

# SpringBoot

尚硅谷 Java 研究院

版本: V1.0

## 第一章 分布式架构

### 1.1 流行分布式架构

#### 流行分布式架构

中国互联网BAT公司中使用的分布式架构。



#### Redis

分布式 数据库

一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。



Apache ZooKeeper

#### ZooKeeper

分布式

一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。



#### Dubbo

分布式

阿里巴巴SOA服务化治理方案的核心框架。

## 1.2 Spring 分布式架构

### Spring 分布式架构

Spring框架随着技术的发展，也不断融合了分布式相关的功能模块，如：Spring Data, Spring Cloud等。



## 第二章 SpringBoot 起步

### 2.1 概述

**Spring Boot** 是由 **Pivotal** 团队提供的全新框架，其设计目的是用来简化新 **Spring** 应用的初始搭建以及开发过程。

该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。

通过这种方式，**Spring Boot** 致力于在蓬勃发展的快速应用开发领域(**rapid application development**)成为领导者。

本章内容只是在 **web** 项目中应用基本的 **Spring Boot** 技术，如果想要学习完整 **Spring Boot** 课程内容，请访问 **Spring Boot** 官网进行学习。

此课件基于 **Maven** 创建项目，如果对于 **Maven** 工具不是很了解，请自行学习相关课件

### 2.2 为什么使用 Spring Boot?

说到为什么使用 **Spring Boot**，就不得不提到 **Spring** 框架的**前世今生**

**Spring** 框架由于其繁琐的配置，一度被人认为“配置地狱”，各种 **XML**、**Annotation** 配置混合使用，让人眼花缭乱，而且如果出错了也很难找出原因。

通过 **SpringMVC** 框架部署和发布 **web** 程序，需要和系统外服务器进行关联，操作繁琐不方便。

**Spring Boot** 是由 **Spring** 官方推出的一个新框架，对 **Spring** 进行了高度封装，是 **Spring** 未来的发展方向。使用 **Spring Boot** 框架后，可以帮助开发者快速搭建 **Spring** 框架，也可以帮助开发者快速启动一个 **Web** 服务，无须依赖外部 **Servlet** 容器，使编码变得简单，使配置变得简单，使部署变得简单，使监控变得简单。

## 2.3 Spring 前世今生

### 2.3.1 Spring1.x 时代

在 Spring1.x 时代，都是通过 xml 文件配置 bean  
随着项目的不断扩大，需要将 xml 配置分放到不同的配置文件中  
需要频繁的在 java 类和 xml 配置文件中切换。

### 2.3.2 Spring2.x 时代

随着 **JDK 1.5** 带来的**注解**支持，Spring2.x 可以使用注解对 Bean 进行申明和注入，大大的减少了 xml 配置文件，同时也大大简化了项目的开发。  
那么，问题来了，究竟是应该使用 xml 还是注解呢？

最佳实践：

应用的基本配置用 xml，比如：数据源、资源文件等；  
业务开发用注解，比如：Service 中注入 bean 等；

### 2.3.3 Spring3.x 到 Spring4.x

从 Spring3.x 开始提供了 Java 配置方式，使用 Java 配置方式可以更好的理解你配置的 Bean，现在我们就处于这个时代，并且 Spring4.x 和 Spring boot 都推荐使用 java 配置的方式。

#### Spring 1.X

使用基本的框架类及配置文件（.xml）实现对象的声明及对象关系的整合。

org.springframework.core.io.ClassPathResource

org.springframework.beans.factory.xml.XmlBeanFactory

org.springframework.context.support.ClassPathXmlApplicationContext

#### Spring 2.X

使用注解代替配置文件中对象的声明。简化配置。

org.springframework.stereotype.@Component

org.springframework.stereotype.@Controller

org.springframework.stereotype.@Service

org.springframework.stereotype.@Repository

org.springframework.stereotype.@Scope

org.springframework.beans.factory.annotation.@Autowired

### Spring 3.X

使用更强大的注解完全代替配置文件。

`org.springframework.context.annotation.AnnotationConfigApplicationContext`

`org.springframework.context.annotation.@Configuration`

`org.springframework.context.annotation.@Bean`

`org.springframework.context.annotation.@Value`

`org.springframework.context.annotation.@Import`

### Spring 4.X

使用条件注解强化之前版本的注解。

`org.springframework.context.annotation.@Conditional`

## 2.3.4 Spring 作者

**Rod Johnson** 在 2002 年编著的《Expert one on one J2EE design and development》一书中，对 Java EE 系统框架臃肿、低效、脱离现实的种种现状提出了质疑，并积极寻求探索革新之道。

以此书为指导思想，他编写了 `interface21` 框架，这是一个力图冲破 J2EE 传统开发的困境，从实际需求出发，着眼于轻便、灵巧，易于开发、测试和部署的轻量级开发框架。

Spring 框架即以 `interface21` 框架为基础，经过重新设计，并不断丰富其内涵，于 **2004 年 3 月 24 日**，发布了 1.0 正式版。

同年他又推出了一部堪称经典的力作《Expert one-on-one J2EE Development without EJB》，该书在 Java 世界掀起了轩然大波，不断改变着 Java 开发者程序设计和开发的思考方式。

在该书中，作者根据自己多年丰富的实践经验，对 EJB 的各种笨重臃肿的结构进行了逐一的分析和否定，并分别以简洁实用的方式替换之。

至此一战功成，Rod Johnson 成为一个改变 Java 世界的大师级人物。

## 2.3.5 案例

### ● Spring1.x

```
public static void main(String[] args) {  
    ApplicationContext beanFactory = new ClassPathXmlApplicationContext("beans.xml");  
    System.out.println("ApplicationContext..."); //立即初始化  
    User user = (User)beanFactory.getBean("user");  
    System.out.println(user);  
}
```

```
public static void main(String[] args) {  
    Resource resource = new ClassPathResource("beans.xml");  
    XmlBeanFactory beanFactory = new XmlBeanFactory(resource);  
    System.out.println("XmlBeanFactory..."); //延迟初始化  
    User user = (User)beanFactory.getBean("user");  
}
```

```
System.out.println(user);  
}
```

