

第一章 Spring Security 概要

1.1 Spring Security 介绍

1. Spring Security 是基于 Spring 的身份认证 (Authentication) 和用户授权 (Authorization) 框架，提供了一套 Web 应用安全性的完整解决方案。其中核心技术使用了 Servlet 过滤器、IOC 和 AOP 等。

2. 什么是身份认证

身份认证指的是用户去访问系统资源时，系统要求验证用户的身份信息，用户身份合法才访问对应资源。

常见的身份认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。

3. 什么是用户授权

当身份认证通过后，去访问系统的资源，系统会判断用户是否拥有访问该资源的权限，只允许访问有权限的系统资源，没有权限的资源将无法访问，这个过程叫用户授权。

比如 会员管理模块有增删改查功能，有的用户只能进行查询，而有的用户可以进行修改、删除。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

1.2 Shiro 与 Spring Security 对比

1.2.1 Shiro 特点

1. Shiro 是 Apache 下的项目，相对简单、轻巧，更容易上手使用。
2. Shiro 权限功能基本都能满足，单点登录都可以实现。且不用与任何的框架或者容器绑定，可以独立运行。

1.2.2 Spring Security 特点

1. Spring Security 相对 Shiro 上手更复杂；
2. Spring Security 功能比 Shiro 更加丰富些；
3. Spring Security 是 Spring 家族的产品，与 Spring 无缝对接，社区资源相对比 Shiro 更加丰富；
4. Spring Security 对 Oauth2 也有支持，Shiro 则需要自己手动实现。而且 Spring Security 的权限细粒度更高。

1.2.3 如何选择

1. 如果项目中不是很庞大，没有用到 Spring，那就不要考虑使用 Spring Security，Shiro 足够满足，建议使用。
2. 如果项目使用 Spring 作为基础，配合 Spring Security 做权限更加方便，而 Shiro 需要和 Spring 进行整合开发。

第二章 开发环境搭建

2.1 框架版本号

依赖	版本
Spring Boot	2.2.0.RELEASE
Spring Security	5.2.0.RELEASE

2.2 JDK 1.8

JDK 软件位于：03-配套软件\jdk-8u151-windows-x64.exe

2.2 MySQL 5.7

MySQL5.7 安装参考：03-配套软件\MySQL 5.7.28 安装与卸载教程.pdf

2.3 Maven 3.9 配置

Maven 软件位于：03-配套软件\apache-maven-3.3.9.zip

- 在 Maven 安装目录下的 settings.xml 配置文件中, 添加如下配置:
- 在自己电脑上创建仓库目录(pom.xml中配置依赖会自动下载): D:\javasource\maven-repository

```
1 <!--开始处更改下载依赖的存放路径，以下目录需要已经创建-->
2 <localRepository>D:\javasource\maven-repository</localRepository>
3
4 <!--在 mirrors 标签下 添加阿里云maven私服库-->
5 <mirrors>
6   <mirror>
7     <id>alimaven</id>
8     <mirrorOf>central</mirrorOf>
9     <name>aliyun maven</name>
10    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
11  </mirror>
12 </mirrors>
13
14 <!-- 在 profiles 标签下指定jdk版本 -->
15 <profile>
16   <id>jdk-1.8</id>
17   <activation>
18     <activeByDefault>true</activeByDefault>
19   </activation>
20   <jdk>1.8</jdk>
```

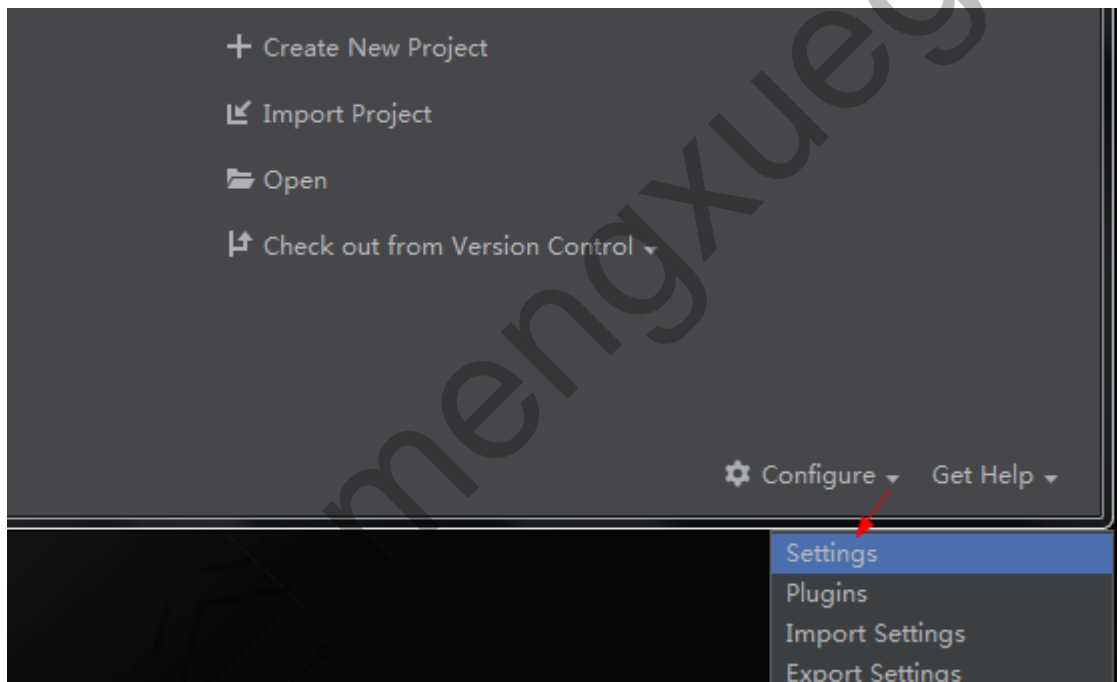
```
20 </activation>
21 <properties>
22   <maven.compiler.source>1.8</maven.compiler.source>
23   <maven.compiler.target>1.8</maven.compiler.target>
24   <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
25 </properties>
26 </profile>
```

2.4 IntelliJ IDEA设置

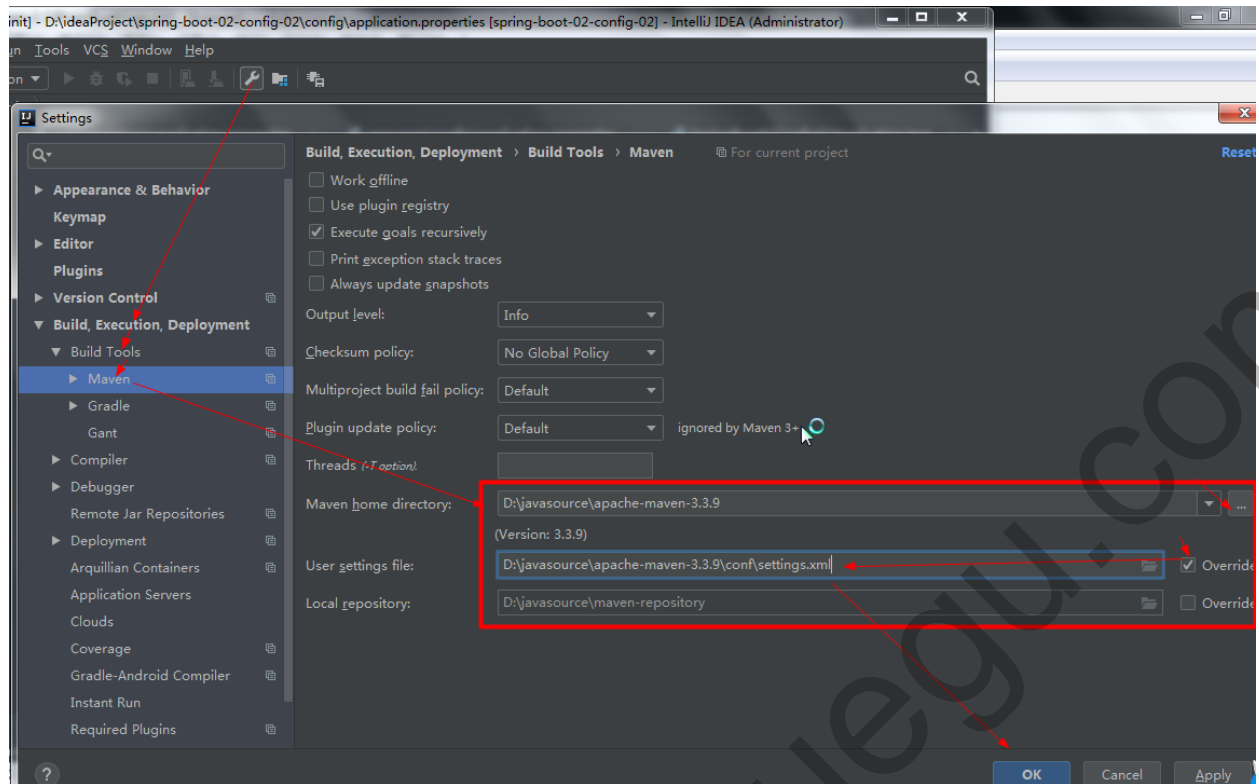
安装很简单,可百度自行安装

2.4.1 IDEA 添加 maven 环境

1. 在idea上将 maven 环境添加进来, 先打开 Settings (Ctrl+Alt+S 打开 Settings 窗口)



2. Build, Execution, Deployment > Build Tools > Maven



2.4.2 加快项目创建

每次创建项目时，IDEA 要使用插件进行创建，这些插件当你创建新的项目时，它每次都会去中央仓库下载，这样使得创建比较慢。应该创建时，让它找本地仓库中的插件进行创建项目。

解决方式：

在 IDEA 的 Settings 窗口的 Build, Execution, Deployment > Build Tools > Maven > Runner 中对 VM Option 设置为 `-DarchetypeCatalog=internal`，如下图：

1566789093279

2.4.3 常用快捷键

ctrl+alt+b 当前接口的实现类有哪些

ctrl+h 打开当前类的实现类窗口

Ctrl+Alt+M 选中代码抽取为一个方法

Ctrl+单击方法或类，进入到父类中

Ctrl+Alt +单击方法或类，进入到子类中

打开Run Dashboard 集中管理运行的应用：View > Tool Windows >Run Dashboard

双击 Shift 键，框中直接搜你想搜的类或者方法

搜索本项目中的方法或者配置信息中的内容 CTRL + Shift + F

Ctrl+N 输入要搜索的类，想搜索的类包括在jar里面，需要勾选“include non-project items”选项，就可以搜索出来

1570676315646

2.5 AdminLTE 介绍与下载

采用 AdminLTE 来完成页面的统一权限管理系统的布局与模板 页面。

2.5.1 AdminLTE 介绍

AdminLTE是一款建立在 bootstrap 和 jquery 之上的开源前端模板，它提供了一系列响应的、可重复使用的组件，并内置了多个模板页面；同时自适应多种屏幕分辨率，兼容PC和移动端。

通过AdminLTE，我们可以快速的创建一个响应式的Html5网站。

AdminLTE框架在网页架构与设计上，有很大的辅助作用，尤其是前端架构设计师，用好 AdminLTE 不但美观，而且可以免去写很大CSS与JS的工作量。

2.5.2 GitHub 下载 AdminLTE

从 Github 下载AdminLTE源代码 <https://github.com/ColorlibHQ/AdminLTE>

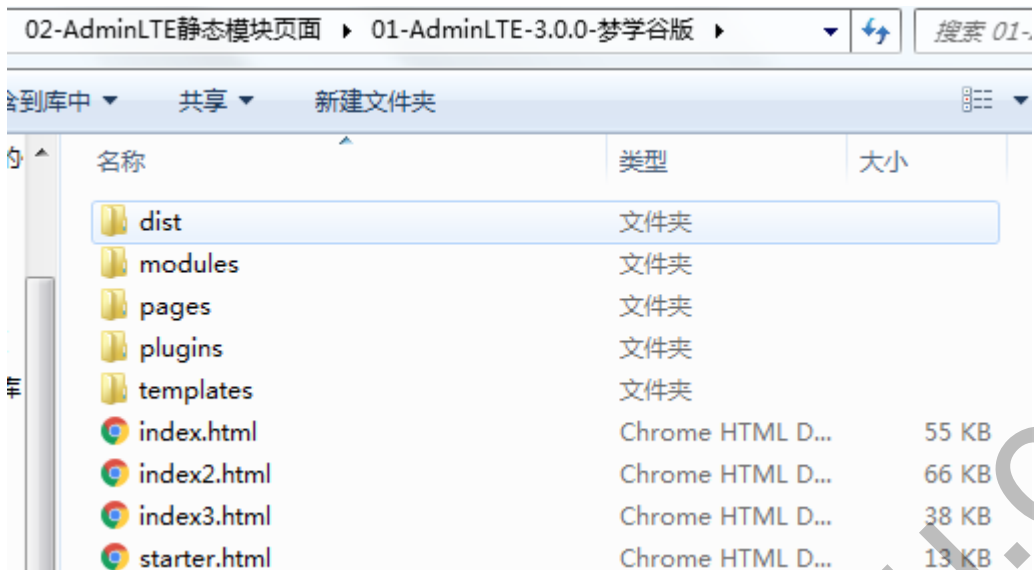
官方指南：<https://adminlte.io/docs/3.0/>



2.5.3 结构介绍

本套课程采用梦学谷改造后的版本:

位于：02-配套资料\02-AdminLTE静态模块页面\01-AdminLTE-3.0.0-梦学谷版



- 1 dist 官方提供的css/img/js...
- 2 modules 梦学谷项目涉及的静态资源，css/img/js...
- 3 pages 官方提供的模板页
- 4 plugins 官方提供的第3方前端插件
- 5 templates 梦学谷项目涉及的模板页面

2.5.4 布局

布局包括四个主要部分：

- 整个页面 .wrapper。包含整个网站的div。
- 主标题 .main-header。包含Logo和导航栏。
- 侧边栏 .sidebar-wrapper。包含用户面板和侧栏菜单。
- 内容 .content-wrapper。包含页面标题和内容。
- 底部 .main-footer。包含版权信息。

2.5.5 图标 icon

参考：<http://fontawesome.dashgame.com/>

1. 将以下代码粘贴到网页HTML代码的 `<head>` 部分

```
1 <link rel="stylesheet" href="https://netdna.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
```

2. 您可以将Font Awesome图标使用在几乎任何地方，只需要使用CSS前缀 `fa`，再加上图标名称。Font Awesome是为使用内联元素而设计的。我们通常更喜欢使用 `<i>`，因为它更简洁。但实际上使用 `` 才能更加语义化。

```
1 <i class="fa fa-weixin"></i> 效果
```

第三章 构建 SpringSecurity 模块化工程

本章节代码备份在 01-课堂源码\01-第1章到第3章节代码 中, 查阅请求认准对应目录下的代码。

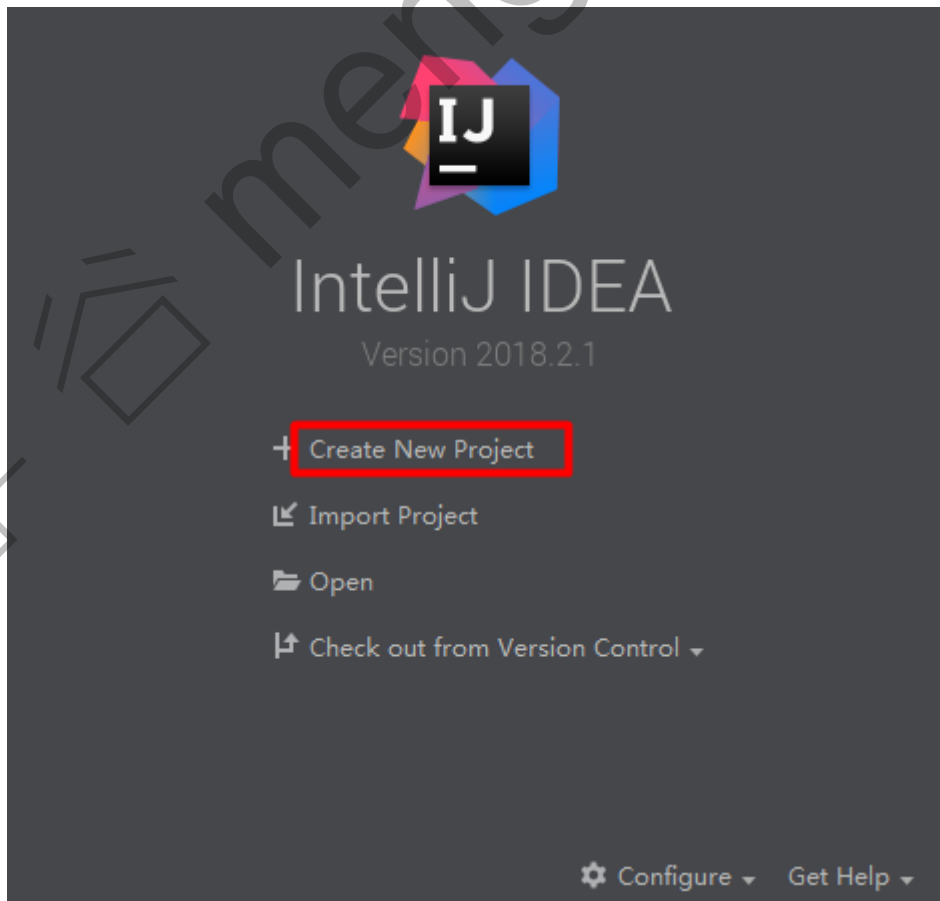
3.1 项目结构

Maven 多模块化构建, 课程共 4 个工程

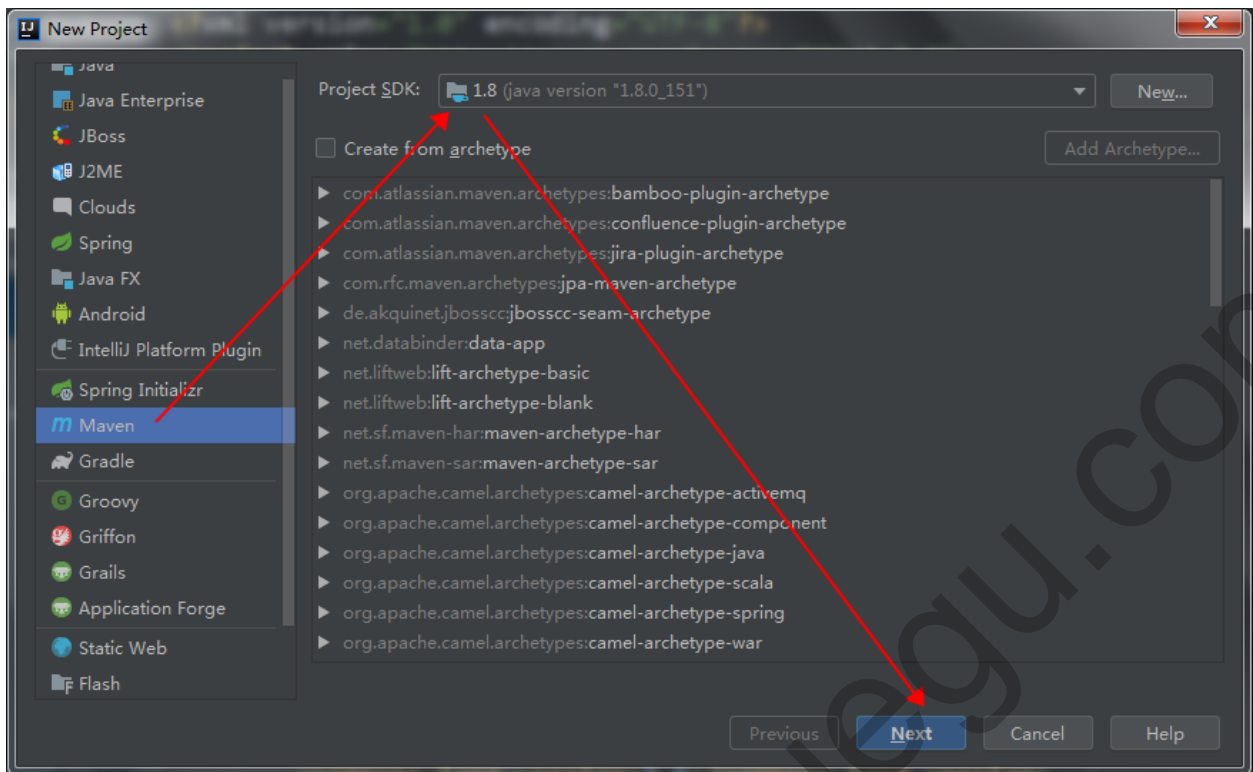
模块名	说明
mengxuegu-security-parent	父模块, pom 类型, 进行统一的版本管理, 聚合管理子模块。
mengxuegu-security-base	基础通用功能管理, 如工具类
mengxuegu-security-core	进行安全管理, 实现身份认证、验证码认证、手机登录、用户授权等
mengxuegu-security-web	Web 业务应用, thymeleaf dao service controller

3.2 创建 mengxuegu-security-parent

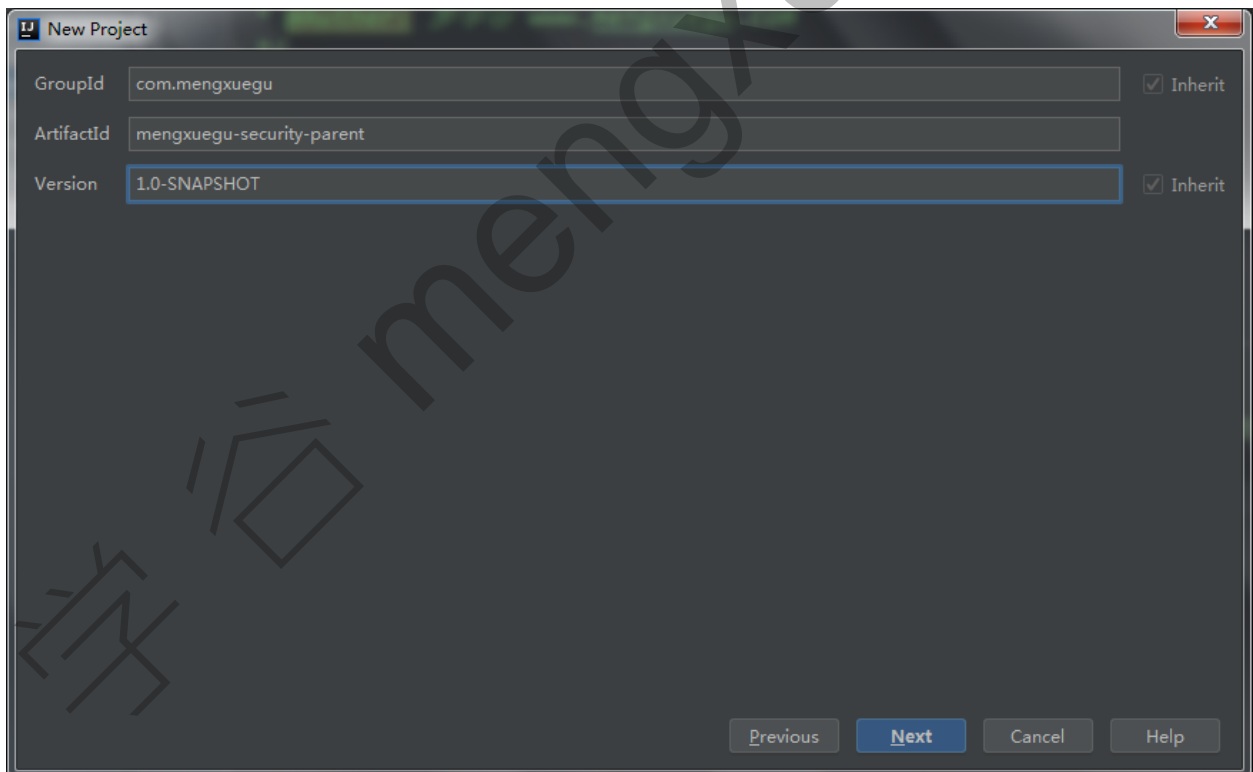
1. 点击 Create New Project



2. 选择 Maven, 选择本地安装的JDK, 点击 Next

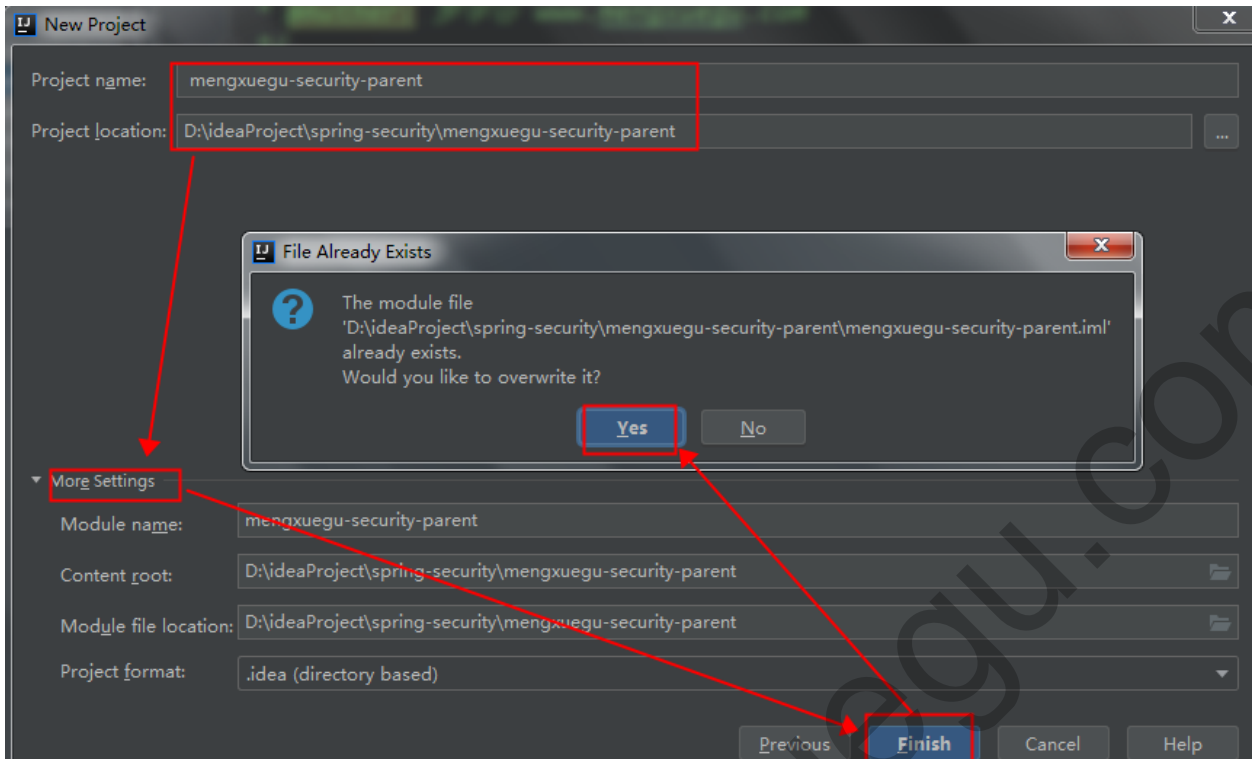


3. 输入GroupID: `com.mengxuegu`、ArtifactID: `mengxuegu-security-parent`

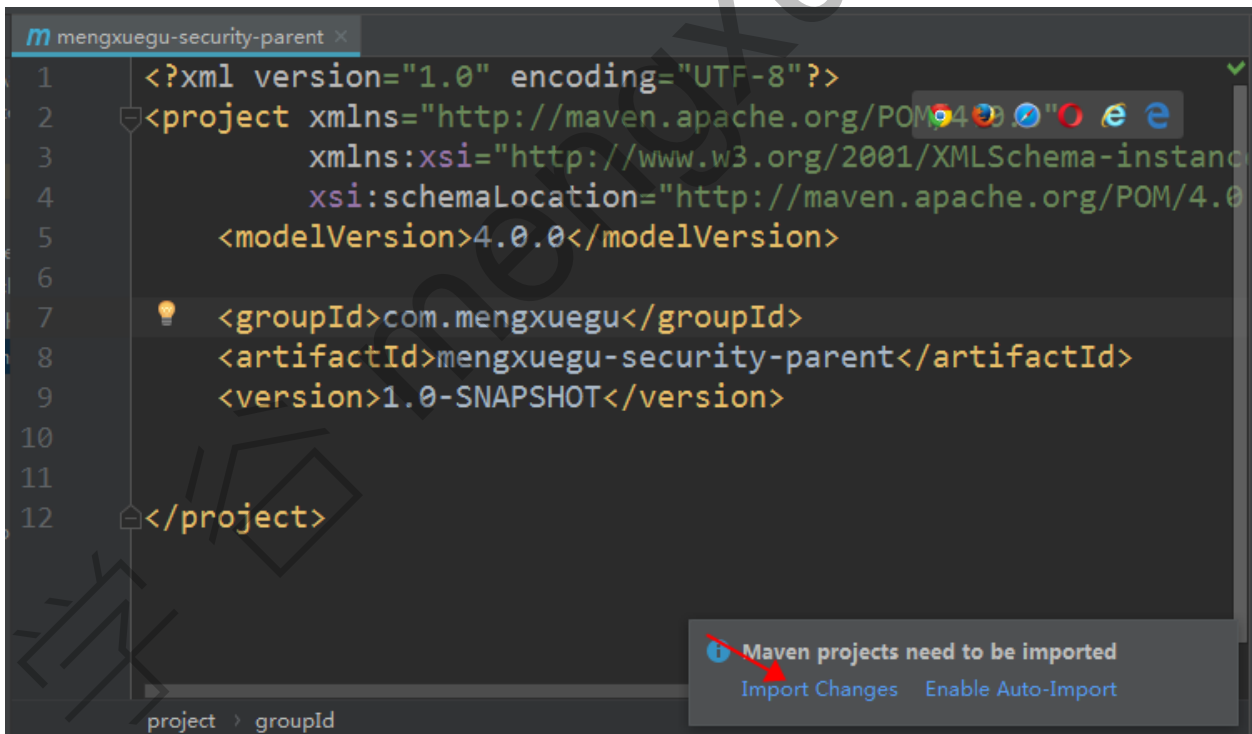


4. 指定Project name: `mengxuegu-security-parent`,

Project location: `D:\ideaProject\spring-security\mengxuegu-security-parent`



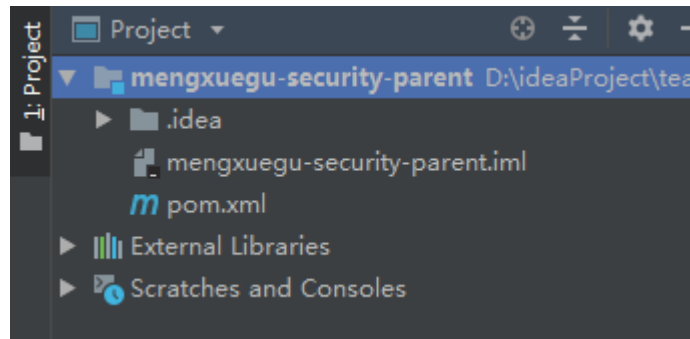
点击 Import Changes 加载



父工程进行统一的管理依赖版本号，没有java代码，是 pom 类型

3.2.1 删除 src 目录

将 mengxuegu-security-parent 下的 src 目录删除（delete键删除）。



3.2.2 修改打包方式为 pom

在 pom.xml 将打包方式指定为 `pom` 类型

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.mengxuegu</groupId>
8   <artifactId>mengxuegu-security-parent</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <!--修改打包方式-->
11  <packaging>pom</packaging>
12
13
14 </project>
```

3.2.3 依赖管理 pom.xml

统一管理Maven依赖的版本，pom.xml 配置如下：

配置内容不要复制下面的，复制网盘资料中的，位于：`02-配套资料\01-pom文件\01-parent-pom.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.mengxuegu</groupId>
8   <artifactId>mengxuegu-security-parent</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <!--修改打包方式-->
11  <packaging>pom</packaging>
12
13  <!-- Spring Boot 父工程-->
```

```
14 <parent>
15 <groupId>org.springframework.boot</groupId>
16 <artifactId>spring-boot-starter-parent</artifactId>
17 <version>2.2.0.RELEASE</version>
18 <relativePath/>
19 </parent>
20
21 <!-- 依赖版本号 -->
22 <properties>
23 <mybatis-plus.version>3.2.0</mybatis-plus.version>
24 <druid.version>1.1.12</druid.version>
25 <oauth2-autoconfigure.version>2.1.3.RELEASE</oauth2-autoconfigure.version>
26 <kaptcha.version>2.3.2</kaptcha.version>
27 <fastjson.version>1.2.8</fastjson.version>
28 <commons-lang.version>2.6</commons-lang.version>
29 <commons-collections.version>3.2.2</commons-collections.version>
30 <commons-io.version>2.6</commons-io.version>
31 <!-- 定义版本号, 子模块直接引用-->
32 <mengxuegu-security.version>1.0-SNAPSHOT</mengxuegu-security.version>
33 </properties>
34
35 <!-- 集中式管理依赖版本号,并没有真实依赖 -->
36 <dependencyManagement>
37 <dependencies>
38 <!-- mybatis-plus 启动器 -->
39 <dependency>
40 <groupId>com.baomidou</groupId>
41 <artifactId>mybatis-plus-boot-starter</artifactId>
42 <version>${mybatis-plus.version}</version>
43 </dependency>
44 <!-- druid 连接池 -->
45 <dependency>
46 <groupId>com.alibaba</groupId>
47 <artifactId>druid</artifactId>
48 <version>${druid.version}</version>
49 </dependency>
50
51 <!-- spring-security-oauth2、spring-security-jwt等 -->
52 <dependency>
53 <groupId>org.springframework.security.oauth.boot</groupId>
54 <artifactId>spring-security-oauth2-autoconfigure</artifactId>
55 <version>${oauth2-autoconfigure.version}</version>
56 </dependency>
57 <!-- kaptcha 用于图形验证码 -->
58 <dependency>
59 <groupId>com.github.penggle</groupId>
60 <artifactId>kaptcha</artifactId>
61 <version>${kaptcha.version}</version>
62 </dependency>
63
64 <!-- 工具类依赖 -->
65 <dependency>
66 <groupId>com.alibaba</groupId>
```

```
67     <artifactId>fastjson</artifactId>
68     <version>${fastjson.version}</version>
69 </dependency>
70
71 <dependency>
72     <groupId>commons-lang</groupId>
73     <artifactId>commons-lang</artifactId>
74     <version>${commons-lang.version}</version>
75 </dependency>
76 <dependency>
77     <groupId>commons-collections</groupId>
78     <artifactId>commons-collections</artifactId>
79     <version>${commons-collections.version}</version>
80 </dependency>
81 <dependency>
82     <groupId>commons-io</groupId>
83     <artifactId>commons-io</artifactId>
84     <version>${commons-io.version}</version>
85 </dependency>
86 </dependencies>
87 </dependencyManagement>
88
89 <!--聚合管理-->
90 <modules>
91 </modules>
92
93 <build>
94     <plugins>
95         <plugin>
96             <groupId>org.apache.maven.plugins</groupId>
97             <artifactId>maven-compiler-plugin</artifactId>
98             <version>3.7.0</version>
99             <configuration>
100                 <source>1.8</source>
101                 <target>1.8</target>
102                 <encoding>UTF-8</encoding>
103             </configuration>
104         </plugin>
105     </plugins>
106 </build>
107
108 </project>
```

