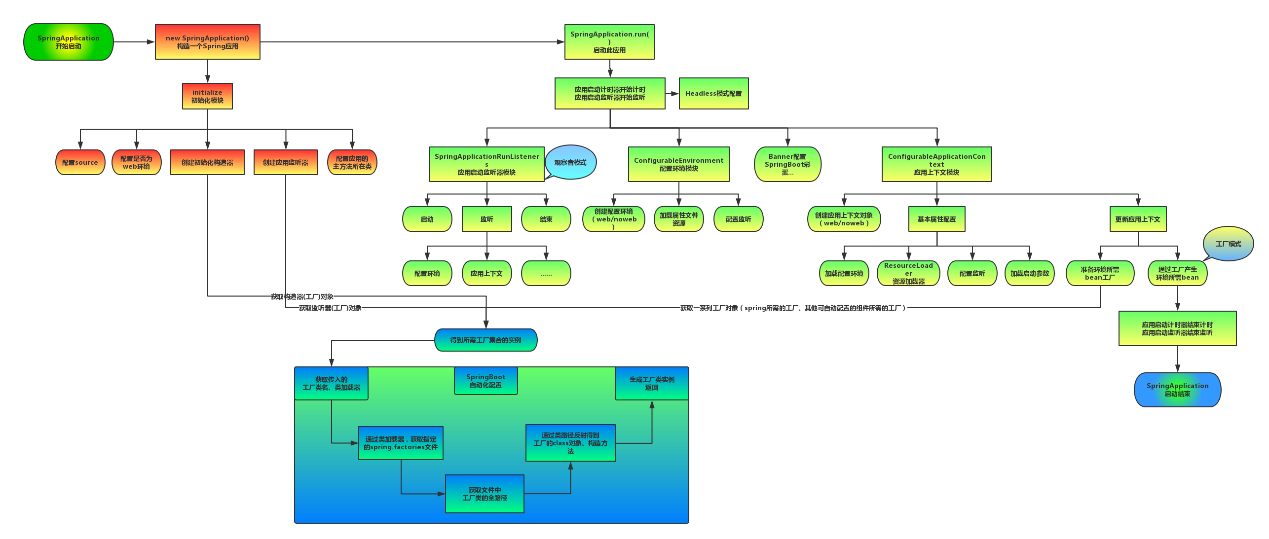
**[SpringBoot启动流程解析](https://www.cnblogs.com/trgl/p/7353782.html)**

参考流程中主要类类图，来分析其启动逻辑和自动化配置原理：



**总览：**上图为[SpringBoot启动结构图](https://www.processon.com/view/link/59812124e4b0de2518b32b6e)，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。

