**Https**

## 一、HTTPS和SSL/TLS协议

**1 要说清楚 HTTPS 协议的实现原理，至少需要如下几个背景知识。**

大致了解几个基本术语（HTTPS、SSL、TLS）的含义

大致了解 HTTP 和 TCP 的关系（尤其是“短连接”VS“长连接”）

大致了解加密算法的概念（尤其是“对称加密与非对称加密”的区别）

大致了解 CA 证书的用途

**2. “HTTP”是干嘛用滴？**

首先，HTTP 是一个网络协议，是专门用来帮你传输 Web 内容滴。关于这个协议，就算你不了解，至少也听说过吧？比如你访问俺的博客的主页，浏览器地址栏会出现如下的网址

大部分网站都是通过 HTTP 协议来传输 Web 页面、以及 Web 页面上包含的各种东东（图片、CSS 样式、JS 脚本）。超文本传输协议HTTP协议被用于在Web浏览器和网站服务器之间传递信息，HTTP协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此，HTTP协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。为了解决HTTP协议的这一缺陷，需要使用另一种协议：安全套接字层超文本传输协议HTTPS，为了数据传输的安全，HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

HTTPS：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。

**HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。**

**3. “SSL/TLS”是干嘛用滴？**

SSL 是洋文“Secure Sockets Layer”的缩写，中文叫做“*安全套接层*”。它是在上世纪90年代中期，由网景公司设计的。（顺便插一句，网景公司不光发明了 SSL，还发明了很多 Web 的基础设施——比如“CSS 样式表”和“JS 脚本”）为啥要发明 SSL 这个协议捏？因为原先互联网上使用的 HTTP 协议是明文的，存在很多缺点——比如传输内容会被偷窥（嗅探）和篡改。发明 SSL 协议，就是为了解决这些问题。到了1999年，SSL 因为应用广泛，已经成为互联网上的事实标准。IETF 就在那年把 SSL 标准化。标准化之后的名称改为 TLS（是“Transport Layer Security”的缩写），中文叫做“传输层安全协议”。很多相关的文章都把这两者并列称呼（SSL/TLS），因为**这两者可以视作同一个东西的不同阶段**。

**4. “HTTPS”是啥意思？**

解释完 HTTP 和 SSL/TLS，现在就可以来解释 HTTPS 啦。咱们通常所说的 HTTPS 协议，说白了就是**“HTTP 协议”和“SSL/TLS 协议”的组合**。你可以把 HTTPS 大致理解为——“HTTP over SSL”或“HTTP over TLS”（反正 SSL 和 TLS 差不多）。

**5. HTTP 的版本和历史**

如今咱们用的 HTTP 协议，版本号是 1.1（也就是 HTTP 1.1）。这个 1.1 版本是1995年底开始起草的（技术文档是 RFC2068），并在1999年正式发布（技术文档是 RFC2616）。

在 1.1 之前，还有曾经出现过两个版本“0.9 和 1.0”，其中的 HTTP 0.9 【没有】被广泛使用，而 HTTP 1.0 被广泛使用过。另外，据说明年（2015）IETF 就要发布 HTTP 2.0 的标准了。俺拭目以待。

