SpringBoot基础



学习目标：



1. 能够理解Spring的优缺点
2. 能够理解SpringBoot的特点
3. 能够理解SpringBoot的核心功能
4. 能够搭建SpringBoot的环境
5. 能够完成application.properties配置文件的配置
6. 能够完成application.yml配置文件的配置
7. 能够使用SpringBoot集成Mybatis
8. 能够使用SpringBoot集成Junit
9. 能够使用SpringBoot集成SpringData JPA

一、SpringBoot简介



1.1 原有Spring优缺点分析



1.1.1 Spring的优点分析

Spring是Java企业版（Java Enterprise Edition，JEE，也称J2EE）的轻量级代替品。无需开发重量级的Enterprise

JavaBean（EJB），Spring为企业级Java开发提供了一种相对简单的方法，通过依赖注入和面向切面编程，用简单

的Java对象（Plain Old Java Object，POJO）实现了EJB的功能。

1.1.2 Spring的缺点分析

虽然Spring的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring用XML配置，而且是很多XML配置。Spring 2.5引入了基于注解的组件扫描，这消除了大量针对应用程序自身组件的显式XML配置。Spring 3.0引入了基于Java的配置，这是一种类型安全的可重构配置方式，可以代替XML。

所有这些配置都代表了开发时的损耗。因为在思考Spring特性配置和解决业务问题之间需要进行思维切换，所以编写配置挤占了编写应用程序逻辑的时间。和所有框架一样，Spring实用，但与此同时它要求的回报也不少。

除此之外，项目的依赖管理也是一件耗时耗力的事情。在环境搭建时，需要分析要导入哪些库的坐标，而且还需要分析导入与之有依赖关系的其他库的坐标，一旦选错了依赖的版本，随之而来的不兼容问题就会严重阻碍项目的开发进度。

1.2 SpringBoot的概述



1.2.1 SpringBoot解决上述Spring的缺点

SpringBoot对上述Spring的缺点进行的改善和优化，基于约定优于配置的思想，可以让开发人员不必在配置与逻辑业务之间进行思维的切换，全身心的投入到逻辑业务的代码编写中，从而大大提高了开发的效率，一定程度上缩短了项目周期。

1.2.2 SpringBoot的特点

 为基于Spring的开发提供更快的入门体验

 开箱即用，没有代码生成，也无需XML配置。同时也可以修改默认值来满足特定的需求

 提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等  SpringBoot不是对Spring功能上的增强，而是提供了一种快速使用Spring的方式

1.2.3 SpringBoot的核心功能

 起步依赖

起步依赖本质上是一个Maven项目对象模型（Project Object Model，POM），定义了对其他库的传递依赖，这些东西加在一起即支持某项功能。

简单的说，起步依赖就是将具备某种功能的坐标打包到一起，并提供一些默认的功能。

 自动配置

Spring Boot的自动配置是一个运行时（更准确地说，是应用程序启动时）的过程，考虑了众多因素，才决定Spring配置应该用哪个，不该用哪个。该过程是Spring自动完成的。

注意：起步依赖和自动配置的原理剖析会在第三章《SpringBoot原理分析》进行详细讲解

二、SpringBoot快速入门

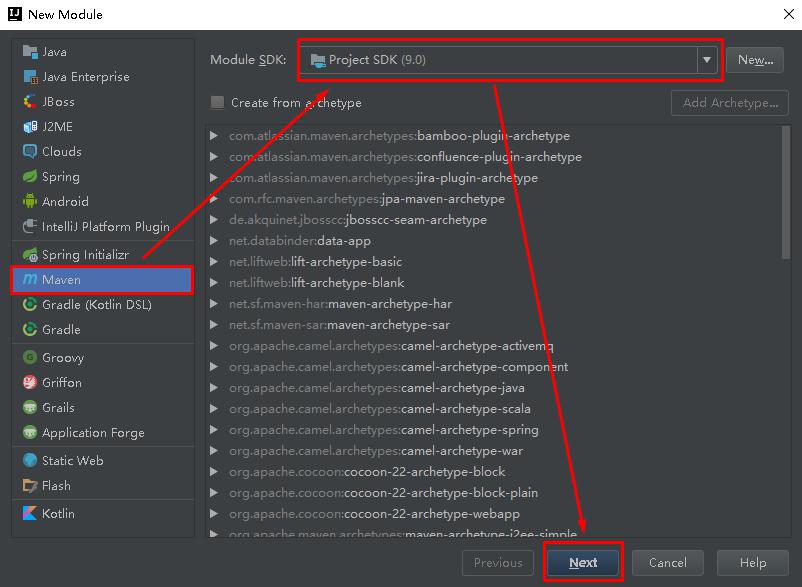


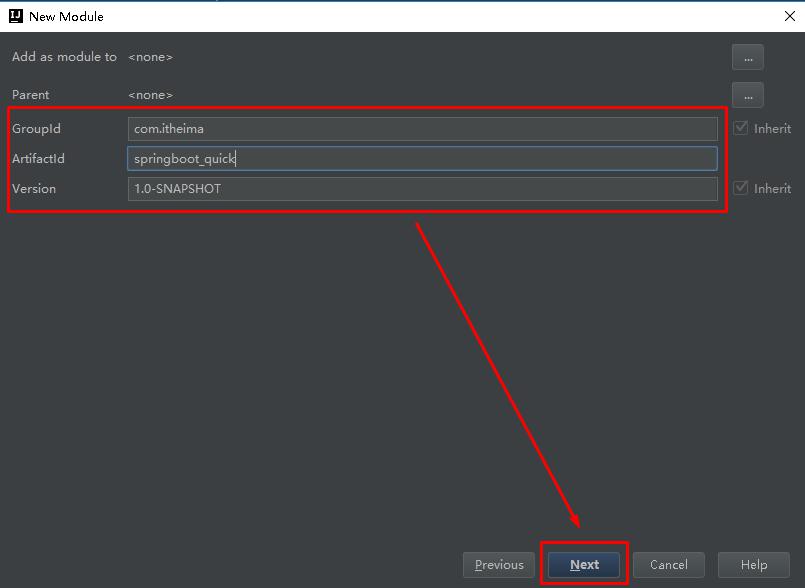
2.1 代码实现

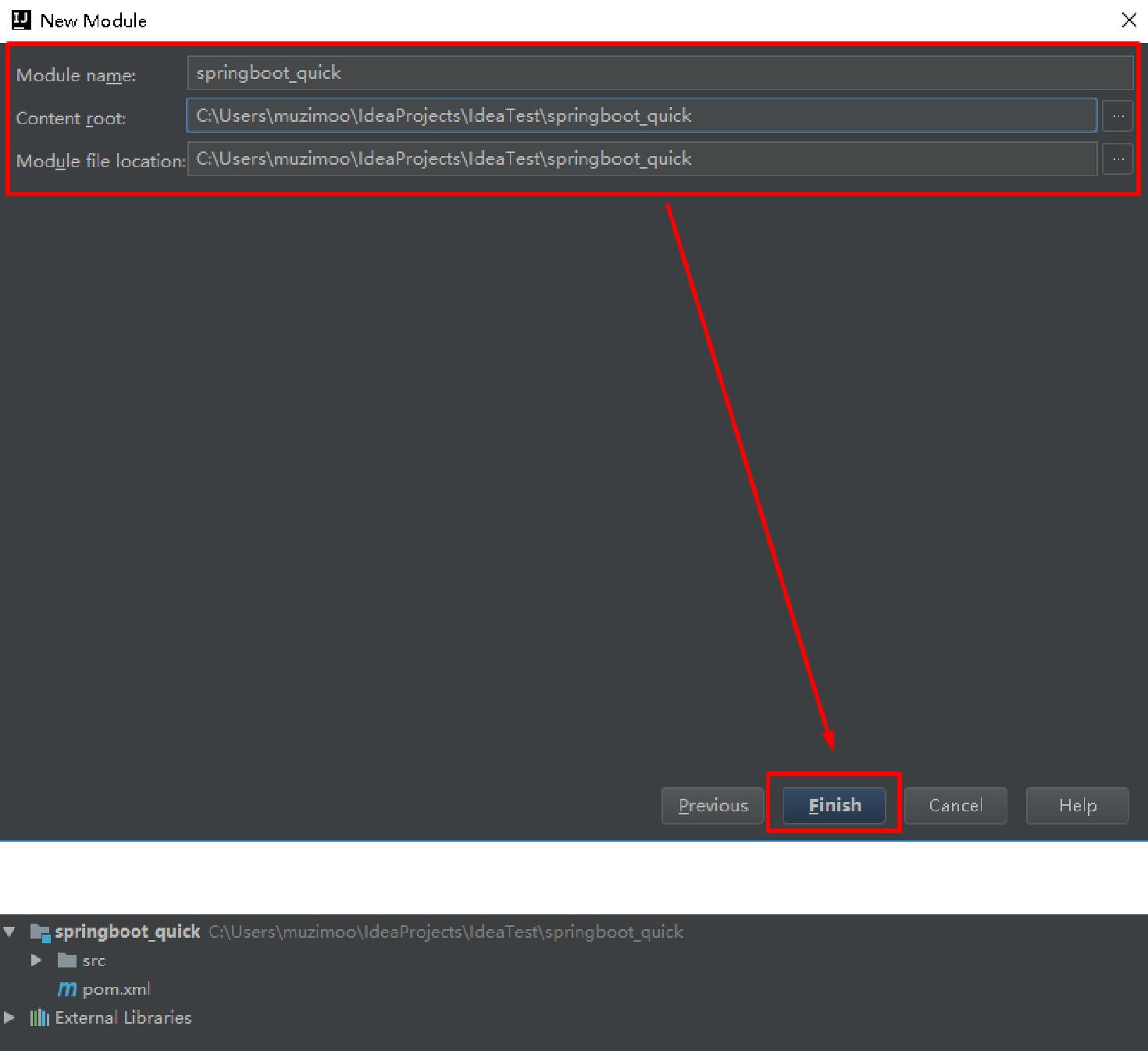


2.1.1 创建Maven工程

使用idea工具创建一个maven工程，该工程为普通的java工程即可

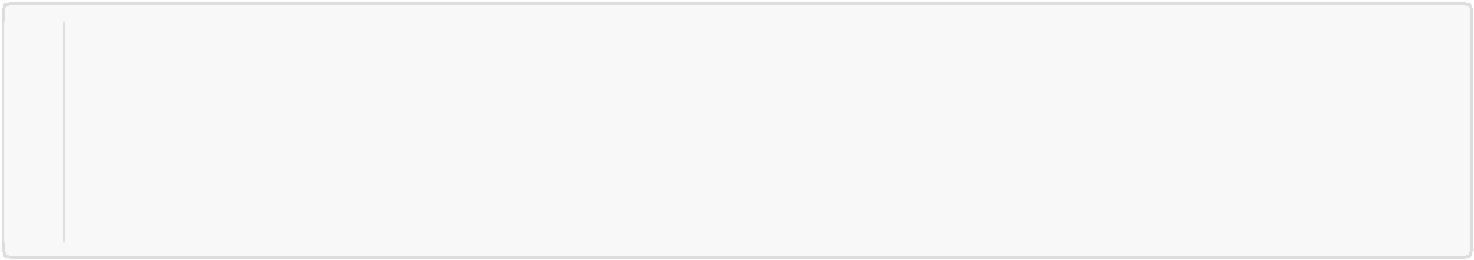






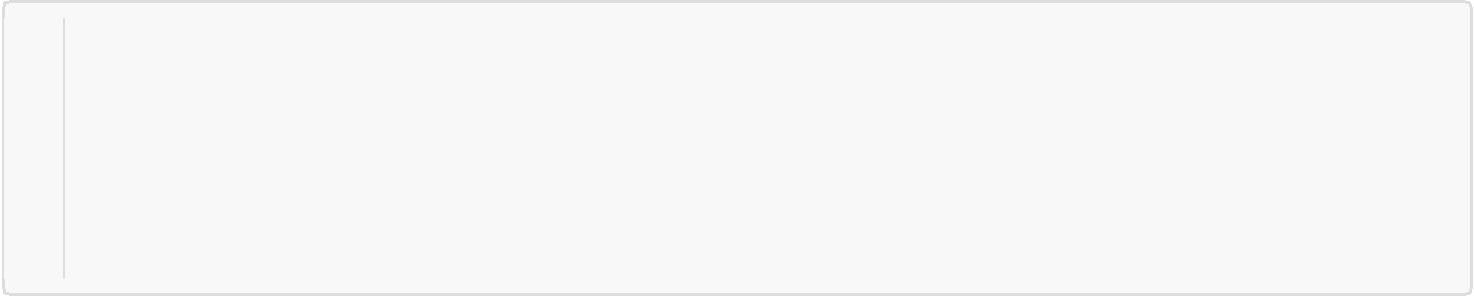
2.1.2 添加SpringBoot的起步依赖

SpringBoot要求，项目要继承SpringBoot的起步依赖spring-boot-starter-parent



* <parent>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-parent</artifactId>
* <version>2.0.1.RELEASE</version> 5 </parent>

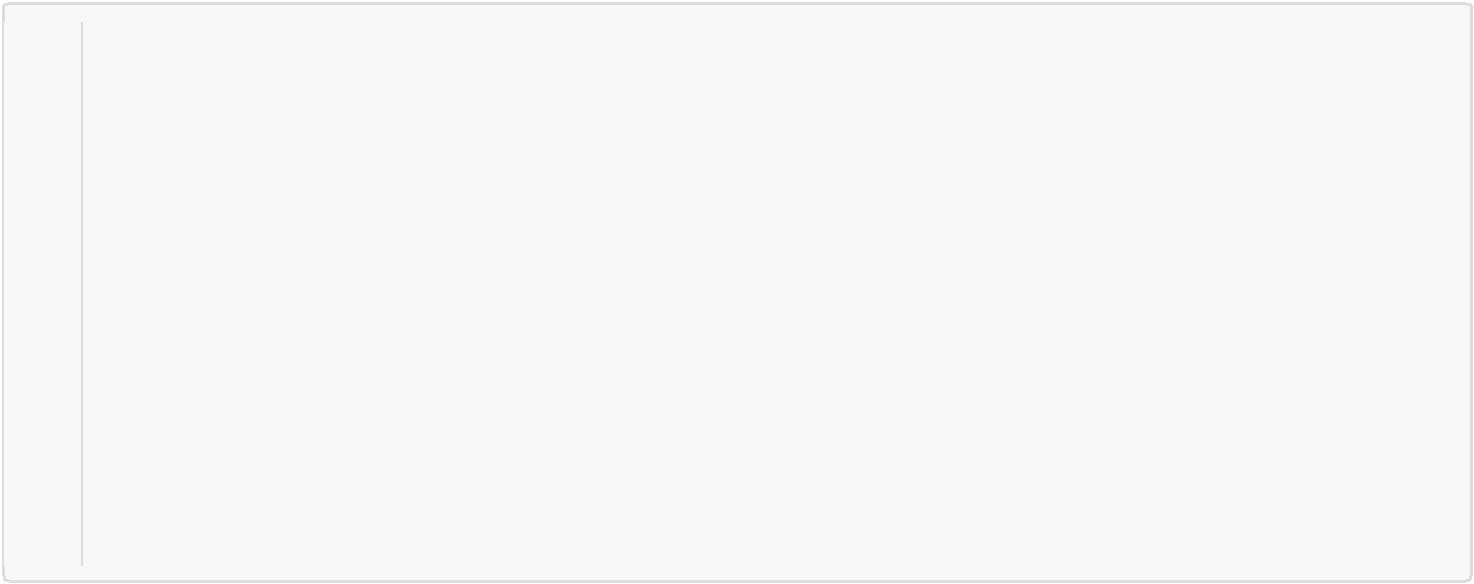
SpringBoot要集成SpringMVC进行Controller的开发，所以项目要导入web的启动依赖

* <dependencies>
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-web</artifactId>
* </dependency>

6 </dependencies>

2.1.3 编写SpringBoot引导类

要通过SpringBoot提供的引导类起步SpringBoot才可以进行访问



* package com.itheima;

2

3 import org.springframework.boot.SpringApplication;

4 import org.springframework.boot.autoconfigure.SpringBootApplication; 5

6 @SpringBootApplication

7 public class MySpringBootApplication {

8

9public static void main(String[] args) {

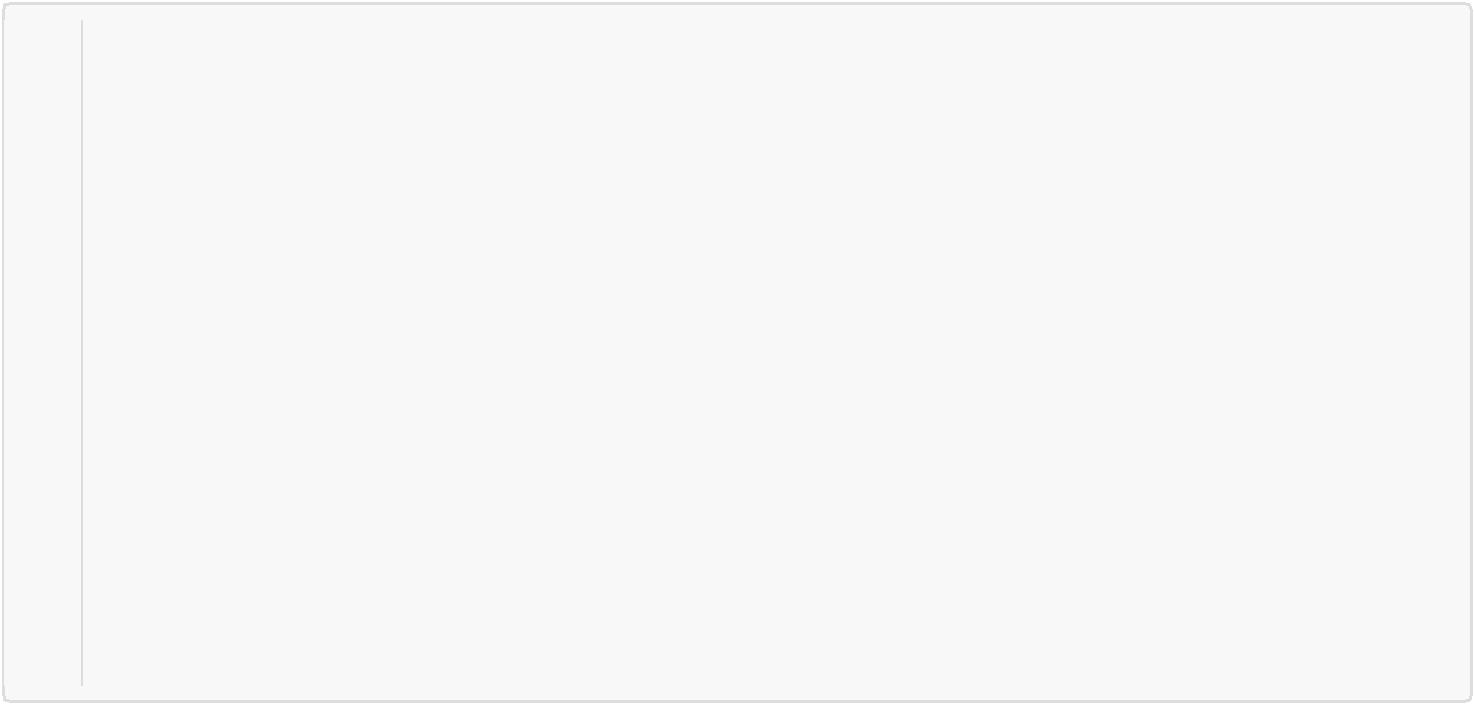
1. SpringApplication.run(MySpringBootApplication.class);
2. }

12

1. }

2.1.4 编写Controller

在引导类MySpringBootApplication同级包或者子级包中创建QuickStartController



* package com.itheima.controller;

2

3 import org.springframework.stereotype.Controller;

4 import org.springframework.web.bind.annotation.RequestMapping; 5 import org.springframework.web.bind.annotation.ResponseBody; 6

