●09-1

springboot配置文件的作用：修改springboot自动配置的默认值

●10-1 yaml基本语法

**字符串**

key: (空格)value

字符串不需要单引号或双引号

双引号：不会转移字符串内的特殊字符

单引号：会转移字符串内的特殊字符

**对象、Map（键值对）**

friends :

lastname: zhangsan

age: 20

行内写法：

friends: {lastname: zhangsan,age: 20}

**数组（List、Set）**

pets ：

- cat

- dog

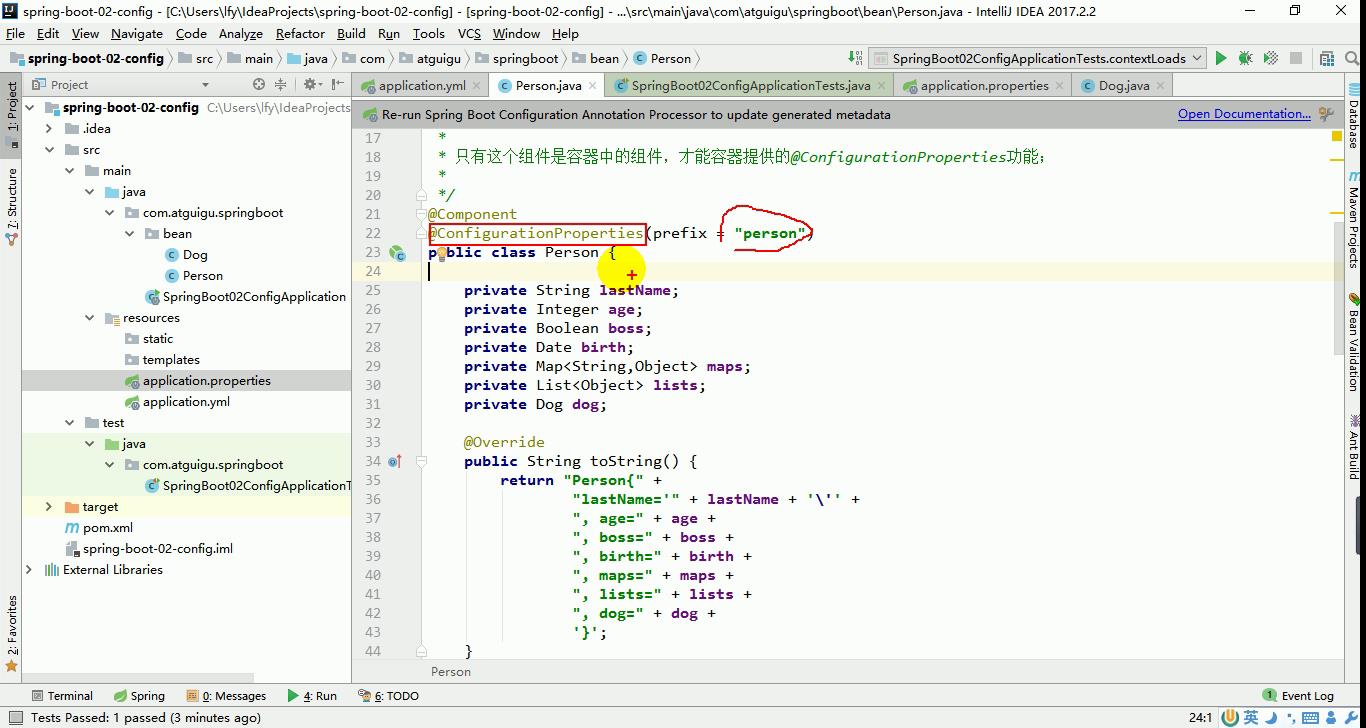
- pig

行内写法

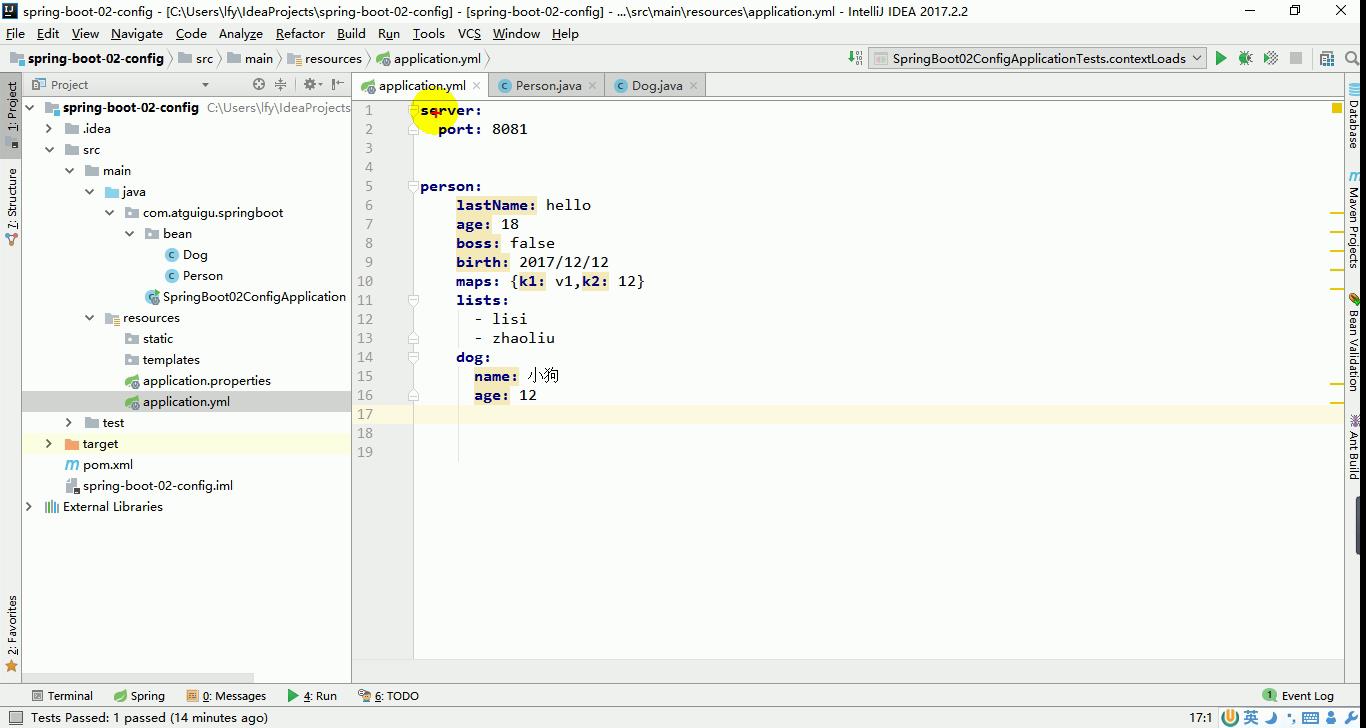
