

1 问题 如何保护我们的程序？

问题：

- 我们的程序需不需要保护？
- 如果需要保护，如何保护？

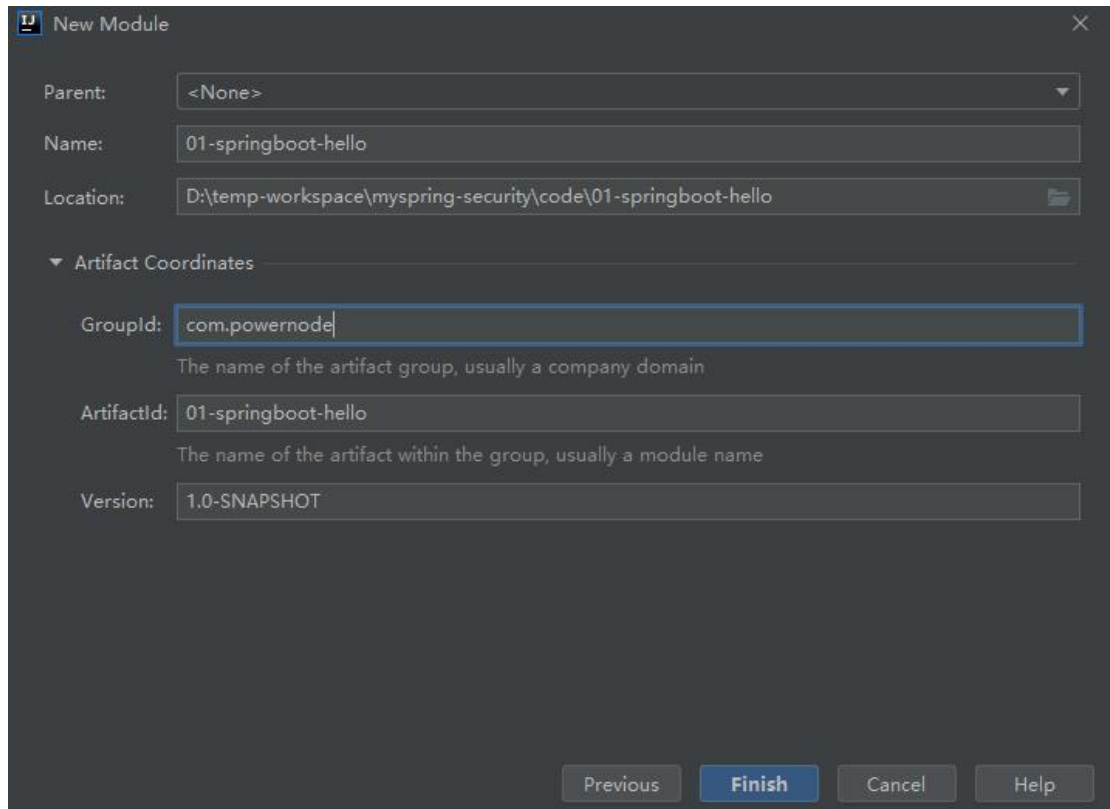
1.1 创建 code 目录

目的：后面的 security 工程均在此目录下学习

创建 code 目录，并使用 idea 打开

1.2 不使用安全框架的 springboot web 程序

1.2.1 新建子模块 springboot-01-hello



1.2.2 添加依赖和 maven 插件等

pom.xml 中部分内容如下，此处使用 bootpom 模板创建并粘贴

```
<parent>
  <artifactId>spring-boot-starter-parent</artifactId>
  <groupId>org.springframework.boot</groupId>
  <version>2.6.13</version>
  <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>

<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <!--springboot web 程序 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!--springboot 单元测试-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <!--自动生成 get、set 和日志对象的 lombok-->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
```

```
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

1.2.3 新建启动类

com.powernode 包下新建启动类 Application，可以使用 bootmain 模板，学员自行创建

1.2.4 新建三个 controller

com.powernode.controller 包下新建三个 controller

学生

```
@RestController
@RequestMapping("/student")
public class StudentController {
    @GetMapping("/query")
    public String queryInfo(){
        return "I am a student,My name is Eric!";
    }
}
```

教师

```
@RestController
@RequestMapping("/teacher")
public class TeacherController {

    @GetMapping("/query")
    public String queryInfo(){
        return "I am a teacher,My name is Thomas!";
    }
}
```

管理员

```
@RestController
@RequestMapping("/admin")
public class AdminController {

    @GetMapping("/query")
    public String queryInfo(){
        return "I am a administrator, My name is Obama!";
    }
}
```

1.2.5 启动程序

启动程序的快捷键: **Ctrl+Shift+F10**

1.2.6 访问程序

通过浏览器访问程序

<http://localhost:8080/student/query>

<http://localhost:8080/teacher/query>

<http://localhost:8080/admin/query>

通过 curl 访问（可以使用 git bash）

