目录

[一．CAS是什么 1](#_Toc533534457)

[1.CAS—单点登录框架 1](#_Toc533534458)

[2.核心票据 2](#_Toc533534459)

[二．CAS简单应用 2](#_Toc533534460)

[1.CAS客户端： 2](#_Toc533534461)

[2.CAS服务端： 3](#_Toc533534462)

[三．自定义登录界面 5](#_Toc533534463)

[1.为不同的服务（应用系统）指定不同的主题 5](#_Toc533534464)

[2.自定义登录界面的修改 5](#_Toc533534465)

[3.关于国际化 6](#_Toc533534466)

[四.单点登录流程（源代码剖析CAS-4.1.7） 6](#_Toc533534467)

[1．用户第一次发送请求---->CAS客户端转发 6](#_Toc533534468)

[2．CAS客户端转发---->CAS服务器登录页 7](#_Toc533534469)

[3．用户登录----->CAS服务器认证 10](#_Toc533534470)

[4．CAS服务器认证---->重定向目标资源 11](#_Toc533534471)

## 一．CAS是什么

### 1.CAS—单点登录框架

单点登录（Single Sign On,即SSO）使得在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。

通俗理解为一个应用登录了，其他被授权的应用就不需要再登录。

应用例子为：淘宝登录了天猫不用再登录。

应用场景：

分布式多系统用户集中管理

用户权限集中管理

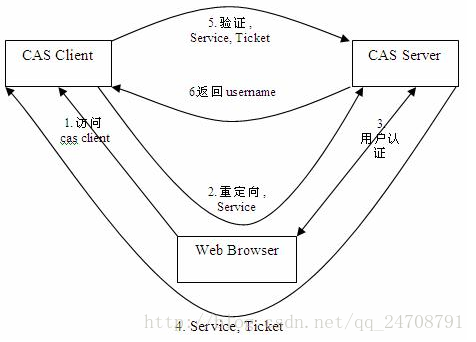
多因素认证（如微信pc端登录手机确认）

从结构上看，CAS 包含两个部分：CAS Server 和 CAS Client。

CAS Server 需要独立部署，会为用户签发两个重要的票据：登录票据（TGT）和服务票据（ST），主要负责对用户的认证工作

CAS Client 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。准确地来说，它以Filter 方式保护受保护的资源。对于访问受保护资源的每个 Web 请求，CAS Client 会分析该请求的 Http 请求中是否包含 ServiceTicket（服务票据，由 CAS Server发出用于标识目标服务）。CAS Client 与受保护的客户端应用部署在一起。

下图是 CAS 最基本的协议过程：



### 2.核心票据

#### TGT

TGT(Ticket Grangting Ticket)是CAS为用户签发的登录票据，拥有了TGT，用户就可以证明自己在CAS成功登录过。TGT封装了Cookie值以及此Cookie值对应的用户信息。用户在CAS认证成功后，生成一个TGT对象，放入自己的缓存（Session）；同时，CAS生成cookie（叫TGC，个人理解，其实就是TGT的SessionId），写入浏览器。TGT对象的ID就是cookie的值，当HTTP再次请求到来时，如果传过来的有CAS生成的cookie，则CAS以此cookie值（SessionId）为key查询缓存中有无TGT（Session），如果有的话，则说明用户之前登录过，如果没有，则用户需要重新登录

#### TGC

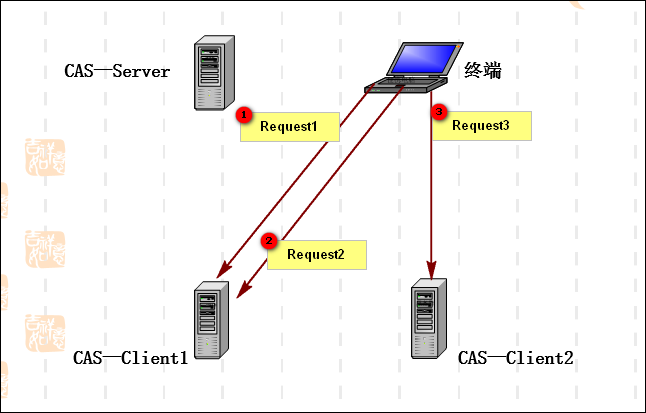
CAS-Server生成TGT放入自己的Session中，而TGC(Ticket-granting cookie)就是这个Session的唯一标识（SessionId），以Cookie形式放到浏览器端，是CAS Server用来明确用户身份的凭证。

