第一讲 SpringBoot 基础

一、SpringBoot 简介

1. 原有 Spring 优缺点分析

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。

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。

2. SpringBoot 的概述

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。

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

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。

二、SpringBoot 快速入门

1. 代码实现

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 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>
</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>
</dependencies>
```

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;

import org.springframework.boot.SpringApplication;
 import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
 public class MySpringBootApplication {

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

④ 编写 Controller

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

```
import org.springframework.stereotype.Controller;
 import org.springframework.web.bind.annotation.RequestMapping;
 import org.springframework.web.bind.annotation.ResponseBody;

@Controller
 public class QuickStartController {
        @RequestMapping("/quick")
    @ResponseBody
    public String quick(){
        return "springboot 访问成功!";
    }
}
```

⑤ 测试

执行 SpringBoot 起步类的主方法，控制台打印日志如下：

```
.   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.1.RELEASE)

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)
... ... ...
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type
[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-05-08 14:30:03.126  INFO 5672 --- [           main]
o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on
startup
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 ''
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. 快速入门解析

① 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

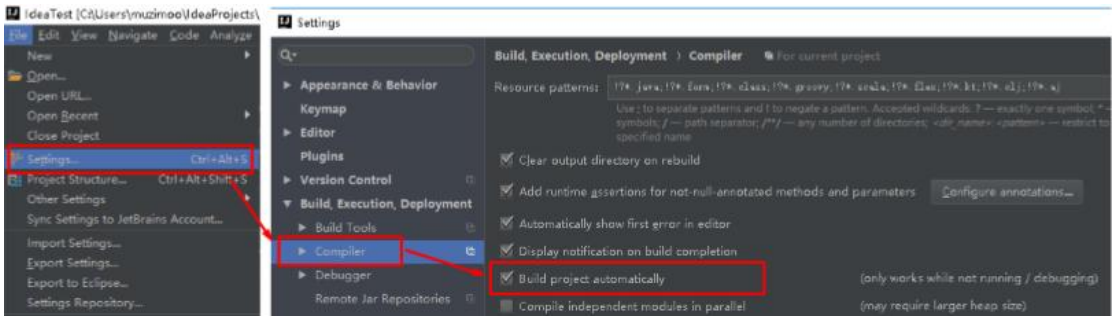
② SpringBoot 工程热部署

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。

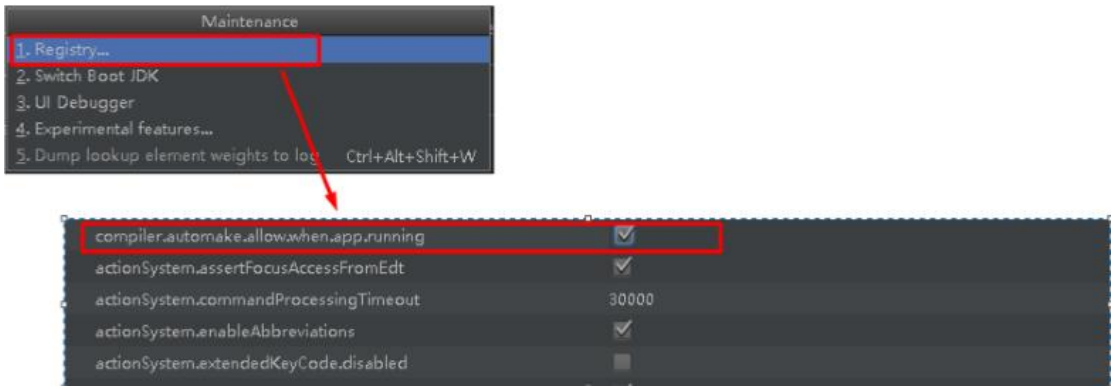
```
<!--热部署配置-->
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
 </dependency>
```

