案。

**二、基于https通信并且使用用户名密码来验证的WS**

在一般的应用中，我们可以通过https来保护我们传输的明文数据。

关键在于我们需要来验证这个客户端过来的请求，即需要具有基本的用户名，密码才能访问我的Web Service，我们称之为Basic Auth。

**2.1 错误做法**

在很多项目中，有些开发队伍为了图省事，客户对环境的掌控也不好，为了验证一个webservice，我们往往会采用以下这样的验证手法：

**第一种：**

|  |
| --- |
| <http://xxxx.xxx.xxx/abc.wsdl?username>=验证个头&password=验证个头 |

服务端拿到这个url把username，password用request.getParameter出来后，和数据库一匹配，验证。

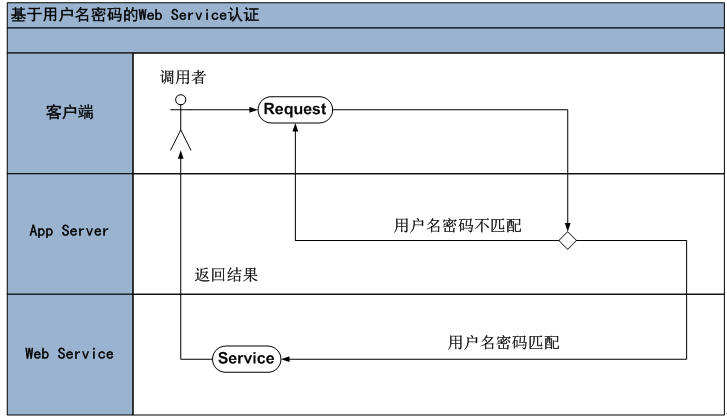
**第二种：**

|  |
| --- |
| <Request xmlns="http://10.225.106.35">      <username>验证个头啊</username>      <password>不要老是你个头你个头</password>      <BusinessData>2007-01-01</BusinessData>  </ Response > |

服务端拿到后把这个soap request body中的<username>和<password>拿出来后和数据库一匹配，又验证了！

这两种做法，无疑是掩耳盗铃！！！（***不要和我说业务实现是最主要的，等你的数据哪天没了，厂长经理的工资被篡改了，如果你愿意被客户做成东方不败，那你尽管去这样做就好了。***）

**2.2 正确的做法**

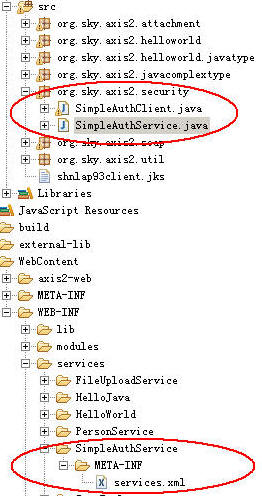


通过上图我们可以看到，如果你的用户名和密码和服务端预设的用户名密码如果不匹配，你的“调用”，根本到达不了具体的Web Service，直接在Web Server端已经被打回来了，即你连wsdl都到达不了。

**一、实际例子**

**3.1 Service端**

我们编写一个Service端



**org.sky.axis2.security.SimpleAuthService**

|  |
| --- |
| package org.sky.axis2.security;  public class SimpleAuthService {           public double getTax(double salary) {                     // System.out.println("input salary=====" + salary);                     if (salary > 10000) {                              return 2000;                     } else if (salary > 1000 && salary <= 10000) {                              return 200;                     } else {                              return 0;                     }           }  } |

service.xml文件的内容

|  |
| --- |
| <service name="SimpleAuthService">           <Description>                     Please Type your service description here           </Description>           <parameter name="ServiceClass" locked="false">org.sky.axis2.security.SimpleAuthService           </parameter>           <messageReceivers>                     <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"                              class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />           </messageReceivers>           <actionMapping>urn:getTax</actionMapping>  </service> |

最重要的来了

**修改web.xml文件**，增加以下的内容

|  |
| --- |
| <security-constraint>                     <web-resource-collection>                              <web-resource-name>Simple Authenticate Web service</web-resource-name>                              <url-pattern>/services/SimpleAuthService</url-pattern>                     </web-resource-collection>                     <auth-constraint>                              <role-name>**bank\_member**</role-name>                     </auth-constraint>           </security-constraint>           <login-config>                     <auth-method>**BASIC**</auth-method>                     <realm-name>Axis Basic Authentication Area</realm-name>           </login-config>           <security-role>                     <role-name>**bank\_member**</role-name>           </security-role> |