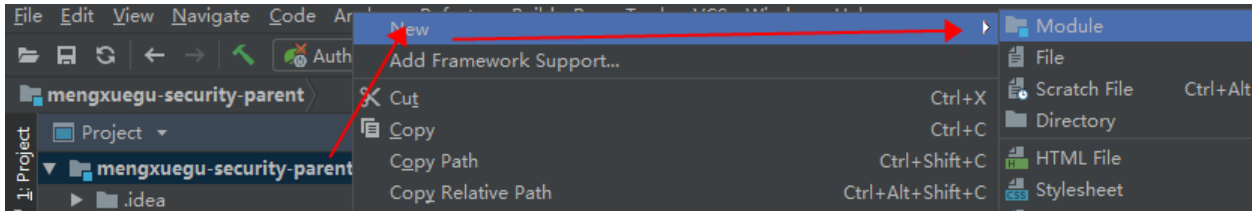
3.3 创建 mengxuegu-security-base

3.3.1 概述

此模块作为基础通用功能管理，直接提供给其他服务引入jar依赖即可使用。

3.3.2 新建 Module

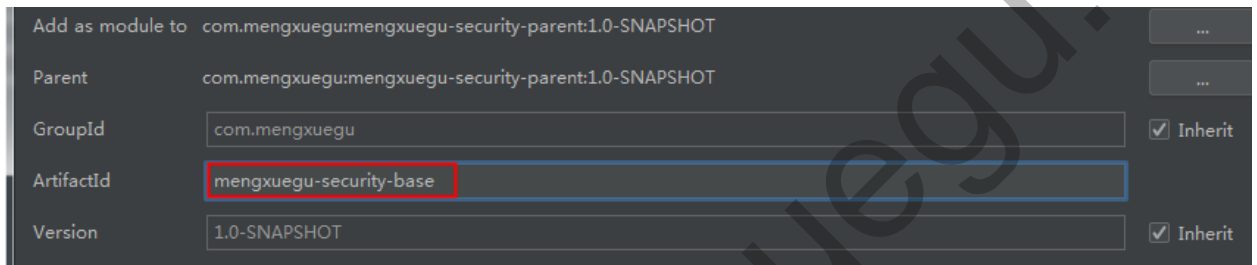
1. 右击 **mengxuegu-security-parent** 模块名，点击 New > Module



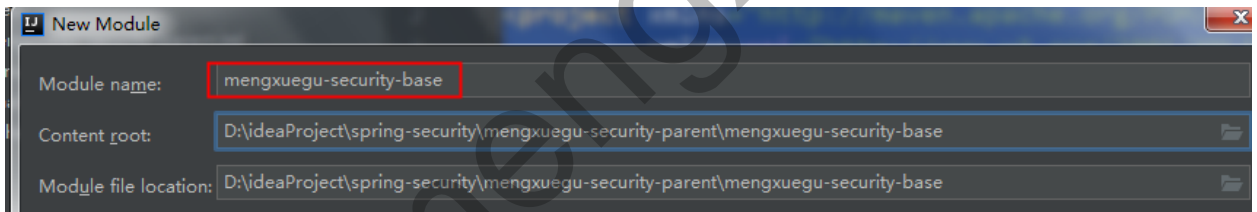
2. 点击左侧 Maven，直接 Next

1568800318597

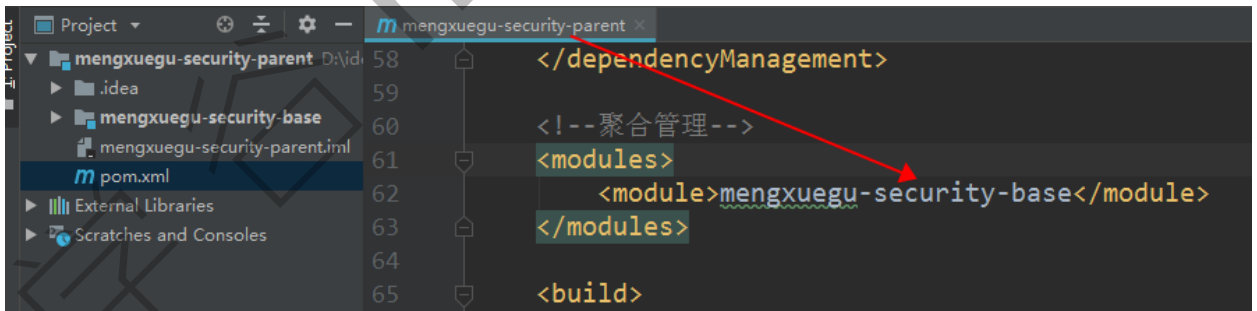
3. ArtifactID 指定为 **mengxuegu-security-base**，注意Parent是默认有的，就是第1步右击的 paren工程名



4. 将 Module name 改为 **mengxuegu-security-base**，并核对保存位置



5. 观察 mengxuegu-security-parent 工程下 modules 标签中有 **mengxuegu-security-base** 子模块



3.3.3 依赖管理 pom.xml

目前引入工具类相关依赖

配置内容不要复制下面的，复制网盘资料中的，位于：02-配套资料\01-pom文件\02-base-pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
      4.0.0.xsd">
5      <parent>
6          <artifactId>mengxuegu-security-parent</artifactId>
7          <groupId>com.mengxuegu</groupId>
8          <version>1.0-SNAPSHOT</version>
9          <relativePath>../pom.xml</relativePath>
10     </parent>
11     <modelVersion>4.0.0</modelVersion>
12
13     <artifactId>mengxuegu-security-base</artifactId>
14
15     <dependencies>
16         <!--json处理工具-->
17         <dependency>
18             <groupId>com.alibaba</groupId>
19             <artifactId>fastjson</artifactId>
20         </dependency>
21         <!--注解方式简化setter/getter/constructor等-->
22         <dependency>
23             <groupId>org.projectlombok</groupId>
24             <artifactId>lombok</artifactId>
25         </dependency>
26         <!--真实依赖 工具类-->
27         <dependency>
28             <groupId>commons-lang</groupId>
29             <artifactId>commons-lang</artifactId>
30         </dependency>
31         <dependency>
32             <groupId>commons-collections</groupId>
33             <artifactId>commons-collections</artifactId>
34         </dependency>
35         <dependency>
36             <groupId>commons-io</groupId>
37             <artifactId>commons-io</artifactId>
38         </dependency>
39     </dependencies>
40 </project>
```

3.3.4 添加 Logback 日志文件

1. 在 `src/main\` 下创建 `resources` 目录存放配置文件

右击目录名 > Mark Directory as > Resources Root, (如果有此目录, 则忽略此操作)

1568600713742

2. 将 02-配套资料 下的 `logback.xml` 复制到 `resources` 目录下

1570761451099

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--梦学谷 www.mengxuegu.com -->
```

```
3 <configuration>
4   <!-- 彩色日志 -->
5   <!-- 彩色日志依赖的渲染类 -->
6   <conversionRule conversionWord="clr"
7     converterClass="org.springframework.boot.logging.logback.ColorConverter" />
8   <conversionRule conversionWord="wex"
9     converterClass="org.springframework.boot.logging.logback.WhitespaceThrowableProxyConverter" />
10  <conversionRule conversionWord="wEx"
11    converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrowableProxyConverter" />
12  <!-- 彩色日志格式 -->
13  <property name="CONSOLE_LOG_PATTERN" value="${CONSOLE_LOG_PATTERN:-%clr(%d{HH:mm:ss.SSS})
14    {faint} %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint}
15    %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}}"/>
16  <!-- ch.qos.logback.core.ConsoleAppender 表示控制台输出 -->
17  <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
18    <layout class="ch.qos.logback.classic.PatternLayout">
19      <pattern>${CONSOLE_LOG_PATTERN}</pattern>
20    </layout>
21  </appender>
22
23  <root level="info">
24    <appender-ref ref="stdout" />
25  </root>
26 </configuration>
```

3.4 创建 mengxuegu-security-core

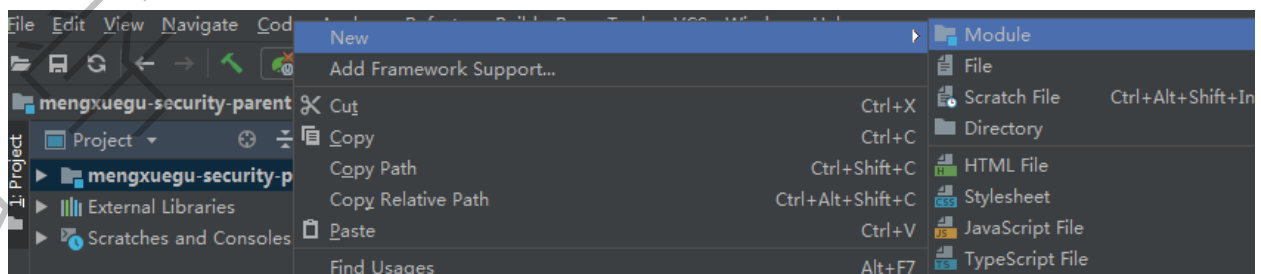
3.4.1 概述

此模块用于进行安全管理，作为一个安全 jar 包，直接提供给其他服务引入jar依赖即可使用。

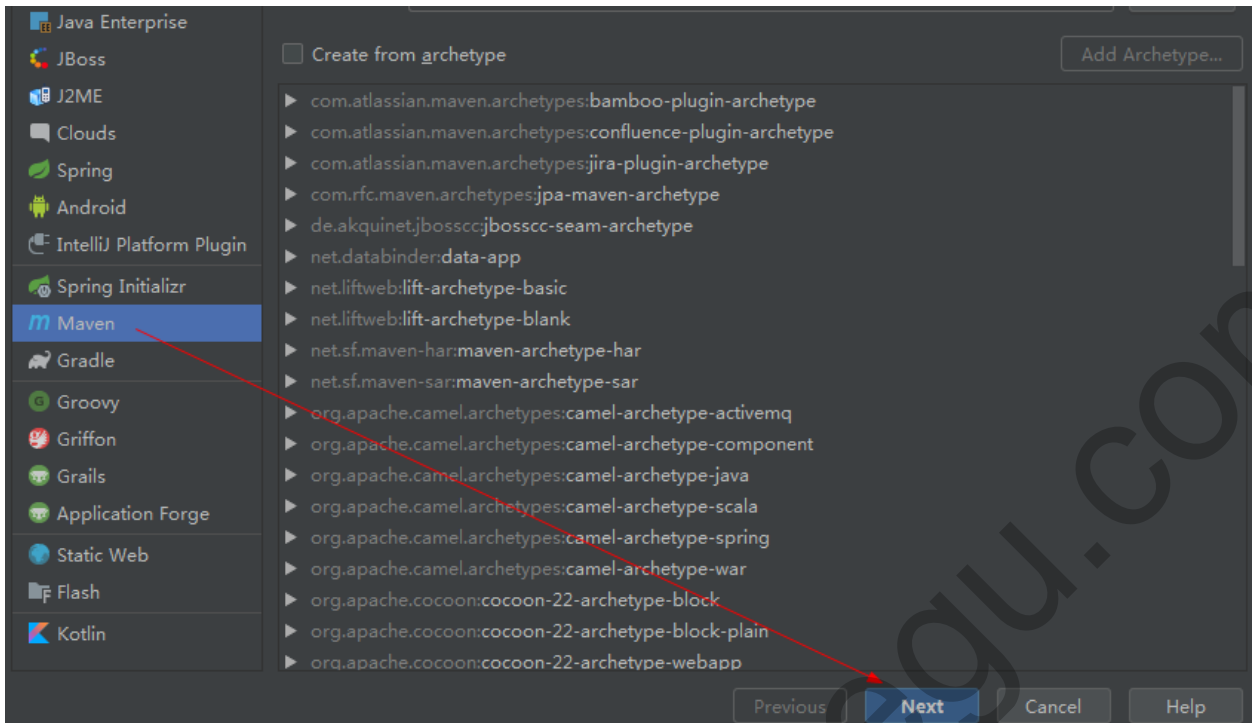
实现用户身份验证和用户授权功能

3.4.2 新建 Module

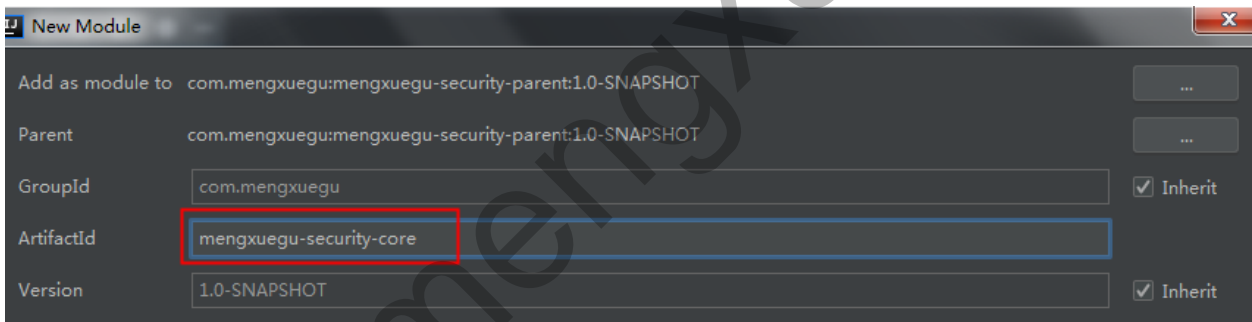
1. 右击 **mengxuegu-security-parent** 模块名，点击 New > Module



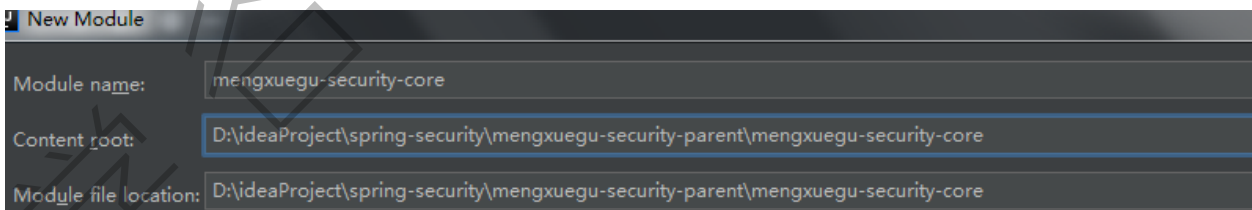
2. 点击左侧 Maven，直接 Next



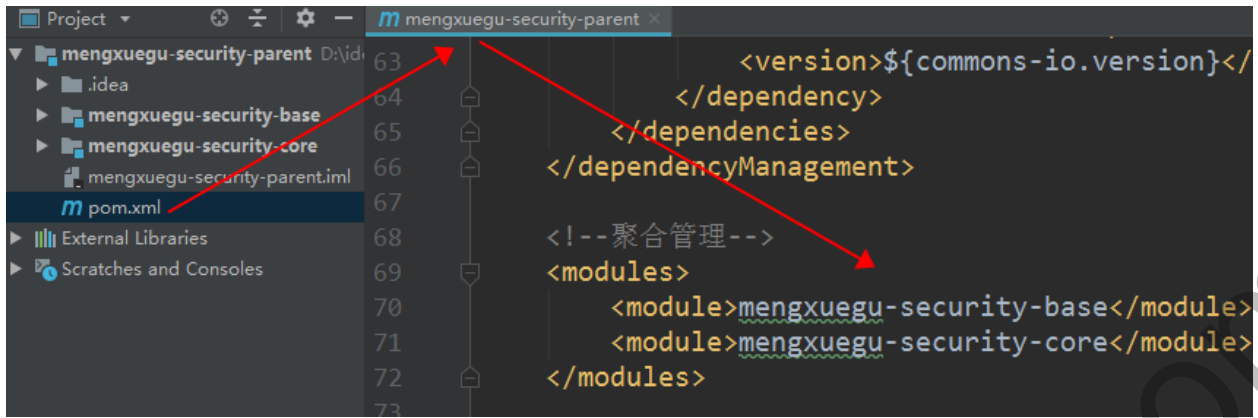
3. ArtifactID 指定为 `mengxuegu-security-core`，注意Parent是默认有的，就是第1步右击的paren工程名



4. 将 Module name 改为 `mengxuegu-security-core`，并核对保存位置



5. 观察 `mengxuegu-security-parent` 工程下 modules 标签中有 `mengxuegu-security-core` 子模块



3.4.3 依赖管理 pom.xml

核心引入 `spring-boot-starter-security` 启动器

配置内容不要复制下面的，复制网盘资料中的，位于：`02-配套资料\01-pom文件\03-core-pom.xml`

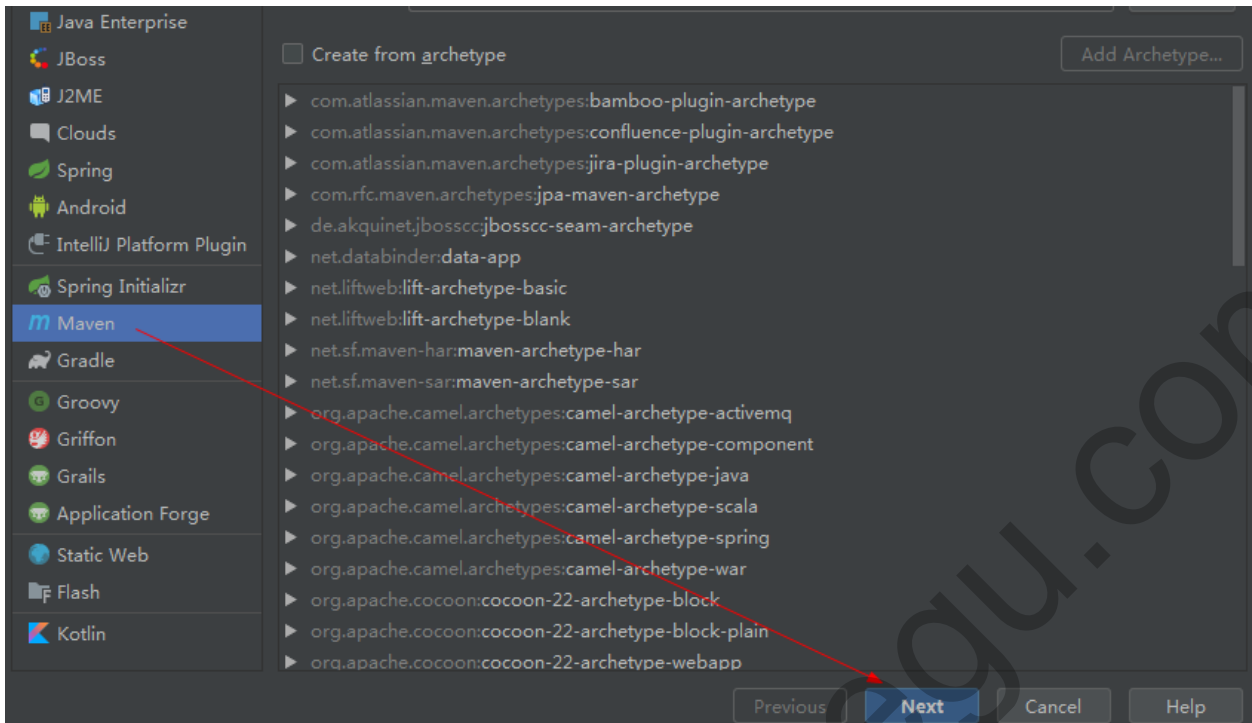
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5     4.0.0.xsd">
6   <parent>
7     <artifactId>mengxuegu-security-parent</artifactId>
8     <groupId>com.mengxuegu</groupId>
9     <version>1.0-SNAPSHOT</version>
10   </parent>
11   <modelVersion>4.0.0</modelVersion>
12   <artifactId>mengxuegu-security-core</artifactId>
13
14   <dependencies>
15     <!-- 依赖mengxuegu-security-base基础模块 -->
16     <dependency>
17       <groupId>com.mengxuegu</groupId>
18       <artifactId>mengxuegu-security-base</artifactId>
19       <version>${mengxuegu-security.version}</version>
20     </dependency>
21
22     <!-- spring security 启动器 -->
23     <dependency>
24       <groupId>org.springframework.boot</groupId>
25       <artifactId>spring-boot-starter-security</artifactId>
26     </dependency>
27     <dependency>
28       <groupId>javax.servlet</groupId>
29       <artifactId>servlet-api</artifactId>
30       <version>2.5</version>
31     </dependency>
32   </dependencies>
```

```
33
34     <!--图形验证码
35     <dependency>
36         <groupId>com.github.penggle</groupId>
37         <artifactId>kaptcha</artifactId>
38     </dependency>
39     -->
40     <!--数据库依赖
41     <dependency>
42         <groupId>org.springframework.boot</groupId>
43         <artifactId>spring-boot-starter-jdbc</artifactId>
44     </dependency>
45     <dependency>
46         <groupId>mysql</groupId>
47         <artifactId>mysql-connector-java</artifactId>
48     </dependency>
49     -->
50     <!--采用redis来管理session
51     <dependency>
52         <groupId>org.springframework.boot</groupId>
53         <artifactId>spring-boot-starter-data-redis</artifactId>
54     </dependency>
55     <dependency>
56         <groupId>org.springframework.session</groupId>
57         <artifactId>spring-session-data-redis</artifactId>
58     </dependency>
59     -->
60
61     <!--解决找不到 javax.annotation.meta.When.MAYBE
62     <dependency>
63         <groupId>com.google.code.findbugs</groupId>
64         <artifactId>annotations</artifactId>
65         <version>3.0.1</version>
66     </dependency>
67     -->
68 </dependencies>
69 </project>
```

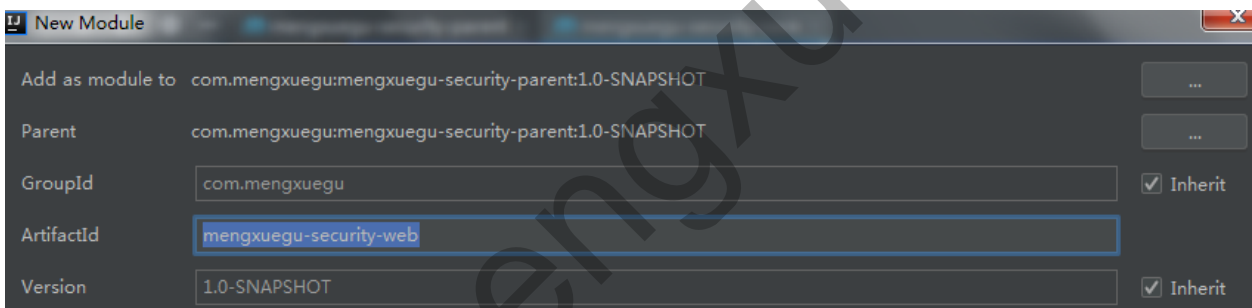
3.5 创建 mengxuegu-security-web

3.5.1 新建 Module

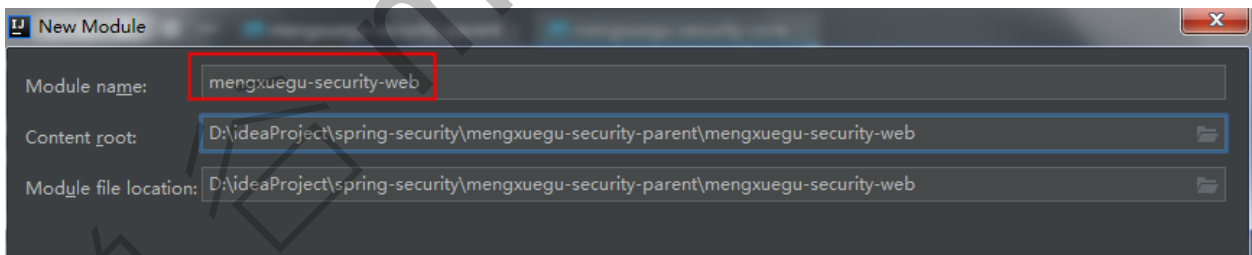
1. 右击 **mengxuegu-security-parent** 模块名，点击 New > Module
2. 点击左侧 Maven，直接 Next



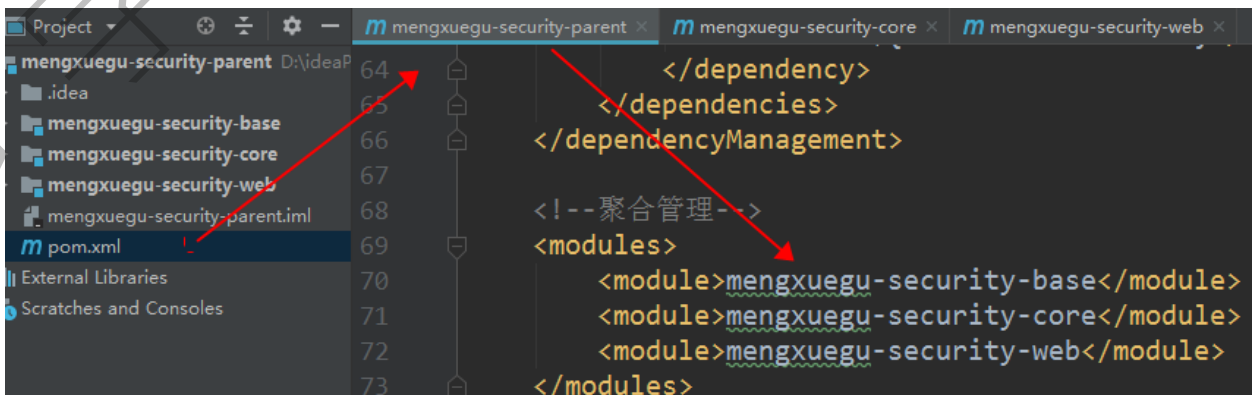
3. ArtifactID 指定为 `mengxuegu-security-web`，注意Parent 是默认有的，就是第1步右击的paren工程



4. 将 Module name 改为 `mengxuegu-security-web`，并核对保存位置



5. 观察 `mengxuegu-security-parent` 工程下 modules 标签中有 `mengxuegu-security-web` 子模块



3.5.2 依赖管理 pom.xml

在 mengxuegu-security-web 工程中的 pom.xml 添加依赖

配置内容不要复制下面的，复制网盘资料中的，位于：02-配套资料\01-pom文件\03-web-pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5         4.0.0.xsd">
6     <parent>
7         <artifactId>mengxuegu-security-parent</artifactId>
8         <groupId>com.mengxuegu</groupId>
9         <version>1.0-SNAPSHOT</version>
10        <relativePath>../pom.xml</relativePath>
11    </parent>
12    <modelVersion>4.0.0</modelVersion>
13    <artifactId>mengxuegu-security-web</artifactId>
14
15    <dependencies>
16        <!-- web启动器, 对springmvc, servlet等支持 -->
17        <dependency>
18            <groupId>org.springframework.boot</groupId>
19            <artifactId>spring-boot-starter-web</artifactId>
20        </dependency>
21        <!-- 权限核心模块, 注意：要放到 web启动器下面-->
22        <dependency>
23            <groupId>com.mengxuegu</groupId>
24            <artifactId>mengxuegu-security-core</artifactId>
25            <version>${mengxuegu-security.version}</version>
26        </dependency>
27        <!-- thymeleaf 模块启动器-->
28        <dependency>
29            <groupId>org.springframework.boot</groupId>
30            <artifactId>spring-boot-starter-thymeleaf</artifactId>
31        </dependency>
32
33        <!--对Thymeleaf添加Spring Security标签支持
34        <dependency>
35            <groupId>org.thymeleaf.extras</groupId>
36            <artifactId>thymeleaf-extras-springsecurity5</artifactId>
37        </dependency>
38        -->
39        <!--mybatis-plus启动器
40        <dependency>
41            <groupId>com.baomidou</groupId>
42            <artifactId>mybatis-plus-boot-starter</artifactId>
43        </dependency>
44        -->
45        <!--druid连接池
46        <dependency>
```

```
47     <groupId>com.alibaba</groupId>
48     <artifactId>druid</artifactId>
49 </dependency>
50 -->
51
52 <!-- application.yml 配置处理器-->
53 <dependency>
54     <groupId>org.springframework.boot</groupId>
55     <artifactId>spring-boot-configuration-processor</artifactId>
56     <optional>true</optional>
57 </dependency>
58
59 <!-- springboot 单元测试 -->
60 <dependency>
61     <groupId>org.springframework.boot</groupId>
62     <artifactId>spring-boot-starter-test</artifactId>
63 </dependency>
64 <!-- 热部署 ctrl+f9-->
65 <dependency>
66     <groupId>org.springframework.boot</groupId>
67     <artifactId>spring-boot-devtools</artifactId>
68 </dependency>
69 </dependencies>
70
71 </project>
```

