**springBoot+springSecurity 数据库动态管理用户、角色、权限**

色和权限，并将用户、角色、权限和资源均采用数据库存储，并且自定义滤器，

并分别实现AccessDecisionManager、InvocationSecurityMetadataSourceService和UserDetailsService，并在配置文件中进行相应配置。

spring security的简单原理：

使用众多的拦截器对url拦截，以此来管理权限。但是这么多拦截器，笔者不可能对其一一来讲，主要讲里面核心流程的两个。

首先，权限管理离不开登陆验证的，所以登陆验证拦截器AuthenticationProcessingFilter要讲；

还有就是对访问的资源管理吧，所以资源管理拦截器AbstractSecurityInterceptor要讲；

但拦截器里面的实现需要一些组件来实现，所以就有了AuthenticationManager、accessDecisionManager等组件来支撑。

现在先大概过一遍整个流程，

1. 用户登陆时，会被AuthenticationProcessingFilter拦截，调用AuthenticationManager的实现，而且AuthenticationManager会调用ProviderManager来获取用户验证信息（不同的Provider调用的服务不同，因为这些信息可以是在数据库上，可以是在LDAP服务器上，可以是xml配置文件上等），如果验证通过后会将用户的权限信息封装一个User放到spring的全局缓存SecurityContextHolder中，以备后面访问资源时使用。

流程：用户登录---🡪AuthenticationProcessingFilter ---🡪ProviderManager--🡪userDetaisService（customUserService）

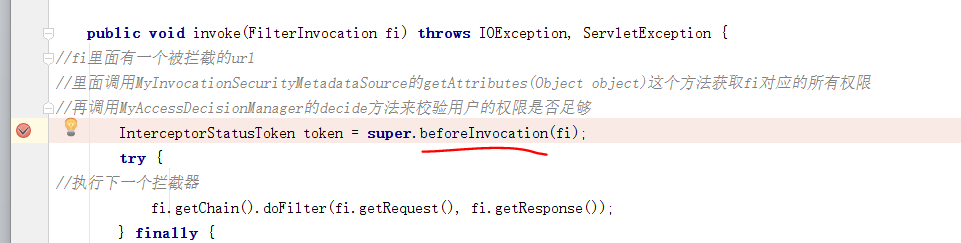
2.访问资源（即授权管理），访问url时，会通过AbstractSecurityInterceptor拦截器拦截，其中会调用FilterInvocationSecurityMetadataSource的方法来获取被拦截url所需的全部权限，在调用授权管理器AccessDecisionManager，这个授权管理器会通过spring的全局缓存SecurityContextHolder获取用户的权限信息，还会获取被拦截的url和被拦截url所需的全部权限，然后根据所配的策略（有：一票决定，一票否定，少数服从多数等），如果权限足够，则返回，权限不够则报错并调用权限不足页面。

（未登录时去访问url时，被定向到登录页面）

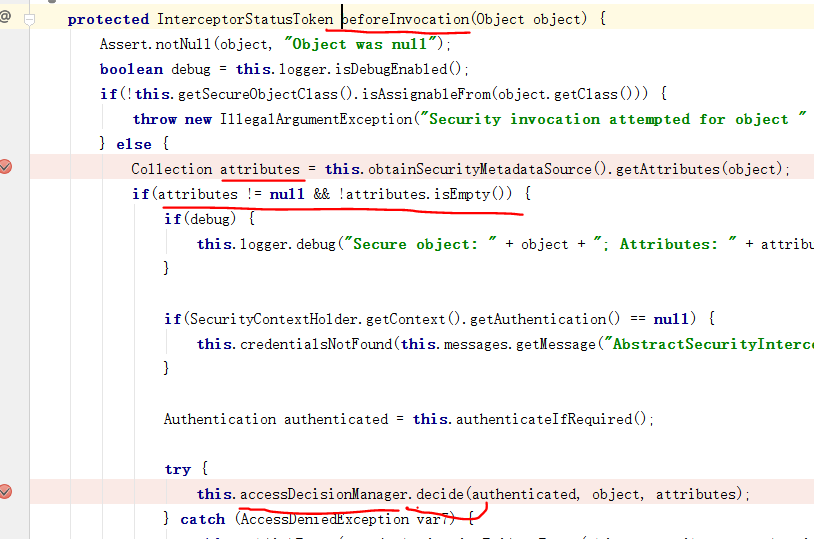
流程：AbstractSecurityInterceptor(MyFilterSecurityInterceptor)-–>

