# 一、自动配置

Spring Boot 提供了对应用进行自动化配置。相比以前 XML 配置方式，很多显式方式申明是不需要的。二者，大多数默认的配置足够实现开发功能，从而更快速开发。

什么是**自动配置**？

Spring Boot 提供了默认的配置，如默认的 Bean ，去运行 Spring 应用。它是非侵入式的，只提供一个默认实现。

大多数情况下，自动配置的 Bean 满足了现有的业务场景，不需要去覆盖。但如果自动配置做的不够好，需要覆盖配置。比如通过命令行动态指定某个 jar ，按不同环境启动（这个例子在第 4 小节介绍）。那怎么办？这里先要考虑到配置的优先级。

Spring Boot 不单单从 application.properties 获取配置，所以我们可以在程序中多种设置配置属性。按照以下列表的优先级排列：

1.命令行参数

2.java:comp/env 里的 JNDI 属性

3.JVM 系统属性

4.操作系统环境变量

5.RandomValuePropertySource 属性类生成的 random.\* 属性

6.应用以外的 application.properties（或 yml）文件

7.打包在应用内的 application.properties（或 yml）文件

8.在应用 @Configuration 配置类中，用 @PropertySource 注解声明的属性文件

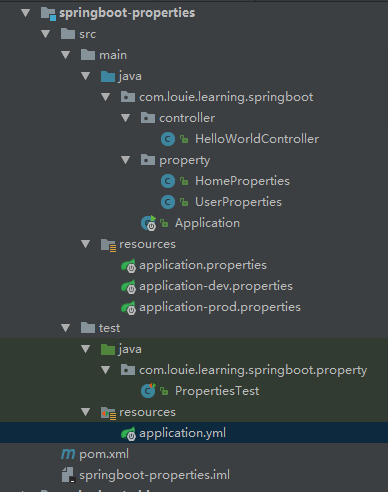
9.SpringApplication.setDefaultProperties 声明的默认属性

可见，命令行参数优先级最高。这个可以根据这个优先级，可以在测试或生产环境中快速地修改配置参数值，而不需要重新打包和部署应用。

还有第 6 点，根据这个在多 moudle 的项目中，比如常见的项目分 api 、service、dao 等 moudles，往往会加一个 deploy moudle 去打包该业务各个子 moudle，应用以外的配置优先。

# 二、自定义属性

项目结构：



在 application.properties 中对应 HomeProperties 对象字段编写属性的 KV 值：

## 家乡属性 Dev  
home.province=浙江  
home.city=温岭  
home.desc=dev: I'm living in ${home.province} ${home.city}.

这里也可以通过占位符，进行属性之间的引用。

然后，编写对应的 HomeProperties Java 对象：

*/\*\*  
 \* 家乡属性  
 \*/*@Component  
@ConfigurationProperties(prefix = "home")  
public class HomeProperties {  
  
 */\*\*  
 \* 省份  
 \*/* private String province;  
  
 */\*\*  
 \* 城市  
 \*/* private String city;  
  
 */\*\*  
 \* 描述  
 \*/* private String desc;  
  
 public String getProvince() {  
 return province;  
 }  
  
 public void setProvince(String province) {  
 this.province = province;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public String getDesc() {  
 return desc;  
 }  
  
 public void setDesc(String desc) {  
 this.desc = desc;  
 }  
  
 @Override  
 public String toString() {  
 return "HomeProperties{" +  
 "province='" + province + '\'' +  
 ", city='" + city + '\'' +  
 ", desc='" + desc + '\'' +  
 '}';  
 }  
}

通过 @ConfigurationProperties(prefix = “home”) 注解，将配置文件中以 home 前缀的属性值自动绑定到对应的字段中。同是用 @Component 作为 Bean 注入到 Spring 容器中。

如果不是用 application.properties 文件，而是用 application.yml 的文件，对应配置如下：

*## 家乡属性***home:  
 province:** 浙江省  
 **city:** 温岭松门  
 **desc:** 我家住在${**home.province**}的${**home.city**}

|  |  |
| --- | --- |
|  |  |

键值对冒号后面，必须空一格。

注意这里，就有一个坑了：

application.properties 配置中文值的时候，读取出来的属性值会出现乱码问题。但是 application.yml 不会出现乱码问题。原因是，Spring Boot 是以 iso-8859 的编码方式读取 application.properties 配置文件。

