单点登陆平台构建

目录

[1 版本 1](#_Toc10107352)

[2 参考资料 1](#_Toc10107353)

[3 CAS实现SSO原理 2](#_Toc10107354)

[3.1 关于sso 2](#_Toc10107355)

[3.1.1 什么是sso 2](#_Toc10107356)

[3.1.2 sso体系中的角色 2](#_Toc10107357)

[3.1.3 sso实现模式的原则 2](#_Toc10107358)

[3.1.4 sso主要的实现方式 2](#_Toc10107359)

[3.2 关于cas 2](#_Toc10107360)

[3.2.1 cas基本原理 3](#_Toc10107361)

[3.2.2 cas代理模式原理 8](#_Toc10107362)

[4 搭建cas认证中心服务 8](#_Toc10107363)

[4.1 下载用于构建cas服务端war包的项目 8](#_Toc10107364)

[4.2 搭建 8](#_Toc10107365)

[4.2.1 导入解压的项目 8](#_Toc10107366)

[4.2.2 修改配置文件 8](#_Toc10107367)

[4.2.3 配置数据库 9](#_Toc10107368)

[4.2.4 自建CA证书及配置https 10](#_Toc10107369)

[4.3 打包部署 11](#_Toc10107370)

[5 构建cas-client（client为springboot） 11](#_Toc10107371)

[6 构建cas – client（client为springsecurity） 12](#_Toc10107372)

[6.1 1. 12](#_Toc10107373)

# 版本

|  |  |  |
| --- | --- | --- |
| jdk | 1.8及以上 |  |
| cas | 4.2 |  |
| mysql | 6.0（要求jdk为1.8） |  |
|  |  |  |
|  |  |  |

# 参考资料

cassso原理 <https://www.cnblogs.com/gy19920604/p/6029210.html>

cas spring客户端集成<https://blog.csdn.net/yelllowcong/article/details/79241777>

# CAS实现SSO原理

cas，即中央认证服务，是耶鲁大学发起的企业级开源项目，旨在为web应用系统提供一种可靠的单点登陆解决方法。它属于WEB SSO。

## 关于sso

### 什么是sso

用户只需要登陆一次，就可以访问所有相互信任的应用系统。

### sso体系中的角色

1. user 多个
2. web应用 多个
3. sso认证中心 一个

### sso实现模式的原则

1. 所有认证登录都在认证中心进行。
2. sso认证中心通过一些方法来告诉web应用当前访问用户究竟是不是已经通过认证的用户。
3. web应用必须信任认证中心。

### sso主要的实现方式

TODO;

## 关于cas

核心概念：

TGC: casserver在浏览器设置的cookie，用于验证此用户。

TGT : casserver在自己服务端保存的ticket。

ST : casserver用TGT生成的一次性票据。

CAS 的 SSO 实现方式可简化理解为： 1 个 Cookie 和 N 个 Session 。 CAS Server 创建 cookie，在所有应用认证时使用，各应用通过创建各自的 Session 来标识用户是否已登录。

用 户在一个应用验证通过后，以后用户在同一浏览器里访问此应用时，客户端应用中的过滤器会在 session 里读取到用户信息，所以就不会去 CAS Server 认证。如果在此浏览器里访问别的 web 应用时，客户端应用中的过滤器在 session 里读取不到用户信息，就会去 CAS Server 的 login 接口认证，但这时CAS Server 会读取到浏览器传来的 cookie （ TGC ），所以 CAS Server 不会要求用户去登录页面登录，只是会根据 service 参数生成一个 Ticket ，然后再和 web 应用做一个验 证 ticket 的交互而已。

### cas基本原理

cas交互最好是基于ssl的。

#### cas结构体系

cas分为两部分：

casserver CAS Server负责完成对用户的认证工作, 需要独立部署, CAS Server 会处理用户名 / 密码等凭证 (Credentials)

casclient 负责处理对客户端受保护资源的访问请求，需要对请求方进行身份认证时，重定向到CAS Server进行认证。（原则上，客户端应用不再接受任何的用户名密码等 Credentials）。