#### ST

ST(ServiceTicket)是CAS为用户签发的访问某一服务票据。用户访问service时，service发现用户没有ST，则要求用户去CAS获取ST。用户向CAS发出获取ST的请求，如果用户的请求中包含cookie，则CAS会以此cookie值为key查询缓存中有无TGT，如果存在TGT，则用此TGT签发一个ST，返回给用户。用户凭借ST去访问service，service拿ST去CAS验证，验证通过后，允许用户访问资源。

为了保证ST的安全性：ST 是基于随机生成的，没有规律性。而且，CAS规定 ST 只能存活一定的时间，然后 CAS Server 会让它失效。而且，CAS 协议规定ST只能使用一次，无论 Service Ticket 验证是否成功， CASServer 都会清除服务端缓存中的该 Ticket ，从而可以确保一个 Service Ticket 不被使用两次。

认证过程：



**Request1**

　　　　【第一步】终端第一次访问CAS—Client1，AuthenticationFilter会截获此请求：**1、**首先，检测本地Session没有缓存有用户信息；**2、**然后，检测到请求信息中没有ST；**3、**所以，CAS—Client1将请求重定向到CAS—Server，并传递 Service （也就是要访问的目的资源地址，以便登录成功过后转回该地址），例：【https://cas:8443/cas/login?service=http0%3A8081%2F】

　　　　【第二步】终端第一次访问CAS—Server：**1、**CAS—Server检测到请求信息中没有TGC，所以跳转到自己的登录页；**2、**终端输入用户名、密码登录CAS—Server，认证成功后，CAS—Server会生成登录票据—TGT（集成了用户信息与ST），并**随机**生成一个服务票据—ST与CAS会话标识—TGC。TGT实际上就是Session，而TGC就是这标识这个Session存到Cookie中的SessionID；ST即，根据Service生成Ticket。**3、**然后，CAS—Server会将Ticket加在url 后面，然后将请求redirect 回客户web 应用，例如URL为【http://192.168.1.90:8081/web1/?ticket=ST-5-Sx6eyvj7cPPCfn0pMZ】

　　　　【第三步】这时，终端携带ticket再次请求CAS—Client1：1、这时客户端的AuthenticationFilter看到ticket 参数后，会跳过，由其后面的TicketValidationFilter 处理；2、TicketValidationFilter 会利用httpclient工具访问cas 服务的/serviceValidate 接口, 将ticket 、service 都传到此接口，由此接口验证ticket 的有效性，即向CAS—Server验证ST的有效性。3、TicketValidationFilter如果得到验证成功的消息，就会把用户信息写入web 应用的session里。至此为止，SSO 会话就建立起来了。

**Request2**

　　　　上面说了SSO 会话已经建立起来了，这时用户在同一浏览器里第二次访问此web 应用（CAS—Client1）时，AuthenticationFilter会在session 里读取到用户信息，这就代表用户已成功登录，所以就不会去CAS 认证了。

**Request3**

　　　　【第一步】与Request1是完全一样的，如下：终端第一次访问CAS—Client2，AuthenticationFilter会截获此请求：**1、**首先，检测本地Session没有缓存有用户信息；**2、**然后，检测到请求信息中没有ST；**3、**所以，CAS—Client1将请求重定向到CAS—Server，并传递 Service （也就是要访问的目的资源地址，以便登录成功过后转回该地址），例：【https://cas:8443/cas/login?service=http0%3A8081%2F】

　　　　【第二步】然后，终端第二次访问CAS—Server：此时，Request中会带有上次生成的TGC，然后根据TGC（SessionID）去查找是否有对应的TGT（Session），如果有，代表此用户已成功登录过，所以此时用户不必再去登录页登录（SSO的体现），而CAS—Server会直接用找到的TGT签发一个ST，然后重定向到CAS—Client2，剩下的如Request1中的【第三步】就完全一样了。

## 二．CAS简单应用

### 1.CAS客户端：

即要使用CAS的应用系统。在其web.xml文件中添加cas过滤器，大略如下：

*<!--该过滤器负责用户的认证工作，必须启用它-->*<**filter**>  
 <**filter-name**>CASFilter</**filter-name**>  
<**filter-class**>org.jasig.cas.client.authentication.AuthenticationFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>casServerLoginUrl</**param-name**>  
 <**param-value**>http://127.0.0.1:38080/cas/login</**param-value**>  
 </**init-param**>  
 <**init-param**>  
 <**param-name**>serverName</**param-name**>  
 <**param-value**>http://127.0.0.1:8082/</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>CASFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

其中，casServerLoginUrl对应着CAS服务端

还有负责对Ticket进行校验的过滤器：

org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter

具体见demo\_service2.