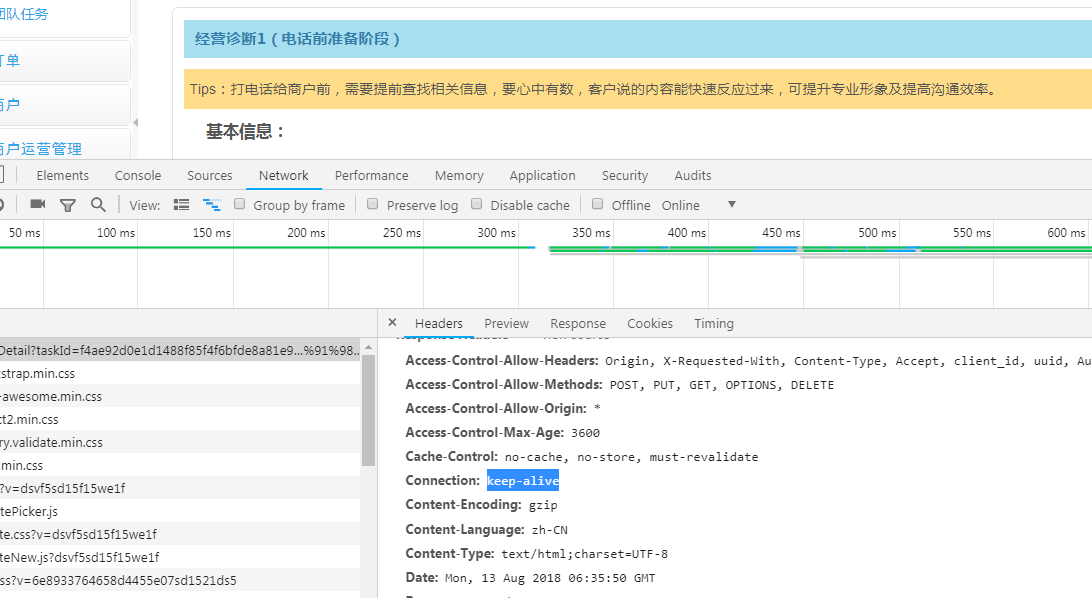
**6 HTTP 和 TCP 之间的关系**

简单地说，**TCP 协议是 HTTP 协议的基石**——HTTP 协议需要依靠 TCP 协议来传输数据。在网络分层模型中，TCP 被称为“传输层协议”，而 HTTP 被称为“应用层协议”。有很多常见的应用层协议是以 TCP 为基础的，比如“FTP、SMTP、POP、IMAP”等。TCP 被称为“面向连接”的传输层协议。关于它的具体细节，俺就不展开了（否则篇幅又失控了）。你只需知道：传输层主要有两个协议，分别是 TCP 和 UDP。TCP 比 UDP 更可靠。你可以把 TCP 协议想象成某个水管，发送端这头进水，接收端那头就出水。并且 TCP 协议能够确保，先发送的数据先到达（与之相反，UDP 不保证这点）。

**7. HTTP 协议如何使用 TCP 连接**

HTTP 对 TCP 连接的使用，分为两种方式：俗称“短连接”和“长连接”（“长连接”又称“持久连接”，洋文叫做“Keep-Alive”或“Persistent Connection”）假设有一个网页，里面包含好多图片，还包含好多【外部的】CSS 文件和 JS 文件。在“短连接”的模式下，浏览器会先发起一个 TCP 连接，拿到该网页的 HTML 源代码（拿到 HTML 之后，这个 TCP 连接就关闭了）。然后，浏览器开始分析这个网页的源码，知道这个页面包含很多外部资源（图片、CSS、JS）。然后针对【每一个】外部资源，再分别发起一个个 TCP 连接，把这些文件获取到本地（同样的，每抓取一个外部资源后，相应的 TCP 就断开）相反，如果是“长连接”的方式，浏览器也会先发起一个 TCP 连接去抓取页面。但是抓取页面之后，该 TCP 连接并不会立即关闭，而是暂时先保持着（所谓的“Keep-Alive”）。然后浏览器分析 HTML 源码之后，发现有很多外部资源，就用刚才那个 TCP 连接去抓取此页面的外部资源。

在 HTTP 1.0 版本，【默认】使用的是“短连接”（那时候是 Web 诞生初期，网页相对简单，“短连接”的问题不大）；到了1995年底开始制定 HTTP 1.1 草案的时候，网页已经开始变得复杂（网页内的图片、脚本越来越多了）。这时候再用短连接的方式，效率太低下了（因为建立 TCP 连接是有“时间成本”和“CPU 成本”滴）。所以，在 HTTP 1.1 中，【默认】采用的是“Keep-Alive”的方式。关于“Keep-Alive”的更多介绍，可以参见维基百科词条（在“这里”）



**8. 啥是“加密”和“解密”**

通俗而言，你可以把“加密”和“解密”理解为某种【互逆的】数学运算。就好比“加法和减法”互为逆运算、“乘法和除法”互为逆运算。**“加密”的过程，就是把“明文”变成“密文”的过程；反之，“解密”的过程，就是把“密文”变为“明文”。在这两个过程中，都需要一个关键的东东——叫做“密钥”——来参与数学运算。**

**9. 啥是“对称加密”**

所谓的“对称加密技术”，意思就是说：“加密”和“解密”使用【相同的】密钥。这个比较好理解。就好比你用 7zip 或 WinRAR 创建一个带密码（口令）的加密压缩包。当你下次要把这个压缩文件解开的时候，你需要输入【同样的】密码。在这个例子中，密码/口令就如同刚才说的“密钥”。

**10. 啥是“非对称加密”**

所谓的“非对称加密技术”，意思就是说：“加密”和“解密”使用【不同的】密钥。这玩意儿比较难理解，也比较难想到。当年“非对称加密”的发明，还被誉为“密码学”历史上的一次革命。

由于篇幅有限，对“非对称加密”这个话题，俺就不展开了。有空的话，再单独写一篇扫盲。

**11. 各自有啥优缺点**

看完刚才的定义，很显然：（从功能角度而言）“非对称加密”能干的事情比“对称加密”要多。这是“非对称加密”的优点。但是“非对称加密”的实现，通常需要涉及到“复杂数学问题”。所以，“非对称加密”的性能通常要差很多（相对于“对称加密”而言）。这两者的优缺点，也影响到了 SSL 协议的设计。

