

# 关于 https 的双向认证详解

## 基本描述

Https 是在基本的 http 通讯协议上增加的 tls 安全协议，tls 安全协议需要 ssl 安全证书作为加解密要素，双向认证是指在客户端确认服务的是否可信任的单向认证基础上增加服务端的对客户端的认证。

## 主要内容

### https 通信协议

是以安全为目标的 HTTP 通道, 简单讲是 HTTP 的安全版。即 HTTP 下加入 SSL 层, HTTPS 的安全基础是 SSL, 因此加密的详细内容就需要 SSL

参见 [百度 https 百科](#)

### tls 安全协议

SSL(Secure Sockets Layer 安全套接层),及其继任者传输层安全 (Transport Layer Security, TLS) 是为网络通信提供安全及数据完整性的一种安全协议。TLS 与 SSL 在传输层对网络连接进行加密

参见 [百度 SSL](#)

### ssl 安全证书

SSL 证书就是遵守 SSL 协议, 由受信任的数字证书颁发机构 CA, 在验证服务器身份后颁发, 具有服务器身份验证和数据传输加密功能

参见 [百度 ssl 证书](#)

### 证书链

~~证书链~~的描述, 证书链的可信传递机制, 以及根证书的来源和查看方式

证书格式

常见证书格式和转换

<https://blog.csdn.net/justinjing0612/article/details/7770301>

## tls 认证过程

SSL 认证是指[客户端](#)到服务器端的认证。主要用来提供对用户和服务器的认证；对传送的数据进行加密和隐藏；确保数据在传送中不被改变，即数据的完整性，现已成为该领域中全球化的标准，即所谓的单向认证

参见 [百度 ssl 认证](#)

## 其他内容

HTTPS 实战之单向验证和双向验证

<https://www.jianshu.com/p/119c4dbb7225>

浅谈 HTTPS (SSL/TLS) 原理

<https://www.jianshu.com/p/41f7ae43e37b>

HTTPS 通信中的身份认证机制

<https://blog.csdn.net/bravegogo/article/details/60766773>

SSL 双向认证以及证书的制作和使用-https+客户端身份验证

<https://blog.csdn.net/soarheaven/article/details/78784848>

X - Certificate and Key management 证书管理工具

<https://hohnstaedt.de/xca/index.php>

ssl 证书相当于安全的钥匙，如果私钥外泄被恶意盗用，会存在安全问题

## 双向认证的常用场景

### 银行网银 U 盾使用

专业版登录支持 U 盾认证多重认证

资金动账类交易 U 盾再次认证确认客户端身份

# 金融支付工具安全数字证书

支付宝数字证书 app 下载安装可以提高 app 支付安全，支付额度  
常用 pc 安装数字证书免除重复的其他安全校验

## 其他对通信安全有较高要求的都可以应用此 https 双向认证

主要成本在于向具有 CA 资质的机构申请有效的证书，非 CA 机构的自制根证书及证书链管理需要自行负责

# https 应用的使用示例

## 工具版本说明

本地操作系统版本: win10

本地 tomcat 版本 9

本地 java 版本 1.8

证书管理工具 xca 2.1.2

Keytool 界面工具 1.6

[http://enkj.jb51.net:81/201703/tools/keytool\\_jb51.rar](http://enkj.jb51.net:81/201703/tools/keytool_jb51.rar)

## 前提概要

客户端和服务端角色根据通信请求来确定, 客户端和服务端需要保持支持一致的 TLS 安全协议版本

## 单向认证说明

单向认证首先需要申请制作证书链

对于客户端请求服务端, 比较常见的客户端是指浏览网页使用的浏览器 (chrome, IE, firefox 等) 以及移动客户端 (ios 或 android app), 服务端一般指客户端请求目标服务的提供者

## 证书链制作

--也可以使用其他证书管理工具（keytool、openssl 等）

先下载安装 xca 工具，地址是 <http://xca.hohnstaedt.de/>

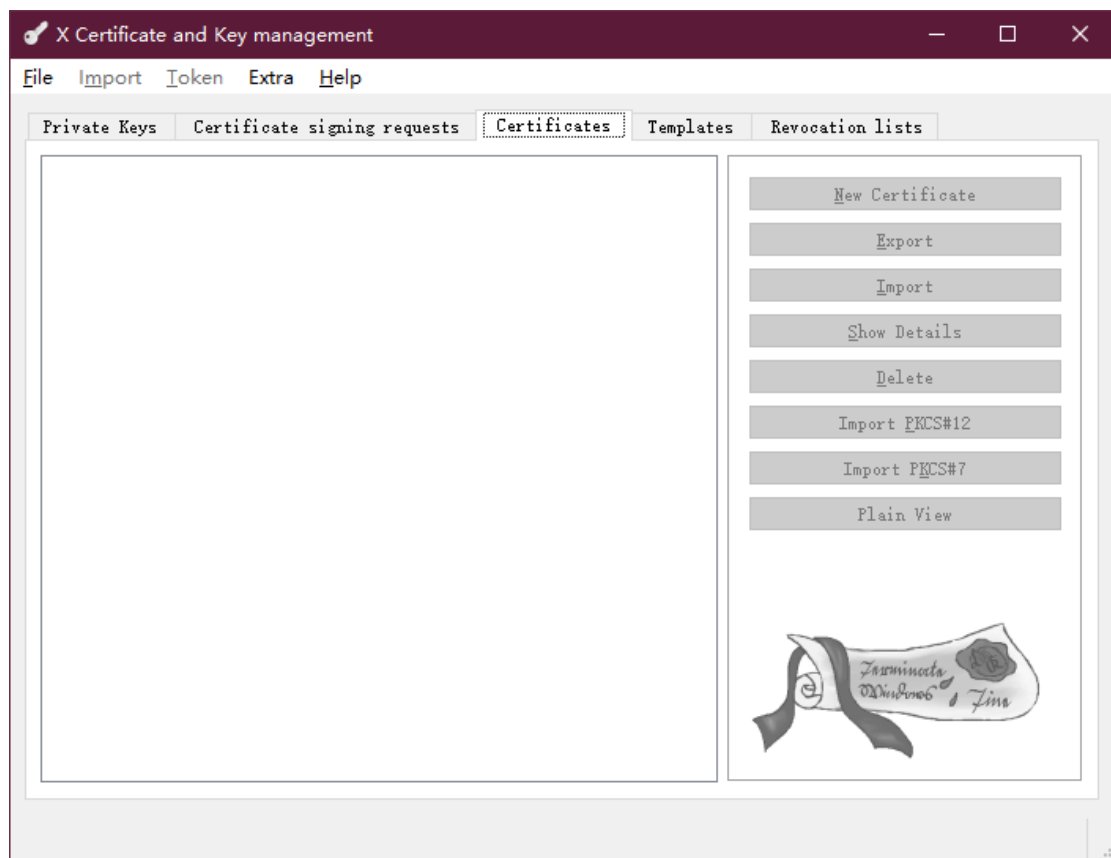
先用 xca 创建一本 ca 证书，

本次说明使用的版本为 2.1.2

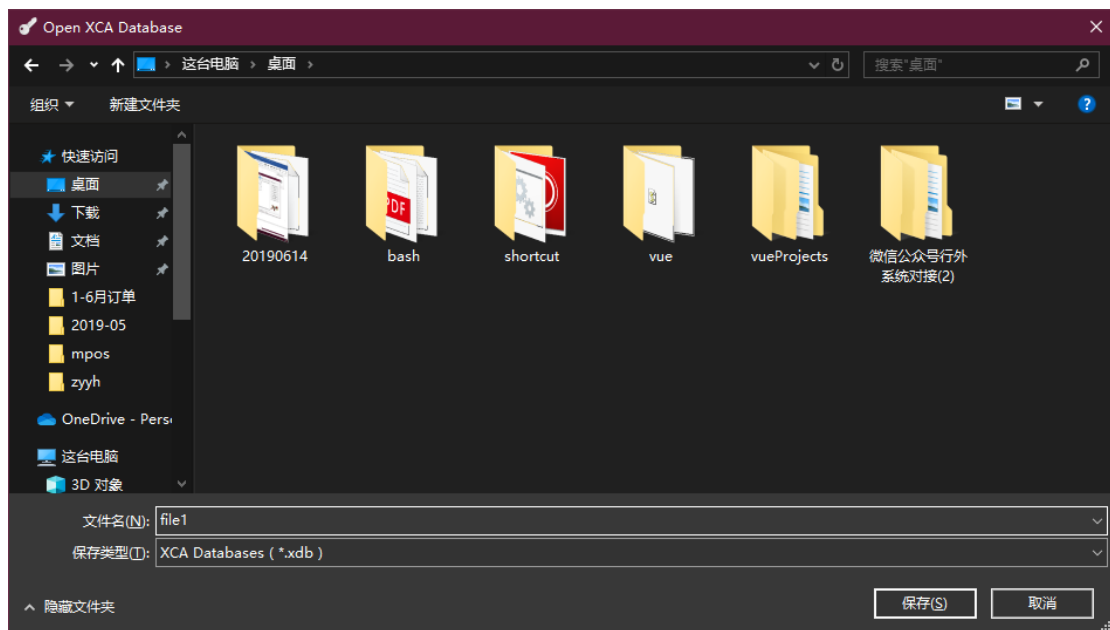
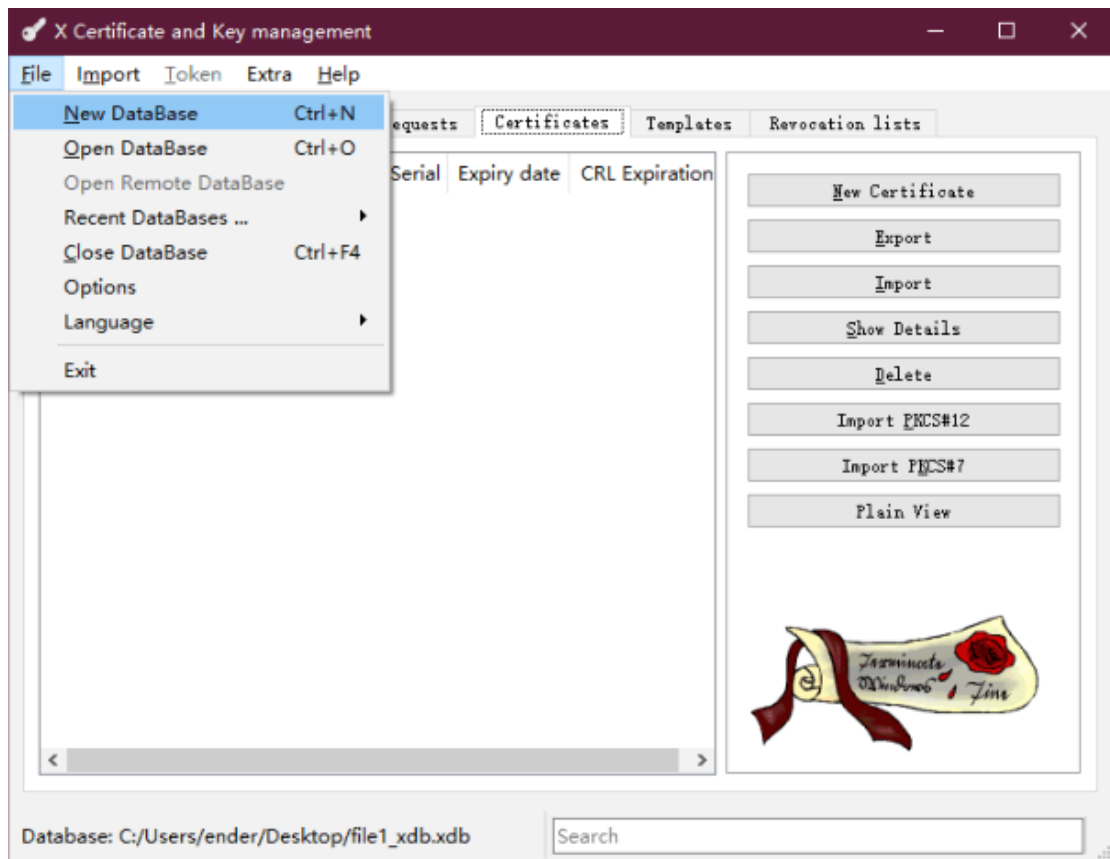
运行

C:\Aprograms\Agreen\xca-portable-2.1.2\xca-portable-2.1.2\xca.exe


后打开的界面




依次 File-- New DataBase,



选择 xdb 文件保存路径,输入文件名, 点击保存,

 New Password

**Password** 


Please enter a password, that will be used to encrypt your private keys in the database:  
C:/Users/ender/Desktop/file1.xdb

Password

Repeat Password

再输入密码（演示密码 123456）


点击 OK 确定后，回到主界面

 X Certificate and Key management

File Import Token Extra Help

Private Keys Certificate signing requests **Certificates** Templates Revocation lists

Internal name	commonName	CA	Serial	Expiry date	CRL Expiration
---------------	------------	----	--------	-------------	----------------



Database: C:/Users/ender/Desktop/file1\_xdb.xdb

切换到 Certificates 页面，点击 New Certificate

X Certificate and Key management

### Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

**Signing request**

☐ Sign this Certificate signing request

☒ Copy extensions from the request

☐ Modify subject of the request

**Signing**

☒ Create a self signed certificate

☐ Use this Certificate for signing

**Signature algorithm**

SHA 256

**Template for the new certificate**

[default] Empty template

Apply extensions Apply subject Apply all

OK Cancel

创建根证书,

在 source 栏, 签名算法选择 SHA 512, 证书模版选择默认 CA, 再点击 Apply all

修改任何内容记得保存

X Certificate and Key management

### Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

**Signing request**

☐ Sign this Certificate signing request

☒ Copy extensions from the request

☐ Modify subject of the request

**Signing**

☒ Create a self signed certificate

☐ Use this Certificate for signing

**Signature algorithm**

SHA 512

**Template for the new certificate**

[default] CA

Apply extensions Apply subject Apply all

OK Cancel

在 extensions 栏, type 选择 Certificate Authority

可选—time range 有效期设置 10 年后点击 apply



X Certificate and Key management

## Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

**X509v3 Basic Constraints**

Type: Certification Authority

Path length:  ☒ Critical

**Key identifier**

☒ Subject Key Identifier

☐ Authority Key Identifier

**Validity**

Not before: 2019-06-14 06:50 GMT

Not after: 2029-06-14 06:50 GMT

**Time range**

Years

☐ Midnight ☐ Local time ☐ No well-defined expiration

X509v3 Subject Alternative Name

X509v3 Issuer Alternative Name

X509v3 CRL Distribution Points

Authority Information Access: OCSP

Subject 栏，填好各个字段，都可以随便填

X Certificate and Key management

## Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

Internal Name: seeme

Distinguished name

countryName	china	organizationalUnitName	110
stateOrProvinceName	010	commonName	cme
localityName	cu	emailAddress	cme@cu.com
organizationName	c2cu		

Type	Content
------	---------

Add  
Delete

Private key

☐ Used keys too [Generate a new key](#)

OK Cancel

点击 Generate a new key 生产私钥

X Certificate and Key management

## New Key

Please give a name to the new key and select the desired keysize

Key properties

Name

Keytype


Keysize

☐ Remember as default

Create Cancel

不修改内容，点击 Create

X Certificate and Key management

 Successfully created the RSA private key 'seeme'

OK

然后点击新建证书的 OK'

X Certificate and Key management

## Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

Internal Name: seeme

Distinguished name

countryName	china	organizationalUnitName	110
stateOrProvinceName	010	commonName	cme
localityName	cu	emailAddress	cme@cu.com
organizationName	c2cu		

Type	Content
------	---------

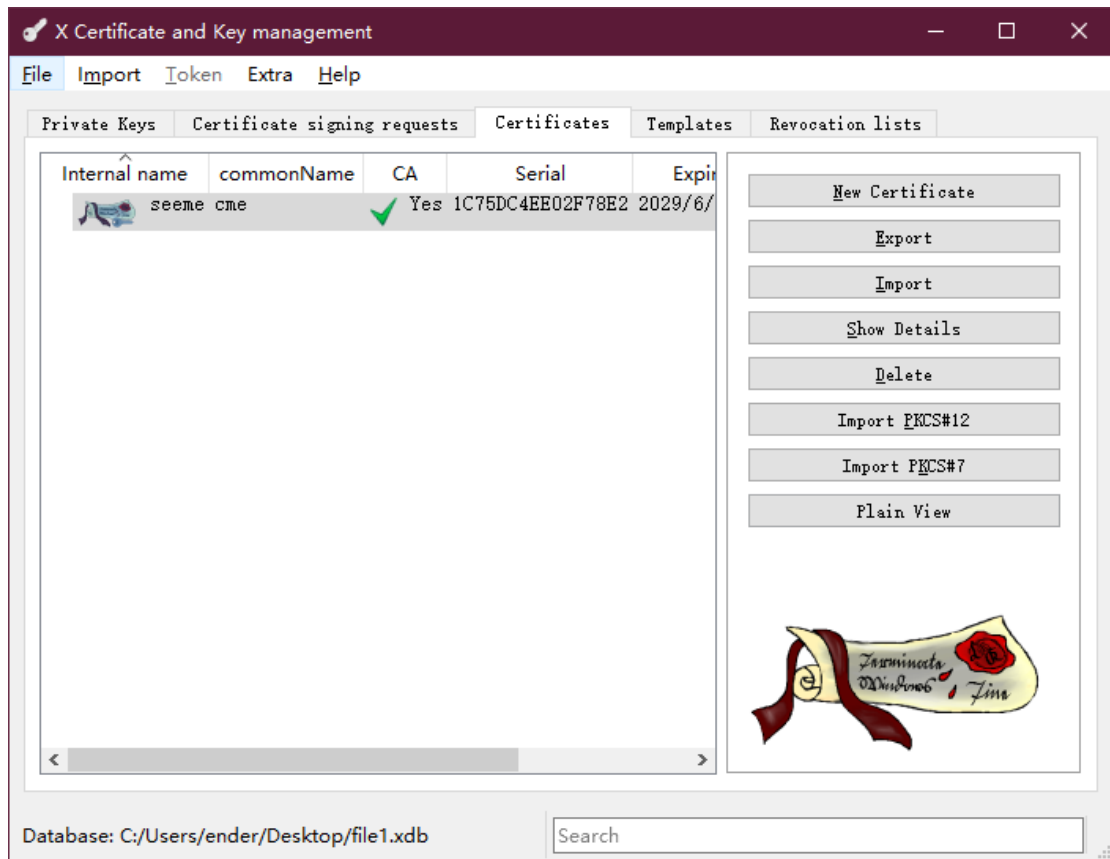
Add  
Delete

Private key

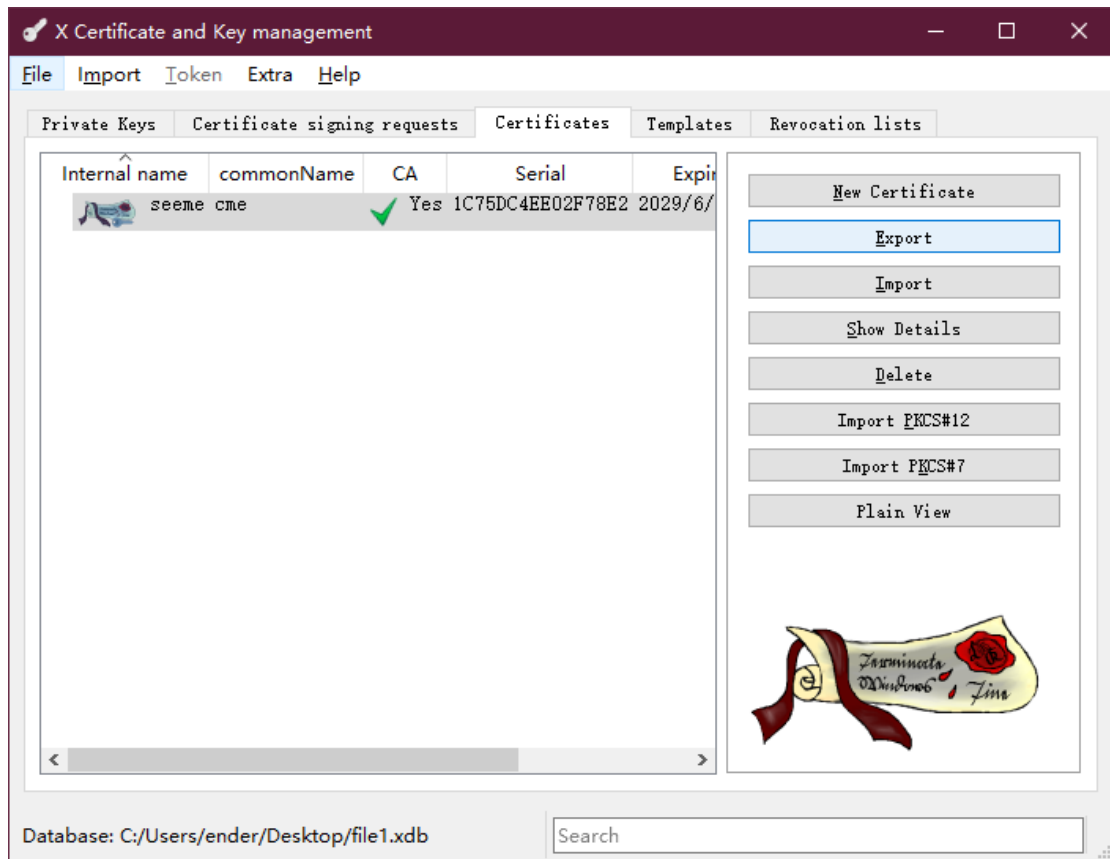
seeme (RSA:2048 bit) ☐ Used keys too [Generate a new key](#)

