# 前期的准备工作

1. 安装openssl和nginx的https模块。
2. 在当前目录下，创建相关文件夹。

mkdir ssl

cd ssl

mkdir demoCA

cd demoCA

mkdir newcerts

mkdir private

touch index.txt

echo '01' **>** serial

1. openssl的配置文件/usr/local/ssl/openssl.cnf中dir默认为demoCA文件夹名称。

# 制作CA证书

这是信任的起点，根证书，所有其他的证书都要经过CA的私钥签名。

1. # 生成CA私钥

openssl genrsa -des3 -out ca.key 1024

1. # X.509 Certificate Signing Request (CSR) Management.

openssl req -new -key ca.key -out ca.csr

1. # X.509 Certificate Data Management.

openssl x509 -req -days 3650 -signkey ca.key -in ca.csr -out ca.crt

1. CA根证书CRT转换为 PKCS #12格式(全称应该叫做 Personal Information Exchange，通常以 p12作为后缀)

openssl pkcs12 -export -clcerts -in ca.crt -inkey ca.key -out ca.p12

# 制作服务器端证书，并用CA签名认证

1. 假设我们需要为 test.com 域名制作证书，先生成 test.com的证书私钥 test.com.pem

openssl genrsa -des3 -out test.com.pem 1024

1. 生成 csr 签名请求

openssl req -new -key test.com.pem -out test.com.csr

这里需要输入国家，地区，组织，email等。最重要的是\*\*common name\*\*，可以写你的名字或者域名。如果为了 https 申请，这个必须和域名一样，即，这里要写test.com，否则会引发浏览器警报，这里可以用 \*.test.com 来做泛域名证书。

1. 用CA证书进行签名

openssl ca -policy policy\_anything -days 1460 -cert ./demoCA/ca.crt -keyfile ./demoCA/ca.key -in test.com.csr -out test.com.crt

1. 把 ca.crt 的内容追加到 test.com.crt后面，因为有些浏览似乎不支持

cat ./demoCA/ca.crt >> test.com.crt

# 制作客户端证书，并用CA签名认证

1. 准备客户端私钥

openssl genrsa -des3 -out clent.pem 2048

1. 生成客户端证书请求

openssl req -new -key client.pem -out client-req.csr

1. CA签名客户端证书请求

openssl ca -policy policy\_anything -days 1460 -cert ./demoCA/ca.crt  -keyfile ./demoCA/ca.key -in client-req.csr -out client.crt

1. 客户端证书CRT转换为 PKCS #12格式(全称应该叫做 Personal Information Exchange，通常以 p12作为后缀)

openssl pkcs12 -export -clcerts -in client.crt -inkey client.pem -out client.p12

# 使用Node作为服务器和客户端进行测试

1. 确保node已经安装。
2. 服务器端代码http-server.js

var https = require('https');

var fs = require('fs');

var options = {

key: fs.readFileSync('/data/ssl/test.com.pem'),

cert: fs.readFileSync('/data/ssl/test.com.crt') ,

**//配置是否必须验证客户端证书，即SSL双向认证**

**// This is necessary only if using the client certificate authentication.**

**requestCert: true,**

**rejectUnauthorized: true,**

**// This is necessary only if the client uses the self-signed certificate.**

**ca: [ fs.readFileSync('/data/ssl/demoCA/ca.crt') ]**

};

https.createServer(options, function(req, res) {

res.writeHead(200);

res.end('hello world');

}).listen(8000);

1. 启动服务器端代码

node http-server.js

1. 浏览器访问

Firefox和Chrome浏览器提示分别为：





1. 导入ca.crt跟证书，再次访问，正常显示
2. 客户端代码http\_client.js

var https = require('https');

var fs = require('fs');

var options = {

hostname: "www.test.com",

port: 8000,

path: '/',

methed: 'GET',

**//配置验证客户端证书，即SSL双向认证**

**key: fs.readFileSync('/data/ssl/client.pem'),**

**cert: fs.readFileSync('/data/ssl/client.crt'),**

**ca: [fs.readFileSync('/data/ssl/demoCA/ca.crt')]**

};

options.agent = new https.Agent(options);

var req = https.request(options, function(res) {

res.setEncoding('utf-8');

res.on('data', function(d) {

console.log(d);

});

});

req.end();

req.on('error', function(e) {

console.log(e);

});

