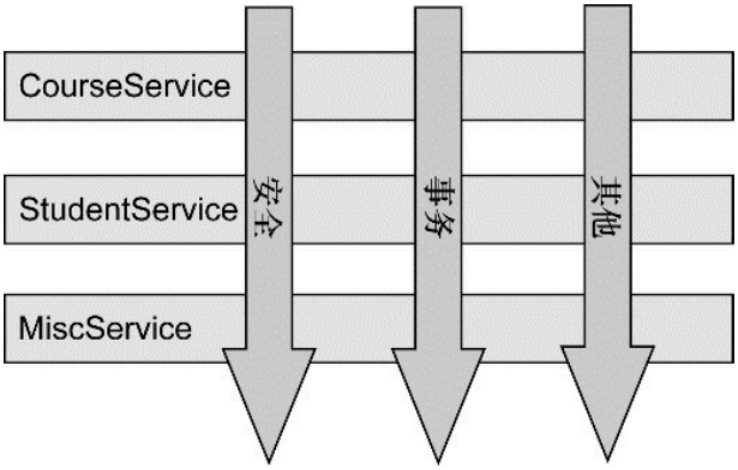
Spring注解驱动开发第25讲——你敢信？面试官竟然让我现场搭建一个AOP测试环境！

什么是AOP?

AOP（Aspect Orient Programming），直译过来就是 面向切面编程 。AOP是一种编程思想，是面向对象编程 （OOP）的一种补充。面向对象编程将程序抽象成各个层次的对象，而面向切面编程是将程序抽象成各个切面。

比如，在《Spring实战（第4版）》中有如下一张图描述了AOP的大体模型 。



切面实现了横切关注点（跨多个应用对象的逻辑）的模块化

从这张图中，我们可以看出：所谓切面，其实就相当于应用对象间的横切点，我们可以将其单独抽象为单独的模块。

总之一句话：AOP是指在程序的运行期间动态地将某段代码切入到指定方法、指定位置进行运行的编程方式。AOP的底层是使用动态代理实现的。

实战案例

（一）导入AOP依赖

要想搭建AOP环境，首先，我们就需要在项目的pom.xml文件中引入AOP的依赖，如下所示。

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>4.3.12.RELEASE</version>
</dependency>
```

AI写代码java运行

其实， Spring AOP 对面向切面编程做了一些简化操作，我们只需要加上几个核心注解，AOP就能工作起来。

（二）定义目标类

在com.meimeixia.aop包下创建一个业务逻辑类，例如MathCalculator，用于处理数学计算上的一些逻辑。比如，我们在MathCalculator类中定义了一个除法操作，返回两个整数类型值相除之后的结果，如下所示。

```
package com.meimeixia.aop;

public class MathCalculator {

    public int div(int i, int j) {
        System.out.println("MathCalculator...div...");
        return i / j;
    }

}
```

AI写代码java运行

现在，我们希望在以上这个业务逻辑类中的除法运算之前，记录一下日志，例如记录一下哪个方法运行了，用的参数是什么，运行结束之后它的返回值又是什么，顺便可以将其打印出来，还有如果运行出异常了，那么就捕获一下异常信息。

或者，你会有这样一个需求，即希望在业务逻辑运行的时候将日志进行打印，而且是在方法运行之前、方法运行结束、方法出现异常等等位置，都希望会有日志打印出来。

（三）定义切面类

在com.meimeixia.aop包下创建一个切面类，例如LogAspects，在该切面类中定义几个打印日志的方法，以这些方法来动态地感知MathCalculator类中的div()方法的运行情况。如果需要切面类来动态地感知目标类方法的运行情况，那么就需要使用Spring AOP中的一系列通知方法了。

AOP中的通知方法及其对应的注解与含义如下：

- 前置通知（对应的注解是@Before）：在目标方法运行之前运行
- 后置通知（对应的注解是@After）：在目标方法运行结束之后运行，**无论目标方法是正常结束还是异常结束都会执行**
- 返回通知（对应的注解是@AfterReturning）：在目标方法正常返回之后运行
- 异常通知（对应的注解是@AfterThrowing）：在目标方法运行出现异常之后运行
- 环绕通知（对应的注解是@Around）：动态代理，我们可以直接手动推进目标方法运行（joinPoint.procced()）

这里我不想一下子就把LogAspects类的完整代码贴出来，虽然可以这样做，但没必要。我的初衷是想向大家阐述这个切面类是如何一点一点写出来的，以及都做了哪些优化。试想一下，你一开始是不是这样写的：

