Springboot shiro 项目心得文档

# 创建项目

1. 创建数据库my\_world      表名分别为t\_base\_user，t\_base\_user\_role，t\_base\_role，t\_base\_menu\_role，t\_base\_menu
2. 通过mybatis plus自定义生成java代码（该工具完整版参考[https://github.com/admincjy/cjygenerator）](https://github.com/admincjy/cjygenerator%EF%BC%89)
3. 创建springboot项目并引入相关jar包和代码（项目地址 [https://github.com/admincjy/shiro](http://xn--ces6a538pmb2a/)）
4. 注意：分页差距的依赖包需要引三个, 还需引入配置文件mybatis-config.xml(具体参考项目)                                                                             <!-- 分页依赖(引三个) -->  
   <dependency>  
   <groupId>com.github.pagehelper</groupId>  
   <artifactId>pagehelper</artifactId>  
   <version>5.0.1</version>  
   </dependency>  
   <dependency>  
   <groupId>com.github.pagehelper</groupId>  
   <artifactId>pagehelper-spring-boot-autoconfigure</artifactId>  
   <version>1.2.5</version>  
   </dependency>  
   <dependency>  
   <groupId>com.github.pagehelper</groupId>  
   <artifactId>pagehelper-spring-boot-starter</artifactId>  
   <version>1.2.5</version>  
   </dependency>

引入shiro的jar包  
<dependency>  
<groupId>org.apache.shiro</groupId>  
<artifactId>shiro-spring</artifactId>  
<version>1.4.0</version>  
</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.4.0</version>

</dependency>

引入其他基本包

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

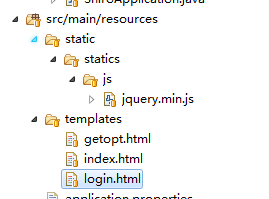
 <!-- mybatis两个包，不然用不了分页 -->  
<dependency>  
<groupId>com.baomidou</groupId>  
<artifactId>mybatis-plus-boot-starter</artifactId>  
<version>3.0.6</version>  
</dependency>  
<dependency>  
<groupId>com.alibaba</groupId>  
<artifactId>druid</artifactId>  
<version>1.1.8</version>  
</dependency>

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

# 实现认证

**将静态html和js文件放入项目中，如图**：



**生成页面跳转类：（注意:页面跳转注解是@Controller不是@RestController）**

@Controller

@RequestMapping("/login")

public class LoginController {

@RequestMapping("/html")

public String loginHtml(){

return "/login";

}

@RequestMapping("/index")

public String indexHtml(){

return "/index";

}

@RequestMapping("/getopt")

public String getoptHtml(){

return "/getopt";

}

}

**自定义realm,这里我定义为MyShiroRealm:（注意:需配置config,不然数据库查询user无法查询要报异常）**

public class MyShiroRealm extends AuthorizingRealm{

@Autowired

private UserMapper userMapper;

/\*

\* 认证（非 Javadoc）

\* @see org.apache.shiro.realm.AuthorizingRealm#doGetAuthorizationInfo(org.apache.shiro.subject.PrincipalCollection)

\*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

String username = (String)principals.getPrimaryPrincipal();

Set<String> roles = userMapper.findRoles(username);

Set<String> permissions = userMapper.findRoles(username);

SimpleAuthorizationInfo simpleAuthorizationInfo=new SimpleAuthorizationInfo();

simpleAuthorizationInfo.setRoles(roles);

simpleAuthorizationInfo.setStringPermissions(permissions);

return simpleAuthorizationInfo;

}

/\*

\* 认证（非 Javadoc）

\* @see org.apache.shiro.realm.AuthenticatingRealm#doGetAuthenticationInfo(org.apache.shiro.authc.AuthenticationToken)

\*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

//从主体传过来的信息中获取用户的输入的账号.

UsernamePasswordToken tokens = (UsernamePasswordToken) token;

String username = tokens.getUsername();

//通过username从数据库中查找 User对象

//实际项目中，这里可以根据实际情况做缓存，如果不做，Shiro自己也是有时间间隔机制，2分钟内不会重复执行该方法

User user = this.findByUsername(username);

//创建返回对象

SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(

username, //用户名

user.getPassword(), //密码

、、//ByteSource.Util.bytes(user.getUsername()+user.getSalt()),//salt=username+salt

getName() //realm name

);

return authenticationInfo;

}

public User findByUsername(String username){

User user = userMapper.findByUsername(username);

if(user == null){

return null;

}

return user;

}

}

**新建config类，我这取名为ShiroConfig：**

@Configuration

public class ShiroConfig {

/\*\*

\* 密码校验规则HashedCredentialsMatcher

\* 这个类是为了对密码进行编码的 ,

\* 防止密码在数据库里明码保存 , 当然在登陆认证的时候 ,

\* 这个类也负责对form里输入的密码进行编码

\* 处理认证匹配处理器：如果自定义需要实现继承HashedCredentialsMatcher

\*/

@Bean("hashedCredentialsMatcher")

public HashedCredentialsMatcher hashedCredentialsMatcher() {

HashedCredentialsMatcher credentialsMatcher = new HashedCredentialsMatcher();

//指定加密方式为MD5

credentialsMatcher.setHashAlgorithmName("MD5");

//加密次数

credentialsMatcher.setHashIterations(1);

credentialsMatcher.setStoredCredentialsHexEncoded(true);

return credentialsMatcher;

}

@Bean("authRealm")

@DependsOn("lifecycleBeanPostProcessor")//可选

public MyShiroRealm authRealm(@Qualifier("hashedCredentialsMatcher") HashedCredentialsMatcher matcher) {

MyShiroRealm authRealm = new MyShiroRealm();

authRealm.setAuthorizationCachingEnabled(false);

authRealm.setCredentialsMatcher(matcher);

return authRealm;

