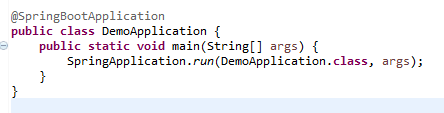
# SpringBoot启动原理



## 注解@SpringBootApplication

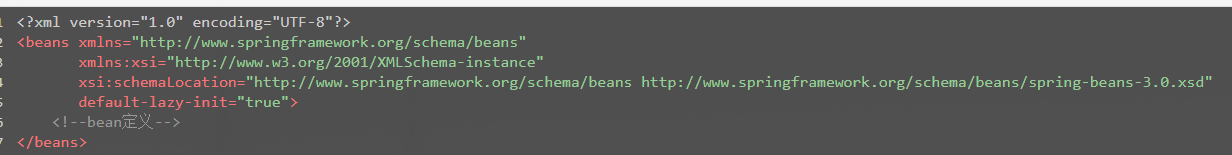
作用：相当于@Configuration @EnableAutoConfiguration @ComponentScan 的集合

* 1. 注解@Configuration -- **将启动类转换成配置类**

JavaConfig形式的Spring IOC容器的配置类使用@Configuration。启动类标注了@Configuration之后，本身其实也是一个IOC容器的配置类。

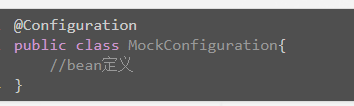
1. XML和config配置对比

（基于XML的配置）



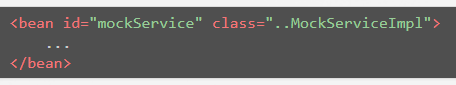
（基于JavaConfig的配置）

任何一个标注了@Configuration的Java类定义都是一个JavaConfig配置类。

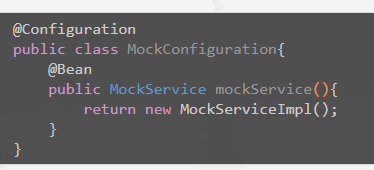


1. 注册bean定义层面

（基于XML的配置）

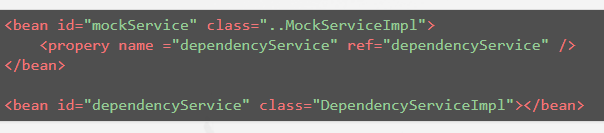


（基于JavaConfig的配置）



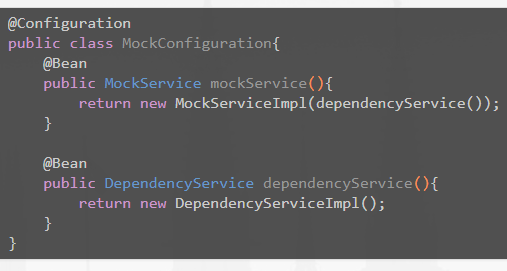
1. 依赖注入关系层面

（基于XML的配置）



（基于JavaConfig的配置）

如果一个bean的定义依赖其他bean,则直接调用对应的JavaConfig类中依赖bean的创建方法就可以了。



1.2 注解@ComponentScan -- **自动扫描符合条件的组件**

(1) @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。**

(2) 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。NJHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH5

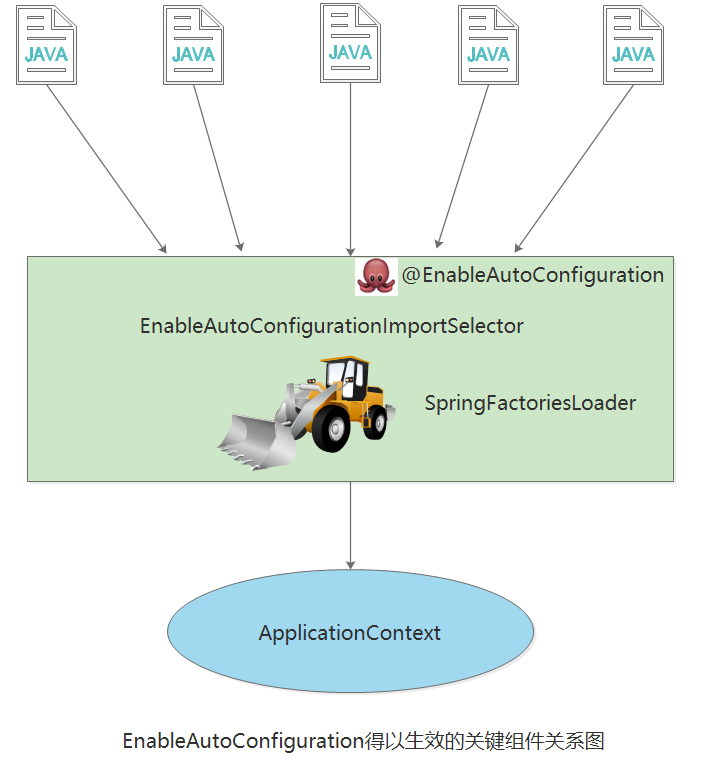
**所以SpringBoot的启动类最好是放在root package下，因为默认不指定basePackages**。

