

精进-Spring-Boot-面试题.md

核心技术篇

Spring Boot 是什么？

Spring Boot 提供了哪些核心功能？

- 1、独立运行 Spring 项目
- 2、内嵌 Servlet 容器
- 3、提供 Starter 简化 Maven 配置
- 4、自动配置 Spring Bean
- 5、准生产的应用监控
- 6、无代码生成和 XML 配置

Spring Boot 有什么优缺点？

Spring Boot、Spring MVC 和 Spring 有什么区别？

Spring Boot 中的 Starter 是什么？

Spring Boot 常用的 Starter 有哪些？

创建一个 Spring Boot Project 的最简单的方法是什么？

如何统一引入 Spring Boot 版本？

运行 Spring Boot 有哪几种方式？

如何打包 Spring Boot 项目？

如果更改内嵌 Tomcat 的端口？

如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？

Spring Boot 的配置文件有哪几种格式？

Spring Boot 默认配置文件是什么？

Spring Boot 如何定义多套不同环境配置？

Spring Boot 有哪些配置方式？

- 1、XML 配置文件。
- 2、注解配置。
- 3、Java Config 配置。

Spring Boot 的核心注解是哪个？

什么是 Spring Boot 自动配置？

Spring Boot 有哪几种读取配置的方式？

- 1、`@Value` 注解，读取配置到属性。最最最常用。
- 2、`@ConfigurationProperties` 注解，读取配置到类上。

使用 Spring Boot 后，项目结构是怎么样的呢？

如何在 Spring Boot 启动的时候运行一些特殊的代码？

Spring Boot 2.X 有什么新特性？

整合篇

如何将内嵌服务器换成 Jetty？

Spring Boot 中的监视器 Actuator 是什么？

如何集成 Spring Boot 和 Spring MVC？

如何集成 Spring Boot 和 Spring Security？

如何集成 Spring Boot 和 Spring Security OAuth2？

如何集成 Spring Boot 和 JPA？

如何集成 Spring Boot 和 MyBatis？

如何集成 Spring Boot 和 RabbitMQ？

如何集成 Spring Boot 和 Kafka？

如何集成 Spring Boot 和 RocketMQ？

Spring Boot 支持哪些日志框架？

精进-Spring-Boot-面试题.md

题目的难度，尽量按照从容易到困难的顺序，逐步下去。

在内容上，我们会分成两大块：

- 核心技术篇，分享 Spring Boot 的核心技术相关的内容。
 - 整合篇，分享 Spring Boot 整合一些框架的面试题，例如 JPA 如何集成到 Spring Boot 中。
-

核心技术篇

Spring Boot 是什么？

[Spring Boot](#) 是 Spring 的子项目，正如其名字，提供 Spring 的引导(**Boot**)的功能。

通过 Spring Boot，我们开发者可以快速配置 Spring 项目，引入各种 Spring MVC、Spring Transaction、Spring AOP、MyBatis 等等框架，而无需不断重复编写繁重的 Spring 配置，降低了 Spring 的使用成本。

犹记当年，Spring XML 为主的时代，大晚上各种搜索 Spring 的配置，苦不堪言。现在有了 Spring Boot 之后，生活真美好。

Spring Boot 提供了各种 Starter 启动器，提供标准化的默认配置。例如：

- [spring-boot-starter-web](#) 启动器，可以快速配置 Spring MVC。
- [mybatis-spring-boot-starter](#) 启动器，可以快速配置 MyBatis。

并且，Spring Boot 基本已经一统 Java 项目的开发，大量的开源项目都实现了其的 Starter 启动器。例如：

- [incubator-dubbo-spring-boot-project](#) 启动器，可以快速配置 Dubbo。
- [rocketmq-spring-boot-starter](#) 启动器，可以快速配置 RocketMQ。