7 @Controller

8 public class QuickStartController {

9

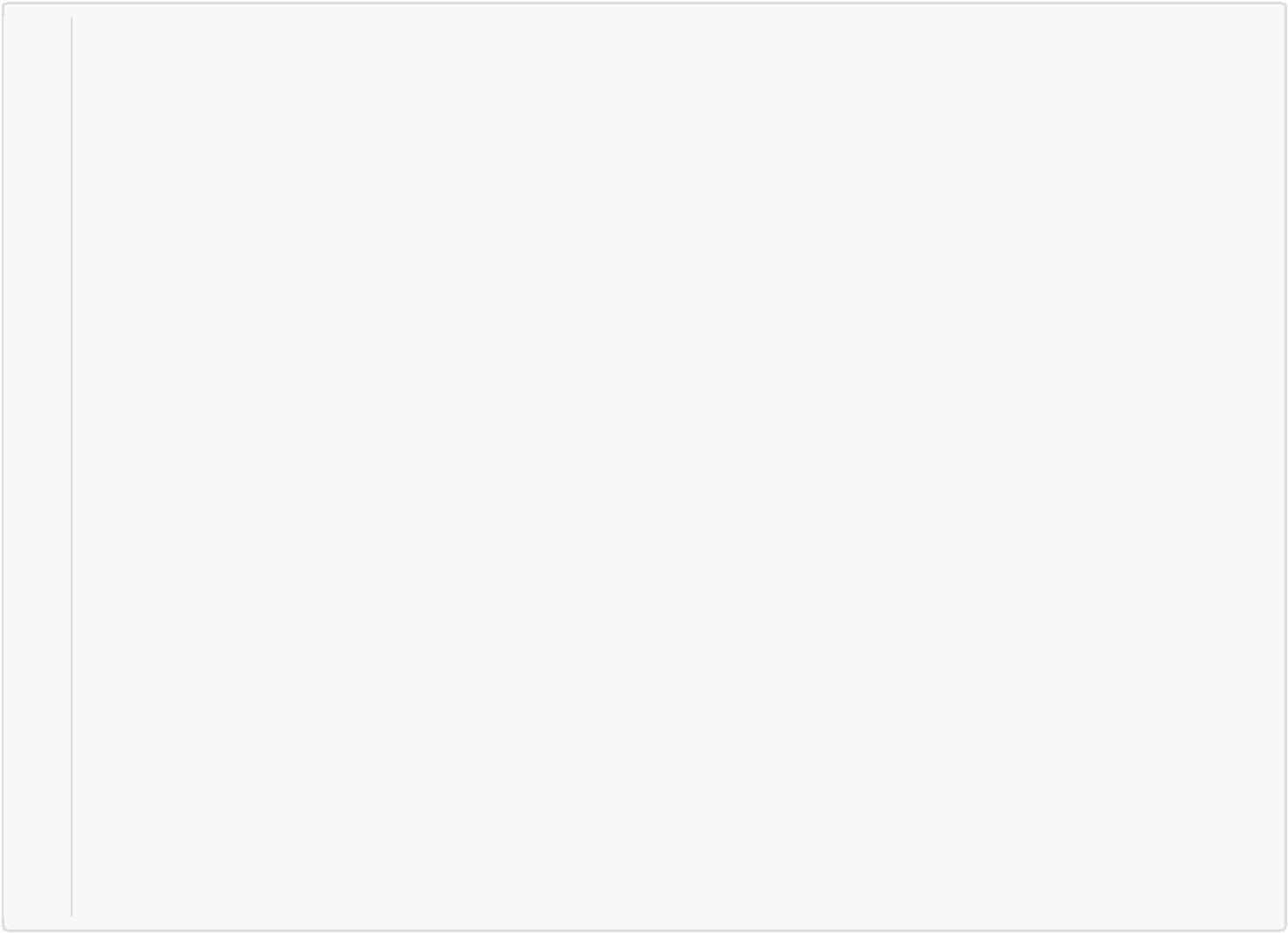
1. @RequestMapping("/quick")
2. @ResponseBody
3. public String quick(){
4. return "springboot 访问成功!";
5. }

15

1. }

2.1.5 测试

执行SpringBoot起步类的主方法，控制台打印日志如下：



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | . | \_\_\_\_ |  | \_ |  | \_\_ \_ | \_ |
| 2 |  | /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ | | | \_\_ \_ \ \ \ \ | | |
| 3 | ( | ( )\\_\_\_ | | '\_ | | '\_| | '\_ \/ | \_` | | \ \ \ \ | |
| 4 |  | \\/ \_\_\_)| | |\_)| | | | | | || | (\_| | | | ) ) ) ) |

* ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /
* =========|\_|==============|\_\_\_/=/\_/\_/\_/

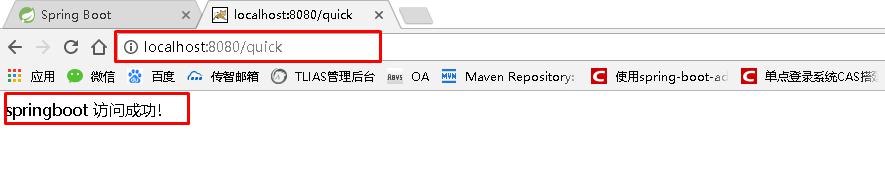
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | :: Spring | Boot :: | (v2.0.1.RELEASE) | |
| 8 |  |  |  |  |
| 9 | 2018-05-08 | 14:29:59.714 | INFO 5672 --- [ | main] |
|  | com.itheima.MySpringBootApplication | | | : Starting MySpringBootApplication on |
|  | DESKTOP-RRUNFUH with PID 5672 | | |  |
|  | (C:\Users\muzimoo\IdeaProjects\IdeaTest\springboot\_quick\target\classes started by | | | |
|  | muzimoo in | C:\Users\muzimoo\IdeaProjects\IdeaTest) | | |

1. ... ... ...
2. o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/\*\*] onto handler of type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler] | | | | |
| 12 | 2018-05-08 14:30:03.126 | INFO 5672 | --- | [ | main] |
|  | o.s.j.e.a.AnnotationMBeanExporter | |  |  | : Registering beans for JMX exposure on |
|  | startup |  |  |  |  |
| 13 | 2018-05-08 14:30:03.196 | INFO 5672 | --- | [ | main] |
|  | o.s.b.w.embedded.tomcat.TomcatWebServer | | |  | : Tomcat started on port(s): 8080 (http) |
|  | with context path '' |  |  |  |  |
| 14 | 2018-05-08 14:30:03.206 | INFO 5672 | --- | [ | main] |
|  | com.itheima.MySpringBootApplication | |  |  | : Started MySpringBootApplication in 4.252 |
|  | seconds (JVM running for | 5.583) |  |  |  |

通过日志发现，Tomcat started on port(s): 8080 (http) with context path '' tomcat已经起步，端口监听8080，web应用的虚拟工程名称为空

打开浏览器访问url地址为：<http://localhost:8080/quick>



2.2 快速入门解析

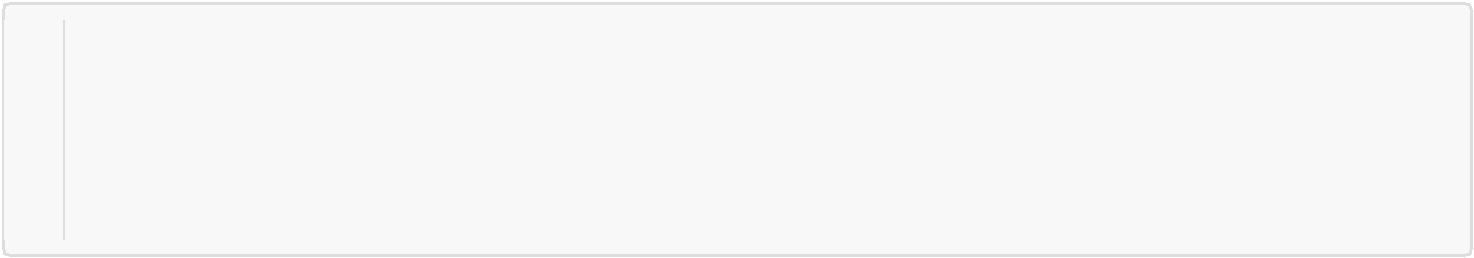


2.2.2 SpringBoot代码解析

 @SpringBootApplication：标注SpringBoot的启动类，该注解具备多种功能（后面详细剖析）  SpringApplication.run(MySpringBootApplication.class) 代表运行SpringBoot的启动类，参数为SpringBoot启动类的字节码对象

2.2.3 SpringBoot工程热部署

我们在开发中反复修改类、页面等资源，每次修改后都是需要重新启动才生效，这样每次启动都很麻烦，浪费了大量的时间，我们可以在修改代码后不重启就能生效，在 pom.xml 中添加如下配置就可以实现这样的功能，我们称之为热部署。



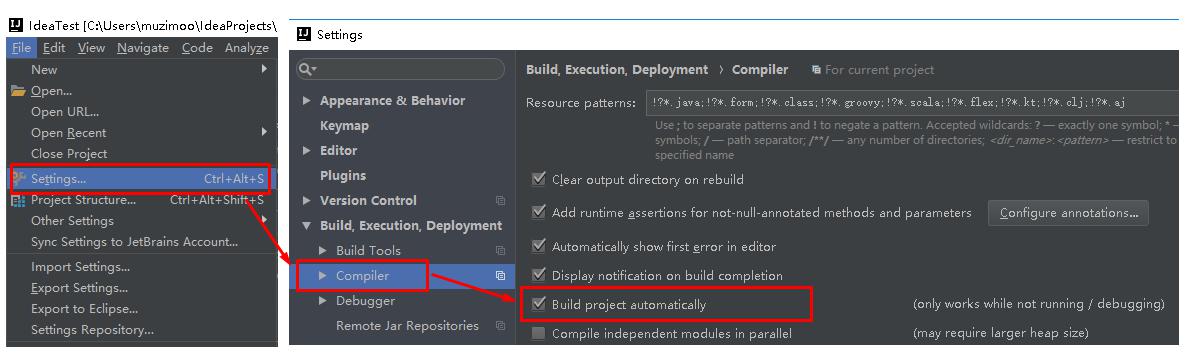
* <!--热部署配置-->

2 <dependency>

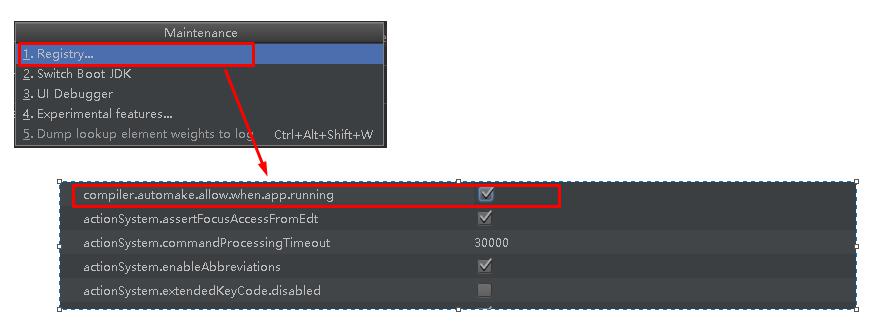
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-devtools</artifactId> 5 </dependency>

注意：IDEA进行SpringBoot热部署失败原因

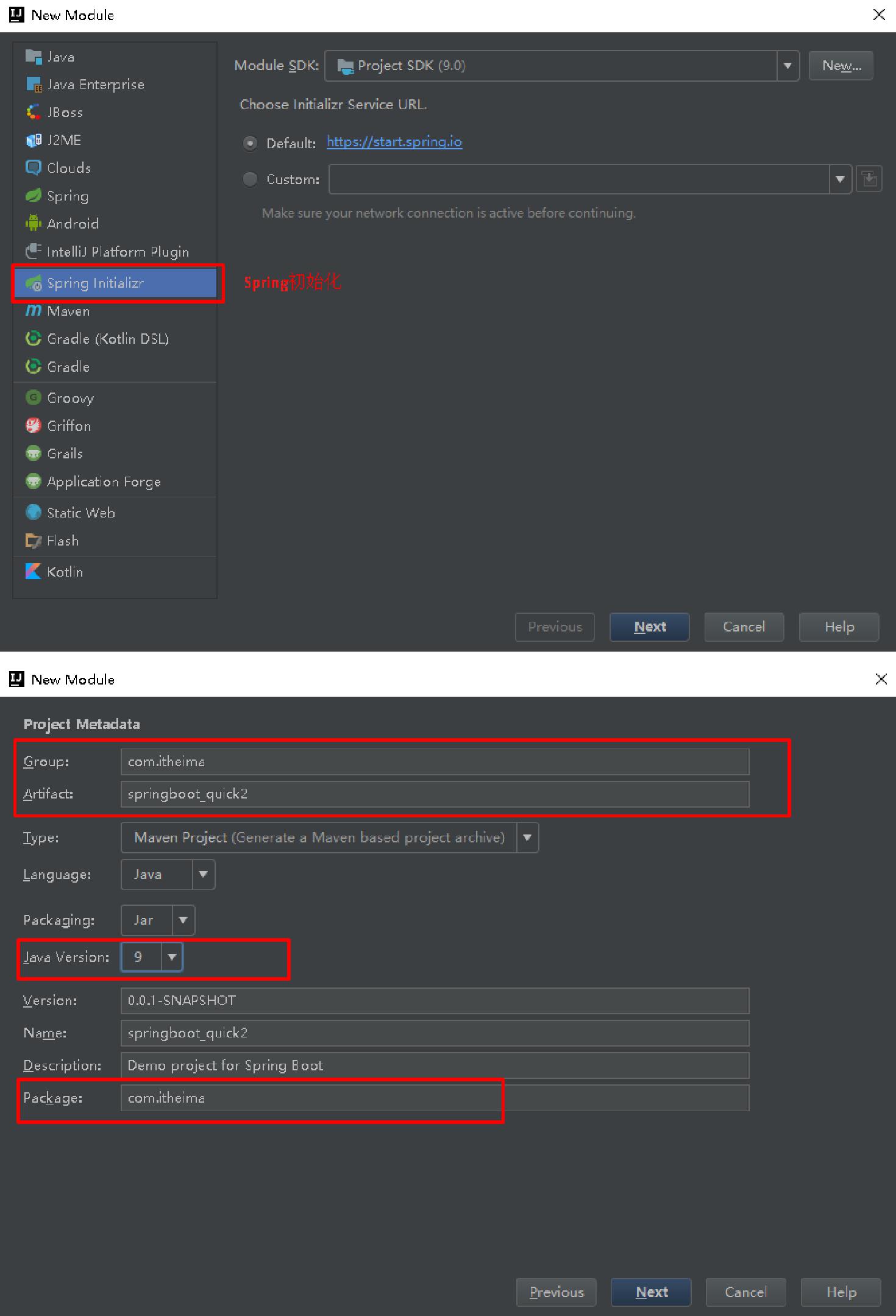
出现这种情况，并不是热部署配置问题，其根本原因是因为Intellij IEDA默认情况下不会自动编译，需要对IDEA进行自动编译的设置，如下：

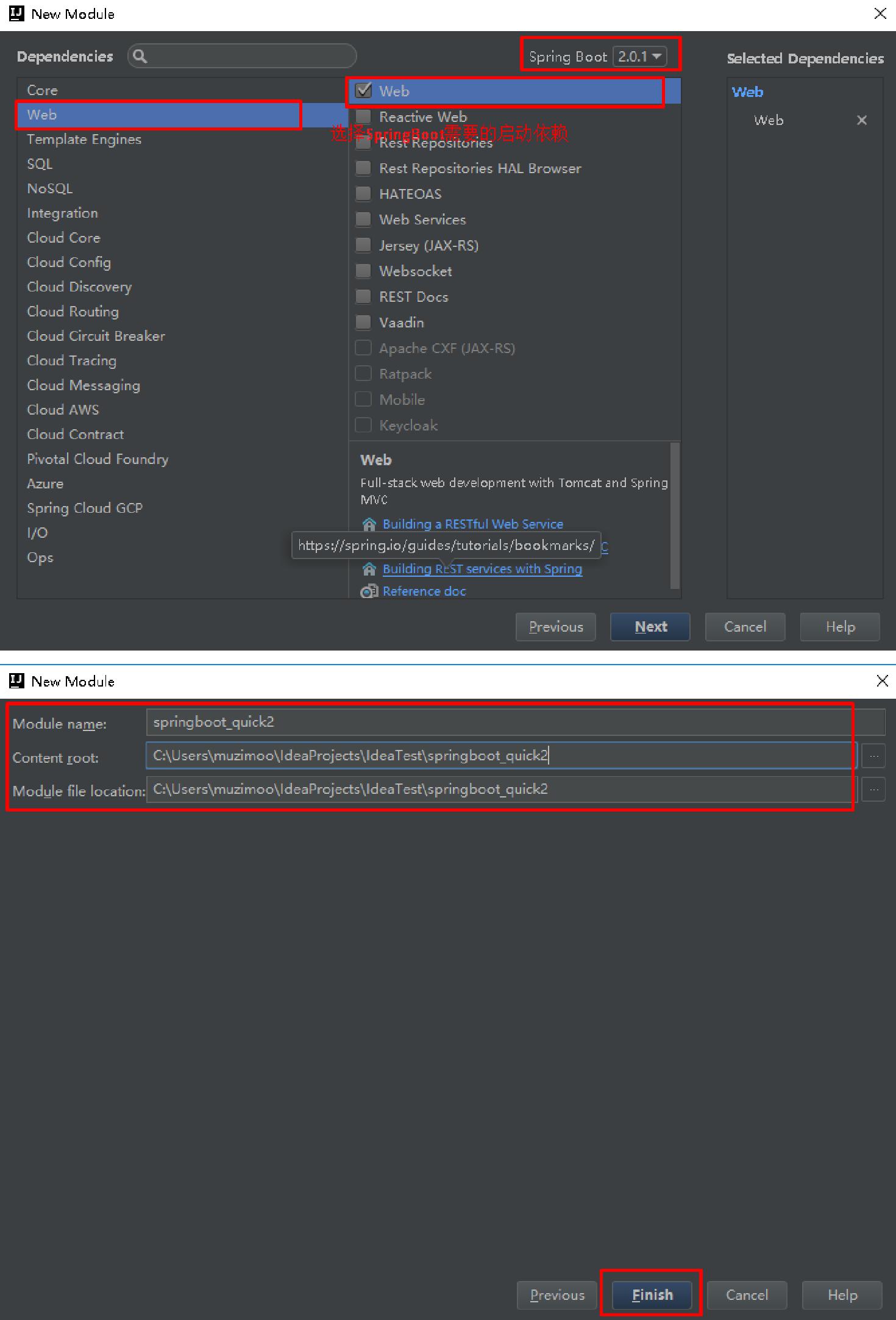


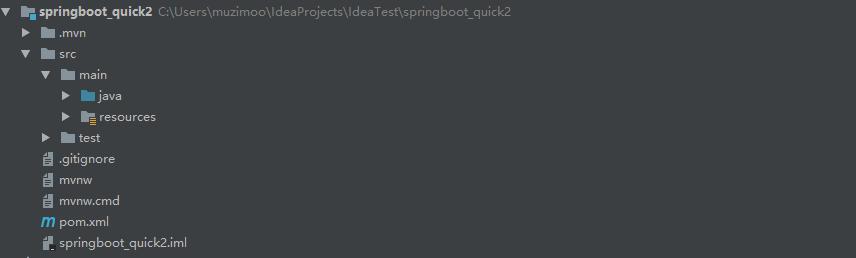
然后 Shift+Ctrl+Alt+/，选择Registry



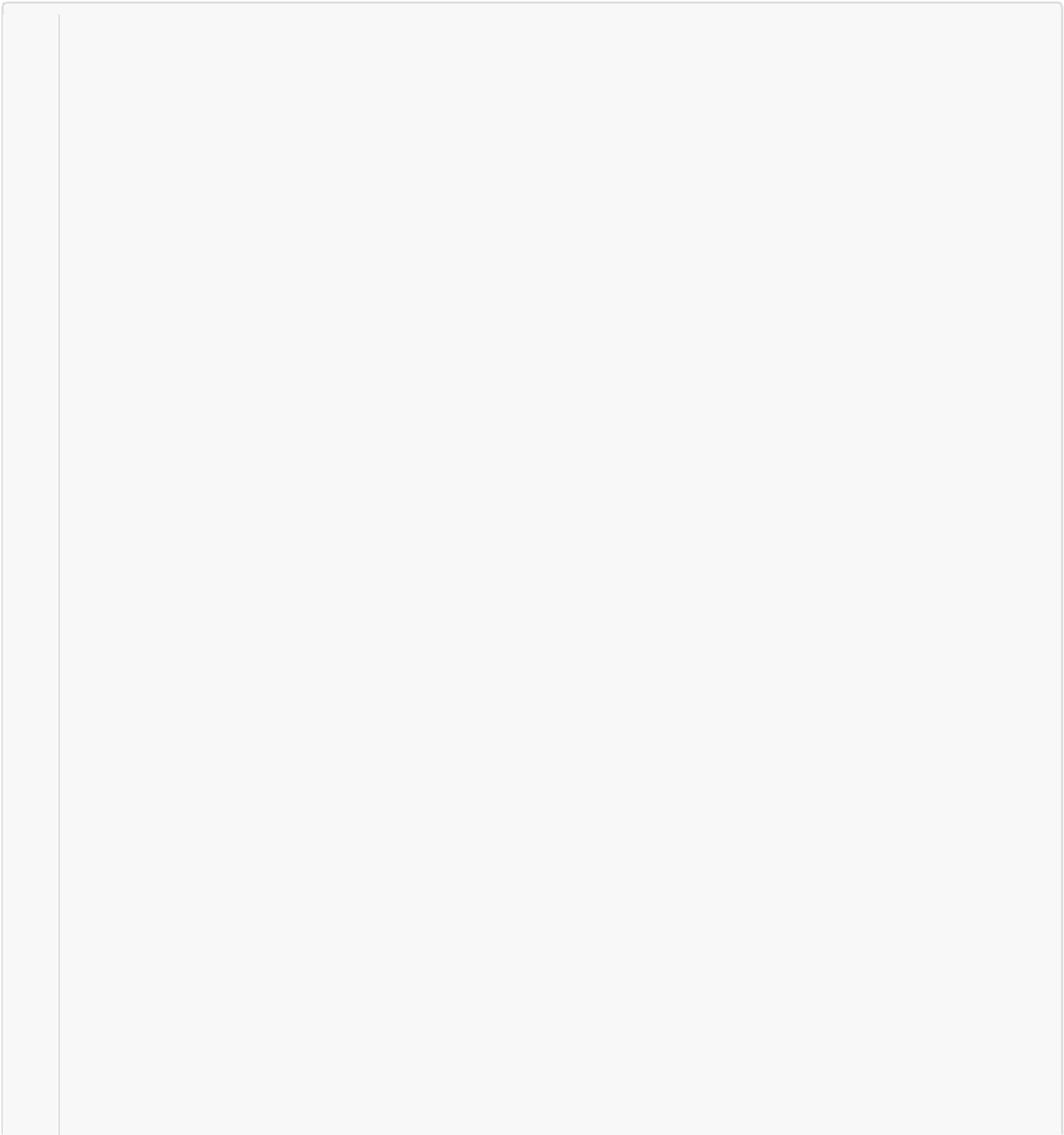
2.2.4 使用idea快速创建SpringBoot项目







通过idea快速创建的SpringBoot项目的pom.xml中已经导入了我们选择的web的起步依赖的坐标



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.itheima</groupId>

<artifactId>springboot\_quick2</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>springboot\_quick2</name>

<description>Demo project for Spring Boot</description>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.1.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>9</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

