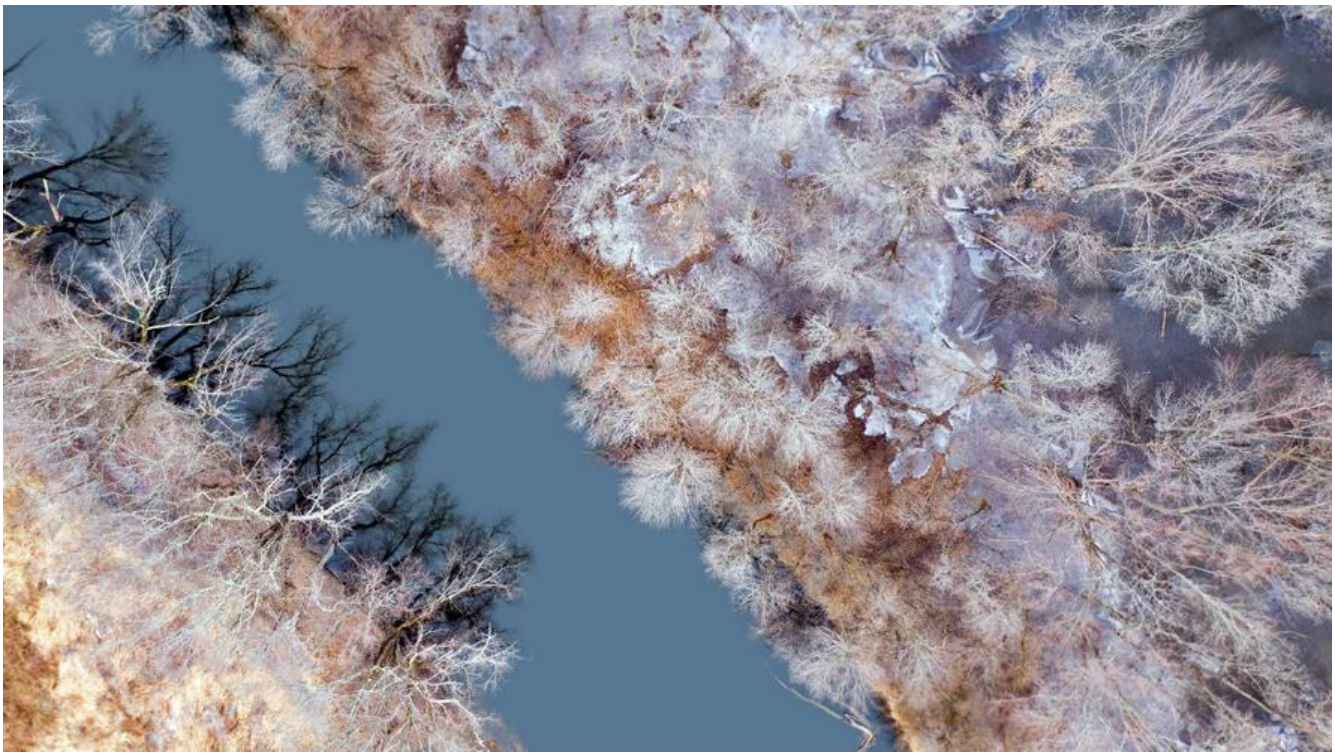
链滴

Security 与响应式 WebFlux(二)

作者：hong1yuan

原文链接：https://ld246.com/article/1599217666900

来源网站：链滴



实践

用Security其实可以有几种玩法，主要需要结合自己的业务，看自己的服务的位置是什么样

比如：角色是**预置**好不变的，还是未预置好**可变的**，权限是**固定的**还是**不固定**的，权限校验是**页面级**还是**接口级**的，接口级权限是以**角色**为维度还是以**权限**为维度。