Spring Boot 提供了哪些核心功能？

1、独立运行 Spring 项目

Spring Boot 可以以 jar 包形式独立运行，运行一个 Spring Boot 项目只需要通过 `java -jar xx.jar` 来运行。

2、内嵌 Servlet 容器

Spring Boot 可以选择内嵌 Tomcat、Jetty 或者 Undertow，这样我们无须以 war 包形式部署项目。

第 2 点是对第 1 点的补充，在 Spring Boot 未出来的时候，大多数 Web 项目，是打包成 war 包，部署到 Tomcat、Jetty 等容器。

3、提供 Starter 简化 Maven 配置

Spring 提供了一系列的 starter pom 来简化 Maven 的依赖加载。例如，当你使用了 `spring-boot-starter-web`，会自动加入如下依赖：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <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">
3   <modelVersion>4.0.0</modelVersion>
4   <parent>
5     <groupId>org.springframework.boot</groupId>
6     <artifactId>spring-boot-starters</artifactId>
7     <version>1.5.4.RELEASE</version>
8   </parent>
9   <artifactId>spring-boot-starter-web</artifactId>
10  <name>Spring Boot Web Starter</name>
11  <description>Starter for building web, including RESTful, applications using Spring
12    MVC. Uses Tomcat as the default embedded container</description>
13  <url>http://projects.spring.io/spring-boot/</url>
14  <organization>
15    <name>Pivotal Software, Inc.</name>
16    <url>http://www.spring.io</url>
17  </organization>
18  <properties>
19    <main.basedir>${basedir}/../..</main.basedir>
20  </properties>
21  <dependencies>
22    <dependency>
23      <groupId>org.springframework.boot</groupId>
24      <artifactId>spring-boot-starter</artifactId>
25    </dependency>
26    <dependency>
27      <groupId>org.springframework.boot</groupId>
28      <artifactId>spring-boot-starter-tomcat</artifactId> 带内嵌的 Tomcat
29    </dependency>
30    <dependency>
31      <groupId>org.hibernate</groupId>
32      <artifactId>hibernate-validator</artifactId> 带来参数校验的依赖
33    </dependency>
34    <dependency>
35      <groupId>com.fasterxml.jackson.core</groupId>
36      <artifactId>jackson-databind</artifactId> 带来 JSON 数据的转换器
37    </dependency>
38    <dependency>
39      <groupId>org.springframework</groupId>
40      <artifactId>spring-web</artifactId>
41    </dependency>
42    <dependency>
43      <groupId>org.springframework</groupId>
44      <artifactId>spring-webmvc</artifactId> 带来 Spring MVC 框架
45    </dependency>
46  </dependencies>
47 </project>
```

4、自动配置 Spring Bean

Spring Boot 检测到特定类的存在，就会针对这个应用做一定的配置，进行自动配置 Bean，这样会极大地减少我们要使用的配置。

当然，Spring Boot 只考虑大多数的开发场景，并不是所有的场景，若在实际开发中我们需要配置 Bean，而 Spring Boot 没有提供支持，则可以自定义自动配置进行解决。

5、准生产的应用监控

Spring Boot 提供基于 HTTP、JMX、SSH 对运行时的项目进行监控。

6、无代码生成和 XML 配置

Spring Boot 没有引入任何形式的代码生成，它是使用的 Spring 4.0 的条件 `@Condition` 注解以实现根据条件进行配置。同时使用了 Maven / Gradle 的依赖传递解析机制来实现 Spring 应用里面的自动配置。

第 6 点是第 3 点的补充。

Spring Boot 有什么优缺点？

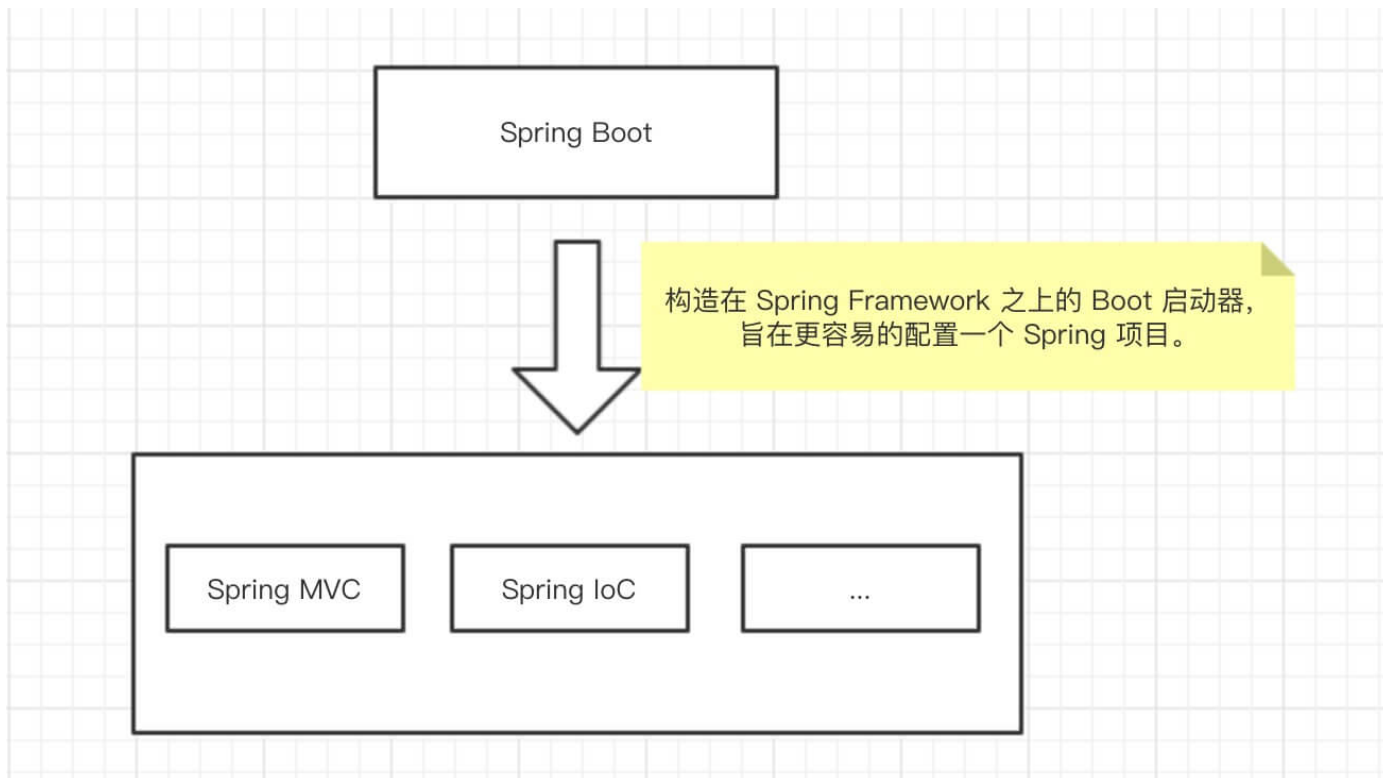
任何技术栈，有优点必有缺点，没有银弹。

Spring Boot、Spring MVC 和 Spring 有什么区别？

Spring 的完整名字，应该是 Spring Framework。它提供了多个模块，Spring IoC、Spring AOP、Spring MVC 等等。所以，Spring MVC 是 Spring Framework 众多模块中的一个。

而 Spring Boot 是构造在 Spring Framework 之上的 Boot 启动器，旨在更容易的配置一个 Spring 项目。

总结说来，如下图所示：



Spring Boot 中的 Starter 是什么？

比较通俗的说法：

FROM [《Spring Boot 中 Starter 是什么》](#)

比如我们要在 Spring Boot 中引入 Web MVC 的支持时，我们通常会引入这个模块 `spring-boot-starter-web`，而这个模块如果解压包出来会发现里面什么都没有，只定义了一些 **POM** 依赖。如下图所示：

创建一个 Spring Boot Project 的最简单的方法是什么？

Spring Initializr 是创建 Spring Boot Projects 的一个很好的工具。打开 `"https://start.spring.io/"` 网站，我们可以看到 Spring Initializr 工具，如下图所示：

Generate a Maven Project with Java and Spring Boot 2.1.1

Project Metadata
Artifact coordinates

Group
com.example

Artifact
demo

Dependencies
Add Spring Boot Starters and dependencies to your application

Search for dependencies
Web, Security, JPA, Actuator, DevTools...

Selected Dependencies
Web × Actuator × DevTools ×

Generate Project

Don't know what to look for? Want more options? [Switch to the full version](#)

- 图中的每一个红线，都可以填写相应的配置。相信胖友都很熟悉，就不哔哔了。
- 点击生 GenerateProject ，生成 Spring Boot Project 。
- 将项目导入 IDEA ，记得选择现有的 Maven 项目。

当然，我们以前使用 IDEA 创建 Spring 项目的方式，也一样能创建 Spring Boot Project 。Spring Initializr 更多的是，提供一个便捷的工具。

如何统一引入 Spring Boot 版本？

目前有两种方式。

① 方式一：继承 `spring-boot-starter-parent` 项目。配置代码如下：

