Spring注解驱动开发第3讲——使用@ComponentScan自动扫描组件并指定扫描规则

写在前面

在实际项目中，我们更多的是使用Spring的包扫描功能对项目中的包进行扫描，凡是在指定的包或其子包中的类上标注了@Repository 、@Service、@Controller、@Component注解的类都会被扫描到，并将这个类注入到Spring容器中。

Spring包扫描功能可以使用XML配置文件进行配置，也可以直接使用@ ComponentScan注解 进行设置，使用@ComponentScan注解进行设置比使用XML配置文件来配置要简单的多。

使用XML文件配置包扫描

我们可以在Spring的XML配置文件中配置包的扫描，在配置包扫描时，需要在Spring的XML配置文件中的beans节点中引入context标签，如下所示。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context-4.2.xsd">
```
AI写代码xml

接下来，我们就可以在XML配置文件中定义要扫描的包了，如下所示。

```xml
<!-- 包扫描：只要是标注了我们熟悉的@Controller、@Service、@Repository、@Component这四个注解中的任何一个的组件，它就会被自动扫描，并加进容器中 -->
<context:component-scan base-package="com.meimeixia"></context:component-scan>
```
AI写代码xml

整个beans.xml配置文件中的内容如下所示。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <!-- 包扫描：只要是标注了我们熟悉的@Controller、@Service、@Repository、@Component这四个注解中的任何一个的组件，它就会被自动扫描，并加进容器中 -->
    <context:component-scan base-package="com.meimeixia"></context:component-scan>

    <!-- 注册组件 -->
    <bean id="person" class="com.meimeixia.bean.Person">
        <property name="age" value="18"></property>
        <property name="name" value="liayun"></property>
    </bean>

</beans>
```
AI写代码xml

这样配置以后，只要在com.meimeixia包下，或者com.meimeixia的子包下标注了@Repository、@Service、@Controller、@Component注解的类都会被扫描到，并自动注入到Spring容器中。

此时，我们分别创建BookDao、BookService以及BookController这三个类，并在这三个类中分别添加@Repository、@Service、@Controller注解，如下所示。

- BookDao

```java
package com.meimeixia.dao;

import org.springframework.stereotype.Repository;

// 名字默认是类名首字母小写
```

```java
@Repository
public class BookDao {

}
```
AI写代码java运行

- BookService

```java
package com.meimeixia.service;

import org.springframework.stereotype.Service;

@Service
public class BookService {

}
```
AI写代码java运行

- BookController

```java
package com.meimeixia.controller;

import org.springframework.stereotype.Controller;

@Controller
public class BookController {

}
```
AI写代码java运行

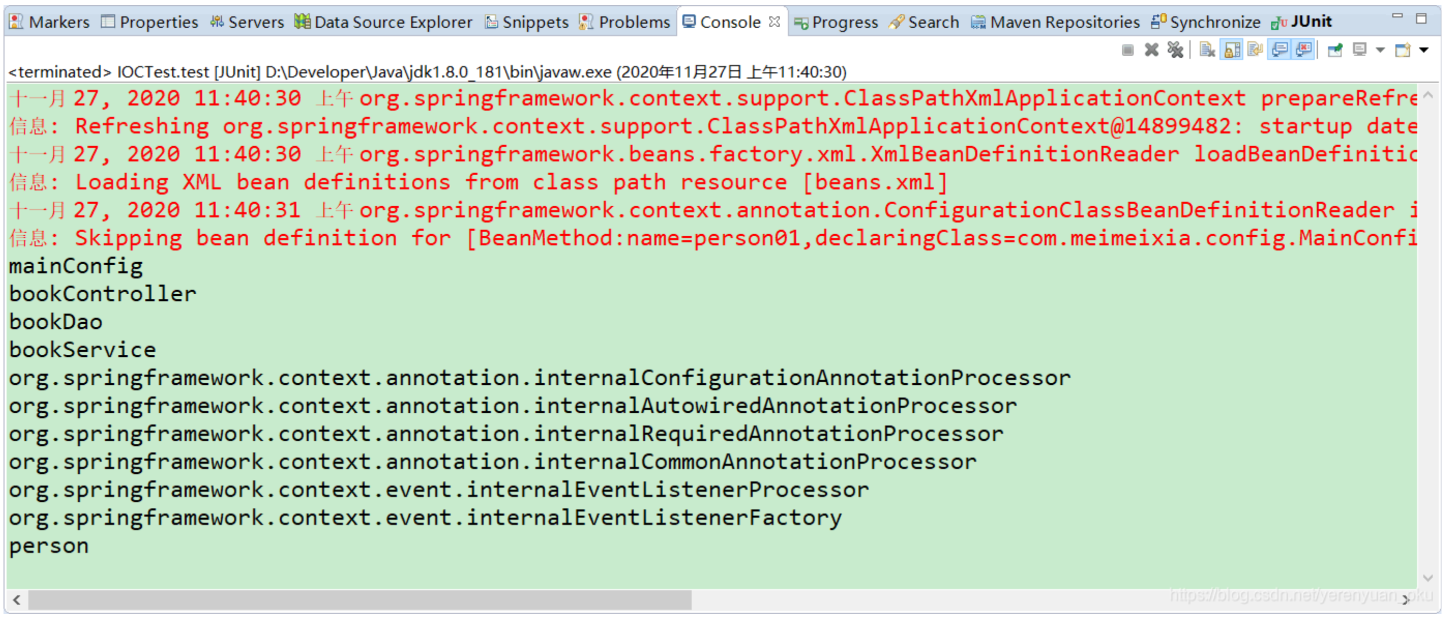
接下来，我们在工程的src/test/java目录下新建一个 单元测试 类来进行测试，例如IOCTest。由于我在这儿使用的是junit来进行测试，因此还须在pom文件中添加对junit的依赖，如下所示。

