

狂神说SpringBoot03 : yaml配置注入

秦疆 狂神说 2020-03-10

狂神说SpringBoot系列连载课程，通俗易懂，基于SpringBoot2.2.5版本，欢迎各位狂粉转发关注学习。未经作者授权，禁止转载



yaml语法学习

配置文件

SpringBoot使用一个全局的配置文件，配置文件名称是固定的

- application.properties
 - 语法结构：key=value
- application.yml
 - 语法结构：key: 空格 value

配置文件的作用：修改SpringBoot自动配置的默认值，因为SpringBoot在底层都给我们自动配置好了；

比如我们可以在配置文件中修改Tomcat 默认启动的端口号！测试一下！

```
server.port=8081
```

yaml概述

YAML是 "YAML Ain't a Markup Language"（YAML不是一种标记语言）的递归缩写。在开发的这种语言时，YAML 的意思其实是："Yet Another Markup Language"（仍是一种标记语言）

这种语言以数据作为中心，而不是以标记语言为重点！

以前的配置文件，大多数都是使用xml来配置；比如一个简单的端口配置，我们来对比下yaml和xml

传统xml配置：

```
<server>      <port>8081</port></server>
```

yaml配置：

```
server:      prot: 8080
```

yaml基础语法

说明：语法要求严格！

- 1、空格不能省略
- 2、以缩进来控制层级关系，只要是左边对齐的一列数据都是同一个层级的。
- 3、属性和值的大小写都是十分敏感的。

字面量：普通的值【数字，布尔值，字符串】

字面量直接写在后面就可以，字符串默认不用加上双引号或者单引号；

```
k: v
```

注意：

- “”双引号，不会转义字符串里面的特殊字符，特殊字符会作为本身想表示的意思；

比如：name: "kuang \n shen" 输出：kuang 换行 shen

- "单引号，会转义特殊字符，特殊字符最终会变成和普通字符一样输出

比如：name: 'kuang \n shen' 输出：kuang \n shen

对象、Map（键值对）

#对象、Map格式k: v1: v2:

在下一行来写对象的属性和值得关系，注意缩进；比如：

```
student:      name: qinjiang      age: 3
```

行内写法

```
student: {name: qinjiang,age: 3}
```

数组（ List、set ）

用 - 值表示数组中的一个元素,比如:

```
pets: - cat - dog - pig
```

行内写法

```
pets: [cat,dog,pig]
```

修改SpringBoot的默认端口号

配置文件中添加，端口号的参数，就可以切换端口；

```
server:  port: 8082
```

注入配置文件

yaml文件更强大的地方在于，他可以给我们的实体类直接注入匹配值！

yaml注入配置文件

- 1、在springboot项目中的resources目录下新建一个文件 application.yml
- 2、编写一个实体类 Dog；

```
package com.kuang.springboot.pojo;

@Component //注册bean到容器中public class Dog {    private String name;    p
    private Integer age;        //有无参构造、get、set方法、toString()方法 }
```

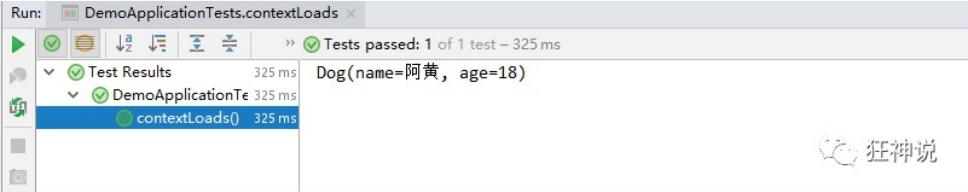
- 3、思考，我们原来是如何给bean注入属性值的！@Value，给狗狗类测试一下：

```
@Component //注册beanpublic class Dog {    @Value("阿黄")    private String name;    @Value("18")    private Integer age;}
```

4、在SpringBoot的测试类下注入狗狗输出一下；

```
@SpringBootTestclass DemoApplicationTests {    @Autowired //将狗狗自动注入进来    Dog dog;    @Test    public void contextLoads() {        System.out.println(dog);    }    //打印看下狗狗对象}
```