**12 CA 证书的原理及用途**

关于这方面，请看俺4年前写的《数字证书及CA的扫盲介绍》。这里就不再重复唠叨了，免得篇幅太长。

**13 HTTPS 协议的需求是啥**

花了好多口水，终于把背景知识说完了。下面正式进入正题。先来说说当初设计 HTTPS 是为了满足哪些需求？很多介绍 HTTPS 的文章一上来就给你讲实现细节。个人觉得：这是不好的做法。早在2009年开博的时候，发过一篇《学习技术的三部曲：WHAT、HOW、WHY》，其中谈到“WHY 型问题”的重要性。一上来就给你讲协议细节，你充其量只能知道 WHAT 和 HOW，无法理解 WHY。俺在前一个章节讲了“背景知识”，在这个章节讲了“需求”。

**14 兼容性**

因为是先有 HTTP 再有 HTTPS。所以，HTTPS 的设计者肯定要考虑到对原有 HTTP 的兼容性。这里所说的兼容性包括很多方面。比如已有的 Web 应用要尽可能无缝地迁移到 HTTPS；比如对浏览器厂商而言，改动要尽可能小基于“兼容性”方面的考虑，很容易得出如下几个结论

（1） HTTPS 还是要基于 TCP 来传输，如果改为 UDP 作传输层，无论是 Web 服务端还是浏览器客户端，都要大改，动静太大了

（2） 单独使用一个新的协议，把 HTTP 协议包裹起来

所谓的“HTTP over SSL”，实际上是在原有的 HTTP 数据外面加了一层 SSL 的封装。HTTP 协议原有的 GET、POST 之类的机制，基本上原封不动

打个比方：如果原来的 HTTP 是塑料水管，容易被戳破；那么如今新设计的 HTTPS 就像是在原有的塑料水管之外，再包一层金属水管。一来，原有的塑料水管照样运行；二来，用金属加固了之后，不容易被戳破。

**15 可扩展性**

前面说了，HTTPS 相当于是“HTTP over SSL”。如果 SSL 这个协议在“可扩展性”方面的设计足够牛逼，那么它除了能跟 HTTP 搭配，还能够跟其它的应用层协议搭配。岂不美哉？

现在看来，当初设计 SSL 的人确实比较牛。如今的 SSL/TLS 可以跟很多常用的应用层协议（比如：FTP、SMTP、POP、Telnet）搭配，来强化这些应用层协议的安全性。接着刚才打的比方：如果把 SSL/TLS 视作一根用来加固的金属管，它不仅可以用来加固输水的管道，还可以用来加固输煤气的管道。

**16 保密性（防泄密）**

HTTPS 需要做到足够好的保密性。说到保密性，首先要能够对抗嗅探（行话叫 Sniffer）。所谓的“嗅探”，通俗而言就是监视你的网络传输流量。如果你使用明文的 HTTP 上网，那么监视者通过嗅探，就知道你在访问哪些网站的哪些页面。嗅探是最低级的攻击手法。除了嗅探，HTTPS 还需要能对抗其它一些稍微高级的攻击手法——比如“重放攻击”（后面讲协议原理的时候，会再聊）。

**17 完整性（防篡改）**

除了“保密性”，还有一个同样重要的目标是“确保完整性”。关于“完整性”这个概念，在之前的博文《扫盲文件完整性校验——关于散列值和数字签名》中大致提过。健忘的同学再去温习一下。在发明 HTTPS 之前，由于 HTTP 是明文的，不但容易被嗅探，还容易被篡改。

举个例子：

比如咱们天朝的网络运营商（ISP）都比较流氓，经常有网友抱怨说访问某网站（本来是没有广告的），竟然会跳出很多中国电信的广告。为啥会这样捏？因为你的网络流量需要经过 ISP 的线路才能到达公网。如果你使用的是明文的 HTTP，ISP 很容易就可以在你访问的页面中植入广告。所以，当初设计 HTTPS 的时候，还有一个需求是“确保 HTTP 协议的内容不被篡改”。

**18 真实性（防假冒）**

在谈到 HTTPS 的需求时，“真实性”经常被忽略。其实“真实性”的重要程度不亚于前面的“保密性”和“完整性”。

举个例子：

你因为使用网银，需要访问该网银的 Web 站点。那么，你如何确保你访问的网站确实是你想访问的网站？（这话有点绕口令）

有些天真的同学会说：通过看网址里面的域名，来确保。为啥说这样的同学是“天真的”？因为 DNS 系统本身是不可靠的（尤其是在设计 SSL 的那个年代，连 DNSSEC 都还没发明）。由于 DNS 的不可靠（存在“域名欺骗”和“域名劫持”），你看到的网址里面的域名【未必】是真实滴！

**19 性能**

再来说最后一个需求——性能。

引入 HTTPS 之后，【不能】导致性能变得太差。否则的话，谁还愿意用？

为了确保性能，SSL 的设计者至少要考虑如下几点

（1） 如何选择加密算法（“对称”or“非对称”）？

（2）如何兼顾 HTTP 采用的“短连接”TCP 方式？

（SSL 是在1995年之前开始设计的，那时候的 HTTP 版本还是 1.0，默认使用的是“短连接”的 TCP 方式——默认不启用 Keep-Alive）

**20 HTTPS和HTTP的区别**

（1）https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

（2）http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

（3）http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

（4）http的连接很简单，是无状态的；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，比http协议安全。

