

通向架構師的道路（第二天）之apache tomcat https應用

分類：

一、總結前一天的學習

在前一天的學習中我們知道、瞭解並掌握了Web Server結合App Server是怎麼樣的一種架構，並且親手通過Apache的Http Server與Tomcat6進行了整合的實驗。

這樣的架構的好處在於：

ü 減輕App Server端的壓力，用Web Server來分壓，即Web Server只負責處理靜態HTML內容，而App Server專職負責處理Java請求，這對系統的performance是一個極大的提升。

ü 安全，Web Server端沒有任何Java源代碼包括編譯後的東西，對Internet開放的只有Web Server，因此黑客就算通過80端口攻入了我們的Web Server，他能得到什麼？除了靜態HTML內容，任何邏輯，口令他都得不到，為什麼？喏。。。因為我們的App Server「躲」在Web Server的屁股後面呢。

需要注意的地方：

ü 如果以這樣的架構出現，你的J2EE 工程，必須在web.xml裡把那些個<servlet-mapping>劃分清楚，比如說：

我們可以知道*.do, *.action, *.jsp是屬於JAVA需要解析的東西對吧！

但是，如果你的servlet寫成這樣

/abc

/123

/def

那麼當我們在作映射時，需要把/abc, /123, /def分別寫成一行行的JKMount語句，是不是。。。OK，假設我們這個工程有100個servlet（這個算少的哦），你該不會在httpd.conf文件中給我寫這樣的無聊的東西100行吧？

所以，我們在規劃我們的servlet時需要有矩可循，即pattern，因此我才一直強調，大家在servlet命名時必須統一成：

/servlet/myServletabc

這樣，我在做這個Web Server到App Server的Mapping時，是不是只要一句：JkMount /servlet/* ajp13就可以搞定啦？

ü 同樣的架構有不同的變種：

² IIS+Tomcat

因為微軟的IIS本身就是一個Web Server，因此通過IIS和Tomcat的一個插件叫「isapi」的也可以作到這樣的架構，但是我強烈不推薦，因為JAVA源於Unix系統，歸於Unix系統，Unix可是不認什麼IIS的，一定請一定用Apache，你是JAVA不是多奶（dot net）。

² Apache+Weblogic

² IBM HttpServer(Apache的一個變種)+IBM WAS6.x/WAS7.X

² Tomcat集群

Apache掛N多個tomcat，由tomcat1...tomcat2...tomcat3...等組成

² Weblogic集群

Apache掛N多個weblogic，由weblogic1...weblogic2...weblogic3...等組成

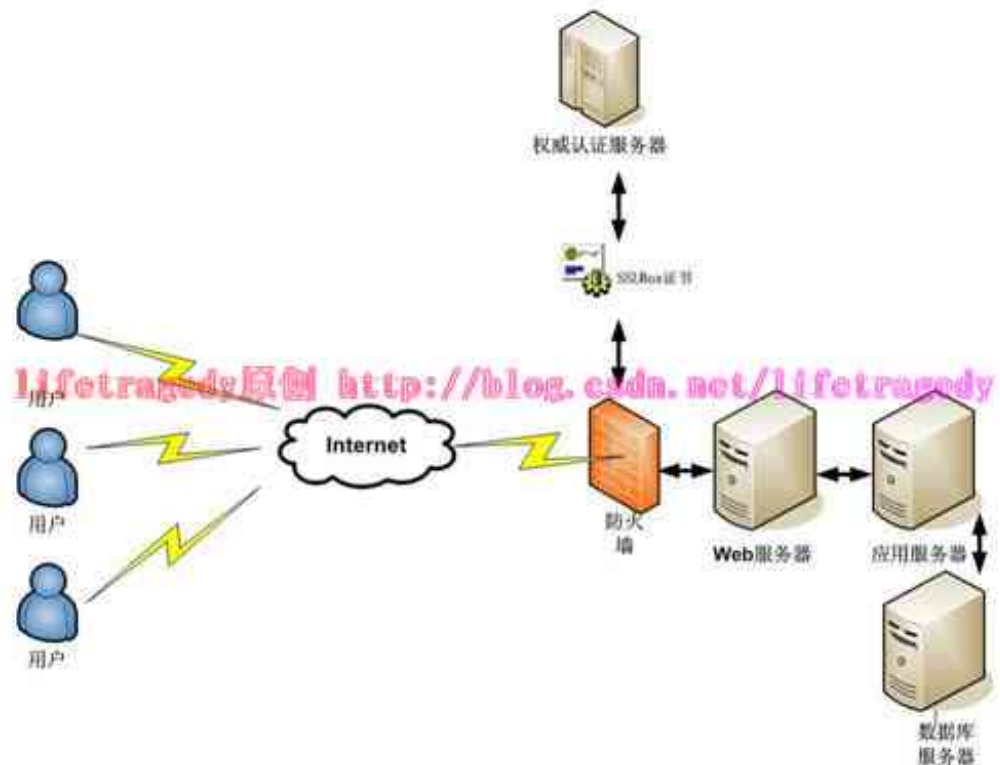
² WASND (IBMWebsphere App Server Network Deployment)

IBM HttpServer掛N多個WAS，由WAS1...WAS2...WAS3...等組成

二、HTTPS

2.1 HTTPS介紹

先來看HTTPS的概念



我們一般的http走的是80端口，而https走的是443端口，有什麼不一樣的地方嗎？

很簡單，我們拿個telnet命令來作個實驗：

telnet127.0.0.1 80，直接就登進了80端口（如果你機器上的Apache開放的話），這樣好極了，所有的http中的get, put, post全部可以被我們截獲，你的上網帳號，你提交的表單信息全部被別人攔截，就算你對一些信息加了密，對於

黑客來說，這些加密被解密只是時間問題，而且一般黑客可以利用雲計算，群集計算對你的加密可以進行「硬殺傷」，即窮舉算法，利用超大規模集群解密的你的算法會很快，電影裡的幾十秒解開一個128位的加密不是神話，是真的！！

因此，我們要讓黑客一開始就攻不進來，連門都進不來，何談拿到我裡面的東西，對不對？

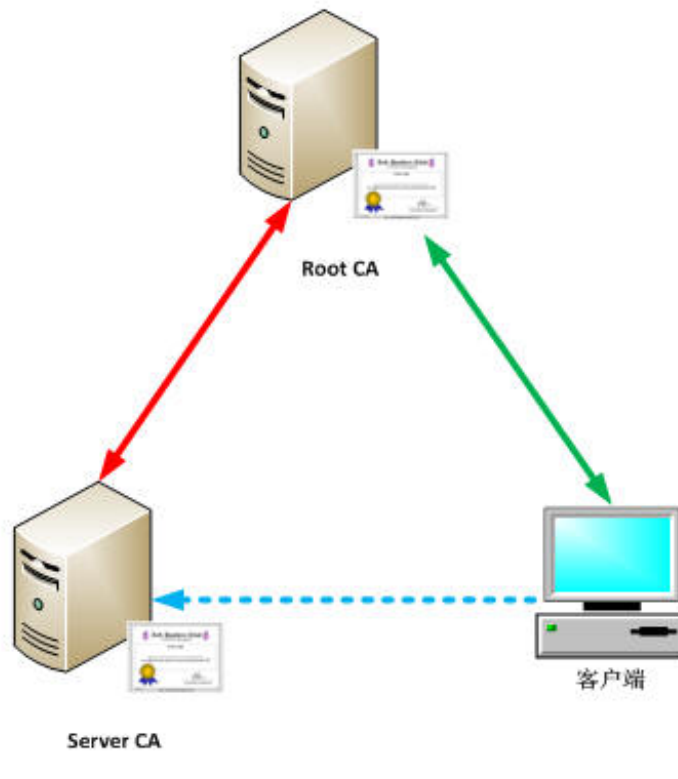
現在我們把我的http通道，變成了https，同時關閉80端口，因此用戶要訪問必須經過443端口。好了，我們再來用：

```
telnet localhost 443
```

你連telnet都進不進去，因此，你就無法再獲取http通道內傳輸的東西了。因此黑客要進入你的網站先要突破這個https，而https使用的是RSA非對稱128位加密，如果是安全交易類甚至會使用RSA 1024位加密算法，除非是世界上最高明的黑客才能突破我們的防線。

2.2 HTTPS的構成

要構成HTTPS，我們需要有一張「根證書」，一張「服務器認證」才能做到一般的https，HTTPS還分成「雙向認證」，在雙向認證的結構中我們需要3張證書即「根證書」，「服務器認證」，「客戶端認證」。為什麼需要這麼多證書？嘿嘿，下面我們來看HTTPS是怎麼構成這個「信任關係鏈」的。



