1. 父工程

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-alibaba-dependencies</artifactId>  
 <version>2021.0.1.0</version>  
 <scope>import</scope>  
 <type>pom</type>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>2021.0.1</version>  
 <scope>import</scope>  
 <type>pom</type>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

1. 引入依赖奥

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

完事再启动你就会发现需要登录了。

------------------------------------------------------------------------------------------------------------------------------

配置类：：

@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
 @Autowired  
 JwtAuthenticationTokenFilter authenticationTokenFilter;  
 @Bean  
 public PasswordEncoder passwordEncoder (){  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }

}

然后写 UserDetailService

@Service  
public class UserDetailService implements UserDetailsService {  
  
 @Autowired  
 PasswordEncoder passwordEncoder;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 //查询用户信息 数据库哦 你懂的权限的都要查出来  
 System.*out*.println("UserDetails");  
 System.*out*.println(username);  
 User userDetails = new User(username, "$2a$10$Vyfeyx66L3vPtd8NlpbdZeO0fT69UcMPPC6aJdEsYVwVczE2vVGGW");  
 ArrayList<String > list = new ArrayList<>();  
 list.add("test"); //权限  
 userDetails.setPermissions(list);  
 return userDetails;  
 }  
}

然后再登录你就会发现 你的登录会走进这里进行验证哦

后来你就可以配置过滤器啦超简单的 权限问题 你需要让你的User 继承一下UserDetails

//这个是过滤器

@Component  
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {  
  
 @Autowired  
 RedisTemplate redisTemplate;  
 @Override  
 protected void doFilterInternal(HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain filterChain) throws ServletException, IOException {  
 System.*out*.println("进得来？");  
 //获取token  
 String token = request.getHeader("token");  
 System.*out*.println("token=="+token);  
 if(token == null){  
 //放行 进去其他的过滤器  
 filterChain.doFilter(request,response);  
 //调用会回来的 不可以让他执行下面的 代码哦  
 return;  
 }  
 //解析token  
  
 //完事从redis中取出来LoginUser 这里要比对的  
 User user = new User("用户名","密码");  
 String pression = redisTemplate.get(token);  
 String substring = "null";  
  
 substring = pression.substring(1, pression.length() - 1);  
  
  
 ArrayList<String> list = new ArrayList<>();  
 list.add(substring);  
 System.*out*.println(substring);  
 user.setPermissions(list);  
 //将LoginUser存入 SecurityContextHolder  
 //三个参数的构造器里面的已认证是true  
 //*TODO 传进去权限信息哦* UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken //这个传进去的是从redis中取出来的对象哦  
 = new UsernamePasswordAuthenticationToken(  
 user  
 ,null, user.getAuthorities());  
 SecurityContextHolder.*getContext*().setAuthentication(usernamePasswordAuthenticationToken);  
 //放行  
 filterChain.doFilter(request,response);  
 }  
}

//User 已经记住那些返回值都设置为true哦

public class User implements UserDetails {  
 private String username;  
 private String password;  
 private ArrayList<String> permissions;  
  
 @JSONField(serialize = false)//不会呗序列化  
 private List<SimpleGrantedAuthority> permissionsList ;  
  
 public User(String username, String password, ArrayList<String> permissions) {  
 this.username = username;  
 this.password = password;  
 this.permissions = permissions;  
 }  
  
  
 public User(String username, String password) {  
 this.username = username;  
 this.password = password;  
 }  
  
 public User() {  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 ", permissions=" + permissions +  
 '}';  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 //将permissions中的string类型的权限封装成她的实现类SimpleGrantedAuthority  
 if (this.permissionsList != null){  
 return this.permissionsList;  
 }  
 List<SimpleGrantedAuthority> list = new ArrayList<>();  
  
 for (String permission : permissions) {  
 SimpleGrantedAuthority authority = new SimpleGrantedAuthority(permission);  
 list.add(authority);  
 }  
 this.permissionsList = list;  
 return permissionsList;  
 }  
  
 @Override  
 public String getPassword() {  
 return this.password;  
 }  
  
 @Override  
 public String getUsername() {  
 return this.username;  
 }  
  
 @Override  
 @JSONField(serialize = false)//不会呗序列化  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 @JSONField(serialize = false)//不会呗序列化  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 @JSONField(serialize = false)//不会呗序列化  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 @JSONField(serialize = false)//不会呗序列化  
 public boolean isEnabled() {  
 return true;  
 }  
  
 public ArrayList<String> getPermissions() {  
 return permissions;  
 }  
  
 public void setPermissions(ArrayList<String> permissions) {  
 this.permissions = permissions;  
 }  
}