OK Cancel

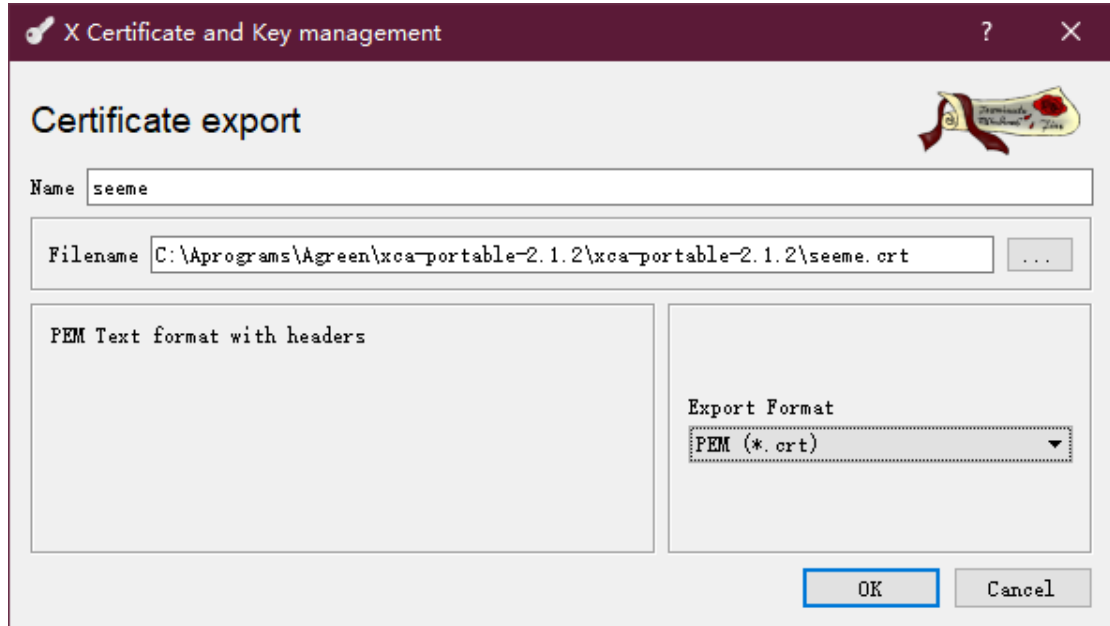
CA 证书做好了，有效期默认 10 年



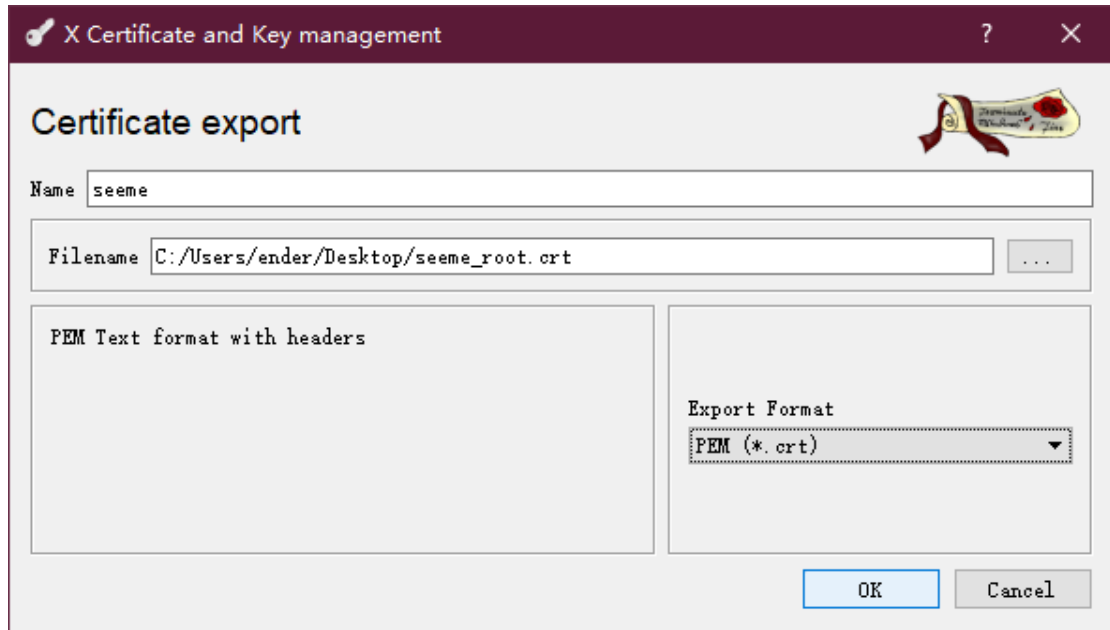
根证书导出成只包含公钥的证书格式，这本根证书就是放在网站上供用户下载安装，或主动安装到客户机器中的



选择创建好的根证书，点击导出 export



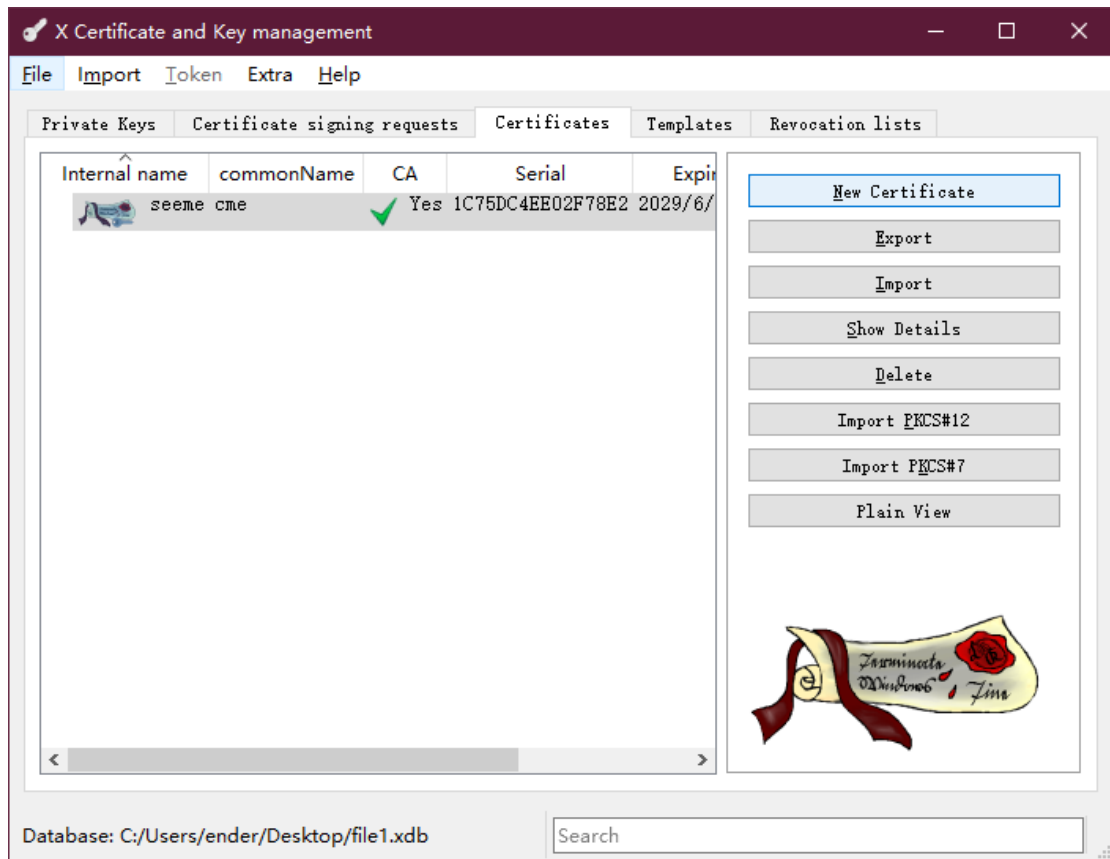
Filename 可以选择导出文件路径



点击 OK

制作服务器证书、客户端证书和制作 CA 证书差不多，只有两个地方不一样：

1 选择已经制作好的根 CA，然后点击 New Certificate1



2

签名时，选择使用根证书，这里是 seeme 根证书进行签名颁发，然后证书模版选择服务器 HTTPS Server（制作客户端证书就选择 HTTPS\_client），其他都和制作根证书一样，然后点击 Apply all（这个一定不能忘），然后再切到 Subject、Extension 页面填写相应的东西就 OK 了

Server 证书



X Certificate and Key management

?

×

Create x509 Certificate

Source

Subject

Extensions

Key usage

Netscape

Advanced

Comment

Signing request

☐ Sign this Certificate signing request

☒ Copy extensions from the request

Show request

☐ Modify subject of the request

Signing

☐ Create a self signed certificate

☒ Use this Certificate for signing

seeme

Signature algorithm

SHA 512

Template for the new certificate

[default] HTTPS\_server

Apply extensions

Apply subject

Apply all

OK

Cancel

X Certificate and Key management

### Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

X509v3 Basic Constraints

Type: End Entity

Path length:  ☒ Critical

Key identifier

☒ Subject Key Identifier

☐ Authority Key Identifier

Validity

Not before: 2019-06-14 07:12 GMT

Not after: 2020-06-13 07:12 GMT

Time range

365 Days

☐ Midnight ☐ Local time ☐ No well-defined expiration

X509v3 Subject Alternative Name ☒ DNS:copycn

X509v3 Issuer Alternative Name

X509v3 CRL Distribution Points

Authority Information Access: OCSP

Extensions 不修改

X Certificate and Key management

### Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

Internal Name: server

Distinguished name

countryName	server	organizationalUnitName	server
stateOrProvinceName	server	commonName	server
localityName	server	emailAddress	server
organizationName	server		

Type	Content
------	---------

Add  
Delete

Private key

Used keys too [Generate a new key](#)

