shiro+cas单点登录技术分析

简介：

面试时问cas，我想到cas锁，其实问的是单点登录。之前没接触过，所以抽时间了解了一下cas的单点登录。虽然没有公司的portal代码，但是看传输格式、及单点客户端配置跟官方推荐的一样，由于没有找到类似公司用portal代理单点登录客户端的例子，所以本文讲的不涉及代理的单点登录(代理的单点登录多涉及到了两个票据PGT PT)

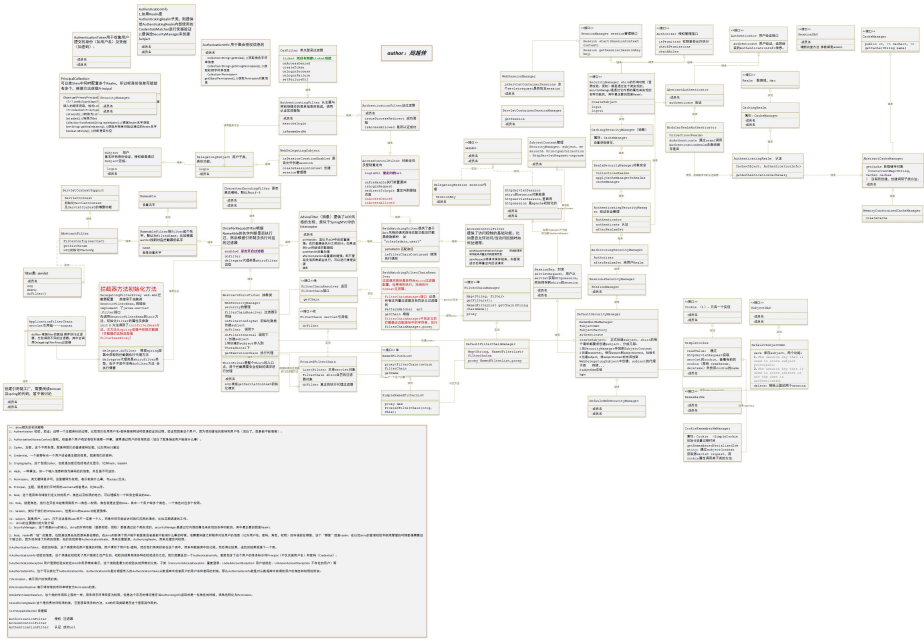
其次发现公司的单点登录都是明文传输的，只要获取TGT、ST（下文会讲解什么意思），就可以登录用户操作账户操作，如演示公司的http://oa.daikuan.com/ 后台管理系统，获取TGT之后，postMan请求为登录状态(可以看到首页的考勤等信息，而不是登陆页面哟)。

但是为了示例代码本地测试运行简单，没有使用ssl协议。并且本文主要介绍简单的(只有两个票据TGT、ST)单点登录流程，不是公司以portal代理的形式的单点流程（但是只要搞清楚基本的流程，portal代理CAS客户端的情景也是迎刃而解）。

本文会涉及到shiro及cas代码的理解，最好能跟着本文debug一遍就会清晰很多。为了更简单的了解其的原理，重复一下本实例代码没有使用portal代理形式、没有使用https协议。　　从网上down的项目地址稍做了修改，下载地址 https://github.com/JavaLover-ZYJ/shiro-cas

shiro、cas介绍：

1. shiro 介绍单点登录之前，先大概了解下shiro。下面是debug流程时画的一个不规范的shiro的类图，简单介绍了一下调用关系及类、方法的作用。



把图的文本重新粘出来一下：

一、shiro相关的名词解释

1、Authentication 校验，验证；证明一个主题身份的过程，比我我们在用用户名+密码登录网站时就是验证的过程，验证我就是这个用户，因为我知道他的密码和用户名（说白了，就是能不能登录）。

2、Authorization(Access Control):授权，检查某个用户有没有权利做哪一件事，通常通过用户的权限完成（说白了就是指定用户能做什么事）。

3、Cipher，加密，这个不用多想，就是将我们的普通密码加密，比如用MD5算法

4、Credential，一个能够标示一个用户或者是主题的信息，就是我们的密码。

5、Cryptography，这个包括Cipher，也就是加密还包括格式化显示，比如hash，base64.

6、Hash，一种算法，讲一个输入信息转换为编码后的信息，并且是不可逆的。

7、Permission，英文翻译是许可，这里翻译为权限，表示能做什么事，与subject无关。

8、Principal，主题，就是我们平时用的username或者是id，比如qq号。

9、Real，这个是用来存储我们定义好的用户、角色以及权限的地方，可以理解为一个和安全相关的dao。

10、Role，就是角色，我们在开发中经常用到用户-->角色-->权限，角色就是这里的Role。其中一个用户有多个角色，一个角色对应多个权限。

11、Session，类似于我们的HttpSession，但是shiro的session功能更强悍。

12、subject，就是用户，user。只不过这里的user并不一定是一个人，而是任何可能会访问我们应用的请求，比如定期调度的工作。

二、 shiro的主要接口的大致介绍

1、SecurityManager，这个类是shiro的核心，shiro的所有功能（登录校验、授权）都是通过这个类实现的，securityManager是通过它内部的属性来实现的各种功能的，其中最主要的就是Reaml。