结果成功输出，@Value注入成功，这是我们原来的办法对吧。



5、我们在编写一个复杂一点的实体类：Person 类

```
@Component //注册bean到容器中public class Person {    private String name;
private Integer age;    private Boolean happy;    private Date birth;    pr
ivate Map<String,Object> maps;    private List<Object> lists;    private Do
g dog;        //有参无参构造、get、set方法、toString()方法    }
```

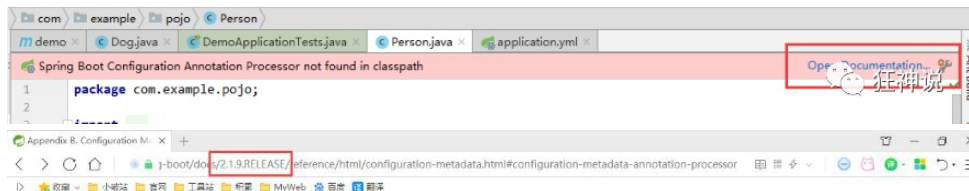
6、我们来使用yaml配置的方式进行注入，大家写的时候注意区别和优势，我们编写一个yaml配置！

```
person:  name: qinjiang  age: 3  happy: false  birth: 2000/01/01  maps: {k1
: v1,k2: v2}  lists:    - code    - girl    - music  dog:    name: 旺财    age:
1
```

7、我们刚才已经把person这个对象的所有值都写好了，我们现在来注入到我们的类中！

/*@ConfigurationProperties作用：将配置文件中配置的每一个属性的值，映射到这个组件中；告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定参数 prefix = "person" ：将配置文件中的person下面的所有属性一一对应*/@Component //注册bean@ConfigurationProperties(prefix = "person")public class Person { private String name; private Integer age; private Boolean happy; private Date birth; private Map<String,Object> maps; private List<Object> lists; private Dog dog;}

8、IDEA 提示，springboot配置注解处理器没有找到，让我们看文档，我们可以查看文档，找到一个依赖！



B.3 Generating Your Own Metadata by Using the Annotation Processor

You can easily generate your own configuration metadata file from items annotated with `@ConfigurationProperties` by using the `spring-boot-configuration-processor` jar. The jar includes a Java annotation processor which is invoked as your project is compiled. To use the processor, include a dependency on `spring-boot-configuration-processor`.

With Maven the dependency should be declared as optional, as shown in the following example:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
```

狂神说

<!-- 导入配置文件处理器，配置文件进行绑定就会有提示，需要重启 --><dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-configuration-processor</artifactId> <optional>true</optional></dependency>

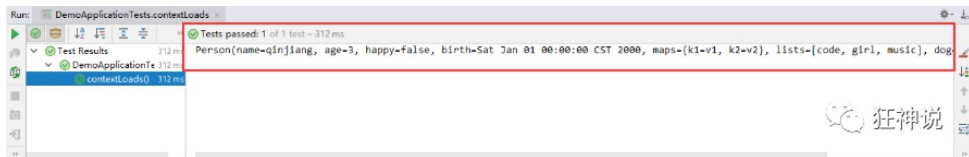
9、确认以上配置都OK之后，我们去测试类中测试一下：

```

@SpringBootTestClass DemoApplicationTests {
    @Autowired    Person person; //将person自动注入进来
    @Test    public void contextLoads() {        System.out.println(person)
; //打印person信息    }
}

```

结果：所有值全部注入成功！



yaml配置注入到实体类完全OK！

课堂测试：

- 1、将配置文件的key 值 和 属性的值设置为不一样，则结果输出为null，注入失败
- 2、在配置一个person2，然后将 @ConfigurationProperties(prefix = "person2") 指向我们的person2；

加载指定的配置文件

@PropertySource：加载指定的配置文件；

@configurationProperties：默认从全局配置文件中获取值；

- 1、我们去在resources目录下新建一个**person.properties**文件

```
name=kuangshen
```

- 2、然后在我们的代码中指定加载**person.properties**文件

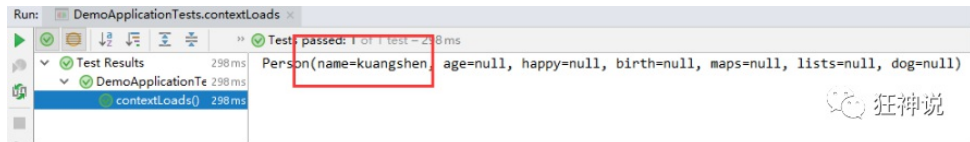
```