}

/\*\*

\* 定义安全管理器securityManager,注入自定义的realm

\* @param authRealm

\* @return

\*/

@Bean("securityManager")

public SecurityManager securityManager(@Qualifier("authRealm") MyShiroRealm authRealm) {

DefaultWebSecurityManager manager = new DefaultWebSecurityManager();

manager.setRealm(authRealm);

return manager;

}

/\*\*

\* 定义shiroFilter过滤器并注入securityManager

\* @param manager

\* @return

\*/

@Bean("shiroFilter")

public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager manager) {

ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();

//设置securityManager

bean.setSecurityManager(manager);

//设置登录页面

//可以写路由也可以写jsp页面的访问路径

bean.setLoginUrl("login/html");

//设置登录成功跳转的页面

bean.setSuccessUrl("/login/index");

//设置未授权跳转的页面

bean.setUnauthorizedUrl("/login/getopt");

//定义过滤器

LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();

filterChainDefinitionMap.put("/login/index", "authc");

filterChainDefinitionMap.put("/login/html", "anon");

filterChainDefinitionMap.put("/user/login", "anon");

// filterChainDefinitionMap.put("/admin", "roles[admin]");

// filterChainDefinitionMap.put("/edit", "perms[delete]");

// filterChainDefinitionMap.put("/druid/\*\*", "anon");

filterChainDefinitionMap.put("/statics/\*\*", "anon");

//需要登录访问的资源 , 一般将/\*\*放在最下边

filterChainDefinitionMap.put("/\*\*", "authc");

bean.setFilterChainDefinitionMap(filterChainDefinitionMap);

return bean;

}

/\*\*

\* Spring的一个bean , 由Advisor决定对哪些类的方法进行AOP代理 .

\* @return

\*/

@Bean

public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {

DefaultAdvisorAutoProxyCreator creator = new DefaultAdvisorAutoProxyCreator();

creator.setProxyTargetClass(true);

return creator;

}

/\*\*

\* 配置shiro跟spring的关联

\* @param securityManager

\* @return

\*/

@Bean

public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(@Qualifier("securityManager") SecurityManager securityManager) {

AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

return advisor;

}

/\*\*

\* lifecycleBeanPostProcessor是负责生命周期的 , 初始化和销毁的类

\* (可选)

\*/

@Bean("lifecycleBeanPostProcessor")

public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {

return new LifecycleBeanPostProcessor();

}

}

这里的注意事项：

SecurityManager 引包必须是import org.apache.shiro.mgt. SecurityManager;

credentialsMatcher.setHashIterations(1);只加密一次就设置为1，根据加密次数写

filterChainDefinitionMap.put("/statics/\*\*", "anon");静态js，css等文件放在Static下面新建的statics文件夹下，不要直接放在static下面，不然无法过滤

**在UserController里面创建注册和登陆的方法：**

/\*

\* 登陆

\*/

@RequestMapping("/login")

public BaseRespMsg login(String username,String password){

//把前端输入的username和password封装为token

UsernamePasswordToken token = new UsernamePasswordToken(username, password);

Subject subject = SecurityUtils.getSubject();

subject.login(token);

System.out.println("是否登录:"+subject.isAuthenticated());

System.out.println("用户权限是否有VIP权限:"+subject.hasRole("vip"));

if(subject.isAuthenticated()==true){

return new BaseRespMsg(0,"登陆成功");

}

new BaseRespMsg();

return BaseRespMsg.error("登陆失败");

}

/\*

\* 注册

\*/

@RequestMapping("/addUser")

public BaseRespMsg addUser(String username,String password){

Md5Hash md5Hash=new Md5Hash(password);

User user=new User();

user.setUsername(username);

user.setPassword(md5Hash.toString());

user.setState(0);

user.setSalt("cjy");

int isOk = targetService.insert(user);

if(isOk==1){

return new BaseRespMsg(0,"添加成功");

}

return BaseRespMsg.error("添加失败");

}

# 加密加盐

**加盐处理首先在注册和添加用户的时候要对密码进行处理：这里加盐是用的用户名（可以自定，但是用UUID不行，不知道为什么）**

/\*  
 \* 注册  
 \*/  
@RequestMapping("/addUser")  
public BaseRespMsg addUser(String username,String password){  
 User user=new User();  
 user.setUsername(username);  
 user.setPassword(this.makePasswordHasSalt(username,password).toString());  
 user.setState(0);  
 user.setSalt(username);  
 int isOk = targetService.insert(user);  
 if(isOk==1){  
 return new BaseRespMsg(0,"添加成功");  
 }  
 return BaseRespMsg.*error*("添加失败");  
 }  
 /

\*这个是重写shiro给密码加盐的方法从而得到加过盐的密码 SimpleHash的四个参数分别为：加密方式，密码，盐值，加密次数

/  
 public Object makePasswordHasSalt(String username,String password){  
 ByteSource salt = ByteSource.Util.*bytes*(username);  
 Object result = new SimpleHash("MD5",password,salt,1);  
 return result;  
 }

**然后修改自定义realm（MyShiroRealm）的登陆验证方法，在SimpleAuthenticationInfo方法里面加入加盐的参数如**：

authenticationInfo = new SimpleAuthenticationInfo(  
 username, //用户名  
 user.getPassword(), //密码  
 ByteSource.Util.*bytes*(user.getSalt()),//salt=salt  
 getName() //realm name  
 );

# 自定义登陆验证

**创建自定义密码验证登陆类（CustomEncryption）继承HashedCredentialsMatcher类,这里的缓存时间设置为10分钟，最多密码输入错误三次就不让其再登陆**

package com.cjy.shiro.utils;  
  
import org.apache.shiro.authc.AuthenticationInfo;  
import org.apache.shiro.authc.AuthenticationToken;  
import org.apache.shiro.authc.ExcessiveAttemptsException;  
import org.apache.shiro.authc.IncorrectCredentialsException;  
import org.apache.shiro.authc.credential.HashedCredentialsMatcher;  
import org.apache.shiro.cache.Cache;  
import org.apache.shiro.cache.CacheManager;  
  
