# 什么是HTTPS？

HTTPS其实是有两部分组成：HTTP + SSL / TLS，

也就是在HTTP上又加了一层处理加密信息的模块，并且会进行身份的验证。

问题：

Firebug和postman之类的浏览器调试工具，为什么获取到的是明文？

解答：

SSL是对传输的数据进行加密，针对的是传输过程的安全。

firebug之类的浏览器调试工具，

因为他们得到的是客户端加密之前/解密之后的数据，因此是明文的。

# 什么是自签名证书？

就是自己生成的证书，并不是官方生成的证书。

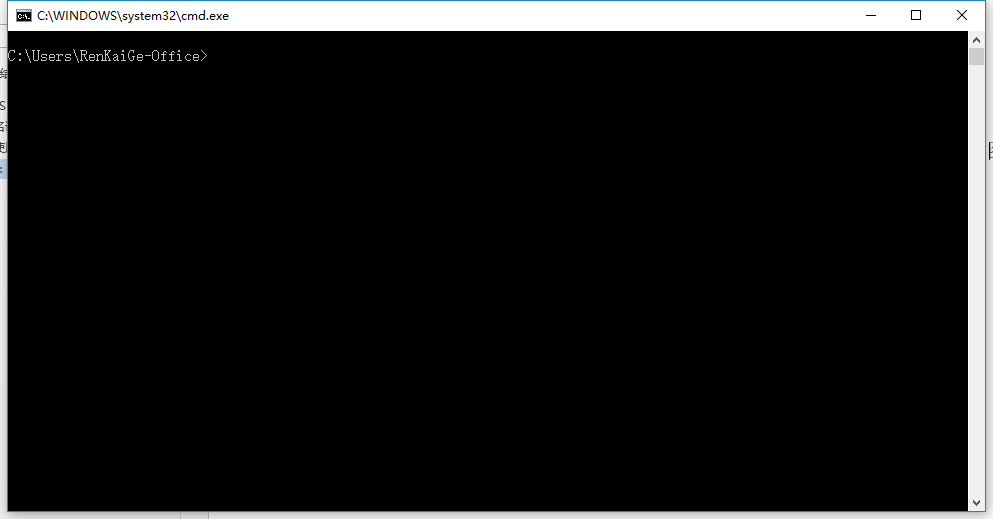
除非是很正式的项目，否则使用自己签发的证书即可，因为官方生成证书是要花钱滴。

# 进入正题，使用JDK自带工具KeyTool 生成自签发证书！

## 第一步：为服务器生成证书

打开CMD命令行工具，cd到C盘根目录或者是jdk的bin目录下，如下图所示：

因为使用的是jdk的keytool所以需要配置java环境变量,没有配置的请百度搜索如何配置java环境变量



使用keytool命令生成证书：

keytool

-genkey

-alias tomcat(别名)

-keypass 123456(别名密码)

-keyalg RSA(算法)

-keysize 1024(密钥长度)

-validity 365(有效期，天单位)

-keystore D:/keys/tomcat.keystore(指定生成证书的位置和证书名称)

-storepass 123456(获取keystore信息的密码)

-dname "CN=localhost,OU=iie,O=iie,L=beijing,ST=beijing,c=zh"(改参数现在也可以不输入等创建时会提示输入)

