

Spring Boot原理及启动流程

 cs.xieyonghui.com/java/spring-boot-start-process_136.html

2019年7月23日



以IDE环境下的Spring Boot应用启动为例，假如，有一个DemoApplication类的主方法中的代码是这样的：

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

那么，运行DemoApplication类的主方法后，Spring Boot启动程序会做如下事情：

查看Spring Boot当前运行环境

查看当前Spring Boot属于哪类应用，SERVLET（Web应用），REACTIVE（Reactive Web应用），NONE（标准应用）。然后，根据应用程序对应的类型，使用不同的spring application context。SERVLET、REACTIVE、NONE 依次对应以下三种application context：

```
org.springframework.web.context.WebApplicationContext;  
org.springframework.boot.web.reactive.context.ReactiveWebApplicationContext;  
org.springframework.web.context.ConfigurableWebApplicationContext;
```

具体实现可以查看以下类中的方法：

```
org.springframework.boot.WebApplicationContext.deduceFromClasspath;
```

读取Spring Boot默认配置项

Spring Boot内置了许多默认配置项，在Spring Boot启动过程，最重要的两类配置项，会在这个阶段加载：

```
org.springframework.context.ApplicationContextInitializer;  
org.springframework.context.ApplicationListener
```

一类用来进行一系列的初始化工作，一类用来发布各种事件。两类配置项对应的具体类清单，可以从下面这组文件中找到：

spring-boot-x.y.z.jar META-INF/spring.factories spring-boot-autoconfigure-x.y.z.jar
META-INF/spring.factories spring-beans-x.y.z.jar META-INF/spring.factories

基本上会包含以下类文件：

ApplicationContextInitializer

```
[org.springframework.boot.context.ConfigurationWarningsApplicationContextInitializer,  
org.springframework.boot.context.ContextIdApplicationContextInitializer,  
org.springframework.boot.context.config.DelegatingApplicationContextInitializer,  
org.springframework.boot.web.context.ServerPortInfoApplicationContextInitializer,  
org.springframework.boot.autoconfigure.SharedMetadataReaderFactoryContextInitializer,  
org.springframework.boot.autoconfigure.logging.ConditionEvaluationReportLoggingListener]
```

ApplicationListener

```
[org.springframework.boot.context.config.ConfigFileApplicationListener,  
org.springframework.boot.context.config.AnsiOutputApplicationListener,  
org.springframework.boot.context.logging.LoggingApplicationListener,  
org.springframework.boot.context.logging.ClasspathLoggingApplicationListener,  
org.springframework.boot.autoconfigure.BackgroundPreinitializer,  
org.springframework.boot.context.config.DelegatingApplicationListener,  
org.springframework.boot.builder.ParentContextCloserApplicationListener,  
org.springframework.boot.ClearCachesApplicationListener,  
org.springframework.boot.context.FileEncodingApplicationListener,  
org.springframework.boot.liquibase.LiquibaseServiceLocatorApplicationListener]
```

Spring Boot会通过如下API获取以上清单中的Class并创建实例，为后续流程使用。

```
SpringApplication.getSpringFactoriesInstances
```

如果有自定义的配置项加入到这个阶段，并对执行顺序有要求，可能会用到如下注解。

```
org.springframework.core.annotation.Order
```

确定Spring Boot应用启动入口

在这阶段，Spring Boot启动程序会通过获取当前线程的调用栈，遍历调用栈找到以main命名的方法，并返回main方法所在的Class，确定启动程序的入口。

```
StackTraceElement[] stackTrace = new RuntimeException().getStackTrace();
```

具体代码可以参考以下类中的方法：

```
org.springframework.boot.SpringApplication.deduceMainApplicationClass
```

为Spring Boot配置Headless项

从J2SE 1.4为awt包加入了java.awt.headless选项，防止java.awt.*包下的某些类运行在没显示器和键盘的设备上时抛出HeadlessException异常。默认设置为true。

```
java.awt.headless=true
```

具体参考[Oracle文档关于headless的说明](#)。

发布Spring Boot启动事件

Spring Boot启动过程各类事件，统一由SpringApplicationRunListeners进行管理，在这样阶段它会发送一个启动事件：

```
org.springframework.boot.context.event.ApplicationStartingEvent;
```

默认会由去将事件分发到各个Listener，

```
org.springframework.boot.context.event.EventPublishingRunListener
```

