# HTTPS双向认证原理

**双向认证的具体流程：**



**流程描述：**

1、客户端向服务端发送https请求。

2、服务端给客户端返回SSL协议版本号、加密算法种类、随机数等信息，同时也返回服务器端的证书，即公钥证书。

3、客户端使用服务端返回的信息验证服务器的合法性，包括：

1. 证书是否过期
2. 发送服务器证书的CA是否可靠
3. 返回的公钥是否能正确解开返回证书中的数字签名
4. 服务器证书上的域名是否和服务器的实际域名相匹配

验证通过后，将继续进行通信，否则，终止通信

4、服务端要求客户端发送客户端的证书，客户端会将自己的证书发送至服务端。

5、验证客户端的证书，如果没有通过验证，拒绝连接，如果通过验证，会获得客户端的公钥。

6、客户端向服务端发送自己所能支持的对称加密方案，供服务器端进行选择。

7、服务器端在客户端提供的加密方案中选择加密程度最高的加密方式。

8、服务器将选择好的加密方案通过明文方式返回给客户端。

9、客户端收到服务端返回的加密方案密文后，使用自己的私钥进行解密，获取具体加密方式，然后，产生该加密方式的随机码，用作加密过程中的密钥，使用之前从服务端证书中获取到的公钥进行加密后，发送给服务端。

10、服务端收到客户端发送的消息后，使用自己的私钥进行解密，获取对称加密的密钥。

在接下来的会话中，服务器和客户端将会使用该密码进行对称加密，保证通信过程中信息的安全。

**证书**

1、每个人都可以使用一些证书生成工具为自己的https站点生成证书（比如jdk的keytool、openssl等），大家称它为“自签名证书”，但是自己生成的证书是不被互联网承认的，所以浏览器会报安全提示，要求你手动安装证书。只有通过权威的CA机构付费获得的证书才能被互联网承认（http://www.cfca.com.cn）。

2、证书（无客户端服务端之分）保存着ip信息、证书过期时间、证书所有者地址信息等。

**安全**

防监听：数据是加密的，所以监听得到的数据是密文，很难解密。

防伪装：伪装分为客户端伪装和服务器伪装，通信双方携带证书，证书相当于身份证，有证书就认为合法，没有证书就认为非法，证书由第三方颁布，很难伪造。

防篡改：https对数据做了摘要，篡改数据会被感知到。

**性能**

对性能相对于http有一定的影响（具体取决于报文的大小、加密的算法、证书位数），这也是很多企业不用https的原因，除非对安全方面要求特别高（如银行系统）。

# HTTPS的优点和缺点

**技术 成本 安全**

趋势 硬件 加密

兼容 证书 劫持

速度 人力 搜索

**HTTPS的优点：**

安全性方面

在目前的技术背景下，HTTPS是现行架构下最安全的处理方案，主要有以下几个好处：

1、用HTTPS协议可认证使用户和服务器，确保数据发送到正确的用户机和服务器;

2、HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

3、HTTPS是现行架构下最安全的处理方案，尽管不是绝对安全，但它大幅添加了中间人攻击的成本。

**HTTPS的缺点：**

技术方面

1、相同网络环境下，HTTPS协议会使页面的加载时间延长近50%，添加10%到20%的耗电。此外，HTTPS协议还会影响缓存，添加数据开销和功耗。

2、HTTPS协议的安全是有范围的，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作使用。

3、最关键的，SSL 证书的信誉链体系并不安全。特别是在某些国家能控制 CA 根证书的情况下，中间人攻击一样可行。

成本方面

1、SSL的专业证书需要购买，功可以越强大的证书费使用越高。个人网站、小网站能选择入门级免费证书。

2、SSL 证书通常需要绑定固定域名，如果域名发生变化，需要重新申请证书;

3、HTTPS 连接服务器端资源占用较高，相同负载下会添加带宽和服务器投入成本;

既然HTTPS有这么多缺点，那是不是就不该做呢，当然不是的，随着技术的发展很多缺点是能优化和弥补的。比方：

打开速度问题完全能通过CDN加速处理，很多IDC也在着手推出免费证书和一站式HTTPS搭建服务，HTTPS成本在未来将会大大缩小!