/\*  
 \* 自定义登陆验密  
 \* cjy  
 \*/  
public class CustomEncryption extends HashedCredentialsMatcher{  
   
 private Cache<String, Integer> cache;  
   
 */\*\*  
 \* 自定义密码错误上限  
 \*/* private Integer retryMax;  
  
  
  
 /\*  
 \* CacheManager缓存管理器  
 \*/  
 public CustomEncryption(CacheManager cacheManager) {  
 cache = cacheManager.getCache("passwordRetryCache");  
 }  
   
 @Override  
 public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) throws ExcessiveAttemptsException {  
 String username = (String)token.getPrincipal();  
 Integer retryCount = cache.get(username);  
 if(retryCount == null) {  
 retryCount = new Integer(1);  
 cache.put(username, retryCount);  
 }  
   
   
 if(retryCount > retryMax) {  
 throw new ExcessiveAttemptsException("您已连续错误达" + retryMax + "次！请10分钟后再试");  
 }  
   
 if(cache.getClass().getName().contains("Cache")){  
 cache.put(username, ++retryCount);  
 }  
 //调用父类的校验方法  
 boolean matches = super.doCredentialsMatch(token, info);  
 if(matches) {  
 cache.remove(username);  
 }else {  
  
 throw new IncorrectCredentialsException("您输入的密码已错误" + (retryCount-1) + "次，最多错误" + retryMax + "次");  
 }  
 return true;  
 }  
  
  
 public void setRetryMax(Integer retryMax) {  
 this.retryMax = retryMax;  
 }  
  
 public Integer getRetryMax() {  
 return retryMax;  
 }  
}

**新增一个缓冲处理器的ehcache-shiro.xml文件出来缓冲：**

<?xml version="1.0" encoding="UTF-8"?>  
<ehcache name="es">  
  
 <diskStore path="java.io.tmpdir"/>  
  
 <!--  
 name:缓存名称。  
 maxElementsInMemory:缓存最大数目  
 maxElementsOnDisk：硬盘最大缓存个数。  
 eternal:对象是否永久有效，一但设置了，timeout将不起作用。  
 overflowToDisk:是否保存到磁盘，当系统当机时  
 timeToIdleSeconds:设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。  
 timeToLiveSeconds:设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。  
 diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.  
 diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。  
 diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。  
 memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。  
 clearOnFlush：内存数量最大时是否清除。  
 memoryStoreEvictionPolicy:  
 Ehcache的三种清空策略;  
 FIFO，first in first out，这个是大家最熟的，先进先出。  
 LFU， Less Frequently Used，就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。  
 LRU，Least Recently Used，最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。  
 -->  
 <defaultCache  
 maxElementsInMemory="10000"  
 eternal="false"  
 timeToIdleSeconds="120"  
 timeToLiveSeconds="120"  
 overflowToDisk="false"  
 diskPersistent="false"  
 diskExpiryThreadIntervalSeconds="120"  
 />  
  
  
 <!-- 登录记录缓存锁定10分钟 -->  
 <cache name="passwordRetryCache"  
 maxEntriesLocalHeap="2000"  
 eternal="false"  
 timeToIdleSeconds="3600"  
 timeToLiveSeconds="0"  
 overflowToDisk="false"  
 statistics="true">  
 </cache>  
  
</ehcache>

**修改ShrioConfig的实现方法，主要是修改之前的HashedCredentialsMatcher（），securityManager（），myShiroRealm（）方法，并新添加一个缓冲处理器方法ehCacheManager（）**

package com.cjy.shiro.config;  
  
import com.cjy.shiro.utils.CustomEncryption;  
import com.cjy.shiro.utils.MyShiroRealm;  
import org.apache.shiro.authc.credential.CredentialsMatcher;  
import org.apache.shiro.cache.CacheManager;  
import org.apache.shiro.cache.ehcache.EhCacheManager;  
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.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.DependsOn;  
  
import java.util.LinkedHashMap;  
  
@Configuration  
public class ShiroConfig {  
 */\*\*  
 \* 密码校验规则HashedCredentialsMatcher  
 \* 这个类是为了对密码进行编码的 ,  
 \* 防止密码在数据库里明码保存 , 当然在登陆认证的时候 ,  
 \* 这个类也负责对form里输入的密码进行编码  
 \* 处理认证匹配处理器：如果自定义需要实现继承HashedCredentialsMatcher  
 \*/* @Bean  
 public CredentialsMatcher credentialsMatcher(CacheManager cacheManager) {  
 CustomEncryption customEncryption = new CustomEncryption(cacheManager);  
 //指定加密方式为MD5  
 customEncryption.setHashAlgorithmName("MD5");  
 //加密次数  
 customEncryption.setHashIterations(1);  
 customEncryption.setStoredCredentialsHexEncoded(true);  
 //密码最对输入错误多少次  
 customEncryption.setRetryMax(3);  
 return customEncryption;  
 }  
  
 */\*\*  
 \* shiro缓存管理器;  
 \* 需要注入对应的其它的实体类中：  
 \* 1、安全管理器：securityManager  
 \* 可见securityManager是整个shiro的核心；  
 \** ***@return*** *\*/* @Bean  
 public EhCacheManager ehCacheManager(){  
 System.*out*.println("ShiroConfiguration.getEhCacheManager()");  
 EhCacheManager cacheManager = new EhCacheManager();  
 cacheManager.setCacheManagerConfigFile("classpath:ehcache-shiro.xml");  
 return cacheManager;  
 }  
  
