springboot使用shiro-整合redis作为缓存(十)

原文地址，转载请注明出处： https://blog.csdn.net/qq_34021712/article/details/80774649 ©王赛超

说在前面

本来的整合过程是顺着博客的顺序来的,越往下,集成的越多,由于之前是使用ehcache缓存,现在改为redis,限制登录人数 以及 限制登录次数等 都需要改动,本篇为了简单,目前先将这两个功能下线，配置暂时是注销的,原类保存,在下篇博客中改。
还有之前是使用SessionListener监听session创建来统计在线人数,在本篇中也将改为统计redis中的key数目。
如果是单机,使用ehcache是最快的,项目一般都不是单节点,为了方便之后使用sso单点登录,以及多节点部署,所以使用 shiro 整合redis。这里有一个开源项目,git地址为: https://github.com/alexxiyang/shiro-redis.git 在此感谢作者无私奉献。

整合过程

shiro用redis实现缓存需要重写 `cache` 、 `cacheManager` 、 `SessionDAO` 和初始化redis配置。

pom添加依赖

```
<!-- 整合shiro框架 -->
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.4.0</version>
</dependency>
<!-- shiro-thymeleaf 2.0.0-->
<dependency>
    <groupId>com.github.theborakompanioni</groupId>
    <artifactId>thymeleaf-extras-shiro</artifactId>
    <version>1.2.1</version>
</dependency>
<!-- shiro-redis -->
<dependency>
  <groupId>org.crazycake</groupId>
    <artifactId>shiro-redis</artifactId>
    <version>3.1.0</version>
</dependency>
```

ShiroConfig.java

```
package com.springboot.test.shiro.config;

import at.pollux.thymeleaf.shiro.dialect.ShiroDialect;
import com.springboot.test.shiro.config.shiro.*;
import org.apache.shiro.codec.Base64;
import org.apache.shiro.session.SessionListener;
import org.apache.shiro.session.mgt.SessionManager;
import org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator;
import org.apache.shiro.session.mgt.eis.SessionDAO;
import org.apache.shiro.session.mgt.eis.SessionIdGenerator;
import org.apache.shiro.spring.LifecycleBeanPostProcessor;
import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.web.filter.authc.FormAuthenticationFilter;
import org.apache.shiro.web.mgt.CookieRememberMeManager;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.apache.shiro.web.servlet.SimpleCookie;
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;
import org.crazycake.shiro.RedisCacheManager;
import org.crazycake.shiro.RedisManager;
import org.crazycake.shiro.RedisSessionDAO;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.config.MethodInvokingFactoryBean;
import org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;
import org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;
import org.springframework.boot.web.servlet.ErrorPage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.web.servlet.handler.SimpleMappingExceptionResolver;

import javax.servlet.Filter;
import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedHashMap;
import java.util.Properties;
```

```java

/**
 * @author: wangsaichao
 * @date: 2018/5/10
 * @description: Shiro配置
 */
@Configuration
public class ShiroConfig {


    /**
     * ShiroFilterFactoryBean 处理拦截资源文件问题。
     * 注意：初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager
     * Web应用中,Shiro可控制的Web请求必须经过Shiro主过滤器的拦截
     * @param securityManager
     * @return
     */
    @Bean(name = "shirFilter")
    public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager securityManager) {

        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

        //必须设置 SecurityManager,Shiro的核心安全接口
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        //这里的/login是后台的接口名,非页面，如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
        shiroFilterFactoryBean.setLoginUrl("/");
        //这里的/index是后台的接口名,非页面,登录成功后要跳转的链接
        shiroFilterFactoryBean.setSuccessUrl("/index");
        //未授权界面,该配置无效，并不会进行页面跳转
        shiroFilterFactoryBean.setUnauthorizedUrl("/unauthorized");

        //自定义拦截器限制并发人数,参考博客：
        LinkedHashMap<String, Filter> filtersMap = new LinkedHashMap<>();
        //限制同一帐号同时在线的个数
        //filtersMap.put("kickout", kickoutSessionControlFilter());
        //统计登录人数
        shiroFilterFactoryBean.setFilters(filtersMap);

