# 极速开发框架SpringBoot及源码分析

主讲：Cat 老师

**北京动力节点教育科技有限公司**  
**2020 • 北京**

**动力节点版权所有，禁止私自传播**

## 快速回顾

What is SpringBoot？

SpringBoot开发运行环境要求

Springboot程序的几种创建方式

SpringBoot开发java程序

SpringBoot开发web程序

SpringBoot main方法分析

SpringBoot项目约定的代码结构

SpringBoot 项目pom.xml继承结构

SpringBoot 项目pom.xml修改默认版本

SpringBoot 整合mybatis

SpringBoot 整合jsp视图展示（也支持一些模板技术）

SpringBoot 项目pom.xml热部署插件devtools

JRebel （这是idea插件，收费的，需要破解）

SpringBoot开发最佳实践

SpringBoot敏感配置信息加密

SpringBoot数据库连接池

SpringBoot跨域资源共享CORS

SpringBoot静态资源处理

SpringBoot核心配置文件（Duration/DataSize单位类型、随机值、占位符、多环境、4种配置文件读取方式）

SpringBoot线程池异步调用（@Async、手动编码）

SpringBoot集成Swagger（OAS、springfox、swagger的关系）

SpringBoot集成websocket

注册Servlets, Filters, Listeners作为Spring Beans

SpringBoot使用拦截器

SpringBoot内嵌Web服务器

SpringBoot打包部署

SpringBoot配置SSL（https）

SpringBoot全局异常处理

SpringBoot 404页面处理

课堂上未讲解部分：

SpringBoot多数据源

SpringBoot服务监控 springboot actuator、springboot admin；

springboot整合thymeleaf模板

SpringBoot RestTemplate

SpringBoot WebClient

Validation验证，@Vilidate注解验证

SpringBoot统一缓存处理

SpringBoot定时任务quartz

## SpringBoot源码构建

1. 安装和配置好JDK环境环境变量（JDK8+）；
2. 安装和配置好Gradle；官网：<https://gradle.org>

安装gradle（需要先安装JDK），Gradle直接解压到一个目录即完成安装；

Gradle配置环境变量，配置GRADLE\_HOME 和Path；

使用gradle -v 验证Gradle环境是否正常；

Springboot源码从2.3.1版本开始由Maven改为了Gradle；

3、当前我使用的是IDEA 2020.1.3 旗舰版

File -->Setting-->Editor-->File Encodings -->全部改为UTF-8

安装代码统计插件：Statistic

安装英文翻译插件：ECTranslation

4、下载SpringBoot源码：<https://github.com/spring-projects/spring-boot>

5、解压下载下来的最新SpringBoot源码：spring-boot-2.3.1.RELEASE.zip

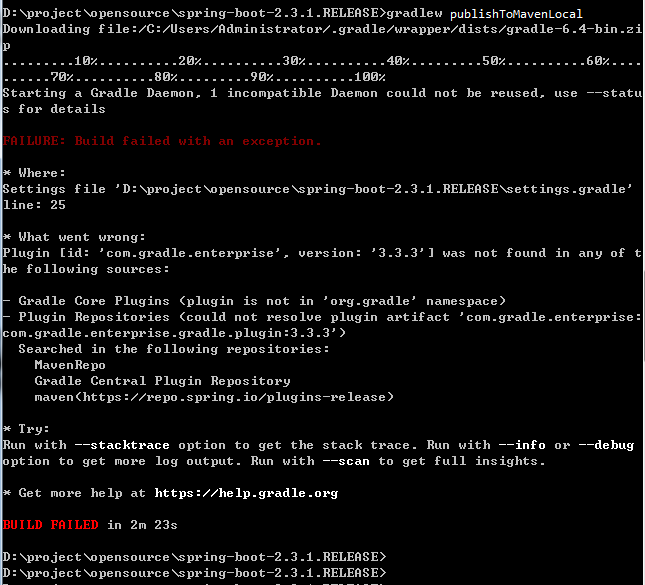
6、进入解压后的目录，比如：D:\project\opensource\spring-boot-2.3.1.RELEASE