```
curl -X GET localhost:8080/teacher/query
```

返回效果如下：

```
$ curl -X GET localhost:8080/teacher/query
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   100      32   100      32    0     0    1684      0  --:--:-- --:--:-- --:--:--  1882t am a teacher,My name is
rthomas
```

curl 的使用可以作为扩展学习，不讲

curl 是常用的命令行工具，用来请求 Web 服务器。它的名字就是命令行（Command Line）下的 URL 工具的意思。

它的功能非常强大，命令行参数多达几十种。如果熟练的话，完全可以取代 Postman 这一类的图形界面工具。

cURL
Command Line URL viewer

参考学习网站：

<https://catonmat.net/cookbooks/curl>

<https://www.ruanyifeng.com/blog/2019/09/curl-reference.html>

<https://www.ruanyifeng.com/blog/2011/09/curl.html>

https://blog.csdn.net/angle_chen123/article/details/120675472

1.2.7 结论

此示例说明：

没有加入安全框架的 springboot web 程序，默认所有资源均不受保护。

1.2.8 问题

我们的项目很多资源必须被保护起来，如何保护？引入安全框架

2 认证授权等基本概念

2.1 认证（authentication）

2.1.1 系统为什么要认证？

认证是为了保护系统的隐私数据与资源，用户的身份合法方可访问该系统的资源。

2.1.2 什么是认证（登录）？

认证：用户认证就是判断一个用户的身份是否合法的过程。

2.1.3 常见的用户身份认证方式

- 用户名密码登录
- 二维码登录
- 手机短信登录
- 指纹认证
- 人脸识别
- 等等...

2.2 会话（session）

2.2.1 什么是会话

用户认证通过后，为了避免用户的每次操作都进行认证可将用户的信息保存在会话中。会话就是系统为了保持当前用户的登录状态所提供的机制，常见的有基于 **session** 方式、基于 **token** 方式等。

2.2.2 基于 session 的认证方式

它的交互流程是，用户认证成功后，在服务端生成用户相关的数据保存在 session(当前会话)中，发给客户端的 session_id 存放到 cookie 中，这样用户客户端请求时带上 session_id 就可以验证服务器端是否存在 session 数据，以此完成用户的合法校验，当用户退出系统或 session 过期销毁时，客户端的 session_id 也就无效了。

2.2.3 基于 token 的认证方式

它的交互流程是，用户认证成功后，服务端生成一个 token 发给客户端，客户端可以放到 cookie 或 localStorage 等存储中，每次请求时带上 token，服务端收到 token 通过验证后即可确认用户身份。可以使用 Redis 存储用户信息（分布式中共享 session）。

基于 session 的认证方式由 Servlet 规范定制，服务端要存储 session 信息需要占用内存资源，客户端需要支持 cookie；基于 token 的方式则一般不需要服务端存储 token，并且不限制客户端的存储方式。如今移动互联网时代更多类型的客户端需要接入系统，系统多是采用前后端分离的架构进行实现，所以基于 token 的方式更适合。

2.3 授权(authorization)

2.3.1 为什么要授权（控制资源被访问）？

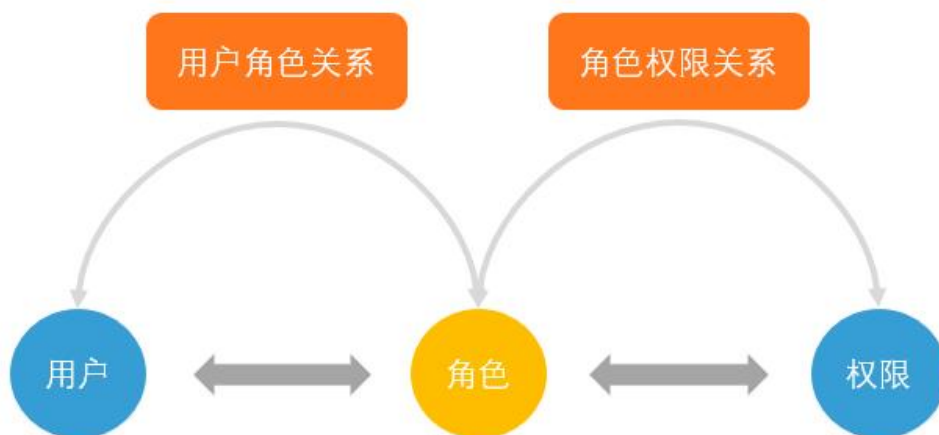
因为不同的用户可以访问的资源是不一样的。

2.3.2 什么是授权（给用户颁发权限）

授权：授权是用户认证通过后，根据用户的权限来控制用户访问资源的过程。

拥有资源的访问权限则正常访问，没有权限则拒绝访问。

2.4 RBAC（Role-Based Access Control） 基于角色的访问控制



用户，角色，权限 本质：就是把权限打包给角色（角色拥有一组权限），分配给用户（用户拥有多个角色）。

最少包括五张表（用户表、角色表、用户角色表、权限表、角色权限表）

3 java 的安全框架实现

主要有三种方式：

- Shiro：轻量级的安全框架，提供认证、授权、会话管理、密码管理、缓存管理等功能
- Spring Security：功能比 Shiro 强大，更复杂，权限控制细粒度更高，对 OAuth2 支持更好，与 Spring 框架无缝集合，使 Spring Boot 集成很快捷。
- 自己写：基于过滤器（filter）和 AOP 来实现，难度大，没必要。

4 Spring Security

4.1 什么是 Spring Security

Spring Security 是一个能够为基于 Spring 的企业应用系统提供**声明式（注解）的安全访**

访问控制解决方案的安全框架。它提供了一组可以在 Spring 应用上下文中配置的 Bean，充分利用了 Spring IoC，DI（控制反转 Inversion of Control ,DI:Dependency Injection 依赖注入）和 AOP（面向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

以上解释来源于百度百科。可以一句话来概括：

SpringSecurity 是一个安全框架。

4.2 官方网址

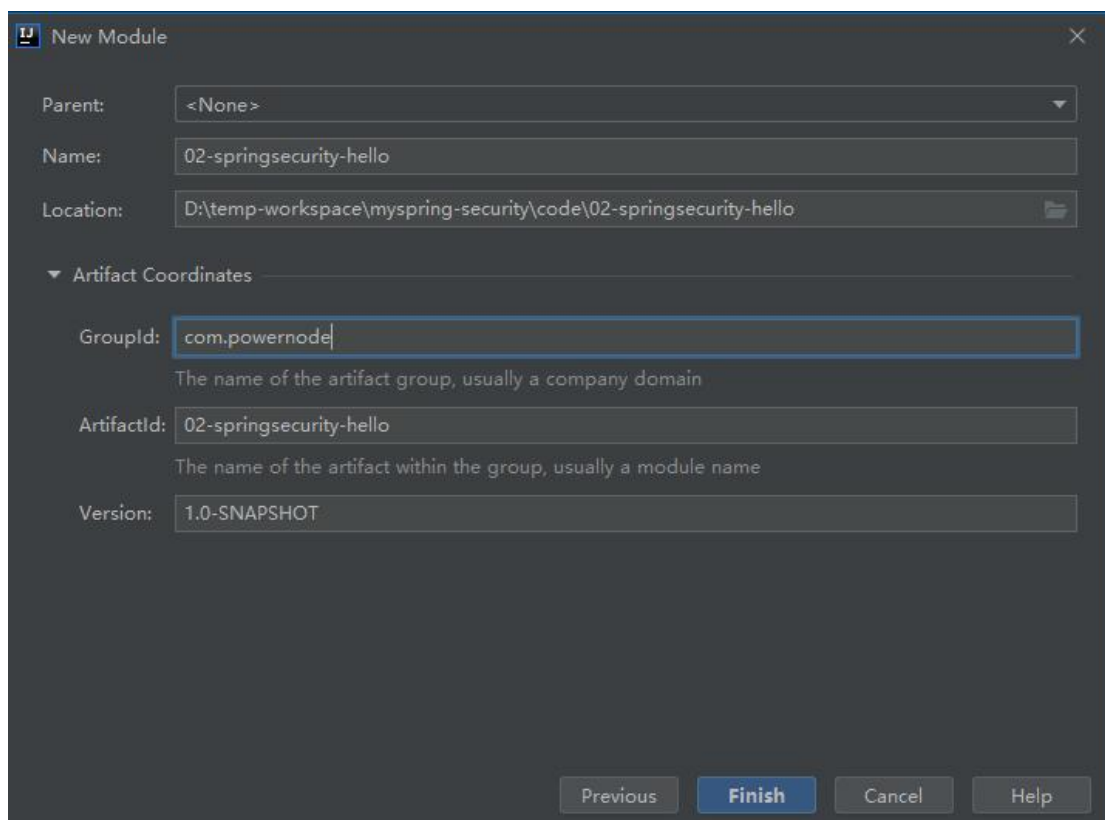
<https://spring.io/projects/spring-security>

查看下官网

5 认证入门

5.1 安全入门项目

5.1.1 新建模块 springsecurity-02-hello



5.1.2 添加依赖和 maven 插件

此处可以使用 bootpom 模板创建临时文件，并拷贝。

5.1.3 添加 spring-boot-starter-security 依赖

```
<dependency>  
  <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

5.1.4 新建启动类

新建启动类 Application，学员自行创建

5.1.5 新建三个 controller

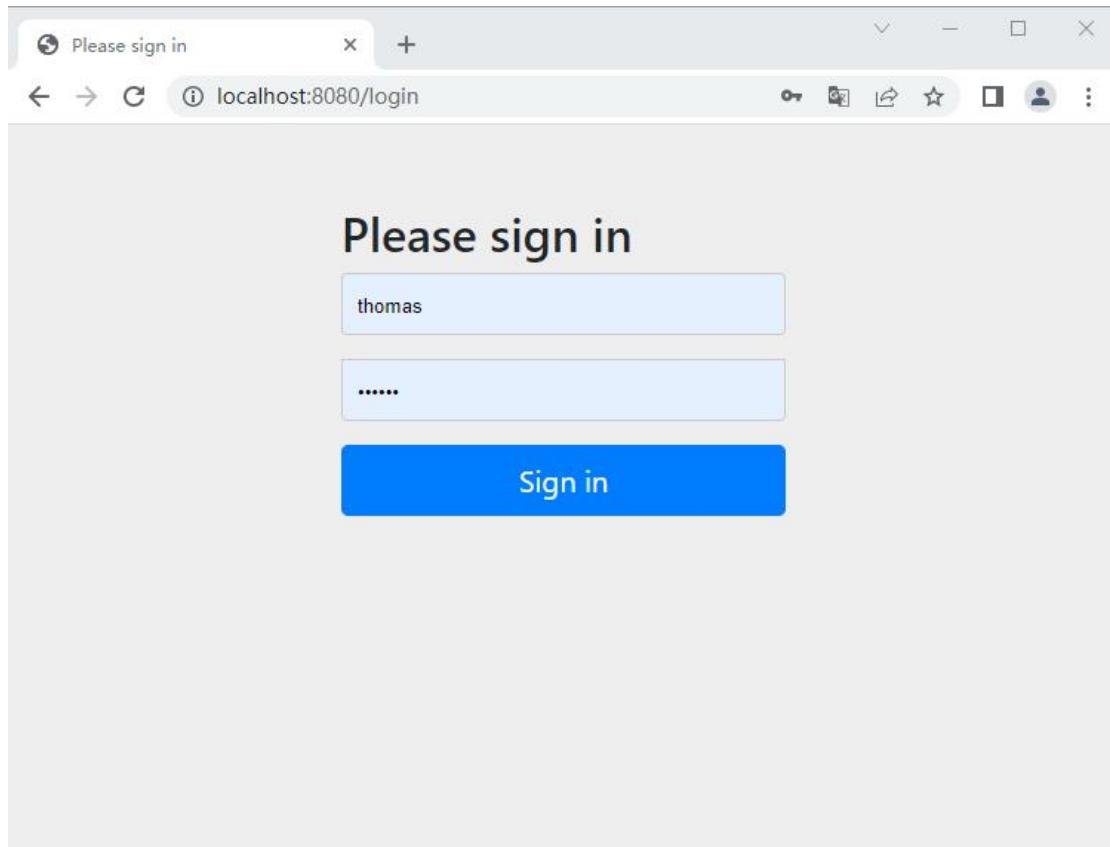
参照 1.2.4 创建，可以直接拷贝过来

小提示：重命名快捷键（shift+F6），移动文件快捷键 F6

5.1.6 启动程序，并使用浏览器访问

<http://localhost:8080/student/query>

系统会跳转到登录页面



运行结果说明：

spring Security 默认拦截了所有请求，但登录退出不拦截

5.1.7 登录系统

使用默认用户 **user** 登录系统，密码是随机生成的 UUID 字符串，可以在控制台（console）上找到。

```
2022-11-03 13:45:12.542 INFO 9556 --- [main] o.apache.catalina.core.StandardService : Starting service [tomcat]
2022-11-03 13:45:12.543 INFO 9556 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.68]
2022-11-03 13:45:12.897 INFO 9556 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-11-03 13:45:12.897 INFO 9556 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complete
2022-11-03 13:45:13.386 WARN 9556 --- [main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: b9354440-915e-4c25-b7ed-dd8e53a8a1f8

This generated password is for development use only. Your security configuration must be updated before running your application in production.

2022-11-03 13:45:13.593 INFO 9556 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework
2022-11-03 13:45:13.794 INFO 9556 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with conte
```