注意：IDEA 进行 SpringBoot 热部署失败原因

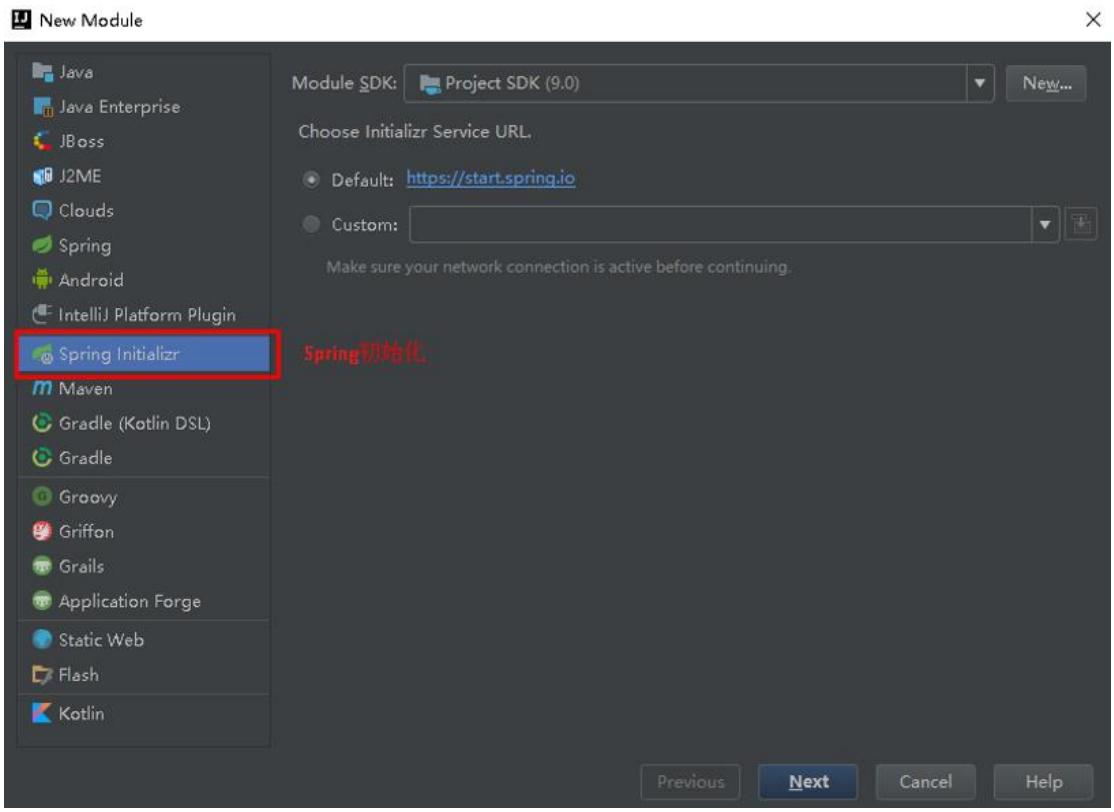
出现这种情况，并不是热部署配置问题，其根本原因是因为 Intellij IEDA 默认情况下不会自动编译，需要对 IDEA 进 行自动编译的设置，如下：