您的名字与姓氏是什么:CN

您的组织单位名称是什么:OU

您的组织名称是什么:O

您所在的城市或区域名称是什么:L

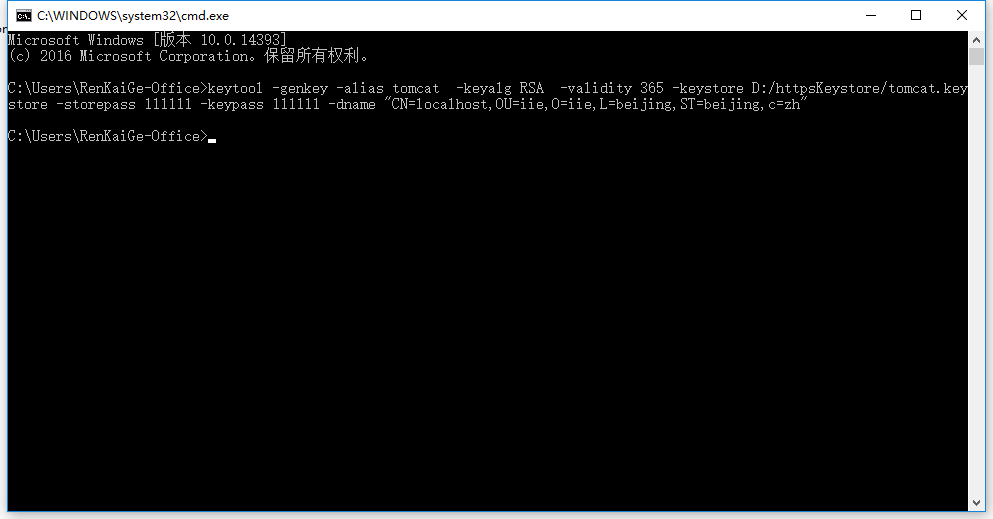
您所在的省/市/自治区名称是什么:ST

该单位的双字母国家/地区代码是什么:C

以上命令可以无先后顺序区分

方便复制版：

keytool -genkey -alias tomcat -keyalg RSA -validity 365 -keystore D:/httpsKeystore/tomcat.keystore -storepass 111111 -keypass 111111 -dname "CN=localhost,OU=iie,O=iie,L=beijing,ST=beijing,c=zh"



### 注意

#### D:/httpsKeystore/目录需要先创建好

#### 提示输入域名(CN)的时候不能输入IP地址(网址,本机用localhost)

## 为客户端生成证书

为浏览器(客户端)生成证书，以便让服务器来验证它。

为了能将证书顺利导入至IE和Firefox(或者客户端使用)，证书格式应该是PKCS12，

因此，使用如下命令生成：

因此，使用如下命令生成：

keytool

-genkey

-alias client

-keypass 123456

-keyalg RSA

-storetype PKCS12

-keypass 123456

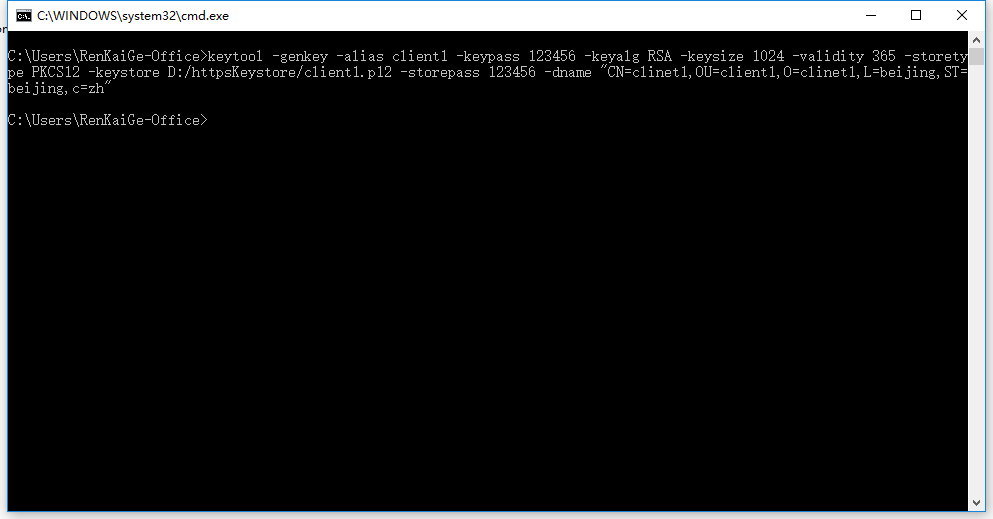
-storepass 123456

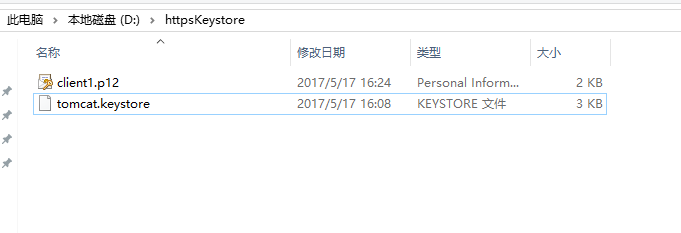
-keystore D:/keys/client.p12

其他同上3.1基本命令

方便复制版本

keytool -genkey -alias client1 -keypass 123456 -keyalg RSA -keysize 1024 -validity 365 -storetype PKCS12 -keystore D:/httpsKeystore/client1.p12 -storepass 123456 -dname "CN=clinet1,OU=client1,O=clinet1,L=beijing,ST=beijing,c=zh"