登录系统，再次访问：

<http://localhost:8080/student/query>

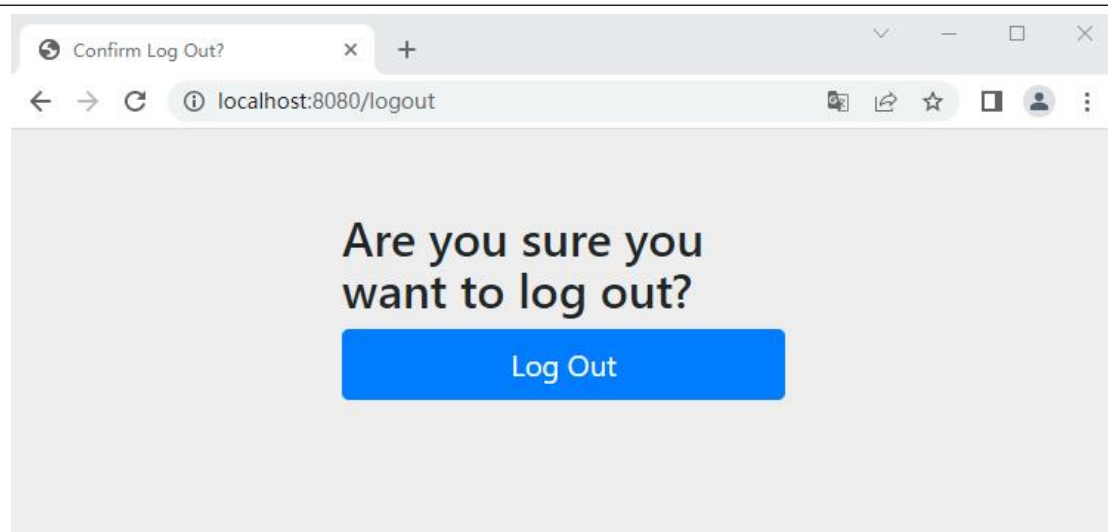
<http://localhost:8080/teacher/query>

<http://localhost:8080/admin/query>

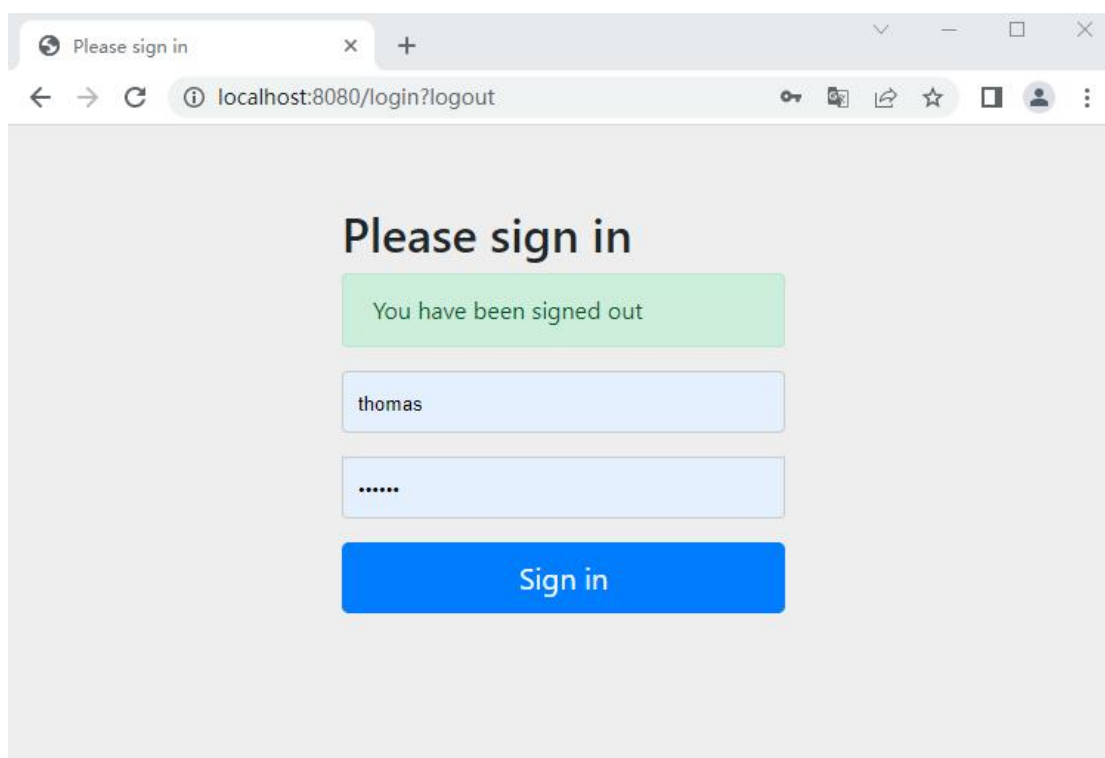
发现登录后的用户均可以正常访问

5.1.8 退出系统

<http://localhost:8080/logout>



单击 Log Out 按钮，成功退出



5.1.9 结论

引入 spring-boot-starter-security 依赖后，项目中除登录退出外所有资源都会被保护起来
认证（登录）用户可以访问所有资源，不经过认证用户任何资源也访问不了。

5.1.10 问题

所有资源均已保护，但是用户只用一个，密码是随机的，只能在开发环境使用

5.2 使用配置文件配置用户名和密码

5.2.1 新建模块 springsecurity-03-configfile

模仿 5.1.1 新建即可。

5.2.2 添加安全依赖

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

5.2.3 新建启动类

新建启动类 Application，学员自行创建

5.2.4 新建三个 controller

参照 1.2.4 创建，可以直接拷贝过来

5.2.5 新建配置文件 application.yml

添加 spring security 配置信息

```
spring:
  security:
    user:
      name: admin
      password: 123456
```

5.2.6 启动运行并使用浏览器测试

发现使用配置文件中的用户名和密码可以正常访问。

配置文件中配置用户后，默认的用户 user 就没有了。

示例说明：可以通过配置文件配置用户和密码，解决了使用随机生成密码的问题。

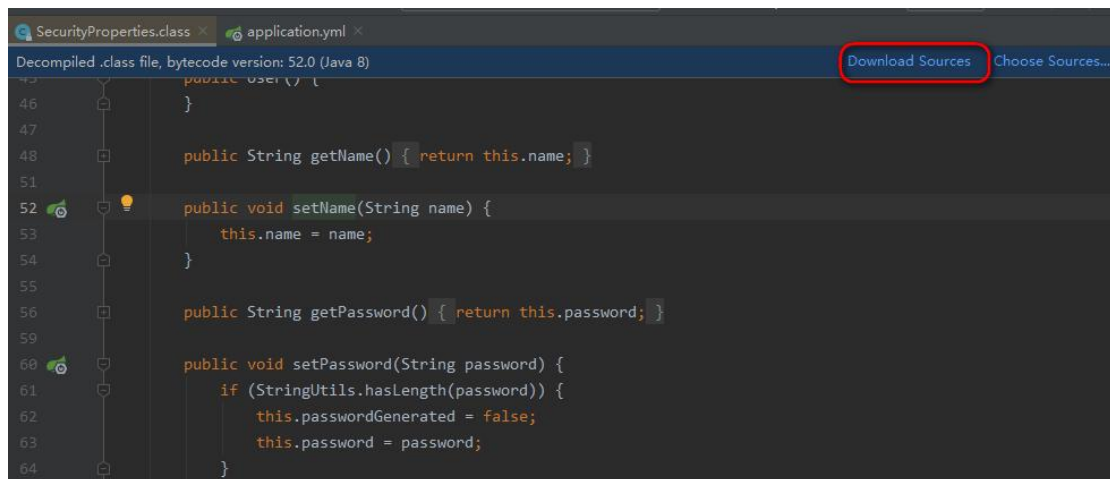
5.2.7 查看源码

问题：

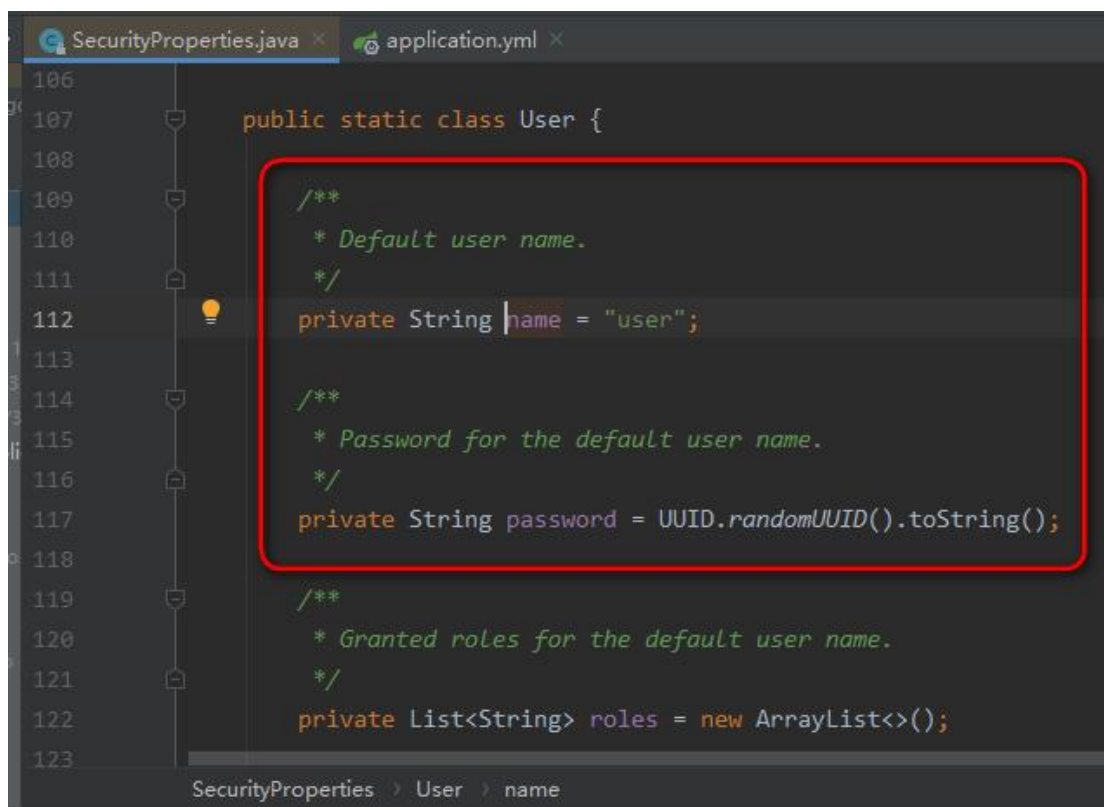
- 查看源码的快捷键是什么？
- 反编译生成代码和源码的区别？
- 如何下载源代码？
- 下载后的源代码存储到哪去了？

