## 一、shiro 单点登录原理

Shiro 1.2开始提供了Jasig CAS单点登录的支持，单点登录主要用于多系统集成，即在多个系统中，用户只需要到一个中央服务器登录一次即可访问这些系统中的任何一个，无须多次登录。

Jasig CAS单点登录系统分为服务器端和客户端，服务器端提供单点登录，多个客户端（子系统）将跳转到该服务器进行登录验证，大体流程如下：

1、访问客户端需要登录的页面http://localhost:9080/ client/，此时会跳到单点登录服务器https://localhost:8443/ server/login?service=https://localhost:9443/ client/cas；

原因：客户端访问任何地址，都会被shirofilter 拦截，shiro 拦截后发现， 应该会使用authc 进行认证 发现用户没有登录，所以会跳转到 shiroFilter 配置的loginUrl 页面

2、如果此时单点登录服务器也没有登录的话，会显示登录表单页面，输入用户名/密码进行登录；

3、登录成功后服务器端会回调客户端传入的地址：[https://localhost:9443/client/cas?ticket=ST-1-eh2cIo92F9syvoMs5DOg-cas01.example.org](https://localhost:9443/-client/cas?ticket=ST-1-eh2cIo92F9syvoMs5DOg-cas01.example.org)，且带着一个ticket；

4、客户端会把ticket提交给服务器来验证ticket是否有效；如果有效服务器端将返回用户身份；

解释：这个时候 访问/cas就会被casFileter 拦截 ，casFilter 内部创建 带ticket 的token,然后 调用subject.login方法，这时候 继承casRealm 的myrealm  进行处理。

默认认证机制 会去服务器端进行ticket 验证。

5、客户端可以再根据这个用户身份获取如当前系统用户/角色/权限信息。

## 二、官方文档注销和单点注销（SLO）

CAS单点登录会话期间可能存在许多活动的应用程序会话，注销和单点注销之间的区别基于*注销*操作时结束的会话数。注销范围取决于执行操作的位置：

1. 应用程序注销 - 结束单个应用程序会话
2. CAS注销 - 结束CAS SSO会话

请注意，在简单的情况下，每种情况下的注销操作都不会影响另一种情况。结束应用程序会话不会结束CAS会话，结束CAS会话不会影响应用程序会话。这是SSO系统的新用户和部署者混淆的常见原因。

CAS中的单一注销支持尝试协调CAS注销和应用程序注销之间的差异。当CAS配置为SLO时，它会尝试向SSO会话期间请求对CAS进行身份验证的每个应用程序发送注销消息。虽然这是一个尽力而为的过程，但在许多情况下它运行良好并通过在登录和注销之间创建对称性来提供一致的用户体验。

### 1、CAS退出

根据[CAS协议](https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol.html)，/logout端点负责销毁当前的SSO会话。注销时，可能还需要重定向回服务。这是通过service参数指定重定向链接来控制的。

默认情况下，重定向行为处于关闭状态，并通过以下设置激活cas.properties：

|  |  |
| --- | --- |
| 1  2 | # Specify whether CAS should redirect to the specified service parameter on /logout requests  # cas.logout.followServiceRedirects=false |

指定的URL必须在CAS的服务注册表中注册并启用。

### 2、单点注销（SLO）

CAS旨在支持单点注销：这意味着除了自己的SSO会话之外，它还能够使客户端应用程序会话无效。  
每当票证授予票证明确到期时，将启动注销协议。不支持注销协议的客户端可能会在其访问日志中发现看似无法执行任何操作的额外请求。

**用法警告！**

单一注销默认情况下处于启用状态。

当CAS会话结束时，它会通知每个服务SSO会话不再有效，并且依赖方需要使其自己的会话无效。

这可以通过两种方式实现：

1. CAS直接向服务发送HTTP POST消息（*反向信道*通信）：这是向服务执行通知的传统方式。
2. CAS使用消息和*RelayState*参数（*前端通道*通信）将服务重定向（HTTP 302）到服务：此功能受SAML SLO启发，如果客户端应用程序由多个服务器组成并使用会话关联，则需要此功能。CAS客户端的预期行为是使应用程序Web会话无效，并使用*RelayState*参数重定向回CAS服务器。