OK Cancel

Subject 填入信息后点击 generate 生成证书

X Certificate and Key management

### New Key

Please give a name to the new key and select the desired keysize

Key properties

Name: server

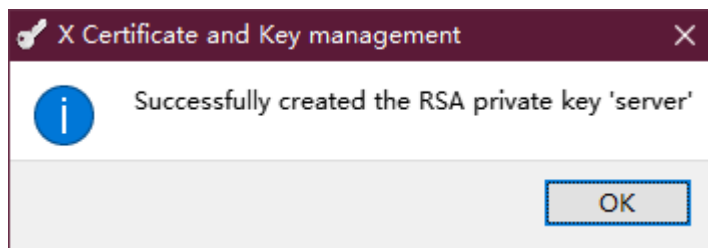
Keytype: RSA

Keysize: 2048 bit

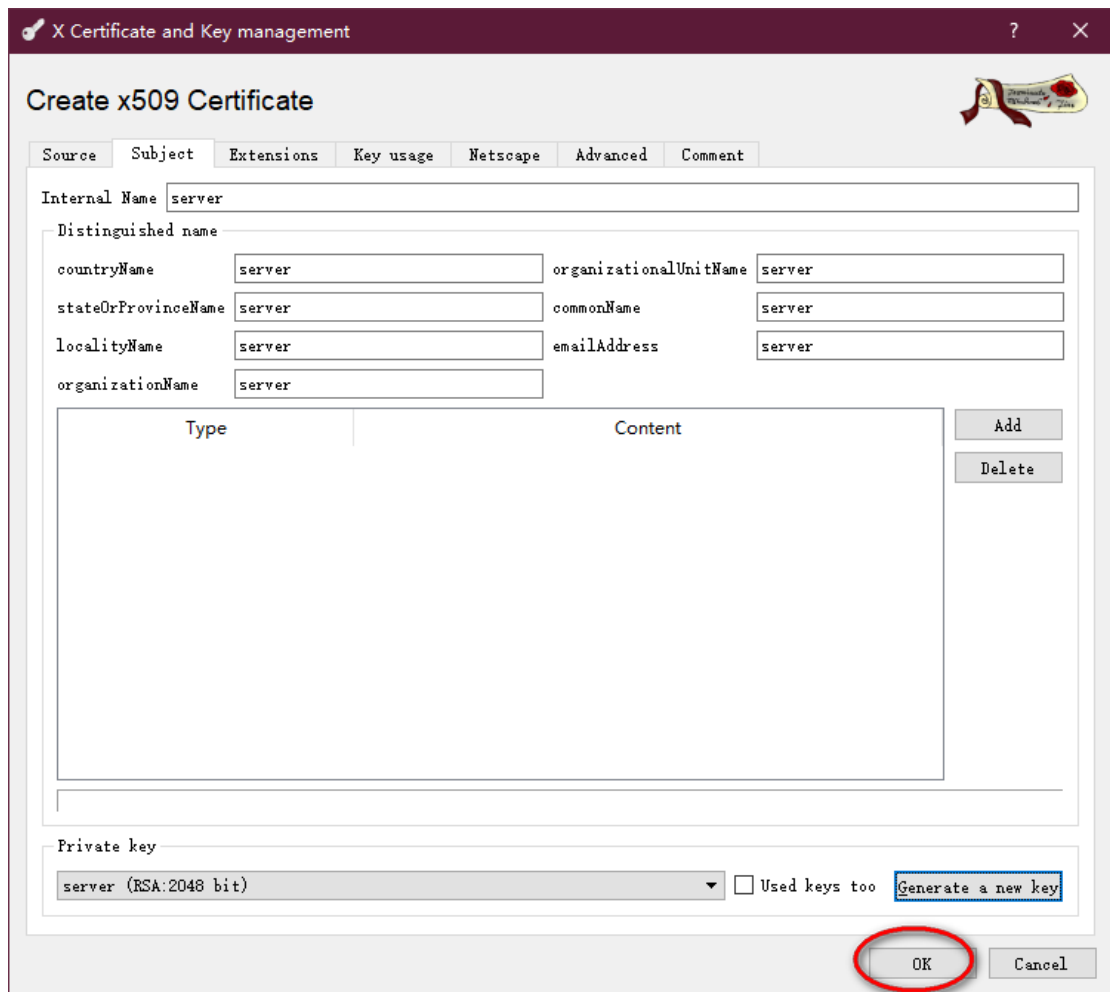
☐ Remember as default

Create Cancel

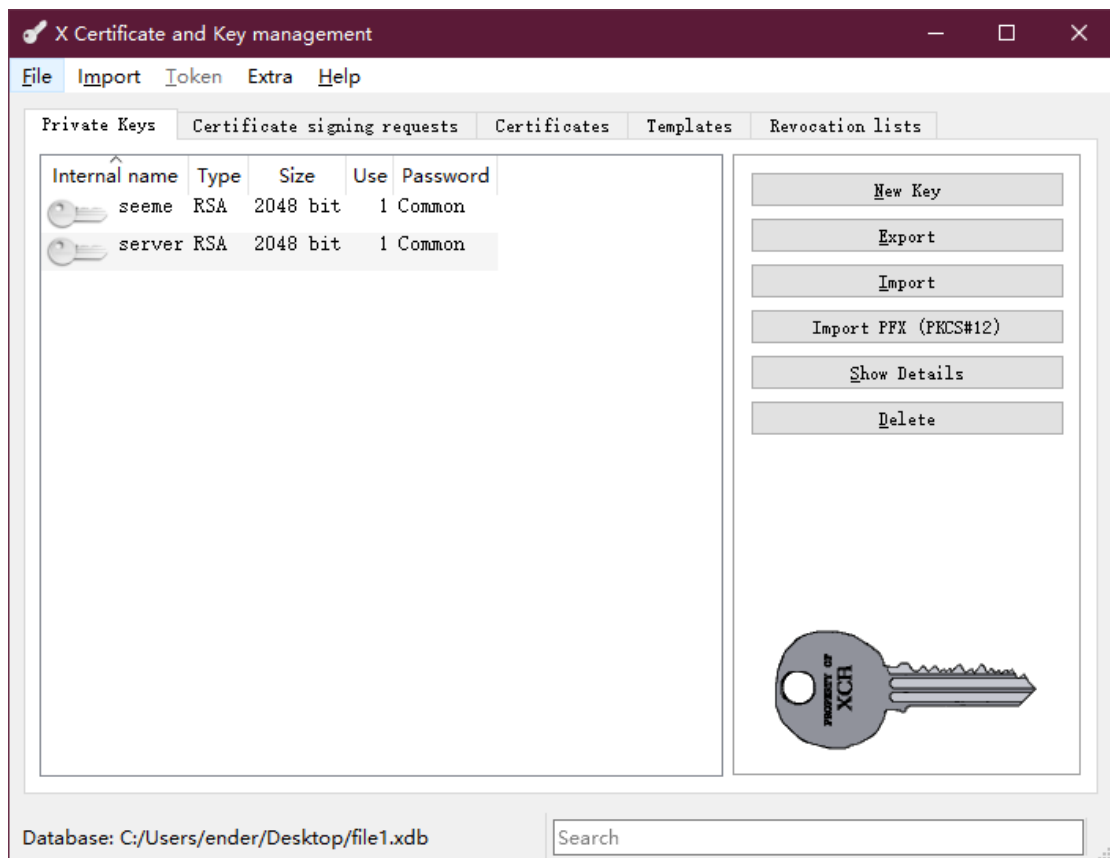
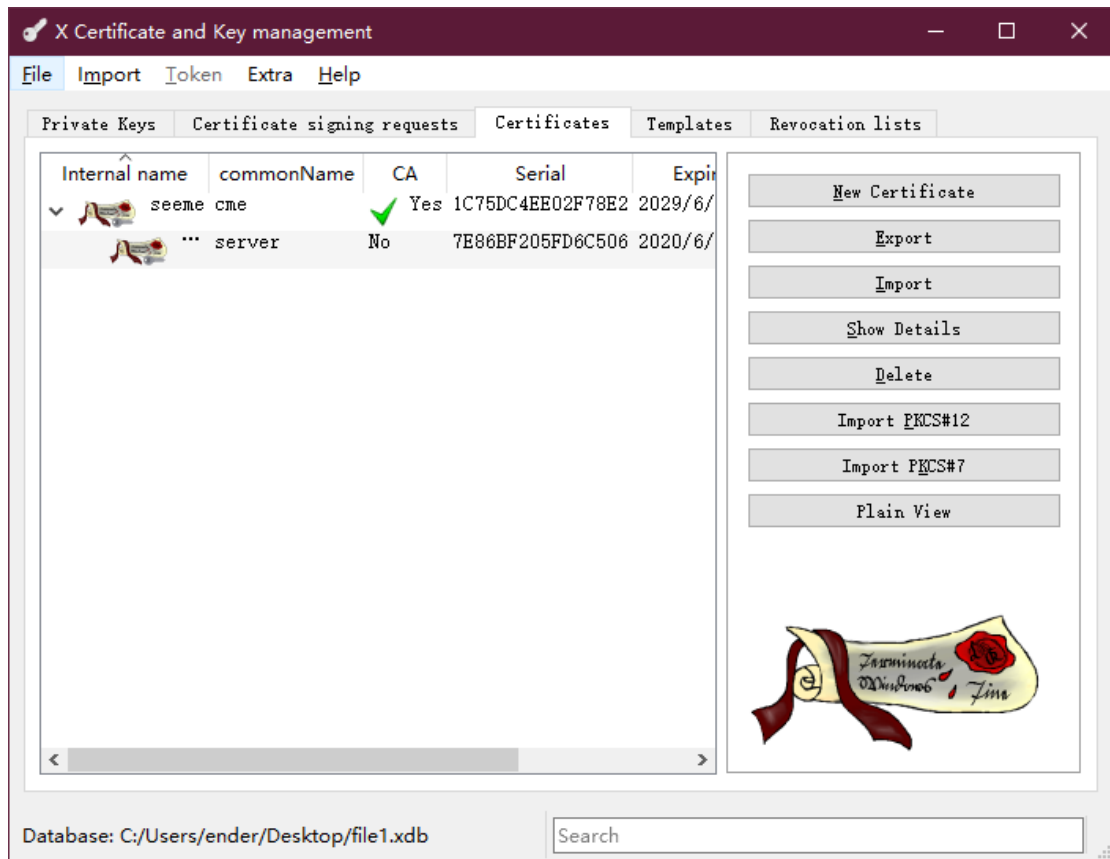
点击 create



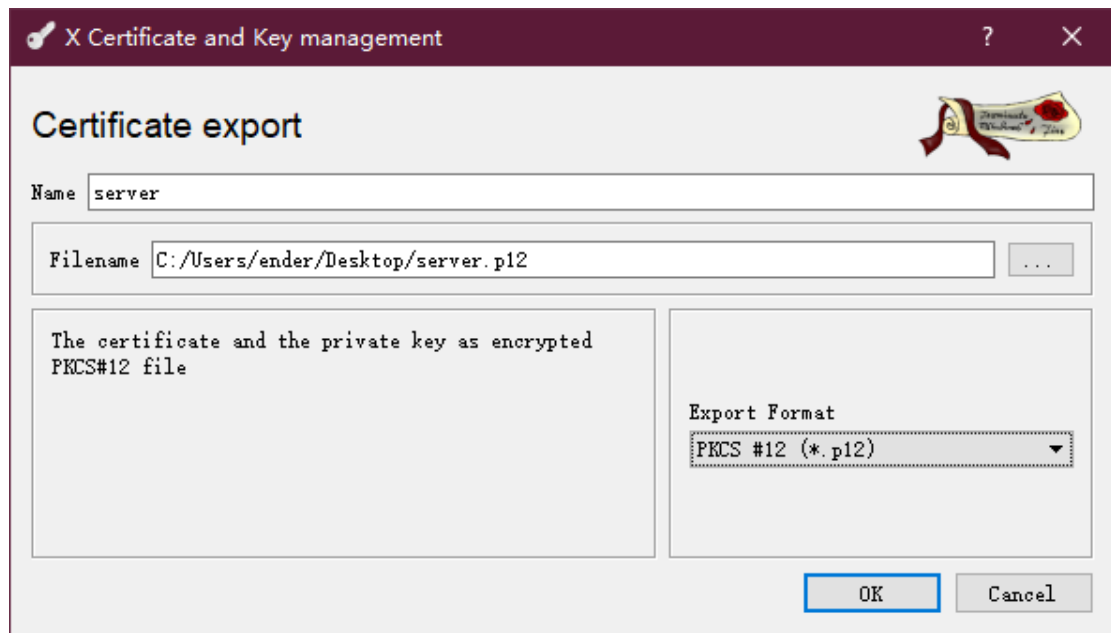
点击新建证书的 OK



Server 证书创建成功后的展示

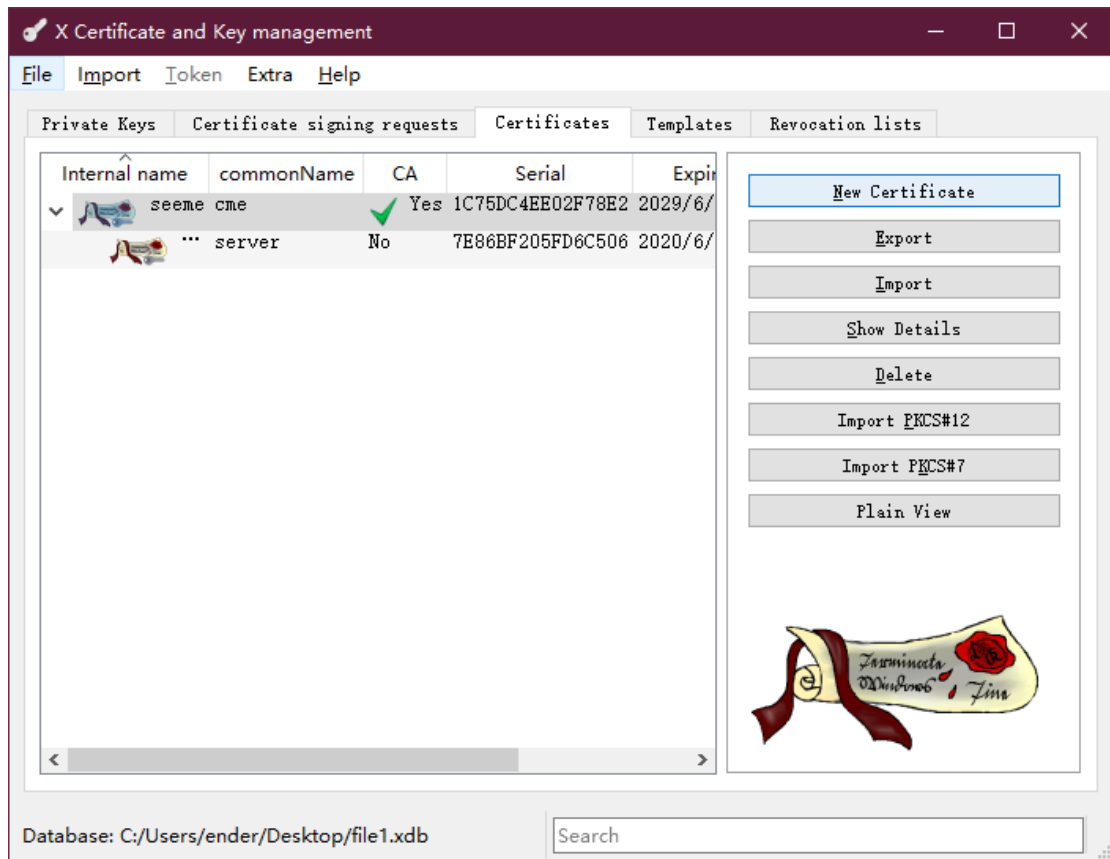


再将服务器证书导出来，选择 p12 格式



输入 server p12 导出的密码  
(演示的密码 112233)

新增 https client 证书



X Certificate and Key management

?

×

Create x509 Certificate

Source

Subject

Extensions

Key usage

Netscape

Advanced

Comment

Signing request

☐ Sign this Certificate signing request

☒ Copy extensions from the request

Show request

☐ Modify subject of the request

Signing

☐ Create a self signed certificate

☒ Use this Certificate for signing

seeme

Signature algorithm

SHA 512

Template for the new certificate

[default] HTTPS\_client

Apply extensions

Apply subject

Apply all

OK

Cancel



## Create x509 Certificate



Source Subject Extensions Key usage Netscape Advanced Comment

## X509v3 Basic Constraints

Type

End Entity

Path length

☒ Critical

## Key identifier

☒ Subject Key Identifier☐ Authority Key Identifier

## Validity

Not before

2019-06-14 07:24 GMT

Not after

2020-06-13 07:24 GMT

## Time range

365

Days

Apply

☐ Midnight☐ Local time☐ No well-defined expiration

X509v3 Subject Alternative Name

Edit

X509v3 Issuer Alternative Name

Edit

X509v3 CRL Distribution Points

Edit

Authority Information Access

OCSP

Edit

OK

Cancel

X Certificate and Key management

### Create x509 Certificate

Source Subject Extensions Key usage Netscape Advanced Comment

Internal Name

Distinguished name

countryName	<input type="text"/>	organizationalUnitName	<input type="text"/>
stateOrProvinceName	<input type="text"/>	commonName	<input type="text"/>
localityName	<input type="text"/>	emailAddress	<input type="text"/>
organizationName	<input type="text"/>		

Type	Content
------	---------

Private key

☐ Used keys too

点击右下角的 generate

X Certificate and Key management

### New Key

Please give a name to the new key and select the desired keysize

Key properties

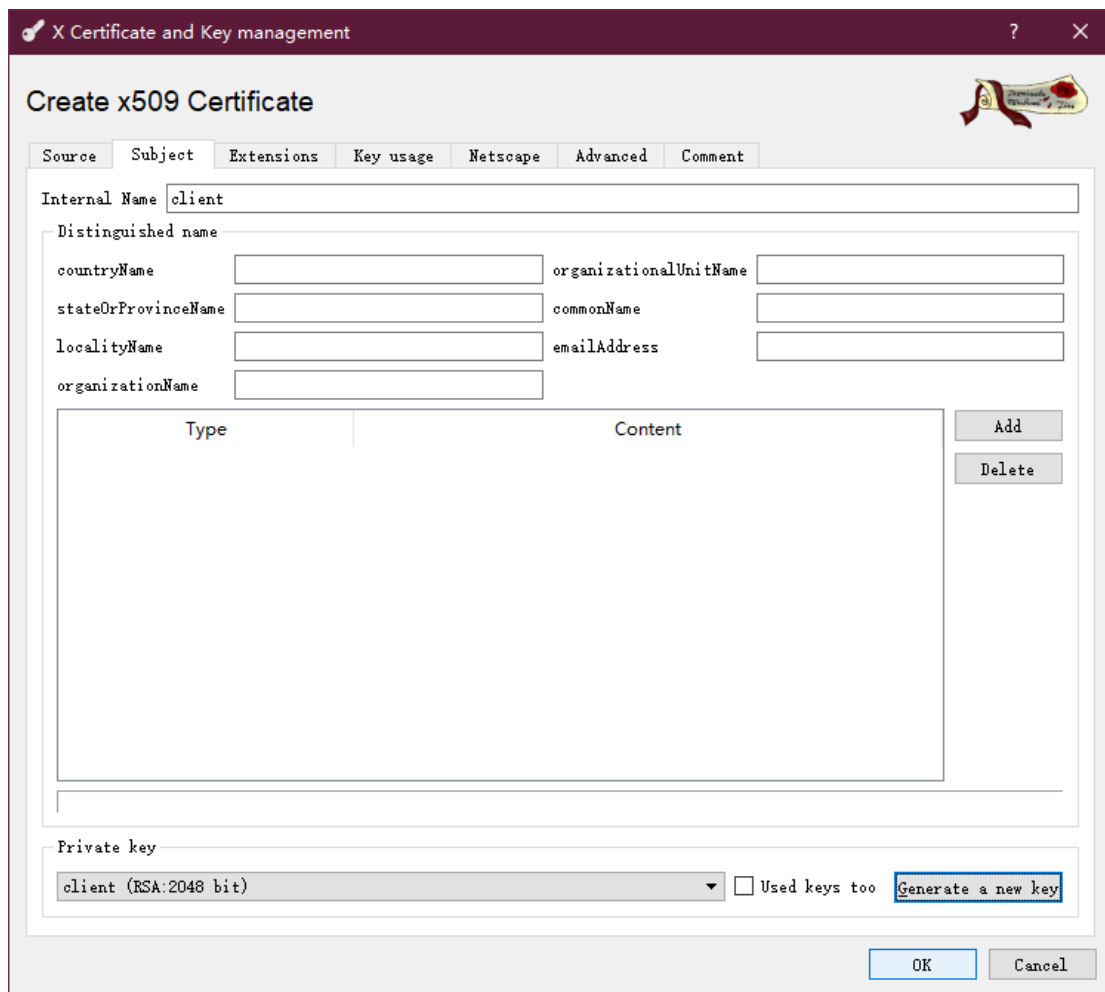
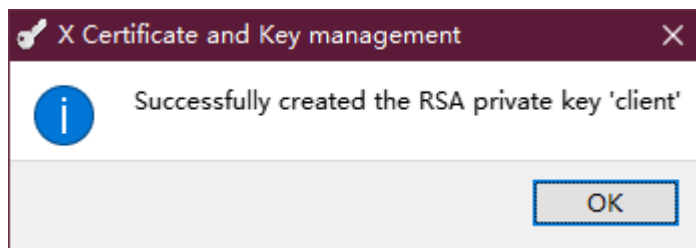
Name

Keytype

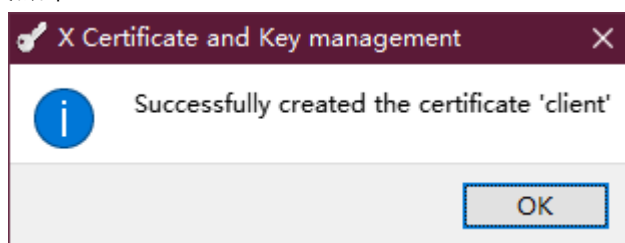
Keysize

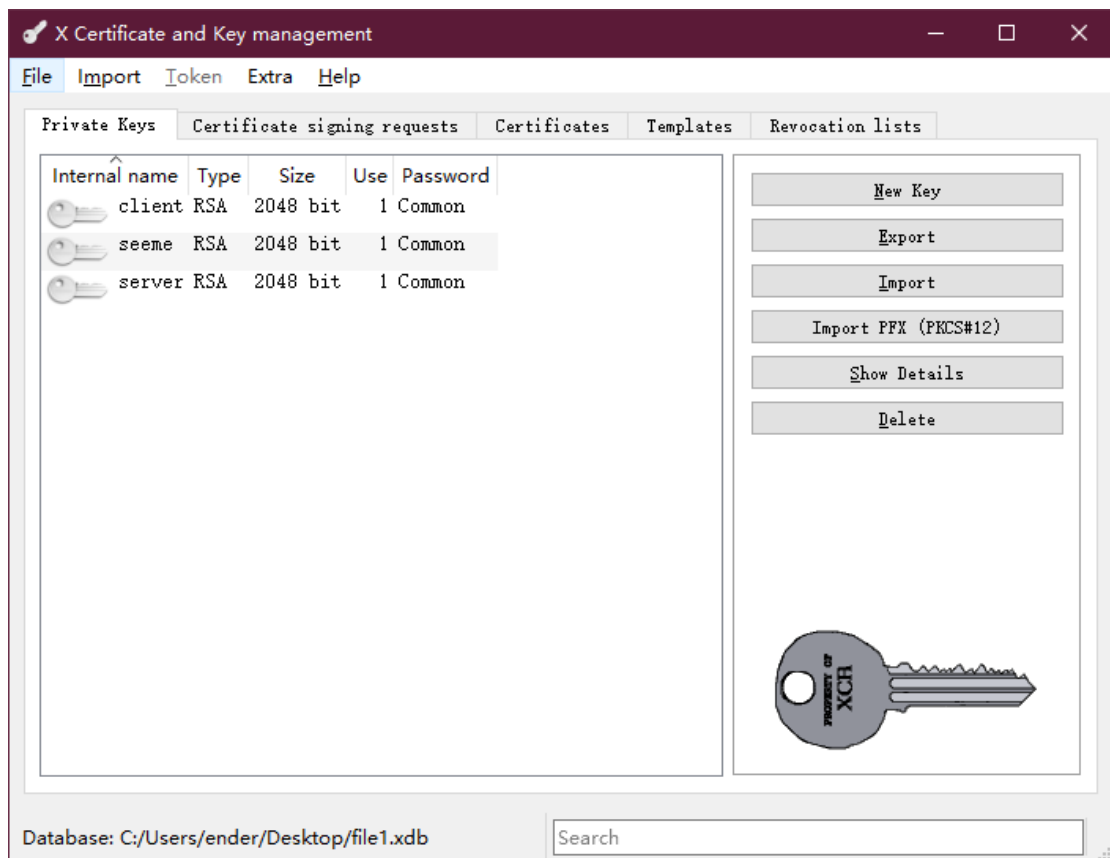
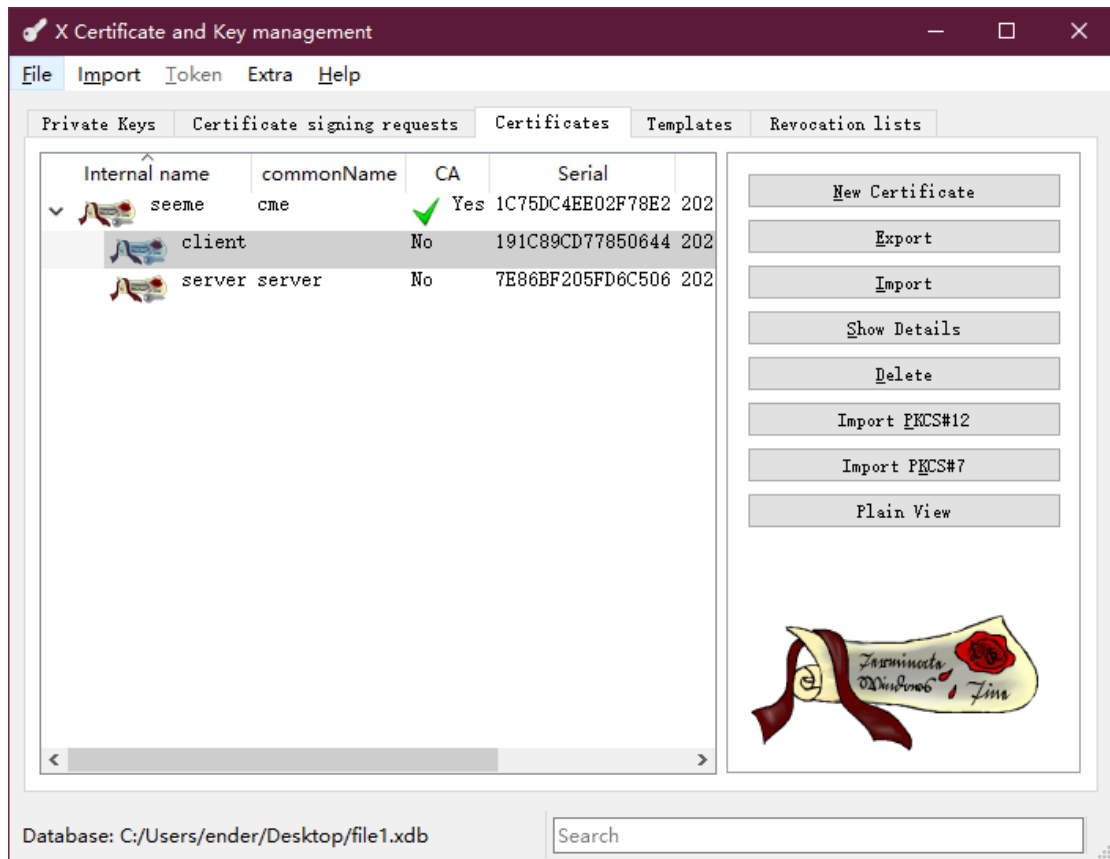
☐ Remember as default

