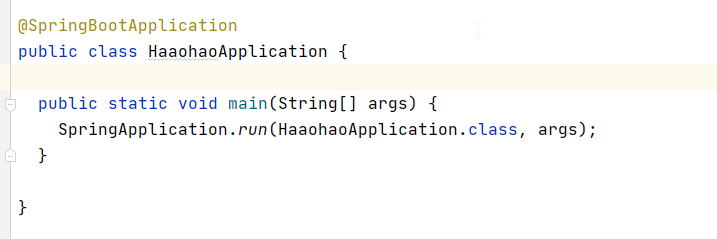
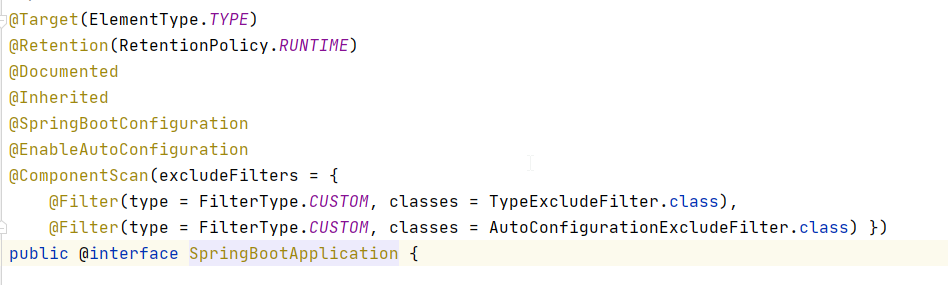
**SpringBoot一些注解含义：**



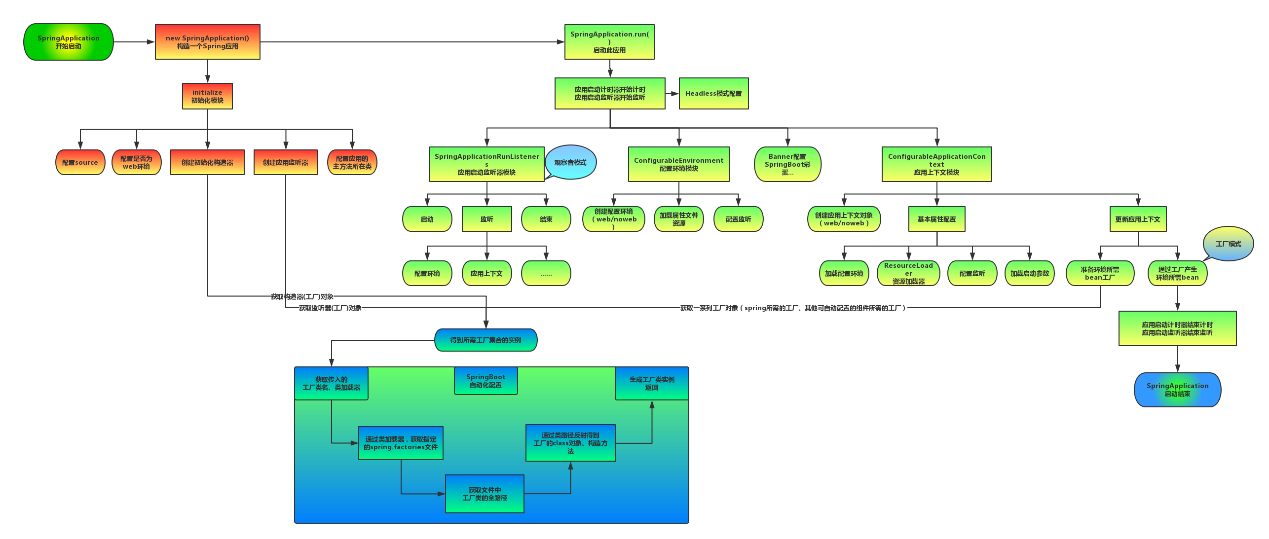


@Configuration（@SpringBootConfiguration），被注解的类将成为一个bean配置类。

@ComponentScan的作用就是自动扫描并加载符合条件的组件，比如@Component和@Repository等，最终将这些bean定义加载到spring容器中。