```
package com.meimeixia.aop;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Before;

/**
 * 切面类
 * @author liayun
 *
 */
public class LogAspects {

    // @Before：在目标方法（即div方法）运行之前切入，public int com.meimeixia.aop.MathCalculator.div(int, int)这一串就是切入点表达式，指定在哪个方
    @Before("public int com.meimeixia.aop.MathCalculator.*(..)")
    public void logStart() {
        System.out.println("除法运行......@Before，参数列表是：{}");
    }

    // 在目标方法（即div方法）结束时被调用
    @After("public int com.meimeixia.aop.MathCalculator.*(..)")
    public void logEnd() {
        System.out.println("除法结束......@After");
    }

    // 在目标方法（即div方法）正常返回了，有返回值，被调用
    @AfterReturning("public int com.meimeixia.aop.MathCalculator.*(..)")
    public void logReturn() {
        System.out.println("除法正常返回......@AfterReturning，运行结果是：{}");
    }

    // 在目标方法（即div方法）出现异常，被调用
    @AfterThrowing("public int com.meimeixia.aop.MathCalculator.*(..)")
    public void logException() {
        System.out.println("除法出现异常......异常信息：{}");
    }

}
```

AI写代码java运行

由于我们现在是对MathCalculator类中的div()方法进行切入，因此在每一个通知方法上都写了 `public int com.meimeixia.aop.MathCalculator.*(..)` 这样一串玩意，其实它就是切入点表达式，即指定在哪个方法切入。但是，你不觉得在每一个通知方法上都写上这样一串玩意，很无聊吗？

如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式，就像下面这样。

```
package com.meimeixia.aop;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

/**

```

```
 * 切面类
 * @author liayun
 *
 */
public class LogAspects {

    // 如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式
    @Pointcut("execution(public int com.meimeixia.aop.MathCalculator.*(..))")
    public void pointCut() {}

    /*********代码省略*********/

}
```

AI写代码java运行

pointCut()方法就是抽取出来的一个公共的切入点表达式，其实该方法的方法名随便写啥都行，但是方法体中啥都别写。

那么问题来了，如何在每一个通知方法上引用这个公共的切入点表达式呢？这得分两种情况来讨论，第一种情况，如果是本类引用，那么可以像下面这样写。

```
package com.meimeixia.aop;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

/**
 * 切面类
 * @author liayun
 *
 */
public class LogAspects {

        // 如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式
        @Pointcut("execution(public int com.meimeixia.aop.MathCalculator.*(..))")
        public void pointCut() {}

        // @Before：在目标方法（即div方法）运行之前切入，public int com.meimeixia.aop.MathCalculator.div(int, int)这一串就是切入点表达式，指定在哪
        // @Before("public int com.meimeixia.aop.MathCalculator.*(..)")
        @Before("pointCut()")
        public void logStart() {
            System.out.println("除法运行.....@Before，参数列表是：{}");
        }

        /*********代码省略*********/

}
```

AI写代码java运行

第二种情况，如果是外部类（即其他的切面类）引用，那么就得在通知注解中写方法的全名了，例如，

```
package com.meimeixia.aop;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

/**
 * 切面类
 * @author liayun
 *
 */
public class LogAspects {

    // 如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式
    @Pointcut("execution(public int com.meimeixia.aop.MathCalculator.*(..))")
    public void pointCut() {}
```

