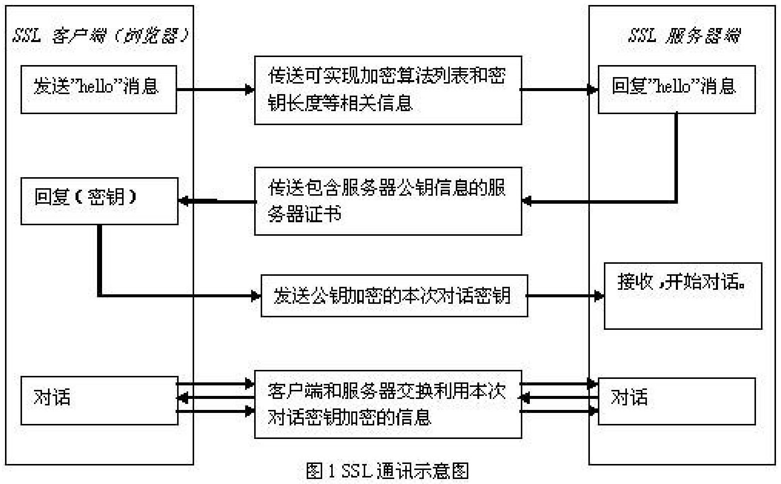
java tomcat 搭建SSL双向认证以及httpclient代码

转自 http://ian.wang/118.htm

技术文档

java tomcat 搭建SSL双向认证以及httpclient代码



SSL 客户端（浏览器）
SSL 服务器端
发送"hello"消息
传送可实现加密算法列表和密钥长度等相关信息
回复"hello"消息
回复（密钥）
传送包含服务器公钥信息的服务器证书
发送公钥加密的本次对话密钥
接收，开始对话。
对话
客户端和服务器交换利用本次对话密钥加密的信息
对话

图 1 SSL 通讯示意图

java tomcat 搭建SSL双向认证以及httpclient代码

一、生成密钥库和证书
可参考以下密钥生成脚本，根据实际情况做必要的修改，其中需要注意的是：服务端的密钥库参数“CN”必须与服务端的IP地址相同，否则会报错，客户端的任意。
key.script
1 、生成服务器证书库

```
keytool -validity 365 -genkey -v -alias server -keyalg RSA -keystore /opt/web/ssl/server.keystore -dname "CN=localhost,OU=sumscope,O=sumscope,L=Pudong,ST=Shanghai,c=com" -storepass 111111 -keypass 111111
```

2 、生成客户端证书库

```
keytool -validity 365 -genkeypair -v -alias client -keyalg RSA -storetype PKCS12 -keystore /opt/web/ssl/client.p12 -dname "CN=client,OU=sumscope,O=sumscope,L=Pudong,ST=Shanghai,c=com" -storepass 222222 -keypass 222222
```

3 、从客户端证书库中导出客户端证书

```
keytool -export -v -alias client -keystore /opt/web/ssl/client.p12 -storetype PKCS12 -storepass 222222 -rfc -file /opt/web/ssl/client.cer
```

4 、从服务器证书库中导出服务器证书

```
keytool -export -v -alias server -keystore /opt/web/ssl/server.keystore -storepass 111111 -rfc -file /opt/web/ssl/server.cer
```

5 、生成客户端信任证书库(由服务端证书生成的证书库)

```
keytool -import -v -alias server -file /opt/web/ssl/server.cer -keystore /opt/web/ssl/client.truststore -storepass 222222
```

6 、将客户端证书导入到服务器证书库(使得服务器信任客户端证书)

```
keytool -import -v -alias client -file /opt/web/ssl/client.cer -keystore /opt/web/ssl/server.keystore -storepass 111111
```

7 、查看证书库中的全部证书

```
keytool -list -keystore /opt/web/ssl/server.keystore -storepass 111111
```

二、Tomat配置
使用文本编辑器编辑${catalina.base}/conf/server.xml
找到Connector port="8443"的标签，取消注释，并修改成如下：

```
    <Connector SSLEnabled="true" clientAuth="true" keystoreFile="F:/ssl/kserver.keystore" keystorePass="111111" maxThreads="150" port="8443" protocol="org.apache.coyote.http11.Http11Protocol" scheme="https" secure="true" sslProtocol="SSL" truststoreFile="F:/ssl/tserver.keystore" truststorePass="111111"/>
```