7、阅读解压后的源码目录下的README.adoc文件，里面告诉了我们如何构建源码；

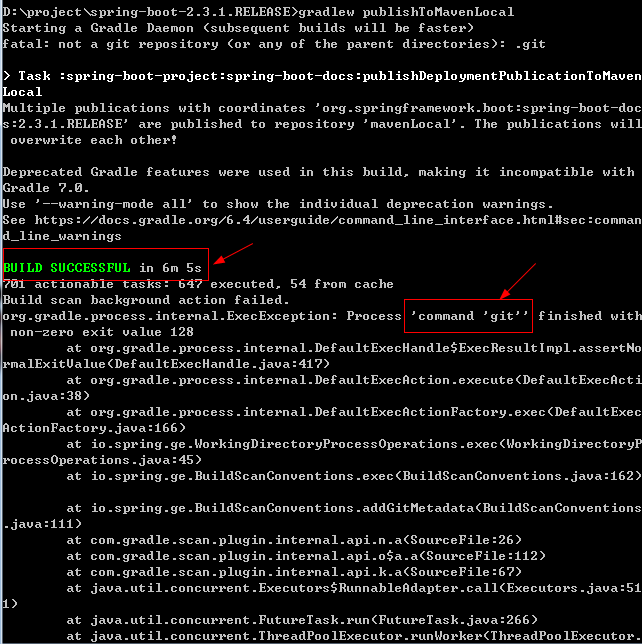
8、进入源码目录的dos窗口，执行：gradlew publishToMavenLocal

修改默认下载gradle文件路径

distributionUrl=file:///C:/Users/Administrator/.gradle/wrapper/gradle-6.4-bin.zip



执行的时候，可能会遇到错误，主要是因为一些插件从gradle网站下载失败导致的，此时需要翻墙软件（科学上网），才能下载成功；

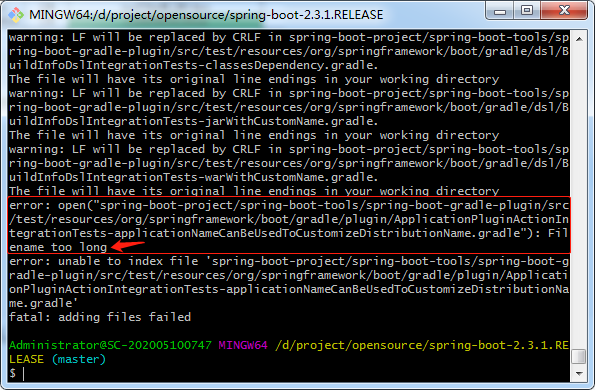


出现上面的 git 的错误，是因为springboot源码构建需要保证源码是处于一个git仓库中，所以我们把源码纳入本地仓库管理中，依次执行如下命令：

创建.git目录： git init

将springboot源码目录加入暂存区： git add \*

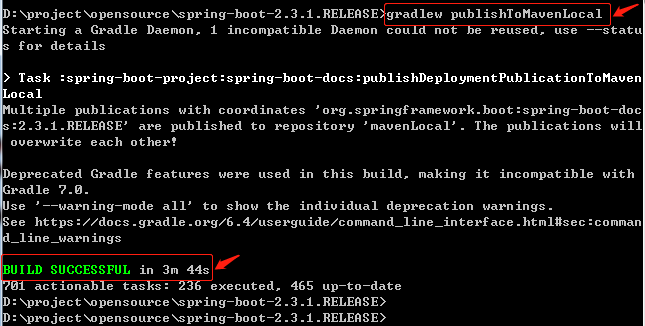
提交springboot源码提交到本地Git仓库： git commit -m 'springboot'



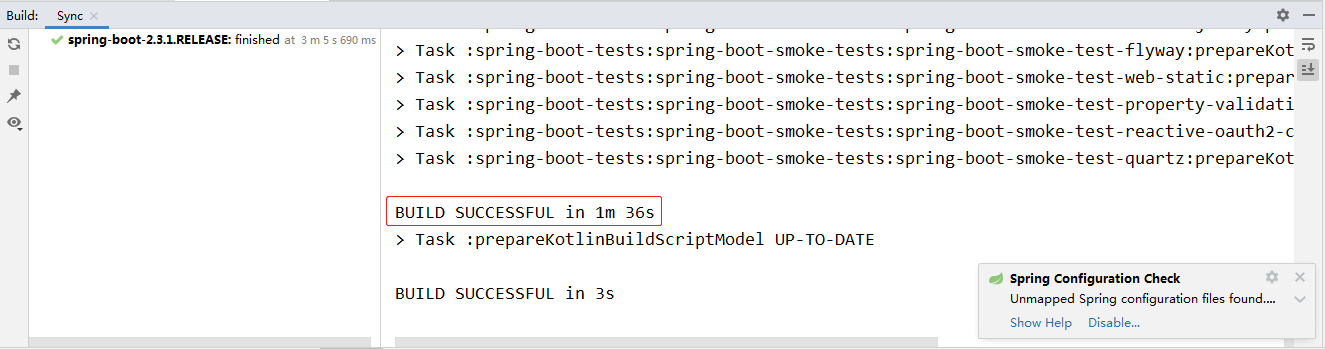
出现error: open:(.....) Filename too long错误，是因为路径太长了，默认是260个字符，解决方法：

git config core.longpaths true

接下来再进行构建：gradlew publishToMavenLocal



然后导入idea，等待加载完成，如果是第一次导入，需要下载一个依赖及插件，可能需要等待一段时间，会慢一些，如果某些插件下载失败，也会报错，通过启动翻墙软件（科学上网），这样保证从国外一些网站下载依赖和插件没有问题，然后构建成功；