我们可以看到：

l   只有在服务端属于bank\_member角色（组）中的人员才被允许访问该web service即：SimplAuthService。

l   而且，该该访问采用“BASIC”模式，即需要用户名和密码来进行访问。

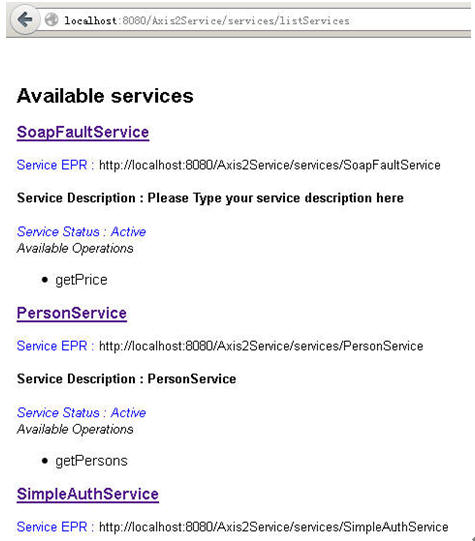
随后，我们打开tomcat所在目录下的conf目录下的tomcat-users.xml如我的是“D:\tomcat\conf\tomcat-users.xml”。

在文件中加入如下内容（注意**红色加粗**的部分）：

|  |
| --- |
| <?xml version='1.0' encoding='utf-8'?>  <tomcat-users>    <role rolename="manager"/>  **<role rolename="bank\_member"/>**    <role rolename="admin"/>    <role rolename="sales"/>    <role rolename="participant"/>    <user username="xxx" password="xxx" roles="admin,manager,participant,sales"/>  **<user username="Wright" password="abcdefg" roles="bank\_member"/>**  </tomcat-users> |

然后我们来布署我们的web service。

布署后我们启动我们的tomcat，访问：<http://localhost:8080/Axis2Service/services/listServices>



我们来点这个SimpleauthService，然后我们可以看到我们的浏览器弹出一个对话框



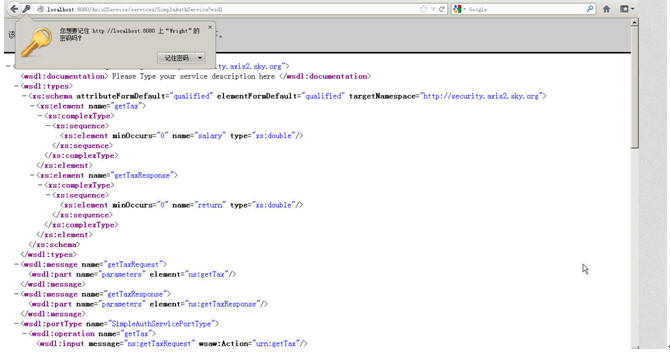
我们输入刚才在tomcat的tomcat-users.xml中定义的Wright这个用户，即

用户名：Wright

密码: abcdefg

**注意大小写要区分啊！**

然后得到wsdl的输出：



如果我们在用户名和密码处输错一个字符，我们将会被迫停留在此对话框上，而得不到我们相要的wsdl的正确输出：



然后我们试着点[取消]按钮，我们将得到

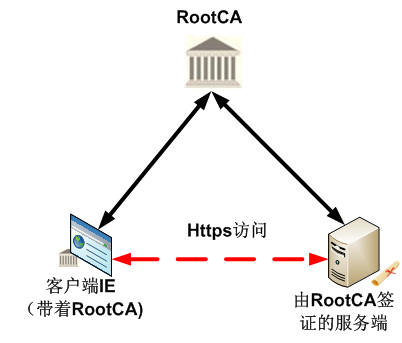


**3.2 制作client端前的准备**

我们前面说过了，我们需要使用https来保护我们传输的用户名和密码，即Wright/abcdefg这个认证。

因此，我们需要实现一个单向的https。

还记得我们在第二天“**apache tomcat https应用**”中所说的那个服务端与客户端与CA之间的互信关系是如何构成的吗？我们来重温一下，看下面这个图：