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```
AI写代码xml

添加完依赖之后，我们就可以在IOCTest测试类中编写如下一个方法来进行测试了，即看一看IOC容器中现在有哪些bean。

```java
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.config.MainConfig;

public class IOCTest {

    @SuppressWarnings("resource")
    @Test
    public void test() {
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");
        // 我们现在就来看一下IOC容器中有哪些bean，即容器中所有bean定义的名字
        String[] definitionNames = applicationContext.getBeanDefinitionNames();
        for (String name : definitionNames) {
            System.out.println(name);
        }
    }

}
```
AI写代码java运行

运行测试用例，输出的结果信息如下图所示。



```
<terminated> IOCTest.test [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 上午11:40:30)
十一月 27, 2020 11:40:30 上午 org.springframework.context.support.ClassPathXmlApplicationContext prepareRefre
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@14899482: startup date
十一月 27, 2020 11:40:30 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitic
信息: Loading XML bean definitions from class path resource [beans.xml]
十一月 27, 2020 11:40:31 上午 org.springframework.context.annotation.ConfigurationClassBeanDefinitionReader i
信息: Skipping bean definition for [BeanMethod:name=person01,declaringClass=com.meimeixia.config.MainConfi
mainConfig
bookController
bookDao
bookService
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
person
```

可以看到，除了输出我们自己创建的bean的名称之外，也输出了Spring内部使用的一些重要的bean的名称。

接下来，我们使用注解来完成这些功能。

使用注解配置包扫描

使用@ComponentScan注解之前我们先将beans.xml配置文件中的下述配置注释掉。

```xml
<context:component-scan base-package="com.meimeixia"></context:component-scan>
```
AI写代码xml

注释掉之后，我们就可以使用@ComponentScan注解来配置包扫描了。使用@ComponentScan注解配置包扫描非常非常easy！只须在我们的MainConfig类上添加@ComponentScan注解，并将扫描的包指定为com.meimeixia即可，如下所示。

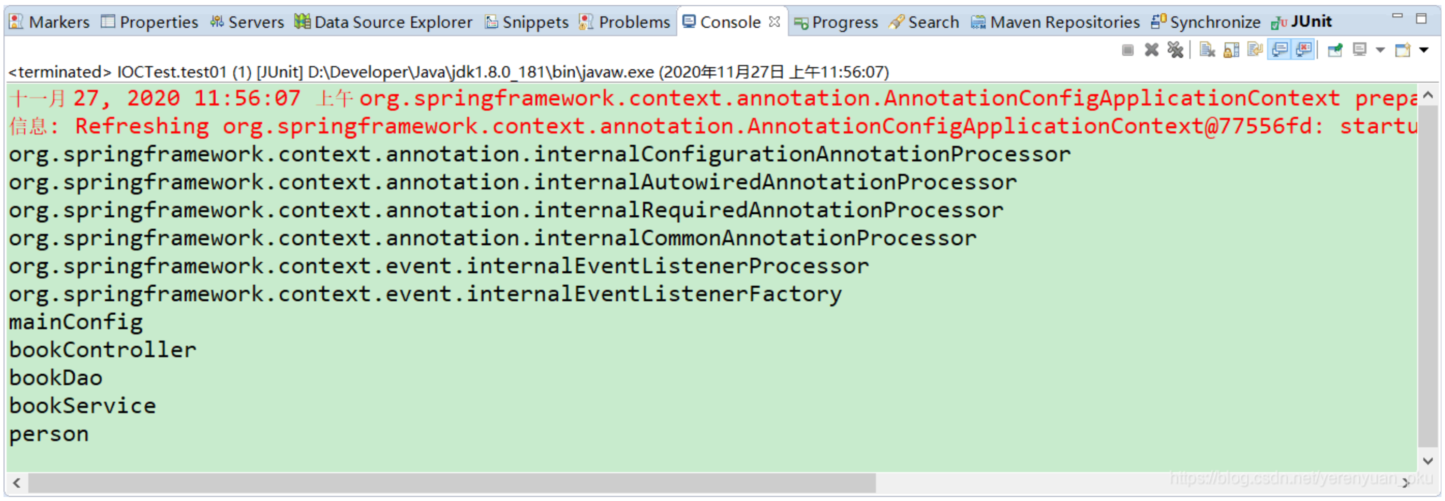
```java
package com.meimeixia.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.meimeixia.bean.Person;
/**
 * 以前配置文件的方式被替换成了配置类，即配置类==配置文件
 * @author liayun
 *
 */
// 这个配置类也是一个组件
@ComponentScan(value="com.meimeixia") // value指定要扫描的包
@Configuration // 告诉Spring这是一个配置类
public class MainConfig {

    // @Bean注解是给IOC容器中注册一个bean，类型自然就是返回值的类型，id默认是用方法名作为id
    @Bean("person")
    public Person person01() {
        return new Person("liayun", 20);
    }

}
```
AI写代码java运行

没错，就是这么简单，只需要在类上添加 `@ComponentScan(value="com.meimeixia")` 这样一个注解即可。

然后，我们在IOCTest类中新增如下一个test01()方法，以便进行测试。

```java
@SuppressWarnings("resource")
@Test
public void test01() {
    AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig.class);
    // 我们现在就来看一下IOC容器中有哪些bean，即容器中所有bean定义的名字

    String[] definitionNames = applicationContext.getBeanDefinitionNames();
    for (String name : definitionNames) {
        System.out.println(name);
    }
}
```
AI写代码java运行

运行以上test01()方法，输出的结果信息如下图所示。



```
<terminated> IOCTest.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 上午11:56:07)
十一月 27, 2020 11:56:07 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepa
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig
bookController
bookDao
bookService
person
```