调研中发现，大多数人对HTTPS持观望态度，他们对HTTPS安全性是认可的，但是从各个层面进行考虑后，做出了目前不做HTTPS网站的决定，主要有以下两种观点：

**正方观点**

1、HTTPS具备更好的加密性，可以避免用户信息泄露;

2、HTTPS复杂的传输方式，降低网站被劫持的风险;

3、搜索引擎已经全面支持HTTPS抓取、收录，并且会优先展现HTTPS结果;

4、从安全角度来说个人觉得要做HTTPS;

5、HTTPS绿锁表示能提升使用户对网站信任程度;

6、基础成本可控，证书及服务器已经有了成型的支持方案;

7、网站加载速度能通过cdn等方式进行弥补，但是安全不可以忽略;

8、HTTPS是网络的发展趋势，早晚都要做;

9、能有效防止山寨、镜像网站;

**反方观点**

1、HTTPS会降低使用户访问速度，添加网站服务器的计算资源耗费;

2、目前搜索引擎只是收录了小部分HTTPS内容，应该保持观望制度;

3、HTTPS需要申请加密协议，添加了经营成本;

4、百度目前对HTTPS的优先展示效果不显著，谷歌较为显著;

5、技术门槛较高，无从下手;

6、目前站点不涉及私密信息，无需HTTPS;

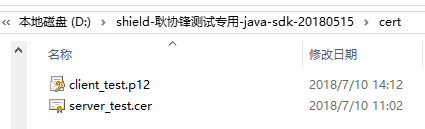
7、兼容性有待提升，如robots不支持/联盟广告不支持等;

8、HTTPS网站的安全程度有限，该被黑还是被黑;

9、HTTPS维护比较麻烦，在搜索引擎支持HTTP的情况，没必要做HTTPS;

# 和API网关集成

证书统一由API网关提供，通过后台管理，下载sdk包，证书路径在cert目录下：



**将服务端公钥证书导入客户端证书容器：**

--如果你用浏览器直接https访问，此步可以忽略，直接安装个人证书到浏览器就行了

|  |
| --- |
| keytool -import -trustcacerts -alias server\_iflytek -file d://cert/server\_test.cer -keystore d://cert/client\_test.p12 |

**查看证书库**

**命令：**

|  |
| --- |
| keytool -list -keystore d://cert/client\_test.p12 -storepass 123456 -v |

# HTTPS双向认证实例

## 创建证书

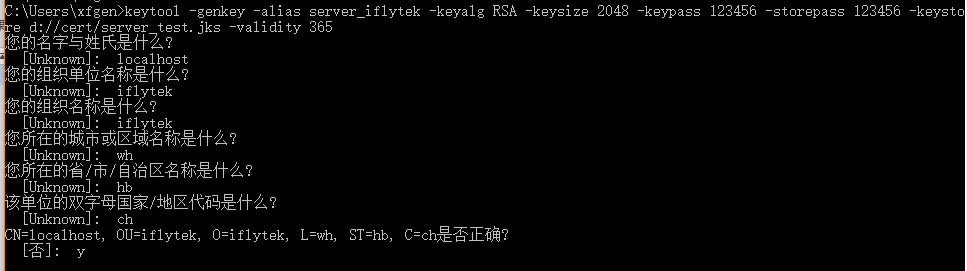
**keytool命令如下：**

-genkey 创建一个证书库，其中包含用户的公钥、私钥和证书  
-alias   产生一个server\_iflytek的别名  
-keystore 指定密钥库的名称(产生的各类信息将不在.jks文件中  
-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  
-storepass 指定密钥库的密码  
-keypass  指定别名条目的密码  
-dname   指定证书拥有者信息

**1.创建服务器证书库**

**命令：**

|  |
| --- |
| keytool -genkey -alias server\_iflytek -keyalg RSA -keysize 2048 -keypass 123456 -storepass 123456 -keystore d://cert/server\_test.jks -validity 365 |



**2.导出服务器证书**

**命令：**

|  |
| --- |
| keytool -export -trustcacerts -alias server\_iflytek -file d://cert/server\_test.cer -rfc -keystore d://cert/server\_test.jks -storepass 123456 |

**3.创建客户端证书**

**命令：**

|  |
| --- |
| keytool -genkey -alias client\_iflytek -keyalg RSA -keysize 2048 -keypass 123456 -storepass 123456 -keystore d://cert/client\_test.p12 -validity 365 -storetype PKCS12 |

