# 一、dubbo的SPI

dubbo的SPI依赖于ExtensionLoader类，这个类可以通过加载指定文件夹下所配的实现类将应用相关类。

## 1.dubbo的SPI扩展

### (1)扩展流程



### (2)相关注解

dubbo通过三个注解@SPI、@Adaptive、@Activate对接口的实现类进行动态扩展。

@SPI(String value):

标注在一个接口上，代表这个接口的实现类可以被扩展，value是该扩展接口默认的实现类。

@Adaptive(String[] value):

①若标注在一个实现类上(不能标注在接口上)，表明这个类是一个自适应的装饰类，在调用getAdaptiveExtension()时不会动态生成编译装饰类。

②若标注在接口的方法上，表明该方法需要被动态生成，在调用getAdaptiveExtension()时会动态生成编译装饰类，没有标记@Adaptive注解的方法默认抛出异常。

③其中的value为动态生成类需要过滤url的字段，会将@SPI中value和@Adaptive中的value先后进行遍历url名字的。

@Activate

@Adaptive是获取一个适配类，而@Activate则可以获取一批适配类，在impl上加上@Adaptive注解在调用Extension. getActivateExtension(URL url, String key)可以获得url满足条件的实现类。

## 2. ExtensionLoader相关api

### ① getExtensionLoader(Interface) 静态方法



### ②ExtensionLoader的<init>

<init>中会保存interface.class，并且初始化内部的objectFactory字段，调用的是

ExtensionLoader.getExtensionLoader(ExtensionFactory.class).getAdaptiveExtension()【1】方法来初始化objectFactory

AdaptiveExtensionFactory被@Adaptive标记并且满足SPI扩展，所以【1】方法获得的其实是AdaptiveExtensionFactory实例。

### ③AdaptiveExtensionFactory的作用

AdaptiveExtensionFactory是装饰类，其内部组合了SpringExtensionFactory和SpiExtensionFactory(分别提供了Spring的IOC和SPI的IOC)

### ④getExtension(key) 实例方法

通过getExtensionLoader(Interface)获得ExtensionLoader的实例之后，可以调用getExtension(key)获取SPI配置的key对应interface的实现类



【1】中主要分成四类：

1）被@Adaptive标记的类

2）含有一个参数为interface的构造函数的实现类（wrapper类）

3）被@Activate标记的类

4）其他类与3）中的类

### ⑤ getAdaptiveExtension() 实例方法

