

由于最近项目中经常使用到AOP，在写这篇博客之前，我对于AspectJ 框架 aop 和 Spring aop 的区别以及概念都是比较模糊的，希望读者可以通过这篇博客，深入的了解 AspectJ 框架是如何进行切面处理的。

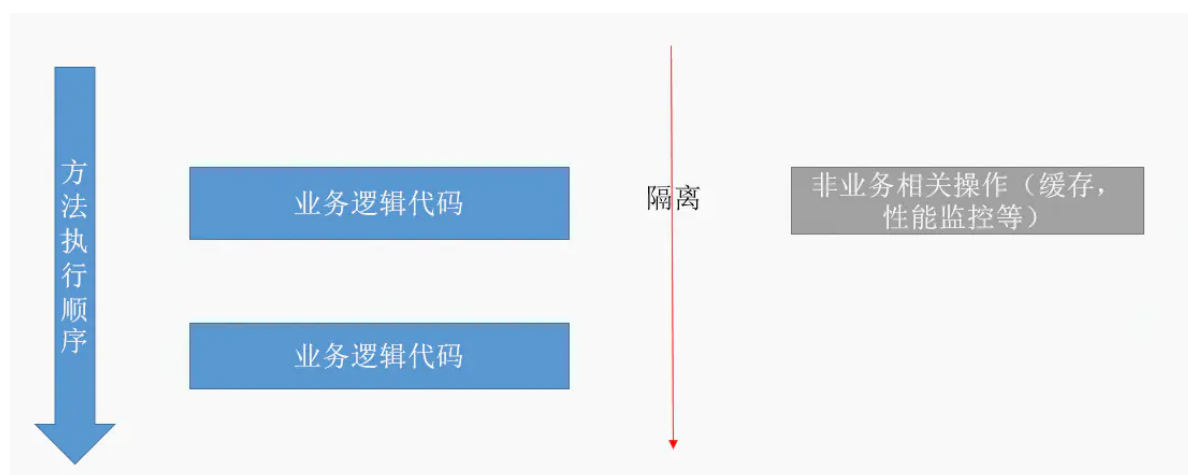


图 1

如图1所示，在编写业务代码过程中，从业务逻辑角度划分，可以分为业务相关代码和非业务相关代码。业务代码是功能实现的基础，但是非业务代码在整个过程中也扮演者一个非常重要的角色，例如：添加缓存，方法性能监控等等。有没有什么办法可以将两者相隔离呢？