**21 HTTPS的工作原理**

我们都知道HTTPS能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用HTTPS协议。

HTTP与HTTPS的区别-马海祥博客。客户端在使用HTTPS方式与Web服务器通信时有以下几个步骤，如图所示。

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。

**22、HTTPS的优点**

尽管HTTPS并非绝对安全，掌握根证书的机构、掌握加密算法的组织同样可以进行中间人形式的攻击，但HTTPS仍是现行架构下最安全的解决方案，主要有以下几个好处

　　（1）使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；

　　（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解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

　　（4）谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。

**23、HTTPS的缺点**

虽然说HTTPS有很大的优势，但其相对来说，还是存在不足之处的

　　（1）HTTPS协议握手阶段比较费时，会使页面的加载时间延长近50%，增加10%到20%的耗电；

　　（2）HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

　　（3）SSL证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

　 （4）SSL证书通常需要绑定IP，不能在同一IP上绑定多个域名，IPv4资源不可能支撑这个消耗。

　　（5）HTTPS协议的加密范围也比较有限，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用。最关键的，SSL证书的信用链体系并不安全，特别是在某些国家可以控制CA根证书的情况下，中间人攻击一样可行。

**24、http切换到HTTPS**

如果需要将网站从http切换到https到底该如何实现呢？这里需要将页面中所有的链接，例如js，css，图片等等链接都由http改为https。

**例如：[http://www.baidu.com改为https://www.baidu.com](http://www.baidu.com改为https:/www.baidu.com)**

BTW，这里虽然将http切换为了https，还是建议保留http。所以我们在切换的时候可以做http和https的兼容，具体实现方式是，去掉页面链接中的http头部，这样可以自动匹配http头和https头。例如：将http://www.baidu.com改为//www.baidu.com。然后当用户从http的入口进入访问页面时，页面就是http，如果用户是从https的入口进入访问页面，页面即使https的。

**公钥密码体制分为三个部分，公钥、私钥、加密解密算法，它的加密解密过程如下：**

@加密：通过加密算法和公钥对内容(或者说明文)进行加密，得到密文。加密过程需要用到公钥。

@解密：通过解密算法和私钥对密文进行解密，得到明文。解密过程需要用到解密算法和私钥。注意，由公钥加密的内容，只能由私钥进行解密，也就是说，由公钥加密的内容，如果不知道私钥，是无法解密的。

## 二、tomcat和https

**（一） 创建tomcat证书**

这里使用JDK自带的keytool工具来生成证书：

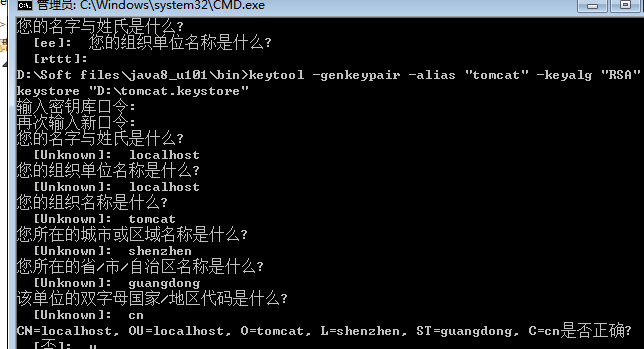
**1. 在jdk的安装目录\bin\keytool.exe下打开keytool.exe**



**2. 在命令行中输入以下命令:**

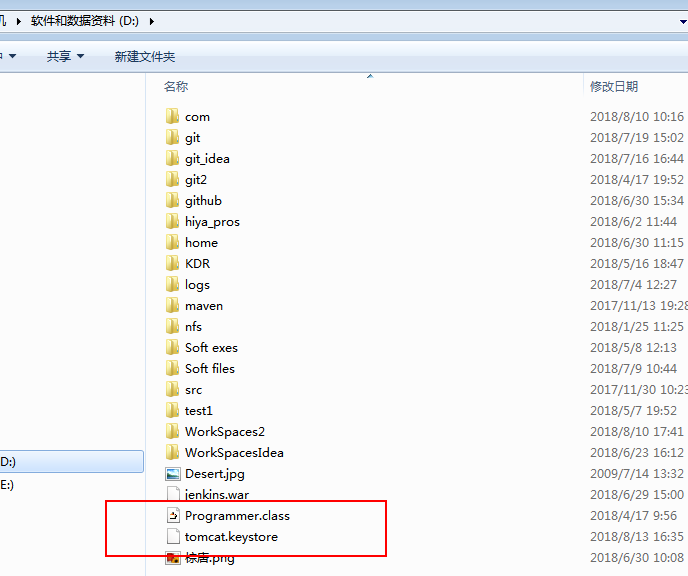
**keytool -genkeypair -alias "tomcat" -keyalg "RSA" -keystore "D:\tomcat.keystore"**





