

Spring Boot开发笔记

学习内容

1.Spring Boot概述

2.Spring Boot入门程序

3.Spring Boot配置详解

4.Spring Boot日志系统详解

5.Spring Boot与Web整合

6.Spring Boot与模板引擎的整合

7.Spring Boot与MyBatis整合

8.Spring Boot与JPA整合

9.Spring Boot与缓存的整合

10.Spring Boot与安全框架（ Shiro、 Spring Security ）

11.Spring Boot与消息队列整合
（ ActiveMQ,RabbitMQ,RocketMQ,Kafka ）

12.Spring Boot与Elasticsearch（ Elastic Stack ）

13.Spring Boot与Mongodb整合

14.Spring Boot与任务调度（ quartz,xxl-job ）

15.Spring Boot与Dubbo(Zookeeper)

16.Spring Boot源码分析， 配置原理

一.Spring Boot概述

1. Spring Boot是什么

- Spring Boot基于Spring框架之上的一个微服务架构开发的一个框架
- 大大简化了Spring的开发。因为Spring Boot提供了大量的自动配置。而且它是基于Java配置方式的开发（全注解）
- Spring Boot与其他第三方的框架集成，实现了自动配置
- Spring Boot和Spring Cloud关系：Spring Cloud的开发需要用Spring Boot，反之不一定
- 官方介绍

```
1 Spring Boot makes it easy to create stand-alone, production-grade Spring
  based Applications that you can "just run".
2 We take an opinionated view of the Spring platform and third-party
  libraries so you can get started with minimum fuss. Most Spring Boot
  applications need very little Spring configuration.
```

2.Spring Boot特点

- ```
1 Create stand-alone Spring applications
2 Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
3 Provide opinionated 'starter' dependencies to simplify your build
 configuration
4 Automatically configure Spring and 3rd party libraries whenever possible
5 Provide production-ready features such as metrics, health checks and
 externalized configuration
6 Absolutely no code generation and no requirement for XML configuration
```

# 二.Spring Boot入门案例

## 2.1 手动开发

### 2.1.1 编写pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://maven.apache.org/POM/4.0.0"
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 <parent>
8 <groupId>org.springframework.boot</groupId>
9 <artifactId>spring-boot-starter-parent</artifactId>
10 <version>2.2.2.RELEASE</version>
11 </parent>
12
13 <groupId>com.bjlemon</groupId>
14 <artifactId>springboot-demo-1</artifactId>
```

```

15 <version>1.0-SNAPSHOT</version>
16
17 <dependencies>
18 <dependency>
19 <groupId>org.springframework.boot</groupId>
20 <artifactId>spring-boot-starter-web</artifactId>
21 </dependency>
22
23 <dependency>
24 <groupId>org.springframework.boot</groupId>
25 <artifactId>spring-boot-starter-test</artifactId>
26 </dependency>
27 </dependencies>
28
29 </project>

```

## 2.1.2 编写springboot配置文件 ( application.properties或 application.yml )

```

1 server:
2 port: 9999
3 spring:
4 application:
5 name: springboot-demo-1
6 mvc:
7 servlet:
8 path: /demo
9 date-format: yyyy/MM/dd
10 jackson:
11 date-format: yyyy/MM/dd

```

## 2.1.3编写启动类

```

1 @SpringBootApplication
2 public class SpringBootDemoApplication {
3
4 public static void main(String[] args) {
5 SpringApplication.run(SpringBootDemoApplication.class, args);
6 }
7 }

```

## 2.1.4 业务方法

```

1 @Controller
2 public class HelloWorldController {
3
4 @GetMapping("/sayHello")
5 @ResponseBody
6 public String sayHello() {
7 return "HelloWorld SpringBoot";
8 }
9 }
10
11 @Controller

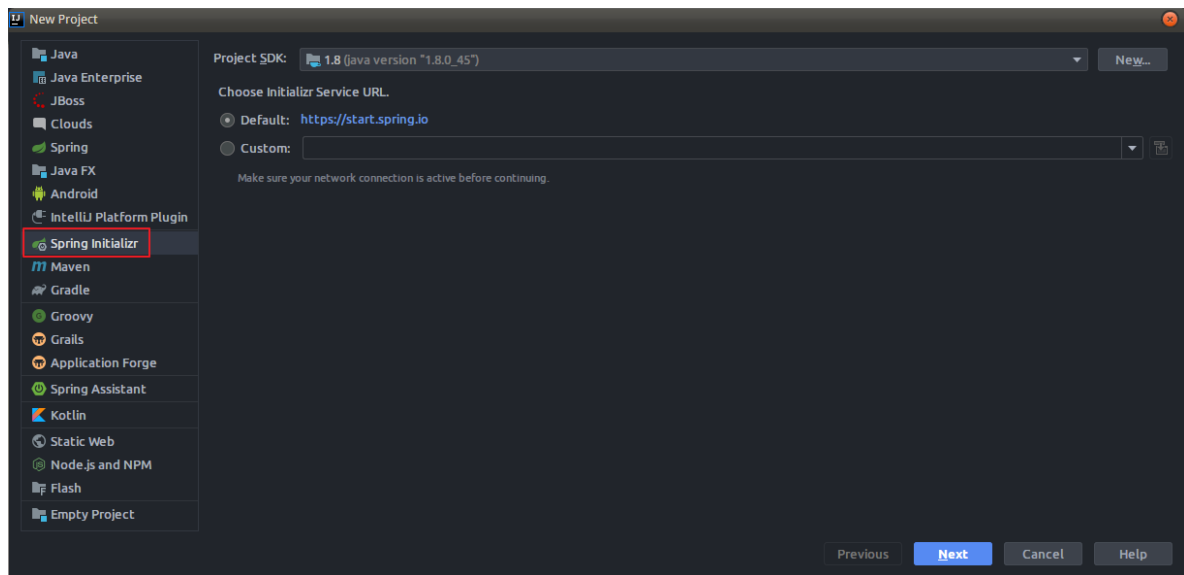
```

```

12 @RequestMapping("/user")
13 public class UserController {
14
15 @PostMapping("/add")
16 @ResponseBody
17 public User add(@RequestBody User user) {
18 System.out.println(user);
19 return user;
20 }
21 }

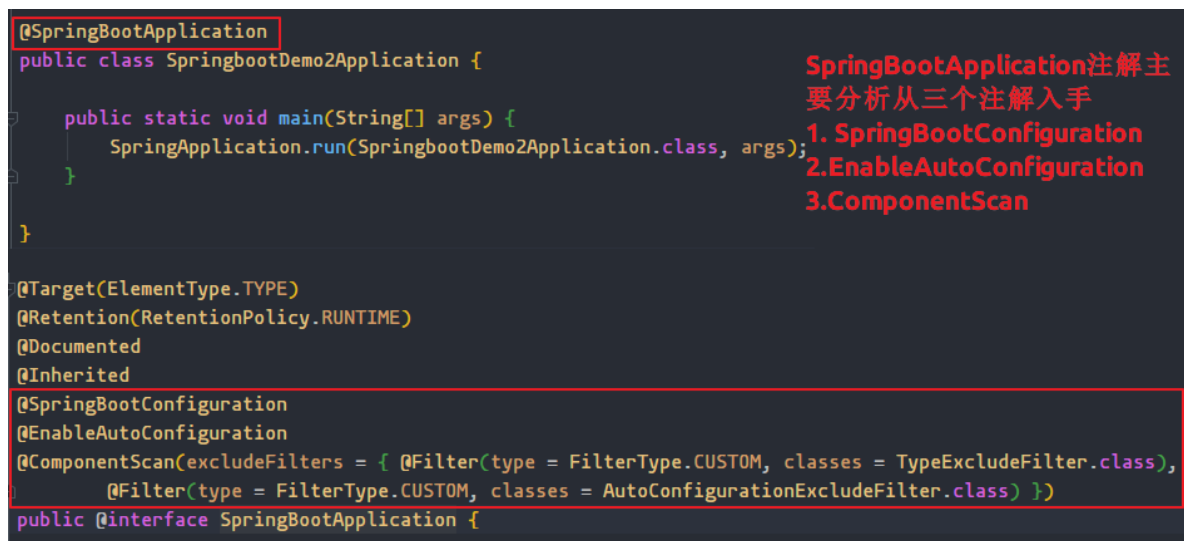
```

## 2.2 向导开发



## 三.分析启动类

### 3.1 分析@SpringBootApplication注解



#### 3.1.1 @SpringBootConfiguration

```

@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Configuration
public @interface SpringBootConfiguration {
 @AliasFor(
 annotation = Configuration.class
)
 boolean proxyBeanMethods() default true;
}

```

- 实际上就是一个配置类

### 3.1.2 @EnableAutoConfiguration

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({AutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {

 /**
 * {@link ImportBeanDefinitionRegistrar} to store the base package from the importing
 * configuration.
 */
 static class Registrar implements ImportBeanDefinitionRegistrar, DeterminableImports {

 @Override
 public void registerBeanDefinitions(AnnotationMetadata metadata, BeanDefinitionRegistry registry) {
 register(registry, new PackageImport(metadata).getPackageName());
 }

 @Override
 public Set<Object> determineImports(AnnotationMetadata metadata) {
 return Collections.singleton(new PackageImport(metadata));
 }
 }

 @Override
 public String[] selectImports(AnnotationMetadata annotationMetadata) {
 if (!isEnabled(annotationMetadata)) {
 return NO_IMPORTS;
 }
 AutoConfigurationMetadata autoConfigurationMetadata = AutoConfigurationMetadataLoader
 .loadMetadata(this.beanClassLoader);
 AutoConfigurationEntry autoConfigurationEntry = getAutoConfigurationEntry(autoConfigurationMetadata,
 annotationMetadata);
 }
}

```

将所有自动配置组件纳入到Spring容器中  
形如：XxxAutoConfiguration对象

将启动类所在包以及子包中的所有组件纳入到Spring容器

## 四.配置

### 4.1 配置方式

#### 4.1.1 application.properties

#### 4.1.2 application.yml

- yaml概述
  - yaml全称：YAML ain't markup language
  - YAML以数据为中心

- 1 | YAML (/ˈjæməl/, 尾音类似camel骆驼) 是一个可读性高, 用来表达数据序列化的格式。YAML 参考了其他多种语言, 包括: C语言、Python、Perl, 并从XML、电子邮件的数据格式(RFC 2822) 中获得灵感。Clark Evans在2001年首次发表了这种语言, 另外Ingy döt Net与Oren Ben-Kiki也是这语言的共同设计者。当前已经有数种编程语言或脚本语言支持(或者说解析) 这种语言。
- 2 | YAML是"YAML Ain't a Markup Language" (YAML不是一种标记语言) 的递归缩写。在开发的这种语言时, YAML 的意思其实是: "Yet Another Markup Language" (仍是一种标记语言), 但为了强调这种语言以数据做为中心, 而不是以标记语言为重点, 而用反向缩略语重命名。

- 语法 (重点)

- 键值, 键与值之间必须有空格

```
1 | k:(空格)v
```

- 以两个空格的缩进控制层次, 左对齐的键属于同一级
- 值的相关问题
  - 字面量。如果值中有转义字符, 那么应该怎么写? 此时需要用到引号来解决。单引号 (不会转义) 和双引号 (会转义)

```
person:
 name: "zhan \n gsan"

Person(name=zhan
gsan)
```

```
person:
 name: 'zhan \n gsan'

Person(name=zhan \n gsan)
```

- 对象

```
1 | ###第一种写法
2 | user:
3 | name: zhangsan
4 | age: 23
5 |
6 | ###第二种写法
7 | user: {userName: "zhangsan", age: 23}
```

- 集合

```
1 #user:
2 # name: zhangsan
3 # age: 23
4 # inters:
5 # - football
6 # - basketball
7 # - volleyball
8
9 user:
10 name: zhangsan
11 age: 23
12 inters: ["a","b","c"]
```

## 4.2 profile

### 4.2.1 多profile文件

- 配置文件可以在不同的环境（dev，test，prod...）下有多个存在
- 配置文件对应着有多个，配置文件的命名规范：application-{profile}.xml

### 4.2.2 yaml支持多文档化

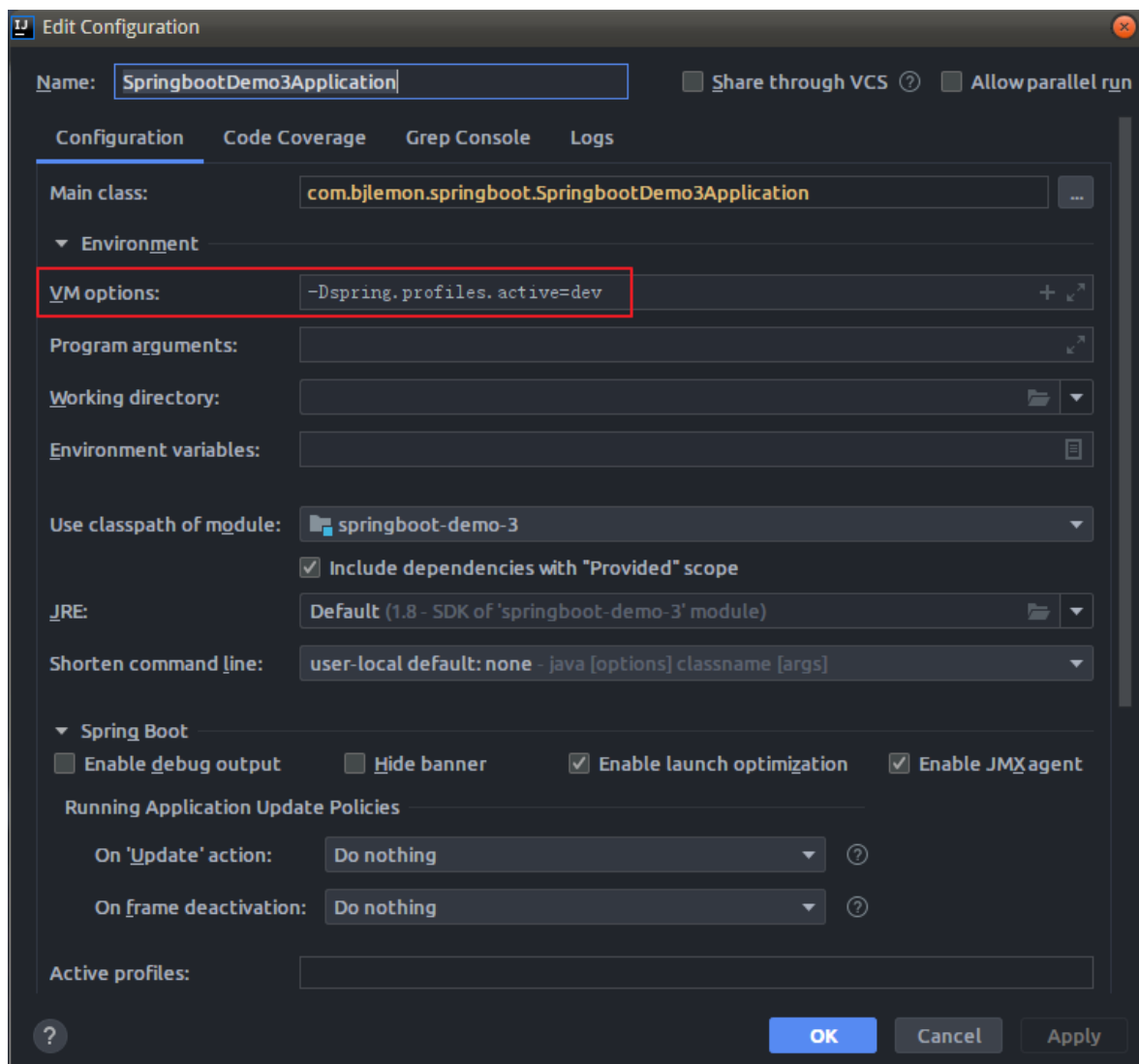
```
1 server:
2 port: 8888
3 spring:
4 profiles: dev
5
6 ---
7
8 server:
9 port: 9999
10 spring:
11 profiles: prod
```

### 4.2.3 如何激活profile

- 第一种激活方式

```
1 spring:
2 profiles:
3 active: prod
```

- 第二种激活方式



- 第三种运行方式

```
1 | java -jar springboot-demo-3-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod
```

## 4.3 配置文件的加载问题

### 4.3.1 加载顺序问题

```
1 | ./config/
2 | ./
3 | classpath:/config/
4 | classpath:/
```

- 优先级由高到低
- 原则：配置互补配置，将所有的配置全部加载，相同配置高优先级会覆盖低优先级

### 4.3.2 可以通过制定加载哪一个配置文件

```
1 | java -jar springboot-demo-3-0.0.1-SNAPSHOT.jar --
 | spring.config.location=classpath:/config/
```



# 五.日志系统

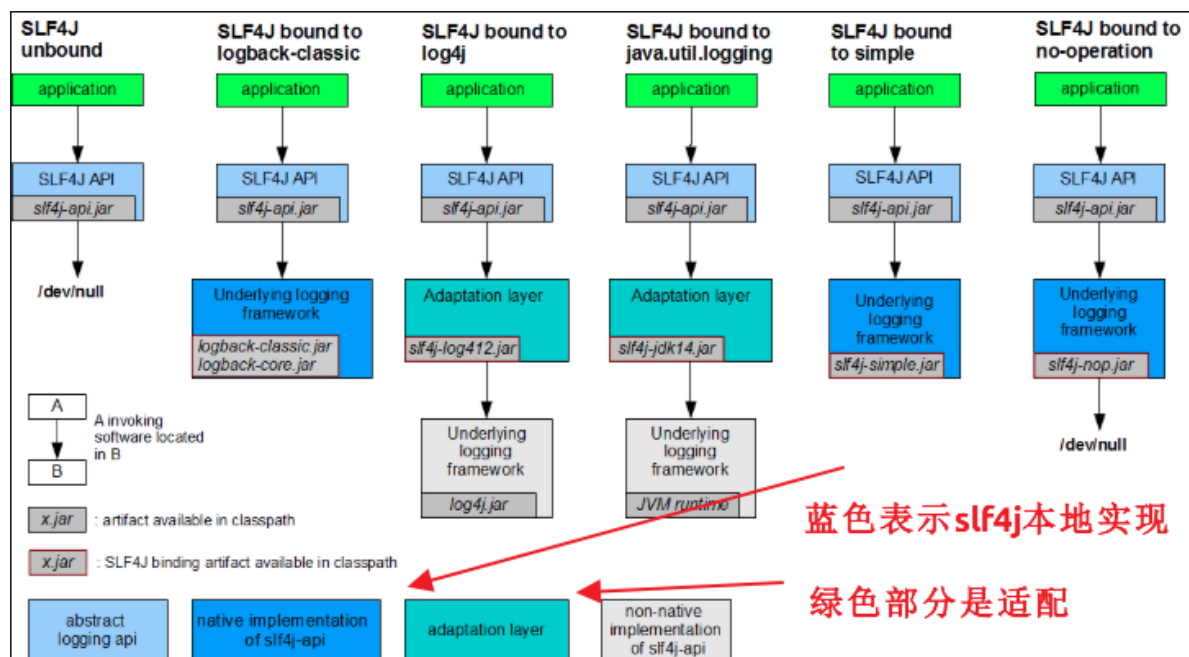
## 5.1 slf4j

- 全称 : simple logging facade for java
- <http://www.slf4j.org/>

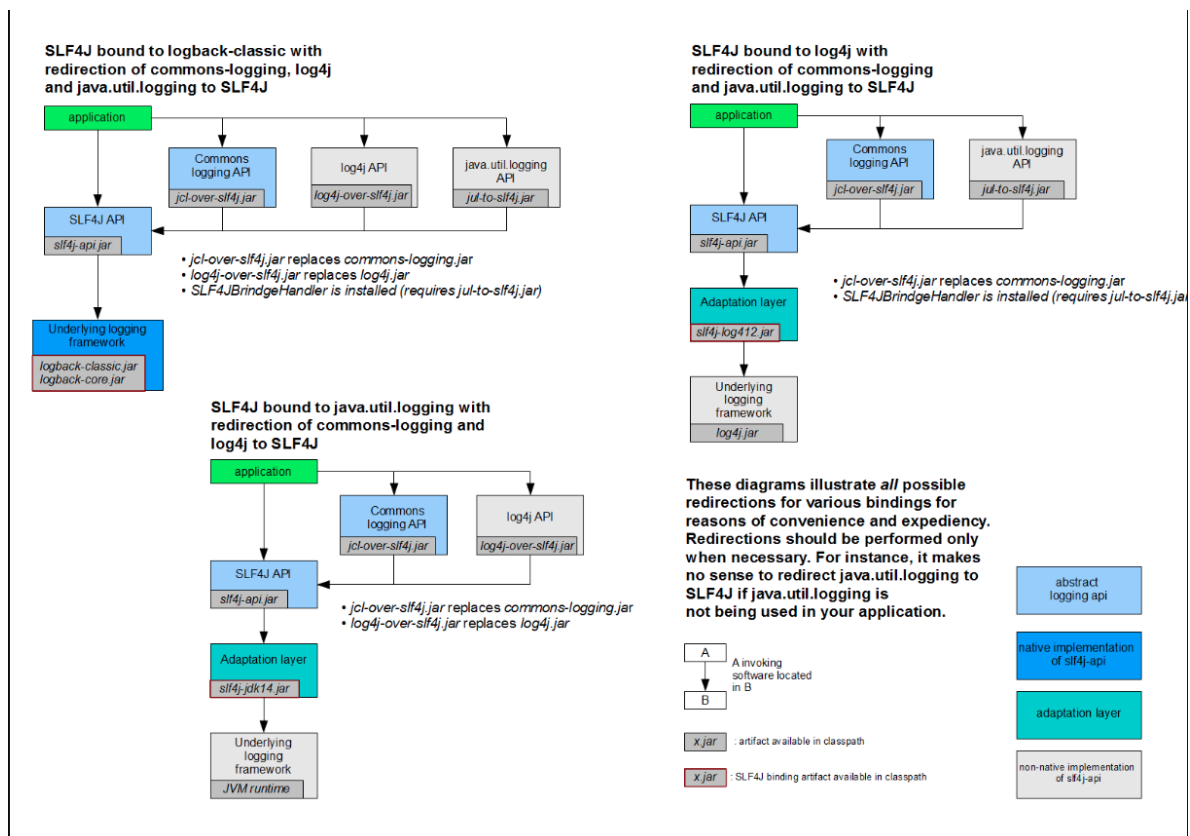
1 The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at deployment time.