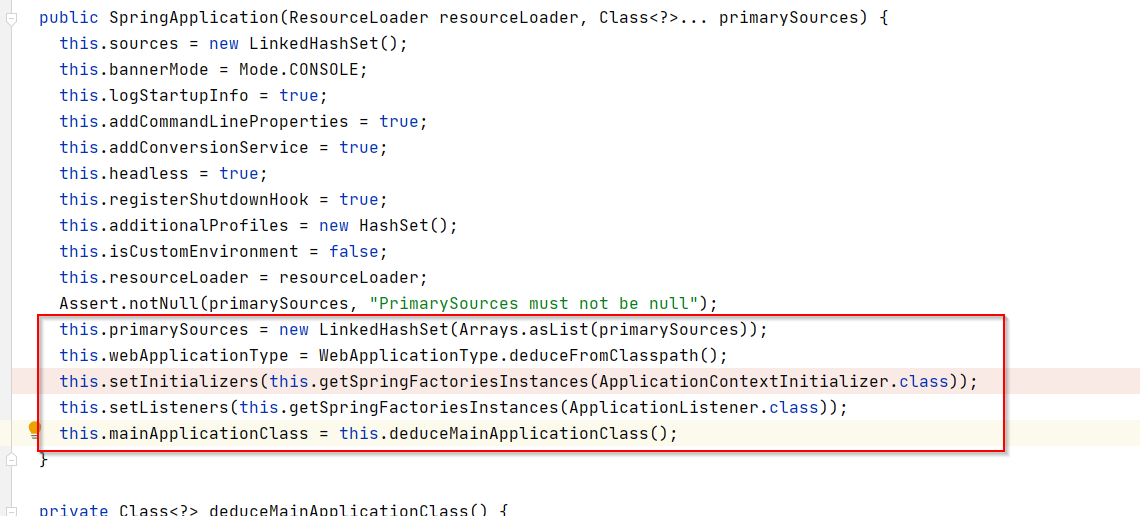
@EnableAutoConfiguration 这个注解的功能很重要，借助@Import的支持，收集和注册依赖包中相关的bean定义。

**启动流程图：**



 上图为[SpringBoot启动结构图](https://www.processon.com/view/link/59812124e4b0de2518b32b6e)，我们发现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，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、及核心的创建上下文环境模块，第三部分是自动化配置模块，该模块作为springboot自动配置核心。

**第一部分：**



**第二部分：**



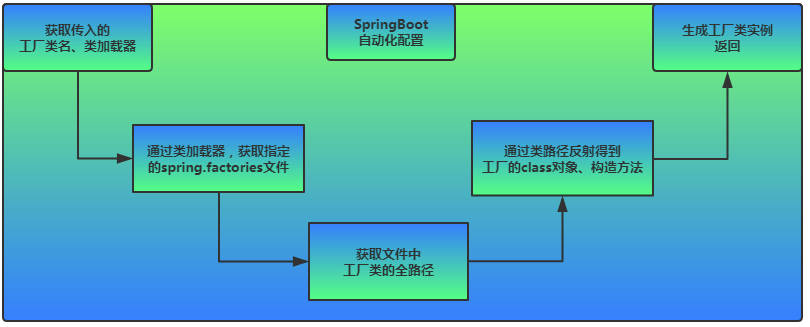
refreshContext在容器中开始实例化我们需要的Bean。

回顾整体流程，Springboot的启动，主要创建了配置环境(environment)、事件监听(listeners)、应用上下文(applicationContext)，并基于以上条件，在容器中开始实例化我们需要的Bean，至此，通过SpringBoot启动的程序已经构造完成，接下来我们来探讨自动化配置是如何实现。