## 让服务器信任客户端证书

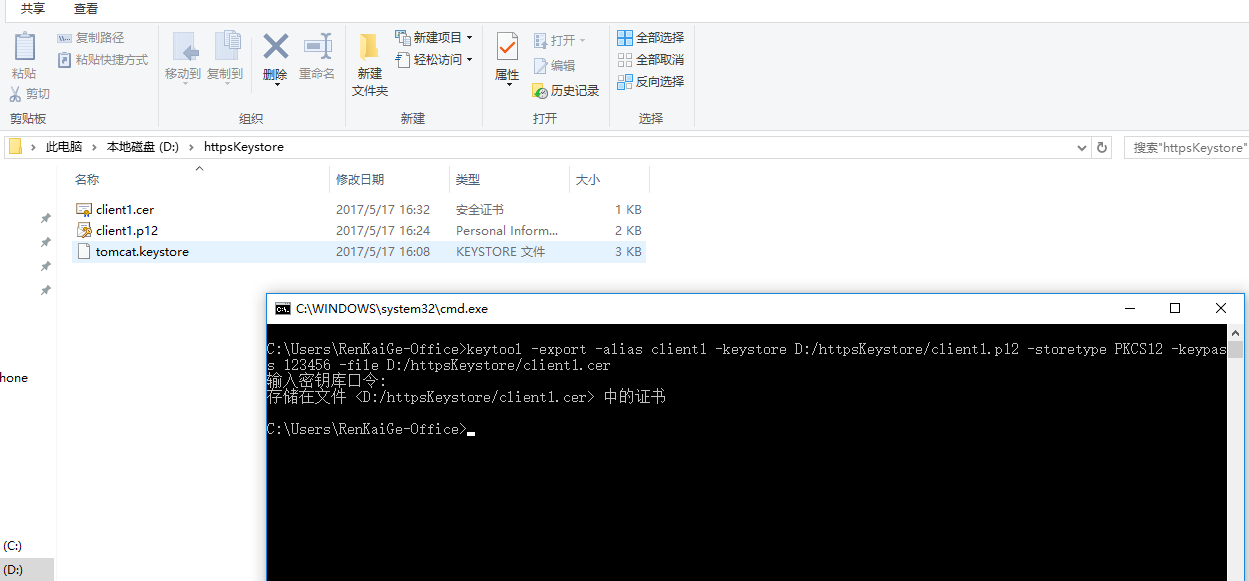
由于不能直接将PKCS12格式的证书库导入，

必须先把客户端证书导出为一个单独的CER文件，使用如下命令：

keytool -export -alias client1 -keystore D:/httpsKeystore/client1.p12 -storetype PKCS12 -keypass 123456 -file D:/httpsKeystore/client1.cer

注意：Keypass：指定CER文件的密码，但会被忽略，而要求重新输入

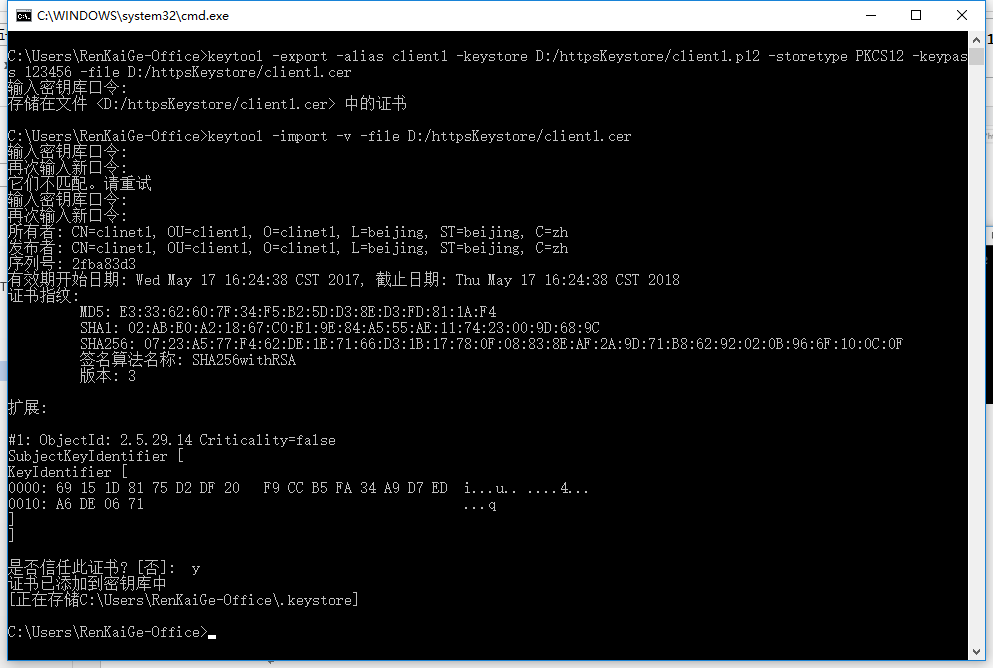
-alias client1别名一定要修改不然会报错已存在



将该文件导入到服务器的证书库，添加为一个信任证书：

keytool -import -v -file D:/httpsKeystore/client1.cer -keystore D:/httpsKeystore/tomcat.keystore -storepass 111111

