

Dubbo之@SPI

原创 www.Rui 2020-09-30 13:14:36 115 收藏
分类专栏: [Dubbo](#)

版权

学习Apache Dubbo，从[官网](#)开始。

关键词: [Dubbo](#) [RPC](#) [SPI](#)

Dubbo的jar包分为两种，一种是org.apache包下的，一种是com.alibaba包下的。本文及后续的使用均来自com.alibaba包下的Dubbo。

如何使用

①创建MAVEN项目，并引入com.alibaba.dubbo依赖。

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>dubbo</artifactId>
  <version>2.6.9</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.13.3</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.30</version>
</dependency>
```

②在resources根路径下创建log4j.properties。

```
log4j.rootLogger=INFO,CONSOLE
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[frame] %d{yyyy-MM-dd HH:mm:ss,SSS} - %-4r %-5p [%t] %C:%L %x - %m%n
```

③创建接口，并在接口上标注@SPI注解。

```
package study.rui.dubbo;

import com.alibaba.dubbo.common.extension.SPI;

@SPI("dog")
public interface AnimalService {

    void say();

}
```

④创建接口实现类。

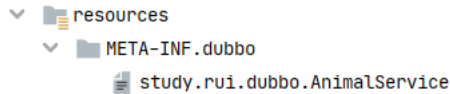
```
package study.rui.dubbo.impl;

import study.rui.dubbo.AnimalService;
```

```
public class DogService implements AnimalService {  
  
    @Override  
    public void say() {  
        System.out.println("dog say woo woo");  
    }  
  
}
```

⑤在resources下创建META-INF/dubbo文件夹，在文件夹下创建文件，文件名是接口的全限定名，内容和property配置文件类似，也是key=value格式。value是接口实现类的全限定名，key是给这个类起的别名，符合命名规范即可。如：

dog=study.rui.dubbo.impl.DogService。（称呼约定，后面内容我将等于号左边叫做配置key，等于号右边叫做配置value。）



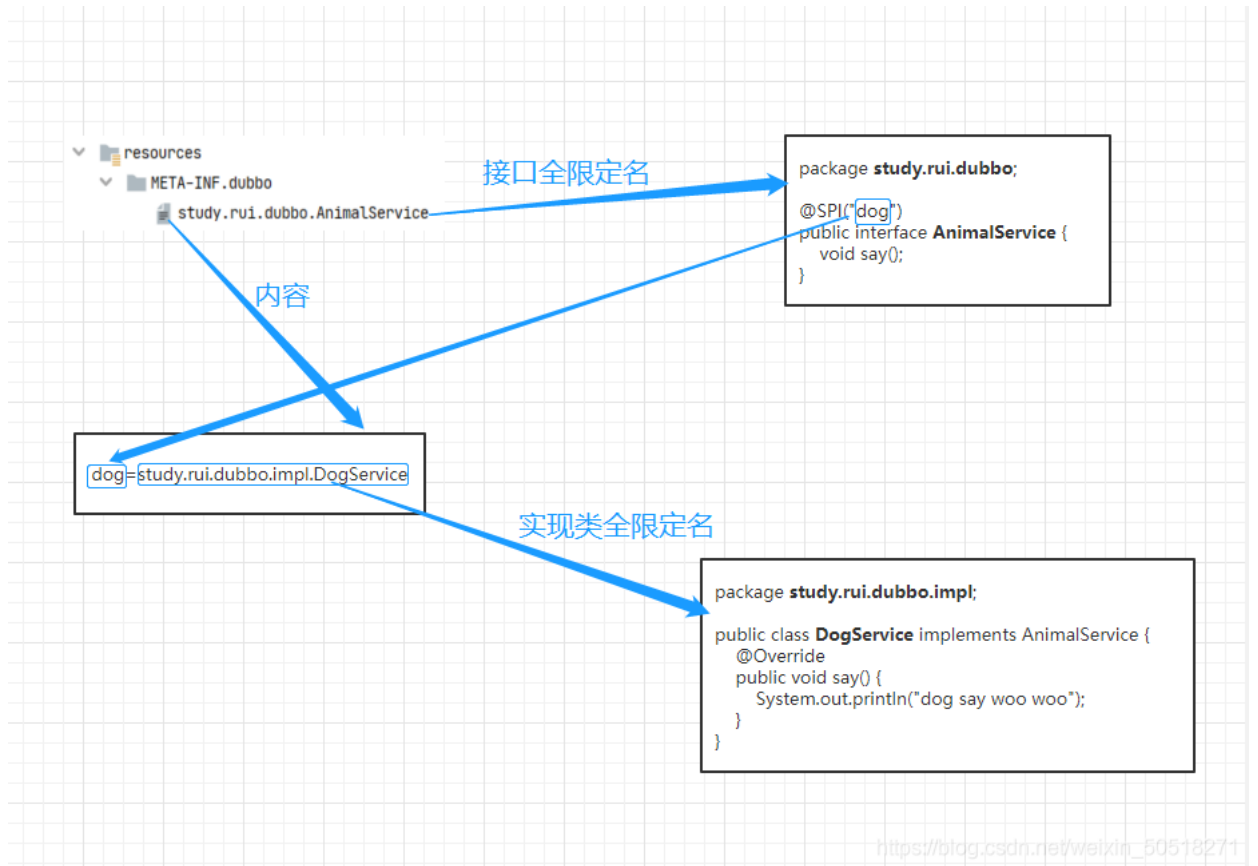
```
resources  
└─ META-INF  
    └─ dubbo  
        └─ study.rui.dubbo.AnimalService
```

⑥运行代码。