以上命令将生产一对非对称密钥和自我签名的证书g:\tomcat.keystore

注意：“名字与姓氏”应该是域名，输成了姓名，和真正运行的时候域名不符，会出问题这里我输入的密码是123456,  域名是以tomcat为例,  省市以广东深圳为例

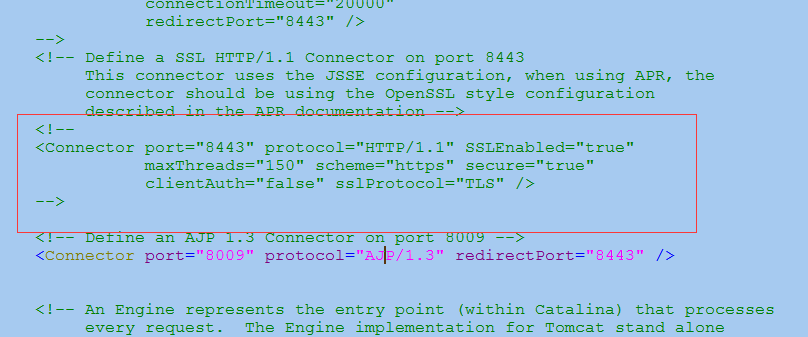


**（二） 配置tomcat服务器**

 定位到tomcat服务器的安装目录, 找到conf下的server.xml文件



找到如下已经被注释的代码：



1 <!--2 <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"3 maxThreads="150" scheme="https" secure="true"4 clientAuth="false" sslProtocol="TLS" />5 -->

去掉注释，修改为：

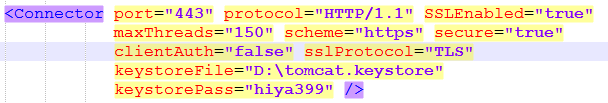
1<Connector port="443" protocol="HTTP/1.1" SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

clientAuth="false" sslProtocol="TLS"

keystoreFile="D:\tomcat.keystore"

keystorePass="hiya399" />



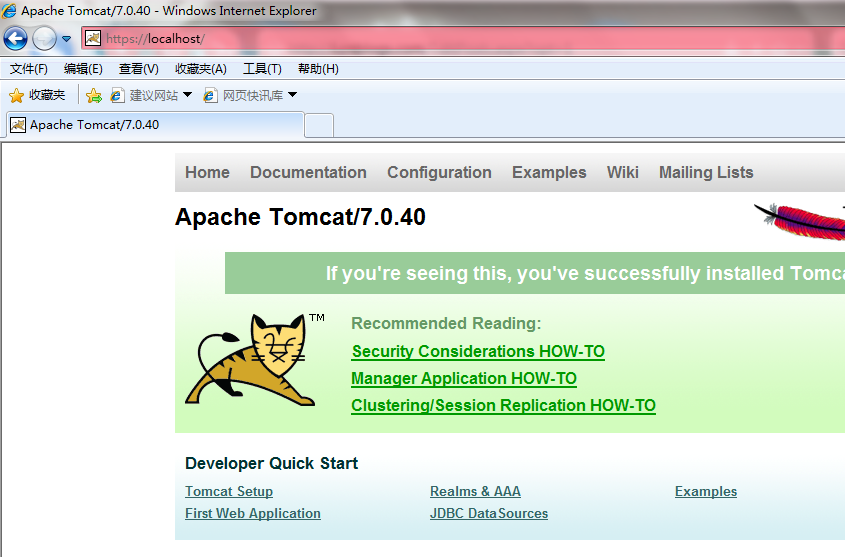
在互联网上, http协议的默认端口是80, https的默认端口是443, 这里将端口改为了443

**（三）启动tomcat服务器**

在IE浏览器中输入: <https://localhost>



选择继续浏览此网站



遇到的问题:

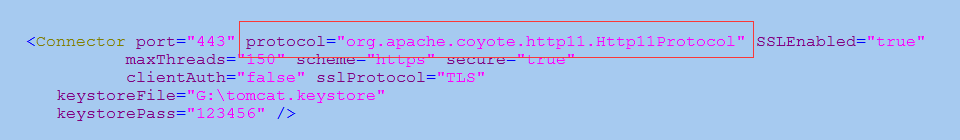
当修改了server.xml的配置后，启动tomcat报错

[IMG_263](https://www.cnblogs.com/wanghaoyuhappy/p/javascript:void(0);)