快捷键：**ctrl+鼠标左键**

application.yml 中将鼠标指定到 name 那，按住 ctrl 键，单击鼠标左键



单击上图中 Download Sources，下载源码文件



```

106
107 public static class User {
108
109     /**
110      * Default user name.
111      */
112     private String name = "user";
113
114     /**
115      * Password for the default user name.
116      */
117     private String password = UUID.randomUUID().toString();
118
119     /**
120      * Granted roles for the default user name.
121      */
122     private List<String> roles = new ArrayList<>();
123

```

可以看到默认用户名为 user，密码为使用 UUID 随机生成的字符串。

5.2.8 问题

Spring Security 配置文件中默认配置用户是单一的用户，大部分系统都有多个用户，多个用户如何配置？

5.3 基于内存的多用户管理

5.3.1 新建模块 springsecurity-04-inmemory

5.3.2 添加依赖

```

<dependency>
    <groupId>org.springframework.boot</groupId>

```

```
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

5.3.3 新建启动类

新建启动类 Application，学员自行创建

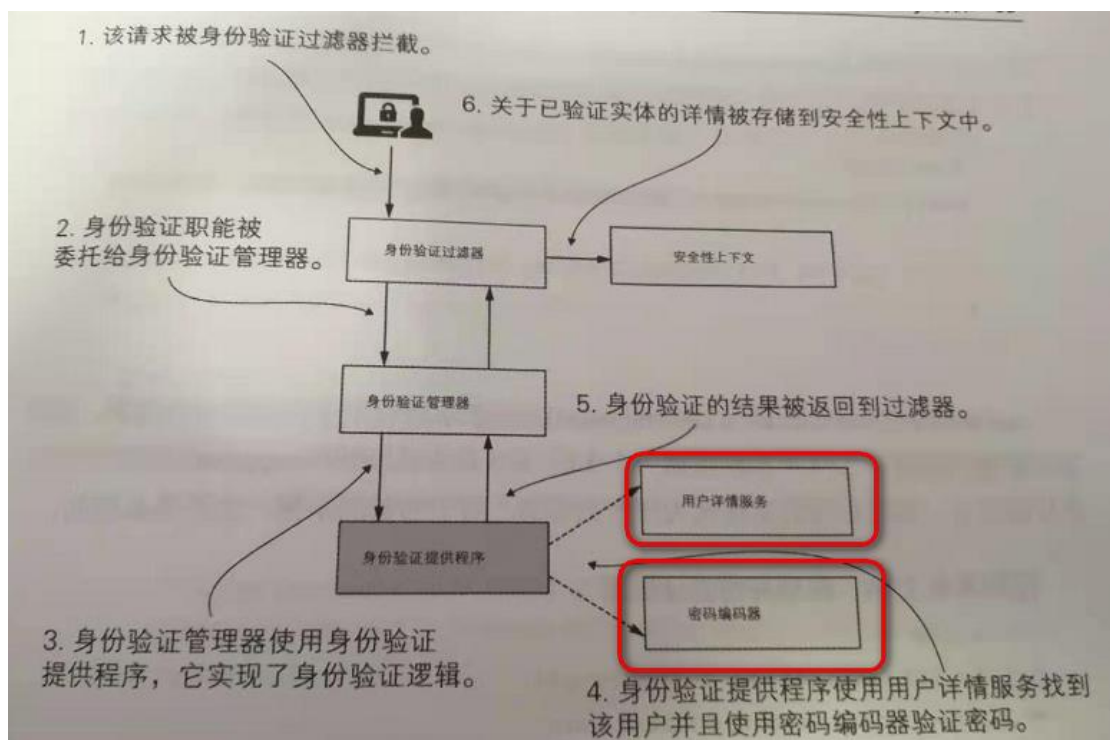
5.3.4 新建三个 controller

参照 1.2.4 创建，可以直接拷贝过来

5.3.5 新建配置文件 application.yml

参照 5.2.5 ，可以直接拷贝内容

5.3.6 新建配置类

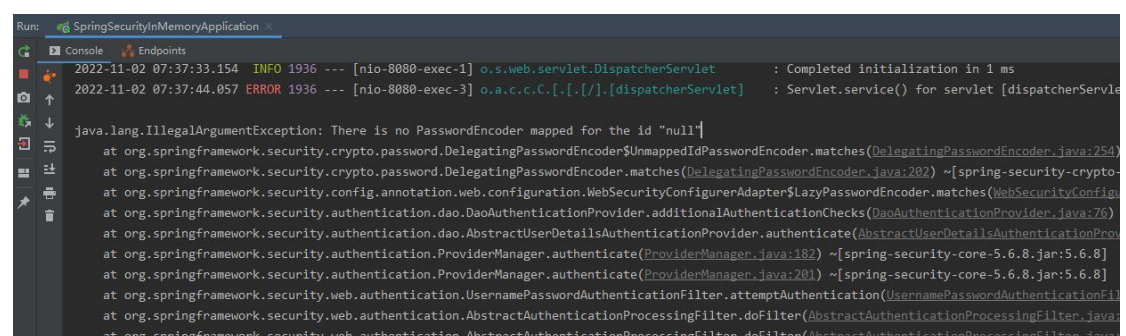


com.powernode.config 包下新建配置类 MySecurityUserConfig，如下：

```
@Configuration
public class MySecurityUserConfig {
    @Bean
    public UserDetailsService userDetailsService() {
        // 使用 org.springframework.security.core.userdetails.User 类来
        // 定义用户
        // 定义两个用户
        UserDetails user1 =
        User.builder().username("eric").password("123456").roles("student").
        build();
        UserDetails user2 =
        User.builder().username("thomas").password("123456").roles("teacher"
        ).build();
        // 创建两个用户
        InMemoryUserDetailsManager userDetailsManager = new
        InMemoryUserDetailsManager();
        userDetailsManager.createUser(user1);
        userDetailsManager.createUser(user2);
        return userDetailsManager;
    }
}
```

5.3.7 启动程序测试

登录页面输入用户名（thomas）和密码（123456），然后单击登录后，控制台报错，如下：



```
Run: SpringSecurityInMemoryApplication
2022-11-02 07:37:33.154 INFO 1936 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-11-02 07:37:44.057 ERROR 1936 --- [nio-8080-exec-3] o.a.c.c.C.[.[./].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"] with root cause: java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
at org.springframework.security.crypto.password.DelegatingPasswordEncoder$UnmappedIdPasswordEncoder.matches(DelegatingPasswordEncoder.java:254)
at org.springframework.security.crypto.password.DelegatingPasswordEncoder.matches(DelegatingPasswordEncoder.java:202) ~[spring-security-crypto-5.6.8.jar:5.6.8]
at org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter$LazyPasswordEncoder.matches(WebSecurityConfigurerAdapter$LazyPasswordEncoder.java:76)
at org.springframework.security.authentication.dao.DaoAuthenticationProvider.additionalAuthenticationChecks(DaoAuthenticationProvider.java:76)
at org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider.authenticate(AbstractUserDetailsAuthenticationProvider.java:182)
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:182) ~[spring-security-core-5.6.8.jar:5.6.8]
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:201) ~[spring-security-core-5.6.8.jar:5.6.8]
at org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter.attemptAuthentication(UsernamePasswordAuthenticationFilter.java:173)
at org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter.doFilter(AbstractAuthenticationProcessingFilter.java:272)
at org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter.doFilter(AbstractAuthenticationProcessingFilter.java:242)
```

报错的原因如下：

这个是因为 **spring Security** 强制要使用密码加密，当然我们也可以不加密，但是官方要求是不管你是否加密，都必须配置一个密码编码（加密）器

5.3.8 添加密码加密器 bean 但是不对密码加密

在 **MySecurityUserConfig** 类中加入以下 bean