3.5.3 配置 application.yml

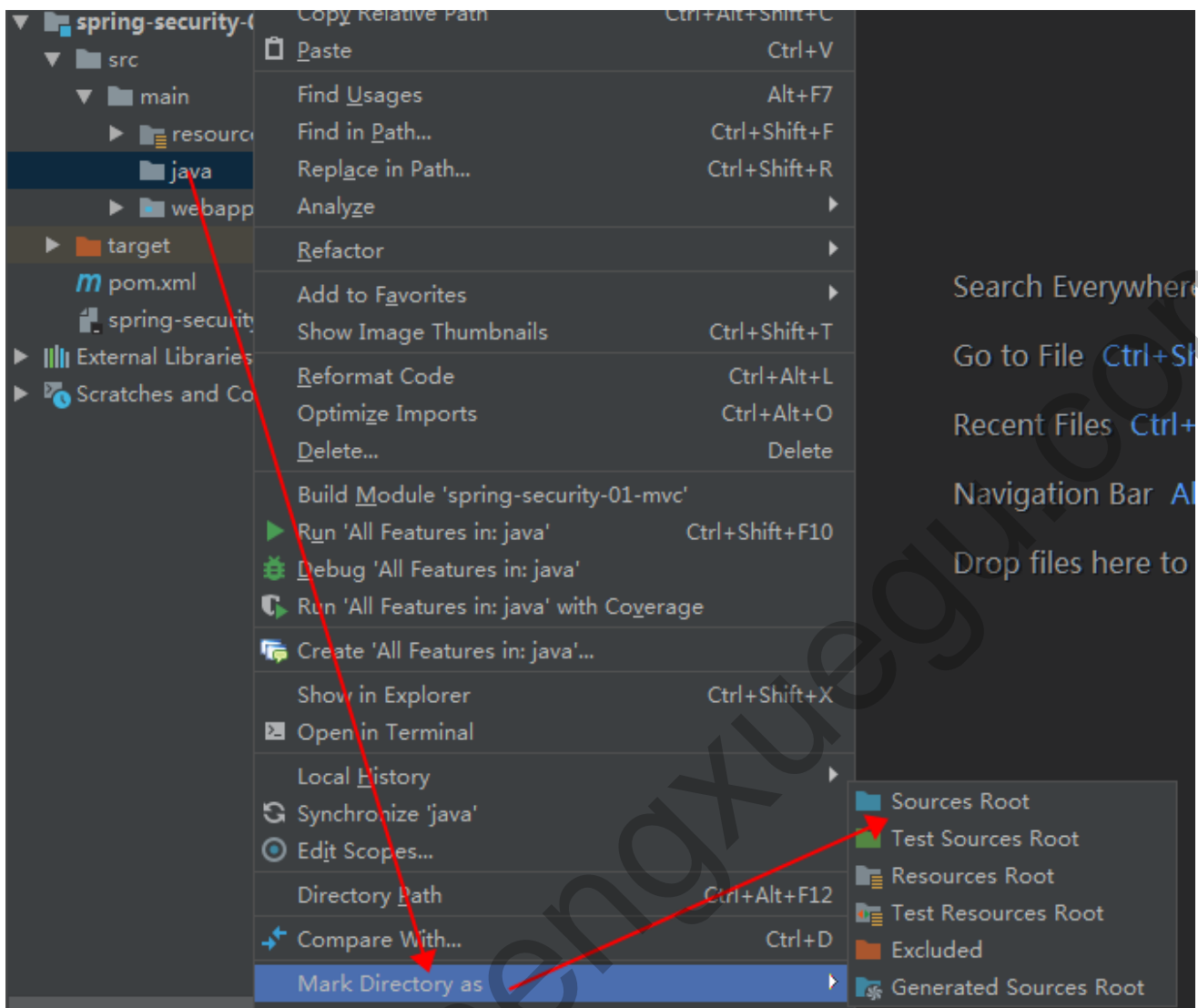
在 `src/main/resources` 下面创建 `application.yml` 文件, 指定端口号 **80**, 后面有作用

注意: 不要复制, 手动输入。冒号后面有一个空格

```
1 server:
2   port: 80
3
4 spring:
5   thymeleaf:
6     cache: false # 关闭 thymeleaf 缓存
```

3.5.4 创建 Controller

1. `src/main` 下创建 `java` 目录, 指定为 `Sources Root` (如果有此目录, 则忽略此操作)



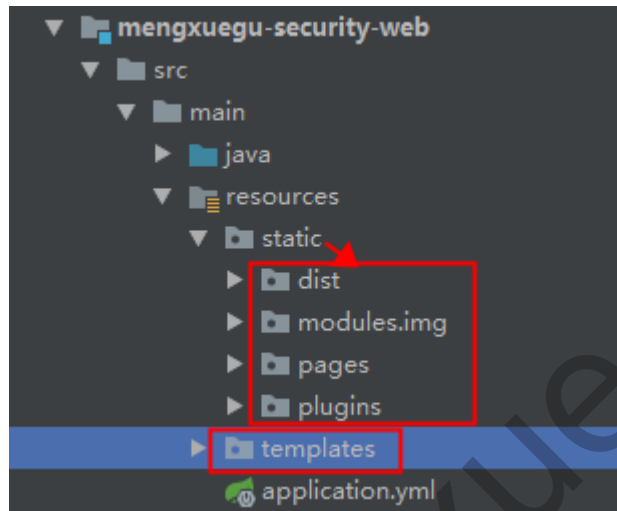
2. 在 `src/main/java` 目录下创建 `com.mengxuegu.web.controller.MainController` 控制类

```
1 package com.mengxuegu.web.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 /**
7  * @Author: 梦学谷 www.mengxuegu.com
8  */
9 @Controller
10 public class MainController {
11
12     // 首页
13     @RequestMapping({"/index", "/", ""})
14     public String index() {
15         return "index";
16     }
17 }
```

3.5.5 AdminLTE 资源添加到项目

1. 在 `resources` 下创建 `static` 目录。
2. 拷贝 02-配套资料\02-AdminLTE静态模块页面\01-AdminLTE-3.0.0-梦学谷版 中 `dist`、`modules`、`plugins` 目录到 `resources/static` 目录下；拷贝 `templates` 目录到 `resources` 目录下。
3. 注意：templates 目录下页面是不能直接通过浏览器访问，需要通过 Controller 跳转指定页面才可访问。

效果如下：



3.5.6 创建启动类

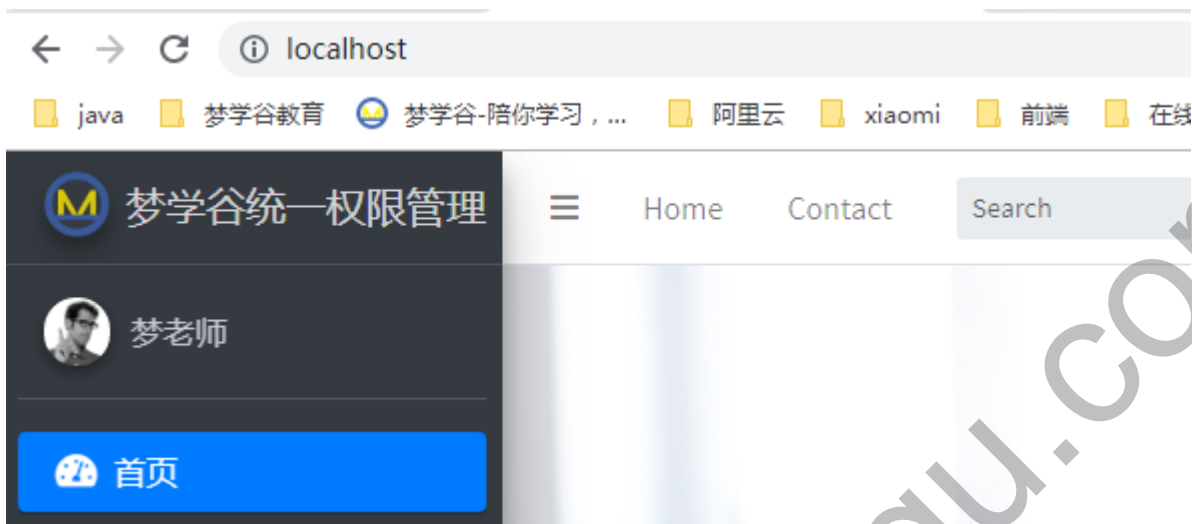
在 `src\main\java` 目录上右击，`New` > `Java Class` 创建类 `com.mengxuegu.WebApplication`

注意：是在 `com.mengxuegu` 下创建启动类，且 `@SpringBootApplication` 注解不要少了

```
1 package com.mengxuegu;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 /**
7  * @Author: 梦学谷 www.mengxuegu.com
8  */
9 @SpringBootApplication
10 public class WebApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(WebApplication.class, args);
14     }
15
16 }
```

3.5.7 测试

1. 运行 WebApplicationin 启动服务
2. 访问：<http://localhost/>（访问的是80端口）



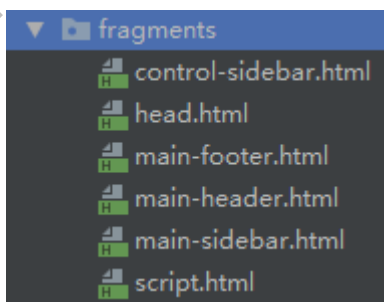
3.6 抽取HTML公共代码

3.6.1 概述

在 AdminLTE 中每个页面有很多相同代码，头部区域、侧边导航、底部区域在每个页面都有，我们可以将它们抽取为公共代码，在其他页面引入即可，从而减少重复代码。

3.6.2 编码流程

1. 在 `templates\fragments` 目录下，存放公共代码片段
 - o html模板中 `th` 红色的, 实际上是不影响的, 有强迫症的可以在第一行加上 `<!--suppress ALL-->`，压制当前文件中 thymeleaf 的警告或者错误



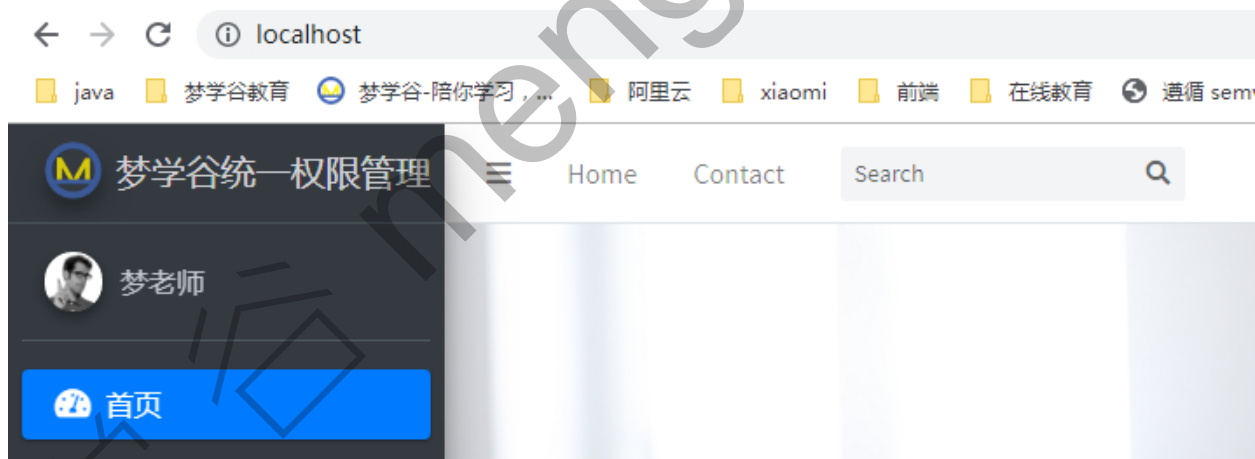
2. 重构 `index.html` 引入公共代码片段

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org"
3     xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
4 <head th:replace="fragments/head">
5 </head>
6
7 <body class="hold-transition sidebar-mini layout-fixed layout-navbar-fixed">
8 <!-- Site wrapper -->
```



```
9 <div class="wrapper">
10 <!-- 头部区域 Navbar -->
11 <th:block th:replace="fragments/main-header"/>
12 <!-- 左侧区域 Main Sidebar Container -->
13 <th:block th:replace="fragments/main-sidebar"/>
14
15 <!-- 右侧主区域 Content Wrapper. Contains page content -->
16 <div class="content-wrapper">
17 
18 </div>
19 <!-- /.content-wrapper -->
20
21 <!-- 右底部区域 -->
22 <th:block th:replace="fragments/main-footer"/>
23 <!-- 右上角工具栏 Control Sidebar -->
24 <th:block th:replace="fragments/control-sidebar"/>
25 </div>
26 <!-- ./wrapper -->
27
28 <!-- js公共代码 -->
29 <th:block th:replace="fragments/script"/>
30 </body>
31 </html>
```

3. 重启项目，访问 <http://localhost/>



第四章 Spring Security 身份认证方式

4.1 HttpBasic 认证方式

4.1.1 添加 Spring Security 启动器

在 mengxuegu-security-core\pom.xml 中添加 spring-boot-starter-security 依赖，如下：

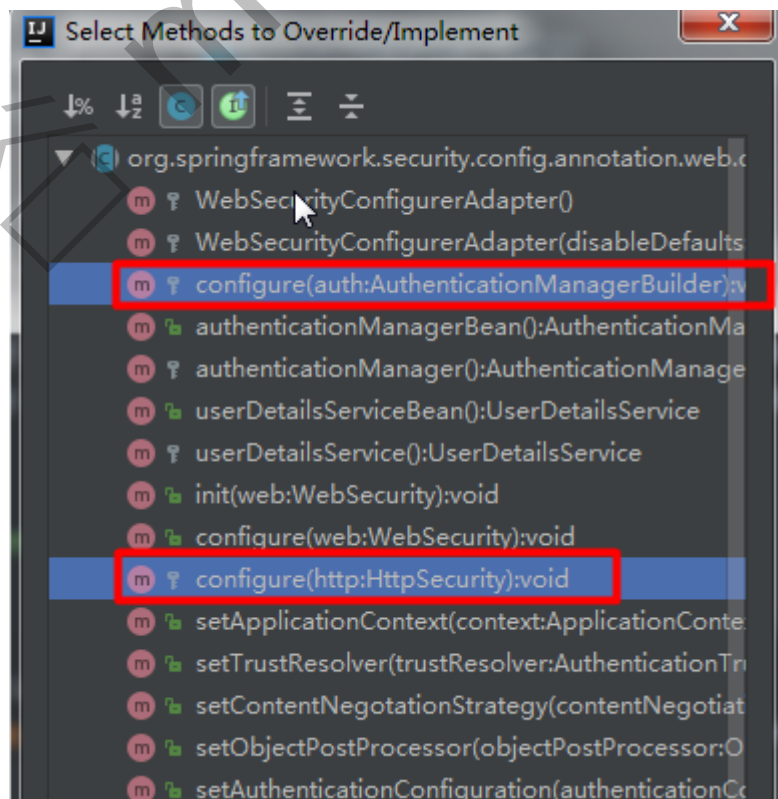
```
1 <!-- spring security 启动器 -->
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-security</artifactId>
5 </dependency>
```

4.1.2 编写 SpringSecurityConfig 安全配置类

SpringSecurityConfig 安全控制配置类作为**安全控制中心**，用于实现身份认证与授权配置功能

步骤：

1. 在 mengxuegu-security-core 中创建 com.mengxuegu.security.config.SpringSecurityConfig 类，继承 WebSecurityConfigurerAdapter 抽象类
2. 类上添加注解 @Configuration 标识为配置类、@EnableWebSecurity 启动 SpringSecurity 过滤器链功能
3. 重写以下两个方法：
 - configure(AuthenticationManagerBuilder auth) 身份认证管理器
 - 认证信息提供方式（用户名、密码、当前用户的资源权限）
 - 可采用内存存储方式，也可能采用数据库方式等
 - configure(HttpSecurity http) 资源权限配置（过滤器链）
 - 拦截的哪一些资源
 - 资源所对应的角色权限
 - 定义认证方式：httpBasic、httpForm
 - 定制登录页面、登录请求地址、错误处理方式
 - 自定义 spring security 过滤器等

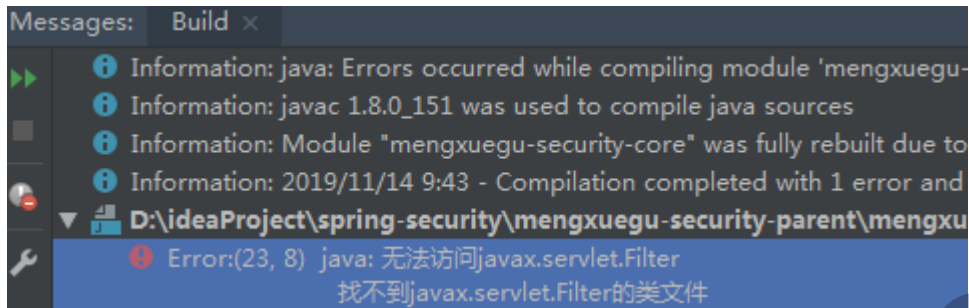


4. 效果如下

```
1 package com.mengxuegu.security.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import
5     org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
8 import
9     org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
10
11 /**
12  * 安全控制中心
13  * @Author: 梦学谷 www.mengxuegu.com
14  */
15 @Configuration
16 @EnableWebSecurity //启动 SpringSecurity 过滤器链功能
17 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
18
19     /**
20      * 认证管理器：
21      * 1、认证信息提供方式（用户名、密码、当前用户的资源权限）
22      * 2、可采用内存存储方式，也可能采用数据库方式等
23      * @param auth
24      * @throws Exception
25      */
26     @Override
27     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
28         super.configure(auth);
29     }
30
31     /**
32      * 资源权限配置（过滤器链）：
33      * 1、被拦截的资源
34      * 2、资源所对应的角色权限
35      * 3、定义认证方式：httpBasic、httpForm
36      * 4、定制登录页面、登录请求地址、错误处理方式
37      * 5、自定义 spring security 过滤器
38      * @param http
39      * @throws Exception
40      */
41     @Override
42     protected void configure(HttpSecurity http) throws Exception {
43         http.httpBasic()
44             .and()
45             .authorizeRequests() // 认证请求
46             .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
47             ; // 分号`; 不要少了
48     }
49 }
```

4.1.3 启动报错

1. 启动项目时报错：无法访问javax.servlet.Filter，找不到javax.servlet.Filter的类文件



解决方式：在 mengxuegu-security-core 模块pom.xml添加如下依赖，点击 import changes，然后把 target 编译目录删除再重启项目

```
1 <!-- spring security 启动器-->
2 <dependency>
3   <groupId>javax.servlet</groupId>
4   <artifactId>servlet-api</artifactId>
5   <version>2.5</version>
6 </dependency>
```

2. 如果重启后还报错（没有报错，那说明你的依赖顺序没有错）

```
java.util.concurrent.ExecutionException: org.apache.catalina.LifecycleException: Failed to start component
    at org.apache.catalina.core.ContainerBase.startInternal(ContainerBase.java:916)
    at org.apache.catalina.core.StandardHost.startInternal(StandardHost.java:841)
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183)
    at org.apache.catalina.core.ContainerBase.startChild.call(ContainerBase.java:1384)
```

解决方式：修改 mengxuegu-security-web 模块 pom.xml 中的 spring-boot-starter-web 依赖移到 mengxuegu-security-core 依赖的上面，点击 import changes，删除所有的 target 编译目录再重启项目

```
1 <!--web启动器,对springmvc,servlet等支持-->
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-web</artifactId>
5 </dependency>
6 <!--注意要放到web启动器的下面-->
7 <dependency>
8   <groupId>com.mengxuegu</groupId>
9   <artifactId>mengxuegu-security-core</artifactId>
10  <version>${mengxuegu-security.version}</version>
11 </dependency>
```

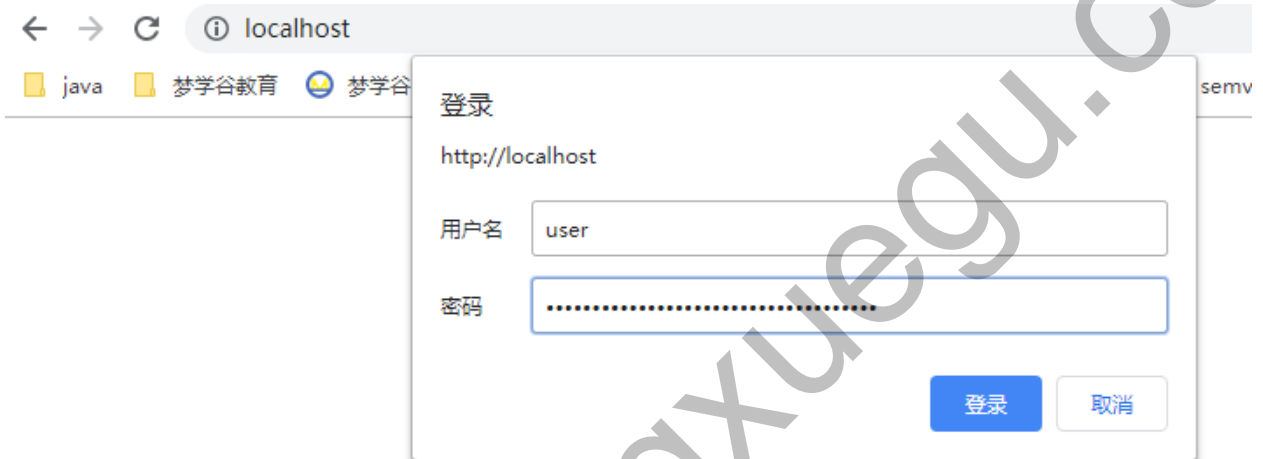
4.1.4 测试效果

1. 运行 mengxuegu-security-web 下的 com.mengxuegu.WebApplication 启动web服务
2. 查看启动日志，控制台会打印出一个随机密码，用于登录使用

```
WebApplication x
initialization completed in 3912 ms
16:16:30.934 INFO 1884 --- [ main ] .s.s.UserDetailsServiceAutoConfigura
Using generated security password: ffe7f906-7b09-4e49-b579-1e3d573b44ff
16:16:31.059 INFO 1884 --- [ main ] o.s.s.web.DefaultSecurityFilterChain
request, [org.springframework.security.web.context.request.async.WebAsyncManager]
```

3. 访问 <http://localhost:8001> 发现会弹出一个登录窗口，要求登录后才可访问首页

用户名默认是 `user`，密码是启动项目时控制台打印的



4. 输入有效身份信息认证成功后，会跳转到上一次引发登录页的地址，如：首页。

4.2 基于内存存储认证信息

4.2.1 概述

上面用户名和密码是 Spring Security 为我们提供的，下面基于内存存储自定义用户名和密码。

4.2.2 实现流程

1. 在 `configure(AuthenticationManagerBuilder auth)` 方法中指定用户名和密码、权限标识

```
1 @Override
2 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
3     // 用户信息存储在内存中
4     auth.inMemoryAuthentication().withUser("mengxuegu")
5         .password("1234").authorities("ADMIN");
6 }
```

2. 重启 `mengxuegu-security-web` 下的 `com.mengxuegu.WebApplication`，控制台没有自动密码了

3. 重新访问 <http://localhost>，一样弹出登录窗口，输入用户名 `mengxuegu`，密码 `1234`

4. 提交登录后还是继续弹出登录页，此时查看 IDEA 控制台，发现报错

```
1 java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
```

```
java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"  
    at org.springframework.security.crypto.password.DelegatingPasswordEncoder$UnmappedIdPasswordEncoder.matches(DelegatingPasswordEncoder.java:61)  
    at org.springframework.security.crypto.password.DelegatingPasswordEncoder.matches(DelegatingPasswordEncoder.java:61)
```

4.3 解决加密解密问题

发送请求后，控制台打印错误日志：

```
1 java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
```

4.3.1 分析问题

- 关注：PasswordEncoder 加密解密统一接口，它有很多的实现类，通过 ctrl+alt+b 查看其实现类
encode 用于加密明文
matches 输入的密码与数据库中的密码对比
upgradeEncoding 是否需要编码，一般不需要。

```
1 public interface PasswordEncoder {  
2  
3     //加密(我们来调用，一般在注册的时候，将前端传过来的密码，进行加密后保存进数据库)  
4     String encode(CharSequence rawPassword);  
5  
6     //加密前后对比(一般用来比对前端提交过来的密码和数据库存储的加密密码，也就是明文和密文的对比)  
7     boolean matches(CharSequence rawPassword, String encodedPassword);  
8  
9     //是否需要再次进行编码增加安全性，默认不需要  
10    default boolean upgradeEncoding(String encodedPassword) {  
11        return false;  
12    }  
13 }
```

- 在 Spring Security 5.0 版本前，加密的 PasswordEncoder 接口默认实现类为 NoOpPasswordEncoder，这个是可以不用加密的，直接使用明文密码存储。当前已经标注过时了。
- 在 Spring Security 5.0 版本后，默认实现类改为了 DelegatingPasswordEncoder，这个实现类要求我们必须对加密后存储。

4.3.2 解决问题

- 在 SpringSecurityConfig 指定 BCryptPasswordEncoder 加密方式
相同的密码，在每次加密后的结果都不一样的，因为它每次都会随机生成盐值，会将随机生成的盐加到密码串中
每次判断时，通过随机生成的盐反推回加密时的密码串，最终判断是否匹配

加密的最终结果分为两部分，盐值 + MD5(password+盐值), 调用 matches(..) 方法的时候，先从密文中得到盐值，用该盐值加密明文和最终密文作对比，

这样可以避免有一个密码被破解，其他相同的密码的帐户都可以破解。因为通过当前机制相同密码生成的密文都不一样。

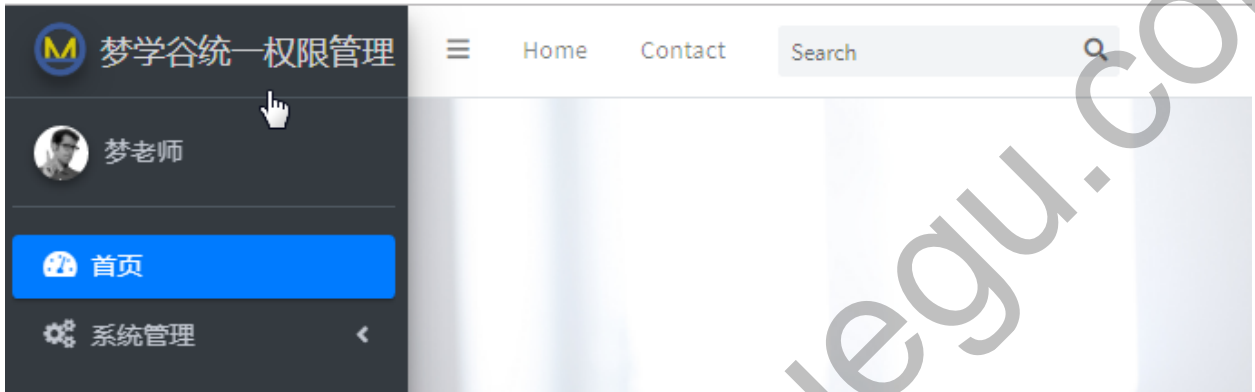
加密过程（注册）：aaa (盐值) + 123(密码明文) > 生成密文 > 最终结果 盐值.密文：aaa.asdIkf 存入数据库

校验过程（登录）：aaa (盐值, 数据库中得到) + 123(用户输入密码)> 生成密文 aaa.asdIkf，与数据库对比一致密码正确

```
1 。。。省略
2
3 @Configuration
4 @EnableWebSecurity //启动 SpringSecurity 过滤器链功能
5 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
6     Logger logger = LoggerFactory.getLogger(getClass());
7     @Bean
8     public PasswordEncoder passwordEncoder() {
9         // 设置默认的加密方式
10        return new BCryptPasswordEncoder();
11    }
12
13    /**
14     * 认证管理器：
15     * 1、认证信息提供方式（用户名、密码、当前用户的资源权限）
16     * 2、可采用内存存储方式，也可能采用数据库方式等
17     * @param auth
18     * @throws Exception
19     */
20    @Override
21    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
22        // 用户信息存储在内存中
23        String password = passwordEncoder().encode("1234");
24        logger.info("加密之后存储的密码：" + password);
25        auth.inMemoryAuthentication().withUser("mengxuegu")
26            .password(password).authorities("ADMIN");
27    }
28
29    /**
30     * 资源权限配置（过滤器链）：
31     * 1、被拦截的资源
32     * 2、资源所对应的角色权限
33     * 3、定义认证方式：httpBasic、httpForm
34     * 4、定制登录页面、登录请求地址、错误处理方式
35     * 5、自定义 spring security 过滤器
36     * @param http
37     * @throws Exception
38     */
39    @Override
40    protected void configure(HttpSecurity http) throws Exception {
41        http.httpBasic()
42            .and()
43            .authorizeRequests() // 认证请求
```

```
44         .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
45         ; // 分号`; 不要少了
46     }
47 }
```

- 重启 `mengxuegu-security-web` 下的 `com.mengxuegu.WebApplication`
- 重新访问 `http://localhost` 弹出登录窗口，输入 `mengxuegu/1234`
控制台不报错，成功跳转到引发认证请求中，即首页。



4.4 HttpForm 表单认证方式

4.4.1 概述

上面弹出登录窗口，体验不太好，希望通过页面的方式展示登录页面，可采用 `HttpForm` 表单认证来实现这个功能。

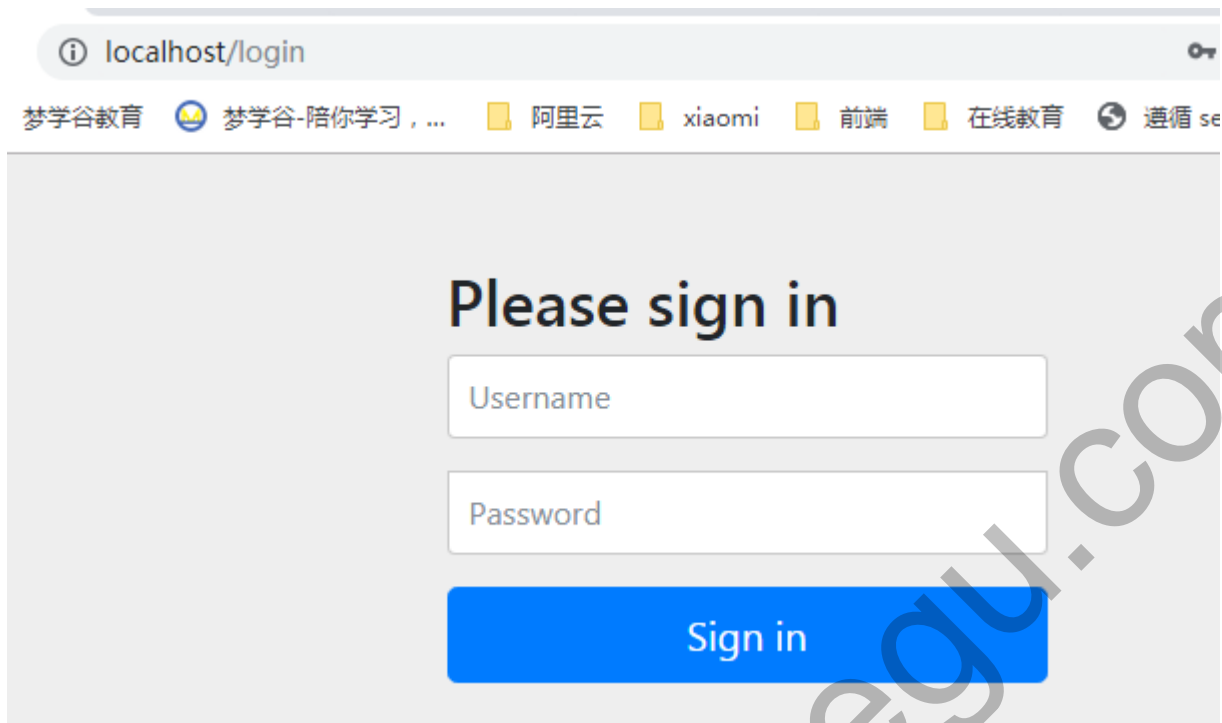
4.4.2 实现流程

1. 在 `SpringSecurityConfig.configure` 中通过 `http.formLogin()` 方法指定为表单认证方式

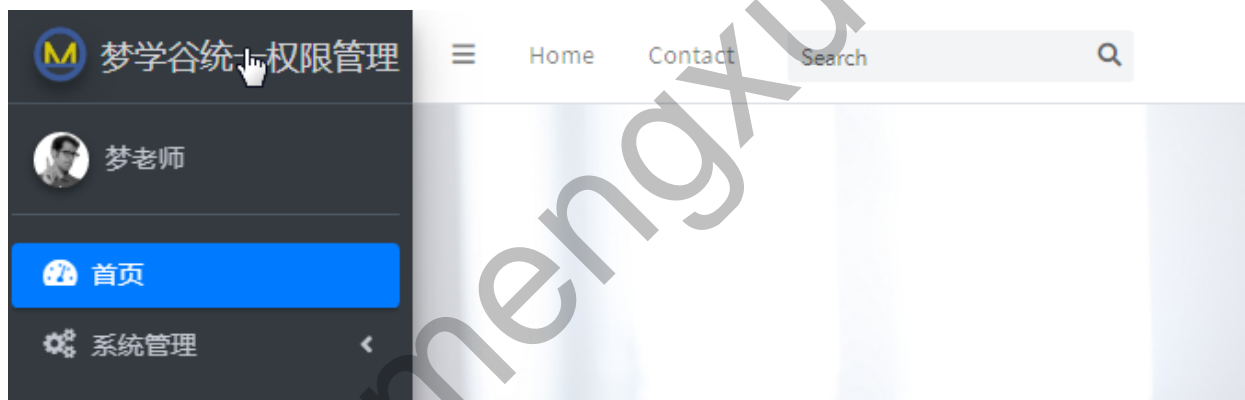
```
1  @Override
2  protected void configure(HttpSecurity http) throws Exception {
3      // http.httpBasic()
4      http.formLogin() // 表单认证
5      .and()
6      .authorizeRequests() // 认证请求
7      .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
8      ; // 分号`; 不要少了
9  }
```

2. 重新启动 `mengxuegu-security-web` 下的 `com.mengxuegu.WebApplication`
3. 访问 `http://localhost` 发现不是弹出登录窗口，而是自动请求一个 `localhost/login` 登录页面。

注意：如果重启项目后，还是一个弹窗，关闭浏览器，再打开浏览器进行访问。还不行就清除浏览器缓存

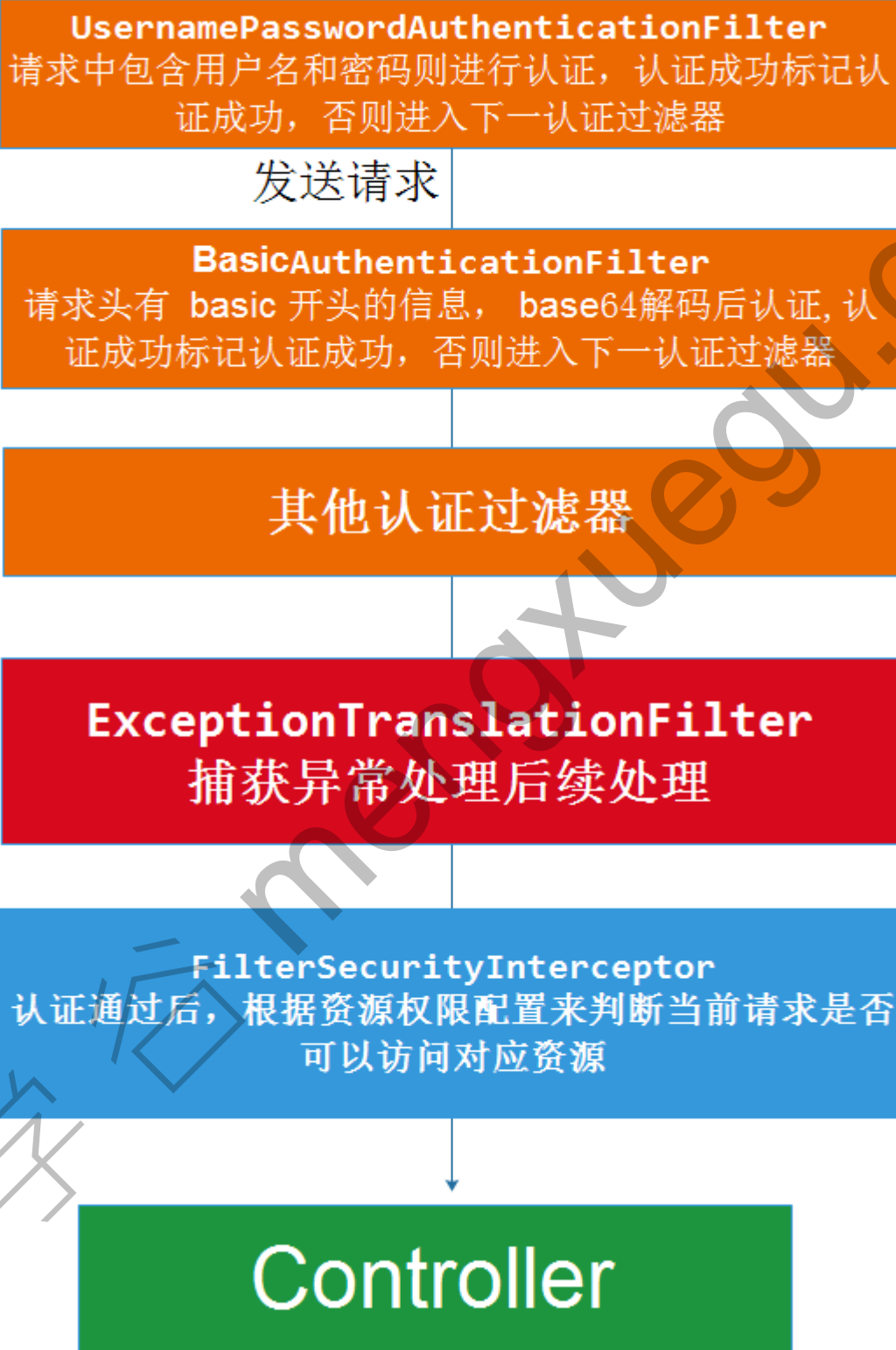


4. 登录页面中输入 mengxuegu/1234 ，验证通过后跳转目标页面。



4.5 分析 SpringSecurity 底层认证流程

SpringSecurity 采用过滤链实现认证与授权



第五章 SpringSecurity 身份认证实战

5.1 自定义登录页面

表单认证方式 Spring Security 默认提供了一个 bootstrap 登录页面，如果希望使用自定义的登录页面怎么办？

5.1.1 指定跳转自定义登录页面的URL

SpringSecurityConfig.configure(HttpSecurity http) 中使用 `loginPage("/login/page")` 指定前往认证请求。