ü 證書我們稱為CA；

ü 根證書叫Root CA；

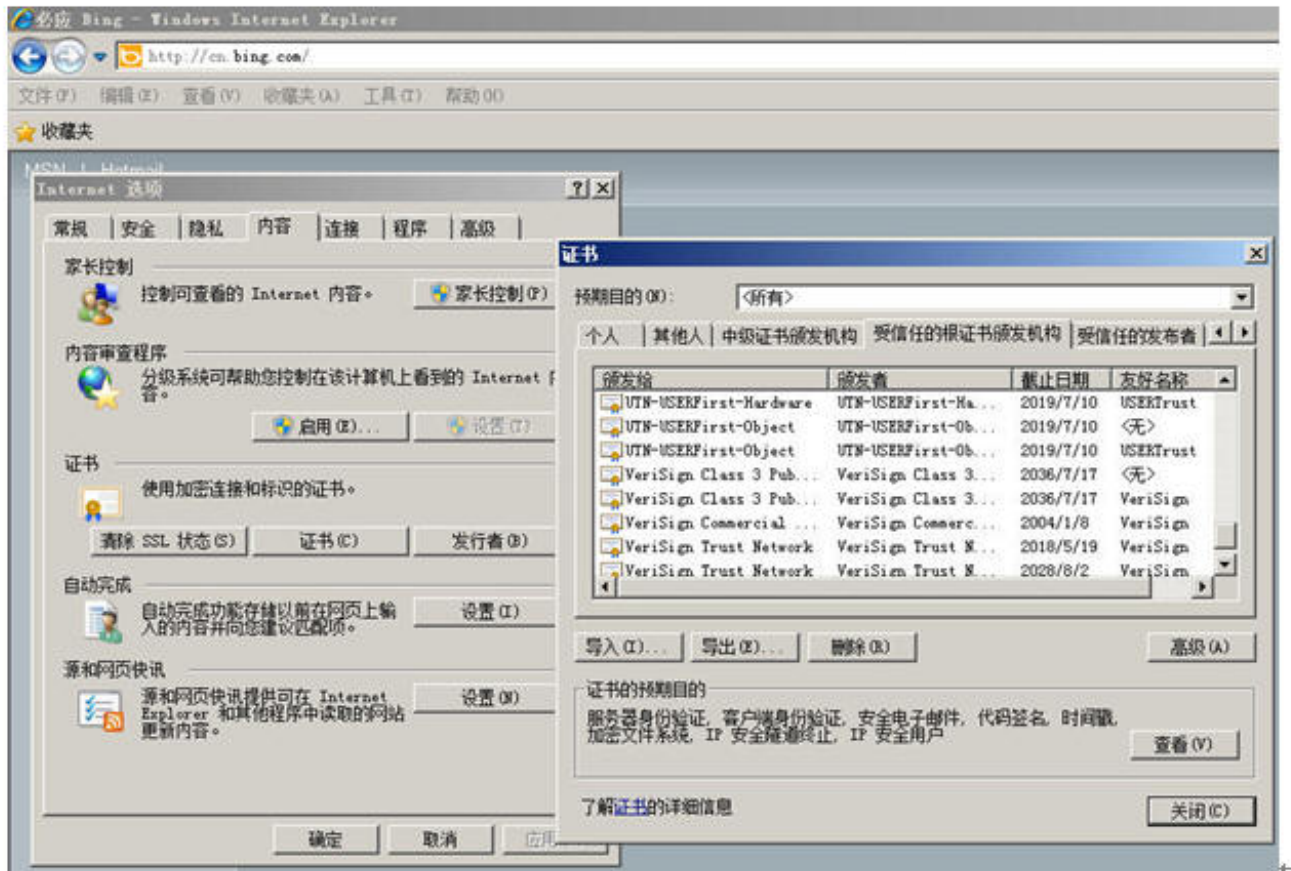
上述這個圖什麼意思？

首先，RootCA是全球的根，這個「樹」的根是全球任何IE、FireFox、Safari裡的證書庫裡都有這個RootCA的，因為它們是權威，所以全球的電子證書拿它們做「根」，這些證書比較具有代表性的是：

ü Verisign

ü RSA

這兩家公司是世界上所有加密算法的「鼻祖」，因此被拜為全球所信任，我們可以在我們的IE中看到這些「根」。



其此，全球的計算器客戶默認在裝完系統後，都會帶有這些ROOT CA，因此「由ROOT CA簽出來的服務器證書將自動被客戶端所信任」。

所以，這個信任關係，就此建立。

在HTTPS是SSL的一種，它們間是如何進行加密傳輸的呢，就是這個「信任關係」，先建立起信任關係，然後再開始數據傳輸，在加密的世界中「建立信任」就需要用到至少2張證書，即ROOT CA， SERVER CA，我們把這個信任建立的過程稱為「Hands Shake」，握手協議。

前面說到了，這個握手分單向和雙向，包括上述這個圖就是一個單向握手，什麼叫單向，什麼叫雙向呢？我們下面來講解：

ü 單向握手信任

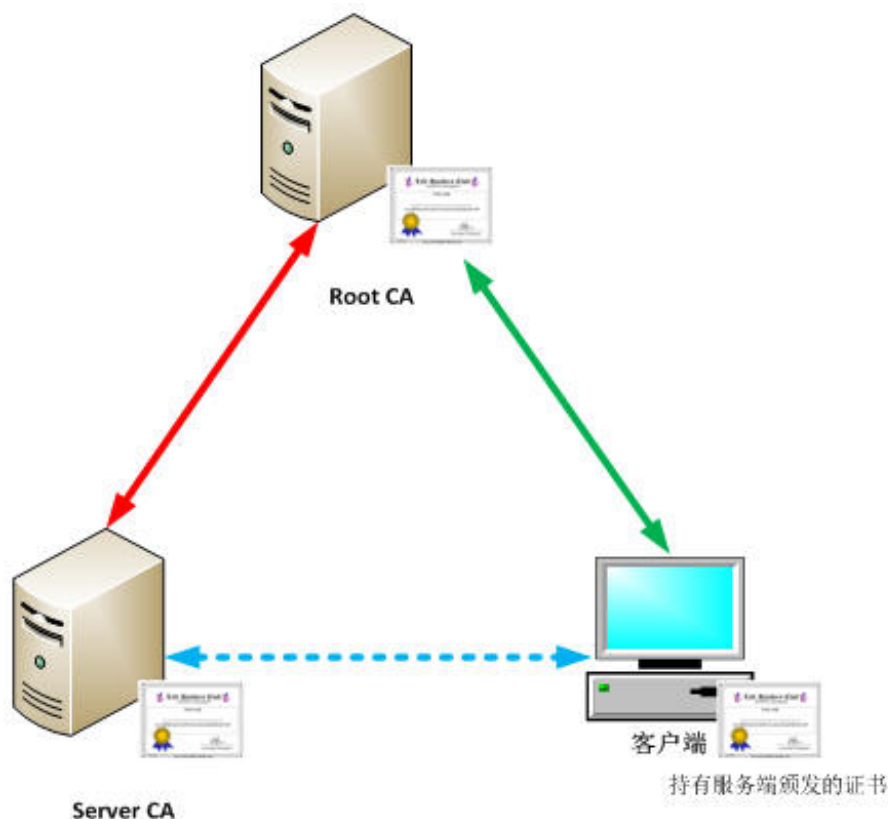
我們又稱它為「包二奶協議」，大家想一下，貪官包二奶和二奶說「你跟著我，我每月給你1萬塊」，他說的這個話，能不能寫下來？能嗎？當然不能，寫下來還得了，將來二奶一不爽把這份白紙黑字的東西交到中紀委還不把這爛貨給雙規了哈？

所以，二奶單向裡信任貪官，這就是二奶協議，即客戶端認為我訪問的這台服務器「是安全的」，因此客戶端可以向服務器發送和提交任何東西。

ü 雙向握手信任

我們又稱它為「君子協定」，呵呵，從這個詞表面上來看就知道這個協議有多牢靠了，首先，它是寫在字面上的，其次，雙方都簽署協議這個信任關係怎麼樣啊？非常牢靠！

即客戶端信任服務器，因此客戶端可以向服務器發送和提交任何東西。同時，服務器也信任客戶端，允許該客戶端向我發送和提交東西。



客戶端單向信任服務器很簡單，只要這個服務器是我客戶端信任的頂級根簽發出來的證書就行，而服務器如何信任客戶端呢？記住下面三句話：

首先，你這個客戶端到我這邊來登記一下；

其次，我給你簽一張證書，你帶回家裝在你的IE裡；

最後，每次訪問時因為你的IE裡裝著我服務器簽出的證書，所以你是我的會員，所以我信任你；

2.3 證書與如何生成證書的基本概念

通過上面的概念，我們知道了，如果建立起這個HTTPS的環境我們需要至少一張服務器證書，對不對？而且這張服務器證書是需要由客戶端信任的「ROOT」級機構所簽發出來的。

