# @Import注解

1. @Import你用过么？是做什么的？
2. @Import使用有几种方式？有何区别？
3. DeferredImportSelector是做什么的？他和ImportSelector有什么区别？
4. 可以介绍介绍一下spring中哪些功能是通过@Import来实现的？
5. 可以介绍一下spring中是如何解析@Import注解的么？

## @Import出现的背景

目前为止，注解的方式批量注册bean，前面2篇文章中，我们介绍了2种方式：

到目前，我们知道的批量定义bean的方式有2种：

1、@Configuration结合@Bean注解的方式

2、@CompontentScan扫描包的方式

下面我们来看几个问题。

### 问题1

如果需要注册的类是在第三方的jar中，那么我们如果想注册这些bean有2种方式：

1、通过@Bean标注方法的方式，一个个来注册

2、@CompontentScan的方式：默认的@CompontentScan是无能为力的，默认情况下只会注册@Compontent标注的类，此时只能自定义@CompontentScan中的过滤器来实现了

这2种方式都不是太好，每次有变化，调整的代码都比较多。

### 问题2

通常我们的项目中有很多子模块，可能每个模块都是独立开发的，最后通过jar的方式引进来，每个模块中都有各自的@Configuration、@Bean标注的类，或者使用@CompontentScan标注的类，**被@Configuration、@Bean、@CompontentScan标注的类，我们统称为bean配置类，配置类可以用来注册bean**，此时如果我们只想使用其中几个模块的配置类，怎么办？

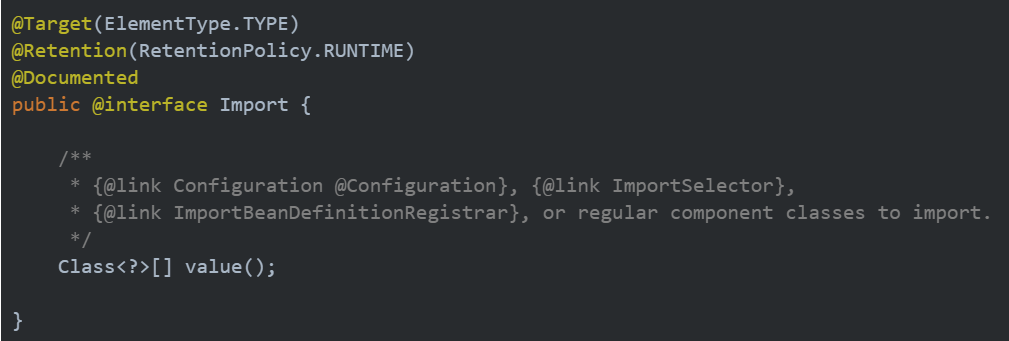
@Import可以很好的解决这2个问题，下面我们来看@Import怎么玩的。

## @Import使用

先看Spring对它的注释，总结下来作用就是和xml配置的 <import />标签作用一样，允许通过它引入@Configuration标注的类 ， 引入ImportSelector接口和ImportBeanDefinitionRegistrar接口的实现，也包括 @Component注解的普通类。

**总的来说：@Import可以用来批量导入需要注册的各种类，如普通的类、配置类，完后完成普通类和配置类中所有bean的注册。**

@Import的源码：



@Import可以使用在任何类上，通常情况下，类和注解上用的比较多。

value：一个Class数组，设置需要导入的类，可以是@Configuration标注的类，可以是ImportSelector接口或者ImportBeanDefinitionRegistrar接口类型的，或者需要导入的普通组件类。

### 使用步骤

1. 将@Import标注在类上，设置value参数
2. 将@Import标注的类作为AnnotationConfigApplicationContext构造参数创建AnnotationConfigApplicationContext对象
3. 使用AnnotationConfigApplicationContext对象

### @Import的value常见的有5种用法

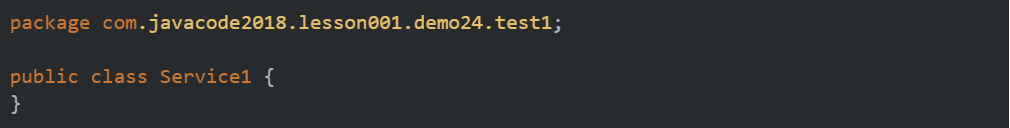
1. value为普通的类
2. value为@Configuration标注的类
3. value为@CompontentScan标注的类
4. value为ImportBeanDefinitionRegistrar接口类型
5. value为ImportSelector接口类型
6. value为DeferredImportSelector接口类型

下面我们通过案例来一个个详细介绍。

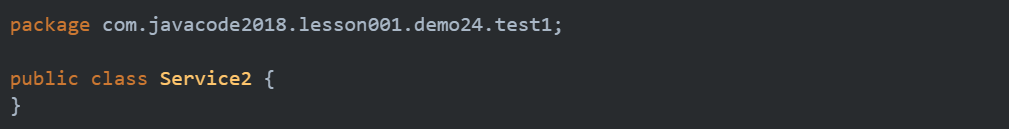
## value为普通的类

来2个类

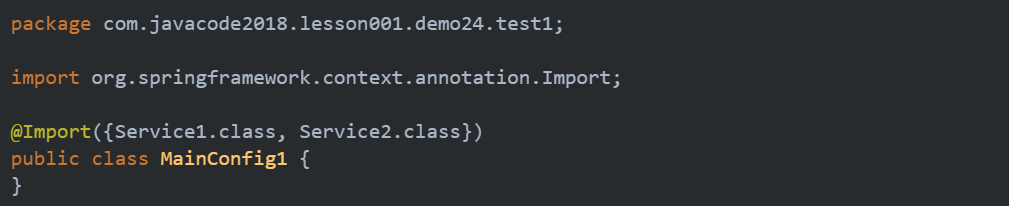
### Service1



### Service2



### 总配置类：使用@Import标注



@Import中导入了2个普通的类：Service1、Service2，这两个类会被自动注册到容器中。

### 测试用例



### 运行输出

部分输出如下：



### 结果分析

从输出中可以看出：

Service1和Service2成功注册到容器了。

通过@Import导入的2个类，bean名称为完整的类名

我们也可以指定被导入类的bean名称，使用@Compontent注解就可以了，如下：



再次运行test1输出：



### 总结一下

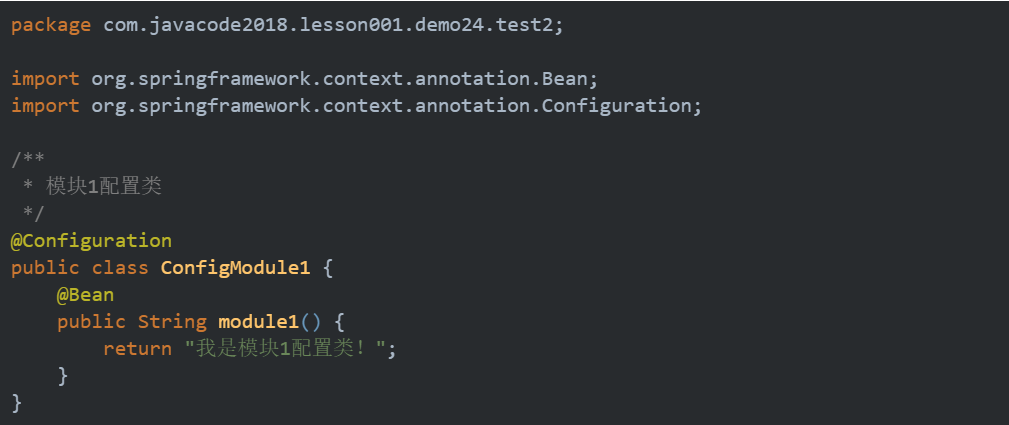
**按模块的方式进行导入，需要哪个导入哪个，不需要的时候，直接修改一下总的配置类，调整一下@Import就可以了，非常方便。**

## value为@Configuration标注的配置类

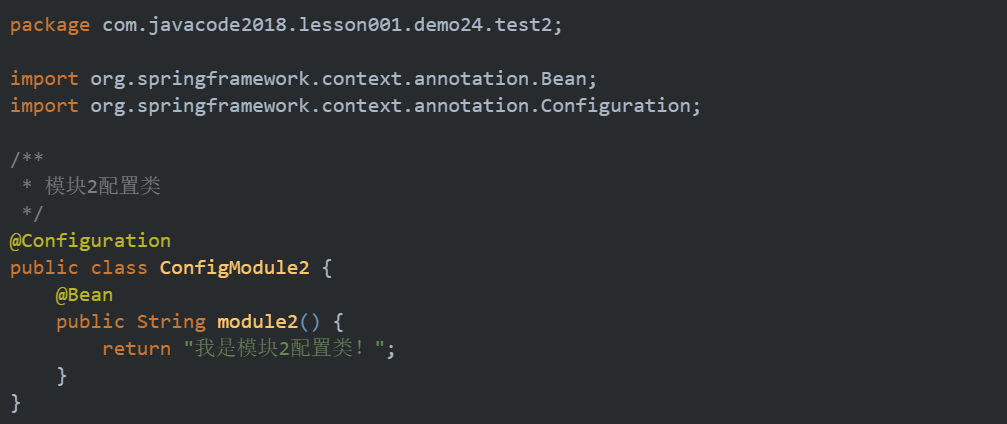
项目比较大的情况下，会按照模块独立开发，每个模块在maven中就表现为一个个的构建，然后通过坐标的方式进行引入需要的模块。

