动态代理模式

有很多种创建动态代理模式的方式

但在这里我们就提供一种java api 中自带的 控制器 代理

代理模式思想：

将主角类中主角可以操作，但不必主角操作的方法抽取出来，只保留主角的核心方法，其他冗余都交给代理做

目标类

1. 主角实现类(leadingImpt) 主角接口(leading)
2. 代理类(proxy)

```
//主角身份为 公司总裁 但他也是一个人,人可以做的事情他一样可以做

//主角接口
interface Leading{
    void sweep();  //扫地方法

    void rinse();  //清洗方法

    void power();  //掌控大权
}

//扫地 清洗 这些方法是个人都可以做，但是掌控大权不一定谁都可以
//主要体现了面向对象的封装技术，将共性的方法抽取出来交给代理
//在这个例子中我们的代理就是专门做扫地，清洗的工作
//那么这个代理可以去很多公司做扫地，清洗，而主角就不行
//因为每个公司的业务不同，而1号主角无法代替2号主角做事情
class LeadingImpt implements Leading{
    public void sweep(){
        System.out.println("我公司总裁会扫地！");
    }
    public void power(){
        System.out.println("我公司总裁掌控大权！");
    }
    public void rinse(){
        System.out.println("我公司总裁会清洗！");
    }
}

class proxy implements Leading{
    private Leading leading;
    public proxy(Leading leading){
```

```
        this.leading = leading;
    }
    public void sweep(){
        System.out.println("我代理会扫地，我帮你！");
    }
    public void power(){
        leading.power(); //这个我不会。。总裁还是你来吧
    }
    public void rinse(){
        System.out.println("我代理会清洗，我帮你！");
    }

}
```

java中的动态代理模式的创建　控制器 与 代理

主要将代理 交给 控制器 用控制器调用

控制器接口：InvocationHandler

代理类：Proxy

它们都属于：java.lang.reflect(反射包)下的

用户要想使用动态代理：

第一步：

实现 控制器接口：InvocationHandler 重写

invoke(Object proxy, Method method, Object[] args) throws Throwable 方法

实现类中存放一个主角接口，可以保证传入的对象是主角的子类，且核心方法都可以调用到

第二步：

通过Proxy类名调用

public static Object newProxyInstance(ClassLoader loader,
Class<?>[] interfaces,
InvocationHandler h)
throws IllegalArgumentException

该静态方法

ClassLoader loader 参数1：建议传入主角的类加载器 ，

主角类.getClass().getLoader()获取到主角类加载器

Class<?>[] interfaces **参数2：**主角的.class 对象 格式 new Class[]{主角.class}

InvocationHandler h **参数3：**将我们实现的控制器对象放入

这样就开始了一个动态代理模式.

总结一句话：

主角接口中只需定义核心方法，其他方法定义在主角的实现类中,

代理可以帮助主角做一些初始化的工作，可以调用主角的核心方法，但不能调用主角的独特方法