(tomcat.keystore的密码)(执行时保持在一行 不要换行)

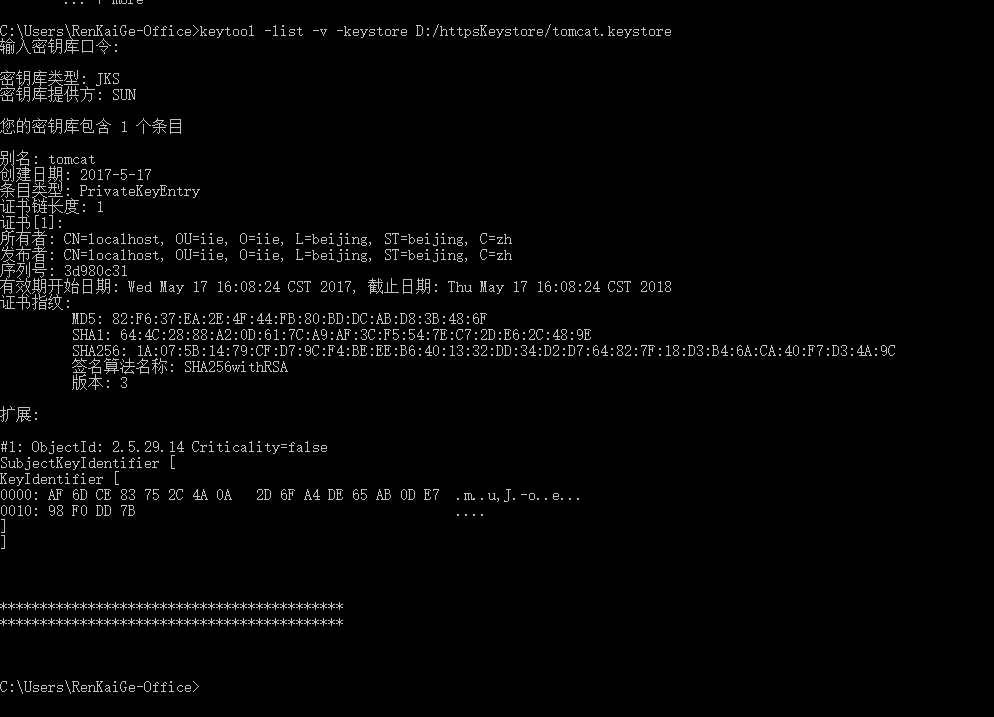


## 查看服务器证书库

完成之后通过list命令查看服务器的证书库，

可以看到两个证书，一个是服务器证书，一个是受信任的客户端证书：

keytool -list -v -keystore D:/httpsKeystore/tomcat.keystore



## 让客户端信任服务器证书

由于是双向SSL认证，客户端也要验证服务器证书，

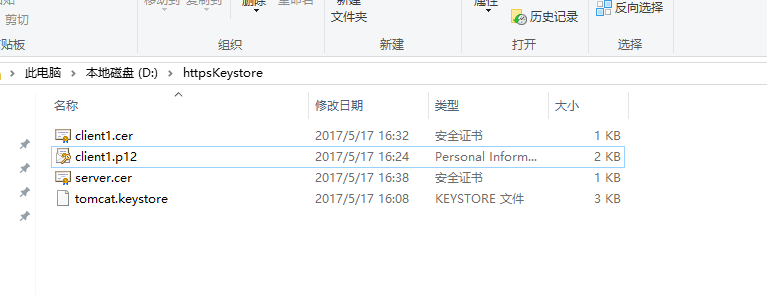
因此，必须把服务器证书添加到浏览器的“受信任的根证书颁发机构”。

由于不能直接将keystore格式的证书库导入，

必须先把服务器证书导出为一个单独的CER文件，使用如下命令：

keytool -keystore D:/httpsKeystore/tomcat.keystore -export –alias tomcat  -file D:/httpsKeystore/server.cer





