

Spring注解驱动开发第4讲——自定义TypeFilter指定@ComponentScan注解的过滤规则

写在前面

Spring的强大之处不仅仅是提供了IOC容器，能够通过过滤规则指定排除和只包含哪些组件，它还能够通过自定义TypeFilter来指定过滤规则。如果Spring内置的过滤规则不能够满足我们的需求，那么我们便可以通过自定义TypeFilter来实现我们自己的过滤规则。

FilterType中常用的规则

在使用@ **ComponentScan**注解 实现包扫描时，我们可以使用@Filter指定过滤规则，在@Filter中，通过type来指定过滤的类型。而@Filter注解中的type属性是一个FilterType枚举，其源码如下图所示。

```
* Copyright 2002-2013 the original author or authors.

package org.springframework.context.annotation;

/**
 * Enumeration of the type filters that may be used in conjunction with
 * {@link ComponentScan @ComponentScan}.
 *
 * @author Mark Fisher
 * @author Juergen Hoeller
 * @author Chris Beams
 * @since 2.5
 * @see ComponentScan
 * @see ComponentScan#includeFilters()
 * @see ComponentScan#excludeFilters()
 * @see org.springframework.core.type.filter.TypeFilter
 */
public enum FilterType {

    /**
     * Filter candidates marked with a given annotation.
     * @see org.springframework.core.type.filter.AnnotationTypeFilter
     */
    ANNOTATION,

    /**
     * Filter candidates assignable to a given type.
     * @see org.springframework.core.type.filter.AssignableTypeFilter
     */
    ASSIGNABLE_TYPE,

    /**
     * Filter candidates matching a given AspectJ type pattern expression.
     * @see org.springframework.core.type.filter.AspectJTypeFilter
     */
    ASPECTJ,

    /**
     * Filter candidates matching a given regex pattern.
     * @see org.springframework.core.type.filter.RegexPatternTypeFilter
     */
    REGEX,

    /**
     * Filter candidates using a given custom
     * {@link org.springframework.core.type.filter.TypeFilter} implementation.
     */
    CUSTOM
}
```

https://blog.csdn.net/yerenyuan_pku

下面我会讲解每一个枚举值的含义。

FilterType.ANNOTATION：按照注解进行包含或者排除

例如，使用@ComponentScan注解进行包扫描时，如果要想按照注解只包含标注了@Controller注解的组件，那么就需要像下面这样写了。

```
1 @ComponentScan(value="com.meimeixia", includeFilters={
2     /*
3     * 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
4     * classes: 我们需要Spring在扫描时，只包含@Controller注解标注的类
5     */
6     @Filter(type=FilterType.ANNOTATION, classes={Controller.class})
7 }, useDefaultFilters=false) // value指定要扫描的包
AI写代码java运行
```

FilterType.ASSIGNABLE_TYPE：按照给定的类型进行包含或者排除

例如，使用@ComponentScan注解进行包扫描时，如果要想按照给定的类型只包含BookService类（接口）或其子类（实现类或子接口）的组件，那么就需要像下面这样写了。

```
1 @ComponentScan(value="com.meimeixia", includeFilters={
2     /*
3     * 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
4     */
5     // 只要是BookService这种类型的组件都会被加载到容器中，不管是它的子类还是它的实现类。记住，只要是BookService这种类型的
6     @Filter(type=FilterType.ASSIGNABLE_TYPE, classes={BookService.class})
7 }, useDefaultFilters=false) // value指定要扫描的包
AI写代码java运行
```

此时，只要是BookService这种类型的组件，都会被加载到容器中。也就是说，当BookService是一个Java类时，该类及其子类都会被加载到Spring容器中；当BookService是一个接口时，其子接口或实现类都会被加载到Spring容器中。

FilterType.ASPECTJ：按照ASPECTJ表达式进行包含或者排除

例如，使用@ComponentScan注解进行包扫描时，按照正则表达式进行过滤，就得像下面这样子写。

```
1 @ComponentScan(value="com.meimeixia", includeFilters={
2     /*
3     * 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
4     */
5     @Filter(type=FilterType.ASPECTJ, classes={AspectJTypeFilter.class})
6 }, useDefaultFilters=false) // value指定要扫描的包
AI写代码java运行
```

这种过滤规则基本上不怎么用！

FilterType.REGEX：按照正则表达式进行包含或者排除

例如，使用@ComponentScan注解进行包扫描时，按照正则表达式进行过滤，就得像下面这样子写。

```
1 @ComponentScan(value="com.meimeixia", includeFilters={
2     /*
3     * 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
4     */
5     @Filter(type=FilterType.REGEX, classes={RegexPatternTypeFilter.class})
6 }, useDefaultFilters=false) // value指定要扫描的包
AI写代码java运行
```

这种过滤规则基本上也不怎么用！

FilterType.CUSTOM：按照自定义规则进行包含或者排除

如果实现自定义规则进行过滤时，自定义规则的类必须是org.springframework.core.type.filter.TypeFilter接口的实现类。

要想按照自定义规则进行过滤，首先我们得创建org.springframework.core.type.filter.TypeFilter接口的一个实现类，例如MyTypeFilter，该实现类的代码一开始如下所示。

```
1 package com.meimeixia.config;
2
3 import java.io.IOException;
4
5 import org.springframework.core.type.classreading.MetadataReader;
6 import org.springframework.core.type.classreading.MetadataReaderFactory;
7 import org.springframework.core.type.filter.TypeFilter;
8
9 public class MyTypeFilter implements TypeFilter {
10
11     /**
12     * 参数:
13     * metadataReader: 读取到的当前正在扫描的类的信息
14     * metadataReaderFactory: 可以获取到其他任何类的信息的（工厂）
15     */
16 }
```

```
16     @Override
17     public boolean match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory) throws IOException {
18
19         return false; // 这儿我们先让其返回false
20
21     }
22 }
23 }
```

AI写代码java运行



当我们实现TypeFilter接口时，需要实现该接口中的match()方法，match()方法的返回值为boolean类型。当返回true时，表示符合规则，会包含在Spring容器中；当返回false时，表示不符合规则，那就是一个都不匹配，自然就都不会被包含在Spring容器中。另外，在match()方法中存在两个参数，分别为MetadataReader类型的参数和MetadataReaderFactory类型的参数，含义分别如下。

- metadataReader：读取到的当前正在扫描的类的信息
- metadataReaderFactory：可以获取到其他任何类的信息的工厂

然后，使用@ComponentScan注解进行如下配置。

```
1 @ComponentScan(value="com.meimeixia", includeFilters={
2     /*
3     * 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
4     */
5     // 指定新的过滤规则，这个过滤规则是我们自个自定义的，过滤规则就是由我们这个自定义的MyTypeFilter类返回true或者false来代表匹配还是没匹配
6     @Filter(type=FilterType.CUSTOM, classes={MyTypeFilter.class})
7 }, useDefaultFilters=false) // value指定要扫描的包
```

AI写代码java运行

FilterType枚举中的每一个枚举值的含义我都讲解完了，说了这么多，其实只有ANNOTATION和ASSIGNABLE_TYPE是比较常用的，ASPECTJ和REGEX不太常用，如果FilterType枚举中的类型无法满足我们的需求时，我们也可以通过实现org.springframework.core.type.filter.TypeFilter接口来自定义过滤规则，此时，将@Filter中的type属性设置为FilterType.CUSTOM，classes属性设置为自定义规则类所对应的Class对象。

实现自定义过滤规则

从上面可以知道，我们在项目的com.meimeixia.config包下新建了一个类，即MyTypeFilter，它实现了org.springframework.core.type.filter.TypeFilter接口。此时，我们先在MyTypeFilter类中打印出当前正在扫描的类名，如下所示。

```
1 package com.meimeixia.config;
2
3 import java.io.IOException;
4
5 import org.springframework.core.io.Resource;
6 import org.springframework.core.type.AnnotationMetadata;
7 import org.springframework.core.type.ClassMetadata;
8 import org.springframework.core.type.classreading.MetadataReader;
9 import org.springframework.core.type.classreading.MetadataReaderFactory;
10 import org.springframework.core.type.filter.TypeFilter;
11
12 public class MyTypeFilter implements TypeFilter {
13
14     /**
15      * 参数:
16      * metadataReader: 读取到的当前正在扫描的类的信息
17      * metadataReaderFactory: 可以获取到其他任何类的信息的（工厂）
18      */
19     @Override
20     public boolean match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory) throws IOException {
21         // 获取当前类注解的信息
22         AnnotationMetadata annotationMetadata = metadataReader.getAnnotationMetadata();
23         // 获取当前正在扫描的类的类信息，比如说它的类型是什么啊，它实现了什么接口啊之类的
24         ClassMetadata classMetadata = metadataReader.getClassMetadata();
25         // 获取当前类的资源信息，比如说类的路径等信息
26         Resource resource = metadataReader.getResource();
27         // 获取当前正在扫描的类的类名
28         String className = classMetadata.getClassName();
29         System.out.println("----> " + className);
30
31         return false;
32     }
33 }
34 }
```