- **Spring2.x**

@Controller

@Service

@Repository

@Scope

@Component

- **Spring3.x**

```
import org.springframework.context.annotation.Bean; //@since 3.0  
import org.springframework.context.annotation.Configuration; //@since 3.0  
// <bean id="user" class="com.atguigu.bean.User"></bean>
```

**@Configuration**

**public class BeanConfig {**

**@Bean**

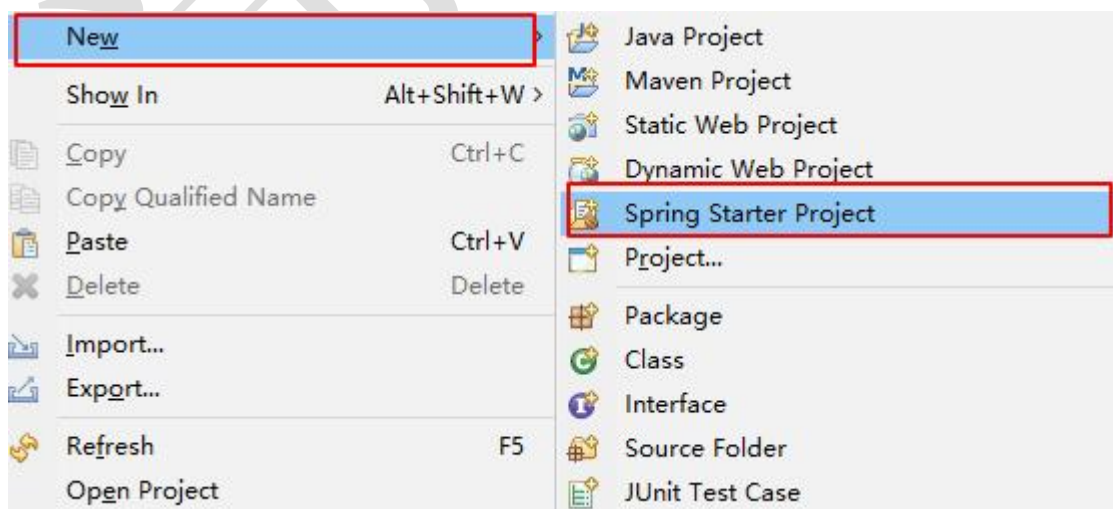
    public User user() { //方法名称作为 bean 的 id  
        return new User();  
    }

**}**

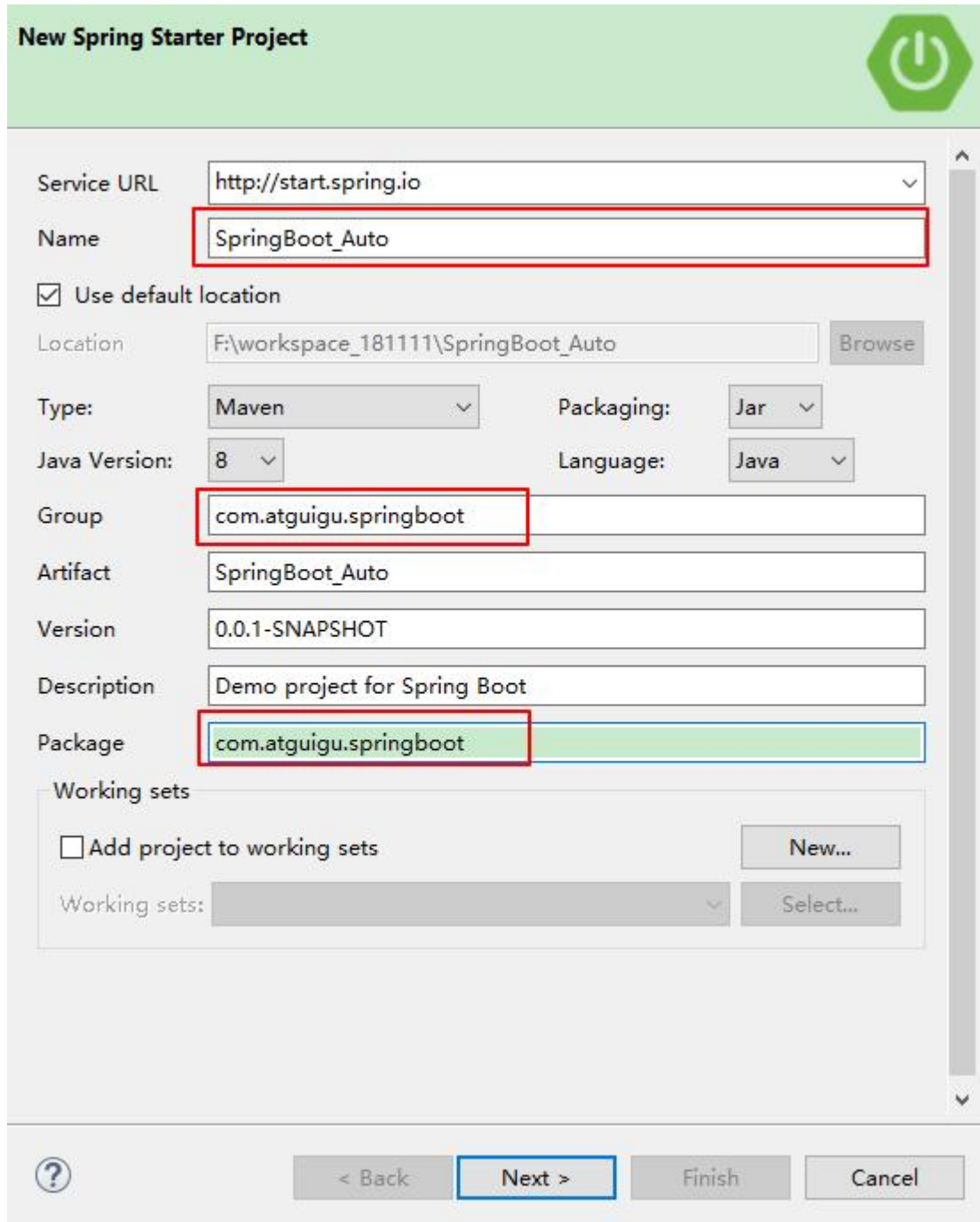
```
ApplicationContext beanFactory = new AnnotationConfigApplicationContext("com.atguigu");
```