FilterInvocationSecurityMetadataSource(MyInvocationSecurityMetadataSourceService)（里的getAttributes方法返回不为null时）--🡪AccessDecisionManager！！！

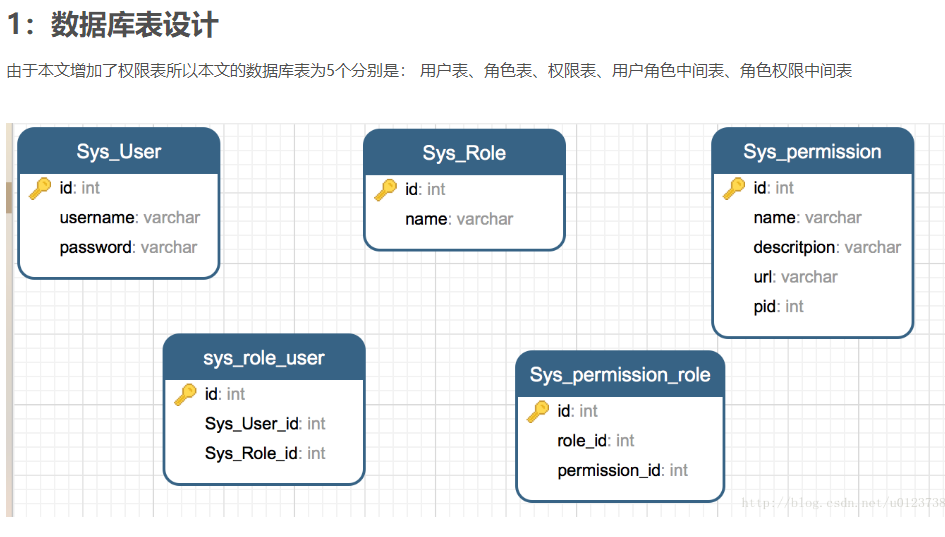
因为



其中beforeInvocation方法实现如下

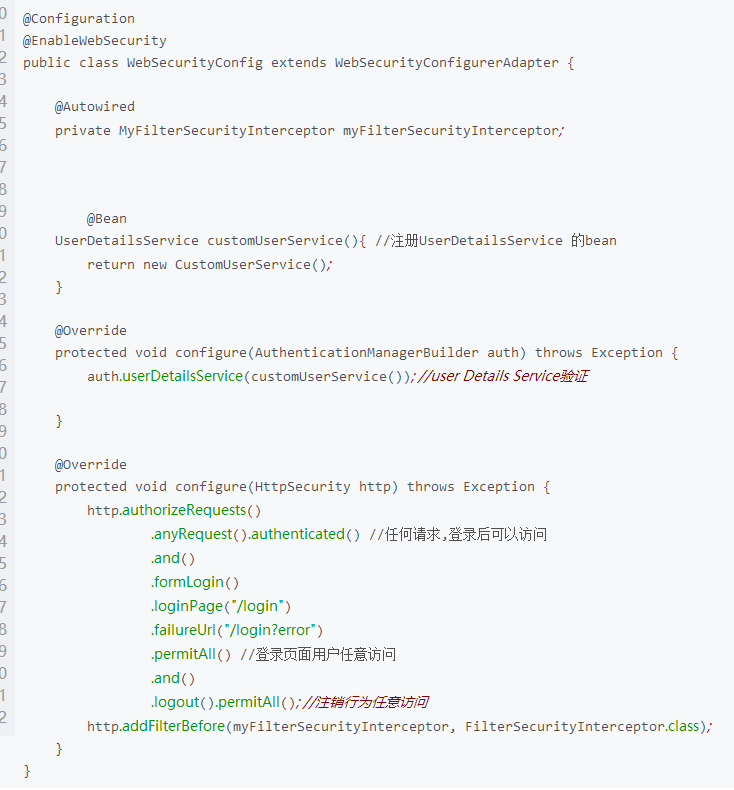


红线的表示，只有MyInvocationSecurityMetadataSourceServic里的getAttributes方法返回不为null时，（而getAttributes方法判断请求的url 是否在权限表中，如果在权限表中，则返回给 decide 方法，用来判定用户是否有此权限。如果不在权限表中则为null则放行）,才有机会调AccessDecisionManager的decide方法（decide方法该方法判断当前用户的权限表里是否有目前访问的url）



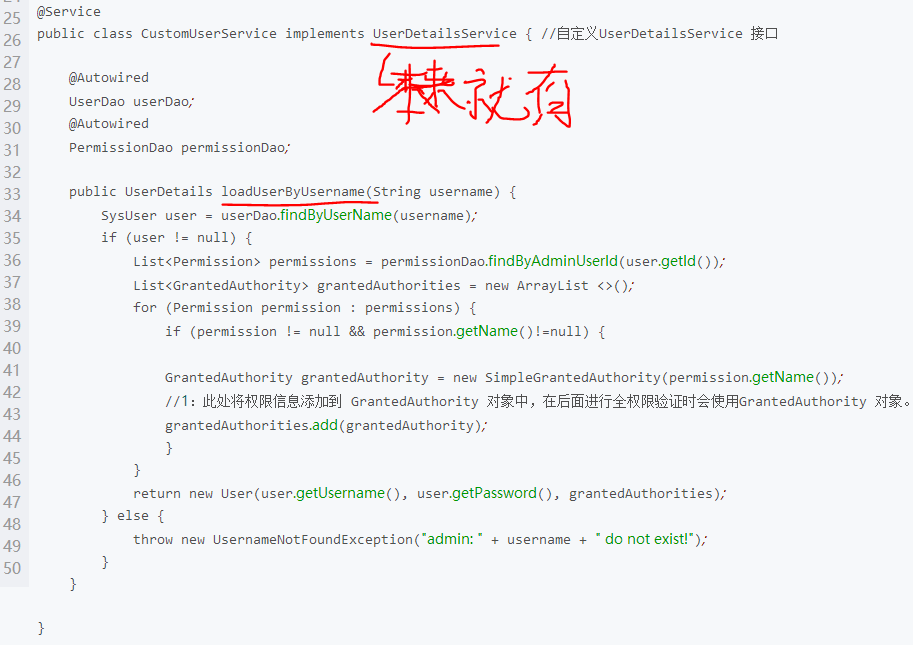
# 3：springSecurity 配置修改

## 3.1 修改 WebSecurityConfig.java



## 3.2 修改CustomUserService

修改CustomUserService.java 内容如下：



## 3.3 新增MyAccessDecisionManager MyAccessDecisionManager.java 文件

@Service

public class MyAccessDecisionManager implements AccessDecisionManager {

//1. decide 方法是判定是否拥有权限的决策方法，

//2.authentication 是释CustomUserService中循环添加到 GrantedAuthority 对象中的权限信息集合.

//3.object 包含客户端发起的请求的requset信息，可转换为 HttpServletRequest request = ((FilterInvocation) object).getHttpRequest();

//4.configAttributes 为MyInvocationSecurityMetadataSource的getAttributes(Object object)这个方法返回的结果，此方法是为了判定用户请求的url 是否在权限表中，如果在权限表中，则返回给 decide 方法，用来判定用户是否有此权限。如果不在权限表中则放行。

@Override

public void decide(Authentication authentication, Object object, Collection<ConfigAttribute> configAttributes) throws AccessDeniedException, InsufficientAuthenticationException {

if(null== configAttributes || configAttributes.size() <=0) {

return;

}

ConfigAttribute c;