AI写代码java运行



然后，我们在MainConfig类中配置自定义过滤规则，如下所示。

```

1 package com.meimeixia.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.ComponentScan;
5 import org.springframework.context.annotation.ComponentScan.Filter;
6 import org.springframework.context.annotation.ComponentScans;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.context.annotation.FilterType;
9
10 import com.meimeixia.bean.Person;
11 /**
12  * 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
13  * @author liayun
14  *
15  */
16 // 这个配置类也是一个组件
17 @ComponentScans(value={
18     @ComponentScan(value="com.meimeixia", includeFilters={
19         /*
20          * 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
21          */
22         // 指定新的过滤规则，这个过滤规则是我们自个自定义的，过滤规则就是由我们这个自定义的MyTypeFilter类返回true或者false来代表匹配还是没匹配
23         @Filter(type=FilterType.CUSTOM, classes={MyTypeFilter.class})
24     }, useDefaultFilters=false) // value指定要扫描的包
25 })
26 @Configuration // 告诉Spring这是一个配置类
27 public class MainConfig {
28
29     // @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
30     @Bean("person")
31     public Person person01() {
32         return new Person("liayun", 20);
33     }
34 }
35
36 AI写代码java运行

```



接着，我们运行IOCTest类中的test01()方法进行测试，该方法的完整代码如下所示。

```

1 @SuppressWarnings("resource")
2 @Test
3 public void test01() {
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig.class);
5     // 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
6     String[] definitionNames = applicationContext.getBeanDefinitionNames();
7     for (String name : definitionNames) {
8         System.out.println(name);
9     }
10 }
11
12 AI写代码java运行

```



此时，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6:46:05)
十一月 27, 2020 6:46:05 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepare
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu
--->com.meimeixia.test.IOCTest
--->com.meimeixia.bean.Person
--->com.meimeixia.config.MyTypeFilter
--->com.meimeixia.controller.BookController
--->com.meimeixia.dao.BookDao
--->com.meimeixia.MainTest
--->com.meimeixia.service.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
mainConfig
person
```

可以看到，已经输出了当前正在扫描的类的名称，同时，除了Spring内置的bean的名称之外，只输出了mainConfig和person，而没有输出使用@Repository、@Service、@Controller这些注解标注的组件的名称。这是因为当前MainConfig类上标注的@ComponentScan注解是使用的自定义规则，而在自定义规则的实现类（即MyTypeFilter类）中，直接返回了false，那么就一个都不匹配了，自然所有的bean就都没被包含进去容器中了。

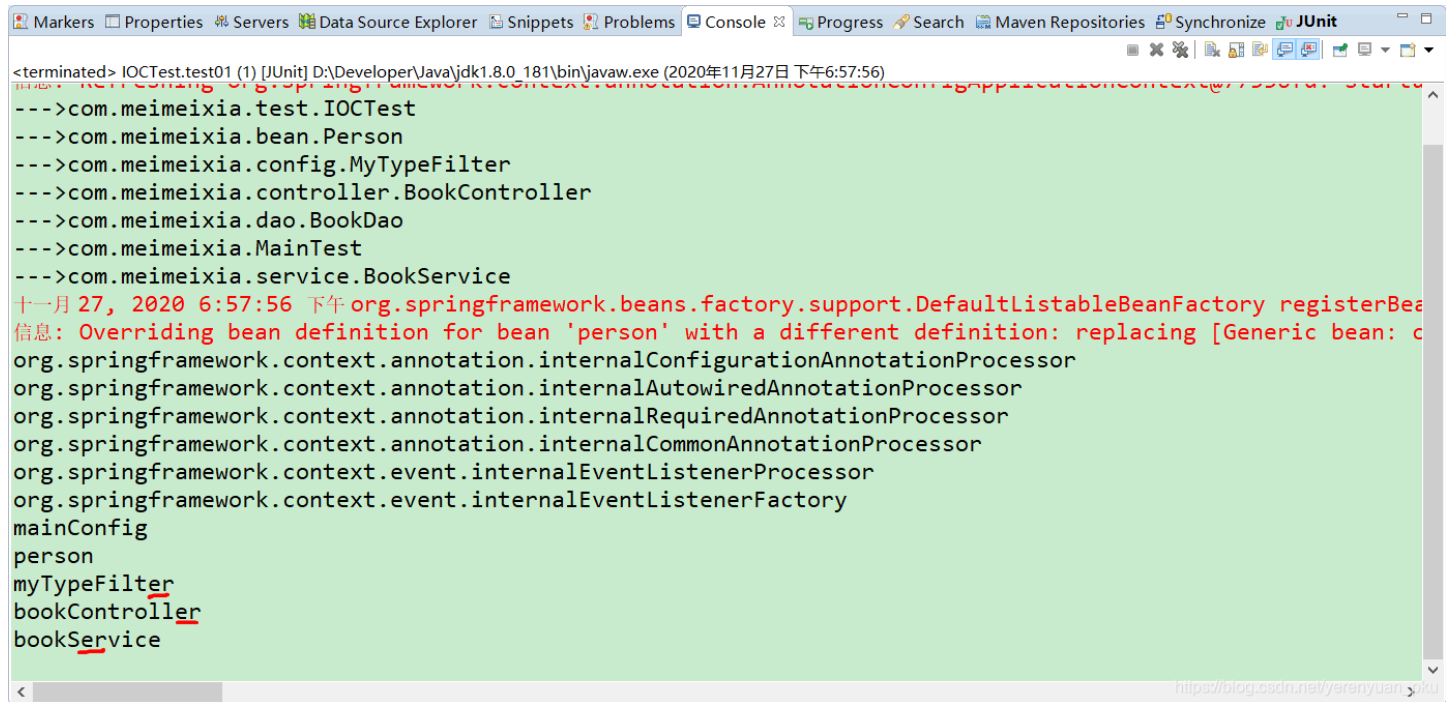
我们可以在MyTypeFilter类中简单的实现一个规则，例如，当前扫描的类名称中包含有"er"字符串的，就返回true，否则就返回false。此时，MyTypeFilter类中match()方法的实现代码如下所示。