## 2.4 自动创建一个 Spring Boot\_HelloWorld（必须联网）

### 2.4.1 在 STS 中右键→Spring Starter Project



## 2.4.2 给工程命名、设置包名等，其他默认即可

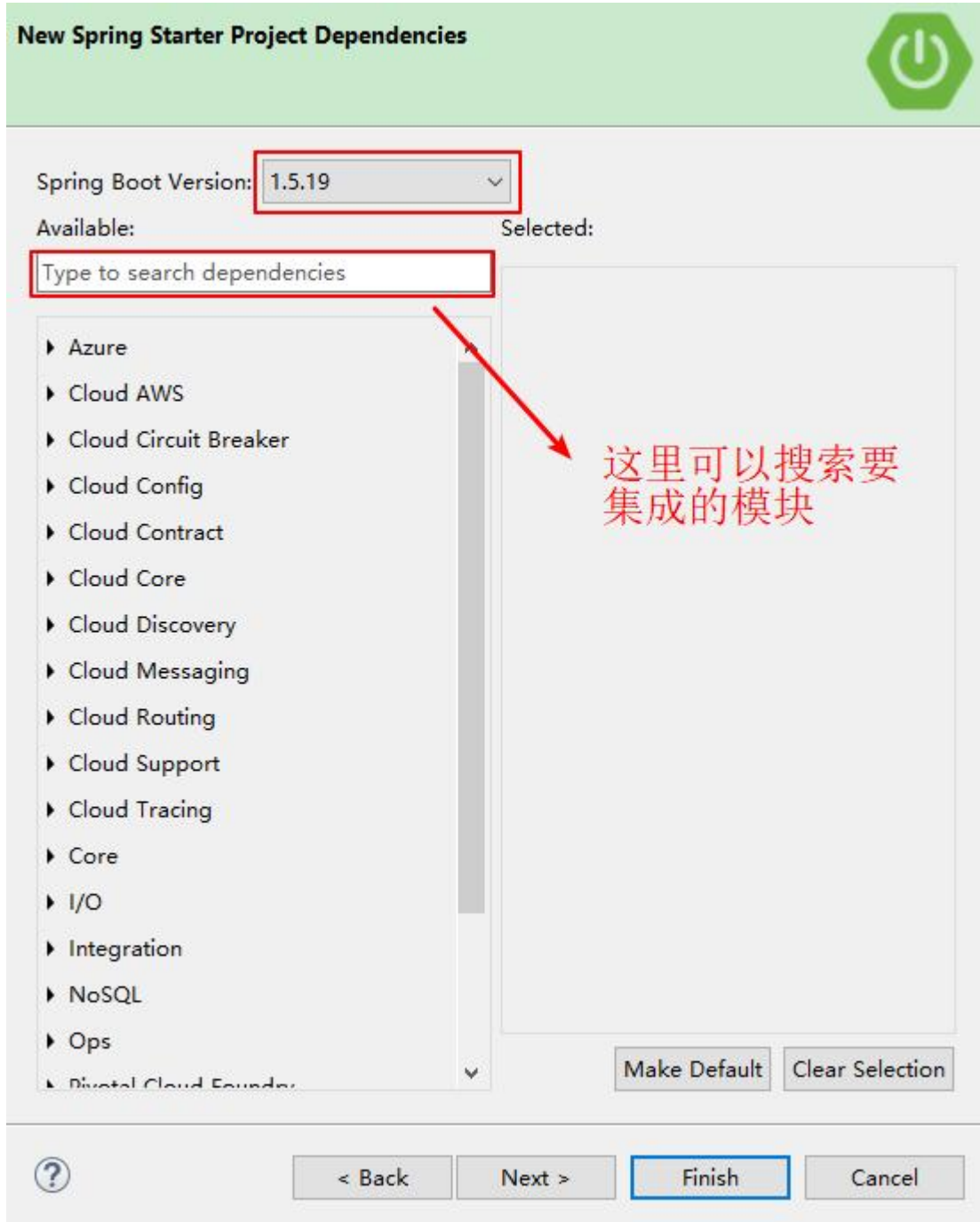


The image shows the 'New Spring Starter Project' dialog box in an IDE. The dialog is titled 'New Spring Starter Project' and has a green power button icon in the top right corner. The fields are as follows:

- Service URL: `http://start.spring.io` (dropdown menu)
- Name: `SpringBoot_Auto` (text field, highlighted with a red box)
- ☒ Use default location
- Location: `F:\workspace_181111\SpringBoot_Auto` (text field, with a 'Browse' button)
- Type: `Maven` (dropdown menu)
- Packaging: `Jar` (dropdown menu)
- Java Version: `8` (dropdown menu)
- Language: `Java` (dropdown menu)
- Group: `com.atguigu.springboot` (text field, highlighted with a red box)
- Artifact: `SpringBoot_Auto` (text field)
- Version: `0.0.1-SNAPSHOT` (text field)
- Description: `Demo project for Spring Boot` (text field)
- Package: `com.atguigu.springboot` (text field, highlighted with a red box)
- Working sets section:
  - ☐ Add project to working sets (with a 'New...' button)
  - Working sets: (dropdown menu, with a 'Select...' button)

At the bottom, there are four buttons: '?', '< Back', 'Next >' (highlighted with a blue box), 'Finish', and 'Cancel'.