```
/*
 * 从 Spring5 开始，强制要求密码要加密
 * 如果非不想加密，可以使用一个过期的 PasswordEncoder 的实例
NoOpPasswordEncoder,
 * 但是不建议这么做，毕竟不安全。
 *
 * @return
 */
@Bean
public PasswordEncoder passwordEncoder(){
    //不对密码进行加密，使用明文
    return NoOpPasswordEncoder.getInstance();
}
```

重启程序再次使用 **thomas/123456** 登录测试，可以登录正常访问了。

使用 **admin/123456** 登录，登录不成功，说明：我们只要添加了安全配置类，那么我们在 **yml** 里面的配置就失效了

此处可以查看一下 **NoOpPasswordEncoder** 源码，再看一下单例模式，加密和密码对比方法

英文小提示：

明文：plaintext

密文：ciphertext

问题:

- 密码为什么要加密? 加密的方式有哪些? 涉及到密码加密问题
- NoOpPasswordEncoder 此类已经过期, 而且还没有加密, 如何解决? 下章解决
- 以学生身份登录, 发现不但可以访问学生的页面, 还可以访问教师的页面和管理员的页面, 如何解决? 权限问题, 后面解决
- 如果要动态的创建用户, 或者修改密码等(不是把用户名和密码写死到代码中), 怎么办? 认证信息要存储到数据库中。

下面一章讲解密码加密问题

6 密码处理

6.1 为什么要加密?

csdn 密码泄露事件

泄露事件经过: <https://www.williamlong.info/archives/2933.html>

泄露数据分析: <https://blog.csdn.net/crazyhacking/article/details/10443849>

6.2 加密方案

密码加密一般使用散列函数, 又称散列算法, 哈希函数, 这些函数都是单向函数(从明文到密文, 反之不行)

常用的散列算法有 MD5 和 SHA

Spring Security 提供多种密码加密方案, 基本上都实现了 PasswordEncoder 接口, 官方推荐使用 BCryptPasswordEncoder

6.3 BCryptPasswordEncoder 类初体验

拷贝 springsecurity-04-inmemory 工程，重命名为 springsecurity-05-encode

test/java 下新建包 com.powernode.password,在该包下新建测试类 PasswordEncoderTest，如下

```
@Slf4j
public class PasswordEncoderTest {

    @Test
    @DisplayName("测试加密类 BCryptPasswordEncoder")
    void testPassword(){
        BCryptPasswordEncoder bCryptPasswordEncoder = new
BCryptPasswordEncoder();
        //加密（明文到密文）
        String encode1 = bCryptPasswordEncoder.encode("123456");
        Log.info("encode1:"+encode1);
        String encode2 = bCryptPasswordEncoder.encode("123456");
        Log.info("encode2:"+encode2);
        String encode3 = bCryptPasswordEncoder.encode("123456");
        Log.info("encode3:"+encode3);
        //匹配方法，判断明文经过加密后是否和密文一样
        boolean result1 = bCryptPasswordEncoder.matches("123456",
encode1);
        boolean result2 = bCryptPasswordEncoder.matches("123456",
encode1);
        boolean result3 = bCryptPasswordEncoder.matches("123456",
encode1);
    }
}
```

```
        log.info(result1+" "+result2+" "+result3);  
        assertTrue(result1);  
        assertTrue(result2);  
        assertTrue(result3);  
    }  
}
```

查看控制台发现特点是：相同的字符串加密之后的结果都不一样，但是比较的时候是一样的，因为加了盐（salt）了。

小提示：

- 开发代码时不允许使用 **main** 方法测试，而是使用单元测试来测试
- 代码中一般不允许使用 **System.out.println** 直接输出，而是使用日志输出
- 单元测试尽量使用断言，而不是使用 **System.out.println** 输出

6.4 使用加密器并且加密

修改 MySecurityUserConfig 类中的加密器 bean

```
@Bean  
public PasswordEncoder passwordEncoder(){  
    //使用加密算法对密码进行加密  
    return new BCryptPasswordEncoder();  
}
```

启动程序测试，发现不能正常登录

原因是输入的密码是进行加密了，但是系统中定义的用户密码没有加密

将用户密码修改成密文，如下

```
@Configuration
```



```
public class MySecurityUserConfig {
    @Bean
    public UserDetailsService userDetailsService() {
        // 使用 org.springframework.security.core.userdetails.User 类来
        // 定义用户
        // 定义两个用户
        UserDetails user1 = User.builder()
            .username("eric")
            .password(passwordEncoder().encode("123456"))
            .roles("student")
            .build();
        UserDetails user2 = User.builder()
            .username("thomas")
            .password(passwordEncoder().encode("123456"))
            .roles("teacher")
            .build();
        // 创建两个用户
        InMemoryUserDetailsManager userDetailsManager = new
        InMemoryUserDetailsManager();
        userDetailsManager.createUser(user1);
        userDetailsManager.createUser(user2);
        return userDetailsManager;
    }

    /*
     * 从 Spring5 开始，强制要求密码要加密
     * @return
     */
    @Bean
    public PasswordEncoder passwordEncoder(){
        // 使用加密算法对密码进行加密
        return new BCryptPasswordEncoder();
    }
}
```

重启程序，再次测试即可，发现登录和访问没问题了

7 查看当前登录用户信息及配置用户权限

复制 springsecurity-05-encode ,复制后为 springsecurity-06-loginuser

7.1 获取当前登录用户信息

新建一个 controller

```
@RestController
public class CurrentLoginUserInfoController {

    /**
     * 从当前请求对象中获取
     */
    @GetMapping("/getLoginUserInfo")
    public Principal getLoginUserInfo(Principal principle){
        return principle;
    }

    /**
     * 从当前请求对象中获取
     */
    @GetMapping("/getLoginUserInfo1")
    public Authentication getLoginUserInfo1(Authentication
authentication){
        return authentication;
    }

    /**
```

```
* 从 SecurityContextHolder 获取
* @return
*/
@GetMapping("/getLoginUserInfo2")
public Authentication getLoginUserInfo(){
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

    return authentication;
}
}
```

注意 Authentication 接口继承自 Principal

重启程序，访问

<http://localhost:8080/getLoginUserInfo>

<http://localhost:8080/getLoginUserInfo1>

<http://localhost:8080/getLoginUserInfo2>

运行结果

```
{
  "authorities": [{
    "authority": "ROLE_teacher"
  }],
  "details": {
    "remoteAddress": "0:0:0:0:0:0:1",
    "sessionId": "34E452050095348E6306CF95B2025CD9"
  },
}
```

```
"authenticated": true,
"principal": {
  "password": null,
  "username": "thomas",
  "authorities": [{
    "authority": "ROLE_teacher"
  }],
  "accountNonExpired": true,
  "accountNonLocked": true,
  "credentialsNonExpired": true,
  "enabled": true
},
"credentials": null,
"name": "thomas"
}
```

- Principal 定义认证的而用户，如果用户使用用户名和密码方式登录，principal 通常就是一个 UserDetails（后面再说）
- Credentials: 登录凭证，一般就是指密码。当用户登录成功之后，登录凭证会被自动擦除，以方式泄露。
- authorities: 用户被授予的权限信息。

7.2 配置用户权限

配置用户权限有两种方式:

- 配置 roles
- 配置 authorities

注意事项:

- 如果给一个用户同时配置 roles 和 authorities，哪个写在后面哪个起作用
- 配置 roles 时，权限名会加上 ROLE_。

修改 WebSecurityConfig 代码中的

```
// 注意 1 哪个写在后面哪个起作用 2 角色变成权限后会加一个 ROLE_前缀，比如 ROLE_teacher
//      UserDetails user2 = User.builder()
//      .username("thomas")
//      .password(passwordEncoder().encode("123456"))
//      .authorities("teacher:add","teacher:update")
//      .roles("teacher")
//      .build();
UserDetails user2 = User.builder()
    .username("thomas")
    .password(passwordEncoder().encode("123456"))
    .roles("teacher")
    .authorities("teacher:add","teacher:update")
    .build();
```

重启程序使用 thomas 登录，然后查看用户认证信息

<http://localhost:8080/getLoginUserInfo>

可以看到 authorities 的情况。

从设计层面讲，角色和权限是两个完全不同的东西

从代码层面来讲，角色和权限并没有太大区别，特别是在 Spring Security 中

8 授权（对 URL 进行授权）

上面讲的实现了认证功能，但是受保护的资源是默认的，默认所有认证（登录）用户均可以访问所有资源，不能根据实际情况进行角色管理，要实现授权功能，需重写 WebSecurityConfigurerAdapter 中的一个 configure 方法

复制 springsecurity-06-loginuser 工程，然后改名为 springsecurity-07-url

新建 WebSecurityConfig 类，重写 configure(HttpSecurity http)方法

WebSecurityConfig 完整代码如下：

```
@Configuration
@Slf4j
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            //角色 student 或者 teacher 都可以访问/student/** 这样的 url
            .mvcMatchers("/student/**").hasAnyRole("student",
"teacher")
            // 角色 teacher 可以访问 teacher/**
            .mvcMatchers("/teacher/**").hasRole("teacher")
            //权限 admin:query 可以访问/admin**
            .mvcMatchers("/admin/**").hasAuthority("admin:query")
        )
        //角色 teacher 或者权限 admin:query 觉可以访问 admin/**
        .mvcMatchers("/admin/**").access("hasRole('teacher')
or hasAuthority('admin:query')")
        //任何请求均需要认证
        .anyRequest().authenticated();
        //使用表单登录
        http.formLogin();
    }
}
```

使用 admin 登录，访问

<http://localhost:8080/teacher/query>

<http://localhost:8080/student/query>

<http://localhost:8080/admin/query>

分别查看效果，实现权限控制

上面是对 URL 资源进行控制，就是哪些权限可以访问哪些 URL。

9 授权（方法级别的权限控制）

上面学习的认证与授权都是基于 URL 的，我们也可以通过注解灵活的配置方法安全，我们先通过@EnableGlobalMethodSecurity 开启基于注解的安全配置。

9.1 新建模块 08-springsecurity-method

9.2 添加依赖

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

9.3 新建启动类并复制 CurrentLoginUserInfoController 类

新建启动类 Application，学员自行创建

9.4 新建 service 及其实现

com.powernode.service 新建教师接口