1 org.apache.catalina.core.StandardService initInternal 2 严重: Failed to initialize connector [Connector[HTTP/1.1-443]] 3 org.apache.catalina.LifecycleException: Failed to initialize component [Connector[HTTP/1.1-443]] 4 at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:106) 5 at org.apache.catalina.core.StandardService.initInternal(StandardService.java:559) 6 at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:102) 7 at org.apache.catalina.core.StandardServer.initInternal(StandardServer.java:814) 8 at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:102) 9 at org.apache.catalina.startup.Catalina.load(Catalina.java:633)10 at org.apache.catalina.startup.Catalina.load(Catalina.java:658)11 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)12 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)13 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)14 at java.lang.reflect.Method.invoke(Method.java:606)15 at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:281)16 at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:455)17 Caused by: org.apache.catalina.LifecycleException: Protocol handler initialization failed18 at org.apache.catalina.connector.Connector.initInternal(Connector.java:983)19 at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:102)20 ... 12 more21 Caused by: java.lang.Exception: Connector attribute SSLCertificateFile must be defined when using SSL with APR22 at org.apache.tomcat.util.net.AprEndpoint.bind(AprEndpoint.java:507)23 at org.apache.tomcat.util.net.AbstractEndpoint.init(AbstractEndpoint.java:610)24 at org.apache.coyote.AbstractProtocol.init(AbstractProtocol.java:429)25 at org.apache.catalina.connector.Connector.initInternal(Connector.java:981)26 ... 13 more

[IMG_264](https://www.cnblogs.com/wanghaoyuhappy/p/javascript:void(0);)

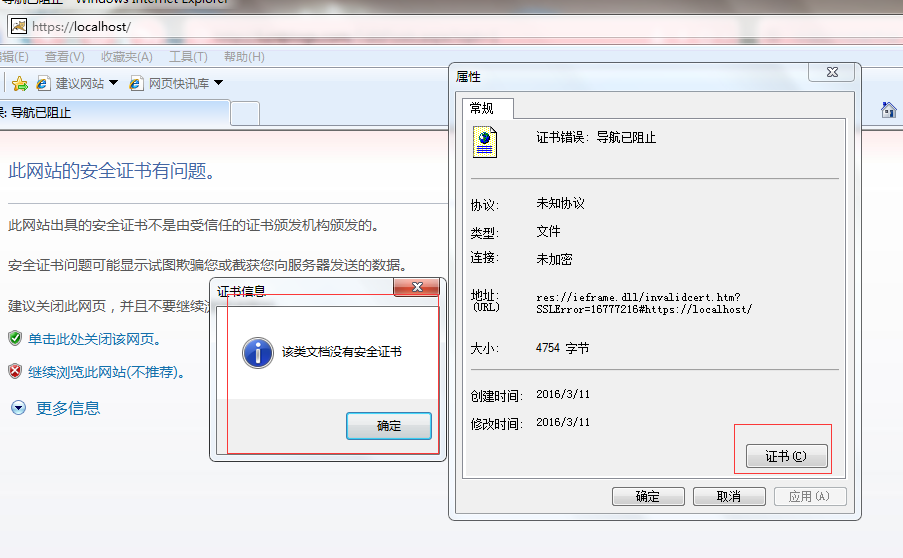
可以将protocol="HTTP/1.1"替换为protocol="org.apache.coyote.http11.Http11Protocol"，问题可以得以解决



