Spring Boot 配置文件

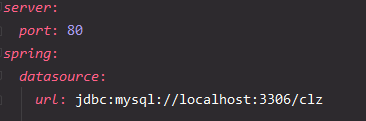
# 配置文件

1. Spring Boot使用一个全局的配置文件，配置文件名是固定的
   1. application.properties
   2. application.yml
2. 配置文件的作用
   1. 修改Spring Boot自动配置的默认值

# YAML方式

1. YAML Ain’t Markup Language
2. 以数据为中心，比json和xml更适合做配置文件

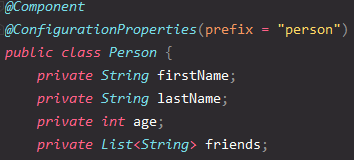
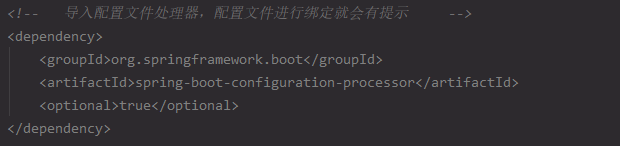
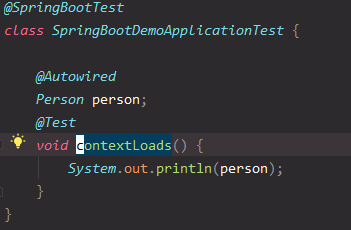
## 基本语法

1. K: V——表示一对键值对（空格必须有）
2. 以空格为缩进来控制层级关系，不允许使用Tab键
3. 大小写敏感

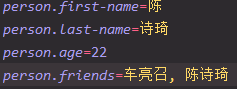
## 数据结构语法

1. 字面量——普通的值（数字，字符串，布尔值）
   1. 默认不带引号
   2. 双引号：不会转义字符
   3. 单引号：会转义字符
2. 对象——Map
   1. 普通写法 
   2. 行内写法 
3. 数组——List，Set
   1. 普通写法 
   2. 行内写法 

## 配置文件值注入

1. 将配置文件中配置的每一个属性的值，映射到组件中
2. @ConfigurationProperties(prefix = “person”)，告诉SpringBoot将本类中的所有属性和配置文件中相关属性进行绑定，prefix = “person”，将配置文件的“person”属性进行一一匹配 
   1. 默认从全局配置文件中获取值
   2. 或者使用@Value()，进行注入
3. 导入配置文件处理器，以后编写配置就会有提示
4. 单元测试 

# Properties方式



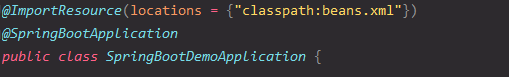
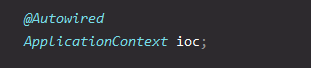
# @Value和@ConfigurationProperties获取值比较

1. @ConfigurationProperties
   1. 批量注入配置文件中的属性
   2. 支持松散语法绑定
      1. firstName
      2. first\_name
      3. first-name
      4. FIRST\_NAME
         1. 推荐系统属性使用这种写法
   3. 不支持表达式
   4. 支持JSR303数据校验
      1. 组件上添加@Validated
      2. 比如属性上添加@Email，将校验格式
   5. 支持复杂类型封装，比如map
2. @Value
   1. 一个个指定
   2. 不支持松散语法绑定
   3. 支持表达式
   4. 不支持JSR303数据校验
   5. 不支持复杂类型封装
3. 如果只需在某个业务逻辑中获取一下配置文件的某项值，使用@Value；如果专门编写了一个JavaBean来和配置文件进行映射时，就使用@ConfigurationProperties

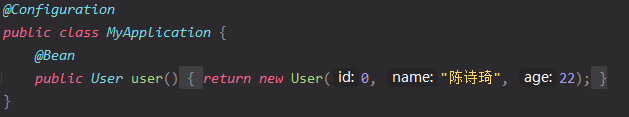
# @PropertySource

* 1. 加载指定的配置文件
     1. @ConfigurationPeroperties默认从全局配置文件中获取值

# @ImportResource

* 1. 导入Spring的配置文件，让配置文件里面的内容生效
  2. Spring boot 中没有Spring的配置文件，我们自己编写的配置文件，也不能自动识别
  3. 想让Spring的配置文件生效，加载进来，将@ImportResource标注在一个配置类上
     1. 其他：获取ioc容器 

# @Configuration和@Bean

* 1. Spring Boot 推荐全注解方式
  2. @Configuration指明当前类是一个配置类
  3. @Bean来给容器中添加组件
  4. 创建一个config包