```
1  @Override
2  protected void configure(HttpSecurity http) throws Exception {
3  //    http.httpBasic()
4      http.formLogin() // 表单认证
5          .loginPage("/login/page") // 交给 /login/page 响应认证(登录)页面
6          .and()
7          .authorizeRequests() // 认证请求
8          .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
9      ; // 分号`;`不要少了
10 }
```

5.1.2 实现登录控制器

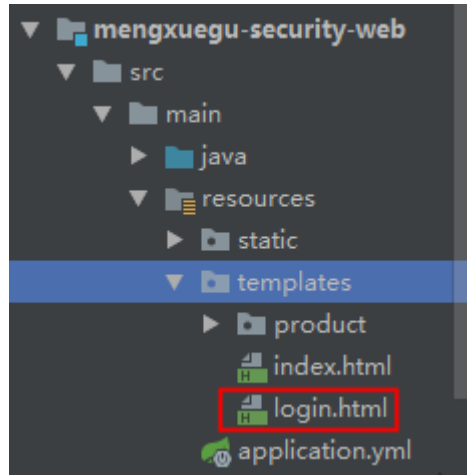
在 mengxuegu-security-**core** 创建 `com.mengxuegu.security.controller.CustomLoginController`，
用于实现认证（登录）处理。

```
1  package com.mengxuegu.security.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import javax.servlet.http.HttpServletRequest;
6  import javax.servlet.http.HttpServletResponse;
7
8  /**
9   * 登录处理
10  * @Author: 梦学谷 www.mengxuegu.com
11  */
12  @Controller
13  public class CustomLoginController {
14
15      /**
16       * 前往认证(登录)页面
17       * @return
18       */
19      @RequestMapping("/login/page")
20      public String toLogin(HttpServletRequest request, HttpServletResponse response) {
21          // 响应一个登录页面 classpath: /templates/login.html
22          return "login";
23      }
24  }
```

```
24  
25 }
```

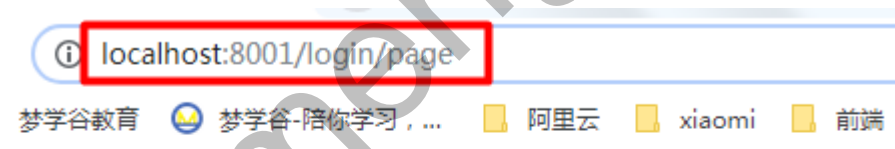
5.1.3 添加登录页面

找到 login.html 静态文件添加到 mengxuegu-security-web 模块的 templates 目录下



5.1.4 测试

- 重启 mengxuegu-security-web 下的 com.mengxuegu.WebApplication
- 访问 <http://localhost> 会进入跳转到 <http://localhost/login/page> ,
并且请求报错 localhost 将您重定向的次数过多



该网页无法正常运行

localhost 将您重定向的次数过多。

尝试清除 Cookie.

ERR_TOO_MANY_REDIRECTS

5.1.5 解决重定向的次数过多

分析问题

因为当前将所有请求都被拦截，拦截后会重写向到认证请求上（默认是 `/login`），当认证通过后可以访问，而现在在认证请求改为 `/login/page`，它也一样被拦截了，拦截后就重写向回认证请求上（修改的 `/login/page` 上），这样反复请求 `/login/page` 拦截后就报重定向次数过多。

解决问题

只需要在 `SpringSecurityConfig` 中放行对应请求资源：

1. 放行跳转认证请求（前往登录页面请求）

```
1  @Override
2  protected void configure(HttpSecurity http) throws Exception {
3      // http.httpBasic()
4      http.formLogin() // 表单认证
5          .loginPage("/login/page") // 交给 /login/page 响应认证(登录)页面
6          .and()
7          .authorizeRequests() // 认证请求
8          .antMatchers("/login/page").permitAll() // 放行跳转认证请求
9          .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
10     ; // 分号`;`不要少了
11 }
```

2. 放行静态资源路径，注意重写是：`configure(WebSecurity web)`

```
1  /**
2   * 释放静态资源
3   * @param web
4   */
5  @Override
6  public void configure(WebSecurity web){
7      web.ignoring().antMatchers("/dist/**", "/modules/**", "/plugins/**");
8  }
```

3. 重启项目，访问 <http://localhost/>，成功重定向到 <http://localhost/login/page> 认证页面

5.2 登录表单提交处理

5.2.1 概述

只要按照 Spring Security 规则进行配置后，Spring Security 会自动帮我们进行身份认证。

5.2.2 实现流程

1. 登录表单默认的 `action="/login"`，通过 `loginProcessingUrl("/login/form")` 修改为 `/login/form`。
2. 登录表单的用户名参数名默认是 `name="username"`，通过 `usernameParameter("name")` 修改为 `name`。
3. 登录表单的密码参考名默认是 `name="password"`，通过 `passwordParameter("pwd")` 修改为 `pwd`。

```
1  @Override
2  protected void configure(HttpSecurity http) throws Exception {
3      // http.httpBasic()
4      http.formLogin() // 表单认证
5          .loginPage("/login/page") // 交给 /login/page 响应认证(登录)页面
6          .loginProcessingUrl("/login/form") // 登录表单提交处理Url, 默认是 /login
7          .usernameParameter("name") // 默认用户名的属性名是 username
8          .passwordParameter("pwd") // 默认密码的属性名是 password
9          .and()
10         .authorizeRequests() // 认证请求
11         .antMatchers("/login/page").permitAll() // 放行跳转登录请求
12         .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
13         ; // 分号`; 不要少了
14     }
```

4. 检查 mengxuegu-security-web 工程中 login.html 页面相关标签属性值是否和上面一致。

注意: html标签上有引入thymeleaf名称空间 <html xmlns:th="http://www.thymeleaf.org">

```
1  <form th:action="@{/login/form}" th:method="post">
2      <div class="form-group has-feedback">
3          <input type="text" name="name" class="form-control" required placeholder="用户名">
4          <span class="glyphicon glyphicon-envelope form-control-feedback"></span>
5      </div>
6      <div class="form-group has-feedback">
7          <input type="password" name="pwd" class="form-control" required placeholder="密码">
8          <span class="glyphicon glyphicon-lock form-control-feedback" ></span>
9      </div>
10
11      ...
12  </form>
```

5.2.3 测试

1. 重启项目
2. 访问 <http://localhost/index> 重定向到登录页
3. 输入错误用户信息，回到登录页面 <http://localhost/login/page?error>
4. 输入有效用户信息 mengxuegu/1234, 进入首页。
5. 如果输入有效用户信息，还是回到登录页，则要禁用 CSRF 攻击。
 - CSRF (Cross-site request forgery) 跨站请求伪造
 - 关闭 CSRF 攻击

```
1  .and().csrf().disable()
```

5.3 登录页面回显提示信息

5.3.1 概述

1. 当提交登录表单数据认证失败后，通过 <http://localhost/login/page?error> 重定向回登录页，此时地址带有一个 `error` 参数，标识认证失败。
并且当用户名或密码错误时，后台会响应提示信息 `Bad credentials`，我们要将提示信息在登录页上回显。
2. 默认情况下，提示信息默认都是英文的，其实是可配置成中文信息。

5.3.2 实现页面回显提示信息

1. <http://localhost/login/page?error> 有 `error` 参数就是认证失败 `th:if="${param.error}"`
2. `login.html` 页面渲染提示信息 `th:text="${session.SPRING_SECURITY_LAST_EXCEPTION?.message}"`

注意 html 标签上有引入 thymeleaf 名称空间 `<html xmlns:th="http://www.thymeleaf.org">`

```
1 <form th:action="@{/login/form}" action="index.html" method="post">
2   <div class="input-group mb-3">
3     <input name="name" type="text" class="form-control" placeholder="用户名">
4     <div class="input-group-append">
5       <div class="input-group-text">
6         <span class="fa fa-user"></span>
7       </div>
8     </div>
9   </div>
10  <div class="input-group mb-3">
11    <input name="pwd" type="password" class="form-control" placeholder="密码">
12    <div class="input-group-append">
13      <div class="input-group-text">
14        <span class="fas fa-lock"></span>
15      </div>
16    </div>
17  </div>
18  <div class="row mb-2">
19    <div class="col-6">
20      <input name="code" type="text" class="form-control" placeholder="验证码">
21    </div>
22    <!--<div class="col-6">-->
23    <!---->
24    <!--</div>-->
25  </div>
26  <!-- 提示信息, 表达式红线没关系, 忽略它 -->
27  <div th:if="${param.error}">
28    <span th:text="${session.SPRING_SECURITY_LAST_EXCEPTION?.message}" style="color:red">用户名或
    密码错误</span>
29  </div>
30
31  <div class="row">
32    <div class="col-8">
```

```
33     <div class="icheck-primary">
34         <input name="remember-me" type="checkbox" id="remember">
35         <label for="remember">
36             记住我
37         </label>
38     </div>
39 </div>
40 <!-- /.col -->
41 <div class="col-4">
42     <button type="submit" class="btn btn-primary btn-block">登录</button>
43 </div>
44 <!-- /.col -->
45 </div>
46 </form>
```

3. 重启项目，当输入错误用户信息时，页面是否会回显 `Bad credentials`

5.3.3 实现中文提示信息

1. 观察 `spring-security-core-xxx.jar` 下有国际化配置文件 `messages_xxx.properties`
2. 默认 `ReloadableResourceBundleMessageSource` 是加载了 `messages.properties` 英文配置文件；
3. 应该手动指定加载 `messages_zh_CN.properties` 中文配置文件。
4. 在 `mengxuegu-security-core` 创建 `com.mengxuegu.security.config.ReloadMessageConfig`

```
1 package com.mengxuegu.security.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.context.support.ReloadableResourceBundleMessageSource;
6
7 /**
8  * 加载认证信息配置类
9  * @Author: 梦学谷 www.mengxuegu.com
10  */
11 @Configuration
12 public class ReloadMessageConfig {
13
14     @Bean // 加载中文的认证提示信息
15     public ReloadableResourceBundleMessageSource messageSource() {
16         ReloadableResourceBundleMessageSource messageSource = new
17         ReloadableResourceBundleMessageSource();
18         // .properties 不要加到后面
19         messageSource.setBasename("classpath:org/springframework/security/messages_zh_CN");
20         return messageSource;
21     }
22 }
```


5.3.4 测试

- 重启项目，当输入错误用户信息时，页面是否会回显 用户名或密码错误

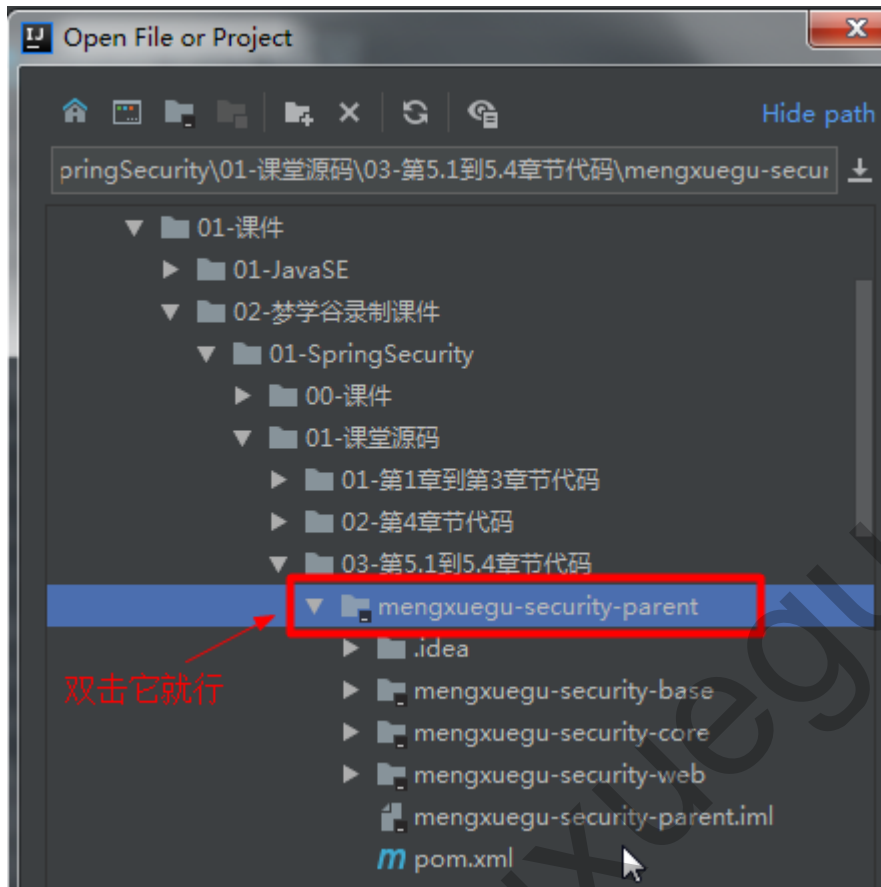


- 用户信息输入正确，会重定向回引发认证的请求中，即首页。

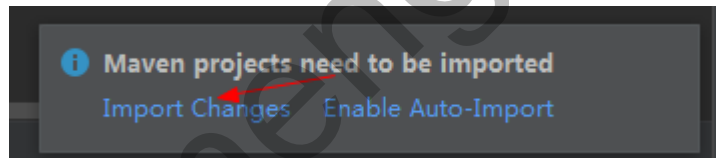
5.4 备份代码

将5.4章节以前的代码备份在 01-课堂源码\03-第5.1到5.4章节代码

idea 导入方式：File 》 Open ,找到项目 mengxuegu-security-parent 双击打开



打开后点击右下角 import Changes



5.5 认证相关URL实现可配置

当前在 SpringSecurityConfig 中配置的认证相关URL是写死，这些 URL 根据应用系统的不同，可能需要配置不同的 URL，那我们可以抽取到 application.yml 进行可配置。

5.5.1 配置 application.yml

1. 在 **mengxuegu-security-web** \src\main\resources\application.yml 文件中配置如下：

```
1 server:
2   port: 80
3
4 mengxuegu:
5   security:
6     authentication:
7       loginPage: /login/page # 响应认证(登录)页面URL
8       loginProcessingUrl: /login/form # 登录表单提交处理Url
```

```
9     usernameParameter: name # 登录表单用户名的属性名
10    passwordParameter: pwd # 登录表单密码的属性名
11    staticPaths: # 静态资源 "/dist/**", "/modules/**", "/plugins/**"
12        - /dist/**
13        - /modules/**
14        - /plugins/**
```

5.5.2 读取自定义配置数据

1. 在 **mengxuegu-security-core** 中创建 `com.mengxuegu.security.properties.SecurityProperties`
 - `@ConfigurationProperties(prefix = "mengxuegu.security")` 绑定 `application.yml` 配置文件中以 `mengxuegu.security` 前缀的数据
 - 类上不要少了 `@Component` 注解

AuthenticationProperties 报错是因为类还未创建出来，继续往下第2步创建即可。

```
1 package com.mengxuegu.security.properties;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.stereotype.Component;
5
6 /**
7  * @Author: 梦学谷 www.mengxuegu.com
8  */
9 @Component // 不要少了
10 @ConfigurationProperties( prefix = "mengxuegu.security")
11 public class SecurityProperties {
12
13     // 将application.yml 中的 mengxuegu.security.authentication 下面的值绑定到此对象中
14     private AuthenticationProperties authentication;
15
16     public AuthenticationProperties getAuthentication() {
17         return authentication;
18     }
19
20     public void setAuthentication(AuthenticationProperties authentication) {
21         this.authentication = authentication;
22     }
23 }
24
```

2. 在 **mengxuegu-security-core** 中创建 `com.mengxuegu.security.properties.AuthenticationProperties`

```
1 package com.mengxuegu.security.properties;
2
3 /**
4  * 认证相关动态配置
5  * @Author: 梦学谷 www.mengxuegu.com
```

```
6  */
7  public class AuthenticationProperties {
8      // application.yml 没配置取默认值
9      private String loginPage = "/login/page";
10     private String loginProcessingUrl = "/login/form";
11     private String usernameParameter = "name";
12     private String passwordParameter = "pwd";
13     private String[] staticPaths = {"/dist/**", "/modules/**", "/plugins/**"};
14
15     public String getLoginPage() {
16         return loginPage;
17     }
18
19     public void setLoginPage(String loginPage) {
20         this.loginPage = loginPage;
21     }
22
23     public String getLoginProcessingUrl() {
24         return loginProcessingUrl;
25     }
26
27     public void setLoginProcessingUrl(String loginProcessingUrl) {
28         this.loginProcessingUrl = loginProcessingUrl;
29     }
30
31     public String getUsernameParameter() {
32         return usernameParameter;
33     }
34
35     public void setUsernameParameter(String usernameParameter) {
36         this.usernameParameter = usernameParameter;
37     }
38
39     public String getPasswordParameter() {
40         return passwordParameter;
41     }
42
43     public void setPasswordParameter(String passwordParameter) {
44         this.passwordParameter = passwordParameter;
45     }
46
47     public String[] getStaticPaths() {
48         return staticPaths;
49     }
50
51     public void setStaticPaths(String[] staticPaths) {
52         this.staticPaths = staticPaths;
53     }
54 }
```

5.5.3 重构 SpringSecurityConfig

1. 将 `SecurityProperties` 注入到 `SpringSecurityConfig`

```
1 @Autowired
2 private SecurityProperties securityProperties;
```

2. 将 `SpringSecurityConfig#configure` 的URL均通过 `securityProperties` 获取

```
1 @Override
2 protected void configure(HttpSecurity http) throws Exception {
3     // http.httpBasic()
4     http.formLogin() // 表单认证
5         .loginPage(securityProperties.getAuthentication().getLoginPage())
6         .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl())
7         .usernameParameter(securityProperties.getAuthentication().getUsernameParameter())
8         .passwordParameter(securityProperties.getAuthentication().getPasswordParameter())
9         .and()
10        .authorizeRequests() // 认证请求
11        .antMatchers(
12            securityProperties.getAuthentication().getLoginPage()
13        ).permitAll()
14        .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
15        ; // 分号`;`不要少了
16    }
17
18    @Override
19    public void configure(WebSecurity web) {
20        web.ignoring().antMatchers(
21            securityProperties.getAuthentication().getStaticPaths());
22    }
```

5.6 动态认证用户信息

5.6.1 概述

当前身份认证的用户名和密码是启动服务器自动生成的，或者是代码中写死的，存储在内存中。而实际项目中应该从动态的从数据库中获取进行身份认证。

5.6.2 实现流程

1. 重点关注 `UserDetailsService` 、 `UserDetails` 接口
2. 自定义一个 `UserDetailsService` 接口的实现类 `CustomUserDetailsService`，实现该接口中的 `loadUserByUsername` 方法，
通过该方法定义获取用户信息的逻辑。

- 从数据库获取到的用户信息封装到 `UserDetail` 接口的实现类中（Spring Security 提供了一个 `org.springframework.security.core.userdetails.User` 实现类封装用户信息）。
- 如果未获取到用户信息，则抛出异常 `throws UsernameNotFoundException`

```
1 public interface UserDetails extends Serializable {  
2     此用户可访问的资源权限  
3     Collection<? extends GrantedAuthority> getAuthorities();  
4     用户名  
5     String getPassword();  
6     密码  
7     String getUsername();  
8     帐户是否过期(true 未过期, false 已过期)  
9     boolean isAccountNonExpired();  
10  
11    帐户是否被锁定 ( true 未锁定, false 已锁定 ), 锁定的用户是可以恢复的  
12    boolean isAccountNonLocked();  
13  
14    密码是否过期 ( 安全级别比较高的系统, 如30天要求更改密码, true 未过期, false 过期 )  
15    boolean isCredentialsNonExpired();  
16  
17    帐户是否可用 ( 一般指定是否删除, 系统一般不会真正的删除用户信息, 而是假删除, 通过一个状态码标志  
    用户被删除 ) 删除的用户是可以恢复的  
18    boolean isEnabled();  
19 }
```

5.6.3 编码实现

1. 在 `mengxuegu-security-web` 创建 `com.mengxuegu.security.CustomUserDetailsService` ,

注意:

- 应用服务中才知道如何获取用户信息，所以在 `mengxuegu-security-web` 中创建。
- `@Component("customUserDetailsService")` 注解在实现类上不要少了

```
1 package com.mengxuegu.security;  
2  
3 import org.slf4j.Logger;  
4 import org.slf4j.LoggerFactory;  
5 import org.springframework.beans.factory.annotation.Autowired;  
6 import org.springframework.security.core.authority.AuthorityUtils;  
7 import org.springframework.security.core.userdetails.User;  
8 import org.springframework.security.core.userdetails.UserDetails;  
9 import org.springframework.security.core.userdetails.UserDetailsService;  
10 import org.springframework.security.core.userdetails.UsernameNotFoundException;  
11 import org.springframework.security.crypto.password.PasswordEncoder;  
12 import org.springframework.stereotype.Component;  
13  
14 /**  
15  * @Author: 梦学谷 www.mengxuegu.com  
16  */  
17 @Component("customUserDetailsService")
```

```
18 public class CustomUserDetailsService implements UserDetailsService {
19     Logger logger = LoggerFactory.getLogger(getClass());
20
21     @Autowired
22     PasswordEncoder passwordEncoder;
23
24     @Override
25     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
26         logger.info("请求认证的用户名1 : " + username);
27
28         // 1. 根据请求用户名向数据库中查询用户信息
29         if("meng".equalsIgnoreCase(username)) {
30             throw new UsernameNotFoundException("用户名或密码错误");
31         }
32         // 如果有此用户信息, 假设数据库查询到的用户密码是 1234
33         String password = passwordEncoder.encode("1234");
34
35         // 2.查询用户拥有权限
36
37         // 3.封装用户信息: username用户名,password数据库中的密码,authorities资源权限标识符
38         // SpringSecurity 底层会校验是否身份合法。
39         return new User(username, password,
40             AuthorityUtils.commaSeparatedStringToAuthorityList("ADMIN"));
41     }
42 }
```

注意：用户名 meng 密码 1234

2. 重构 com.mengxuegu.security.config.SpringSecurityConfig

- 注入 CustomUserDetailsService
- configure(AuthenticationManagerBuilder auth) 方法中指定认证方式

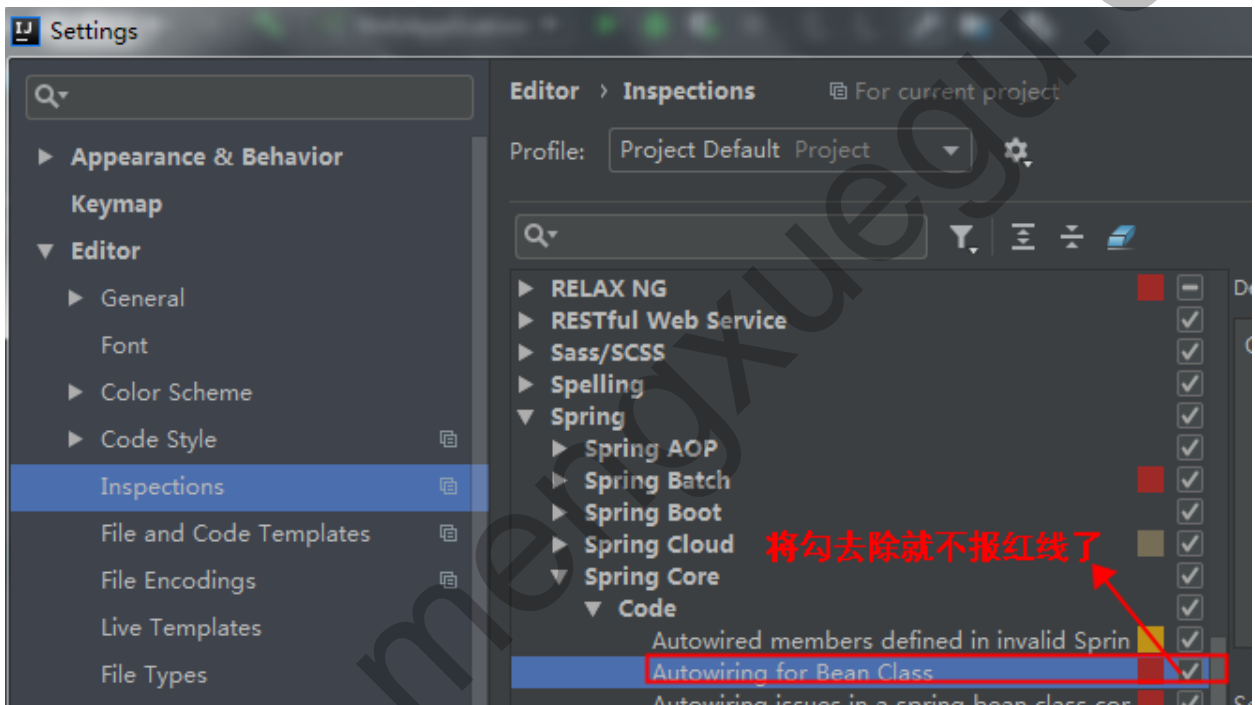
```
1 ...省略
2
3 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
4
5     ...省略
6
7     @Autowired
8     private UserDetailsService customUserDetailsService;
9     /**
10      * 认证管理器：
11      * 1、认证信息提供方式（用户名、密码、当前用户的资源权限）
12      * 2、可采用内存存储方式，也可能采用数据库方式等
13      * @param auth
14      * @throws Exception
15      */
16     @Override
17     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
18         // 用户信息存储在内存中
19         // String password = passwordEncoder().encode("1234");
```



```
20 // logger.info("加密之后存储的密码：" + password);
21 // auth.inMemoryAuthentication().withUser("mengxuegu")
22 //     .password(password).authorities("ADMIN");
23 // 用户信息存储在数据库中
24 auth.userDetailsService(customUserDetailsService);
25
26 }
27
28 ...省略
29 }
```

3. 如果上面 `customUserDetailsService` 有红色波浪线，idea自身检测的问题这不是bug，不影响项目运行。

也可让idea不检测，就会去除红线，如下操作：



4. 运行 WebApplication，重启项目

5. 访问 <http://localhost/index> 被拦截，在输入非 meng 用户名，提示 用户名或密码错误

用户名或密码错误

☐ 记住我

登录

5.7 自定义认证成功处理器

5.7.1 概述

重点关注 `AuthenticationSuccessHandler` 接口

当前登录成功后，跳转到之前请求的 url，而现在希望登录成功后，实现其他的业务逻辑。比如累计积分、通过 Ajax 请求响应一个JSON数据，前端接收到响应的数据进行跳转。那可以使用自定义登录成功处理逻辑。

5.7.2 编码实现

1. 将 02-配套资料 目录下的 `MengxueguResult.java` 工具类拷贝到 `mengxuegu-security-base` 模块中的 `com.mengxuegu.base.result` 包下。用于封装响应JSON数据。
2. 创建 `com.mengxuegu.security.authentication.CustomAuthenticationSuccessHandler`，实现 `CustomAuthenticationSuccessHandler` 接口

注意:实现类上不要少了注解 `@Component("customAuthenticationSuccessHandler")`

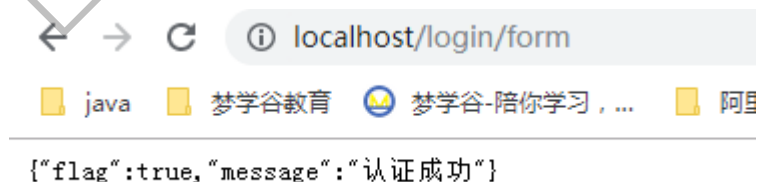
```
1 package com.mengxuegu.security.authentication;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import org.springframework.security.core.Authentication;
5 import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
6 import org.springframework.stereotype.Component;
7 import javax.servlet.ServletException;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10 import java.io.IOException;
11
12 /**
13  * 认证成功处理器
14  * @Author: 梦学谷 www.mengxuegu.com
15  */
16 @Component("customAuthenticationSuccessHandler")
17 public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {
18
19     /**
20      * 认证成功后处理逻辑
21      * @param authentication 封装了用户信息 UserDetails，访问IP等
22      */
23     @Override
24     public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
25         Authentication authentication) throws IOException, ServletException {
26         // 当认证成功后，响应 JSON 数据给前端
27         MengxueguResult result = MengxueguResult.ok("认证成功");
28         response.setContentType("application/json;charset=UTF-8");
29         response.getWriter().write(result.toJsonString());
30     }
31 }
```

3. 在 `SpringSecurityConfig` 中注入 和 引用自定义认证成功处理器 `customAuthenticationSuccessHandler`

```
1 package com.mengxuegu.security.config;
2
3 ...import 省略
4
5 @Configuration
6 @EnableWebSecurity //启动 SpringSecurity 过滤器链功能
7 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
8
9     ...省略
10
11     // 注入自定义的认证成功处理器
12     @Autowired
13     private AuthenticationSuccessHandler customAuthenticationSuccessHandler;
14
15     @Override
16     protected void configure(HttpSecurity http) throws Exception {
17         http.formLogin() // 表单认证
18             .loginPage(securityProperties.getAuthentication().getLoginPage())
19             .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl())
20             .usernameParameter(securityProperties.getAuthentication().getUsernameParameter())
21             .passwordParameter(securityProperties.getAuthentication().getPasswordParameter())
22             .successHandler(customAuthenticationSuccessHandler) // 认证成功处理器
23             .and()
24             .authorizeRequests() // 认证请求
25             .antMatchers(
26                 securityProperties.getAuthentication().getLoginPage()
27             ).permitAll()
28             .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
29     ; // 分号`; 不要少了
30     }
31 }
```

5.7.3 测试

1. 重启项目，访问 <http://localhost> 跳转到登录页面
2. 登录成功后，页面响应数据



5.8 自定义认证失败处理器

5.8.1 概述

重点关注 `AuthenticationFailureHandler` 接口

登录错误后记录日志，当次数超过3次后，2小时内不允许登录，那可以使用自定义登录失败后，进行逻辑处理。

5.8.2 编码实现

1. 创建 `com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler`，实现 `AuthenticationFailureHandler` 接口

注意:实现类上不要少了注解 `@Component("customAuthenticationFailureHandler")`

```
1 package com.mengxuegu.security.authentication;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import org.springframework.http.HttpStatus;
5 import org.springframework.security.core.AuthenticationException;
6 import org.springframework.security.web.authentication.AuthenticationFailureHandler;
7 import org.springframework.stereotype.Component;
8
9 import javax.servlet.ServletException;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12 import java.io.IOException;
13
14 /**
15  * 认证失败处理器
16  * @Author: 梦学谷 www.mengxuegu.com
17  */
18 @Component("customAuthenticationFailureHandler")
19 public class CustomAuthenticationFailureHandler implements AuthenticationFailureHandler {
20     @Override
21     public void onAuthenticationFailure(HttpServletRequest request,
22         HttpServletResponse response, AuthenticationException exception) throws IOException,
23         ServletException {
24         // 认证失败状态码 401
25         MengxueguResult result = MengxueguResult.build(
26             HttpStatus.UNAUTHORIZED.value(), exception.getMessage());
27         response.setContentType("application/json;charset=UTF-8");
28         response.getWriter().write(result.toJsonString());
29     }
30 }
```

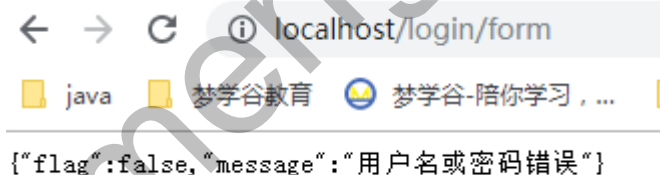
2. 在 `SpringSecurityConfig` 中注入 和 引用自定义认证失败处理器 `customAuthenticationFailureHandler`

```
1 ... 省略
2
3 // 注入自定义的认证成功处理器
4 @Autowired
5 private AuthenticationSuccessHandler customAuthenticationSuccessHandler;
6
7 @Autowired
```

```
8 private AuthenticationFailureHandler customAuthenticationFailureHandler;
9
10 @Override
11 protected void configure(HttpSecurity http) throws Exception {
12     // http.httpBasic()
13     http.formLogin() // 表单认证
14         .loginPage(securityProperties.getAuthentication().getLoginPage())
15         .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl())
16         .usernameParameter(securityProperties.getAuthentication().getUsernameParameter())
17         .passwordParameter(securityProperties.getAuthentication().getPasswordParameter())
18         .successHandler(customAuthenticationSuccessHandler) // 认证成功处理器
19         .failureHandler(customAuthenticationFailureHandler) // 认证失败处理器
20         .and()
21         .authorizeRequests() // 认证请求
22         .antMatchers(
23             securityProperties.getAuthentication().getLoginPage()
24             ).permitAll()
25         .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
26     ; // 分号`; 不要少了
27 }
28 ...省略
29 }
```

3. 重启项目，访问 <http://localhost>

4. 输入错误用户名和密码后，页面响应数据



localhost/login/form

java 梦学谷教育 梦学谷-陪你学习，...

