动态代理

动态代理模式的核心特点：一个代理类可以代理所有需要被代理的接口的子类对象，如下所示：

N个接口， N+1个类， 其中 N 个真是业务类，1个代理类(在运行时动态获取当前要代理的接口)。

所以要使用动态代理，那么其代理类类特点必须一致(代理流程一致)。

在Java动态代理中，有两个重要的接口和类，首先我们来熟悉一下。

- InvocationHandler接口

每一个动态代理类都必须要实现InvocationHandler这个接口，这个接口中只存在一个抽象方法。并且为每个代理类的实例对象都关联一个handler，当我们通过代理对象调用一个方法的时候，这个方法的调用就会被转为由InvocationHandler这个接口的 invoke 方法来进行调用。

```
/* @proxy:     指代我们所代理的那个真实对象
 * @method:    指代的是我们所要调用真实对象的某个方法的Method对象
 * @args:      指代的是调用真实对象某个方法时接受的参数
 *
 * @return     方法的返回值
 * @throws     Throwable 可能产生的异常
 */
public Object invoke(Object proxy, Method method, Object[] args)
        throws Throwable;
```

- Proxy类

Proxy这个类的作用就是用来动态创建一个代理对象的类，我们一般使用该类中的：

```
/*
 * @loader: 一个ClassLoader对象，定义了由哪个ClassLoader对象来对生成的代理对象进行加载
 * @interfaces: 一个Interface对象的数组，表示的是要给需要代理的对象提供一组什么接口
 * 如果我提供了一组接口给它，那么这个代理对象就宣称实现了该接口(多态)，这样我就能调用这组接口中的方法了
 * @h: 一个InvocationHandler对象，表示的是当我这个动态代理对象在调用方法的时候
 * 会关联到哪一个InvocationHandler对象上
 */
public static Object newProxyInstance(ClassLoader loader,
                                      Class<?>[] interfaces,
                                      InvocationHandler h)
```

接下来，我们来实战演练一下。

```
// 业务接口
interface ISubject {
    public void eat();
}
```

```java
// 真实业务类
class RealSubject implements ISubject {
    @Override
    public void eat() {
        System.out.println("吃饭");
    }
}

// 动态代理类必须实现 InvocationHandler 接口
class ProxySubject implements InvocationHandler {

    // 指向绑定的接口对象，因为是动态代理，所以接口对象的引用得用Object类
    private Object target;

    // 传入要绑定的真实业务类
    public Object Binder(Object target) {
        this.target = target;
        // 返回该真实业务类的代理业务类
        return Proxy.newProxyInstance(target.getClass().getClassLoader(),
                                      target.getClass().getInterfaces(), this);
    }

    // 辅助业务1
    public void prepareEat() {
        System.out.println("洗手");
    }

    // 辅助业务2
    public void afterEat() {
        System.out.println("洗碗");
    }

    // 这是 InvocationHandler接口唯一的方法，JDK规定要想实现动态代理，必须覆写该方法
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        this.prepareEat();
        Object ret = method.invoke(this.target, args);
        this.afterEat();
        return ret;
    }

}


public class Test {
    public static void main(String[] args) throws Exception {
        // Binder传入真实业务类返回该类对应的代理业务类
        ISubject subject = (ISubject) new ProxySubject().Binder(new RealSubject());
        subject.eat();
    }
}
```