显示成功，接下来就可以开始在源码的基础上创建一个测试项目，进行源码的阅读；

如果遇到源码发乳Idea中报一个format的格式错误的话，请dos窗口下执行：

gradlew -p buildSrc format

## 验证SpringBoot源码环境是否正常

创建一个spring-boot-zweb模块，编写springboot代码并运行，验证springboot源码环境构建是否正常；

加入springboot的依赖，根据需要添加依赖，举例如下：

dependencies **{** testCompile group: 'junit', name: 'junit', version: '4.12'  
  
 api(platform(project(":spring-boot-project:spring-boot-parent")))  
 api(project(":spring-boot-project:spring-boot-starters:spring-boot-starter-web"))  
**}**

最后运行spring-boot-zweb的main方法测试一下springboot环境是否正常？

一般情况下，第一次运行会比较慢，后面的运行会变快；

## 阅读源码指导原则

1、寻找入口

2、断点跟踪

3、先粗后细

4、精略结合

5、猜想验证

我们有了前面大量的源码阅读经验，相信大家对源码阅读并不感到陌生；

## SpringBoot启动流程源码分析

SpringBoot启动流程（SpringBoot启动原理）

启动流程主要分为三个部分：

第一部分进行SpringApplication的初始化模块，配置一些基本的环境变量、资源、构造器、监听器；（了解一下）

第二部分实现了应用具体的启动方案，包括启动流程的监听模块、加载配置环境模块、及核心的创建Spring上下文环境模块；（掌握）

第三部分是自动化配置模块，该模块是SpringBoot自动配置的核心；（掌握）

**宏观步骤：（概括为18个小步骤）**

<https://www.processon.com/view/link/5dc15b71e4b0335f1e4eb6fe>

1、创建并启动计时监控类StopWatch

2、初始化应用上下文和异常报告集合（ConfigurableApplicationContext和Collection<SpringBootExceptionReporter>）

3、设置系统属性 configureHeadlessProperty()

4、创建Spring运行的监听器并启动

5、初始化默认应用参数类

6、准备Spring环境

7、创建beanner打印类

8、创建应用Spring上下文（也就是spring容器）

9、准备异常报告器

10、准备应用上下文（也就是spring容器）

11、刷新上下文（也就是spring容器）

12、应用上下文刷新后置处理（也就是spring容器）

13、停止计时监控类

14、输出日志信息

15、发布应用上下文启动完成事件

16、执行所有Runner运行器（如果实现CommandLineRunner接口）

17、发布应用上下文就绪事件

18、完成返回容器对象

## SpringBoot启动流程源码分析

1、入口main方法执行；

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

2、执行run方法；

public static ConfigurableApplicationContext run(Class<?> primarySource,

String... args) {

//调用run() 方法

return run(new Class<?>[] { primarySource }, args);

}

1. 看一下run(new Class<?>[] { primarySource }, args);方法；

public static ConfigurableApplicationContext run(Class<?>[] primarySources,

String[] args) {

return new SpringApplication(primarySources).run(args);

}

//这个run方法主要做两件事情

//1、new了一个SpringApplication 这么一个对象；

1. 把main方法这个类赋给一个成员变量；
2. 判断是java项目还是web项目；
3. 设置了一个初始化器和事件发送的监听器；

//2、然后执行SpringApplication对象的run()方法；

4、先看做的第一件事情，new了一个SpringApplication对象

public SpringApplication(ResourceLoader resourceLoader, Class<?>... primarySources) {

this.resourceLoader = resourceLoader;

Assert.notNull(primarySources, "PrimarySources must not be null");

//1、先把主类保存起来

this.primarySources = new LinkedHashSet<>(Arrays.asList(primarySources));

//2、判断运行项目的类型

this.webApplicationType = WebApplicationType.deduceFromClasspath();

//3、扫描当前路径下META-INF/spring.factories文件的，加载ApplicationContextInitializer接口实例

setInitializers((Collection) getSpringFactoriesInstances(ApplicationContextInitializer.class));