备注：
keystoreFile：指定服务器密钥库，可以配置成绝对路径，如“/opt/web/ssl/server.keystore”。
keystorePass：密钥库生成时的密码
truststoreFile：受信任密钥库，和密钥库相同即可
truststorePass：受信任密钥库密码
clientAuth：是否验证客户端的证书，为FALSE时，可以使用浏览器访问，为"true"时浏览器必须添加自己的证书才可以被服务器接受。

三、建立演示项目
项目结构图:
项目名称：SSL（随意）

1. SSLServlet.java

```
package ian.wang.ssl.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.security.cert.X509Certificate;

public class SSLServlet extends HttpServlet {
    private static final String ATTR_CER = "javax.servlet.request.X509Certificate";
    private static final String CONTENT_TYPE = "text/plain;charset=UTF-8";
    private static final String DEFAULT_ENCODING = "UTF-8";
    private static final String SCHEME_HTTPS = "https";

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        response.setCharacterEncoding(DEFAULT_ENCODING);
        PrintWriter out = response.getWriter();
        out.println("cmd=["+request.getParameter("cmd")+"], data=["+request.getParameter("data")+"]");
        X509Certificate[] certs = (X509Certificate[]) request.getAttribute(ATTR_CER);
        if (certs != null) {
            int count = certs.length;
            out.println("共检测到[" + count + "]个客户端证书");
            for (int i = 0; i < count; i++) {
                out.println("客户端证书 [" + (++i) + "]：  ");
                out.println("校验结果：" + verifyCertificate(certs[--i]));
                out.println("证书详细：\r" + certs[i].toString());
            }
        } else {
            if (SCHEME_HTTPS.equalsIgnoreCase(request.getScheme())) {
                out.println("这是一个HTTPS请求，但是没有可用的客户端证书");
            } else {
                out.println("这不是一个HTTPS请求，因此无法获得客户端证书列表 ");
            }
        }
        out.close();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }


    private boolean verifyCertificate(X509Certificate certificate) {
        boolean valid = false;
        try {
            certificate.checkValidity();
            valid=true;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return valid;
    }
}
```

2. web.xml

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。

```
<?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>
```