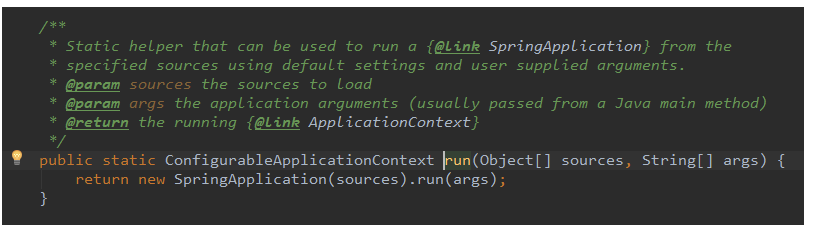
**启动：**

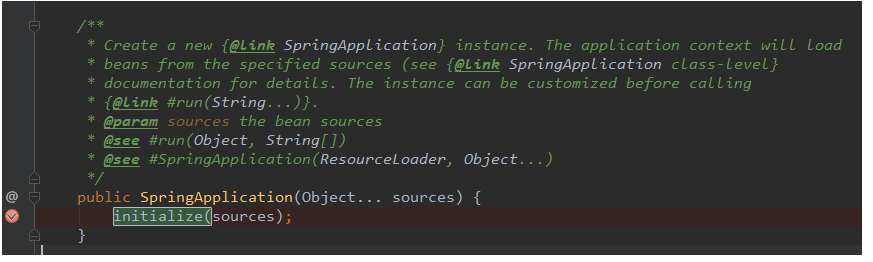
      每个SpringBoot程序都有一个主入口，也就是main方法，main里面调用SpringApplication.run()启动整个spring-boot程序，该方法所在类需要使用@SpringBootApplication注解，以及@ImportResource注解(if need)，@SpringBootApplication包括三个注解，功能如下：@EnableAutoConfiguration：SpringBoot根据应用所声明的依赖来对Spring框架进行自动配置

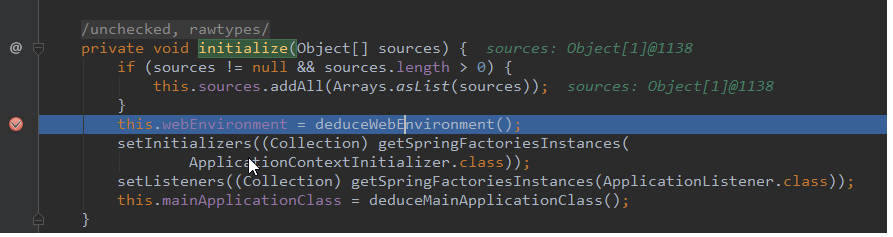
@SpringBootConfiguration(内部为@Configuration)：被标注的类等于在spring的XML配置文件中(applicationContext.xml)，装配所有bean事务，提供了一个spring的上下文环境

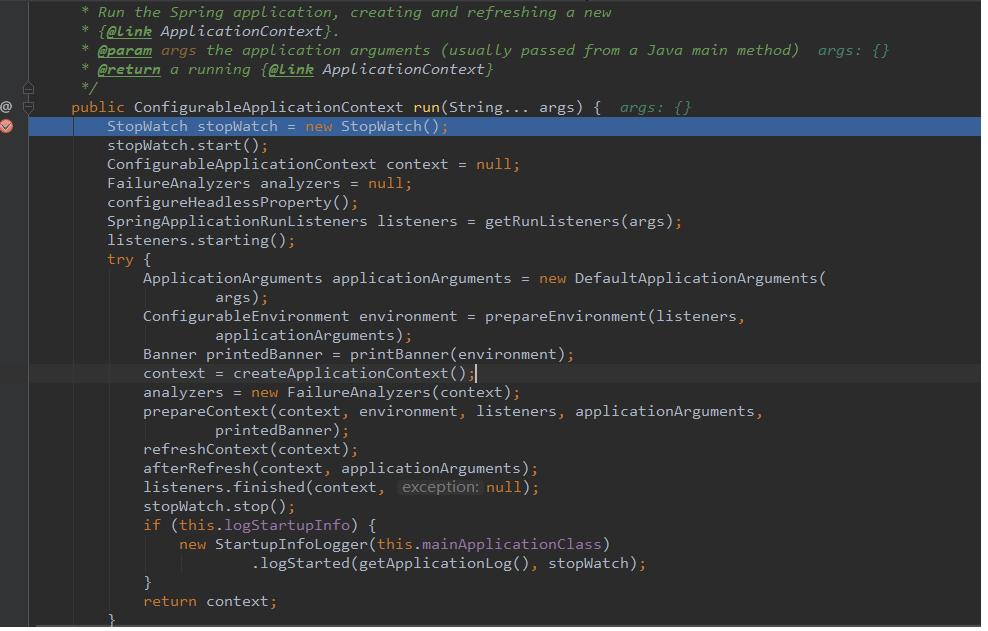
@ComponentScan：组件扫描，可自动发现和装配Bean，默认扫描SpringApplication的run方法里的启动类 所在的包路径下文件，所以最好将该 启动类 放到根包路径下。