String needRole;

for(Iterator<ConfigAttribute> iter = configAttributes.iterator(); iter.hasNext(); ) {

c = iter.next();

needRole = c.getAttribute();

for(GrantedAuthority ga : authentication.getAuthorities()) {//authentication 为用户登录时调用CustomUserService的loadUserByUsername

方法循环添加到 GrantedAuthority 对象中的权限信息集合，放到spring的全局缓存SecurityContextHolder中，以备后面访问资源时使用（即给这里用）

if(needRole.trim().equals(ga.getAuthority())) {

return;

}

}

}

throw new AccessDeniedException("no right");

}

@Override

public boolean supports(ConfigAttribute attribute) {

return true;

}

@Override

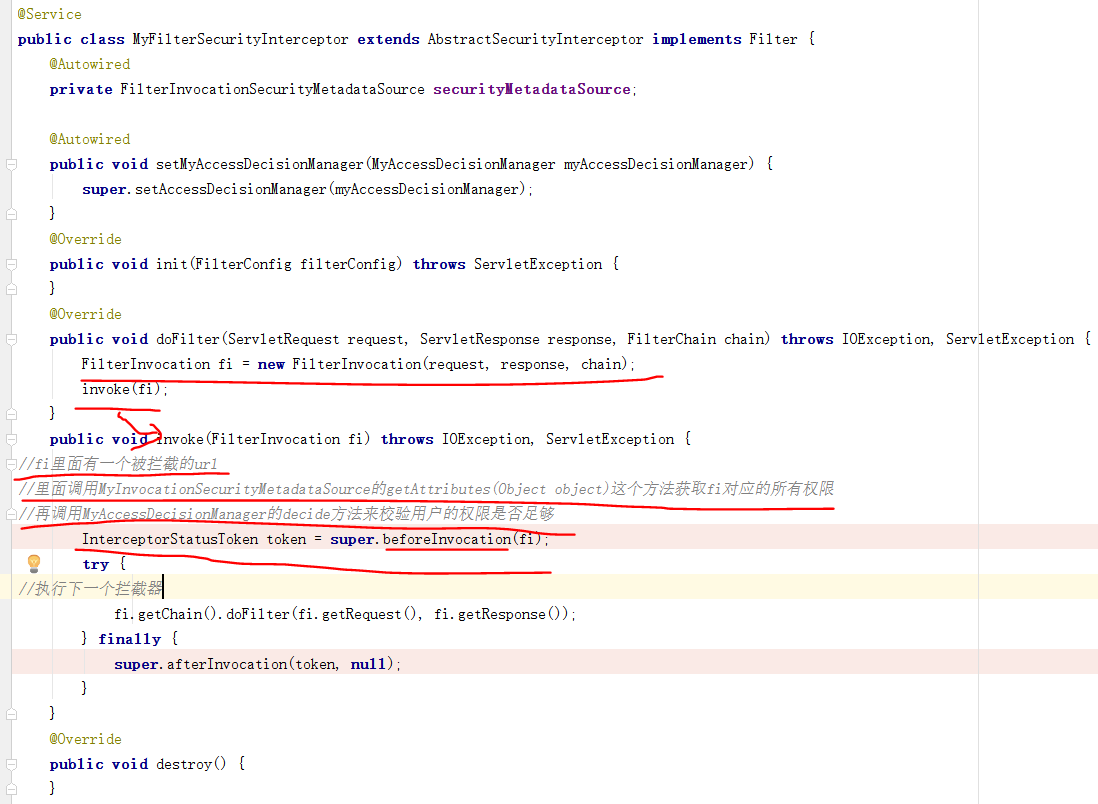
public boolean supports(Class<?> clazz) {

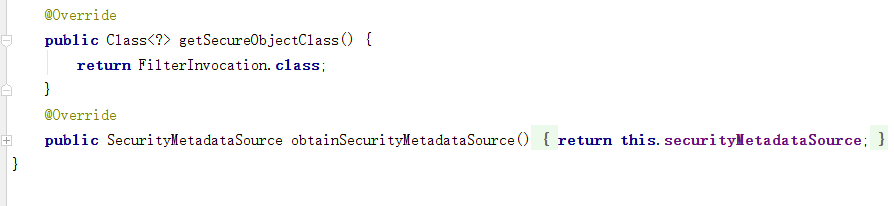
return true;