**用法警告！**

此时的前通道SLO仍处于试验阶段。

### 3、SLO请求

完成通知的方式（*后台*或*前台*通道）通过logoutType属性在服务级别配置。LogoutType.BACK\_CHANNEL默认情况下，此值设置为。传递消息或将重定向发送到原始CAS协议票证请求的*服务*参数中显示的URL 。

一个示例SLO消息：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <samlp:LogoutRequest  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"  ID="[RANDOM ID]"  Version="2.0"  IssueInstant="[CURRENT DATE/TIME]">  <saml:NameID>@NOT\_USED@</saml:NameID>  <samlp:SessionIndex>[SESSION IDENTIFIER]</samlp:SessionIndex>  </samlp:LogoutRequest> |

会话标识符是最初向CAS验证时提供给服务的CAS服务票证ID。会话标识符用于将CAS会话与应用会话相关联; 例如，SLO会话标识符映射到servlet会话，该servlet会话随后可以被销毁以终止应用程序会话。

### 4、关闭单点注销

要禁用单点注销，请在cas.properties文件中调整以下设置：

|  |  |
| --- | --- |
| 1  2 | # To turn off all back channel SLO requests set slo.disabled to true  # slo.callbacks.disabled=false |

### 5、单次注销每项服务

使用CAS注册的应用程序可以选择通过[Service Managament](https://apereo.github.io/cas/4.2.x/installation/Service-Management.html)组件单独控制单个注销行为。服务注册表中的每个注册服务都将包含描述如何提交注销请求的配置。此行为通过logoutType属性进行控制，该属性允许指定是应通过后/前通道提交注销请求还是为此应用程序关闭注销请求。

示例配置如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | {  "@class" : "org.jasig.cas.services.RegexRegisteredService",  "serviceId" : "testId",  "name" : "testId",  "id" : 1,  "logoutType" : "BACK\_CHANNEL"  } |

### 6、注销请求的服务端点

默认情况下，注销请求将提交到原始服务标识。CAS可以选择将此类请求提交给与原始服务ID不同的特定服务端点，当然也可以在每个服务级别进行配置。如果与CAS集成的应用程序不完全使用支持拦截此类请求的CAS客户端，而是为其注销操作公开不同的端点，则此功能非常有用。

要配置特定于服务的端点，请尝试以下示例：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | {  "@class" : "org.jasig.cas.services.RegexRegisteredService",  "serviceId" : "testId",  "name" : "testId",  "id" : 1,  "logoutType" : "BACK\_CHANNEL",  "logoutUrl" : "https://web.application.net/logout",  } |

### 7、异步SLO消息

默认情况下，反向通道注销消息以异步方式发送到端点。要允许同步消息，请在以下位置修改以下设置cas.properties：

|  |  |
| --- | --- |
| 1  2 | # To send callbacks to endpoints synchronously, set this to false  # slo.callbacks.asynchronous=true |