**4.导出客户端证书**

**命令：**

|  |
| --- |
| keytool -export -trustcacerts -alias client\_iflytek -file d://cert/client\_test.cer -rfc -keystore d://cert/client\_test.p12 -storepass 123456 -storetype PKCS12 |

**5.** **交换导入服务端和客户端证书，作为双方信任证书**

**将客户端证书导入服务端：**

|  |
| --- |
| keytool -import -trustcacerts -alias client\_iflytek -file d://cert/client\_test.cer -keystore d://cert/server\_test.jks |

**将服务端证书导入客户端：**

--如果你用浏览器直接https访问，此步可以忽略，直接安装个人证书到浏览器就行了

|  |
| --- |
| keytool -import -trustcacerts -alias server\_iflytek -file d://cert/server\_test.cer -keystore d://cert/client\_test.p12 |

**6.查看证书库**

**命令：**

|  |
| --- |
| keytool -list -keystore d://cert/server\_test.jks -storepass 123456 -v |

## 服务端实现

**初始化SSLContext**

|  |
| --- |
| //初始化证书库（包含服务端和信任的客户端证书） KeyStore ks = KeyStore.*getInstance*(AppConstant.*KEY\_STORE\_INSTANCE\_NAME*); ks.load(SslKeyStore.*getKeyStoreStream*(), SslKeyStore.*getKeyStorePassword*()); //决定将哪一个认证证书发送给对端服务器 KeyManagerFactory kmf = KeyManagerFactory.*getInstance*(algorithm); kmf.init(ks, SslKeyStore.*getCertificatePassword*()); //决定对端的认证证书是否被信任 TrustManagerFactory tmf = TrustManagerFactory.*getInstance*(algorithm); tmf.init(ks) ; //初始化SSLContext *sslContext* = SSLContext.*getInstance*(AppConstant.*SSL\_CONTEXT\_INSTANCE\_NAME*); *sslContext*.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null); |

**设置是否是双向认证**

|  |
| --- |
| SSLEngine sslEngine = *sslContext*.createSSLEngine(); sslEngine.setUseClientMode(false); sslEngine.setNeedClientAuth(true); return sslEngine ; |

## 客户端实现

**初始化SSLContext**

|  |
| --- |
| //决定将哪一个认证证书发送给对端服务器 KeyManagerFactory kmf = KeyManagerFactory.*getInstance*(algorithm); kmf.init(*getkeyStore*(),SslKeyStore.*getKeyStorePassword*()); //决定对端的认证证书是否被信任 TrustManagerFactory tmf=TrustManagerFactory.*getInstance*(algorithm); tmf.init(*getTrustStore*()); //初始化SSLContext *sslContext* = SSLContext.*getInstance*(AppConstant.*SSL\_CONTEXT\_INSTANCE\_NAME*); *sslContext*.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null); |

**https请求设置**

|  |
| --- |
| // 打开和URL之间的连接 if(connection instanceof HttpsURLConnection){  ((HttpsURLConnection)connection)  .setSSLSocketFactory(SslContextFactory.*getSSLContext*().getSocketFactory()); } |

## Tomcat web容器配置

|  |  |
| --- | --- |
| **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 |

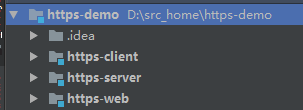
**server.xml –单向认证**

|  |
| --- |
| **<Connector port="8443"**  **protocol="org.apache.coyote.http11.Http11Protocol"**  **SSLEnabled="true"**  **clientAuth="false"**  **maxThreads="200"**  **scheme="https"**  **secure="true"**  **sslProtocol="TLS"**  **keystoreFile="d:\cert\server\_test.jks" keystorePass="123456"/>** |

**server.xml –双向认证**

|  |
| --- |
| **<Connector port="8443"**  **protocol="org.apache.coyote.http11.Http11Protocol"**  **SSLEnabled="true"**  **clientAuth="true"**  **maxThreads="200"**  **scheme="https"**  **secure="true"**  **sslProtocol="TLS"**  **keystoreFile="d:\cert\server\_test.jks" keystorePass="123456"**  **truststoreFile="d:\cert\server\_test.jks" truststorePass="123456"/>** |

## Demo说明

https-client 客户端实现，直接运行 HttpsMain

https-server 服务端实现，直接运行 ServerStartUp

https-web web端实现，依赖web容器，如tomcat