@Component  
public class MyInvocationSecurityMetadataSourceService implements FilterInvocationSecurityMetadataSource {  
  
 @Autowired  
 private PermissionMapper permissionMapper;  
  
 /\*\*  
 \* 每一个资源所需要的角色 Collection<ConfigAttribute>决策器会用到  
 \*/  
 private static HashMap<String, Collection<ConfigAttribute>> map =null;  
  
  
 /\*\*  
 \* 返回请求的资源需要的角色  
 \*/  
 @Override  
 public Collection<ConfigAttribute> getAttributes(Object o) throws IllegalArgumentException {  
 if (null == map) {  
 loadResourceDefine();  
 }  
 //object 中包含用户请求的request 信息  
 HttpServletRequest request = ((FilterInvocation) o).getHttpRequest();  
 for (Iterator<String> it = map.keySet().iterator() ; it.hasNext();) {  
 String url = it.next();  
 if (new AntPathRequestMatcher( url ).matches( request )) {  
 return map.get( url );  
 }  
 }  
  
 return null;  
 }  
  
 @Override  
 public Collection<ConfigAttribute> getAllConfigAttributes() {  
 return null;  
 }  
  
 @Override  
 public boolean supports(Class<?> aClass) {  
 return true;  
 }  
  
 /\*\*  
 \* 初始化 所有资源 对应的角色  
 \*/  
 public void loadResourceDefine() {  
 map = new HashMap<>(16);  
 //权限资源 和 角色对应的表 也就是 角色权限 中间表  
 List<RolePermisson> rolePermissons = permissionMapper.getRolePermissions();  
  
 //某个资源 可以被哪些角色访问  
 for (RolePermisson rolePermisson : rolePermissons) {  
  
 String url = rolePermisson.getUrl();  
 String roleName = rolePermisson.getRoleName();  
 ConfigAttribute role = new SecurityConfig(roleName);  
  
 if(map.containsKey(url)){  
 map.get(url).add(role);  
 }else{  
 List<ConfigAttribute> list = new ArrayList<>();  
 list.add( role );  
 map.put( url , list );  
 }  
 }  
 }  
  
  
}

MyInvocationSecurityMetadataSourceService 类实现了 FilterInvocationSecurityMetadataSource，FilterInvocationSecurityMetadataSource 的作用是用来储存请求与权限的对应关系。

FilterInvocationSecurityMetadataSource接口有3个方法：

* boolean supports(Class<?> clazz)：指示该类是否能够为指定的方法调用或Web请求提供ConfigAttributes。
* Collection

 getAllConfigAttributes()：Spring容器启动时自动调用, 一般把所有请求与权限的对应关系也要在这个方法里初始化, 保存在一个属性变量里。

* Collection

 getAttributes(Object object)：当接收到一个http请求时, filterSecurityInterceptor会调用的方法. 参数object是一个包含url信息的HttpServletRequest实例. 这个方法要返回请求该url所需要的所有权限集合。

/\*\*  
 \* 决策器  
 \*/  
@Component  
public class MyAccessDecisionManager implements AccessDecisionManager {  
  
 private final static Logger logger = LoggerFactory.getLogger(MyAccessDecisionManager.class);  
  
 /\*\*  
 \* 通过传递的参数来决定用户是否有访问对应受保护对象的权限  
 \*  
 \* @param authentication 包含了当前的用户信息，包括拥有的权限。这里的权限来源就是前面登录时UserDetailsService中设置的authorities。  
 \* @param object 就是FilterInvocation对象，可以得到request等web资源  
 \* @param configAttributes configAttributes是本次访问需要的权限  
 \*/  
 @Override  
 public void decide(Authentication authentication, Object object, Collection<ConfigAttribute> configAttributes) throws AccessDeniedException, InsufficientAuthenticationException {  
 if (null == configAttributes || 0 >= configAttributes.size()) {  
 return;  
 } else {  
 String needRole;  
 for(Iterator<ConfigAttribute> iter = configAttributes.iterator(); iter.hasNext(); ) {  
 needRole = iter.next().getAttribute();  
  
 for(GrantedAuthority ga : authentication.getAuthorities()) {  
 if(needRole.trim().equals(ga.getAuthority().trim())) {  
 return;  
 }  
 }  
 }  
 throw new AccessDeniedException("当前访问没有权限");  
 }  
  