2、Real，realm有“域”的意思，也就是说某些东西原来是在哪的，在shiro判断某个用户能不能登录或者是能不能做什么事的时候，他需要知道之前程序对该用户的信息（比如用户名，密码，角色，权限）的存储放在哪里，这个“哪里”就是realm。在以后shiro的登录校验和权限管理的时候都是需要这个接口的，因为他存储了所有的信息。他的实现类有AuthenticationRealm，用来处理登录，AuthorizingRealm，用来处理访问权限。

3.AuthenticationToken，校验的标签，这个类使用在用户登录的时候，用户填写了用户名+密码，然后我们将其封装在这个类中，用来和数据库中的比较，然后得出结果，返回的结果就是下一个类。

4.AuthenticationInfo 校验的信息，这个类是在校验完了用户登录之后产生的，校验的结果有很多种在校验成功之后，我们就要返回一个AuthenticationInfo，里面包含了这个用户的很多标示符Principal（不仅仅是用户名）和密码（Credential）。

5.AuthenticationException 用户登录校验失败在shiro中用异常来表示，这个类就是最大的校验失败异

常的父类。子类（ConcurrentAccessException　重复登录、LockedAccountException　用户被锁定、UnknownAccountException　不存在的用户）等

6.AuthorizationInfo，这个可以类比于AuthenticationInfo，AuthenticationInfo是对根据传入的AuthenticationToken从数据库中或者用户的用户名和密码的封装，那么AuthorizationInfo就是对从数据库中获得的用户的角色和权限的封装。

7,Permission，表示用户的权限的类。

8.PermissionResolver 表示将存储的字符串转变为Permission的类。

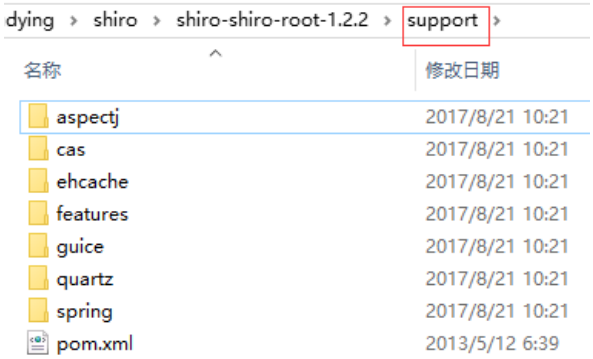
9,RolePermissionResolver，这个类的作用和上面的一样，用来将字符串转变为权限，但是这个存在的情况是在当AuthorizingInfo返回的是一些角色的时候，将角色转化为Permission。

10,AuthorizingRealm 这个是负责访问权限的类，它里面有很多的方法，6-9的所有类都是在这个里面起作用的。

11.PrincipalCollection 简答的理解是一个身份集合，暂不考虑多域的情况。

对于shiro再此不过多的展开表述，大家应该都有所了解，教程也有很多。可以把shiro看做是一套过滤器，其中包括授权、认证、自定义拦截器。支持单点、缓存、会话管理等。

2.cas：对于cas，是shiro对它进行的支持(除了cas之外还有aspectj、ehcache.. 如下图)



cas是一个基于HTTP(S)的协议，这就要求其每一个组成部分可以通过特定的URI访问到,shiro的源码上会对cas服务发送请求,最常见的就是验证票据的有效性。

先介绍一下TGT ST（还有PGT PT票据，是属于代理客户端的　此暂不考虑）

TGT:存放用户身份认证凭证的cookie，在浏览器和CAS服务端用来明确用户身份的凭证。对应多个系统是唯一的，cookie只存在cas服务端的域下。

ST：是每个应用在验证TGT通过后，生成的ST用于cas客户端与浏览器交互，CAS客户端会以此cookie值为key查询cas服务端缓存中有无TGT，验证通过后，允许用户访问资源。

下面列一下cas的主要请求地址：

1./login　　URI通过两种行为运转：一是作为一个凭证索取者，二是作为凭证接收者。根据它对凭证的反应来区分他是作为凭证索取者还是凭证接收者。

2./logout　　用于销毁客户端的CAS单点登录会话

3./validate　检查ST的有效性，/validate是CAS1.0协议的一部分，因此它不能处理代理认证。当一个代理凭证被传递给/validate时，CAS必须发出一个服务凭证验证失败的响应

4./serviceValidate　　检查ST的有效性，并且返回一个XML片段。

5./proxyValidate　　必须执行与/serviceValidate相同的验证任务，并且额外地还要能验证PT。

常用的请求地址就是 login、logout、serviceValidate。

单点登录客户端配置

1.web.xml　我们先来看cas客户端web.xml中的主要配置：

```
1.          <filter>
2.              <filter-name>singleSignOutFilter</filter-name>
3.              <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>
4.          </filter>
5.          <filter-mapping>
6.              <filter-name>singleSignOutFilter</filter-name>
7.              <url-pattern>/*</url-pattern>
8.          </filter-mapping>
9.
10.          <filter>
11.                  <filter-name>shiroFilter</filter-name> <!-- 注意这里的名称 要和spring-shiro.xml
中过滤器的id一样-->
12.                  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
13.                  <init-param>
14.                          <param-name>targetFilterLifecycle</param-name>
15.                          <param-value>true</param-value>
16.                  </init-param>
17.          </filter>
18.
19.          <filter-mapping>
20.                  <filter-name>shiroFilter</filter-name>
21.                  <url-pattern>/*</url-pattern>
22.          </filter-mapping>
23.
24.          <servlet>
25.                  <servlet-name>dispatcher</servlet-name>
26.                  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
27.                  <init-param>
28.                          <param-name>contextConfigLocation</param-name>
29.                          <param-value></param-value>
30.                  </init-param>
31.                  <load-on-startup>1</load-on-startup>
32.          </servlet>
33.
34.          <servlet-mapping>
35.                  <servlet-name>dispatcher</servlet-name>
36.                  <url-pattern>/</url-pattern>
37.          </servlet-mapping>
```