- springboot可以跟很多的第三个框架进行集成，而这些第三方的框架所使用的日志产品各不相同，如何统一？

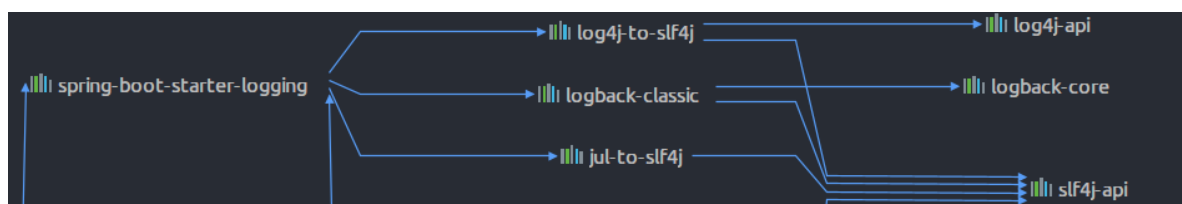
## 5.2 slf4j工作原理



## 5.3 slf4j如何统一其他产品



## 5.3 SpringBoot如何实现日志

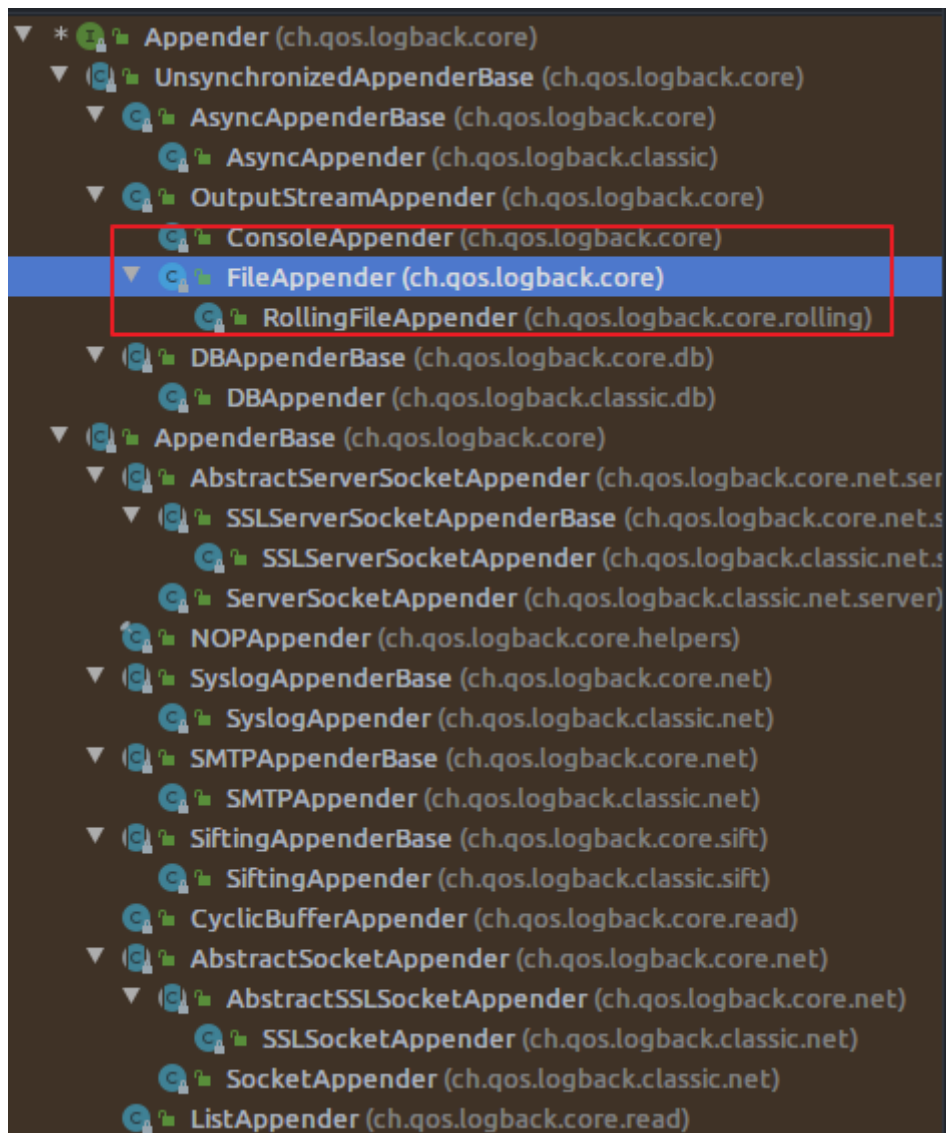


### 5.3.1 logback

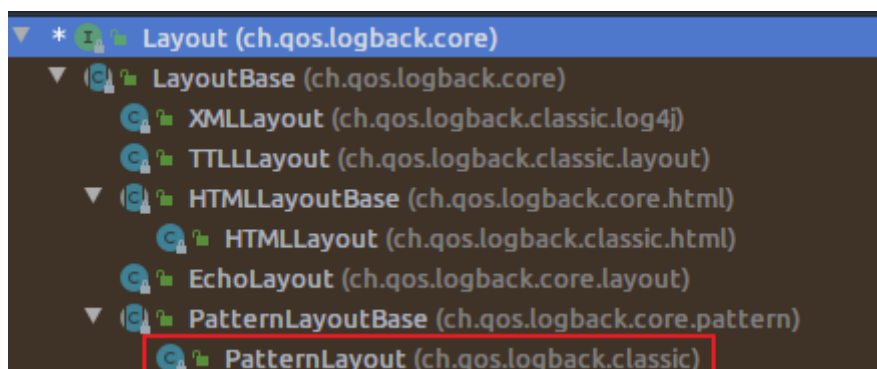
- <http://logback.qos.ch/>

- Logback is intended as a successor to the popular log4j project, picking up where log4j leaves off.
- 
- Logback's architecture is sufficiently generic so as to apply under different circumstances. At present time, logback is divided into three modules, logback-core, logback-classic and logback-access.
- 
- The logback-core module lays the groundwork for the other two modules. The logback-classic module can be assimilated to a significantly improved version of log4j. Moreover, logback-classic natively implements the SLF4J API so that you can readily switch back and forth between logback and other logging frameworks such as log4j or java.util.logging (JUL).
- 
- The logback-access module integrates with Servlet containers, such as Tomcat and Jetty, to provide HTTP-access log functionality. Note that you could easily build your own module on top of logback-core.

- Appender



- Layout



### 5.3.2 默认实现

```
1 logging:
2 level:
3 root: debug
4 # file: c:/springboot.log
```

#### 5.3.3 指定配置

- 类路径下放置一个自定义的日志配置文件
- logback-xxx.xml或logback.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration debug="true" scan="true" scanPeriod="1 seconds">
3
4 <contextName>logback</contextName>
5 <!--定义参数,后面可以通过${app.name}使用-->
6 <property name="app.name" value="logback_test"/>
7 <!--ConsoleAppender 用于在屏幕上输出日志-->
8 <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
9 <!--定义了一个过滤器,在LEVEL之下的日志输出不会被打印出来-->
10 <!--这里定义了DEBUG, 也就是控制台不会输出比ERROR级别小的日志-->
11 <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
12 <level>DEBUG</level>
13 </filter>
14 <!-- encoder 默认配置为PatternLayoutEncoder -->
15 <!--定义控制台输出格式-->
16 <encoder>
17 <pattern>%d [%thread] %-5level %logger{36} [%file : %line] -
18 %msg%n</pattern>
19 </encoder>
20 </appender>
21
22 <appender name="file"
23 class="ch.qos.logback.core.rolling.RollingFileAppender">
24 <!--定义日志输出的路径-->
25 <!--这里的scheduler.manager.server.home 没有在上面的配置中设定, 所以会使
26 用java启动时配置的值-->
27 <!--比如通过 java -Dscheduler.manager.server.home=/path/to xxxx 配置该
28 属性-->
29 <file>${scheduler.manager.server.home}/logs/${app.name}.log</file>
30 <!--定义日志滚动的策略-->
31 <rollingPolicy
32 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
33 <!--定义文件滚动时的文件名的格式-->
34
35 <fileNamePattern>${scheduler.manager.server.home}/logs/${app.name}.%d{yyyy-
36 MM-dd.HH}.log.gz
37 </fileNamePattern>
38 <!--60天的时间周期, 日志量最大20GB-->
39 <maxHistory>60</maxHistory>
40 <!-- 该属性在 1.1.6版本后 才开始支持-->
41 <totalSizeCap>20GB</totalSizeCap>
42 </rollingPolicy>
43 <triggeringPolicy
44 class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
45 <!--每个日志文件最大100MB-->
46 <maxFileSize>100MB</maxFileSize>
47 </triggeringPolicy>
48 <!--定义输出格式-->
49 <encoder>
50 <pattern>%d [%thread] %-5level %logger{36} [%file : %line] -
51 %msg%n</pattern>
52 </encoder>
53 </appender>
54
55 <!--root是默认的logger 这里设定输出级别是debug-->
56 <root level="trace">
57 <!--定义了两个appender, 日志会通过往这两个appender里面写-->
58 <appender-ref ref="stdout"/>

```

```

50 <appender-ref ref="file"/>
51 </root>
52
53 <!--对于类路径以 com.example.logback 开头的Logger,输出级别设置为warn,并且只输出到控制台-->
54 <!--这个logger没有指定appender, 它会继承root节点中定义的那些appender-->
55 <logger name="com.example.logback" level="warn"/>
56
57 <!--通过 LoggerFactory.getLogger("mytest") 可以获取到这个logger-->
58 <!--由于这个logger自动继承了root的appender, root中已经有stdout的appender了, 自己这边又引入了stdout的appender-->
59 <!--如果没有设置 additivity="false" ,就会导致一条日志在控制台输出两次的情况-->
60 <!--additivity表示要不要使用rootLogger配置的appender进行输出-->
61 <logger name="mytest" level="info" additivity="false">
62 <appender-ref ref="stdout"/>
63 </logger>
64
65 <!--由于设置了 additivity="false" , 所以输出时不会使用rootLogger的appender-->
66 <!--但是这个logger本身又没有配置appender, 所以使用这个logger输出日志的话就不会输出到任何地方-->
67 <logger name="mytest2" level="info" additivity="false"/>
68 </configuration>

```

## 六.SpringBoot与Web集成

### 6.0 SpringBoot如何集成

- XXXWebXXXAutoConfiguration
- WebMvcProperties , ResourceProperties

### 6.1 静态资源映射规则

- 定义静态资源的路径

```
1 spring.resources.static-locations=classpath:/demo/
```

- 如果按照上述配置了静态资源的路径, 那么默认的静态资源的路径就失效。因此建议: 如果真的配置该项, 那么建议将默认四个路径也加上。

```

1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-web-demo
6 resources:
7 static-locations:
8 - classpath:/META-INF/resources/
9 - classpath:/resources/
10 - classpath:/static/
11 - classpath:/public/
12 - classpath:/demo/

```

- 分析ResourceProperties源码

```
@ConfigurationProperties(prefix = "spring.resources", ignoreUnknownFields = false)
public class ResourceProperties {

 private static final String[] CLASSPATH_RESOURCE_LOCATIONS = { "classpath:/META-INF/resources/",
 "classpath:/resources/", "classpath:/static/", "classpath:/public/" };

 /**
 * Locations of static resources. Defaults to classpath:[/META-INF/resources/,
 * /resources/, /static/, /public/].
 */
 private String[] staticLocations = CLASSPATH_RESOURCE_LOCATIONS;
}
```

- 分析WebMvcAutoConfiguration

```
protected void addResourceHandlers(ResourceHandlerRegistry registry) {
 this.configurers.addResourceHandlers(registry);
}

public void addResourceHandlers(ResourceHandlerRegistry registry) {
 Iterator var2 = this.delegates.iterator();

 while(var2.hasNext()) {
 WebMvcConfigurer delegate = (WebMvcConfigurer)var2.next();
 delegate.addResourceHandlers(registry);
 }
}

@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
 if (!this.resourceProperties.isAddMappings()) {
 logger.debug("Default resource handling disabled");
 return;
 }

 Duration cachePeriod = this.resourceProperties.getCache().getPeriod();
 CacheControl cacheControl = this.resourceProperties.getCache().getCacheControl().toHttpCacheControl();
 if (!registry.hasMappingForPattern(pathPattern: "/webjars/**")) {
 customizeResourceHandlerRegistration(registry.addResourceHandler(pathPatterns: "/webjars/**")
 .addResourceLocations("classpath:/META-INF/resources/webjars/")
 .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
 }

 String staticPathPattern = this.mvcProperties.getStaticPathPattern();
 if (!registry.hasMappingForPattern(staticPathPattern)) {
 customizeResourceHandlerRegistration(registry.addResourceHandler(staticPathPattern)
 .addResourceLocations(getResourceLocations(this.resourceProperties.getStaticLocations()))
 .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl));
 }
}
```

- 所有的/webjars/\*\*都会在classpath:/META-INF/resources/webjars/下去找到对应的静态资源

## 6.2 SpringBoot与JSP集成（非重点）

### 6.2.1 简介

- JSP性能较差，占用带宽比较大。我们后续的项目开发建议不要使用JSP。
- SpringBoot官方团队建议使用模板技术（Freemarker, Thymeleaf）

### 6.2.2 如何实现

- 编写pom.xml

```
1 <dependency>
2 <groupId>org.apache.tomcat.embed</groupId>
3 <artifactId>tomcat-embed-jasper</artifactId>
4 </dependency>
```

- 编写配置文件

```
1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-jsp-dmeo
6 mvc:
7 view:
8 prefix: /WEB-INF/views/
9 suffix: .jsp
```

- 编写Controller

```
1 @Controller
2 @RequestMapping("/jsp")
3 public class JspController {
4
5 @GetMapping("/helloworld")
6 public String helloworld(Model model) {
7 model.addAttribute("info", "SpringBoot整合JSP");
8 return "helloworld";
9 }
10 }
```

- 编写jsp

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <html>
3 <head>
4 <title>Title</title>
5 </head>
6 <body>
7 ${info}
8 </body>
9 </html>
```

## 6.3 SpringBoot与Servlet、Filter以及Listener集成

### 6.3.1 第一种集成方式

```

1 @SpringBootApplication
2 @ServletComponentScan
3 public class SpringbootServletDemoApplication {
4
5 public static void main(String[] args) {
6 SpringApplication.run(SpringbootServletDemoApplication.class, args);
7 }
8 }

```

### 6.3.2 第二种集成方式

```

1 @Configuration
2 public class WebServletConfiguration implements ServletContextInitializer {
3
4 @Override
5 public void onStartup(ServletContext servletContext) throws
ServletException {
6 ServletRegistration.Dynamic helloworldServlet =
servletContext.addServlet("HelloworldServlet", HelloworldServlet.class);
7 helloworldServlet.addMapping("/helloworld", "/demo");
8 helloworldServlet.setLoadOnStartup(3);
9 helloworldServlet.setInitParameter("username", "zhangsan");
10 helloworldServlet.setInitParameter("password", "admin");
11
12 FilterRegistration.Dynamic helloworldFilter =
servletContext.addFilter("HelloworldFilter", HelloworldFilter.class);
13
14 helloworldFilter.addMappingForUrlPatterns(EnumSet.of(DispatcherType.REQUEST
), true, "/*");
15
16 servletContext.addListener(HelloworldListener.class);
17 }
18 }

```

### 6.3.3 第三种集成方式

```

1 @Configuration
2 public class WebServletConfiguration1 {
3
4 @Bean
5 public ServletRegistrationBean<HelloworldServlet>
servletServletRegistrationBean() {
6 ServletRegistrationBean<HelloworldServlet> servletRegistrationBean =
new ServletRegistrationBean<>();
7 servletRegistrationBean.setServlet(new HelloworldServlet());
8 servletRegistrationBean.addUrlMappings("/helloworld", "/demo");
9 return servletRegistrationBean;
10 }
11
12 @Bean
13 public FilterRegistrationBean<HelloworldFilter>
filterFilterRegistrationBean() {
14 FilterRegistrationBean<HelloworldFilter> filterRegistrationBean =
new FilterRegistrationBean<>();
15 filterRegistrationBean.setFilter(new HelloworldFilter());

```



```

16 filterRegistrationBean.setUrlPatterns(Arrays.asList("/*"));
17 return filterRegistrationBean;
18 }
19
20 @Bean
21 public ServletListenerRegistrationBean<HelloWorldListener>
22 servletListenerRegistrationBean() {
23 ServletListenerRegistrationBean<HelloWorldListener>
24 servletListenerRegistrationBean = new ServletListenerRegistrationBean<>();
25 servletListenerRegistrationBean.setListener(new
26 HelloWorldListener());
27 return servletListenerRegistrationBean;
28 }
29 }

```

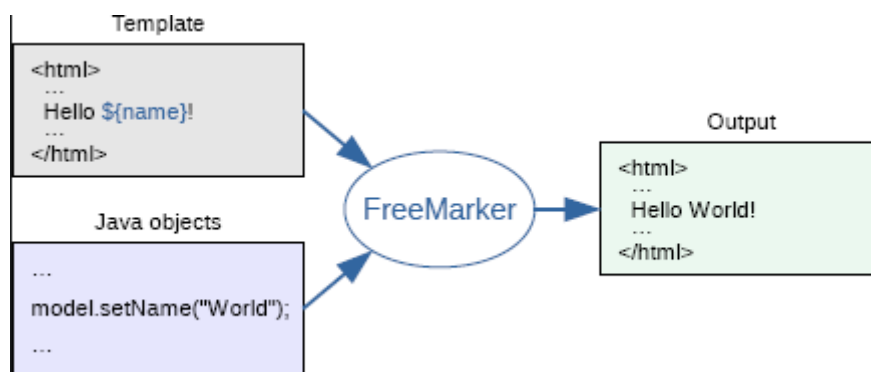
## 6.4 SpringBoot与模板的集成

### 6.4.0 模板技术概述

- 模板引擎：为了使用户界面与业务数据分离。最终生成特定格式的文档，用于网站生成一个静态页面（HTML文档）
- 模板就是用户界面+业务数据=结果（静态页面方式输出）
- 模板比JSP更轻量级，而且性能更好，渲染效率高，不占网络带宽
- 优秀的模板框架
  - Freemarker
  - Velocity
  - Thymeleaf

### 6.4.1 Freemarker

- <https://freemarker.apache.org/>



- 简介

```
1 Apache FreeMarker™ is a template engine: a Java library to generate text
 output (HTML web pages, e-mails, configuration files, source code, etc.)
 based on templates and changing data. Templates are written in the
 FreeMarker Template Language (FTL), which is a simple, specialized
 language (not a full-blown programming language like PHP). Usually, a
 general-purpose programming language (like Java) is used to prepare the
 data (issue database queries, do business calculations). Then, Apache
 FreeMarker displays that prepared data using templates. In the template
 you are focusing on how to present the data, and outside the template
 you are focusing on what data to present.
```

## 6.4.2 SpringBoot与Freemarker

- 编写pom.xml

```
1 <dependency>
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-starter-freemarker</artifactId>
4 </dependency>
```

- 编写application.yml配置文件

```
1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-freemarker-demo
6 freemarker:
7 suffix: .ftl
8 enabled: true
9 cache: false
10 template-loader-path: classpath:/templates/
```

## 6.4.2 Thymeleaf ( 重点 )

- 概述

```
1 Thymeleaf is a modern server-side Java template engine for both web and
 standalone environments.
2
3 Thymeleaf's main goal is to bring elegant natural templates to your
 development workflow – HTML that can be correctly displayed in browsers and
 also work as static prototypes, allowing for stronger collaboration in
 development teams.
4
5 with modules for Spring Framework, a host of integrations with your favourite
 tools, and the ability to plug in your own functionality, Thymeleaf is ideal
 for modern-day HTML5 JVM web development – although there is much more it can
 do
```

- 适用于Web环境和独立JVM环境
- Java服务器端模板引擎

- 如果服务器端没有提供数据，那么就以静态页面显示，但是如果提供了数据那么就用服务器端数据来替换掉静态数据
- Thymeleaf与前端框架（Vue）区别
  - vue异步请求数据，后端给前端发送数据（json）。然后前端vue的指令进行解析和渲染。页面的展现可能会产生延迟，而且数据爬虫，搜索引擎抓取不到异步加载的数据
  - thymeleaf是一种后端页面的渲染，然后在浏览器上显示。以静态页面展现基本不会产生延迟，而且搜索引擎能抓取数据
- 标签

| 标签名称      | 功能             |
|-----------|----------------|
| th:text   | 文本内容显示         |
| th:utext  | 支持html文本       |
| th:id     | 替换id           |
| th:each   | 循环             |
| th:if     | 条件分支           |
| th:switch | 条件分支           |
| th:case   | 与th:switch搭配使用 |
| th:value  | 属性赋值           |