### 2.4.3 设置 SpringBoot 的工程为 1.5.X 版本



The image shows the 'New Spring Starter Project Dependencies' dialog box. At the top, there is a green header bar with a power icon. Below the header, the 'Spring Boot Version:' is set to '1.5.19'. Underneath, there are two sections: 'Available:' and 'Selected:'. The 'Available:' section contains a search bar with the placeholder text 'Type to search dependencies'. Below the search bar is a list of dependency categories, including Azure, Cloud AWS, Cloud Circuit Breaker, Cloud Config, Cloud Contract, Cloud Core, Cloud Discovery, Cloud Messaging, Cloud Routing, Cloud Support, Cloud Tracing, Core, I/O, Integration, NoSQL, Ops, and Pivotal Cloud Foundry. A red arrow points from the search bar to the text '这里可以搜索要集成的模块' (Here you can search for modules to integrate). At the bottom right of the 'Available:' section are two buttons: 'Make Default' and 'Clear Selection'. At the very bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

New Spring Starter Project Dependencies

Spring Boot Version: 1.5.19

Available: Selected:

Type to search dependencies

- ▶ Azure
- ▶ Cloud AWS
- ▶ Cloud Circuit Breaker
- ▶ Cloud Config
- ▶ Cloud Contract
- ▶ Cloud Core
- ▶ Cloud Discovery
- ▶ Cloud Messaging
- ▶ Cloud Routing
- ▶ Cloud Support
- ▶ Cloud Tracing
- ▶ Core
- ▶ I/O
- ▶ Integration
- ▶ NoSQL
- ▶ Ops
- ▶ Pivotal Cloud Foundry

这里可以搜索要集成的模块

Make Default Clear Selection

< Back Next > Finish Cancel



## 2.4.4 点击 Next ，所有选项默认即可，点击 Finish

## 2.5 手动创建 Spring Boot\_HelloWorld

### 2.5.1 创建 Maven 项目

- 在 Spring Tool Suite 中创建 Maven jar 项目
- 修改项目中的 pom.xml 文件，设置 JDK 编译版本为 1.8

```
<project>
...
<build>
  <plugins>
    <!-- 修改 maven 默认的 JRE 编译版本，1.8 代表 JRE 编译的版本，根据自己的安装版本
选择 1.7 或 1.8 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
...
</project>
```

- Spring Boot 框架最低 JDK 版本要求 1.6，但是 Spring Boot 官方公布的一些功能使用 1.8 性能会高很多，所以本章我们选用 JDK1.8 版本
- SSM 基础架构中，需要生成 web.xml 文件，Spring Boot 框架中为什么没有？  
Spring Boot 框架开发 web 系统，是基于 servlet3.0 或以上规范，无需 web.xml 文件

### 2.5.2 集成 Spring Boot 框架

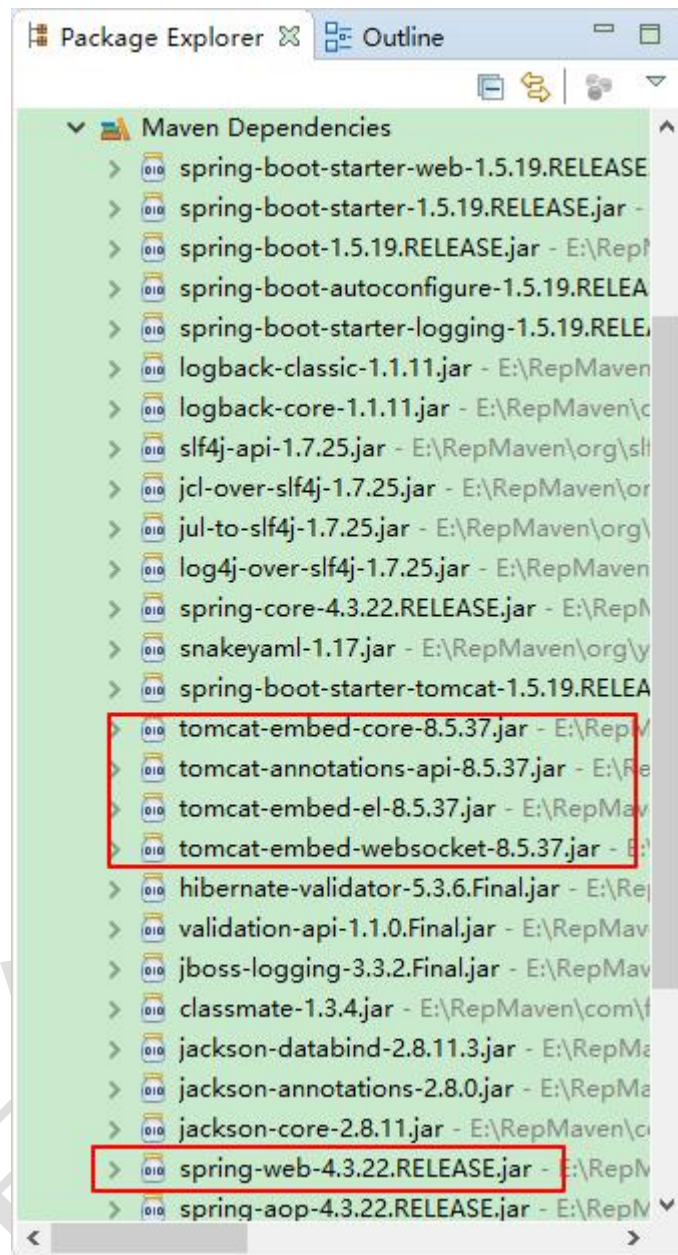
- 修改 pom.xml 文件，增加 Spring Boot 框架的依赖关系及对 Web 环境的支持。

```
<project>
...
<parent>
  <groupId>org.springframework.boot</groupId>
```



```
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.5.19.RELEASE</version>
</parent>
...
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
...
</project>
```

- Spring Boot 版本为官方最新正式版 1.5.19.RELEASE
- 以往的项目中，所有类库的依赖关系都需要我们自己导入到 pom.xml 文件中，但是 Spring Boot 项目增加 spring-boot-starter-web 依赖后，会自动加载 web 环境配置相关依赖(SpringMVC, Tomcat)，简化了我们的操作。
- spring-boot-starter-parent: 继承 Spring Boot 的相关参数
- spring-boot-starter-xxx: 代表一个 Spring Boot 模块(参考附录 1.Spring Boot 相关模块)
- spring-boot-starter-web: 代表 Web 模块，在这个模块中包含了许多依赖的 JAR 包