1.3 注解@EnableAutoConfiguration -- **将bean加载到IOC容器**

(1) 作用：@EnableAutoConfiguration也是借助@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。

(2) 原理：@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。



1.4 自动配置幕后英雄：SpringFactoriesLoader详解

(1) 作用：SpringFactoriesLoader属于Spring框架私有的一种扩展方案，其主要功能就是**从指定的配置文件META-INF/spring.factories加载配置**。

(2) 原理：配合@EnableAutoConfiguration使用的话，它更多是提供一种配置查找的功能支持，即根据@EnableAutoConfiguration的完整类名

org.springframework.boot.autoconfigure.EnableAutoConfiguration作为查找的Key,获取对应的一组@Configuration类

(3) @EnableAutoConfiguration自动配置：

从classpath中搜寻所有的META-INF/spring.factories配置文件，并将其中org.springframework.boot.autoconfigure.EnableautoConfiguration对应的配置项通过反射（Java Refletion）实例化为对应的标注了@Configuration的JavaConfig形式的IOC容器配置类，然后

//不理解，需要结合Spring框架来看

## 2. SpringApplication.run()执行流程

1. 如果我们使用的是SpringApplication的静态run方法，那么，这个方法里面首先要创建一个SpringApplication对象实例，然后调用这个创建好的SpringApplication的实例方法。在SpringApplication实例初始化的时候，它会提前做几件事情：

a) 根据classpath里面是否存在某个特征类

（org.springframework.web.context.ConfigurableWebApplicationContext）来决定是否应该创建一个为Web应用使用的ApplicationContext类型。

1. 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationContextInitializer。

c) 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationListener。

1. 推断并设置main方法的定义类。
2. SpringApplication实例初始化完成并且完成设置后，就开始执行run方法的逻辑了，方法执行伊始，首先遍历执行所有通过SpringFactoriesLoader可以查找到并加载的SpringApplicationRunListener。调用它们的started()方法，告诉这些

SpringApplicationRunListener，“嘿，SpringBoot应用要开始执行咯！”。

1. 创建并配置当前Spring Boot应用将要使用的Environment（包括配置要使用的PropertySource以及Profile）。
2. 遍历调用所有SpringApplicationRunListener的environmentPrepared()的方法，告诉他们：“当前SpringBoot应用使用的Environment准备好了咯！”。
3. 如果SpringApplication的showBanner属性被设置为true，则打印banner。
4. 根据用户是否明确设置了applicationContextClass类型以及初始化阶段的推断结果，决定该为当前SpringBoot应用创建什么类型的ApplicationContext并创建完成，然后根据条件决定是否添加ShutdownHook，决定是否使用自定义的BeanNameGenerator，决定是否使用自定义的ResourceLoader，当然，最重要的，将之前准备好的Environment设置给创建好的ApplicationContext使用。
5. ApplicationContext创建好之后，SpringApplication会再次借助Spring-FactoriesLoader，查找并加载classpath中所有可用的ApplicationContext-Initializer，然后遍历调用这些ApplicationContextInitializer的initialize（applicationContext）方法来对已经创建好的ApplicationContext进行进一步的处理。
6. 遍历调用所有SpringApplicationRunListener的contextPrepared()方法。
7. 最核心的一步，将之前通过@EnableAutoConfiguration获取的所有配置以及其他形式的IoC容器配置加载到已经准备完毕的ApplicationContext。
8. 遍历调用所有SpringApplicationRunListener的contextLoaded()方法。
9. 调用ApplicationContext的refresh()方法，完成IoC容器可用的最后一道工序。
10. 查找当前ApplicationContext中是否注册有CommandLineRunner，如果有则遍历执行它们。
11. 正常情况下，遍历执行SpringApplicationRunListener的finished()方法、（如果整个过程出现异常，则依然调用所有SpringApplicationRunListener的finished()方法，只不过这种情况下会将异常信息一并传入处理）
12. 去除事件通知点后，整个流程如下:

