【第54话：Spring Boot主方法后那些不为人知的秘密】

Hello 小伙伴吗，这结课给大家讲一下这几年面试官非常爱问的一个问题：“请说一下Spring Boot原理”。

随着Spring Boot 在Java项目中使用的越来越多，Spring Boot的面试题也越来越多。其中一个出现频率最高，同时也考验我们功底的问题就是“Spring Boot原理”。

想要搞懂Spring Boot原理，就必须从Spring Boot启动类开始进行分析。

我们开发任何一个Spring Boot项目，都会用到如下的启动类

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

从上面代码可以看出，Annotation定义（@SpringBootApplication）和类定义（SpringApplication.run）最为耀眼，所以要揭开SpringBoot的神秘面纱，我们要从这两位开始就可以了。

SpringBootApplication背后的秘密

@SpringBootApplication注解是Spring Boot的核心注解，它其实是一个组合注解：

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
...
}
```

虽然定义使用了多个Annotation进行了原信息标注，但实际上重要的只有三个Annotation：

- @Configuration（@SpringBootConfiguration点开查看发现里面还是应用了@Configuration）
- @EnableAutoConfiguration
- @ComponentScan

即 @SpringBootApplication = (默认属性)@Configuration + @EnableAutoConfiguration + @ComponentScan。

所以，如果我们使用如下的SpringBoot启动类，整个SpringBoot应用依然可以与之前的启动类功能对等：

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

@Configuration

这里的@Configuration对我们来说不陌生，它就是JavaConfig形式的Spring Ioc容器的配置类使用的那个@Configuration，SpringBoot社区推荐使用基于JavaConfig的配置形式，所以，这里的启动类标注了@Configuration之后，本身其实也是一个IoC容器的配置类。

@Configuration的注解类标识这个类可以使用Spring IoC容器作为bean定义的来源。

@ComponentScan

@ComponentScan这个注解在Spring中很重要，它对应XML配置中的元素，@ComponentScan的功能其实就是自动扫描并加载符合条件的组件（比如@Component和@Repository等）或者bean定义，最终将这些bean定义加载到IoC容器中。

我们可以通过basePackages等属性来细粒度的定制@ComponentScan自动扫描的范围，如果不指定，则默认Spring框架实现会从声明@ComponentScan所在类的package进行扫描。

注：所以SpringBoot的启动类最好是放在root package下，因为默认不指定basePackages。

@EnableAutoConfiguration

@EnableAutoConfiguration这个注解最为重要，所以放在最后来解读，这个注解我们在下面详细讲解，简单概括一下就是，借助@Import的支持，收集和注册特定场景相关的bean定义。

@EnableAutoConfiguration是借助@Import的帮助，将所有符合自动配置条件的bean定义加载到IoC容器，仅此而已！

```
    @EnableAutoConfiguration会根据类路径中的jar依赖为项目进行自动配置，如：添加了spring-boot-starter-web依赖，会自动添加Tomcat和Spring MVC的依赖，Spring Boot会对Tomcat和Spring MVC进行自动配置。

@EnableAutoConfiguration作为一个复合Annotation，其自身定义关键信息如下：
```

```
@SuppressWarnings("deprecation")
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
    ...
}
```

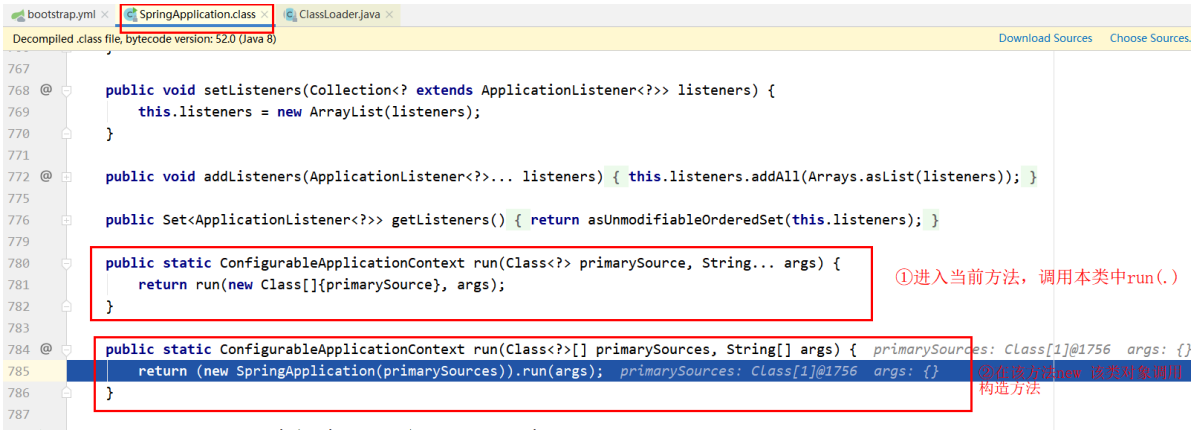
其中，最关键的要属@Import(EnableAutoConfigurationImportSelector.class)，借助EnableAutoConfigurationImportSelector，@EnableAutoConfiguration可以帮助SpringBoot应用将所有符合条件的@Configuration配置都加载到当前SpringBoot创建并使用的IoC容器。就像一只“八爪鱼”一样，借助于Spring框架原有的一个工具类：SpringFactoriesLoader的支持，@EnableAutoConfiguration可以智能的自动配置功效才得以大功告成！

SpringApplication.run()背后的秘密

每个SpringBoot程序都有一个主入口，也就是main方法，main里面调用SpringApplication.run()启动整个spring-boot程序，该方法所在类需要使用@SpringBootApplication注解，@SpringBootApplication包括三个注解上面已经说明

我们在SpringApplication.run()打断点

可知进入了SpringApplication的run()方法 如下所示



进入指定的构造方法如图所示