3. index.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<!doctype html>
<html lang="zh-cn">
<head>
  <title>客户端证书上传</title>
  <meta http-equiv="pragma" content="no-cache">
  <meta http-equiv="cache-control" content="no-cache">
  <meta http-equiv="expires" content="0">
</head>
<body>
<form action="sslServlet" method="post">
  <input type="submit" value="提交证书"/>
</form>
</body>
</html>
```

四、演示及配置
发布演示项目，通过浏览器访问： http://127.0.0.1:8080/SSL 或 https://127.0.0.1:8443/SSL ，提示无法访问，需要导入客户端SSL证书
:
双击"client.p12"或在浏览器的工具，输入生成密钥时的客户端密码"222222",刷新浏览器即可正常访问了。

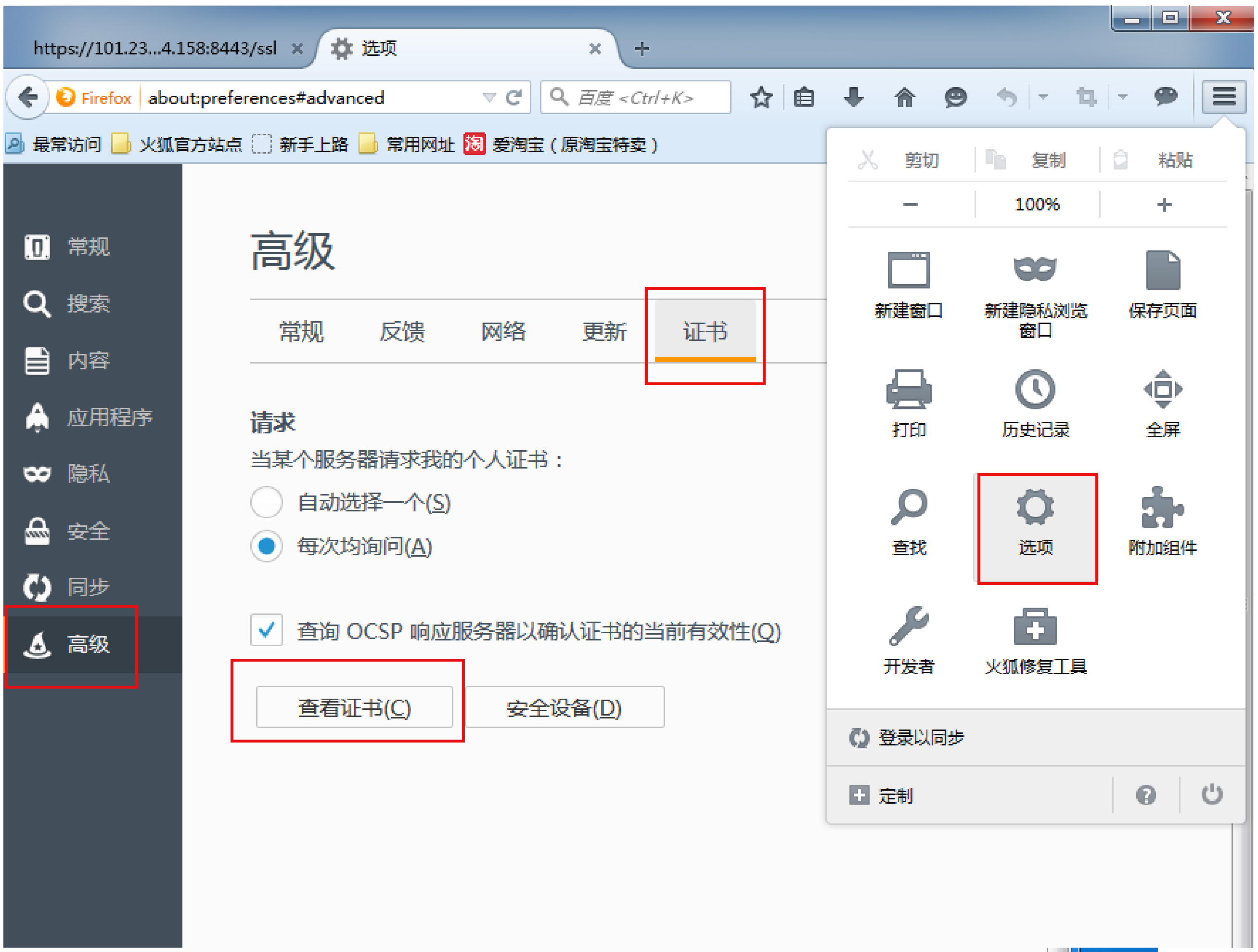
五、HttpClient模拟SSL Post请求

```
package test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;

import org.apache.commons.io.IOUtils;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
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;

public class SSLHttpRequest {


    private static String CLIENT_KEY_STORE = "keys/kclient2.keystore";
    private static String CLIENT_TRUST_KEY_STORE = "keys/tclient.keystore";
    private static String CLIENT_KEY_STORE_PASSWORD = "111111";
    private static String CLIENT_TRUST_KEY_STORE_PASSWORD = "111111";
    private static String CLIENT_KEY_PASS = "111111";

    public final static void main(String[] args) throws Exception {

        //服务端信任的客户端的证书库（可以包含很多客户端的证书）
        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
        FileInputStream instream = new FileInputStream(new File(CLIENT_TRUST_KEY_STORE));
        try {
            trustStore.load(instream, CLIENT_TRUST_KEY_STORE_PASSWORD.toCharArray());
        } finally {
            instream.close();
        }
        //服务端的证书
        KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
        FileInputStream keyStoreInput = new FileInputStream(new File(CLIENT_KEY_STORE));
        try {
            keyStore.load(keyStoreInput, CLIENT_KEY_STORE_PASSWORD.toCharArray());
        } finally {
            keyStoreInput.close();
        }

