# 通过Maven创建springboot

1 创建空maven项目

2 修改pom文件

—— 添加父依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.2.0.RELEASE</version>  </parent> |

—— 添加启动依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

—— 添加打包（jar包）插件（打包是添加也可以）

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

—— 测试依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  <exclusions>  <exclusion>  <groupId>org.junit.vintage</groupId>  <artifactId>junit-vintage-engine</artifactId>  </exclusion>  </exclusions> </dependency> |

—— 书写启动类

|  |
| --- |
| /\*\*  \* @SpringBootApplication 来标注一个主程序类，说明这是一个Spring Boot应用  \*/  @SpringBootApplication  public class HelloWorldMainApplication {  public static void main(String[] args) {  // Spring应用启动起来  SpringApplication.run(HelloWorldMainApplication.class,args);  }  } |

# 配置文件

## 1、配置文件

SpringBoot使用一个全局的配置文件，配置文件名是固定的；

•application.properties

•application.yml

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

## 2、YAML语法：

1、基本语法

k:(空格)v：表示一对键值对（空格必须有）；

以“空格“的缩进来控制层级关系；只要是左对齐的一列数据，都是同一个层级的，属性和值也是大小写敏感

|  |
| --- |
| server:  port: 8081  path: /hello |

2、值的写法

**字面量**：普通的值（数字，字符串，布尔）

k: v：字面直接来写；

​ 字符串默认不用加上单引号或者双引号；

​ ""：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

​ name: "zhangsan \n lisi"：输出；zhangsan 换行 lisi

​ ''：单引号；会转义特殊字符，特殊字符最终只是一个普通的字符串数据

​ name: ‘zhangsan \n lisi’：输出；zhangsan \n lisi

**对象**、Map（属性和值）（键值对）：

​ k: v：在下一行来写对象的属性和值的关系；注意缩进

​ 对象还是k: v的方式

|  |
| --- |
| friends:  lastName: zhangsan  age: 20 |

行内写法

|  |
| --- |
| friends: {lastName: zhangsan,age: 18} |

数组（List、Set）：

用- 值表示数组中的一个元素

|  |
| --- |
| pets:  - cat  - dog  - pig |

行内写法

|  |
| --- |
| pets: [cat,dog,pig] |

## 3、配置文件值注入

===============================配置文件值注入==================================

|  |
| --- |
| person:  lastName: hello  age: 18  boss: false  birth: 2017/12/12  maps: {k1: v1,k2: 12}  lists:  - lisi  - zhaoliu  dog:  name: 小狗  age: 12 |

javaBean：

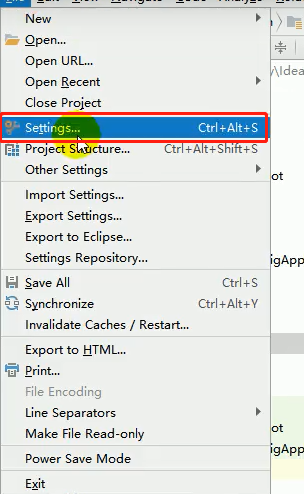
|  |
| --- |
| /\*\*  \* 将配置文件中配置的每一个属性的值，映射到这个组件中  \* @ConfigurationProperties：告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定；  \* prefix = "person"：配置文件中哪个下面的所有属性进行一一映射  \*  \* 只有这个组件是容器中的组件，才能容器提供的@ConfigurationProperties功能；  \* @Component 将当前类交个springboot容器管理  \*  \*/  @Component  @ConfigurationProperties(prefix = "person")  public class Person {  private String lastName;  private Integer age;  private Boolean boss;  private Date birth;  private Map<String,Object> maps;  private List<Object> lists;  private Dog dog; |