```
import com.alibaba.dubbo.common.extension.ExtensionLoader;  
import study.rui.dubbo.AnimalService;  
  
public class SPITest {  
  
    public static void main(String[] args) {  
  
        ExtensionLoader<AnimalService> loader = ExtensionLoader.getExtensionLoader(AnimalService.class);  
        // 创建DogService实例  
        AnimalService animalService = loader.getExtension("dog");  
        // 结果输出: dog say woo woo  
        animalService.say();  
  
    }  
  
}
```

■ 总结说明

①学习Dubbo的SPI之前，可以先了解一下Java的SPI，主要的类是：ServiceLoader。对于ServiceLoader的扩展，在很多框架中都能看见它的影子，比如Spring Boot用它实现了自动注入。



②关系梳理图。

③接口的实现类必须提供一个public或protected的无参构造函数。因为Dubbo在创建实例的时候，使用的是：

```
private T createExtension(String name) {
    Class<?> clazz = (Class)this.getExtensionClasses().get(name);
    if (clazz == null) {
        throw this.findException(name);
    } else {
        try {
            T instance = EXTENSION_INSTANCES.get(clazz);
            if (instance == null) {
                EXTENSION_INSTANCES.putIfAbsent(clazz, clazz.newInstance());
                instance = EXTENSION_INSTANCES.get(clazz);
            }
        }
    }
}
```

④@SPI，只能作用于接口上。这取决于要获取接口的扩展实现类，需要先获取接口的ExtensionLoader实例。ExtensionLoader的构造器被私有化，正常无法通过new来创建实例，（当然如果通过反射获取，那我就告你不礼），可以通过对外暴露的静态方法getExtensionLoader来获取ExtensionLoader实例。获取指定类型的扩展加载器有两个必要条件：类型必须是接口、接口上必须标注@SPI注解。（见源码）

```
public static <T> ExtensionLoader<T> getExtensionLoader(Class<T> type) {
    .....
    if (!type.isInterface()) {
        // 检查参数是否是接口类型，不是，抛异常
        throw new IllegalArgumentException("Extension type(" + type + ") is not interface!");
    } else if (!withExtensionAnnotation(type)) {
        // 检查接口上是否标注@SPI，没有，抛异常
        throw new IllegalArgumentException("Extension type(" + type + ") is not extension, because WITHOUT @" + SPI.class.getName());
    }
    .....
}
```

⑤【一夫一妻制】@SPI标注的接口和ExtensionLoader实例是一一对应的关系。一个SPI接口只有一个属于它的ExtensionLoader实例，并且一个ExtensionLoader实例只能被一个接口拥有，对应的关系被缓存在EXTENSION_LOADERS中（结婚证）。设计成一夫一妻制解决什么问题呢。比如，例子中dog对应了AnimalService的实现类DogService，那又来了个CarService接口，人家也想起名dog来对应哈弗大狗这种车(HAVADogService)。如果AnimalService和CarService共享一个ExtensionLoader实例，那通过

getExtensionLoader("dog"), 到底给你返回DogService还是HAVADogService实例呢。（好好想想我们为什么要实行一夫一妻制，你品，你细品）

⑥@SPI注解有个value属性，用于指定当前接口的默认扩展实现类。value的值就是步骤3中配置文件中的配置key。如果没有设置value的值，则调用如下两个get方法，会返回null。因为我在步骤1中指定了value的值为dog，所有这里会返回DogService实例。

```
ExtensionLoader<AnimalService> loader = ExtensionLoader.getExtensionLoader(AnimalService.class);
AnimalService service1 = loader.getExtension("true");
AnimalService service2 = loader.getDefaultExtension();
service1.say();
service2.say();
```

⑦【吕子乔也叫吕小布】人有大名和小名，甚至还有外号。扩展实现类当然也可以同时拥有好几个名称。在步骤3中，将dog=study.rui.dubbo.impl.DogService改成dog,gou=study.rui.dubbo.impl.DogService。等号左边配置多个名称时，用英文逗号分开，即一个接口实现类对应多个配置key。这样通过getExtension("dog")或getExtension("gou")都能获取到DogService实例。但不管名称有多少个，人都是同一个人，所以这里获取到的都是单例实例。因为ExtensionLoader是将Class对象和实现类实例缓存在EXTENSION_INSTANCES中的。

⑧【狡兔三窟】步骤3中创建的配置文件。不仅仅可以放在META-INF/dubbo路径下，还可以放在META-INF/dubbo/internal和META-INF/services。Dubbo会从这三个地址去加载指定接口全限定名（getExtensionLoader方法的参数）对应的文件内容。

⑨除了通过ExtensionLoader提供的方法获取扩展实现类实例，还可以通过ExtensionFactory的getExtension方法去获取。ExtensionFactory是一个接口，Dubbo为我们提供了三个实现类，分别是AdaptiveExtensionFactory、SpiExtensionFactory和SpringExtensionFactory。AdaptiveExtensionFactory是基础的ExtensionFactory，可通过其获取到其他两个ExtensionFactory。ExtensionLoader的私有构造函数中有这么一段代码：

```
// 当参数的type类型不是ExtensionFactory类型，通过getAdaptiveExtension()
// 获取到的就是AdaptiveExtensionFactory。这里涉及到了@Adaptive知识点了，将在下一节讲解
this.objectFactory = type == ExtensionFactory.class ? null : (ExtensionFactory)getExtensionLoader(ExtensionFactory.class)
```

其中objectFactory是ExtensionFactory，其在依赖注入的方法中被使用到。关于Dubbo提供的依赖注入，将在后面的文章说明。