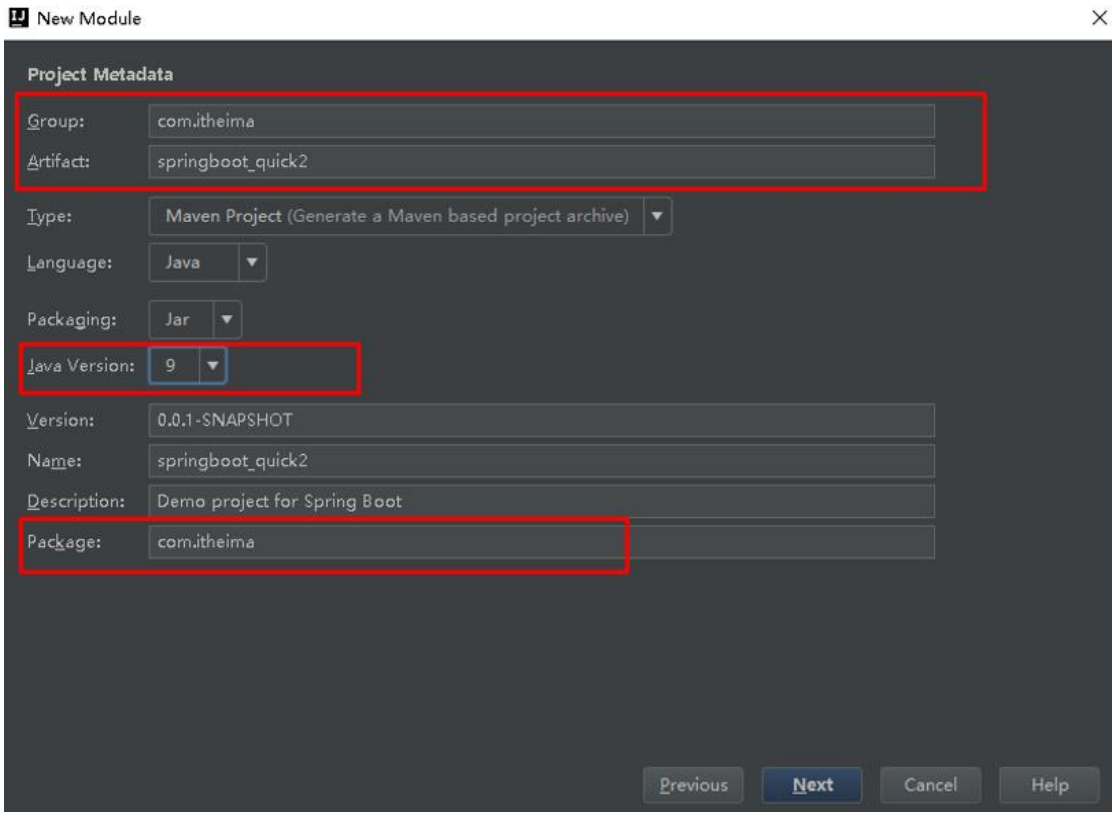


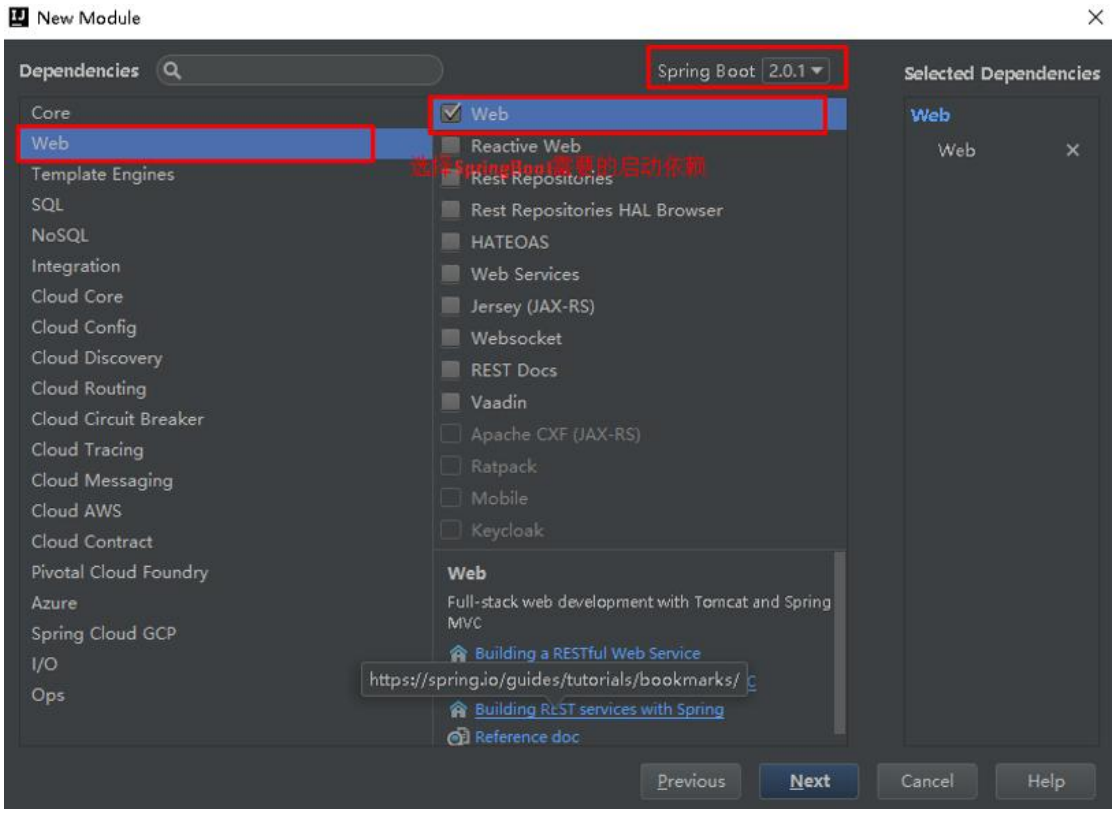
然后 Shift+Ctrl+Alt+/，选择 Registry

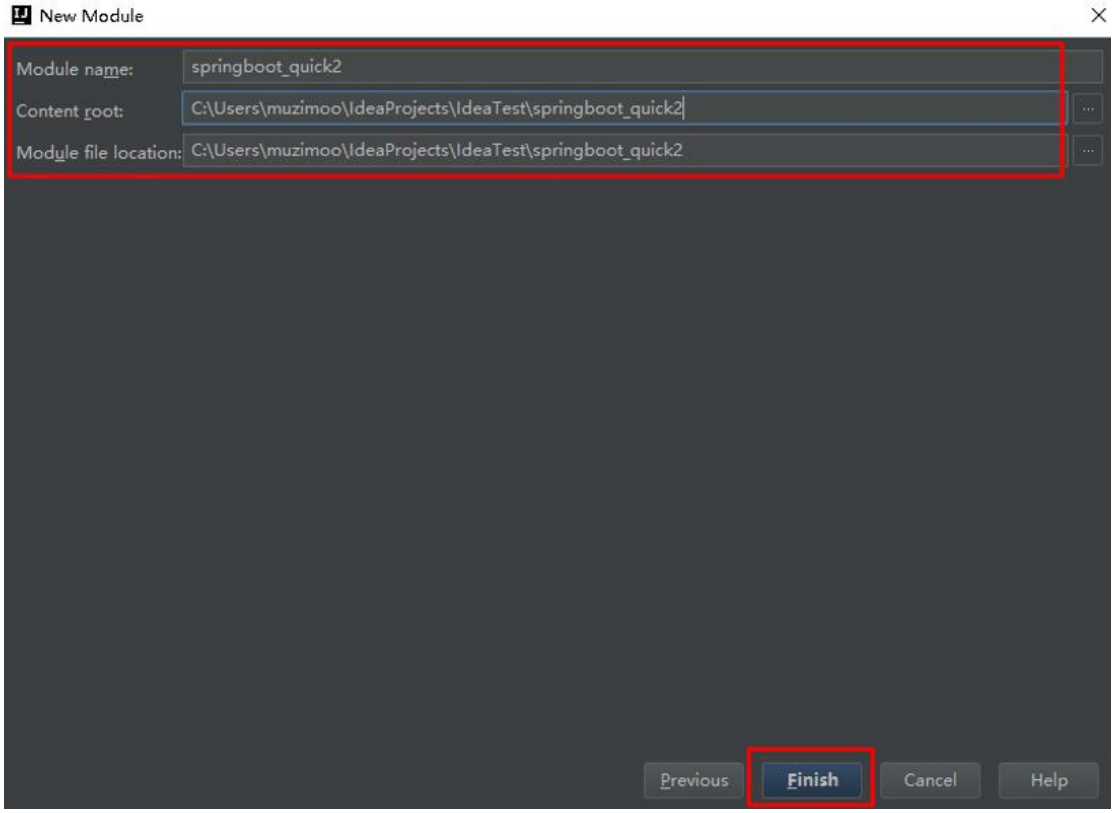


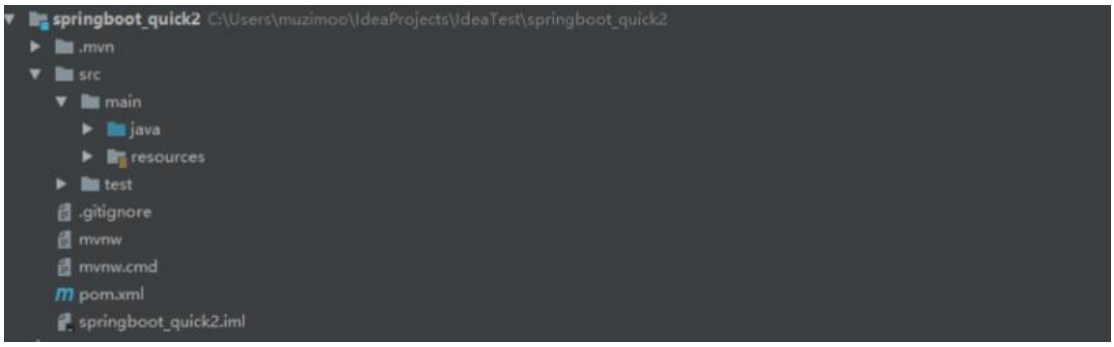
③ 使用 idea 快速创建 SpringBoot 项目






New Module
Dependencies
Spring Boot 2.0.1
Selected Dependencies
Core
Web
Template Engines
SQL
NoSQL
Integration
Cloud Core
Cloud Config
Cloud Discovery
Cloud Routing
Cloud Circuit Breaker
Cloud Tracing
Cloud Messaging
Cloud AWS
Cloud Contract
Pivotal Cloud Foundry
Azure
Spring Cloud GCP
I/O
Ops
Web
Reactive Web
Rest Repositories
Rest Repositories HAL Browser
HATEOAS
Web Services
Jersey (JAX-RS)
Websocket
REST Docs
Vaadin
Apache CXF (JAX-RS)
Ratpack
Mobile
Keycloak
选择SpringBoot提供的启动依赖
Web
Full-stack web development with Tomcat and Spring MVC
Building a RESTful Web Service
https://spring.io/guides/tutorials/bookmarks/
Building REST services with Spring
Reference doc
Previous
Next
Cancel
Help


New Module
Module name:
springboot_quick2
Content root:
C:\Users\muzimoo\IdeaProjects\IdeaTest\springboot_quick2
Module file location:
C:\Users\muzimoo\IdeaProjects\IdeaTest\springboot_quick2
Previous
Finish
Cancel
Help



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

```
<?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>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