- springboot与thymeleaf集成

```

1 <dependency>
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-starter-thymeleaf</artifactId>
4 </dependency>

```

```

1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-thymeleaf-demo
6 thymeleaf:
7 enabled: true
8 encoding: UTF-8
9 cache: false

```

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="UTF-8">
5 <title>${title}</title>
6 </head>
7 <body>
8 <table border="1" cellpadding="0" cellspacing="0">
9 <thead>
10 <tr>
11 <th>用户编号</th>

```

```

12 <th>用户名称</th>
13 <th>用户密码</th>
14 <th>用户薪资</th>
15 <th>用户生日</th>
16 </tr>
17 </thead>
18
19 <tbody>
20 <tr th:each="person : ${personList}">
21 <td th:text="${person.id}"></td>
22 <td th:text="${person.name}"></td>
23 <td th:text="${person.password}"></td>
24 <td th:text="${person.salary}"></td>
25 <td th:text="${#dates.format(person.birthday, 'yyyy-MM-dd')}"></td>
26 </tr>
27 </tbody>
28 </table>
29 </body>
30 </html>

```

- 分析Thymeleaf源码

```

@ConfigurationProperties(prefix = "spring.thymeleaf")
public class ThymeleafProperties {

 private static final Charset DEFAULT_ENCODING = StandardCharsets.UTF_8;

 public static final String DEFAULT_PREFIX = "classpath:/templates/";

 public static final String DEFAULT_SUFFIX = ".html";
}

```

- Thymeleaf语法
- 表达式

```

1 Variable Expressions: ${...}
2 Selection Variable Expressions: *{...}
3 Message Expressions: #{...}
4 Link URL Expressions: @{...}
5 Fragment Expressions: ~{...}

```

- Expression Basic Objects

```

1 #ctx : the context object.
2 #vars: the context variables.
3 #locale : the context locale.
4 #request : (only in Web Contexts) the HttpServletRequest object.
5 #response : (only in Web Contexts) the HttpServletResponse object.
6 #session : (only in Web Contexts) the HttpSession object.
7 #servletContext : (only in Web Contexts) the ServletContext object.

```

- Expression Utility Objects

```

1 #execInfo : information about the template being processed.
2 #messages : methods for obtaining externalized messages inside variables
 expressions, in the same way as they would be obtained using #{...} syntax.

```

```

3 #uris : methods for escaping parts of URLs/URIs
4 #conversions : methods for executing the configured conversion service
 (if any).
5 #dates : methods for java.util.Date objects: formatting, component
 extraction, etc.
6 #calendars : analogous to #dates , but for java.util.Calendar objects.
7 #numbers : methods for formatting numeric objects.
8 #strings : methods for String objects: contains, startsWith,
 prepending/appending, etc.
9 #objects : methods for objects in general.
10 #booleans : methods for boolean evaluation.
11 #arrays : methods for arrays.
12 #lists : methods for lists.
13 #sets : methods for sets.
14 #maps : methods for maps.
15 #aggregates : methods for creating aggregates on arrays or collections.
16 #ids : methods for dealing with id attributes that might be repeated (for
 example, as a result of an iteration).

```

## 6.5 SpringBoot与SpringMVC集成原理（自动配置原理）

### 6.5.1 自动配置如何实现

- <https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html#boot-features-developing-web-applications>

```

1 Inclusion of ContentNegotiatingViewResolver and BeanNameViewResolver beans.
2 Support for serving static resources, including support for webJars (covered
 later in this document)).
3 Automatic registration of Converter, GenericConverter, and Formatter beans.
4 Support for HttpMessageConverters (covered later in this document).
5 Automatic registration of MessageCodesResolver (covered later in this
 document).
6 Static index.html support.
7 Custom Favicon support (covered later in this document).
8 Automatic use of a ConfigurableWebBindingInitializer bean (covered later in
 this document).

```

- WebMvcAutoConfiguration。这个类中实现了一些默认配置，比如视图解析器，转换器、格式化器，静态资源的映射...
- 问题：我们需要定制一些配置，那么应该怎么办？
  - 编写配置类，继承WebMvcConfigurerAdapter或实现WebMvcConfigurer，此时不能在这个配置类上加上@EnableWebMvc
  - 如果想完全掌管SpringMVC（不想使用SpringBoot与SpringMVC的一些默认配置），此时就需要加上@Configuration和@EnableWebMvc注解

```

1 If you want to keep Spring Boot MVC features and you want to add additional
 MVC configuration (interceptors, formatters, view controllers, and other
 features), you can add your own @Configuration class of type WebMvcConfigurer
 but without @EnableWebMvc. If you wish to provide custom instances of
 RequestMappingHandlerMapping, RequestMappingHandlerAdapter, or
 ExceptionHandlerExceptionResolver, you can declare a
 WebMvcRegistrationsAdapter instance to provide such components.

```

- 为什么加上了@EnableWebMvc，自动配置就失效？

```

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
@Import({DelegatingWebMvcConfiguration.class})
public @interface EnableWebMvc {
}

@Configuration(proxyBeanMethods = false)
public class DelegatingWebMvcConfiguration extends WebMvcConfigurationSupport {

 .
 .
 .
 @Configuration(proxyBeanMethods = false)
 @ConditionalOnWebApplication(type = Type.SERVLET)
 @ConditionalOnClass({ Servlet.class, DispatcherServlet.class, WebMvcConfigurer.class })
 @ConditionalOnMissingBean(WebMvcConfigurationSupport.class)
 @AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE + 10)
 @AutoConfigureAfter({ DispatcherServletAutoConfiguration.class, TaskExecutionAutoConfiguration.class,
 ValidationAutoConfiguration.class })
 public class WebMvcAutoConfiguration {

```

1. 如果加上了@EnableWebMvc注解，那么意味着向容器中加入一个组件（DelegatingWebMvcConfiguration）。而这个对象本质上就是一个WebMvcConfigurationSupport类型的对象

2. SpringBoot与SpringMVC自动配置是需要有WebMvcAutoConfiguration来支持，但是必须满足：容器中要没有WebMvcConfigurationSupport对象，自动配置才会生效

## 七. SpringBoot与持久化层集成

### 7.1 SpringBoot与MyBatis

#### 7.1.1 逆向工程

- 导入项目（generatorSqlmapCustom），删除掉src下的所有的包以及类（GeneratorSqlmap类不能删）
- 修改generatorConfig.xml文件

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE generatorConfiguration
3 PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
4 "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
5
6 <generatorConfiguration>
7 <context id="testTables" targetRuntime="MyBatis3">
8 <commentGenerator>
9 <!-- 是否去除自动生成的注释 true: 是 : false:否 -->
10 <property name="suppressAllComments" value="true"/>
11 </commentGenerator>
12 <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->
13 <jdbcConnection driverClass="com.mysql.jdbc.Driver"
14 connectionURL="jdbc:mysql://localhost:3306/springboot-mybatis"
15 userId="root"
16 password="root">
17 </jdbcConnection>
18 <!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，
19 为true时把JDBC DECIMAL和NUMERIC类型解析为java.math.BigDecimal -->
20 <javaTypeResolver>
21 <property name="forceBigDecimals" value="false"/>
22 </javaTypeResolver>
23
24 <!-- targetProject:生成PO类的位置 -->
25 <javaModelGenerator targetPackage="com.bjlemon.springboot.domain"
26 targetProject=".\\src">
27 <!-- enableSubPackages:是否让schema作为包的后缀 -->
28 <property name="enableSubPackages" value="false"/>
29 <!-- 从数据库返回的值被清理前后的空格 -->

```

```

30 <property name="trimStrings" value="true"/>
31 </javaModelGenerator>
32 <!-- targetProject: mapper映射文件生成的位置 -->
33 <sqlMapGenerator targetPackage="com.bjlemon.springboot.mapper"
34 targetProject=".\\src">
35 <!-- enableSubPackages: 是否让schema作为包的后缀 -->
36 <property name="enableSubPackages" value="false"/>
37 </sqlMapGenerator>
38 <!-- targetPackage: mapper接口生成的位置 -->
39 <javaClientGenerator type="XMLMAPPER"
40 targetPackage="com.bjlemon.springboot.mapper"
41 targetProject=".\\src">
42 <!-- enableSubPackages: 是否让schema作为包的后缀 -->
43 <property name="enableSubPackages" value="false"/>
44 </javaClientGenerator>
45 <!-- 指定数据库表 -->
46 <table schema="" tableName="springboot_mybatis_user"
47 domainObjectName="User"/>
48 <table schema="" tableName="springboot_mybatis_department"
49 domainObjectName="Department"/>
50 </context>
51 </generatorConfiguration>

```

- 运行GeneratorSqlmap类中的main()方法

## 7.1.2 编写pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://maven.apache.org/POM/4.0.0"
4 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5 https://maven.apache.org/xsd/maven-4.0.0.xsd">
6 <modelVersion>4.0.0</modelVersion>
7 <parent>
8 <groupId>org.springframework.boot</groupId>
9 <artifactId>spring-boot-starter-parent</artifactId>
10 <version>2.2.1.RELEASE</version>
11 <relativePath/> <!-- lookup parent from repository -->
12 </parent>
13 <groupId>com.bjlemon</groupId>
14 <artifactId>springboot-mybatis-demo</artifactId>
15 <version>0.0.1-SNAPSHOT</version>
16 <name>springboot-mybatis-demo</name>
17
18 <properties>
19 <java.version>1.8</java.version>
20 </properties>
21
22 <dependencies>
23 <dependency>
24 <groupId>org.springframework.boot</groupId>
25 <artifactId>spring-boot-starter-thymeleaf</artifactId>
26 </dependency>
27
28 <dependency>
29 <groupId>org.springframework.boot</groupId>
30 <artifactId>spring-boot-starter-web</artifactId>
31 </dependency>
32 </dependencies>
33
34 <build>
35 <plugins>
36 <plugin>
37 <groupId>org.springframework.boot</groupId>
38 <artifactId>spring-boot-maven-plugin</artifactId>
39 </plugin>
40 </plugins>
41 </build>
42
43 <test>
44 <resources>
45 <resource>
46 <directory>src/test/resources</directory>
47 </resource>
48 </resources>
49 </test>
50
51 <reporting>
52 <plugins>
53 <plugin>
54 <groupId>org.apache.maven.plugins</groupId>
55 <artifactId>maven-project-info-reports-plugin</artifactId>
56 </plugin>
57 </plugins>
58 </reporting>
59
60 <profiles>
61 <profile>
62 <id>dev</id>
63 <activation>
64 <activeByDefault>true</activeByDefault>
65 </activation>
66 <properties>
67 <spring.profiles.active>dev</spring.profiles.active>
68 </properties>
69 </profile>
70 </profiles>
71
72 <repositories>
73 <repository>
74 <id>central</id>
75 <url>https://repo.maven.apache.org/maven2</url>
76 </repository>
77 </repositories>
78
79 <pluginRepositories>
80 <pluginRepository>
81 <id>central</id>
82 <url>https://repo.maven.apache.org/maven2</url>
83 </pluginRepository>
84 </pluginRepositories>
85
86 <reporting>
87 <plugins>
88 <plugin>
89 <groupId>org.apache.maven.plugins</groupId>
90 <artifactId>maven-project-info-reports-plugin</artifactId>
91 </plugin>
92 </plugins>
93 </reporting>
94
95 <build>
96 <plugins>
97 <plugin>
98 <groupId>org.springframework.boot</groupId>
99 <artifactId>spring-boot-maven-plugin</artifactId>
100 </plugin>
101 </plugins>
102 </build>
103
104 <test>
105 <resources>
106 <resource>
107 <directory>src/test/resources</directory>
108 </resource>
109 </resources>
110 </test>
111
112 <reporting>
113 <plugins>
114 <plugin>
115 <groupId>org.apache.maven.plugins</groupId>
116 <artifactId>maven-project-info-reports-plugin</artifactId>
117 </plugin>
118 </plugins>
119 </reporting>
120
121 <profiles>
122 <profile>
123 <id>dev</id>
124 <activation>
125 <activeByDefault>true</activeByDefault>
126 </activation>
127 <properties>
128 <spring.profiles.active>dev</spring.profiles.active>
129 </properties>
130 </profile>
131 </profiles>
132
133 <repositories>
134 <repository>
135 <id>central</id>
136 <url>https://repo.maven.apache.org/maven2</url>
137 </repository>
138 </repositories>
139
140 <pluginRepositories>
141 <pluginRepository>
142 <id>central</id>
143 <url>https://repo.maven.apache.org/maven2</url>
144 </pluginRepository>
145 </pluginRepositories>
146
147 <reporting>
148 <plugins>
149 <plugin>
150 <groupId>org.apache.maven.plugins</groupId>
151 <artifactId>maven-project-info-reports-plugin</artifactId>
152 </plugin>
153 </plugins>
154 </reporting>
155
156 <build>
157 <plugins>
158 <plugin>
159 <groupId>org.springframework.boot</groupId>
160 <artifactId>spring-boot-maven-plugin</artifactId>
161 </plugin>
162 </plugins>
163 </build>
164
165 <test>
166 <resources>
167 <resource>
168 <directory>src/test/resources</directory>
169 </resource>
170 </resources>
171 </test>
172
173 <reporting>
174 <plugins>
175 <plugin>
176 <groupId>org.apache.maven.plugins</groupId>
177 <artifactId>maven-project-info-reports-plugin</artifactId>
178 </plugin>
179 </plugins>
180 </reporting>
181
182 <profiles>
183 <profile>
184 <id>dev</id>
185 <activation>
186 <activeByDefault>true</activeByDefault>
187 </activation>
188 <properties>
189 <spring.profiles.active>dev</spring.profiles.active>
190 </properties>
191 </profile>
192 </profiles>
193
194 <repositories>
195 <repository>
196 <id>central</id>
197 <url>https://repo.maven.apache.org/maven2</url>
198 </repository>
199 </repositories>
200
201 <pluginRepositories>
202 <pluginRepository>
203 <id>central</id>
204 <url>https://repo.maven.apache.org/maven2</url>
205 </pluginRepository>
206 </pluginRepositories>
207
208 <reporting>
209 <plugins>
210 <plugin>
211 <groupId>org.apache.maven.plugins</groupId>
212 <artifactId>maven-project-info-reports-plugin</artifactId>
213 </plugin>
214 </plugins>
215 </reporting>
216
217 <build>
218 <plugins>
219 <plugin>
220 <groupId>org.springframework.boot</groupId>
221 <artifactId>spring-boot-maven-plugin</artifactId>
222 </plugin>
223 </plugins>
224 </build>
225
226 <test>
227 <resources>
228 <resource>
229 <directory>src/test/resources</directory>
230 </resource>
231 </resources>
232 </test>
233
234 <reporting>
235 <plugins>
236 <plugin>
237 <groupId>org.apache.maven.plugins</groupId>
238 <artifactId>maven-project-info-reports-plugin</artifactId>
239 </plugin>
240 </plugins>
241 </reporting>
242
243 <profiles>
244 <profile>
245 <id>dev</id>
246 <activation>
247 <activeByDefault>true</activeByDefault>
248 </activation>
249 <properties>
250 <spring.profiles.active>dev</spring.profiles.active>
251 </properties>
252 </profile>
253 </profiles>
254
255 <repositories>
256 <repository>
257 <id>central</id>
258 <url>https://repo.maven.apache.org/maven2</url>
259 </repository>
260 </repositories>
261
262 <pluginRepositories>
263 <pluginRepository>
264 <id>central</id>
265 <url>https://repo.maven.apache.org/maven2</url>
266 </pluginRepository>
267 </pluginRepositories>
268
269 <reporting>
270 <plugins>
271 <plugin>
272 <groupId>org.apache.maven.plugins</groupId>
273 <artifactId>maven-project-info-reports-plugin</artifactId>
274 </plugin>
275 </plugins>
276 </reporting>
277
278 <build>
279 <plugins>
280 <plugin>
281 <groupId>org.springframework.boot</groupId>
282 <artifactId>spring-boot-maven-plugin</artifactId>
283 </plugin>
284 </plugins>
285 </build>
286
287 <test>
288 <resources>
289 <resource>
290 <directory>src/test/resources</directory>
291 </resource>
292 </resources>
293 </test>
294
295 <reporting>
296 <plugins>
297 <plugin>
298 <groupId>org.apache.maven.plugins</groupId>
299 <artifactId>maven-project-info-reports-plugin</artifactId>
300 </plugin>
301 </plugins>
302 </reporting>
303
304 <profiles>
305 <profile>
306 <id>dev</id>
307 <activation>
308 <activeByDefault>true</activeByDefault>
309 </activation>
310 <properties>
311 <spring.profiles.active>dev</spring.profiles.active>
312 </properties>
313 </profile>
314 </profiles>
315
316 <repositories>
317 <repository>
318 <id>central</id>
319 <url>https://repo.maven.apache.org/maven2</url>
320 </repository>
321 </repositories>
322
323 <pluginRepositories>
324 <pluginRepository>
325 <id>central</id>
326 <url>https://repo.maven.apache.org/maven2</url>
327 </pluginRepository>
328 </pluginRepositories>
329
330 <reporting>
331 <plugins>
332 <plugin>
333 <groupId>org.apache.maven.plugins</groupId>
334 <artifactId>maven-project-info-reports-plugin</artifactId>
335 </plugin>
336 </plugins>
337 </reporting>
338
339 <build>
340 <plugins>
341 <plugin>
342 <groupId>org.springframework.boot</groupId>
343 <artifactId>spring-boot-maven-plugin</artifactId>
344 </plugin>
345 </plugins>
346 </build>
347
348 <test>
349 <resources>
350 <resource>
351 <directory>src/test/resources</directory>
352 </resource>
353 </resources>
354 </test>
355
356 <reporting>
357 <plugins>
358 <plugin>
359 <groupId>org.apache.maven.plugins</groupId>
360 <artifactId>maven-project-info-reports-plugin</artifactId>
361 </plugin>
362 </plugins>
363 </reporting>
364
365 <profiles>
366 <profile>
367 <id>dev</id>
368 <activation>
369 <activeByDefault>true</activeByDefault>
370 </activation>
371 <properties>
372 <spring.profiles.active>dev</spring.profiles.active>
373 </properties>
374 </profile>
375 </profiles>
376
377 <repositories>
378 <repository>
379 <id>central</id>
380 <url>https://repo.maven.apache.org/maven2</url>
381 </repository>
382 </repositories>
383
384 <pluginRepositories>
385 <pluginRepository>
386 <id>central</id>
387 <url>https://repo.maven.apache.org/maven2</url>
388 </pluginRepository>
389 </pluginRepositories>
390
391 <reporting>
392 <plugins>
393 <plugin>
394 <groupId>org.apache.maven.plugins</groupId>
395 <artifactId>maven-project-info-reports-plugin</artifactId>
396 </plugin>
397 </plugins>
398 </reporting>
399
400 <build>
401 <plugins>
402 <plugin>
403 <groupId>org.springframework.boot</groupId>
404 <artifactId>spring-boot-maven-plugin</artifactId>
405 </plugin>
406 </plugins>
407 </build>
408
409 <test>
410 <resources>
411 <resource>
412 <directory>src/test/resources</directory>
413 </resource>
414 </resources>
415 </test>
416
417 <reporting>
418 <plugins>
419 <plugin>
420 <groupId>org.apache.maven.plugins</groupId>
421 <artifactId>maven-project-info-reports-plugin</artifactId>
422 </plugin>
423 </plugins>
424 </reporting>
425
426 <profiles>
427 <profile>
428 <id>dev</id>
429 <activation>
430 <activeByDefault>true</activeByDefault>
431 </activation>
432 <properties>
433 <spring.profiles.active>dev</spring.profiles.active>
434 </properties>
435 </profile>
436 </profiles>
437
438 <repositories>
439 <repository>
440 <id>central</id>
441 <url>https://repo.maven.apache.org/maven2</url>
442 </repository>
443 </repositories>
444
445 <pluginRepositories>
446 <pluginRepository>
447 <id>central</id>
448 <url>https://repo.maven.apache.org/maven2</url>
449 </pluginRepository>
450 </pluginRepositories>
451
452 <reporting>
453 <plugins>
454 <plugin>
455 <groupId>org.apache.maven.plugins</groupId>
456 <artifactId>maven-project-info-reports-plugin</artifactId>
457 </plugin>
458 </plugins>
459 </reporting>
460
461 <build>
462 <plugins>
463 <plugin>
464 <groupId>org.springframework.boot</groupId>
465 <artifactId>spring-boot-maven-plugin</artifactId>
466 </plugin>
467 </plugins>
468 </build>
469
470 <test>
471 <resources>
472 <resource>
473 <directory>src/test/resources</directory>
474 </resource>
475 </resources>
476 </test>
477
478 <reporting>
479 <plugins>
480 <plugin>
481 <groupId>org.apache.maven.plugins</groupId>
482 <artifactId>maven-project-info-reports-plugin</artifactId>
483 </plugin>
484 </plugins>
485 </reporting>
486
487 <profiles>
488 <profile>
489 <id>dev</id>
490 <activation>
491 <activeByDefault>true</activeByDefault>
492 </activation>
493 <properties>
494 <spring.profiles.active>dev</spring.profiles.active>
495 </properties>
496 </profile>
497 </profiles>
498
499 <repositories>
500 <repository>
501 <id>central</id>
502 <url>https://repo.maven.apache.org/maven2</url>
503 </repository>
504 </repositories>
505
506 <pluginRepositories>
507 <pluginRepository>
508 <id>central</id>
509 <url>https://repo.maven.apache.org/maven2</url>
510 </pluginRepository>
511 </pluginRepositories>
512
513 <reporting>
514 <plugins>
515 <plugin>
516 <groupId>org.apache.maven.plugins</groupId>
517 <artifactId>maven-project-info-reports-plugin</artifactId>
518 </plugin>
519 </plugins>
520 </reporting>
521
522 <build>
523 <plugins>
524 <plugin>
525 <groupId>org.springframework.boot</groupId>
526 <artifactId>spring-boot-maven-plugin</artifactId>
527 </plugin>
528 </plugins>
529 </build>
530
531 <test>
532 <resources>
533 <resource>
534 <directory>src/test/resources</directory>
535 </resource>
536 </resources>
537 </test>
538
539 <reporting>
540 <plugins>
541 <plugin>
542 <groupId>org.apache.maven.plugins</groupId>
543 <artifactId>maven-project-info-reports-plugin</artifactId>
544 </plugin>
545 </plugins>
546 </reporting>
547
548 <profiles>
549 <profile>
550 <id>dev</id>
551 <activation>
552 <activeByDefault>true</activeByDefault>
553 </activation>
554 <properties>
555 <spring.profiles.active>dev</spring.profiles.active>
556 </properties>
557 </profile>
558 </profiles>
559
560 <repositories>
561 <repository>
562 <id>central</id>
563 <url>https://repo.maven.apache.org/maven2</url>
564 </repository>
565 </repositories>
566
567 <pluginRepositories>
568 <pluginRepository>
569 <id>central</id>
570 <url>https://repo.maven.apache.org/maven2</url>
571 </pluginRepository>
572 </pluginRepositories>
573
574 <reporting>
575 <plugins>
576 <plugin>
577 <groupId>org.apache.maven.plugins</groupId>
578 <artifactId>maven-project-info-reports-plugin</artifactId>
579 </plugin>
580 </plugins>
581 </reporting>
582
583 <build>
584 <plugins>
585 <plugin>
586 <groupId>org.springframework.boot</groupId>
587 <artifactId>spring-boot-maven-plugin</artifactId>
588 </plugin>
589 </plugins>
590 </build>
591
592 <test>
593 <resources>
594 <resource>
595 <directory>src/test/resources</directory>
596 </resource>
597 </resources>
598 </test>
599
600 <reporting>
601 <plugins>
602 <plugin>
603 <groupId>org.apache.maven.plugins</groupId>
604 <artifactId>maven-project-info-reports-plugin</artifactId>
605 </plugin>
606 </plugins>
607 </reporting>
608
609 <profiles>
610 <profile>
611 <id>dev</id>
612 <activation>
613 <activeByDefault>true</activeByDefault>
614 </activation>
615 <properties>
616 <spring.profiles.active>dev</spring.profiles.active>
617 </properties>
618 </profile>
619 </profiles>
620
621 <repositories>
622 <repository>
623 <id>central</id>
624 <url>https://repo.maven.apache.org/maven2</url>
625 </repository>
626 </repositories>
627
628 <pluginRepositories>
629 <pluginRepository>
630 <id>central</id>
631 <url>https://repo.maven.apache.org/maven2</url>
632 </pluginRepository>
633 </pluginRepositories>
634
635 <reporting>
636 <plugins>
637 <plugin>
638 <groupId>org.apache.maven.plugins</groupId>
639 <artifactId>maven-project-info-reports-plugin</artifactId>
640 </plugin>
641 </plugins>
642 </reporting>
643
644 <build>
645 <plugins>
646 <plugin>
647 <groupId>org.springframework.boot</groupId>
648 <artifactId>spring-boot-maven-plugin</artifactId>
649 </plugin>
650 </plugins>
651 </build>
652
653 <test>
654 <resources>
655 <resource>
656 <directory>src/test/resources</directory>
657 </resource>
658 </resources>
659 </test>
660
661 <reporting>
662 <plugins>
663 <plugin>
664 <groupId>org.apache.maven.plugins</groupId>
665 <artifactId>maven-project-info-reports-plugin</artifactId>
666 </plugin>
667 </plugins>
668 </reporting>
669
670 <profiles>
671 <profile>
672 <id>dev</id>
673 <activation>
674 <activeByDefault>true</activeByDefault>
675 </activation>
676 <properties>
677 <spring.profiles.active>dev</spring.profiles.active>
678 </properties>
679 </profile>
680 </profiles>
681
682 <repositories>
683 <repository>
684 <id>central</id>
685 <url>https://repo.maven.apache.org/maven2</url>
686 </repository>
687 </repositories>
688
689 <pluginRepositories>
690 <pluginRepository>
691 <id>central</id>
692 <url>https://repo.maven.apache.org/maven2</url>
693 </pluginRepository>
694 </pluginRepositories>
695
696 <reporting>
697 <plugins>
698 <plugin>
699 <groupId>org.apache.maven.plugins</groupId>
700 <artifactId>maven-project-info-reports-plugin</artifactId>
701 </plugin>
702 </plugins>
703 </reporting>
704
705 <build>
706 <plugins>
707 <plugin>
708 <groupId>org.springframework.boot</groupId>
709 <artifactId>spring-boot-maven-plugin</artifactId>
710 </plugin>
711 </plugins>
712 </build>
713
714 <test>
715 <resources>
716 <resource>
717 <directory>src/test/resources</directory>
718 </resource>
719 </resources>
720 </test>
721
722 <reporting>
723 <plugins>
724 <plugin>
725 <groupId>org.apache.maven.plugins</groupId>
726 <artifactId>maven-project-info-reports-plugin</artifactId>
727 </plugin>
728 </plugins>
729 </reporting>
730
731 <profiles>
732 <profile>
733 <id>dev</id>
734 <activation>
735 <activeByDefault>true</activeByDefault>
736 </activation>
737 <properties>
738 <spring.profiles.active>dev</spring.profiles.active>
739 </properties>
740 </profile>
741 </profiles>
742
743 <repositories>
744 <repository>
745 <id>central</id>
746 <url>https://repo.maven.apache.org/maven2</url>
747 </repository>
748 </repositories>
749
750 <pluginRepositories>
751 <pluginRepository>
752 <id>central</id>
753 <url>https://repo.maven.apache.org/maven2</url>
754 </pluginRepository>
755 </pluginRepositories>
756
757 <reporting>
758 <plugins>
759 <plugin>
760 <groupId>org.apache.maven.plugins</groupId>
761 <artifactId>maven-project-info-reports-plugin</artifactId>
762 </plugin>
763 </plugins>
764 </reporting>
765
766 <build>
767 <plugins>
768 <plugin>
769 <groupId>org.springframework.boot</groupId>
770 <artifactId>spring-boot-maven-plugin</artifactId>
771 </plugin>
772 </plugins>
773 </build>
774
775 <test>
776 <resources>
777 <resource>
778 <directory>src/test/resources</directory>
779 </resource>
780 </resources>
781 </test>
782
783 <reporting>
784 <plugins>
785 <plugin>
786 <groupId>org.apache.maven.plugins</groupId>
787 <artifactId>maven-project-info-reports-plugin</artifactId>
788 </plugin>
789 </plugins>
790 </reporting>
791
792 <profiles>
793 <profile>
794 <id>dev</id>
795 <activation>
796 <activeByDefault>true</activeByDefault>
797 </activation>
798 <properties>
799 <spring.profiles.active>dev</spring.profiles.active>
800 </properties>
801 </profile>
802 </profiles>
803
804 <repositories>
805 <repository>
806 <id>central</id>
807 <url>https://repo.maven.apache.org/maven2</url>
808 </repository>
809 </repositories>
810
811 <pluginRepositories>
812 <pluginRepository>
813 <id>central</id>
814 <url>https://repo.maven.apache.org/maven2</url>
815 </pluginRepository>
816 </pluginRepositories>
817
818 <reporting>
819 <plugins>
820 <plugin>
821 <groupId>org.apache.maven.plugins</groupId>
822 <artifactId>maven-project-info-reports-plugin</artifactId>
823 </plugin>
824 </plugins>
825 </reporting>
826
827 <build>
828 <plugins>
829 <plugin>
830 <groupId>org.springframework.boot</groupId>
831 <artifactId>spring-boot-maven-plugin</artifactId>
832 </plugin>
833 </plugins>
834 </build>
835
836 <test>
837 <resources>
838 <resource>
839 <directory>src/test/resources</directory>
840 </resource>
841 </resources>
842 </test>
843
844 <reporting>
845 <plugins>
846 <plugin>
847 <groupId>org.apache.maven.plugins</groupId>
848 <artifactId>maven-project-info-reports-plugin</artifactId>
849 </plugin>
850 </plugins>
851 </reporting>
852
853 <profiles>
854 <profile>
855 <id>dev</id>
856 <activation>
857 <activeByDefault>true</activeByDefault>
858 </activation>
859 <properties>
860 <spring.profiles.active>dev</spring.profiles.active>
861 </properties>
862 </profile>
863 </profiles>
864
865 <repositories>
866 <repository>
867 <id>central</id>
868 <url>https://repo.maven.apache.org/maven2</url>
869 </repository>
870 </repositories>
871
872 <pluginRepositories>
873 <pluginRepository>
874 <id>central</id>
875 <url>https://repo.maven.apache.org/maven2</url>
876 </pluginRepository>
877 </pluginRepositories>
878
879 <reporting>
880 <plugins>
881 <plugin>
882 <groupId>org.apache.maven.plugins</groupId>
883 <artifactId>maven-project-info-reports-plugin</artifactId>
884 </plugin>
885 </plugins>
886 </reporting>
887
888 <build>
889 <plugins>
890 <plugin>
891 <groupId>org.springframework.boot</groupId>
892 <artifactId>spring-boot-maven-plugin</artifactId>
893 </plugin>
894 </plugins>
895 </build>
896
897 <test>
898 <resources>
899 <resource>
900 <directory>src/test/resources</directory>
901 </resource>
902 </resources>
903 </test>
904
905 <reporting>
906 <plugins>
907 <plugin>
908 <groupId>org.apache.maven.plugins</groupId>
909 <artifactId>maven-project-info-reports-plugin</artifactId>
910 </plugin>
911 </plugins>
912 </reporting>
913
914 <profiles>
915 <profile>
916 <id>dev</id>
917 <activation>
918 <activeByDefault>true</activeByDefault>
919 </activation>
920 <properties>
921 <spring.profiles.active>dev</spring.profiles.active>
922 </properties>
923 </profile>
924 </profiles>
925
926 <repositories>
927 <repository>
928 <id>central</id>
929 <url>https://repo.maven.apache.org/maven2</url>
930 </repository>
931 </repositories>
932
933 <pluginRepositories>
934 <pluginRepository>
935 <id>central</id>
936 <url>https://repo.maven.apache.org/maven2</url>
```