进入Run 方法：









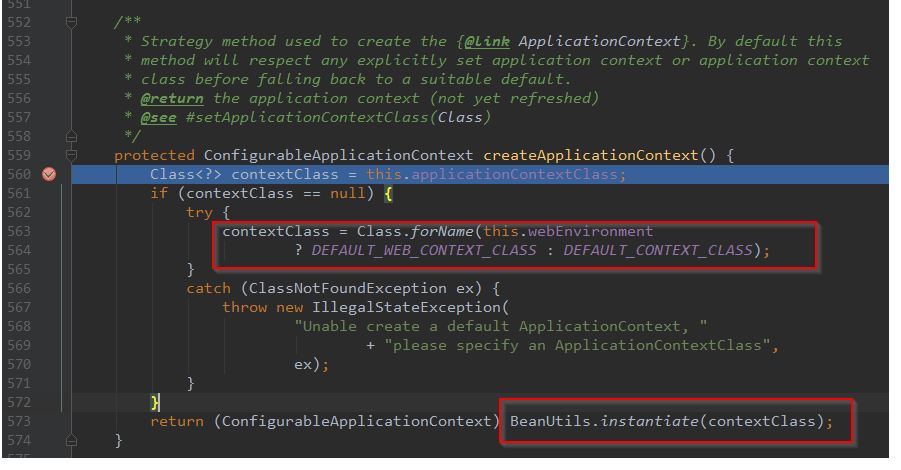
该方法中实现了如下几个关键步骤：

1.创建了应用的监听器SpringApplicationRunListeners并开始监听

2.加载SpringBoot配置环境(ConfigurableEnvironment)，如果是通过web容器发布，会加载StandardEnvironment，其最终也是继承了ConfigurableEnvironment

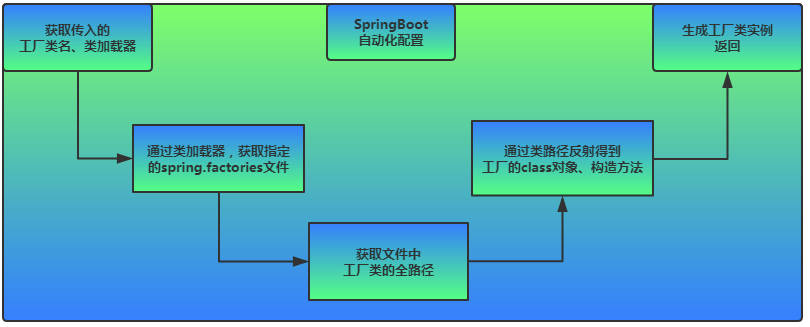
3.配置环境(Environment)加入到监听器对象中(SpringApplicationRunListeners)