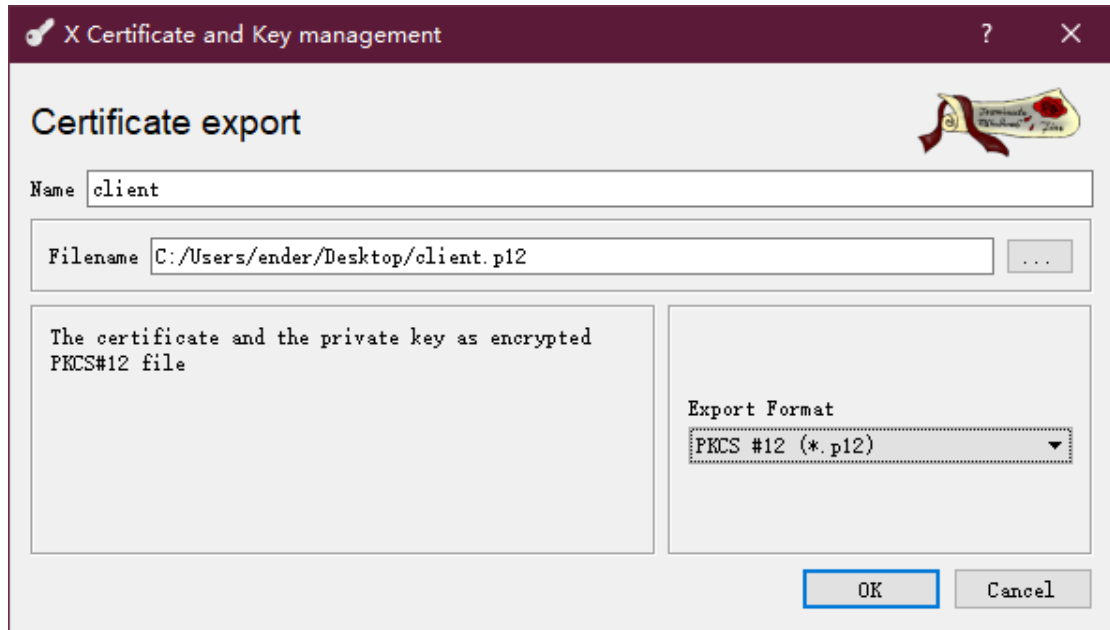
点击 create



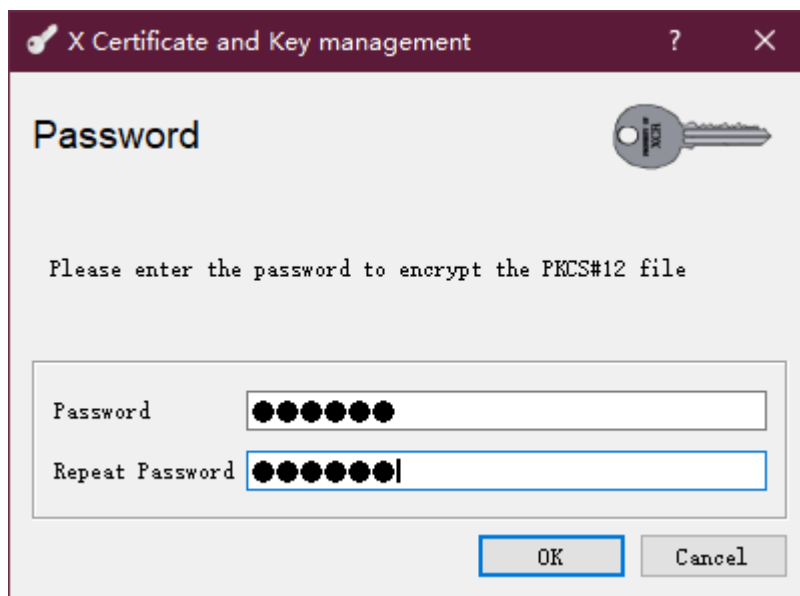
点击 OK



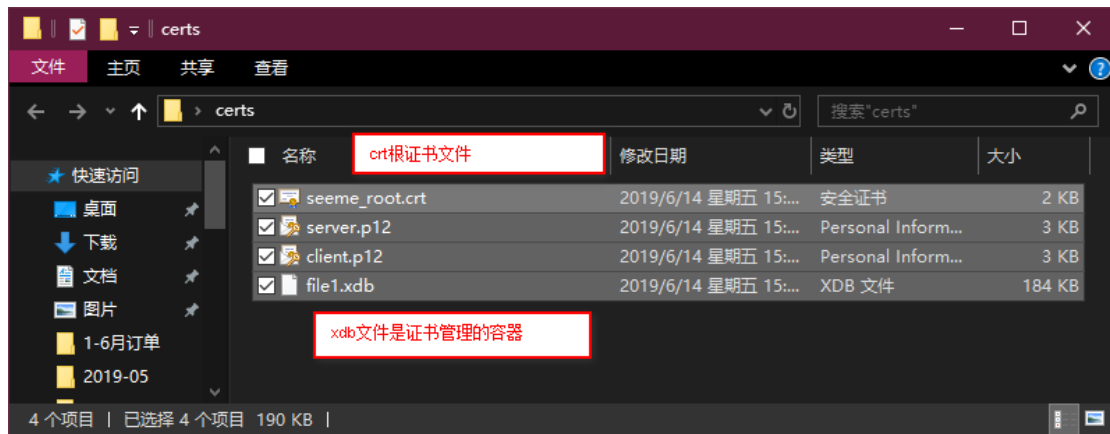




演示的密码 111222



所有的证书文件



## 演示的证书 passwords 密码

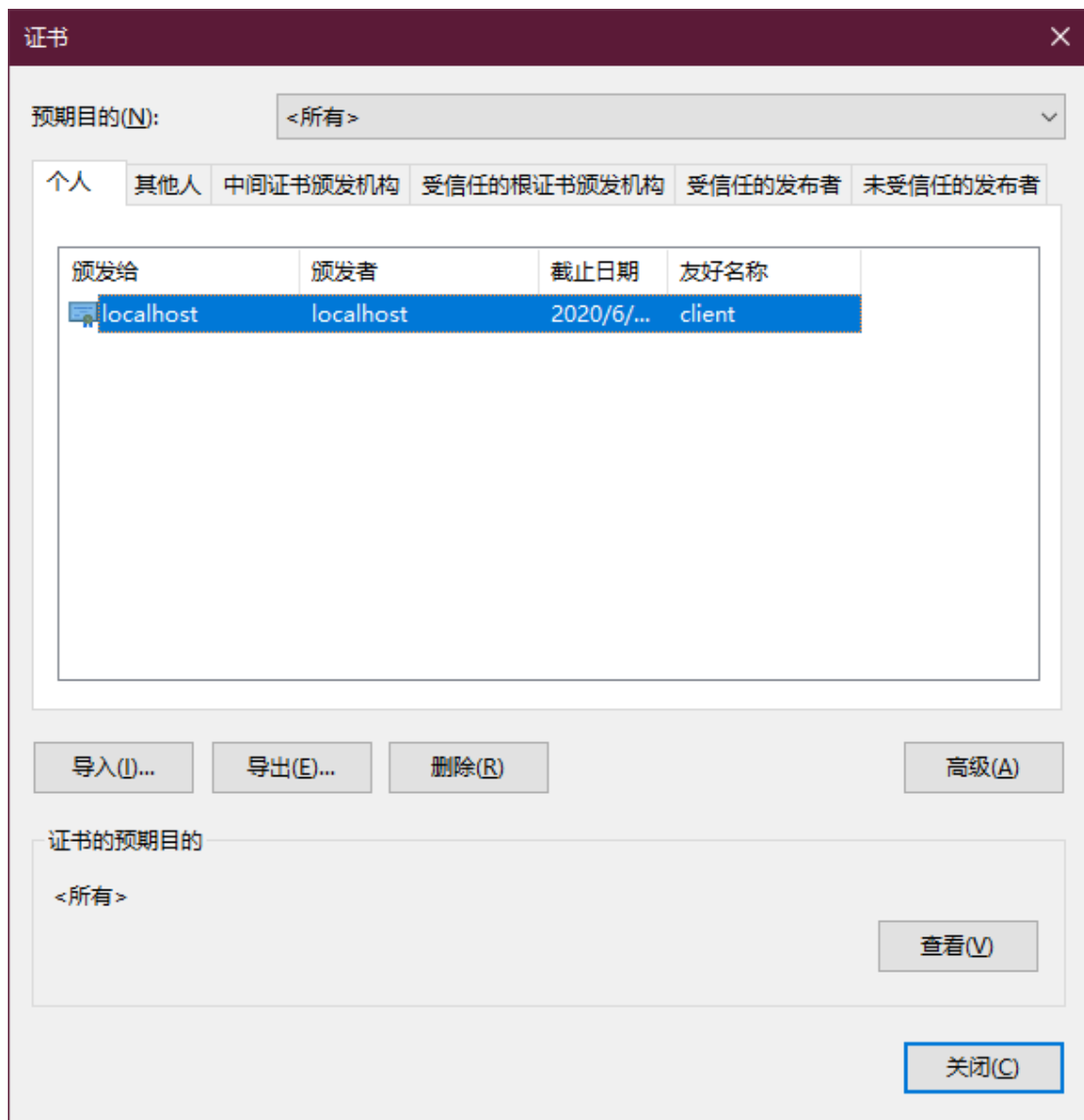
Xdb 证书容器 123456

服务器私钥 123123

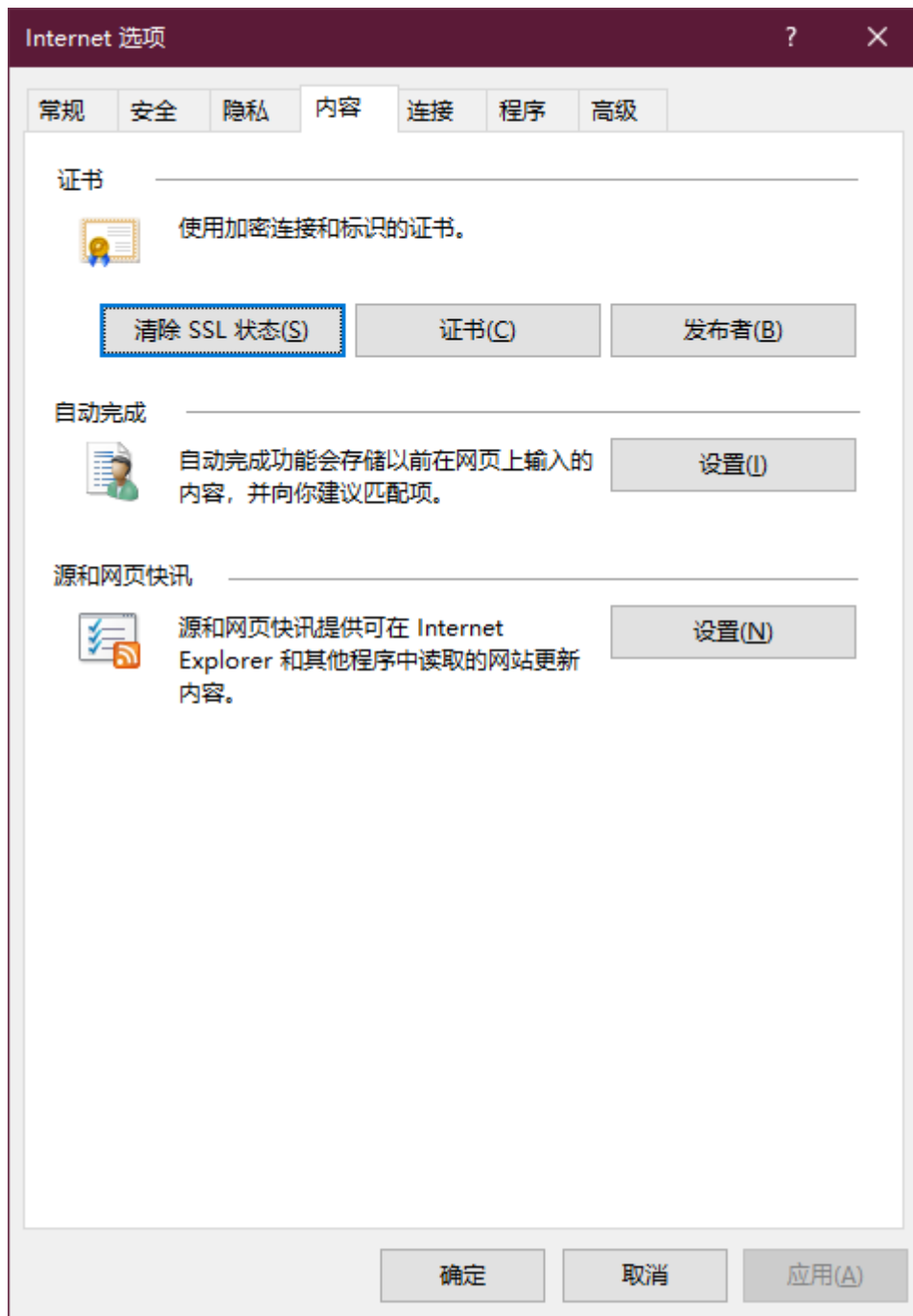
客户端私钥 111222

## 个人客户端证书导入注意

修改本地客户端证书（删除或者重新导入）



千万记得点击清除 SSL 状态



## tomcat 配置准备

作为双向认证支持的 web 容器，用作 java 实现的客户端和服务端交叉验证双向认证有效性



## Tomcat9 下载

<https://tomcat.apache.org/download-90.cgi>

导出的服务器 p12 证书放在 tomcat 的 conf/sslcerts 目录下，（当然也可以使用绝对路径）

## 修改默认配置，打开 https，设置单向认证

Conf/server.xml

Service 节点新增

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" secure="true" sslProtocol="TLS">
  <SSLHostConfig>
    <Certificate certificateKeystoreFile="conf/sslcerts/server.p12"
certificateKeystoreType="PKCS12" certificateKeystorePassword="123123" />
  </SSLHostConfig>
</Connector>
```

Conf/web.xml

welcome-file-list 节点后新增

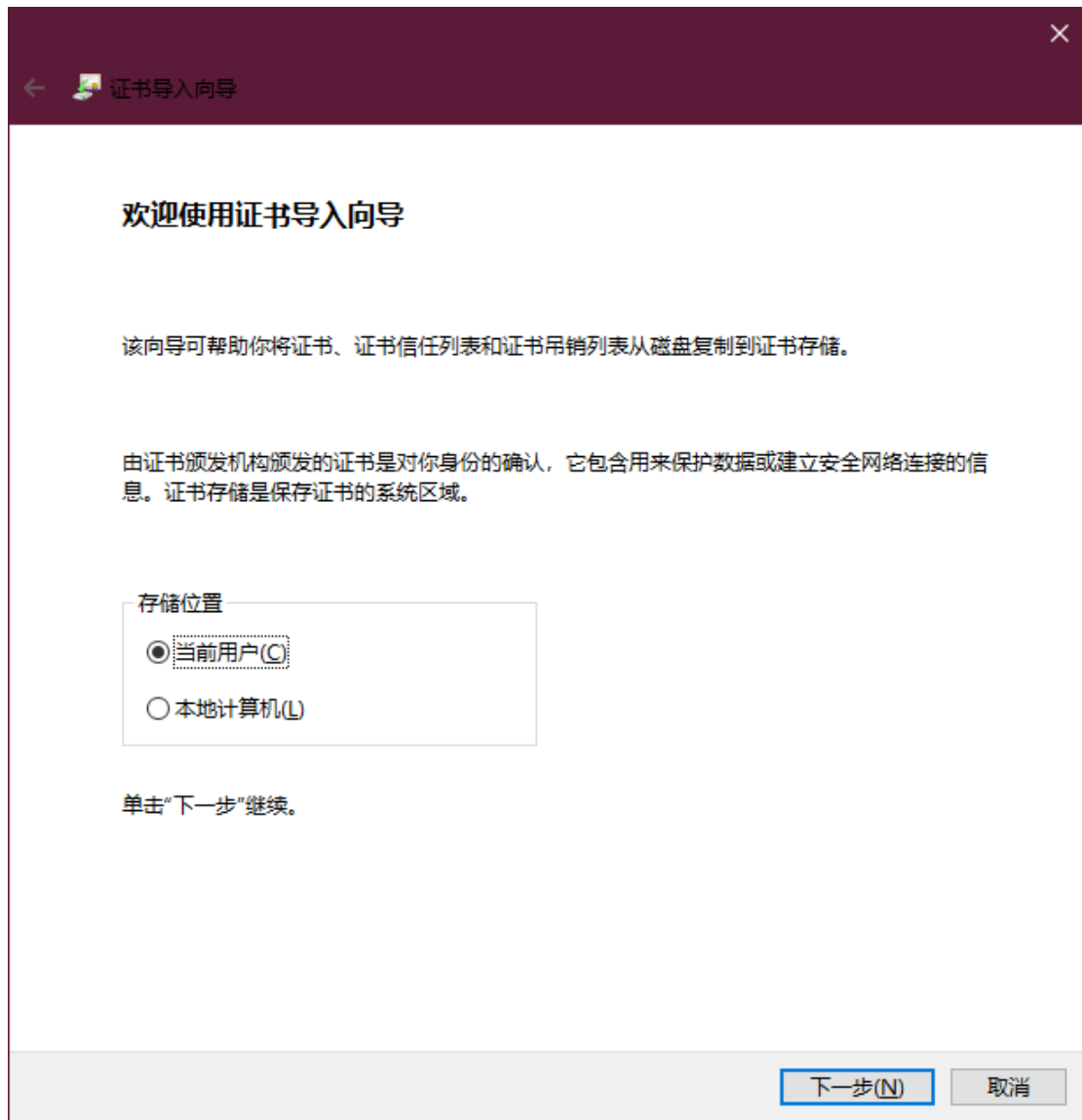
```
<login-config>
  <!-- Authorization setting for SSL -->
  <auth-method>CLIENT-CERT</auth-method>
  <realm-name>Client Cert Users-only Area</realm-name>
</login-config>
<security-constraint>
  <!-- Authorization setting for SSL -->
  <web-resource-collection >
    <web-resource-name >SSL</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