//4、扫描当前路径下META-INF/spring.factories文件的，加载ApplicationListener接口实例

setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class));

//判断main方法的类

this.mainApplicationClass = deduceMainApplicationClass();

}

这里面用到了一种类似于Java SPI机制的方式扫描META-INF/spring.factories这个文件，并且加载 ApplicationContextInitializer、ApplicationListener 接口实例；

（1）ApplicationContextInitializer 这个类当springboot上下文Context初始化完成后会调用；

（2）ApplicationListener 当springboot启动时事件change后都会触发

我们可以自定义实现ApplicationContextInitializer、ApplicationListener 接口，然后运行时会被触发执行；

5、然后我们看做的第二件事情，执行new出来的SpringApplication对象的run()方法；

public ConfigurableApplicationContext run(String... args) {

<!--这是一个计时器，不是很重要-->

StopWatch stopWatch = new StopWatch();

stopWatch.start();

ConfigurableApplicationContext context = null;

Collection<SpringBootExceptionReporter> exceptionReporters = new ArrayList<>();

<!--这是设置了一些环境变量信息，不是很重要-->

configureHeadlessProperty();

<!--获取事件监听器SpringApplicationRunListener类型，并且执行starting()方法-->

SpringApplicationRunListeners listeners = getRunListeners(args);

listeners.starting();

try {

<!--把参数args封装成DefaultApplicationArguments，了解一下即可-->

ApplicationArguments applicationArguments = new DefaultApplicationArguments(

args);

<!--这个是准备环境，并且把环境跟spring上下文绑定好，并且执行environmentPrepared()方法-->

ConfigurableEnvironment environment = prepareEnvironment(listeners,

applicationArguments);

<!--判断一些环境的值，并设置一些环境的值-->

configureIgnoreBeanInfo(environment);

<!--打印banner-->

Banner printedBanner = printBanner(environment);

<!--创建spring ioc上下文，根据项目类型创建上下文-->

context = createApplicationContext();

<!--获取异常报告事件监听-->

exceptionReporters = getSpringFactoriesInstances(

SpringBootExceptionReporter.class,

new Class[] { ConfigurableApplicationContext.class }, context);

<!--准备spring ioc上下文，执行完成后调用contextPrepared()方法,

contextLoaded()方法-->

prepareContext(context, environment, listeners, applicationArguments,

printedBanner);

<!--这是刷新和创建spring ioc容器，这里会扫描bean定义并初始化单例bean对象-->

//这个refreshContext()不仅创建bean，还启动了内嵌的web容器，非常重要

refreshContext(context);

<!--什么事都没有做-->

afterRefresh(context, applicationArguments);

stopWatch.stop();

if (this.logStartupInfo) {

new StartupInfoLogger(this.mainApplicationClass)

.logStarted(getApplicationLog(), stopWatch);

}

<!--执行ApplicationRunListeners中的started()方法-->

listeners.started(context);

<!--执行Runner（ApplicationRunner和CommandLineRunner）-->

callRunners(context, applicationArguments);

}

catch (Throwable ex) {

handleRunFailure(context, listeners, exceptionReporters, ex);

throw new IllegalStateException(ex);

}

listeners.running(context);

return context;

}

6、接下来我们重点看一下run方法里面的几个重要步骤，首先看看红色标记的创建spring容器上下文context = createApplicationContext();这段代码主要是根据项目类型创建Spring上下文，并且会注入几个核心组件类；

protected ConfigurableApplicationContext createApplicationContext() {

Class<?> contextClass = this.applicationContextClass;

if (contextClass == null) {

try {

switch (this.webApplicationType) {

case SERVLET:

contextClass = Class.forName(DEFAULT\_SERVLET\_WEB\_CONTEXT\_CLASS);

break;

case REACTIVE:

contextClass = Class.forName(DEFAULT\_REACTIVE\_WEB\_CONTEXT\_CLASS);

break;

default:

contextClass = Class.forName(DEFAULT\_CONTEXT\_CLASS);

}

}

catch (ClassNotFoundException ex) {

throw new IllegalStateException(

"Unable create a default ApplicationContext, please specify an ApplicationContextClass", ex);

}

}

return (ConfigurableApplicationContext) BeanUtils.instantiateClass(contextClass);

}

public AnnotationConfigServletWebServerApplicationContext() {

//1：会去注入一些spring核心组件

this.reader = new AnnotatedBeanDefinitionReader(this);

this.scanner = new ClassPathBeanDefinitionScanner(this);

}

Web类型项目创建上下文对象 AnnotationConfigServletWebServerApplicationContext，这里会把 ConfigurationClassPostProcessor 、

AutowiredAnnotationBeanPostProcessor 等一些核心组件加入到Spring容器，这其实已经是spring ioc容器的源码，如果了解spring ioc源码，那么对springboot源码的阅读会有很大的帮助；

7、然后再看一下run方法里面另一个比较重要的步骤，红色标记的创建和刷新spring容器refreshContext()方法，

refreshContext(context) 这个方法，该方法启动spring的代码加载了bean，还启动了内嵌的web容器；

private void refreshContext(ConfigurableApplicationContext context) {

//刷新和创建spring容器

refresh(context);

if (this.registerShutdownHook) {

try {

context.registerShutdownHook();

}

catch (AccessControlException ex) {

// Not allowed in some environments.