        // 配置访问权限 必须是LinkedHashMap，因为它必须保证有序
        // 过滤链定义，从上向下顺序执行，一般将 /**放在最为下边 --> : 这是一个坑，一不小心代码就不好使了
        LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
        //配置不登录可以访问的资源，anon 表示资源都可以匿名访问
        //配置记住我或认证通过可以访问的地址
        filterChainDefinitionMap.put("/login", "anon");
        filterChainDefinitionMap.put("/", "anon");
        filterChainDefinitionMap.put("/css/**", "anon");
        filterChainDefinitionMap.put("/js/**", "anon");
        filterChainDefinitionMap.put("/img/**", "anon");
        filterChainDefinitionMap.put("/druid/**", "anon");
        //解锁用户专用 测试用的
        filterChainDefinitionMap.put("/unlockAccount","anon");
        filterChainDefinitionMap.put("/Captcha.jpg","anon");
        //logout是shiro提供的过滤器
        filterChainDefinitionMap.put("/logout", "logout");
        //此时访问/user/delete需要delete权限,在自定义Realm中为用户授权。
        //filterChainDefinitionMap.put("/user/delete", "perms[\"user:delete\"]");

        //其他资源都需要认证  authc 表示需要认证才能进行访问 user表示配置记住我或认证通过可以访问的地址
        //如果开启限制同一账号登录,改为 .put("/**", "kickout,user");
        filterChainDefinitionMap.put("/**", "user");

        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);

        return shiroFilterFactoryBean;
    }

    /**
     * 配置核心安全事务管理器
     * @return
     */
    @Bean(name="securityManager")
    public SecurityManager securityManager() {
        DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
        //设置自定义realm.
        securityManager.setRealm(shiroRealm());
        //配置记住我
        securityManager.setRememberMeManager(rememberMeManager());
        //配置redis缓存
        securityManager.setCacheManager(cacheManager());
```

```
        //配置自定义session管理，使用redis
        securityManager.setSessionManager(sessionManager());
        return securityManager;
    }

    /**
     * 配置Shiro生命周期处理器
     * @return
     */
    @Bean(name = "lifecycleBeanPostProcessor")
    public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {
        return new LifecycleBeanPostProcessor();
    }

    /**
     *  身份认证realm；(这个需要自己写，账号密码校验；权限等)
     * @return
     */
    @Bean
    public ShiroRealm shiroRealm(){
        ShiroRealm shiroRealm = new ShiroRealm();
        shiroRealm.setCachingEnabled(true);
        //启用身份验证缓存，即缓存AuthenticationInfo信息，默认false
        shiroRealm.setAuthenticationCachingEnabled(true);
        //缓存AuthenticationInfo信息的缓存名称 在ehcache-shiro.xml中有对应缓存的配置
        shiroRealm.setAuthenticationCacheName("authenticationCache");
        //启用授权缓存，即缓存AuthorizationInfo信息，默认false
        shiroRealm.setAuthorizationCachingEnabled(true);
        //缓存AuthorizationInfo信息的缓存名称  在ehcache-shiro.xml中有对应缓存的配置
        shiroRealm.setAuthorizationCacheName("authorizationCache");
        //配置自定义密码比较器
        //shiroRealm.setCredentialsMatcher(retryLimitHashedCredentialsMatcher());
        return shiroRealm;
    }

    /**
     * 必须（thymeleaf页面使用shiro标签控制按钮是否显示）
     * 未引入thymeleaf包，Caused by: java.lang.ClassNotFoundException: org.thymeleaf.dialect.AbstractProcessorDialect
     * @return
     */
    @Bean
    public ShiroDialect shiroDialect() {
        return new ShiroDialect();
    }

    /**
     * 开启shiro 注解模式
     * 可以在controller中的方法前加上注解
     * 如 @RequiresPermissions("userInfo:add")
     * @param securityManager
     * @return
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(@Qualifier("securityManager") SecurityManager secur
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }

    /**
     * 解决：  无权限页面不跳转 shiroFilterFactoryBean.setUnauthorizedUrl("/unauthorized") 无效
     * shiro的源代码ShiroFilterFactoryBean.Java定义的filter必须满足filter instanceof AuthorizationFilter,
     * 只有perms，roles，ssl，rest，port才是属于AuthorizationFilter，而anon，authcBasic，auchc，user是AuthenticationFilter,