```
1 <parent>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-parent</artifactId>
4   <version>1.5.1.RELEASE</version>
5 </parent>
```

② 方式二：导入 `spring-boot-dependencies` 项目依赖。配置代码如下：

```
1 <dependencyManagement>
2   <dependencies>
3     <dependency>
4       <groupId>org.springframework.boot</groupId>
5       <artifactId>spring-boot-dependencies</artifactId>
6       <version>1.5.1.RELEASE</version>
7       <type>pom</type>
8       <scope>import</scope>
9     </dependency>
10  </dependencies>
11 </dependencyManagement>
```

如何选择？

因为一般我们的项目中，都有项目自己的 Maven parent 项目，所以【方式一】显然会存在冲突。所以实际场景下，推荐使用【方式二】。

详细的，推荐阅读 [《Spring Boot 不使用默认的 parent，改用自己的项目的 parent》](#) 文章。

另外，在使用 Spring Cloud 的时候，也可以使用这样的方式。

运行 Spring Boot 有哪几种方式？

- 1、打包成 Fat Jar，直接使用 `java -jar` 运行。目前主流的做法，推荐。
- 2、在 IDEA 或 Eclipse 中，直接运行应用的 Spring Boot 启动类的 `#main(String[] args)` 启动。适用于开发调试场景。
- 3、如果是 Web 项目，可以打包成 War 包，使用外部 Tomcat 或 Jetty 等容器。

如何打包 Spring Boot 项目？

通过引入 `spring-boot-maven-plugin` 插件，执行 `mvn clean package` 命令，将 Spring Boot 项目打成一个 Fat Jar。后续，我们就可以直接使用 `java -jar` 运行。

关于 `spring-boot-maven-plugin` 插件，更多详细的可以看看 [《创建可执行 jar》](#)。

如果更改内嵌 Tomcat 的端口？

- 方式一，修改 `application.properties` 配置文件的 `server.port` 属性。

