Dubbo之@Activate

原创 www.Rui 2020-09-14 19:38:30 177 收藏 版权
分类专栏： Dubbo

建议先阅读上一篇文章Dubbo之@Adaptive，效果更佳。

> **关键词：**@Activate

在《Dubbo之@SPI》和《Dubbo之@Adaptive》中分别介绍了**@SPI**和**@Adaptive**的使用和功能，你会发现，前面这两个注解，通常方法只能获取到一个结果，可是有时候需要获取到多个结果该怎么办？这个时候，**@Activate**就站出来了。

@Activate可以标注在多个实现类上，并通过方法getActivateExtension获取到的也是一个集合。我感觉这个注解的逻辑是三个注解中最绕人的。@Activate可以作用在类上和方法上。

```
public @interface Activate {
    // 设置该实现类的分组名。当获取默认实现类的时候，会过滤掉分组不匹配的实现类
    String[] group() default {};
    // 用于默认实现类的过滤条件。当配置的value值，在指定的URL中有个参数key和其一样（key.equals(value)）或key以.+value结尾（key.endW
    // 并且通过url.getParamter(key)!=null，则匹配，否则过滤掉此实现类
    String[] value() default {};
    // 对默认实现类进行排序，即设置调用的先后顺序。设置在哪些Activate之前被调用
    // 填写扩展实现类的配置key
    // 排序逻辑见ActivateComparator类
    String[] before() default {};
    // 设置在哪些扩展实现类之后被调用
    String[] after() default {};
   // 排序。数值越小，排序越前
    int order() default 0;
}
```

如何使用

关于标注在类上的使用，就不去创建接口和配置了。我们来看下Dubbo自带的**com.alibaba.dubbo.rpc.Filter**接口，该接口是一个SPI接口，实现类上被标注了@Adaptive接口。配置文件的内容如下。

```
cache=com.alibaba.dubbo.cache.filter.CacheFilter
validation=com.alibaba.dubbo.validation.filter.ValidationFilter
echo=com.alibaba.dubbo.rpc.filter.EchoFilter
generic=com.alibaba.dubbo.rpc.filter.GenericFilter
genericimpl=com.alibaba.dubbo.rpc.filter.GenericImplFilter
token=com.alibaba.dubbo.rpc.filter.TokenFilter
accesslog=com.alibaba.dubbo.rpc.filter.AccessLogFilter
activelimit=com.alibaba.dubbo.rpc.filter.ActiveLimitFilter
classloader=com.alibaba.dubbo.rpc.filter.ClassLoaderFilter
context=com.alibaba.dubbo.rpc.filter.ContextFilter
consumercontext=com.alibaba.dubbo.rpc.filter.ConsumerContextFilter
exception=com.alibaba.dubbo.rpc.filter.ExceptionFilter
executelimit=com.alibaba.dubbo.rpc.filter.ExecuteLimitFilter
deprecated=com.alibaba.dubbo.rpc.filter.DeprecatedFilter
compatible=com.alibaba.dubbo.rpc.filter.CompatibleFilter
timeout=com.alibaba.dubbo.rpc.filter.TimeoutFilter
trace=com.alibaba.dubbo.rpc.protocol.dubbo.filter.TraceFilter
future=com.alibaba.dubbo.rpc.protocol.dubbo.filter.FutureFilter
monitor=com.alibaba.dubbo.monitor.support.MonitorFilter
```

Dubbo为我们提供了19个实现类。如何去获取到这些实现类的实例呢，是通过**ExtensionLoader**的**getActivateExtension**方法。该方法有四个重载方法。可分组用于指定**URL**，URL中的参数key，分组名，来作为参数筛选符合条件的实现类实例。