```
19
20      // @Before：在目标方法（即div方法）运行之前切入，public int com.meimeixia.aop.MathCalculator.div(int, int)这一串就是切入点表达式，指定在哪个方
21      // @Before("public int com.meimeixia.aop.MathCalculator.*(..)")
22      @Before("pointCut()")
23      public void logStart() {
24          System.out.println("除法运行......@Before，参数列表是：{}");
25      }
26
27      // 在目标方法（即div方法）结束时被调用
28      // @After("pointCut()")
29      @After("com.meimeixia.aop.LogAspects.pointCut()")
30      public void logEnd() {
31          System.out.println("除法结束......@After");
32      }
33
34      // 在目标方法（即div方法）正常返回了，有返回值，被调用
35      @AfterReturning("pointCut()")
36      public void logReturn() {
37          System.out.println("除法正常返回......@AfterReturning，运行结果是：{}");
38      }
39
40      // 在目标方法（即div方法）出现异常，被调用
41      @AfterThrowing("pointCut()")
42      public void logException() {
43          System.out.println("除法出现异常......异常信息：{}");
44      }
45
46  }
```

AI写代码java运行

最后，千万别忘了一点，那就是必须告诉Spring哪个类是切面类，要做到这一点很简单，只需要给切面类上加上一个@Aspect注解即可。

```
1   package com.meimeixia.aop;
2
3   import org.aspectj.lang.annotation.After;
4   import org.aspectj.lang.annotation.AfterReturning;
5   import org.aspectj.lang.annotation.AfterThrowing;
6   import org.aspectj.lang.annotation.Aspect;
7   import org.aspectj.lang.annotation.Before;
8   import org.aspectj.lang.annotation.Pointcut;
9
10  /**
11   * 切面类
12   *
13   * @Aspect：告诉Spring当前类是一个切面类，而不是一些其他普通的类
14   * @author liayun
15   *
16   */
17  @Aspect
18  public class LogAspects {
19
20      // 如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式
21      @Pointcut("execution(public int com.meimeixia.aop.MathCalculator.*(..))")
22      public void pointCut() {}
23
24      // @Before：在目标方法（即div方法）运行之前切入，public int com.meimeixia.aop.MathCalculator.div(int, int)这一串就是切入点表达式，指定在哪个方
25      // @Before("public int com.meimeixia.aop.MathCalculator.*(..)")
26      @Before("pointCut()")
27      public void logStart() {
28          System.out.println("除法运行......@Before，参数列表是：{}");
29      }
30
31      // 在目标方法（即div方法）结束时被调用
32      // @After("pointCut()")
33      @After("com.meimeixia.aop.LogAspects.pointCut()")
34      public void logEnd() {
35          System.out.println("除法结束......@After");
36      }
37
38      // 在目标方法（即div方法）正常返回了，有返回值，被调用
39      @AfterReturning("pointCut()")
40      public void logReturn() {
41          System.out.println("除法正常返回......@AfterReturning，运行结果是：{}");
42      }
43
```

```
    // 在目标方法（即div方法）出现异常，被调用
    @AfterThrowing("pointCut()")
    public void logException() {
        System.out.println("除法出现异常......异常信息：{}");
    }

}
```

AI写代码java运行

至此，切面类的完整代码我们就写出来了。

（四）将目标类和切面类加入到IOC容器

在com.meimeixia.config包中，新建一个配置类，例如MainConfigOfAOP，并使用@Configuration注解标注这是一个Spring的配置类，同时使用@EnableAspectJAutoProxy注解开启基于注解的AOP模式。在MainConfigOfAOP配置类中，使用@Bean注解将业务逻辑类（目标方法所在类）和切面类都加入到IOC容器中，如下所示。

```
package com.meimeixia.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

import com.meimeixia.aop.LogAspects;
import com.meimeixia.aop.MathCalculator;

/**
 * AOP：面向切面编程，其底层就是动态代理
 *      指在程序运行期间动态地将某段代码切入到指定方法指定位置进行运行的编程方式。
 *
 * @author liayun
 *
 */
@EnableAspectJAutoProxy
@Configuration
public class MainConfigOfAOP {

