Spring Security + Jwt 登录实现

> 💡 ps：如果有问题请麻烦指正下，感谢！！！

Spring Security是一个提供身份验证、授权和针对常见攻击的保护的框架。 凭借对保护命令式和反应式应用程序的一流支持，它是保护基于 Spring 的应用程序的事实标准。

技术栈

- Java 17
- SpringBoot `3.1.5`
- Security `6.1.5`
- Jwt `0.9.1`

登录访问由 Spring Security 和 Jwt 完成，用户权限通过角色实现。

初始

> 💡 Tips：通过下载由 Spring Initializr 准备的最小 SpringBoot+SpringSecurity 应用程序

spring security 引用 XML

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

新建一个控制类验证请求登录拦截

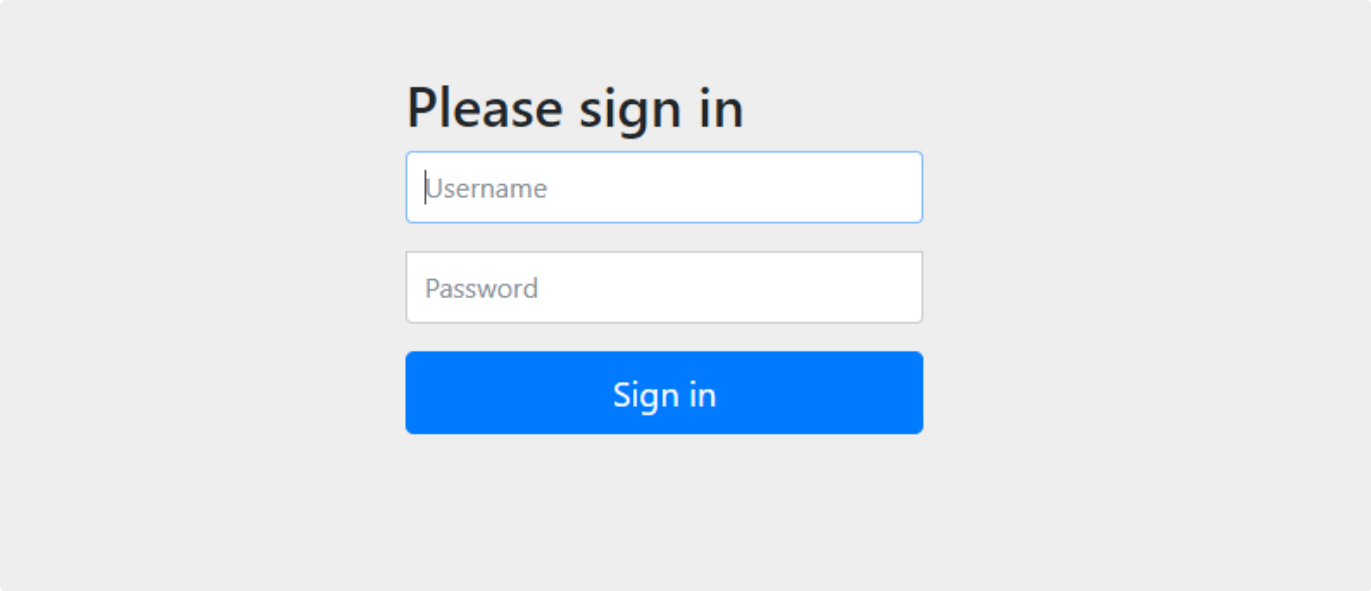
请求验证示例 Java

```java
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World";
    }
}
```

选中 HelloSecurityApplication.java 启动应用程序

控制台会打印对应 password `例如: Using generated security password: 55bcf6ab-f685-4a53-8102-8549aa1e2199`

执行访问 localhost:8080/hello



输入对应输入值进行访问，用户名默认为 `user`，密码为上述生成。

Security + Jwt 实现账号密码登录验证

Jwt 引用 XML

```xml
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

实现流程

1. 新建 SecurityFilterChain 实例，使用 SecurityFilterChain 来确定应为当前请求调用哪些Spring Security 实例。其中可以配置各种 Fileter 这些过滤器可用于许多不同的目的，例如身份验证、授权、漏洞利用保护等。 筛选器按特定顺序执行，以确保在正确的时间调用它们，例如，应在执行授权之前调用执行身份验证的 。
2. 新建自定义 Filter 过滤器添加到 SecurityFilterChain 过滤器链中下述代码中扩展了 OncePerRequestFilter 保证每个请求执行一次的筛选器基类调度。主要是为了通过 Jwt 验证请求中 Token 的合法性。

部分代码块实现

基础配置

SecurityConfig.java

Security核心配置 Java