     * 所以unauthorizedUrl设置后页面不跳转 Shiro注解模式下，登录失败与没有权限都是通过抛出异常。
     * 并且默认并没有去处理或者捕获这些异常。在SpringMVC下需要配置捕获相应异常来通知用户信息
     * @return
     */
    @Bean
    public SimpleMappingExceptionResolver simpleMappingExceptionResolver() {
        SimpleMappingExceptionResolver simpleMappingExceptionResolver=new SimpleMappingExceptionResolver();
        Properties properties=new Properties();
        //这里的 /unauthorized 是页面，不是访问的路径
        properties.setProperty("org.apache.shiro.authz.UnauthorizedException","/unauthorized");
        properties.setProperty("org.apache.shiro.authz.UnauthenticatedException","/unauthorized");
        simpleMappingExceptionResolver.setExceptionMappings(properties);
        return simpleMappingExceptionResolver;
    }

    /**
```

```
     * 解决spring-boot Whitelabel Error Page
     * @return
     */
    @Bean
    public EmbeddedServletContainerCustomizer containerCustomizer() {

        return new EmbeddedServletContainerCustomizer() {
            @Override
            public void customize(ConfigurableEmbeddedServletContainer container) {

                ErrorPage error401Page = new ErrorPage(HttpStatus.UNAUTHORIZED, "/unauthorized.html");
                ErrorPage error404Page = new ErrorPage(HttpStatus.NOT_FOUND, "/404.html");
                ErrorPage error500Page = new ErrorPage(HttpStatus.INTERNAL_SERVER_ERROR, "/500.html");

                container.addErrorPages(error401Page, error404Page, error500Page);
            }
        };
    }

    /**
     * cookie对象;会话Cookie模板 ,默认为: JSESSIONID 问题: 与SERVLET容器名冲突,重新定义为sid或rememberMe，自定义
     * @return
     */
    @Bean
    public SimpleCookie rememberMeCookie(){
        //这个参数是cookie的名称，对应前端的checkbox的name = rememberMe
        SimpleCookie simpleCookie = new SimpleCookie("rememberMe");
        //setcookie的httponly属性如果设为true的话，会增加对xss防护的安全系数。它有以下特点：
        //setcookie()的第七个参数
        //设为true后，只能通过http访问，javascript无法访问
        //防止xss读取cookie
        simpleCookie.setHttpOnly(true);
        simpleCookie.setPath("/");
        //<!-- 记住我cookie生效时间30天 ,单位秒;-->
        simpleCookie.setMaxAge(2592000);
        return simpleCookie;
    }

    /**
     * cookie管理对象;记住我功能,rememberMe管理器
     * @return
     */
    @Bean
    public CookieRememberMeManager rememberMeManager(){
        CookieRememberMeManager cookieRememberMeManager = new CookieRememberMeManager();
        cookieRememberMeManager.setCookie(rememberMeCookie());
        //rememberMe cookie加密的密钥 建议每个项目都不一样 默认AES算法 密钥长度(128 256 512 位)
        cookieRememberMeManager.setCipherKey(Base64.decode("4AvVhmFLUs0KTA3Kprsdag=="));
        return cookieRememberMeManager;
    }

    /**
     * FormAuthenticationFilter 过滤器 过滤记住我
     * @return
     */
    @Bean
    public FormAuthenticationFilter formAuthenticationFilter(){
        FormAuthenticationFilter formAuthenticationFilter = new FormAuthenticationFilter();
        //对应前端的checkbox的name = rememberMe
        formAuthenticationFilter.setRememberMeParam("rememberMe");
        return formAuthenticationFilter;
    }

    /**
     * shiro缓存管理器;
     * 需要添加到securityManager中
     * @return
     */
    @Bean
    public RedisCacheManager cacheManager(){
        RedisCacheManager redisCacheManager = new RedisCacheManager();
        redisCacheManager.setRedisManager(redisManager());
        //redis中针对不同用户缓存
        redisCacheManager.setPrincipalIdFieldName("username");
        //用户权限信息缓存时间
        redisCacheManager.setExpire(200000);
        return redisCacheManager;
    }

```

```
    /**
     * 让某个实例的某个方法的返回值注入为Bean的实例
     * Spring静态注入
     * @return
     */
    @Bean
    public MethodInvokingFactoryBean getMethodInvokingFactoryBean(){
        MethodInvokingFactoryBean factoryBean = new MethodInvokingFactoryBean();