4.创建run方法的返回对象：ConfigurableApplicationContext(应用配置上下文) ，我们可以看一下创建方法：



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。

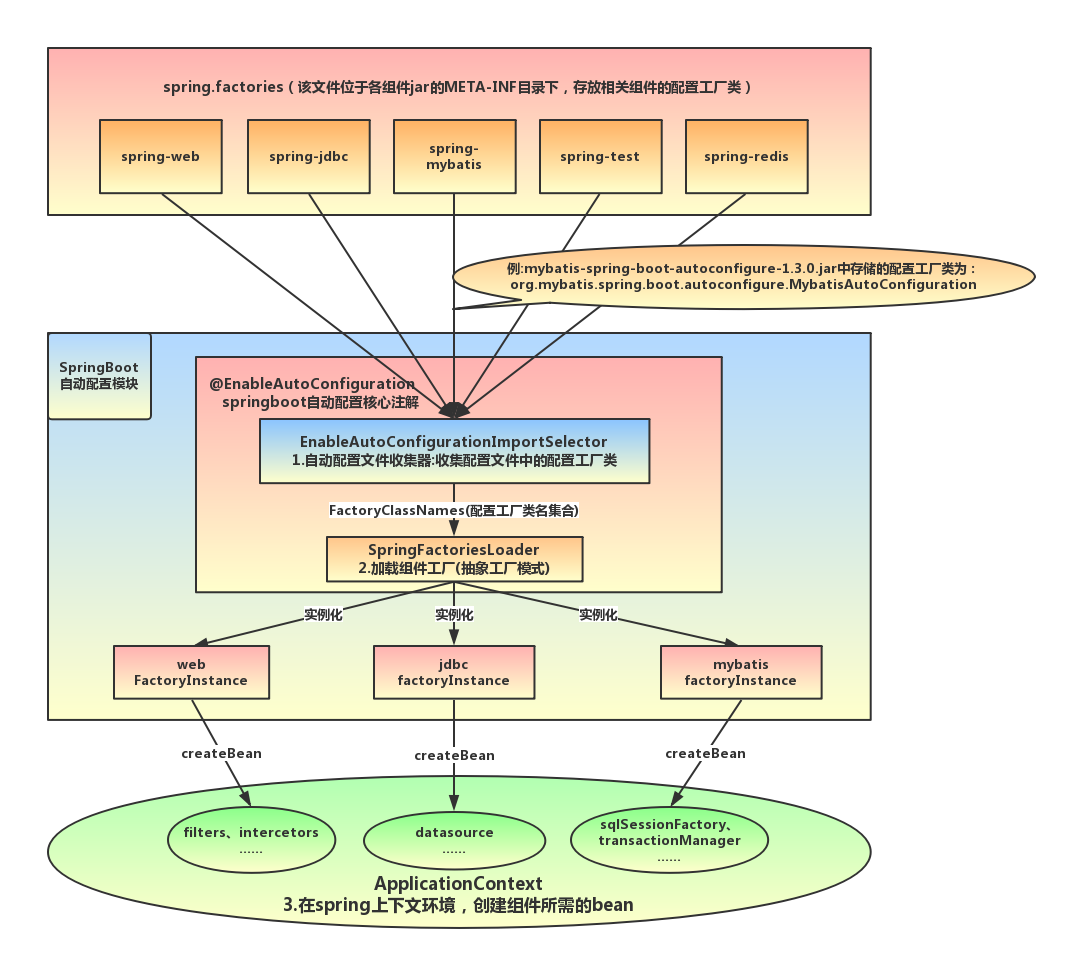
**自动化配置：**



SpringBoot自动配置模块

       该配置模块的主要使用到了SpringFactoriesLoader，即Spring工厂加载器，该对象提供了loadFactoryNames方法，入参为factoryClass和classLoader，即需要传入上图中的工厂类名称和对应的类加载器，方法会根据指定的classLoader，加载该类加器搜索路径下的指定文件，即spring.factories文件，传入的工厂类为接口，而文件中对应的类则是接口的实现类，或最终作为实现类，所以文件中一般为如下图这种一对多的类名集合，获取到这些实现类的类名后，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