```
{"flag":false,"message":"用户名或密码错误"}
```

5.9 升级异步请求认证处理

1. 如果是通过ajax发送请求, 应该响应 JSON 通知前端认证成功或失败
2. 否则直接重定向回来源请求

要实现这个效果应该在 `CustomAuthenticationFailureHandler` 和 `CustomAuthenticationSuccessHandler` 加上一个类型判断，且我们将这些类型都可配置的。

5.9.1 创建响应类型枚举类

在 `mengxuegu-security-core` 中创建枚举类 `com.mengxuegu.security.properties.LoginResponseType`

```
1 package com.mengxuegu.security.properties;
2 /**
3  * 认证响应类型
```

```
4  * @Author: 梦学谷 www.mengxuegu.com
5  */
6  public enum LoginResponseType {
7      /**
8       * 响应 JSON 字符串
9       */
10     JSON,
11
12     /**
13      * 重定向地址
14      */
15     REDIRECT
16 }
```

5.9.2 配置 application.yml

1. 在 **mengxuegu-security-web** \src\main\resources\application.yml 文件中配置登录类型

- REDIRECT 表示重向一个页面
- JSON 响应JSON字符串

```
1  server:
2    port: 80
3
4  spring:
5    thymeleaf:
6      cache: false #关闭thymeleaf缓存
7
8  mengxuegu:
9    security:
10     authentication:
11       loginPage: /login/page # 响应认证(登录)页面的URL
12       loginProcessingUrl: /login/form # 登录表单提交处理的url
13       usernameParameter: name # 登录表单提交的用户名的属性名
14       passwordParameter: pwd # 登录表单提交的密码的属性名
15       staticPaths: # 静态资源 "/dist/**", "/modules/**", "/plugins/**"
16         - /dist/**
17         - /modules/**
18         - /plugins/**
19       loginType: REDIRECT # 认证之后 响应的类型 : JSON/REDIRECT
```

5.9.3 读取自定义配置数据

1. 在 com.mengxuegu.security.properties.AuthenticationProperties 添加 loginType 属性

```
1  package com.mengxuegu.security.properties;
2
3  /**
```

```
4  * 认证相关动态配置
5  * @Author: 梦学谷 www.mengxuegu.com
6  */
7  public class AuthenticationProperties {
8      // application.yml 没配置取默认值
9      private String loginPage = "/login/page";
10     private String loginProcessingUrl = "/login/form";
11     private String usernameParameter = "name";
12     private String passwordParameter = "pwd";
13     private String[] staticPaths = {"/dist/**", "/modules/**", "/plugins/**"};
14
15     /**
16     * 登录成功后响应 JSON，还是重定向
17     * 如果application.yml 中没有配置，则取此初始值 REDIRECT
18     */
19     private LoginResponseType loginType = LoginResponseType.REDIRECT;
20
21     public LoginResponseType getLoginType() {
22         return loginType;
23     }
24
25     public void setLoginType(LoginResponseType loginType) {
26         this.loginType = loginType;
27     }
28
29     public String getLoginPage() {
30         return loginPage;
31     }
32
33     public void setLoginPage(String loginPage) {
34         this.loginPage = loginPage;
35     }
36
37     public String getLoginProcessingUrl() {
38         return loginProcessingUrl;
39     }
40
41     public void setLoginProcessingUrl(String loginProcessingUrl) {
42         this.loginProcessingUrl = loginProcessingUrl;
43     }
44
45     public String getUsernameParameter() {
46         return usernameParameter;
47     }
48
49     public void setUsernameParameter(String usernameParameter) {
50         this.usernameParameter = usernameParameter;
51     }
52
53     public String getPasswordParameter() {
54         return passwordParameter;
55     }
56
```



```
57 public void setPasswordParameter(String passwordParameter) {
58     this.passwordParameter = passwordParameter;
59 }
60
61 public String[] getStaticPaths() {
62     return staticPaths;
63 }
64
65 public void setStaticPaths(String[] staticPaths) {
66     this.staticPaths = staticPaths;
67 }
68 }
```

5.9.4 重构成功处理器

重构 com.mengxuegu.security.authentication.CustomAuthenticationSuccessHandler

1. 将实现改为**继承** **extends SavedRequestAwareAuthenticationSuccessHandler** 默认实现类
2. 加上判断响应 JSON 还是 重定向回认证来源请求
3. 采用 SecurityProperties.authentication.loginType 配置值进行判断认证响应类型。

```
1 package com.mengxuegu.security.authentication;
2
3 import com.alibaba.fastjson.JSON;
4 import com.mengxuegu.base.result.MengxueguResult;
5 import com.mengxuegu.security.properties.LoginResponseType;
6 import com.mengxuegu.security.properties.SecurityProperties;
7 import org.slf4j.Logger;
8 import org.slf4j.LoggerFactory;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.security.core.Authentication;
11 import org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler;
12 import org.springframework.stereotype.Component;
13
14 import javax.servlet.ServletException;
15 import javax.servlet.http.HttpServletRequest;
16 import javax.servlet.http.HttpServletResponse;
17 import java.io.IOException;
18
19 /**
20  * 认证成功处理器
21  * 1. 决定 响应json还是跳转页面，或者认证成功后进行其他处理
22  * @Author: 梦学谷 www.mengxuegu.com
23  */
24 @Component("customAuthenticationSuccessHandler")
25 //public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {
26 public class CustomAuthenticationSuccessHandler extends
    SavedRequestAwareAuthenticationSuccessHandler {
27     Logger logger = LoggerFactory.getLogger(getClass());
```

```
28 @Autowired
29 SecurityProperties securityProperties;
30
31 @Override
32 public void onAuthenticationSuccess(HttpServletRequest request,
33     HttpServletResponse response, Authentication authentication) throws IOException, ServletException
34 {
35     if(LoginResponseType.JSON.equals(
36         securityProperties.getAuthentication().getLoginType())) {
37         // 认证成功后，响应JSON字符串
38         MengxueguResult result = MengxueguResult.ok("认证成功");
39         response.setContentType("application/json;charset=UTF-8");
40         response.getWriter().write(result.toJsonString());
41     } else {
42         // 重定向到上次请求的地址上，引发跳转到认证页面的地址
43         logger.info("authentication: " + JSON.toJSONString(authentication));
44         super.onAuthenticationSuccess(request, response, authentication);
45     }
46
47 }
48 }
```

4. 重启测试，当 `application.yml` 配置不同登录类型，登录用户信息正确时，是否响应效果不一样。

5.9.5 重构失败处理器

1. 将实现改为继承 `extends SimpleUrlAuthenticationFailureHandler` 类
2. 加上判断响应 JSON 还是 重定向回认证来源请求
3. 采用 `SecurityProperties.authentication.loginType` 配置值进行判断认证响应类型。
4. 要指定重定向回登录页时，要指定上次请求地址加 `?error`

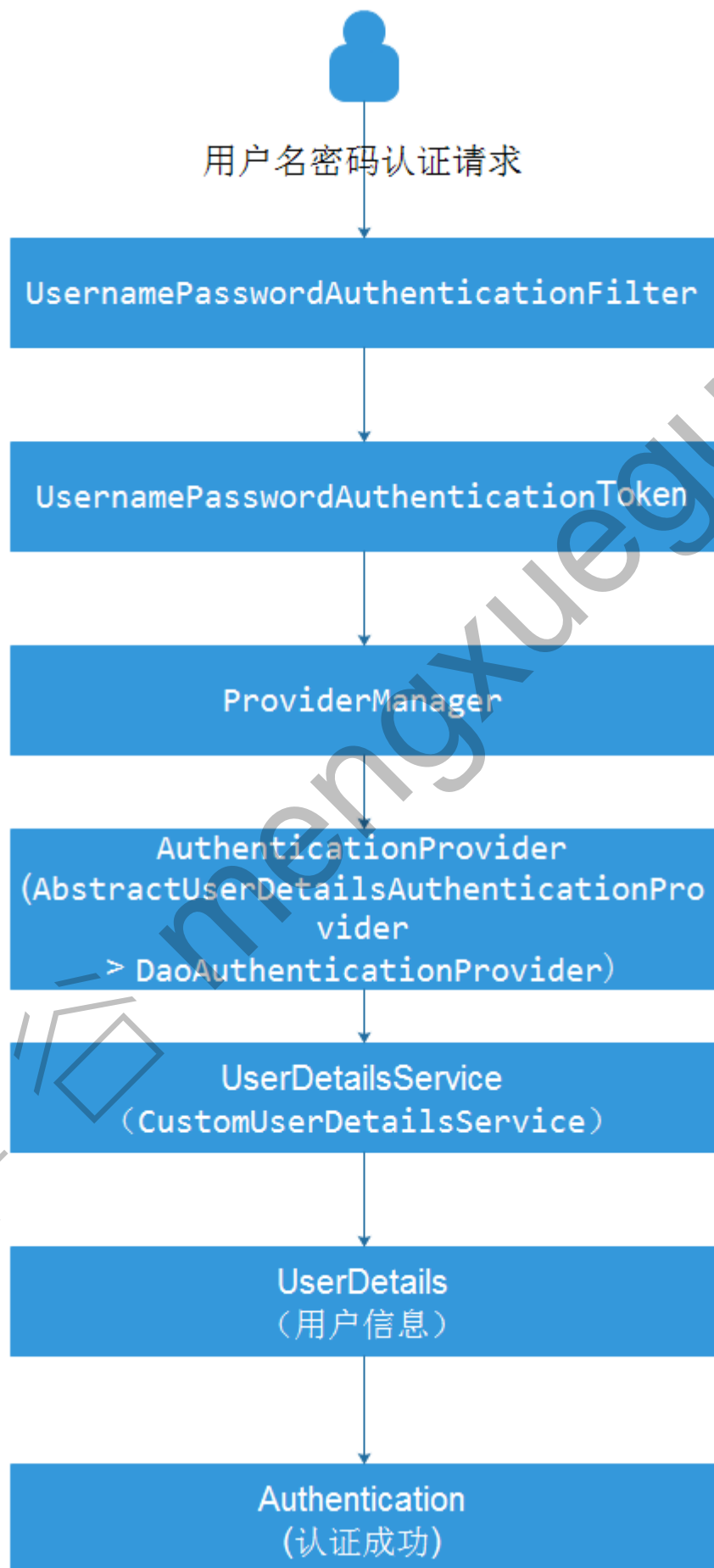
```
1 package com.mengxuegu.security.authentication;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import com.mengxuegu.security.properties.LoginResponseType;
5 import com.mengxuegu.security.properties.SecurityProperties;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.security.core.AuthenticationException;
9 import org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler;
10 import org.springframework.stereotype.Component;
11
12 import javax.servlet.ServletException;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15 import java.io.IOException;
16
17 /**
18  * 处理失败认证的
```

```
19  * @Author: 梦学谷 www.mengxuegu.com
20  */
21  @Component("customAuthenticationFailureHandler")
22  //public class CustomAuthenticationFailureHandler implements AuthenticationFailureHandler {
23  public class CustomAuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {
24
25      @Autowired
26      SecurityProperties securityProperties;
27      /**
28       *
29       * @param exception 认证失败时抛出异常
30       */
31      @Override
32      public void onAuthenticationFailure(HttpServletRequest request,
33          HttpServletResponse response, AuthenticationException exception) throws IOException,
34          ServletException {
35          if(LoginResponseType.JSON.equals(
36              securityProperties.getAuthentication().getLoginType())) {
37              // 认证失败响应JSON字符串，
38              MengxueguResult result = MengxueguResult.build(HttpStatus.UNAUTHORIZED.value(),
39                  exception.getMessage());
40              response.setContentType("application/json;charset=UTF-8");
41              response.getWriter().write(result.toJsonString());
42          } else {
43              // 重写向回认证页面，注意加上 ?error
44              super.setDefaultFailureUrl(
45                  securityProperties.getAuthentication().getLoginPage() + "?error");
46              super.onAuthenticationFailure(request, response, exception);
47          }
48      }
```

5. 重启测试，当 application.yml 配置不同登录类型，登录用户信息错误时，是否响应效果不一样。

5.10 分析用户名密码认证底层源码

认证流程图



UsernamePasswordAuthenticationFilter#attemptAuthentication

```
1 public Authentication attemptAuthentication(HttpServletRequest request,
2     HttpServletResponse response) throws AuthenticationException {
3     // post 请求方式
4     if (postOnly && !request.getMethod().equals("POST")) {
5         throw new AuthenticationServiceException(
6             "Authentication method not supported: " + request.getMethod());
7     }
8     // 从 request 中获取用户名、密码
9     String username = obtainUsername(request);
10    String password = obtainPassword(request);
11
12    if (username == null) {
13        username = "";
14    }
15    if (password == null) {
16        password = "";
17    }
18    username = username.trim();
19    // 将username和 password 构造成一个 UsernamePasswordAuthenticationToken 实例，
20    // 其中构建器中会是否认证设置为 authenticated=false 未认证
21    UsernamePasswordAuthenticationToken authRequest =
22        new UsernamePasswordAuthenticationToken(username, password);
23
24    //向 authRequest 对象中设置详细属性值。如添加了 remoteAddress、sessionId 值
25    setDetails(request, authRequest);
26    //调用 AuthenticationManager 的实现类 ProviderManager 进行验证
27    return this.getAuthenticationManager().authenticate(authRequest);
28 }
```

ProviderManager#authenticate

```
1 public Authentication authenticate(Authentication authentication)
2     throws AuthenticationException {
3     //获取当前的Authentication的认证类型
4     Class<? extends Authentication> toTest = authentication.getClass();
5     AuthenticationException lastException = null;
6     AuthenticationException parentException = null;
7     Authentication result = null;
8     Authentication parentResult = null;
9     boolean debug = logger.isDebugEnabled();
10    // 迭代认证提供者,不同认证方式有不同提供者,如:用户名密码认证提供者,手机短信认证提供者
11    for (AuthenticationProvider provider : getProviders()) {
12        // 选取当前认证方式对应的提供者
13        if (!provider.supports(toTest)) {
14            continue;
15        }
16        if (debug) {
```

```
17     logger.debug("Authentication attempt using "
18         + provider.getClass().getName());
19     }
20     try {
21         // 进行认证操作
22         // AbstractUserDetailsAuthenticationProvider》 DaoAuthenticationProvider
23         result = provider.authenticate(authentication);
24         if (result != null) {
25             //认证通过的话，将认证结果的details赋值到当前认证对象authentication。然后跳出循环
26             copyDetails(authentication, result);
27             break;
28         }
29     }
30     catch (AccountStatusException | InternalAuthenticationServiceException e) {
31         prepareException(e, authentication);
32         throw e;
33     } catch (AuthenticationException e) {
34         lastException = e;
35     }
36 }
37
38 //
39 if (result == null && parent != null) {
40     try {
41         result = parentResult = parent.authenticate(authentication);
42     }
43     catch (ProviderNotFoundException e) {
44     }
45     catch (AuthenticationException e) {
46         lastException = parentException = e;
47     }
48 }
49
50 if (result != null) {
51     if (eraseCredentialsAfterAuthentication
52         && (result instanceof CredentialsContainer)) {
53         ((CredentialsContainer) result).eraseCredentials();
54     }
55     if (parentResult == null) {
56         eventPublisher.publishAuthenticationSuccess(result);
57     }
58     return result;
59 }
60
61 if (lastException == null) {
62     lastException = new ProviderNotFoundException(messages.getMessage(
63         "ProviderManager.providerNotFound",
64         new Object[] { toTest.getName() },
65         "No AuthenticationProvider found for {0}"));
66 }
67
68 if (parentException == null) {
69     prepareException(lastException, authentication);
```

```
70     }  
71  
72     throw lastException;  
73 }
```

AbstractUserDetailsAuthenticationProvider

AbstractUserDetailsAuthenticationProvider是 AuthenticationProvider 的核心实现类

```
1  public Authentication authenticate(Authentication authentication)  
2      throws AuthenticationException {  
3      //如果authentication不是UsernamePasswordAuthenticationToken类型，则抛出异常  
4      Assert.isInstanceOf(UsernamePasswordAuthenticationToken.class, authentication,  
5          () -> messages.getMessage(  
6              "AbstractUserDetailsAuthenticationProvider.onlySupports",  
7              "Only UsernamePasswordAuthenticationToken is supported"));  
8  
9      // 获取用户名  
10     String username = (authentication.getPrincipal() == null) ? "NONE_PROVIDED"  
11         : authentication.getName();  
12  
13     //从缓存中获取UserDetails  
14     boolean cacheWasUsed = true;  
15     UserDetails user = this.userCache.getUserFromCache(username);  
16  
17     //当缓存中没有UserDetails，则从子类DaoAuthenticationProvider中获取  
18     if (user == null) {  
19         cacheWasUsed = false;  
20  
21         try {  
22             //子类DaoAuthenticationProvider中实现获取用户信息,  
23             // 就是调用对应UserDetailsService#loadUserByUsername  
24             user = retrieveUser(username,  
25                 (UsernamePasswordAuthenticationToken) authentication);  
26         }  
27         catch (UsernameNotFoundException notFound) {  
28             ....  
29         }  
30         ...  
31     }  
32  
33     try {  
34         //前置检查。DefaultPreAuthenticationChecks 检测帐户是否锁定，是否可用，是否过期  
35         preAuthenticationChecks.check(user);  
36         // 检查密码是否正确  
37         additionalAuthenticationChecks(user,  
38             (UsernamePasswordAuthenticationToken) authentication);  
39     }  
40     catch (AuthenticationException exception) {  
41         // 异常则是重新认证  
42         if (cacheWasUsed) {  
43             cacheWasUsed = false;
```

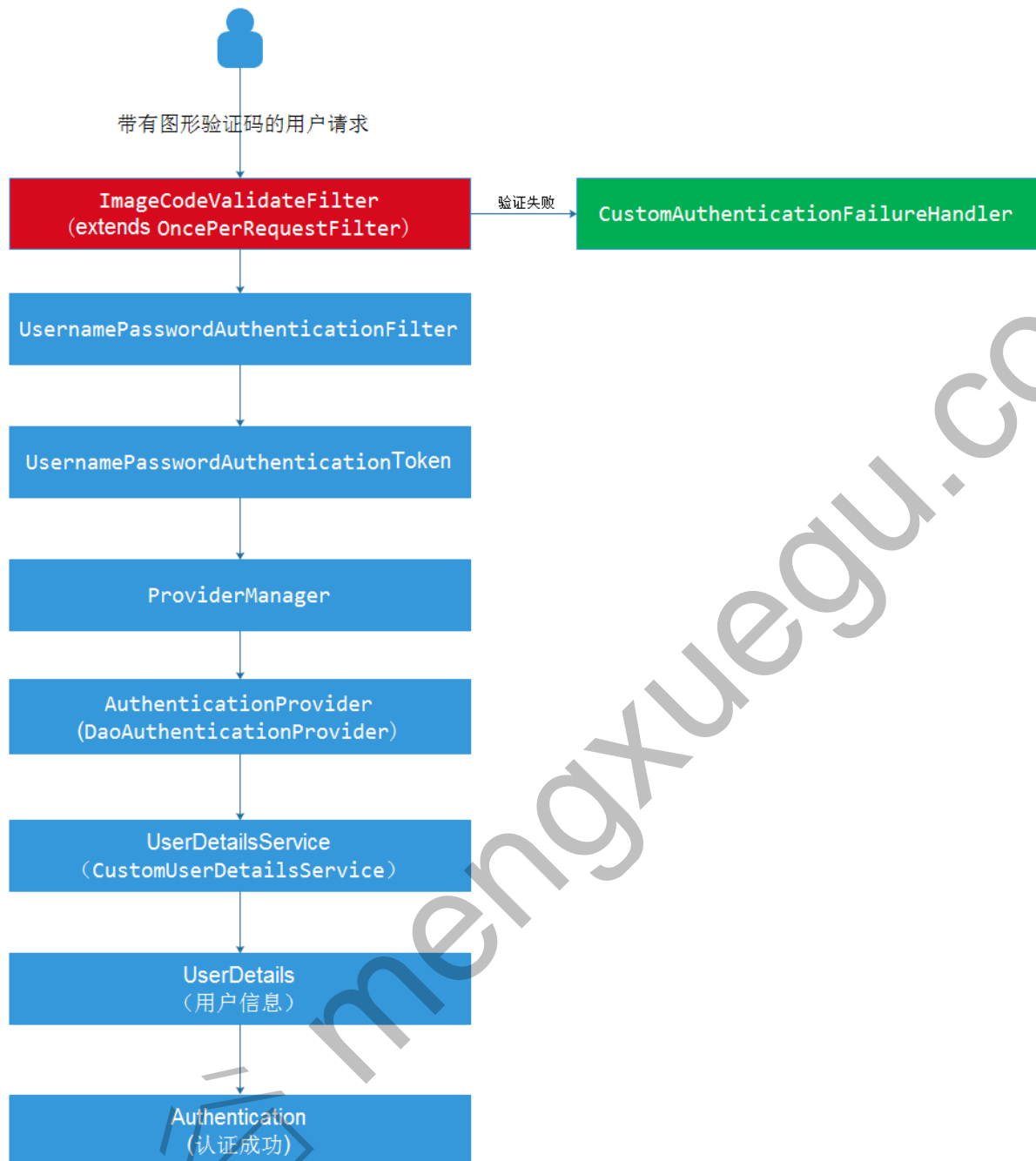


```
44      // 调用 loadUserByUsername 查询登录用户信息
45      user = retrieveUser(username,
46          (UsernamePasswordAuthenticationToken) authentication);
47      preAuthenticationChecks.check(user);
48      additionalAuthenticationChecks(user,
49          (UsernamePasswordAuthenticationToken) authentication);
50  }
51  else {
52      throw exception;
53  }
54  }
55  //后检查。由DefaultPostAuthenticationChecks实现(检测密码是否过期)
56  postAuthenticationChecks.check(user);
57  if (! cacheWasUsed) { //是否放到缓存中
58      this.userCache.putUserInCache(user);
59  }
60  Object principalToReturn = user;
61  if (forcePrincipalAsString) {
62      principalToReturn = user.getUsername();
63  }
64  //将认证成功用户信息封装成 UsernamePasswordAuthenticationToken 对象并返回
65  return createSuccessAuthentication(principalToReturn, authentication, user);
66  }
```

第六章 Spring Security 个性化认证实战

6.1 图形验证码认证功能

6.1.1 分析实现流程



6.1.2 生成一张图形验证码

Kaptcha 是谷歌提供的一个生成图形验证码的 jar 包, 只要简单配置属性就可以生成。

参考：<https://github.com/penggle/kaptcha>

1. 添加 Kaptcha 依赖，在 mengxuegu-security-core\pom.xml 中

```
1 <!--短信验证码-->
2 <dependency>
3   <groupId>com.github.penggle</groupId>
4   <artifactId>kaptcha</artifactId>
5 </dependency>
```

2. 生成验证码配置类，在 mengxuegu-security-core 模块中创建

com.mengxuegu.security.authentication.code.KaptchaImageCodeConfig

可直接拷贝 02-配套资料\KaptchaImageCodeConfig.java

```
1 package com.mengxuegu.security.authentication.code;
2
3 import com.google.code.kaptcha.Constants;
4 import com.google.code.kaptcha.impl.DefaultKaptcha;
5 import com.google.code.kaptcha.util.Config;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 import java.util.Properties;
10
11 /**
12  * 生成验证码配置类
13  * @Author: 梦学谷 www.mengxuegu.com
14  */
15 @Configuration
16 public class KaptchaImageCodeConfig {
17
18     @Bean
19     public DefaultKaptcha getDefaultKaptcha(){
20         DefaultKaptcha defaultKaptcha = new DefaultKaptcha();
21         Properties properties = new Properties();
22         properties.setProperty(Constants.KAPTCHA_BORDER, "yes");
23         properties.setProperty(Constants.KAPTCHA_BORDER_COLOR, "192,192,192");
24         properties.setProperty(Constants.KAPTCHA_IMAGE_WIDTH, "110");
25         properties.setProperty(Constants.KAPTCHA_IMAGE_HEIGHT, "36");
26         properties.setProperty(Constants.KAPTCHA_TEXTPRODUCER_FONT_COLOR, "blue");
27         properties.setProperty(Constants.KAPTCHA_TEXTPRODUCER_FONT_SIZE, "28");
28         properties.setProperty(Constants.KAPTCHA_TEXTPRODUCER_FONT_NAMES, "宋体");
29         properties.setProperty(Constants.KAPTCHA_TEXTPRODUCER_CHAR_LENGTH, "4");
30         // 图片效果
31         properties.setProperty(Constants.KAPTCHA_OBSCURIFICATOR_IMPL,
32             "com.google.code.kaptcha.impl.ShadowGimpy");
33         Config config = new Config(properties);
34         defaultKaptcha.setConfig(config);
35         return defaultKaptcha;
36     }
37 }
```

3. 在 CustomLoginController 提供请求接口，将验证码图片数据流写出

```
1 package com.mengxuegu.security.controller;
2
3 import com.google.code.kaptcha.impl.DefaultKaptcha;
4 import org.slf4j.Logger;
5 import org.slf4j.LoggerFactory;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import javax.imageio.ImageIO;
10 import javax.servlet.ServletOutputStream;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 import java.awt.image.BufferedImage;
14 import java.io.IOException;
15
16 /**
17  * @Author: 梦学谷 www.mengxuegu.com
18  */
19 @Controller
20 public class CustomLoginController {
21     Logger logger = LoggerFactory.getLogger(getClass());
22     public static final String SESSION_KEY = "SESSION_KEY_IMAGE_CODE";
23
24     /**
25      * 前往认证(登录)页面
26      * @return
27      */
28     @RequestMapping("/login/page")
29     public String toLogin() {
30         return "login"; // classpath: /templates/login.html
31     }
32
33     @Autowired
34     private DefaultKaptcha defaultKaptcha;
35
36     /**
37      * 获取图形验证码
38      */
39     @RequestMapping("/code/image")
40     public void imageCode(HttpServletRequest request, HttpServletResponse response) throws
41         IOException {
42         // 1. 获取验证码字符串
43         String code = defaultKaptcha.createText();
44         logger.info("生成的图形验证码是：" + code);
45         // 2. 字符串把它放到session中
46         request.getSession().setAttribute(SESSION_KEY, code);
47         // 3. 获取验证码图片
48         BufferedImage image = defaultKaptcha.createImage(code);
49         // 4. 将验证码图片把它写出去
50         ServletOutputStream out = response.getOutputStream();
51         ImageIO.write(image, "jpg", out);
52     }
53 }
```

```
51  
52 }
```

4. 在 `SpringSecurityConfig.configure(HttpSecurity http)` 放行 `/code/image` 资源权限

```
1 .antMatchers(securityProperties.getAuthentication().getLoginPage(),  
2     "/code/image").permitAll()
```

5. 重构 `mengxuegu-security-web` 模块的 `login.html` 页面，调用验证码接口渲染图片

```
1 <div class="row mb-2 ">  
2   <div class="col-6">  
3     <input name="code" type="text" class="form-control" placeholder="验证码">  
4   </div>  
5   <div class="col-6">  
6       
7   </div>  
8 </div>
```

6. 效果

梦学谷统一权限管理

请登录

☒ 使用短信验证登录

用户名

密码

验证码

☐ 记住我

登录

[忘记密码](#)

[注册帐号](#)

kaptcha参数说明

Constant	描述	默认值
kaptcha.border	图片边框，合法值：yes, no	yes
kaptcha.border.color	边框颜色，合法值：r,g,b (and optional alpha) 或者 white,black,blue.	black
kaptcha.border.thickness	边框厚度，合法值：>0	1
kaptcha.image.width	图片宽	200
kaptcha.image.height	图片高	50
kaptcha.producer.impl	图片实现类	com.google.code.kaptcha.impl.DefaultKaptcha
kaptcha.textproducer.impl	文本实现类	com.google.code.kaptcha.text.impl.DefaultTextCreator
kaptcha.textproducer.char.string	文本集合，验证码值从此集合中获取	abcde2345678gfynmnpwx
kaptcha.textproducer.char.length	验证码长度	5
kaptcha.textproducer.font.names	字体	Arial, Courier
kaptcha.textproducer.font.size	字体大小	40px.
kaptcha.textproducer.font.color	字体颜色，合法值：r,g,b 或者 white,black,blue.	black
kaptcha.textproducer.char.space	文字间隔	2
kaptcha.noise.impl	干扰实现类	com.google.code.kaptcha.impl.DefaultNoise
kaptcha.noise.color	干扰线颜色，合法值：r,g,b 或者 white,black,blue.	black
kaptcha.obscurificator.impl	图片样式：水纹com.google.code.kaptcha.impl.WaterRipple 鱼眼 com.google.code.kaptcha.impl.FishEyeGimpy 阴影 com.google.code.kaptcha.impl.ShadowGimpy	com.google.code.kaptcha.impl.WaterRipple
kaptcha.background.impl	背景实现类	com.google.code.kaptcha.impl.DefaultBackground
kaptcha.background.clear.from	背景颜色渐变，开始颜色	light grey
kaptcha.background.clear.to	背景颜色渐变，结束颜色	white
kaptcha.word.impl	文字渲染器	com.google.code.kaptcha.text.impl.DefaultWordRenderer
kaptcha.session.key	session key	KAPTCHA_SESSION_KEY
kaptcha.session.date	session date	KAPTCHA_SESSION_DATE

6.1.3 实现验证码校验过滤器

1. 创建 `com.mengxuegu.security.authentication.code.ImageCodeValidateFilter`，继承 `OncePerRequestFilter`（在所有请求前都被调用一次）
2. 在类上加上注解 `@Component("imageCodeValidateFilter")`
3. 如果是登录请求（请求地址：`/login/form`，请求方式：`post`），校验验证码输入是否正确
 - 校验不合法时，提示信息通过自定义异常 `ValidateCodeExcetipn` 抛出，此异常要继承 `org.springframework.security.core.AuthenticationException`，它是认证的父异常类。
 - 捕获 `ImageCodeException` 异常，交给失败处理器 `CustomAuthenticationFailureHandler`。
4. 如果非登录请求，则放行请求 `filterChain.doFilter(request, response)`