运行代码：

```
import com.alibaba.dubbo.common.URL;
import com.alibaba.dubbo.common.extension.ExtensionLoader;
import com.alibaba.dubbo.rpc.Filter;

import java.util.List;

public class ActivateTest {

    public static void main(String[] args) {

        ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
        URL url = new URL("dubbo", "127.0.0.1", 21880);
        // key指定为空字符串，即没有给出明确的key
        List<Filter> filterList = loader.getActivateExtension(url, "");
        for (Filter filter : filterList) {
            System.out.println(filter.getClass().getName());
        }

    }
}
```

控制台输出结果：（注意这里面是没有CacheFilter，并且顺序和配置文件的顺序不一致）

```
com.alibaba.dubbo.rpc.filter.EchoFilter
com.alibaba.dubbo.rpc.filter.ClassLoaderFilter
com.alibaba.dubbo.rpc.filter.GenericFilter
com.alibaba.dubbo.rpc.filter.ConsumerContextFilter
com.alibaba.dubbo.rpc.filter.ContextFilter
com.alibaba.dubbo.rpc.protocol.dubbo.filter.FutureFilter
com.alibaba.dubbo.rpc.protocol.dubbo.filter.TraceFilter
com.alibaba.dubbo.rpc.filter.TimeoutFilter
com.alibaba.dubbo.monitor.support.MonitorFilter
com.alibaba.dubbo.rpc.filter.ExceptionFilter
```

源码解读

此时你是否会有疑惑。不指定key，竟然也会获取到这么多的实现类？可为什么不是所有的实现类呢，偏偏是这些？下面我们来看看源码。

```
public List<T> getActivateExtension(URL url, String key) {
    // 调用了三个参数的重载方法，如果不指定group，则第三个参数group就是null
    return this.getActivateExtension(url, (String)key, (String)null);
}

public List<T> getActivateExtension(URL url, String key, String group) {
    // 根据key去给定的url中获取参数值
    String value = url.getParameter(key);
    // 如果获取到的value不为空和空字符串，就根据逗号分隔value，可见这里可以是多个值
    return this.getActivateExtension(url, value != null && value.length() != 0 ? Constants.COMMA_SPLIT_PATTERN.split(va
}
```

```
public List<T> getActivateExtension(URL url, String[] values, String group) {
    // 用于存放符合条件的实现类实例
    List<T> exts = new ArrayList();
    // 将values数组改造成List
    List<String> names = values == null ? new ArrayList(0) : Arrays.asList(values);
    String name;
    // 如果名称集合中不含有-default
    // -default前面的-，是减号，就是去掉默认的意思。哪些是默认的呢，就是配置文件中配置的并且类上标了Activate的实现类
    // 下面是匹配默认的扩展实现类
    if (!((List)names).contains("-default")) {
```

```
        this.getExtensionClasses();
        Iterator var6 = this.cachedActivates.entrySet().iterator();

        while(var6.hasNext()) {
            Entry<String, Activate> entry = (Entry)var6.next();
            // 这里的name就是配置文件中的配置key
            name = (String)entry.getKey();
            // 获取到每个实现类上的注解信息
            Activate activate = (Activate)entry.getValue();
            // 如果注解配置的group和参数group相同或参数group==null
            if (this.isMatchGroup(group, activate.group())) {
                T ext = this.getExtension(name);
                // 避免重复加入集合的处理，以及没有设置去掉该类
                // 还要满足isActive这个方法。这个方法就是看URL中的参数，有没有一个与@Activate配置的value值一样的key，有则匹配
                if (!((List)names).contains(name) && !((List)names).contains("-" + name) && this.isActive(activate, url
                    exts.add(ext);
                }
            }
        }

        Collections.sort(exts, ActivateComparator.COMPARATOR);
    }

    List<T> usrs = new ArrayList();
    // 根据名称来匹配（名称是直接通过参数传参，或者通过指定key，去url中获取对应的名称）
    // 获取两种：1、显示指定了名称的默认实现类
    // 2、没有标注@Activate注解的实现类
    for(int i = 0; i < ((List)names).size(); ++i) {
        name = (String)((List)names).get(i);
        // 也是要没有设置去掉这个类
        if (!name.startsWith("-") && !((List)names).contains("-" + name)) {
            if ("default".equals(name)) {
                if (!usrs.isEmpty()) {
                    exts.addAll(0, usrs);
                    usrs.clear();
                }
            } else {
                T ext = this.getExtension(name);
                usrs.add(ext);
            }
        }
    }

    if (!usrs.isEmpty()) {
        exts.addAll(usrs);
    }

    return exts;
}
```