  
 @Bean  
 @DependsOn("lifecycleBeanPostProcessor")//可选  
 public MyShiroRealm myShiroRealm(CredentialsMatcher credentialsMatcher) {  
 MyShiroRealm myShiroRealm = new MyShiroRealm();  
// myShiroRealm.setAuthorizationCachingEnabled(false);  
 myShiroRealm.setCredentialsMatcher(credentialsMatcher);  
 return myShiroRealm;  
 }  
  
  
 */\*\*  
 \* 定义安全管理器securityManager,注入自定义的realm  
 \** ***@param*** *credentialsMatcher  
 \** ***@return*** *\*/* @Bean  
 public SecurityManager securityManager(CredentialsMatcher credentialsMatcher) {  
 DefaultWebSecurityManager manager = new DefaultWebSecurityManager();  
 manager.setRealm(myShiroRealm(credentialsMatcher));  
 //注入缓存管理器;  
 manager.setCacheManager(ehCacheManager());  
 return manager;  
 }  
  
  
 */\*\*  
 \* 定义shiroFilter过滤器并注入securityManager  
 \** ***@param*** *manager  
 \** ***@return*** *\*/* @Bean  
 public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager manager) {  
 ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();  
 //设置securityManager  
 bean.setSecurityManager(manager);  
 //设置登录页面  
 //可以写路由也可以写jsp页面的访问路径  
 bean.setLoginUrl("login/html");  
 //设置登录成功跳转的页面  
 bean.setSuccessUrl("/login/index");  
 //设置未授权跳转的页面  
 bean.setUnauthorizedUrl("/login/getopt");  
 //定义过滤器  
 LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();  
  
 filterChainDefinitionMap.put("/login/index", "authc");  
 filterChainDefinitionMap.put("/login/html", "anon");  
 filterChainDefinitionMap.put("/user/login", "anon");  
 filterChainDefinitionMap.put("/login/getopt", "anon");  
 filterChainDefinitionMap.put("/user/addUser", "anon");  
// filterChainDefinitionMap.put("/admin", "roles[admin]");  
// filterChainDefinitionMap.put("/edit", "perms[delete]");  
// filterChainDefinitionMap.put("/druid/\*\*", "anon");  
 filterChainDefinitionMap.put("/statics/\*\*", "anon");  
 //需要登录访问的资源 , 一般将/\*\*放在最下边  
 filterChainDefinitionMap.put("/\*\*", "authc");  
 bean.setFilterChainDefinitionMap(filterChainDefinitionMap);  
 return bean;  
 }  
  
 */\*\*  
 \* Spring的一个bean , 由Advisor决定对哪些类的方法进行AOP代理 .  
 \** ***@return*** *\*/* @Bean  
 public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {  
 DefaultAdvisorAutoProxyCreator creator = new DefaultAdvisorAutoProxyCreator();  
 creator.setProxyTargetClass(true);  
 return creator;  
 }  
  
 */\*\*  
 \* 配置shiro跟spring的关联  
 \** ***@param*** *securityManager  
 \** ***@return*** *\*/* @Bean  
 public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(@Qualifier("securityManager") SecurityManager securityManager) {  
 AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();  
 advisor.setSecurityManager(securityManager);  
 return advisor;  
 }  
  
 */\*\*  
 \* lifecycleBeanPostProcessor是负责生命周期的 , 初始化和销毁的类  
 \* (可选)  
 \*/* @Bean  
 public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {  
 return new LifecycleBeanPostProcessor();  
 }  
  
}

# 阶段性总结

**学习文档http://www.cnblogs.com/learnhow/p/5694876.html**

**Shiro的核心有SecurityManager，它负责安全认证与授权 Realm，区域可自定义，继承AuthorizingRealm config配置类的方法都要@bean**  **继承类重写方法时多看源码**

# Redis

**Redis 应用场景：缓冲，任务队列，应用排行榜，数据过程处理，网站访问统计，分布式集群架构中的session分离。**

**Jedis是redis的java客户端的开发包**

## Redis实现session管理

### 引入jedis包依赖

<dependency>  
 <groupId>redis.clients</groupId>  
 <artifactId>jedis</artifactId>  
 <version>2.8.0</version>  
</dependency>

### 创建RedisConfig连接配置类

@Configuration  
@EnableAutoConfiguration  
public class RedisConfig {  
  
 private Logger logger = LoggerFactory.*getLogger*(RedisConfig.class);  
  
 */\*\*  
 \* SpringSession 需要注意的就是redis需要2.8以上版本，然后开启事件通知，在redis配置文件里面加上  
 \* notify-keyspace-events Ex  
 \* Keyspace notifications功能默认是关闭的（默认地，Keyspace 时间通知功能是禁用的，因为它或多或少会使用一些CPU的资源）。  
 \* 或是使用如下命令：  
 \* redis-cli config set notify-keyspace-events Egx  
 \* 如果你的Redis不是你自己维护的，比如你是使用阿里云的Redis数据库，你不能够更改它的配置，那么可以使用如下方法：在applicationContext.xml中配置  
 \* <util:constant static-field="org.springframework.session.data.redis.config.ConfigureRedisAction.NO\_OP"/>  
 \** ***@return*** *\*/* @Value("${spring.redis.host}")  
 private String host;  
  
 @Value("${spring.redis.port}")  
 private int port;  
  
 @Value("${spring.redis.timeout}")  
 private int timeout;  
  
 @Value("${spring.redis.jedis.pool.max-active}")  
 private int maxActive;  
  
 @Value("${spring.redis.jedis.pool.max-idle}")  
 private int maxIdle;  
  
 @Value("${spring.redis.jedis.pool.min-idle}")  
 private int minIdle;  
  
