```
Project ~
                                    © ProxyFactory.java × © ClaculatorStaticProxy.java
                                                                                   TestCal.java
                                                                                                    C LogAspect.java
                                              package com.atguigu.spring6.aop.example;
                                                                                                                                 A3 ^
spring6 C:\Users\19162\Desktop\gitCode\:
> 🗀 .idea
                                             import java.lang.reflect.InvocationHandler:
> 📑 atguigu-spring
                                             import java.lang.reflect.Method;
> 📑 sping6-junit
                                             import java.lang.reflect.Proxy;

√ □ spring6-aop

                                             import java.util.Arrays;
  ∨ 🗀 src
    🗸 🗀 main
      java
                                             public class ProxyFactory {
         9
                                             // 2、目标对象
4 usages

√ is spring6.aop

             > 🖻 annoaop
                                                 private Object target:
              ✓ i example
                  ① Calculator
                                                public ProxyFactory(Object target) { this.target = target; }
                  © CalculatorImpl
                  © CalculatorLogImpl
                                                  1、返回代理对象
                  © ClaculatorStaticPro:
                 © ProxyFactory
                                                public Object getProxy(){
             > iii xmlaop
                                                      * Proxy.newProxyInstance()方法
             © Main
                                                      * 右三个参数
                                                     * 第一个参数: ClassLoader: 加载动态生成代理类的来加载器
      > resources
                                                      * 第二个参数: Class<?>[] interfaces: 目标对象实现的所以接口的class类型数字
    \oslash .gitignore
                                                      * 第三个参数: InvocationHandler: 设置代理对象实现目标对象方法的过程
    m pom.xml
> 📑 spring6-i18n
> 📑 spring6-ioc-xml
                                                        第一个参数: ClassLoader: 加载动态生成代理类的来加载器
> 3 spring6-reflect
                                                     ClassLoader classLoader = target.getClass().getClassLoader();
> 🛅 spring6-resources
                                                           小念地, Classe2>[] interfa
                                    25
> 🛅 atguigu-spring
                                    26
                                                      第一个参数: ClassLoader: 加载动态生成代理类的来加载器
> 🛅 sping6-junit
                                                    ClassLoader classLoader = target.getClass().getClassLoader();

√ ☐ spring6-aop

                                   28
                                                      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();

∨ □ main

                                  30
                                                     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
      java

√ i com.atguigu

                                                    {\tt InvocationHandler\ invocationHandler\ =\ new\ InvocationHandler()\{}

√ Spring6.aop

             > 🗟 annoaop
                                                          第二个参数: 需要重写目标对象的方法
                                    34
                                                         第三个参数: method方法里面参数
             ✓ ⊚ example
                                                        ≗ eva
                 ① Calculator
                                    35
                 © CalculatorImpl
                                    36 C @
                                                        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
                 © CalculatorLogImpl
                 ClaculatorStaticPro:
                                    38
                                                            System.out.println("[动态代理][日志]"+method.getName()+",参数:"+ Arrays.toString(
                 © ProxyFactory
                 © TestCal
                                    40
                                                           Object result = method.invoke(target.args):
             > a xmlaop
                                                            System.out.println("[动态代理][日志]"+method.getName()+", 结果:"+ result);
      > resources
                                                           return result;
    aitianore .
   mx.moq m
                                                    return Proxy.newProxyInstance(classLoader,interfaces,invocationHandler);
> 🕞 spring6-i18n
> 📴 spring6-ioc-xml
> 📑 spring6-reflect
```

代码:

```
package com.atguigu.spring6.aop.example;
import java.lang.reflect.lnvocationHandler;
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);
   }
```

```
Project \vee
                                   © TestCal.java ×
                                           package com.atguigu.spring6.aop.example;
code\s
> 🗀 .idea
> 🛅 atguigu-spring
                                     3 🗅
                                          public class TestCal {
> 🛅 sping6-junit
                                              ≗ eva
∨ 🕞 spring6-aop
                                               public static void main(String[] args) {
                                     4 >
  ∨ 🗀 src
                                                   创建代理对象(动态)
    🗸 🗀 main
                                                  ProxyFactory proxyFactory = new ProxyFactory(new CalculatorImpl());
      java
                                   7
                                                Calculator proxy = (Calculator) proxyFactory.getProxy();

√ o com.atguigu

                                                   proxy.add(1,2);

√ ispring6.aop

                                                  proxy.mul( i: 1, j: 2);
                                    10
             > 🗟 annoaop
                                           }
             ① Calculator
                 © CalculatorImpl
                 © CalculatorLogImpl
                 © ClaculatorStaticPro:
                 © ProxyFactory
                © TestCal
             > a xmlaop
             ( Main
```

后面都是基于这个原理实现的 AOP