```

29 </dependency>
30
31 <dependency>
32 <groupId>org.mybatis.spring.boot</groupId>
33 <artifactId>mybatis-spring-boot-starter</artifactId>
34 <version>2.1.1</version>
35 </dependency>
36
37 <dependency>
38 <groupId>org.springframework.boot</groupId>
39 <artifactId>spring-boot-devtools</artifactId>
40 <scope>runtime</scope>
41 <optional>true</optional>
42 </dependency>
43
44 <dependency>
45 <groupId>mysql</groupId>
46 <artifactId>mysql-connector-java</artifactId>
47 <scope>runtime</scope>
48 </dependency>
49
50 <dependency>
51 <groupId>org.springframework.boot</groupId>
52 <artifactId>spring-boot-configuration-processor</artifactId>
53 <optional>true</optional>
54 </dependency>
55
56 <dependency>
57 <groupId>org.projectlombok</groupId>
58 <artifactId>lombok</artifactId>
59 <optional>true</optional>
60 </dependency>
61
62 <dependency>
63 <groupId>org.springframework.boot</groupId>
64 <artifactId>spring-boot-starter-test</artifactId>
65 <scope>test</scope>
66 <exclusions>
67 <exclusion>
68 <groupId>org.junit.vintage</groupId>
69 <artifactId>junit-vintage-engine</artifactId>
70 </exclusion>
71 </exclusions>
72 </dependency>
73
74 <!-- https://mvnrepository.com/artifact/com.alibaba/druid-spring-
boot-starter -->
75 <dependency>
76 <groupId>com.alibaba</groupId>
77 <artifactId>druid-spring-boot-starter</artifactId>
78 <version>1.1.21</version>
79 </dependency>
80
81 <!--
https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper-
spring-boot-starter -->
82 <dependency>
83 <groupId>com.github.pagehelper</groupId>

```



```

84 <artifactId>pagehelper-spring-boot-starter</artifactId>
85 <version>1.2.13</version>
86 </dependency>
87
88 <dependency>
89 <groupId>junit</groupId>
90 <artifactId>junit</artifactId>
91 <scope>test</scope>
92 </dependency>
93
94 </dependencies>
95
96 <build>
97 <plugins>
98 <plugin>
99 <groupId>org.springframework.boot</groupId>
100 <artifactId>spring-boot-maven-plugin</artifactId>
101 </plugin>
102 </plugins>
103 </build>
104 </project>

```

### 7.1.3 编写application.yml文件

```

1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-mybatis-demo
6 thymeleaf:
7 cache: false
8 enabled: true
9 encoding: UTF-8
10 datasource:
11 driver-class-name: com.mysql.cj.jdbc.Driver
12 url: jdbc:mysql:///springboot-mybatis?
13 useUnicode=true&characterEncoding=utf8&serverTimezone=UTC
14 username: root
15 password: root
16 type: com.alibaba.druid.pool.DruidDataSource
17 druid:
18 initial-size: 5
19 max-active: 20
20 min-idle: 5
21 max-wait: 60000
22 mybatis:
23 mapper-locations: classpath:mappers/*.xml
24 type-aliases-package: com.bjlemon.springboot.domain
25 configuration:
26 cache-enabled: true
27 lazy-loading-enabled: true
28 aggressive-lazy-loading: false

```

### 7.1.4 编写启动类

```

1 @SpringBootApplication
2 @MapperScan(basePackages = {"com.bjlemon.springboot.mapper"})
3 public class SpringbootMybatisDemoApplication {
4
5 public static void main(String[] args) {
6 SpringApplication.run(SpringbootMybatisDemoApplication.class, args);
7 }
8
9 }

```

## 7.1.5 编写业务逻辑层 ( DepartmentService )

```

1 package com.bjlemon.springboot.service.impl;
2
3 import com.bjlemon.springboot.domain.Department;
4 import com.bjlemon.springboot.domain.DepartmentExample;
5 import com.bjlemon.springboot.domain.User;
6 import com.bjlemon.springboot.mapper.DepartmentMapper;
7 import com.bjlemon.springboot.mapper.UserMapper;
8 import com.bjlemon.springboot.service.DepartmentService;
9 import com.bjlemon.springboot.vo.DepartmentQueryVO;
10 import com.github.pagehelper.PageHelper;
11 import com.github.pagehelper.PageInfo;
12 import org.apache.commons.collections.CollectionUtils;
13 import org.apache.commons.lang3.StringUtils;
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.stereotype.Service;
16 import org.springframework.transaction.annotation.Transactional;
17
18 import java.util.List;
19
20 /**
21 * @author jeffzhou
22 * @version 1.0.0
23 * @ClassName DepartmentServiceImpl.java
24 * @Description TODO
25 * @createTime 2020年01月12日 20:08:00
26 */
27 @Service
28 @Transactional
29 public class DepartmentServiceImpl implements DepartmentService {
30
31 @Autowired
32 private DepartmentMapper departmentMapper;
33
34 @Autowired
35 private UserMapper userMapper;
36
37 @Override
38 public void addDepartment(Department department) {
39 if (department == null) {
40 throw new IllegalArgumentException("");
41 }
42
43 this.departmentMapper.insertSelective(department);
44 }

```

```
45
46 @Override
47 public void deleteDepartment(Department department) {
48 if (department == null) {
49 throw new IllegalArgumentException("");
50 }
51
52 Integer departmentId = department.getDepartmentId();
53 List<User> userList =
this.userMapper.findUsersByDepartmentId(departmentId);
54 if (CollectionUtils.isEmpty(userList)) {
55 throw new RuntimeException("部门下有员工，不能删除!");
56 }
57
58 this.departmentMapper.deleteByPrimaryKey(departmentId);
59 }
60
61 @Override
62 public void modifyDepartment(Department department) {
63 if (department == null) {
64 throw new IllegalArgumentException("");
65 }
66
67 this.departmentMapper.updateByPrimaryKeySelective(department);
68 }
69
70 @Override
71 public Department findDepartmentById(Integer id) {
72 return this.departmentMapper.selectByPrimaryKey(id);
73 }
74
75 @Override
76 public List<Department> findAllDepartmentList() {
77 DepartmentExample departmentExample = new DepartmentExample();
78 return this.departmentMapper.selectByExample(departmentExample);
79 }
80
81 @Override
82 public List<Department> findDepartmentListByQueryVO(DepartmentQueryVO
departmentQueryVO) {
83 DepartmentExample departmentExample = new DepartmentExample();
84
85 if (departmentQueryVO == null) {
86 return this.findAllDepartmentList();
87 }
88
89 String departmentName = departmentQueryVO.getDepartmentName();
90 String departmentLocation =
departmentQueryVO.getDepartmentLocation();
91
92 DepartmentExample.Criteria criteria =
departmentExample.createCriteria();
93
94 if (StringUtils.isNotBlank(departmentName)) {
95 criteria.andDepartmentNameLike("%" + departmentName + "%");
96 }
97
98 if (StringUtils.isNotBlank(departmentLocation)) {
```

```

99 criteria.andDepartmentLocationLike("'" + departmentLocation +
100 "%");
101 }
102 return this.departmentMapper.selectByExample(departmentExample);
103 }
104
105 @Override
106 public PageInfo<Department> findDepartmentPaginationList(Integer
107 pageNum, Integer pageSize) {
108 if (pageNum == null || pageNum <= 0) {
109 throw new IllegalArgumentException("");
110 }
111 if (pageSize == null || pageSize <= 0) {
112 throw new IllegalArgumentException("");
113 }
114
115 PageHelper.startPage(pageNum, pageSize);
116 DepartmentExample departmentExample = new DepartmentExample();
117 List<Department> departmentList =
118 this.departmentMapper.selectByExample(departmentExample);
119
120 return new PageInfo<>(departmentList);
121 }

```

## 7.2 SpringBoot与MP ( MyBatis-Plus ) 集成

### 7.2.1 MP概述

- MP是MyBatis-plus的缩写。实际上就是MyBatis的插件，或者是MyBatis的加强版
- MP是baomidou团队开发
- 官网：<https://github.com/baomidou/mybatis-plus>
- 简介

1 MyBatis-Plus is an powerful enhanced toolkit of MyBatis for simplify development. This toolkit provides some efficient, useful, out-of-the-box features for MyBatis, use it can effectively save your development time.

- 特点

```

1 Fully compatible with MyBatis
2 Auto configuration on startup
3 Out-of-the-box interfaces for operate database
4 Powerful and flexible where condition wrapper
5 Multiple strategy to generate primary key
6 Lambda-style API
7 Almighty and highly customizable code generator
8 Automatic paging operation
9 SQL Inject defense
10 Support active record
11 Support pluggable custom interface
12 Build-in many useful extensions

```

## 7.2.2 SpringBoot与MP集成

```
1 <dependency>
2 <groupId>com.baomidou</groupId>
3 <artifactId>mybatis-plus-boot-starter</artifactId>
4 <version>3.2.0</version>
5 </dependency>
6
7 <dependency>
8 <groupId>mysql</groupId>
9 <artifactId>mysql-connector-java</artifactId>
10 <scope>runtime</scope>
11 </dependency>
12
13 <!-- https://mvnrepository.com/artifact/com.alibaba/druid-spring-boot-
14 starter -->
15 <dependency>
16 <groupId>com.alibaba</groupId>
17 <artifactId>druid-spring-boot-starter</artifactId>
18 <version>1.1.21</version>
19 </dependency>
```

```
1 server:
2 port: 8888
3 spring:
4 application:
5 name: springboot-mybatis-plus-demo
6 thymeleaf:
7 cache: false
8 enabled: true
9 encoding: UTF-8
10 datasource:
11 driver-class-name: com.mysql.cj.jdbc.Driver
12 url: jdbc:mysql:///springboot-mybatis?
13 useUnicode=true&characterEncoding=utf8&serverTimezone=UTC
14 username: root
15 password: root
16 type: com.alibaba.druid.pool.DruidDataSource
17 druid:
18 initial-size: 5
19 max-active: 20
20 min-idle: 5
21 max-wait: 60000
22 mybatis-plus:
23 configuration:
24 ### domain-----userId
25 ### table-----user_id
26 map-underscore-to-camel-case: true
27 use-generated-keys: true
28 global-config:
29 db-config:
30 table-prefix: springboot_mybatis_
```

