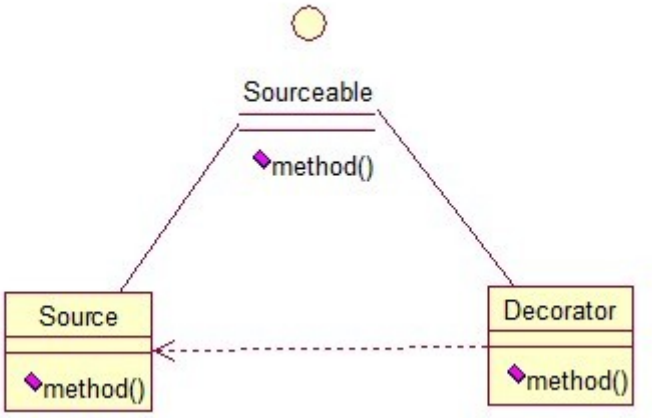
顾名思义，装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例，关系图如下：



Source类是被装饰类，Decorator类是一个装饰类，可以为Source类动态的添加一些功能，代码如下：

[java] view plaincopy

```
public interface Sourceable {
    public void method();
}
```

[java] view plaincopy

```
public class Source implements Sourceable {

    @Override
    public void method() {
        System.out.println("the original method!");
    }
}
```

[java] view plaincopy

```
public class Decorator implements Sourceable {

    private Sourceable source;

    public Decorator(Sourceable source){
```

```
            super();
            this.source = source;
        }
        @Override
        public void method() {
            System.out.println("before decorator!");
            source.method();
            System.out.println("after decorator!");
        }
    }
```

测试类：

[java] view plaincopy

```
    public class DecoratorTest {

        public static void main(String[] args) {
            Sourceable source = new Source();
            Sourceable obj = new Decorator(source);
            obj.method();
        }
    }
```

输出：

before decorator!

the original method!

after decorator!

装饰器模式的应用场景：

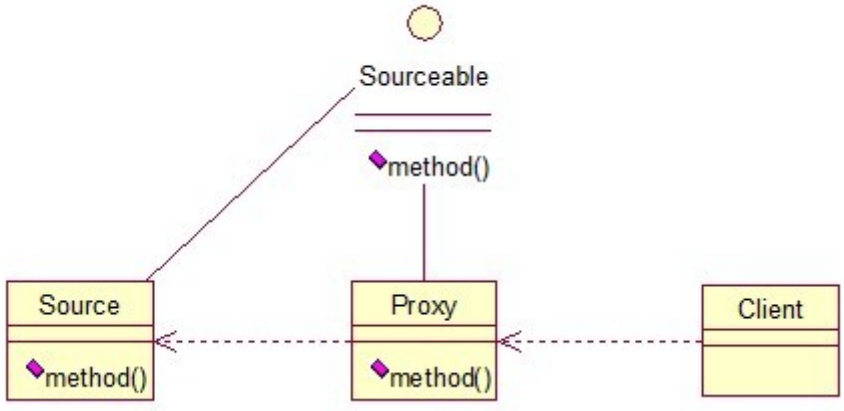
1、需要扩展一个类的功能。

2、动态的为一个对象增加功能，而且还能动态撤销。（继承不能做到这一点，继承的功能是静态的，不能动态增删。）

缺点：产生过多相似的对象，不易排错！

8、代理模式（Proxy）

其实每个模式名称就表明了该模式的作用，代理模式就是多一个代理类出来，替原对象进行一些操作，比如我们在租房子的时候回去找中介，为什么呢？因为你对该地区房屋的信息掌握的不够全面，希望找一个更熟悉的人去帮你做，此处的代理就是这个意思。再如我们有的时候打官司，我们需要请律师，因为律师在法律方面有专长，可以替我们进行操作，表达我们的想法。先来看看关系图：



根据上文的阐述，代理模式就比较容易的理解了，我们看下代码：

[java] view plaincopy

```
public interface Sourceable {
    public void method();
}
```

[java] view plaincopy

```
public class Source implements Sourceable {

    @Override
    public void method() {
        System.out.println("the original method!");
    }
}
```

[java] view plaincopy

```
public class Proxy implements Sourceable {
```

```
    private Source source;
    public Proxy(){
        super();
        this.source = new Source();
    }
    @Override
    public void method() {
        before();
        source.method();
        atfer();
    }
    private void atfer() {
        System.out.println("after proxy!");
    }
    private void before() {
        System.out.println("before proxy!");
    }
}
```

测试类：

[java] view plaincopy

```
public class ProxyTest {

    public static void main(String[] args) {
        Sourceable source = new Proxy();
        source.method();
    }

}
```

输出：

before proxy!

the original method!

after proxy!

代理模式的应用场景：

如果已有的方法在使用的时候需要对原有的方法进行改进，此时有两种办法：

1、修改原有的方法来适应。这样违反了“对扩展开放，对修改关闭”的原则。

2、就是采用一个代理类调用原有的方法，且对产生的结果进行控制。这种方法就是代理模式。

使用代理模式，可以将功能划分的更加清晰，有助于后期维护！