

# 第7章 Spring Boot安全管理

《Spring Boot企业级开发教程（第2版）》



黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌

# 学习目标/Target



了解安全框架概述，能够简述Spring Security和Shiro的作用

掌握Spring Security入门案例，能够基于Spring Boot项目完成Spring Security入门案例

了解Spring Security结构总览，能够简述Spring Security过滤器处理请求的流程

熟悉Spring Security认证流程，能够简述Spring Security的认证流程

掌握Spring Security自定义身份认证，能够基于内存身份认证、JDBC身份认证和自定义UserDetailsService实现用户身份认证

# 学习目标/Target



熟悉Spring Security授权流程，能够简述Spring Security的授权流程

掌握Spring Security自定义授权，能够使用Web授权和方法授权实现用户授权管理

掌握动态展示菜单，能够通过Spring Security的授权管理实现动态展示菜单

掌握Spring Security会话管理，能够在Spring Security中获取认证后的用户信息，以及进行会话控制

掌握Spring Security用户退出，能够在Spring Boot项目中实现使用Spring Security实现用户退出

## 章节概述/ Summary



实际开发中，开发者为了确保Web应用的[安全性](#)，通常需要保护Web应用的用户信息、数据信息等资源不受侵害，例如，对于某些指定的功能，需要先对访问的用户进行[身份验证](#)，验证通过后还需要[具备](#)相关[权限](#)之后才可以操作。下面将对Spring Boot的[安全管理](#)进行详细地讲解。



7.1

安全框架概述

7.2

Spring Security基础入门

7.3

Spring Security认证管理

7.4

Spring Security授权管理

7.5

Spring Security会话管理和用户退出



## 7.1

# 安全框架概述



了解安全框架概述，能够简述Spring Security和Shiro的作用



Java中的安全框架通常是指解决Web应用安全问题的框架，如果开发Web应用时没有使用安全框架，开发者需要自行编写代码增加Web应用安全性。自行实现Web应用的安全性并不容易，需要考虑不同的认证和授权机制、网络关键数据传输加密等多方面的问题，为此Web应用中通常会选择使用一些成熟的安全框架，这些安全框架基本都提供了一套Web应用安全性的完整解决方案，以便提升Web应用的安全性。

Java中常用的安全框架有Spring Security和Shiro，这两个安全框架都提供了强大功能，可以很容易实现Web应用的很多安全防护。下面对这两个安全框架的特点进行讲解。





### 1.Spring Security

Spring Security是Spring生态系统中重要的一员，是一个基于AOP思想和Servlet过滤器实现的安全框架，它提供了完善的认证机制和方法级的授权功能，是一款非常优秀的权限管理框架。Spring Security伴随着Spring生态系统不断修正、升级，使用Spring Security 减少了为企业系统安全控制编写大量重复代码的工作，在Spring Boot项目中使用Spring Security十分简单。



### 1.Spring Security

使用Spring Security可以很方便地实现Authentication（认证）和 Authorization（授权），其中认证是指验证用户身份的过程，授权是指验证用户是否有权限访问特定资源的过程。Spring Security在架构上将认证与授权分离，并提供了扩展点。

Spring Security具有以下的特点。

- **灵活**：Spring Security 提供了一系列可扩展的模块，可以根据具体需求进行选择和配置。
- **安全**：Spring Security 集成了一系列安全措施，包括 XSS ( Cross-Site Scripting ， 跨站脚本)攻击防范、CSRF 攻击防范、点击劫持攻击防范等。
- **易用**：Spring Security 提供了一系列快捷配置选项，可以使开发人员轻松地实现认证和授权等功能。
- **社区支持**：Spring Security作为Spring 生态系统的一部分，与Spring无缝整合，并且得到了社区广泛的支持和更新维护。



### 2.Shiro

Shiro是apache旗下一个开源框架，它将软件系统的安全认证相关功能抽取出来，实现用户身份认证，授权、加密、会话管理等功能，组成了一个通用的安全认证框架。

Shiro具有如下特点。

- 易于理解的 Java Security API。
- 简单的身份认证，支持LDAP, JDBC 等多种数据源。
- 支持对角色的简单鉴权，也支持细粒度的鉴权。
- 支持一级缓存，以提升应用程序的性能。
- 内置的基于 POJO 企业会话管理，适用于Web以及非Web的环境。
- 不跟任何的框架或者容器捆绑，可以独立运行。



不管Spring Security还是Shiro，在进行安全管理的过程中都涉及权限管理的两个重要概念：**认证和授权**。权限管理是指根据系统设置的安全规则或者安全策略，用户可以访问且只能访问自己被授权的资源。实现权限管理通常需要三个对象，分别为**用户**、**角色**、**权限**，这三个对象的说明如下。

- **用户**：主要包含用户名、密码和当前用户的角色信息，可以实现**认证**操作。
- **角色**：主要包含角色名称、角色描述和当前角色拥有的**权限信息**，可以实现**授权**操作。
- **权限**：权限也可以称为菜单，主要包含当前**权限名称**、url地址等信息，可以实现**动态展示菜单**。



## 7.2

# Spring Security基础入门



掌握Spring Security入门案例，能够  
基于Spring Boot项目完成Spring  
Security入门案例

## >>> 7.2.1 Spring Security入门案例



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

本入门案例主要演示Spring Security在Spring Boot中的安全管理效果。为了更好地使用Spring Boot整合实现Spring Security安全管理功能，体现案例中Authentication（认证）和Authorization（授权）功能的实现，本案例在Spring Boot项目中结合Spring MVC和Thymeleaf实现访问图书管理后台页面。

## >>> 7.2.1 Spring Security入门案例

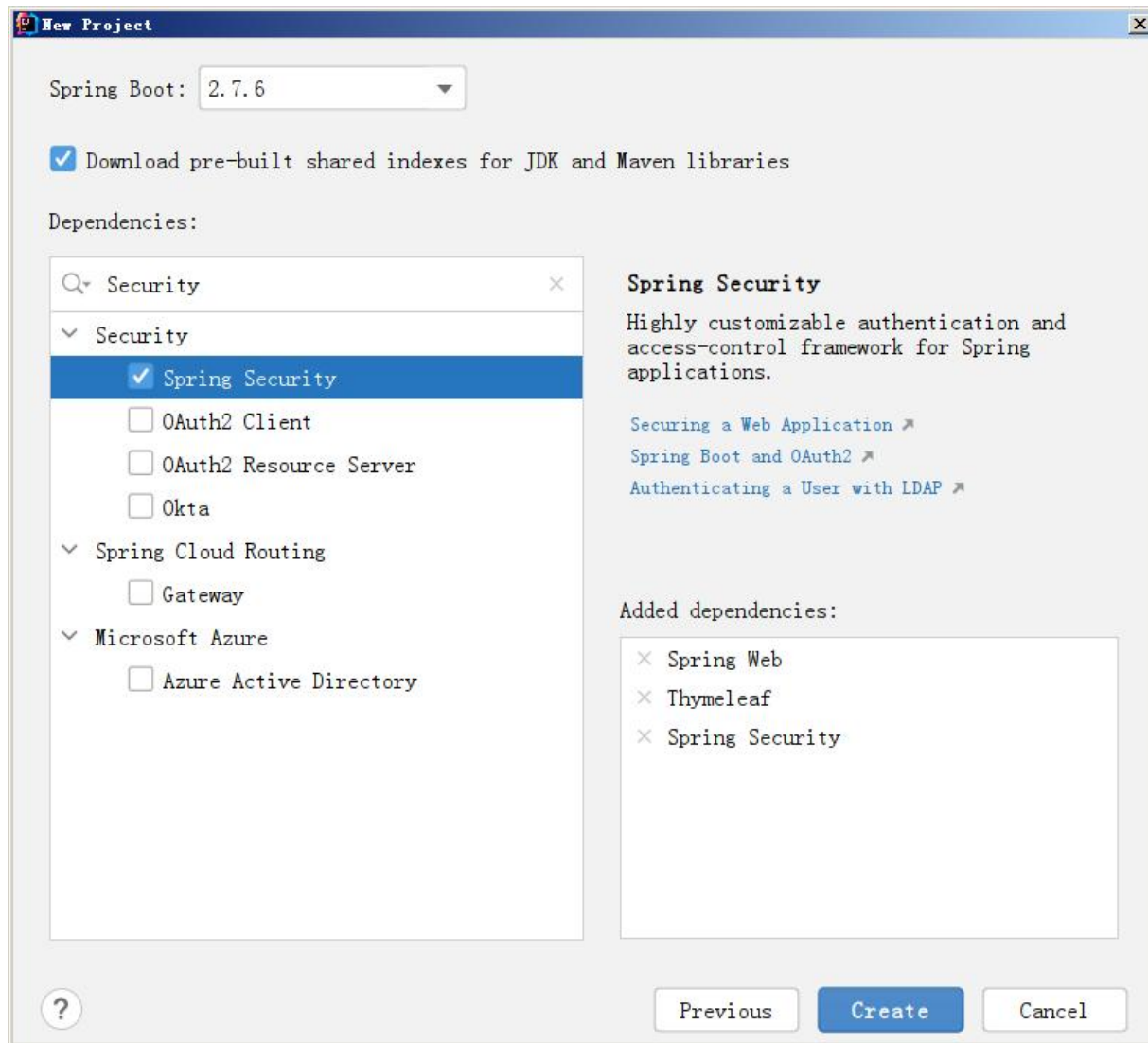


黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 1.创建Spring Boot项目

使用Spring Initializr方式创建一个名为chapter07的Spring Boot项目, 在Dependencies依赖选择中选择Spring Web、Thymeleaf和Spring Security的依赖, 然后根据提示完成项目创建。

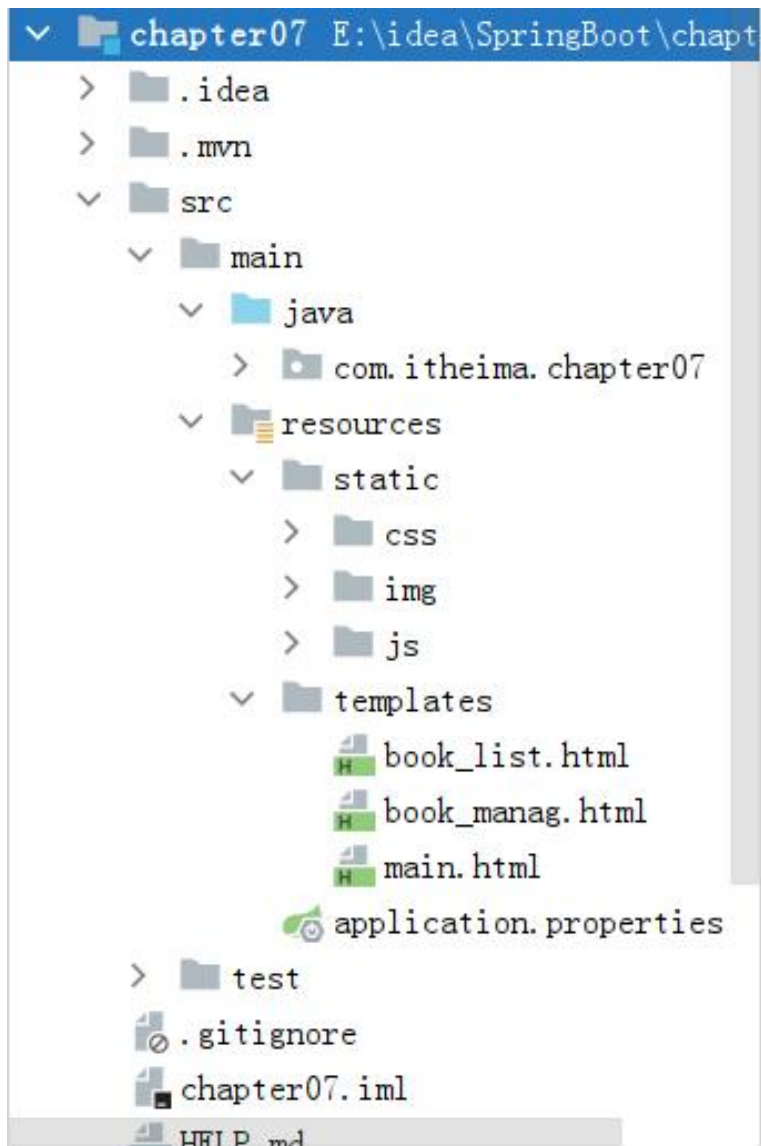






### 2.导入页面资源

在项目的resources目录的templates和static文件夹中，分别引入案例的模板文件，以及页面所需的静态资源文件。





### 2.导入页面资源

导入的页面文件有3个，其中main.html为图书管理的主页面，book\_list.html为图书列表页面，该页面展示所有可阅读的图书信息，book\_manag.html为图书管理页面，该页面可以对图书进行增删改等操作，这3个页面的核心代码如文件7-1~文件7-3所示。



#### 源代码

文件7-1 [main.html](#)

文件7-2 [book\\_list.html](#)

文件7-3 [book\\_manag.html](#)