```java
@EnableWebSecurity
@Configuration
public class SecurityConfig {

    private final SecurityProperties securityProperties;

    private final TokenCustomProperties tokenCustomProperties;

    private final JwtTokenUtil jwtTokenUtil;

    private final UserDetailsService userDetailsService;

    private final RestAuthenticationEntryPoint restAuthenticationEntryPoint;

    private final RestfulAccessDeniedHandler restfulAccessDeniedHandler;

    private final RedisUtil redisUtil;

    public SecurityConfig(SecurityProperties securityProperties,
                          TokenCustomProperties tokenCustomProperties,
                          JwtTokenUtil jwtTokenUtil,
                          UserDetailsService userDetailsService,
                          RestAuthenticationEntryPoint restAuthenticationEntryPoint,
```

JwtAuthenticationTokenFilter.java

在用户名和密码校验前添加的过滤器，如果请求中有jwt的token且有效，会取出token中的... Java

```java
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    private static final Logger logger = LoggerFactory.getLogger(JwtAuthenticationTokenFilter.class);

    private final TokenCustomProperties tokenCustomProperties;

    private final JwtTokenUtil jwtTokenUtil;

    private final UserDetailsService userDetailsService;

    private final RedisUtil redisUtil;

    public JwtAuthenticationTokenFilter(TokenCustomProperties tokenCustomProperties, JwtTokenUtil jwtTokenUtil, UserDetailsService userDetailsService, RedisUtil redisUtil) {
        this.tokenCustomProperties = tokenCustomProperties;
        this.jwtTokenUtil = jwtTokenUtil;
        this.userDetailsService = userDetailsService;
        this.redisUtil = redisUtil;
    }

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain chain) throws S
```

RestAuthenticationEntryPoint.java

当未登录或者token失效访问接口时，自定义的返回结果 Java

```java
@Component
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");
        response.getWriter().println(JSONUtil.parse(R.fail(authException.getMessage())));
        response.getWriter().flush();
    }
}
```

RestfulAccessDeniedHandler.java

当访问接口没有权限时，自定义的返回结果 Java

```java
@Component
public class RestfulAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request,
                       HttpServletResponse response,
                       AccessDeniedException e) throws IOException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");
        response.getWriter().println(JSONUtil.parse(R.fail(e.getMessage())));
        response.getWriter().flush();
    }
}
```

JwtTokenUtil.java

JwtToken生成的工具类 Java

```java
@Component
public class JwtTokenUtil {

    private static final Logger logger = LoggerFactory.getLogger(JwtTokenUtil.class);
    private static final String CLAIM_KEY_USERNAME = "sub";
    private static final String CLAIM_KEY_CREATED = "created";

    @Resource
    private TokenCustomProperties tokenCustomProperties;

    /**
     * 根据负责生成JWT的token
     */
    private String generateToken(Map<String, Object> claims) {
        return Jwts.builder()
                .setClaims(claims)
                .setExpiration(generateExpirationDate())
                .signWith(SignatureAlgorithm.HS512, tokenCustomProperties.getSecret())
                .compact();
    }

    /**
     * 从token中获取JWT中的负载
```

TokenCustomProperties.java

yml 配置项 Java

```java
@Getter
@Setter
@ConfigurationProperties(prefix = "token")
public class TokenCustomProperties {

    /**
     * 自定义请求头
     */
    private String header;

    /**
     * token 密钥
     */
    private String secret;

    /**
     * 过期时间(ms)
     */
    private Integer expireTime;

    /**
     * token前缀
     */
    private String prefix;
}
```

application.yml

其余配置自定义如msql连接等。

token 配置 YAML

```yaml
token:
  prefix: Bearer
  header: Authorization
  secret: dandelion_dev
  # 过期时间 m
  expireTime: 600
```

部分主要业务实现

UserDetailServiceImpl.java

主要是为实现 UserDetailsService 上述 Jwt 中会调用该方法获取自定义用户 Java

```java
@Service
public class UserDetailServiceImpl implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // 返回对应 用户信息
        return null;
    }

}
```

LoginController.java

登录控制器 Java

```java
@RestController
@RequestMapping("/login")
public class LoginController {

    @Resource
    private LoginService loginService;

    /**
     * 登录
     *
     * @param loginRequest 登录参数对象{"userName":"admin","password":"123456"}
     * @return .
     */
    public R<LoginVo> login(@RequestBody LoginRequest loginRequest) {
        String userName = loginRequest.getUserName();
        String password = loginRequest.getPassword();
        String token = loginService.login(userName, password);
        return R.success(LoginVo.builder().token(token).build());
    }