        factoryBean.setStaticMethod("org.apache.shiro.SecurityUtils.setSecurityManager");
        factoryBean.setArguments(new Object[]{securityManager()});
        return factoryBean;
    }

    /**
     * 配置session监听
     * @return
     */
    @Bean("sessionListener")
    public ShiroSessionListener sessionListener(){
        ShiroSessionListener sessionListener = new ShiroSessionListener();
        return sessionListener;
    }

    /**
     * 配置会话ID生成器
     * @return
     */
    @Bean
    public SessionIdGenerator sessionIdGenerator() {
        return new JavaUuidSessionIdGenerator();
    }

    @Bean
    public RedisManager redisManager(){
        RedisManager redisManager = new RedisManager();
        redisManager.setHost("127.0.0.1");
        redisManager.setPort(6379);
        redisManager.setPassword("123456");
        return redisManager;
    }



    /**
     * SessionDAO的作用是为Session提供CRUD并进行持久化的一个shiro组件
     * MemorySessionDAO 直接在内存中进行会话维护
     * EnterpriseCacheSessionDAO  提供了缓存功能的会话维护，默认情况下使用MapCache实现，内部使用ConcurrentHashMap保存缓存的会话。
     * @return
     */
    @Bean
    public SessionDAO sessionDAO() {
        RedisSessionDAO redisSessionDAO = new RedisSessionDAO();
        redisSessionDAO.setRedisManager(redisManager());
        //session在redis中的保存时间,最好大于session会话超时时间
        redisSessionDAO.setExpire(12000);
        return redisSessionDAO;
    }

    /**
     * 配置保存sessionId的cookie
     * 注意：这里的cookie 不是上面的记住我 cookie 记住我需要一个cookie session管理 也需要自己的cookie
     * 默认为: JSESSIONID 问题: 与SERVLET容器名冲突,重新定义为sid
     * @return
     */
    @Bean("sessionIdCookie")
    public SimpleCookie sessionIdCookie(){
        //这个参数是cookie的名称
        SimpleCookie simpleCookie = new SimpleCookie("sid");
        //setcookie的httponly属性如果设为true的话，会增加对xss防护的安全系数。它有以下特点：

        //setcookie()的第七个参数
        //设为true后，只能通过http访问，javascript无法访问
        //防止xss读取cookie
        simpleCookie.setHttpOnly(true);
        simpleCookie.setPath("/");
        //maxAge=-1表示浏览器关闭时失效此Cookie
        simpleCookie.setMaxAge(-1);
        return simpleCookie;
    }
```

```
354
355     /**
356      * 配置会话管理器，设定会话超时及保存
357      * @return
358      */
359     @Bean("sessionManager")
360     public SessionManager sessionManager() {
361         DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();
362         Collection<SessionListener> listeners = new ArrayList<SessionListener>();
363         //配置监听
364         listeners.add(sessionListener());
365         sessionManager.setSessionListeners(listeners);
366         sessionManager.setSessionIdCookie(sessionIdCookie());
367         sessionManager.setSessionDAO(sessionDAO());
368         sessionManager.setCacheManager(cacheManager());
369
370         //全局会话超时时间（单位毫秒），默认30分钟  暂时设置为10秒钟 用来测试
371         sessionManager.setGlobalSessionTimeout(1800000);
372         //是否开启删除无效的session对象  默认为true
373         sessionManager.setDeleteInvalidSessions(true);
374         //是否开启定时调度器进行检测过期session 默认为true
375         sessionManager.setSessionValidationSchedulerEnabled(true);
376         //设置session失效的扫描时间, 清理用户直接关闭浏览器造成的孤立会话 默认为 1个小时
377         //设置该属性 就不需要设置 ExecutorServiceSessionValidationScheduler 底层也是默认自动调用ExecutorServiceSessionValidationScheduler
378         //暂时设置为 5秒 用来测试
379         sessionManager.setSessionValidationInterval(3600000);
380         //取消url 后面的 JSESSIONID
381         sessionManager.setSessionIdUrlRewritingEnabled(false);
382         return sessionManager;
383
384     }
385
386
387     /**
388      * 并发登录控制
389      * @return
390      */
391 //    @Bean
392 //    public KickoutSessionControlFilter kickoutSessionControlFilter(){
393 //        KickoutSessionControlFilter kickoutSessionControlFilter = new KickoutSessionControlFilter();
394 //        //用于根据会话ID，获取会话进行踢出操作的；