### 8、SSO会议与应用程序会话

为了更好地理解CAS的SSO会话管理以及它如何看待应用程序会话，首先要考虑的一个重要注意事项是：

**CAS不是会话管理员**

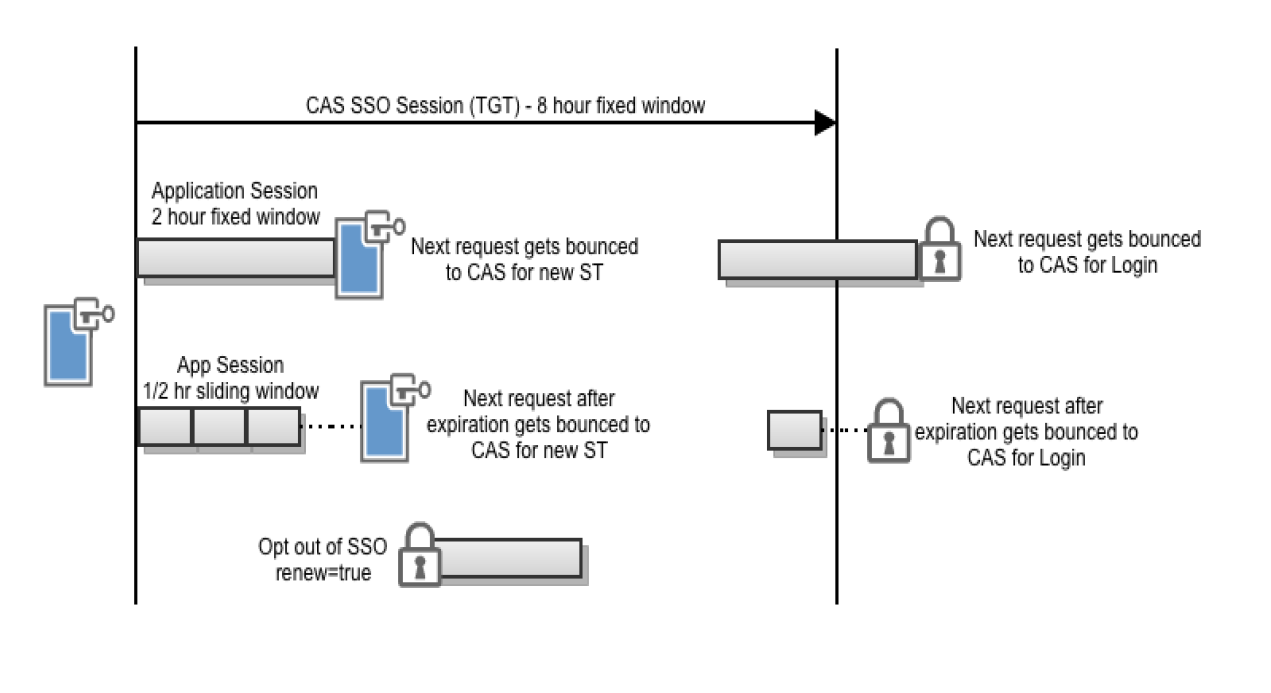
应用程序会话是应用程序的责任。

CAS希望以TicketGrantingTicket一个TGT id 的形式维护和控制SSO会话，TGT id以安全cookie的形式在用户代理和CAS服务器之间共享。

CAS不是应用程序会话管理器，因为应用程序负责维护和控制自己的应用程序会话。验证完成后，CAS通常在应用程序会话方面不合适。因此，应用会话本身的到期策略完全独立于CAS，并且可以在应用会话到期的情况下根据理想的用户体验松散地协调和调整。

如果未激活Single Logout，通常，应用程序可能会公开注销端点以销毁会话，然后将代理重定向到CAS logout端点，以便完全销毁SSO会话。

这是一个简要的图表，演示了各种应用程序会话配置以及与CAS的交互：



## 三、解决shiro无法单点注销问题

退出流程：

1、A应用退出时，请求cas服务端，销毁服务端票据

2、cas服务端向单点注册的B应用发送退出请求

3、B应用收到请求后，退出

### 1、单点登出官方解决方案

在web.xml中配置对应的监听器和过滤即可

<!-- 单点登出监听器 -->

<listener>

<listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-class>

</listener>

<!-- 单点登出 -->

<filter>

<filter-name>CAS Single Sign Out Filter</filter-name>

<filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS Single Sign Out Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

**cas的单点注销默认是关闭的，需要自己打开：/WEB-INF/cas.properties**

# To turn off all back channel SLO requests set slo.disabled to true

# slo.callbacks.disabled=true

slo.callbacks.disabled=false

**SingleSignOutFilter.java源码简单解析：**

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest)servletRequest;

//cas中央认证成功后，回调客户端时链接会携带ticket信息，所以进入第一个判断，绑定ticket和session

if (handler.isTokenRequest(request)) {

handler.recordSession(request);

} else {

//cas服务器退出时，回调客户端链接不会携带ticket信息，但是携带logoutRequest参数，所以进入第二个判断语句

if (handler.isLogoutRequest(request)) {

//通过ticket获取session，然后销毁（只支持容器管理的session）

handler.destroySession(request);

return;

}

this.log.trace("Ignoring URI " + request.getRequestURI());

}

filterChain.doFilter(servletRequest, servletResponse);

}

但是官方解决方案只是对标准的http session进行处理，如果项目中使用了Shiro Session进行管理的话，官方解决方案无法实现单点登出效果。需要自行实现收到CAS服务器发过来的logout请求时，自行处理销毁Shiro Session的逻辑。

### 2、最终解决思路：重写SingleSignOutFilter.java

1 、首先记录CAS Token, 在登录后CAS Server回调时进行。将Token和session id绑定起来（用HashMap）。

@Override

protected boolean preHandle(ServletRequest request, ServletResponse response) throws Exception {