1. 启动客户端代码

node http-client.js

正常输出结果

# Java代码实现HTTPS接口访问（单向或双向认证）

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.io.UnsupportedEncodingException;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**import** java.security.KeyManagementException;

**import** java.security.KeyStore;

**import** java.security.KeyStoreException;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.SecureRandom;

**import** java.security.UnrecoverableKeyException;

**import** java.security.cert.CertificateException;

**import** java.util.List;

**import** java.util.Map;

**import** javax.net.ssl.HostnameVerifier;

**import** javax.net.ssl.HttpsURLConnection;

**import** javax.net.ssl.KeyManager;

**import** javax.net.ssl.KeyManagerFactory;

**import** javax.net.ssl.SSLContext;

**import** javax.net.ssl.SSLSession;

**import** javax.net.ssl.TrustManager;

**import** javax.net.ssl.TrustManagerFactory;

**public** **class** HttpTest {

**public** **static** String *KEY\_STORE\_FILE*="G:\\client.p12";

**public** **static** String *KEY\_STORE\_PASS*="123456";

**public** **static** String *TRUST\_STORE\_FILE*="G:\\ca.p12";

**public** **static** String *TRUST\_STORE\_PASS*="123456";

**private** **static** SSLContext *sslContext*;

/\*\*

\* 向指定URL发送GET方法的请求

\*

\* **@param** url

\* 发送请求的URL

\* **@param** param

\* 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。

\* **@return** URL 所代表远程资源的响应结果

\*

\*/

**public** **static** String sendGet(String url, String param) {

String result = "";

BufferedReader in = **null**;

**try** {

String urlNameString = url + "?" + param;

URL realUrl = **new** URL(urlNameString);

// 打开和URL之间的连接

HttpURLConnection connection = (HttpURLConnection) realUrl.openConnection();

// 打开和URL之间的连接

**if**(connection **instanceof** HttpsURLConnection){

((HttpsURLConnection)connection)

.setSSLSocketFactory(*getSSLContext*().getSocketFactory());

}

// 设置通用的请求属性

connection.setRequestProperty("accept", "\*/\*");

connection.setRequestProperty("connection", "Keep-Alive");

connection.setRequestProperty("user-agent",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");

// 建立实际的连接

connection.connect();

// 获取所有响应头字段

Map<String, List<String>> map = connection.getHeaderFields();

// 遍历所有的响应头字段

**for** (String key : map.keySet()) {

//System.out.println(key + "--->" + map.get(key));

}

// 定义 BufferedReader输入流来读取URL的响应

**if**(connection.getResponseCode()==200){

in = **new** BufferedReader(**new** InputStreamReader(

connection.getInputStream()));

}**else**{

in = **new** BufferedReader(**new** InputStreamReader(

connection.getErrorStream()));

}

String line;

**while** ((line = in.readLine()) != **null**) {

result += line;

}

} **catch** (Exception e) {

System.***out***.println("发送GET请求出现异常！" + e);

e.printStackTrace();

}

// 使用finally块来关闭输入流

**finally** {

**try** {

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

in.close();

}

} **catch** (Exception e2) {

e2.printStackTrace();

}

}

**return** result;

}

/\*\*

\* 向指定 URL 发送POST方法的请求

\*

\* **@param** url

\* 发送请求的 URL

\* **@param** param

\* 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。

\* **@return** 所代表远程资源的响应结果

\*/

**public** **static** String sendPost(String url, String param) {

PrintWriter out = **null**;

BufferedReader in = **null**;

String result = "";

**try** {

URL realUrl = **new** URL(url);

// 打开和URL之间的连接

HttpURLConnection conn = (HttpURLConnection) realUrl.openConnection();

**if**(conn **instanceof** HttpsURLConnection){

((HttpsURLConnection)conn)

.setSSLSocketFactory(*getSSLContext*().getSocketFactory());

}

// 设置通用的请求属性

conn.setRequestProperty("accept", "\*/\*");

conn.setRequestProperty("connection", "Keep-Alive");

conn.setRequestProperty("user-agent",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");

// 发送POST请求必须设置如下两行

conn.setDoOutput(**true**);

conn.setDoInput(**true**);

// 获取URLConnection对象对应的输出流

out = **new** PrintWriter(conn.getOutputStream());

// 发送请求参数

out.print(param);

// flush输出流的缓冲

out.flush();