### 2.CAS服务端：

CAS服务端其实就是一个war包。发挥主要作用的就是cas-server-webapp，将其生成的war文件放在tomcat下启动即可看到登录页面。

#### 2.1去除https认证

CAS默认使用的是HTTPS协议，如果让CAS使用HTTP协议，需要修改配置

在WEB-INF/deployerConfigContext.xml文件中：

<**bean id="proxyAuthenticationHandler" class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"  
 p:httpClient-ref="supportsTrustStoreSslSocketFactoryHttpClient" p:requireSecure="false"**/>

spring-configuration/ticketGrantingTicketCookieGenerator.xml文件中：

<**bean id="ticketGrantingTicketCookieGenerator" class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"  
 c:casCookieValueManager-ref="cookieValueManager"  
 p:cookieSecure="false"  
 p:cookieMaxAge="-1"  
 p:cookieName="TGC"  
 p:cookiePath=""**/>

spring-configuration/warnCookieGenerator.xml文件中：

<**bean id="warnCookieGenerator" class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"  
 p:cookieSecure="false"  
 p:cookieMaxAge="-1"  
 p:cookieName="CASPRIVACY"  
 p:cookiePath=""**/>

services/HTTPSandIMAPS-10000001.json文件中：

**"serviceId"** : **"^(https|imaps|http)://.\*"**,

#### 2.2关于登录名和密码

在CAS Server中默认的登录名/密码为casuser/Mellon

因为在deployerConfigContext.xml中是这样配置的

*<bean id="primaryAuthenticationHandler"  
 class="org.jasig.cas.authentication.AcceptUsersAuthenticationHandler">  
 <property name="users">  
 <map>  
 <entry key="casuser" value="Mellon"/>  
 </map>  
 </property>  
</bean>*

如果要通过数据库来验证用户名、密码，则需要做如下更改：

添加maven依赖，cas-server-support-jdbc

在authenticationManager中更换验证Handle：

*<!--<entry key-ref="primaryAuthenticationHandler" value-ref="primaryPrincipalResolver" />-->*<**entry key-ref="dbAuthHandler" value-ref="primaryPrincipalResolver"**/>

其中dbAuthHandler为：

<**bean id="dbAuthHandler" class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler"**>  
 <**property name="dataSource" ref="dataSource"**/>  
 <**property name="sql" value="select password from tb\_user where user\_name=?"**/>  
 <**property name="passwordEncoder" ref="MD5PasswordEncoder"**/>  
</**bean**>

*<!--数据验证模式 cas默认MD5加密类，返回值：加密后的字符串-->*<**bean id="MD5PasswordEncoder" class="org.jasig.cas.authentication.handler.DefaultPasswordEncoder" c:encodingAlgorithm="MD5"  
 p:characterEncoding="UTF-8"** />

通过看QueryDatabaseAuthenticationHandler.java的源码，可以知道，cas通过passwordEncoder指定的类计算用户输入的密码经过特定编码的字符串，将这个字符串与通过sql找到的password字符串比较

可以实现自己的密码加密类

## 三．自定义登录界面

### 1.为不同的服务（应用系统）指定不同的主题

对于一个很大的网站，不同的子项目下面，可能登录的风格和样式不一样，所以我们需要配置这个service目录，设定每一个子网站对应的请求样式，如果不设定，可以指定默认样式

Services文件夹下指定 需要配置自定义登录的网站模版

Apereo-10000002.json 、HTTPSandIMAPS-10000001.json 这个json配置文件是系统默认的，不要修改它，只能覆盖它； json配置文件的命名必须是主题名称-id.json的这种方式，不然找不到属性配置文件。属性配置文件为主题名称.properties

 json配置文件中的参数说明



具体可以查看demo中demo1-1000003.json、demo1.properties

注：在jsp文件中，引用css文件地址时是指定的${ standard.custom.css.file }，所以在配置文件中是指定standard.custom.css.file的值。

要注意参数evaluationOrder的值，代表了匹配顺序

### 2.自定义登录界面的修改

在文件login-webflow.xml中 ,<view-state id="viewLoginForm" view="casLoginView" model="credential">指定了服务器的登录页。之所以这里的casLoginView指向了/WEB-INF/view/jsp/default/ui/下的casLoginView.jsp，是因为在cas\_servlet.xml中配置了：

<**bean id="internalViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"  
 p:viewClass="org.springframework.web.servlet.view.JstlView"  
p:prefix="${cas.viewResolver.defaultViewsPathPrefix:/WEB-INF/view/jsp/default/ui/}"  
 p:suffix=".jsp"  
 p:order="3"**/>

