链滴

SpringBoot- 启动流程分析

作者：Ethan

原文链接：https://ld246.com/article/1524815782559

来源网站：链滴

从SpringApplication.run();开始：

```
/**
 * Static helper that can be used to run a {@link
 * SpringApplication} from the
 * specified source using default settings.
 * @param source the source to load
 * @param args the application arguments (usually
 * passed from a Java main method)
 * @return the running {@link ApplicationContext}
 */
public static ConfigurableApplicationContext run(Object source, String... args) {
    return run(new Object[] { source }, args);
}```

进入重构run()方法

/**

- Static helper that can be used to run a {@link
- SpringApplication} from the
- specified sources using default settings and
- user supplied arguments.
- @param sources the sources to load
- @param args the application arguments (usually
- passed from a Java main method)
- @return the running {@link ApplicationContext}
```

```
*/
public static ConfigurableApplicationContext run(Object[] sources, String[] args) {
return new SpringApplication(sources).run(args);
}```
```

从这里可以看到首先创建了一个SpringApplication实例，然后在调用的其run()方法。首先我们先去建实例这一流程：

```
/**
 * Create a new {@link SpringApplication}
 * instance. The application context will load
 * beans from the specified sources (see {@link
 * SpringApplication class-level}
 * documentation for details. The instance can be
 * customized before calling
 * {@link #run(String...)}.
 * @param sources the bean sources
 * @see #run(Object, String[])
 * @see #SpringApplication(ResourceLoader,
 * Object...)
 */
public SpringApplication(Object... sources) {
    initialize(sources);
}```
```

可以看到其调用了initialize()方法

```
@SuppressWarnings({ "unchecked", "rawtypes" })
private void initialize(Object[] sources) {
if (sources != null && sources.length > 0) {
this.sources.addAll(Arrays.asList(sources));
}
this.webEnvironment = deduceWebEnvironment();
setInitializers((Collection) getSpringFactoriesInstances(
ApplicationContextInitializer.class));
setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class));
this.mainApplicationClass = deduceMainApplicationClass();
}```
```

从面的代码可以看到初始化过程做了以下几件事情

this.webEnvironment = deduceWebEnvironment();

这一个方法决定创建的是一个WEB应用还是一个SPRING的标准Standalone应用。如果入方法可以看其是怎么判断的：

```
private boolean deduceWebEnvironment() {
    for (String className : WEB_ENVIRONMENT_CLASSES) {
        if (!ClassUtils.isPresent(className, null)) {
            return false;
        }
    }
    return true;
}```

可以看到是根据org.springframework.util.ClassUtils的静态方法去判断classpath里面是否有WEB\_NVIRONMENT\_CLASSES包含的类，如果有都包含则返回true则表示启动一个WEB应用，否则返回false启动一个标准Spring的应用。然后通过代码：

```
private static final String[] WEB_ENVIRONMENT_CLASSES =


  { "javax.servlet.Servlet",
    "org.springframework.web.context.ConfigurableWebApplicationContext" };```
```

可以看到是否启动一个WEB应用就是取决于classpath下是否有javax.servlet.Servlet和org.springframework.web.context.ConfigurableWebApplicationContext。然后进入下一个阶段：

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

这个方法则是初始化classpath下的所有的可用的ApplicationContextInitializer

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

这个方法则是初使化classpath下的所有的可用的ApplicationListener

```
this.mainApplicationClass = deduceMainApplicationClass();

private Class<?> deduceMainApplicationClass() {
    try {
        StackTraceElement[] stackTrace = new RuntimeException().getStackTrace();
        for (StackTraceElement stackTraceElement : stackTrace) {
            if ("main".equals(stackTraceElement.getMethodName())) {
                return Class.forName(stackTraceElement.getClassName());
            }
        }
    }
    catch (ClassNotFoundException ex) {
        // Swallow and continue
    }
    return null;
}```
```

最后找出main方法的全类名并返回其实例并设置到SpringApplication的this.mainApplicationClass成初始化。然后调用SpringApplication实例的run方法来启动应用，代码如下：
```

```
/**
● Run the Spring application, creating and
● refreshing a new
● {@link ApplicationContext}.
● @param args the application arguments (usually
● passed from a Java main method)
● @return a running {@link ApplicationContext}
*/
public ConfigurableApplicationContext run(String... args) {
StopWatch stopWatch = new StopWatch();
stopWatch.start();
ConfigurableApplicationContext context = null;
configureHeadlessProperty();
SpringApplicationRunListeners listeners = getRunListeners(args);
listeners.started();
try {
ApplicationArguments applicationArguments = new DefaultApplicationArguments(
args);
ConfigurableEnvironment environment = prepareEnvironment(listeners,
applicationArguments);
Banner printedBanner = printBanner(environment);
context = createApplicationContext();
prepareContext(context, environment, listeners, applicationArguments,
printedBanner);
refreshContext(context);
afterRefresh(context, applicationArguments);
listeners.finished(context, null);
stopWatch.stop();
if (this.logStartupInfo) {
new StartupInfoLogger(this.mainApplicationClass)
.logStarted(getApplicationLog(), stopWatch);
}
return context;
}
catch (Throwable ex) {
handleRunFailure(context, listeners, ex);
throw new IllegalStateException(ex);
```