HttpServletRequest req = (HttpServletRequest)request;

//cas中央认证成功后，链接中含有token参数，记录token和sessionID

if (HANDLER.isTokenRequest((HttpServletRequest)req)) {

HANDLER.recordSession(req);

return true;

//cas服务器退出后，链接中含有logoutRequest参数，在之前记录的session中设置logoutRequest参数为true，因为Subject是和线程是绑定的，所以无法获取登录的Subject直接logout

} else if (HANDLER.isLogoutRequest(req)) {

HANDLER.invalidateSession(req,sessionManager);

// Do not continue up filter chain

return false;

} else {

log.trace("Ignoring URI " + req.getRequestURI());

}

Subject subject = SecurityUtils.getSubject();

Session session = subject.getSession(false);

//用户访问时，判断session中是否做了退出标记，有则subject.logout退出

if (session!=null&&session.getAttribute(HANDLER.getLogoutParameterName())!=null) {

try {

subject.logout();

} catch (SessionException ise) {

log.debug("Encountered session exception during logout. This can generally safely be ignored.", ise);

}

}

return true;

}

**在spring-shiro.xml中配置：重写后的不在web.xml中配置**

<!-- 单点退出：拦截回调地址；

登陆成功后绑定ticket和session；

单点退出时在session中设置退出标记；

用户访问资源时有退出标记则调subject.logout() -->

<bean id="casLogoutFilter" class="com.hys.login.manager.CasLogoutFilter">

<property name="sessionManager" ref="sessionManager"/>

</bean>

<!-- shiro的web过滤器,和web.xml中的filtername同名 -->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<property name="securityManager" ref="securityManager"/>

<!-- https://localhost:8443/login：cas服务器端登陆地址；

service=https://localhost:9443/cas：验证成功后的回调地址 -->

<property name="loginUrl" value="${cas.server}/cas-server/login?service=${cas.client}/cas-client/cas"/>

<!-- 登陆成功后返回的地址-->

<property name="successUrl" value="/"/>

<!-- 没有授权返回地址 -->

<property name="unauthorizedUrl" value="/" />

<!-- 自定义拦截器 -->

<property name="filters">

<util:map>

<entry key="authc" value-ref="formAuthenticationFilter"/>

<entry key="cas" value-ref="casFilter"/>

<entry key="logout" value-ref="logoutFilter"/>

<entry key="casLogout" value-ref="casLogoutFilter"/>

</util:map>

</property>

<!-- 定义url和filter的关系：格式 资源 = 过滤器 -->

<property name="filterChainDefinitions">

<value>

<!-- 匿名访问的资源 -->

/anonymous/\*\* = anon

<!-- 登陆地址 -->

/login = authc

<!-- 拦截回调-->

/cas = casLogout,cas

<!-- 设置退出地址 -->

/logout = logout

<!-- user：需要身份认证或RememberMe登录的都可以访问 -->

/\*\* = casLogout,user

</value>

</property>

</bean>

这个方法的解决思路是：   
1 首先记录CAS Token, 在登录后CAS Server回调时进行。将Token和Shiro session id绑定起来（用HashMap）。

2 然后收到CAS Server发过来的登出post请求时，根据token取得Shiro session id，将session id做一个清理标记。   
这时候不能直接进行登出处理，因为拥有logout方法的Shiro Subject对象是和线程绑定的，所以这里不能获取到正确的Subject对象。

3 用户再次使用网站时，这时候检查用户的Shiro session id是不是已被标记为清理。   
如果已被标记，此时掉用Subject对象的logout方法，进行登出处理。

### 3、源码



## 四．安全证书配置

### 1、cas-server服务端

本教程由于是演示所以用JDK自带的keytool工具生成证书；如果以后在产品环境中使用肯定要去证书提供商购买，证书认证一般都是由 VeriSign认证，中文官方网站： http://www.verisign.com/cn/。

**1>．打开cmd命令窗口（管理员身份打开），生成证书**

keytool -genkey -alias ssodemo -keyalg RSA -keysize 1024 -keypass 123456 -validity 365 -keystore c:\wsriakey -storepass 123456

【说明】：-alias后面的别名可以自定义；-validity指定证书有效期；-keypass指定证书密钥库的密码；-keystore指定证书的位置，这里指定放在c盘根目录，密钥库名称可以自定义，这里是wsriakey；-storepass和前面-keypass密码相同，否则下面tomcat 配置https 会访问失败。