395 //        kickoutSessionControlFilter.setSessionManager(sessionManager());
396 //        //使用cacheManager获取相应的cache来缓存用户登录的会话；用于保存用户—会话之间的关系的；
397 //        kickoutSessionControlFilter.setCacheManager(cacheManager());
398 //        //是否踢出后来登录的，默认是false；即后者登录的用户踢出前者登录的用户；
399 //        kickoutSessionControlFilter.setKickoutAfter(false);
400 //        //同一个用户最大的会话数，默认1；比如2的意思是同一个用户允许最多同时两个人登录；
401 //        kickoutSessionControlFilter.setMaxSession(1);
402 //        //被踢出后重定向到的地址；
403 //        kickoutSessionControlFilter.setKickoutUrl("/login?kickout=1");
404 //        return kickoutSessionControlFilter;
405 //    }
406
407     /**
408      * 配置密码比较器
409      * @return
410      */
411 //    @Bean("credentialsMatcher")
412 //    public RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher(){
413 //        RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher = new RetryLimitHashedCredentialsMatcher(cacheManager(
414 //
415 //        //如果密码加密,可以打开下面配置
416 //        //加密算法的名称
417 //        //retryLimitHashedCredentialsMatcher.setHashAlgorithmName("MD5");
418 //        //配置加密的次数
419 //        //retryLimitHashedCredentialsMatcher.setHashIterations(1024);
420 //        //是否存储为16进制
421 //        //retryLimitHashedCredentialsMatcher.setStoredCredentialsHexEncoded(true);
422 //
423 //        return retryLimitHashedCredentialsMatcher;
424 //    }
425
426
427 }
```

ShiroRealm.java

```java
package com.springboot.test.shiro.config.shiro;

import com.springboot.test.shiro.modules.user.dao.PermissionMapper;
import com.springboot.test.shiro.modules.user.dao.RoleMapper;
import com.springboot.test.shiro.modules.user.dao.entity.Permission;
import com.springboot.test.shiro.modules.user.dao.entity.Role;
import com.springboot.test.shiro.modules.user.dao.UserMapper;
import com.springboot.test.shiro.modules.user.dao.entity.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

/**
 * @author: wangsaichao
 * @date: 2018/5/10
 * @description: 在Shiro中，最终是通过Realm来获取应用程序中的用户、角色及权限信息的
 * 在Realm中会直接从我们的数据源中获取Shiro需要的验证信息。可以说，Realm是专用于安全框架的DAO.
 */
public class ShiroRealm extends AuthorizingRealm {

    @Autowired
    private UserMapper userMapper;

    @Autowired
    private RoleMapper roleMapper;

    @Autowired
    private PermissionMapper permissionMapper;

    /**
     * 验证用户身份
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

        //获取用户名密码 第一种方式
        //String username = (String) authenticationToken.getPrincipal();
        //String password = new String((char[]) authenticationToken.getCredentials());

        //获取用户名 密码 第二种方式
        UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) authenticationToken;
        String username = usernamePasswordToken.getUsername();
        String password = new String(usernamePasswordToken.getPassword());

        //从数据库查询用户信息
        User user = this.userMapper.findByUserName(username);

        //可以在这里直接对用户名校验,或者调用 CredentialsMatcher 校验
        if (user == null) {
            throw new UnknownAccountException("用户名或密码错误！");
        }
        //这里将 密码对比 注销掉,否则 无法锁定  要将密码对比 交给 密码比较器
        //if (!password.equals(user.getPassword())) {
        //    throw new IncorrectCredentialsException("用户名或密码错误！");
        //}
        if ("1".equals(user.getState())) {
            throw new LockedAccountException("账号已被锁定,请联系管理员！");
        }

        //调用 CredentialsMatcher 校验 还需要创建一个类 继承CredentialsMatcher  如果在上面校验了,这个就不需要了
        //配置自定义权限登录器 参考博客:

        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), getName());
        return info;
    }

    /**
     * 授权用户权限
     * 授权的方法是在碰到<shiro:hasPermission name=''></shiro:hasPermission>标签的时候调用的
```

```
     * 它会去检测shiro框架中的权限(这里的permissions)是否包含有该标签的name值,如果有,里面的内容显示
     * 如果没有,里面的内容不予显示(这就完成了对于权限的认证.)