}

}```

由于过程比较长，则把详细说明放入下一篇博客进行说明，这里只做简要的过程说明run()方法的代码：蓝色部分为加载SpringApplicationRunListener对整个容器的初始化过程进行监听，这里先不做解，然后先观察剩下的几行的代码：

```
ApplicationArguments applicationArguments = new DefaultApplicationArguments(args); Conf
gurableEnvironment environment = prepareEnvironment(listeners, applicationArguments); Ba
ner printedBanner = printBanner(environment); context = createApplicationContext(); prepar
Context(context, environment, listeners, applicationArguments, printedBanner); refreshContex
(context); afterRefresh(context, applicationArguments);
```

首先是获取启动时传入参数args并初始化为ApplicationArguments对象

SpringApplication.run(Application.class, args);取这里传入值。

然后配置SpringBoot应用的环境：

```
ConfigurableEnvironment environment = prepareEnvironment(listeners, applicationArgument
);
```

下面的则打印标志这个方法不说明，因为没有什么实质性作用，反应到控制台则是以下的效果如果确想玩玩修改一下标志，那可以在项目的classpath下新建一个banner.txt文件，把想打印到控制台的数放到文件中即可。比如：

src/main/resources/banner.txt中加入以下内容：

```
_ _ _ _ __ __ _ _ ____ _ _ ___ |||||||\\//||||||||||/||||||||||\V
|||||||||/|||||||||} {||||||_||||/|||||||||||/^\|||||_|||||//|||____|||\
//\ \\ /||\ /||//|_|
```

那么启动的时候就可以看到些标识。

然后下面代码就是比较核心的：

```
context = createApplicationContext(); prepareContext(context, environment, listeners, applicat
onArguments, printedBanner); refreshContext(context); afterRefresh(context, applicationArg
ments);
```

首先是createApplicationContext()方法：

```
/**

 * Strategy method used to create the {@link
 * ApplicationContext}. By default this
 * method will respect any explicitly set
 * application context or application context
 * class before falling back to a suitable
 * default.
 * @return the application context (not yet
 * refreshed)
```

```
 * @see #setApplicationContextClass(Class)
 */
protected ConfigurableApplicationContext createApplicationContext() {
    Class<?> contextClass = this.applicationContextClass;
    if (contextClass == null) {
        try {
            contextClass = Class.forName(this.webEnvironment
                    ? DEFAULT_WEB_CONTEXT_CLASS : DEFAULT_CONTEXT_CLASS);
        }
        catch (ClassNotFoundException ex) {
            throw new IllegalStateException(
                    "Unable create a default ApplicationContext, "
                            + "please specify an ApplicationContextClass",
                    ex);
        }
    }
    return (ConfigurableApplicationContext) BeanUtils.instantiate(contextClass);
}```

可以看出根据这前初始化过程初始化的this.webEnvironment来决定初始化一个什么容器。如果class
ath下是否有javax.servlet.Servlet和
org.springframework.web.context.ConfigurableWebApplicationContext类，
则使用DEFAULT\_WEB\_CONTEXT\_CLASS初始化容器，如果不存在则用DEFAULT\_CONTEXT_CL
SS初始化容器。

public static final String DEFAULT_CONTEXT_CLASS = "org.springframework.context."

        + "annotation.AnnotationConfigApplicationContext";

```
/**
 * The class name of application context that
 * will be used by default for web
 * environments.
 */
public static final String DEFAULT_WEB_CONTEXT_CLASS = "org.springframework."
        + "boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext";```

以上是代码指定了容器的类名，最后通过Spring的工具类初始化容器类bean

BeanUtils.instantiate(contextClass);

完成容器的创建工作。然后执行以下的几个步骤完成整个容器的创建与启动以及bean的注入功能。

prepareContext(context, environment, listeners, applicationArguments, printedBanner);

以下这一句代码是实现spring-boot-starter-*的自动化配置的关键。包括spring.factories的加载以
自动化配置类的加载以及注入容器的功能可以参考《SpringBoot-自动配置源码解析》

refreshContext(context); afterRefresh(context, applicationArguments);

至此通过SpringBoot启动的容器已经构造完成。这里忽略了启动流程中的收集各种Listener，创建Env
ronment及Environment的初始化的因为这些地方都是SpringBoot提供的各种扩展点，后面的博客

详细的说明各个扩展点的用处以及扩展的方式。