1）  由于服务端是我由我们的CA即RootCA签发的；

2）  而我们的客户端的根信任域内装着我们的RootCA这张证书；

3）  因此当我们的客户端去访问我们的服务端时，客户端client和服务端之间搭成了“互信”；

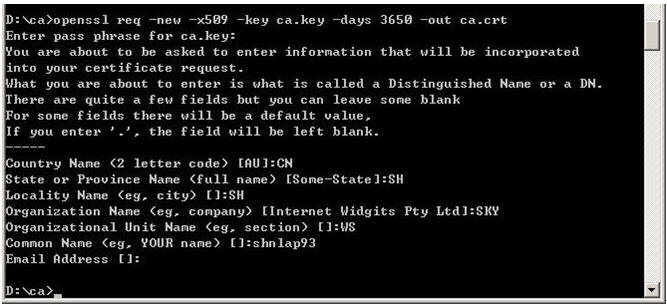
我们来重温一下这个环境搭建的过程。

**3.2.1 制作CA**

**1）先产生KEY**

|  |
| --- |
| openssl genrsa -des3 -out shnlap93.key 1024 |

**2）通过key产生ca证书**



我们生成了一个有效日期为3650天（10年）的RootCA。

**注意：**

如果不加这个-days参数，默认产生的证书文件的有效期为30天，即一个月的有效期。

**3.2.2 制作tomcat的证书即jks格式文件**

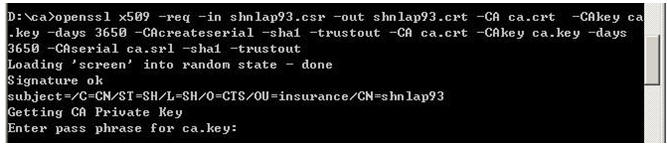
**1）产生shnlap93.jks文件**

|  |
| --- |
| keytool -genkey -alias shnlap93X509 -keyalg RSA -keysize 1024 -dname "CN=shnlap93, OU=insurance, O=CTS, L=SH, S=SH, C=CN" -keypass aaaaaa -keystore shnlap93.jks -storepass aaaaaa |

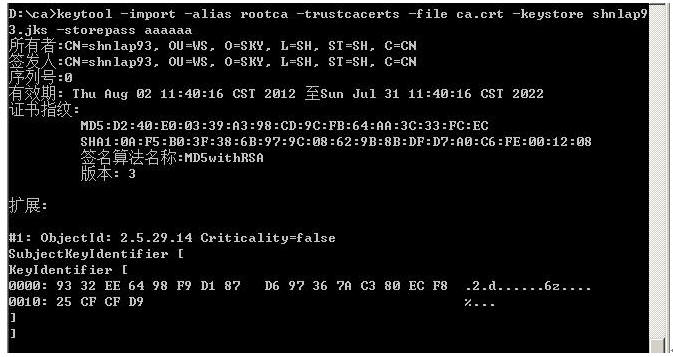
**2）通过JKS产生csr（证书请求）文件**

http://my.csdn.net/uploads/201208/02/1343895729_2739.jpg

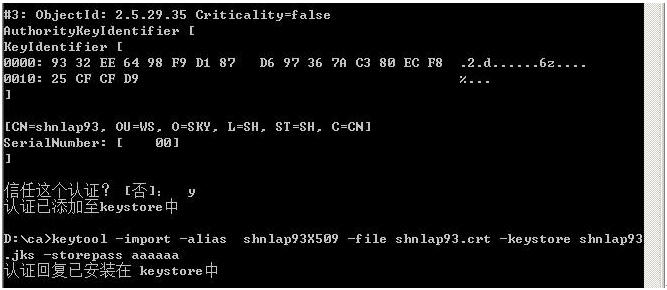
**3）使用我们的RootCA签名该csr文件并生成crt（证书文件）**



**4）   将RootCA作为“信任域”即trustcacerts导入原来的jks文件**



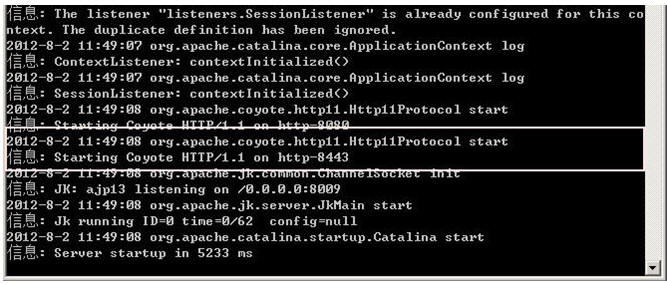
**5）   将被签名后的证书文件导入原来的jks文件**



这样，我们勾成了上述的“三角形”互信关系，就可以把这个shnlap93.jks文件扔给tomcat，然后修改tomcat的conf目录下的server.xml文件。