```

1 package com.mengxuegu.security.authentication.code;
2
3 import com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler;
4 import com.mengxuegu.security.authentication.exception.ValidateCodeExcetipn;
5 import com.mengxuegu.security.controller.CustomLoginController;
6 import com.mengxuegu.security.properites.SecurityProperties;
7 import org.apache.commons.lang.StringUtils;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.security.core.AuthenticationException;
10 import org.springframework.stereotype.Component;
  
```

```
11 import org.springframework.web.filter.OncePerRequestFilter;
12
13 import javax.servlet.FilterChain;
14 import javax.servlet.ServletException;
15 import javax.servlet.http.HttpServletRequest;
16 import javax.servlet.http.HttpServletResponse;
17 import java.io.IOException;
18
19 /**
20  * OncePerRequestFilter: 所有请求之前被调用一次
21  * @Author: 梦学谷 www.mengxuegu.com
22  */
23 @Component("imageCodeValidateFilter")
24 public class ImageCodeValidateFilter extends OncePerRequestFilter {
25
26     @Autowired
27     SecurityProperties securityProperties;
28
29     @Autowired
30     CustomAuthenticationFailureHandler customAuthenticationFailureHandler;
31     @Override
32     protected void doFilterInternal(HttpServletRequest request,
33         HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
34         // 1. 如果是post方式的登录请求，则校验输入的验证码是否正确
35         if(securityProperties.getAuthentication().getLoginProcessingUrl()
36             .equals(request.getRequestURI())
37             && request.getMethod().equalsIgnoreCase("post")) {
38             try {
39                 // 校验验证码合法性
40                 validate(request);
41             } catch (AuthenticationException e) {
42                 // 交给失败处理器进行处理异常
43                 customAuthenticationFailureHandler.onAuthenticationFailure(request, response, e);
44                 // 一定要记得结束
45                 return;
46             }
47         }
48
49         // 放行请求
50         filterChain.doFilter(request, response);
51     }
52
53     private void validate(HttpServletRequest request) {
54         // 先获取session中的验证码
55         String sessionCode = (String)request.getSession().getAttribute(CustomLoginController.SESSION_KEY);
56         // 获取用户输入的验证码
57         String inputCode = request.getParameter("code");
58         // 判断是否正确
59         if(StringUtils.isBlank(inputCode)) {
60             throw new ValidateCodeException("验证码不能为空");
61         }
62
63         if(inputCode.equalsIgnoreCase(sessionCode)) {
```



```
64         throw new ValidateCodeException("验证码输入错误");
65     }
66 }
67 }
```

6.1.4 创建验证码异常类

创建 `com.mengxuegu.security.authentication.exception.ValidateCodeExcetipn` 异常类，它继

承 `AuthenticationException`

特别注意是：`org.springframework.security.core.AuthenticationException`

视频中讲的是在 `.authentication.code` 包下，后面会把它移动 `.authentication.exception` 包

```
1 package com.mengxuegu.security.authentication.exception;
2
3 import org.springframework.security.core.AuthenticationException;
4
5 /**
6  * @Author: 梦学谷 www.mengxuegu.com
7  */
8 public class ValidateCodeExcetipn extends AuthenticationException {
9     public ValidateCodeExcetipn(String msg, Throwable t) {
10         super(msg, t);
11     }
12
13     public ValidateCodeExcetipn(String msg) {
14         super(msg);
15     }
16 }
```

6.1.5 重构 SpringSecurityConfig

将校验过滤器 `imageCodeValidateFilter` 添加到 `UsernamePasswordAuthenticationFilter` 前面

在 `com.mengxuegu.security.config.SpringSecurityConfig` 中完成以下操作：

1. 注入 `ImageCodeValidateFilter` 实例

```
1 // 验证码校验过滤器
2 @Autowired
3 ImageCodeValidateFilter imageCodeValidateFilter;
```

2. 把 `ImageCodeValidateFilter` 添加 `UsernamePasswordAuthenticationFilter` 实例前

```
1 http.addFilterBefore(imageCodeValidateFilter,  
2     UsernamePasswordAuthenticationFilter.class)  
3     .formLogin()
```

3. SpringSecurityConfig 完整代码

```
1 package com.mengxuegu.security.config;  
2  
3 import com.mengxuegu.security.authentication.code.ImageCodeValidateFilter;  
4 import com.mengxuegu.security.properties.SecurityProperties;  
5 import org.slf4j.Logger;  
6 import org.slf4j.LoggerFactory;  
7 import org.springframework.beans.factory.annotation.Autowired;  
8 import org.springframework.context.annotation.Bean;  
9 import org.springframework.context.annotation.Configuration;  
10 import  
    org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder  
    ;  
11 import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
12 import org.springframework.security.config.annotation.web.builders.WebSecurity;  
13 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
14 import  
    org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
15 import org.springframework.security.core.userdetails.UserDetailsService;  
16 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
17 import org.springframework.security.crypto.password.PasswordEncoder;  
18 import org.springframework.security.web.authentication.AuthenticationFailureHandler;  
19 import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  
20 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
21  
22  
23 /**  
24  * 安全控制中心  
25  * @Author: 梦学谷 www.mengxuegu.com  
26  */  
27 @Configuration  
28 @EnableWebSecurity //启动 SpringSecurity 过滤器链功能  
29 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {  
30  
31     @Autowired  
32     SecurityProperties securityProperties;  
33  
34     @Bean  
35     public PasswordEncoder passwordEncoder() {  
36         // 设置默认的加密方式  
37         return new BCryptPasswordEncoder();  
38     }  
39  
40     // 报红线没关系，idea识别问题  
41     @Autowired  
42     private UserDetailsService customUserDetailsService;  
43
```

```
44 // 注入自定义的认证成功处理器
45 @Autowired
46 private AuthenticationSuccessHandler customAuthenticationSuccessHandler;
47
48 @Autowired
49 private AuthenticationFailureHandler customAuthenticationFailureHandler;
50
51 // 验证码校验过滤器
52 @Autowired
53 private ImageCodeValidateFilter imageCodeValidateFilter;
54
55 /**
56  * 认证管理器：
57  * 1、认证信息提供方式（用户名、密码、当前用户的资源权限）
58  * 2、可采用内存存储方式，也可能采用数据库方式等
59  * @param auth
60  * @throws Exception
61  */
62 @Override
63 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
64     // 用户信息存储在内存中
65     // String password = passwordEncoder().encode("1234");
66     // logger.info("加密之后存储的密码：" + password);
67     // auth.inMemoryAuthentication().withUser("mengxuegu")
68     //     .password(password).authorities("ADMIN");
69     auth.userDetailsService(customUserDetailsService);
70 }
71
72 /**
73  * 资源权限配置（过滤器链）：
74  * 1、被拦截的资源
75  * 2、资源所对应的角色权限
76  * 3、定义认证方式：httpBasic、httpForm
77  * 4、定制登录页面、登录请求地址、错误处理方式
78  * 5、自定义 spring security 过滤器
79  * @param http
80  * @throws Exception
81  */
82 @Override
83 protected void configure(HttpSecurity http) throws Exception {
84     // http.httpBasic()
85     http.addFilterBefore(imageCodeValidateFilter, UsernamePasswordAuthenticationFilter.class)
86         .formLogin() // 表单认证
87         .loginPage(securityProperties.getAuthentication().getLoginPage())
88         .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl())
89         .usernameParameter(securityProperties.getAuthentication().getUsernameParameter())
90         .passwordParameter(securityProperties.getAuthentication().getPasswordParameter())
91         .successHandler(customAuthenticationSuccessHandler) //认证成功处理器
92         .failureHandler(customAuthenticationFailureHandler) //认证失败处理器
93         .and()
94         .authorizeRequests() // 认证请求
95         .antMatchers(
```

```
96         securityProperties.getAuthentication().getLoginPage(),
97         "/code/image"
98     ).permitAll()
99     .anyRequest().authenticated() // 所有进入应用的HTTP请求都要进行认证
100 ; // 分号`不要少了
101
102 }
103
104 /**
105  * 释放静态资源
106  * @param web
107  */
108 @Override
109 public void configure(WebSecurity web) {
110     web.ignoring().antMatchers(
111         securityProperties.getAuthentication().getStaticPaths());
112 }
113 }
```

6.1.6 测试验证码认证效果



6.1.7 解决页面不回显提示信息

问题

如果验证码输入错误时，页面不回显提示信息，在控制台打印了日志信息

```
com.mengxuegu.security.authentication.exception.ValidateCodeExcetipn: 验证码输入错误
at com.mengxuegu.security.authentication.code.ImageCodeValidateFilter.validate(ImageCodeValidat
at com.mengxuegu.security.authentication.code.ImageCodeValidateFilter.doFilterInternal(ImageCod
at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainPro
```

解决

确保 `ValidateCodeExcetipn` 和 `ImageCodeValidateFilter` 类中：

AuthenticationException不要导错包了，是org.springframework.security.core.AuthenticationException

```
ValidateCodeExcetipn.java
1  mengxuegu.security.code.exception;
2
3
4  ringframework.security.core.AuthenticationException;
5

ImageCodeValidateFilter.java
4  gxuegu.security.code.exception.ValidateCodeExcetipn;
5  gxuegu.security.controller.CustomLoginController;
6  che.commons.lang.StringUtils;
7  ingframework.beans.factory.annotation.Autowired;
8  ingframework.security.core.AuthenticationException;
```

6.2 Remember-Me 记住我功能

效果: 登录后会记住用户令牌，不用反复登录。

6.2.1 分析 Remember-Me 实现流程

1. 用户选择了“记住我”成功登录后，将会把username、随机生成的序列号、生成的token存入一个数据库表中，同时将它们的组合生成一个cookie发送给客户端浏览器。
2. 当没有登录的用户访问系统时，首先检查 remember-me 的 cookie 值，有则检查其值包含的 username、序列号和 token 与数据库中是否一致，一致则通过验证。
并且系统还会重新生成一个新的 token 替换数据库中对应旧的 token，序列号 series 保持不变，同时删除旧的 cookie，重新生成 cookie 值（新的 token + 旧的序列号 + username）发送给客户端。
3. 如果对应cookie不存在，或者包含的username、序列号和token 与数据库中保存的不一致，那么将会引导用户到登录页面。
因为cookie被盗用后还可以在用户下一次登录前顺利的进行登录，所以如果你的应用对安全性要求比较高就不要使用Remember-Me功能。

6.2.2 实现用户名密码 Remember-Me 功能

1. mengxuegu-security-core 的 pom.xml 引入依赖

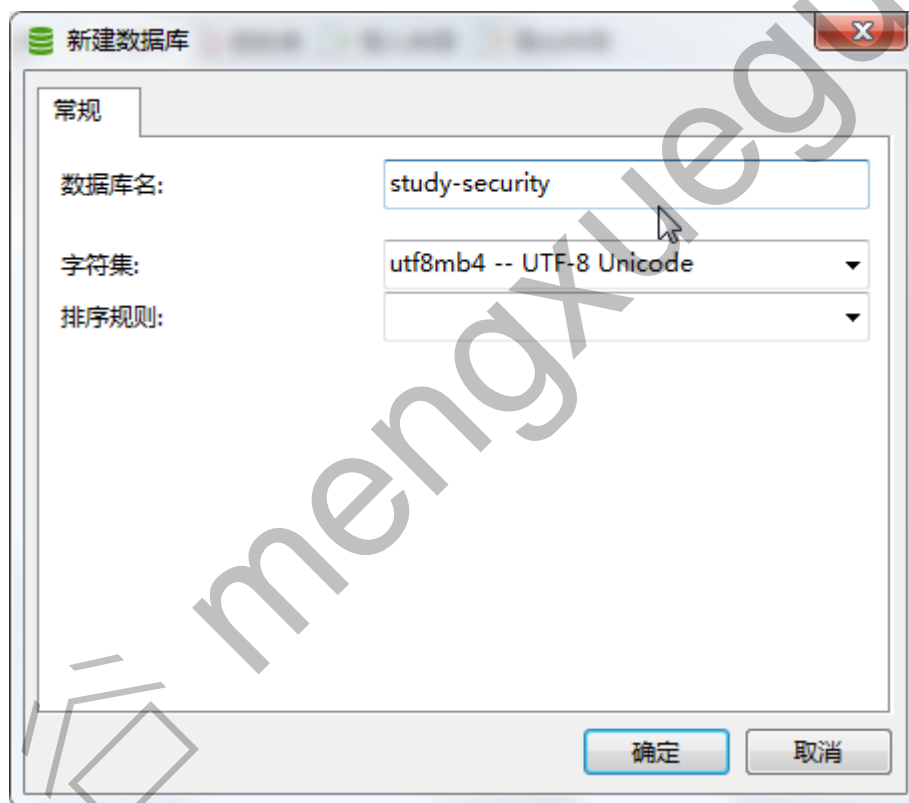
```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-jdbc</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>mysql</groupId>
7   <artifactId>mysql-connector-java</artifactId>
8 </dependency>
```

2. mengxuegu-security-web 的 application.yml 配置数据源,

注意： 不要复制当前pdf文档的，复制 01-课堂源码\07-第6.2章节RememberMe代码 目录下的

```
1 spring
2 thymeleaf:
3   cache: false #关闭thymeleaf缓存
4 # 数据源配置
5 datasource:
6   username: root
7   password: root
8   url: jdbc:mysql://127.0.0.1:3306/study-security?
serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8
9   #mysql8版本以上驱动包指定新的驱动类
10  driver-class-name com.mysql.cj.jdbc.Driver
```

3. 在数据库中创建一个 study-security



4. 使用 JdbcTokenRepository 实现类

```
1 ...省略
2 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
3   ...省略
4
5   // 记住我功能
6   @Autowired
7   DataSource dataSource;
8   @Bean
9   public JdbcTokenRepositoryImpl jdbcTokenRepository() {
10     JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
11     jdbcTokenRepository.setDataSource(dataSource);
12
13     // 是否启动时自动创建表，第一次启动创建就行，后面启动把这个注释掉,不然报错已存在表
```

```
13 // jdbcTokenRepository.setCreateTableOnStartup(true);
14 return jdbcTokenRepository;
15 }
16
17 ...省略
18 }
```

5. 安全配置 SpringSecurityConfig#configure(HttpSecurity http)

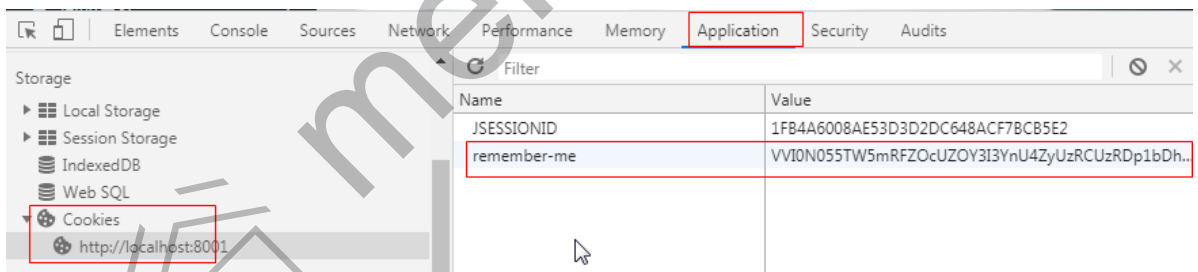
```
1 .and()
2 .rememberMe() //记住我
3 .tokenRepository(jdbcTokenRepository()) // 保存登录信息
4 .tokenValiditySeconds(60*60*24*7) // 记住我有效时长（秒）
```

6. 修改mengxuegu-security-web 模块中的 login.html 记住我的 name="remember-me"

```
1 <input type="checkbox" name="remember-me" > 记住我
```

7. 测试效果

1. 重启，数据库会自动创建表 persistent_logins
2. 访问首页跳转到登录页面，登录成功后，查看浏览器 cookies，有保存 token 值。
数据库中也有数据。



3. 再重启，Session会把清除。再次访问首页，不会跳转到登录页，因为上次已经记录了。这时浏览器会带着Cookie中保存的token从数据库查找用户名，然后进行自动登录。

8. 报错:

Caused by: java.sql.SQLException: Table 'persistent_logins' already exists

解决方法：

将下面启动服务时开启创建表注释掉

```
1 // jdbcTokenRepository.setCreateTableOnStartup(true);
```


6.2.3 分析 Remember-Me 底层源码实现

1571648816697

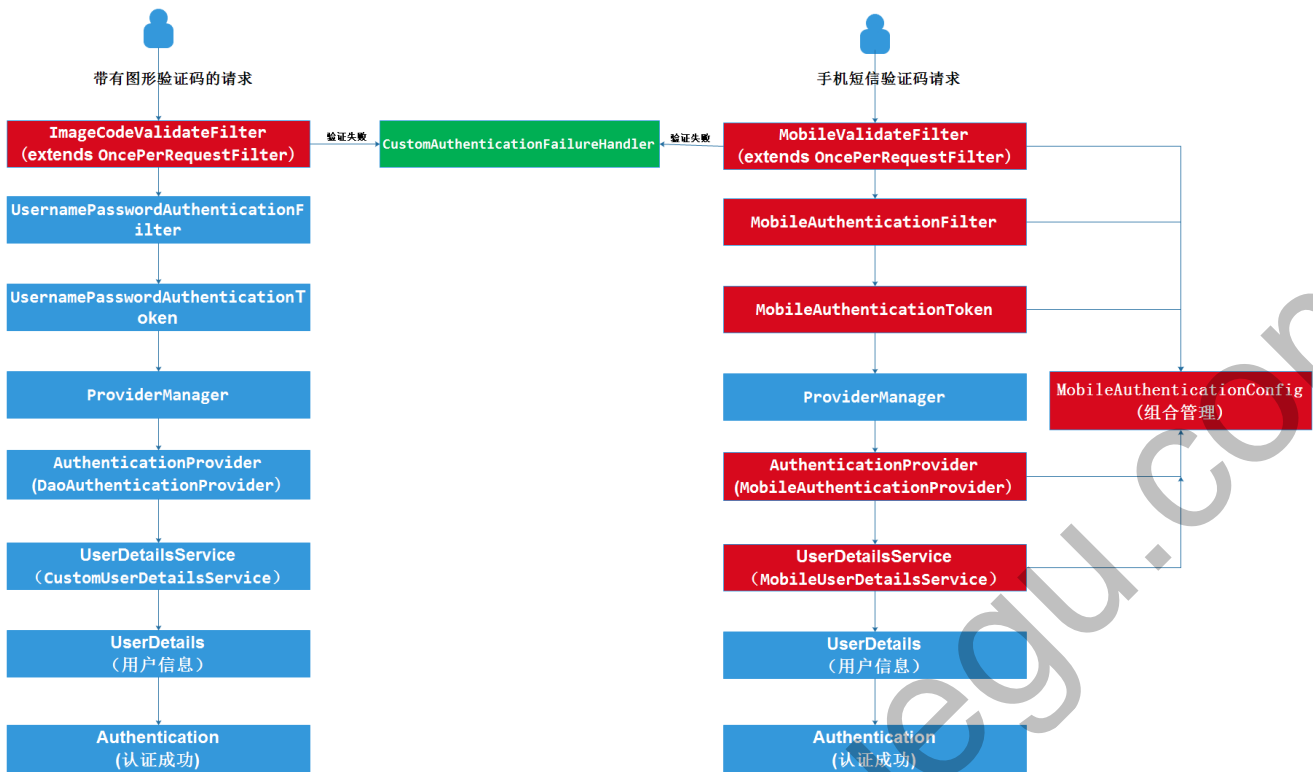
- `UsernamePasswordAuthenticationFilter` 拥有一个 `RememberMeServices` 的引用，默认是一个空实现的 `NullRememberMeServices`，而实际当我们通过 `rememberMe()` 启用 Remember-Me 时，它是一个具体的实现。
- 用户的请求会先通过 `UsernamePasswordAuthenticationFilter`，当认证成功后会调用 `RememberMeServices` 的 `loginSuccess()` 方法，否则调用 `RememberMeServices` 的 `loginFail()` 方法。
`UsernamePasswordAuthenticationFilter` 不会调用 `RememberMeServices` 的 `autoLogin()` 方法进行自动登录的。
- 当执行到 `RememberMeAuthenticationFilter` 时，如果检测到还没有认证成功时，那么 `RememberMeAuthenticationFilter` 会尝试着调用所包含的 `RememberMeServices` 的 `autoLogin()` 方法进行自动登录。
- `PersistentTokenBasedRememberMeServices` 是 `RememberMeServices` 的启动 Remember-Me 默认实现，它会通过 cookie 值进行查询数据库存储的记录，来实现自动登录，并重新生成新的 cookie 存储。

6.3 手机短信验证码认证功能

6.3.1 分析实现流程

手机号登录是不需要密码的，通过短信验证码实现免密登录功能。

1. 向手机发送手机验证码，使用第三方短信平台 SDK 发送，如：阿里云短信服务（阿里大于）
2. 登录表单输入短信验证码
3. 使用自定义过滤器 `MobileValidateFilter`
4. 当验证码校验通过后，进入自定义手机认证过滤器 `MobileAuthenticationFilter` 校验手机号是否存在
5. 自定义 `MobileAuthenticationToken` 提供给 `MobileAuthenticationFilter`
6. 自定义 `MobileAuthenticationProvider` 提供给 `ProviderManager` 处理
7. 创建针对手机号查询用户信息的 `MobileUserDetailsService`，交给 `MobileAuthenticationProvider`
8. 自定义 `MobileAuthenticationConfig` 配置类将上面组件连接起来，添加到容器中
9. 将 `MobileAuthenticationConfig` 添加到 `SpringSecurityConfig` 安全配置的过滤器链上。



6.2.2 创建短信发送服务接口

1. 定义短信发送服务接口 `com.mengxuegu.security.authentication.mobile.SmsSend`

```
1 package com.mengxuegu.security.authentication.mobile;
2 /**
3  * @Author: 梦学谷 www.mengxuegu.com
4  */
5 public interface SmsSend {
6     boolean sendSms(String mobile, String content);
7 }
```

2. 实现短信发送服务接口 `com.mengxuegu.security.authentication.mobile.SmsCodeSender`

```
1 package com.mengxuegu.security.authentication.mobile;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.stereotype.Component;
6
7 /**
8  * 发送短信验证码的实现
9  * @Author: 梦学谷 www.mengxuegu.com
10 */
11 public class SmsCodeSender implements SmsSend {
12     Logger logger = LoggerFactory.getLogger(getClass());
13 }
```

```
14  @Override
15  public boolean sendSms(String mobile, String content) {
16      String sendContent = String.format("梦学员，验证码%s，请勿泄露他人。", content);
17      logger.info("向手机号" + mobile + "发送的短信为:" + sendContent);
18      return true;
19  }
20
21 }
```

3. 在 `com.mengxuegu.security.config` 创建 `SeurityConfigBean` 将 `SmsCodeSender` 添加到Spring容器。

也可以直接在类上加上 `@Component` 注解，但是不利于应用的扩展，因为短信服务提供商有非常多，实现就不一样，所以采用下面方式添加到容器：

```
1  @Configuration
2  public class SeurityConfigBean{
3
4      /**
5       * @ConditionalOnMissingBean(SmsSend.class)
6       * 默认采用SmsCodeSender实例，但如果容器中有其他 SmsSend 类型的实例，则当前实例失效
7       */
8      @Bean
9      @ConditionalOnMissingBean(SmsSend.class)
10     public SmsSend smsSend(){
11         return new SmsCodeSender();
12     }
13 }
14
```

4. 在 `mengxuegu-security-web` 中创建一个 `MobileSmsCodeSender` 短信服务接口的实现，来替换默认实现 `SmsCodeSender`，后面再进行测试这个接口。

```
1  package com.mengxuegu.security;
2
3  import com.mengxuegu.security.authentication.mobile.SmsSend;
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6
7  /**
8   * @Author: 梦学谷 www.mengxuegu.com
9   */
10 // @Component
11 public class MobileSmsCodeSender implements SmsSend {
12     Logger logger = LoggerFactory.getLogger(getClass());
13
14     @Override
15     public boolean sendSms(String mobile, String content) {
16         logger.info("Web应用新的短信验证码接口---向手机号"+mobile+"发送了验证码为：" + content);
17         return false;
18     }
19 }
20
```

6.2.3 手机登录页与发送短信验证码

1. 创建 `com.mengxuegu.security.controller.CustomMobileController` 添加如下代码:

```
1 package com.mengxuegu.security.controller;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import com.mengxuegu.security.authentication.mobile.SmsSend;
5 import org.apache.commons.lang.RandomStringUtils;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.ResponseBody;
10
11 import javax.servlet.http.HttpServletRequest;
12
13 /**
14  * 关于手机登录控制层
15  * @Author: 梦学谷 www.mengxuegu.com
16  */
17 @Controller
18 public class MobileLoginController {
19
20     public static final String SESSION_KEY = "SESSION_KEY_MOBILE_CODE";
21
22     /**
23      * 前往手机验证码登录页
24      * @return
25      */
26     @RequestMapping("/mobile/page")
27     public String toMobilePage() {
28         return "login-mobile"; // templates/login-mobile.html
29     }
30
31     @Autowired
32     SmsSend smsSend;
33
34     /**
35      * 发送手机验证码
36      * @return
37      */
38     @ResponseBody //响应json字符串
39     @RequestMapping("/code/mobile")
40     public MengxueguResult smsCode(HttpServletRequest request) {
41         // 1. 生成一个手机验证码
42         String code = RandomStringUtils.randomNumeric(4);
43         // 2. 将手机验证码保存到session中
44         request.getSession().setAttribute(SESSION_KEY, code);
45
46         // 3. 发送验证码到用户手机上
```

```
46     String mobile = request.getParameter("mobile");
47     smsSend.sendSms(mobile, code);
48
49     return MengxueguResult.ok();
50 }
51 }
```

2. 在 **mengxuegu-security-web** 模块添加手机登录页面 `login-mobile.html` 到 `templates` 目录下

手机登录表单核心代码

```
th:action="@{/mobile/form}"
```

```
th:attr="code_url=@{/code/mobile?mobile=}"
```

```
1  <a th:href="@{/login/page}" href="login.html" style="float:right;font-size: 13px;">
2      <i class="fa fa-lock mr-1"></i>使用密码验证登录
3  </a>
4  <form th:action="@{/mobile/form}" action="index.html" method="post">
5      <div class="input-group mb-3">
6          <input id="mobile" name="mobile" type="text" class="form-control" placeholder="手机号码">
7          <div class="input-group-append">
8              <div class="input-group-text">
9                  <span class="fa fa-user"></span>
10             </div>
11         </div>
12     </div>
13
14     <div class="mb-3 row">
15         <div class="col-7">
16             <input type="text" name="code" class="form-control" placeholder="验证码">
17         </div>
18         <div class="col-5">
19             <a id="sendCode" th:attr="code_url=@{/code/mobile?mobile=}" class="btn btn-outline-primary
20             btn-large" href="#"> 获取验证码 </a>
21         </div>
22     </div>
23
24     <!-- 提示信息, 表达式红线没关系, 忽略它 -->
25     <div th:if="${param.error}">
26         <span th:text="${session.SPRING_SECURITY_LAST_EXCEPTION?.message}" style="color:red"></span>
27     </div>
28
29     <div class="row">
30         <div class="col-8">
31             <div class="icheck-primary">
32                 <input name="remember-me" type="checkbox" id="remember">
33                 <label for="remember">
34                     记住我
35                 </label>
36             </div>
37         </div>
38     </div>
```

```
39         <button type="submit" class="btn btn-primary btn-block">登录</button>
40     </div>
41     <!-- /.col -->
42 </div>
43
44 </form>
```

3. 在 SpringSecurityConfig 放行手机登录相关请求URL

```
1 .antMatchers(securityProperties.getAuthentication().getLoginPage(),
2     "/code/image", "/mobile/page", "/code/mobile").permitAll()
```

6.2.4 实现短信验证码校验过滤器 MobileValidateFilter

校验输入的验证码与发送的短信验证是否一致。

创建 `com.mengxuegu.security.authentication.mobile.MobileValidateFilter` ,

与图形验证码过滤器 `ImageCodeValidateFilter` 实现类似。

不要少了 `@Component`

```
1 package com.mengxuegu.security.authentication.mobile;
2
3 import com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler;
4 import com.mengxuegu.security.authentication.exception.ValidateCodeException;
5 import com.mengxuegu.security.controller.MobileLoginController;
6 import org.apache.commons.lang.StringUtils;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.security.core.AuthenticationException;
9 import org.springframework.stereotype.Component;
10 import org.springframework.web.filter.OncePerRequestFilter;
11 import javax.servlet.FilterChain;
12 import javax.servlet.ServletException;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15 import java.io.IOException;
16
17 /**
18  * 校验用户输入的手机验证码是否正确
19  * @Author: 梦学谷 www.mengxuegu.com
20  */
21 @Component // 不要少了
22 public class MobileValidateFilter extends OncePerRequestFilter {
23
24     @Autowired
25     CustomAuthenticationFailureHandler customAuthenticationFailureHandler;
26
27     @Override
28     protected void doFilterInternal(HttpServletRequest request,
```

```
29         HttpServletResponse response,
30         FilterChain filterChain) throws ServletException, IOException {
31     if("/mobile/form".equals(request.getRequestURI())
32     && "post".equalsIgnoreCase(request.getMethod())) {
33         try {
34             // 校验验证码合法性
35             validate(request);
36         } catch (AuthenticationException e) {
37             // 交给失败处理器进行处理异常
38             customAuthenticationFailureHandler.onAuthenticationFailure(request, response, e);
39             // 一定要记得结束
40             return;
41         }
42     }
43
44     // 非手机验证码登录，则直接放行
45     filterChain.doFilter(request, response);
46
47 }
48
49 private void validate(HttpServletRequest request) {
50     // 先获取session中的验证码
51     String sessionCode =
52         (String)request.getSession().getAttribute(MobileLoginController.SESSION_KEY);
53     // 获取用户输入的验证码
54     String inputCode = request.getParameter("code");
55     // 判断是否正确
56     if(StringUtils.isBlank(inputCode)) {
57         throw new ValidateCodeException("短信验证码不能为空");
58     }
59
60     if(inputCode.equalsIgnoreCase(sessionCode)) {
61         throw new ValidateCodeException("短信验证码输入错误");
62     }
63 }
64 }
```

6.2.5 实现手机认证过滤器 MobileAuthenticationFilter

创建 `com.mengxuegu.security.authentication.mobile.MobileAuthenticationFilter` ,

模仿 `UsernamePasswordAuthenticationFilter` 代码进行改造

```
1 package com.mengxuegu.security.authentication.mobile;
2
3 import org.springframework.lang.Nullable;
4 import org.springframework.security.authentication.AuthenticationServiceException;
5 import org.springframework.security.core.Authentication;
6 import org.springframework.security.core.AuthenticationException;
7 import org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter;
8 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
```



```
9
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 /**
14  * @Author: 梦学谷 www.mengxuegu.com
15  */
16 public class MobileAuthenticationFilter extends AbstractAuthenticationProcessingFilter {
17
18
19     private String mobileParameter = "mobile";
20     private boolean postOnly = true;
21
22     public MobileAuthenticationFilter() {
23         super(new AntPathRequestMatcher("/mobile/form", "POST"));
24     }
25
26     public Authentication attemptAuthentication(HttpServletRequest request,
27                                             HttpServletResponse response) throws AuthenticationException {
28         if (postOnly && !request.getMethod().equals("POST")) {
29             throw new AuthenticationServiceException(
30                 "Authentication method not supported: " + request.getMethod());
31         }
32         // 从请求中获取手机号码, 来验证手机号是否有效
33         String mobile = obtainMobile(request);
34
35         if (mobile == null) {
36             mobile = "";
37         }
38
39         mobile = mobile.trim();
40
41         MobileAuthenticationToken authRequest = new MobileAuthenticationToken(mobile);
42
43         // Allow subclasses to set the "details" property
44         setDetails(request, authRequest);
45
46         return this.getAuthenticationManager().authenticate(authRequest);
47     }
48
49
50     /**
51     * 获取请求中的手机号码
52     */
53     @Nullable
54     protected String obtainMobile(HttpServletRequest request) {
55         return request.getParameter(mobileParameter);
56     }
57
58     /**
59     * 将请求中的 sessionId 和 host 主机ip放到 MobileAuthenticationToken
60     * set
61     */
62 }
```