### 3.创建控制器类

在项目的java目录下创建包com.itheima.chapter07.controller，并在该包下创建实体类控制器类BookController，在该类中定义处理图书列表和图书管理请求的方法，具体如文件7-4所示。



#### 源代码

文件7-4

BookController.java





### 4.添加配置类

导入的页面文件有3个，其中main.html为图书管理的[主页面](#)，book\_list.html为[图书列表页面](#)，该页面在项目的java目录下创建包com.itheima.chapter07.config，并在该包下创建[配置类WebMvcConfig](#)，在该类中[添加视图路径映射](#)，实现访问项目[首页自动映射到后台管理首页](#)，具体如文件7-5所示。



#### 源代码

文件7-5

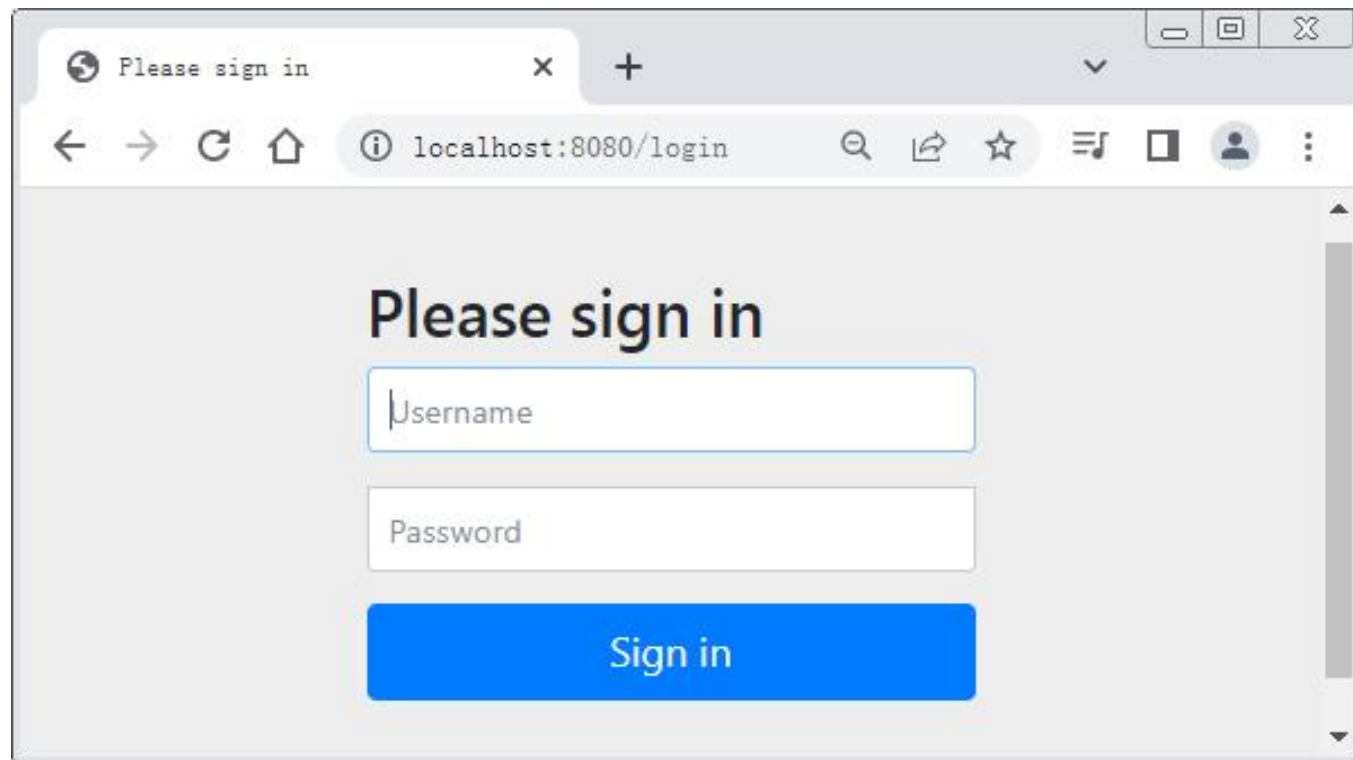
[WebMvcConfig.java](#)





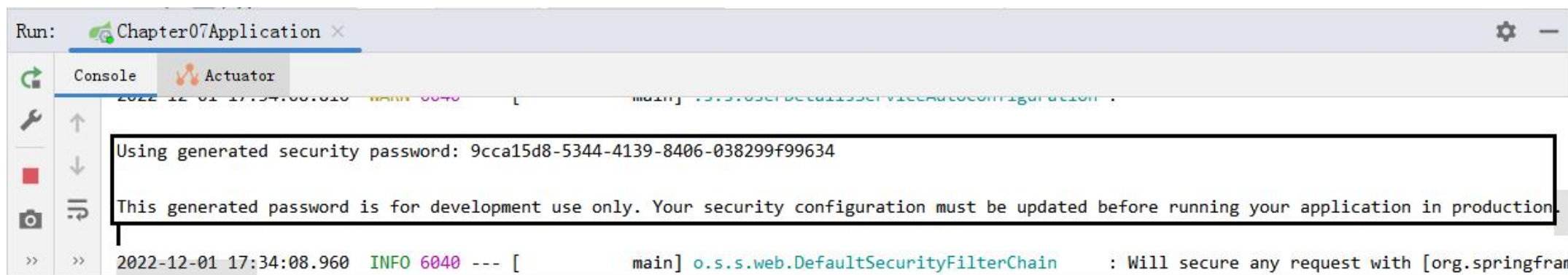
### 5.测试项目效果

启动项目，在浏览器访问 “<http://localhost:8080/>” 。



### 5.测试项目效果

查看IDEA控制台信息，信息中包含一些特别的内容。

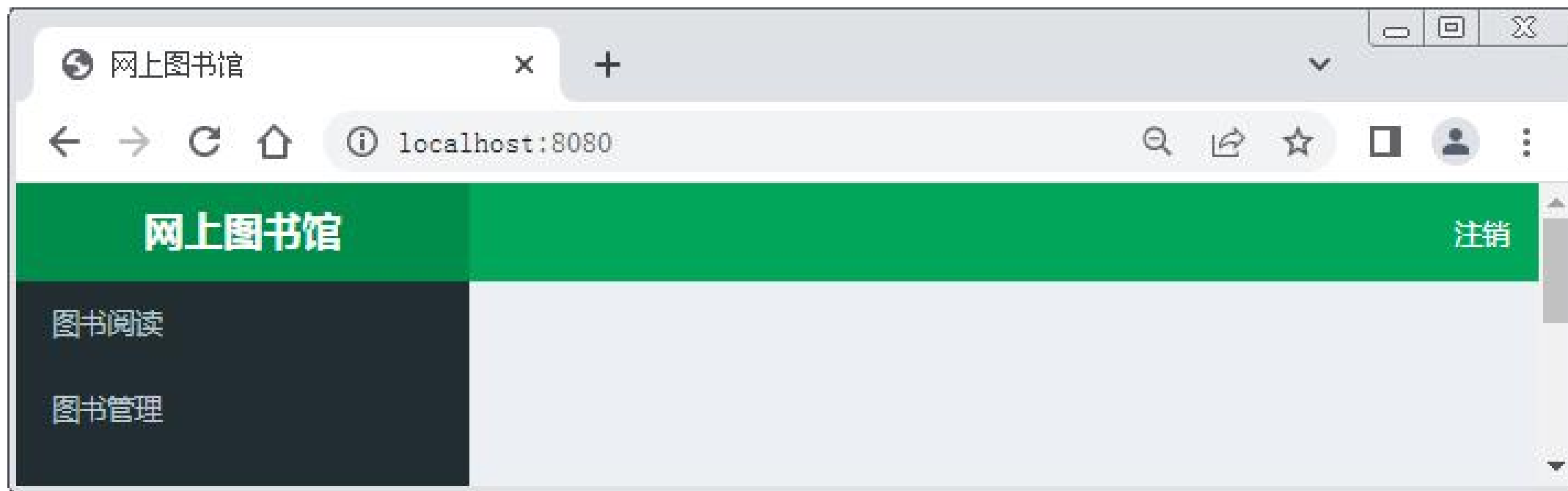


```
Run: Chapter07Application x
Console Actuator
2022-12-01 17:34:08.910 INFO 6040 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.DefaultSecurityFilterChain]
Using generated security password: 9cca15d8-5344-4139-8406-038299f99634
This generated password is for development use only. Your security configuration must be updated before running your application in production.
2022-12-01 17:34:08.960 INFO 6040 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.DefaultSecurityFilterChain]
```



### 5.测试项目效果

在默认登录页面中使用Spring Security提供的账号，以及生成的随机密码进行登录。





了解Spring Security结构总览，能够简述Spring Security过滤器处理请求的流程



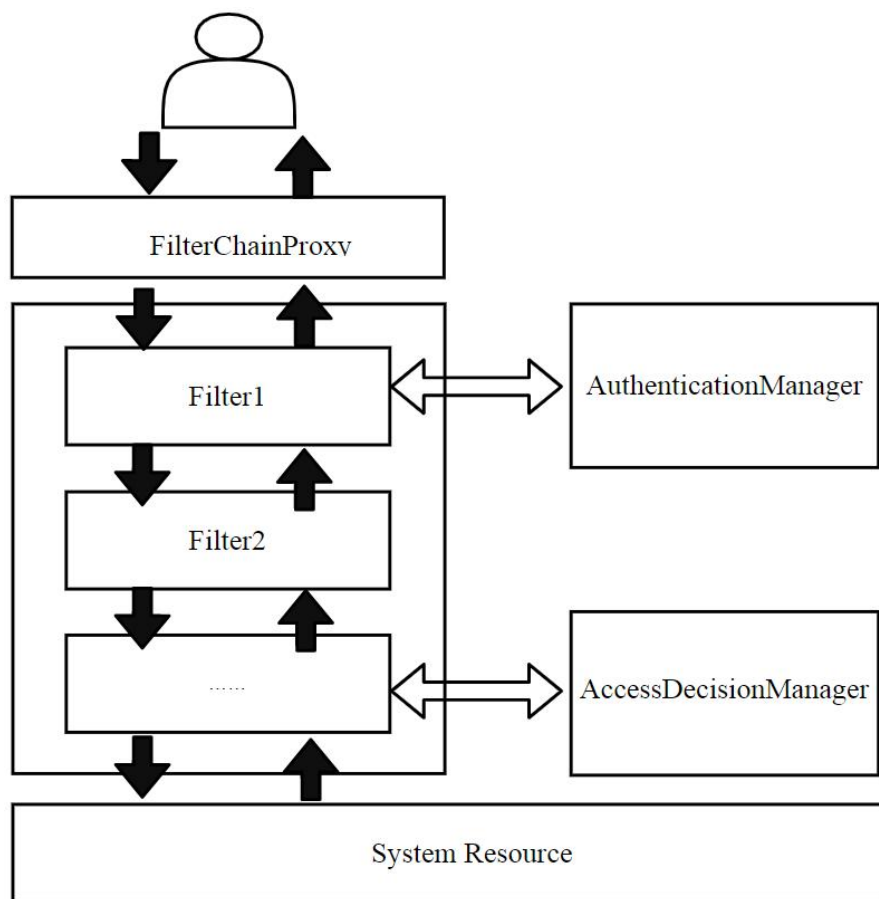
## >>> 7.2.2 Spring Security结构总览



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

当初始化Spring Security时，会创建一个类型为org.springframework.security.web.FilterChainProxy，名称为springSecurityFilterChain过滤器，这个过滤器实现了javax.servlet.Filter接口，外部请求系统资源时会经过此过滤器。





Spring Security安全管理的实现主要是由过滤器链中一系列过滤器相互配合完成，下面对过滤器链中主要的几个过滤器及其作用分别进行说明。

- **SecurityContextPersistenceFilter**: 是整个拦截过程的入口和出口，也就是第一个和最后一个拦截器。
- **UsernamePasswordAuthenticationFilter**: 用于处理来自表单提交的认证，提交的表单必须提供对应的用户名和密码。
- **FilterSecurityInterceptor**: 用于保护Web资源，获取所配置资源访问的授权信息。
- **CsrfFilter**: Spring Security会对所有Post请求验证是否包含系统生成的csrf的token信息，如果不包含，则报错，起到防止csrf攻击的效果。
- **ExceptionTranslationFilter**: 能够捕获来自FilterChain所有的异常，并进行处理。
- **DefaultLoginPageGeneratingFilter**: 如果没有在配置文件中指定认证页面，则由该过滤器生成一个默认认证页面。



## 7.3

# Spring Security认证管理



认证是Spring Security的核心功能之一，Spring Security所提供的认证可以更好地保护系统的隐私数据与资源，只有当用户的身份合法后方可访问该系统的资源。Spring Security提供了默认的认证相关配置，开发者也可以根据自己实际的环境进行自定义身份认证配置。下面对Spring Security的认证流程以及自定义认证进行讲解。