pet： [cat,dog,pig]

●12-1 获取yaml配置文件(sp\_test01)

1、Java中使用@ConfigurationProperties



2、yml文件如下配置



可以导入配置文件处理器spring-boot-configuration-processor，以后会有提示

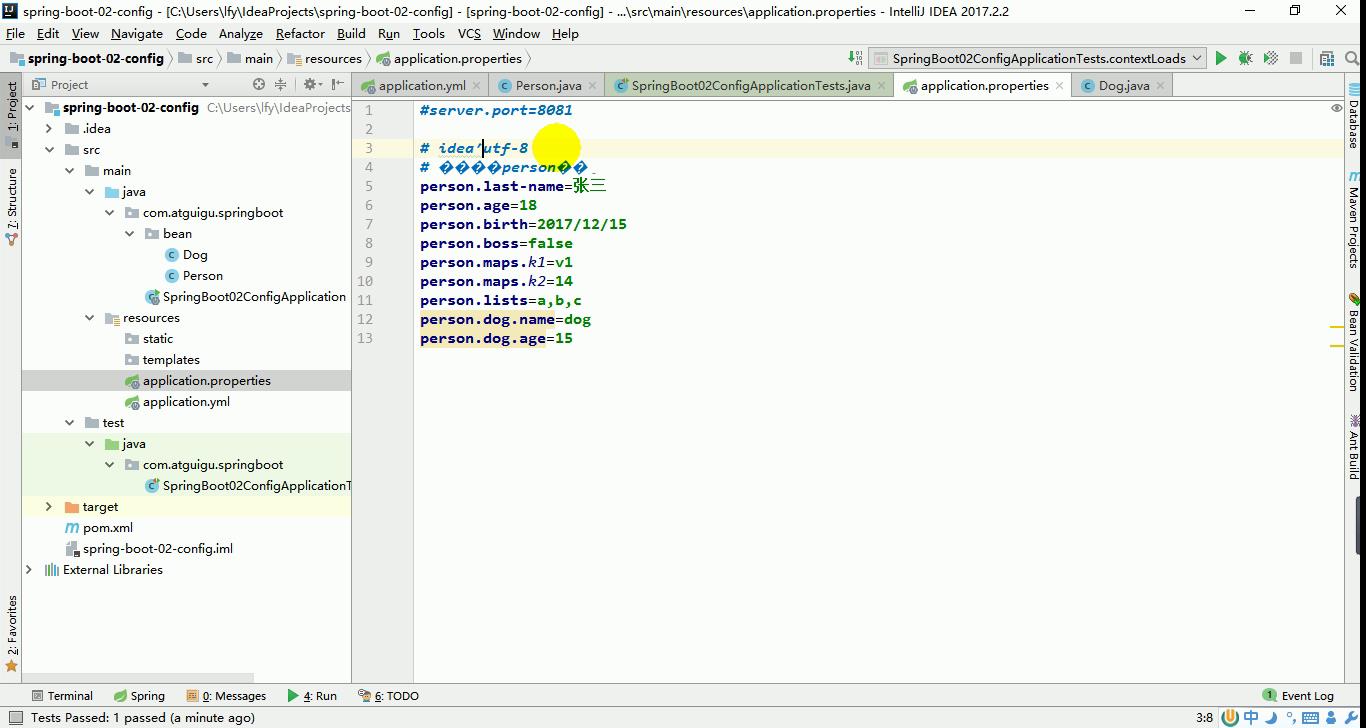
运行结果：

Person [lastName=hello, age=14, boss=false, birth=Fri Dec 12 00:00:00 CST 2008, maps={key1=value1, key2=value2}, lists=[lisi, wangwu, zhangsan], dog=Dog [name=bear, age=12]]