# 安装 CA 根证书

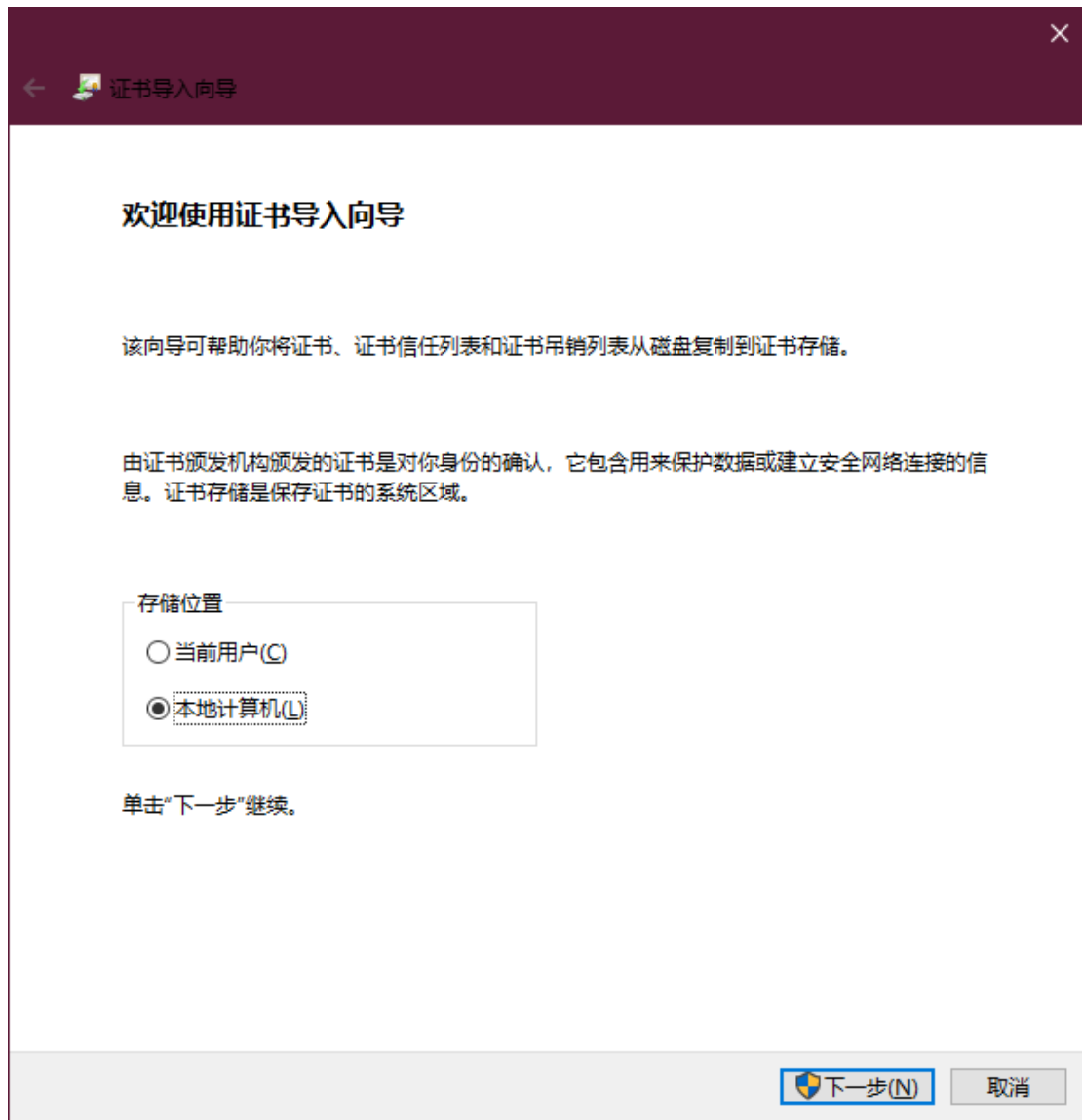
双击 root.crt

点击安装证书

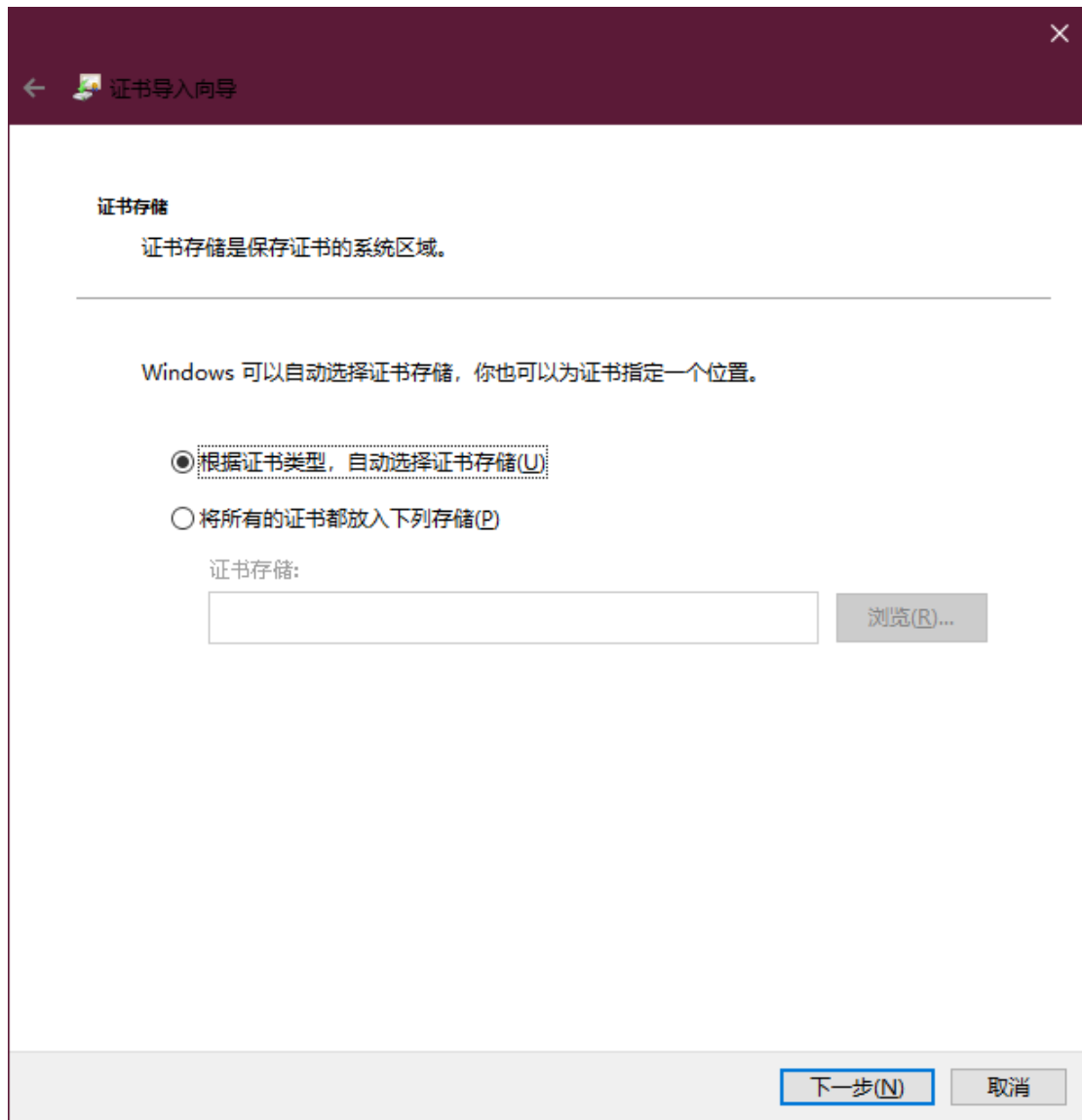




选择本地计算机

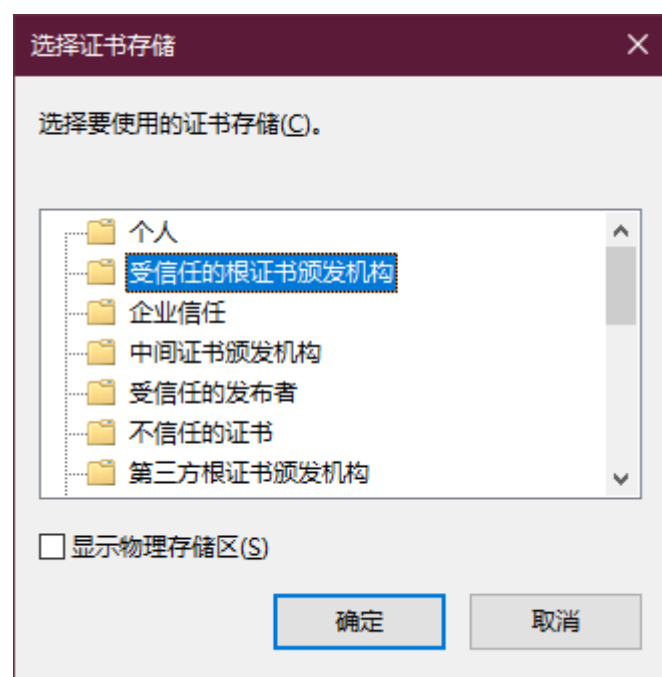


点下一步

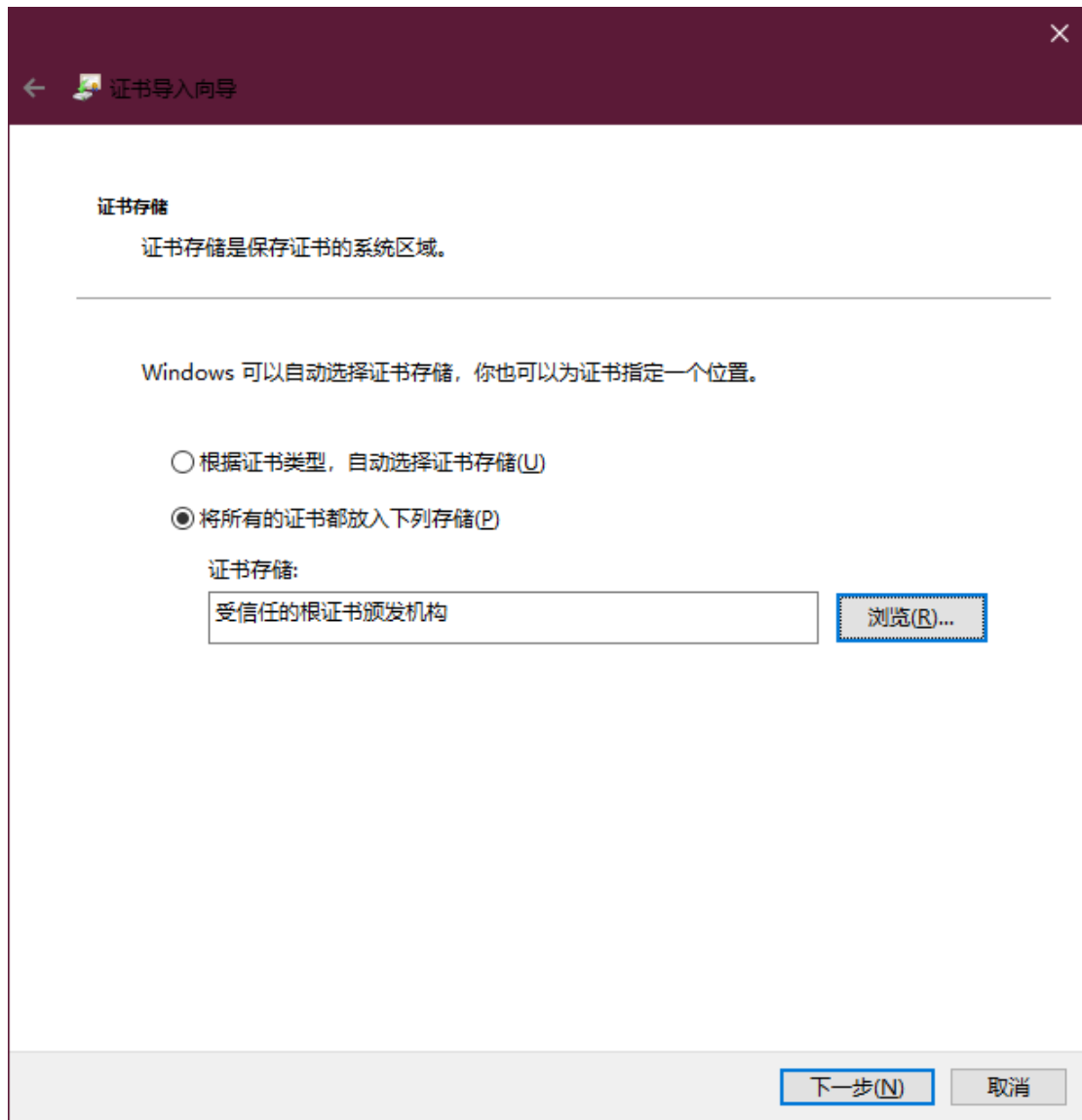


选择 将所有的证书都放入下列存储

点击浏览选择受信任的根证书机构



点击确定

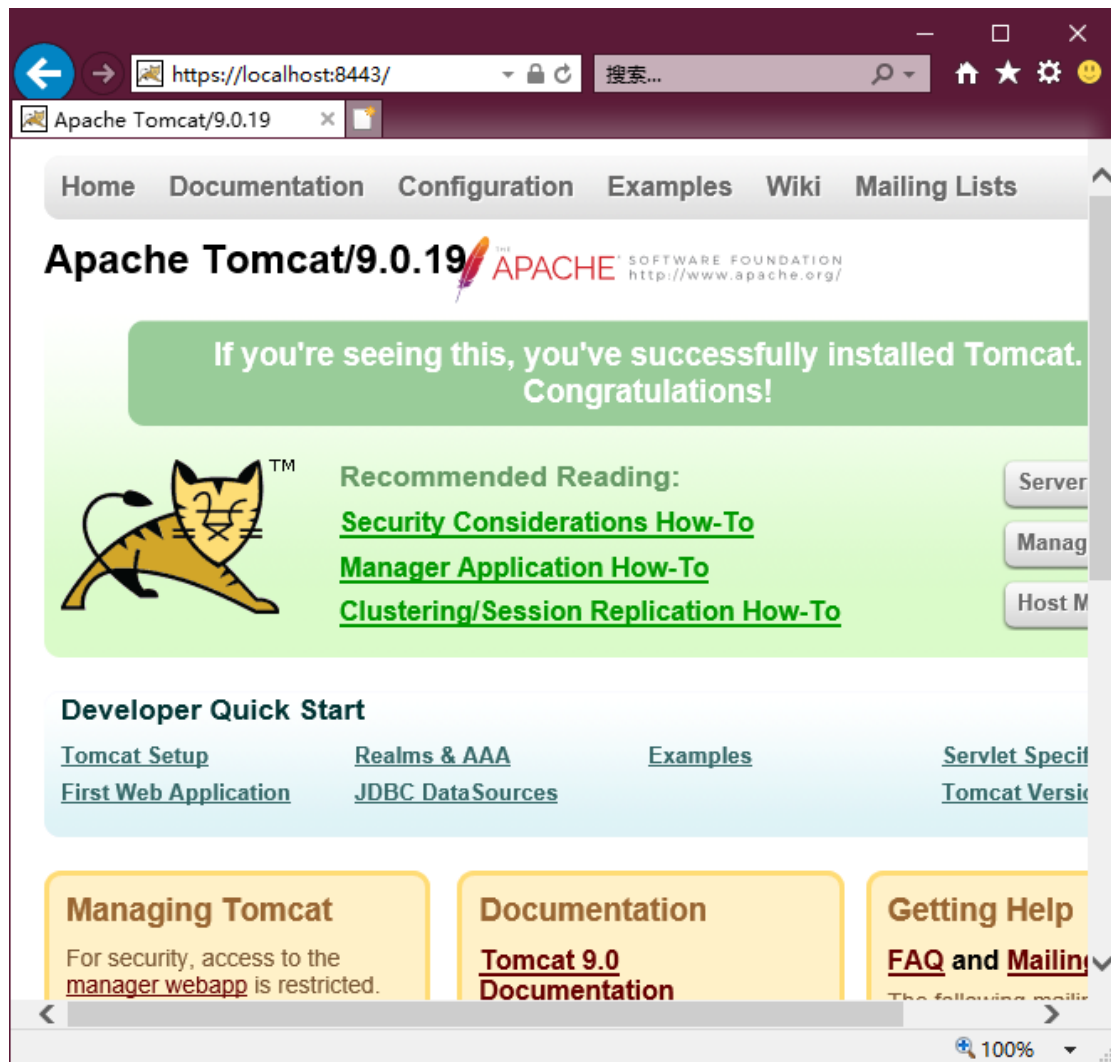


点击下一步直到完成

验证单向认证 tomcat

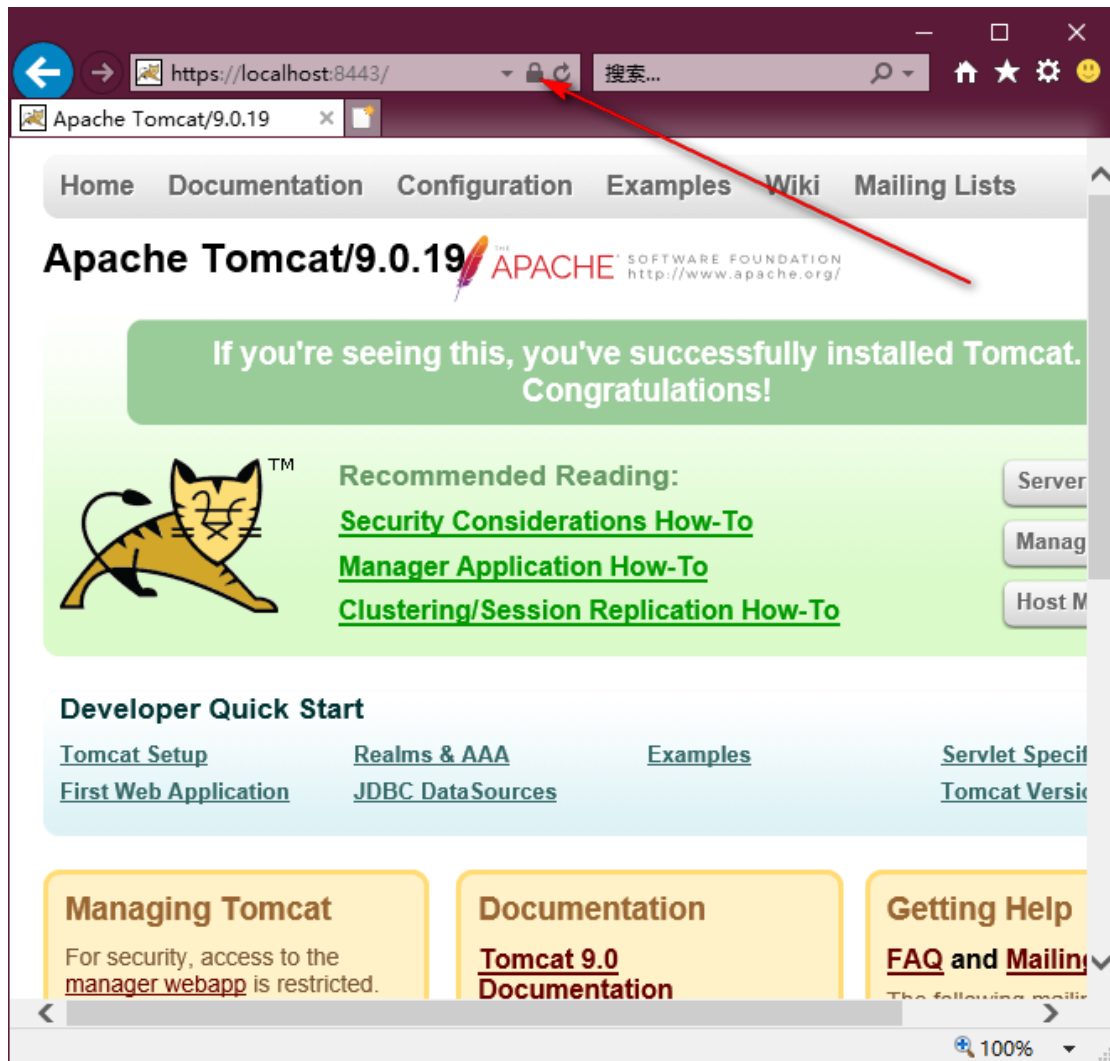
启动 tomcat

打开地址 <https://localhost:8443/>



点击证书查看按钮（IE 为一个锁的形状）

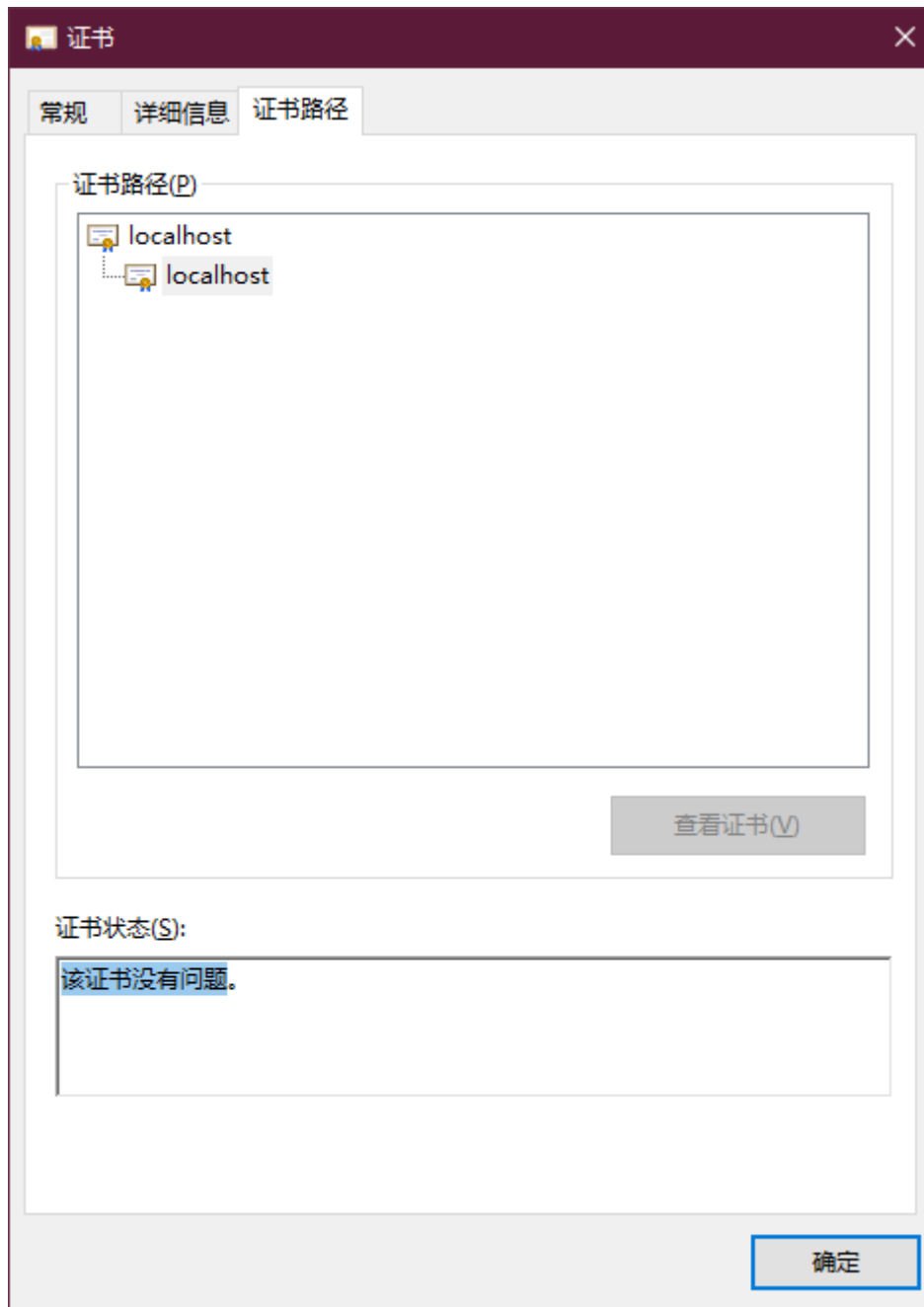




点击查看证书



点击证书路径



显示 该证书没有问题 就是正常的

## 配置双向认证 tomcat

## 配置需求说明

本次使用的 tomcat9

## 服务端

在原有服务的私钥证书配置好的基础上，新增证书容器（包含客户端证书的根证书，能认证客户端公钥有效性的）

## 客户端

在原有的（安装服务端证书的根证书后）认证服务端证书的同时，提供客户端公钥证书给服务端 ssl 请求

## Tomcat 配置修改

在新增的 connector 中

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
```

```
sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"
```

```
port="8443" maxThreads="200"
```

```
scheme="https" secure="true" SSLEnabled="true"
```

```
keystoreFile="conf/sslcerts/localhost.p12" keystorePass="123123"
```

```
truststoreFile="conf/sslcerts/trustme.jks" truststorePass="1q2w3e"
```

```
clientAuth="true" sslProtocol="TLS" />
```

重启 tomcat 后

## 双向验证服务端检查

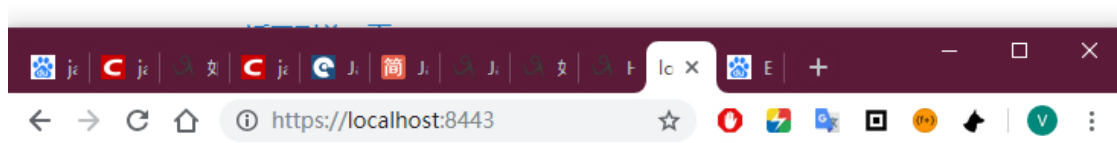
打开 IE（显示不明确），chrome(本地暂无客户端私钥，客户端证书暂时不提供)



