SpringBoot 注解

Java基础加强总结(一)——注解(Annotation)

https://www.cnblogs.com/xdp-gacl/p/3622275.html

SpringBoot

注解@SpringBootApplication 解析

https://blog.csdn.net/claram/article/details/75125749

`@Configuration` 注解，指定类是 Bean 定义的配置类。

`#ComponentScan` 注解，扫描指定包下的 Bean 们。

`@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。

SpringBoot-@PathVariable

https://www.cnblogs.com/fangpengchengbupter/p/7823493.html

8、SpringBoot几个常用的注解

（1）@RestController和@Controller指定一个类，作为控制器的注解

（2）@RequestMapping方法级别的映射注解，这一个用过Spring MVC的小伙伴相信都很熟悉

（3）@EnableAutoConfiguration和@SpringBootApplication是类级别的注解，根据maven依赖的jar来自动猜测完成正确的spring的对应配置，只要引入了spring-boot-starter-web的依赖，默认会自动配置Spring MVC和tomcat容器

（4）@Configuration类级别的注解，一般这个注解，我们用来标识main方法所在的类,完成元数据bean的初始化。

（5）@ComponentScan类级别的注解，自动扫描加载所有的Spring组件包括Bean注入，一般用在main方法所在的类上

（6）@ImportResource类级别注解，当我们必须使用一个xml的配置时，使用@ImportResource和@Configuration来标识这个文件资源的类。

（7）@Autowired注解，一般结合@ComponentScan注解，来自动注入一个Service或Dao级别的Bean
（8）@Component类级别注解，用来标识一个组件，比如我自定了一个filter，则需要此注解标识之后，Spring Boot才会正确识别。

1、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。

- 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。

2、Spring Boot 有什么优缺点？

芳芳：任何技术栈，有优点必有缺点，没有银弹。

另外，这个问题的回答，我们是基于 《Spring Boot浅谈(是什么/能干什么/优点和不足)》 整理，所以胖友主要看下这篇文章。

Spring Boot 的优点

芳芳：优点和 「Spring Boot 提供了哪些核心功能？」 问题的答案，是比较重叠的。

- 1、使【编码】变简单。
- 2、使【配置】变简单。
- 3、使【部署】变简单。
- 4、使【监控】变简单。

Spring Boot 的缺点

芳芳：如下的缺点，基于 《Spring Boot浅谈(是什么/能干什么/优点和不足)》，考虑的出发点是把 Spring Boot 作为微服务的框架的选型的角度进行考虑。

- 1、没有提供相应的【服务发现和注册】的配套功能。

芳芳：当然，实际上 Spring Boot 本身是不需要提供这样的功能。服务发现和注册的功能，是在 Spring Cloud 中进行提供。

- 2、自身的 acturator 所提供的【监控功能】，也需要与现有的监控对接。
- 3、没有配套的【安全管控】方案。

芳芳：关于这一点，芳芳也有点迷糊，Spring Security 是可以比较方便的集成到 Spring Boot 中，所以不晓得这里的【安全管控】的定义是什么。所以这一点，面试的时候回答，可以暂时先省略。

