springboot+shiro+redis项目整合

介绍：

Apache Shiro是一个强大且易用的Java安全框架,执行身份验证、授权、密码学和会话管理。使用Shiro的易于理解的API,您可以快速、轻松地获得任何应用程序,从最小的移动应用程序到最大的网络和企业应用程序。（摘自百度百科）

本文使用springboot+mybatisplus＋shiro实现数据库动态的管理用户、角色、权限管理，在本文的最后我会提供源码的下载地址，想看到效果的小伙伴可以直接下载运行就ok了

因为shiro的功能比较多，本章只介绍如下几个功能

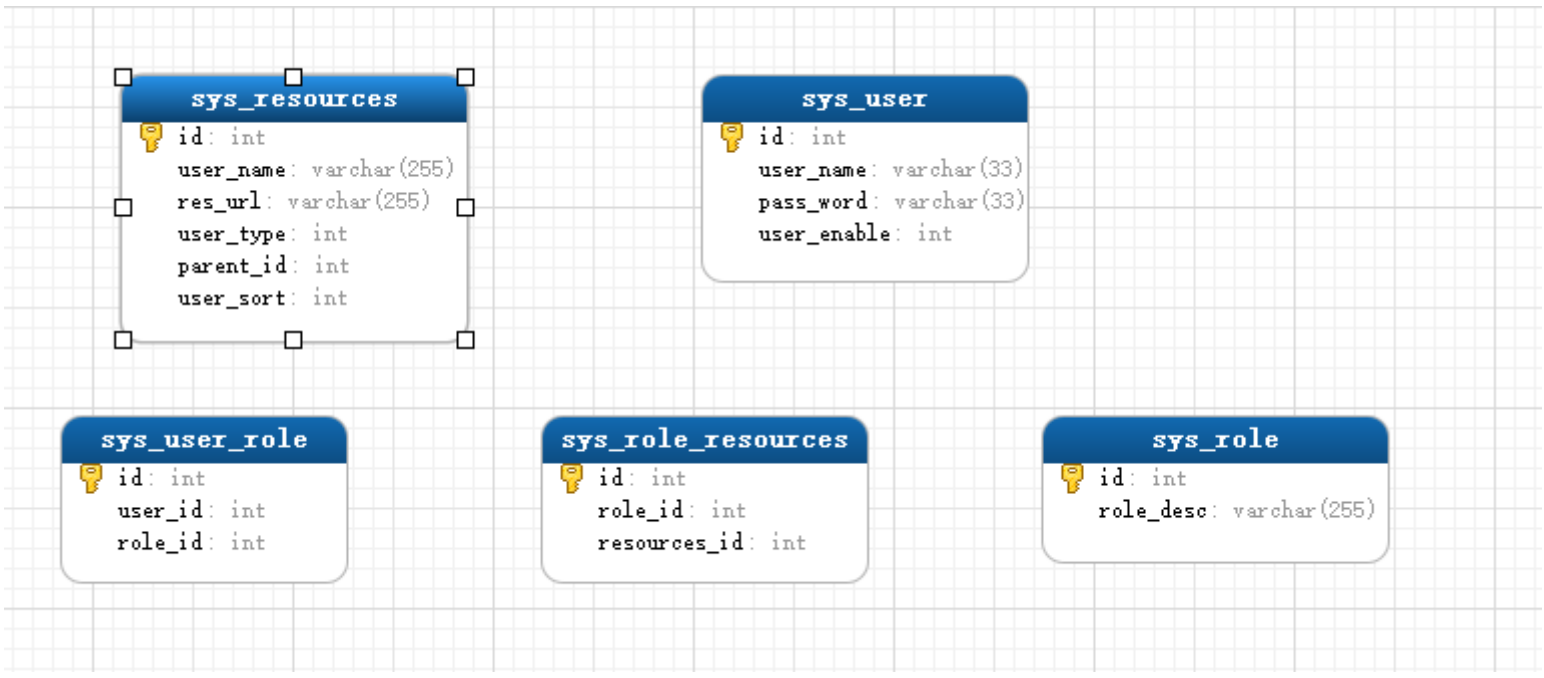
1.当用户没有登陆时只能访问登陆界面

2.当用户登陆成功后，只能访问该用户下仅有的权限

3.一个用户不能两个人同时在线

一、数据库设计

本文的数据库表为5个分别是： 用户表、角色表、权限表、用户角色中间表、角色权限中间表，表的结构和数据项目中会提供（sql和redis工具下方的下载地址中都会有）



二、引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.chaoqi</groupId>
    <artifactId>springboot_mybatisplus</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>springboot_mybatisplus</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.0.RELEASE</version>
        <relativePath/>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
```