假如项目中有2个模块，2个模块都有各自的配置类，如下

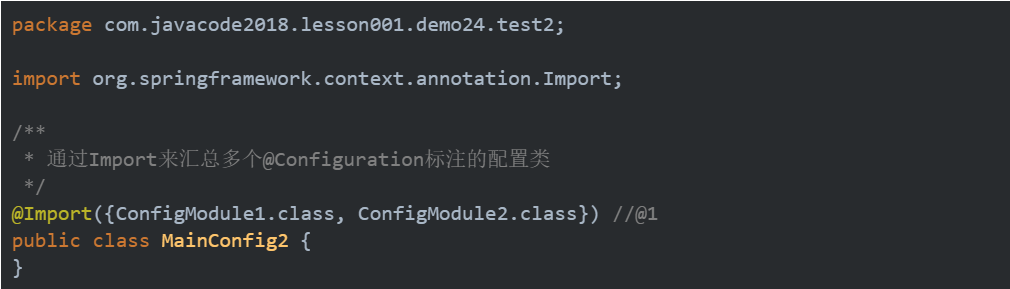
### 模块1配置类



### 模块2配置类



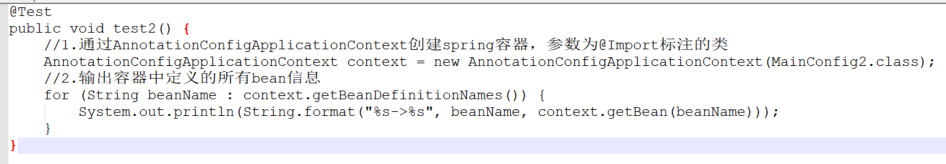
### 总配置类：通过@Import导入2个模块的配置类



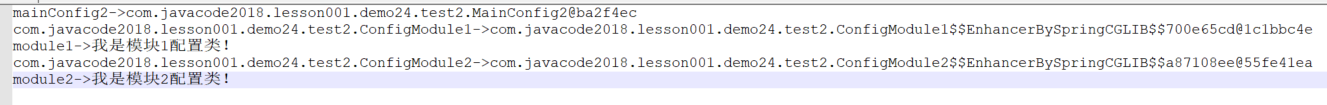
@Import导入了2个模块中的模块配置类，可以按需导入。

### 测试用例

ImportTest中新增个方法



### 运行输出



## value为@CompontentScan标注的类

项目中分多个模块，每个模块有各自独立的包，我们在每个模块所在的包中配置一个@CompontentScan类，然后通过@Import来导入需要启用的模块。

### 定义模块1

2个组件和一个组件扫描类，模块1所有类所在的包为：

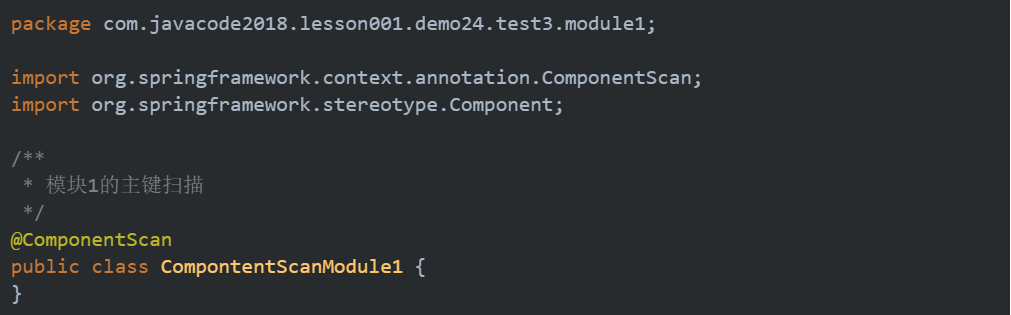


#### 组件1：Module1Service1



#### 组件扫描类：CompontentScanModule1

负责扫描当前模块中的组件



### 同样的方式定义模块2

2个组件和一个组件扫描类，模块1所有类所在的包为：



#### 组件1：Module2Service1

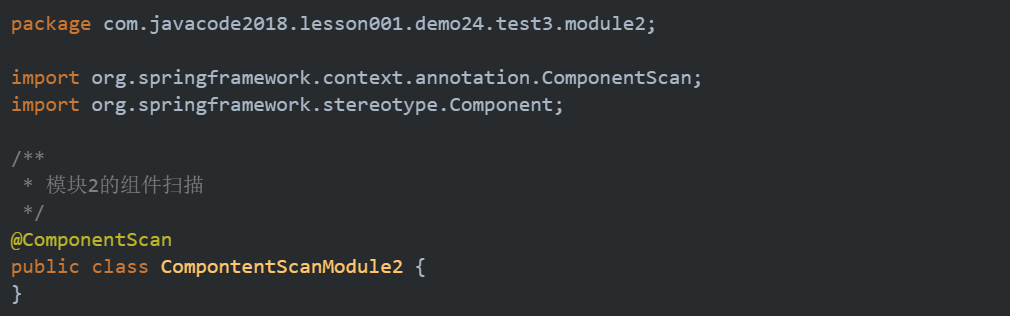


#### 组件2：Module2Service2

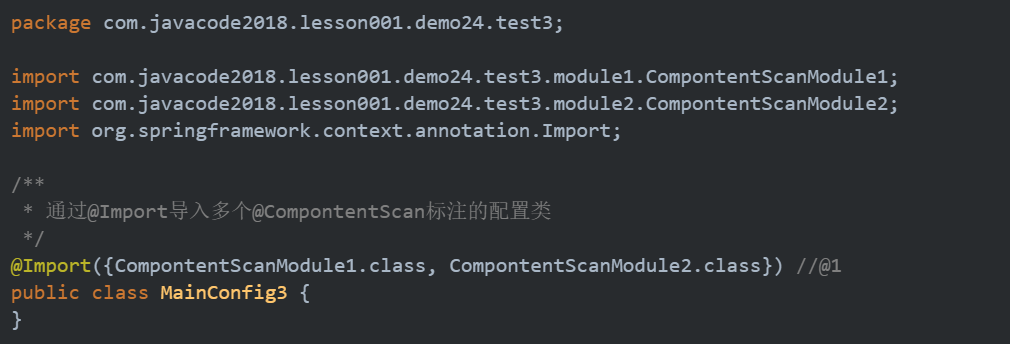


#### 组件扫描类：CompontentScanModule1

负责扫描当前模块中的组件



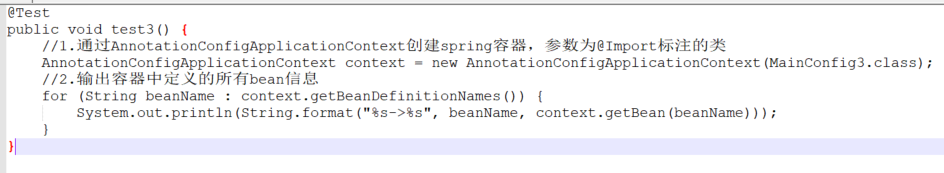
### 总配置类：通过@Import导入每个模块中的组件扫描类



@Import导入了2个模块中的组件扫描类，可以按需导入。

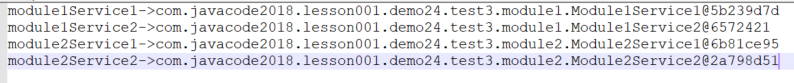
### 测试用例

ImportTest中新增个方法



### 运行输出

部分输出如下：



两个模块中通过@Compontent定义的4个bean都输出了。

如果只想注册模块1中的bean，只需要修改一下@Import，去掉CompontentScanModule2，如下：



再次运行输出：



此时模块2的bean就没有了。

## 先来了解一下相关的几个接口

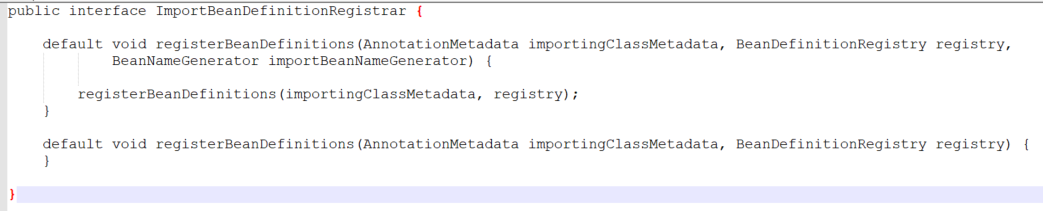
### ImportBeanDefinitionRegistrar接口

**这个接口提供了通过spring容器api的方式直接向容器中注册bean**。

接口的完整名称：



源码：



2个默认方法，都可以用来调用spring容器api来注册bean。

#### 2个方法中主要有3个参数

##### importingClassMetadata

AnnotationMetadata类型的，通过这个可以获取被@Import注解标注的类所有注解的信息。

##### registry

BeanDefinitionRegistry类型，是一个接口，内部提供了注册bean的各种方法。

##### importBeanNameGenerator

BeanNameGenerator类型，是一个接口，内部有一个方法，用来生成bean的名称。

关于BeanDefinitionRegistry和BeanNameGenerator接口在来细说一下。

### BeanDefinitionRegistry接口：bean定义注册器

bean定义注册器，提供了bean注册的各种方法，来看一下源码：

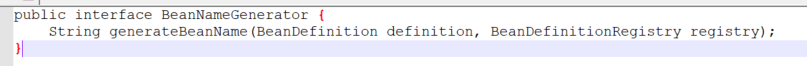


基本上所有bean工厂都实现了这个接口，让bean工厂拥有bean注册的各种能力。

上面我们用到的AnnotationConfigApplicationContext类也实现了这个接口。

### BeanNameGenerator接口：bean名称生成器

bean名称生成器，这个接口只有一个方法，用来生成bean的名称：



spring内置了3个实现

#### DefaultBeanNameGenerator

默认bean名称生成器，xml中bean未指定名称的时候，默认就会使用这个生成器，默认为：完整的类名#bean编号

#### AnnotationBeanNameGenerator

注解方式的bean名称生成器，比如通过@Component(bean名称)的方式指定bean名称，如果没有通过注解方式指定名称，默认会将完整的类名作为bean名称。

#### FullyQualifiedAnnotationBeanNameGenerator

将完整的类名作为bean的名称

### BeanDefinition接口：bean定义信息

用来表示bean定义信息的接口，我们向容器中注册bean之前，会通过xml或者其他方式定义bean的各种配置信息，bean的所有配置信息都会被转换为一个BeanDefinition对象，然后通过容器中BeanDefinitionRegistry接口中的方法，将BeanDefinition注册到spring容器中，完成bean的注册操作。

这个接口有很多实现类，有兴趣的可以去看看源码，BeanDefinition的各种用法，以后会通过专题细说。

## value为ImportBeanDefinitionRegistrar接口类型

### 用法（4个步骤）

1. 定义ImportBeanDefinitionRegistrar接口实现类，在registerBeanDefinitions方法中使用registry来注册bean

2. 使用@Import来导入步骤1中定义的类

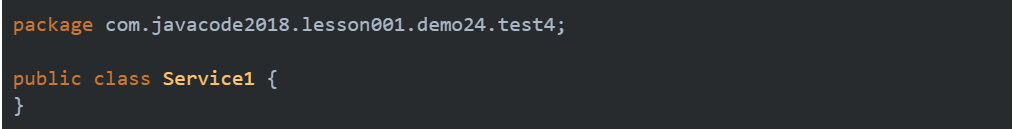
3. 使用步骤2中@Import标注的类作为AnnotationConfigApplicationContext构造参数创建spring容器

4. 使用AnnotationConfigApplicationContext操作bean

### 案例

来2个普通的类。

#### Service1



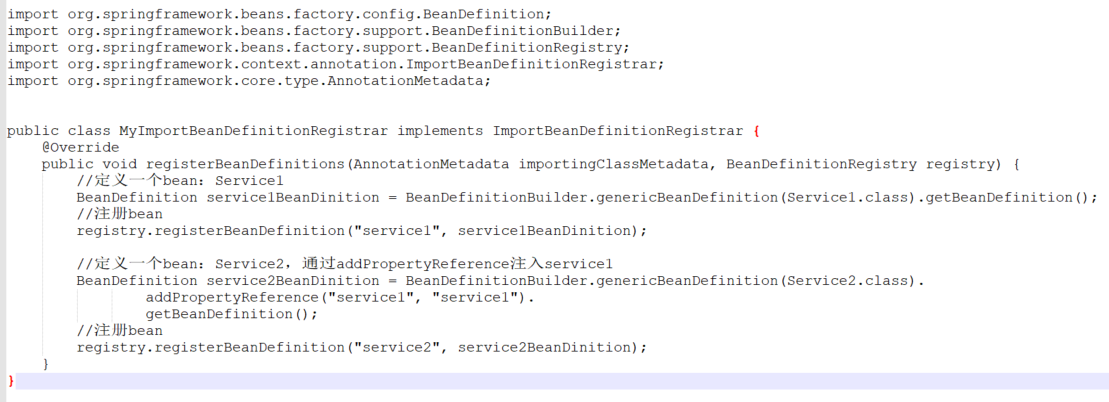
#### Service2

这个类中需要注入Service1



来个类实现ImportBeanDefinitionRegistrar接口，然后在里面实现上面2个类的注册，如下：

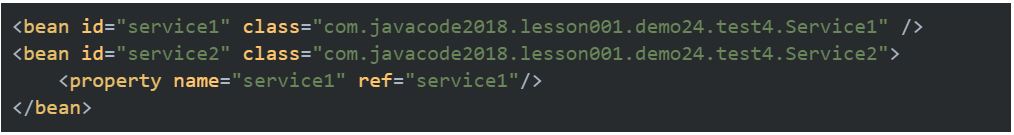
#### MyImportBeanDefinitionRegistrar



注意上面的registerBeanDefinitions方法，内部注册了2个bean，Service1和Service2。

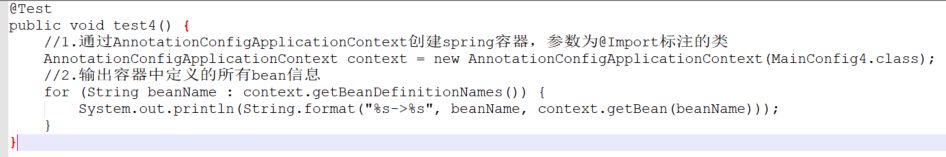
上面使用了BeanDefinitionBuilder这个类，这个是BeanDefinition的构造器，内部提供了很多静态方法方便构建BeanDefinition对象。

