动态代理

1.代理模式：
无法访问目标对象，通过代理对象进行访问，而且增强式的访问。适合进行业务的扩展。

2.代理模式的功能
- 1.增强功能
- 2.限制目标对象的访问

3.代理模式的分类

2.动态代理

1.静态代理
- 1.**目标对象和代理对象实现同一个业务接口**
- 2.代理对象实现功能时必须依靠目标对象自己的实现
- 3.当业务发生变化时，要进行大量代码的改动，实现复杂
- 4.代理类是以java的文件形式存在，在调用前就已存在，所以比较死板
- 5.静态代理代码实现

```
public class Agent implements YeWu {

    //代理刘德华,还要代理周润发
    YeWu yeWu;
     public Agent(YeWu yeWu){
     this.yeWu = yeWu;
    }
    @Override
    public void sing( ) {
     System.out.println("预订场地.....");
     System.out.println("预订时间.....");
     //请华仔来唱(主业务功能由目标对象亲自实现)
     yeWu.sing();
     System.out.println("结算费用.....");
    }
}
```

1.JDK动态代理

1.特点
- 1.**代理对象不需要实现业务接口**
- 2.**代理对象是在程序运行时动态的在内存中构建**
- 3.**目标对象必须要实现接口，才可以使用JKD的动态代理**

2.使用到的三个类
- 1.**Proxy:负责生成代理对象**
  使用newProxyInstance(ClassLoader loader, Class<?>[] interfaces,
  InvocationHandler handler)
  ClassLoader loader：加载目标对象
  Class<?>[] interfaces：得到目标对象实现的所有接口
  InvocationHandler handler：增强功能写代理的地方
- 2.Method:目标方法的回调用，使用反射的机制
  Object obj = **method.invoke(yewu,args)**;
- 3.**InvocationHandler：增强功能的代码，当外部目标对象方法被调用时，此接口的实现类被执行**

3.代码实现

```
//动态的生成代理对象
  public Object getAgent(){
      return Proxy.newProxyInstance(
  // ClassLoader loader,目标对象的类加载器
     yeWu.getClass().getClassLoader(),
  // Class<?>[] interfaces,目标对象实现的所有业务接口,得到这些接口中的业务方法进行功能增强
     yeWu.getClass().getInterfaces(),
  //InvocationHandler h,增强功能处理器,所有增强的功能都写在这里
     new InvocationHandler() {
              @Override
              public Object invoke(
   //生成的代理对象
   Object proxy,
   //正在被调用的目标方法 sing();  show();  eat();
   Method method,
   //目标方法的参数
   Object[] args) throws Throwable {
   //此方法中就是代理功能的实现,+目标对象调用自己的业务方法
   System.out.println("预订场地.....");
   System.out.println("预订时间.....");
   //                 System.out.println("目标方法的参数:" + Arrays.toString(args));
   //请目标对象来实现业务功能
    Object returnValue = method.invoke(yeWu,args);  //args是目标方法的参数,相当于调用sing(),show(),eat()
    System.out.println("结算费用.....");
    return ((String)returnValue).toUpperCase(); //是目标方法的返回值
              }});
```

2.CGLib动态代理

1.特点
- 1.这是一种子类代理模式.在程序运行时,动态生成代理对象,实现功能增强
- 2.目标对象不需要实现接口, 使用子类来增强功能.
- 3.规避了JDK动态代理中,目标对象必须实现接口的弊端.
- 4.被代理的类不能是final的,被代理的方法也不能是final的.
- 5.底层是由字节码处理框架ASM来完成的

2.代码实现略

4.面向接口编程
- 1.类中的成员方法设计成接口
- 2.方法的参数设计为接口
- 3.方法的返回值设计为接口
- 4.使用时使接口指向实现类