复制casLoginView.jsp重命名为casLoginViewRe.jsp，改写casLoginViewRe.jsp。div id=list-language没啥用，可以删掉。<c:if test="${not pageContext.request.secure}">这个标签是https error提示，也可以删掉。在<div class="sidebar-content">之中添加自己的内容就可以了。

改为<view-state id="viewLoginForm" view="casLoginViewRe" model="credential">则指向了casLoginViewRe.jsp

### 3.关于国际化

<%

org.springframework.web.servlet.i18n.CookieLocaleResolver clr=new org.springframework.web.servlet.i18n.CookieLocaleResolver();

clr.setLocale(request,response,Locale.CHINA);  //这里根据不同情况显示不同国际化

%>

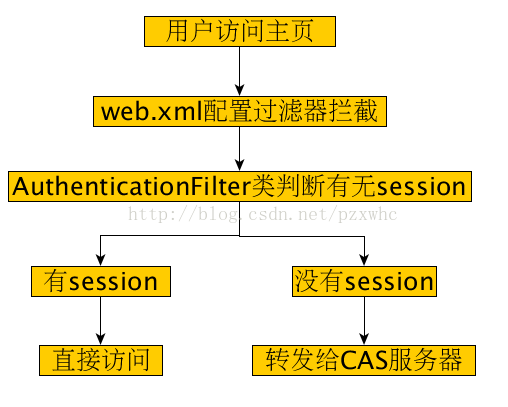
通过该端代码放在jsp页面就可以实现不同jsp登录页面实现不同的国际化提示

其实在cas-servlet.xml中有CookieLocaleResolver类的实例化：<bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver" p:defaultLocale="en" />，这里的改动对于谷歌浏览器有效果，而对于IE没有。

## 四.单点登录流程（源代码剖析CAS-4.1.7）

### 1．用户第一次发送请求---->CAS客户端转发

1.用户首次访问，过滤器拦截；2.过滤器doFilter()具体操作；3.过滤器转发操作；



关于转发给CAS Server的代码如下：

**final** String urlToRedirectTo = CommonUtils.*constructRedirectUrl*(**this**.**casServerLoginUrl**,  
 getProtocol().getServiceParameterName(), modifiedServiceUrl, **this**.**renew**, **this**.**gateway**);  
  
**this**.**authenticationRedirectStrategy**.redirect(request, response, urlToRedirectTo);

### 2．CAS客户端转发---->CAS服务器登录页

请求交给Spring MVC进行处理；Spring MVC把请求交给工作流webflow进行处理；工作流处理请求流程

<**servlet**>  
 <**servlet-name**>cas</**servlet-name**>  
 <**servlet-class**>  
 org.springframework.web.servlet.DispatcherServlet  
 </**servlet-class**>  
 <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>/WEB-INF/cas-servlet.xml, /WEB-INF/cas-servlet-\*.xml</**param-value**>  
 </**init-param**>  
 …………  
</**servlet**>

在文件cas\_servlet.xml中配置了各种操作对应的项，比如登录、登出

*<!--将特定应用程序资源映射到流-->*<**bean id="loginFlowHandlerMapping" class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping"  
 p:flowRegistry-ref="loginFlowRegistry" p:order="2"**>  
 <**property name="interceptors"**>  
 <**array value-type="org.springframework.web.servlet.HandlerInterceptor"**>  
 <**ref bean="localeChangeInterceptor"** />  
 </**array**>  
 </**property**>  
</**bean**>

<!—为login登录请求开启流处理-->

<**bean id="loginHandlerAdapter" class="org.jasig.cas.web.flow.SelectiveFlowHandlerAdapter"  
 p:supportedFlowId="login" p:flowExecutor-ref="loginFlowExecutor" p:flowUrlHandler-ref="loginFlowUrlHandler"** />  
  
<**bean id="loginFlowUrlHandler" class="org.jasig.cas.web.flow.CasDefaultFlowUrlHandler"** />  
   
<**bean name="loginFlowExecutor" class="org.springframework.webflow.executor.FlowExecutorImpl"   
 c:definitionLocator-ref="loginFlowRegistry"  
 c:executionFactory-ref="loginFlowExecutionFactory"  
 c:executionRepository-ref="loginFlowExecutionRepository"** />  
   
<**bean name="loginFlowExecutionFactory" class="org.springframework.webflow.engine.impl.FlowExecutionImplFactory"  
 p:executionKeyFactory-ref="loginFlowExecutionRepository"**/>  
  