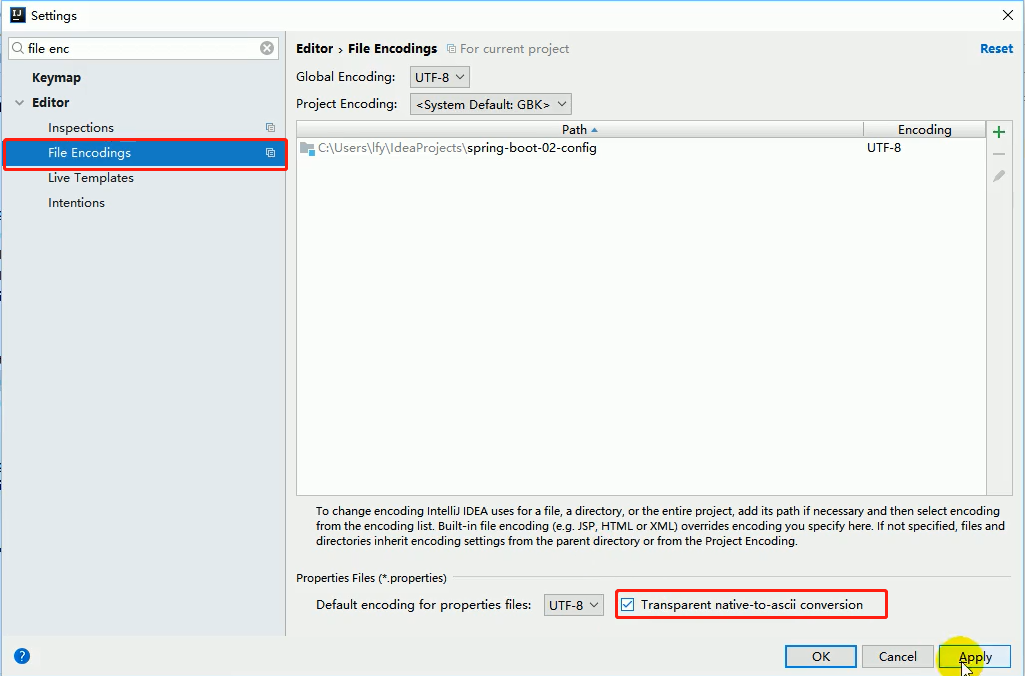
我们可以导入配置文件处理器，以后编写配置就有提示了

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

1. properties配置文件在idea中默认utf-8可能会乱码

调整





2、@Value获取值和@ConfigurationProperties获取值比较

|  |  |  |
| --- | --- | --- |
|  | @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 |

配置文件yml还是properties他们都能获取到值；

如果说，我们只是在某个业务逻辑中需要获取一下配置文件中的某项值，使用

如果说，我们专门编写了一个javaBean来和配置文件进行映射，我们就直接使用

3、配置文件注入值数据校验

|  |
| --- |
| @Component  @ConfigurationProperties(prefix = "person")  @Validated  public class Person {  /\*\*  \* <bean class="Person">  \* <property name="lastName" value="字面量/${key}从环境变量、配置文件中获取值/#{SpEL}"></property>  \* <bean/>  \*/  //lastName必须是邮箱格式  @Email  //@Value("${person.last-name}")  private String lastName;  //@Value("#{11\*2}")  private Integer age;  //@Value("true")  private Boolean boss;  private Date birth;  private Map<String,Object> maps;  private List<Object> lists;  private Dog dog; |

4、@PropertySource&@ImportResource&@Bean

@\*\*PropertySource\*\*：加载指定的配置文件(外部配置文件)；

|  |
| --- |
| /\*\*  \* 将配置文件中配置的每一个属性的值，映射到这个组件中  \* @ConfigurationProperties：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；  \* prefix = "person"：配置文件中哪个下面的所有属性进行一一映射  \*  \* 只有这个组件是容器中的组件，才能容器提供的@ConfigurationProperties功能；  \* @ConfigurationProperties(prefix = "person")默认从全局配置文件中获取值；  \*  \*/  @PropertySource(value = {"classpath:person.properties"})  @Component  @ConfigurationProperties(prefix = "person")  //@Validated  public class Person {  /\*\*  \* <bean class="Person">  \* <property name="lastName" value="字面量/${key}从环境变量、配置文件中获取值/#{SpEL}"></property>  \* <bean/>  \*/  //lastName必须是邮箱格式  // @Email  //@Value("${person.last-name}")  private String lastName;  //@Value("#{11\*2}")  private Integer age;  //@Value("true")  private Boolean boss; |

@\*\*ImportResource\*\*：导入Spring的配置文件，让配置文件里面的内容生效；

Spring Boot里面没有Spring的配置文件，我们自己编写的配置文件，也不能自动识别；

想让Spring的配置文件生效，加载进来；@\*\*ImportResource\*\*标注在一个配置类上

|  |
| --- |
| @ImportResource(locations = {"classpath:beans.xml"})  导入Spring的配置文件让其生效  类略 |

Xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  <bean id="helloService" class="com.atguigu.springboot.service.HelloService"></bean>  </beans> |