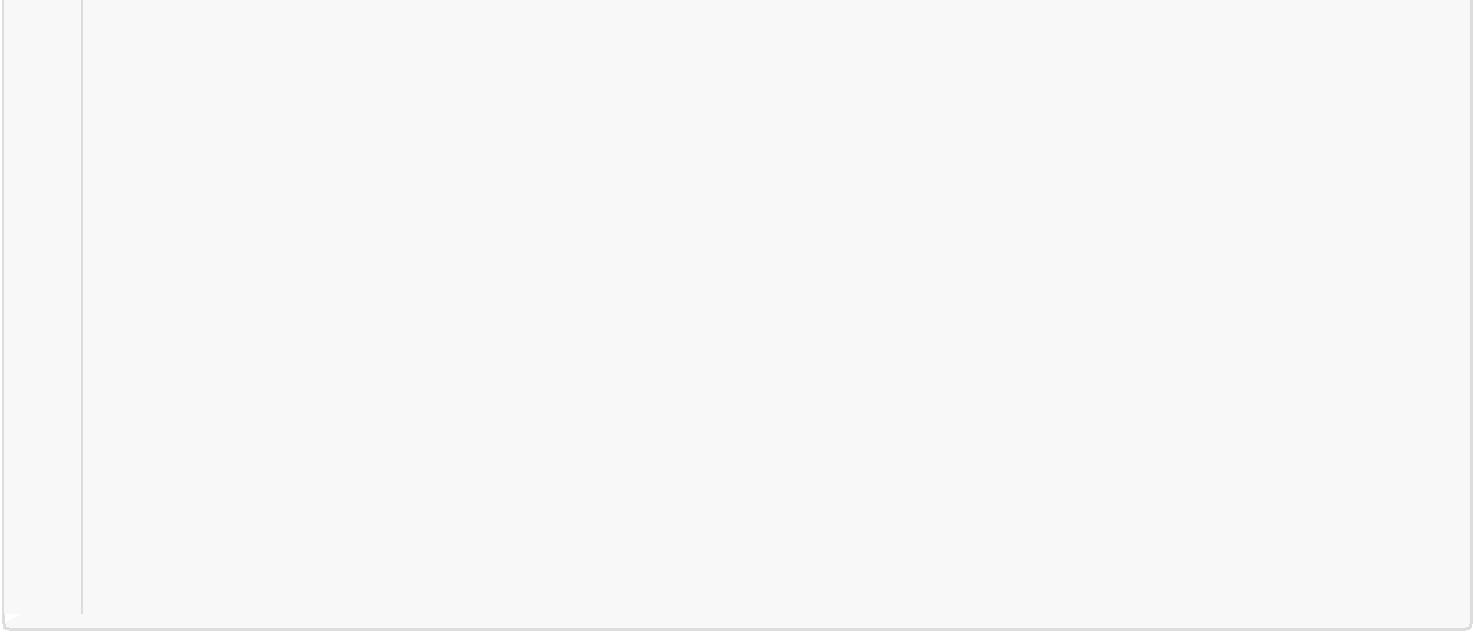
</dependency>

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

1. </dependency>
2. </dependencies>
3. <build>
4. <plugins>

|  |  |
| --- | --- |
| 42 | <plugin> |
| 43 | <groupId>org.springframework.boot</groupId> |
| 44 | <artifactId>spring-boot-maven-plugin</artifactId> |
| 45 | </plugin> |

1. </plugins>
2. </build>

48

49

1. </project>

可以使用快速入门的方式创建Controller进行访问，此处不再赘述

三、SpringBoot原理分析

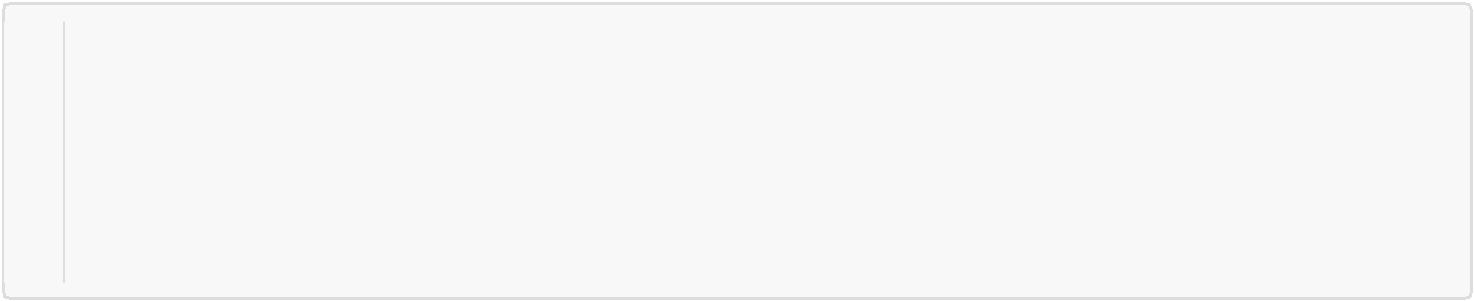


3.1 起步依赖原理分析



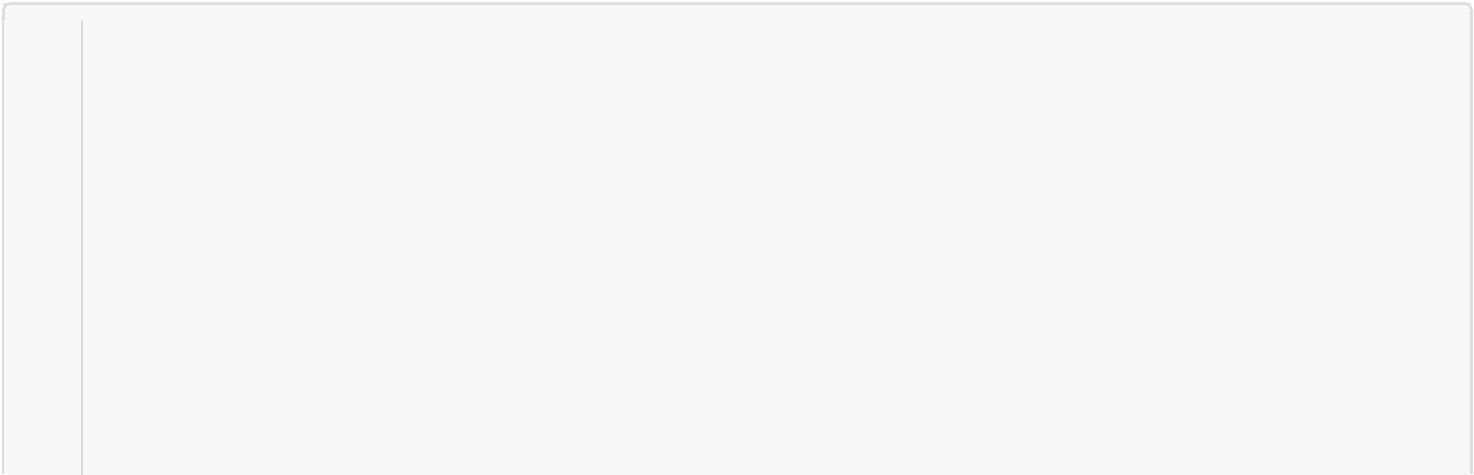
3.1.1 分析spring-boot-starter-parent

按住Ctrl点击pom.xml中的spring-boot-starter-parent，跳转到了spring-boot-starter-parent的pom.xml，xml配置如下（只摘抄了部分重点配置）：



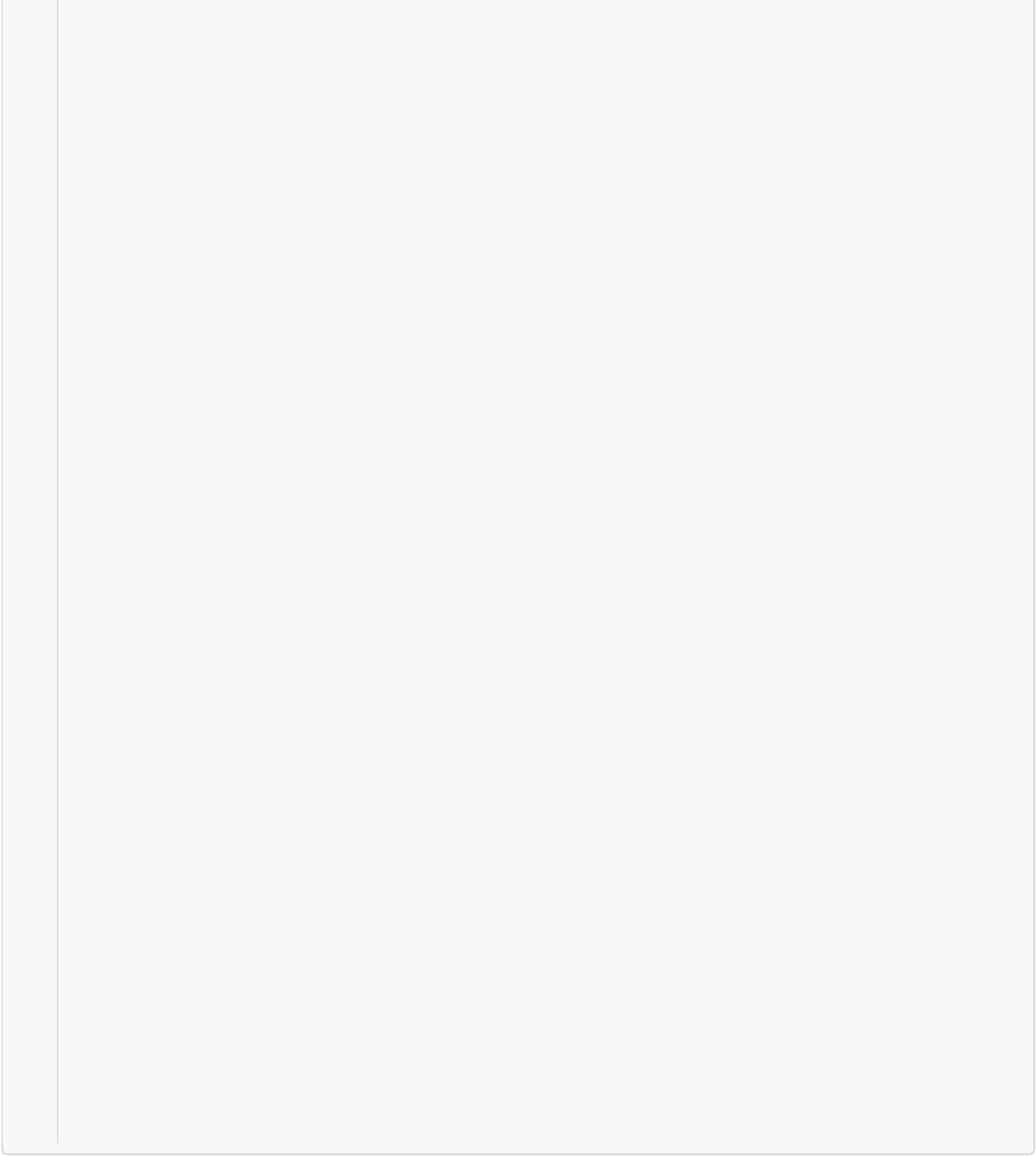
* <parent>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-dependencies</artifactId>
* <version>2.0.1.RELEASE</version>
* <relativePath>../../spring-boot-dependencies</relativePath> 6 </parent>

按住Ctrl点击pom.xml中的spring-boot-starter-dependencies，跳转到了spring-boot-starter-dependencies的pom.xml，xml配置如下（只摘抄了部分重点配置）：



* <properties>
* <activemq.version>5.15.3</activemq.version>
* <antlr2.version>2.7.7</antlr2.version>
* <appengine-sdk.version>1.9.63</appengine-sdk.version>
* <artemis.version>2.4.0</artemis.version>
* <aspectj.version>1.8.13</aspectj.version>
* <assertj.version>3.9.1</assertj.version>
* <atomikos.version>4.0.6</atomikos.version>

9<bitronix.version>2.1.4</bitronix.version>

1. <build-helper-maven-plugin.version>3.0.0</build-helper-maven-plugin.version>
2. <byte-buddy.version>1.7.11</byte-buddy.version>
3. ... ... ...
4. </properties>
5. <dependencyManagement>
6. <dependencies>
7. <dependency>

|  |  |
| --- | --- |
| 17 | <groupId>org.springframework.boot</groupId> |
| 18 | <artifactId>spring-boot</artifactId> |
| 19 | <version>2.0.1.RELEASE</version> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 22 | <groupId>org.springframework.boot</groupId> |
| 23 | <artifactId>spring-boot-test</artifactId> |
| 24 | <version>2.0.1.RELEASE</version> |

1. </dependency>
2. ... ... ...
3. </dependencies>
4. </dependencyManagement>
5. <build>
6. <pluginManagement>
7. <plugins>

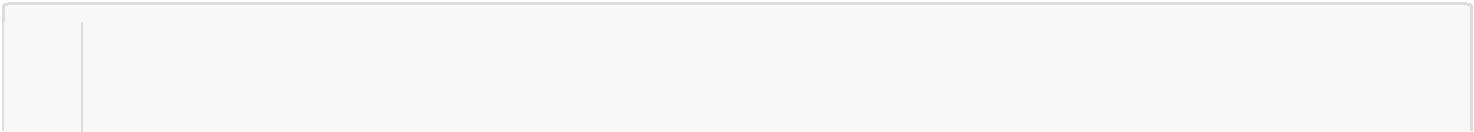
|  |  |
| --- | --- |
| 32 | <plugin> |
| 33 | <groupId>org.jetbrains.kotlin</groupId> |
| 34 | <artifactId>kotlin-maven-plugin</artifactId> |
| 35 | <version>${kotlin.version}</version> |
| 36 | </plugin> |
| 37 | <plugin> |
| 38 | <groupId>org.jooq</groupId> |
| 39 | <artifactId>jooq-codegen-maven</artifactId> |
| 40 | <version>${jooq.version}</version> |
| 41 | </plugin> |
| 42 | <plugin> |
| 43 | <groupId>org.springframework.boot</groupId> |
| 44 | <artifactId>spring-boot-maven-plugin</artifactId> |
| 45 | <version>2.0.1.RELEASE</version> |
| 46 | </plugin> |
| 47 | ... ... ... |

1. </plugins>
2. </pluginManagement>
3. </build>

从上面的spring-boot-starter-dependencies的pom.xml中我们可以发现，一部分坐标的版本、依赖管理、插件管理已经定义好，所以我们的SpringBoot工程继承spring-boot-starter-parent后已经具备版本锁定等配置了。所以起步依赖的作用就是进行依赖的传递。

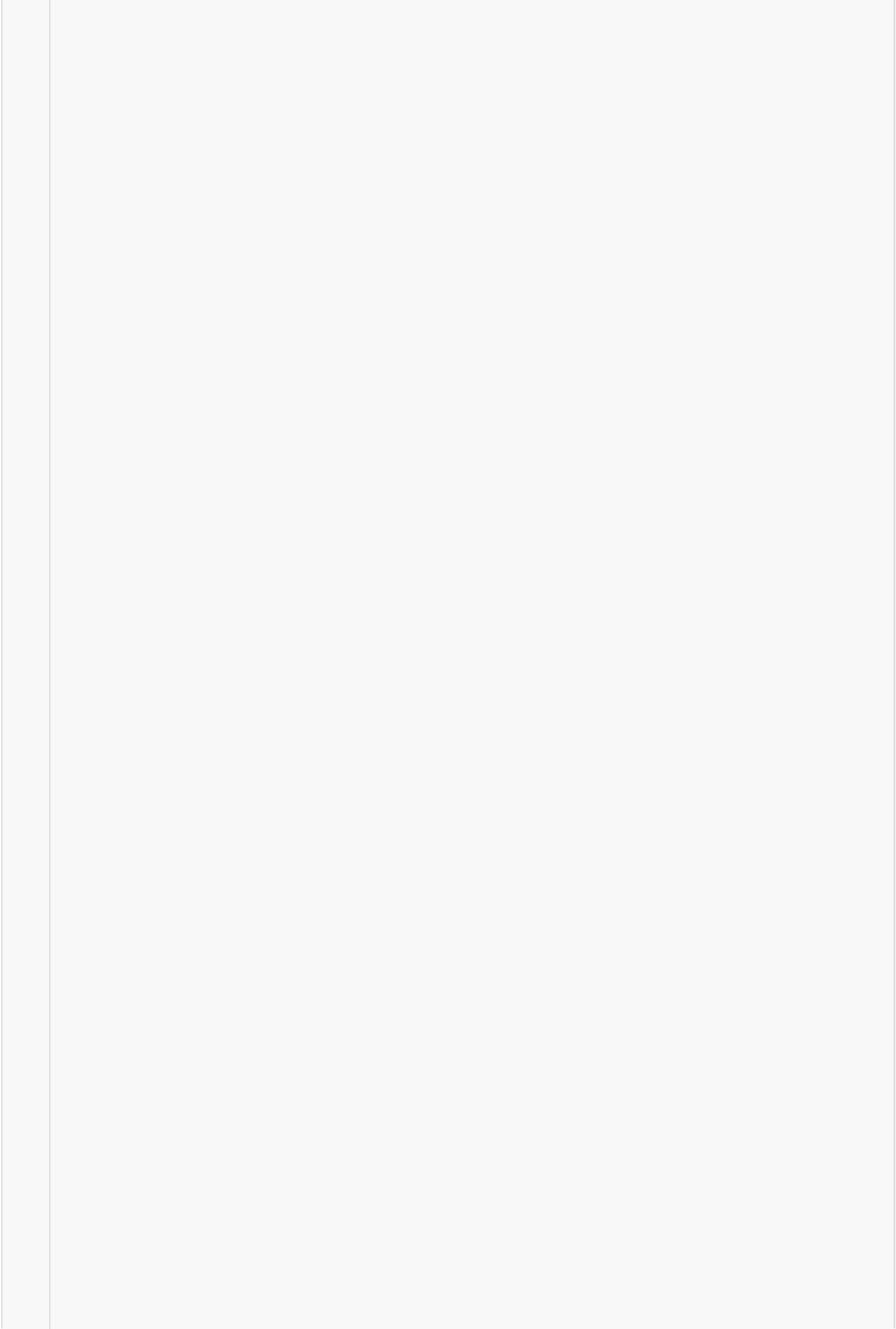
3.1.2 分析spring-boot-starter-web

按住Ctrl点击pom.xml中的spring-boot-starter-web，跳转到了spring-boot-starter-web的pom.xml，xml配置如下（只摘抄了部分重点配置）：



* <?xml version="1.0" encoding="UTF-8"?>

2 <project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd"

xmlns="http://maven.apache.org/POM/4.0.0"

* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
* <modelVersion>4.0.0</modelVersion>
* <parent>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starters</artifactId>
* <version>2.0.1.RELEASE</version>
* </parent>

1. <groupId>org.springframework.boot</groupId>
2. <artifactId>spring-boot-starter-web</artifactId>
3. <version>2.0.1.RELEASE</version>
4. <name>Spring Boot Web Starter</name>

14

1. <dependencies>
2. <dependency>

|  |  |
| --- | --- |
| 17 | <groupId>org.springframework.boot</groupId> |
| 18 | <artifactId>spring-boot-starter</artifactId> |
| 19 | <version>2.0.1.RELEASE</version> |
| 20 | <scope>compile</scope> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 23 | <groupId>org.springframework.boot</groupId> |
| 24 | <artifactId>spring-boot-starter-json</artifactId> |
| 25 | <version>2.0.1.RELEASE</version> |
| 26 | <scope>compile</scope> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 29 | <groupId>org.springframework.boot</groupId> |
| 30 | <artifactId>spring-boot-starter-tomcat</artifactId> |
| 31 | <version>2.0.1.RELEASE</version> |
| 32 | <scope>compile</scope> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 35 | <groupId>org.hibernate.validator</groupId> |
| 36 | <artifactId>hibernate-validator</artifactId> |
| 37 | <version>6.0.9.Final</version> |
| 38 | <scope>compile</scope> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 41 | <groupId>org.springframework</groupId> |
| 42 | <artifactId>spring-web</artifactId> |
| 43 | <version>5.0.5.RELEASE</version> |
| 44 | <scope>compile</scope> |

1. </dependency>
2. <dependency>

|  |  |
| --- | --- |
| 47 | <groupId>org.springframework</groupId> |
| 48 | <artifactId>spring-webmvc</artifactId> |
| 49 | <version>5.0.5.RELEASE</version> |
| 50 | <scope>compile</scope> |

1. </dependency>
2. </dependencies>
3. </project>

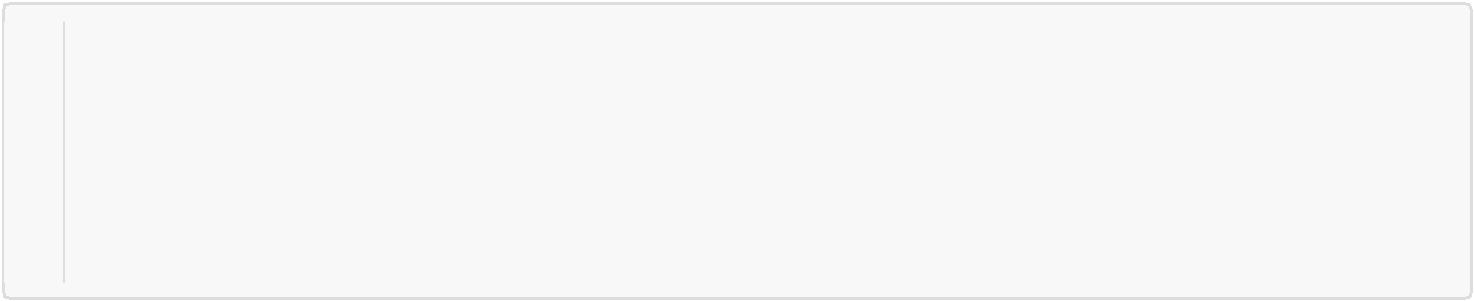
54

从上面的spring-boot-starter-web的pom.xml中我们可以发现，spring-boot-starter-web就是将web开发要使用的spring-web、spring-webmvc等坐标进行了“打包”，这样我们的工程只要引入spring-boot-starter-web起步依赖的坐标就可以进行web开发了，同样体现了依赖传递的作用。