所以一般生成證書由以下幾個步驟構成：

1. 生成一對不對稱密鑰，即公鑰public key和私鑰 private key
2. 用密鑰產生請求，同時把我的請求交給ROOT機構

3. ROOT機構對我提交的請求進行「簽名」Sign

這個被簽完名後的「請求」就稱為證書。

上面多出來了密鑰，公鑰，私鑰三個名詞，下面來做解釋。

先來看一個真理：

1) 密鑰，密鑰為一對，即一把公鑰，一把私鑰

2) 一把私鑰可以對應多把公鑰，而這些公鑰只可能來源於一把私鑰

3) 公鑰加密，私鑰解密

大家想一下，公鑰加密，這是「公」就是人人有這把KEY，都可以用來加密，但是能打開我這扇門的因該只有一個人是吧？這就是為什麼說「私鑰」解密。

倒過來

私鑰「加密」（被稱為簽名），公鑰「解密」（被稱為認證）

把上面這個「真理」倒過來，呵呵，好玩了，因為public key只能用來加密而解密必須是private key，因此公鑰是不能加密的，公鑰也不能用來解密，那麼它們該怎麼做？

HP公司是賣打印機的，它有100個代理，都為它銷售打印機。

客戶來到了某個代理公司，問：你憑什麼說你是HP的代理

代理公司說：請看，這是HP公司給我證書

客戶問：你的證書不能偽造嗎？

代理公司答：請看，下面有一個防偽條形碼

於是，客戶把這個防偽條形碼用手機拍下來，來到HP公司說：這是不是你們的代理？

HP公司說：你等一下！隨後HP公司拿出以前給這家代理公司的公鑰，對這個簽名做一個「解密」（被稱為雜湊算法），也算出來一個防偽條形碼，然後拿這個防偽條形碼和客戶帶來的防偽條形碼一比較，完全一致，所以HP和客戶說：您可以完全相信這家公司，這家公司是我代理的。

這邊這個防偽條形碼就是代理公司用HP在頒發證書時給它時用HP的私鑰的「簽名」；

HP公司用為當初給代理商發放證書時同時生成的密鑰對裡的公鑰對這個簽名做一個雜湊算法，然後拿算出來的防偽條形碼再和客戶帶來的這個防偽條形碼比對的這個過程被稱為「認證」；

一起來看看證書裡的防偽條形碼吧



我們把它稱為電子「指紋」，一般用的是SHA或者是MD5算法，因為MD5和SHA是不可逆唯一性算法，所以把它比喻成「指紋」再恰當不過了。

2.4 實際開發實驗中如何產生證書

實際產生證書時，我們需要生成請求，但不是說我們把請求交給Verisign或者一些信息機構，它們就幫我們簽的，這是要收費的，一般簽個名：50-500美金不等，有時還分為每年必須去簽一次，要不然就會失效。

所以我們在實際開發環境中，為了做實驗或者是搭建模擬環境，不可能會去花錢買個證書的，有時我們環境要搭建幾套，怎麼辦？這錢花的沒有明堂的。

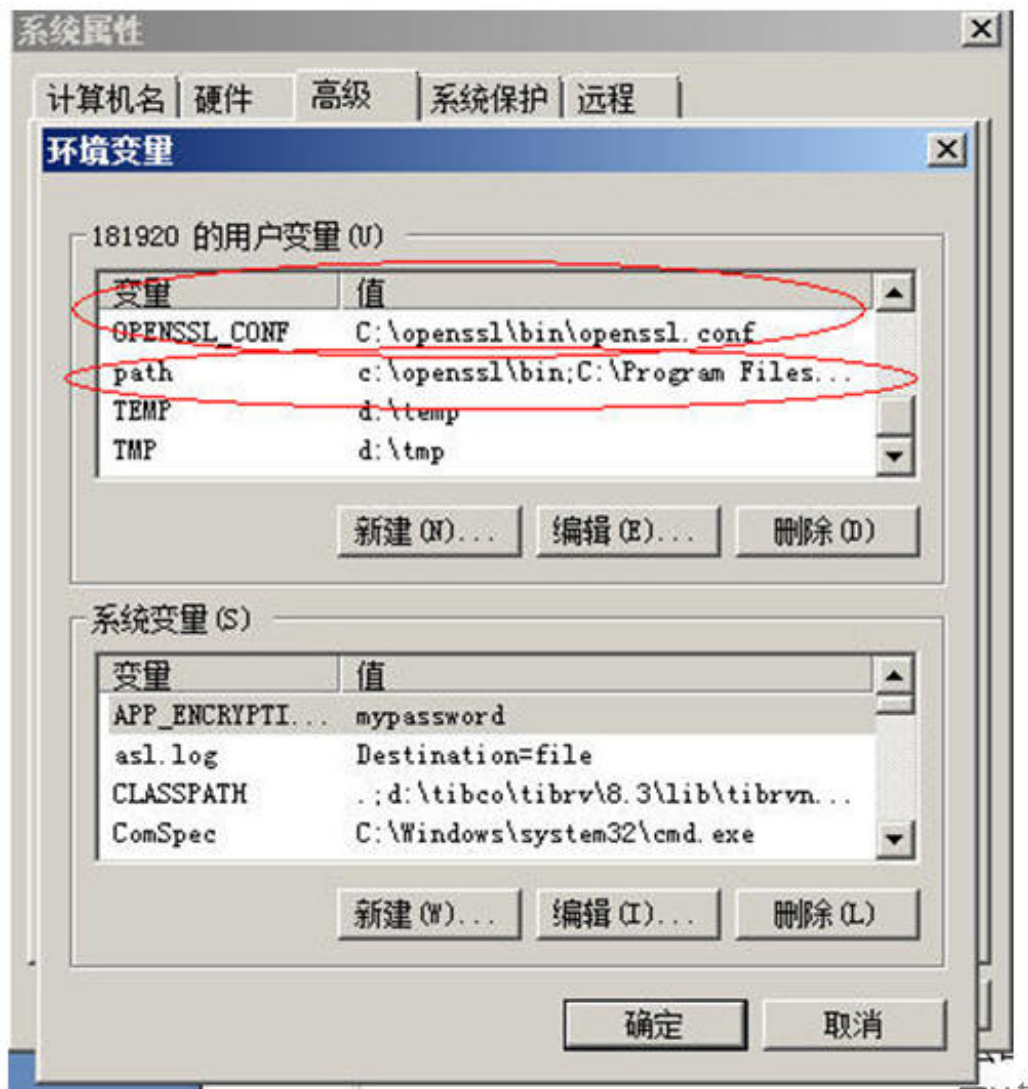
所以，在實際開發環境中，我們自己來模擬這個ROOTCA，然後用自己模擬出來的ROOTCA去簽我們服務器的證書，這個過程就被稱為「自簽」。

2.5 使用OpenSSL來簽證書

OpenSSL就是這麼一個自簽，加密的命令行工具，它是從UNIX下分離出來的一個項目，但也有FOR WINDOWS平台的，比如說我給你們用的這個OPENSSL，就是For WIN的，但是由於它是從UNIX/LINUX下產生的，因此它內部的配置還是用的是LINUX/UNIX的盤符與路徑，需要手動去校正，當然我已經做好了校正，因此直接打了個壓縮包放在了FTP上，大家拿下來後解壓後就可以直接用了。

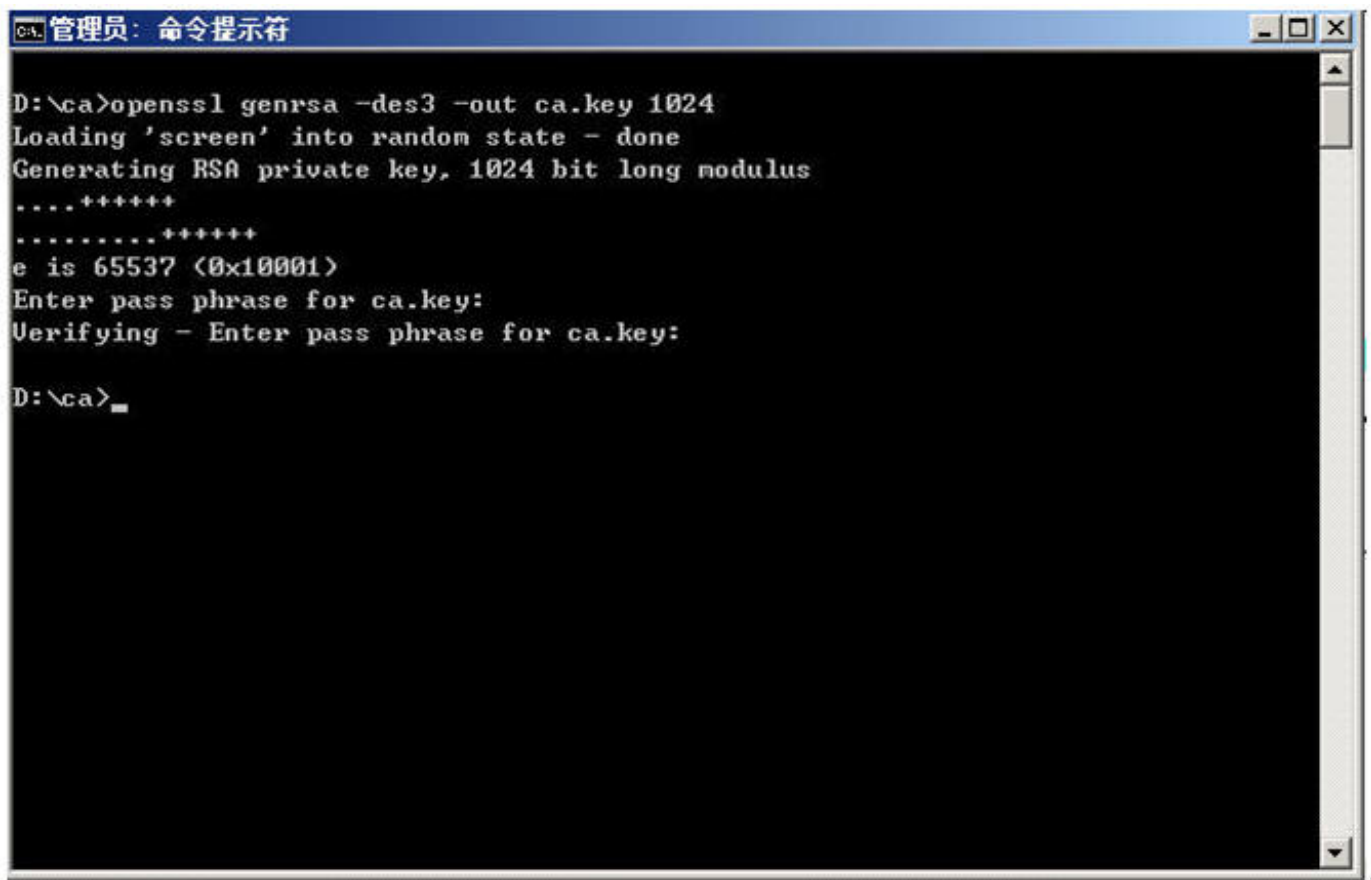
如果你們是自己從網上官方網站下載的OPENSSL需要手動去改它的盤符和路徑，要不然是用不起來的。