熟悉Spring Security认证流程，能够  
简述Spring Security的认证流程

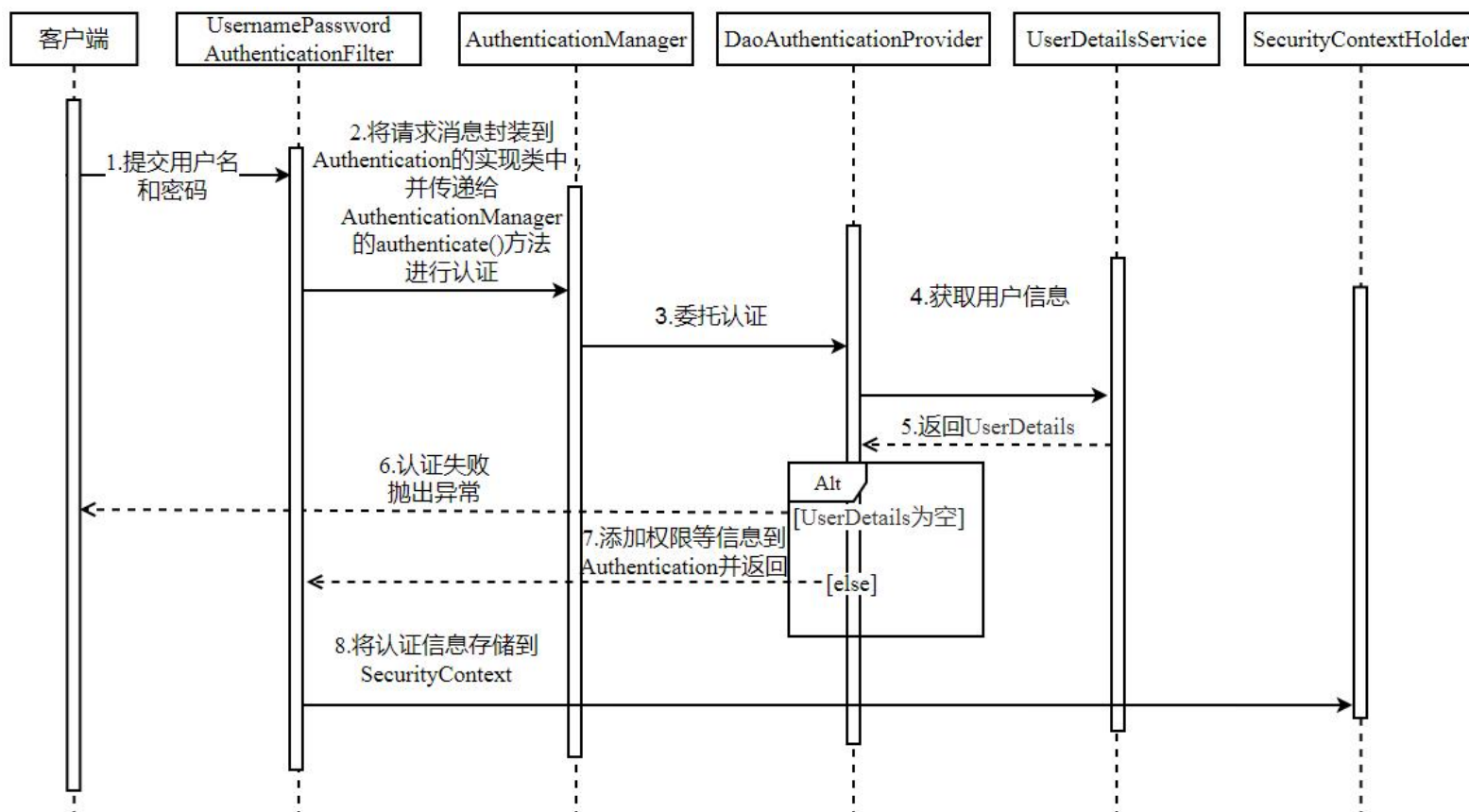
## >>> 7.3.1 Spring Security认证流程



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

用户认证就是判断一个用户的身份是否合法的过程，用户访问系统资源时系统要求验证用户的身份信息，身份合法方可继续访问，否则拒绝其访问。



## >>> 7.3.1 Spring Security认证流程



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

Spring Security的认证流程进行详细介绍。

- ① 用户提交用户名和密码进行认证请求后，被SecurityFilterChain中的 UsernamePasswordAuthenticationFilter过滤器获取到，将用户名和密码封装到UsernamePasswordAuthenticationToken对象中，该对象为Authentication的实现类。
- ② 过滤器将封装用户名和密码的Authentication对象提交至AuthenticationManager（认证管理器）进行认证。
- ③ AuthenticationManager根据当前的认证类型进行认证，认证时会根据提交的用户信息最终返回一个SpringSecurity的UserDetails对象，如果返回的UserDetails对象为空，则说明认证失败，抛出异常。



- ④ 如果返回的UserDetails对象不为空，则返回UserDetails对象，最后AuthenticationManager 认证管理器返回一个被填充满了信息的Authentication 实例，包括权限信息，身份信息，细节信息，但密码通常会被移除。
- ⑤ SecurityContextHolder安全上下文容器存放填充了信息的Authentication，认证成功后通过 SecurityContextHolder.getContext().setAuthentication()方法，将Authentication设置到其中。





掌握Spring Security自定义身份认证，  
能够基于内存身份认证、JDBC身份认  
证和自定义UserService实现用  
户身份认证

尽管项目启动时，Spring Security会提供了默认的用户信息，可以快速认证和启动，但大多数应用程序都希望使用自定义的[用户认证](#)。对于自定义用户认证，Spring Security提供了多种认证方式，常用的有[In-Memory Authentication](#)（内存身份认证）、[JDBC Authentication](#)（JDBC身份认证）和[UserDetailsService](#)（身份详情服务）。下面对Spring Security的这三种自定义身份认证进行详细讲解。



### 1.内存身份认证

以内存身份认证时，需要在Spring Security的相关组件中进行指定当前认证方式为内存身份认证。Spring Security 5.7.1开始Spring Security将WebSecurityConfigurerAdapter类标注为过时，推荐直接声明配置类，在配置类中直接定义组件的信息。

本书使用Spring Boot 2.7.6，其对应的Spring Security版本为5.7.5。自定义内存身份认证时，可以通过InMemoryUserDetailsManager类实现，InMemoryUserDetailsManager是UserDetailsService的一个实现类，方便在内存中创建一个用户。对此，只需在自定义配置类中创建InMemoryUserDetailsManager实例，在该实例中指定该实例的认证信息，并存入在Spring容器中即可。



### 1.内存身份认证

#### (1) 创建配置类

在chapter07项目中创建名为com.itheima.config的包，并在该包下创建一个配置类WebSecurityConfig，在该类中创建UserDetailsService类型的InMemoryUserDetailsManager实例对象交由Spring容器管理，具体如文件7-6所示。



#### 源代码

文件7-6

[WebSecurityConfig.java](#)





### 1.内存身份认证

进行自定义用户认证时，需要注意以下几个问题。

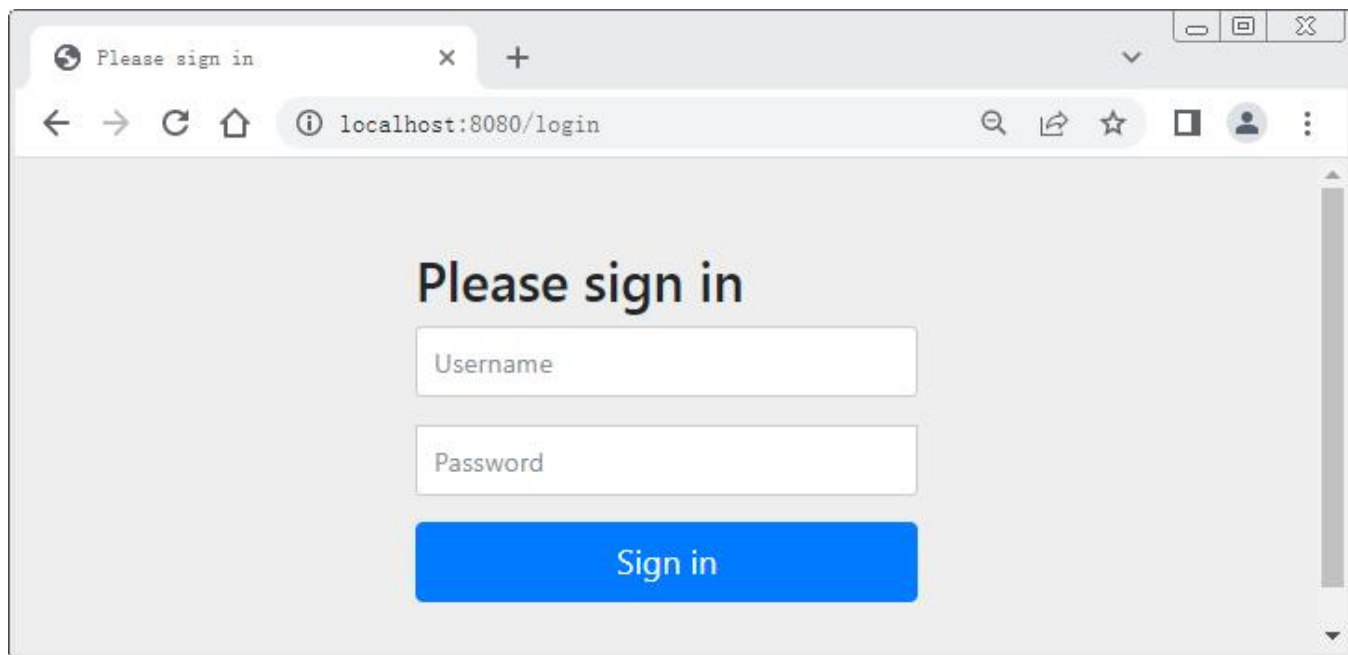
- 提交认证时会对输入的密码使用密码编译器进行加密并与正确的密码进行校验。如果不想对输入的密码进行加密，需要在密码前对使用{noop}进行标注。
- 从Spring Security 5开始，自定义用户认证如果没有设置密码编码器，也没有在密码前使用{noop}进行标注，会认证失败。
- 自定义用户认证时，可以定义用户角色roles，也可以定义用户权限authorities，在进行赋值时，权限通常是在角色值的基础上添加“ROLE\_”前缀。
- 自定义用户认证时，可以为某个用户一次指定多个角色或权限。



### 1.内存身份认证

#### (2) 验证内存身份认证

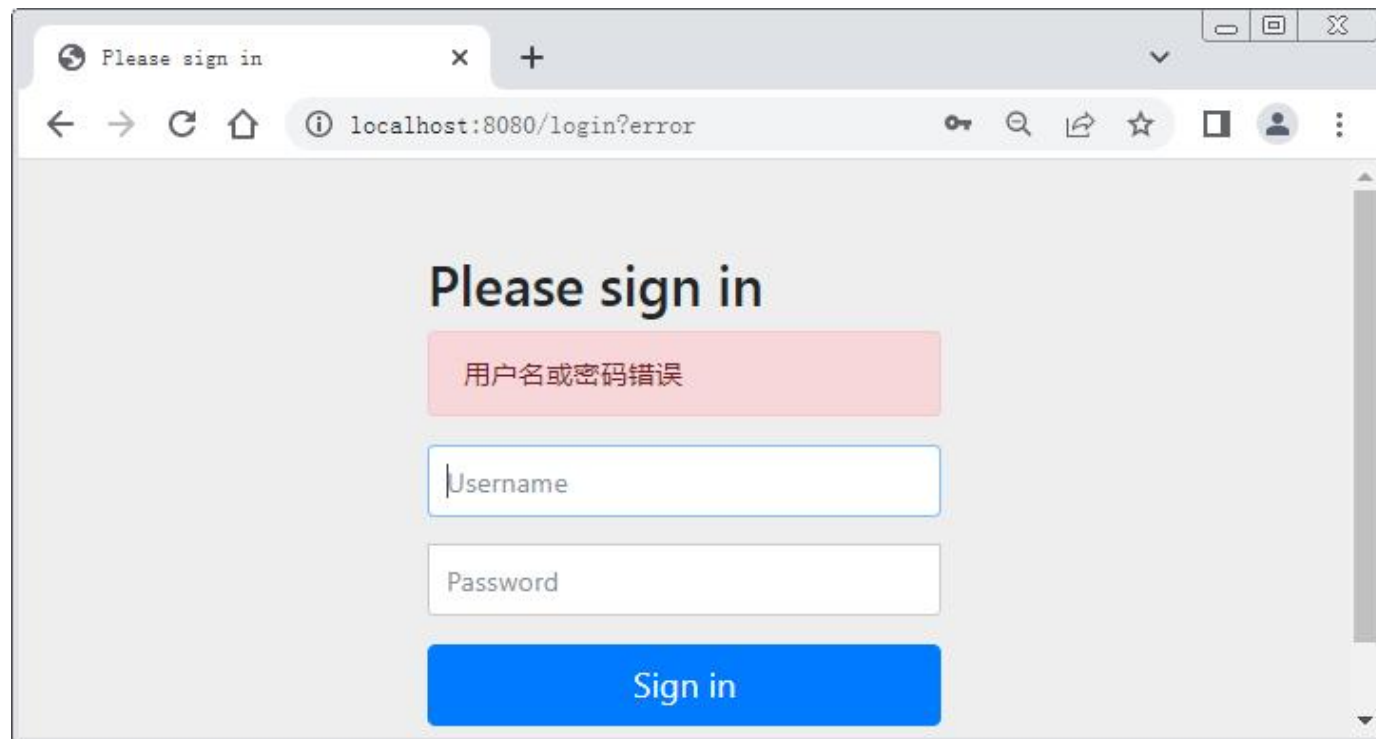
启动项目后，查看控制台输出的信息，发现没有默认安全管理时随机生成了密码。在浏览器访问项目首页 “<http://localhost:8080/>” 。