}

}

## 3.4 新增 MyFilterSecurityInterceptor MyFilterSecurityInterceptor.java 文件





## 3.5

## 新增 MyInvocationSecurityMetadataSourceService

@Service

public class MyInvocationSecurityMetadataSourceService implements

FilterInvocationSecurityMetadataSource {

@Autowired

private PermissionDao permissionDao;

private HashMap<String, Collection<ConfigAttribute>> map =null;

/\*\*

\* 加载权限表中所有权限

\*/

public void loadResourceDefine(){

map = new HashMap<>();

Collection<ConfigAttribute> array;

ConfigAttribute cfg;

List<Permission> permissions = permissionDao.findAll();

for(Permission permission : permissions) {

array = new ArrayList<>();

cfg = new SecurityConfig(permission.getName());

//此处只添加了用户的名字，其实还可以添加更多权限的信息，例如请求方法到ConfigAttribute的集合中去。此处添加的信息将会作为MyAccessDecisionManager类的decide的第三个参数。

array.add(cfg);

//用权限的getUrl() 作为map的key，用ConfigAttribute的集合作为 value，

map.put(permission.getUrl(), array);

}

}

//此方法是为了判定用户请求的url 是否在权限表中，如果在权限表中，则返回给 decide 方法，用来判定用户是否有此权限。如果不在权限表中则放行。

@Override

public Collection<ConfigAttribute> getAttributes(Object object) throws IllegalArgumentException {

if(map ==null) loadResourceDefine();

//object 中包含用户请求的request 信息

HttpServletRequest request = ((FilterInvocation) object).getHttpRequest();

AntPathRequestMatcher matcher;

String resUrl;

for(Iterator<String> iter = map.keySet().iterator(); iter.hasNext(); ) {

resUrl = iter.next();

matcher = new AntPathRequestMatcher(resUrl);

if(matcher.matches(request)) {

return map.get(resUrl);

}

}

return null;

}

@Override

public Collection<ConfigAttribute> getAllConfigAttributes() {

return null;

}

@Override

public boolean supports(Class<?> clazz) {

return true;

}

}

# 5：修改HomeController.java 文件



**6.添加config**

@Configuration  
@EnableWebSecurity  
**public class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {  
  
 @Autowired  
 **private** MyFilterSecurityInterceptor **myFilterSecurityInterceptor**;  
 @Autowired  
 UserDetailsService **customUserService**;  
 @Override  
 **protected void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  
 *//用户登陆，会被AuthenticationProcessingFilter拦截，调用AuthenticationManager的实现  
 //而AuthenticationManager调用Provider，provider调用userDetaisService（customUserService）来根据username获取真实的数据库信息。* auth.userDetailsService(**customUserService**); *//user Details Service验证* }  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http.authorizeRequests()  
 .antMatchers(**"/css/\*\*"**).permitAll()  
 .anyRequest().authenticated() *//任何请求,登录后可以访问(这样的请求才会执行CustomUserService类的方法)* .and()  
 .formLogin() *//用户登陆，会被AuthenticationProcessingFilter拦截，调用AuthenticationManager的实现，而AuthenticationManager调用Provider，provider调用userDetaisService（customUserService类）来根据username获取真实的数据库信息。* .loginPage(**"/login"**)  
 .defaultSuccessUrl(**"/"**)  
 .failureUrl(**"/login?error"**)  
 .permitAll() *//登录页面用户任意访问* .and()  
 .logout().permitAll(); *//注销行为任意访问* http.addFilterBefore(**myFilterSecurityInterceptor**, FilterSecurityInterceptor.**class**);  
 }  
  
}

# 7.测试检验

启动访问 <http://localhost:8082/> 到登录页面