 }  
  
 /\*\*  
 \* 表示此AccessDecisionManager是否能够处理传递的ConfigAttribute呈现的授权请求  
 \*/  
 @Override  
 public boolean supports(ConfigAttribute configAttribute) {  
 return true;  
 }  
  
 /\*\*  
 \* 表示当前AccessDecisionManager实现是否能够为指定的安全对象（方法调用或Web请求）提供访问控制决策  
 \*/  
 @Override  
 public boolean supports(Class<?> aClass) {  
 return true;  
 }  
  
}

MyAccessDecisionManager 类实现了AccessDecisionManager接口，AccessDecisionManager是由AbstractSecurityInterceptor调用的，它负责鉴定用户是否有访问对应资源（方法或URL）的权限。

@Component  
public class MyFilterSecurityInterceptor extends AbstractSecurityInterceptor implements Filter {  
  
  
 @Autowired  
 private FilterInvocationSecurityMetadataSource securityMetadataSource;  
  
 @Autowired  
 public void setMyAccessDecisionManager(MyAccessDecisionManager myAccessDecisionManager) {  
 super.setAccessDecisionManager(myAccessDecisionManager);  
 }  
  
  
 @Override  
 public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  
  
 FilterInvocation fi = new FilterInvocation(servletRequest, servletResponse, filterChain);  
 invoke(fi);  
 }  
  
 public void invoke(FilterInvocation fi) throws IOException, ServletException {  
  
 InterceptorStatusToken token = super.beforeInvocation(fi);  
 try {  
 //执行下一个拦截器  
 fi.getChain().doFilter(fi.getRequest(), fi.getResponse());  
 } finally {  
 super.afterInvocation(token, null);  
 }  
 }  
  
 @Override  
 public Class<?> getSecureObjectClass() {  
 return FilterInvocation.class;  
 }  
  
 @Override  
 public SecurityMetadataSource obtainSecurityMetadataSource() {  
  
 return this.securityMetadataSource;  
 }  
   
   
}

每种受支持的安全对象类型（方法调用或Web请求）都有自己的拦截器类，它是AbstractSecurityInterceptor的子类，AbstractSecurityInterceptor 是一个实现了对受保护对象的访问进行拦截的抽象类。

AbstractSecurityInterceptor的机制可以分为几个步骤：

* 1. 查找与当前请求关联的“配置属性（简单的理解就是权限）”
* 2. 将 安全对象（方法调用或Web请求）、当前身份验证、配置属性 提交给决策器（AccessDecisionManager）
* 3. （可选）更改调用所根据的身份验证
* 4. 允许继续进行安全对象调用(假设授予了访问权)
* 5. 在调用返回之后，如果配置了AfterInvocationManager。如果调用引发异常，则不会调用AfterInvocationManager。

AbstractSecurityInterceptor中的方法说明：

* beforeInvocation()方法实现了对访问受保护对象的权限校验，内部用到了AccessDecisionManager和AuthenticationManager；
* finallyInvocation()方法用于实现受保护对象请求完毕后的一些清理工作，主要是如果在beforeInvocation()中改变了SecurityContext，则在finallyInvocation()中需要将其恢复为原来的SecurityContext，该方法的调用应当包含在子类请求受保护资源时的finally语句块中。
* afterInvocation()方法实现了对返回结果的处理，在注入了AfterInvocationManager的情况下默认会调用其decide()方法。

了解了AbstractSecurityInterceptor，就应该明白了，我们自定义MyFilterSecurityInterceptor就是想使用我们之前自定义的 AccessDecisionManager 和 securityMetadataSource。

**SecurityConfig**

@EnableWebSecurity注解以及WebSecurityConfigurerAdapter一起配合提供基于web的security。自定义类 继承了WebSecurityConfigurerAdapter来重写了一些方法来指定一些特定的Web安全设置。

<https://mp.weixin.qq.com/s?__biz=MzI0NTYzMjM0Ng==&mid=2247484315&idx=1&sn=122655740e497e57fa7a5c5c93d15ef4&scene=21>