ü 設置環境變量



把c:\openssl\bin\openssl.cnf設成OPENSSL_CONF這樣的一個變量，同時把c:\openssl\bin目錄加到你的path裡去（根據你們自己的解壓後的openssl的實際路徑）。

ü 生成根證書所用的密鑰



```
管理员: 命令提示符

D:\ca>openssl genrsa -des3 -out ca.key 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:

D:\ca>
```

提示輸入密碼我們使用：aaaaaa

再次輸入確認密碼

(密鑰，由其是private key是由口令保護的)

去除CA密鑰的口令

為什麼我們要把好好的口令保護給去除呢？這邊不是去除而是代表這個證書在被應用程序啟動時不需要顯示的提示用戶輸入口令，要不然我們會出現下面這種情況：

在啟動HTTPS協議的服務器時，一般我們點一下service-

>apache2.x啟動，就啟動了，但如果這個https所帶的證書是沒有經過上述這道手續後處理的話，這個服務在啟動時會失敗，而需要切換成手動命令行啟動，就是黑屏！在黑屏狀態下，apache2.x服

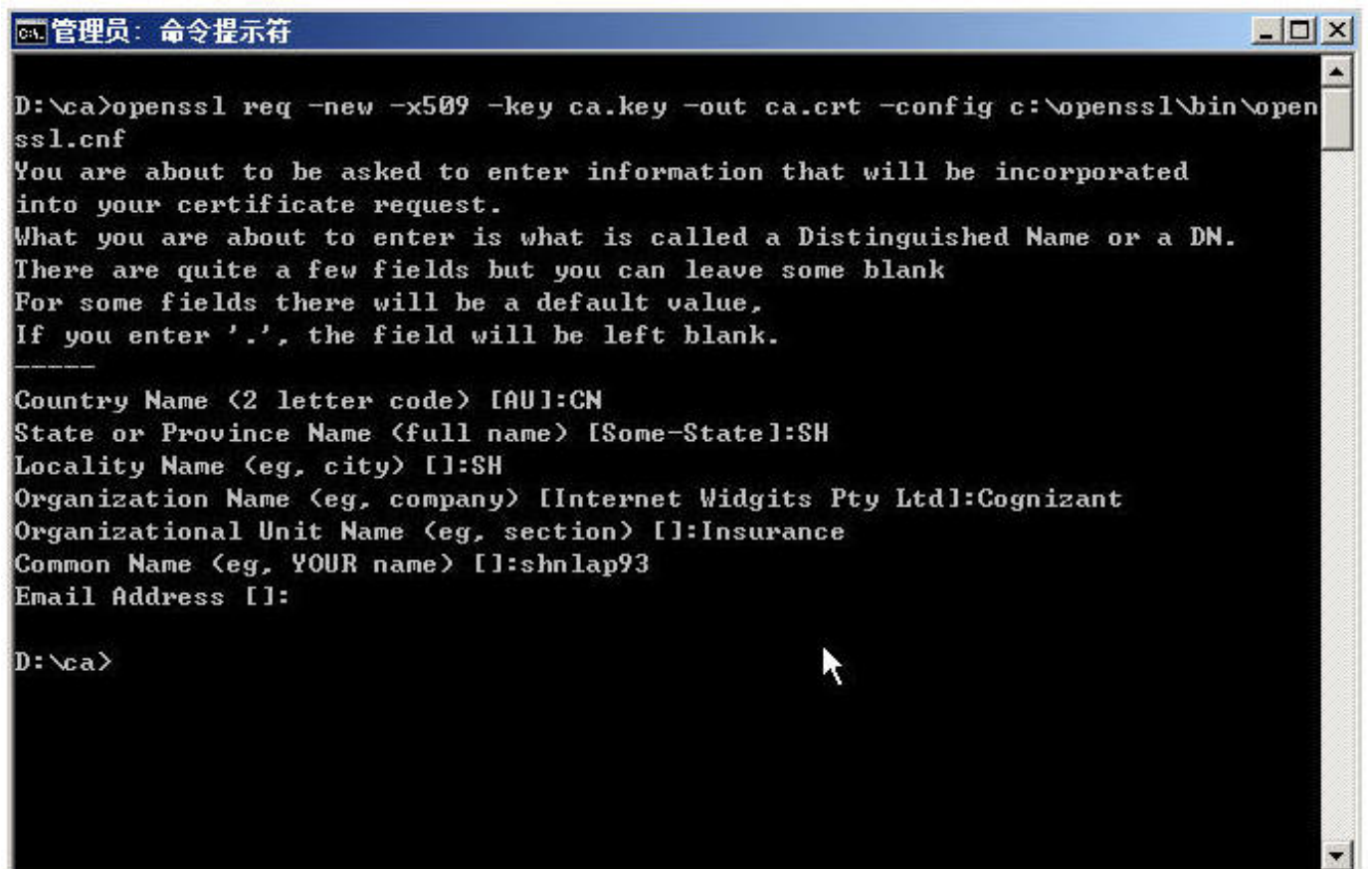
務器啟動時會提示你要求：輸入口令，這個太麻煩了，一般啟動服務器服務的一定是超級管理員，因此一般情況下沒必要在啟動相關服務時再輸入一遍口令了。

ü 生成CA即ROOT CA證書並自簽

網
上
有
很
多
說



```
管理员: 命令提示符
D:\ca>openssl rsa -in ca.key -out ca.key
Enter pass phrase for ca.key:
writing RSA key
D:\ca>
```



```
管理员: 命令提示符
D:\ca>openssl req -new -x509 -key ca.key -out ca.crt -config c:\openssl\bin\open
ssl.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:SH
Locality Name (eg, city) []:SH
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Cognizant
Organizational Unit Name (eg, section) []:Insurance
Common Name (eg, YOUR name) []:shnlap93
Email Address []:
D:\ca>
```

法，說是先產生CA的Request請求，再用ca.key去自簽，我給大家介紹一條一步到位的產生ca ROOT證書的命令，為

了安全，我們在最後加上「-configC:\openssl\bin\openssl.cnf」，以使openssl工具可以找到相應的config文件（有些系統在指定了OPENSSL_CONF環境變量後一般就不需要在命令行裡去手工指定這個-config變量了）。