### 1.内存身份认证

使用不是文件7-6中设置的用户信息进行登录。



## >>> 7.3.2 Spring Security自定义身份认证

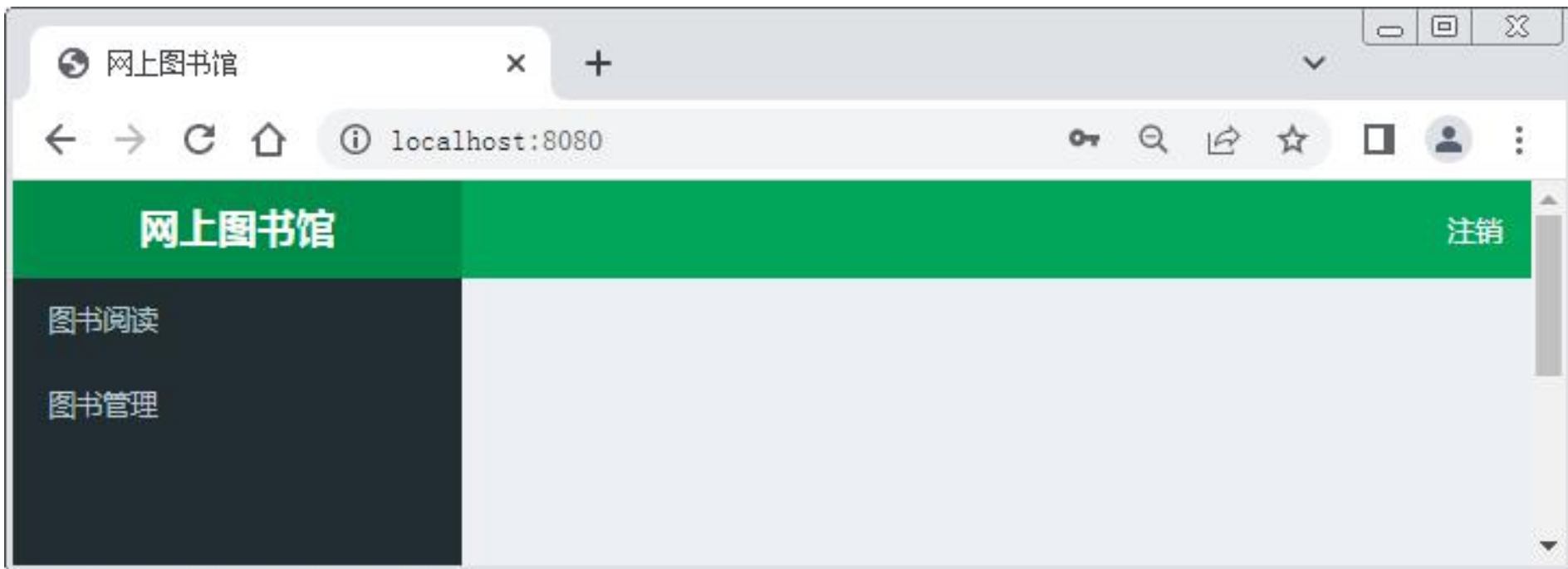


黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 1.内存身份认证

使用文件7-6中设置的用户信息进行登录。







### 2.JDBC身份认证

JDBC身份认证是通过JDBC连接数据库，基于数据库中已有的用户信息进行身份认证，这样避免了内存身份认证的弊端，可以实现对系统已注册的用户进行身份认证。JdbcUserDetailsManager是Spring Security内置的UserDetailsService的实现类，使用JdbcUserDetailsManager可以通过JDBC将数据库和Spring Security连接起来。下面对JDBC身份认证方式进行讲解。



### 2.JDBC身份认证

#### (1) 数据准备

使用之前创建的名为springbootdata的数据库，在该数据库中创建三个表user、priv和user\_priv，并预先插入几条测试数据。准备数据的SQL语句如文件7-7所示。



#### 源代码

文件7-7  
[security.sql](#)





### 2.JDBC身份认证

#### (2) 配置依赖

打开chapter07项目的pom.xml文件，在该文件中添加JDBC的启动器依赖。





### 2.JDBC身份认证

#### (3) 设置配置信息

在项目 chapter07 中创建配置文件 `application.yml`，在该文件中设置数据库连接的相关配置信息，具体如文件 7-8 所示。



#### 源 代 码

文件7-8  
`application.yml`





### 2.JDBC身份认证

#### (4) 修改配置类

修改文件7-6中WebSecurityConfig配置类userDetailsService()方法，将该方法创建的实例对象修改为JdbcUserDetailsManager，修改后的代码如文件7-9所示。



#### 源代码

文件7-9

WebSecurityConfig.java

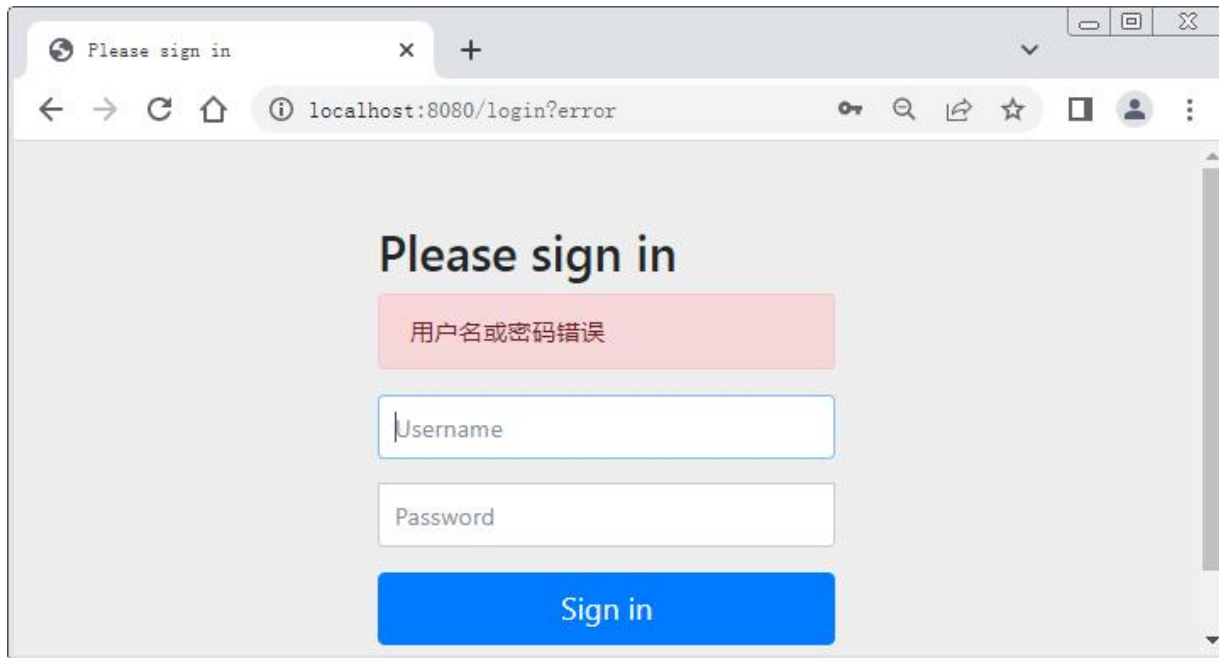




### 2.JDBC身份认证

#### (5) 效果测试

重启chapter07项目进行效果测试，项目启动成功后，通过浏览器访问“<http://localhost:8080/>”后使用错误的数据库用户信息进行登录。



## >>> 7.3.2 Spring Security自定义身份认证

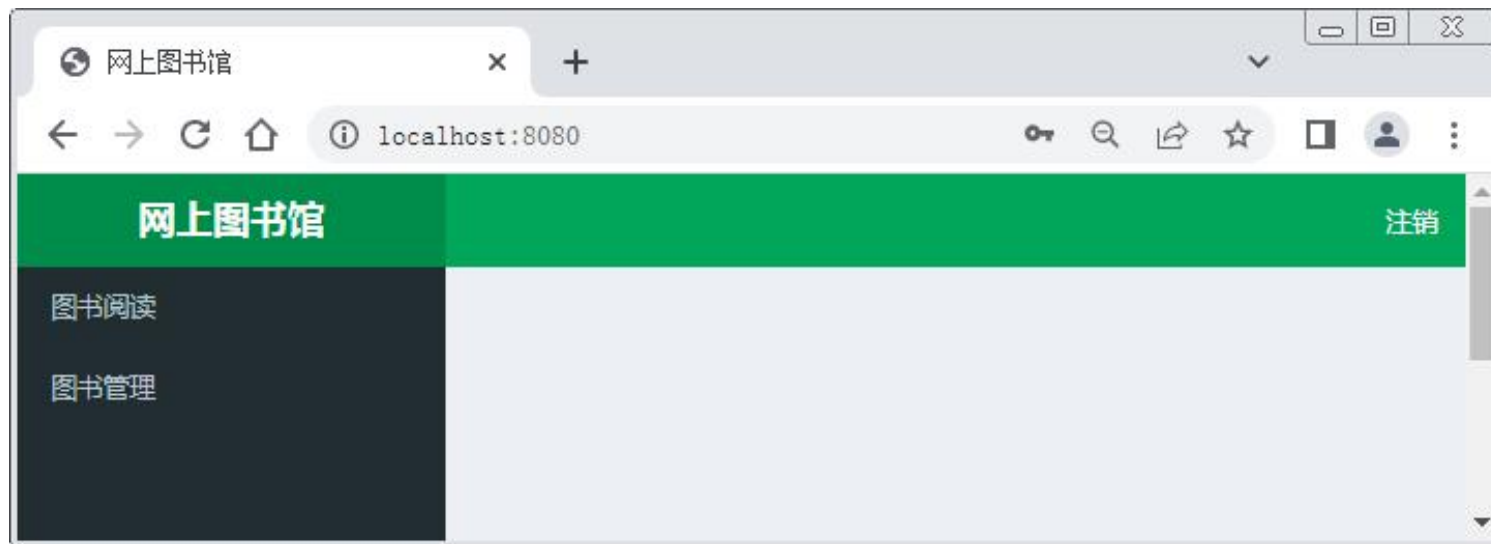


黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 2.JDBC身份认证

使用正确的数据库用户信息进行登录。





### 3.自定义UserDetailsService身份认证

使用InMemoryUserDetailsManager和JdbcUserDetailsManager进行身份认证时，其真正的认证逻辑都在UserDetailsService接口重写的loadUserByUsername()方法中。对于一个完善的项目来说，通常会实现用户信息查询服务，对此可以自定义一个UserDetailsService实现类，重写该接口的loadUserByUsername()方法，在该方法中查询用户信息，将查询到的用户信息填充到UserDetails对象返回，以实现用户的身份认证。下面通过案例对自定义UserDetailsService进行身份验证的实现进行演示。





### 3.自定义UserDetailsService身份认证

#### (1) 创建实体类

在项目chapter07的java目录下创建包com.itheima.chapter07.entity，在该包下创建用户实体类User Dto和权限实体类Privilege，具体如文件7-10和文件7-11所示。



#### 源代码

文件7-10 [UserDto.java](#)

文件7-11 [Privilege.java](#)





### 3.自定义UserDetailsService身份认证

#### (2) 创建用户持久层接口

在项目chapter07的java目录下创建包com.itheima.chapter07.dao，在该包下创建用户持久层接口，在接口中定义查询用户及角色信息的方法，具体如文件7-12所示。



#### 源代码

文件7-12  
UserDao.java





### 3.自定义UserDetailsService身份认证

#### (3) 封装用户认证信息

在项目chapter07的java目录下创建包com.itheima.chapter07.service，在该包下创建UserDetailsService类，该类实现UserDetailsService接口，并在重写的loadUserByUsername()方法中封装用户认证信息，具体如文件7-13所示。