复制代码

其中有两点要注意：

a.shiro配置 （需写在Spring MVC Servlet配置之前）

b.单点登出要配置在shiroFilter之前

在上面的配置中是否有留意到这货：DelegatingFilterProxy，它会自动的把filter请求交给相应名称的bean处理。例如在启动时，spring会有一个filter请求，这个请求转交给了shiroFilter这个bean去处理了。so^接下来我们就得去找找看shiroFilter在哪———— spring-shiro.xml

2.spring-shiro.xml 的过滤器配置

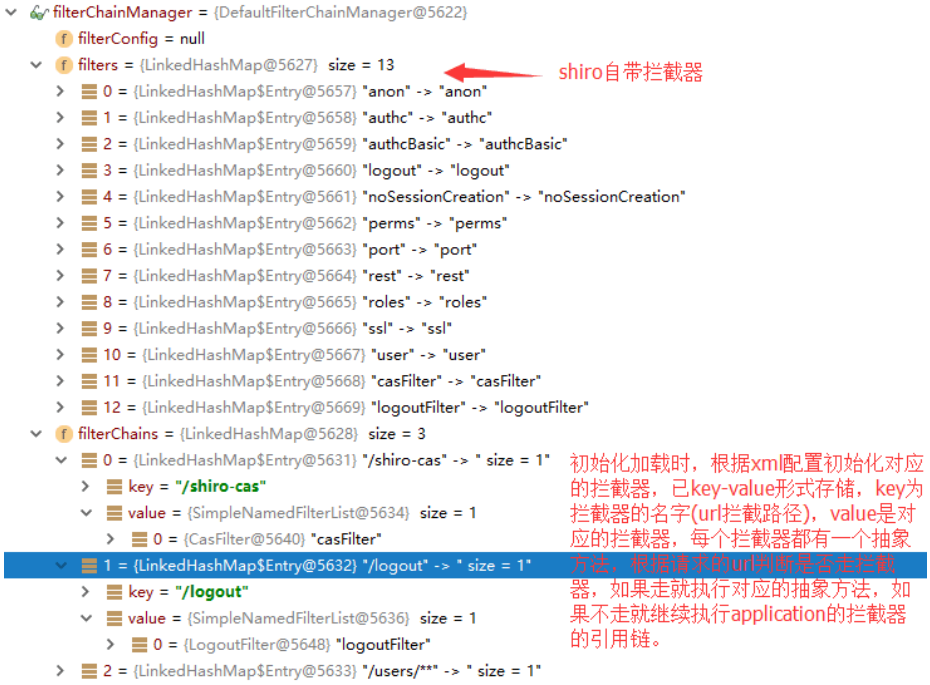
```
1.
2.          <!-- Shiro Filter -->
3.          <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
4.                  <property name="securityManager" ref="securityManager" />
5.                  <!-- 设定用户的登录链接，这里为cas登录页面的链接地址可配置回调地址 -->
6.                  <property name="loginUrl" value="${shiro.loginUrl}" />
7.                  <property name="filters">
8.                          <map>
9.                                  <!-- 添加casFilter到shiroFilter -->
10.                                 <entry key="casFilter" value-ref="casFilter" />
11.                                 <entry key="logoutFilter" value-ref="logoutFilter" />
12.                         </map>
13.                 </property>
14.                 <property name="filterChainDefinitions">
15.                         <value>
16.                                 /shiro-cas = casFilter
17.                                 /logout = logoutFilter
18.                                 /users/** = user
19.                         </value>
20.                 </property>
21.         </bean>
22.
23.         <bean id="casFilter" class="org.apache.shiro.cas.CasFilter">
24.                 <!-- 配置验证错误时的失败页面 -->
25.                 <property name="failureUrl" value="${shiro.failureUrl}" />
26.                 <property name="successUrl" value="${shiro.successUrl}" />
27.         </bean>
28.
29.         <bean id="logoutFilter" class="org.apache.shiro.web.filter.authc.LogoutFilter">
30.                 <!-- 配置验证错误时的失败页面 -->
31.                 <property name="redirectUrl" value="${shiro.logoutUrl}" />
32.         </bean>
33.
34.         <bean id="casRealm" class="com.spring.mybatis.realm.UserRealm">
35.                 <!-- 认证通过后的默认角色 -->
36.                 <property name="defaultRoles" value="ROLE_USER" />
37.                 <!-- cas服务端地址前缀 -->
38.                 <property name="casServerUrlPrefix" value="${shiro.cas.serverUrlPrefix}" />
39.                 <!-- 应用服务地址，用来接收cas服务端票据 -->
40.                 <property name="casService" value="${shiro.cas.service}" />
41.         </bean>
42.
43.         <!-- Shiro's main business-tier object for web-enabled applications -->
44.         <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
45.                 <!-- <property name="sessionManager" ref="sessionManager" /> -->
46.                 <property name="subjectFactory" ref="casSubjectFactory"></property>
47.                 <property name="realm" ref="casRealm" />
```

```
48.         </bean>
49.
50.         <bean id="casSubjectFactory" class="org.apache.shiro.cas.CasSubjectFactory"></bean>
51.         <bean
52.
class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
53.                 <property name="securityManager" ref="securityManager" />
54.         </bean>
```