以业务为导向结合与利用框架达到自身的目地我认为还是比较重要的，**不应该为了迎合技术而改变自的业务。**

一.利用Security自身鉴权进行校验

第一步，配置security配置文件，相当于config

```
@EnableWebFluxSecurity
public class SecurityConfig {


    @Autowired
    private AuthenticationSuccessHandler authenticationSuccessHandler;
    @Autowired
    private AuthenticationFaillHandler authenticationFaillHandler;
    @Autowired
    private CustomHttpBasicServerAuthenticationEntryPoint customHttpBasicServerAuthenticat
onEntryPoint;


    //security的鉴权排除列表
    private static final String[] excludedAuthPages = {
            "/auth/login",
            "/auth/logout",
```

```
            "/health",
            "/api/socket/**"
//            "/user/**"
    };

    @Bean
    SecurityWebFilterChain webFluxSecurityFilterChain(ServerHttpSecurity http) throws Excepti
n {
        http
                .authorizeExchange()
                .pathMatchers(excludedAuthPages).permitAll()  //无需进行权限过滤的请求路径
                .pathMatchers(HttpMethod.OPTIONS).permitAll() //option 请求默认放行
                .anyExchange().authenticated()
                .and()
                .httpBasic()
                .and()
                .formLogin().loginPage("/auth/login")
                .authenticationSuccessHandler(authenticationSuccessHandler) //认证成功
                .authenticationFailureHandler(authenticationFaillHandler) //登陆验证失败
                .and().exceptionHandling().authenticationEntryPoint(customHttpBasicServerAuthent
cationEntryPoint)  //基于http的接口请求鉴权失败
                .and() .csrf().disable()//必须支持跨域
                .logout().disable()
                ;
        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return  NoOpPasswordEncoder.getInstance(); //默认
    }


}
```

利用webFlux独有的@EnableWebFluxSecurity进行开启webflux的Security

可以定义各种需要鉴权或不需要鉴权的列表，可以指定鉴权的方式如图上遇到**excludedAuthPages** 据列表则permitAll()进行放行 也可以去配置相应的地址，需要指定角色与指定权限才能访问

如下：

```
                .pathMatchers(excludedAuthPages).hasRole("admin")
                .pathMatchers(excludedAuthPages).hasAuthority("T0001")
```

