代理设计模式

代理模式理解：

代理模式的优点|：

**避免直接访问目标对象**

**功能型扩展**

一开始是写静态代理模式

目标类：

**public** **class** WanZong {

**public** **void** jieqian(){

System.*out*.println("王总借钱了");

}

}

代理类：

**public** **class** XiaoMi {

WanZong wanZong = **new** WanZong();//持有对象引用

**public** **void** jieqian(){

System.*out*.println("小蜜代理借钱");

wanZong.jieqian();

}

}

测试类：

**public** **class** Test {

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

XiaoMi xiaoMi = **new** XiaoMi();

xiaoMi.jieqian();

}

}

从上面可知，其我们不经过目标类实现功能，这也就是当各个银行跨行转账时，就必须使用其中的引用对象，但是由于保密机制，就不能真正的给其真实对象，只能是代理模式

静态代理模式缺点：

每一个目标类，都要一个代理对象，所以不符合java的特性，所以要使用动态代理模式

**动态代理模式：**

1. 所有目标对象必须使用同一个接口

**public** **interface** LaoZong {

**public** **void** jieqian();

**public** **class** WuZong **implements** LaoZong {

@Override

**public** **void** jieqian() {

System.*out*.println("吴总借给我钱咯");

}

1. 动态代理类型实现InvocationHandler 接口

**public** **class** DynaPxoy **implements** InvocationHandler {

//定义一个目标对象

**private** Object targetObject;

**public** Object getPoxyInstance(Object o){

targetObject = o;

**return** Proxy.*newProxyInstance*(DynaPxoy.**class**.getClassLoader(), targetObject.getClass().getInterfaces(), **this**);

}

@Override

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

**throws** Throwable {

// **TODO** Auto-generated method stub

**return** method.invoke(targetObject, args);

}

}

1. Test

**public** **class** Test {

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

LaoZong lao = (LaoZong) **new** DynaPxoy().getPoxyInstance(**new** WuZong());

lao.jieqian();

}

}