三、SpringBoot 原理分析

1. 起步依赖原理分析

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

<build-helper-maven-plugin.version>3.0.0</build-helper-maven-plugin.version>
    <byte-buddy.version>1.7.11</byte-buddy.version>
    ... ... ...
</properties>
 <dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot</artifactId>
            <version>2.0.1.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-test</artifactId>
            <version>2.0.1.RELEASE</version>
        </dependency>
        ... ... ...
```

```
    </dependencies>
</dependencyManagement>
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.jetbrains.kotlin</groupId>
                <artifactId>kotlin-maven-plugin</artifactId>
                <version>${kotlin.version}</version>
            </plugin>
            <plugin>
                <groupId>org.jooq</groupId>
                <artifactId>jooq-codegen-maven</artifactId>
                <version>${jooq.version}</version>
            </plugin>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>2.0.1.RELEASE</version>
            </plugin>
            ... ... ...
        </plugins>
    </pluginManagement>
</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。

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"?>
<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>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.0.1.RELEASE</version>
    <name>Spring Boot Web Starter</name>
      <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
            <version>2.0.1.RELEASE</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-json</artifactId>
            <version>2.0.1.RELEASE</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>2.0.1.RELEASE</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.hibernate.validator</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>6.0.9.Final</version>
```

```
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>5.0.5.RELEASE</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.5.RELEASE</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
</project>
```

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。

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);
    }
}
```

注解@SpringBootApplication 的源码

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
```

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class), @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

    /**
     * Exclude specific auto-configuration classes such that they will never be applied.
     * @return the classes to exclude
     */
    @AliasFor(annotation = EnableAutoConfiguration.class)
    Class<?>[] exclude() default {};

    ... ... ...

