

09、原理解析

1、Profile功能

为了方便多环境适配，springboot简化了profile功能。

1、application-profile功能

- 默认配置文件 application.yaml; 任何时候都会加载
- 指定环境配置文件 application-{env}.yaml
- 激活指定环境
 - 配置文件激活
 - 命令行激活: `java -jar xxx.jar --spring.profiles.active=prod --person.name=haha`
 - 修改配置文件的任意值, 命令行优先
- 默认配置与环境配置同时生效
- 同名配置项, profile配置优先

2、@Profile条件装配功能

Java

[复制代码](#)

```
1  @Configuration(proxyBeanMethods = false)
2  @Profile("production")
3  public class ProductionConfiguration {
4
5      // ...
6
7  }
```

3、profile分组

Plain Text

[复制代码](#)

```
1  spring.profiles.group.production[0]=proddb
2  spring.profiles.group.production[1]=prodmq
3
4  使用: --spring.profiles.active=production  激活
```

2、外部化配置

<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config> <<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config>>

1. Default properties (specified by setting `SpringApplication.setDefaultProperties`).
2. `@PropertySource` <<https://docs.spring.io/spring/docs/5.3.1/javadoc-api/org/springframework/context/annotation/PropertySource.html>> annotations on your `@Configuration` classes. Please note that such property sources are not added to the `Environment` until the application context is being refreshed. This is too late to configure certain properties such as `logging.*` and `spring.main.*` which are read before refresh begins.
3. **Config data (such as `application.properties` files)**
4. A `RandomValuePropertySource` that has properties only in `random.*`.
5. OS environment variables.
6. Java System properties (`System.getProperties()`).
7. JNDI attributes from `java:comp/env`.
8. `ServletContext` init parameters.
9. `ServletConfig` init parameters.
10. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
11. Command line arguments.
12. `properties` attribute on your tests. Available on `@SpringBootTest` <<https://docs.spring.io/spring-boot/docs/2.4.0/api/org/springframework/boot/test/context/SpringBootTest.html>> and the test annotations for testing a particular slice of your application <<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-testing-spring-boot-applications-testing-autoconfigured-tests>> .
13. `@TestPropertySource` <<https://docs.spring.io/spring/docs/5.3.1/javadoc-api/org/springframework/test/context/TestPropertySource.html>> annotations on your tests.
14. Devtools global settings properties <<https://docs.spring.io/spring-boot/docs/current/reference/html/using-spring-boot.html#using-boot-devtools-globalsettings>> in the `$HOME/.config/spring-boot` directory when devtools is active.

1、外部配置源

常用：Java属性文件、YAML文件、环境变量、命令行参数；

2、配置文件查找位置

- (1) classpath 根路径
- (2) classpath 根路径下config目录
- (3) jar包当前目录
- (4) jar包当前目录的config目录
- (5) /config子目录的直接子目录

3、配置文件加载顺序：

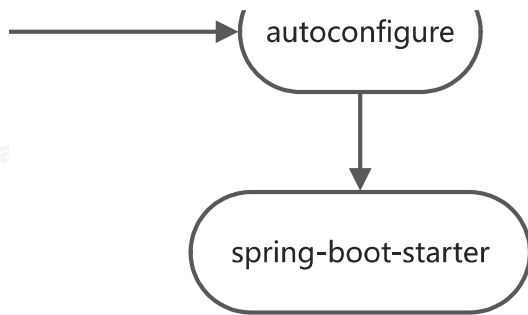
1. 当前jar包内部的application.properties和application.yml
2. 当前jar包内部的application-{profile}.properties 和 application-{profile}.yml
3. 引用的外部jar包的application.properties和application.yml
4. 引用的外部jar包的application-{profile}.properties 和 application-{profile}.yml

4、指定环境优先，外部优先，后面的可以覆盖前面的同名配置项

3、自定义starter

1、starter启动原理

- starter-pom引入 autoconfigurer 包



- autoconfigure包中配置使用 **META-INF/spring.factories** 中 **EnableAutoConfiguration** 的值，使得项目启动加载指定的自动配置类
- 编写自动配置类 `xxxAutoConfiguration -> xxxProperties`

- **@Configuration**
- **@Conditional**
- **@EnableConfigurationProperties**
- **@Bean**
-

引入starter --- xxxAutoConfiguration --- 容器中放入组件 ---- 绑定xxxProperties ---- 配置项

2、自定义starter

atguigu-hello-spring-boot-starter (启动器)

atguigu-hello-spring-boot-starter-autoconfigure (自动配置包)