```
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- reids -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <!--添加jsp依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
        </dependency>
        <!-- SpringBoot - MyBatis 逆向工程 -->
        <dependency>
            <groupId>org.mybatis.generator</groupId>
            <artifactId>mybatis-generator-core</artifactId>
            <version>1.3.2</version>
        </dependency>
        <!-- MyBatis 通用 Mapper -->
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper-spring-boot-starter</artifactId>
            <version>1.1.4</version>
        </dependency>

        <!-- shiro -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-ehcache</artifactId>
            <version>1.4.0</version>
        </dependency>

        <!-- shiro+redis缓存插件 -->
        <dependency>
            <groupId>org.crazycake</groupId>
            <artifactId>shiro-redis</artifactId>
            <version>2.4.2.1-RELEASE</version>
        </dependency>

        <!-- fastjson阿里巴巴jSON处理器 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.13</version>
        </dependency>
        <!--<dependency>-->
        <!--<groupId>org.springframework.boot</groupId>-->
        <!--<artifactId>spring-boot-starter-security</artifactId>-->
        <!--</dependency>-->

        <!--工具类-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.7</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.mybatis.generator</groupId>
                <artifactId>mybatis-generator-maven-plugin</artifactId>
                <version>1.3.2</version>
                <configuration>
                    <configurationFile>src/main/resources/generatorConfig.xml</configurationFile>
```

```
                <verbose>true</verbose>
                <overwrite>true</overwrite>
            </configuration>
            <executions>
                <execution>
                    <id>Generate MyBatis Artifacts</id>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                </execution>
            </executions>
            <dependencies>
                <dependency>
                    <groupId>org.mybatis.generator</groupId>
                    <artifactId>mybatis-generator-core</artifactId>
                    <version>1.3.2</version>
                </dependency>
                <dependency>
                    <groupId>tk.mybatis</groupId>
                    <artifactId>mapper</artifactId>
                    <version>3.5.0</version>
                </dependency>
            </dependencies>
        </plugin>
    </plugins>
</build>

</project>
```

三、编辑application.yml

```
server:
  port: 8080

spring:
  mvc:
    view:
      prefix: /WEB-INF/jsp/
      suffix: .jsp
  datasource:
    url: jdbc:mysql://localhost:3306/shiro?characterEncoding=UTF-8&useUnicode=true&useSSL=false
    username: root
    password: 123456
    driver-class-name: com.mysql.jdbc.Driver

  redis:
    host: localhost
    port: 6379
    jedis:
      pool:
        max-idle: 8
        min-idle: 0
        max-active: 8
        max-wait: -1
    timeout: 0

mybatis:
  mapper-locations: classpath:mapper/*.xml
  type-aliases-package: com.chaoqi.springboot_mybatisplus.domain
```

四、创建ShiroConfig配置

```
package com.chaoqi.springboot_shiro_redis.config;

import com.chaoqi.springboot_shiro_redis.secutity.KickoutSessionControlFilter;
import com.chaoqi.springboot_shiro_redis.secutity.MyShiroRealm;
import org.apache.shiro.mgt.SecurityManager;
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.web.mgt.DefaultWebSecurityManager;
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.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;
import org.springframework.context.annotation.Bean;
```

```java
import org.springframework.context.annotation.Configuration;

import javax.servlet.Filter;
import java.util.LinkedHashMap;
import java.util.Map;


@Configuration
public class ShiroConfig {

    @Bean
    public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        // 没有登陆的用户只能访问登陆页面
        shiroFilterFactoryBean.setLoginUrl("/auth/login");
        // 登录成功后要跳转的链接
        shiroFilterFactoryBean.setSuccessUrl("/auth/index");
        // 未授权界面; ----这个配置了没卵用，具体原因想深入了解的可以自行百度
        //shiroFilterFactoryBean.setUnauthorizedUrl("/auth/403");
        //自定义拦截器
        Map<String, Filter> filtersMap = new LinkedHashMap<String, Filter>();
        //限制同一帐号同时在线的个数。
        filtersMap.put("kickout", kickoutSessionControlFilter());
        shiroFilterFactoryBean.setFilters(filtersMap);
        // 权限控制map.
        Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();
        filterChainDefinitionMap.put("/css/**", "anon");
        filterChainDefinitionMap.put("/js/**", "anon");
        filterChainDefinitionMap.put("/img/**", "anon");
        filterChainDefinitionMap.put("/auth/login", "anon");
        filterChainDefinitionMap.put("/auth/logout", "logout");
        filterChainDefinitionMap.put("/auth/kickout", "anon");
        filterChainDefinitionMap.put("/**", "authc,kickout");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilterFactoryBean;
    }