●12-1 获取properties配置文件(sp\_test02)

1、Java文件使用@ConfigurationProperties，与sp \_test01相同

2、properties文件如下配置



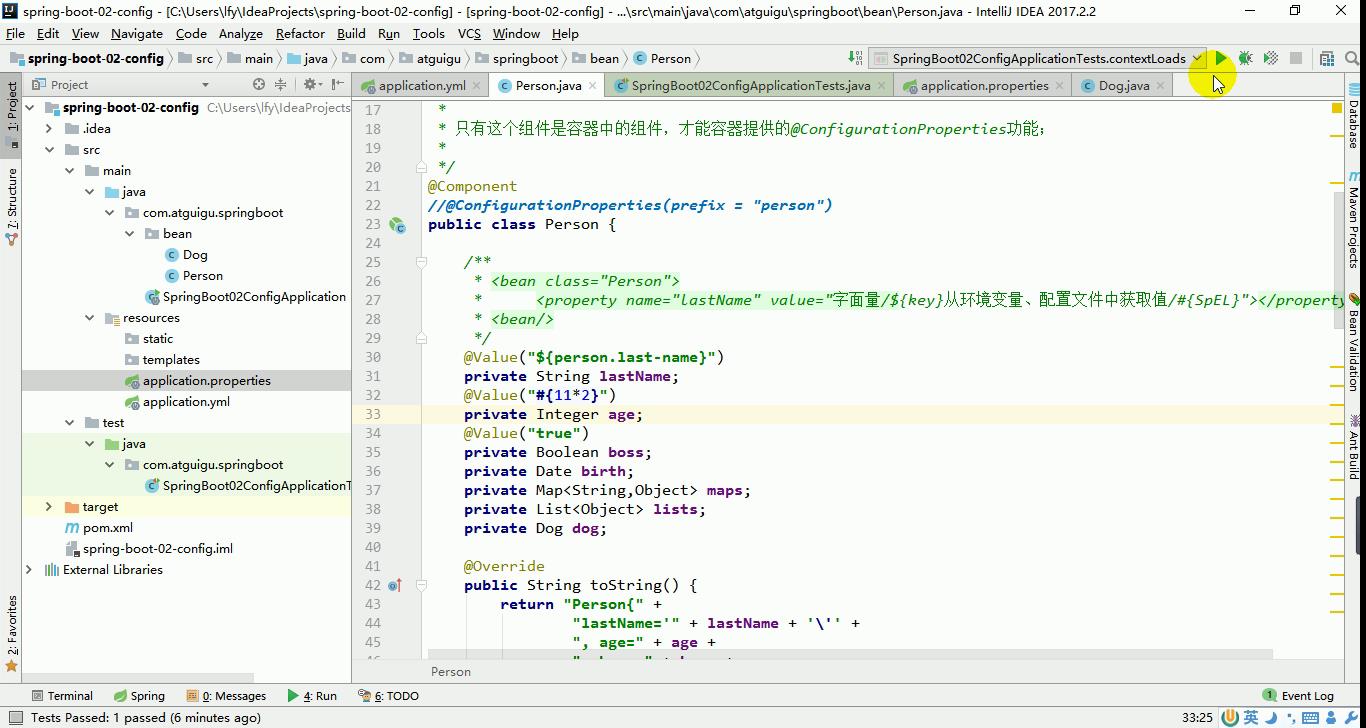
●13-1@Value获取值

@Value获取类型

1、字面常量

2、${key} 从环境变量或配置文件中获取值

3、#{SpEL}



运行结果：

Person [lastName=hello, age=33, boss=true, birth=null, maps=null, lists=null, dog=null]



补充:松散绑定 (表示驼峰式、下划线(\_)、短横线(-))