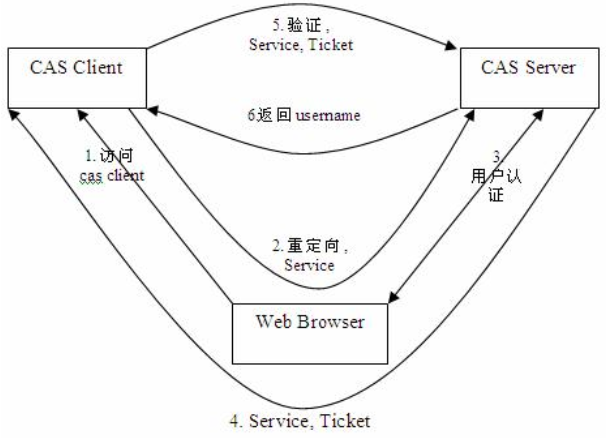
**CAS Client 与受保护的客户端应用部署在一起，以 Filter 方式保护受保护的资源。**

#### cas原理和协议

核心思想：

tgc：是存储在浏览器cookie中的，在任意一个应用下登陆，所有应用就都可以用这个cookie认证了。所以实现了单点登陆。

ST: 即ticket。CAS中casclient的认证都是通过它进行的。



##### step1访问服务：

用户通过浏览器访问被保护的应用，casclient与受保护的应用部署在一起，以filter方式保护受保护的资源，当请求过来时，查看是否已经有session，如果没有，casclient会分析http请求中是否包含ticket（service ticket 即：ST），如果还是没有，说明是没认证过的，于是casclient会重定向请求到sso认证中心。

##### step2 casclient重定向到认证中心：

casclient携带着service（service指目的资源地址）将请求重定向到sso认证中心

##### step3 sso认证中心认证：

1，如果认证通过，为客户端浏览器设置一个**Ticket Granted Cookie（TGC），并在casserver即认证中心生成一个ticketGrantedTicket（TGT）,可以认为它是账号密码生成的一张票，存在认证中心。**凭借TGC ， User 可以免去输入密码以获取访问其它服务的 Service Ticket

2，如果认证通过，sso认证中心生成ticket（**service ticket 即：ST**）并缓存，以待将来验证。

ST是由TGT生成的一次性票据，用于验证，只能用一次。相当于是server发给client的一个一次性票，然后Client拿着这个票再来找Server验证，看看是不是Server签发的。

##### step4 sso认证中心产生并发放票据：

sso认证中心产生唯一票据并把票据ST发放给casclient

##### step5 认证票据：

casclient在拿到Service和新产生的 Ticket（ST）后，拿着service和ST到sso认证中心认证。如果认证通过，创建session。

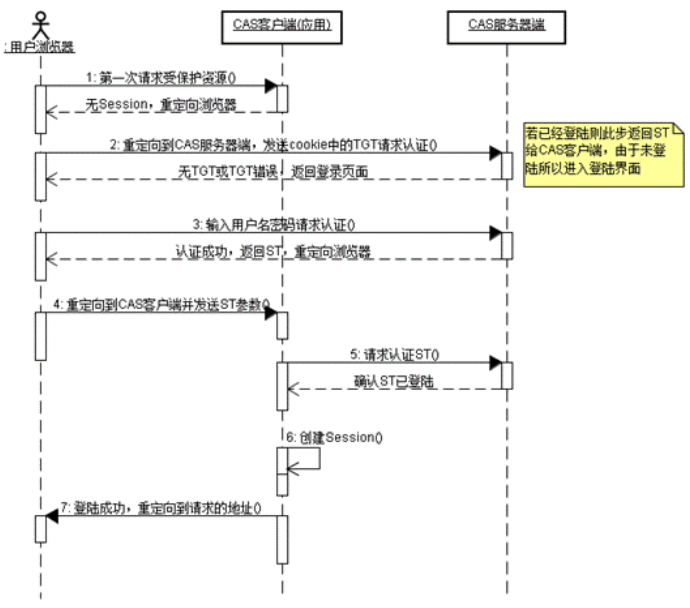
##### step6 返回user信息。

在如上六个步骤中，cas是这样实现单点登陆sso的：当用户访问另外一个应用时，再次被重定向到casserver时，casserver主动获取TGC cookie，然后做如下事情：