上面定义的2个bean，和下面xml方式效果一样：

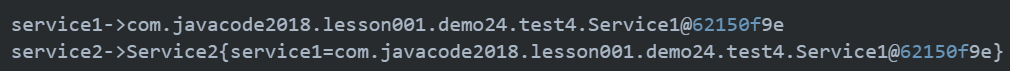


#### 来个测试用例

ImportTest中新增个方法



#### 运行输出



## value为ImportSelector接口类型

先来看一下ImportSelector接口

### ImportSelector接口

导入选择器，看一下源码：

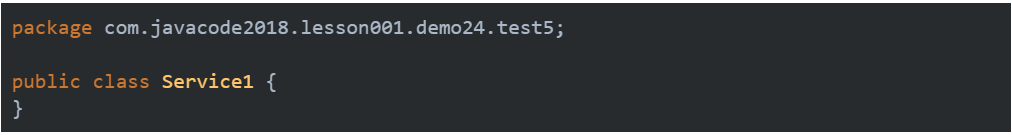


### 用法（4个步骤）

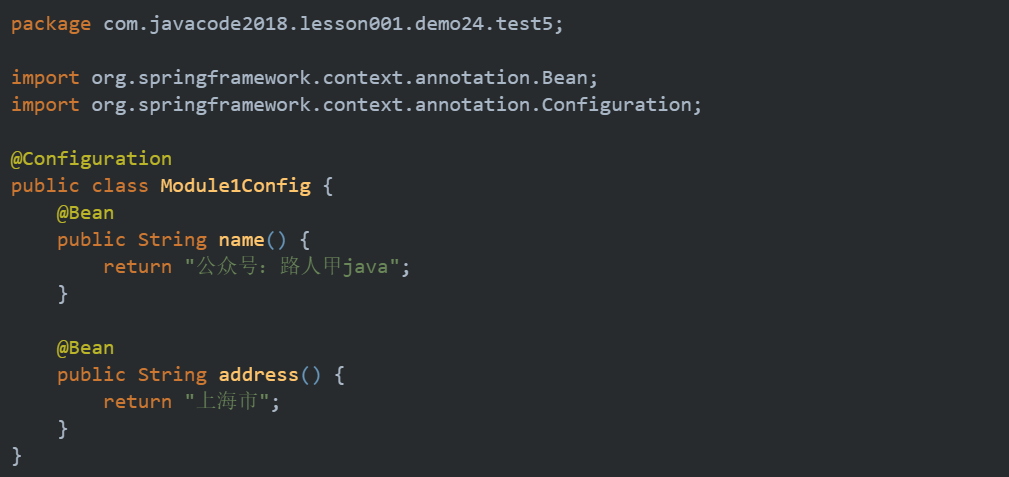
1. 定义ImportSelector接口实现类，在selectImports返回需要导入的类的名称数组  
   2. 使用@Import来导入步骤1中定义的类  
   3. 使用步骤2中@Import标注的类作为AnnotationConfigApplicationContext构造参数创建spring容器  
   4. 使用AnnotationConfigApplicationContext操作bean

