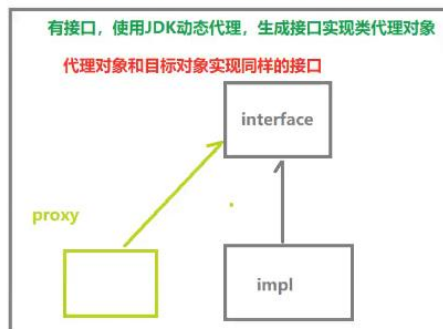


动态代理的两种情况：有接口和没有接口的区别

1、动态代理分类：JDK动态代理和cglib动态代理



2、AspectJ，是AOP框架，Spring只是借用了AspectJ中的注解

- √ 5、面向切面：AOP
 - > 5.1、场景模拟
 - > 5.2、代理模式
 - > 5.3、AOP概念及相关术语
 - √ 5.4、基于注解的AOP
 - 5.4.1、技术说明
 - 5.4.2、准备工作**
 - 5.4.3、创建切面类并配置
 - 5.4.4、各种通知
 - 5.4.5、切入点表达式语法
 - 5.4.6、重用切入点表达式
 - 5.4.7、获取通知的相关信息
 - 5.4.8、环绕通知
 - 5.4.9、切面的优先级
 - √ 5.5、基于XML的AOP
 - 5.5.1、准备工作