 @Value("${spring.redis.jedis.pool.max-wait}")  
 private long maxWaitMillis;  
 @Bean  
 public JedisPool redisPoolFactory(){  
 JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();  
 jedisPoolConfig.setMaxIdle(maxIdle);  
 jedisPoolConfig.setMaxWaitMillis(maxWaitMillis);  
 jedisPoolConfig.setMaxTotal(maxActive);  
 jedisPoolConfig.setMinIdle(minIdle);  
 JedisPool jedisPool = new JedisPool(jedisPoolConfig,host,port,timeout,null);  
 logger.info("JedisPool注入成功！");  
 logger.info("redis地址：" + host + ":" + port);  
 return jedisPool;  
 }  
}

### 创建JedisUtil工具类使用redis

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
import redis.clients.jedis.Jedis;  
import redis.clients.jedis.JedisPool;  
  
import java.util.Set;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/25.  
 \*/*@Component  
public class JedisUtil {  
  
 @Autowired  
 private JedisPool jedisPool;  
  
 private JedisUtil() {  
 }  
  
 public void set(byte[] key, byte[] value) {  
 try (Jedis jedis = getResource()) {  
 jedis.set(key, value);  
 }  
 }  
  
 public void expire(byte[] key, int expireTime) {  
 try (Jedis jedis = getResource()) {  
 jedis.expire(key, expireTime);  
 }  
 }  
  
 private Jedis getResource() {  
 return jedisPool.getResource();  
 }  
  
 public byte[] get(byte[] key) {  
 try (Jedis jedis = getResource()) {  
 return jedis.get(key);  
 }  
 }  
  
 public void delete(byte[] key) {  
 try (Jedis jedis = getResource()) {  
 jedis.del(key);  
 }  
 }  
  
 public Set<byte[]> keys(String prefix) {  
 try (Jedis jedis = getResource()) {  
 return jedis.keys((prefix + "\*").getBytes());  
 }  
 }  
}

### 创建RedisSessionDao的dao重写起自带的sessionDao

package com.cjy.buy.untils;  
  
import org.apache.commons.collections.CollectionUtils;  
import org.apache.shiro.session.Session;  
import org.apache.shiro.session.UnknownSessionException;  
import org.apache.shiro.session.mgt.eis.AbstractSessionDAO;  
import org.springframework.util.SerializationUtils;  
  
import javax.annotation.Resource;  
import java.io.Serializable;  
import java.util.Collection;  
import java.util.HashSet;  
import java.util.Set;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/25.  
 \*/*public class RedisSessionDao extends AbstractSessionDAO {  
  
 @Resource  
 private JedisUtil jedisUtil;  
  
 private final String SHIRO\_SESSION\_PREFIX = "redis-session:";  
  
  
 @Override  
 protected Serializable doCreate(Session session) {  
  
 Serializable sessionId = generateSessionId(session);  
 //将session和sessionId捆绑  
 assignSessionId(session, sessionId);  
 saveSession(session);  
  
 return sessionId;  
 }  
  
 @Override  
 protected Session doReadSession(Serializable sessionId) {  
  
 System.*out*.println("read session");  
 if (sessionId == null) {  
 return null;  
 }  
 byte[] key = getKey(sessionId.toString());  
 byte[] value = jedisUtil.get(key);  
  
 return (Session) SerializationUtils.*deserialize*(value);  
 }  
  
 @Override  
 public void update(Session session) throws UnknownSessionException {  
  
 if (session != null && session.getId() != null) {  
 saveSession(session);  
 }  
 }  
  
 @Override  
 public void delete(Session session) {  
  
 if (session == null || session.getId() == null) {  
 return;  
 }  
  
 byte[] key = getKey(session.getId().toString());  
 jedisUtil.delete(key);  
 }  
  
 @Override  
 public Collection<Session> getActiveSessions() {  
  
 Set<byte[]> keys = jedisUtil.keys(SHIRO\_SESSION\_PREFIX);  
 Set<Session> sessions = new HashSet<>();  
 if (CollectionUtils.*isEmpty*(keys)) {  
 return sessions;  
 }  
  
 for (byte[] key : keys) {  
 Session session = (Session) SerializationUtils.*deserialize*(jedisUtil.get(key));  
 sessions.add(session);  
 }  
  
 return sessions;  
 }  
  
 private void saveSession(Session session) {  
  
 if (session != null && session.getId() != null) {  
 byte[] key = getKey(session.getId().toString());  
 byte[] value = SerializationUtils.*serialize*(session);  
  
 jedisUtil.set(key, value);  
 jedisUtil.expire(key, 600);  
 }  
 }  
  
 private byte[] getKey(String key) {  
  
 return (SHIRO\_SESSION\_PREFIX + key).getBytes();  
 }  
  
}

### 创建RedisSessionManager重写原来的SessionManager

在这里解决多次访问sessionManger的问题，每次取session最先在redis访问一次，以后都从request里面去访问

package com.cjy.buy.manager;  
  
import org.apache.shiro.session.Session;  
import org.apache.shiro.session.UnknownSessionException;  
import org.apache.shiro.session.mgt.SessionKey;  
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;  
import org.apache.shiro.web.session.mgt.WebSessionKey;  
  
import javax.servlet.ServletRequest;  
import java.io.Serializable;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/25.  
 \* 解决多次访问sessionManger的问题  
 \*/*public class RedisSessionManager extends DefaultWebSessionManager {  
 @Override  
 protected Session retrieveSession(SessionKey sessionKey) throws UnknownSessionException {  
  
 Serializable sessionId = getSessionId(sessionKey);  
 ServletRequest request = null;  
 if (sessionKey instanceof WebSessionKey) {  
 request = ((WebSessionKey) sessionKey).getServletRequest();  
 }  
  
 if (request != null && sessionId != null) {  
 Session session = (Session) request.getAttribute(sessionId.toString());  
 if (session != null) {  
 return session;  
 }  
 }  
  
 Session session = super.retrieveSession(sessionKey);  
 if (request != null && sessionId != null) {  
 request.setAttribute(sessionId.toString(), session);  
 }  
 return session;  
 }  
  