        // Trust own CA and all self-signed certs
        SSLContext sslcontext = SSLContexts.custom()
                .loadTrustMaterial(trustStore, new TrustSelfSignedStrategy())
                .loadKeyMaterial(keyStore, CLIENT_KEY_PASS.toCharArray())
                .build();
        // Allow TLSv1 protocol only 必须加上TLSv1 , 不然不行
        SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(
                sslcontext,
                new String[]{"SSLv3","TLSv1"},
                null,
                new HostnameVerifier() {

                    @Override
                    public boolean verify(String hostname, SSLSession session) {
                        hostname = "fhl";//证书生成时组织的名称  , 必须
                        return SSLConnectionSocketFactory.getDefaultHostnameVerifier().verify(hostname, session);
                    }
                });
        CloseableHttpClient httpclient = HttpClients.custom()
                .setSSLSocketFactory(sslsf)
                .build();
        try {
            URIBuilder builder = new URIBuilder("https://127.0.0.1:8443/SSL-Service/SSLServlet");
            builder.setParameter("cmd", "value1")
            .setParameter("data", "value2");
            HttpPost httpPost = new HttpPost(builder.build());
            System.out.println("executing request" + httpPost.getRequestLine());

            CloseableHttpResponse response = httpclient.execute(httpPost);
            try {
                HttpEntity entity = response.getEntity();