见下图：



【注意】：第一个让你输入的“您的名字与姓氏是什么”，请必须输入在C:\Windows\System32\drivers\etc\hosts文件中加入的服务端的域名。

我这里也就是sso.server.com，为何这么做？

首先cas只能通过域名来访问，不能通过ip访问，同时上方是生成证书，所以要求比较严格，如果不这么做的话，既是最终按照教程配置完成，cas也可以正常访问,访问一个客户端应用也能进入cas验证首页，但是，当输入信息正确后，cas在回调转入你想访问的客户端应用的时候，会出现No subject alternative names present错误异常信息，这个错误也就是在上面输入的第一个问题答案不是域名导致、或者与hosts文件配置的不一致导致。

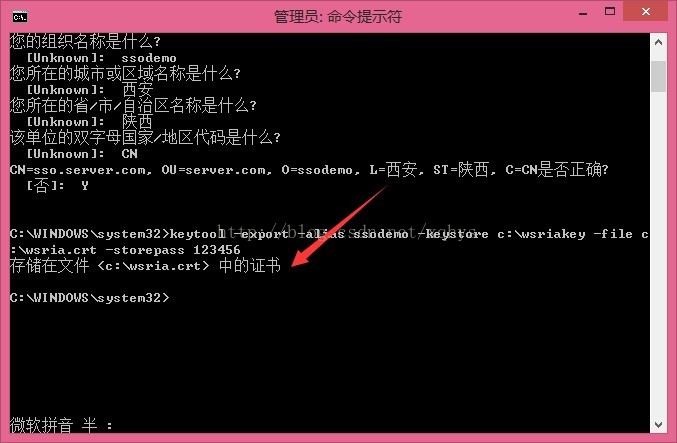
**2>．导出证书：**

在cmd窗口继续输入以下命令，导出证书:

Keytool -export -alias ssodemo -keystore c:\wsriakey -file c:\wsria.crt -storepass 123456

【说明】：-alias后面的名称要与生成证书的命令里面的alias的名称一致； –keystore指定证书存放的位置，这里我放在C盘根目录，同时证书名称要与”生成证书”对应的命令里的keystore名称一致，这里是wsriakey；-file指定导出证书的路径，我也指定在c盘根目录；-storepass的证书密码要与上面输入的密码一致。

如下图：



我们再看看c盘下面是否生成crt文件:

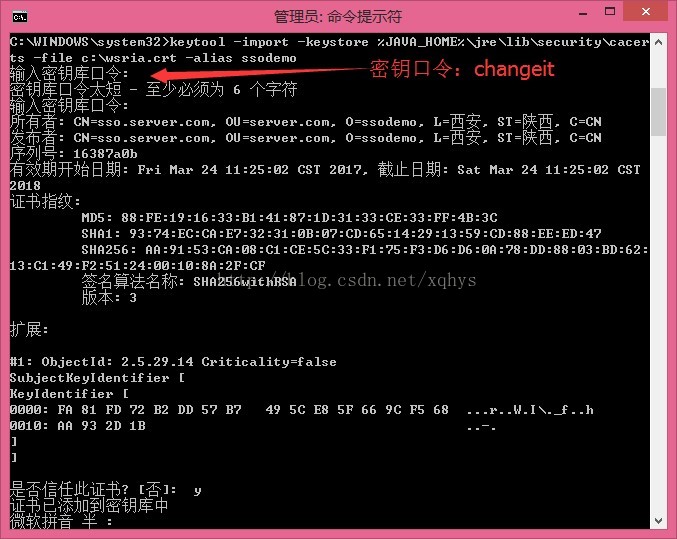
https://img-blog.csdn.net/20170325021035824?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQveHFoeXM=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

**3>．导入证书：**

keytool -import -keystore %JAVA\_HOME%\jre\lib\security\cacerts -file c:\wsria.crt -alias ssodemo

【说明】：命令中指定了JAVA\_HOME，意思是将证书导入到客户端证书库，也就是jdk证书库中，因为客户端应用运行在本地，需要jdk的支持。-file指定证书的位置，也就是上一步导出证书的位置，即c:\wsria.crt。