```
public interface TeacherService {
    String add();
    String update();
    String delete();
    String query();
}
```

com.powernode.service.impl 实现接口

```
@Service
```

```
@Slf4j

public class TeacherServiceImpl implements TeacherService {

    @Override

    public String add() {

        Log.info("添加教师成功");

        return "添加教师成功";

    }


    @Override

    public String update() {

        Log.info("修改教师成功");

        return "修改教师成功";

    }


    @Override

    public String delete() {

        Log.info("删除教师成功");

        return "删除教师成功";

    }


    @Override

    public String query() {

        Log.info("查询教师成功");

        return "查询教师成功";

    }

}
```


9.5 修建 TeacherController

```
@RestController
@RequestMapping("/teacher")
public class TeacherController {

    @Resource
    private TeacherService teacherService;

    @GetMapping("/query")
    public String queryInfo() {
        return teacherService.query();
    }

    @GetMapping("/add")
    public String addInfo() {
        return teacherService.add();
    }

    @GetMapping("/update")
    public String updateInfo() {
        return teacherService.update();
    }

    @GetMapping("/delete")
    public String deleteInfo() {
```

```
        return teacherService.delete();  
    }  
}
```

9.6 新建安全配置类

com.powernode.config 下新建用户配置类

```
@Configuration  
public class MySecurityUserConfig {  
    @Bean  
    public UserDetailsService userDetailService() {  
        // 使用 org.springframework.security.core.userdetails.User 类来  
        // 定义用户  
        // 定义两个用户  
        UserDetails user1 = User.builder()  
            .username("eric")  
            .password(passwordEncoder().encode("123456"))  
            .roles("student")  
            .build();  
  
        UserDetails user2 = User.builder()  
            .username("thomas")  
            .password(passwordEncoder().encode("123456"))  
            .roles("teacher")  
            .build();  
  
        UserDetails user3 = User.builder()  
            .username("admin")  
            .password(passwordEncoder().encode("123456"))  
            .authorities("teacher:add", "teacher:update")  
            .roles("teacher")  
            .build();  
  
        // 创建两个用户  
        InMemoryUserDetailsManager userDetailsManager = new  
        InMemoryUserDetailsManager();  
        userDetailsManager.createUser(user1);  
        userDetailsManager.createUser(user2);  
        return userDetailsManager;  
    }  
}
```

```
    }

    /**
     * 从 Spring5 开始，强制要求密码要加密
     * @return
     */
    @Bean
    public PasswordEncoder passwordEncoder(){
        //使用加密算法对密码进行加密
        return new BCryptPasswordEncoder();
    }
}
```

新建 WebSecurityConfig 类

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //任何访问均需要认证
        http.authorizeRequests().anyRequest().authenticated();
        http.formLogin().permitAll();
    }
}
```

9.7 启动程序并访问

访问以下地址

<http://localhost:8080/teacher/add>

<http://localhost:8080/teacher/update>

<http://localhost:8080/teacher/delete>

<http://localhost:8080/teacher/query>

通过 admin 或 thomas 登录均可以访问所有资源

9.8 修改安全配置类 WebSecurityConfig

加上启用全局方法安全注解

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

修改后，完整代码如下：

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //任何访问均需要认证
        http.authorizeRequests().anyRequest().authenticated();
        //登录放行
        http.formLogin().permitAll();
    }
}
```

9.9 修改 TeacherServiceImpl

在每个方法上加上前置授权注解：@PreAuthorize

完整代码如下：

```
@Service
@Slf4j
public class TeacherServiceImpl implements TeacherService {
```

```
@Override
@PreAuthorize("hasAuthority('teacher:add') OR hasRole('teacher')")
public String add() {
    Log.info("添加教师成功");
    return "添加教师成功";
}

@Override
@PreAuthorize("hasAuthority('teacher:update')")
public String update() {
    Log.info("修改教师成功");
    return "修改教师成功";
}

@Override
@PreAuthorize("hasAuthority('teacher:delete')")
public String delete() {
    Log.info("删除教师成功");
    return "删除教师成功";
}

@Override
@PreAuthorize("hasRole('teacher')")
public String query() {
    Log.info("查询教师成功");
    return "查询教师成功";
}
```

```
}  
  
}
```

9.10 启动并运行

运行程序分别使用 admin 和 teacher 登录，可以查看不同效果

<http://localhost:8080/teacher/add>
<http://localhost:8080/teacher/update>
<http://localhost:8080/teacher/delete>
<http://localhost:8080/teacher/query>

发现 thomas 可以访问添加和查询，别的不能访问，amdin 可以访问添加和更新，别的不能访问。

代码说明：

- EnableGlobalMethodSecurity 注解的属性 prePostEnabled = true 解锁@PreAuthorize 和 @PostAuthorize 注解，@PreAuthorize 在方法执行前进行验证，@PostAuthorize 在方法执行后进行验证
- EnableGlobalMethodSecurity 的 securedEnabled = true 解锁@Secured 注解，@Secured 和 @PreAuthorize 用法基本一样 @Secured 对应的角色必须要有 ROLE_前缀

10 SpringSecurity 返回 json

前后端分离成为企业应用开发中的主流，前后端分离通过 json 进行交互，登录成功和失败后不用页面跳转，而是一段 json 提示

10.1 新建模块 09-springsecurity-json

10.2 添加依赖

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

10.3 新建三个 controller 和获取登录用户信息的 controller

参照 1.2.4 创建，可以直接拷贝过来

10.4 新建启动类

com.powernode 下新建 Application 类，学员自行创建

10.5 创建统一响应类 JsonResult

在 com.powernode.vo 中创建该类

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class JsonResult {
    private Integer code;
    private String msg;
    private Object data;
    public JsonResult(Integer code, String msg) {
```

```
        this.code = code;

        this.msg = msg;
    }
}
```

10.6 创建登录成功处理器

com.powernode.config 包下创建

```
@Component
public class MyAutheticationSuccessHandle implements
AuthenticationSuccessHandler {
@Resource
    private ObjectMapper objectMapper;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletRequest response, Authentication authentication) throws
IOException, ServletException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=utf-8");
        HttpResult httpResult = new HttpResult(200, "登录成功",
authentication);
        String str = objectMapper.writeValueAsString(httpResult);
        response.getWriter().write(str);
        response.getWriter().flush();
    }
}
```

10.7 创建登录失败处理器

```
/**
 * 登陆失败的处理器
 */
@Component
public class AppAuthenticationFailureHandler implements
AuthenticationFailureHandler {
```



```
@Resource
private ObjectMapper objectMapper;

/**
 * @param request 当前的请求对象
 * @param response 当前的响应对象
 * @param exception 失败的原因的异常
 * @throws IOException
 * @throws ServletException
 */
@Override
public void onAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response, AuthenticationException exception)
throws IOException, ServletException {
    System.err.println("登陆失败");
    //设置响应编码
    response.setCharacterEncoding("UTF-8");
    response.setContentType("application/json;charset=utf-8");
    //返回 JSON 出去
    HttpResult result=new HttpResult(-1,"登陆失败");
    if(exception instanceof BadCredentialsException){
        result.setData("密码不正确");
    }else if(exception instanceof DisabledException){
        result.setData("账号被禁用");
    }else if(exception instanceof UsernameNotFoundException){
        result.setData("用户名不存在");
    }else if(exception instanceof CredentialsExpiredException){
        result.setData("密码已过期");
    }else if(exception instanceof AccountExpiredException){
        result.setData("账号已过期");
    }else if(exception instanceof LockedException){
        result.setData("账号被锁定");
    }else{
        result.setData("未知异常");
    }
    //把 result 转成 JSON
    String json = objectMapper.writeValueAsString(result);
}
```

```
        //响应出去
        PrintWriter out = response.getWriter();
        out.write(json);
        out.flush();
    }
}
```

10.8 创建无权限处理器

```
/**
 * 无权限的处理器
 */
@Component
public class AppAccessDeniedHandler implements AccessDeniedHandler {

    //声明一个把对象转成 JSON 的对象
    @Resource
    private ObjectMapper objectMapper;

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse
response, AccessDeniedException accessDeniedException) throws
IOException, ServletException {
        //设置响应编码
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=utf-8");
        //返回 JSON 出去
        HttpResult result=new HttpResult(-1,"您没有权限访问");
        //把 result 转成 JSON
        String json = objectMapper.writeValueAsString(result);
        //响应出去
        PrintWriter out = response.getWriter();
        out.write(json);
        out.flush();
    }
}
```