                System.out.println("----------------------------------------");
                System.out.println(response.getStatusLine());
//                if (entity != null) {
//                    System.out.println("Response content length: " + entity.getContentLength());
//                }
                InputStream iStream=entity.getContent();
                byte buffer[]=new byte[1024];
                int len=0;
                ByteArrayOutputStream os=new ByteArrayOutputStream();
                while ((len=IOUtils.read(iStream, buffer,0,1024))!=0) {
                    os.write(buffer, 0, len);
                }
               System.out.println( new String(os.toByteArray()));
                EntityUtils.consume(entity);
            } finally {
                response.close();
            }
        } finally {
            httpclient.close();
        }
    }
}
```

注意：火狐和chrome、IE目前不支持导入个人签名的证书，所以在浏览器中没法使用。

六、使用浏览器访问 https应用截图

1. 打开浏览器，访问测试网址：https://101.231.124.155:8443/ssl ，由于该应用配置了 Tomcat SSL双向认证，需要客户端提供证书文件导入成功了，才能正常访问。在Firefox 浏览器中，导入客户端证书，在 Firefox 选项 - 高级 - 证书 中， 点击 查看证书。

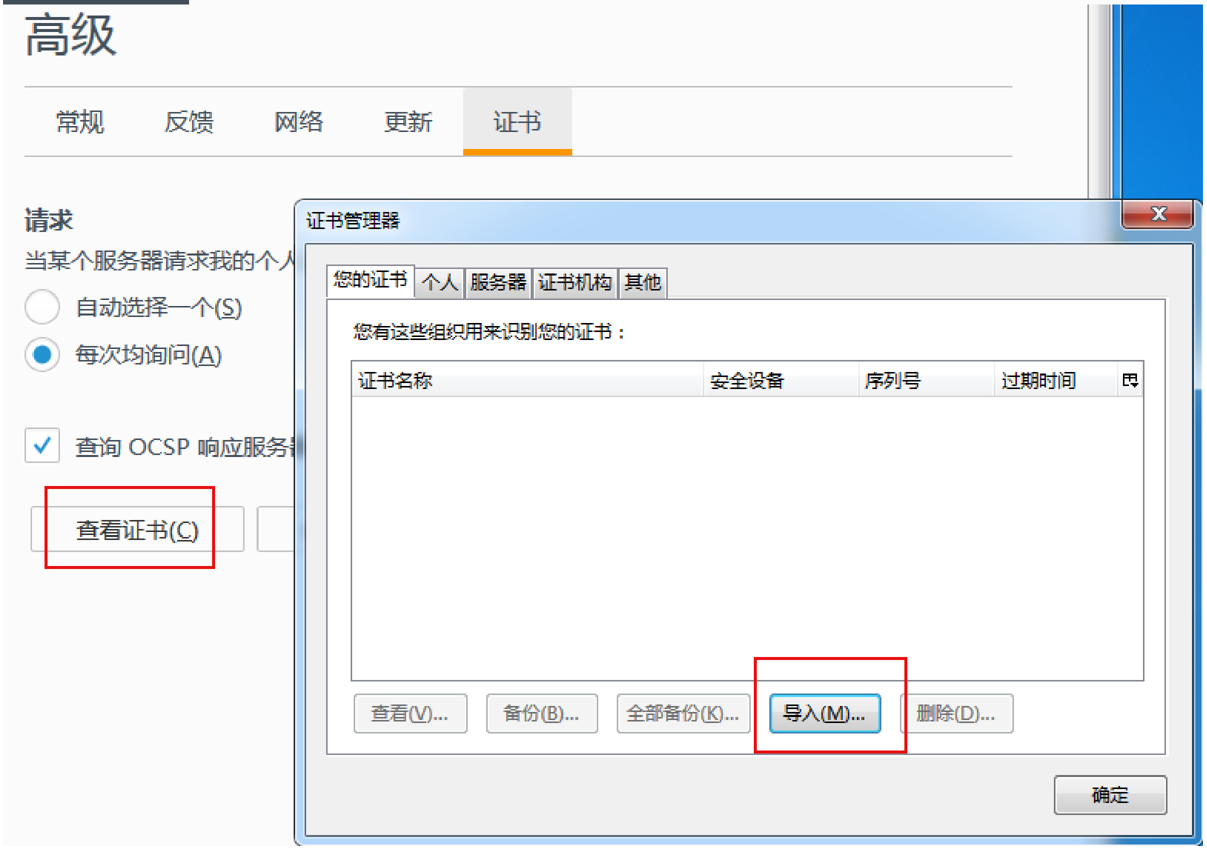


请求

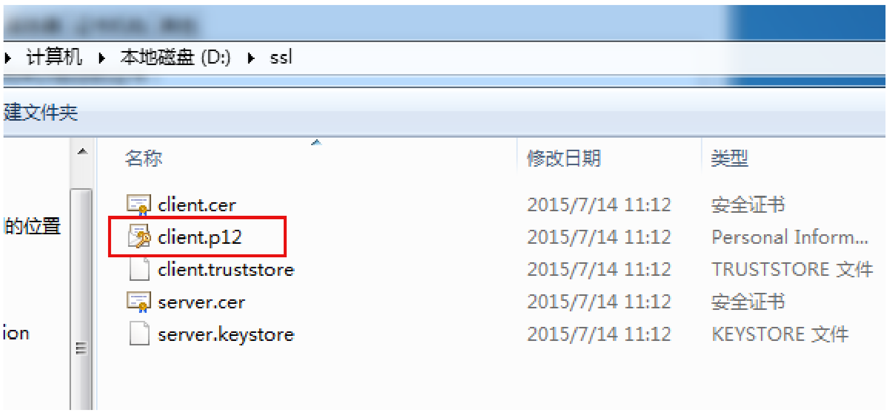
当某个服务器请求我的个人证书：

自动选择一个(S) [✓ 每次均询问(A)]

✓ 查询 OCSP 响应服务器以确认证书的当前有效性(Q)

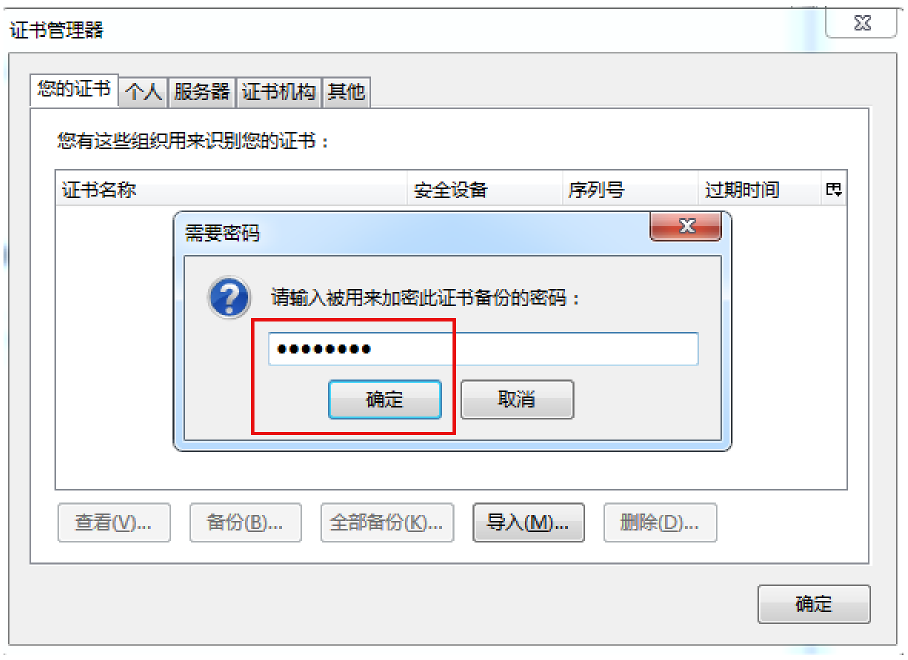

高级
常规 反馈 网络 更新 证书
请求
当某个服务器请求我的个人
自动选择一个(S)
每次均询问(A)
查询 OCSP 响应服务
查看证书(C)
证书管理器
您的证书 个人 服务器 证书机构 其他
您有这些组织用来识别您的证书：
证书名称 安全设备 序列号 过期时间
查看(V)... 备份(B)... 全部备份(K)... 导入(M)... 删除(D)...
确定

2. 点击“导入”， 选择客户端证书文件 client.p12

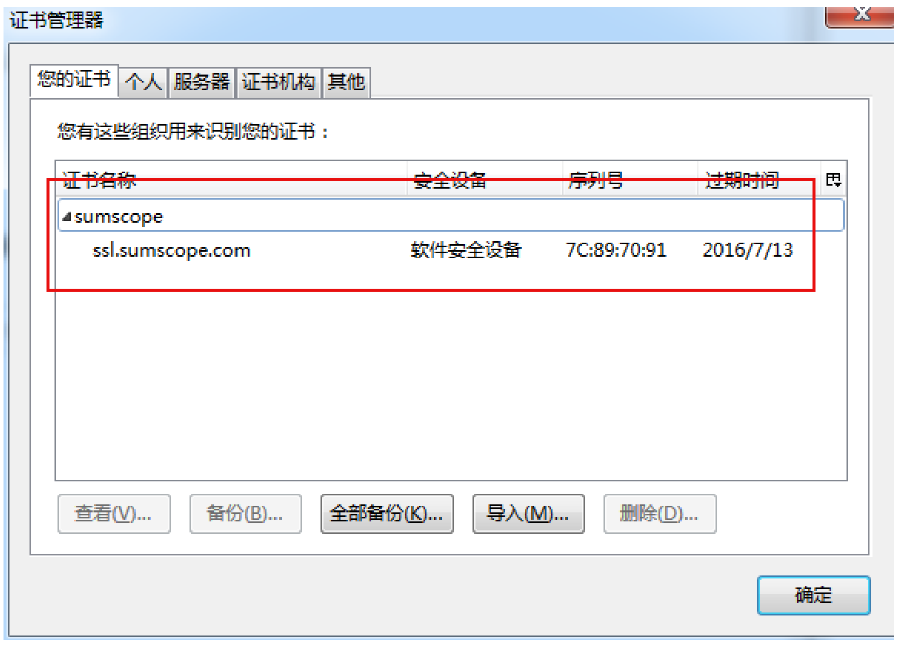


名称	修改日期	类型
client.cer	2015/7/14 11:12	安全证书
client.p12	2015/7/14 11:12	Personal Inform...