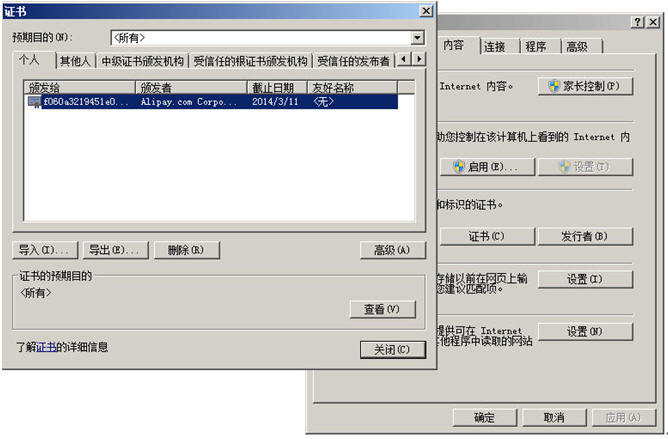
|  |
| --- |
| <Connector executor="tomcatThreadPool"                 port="8443" protocol="HTTP/1.1"                 connectionTimeout="20000"                 secure="true" SSLEnabled="true"                 clientAuth="false" sslProtocol="TLS"                 keystoreFile="d:/tomcat/conf/shnlap93.jks" keystorePass="aaaaaa"  /> |