10.9 创建登出（退出）处理器

```
/**
 * 退出成功的处理器
 */
@Component
public class AppLogoutSuccessHandler implements LogoutSuccessHandler {

    //声明一个把对象转成 JSON 的对象
    @Resource
    private ObjectMapper objectMapper;

    /**
     *
     * @param request
     * @param response
     * @param authentication 当前退出的用户对象
     * @throws IOException
     * @throws ServletException
     */
    @Override
    public void onLogoutSuccess(HttpServletRequest request,
        HttpServletResponse response, Authentication authentication) throws
        IOException, ServletException {
        System.out.println("退出成功");
        //设置响应编码
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=utf-8");
        //返回 JSON 出去
        HttpStatus result=new HttpStatus(200,"退出成功");
        //把 result 转成 JSON
        String json = objectMapper.writeValueAsString(result);
        //响应出去
        PrintWriter out = response.getWriter();
        out.write(json);
        out.flush();
    }
}
```

10.10 创建用户配置类

```
@Configuration
public class MySecurityUserConfig {
    @Bean
    public UserDetailsService userDetailsService() {
        // 使用 org.springframework.security.core.userdetails.User 类来
        // 定义用户
        // 定义用户
        UserDetails user1 = User.builder()
            .username("eric")
            .password(passwordEncoder().encode("123456"))
            .roles("student")
            .build();
        UserDetails user2 = User.builder()
            .username("thomas")
            .password(passwordEncoder().encode("123456"))
            .roles("teacher")
            .build();
        UserDetails user3 = User.builder()
            .username("admin")
            .password(passwordEncoder().encode("123456"))
            .roles("admin")
            .build();
        // 创建用户
        InMemoryUserDetailsManager userDetailsManager = new
        InMemoryUserDetailsManager();
        userDetailsManager.createUser(user1);
        userDetailsManager.createUser(user2);
        userDetailsManager.createUser(user3);
        return userDetailsManager;
    }

    /*
     * 从 Spring5 开始，强制要求密码要加密
     * @return
     */
    @Bean
```

```
public PasswordEncoder passwordEncoder(){
    //使用加密算法对密码进行加密
    return new BCryptPasswordEncoder();
}
}
```

10.11 安全配置类 WebSecurityConfig

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    // 注入登陆成功的处理器
    @Autowired
    private AuthenticationSuccessHandler successHandler;

    // 注入登陆失败的处理器
    @Autowired
    private AppAuthenticationFailureHandler failureHandler;

    // 注入没有权限的处理器
    @Autowired
    private AppAccessDeniedHandler accessDeniedHandler;

    // 注入退出成功的处理器
    @Autowired
    private AppLogoutSuccessHandler logoutSuccessHandler;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.exceptionHandling().accessDeniedHandler(accessDeniedHandler);

        http.formLogin().successHandler(successHandler).failureHandler(failureHandler).permitAll();
        http.logout().logoutSuccessHandler(logoutSuccessHandler);

        http.authorizeRequests().mvcMatchers("/teacher/**").hasRole("teacher");
    }
}
```

```
").anyRequest().authenticated();  
    }  
}
```

10.12 启动程序

10.13 访问测试

可以使用 admin 用户实验登录失败、登录成功、退出和访问 <http://localhost:8080/teacher/query> 查看无权访问的效果

11 使用 UserDetailsService 获取用户认证信息

11.1 新建子模块 10-springsecurity-userdetailservice

11.2 添加依赖

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

11.3 新建启动类

com.powernode 包下新建启动类 Application，学员自行创建

11.4 新建三个 controller

参照 1.2.4 创建，可以直接拷贝过来

11.5 新建获取登录用户认证信息的 controller

拷贝 7.1 即可

11.6 新建用户信息类

com.powernode.vo 包下新建 SecurityUser 类，该类实现接口 UserDetails 接口

```
public class SecurityUser implements UserDetails {
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public String getPassword() {
        //明文为 123456
        return
"$2a$10$KyXAnVcsrLaHMWpd3e2xhe6JmzBi.3AgMhteFq8t8kjsxmWL8o1EDq";
    }

    @Override
    public String getUsername() {
        return "thomas";
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
}
```

```

    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}

```

代码说明：

用户实体类需要实现 UserDetails 接口，并实现该接口中的 7 个方法， UserDetails 接口的 7 个方法如下图：

方法名	解释
getAuthorities();	获取当前用户对象所具有的角色信息
getPassword();	获取当前用户对象的密码
getUsername();	获取当前用户对象的用户名
isAccountNonExpired();	当前账户是否未过期
isAccountNonLocked();	当前账户是否未锁定
isCredentialsNonExpired();	当前账户密码是否未过期
isEnabled();	当前账户是否可用

11.7 新建类实现 UserDetailsService 接口

com.powernode.service.impl 包下新建 UserServiceImpl 实现 UserDetailsService

```

@Service
public class UserServiceImpl implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        SecurityUser securityUser= new SecurityUser();
        if(username==null
|| !username.equals(securityUser.getUsername())){
            throw new UsernameNotFoundException("该用户不存在");
        }
        return securityUser;
    }
}

```


11.8 新建安全配置类

com.powernode.config 下新建 WebSecurityConfig 类，配置密码编码器

```
@Configuration
@Slf4j

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean

    public PasswordEncoder passwordEncoder(){

        return new BCryptPasswordEncoder();

    }

}
```

启动程序，并使用浏览器访问程序：<http://localhost:8080/student/query>

发现需要登录，使用 thomas/123456 登录后，即可正常访问。

访问：<http://localhost:8080/getLoginUserInfo>

发现该用户并没有权限信息

```
{
  "authorities": [],
  "details": {
    "remoteAddress": "0:0:0:0:0:0:1",
    "sessionId": "A17C7B2F8A0EBACACB7367421E17F1B2"
  },
  "authenticated": true,
  "principal": {
    "username": "thomas",
    "password": "$2a$10$KyXAnVcsrLaHMWpd3e2xhe6JmzBi.3AgMhteFq8t8kjsxmwL8o1EDq",
    "enabled": true,
    "accountNonLocked": true,
    "credentialsNonExpired": true,
    "accountNonExpired": true,
    "authorities": null
  },
  "credentials": null,
  "name": "thomas"
}
```

11.9 配置用户权限信息

修改 SecurityUser 类中的 getAuthorities 方法

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    GrantedAuthority g1=()->"student:query"; //使用 lambda 表达式
//    GrantedAuthority g1=new
SimpleGrantedAuthority("student:query");
    List<GrantedAuthority> grantedAuthorityList=new ArrayList<>();
    grantedAuthorityList.add(g1);
    return grantedAuthorityList;
}
```

11.10 修改要访问 controller 中的方法需要哪些权限

修改 WebSecurityConfig，添加全局方法拦截注解

@EnableGlobalMethodSecurity(prePostEnabled = true)

注意可以去掉：@Configuration 注解了

修改 StudentController

添加 @PreAuthorize("hasAuthority('student:query')") 注解修改后如下：

```
@RestController
@RequestMapping("/student")
public class StudentController {

    @GetMapping("/query")
    @PreAuthorize("hasAuthority('student:query')")
    public String queryInfo(HttpServletRequest request){
        return "I am a student,My name is XXX";
    }
}
```

修改 TeacherController

添加 @PreAuthorize("hasAuthority('teacher:query')") 注解修改后如下:

```
@RestController
@RequestMapping("/teacher")
public class TeacherController {

    @GetMapping("/query")
    @PreAuthorize("hasAuthority('teacher:query')")
    public String queryInfo(){
        return "I am a teacher,My name is Thomas";
    }
}
```

启动测试, 使用 thomas/123456 登录系统, 发现可以访问 student/query, 不可以访问 teacher/query

再次访问: <http://localhost:8080/getLoginUserInfo> 查看用户信息

```
{
  "authorities": [{
    "authority": "student:query"
  }],
  "details": {
    "remoteAddress": "0:0:0:0:0:0:0:1",
    "sessionId": "AF796B8BA6FD454B71EED72D92997F72"
  },
  "authenticated": true,
  "principal": {
    "username": "thomas",
    "password": "$2a$10$KyXAnVcsrLaHMWpd3e2xhe6JmzBi.3AgMhteFq8t8kjmL8o1EDq",
    "enabled": true,
    "authorities": [{
      "authority": "student:query"
    }],
    "accountNonExpired": true,
    "accountNonLocked": true,
    "credentialsNonExpired": true
  },
  "credentials": null,
  "name": "thomas"
}
```

11.11 为什么讲这个示例？

是为了使用数据库存储用户角色权限信息做准备，只要从数据库中取出数据存储到实现 UserDetails 的接口的类中即可，比如 SecurityUser 中即可。

12 基于数据库的认证

12.1 创建数据库 security_study 和表

创建数据库 security_study

导入数据库脚本 security_study.sql

12.2 创建模块 11-springsecurity-database-authentication

12.3 添加依赖

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.6.13</version>
</parent>

<properties>
  <maven.compiler.source>8</maven.compiler.source>
  <maven.compiler.target>8</maven.compiler.target>
  <java.version>8</java.version>
</properties>
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.2.2</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
```

```
<artifactId>lombok</artifactId>
<optional>true</optional>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