标准方式 person.firstName

方式一 person.first-name

方式二 persion.first\_name

如果说，在某个业务逻辑中需要获取配置文件的某项值，使用@Value

如果说，专门写了一javaBean来和配置文件进行映射，可直接使用@ConfigurationProperties



●14-1@PropertySource

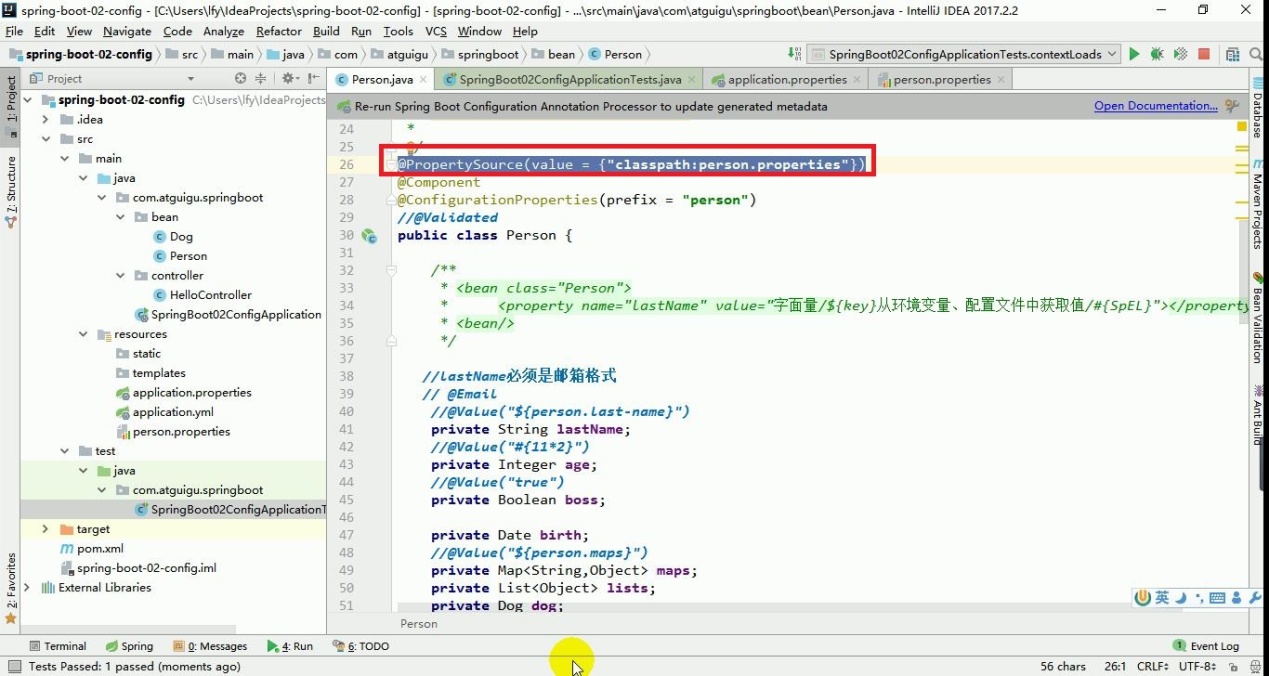
@ConfigurationProperties与下面注解配合使用

-@Bean结合为属性赋值

-@PropertySource 加载指定的properties文件

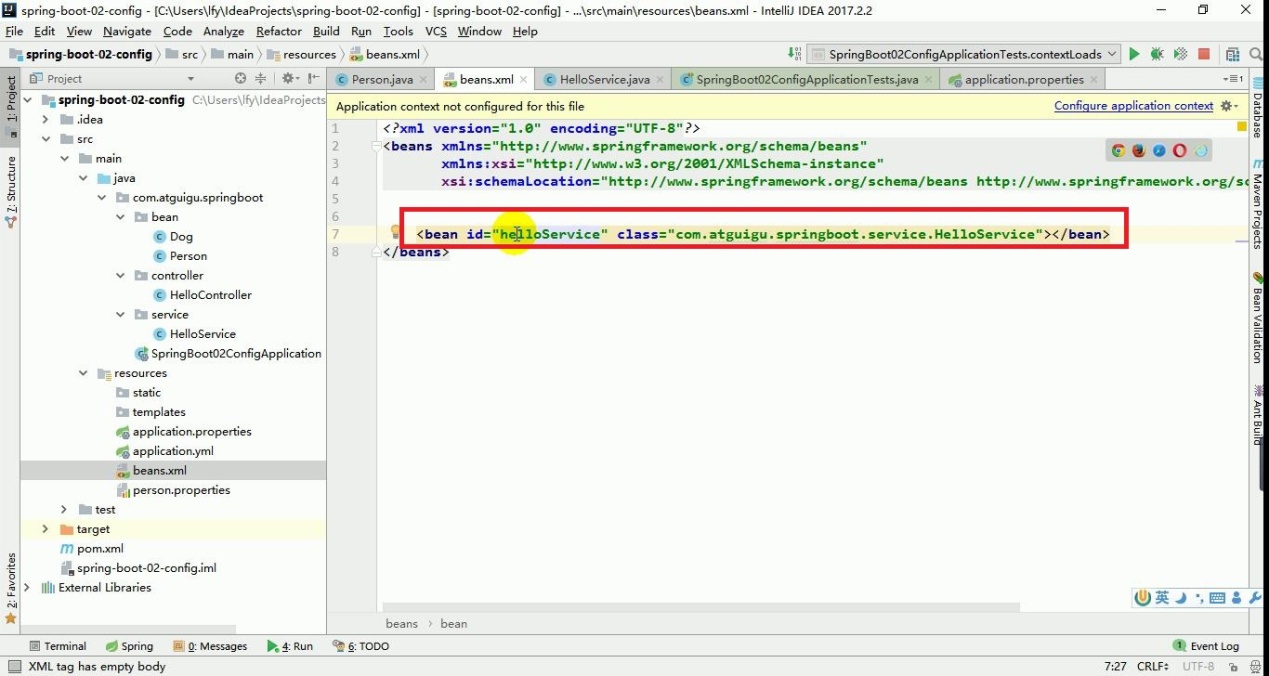
-@ImportSource 导入Spring 的配置文件，使其内容生效

