pom.xml

- spring-boot-dependencies ：核心依赖在父工程中
- 所以引入依赖dependency的时候不需要指定版本：

启动器

官方启动器

```
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

- 启动器说白了就是Springboot的启动场景；
- 比如spring-boot-stater-web，就配置了web场景所需的全部依赖
- SpringBoot将所有的功能场景都抽取出来，做成一个个的starter（启动器），只需要在项目中引入这些starter即可，所有相关的依赖都会导入进来 ， 我们要用什么功能就导入什么样的场景启动器即可 ；也可以自己自定义 starter；

主程序

```
// 标注该类是一个应用，直接运行main方法就启动了
@SpringBootApplication
public class MainApplication {
    public static void main(String[] args) {
        //  启动应用
        SpringApplication.run(MainApplication.class, args);
    }
}
```

@SpringBootApplication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})}
```

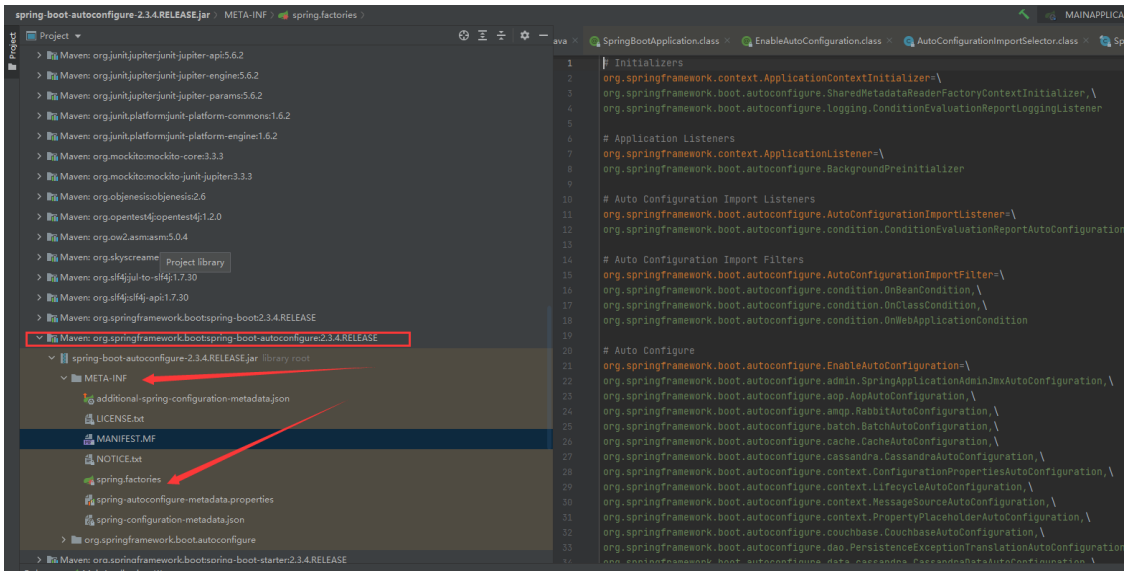
其中包括：

- EnableAutoConfiguration:表示自动配置，也是一个组合注解：

```
@AutoConfigurationPackage
@Import({AutoConfigurationImportSelector.class})
    AutoConfigurationImportSelector ： 自动导入选择器
```

Springboot所有的自动配置，都在启动类中扫描并加载。

配置都在spring.factories里面



在该文件里有所有的配置，但是只有一部分被加载进来，利用注解@ConditionalOnXXX进行判断

- @SpringBootConfiguration:代表是一个配置类，就相当如xml文件
- @ComponentScan： 扫描当前主启动类同级的包

总结

1. 启动应用的时候，从spring.factories里面获取指定的值
2. 根据starter，需要配置的类就会生效进入容器，
3. 自动配置包：spring-boot-autoconfigure-x.x.x.RELEASE.jar