@PropertySource(value = "classpath:person.properties")@Component //注册beanp
public class Person {

```

```
@Value("${name}")    private String name;
..... }
```

3、再次输出测试一下：指定配置文件绑定成功！



配置文件占位符

配置文件还可以编写占位符生成随机数

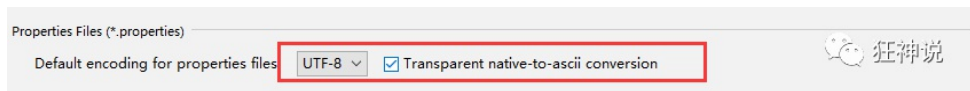
```
person:    name: qinjiang${random.uuid} # 随机uuid    age: ${random.int} #
随机int    happy: false    birth: 2000/01/01    maps: {k1: v1,k2: v2}    lis
ts:        - code        - girl        - music    dog:        name: ${person.hello:
other}_旺财        age: 1
```

回顾properties配置

我们上面采用的yaml方法都是最简单的方式，开发中最常用的；也是springboot所推荐的！那我们来唠唠其他的实现方式，道理都是相同的；写还是那样写；配置文件除了yaml还有我们之前常用的properties，我们没有讲，我们来唠唠！

【注意】properties配置文件在写中文的时候，会有乱码，我们需要去IDEA中设置编码格式为UTF-8；

settings-->FileEncodings 中配置；



测试步骤：

1、新建一个实体类User

```
@Component //注册beanpublic class User {    private String name;    private int age;    private String sex;}
```

2、编辑配置文件 user.properties

```
user1.name=kuangshenuser1.age=18user1.sex=男
```

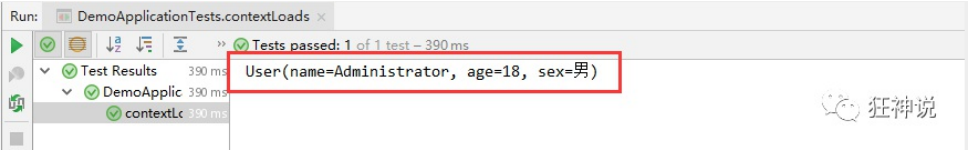
3、我们在User类上使用@Value来进行注入！

```
@Component //注册bean@PropertySource(value = "classpath:user.properties")public class User {    //直接使用@value    @Value("${user.name}") //从配置文件中取值    private String name;    @Value("#{9*2}") // #{SPEL} Spring表达式    private int age;    @Value("男") // 字面量    private String sex;}
```

4、Springboot测试

```
@SpringBootTestclass DemoApplicationTests {  
    @Autowired    User user;  
    @Test    public void contextLoads() {        System.out.println(user);  
    }  
}
```

结果正常输出:



对比小结

@Value这个使用起来并不友好！我们需要为每个属性单独注解赋值，比较麻烦；我们来看个功能对比图

| | @ConfigurationProperties | @Value |
|------------|--------------------------|--------|
| 功能 | 批量注入配置文件中的属性 | 一个个指定 |
| 松散绑定（松散语法） | 支持 | 不支持 |
| SpEL | 不支持 | 支持 |
| JSR303数据校验 | 支持 | 不支持 |
| 复杂类型封装 | 支持 | 不支持 |

- 1、@ConfigurationProperties只需要写一次即可， @Value则需要每个字段都添加
- 2、松散绑定：这个什么意思呢？比如我的yml中写的last-name，这个和lastName是一样的， -后面跟着的字母默认是大写的。这就是松散绑定。可以测试一下
- 3、JSR303数据校验， 这个就是我们可以在字段是增加一层过滤器验证， 可以保证数据的合法性
- 4、复杂类型封装， yml中可以封装对象， 使用value就不支持

结论：

配置yml和配置properties都可以获取到值， 强烈推荐 yml；

如果我们在某个业务中，只需要获取配置文件中的某个值，可以使用一下 @value；

如果说，我们专门编写了一个JavaBean来和配置文件进行一一映射，就直接@configurationProperties， 不要犹豫！



🔊 长按关注

据说关注小狂神的人都
走向人生巅峰了，还不
长按关注一下？



喜欢就在看！