默认会使用当前的启动线程去发布事件；也就是说，监听器有机会阻塞启动过程，如果，为应用配置了Executor，则事件会通过Executor异步执行，能够避免此类情况发生，但会增加开销，异步通常有以下两种子类可选：

```
org.springframework.core.task.SimpleAsyncTaskExecutor;  
org.springframework.core.task.SyncTaskExecutor;
```

注：在执行前会统一调用GenericApplicationListener.supportsSourceType方法，过滤掉与此事件不相关的监听器。

读取Spring Boot应用程序配置数据

根据在第一步时确定的应用运行环境，Spring Boot会创建对应的环境配置对象：

```
SpringApplication.getOrCreateEnvironment
```

标准的环境对象，把环境配置项分在四个命名空间下：

```
StandardEnvironment {activeProfiles=[], defaultProfiles=[default],  
propertySources=[PropertiesPropertySource {name='systemProperties'},  
SystemEnvironmentPropertySource {name='systemEnvironment'}}]}
```

解析配置文件的前提，是正确设置数据类型转换，默认支持的转换类型，可以通过查看以下类型转换器确定：

```
org.springframework.core.convert.support.DefaultConversionService;  
org.springframework.boot.convert.ApplicationConversionService;
```

发送ApplicationEnvironmentPreparedEvent事件

在Spring Boot读取配置文件的数据时，会发出ApplicationEnvironmentPreparedEvent事件，

```
org.springframework.boot.context.event.ApplicationEnvironmentPreparedEvent
```

监听此事件的监听器有机会通过getEnvironment方法拿到环境配置数据。

设置Spring Boot日志项

在这个阶段Spring Boot允许给日志加点料，改变一下日志输出时的字体颜色，以下是以彩色输出日志：

```
spring.output.ansi.enabled=ALWAYS
```

绑定环境变量数据

Spring提供环境变量和配置文件的绑定方式多到眼花缭乱，这里就不细说了。

Spring Boot Banner输出

启动过程中经典的spring boot图案，就在此时通过以下类进行输出：

```
org.springframework.boot.SpringBootBanner
```

依赖于灵活的扩展性，可以直接在resource目录下放置图片做为logo，Spring Boot能对图片进行转换。名称为以下格式：banner.[gif, jpg, png]；或通过spring.banner.image.location指定图片路径。或通过banner.txt放置文字。具体可以参考以下两个类：

```
org.springframework.boot.SpringApplicationBannerPrinter org.springframework.boot.ImageBanner
```

创建ApplicationContext

还是那熟悉的spring味道，根据环境的不同，有三种不同的上下文对象可选。

```
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext（  
目）org.springframework.boot.web.reactive.context.AnnotationConfigReactiveWebServerApplicationCo  
目）org.springframework.context.annotation.AnnotationConfigApplicationContext（标准环境，非web项  
目）
```

创建异常处理机制

此阶段建立起异常处理机制，负责异常处理的是以下类：

```
org.springframework.boot.diagnostics.FailureAnalyzers
```

初始化Spring Boot应用程序

在这个阶段会初始化应用程序上下文对象，在META-INF/spring.factories配置文件中默认的一系统初始化类粉墨登场了。

发布ApplicationContextInitializedEvent事件

又要发布事件了，这次是ApplicationContextInitializedEvent事件，还是由EventPublishingRunListener进行分发，监听者有机会通过getApplicationContext获取应用程序上下文。

加载应用程序

此时，会把应用程序入口类加入到Spring容器中。示例中为DemoApplication。

发布EventPublishingRunListener事件

发布ApplicationPreparedEvent事件，监听者通过getApplicationContext方法能够拿到应用程序上下文对象。如：

```
org.springframework.context.annotation.AnnotationConfigApplicationContext;
```

开启依赖注入之旅

经典依赖注入就发现在这个阶段，开始打针了各个对象准备好了没，准备好了就要开始表演了。

发布ContextRefreshedEvent事件

由ApplicationStartingEvent发起的ContextRefreshedEvent事件，通过EventPublishingRunListener发布出去，通知监听者依赖注入之旅结束。

发布ApplicationStartedEvent事件

由ApplicationStartingEvent发起的ApplicationStartedEvent事件，通过EventPublishingRunListener分发出去，通知监听者应用程序启动完成。

发布ApplicationReadyEvent事件

由ApplicationStartingEvent发起的ApplicationReadyEvent事件，通过EventPublishingRunListener，通知监听者应用程序已就绪。

注：应用程序自然关闭时会发布ContextClosedEvent事件。