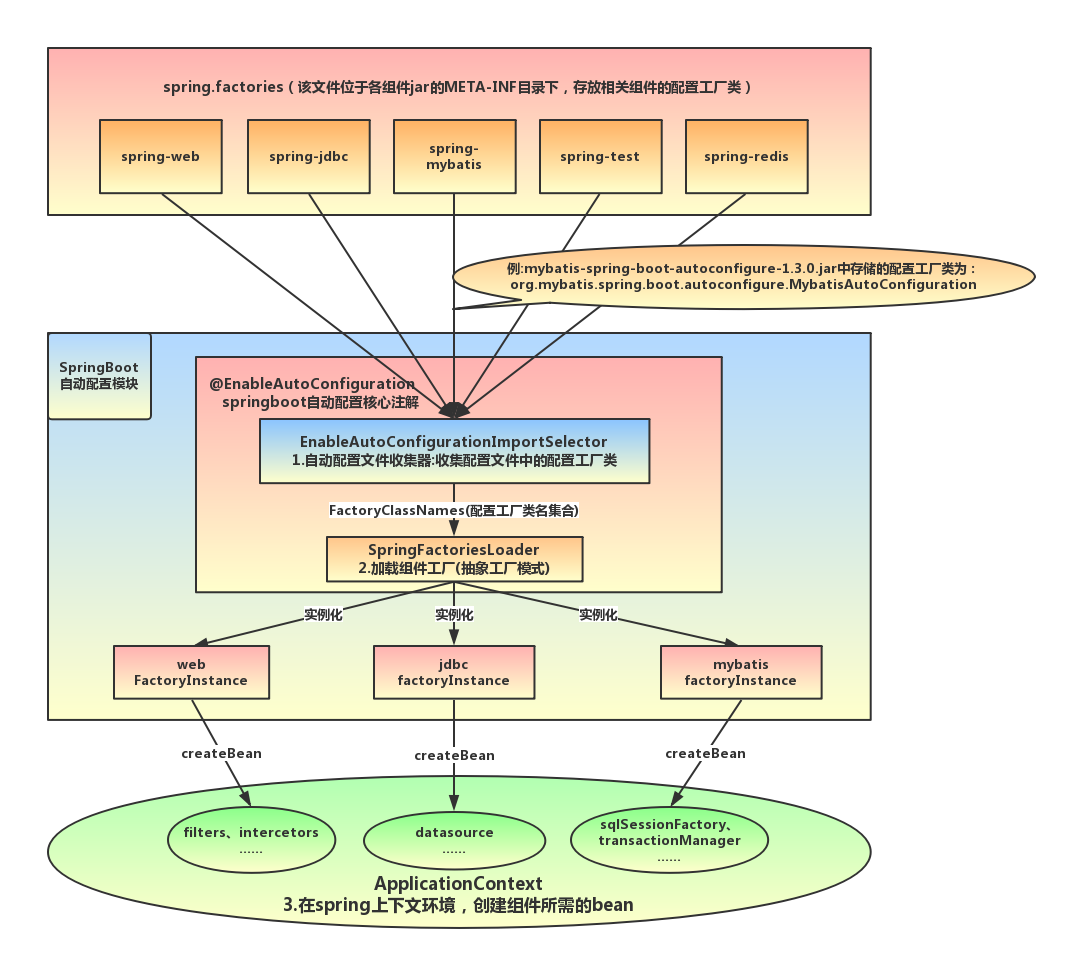
**第三部分：Spring 自动化配置**

该配置模块的主要使用到了SpringFactoriesLoader，即Spring工厂加载器.

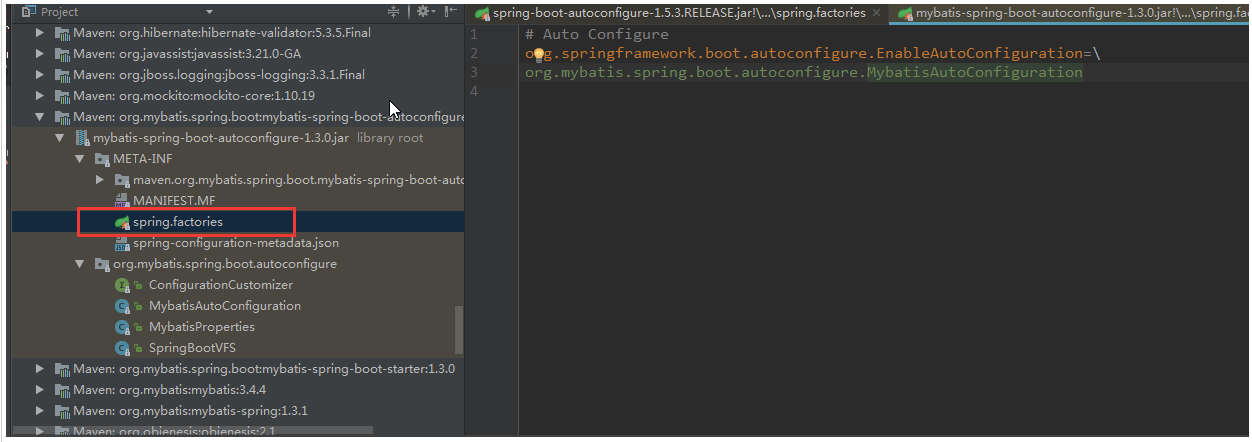
通过loadFactoryNames方法返回类名集合，方法调用方得到这些集合后，再通过反射获取这些类的类对象、构造方法，最终生成实例。

