Spring Boot使用习惯优于配置（项目中存在大量的配置，此外还内置了一个习惯性的配置，让你无需手动进行配置）的理念让你的项目快速运行起来。

### 1、使用配置文件注入属性

Spring Boot默认的配置文件，src/main/resources/application.properties或者

src/main/resources/application.yml，在这里我们可以配置一些常量。

首先我们使用配置文件给一个类注入相关的属性：

com.wuwii.controller.pet.no=${random.uuid}

com.wuwii.controller.pet.name=Tom

通过注解@Value(value=”${config.name}”)就可以绑定到你想要的属性上面。

@RestController

@RequestMapping("/pet")

public class PetController {

@Value("${com.wuwii.controller.pet.no}")

private String no;

@Value("${com.wuwii.controller.pet.name}")

private String name;

@RequestMapping("/d")

public String detail() {

return "no: " + no + ", name: " + name;

}

}

一个个绑定数据不方便，可以新建一个Bean，专门用来绑定注入的属性使用注解@ConfigurationProperties(prefix=”prefix”)，不过需要注意的是先要引入相关依赖。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

通过使用spring-boot-configuration-processor jar，你可以从被@ConfigurationProperties注解的节点产生自己的配置元数据文件。

@ConfigurationProperties(prefix = "com.wuwii.controller.pet")

public class PetBean {

private String no;

private String name;

public String getNo() {

return no;

}

public void setNo(String no) {

this.no = no;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

注意在启动类上加上注解：

@EnableConfigurationProperties({PetBean.class})，开启配置属性

### 2、使用自定义的配置文件

在resources目录下创建一个bean/pet.properties：

com.wuwii.name=Tom

com.wuwii.no=123456

新建一个PetBean1.java：

@PropertySource这个注解可以指定具体的属性配置文件，优先级比较低。

@Configuration

@ConfigurationProperties(prefix = "com.wuwii")

@PropertySource("classpath:bean/pet.properties")

public class PetBean1 {

private String no;

private String name;

//...getter/setter方法

### 3、应用配置文件（.properties或.yml）

.yml中冒号的后面一定要加空格。

server.port=8080

server:

port: 8080

### 4、随机数

配置文件中${random}可以用来生成各种不同类型的随机值。

my.secret=${random.value}

my.number=${random.int}

my.bignumber=${random.long}

my.number.less.than.ten=${random.int(10)}

my.number.in.range=${random.int[1024,65536]}

### 4、属性占位符

app.name=MyApp

app.description=${app.name} is a Spring Boot application

可以在配置文件中引用前面配置过的属性（优先级前面配置过的这里都能用）。

通过${app.name:默认名称}方法还可以设置默认值，当找不到引用的属性时，会使用默认的属性。

### 5、属性名匹配规则

例如有如下配置对象：

@Component

@ConfigurationProperties(prefix="person")

public class ConnectionSettings {

private String firstName;

}

FirstName可以使用的属性名如下：

* person.firstName，标准驼峰式。
* person.first-name，虚线分割方式，推荐在.properties和.yml配置文件中使用。
* PSERSON\_FIRST\_NAME，大写下划线形式，建议在系统环境变量中使用。

### 6、属性验证

可以使用JSR-303注解进行验证，例如：

@Component

@ConfigurationProperties(prefix="connection")

public class ConnectionSettings {

@NotNull

private InetAddress remoteAddress;

// ... getters and setters

}

### 7、配置文件的优先级

1. 命令行参数
2. 来自java:comp/env的JNDI属性
3. Java系统属性（System.getProperties()）
4. 操作系统环境变量
5. RandomValuePropertySource配置的random.\*属性值
6. jar包外部的application-{profile}.properties或application.yml(带spring.profile)配置文件
7. jar包内部的application-{profile}.properties或application.yml(带spring.profile)配置文件
8. jar包外部的application.properties或application.yml(不带spring.profile)配置文件
9. jar包内部的application.properties或application.yml(不带spring.profile)配置文件
10. @Configuration注解类上的@PropertySource
11. 通过SpringApplication.setDefaultProperties指定的默认属性

### 8、Profile-多环境配置

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}对应你的环境标识，比如：

application-dev.properties：开发环境

application-prod.properties：生产环境

然后在application.properties加入：

spring.profiles.active=dev

或application.yml中加入：

spring:

profiles:

active: dev

或启动命令：

java -jar xxx.jar --spring.profiles.active=dev

常见的应用程序属性：

[Appendix A. Common application properties](https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html)