

5-1

第 5-1 课：使用 Spring Boot Security 进行安全控制

《精通 Spring Boot 42 讲》共分五大部分，这是最后一部分的内容了，主要包含安全、测试、部署、监控及综合实践。对于安全访问控制主要讲解 Spring Boot Security 的使用；监控主要使用 Spring Boot Actuator 和 Spring Boot Admin，在实际的生产应用中这两个组件非常实用；Spring Boot 对测试的支持是全面的，这一部分将会对其进行整理汇总；Spring Boot 部署很简单，如果结合了 Docker 的使用，更方便部署、运维、水平扩展；最后，将用一个真实的实战案例来回顾 Spring Boot 课程内容。

安全是一个企业的底裤，为企业阻挡了外部非正常的访问，保证了企业内部数据安全；业内已经有多起因数据泄露给公司造成重大损失的事件，到现在安全问题越发受到行业内公司的重视。数据泄露很大一部分原因是非正常权限访问导致，是合适的安全框架保护企业服务安全变的非常紧迫，在 Java 领域 Spring Security 无疑是最佳选择之一。

Spring Security 介绍

Spring Security 是一个能够基于 Spring 的企业应用系统提供声明式的安全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。

Spring Security 的前身是 Acegi Security，它是一个基于 Spring AOP 和 Servlet 过滤器的安全框架。它提供全面的安全性解决方案，同时在 Web 请求级和方法调用级处理身份确认和授权，为基于 J2EE 企业应用软件提供了全面安全服务。

Spring Boot 提供了集成 Spring Security 的组件包 spring-boot-starter-security，方便我们在 Spring Boot 项目中使用 Spring Security。

快速上手

先来做一个 Web 系统。

（1）添加依赖：

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

（2）配置文件

配置文件中将 Thymeleaf 的缓存先去掉。

```
spring.thymeleaf.cache=false
```

（3）创建页面

在 resources/templates 目录下创建页面 index.html，在页面简单写两句话。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>index</title>
</head>
<body>
<h1>Hello! </h1>
<p>今天天气很好，来一个纯洁的微笑吧！ </p>
</body>
</html>
```

(4) 添加访问入口

创建 SecurityController 类，在类中添加访问页面的入口：

```
@Controller
public class SecurityController {
    @RequestMapping("/")
    public String index() {
        return "index";
    }
}
```

添加完成后启动项目，在浏览器中访问地址：http://localhost:8080/ (http://localhost:8080/)，页面展示结果如下：

Hello!

今天天气很好，来一个纯洁的微笑吧！

以上完成了一个特别简单的 Web 页面请求、展示信息。

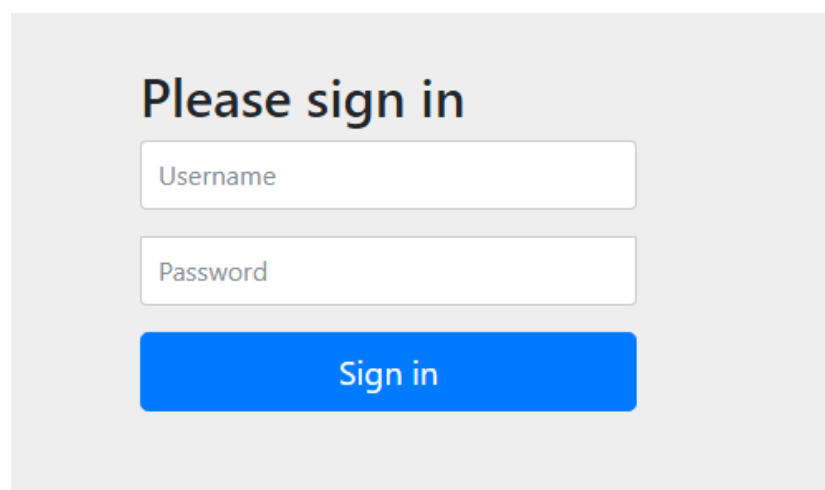
(5) 添加 Spring Security 依赖

现在在项目中添加 spring-boot-starter-security 的依赖包。

在 pom.xml 添加：

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

添加完成后重启项目，再次访问地址：<http://localhost:8080/> (<http://localhost:8080/>)，页面会自动弹出了一个登录框，如下：



说明 Spring Security 自动给所有访问请求做了登录保护，那么这个登录名和密码是什么呢，如果观察比较仔细的话，会发现添加了 spring-boot-starter-security 依赖包重启后的项目，在控制台打印了一长串字符，如下：