1、与@PropertySource配合使用

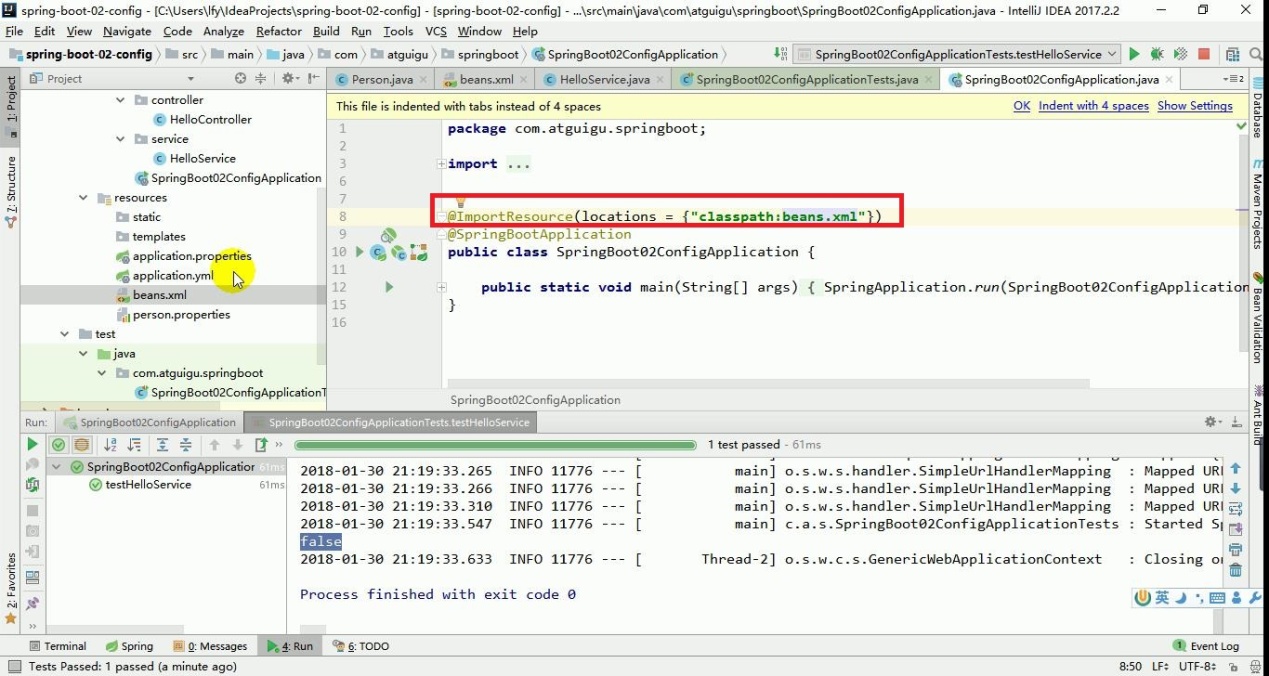


2、与@PropertySource配合使用

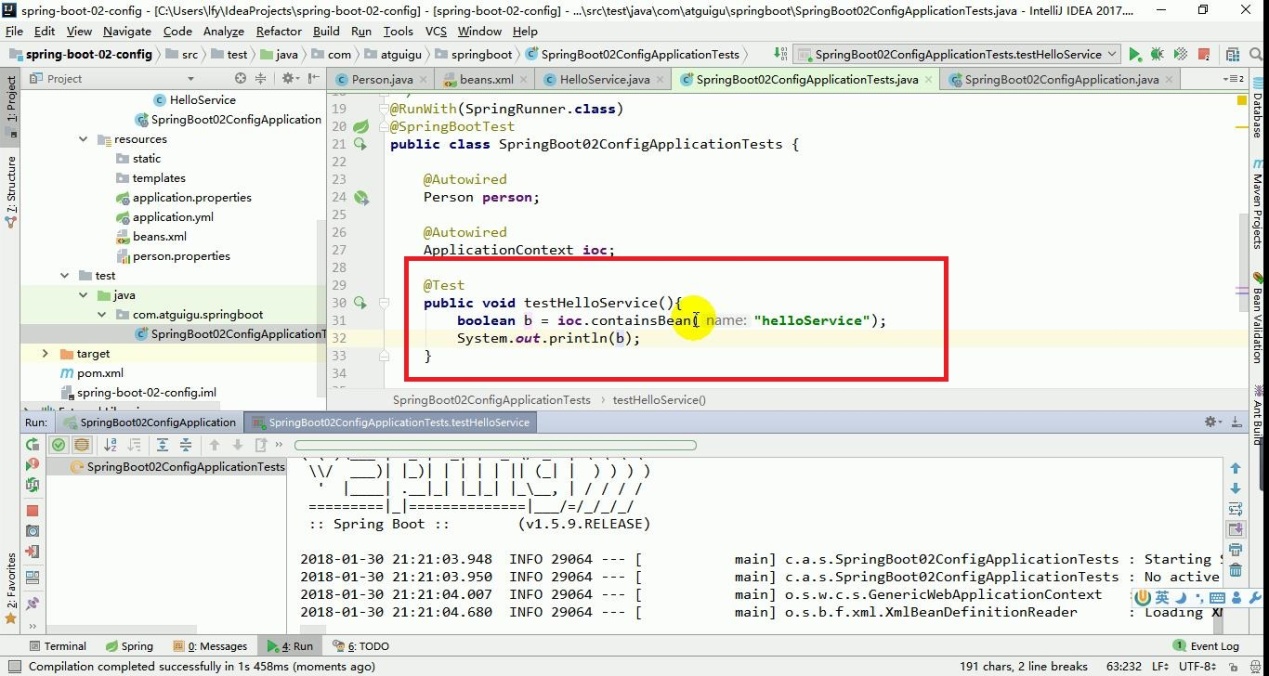
xml配置文件



引入



测试



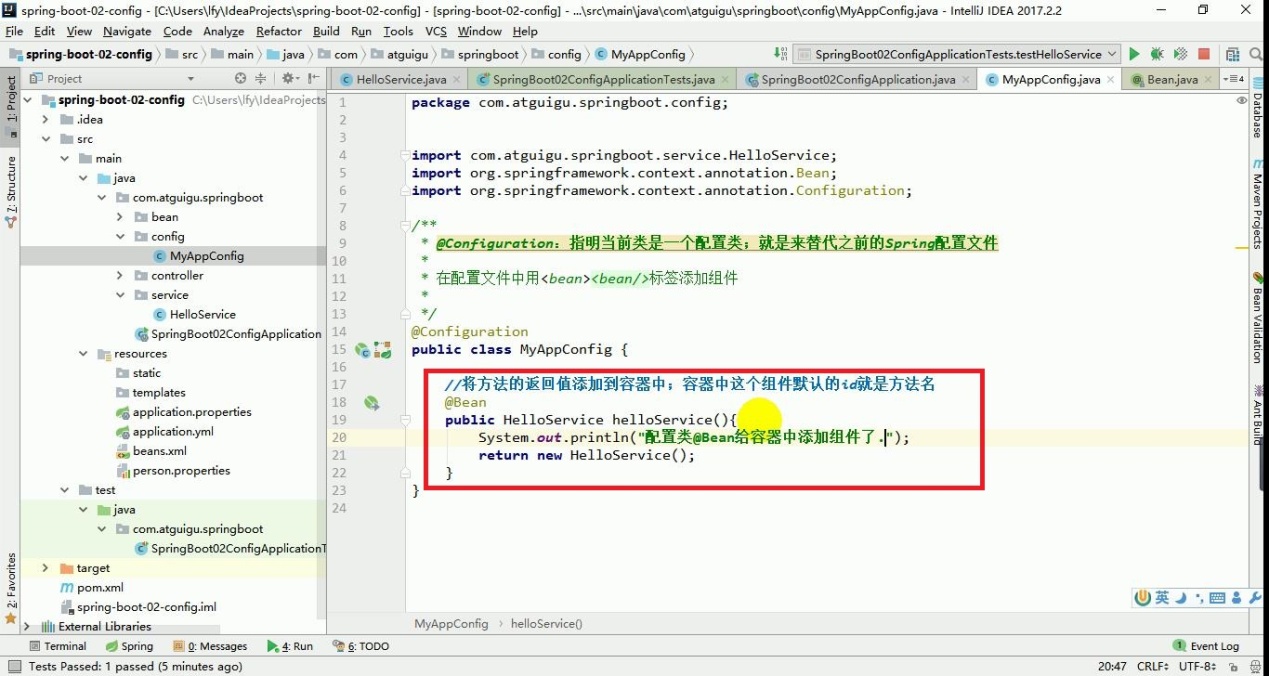
3、与@Bean配合使用

容器添加组件方式：

方式一： 配置类=======Spring配置文件

方式二：使用注解@Bean（推荐）

@Bean作用：将方法的返回值添加到容器中，组件默认 的id就是方法名



●15-1配置文件的占位符

一、RandomValuePropertySource配置文件中可以使用的随机数

