```
package com.proxy.dynamic;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

public class ProxyFactory {

  //维护一个目标对象 , Object
  private Object target;

  //构造器， 对target 进行初始化
  public ProxyFactory(Object target) {

    this.target = target;
  }

  //给目标对象 生成一个代理对象
  public Object getProxyInstance() {

    //说明
    /*
     *  public static Object newProxyInstance(ClassLoader loader,
                                  Class<?>[] interfaces,
                                  InvocationHandler h)

        //1. ClassLoader loader ： 指定当前目标对象使用的类加载器, 获取加载器的方法固定
        //2. Class<?>[] interfaces: 目标对象实现的接口类型，使用泛型方法确认类型
        //3. InvocationHandler h : 事情处理，执行目标对象的方法时，会触发事情处理器方法, 会把当前执行的目标对象方法作为参数传入
     */
    return Proxy.newProxyInstance(target.getClass().getClassLoader(),
        target.getClass().getInterfaces(),
        new InvocationHandler() {

          @Override
          public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            // TODO Auto-generated method stub
            System.out.println("JDK代理开始~~");
            //反射机制调用目标对象的方法
            Object returnVal = method.invoke(target, args);
            System.out.println("JDK代理提交");
            return returnVal;
          }
```

```
            });
    }


}
```