## 无法安全地连接到此页面

这可能是因为该站点使用过期的或不安全的 TLS 安全设置。如果这种情况持续发生，请与网站系。

尝试此操作：



### This site can't provide a secure connection

**localhost** didn't accept your login certificate, or one may not have been provided.

Try contacting the system admin.

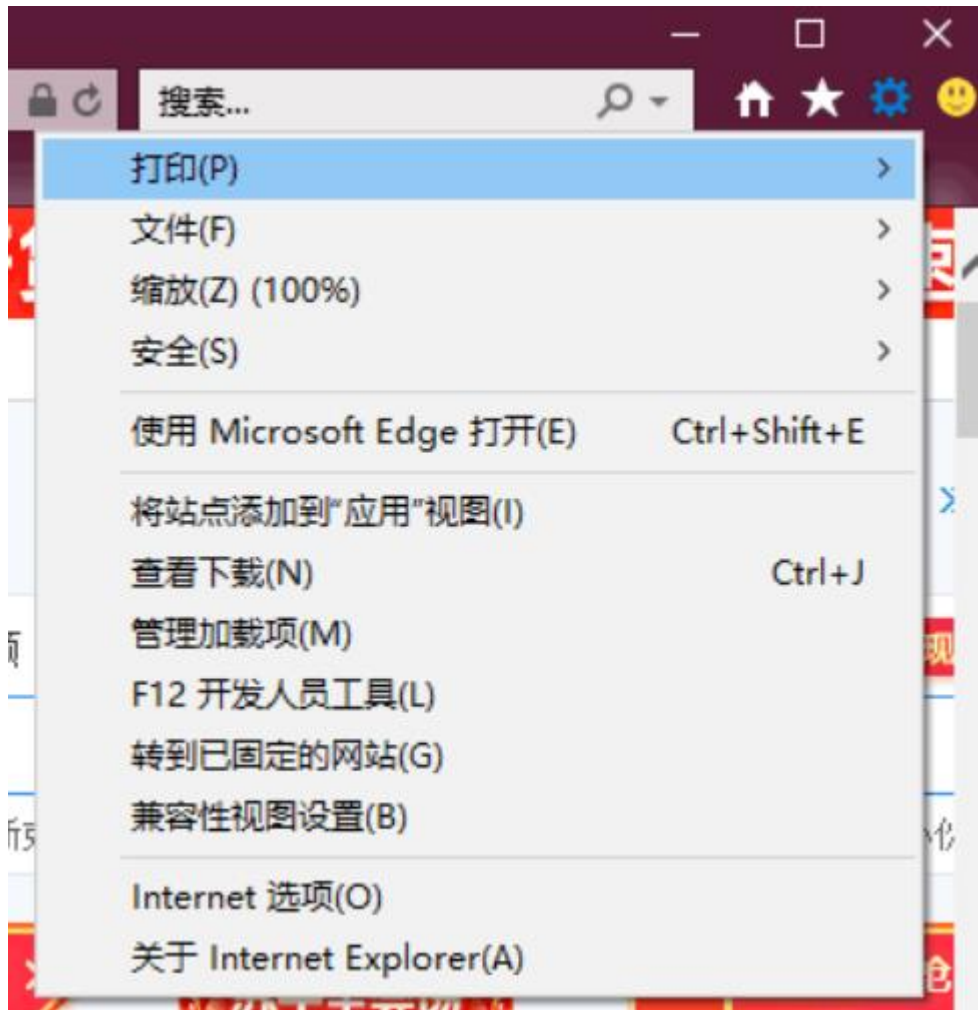
**ERR\_BAD\_SSL\_CLIENT\_AUTH\_CERT**

提示 ERR\_BAD\_SSL\_CLIENT\_AUTH\_CERT 表示当前浏览器访问 SSL 核验客户端证书失败

## 双向认证客户端检查

### 导入客户端证书到本机证书内容

IE - 工具 -



打开 internet 选项

Internet 选项

?

×

常规

安全

隐私


内容

连接

程序

高级

主页



若要创建多个主页标签页，请在每行输入一个地址(R)。

http://www.2345.com/?11319

^

v

使用当前页(C)

使用默认值(F)

使用新标签页(U)

启动

☐ 从上次会话中的标签页开始(B)

☒ 从主页开始(H)

标签页

更改网页在标签页中的显示方式。

标签页(I)

浏览历史记录

删除临时文件、历史记录、Cookie、保存的密码和网页表单信息。

☐ 退出时删除浏览历史记录(W)

删除(D)...

设置(S)

外观

颜色(O)

语言(L)

字体(N)

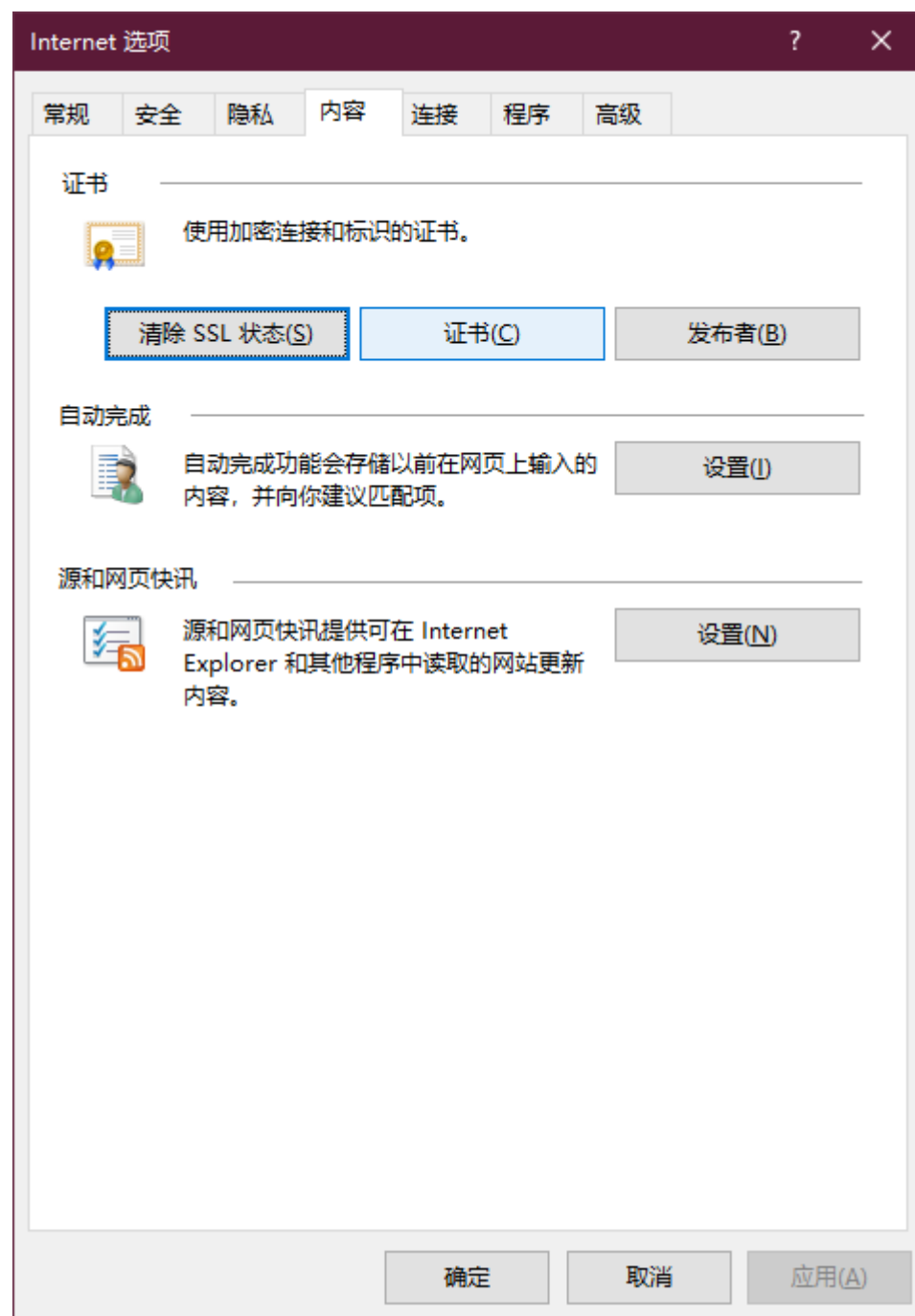
辅助功能(E)

确定

取消

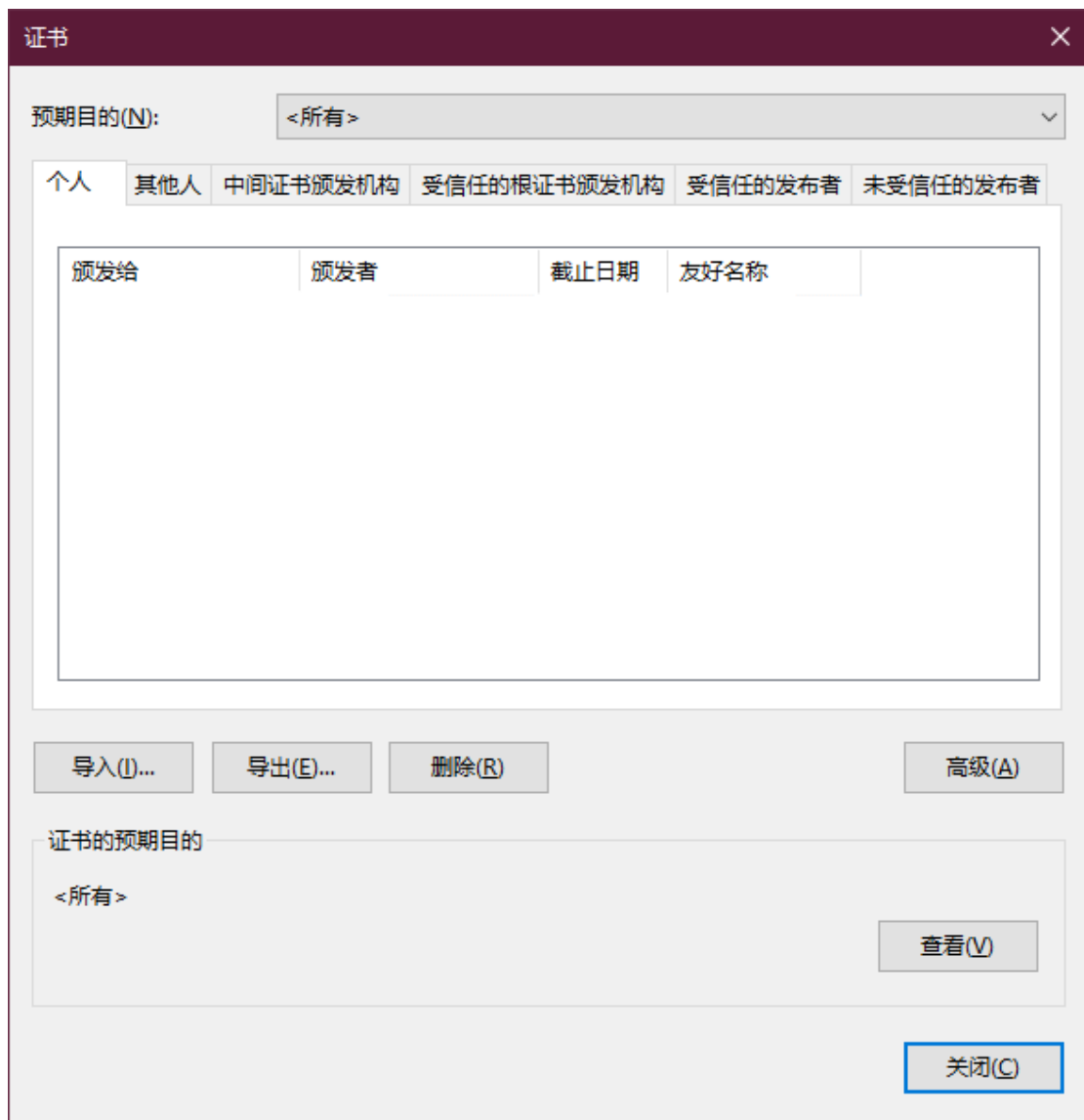
应用(A)

点击内容栏

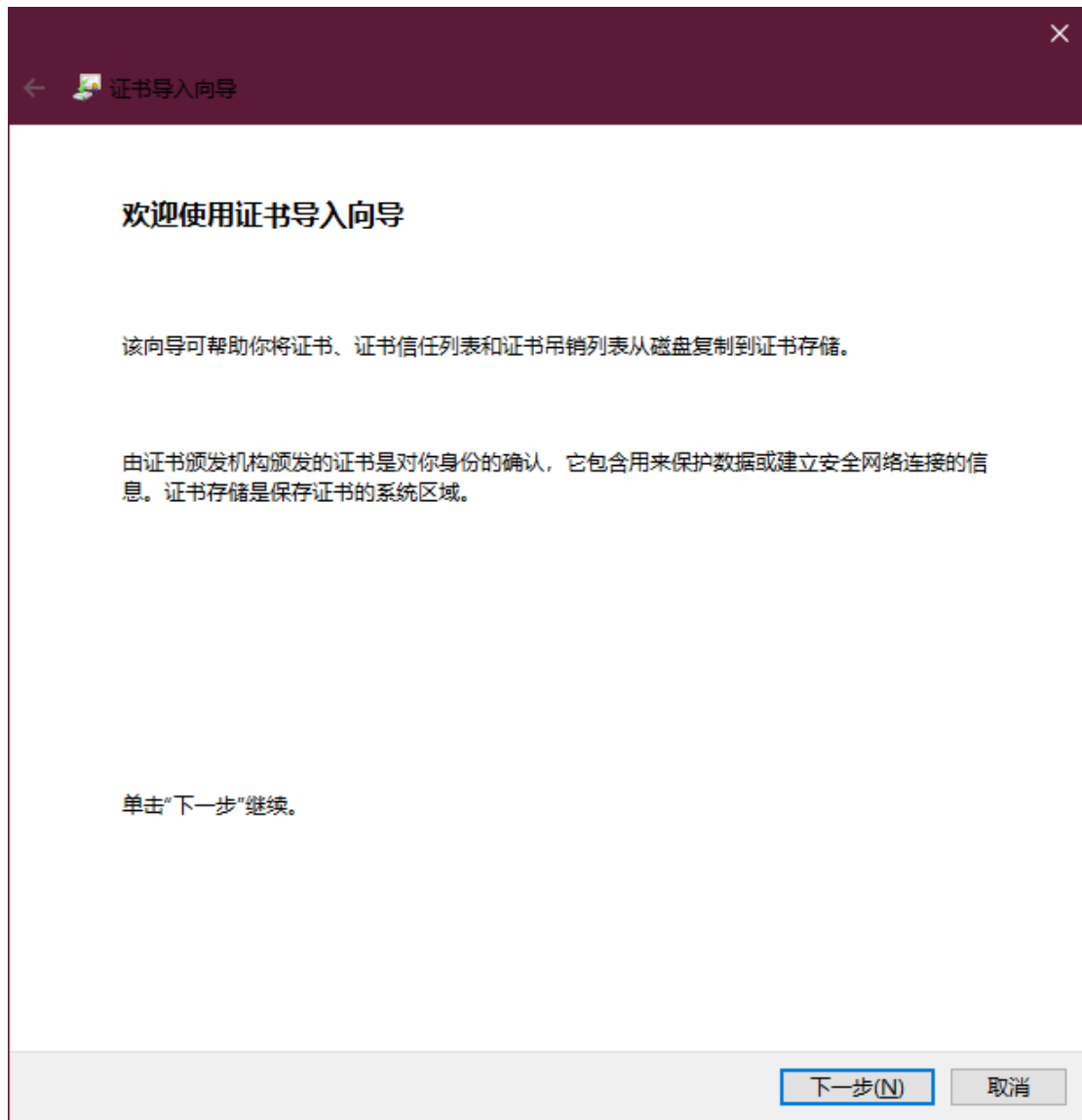


点击证书





点击导入证书



点击下一步

← 证书导入向导

要导入的文件

指定要导入的文件。

文件名(F):

浏览(R)...

注意: 用下列格式可以在一个文件中存储多个证书:

个人信息交换- PKCS #12 (.PFX,.P12)

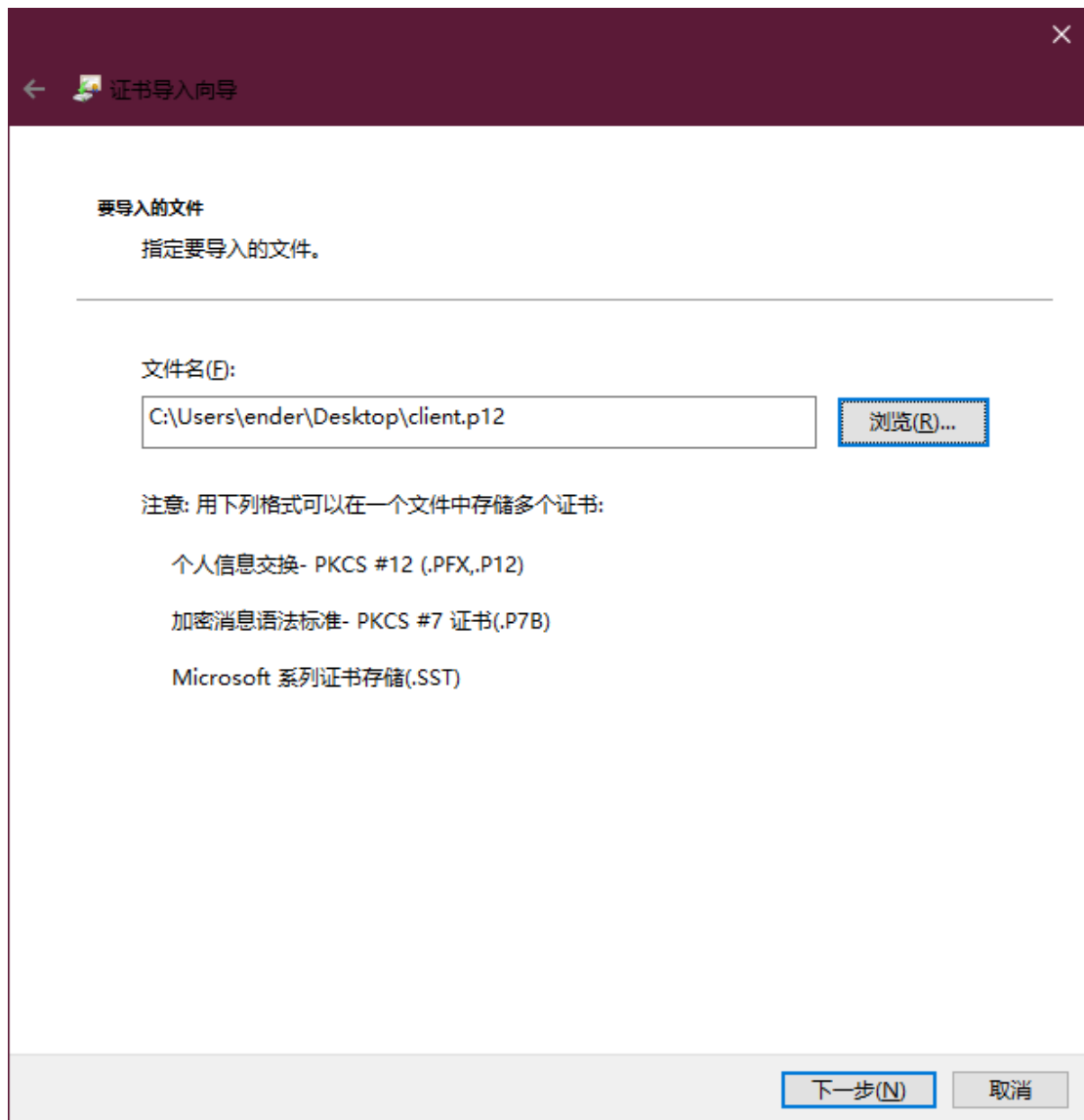
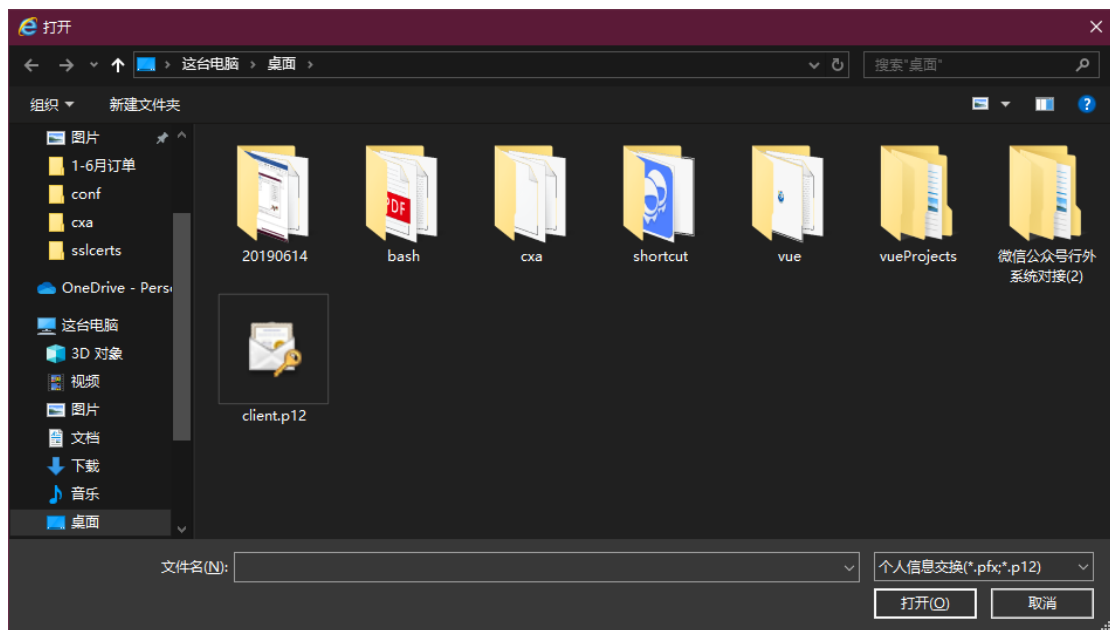
加密消息语法标准- PKCS #7 证书(.P7B)

Microsoft 系列证书存储(.SST)

下一步(N)

取消

点击浏览选择客户端私钥文件



点击下一步

← 证书导入向导

×

私钥保护

