21_精选面试题：@Value 是如何将外部化配置注入到 spring bean 中？

1、开篇

前面两节课介绍了 Spring IOC 的依赖查找和依赖注入的两种方式，实际上针对这两种方式都对应了依赖来源。今天给大家讲解@Value 如何将外部化配置注入到 spring bean 中，包括如下内容：

- @Value 获取配置的简介
- 基于配置文件注入
- 基于非配置文件注入
- 默认值注入

2、@Value 获取配置的简介

@Value 是一种获取外部配置的方式，是 SpringBoot 中的一种用法，在 SpringBoot 允许将配置进行外部化（externalize），这样就能够在不同的环境下使用相同的代码。可以使用 properties 文件，yaml 文件，环境变量和命令行参数来外部化配置。使用@Value 注解，可以直接将属性值注入到 beans 中，然后通过 Spring 的 Environment 抽象或通过@ConfigurationProperties 绑定到结构化对象来访问。这里我们介绍三种配置文件的注入方式，分别是基于配置文件注入、基于非配置文件注入、默认值注入。

3、基于配置文件注入

基于配置文件注入，顾名思义起源头来自于配置文件。这些配置文件诸如 application.properties 或自定义的*.properties 文件。

例如，application.properties 配置文件中定义属性值的形式如下：

```
user.name=admin
```

假设存在自定义配置文件 my.properties，配置文件中定义的属性如下：

```
user.password=123456
```

如果需要做在类中使用这两个配置值如何做呢，看如下编码：

```
@PropertySource("classpath:my.properties")
@RestController
public class ValueController {
    /**
     * 获取位于 application.properties 中配置的属性
     */
    @Value("${user.name}")
    private String name;

    /**
     * 获取位于 my.properties 中的配置属性
     */
    @Value("${user.password}")
    private String password;
}
```

从上面代码可以看出从 application.properties 配置中获取 user.name 的信息填充到变量 name 中，再从自定义配置文件 my.properties 中获取 user.password 信息填充到 password 变量中。

需要注意的是，如果是自定义的 my.properties 文件，需要在某个类中通过 @PropertySource 引入该配置文件，而 application.properties 中的属性会自动被加载。

通过@Value 注入单个属性的同时，也可以注入数组和列表形式。如果存在以下配置：

```
tools=car,train,airplane
```

那么可以通过如下代码进行注入：

```
/**
* 注入数组（自动根据","分割）
*/
@Value("${tools}")
```

```
private String[] toolArray;
/**
* 注入列表形式（自动根据","分割）
*/
@Value("${tools}")
private List<String> toolList;
```

Spring 默认情况下会以“,”进行分割，将配置文件中的 tool 包含内容切割成“car”、“train”、“airplane”组成的数组或者列表然后保存到 toolArray 的数组和 toolList 的列表中。

4、基于非配置文件注入

基于非配置文件注入，需要用到 SpEL（Spring Expression Language）即 Spring 表达式语言对@Value 进行修饰从而传递配置信息。

下面就来看看应用场景：

注入普通字符串，相当于直接给属性默认值，代码如下：

```
@Value("测试")
private String wechatSubscription;
```

注入操作系统属性，例如操作系统的名字，代码如下：

```
@Value("#{systemProperties['os.name']}")
private String systemPropertiesName;
```

注入表达式结果，代码如下：

```
@Value("#{ T(java.lang.Math).random() * 100.0 }")
private double randomNumber;
```

注入其他 Bean 属性：注入 config 对象的属性 tool，代码如下：

```
@Value("#{config.tool}")
private String tool;
```

注入列表形式（自动根据"|"分割），代码如下：

```
@Value("#{'${words}'.split('\\|')}")
private List<String> numList;
```

注入文件资源，代码如下：

```
@Value("classpath:config.xml")
private Resource resourceFile;
```

注入 URL 资源，代码如下：

```
@Value("http://www.choupangxia.com")
private URL homePage;
```

5、默认值注入

说了配置文件和非配置文件的@Value 注入，这里再加入默认值的注入方式，顾名思义就是在配置为空或者没有设置具体值的时候，使用的默认值填充目标对象。这里我们整理了几种情况供大家参考。

如果属性中未配置 IP，则使用默认值，代码如下：

```
@Value("${ip:127.0.0.1}")
private String ip;
```

如果系统属性中未获取到 port 的值，则使用 8888，代码如下：

```
@Value("#{systemProperties['port']?:'8888'}")
private String port;
```

其中${}中直接使用“:”对未定义或为空的值进行默认值设置，而#{}则需要使用“?:”对未设置的属性进行默认值设置。

6、总结

本节课介绍@Value 基本用途和定义，并且通过基于配置文件的注入、基于非配置文件的注入以及默认注入三个方面给大家讲解了@Value 的使用方法。下节课会聊聊 spring IOC 如何解决循环依赖问题。下期见，拜拜。