可以看到使用@ComponentScan注解同样输出了容器中bean的名称。

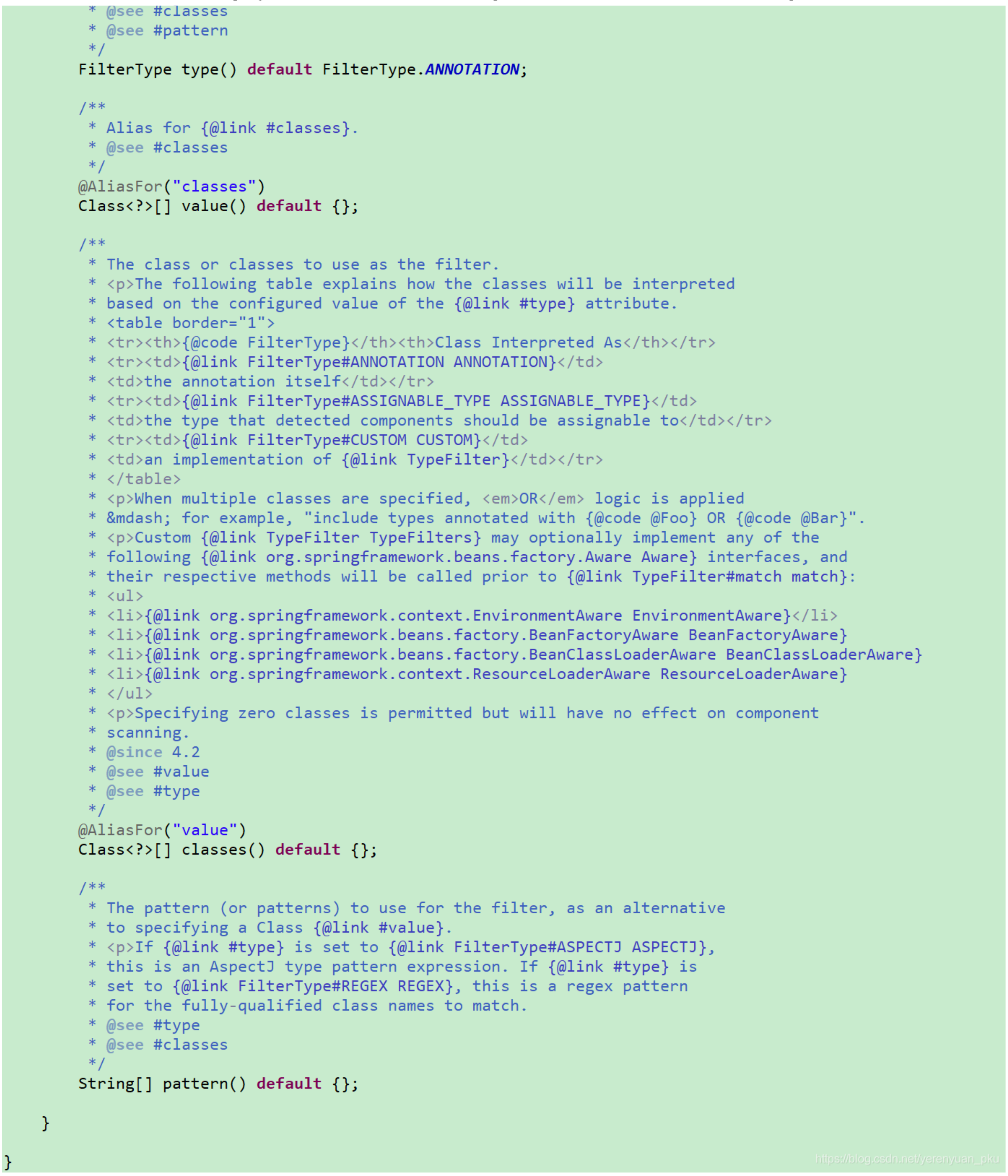
既然使用XML配置文件和注解的方式都能够将相应的类注入到Spring容器当中，那我们是使用XML配置文件还是使用注解呢？我更倾向于使用注解，如果你确实喜欢使用XML配置文件来进行配置，也可以啊，哈哈，个人喜好嘛！好了，我们继续。

