<https://mp.weixin.qq.com/s/P1OPUINsOjy_vrDNNgDaDg>

https://mp.weixin.qq.com/s/tKr\_shLQnvcQADr4mvcU3A

# 自定义 Spring Boot 中的 starter

我们使用 Spring Boot，基本上都是沉醉在它 Stater 的方便之中。Starter 为我们带来了众多的自动化配置，有了这些自动化配置，我们可以不费吹灰之力就能搭建一个生产级开发环境，有的小伙伴会觉得这个 Starter 好神奇呀！其实 Starter 也都是 Spring + SpringMVC 中的基础知识点实现的，今天松哥就来带大家自己来撸一个 Starter ，慢慢揭开 Starter 的神秘面纱！

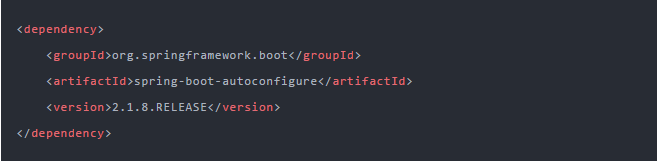
# 1.核心知识

其实 Starter 的核心就是条件注解 @Conditional ，当 classpath 下存在某一个 Class 时，某个配置才会生效，前面松哥已经带大家学习过不少 Spring Boot 中的知识点，有的也涉及到源码解读，大伙可能也发现了源码解读时总是会出现条件注解，其实这就是 Starter 配置的核心之一，大伙有兴趣可以翻翻历史记录，看看松哥之前写的关于 Spring Boot 的文章，这里我就不再重复介绍了。

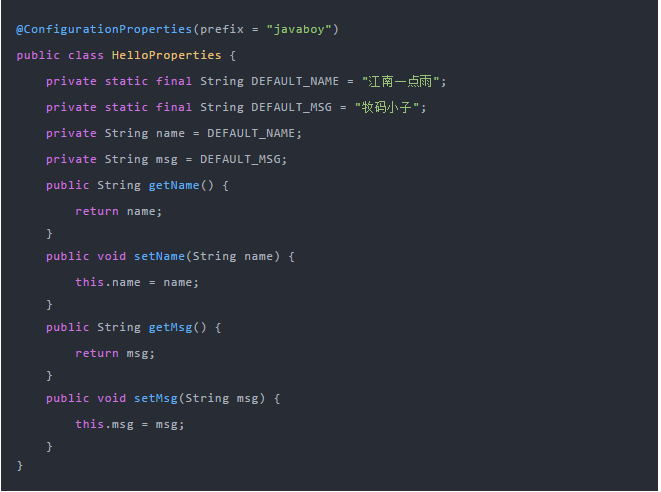
# 2.定义自己的 Starter

## 2.1定义

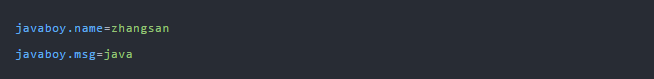
所谓的 Starter ，其实就是一个普通的 Maven 项目，因此我们自定义 Starter ，需要首先创建一个普通的 Maven 项目，创建完成后，添加 Starter 的自动化配置类即可，如下：



配置完成后，我们首先创建一个 HelloProperties 类，用来接受 application.properties 中注入的值，如下：

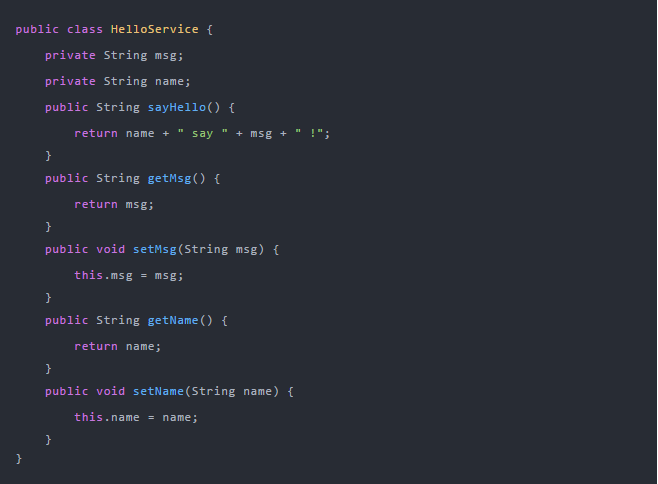


这个配置类很好理解，将 application.properties 中配置的属性值直接注入到这个实例中， @ConfigurationProperties 类型安全的属性注入，即将 application.properties 文件中前缀为 javaboy 的属性注入到这个类对应的属性上， 最后使用时候，application.properties 中的配置文件，大概如下：