}

### 最后在shiroConfig里面添加session配置

@Bean  
public RedisSessionDao sessionDao(){  
 RedisSessionDao sessionDao=new RedisSessionDao();  
 return sessionDao;  
}  
  
@Bean  
public RedisSessionManager sessionManager(){  
 RedisSessionManager sessionManager=new RedisSessionManager();  
 sessionManager.setSessionDAO(sessionDao());  
 return sessionManager;  
}

@Bean("securityManager")  
public SecurityManager securityManager(@Qualifier("authRealm") MyRealm authRealm) {  
 DefaultWebSecurityManager manager = new DefaultWebSecurityManager();  
 manager.setRealm(authRealm);  
 manager.setSessionManager(sessionManager());  
  
 return manager;  
}

## Rediss实现缓存管理

### 重写RedisCache类并实现源码的缓存

package com.cjy.buy.untils;  
  
import org.apache.shiro.cache.Cache;  
import org.apache.shiro.cache.CacheException;  
import org.springframework.stereotype.Component;  
import org.springframework.util.SerializationUtils;  
  
import javax.annotation.Resource;  
import java.util.Collection;  
import java.util.Set;  
  
*/\*\*  
 \** ***@author*** *cheng  
 \* 2018/11/4 21:15  
 \*/*@Component  
public class RedisCache<K, V> implements Cache<K, V> {  
  
 private final String CACHE\_PREFIX = "redis-cache:";  
  
 @Resource  
 private JedisUtil jedisUtil;  
  
 @SuppressWarnings("unchecked")  
 @Override  
 public V get(K k) throws CacheException {  
  
 System.*out*.println("从 redis 获取权限数据");  
  
 byte[] value = jedisUtil.get(getKey(k));  
 if (value != null) {  
 return (V) SerializationUtils.*deserialize*(value);  
 }  
 return null;  
 }  
  
 @Override  
 public V put(K k, V v) throws CacheException {  
  
 byte[] key = getKey(k);  
 byte[] value = SerializationUtils.*serialize*(v);  
 jedisUtil.set(key, value);  
 jedisUtil.expire(key, 600);  
  
 return v;  
 }  
  
 @SuppressWarnings("unchecked")  
 @Override  
 public V remove(K k) throws CacheException {  
  
 byte[] key = getKey(k);  
 byte[] value = jedisUtil.get(key);  
 jedisUtil.delete(key);  
  
 if (value != null) {  
 return (V) SerializationUtils.*deserialize*(value);  
 }  
 return null;  
 }  
  
 @Override  
 public void clear() throws CacheException {  
 // 此方法不需要重写  
 }  
  
 @Override  
 public int size() {  
 return 0;  
 }  
  
 @Override  
 public Set<K> keys() {  
 return null;  
 }  
  
 @Override  
 public Collection<V> values() {  
 return null;  
 }  
  
 private byte[] getKey(K k) {  
  
 if (k instanceof String) {  
 return (CACHE\_PREFIX + k).getBytes();  
 }  
 return SerializationUtils.*serialize*(k);  
 }  
}

### 自定义缓存管理器RedisCacheManager

package com.cjy.buy.manager;  
  
import com.cjy.buy.untils.RedisCache;  
import org.apache.shiro.cache.Cache;  
import org.apache.shiro.cache.CacheException;  
import org.apache.shiro.cache.CacheManager;  
  
import javax.annotation.Resource;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/25.  
 \*/*public class RedisCacheManager implements CacheManager{  
 @Resource  
 private RedisCache redisCache;  
  
 @Override  
 public <K, V> Cache<K, V> getCache(String s) throws CacheException {  
 return redisCache;  
 }  
}

### 最后在shiroConfig里面添加缓存配置

@Bean("redisCacheManager")  
public RedisCacheManager redisCacheManager(){  
 RedisCacheManager redisCacheManager=new RedisCacheManager();  
 return redisCacheManager;  
}

@Bean("securityManager")  
public SecurityManager securityManager(@Qualifier("authRealm") MyRealm authRealm) {  
 DefaultWebSecurityManager manager = new DefaultWebSecurityManager();  
 manager.setRealm(authRealm);  
 manager.setSessionManager(sessionManager());  
 manager.setCacheManager(redisCacheManager());  
   
 return manager;  
}

## 实现记住我的功能

### 在shiroConfig里面添加记住我的cookie配置

@Bean  
public SimpleCookie cookie(){  
 SimpleCookie simpleCookie=new SimpleCookie();  
 simpleCookie.setName("rememberMe");  
 //记着我的cookie的持续时间  
 simpleCookie.setMaxAge(3600);  
 return simpleCookie;  
}  
  
@Bean  
public CookieRememberMeManager cookieRememberMeManager(){  
 CookieRememberMeManager cookieRememberMeManager=new CookieRememberMeManager();  
 cookieRememberMeManager.setCookie(cookie());  
 return cookieRememberMeManager;  
}

@Bean("securityManager")  
public SecurityManager securityManager(@Qualifier("authRealm") MyRealm authRealm) {  
 DefaultWebSecurityManager manager = new DefaultWebSecurityManager();  
 manager.setRealm(authRealm);  
 manager.setSessionManager(sessionManager());  
 manager.setCacheManager(redisCacheManager());  
 manager.setRememberMeManager(cookieRememberMeManager());  
 return manager;  
}

### 在登陆的方法里面设置token是否实现记住我的功能

token.setRememberMe(rememberMe);

UsernamePasswordToken token = new UsernamePasswordToken(username, password);  
Subject subject = SecurityUtils.*getSubject*();  
new BaseRespMsg();  
try {  
 //进行验证，这里可以捕获异常，然后返回对应信息  
 token.setRememberMe(rememberMe);  
 subject.login(token);