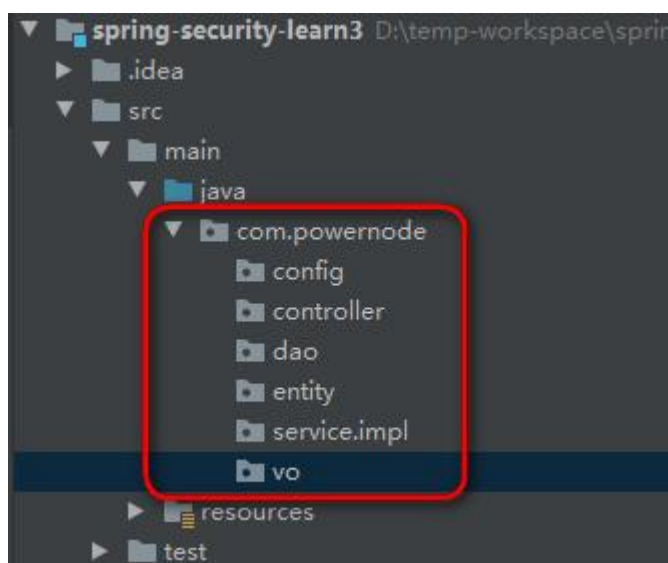
12.4 配置数据源和 mybatis

新建配置文件 application.yml 并配置数据源和 mybatis

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://127.0.0.1:3306/security_study?useUnicode=true&character
Encoding=UTF-8&serverTimezone=Asia/Shanghai
    username: root
    password: root
mybatis:
  type-aliases-package: com.powernode.entity
  configuration:
```

```
map-underscore-to-camel-case: true  
  
log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  
  
mapper-locations: classpath:mapper/*.xml
```

12.5 新建各个包



12.6 新建用户实体类

com.pownode.entity 包下新建用户实体类

```
@Data  
public class SysUser implements Serializable {  
    private Integer userId;  
    private String username;  
    private String password;  
    private String sex;  
    private String address;  
    private Integer enabled;  
    private Integer accountNoExpired;  
    private Integer credentialsNoExpired;  
    private Integer accountNoLocked;
```

```
}
```

12.7 新建用户 mapper 和映射文件

com.powernode.dao 下新建

```
public interface SysUserDao {  
    /**  
     * 根据用户名获取用户信息  
     * @param username  
     * @return  
     */  
    SysUser getByUserName(@Param("username") String username);  
}
```

mapper 下新建映射文件 SysUserMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper  
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.powernode.dao.SysUserDao">  
    <select id="getByUserName" resultType="sysUser">  
        select  
user_id,username,password,sex,address,enabled,account_no_expired,cre  
dentials_no_expired,account_no_locked  
        from sys_user where username=#{username}  
    </select>  
</mapper>
```


注意 select 后面不要使用*。

12.8 新建启动类

com.powernode 包下新建启动类

```
@SpringBootApplication
@MapperScan("com.powernode.dao")
public class Application {

    public static void main(String[] args) {

        SpringApplication.run(Application.class,args);

    }

}
```

12.9 单元测试

测试 dao

```
@SpringBootTest
class SysUserDaoTest {

    @Resource
    private SysUserDao sysUserDao;

    @Test
    void getByUserName() {

        SysUser sysUser = sysUserDao.getByUserName("obama");

        assertNotNull(sysUser);

    }

}
```

注意单元测试要测试哪些：dao--service-controller，实体类一般不需要测试

12.10 新建安全用户类

com.powernode.vo 包下新建类

```
public class SecurityUser implements UserDetails {
    private final SysUser sysUser;

    public SecurityUser(SysUser sysUser) {
        this.sysUser=sysUser;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public String getPassword() {
        String userPassword=this.sysUser.getPassword();
        //注意清除密码
        this.sysUser.setPassword(null);
        return userPassword;
    }

    @Override
    public String getUsername() {
        return sysUser.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return sysUser.getAccountNoExpired().equals(1);
    }

    @Override
    public boolean isAccountNonLocked() {
        return sysUser.getAccountNoLocked().equals(1);
    }
}
```

```
@Override
public boolean isCredentialsNonExpired() {
    return sysUser.getCredentialsNoExpired().equals(1);
}

@Override
public boolean isEnabled() {
    return sysUser.getEnabled().equals(1);
}
}
```

12.11 新建 UserServiceImpl 实现 UserDetailsService 接口

```
@Service
public class UserServiceImpl implements UserDetailsService {
    @Resource
    private SysUserDao sysUserDao;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        SysUser sysUser = sysUserDao.getByUserName(username);
        if(null==sysUser){
            throw new UsernameNotFoundException("账号不存在");
        }
        return new SecurityUser(sysUser);
    }
}
```

12.12 新建 service 单元测试

```
@SpringBootTest
class UserServiceImplTest {

    @Resource
    private UserServiceImpl userService;
```

```
@Test

void loadUserByUsername() {
    UserDetails userDetails =
userService.loadUserByUsername("obama");
    assertNotNull(userDetails);
}
}
```

12.13 新建两个控制器

```
@RestController
@Slf4j
@RequestMapping("/student")
public class StudentController {
    @GetMapping("/query")
    public String queryInfo(){
        return "query student";
    }
    @GetMapping("/add")
    public String addInfo(){
        return "add student!";
    }
    @GetMapping("/update")
    public String updateInfo(){
        return "update student";
    }
    @GetMapping("/delete")
```

```
public String deleteInfo(){
    return "delete student!";
}

@GetMapping("/export")
public String exportInfo(){
    return "export student!";
}
}
```

```
@RestController
@Slf4j
@RequestMapping("/teacher")
public class TeacherController {
    @GetMapping("/query")
    @PreAuthorize("hasAuthority('teacher:query')")
    public String queryInfo(){
        return "I am a teacher!";
    }
}
```

12.14 新建获取登录用户认证信息的 controller

从 7.1 中拷贝即可

12.15 新建 web 安全配置类

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
@Slf4j
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
```

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().anyRequest().authenticated();
    http.formLogin();
}
}
```

启动并进行各种测试

使用 thomas 和 obama 分别登录测试，发现 student/query 等能访问，teacher/query 不能访问，原因

http://localhost:8080/getLoginUserInfo

```
{
  "authorities": [],
  "details": {
    "remoteAddress": "0:0:0:0:0:0:0:1",
    "sessionId": "C963FFD0520C99114C649CE8AC5B11AC",
    "authenticated": true,
    "principal": {
      "userId": 2,
      "username": "thomas",
      "password": "$2a$10$KyXAnVcsrlaH0Wpd3e2che6JnzBi.3AgMuteFq8t8kxwL8olEDq",
      "sex": "男",
      "address": "北京",
      "enabled": 1,
      "accountNoExpired": 1,
      "credentialsNoExpired": 1,
      "accountNoLocked": 1,
      "accountNonLocked": true,
      "credentialsNonExpired": true,
      "accountNonExpired": true,
      "authorities": null,
      "credentials": null,
      "name": "thomas"
    }
  }
}
```

发现用户没有权限，但是/teacher/query 需要访问权限

13 基于数据库的方法授权

13.1 新建模块

复制 11-springsecurity-database-authentication 改名为

12-springsecurity-database-authorization-method

注意这个工程已经有认证功能了。下面咱们看下如何设置用户的权限

13.2 新建菜单（权限）实体类

```
@Data
public class SysMenu implements Serializable {
```

```
private Integer id;  
private Integer pid;  
private Integer type;  
private String name;  
private String code;  
}
```

13.3 新建权限 mapper 和映射文件

```
public interface SysMenuDao {  
    List<String> queryPermissionByUserId(@Param("userId") Integer  
userId);  
}
```

映射文件 SysMenuMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper  
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.powernode.dao.SysMenuDao">  
    <select id="queryPermissionByUserId" resultType="string">  
        SELECT distinct sm.`code` FROM `sys_role_user` sru inner join  
sys_role_menu srm  
on sru.rid=srm.rid inner join sys_menu sm on srm.mid=sm.id where
```

```
sru.uid=#{userId} and sm.delete_flag=0
    </select>
</mapper>
```

13.4 权限 dao 的单元测试

```
@SpringBootTest
class SysMenuDaoTest {

    @Resource
    private SysMenuDao sysMenuDao;

    @Test
    void queryPermissionByUserId() {
        List<String> menuList = sysMenuDao.queryPermissionByUserId(1);
        assertTrue(!menuList.isEmpty());
    }
}
```

13.5 修改 SecurityUser 实体类

加入一个属性

```
private List<SimpleGrantedAuthority> simpleGrantedAuthorities;
```

修改方法 getAuthorities

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return simpleGrantedAuthorities;
}
```

添加一个 set 方法


```
public void setSimpleGrantedAuthorities(List<SimpleGrantedAuthority>
simpleGrantedAuthorities) {
    this.simpleGrantedAuthorities = simpleGrantedAuthorities;
}
```

13.6 修改 UserServiceImpl

增加设置权限的步骤，修改后如下：

```
@Service
@Slf4j
public class UserServiceImpl implements UserDetailsService {
    @Resource
    private SysUserDao sysUserDao;
    @Resource
    private SysMenuDao sysMenuDao;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        SysUser sysUser = sysUserDao.getByUserName(username);
        if(null==sysUser){
            throw new UsernameNotFoundException("账号不存在");
        }
        List<String>
strList=sysMenuDao.queryPermissionByUserId(sysUser.getUserId());
        //使用 stream 流来转换
        List<SimpleGrantedAuthority>
grantedAuthorities=strList.stream().map(SimpleGrantedAuthority::new)
.collect(toList());
        SecurityUser securityUser = new SecurityUser(sysUser);
        securityUser.setSimpleGrantedAuthorities(grantedAuthorities);
        return securityUser;
    }
}
```

启动并进行各种测试

使用 thomas 和 obama 分别登录测试，发现已经有权限功能了