## 附录 2. 项目报小红叉

### 2.5.3 增加程序代码

- 集成环境,启动服务器
- 在 `src/main/java` 目录中增加类 `com.atguigu.springboot.SpringBootSelfApplication`, 并增加相应代码。

```
package com.atguigu.springboot;
```

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

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

```
@SpringBootApplication
public class SpringBootSelfApplication {

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

- Spring Boot 项目中都会有一个以 Application 结尾的应用类,然后有一个标准的 Java 入口方法 main 方法。通过这个方法启动 Spring Boot 项目,方法中无需放入任何业务逻辑。
- @SpringBootApplication 注解是 Spring Boot 核心注解
- 右键点击项目或项目中的 SpringBootSelfApplication 类,选择菜单 Run as Spring Boot App,控制台出现下面内容表示服务启动成功。

```

 ____  _
/ \  / \  _   _
((  / \  / \  / \  / \
W  / \  / \  / \  / \
'  / \  / \  / \  / \
=====|_|=====|_|/=///
:: Spring Boot ::      (v1.5.19.RELEASE)

2019-01-17 10:44:59.285 INFO 5432 --- [main] c.a.s.SpringBootSelfApplication : Starting
SpringBootSelfApplication on HanYanBing with PID 5432 (F:\workspace_181111\SpringBoot_Self\target\classes started
by HanZong in F:\workspace_181111\SpringBoot_Self)

2019-01-17 10:44:59.285 INFO 5432 --- [main] c.a.s.SpringBootSelfApplication : No active profile
set, falling back to default profiles: default

2019-01-17 10:44:59.335 INFO 5432 --- [main] ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@1d7acb34: startup
date [Thu Jan 17 10:44:59 CST 2019]; root of context hierarchy

2019-01-17 10:45:00.453 INFO 5432 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat
```

initialized with port(s): 8080 (http)

2019-01-17 10:45:00.474 INFO 5432 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2019-01-17 10:45:00.474 INFO 5432 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.37

2019-01-17 10:45:00.593 INFO 5432 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2019-01-17 10:45:00.593 INFO 5432 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1267 ms

2019-01-17 10:45:00.786 INFO 5432 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]

2019-01-17 10:45:00.787 INFO 5432 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [/]

2019-01-17 10:45:00.787 INFO 5432 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/]

2019-01-17 10:45:00.787 INFO 5432 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [/]

2019-01-17 10:45:00.787 INFO 5432 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [/]

2019-01-17 10:45:01.073 INFO 5432 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice:  
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@1d7acb34: startup date [Thu Jan 17 10:44:59 CST 2019]; root of context hierarchy

2019-01-17 10:45:01.134 INFO 5432 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)

2019-01-17 10:45:01.135 INFO 5432 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)

```
2019-01-17 10:45:01.160 INFO 5432 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL
path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-01-17 10:45:01.160 INFO 5432 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL
path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-01-17 10:45:01.194 INFO 5432 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL
path [/**/favicon.ico] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-01-17 10:45:01.320 INFO 5432 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering
beans for JMX exposure on startup

2019-01-17 10:45:01.405 INFO 5432 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started
on port(s): 8080 (http)

2019-01-17 10:45:01.407 INFO 5432 --- [main] c.a.s.SpringBootSelfApplication : Started
SpringBootSelfApplication in 2.331 seconds (JVM running for 3.289)
```

## 2.5.4 集成了 Tomcat 服务器

- 当增加 Web 依赖后执行 main 方法，等同于启动 Tomcat 服务器，默认端口号为 8080。
- 如果想要修改默认的 Tomcat 服务器端口号，可以通过全局配置文件进行配置，在 src/main/resources/ 目录中增加 application.properties（必须叫这个名字）文件。

```
server.context-path=/
server.port=10010
server.session.timeout=60
server.tomcat.max-threads=800
server.tomcat.uri-encoding=UTF-8
```

- Spring Boot 会自动读取 src/main/resources/ 路径或类路径下 /config 路径中的 application.properties 文件或 application.yml 文件。

## 2.5.5 为什么还会有配置文件

Spring Boot 我们称之为微框架，这里的“微”不是小和少的意思，而是“简”的意思，简单，简洁。项目中大部分的基础配置由 Spring Boot 框架帮我们自动集成，简化了我们的配置，但是框架自身为了扩展性，依然需要提供配置文件。

上面的代码中只是简单的应用了 Spring Boot 框架，但是我们真正要做的是将 Spring Boot 应用到项目中，所以接下来我们增加对 SpringMVC 框架，Mybatis 框架的集成。

## 第三章 SpringBoot 项目整合案例

### 3.1 集成 Spring & SpringMVC

- 基本的 Spring Boot 环境已经构建好了，现在需要配置 Spring 框架及 SpringMVC 框架的业务环境

#### 3.1.1 @ComponentScan 注解

//设置自动扫描的包，如果不设置默认扫描当前类所在的包及其子包

```
@ComponentScan(basePackages="com.atguigu.springboot")
```

```
@SpringBootApplication
```

```
public class SpringBootSelfApplication {
```

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

```
        SpringApplication.run(SpringBootSelfApplication.class, args);
```

```
    }
```

```
}
```

#### 3.1.2 默认扫描

默认扫描当前包 com.atguigu.springboot 和子包 com.atguigu.springboot.\*

如果还需要扫描其他的包，那么需要增加@ComponentScan 注解,指定包名进行扫描。

#### 3.1.3 增加控制器代码

在 src/main/java 目录中增加类 com.atguigu.springboot.handler.EmployeeHandler，并增加相应代码。

```
@Controller
```

```
public class EmployeeHandler {
```

```
    @ResponseBody
```

```
    @RequestMapping("/getEmp")
```

```
    public Object getEmployee() {
```

```
        //创建一个 Map
```

```
        Map<String, Object> map = new HashMap<>();
```

```
        //获取一个 Employee 对象，并放到 map 中
```

```
        map.put("emp", new Employee(1, "张三", "zhangsan@atguigu.com", 10000.00, new
```