随后我们启动tomcat。



如果得到上面截图中白色方框标出的输出的话则代表我们的tomcat已经以https方式在运行了。

**3.2.3 将RootCA导入IE的根证书列表中去**

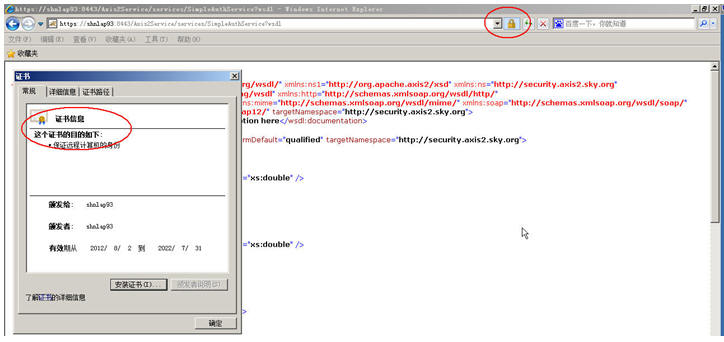


按[导入]，选择我们的ca.crt文件。

然后我们在IE中打入：<https://shnlap93:8443/Axis2Service/services/SimpleAuthService?wsdl>

由于是https绑定证书文件中的CN(Common Name)，因此此时我们必须使用“主机名”来代替原来的IP地址来访问。

在弹出的需要输入用户名和密码的对话框中输入”Wright/abcdefg”，我们即可得到如下的输出：



我们可以看到，该证书是有效证书，而不会弹出一个“你是否信任”一类的对话框再次让你确认是否相信该https连接了。其原因就在于

**因为我们在我们的IE浏览器中已经建立起下述这样的一个三角互信关系来了：**

1）  由于服务端是我由我们的CA即RootCA签发的；

2）  而我们的客户端的根信任域内装着我们的RootCA这张证书；

3） 因此当我们的客户端去访问我们的服务端时，客户端client和服务端之间搭成了“互信”；

**3.3 需要为我们的Axis2的调用客户端也建立起https中的互信**

上面是我们打开一个IE后访问https的链接所需要的步骤，现在我们这样来想：

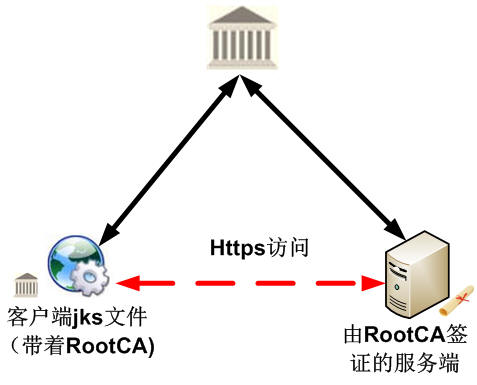
l   我们现在将要运行的是一个Java应用程序；

l   该应用程序将通过https这样的链接来访问我们的tomcat中的webservice；

**而不再是一个IE来访问我们的web service喽！！！**

那么，我们应该为我们的Java应用程序，也配置一个上述这样的三角信任关系，对不对？

先来看下面这个示例图，和IE端配置信任关系稍稍有点不一样

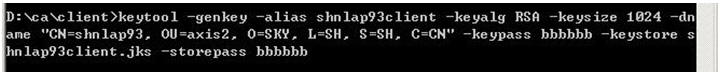


由于我们的是一个Java应用程序，因此我们的应用程序需要带着一个jks文件，我们把它称为client.jks吧。

**这个client.jks文件的trustcacerts域里**，**只要带有RootCA的信息**，**是不是这个client就可以在访问我们的服务器时，和我们的服务器也建立起一个三角互信关系了？**

我们来动手吧，只需要两步即可。

**1）   建立客户端所需的JKS文件**



这边的密码，我用的是六个b，和服务端的证书的密码区分开来。

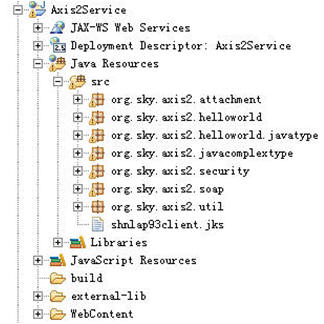
**1）   将RootCA作为“信任域”即trustcacerts导入客户端的jks文件**

|  |
| --- |
| **keytool -import -alias rootca -trustcacerts -file ca.crt -keystore shnlap93client.jks -storepass bbbbbb** |

这样，这个客户端的jks文件我们就可以给我们的axis2的客户端用了。

**3.4 调用客户端**

先将shnlap93client.jks放置在我们工程的src目录，使得这个jks文件会被自动编译到classpath下。



**org.sky.axis2.security.SimpleAuthClient**

|  |
| --- |
| package org.sky.axis2.security;  import java.net.URL;  import java.util.ArrayList;  import java.util.List;      import javax.xml.namespace.QName;    import org.apache.axiom.om.OMAbstractFactory;  import org.apache.axiom.om.OMElement;  import org.apache.axiom.om.OMNamespace;  import org.apache.axiom.soap.SOAPBody;  import org.apache.axiom.soap.SOAPEnvelope;  import org.apache.axiom.soap.SOAPFactory;  import org.apache.axis2.addressing.EndpointReference;  import org.apache.axis2.client.OperationClient;  import org.apache.axis2.client.Options;  import org.apache.axis2.client.ServiceClient;  import org.apache.axis2.context.MessageContext;  import org.apache.axis2.transport.http.HTTPConstants;  import org.apache.axis2.transport.http.HttpTransportProperties.Authenticator;  import org.apache.axis2.wsdl.WSDLConstants;    public class SimpleAuthClient {           private static EndpointReference targetEPR = new EndpointReference(                              "**https://shnlap93:8443/Axis2Service/services/SimpleAuthService**");  **private final static String usr = "Wright";**  **private final static String pwd = "abcdefg";**  **private final static String jksFile = "shnlap93client.jks";**  **private final static String jksFilePWD = "bbbbbb";**             private String getJskFilePath() {                     String filePath = "";                     ClassLoader classLoader = Thread.currentThread()                                       .getContextClassLoader();                     URL url = classLoader.getResource(jksFile);                     if (url == null) {                              classLoader = ClassLoader.getSystemClassLoader();                              url = classLoader.getResource(jksFile);                     }                     filePath = url.getPath();                     System.out.println(filePath);                     return filePath;             }             public void getTax() {    **System.setProperty("javax.net.ssl.trustStore", getJskFilePath());// //**  **System.setProperty("javax.net.ssl.trustStorePassword", jksFilePWD);**                     ServiceClient sender = null;  **Authenticator authenticator = new Authenticator();**  **List<String> auth = new ArrayList<String>();**  **auth.add(Authenticator.BASIC);**  **authenticator.setAuthSchemes(auth);**  **authenticator.setUsername(usr);**  **authenticator.setPassword(pwd);**  **authenticator.setPreemptiveAuthentication(true);**                     Options options = new Options();                     options.setTo(targetEPR);                     options.setAction("urn:getTax");  **options.setProperty(HTTPConstants.AUTHENTICATE, authenticator);**                     try {                              sender = new ServiceClient();                              sender.setOptions(options);                              OperationClient mepClient = sender                                                 .createClient(ServiceClient.ANON\_OUT\_IN\_OP);                              MessageContext mc = new MessageContext();                              SOAPFactory fac = OMAbstractFactory.getSOAP11Factory();                              SOAPEnvelope env = fac.getDefaultEnvelope();                              OMNamespace omNs = fac.createOMNamespace(                                                 "http://security.axis2.sky.org", "");                              OMElement getTax = fac.createOMElement("getTax", omNs);                              OMElement salaryEle = fac.createOMElement("salary", omNs);                              salaryEle.setText("2100");                              getTax.addChild(salaryEle);                              env.getBody().addChild(getTax);                              mc.setEnvelope(env);                              mepClient.addMessageContext(mc);                              System.out.println("message====" + env);                              mepClient.execute(true);                              MessageContext response = mepClient                                                 .getMessageContext(WSDLConstants.MESSAGE\_LABEL\_IN\_VALUE);                              SOAPBody body = response.getEnvelope().getBody();                              OMElement element = body.getFirstElement().getFirstChildWithName(                                                 new QName("http://security.axis2.sky.org", "return"));                              System.out.println(element.getText());                     } catch (Exception e) {                              e.printStackTrace();                     } finally {                              if (sender != null)                                       try {                                                 sender.cleanup();                                       } catch (Exception e) {                                       }                     }             }             public static void main(String[] args) {                     SimpleAuthClient client = new SimpleAuthClient();                     client.getTax();           }    } |