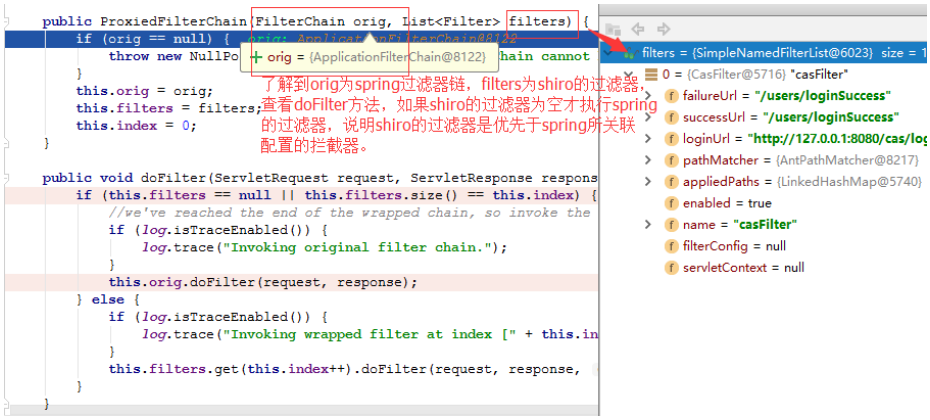
复制代码

（注意web.xml中的shiroFilter的名字 一定要与spring-shiro.xml中的bean的名字一样）

spring-shiro.xml中先来了解下shiroFilter的配置，cas客户端在启动过程中会把shiroFilter中过滤器的key值(请求的url地址)、对应的的过滤器的类 放到一个引用链的对象里面一一对应(就是filterChainManager)，根据请求路径url匹配对应的key，从而找到对应的过滤器，执行对应过滤器的代理方法(也就是图中描述的抽象方法)。如图所示，filters是shiro自带的过滤器，filterchains是我们要执行拦截的过滤器。此步骤为项目启动时的初始化加载。



通过web.xml的shiro拦截器拦截所有的请求，交接给shiroFilter这个bean处理，shiroFilter看是否有filterChains所配置的拦截器。如果没有则继续执行spring的引用链，如果有则执行shiro拦截器。并且shiro的连接器优先于spring的拦截器，如下图即可知道。 先判断shiro的拦截器是否为空，如果为空才执行ApplicationFilterChain的拦截器。



spring-shiro.xml中接下来强调的是casRealm。我们要实现的认证、预授权操作都在此自定义的Realm(也就是casRealm)中实现操作。cas客户端实现的UserRealm类继承自casRealm(代码如下)。里面的doGetAuthorizationInfo方法是对角色和权限的管理，此处不做介绍。主要介绍下doGetAuthenticationInfo方法，它主要是认证、验证身份。方法里面调用父类的doGetAuthenticationInfo(token)方法就是根据传递的token组装参数，向cas服务端发送/serviceValidate请求检查ST的有效性，并且返回一个XML片段。如果解析xml验证ST有效，且开启用户授权信息的缓存则更新缓存，另外重新创建subject，更新session设置Principal为解析的用户ID及权限、isAuthentication身份验证参数为true，则用户下次请求时直接根据session的principal及isAuthentication作为判断用户是否登陆(如果开启缓存，优先查询缓存中的信息)，不用再重新向cas服务端发送请求验证了(此步骤后文会详细描述)。验证完ST的有效性后可以根据用户的ID，查询出用户对象，并放到session中的一系列操作，都可以在doGetAuthenticationInfo方法中操作。

```
1. public class UserRealm extends CasRealm {
2.
3.          @Resource
4.          private RoleService roleService;
5.
6.          @Resource
7.          private UserService userService;
8.
9.          protected final Map<String, SimpleAuthorizationInfo> roles = new
ConcurrentHashMap<String, SimpleAuthorizationInfo>();
10.          /**
11.           * 设置角色和权限信息
12.           */
13.          @Override
14.          protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
15.                  String account = (String) principals.getPrimaryPrincipal();
16.                  SimpleAuthorizationInfo authorizationInfo = null;
17.                  if (authorizationInfo == null) {
18.                          authorizationInfo = new SimpleAuthorizationInfo();
19.
authorizationInfo.addStringPermissions(roleService.getPermissions(account));
20.                          authorizationInfo.addRoles(roleService.getRoles(account));
21.                          roles.put(account, authorizationInfo);
22.                  }
23.
24.                  return authorizationInfo;
25.          }
26.          /**
27.           * 1、CAS认证 ,验证用户身份
28.           * 2、将用户基本信息设置到会话中
29.           */
30.          protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) {
31.                  System.out.println("==================================================");
32.                  //此步骤是执行验证ST是否有效
33.                  AuthenticationInfo authc = super.doGetAuthenticationInfo(token);
```

```
34.
35.                 String account = (String) authc.getPrincipals().getPrimaryPrincipal();
36.
37.                 User user = userService.getUserByAccount(account);
38.
39.                 SecurityUtils.getSubject().getSession().setAttribute("user", user);
40.
41.                 return authc;
42.         }
43. }
```