<**bean id="loginFlowExecutionRepository" class=" org.jasig.spring.webflow.plugin.ClientFlowExecutionRepository"  
 c:flowExecutionFactory-ref="loginFlowExecutionFactory"  
 c:flowDefinitionLocator-ref="loginFlowRegistry"  
 c:transcoder-ref="loginFlowStateTranscoder"** />  
  
<**webflow:flow-registry id="loginFlowRegistry" flow-builder-services="builder" base-path="/WEB-INF/webflow"**>  
 <**webflow:flow-location-pattern value="/login/\*-webflow.xml"**/>  
</**webflow:flow-registry**>

然后，在给springMVC的请求路径如果是login时，则有springMVC交给webflow处理，即去找login文件夹下的login-webflow.xml。相应的，退出时去找loginout.xml

同时，在cas\_servlet.xml中定义了视图解析工厂：

<**webflow:flow-builder-services id="builder" view-factory-creator="viewFactoryCreator" expression-parser="expressionParser"** />

<**bean id="viewFactoryCreator" class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator"**>  
 <**property name="viewResolvers"**>  
 <**util:list**>  
 <**ref bean="viewResolver"**/>  
 <**ref bean="internalViewResolver"**/>  
 </**util:list**>  
 </**property**>  
</**bean**>

视图解析器将jsp文件映射成视图名称

在工作流处理请求中（login-webflow.xml）：

1.首先判断有无TGT

<**action-state id="ticketGrantingTicketCheck"**>  
 <**evaluate expression="ticketGrantingTicketCheckAction"**/>  
 <**transition on="notExists" to="gatewayRequestCheck"**/>  
 <**transition on="invalid" to="terminateSession"**/>  
 <**transition on="valid" to="hasServiceCheck"**/>  
</**action-state**>

在ticketGrantingTicketCheckAction中判断TGT状态，根据返回值是noExists/invalid/valid去向不同的节点。

例如不存在，则去到gatewayRequestCheck：

<**decision-state id="gatewayRequestCheck"**>  
 <**if test="requestParameters.gateway != '' and requestParameters.gateway != null and flowScope.service != null"  
 then="gatewayServicesManagementCheck" else="serviceAuthorizationCheck"**/>  
</**decision-state**>

在gatewayRequestCheck中判断requestParameters.gateway，首次访问时flowScope.service为false，那么就会去到serviceAuthorizationCheck

<**action-state id="serviceAuthorizationCheck"**>  
 <**evaluate expression="serviceAuthorizationCheck"**/>  
 <**transition to="generateLoginTicket"**/>  
</**action-state**>

不管serviceAuthorizationCheck返回什么，接着会转去generateLoginTicket节点进行处理。

这里看一下serviceAuthorizationCheck

<**bean id="serviceAuthorizationCheck" class="org.jasig.cas.web.flow.ServiceAuthorizationCheck"  
 c:servicesManager-ref="servicesManager"** />

<**bean id="servicesManager" class="org.jasig.cas.services.DefaultServicesManagerImpl"  
 c:serviceRegistryDao-ref="serviceRegistryDao"**/>

<**beanid="serviceRegistryDao"class="org.jasig.cas.services.JsonServiceRegistryDao"c:configDirectory="${service.registry.config.location:classpath:services}"** />

从JsonServiceRegistryDao里的代码可以看到这里实现的是为不同的服务定义不同的主题。

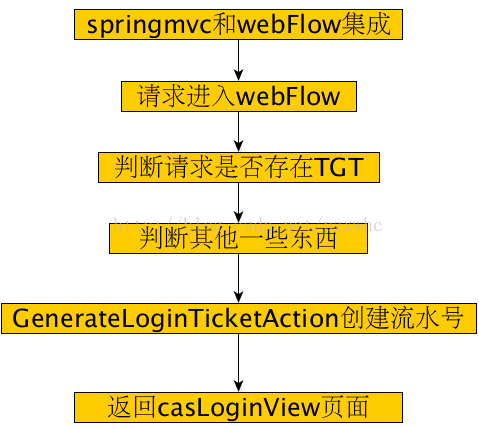
在generateLoginTicket中，执行genarateLoginTicketAction.generate方法，生成一个流水号(login ticket)放入flowscope中，返回值为generated转到viewLoginForm

<**action-state id="generateLoginTicket"**>  
 <**evaluate expression="generateLoginTicketAction.generate(flowRequestContext)"**/>  
 <**transition on="generated" to="viewLoginForm"**/>  
</**action-state**>

