1. 从spring-boot-starter-parent 项目继承，并将依赖项声明为一个或多个“Starters”
2. @SpringBootApplication annotation通常放在你的主类上，它隐式地为某些项目定义了一个基本的“搜索包”
3. 如果您不想使用 @SpringBootApplication ，则 @EnableAutoConfiguration 和 @ComponentScan 注释将定义该行为，以便您也可以使用该注释
4. Spring Boot支持基于Java的配置。 虽然可以将SpringApplication 与XML源一起使用，但我们通常建议您的主要源是一个@Configuration 类。 通常，定义main 方法的类作为主要的@Configuration 是一个很好的候选@Configuration 。
5. 不需要的特定自动配置类正在应用，则可以使用 @EnableAutoConfiguration 的exclude属性来禁用它们@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
6. @SpringBootApplication 注释是相当于使用 @Configuration ， @EnableAutoConfiguration 和 @ComponentScan 与他们的默认属性
7. 则可以使用 java -jar 运行应用程序，如以下示例所示：$ java -jar target/myapplication-0.0.1-SNAPSHOT.jar
8. 也可以在启用远程调试支持的情况下运行打包的应用程序。

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myapplication-0.0.1-SNAPSHOT.jar

1. Spring Boot Maven插件包含一个可用于快速编译和运行应用程序的run 目标。mvn spring-boot:run
2. 每当类路径上的文件发生更改时，使用 spring-boot-devtools 自动重新启动
3. Spri n g Boot提供的重启技术通过使用两个类加载器来工作。 不改变的类（例如来自第三方j ar的类）被加载到基类加载器中。 您正在开发的类将加载到重启类加载器中。 当应用程序重新启动时， 重新启动类加载器将被丢弃并创建一个新类。 这种方法意味着应用程序重启通常比“冷启动”快得多
4. 某些资源不一定需要在更改时触发重新启动。 例如，可以就地编辑T h y mel eaf模板。 默认情况下，在改变资源 /META-INF/maven ， /META-INF/resources ，/resources ， /static ， /public ，或 /templates 不会触发重启但并触发l i v e rel oad 。 如果您想自定义这些排除项，可以使用 spring.devtools.restart.exclude 属性，例如，要仅排除 /static 和 /public 您可以设置以下属性
5. spring.devtools.restart.exclude=static/\*\*,public/\*\*
6. 如果您想保留这些默认设置并 添加其他排除项，请改为使用 spring.devtools.restart.additional-exclude 属性
7. 如果您不想使用重新启动功能，则可以使用spring.devtools.restart.enabled 属性将其禁用
8. 如果您使用 java -jar 运行应用程序，则可以按如下所示启用 debug 属性：

java -jar myproject-0.0.1-SNAPSHOT.jar --debug

1. 启动时打印的横幅可以通过将 banner.txt 文件添加到类路径中或通过将 spring.banner.location 属性设置为此类文件的位置来更改
2. 您需要访问传递给 SpringApplication.run(… ) 的应用程序参数，则可以注入一个 org.springframework.boot.ApplicationArguments bean
3. 如果您需要在 SpringApplication 启动后运行某些特定代码，则可以实现 ApplicationRunner 或 CommandLineRunner 接口
4. 通过使 用 @Value 注释，将属性值直接注入到bean中
5. RandomValuePropertySource 对于注入随机值my.secret=${random.value}
6. SpringApplication 将任何命令行选项参数（即以 -- 开头的参数（如 --server.port=9000 ））转换为 property ，并将它们添加到Spring Environment 。 如前所述，命令行属性始终优先于其他属性源。
7. 如果您不喜欢 application.properties 作为配置文件名，则可以通过指定 spring.config.name 环境属性来切换到另一个文件名
8. 除 application.properties 文件外，还可以使用以下命名约定来定义配置文件特定的属性： application-{profile}.properties
9. 注册 @ConfigurationProperties bean时，该bean具有常规名称： - ，其中 是 @ConfigurationProperties 注释中指定的环境键前缀， 是Bean的完全限定名称。
10. Spring Profiles提供了一种分离部分应用程序配置的方法，并使其仅在特定环境中可用。 任何 @Component 或 @Configuration 都可以用 @Profile 标记以限制其加载时间，@Profile("production")
11. 用 spring.profiles.active Environment 属性来指定哪些配置文件处于活动状态
12. Spring Boot将Commons Logging用于所有内部日志记录，但将底层日志实现保留为打开状态。 提供用于默认配置Java Util Logging ， Log4J2 ，和Logback
13. 默认情况下，如果使用“启动器”，则使用Logback进行日志记录
14. 默认情况下，会记录 ERROR 级别， WARN 级别和 INFO 级别的消息。 您还可以通过以 --debug 标志启动应用程序来启 用“调试”模式。 java -jar myapp.jar --debug
15. 除了输出控制台之外还想写日志文件，则需要设置 logging.file 或 logging.path 属性
16. 日志记录系统都可以使用 logging.level.= 其中 level 是TRACE，DEBUG，INFO，WARN，ERROR，FATAL或OFF之一）

logging.level.root=WARN

logging.level.org.springframework.web=DEBUG

logging.level.org.hibernate=ERROR

1. 标签可让您根据活动的Spring配置文件选择性地包含或排除配置部分
2. 使用 标签可以显示Spring Environment 属性以便在Logback中使用。
3. 使用 标签可以显示Spring Environment 属性以便在Logback中使用
4. 你可以添加自己 @Configuration 类型的 类 WebMvcConfigurer
5. Spring MVC使用 HttpMessageConverter 接口来转换HTTP请求和响应
6. 如果您需要添加或自定义转换器，则可以使用Spring Boot的 HttpMessageConverters 类，

@Configuration

public class MyConfiguration {

@Bean

public HttpMessageConverters customConverters() {

HttpMessageConverter additional = ...

HttpMessageConverter another = ...

return new HttpMessageConverters(additional, another);

}

}

1. 目录称为静态内容 /static （或 /public 或者 /resources 或者 /META-INF/resources 在classpath
2. 还可以使用 spring.resources.static-locations 属性（使用目录位置列表替换缺省值）来自定义静态资源位置
3. Spring Boot支持静态和模板欢迎页面。 它首先在配置的静态内容位置中查找 index.html 文件
4. Spring Boot favicon.ico 在配置的静态内容位置和类路径的根目录
5. 定义一个用 @ControllerAdvice 注释的类来定制JSON文档以返回特定的控制器和/或异常类型
   1. @ControllerAdvice(basePackageClasses = AcmeController.class)
   2. public class AcmeControllerAdvice extends ResponseEntityExceptionHandler {
   3. @ExceptionHandler(YourException.class)
   4. @ResponseBody
   5. ResponseEntity handleControllerException(HttpServletRequest request, Throwable ex) {
   6. HttpStatus status = getStatus(request);
   7. return new ResponseEntity<>(new CustomErrorType(status.value(), ex.getMessage()), status);
   8. }
   9. private HttpStatus getStatus(HttpServletRequest request) {
   10. Integer statusCode = (Integer) request.getAttribute("javax.servlet.error.status\_code");
   11. if (statusCode == null) {
   12. return HttpStatus.INTERNAL\_SERVER\_ERROR;
   13. }
   14. return HttpStatus.valueOf(statusCode);
   15. }
   16. }