复制代码

最后在大致了解下ShiroFilterFactoryBean，它引入了securityManager的管理，（SecurityManager：我们知道其在Shiro中的地位，类似于一个“安全大管家”，相当于SpringMVC中的DispatcherServlet或者Struts2中的FilterDispatcher，是Shiro的心脏，所有具体的交互都通过SecurityManager进行控制，它管理着所有Subject、且负责进行认证和授权、及会话、缓存的管理）。CasRealm也托管在securityManager

cas中ST、TGT缓存过期策略

cas客户端的缓存失效就是session失效(失效时间一般在web.xml下配置)，失效后重定向到cas服务端 ，带着cas服务端域下的cookies值CASTGC再次向cas服务端发送请求，cas服务端验证成功后生成ST。然后带着ST的参数重定向到cas客户端，客户端验证ST的有效性....(此步骤后文会详细描述)。这里的ST及TGT都在cas服务端缓存，先来了解下他俩的缓存失效机制。在配置文件ticketExpirationPolicies.xml下

```
1.
2.     <!-- Expiration policies -->
3.     <!-- ST票据过期配置，默认时间是10秒钟，使用次数为10 次或者超过100秒没有应用均会引起st过期，具体配置如-->
4.     <util:constant id="SECONDS" static-field="java.util.concurrent.TimeUnit.SECONDS"/>
5.     <bean id="serviceTicketExpirationPolicy"
class="org.jasig.cas.ticket.support.MultiTimeUseOrTimeoutExpirationPolicy"
6.           c:numberOfUses="10" c:timeToKill="${st.timeToKillInSeconds:100}" c:timeUnit-
ref="SECONDS"/>
7.
8.     <!-- Provides both idle and hard timeouts, for instance 2 hour sliding window with an 8 hour
max lifetime -->
9.     <!-- TGT票据过期配置，默认时间是两小时，当用户在2个小时（7200秒）之内不动移动鼠标或者进行系统超过8个小时（28800秒），则tgt过期，具体配置如下：-->
10.     <bean id="grantingTicketExpirationPolicy"
class="org.jasig.cas.ticket.support.TicketGrantingTicketExpirationPolicy"
11.           p:maxTimeToLiveInSeconds="${tgt.maxTimeToLiveInSeconds:28800}"
12.           p:timeToKillInSeconds="${tgt.timeToKillInSeconds:7200}"/>
13.     </beans>
```

复制代码

描述已经很清楚了，看一下代码的判断：

```
1.     public boolean isExpired(final TicketState ticketState) {
```

```
        // Ticket has been used, check maxTimeToLive (hard window)
        if ((System.currentTimeMillis() - ticketState.getCreationTime() >= maxTimeToLiveInMilliSeconds)) {
            LOGGER.debug("Ticket is expired because the time since creation is greater than maxTimeToLiveInMilliSeconds");
            return true;
        }

        // Ticket is within hard window, check timeToKill (sliding window)
        if ((System.currentTimeMillis() - ticketState.getLastTimeUsed() >= timeToKillInMilliSeconds)) {
            LOGGER.debug("Ticket is expired because the time since last use is greater than timeToKillInMilliseconds");
            return true;
        }

        return false;
    }
```