```
1 server.port=9090
```

- 方式二，通过启动命令增加 `server.port` 参数进行修改。

```
1 java -jar xxx.jar --server.port=9090
```


当然，以上的方式，不仅仅适用于 Tomcat，也适用于 Jetty、Undertow 等服务器。

如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？

一共有三种方式，可以实现效果：

- 【推荐】`spring-boot-devtools` 插件。注意，这个工具需要配置 IDEA 的自动编译。
- Spring Loaded 插件。

Spring Boot 2.X 后，官方宣布不再支持 Spring Loaded 插件 的更新，所以基本可以无视它了。

- [JRebel](#) 插件，需要付费。

关于如何使用 `spring-boot-devtools` 和 Spring Loaded 插件，胖友可以看看 [《Spring Boot 学习笔记：Spring Boot Developer Tools 与热部署》](#)。

Spring Boot 的配置文件有哪几种格式？

Spring Boot 目前支持两种格式的配置文件：

- `.properties` 格式。示例如下：

```
1 server.port = 9090
```

- `.yaml` 格式。示例如下：

```
1 server:
2   port: 9090
```

可能有胖友不了解 **YAML 格式**？

YAML 是一种人类可读的数据序列化语言，它通常用于配置文件。

- 与 Properties 文件相比，如果我们想要在配置文件中添加复杂的属性 YAML 文件就更加**结构化**。从上面的示例，我们可以看出 YAML 具有**分层**配置数据。
- 当然 YAML 在 Spring 会存在一个缺陷，`@PropertySource` 注解不支持读取 YAML 配置文件，仅支持 Properties 配置文件。
 - 不过这个问题也不大，可以麻烦一点使用 `@Value` 注解，来读取 YAML 配置项。

实际场景下，芳芳相对比较喜欢使用 Properties 配置文件。个人喜欢~当然，YAML 已经越来越流行了。

Spring Boot 默认配置文件是什么？

对于 Spring Boot 应用，默认的配置文件是根目录下的 **application** 配置文件，当然可以是 Properties 格式，也可以是 YAML 格式。

可能有胖友说，我在网上看到面试题中，说还有一个根目录下的 **bootstrap** 配置文件。这个是 Spring Cloud 新增的启动配置文件，需要引入 `spring-cloud-context` 依赖后，才会进行加载。它的特点和用途主要是：

参考 [《Spring Cloud 中配置文件名 bootstrap.yml 和 application.yml 区别》](#) 文章。

- 【特点】因为 bootstrap 由父 ApplicationContext 加载，比 application 优先加载。
- 【特点】因为 bootstrap 优先于 application 加载，所以不会被它覆盖。
- 【用途】使用配置中心 Spring Cloud Config 时，需要在 bootstrap 中配置配置中心的地址，从而实现父 ApplicationContext 加载时，从配置中心拉取相应的配置到应用中。

另外，[《Appendix A. Common application properties》](#) 中，有 application 配置文件的通用属性列表。

Spring Boot 如何定义多套不同环境配置？

在 Spring Boot 中，除了我们常用的 application 配置文件之外，还有：

- 系统环境变量
- 命令行参数
- 等等...

参考 [《Externalized Configuration》](#) 文档，我们整理顺序如下：

1. `spring-boot-devtools` 依赖的 `spring-boot-devtools.properties` 配置文件。这个灰常小众，具体说明可以看看 [《Spring Boot参考文档_\(12\)_开发者工具》](#)，建议无视。
2. 单元测试上的 `@TestPropertySource` 和 `@SpringBootTest` 注解指定的参数。前者的优先级高于后者。可以看看 [《Spring、Spring Boot 和TestNG 测试指南 - @TestPropertySource》](#) 一文。
3. 命令行指定的参数。例如 `java -jar springboot.jar --server.port=9090`。
4. 命令行中的 `spring.application.json` 指定参数。例如 `java -Dspring.application.json='{ "name": "Java" }' -jar springboot.jar`。
5. ServletConfig 初始化参数。
6. ServletContext 初始化参数。
7. JNDI 参数。例如 `java:comp/env`。
8. Java 系统变量，即 `System#getProperties()` 方法对应的。
9. 操作系统环境变量。
10. RandomValuePropertySource 配置的 `random.*` 属性对应的值。
11. Jar 外部的带指定 profile 的 application 配置文件。例如 `application-{profile}.yaml`。
12. Jar 内部的带指定 profile 的 application 配置文件。例如 `application-{profile}.yaml`。
13. Jar 外部 application 配置文件。例如 `application.yaml`。
14. Jar 内部 application 配置文件。例如 `application.yaml`。
15. 在自定义的 `@Configuration` 类中定于的 `@PropertySource`。
16. 启动的 `main` 方法中，定义的默认配置。即通过 `SpringApplication#setDefaultProperties(Map<String, Object> defaultProperties)` 方法进行设置。

嘿嘿，是不是很多很长，不用真的去记住。

- 一般来说，面试官不会因为这个问题回答的不好，对你扣分。
- 实际使用时，做下测试即可。
- 每一种配置方式的详细说明，可以看看 [《Spring Boot 参考指南（外部化配置）》](#)。

Spring Boot 有哪些配置方式？

和 Spring 一样，一共提供了三种方式。

1、XML 配置文件。

Bean 所需的依赖项和服务在 XML 格式的配置文件中指定。这些配置文件通常包含许多 bean 定义和特定于应用程序的配置选项。它们通常以 bean 标签开头。例如：