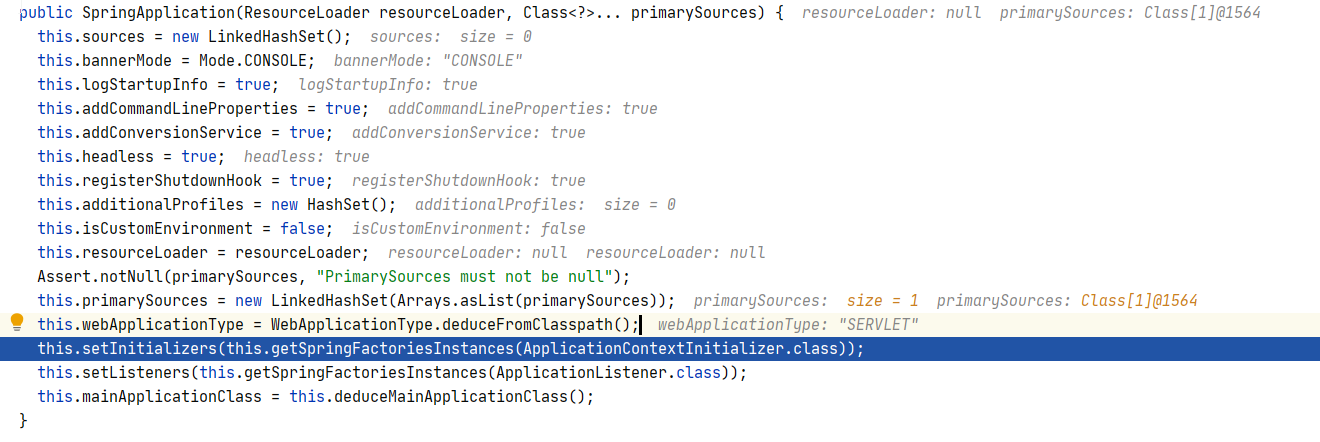


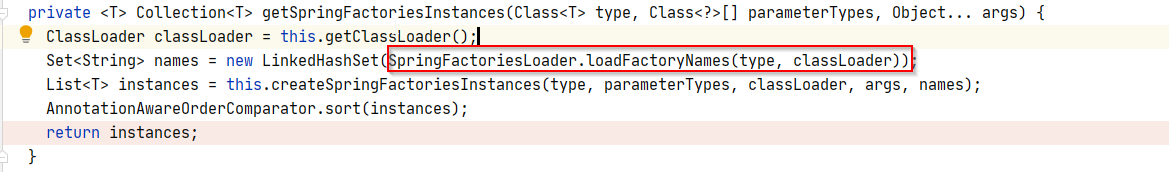
通过下图进一步理解自动配置的流程：

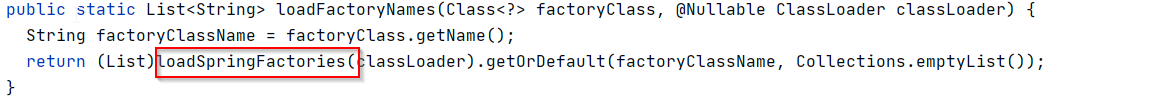


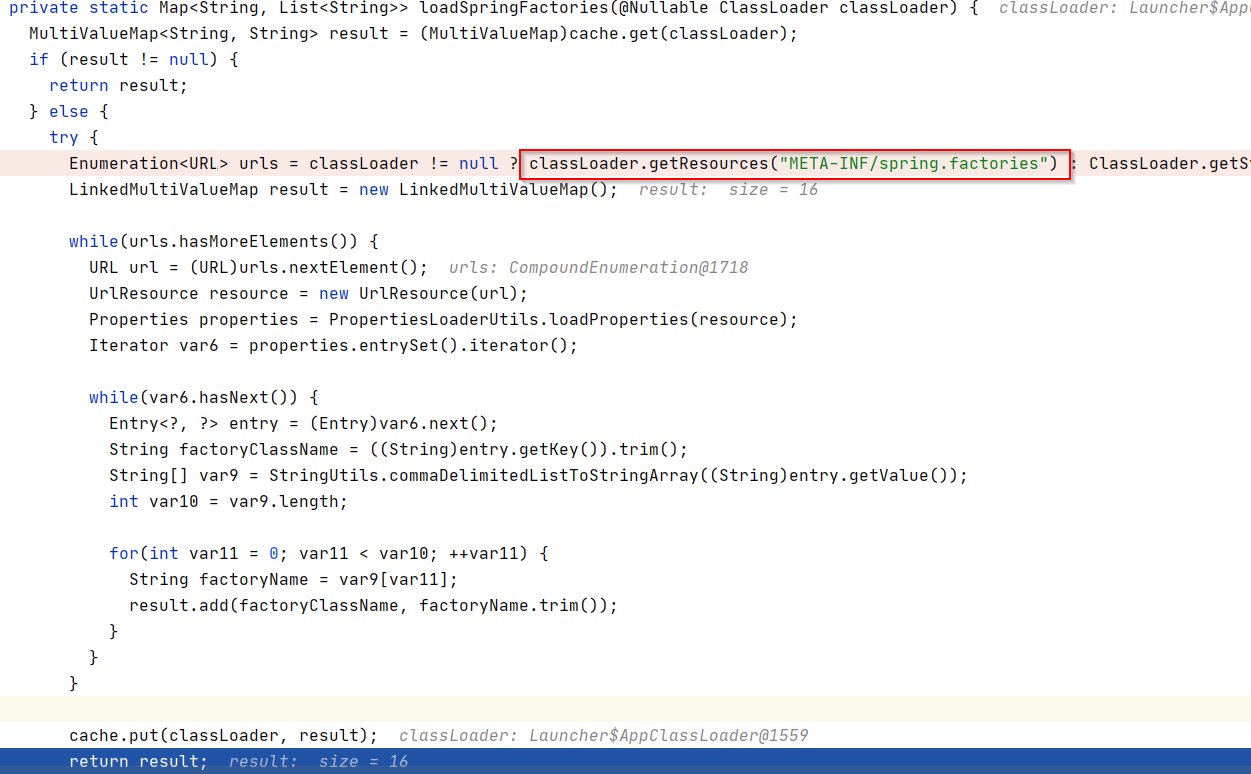
以myBatis 加载为例子：

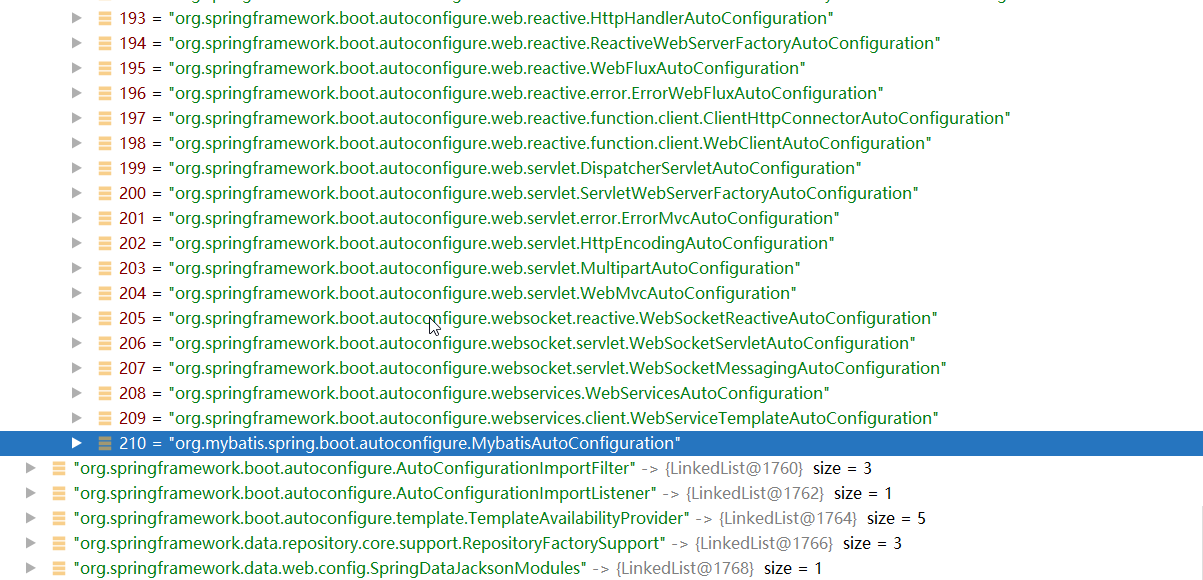


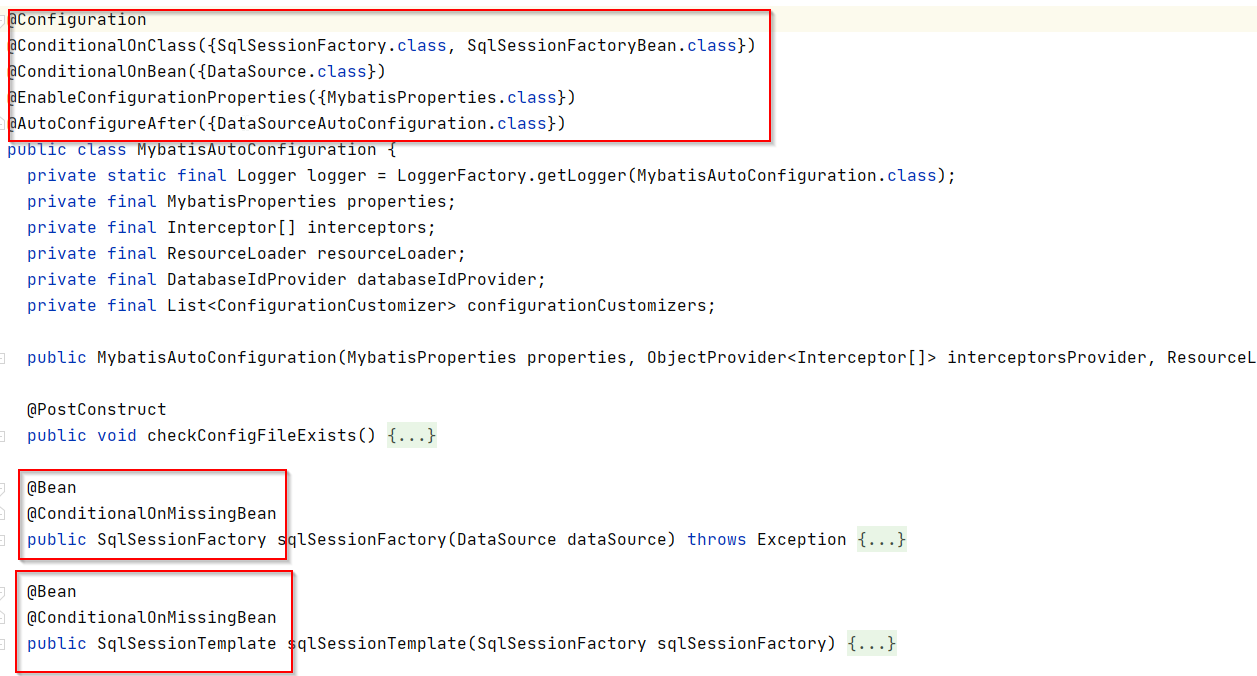