在viewLoginForm中，属性view绑定了相应的jsp界面

<**view-state id="viewLoginForm" view="casLoginViewRe" model="credential"**>  
 <**binder**>  
 <**binding property="username" required="true"**/>  
 <**binding property="password" required="true"**/>  
 </**binder**>  
 <**on-entry**>  
 <**set name="viewScope.commandName" value="'credential'"**/></**on-entry**>  
 <**transition on="submit" bind="true" validate="true" to="realSubmit"**/>  
</**view-state**>

整个过程如图：



### 3．用户登录----->CAS服务器认证

realSubmit节点会去执行authenticationViaFormAction.submit方法，对用户名密码进行认证，根据输入的credentials生成TGT，再根据不同的返回状态去不同的节点

<**action-state id="realSubmit"**>  
 <**evaluate expression="authenticationViaFormAction.submit(flowRequestContext, flowScope.credential, messageContext)"**/>  
 <**transition on="warn" to="warn"**/><**transition on="success" to="sendTicketGrantingTicket"**/>  
 <**transition on="successWithWarnings" to="showMessages"**/>  
 <**transition on="authenticationFailure" to="handleAuthenticationFailure"**/>  
 <**transition on="error" to="generateLoginTicket"**/>  
</**action-state**>

在submit方法中的createTicketGrantingTicket下的this.centralAuthenticationService.createTicketGrantingTicket()来完成认证

AuthenticationViaFormAction的submit要做的就是判断FlowScope和request中的loginTicket是否相同。如果不同跳转到错误页面，如果相同，则根据用户凭证生成TGT（登录成功票据），并放到requestScope作用域中，同时把TGT缓存到服务器的cache<ticketId,TGT>中。登录流程流转到下个state（sendTicketGrantingTicket）。

具体实现类是CentralAuthenticationServiceImpl中的createTicketGrantingTicket方法。具体看一下这个方法，其中主要如下：

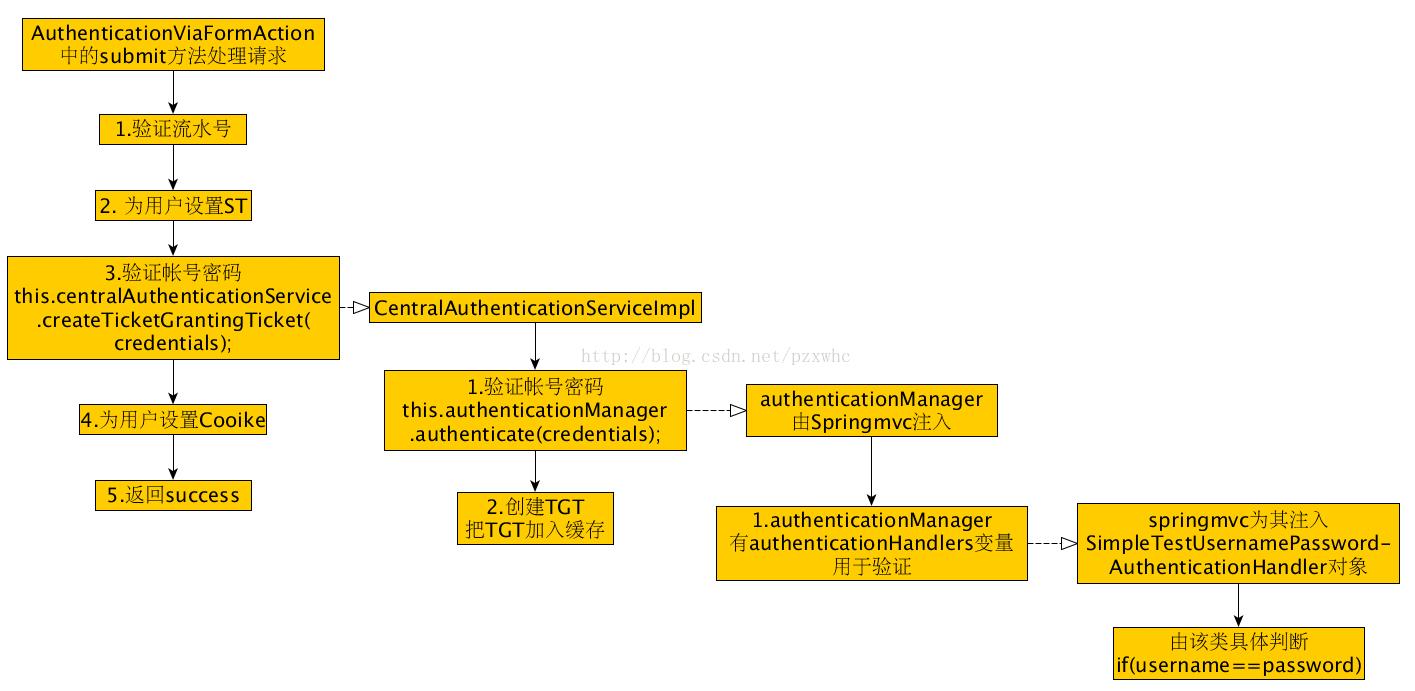
**final** Authentication authentication = **this**.**authenticationManager**.authenticate(credentials);  
**final** TicketGrantingTicket ticketGrantingTicket = **new** TicketGrantingTicketImpl(  
 **this**.**ticketGrantingTicketUniqueTicketIdGenerator** .getNewTicketId(TicketGrantingTicket.***PREFIX***),  
 authentication, **this**.**ticketGrantingTicketExpirationPolicy**);  
**this**.**ticketRegistry**.addTicket(ticketGrantingTicket);  
**return** ticketGrantingTicket;