关于@ComponentScan注解

我们点开ComponentScan注解类并查看其源码，如下图所示。

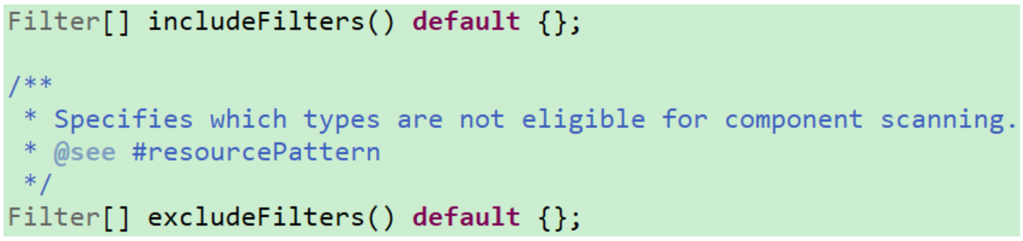
```java
 * Copyright 2002-2016 the original author or authors.

package org.springframework.context.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Repeatable;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.springframework.beans.factory.support.BeanNameGenerator;
import org.springframework.core.annotation.AliasFor;
import org.springframework.core.type.filter.TypeFilter;

/**
 * Configures component scanning directives for use with @{@link Configuration} classes.
 * Provides support parallel with Spring XML's {@code <context:component-scan>} element.
 *
 * <p>Either {@link #basePackageClasses} or {@link #basePackages} (or its alias
 * {@link #value}) may be specified to define specific packages to scan. If specific
 * packages are not defined, scanning will occur from the package of the
 * class that declares this annotation.
 *
 * <p>Note that the {@code <context:component-scan>} element has an
 * {@code annotation-config} attribute; however, this annotation does not. This is because
 * in almost all cases when using {@code @ComponentScan}, default annotation config
 * processing (e.g. processing {@code @Autowired} and friends) is assumed. Furthermore,
 * when using {@link AnnotationConfigApplicationContext}, annotation config processors are
 * always registered, meaning that any attempt to disable them at the
 * {@code @ComponentScan} level would be ignored.
 *
 * <p>See {@link Configuration @Configuration}'s Javadoc for usage examples.
 *
 * @author Chris Beams
 * @author Juergen Hoeller
 * @author Sam Brannen
 * @since 3.1
 * @see Configuration
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
@Repeatable(ComponentScans.class)
public @interface ComponentScan {

    /**
     * Alias for {@link #basePackages}.
     * <p>Allows for more concise annotation declarations if no other attributes
     * are needed &mdash; for example, {@code @ComponentScan("org.my.pkg")}
     * instead of {@code @ComponentScan(basePackages = "org.my.pkg")}.
     */
    @AliasFor("basePackages")
    String[] value() default {};

    /**
     * Base packages to scan for annotated components.
     * <p>{@link #value} is an alias for (and mutually exclusive with) this
     * attribute.
     * <p>Use {@link #basePackageClasses} for a type-safe alternative to
     * String-based package names.
     */
    @AliasFor("value")
    String[] basePackages() default {};

    /**
     * Type-safe alternative to {@link #basePackages} for specifying the packages
     * to scan for annotated components. The package of each class specified will be scanned.
     * <p>Consider creating a special no-op marker class or interface in each package
     * that serves no purpose other than being referenced by this attribute.
     */
    Class<?>[] basePackageClasses() default {};

    /**
     * The {@link BeanNameGenerator} class to be used for naming detected components
```

```
	 * within the Spring container.
	 * <p>The default value of the {@link BeanNameGenerator} interface itself indicates
	 * that the scanner used to process this {@code @ComponentScan} annotation should
	 * use its inherited bean name generator, e.g. the default
	 * {@link AnnotationBeanNameGenerator} or any custom instance supplied to the
	 * application context at bootstrap time.
	 * @see AnnotationConfigApplicationContext#setBeanNameGenerator(BeanNameGenerator)
	 */
	Class<? extends BeanNameGenerator> nameGenerator() default BeanNameGenerator.class;

	/**
	 * The {@link ScopeMetadataResolver} to be used for resolving the scope of detected components.
	 */
	Class<? extends ScopeMetadataResolver> scopeResolver() default AnnotationScopeMetadataResolver.class;

	/**
	 * Indicates whether proxies should be generated for detected components, which may be
	 * necessary when using scopes in a proxy-style fashion.
	 * <p>The default is defer to the default behavior of the component scanner used to
	 * execute the actual scan.
	 * <p>Note that setting this attribute overrides any value set for {@link #scopeResolver}.
	 * @see ClassPathBeanDefinitionScanner#setScopedProxyMode(ScopedProxyMode)
	 */
	ScopedProxyMode scopedProxy() default ScopedProxyMode.DEFAULT;

	/**
	 * Controls the class files eligible for component detection.
	 * <p>Consider use of {@link #includeFilters} and {@link #excludeFilters}
	 * for a more flexible approach.
	 */
	String resourcePattern() default ClassPathScanningCandidateComponentProvider.DEFAULT_RESOURCE_PATTERN;

	/**
	 * Indicates whether automatic detection of classes annotated with {@code @Component}
	 * {@code @Repository}, {@code @Service}, or {@code @Controller} should be enabled.
	 */
	boolean useDefaultFilters() default true;

	/**
	 * Specifies which types are eligible for component scanning.
	 * <p>Further narrows the set of candidate components from everything in {@link #basePackages}
	 * to everything in the base packages that matches the given filter or filters.
	 * <p>Note that these filters will be applied in addition to the default filters, if specified.
	 * Any type under the specified base packages which matches a given filter will be included,
	 * even if it does not match the default filters (i.e. is not annotated with {@code @Component}).
	 * @see #resourcePattern()
	 * @see #useDefaultFilters()
	 */
	Filter[] includeFilters() default {};

	/**
	 * Specifies which types are not eligible for component scanning.
	 * @see #resourcePattern
	 */
	Filter[] excludeFilters() default {};

	/**
	 * Specify whether scanned beans should be registered for lazy initialization.
	 * <p>Default is {@code false}; switch this to {@code true} when desired.
	 * @since 4.1
	 */
	boolean lazyInit() default false;


	/**
	 * Declares the type filter to be used as an {@linkplain ComponentScan#includeFilters
	 * include filter} or {@linkplain ComponentScan#excludeFilters exclude filter}.
	 */
	@Retention(RetentionPolicy.RUNTIME)
	@Target({})
	@interface Filter {

		/**
		 * The type of filter to use.
		 * <p>Default is {@link FilterType#ANNOTATION}.
```



```
         * @see #classes
         * @see #pattern
         */
        FilterType type() default FilterType.ANNOTATION;

        /**
         * Alias for {@link #classes}.
         * @see #classes
         */
        @AliasFor("classes")
        Class<?>[] value() default {};

        /**
         * The class or classes to use as the filter.
         * <p>The following table explains how the classes will be interpreted
         * based on the configured value of the {@link #type} attribute.
         * <table border="1">
         * <tr><th>{@code FilterType}</th><th>Class Interpreted As</th></tr>
         * <tr><td>{@link FilterType#ANNOTATION ANNOTATION}</td>
         * <td>the annotation itself</td></tr>
         * <tr><td>{@link FilterType#ASSIGNABLE_TYPE ASSIGNABLE_TYPE}</td>
         * <td>the type that detected components should be assignable to</td></tr>
         * <tr><td>{@link FilterType#CUSTOM CUSTOM}</td>
         * <td>an implementation of {@link TypeFilter}</td></tr>
         * </table>
         * <p>When multiple classes are specified, <em>OR</em> logic is applied
         * &mdash; for example, "include types annotated with {@code @Foo} OR {@code @Bar}".
         * <p>Custom {@link TypeFilter TypeFilters} may optionally implement any of the
         * following {@link org.springframework.beans.factory.Aware Aware} interfaces, and
         * their respective methods will be called prior to {@link TypeFilter#match match}:
         * <ul>
         * <li>{@link org.springframework.context.EnvironmentAware EnvironmentAware}</li>
         * <li>{@link org.springframework.beans.factory.BeanFactoryAware BeanFactoryAware}
         * <li>{@link org.springframework.beans.factory.BeanClassLoaderAware BeanClassLoaderAware}
         * <li>{@link org.springframework.context.ResourceLoaderAware ResourceLoaderAware}
         * </ul>
         * <p>Specifying zero classes is permitted but will have no effect on component
         * scanning.
         * @since 4.2
         * @see #value
         * @see #type
         */
        @AliasFor("value")
        Class<?>[] classes() default {};

        /**
         * The pattern (or patterns) to use for the filter, as an alternative
         * to specifying a Class {@link #value}.
         * <p>If {@link #type} is set to {@link FilterType#ASPECTJ ASPECTJ},
         * this is an AspectJ type pattern expression. If {@link #type} is
         * set to {@link FilterType#REGEX REGEX}, this is a regex pattern
         * for the fully-qualified class names to match.
         * @see #type
         * @see #classes
         */
        String[] pattern() default {};

    }

}
```