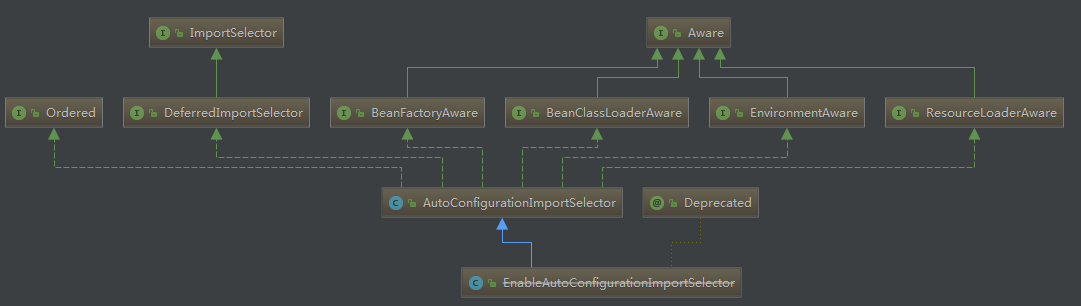


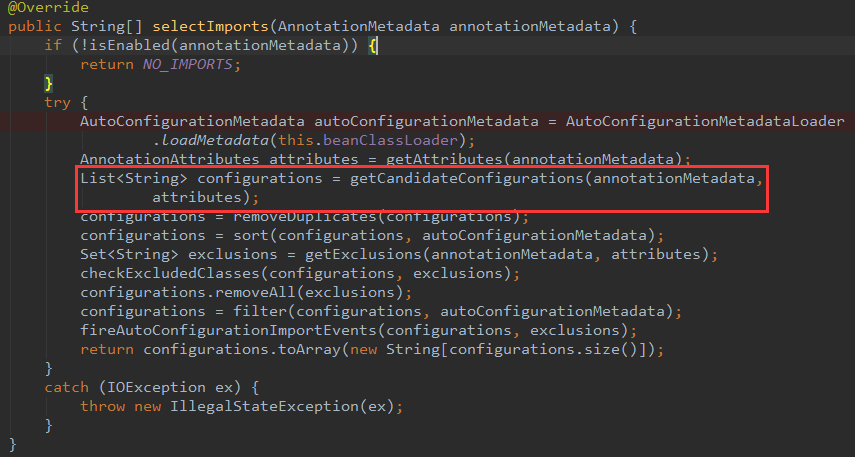


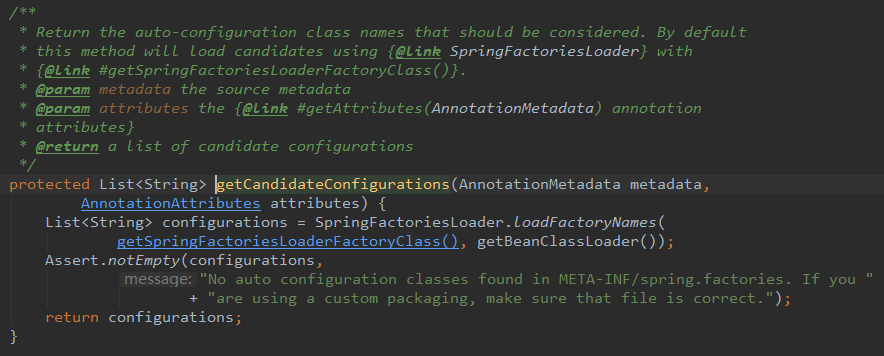
SpringBoot自动化配置关键组件关系图

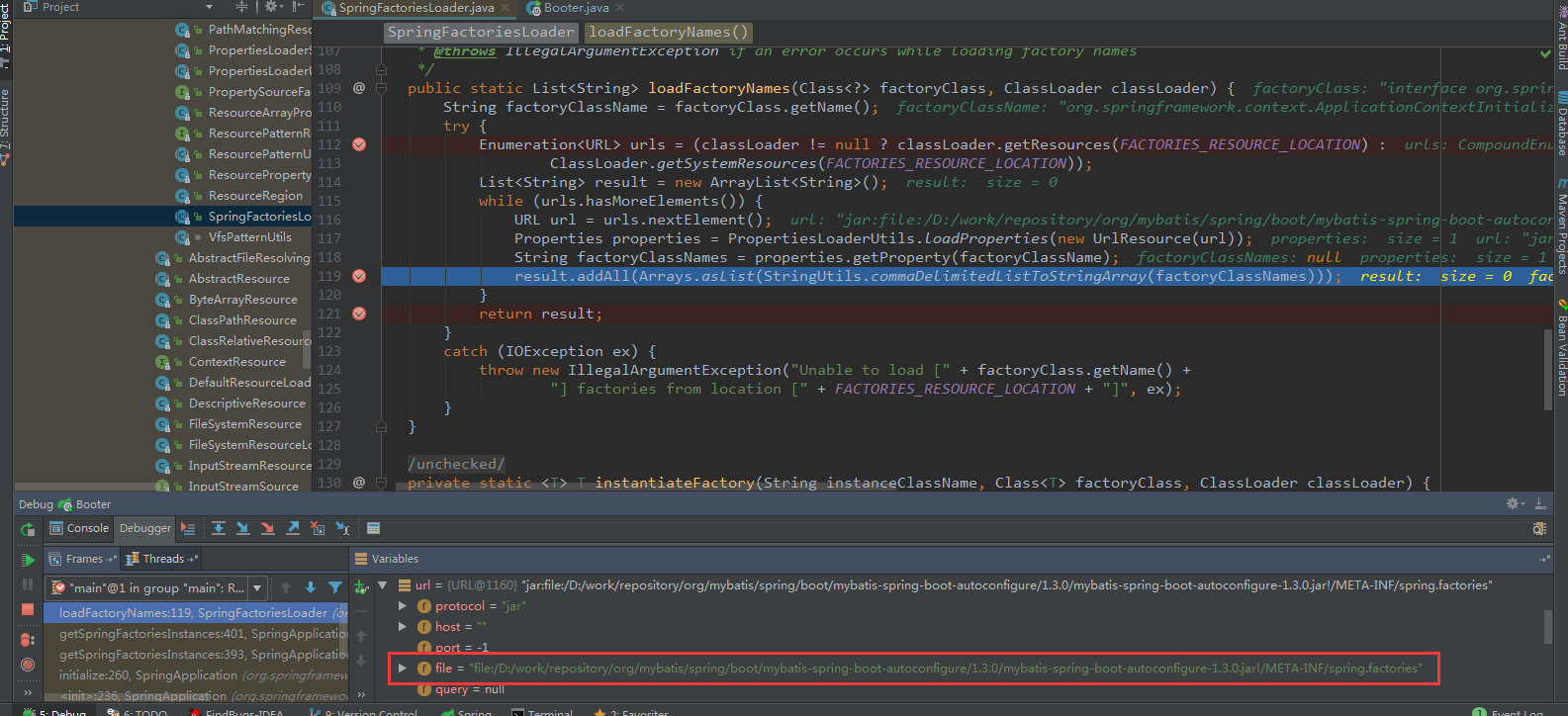
      mybatis-spring-boot-starter、spring-boot-starter-web等组件的META-INF文件下均含有spring.factories文件，自动配置模块中，SpringFactoriesLoader收集到文件中的类全名并返回一个类全名的数组，返回的类全名通过反射被实例化，就形成了具体的工厂实例，工厂实例来生成组件具体需要的bean。