# 配置文件占位符

1. 配置文件中可以使用随机数
   1. ${random.value}
   2. ${random.int}
   3. ${random.int(10)}
   4. ${random.int[1024, 65536]}
2. 属性配置占位符
   1. 可以在配置文件中引用钱买你配置过的属性
   2. ${app.name:默认值}来指定找不到属性时的默认值

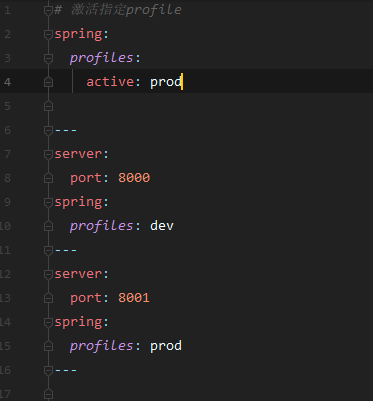
# Profile

Profile时Spring对不同环境提供不同不同配置功能的支持，可以通过激活、指定参数等方式快速切换环境

## 多Profile文件模式

* 1. 我们在主配置文件编写的时候，文件名可以是 application-(profile).properties/yml
  2. 默认使用application.properties/yml的配置

## Yml支持多文档块模式



## 激活指定profile

* 1. 在配置文件中指定： spring.profiles.active=dev
  2. 命令行参数：
     1. --spring.profiles.active=dev
  3. 虚拟机参数：
     1. -Dspring.profiles.active=dev

# 配置文件加载顺序

1. File:./config/
2. File:./
3. Classpath:/config/
4. Classpath:/

* 以上是按照优先级由高到低的顺序，高优先级配置内容会覆盖低优先级配置的内容
* 还可以通过配置spring.config.location来改变默认配置
* 项目打包后还可以通过命令行从外部加载配置文件

# 外部配置文件加载顺序

**SpringBoot也可以从一下位置加载配置；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参数**

1. **命令行参数**
   1. Java -jar springboot --server.port=8082 --server.context.path=/abc
      1. 多个配置用空格分开；--配置项=值
2. 来自java:comp/env的NDI属性
3. Java系统属性
4. 操作系统环境变量
5. RandomValuePropertySource配置的random.\*属性
6. **Jar包外部的application-{profile}.properties（带spring.properties）**
7. **Jar包内部的application-{profile}.properties（带spring.properties）**
8. **Jar包外部的application.properties（不带spring.properties）**
9. **Jar包内部的application.properties（不带spring.properties）**
10. @Configuration注解类上的@PropertySource
11. 通过SpringApplication.setDefaultProperties指定的默认属性