}

}

}

protected void refresh(ApplicationContext applicationContext) {

Assert.isInstanceOf(AbstractApplicationContext.class, applicationContext);

//进入refresh()方法

((AbstractApplicationContext) applicationContext).refresh();

}

接下来进入到Spring ioc容器刷新和创建的12个步骤的源码中，AbstractApplicationContext - >refresh()方法，在这12个步骤中，这里主要看一下onRefresh() 这个方法，但是发现这个方法里面什么都没有，显然是一个钩子方法，它会钩到它子类重写onRefresh()方法，所以去看子类里面的onRefresh()方法；

由于我们这里是一个Web项目，所以就可以看到 ServletWebServerApplicationContext 这个类，这个类下面的 onRefresh() 方法：

protected void onRefresh() {

super.onRefresh();

try {

//看到了内嵌web容器的影子

createWebServer();

}

catch (Throwable ex) {

throw new ApplicationContextException("Unable to start web server", ex);

}

}

private void createWebServer() {

WebServer webServer = this.webServer;

ServletContext servletContext = getServletContext();

if (webServer == null && servletContext == null) {

//1、获取webServerFactory

ServletWebServerFactory factory = getWebServerFactory();

this.webServer = factory.getWebServer(getSelfInitializer());

}

else if (servletContext != null) {

try {

getSelfInitializer().onStartup(servletContext);

}

catch (ServletException ex) {

throw new ApplicationContextException("Cannot initialize servlet context",

ex);

}

}

initPropertySources();

}

继续看下getWebServletFactory() 这个方法，这里面选择出哪种类型的web容器

protected ServletWebServerFactory getWebServerFactory() {

// Use bean names so that we don't consider the hierarchy

String[] beanNames = getBeanFactory()

.getBeanNamesForType(ServletWebServerFactory.class);

if (beanNames.length == 0) {

throw new ApplicationContextException(

"Unable to start ServletWebServerApplicationContext due to missing "

+ "ServletWebServerFactory bean.");

}

if (beanNames.length > 1) {

throw new ApplicationContextException(

"Unable to start ServletWebServerApplicationContext due to multiple "

+ "ServletWebServerFactory beans : "

+ StringUtils.arrayToCommaDelimitedString(beanNames));

}

return getBeanFactory().getBean(beanNames[0], ServletWebServerFactory.class);

}

再回头去看factory.getWebServer(getSelfInitializer()) ，转到定义就会看到很熟悉的名字tomcat

public WebServer getWebServer(ServletContextInitializer... initializers) {

//看到了我们熟悉的tomcat

Tomcat tomcat = new Tomcat();

File baseDir = (this.baseDirectory != null ? this.baseDirectory

: createTempDir("tomcat"));

tomcat.setBaseDir(baseDir.getAbsolutePath());

Connector connector = new Connector(this.protocol);

tomcat.getService().addConnector(connector);

customizeConnector(connector);

tomcat.setConnector(connector);

tomcat.getHost().setAutoDeploy(false);

configureEngine(tomcat.getEngine());

for (Connector additionalConnector : this.additionalTomcatConnectors) {

tomcat.getService().addConnector(additionalConnector);

}

prepareContext(tomcat.getHost(), initializers);

return getTomcatWebServer(tomcat);

}

内置的Servlet容器就是在onRefresh() 方法里面启动的，至此一个Servlet容器就启动OK了；

**总结：**

1、new了一个SpringApplication对象，使用一种类似于SPI机制的技术加载ApplicationContextInitializer、ApplicationListener 接口实例；（非重点）

2、调用SpringApplication.run() 方法，run方法里面大概有18个小步骤；

3、调用createApplicationContext()方法创建Spring IOC上下文对象，创建上下文对象同时会注册spring的核心组件类；

ConfigurationClassPostProcessor、AutowiredAnnotationBeanPostProcessor等；

4、调用refreshContext() 方法启动Spring容器（12步）和内置的Servlet容器；