```
2018-11-09 12:27:46.052 INFO 26240 --- [ restartedMain] .s.s.User
DetailsServiceAutoConfiguration :
```

```
Using generated security password: d2c87183-ada6-4f26-b803-db2e60b0
1079
```

根据打印信息可以看出，这应该就是登录的密码了。

(6) 进行分析

根据上面的打印信息，可以看出密码是由 UserDetailsServiceAutoConfiguration 类打印出的，在 IDEA 连续按两次 Shift 键，调出 IDEA 的类搜索框，输出类名 UserDetailsServiceAutoConfiguration，查看它的源码，具体打印代码如下：

```

private String getOrDeducePassword(User user, PasswordEncoder encoder) {
    String password = user.getPassword();
    if (user.isPasswordGenerated()) {
        logger.info(String.format("%n%nUsing generated security password: %s%n", user.getPassword()));
    }

    return encoder == null && !PASSWORD_ALGORITHM_PATTERN.matcher(password).matches() ? "{noop}" + password : password;
}

```

可以看出 User 就是我们需要的登录用户信息，打开 User 其源码如下：

```

public static class User {
    private String name = "user";
    private String password = UUID.randomUUID().toString();
    private List<String> roles = new ArrayList<>();
    private boolean passwordGenerated = true;
    //省略一部分
    public void setPassword(String password) {
        if (!StringUtils.hasLength(password)) {
            return;
        }
        this.passwordGenerated = false;
        this.password = password;
    }
    //省略一部分
}

```

根据 User 类的信息发现，passwordGenerated 默认值为 true，当用户被设置密码时更新为 false；也就是说如果没有设置密码 passwordGenerated 的值为 true。password 的值默认由 UUID 生产的一段随机字符串，用户名默认为 user。综上，用户名 user 和控制台打印的密码便是系统默认的登录和密码，登录成功后跳转到首页。

当然，如果想修改用户名和密码，可以在 application.properties 重新进行配置，例如：

```

# security
spring.security.user.name=admin
spring.security.user.password=admin

```

配置完成之后重启项目，再次访问 <http://localhost:8080/> (<http://localhost:8080/>)，在跳转出来的登录页面输入上述用户名和密码，可以登录成功。

登录认证

上述是 Spring Security 最简单的集成演示，在实际项目使用过程中，有的页面不需要进行验证，有的页面需要进行验证，账户密码需要存储到数据库、角色权限相关联等，其实这些 Spring Security 轻松可实现。

创建页面 content.html，此页面只有登录用户才可查看，否则会跳转到登录页面，登录成功后才能访问。可以自定义登录页面，当用户未登录时跳转到自定义登录页面。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<body>
<h1>content</h1>
<p>我是登录后才可以看的页面</p>
</body>
</html>
```

登录页面：

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
    <title>login</title>
</head>
<body>
<div th:if="${param.error}">
    用户名或密码错
</div>
<div th:if="${param.logout}">
    您已注销成功
</div>
<form th:action="@{/login}" method="post">
    <div><label> 用户名 : <input type="text" name="username"/> </label></div>
    <div><label> 密 码 : <input type="password" name="password"/> </label></div>
    <div><input type="submit" value="登录"/></div>
</form>
</body>
</html>

```

后台添加访问入口:

```

@RequestMapping("/content")
public String content() {
    return "content";
}

@RequestMapping(value = "/login", method = RequestMethod.GET)
public String login() {
    return "login";
}

```

进行配置 index.html 可以直接访问, 但 content.html 需要登录后才可查看, 没有登录自动调整到 login.html , 创建 SecurityConfig 类继承于 WebSecurityConfigurerAdapter。