${random.value},${random.int },${random.long},

${random.int(10)}, ${random.int[1024,65536]}

二、属性配置占位符

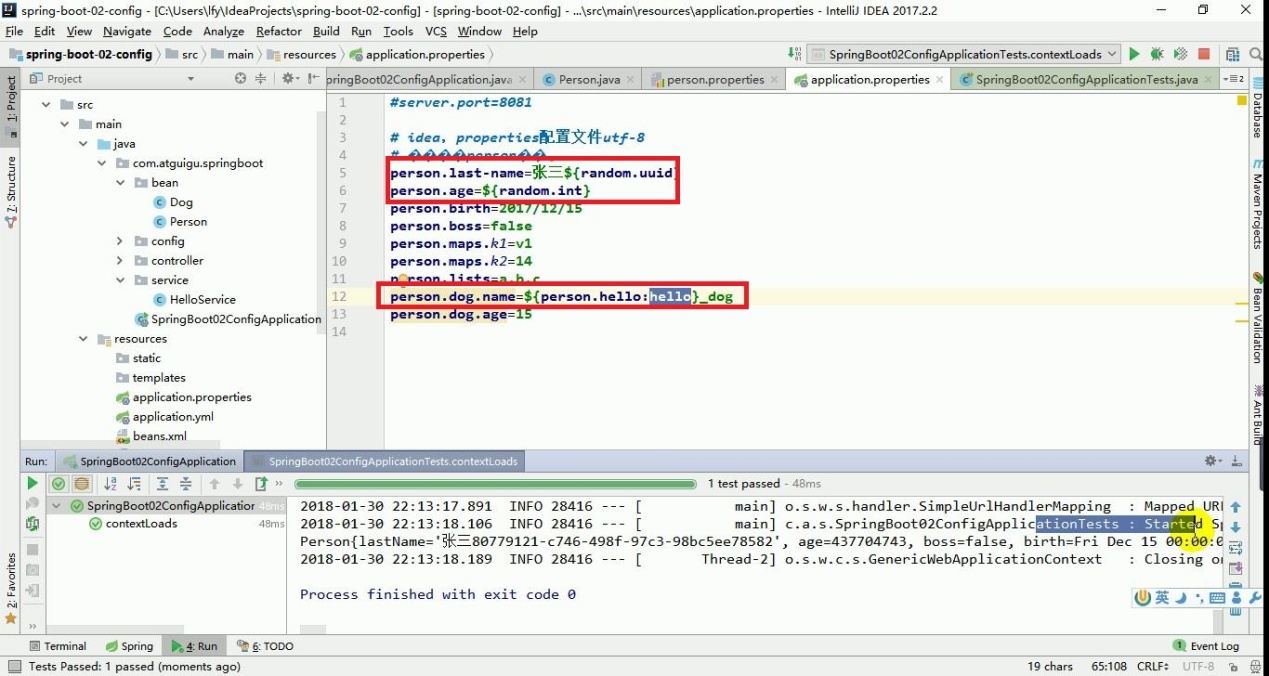
app.name = Myapp

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

说明：可以在配置文件中引用前面配置的属性（优先级前面配过的这里都能用）

${ app.name :默认值}来指定找不到属性的默认值

例如：



●16 Profile多环境支持

为方便切换环境，profile是spring对不同环境提供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。