3.2 自动配置原理解析



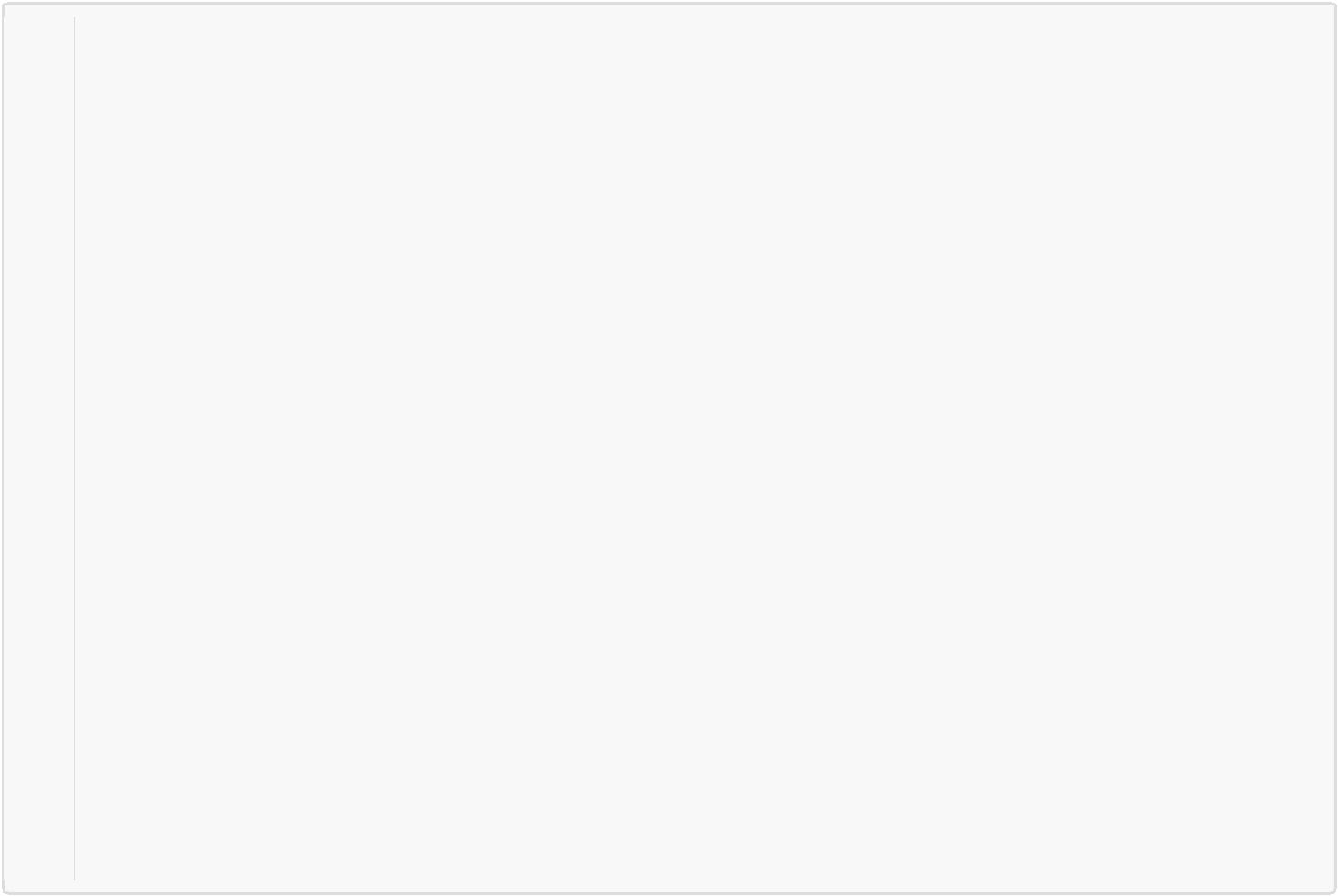
按住Ctrl点击查看启动类MySpringBootApplication上的注解@SpringBootApplication



* @SpringBootApplication
* public class MySpringBootApplication {
* public static void main(String[] args) {
* SpringApplication.run(MySpringBootApplication.class);
* }

6 }

注解@SpringBootApplication的源码



* @Target(ElementType.TYPE)
* @Retention(RetentionPolicy.RUNTIME)

3 @Documented

4 @Inherited

5 @SpringBootConfiguration

6 @EnableAutoConfiguration

7 @ComponentScan(excludeFilters = {

8@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),

9@Filter(type = FilterType.CUSTOM, classes =

1. public @interface SpringBootApplication {

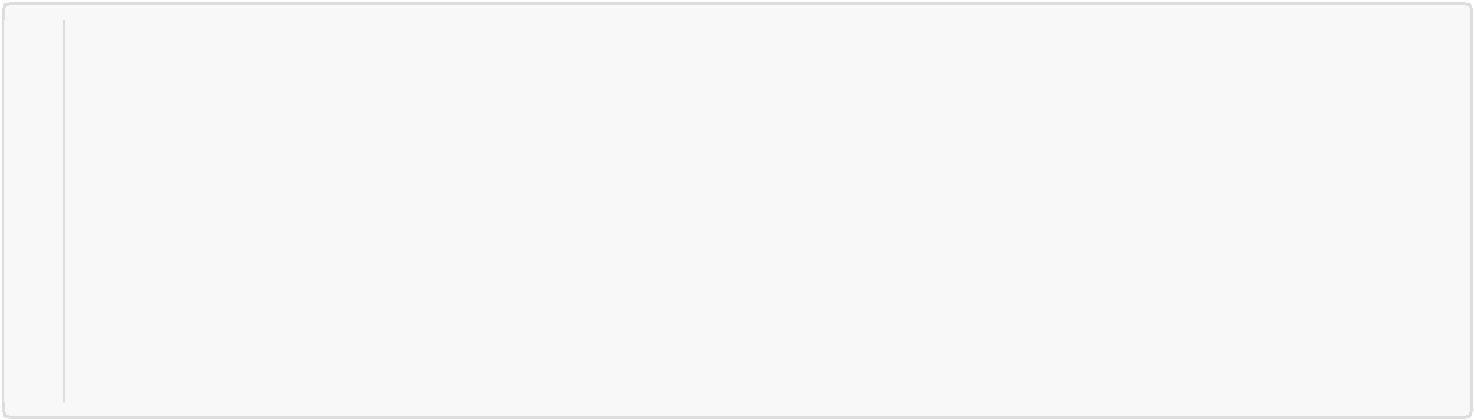
|  |  |
| --- | --- |
| 12 | /\*\* |
| 13 | \* Exclude specific auto-configuration classes such that they will never be |
|  | applied. |
| 14 | \* @return the classes to exclude |
| 15 | \*/ |
| 16 | @AliasFor(annotation = EnableAutoConfiguration.class) |
| 17 | Class<?>[] exclude() default {}; |
| 18 |  |
| 19 | ... ... ... |
| 20 |  |
| 21 | } |

其中，

@SpringBootConfiguration：等同与@Configuration，既标注该类是Spring的一个配置类

@EnableAutoConfiguration：SpringBoot自动配置功能开启

按住Ctrl点击查看注解@EnableAutoConfiguration



* @Target(ElementType.TYPE)
* @Retention(RetentionPolicy.RUNTIME)

3 @Documented

4 @Inherited

5 @AutoConfigurationPackage

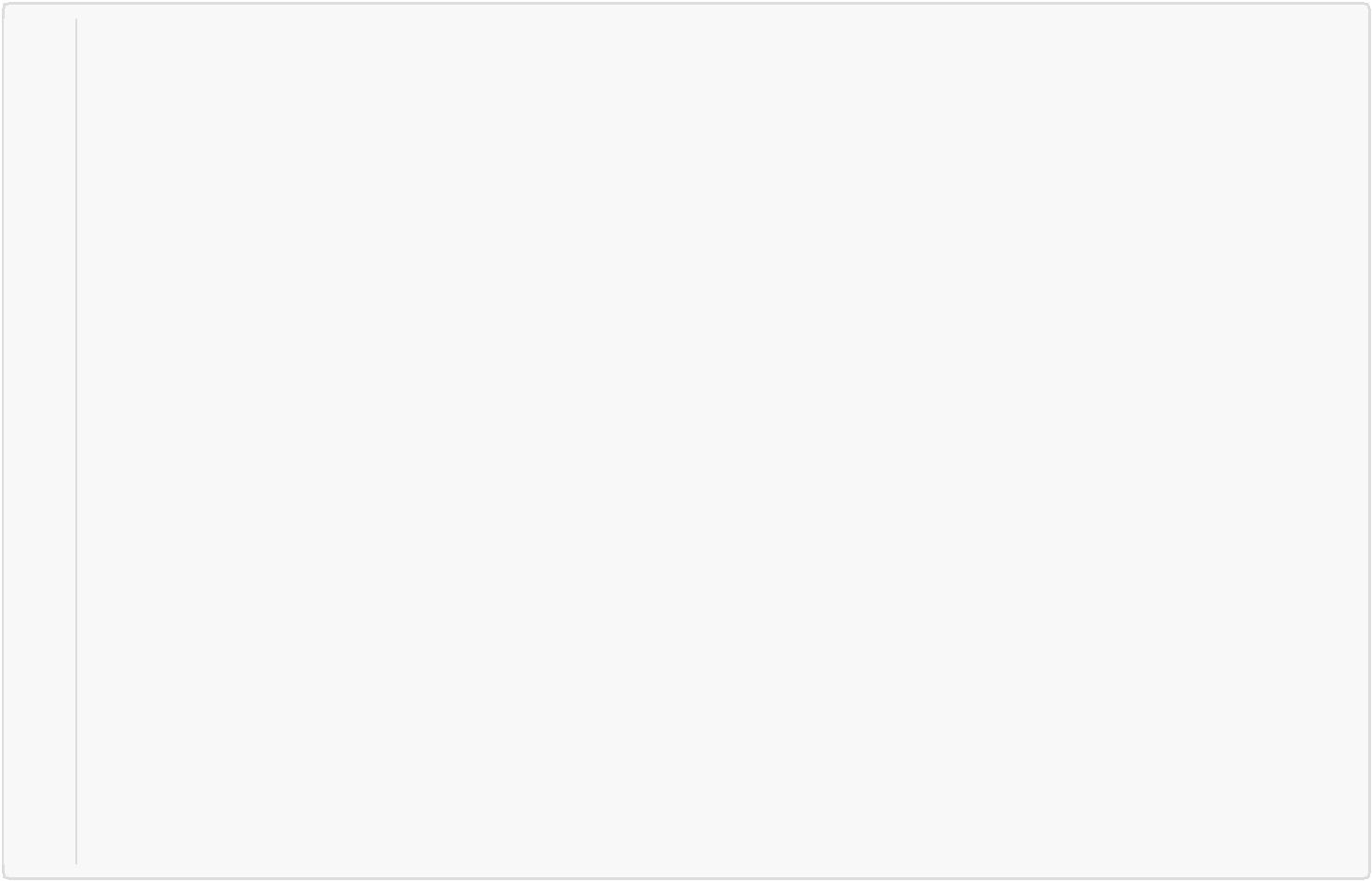
6 @Import(AutoConfigurationImportSelector.class)

7 public @interface EnableAutoConfiguration {

8... ... ...

9 }

其中，@Import(AutoConfigurationImportSelector.class) 导入了AutoConfigurationImportSelector类按住Ctrl点击查看AutoConfigurationImportSelector源码



* public String[] selectImports(AnnotationMetadata annotationMetadata) {
* ... ... ...
* List<String> configurations = getCandidateConfigurations(annotationMetadata,

|  |  |
| --- | --- |
| 4 | attributes); |

* configurations = removeDuplicates(configurations);
* Set<String> exclusions = getExclusions(annotationMetadata, attributes);

7checkExcludedClasses(configurations, exclusions);

* configurations.removeAll(exclusions);
* configurations = filter(configurations, autoConfigurationMetadata);

10fireAutoConfigurationImportEvents(configurations, exclusions);

11return StringUtils.toStringArray(configurations);

12 }

13

14

15 protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,

|  |  |
| --- | --- |
| 16 | AnnotationAttributes attributes) { |

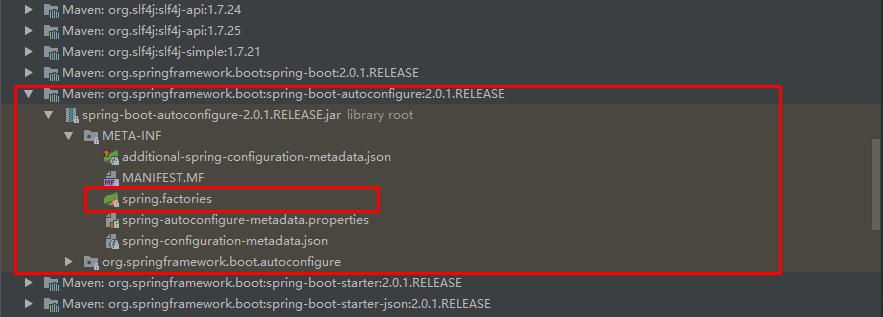
1. List<String> configurations = SpringFactoriesLoader.loadFactoryNames(

|  |  |
| --- | --- |
| 18 | getSpringFactoriesLoaderFactoryClass(), getBeanClassLoader()); |
| 19 |  |

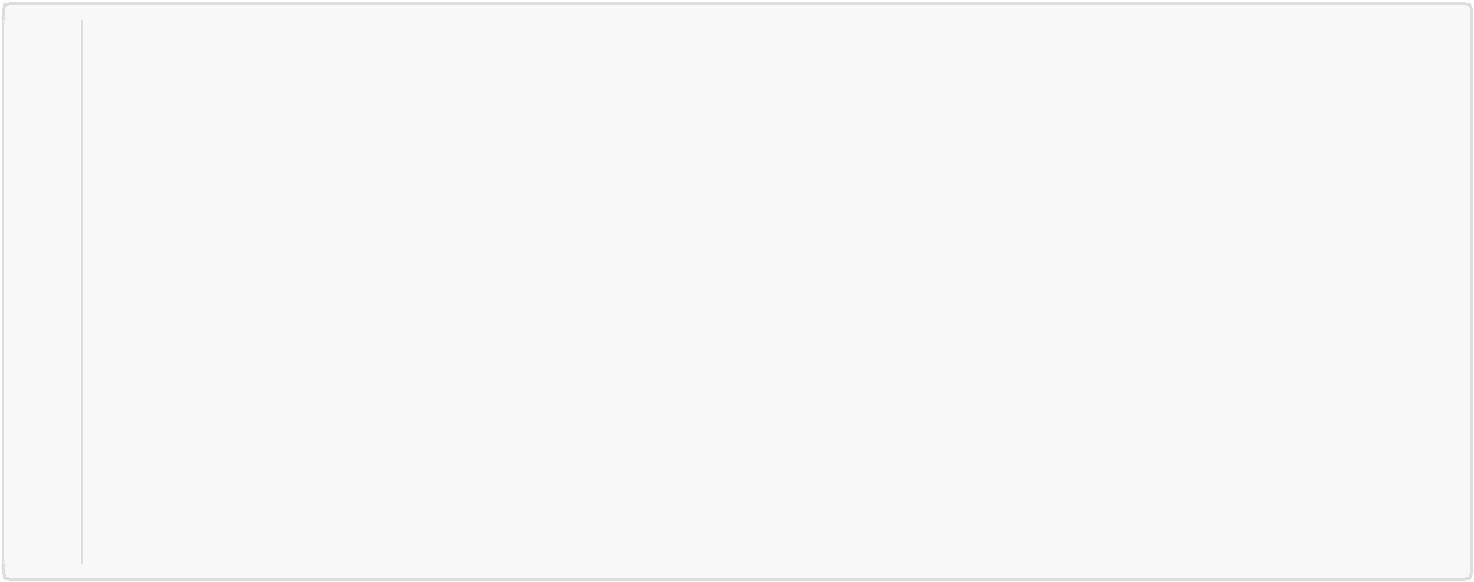
1. return configurations;
2. }

22

其中，SpringFactoriesLoader.loadFactoryNames 方法的作用就是从META-INF/spring.factories文件中读取指定类对应的类名称列表



spring.factories 文件中有关自动配置的配置信息如下：



* ... ... ...

2

3 org.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAutoConf iguration,\

4 org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration ,\

5 org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfigu ration,\

6 org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration,\

7 org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\

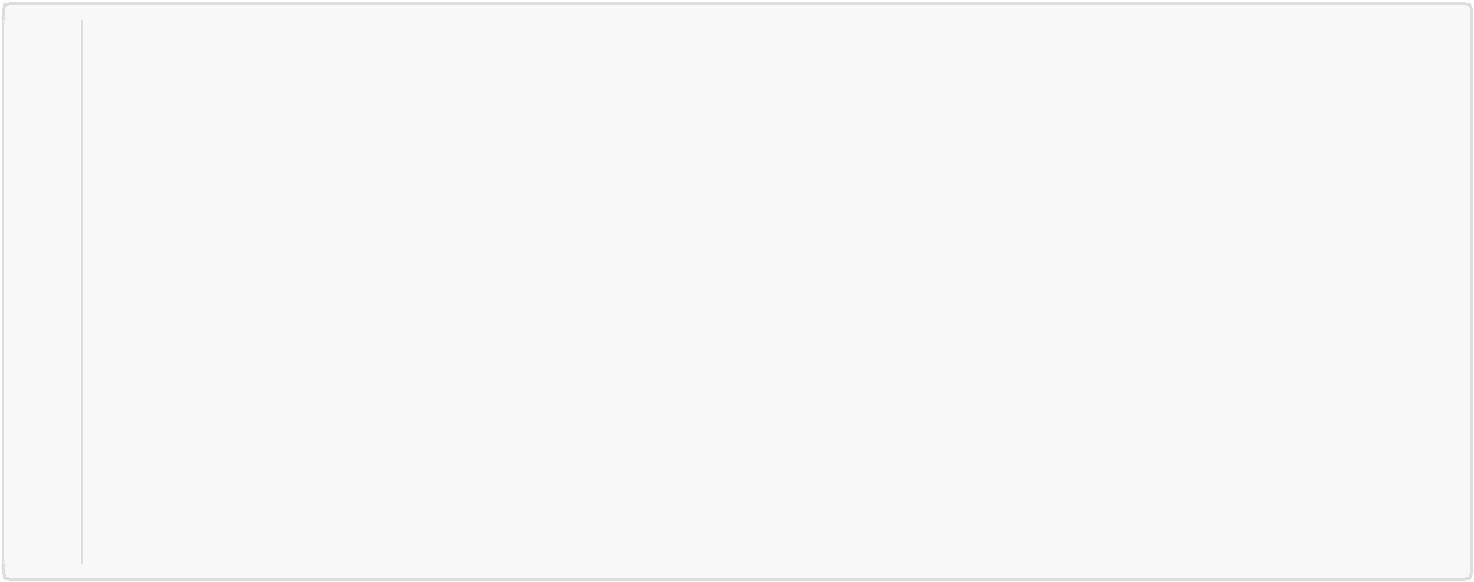
8 org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration,\

9

1. ... ... ...

上面配置文件存在大量的以Configuration为结尾的类名称，这些类就是存有自动配置信息的类，而SpringApplication在获取这些类名后再加载

我们以ServletWebServerFactoryAutoConfiguration为例来分析源码：



* @Configuration
* @AutoConfigureOrder(Ordered.HIGHEST\_PRECEDENCE)

3 @ConditionalOnClass(ServletRequest.class)

4 @ConditionalOnWebApplication(type = Type.SERVLET)

5 @EnableConfigurationProperties(ServerProperties.class)

6 @Import({ ServletWebServerFactoryAutoConfiguration.BeanPostProcessorsRegistrar.class,

* ServletWebServerFactoryConfiguration.EmbeddedTomcat.class,
* ServletWebServerFactoryConfiguration.EmbeddedJetty.class,
* ServletWebServerFactoryConfiguration.EmbeddedUndertow.class }) 10 public class ServletWebServerFactoryAutoConfiguration {

11... ... ...