4、SpringBoot原理

Spring原理【[Spring注解 <https://www.bilibili.com/video/BV1gW411W7wy?p=1>](https://www.bilibili.com/video/BV1gW411W7wy?p=1)】、**SpringMVC原理**、**自动配置原理**、SpringBoot原理

1、SpringBoot启动过程

- 创建 **SpringApplication**
 - 保存一些信息。
 - 判定当前应用的类型。ClassUtils。Servlet
 - **bootstrappers**: 初始启动引导器 (List<Bootstrapper>) : 去spring.factories文件中找 org.springframework.boot.**Bootstrapper**
 - 找 **ApplicationContextInitializer**; 去spring.factories找 **ApplicationContextInitializer**
 - List<ApplicationContextInitializer<?>> **initializers**
 - 找 **ApplicationListener** ; 应用监听器。去spring.factories找 **ApplicationListener**


- List<ApplicationListener<?>> listeners

- 运行 **SpringApplication**

- **StopWatch**
- 记录应用的启动时间
- 创建引导上下文（Context环境）createBootstrapContext()
 - 获取到所有之前的 **bootstrappers** 挨个执行 initialize() 来完成对引导启动器上下文环境设置
- 让当前应用进入 **headless** 模式。java.awt.headless
- 获取所有 **RunListener**（运行监听器）【为了方便所有Listener进行事件感知】
 - getSpringFactoriesInstances 去spring.factories找 **SpringApplicationRunListener**.
- 遍历 **SpringApplicationRunListener** 调用 **starting** 方法;
 - 相当于通知所有感兴趣系统正在启动过程的人，项目正在 **starting**。
- 保存命令行参数；ApplicationArguments
- 准备环境 prepareEnvironment () ;
 - 返回或者创建基础环境信息对象。StandardServletEnvironment
 - 配置环境信息对象。
 - 读取所有的配置源的配置属性值。
 - 绑定环境信息
 - 监听器调用 listener.environmentPrepared(); 通知所有的监听器当前环境准备完成
- 创建IOC容器（createApplicationContext ()）
 - 根据项目类型（Servlet）创建容器，
 - 当前会创建 **AnnotationConfigServletWebServerApplicationContext**
- 准备ApplicationContext IOC容器的基本信息 prepareContext()
 - 保存环境信息
 - IOC容器的后置处理流程。
 - 应用初始化器；applyInitializers;
 - 遍历所有的 **ApplicationContextInitializer** 。调用 initialize。来对ioc容器进行初始化扩展功能
 - 遍历所有的 listener 调用 contextPrepared。EventPublishRunListenr；通知所有的监听器contextPrepared
 - 所有的监听器 调用 contextLoaded。通知所有的监听器 contextLoaded;
- 刷新IOC容器。refreshContext
 - 创建容器中的所有组件（Spring注解）
- 容器刷新完成后工作？afterRefresh
- 所有监听器 调用 listeners.started(context); 通知所有的监听器 started
- 调用所有runners；callRunners()
 - 获取容器中的 **ApplicationRunner**
 - 获取容器中的 **CommandLineRunner**
 - 合并所有runner并且按照@Order进行排序
 - 遍历所有的runner。调用 run 方法

- 如果以上有异常，
 - 调用Listener 的 failed
- 调用所有监听器的 running 方法 listeners.running(context); 通知所有的监听器 running
- running如果有问题。继续通知 failed 。调用所有 Listener 的 failed; 通知所有的监听器 failed

Java

 复制代码

```
1  public interface Bootstrapper {
2
3      /**
4       * Initialize the given {@link BootstrapRegistry} with any required regist
5       * @param registry the registry to initialize
6       */
7      void initialize(BootstrapRegistry registry);
8
9  }
```

atguigu.com

atguigu.com

2

Java | 复制代码

```
1  @FunctionalInterface
2  public interface ApplicationRunner {
3
4      /**
5       * Callback used to run the bean.
6       * @param args incoming application arguments
7       * @throws Exception on error
8       */
9      void run(ApplicationArguments args) throws Exception;
10
11 }
```

atguigu.com

atguigu.com

Java

 复制代码

```
1  @FunctionalInterface
2  public interface CommandLineRunner {
3
4      /**
5       * Callback used to run the bean.
6       * @param args incoming main method arguments
7       * @throws Exception on error
8       */
9      void run(String... args) throws Exception;
10
11 }
```

2、Application Events and Listeners

<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-application-events-and-listeners> <<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-application-events-and-listeners>>

ApplicationContextInitializer

ApplicationListener

SpringApplicationRunListener

3、ApplicationRunner 与 CommandLineRunner