步骤：

第一步 引入aop相关依赖

第二步 创建目标资源

- (1) 接口
- (2) 实现类

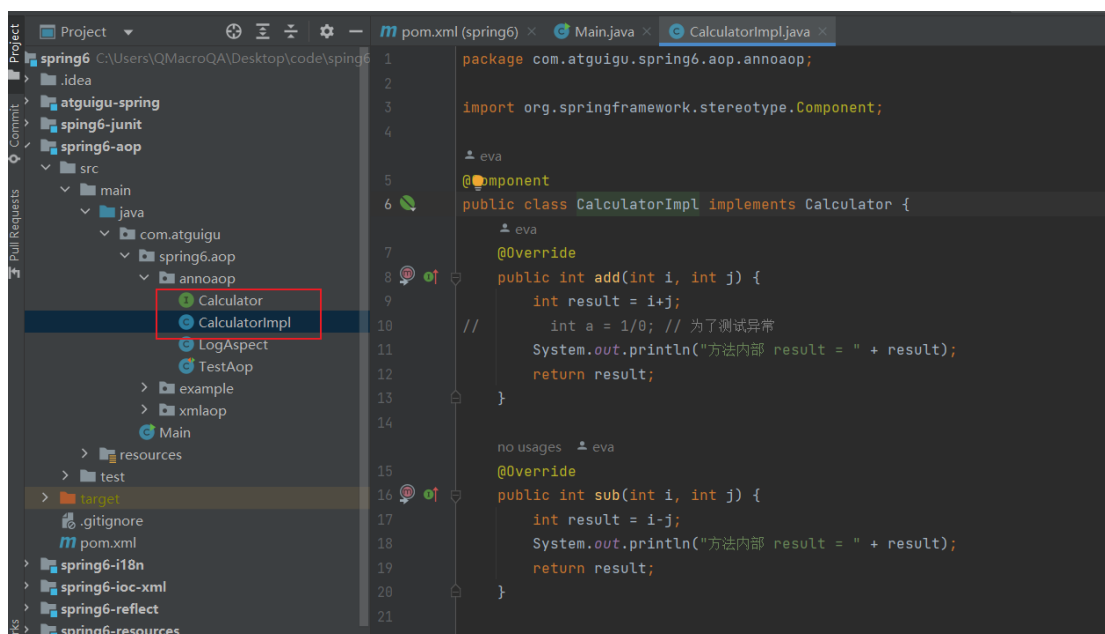
第三步 创建切面类

- (1) 切入点
- (2) 通知类型

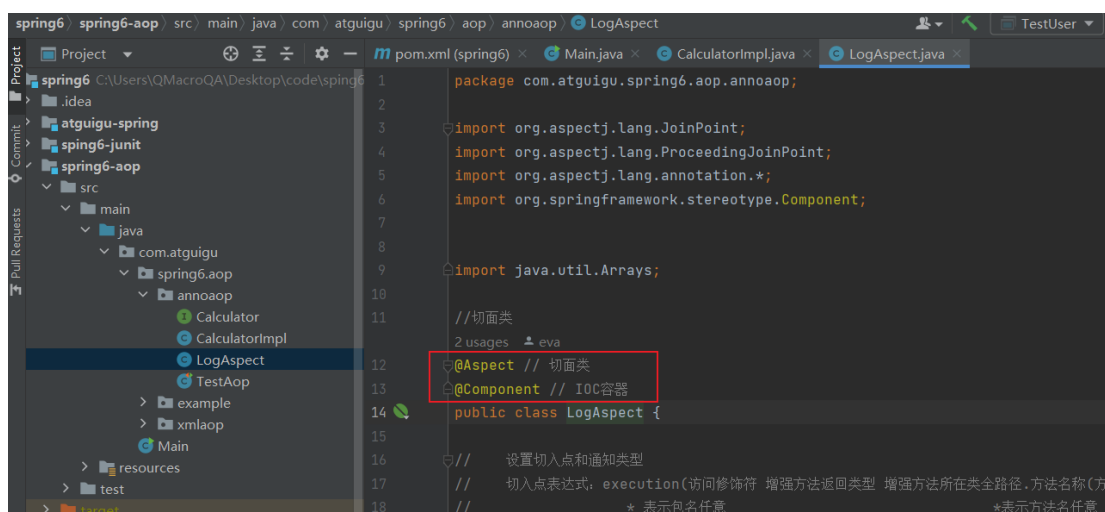
1、引入相关依赖

```
<!-- 实现AOP需要引入的依赖-->
<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>6.1.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-aspects -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>6.0.9</version>
</dependency>
```