- 4、对于 REST 的落地，还需要自行结合实际进行 URI 的规范化工作

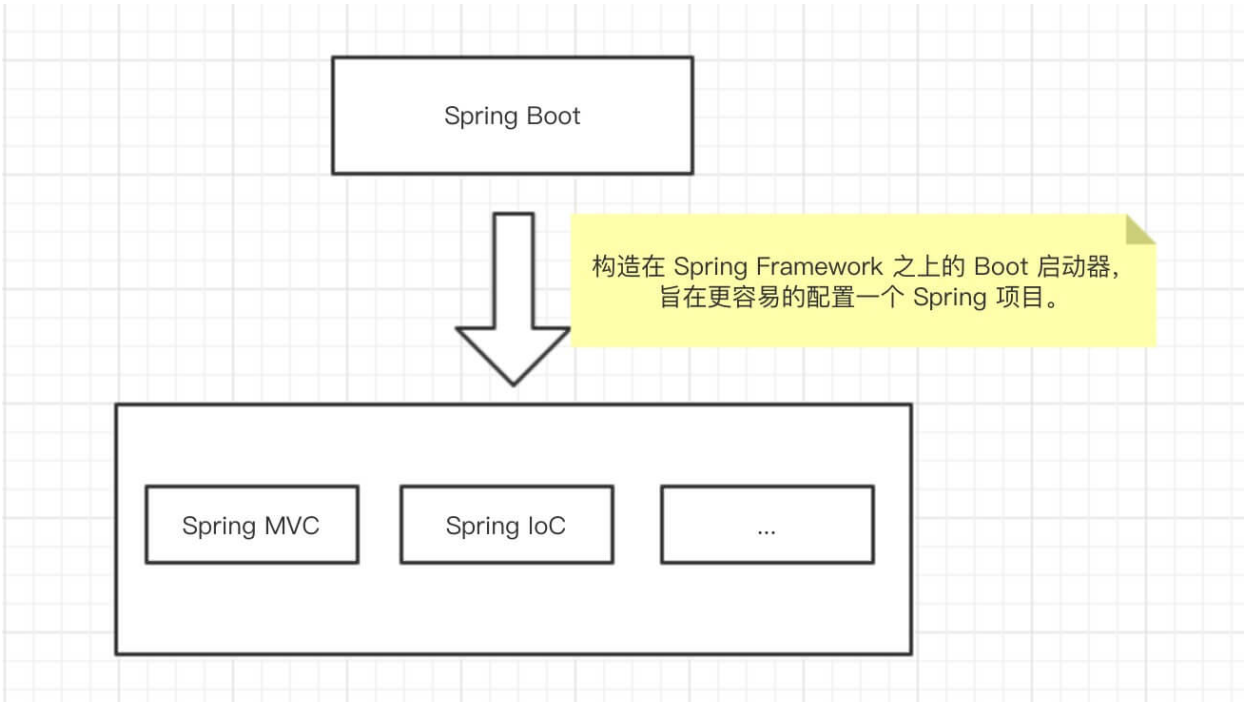
芳芳：这个严格来说，不算缺点。本身，是规范的范畴。

3、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：

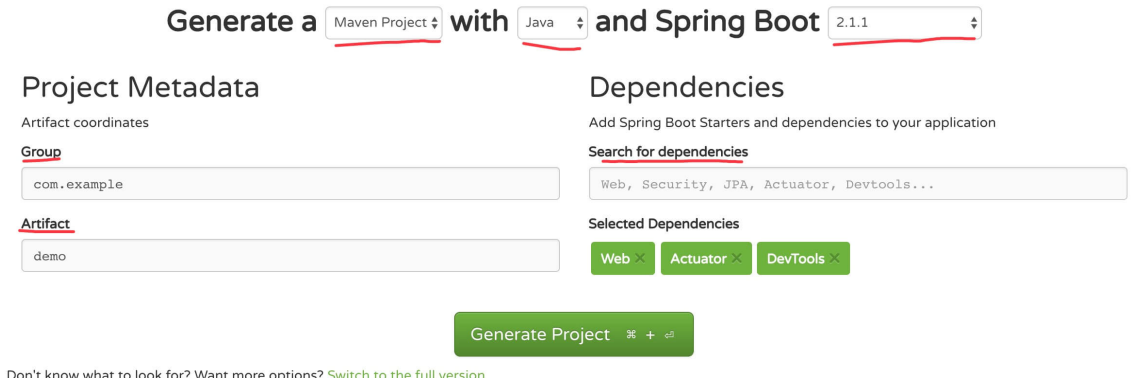


4、Spring Boot 常用的 Starter 有哪些？

- `spring-boot-starter-web` ：提供 `Spring MVC + 内嵌的 Tomcat` 。
- `spring-boot-starter-data-jpa` ：提供 `Spring JPA + Hibernate` 。
- `spring-boot-starter-data-redis` ：提供 `Redis` 。
- `mybatis-spring-boot-starter` ：提供 `MyBatis` 。

5、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：



- 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。

6、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：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.1.RELEASE</version>
</parent>
```

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

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>1.5.1.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</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。

7、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。

8、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》。

9、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。

```
server.port=9090
```

- 方式二，通过启动命令增加 `server.port` 参数进行修改。

```
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。

10、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》 。

11、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：

```
server.port = 9090
```

- `.yaml` 格式。示例如下：

```
server:
    port: 9090
```

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。

12、Spring Boot 默认配置文件是什么？

对于 Spring Boot 应用，默认的配置文件根目录下的 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。

13、Spring Boot 如何定义多套不同环境配置？

可以参考 《Spring Boot 教程 - Spring Boot Profiles 实现多环境下配置切换》 一文。

但是，需要考虑一个问题，生产环境的配置文件的安全性，显然我们不能且不应该把生产的配置放到项目的 Git 仓库中进行管理。那么应该怎么办呢？

- 方案一，生产环境的配置文件放在生产环境的服务器中，以 `java -jar myproject.jar --spring.config.location=/xxx/yyy/application-prod.properties` 命令，设置 参数 `spring.config.location` 指向配置文件。
- 方案二，使用 Jenkins 在执行打包，配置上 Maven Profile 功能，使用服务器上的配置文件。
- 😺 整体来说，和【方案一】的差异是，将配置文件打包进了 Jar 包中。
- 方案三，使用配置中心。

14、Spring Boot 配置加载顺序？
SpringCloud微服务实战
1、命令行中传参
2、SPRING_APPLICATION_JSON中属性。SPRING_APPLICATION_JSON是以JSON格式配置在系统变量中内容
3、java:comp/env 中的JNDI属性
4、java的系统属性，可以通过System.getPropertities()获得内容
5、操作系统的环境变量
6、通过random.*配置的随机变量

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。

15、Spring Boot 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：

```
<bean id="studentBean" class="org.edureka.firstSpring.StudentBean">
    <property name="name" value="Edureka"></property>
</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，您需要在使用它之前在 Spring 配置文件中启用它。例如：

```
<beans>
<context:annotation-config/>
<!-- bean definitions go here -->
</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。
  - 例如：

```
@Configuration
public class StudentConfig {

    @Bean
    public StudentBean myStudent() {
        return new StudentBean();
    }

}
```

    - 是不是很熟悉 😈

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，艿艿喜欢使用 XML 。
- Spring MVC 请求的配置，艿艿喜欢使用 @RequestMapping 注解。
- Spring MVC 拦截器的配置，艿艿喜欢 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。

16、Spring Boot 的核心注解是哪个？

```
package cn.iocoder.skywalking.web01;

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

@SpringBootApplication
public class Web01Application {

    public static void main(String[] args) {
        SpringApplication.run(Web01Application.class, args);
    }
```

```
}
```

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

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

```
// SpringBootApplication.java

@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 {
    @AliasFor(
        annotation = EnableAutoConfiguration.class
    )
    Class<?>[] exclude() default {};

    @AliasFor(
        annotation = EnableAutoConfiguration.class
    )
    String[] excludeName() default {};

    @AliasFor(
        annotation = ComponentScan.class,
        attribute = "basePackages"
    )
    String[] scanBasePackages() default {};

    @AliasFor(
        annotation = ComponentScan.class,
        attribute = "basePackageClasses"
    )
    Class<?>[] scanBasePackageClasses() default {};
}
```

- 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。

17、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》 。

18、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。