链滴

SpringBoot 配置文件详细解析

作者：Javen

原文链接：https://ld246.com/article/1497246905543

来源网站：链滴

相信很多人选择Spring Boot主要是考虑到它既能兼顾Spring的强大功能，还能实现快速开发的便捷我们在Spring Boot使用过程中，最直观的感受就是没有了原来自己整合Spring应用时繁多的XML配内容，替代它的是在pom.xml中引入模块化的Starter POMs，其中各个模块都有自己的默认配置，以如果不是特殊应用场景，就只需要在application.properties中完成一些属性配置就能开启各模块应用。

在之前的各篇文章中都有提及关于application.properties的使用，主要用来配置数据库连接、日志关配置等。除了这些配置内容之外，本文将具体介绍一些在application.properties配置中的其他特和使用方法。

自定义属性与加载

我们在使用Spring Boot的时候，通常也需要定义一些自己使用的属性，我们可以如下方式直接定义：

```
xyz.aihiaihi.javen.name=七三
xyz.aihiaihi.javen.title=Spring Boot教程
```

然后通过@Value("${属性名}")注解来加载对应的配置属性，具体如下：

```
@Component
public class BlogProperties {
    @Value("${xyz.aihiaihi.javen.name}")
    private String name;
    @Value("$xyz.aihiaihi.javen.title}")
    private String title;
    // 省略getter和setter
}
```

按照惯例，通过单元测试来验证BlogProperties中的属性是否已经根据配置文件加载了。

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(Application.class)
public class ApplicationTests {
    @Autowired
    private BlogProperties blogProperties;
    @Test
    public void getHello() throws Exception {
        Assert.assertEquals(blogProperties.getName(), "七三");
        Assert.assertEquals(blogProperties.getTitle(), "Spring Boot教程");
    }
}
```

参数间的引用

在application.properties中的各个参数之间也可以直接引用来使用，就像下面的设置：

```
xyz.aihiaihi.javen.name=七三
xyz.aihiaihi.javen.title=Spring Boot教程
xyz.aihiaihi.javen.desc=${com.didispace.blog.name}正在努力写《${com.didispace.blog.title}
```

com.didispace.blog.desc参数引用了上文中定义的name和title属性，最后该属性的值就是七三正在力写《Spring Boot教程》。

使用随机数

在一些情况下，有些参数我们需要希望它不是一个固定的值，比如密钥、服务端口等。Spring Boot属性配置文件中可以通过${random}来产生int值、long值或者string字符串，来支持属性的随机值。

随机字符串

xyz.aihiaihi.javen.value=${random.value}

随机int

xyz.aihiaihi.javen.number=${random.int}

随机long

xyz.aihiaihi.javen.bignumber=${random.long}

10以内的随机数

xyz.aihiaihi.javen.test1=${random.int(10)}

10-20的随机数

xyz.aihiaihi.javen.test2=${random.int[10,20]}

通过命令行设置属性值

相信使用过一段时间Spring Boot的用户，一定知道这条命令：java -jar xxx.jar --server.port=8888通过使用–server.port属性来设置xxx.jar应用的端口为8888。

在命令行运行时，连续的两个减号--就是对application.properties中的属性值进行赋值的标识。所以java -jar xxx.jar --server.port=8888命令，等价于我们在application.properties中添加属性server.port=8888，该设置在样例工程中可见，读者可通过删除该值或使用命令行来设置该值来验证。

通过命令行来修改属性值固然提供了不错的便利性，但是通过命令行就能更改应用运行的参数，那岂是很不安全？是的，所以Spring Boot也贴心的提供了屏蔽命令行访问属性的设置，只需要这句设置能屏蔽：

SpringApplication.setAddCommandLineProperties(false)。

多环境配置

我们在开发Spring Boot应用时，通常同一套程序会被应用和安装到几个不同的环境，比如：开发、试、生产等。其中每个环境的数据库地址、服务器端口等等配置都会不同，如果在为不同环境打包时要频繁修改配置文件的话，那必将是个非常繁琐且容易发生错误的事。

对于多环境的配置，各种项目构建工具或是框架的基本思路是一致的，通过配置多份不同环境的配置件，再通过打包命令指定需要打包的内容之后进行区分打包，Spring Boot也不例外，或者说更加简。

在Spring Boot中多环境配置文件名需要满足application-{profile}.properties的格式，其中{profile}应你的环境标识，比如：

application-dev.properties：开发环境
application-test.properties：测试环境
application-prod.properties：生产环境

至于哪个具体的配置文件会被加载，需要在application.properties文件中通过spring.profiles.active性来设置，其值对应{profile}值。

如：spring.profiles.active=test就会加载application-test.properties配置文件内容

下面，以不同环境配置不同的服务端口为例，进行样例实验。

针对各环境新建不同的配置文件application-dev.properties、application-test.properties、application-prod.properties

在这三个文件均都设置不同的server.port属性，如：dev环境设置为1111，test环境设置为2222，prd环境设置为3333

application.properties中设置spring.profiles.active=dev，就是说默认以dev环境设置

测试不同配置的加载

执行java -jar xxx.jar，可以观察到服务端口被设置为1111，也就是默认的开发环境（dev）

执行java -jar xxx.jar --spring.profiles.active=test，可以观察到服务端口被设置为2222，也就是测环境的配置（test）

执行java -jar xxx.jar --spring.profiles.active=prod，可以观察到服务端口被设置为3333，也就是产环境的配置（prod）

按照上面的实验，可以如下总结多环境的配置思路：

application.properties中配置通用内容，并设置spring.profiles.active=dev，以开发环境为默认置

application-{profile}.properties中配置各个环境不同的内容

通过命令行方式去激活不同环境的配置