|  |
| --- |
| **HttpClient连接SSL** |
| 构建安全传输通道 |
|  |
| [企业应用系统之间的交互式必不可少的,一个安全的交互通道是保证信息安全的基础,本文简单地介绍怎么搭建一个SSL服务,并利用httpClient对其进行访问] |
|  |
| **wanghetommy@163.com** |
| **2011-3-1** |
|  |

说明:本文只提供httpClient访问https简单的示例,为需要者方面的人做参考.并未做深入的分析,欢迎致mail讨论;其相关基础知识网上一大篓筐;

开发工具:myEclipse

测试环境:Tomcat + windows +HttpClient4.1

1. 生成**KeyStore**

**打开cmd，输入(jdk环境变量当然是配置好的):**

keytool -genkey -alias tomcat -keyalg RSA –validity 60 -keystore D:\tomcat.keystore

**cmd输出:**  
输入keystore密码：\*\*\*\*\*\*  
您的名字与姓氏是什么？  
[Unknown]： localhost  
您的组织单位名称是什么？  
[Unknown]： it  
您的组织名称是什么？  
[Unknown]： dev  
您所在的城市或区域名称是什么？  
[Unknown]： bj  
您所在的州或省份名称是什么？  
[Unknown]： bj  
该单位的两字母国家代码是什么  
[Unknown]： CN  
CN=localhost, OU= it, O= dev, L=bj, ST=bj, C=CN 正确吗？  
[否]： Y

输入的主密码（如果和 keystore 密码相同，按回车）：\*\*\*\*\*\*\*

参数说明:

**-genkey**表示生成密钥   
**-validity**指定证书有效期，这里是**60**天   
**-alias**指定别名，这里是**tomcat**  
**-keyalg**指定算法，这里是**RSA**   
**-keystore**指定存储位置，这里是**D:\ tomcat.keystore**

使用的自定义密码为 **123456**

Keytool 详细命令说明请参考百度百科;

\*其中 您的名字与姓氏是什么？ localhost是网站的域名或者ip,根据实际情况填写。否者会出现证书上的名称无效;

1. 配置tomcat服务器支持SSL
2. 将生成的 tomcat.keystore文件，放到%TOMCAT\_HOME%/conf目录中(其他也OK).

修改%TOMCAT\_HOME%/conf/server.xml,新增Connector(原注释里有):

<Connector

SSLEnabled="true"

URIEncoding="UTF-8"

clientAuth="false"

keystoreFile="conf/tomcat.keystore"

keystorePass="123456"

maxThreads="150"

port="8443"

protocol="HTTP/1.1"

scheme="https"

secure="true"

sslProtocol="TLS" />

这样你的tomcat就支持https访问了;

**属性说明(来源网上资源)：**

**port**： 这个port属性(默认值是8443)是 TCP/IP端口数码，Tomcat在其上监听安全连接。你可以把它更改成任何你愿意要的数值(如默认的https通信，数目是443)。不过，在许多操作系统中，要想在比1024小的端口数码上运行Tomcat，需要特殊的设置(它超出了这个文档资料的范围)。  
**redirectPort**： 如果你在这里更改端口数值，你还必须更改在non-SSL连接器上的redirectPort 这个属性特定的值。这允许Tomcat自动地redirect那些试图访问有安全限制页面的用户，指明根据 Servlet 2.4 Specification要求，SSL是必需的  
**clientAuth**： 如果你想要Tomcat要求所有的SSL客户在使用这个socket时出示用户认证书，把这个值设定为 true 。如果你想要Tomcat要求出示用户认证书，但是如果没有认证书也可以， 就把这个值设定为want 。  
**keystoreFile**： 如果你产生的keystore文件不在Tomcat期望的默认地方(一个叫做.keystore 的文件在Tomcat运行的主目录)，就添加这个属性。你可以指定一个绝对路径名称， 或者一个由$CATALINA\_BASE环境变量而派生的相对路径名称。  
**keystorePass**： 如果你使用一个不同的keystore(以及认证书)密码，而不是Tomcat期望的密码 (就是changeit)，添加这个元素。  
**keystoreType**： 如果使用一个PKCS12 keystore的话，就添加这个element。 有效的值是JKS 和 PKCS12  
**sslProtocol**： 要在这个socket上被使用的加密／解密协定。如果你在使用Sun的JVM，我们不提倡更改 这个值。据报道，TLS协定的IBM's 1.4.1 实现与一些通用的浏览器不兼容。 如果是这样，就使用value SSL  
   