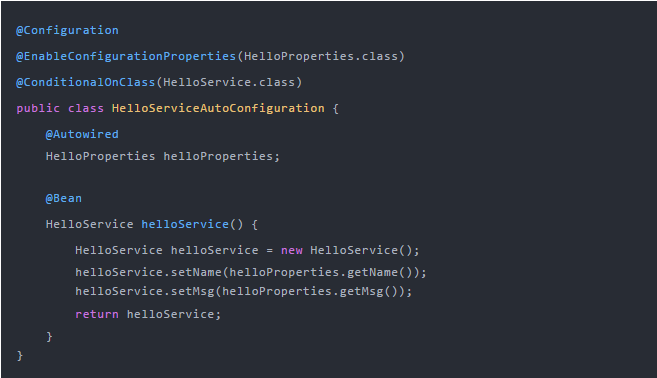
关注类型安全的属性注入，读者可以参考松哥之前的这篇文章：[**Spring Boot中的yaml配置简介**](https://mp.weixin.qq.com/s?__biz=MzI1NDY0MTkzNQ==&mid=2247485055&idx=1&sn=54ebd48d58eee3b36c107a73bdc9f13f&scene=21#wechat_redirect)，这篇文章虽然是讲 yaml 配置，但是关于类型安全的属性注入和 properties 是一样的。

配置完成 HelloProperties 后，接下来我们来定义一个 HelloService ，然后定义一个简单的 say 方法， HelloService 的定义如下：



这个很简单，没啥好说的。

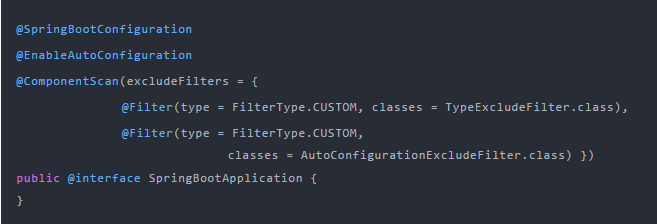
接下来就是我们的重轴戏，自动配置类的定义，用了很多别人定义的自定义类之后，我们也来自己定义一个自定义类。先来看代码吧，一会松哥再慢慢解释：



关于这一段自动配置，解释如下：

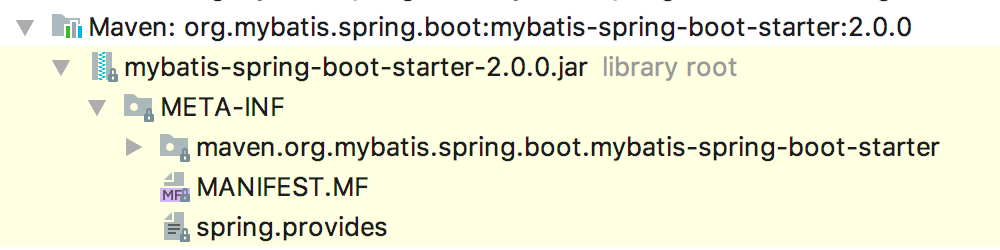
* 首先 @Configuration 注解表明这是一个配置类。
* @EnableConfigurationProperties 注解是使我们之前配置的 @ConfigurationProperties 生效，让配置的属性成功的进入 Bean 中。
* @ConditionalOnClass 表示当项目当前 classpath 下存在 HelloService 时，后面的配置才生效。
* 自动配置类中首先注入 HelloProperties ，这个实例中含有我们在 application.properties 中配置的相关数据。
* 提供一个 HelloService 的实例，将 HelloProperties 中的值注入进去。

做完这一步之后，我们的自动化配置类就算是完成了，接下来还需要一个 spring.factories 文件，那么这个文件是干嘛的呢？大家知道我们的 Spring Boot 项目的启动类都有一个 @SpringBootApplication 注解，这个注解的定义如下：



大家看到这是一个组合注解，其中的一个组合项就是 @EnableAutoConfiguration ，这个注解是干嘛的呢？

@EnableAutoConfiguration 表示启用 Spring 应用程序上下文的自动配置，该注解会自动导入一个名为 **AutoConfigurationImportSelector** 的类,而这个类会去读取一个名为 spring.factories 的文件, spring.factories 中则定义需要加载的自动化配置类，我们打开任意一个框架的 Starter ，都能看到它有一个 spring.factories 文件，例如 MyBatis 的 Starter 如下：