```
1 @Data
2 @NoArgsConstructor
```

```

3 @AllArgsConstructor
4 //@TableName(value = "springboot_mybatis_department")
5 public class Department implements Serializable {
6
7 private static final long serialVersionUID = 626719896969191791L;
8 @TableId(value = "department_id", type = IdType.AUTO)
9 private Integer id;
10
11 @TableField(value = "department_name")
12 private String name;
13
14 @TableField(value = "department_location")
15 private String location;
16
17 @Override
18 public boolean equals(Object o) {
19 if (this == o) return true;
20 if (o == null || getClass() != o.getClass()) return false;
21
22 Department that = (Department) o;
23
24 if (id != null ? !id.equals(that.id) : that.id != null) return
false;
25 if (name != null ? !name.equals(that.name) : that.name != null)
return false;
26 return location != null ? location.equals(that.location) :
that.location == null;
27 }
28
29 @Override
30 public int hashCode() {
31 int result = id != null ? id.hashCode() : 0;
32 result = 31 * result + (name != null ? name.hashCode() : 0);
33 result = 31 * result + (location != null ? location.hashCode() :
0);
34 return result;
35 }
36
37 @Override
38 public String toString() {
39 return "Department{" +
40 "id=" + id +
41 ", name='" + name + '\'' +
42 ", location='" + location + '\'' +
43 '}';
44 }
45 }
46

```

```

1 @Data
2 @NoArgsConstructor
3 @AllArgsConstructor
4 @TableName(value = "springboot_mybatis_user")
5 public class User implements Serializable {
6
7 private static final long serialVersionUID = -6048472400496098620L;
8 @TableId(value = "user_id", type = IdType.AUTO)

```

```

9 private Integer id;
10
11 @TableField(value = "user_name")
12 private String name;
13
14 @TableField(value = "user_password")
15 private String password;
16
17 @TableField(value = "user_salary")
18 private Float salary;
19
20 @TableField(value = "user_birthday")
21 private Date birthday;
22
23 private Department department;
24
25 @Override
26 public boolean equals(Object o) {
27 if (this == o) return true;
28 if (o == null || getClass() != o.getClass()) return false;
29
30 User user = (User) o;
31
32 if (id != null ? !id.equals(user.id) : user.id != null) return
false;
33 if (name != null ? !name.equals(user.name) : user.name != null)
return false;
34 if (password != null ? !password.equals(user.password) :
user.password != null) return false;
35 if (salary != null ? !salary.equals(user.salary) : user.salary !=
null) return false;
36 return birthday != null ? birthday.equals(user.birthday) :
user.birthday == null;
37 }
38
39 @Override
40 public int hashCode() {
41 int result = id != null ? id.hashCode() : 0;
42 result = 31 * result + (name != null ? name.hashCode() : 0);
43 result = 31 * result + (password != null ? password.hashCode() :
0);
44 result = 31 * result + (salary != null ? salary.hashCode() : 0);
45 result = 31 * result + (birthday != null ? birthday.hashCode() :
0);
46 return result;
47 }
48
49 @Override
50 public String toString() {
51 return "User{" +
52 "id=" + id +
53 ", name='" + name + '\'' +
54 ", password='" + password + '\'' +
55 ", salary=" + salary +
56 ", birthday=" + birthday +
57 '}';
58 }
59 }

```

```

1 package com.bjlemon.springboot.mapper;
2
3 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
4 import com.bjlemon.springboot.domain.User;
5 import org.apache.ibatis.annotations.*;
6
7 import java.util.List;
8
9 /**
10 * @author jeffzhou
11 * @version 1.0.0
12 * @ClassName DepartmentMapper.java
13 * @Description TODO
14 * @createTime 2020年01月12日 21:43:00
15 */
16 public interface UserMapper extends BaseMapper<User> {
17
18 /**@Select(value = "SELECT user_id id, user_name name, user_password
19 password, user_salary salary, user_birthday birthday " +
20 "FROM.springboot_mybatis_user " +
21 "WHERE department_id = #{id}")
22 List<User> findUsersByDepartmentId(@Param(value = "id") Integer
23 departmentId);*/
24
25 /**@Select(value = "SELECT user_id, user_name, user_password,
26 user_salary, user_birthday " +
27 "FROM.springboot_mybatis_user " +
28 "WHERE department_id = #{id}")
29 @Results(id = "UserBaseResultMap", value = {
30 @Result(property = "id", column = "user_id", id = true),
31 @Result(property = "name", column = "user_name"),
32 @Result(property = "password", column = "user_password"),
33 @Result(property = "salary", column = "user_salary"),
34 @Result(property = "birthday", column = "user_birthday")
35 })
36 List<User> findUsersByDepartmentId(@Param(value = "id") Integer
37 departmentId);*/
38
39 @Select(value = "SELECT " +
40 "smu.user_id,\n" +
41 " smu.user_name,\n" +
42 " smu.user_password,\n" +
43 " smu.user_salary,\n" +
44 " smu.user_birthday,\n" +
45 " smd.department_id,\n" +
46 " smd.department_name,\n" +
47 " smd.department_location\n" +
48 "FROM.springboot_mybatis_user smu\n" +
49 " " +
50 "LEFT JOIN.springboot_mybatis_department smd ON
51 smu.department_id = smd.department_id\n" +
52 "WHERE smd.department_id = #{id}")
53 @Results(id = "UserResultMap", value = {
54 @Result(property = "id", column = "user_id", id = true),
55 @Result(property = "name", column = "user_name"),

```



```

51 @Result(property = "password", column = "user_password"),
52 @Result(property = "salary", column = "user_salary"),
53 @Result(property = "birthday", column = "user_birthday"),
54 @Result(property = "department", column = "department_id", one =
@One(select = "com.bjlemon.springboot.mapper.DepartmentMapper.findById"))
55 })
56 List<User> findUsersByDepartmentId(@Param(value = "id") Integer
departmentId);
57
58 }
59

```

```

1 public interface DepartmentMapper extends BaseMapper<Department> {
2
3 @Select("SELECT * FROM.springboot_mybatis_department WHERE department_id
= #{id}")
4 @Results(id = "DepartmentResultMap", value = {
5 @Result(id = true, property = "id", column = "department_id"),
6 @Result(property = "name", column = "department_name"),
7 @Result(property = "location", column = "department_location")
8 })
9 Department findById(Integer id);
10 }

```

```

1 @SpringBootApplication
2 @MapperScan(basePackages = {"com.bjlemon.springboot.mapper"})
3 public class SpringbootMybatisPlusDmoApplication {
4
5 public static void main(String[] args) {
6 SpringApplication.run(SpringbootMybatisPlusDmoApplication.class,
args);
7 }
8
9 }

```

## 7.3 SpringBoot与JPA集成

### 7.3.1 JPA概述

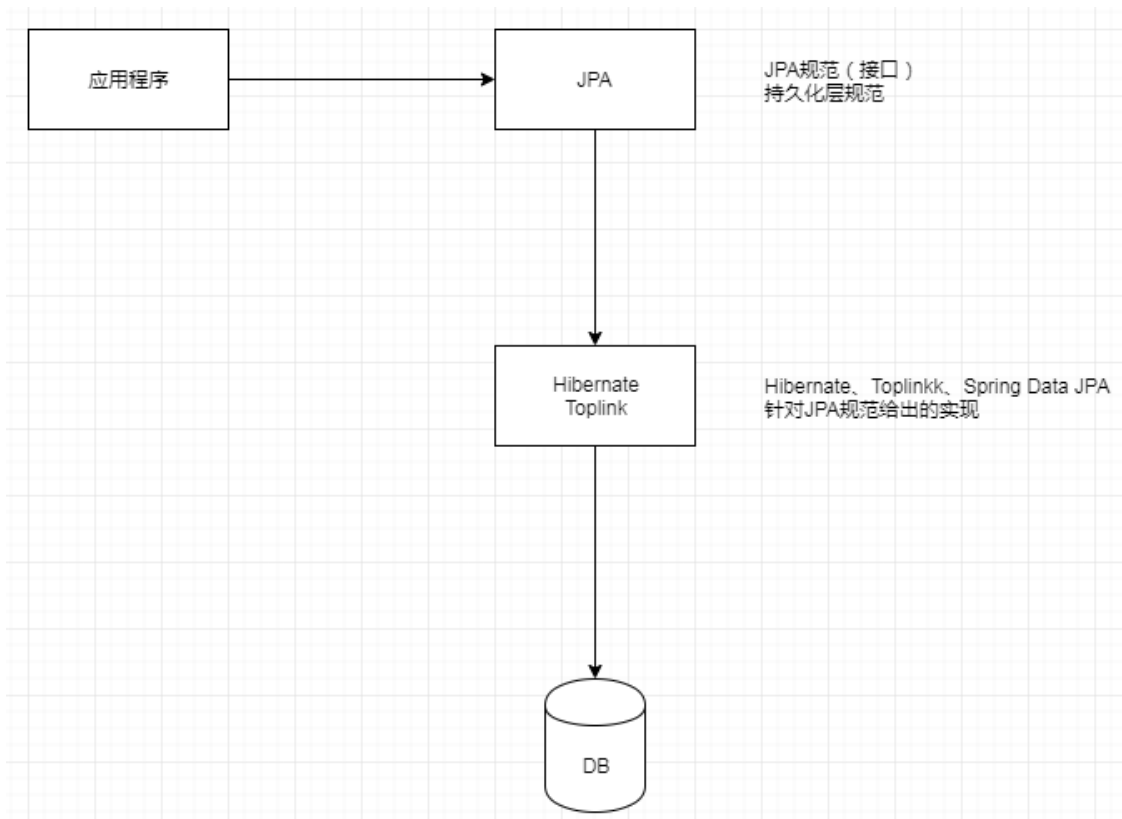
- Java Persistence API。Java持久化API
- JPA是SUN公司提出的ORM ( Object Relationship Mapping ) 规范
- JPA通过注解描述对象与关系的映射。以前Hibernate是通过XML文件来进行映射

### 7.3.2 JPA优势

- 标准：由JCP组织制定
- 各种容器支持度很高
- 使用成本较低：当时JPA之前有一个EJB ( EntityBean 成本非常高 )
- 查询语言 ( JPQL 纯面向对象的查询语言 )

### 7.3.3 JPA实现

- Hibernate
- Toplink ( Oracle )
- Spring Data JPA



### 7.3.4 JPA开发

```

1 <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-
2 entitymanager -->
3 <dependency>
4 <groupId>org.hibernate</groupId>
5 <artifactId>hibernate-entitymanager</artifactId>
6 <version>5.4.10.Final</version>
7 </dependency>
8 <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-c3p0 -->
9 <dependency>
10 <groupId>org.hibernate</groupId>
11 <artifactId>hibernate-c3p0</artifactId>
12 <version>5.4.10.Final</version>
13 </dependency>
14
15 <dependency>
16 <groupId>mysql</groupId>
17 <artifactId>mysql-connector-java</artifactId>
18 <version>5.1.48</version>
19 <scope>runtime</scope>
20 </dependency>

```

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/persistence"

```

```

3 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
4 version="2.0">
5
6 <persistence-unit name="jpa-demo" transaction-type="RESOURCE_LOCAL">
7 <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
8
9 <properties>
10 <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
11 <property name="hibernate.connection.url"
value="jdbc:mysql://127.0.0.1:3306/jpa_demo"/>
12 <property name="hibernate.connection.username" value="root"/>
13 <property name="hibernate.connection.password" value="root"/>
14 <!--说明:数据库的方言，用于存放不同数据库之间的SQL语句差异。-->
15 <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect"/>
16
17 <!--
18 create:每次启动服务器重新建表
19 update:更新表结构
20 -->
21 <property name="hibernate.hbm2ddl.auto" value="update"/>
22 <!--每次执行操作会发SQL语句-->
23 <property name="hibernate.show_sql" value="true"/>
24 <!--格式化SQL语句-->
25 <property name="hibernate.format_sql" value="true"/>
26 </properties>
27 </persistence-unit>
28 </persistence>

```

```

1 package com.bjlemon.jpa.domain;
2
3 import Lombok.Data;
4
5 import javax.persistence.*;
6 import java.io.Serializable;
7 import java.util.Date;
8
9 /**
10 * @author jeffzhou
11 * @version 1.0.0
12 * @ClassName User.java
13 * @Description TODO
14 * @createTime 2020年01月14日 20:59:00
15 */
16 @Data
17 @Entity
18 @Table(name = "jpa_user")
19 public class User implements Serializable {
20
21 private static final long serialVersionUID = 1888819090828579238L;
22
23 @Id
24 // 主键生成策略。其中strategy注解的属性，该属性如果不指定值那么就会使用默认值
25 // (AUTO)
26 // 根据底层数据库自动选择主键生成策略

```

```

26 // 当前数据库是mysql数据库，又由于id是整型，因此建表时会将其字段做成自动增长
27 @GeneratedValue(strategy = GenerationType.IDENTITY)
28 @Column(name = "user_id", length = 4)
29 private Integer id;
30
31 @Column(name = "user_name", length = 20, nullable = false)
32 private String name;
33
34 @Column(name = "user_password", length = 20, nullable = false)
35 private String password;
36
37 @Column(name = "user_salary", length = 6, precision = 2, nullable =
false)
38 private Float salary;
39
40 @Column(name = "user_birthday", nullable = false)
41 @Temporal(TemporalType.DATE)
42 private Date birthday;
43 }
44

```

```

1 package com.bjlemon.jpa.test;
2
3 import com.bjlemon.jpa.domain.User;
4 import com.bjlemon.jpa.util.JpaUtils;
5 import org.junit.Test;
6
7 import javax.persistence.EntityManager;
8 import javax.persistence.EntityTransaction;
9 import java.util.Collections;
10 import java.util.Date;
11 import java.util.List;
12
13 /**
14 * @author jeffzhou
15 * @version 1.0.0
16 * @ClassName UserTest.java
17 * @Description TODO
18 * @createTime 2020年01月14日 21:25:00
19 */
20 public class UserTest {
21
22 @Test
23 public void testAdd() {
24 EntityManager entityManager = null;
25 EntityTransaction transaction = null;
26 User user = null;
27 try {
28 entityManager = JpaUtils.getEntityManager();
29 transaction = entityManager.getTransaction();
30 transaction.begin();
31
32 user = new User();
33 user.setName("zhangsan");
34 user.setPassword("admin");
35 user.setSalary(12.34F);
36 user.setBirthday(new Date());
37
38 } catch (Exception e) {
39 e.printStackTrace();
40 } finally {
41 if (transaction != null) {
42 transaction.rollback();
43 }
44 if (entityManager != null) {
45 entityManager.close();
46 }
47 }
48 }
49 }

```

```

37 entityManager.persist(user);
38
39
40 transaction.commit();
41 } catch (Exception e) {
42 e.printStackTrace();
43 transaction.rollback();
44 } finally {
45 JpaUtils.closeEntityManager(entityManager);
46 }
47 }
48
49 @Test
50 public void testDelete() {
51 EntityManager entityManager = null;
52 EntityTransaction transaction = null;
53 User user = null;
54 try {
55 entityManager = JpaUtils.getEntityManager();
56 transaction = entityManager.getTransaction();
57 transaction.begin();
58
59 // 查询的是一个代理对象 (User), 使用到了延迟加载技术
60 // 问题: 不要过早关闭EntityManager, 如果关了那么代理对象不能初始化, 导致出问题
61 // 怎么解决问题: web项目配置一个过滤器
62 // (OpenEntityManagerInViewFilter)
63 user = entityManager.getReference(User.class, 3);
64 // Hibernate.initialize(user);
65
66 // 真正查询User类型的对象
67 // user = entityManager.find(User.class, 1);
68 entityManager.remove(user);
69
70 transaction.commit();
71 } catch (Exception e) {
72 e.printStackTrace();
73 transaction.rollback();
74 } finally {
75 JpaUtils.closeEntityManager(entityManager);
76 }
77 }
78
79 @Test
80 public void testUpdate() {
81 EntityManager entityManager = null;
82 EntityTransaction transaction = null;
83 User user = null;
84 try {
85 entityManager = JpaUtils.getEntityManager();
86 transaction = entityManager.getTransaction();
87 transaction.begin();
88
89 // 真正查询User类型的对象
90 user = entityManager.find(User.class, 1);
91 user.setName("C罗");
92 user.setSalary(99.00F);

```

```

93 entityManager.merge(user);
94
95
96 transaction.commit();
97 } catch (Exception e) {
98 e.printStackTrace();
99 transaction.rollback();
100 } finally {
101 JpaUtils.closeEntityManager(entityManager);
102 }
103 }
104
105 @Test
106 public void testFindAll() {
107 EntityManager entityManager = null;
108 EntityTransaction transaction = null;
109 User user = null;
110 try {
111 entityManager = JpaUtils.getEntityManager();
112 transaction = entityManager.getTransaction();
113 transaction.begin();
114
115
116 List userList = entityManager.createQuery("from User
u").getResultList();
117 userList.stream().forEach(System.out::println);
118
119 transaction.commit();
120 } catch (Exception e) {
121 e.printStackTrace();
122 transaction.rollback();
123 } finally {
124 JpaUtils.closeEntityManager(entityManager);
125 }
126 }
127
128 @Test
129 public void testFind() {
130 EntityManager entityManager = null;
131 EntityTransaction transaction = null;
132 User user = null;
133 try {
134 entityManager = JpaUtils.getEntityManager();
135 transaction = entityManager.getTransaction();
136 transaction.begin();
137
138 user = (User) entityManager.createQuery("from User u where
u.name = ?1 and u.password = ?2")
139 .setParameter(1, "zhangsan")
140 .setParameter(2, "admin")
141 .getSingleResult();
142 System.out.println(user);
143
144 transaction.commit();
145 } catch (Exception e) {
146 e.printStackTrace();
147 transaction.rollback();
148 } finally {

```

```

149 JpaUtils.closeEntityManager(entityManager);
150 }
151 }
152
153 @Test
154 public void testFind1() {
155 EntityManager entityManager = null;
156 EntityTransaction transaction = null;
157 User user = null;
158 try {
159 entityManager = JpaUtils.getEntityManager();
160 transaction = entityManager.getTransaction();
161 transaction.begin();
162
163 user = (User) entityManager.createQuery("from User u where
164 u.name = :uname and u.password = :pwd")
165 .setParameter("uname", "zhangsan")
166 .setParameter("pwd", "admin")
167 .getSingleResult();
168 System.out.println(user);
169
170 transaction.commit();
171 } catch (Exception e) {
172 e.printStackTrace();
173 transaction.rollback();
174 } finally {
175 JpaUtils.closeEntityManager(entityManager);
176 }
177 }
178
179 @Test
180 public void testFind2() {
181 EntityManager entityManager = null;
182 EntityTransaction transaction = null;
183 List<User> userList = Collections.EMPTY_LIST;
184 try {
185 entityManager = JpaUtils.getEntityManager();
186 transaction = entityManager.getTransaction();
187 transaction.begin();
188
189 Integer pageNum = 2;
190 Integer pageSize = 3;
191 userList = entityManager.createQuery("from User u")
192 .setFirstResult((pageNum - 1) * pageSize)
193 .setMaxResults(3)
194 .getResultList();
195
196 userList.stream().forEach(System.out::println);
197
198 transaction.commit();
199 } catch (Exception e) {
200 e.printStackTrace();
201 transaction.rollback();
202 } finally {
203 JpaUtils.closeEntityManager(entityManager);
204 }
205 }

```

## 7.3.5 Spring Data JPA

- Spring Data概述

- 1 Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.
- 2
- 3 It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

- Spring Data特点

- 1 Powerful repository and custom object-mapping abstractions
- 2
- 3 Dynamic query derivation from repository method names
- 4
- 5 Implementation domain base classes providing basic properties
- 6
- 7 Support for transparent auditing (created, last changed)
- 8
- 9 Possibility to integrate custom repository code
- 10
- 11 Easy Spring integration via JavaConfig and custom XML namespaces
- 12
- 13 Advanced integration with Spring MVC controllers
- 14
- 15 Experimental support for cross-store persistence

- Spring Data JPA概述

- 1 Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.
- 2
- 3 Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

- Spring Data JPA特点

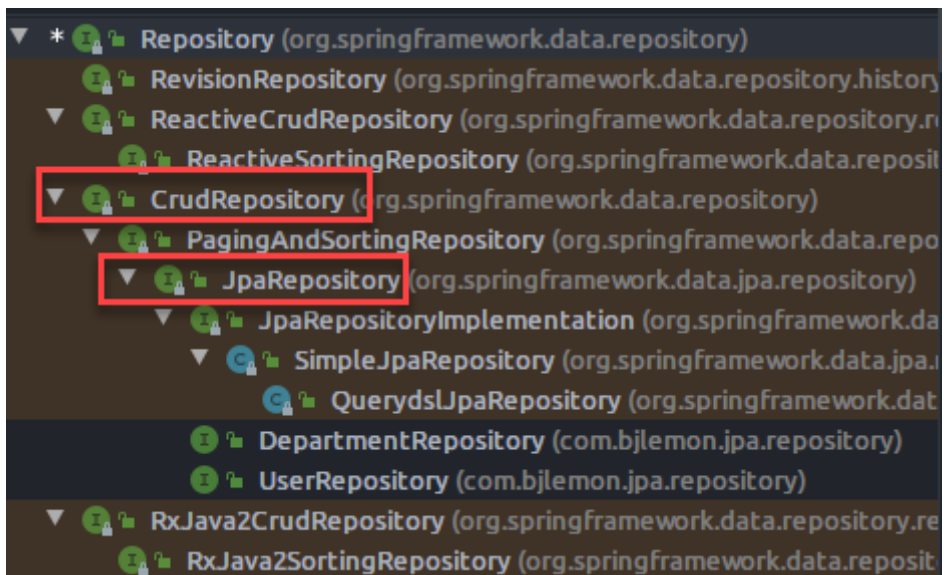


```

1 Sophisticated support to build repositories based on Spring and JPA
2
3 Support for Querydsl predicates and thus type-safe JPA queries
4
5 Transparent auditing of domain class
6
7 Pagination support, dynamic query execution, ability to integrate custom
 data access code
8
9 Validation of @Query annotated queries at bootstrap time
10
11 Support for XML based entity mapping
12
13 JavaConfig based repository configuration by introducing
 @EnableJpaRepositories.