由於我們產生的證書為:X509格式，因此需要按照X509格式填入相關的值。

² AU-國家家的縮寫，如：CHINA=CN，美國=USA，英國=UK，日本=JP

² State or Province Name-省/洲的縮寫或者是全稱，如：上海=SH

² Locality Name-城市的全稱或者是縮寫，如：上海=SH

² Organization Name-公司名，如：Cognizant

² Common Name-要安裝這台證書的主機名，證書是和主機名綁定的，如果證書裡的主機名和你實際的主機名不符，這張證書就是非法的證書。

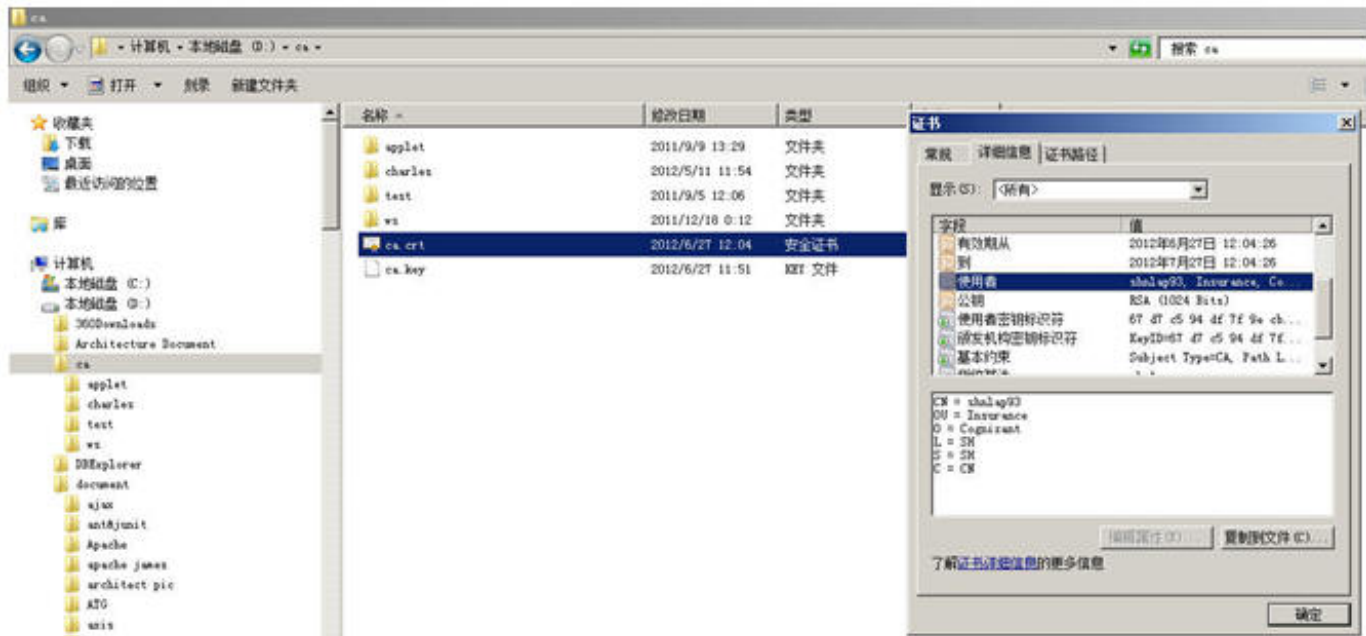
我們不能夠填IP，一定一定要填主機名即域名www.xxx.com這樣的東西，比如說我填的是shnlap93，但我的主機怎麼知道shnlap93是指:10.225.106.35或者說是指localhost這台機器呢？

打開C:\Windows\System32\drivers\etc\hosts這個文件，如下：

```
localhost shnlap93
10.225.106.35 shnlap93
```

看到了吧？所以當我們使用pint shnlap93時，它是不是就可以知道shnlap93=10.225.106.35啦？

EmailAddress-郵件地址，愛填不填，可以跳過，反正我們是「自簽」。



Look，我們的CA證書生成了，可以雙擊這張證書，查看信息後關閉它。

目前這張ROOT 證書，只是個自簽的產品，因為是自簽，一般其它客戶端的IE裡因此是不會帶有這張根證書的。

要其實客戶端也能信任這張根證書，我們必須怎麼辦？

將它安裝到我們的IE的信任域裡。

ü 將ROOT CA導入客戶端的根級信任域，有多少台客戶端，每個客戶端都要導一邊這個證書！

所以說如果我們擁有世界級的根證書該多好啊，電腦上默認就帶有我們的證書，因此知道這幫世界級的根證書機構為什麼能掙錢了吧？50-500美金簽張證書，幾秒鐘的事，CALL！！！

點
[導
入]
按鈕

下一
步，
下一
步，
此時
會有一
個彈出
框，選



「yes（是）」完成導入。

再來打開我們的ca.crt文件

發現了沒有，這張證書是有效的證書了，所以在「證書信息」前原有的一個紅叉叉，消失了。

ü 生成Web服務器端證書密鑰

我們的root證書有了，現在可以生成Web服務器端的證書了，並且用root ca去簽名

先生成密鑰，密碼6個a

去除密碼（提示：enter pass phrase for server.key時輸入剛才生成密鑰時的密碼即6個a。

ü 生成Web服務器端證書的簽名請求

生成服務器端證書請求時需要輸入server端key的口令，我們為了方便，也用6個a。

ü 用Root CA去對Web服務器的證書請求即csr(certificat request)進行簽名認證

輸入y並回車

此時它會提示：

1 out of
1 certificate
requests
certified,
commit?
[y/n], 再輸入y
並回車

Web服務器的
server.crt證書
生成完畢。

註：

如果在操作時
有任何錯，必
須連同生成
的.key,.csr,
.crt文件全部刪
除重頭來一遍

我們來看看，
這個server.crt
文件，雙擊
它。

首先，我們看
到該證書的
「證書信息」
前沒有紅色的

大叉，然後是證書信息正是我們剛才輸入的內容，為什麼沒有大叉？

因為我們的RootCA根證書裝在我們IE的根級信任域裡，又因為我們的客戶端信任我們的RootCA，因此當我們的客戶端打開由RootCA簽出來的server.crt時，這根「信任鏈」被建立了起來，所以客戶端自動單向信任我們的server.crt，對不對？