```
public SpringApplication(ResourceLoader resourceLoader, Class<?>... primarySources) {
    this.resourceLoader = resourceLoader; 获取资源加载器
    Assert.notNull(primarySources, message: "PrimarySources must not be null"); 断言判断主类不能为空
    this.primarySources = new LinkedHashSet<>(Arrays.asList(primarySources)); 记录主类
    this.webApplicationType = WebApplicationType.deduceFromClasspath(); 判断类型，可取值NONE，SERVLET，REACTIVE
    this.bootstrapRegistryInitializers = getBootstrapRegistryInitializersFromSpringFactories(); 获取到初始化器，默认为0
    setInitializers((Collection) getSpringFactoriesInstances(ApplicationContextInitializer.class)); 从META-INF/spring.factories中读取初始化器，共7个
    setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class)); 从META-INF/spring.factories中读取监听器，共8个
    this.mainApplicationClass = deduceMainApplicationClass(); 通过栈遍历判断main方法所在类
}
```

主要可以看到在读取的初始化容器共7个 个 监听器共8个 执行结束构造方法 然后调用run方法

run方法如图所示

```java
public ConfigurableApplicationContext run(String... args) {
    StopWatch stopWatch = new StopWatch(); 创建计时器
    stopWatch.start(); 启动计时器，记录当前时间
    DefaultBootstrapContext bootstrapContext = createBootstrapContext(); 创建默认独立上下文对象
    ConfigurableApplicationContext context = null; 声明上下文对象
    configureHeadlessProperty(); 设置系统属性java.awt.headless=true,表示简单图像处理器。多用于在缺少显示器、键盘、鼠标等情况下。一些工具需要设置该值为true
    SpringApplicationRunListeners listeners = getRunListeners(args); 获取到所有监听器
    listeners.starting(bootstrapContext, this.mainApplicationClass); 启动所有监听器
    try {
        ApplicationArguments applicationArguments = new DefaultApplicationArguments(args); 解析命令行参数，存储到ApplicationArguments中
        ConfigurableEnvironment environment = prepareEnvironment(listeners, bootstrapContext, applicationArguments); 根据默认上下文，监听器和命令行参数准备配置环境
        configureIgnoreBeanInfo(environment); 设置环境中需要忽略的参数
        Banner printedBanner = printBanner(environment); 打印Banner
        context = createApplicationContext(); 根据应用类型选择创建什么样的上下文对象
        context.setApplicationStartup(this.applicationStartup); Spring 5.3新加功能，负责记录应用执行步骤和时间
        prepareContext(bootstrapContext, context, environment, listeners, applicationArguments, printedBanner); 把所有内容都设置到上下文中
        refreshContext(context); 注册钩子，刷新上下文。如果是web项目，此步骤执行完成后Tomcat启动成功
        afterRefresh(context, applicationArguments); 刷新成功后的回调
        stopWatch.stop(); 停止计时器
        if (this.logStartupInfo) { 判断是否需要打印启动信息
            new StartupInfoLogger(this.mainApplicationClass).logStarted(getApplicationLog(), stopWatch); 打印启动信息
        }
        listeners.started(context); 监听器监听上下文，StartupStep记录执行步骤。
        callRunners(context, applicationArguments); 启动所有运行器
    }
    catch (Throwable ex) { 操作失败时执行
        handleRunFailure(context, ex, listeners);
        throw new IllegalStateException(ex);
    }

    try {
        listeners.running(context); 启动完成时调用，监听器处于运行状态
    }
    catch (Throwable ex) { 操作失败时执行
        handleRunFailure(context, ex, listeners: null);
        throw new IllegalStateException(ex);
    }
    return context; 返回上下文对象，项目类型为SERVLET时上下文对象为AnnotationConfigServletWebServerApplicationContext
}
```

流程总结（面试回答内容示范）

SpringBoot项目加载当前类注解@SpringBootApplication

1.@Configuration（@SpringBootConfiguration点开查看发现里面还是应用了@Configuration）

2.@EnableAutoConfiguration

3.@ComponentScan

SpringBoot项目通过SpringApplication.run()作为启动入口

1. 进入SpringApplication构造方法，从META-INF/spring.factories中读取初始化和监听器，并记录主类。
2. 记录当前时间后创建负责启动Spring应用程序的DefaultBootstrapContext
3. 声明Spring容器对象ConfigurableApplicationContext
4. 启动Headless服务器，设置系统属性
5. 创建监听器，并记录容器处于starting状态。
6. 准备环境Environment并打印Banner
7. 执行标准容器启动流程：创建容器(create) -> 准备容器启动的环境(prepare) -> 启动容器(refresh) -> 容器启动后收尾工作(after-refresh)
8. 打印容器启动时间，监听器切换为started状态
9. 调用容器运行过程中所涉及到所有启动类Runner
10. 监听器切换为ready状态。到此容器正式启动成功。