SpringBoot推荐给容器中添加组件的方式；推荐使用全注解的方式；不适用spring配置文件xml

1、配置类\*\*@Configuration\*\*------>Spring配置文件

1. 使用\*\*@Bean\*\*给容器中添加组件

|  |
| --- |
| /\*\*  \* @Configuration：指明当前类是一个配置类；就是来替代之前的Spring配置文件  \*  \* 在配置文件中用<bean><bean/>标签添加组件  \*  \*/  @Configuration  public class MyAppConfig {  //将方法的返回值添加到容器中；容器中这个组件默认的id就是方法名  @Bean  public HelloService helloService02(){  System.out.println("配置类@Bean给容器中添加组件了...");  return new HelloService();  }  } |

==============================配置文件值注入==================================

## 4配置文件占位符${}

1. 随机数

|  |
| --- |
| ${random.value}、${random.int}、${random.long}  ${random.int(10)}、${random.int[1024,65536]} |

1. 占位符获取之前配置的值，如果没有可以是用:指定默认值

|  |
| --- |
| person.last-name=张三${random.uuid}  person.age=${random.int}  person.birth=2017/12/15  person.boss=false  person.maps.k1=v1  person.maps.k2=14  person.lists=a,b,c  person.dog.name=${person.hello:hello}\_dog  person.dog.age=15 |

## 5、Profile 多环境支持

1、多Profile文件

我们在主配置文件编写的时候，文件名可以是 application-{profile}.properties/yml

例如：application.properties；application-dev.properties；application-pord.properties；

默认使用application.properties的配置；

2、yml(application.yml)支持多文档块方式 三个中横线隔开 ---

|  |
| --- |
| server:  port: 8081  spring:  profiles:  active: prod  ---  server:  port: 8083  spring:  profiles: dev  ---  server:  port: 8084  spring:  profiles: prod #指定属于哪个环境 |

3、激活指定profile

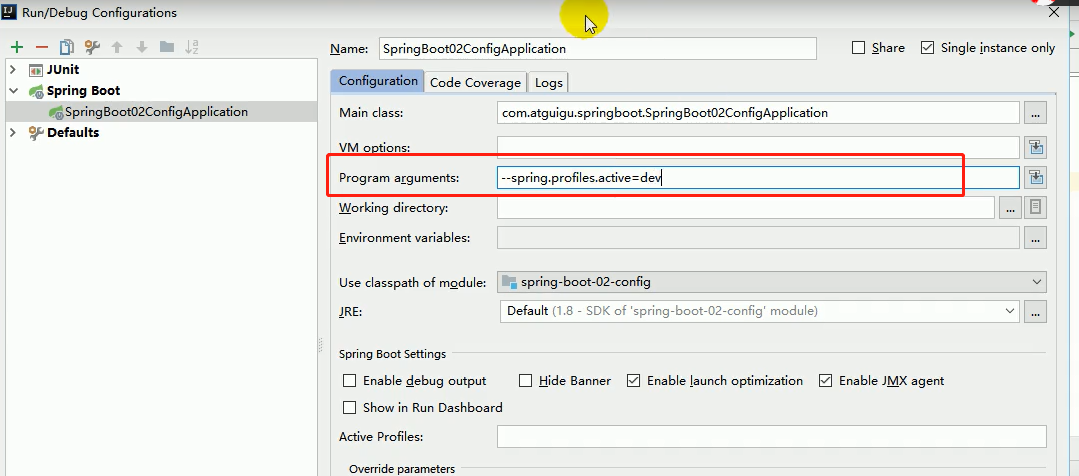
1、在配置文件中指定 spring.profiles.active=dev

2、命令行：

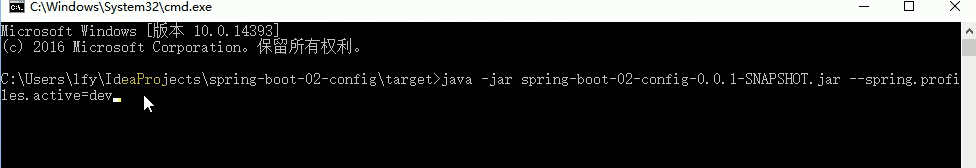
​ java -jar spring-boot-02-config-0.0.1-SNAPSHOT.jar --spring.profiles.active=dev；