### 案例

#### 来个普通类：Service1

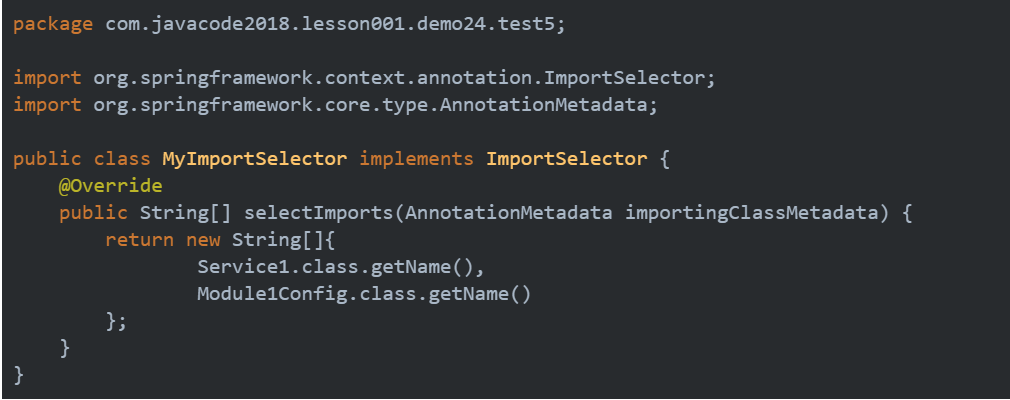


#### 来个@Configuration标注的配置类：Module1Config



上面定义了两个string类型的bean：name和address

#### 下面自定义一个ImportSelector，然后返回上面2个类的名称

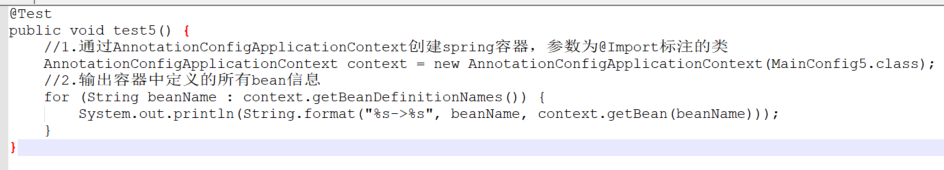


#### 来个@Import标注的类，导入MyImportSelector



#### 新增测试用例

ImportTest中新增个方法



#### 运行输出

部分输出如下：



输出中可以看到Service1以及Module1Config中定义的2个bean都有了。

## 来一个牛逼的案例

### 需求

凡是类名中包含service的，调用他们内部任何方法，我们希望调用之后能够输出这些方法的耗时。

### 实现分析

之前我们讲过代理， 此处我们就可以通过代理来实现，bean实例创建的过程中，我们可以给这些bean生成一个代理，在代理中统计方法的耗时，这里面有2点：

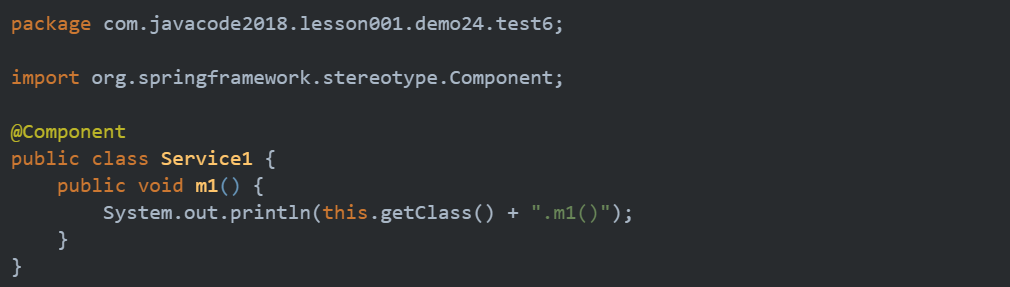
1、创建一个代理类，通过代理来间接访问需要统计耗时的bean对象；

2、拦截bean的创建，给bean实例生成代理

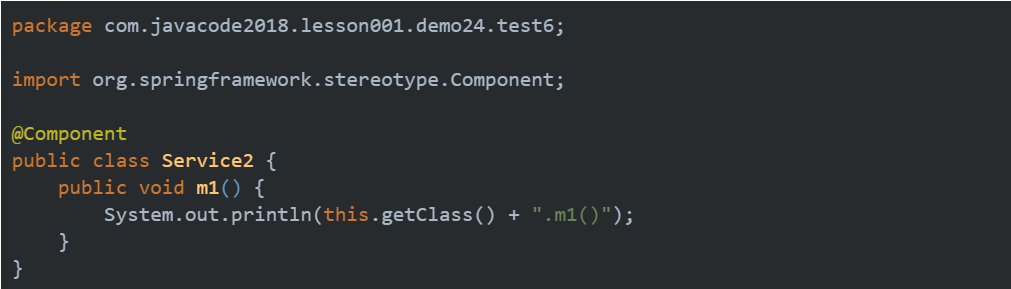
### 具体实现

#### 先来两个Service类

##### Service1

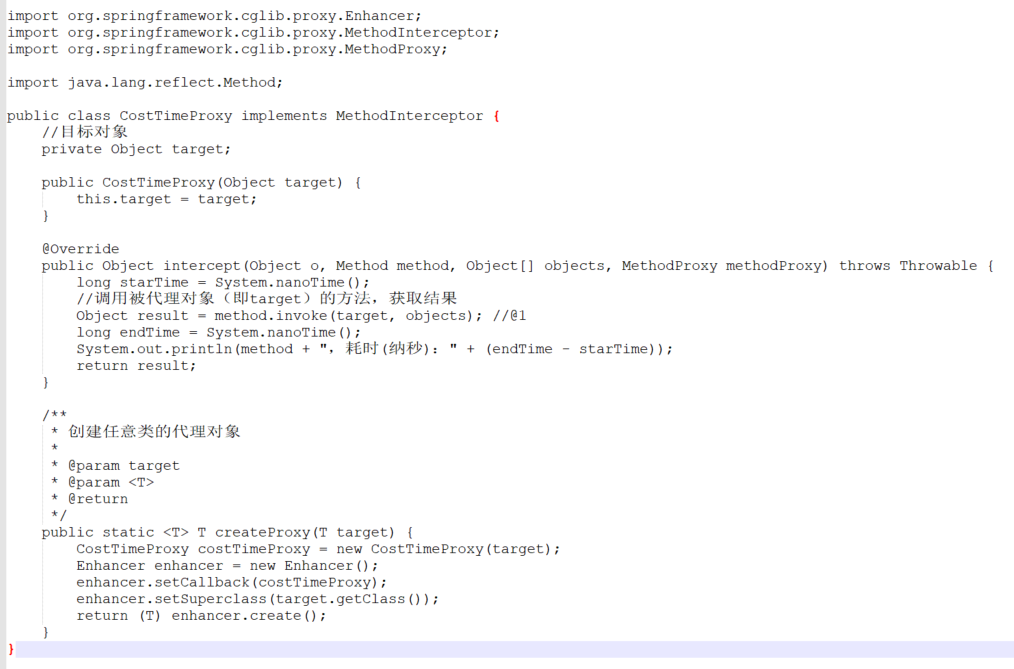


##### Service2



#### 创建统计耗时的代理类

下面我们使用cglib来实现一个代理类，如下：



createProxy方法可以用来给某个对象生成代理对象

需要了解cglib的可以看：

<https://mp.weixin.qq.com/s/xvidIGCL4mIYfGni4pq9wQ>

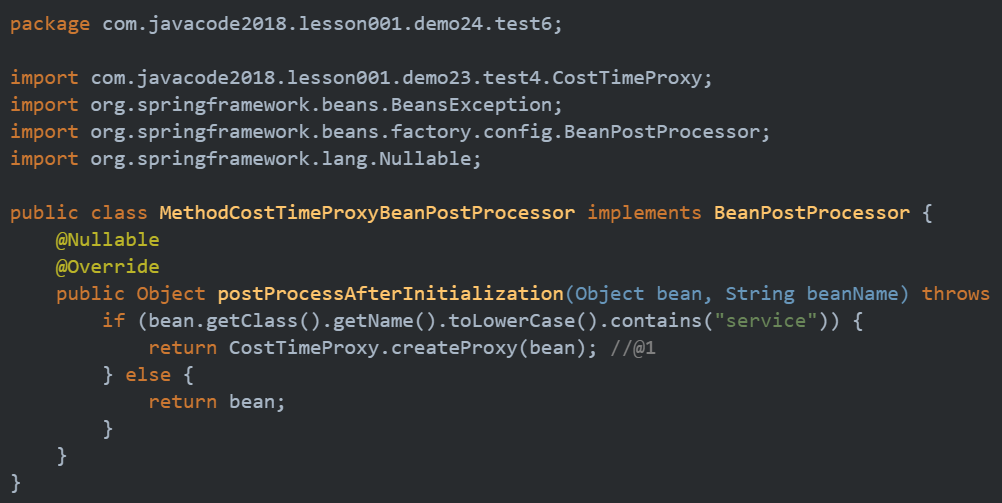
#### 拦截bean实例的创建，返回代理对象

这里我们需要用到spring中的一个接口：



这个接口是bean处理器，内部有2个方法，分别在bean初始化前后会进行调用，以后讲声明周期的时候还会细说的，这里你只需要知道bean初始化之后会调用postProcessAfterInitialization方法就行，这个方法中我们会给bean创建一个代理对象。

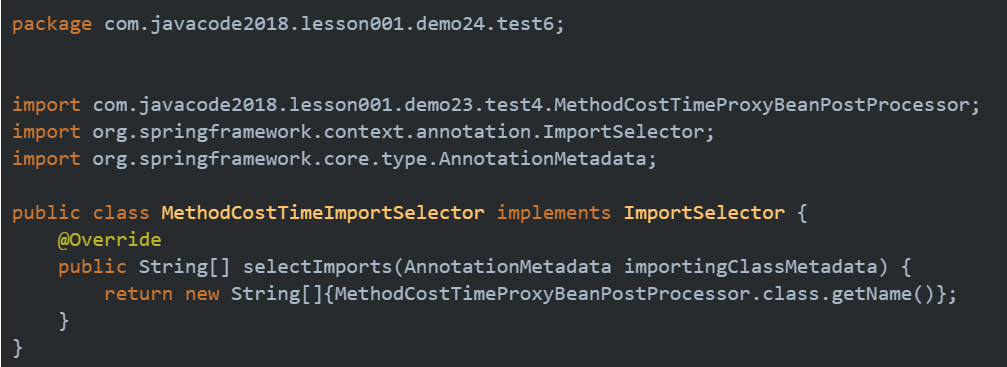
下面我们创建一个BeanPostProcessor实现类：



@1：使用上面创建代理类来给当前bean对象创建一个代理

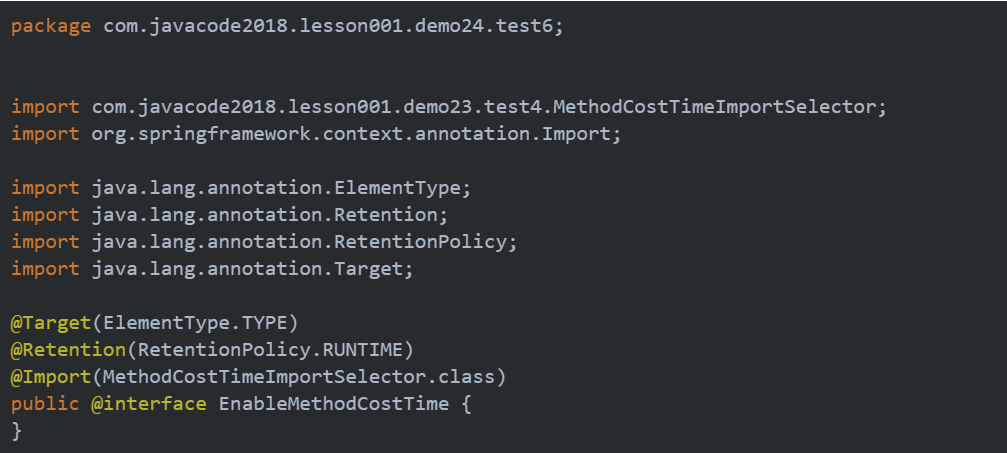
需要将MethodCostTimeProxyBeanPostProcessor注册到容器中才会起作用，下面我们通过@Import结合ImportSelector的方式来导入这个类，将其注册到容器中。