```
        Department(1001, "开发部"));
```

```
        return map;
    }
}
```

### 3.1.4 执行 main 方法启动应用

- 访问路径 `http://localhost:8080[/应用路径名称]/getEmp` 页面打印 **JSON** 字符串即可



localhost:10010/getEmp

应用 Google W 维基百科, 自由的百科全书 首页 - beego: 简约 & GitHub 首页 - Golang 中国 Go Doc Go Walker

```
{"emp": {"id": 1, "lastName": "张三", "email": "zhangsan@atguigu.com", "salary": 10000.0, "dept": {"id": 1001, "name": "开发部"}}
```

### 3.1.5 @Controller 和 @RestController 区别

官方文档: `@RestController` is a stereotype annotation that combines `@ResponseBody` and `@Controller`. 表示 `@RestController` 等同于 `@Controller + @ResponseBody`, 所以上面的代码可以变为:

**@RestController**

```
public class EmployeeHandler {

    @RequestMapping("/getEmp")
    public Object getEmployee() {
        //创建一个 Map
        Map<String, Object> map = new HashMap<>();
        //获取一个 Employee 对象, 并放到 map 中
        map.put("emp", new Employee(1, "张三", "zhangsan@atguigu.com", 10000.00, new
Department(1001, "开发部")));
        return map;
    }
}
```

### 3.1.6 增加服务层代码

Service 接口, ServiceImpl 实现类的使用和 SSM 架构中的使用方式完全相同。

- Service 接口

```
public interface EmployeeService {

    Employee getEmployeeById(Integer id);
}
```



- Service 接口的实现

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    @Override
    public Employee getEmployeeById(Integer id) {
        //正常情况是调用持久层根据员工的 id 从数据库中查询出对应的员工的信息
        Employee employee = new Employee(1, "张三 2", "zhangsan@atguigu.com", 10000.00, new
        Department(1001, "开发部"));
        return employee;
    }
}
```

- 添加了服务层之后表现层的代码

```
@RestController
public class EmployeeHandler {

    @Autowired
    private EmployeeService employeeService;

    @RequestMapping("/getEmp")
    public Object getEmployee() {
        //创建一个 Map
        Map<String , Object> map = new HashMap<>();
        //调用 EmployeeService 获取 Employee 对象，并放到 map 中
        Employee employeeById = employeeService.getEmployeeById(1);
        map.put("emp", employeeById);
        return map;
    }
}
```

## 附录 1 SpringBoot 相关模块

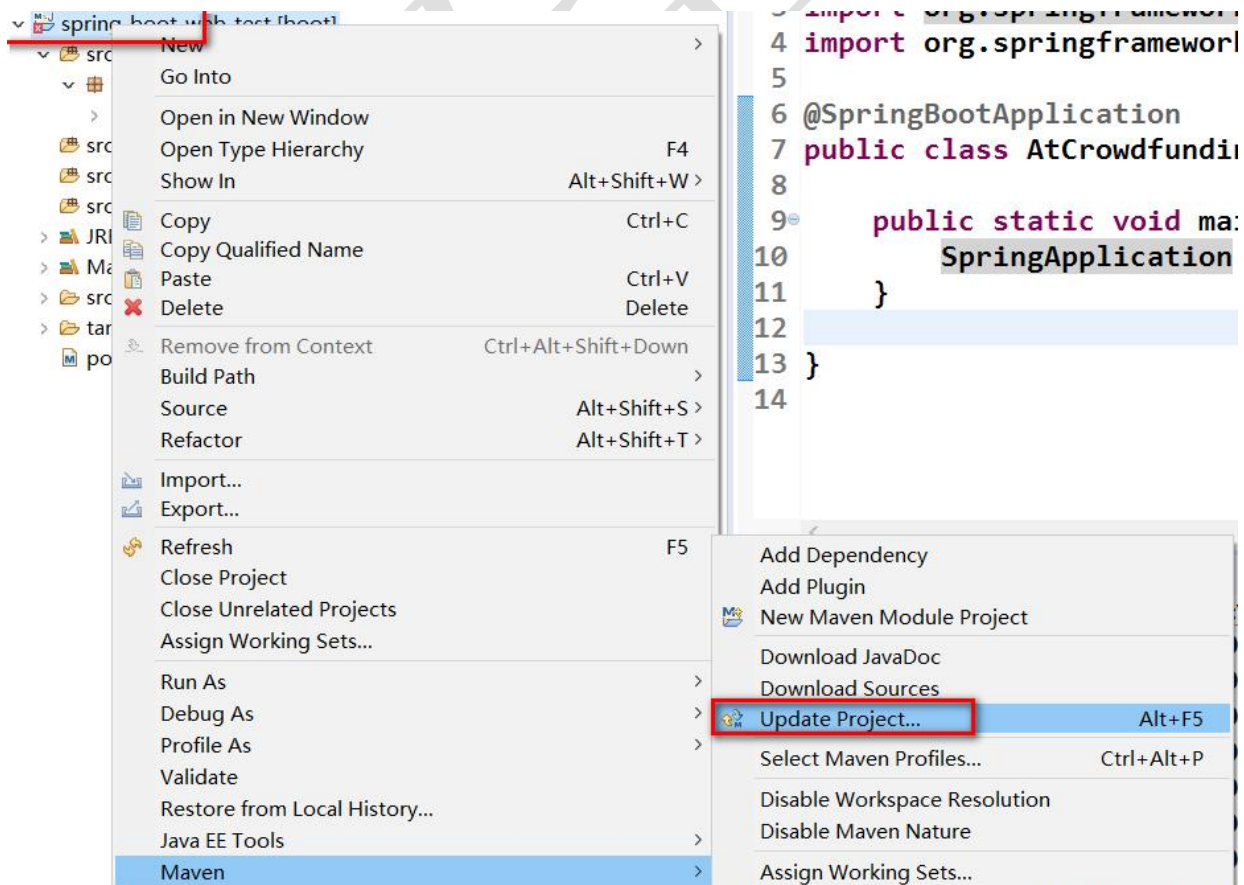
表 6-1 官方提供的 starter pom

名 称	描 述
spring-boot-starter	Spring Boot 核心 starter，包含自动配置、日志、yaml 配置文件的支持
spring-boot-starter-actuator	准生产特性，用来监控和管理应用
spring-boot-starter-remote-shell	提供基于 ssh 协议的监控和管理
spring-boot-starter-amqp	使用 spring-rabbit 来支持 AMQP
spring-boot-starter-aop	使用 spring-aop 和 AspectJ 支持面向切面编程
spring-boot-starter-batch	对 Spring Batch 的支持
spring-boot-starter-cache	对 Spring Cache 抽象的支持
spring-boot-starter-cloud-connectors	对云平台（Cloud Foundry、Heroku）提供的服务提供简化的连接方式