```
62     protected void setDetails(HttpServletRequest request,
63                               MobileAuthenticationToken authRequest) {
64         authRequest.setDetails(authenticationDetailsSource.buildDetails(request));
65     }
66
67     public void setPostOnly(boolean postOnly) {
68         this.postOnly = postOnly;
69     }
70
71     public String getMobileParameter() {
72         return mobileParameter;
73     }
74
75     public void setMobileParameter(String mobileParameter) {
76         this.mobileParameter = mobileParameter;
77     }
78 }
```

6.2.6 封装手机认证Token MobileAuthenticationToken

创建 `com.mengxuegu.security.authentication.mobile.MobileAuthenticationToken`

提供给上面自定义的 `MobileAuthenticationFilter` 使用。

```
1  package com.mengxuegu.security.authentication.mobile;
2
3  import org.springframework.security.authentication.AbstractAuthenticationToken;
4  import org.springframework.security.core.GrantedAuthority;
5  import org.springframework.security.core.SpringSecurityCoreVersion;
6  import java.util.Collection;
7
8  /**
9   * @Author: 梦学谷 www.mengxuegu.com
10  */
11  public class MobileAuthenticationToken extends AbstractAuthenticationToken {
12
13      private static final long serialVersionUID = SpringSecurityCoreVersion.SERIAL_VERSION_UID;
14
15      // 认证之前放手机号，认证之后放用户信息
16      private final Object principal;
17
18      /**
19       * 开始认证时,创建一个MobileAuthenticationToken实例 接收的是手机号码, 并且 标识未认证
20       * @param principal 手机号
21       */
22      public MobileAuthenticationToken(Object principal) {
23          super(null);
24          this.principal = principal; // 手机号
25          setAuthenticated(false);
26      }
27
28 }
```

```
28  /**
29   * 当认证通过后,会重新创建一个新的MobileAuthenticationToken,来标识它已经认证通过,
30   * @param principal 用户信息
31   * @param authorities 用户权限
32   */
33  public MobileAuthenticationToken(Object principal, Collection<? extends GrantedAuthority> authorities) {
34      super(authorities);
35      this.principal = principal; // 用户信息
36      super.setAuthenticated(true); // 标识已经认证通过
37  }
38
39  /**
40   * 在父类中是一个抽象方法,所以要实现,但是它是密码,而当前不需要,则直接返回null
41   * @return
42   */
43  @Override
44  public Object getCredentials() {
45      return null;
46  }
47
48  /**
49   * 手机号码
50   * @return
51   */
52  public Object getPrincipal() {
53      return this.principal;
54  }
55
56  public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException {
57      if (isAuthenticated) {
58          throw new IllegalArgumentException(
59              "Cannot set this token to trusted - use constructor which takes a GrantedAuthority list instead");
60      }
61
62      super.setAuthenticated(false);
63  }
64
65  @Override
66  public void eraseCredentials() {
67      super.eraseCredentials();
68  }
69  }
```

6.2.7 实现手机认证提供者 MobileAuthProvider

创建 `com.mengxuegu.security.authentication.mobile.MobileAuthProvider` ,

提供给底层 ProviderManager 使用

```
1  package com.mengxuegu.security.authentication.mobile;
2
```

```
3 import org.springframework.security.authentication.AuthenticationProvider;
4 import org.springframework.security.authentication.AuthenticationServiceException;
5 import org.springframework.security.core.Authentication;
6 import org.springframework.security.core.AuthenticationException;
7 import org.springframework.security.core.userdetails.UserDetails;
8 import org.springframework.security.core.userdetails.UserDetailsService;
9
10 /**
11  * @Author: 梦学谷 www.mengxuegu.com
12  */
13
14 public class MobileAuthenticationProvider implements AuthenticationProvider {
15
16     UserDetailsService userDetailsService;
17
18     /**
19      * 处理认证:
20      * 1. 通过 手机号 去数据库查询用户信息(UserDeatilsService)
21      * 2. 再重新构建认证信息
22      * @param authentication
23      * @return
24      * @throws AuthenticationException
25      */
26     @Override
27     public Authentication authenticate(Authentication authentication) throws AuthenticationException {
28
29         MobileAuthenticationToken mobileAuthenticationToken
30             = (MobileAuthenticationToken)authentication;
31
32         // 获取用户输入的手机号
33         String mobile = (String)mobileAuthenticationToken.getPrincipal();
34         // 查询数据库
35         UserDetails userDetails = userDetailsService.loadUserByUsername(mobile);
36         if(userDetails == null) {
37             throw new AuthenticationServiceException("该手机未注册");
38         }
39
40         // 查询到了用户信息, 则认证通过, 就重新构建 MobileAuthenticationToken 实例
41         MobileAuthenticationToken authenticationToken =
42             new MobileAuthenticationToken(userDetails, userDetails.getAuthorities());
43
44         authenticationToken.setDetails(mobileAuthenticationToken.getDetails());
45         return authenticationToken;
46     }
47
48     /**
49      * 通过此方法, 来判断 采用哪一个 AuthenticationProvider
50      * @param authentication
51      * @return
52      */
53     @Override
54     public boolean supports(Class<?> authentication) {
55
56         return MobileAuthenticationToken.class.isAssignableFrom(authentication);
57     }
58 }
```

```
56     }
57
58     /**
59     * 注入 MobileUserDetailsService
60     * @param userDetailsService
61     */
62     public void setUserDetailsService(UserDetailsService userDetailsService) {
63         this.userDetailsService = userDetailsService;
64     }
65 }
```

6.2.8 手机号获取用户信息 MobileUserDetailsService

在 mengxuegu-security-web 模块中创建 UserDetailsService 实现类：

com.mengxuegu.security.MobileUserDetailsService

注意：不要注入 PasswordEncoder

```
1  package com.mengxuegu.security;
2
3  import org.slf4j.Logger;
4  import org.slf4j.LoggerFactory;
5  import org.springframework.security.core.authority.AuthorityUtils;
6  import org.springframework.security.core.userdetails.User;
7  import org.springframework.security.core.userdetails.UserDetails;
8  import org.springframework.security.core.userdetails.UserDetailsService;
9  import org.springframework.security.core.userdetails.UsernameNotFoundException;
10 import org.springframework.stereotype.Component;
11
12
13 /**
14 * 通过 手机号查询用户信息
15 * @Author: 梦学谷 www.mengxuegu.com
16 */
17 @Component("mobileUserDetailsService")
18 public class MobileUserDetailsService implements UserDetailsService {
19     Logger logger = LoggerFactory.getLogger(getClass());
20
21     @Override
22     public UserDetails loadUserByUsername(String mobile) throws UsernameNotFoundException {
23         logger.info("请求的手机号是：" + mobile);
24         // 1. 通过手机号查询用户信息
25
26         // 2. 如果有此用户，则查询用户权限
27         // 3. 封装用户信息
28         return new User(mobile, "",
29             true, true, true, true,
30             AuthorityUtils.commaSeparatedStringToAuthorityList("ADMIN"));
31     }
32 }
```

6.2.9 自定义管理认证配置 MobileAuthenticationConfig

创建 com.mengxuegu.security.authentication.mobile.MobileAuthenticationConfig 类

将上面定义的组件绑定起来，添加到容器中：

```
1 package com.mengxuegu.security.authentication.mobile;
2
3 import com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler;
4 import com.mengxuegu.security.authentication.CustomAuthenticationSuccessHandler;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.security.authentication.AuthenticationManager;
7 import org.springframework.security.config.annotation.SecurityConfigurerAdapter;
8 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import org.springframework.security.web.DefaultSecurityFilterChain;
11 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
12 import org.springframework.stereotype.Component;
13
14 /**
15  * 将定义的手机短信认证相关的组件组合起来，一起添加到容器中
16  * @Author: 梦学谷 www.mengxuegu.com
17  */
18 @Component
19 public class MobileAuthenticationConfig
20     extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {
21
22     @Autowired
23     CustomAuthenticationSuccessHandler customAuthenticationSuccessHandler;
24
25     @Autowired
26     CustomAuthenticationFailureHandler customAuthenticationFailureHandler;
27
28     @Autowired
29     UserDetailsService mobileUserDetailsService;
30
31     @Override
32     public void configure(HttpSecurity http) throws Exception {
33         // 创建校验手机号过滤器实例
34         MobileAuthenticationFilter mobileAuthenticationFilter
35             = new MobileAuthenticationFilter();
36         // 接收 AuthenticationManager 认证管理器
37         mobileAuthenticationFilter.setAuthenticationManager(
38             http.getSharedObject(AuthenticationManager.class));
39
40         // 采用哪个成功、失败处理器
41         mobileAuthenticationFilter.setAuthenticationSuccessHandler(
42             customAuthenticationSuccessHandler);
43         mobileAuthenticationFilter.setAuthenticationFailureHandler(
44             customAuthenticationFailureHandler);
45     }
46 }
```

```
45
46 // 为 Provider 指定明确的mobileUserDetailsService 来查询用户信息
47 MobileAuthenticationProvider provider = new MobileAuthenticationProvider();
48 provider.setUserDetailsService(mobileUserDetailsService);
49
50 // 将 provider 绑定到 HttpSecurity 上面，
51 // 并且将 手机认证加到 用户名密码认证之后
52 http.authenticationProvider(provider)
53     .addFilterAfter(mobileAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
54
55 }
56 }
57 \
```

6.2.9 绑定到安全配置 SpringSecurityConfig

1. 向 SpringSecurityConfig 中注入 MobileValidateFilter 和 MobileAuthenticationConfig 实例
2. 将 MobileValidateFilter 实例添加到 UsernamePasswordAuthenticationFilter 前面

```
1 http.addFilterBefore(mobileValidateFilter,
2     UsernamePasswordAuthenticationFilter.class)
3 .addFilterBefore(imageCodeValidateFilter,
4     UsernamePasswordAuthenticationFilter.class)
```

3. 在 SpringSecurityConfig#configure(HttpSecurity http) 方法体最后
调用 apply 添加 smsCodeAuthenticationConfig

```
1 http.apply(mobileAuthenticationConfig);
```

4. SpringSecurityConfig 完整代码

```
1 package com.mengxuegu.security.config;
2
3 import com.mengxuegu.security.authentication.code.ImageCodeValidateFilter;
4 import com.mengxuegu.security.authentication.mobile.MobileAuthenticationConfig;
5 import com.mengxuegu.security.authentication.mobile.MobileValidateFilter;
6 import com.mengxuegu.security.properties.SecurityProperties;
7 import org.slf4j.Logger;
8 import org.slf4j.LoggerFactory;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.Configuration;
12 import
    org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder
    ;
13 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
14 import org.springframework.security.config.annotation.web.builders.WebSecurity;
15 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
```



```
16 import
    org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
17 import org.springframework.security.core.userdetails.UserDetailsService;
18 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
19 import org.springframework.security.crypto.password.PasswordEncoder;
20 import org.springframework.security.web.authentication.AuthenticationFailureHandler;
21 import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
22 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
23 import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
24
25 import javax.sql.DataSource;
26
27
28 /**
29  * alt+/ 导包
30  * ctrl+o 覆盖
31  * @Author: 梦学谷 www.mengxuegu.com
32  */
33 @Configuration
34 @EnableWebSecurity // 开启springsecurity过滤链 filter
35 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
36     Logger logger = LoggerFactory.getLogger(getClass());
37
38
39     @Bean
40     public PasswordEncoder passwordEncoder() {
41         // 明文+随机盐值》加密存储
42         return new BCryptPasswordEncoder();
43     }
44
45     @Autowired
46     UserDetailsService customUserDetailsService;
47
48     /**
49      * 认证管理器：
50      * 1. 认证信息（用户名，密码）
51      * @param auth
52      * @throws Exception
53      */
54     @Override
55     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
56         // 数据库存储的密码必须是加密后的，否则会报错：There is no PasswordEncoder mapped for the id
57         // "null"
58         // String password = passwordEncoder().encode("1234");
59         // logger.info("加密之后存储的密码：" + password);
60         // auth.inMemoryAuthentication().withUser("mengxuegu")
61         //     .password(password).authorities("ADMIN");
62         auth.userDetailsService(customUserDetailsService);
63     }
64
65
66     // 配置文件参数
```

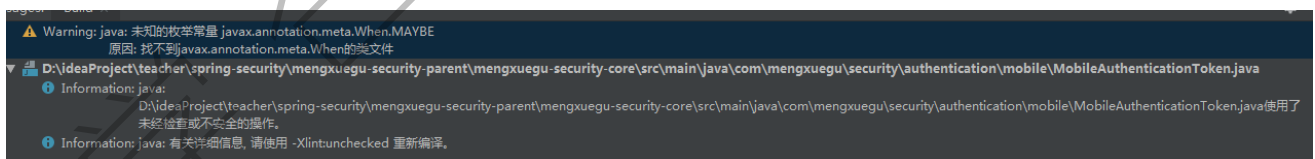
```
67     @Autowired
68     private SecurityProperties securityProperties;
69
70     @Autowired
71     private AuthenticationSuccessHandler customAuthenticationSuccessHandler;
72
73     @Autowired
74     private AuthenticationFailureHandler customAuthenticationFailureHandler;
75
76     @Autowired
77     private ImageCodeValidateFilter imageCodeValidateFilter;
78
79     @Autowired
80     DataSource dataSource;
81
82     @Autowired
83     private MobileValidateFilter mobileValidateFilter;
84
85     @Autowired
86     private MobileAuthenticationConfig mobileAuthenticationConfig;
87     /**
88      * 记住我功能
89      * @return
90      */
91     @Bean
92     public JdbcTokenRepositoryImpl jdbcTokenRepository() {
93         JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
94         jdbcTokenRepository.setDataSource(dataSource);
95         // 是否启动项目时自动创建表，true自动创建
96         // jdbcTokenRepository.setCreateTableOnStartup(true);
97         return jdbcTokenRepository;
98     }
99     /**
100      * 当你认证成功之后，springsecurity它会重写向你上一次请求上
101      * 资源权限配置：
102      * 1. 被拦截的资源
103      * @param http
104      * @throws Exception
105      */
106     @Override
107     protected void configure(HttpSecurity http) throws Exception {
108         // http.httpBasic() // 采用 httpBasic认证方式
109         // 校验手机验证码过滤器
110         http.addFilterBefore(mobileValidateFilter, UsernamePasswordAuthenticationFilter.class)
111             .addFilterBefore(imageCodeValidateFilter, UsernamePasswordAuthenticationFilter.class)
112             .formLogin() // 表单登录方式
113             .loginPage(securityProperties.getAuthentication().getLoginPage())
114             .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl()) // 登录表单提
115             交处理url，默认是/login
116             .usernameParameter(securityProperties.getAuthentication().getUsernameParameter()) // 默认的
117             是 username
118             .passwordParameter(securityProperties.getAuthentication().getPasswordParameter()) // 默认的
```

```
是 password
117     .successHandler(customAuthenticationSuccessHandler)
118     .failureHandler(customAuthenticationFailureHandler)
119     .and()
120     .authorizeRequests() // 认证请求
121     .antMatchers(securityProperties.getAuthentication().getLoginPage(),
122         "/code/image", "/mobile/page", "/code/mobile").permitAll() // 放行/login/page不需要认证可访
    问
123     .anyRequest().authenticated() //所有访问该应用的http请求都要通过身份认证才可以访问
124     .and()
125     .rememberMe() // 记住功能配置
126     .tokenRepository(jdbcTokenRepository()) //保存登录信息
127     .tokenValiditySeconds(60*60*24*7) //记住我有效时长
128     ; // 注意不要少了分号
129
130     // 将手机相关的配置绑定过滤器链上
131     http.apply(mobileAuthenticationConfig);
132 }
133
134 /**
135  * 静态资源放行
136  * @param web
137  * @throws Exception
138  */
139 @Override
140 public void configure(WebSecurity web){
141     web.ignoring().antMatchers(
142         securityProperties.getAuthentication().getStaticPaths());
143 }
144 }
```

6.2.10 编译报错 未知的枚举常量

问题：

Warning:java: 未知的枚举常量 javax.annotation.meta.When.MAYBE



解决办法：

原因是找不到默认的 javax.annotation.meta.When的类文件，缺少对应第三方依赖包，添加对应依赖包即可。

```
1 <dependency>
2   <groupId>com.google.code.findbugs</groupId>
3   <artifactId>annotations</artifactId>
4   <version>3.0.1</version>
5 </dependency>
```

6.2.11 重构失败处理器回到手机登录页

验证码认证失败后，会回到用户名密码登录页，原因是在失败处理器写死了。应该动态的重写向回上一次请求路径。

```
1 package com.mengxuegu.security.authentication;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import com.mengxuegu.security.properites.LoginResponseType;
5 import com.mengxuegu.security.properites.SecurityProperties;
6 import org.apache.commons.lang.StringUtils;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.HttpStatus;
9 import org.springframework.security.core.AuthenticationException;
10 import org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler;
11 import org.springframework.stereotype.Component;
12
13 import javax.servlet.ServletException;
14 import javax.servlet.http.HttpServletRequest;
15 import javax.servlet.http.HttpServletResponse;
16 import java.io.IOException;
17
18 /**
19  * 处理失败认证的
20  * @Author: 梦学谷 www.mengxuegu.com
21  */
22 @Component("customAuthenticationFailureHandler")
23 //public class CustomAuthenticationFailureHandler implements AuthenticationFailureHandler {
24 public class CustomAuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {
25
26     @Autowired
27     SecurityProperties securityProperties;
28     /**
29      *
30      * @param exception 认证失败时抛出异常
31      */
32     @Override
33     public void onAuthenticationFailure(HttpServletRequest request,
34         HttpServletResponse response, AuthenticationException exception) throws IOException,
35         ServletException {
36         if(LoginResponseType.JSON.equals(securityProperties.getAuthentication().getLoginType())) {
37             // 认证失败响应JSON字符串，
38             MengxueguResult result = MengxueguResult.build(HttpStatus.UNAUTHORIZED.value(),
39                 exception.getMessage());
40             response.setContentType("application/json;charset=UTF-8");
41             response.getWriter().write(result.toJsonString());
42         }else {
43             // 重写向回认证页面，注意加上 ?error
44             // super.setDefaultFailureUrl(securityProperties.getAuthentication().getLoginPage()+"?error");
45             // 获取上一次请求路径
46             String referer = request.getHeader("Referer");
47             logger.info("referer:" + referer);
48             String lastUrl = StringUtils.substringBefore(referer,"?");
49             logger.info("上一次请求的路径 : " + lastUrl);
50             super.setDefaultFailureUrl(lastUrl+"?error");
51         }
52     }
53 }
```

```
49         super.onAuthenticationFailure(request, response, exception);
50     }
51
52 }
53 }
54
```

6.4 实现手机登录RememberMe功能

6.4.1 分析实现

1. 在 `UsernamePasswordAuthenticationFilter` 拥有一个 `RememberMeServices` 的引用，其实这个接收引用的是其父抽象类 `AbstractAuthenticationProcessingFilter` 提供的 `setRememberMeServices` 方法。
2. 而在实现手机短信验证码登录时，我们自定了一个 `MobileAuthenticationFilter` 也一样的继承了 `AbstractAuthenticationProcessingFilter` 它，我们只要向其 `setRememberMeServices` 方法手动注入一个 `RememberMeServices` 实例即可。

6.4.2 编码实现

1. 在自定义的 `com.mengxuegu.security.authentication.mobile.MobileAuthenticationConfig` 中向 `MobileAuthenticationFilter` 注入 `RememberMeServices` 实例，该实例从共享对象中就可以获取到。

```
1 // 为了实现手机登录也拥有记住我的功能,将RememberMeServices传入
2 smsCodeAuthenticationFilter.setRememberMeServices(
3     http.getSharedObject(RememberMeServices.class));
```

2. 检查 记住我 的 `input` 标签的 `name="remember-me"`

```
1 <div class="col-8">
2     <div class="icheck-primary">
3         <input name="remember-me" type="checkbox" id="remember">
4         <label for="remember">
5             记住我
6         </label>
7     </div>
8 </div>
```

3. `MobileAuthenticationConfig` 完整代码

```
1 package com.mengxuegu.security.authentication.mobile;
2
3 import com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler;
4 import com.mengxuegu.security.authentication.CustomAuthenticationSuccessHandler;
5 import org.springframework.beans.factory.annotation.Autowired;
```

```
6 import org.springframework.security.authentication.AuthenticationManager;
7 import org.springframework.security.config.annotation.SecurityConfigurerAdapter;
8 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import org.springframework.security.web.DefaultSecurityFilterChain;
11 import org.springframework.security.web.authentication.RememberMeServices;
12 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
13 import org.springframework.stereotype.Component;
14
15 /**
16  * 用于组合其他关于手机登录的组件
17  * @Author: 梦学谷 www.mengxuegu.com
18  */
19 @Component
20 public class MobileAuthenticationConfig
21     extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {
22     @Autowired
23     CustomAuthenticationSuccessHandler customAuthenticationSuccessHandler;
24     @Autowired
25     CustomAuthenticationFailureHandler customAuthenticationFailureHandler;
26     @Autowired
27     UserDetailsService mobileUserDetailsService;
28
29     @Override
30     public void configure(HttpSecurity http) throws Exception {
31         MobileAuthenticationFilter mobileAuthenticationFilter = new MobileAuthenticationFilter();
32         // 获取容器中已经存在的AuthenticationManager对象，并传入 mobileAuthenticationFilter 里面
33         mobileAuthenticationFilter.setAuthenticationManager(
34             http.getSharedObject(AuthenticationManager.class));
35         //记住我功能
36         mobileAuthenticationFilter.setRememberMeServices(
37             http.getSharedObject(RememberMeServices.class));
38
39         // 传入 失败与成功处理器
40         mobileAuthenticationFilter.setAuthenticationSuccessHandler(
41             customAuthenticationSuccessHandler);
42         mobileAuthenticationFilter.setAuthenticationFailureHandler(
43             customAuthenticationFailureHandler);
44
45         // 构建一个MobileAuthenticationProvider实例，接收 mobileUserDetailsService 通过手机号查询用户信
46         MobileAuthenticationProvider provider = new MobileAuthenticationProvider();
47         provider.setUserDetailsService(mobileUserDetailsService);
48
49         // 将provider绑定到 HttpSecurity上，将手机号认证过滤器绑定到用户名密码认证过滤器之后
50         http.authenticationProvider(provider)
51             .addFilterAfter(mobileAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
52     }
53 }
54 }
55
```

6.4.3 测试

1. 重启项目
2. 访问 <http://localhost/mobile/page> 输入手机号与验证码, 勾选 记住我, 点击登录

3. 查看数据库中 persistent_logins 表的记录,

username	series	token
▶ 16888888888	b+JtxYuNA7Gc0wgnZlGm	Ctb+YX0gOI6tfo1bqDCYDA==

4. 关闭浏览器, 再重新打开浏览器访问 <http://localhost/index>, 发现会跳转回用户名密码登录页, 而正常应该勾选了 记住我, 这一步应该是可以正常访问的.

上面要求认证的原因是:

数据库中 username 为 手机号 16888888888, 当你访问 <http://localhost/index> 默认RememberMeServices 是调用 CustomUserDetailsService 通过用户名查询, 而当前在 CustomUserDetailsService 判断了用户名为 meng 才通过认证, 而此时传入的用户名是 16888888888, 所以查询不到 16888888888 用户数据.

解决方案:

1. 数据库中的 persistent_logins 表为什么存储的是手机号?

原因是当前在 MobileUserDetailsService 中返回的 User 对象中的 username 属性设置的是手机号 mobile, 而应该设置这个手机号所对应的那个用户名. 比如当前username 的值写死为 meng

```
1 @Component("mobileUserDetailsService") // 一定不要少了
2 public class MobileUserDetailsService implements UserDetailsService {
3     Logger logger = LoggerFactory.getLogger(getClass());
4     @Override
5     public UserDetails loadUserByUsername(String mobile) throws UsernameNotFoundException {
```



```
6     logger.info("请求的手机号是：" + mobile);
7     // 1. 通过手机号查询用户信息
8     // 2. 如果有用户信息，则再获取权限资源
9     // 3. 封装用户信息
10    // username 不要放手机号，放用户名，因为可以通过手机号查询出用户名
11    return new User("meng", "", true, true, true, true,
12        AuthorityUtils.commaSeparatedStringToAuthorityList("ADMIN"));
13 }
14 }
```

2. 勾选 记住我 重新登录，发现表中的username字段值就是 meng，

3. 关闭浏览再打开访问<http://localhost/> 就无需手动登录认证了。

因为默认采用的 CustomUserDetailsService 查询可查询到用户名为 meng 的信息，即认证通过。

username	series	token
meng	rrqmUy2mKjnVtKq69g==	8dc66XmJtKxr/2cj3iSCLw==

6.5 获取当前用户认证信息

6.5.1 概要

在任意地方（Controller/Service等），通过 SecurityContextHolder 类获取 getContext 上下文，getAuthentication 获取当前用户认证信息，getPrincipal 获取 UserDetails。

6.5.2 三种获取方式

重构 mengxuegu-security-web 模块的 com.mengxuegu.web.controller.MainController

```
1 package com.mengxuegu.web.controller;
2
3 import org.springframework.security.core.Authentication;
4 import org.springframework.security.core.annotation.AuthenticationPrincipal;
5 import org.springframework.security.core.context.SecurityContextHolder;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.ResponseBody;
10
11 import java.util.Map;
12
13 /**
14  * @Author: 梦学谷 www.mengxuegu.com
15  */
16 @Controller
17 public class MainController {
18
19     // 首页
```

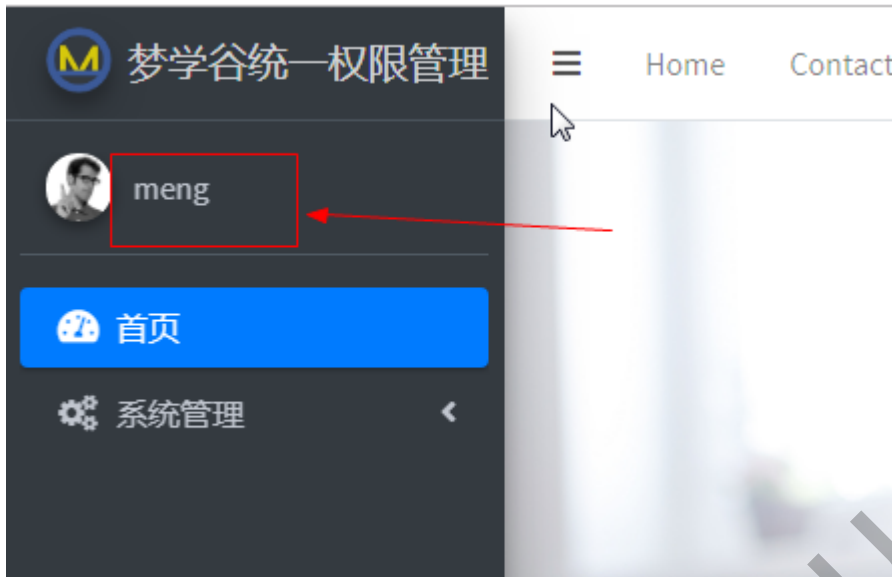
```
20 @RequestMapping("/{index}", "/", "")
21 public String index(Map<String, Object> map) {
22     // 方式1: 获取登录用户信息
23     Object principal =
24         SecurityContextHolder.getContext().getAuthentication().getPrincipal();
25     if(principal != null && principal instanceof UserDetails) {
26         UserDetails userDetails = (UserDetails) principal;
27         map.put("username", userDetails.getUsername());
28     }
29     return "index";
30 }
31
32 /**
33  * 获取当前登录用户信息, 方式2 注入 Authentication
34  * @return
35  */
36 @RequestMapping("/user/info")
37 @ResponseBody
38 public Object userInfo(Authentication authentication) {
39     return authentication.getPrincipal();
40 }
41
42 /**
43  * 获取当前登录用户信息, 方式3 注入 UserDetails
44  * @return
45  */
46 @RequestMapping("/user/info2")
47 @ResponseBody
48 public Object userInfo2(@AuthenticationPrincipal UserDetails userDetails) {
49     return userDetails;
50 }
51 }
```

6.5.3 重构左侧菜单显示用户名

打开 fragments/main-sidebar.html 页面

Ctrl + F 搜索: 梦老师 , 在标签上使用 th:text="\${username}" 获取用户名

```
1 <div class="info">
2     <a th:text="${username}" href="#" class="d-block">梦老师</a>
3 </div>
```



6.6 重构实现路径可配置

1. 在 application.yml 添加四个属性 `imageCodeUrl` `mobileCodeUrl` `mobilePage` `tokenValiditySeconds`

```
1 mengxuegu:
2   security:
3     authentication:
4       loginPage: /login/page # 响应认证(登录)页面的URL
5       loginProcessingUrl: /login/form # 登录表单提交处理的url
6       usernameParameter: name # 登录表单提交的用户名的属性名
7       passwordParameter: pwd # 登录表单提交的密码的属性名
8       staticPaths: # 静态资源 "/dist/**", "/modules/**", "/plugins/**"
9         - /dist/**
10        - /modules/**
11        - /plugins/**
12       loginType: REDIRECT # 认证之后 响应的类型: JSON/REDIRECT
13
14       imageCodeUrl: /code/image # 获取图形验证码 url
15       mobileCodeUrl: /code/mobile # 发送手机验证码 url
16       mobilePage: /mobile/page # 前往手机登录页面地址
17       tokenValiditySeconds: 604800 # 记住我有效时长, 单位秒, 注意不要用乘法*, 会被认为字符串
```

2. 在 `AuthenticationProperties` 添加以下属性, 在类上加了 `@Data` 注解, 把 setter / getter 方法全部删除

```
1 package com.mengxuegu.security.properites;
2
3 import lombok.Data;
4
5 /**
6  * @Author: 梦学谷 www.mengxuegu.com
7  */
8 @Data
9 public class AuthenticationProperties {
```

```
10
11     private String loginPage = "/login/page";
12     private String loginProcessingUrl = "/login/form";
13     private String usernameParameter = "name";
14     private String passwordParameter = "pwd";
15     private String[] staticPaths = {"/dist/**", "/modules/**", "/plugins/**"};
16
17     /**
18      * 认证响应的类型：JSON、REDIRECT 重定向
19      */
20     private LoginResponseType loginType = LoginResponseType.REDIRECT;
21
22     /**
23      * 获取图形验证码 url
24      */
25     private String imageCodeUrl = "/code/image";
26
27     /**
28      * 发送手机验证码 url
29      */
30     private String mobileCodeUrl = "/code/mobile";
31
32     /**
33      * 前往手机登录页面地址
34      */
35     private String mobilePage = "/mobile/page";
36
37     /**
38      * 记住我有效时长
39      */
40     private Integer tokenValiditySeconds = 60*60*24*7;
41 }
```

3. SpringSecurityConfig 完整版

```
1 package com.mengxuegu.security.config;
2
3 ...
4
5 /**
6  * alt+/ 导包
7  * ctrl+o 覆盖
8  * @Author: 梦学谷 www.mengxuegu.com
9  */
10 @Configuration
11 @EnableWebSecurity // 开启springsecurity过滤链 filter
12 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
13     Logger logger = LoggerFactory.getLogger(getClass());
14
15
16     @Bean
17     public PasswordEncoder passwordEncoder() {
```

```
18     // 明文+随机盐值》加密存储
19     return new BCryptPasswordEncoder();
20 }
21
22 @Autowired
23 UserDetailsService customUserDetailsService;
24
25 /**
26  * 认证管理器：
27  * 1. 认证信息（用户名，密码）
28  * @param auth
29  * @throws Exception
30  */
31 @Override
32 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
33     // 数据库存储的密码必须是加密后的，否则会报错：There is no PasswordEncoder mapped for the id
    "null"
34     // String password = passwordEncoder().encode("1234");
35     // logger.info("加密之后存储的密码：" + password);
36     // auth.inMemoryAuthentication().withUser("mengxuegu")
37     //     .password(password).authorities("ADMIN");
38     auth.userDetailsService(customUserDetailsService);
39 }
40
41
42
43 // 配置文件参数
44 @Autowired
45 private SecurityProperties securityProperties;
46
47 @Autowired
48 private AuthenticationSuccessHandler customAuthenticationSuccessHandler;
49
50 @Autowired
51 private AuthenticationFailureHandler customAuthenticationFailureHandler;
52
53 @Autowired
54 private ImageCodeValidateFilter imageCodeValidateFilter;
55
56 @Autowired
57 DataSource dataSource;
58
59 // 校验手机验证码
60 @Autowired
61 private MobileValidateFilter mobileValidateFilter;
62
63 // 校验手机号是否存在，就是手机号认证
64 @Autowired
65 private MobileAuthenticationConfig mobileAuthenticationConfig;
66
67 /**
68  * 记住我功能
69  * @return
```