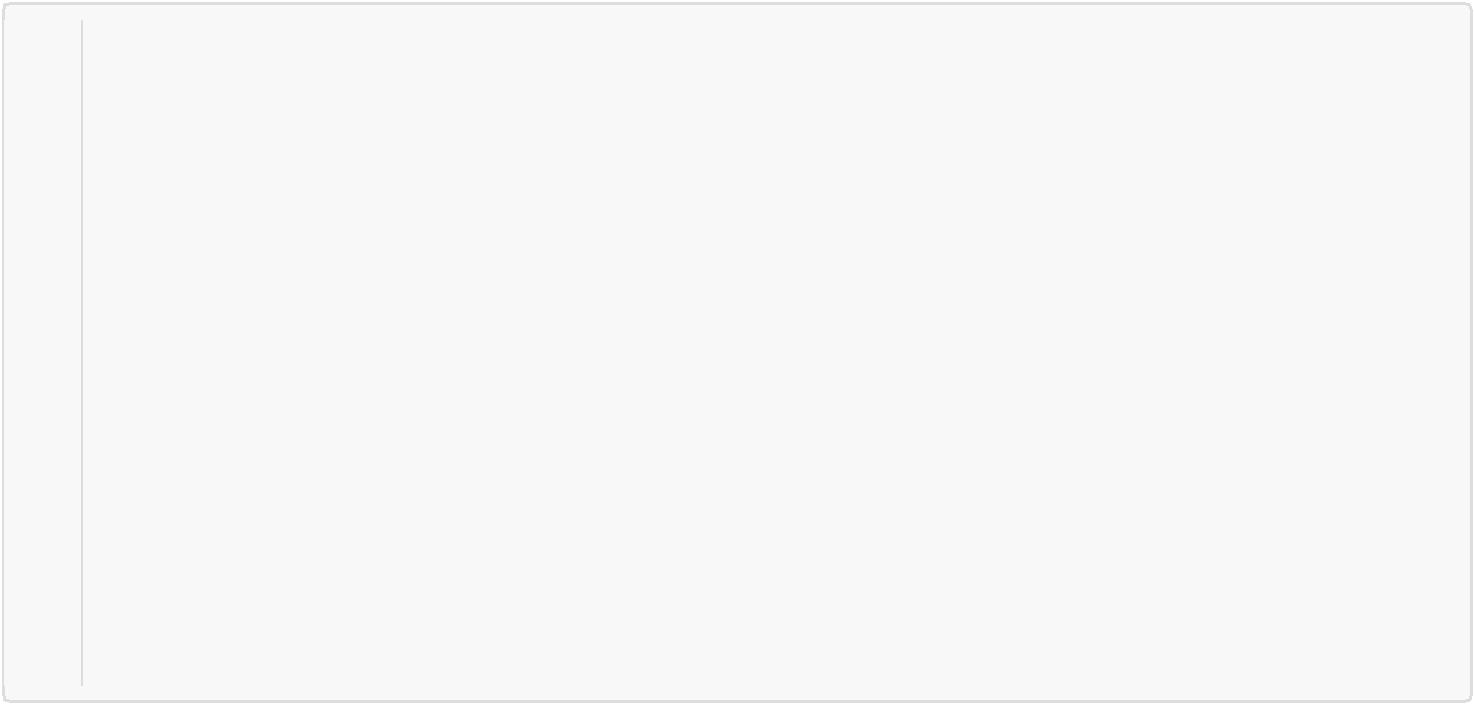
12 }

13

其中，

@EnableConfigurationProperties(ServerProperties.class) 代表加载ServerProperties服务器配置属性类

进入ServerProperties.class源码如下：



* @ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)

2 public class ServerProperties {

3

* /\*\*
* \* Server HTTP port.
* \*/
* private Integer port;

8

* /\*\*

1. \* Network address to which the server should bind.
2. \*/
3. private InetAddress address;

13

1. ... ... ...
2. }

其中，

prefix = "server" 表示SpringBoot配置文件中的前缀，SpringBoot会将配置文件中以server开始的属性映射到该类

的字段中。映射关系如下：



四、SpringBoot的配置文件



4.1 SpringBoot配置文件类型



4.1.1 SpringBoot配置文件类型和作用

SpringBoot是基于约定的，所以很多配置都有默认值，但如果想使用自己的配置替换默认配置的话，就可以使用application.properties或者application.yml（application.yaml）进行配置。

SpringBoot默认会从Resources目录下加载application.properties或application.yml（application.yaml）文件

其中，application.properties文件是键值对类型的文件，之前一直在使用，所以此处不在对properties文件的格式进行阐述。除了properties文件外，SpringBoot还可以使用yml文件进行配置，下面对yml文件进行讲解。

4.1.2 application.yml配置文件

4.1.2.1 yml配置文件简介

YML文件格式是YAML (YAML Aint Markup Language)编写的文件格式，YAML是一种直观的能够被电脑识别的的数据数据序列化格式，并且容易被人类阅读，容易和脚本语言交互的，可以被支持YAML库的不同的编程语言程序导入，比如： C/C++, Ruby, Python, Java, Perl, C#, PHP等。YML文件是以数据为核心的，比传统的xml方式更加简洁。

YML文件的扩展名可以使用.yml或者.yaml。

4.1.2.2 yml配置文件的语法

4.1.2.2.1 配置普通数据

语法： key: value



 示例代码：



1 name: haohao



 注意：value之前有一个空格

4.1.2.2.2 配置对象数据

 语法：

key:

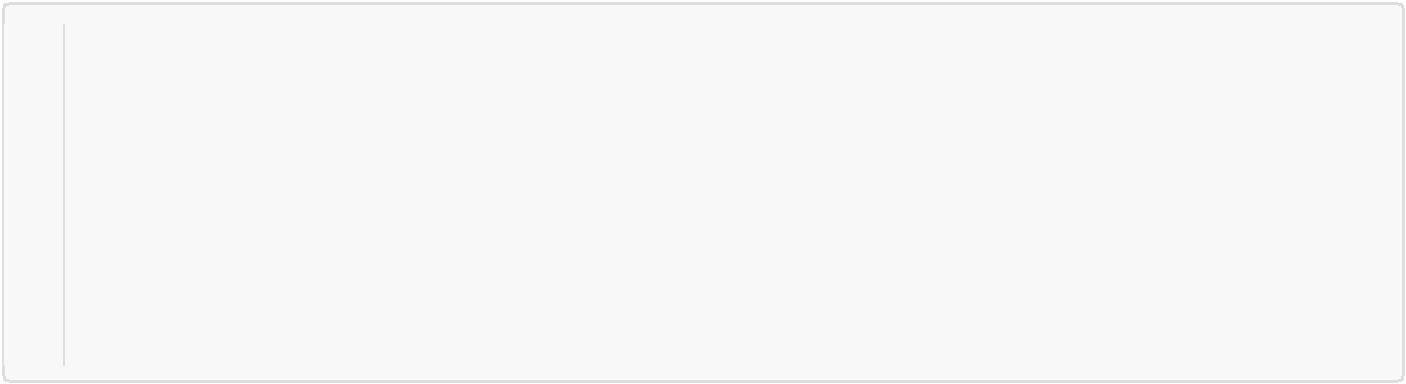
key1: value1

key2: value2

或者：

key: {key1: value1,key2: value2}

 示例代码：



1 person:



* name: haohao

3age: 31

4addr: beijing

5

6 #或者

7

8 person: {name: haohao,age: 31,addr: beijing}

 注意：key1前面的空格个数不限定，在yml语法中，相同缩进代表同一个级别

4.1.2.2.2 配置Map数据

同上面的对象写法

4.1.2.2.3 配置数组（List、Set）数据

 语法：

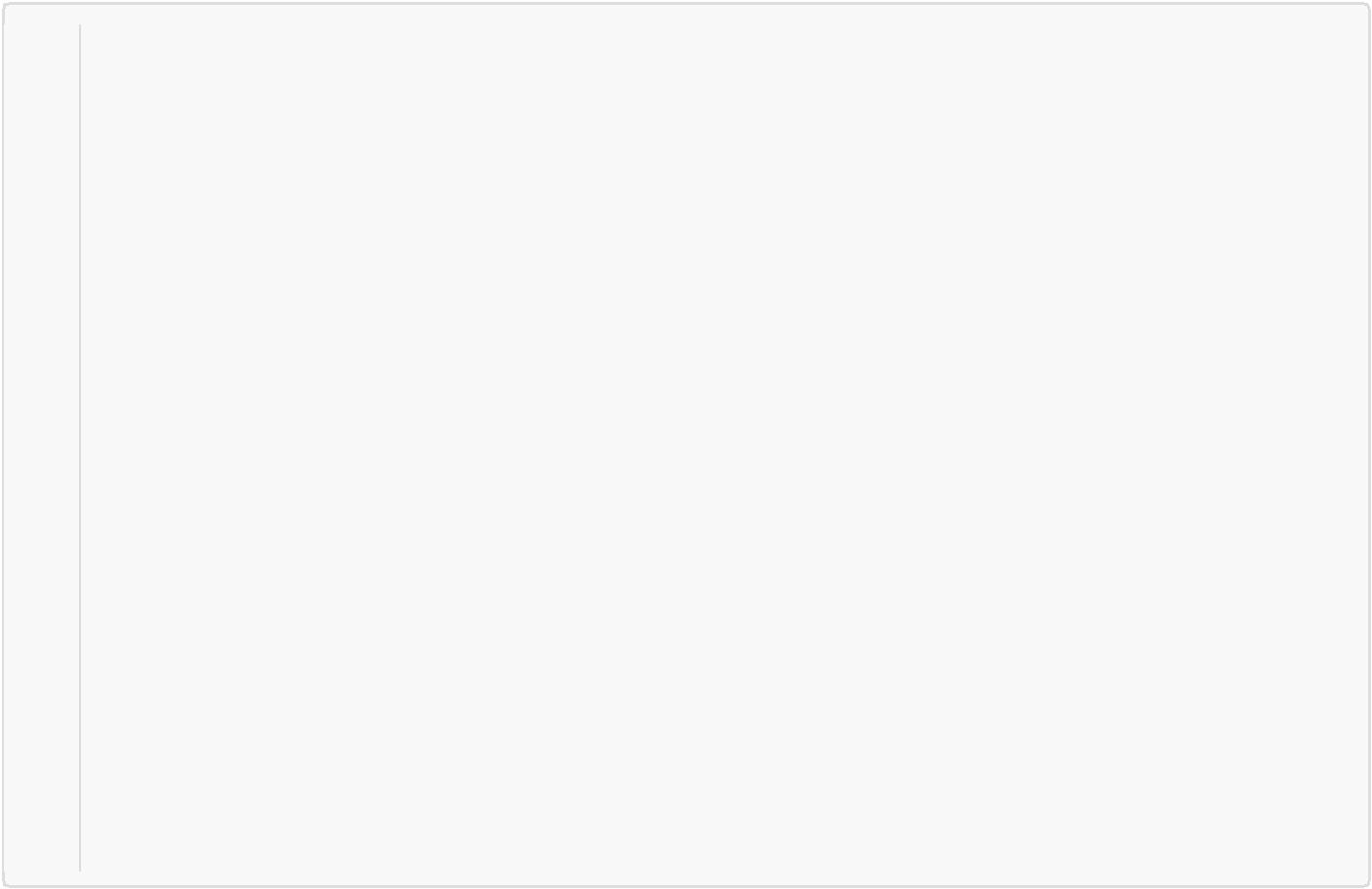
key:

* value1
* value2

或者：

key: [value1,value2]

 示例代码：



1 city:



* - beijing

3- tianjin

4- shanghai

5- chongqing

6

7 #或者

8

9 city: [beijing,tianjin,shanghai,chongqing]

10

1. #集合中的元素是对象形式
2. student:
3. - name: zhangsan
4. age: 18
5. score: 100
6. - name: lisi
7. age: 28
8. score: 88
9. - name: wangwu
10. age: 38
11. score: 90

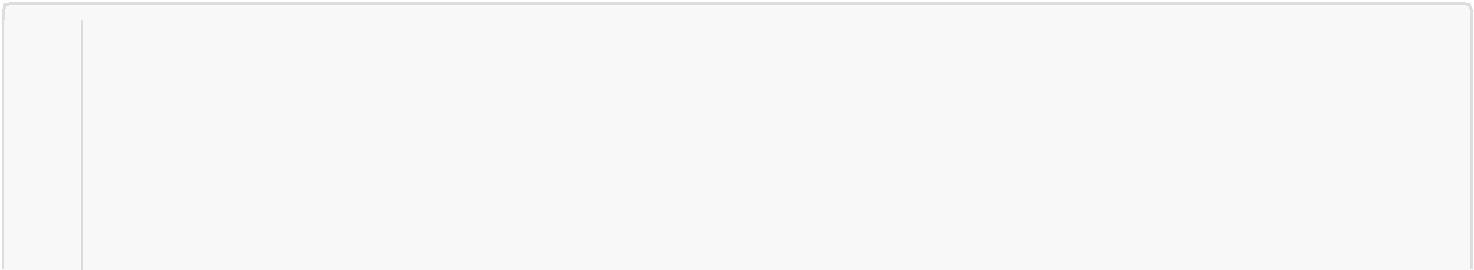
 注意：value1与之间的 - 之间存在一个空格

4.1.3 SpringBoot配置信息的查询

上面提及过，SpringBoot的配置文件，主要的目的就是对配置信息进行修改的，但在配置时的key从哪里去查询呢？我们可以查阅SpringBoot的官方文档

[文档URL：https://docs.spring.io/spring-boot/docs/2.0.1.RELEASE/reference/htmlsingle/#common-application-properties](https://docs.spring.io/spring-boot/docs/2.0.1.RELEASE/reference/htmlsingle/#common-application-properties)

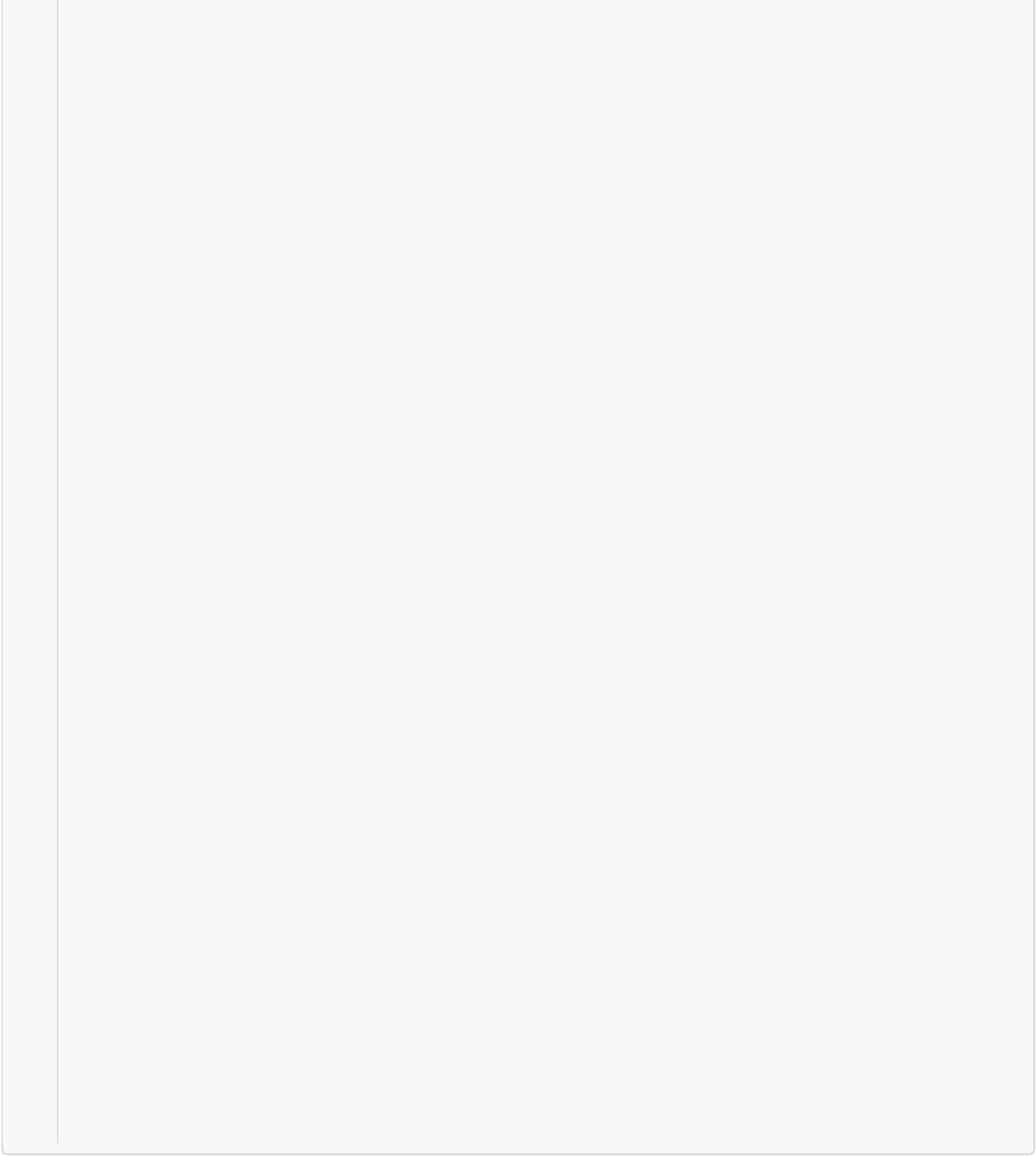
常用的配置摘抄如下：



* # QUARTZ SCHEDULER (QuartzProperties)
* spring.quartz.jdbc.initialize-schema=embedded # Database schema initialization mode.

3 spring.quartz.jdbc.schema=classpath:org/quartz/impl/jdbcjobstore/tables\_@@platform@@. sql # Path to the SQL file to use to initialize the database schema.

4 spring.quartz.job-store-type=memory # Quartz job store type.

* spring.quartz.properties.\*= # Additional Quartz Scheduler properties.

6

7 # ----------------------------------------

8 # WEB PROPERTIES

9 # ----------------------------------------

10

1. # EMBEDDED SERVER CONFIGURATION (ServerProperties)
2. server.port=8080 # Server HTTP port.
3. server.servlet.context-path= # Context path of the application.
4. server.servlet.path=/ # Path of the main dispatcher servlet.

15

1. # HTTP encoding (HttpEncodingProperties)
2. spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and responses. Added to

the "Content-Type" header if not set explicitly.

18

1. # JACKSON (JacksonProperties)
2. spring.jackson.date-format= # Date format string or a fully-qualified date format class name. For instance, `yyyy-MM-dd HH:mm:ss`.

21

1. # SPRING MVC (WebMvcProperties)
2. spring.mvc.servlet.load-on-startup=-1 # Load on startup priority of the dispatcher servlet.
3. spring.mvc.static-path-pattern=/\*\* # Path pattern used for static resources.
4. spring.mvc.view.prefix= # Spring MVC view prefix.
5. spring.mvc.view.suffix= # Spring MVC view suffix.

27

1. # DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
2. spring.datasource.driver-class-name= # Fully qualified name of the JDBC driver. Auto-detected based on the URL by default.
3. spring.datasource.password= # Login password of the database.
4. spring.datasource.url= # JDBC URL of the database.
5. spring.datasource.username= # Login username of the database.

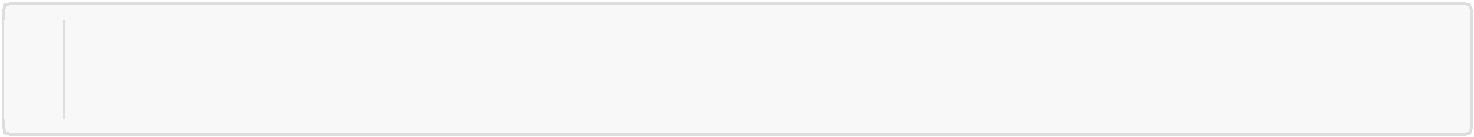
33

1. # JEST (Elasticsearch HTTP client) (JestProperties)
2. spring.elasticsearch.jest.password= # Login password.
3. spring.elasticsearch.jest.proxy.host= # Proxy host the HTTP client should use.
4. spring.elasticsearch.jest.proxy.port= # Proxy port the HTTP client should use.
5. spring.elasticsearch.jest.read-timeout=3s # Read timeout.
6. spring.elasticsearch.jest.username= # Login username.