为了保证安全，已用密码保护私钥。

为私钥键入密码。

密码(P):

☐ 显示密码(D)

导入选项(I):

☐ 启用强私钥保护(E)。如果启用这个选项，每次应用程序使用私钥时，你都会收到提示。

☐ 标志此密钥为可导出的密钥(M)。这将允许你在稍后备份或传输密钥。

☐ 使用虚拟化安全保护私钥(不可导出)(P)

☒ 包括所有扩展属性(A)。

下一步(N)

取消

输入客户端私钥密码

← 证书导入向导

×

私钥保护

为了保证安全，已用密码保护私钥。

为私钥键入密码。

密码(P):

•••••

☐ 显示密码(D)

导入选项(I):

☒ 启用强私钥保护(E)。如果启用这个选项，每次应用程序使用私钥时，你都会收到提示。

☐ 标志此密钥为可导出的密钥(M)。这将允许你在稍后备份或传输密钥。

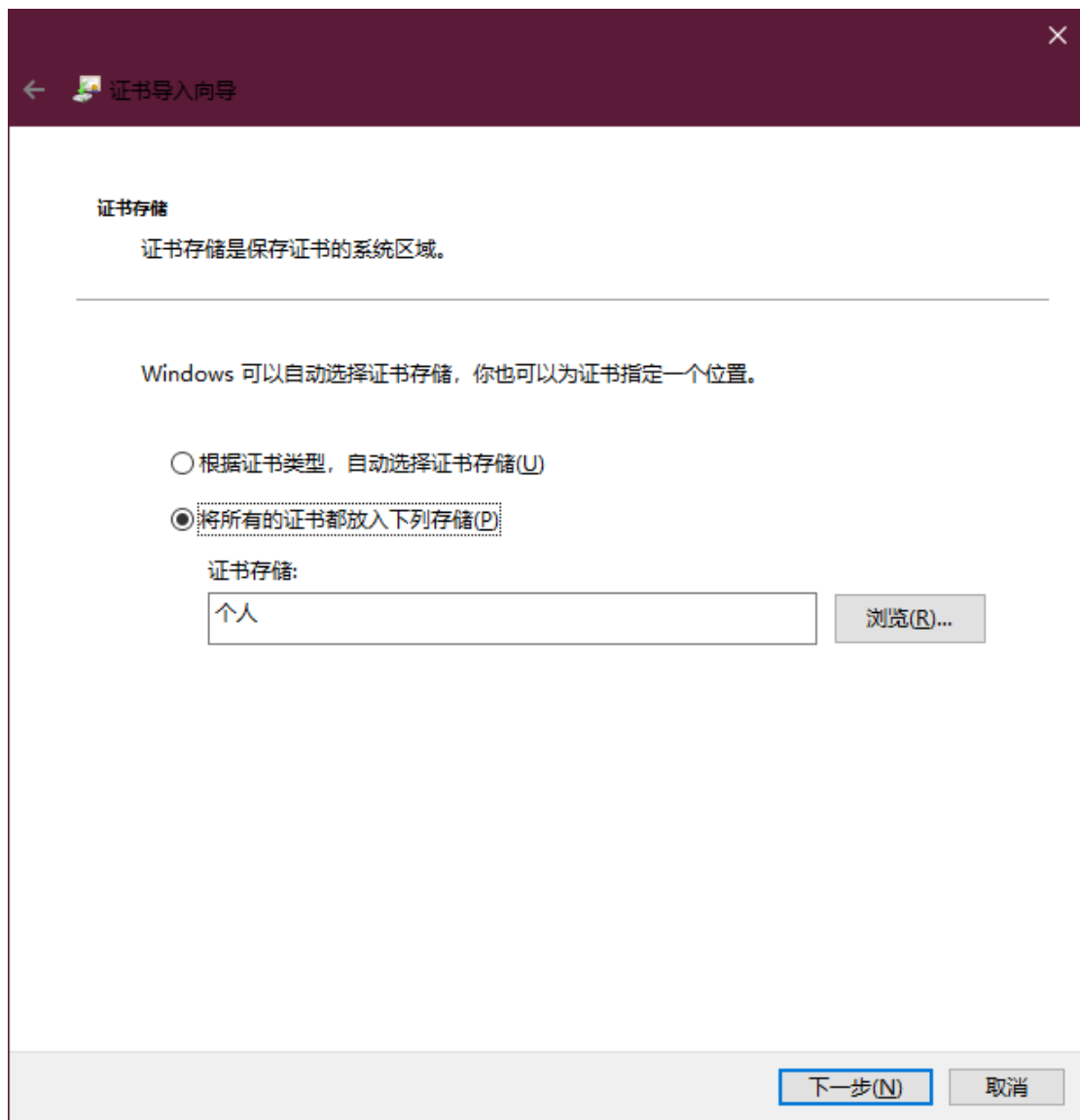
☐ 使用虚拟化安全保护私钥(不可导出)(P)

☒ 包括所有扩展属性(A)。

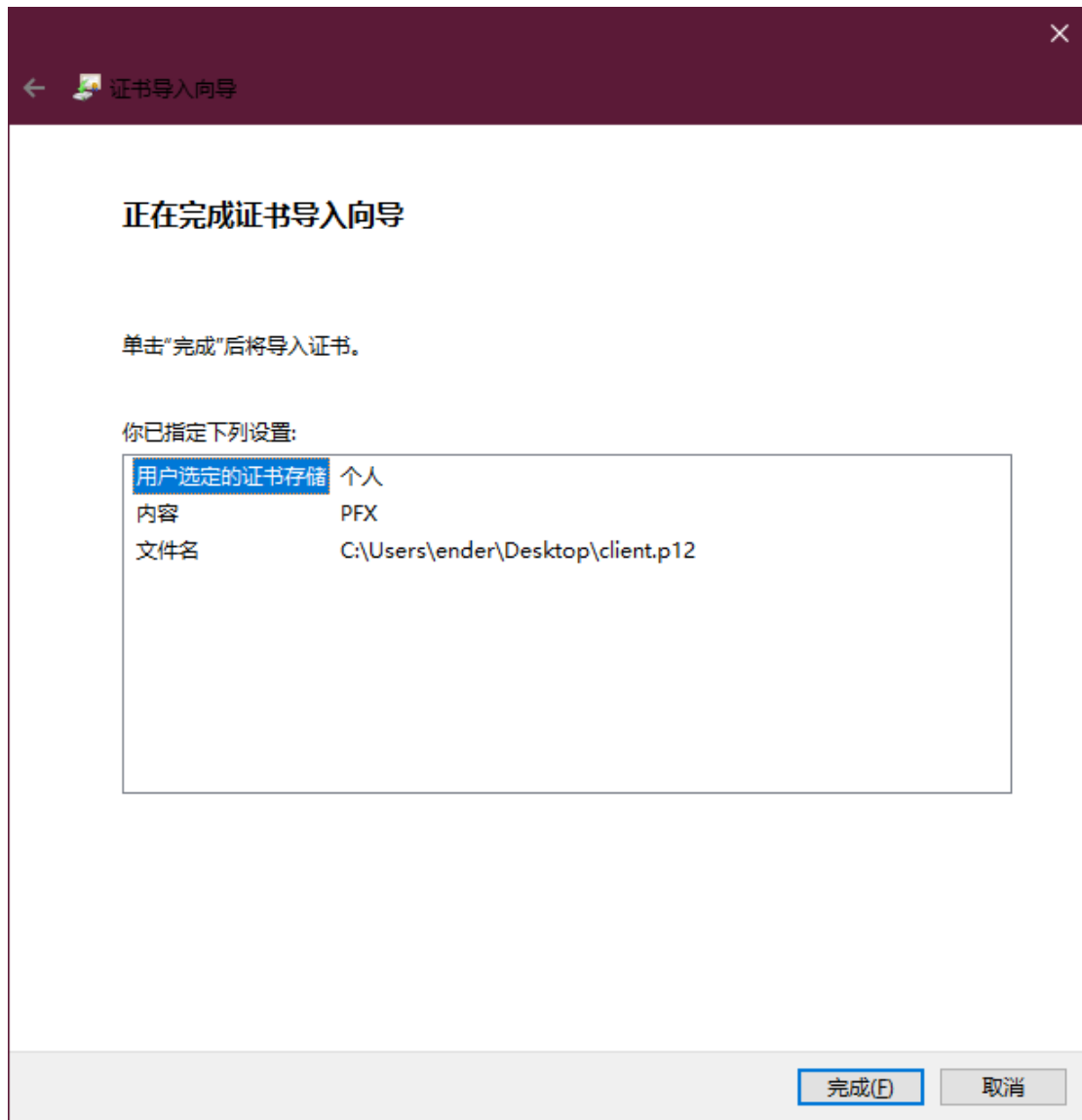
下一步(N)

取消

勾选 启用强私钥保护

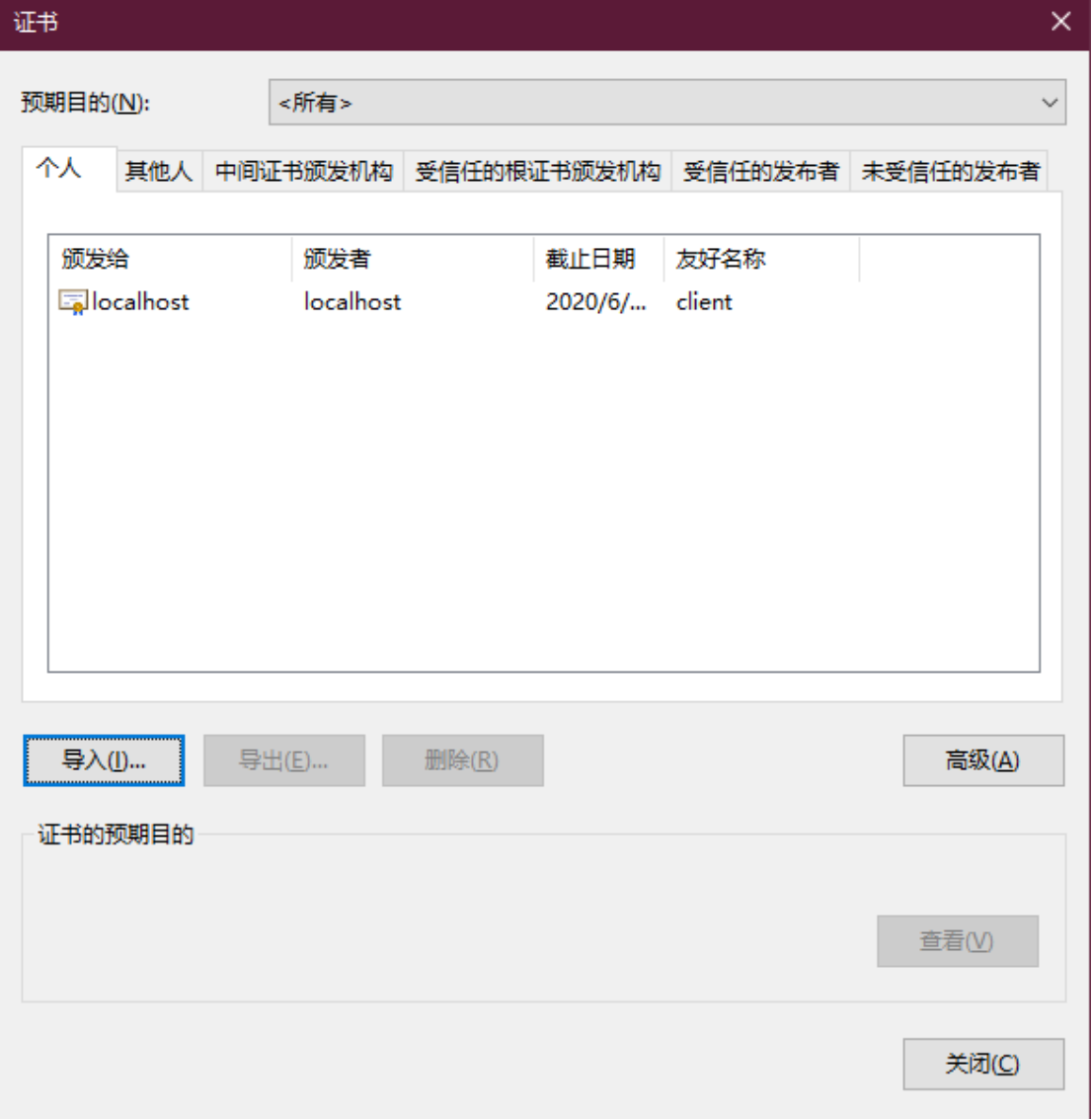


点击下一步直到导入成功,

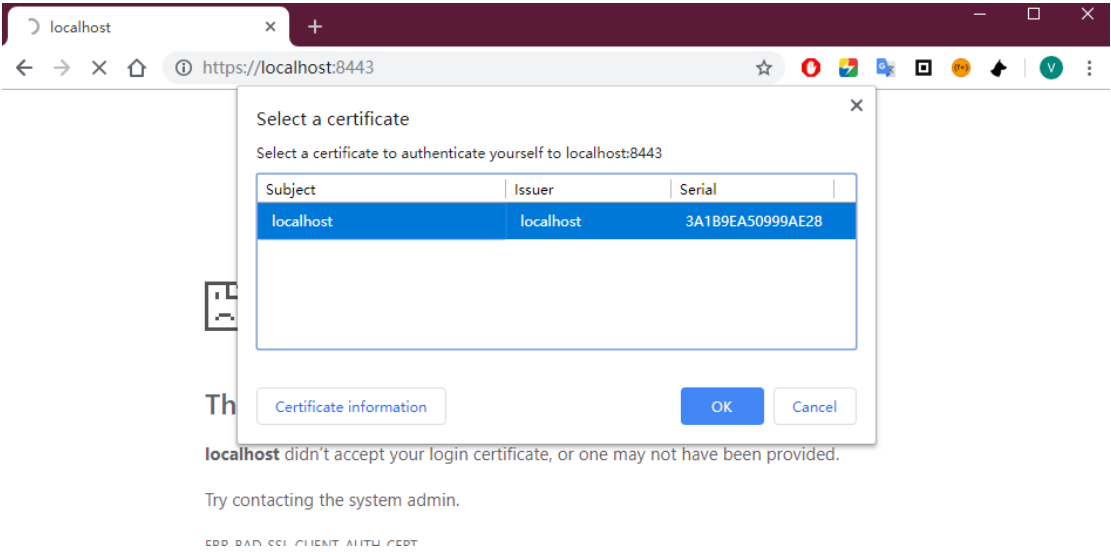


个人类的证书列表里面会多出刚导入的客户端证书

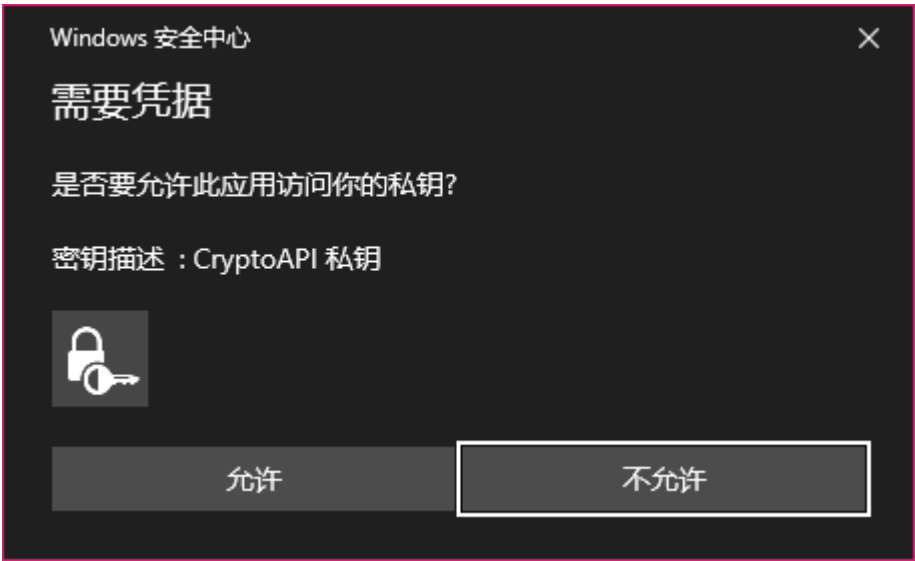




## 刷新浏览器



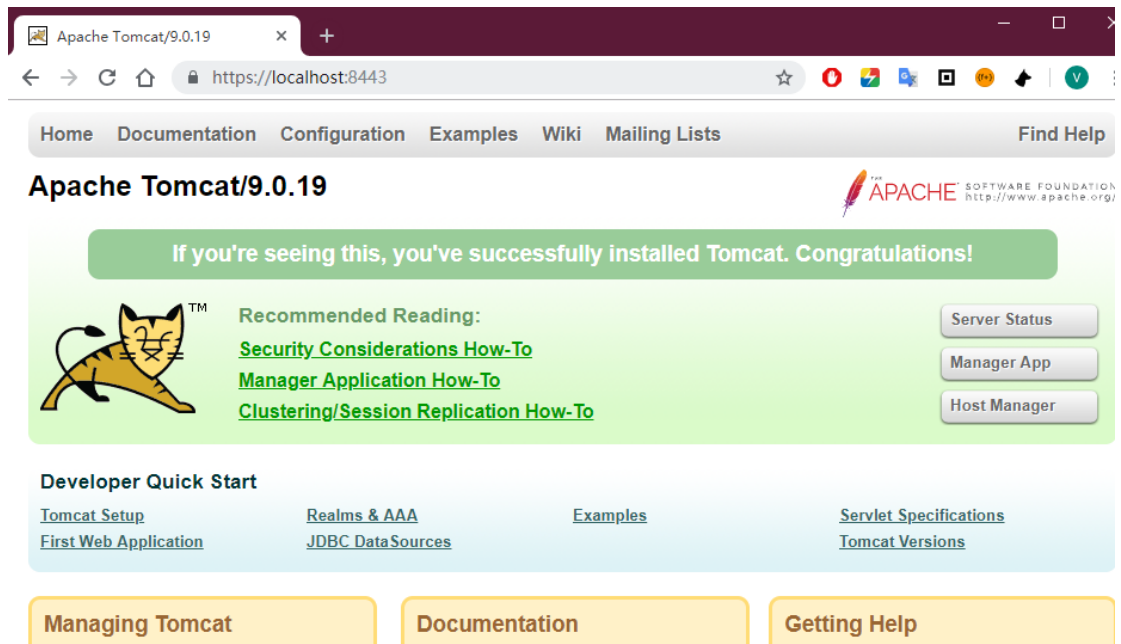
## 选择证书



选择后续所有的允许  
最好能成功打开页面

## 注意双向认证权限读取申请

浏览器针对同一地址的双向认证才会弹出证书选择框  
后续出现的安全要点允许  
有需要重复操作的请按照[个人客户端证书导入注意](#)



双向认证 Java 的 https 客户端请求

## Keystool 工具使用

请参考网络使用说明

依照 java 的 keytool 命令界面化工具，生成 java 专有的 jks 格式容器文件

## 依赖包

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-io</artifactId>
    <version>1.3.2</version>
</dependency>
<!--      https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient
-->

<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.9</version>
</dependency>
```