// 定义BufferedReader输入流来读取URL的响应

**if**(conn.getResponseCode()==200){

in = **new** BufferedReader(

**new** InputStreamReader(conn.getInputStream()));

}**else**{

in = **new** BufferedReader(

**new** InputStreamReader(conn.getErrorStream()));

}

String line="";

**while** ((line = in.readLine()) != **null**) {

result += line;

}

} **catch** (Exception e) {

System.***out***.println("发送 POST 请求出现异常！"+e);

e.printStackTrace();

}

//使用finally块来关闭输出流、输入流

**finally**{

**try**{

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

out.close();

}

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

in.close();

}

}**catch**(IOException ex){

ex.printStackTrace();

}

}

**return** result;

}

**public** **static** SSLContext getSSLContext(){

**long** time1=System.*currentTimeMillis*();

**if**(*sslContext*==**null**){

**try** {

KeyManagerFactory kmf = KeyManagerFactory.*getInstance*("SunX509");

kmf.init(*getkeyStore*(),*KEY\_STORE\_PASS*.toCharArray());

KeyManager[] keyManagers = kmf.getKeyManagers();

TrustManagerFactory trustManagerFactory=TrustManagerFactory.*getInstance*("SunX509");

trustManagerFactory.init(*getTrustStore*());

TrustManager[] trustManagers= trustManagerFactory.getTrustManagers();

*sslContext* = SSLContext.*getInstance*("TLS");

**//keyManagers为null，为SSL单向认证；不为null，SSL双向认证**

*sslContext*.init(keyManagers, trustManagers, **new** SecureRandom());

HttpsURLConnection.*setDefaultHostnameVerifier*(**new** HostnameVerifier() {

@Override

**public** **boolean** verify(String hostname, SSLSession session) {

**return** **true**;

}

});

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (UnrecoverableKeyException e) {

e.printStackTrace();

} **catch** (KeyStoreException e) {

e.printStackTrace();

} **catch** (KeyManagementException e) {

e.printStackTrace();

}

}

**long** time2=System.*currentTimeMillis*();

System.***out***.println("SSLContext 初始化时间："+(time2-time1));

**return** *sslContext*;

}

**public** **static** KeyStore getkeyStore(){

KeyStore keySotre=**null**;

**try** {

keySotre = KeyStore.*getInstance*("PKCS12");

FileInputStream fis = **new** FileInputStream(**new** File(*KEY\_STORE\_FILE*));

keySotre.load(fis, *KEY\_STORE\_PASS*.toCharArray());

fis.close();

} **catch** (KeyStoreException e) {

e.printStackTrace();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (CertificateException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** keySotre;

}

**public** **static** KeyStore getTrustStore() **throws** IOException{

KeyStore trustKeyStore=**null**;

FileInputStream fis=**null**;

**try** {

trustKeyStore=KeyStore.*getInstance*("PKCS12");

fis = **new** FileInputStream(**new** File(*TRUST\_STORE\_FILE*));

trustKeyStore.load(fis, *TRUST\_STORE\_PASS*.toCharArray());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (KeyStoreException e) {

e.printStackTrace();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (CertificateException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}**finally**{

fis.close();

}

**return** trustKeyStore;

}

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

Long time1=System.*currentTimeMillis*();

// int k=0;

// for(int i=0;i<100;++i){

String result=*sendGet*("https://www.test.com:8000/", "");

System.***out***.println(result);

// ++k;

// }

Long time2=System.*currentTimeMillis*();

// System.out.println("平均耗费时间:"+(time2-time1)/k);

}

}

# 问题

1. 问题1

**问题描述：制作服务器或客户端证书错误**

failed to update database

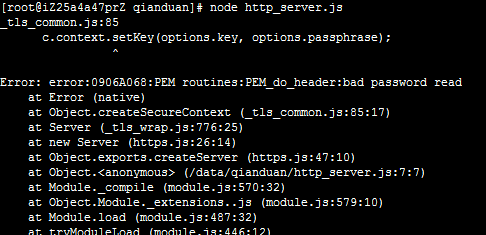
TXT\_DB error number 2

**问题解决方案：**

删除demoCA/index.txt，然后touch demoCA/index.txt

1. 问题2

**问题描述：启动http\_server.js错误**



**问题解决方案：**

openssl rsa -in test.com.pem -out newkey.pem && mv newkey.pem test.com.pem

openssl rsa -in client.pem -out newkey.pem && mv newkey.pem client.pem