1. 如果持有tgc，且还没失效，那么走step4
2. 如果没有tgc，或已经失效，则走step3

sso认证中心返回user信息并重定向到service指定的目的资源地址。

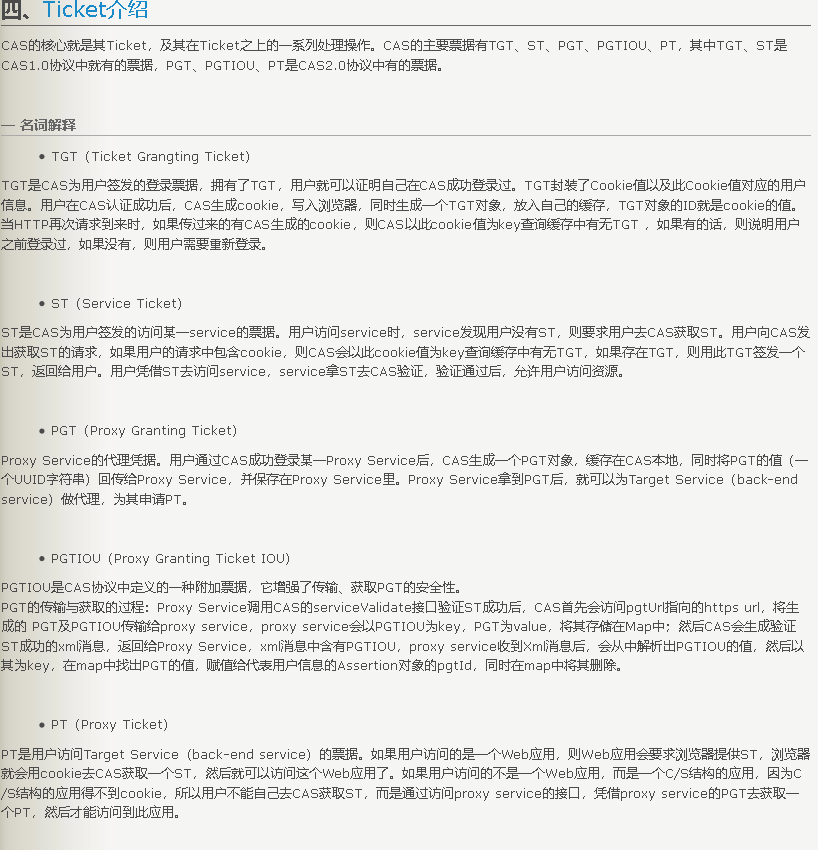
时序图：



#### 登出

CAS server接受请求后，会检测用户的TCG Cookie，把对应的session清除，同时会找到所有通过该TGC sso登录的应用服务器URL提交请求,所有收到请求的应用服务器application会解析这个参数，取得sessionId，根据这个Id取得session后，把session删除。  
这样就实现单点登出的功能

#### 名词解释



### cas代理模式原理

TODO;

<https://www.cnblogs.com/gy19920604/p/6029210.html> 3.2.2

# 搭建cas认证中心服务

注意：mysql如果数据库用6需要用jdk8

搭建后的实例

<https://github.com/likemytea/cas-server-v42.git>

<https://github.com/likemytea/cas-overlay.git> <--废弃

## 下载用于构建cas服务端war包的项目

进入<https://github.com/apereo/cas-overlay-template>，默认的是master版本，是用gradle构建的，4.2或5.3的版本是用maven构建的，我们选择它们里边的一个。然后下载。