复制代码

过期策略是当前时间跟创建/最近使用时间的差值计算的，cas也有自己的过期(类似于session监听)监听来删除缓存的过期数据。

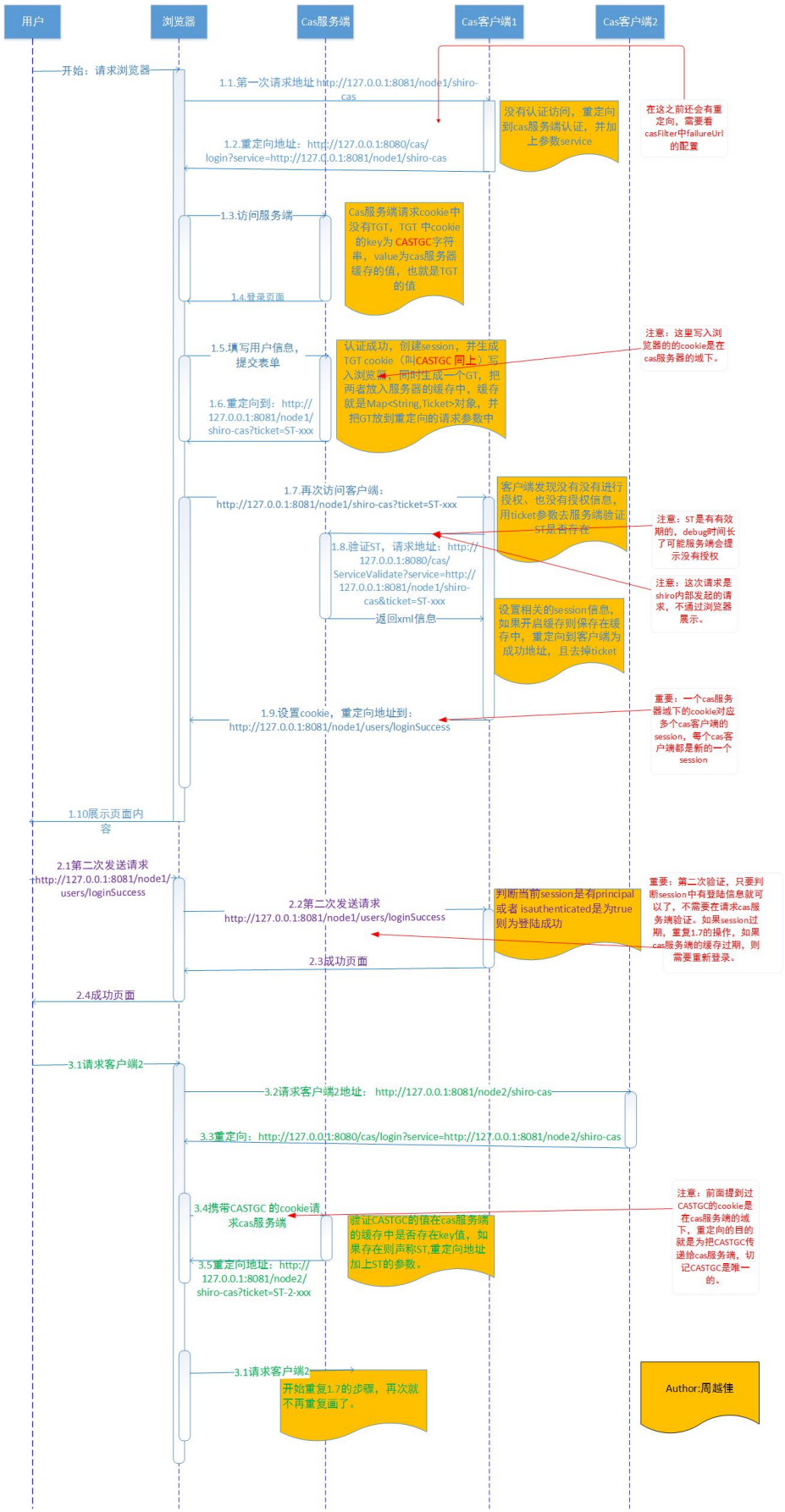
流程整体上先介绍这么多，接下来我们用详细的流程图来描述一下单点登录的请求流程，就更清晰了。

重点在下面

↓↓
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

单点登录请求流程图

先上图：描述了本示例的请求流程图。



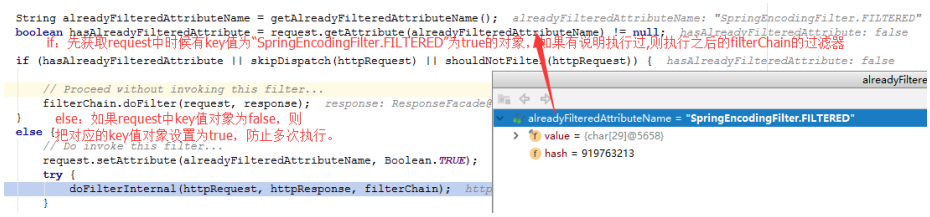
接下来按照流程图的序号，详细的说一下流程(与图片的描述会有重复)，流程会讲的比较细，涉及到个人认为比较重要的源码流程，有兴趣可以按照上面的流程图debug一下代码。

前提：分别启动cas服务端(使用8080端口) ，cas客户端1(使用8081端口)，cas客户端2(使用8082端口)。

一. 第一次访问cas客户端1

1.1 发送http://127.0.0.1:8081/node1/shiro-cas请求 ，访问cas客户端1， (注意用“【】” 包含的就每次请求cas客户端都会走的步骤，下面不在重复描述此步骤。开始)【先经过OncePerRequestFilter过滤器，它可以判

断同一个请求不会多次执行。如下截图：



防止同一请求多次执行。接下来创建subject，注意针对每一个请求或第一次验证登陆后都会创建subject，所以在这第一次详细说一下创建subject的流程，之后就一笔带过。

创建subject流程就是完善subjectContext对象的值，先介绍下subjectContext对象的结构就是Map<String, Object> backingMap;对象，其中的key-value就是如下字符串-对象：

1. private static final String SECURITY_MANAGER = DefaultSubjectContext.class.getName() + ".SECURITY_MANAGER"; 存储securityManager对象的key
2. private static final String SESSION_ID = DefaultSubjectContext.class.getName() + ".SESSION_ID";.....
3. private static final String AUTHENTICATION_TOKEN = DefaultSubjectContext.class.getName() + ".AUTHENTICATION_TOKEN"; 存储authentication的token对象的key
4. private static final String AUTHENTICATION_INFO = DefaultSubjectContext.class.getName() + ".AUTHENTICATION_INFO"; 存储authentication信息对象的key
5. private static final String SUBJECT = DefaultSubjectContext.class.getName() + ".SUBJECT"; 存储subject对象的key
6. private static final String PRINCIPALS = DefaultSubjectContext.class.getName() + ".PRINCIPALS"; 存储principals对象的key
7. private static final String SESSION = DefaultSubjectContext.class.getName() + ".SESSION"; 存储session对象的key
8. private static final String AUTHENTICATED = DefaultSubjectContext.class.getName() + ".AUTHENTICATED";.....
9. private static final String HOST = DefaultSubjectContext.class.getName() + ".HOST";.....
10. public static final String SESSION_CREATION_ENABLED = DefaultSubjectContext.class.getName() + ".SESSION_CREATION_ENABLED";.....