这里，我们着重来看ComponentScan类中的如下两个方法。



```
Filter[] includeFilters() default {};

/**
 * Specifies which types are not eligible for component scanning.
 * @see #resourcePattern
 */
Filter[] excludeFilters() default {};
```

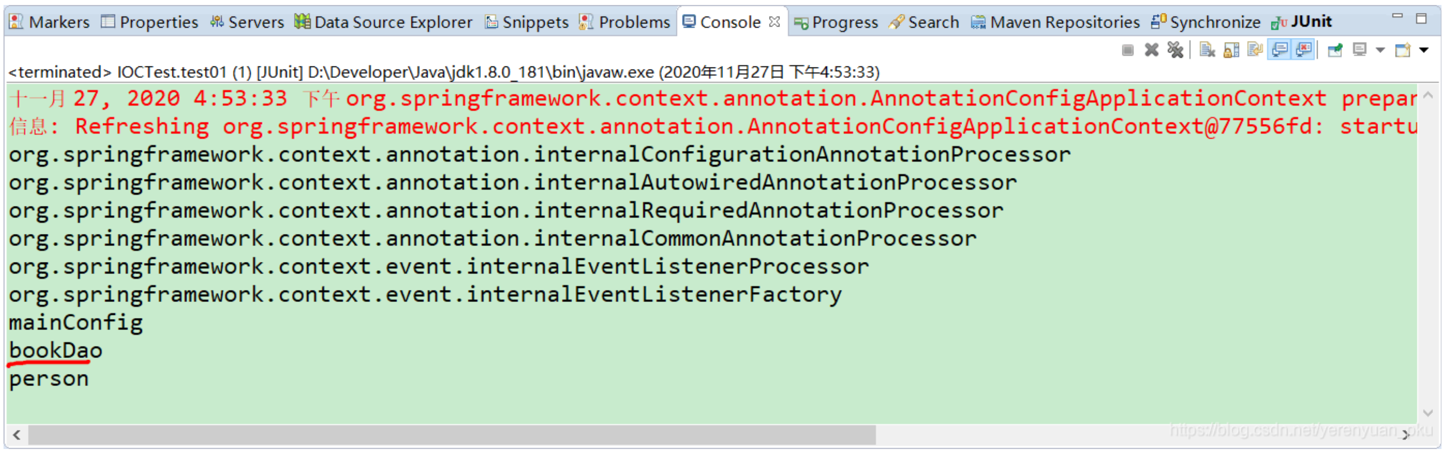
includeFilters()方法指定Spring扫描的时候按照什么规则只需要包含哪些组件，而excludeFilters()方法指定Spring扫描的时候按照什么规则排除哪些组件。两个方法的返回值都是Filter[]数组，在ComponentScan注解类的内部存在Filter注解类，大家可以看下上面的代码。

扫描时排除注解标注的类

现在有这样一个需求，除了@Controller和@Service标注的组件之外，IOC容器中剩下的组件我都要，即相当于是我要排除@Controller和@Service这俩注解标注的组件。要想达到这样一个目的，我们可以在MainConfig类上通过@ComponentScan注解的excludeFilters()方法实现。例如，我们在MainConfig类上添加了如下的注解。

```java
@ComponentScan(value="com.meimeixia", excludeFilters={
        /*
         * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
         * classes：除了@Controller和@Service标注的组件之外，IOC容器中剩下的组件我都要，即相当于是我要排除@Controller和@Service这俩注解标注的组件。
         */
        @Filter(type=FilterType.ANNOTATION, classes={Controller.class, Service.class})
}) // value指定要扫描的包
```
AI写代码java运行

这样，我们就使得Spring在扫描包的时候排除了使用@Controller和@Service这俩注解标注的类。你要是不信的话，那么可以运行一下IOCTest类中的test01()方法，看看输出的结果信息到底是什么。



```
<terminated> IOCTest.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 下午4:53:33)
十一月 27, 2020 4:53:33 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepar
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig
bookDao
person
```

从上图中可以清楚地看到，输出的结果信息中不再输出bookController和bookService了，这已然说明了Spring在进行包扫描时，忽略了@Controller和@Service这俩注解标注的类。