名 称	描 述
spring-boot-starter-data-elasticsearch	通过 spring-data-elasticsearch 对 Elasticsearch 支持
spring-boot-starter-data-gemfire	通过 spring-data-gemfire 对分布式存储 GemFire 的支持
spring-boot-starter-data-jpa	对 JPA 的支持，包含 spring-data-jpa、spring-orm 和 Hibernate
spring-boot-starter-data-mongodb	通过 spring-data-mongodb，对 MongoDB 进行支持
spring-boot-starter-data-rest	通过 spring-data-rest-webmvc 将 Spring Data repository 暴露为 REST 形式的服务
spring-boot-starter-data-solr	通过 spring-data-solr 对 Apache Solr 数据检索平台的支持
spring-boot-starter-freemarker	对 FreeMarker 模板引擎的支持
spring-boot-starter-groovy-templates	对 Groovy 模板引擎的支持
spring-boot-starter-hateoas	通过 spring-hateoas 对基于 HATEOAS 的 REST 形式的网络服务的支持
spring-boot-starter-hornetq	通过 HornetQ 对 JMS 的支持
spring-boot-starter-integration	对系统集成框架 spring-integration 的支持
spring-boot-starter-jdbc	对 JDBC 数据库的支持
spring-boot-starter-jersey	对 Jersey REST 形式的网络服务的支持
spring-boot-starter-jta-atomikos	通过 Atomikos 对分布式事务的支持
spring-boot-starter-jta-bitronix	通过 Bitronix 对分布式事务的支持
spring-boot-starter-mail	对 javax.mail 的支持
spring-boot-starter-mobile	对 spring-mobile 的支持
spring-boot-starter-mustache	对 Mustache 模板引擎的支持
spring-boot-starter-redis	对键值对内存数据库 Redis 的支持，包含 spring-redis

spring-boot-starter-security	对 spring-security 的支持
spring-boot-starter-social-facebook	通过 spring-social-facebook 对 Facebook 的支持
spring-boot-starter-social-linkedin	通过 spring-social-linkedin 对 LinkedIn 的支持
spring-boot-starter-social-twitter	通过 spring-social-twitter 对 Twitter 的支持
spring-boot-starter-test	对常用的测试框架 JUnit、Hamcrest 和 Mockito 的支持，包含 spring-test 模块
spring-boot-starter-thymeleaf	对 Thymeleaf 模板引擎的支持，包含于 Spring 整合的配置
spring-boot-starter-velocity	对 Velocity 模板引擎的支持
spring-boot-starter-web	对 Web 项目开发的支持，包含 Tomcat 和 spring-webmvc
spring-boot-starter-Tomcat	Spring Boot 默认的 Servlet 容器 Tomcat
spring-boot-starter-Jetty	使用 Jetty 作为 Servlet 容器替换 Tomcat
spring-boot-starter-undertow	使用 Undertow 作为 Servlet 容器替换 Tomcat
spring-boot-starter-logging	Spring Boot 默认的日志框架 Logback
spring-boot-starter-log4j	支持使用 Log4J 日志框架
spring-boot-starter-websocket	对 WebSocket 开发的支持
spring-boot-starter-ws	对 Spring Web Services 的支持

## 附录 2 项目报小红叉



```

1  import org.springframework.boot.SpringApplication;
2  import org.springframework.boot.autoconfigure.SpringBootApplication;
3
4  import org.springframework.web.bind.annotation.RequestMapping;
5
6  @SpringBootApplication
7  public class AtCrowdfundin
8
9      public static void main(String[] args) {
10      SpringApplication.run(AtCrowdfundin.class, args);
11  }
12
13 }
14

```

More Java - Big Data - Frontend - python Artificial Intelligence resources download, can be accessed via Baidu: 尚硅谷官网

## 附录 3 yml 详细配置

## 附录 4 相关注解

### 4.1 关于配置

- `@ImportResource`

导入 XML 文件.

```
@ImportResource({"classpath:some-context.xml","classpath:other-context.xml"})
```

- `@Configuration`

`@Configuration`

```
public class MyWebApplicationConfig extends WebMvcConfigurerAdapter{  
    @Override  
    public void addInterceptors(InterceptorRegistry registry) {  
        registry.addInterceptor(new MyInterceptor()).addPathPatterns("/**");  
        super.addInterceptors(registry);  
    }  
}
```

### 4.2 java -jar xx.jar --server.port=9090

### 4.3 常规属性配置

application.properties

`book.author=zhangsan`

`book.price=22.0`

在类的成员变量上使用 `@Value` 注解

```
@Value("${book.author}")
```

```
private String bookAuthor
```

采用 `@ConfigurationProperties` 自动注入

`org.springframework.boot.context.properties.ConfigurationProperties`

```
@Component
```

```
@ConfigurationProperties(prefix="book",locations="classpath:config/application.properties",ignoreUnknownFields = true)
```

```
public class Book{  
    private String author ;
```



```
private double price ;  
private String email ;  
// Get/set...  
}
```

## 4.4 日志配置

Spring boot 使用 Logback 作为日志框架.

logging.file=D:/temp/log.log

logging.level.org.springframework.web=DEBUG

## 4.5 自动扫描

@EnableAutoConfiguration 默认扫描 main 类所在的包及子包.