    @Bean
    public SecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        // 设置realm.
        securityManager.setRealm(myShiroRealm());
        // 自定义缓存实现 使用redis
        securityManager.setCacheManager(cacheManager());
        // 自定义session管理 使用redis
        securityManager.setSessionManager(sessionManager());
        return securityManager;
    }

    /**
     * 身份认证realm; (这个需要自己写，账号密码校验；权限等)
     *
     * @return
     */
    @Bean
    public MyShiroRealm myShiroRealm() {
        MyShiroRealm myShiroRealm = new MyShiroRealm();
        return myShiroRealm;
    }

    /**
     * cacheManager 缓存 redis实现
     * 使用的是shiro-redis开源插件
     *
     * @return
     */
    public RedisCacheManager cacheManager() {
        RedisCacheManager redisCacheManager = new RedisCacheManager();
        redisCacheManager.setRedisManager(redisManager());
        return redisCacheManager;
    }

    /**
     * 配置shiro redisManager
     * 使用的是shiro-redis开源插件
     *
     * @return
     */
    public RedisManager redisManager() {
        RedisManager redisManager = new RedisManager();
        redisManager.setHost("localhost");
        redisManager.setPort(6379);
```

```
        redisManager.setExpire(1800);// 配置缓存过期时间
        redisManager.setTimeout(0);
        // redisManager.setPassword(password);
        return redisManager;
    }

    /**
     * Session Manager
     * 使用的是shiro-redis开源插件
     */
    @Bean
    public DefaultWebSessionManager sessionManager() {
        DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();
        sessionManager.setSessionDAO(redisSessionDAO());
        return sessionManager;
    }

    /**
     * RedisSessionDAO shiro sessionDao层的实现  通过redis
     * 使用的是shiro-redis开源插件
     */
    @Bean
    public RedisSessionDAO redisSessionDAO() {
        RedisSessionDAO redisSessionDAO = new RedisSessionDAO();
        redisSessionDAO.setRedisManager(redisManager());
        return redisSessionDAO;
    }

    /**
     * 限制同一账号登录同时登录人数控制
     *
     * @return
     */
    @Bean
    public KickoutSessionControlFilter kickoutSessionControlFilter() {
        KickoutSessionControlFilter kickoutSessionControlFilter = new KickoutSessionControlFilter();
        kickoutSessionControlFilter.setCacheManager(cacheManager());
        kickoutSessionControlFilter.setSessionManager(sessionManager());
        kickoutSessionControlFilter.setKickoutAfter(false);
        kickoutSessionControlFilter.setMaxSession(1);
        kickoutSessionControlFilter.setKickoutUrl("/auth/kickout");
        return kickoutSessionControlFilter;
    }


    /***
     * 授权所用配置
     *
     * @return
     */
    @Bean
    public DefaultAdvisorAutoProxyCreator getDefaultAdvisorAutoProxyCreator() {
        DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator = new DefaultAdvisorAutoProxyCreator();
        defaultAdvisorAutoProxyCreator.setProxyTargetClass(true);
        return defaultAdvisorAutoProxyCreator;
    }

    /***
     * 使授权注解起作用不如不想配置可以在pom文件中加入
     * <dependency>
     *<groupId>org.springframework.boot</groupId>
     *<artifactId>spring-boot-starter-aop</artifactId>
     *</dependency>
     * @param securityManager
     * @return
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }

    /**
     * Shiro生命周期处理器
     *
     */
    @Bean
    public LifecycleBeanPostProcessor getLifecycleBeanPostProcessor() {
        return new LifecycleBeanPostProcessor();
    }

}
```

五、自定义Realm

```
package com.chaoqi.springboot_shiro_redis.secutity;

import com.chaoqi.springboot_shiro_redis.service.SysRoleService;
import com.chaoqi.springboot_shiro_redis.service.UserService;
import com.chaoqi.springboot_shiro_redis.dao.domain.SysUser;
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.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.*;

public class MyShiroRealm extends AuthorizingRealm {
    private static org.slf4j.Logger logger = LoggerFactory.getLogger(MyShiroRealm.class);

    //如果项目中用到了事物，@Autowired注解会使事物失效，可以自己用get方法获取值
    @Autowired
    private SysRoleService roleService;
    @Autowired
    private UserService userService;

