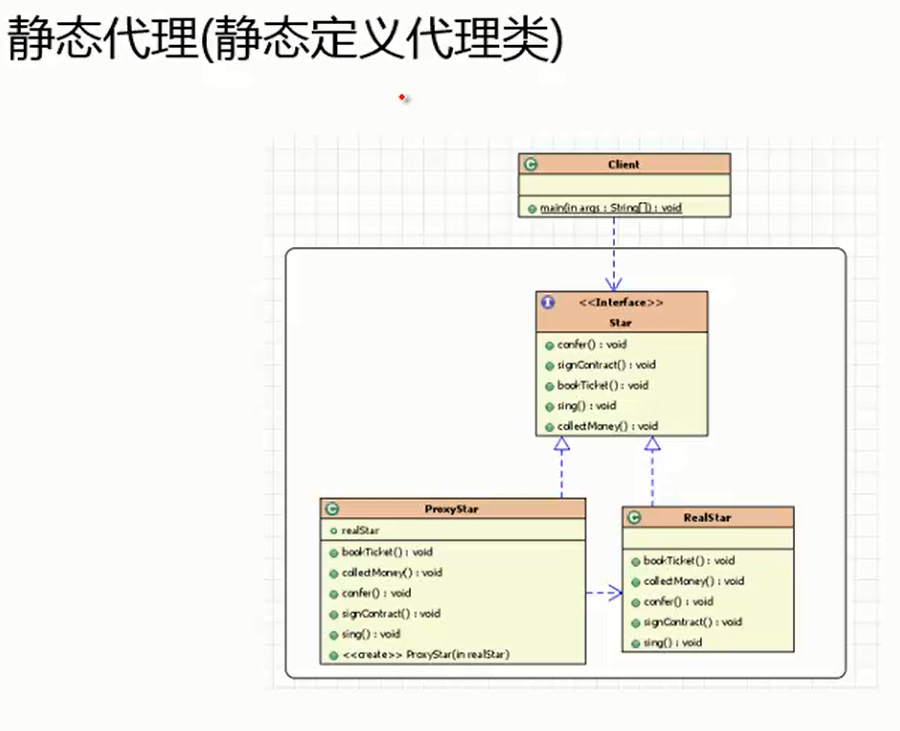
设计模式是一门艺术，如果真正了解这门艺术，你会发现，世界都将变得更加优美。

**定义**

为其它的对象提供一种代理，以控制这个对象的访问

**使用场景**

当不想直接访问某个对象的时候，就可以通过代理



**package** staticProxy;  
  
**public interface** Star {  
 **void** plan();  
  
 **void** talk();  
  
 **void** sell();  
  
 **void** sing();  
  
 **void** rich();  
  
}

**package** staticProxy;  
  
**public class** StaticStar **implements** Star {  
 **private** Star **star**;  
  
 **public** StaticStar(Star star) {  
 **this**.**star** = star;  
 }  
  
 @Override  
 **public void** plan() {  
 System.***out***.println(**"经纪人计划开始演唱会"**);  
 }  
  
 @Override  
 **public void** talk() {  
 System.***out***.println(**"经纪人找合作人谈条件"**);  
  
 }  
  
 @Override  
 **public void** sell() {  
 System.***out***.println(**"经纪人找人卖票"**);  
 }  
  
 @Override  
 **public void** sing() {  
 **star**.sing();  
 }  
  
 @Override  
 **public void** rich() {  
 System.***out***.println(**"经纪人收钱"**);  
 }  
  
 **public void** action() {  
 **this**.plan();  
 **this**.talk();  
 **this**.sell();  
 **this**.sing();  
 **this**.rich();  
 }  
}

**package** staticProxy;  
  
**public class** RealStar **implements** Star {  
 @Override  
 **public void** plan() {  
  
 }  
  
 @Override  
 **public void** talk() {  
  
 }  
  
 @Override  
 **public void** sell() {  
  
 }  
  
 @Override  
 **public void** sing() {  
 System.***out***.println(**"周杰伦唱歌"**);  
 }  
  
 @Override  
 **public void** rich() {  
  
 }  
}

**package** staticProxy;  
  
**public class** Test {  
 **public static void** main(String[] args) {  
 RealStar realStar = **new** RealStar();  
 StaticStar staticStar = **new** StaticStar(realStar);  
  
 staticStar.action();  
  
 }  
}

经纪人计划开始演唱会

经纪人找合作人谈条件

经纪人找人卖票

周杰伦唱歌

经纪人收钱

动态代理

使用动态代理

(1)InvocationHandler接口

在使用动态代理时，我们需要定义一个位于代理类与委托类之间的中介类，这个中介类被要求实现InvocationHandler接口，这个接口的定义如下：

1. **public** interface InvocationHandler {
3. Object invoke(Object proxy, Method method, Object[] args);
5. }

从InvocationHandler这个名称我们就可以知道，实现了这个接口的中介类用做“调用处理器”。当我们调用代理类对象的方法时，这个“调用”会转送到invoke方法中，代理类对象作为proxy参数传入，参数method标识了我们具体调用的是代理类的哪个方法，args为这个方法的参数。这样一来，我们对代理类中的所有方法的调用都会变为对invoke的调用，这样我们可以在invoke方法中添加统一的处理逻辑(也可以根据method参数对不同的代理类方法做不同的处理)。因此我们只需在中介类的invoke方法实现中输出“before”，然后调用委托类的invoke方法，再输出“after”。下面我们来一步一步具体实现它。

1. @Override
3. **public** Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
5. System.**out**.println("before");
7. Object result = method.invoke(obj, args);
9. System.**out**.println("after");
11. **return** result;
13. }
15. }

实现：

**package** dynamicProxy;  
  
**public interface** Star {  
  
 **void** plan();  
  
 **void** talk();  
  
 **void** sell();  
  
 **void** sing();  
  
 **void** rich();  
  
}

**package** dynamicProxy;  
  
**public class** RealStar **implements** Star {  
 @Override  
 **public void** plan() {  
  
 }  
  
 @Override  
 **public void** talk() {  
  
 }  
  
 @Override  
 **public void** sell() {  
  
 }  
  
 @Override  
 **public void** sing() {  
 System.***out***.println(**"周杰伦唱歌"**);  
 }  
  
 @Override  
 **public void** rich() {  
  
 }  
}

**package** dynamicProxy;  
  
**import** observer.Observer;  
  
**import** java.lang.reflect.InvocationHandler;  
**import** java.lang.reflect.Method;  
  
**public class** StarHandler **implements** InvocationHandler {  
  
 **private** Star **realstar**;  
  
 **public** StarHandler(Star realstar) {  
 **this**.**realstar** = realstar;  
 }  
  
 @Override  
 **public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {  
 Object object = **null**;  
 System.***out***.println(**"代理人谈条件"**);  
 System.***out***.println(**"代理人找人卖票"**);  
 **if** (method.getName().equals(**"sing"**)) {  
 object = method.invoke(**realstar**, args);  
 }  
 System.***out***.println(**"代理人收钱"**);  
 **return** object;  
 }  
}

**package** dynamicProxy;  
  
**import** java.lang.reflect.Proxy;  
  
**public class** Test {  
 **public static void** main(String[] args) {  
 Star realStar = **new** RealStar();  
 StarHandler starHandler = **new** StarHandler(realStar);  
  
 Star proxy = (Star) Proxy.*newProxyInstance*(ClassLoader.*getSystemClassLoader*(), **new** Class[]{Star.**class**}, starHandler);  
 proxy.sing();  
 }  
}