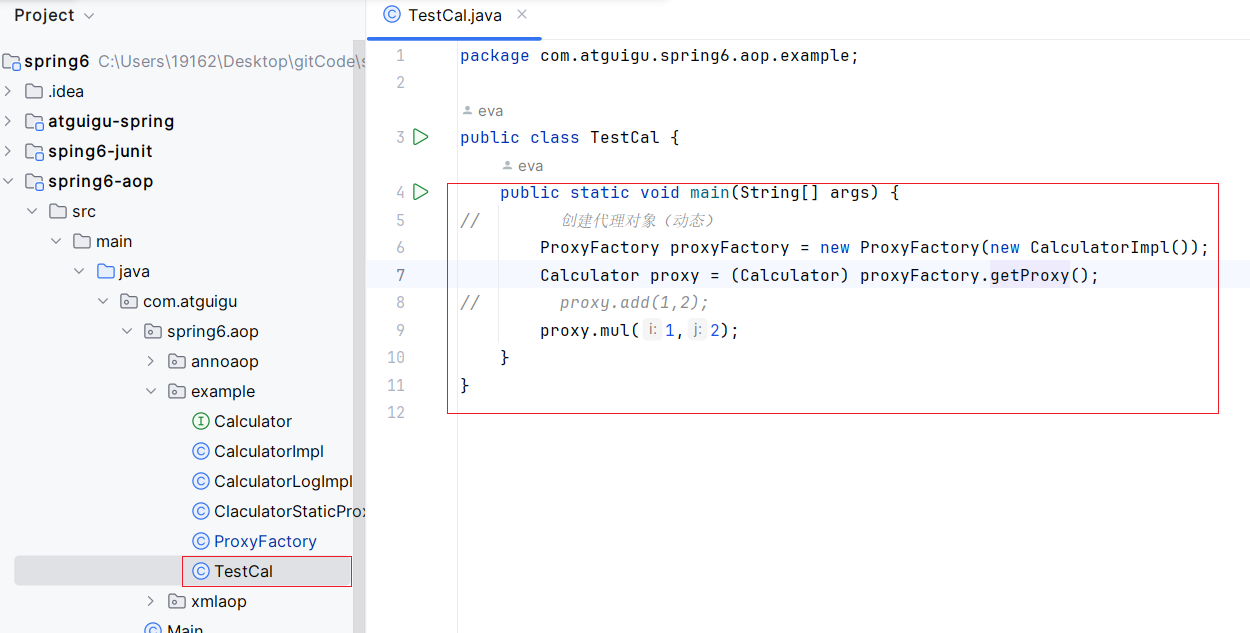


代码：

|  |
| --- |
| package com.atguigu.spring6.aop.example;  import java.lang.reflect.InvocationHandler; import java.lang.reflect.Method; import java.lang.reflect.Proxy; import java.util.Arrays;  public class ProxyFactory { *// 2、目标对象* private Object target;   public ProxyFactory(Object target) {  this.target = target;  }  *// 1、返回代理对象* public Object getProxy(){  */\*\*  \* Proxy.newProxyInstance()方法  \* 有三个参数  \* 第一个参数：ClassLoader：加载动态生成代理类的来加载器  \* 第二个参数：Class*<?>*[] interfaces：目标对象实现的所以接口的class类型数字  \* 第三个参数：InvocationHandler：设置代理对象实现目标对象方法的过程  \*  \*/ // 第一个参数：ClassLoader：加载动态生成代理类的来加载器* ClassLoader classLoader = target.getClass().getClassLoader(); *// 第二个参数：Class<?>[] interfaces：目标对象实现的所有接口的class类型数字* Class<?>[] interfaces = target.getClass().getInterfaces(); *// 第三个参数：InvocationHandler：设置代理对象实现目标对象方法的过程，用匿名内部类实现invoke方法* InvocationHandler invocationHandler = new InvocationHandler(){ *// 第一个参数：代理对象 // 第二个参数：需要重写目标对象的方法 // 第三个参数：method方法里面参数* @Override  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable { *// 方法调用之前输出* System.*out*.println("[动态代理][日志]"+method.getName()+"，参数:"+ Arrays.*toString*(args)); *// 调用目标的方法* Object result = method.invoke(target,args); *// 方法调用之后输出* System.*out*.println("[动态代理][日志]"+method.getName()+"，结果:"+ result);  return result;  }  };  return Proxy.*newProxyInstance*(classLoader,interfaces,invocationHandler);  } } |

测试：



后面都是基于这个原理实现的AOP