#### MethodCostTimeImportSelector

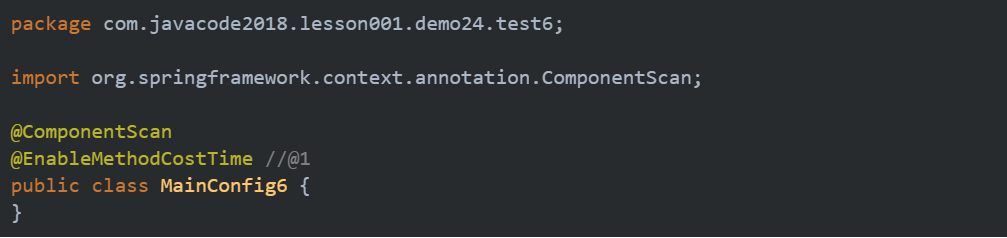


#### 来一个@Import来导入MethodCostTimeImportSelector

下面我们使用注解的方式，在注解上使用@Import，如下：



#### 来一个总的配置类

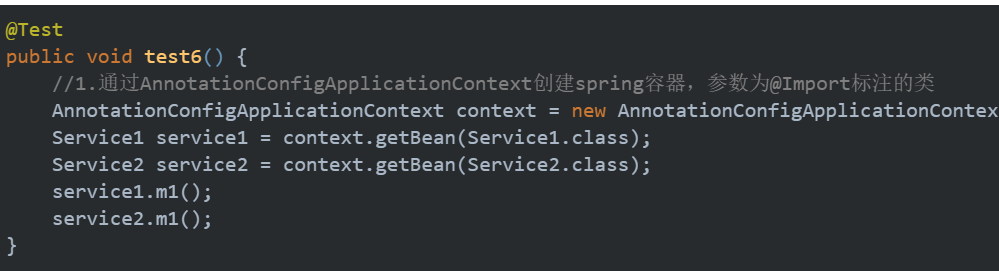


上面使用了@CompontentScan注解，此时会将Servce1和Service2这两个类注册到容器中。

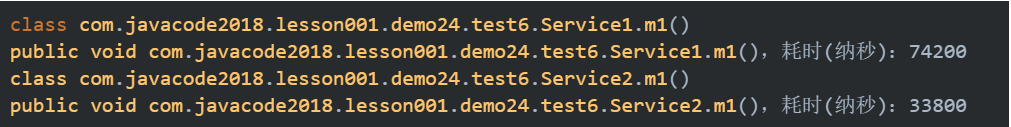
@1：此处使用了@EnableMethodCostTime注解，而@EnableMethodCostTime注解上使用了@Import(MethodCostTimeImportSelector.class)，此时MethodCostTimeImportSelector类中的MethodCostTimeProxyBeanPostProcessor会被注册到容器，会拦截bean的创建，创建耗时代理对象。

#### 来个测试用例

ImportTest中新增个方法



上面会调用service1和service2的方法



**太牛逼了，需求实现了。**

**如果我们不想开启方法耗时统计，只需要将MainConfig6上的@EnableMethodCostTime去掉就可以了，用起来是不是特别爽。**

**spring中有很多类似的注解，以@EnableXXX开头的注解，基本上都是通过上面这种方式实现的，如：**



继续向下看，还有一个更牛逼的接口DeferredImportSelector。

## DeferredImportSelector接口

**先给你透露一下，springboot中的核心功能@EnableAutoConfiguration就是靠DeferredImportSelector来实现的。**

DeferredImportSelector是ImportSelector的子接口，既然是ImportSelector的子接口，所以也可以通过@Import进行导入，这个接口和ImportSelector不同地方有两点：

1. 延迟导入
2. 指定导入的类的处理顺序

### 延迟导入

比如@Import的value包含了多个普通类、多个@Configuration标注的配置类、多个ImportSelector接口的实现类，多个ImportBeanDefinitionRegistrar接口的实现类，还有DeferredImportSelector接口实现类，此时spring处理这些被导入的类的时候，**会将DeferredImportSelector类型的放在最后处理，会先处理其他被导入的类，其他类会按照value所在的前后顺序进行处理**。

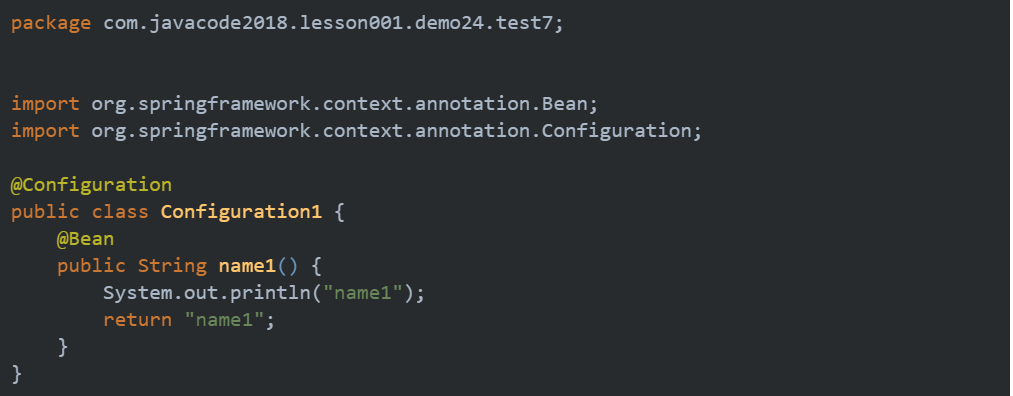
那么我们是可以做很多事情的，比如我们可以在DeferredImportSelector导入的类中判断一下容器中是否已经注册了某个bean，如果没有注册过，那么再来注册。

以后我们会讲到另外一个注解@Conditional，这个注解可以按条件来注册bean，比如可以判断某个bean不存在的时候才进行注册，某个类存在的时候才进行注册等等各种条件判断，通过@Conditional来结合DeferredImportSelector可以做很多事情。