完事之后需要将 过滤器注册给 security 通过配置类哦

由于接下来测试需要自己写登录接口 所以 需要给配置类里添加允许匿名访问

@Autowired  
JwtAuthenticationTokenFilter authenticationTokenFilter;

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable().  
 //不通过Session获取SecurityContext  
 sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口允许匿名访问  
 .antMatchers("/user/login").anonymous()  
 //出上面以外的所有请求全部需要鉴全认证  
 .anyRequest().authenticated();  
 //添加过滤器  
 http.addFilterBefore(authenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
  
   
}

//这个是 登录Controller调用的service

@Service  
public class LoginServiceImpl implements LoginService {  
  
 @Autowired  
 public AuthenticationManager authenticationManager;  
 @Autowired  
 RedisTemplate redisTemplate;  
 @Override  
 public User login(String name,String password) {  
 System.*out*.println("service");  
 UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken  
 = new UsernamePasswordAuthenticationToken(name,password);  
  
 Authentication authenticate = authenticationManager.  
 authenticate(usernamePasswordAuthenticationToken);//这里面就执行了UserDetialService  
  
 User user = (User) authenticate.getPrincipal();  
 redisTemplate.set("user",user.getPermissions().toString(),10\*60L);  
 return user;  
 }}

//这个时候如果你的请求头里有 所需要的token 那么就不需认证 直接过

然后就是两个异常处理器 一个是认证异常 一个是授权异常

认证：

@Component  
public class AuthenticationEntryPointImpl implements AuthenticationEntryPoint {  
 @Override  
 public void commence(HttpServletRequest request,  
 HttpServletResponse response,  
 AuthenticationException authException) throws IOException, ServletException {  
 MyResult result = new MyResult(500,"认证失败啊");  
  
 //认证失败处理异常  
 *renderString*(response, JSON.*toJSONString*(result));  
 }  
 public static void renderString(HttpServletResponse response,String msg){  
 response.setStatus(200);  
 response.setContentType("application/json");  
 response.setCharacterEncoding("utf-8");  
 try {  
 response.getWriter().print(msg);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

授权：

@Component  
public class AccessDeniedHandlerImpl implements AccessDeniedHandler {  
 @Override  
 public void handle(HttpServletRequest request,  
 HttpServletResponse response,  
 AccessDeniedException accessDeniedException) throws IOException, ServletException {  
 MyResult result = new MyResult(555,"授权失败");  
  
 //授权失败处理异常  
 *renderString*(response, JSON.*toJSONString*(result));  
 }  
 public static void renderString(HttpServletResponse response,String msg){  
 response.setStatus(200);  
 response.setContentType("application/json");  
 response.setCharacterEncoding("utf-8");  
 try {  
 response.getWriter().print(msg);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

完事之后注入到容器中 再配置类中搞

@Autowired  
AuthenticationEntryPointImpl authenticationEntryPoint;//认证异常  
@Autowired  
AccessDeniedHandlerImpl accessDeniedHandler; // 授权异常

//添加异常处理器  
http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);

这样 再登录的时候和权限不足的时候就会有特定的返回值 你懂的

然后 就是授权也可以自定义哦 其实就是模仿写的 开启授权的时候别忘了在配置类中开启授权

@EnableGlobalMethodSecurity(prePostEnabled = true)//这个是开启 权限哦

@Component("ex")  
public class MyExpressionRoot {  
  
  
 public final boolean hasAuthority(String authority){  
 //获取当前用户的权限  
 Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  
 User user = (User) authentication.getPrincipal();  
 ArrayList<String> permissions = user.getPermissions();  
  
 //判断用户权限集合中是否存在 authority  
 return permissions.contains(authority);  
 }  
}

于是乎你就可以在接口上加权限了哦

@GetMapping("test")  
//@PreAuthorize("hasAnyAuthority('test')")//使用之前要在配置类中添加开启注解  
@PreAuthorize("@ex.hasAuthority('test')")//使用之前要在配置类中添加开启注解  
  
public String test(){  
 return "66666666666";  
}  
  
@GetMapping("test2")  
@PreAuthorize("hasAnyAuthority('test')")  
public String test2(){  
 return "6666222222226666666";  
}

完结撒花了