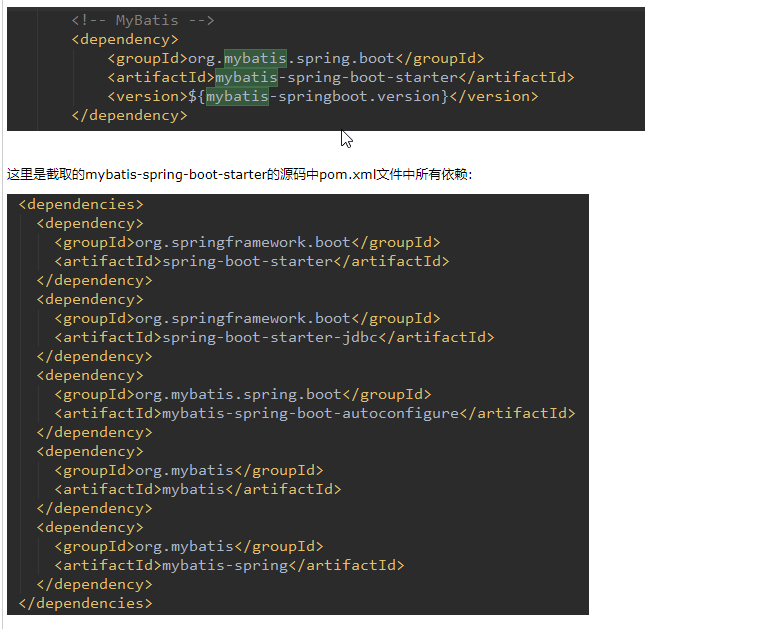






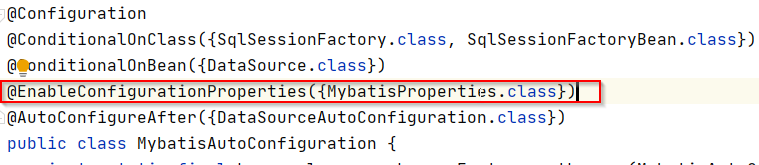
以上配置可以保证sqlSessionFactory、sqlSessionTemplate、dataSource等mybatis所需的组件均可被自动配置。通过分析我们可以发现，只要一个基于SpringBoot项目的类路径下存在SqlSessionFactory.class, SqlSessionFactoryBean.class，并且容器中已经注册了dataSourceBean，就可以触发自动化配置，意思说我们只要在maven的项目中加入了mybatis所需要的若干依赖，就可以触发自动配置，但引入mybatis原生依赖的话，每集成一个功能都要去修改其自动化配置类，那就得不到开箱即用的效果了。所以Spring-boot为我们提供了统一的starter可以直接配置好相关的类，触发自动配置所需的依赖(mybatis)如下：

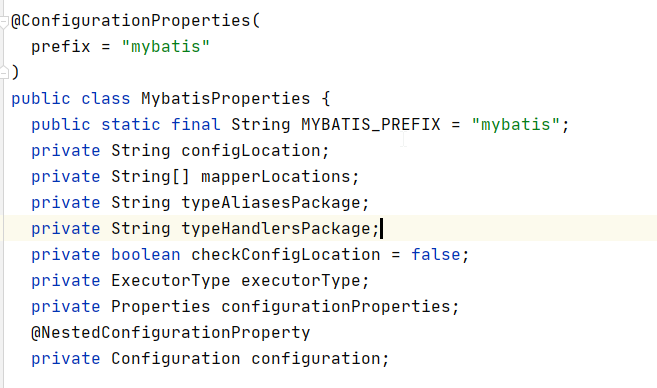
因为maven依赖的传递性，我们只要依赖starter就可以依赖到所有需要自动配置的类，实现开箱即用的功能。也体现出Springboot简化了Spring框架带来的大量XML配置以及复杂的依赖管理，让开发人员可以更加关注业务逻辑的开发。



接下来介绍下Bean参数配置：







@ConfigurationProperties注解的作用是把yml或者properties配置文件转化为bean。

@EnableConfigurationProperties注解的作用是使@ConfigurationProperties注解生效。如果只配置@ConfigurationProperties注解，在spring容器中是获取不到yml或者properties配置文件转化的bean。