<https://mp.weixin.qq.com/s/6_ybZtLM4ZqJJx7Cb3yniw>

# Spring Boot2 系列教程(七)理解自动化配置的原理

Spring Boot 中的自动化配置确实够吸引人，甚至有人说 Spring Boot 让 Java 又一次焕发了生机，这话虽然听着有点夸张，但是不可否认的是，曾经臃肿繁琐的 Spring 配置确实让人感到头大，而 Spring Boot 带来的全新自动化配置，又确实缓解了这个问题。

你要是问这个自动化配置是怎么实现的，很多人会说不就是 starter 嘛！那么 starter 的原理又是什么呢？松哥以前写过一篇文章，介绍了自定义 starter：

* [**徒手撸一个 Spring Boot 中的 Starter ，解密自动化配置黑魔法！**](https://mp.weixin.qq.com/s?__biz=MzI1NDY0MTkzNQ==&mid=2247485546&idx=1&sn=b06b411b6bab393db625ac0c337caa4b&scene=21#wechat_redirect)

这里边有一个非常关键的点，那就是**条件注解**，甚至可以说条件注解是整个 Spring Boot 的基石。

条件注解并非一个新事物，这是一个存在于 Spring 中的东西，我们在 Spring 中常用的 profile 实际上就是条件注解的一个特殊化。

想要把 Spring Boot 的原理搞清，条件注解必须要会用，因此今天松哥就来和大家聊一聊条件注解。

# 定义

Spring4 中提供了更加通用的条件注解，让我们可以在满足不同条件时创建不同的 Bean，这种配置方式在 Spring Boot 中得到了广泛的使用，大量的自动化配置都是通过条件注解来实现的，查看松哥之前的 Spring Boot 文章，凡是涉及到源码解读的文章，基本上都离不开条件注解：@Conditional

# 实践

抛开 Spring Boot，我们来单纯的看看在 Spring 中条件注解的用法。

首先我们来创建一个普通的 Maven 项目，然后引入 spring-context，如下：



然后定义一个 Food 接口：

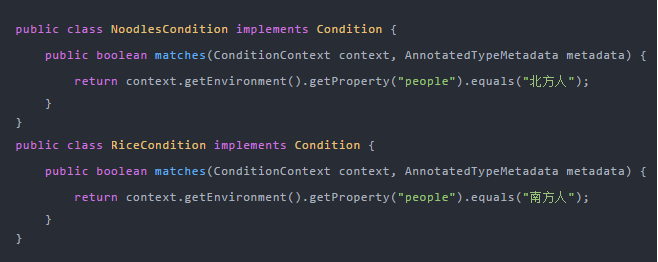


Food 接口有一个 showName 方法和两个实现类：



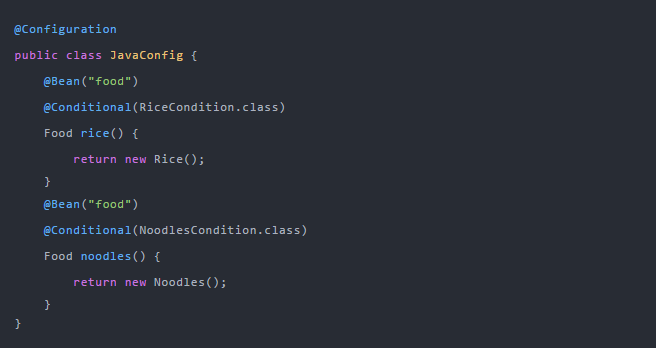
分别是 Rice 和 Noodles 两个类，两个类实现了 showName 方法，然后分别返回不同值。

接下来再分别创建 Rice 和 Noodles 的条件类，如下：



在 matches 方法中做条件属性判断，当系统属性中的 people 属性值为 '北方人' 的时候，NoodlesCondition 的条件得到满足，当系统中 people 属性值为 '南方人' 的时候，RiceCondition 的条件得到满足，换句话说，哪个条件得到满足，一会就会创建哪个 Bean 。