回车之后，会让你输入密钥库口令，注意，这里的密码必须要输入changeit，不能输入上面指定的密码123456，切记，否则导入客户端证书会有问题，如果是多台机器演示，需要在每一台客户端导入该证书，步骤都是一样的。当看到提示“是否信任此证书”，输入y回车即可，见下图：(说明，命令中的-alias后面的别名可以自定义,如果出现”证书未导入，别名<\*\*\*>已经存在”的错误，该意思是说客户端的密钥库中已经存在该别名证书了，重新指定其他别名即可)



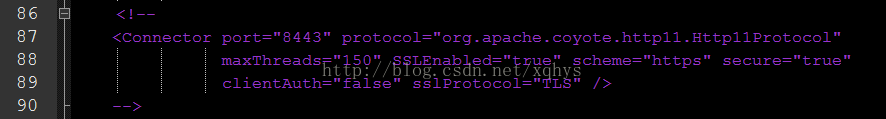
至此，CAS所需的证书环境，已经配置好。

  下面，开始我们的CAS服务、Tomcat、以及多客户端的配置及测试访问。

**4>．配置HTTPS**

由于CAS是基于HTTPS协议，所以需要配置服务端的tomcat，使之支持SSL安全协议访问。当然也可以设置cas支持http访问，这个之后再补充讨论，下面先配置https协议。

解压apache-tomcat-7.0.76-windows-x64，我本地路径为D:\gongju\apache-tomcat-7.0.76，编辑D:\gongju\apache-tomcat-7.0.76\conf\server.xml，找到下面片段：



去掉注释，修改成：

<Connectorport="8443" protocol="org.apache.coyote.http11.Http11Protocol"

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

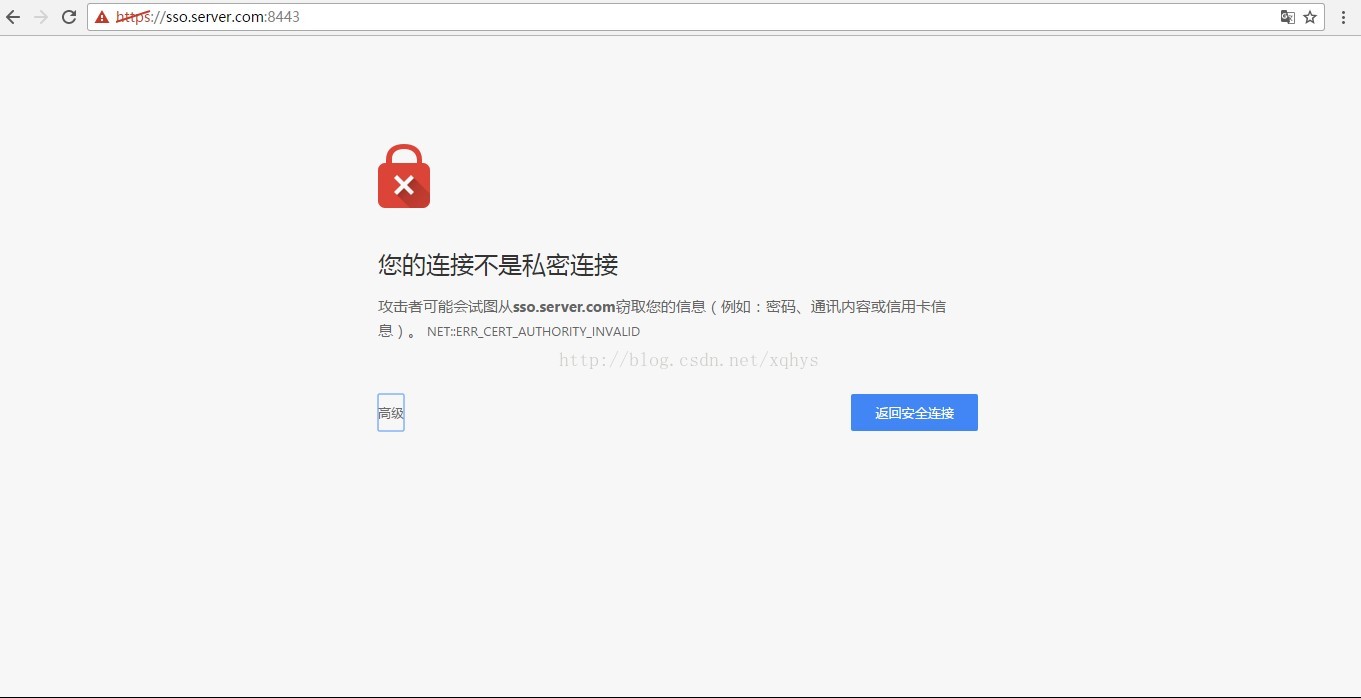
keystoreFile="C:/wsriakey" keystorePass="123456"