#### 源代码

文件7-13

UserDetailsServiceImpl.java

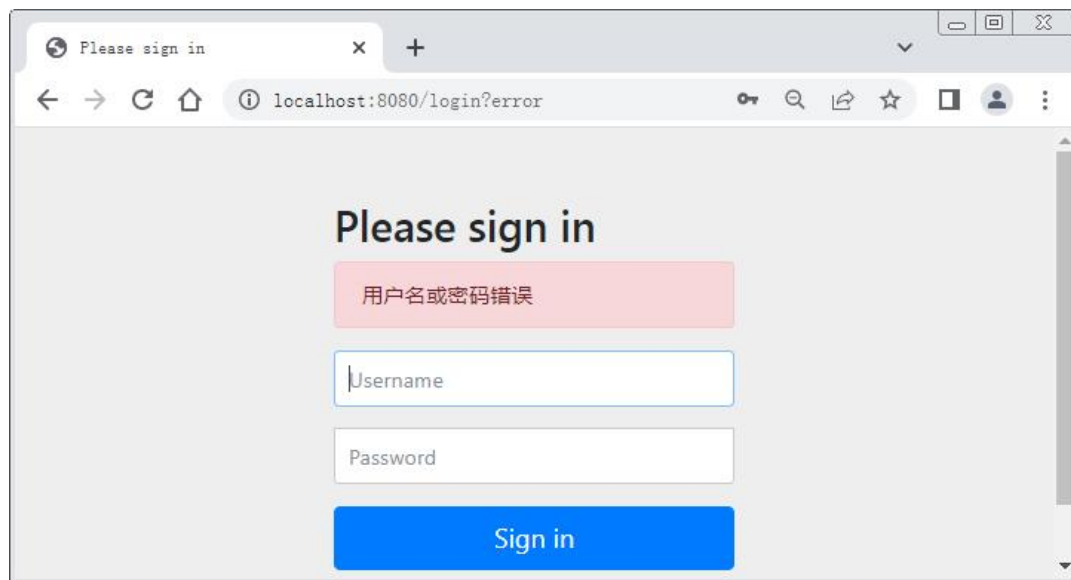




### 3.自定义UserDetailsService身份认证

#### (4) 效果测试

将文件7-8中的userDetailsService()方法进行注释，使用自定义UserDetailsService身份认证。重启chapter07项目进行效果测试，项目启动成功后，在浏览器通过“<http://localhost:8080/>”访问项目首页后，使用错误的数据库用户信息进行登录。



## >>> 7.3.2 Spring Security自定义身份认证

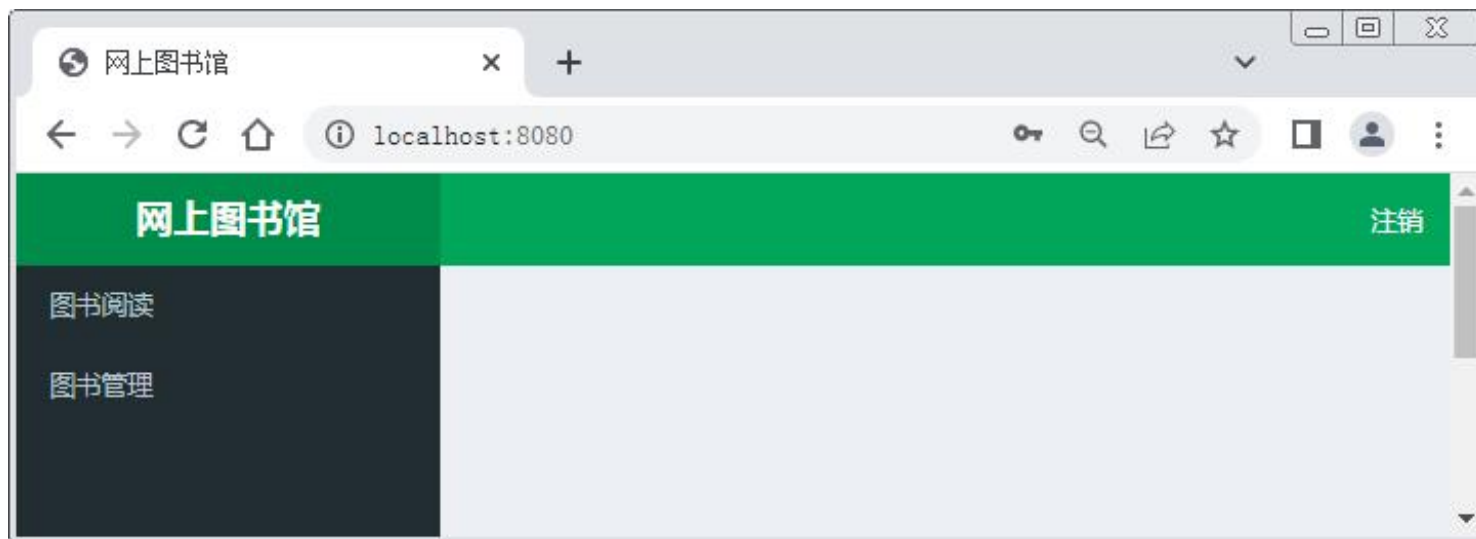


黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 3.自定义UserDetailsService身份认证

使用正确的数据库用户信息进行登录。





## 7.4

# Spring Security授权管理



授权是Spring Security的核心功能之一，是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可正常访问，没有访问的权限时则会被拒绝访问。认证是为了保证用户身份的合法性，而授权则是为了更细粒度地对隐私数据进行划分，授权是在认证通过后发生的，以控制不同的用户访问不同的资源。Spring Security提供了授权方法，开发者通过这些方法进行用户访问控制，下面对Spring Security的授权流程以及自定义授权进行讲解。

## >>> 7.4.1 Spring Security授权流程



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌



熟悉Spring Security授权流程，能够  
简述Spring Security的授权流程



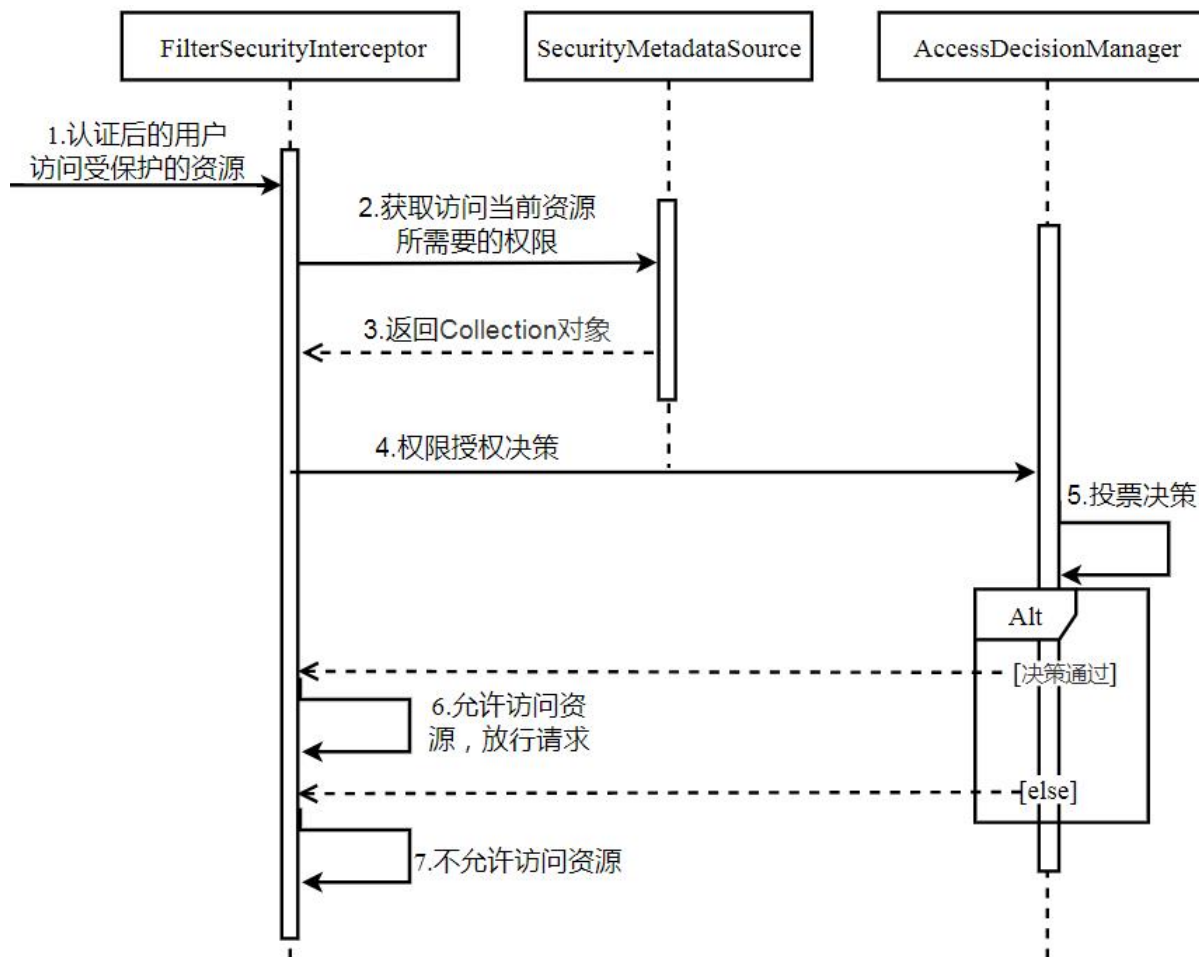
## >>> 7.4.1 Spring Security授权流程



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

实现授权需要对用户的访问进行拦截校验，校验用户的权限是否可以操作指定的资源。Spring Security使用标准Filter建立了对Web请求的拦截，最终实现对资源的授权访问。





- ①**拦截请求**。已认证用户访问受保护的Web资源将被SecurityFilterChain中FilterSecurityInterceptor实例对象拦截。
- ②**获取资源访问策略**。FilterSecurityInterceptor实例对象会通过SecurityMetadataSource的子类DefaultFilterInvocationSecurityMetadataSource实例对象中**获取要访问当前资源**所需要的**权限**，权限**封装在Collection**实例对象中。SecurityMetadataSource是读取访问策略的抽象，具体读取的内容，就是**开发者配置的访问规则**。
- ③FilterSecurityInterceptor通过AccessDecisionManager进行**授权决策**，若决策通过，则允许访问资源，否则将禁止访问。AccessDecisionManager中包含一系列AccessDecisionVoter，可对当前认证过的身份是否有权访问对应的资源进行投票，AccessDecisionManager根据**投票结果**做出**最终决策**。



掌握Spring Security自定义授权，能够使用Web授权和方法授权实现用户授权管理



根据授权的**位置和形式**，通常可以将授权的方式分为**Web授权**和**方法授权**，这两种授权方式都会调用**AccessDecisionManager**进行授权决策。下面分别对这两种自定义授权的方式进行讲解。

### 1.Web授权

Spring Security的底层实现本质是通过多个**Filter**形成的**过滤器链**完成，过滤器链中提供了**默认的安全拦截机制**，设置安全拦截规则，以**控制**用户的**访问**。**HttpSecurity**是SecurityBuilder接口的实现类，是HTTP安全相关的构建器，Spring Security中可以通过HttpSecurity对象**设置安全拦截规则**，并通过该对象**构建过滤器链**。

1.Web授权

HttpSecurity可以根据不同的业务场景，对不同的URL采用不同的权限处理策略。当开发者需要配置项目的安全拦截规则时，可以调用HttpSecurity对象对应的方法实现。

HttpSecurity类的常用方法

方法	作用
authorizeRequests()	开启基于HttpServletRequest请求访问的限制
formLogin()	开启基于表单的用户登录
httpBasic()	开启基于HTTP请求的Basic认证登录
logout()	开启退出登录的支持
sessionManagement()	开启Session管理配置
rememberMe()	开启“记住我”功能
csrf()	配置CSRF跨站请求伪造防护功能

1.Web授权

通过authorizeRequests()方法可以添加用户请求控制的规则，这些规则通过用户请求控制的相关方法指定。

用户请求控制的常用方法

方法	作用
antMatchers(String... antPatterns)	开启Ant风格的路径匹配
mvcMatchers(String... patterns)	开启MVC风格的路径匹配，与Ant风格类似
regexMatchers(String... regexPatterns)	开启正则表达式的路径匹配
and()	功能连接符
anyRequest()	匹配任何请求
rememberMe()	开启“记住我”功能
access(String attribute)	使用基于SpEL表达式的角色进行匹配

### 1.Web授权

用户请求控制的常用方法

方法	作用
hasAnyRole(String... roles)	匹配用户是否有参数中的任意角色
hasRole(String role)	匹配用户是否有某一个角色
hasAnyAuthority(String... authorities)	匹配用户是否有参数中的任意权限
hasAuthority(String authority)	匹配用户是否有某一个权限
authenticated()	匹配已经登录认证的用户
fullyAuthenticated()	匹配完整登录认证的用户（非rememberMe登录用户）
hasIpAddress(String ipAddressExpression)	匹配某IP地址的访问请求
permitAll()	无条件对请求进行放行

### 1.Web授权

通过HttpSecurity类的formLogin()方法开启基于表单的用户登录后，可以指定表单认证的相关设置。

基于表单的身份验证的常见方法

方法	作用
loginPage(String loginPage)	指定自定义登录界面，不使用SpringSecurity默认登录界面
loginProcessingUrl(String loginProcessingUrl)	指定处理登录的请求url，为表单提交用户信息的Action
successForwardUrl(String forwardUrl)	指定登录成功后默认跳转的路径