### 在前端登陆页面传一个boolean值的remenberMe过来，实现是否记住我的功能

## 总结

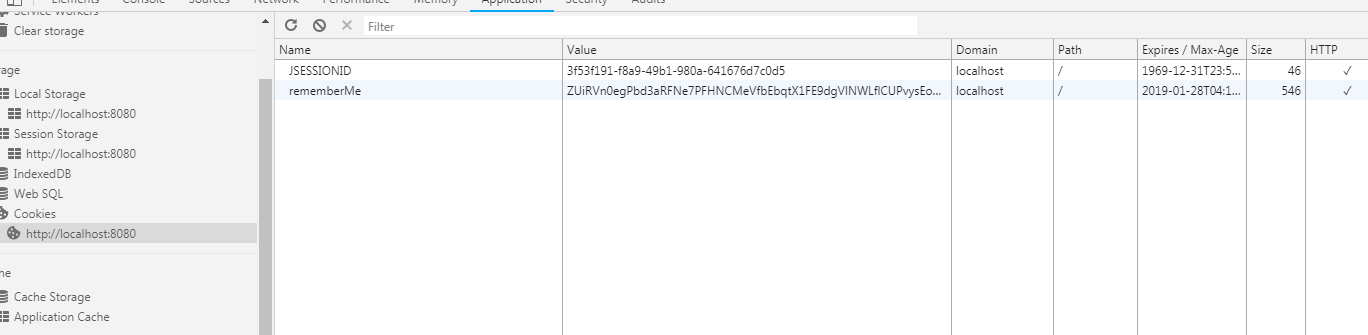
遇到很多坑

### 1，spring的xml文件转成springboot配置类，<bean>的地方springboot要@bean,对照例子如下：

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:util="http://www.springframework.org/schema/util"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">  
  
 <import resource="spring-dao.xml"/>  
 <import resource="spring-redis.xml"/>  
 <context:component-scan base-package="com.cheng"/>  
  
 <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">  
 <property name="securityManager" ref="securityManager"/>  
 <property name="loginUrl" value="login.html"/>  
 <property name="unauthorizedUrl" value="403.html"/>  
 <property name="filterChainDefinitions">  
 <value>  
 /login.html = anon  
 /subLogin = anon  
 <!-- /testRole = roles["admin"]  
 /testRole1 = roles["admin","admin1"]  
 /testPerms = perms["user:delete"]  
 /testPerms1 = perms["user:delete","user:add"]-->  
  
 <!--/testRole = roles["admin","admin1"]-->  
 /testRole1 = rolesOr["admin","admin1"]  
 /\* = authc  
 </value>  
 </property>  
 <property name="filters">  
 <util:map>  
 <entry key="rolesOr" value-ref="rolesOrFilter"/>  
 </util:map>  
 </property>  
 </bean>  
  
 <bean id="rolesOrFilter" class="com.cheng.filter.RolesOrFilter"/>  
  
 <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">  
 <!-- 创建 SecurityManager 对象 -->  
 <property name="realm" ref="realm"/>  
 <property name="sessionManager" ref="sessionManager"/>  
 <property name="cacheManager" ref="cacheManager"/>  
 <property name="rememberMeManager" ref="cookieRememberMeManager"/>  
 </bean>  
  
 <bean id="realm" class="com.cheng.shiro.realm.CustomRealm">  
 <property name="credentialsMatcher" ref="credentialsMatcher"/>  
 </bean>  
  
 <bean id="credentialsMatcher" class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">  
 <property name="hashAlgorithmName" value="md5"/>  
 <property name="hashIterations" value="1"/>  
 </bean>  
  
 <!-- 使用默认的 sessionManager 会频繁的操作读 redis 使用自定义的 CustomSessionManager -->  
 <!--<bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">-->  
 <bean id="sessionManager" class="com.cheng.session.CustomSessionManager">  
 <property name="sessionDAO" ref="sessionDao"/>  
 </bean>  
  
 <bean id="sessionDao" class="com.cheng.session.RedisSessionDao"/>  
  
 <bean id="cacheManager" class="com.cheng.cache.RedisCacheManager"/>  
  
 <bean id="cookieRememberMeManager" class="org.apache.shiro.web.mgt.CookieRememberMeManager">  
 <property name="cookie" ref="cookie"/>  
 </bean>  
  
 <bean id="cookie" class="org.apache.shiro.web.servlet.SimpleCookie">  
 <constructor-arg value="rememberMe"/>  
 <property name="maxAge" value="600000"/>  
 </bean>  
</beans>

package com.cjy.buy.config;  
  
import com.cjy.buy.manager.RedisCacheManager;  
import com.cjy.buy.manager.RedisSessionManager;  
import com.cjy.buy.untils.MyRealm;  
import com.cjy.buy.untils.RedisSessionDao;  
import org.apache.shiro.authc.credential.HashedCredentialsMatcher;  
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.CookieRememberMeManager;  
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;  
import org.apache.shiro.web.servlet.SimpleCookie;  
import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.DependsOn;  
  
import java.util.LinkedHashMap;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/25.  
 \*/*@Configuration  
public class ShiroConfig {  
 */\*\*  
 \* 密码校验规则HashedCredentialsMatcher  
 \* 这个类是为了对密码进行编码的 ,  
 \* 防止密码在数据库里明码保存 , 当然在登陆认证的时候 ,  
 \* 这个类也负责对form里输入的密码进行编码  
 \* 处理认证匹配处理器：如果自定义需要实现继承HashedCredentialsMatcher  
 \*/* @Bean("hashedCredentialsMatcher")  
 public HashedCredentialsMatcher hashedCredentialsMatcher() {  
 HashedCredentialsMatcher credentialsMatcher = new HashedCredentialsMatcher();  
 //指定加密方式为MD5  
 credentialsMatcher.setHashAlgorithmName("MD5");  
 //加密次数  
 credentialsMatcher.setHashIterations(1);  
 credentialsMatcher.setStoredCredentialsHexEncoded(true);  
 return credentialsMatcher;  
 }  
  