## 配置tomcat

<Connector  port="8443"

protocol="org.apache.coyote.http11.Http11NioProtocol" SSLEnabled="true"

maxThreads="150"

scheme="https"

secure="true"

clientAuth="true"

sslProtocol="TLS"

keystoreFile="D:/keys/tomcat.keystore"

keystorePass="123456"

truststoreFile="D:/keys/tomcat.keystore"

truststorePass="123456" />

属性说明：

clientAuth:设置是否双向验证，默认为false，设置为true代表双向验证

keystoreFile:服务器证书文件路径

keystorePass:服务器证书密码

truststoreFile:用来验证客户端证书的根证书，此例中就是服务器证书

truststorePass:根证书密码

### 注意

设置clientAuth属性为True时，需要手动导入客户端证书才能访问。

要访问https请求 需要访问8443端口，访问http请求则访问Tomcat默认端口（你自己设置的端口，默认8080）即可。

## 总结

经过以上几步，你使用HTTPS 端口为8443 进行访问的时候 就是经过SSL信息加密，不怕被截获了。

通话的双方，必须是都拥有证书的端，才能进行会话，换句话说，就是只有安装了咱证书的客户端，才能与服务器通信。

# 小功能

设置强制https访问tomcat

说明：该演示项目强制使用了SSL，即普通的HTTP请求也会强制重定向为HTTPS请求，配置在最下面，可以去除，这样HTTP和HTTPS都可以访问。

修改web.xml

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

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<session-config>

<session-timeout>30</session-timeout>

</session-config>

<servlet>

<servlet-name>SSLServlet</servlet-name>

<servlet-class>ian.wang.ssl.servlet.SSLServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>SSLServlet</servlet-name>

<url-pattern>/sslServlet</url-pattern>

</servlet-mapping>

<!-- 强制SSL配置，即普通的请求也会重定向为SSL请求 -->

<security-constraint>

<web-resource-collection>

<web-resource-name>SSL</web-resource-name>

<url-pattern>/\*</url-pattern> <!-- 全站使用SSL -->

</web-resource-collection>

<user-data-constraint>

<description>SSL required</description>

<!-- CONFIDENTIAL: 要保证服务器和客户端之间传输的数据不能够被修改，且不能被第三方查看到 -->

<!-- INTEGRAL: 要保证服务器和client之间传输的数据不能够被修改 -->

<!-- NONE: 指示容器必须能够在任一的连接上提供数据。（即用HTTP或HTTPS，由客户端来决定） -->

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

</web-app>

完成以上步骤后，在浏览器中输入http的访问地址也会自动转换为https了。

# 附录

keytool常用命令

-alias       产生别名

-keystore    指定密钥库的名称(就像数据库一样的证书库，可以有很多个证书，cacerts这个文件是jre自带的，

             你也可以使用其它文件名字，如果没有这个文件名字，它会创建这样一个)

-storepass   指定密钥库的密码

-keypass     指定别名条目的密码

-list        显示密钥库中的证书信息

-v           显示密钥库中的证书详细信息

-export      将别名指定的证书导出到文件

-file        参数指定导出到文件的文件名

-delete      删除密钥库中某条目

-import      将已签名数字证书导入密钥库

-keypasswd   修改密钥库中指定条目口令

-dname       指定证书拥有者信息

-keyalg      指定密钥的算法

-validity    指定创建的证书有效期多少天

-keysize     指定密钥长度

使用说明：

导入一个证书命令可以如下：

keytool -import -keystore cacerts -storepass 666666 -keypass 888888 -alias alibabacert -file C:\alibabajava\cert\test\_root.cer

其中-keystore cacerts中的cacerts是jre中默认的证书库名字，也可以使用其它名字

-storepass 666666中的666666是这个证书库的密码

-keypass 888888中的888888是这个特定证书的密码

-alias alibabacert中的alibabacert是你导入证书的别名，在其它操作命令中就可以使用它

-file C:\alibabajava\cert\test\_root.cer中的文件路径就是要导入证书的路径

浏览证书库里面的证书信息，可以使用如下命令：

keytool -list -v -alias tomcat -keystore D:/httpsKeystore/tomcat.keystore -storepass 111111

要删除证书库里面的某个证书，可以使用如下命令：

keytool -delete -alias tomcat -keystore D:/httpsKeystore/tomcat.keystore -storepass 111111

要导出证书库里面的某个证书，可以使用如下命令：

keytool -export -keystore cacerts -storepass 666666 -alias alibabacert -file F:\alibabacert\_root.cer

要修改某个证书的密码（注意：有些数字认证没有私有密码，只有公匙，这种情况此命令无效）

这个是交互式的，在输入命令后，会要求你输入密码

keytool -keypasswd -alias alibabacert -keystore cacerts

这个不是交互式的，输入命令后直接更改

Keytool -keypasswd -alias alibabacert -keypass 888888 -new 123456 -storepass 666666 -keystore cacerts