    // 将业务逻辑类（目标方法所在类）加入到容器中
    @Bean
    public MathCalculator calculator() {
        return new MathCalculator();
    }

    // 将切面类加入到容器中
    @Bean
    public LogAspects logAspects() {
        return new LogAspects();
    }

}
```

AI写代码java运行

一定不要忘了给MainConfigOfAOP配置类标注@EnableAspectJAutoProxy注解哟！在Spring中，未来会有很多的@EnableXxx注解，它们的作用都是开启某一项功能，来替换我们以前的那些配置文件。

（五）测试

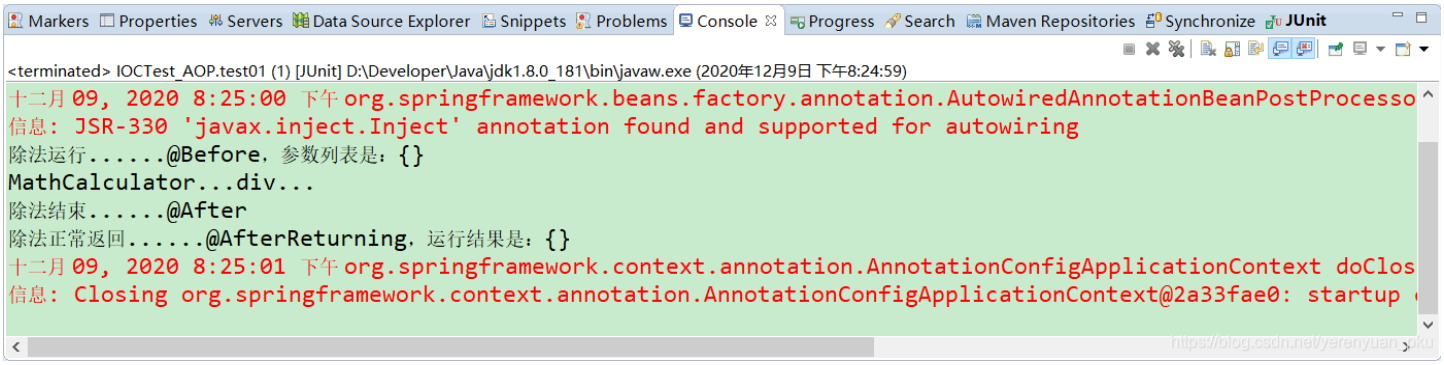
首先，在com.meimeixia.test包中创建一个单元测试类，例如IOCTest_AOP，并在该测试类中创建一个test01()方法，如下所示。

```
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.aop.MathCalculator;
import com.meimeixia.config.MainConfigOfAOP;

public class IOCTest_AOP {


```

```
    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfAOP.class);

        // 不要自己创建这个对象
        // MathCalculator mathCalculator = new MathCalculator();
        // mathCalculator.div(1, 1);

        // 我们要使用Spring容器中的组件
        MathCalculator mathCalculator = applicationContext.getBean(MathCalculator.class);
        mathCalculator.div(1, 1);