```
1 package com.meimeixia.config;
2
3 import java.io.IOException;
4
5 import org.springframework.core.io.Resource;
6 import org.springframework.core.type.AnnotationMetadata;
7 import org.springframework.core.type.ClassMetadata;
8 import org.springframework.core.type.classreading.MetadataReader;
9 import org.springframework.core.type.classreading.MetadataReaderFactory;
10 import org.springframework.core.type.filter.TypeFilter;
11
12 public class MyTypeFilter implements TypeFilter {
13
14     /**
15      * 参数:
16      * metadataReader: 读取到的当前正在扫描的类的信息
17      * metadataReaderFactory: 可以获取到其他任何类的信息的 (工厂)
18      */
19     @Override
20     public boolean match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory) throws IOException {
21         // 获取当前类注解的信息
22         AnnotationMetadata annotationMetadata = metadataReader.getAnnotationMetadata();
23         // 获取当前正在扫描的类的类信息，比如说它的类型是什么啊，它实现了什么接口啊之类的
24         ClassMetadata classMetadata = metadataReader.getClassMetadata();
25         // 获取当前类的资源信息，比如说类的路径等信息
26         Resource resource = metadataReader.getResource();
27         // 获取当前正在扫描的类的类名
28         String className = classMetadata.getClassName();
29         System.out.println("---->" + className);
30
31         // 现在来指定一个规则
32         if (className.contains("er")) {
33             return true; // 匹配成功，就会被包含在容器中
34         }
35
36         return false; // 匹配不成功，所有的bean都会被排除
37     }
38 }
39
```

AI写代码java运行

此时，在com.meimeixia包下的所有类都会通过MyTypeFilter类中的match()方法来验证类名中是否包含有"er"字符串，若包含则返回true，否则返回false。

最后，我们再次运行IOCTest类中的test01()方法进行测试，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6:57:56)
--->com.meimeixia.test.IOCTest
--->com.meimeixia.bean.Person
--->com.meimeixia.config.MyTypeFilter
--->com.meimeixia.controller.BookController
--->com.meimeixia.dao.BookDao
--->com.meimeixia.MainTest
--->com.meimeixia.service.BookService
十一月 27, 2020 6:57:56 下午 org.springframework.beans.factory.support.DefaultListableBeanFactory registerBean
信息: Overriding bean definition for bean 'person' with a different definition: replacing [Generic bean: c
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
person
myTypeFilter
bookController
bookService
```

此时，结果信息中输出了使用@Service和@Controller这两注解标注的组件的名称，分别是bookController和bookService。

从以上输出的结果信息中，你还可以看到输出了一个myTypeFilter，你不禁要问了，为什么会有myTypeFilter呢？这就是因为我们现在扫描的是com.meimeixia包，该包下的每一个类都会进到这个自定义规则里面进行匹配，若匹配成功，则就会被包含在容器中。