复制代码

创建subject流程：

a.创建subjectContext对象，给它设置SecurityManager对象(就是设置backingMap 的对象)；

b.创建session，session还是servlet的HttpServletRequest.getSession(false);去创建的,然后shiro又封装了一下变成了自己的HttpServletSession对象，然后放到backingMap 的对象中。

c.创建Principal ，也就是主题，常见的就是用户名。从cookie ，session中获取，然后放到backingMap 的对象中。当然第一次访问为空。

d.创建subject的代理对象，

e.保存subject，就是把 Principal及Authentication保存到session中，第一次访问都为空。以后的请求也会判断session是否有Principal或Authentication，有说明是登录状态了。

展示一下保存subject中proncipal的代码：

```
1. protected void mergePrincipals(Subject subject) {
2.         //merge PrincipalCollection state:
3.
4.         PrincipalCollection currentPrincipals = null;
5.
6.         //SHIRO-380: added if/else block - need to retain original (source) principals
7.         //This technique (reflection) is only temporary - a proper long term solution needs to
```

```
be found,
8.          //but this technique allowed an immediate fix that is API point-version forwards and
backwards compatible
9.          //
10.          //A more comprehensive review / cleaning of runAs should be performed for Shiro 1.3 /
2.0 +
11.          if (subject.isRunAs() && subject instanceof DelegatingSubject) {
12.              try {
13.                  Field field = DelegatingSubject.class.getDeclaredField("principals");
14.                  field.setAccessible(true);
15.                  currentPrincipals = (PrincipalCollection)field.get(subject);
16.              } catch (Exception e) {
17.                  throw new IllegalStateException("Unable to access DelegatingSubject principals
property.", e);
18.              }
19.          }
20.          if (currentPrincipals == null || currentPrincipals.isEmpty()) {
21.              currentPrincipals = subject.getPrincipals();
22.          }
23.
24.          Session session = subject.getSession(false);
25.
26.          if (session == null) {
27.              if (!CollectionUtils.isEmpty(currentPrincipals)) {
28.                  session = subject.getSession();
29.                  session.setAttribute(DefaultSubjectContext.PRINCIPALS_SESSION_KEY,
currentPrincipals);
30.              }
31.              //otherwise no session and no principals - nothing to save
32.          } else {
33.              PrincipalCollection existingPrincipals =
34.                      (PrincipalCollection)
session.getAttribute(DefaultSubjectContext.PRINCIPALS_SESSION_KEY);
35.
36.              if (CollectionUtils.isEmpty(currentPrincipals)) {
37.                  if (!CollectionUtils.isEmpty(existingPrincipals)) {
38.                      session.removeAttribute(DefaultSubjectContext.PRINCIPALS_SESSION_KEY);
39.                  }
40.                  //otherwise both are null or empty - no need to update the session
41.              } else {
42.                  if (!currentPrincipals.equals(existingPrincipals)) {
43.                      session.setAttribute(DefaultSubjectContext.PRINCIPALS_SESSION_KEY,
currentPrincipals);
44.                  }
45.                  //otherwise they're the same - no need to update the session
46.              }
47.          }
48.      }
```

复制代码

创建为subject之后，保存 subject 及 securityManager到ThreadContext 也就是ThreadLocal<Map<Object, Object>> resources对象中(在类ThreadContext中，可以查看代码)，对于当前的请求想获取subject对象数据可以从ThreadLocal中直接获取。】（结束。注意用“【】” 包含的就每次请求cas客户端都会走的步骤，下面不在重复描述此步骤）

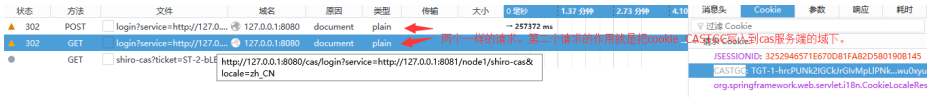
其次根据url判断是匹配，在前面说的 spring-shiro.xml 配置文件配置的过滤器，没有匹配就执行非shiro过滤器，本次请求路径的/shiro-cas 正好匹配casFilter过滤器，接下来经过Advice通知代理就执行casFilter过滤器。casFilter单点登录过滤器本身不会验证用户是否登录，casFilter的流程就是带着ST去cas服务端验证是否有效，通过subject的代理对象调用login方法，跳到前面所讲的UserRealm类的doGetAuthenticationInfo方法中，向发送http(就是这个请求 ./serviceValidate 检查ST的有效性，并且返回一个XML片段。)请求。由于第一次没有token参数，也就没有ST，所以失败(实际是报错了，被trycatch住了)。

1.2.然后重定向到地址：http://127.0.0.1:8080/cas/login? ... 081/node1/shiro-cas到浏览器 ，地址也是再spring-shiro.xml配置好的。

1.3.浏览器发起http://127.0.0.1:8080/cas/login? ... 081/node1/shiro-cas请求，cas服务端发现没有CASTGC的cookies参数，不能与缓存的TGT做比较。(注意：cookie中CASTGC字符串的value为cas服务器缓存TGT的key),所以重定向到登录页面。