```

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/", "/home").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
            // .loginPage("/login")
            .permitAll()
            .and()
            .logout()
            .permitAll()
            .and()
            .csrf()
            .ignoringAntMatchers("/logout");
    }
}

```

- @EnableWebSecurity, 开启 Spring Security 权限控制和认证功能。
- antMatchers("/", "/home").permitAll(), 配置不用登录可以访问的请求。
- anyRequest().authenticated(), 表示其他的请求都必须要有权限认证。
- formLogin(), 定制登录信息。
- loginPage("/login"), 自定义登录地址, 若注释掉则使用默认登录页面。
- logout(), 退出功能, Spring Security 自动监控了 /logout。
- ignoringAntMatchers("/logout"), Spring Security 默认启用了同源请求控制, 在这里选择忽略退出请求的同源限制。

我们在 index 页面添加一个挑战 content 页面的链接, 同时在 content 页面添加一个退出的链接。

index 页面:

```
<p>点击 <a th:href="@{/content}">这里</a> 进入受限页面</p>
```

content 页面:


```
<form method="post" action="/logout">
    <button type="submit">退出</button>
</form>
```

退出请求默认只支持 post 请求，修改完成之后重启项目，访问地址 `http://localhost:8080/` (`http://localhost:8080/`) 可以看到 index 页面内容，点击链接跳转到 content 页面时，会自动跳转到 `http://localhost:8080/login` (`http://localhost:8080/login`) 登录页面，登录成功后会自动跳转到 `http://localhost:8080/content` (`http://localhost:8080/content`)，在 content 页面单击“退出”按钮，会退出登录状态，跳转到登录页面并提示已经退出。

登录、退出、请求受限页面，退出后跳转到登录页面，是最常见的安全控制案例，是账户系统最基本的安全保障，接下来介绍如何通过角色来控制权限。

角色权限

也可以在 Java 代码中配置用户登录名和密码，在上面创建的 `SecurityConfig` 类中添加方法 `configureGlobal()`。

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
        .passwordEncoder(new BCryptPasswordEncoder())
        .withUser("user")
        .password(new BCryptPasswordEncoder()
            .encode("123456")).roles("USER");
}
```

在 Spring Boot 2.x 中配置密码需要指明密码的加密方式。当在配置文件和 `SecurityConfig` 类中都配置了用户名和密码时，会使用代码中的用户名和密码。添加完上述代码，重启项目后，即可用最新的用户名和密码登录系统。

在上述代码中有这么一段 `roles("USER")` 指明了用户角色，角色就是 Spring Security 最重要的概念之一，往往通过用户来控制权限比较繁琐，在实际项目中，往往都是将用户关联到角色，给角色赋予一定的权限，通过角色来控制用户访问请求。

为了演示不同角色拥有不同权限，再添加一个管理员 admin 和 角色 ADMIN。

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
        .passwordEncoder(new BCryptPasswordEncoder())
        .withUser("user")
            .password(new BCryptPasswordEncoder()
                .encode("123456")).roles("USER")
        .and()
        .withUser("admin")
            .password(new BCryptPasswordEncoder()
                .encode("admin")).roles("ADMIN", "USER");
}
```

admin 用户拥有 USER 和 ADMIN 的角色，user 用户拥有 USER 角色，添加 admin.html 页面设置只有 ADMIN 角色的用户才可以访问。

admin.html:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
    <title>admin</title>
</head>
<body>
    <h1>admin</h1>
    <p>管理员页面</p>
    <p>点击 <a th:href="@{/}">这里</a> 返回首页</p>
</body>
</html>
```

添加后端访问:

```
@RequestMapping("/admin")
public String admin() {
    return "admin";
}
```