```

- Spring Data JPA开发
  - Repository接口



- 开发中一般使用JpaRepository接口
- 开发步骤

```

1 <dependencies>
2 <dependency>
3 <groupId>org.springframework</groupId>
4 <artifactId>spring-context</artifactId>
5 <version>5.2.2.RELEASE</version>
6 </dependency>
7
8 <dependency>
9 <groupId>org.springframework</groupId>
10 <artifactId>spring-context-support</artifactId>
11 <version>5.2.2.RELEASE</version>
12 </dependency>
13
14 <dependency>
15 <groupId>org.springframework</groupId>
16 <artifactId>spring-tx</artifactId>
17 <version>5.2.2.RELEASE</version>
18 </dependency>

```

```

19
20 <dependency>
21 <groupId>org.springframework</groupId>
22 <artifactId>spring-test</artifactId>
23 <version>5.2.2.RELEASE</version>
24 </dependency>
25
26 <dependency>
27 <groupId>junit</groupId>
28 <artifactId>junit</artifactId>
29 <version>4.12</version>
30 <scope>test</scope>
31 </dependency>
32
33 <dependency>
34 <groupId>mysql</groupId>
35 <artifactId>mysql-connector-java</artifactId>
36 <version>5.1.48</version>
37 <scope>runtime</scope>
38 </dependency>
39
40 <!-- https://mvnrepository.com/artifact/com.alibaba/druid -->
41 <dependency>
42 <groupId>com.alibaba</groupId>
43 <artifactId>druid</artifactId>
44 <version>1.1.21</version>
45 </dependency>
46
47 <dependency>
48 <groupId>org.springframework.data</groupId>
49 <artifactId>spring-data-jpa</artifactId>
50 <version>2.2.3.RELEASE</version>
51 </dependency>
52
53 <!--
54 https://mvnrepository.com/artifact/org.hibernate/hibernate-
55 entitymanager -->
56 <dependency>
57 <groupId>org.hibernate</groupId>
58 <artifactId>hibernate-entitymanager</artifactId>
59 <version>5.4.10.Final</version>
60 </dependency>
61
62 <dependency>
63 <groupId>org.projectlombok</groupId>
64 <artifactId>lombok</artifactId>
65 <version>1.18.10</version>
66 <scope>provided</scope>
67 </dependency>
68 </dependencies>

```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns:context="http://www.springframework.org/schema/context"
4 xmlns:tx="http://www.springframework.org/schema/tx"

```

```

4 xmlns:jpa="http://www.springframework.org/schema/data/jpa"
 xmlns="http://www.springframework.org/schema/beans"
5
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd
 http://www.springframework.org/schema/context
 https://www.springframework.org/schema/context/spring-context.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx.xsd
 http://www.springframework.org/schema/data/jpa
 https://www.springframework.org/schema/data/jpa/spring-jpa.xsd">
6
7 <context:component-scan base-package="com.bjlemon.jpa"/>
8
9 <context:property-placeholder
 location="classpath*:druidconfig.properties"/>
10
11
12 <bean id="dataSource"
 class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
13 destroy-method="close">
14 <property name="driverClassName"
 value="${jdbc.driverClassName}"/>
15 <property name="url" value="${jdbc.url}"/>
16 <property name="username" value="${jdbc.username}"/>
17 <property name="password" value="${jdbc.password}"/>
18 <property name="maxActive" value="${jdbc.maxActive}"/>
19 <!-- <property name="maxIdle"
 value="${jdbc.maxIdle}"/>-->
20 <property name="initialSize" value="${jdbc.initialSize}"/>
21 <property name="minIdle" value="${jdbc.minIdle}"/>
22 <property name="maxWait" value="${jdbc.maxWait}"/>
23 </bean>
24
25 <bean id="entityManagerFactory"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFacto
 ryBean">
26 <property name="dataSource" ref="dataSource"/>
27 <property name="jpaVendorAdapter">
28 <bean
 class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter
 ">
29 <property name="showSql" value="true"/>
30 <property name="generateDdl" value="true"/>
31 </bean>
32 </property>
33
34 <property name="jpaDialect">
35 <bean
 class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"/>
36 </property>
37
38 <property name="packagesToScan"
 value="com.bjlemon.jpa.domain"/>
39 </bean>
40
41 <bean id="transactionManager"
 class="org.springframework.orm.jpa.JpaTransactionManager">

```

```

42 <property name="entityManagerFactory"
ref="entityManagerFactory"/>
43 </bean>
44
45 <tx:annotation-driven transaction-
manager="transactionManager"/>
46
47 <jpa:repositories base-package="com.bjlemon.jpa.repository"
48 transaction-manager-ref="transactionManager"
49 entity-manager-factory-
ref="entityManagerFactory"/>
50
51 </beans>

```

```

1 package com.bjlemon.jpa.domain;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import javax.persistence.*;
8 import java.io.Serializable;
9 import java.util.HashSet;
10 import java.util.Set;
11
12 /**
13 * @author jeffzhou
14 * @version 1.0.0
15 * @ClassName Department.java
16 * @Description TODO
17 * @createTime 2020年01月14日 22:40:00
18 */
19 @Data
20 @NoArgsConstructor
21 @AllArgsConstructor
22 @Entity
23 @Table(name = "jpa_department")
24 public class Department implements Serializable {
25
26 private static final long serialVersionUID =
-428052927254415765L;
27
28 @Id
29 @GeneratedValue(strategy = GenerationType.IDENTITY)
30 @Column(name = "department_id", length = 4)
31 private Integer id;
32
33 @Column(name = "department_name", length = 20, nullable =
false)
34 private String name;
35
36 @Column(name = "department_location", length = 50, nullable =
false)
37 private String location;
38
39 @OneToMany(mappedBy = "department")
40 private Set<User> users = new HashSet<>();
41 }

```

```

1 package com.bjlemon.jpa.domain;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import javax.persistence.*;
8 import java.io.Serializable;
9 import java.util.Date;
10
11 /**
12 * @author jeffzhou
13 * @version 1.0.0
14 * @ClassName User.java
15 * @Description TODO
16 * @createTime 2020年01月14日 22:41:00
17 */
18 @Data
19 @NoArgsConstructor
20 @AllArgsConstructor
21 @Entity
22 @Table(name = "jpa_user")
23 public class User implements Serializable {
24
25 private static final long serialVersionUID =
26 1888819090828579238L;
27 @Id
28 @GeneratedValue(strategy = GenerationType.IDENTITY)
29 @Column(name = "user_id", length = 4)
30 private Integer id;
31
32 @Column(name = "user_name", length = 20, nullable = false)
33 private String name;
34
35 @Column(name = "user_password", length = 20, nullable = false)
36 private String password;
37
38 @Column(name = "user_salary", length = 6, precision = 2,
39 nullable = false)
40 private Float salary;
41
42 @Column(name = "user_birthday", nullable = false)
43 @Temporal(TemporalType.DATE)
44 private Date birthday;
45
46 @ManyToOne
47 @JoinColumn(name = "department_id", nullable = true)
48 private Department department;
49 }

```

```

1 public interface DepartmentRepository extends
 JpaRepository<Department, Integer> {
2
3 }

```

```

1 public interface UserRepository extends JpaRepository<User,
 Integer> {
2 @Query("select u from User u where u.name like ?1")
3 List<User> findUsersByName(String name);
4
5 @Query("select u from User u where u.name like ?1 and u.salary
 = ?2")
6 List<User> findUsersByNameAndSalary(String name, Float salary);
7
8 @Query(value = "select * from jpa_user where user_name like ?
 and user_salary = ?", nativeQuery = true)
9 List<User> findUsersByNameAndSalary1(String name, Float
 salary);
10 }

```

```

1 @Service
2 public class StudentService implements IStudentService {
3
4 @Autowired
5 private IStudentRepository repository;
6
7 //无关代码略
8
9 @Override
10 public List<Student> getStudent(String studentNumber, String
 name, String nickName,
11 Date birthday, String courseName, float
 chineseScore, float mathScore,
12 float englishScore, float performancePoints) {
13 Specification<Student> specification = new
 Specification<Student>(){
14
15 @Override
16 public Predicate toPredicate(Root<Student> root,
 CriteriaQuery<?> query, CriteriaBuilder cb) {
17 //用于暂时存放查询条件的集合
18 List<Predicate> predicatesList = new ArrayList<>();
19 //-----
20 //查询条件示例
21 //equal示例
22 if (!StringUtils.isEmpty(name)){
23 Predicate namePredicate =
 cb.equal(root.get("name"), name);
24 predicatesList.add(namePredicate);
25 }
26 //like示例
27 if (!StringUtils.isEmpty(nickName)){
28 Predicate nickNamePredicate =
 cb.like(root.get("nickName"), '%' + nickName + '%');
29 predicatesList.add(nickNamePredicate);
30 }

```

```

31 //between示例
32 if (birthday != null) {
33 Predicate birthdayPredicate =
cb.between(root.get("birthday"), birthday, new Date());
34 predicatesList.add(birthdayPredicate);
35 }
36
37 //关联表查询示例
38 if (!StringUtils.isEmpty(courseName)) {
39 Join<Student,Teacher> joinTeacher =
root.join("teachers",JoinType.LEFT);
40 Predicate coursePredicate =
cb.equal(joinTeacher.get("courseName"), courseName);
41 predicatesList.add(coursePredicate);
42 }
43
44 //复杂条件组合示例
45 if (chineseScore!=0 && mathScore!=0 &&
englishScore!=0 && performancePoints!=0) {
46 Join<Student,Examination> joinExam =
root.join("exams",JoinType.LEFT);
47 Predicate predicateExamChinese =
cb.ge(joinExam.get("chineseScore"),chineseScore);
48 Predicate predicateExamMath =
cb.ge(joinExam.get("mathScore"),mathScore);
49 Predicate predicateExamEnglish =
cb.ge(joinExam.get("englishScore"),englishScore);
50 Predicate predicateExamPerformance =
cb.ge(joinExam.get("performancePoints"),performancePoints);
51 //组合
52 Predicate predicateExam =
cb.or(predicateExamChinese,predicateExamEnglish,predicateExamMath);
53 Predicate predicateExamAll =
cb.and(predicateExamPerformance,predicateExam);
54 predicatesList.add(predicateExamAll);
55 }
56 //-----
57 //排序示例(先根据学号排序, 后根据姓名排序)
58
59 query.orderBy(cb.asc(root.get("studentNumber")),cb.asc(root.get("name")));
60 //-----
61 //最终将查询条件拼好然后return
62 Predicate[] predicates = new
Predicate[predicatesList.size()];
63 return cb.and(predicatesList.toArray(predicates));
64 }
65
66 };
67 return repository.findAll(specification);
68 }
69
70 }
71

```

## 7.4 SpringBoot与Spring Data JPA整合

### 7.4.1 编写pom.xml

```
1 <dependency>
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-starter-data-jpa</artifactId>
4 </dependency>
5 <dependency>
6 <groupId>org.springframework.boot</groupId>
7 <artifactId>spring-boot-starter-thymeleaf</artifactId>
8 </dependency>
9 <dependency>
10 <groupId>org.springframework.boot</groupId>
11 <artifactId>spring-boot-starter-web</artifactId>
12 </dependency>
13
14 <dependency>
15 <groupId>org.springframework.boot</groupId>
16 <artifactId>spring-boot-devtools</artifactId>
17 <scope>runtime</scope>
18 <optional>true</optional>
19 </dependency>
20 <dependency>
21 <groupId>mysql</groupId>
22 <artifactId>mysql-connector-java</artifactId>
23 <scope>runtime</scope>
24 </dependency>
25 <dependency>
26 <groupId>org.springframework.boot</groupId>
27 <artifactId>spring-boot-configuration-processor</artifactId>
28 <optional>true</optional>
29 </dependency>
30 <dependency>
31 <groupId>org.projectlombok</groupId>
32 <artifactId>lombok</artifactId>
33 <optional>true</optional>
34 </dependency>
35 <dependency>
36 <groupId>org.springframework.boot</groupId>
37 <artifactId>spring-boot-starter-test</artifactId>
38 <scope>test</scope>
39 <exclusions>
40 <exclusion>
41 <groupId>org.junit.vintage</groupId>
42 <artifactId>junit-vintage-engine</artifactId>
43 </exclusion>
44 </exclusions>
45 </dependency>
```

### 7.4.2 编写application.yml

```
1 server:
2 port: 8888
3 spring:
```



```

4 application:
5 name: springboot-data-jpa-demo
6 thymeleaf:
7 cache: false
8 enabled: true
9 encoding: UTF-8
10 datasource:
11 driver-class-name: com.mysql.cj.jdbc.Driver
12 url: jdbc:mysql:///springboot-data-jpa?
useUnicode=true&characterEncoding=utf8&serverTimezone=UTC
13 username: root
14 password: root
15 type: com.alibaba.druid.pool.DruidDataSource
16 druid:
17 initial-size: 5
18 max-active: 20
19 min-idle: 5
20 max-wait: 60000
21 jpa:
22 show-sql: true
23 generate-ddl: true
24 open-in-view: true

```

### 7.4.3 编写领域对象

...

## 八.SpringBoot与权限管理系统整合（Spring Security）

### 8.1 什么是权限管理系统

#### 8.1.1 认证

- 登录验证。用户登录到系统，必须首先登录。系统会首先根据用户的身份（用户名称）查询数据库，如果查询到该用户再拿数据库中的密码与输入的凭证（密码）进行匹配，如果匹配成功则认为认证成功
- 用户有身份（用户名称）和凭证（密码）

#### 8.1.2 授权

- 用户的权限的访问控制。该用户认证成功后，然后系统会对这个用户进行访问控制。判断该用户对某个资源（菜单或链接）能不能访问。
- 判断用户对这个资源有没有访问权限的问题
- 基于角色的访问控制（粗粒度）
- 基于资源的访问控制（细粒度）

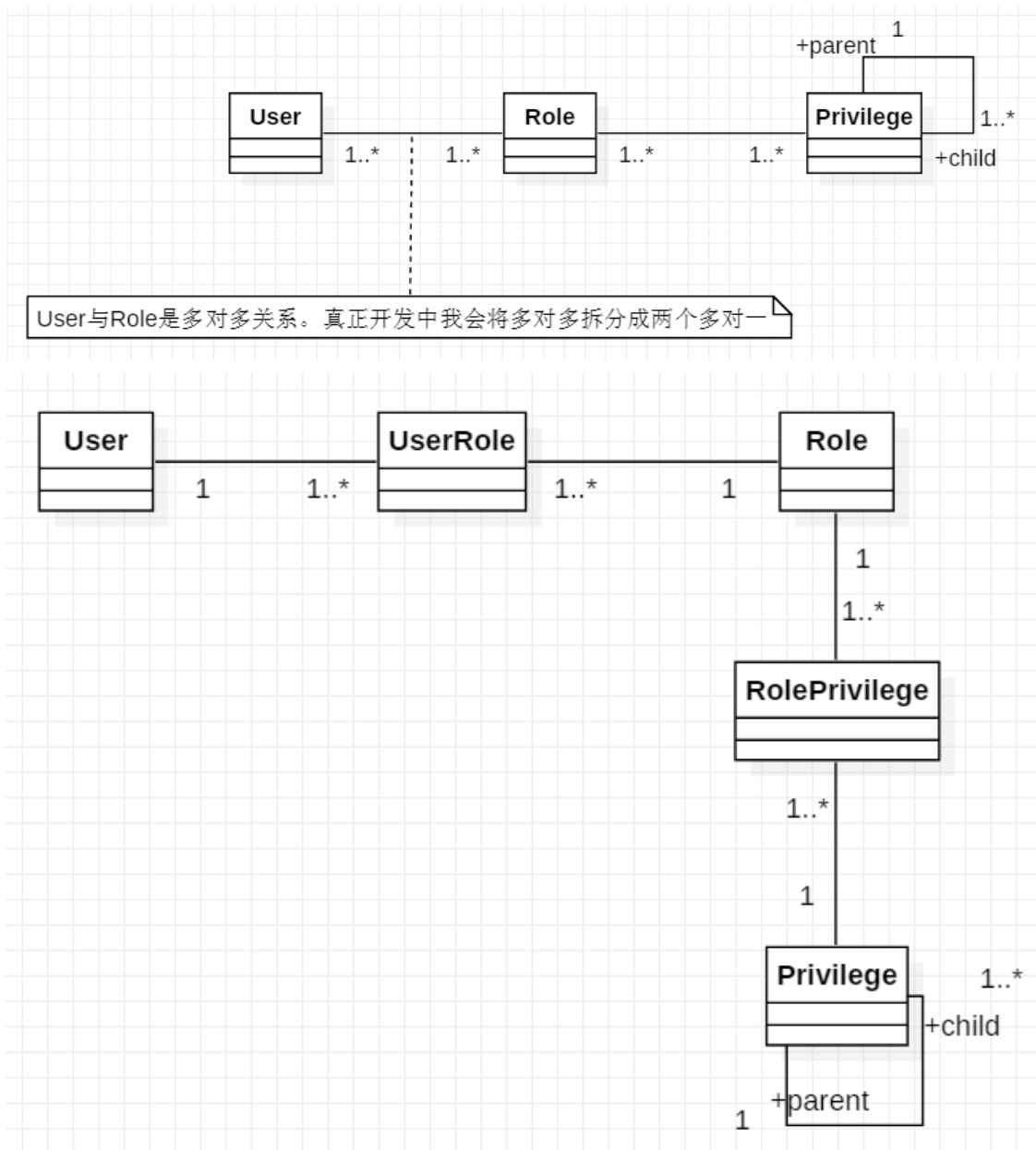
### 8.2 相关术语

- 主体（subject）：用户或第三方系统
- 资源：模块，菜单

- 权限：操作
- 角色：先给角色分配权限，然后将角色分配给用户
- 身份：用户名称
- 凭证：用户密码

## 8.3 权限管理系统模型

- 用户
- 角色
- 权限：菜单和操作都属于权限的概念，因此这个实体的概念是一个无限极分类（Tree）



## 8.4 权限管理系统实现方式

- RBAC（基于角色的访问控制 Role-Based Access Control）
- 基于资源的访问控制（Resource-Based Access Control）
- 基于URL请求访问控制

## 8.5 Spring Security

### 8.5.1 概述

- Spring Security是Spring框架的子项目。主要用来实现权限管理。是采用AOP思想来实现权限管理
- Spring Security与Shiro ( Apache ) 是业界主要的两个权限管理框架
- Spring Security提供了完善的认证机制以及方法级别的授权。

### 8.5.2 官方介绍

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

```
1 Spring Security is a powerful and highly customizable authentication and
 access-control framework. It is the de-facto standard for securing Spring-
 based applications.
2
3 Spring Security is a framework that focuses on providing both authentication
 and authorization to Java applications. Like all Spring projects, the real
 power of Spring Security is found in how easily it can be extended to meet
 custom requirements
```

- 特点

```
1 Comprehensive and extensible support for both Authentication and
 Authorization
2
3 Protection against attacks like session fixation, clickjacking, cross site
 request forgery, etc
4
5 Servlet API integration
6
7 Optional integration with Spring web MVC
8
9 Much more...
```

### 8.5.3 入门案例

- 编写pom.xml

```
1 <dependency>
2 <groupId>org.springframework.security</groupId>
3 <artifactId>spring-security-config</artifactId>
4 <version>5.1.5.RELEASE</version>
5 </dependency>
6
7 <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-
 security-taglibs -->
8 <dependency>
9 <groupId>org.springframework.security</groupId>
10 <artifactId>spring-security-taglibs</artifactId>
11 <version>5.1.5.RELEASE</version>
12 </dependency>
```

- 编写web.xml ( 核心 )

```

1 <filter>
2 <filter-name>springSecurityFilterChain</filter-name>
3 <filter-
4 class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
5 </filter>
6
7 <filter-mapping>
8 <filter-name>springSecurityFilterChain</filter-name>
9 <url-pattern>/*</url-pattern>
10 </filter-mapping>

```

- 编写applicationContext-security.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns:security="http://www.springframework.org/schema/security"
4 xmlns="http://www.springframework.org/schema/beans"
5 xsi:schemaLocation="http://www.springframework.org/schema/beans
6 http://www.springframework.org/schema/beans/spring-beans.xsd
7 http://www.springframework.org/schema/security
8 http://www.springframework.org/schema/security/spring-security.xsd">
9
10 <!--
11 auto-config="true": 自动配置Spring Security组件
12 use-expressions:true表示使用Spring EL表达式配置Spring Security
13 -->
14 <security:http auto-config="true" use-expressions="true">
15 <!-- 需要有ROLE_USER或ROLE_ADMIN角色就可以访问系统资源-->
16 <security:intercept-url pattern="/**"
17 access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"/>
18 </security:http>
19
20 <!-- 配置用户信息-->
21 <security:authentication-manager>
22 <security:authentication-provider>
23 <security:user-service>
24 <security:user name="zhangsan" authorities="ROLE_ADMIN"
25 password="{noop}admin"/>
26 <security:user name="lisi" authorities="ROLE_USER"
27 password="{noop}admin"/>
28 </security:user-service>
29 </security:authentication-provider>
30 </security:authentication-manager>
31 </beans>

```

## 8.5.4 Spring Security过滤器

- SecurityContextPersistenceFilter：在Session中保存SecurityContext(认证信息)。实际上使用的是SecurityContextRepository来实现。
- UsernamePasswordAuthenticationFilter
- LogoutFilter
- CsrfFilter：跨域请求伪造。SpringSecurity会对所有的post请求验证是否包含系统生成的csrf的token值，如果不包含会报错，可以防止csrf攻击
- ...