简单总结下：获取@Activate标注的接口实现类实例，是通过调用getActivateExtension方法的。先根据参数筛选默认实现类，再根去根据参数key（或直接指定名称）的名称去匹配不在默认分组中的实现类。调用方法的时候，可以指定一个key，用于去url中根据key获取值去获取对应的实现类；可以指定分组来过滤默认的实现类（在没有-default的前提下），如果没有指定分组，则从所有的默认实现类中去匹配。默认实现类被匹配到的情况除了分组要是指定分组下的，还有就是url中含有一个参数，它的key等于@Activate注解中设置的value值，或者以"."+value值结尾，并且url.getParamter(key)不为空

继续使用

下面我们来更灵活的使用getActivateExtension方法。

```
// 将不会获取到任何结果，因为去掉了默认实现类
ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
URL url = new URL("dubbo", "127.0.0.1", 21880);
URL newUrl = url.addParameter("key", "-default");
List<Filter> filterList = loader.getActivateExtension(newUrl, "key");
System.out.println(filterList.size());
```

```
// 将只获取到CacheFilter，虽然去掉了默认实现类，CacheFilter也属于默认实现类，但后面又显示地加上了cache
// 于是会获取配置key：cache对应的实现类
ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
URL url = new URL("dubbo", "127.0.0.1", 21880);
URL newUrl = url.addParameter("key", "-default,cache");
List<Filter> filterList = loader.getActivateExtension(newUrl, "key");
System.out.println(filterList.size());
for (Filter filter : filterList) {
    System.out.println(filter.getClass().getName());
}
```

```
// 这里还是只会获取一个CacheFilter。
// 虽然没有去掉默认实现类，但因为所有的默认实现类的分组都不匹配group这个单词
// 那为什么CacheFilter没有被过滤掉呢。因为在获取默认实现类的时候，其实CacheFilter
// 是被过滤掉了的，但因为又显示指定了cache，又会获取一次。
ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
URL url = new URL("dubbo", "127.0.0.1", 21880);
URL newUrl = url.addParameter("key", "cache");
List<Filter> filterList = loader.getActivateExtension(newUrl, "key", "group");
System.out.println(filterList.size());
for (Filter filter : filterList) {
    System.out.println(filter.getClass().getName());
}
```

```
// 除了设置key，去URL中获取参数key的值，还可以直接指定这个值
// 不用在去参数里面获取了，直接给
ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
URL url = new URL("dubbo", "127.0.0.1", 21880);
String[] values = {"cache"};
List<Filter> filterList = loader.getActivateExtension(url, values);
System.out.println(filterList.size());
for (Filter filter : filterList) {
    System.out.println(filter.getClass().getName());
}
```

```
// 这样也会获取到cache的，因为URL中有参数的key是cache，匹配上了@Activate注解
// 配置的value值。这是isActive方法的满足条件
ExtensionLoader<Filter> loader = ExtensionLoader.getExtensionLoader(Filter.class);
URL url = new URL("dubbo", "127.0.0.1", 21880);
URL newUrl = url.addParameter("cache", "1");
List<Filter> filterList = loader.getActivateExtension(newUrl, "");
System.out.println(filterList.size());
for (Filter filter : filterList) {
    System.out.println(filter.getClass().getName());
}
```

关于排序：对于默认的实现类，会对其进行排序，逻辑在ActivateComparator

```
return n1 > n2 ? 1 : -1;
```

先根据before和after排序。没有配置before和after的值的话，再根据order进行升序排序。没有设置order的话，默认值是0。

欢迎评论指导，留下您宝贵的建议或意见，谢谢！