## 搭建

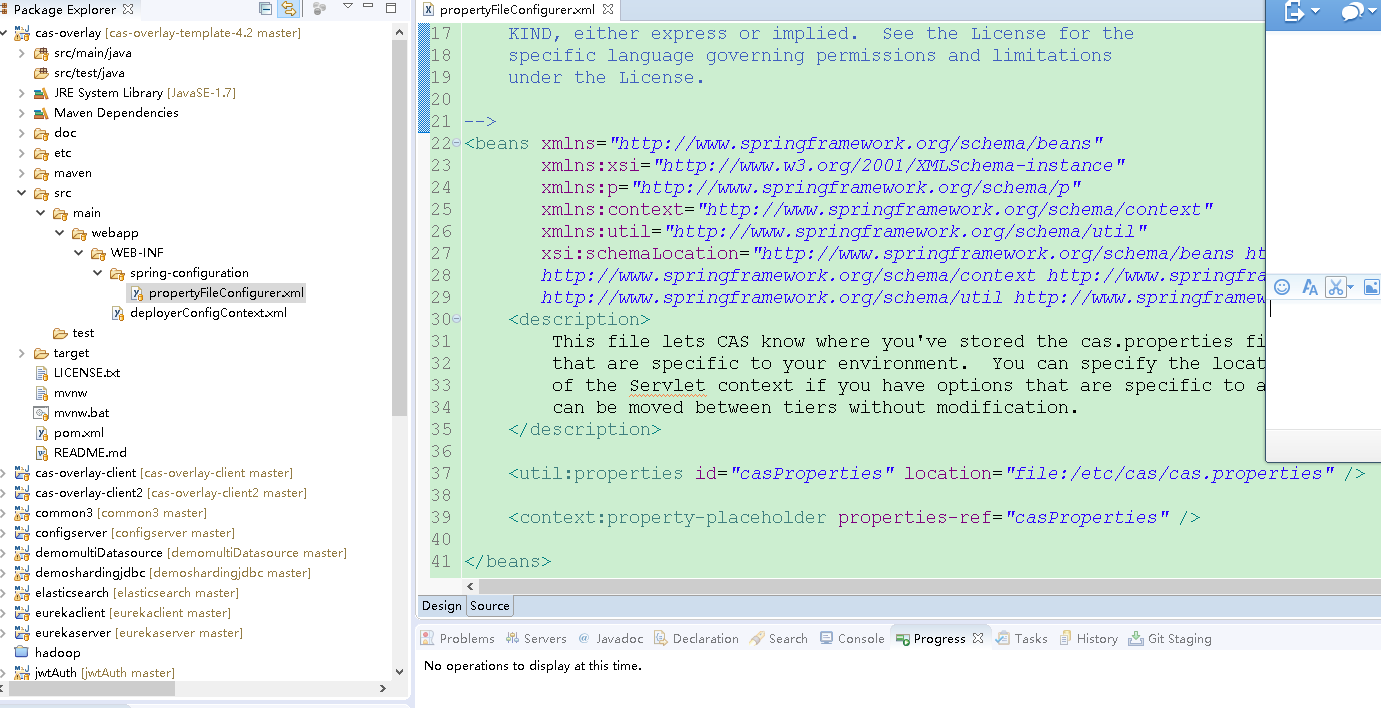
### 导入解压的项目

略

### 修改配置文件

#### propertyFileConfigurer.xml

在较新的版本中，cas的很多配置都放在了cas.properties里，这个文件可以放在服务器的任何位置，但是需要修改propertyFileConfigurer.xml这个配置文件，告诉cas系统cas.properties需要去哪个位置加载



#### cas.properties

cas.jdbc.authn.query.encode.salt=

cas.jdbc.authn.query.sql=SELECT password FROM sys\_user WHERE username=?

### 配置数据库

#### 1.添加jar包

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.29</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>6.0.6</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.jasig.cas</groupId>

<artifactId>cas-server-support-jdbc</artifactId>

<version>${cas.version}</version>

</dependency>

**需要注意的是：mysql connector6.0以上版本的driveClassName用的是com.mysql.cj.jdbc.Driver，连接字符串也需要增加serverTimezone=UTC否则会报错**

#### 2.修改用户认证的类

如下图，打包后获取xml后，再进行如下四步：



#### 重写的handler类

使用自定义类实现从数据库获取账户信息校验，我的这个类是重写了源码里的queryDatabaseAuthenticationHandler

### 自建CA证书及配置https

http和https的区别是什么：核心意义有两条1.https是加密传输 2.使用权威机构颁发的证书，证明我自己是正规的网站。

#### 在casserver上创建证书

**创建证书**

keytool -genkeypair -keyalg RSA -alias liuxingsso2019 -dname "cn=sso.chenxing.com" -keystore /root/liuxing/keystore/liuxingsso201905.keystore -storepass liuxing