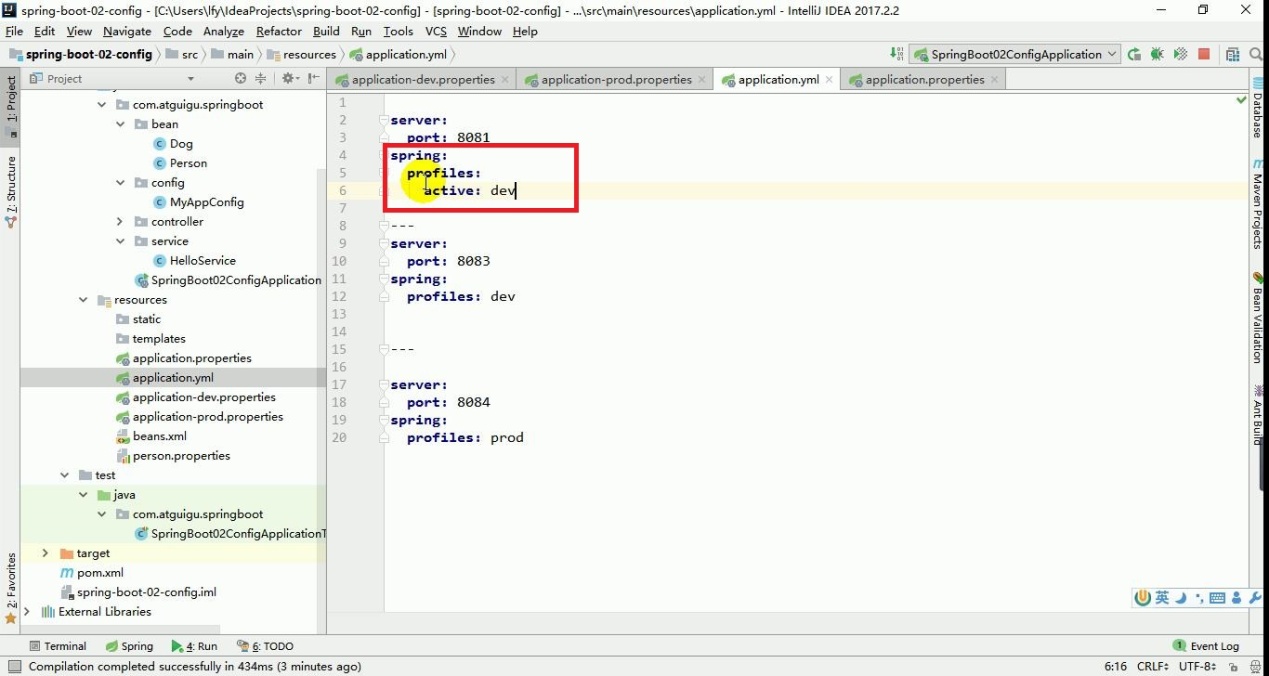
1、多profile文件形式：

-格式：application-{profile}.properties

application-dev.properties

application-prod.properties

2、多profile文档块模式(针对yaml文件)：

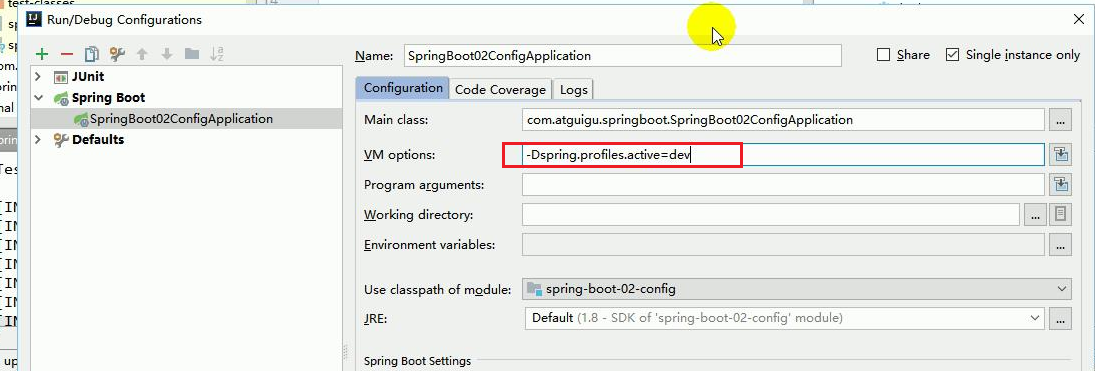


3、激活方式

-命令行 --spring.profiles.active =dev

-配置文件 spring.profiles.active = dev

-jvm参数 -Dspring.profiles.active =dev



●17 配置文件加载位置

配置文件application.properites 和application.yml可以放在以下位置

-file: /config/ (file：当前项目文件路径)

-file: ./

-classpath: /config/ (classpath：/WEB-INF/classpath)

-classpath: /

以上加载顺序按照**优先级有高到低，**所有位置的配置文件都会被加载，并且配置内容是高优先级**覆盖**低优先级。（互补配置）

可以通过配置**spring.config.location** 改变默认位置,应用场景:**项目打包好之后，可以用命令行参数的形式来指定文件 中的新位置；指定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**。

●18 外部配置的加载顺序

参考文档：<https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#boot-features-external-config>

springboot支持外部配置方式，以下 优先级 从高到低

1、**命令行参数**

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

(需要配置多个项目，可以写配置文件)

2、来自Java：comp/env的JNDI

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

注意：6、7、8、9原则总结：

①优先加载带profile，再加载不带profile。

②由jar包外向jar包内 寻找。

●19 自动配置原理

以HttpEncodingAutoConfiguration（Http编码自动配置）为 例，说明自动配置原理

@Configuration //表示这是配置类，给容器中添加组件

@EnableConfigurationProperties(HttpEncodingProperties.**class**)

/**/ 启用指定类（HttpEncodingProperties）的ConfigurationProperities功能（见下面HttpEncodingProperties），将配置文件中对应的值和HttpEncodingProperties绑定起来；并把HttpEncodingProperties加入到IOC容器中**

@ConditionalOnWebApplication(type = ConditionalOnWebApplication.Type.***SERVLET***)

**// Spring底层的@Conditional注解，根据不同的条件，如果满足指定的条件，整个配置类里面的配置就会生效； 判断当前应用是否是Web应用，如果是Web应用，当前配置类生效**

@ConditionalOnClass(CharacterEncodingFilter.**class**)

**// 判断当前项目有没有这个类CharacterEncodingFilter；在SpringMVC中进行解决的过滤器**

@ConditionalOnProperty(prefix = "spring.http.encoding", value = "enabled", matchIfMissing = **true**)