**注意红色标粗的部分，由其是：**

|  |
| --- |
| **System.setProperty("javax.net.ssl.trustStore", getJskFilePath());// //**  **System.setProperty("javax.net.ssl.trustStorePassword", jksFilePWD);** |

由于https需要通过client的jks与server的jks建立起三角互信关系，因此我们需要通过程序去访问这个jks文件，访问这个jks文件我们需要知道：

1)      该jks所在路径；

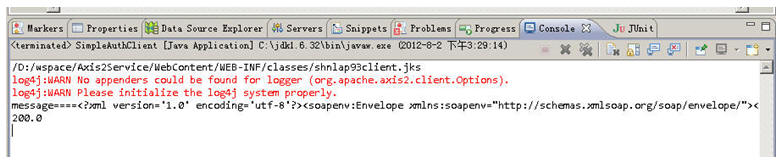
2)      该jks的密码（六个b）；

对吧？

|  |
| --- |
| **Authenticator authenticator = new Authenticator();**  **List<String> auth = new ArrayList<String>();**  **auth.add(Authenticator.BASIC);**  **authenticator.setAuthSchemes(auth);**  **authenticator.setUsername(usr);**  **authenticator.setPassword(pwd);**  **authenticator.setPreemptiveAuthentication(true);**  **options.setProperty(HTTPConstants.AUTHENTICATE, authenticator);** |

上述代码做的事就是把“Wright/abcdefg”这对用户名及密码，set到一个Authenticator对象中去，然后将该对象与需要访问的soap request进行绑定。

然后我们来看运行结果：

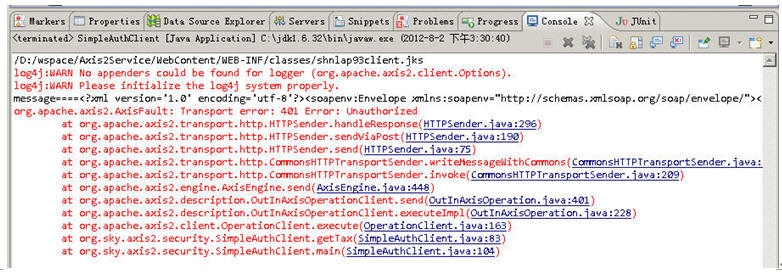


我们把：

**privatefinal static String usr = "Wright";**

**privatefinal static String pwd = "abcdefg";**

中的任何一个值改动一下比如说我们把usr改成**”abc”**，然后再来看运行结果。



注意到这个org.apache.axis2.AxisFault: Transport error: 401 Error: Unauthorized

了吗？

再来比较我们在浏览器中的当弹出要求输入的用户名和密码的对话框时，我们点[取消]按钮得到的输出：



明白了吧？

这就是基于用户名密码且通过https来访问web service的详细过程。

但是这种方法的缺点是，用户名和密码还是以明文方式传输，且用户名和密码是预先配置在web server端的。

以后我们会将用户名密码以加密形式传输且是动态从数据库中获取的web service的认证。

结束今天教程！！！