     *
     * shiro的权限授权是通过继承AuthorizingRealm抽象类，重载doGetAuthorizationInfo();
     * 当访问到页面的时候，链接配置了相应的权限或者shiro标签才会执行此方法否则不会执行
     * 所以如果只是简单的身份认证没有权限的控制的话，那么这个方法可以不进行实现，直接返回null即可。
     *
     * 在这个方法中主要是使用类：SimpleAuthorizationInfo 进行角色的添加和权限的添加。
     * authorizationInfo.addRole(role.getRole()); authorizationInfo.addStringPermission(p.getPermission());
     *
     * 当然也可以添加set集合：roles是从数据库查询的当前用户的角色，stringPermissions是从数据库查询的当前用户对应的权限
     * authorizationInfo.setRoles(roles); authorizationInfo.setStringPermissions(stringPermissions);
     *
     * 就是说如果在shiro配置文件中添加了filterChainDefinitionMap.put("/add", "perms[权限添加]");
     * 就说明访问/add这个链接必须要有“权限添加”这个权限才可以访问
     *
     * 如果在shiro配置文件中添加了filterChainDefinitionMap.put("/add", "roles[100002]，perms[权限添加]");
     * 就说明访问/add这个链接必须要有 "权限添加" 这个权限和具有 "100002" 这个角色才可以访问
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

        System.out.println("查询权限方法调用了！！！");

        //获取用户
        User user = (User) SecurityUtils.getSubject().getPrincipal();

        //获取用户角色
        Set<Role> roles =this.roleMapper.findRolesByUserId(user.getUid());
        //添加角色
        SimpleAuthorizationInfo authorizationInfo =  new SimpleAuthorizationInfo();
        for (Role role : roles) {
            authorizationInfo.addRole(role.getRole());
        }

        //获取用户权限
        Set<Permission> permissions = this.permissionMapper.findPermissionsByRoleId(roles);
        //添加权限
        for (Permission permission:permissions) {
            authorizationInfo.addStringPermission(permission.getPermission());
        }

        return authorizationInfo;
    }

    /**
     * 重写方法,清除当前用户的的 授权缓存
     * @param principals
     */
    @Override
    public void clearCachedAuthorizationInfo(PrincipalCollection principals) {
        super.clearCachedAuthorizationInfo(principals);
    }

    /**
     * 重写方法，清除当前用户的 认证缓存
     * @param principals
     */
    @Override
    public void clearCachedAuthenticationInfo(PrincipalCollection principals) {
        super.clearCachedAuthenticationInfo(principals);
    }

    @Override
    public void clearCache(PrincipalCollection principals) {
        super.clearCache(principals);
    }

    /**
     * 自定义方法：清除所有 授权缓存
     */
    public void clearAllCachedAuthorizationInfo() {
        getAuthorizationCache().clear();
    }

    /**
     * 自定义方法：清除所有 认证缓存
```

```
159      */
160     public void clearAllCachedAuthenticationInfo() {
161         getAuthenticationCache().clear();
162     }
163 
164     /**
165      * 自定义方法：清除所有的  认证缓存  和 授权缓存
166      */
167     public void clearAllCache() {
168         clearAllCachedAuthenticationInfo();
169         clearAllCachedAuthorizationInfo();
170     }
171 
172 }
```

我们可以看一下,在redis中的缓存,如下：