**// 判断配置文件中是否存在某个配置spring.http.encoding.enabled；如果不存在，判断也是成立的（**matchIfMissing = **true），即使配置文件中不配置，也生效**

**public** **class** HttpEncodingAutoConfiguration {

**private** **final** HttpEncodingProperties properties; **//它已经和SpringBoot配置文件映射了**

**//只有一个有参构造器的情况下，参数的值就会从容器中拿**

**public** HttpEncodingAutoConfiguration(HttpEncodingProperties properties) {

**this**.properties = properties;

}

@Bean **//给容器中添加一个组件，这个组件中的某些值需要从properties中获取**

@ConditionalOnMissingBean(CharacterEncodingFilter.**class**)

**public** CharacterEncodingFilter characterEncodingFilter() {

CharacterEncodingFilter filter = **new** OrderedCharacterEncodingFilter();

filter.setEncoding(**this**.properties.getCharset().name());

filter.setForceRequestEncoding(**this**.properties.shouldForce(Type.***REQUEST***));

filter.setForceResponseEncoding(**this**.properties.shouldForce(Type.***RESPONSE***));

**return** filter;

}

**小结：类上面的注解根据当前不同的条件判断，决定这个类（**HttpEncodingAutoConfiguration**）是否生效？如生效，看类中的配置，此类就会给容器添加各种组件；这些总结的属性是从对应的properties类中获取中，而properties类中的每一个属性又是和配置文件（properties，yml）绑定的。**

----------------------------------------------------------------------------------------------------

@ConfigurationProperties(prefix = "spring.http.encoding")

**//从配置文件中获取指定的值和bean的属性**

**public** **class** HttpEncodingProperties {

**小结：**所有配置文件中配置的属性都是在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这个属性类

---------------------------------------------------------------------------------------------------

**总结：**

**1）SpringBoot启动会加载大量的自动配置类**

**2）要看下需要的功能有无SpringBoot默认写好的自动配置类**

**3）如存在自动配置类，看下类中配置了那些组件；只要我们用到的组件有，则不需配置**

**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。我们则可以在配置文件中指定这些属性的值**

xxxxAutoConfiguration ---- 自动配置类，为容器添加组件

xxxxProperties ----- 封装配置文件中的相关属性