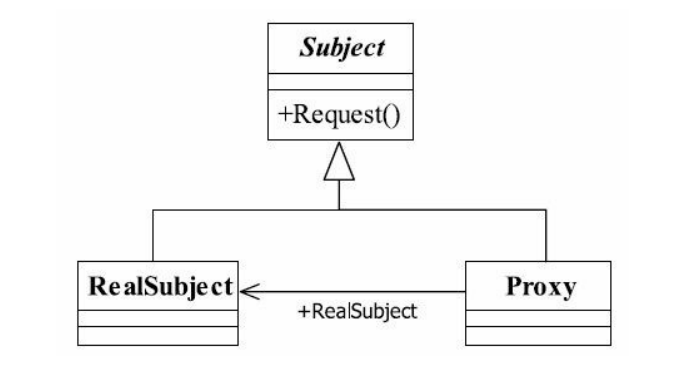
代理模式

## 1 定义

为其他对象提供一种代理以控制对这个对象的访问。

## 2 类图



代理类与实体类都继承自同一接口，代理类持有实体类的引用，通过调用代理类的方法，实现对实体类的调用。

Subject类可以是接口，也可以是抽象类。

## 3 实质

代理模式的实质是：在不改变原有代码（或不知道）的基础上，在方法的前后加强逻辑。

例子：代理玩游戏，根据游戏时间收费。或者对jar包中的类方法在调用前后打log。

## 4 代码

**interface** **IRun** {

**public** void run();

}

**class** **Person** **implements** IRun{

**public** void run() {

System.out.println("person running.....");

}

}

**class** **PersonProxy** **implements** IRun{

**private** Person person;

PersonProxy(Person person){

**this**.person = person;

}

**public** void run() {

*//运行之前的处理*

person.run();

*//运行之后的处理*

}

}

## 5 使用场景

1远程代理

为一个对象在不同的地址空间提供局部代表。隐藏对象存在于不同地址空间的事实。

2虚代理

根据需要创建开销很大的对象。实际上是Copy-on-write的优化方式，即用代理延迟这一copy过程，只有在修改的时候才进行拷贝。

例如在网页加载图片时，使用ImageProxy代理类，先返回图片的长宽占据位置，然后再加载具体的图片。

3 保护代理

控制对原始对象的访问，或者进行一些附加操作。

## 6 优点

1职责清晰

真实的角色只实现实际的业务逻辑，不用关心非职责的事物，通过后期的代理完成一些事物。

2 高扩展性

具体实体可以随时变化，只需要实现对应接口，代理类不用做修改。

## 7 缺点

普通代理只能代理固定接口的实现类。

## 8为什么要用代理？

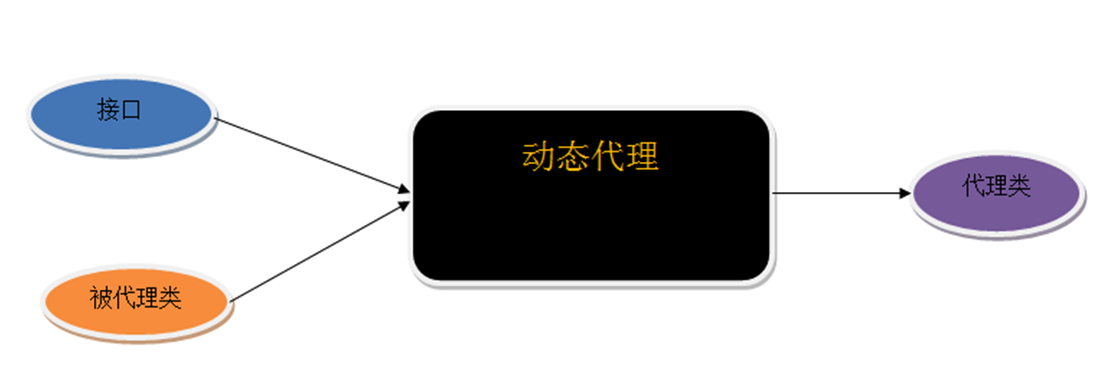
打官司要找律师，你只需要完成自己的答辩，其它事前调查、事后追查都交给律师搞定，减轻负担。