40

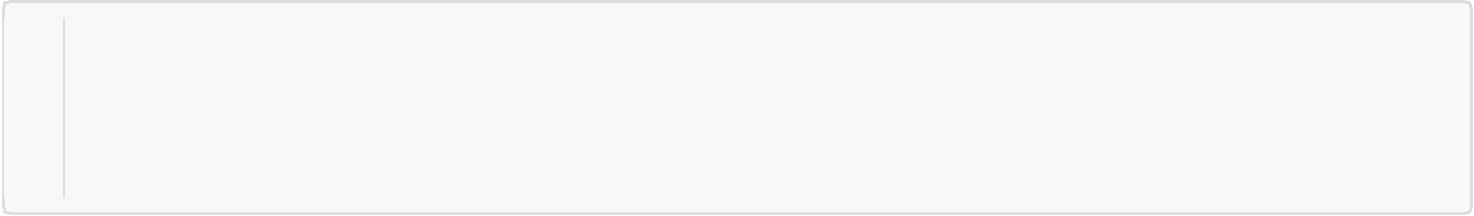
我们可以通过配置application.poperties 或者 application.yml 来修改SpringBoot的默认配置例如：

application.properties文件



* server.port=8888
* server.servlet.context-path=demo

application.yml文件

* server:
* port: 8888
* servlet:
* context-path: /demo

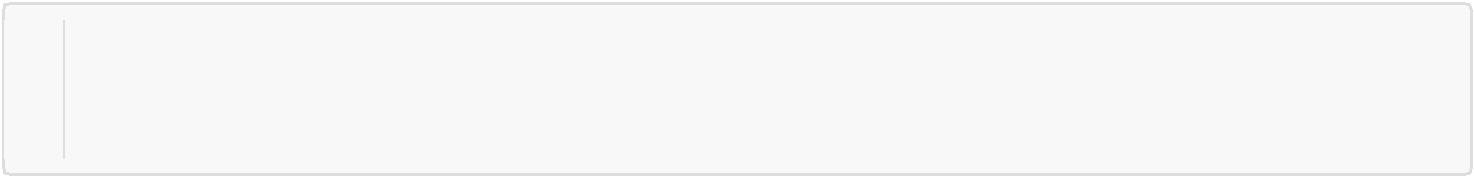
4.2 配置文件与配置类的属性映射方式



4.2.1 使用注解@Value映射

我们可以通过@Value注解将配置文件中的值映射到一个Spring管理的Bean的字段上例如：

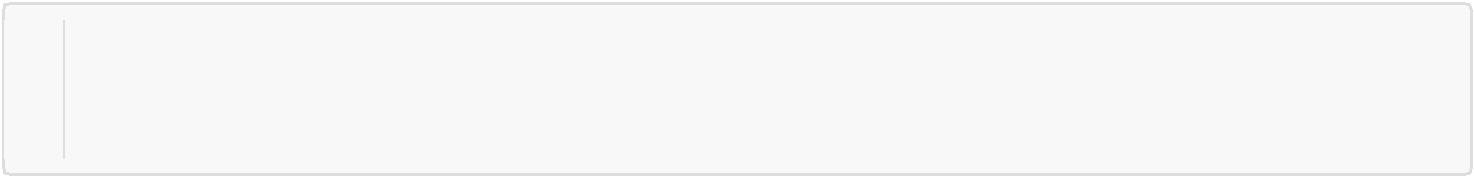
application.properties配置如下：



* person:
* name: zhangsan

3age: 18

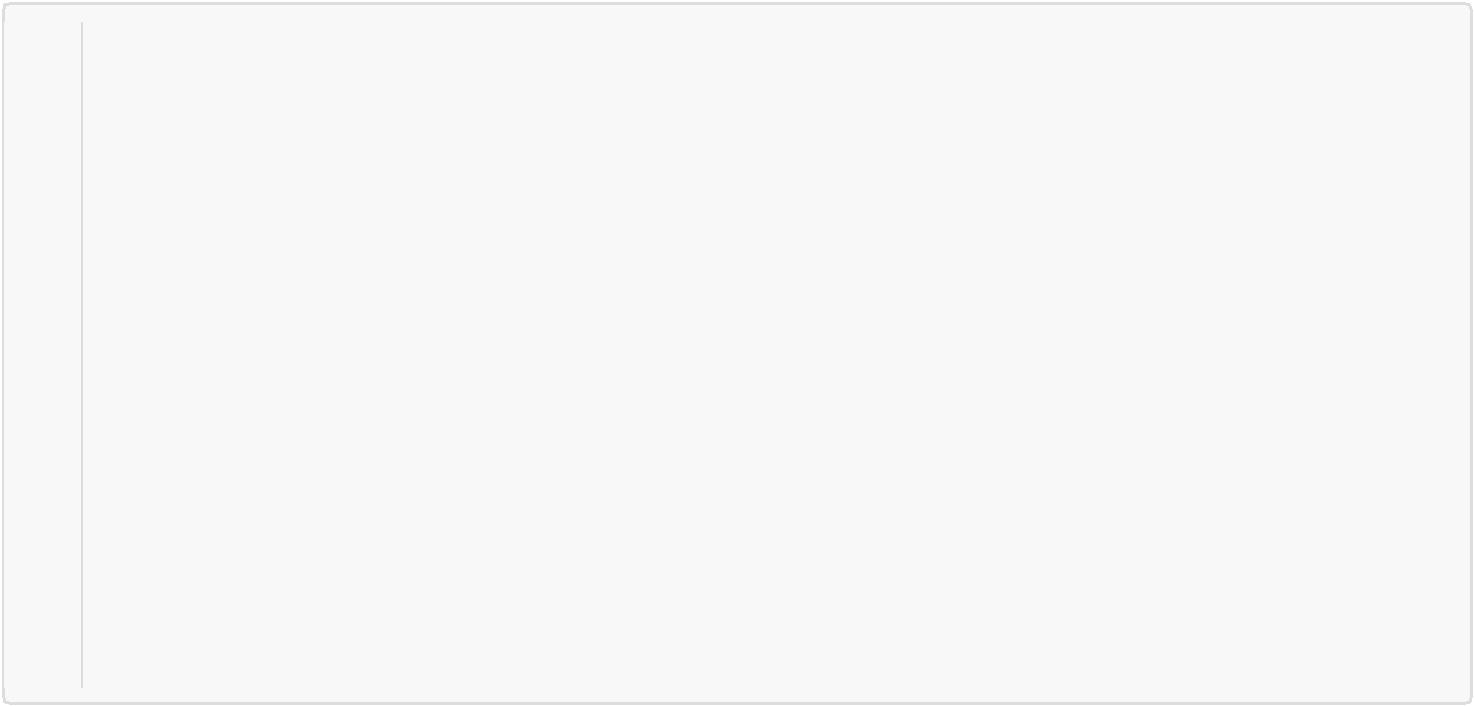
或者，application.yml配置如下：



* person:
* name: zhangsan

3age: 18

实体Bean代码如下：



* @Controller
* public class QuickStartController {

3

* @Value("${person.name}")

5private String name;

* @Value("${person.age}")

7private Integer age;

8

9

1. @RequestMapping("/quick")
2. @ResponseBody
3. public String quick(){
4. return "springboot 访问成功! name="+name+",age="+age;
5. }

15

1. }

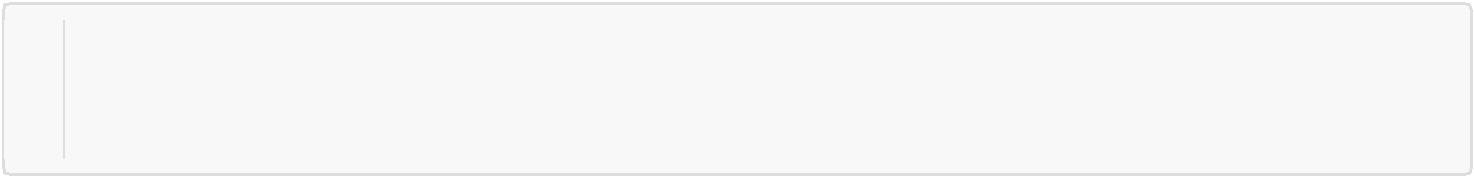
浏览器访问地址：<http://localhost:8080/quick>结果如下：



4.2.2 使用注解@ConfigurationProperties映射

通过注解@ConfigurationProperties(prefix="配置文件中的key的前缀")可以将配置文件中的配置自动与实体进行映射

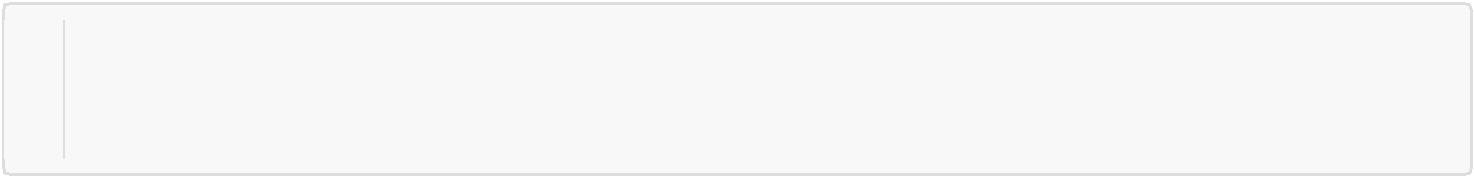
application.properties配置如下：



* person:
* name: zhangsan

3age: 18

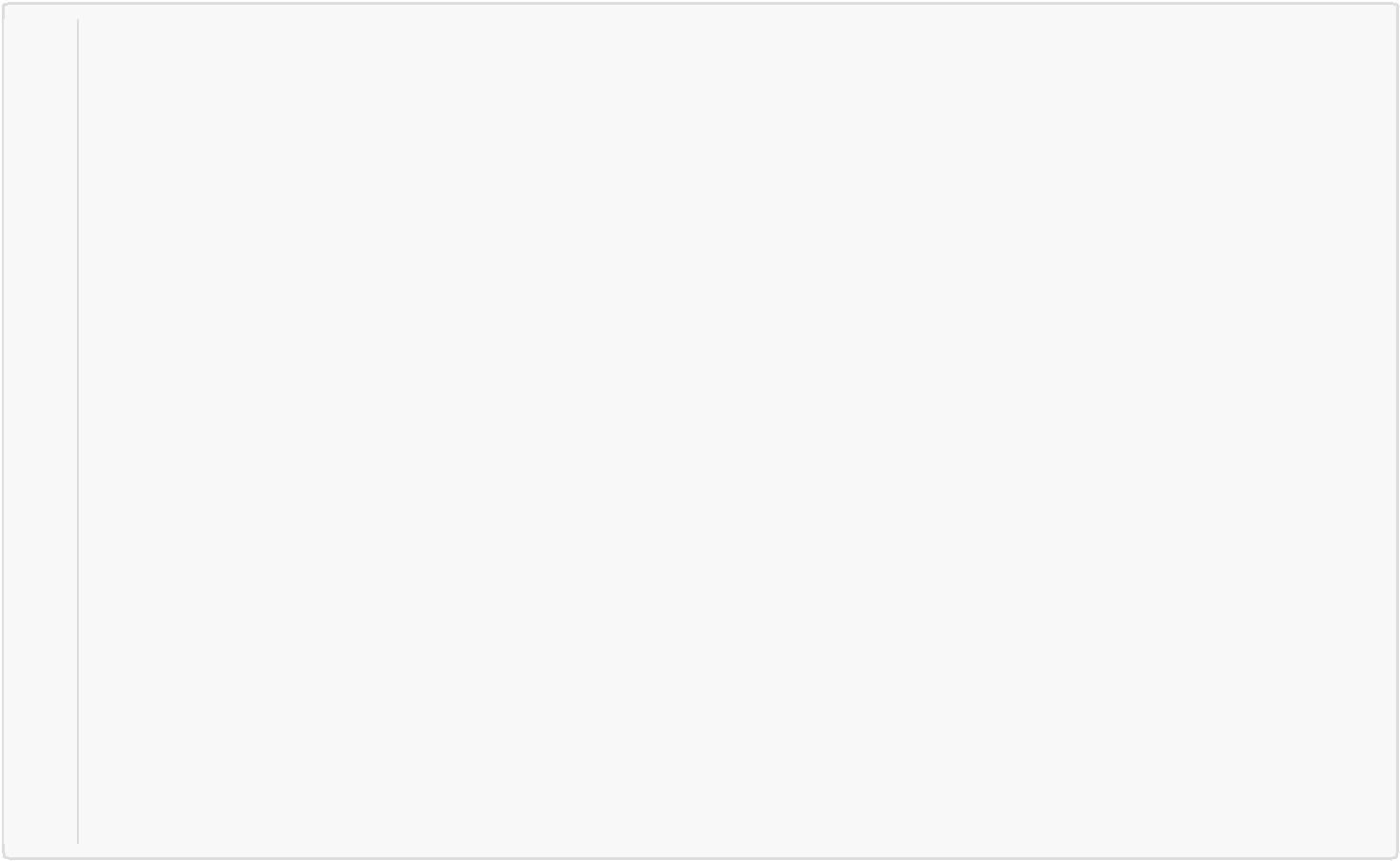
或者，application.yml配置如下：



* person:
* name: zhangsan

3age: 18

实体Bean代码如下：



* @Controller
* @ConfigurationProperties(prefix = "person")

3 public class QuickStartController {

4

5private String name;

6private Integer age;

7

* @RequestMapping("/quick")
* @ResponseBody

1. public String quick(){
2. return "springboot 访问成功! name="+name+",age="+age;
3. }

13

1. public void setName(String name) {
2. this.name = name;
3. }

17

1. public void setAge(Integer age) {
2. this.age = age;
3. }
4. }

浏览器访问地址：<http://localhost:8080/quick>结果如下：



注意：使用@ConfigurationProperties方式可以进行配置文件与实体字段的自动映射，但需要字段必须提供set方法才可以，而使用@Value注解修饰的字段不需要提供set方法

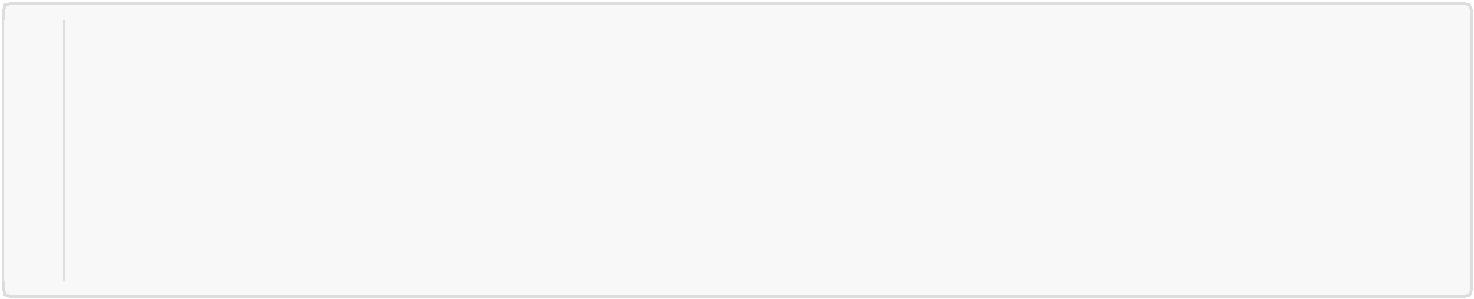
五、SpringBoot与整合其他技术



5.1 SpringBoot整合Mybatis



5.1.1 添加Mybatis的起步依赖

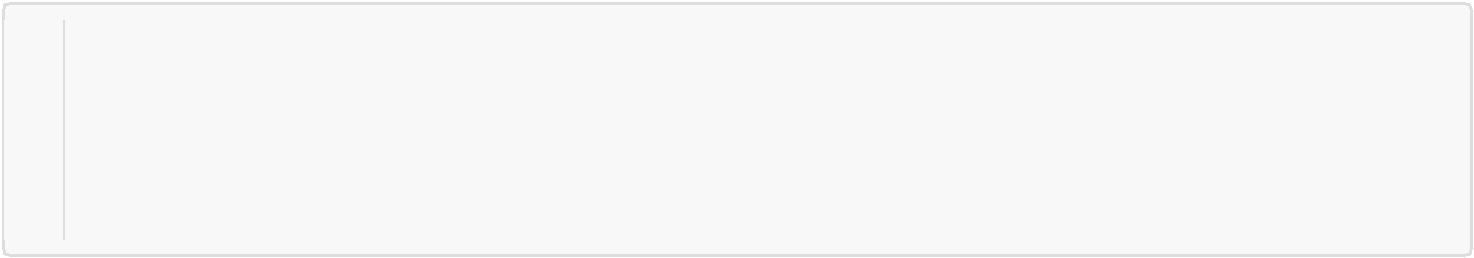


* <!--mybatis起步依赖-->

2 <dependency>

* <groupId>org.mybatis.spring.boot</groupId>
* <artifactId>mybatis-spring-boot-starter</artifactId>
* <version>1.1.1</version> 6 </dependency>

5.1.2 添加数据库驱动坐标



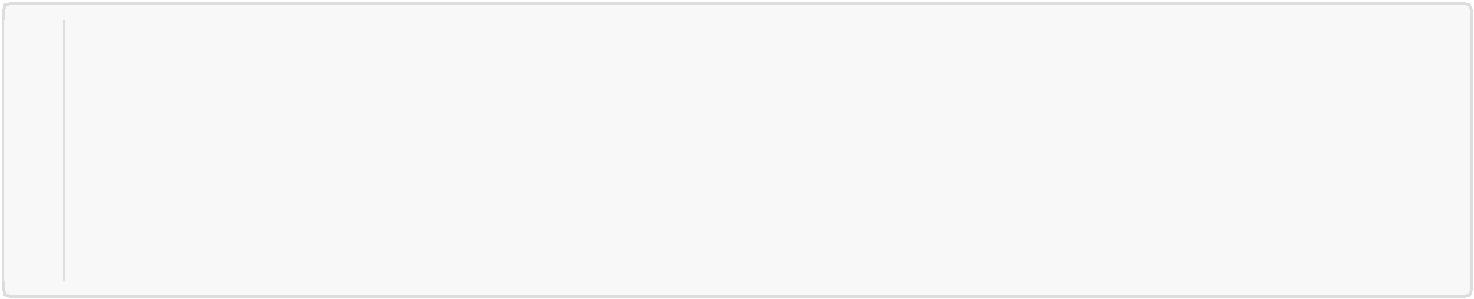
* <!-- MySQL连接驱动 -->

2 <dependency>

* <groupId>mysql</groupId>
* <artifactId>mysql-connector-java</artifactId> 5 </dependency>

5.1.3 添加数据库连接信息

在application.properties中添加数据量的连接信息



* #DB Configuration:
* spring.datasource.driverClassName=com.mysql.jdbc.Driver

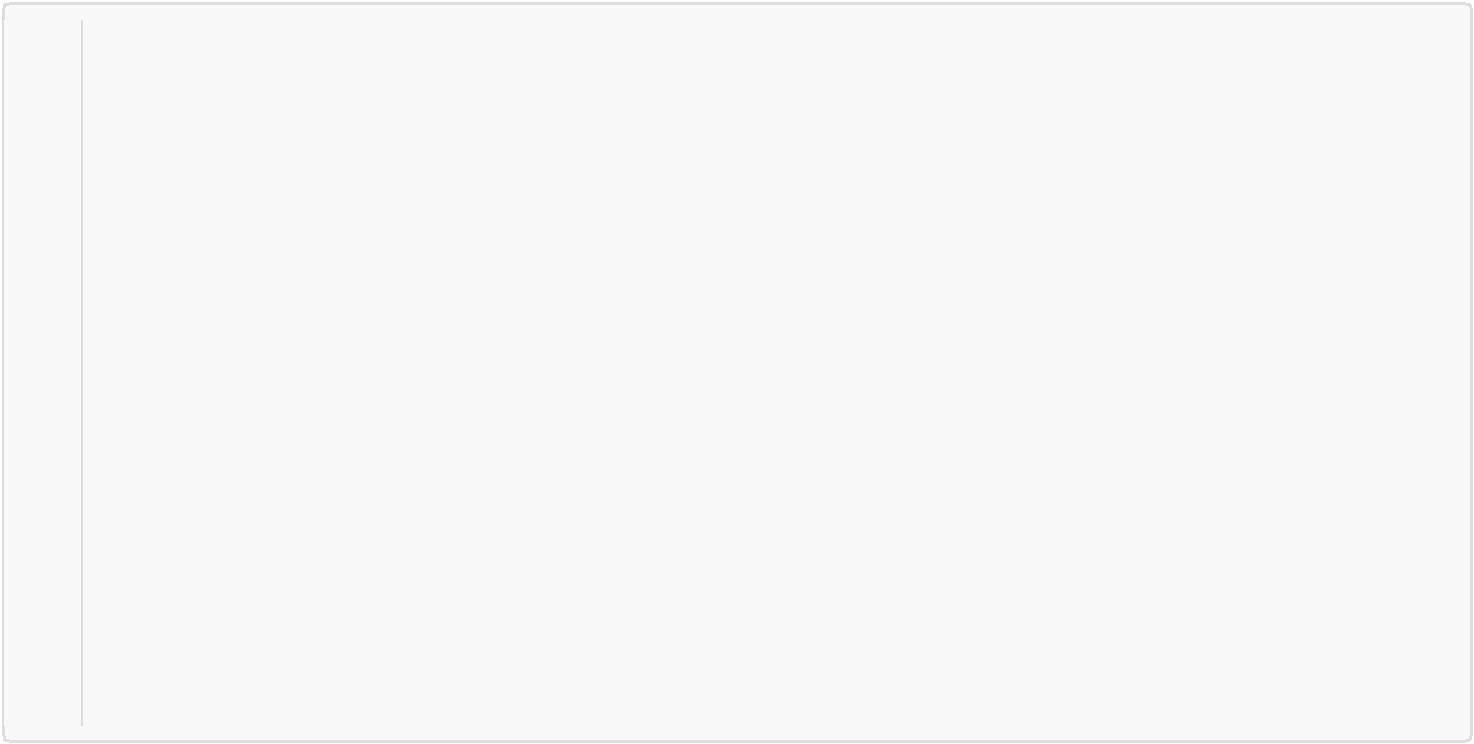
3 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test? useUnicode=true&characterEncoding=utf8

4 spring.datasource.username=root

5 spring.datasource.password=root

5.1.4 创建user表

在test数据库中创建user表



* -- ----------------------------

2 -- Table structure for `user`

3 -- ----------------------------

4 DROP TABLE IF EXISTS `user`;

5 CREATE TABLE `user` (

6`id` int(11) NOT NULL AUTO\_INCREMENT,

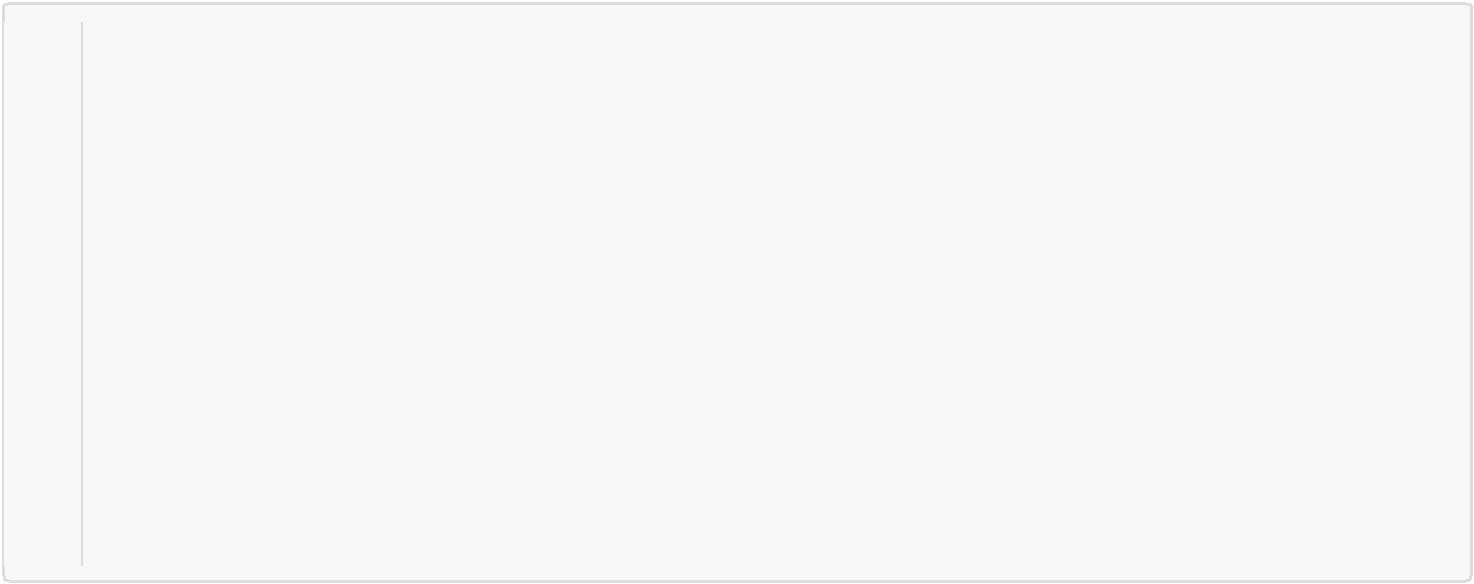
7`username` varchar(50) DEFAULT NULL,

8`password` varchar(50) DEFAULT NULL,

9`name` varchar(50) DEFAULT NULL,

1. PRIMARY KEY (`id`)
2. ) ENGINE=InnoDB AUTO\_INCREMENT=10 DEFAULT CHARSET=utf8;
3. -- ----------------------------
4. -- Records of user
5. -- ----------------------------
6. INSERT INTO `user` VALUES ('1', 'zhangsan', '123', '张三');
7. INSERT INTO `user` VALUES ('2', 'lisi', '123', '李四');