### 1.Web授权

下面通过案例演示在Spring Boot项目中使用Spring Security的Web授权方式进行权限管理。

#### (1) 导入登录页面

在项目chapter07的resources目录的templates文件夹中导入自定义的登录页面。





### 1.Web授权

#### (2) 编辑配置类

在项目chapter07的WebSecurityConfig配置类中使用HttpSecurity对象设置安全拦截规则，并创建SecurityFilterChain对象交由Spring管理，具体代码如文件7-15所示。



#### 源代码

文件7-15

[WebSecurityConfig.java](#)





### 1.Web授权

#### (2) 编辑配置类

文件7-5的WebMvcConfig配置类中添加loginview的视图映射，具体代码如文件7-16所示。



#### 源代码

文件7-16

[WebMvcConfig.java](#)



## 7.4.2 Spring Security自定义授权



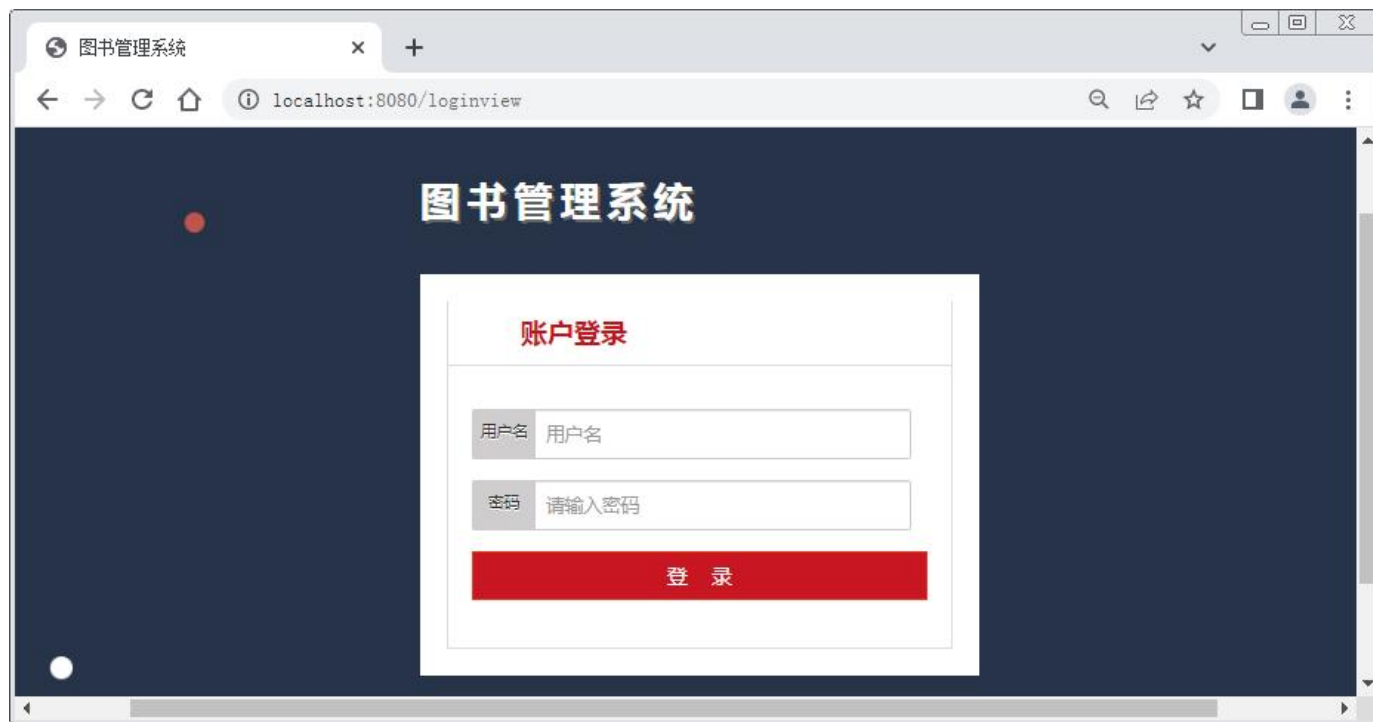
黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 1.Web授权

#### (3) 测试效果

启动项目chapter07，在浏览器中通过“<http://localhost:8080/>”访问项目首页。



## >>> 7.4.2 Spring Security自定义授权

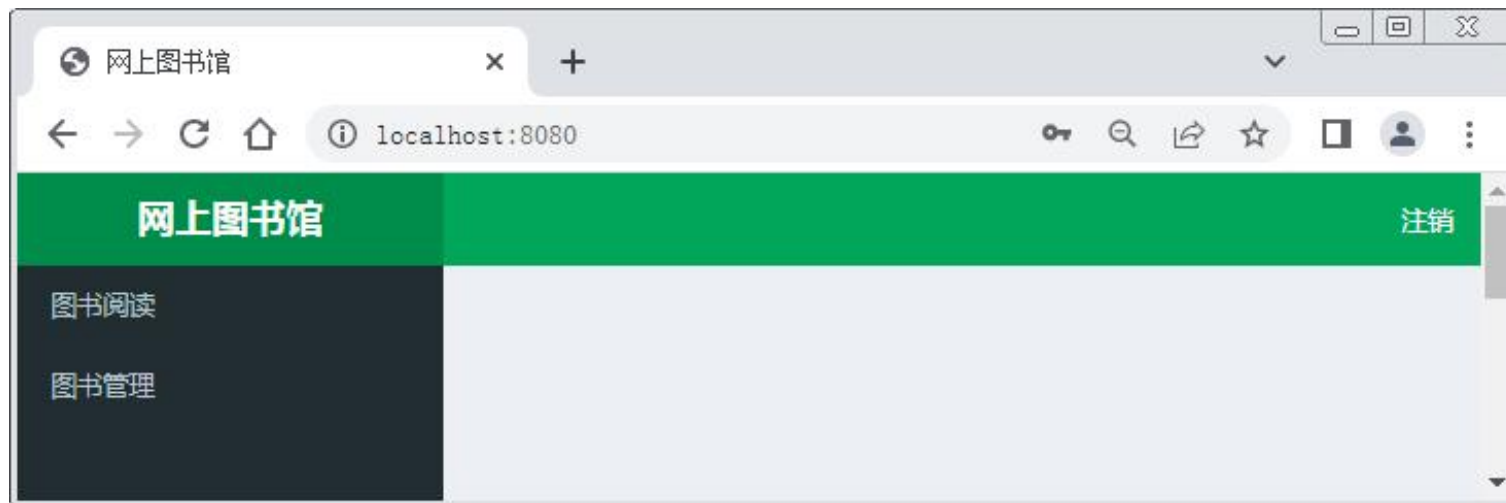


黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 1.Web授权

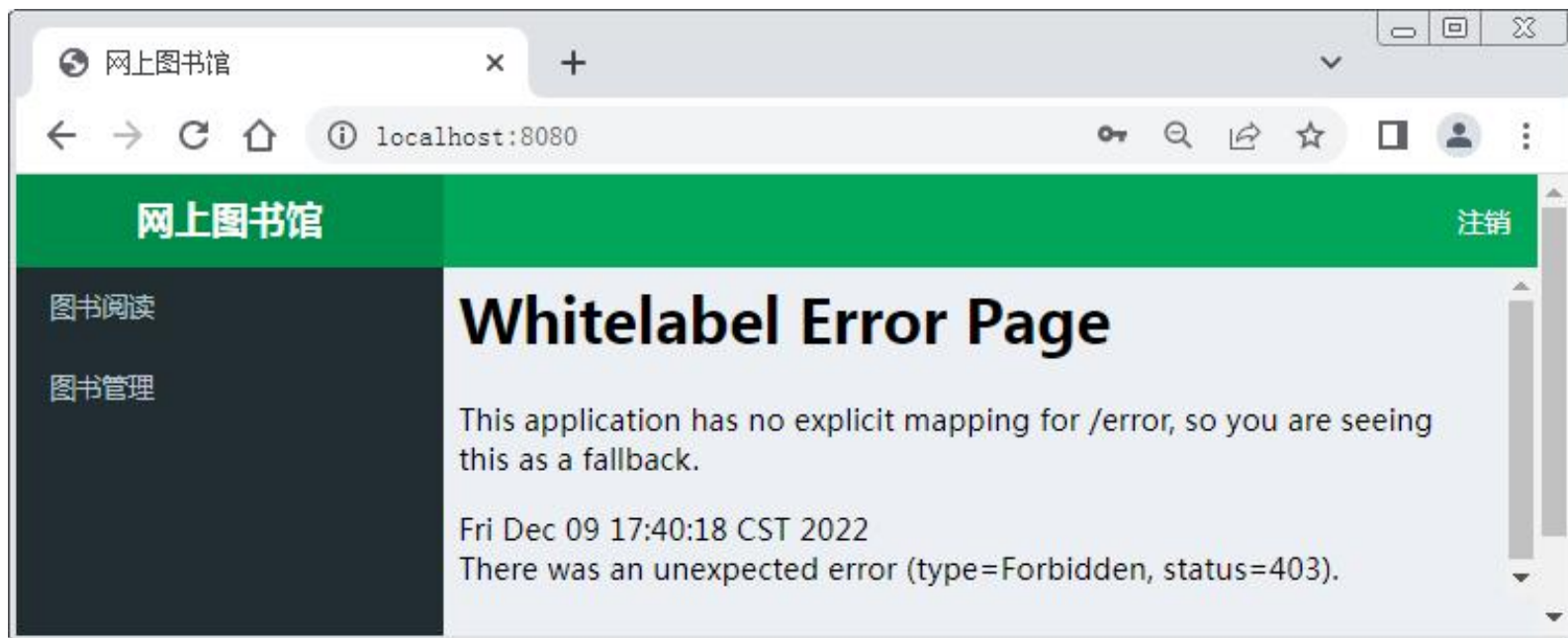
在登录页面使用zhangsan的用户信息进行登录。





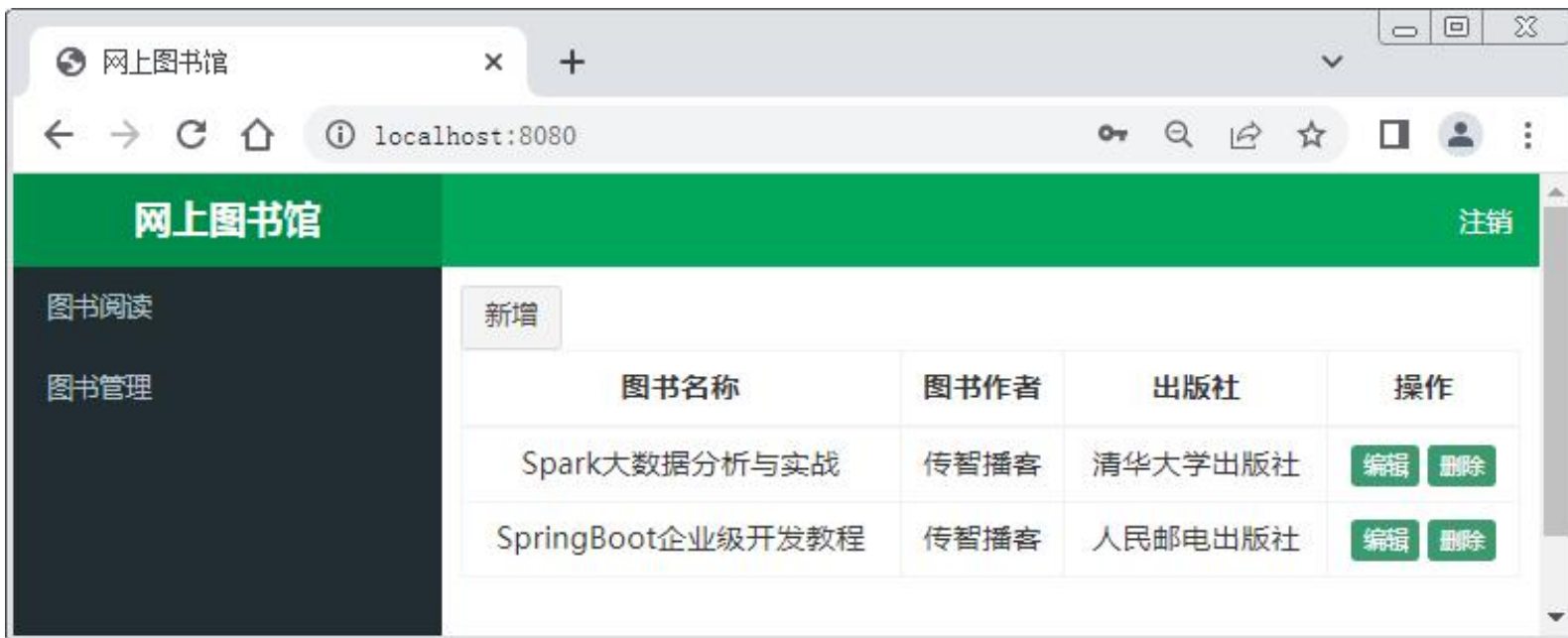
### 1.Web授权

图书管理需要角色为ROLE\_ADMIN的用户才可以访问，用户zhangsan的角色为ROLE\_COMMON，在后台首页单击“图书管理”链接。



### 1.Web授权

使用用户lisi进行登录，lisi对应的角色为ROLE\_ADMIN，登录成功再次访问“图书管理”。





### 2.方法授权

Spring Security除了可以在配置类中通过创建过滤器链设置安全拦截规则外，还可以使用@Secured、@RolesAllowed和@PreAuthorize注解控制类中所有方法或者单独某个方法的访问权限，以实现访问授权管理。