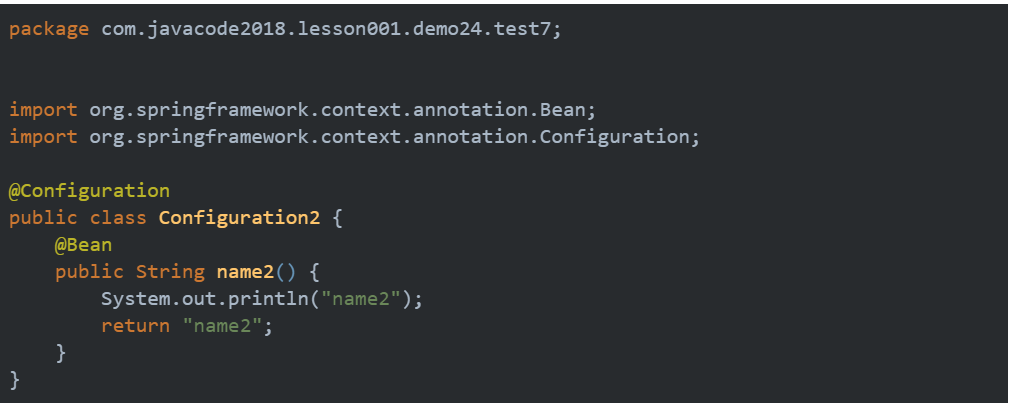
### 来个延迟导入的案例

来3个配置类，每个配置类中都通过@Bean定一个string类型的bean，内部输出一句文字。

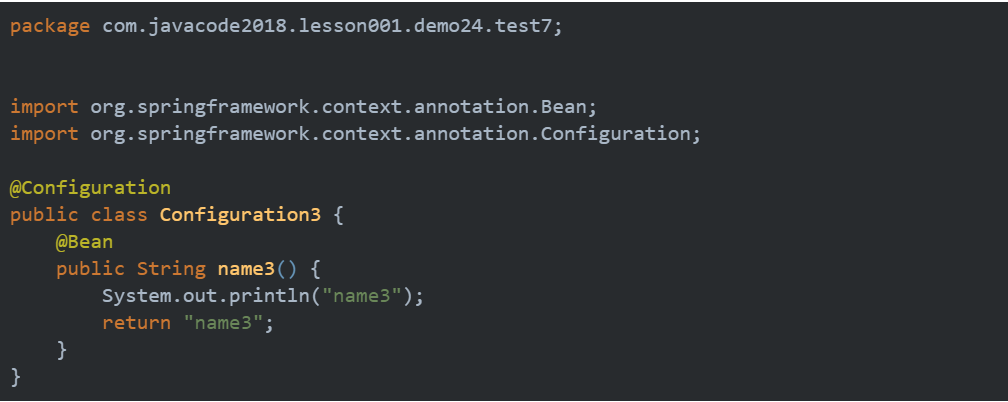
#### Configuration1



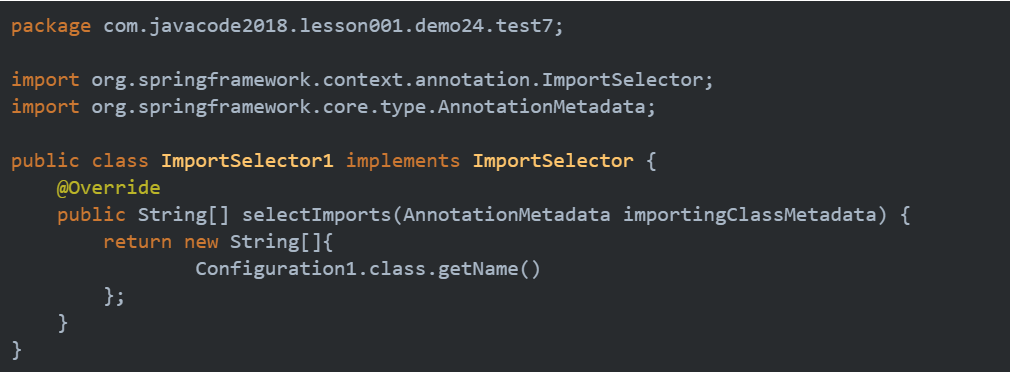
#### Configuration2



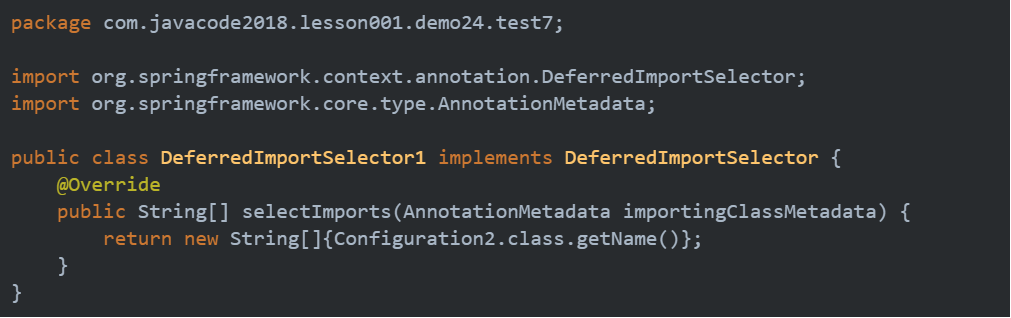
#### Configuration3



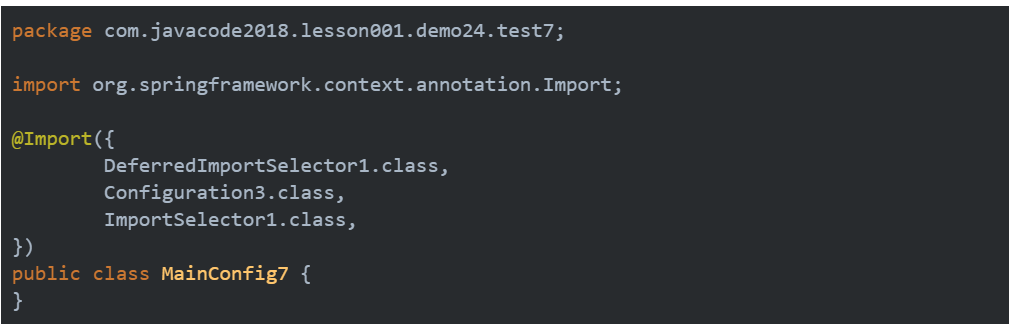
#### 来一个ImportSelector实现类，导入Configuration1