扫描时只包含注解标注的类

我们也可以使用ComponentScan注解类中的includeFilters()方法来指定Spring在进行包扫描时，只包含哪些注解标注的类。

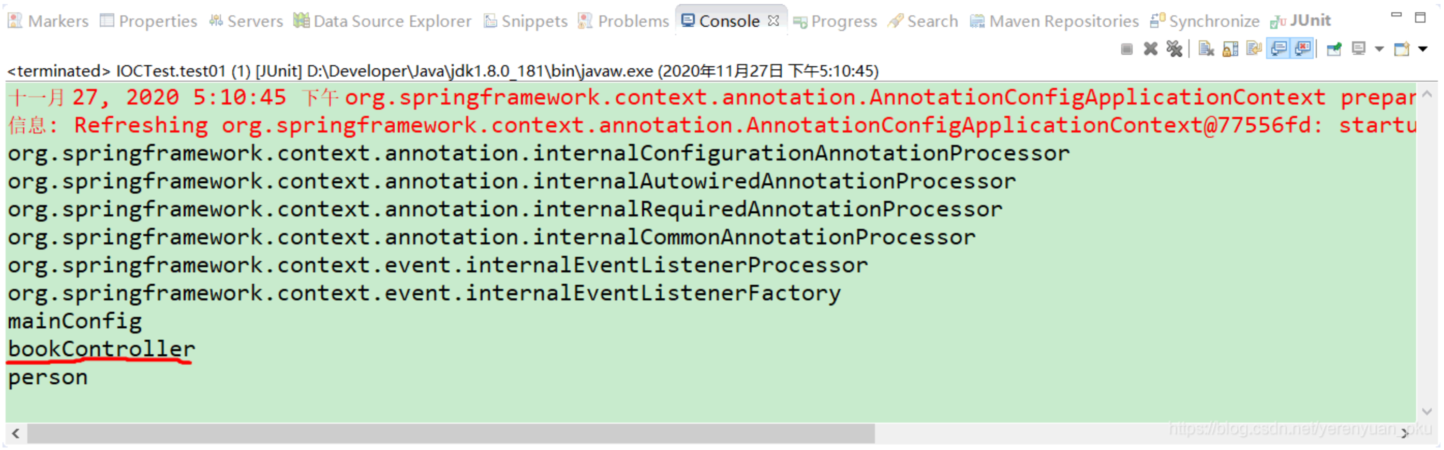
这里需要注意的是，当我们使用includeFilters()方法来指定只包含哪些注解标注的类时，需要禁用掉默认的过滤规则。 还记得我们以前在XML配置文件中配置这个只包含的时候，应该怎么做吗？我们需要在XML配置文件中先配置好 `use-default-filters="false"`，也就是禁用掉默认的过滤规则，因为默认的过滤规则就是扫描所有的，只有我们禁用掉默认的过滤规则之后，只包含才能生效。

```xml
<context:component-scan base-package="com.meimeixia" use-default-filters="false"></context:component-scan>
```
AI写代码xml

现在有这样一个需求，我们需要Spring在扫描时，只包含@Controller注解标注的类。要想达到这样一个目的，我们该怎么做呢？可以在MainConfig类上添加@ComponentScan注解，设置只包含@Controller注解标注的类，并禁用掉默认的过滤规则，如下所示。

```java
@ComponentScan(value="com.meimeixia", includeFilters={
        /*
         * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
         * classes：我们需要Spring在扫描时，只包含@Controller注解标注的类
         */
        @Filter(type=FilterType.ANNOTATION, classes={Controller.class})
}, useDefaultFilters=false) // value指定要扫描的包
```
AI写代码java运行

此时，我们再次运行IOCTest类中的test01()方法，输出的结果信息如下图所示。


Markers Properties Servers Data Source Explorer Snippets Problems Console Progress Search Maven Repositories Synchronize JUnit

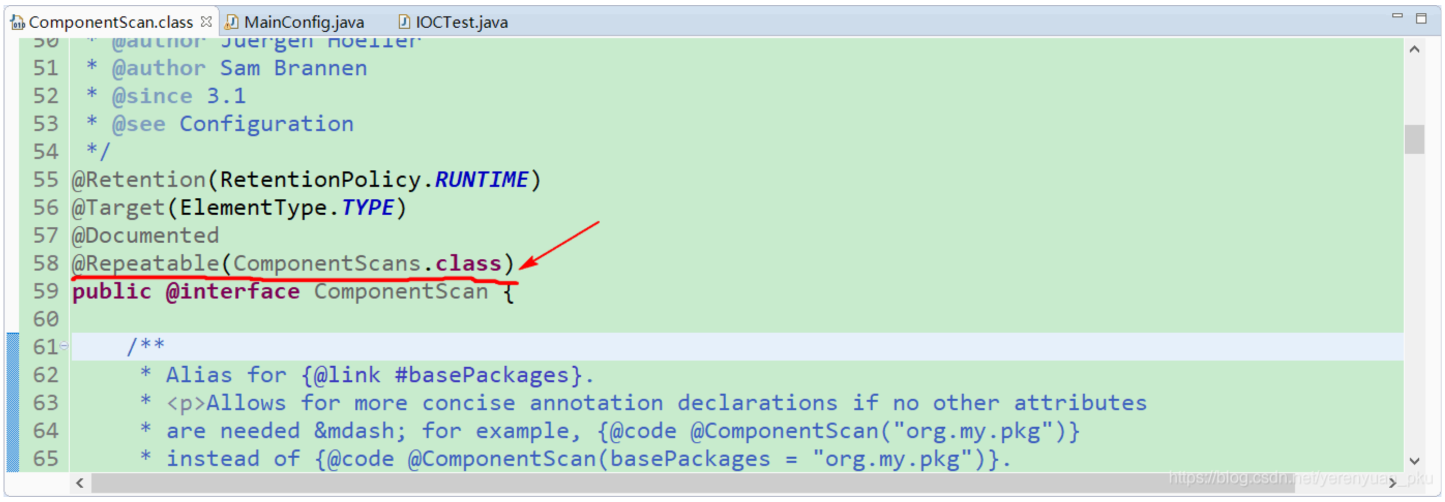
```
<terminated> IOCTest.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 下午5:10:45)
十一月 27, 2020 5:10:45 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepar
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig
bookController
person
```