```
1 <bean id="studentBean" class="org.edureka.firstSpring.StudentBean">
2     <property name="name" value="Edureka"></property>
3 </bean>
```

2、注解配置。

您可以通过在相关的类，方法或字段声明上使用注解，将 Bean 配置为组件类本身，而不是使用 XML 来描述 Bean 装配。默认情况下，Spring 容器中未打开注解装配。因此，您需要在用它之前在 Spring 配置文件中启用它。例如：

```
1 <beans>
2 <context:annotation-config/>
3 <!-- bean definitions go here -->
4 </beans>
```

3、Java Config 配置。

Spring 的 Java 配置是通过使用 @Bean 和 @Configuration 来实现。

- @Bean 注解扮演与 <bean /> 元素相同的角色。
- @Configuration 类允许通过简单地调用同一个类中的其他 @Bean 方法来定义 Bean 间依赖关系。

```
1 @Configuration
2 public class StudentConfig {
3     @Bean
4     public StudentBean myStudent() {
5         return new StudentBean();
6     }
7 }
```

目前主要使用 **Java Config** 配置为主。当然，三种配置方式是可以混合使用的。例如说：

- Dubbo 服务的配置，芬芳喜欢使用 XML 。
- Spring MVC 请求的配置，芬芳喜欢使用 `@RequestMapping` 注解。
- Spring MVC 拦截器的配置，芬芳喜欢 Java Config 配置。

另外，现在已经是 Spring Boot 的天下，所以更加是 **Java Config** 配置为主。

Spring Boot 的核心注解是哪个？

```
1 package cn.iocoder.skywalking.web01;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Web01Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Web01Application.class, args);
11     }
12
13 }
```

- `@SpringBootApplication` 注解，就是 Spring Boot 的核心注解。

`org.springframework.boot.autoconfigure.SpringBootApplication` 注解的代码如下：

```
1 // SpringBootApplication.java
2
3 @Target({ElementType.TYPE})
4 @Retention(RetentionPolicy.RUNTIME)
5 @Documented
6 @Inherited
7 @SpringBootConfiguration
8 @EnableAutoConfiguration
9 @ComponentScan(
10     excludeFilters = {@Filter(
11         type = FilterType.CUSTOM,
12         classes = {TypeExcludeFilter.class}
13     )}, @Filter(
14         type = FilterType.CUSTOM,
15         classes = {AutoConfigurationExcludeFilter.class}
```