```
70     */
71     @Bean
72     public JdbcTokenRepositoryImpl jdbcTokenRepository() {
73         JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
74         jdbcTokenRepository.setDataSource(dataSource);
75         // 是否启动项目时自动创建表，true自动创建
76         // jdbcTokenRepository.setCreateTableOnStartup(true);
77         return jdbcTokenRepository;
78     }
79     /**
80     * 当你认证成功之后，springsecurity它会重写向到你上一次请求上
81     * 资源权限配置：
82     * 1. 被拦截的资源
83     * @param http
84     * @throws Exception
85     */
86     @Override
87     protected void configure(HttpSecurity http) throws Exception {
88         // http.httpBasic() // 采用 httpBasic认证方式
89         // 校验手机验证码过滤器
90         http.addFilterBefore(mobileValidateFilter, UsernamePasswordAuthenticationFilter.class)
91             .addFilterBefore(imageCodeValidateFilter, UsernamePasswordAuthenticationFilter.class)
92             .formLogin() // 表单登录方式
93             .loginPage(securityProperties.getAuthentication().getLoginPage())
94             .loginProcessingUrl(securityProperties.getAuthentication().getLoginProcessingUrl()) // 登录表单提
交处理url，默认是/login
95             .usernameParameter(securityProperties.getAuthentication().getUsernameParameter()) // 默认的
是 username
96             .passwordParameter(securityProperties.getAuthentication().getPasswordParameter()) // 默认的
是 password
97             .successHandler(customAuthenticationSuccessHandler)
98             .failureHandler(customAuthenticationFailureHandler)
99             .and()
100            .authorizeRequests() // 认证请求
101            .antMatchers(securityProperties.getAuthentication().getLoginPage(),
102            // "/code/image", "/mobile/page", "/code/mobile"
103                securityProperties.getAuthentication().getImageCodeUrl(),
104                securityProperties.getAuthentication().getMobilePage(),
105                securityProperties.getAuthentication().getMobileCodeUrl()
106            ).permitAll() // 放行/login/page不需要认证可访问
107            .anyRequest().authenticated() // 所有访问该应用的http请求都要通过身份认证才可以访问
108            .and()
109            .rememberMe() // 记住功能配置
110            .tokenRepository(jdbcTokenRepository()) // 保存登录信息
111            .tokenValiditySeconds(securityProperties.getAuthentication().getTokenValiditySeconds()) // 记住我
有效时长
112            ; // 注意不要少了分号
113
114            // 将手机认证添加到过滤器链上
115            http.apply(mobileAuthenticationConfig);
116
117     }
118
```

```
119  /**
120   * 一般是指针对静态资源放行
121   * @param web
122   * @throws Exception
123   */
124   @Override
125   public void configure(WebSecurity web){
126       web.ignoring().antMatchers(securityProperties.getAuthentication().getStaticPaths());
127   }
128 }
```

第七章 Session 会话管理

7.1 配置 Session 会话超时

7.1.1 application.yml 配置超时时间

```
1 server:
2   port: 80
3   servlet:
4     session:
5       timeout: 1m # session超时时间默认30m (30分钟)，至少设置1分钟
```

7.1.2 底层源码分析

TomcatServletWebServerFactory#configureSession

```
1 private void configureSession(Context context) {
2     // 获取超时时长
3     long sessionTimeout = this.getSessionTimeoutInMinutes();
4 }
5 private long getSessionTimeoutInMinutes() {
6     // Math.max(sessionTimeout.toMinutes(), 1L) 至少1分钟，配置小于1分钟也取1分钟
7     Duration sessionTimeout = this.getSession().getTimeout();
8     return this.isZeroOrLess(sessionTimeout) ? 0L : Math.max(sessionTimeout.toMinutes(), 1L);
9 }
```

7.2 自定义 Session 失效处理

默认情况下，当 session 失效后会请求回认证页面。我们可以自定义 session 失效后，响应不同的结果。

1. 添加session失效处理：com.mengxuegu.security.config.SpringSecurityConfig


```
1 注入session失败策略
2 @Autowired
3 private InvalidSessionStrategy invalidSessionStrategy;
4
5 配置 session 管理
6 .and()
7 .sessionManagement()
8 .invalidSessionStrategy(invalidSessionStrategy) // session失效后处理逻辑
```

2. 在 mengxuegu-security-core 工程中创建

com.mengxuegu.security.authentication.session.CustomInvalidSessionStrategy

```
1 package com.mengxuegu.security.authentication.session;
2
3 import com.mengxuegu.base.result.MengxueguResult;
4 import org.springframework.http.HttpStatus;
5 import org.springframework.security.web.session.InvalidSessionStrategy;
6
7 import javax.servlet.http.Cookie;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10 import java.io.IOException;
11
12 /**
13  * session失效后的处理逻辑
14  * @Author: 梦学谷 www.mengxuegu.com
15  */
16 public class CustomInvalidSessionStrategy implements InvalidSessionStrategy {
17
18     @Override
19     public void onInvalidSessionDetected(HttpServletRequest request,
20         HttpServletResponse response) throws IOException {
21         // 将浏览器的sessionid清除，不关闭浏览器cookie不会被删除，一直请求都提示：Session失效
22         cancelCookie(request,response);
23         MengxueguResult result = MengxueguResult.build(
24             HttpStatus.UNAUTHORIZED.value(), "登录已超时, 请重新登录");
25         response.setContentType("application/json;charset=utf-8");
26         response.getWriter().write(result.toJsonString());
27     }
28     // 参考记住我功能的 AbstractRememberMeServices 代码
29     protected void cancelCookie(HttpServletRequest request, HttpServletResponse response) {
30         Cookie cookie = new Cookie("JSESSIONID", null);
31         cookie.setMaxAge(0);
32         cookie.setPath(getCookiePath(request));
33         response.addCookie(cookie);
34     }
35     private String getCookiePath(HttpServletRequest request) {
36         String contextPath = request.getContextPath();
37         return contextPath.length() > 0 ? contextPath : "/";
38     }
39 }
```

39 }

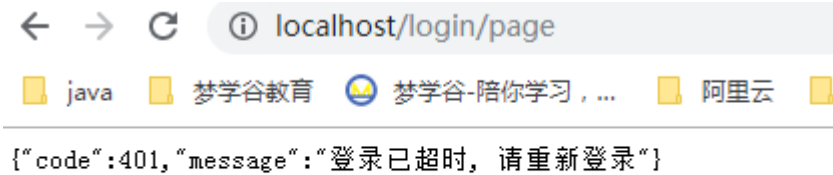
3. 注入Session失效策略实例, 在 com.mengxuegu.security.config.SecurityConfigBean 添加方法注入
方便web应用覆盖此实现,定义不同逻辑

```
1 package com.mengxuegu.security.config;
2
3 import com.mengxuegu.security.authentication.mobile.SmsCodeSender;
4 import com.mengxuegu.security.authentication.mobile.SmsSend;
5 import com.mengxuegu.security.authentication.session.CustomInvalidSessionStrategy;
6 import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.security.web.session.InvalidSessionStrategy;
10
11 /**
12  * 主要为容器中添加Bean实例
13  * @Author: 梦学谷 www.mengxuegu.com
14  */
15 @Configuration
16 public class SecurityConfigBean {
17
18     /**
19      * 注入Session失效策略实例
20      * @return
21      */
22     @Bean
23     @ConditionalOnMissingBean(InvalidSessionStrategy.class)
24     public InvalidSessionStrategy invalidSessionStrategy() {
25         return new CustomInvalidSessionStrategy();
26     }
27
28     /**
29      * @ConditionalOnMissingBean(SmsSend.class)
30      * 默认情况下,采用的是SmsCodeSender实例,
31      * 但是如果容器当中有其他的SmsSend类型的实例,
32      * 那当前的这个SmsCodeSender就失效了
33      * @return
34      */
35     @Bean
36     @ConditionalOnMissingBean(SmsSend.class)
37     public SmsSend smsSend() {
38         return new SmsCodeSender();
39     }
40
41 }
```

4. 测试

1. 先登录

2. 再重启项目让session失效
3. 再访问: 提示超时



7.3 用户只允许一个地方登录

只允许一个用户在一个地方登录，也是每个用户在系统中只能有一个Session。

7.3.1 情景一

说明

如果同一用户在第2个地方登录，则将第1个踢下线。

实现步骤

重构 com.mengxuegu.security.config.SpringSecurityConfig

```
1 注入session失败策略
2  @Autowired
3  private InvalidSessionStrategy invalidSessionStrategy;
4  @Autowired
5  private SessionInformationExpiredStrategy sessionInformationExpiredStrategy;
6
7
8 配置 Session 管理
9  .and()
10 .sessionManagement()
11 .invalidSessionStrategy(invalidSessionStrategy) // session失效后处理逻辑
12 .maximumSessions(1) // 每个用户在系统中的最大session数
13 .expiredSessionStrategy(sessionInformationExpiredStrategy) // 当用户达到最大session数后，则调用此处的实现
```

自定义 SessionInformationExpiredStrategy 实现类来定制策略

```
1 package com.mengxuegu.security.authentication.session;
2
3 import com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.security.authentication.AuthenticationServiceException;
6 import org.springframework.security.core.AuthenticationException;
7 import org.springframework.security.core.userdetails.UserDetails;
```

```
8 import org.springframework.security.web.session.SessionInformationExpiredEvent;
9 import org.springframework.security.web.session.SessionInformationExpiredStrategy;
10 import org.springframework.stereotype.Component;
11
12 import javax.servlet.ServletException;
13 import java.io.IOException;
14
15 /**
16  * 当同一用户的 session 达到指定数量时，执行此类
17  * @Author: 梦学谷 www.mengxuegu.com
18  */
19 public class CustomSessionInformationExpiredStrategy implements SessionInformationExpiredStrategy {
20
21     @Autowired
22     CustomAuthenticationFailureHandler customAuthenticationFailureHandler;
23
24     @Override
25     public void onExpiredSessionDetected(SessionInformationExpiredEvent event) throws IOException {
26         // 退出的用户
27         UserDetails userDetails =
28             (UserDetails)event.getSessionInformation().getPrincipal();
29
30         AuthenticationException exception =
31             new AuthenticationServiceException(String.format("[%s]用户在另外一台电脑登录，您已被下线",
32                 userDetails.getUsername()));
33         try {
34             event.getRequest().setAttribute("toAuthentication", true);
35             customAuthenticationFailureHandler.onAuthenticationFailure(
36                 event.getRequest(), event.getResponse(), exception);
37         } catch (ServletException e) {
38             e.printStackTrace();
39         }
40     }
```

重构 com.mengxuegu.security.authentication.CustomAuthenticationFailureHandler 处理器:

如果isAuthentication值为true，说明是session数量超过，则回到 /login/page

```
1 }else {
2     // 获取上一次请求路径
3     String referer = request.getHeader("Referer");
4     logger.info("referer:" + referer);
5     // String lastUrl = StringUtils.substringBefore(referer,"?");
6
7     // 如果需要认证，跳转 /login/page, 否则截取
8     Object toAuthentication = request.getAttribute("toAuthentication");
9     String lastUrl = toAuthentication != null?
10         securityProperties.getAuthentication().getLoginPage() :
11         StringUtils.substringBefore(referer,"?");
12
13     logger.info("上一次请求的路径 : " + lastUrl);
14     super.setDefaultFailureUrl(lastUrl + "?error");
```

```
15  
16     super.onAuthenticationFailure(request, response, exception);  
17 }
```

com.mengxuegu.security.config.SecurityConfigBean 添加方法注入SessionInformationExpiredStrategy实现方便web应用覆盖此实现,定义不同逻辑

```
1 @Bean  
2 @ConditionalOnMissingBean(SessionInformationExpiredStrategy.class)  
3 public SessionInformationExpiredStrategy sessionInformationExpiredStrategy() {  
4     return new CustomSessionInformationExpiredStrategy();  
5 }
```

测试：

1. 谷歌浏览器用户名密码登录
2. 再QQ浏览器用户名密码登录
3. 回到谷歌浏览器刷新请求，发现回到登录页面，提示被下线



7.3.2 情景二

说明

如果同一用户在第2个地方登录时，则不允许他二次登录。

实现步骤

1. 在 SpringSecurityConfig 添加 `maxSessionsPreventsLogin(true)` 后不允许二次登录

```
1 .and()
2 .sessionManagement()
3 .invalidSessionStrategy(invalidSessionStrategy) // session失效后处理逻辑
4 .maximumSessions(1) // 每个用户在系统中的最大session数
5 .expiredSessionStrategy(sessionInformationExpiredStrategy)// 当用户达到最大session数后，则调用此处的实现
6 .maxSessionsPreventsLogin(true)// 当一个用户达到最大session数，则不允许后面进行登录
```

2. 测试

1. 谷歌浏览器用户名密码登录
2. 再QQ浏览器用户名密码登录，发现不允许登录



梦学谷统一权限管理

请登录

☒ 使用短信验证登录

用户名 

密码 

验证码 

已经超过了当前主体(1)被允许的最大会话数量

☐ 记住我

[忘记密码](#)

[注册帐号](#)

登录

7.3.3 解决同一用户的手机重复登录问题

问题描述：

使用了 admin 用户名登录后，还可以使用这个用户的手机号登录。

正常应该是同一个用户，系统中只能用用户名或手机号登录一次。

解决问题：

原因是 `MobileAuthenticationFilter` 继承的类 `AbstractAuthenticationProcessingFilter` 中，默认使用了 `NullAuthenticatedSessionStrategy` 实例管理 Session，我们应该指定用户名密码过滤器中所使用的那个 `SessionAuthenticationStrategy` 实现类 `CompositeSessionAuthenticationStrategy`。

在 `com.mengxuegu.security.authentication.mobile.MobileAuthenticationConfig` 指定即可解决：

```
1 // session 会话管理功能
2 mobileAuthenticationFilter.setSessionAuthenticationStrategy(http.getSharedObject(SessionAuthenticationStrategy.class));
```

7.4 Redis 实现 Session 高可用集群

7.4.1 概述

如果采用默认的单机版 Session 存储身份信息时，一旦这台服务器挂了就没法使用。

那我们可以为项目搭建集群，部署到AB 两台服务器上，但是部署到AB 两台服务器上，Session就没有保证一致，

假设 user1 第1次登录访问的是 A服务器，后面访问资源时请求到了B服务器，而当前B并没有存储用户session，这样又会要求用户再次登录，

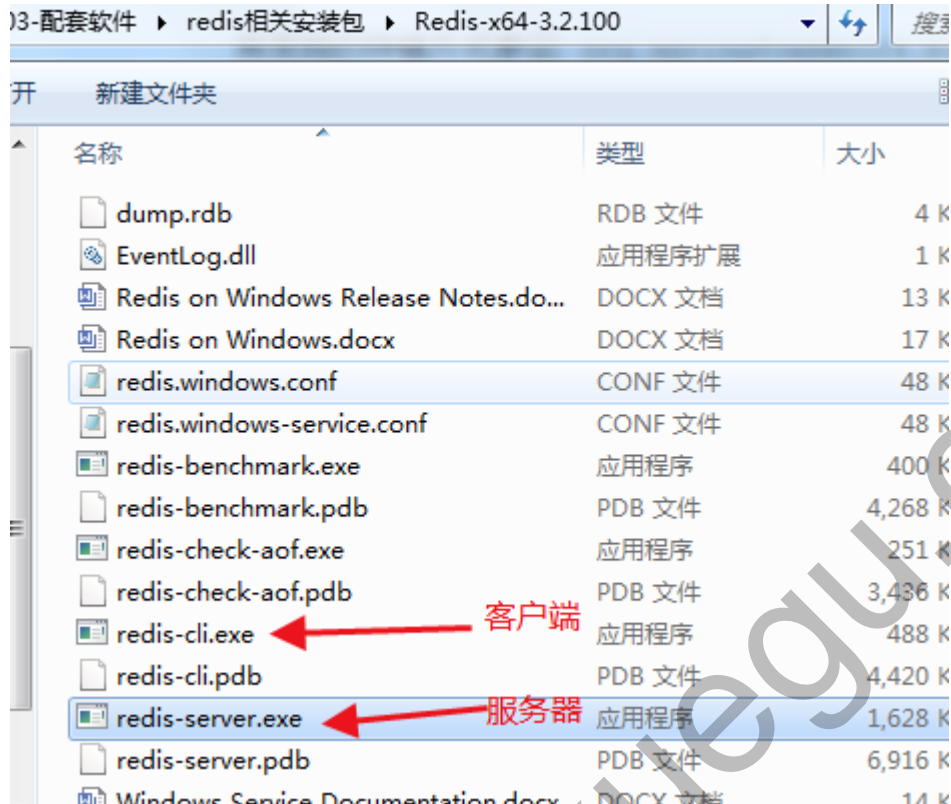
所以目前方式也是不可以的，不能让用户到两台机器上各认证一次，应该在一台机器上认证即可。

不管用户访问哪台服务器，我们将用户的session 都放到 redis，后面请求都从 redis 获取 session，这样可以保证一致性。

7.4.2 操作步骤

所支持的存储方式参见：`org.springframework.boot.autoconfigure.session.StoreType`

1. 以 管理员身份运行 03-配套软件\redis相关安装包\Redis-x64-3.2.100\redis-server.exe，



2. mengxuegu-security-core\pom.xml 添加对 redis 的支持

```
1 <!--采用redis来管理session-->
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-data-redis</artifactId>
5 </dependency>
6 <dependency>
7   <groupId>org.springframework.session</groupId>
8   <artifactId>spring-session-data-redis</artifactId>
9 </dependency>
```

3. application.yml 指定存储方式 redis

```
1 spring:
2   session:
3     store-type: redis # 使用 redis 存储session
4     # redis: # 本地环境可以不配置
5     # port: 6379 # redis服务端端口号
6     # host: 127.0.0.1
```

4. 重启项目, 进行登录

5. 管理员身份运行客户端 03-配套软件\redis相关安装包\Redis-x64-3.2.100\redis-cli.exe

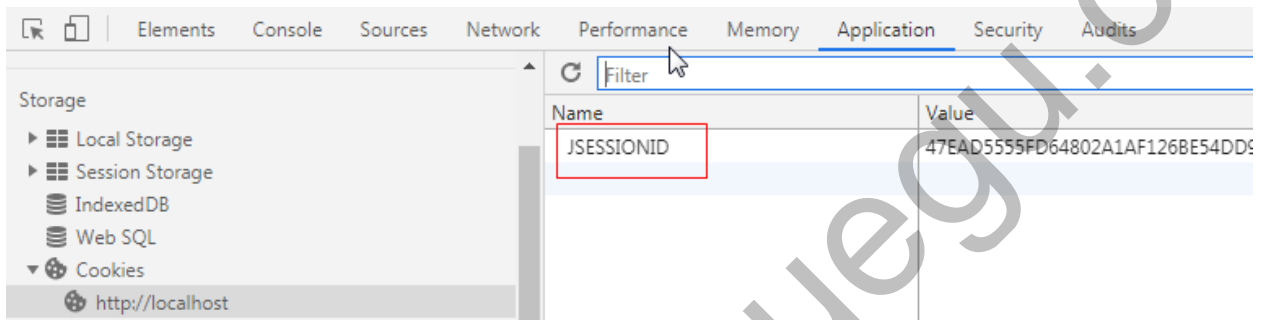
执行 `keys *` 命令查看保存的数据

```
127.0.0.1:6379> keys *
1) "spring:session:sessions:expires:4f299d2c-8a8e-4034-8fa3-22cd41f6834e"
2) "spring:session:index:org.springframework.session.FindByIndexNameSessionRepository.PRINCIPAL_NAME_INDEX_NAME:meng"
3) "spring:session:sessions:4f299d2c-8a8e-4034-8fa3-22cd41f6834e"
4) "spring:session:expirations:1574153220000"
```

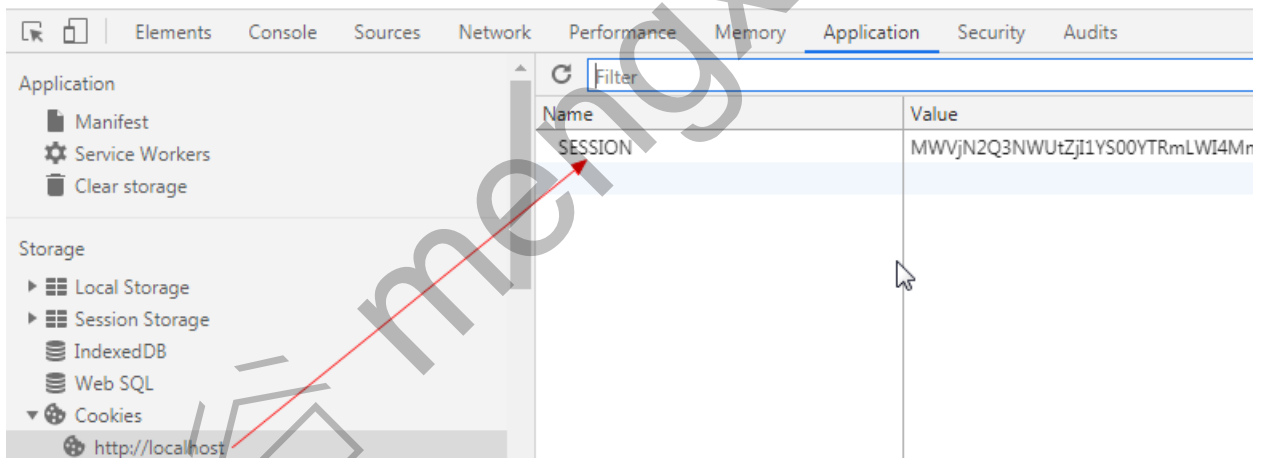
7.4.3 指定 Cookie 中保存SessionID名称

说明

1. 默认情况下，浏览器的 Cookie 中保存 SessionID 名称是 JSESSIONID，



2. 当使用redis保存session信息后，浏览器的 Cookie 中保存 SessionID 名称是 SESSION，



这样会导致当session失效后，在 CustomInvalidSessionStrategy 中只将 JSESSIONID 清除是不行的，不管刷新请求多少次，都是提示：登录已超时，请重新登录。

现象

1. 先登录
2. redis 使用 flushall 命令清除所有数据，即将session值失效

```
127.0.0.1:6379> flushall
OK
127.0.0.1:6379>
```

3. 再刷新多次浏览器，一直提示：登录已超时，请重新登录。

← → ↺ ⓘ localhost I

java 梦学谷教育 梦学谷-陪你学习, ... 阿里云

{"code":401,"message":"登录已超时，请重新登录"}

解决

统一指定浏览器中 Cookie 保存的 SessionID 名称为 JSESSIONID

```
1 server:
2   port: 80
3   servlet:
4     session:
5       timeout: 10m # session会话超时时间，默认情况下是30分钟（m），不能小于1分钟
6       cookie:
7         name: JSESSIONID # 统一指定浏览器中 Cookie 保存的SessionID名称
```

如果想关闭 redis 存储 session, 作如下配置：

```
1 spring
2   session:
3     store-type: none
```

未关闭，则每次要启动redis服务器，Redis-x64-3.2.100\redis-server.exe

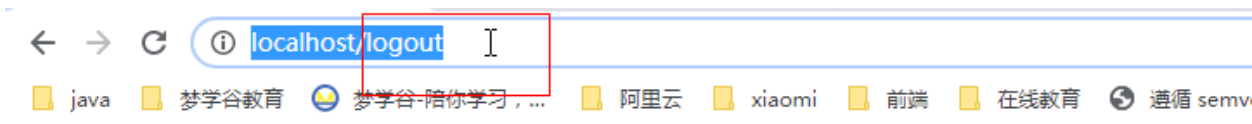
7.5 退出系统

7.5.1 退出系统默认配置

1. 默认请求 /logout 即可退出系统，在 templates/fragments/main-header.html 指定退出URL，如下：

```
1 <a th:href="@{/logout}" href="login.html" class="dropdown-item" >
2   <i class="fa fa-sign-out mr-2"></i> 退出系统
3 </a>
```

2. 点击 退出系统 会报404



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Nov 19 16:42:31 CST 2019

There was an unexpected error (type=Not Found, status=404).

No message available

会报-报404原因:

默认情况下 `/logout` 必须使用POST提交才可以起作用, 原因是防止 csrf (跨站请求伪造) 功能默认开启了

解决问题:

在 `SpringSecurityConfig` 关闭 csrf, 即可使用 get 方式请求 `/logout`

```
1 ; // 注意不要少了分号
2
3 http.csrf().disable(); // 关闭跨站请求伪造
```

退出后的效果 : `/login/page?error`



3. 底层默认退出处理操作 :

1. 清除浏览器 remember-me 的 Cookie、通过用户名删除 remember-me 数据库记录
2. 将当前用户的 Session 失效, 清空当前用户的 Authentication
3. 重写向到登录页面, 带上error请求参数, 地址 : `/login/page?error`

7.5.2 退出底层源码分析

关注 `org.springframework.security.web.authentication.logout.LogoutFilter`

```
1 public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException,
   ServletException {
2     HttpServletRequest request = (HttpServletRequest)req;
3     HttpServletResponse response = (HttpServletResponse)res;
4     // 请求URL是否为 /logout
5     if (this.requiresLogout(request, response)) {
6         // 获取认证信息
7         Authentication auth = SecurityContextHolder.getContext().getAuthentication();
8         if (this.logger.isDebugEnabled()) {
9             this.logger.debug("Logging out user " + auth + " and transferring to logout destination");
10        }
11        // 退出处理，调用 CompositeLogoutHandler#logout，参见下面第2步
12        this.handler.logout(request, response, auth);
13        // 重写向/login/page?error, 调用 SimpleUrlLogoutSuccessHandler#onLogoutSuccess
14        this.logoutSuccessHandler.onLogoutSuccess(request, response, auth);
15    } else {
16        chain.doFilter(request, response);
17    }
18 }
```

`CompositeLogoutHandler#logout` 退出逻辑处理

```
1 public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication)
   {
2     /*
3     有3个处理操作：
4     0 = 清除浏览器 remember-me 的cookie、通过用户名删除 remember-me记录
5     0 = {PersistentTokenBasedRememberMeServices@8916}
6     1 = 将当前用户的 Session 失效，清空当前用户的 Authentication
7     1 = {SecurityContextLogoutHandler@9174}
8     2 = 成功退出系统通知，在响应头中发送http状态代码，默认情况下它会发送httpstatus.ok
9     2 = {LogoutSuccessEventPublishingLogoutHandler@9175}
10    */
11    for (LogoutHandler handler : this.logoutHandlers) {
12        handler.logout(request, response, authentication);
13    }
14 }
```

7.5.3 解决退出不允许再次登录

现象

配置了 `.maxSessionsPreventsLogin(true)` 开启了前面已登录，不允许再重复登录

上面默认情况，如果登录后，然后请求 `/logout` 退出，再重新登录时，会提示不能重复登录。

梦学谷统一权限管理

请登录

☒ 使用短信验证登录

用户名 

密码 

验证码 

当前用户已经在另外一台电脑登录了, 不允许重复登录

☐ 记住我

忘记密码

注册帐号

登录

源码分析

每次登录请求都会执行 `ConcurrentSessionControlAuthenticationStrategy#onAuthentication` 判断用户在系统是否已经存在 session了, 且判断是否已经超过限制的session数量了, 超出抛异常原因是退出时, 并没有将 `SessionRegistryImpl.principals` 缓存的用户信息.

```
1
2 public void onAuthentication(Authentication authentication,
3     HttpServletRequest request, HttpServletResponse response) {
4     // 获取当前用户在系统中的所有 session
5     final List<SessionInformation> sessions = sessionRegistry.getAllSessions(
6         authentication.getPrincipal(), false);
7
8     int sessionCount = sessions.size();
9     int allowedSessions = getMaximumSessionsForThisUser(authentication);
10
11     if (sessionCount < allowedSessions) {
12         // 用户session总个数 小于 设置的最大session数, 则直接通过
13         return;
14     }
15
16     if (allowedSessions == -1) {
17         // -1 不受限制 maximumSessions(-1), 则直接通过
18         return;
19     }
```

```
20
21 // 创建session
22 if (sessionCount == allowedSessions) {
23     HttpSession session = request.getSession(false);
24
25     if (session != null) {
26         // Only permit it though if this request is associated with one of the
27         // already registered sessions
28         for (SessionInformation si : sessions) {
29             if (si.getSessionId().equals(session.getId())) {
30                 return;
31             }
32         }
33     }
34     // If the session is null, a new one will be created by the parent class,
35     // exceeding the allowed number
36 }
37 // 超出允许的 session 的个数，maxSessionsPreventsLogin(true)抛出异常不让登录
38 allowableSessionsExceeded(sessions, allowedSessions, sessionRegistry);
39 // exceptionIfMaximumExceeded 对应就是 maxSessionsPreventsLogin(true)
40 }
```

解决方案

添加一个退出处理器 `com.mengxuegu.security.authentication.session.CustomLogoutHandler` ,将用户信息从缓存中清除

```
1 package com.mengxuegu.security.authentication.session;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.core.Authentication;
5 import org.springframework.security.core.session.SessionRegistry;
6 import org.springframework.security.web.authentication.logout.LogoutHandler;
7 import org.springframework.stereotype.Component;
8
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 /**
13  * @Author: 梦学谷 www.mengxuegu.com
14  */
15 @Component
16 public class CustomLogoutHandler implements LogoutHandler {
17
18     @Autowired
19     private SessionRegistry sessionRegistry;
20
21     @Override
22     public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {
23         // 配置 .maxSessionsPreventsLogin(true) 开启了前面已登录，不允许再重复登录
24
25         // 默认情况，如果登录后，然后请求 /logout 退出，再重新登录时，会提示不能重复登录。
26     }
27 }
```



```
25 // 清除session信息，原因是并没有从 SessionRegistryImpl.principals 移除用户信息
26 sessionRegistry.removeSessionInformation(request.getSession().getId());
27 }
28
29 }
```

SpringSecurityConfig 注入 customLogoutHandler

```
1 @Autowired
2 private LogoutHandler customLogoutHandler;
3
4 @Bean
5 public SessionRegistry sessionRegistry(){
6     return new SessionRegistryImpl();
7 }
8
9 @Override
10 protected void configure(HttpSecurity http) throws Exception {
11
12     .and()
13     .sessionManagement()// session管理
14     .invalidSessionStrategy(invalidSessionStrategy) //当session失效后的处理类
15     .maximumSessions(1) // 每个用户在系统中最多可以有多少个session
16     .expiredSessionStrategy(sessionInformationExpiredStrategy)// 当用户达到最大session数后，则调用此处的
    实现
17     .maxSessionsPreventsLogin(true) // 当一个用户达到最大session数,则不允许后面再登录
18     .sessionRegistry(sessionRegistry()) // ++++
19     .and().and() // 两个and
20     .logout().addLogoutHandler(new CustomLogoutHandler()) // +++ 退出处理
21     ;
22
23     http.csrf().disable(); // 关闭跨站请求伪造
24     //将手机认证添加到过滤器链上
25     http.apply(mobileAuthenticationConfig);
26 }
```

7.5.4 自定义退出处理

在 `com.mengxuegu.security.config.SpringSecurityConfig` 做如下配置：

1. 默认退出系统处理的URL是 `/logout`，修改为 `/user/logout`
 - 修改1：SpringSecurityConfig

```
1 .and().and()
2 .logout()
3 .addLogoutHandler(customLogoutHandler) // 退出清除缓存
4 .logoutUrl("/user/logout") //退出系统URL
5
```

- 修改2：main-header.html

```
1 <a th:href="@{/user/logout}" href="login.html" class="dropdown-item" >
2   <i class="fa fa-sign-out mr-2"></i> 退出系统
3 </a>
```

2. 默认退出成功后的请求URL是 /login/page?error ，修改为跳转指定页面

- 修改：SpringSecurityConfig，注意：指定URL记得放行

```
1 .and().and()
2 .logout()
3 .addLogoutHandler(customLogoutHandler) // 退出清除缓存
4 .logoutUrl("/user/logout") //退出系统URL
5 .logoutSuccessUrl("/mobile/page") //退出成功后请求 /mobile/page 回到手机登录页
```

3. 删除特定的 Cookie 值

```
1 .and().and()
2 .logout()
3 .addLogoutHandler(customLogoutHandler) // 退出清除缓存
4 .logoutUrl("/user/logout") //退出系统URL
5 .logoutSuccessUrl("/mobile/page") //退出成功后请求 /mobile/page 回到手机登录页
6 .deleteCookies("JSESSIONID") // 删除特定的Cookie值
```