注意这里，还有一个坑：

如果定义一个键值对 user.name=xxx ,这里会读取不到对应写的属性值。为什么呢？Spring Boot 的默认 StandardEnvironment 首先将会加载 “systemEnvironment” 作为首个PropertySource. 而 source 即为System.getProperties().当 getProperty时,按照读取顺序,返回 “systemEnvironment” 的值.即 System.getProperty(“user.name“)

（Mac 机子会读自己的登录账号，这里感谢我的死党 <http://rapharino.com/>）

# 三、random.\* 属性

Spring Boot 通过 RandomValuePropertySource 提供了很多关于随机数的工具类。概括可以生成随机字符串、随机 int 、随机 long、某范围的随机数。

运行 springboot-properties 工程 org.spring.springboot.property.PropertiesTest 测试类的 randomTestUser 方法。多次运行，可以发现每次输出不同 User 属性值：

|  |
| --- |
| UserProperties{id=-3135706105861091890, age=41, desc='泥瓦匠叫做3cf8fb2507f64e361f62700bcbd17770', uuid='582bcc01-bb7f-41db-94d5-c22aae186cb4'} |

application.yml 方式的配置如下（ application.properties 形式这里不写了）：

*## 随机属性***user:  
 id:** ${**random.long**}  
 **age:** ${**random.int[1,200]**}  
 **desc:** 泥瓦匠叫做${**random.value**}  
 **uuid:** ${**random.uuid**}

|  |  |
| --- | --- |
|  |  |

# 四、多环境配置

很多场景的配置，比如数据库配置、Redis 配置、注册中心和日志配置等。在不同的环境，我们需要不同的包去运行项目。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-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属性来设置，其值对应{profile}值。

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

# Spring Profiles Active  
spring.profiles.active=dev

|  |  |
| --- | --- |
|  |  |

那运行 springboot-properties 工程中 Application 应用启动类，从控制台中可以看出，是加载了 application-dev.properties 的属性输出：

HomeProperties{province='ZheJiang', city='WenLing', desc='dev: I'm living in ZheJiang WenLing.'}

|  |  |
| --- | --- |
|  |  |

将 spring.profiles.active 设置成 prod，重新运行，可得到 application-prod.properties的属性输出：

HomeProperties{province='ZheJiang', city='WenLing', desc='prod: I'm living in ZheJiang WenLing.'}

|  |  |
| --- | --- |
|  |  |

根据优先级，顺便介绍下 jar 运行的方式，通过设置 -Dspring.profiles.active=prod 去指定相应的配置:

mvn package

java -jar -Dspring.profiles.active=prod springboot-properties-0.0.1-SNAPSHOT.jar

## **多环境高级应用**

     在某些情况下，应用的某些业务逻辑可能需要有不同的实现。例如邮件服务，假设EmailService中包含的send(String email)方法向指定地址发送电子邮件，但是我们仅仅希望在生产环境中才执行真正发送邮件的代码，而开发环境里则不发送以免向用户发送无意义的垃圾邮件。

       我们可以借助Spring的注解@Profile实现这样的功能，这样需要定义两个实现EmailService借口的类：

/\*\*

 \* 发送邮件接口.

 \*/

**public interface** EmailService {

    /\*\*发送邮件\*/

**public void** send();

}

发送邮件的具体实现（dev-开发环境的代码）：

@Service

@Profile("dev") //开发环境的时候.

**public class** DevEmailServiceImpl **implements** EmailService{

    @Override

**public void** send() {

       System.**out**.println("DevEmailServiceImpl.send().开发环境不执行邮件的发送.");

    }

}

发送邮件的具体实现（prod-生产环境的代码）：

@Service

@Profile("prod") //生产环境.

**public class** ProdEmailServiceImpl2 **implements** EmailService{

    @Override

**public void** send() {

       System.**out**.println("ProdEmailServiceImpl.send().生产环境执行邮件的发送.");

       //具体的邮件发送代码.

       //mail.send();

    }

}

@Profile("dev")表明只有Spring定义的Profile为dev时才会实例化DevEmailService这个类。

|  |  |
| --- | --- |
|  |  |

# 五、小结

常用的样板配置在 Spring Boot 官方文档给出，我们常在 application.properties（或 yml）去配置各种常用配置：

<http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

感谢资料：

<http://blog.didispace.com/springbootproperties/>

<https://docs.spring.io/spring-boot/docs>