    /**
     * 认证信息.(身份验证) : Authentication 是用来验证用户身份
     *
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authcToken) throws AuthenticationException {
        logger.info("---------------- 执行 Shiro 凭证认证 ----------------------");
        UsernamePasswordToken token = (UsernamePasswordToken) authcToken;
        String name = token.getUsername();
        String password = String.valueOf(token.getPassword());
        SysUser user = new SysUser();
        user.setUserName(name);
        user.setPassWord(password);
        // 从数据库获取对应用户名密码的用户
        SysUser userList = userService.getUser(user);
        if (userList != null) {
            // 用户为禁用状态
            if (userList.getUserEnable() != 1) {
                throw new DisabledAccountException();
            }
            logger.info("---------------- Shiro 凭证认证成功 ----------------------");
            SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(
                    userList, //用户
                    userList.getPassWord(), //密码
                    getName()  //realm name
            );
            return authenticationInfo;
        }
        throw new UnknownAccountException();
    }

    /**
     * 授权
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        logger.info("---------------- 执行 Shiro 权限获取 ----------------------");
        Object principal = principals.getPrimaryPrincipal();
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        if (principal instanceof SysUser) {
            SysUser userLogin = (SysUser) principal;
            Set<String> roles = roleService.findRoleNameByUserId(userLogin.getId());
            authorizationInfo.addRoles(roles);

            Set<String> permissions = userService.findPermissionsByUserId(userLogin.getId());
            authorizationInfo.addStringPermissions(permissions);
        }
        logger.info("---- 获取到以下权限 ----");
        logger.info(authorizationInfo.getStringPermissions().toString());
        logger.info("---------------- Shiro 权限获取成功 ----------------------");
        return authorizationInfo;
    }
```

```
}
```

六、限制并发人数登陆

```
package com.chaoqi.springboot_shiro_redis.secutity;

import com.alibaba.fastjson.JSON;
import com.chaoqi.springboot_shiro_redis.dao.domain.SysUser;
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheManager;
import org.apache.shiro.session.Session;
import org.apache.shiro.session.mgt.DefaultSessionKey;
import org.apache.shiro.session.mgt.SessionManager;
import org.apache.shiro.subject.Subject;
import org.apache.shiro.web.filter.AccessControlFilter;
import org.apache.shiro.web.util.WebUtils;

import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Serializable;
import java.util.Deque;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

public class KickoutSessionControlFilter extends AccessControlFilter {

    private String kickoutUrl; //踢出后到的地址
    private boolean kickoutAfter = false; //踢出之前登录的/之后登录的用户 默认踢出之前登录的用户
    private int maxSession = 1; //同一个帐号最大会话数 默认1

    private SessionManager sessionManager;
    private Cache<String, Deque<Serializable>> cache;

    public void setKickoutUrl(String kickoutUrl) {
        this.kickoutUrl = kickoutUrl;
    }

    public void setKickoutAfter(boolean kickoutAfter) {
        this.kickoutAfter = kickoutAfter;
    }

    public void setMaxSession(int maxSession) {
        this.maxSession = maxSession;
    }

    public void setSessionManager(SessionManager sessionManager) {
        this.sessionManager = sessionManager;
    }
    //设置Cache的key的前缀
    public void setCacheManager(CacheManager cacheManager) {
        this.cache = cacheManager.getCache("shiro_redis_cache");
    }

    @Override
    protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception {
        return false;
    }

    @Override
    protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
        Subject subject = getSubject(request, response);
        if(!subject.isAuthenticated() && !subject.isRemembered()) {
            //如果没有登录，直接进行之后的流程
            return true;
        }


        Session session = subject.getSession();
        SysUser user = (SysUser) subject.getPrincipal();
        String username = user.getUserName();
        Serializable sessionId = session.getId();

        //读取缓存    没有就存入
        Deque<Serializable> deque = cache.get(username);

        //如果此用户没有session队列，也就是还没有登录过，缓存中没有
```

```
//就new一个空队列，不然deque对象为空，会报空指针
if(deque==null){
    deque = new LinkedList<Serializable>();
}

//如果队列里没有此sessionId，且用户没有被踢出；放入队列
if(!deque.contains(sessionId) && session.getAttribute("kickout") == null) {
    //将sessionId存入队列
    deque.push(sessionId);
    //将用户的sessionId队列缓存
    cache.put(username, deque);
}

//如果队列里的sessionId数超出最大会话数，开始踢人
while(deque.size() > maxSession) {
    Serializable kickoutSessionId = null;
    if(kickoutAfter) { //如果踢出后者
        kickoutSessionId = deque.removeFirst();
        //踢出后再更新下缓存队列
        cache.put(username, deque);
    } else { //否则踢出前者
        kickoutSessionId = deque.removeLast();
        //踢出后再更新下缓存队列
        cache.put(username, deque);
    }

    try {
        //获取被踢出的sessionId的session对象
        Session kickoutSession = sessionManager.getSession(new DefaultSessionKey(kickoutSessionId));
        if(kickoutSession != null) {
            //设置会话的kickout属性表示踢出了
            kickoutSession.setAttribute("kickout", true);
        }
    } catch (Exception e) {//ignore exception
    }
}