下面我們來做一個實驗，把我們的Root CA從我們的根級信任域中刪除。

選中這個shnlap93的根級證書，點[刪除]，會彈出兩次確認框，選「yes」確認刪除掉它。

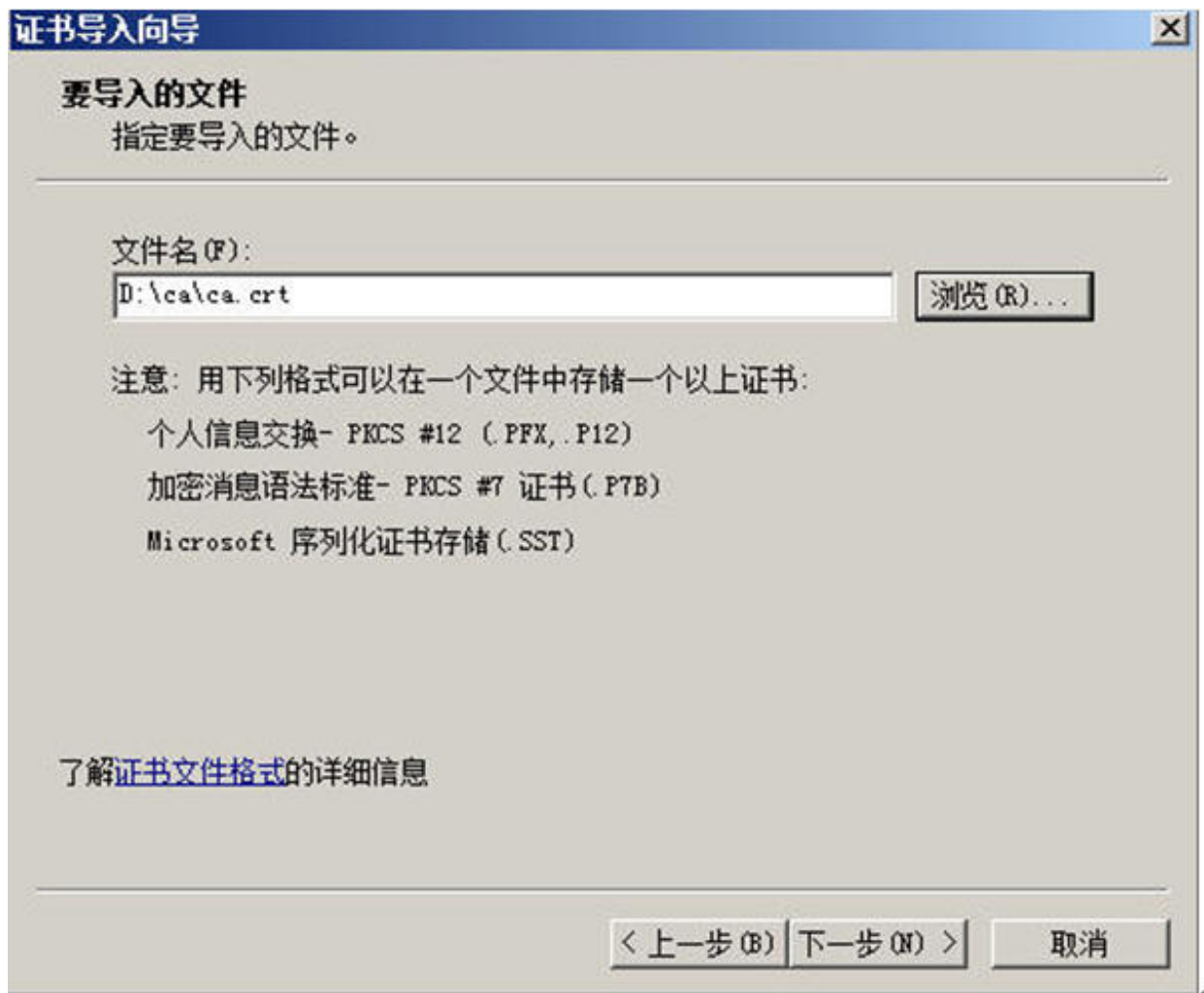
關閉IE，然後我們再次雙擊我們的server.crt文件，來查看證書內容。

我們看到了什麼？「不能驗證該證書。」

重新導入我們的Root CA至IE的根級信任域（見將ROOT CA導入客戶端的根級信任域）。

再次打開server.crt查看證書內容。

一切回覆正常了。



2.6 為Apache HttpServer佈署https協議

ü 用文本編輯器打開httpd.conf文件，找到如下這一行


```
#Include conf/extra/httpd-ssl.conf
```

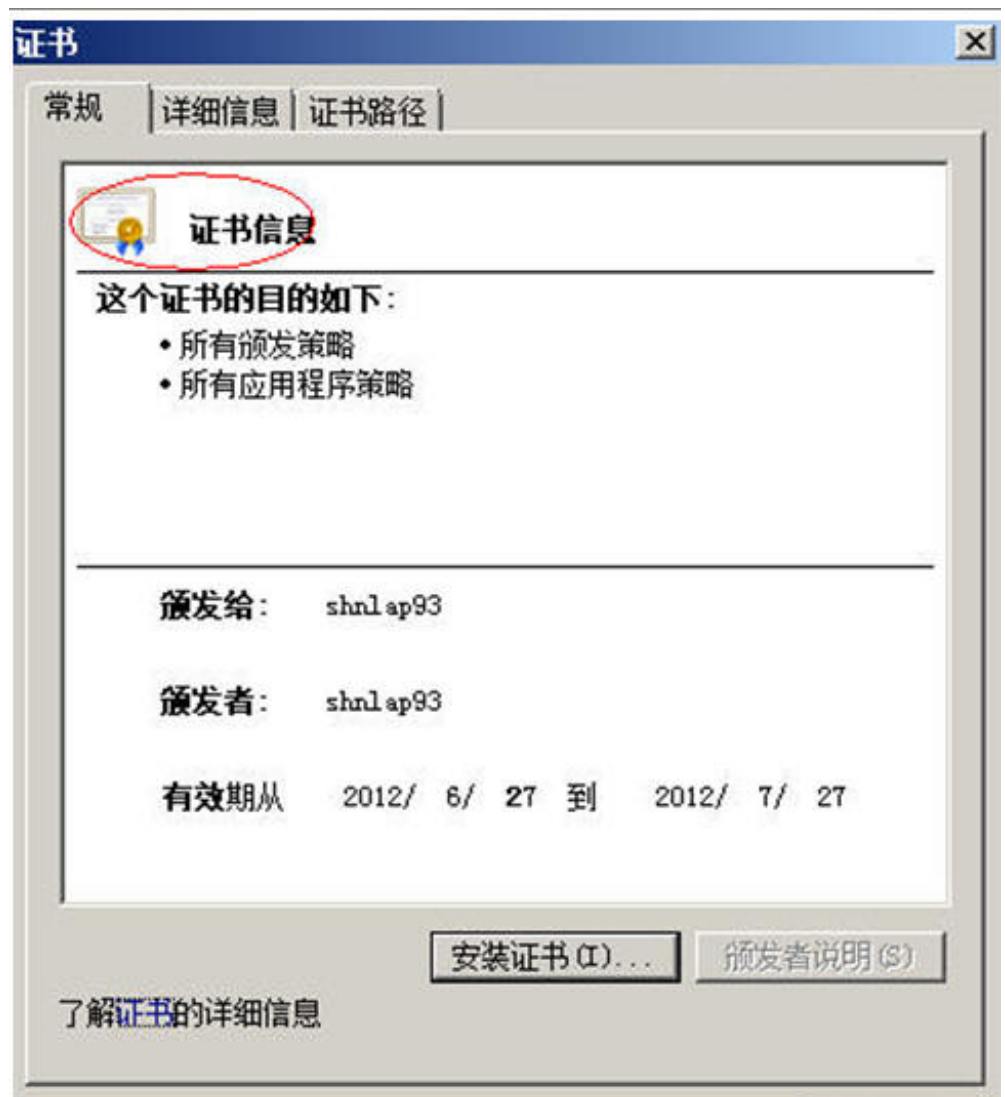
這行默認是被註釋掉的，因此請把它放開，修改成如下

```
Include conf/extra/httpd-ssl.conf
```

ü 打開

D:\tools\httpd\conf\extra\裡的
httpd-ssl.conf文件

在開頭處添加如下這一行語句



```
#  
# This is the Apache server configuration file providing SSL support.  
# It contains the configuration directives to instruct the server how to  
# serve pages over an https connection. For detailing information about these  
# directives see <URL:http://httpd.apache.org/docs/2.2/mod/mod_ssl.html>  
#  
# Do NOT simply read the instructions in here without understanding  
# what they do. They're here only as hints or reminders. If you are unsure  
# consult the online docs. You have been warned.  
#  
LoadModule ssl_module modules/mod_ssl.so
```

然後找到下面這一行

SSLCertificateFile
"D:/tools/httpd/

把它改成：

```
管理员： 命令提示符

D:\ca>openssl genrsa -des3 -out server.key 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:

D:\ca>
```

```
管理员： 命令提示符

D:\ca>openssl rsa -in server.key -out server.key
Enter pass phrase for server.key:
writing RSA key

D:\ca>
```

SSLCertificateFile "D:/tools/httpd/cert/server.crt"

再找到下面这一行

把它改成

```
D:\ca>openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:SH
Locality Name (eg, city) []:SH
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Cognizant
Organizational Unit Name (eg, section) []:Insurance-DART
Common Name (eg, YOUR name) []:shnlap93
Email Address []:
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:aaaaaa
An optional company name []:aaaaaa
```

```
D:\ca>
```

```
D:\ca>openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key
Using configuration from C:\openssl\bin\openssl.cnf
Loading 'screen' into random state - done
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 4 (0x4)
    Validity
        Not Before: Jun 27 04:54:04 2012 GMT
        Not After : Jun 27 04:54:04 2013 GMT
    Subject:
        countryName           = CN
        stateOrProvinceName   = SH
        organizationName      = Cognizant
        organizationalUnitName = Insurance-DART
        commonName             = shnlap93
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            45:BF:AC:49:7A:D1:ED:5F:7C:E3:E6:20:B8:FF:E9:01:07:3F:63:03
        X509v3 Authority Key Identifier:
            keyid:67:D7:C5:94:DF:7F:9E:CB:4D:54:36:4A:15:22:CC:DA:C1:0C:AD:8
6
Certificate is to be certified until Jun 27 04:54:04 2013 GMT (365 days)
Sign the certificate? [y/n]:
```

SSLCertificateKeyFile

```
"D:\tools\httpd\conf\extra\httpd-ssl.conf"
Certificate is to be certified until Jun 27 04:54:04 2013 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
D:\ca>
```

然後
把我們
在我們

們的Apache HttpServer的安裝目錄下手工建一個目錄叫cert的目錄，並把我們在前面生成的server.crt與server.key文件拷入d:\tools\httpd\cert目錄內。

在httpd.conf文件中搜索「ServerName」

搜到下面這樣的一句

```
ServerName 10.225.101.35:80
```

把它改成你的主機名

```
ServerName shnlap93:80
```

此處的shnlap93是你的主機名

再繼續在httpd.conf文件中搜索「VirtualHost *」

搜到下面這一句

```
<VirtualHost *>
```

把它改成

```
<VirtualHost shnlap93:80>
```

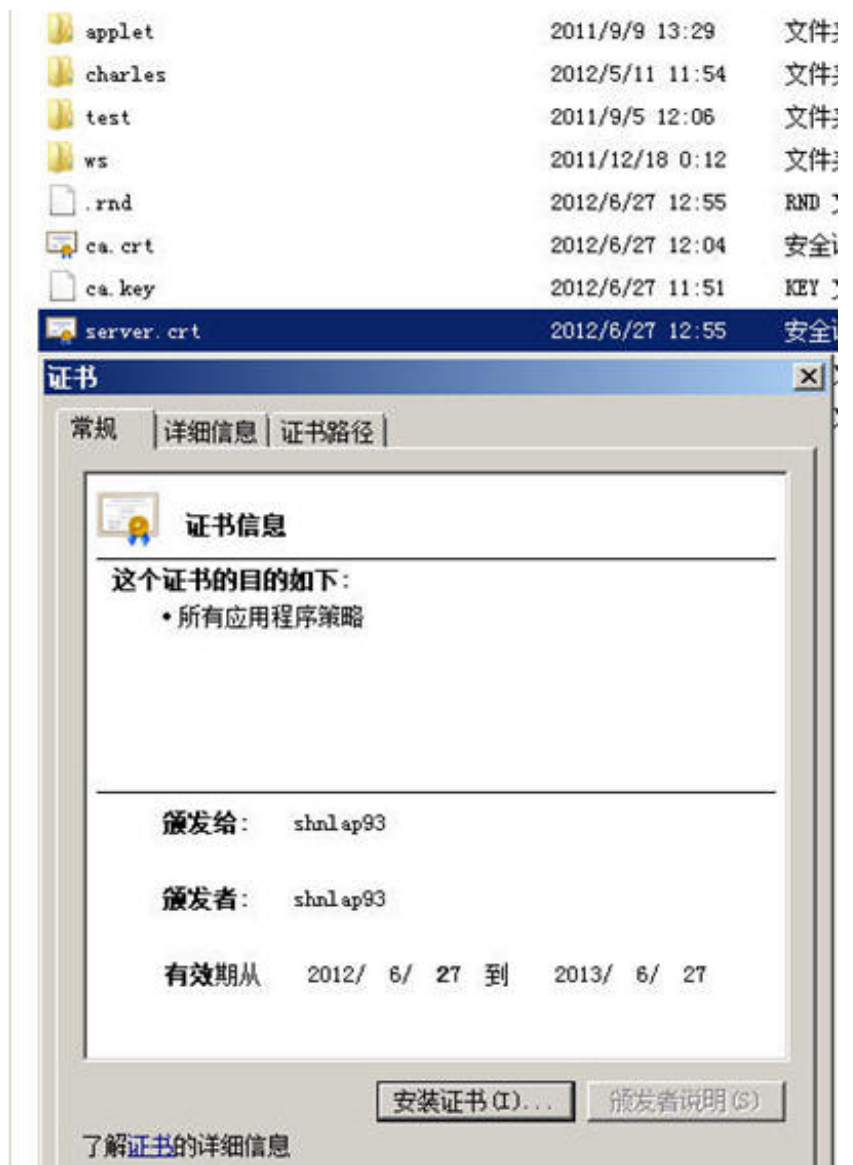
在D:\tools\httpd\conf\extra\httpd-ssl.conf文件中查找

