Java 设计模式之代理模式（十二）

一、前言

今天介绍结构型模式中的最后一个模式–代理模式。上篇 Java 设计模式主题为[《Java 设计模式之享元模式（十一）》]。

二、简单介绍

2.1 定义

代理（Proxy）模式是结构型的设计模式之一，它可以为其他对象提供一种代理（Proxy）以控制对这个对象的访问。

所谓代理，是指具有与被代理的对象具有相同的接口的类，客户端必须通过代理与被代理的目标类交互，而代理一般在交互的过程中（交互前后），进行某些特别的处理。

2.2 参与角色

1. 抽象主题（Subject）：真实主题与代理主题的共同接口。
2. 真实主题（RealSubject）：实现抽象主题，定义真实主题所要实现的业务逻辑，供代理主题调用。
3. 代理主题（Proxy）：实现抽象主题，是真实主题的代理。通过真实主题的业务逻辑方法来实现抽象方法，并可以附加自己的操作。

2.3 应用场景

1. 需要控制对目标对象的访问。
2. 需要对目标对象进行方法增强。如：添加日志记录，计算耗时等。
3. 需要延迟加载目标对象。

三、实现方式

代理模式分类：静态代理和动态代理。

区别：静态代理需要手动指定代理对象，动态代理由系统自动生成代理对象。

我们以交学费为例，由于小孩年龄小，安全意识不高，为了安全起见，由父母代理交学费，小孩负责上学。

3.1 静态代理

抽象主题：

```
public interface Child {

    public void payTuition();

    public void goSchool();

}
```

真实主题：

```
public class RealChild implements Child {

    @Override
    public void payTuition() {
        System.out.println("小孩交学费");
    }

    @Override
    public void goSchool() {
        System.out.println("小孩上学");
    }

}
```

此处 goSchool 方法就是供代理对象调用。

代理主题:

```
public class Parent implements Child {

    private Child child;

    public Parent(Child child) {
        this.child = child;
    }

    @Override
    public void payTuition() {
        System.out.println("父母交学费");
    }

    @Override
    public void goSchool() {
        this.child.goSchool();
    }

}
```

父母代理孩子交学费，由小孩自己上学。

客户端：

```
public class Client {

    public static void main(String[] args) {

        Child child = new RealChild();

        // 手动创建代理对象
        Child proxy = new Parent(child);

        proxy.payTuition();

        proxy.goSchool();
    }
}
```

打印结果：

```
父母交学费
小孩上学
```

3.2 动态代理

动态代理的实现手段：JDK 自带的 Proxy 类、CGlib、Javaassist 等。

本次测试使用 Proxy 类演示，抽象主题和真实主题不变，我们创建处理器类：

```
public class ChildHandler implements InvocationHandler {

    public Child child;

    public ChildHandler(Child child) {
        this.child = child;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args)
            throws Throwable {

        Object obj = null;

        // 访问控制
        if ("goSchool".equals(method.getName())) {
            System.out.println("父母交学费");

            obj = method.invoke(child, args);
        }

        return obj;
    }

}
```

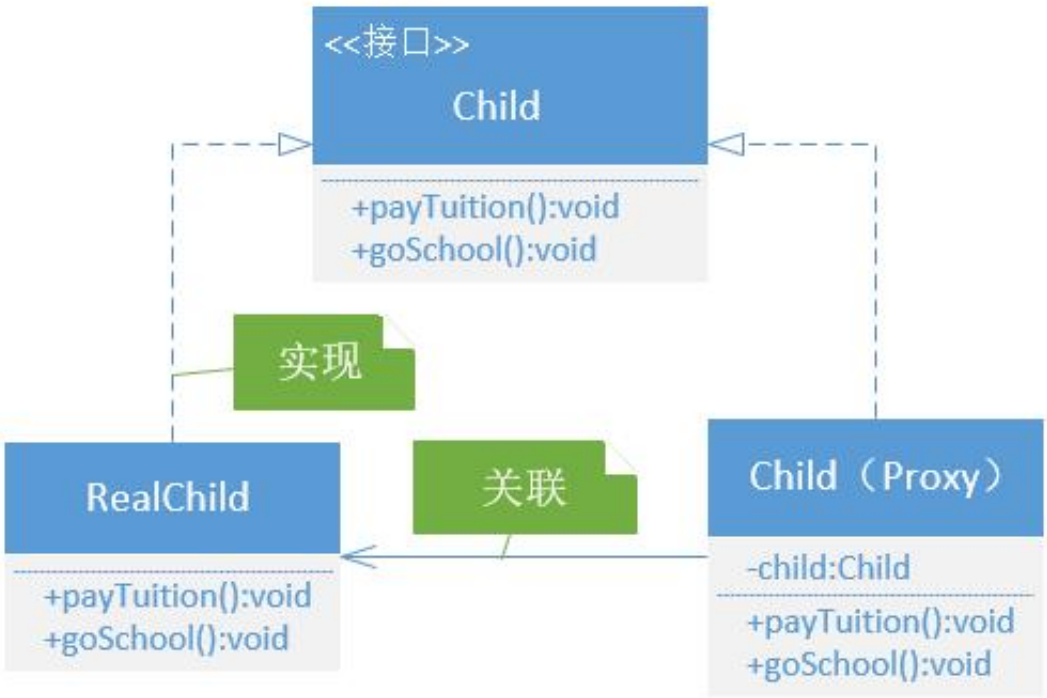
客户端：

```
public class Client {

    public static void main(String[] args) {

        Child child = new RealChild();

        ChildHandler handler = new ChildHandler(child);

        // 代理对象
        Child proxy = (Child) Proxy.newProxyInstance(
                child.getClass().getClassLoader(),
                child.getClass().getInterfaces(),
                handler);

        proxy.payTuition();

        proxy.goSchool();
    }
}
```

打印结果与上文的相同。

在执行 proxy.payTuition() 时，并不是打印“小孩交学费”，而是“父母交学费”，达到了对目标对象访问控制的目的，即控制小孩交学费的行为，让父母代交学费。

UML 类图表示如下：


<<接口>>
Child
+payTuition():void
+goSchool():void
实现
RealChild
+payTuition():void
+goSchool():void
关联
Child（Proxy）
-child:Child
+payTuition():void
+goSchool():void