//如果被踢出了，直接退出，重定向到踢出后的地址
if (session.getAttribute("kickout") != null) {
    //会话被踢出了
    try {
        //退出登录
        subject.logout();
    } catch (Exception e) { //ignore
    }
    saveRequest(request);

    Map<String, String> resultMap = new HashMap<String, String>();
    //判断是不是Ajax请求
    if ("XMLHttpRequest".equalsIgnoreCase(((HttpServletRequest) request).getHeader("X-Requested-With"))) {
        resultMap.put("user_status", "300");
        resultMap.put("message", "您已经在其他地方登录，请重新登录！");
        //输出json串
        out(response, resultMap);
    }else{
            //重定向
            WebUtils.issueRedirect(request, response, kickoutUrl);
        }
        return false;
    }
    return true;
}
private void out(ServletResponse hresponse, Map<String, String> resultMap)
        throws IOException {
    try {
        hresponse.setCharacterEncoding("UTF-8");
        PrintWriter out = hresponse.getWriter();
        out.println(JSON.toJSONString(resultMap));
        out.flush();
        out.close();
    } catch (Exception e) {
        System.err.println("KickoutSessionFilter.class 输出JSON异常，可以忽略。");
    }
}
}
```

七、异常处理类，拦截未授权页面（未授权页面有三种实现方式，我这里使用异常处理）

```java
package com.chaoqi.springboot_shiro_redis.exception;

import org.apache.shiro.authz.AuthorizationException;
import org.apache.shiro.authz.UnauthorizedException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;

/**
 * 全局异常处理类
 */
@ControllerAdvice
public class CtrlExceptionHandler {
    private static Logger logger = LoggerFactory.getLogger(CtrlExceptionHandler.class);

    //拦截未授权页面
    @ResponseStatus(value = HttpStatus.FORBIDDEN)
    @ExceptionHandler(UnauthorizedException.class)
    public String handleException(UnauthorizedException e) {
        logger.debug(e.getMessage());
        return "403";
    }

    @ResponseStatus(value = HttpStatus.FORBIDDEN)
    @ExceptionHandler(AuthorizationException.class)
    public String handleException2(AuthorizationException e) {
        logger.debug(e.getMessage());
        return "403";
    }
}
```

八、最后附上logincontroller的代码，调用login就可以调到登陆页面

```java
package com.chaoqi.springboot_shiro_redis.web;

import com.chaoqi.springboot_shiro_redis.dao.domain.SysUser;
import com.chaoqi.springboot_shiro_redis.utils.RequestUtils;
import org.apache.commons.lang3.StringUtils;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.DisabledAccountException;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;


@Controller
@RequestMapping(value = "/auth")
public class LoginController {

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public String submitLogin(String username, String password, HttpServletRequest request) {
        try {
            UsernamePasswordToken token = new UsernamePasswordToken(username, password);
            Subject subject = SecurityUtils.getSubject();
            subject.login(token);
            SysUser user = (SysUser) subject.getPrincipal();
        } catch (DisabledAccountException e) {
            request.setAttribute("msg", "账户已被禁用");
            return "login";
        } catch (AuthenticationException e) {
            request.setAttribute("msg", "用户名或密码错误");
            return "login";
        }

        // 执行到这里说明用户已登录成功
        return "redirect:/auth/index";
    }


    @RequestMapping(value = "/login", method = RequestMethod.GET)
```

```
    public String loginPage() {
        return "login";
    }

    @RequestMapping(value = "/index", method = RequestMethod.GET)
    public String loginSuccessMessage(HttpServletRequest request) {
        String username = "未登录";
        SysUser currentLoginUser = RequestUtils.currentLoginUser();

        if (currentLoginUser != null && StringUtils.isNotEmpty(currentLoginUser.getUserName())) {
            username = currentLoginUser.getUserName();
        } else {
            return "redirect:/auth/login";
        }
        request.setAttribute("username", username);
        return "index";
    }

    //被踢出后跳转的页面
    @RequestMapping(value = "/kickout", method = RequestMethod.GET)
    public String kickOut() {
        return "kickout";
    }
}
```

至此shiro整合完成，源码下载地址为：https://github.com/caicahoqi/ChaoqiIsPrivateLibrary