}
```

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)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {    ... ... ... }
```

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, attributes);
        configurations = removeDuplicates(configurations);
        Set<String> exclusions = getExclusions(annotationMetadata, attributes);
        checkExcludedClasses(configurations,exclusions);
        configurations.removeAll(exclusions);
        configurations = filter(configurations,
 autoConfigurationMetadata);
        fireAutoConfigurationImportEvents(configurations,exclusions);
        return StringUtils.toStringArray(configurations);
 }


protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,
AnnotationAttributes attributes) {
        List<String>                    configurations                    =
SpringFactoriesLoader.loadFactoryNames(                getSpringFactoriesLoa
derFactoryClass(),getBeanClassLoader());
                return configurations;
 }
```

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：

```
... ... ...

org.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAut
oConf iguration,\
org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfigu
ration ,\
org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAut
oConfigu ration,\
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfigurati
on,\
```

```
org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration,\

... ... ...
```

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)
 @ConditionalOnClass(ServletRequest.class)
 @ConditionalOnWebApplication(type = Type.SERVLET)
 @EnableConfigurationProperties(ServerProperties.class)

@Import({ ServletWebServerFactoryAutoConfiguration.BeanPostProcessorsRegistrar.class, ServletWebServerFactoryConfiguration.EmbeddedTomcat.class, ServletWebServerFactoryConfiguration.EmbeddedJetty.class, ServletWebServerFactoryConfiguration.EmbeddedUndertow.class })
 public class ServletWebServerFactoryAutoConfiguration {
     ... ... ...
 }
```

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)
 public class ServerProperties {

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

     /**
```

```
     * Network address to which the server should bind.
     */
    private InetAddress address;
      ... ... ...
}
```

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

1. SpringBoot 配置文件类型

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。

② application.yml 配置文件

- 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。

- yml 配置文件的语法
  - ➢ 配置普通数据

  语法： key: value

  示例代码：

```
name: haohao
```

  注意：value 之前有一个空格

  - ➢ 配置对象数据

  语法：

  ```
  key:
  key1: value1
  key2: value2
  ```

  或者：

  ```
  key: {key1: value1,key2: value2}
  ```

  示例代码：

```
person:
  name: haohao
  age: 31
  addr: beijing

#或者

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

  - ➢ 配置 Map 数据

  同上面的对象写法

  - ➢ 配置数组（List、Set）数据

  语法：

key:

- value1

- value2 或者：

key: [value1,value2]

示例代码：

```
city:
  - beijing
  - tianjin
  - shanghai
  - chongqing

#或者

city: [beijing,tianjin,shanghai,chongqing]

#集合中的元素是对象形式
student:
  - name: zhangsan
    age: 18
    score: 100
  - name: lisi
    age: 28
    score: 88
  - name: wangwu
    age: 38
    score: 90
```

注意：value1 与之间的 - 之间存在一个空格

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-applicationproperties

常用的配置摘抄如下：

```
# QUARTZ SCHEDULER (QuartzProperties)
 spring.quartz.jdbc.initialize-schema=embedded # Database schema initialization mode.
spring.quartz.jdbc.schema=classpath:org/quartz/impl/jdbcjobstore/tables_@@platform@@.
sql # Path to the SQL file to use to initialize the database schema.

spring.quartz.job-store-type=memory # Quartz job store type.
spring.quartz.properties.*= # Additional Quartz Scheduler properties.

# ----------------------------------------
# WEB PROPERTIES
# ----------------------------------------

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

# HTTP encoding (HttpEncodingProperties)
 spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and responses. Added to the "Content-Type" header if not set explicitly.

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

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

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
```

```
spring.datasource.driver-class-name= # Fully qualified name of the JDBC driver. Autodetected based on the URL by default.
spring.datasource.password= # Login password of the database.
spring.datasource.url= # JDBC URL of the database.
spring.datasource.username= # Login username of the database.

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

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
```