搜「VirtualHost _default_:443」

然後把位於< VirtualHost _default_:443>段內的頭三行改成如下格式

² 確保你的http的發布目錄在d:/www

² 確保你的HTTPS的主機名為shnlap93:443(這邊的名字和生成證書裡的common name必須完全一模一樣連大小寫都必須一樣)



DocumentRoot
"D:/www"

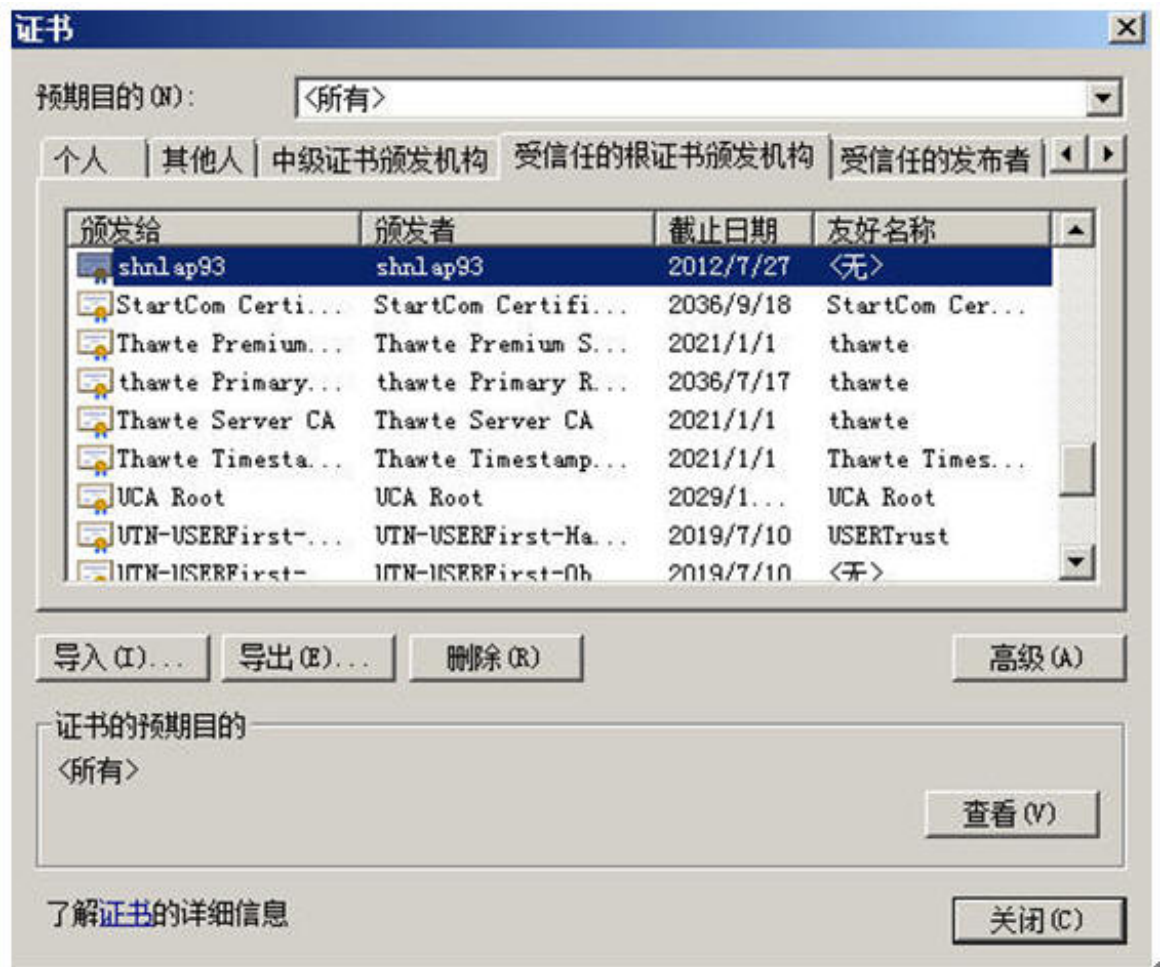
ServerName
shnlap93:443

ServerAdmin
admin@localhost

然後在下一個

「</VirtualHost>」

結束前，填入下面這
幾行語句



DirectoryIndex index.html index.htm index.jsp index.action

JkMount /*WEB-INF ajp13

JkMount /*j_spring_security_check ajp13

JkMount /*.action ajp13

JkMount /servlet/* ajp13

JkMount /*.jsp ajp13

JkMount /*.do ajp13

JkMount /*.action ajp13

JkMount /*ckeditor/editor/filemanager/connectors/*.* ajp13

JkMount /ckeditor/editor/filemanager/connectors/* ajp13

在重啟我們的Apache服務前先Test Configuration一下，如果一切無誤，可以重啟了。

然後我們來實驗一下我們的Web Server的https的效果

看到沒有，這個紅圈圈起來的地方，目前是正常的，顯示金黃色的一把鑰匙。

如果你的Root CA沒有裝入IE的根級信任域，此時你敲

入<https://shnlap93/cbbs>時，你會被提示說「該證書不被任何」，然後讓你點一下「確認」按鈕，點完信任後能進入我們的Web應用，但是，原先應該顯示「金黃色鑰匙」的地方會顯示一個紅色的圈圈，並且當你查看證書信息時，這個地方也會顯示「證書不受信任」，並且顯示一個紅色的大叉。

2.7 為Tomcat也佈署https協議

我們的Apache HttpServer已經走https協議了，不是已經enough了嗎？NO，遠遠不夠，如果你沒有用到任何App Server即tomcat/weblogic/was那麼我們說我們的應用已經走https協議了，但是因為我們的架構是Web Server + App Server，因此，我們的App Server也必須走https協議。

如果只是Web Server走https協議，而App Server沒有走https協議，這就叫「假https架構」，是一種極其偷懶和不負責任的做法。

概念同產生Apache的HttpServer的證書一樣，只是這邊的信任域有點不一樣。

Web的信任域就是你的IE裡的内容裡的證書裡的「根級信任域」，App Server的信任域是打不開也不能訪問這塊地方的，而且App Server的信任域格式也不是crt文件，而是.jks(java key store的簡稱)。