### 2.方法授权

使用@Secured和@RolesAllowed注解时，只需在注解中指定访问当前注解标注的类或方法所需要具有的角色，允许多个角色访问时，使用大括号对角色信息进行包裹，角色信息之间使用分号分隔即可。

```
@RequestMapping("list")
@Secured({"ROLE_ADMIN","ROLE_COMMON"})
public String findList() {
    return "book_list";
}

@RequestMapping("admin/manag")
@RolesAllowed("ROLE_ADMIN")
public String findManagList() {
    return "book_manag";
}
```



### 2.方法授权

@PreAuthorize注解会在方法执行前进行权限验证，支持SpEL表达式。

```
@RequestMapping("list")
@PreAuthorize("hasAnyRole('ROLE_ADMIN','ROLE_COMMON')")
public String findList() {
    return "book_list";
}

@RequestMapping("admin/manag")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String findManagList() {
    return "book_manag";
}
```



### 2.方法授权



@Secured、@RolesAllowed和@PreAuthorize注解都可以对方法的访问进行权限控制。

@Secured为Spring Security提供的注解。

@RolesAllowed为基于JSR 250规范的注解。

@PreAuthorize支持SpEL表达式。



### 2.方法授权

Spring Security默认是禁用方法级别的安全控制注解，要想使用注解进行方法授权，可以使用 `@EnableGlobalMethodSecurity` 注解开启基于方法的安全认证机制，该注解可以标注在任意配置类上。

```
@Configuration
```

```
@EnableGlobalMethodSecurity(securedEnabled = true, jsr250Enabled = true,  
    prePostEnabled = true)
```

```
public class WebSecurityConfig {.....}
```



### 2.方法授权

#### (1) 开启基于方法的安全认证机制

在项目chapter07的WebSecurityConfig配置类中开启基于方法的安全认证机制，并将类中HttpSecurity对象设置的访问“图书管理”拦截规则删除，以确保对资源授权地为方法授权，修改后的代码如文件7-17所示。



#### 源代码

文件7-17

[WebSecurityConfig.java](#)





### 2.方法授权

#### (2) 方法授权

在文件7-4 `BookController`类的`findManagList()`方法上使用注解指定访问该方法所需的角色，具体如文件7-18所示。



#### 源代码

文件7-18

[BookController.java](#)

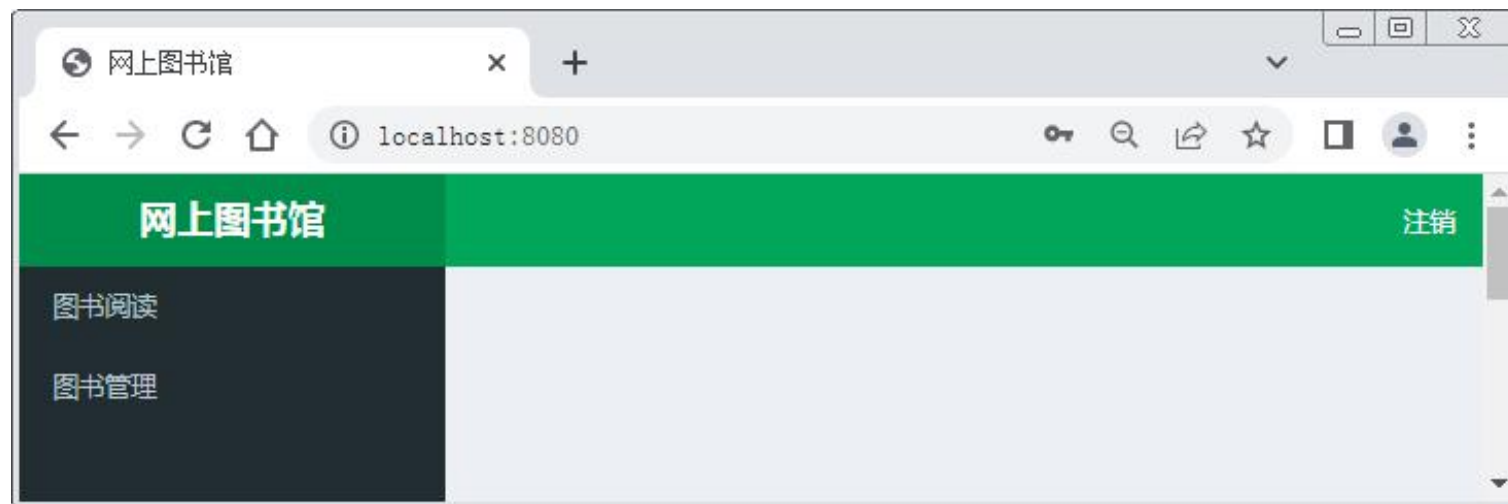




### 2.方法授权

#### (3) 测试效果

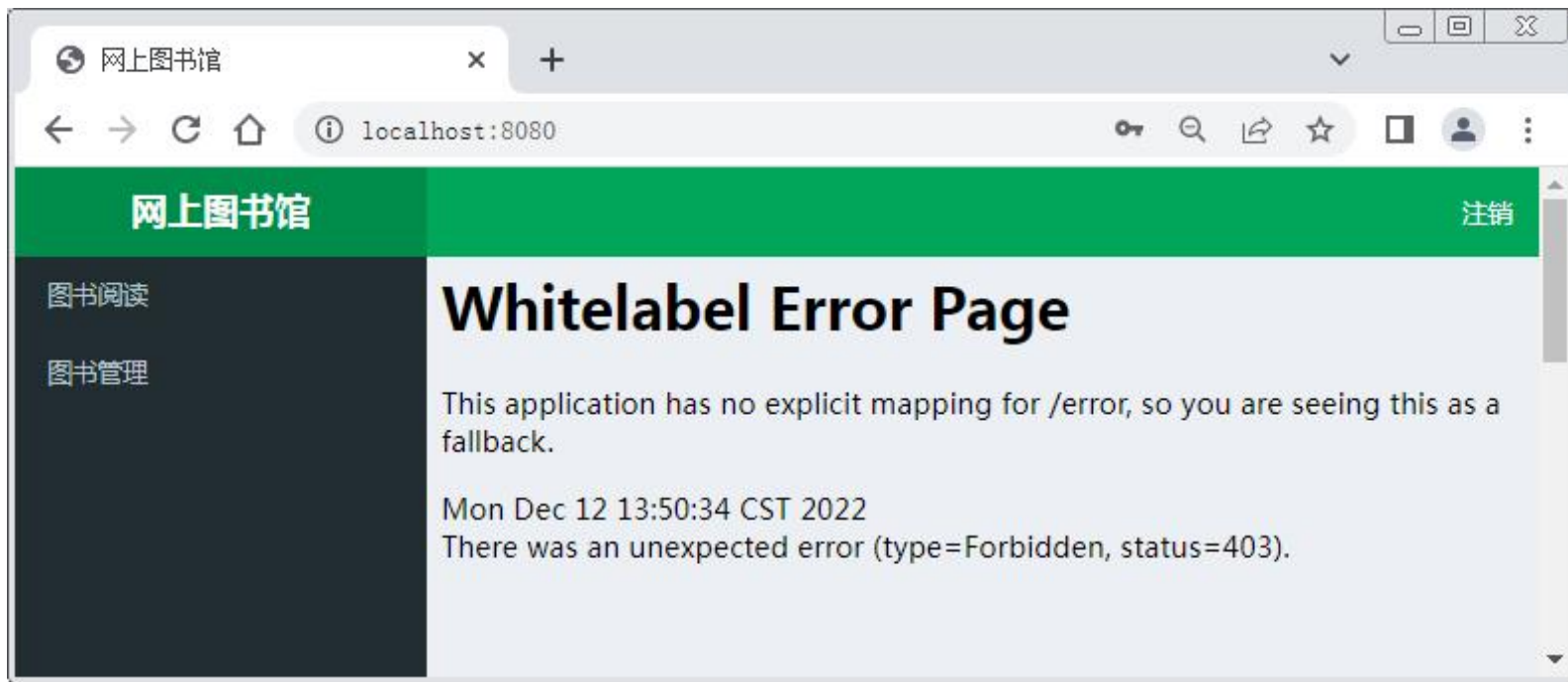
启动项目chapter07，在浏览器中通过“<http://localhost:8080/>”访问项目首页后，使用用户zhangsan登录系统。





### 2.方法授权

单击左侧的“图书管理”链接。

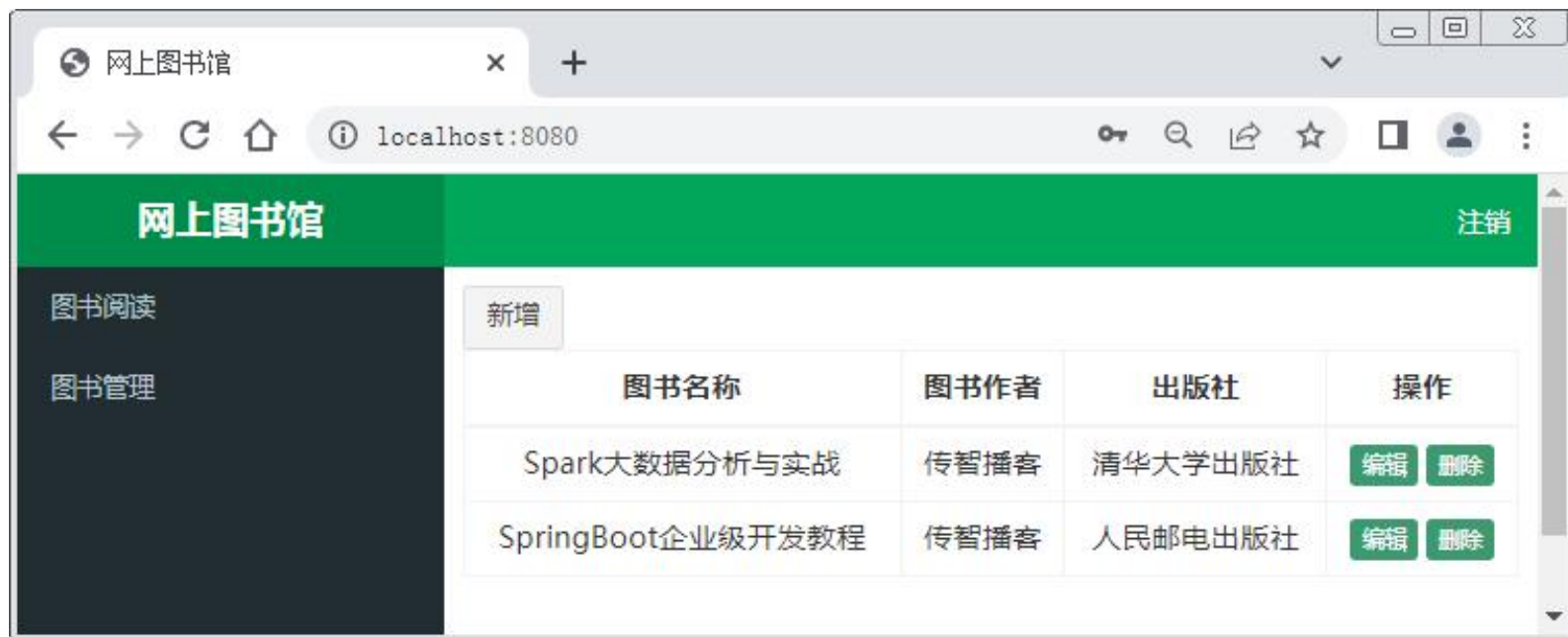






### 2.方法授权

在用户登录页面使用用户lisi进行登录，登录成功后再次访问“图书管理”。





掌握动态展示菜单，能够通过Spring Security的授权管理实现动态展示菜单



在**前面**的讲解中，只是通过Spring Security对后台资源的访问根据角色进行权限控制，前端页面并没有做任何处理，**不同角色**能看到的前端页面是一样的，即使当前用户**没有对应的访问权限**，依然**能看到对应的菜单**，用户体验较差。下面在前面案例的基础上，讲解如何使用Spring Security与Thymeleaf整合实现前端页面**根据**登录用户的**角色动态展示菜单**。



## 7.4.3 动态展示菜单



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

### 1. 添加依赖

在chapter07项目的pom.xml文件中添加Thymeleaf与Spring Security 5的集成包：[thymeleaf-extras-springsecurity5](#)。