## 4.6 条件注解

@ConditionalOnBean: 当容器里有指定的 Bean 的条件下。

@ConditionalOnClass: 当类路径下有指定的类的条件下。

@ConditionalOnExpression: 基于 SpEL 表达式作为判断条件。

@ConditionalOnJava: 基于 JVM 版本作为判断条件。

@ConditionalOnJndi: 在 JNDI 存在的条件下查找指定的位置。

@ConditionalOnMissingBean: 当容器里没有指定 Bean 的情况下。

@ConditionalOnMissingClass: 当类路径下没有指定的类的条件下。

@ConditionalOnNotWebApplication: 当前项目不是 Web 项目的条件下。

@ConditionalOnProperty: 指定的属性是否有指定的值。

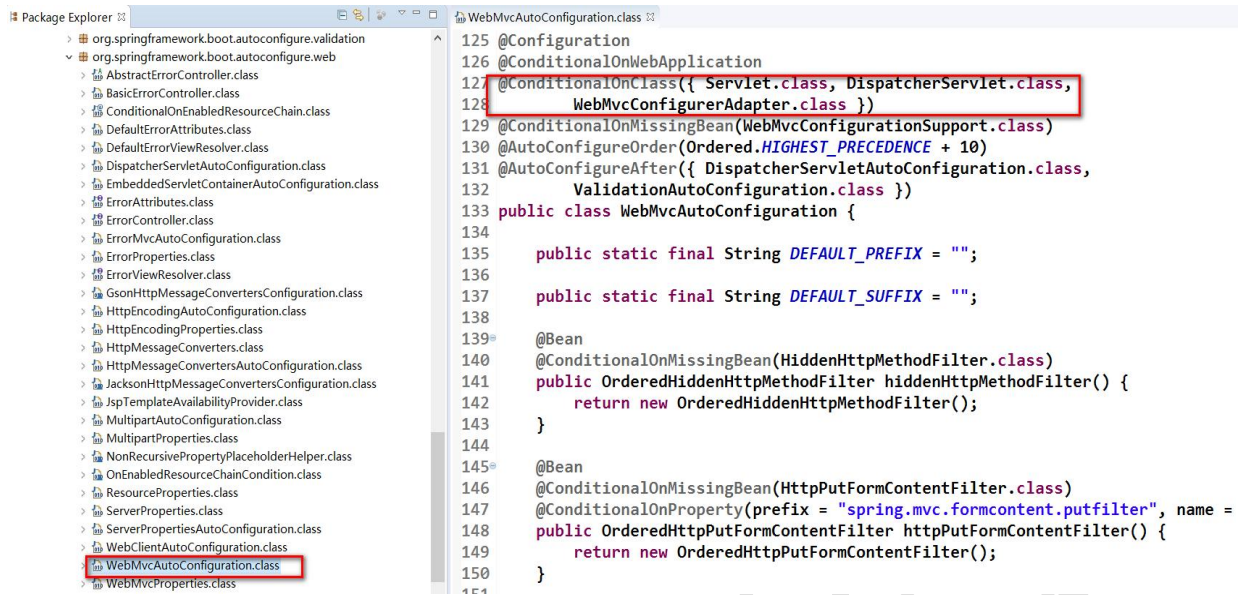
@ConditionalOnResource: 类路径是否有指定的值。

@ConditionalOnSingleCandidate: 当指定 Bean 在容器中只有一个，或者虽然有多个但是指定首选的 Bean。

@ConditionalOnWebApplication: 当前项目是 Web 项目的条件下。

```
spring-boot-autoconfigure-1.5.7.RELEASE.jar - D:\RepMaven\org\springfra
> org.springframework.boot.autoconfigure
> org.springframework.boot.autoconfigure.admin
> org.springframework.boot.autoconfigure.amqp
> org.springframework.boot.autoconfigure.aop
> org.springframework.boot.autoconfigure.batch
> org.springframework.boot.autoconfigure.cache
> org.springframework.boot.autoconfigure.cassandra
> org.springframework.boot.autoconfigure.cloud
v org.springframework.boot.autoconfigure.condition
  > AbstractNestedCondition.class
  > AllNestedConditions.class
  > AnyNestedCondition.class
  > BeanTypeRegistry.class
  > ConditionalOnBean.class
  > ConditionalOnClass.class
  > ConditionalOnCloudPlatform.class
  > ConditionalOnExpression.class
  > ConditionalOnJava.class
  > ConditionalOnJndi.class
  > ConditionalOnMissingBean.class
  > ConditionalOnMissingClass.class
  > ConditionalOnNotWebApplication.class
  > ConditionalOnProperty.class
  > ConditionalOnResource.class
  > ConditionalOnSingleCandidate.class
  > ConditionalOnWebApplication.class
  > ConditionEvaluationReport.class
  > ConditionEvaluationReportAutoConfigurationImportListener.class
  > ConditionMessage.class
  > ConditionOutcome.class
  > NoneNestedConditions.class
  > OnBeanCondition.class
  > OnClassCondition.class
  > OnCloudPlatformCondition.class
  > OnExpressionCondition.class
  > OnJavaCondition.class
  > OnJndiCondition.class
```

- 案例:  
是否自动加载 SpringMVC 相关配置.



## 4.7 @RestController

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    String value() default "";
}
```

## 4.8 修改 Favicon(偏爱图标; 网站图标)配置

Spring.mvc.favicon.enabled=false 默认 true 开启.

在 META-INF/resources/下,类路径 resources/下,类路径 static/下或类路径 public 下,增加 favicon.ico 文件  
就可以修改掉默认图标



## 附录 5 字符编码

### 5.1 字符编码过滤器

#### 5.1.1 之前配置

<!-- 配置字符集 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<!--Spring 里的字符过滤器 CharacterEncodingFilter 是针对请求的, forceEncoding=true 意思是指无论客户端请求是否包含了编码, 都用过滤器里的编码来解析请求 -->

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

#### 5.1.2 现在配置

- 默认配置

```
# default UTF-8 and true
spring.http.encoding.charset=UTF-8
spring.http.encoding.force=true
```

- [org.springframework.boot.autoconfigure.web.HttpEncodingProperties](#)
- [org.springframework.boot.autoconfigure.web.HttpEncodingAutoConfiguration](#)

## 作业

1. 练习联网创建 springboot 项目,以及手动创建 springboot 项目
2. 完成 SpringBoot 项目与 Spring, SpringMVC 集成开发