## 8.5.5 使用自定义的认证页面

```
1 <!--静态资源的放行-->
2 <security:http pattern="/js/**" security="none"/>
3 <security:http pattern="/css/**" security="none"/>
4 <security:http pattern="/images/**" security="none"/>
5
6 <!--
7 auto-config="true": 自动配置Spring Security组件
8 use-expressions:true表示使用Spring EL表达式配置Spring Security
9 -->
10 <security:http auto-config="true" use-expressions="true">
11
12 <!--进入登录页面是可以匿名访问，登录的处理不用认证-->
13 <security:intercept-url pattern="/user/login" access="permitAll()"/>
14
15 <!--需要有ROLE_USER或ROLE_ADMIN角色就可以访问系统资源-->
16 <security:intercept-url pattern="/**"
17 access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"/>
18
19 <!--指定自定义的认证页面-->
20 <!--
21 login-page: 如何进入登录页面
22 login-processing-url: 该值为"/login"
23 default-target-url: 认证成功后的跳转"/index"
24 authentication-failure-forward-url: 认证失败后交给哪一个URL来处理"/user/loginFailure"
25 -->
26 <security:form-login login-page="/user/login"
27 login-processing-url="/login"
28 authentication-success-handler-
29 ref="customAuthenticationSuccessHandler"
30 authentication-failure-handler-
31 ref="customAuthenticationFailureHandler"/>
32
33 <!--禁止csrf防护 跨站请求伪造-->
34 <security:csrf disabled="true"/>
35
36 </security:http>
37
38 <!--配置用户信息-->
39 <security:authentication-manager>
40 <security:authentication-provider>
41 <security:user-service>
42 <security:user name="zhangsan" authorities="ROLE_ADMIN"
43 password="{noop}admin"/>
44 <security:user name="lisi" authorities="ROLE_USER" password="
45 {noop}admin"/>
46 </security:user-service>
47 </security:authentication-provider>
48 </security:authentication-manager>
49
50 <bean id="customAuthenticationSuccessHandler"
51 class="com.bjlemon.security.web.handler.CustomAuthenticationSuccessHandler"
52 />
```

```

48 <bean id="customAuthenticationFailureHandler"
49
 class="com.bjlemon.security.web.handler.CustomAuthenticationFailureHandler"
/>

```

```

1 /**
2 * @author jeffzhou
3 * @version 1.0.0
4 * @ClassName CustomAuthenticationSuccessHandler.java
5 * @Description TODO
6 * @createTime 2020年02月11日 20:36:00
7 */
8 public class CustomAuthenticationSuccessHandler implements
AuthenticationSuccessHandler {
9
10 public void onAuthenticationSuccess(HttpServletRequest request,
11 HttpServletResponse response,
12 Authentication authentication)
13 throws IOException, ServletException {
14 User user = (User) authentication.getPrincipal();
15 HttpSession session = request.getSession(true);
16 session.setAttribute("user", user);
17 response.sendRedirect(request.getContextPath() + "/index");
18 }
19 }

```

```

1 /**
2 * @author jeffzhou
3 * @version 1.0.0
4 * @ClassName CustomAuthenticationFailureHandler.java
5 * @Description TODO
6 * @createTime 2020年02月11日 20:42:00
7 */
8 public class CustomAuthenticationFailureHandler implements
AuthenticationFailureHandler {
9
10 public void onAuthenticationFailure(HttpServletRequest request,
11 HttpServletResponse response,
12 AuthenticationException exception)
13 throws IOException, ServletException {
14 response.sendRedirect(request.getContextPath() + "/error");
15 }
16 }

```

## 8.5.6 认证流程

- 分析UsernamePasswordAuthenticationFilter源码
- 实际上调用了AbstractAuthenticationProcessingFilter的doFilter()

```

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
 throws IOException, ServletException {

 HttpServletRequest request = (HttpServletRequest) req;
 HttpServletResponse response = (HttpServletResponse) res;

 if (!requiresAuthentication(request, response)) {
 chain.doFilter(request, response);

 return;
 }

 if (logger.isDebugEnabled()) {
 logger.debug("Request is to process authentication");
 }

 Authentication authResult;

 try {
 authResult = attemptAuthentication(request, response);
 if (authResult == null) {
 // return immediately as subclass has indicated that it hasn't completed
 // authentication
 return;
 }
 sessionStrategy.onAuthentication(authResult, request, response);
 }
 catch (InternalAuthenticationServiceException failed) {
 logger.error(
 "An internal error occurred while trying to authenticate: " + failed);
 unsuccessfulAuthentication(request, response, failed);

 return;
 }
 catch (AuthenticationException failed) {
 // Authentication success
 if (continueChainBeforeSuccessfulAuthentication) {
 chain.doFilter(request, response);
 }

 successfulAuthentication(request, response, chain, authResult);
 }
}

```

认证失败的处理（自定义认证失败处理器）

认证成功处理（自定义认证成功处理器）

```

public Authentication attemptAuthentication(HttpServletRequest request,
 HttpServletResponse response) throws AuthenticationException {
 if (postOnly && !request.getMethod().equals("POST")) {
 throw new AuthenticationServiceException(
 "Authentication method not supported: " + request.getMethod());
 }

 String username = obtainUsername(request);
 String password = obtainPassword(request);

 if (username == null) {
 username = "";
 }

 if (password == null) {
 password = "";
 }

 username = username.trim();

 UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(
 username, password);

 // Allow subclasses to set the "details" property
 setDetails(request, authRequest);

 return this.getAuthenticationManager().authenticate(authRequest);
}

protected AuthenticationManager getAuthenticationManager() {
 return authenticationManager;
}

protected String obtainUsername(HttpServletRequest request) {
 return request.getParameter(usernameParameter);
}

public static final String SPRING_SECURITY_FORM_USERNAME_KEY = "username";
public static final String SPRING_SECURITY_FORM_PASSWORD_KEY = "password";

private String usernameParameter = SPRING_SECURITY_FORM_USERNAME_KEY;
private String passwordParameter = SPRING_SECURITY_FORM_PASSWORD_KEY;

```

- 真正的认证实现其实是在s中的authenticate()

```
public Authentication authenticate(Authentication authentication) authentication: "org.springframework
 throws AuthenticationException {
 Class<? extends Authentication> toTest = authentication.getClass(); toTest: "class org.springframework
 AuthenticationException lastException = null; lastException: null
 AuthenticationException parentException = null; parentException:
 Authentication result = null; result: null
 Authentication parentResult = null; parentResult: null
 boolean debug = logger.isDebugEnabled(); debug: false

 for (AuthenticationProvider provider : getProviders()) { provider: DaoAuthenticationProvider@6258
 if (!provider.supports(toTest)) { provider: DaoAuthenticationProvider@6258 toTest: "class org
 continue;
 }

 if (debug) {
 logger.debug("Authentication attempt using "
 + provider.getClass().getName());
 }

 try {
 result = provider.authenticate(authentication);

 if (result != null) {
 copyDetails(authentication, result);
 break;
 }
 } catch (AccountStatusException e) {
 prepareException(e, authentication);
 // SEC-546: Avoid polling additional providers if auth failure is due to
 // invalid account status
 throw e;
 } catch (InternalAuthenticationServiceException e) {
 prepareException(e, authentication);
 }
 }
}
```

循环所有的 AuthenticationProvider, 匹配当前认证类型

```
public Authentication authenticate(Authentication authentication) authentication: "org.springframework.secur
 throws AuthenticationException {
 Assert.isInstanceOf(UsernamePasswordAuthenticationToken.class, authentication,
 () -> messages.getMessage("messageSourceAccessor.96266",
 code: "AbstractUserDetailsAuthenticationProvider.onlySupports",
 defaultMessage: "Only UsernamePasswordAuthenticationToken is supported"));

 // Determine username
 String username = (authentication.getPrincipal() == null) ? "NONE_PROVIDED" : authentication.getName(); username: "zhangsan"
 : authentication.getName(); authentication: "org.springframework.security.authentication.UsernamePasswordAuthenticationToken@6258"

 boolean cacheWasUsed = true; cacheWasUsed: false
 UserDetails user = this.userCache.getUserFromCache(username); null userCache: Null

 if (user == null) {
 cacheWasUsed = false; cacheWasUsed: false

 try {
 user = retrieveUser(username, user: null username: "zhangsan"
 (UsernamePasswordAuthenticationToken authentication));
 } catch (UsernameNotFoundException notFound) {
 logger.debug("User '" + username + "' not found");

 if (hideUserNotFoundExceptions) {
 throw new BadCredentialsException(messages.getMessage(
 code: "AbstractUserDetailsAuthenticationProvider.badCredentials",
 defaultMessage: "Bad credentials"));
 } else {
 throw notFound;
 }
 }
 }
 Assert.notNull(user,

 protected final UserDetails retrieveUser(String username, UsernamePasswordAuthenticationToken authentication) throws AuthenticationException {
 prepareException(new InternalAuthenticationServiceException("UserDetailsService returned null, which is an interface contract violation"));
 try {
 UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username);
 if (loadedUser == null) {
 throw new InternalAuthenticationServiceException("UserDetailsService returned null, which is an interface contract violation");
 }
 return loadedUser;
 } catch (UsernameNotFoundException ex) {
 mitigateAgainstTimingAttack(authentication);
 throw ex;
 } catch (InternalAuthenticationServiceException ex) {
 throw ex;
 } catch (Exception ex) {
 throw new InternalAuthenticationServiceException(ex.getMessage(), ex);
 }
 }

 public UserDetails loadUserByUsername(String username) username: "zhangsan"
 throws UsernameNotFoundException {
 UserDetails user = users.getUsernameByUsername(username.toLowerCase()); users: size = 2 user: null

 if (user == null) {
 throw new UsernameNotFoundException(username);
 }

 return new User(user.getUsername(), user.getPassword(), user.isEnabled(),
 user.isAccountNonExpired(), user.isCredentialsNonExpired(),
 user.isAccountNonLocked(), user.getAuthorities());
 }
}
```

1. this.getUserDetailsService()得到 UserDetailsService的实现类的实例  
2. 调用loadUserByUsername()

- 上面的方法最终返回的是UsernamePasswordAuthenticationToken对象



```

public class UsernamePasswordAuthenticationToken extends AbstractAuthenticationToken {
 private static final long serialVersionUID = SpringSecurityCoreVersion.SERIAL_VERSION_UID;

 // ~ Instance fields
 // =====

 private final Object principal;
 private Object credentials;

 // ~ Constructors
 // =====

 /**
 * This constructor can be safely used by any one that wishes to create a
 * <code>UsernamePasswordAuthenticationToken</code>, as the {@link #isAuthenticated()}
 * will return <code>false</code>.
 */
 public UsernamePasswordAuthenticationToken(Object principal, Object credentials) {
 super(authorities: null);
 this.principal = principal;
 this.credentials = credentials;
 setAuthenticated(false);
 }

 public AbstractAuthenticationToken(Collection? extends GrantedAuthority? authorities) {
 if (authorities == null) {
 this.authorities = AuthorityUtils.NO_AUTHORITIES;
 return;
 }
 for (GrantedAuthority a : authorities) {
 if (a == null) {
 throw new IllegalArgumentException(
 "Authorities collection cannot contain any null elements");
 }
 }
 ArrayList<GrantedAuthority> temp = new ArrayList<>(
 authorities.size());
 temp.addAll(authorities);
 this.authorities = Collections.unmodifiableList(temp);
 }
}

```

## 8.5.7 使用数据库完成认证（没有真正访问数据库）

```

1 public interface UserService extends UserDetailsService {
2
3 }

```

```

1 @Service
2 @Transactional
3 public class UserServiceImpl implements UserService {
4
5 @Autowired
6 private UserDao userDao;
7
8 public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
9 List<SimpleGrantedAuthority> authorities = new
ArrayList<SimpleGrantedAuthority>();
10 User user = this.userDao.findByName(username);
11
12 if (user == null) {
13 return null;
14 }
15
16 Set<Role> roles = user.getRoles();
17 if (!CollectionUtils.isEmpty(roles)) {
18 for (Role role : roles) {
19 authorities.add(new SimpleGrantedAuthority(role.getName()));
20 }
21 }
22
23 // 没有加密需要加上"{noop}"
24 // return new
org.springframework.security.core.userdetails.User(user.getName(), "{noop}"
+ user.getPassword(), authorities);
25 return new
org.springframework.security.core.userdetails.User(user.getName(),
user.getPassword(), authorities);
26
27 }
28 }

```

```

1 public interface UserDao {
2
3 User findByName(String name);
4 }

```

```

1 @Repository
2 public class UserDaoImpl implements UserDao {
3
4 @Autowired
5 private BCryptPasswordEncoder bCryptPasswordEncoder;
6
7 public User findByName(String name) {
8 User user = null;
9 if ("zhangsan".equals(name)) {
10 user = new User();
11 user.setId(1);
12 user.setName(name);
13 String password = bCryptPasswordEncoder.encode("admin");
14 user.setPassword(password);
15 user.setSalary(12.34F);
16 user.setBirthday(new Date());
17
18 Set<Role> roles = new HashSet<Role>();
19 roles.add(new Role(1, "ROLE_ADMIN"));
20 user.setRoles(roles);
21 } else if ("lisi".equals(name)) {
22 user = new User();
23 user.setId(2);
24 user.setName(name);
25 String password = bCryptPasswordEncoder.encode("admin");
26 user.setPassword(password);
27 user.setSalary(34.34F);
28 user.setBirthday(new Date());
29
30 Set<Role> roles = new HashSet<Role>();
31 roles.add(new Role(1, "ROLE_USER"));
32 user.setRoles(roles);
33 }
34
35 return user;
36 }
37 }

```

```

1 <security:authentication-manager>
2 <security:authentication-provider user-service-ref="userServiceImpl">
3 <security:password-encoder ref="bCryptPasswordEncoder"/>
4 </security:authentication-provider>
5 </security:authentication-manager>
6
7 <bean id="bCryptPasswordEncoder"
8 class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>

```

## 8.5.8 获取用户的状态（异常处理解决方案）

- 可以根据框架自身提供的User对象来获取相关数据

- 分析User源码

```
public class User implements UserDetails, CredentialsContainer {

 private static final long serialVersionUID = SpringSecurityCoreVersion.SERIAL_VERSION_UID;

 private static final Log logger = LoggerFactory.getLog(User.class);

 // ~ Instance fields
 // =====
 private String password;
 private final String username;
 private final Set<GrantedAuthority> authorities;
 private final boolean accountNonExpired;
 private final boolean accountNonLocked;
 private final boolean credentialsNonExpired;
 private final boolean enabled;
```

- 上述源码我们可以知道用户的信息包括了：username，password，authorities，enabled（可用），accountNonExpired（账户是否失效），accountNonLock（账户是否锁定），credentialsNonExpired（凭证是否失效）

```
1 public class CustomAuthenticationFailureHandler implements
 AuthenticationFailureHandler {
2
3 public void onAuthenticationFailure(HttpServletRequest request,
4 HttpServletResponse response,
5 AuthenticationException exception)
6 throws IOException, ServletException {
7 HttpSession session = request.getSession(true);
8 if (exception instanceof DisabledException) {
9 session.setAttribute("errorMessage", "账户不可用");
10 } else if (exception instanceof AccountExpiredException) {
11 session.setAttribute("errorMessage", "账户过期");
12 } else if (exception instanceof CredentialsExpiredException) {
13 session.setAttribute("errorMessage", "账户凭证过期");
14 } else if (exception instanceof LockedException) {
15 session.setAttribute("errorMessage", "账户已锁住");
16 } else if (exception instanceof
17 InternalAuthenticationServiceException) {
18 session.setAttribute("errorMessage", "账户不存在");
19 } else if (exception instanceof BadCredentialsException) {
20 session.setAttribute("errorMessage", "账户凭证错误");
21 }
22 response.sendRedirect(request.getContextPath() + "/error");
23 }
24 }
```

## 8.5.9 注销账户

```
1 退出
```

```

public class SecurityContextLogoutHandler implements LogoutHandler {
 protected final Log logger = LoggerFactory.getLog(this.getClass()); logger: LogAdapter$Slf4jLog@6218

 private boolean invalidateHttpSession = true; invalidateHttpSession: true
 private boolean clearAuthentication = true; clearAuthentication: true

 public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication) { authentication: "org.springframework.
 Assert.notNull(request, message: "HttpServletRequest required"); request:
 if (invalidateHttpSession) {
 HttpSession session = request.getSession(create: false);
 if (session != null) {
 logger.debug(0: "Invalidating session: " + session.getId());
 session.invalidate();
 }
 }

 if (clearAuthentication) {
 SecurityContext context = SecurityContextHolder.getContext();
 context.setAuthentication(null);
 }

 SecurityContextHolder.clearContext();
 }
}

```

- 注意：一旦开启了csrf(cross)功能，logout处理器只支持POST提交

```

@Override
public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {
 for (LogoutHandler handler : this.logoutHandlers) {
 handler.logout(request, response, authentication);
 }
}

public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {
 this.csrfTokenRepository.saveToken(token: null, request, response);
}

public void logout(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {
 Assert.notNull(request, message: "HttpServletRequest required");
 if (invalidateHttpSession) {
 HttpSession session = request.getSession(create: false);
 if (session != null) {
 logger.debug(0: "Invalidating session: " + session.getId());
 session.invalidate();
 }
 }

 if (clearAuthentication) {
 SecurityContext context = SecurityContextHolder.getContext();
 context.setAuthentication(null);
 }

 SecurityContextHolder.clearContext();
}

```

有两个LogoutHandler  
1.CsrfLogoutHandler  
2.SecurityContextLogoutHandler

CsrfLogoutHandler

SecurityContextLogoutHandler

## 8.5.10 remember me

- 分析源码

```

public abstract class AbstractRememberMeServices implements RememberMeServices,
 InitializingBean, LogoutHandler {

 @Override
 public final void loginSuccess(HttpServletRequest request,
 HttpServletResponse response, Authentication successfulAuthentication) {

 if (!rememberMeRequested(request, parameter)) {
 logger.debug("Remember-me login not requested.");
 return;
 }

 onLoginSuccess(request, response, successfulAuthentication);
 }

 protected boolean rememberMeRequested(HttpServletRequest request, String parameter) {

 if (alwaysRemember) {
 return true;
 }

 String paramValue = request.getParameter(parameter);

 if (paramValue != null) {
 if (paramValue.equalsIgnoreCase("true") || paramValue.equalsIgnoreCase("on")
 || paramValue.equalsIgnoreCase("yes") || paramValue.equalsIgnoreCase("1")) {
 return true;
 }
 }

 if (logger.isDebugEnabled()) {
 logger.debug("Did not send remember-me cookie (principal did not set parameter '"
 + parameter + "')");
 }

 return false;
 }
}

```

判断页面是否勾选了记住我

```

protected void onLoginSuccess(HttpServletRequest request,
 HttpServletResponse response, Authentication successfulAuthentication) {
 String username = successfulAuthentication.getName(); 获取用户名

 logger.debug("Creating new persistent login for user " + username);

 PersistentRememberMeToken persistentToken = new PersistentRememberMeToken(
 username, generateSeriesData(), generateTokenData(), new Date()); 创建记住我的token

 try {
 tokenRepository.createNewToken(persistentToken); 将token持久化了数据库
 addCookie(persistentToken, request, response); 将token写入到cookie中
 } catch (Exception e) {
 logger.error("Failed to save persistent token ", e);
 }
}

```

- 如何实现

```

1 <%@ page contentType="text/html; charset=UTF-8" language="java"
 isELIgnored="false" %>
2 <%@ taglib prefix="security"
 uri="http://www.springframework.org/security/tags" %>
3 <html>
4 <head>
5 <title>Title</title>
6 </head>
7 <body>
8 <form action="${pageContext.request.contextPath}/login" method="post">
9 <security:csrfInput/>
10 用户名称:<input type="text" name="username">

11 用户密码:<input type="password" name="password">


```

```

12 <input type="checkbox" name="remember-me" value="true">记住我
13 <input type="submit" value="登录">

14 </form>
15 </body>
16 </html>

```

```

1 <!--开启了rememberme过滤器，设置token添加的cookie有效时间为60秒-->
2 <security:remember-me token-validity-seconds="60"/>

```

- 如果需要实现Token的持久化，那么需要建立一张表（persistent\_logins（username, series, token, last\_used））

## 8.5.11 如何在页面上显示用户数据

```

1 欢迎${user.username}登录系统!

2 欢迎<security:authentication property="name"/>登录系统!

3 欢迎<security:authentication property="principal.username"/>登录系统!