可以指定这个A数组中的地址admin才能访问，指定B数组中的地址T0001权限才能访问

防止CSRF攻击关闭是为了跨域的支持，不懂CSRF可以百度、google。

AuthenticationSuccessHandler 与 AuthenticationFailureHandler 是鉴权成功或失败后进入的逻辑

这个需要重写方法 之后会贴代码。

CustomHttpBasicServerAuthenticationEntryPoint 是http请求失败后的方法 差不多的功效

passwordEncoder是密码加密的方式，可以不写，如果用了什么MD5啊 或盐值加密码等方式，可配置一下。

formLogin().loginPage("/auth/login") 是用来配置Security的登录地址的

二.配置登录成功与失败的返回类

1.登录成功

这里面就可以去写一些业务逻辑 比如登录失败 只让重试密码三次或者登录成功 token的时长等

```
@Component
public class AuthenticationSuccessHandler extends WebFilterChainServerAuthenticationSucce
sHandler {

    @Override
    public Mono<Void> onAuthenticationSuccess(WebFilterExchange webFilterExchange, Aut
entication authentication){
        ServerWebExchange exchange = webFilterExchange.getExchange();
        ServerHttpResponse response = exchange.getResponse();
        //设置headers
        HttpHeaders httpHeaders = response.getHeaders();
        httpHeaders.add("Content-Type", "application/json; charset=UTF-8");
        httpHeaders.add("Cache-Control", "no-store, no-cache, must-revalidate, max-age=0");
        //设置body
       byte[]   dataBytes={};
        ObjectMapper mapper = new ObjectMapper();
        try {
            User user=(User)authentication.getPrincipal();
            AuthUserDetails userDetails=buildUser(user);
            byte[] authorization=(userDetails.getUsername()+":"+userDetails.getPassword()).getBy
es();
            String token= Base64.getEncoder().encodeToString(authorization);
            httpHeaders.add(HttpHeaders.AUTHORIZATION, token);
            dataBytes=mapper.writeValueAsBytes(Result.success(userDetails));
        }
        catch (Exception ex){
            ex.printStackTrace();
            JsonObject result = new JsonObject();
            result.addProperty("status", GatewayErrorCodeEnum.STAFF_NOT_EXIST.getCode());
            result.addProperty("message", GatewayErrorCodeEnum.STAFF_NOT_EXIST.getMessage
));
            dataBytes=result.toString().getBytes();
        }
        DataBuffer bodyDataBuffer = response.bufferFactory().wrap(dataBytes);
        return response.writeWith(Mono.just(bodyDataBuffer));
    }



    private AuthUserDetails buildUser(User user){
        AuthUserDetails userDetails=new AuthUserDetails();
        userDetails.setUsername(user.getUsername());
```

```
        userDetails.setPassword(user.getPassword().substring(user.getPassword().lastIndexOf("}"
+1,user.getPassword().length()));
        return userDetails;
    }
```

主要这块功能就是通过已经成功登录的用户上获取用户相关的**authentication.getPrincipal()** 中的息，之后利用JWT（json web token）等相应方式进行生成用户唯一的Token 之后用户可以利用TOKN进行鉴权的操作

2.登录失败

```
@Component
public class AuthenticationFaillHandler  implements ServerAuthenticationFailureHandler {

    @Override
    public Mono<Void> onAuthenticationFailure(WebFilterExchange webFilterExchange, Auth
nticationException e) {
        ServerWebExchange exchange = webFilterExchange.getExchange();
        ServerHttpResponse response = exchange.getResponse();
        //设置headers
        HttpHeaders httpHeaders = response.getHeaders();
        httpHeaders.add("Content-Type", "application/json; charset=UTF-8");
        httpHeaders.add("Cache-Control", "no-store, no-cache, must-revalidate, max-age=0");
        //设置body
        byte[]   dataBytes={};
        try {
            ObjectMapper mapper = new ObjectMapper();
            dataBytes=mapper.writeValueAsBytes(Result.fail(GatewayErrorCodeEnum.STAFF_NOT
EXIST.getCode(),GatewayErrorCodeEnum.STAFF_NOT_EXIST.getMessage()));
        }
        catch (Exception ex){
            ex.printStackTrace();
        }
        DataBuffer bodyDataBuffer = response.bufferFactory().wrap(dataBytes);
        return response.writeWith(Mono.just(bodyDataBuffer));
    }
}
```

同理，登录失败的一些操作

http请求

```
@Component
public class CustomHttpBasicServerAuthenticationEntryPoint extends HttpBasicServerAuthent
cationEntryPoint /* implements ServerAuthenticationEntryPoint */{


    private static final String WWW_AUTHENTICATE = "WWW-Authenticate";
    private static final String DEFAULT_REALM = "Realm";
    private static String WWW_AUTHENTICATE_FORMAT = "Basic realm=\"%s\"";
    private String headerValue = createHeaderValue("Realm");
    public CustomHttpBasicServerAuthenticationEntryPoint() {
```

```
    }



    public void setRealm(String realm) {
        this.headerValue = createHeaderValue(realm);
    }

    private static String createHeaderValue(String realm) {
        Assert.notNull(realm, "realm cannot be null");
        return String.format(WWW_AUTHENTICATE_FORMAT, new Object[]{realm});
    }

    @Override
    public Mono<Void> commence(ServerWebExchange exchange, AuthenticationException e)
{
            ServerHttpResponse response = exchange.getResponse();
            response.setStatusCode(HttpStatus.UNAUTHORIZED);
            response.getHeaders().add("Content-Type", "application/json; charset=UTF-8");
            response.getHeaders().set(HttpHeaders.AUTHORIZATION, this.headerValue);
            JsonObject result = new JsonObject();
            result.addProperty("status", GatewayErrorCodeEnum.STAFF_NOT_EXIST.getCode());
            result.addProperty("message", GatewayErrorCodeEnum.STAFF_NOT_EXIST.getMessage
));
            byte[] dataBytes=result.toString().getBytes();
            DataBuffer bodyDataBuffer = response.bufferFactory().wrap(dataBytes);
            return response.writeWith(Mono.just(bodyDataBuffer));
    }
}
```

