3、AOP

笔记本： spring
创建时间： 2022/4/3 17:03 **更新时间：** 2022/4/20 10:34
作者： 雷丰阳

AOP：（Aspect Oriented Programming）面向切面编程；
OOP：（Object Oriented Programming ）面向对象编程；

面向切面编程：基于OOP基础之上新的编程思想；
指在程序运行期间，将某段代码动态的切入到指定方法的指定位置进行运行的这种编程方式，面向切面编程；

场景：计算器运行计算方法的时候进行日志记录；
加日志记录：
1）、直接编写在方法内部；不推荐，修改维护麻烦；

日志记录：系统的辅助功能；
业务逻辑：（核心功能）
耦合；
2）、我们希望的是；
业务逻辑：（核心功能）；日志模块；在核心功能运行期间，自己动态的加上；
运行的时候，日志功能可以加上；
可以使用动态代理来将日志代码动态的在目标方法执行前后先进行执行；

```
package com.atguigu.proxy;


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


import com.atguigu.inter.Calculator;
import com.atguigu.utils.LogUtils;

/**
 * 帮Calculator.java生成代理对象的类
 * Object newProxyInstance
 * (ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)
 *
 * @author lfy
 *
 */
public class CalculatorProxy {

    /**
     * 为传入的参数对象创建一个动态代理对象
     * @param calculator
     * @return
     *
     * Calculator calculator:被代理对象；（宝宝）
     * 返回的：宋喆
     */
    public static Calculator getProxy(final Calculator calculator) {
        // TODO Auto-generated method stub

        //方法执行器。帮我们目标对象执行目标方法
        InvocationHandler h = new InvocationHandler() {
            /**
```

```
             * Object proxy：代理对象；给jdk使用，任何时候都不要动这个对象
             * Method method：当前将要执行的目标对象的方法
             * Object[] args：这个方法调用时外界传入的参数值
             */
            @Override
            public Object invoke(Object proxy, Method method, Object[] args)
                    throws Throwable {

                //System.out.println("这是动态代理将要帮你执行方法...");
                Object result = null;
                try {
                    LogUtils.logStart(method, args);

                    // 利用反射执行目标方法
                    //目标方法执行后的返回值
                    result = method.invoke(calculator, args);
                    LogUtils.logReturn(method, result);
                } catch (Exception e) {
                    LogUtils.logException(method,e);
                }finally{
                    LogUtils.logEnd(method);

                }

                //返回值必须返回出去外界才能拿到真正执行后的返回值
                return result;
            }
        };
        Class<?>[] interfaces = calculator.getClass().getInterfaces();
        ClassLoader loader = calculator.getClass().getClassLoader();

        //Proxy为目标对象创建代理对象；
        Object proxy = Proxy.newProxyInstance(loader, interfaces, h);
        return (Calculator) proxy;
    }

}
```

动态代理：

1）、写起来难；

2）、jdk默认的动态代理，如果目标对象没有实现任何接口，是无法为他创建代理对象的；

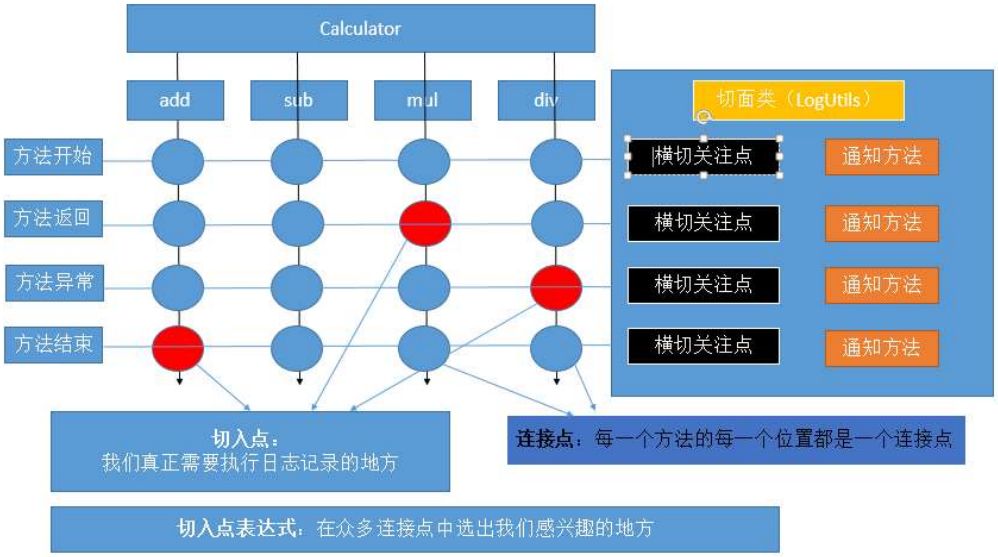
Spring动态代理难；Spring实现了AOP功能；底层就是动态代理；

1）、可以利用Spring一句代码都不写的去创建动态代理；

实现简单，而且没有强制要求目标对象必须实现接口；

将某段代码（日志）动态的切入（不把日志代码写死在业务逻辑方法中）到指定方法（加减乘除）的指定位置（方法的开始、结束、异常。。。）进行运行的这种编程方式（Spring简化了面向切面编程）

AOP专业术语；



AOP使用步骤：

1）、导包；

commons-logging-1.1.3.jar
spring-aop-4.0.0.RELEASE.jar
spring-beans-4.0.0.RELEASE.jar
spring-context-4.0.0.RELEASE.jar
spring-core-4.0.0.RELEASE.jar
spring-expression-4.0.0.RELEASE.jar

Spring支持面向切面编程的包是：
spring-aspects-4.0.0.RELEASE.jar：基础版加强版的面向切面编程（即使目标对象没有实现任何接口也能创建动态代理）
com.springsource.net.sf.cglib-2.2.0.jar
com.springsource.org.aopalliance-1.0.0.jar
com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

2）、写配置；

1）、将目标类和切面类（封装了通知方法（在目标方法执行前后执行的方法））加入到ioc容器中

2）、还应该告诉Spring到底哪个是切面类@Aspect

3）、告诉Spring，切面类里面的每一个方法，都是何时何地运行；

```
@Before("execution(public int com.atguigu.impl.MyMathCalculator.*(int,
int))")
    public static void logStart(){
        System.out.println("【xxx】方法开始执行，用的参数列表【xxx】");
    }

    //想在目标方法正常执行完成之后执行
    @AfterReturning("execution(public int
com.atguigu.impl.MyMathCalculator.*(int, int))")
    public static void logReturn(){
        System.out.println("【xxxx】方法正常执行完成，计算结果是：");
    }
```

```
    //想在目标方法出现异常的时候执行
    @AfterThrowing("execution(public int com.atguigu.impl.MyMathCalculator.*
(int, int))")
    public static void logException() {
        System.out.println("【xxxx】方法执行出现异常了，异常信息是：；这个异常已
经通知测试小组进行排查");
    }

    //想在目标方法结束的时候执行
    @After("execution(public int com.atguigu.impl.MyMathCalculator.*(int,
int))")
    public static void logEnd() {
        System.out.println("【xxx】方法最终结束了");
    }
```

4）、开启基于注解的AOP模式

3）、测试；

```
Calculator bean = ioc.getBean(Calculator.class);
        bean.add(2, 1);
```

AOP使用场景：

1）、AOP加日志保存到数据库；
2）、AOP做权限验证；
3）、AOP做安全检查；
4）、AOP做事务控制；