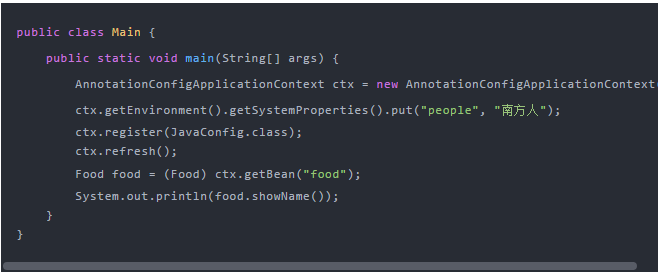
接下来我们来配置 Rice 和 Noodles ：



这个配置类，大家重点注意两个地方：

* 两个 Bean 的名字都为 food，这不是巧合，而是有意取的。两个 Bean 的返回值都为其父类对象 Food。
* 每个 Bean 上都多了 @Conditional 注解，当 @Conditional 注解中配置的条件类的 matches 方法返回值为 true 时，对应的 Bean 就会生效。

配置完成后，我们就可以在 main 方法中进行测试了：



首先我们创建一个 AnnotationConfigApplicationContext 实例用来加载 Java 配置类，然后我们添加一个 property 到 environment 中，添加完成后，再去注册我们的配置类，然后刷新容器。容器刷新完成后，我们就可以从容器中去获取 food 的实例了，这个实例会根据 people 属性的不同，而创建出来不同的 Food 实例。

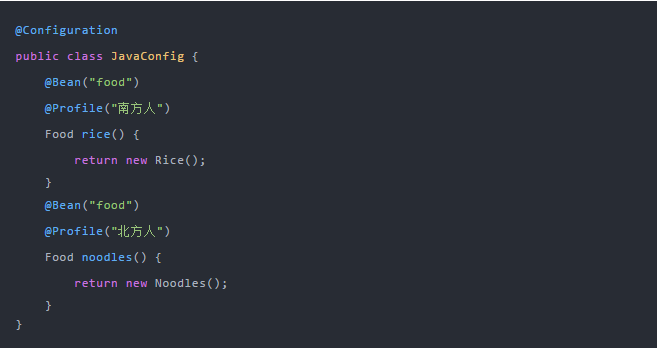
这个就是 Spring 中的条件注解。

# 进化

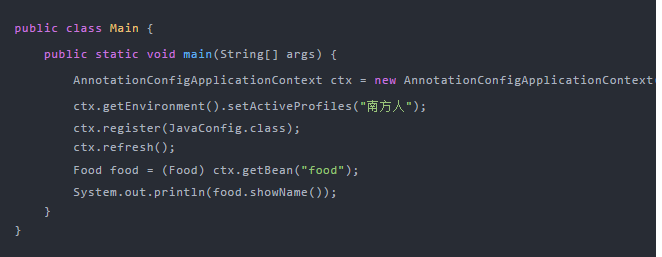
条件注解还有一个进化版，那就是 Profile。我们一般利用 Profile 来实现在开发环境和生产环境之间进行快速切换。其实 Profile 就是利用条件注解来实现的。

还是刚才的例子，我们用 Profile 来稍微改造一下：

首先 Food、Rice 以及 Noodles 的定义不用变，条件注解这次我们不需要了，我们直接在 Bean 定义时添加 @Profile 注解，如下：



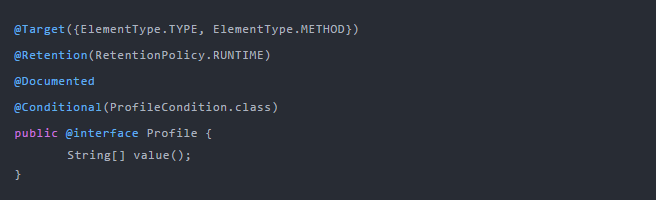
这次不需要条件注解了，取而代之的是 @Profile 。然后在 Main 方法中，按照如下方式加载 Bean：



效果和上面的案例一样。

这样看起来 @Profile 注解貌似比 @Conditional 注解还要方便，那么 @Profile 注解到底是什么实现的呢？

我们来看一下 @Profile 的定义：



可以看到，它也是通过条件注解来实现的。条件类是 ProfileCondition ，我们来看看：



看到这里就明白了，其实还是我们在条件注解中写的那一套东西，只不过 @Profile 注解自动帮我们实现了而已。

@Profile 虽然方便，但是不够灵活，因为具体的判断逻辑不是我们自己实现的。而 @Conditional 则比较灵活。

# 结语

两个例子向大家展示了条件注解在 Spring 中的使用，**它的一个核心思想就是当满足某种条件的时候，某个 Bean 才会生效，而正是这一特性，支撑起了 Spring Boot 的自动化配置。**