5.1.5 创建实体Bean



* public class User {
* // 主键
* private Long id;

4// 用户名

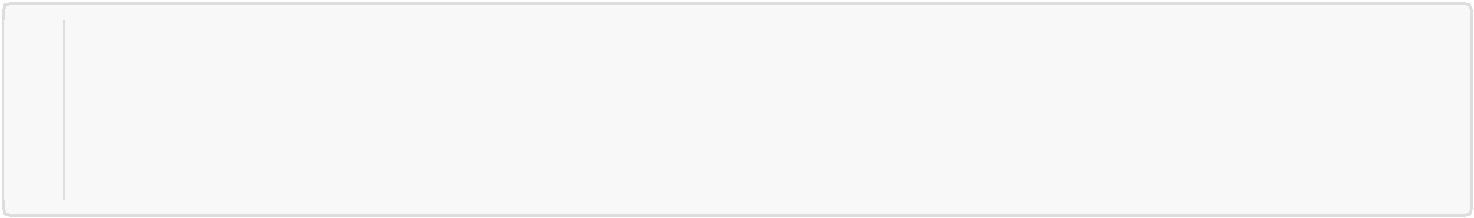
5private String username;

* // 密码
* private String password;
* // 姓名
* private String name;

10

1. //此处省略getter和setter方法 .. ..
2. }

5.1.6 编写Mapper



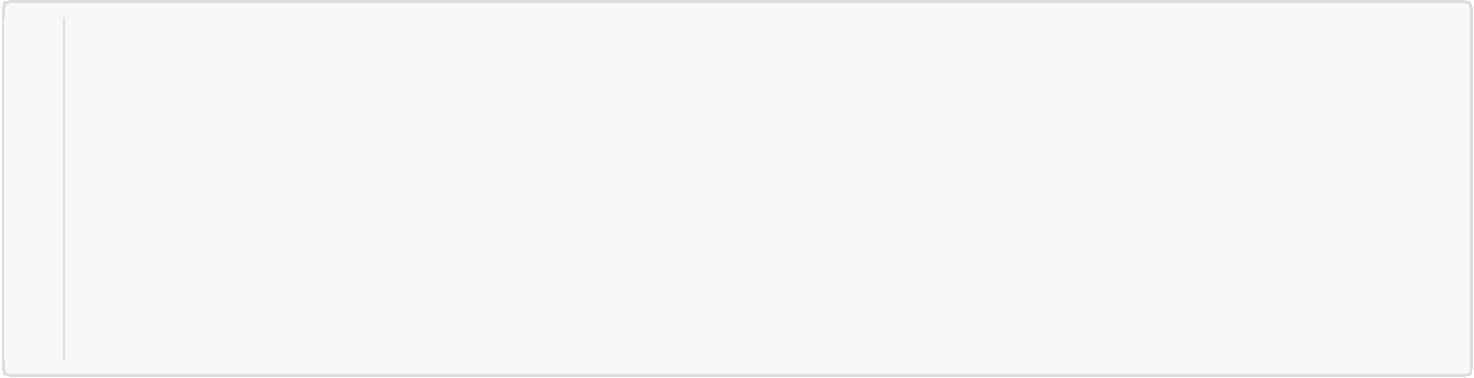
* @Mapper
* public interface UserMapper {
* public List<User> queryUserList();

4 }

注意：@Mapper标记该类是一个mybatis的mapper接口，可以被spring boot自动扫描到spring上下文中

5.1.7 配置Mapper映射文件

在src\main\resources\mapper路径下加入UserMapper.xml配置文件"

* <?xml version="1.0" encoding="utf-8" ?>
* <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
* <mapper namespace="com.itheima.mapper.UserMapper">

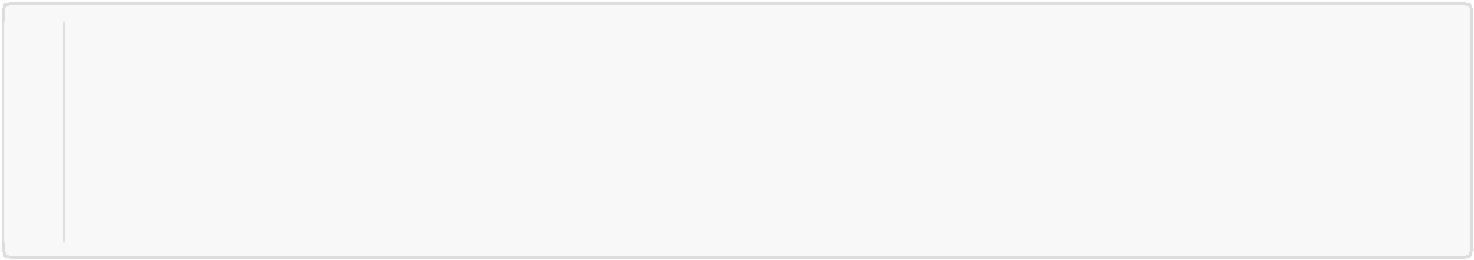
4<select id="queryUserList" resultType="user">

5select \* from user

* </select>

7 </mapper>

5.1.8 在application.properties中添加mybatis的信息



* #spring集成Mybatis环境

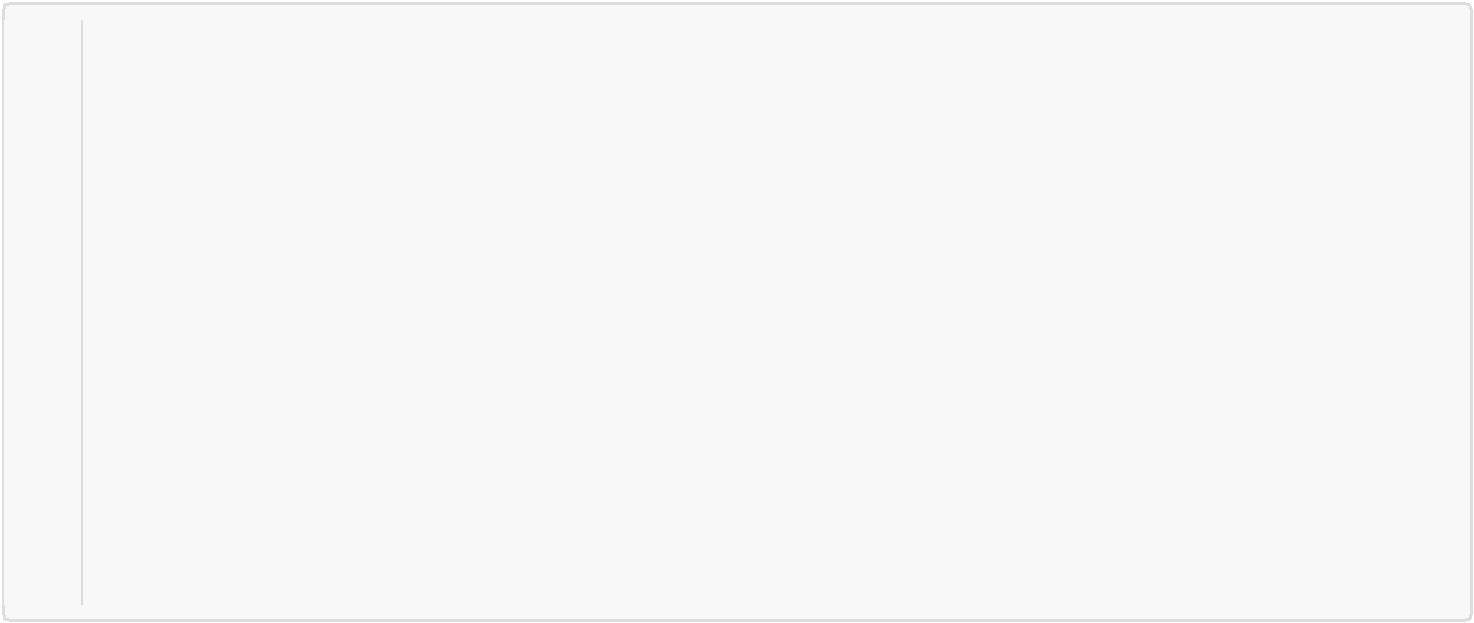
2 #pojo别名扫描包

3 mybatis.type-aliases-package=com.itheima.domain

4 #加载Mybatis映射文件

5 mybatis.mapper-locations=classpath:mapper/\*Mapper.xml

5.1.9 编写测试Controller



* @Controller
* public class MapperController {

3

* @Autowired
* private UserMapper userMapper;

6

* @RequestMapping("/queryUser")
* @ResponseBody
* public List<User> queryUser(){

1. List<User> users = userMapper.queryUserList();
2. return users;
3. }

13

1. }

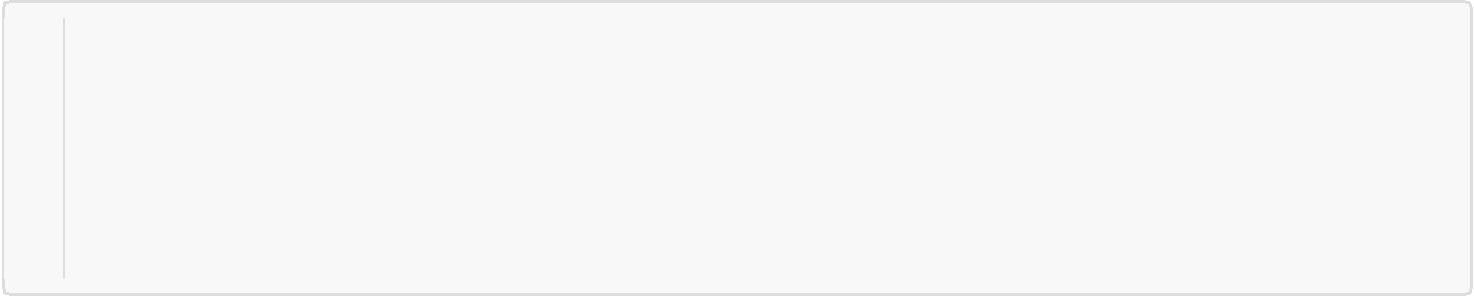
5.1.10 测试



5.2 SpringBoot整合Junit



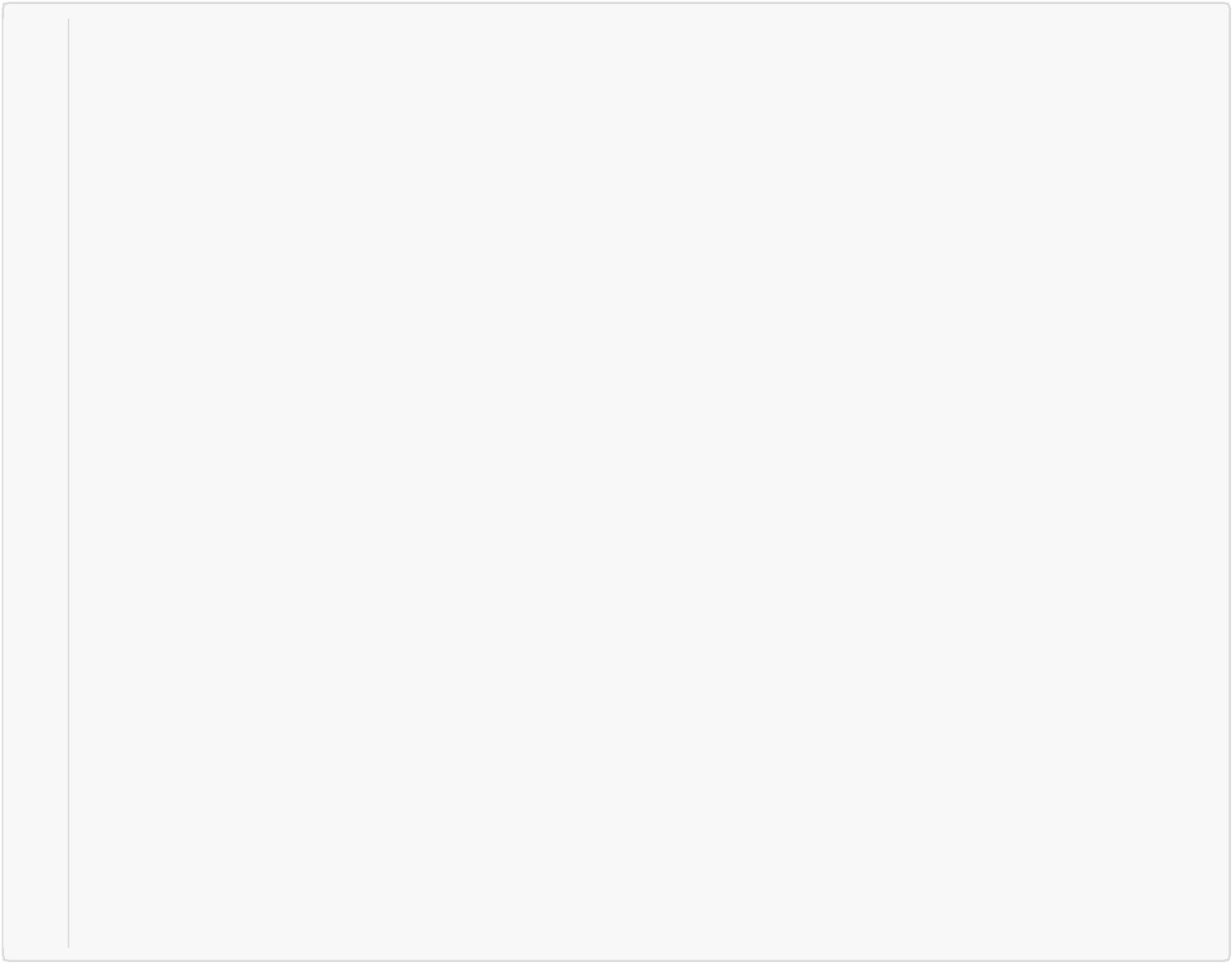
5.2.1 添加Junit的起步依赖

* <!--测试的起步依赖-->

2 <dependency>

* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-test</artifactId>
* <scope>test</scope> 6 </dependency>

5.2.2 编写测试类



* package com.itheima.test;

2

3 import com.itheima.MySpringBootApplication; 4 import com.itheima.domain.User;

5 import com.itheima.mapper.UserMapper; 6 import org.junit.Test;

7 import org.junit.runner.RunWith;

8 import org.springframework.beans.factory.annotation.Autowired; 9 import org.springframework.boot.test.context.SpringBootTest;

1. import org.springframework.test.context.junit4.SpringRunner;
2. import java.util.List;
3. @RunWith(SpringRunner.class)
4. @SpringBootTest(classes = MySpringBootApplication.class)
5. public class MapperTest {
6. @Autowired
7. private UserMapper userMapper;
8. @Test
9. public void test() {
10. List<User> users = userMapper.queryUserList();
11. System.out.println(users);
12. }

26

1. }

其中，

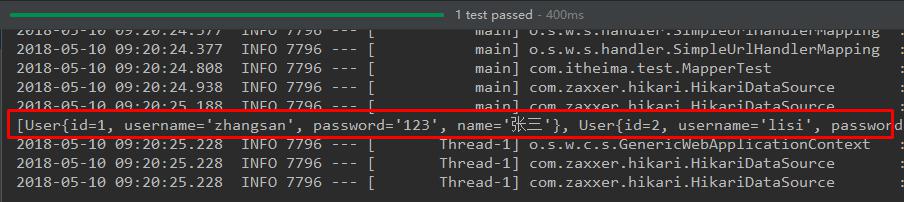
SpringRunner继承自SpringJUnit4ClassRunner，使用哪一个Spring提供的测试测试引擎都可以



* public final class SpringRunner extends SpringJUnit4ClassRunner

@SpringBootTest的属性指定的是引导类的字节码对象

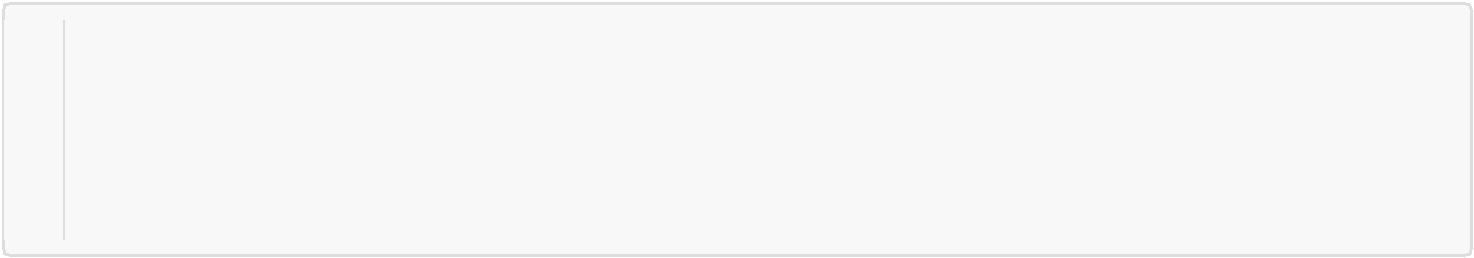
5.2.3 控制台打印信息



5.3 SpringBoot整合Spring Data JPA



5.3.1 添加Spring Data JPA的起步依赖

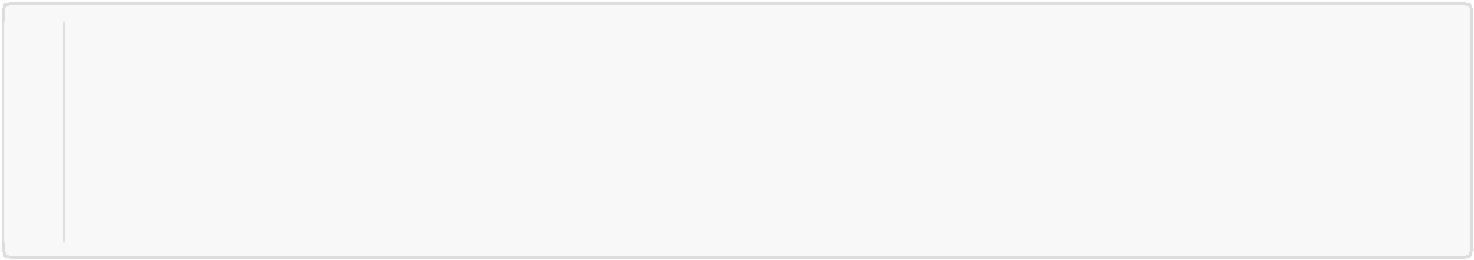


* <!-- springBoot JPA的起步依赖 -->

2 <dependency>

* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-data-jpa</artifactId> 5 </dependency>

5.3.2 添加数据库驱动依赖

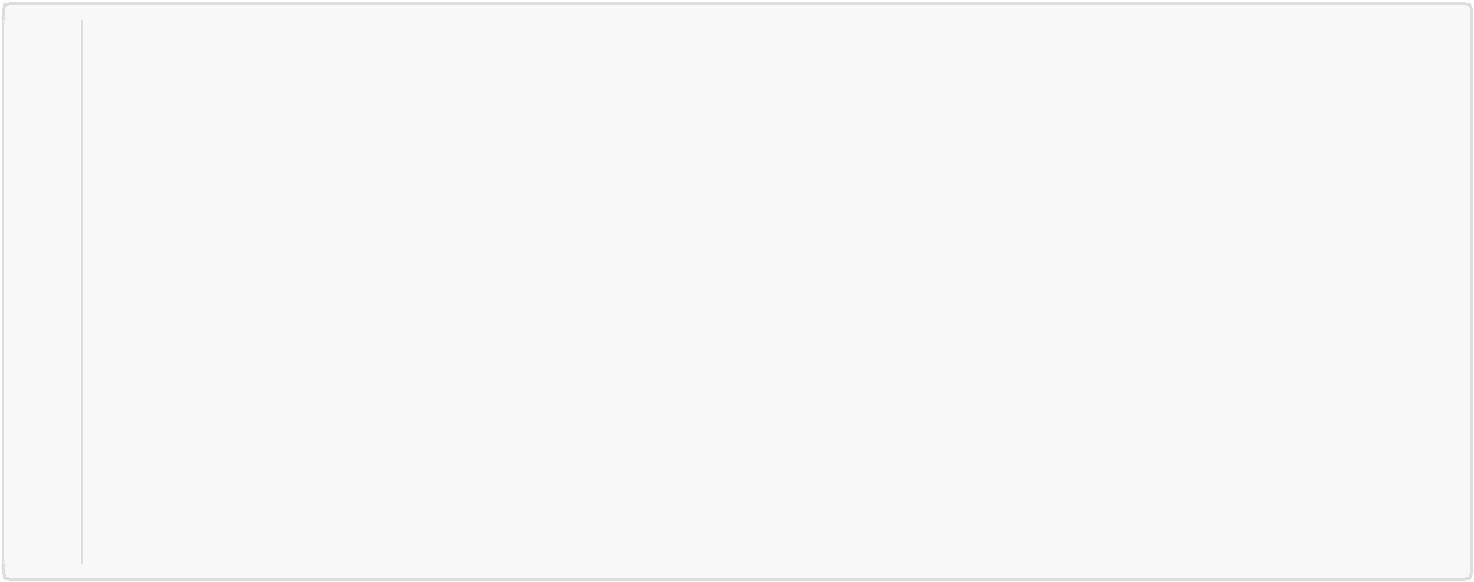


* <!-- MySQL连接驱动 -->

2 <dependency>

* <groupId>mysql</groupId>
* <artifactId>mysql-connector-java</artifactId> 5 </dependency>