**其他参考官方文档**

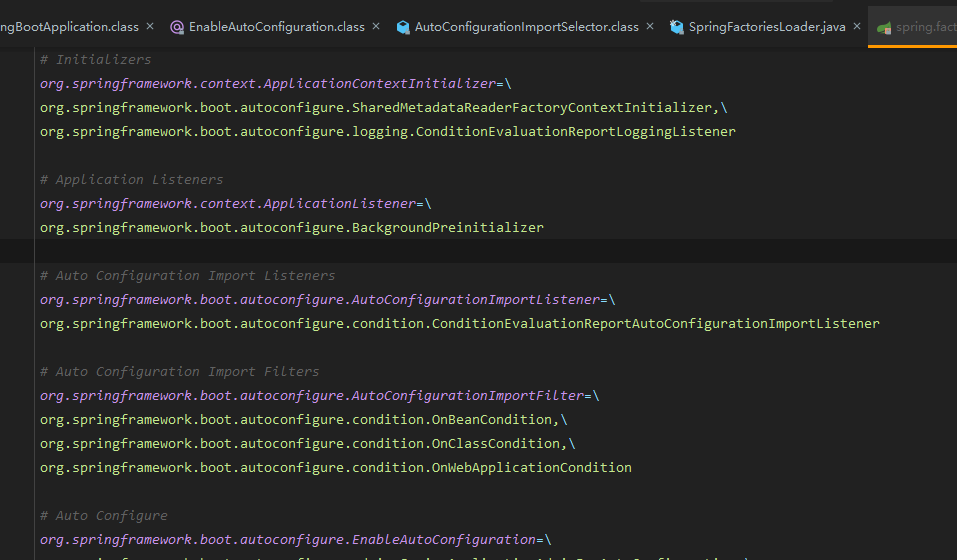
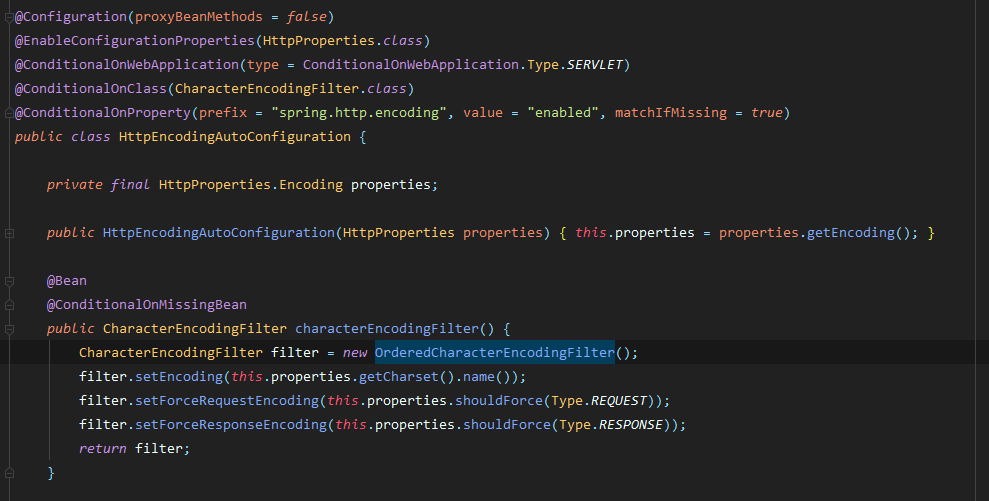
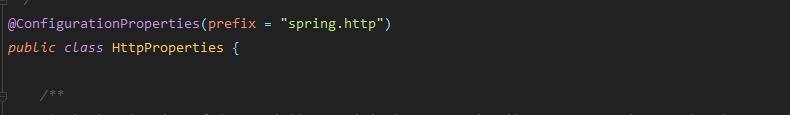
<https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config>

# 自动配置原理

配置文件能配置的属性在以下官方文档里面：

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

## 自动配置原理

1. SpringBoot启动时加载主配置类，开启自动自动配置@EnableAutoConfiguration
2. @EnableAutoConfiguration作用：
   1. 利用EnableAutoConfigurationImportSelector给容器中导入一些组件
   2. 可以插入seectImports()方法的内容；
   3. List<String> configuration = getCandiateConfigurations(annotationMetadata, attributes)； 获取候选的配置
      1. SpringFactoriesLoader.loadFactoryNames()
      2. 扫描所有jar包类路径下 META-INF/spring.factories
      3. 把扫描到的这些文件的内容包装成properties对象
      4. 从properties中获取到EnableAutoConfiguration.class类（类名）对应的值，然后将他们添加在容器中
   4. **将类路径下 META-INF/spring.factories里面的所有EnableAutoConfiguration的值加入到了容器中**每一个这样的xxxAutoConfiguration类都是容器中的一个组件，都加入到容器中；用他们来做自动配置
   5. 每一个自动配置类进行自动配置功能；
   6. 以**HttpEncodingConfiguration（Http编码自动配置）**为例解释自动配置原理根据当前不同的条件判断决定这个配置类是否生效？一旦这个配置类生效，这个配置类就会给容器中添加各种组件；这些组件都是从对应的properties类中获取的，这些类里面的每一个属性又是和配置文件绑定的；
   7. 所有在配置文件中能配置的属性都是在xxxProperties类中封装着；配置文件能配置什么就可以参照某个功能对应的这个属性类
3. SpringBoot的精髓
   1. SpringBoot启动会大量的自动装配配置类
   2. 我们看我们需要的功能有没有SpringBoot默认写好的自动配置类；
   3. 我们再来看这个自动配置类中到底配置了哪些组件；（只有要我们要用的组件，我们就不需要再来配置了）；
   4. 给容器中自动配置类添加组件的时候，会从properties类中获取某些属性，我们就可以在配置文件中指定这些属性的值；
   5. XxxAutoConfiguration：自动配置类；给容器中添加组件
   6. XxxProperties：封装配置文件中相关属性

## 细节

1. @Conditional派生注解
   1. 作用：只有@Conditional指定的条件成立，才给容器中添加组件，配置里面的所有内容才生效；
2. 通过启动debug=true属性；来让控制台打印自动配置报告，这样我们就可以很方便的知道，哪些自动配置类生效。