可以看到，在输出的结果中，只包含了@Controller注解标注的组件名称，并没有输出@Service和@Repository这俩注解标注的组件名称。

温馨提示：在使用includeFilters()方法来指定只包含哪些注解标注的类时，结果信息中会一同输出Spring内部的组件名称。

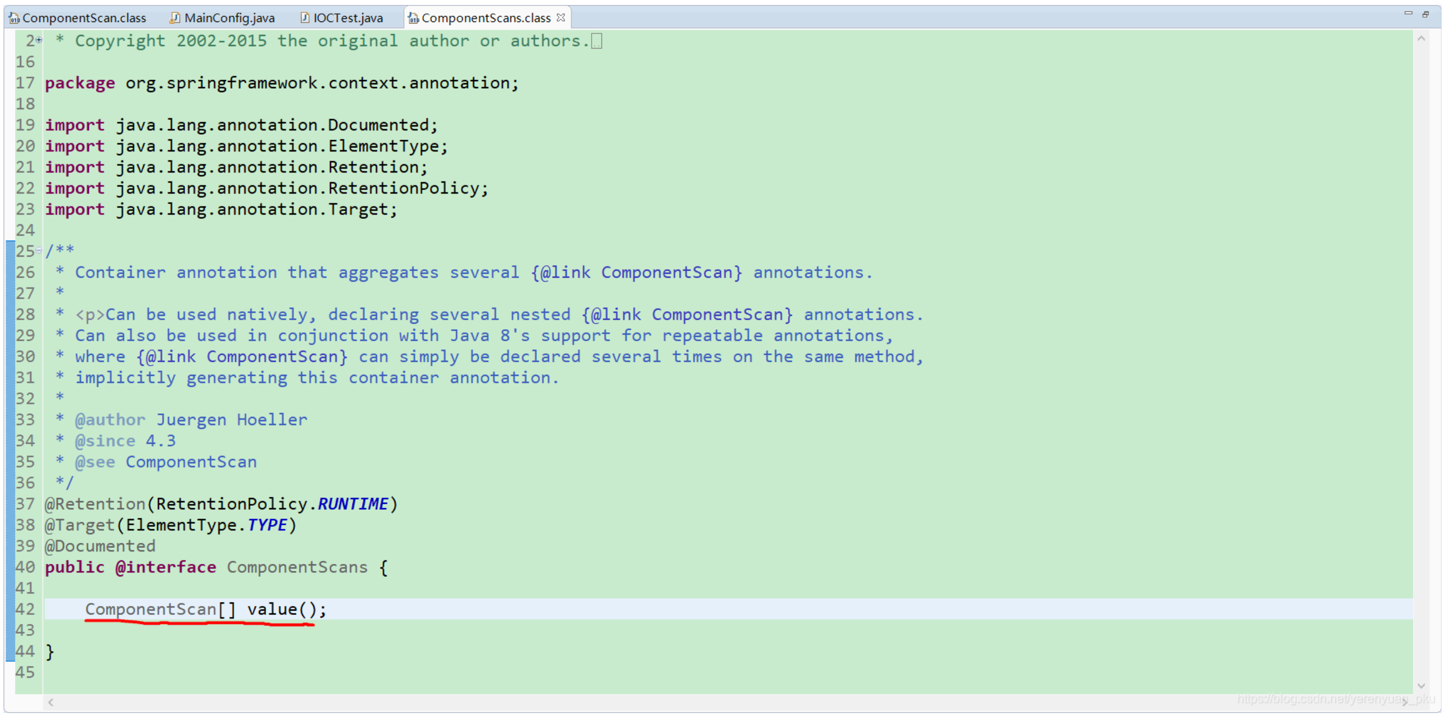
重复注解

不知道小伙伴们有没有注意到ComponentScan注解类上有一个如下所示的注解。



```
 * @author Juergen Hoeller
 * @author Sam Brannen
 * @since 3.1
 * @see Configuration
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
@Repeatable(ComponentScans.class)
public @interface ComponentScan {

    /**
     * Alias for {@link #basePackages}.
     * <p>Allows for more concise annotation declarations if no other attributes
     * are needed &mdash; for example, {@code @ComponentScan("org.my.pkg")}
     * instead of {@code @ComponentScan(basePackages = "org.my.pkg")}.
```

我们先来看看@ComponentScans注解是个啥，如下图所示。


ComponentScan.class MainConfig.java IOCTest.java ComponentScans.class

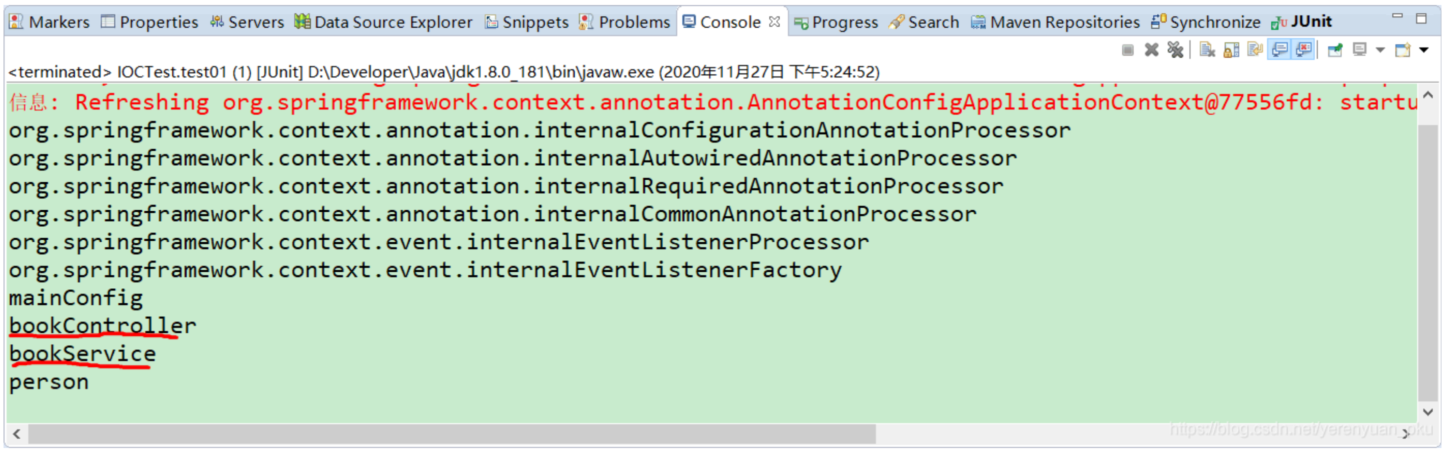
```
 * Copyright 2002-2015 the original author or authors.

package org.springframework.context.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Container annotation that aggregates several {@link ComponentScan} annotations.
 *
 * <p>Can be used natively, declaring several nested {@link ComponentScan} annotations.
 * Can also be used in conjunction with Java 8's support for repeatable annotations,
 * where {@link ComponentScan} can simply be declared several times on the same method,
 * implicitly generating this container annotation.
 *
 * @author Juergen Hoeller
 * @since 4.3
 * @see ComponentScan
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
public @interface ComponentScans {

    ComponentScan[] value();

}
```