**（四）安装tomcat的证书在浏览器中**

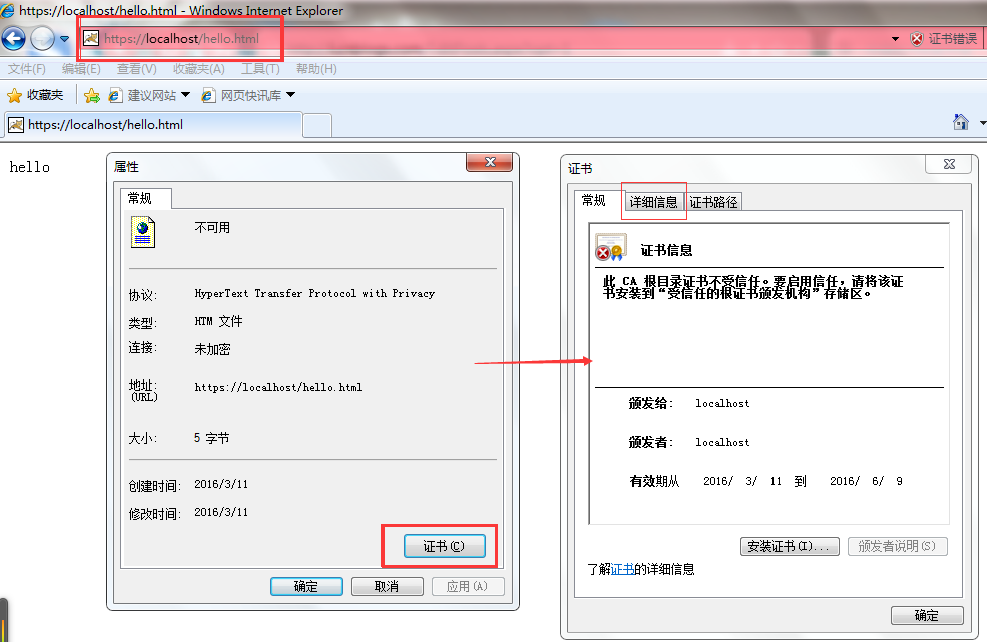
这里浏览器显示证书错误, 有红色警告

在浏览器中右击选择属性, 查看不到证书



在这里我为了能导出证书, 在tomcat的根目录下新建了一个hello.html的页面

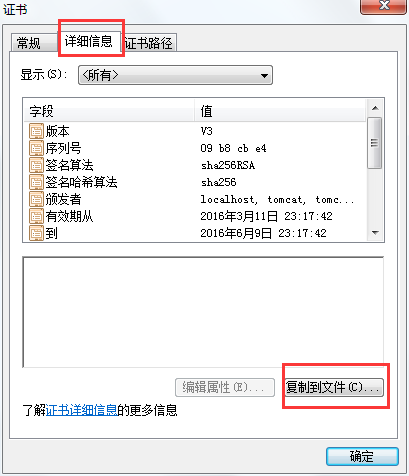
下面可以访问到



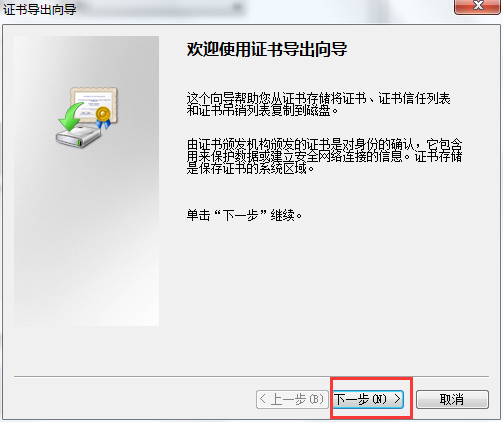
将证书导出来

注意:这个地方有部分电脑的"复制文件"的按钮是不能点的, 导不出来

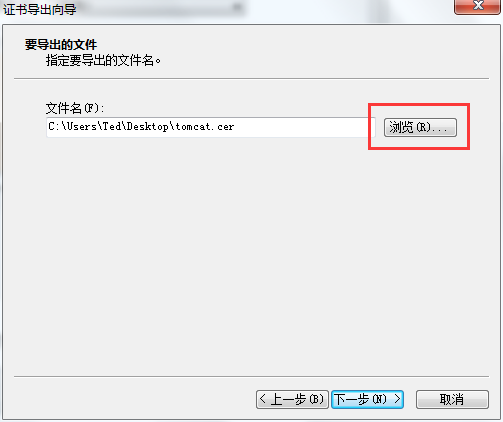
可以尝试装个虚拟机试一下

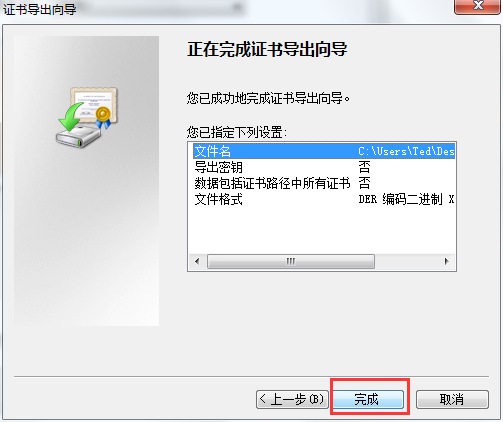


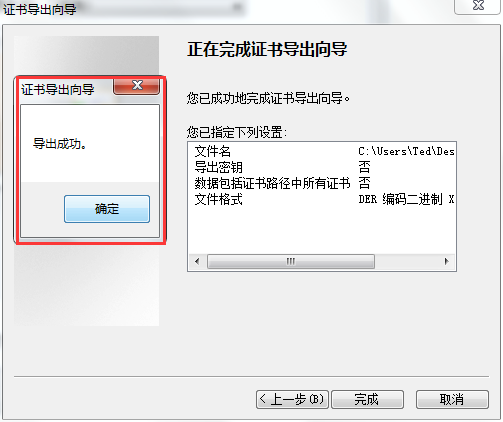
下面就下一步就行









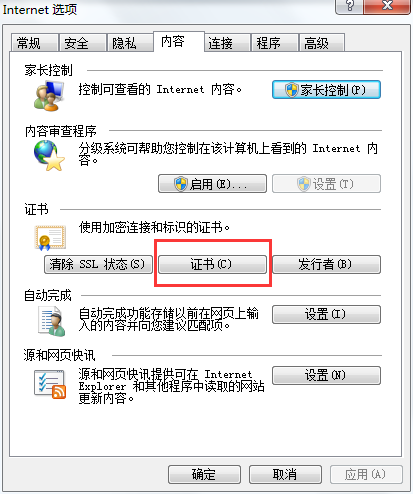


我将证书导出到了桌面上,

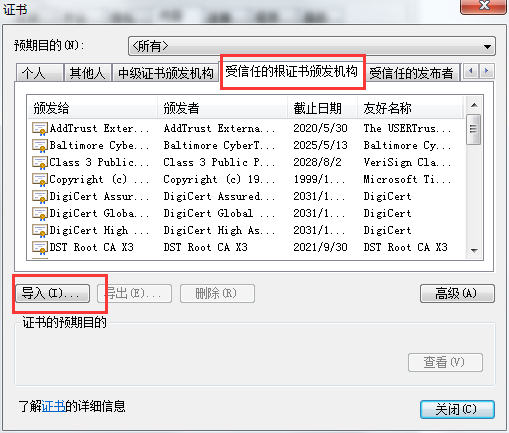
得到了一个cer的证书文件



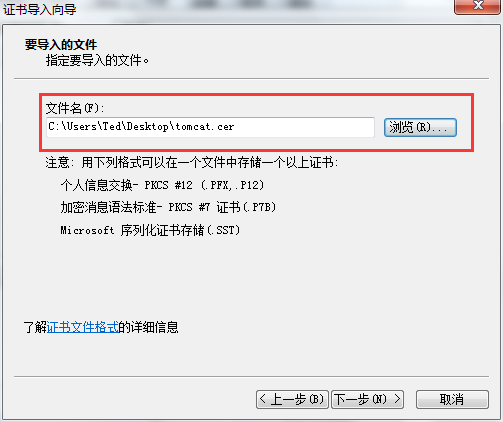
下面打开浏览器的Internet选项  ----> 内容  ----> 证书



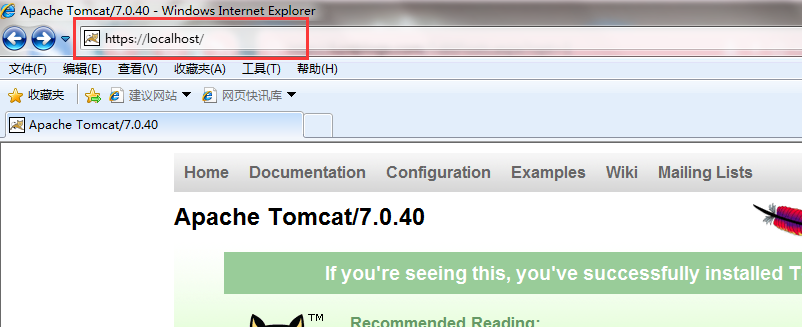
选择"受信任的根证书颁发机构"  ----> 导入证书



将生成的证书导入进来



其余的步骤直接下一步即可



## 三、nginx和https

**1、首先，进入你想创建证书和私钥的目录，例如：**

cd /etc/nginx/

**2、创建服务器私钥，命令会让你输入一个口令：**

openssl genrsa -des3 -out server.key 1024

**3、创建签名请求的证书（CSR）：**

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

**4、在加载SSL支持的Nginx并使用上述私钥时除去必须的口令：**

cp server.key server.key.org

openssl rsa -in server.key.org -out server.key

**5、最后标记证书使用上述私钥和CSR：**

openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

**6、配置nginx**

cd /etc/nginx

vim nginx.conf

#

HTTPS server configuration

#

server {

listen 443;