## 9 变形

## 10 动态代理

动态代理是在实现阶段不用关心代理谁，而在运行阶段才指定代理哪一个对象。不需要自己写代理类，由代码自动生成代理类。典型应用包括AOP。

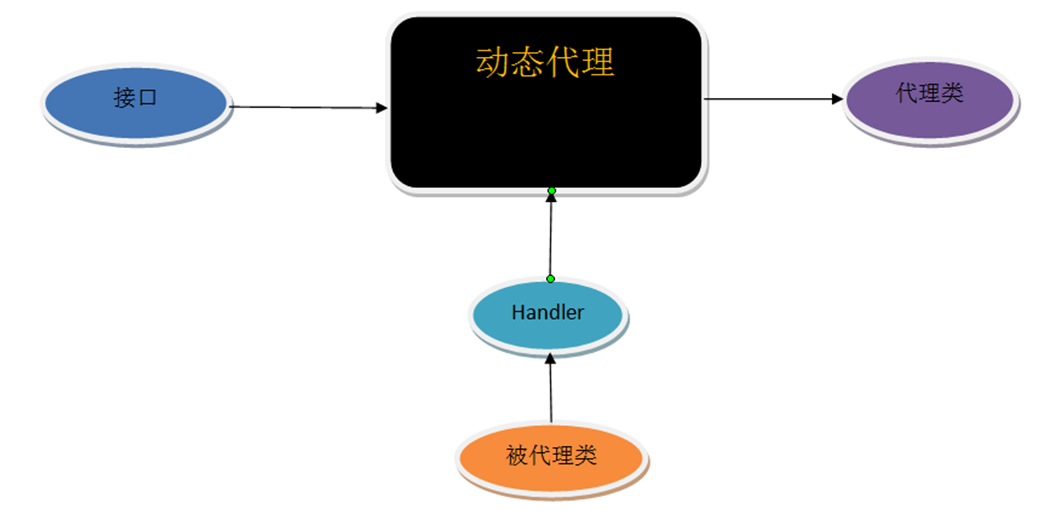
### 10.1 理解

我们可以先把动态代理当做一个黑箱子，然后看它的输入与输出。输入：被代理的类以及它实现的接口。输出：代理类。

根据上图，大致可以得知动态代理的过程为：

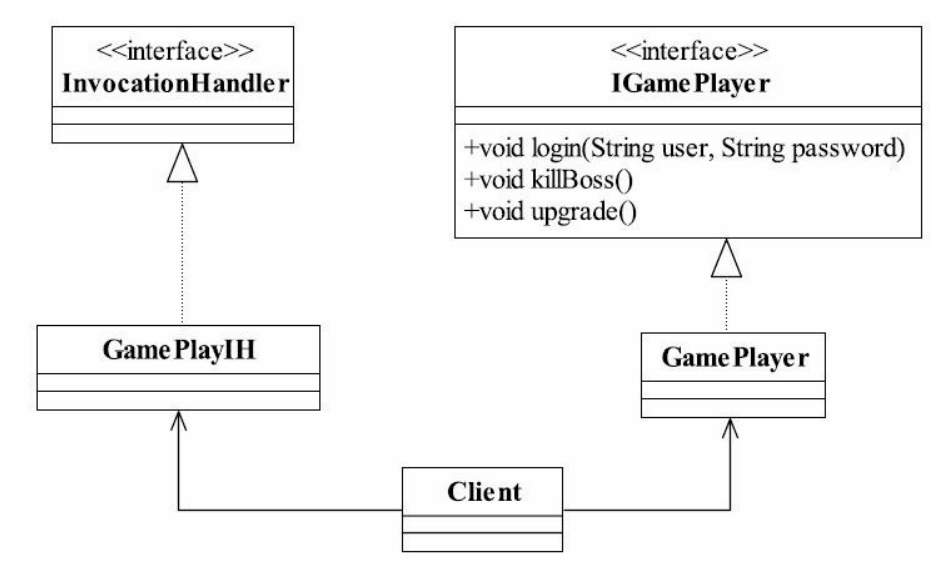
1. 根据输入的接口，利用反射机制，肯定可以拿到有哪些方法。
2. 根据输入的被代理类，同样利用反射机制，可以调用其实现的方法。

到这里好像少了一点东西，没错，就是少了方法调用前后的加强逻辑。



被代理类不再直接交给黑箱子，而是先给Handler这样一个类，再给黑箱子，在Handler中添加相应的逻辑。

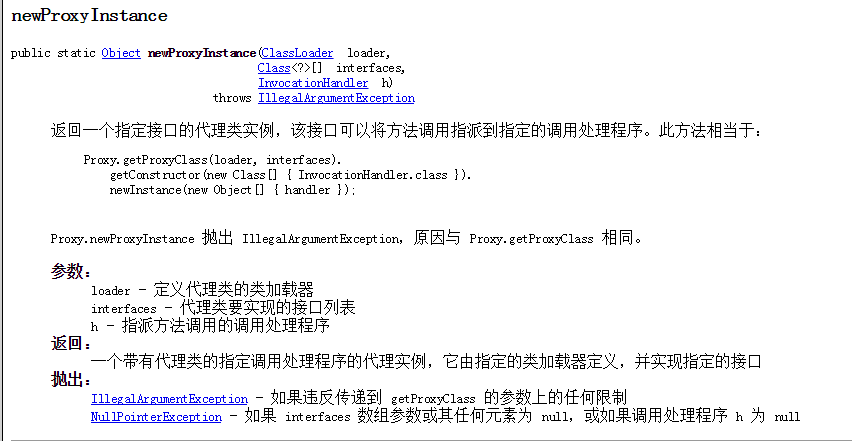
### 10.2 java中的动态代理



黑箱子：Proxy类

Handler：InvocationHandler的子类

利用Proxy的newProxyInstance静态方法，就可以动态生成一个代理类。



10.3 代码

Subject类

**public** **interface** **Subject** {

**public** int sum(int a,int b);

}

RealSubject类

**public** **class** **RealSubject** **implements** Subject{

**public** int sum(int a, int b) {

**return** a+b;

}

}

ProxyHandler类

**public** **class** **ProxyHandler** **implements** InvocationHandler{

**private** Object proxiedObject;

**public** ProxyHandler(Object object) {

proxiedObject = object;

}

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

*// TODO Auto-generated method stub*

System.out.println("before do");

**if**(method.getName().equals("sum")){

int a = (int)method.invoke(proxiedObject, args);

System.out.println("after do ## result"+a);

**return** a;

}**else** **if** (method.getName().equals("doSomething")) {

method.invoke(proxiedObject, args);

System.out.println("after do");

}

**return** **null**;

}

}

Client类

**final** **class** **Client** {

**public** **static** void main(String[] args) **throws** Exception{

*// TODO Auto-generated method stub*

Subject realSubject = **new** RealSubject();

Subject proxy = (Subject) Proxy.newProxyInstance(Subject.class.getClassLoader(),

**new** Class[]{Subject.class},

**new** ProxyHandler(realSubject));

int a = proxy.sum(1,2);

}

}

Proxy.newProxyInstance生成代理类，调用代理类的sum，就会走到invoke方法当中（代理类对象持有ProxyHandler对象的实例）。Invoke方法的三个参数分别为代理类实例，调用的方法以及方法的参数。