Spring Boot 笔记

# Spring boot 入门

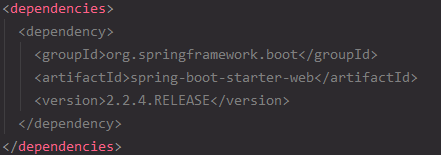
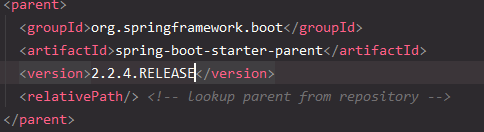
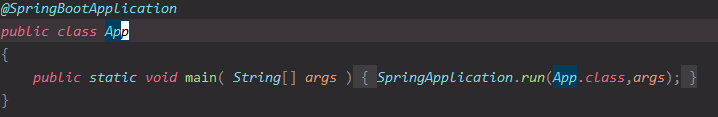
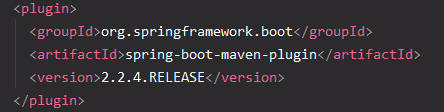
## Spring boot简介

1. 快速创建独立运行的spring项目以及主流框架继承
2. 使用嵌入式的Servlet容器，应用无需打包成war包
3. Starters自动依赖与版本控制
4. 大量的自动配置，简化开发，也可修改默认值
5. 无需配置XML，无需代码生成，开箱即用
6. 准生产环境的运行时应用监控
7. 与云计算的天然集成

## 微服务

1. 2014年martin fowler
2. 架构风格
3. 一个应用应该是一组小型服务，可以通过HTTP方式进行互通
4. 每一个功能元素最终都是一个可独立替换和独立升级的软件单元

## Hello World

1. 创建一个maven项目
2. 导入spring boot相关依赖
3. 编写controller
4. 编写主类
5. 运行主程序
   1. 自带tomcat
6. 打包
   1. 导入插件
      1. 打包成jar包
   2. mvn package
   3. 运行jar包 

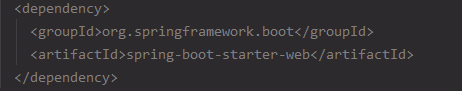
## Hello World探究

### Pom.xml

#### 父项目

1. Springboot父pom管理所有依赖版本
2. 以后导入依赖无需写版本（没有在dependencies里面的依赖需要声明版本号）

#### 启动器

1. Spring-boot-starter-web
   1. Spring-boot-starter
      1. Spring-boot场景启动器
      2. 帮我们导入了web模块模块正常运行所依赖的组件
2. Spring-boot将所有功能场景都抽取出来，做成一个个的starter启动器，只需在项目里面引入这些starter相关场景的所有依赖都会导入进来。

### 主程序类

1. @SpringBootApplication
   1. 标注这个类是SpringBoot的主配置类，springboot就应该运行这个类的main方法来启动springboot应用
   2. **只有主配置类所在包及下面所有子包里面的所有组件扫描到spring容器**
   3. SpringBoot在启动的时候从类路径下的META-INF/spring.factories中获取EnableAutoCondiguration指定的值，将这些值作为自动配置类导入到容器中，自动配置类就生效，帮我们自动配置工作；

## 快速创建

1. 使用Spring Initializer快速创建项目
   1. 选择spring web
   2. 默认生成Spring Boot项目：
      1. 主程序已经生成好
      2. resource文件夹中的目录结构
         1. static：保存所有的静态资源js,css,images；
         2. template：保存所有的模板页面（Spring Boot默认jar包使用嵌入的Tomcat，默认不支持JSP）
         3. application.properties：spring boot应用的默认配置文件；可以修改一些默认设置；

# Spring Boot 配置文件

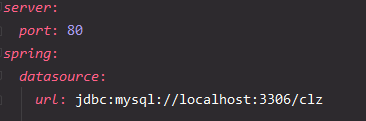
## 配置文件

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

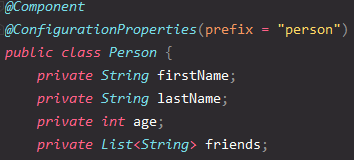
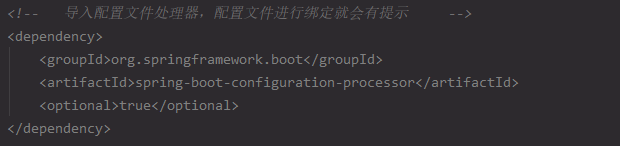
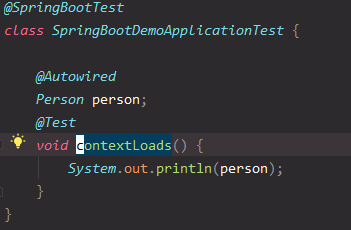
### 基本语法

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

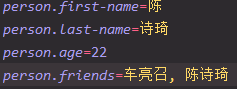
### 数据结构语法

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



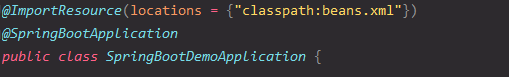
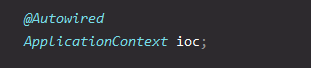
## @Value和@ConfigurationProperties获取值比较

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

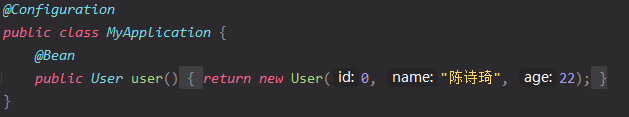
## @PropertySource

* 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

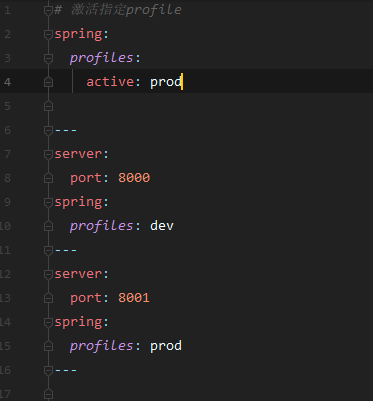
## Profile

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