这一部分包括了验证账号密码，创建TGT并且缓存TGT,以便在后面如果用户在其他子系统登录的时候根据这个缓存了的值就不需要用户再登录了。

这里的this. authenticationManager是通过deployConfigContext.xml中定义注入的：

<**bean id="authenticationManager" class="org.jasig.cas.authentication.PolicyBasedAuthenticationManager"**>  
 <**constructor-arg**>  
 <**map**><**entry key-ref="proxyAuthenticationHandler" value-ref="proxyPrincipalResolver"** /><**entry key-ref="dbAuthHandler" value-ref="primaryPrincipalResolver"**/>  
 </**map**>  
 </**constructor-arg**><**property name="authenticationPolicy"**>  
 <**bean class="org.jasig.cas.authentication.AnyAuthenticationPolicy"** />  
 </**property**>  
</**bean**>

整个过程如下：



### 4．CAS服务器认证---->重定向目标资源

authenticationViaFormAction.submit方法执行后，根据不同的返回状态去到不到的节点。如果是success，去到sendTicketGrantingTicket

<**action-state id="sendTicketGrantingTicket"**>  
 <**evaluate expression="sendTicketGrantingTicketAction"**/>  
 <**transition to="serviceCheck"**/>  
</**action-state**>

sendTicketGrantingTicketAction是用来控制TGT创建和销毁的，然后走到serviceCheck：

<**decision-state id="serviceCheck"**>  
 <**if test="flowScope.service != null" then="generateServiceTicket" else="viewGenericLoginSuccess"**/>  
</**decision-state**>

判断flowScope.service是否为null，然后走向不同的节点

<**action-state id="generateServiceTicket"**>  
 <**evaluate expression="generateServiceTicketAction"**/>  
 <**transition on="success" to="warn"**/>  
 <**transition on="authenticationFailure" to="handleAuthenticationFailure"**/>  
 <**transition on="error" to="generateLoginTicket"**/>  
 <**transition on="gateway" to="gatewayServicesManagementCheck"**/>  
</**action-state**>

其中，generateServiceTicketAction是用来为TGT和service生成service ticket的

如果为success，则去到warn

<**decision-state id="warn"**>  
 <**if test="flowScope.warnCookieValue" then="showWarningView" else="redirect"**/>  
</**decision-state**>

<**action-state id="redirect"**>  
 <**evaluate expression="flowScope.service.getResponse(requestScope.serviceTicketId)"  
 result-type="org.jasig.cas.authentication.principal.Response" result="requestScope.response"**/>  
 <**transition to="postRedirectDecision"**/>  
</**action-state**>  
  
<**decision-state id="postRedirectDecision"**>  
 <**if test="requestScope.response.responseType.name() == 'POST'" then="postView" else="redirectView"**/>  
</**decision-state**>

<**end-state id="redirectView" view="externalRedirect:#{requestScope.response.url}"**/>

## 五、CAS+验证码

### 1.直接需要输入验证码

1.添加依赖：

<dependency>  
 <groupId>com.octo.captcha</groupId>  
 <artifactId>jcaptcha</artifactId>  
 <version>1.0</version>  
</dependency>

2. 修改CAS登录页面，在密码输入框下面添加如下代码

<**div class="row fl-controls-left"**>  
 <**label for="password" class="fl-label"**><**spring:message code="screen.welcome.label.vcode"** /></**label**>  
 <**img alt="vcode" src="captcha.jpg" height="50px;" width="190px;" style="padding-top**:5**px**;**padding-left**:5**px**;**"**>  
 <**spring:message code="screen.welcome.label.vcode.accesskey" var="vCodeAccessKey"** />  
 <**form:input cssClass="required" cssErrorClass="error" id="vcode" size="25" tabindex="1" accesskey="${**vCodeAccessKey**}" path="vcode" autocomplete="false" htmlEscape="true"** />  
</**div**>