下面來做一個tomcat的https佈署。

原有ca.crt和ca.key繼續有用，因為ROOT CA都是一個，而且必須一定始終是唯一的一個，對吧？

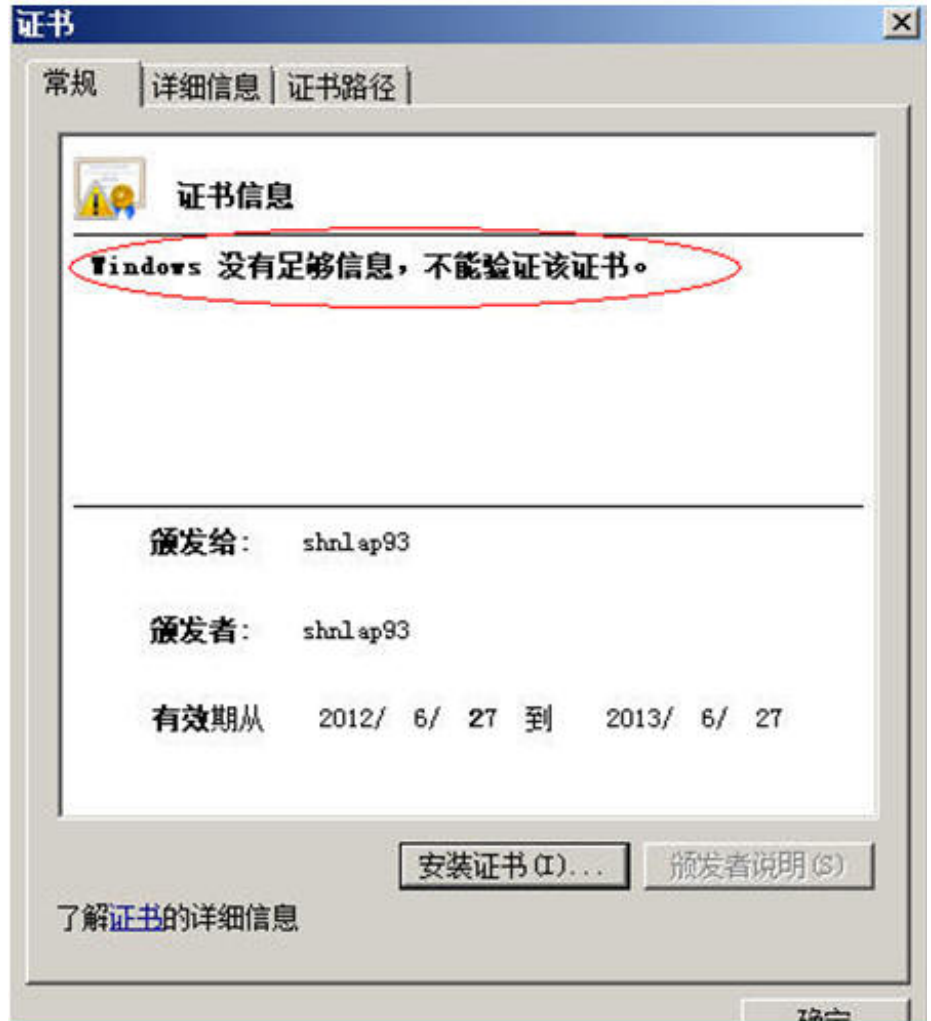
我們直接從server.jks來做起。

說JKS，這東西好玩的很，jks文件其實就是把key文件與crt文件合在一起，以java key store的格式來存儲而已。

2.8 生成Tomcat的SSL證書

為Tomcat的server所在的服務器生成一個server.jks文件

很多網上的資料是拿原先的server.crt文件轉成keystore文件，其實是不對的，需要單獨生成一張server.jks文件。



怎麼生成證書？回顧一下上文：

- 1) 生成KEY
- 2) 生成證書請求
- 3) 用CA簽名

下面開始使用%JAVA_HOME%\bin目錄下的keytool工具來產生證書

ü 生成JKS密鑰對，密碼使用6個a，alias代表「別名」，CN代表Common Name，必須與主機名完全一致，錯了不要怪我自己負責。

```
keytool -genkey -alias shnlap93X509 -keyalg RSA -keysize 1024 -dname "CN=shnlap93, OU=insurance-dart, O=Cognizant, L=SH, S=SH, C=CN" -keypass aaaaaa -keystore shnlap93.jks -storepass aaaaaa
```

ü 生成JSK的CSR

```
keytool -certreq -alias shnlap93X509 -sigalg "MD5withRSA" -file shnlap93.csr -keypass aaaaaa -keystore shnlap93.jks -storepass aaaaaa
```

此處注意：

Alias名必須和上面一致

密碼和上面一致

ü 使用openssl結合ca.crt與ca.key為jsk的csr來簽名認證並產生jks格式的crt

```
openssl x509 -req -in shnlap93.csr -out shnlap93.crt -CA ca.crt -CAkey ca.key -days 3650 -CAcreateserial -sha1 -trustout -CA ca.crt -CAkey ca.key -days 3650 -CAserial ca.srl -sha1 -trustout
```

提示yes和no時選yes

於是，我們有了一張符合jks格式的crt證書叫shnlap93.crt文件，來查看它。

證書上沒有紅色的大叉，因為我們的Root CA裝在我們的IE的根級信任域中，OK，下面來了，生成這個JKS，就是把crt和key合在一起，來了！

ü 生成符合x509格式的jks文件

1) 將Root CA導入jks信任域

```
keytool -import -alias rootca -trustcacerts -file ca.crt -keystore shnlap93.jks -storepass aaaaaa
```

前面我說了，jks信任域是讀不到IE的根級信任域的，因此要手動把ca.crt文件導入jks的信任域

1) 現在我們的shnlap93.jks文件中有兩個realm(域)，一個是：

2 本身的csr(產生的簽書請求認證的域，還沒被認證，因為認證的內容變成了shnlap93.crt文件了是不是？)

2 剛才步驟中導入的Root CA認證域

那麼客戶端信任Root CA沒有問題，但由於server端本身的信任域還只是處於「請求被簽名」的狀態，那麼客戶端如何去信任這個jks文件呢？

答案就是：補鏈，補這根信任鏈！

```
keytool -import -alias shnlap93X509 -file shnlap93.crt -keystore shnlap93.jks -storepass aaaaaa
```

我們不是生成過shnlap93.crt嗎？把它導入jks不就是能夠把原有的「正在處於請求被簽名認證」這個狀態改成「已經被Root CA簽名認證」了嗎？對吧？

所以，我們用於佈署tomcat的ssl證書的jks格式的文件shnlap93.jks已經完整了。

2.9 佈署Tomcat上的Https協議

Apache HttpServer走的是443端口，Tomcat走的就是8443端口。

打開tomcat的conf目錄下的server.xml，我們來找下面這一段：



```

<!--

<Connector
executor="tomcatThreadPool"

    port="8080"
    protocol="HTTP/1.1"

    connectionTimeout="20000"

    redirectPort="8443"

/>

-->

```

默認情況下，它是由「<!-- -->」這樣的標籤給註釋起來的，我們把它放開

```

D:\ca>keytool -import -alias rootca -trustcacerts -file ca.crt -keystore shnlap93.jks -storepass aaaaaa
所有者:CN=shnlap93, OU=Insurance, O=Cognizant, L=SH, ST=SH, C=CN
簽發人:CN=shnlap93, OU=Insurance, O=Cognizant, L=SH, ST=SH, C=CN
序列號:0
有效期: Wed Jun 27 12:04:26 CST 2012 至 Fri Jul 27 12:04:26 CST 2012
证书指纹:
    MD5:98:94:B6:8B:E1:F2:02:D8:0A:2B:A9:AD:FF:F4:CE:D3
    SHA1:12:11:D1:E2:44:D8:E4:EE:70:3E:05:2E:F7:00:CE:70:A8:D9:01:33
    签名算法名称:MD5withRSA
    版本: 3

扩展:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 67 D7 C5 94 DF 7F 9E CB 4D 54 36 4A 15 22 CC DA g.....MT6J..."
0010: C1 0C AD 86 ....
]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
CA:true
PathLen:2147483647
]

#3: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 67 D7 C5 94 DF 7F 9E CB 4D 54 36 4A 15 22 CC DA g.....MT6J..."
0010: C1 0C AD 86 ....
]
]

[CN=shnlap93, OU=Insurance, O=Cognizant, L=SH, ST=SH, C=CN]
SerialNumber: [ 00]
]

信任这个认证? [否]: y
认证已添加至keystore中

D:\ca>

```

```

C:\>管理员: 命令提示符

D:\ca>keytool -import -alias shnlap93X509 -file shnlap93.crt -keystore shnlap93.jks -storepass aaaaaa
认证回复已安装在 keystore 中

D:\ca>

```

```

<!-- enable tomcat ssl -->

    port="8443" protocol="HTTP/1.1"
    connectionTimeout="20000"
    secure="true" SSLEnabled="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreFile="d:/tomcat/conf/shnlap93.jks" keystorePass="aaaaaa"

/>

```

² clientAuth=「 false」

如果該值為「true」就代表要啟用雙向認證

² keystoreFile就是我們生成的keystore文件所在的完全路徑

² keystorePass就是我們生成keystore時的password

重啟tomcat,輸入<https://shnlap93:8080/cbbs>, 可以看到登錄網頁,且右上方的SSL連接信息為「金黃色的鑰匙」而不是紅色大叉,那麼一切就成功了。

重啟Apache,然後在ie地址欄輸入: <https://shnlap93/cbbs>, 用sally/abcdefg, 一切成功。

注意：

當啟用了https協議後，你在ie地址欄就不能再用localhost了，為什麼？因為我們的證書中的CN(Common Name)填入的是主機名，如果你還用:<https://localhost>，那麼你也能正常進入網址，只是多了幾步https不被信任的警告框，並且你右上角的ssl連接信息為紅色的大叉，對於這樣的https連接，如果換成是購物網站，你敢信任嗎？

2.10 apache https + tomcat https

² 假https

前面說過，如果你是在Apache上啟用了https而沒有在tomcat上啟用https協議，那麼我們在tomcat中佈署一個servlet，含一條打印語句：System.out.println(「」+request.getScheme())，那麼它將打印出來**http**。

² 真https

如果我們在Apache上啟用了https通時在tomcat上也啟用了https，那麼我們如果有這樣的一條語句：System.out.println(「」+request.getScheme())，它打印出來的將是：**https**。