```

16    }}
17    )
18    public @interface SpringBootApplication {
19        @AliasFor(
20            annotation = EnableAutoConfiguration.class
21        )
22        Class<?>[] exclude() default {};
23
24        @AliasFor(
25            annotation = EnableAutoConfiguration.class
26        )
27        String[] excludeName() default {};
28
29        @AliasFor(
30            annotation = ComponentScan.class,
31            attribute = "basePackages"
32        )
33        String[] scanBasePackages() default {};
34
35        @AliasFor(
36            annotation = ComponentScan.class,
37            attribute = "basePackageClasses"
38        )
39        Class<?>[] scanBasePackageClasses() default {};
40    }

```

- 它组合了 3 个注解，详细说明，胖友看看 [《Spring Boot 系列：@SpringBootApplication 注解》](#)。
- `@Configuration` 注解，指定类是 **Bean** 定义的配置类。

`@Configuration` 注解，来自 `spring-context` 项目，用于 Java Config，不是 Spring Boot 新带来的。

- `@ComponentScan` 注解，扫描指定包下的 Bean 们。

`@ComponentScan` 注解，来自 `spring-context` 项目，用于 Java Config，不是 Spring Boot 新带来的。

- `@EnableAutoConfiguration` 注解，打开自动配置的功能。如果我们想要关闭某个类的自动配置，可以设置注解的 `exclude` 或 `excludeName` 属性。

`@EnableAutoConfiguration` 注解，来自 `spring-boot-autoconfigure` 项目，它才是 **Spring Boot** 新带来的。

什么是 Spring Boot 自动配置？

在「[Spring Boot 的核心注解是哪个？](#)」中，我们已经看到，使用 `@EnableAutoConfiguration` 注解，打开 Spring Boot 自动配置的功能。具体如何实现的，可以看看如下两篇文章：

- [《@EnableAutoConfiguration 注解的工作原理》](#)。
- [《一个面试题引起的 Spring Boot 启动解析》](#)
- 建议，能一边调试，一边看这篇文章。调试很简单，任一搭建一个 Spring Boot 项目即可。

如下是一个比较简单的总结：

1. Spring Boot 在启动时扫描项目所依赖的 jar 包，寻找包含 `spring.factories` 文件的 jar 包。
2. 根据 `spring.factories` 配置加载 `AutoConfigure` 类。
3. 根据 [@Conditional 等条件注解](#) 的条件，进行自动配置并将 Bean 注入 Spring IoC 中。

Spring Boot 有哪几种读取配置的方式？

Spring Boot 目前支持 2 种读取配置：

1、@Value 注解，读取配置到属性。最最最常用。

另外，支持和 `@PropertySource` 注解一起使用，指定使用的配置文件。

2、@ConfigurationProperties 注解，读取配置到类上。

另外，支持和 `@PropertySource` 注解一起使用，指定使用的配置文件。

详细的使用方式，可以参考 [《Spring Boot 读取配置的几种方式》](#)。

使用 Spring Boot 后，项目结构是怎么样的呢？

我们先来说说项目的分层。一般来说，主流的有两种方式：

- 方式一，`controller`、`service`、`dao` 三个包，每个包下面添加相应的 `XXXController`、`YYYService`、`ZZZDAO`。
- 方式二，按照业务模块分包，每个包里面放 `Controller`、`Service`、`DAO` 类。例如，业务模块分成 `user`、`order`、`item` 等等包，在 `user` 包里放 `UserController`、`UserService`、`UserDAO` 类。

那么，使用 Spring Boot 的项目怎么分层呢？芳芳自己的想法

- 现在项目都会进行服务化分拆，每个项目不会特别复杂，所以建议使用【方式一】。
- 以前的项目，大多是单体的项目，动则项目几万到几十万的代码，当时多采用【方式二】。

下面是一个简单的 Spring Boot 项目的 Demo，如下所示：



如何在 Spring Boot 启动的时候运行一些特殊的代码？

如果需要在 SpringApplication 启动后执行一些特殊的代码，你可以实现 `ApplicationRunner` 或 `CommandLineRunner` 接口，这两个接口工作方式相同，都只提供单一的 `run` 方法，该方法仅在 `SpringApplication#run(...)` 方法完成之前调用。

一般情况下，我们不太会使用该功能。如果真需要，胖友可以详细看看 《使用 `ApplicationRunner` 或 `CommandLineRunner` 》。

Spring Boot 2.X 有什么新特性?

1. 起步 JDK 8 和支持 JDK 9
2. 第三方库的升级
3. Reactive Spring
4. HTTP/2 支持
5. 配置属性的绑定
6. Gradle 插件
7. Actuator 改进
8. 数据支持的改进
9. Web 的改进
10. 支持 Quartz 自动配置
11. 测试的改进
12. 其它...

详细的说明, 可以看看 《Spring Boot 2.0系列文章(二): Spring Boot 2.0 新特性详解》。

整合篇

如何将内嵌服务器换成 Jetty ?

默认情况下, `spring-boot-starter-web` 模块使用 Tomcat 作为内嵌的服务器。所以需要去除对 `spring-boot-starter-tomcat` 模块的引用, 添加 `spring-boot-starter-jetty` 模块的引用。代码如下:

```
1  <dependency>
2      <groupId>org.springframework.boot</groupId>
3      <artifactId>spring-boot-starter-web</artifactId>
4      <exclusions>
5          <exclusion> <!-- 去除 Tomcat -->
6              <groupId>org.springframework.boot</groupId>
7              <artifactId>spring-boot-starter-tomcat</artifactId>
8          </exclusion>
9      </exclusions>
10 </dependency>
11 <dependency> <!-- 引入 Jetty -->
12     <groupId>org.springframework.boot</groupId>
13     <artifactId>spring-boot-starter-jetty</artifactId>
14 </dependency>
```

Spring Boot 中的监视器 Actuator 是什么？

`spring-boot-actuator` 提供 Spring Boot 的监视器功能，可帮助我们访问生产环境中正在运行的应用程序的当前状态。

- 关于 Spring Boot Actuator 的教程，可以看看 [《Spring Boot Actuator 使用》](#)。
- 上述教程是基于 Spring Boot 1.X 的版本，如果胖友使用 Spring Boot 2.X 的版本，你将会发现 `/beans` 等 Endpoint 是不存在的，参考 [《Spring boot 2 - Actuator endpoint, where is /beans endpoint》](#) 问题来解决。

安全性

Spring Boot 2.X 默认情况下，`spring-boot-actuator` 产生的 Endpoint 是没有安全保护的，但是 Actuator 可能暴露敏感信息。

所以一般的做法是，引入 `spring-boot-start-security` 依赖，使用 Spring Security 对它们进行安全保护。

如何集成 Spring Boot 和 Spring MVC ？

1. 引入 `spring-boot-starter-web` 的依赖。
2. 实现 `WebMvcConfigurer` 接口，可添加自定义的 Spring MVC 配置。

因为 Spring Boot 2 基于 JDK 8 的版本，而 JDK 8 提供 `default` 方法，所以 Spring Boot 2 废弃了 `WebMvcConfigurerAdapter` 适配类，直接使用 `WebMvcConfigurer` 即可。

```
1 // WebMvcConfigurer.java
2 public interface WebMvcConfigurer {
3
4     /** 配置路径匹配器 */
5     default void configurePathMatch(PathMatchConfigurer configurer) {}
6
7     /** 配置内容裁决的一些选项 */
8     default void configureContentNegotiation(ContentNegotiationConfigurer
9 configurer) { }
10
11     /** 异步相关的配置 */
12     default void configureAsyncSupport(AsyncSupportConfigurer configurer) { }
13
14     default void configureDefaultServletHandling(DefaultServletHandlerConfigurer
15 configurer) { }
16
17     default void addFormatters(FormatterRegistry registry) {
18 }
19
20     /** 添加拦截器 */
21     default void addInterceptors(InterceptorRegistry registry) { }
22
23     /** 静态资源处理 */
24     default void addResourceHandlers(ResourceHandlerRegistry registry) { }
```