其中，screen.welcome.label.vcode以及screen.welcome.label.vcode.accesskey需要根据自己需要去添加修改相应的messages\_xxxxx.properties文件内容，一般只需要改中文和英文即可

3.在login-webflow.xml中修改credential为自定义的UsernamePasswordVCodeCredentials

<**var name="credential" class="pers.chai.demo.UsernamePasswordVCodeCredentials"**/>

4. 原CAS提交在客户端验证之后就会提交到realSubmit，在这里我们修改login-webflow.xml中的内容为提交到vcodeSubmit

<**view-state id="viewLoginForm" view="casLoginViewRe" model="credential"**>  
 <**binder**>  
 <**binding property="username" required="true"**/>  
 <**binding property="password" required="true"**/>  
 <**binding property="vcode" required="true"**/>  
 </**binder**>  
 <**on-entry**>  
 <**set name="viewScope.commandName" value="'credential'"**/></**on-entry**>  
 <**transition on="submit" bind="true" validate="true" to="vcodeSubmit"**/>  
</**view-state**>  
  
<**action-state id="vcodeSubmit"**>  
 <**evaluate expression="vcodeViaFormAction.validatorCode(flowRequestContext, flowScope.credential, messageContext)"** />  
 <**transition on="success" to="realSubmit"** />  
 <**transition on="error" to="generateLoginTicket"** />  
</**action-state**>

5.添加完上述代码后，在cas-servlet.xml文件，添加下面代码：

<**bean id="vcodeViaFormAction" class="pers.chai.demo.VcodeAuthenticationViaFormAction"** />

6. 最后一步则是注册验证码生成器,在web.xml文件,加入以下代码：

<**servlet**>  
 <**servlet-name**>jcaptcha</**servlet-name**>  
 <**servlet-class**>pers.chai.demo.ImageCaptchaServlet</**servlet-class**>  
 <**load-on-startup**>0</**load-on-startup**>  
</**servlet**>  
<**servlet-mapping**>  
 <**servlet-name**>jcaptcha</**servlet-name**>  
 <**url-pattern**>/captcha.jpg</**url-pattern**>  
</**servlet-mapping**>

### 2.几次错误输入后需要输入验证码

1.添加UsersAuthenticationHandler类，并重写，其继承自AbstractUsernamePasswordAuthenticationHandler。在这个类中完成用户名、密码的校验和当校验不通过时增加计数。

<**bean id="codeAuthHandler" class="pers.chai.demo2.authcode.UsersAuthenticationHandler"**>  
 <**property name="accountService" ref="accountService"**/>  
</**bean**>

然后把codeAuthHandler交给authenticationManager管理。

2.类MyAuthenticationViaFormAction中完成点击“登录”时的逻辑

3.在casLoginVeiwRe.jsp中加入<c:if test="${errorNum>=5}"> </c:if>判断是否需要显示验证码

4.在web.xml中添加

<**servlet-mapping**>  
 <**servlet-name**>jcaptcha</**servlet-name**>  
 <**url-pattern**>/captcha.htm</**url-pattern**>  
</**servlet-mapping**>

我一开始参考博文：

<https://my.oschina.net/u/1412897/blog/1560440?tdsourcetag=s_pcqq_aiomsg>

<https://www.cnblogs.com/wangyang108/p/5844430.html>

的方式，希望用配置CaptchaImageCreateController的方式实现验证码的生成，但是总是显示不出来，也没有定位到具体的原因。？？？

最后只能在上一节的基础上作更改，最后成功了，得到了想要的效果。

## 六、登出

参考博文：<https://my.oschina.net/thinwonton/blog/1475562?from=timeline>

整个注销流程大致可以分为TGT解码和ticket销毁两个步骤

1. TGT解码

CAS服务接收请求后，获取浏览器的cookie中的tgc信息，对tgc信息进行解密，解密后将获取到tgt的ID，然后由CentralAuthenticationServiceImpl 类的 destroyTicketGrantingTicket（）方法注销该TGT。

1. ticket销毁

先完成客户端的ticket（ST）销毁，备注（2）完成CAS服务器的ticket(TGT)销毁

CentralAuthenticationServiceImpl 类的 destroyTicketGrantingTicket()方法

如果想要登出后跳转到service后的地址，需要把cas\_servlet.xml里的logoutAction参数的cas.logout.followServiceRedirects设为true

http://10.192.33.26:38080/cas/logout?service=http%3A%2F%2F10.192.33.37%3A8081%2Fdemo1%2Fbye