    /**
     * 登出
     *
     * @return .
     */
    @PostMapping("/logout")
    public R<Boolean> logout() {
        return R.success("登出成功", loginService.logout());
    }
}
```

LoginService.java

根据 LoginServiceImpl.java 直接反写即可。

LoginRequest.java

自行定义主要就是 `{"userName":"admin","password":"123456"}`

R.java

同上自定义

LoginVo.ava

同上自定义

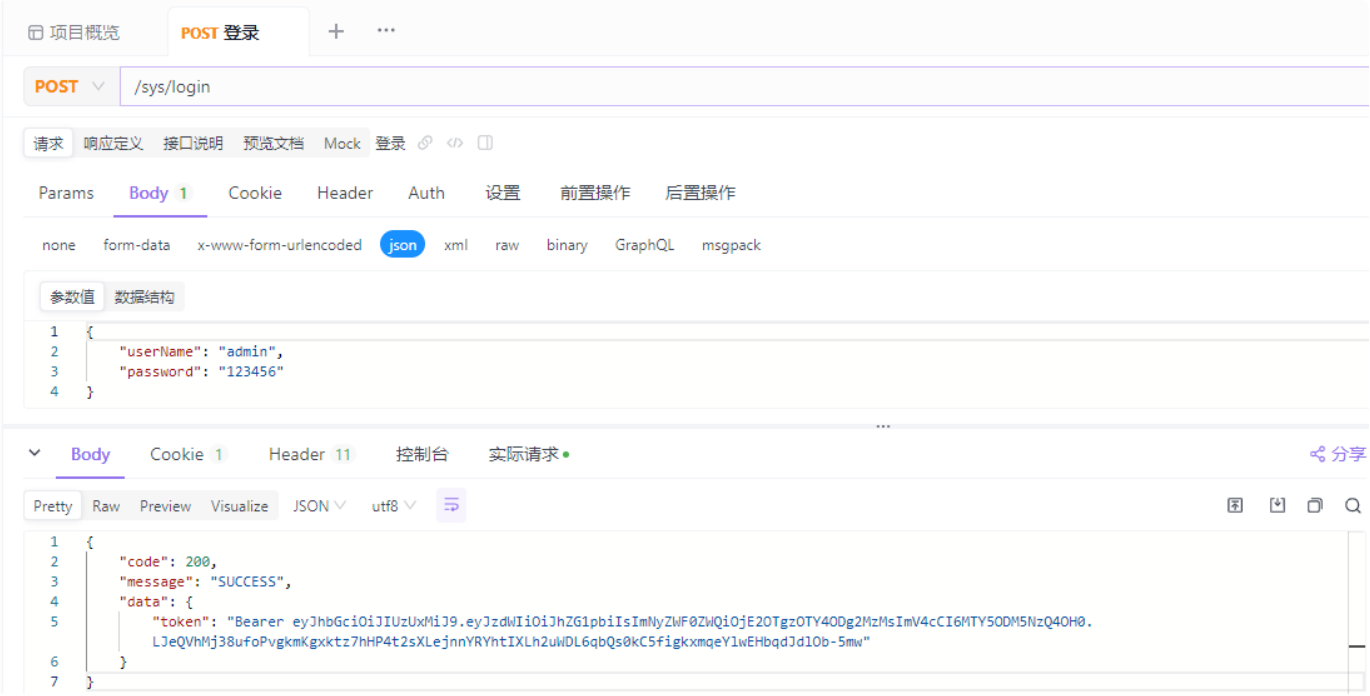
LoginServiceImpl.java

登录实现 Java

```java
@Service
public class LoginServiceImpl implements LoginService {

    @Resource
    private TokenCustomProperties tokenCustomProperties;
    @Resource
    private UserDetailsService userDetailsService;
    @Resource
    private PasswordEncoder passwordEncoder;
    @Resource
    private JwtTokenUtil jwtTokenUtil;
    @Resource
    private RedisUtil redisUtil;

    @Override
    public String login(String username, String password) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        String userDetailsPassword = userDetails.getPassword();
        boolean matches = passwordEncoder.matches(password, userDetailsPassword);
        if(!matches){
            throw new BadCredentialsException("密码不正确");
        }
        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(authentication);
        return tokenCustomProperties.getPrefix().concat(" ").concat(jwtTokenUtil.generateToken(userDetails));
    }

    @Override
    public boolean logout() {
        // 获取SecurityContextHolder里的用户id
        UsernamePasswordAuthenticationToken authentication =
                (UsernamePasswordAuthenticationToken) SecurityContextHolder.getContext().getAuthentication();
        UserDetailImpl userDetails = (UserDetailImpl) authentication.getPrincipal();
        String username= userDetails.getUsername();
        redisUtil.del(RedisConstant.TOKEN.concat(username));
        return true;
    }
```

```
38    }
```

登录验证请求 localhost:8080/login 获取对应账户 token 值



验证

未使用 token 返回请求示例

使用正确 token 返回请求示例


GET
/redis/get/{key}
请求
响应定义
接口说明
预览文档
Mock
get
Params 1
Body
Cookie
Header 1
Auth
设置
前置操作
后置操作
参数名
参数值
类型
说明
Authorization
Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pl
string
Body
Cookie 1
Header 11
控制台
实际请求
分享
Pretty
Raw
Preview
Visualize
JSON
utf8
{
"code": 200,
"message": "SUCCESS",
"data": "张三"
}