1.4.展示登录页面，填写用户名、密码。

1.5.点击登录，再次发送请求http://127.0.0.1:8080/cas/login? ... 081/node1/shiro-cas。认证成功，创建session，并生成TGT （cookie中CASTGC的值就为cas服务端缓存TGT的key ），完成后执行一个重定向还是同一个请求：http://127.0.0.1:8080/cas/login? ... 081/node1/shiro-cas，这个重定向的作用是把cookie的值CASTGC写入浏览器cas服务端的域下。如下图：



同时生成一个GT，把两者放入服务器的缓存中，缓存就是Map<String,Ticket>对象，并把GT放到重定向的请求参数中。

1.6. 重定向到cas的客户端1，地址：http://127.0.0.1:8081/node1/shir ... K-cas01.example.org，并携带ticket参数，参数值就是ST。

1.7.发送如上地址 127.0.0.1:8081/node1/shiro-cas ? ticket=ST-2-bLBVE9eJSebV1sDP7x1K-cas01.example.org ， 执行 1.1中“【】”中的方法，根据请求地址判断执行casFilter过滤器，casFilter的流程就是带着ST去cas服务端验证是否有效，不判断是否登陆。通过subject的代理对象调用login方法，跳到前面所讲的UserRealm类的doGetAuthenticationInfo方法中，发送http请求。

1.8.向cas服务端发送/serviceValidate 的http请求（这次请求是shiro内部发起的请求，不通过浏览器展示。），检查ST的有效性，并且返回一个XML片段（注意:此处debug一定要注意，上面讲到cas服务端的ST的过期策略为100秒，如果debug停顿时间过长，验证ST就为未授权状态）如果为未授权，则重定向请求：http://127.0.0.1:8080/cas/login ? service=http://127.0.0.1:8081/node1/shiro-cas并带着cas服务端域下的CASTGC cookie，再次从1.5的步骤开始执行,不用重新登录只是带着CASTGC去申请前ST，如果TGT过期才需要登录。解析xml已经授权，则返回解析xml格式如下：

```
1. <cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>
2.         <cas:authenticationSuccess>
3.                 <cas:user>admin</cas:user>
4.         </cas:authenticationSuccess>
5. </cas:serviceResponse>
```

复制代码

再次创建subject（同1.1步骤中 执行创建subject一样的方法），更新session的Principal主题为 admin(权

限为空）、Authentication为授权状态（判断用户是否是登陆状态就是根据这两个参数，前提是缓存关闭，不然先查询缓存）。如果开启缓存则保存在缓存中，重定向到客户端为成功地址，且去掉ticket。

1.9.设置session，重定向地址到：http://127.0.0.1:8081/node1/users/loginSuccess

1.10.展示页面内容

二.第二次访问cas客户端1

2.1.再次发送http://127.0.0.1:8081/node1/shiro-cas请求浏览器(图中没有画这次请求），访问cas客户端1，执行 1.1中“【】”中的方法，由于没有token失败，重定向到浏览器地址：http://127.0.0.1:8081/node1/users/loginSuccess

2.2. 浏览器发送地址：http://127.0.0.1:8081/node1/users/loginSuccess 第二次请求cas客户端，第二次执行 1.1中“【】”中的方法，根据请求路径/users 判断执行的过滤器为shiro自带的userFilter过滤器，判断当前session是有principal 或者 isauthenticated是为true则为登陆成功（重要：第二次验证，只要判断session中有登陆信息就可以了，不需要在请求cas服务端验证。如果session过期，重复1.5的操作，不用重新登录只是带着CASTGC去申请前ST，如果TGT过期才需要登录，如果cas服务端的缓存过期，则需要重新登录。）

2.3返回浏览器成功页面

2.4展示成功页面

三.第一次访问cas客户端2

3.1 用户浏览器输入地址：http://127.0.0.1:8081/node2/shiro-cas

3.2 浏览器发送请求 http://127.0.0.1:8081/node2/shiro-cas。访问cas客户端2，执行 1.1中“【】”中的方法，由于没有token失败，则同上重定向到：http://127.0.0.1:8081/node1/users/loginSuccess 第二次请求cas客户端，第二次执行 1.1中“【】”中的方法，根据请求路径/users 判断执行的过滤器为shiro自带的userFilter过滤器，判断当前session没有principal 且 isauthenticated是为false。则需要向cas服务端申请ST，然后重定向到

3.3 重定向到浏览器的地址：http://127.0.0.1:8080/cas/login? ... 081/node2/shiro-cas

3.4 浏览器发送请求 http://127.0.0.1:8080/cas/login? ... 081/node2/shiro-cas ，并且携带cas服务端域下的CASTGC的cookie，验证CASTGC的值在cas服务端的缓存中key值中存在，如果存在则重新重生一个ST,重定向地址加上ST的参数。

3.5 重定向到浏览器地址：http://127.0.0.1:8081/node2/shir ... K-cas02.example.org

3.6 接下来开始重复1.7的步骤 。。。

总的来说，cas服务端cookies的值CASTGC，是全局的一个票据，对应cas客户端多个session(切记是1：多的关系)，每个session对应一个ST，它由TGT颁发的票据，ST用完就销毁或直接过期，登陆信息由cas客户端的session保持。如果session失效就重定向到cas服务端的地址(http://127.0.0.1:8080/cas/login.....)，然后带着CASTGC去获取ST。如果CASTGC验证失效，则需要用户登录。

文章写的比较草也比较乱，有错误请提出。

谢谢！

----------完----------