### 2. 修改页面代码

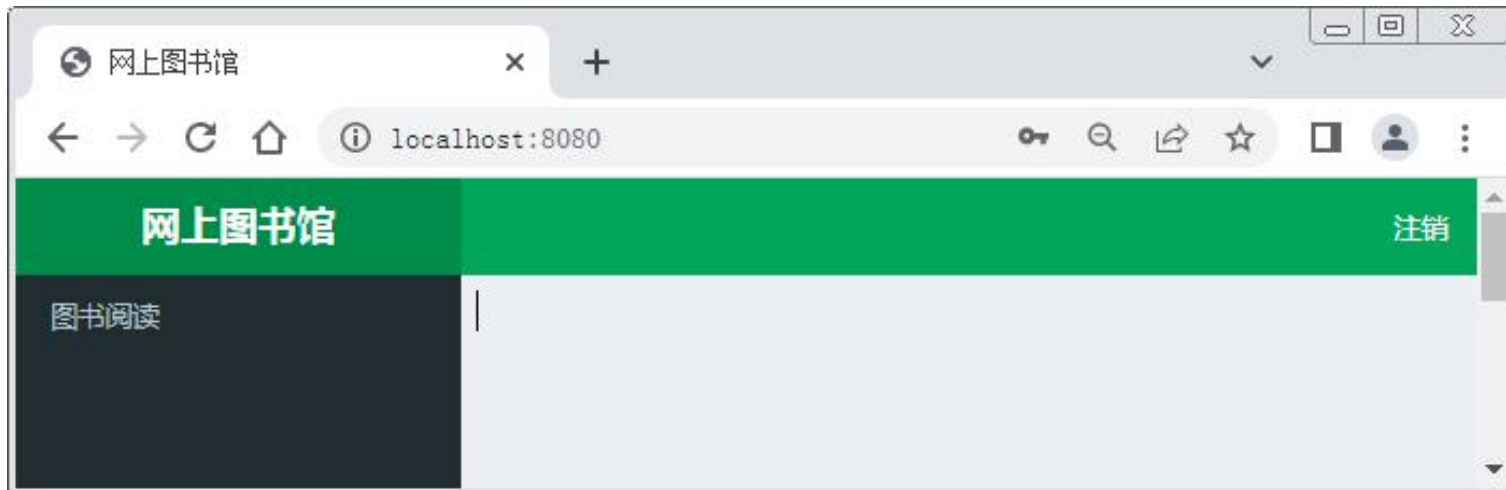
打开chapter07项目后台首页[main.html](#)，引入Spring Security安全标签，并在页面中根据需求使用Spring Security标签指定为不同角色显示不同的页面内容，实现动态展示控制，具体如文件7-19所示。





### 3. 效果测试

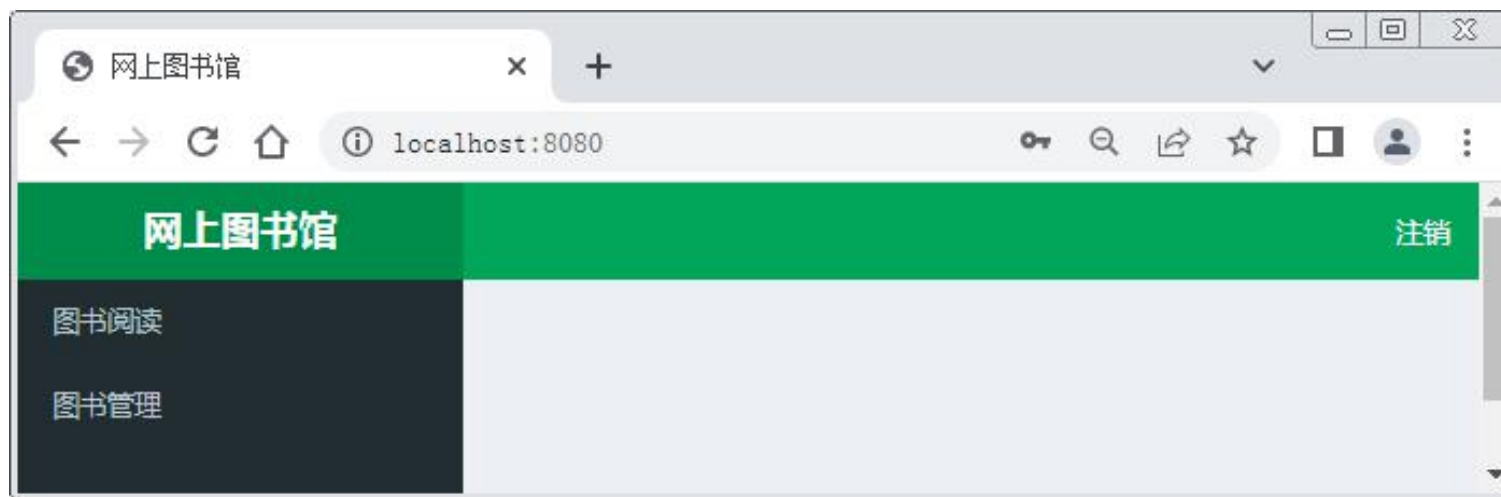
重启chapter07项目进行效果测试，项目启动后，使用用户zhangsan的信息进行登录，登录后展示后台首页。





### 3. 效果测试

使用用户lisi的信息进行登录，登录后展示后台首页。





## 7.5

# Spring Security会话管理和 用户退出



## 7.5 Spring Security会话管理和用户退出



黑马程序员  
www.itheima.com

传智教育旗下  
高端IT教育品牌

用户认证通过后，为了避免用户的每次操作都进行认证，可以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。Spring Security提供会话管理功能，只需要配置即可使用。同时，如果想结束当前会话，可以在自定义退出功能中销毁会话中的用户信息。下面将在Spring Boot项目中基于Spring Security实现会话管理和用户退出。



掌握Spring Security会话管理，能够在Spring Security中获取认证后的用户信息，以及进行会话控制



### 1. 获取用户信息

Spring Security对用户信息认证通过后，会将用户信息存入Spring Security应用的上下文对象SecurityContext中，SecurityContext与当前线程进行绑定，需要获取用户信息时，可以通过SecurityContextHolder获取SecurityContext对象，进而使用SecurityContext对象获取用户信息。

```
Authentication authentication =  
    SecurityContextHolder.getContext().getAuthentication();  
if (!authentication.isAuthenticated()) {  
    return null;  
}  
UserDetails userDetails = (UserDetails) authentication.getPrincipal();  
String username = userDetails.getUsername();
```



### 2.会话控制

默认情况下，Spring Security会为每个登录成功的用户新建一个Session对象进行会话管理，开发者也可以根据具体的需求对Session的创建进行控制。Spring Security管理Session的创建策略有以下四种。

- **always**：如果没有Session就创建一个。
- **ifRequired**：如果需要就创建一个Session，是默认的创建策略。
- **never**：Spring Security将不会创建Session，但是如果项目中其他地方创建了Session，那么Spring Security可以使用它。
- **stateless**：Spring Security将绝对不会创建Session，也不使用Session。



### 2.会话控制

当项目中不想自动创建Session，但是想要使用项目中其他地方创建的Session时，可以选择使用never策略。

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.NEVER);
    return http.build();
}
```



### 2.会话控制

Spring Security对用户信息认证通过后，会将用户信息存入Spring Security应用的上下文对象。如果项目中允许创建Session，默认情况下Session的超时时间为30分钟，如果想要对Session默认的超时时间进行修改，可以在Spring Boot的配置文件中设置Session的超时时间。

```
server:  
  servlet:  
    session:  
      timeout: 86400s
```



掌握Spring Security用户退出，能够在Spring Boot项目中实现使用Spring Security实现用户退出



Spring security默认实现了用户退出的功能，用户退出主要考虑退出后会话如何管理以及跳转到哪个页面。HttpSecurity类提供了logout()方法开启退出登录的支持，默认触发用户退出操作的URL为“/logout”，用户退出时同时也会清除Session等默认用户配置。

用户退出登录的逻辑是由过滤器LogoutFilter执行的，但是项目开发时一般不会选择直接操作LogoutFilter，而是通过LogoutConfigurer对LogoutFilter进行配置，HttpSecurity类logout()方法的返回值就是一个LogoutConfigurer对象，该对象提供了一系列设置用户退出的方法。



用户退出的主要方法

方法	作用
logoutUrl(String logoutUrl)	用户退出处理控制URL，默认为post请求的/logout
logoutSuccessUrl(String logoutSuccessUrl)	用户退出成功后的重定向地址
logoutSuccessHandler( LogoutSuccessHandler logoutSuccessHandler)	用户退出成功后的处理器设置
deleteCookies(String... cookieNamesToClear)	用户退出后删除指定Cookie
invalidateHttpSession( boolean invalidateHttpSession)	用户退出后是否立即清除Session，默认为true
clearAuthentication( boolean clearAuthentication)	用户退出后是否立即清除Authentication用户认证信息，默认为true



下面通过案例演示用户退出的实现。

### 1. 设置用户退出的请求路径

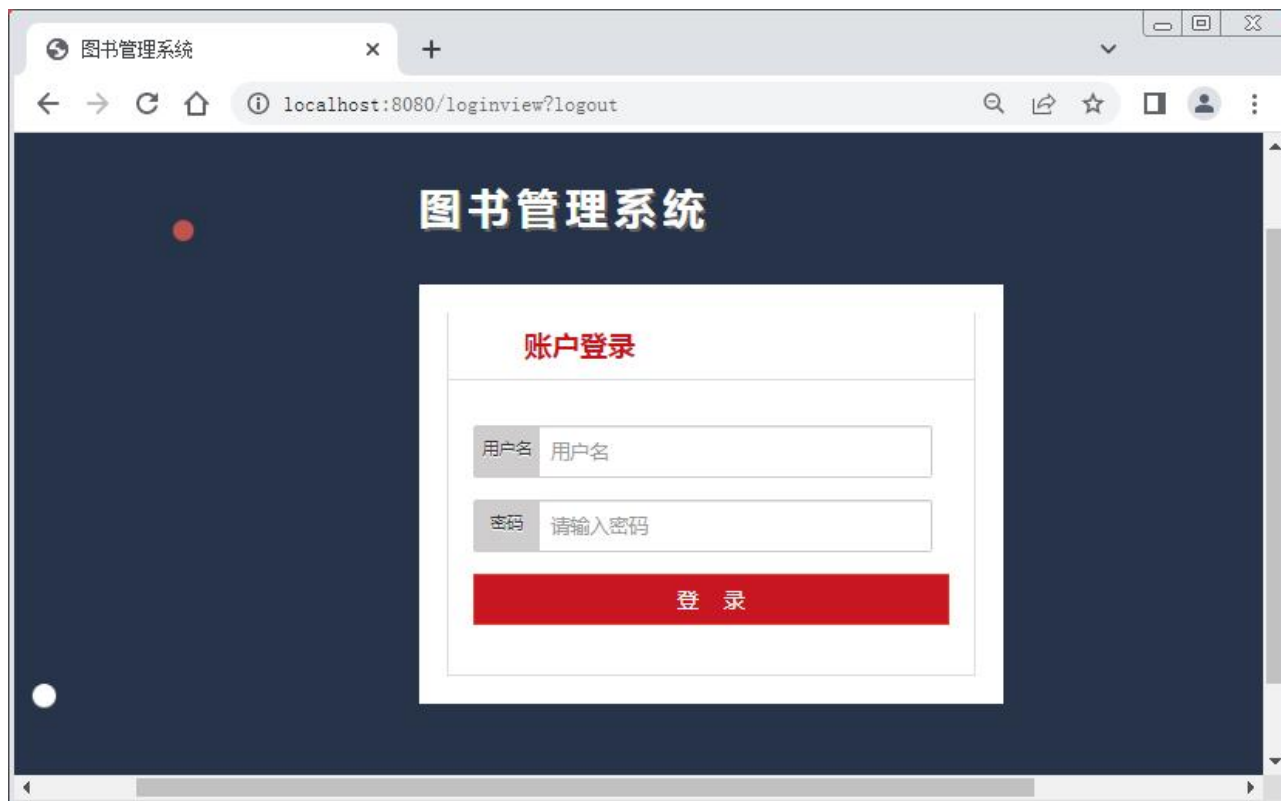
项目后台首页main.html右上方有一个“注销”菜单，可以在该菜单中设置用户退出的请求路径，具体如文件7-20所示。





### 2. 效果测试

重启chapter07项目进行效果测试，项目启动成功后，使用正确的用户信息登录后进入后台首页，单击后台首页页面右上角的“注销”。





### 本章小结

本章主要对Spring Boot项目中的安全管理进行了讲解。首先讲解了安全框架概述；然后讲解了Spring Security基础入门；接着讲解了Spring Security认证管理和授权管理；最后对Spring Security会话管理和用户退出进行了讲解。通过本章的学习，希望大家可以在Spring Boot项目中正确使用Spring Security进行安全管理。

為千萬學生少走彎路而著書



黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