        // 关闭容器
        applicationContext.close();
    }

}
```

AI写代码java运行

然后，运行以上IOCTest_AOP类中的test01()方法，输出的结果信息如下所示。



可以看到，执行了切面类中的方法，并打印出了相关信息。**但是并没有打印参数列表和运行结果。**

如果需要在切面类中打印出参数列表和运行结果，那么该怎么办呢？别急，我们继续往下看。

要想打印出参数列表和运行结果，就需要对LogAspects切面类中的方法进行优化，优化后的结果如下所示。

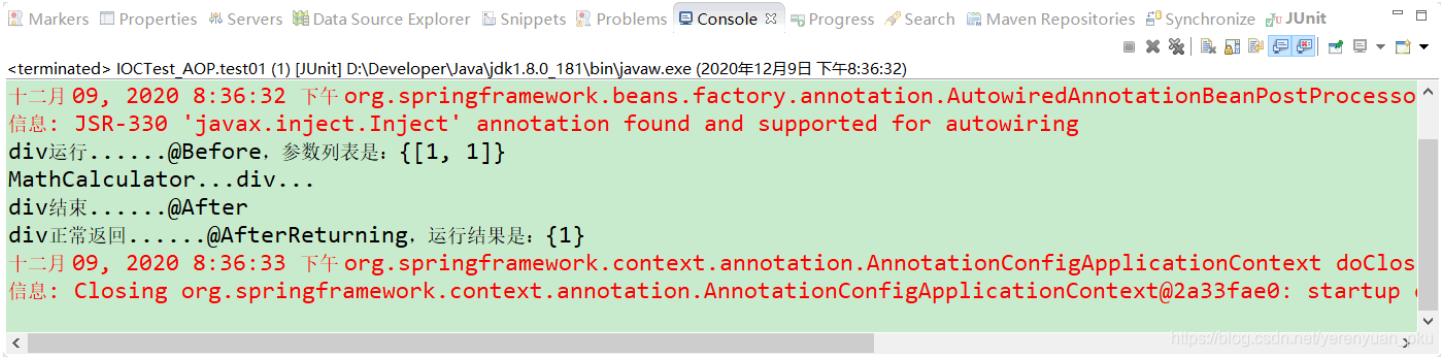
```
package com.meimeixia.aop;

import java.util.Arrays;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

/**
 * 切面类
 *
 * @Aspect：告诉Spring当前类是一个切面类，而不是一些其他普通的类
 * @author liayun
 *
 */
@Aspect
public class LogAspects {

    // 如果切入点表达式都一样的情况下，那么我们可以抽取出一个公共的切入点表达式
    // 1. 本类引用
    // 2. 如果是外部类，即其他的切面类引用，那就在这@After("...")写的是方法的全名，而我们就要把切入点写在这儿@Pointcut("...")
    @Pointcut("execution(public int com.meimeixia.aop.MathCalculator.*(..))")
    public void pointCut() {}

    // @Before：在目标方法（即div方法）运行之前切入，public int com.meimeixia.aop.MathCalculator.div(int, int)这一串就是切入点表达式，指定在哪个方
    // @Before("public int com.meimeixia.aop.MathCalculator.*(..)")
    @Before("pointCut()")
    public void logStart(JoinPoint joinPoint) {
        // System.out.println("除法运行......@Before，参数列表是：{}");

```

```
        Object[] args = joinPoint.getArgs(); // 拿到参数列表，即目标方法运行需要的参数列表
        System.out.println(joinPoint.getSignature().getName() + "运行......@Before，参数列表是：{" + Arrays.asList(args) + "}");
    }

    // 在目标方法（即div方法）结束时被调用
    // @After("pointCut()")
    @After("com.meimeixia.aop.LogAspects.pointCut()")
    public void logEnd(JoinPoint joinPoint) {
        // System.out.println("除法结束......@After");

        System.out.println(joinPoint.getSignature().getName() + "结束......@After");
    }

    // 在目标方法（即div方法）正常返回了，有返回值，被调用
    // @AfterReturning("pointCut()")
    @AfterReturning(value="pointCut()", returning="result") // returning来指定我们这个方法的参数谁来封装返回值
    /*
     * 如果方法正常返回，我们还想拿返回值，那么返回值又应该怎么拿呢？
     */
    public void logReturn(JoinPoint joinPoint, Object result) { // 一定要注意：JoinPoint这个参数要写，一定不能写到后面，它必须出现在参数列表的第一
        // System.out.println("除法正常返回......@AfterReturning，运行结果是：{}");

        System.out.println(joinPoint.getSignature().getName() + "正常返回......@AfterReturning，运行结果是：{" + result + "}");
    }

    // 在目标方法（即div方法）出现异常，被调用
    @AfterThrowing("pointCut()")
    public void logException() {
        System.out.println("除法出现异常......异常信息：{}");
    }