server\_name 本机的IP地址;

ssl on;

ssl\_certificate /etc/nginx/server.crt;

ssl\_certificate\_key /etc/nginx/server.key;

ssl\_session\_timeout 5m;

ssl\_protocols SSLv2 SSLv3 TLSv1;

ssl\_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

ssl\_prefer\_server\_ciphers on;

location / {

#root html;

#index testssl.html index.html index.htm;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://IP地址/ssl/;

}

}

**7 重启nginx，在浏览器输入**：https://IP地址 会跳转到 http://IP地址/ssl/ 这个地址（后期调整为webservice接口地址即可）

**8 受浏览器信任的StartSSL免费SSL证书：**

跟VeriSign一样，StartSSL（网址：http://www.statssl.com，公司名：StartCom）也是一家CA机构，它的根证书很 久之前就被一些具有开源背景的浏览器支持（Firefox浏览器、谷歌Chrome浏览器、苹果Safari浏览器等）。

**9 项目需要，将访问目录 \services\ 由http访问 重定向到 https （解决方法：nginx rewrite 加上 location 方式实现）**

location ~ /services/.\*$ {

if ($server\_port ~ "^80$"){

set $rule\_0 1$rule\_0;

}

if ($rule\_0 = "1"){

rewrite /(.\*) https://IP地址/$1 permanent; break;

}

}

**10 配置结束上传以后用nginx -t 测试下配置无误 就reload一下nginx服务 检查443端口是否在监听**

/usr/local/nginx/sbin/nginx -t

nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok

nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful (显示表示配置文件没有错误)

service nginx reload (重新加载nginx服务)

netstat -lan | grep 443 (查看443端口)

tcp 0 0 0.0.0.0:443 0.0.0.0:\* LISTEN (有看到这一行 就表示HTTPS已经在工作了)