clientAuth="false" sslProtocol="TLS"/>

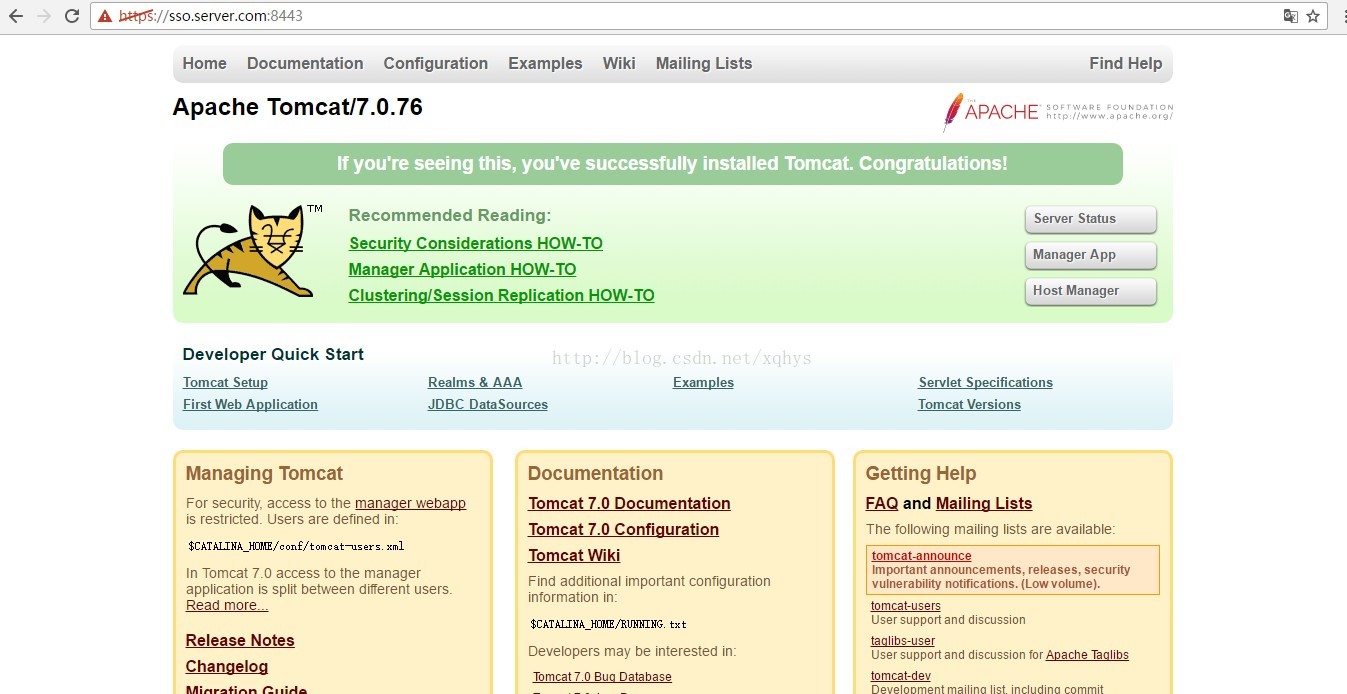
其中，keystoreFile就是创建证书的路径，keystorePass就是创建证书的密码

**5>．验证HTTPS配置**

其他按照默认配置不作修改，双击D:\gongju\apache-tomcat-7.0.76\bin\startup.bat启动tomcat 验证https是否配置成功，我本地使用google览器访问，在地址栏输入https://sso.server.com:8443/出现下面画面，其实这就表明cas服务端tomcat的https协议配置没问题了。



点击【高级】,点击出现的【继续前往sso.server.com（不安全）】，将会出现下面tomcat默认首页，tomcat已经支持https协议访问了，一切ok!



### 2、cas-client客户端

根据cas-server服务端生成的安全证书，在客户端导出证书，然后导入证书。（就是服务端的后两步）