  
 @Bean("authRealm")  
 @DependsOn("lifecycleBeanPostProcessor")//可选  
 public MyRealm authRealm(@Qualifier("hashedCredentialsMatcher") HashedCredentialsMatcher matcher) {  
 MyRealm authRealm = new MyRealm();  
 //这里控制是否需要缓存 true  
 authRealm.setAuthorizationCachingEnabled(true);  
 authRealm.setCredentialsMatcher(matcher);  
 return authRealm;  
 }  
  
 @Bean  
 public RedisSessionDao sessionDao(){  
 RedisSessionDao sessionDao=new RedisSessionDao();  
 return sessionDao;  
 }  
  
 @Bean  
 public RedisSessionManager sessionManager(){  
 RedisSessionManager sessionManager=new RedisSessionManager();  
 sessionManager.setSessionDAO(sessionDao());  
 return sessionManager;  
 }  
  
  
 @Bean  
 public SimpleCookie cookie(){  
 SimpleCookie simpleCookie=new SimpleCookie();  
 simpleCookie.setName("rememberMe");  
 //记着我的cookie的持续时间  
 simpleCookie.setMaxAge(3600);  
 return simpleCookie;  
 }  
  
 @Bean  
 public CookieRememberMeManager cookieRememberMeManager(){  
 CookieRememberMeManager cookieRememberMeManager=new CookieRememberMeManager();  
 cookieRememberMeManager.setCookie(cookie());  
 return cookieRememberMeManager;  
 }  
  
 @Bean("redisCacheManager")  
 public RedisCacheManager redisCacheManager(){  
 RedisCacheManager redisCacheManager=new RedisCacheManager();  
 return redisCacheManager;  
 }  
 */\*\*  
 \* 定义安全管理器securityManager,注入自定义的realm  
 \** ***@param*** *authRealm  
 \** ***@return*** *\*/* @Bean("securityManager")  
 public SecurityManager securityManager(@Qualifier("authRealm") MyRealm authRealm) {  
 DefaultWebSecurityManager manager = new DefaultWebSecurityManager();  
 manager.setRealm(authRealm);  
 manager.setSessionManager(sessionManager());  
 manager.setCacheManager(redisCacheManager());  
 manager.setRememberMeManager(cookieRememberMeManager());  
 return manager;  
 }  
  
// @Bean("advisor")  
// public AuthorizationAttributeSourceAdvisor advisor(@Qualifier("authRealm") MyRealm authRealm) {  
// AuthorizationAttributeSourceAdvisor advisor=new AuthorizationAttributeSourceAdvisor();  
// advisor.setSecurityManager(securityManager(authRealm));  
// return advisor;  
// }  
  
  
 */\*\*  
 \* 定义shiroFilter过滤器并注入securityManager  
 \** ***@param*** *manager  
 \** ***@return*** *\*/* @Bean("shiroFilter")  
 public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager manager) {  
 ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();  
 //设置securityManager  
 bean.setSecurityManager(manager);  
 //设置登录页面  
 //可以写路由也可以写jsp页面的访问路径  
 bean.setLoginUrl("/html");  
 //设置登录成功跳转的页面  
 bean.setSuccessUrl("index");  
 //设置未授权跳转的页面  
 bean.setUnauthorizedUrl("getopt");  
 //定义过滤器  
 LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();  
  
 filterChainDefinitionMap.put("index", "authc");  
 filterChainDefinitionMap.put("html", "anon");  
 filterChainDefinitionMap.put("/user/login", "anon");  
// filterChainDefinitionMap.put("/admin", "roles[admin]");  
// filterChainDefinitionMap.put("/edit", "perms[delete]");  
// filterChainDefinitionMap.put("/druid/\*\*", "anon");  
 filterChainDefinitionMap.put("/statics/\*\*", "anon");  
 //需要登录访问的资源 , 一般将/\*\*放在最下边  
 filterChainDefinitionMap.put("/\*\*", "authc");  
 bean.setFilterChainDefinitionMap(filterChainDefinitionMap);  
 return bean;  
 }  
  
 */\*\*  
 \* Spring的一个bean , 由Advisor决定对哪些类的方法进行AOP代理 .  
 \** ***@return*** *\*/* @Bean  
 public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {  
 DefaultAdvisorAutoProxyCreator creator = new DefaultAdvisorAutoProxyCreator();  
 creator.setProxyTargetClass(true);  
 return creator;  
 }  
  
 */\*\*  
 \* 配置shiro跟spring的关联  
 \** ***@param*** *securityManager  
 \** ***@return*** *\*/* @Bean  
 public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(@Qualifier("securityManager") SecurityManager securityManager) {  
 AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();  
 advisor.setSecurityManager(securityManager);  
 return advisor;  
 }  
  
 */\*\*  
 \* lifecycleBeanPostProcessor是负责生命周期的 , 初始化和销毁的类  
 \* (可选)  
 \*/* @Bean("lifecycleBeanPostProcessor")  
 public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {  
 return new LifecycleBeanPostProcessor();  
 }  
  
}

### 2，实现全部功能后可以在前端F12里面看到



### 3，有一个控制缓存的方法，一定要设置为true才会缓存

authRealm.setAuthorizationCachingEnabled(true);

@Bean("authRealm")  
@DependsOn("lifecycleBeanPostProcessor")//可选  
public MyRealm authRealm(@Qualifier("hashedCredentialsMatcher") HashedCredentialsMatcher matcher) {  
 MyRealm authRealm = new MyRealm();  
 //这里控制是否需要缓存 true  
 authRealm.setAuthorizationCachingEnabled(true);  
 authRealm.setCredentialsMatcher(matcher);  
 return authRealm;  
}