我们再将上述的 `configure()` 方法修改如下：

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/resources/**", "/").permitAll()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .antMatchers("/content/**").access("hasRole('ADMIN') or hasRole('USER')")
        .anyRequest().authenticated()
        .and()
        .formLogin()
        //                .loginPage("/login")
        .permitAll()
        .and()
        .logout()
        .permitAll()
        .and()
        .csrf()
        .ignoringAntMatchers("/logout");
}
```

重点看这些：

- `antMatchers("/resources/**", "/").permitAll()`，地址 `/resources/ **` 和 `/` 所有用户都可访问，`permitAll` 表示该请求任何人都可以访问；
- `antMatchers("/admin/**").hasRole("ADMIN")`，地址 `/admin/**` 开头的请求地址，只有拥有 `ADMIN` 角色的用户才可以访问；
- `antMatchers("/content/**").access("hasRole('ADMIN') or hasRole('USER')")`，地址 `/content/**` 开头的请求地址，可以给角色 `ADMIN` 或者 `USER` 的用户来使用；
- `antMatchers("/admin/**").hasIpAddress("192.168.11.11")`，只有固定 IP 地址的用户可以访问。

更多的权限控制方式参看下表：

方法名	解释
<code>access(String)</code>	Spring EL 表达式结果为 <code>true</code> 时可访问
<code>anonymous()</code>	匿名可访问
<code>denyAll()</code>	用户不可以访问

fullyAuthenticated()	用户完全认证可访问（非 remember me 下自动登录）
hasAnyAuthority(String...)	参数中任意权限的用户可访问
hasAnyRole(String...)	参数中任意角色的用户可访问
hasAuthority(String)	某一权限的用户可访问
hasRole(String)	某一角色的用户可访问
permitAll()	所有用户可访问
rememberMe()	允许通过 remember me 登录的用户访问
authenticated()	用户登录后可访问
hasIpAddress(String)	用户来自参数中的 IP 时可访问

配置完成重新启动项目，使用用户 admin 登录系统，所有页面都可以访问，使用 user 登录系统，只可访问不受限地址和以 "/content/**" 开头的请求，说明权限配置成功。

值得注意的是 hasRole() 和 access() 虽然都可以给角色赋予权限，但有所区别，比如 hasRole() 修饰的角色 "/admin/**"，那么拥有 ADMIN 权限的用户访问地址 xxx/admin 和 xxx/admin/* 均可，如果使用 access() 修饰的角色，那么访问地址 xxx/admin 权限受限，请求 xxx/admin/ 可以通过。

方法级别的安全

上面是通过请求路径来控制权限，也可以在方法上添加注解来限制控制访问权限。

@PreAuthorize / @PostAuthorize

Spring 的 @PreAuthorize/@PostAuthorize 注解更适合方法级的安全，也支持 Spring EL 表达式语言，提供了基于表达式的访问控制。

- @PreAuthorize 注解：适合进入方法前的权限验证，@PreAuthorize 可以将登录用户的角色 / 权限参数传到方法中。
- @PostAuthorize 注解：使用并不多，在方法执行后再进行权限验证。

```
@PreAuthorize("hasAuthority('ADMIN')")
@RequestMapping("/admin")
public String admin() {
    return "admin";
}
```

这样只要拥有角色 ADMIN 的用户才可以访问此方法。

@Secured

此注释是用来定义业务方法的安全配置属性的列表，可以在需要安全 [角色 / 权限等] 的方法上指定 @Secured，并且只有那些角色 / 权限的用户才可以调用该方法。如果有人不具备要求的角色 / 权限但试图调用此方法，将会抛出 AccessDenied 异常。

示例：

```
public interface UserService {

    List<User> findAllUsers();

    @Secured("ADMIN")
    void updateUser(User user);

    @Secured({ "USER", "ADMIN" })
    void deleteUser();
}
```

如此项目中便可根据角色来控制用户拥有不同的权限。为了方便演示，内容中所有用户和角色信息均写死在代码中，在实际项目使用中，会将用户、角色、权限控制等信息存储到数据库中，以更加方便灵活的方式去配置整个项目的权限。

总结

通过本课内容的学习，我们了解到 Spring Security 是一个专注认证和权限控制的一套安全框架。Spring Boot 有对应的组件包帮助集成，在 Spring Boot 项目中，可以通过不同的注解和配置来控制不同用户、不同角色的访问权限。Spring Security 是一款非常强大的安全控制框架，本课内容只是演示了常见的使用场景，若大家感兴趣可以线下继续学习了解。

点击这里下载源码 (https://github.com/ityouknow/spring-boot-learning/tree/gitbook_column2.0)。



(/gitchat/column/5b86228ce15aa17d68b5b55a/topic/5c21de564fcd483b0264f22f) (/gitcha