}
```

AI写代码java运行

这里，需要注意的是，JoinPoint参数一定要放在参数列表的第一位，否则Spring是无法识别的，那自然就会报错了。

此时，我们再次运行IOCTest_AOP类中的test01()方法，输出的结果信息如下所示。



如果目标方法运行时出现了异常，而我们又想拿到这个异常信息，那么该怎么办呢？只须对LogAspects切面类中的logException()方法进行优化即可，优化后的结果如下所示。

```
// 在目标方法（即div方法）出现异常，被调用
// @AfterThrowing("pointCut()")
@AfterThrowing(value="pointCut()", throwing="exception")
public void logException(JoinPoint joinPoint, Exception exception) {
    // System.out.println("除法出现异常......异常信息：{}");

    System.out.println(joinPoint.getSignature().getName() + "出现异常......异常信息：{" + exception + "}");
}
```

AI写代码java运行

可以看到，JoinPoint参数是放在了参数列表的第一位。

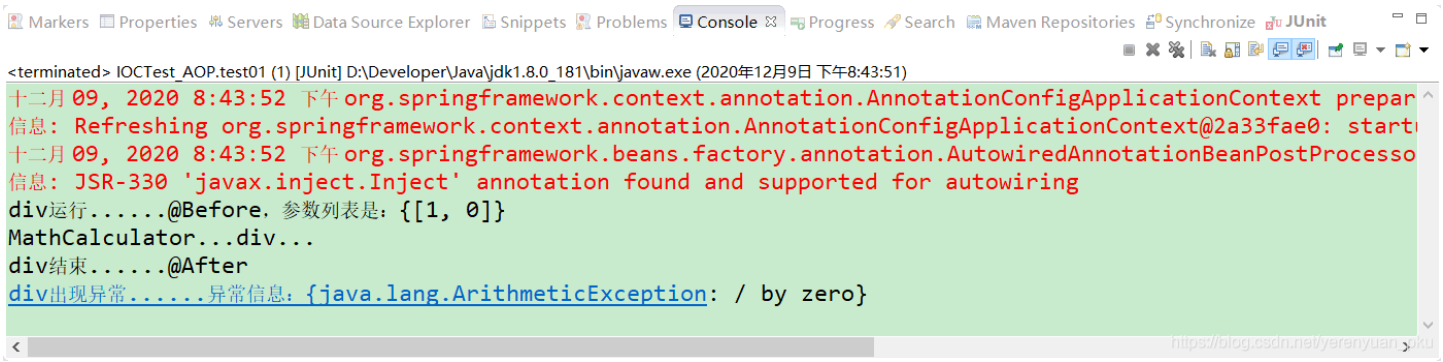
接下来，我们就在MathCalculator类的div()方法中模拟抛出一个除零异常，来测试下异常情况，如下所示。

```
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

```

```java
import com.meimeixia.aop.MathCalculator;
import com.meimeixia.config.MainConfigOfAOP;

public class IOCTest_AOP {

    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfAOP.class);

        // 不要自己创建这个对象
        // MathCalculator mathCalculator = new MathCalculator();
        // mathCalculator.div(1, 1);

        // 我们要使用Spring容器中的组件
        MathCalculator mathCalculator = applicationContext.getBean(MathCalculator.class);
        mathCalculator.div(1, 0);

        // 关闭容器
        applicationContext.close();
    }

}
```

AI写代码java运行

此时，我们运行以上test01()方法，输出的结果信息如下所示。



可以看到，正确的输出了切面中打印的信息，包括除零异常的信息。

至此，我们的AOP测试环境就搭建成功了。

小结

搭建AOP测试环境时，虽然步骤繁多，但是我们只要牢牢记住以下三点，就会无往而不利了。

1. 将切面类和业务逻辑组件（目标方法所在类）都加入到容器中，并且要告诉Spring哪个类是切面类（标注了@Aspect注解的那个类）。
2. 在切面类上的每个通知方法上标注通知注解，告诉Spring何时何地运行，当然最主要的是要写好切入点表达式，这个切入点表达式可以参照官方文档来写。
3. **开启基于注解的AOP模式，即加上@EnableAspectJAutoProxy注解，这是最关键的一点。**