#### 来一个DeferredImportSelector实现类，导入Configuration2



#### 来一个总的配置类



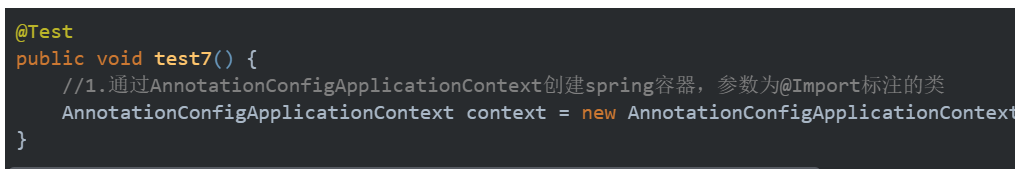
注意上面的@Import中被导入类的顺序：

DeferredImportSelector1->Configuration3->ImportSelector1

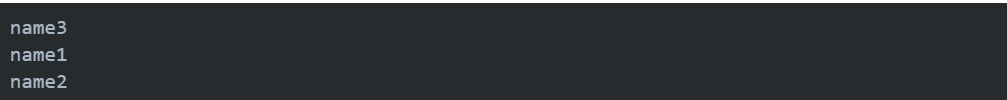
下面来个测试用例，看一下3个配置文件中@Bean标注的方法被执行的先后顺序。

#### 测试用例

ImportTest中新增个方法



#### 运行输出

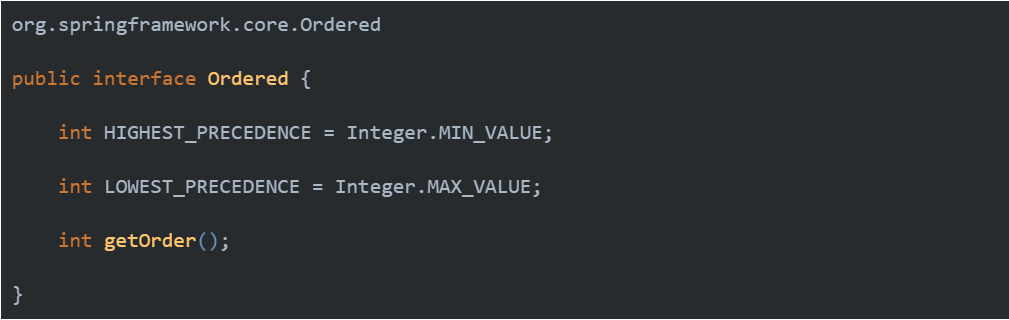


**输出的结果结合一下@Import中被导入的3个类的顺序，可以看出DeferredImportSelector1是被最后处理的，其他2个是按照value中所在的先后顺序处理的。**

### 指定导入的类的处理顺序

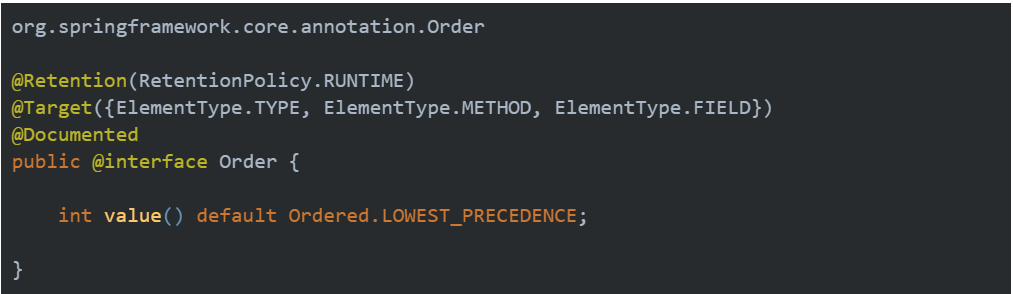
当@Import中有多个DeferredImportSelector接口的实现类时候，可以指定他们的顺序，指定顺序常见2种方式

#### 实现Ordered接口的方式



value的值越小，优先级越高。

#### 实现Order注解的方式



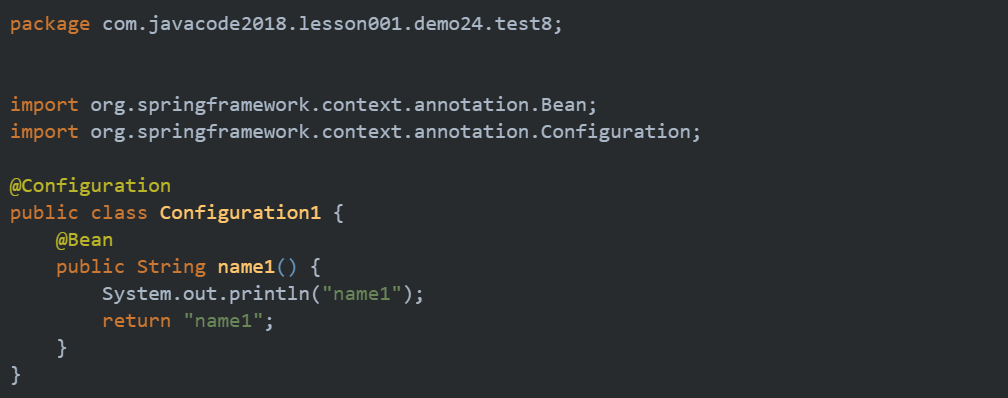
value的值越小，优先级越高。

下面我们来个案例感受一下。

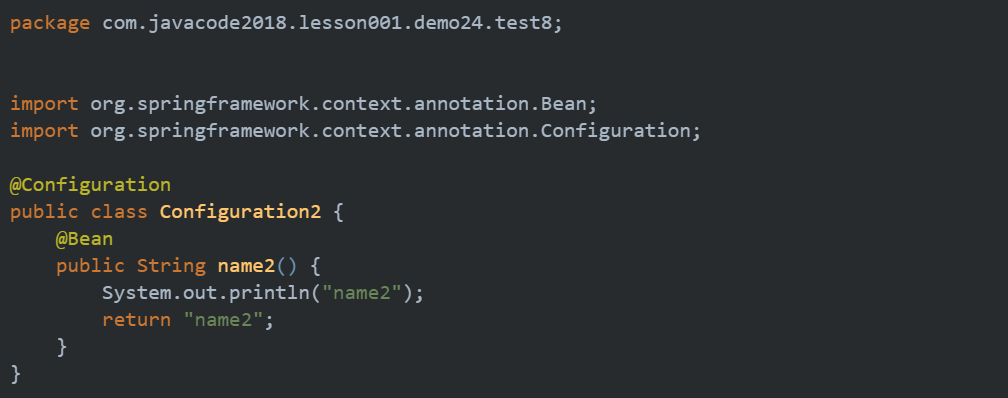
### 来个指定导入类处理顺序的案例

来2个配置类，内部都有一个@Bean标注的方法，用来注册一个bean，方法内部输出一行文字

#### Configuration1



#### Configuration2



来2个DeferredImportSelector实现类，分别来导入上面2个配置文件，顺便通过Ordered接口指定一下顺序

#### DeferredImportSelector1



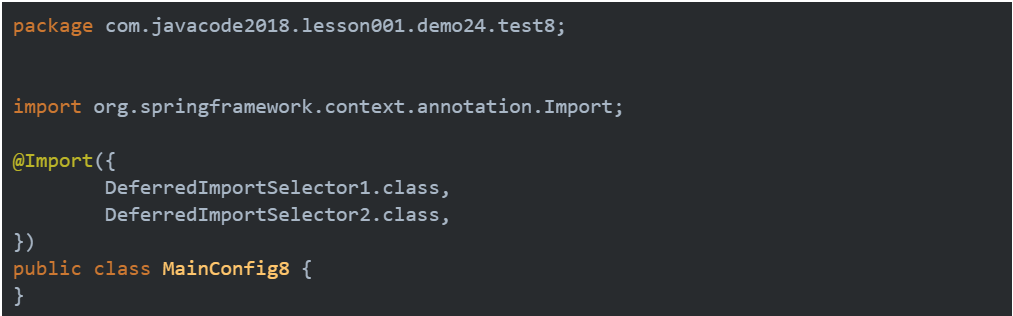
#### DeferredImportSelector2



**DeferredImportSelector1的order为2，DeferredImportSelector2的order为1，order值越小优先级越高。**

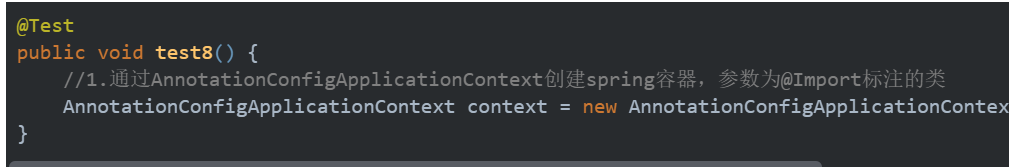
来个总的配置类，引入上面两个ImportSelector

#### MainConfig8



#### 测试用例

ImportTest中新增个方法



#### 运行输出



**结果配合order的值，按照order从小到大来处理，可以看出DeferredImportSelector2先被处理的。**

## Spring中这块的源码

**@Import注解是被下面这个类处理的**

org.springframework.context.annotation.ConfigurationClassPostProcessor

前面介绍的@Configuration、@Bean、@CompontentScan、@CompontentScans都是被这个类处理的，这个类是高手必经之路，建议花点时间研究研究。

## 总结

1. **@Import可以用来批量导入任何普通的组件、配置类，将这些类中定义的所有bean注册到容器中**
2. **@Import常见的5种用法需要掌握**
3. **掌握ImportSelector、ImportBeanDefinitionRegistrar、DeferredImportSelector的用法**
4. **DeferredImportSelector接口可以实现延迟导入、按序导入的功能**
5. **spring中很多以@Enable开头的都是使用@Import集合ImportSelector方式实现的**
6. **BeanDefinitionRegistry接口：bean定义注册器，这个需要掌握常见的方法**