5.3.3 在application.properties中配置数据库和jpa的相关属性



* #DB Configuration:
* spring.datasource.driverClassName=com.mysql.jdbc.Driver

3 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test? useUnicode=true&characterEncoding=utf8

4 spring.datasource.username=root

5 spring.datasource.password=root

6

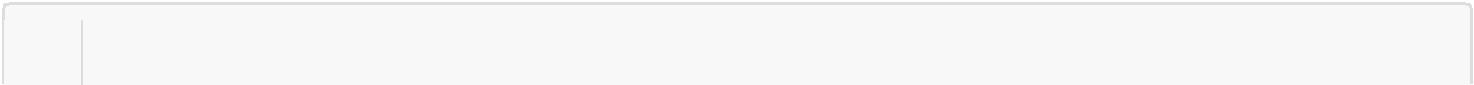
7 #JPA Configuration:

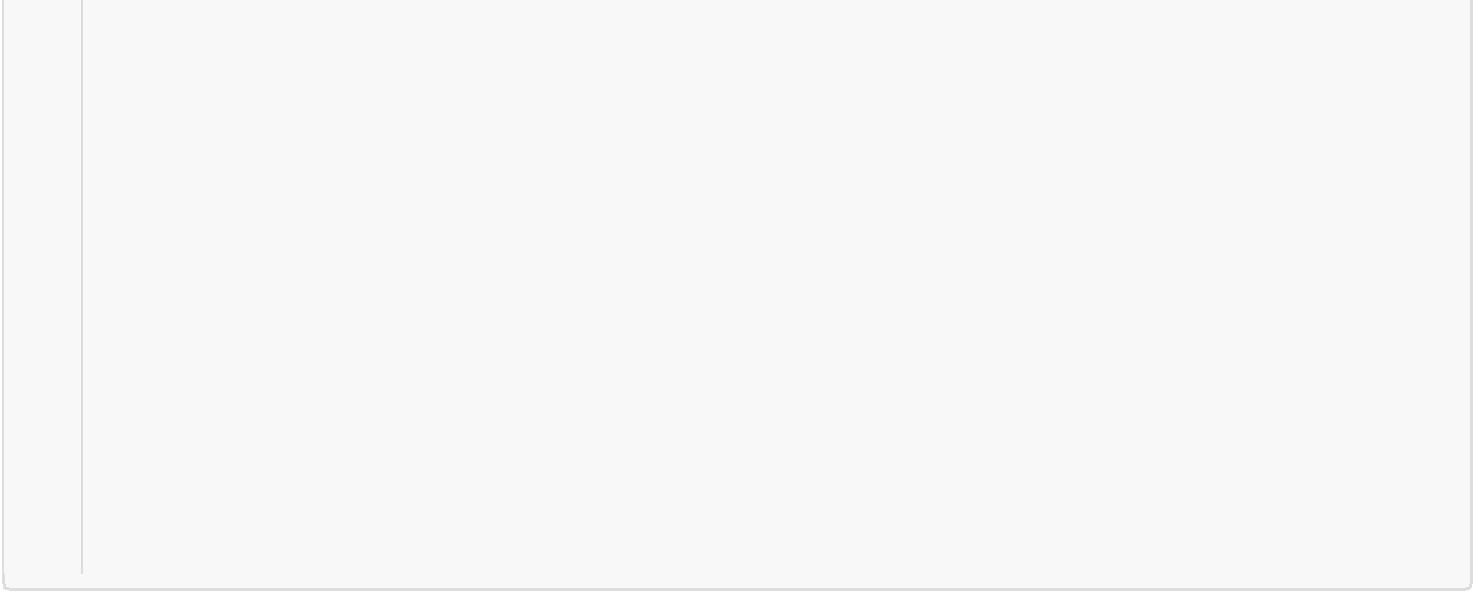
8 spring.jpa.database=MySQL

9 spring.jpa.show-sql=true

1. spring.jpa.generate-ddl=true
2. spring.jpa.hibernate.ddl-auto=update
3. spring.jpa.hibernate.naming\_strategy=org.hibernate.cfg.ImprovedNamingStrategy

5.3.4 创建实体配置实体



* @Entity
* public class User {
* // 主键
* @Id
* @GeneratedValue(strategy = GenerationType.IDENTITY)

6private Long id;

7// 用户名

8private String username;

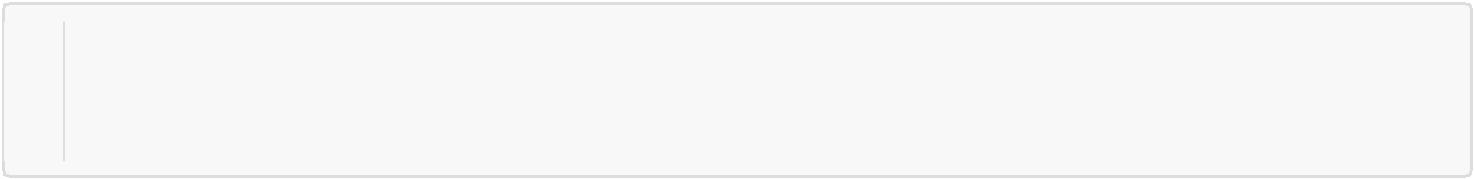
* // 密码

1. private String password;
2. // 姓名
3. private String name;

13

1. //此处省略setter和getter方法... ...
2. }

5.3.5 编写UserRepository

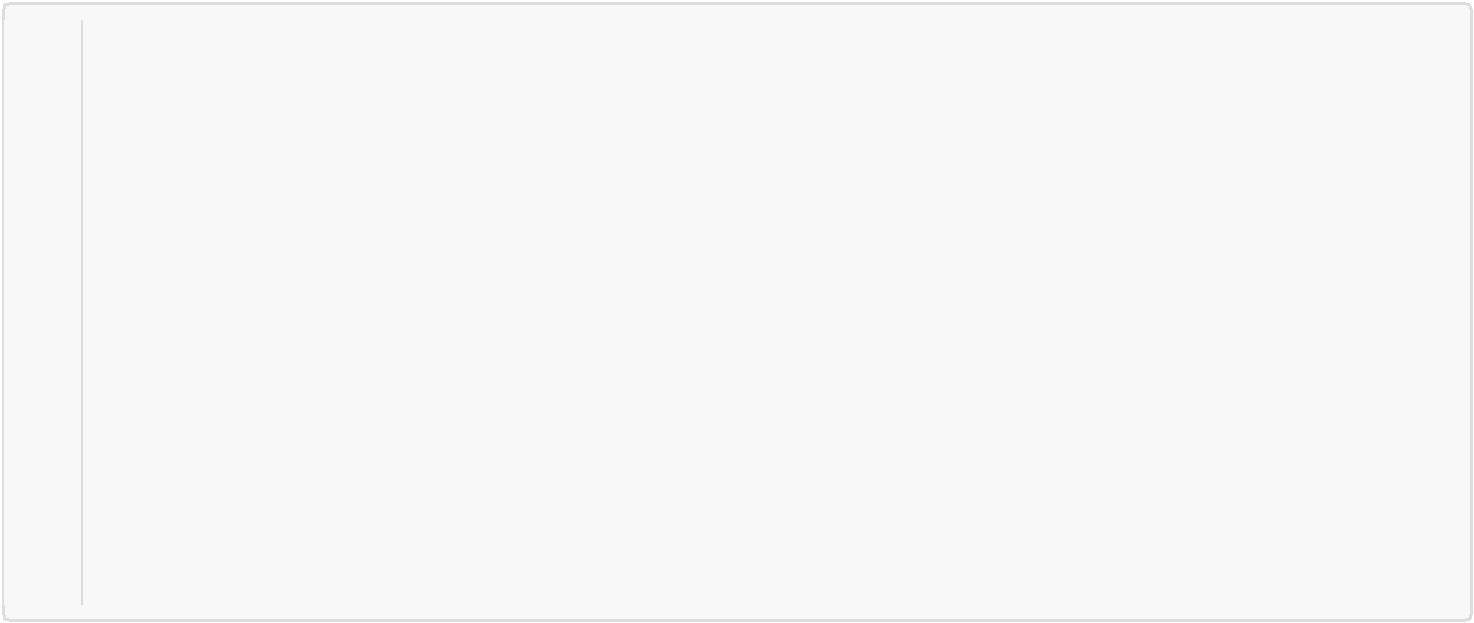


* public interface UserRepository extends JpaRepository<User,Long>{

2public List<User> findAll();

3 }

5.3.6 编写测试类



* @RunWith(SpringRunner.class)
* @SpringBootTest(classes=MySpringBootApplication.class)

3 public class JpaTest {

4

* @Autowired
* private UserRepository userRepository;

7

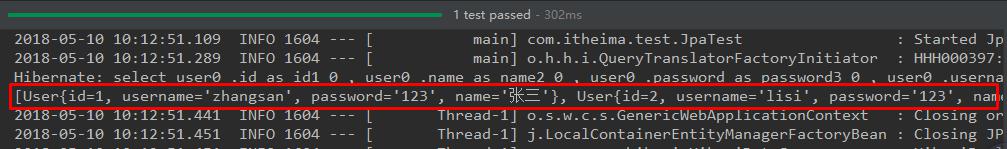
* @Test
* public void test(){

1. List<User> users = userRepository.findAll();
2. System.out.println(users);
3. }

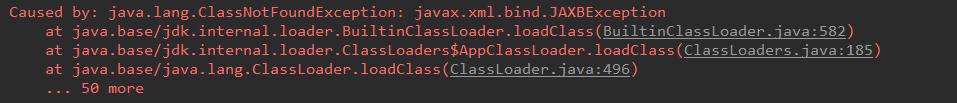
13

1. }

5.3.7 控制台打印信息

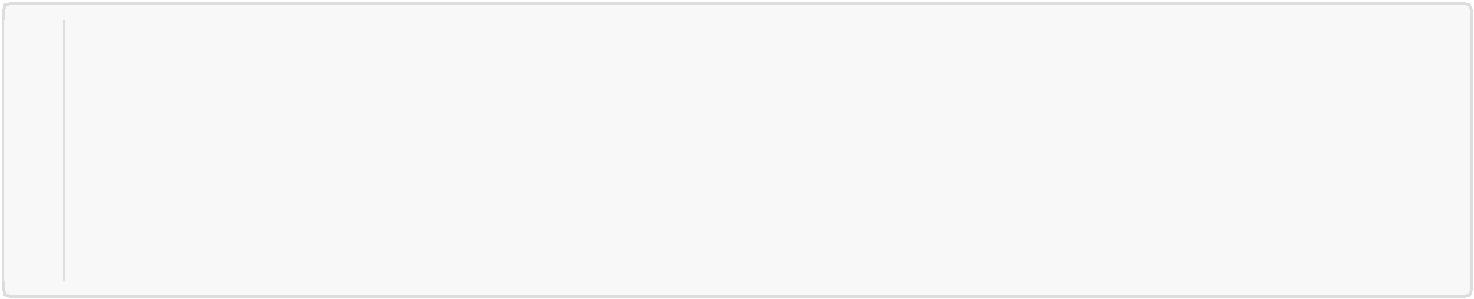


注意：如果是jdk9，执行报错如下：



原因：jdk缺少相应的jar

解决方案：手动导入对应的maven坐标，如下：



* <!--jdk9需要导入如下坐标-->

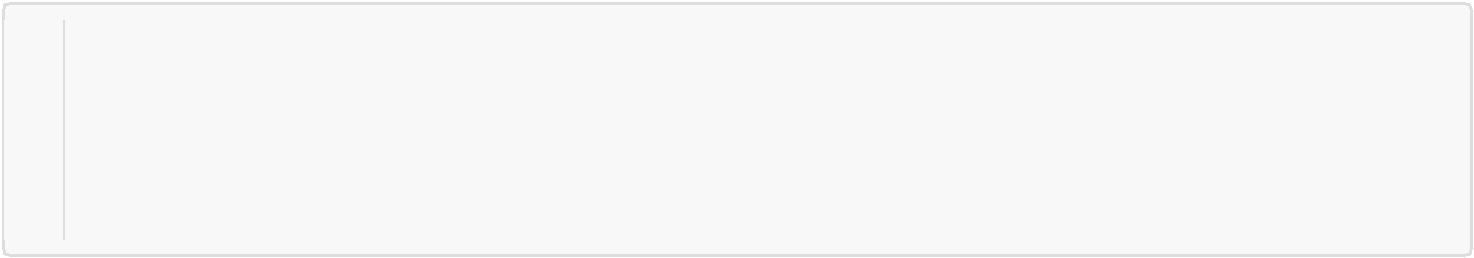
2 <dependency>

* <groupId>javax.xml.bind</groupId>
* <artifactId>jaxb-api</artifactId>
* <version>2.3.0</version> 6 </dependency>

5.4 SpringBoot整合Redis



5.4.1 添加redis的起步依赖

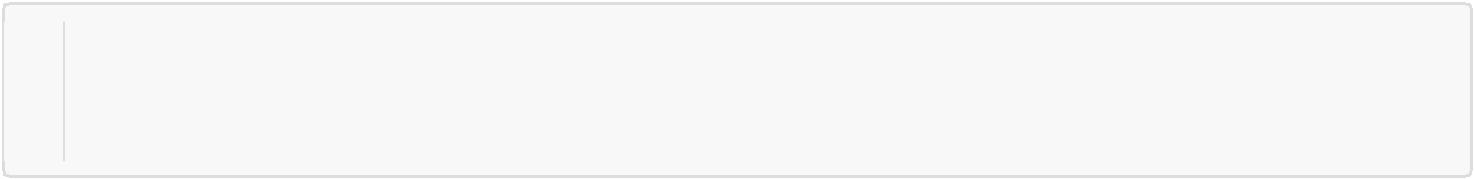


* <!-- 配置使用redis启动器 -->

2 <dependency>

* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-data-redis</artifactId> 5 </dependency>

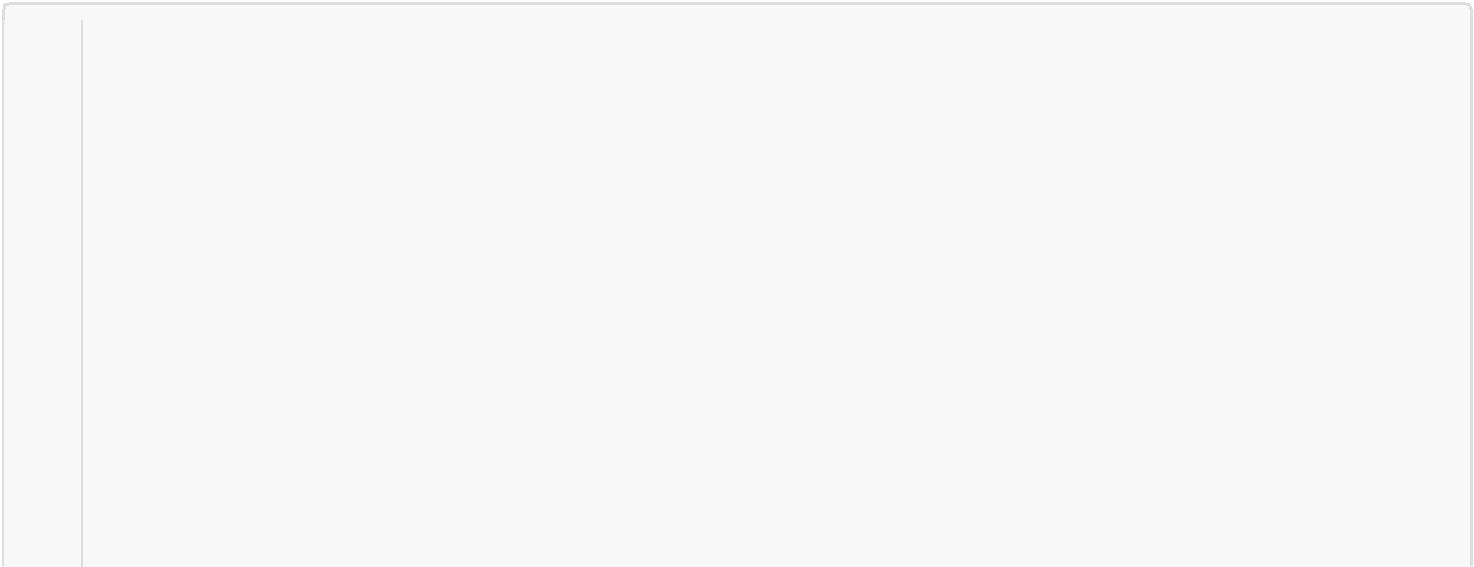
5.4.2 配置redis的连接信息



* #Redis
* spring.redis.host=127.0.0.1

3 spring.redis.port=6379

5.4.3 注入RedisTemplate测试redis操作



* @RunWith(SpringRunner.class)
* @SpringBootTest(classes = SpringbootJpaApplication.class)

3 public class RedisTest {

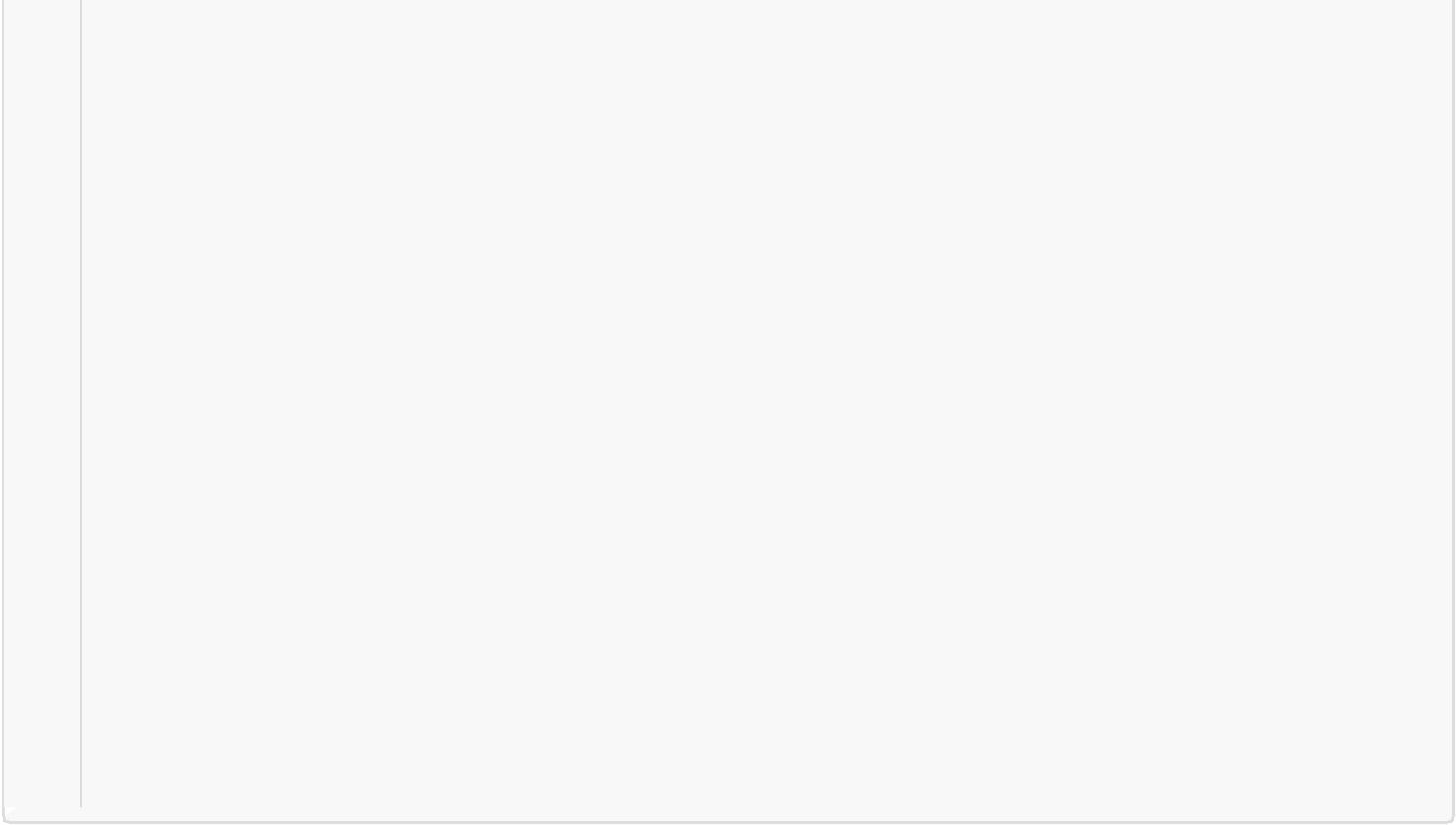
4

* @Autowired
* private UserRepository userRepository;

7

* @Autowired
* private RedisTemplate<String, String> redisTemplate;

10

1. @Test
2. public void test() throws JsonProcessingException {
3. //从redis缓存中获得指定的数据
4. String userListData = redisTemplate.boundValueOps("user.findAll").get();
5. //如果redis中没有数据的话
6. if(null==userListData){

|  |  |
| --- | --- |
| 17 | //查询数据库获得数据 |
| 18 | List<User> all = userRepository.findAll(); |
| 19 | //转换成json格式字符串 |
| 20 | ObjectMapper om = new ObjectMapper(); |
| 21 | userListData = om.writeValueAsString(all); |
| 22 | //将数据存储到redis中，下次在查询直接从redis中获得数据，不用在查询数据库 |
| 23 | redisTemplate.boundValueOps("user.findAll").set(userListData); |
| 24 | System.out.println("===============从数据库获得数据==============="); |

1. }else{

26 System.out.println("===============从redis缓存中获得数据===============");

1. }
2. System.out.println(userListData);
3. }
4. }