1. 环境信息

服务器端使用tomcat

1. 单向认证：

简介：客户端向服务器发送消息，服务器接到消息后，用服务器端的密钥库中的私钥对数据进行加密，然后把加密后的数据和服务器端的公钥一起发送到客户端，客户端用服务器发送来的公钥对数据解密，然后在用传到客户端的服务器公钥对数据加密传给服务器端，服务器用私钥对数据进行解密，这就完成了客户端和服务器之间通信的安全问题，但是单向认证没有验证客户端的合法性。

1. 使用jdk的keytool工具生成keystore或jks文件。

keytool -genkeypair -alias keystorefilename -keyalg rsa -keystore c:/keystorefilename.jks -validity 365

输入密钥库口令:123456

再次输入新口令:123456

您的名字与姓氏是什么?

[Unknown]: 198.192.1.1

您的组织单位名称是什么?

[Unknown]: wzj

您的组织名称是什么?

[Unknown]: wzj

您所在的城市或区域名称是什么?

[Unknown]: sz

您所在的省/市/自治区名称是什么?

[Unknown]: gd

该单位的双字母国家/地区代码是什么?

[Unknown]: zh

CN=198.192.1.1, OU=wzj, O=wzj, L=sz, ST=gd, C=zh是否正确?

[否]: y

输入 <keystorefilename> 的密钥口令

(如果和密钥库口令相同, 按回车):

keystorefilename:要生成的keystore文件名，可以自己定义

c:/keystorefilename.jks：要生成keystore文件的位置，可以是绝对或者相对路径

365：证书从生成之日起的有效期，单位是天

操作完成后在当前用户的home目录下生成了keystorefilename.jks文件，可以将其生成或移动到和server.xml同级目录下，以方便配置时引用。

1. 在server.xml中引用jks文件，配置单向认证：此处将jks文件放置在conf目录下

Jboss参数：

<Connector

port="443"

protocol="HTTP/1.1"

SSLEnabled="true"

maxThreads="150"

scheme="<strong style="color: black; font-weight: bold; background-color: rgb(153, 255, 153);">https</strong>"

secure="true"

clientAuth="false"

sslProtocol="TLS"

keystoreFile="conf/keystorefilename.jks"

keystorePass="123456"

keystoreType="jks" />

tomcat参数：

<Connector

port="8443" maxHttpHeaderSize="8192"

maxThreads="150" minSpareThreads="25"

maxSpareThreads="75"

enableLookups="false"

disableUploadTimeout="true"

acceptCount="100" scheme="https"

secure="true"

clientAuth="false" sslProtocol="TLS"

keystoreFile="conf/tomcat.keystore"

keystorePass="123456" />

这样单向认证就配置好了，它不要求验证客户端信息，也就是所有客户端都可以访问到，而且，http和https都可以访问。

如果是浏览器访问的话，会提示“此网站的安全证书有问题，是否继续访问”。因为这个证书是用最简单的方法自己制作的，没有经过权威CA的签署，所以浏览器是不承认其安全性的。

1. 如果是用代码访问的话，则需要用jks文件生成证书文件.cer，再进行步骤4操作:

keytool -exportcert -alias keystorefilename -file keystorefilename.cer -keystore keystorefilename.jks

1. 将keystorefilename.cer给客户端，客户端用如下命令得到服务端的jks文件：

keytool -importcert -alias keystorefilename -file keystorefilename.cer -keystore server.jks

这样客户端就得到了服务端的keystore文件，把这个文件作为客户端代码的keystore,才能正常访问到。可以理解为因为是用代码来访问服务端，没有用户手工确认的过程，所以需要把证书加进来确认。

参数说明：

clientAuth：如果想要Tomcat为了使用这个socket而要求所有SSL客户出示一个客户证书，置该值为true。

keystoreFile：如果创建的keystore文件不在Tomcat认为的缺省位置（一个在Tomcat运行的home目录下的叫.keystore的文件），则加上该属性。可以指定一个绝对路径或依赖$CATALINA\_BASE环境变量的相对路径。

keystorePass：如果使用了一个与Tomcat预期不同的keystore（和证书）密码，则加入该属性。

keystoreType：如果使用了一个PKCS12 keystore，加入该属性。有效值是JKS和PKCS12。

sslProtocol：socket使用的加密/解密协议。如果使用的是Sun的JVM，则不建议改变这个值。据说IBM的1.4.1版的TLS协议的实现和一些流行的浏览器不兼容。这种情况下，使用SSL。

Ciphers：此socket允许使用的被逗号分隔的密码列表。缺省情况下，可以使用任何可用的密码。

Algorithm：使用的X509算法。缺省为Sun的实现（SunX509）。对于IBM JVMS应该使用ibmX509。对于其它JVM，参考JVM文档取正确的值。

truststoreFile：用来验证客户证书的TrustStore文件。

truststorePass：访问TrustStore使用的密码。缺省值是keystorePass。

truststoreType：如果使用一个不同于正在使用的KeyStore的TrustStore格式，加入该属性。有效值是JKS和PKCS12。

1. 双向认证：

简介：客户端向服务器发送消息，首先把消息用客户端证书加密然后连同时把客户端证书一起发送到服务器端，服务器接到消息后用首先用客户端证书把 消息解密，然后用服务器私钥把消息加密，把服务器证书和消息一起发送到客户端，客户端用发来的服务器证书对消息进行解密，然后用服务器的证书对消息加密， 然后在用客户端的证书对消息在进行一次加密，连同加密消息和客户端证书一起发送到服务器端，到服务器端首先用客户端传来的证书对消息进行解密，确保消息是 这个客户发来的，然后用服务器端的私钥对消息在进行解密这个便得到了明文数据。

双向认证比较复杂，需要客户端的校验，也就是说不是随便什么客户端都能访问到的，客户端要调用服务端，同样也要制作证书。

1. 制作客户端证书：

keytool -genkeypair -alias client -keyalg rsa -keystore c:/client. keystore -validity 365

1. 制作服务端证书：

keytool -genkeypair -alias server -keyalg rsa -keystore c:/ server. keystore -validity 365

1. 让客户端信任服务端证书；
   1. 将服务端证书导出为一个.cer文件，将.cer文件给客户端。

keytool -exportcert -alias server -file server.cer -keystore server.keystore

* 1. 在客户端将server.cer导出为一个服务端的trustkeyStore秘钥库文件,放在客户端：

keytool -importcert -alias server -file server.cer -keystore client.keystore

1. 让服务端信任客户端证书：
   1. 将客户端证书导出为一个.cer文件，将.cer文件给服务端。

keytool -exportcert -alias client -file client.cer -keystore server.keystore

* 1. 在服务端将client.cer导出为客户端的trustkeyStore秘钥库文件,放在服务端：

keytool -importcert -alias client -file client.cer -keystore server.keystore

1. 在客户端配置tomcat中server.xml,其中trustkeystore配置服务端证书，keystore配置客户端自己的证书。
2. 在服务端配置tomcat中server.xml,其中trustkeystore配置客户端证书，keystore配置服务端自己的证书。
3. 查看秘钥库文件中信息：

keytool -v -list -keystore D:\tomcat.keystore