之前我们提到了EnableAutoConfiguration注解，其类图如下

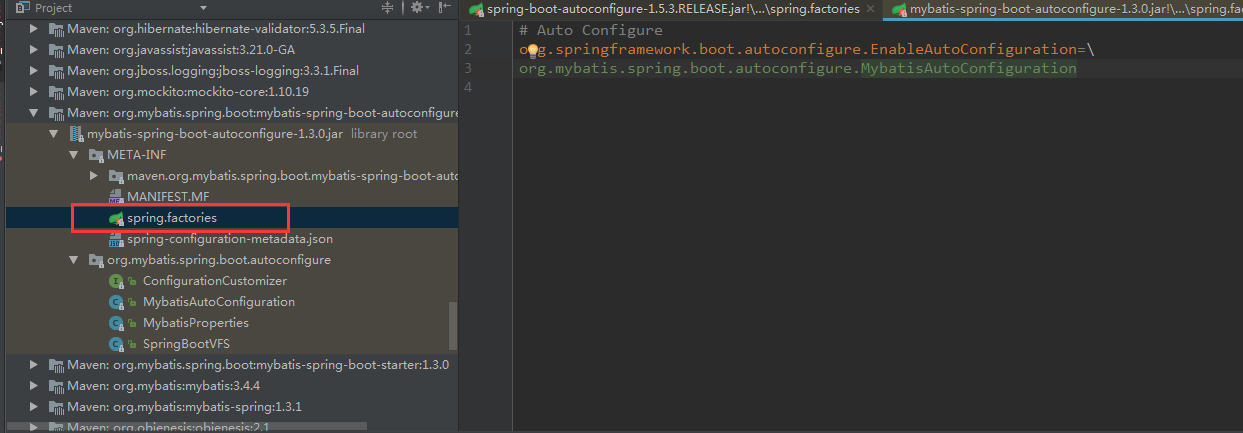


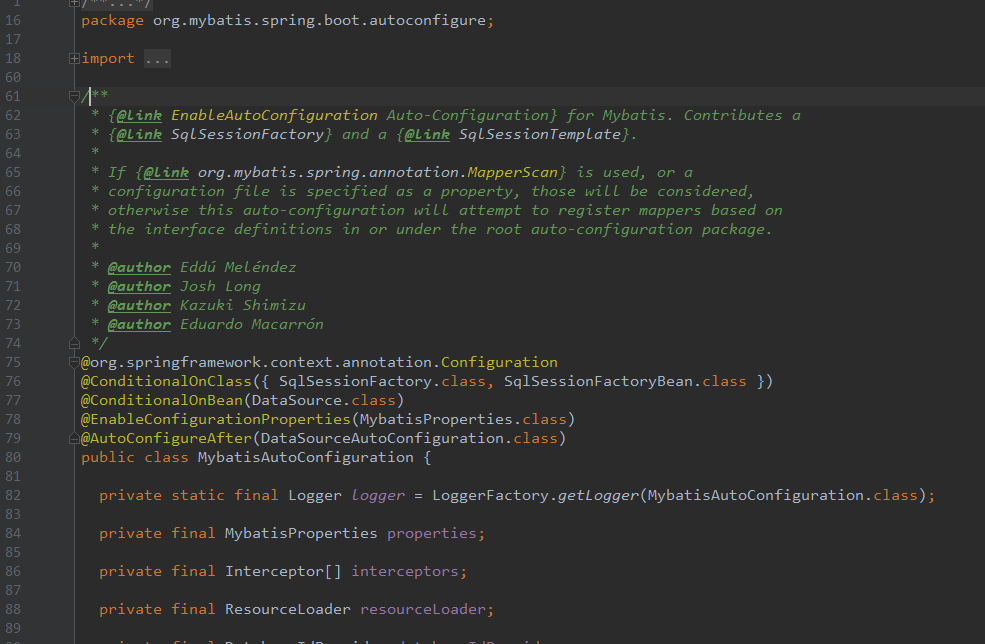






在上面的代码可以看到自动配置器会跟根据传入的factoryClass.getName()到项目系统路径下所有的spring.factories文件中找到相应的key，从而加载里面的类。我们就选取这个mybatis-spring-boot-autoconfigure下的spring.factories文件





发现@Spring的Configuration，俨然是一个通过注解标注的springBean，继续向下看，

@ConditionalOnClass({ SqlSessionFactory.class, SqlSessionFactoryBean.class})这个注解的意思是：当存在SqlSessionFactory.class, SqlSessionFactoryBean.class这两个类时才解析MybatisAutoConfiguration配置类,否则不解析这一个配置类，make sence，我们需要mybatis为我们返回会话对象，就必须有会话工厂相关类

@CondtionalOnBean(DataSource.class):只有处理已经被声明为bean的dataSource

@ConditionalOnMissingBean(MapperFactoryBean.class)这个注解的意思是如果容器中不存在name指定的bean则创建bean注入，否则不执行（该类源码较长，篇幅限制不全粘贴）

       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，@Configuration注解已经提供了Spring的上下文环境，所以以上组件的配置方式与Spring启动时通过mybatis.xml文件进行配置起到一个效果。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