那么我们自定义 Starter 当然也需要这样一个文件，我们首先在 Maven 项目的 resources 目录下创建一个名为 META-INF 的文件夹，然后在文件夹中创建一个名为 spring.factories 的文件，文件内容如下：

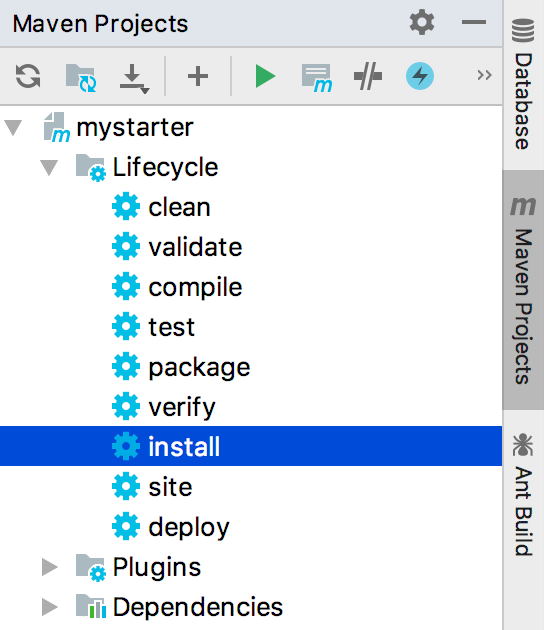
org.springframework.boot.autoconfigure.EnableAutoConfiguration=org.javaboy.mystarter.HelloServiceAutoConfiguration

在这里指定我们的自动化配置类的路径即可。

如此之后我们的自动化配置类就算完成了。

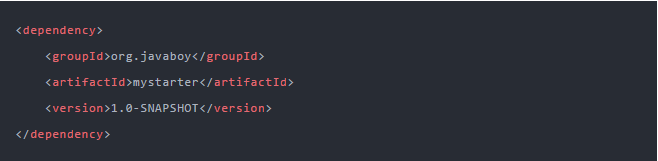
## 2.2本地安装

如果在公司里，大伙可能需要将刚刚写好的自动化配置类打包，然后上传到 Maven 私服上，供其他同事下载使用，我这里就简单一些，我就不上传私服了，我将这个自动化配置类安装到本地仓库，然后在其他项目中使用即可。安装方式很简单，在 IntelliJ IDEA 中，点击右边的 Maven Project ，然后选择 Lifecycle 中的 install ，双击即可，如下：

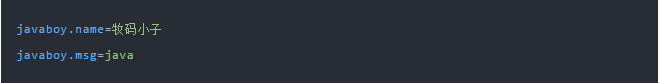
双击完成后，这个 Starter 就安装到我们本地仓库了，当然小伙伴也可以使用 Maven 命令去安装。

# 3.使用 Starter

接下来，我们来新建一个普通的 Spring Boot 工程，这个 Spring Boot 创建成功之后，加入我们自定义 Starter 的依赖，如下：



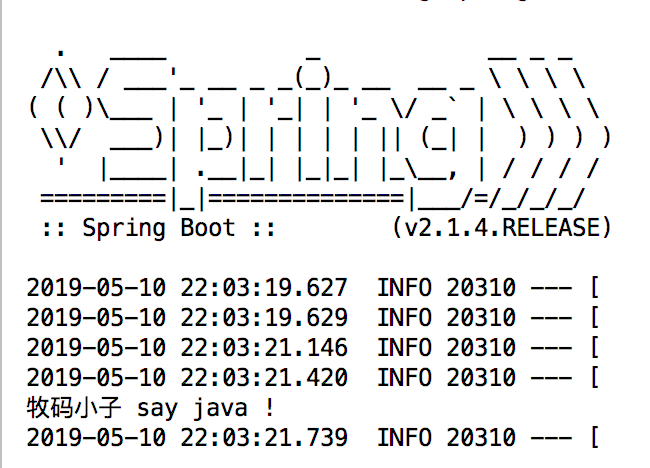
此时我们引入了上面自定义的 Starter ，也即我们项目中现在有一个默认的 HelloService 实例可以使用，而且关于这个实例的数据，我们还可以在 application.properties 中进行配置，如下：



配置完成后，方便起见，我这里直接在单元测试方法中注入 HelloSerivce 实例来使用，代码如下：



执行单元测试方法，打印日志如下：



好了，一个简单的自动化配置类我们就算完成了，是不是很简单！

# 4.总结

本文主要带领小伙伴自己徒手撸一个 Starter ，使用这种方式帮助大家揭开 Starter 的神秘面纱！大伙有问题可以留言讨论。