```
package com.proxy.cglib;

import java.lang.reflect.Method;

import net.sf.cglib.core.DebuggingClassWriter;
import net.sf.cglib.proxy.Enhancer;
import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

public class ProxyFactory implements MethodInterceptor {

  //维护一个目标对象
  private Object target;

  //构造器，传入一个被代理的对象
  public ProxyFactory(Object target) {
    this.target = target;
  }

  //返回一个代理对象:  是 target 对象的代理对象
  public Object getProxyInstance() {
    System.setProperty(DebuggingClassWriter.DEBUG_LOCATION_PROPERTY,
"D:\\workspace\\DesignPattern\\out\\production\\DesignPattern\\com\\proxy");

    //1. 创建一个工具类
    Enhancer enhancer = new Enhancer();
    //2. 设置父类
    enhancer.setSuperclass(target.getClass());
    //3. 设置回调函数
    enhancer.setCallback(this);
    //4. 创建子类对象，即代理对象
    return enhancer.create();

  }


  //重写  intercept 方法，会调用目标对象的方法
  @Override
  public Object intercept(Object arg0, Method method, Object[] args, MethodProxy
arg3) throws Throwable {
    // TODO Auto-generated method stub
    System.out.println("Cglib代理模式 ~~ 开始");
    Object returnVal = method.invoke(target, args);
    System.out.println("Cglib代理模式 ~~ 提交");
    return returnVal;
  }
```

```
}
```