```

23
24     /** 解决跨域问题 */
25     default void addCorsMappings(CorsRegistry registry) { }
26
27     default void addViewControllers(ViewControllerRegistry registry) { }
28
29     /** 配置视图解析器 */
30     default void configureViewResolvers(ViewResolverRegistry registry) { }
31
32     /** 添加参数解析器 */
33     default void addArgumentResolvers(List<HandlerMethodArgumentResolver>
resolvers) {
34         }
35
36     /** 添加返回值处理器 */
37     default void addReturnValueHandlers(List<HandlerMethodReturnValueHandler>
handlers) { }
38
39     /** 这里配置视图解析器 */
40     default void configureMessageConverters(List<HttpMessageConverter<?>>
converters) { }
41
42     /** 配置消息转换器 */
43     default void extendMessageConverters(List<HttpMessageConverter<?>> converters)
{ }
44
45     /** 配置异常处理器 */
46     default void configureHandlerExceptionResolvers(List<HandlerExceptionResolver>
resolvers) { }
47
48     default void extendHandlerExceptionResolvers(List<HandlerExceptionResolver>
resolvers) { }
49
50     @Nullable
51     default Validator getValidator() { return null; }
52
53     @Nullable
54     default MessageCodesResolver getMessageCodesResolver() { return null; }
55
56 }

```

在使用 Spring MVC 时，我们一般会做如下几件事情：

1. 实现自己项目需要的拦截器，并在 WebMvcConfigurer 实现类中配置。可参见 [MvcConfiguration](#) 类。
2. 配置 `@ControllerAdvice` + `@ExceptionHandler` 注解，实现全局异常处理。可参见

[GlobalExceptionHandler](#) 类。

3. 配置 `@ControllerAdvice`，实现 `ResponseBodyAdvice` 接口，实现全局统一返回。可参见 [GlobalResponseBodyAdvice](#)。

当然，有一点需要注意，`WebMvcConfigurer`、`ResponseBodyAdvice`、`@ControllerAdvice`、`@ExceptionHandler` 接口，都是 Spring MVC 框架自身已有的东西。

- `spring-boot-starter-web` 的依赖，帮我们解决的是 Spring MVC 的依赖以及相关的 Tomcat 等组件。

如何集成 Spring Boot 和 Spring Security ?

目前比较主流的安全框架有两个：

1. Spring Security
2. Apache Shiro

对于任何项目来说，安全认证总是少不了，同样适用于使用 Spring Boot 的项目。相对来说，Spring Security 现在会比 Apache Shiro 更流行。

Spring Boot 和 Spring Security 的配置方式比较简单：

1. 引入 `spring-boot-starter-security` 的依赖。
2. 继承 `WebSecurityConfigurerAdapter`，添加自定义的安全配置。

当然，每个项目的安全配置是不同的，需要胖友自己选择。更多详细的使用，建议认真阅读如下文章：

- [《Spring Boot中 使用 Spring Security 进行安全控制》](#)，快速上手。
- [《Spring Security 实现原理与源码解析系统——精品合集》](#)，深入源码。

另外，安全是一个很大的话题，感兴趣的胖友，可以看看 [《Spring Boot 十种安全措施》](#) 一文。

如何集成 Spring Boot 和 Spring Security OAuth2 ?

参见 [《Spring Security OAuth2 入门》](#) 文章，内容有点多。

如何集成 Spring Boot 和 JPA ?

1. 引入 `spring-boot-starter-data-jpa` 的依赖。
2. 在 application 配置文件中，加入 JPA 相关的少量配置。当然，数据库的配置也要添加进去。
3. 具体编码。

详细的使用，胖友可以参考：

- [《一起来学 SpringBoot 2.x | 第六篇：整合 Spring Data JPA》](#)

有两点需要注意：

- Spring Boot 2 默认使用的数据库连接池是 [HikariCP](#)，目前最好的性能的数据库连接池的实现。
- `spring-boot-starter-data-jpa` 的依赖，使用的默认 JPA 实现是 Hibernate 5.X。

如何集成 Spring Boot 和 MyBatis ?

1. 引入 `mybatis-spring-boot-starter` 的依赖。
2. 在 application 配置文件中，加入 MyBatis 相关的少量配置。当然，数据库的配置也要添加进去。
3. 具体编码。

详细的使用，胖友可以参考：

- [《一起来学 SpringBoot 2.x | 第七篇：整合 Mybatis》](#)

如何集成 Spring Boot 和 RabbitMQ ?

1. 引入 `spring-boot-starter-amqp` 的依赖
2. 在 application 配置文件中，加入 RabbitMQ 相关的少量配置。
3. 具体编码。

详细的使用，胖友可以参考：

- [《一起来学 SpringBoot 2.x | 第十二篇：初探 RabbitMQ 消息队列》](#)
- [《一起来学 SpringBoot 2.x | 第十三篇：RabbitMQ 延迟队列》](#)

如何集成 Spring Boot 和 Kafka ?

1. 引入 `spring-kafka` 的依赖。
2. 在 application 配置文件中，加入 Kafka 相关的少量配置。
3. 具体编码。

详细的使用，胖友可以参考：

- [《Spring Boot系列文章（一）：SpringBoot Kafka 整合使用》](#)

如何集成 Spring Boot 和 RocketMQ ?

1. 引入 `rocketmq-spring-boot` 的依赖。
2. 在 application 配置文件中，加入 RocketMQ 相关的少量配置。
3. 具体编码。

详细的使用，胖友可以参考：

- [《我用这种方法在 Spring 中实现消息的发送和消费》](#)

Spring Boot 支持哪些日志框架？


Spring Boot 支持的日志框架有：

- Logback
- Log4j2
- Log4j
- Java Util Logging

默认使用的是 Logback 日志框架，也是目前较为推荐的，具体配置，可以参见 [《一起来学 SpringBoot 2.x | 第三篇：SpringBoot 日志配置》](#)。

因为 Log4j2 的性能更加优秀，也有人在生产上使用，可以参考 [《Spring Boot Log4j2 日志性能之巅》](#) 配置。

666. 彩蛋

 看完之后，复习复习 Spring Boot 美滋滋。有一种奇怪的感觉，把面试题写成了 Spring 的学习指南。

当然，如果胖友有新的面试题，欢迎在星球一起探讨补充。

参考和推荐如下文章：

- 我有面试宝典 [[《经验分享》Spring Boot面试题总结》](#)]
- Java 知音 [《Spring Boot 面试题精华》](#)
- 祖大帅 [《一个面试题引起的 Spring Boot 启动解析》](#)
- 大胡子叔叔 [《Spring Boot + Spring Cloud 相关面试题》](#)
- 墨斗鱼博客 [《20 道 Spring Boot 面试题》](#)
- 夕阳雨晴 [《Spring Boot Starter 的面试题》](#)