可以看到，在ComponentScans注解类的内部只声明了一个返回ComponentScan[]数组的value()方法，说到这里，大家是不是就明白了，没错，这在Java 8中是一个重复注解。

如果你用的是Java 8，那么@ComponentScan注解就是一个重复注解，也就是说我们可以在一个类上重复使用这个注解，如下所示。

```java
@ComponentScan(value="com.meimeixia", includeFilters={
        /*
         * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
         * classes：我们需要Spring在扫描时，只包含@Controller注解标注的类
         */
        @Filter(type=FilterType.ANNOTATION, classes={Controller.class})
}, useDefaultFilters=false) // value指定要扫描的包
@ComponentScan(value="com.meimeixia", includeFilters={
        /*
         * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
         * classes：我们需要Spring在扫描时，只包含@Service注解标注的类
         */
        @Filter(type=FilterType.ANNOTATION, classes={Service.class})
}, useDefaultFilters=false) // value指定要扫描的包
```
AI写代码java运行

运行IOCTest类中的test01()方法，输出的结果信息如下图所示。



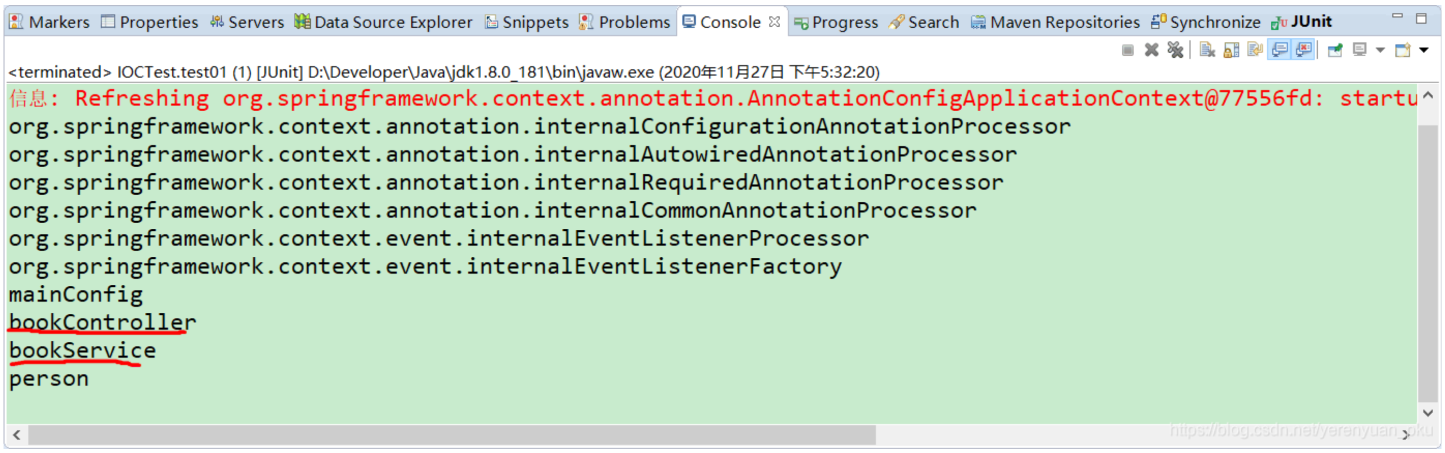
```
<terminated> IOCTest.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 下午5:24:52)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig
bookController
bookService
person
```

可以看到，同时输出了@Controller注解和@Service注解标注的组件名称。

当然了，如果你使用的是Java 8之前的版本，那也没有问题，虽然我们再也不能直接在类上写多个@ComponentScan注解了，但是我们可以在类上使用@ComponentScans注解，同样也可以指定多个@ComponentScan，如下所示。

```java
@ComponentScans(value={
        @ComponentScan(value="com.meimeixia", includeFilters={
                /*
                 * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
                 * classes：我们需要Spring在扫描时，只包含@Controller注解标注的类
                 */
                @Filter(type=FilterType.ANNOTATION, classes={Controller.class})
        }, useDefaultFilters=false), // value指定要扫描的包
        @ComponentScan(value="com.meimeixia", includeFilters={
                /*
                 * type：指定你要排除的规则，是按照注解进行排除，还是按照给定的类型进行排除，还是按照正则表达式进行排除，等等
                 * classes：我们需要Spring在扫描时，只包含@Service注解标注的类
                 */
                @Filter(type=FilterType.ANNOTATION, classes={Service.class})
        }, useDefaultFilters=false) // value指定要扫描的包
})
```
AI写代码java运行

再次运行IOCTest类中的test01()方法，输出的结果信息如下图所示。


Markers Properties Servers Data Source Explorer Snippets Problems Console Progress Search Maven Repositories Synchronize JUnit

```
<terminated> IOCTest.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月27日 下午5:32:20)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig
bookController
bookService
person
```

与使用多个@ComponentScan注解输出的结果信息相同。

小结

我们可以使用@ComponentScan注解来指定Spring扫描哪些包，可以使用excludeFilters()方法来指定扫描时排除哪些组件，也可以使用includeFilters()方法来指定扫描时只包含哪些组件。当使用includeFilters()方法指定只包含哪些组件时，需要禁用掉默认的过滤规则。