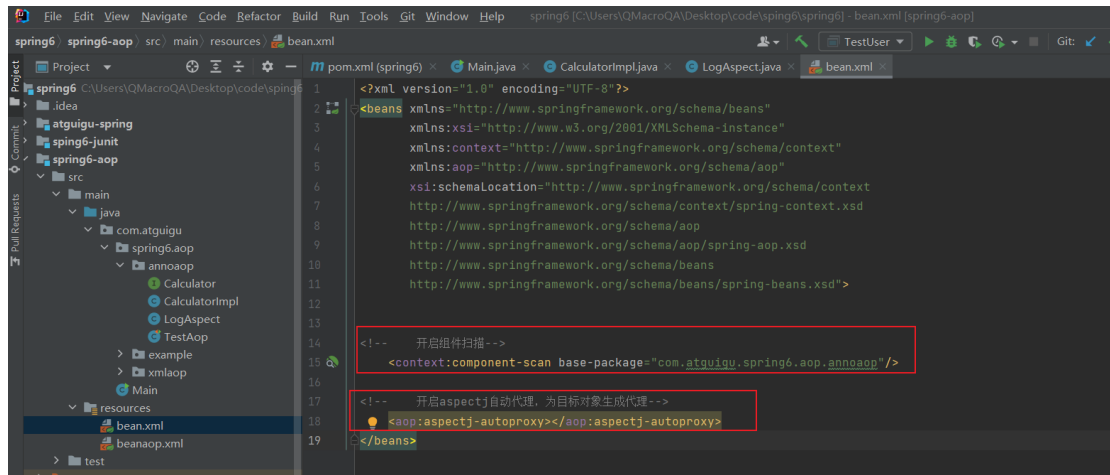
2、创建相关资源



3、创建切面类

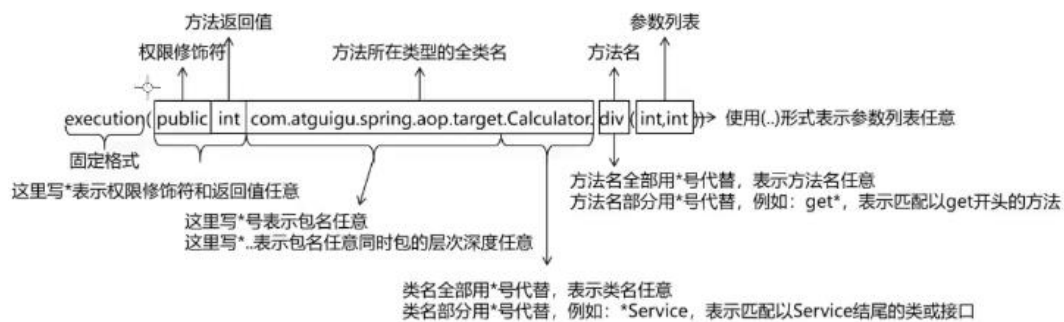


4、开启组件扫描、开启 Aspectk 自动代理



5、设置切入点 and 通知类型

切入点表达式：一定记不住



6、前置



```
// 超前入主:
// 前置 @Before(value="切入点表达式配置切入点")
// @Before(value = "execution(* com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))")
@Before(value = "execution(public int com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))")
public void beforeMethod(JoinPoint joinPoint) {
    String methodName = joinPoint.getSignature().getName();
    Object[] args = joinPoint.getArgs();
    System.out.println("Logger-->前置通知, 方法名称: "+methodName+", 参数: "+args);
}
}
```

7、后置，使用 joinPoint 获取切入函数的相关信息

```
// 后置 @After()
// @After(value = "pointCut()")
@After(value = "com.atguigu.spring6.aop.annoaop.LogAspect.pointCut()")
public void afterMethod(JoinPoint joinPoint){
    String methodName = joinPoint.getSignature().getName();
    System.out.println("Logger-->后置通知, 方法名称: "+methodName);
}
}
```

8、返回，有返回值的通知可以获取，returning

```
// 返回 @AfterReturning
@AfterReturning(value = "execution(* com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))",returning = "result")
public void afterReturning(JoinPoint joinPoint, Object result){
    String methodName = joinPoint.getSignature().getName();
    System.out.println("Logger-->返回通知, 方法名称: "+methodName+", 返回名称: "+result);
}
}
```

8、异常通知，异常情况才执行，可以抛出异常信息

```
// 异常 @AfterThrowing 获取到目标方法异常信息
// 目标方法出现异常，这个通知执行
@AfterThrowing(value = "execution(* com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))",throwing = "ex")
public void afterThrowing(JoinPoint joinPoint, Throwable ex){
    String methodName = joinPoint.getSignature().getName();
    System.out.println("Logger-->异常通知, 方法名称: "+methodName+"异常信息: "+ex);
}
}
```

9、环绕通知

```
// 环绕 @Around()
@Around("execution(* com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))")
public Object around(ProceedingJoinPoint joinPoint){
    String methodName = joinPoint.getSignature().getName();
    Object[] args = joinPoint.getArgs();
    String argString = Arrays.toString(args);
    Object result = null;
    try{
        System.out.println("环绕通知==目标方法出现返回值之前执行");
        // 调用目标方法
        result = joinPoint.proceed();
        System.out.println("环绕通知==目标方法出现返回值之后执行");
    }catch (Throwable throwable){
        throwable.printStackTrace();
        System.out.println("环绕通知==目标方法出现异常执行");
    }finally {
        System.out.println("环绕通知==目标方法执行完毕");
    }
    return result;
}
}
```

10、为了不重复写入切入点表达式，写一个切入点表达式：

```
// 重用切入点表达式
1 usage  👤 eva
@Pointcut(value = "execution(* com.atguigu.spring6.aop.annoaop.CalculatorImpl.*(..))")
public void pointCut(){
}
}
```

使用:

```
// 后置 @After()
    👤 eva
@After(value = "pointCut()")
// @After(value = "com.atguigu.spring6.aop.annoaop.LogAspect.pointCut()")
public void afterMethod(JoinPoint joinPoint){
    String methodName = joinPoint.getSignature().getName();
    System.out.println("Logger-->后置通知, 方法名称: "+methodName);
}
}
```

执行:

```
package com.atguigu.spring6.aop.annoaop;

import ...

    👤 eva
public class TestAop {

    👤 eva
    @Test
    public void test01(){
        ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

        Calculator calculator = context.getBean(Calculator.class);
        calculator.add(2, 3);
    }
}
```

D:\java\jdk-17.0.5\bin\java.exe ...

环绕通知==目标方法之前执行

Logger-->前置通知, 方法名称: add, 参数: [2, 3]

方法内部 result = 5

Logger-->返回通知, 方法名称: add, 返回结果: 5

Logger-->后置通知, 方法名称: add

环绕通知==目标方法返回值之后

环绕通知==目标方法执行完毕执行