同理，里面的一些枚举是一些错误码之类的，自己替换一下就好

登录认证

首先 可以重写Security用户信息的类 让自己需要的字段加上

```
public class AuthUserDetails implements UserDetails {

    private String username;
    @JsonIgnore
    private String password;
    private Collection<String> roles;
    private String token;
}
```

之后通过重写方法进行寻找用户

```
@Component
public class SecurityUserDetailsService implements ReactiveUserDetailsService {

     @Value("${spring.security.user.name}")
     private   String userName;
```

```
    @Value("${spring.security.user.password}")
    private   String password;


    @Override
    public Mono<UserDetails> findByUsername(String username) {
      //todo 预留调用数据库根据用户名获取用户
       if(StringUtils.equals(userName,username)){
          UserDetails user = User.withUsername(userName)
                .password(password)
                 .roles("admin").authorities(AuthorityUtils.commaSeparatedStringToAuthorityList(
admin"))
                 .build();
          return Mono.just(user);
       }
       else{
          return Mono.error(new UsernameNotFoundException("User Not Found"));

       }

    }



}
```

实现**ReactiveUserDetailsService** 方法后可以去查询用户 这里只是写个例子，真实情况你可以通过库，通过feign 等方式获取到用户的信息，之后Security会通过你查询到的用户信息先去与你提供的户名与密码进行校验 看密码通没通过，密码的加密方式 就是前面配置的PASSWORD规则，之后根据的角色与权限使你当前线程拥有这些角色与权限可以访问的地址。

这里登录成功就会走上面讲过的成功类与失败类中了。

到这还差一点就可以了 就是生成Token后的拦截 解析并且让Security注入进当前角色

有两种方式 一种是在SecurityConfig里 配置webFlux的拦截器 之后在拦截器，把token解析加上让security当前线程获得认证的角色和权限

```
http.addFilterBefore()
```

他有两个参数，一个是拦截器类是哪个，一个是在哪拦截，我之前用的是AUTHENTICATION，在认前之类的，这块网上没有什么文档，spring官网也没有专门的讲解，不过看源码里有

```
public ServerHttpSecurity addFilterBefore(WebFilter webFilter, SecurityWebFiltersOrder order)
{
       this.webFilters.add(new OrderedWebFilter(webFilter, order.getOrder() - 1));
       return this;
    }
```

另一种 也是在SecurityConfig中 配置，配置**securityContextRepository** 这个是Security上下文存库，可以重写这个方法去让你拥有权限

第一种方法我已经实验过了，是可以的，但是针对我们的业务不实用，我就把代码给删除了，所以想用第一种方式的自己研究一下吧

第二种方式我也实现了，不过我们的业务是动态**RBAC（Role-Based Access Control）动态角色**以上面的Security + WebFlux的方式不适用，所以我用了另一种Security的配置，也实现了**security ontextRepository**的用法，方法是同用的，所以这里先到这里

总结

利用Security的认证方法进行安全认证虽然很不错，他在接口层也可以用注解的方式去鉴权，但现在类似OA的权限系统已经变成了随时加新角色，随时加权限的时代，以前的方式需要一改角色和权限去改代码的方式已经不适合现在的需求了，所以上述代码能解决一定需求，根据动态RBAC的功能实不了，所以下一期会讲可以**动态RBAC**的功能并且**WebFlux的mvc三层实现**与**Feign调用不了坑的填入**