上面的命令：指定使用RSA算法，生成别名为liuxingsso2019的证书，口令为liuxing，证书的DN为 "cn=sso.chenxing.com"，注意：域名DN不能输入ip

**查看证书库里边所有证书**

keytool -list -keystore liuxingsso201905.keystore

**查看某个证书**

keytool -list -keystore liuxingsso201905.keystore -alias liuxingsso2019

**导出证书**

keytool -export -file /root/liuxing/keystore/givejdk.crt -alias liuxingsso2019 -keystore /root/liuxing/keystore/liuxingsso201905.keystore

在指定的路径生成crt后，就可以把它分发给应用的jdk使用了。后续讲解在客户端的jvm怎么导入证书。

**在casclient所在的应用服务环境中导入证书**

看后面章节

#### 在casserver的servlet容器（tomcat）中导入密钥库。

修改server.xml文件

**第一步**：找到AprLifecycleListener 这个监听类的配置并将SSLEngine修改为on

<Listenerclass    Name="org.apache.catalina.core.AprLifecycleListener"      SSLEngine="on" />

**第二步**：关闭AJP/1.3协议(也就是直接注释掉)

<!-- <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" /> -->

**第三步**：修改web监听端口得相关配置如下：

<Connector port="443" protocol="HTTP/1.1"

connectionTimeout="20000" SSLEnabled="true" maxThreads="150" scheme="https" secure="true" clientAuth="false" sslProtocol="TLS" keystoreFile

="/root/liuxing/keystore/liuxingsso201905.keystore" keystorePass="liuxing" />

#### 在casclient所在的应用服务环境中导入证书

首先：这个casclient必须是https的，因为如果是http的，后面调https的认证中心，会有各种异常出现。

把上面章节中导出的givejdk.crt证书复制到应用服务器的jvm的路径（本机tomcat上是C:\Program Files\Java\jdk1.8.0\_25\jre\lib\security）下。然后执行命令：

keytool -import -alias liuxingsso2019 -keystore cacerts -file givejdk.crt

## 打包部署

mvn clean package –DskipTests

至此其实就可以打包部署了，访问方式：<https://172.16.176.210/cas/login>

# 构建cas-client（client为springboot）

demo : https://github.com/likemytea/cas-overlay-client

<https://blog.csdn.net/yelllowcong/article/details/79241777>

1. 创建过滤器类：用于匹配不需要登陆就能访问的资源
2. 创建首页控制器（及jsp）和登出控制器
3. 启动类中添加各种filter
4. 配置jar包
5. 配置https，及导入CA证书

# 构建cas – client（client为springsecurity）

cas-client为springsecurity，cas-server为cas apereo server

demo: https://github.com/likemytea/manage-sys

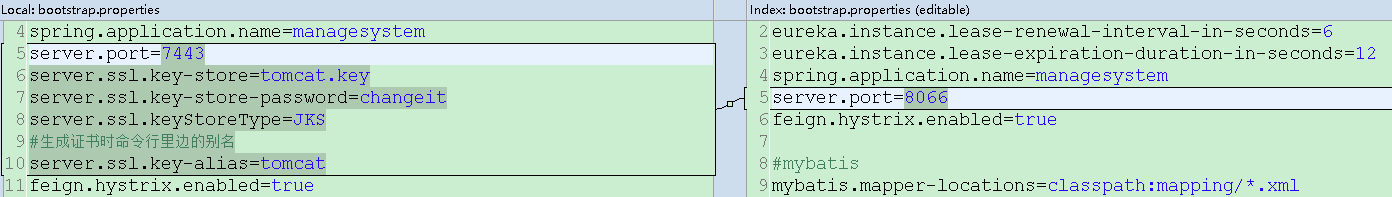
## 项目改为https

### 使用jdk生成证书

keytool -genkeypair -alias tomcat -keyalg RSA -keystore E:\tomcat.key

然后将生成的文件放到项目的根目录下。

### 修改bootstrap.properties



## casclient替换springsecurity

TODO;