```

## 8.5.12 授权

- 授权就是进行权限的访问控制（判断当前用户对这个资源是否可以访问）
- 第一种实现方式：如果这个用户没有这个权限，那么这个资源就不要出现（不要显示在页面上）

```

1 <security:authorize access="hasAnyRole('ROLE_ADMIN')">
2 用户添加
3 用户删除
4 用户修改
5 用户查询
6 </security:authorize>
7
8 <security:authorize access="hasAnyRole('ROLE_USER')">
9 用户查询
10 </security:authorize>

```

- 第二种实现方式：资源还是全部显示，但是当该用户点击了这个资源，提示用户没有权限
  - 使用注解的方式来控制
  - 需要打开注解支持
    - 如果注解放在controller上，对应的注解应该放在mvc配置文件中（web容器）
    - 如果注解放在service上，对应的注解放在spring配置文件上（根容器）

```

1 <security:global-method-security jsr250-annotations="enabled" pre-post-
2 annotations="enabled" secured-annotations="enabled"/>

```

```

1 @Controller
2 @RequestMapping("/user")
3 public class UserController {
4
5 @GetMapping("/login")

```

```

6 public String login() {
7 return "login";
8 }
9
10 @GetMapping("/list")
11 public String list() {
12 System.out.println("查询列表");
13 return "list";
14 }
15
16 @GetMapping("/add")
17 @ResponseBody
18 @Secured({"ROLE_ADMIN"})
19 public String add() {
20 System.out.println("add");
21 return "success";
22 }
23
24 @GetMapping("/delete")
25 @ResponseBody
26 @Secured({"ROLE_ADMIN"})
27 public String delete() {
28 System.out.println("delete");
29 return "success";
30 }
31
32 @GetMapping("/edit")
33 // @Secured({"ROLE_ADMIN"})
34 @ResponseBody
35 @PreAuthorize(value = "hasAnyRole('ROLE_ADMIN')")
36 public String edit() {
37 System.out.println("edit");
38 return "success";
39 }
40
41 @GetMapping("/findAll")
42 @ResponseBody
43 // @Secured({"ROLE_ADMIN", "ROLE_USER"})
44 @PreAuthorize(value = "hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
45 public String findAll() {
46 System.out.println("findAll");
47 return "success";
48 }
49
50 }

```

- 如果出现授权失败，那么页面会显示403错误。但是这个页面友好度很差。
  - 交给spring security给我们解决

```

1 public class CustomDeniedHandler implements AccessDeniedHandler {
2
3 public void handle(HttpServletRequest request,
4 HttpServletResponse response,
5 AccessDeniedException accessDeniedException)
6 throws IOException, ServletException {
7 HttpSession session = request.getSession(true);
8 session.setAttribute("errorMessage", "您没有权限!");
9 response.sendRedirect("/error");
10 }
11 }

```

```

1 <security:access-denied-handler ref="customDeniedHandler"/>

```

- 交给spring mvc来处理

```

1 @ControllerAdvice
2 public class CustomExceptionHandlerResolver {
3
4 @ExceptionHandler(AccessDeniedException.class)
5 public String handleException(HttpSession session) {
6 session.setAttribute("errorMessage", "您没有权限!");
7 return "redirect:/error";
8 }
9 }

```

## 8.6 SpringBoot与Spring Security整合（单体应用）

### 8.6.1 Spring Security提供的登录页面

```

1 <dependency>
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-starter-security</artifactId>
4 </dependency>
5 <dependency>
6 <groupId>org.springframework.boot</groupId>
7 <artifactId>spring-boot-starter-thymeleaf</artifactId>
8 </dependency>
9 <dependency>
10 <groupId>org.springframework.boot</groupId>
11 <artifactId>spring-boot-starter-web</artifactId>
12 </dependency>
13
14 <dependency>
15 <groupId>org.springframework.boot</groupId>
16 <artifactId>spring-boot-devtools</artifactId>
17 <scope>runtime</scope>
18 <optional>true</optional>
19 </dependency>
20 <dependency>
21 <groupId>org.springframework.boot</groupId>
22 <artifactId>spring-boot-configuration-processor</artifactId>
23 <optional>true</optional>
24 </dependency>

```



```

25 <dependency>
26 <groupId>org.projectlombok</groupId>
27 <artifactId>lombok</artifactId>
28 <optional>true</optional>
29 </dependency>
30 <dependency>
31 <groupId>org.springframework.boot</groupId>
32 <artifactId>spring-boot-starter-test</artifactId>
33 <scope>test</scope>
34 <exclusions>
35 <exclusion>
36 <groupId>org.junit.vintage</groupId>
37 <artifactId>junit-vintage-engine</artifactId>
38 </exclusion>
39 </exclusions>
40 </dependency>
41 <dependency>
42 <groupId>org.springframework.security</groupId>
43 <artifactId>spring-security-test</artifactId>
44 <scope>test</scope>
45 </dependency>

```

## 8.6.2 使用自定义的认证页面

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://maven.apache.org/POM/4.0.0"
4 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5 https://maven.apache.org/xsd/maven-4.0.0.xsd">
6 <modelVersion>4.0.0</modelVersion>
7 <parent>
8 <groupId>org.springframework.boot</groupId>
9 <artifactId>spring-boot-starter-parent</artifactId>
10 <version>2.2.4.RELEASE</version>
11 <relativePath/> <!-- lookup parent from repository -->
12 </parent>
13 <groupId>com.bjlemon</groupId>
14 <artifactId>springboot-security-demo</artifactId>
15 <version>0.0.1-SNAPSHOT</version>
16 <name>springboot-security-demo</name>
17 <packaging>war</packaging>
18
19 <properties>
20 <java.version>1.8</java.version>
21 </properties>
22
23 <dependencies>
24 <dependency>
25 <groupId>org.springframework.boot</groupId>
26 <artifactId>spring-boot-starter-security</artifactId>
27 </dependency>
28
29 <dependency>
30 <groupId>org.springframework.security</groupId>
31 <artifactId>spring-security-taglibs</artifactId>
32 <version>5.2.1.RELEASE</version>
33 </dependency>

```

```
32
33 <dependency>
34 <groupId>org.springframework.boot</groupId>
35 <artifactId>spring-boot-starter-web</artifactId>
36 </dependency>
37
38 <dependency>
39 <groupId>org.mybatis.spring.boot</groupId>
40 <artifactId>mybatis-spring-boot-starter</artifactId>
41 <version>2.1.1</version>
42 </dependency>
43
44 <!--
45 https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper-
46 spring-boot-starter -->
47 <dependency>
48 <groupId>com.github.pagehelper</groupId>
49 <artifactId>pagehelper-spring-boot-starter</artifactId>
50 <version>1.2.13</version>
51 </dependency>
52
53 <dependency>
54 <groupId>com.alibaba</groupId>
55 <artifactId>druid-spring-boot-starter</artifactId>
56 <version>1.1.21</version>
57 </dependency>
58
59 <dependency>
60 <groupId>org.springframework.boot</groupId>
61 <artifactId>spring-boot-devtools</artifactId>
62 <scope>runtime</scope>
63 <optional>true</optional>
64 </dependency>
65 <dependency>
66 <groupId>mysql</groupId>
67 <artifactId>mysql-connector-java</artifactId>
68 <scope>runtime</scope>
69 </dependency>
70
71 <dependency>
72 <groupId>org.springframework.boot</groupId>
73 <artifactId>spring-boot-configuration-processor</artifactId>
74 <optional>true</optional>
75 </dependency>
76 <dependency>
77 <groupId>org.projectlombok</groupId>
78 <artifactId>lombok</artifactId>
79 <optional>true</optional>
80 </dependency>
81 <dependency>
82 <groupId>org.springframework.boot</groupId>
83 <artifactId>spring-boot-starter-test</artifactId>
84 <scope>test</scope>
85 <exclusions>
86 <exclusion>
87 <groupId>org.junit.vintage</groupId>
88 <artifactId>junit-vintage-engine</artifactId>
```

```

88 </exclusion>
89 </exclusions>
90 </dependency>
91 <dependency>
92 <groupId>org.springframework.security</groupId>
93 <artifactId>spring-security-test</artifactId>
94 <scope>test</scope>
95 </dependency>
96
97 <dependency>
98 <groupId>org.apache.commons</groupId>
99 <artifactId>commons-lang3</artifactId>
100 <version>3.9</version>
101 </dependency>
102
103 <dependency>
104 <groupId>commons-collections</groupId>
105 <artifactId>commons-collections</artifactId>
106 <version>3.2.2</version>
107 </dependency>
108
109 <dependency>
110 <groupId>org.springframework.boot</groupId>
111 <artifactId>spring-boot-starter-tomcat</artifactId>
112 </dependency>
113
114 <dependency>
115 <groupId>org.apache.tomcat.embed</groupId>
116 <artifactId>tomcat-embed-jasper</artifactId>
117 </dependency>
118 </dependencies>
119
120 <build>
121 <plugins>
122 <plugin>
123 <groupId>org.springframework.boot</groupId>
124 <artifactId>spring-boot-maven-plugin</artifactId>
125 </plugin>
126 </plugins>
127 </build>
128 </project>

```

```

1 server:
2 port: 8080
3 spring:
4 application:
5 name: springboot-security-demo
6 datasource:
7 driver-class-name: com.mysql.cj.jdbc.Driver
8 url: jdbc:mysql:///springboot_security_demo?
9 useUnicode=true&characterEncoding=utf8&serverTimezone=UTC
10 username: root
11 password: root
12 type: com.alibaba.druid.pool.DruidDataSource
13 druid:
14 initial-size: 5
15 max-active: 20

```

```

15 min-idle: 5
16 max-wait: 60000
17 mvc:
18 view:
19 suffix: .jsp
20 prefix: /WEB-INF/views/
21 mybatis:
22 mapper-locations: classpath:/mappers/*.xml
23 type-aliases-package: com.bjlemon.springboot.domain
24 configuration:
25 lazy-loading-enabled: true
26 aggressive-lazy-loading: false
27 cache-enabled: true

```

```

1 package com.bjlemon.springboot.service.impl;
2
3 import com.bjlemon.springboot.domain.*;
4 import com.bjlemon.springboot.mapper.RoleMapper;
5 import com.bjlemon.springboot.mapper.UserMapper;
6 import com.bjlemon.springboot.mapper.UserRoleMapper;
7 import com.bjlemon.springboot.service.UserService;
8 import org.apache.commons.collections.CollectionUtils;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.security.core.GrantedAuthority;
11 import org.springframework.security.core.authority.SimpleGrantedAuthority;
12 import org.springframework.security.core.userdetails.UserDetails;
13 import
14 org.springframework.security.core.userdetails.UsernameNotFoundException;
15 import org.springframework.stereotype.Service;
16 import org.springframework.transaction.annotation.Transactional;
17
18 import java.util.ArrayList;
19 import java.util.List;
20
21 /**
22 * @author jeffzhou
23 * @version 1.0.0
24 * @ClassName UserServiceImpl.java
25 * @Description TODO
26 * @createTime 2020年02月18日 20:44:00
27 */
28 @Service
29 @Transactional
30 public class UserServiceImpl implements UserService {
31
32 @Autowired
33 private UserMapper userMapper;
34
35 @Autowired
36 private UserRoleMapper userRoleMapper;
37
38 @Autowired
39 private RoleMapper roleMapper;
40
41 /**
42 * @description 根据用户名称查询该用户，同时将该用户的所有的权限查询出来

```

```

43 * @author admin
44 * @updateTime 2020/2/18 20:49
45 */
46 @Override
47 public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
48 UserDetails userDetails = null;
49 List<GrantedAuthority> grantedAuthorityList = new ArrayList<>();
50 List<String> roleNameList = new ArrayList<>();
51
52 UserExample userExample = new UserExample();
53 UserExample.Criteria criteria = userExample.createCriteria();
54 criteria.andUserNameEqualTo(username);
55 List<User> userList =
this.userMapper.selectByExample(userExample);
56
57 if (CollectionUtils.isNotEmpty(userList)) {
58 User user = userList.get(0);
59
60 // TODO 查询该用户的所有的角色
61 UserRoleExample userRoleExample = new UserRoleExample();
62 UserRoleExample.Criteria userRoleExampleCriteria =
userRoleExample.createCriteria();
63 userRoleExampleCriteria.andUserIdEqualTo(user.getUserId());
64 List<UserRoleKey> userRoleKeyList =
this.userRoleMapper.selectByExample(userRoleExample);
65 if (CollectionUtils.isNotEmpty(userRoleKeyList)) {
66 for (UserRoleKey userRoleKey : userRoleKeyList) {
67 Integer roleId = userRoleKey.getRoleId();
68 Role role =
this.roleMapper.selectByPrimaryKey(roleId);
69 String roleName = role.getRoleName();
70 roleNameList.add(roleName);
71 }
72 }
73
74 // TODO 角色封装成SimpleGrantedAuthority
75 for (String roleName : roleNameList) {
76 SimpleGrantedAuthority grantedAuthority = new
SimpleGrantedAuthority(roleName);
77 grantedAuthorityList.add(grantedAuthority);
78 }
79
80 // TODO 根据用户查询该用户的权限
81 List<Permission> permissionList =
this.userMapper.findPermissionsByUserId(user.getUserId());
82 if (CollectionUtils.isNotEmpty(permissionList)) {
83 for (Permission permission : permissionList) {
84 String permissionName =
permission.getPermissionName();
85 SimpleGrantedAuthority grantedAuthority = new
SimpleGrantedAuthority(permissionName);
86 grantedAuthorityList.add(grantedAuthority);
87 }
88 }
89 }
90 }

```

```

91 userDetails = new
org.springframework.security.core.userdetails.User(user.getUserName(),
92 user.getUserPassword(),
93 user.getUserStatus() == 1,
94 true,
95 true,
96 true,
97 grantedAuthorityList);
98 } else {
99 return null;
100 }
101 return userDetails;
102 }
103 }

```

```

1 package com.bjlemon.springboot.config;
2
3 import com.bjlemon.springboot.encoder.CustomPasswordEncoder;
4 import com.bjlemon.springboot.service.UserService;
5 import
com.bjlemon.springboot.web.handler.CustomAuthenticationFailureHandler;
6 import
com.bjlemon.springboot.web.handler.CustomAuthenticationSuccessHandler;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.context.annotation.Configuration;
9 import
org.springframework.security.config.annotation.authentication.builders.Authen
nticationManagerBuilder;
10 import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
11 import
org.springframework.security.config.annotation.web.configuration.EnablewebSe
curity;
12 import
org.springframework.security.config.annotation.web.configuration.WebSecurity
ConfigurerAdapter;
13
14 /**
15 * @author jeffzhou
16 * @version 1.0.0
17 * @ClassName SecurityConfig.java
18 * @Description TODO
19 * @createTime 2020年02月18日 20:45:00
20 */
21 @Configuration
22 @EnablewebSecurity
23 public class SecurityConfig extends WebSecurityConfigurerAdapter {
24
25 @Autowired
26 private UserService userService;
27
28 @Override
29 protected void configure(HttpSecurity http) throws Exception {
30 http.authorizeRequests()
31 .antMatchers("/error").permitAll()
32 // .antMatchers("/user/login").permitAll() // 当请求
是"/user/login"时，不需要认证。但是由于指定了登录的跳转，这行代码可以不用写出

```

```

33 // .antMatchers("/**").hasAnyRole("ADMIN", "USER")
34 .anyRequest().authenticated() // 除了上面的请求，都需要进
行认证
35 .and()
36 .formLogin()
37 .loginPage("/user/login")
38 .loginProcessingUrl("/login")
39 .successHandler(new CustomAuthenticationSuccessHandler())
40 .failureHandler(new CustomAuthenticationFailureHandler())
41 .permitAll()
42 .and()
43 .logout()
44 .logoutUrl("/logout")
45 .logoutSuccessUrl("/user/login")
46 .invalidateHttpSession(true)
47 .permitAll()
48 .and()
49 .csrf()
50 .disable();
51 }
52
53 @Override
54 protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
55 auth.userDetailsService(this.userService).passwordEncoder(new
CustomPasswordEncoder());
56 }
57 }

```

```

1 @SpringBootApplication
2 @MapperScan(value = "com.bjlemon.springboot.mapper")
3 @EnableGlobalMethodSecurity(jsr250Enabled = true, prePostEnabled = true,
securedEnabled = true)
4 public class SpringbootSecurityDemoApplication {
5
6 public static void main(String[] args) {
7 SpringApplication.run(SpringbootSecurityDemoApplication.class,
args);
8 }
9
10 }

```

```

1 package com.bjlemon.springboot.web.controller;
2
3 import org.springframework.security.access.annotation.Secured;
4 import org.springframework.security.access.prepost.PreAuthorize;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.ResponseBody;
9
10 /**
11 * @author jeffzhou
12 * @version 1.0.0
13 * @ClassName UserController.java
14 * @Description TODO

```

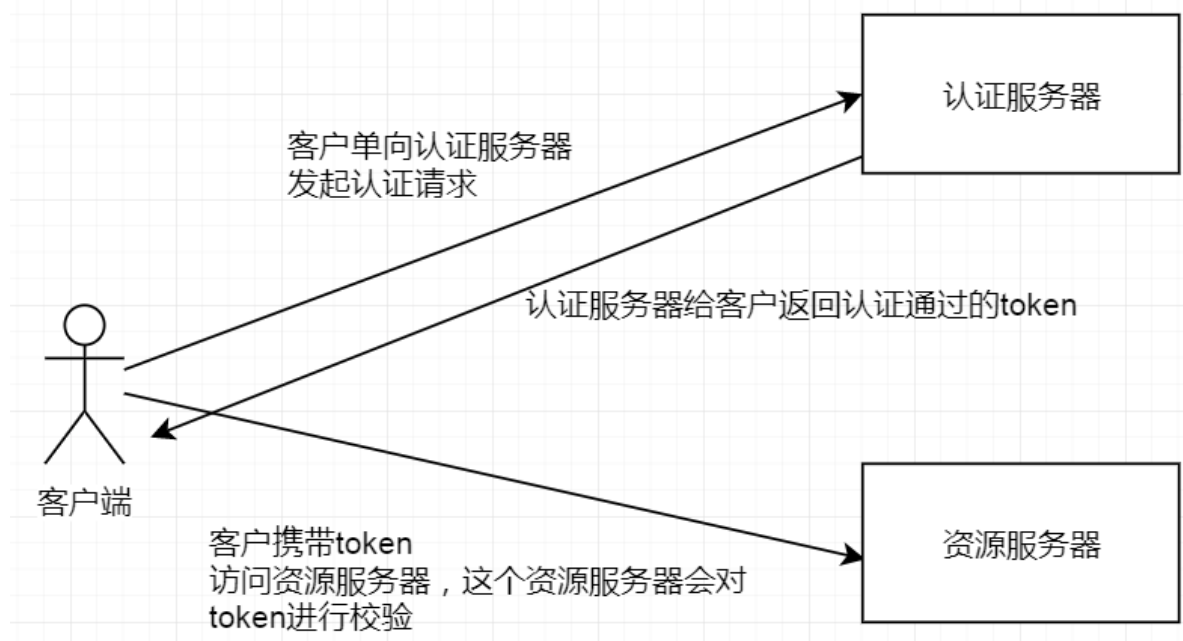
```
15 * @createTime 2020年02月15日 21:54:00
16 */
17 @Controller
18 @RequestMapping("/user")
19 public class UserController {
20
21 @GetMapping("/login")
22 public String login() {
23 return "login";
24 }
25
26
27 @GetMapping("/add")
28 @ResponseBody
29 // @Secured({"ROLE_ADMIN"})
30 // @PreAuthorize(value = "hasAuthority('user:add')")
31 @PreAuthorize(value = "hasAuthority('user:add')")
32 public String add() {
33 System.out.println("add");
34 return "success";
35 }
36
37 @GetMapping("/delete")
38 @ResponseBody
39 // @Secured({"ROLE_ADMIN"})
40 @PreAuthorize(value = "hasAuthority('user:delete')")
41 public String delete() {
42 System.out.println("delete");
43 return "success";
44 }
45
46 @GetMapping("/edit")
47 // @Secured({"ROLE_ADMIN"})
48 @ResponseBody
49 @PreAuthorize(value = "hasAuthority('user:edit')")
50 public String edit() {
51 System.out.println("edit");
52 return "success";
53 }
54
55 @GetMapping("/findAll")
56 @ResponseBody
57 @Secured({"ROLE_ADMIN", "ROLE_USER"})
58 // @PreAuthorize(value = "hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
59 public String findAll() {
60 System.out.println("findAll");
61 return "success";
62 }
63 }
```

## 九.分布式认证授权的解决方案

### 9.1 概念



- 分布式认证也就是单点登录
- 单点登录：在分布式系统中，用户只需认证一次（认证服务器）就可以访问其他资源系统（其他的访问）
- 传统实现
  - Session共享：一旦认证成功，这台服务器就会产生一个Session，但是默认情况下Session不会共享
- 基于JWT+RSA



- 上述的方案实际上就是两大步骤
  - 用户认证
  - 身份校验

## 9.2 JWT

- JWT全称：JSON Web Token
- JWT的Token由三部分组成([http://www.ruanyifeng.com/blog/2018/07/json\\_web\\_token-tutorial.html](http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html))
  - 头部 ( Header )
  - 载荷 ( Payload )
  - 签名 ( Signature )

```

1 | HMACSHA256(
2 | base64UrlEncode(header) + "." +
3 | base64UrlEncode(payload),
4 | secret)

```

- 签名很重要，但是安全性不高，我们必须加密

## 9.3 RSA

- RSA称为非对称加密算法
- 公钥和私钥

## 9.4 SpringSecurity + JWT + RSA

---