**ciphers**： 这个socket允许使用的由逗号分隔开的加密密码列单。默认的情况下，任何可用的密码都允许被使用。  
**algorithm**： 可用的X509算法。默认是Sun的实现( SunX509 )。 对于IBM JVMs，你应该使用值 IbmX509。对于其他卖主，查阅JVM文档资料来 找正确的值。  
**truststoreFile**： 用来验证用户认证书的TrustStore文件。  
**truststorePass**： 访问TrustStore的密码。默认值就是keystorePass的值。  
**truststoreType**： 如果你在使用与KeyStore不同格式的TrustStore，添加这个元素。 合法的值是JKS和PKCS12  
**keyAlias**： 如果 keystore 里面有多个 key，你可以为用这个选项为加入的 key 起一个名字。 如果没有指定名字，使用时 keystore 内的第一个 key 将会被使用

1. 用浏览器访问你的应用

输入:https://localhost:8443/myDemo

你会发现:

ssl.jpg

你的应用已经处于SLL安全通道中了.

1. 用httpClient访问https

利用官方的一个例子来说明:

**public** **class** ClientCustomSSL {

**public** **final** **static** **void** main(String[] args) **throws** Exception {

DefaultHttpClient httpclient = **new** DefaultHttpClient();

**try** {

KeyStore trustStore = KeyStore.*getInstance*(KeyStore.*getDefaultType*());

FileInputStream instream =

**new** FileInputStream(**new** File("d:\\tomcat.keystore"));

**try** {

//加载keyStore d:\\tomcat.keystore

trustStore.load(instream, "123456".toCharArray());

} **finally** {

**try** { instream.close(); } **catch** (Exception ignore) {}

}

//穿件Socket工厂,将trustStore注入

SSLSocketFactory socketFactory = **new** SSLSocketFactory(trustStore);

//创建Scheme

Scheme sch = **new** Scheme("https", 8443, socketFactory);

//注册Scheme

httpclient.getConnectionManager().getSchemeRegistry().register(sch);

//创建http请求(get方式)

HttpGet httpget =

**new** HttpGet("https://localhost:8443/myDemo/Ajax/serivceJ.action");

System.*out*.println("executing request" + httpget.getRequestLine());

HttpResponse response = httpclient.execute(httpget);

HttpEntity entity = response.getEntity();

System.*out*.println("----------------------------------------");

System.*out*.println(response.getStatusLine());

**if** (entity != **null**) {

System.*out*.println("Response content length: " +

entity.getContentLength());

String ss = EntityUtils.toString(entity);

System.out.println(ss);

EntityUtils.*consume*(entity);

}

} **finally** {

httpclient.getConnectionManager().shutdown();

}

}

}

运行程序:

executing requestGET https://localhost:8443/myDemo/Ajax/serivceJ.action HTTP/1.1

----------------------------------------

HTTP/1.1 200 OK

Response content length: 12

hello world!

服务器端的action方法不用多说:

**public** **void** serivceJ() {

**try** {

HttpServletResponse response =

ServletActionContext.*getResponse*();

System.*out*.println("request...serivceJ");

response.setCharacterEncoding("UTF-8");

response.getWriter().write("hello world!");

} **catch** (IOException e) {

e.printStackTrace();

}

}

到此,一个完整的httpClient 访问https的流程就走OK了；

注意:生成keyStore的jdk和myEclipse的jdk要一致.否则可能出现错误;