2. 配置文件与配置类的属性映射方式

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
  age: 18
```

或者，application.yml 配置如下：

```
person:
  name: zhangsan
  age: 18
```

实体 Bean 代码如下：

```
@Controller
 public class QuickStartController {

    @Value("${person.name}")
    private String name;
    @Value("${person.age}")
    private Integer age;

    @RequestMapping("/quick")
    @ResponseBody
    public String quick(){
        return "springboot 访问成功! name="+name+",age="+age;
    }
}
```

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

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：

或者，application.yml 配置如下：

```
person:
```

```
  name: zhangsan
  age: 18
```

实体 Bean 代码如下：

```
@Controller @ConfigurationProperties(prefix = "person") public class QuickStartController {

    private String name;    private Integer age;

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

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

    public void setAge(Integer age) { this.age = age; } }
```

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

1. SpringBoot 整合 Mybatis

① 添加 Mybatis 的起步依赖

```
<!--mybatis 起步依赖-->
 <dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
```

```
    <version>1.1.1</version>
</dependency>
```

② 添加数据库驱动坐标

```
<!-- MySQL 连接驱动 -->
 <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
 </dependency>
```

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
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8
 spring.datasource.username=root
spring.datasource.password=root
```

④ 创建 user 表

在 test 数据库中创建 user 表

```
-- ----------------------------
-- Table structure for `user`
 -- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(50) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
```

```
 ) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8;


-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'zhangsan', '123', '张三');
INSERT INTO `user` VALUES ('2', 'lisi', '123', '李四');
```

⑤ 创建实体 Bean

```
public class User {
    // 主键
    private Long id;
    // 用户名
    private String username;
    // 密码
    private String password;
    // 姓名
    private String name;
      //此处省略 getter 和 setter 方法 .. ..
    }
```

⑥ 编写 Mapper

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

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

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">
     <select id="queryUserList" resultType="user">
         select * from user
     </select>
 </mapper>
```

⑧ 在 application.properties 中添加 mybatis 的信息

```
#spring 集成 Mybatis 环境
 #pojo 别名扫描包
 mybatis.type-aliases-package=com.itheima.domain
#加载 Mybatis 映射文件
mybatis.mapper-locations=classpath:mapper/*Mapper.xml
```

⑨ 编写测试 Controller

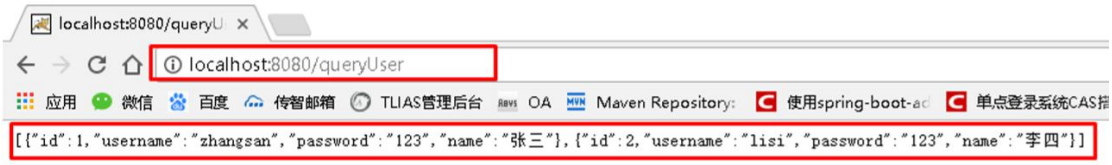
```
@Controller
 public class MapperController {

     @Autowired
     private UserMapper userMapper;

     @RequestMapping("/queryUser")
     @ResponseBody      public List<User> queryUser(){
         List<User> users = userMapper.queryUserList();
         return users;
     }

}
```

10 测试


localhost:8080/queryU
localhost:8080/queryUser
应用
微信
百度
传智邮箱
TLIAS管理后台
OA
Maven Repository:
使用spring-boot-ac
单点登录系统CAS
[{"id":1,"username":"zhangsan","password":"123","name":"张三"},{"id":2,"username":"lisi","password":"123","name":"李四"}]