​ 可以直接在测试的时候，配置传入命令行参数

Idea 中



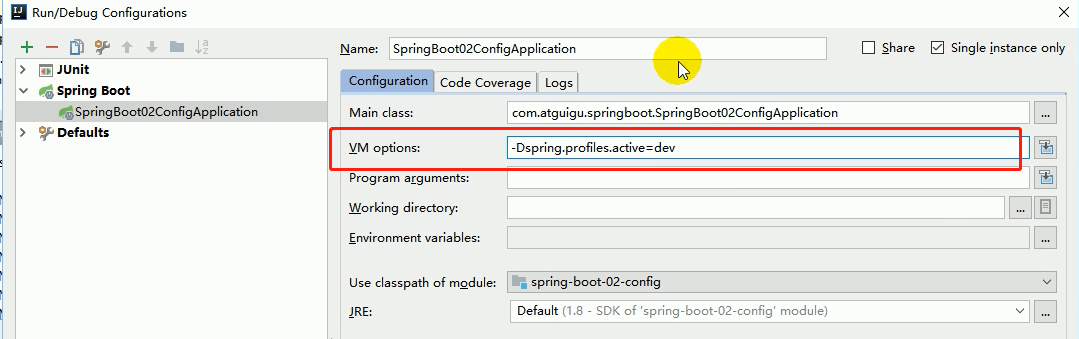
Cmd 中



3、虚拟机参数；

​ -Dspring.profiles.active=dev

Idea中



## 6、配置文件加载位置

springboot 启动会扫描以下位置的application.properties或者application.yml文件作为Spring boot的默认配置文件

–file:./config/

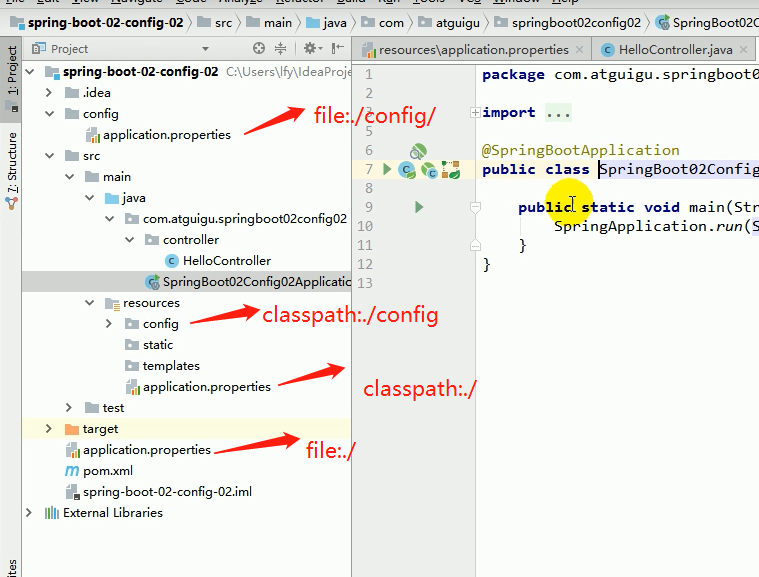
–file:./

–classpath:/config/

–classpath:/

优先级由高到底，高优先级的配置会覆盖低优先级的配置；

SpringBoot会从这四个位置全部加载主配置文件；\*\*互补配置\*\*；



我们还可以通过spring.config.location来改变默认的配置文件位置==

\*\*项目打包好以后，我们可以使用命令行参数的形式，启动项目的时候来指定配置文件的新位置；指定配置文件和默认加载的这些配置文件共同起作用形成互补配置；\*\*

java -jar spring-boot-02-config-02-0.0.1-SNAPSHOT.jar --spring.config.location=G:/application.properties

在部署项目时使用

## 7、外部配置加载顺序

SpringBoot也可以从以下位置加载配置； 优先级从高到低；高优先级的配置覆盖低优先级的配置，所有的配置会形成互补配置

1.命令行参数

所有的配置都可以在命令行上进行指定

|  |
| --- |
| java -jar spring-boot-02-config-02-0.0.1-SNAPSHOT.jar --server.port=8087 --server.context-path=/abc |

多个配置用空格分开； --配置项=值

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

==\*\*由jar包外向jar包内进行寻找；\*\*==

==\*\*优先加载带profile\*\*==

\*\*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\*\*

==\*\*再来加载不带profile\*\*==

\*\*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\*\*

需要现在application.properties中激活 application-{profile}.properties 否则将不会加载application-{profile}.properties

10.@Configuration注解类上的@PropertySource

11.通过SpringApplication.setDefaultProperties指定的默认属性

所有支持的配置加载来源；

[参考官方文档](https://docs.spring.io/spring-boot/docs/1.5.9.RELEASE/reference/htmlsingle/#boot-features-external-config)