client.truststore	2015/7/14 11:12	TRUSTSTORE 文件
server.cer	2015/7/14 11:12	安全证书
server.keystore	2015/7/14 11:12	KEYSTORE 文件

3．输入客户端证书密码

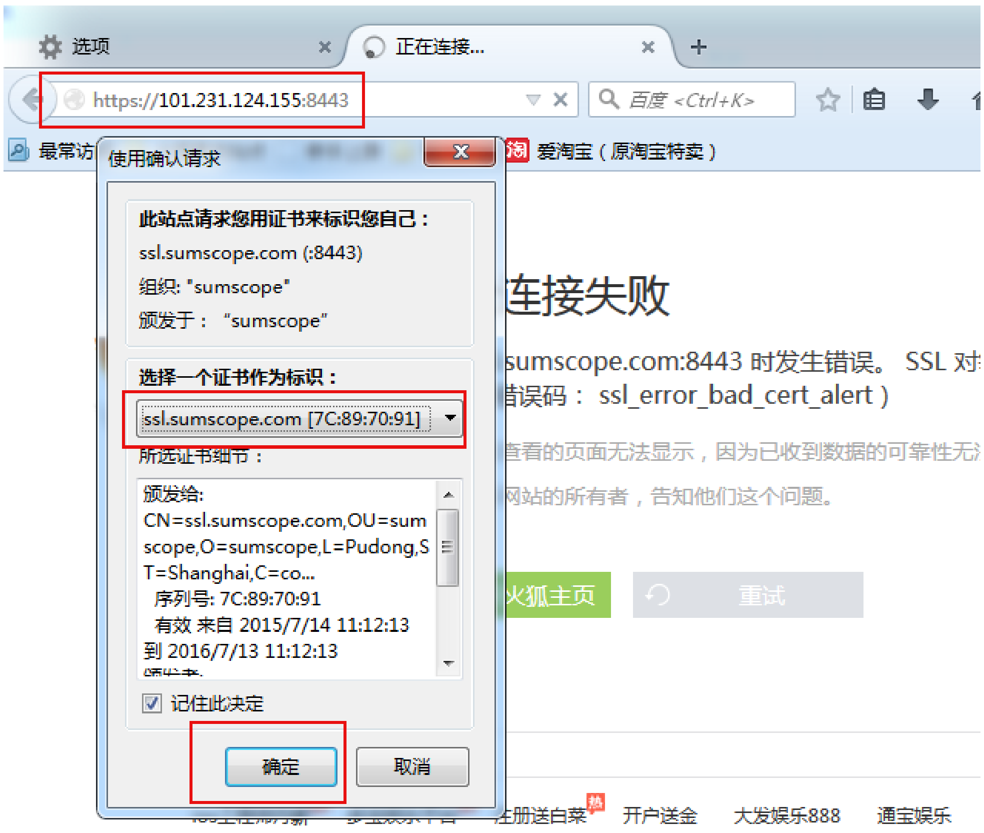


4．客户端证书验证成功后，显示证书如下



证书名称	安全设备	序列号	过期时间
◢ sumscope			
ssl.sumscope.com	软件安全设备	7C:89:70:91	2016/7/13

5．打开网址，输入： `https://101.231.124.155:8443` ，浏览器会提示 选择已安装的证书，点击“确认”。



sumscope.com:8443 时发生错误。 SSL 对
错误码： ssl_error_bad_cert_alert)

查看的页面无法显示，因为已收到数据的可靠性无法
网站的所有者，告知他们这个问题。

使用确认请求

此站点请求您用证书来标识您自己：

ssl.sumscope.com (:8443)

组织: "sumscope"

颁发于： "sumscope"

选择一个证书作为标识：

ssl.sumscope.com [7C:89:70:91]

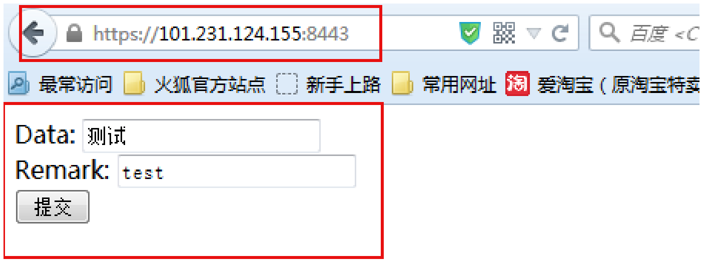
所选证书细节：

颁发给:
CN=ssl.sumscope.com,OU=sumscope,O=sumscope,L=Pudong,ST=Shanghai,C=co...
序列号: 7C:89:70:91
有效 来自 2015/7/14 11:12:13 到 2016/7/13 11:12:13

☑ 记住此决定

确定　取消

6．显示如下，则表示SSL证书导入成功， 测试页面正常显示了。



Data: 测试

Remark: test

提交