## Java client demo

```
package com.web.ssl.twoway.ssltwoway;

import com.web.ssl.twoway.ssltwoway.Constants;
import org.apache.commons.io.IOUtils;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.ssl.SSLContexts;
import org.apache.http.util.EntityUtils;
import com.web.ssl.twoway.ssltwoway.Constants;

import javax.net.ssl.SSLContext;
import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

public class SSLhttps {

    public static void main(String[] args) throws Exception {
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        keyStore.load(new FileInputStream(new File(Constants.clientPrivateFile)),
Constants.clientPrivatePassword.toCharArray());
        SSLContext sslcontext = SSLContexts.custom()
            //忽略掉对服务器端证书的校验
            .loadTrustMaterial(new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] chain, String authType)
throws CertificateException {
                    return true;
                }
            })
            .build();

        //加载服务端提供的 truststore(如果服务器提供 truststore 的话就不用忽略对服务器端证书的校验了)
```

```

        .loadTrustMaterial(new File(Constants.trustAcFile),
Constants.trustAcFilePassword.toCharArray(),
        new TrustSelfSignedStrategy())
        .loadKeyMaterial(keyStore,
Constants.clientPrivatePassword.toCharArray())
        .build();
        SSLConnectionSocketFactory sslConnectionSocketFactory = new
SSLConnectionSocketFactory(
        sslcontext,
        new String[]{"TLSv1"},
        null,
        SSLConnectionSocketFactory.getDefaultHostnameVerifier());
        CloseableHttpClient httpclient = HttpClients.custom()
        .setSSLSocketFactory(sslConnectionSocketFactory)//取消改行注释不导入
运行环境的客户端证书 AAA
        .build();

        try {

            HttpGet httpget = new HttpGet("https://localhost:8443/");
//            若注释 AAA 行的内容，请求需要双向认证的服务会出现以下异常
//unable to find valid certification path to requested target
            System.out.println("Executing request " + httpget.getRequestLine());

            CloseableHttpResponse response = httpclient.execute(httpget);
            try {
                HttpEntity entity = response.getEntity();
                System.out.println(response.getStatusLine());
                System.out.println(IOUtils.toString(entity.getContent()));
                EntityUtils.consume(entity);
            } finally {
                response.close();
            }
        } finally {
            httpclient.close();
        }
    }
}

```

## Java httpsServer demo

```

package com.web.ssl.twoway.ssltwoway;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.security.KeyStore;
import java.security.cert.Certificate;

import javax.net.ssl.KeyManager;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.security.cert.X509Certificate;

public class HTTPSServer {
    private boolean isServerDone = false;

    public static void main(String[] args) {
        HTTPSServer server = new HTTPSServer();
        server.run();
    }

    // Create the and initialize the SSLContext
    private SSLContext createSSLContext() {
        try {
            KeyStore keyStore = KeyStore.getInstance("PKCS12");
            keyStore.load(new FileInputStream(Constants.serverPrivateFile),
Constants.serverPrivatePassword.toCharArray());

            // Create key manager
            KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance("SunX509");
            keyManagerFactory.init(keyStore,
Constants.serverPrivatePassword.toCharArray());
            KeyManager[] km = keyManagerFactory.getKeyManagers();

```

```

        KeyStore keyStoreTrust = KeyStore.getInstance("JKS");
        keyStoreTrust.load(new FileInputStream(Constants.trustAcFile),
Constants.trustAcFilePassword.toCharArray());
        // Create trust manager
        TrustManagerFactory trustManagerFactory =
TrustManagerFactory.getInstance("SunX509");
        trustManagerFactory.init(keyStoreTrust);
        TrustManager[] tm = trustManagerFactory.getTrustManagers();

        // Initialize SSLContext
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(km, tm, null);

        return sslContext;
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return null;
}

// Start to run the server
public void run() {
    SSLContext sslContext = this.createSSLContext();

    try {
        // Create server socket factory
        SSLServerSocketFactory sslServerSocketFactory =
sslContext.getServerSocketFactory();

        // Create server socket
        SSLServerSocket sslServerSocket =
sslServerSocketFactory.createServerSocket(Constants.DEFAULT_PORT);

        System.out.println("SSL server started");
        while (!isServerDone) {
            SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();

            // Start the server thread
            new ServerThread(sslSocket).start();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

```

    }

    // Thread handling the socket from client
    static class ServerThread extends Thread {
        private SSLSocket sslSocket = null;

        ServerThread(SSLSocket sslSocketInit) {
            sslSocket = sslSocketInit;
            sslSocket.setEnabledCipherSuites(sslSocket.getSupportedCipherSuites());
            sslSocket.setEnabledProtocols(sslSocket.getSupportedProtocols());
            sslSocket.setNeedClientAuth(true);
            sslSocket.setWantClientAuth(true);
            sslSocket.setEnableSessionCreation(true);
        }

        public void run() {

            try {
                // Start handshake
                sslSocket.startHandshake();

                // Get session after the connection is established
                SSLSession sslSession = sslSocket.getSession();

                System.out.println("SSLSession :");
                System.out.println("\tProtocol : " + sslSession.getProtocol());
                System.out.println("\tCipher suite : " + sslSession.getCipherSuite());

                // Start handling application content
                InputStream inputStream = sslSocket.getInputStream();
                OutputStream outputStream = sslSocket.getOutputStream();

                BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
                PrintWriter printWriter = new PrintWriter(new
OutputStreamWriter(outputStream));

                String line = null;
                while ((line = bufferedReader.readLine()) != null) {
                    System.out.println("Inut : " + line);

                    if (line.trim().isEmpty()) {
                        break;
                    }
                }
            }
        }
    }

```



```

    }
    boolean clientCertValid = false;
    try {
        Certificate[] c1 = sslSession.getLocalCertificates();
        for (Certificate c1One : c1) {
            System.out.println("getLocalCertificates : " + c1One.toString());
        }
        X509Certificate[] c2 = sslSession.getPeerCertificateChain();
        for (X509Certificate c1One : c2) {
            System.out.println("getPeerCertificateChain      :      " +
c1One.toString());
        }
        Certificate[] c3 = sslSession.getPeerCertificates();
        for (Certificate c1One : c3) {
            System.out.println("getPeerCertificates : " + c1One.toString());
        }
        clientCertValid = true;
    } catch (Exception cex) {

    }
    if (clientCertValid) {
        // Write data
        printWriter.print("HTTP/1.1      200      OK\r\nServer:
johnserver/1.0.8.18\r\nContent-Length: 3\r\n\r\nok\n");

        printWriter.flush();
    } else {
        // Write data
        printWriter.print("HTTP/1.1      200      FAILED\r\nServer:
johnserver/1.0.8.18\r\nContent-Length: 6\r\n\r\nFAILED\n");

        printWriter.flush();
    }

    sslSocket.close();
} catch (Exception ex) {
    ex.printStackTrace();
}
}
}
}

```