AOP：全称 Aspect Oriented Programming，面向切面编程。

AOP 技术为业务逻辑代码与非业务逻辑代码的隔离提供了一种有效的解决方案。



图二

如上图所示，使用 AOP 技术，可以将业务逻辑代码与非业务逻辑代码相互隔离，互不影响。

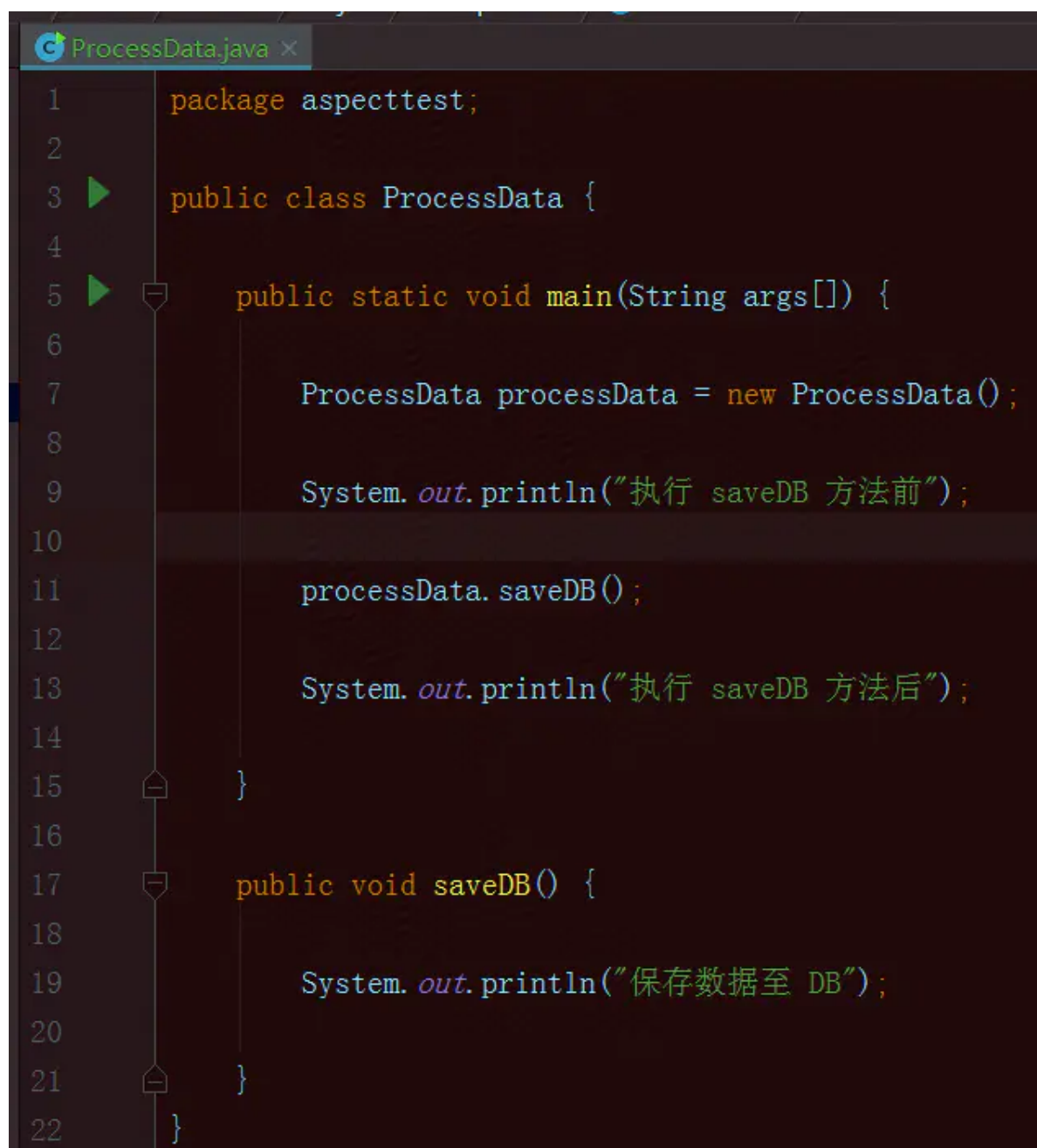
AOP 技术如何实现？

最早的实现 AOP 技术使用到的是 AspectJ 框架，实现的原理为静态织入。在编译时期，优先编译非业务相关代码，然后在编译业务代码时将非业务代码织入，以达到横切的效果（本文主要讲解的 AspectJ 框架）。在讲解实例之前，先普及两个概念

1、pointcut：切点，即需要使用 AOP 的点。

2、advice：通知，即想要在切点处进行哪些具体的操作。

下面介绍 AspectJ 实现 AOP 的两种形式，编码以及使用注解的方式。



```
1 package aspecttest;
2
3 public class ProcessData {
4
5     public static void main(String args[]) {
6
7         ProcessData processData = new ProcessData();
8
9         System.out.println("执行 saveDB 方法前");
10
11         processData.saveDB();
12
13         System.out.println("执行 saveDB 方法后");
14
15     }
16
17     public void saveDB() {
18
19         System.out.println("保存数据至 DB");
20
21     }
22 }
```

图三

如图三所示，ProcessData 类中有一个 saveDB() 保存数据的方法，在执行该方法前，需要打印一下请求的日志，请求完成之后，也需要打印日志。打印日志和业务逻辑其实是没有任何关联的，但是又是必须需要的，写在业务逻辑代码中，会显得很臃肿，这时就可以使用 AOP 技术。

1、使用编码方式使用 AspectJ 框架

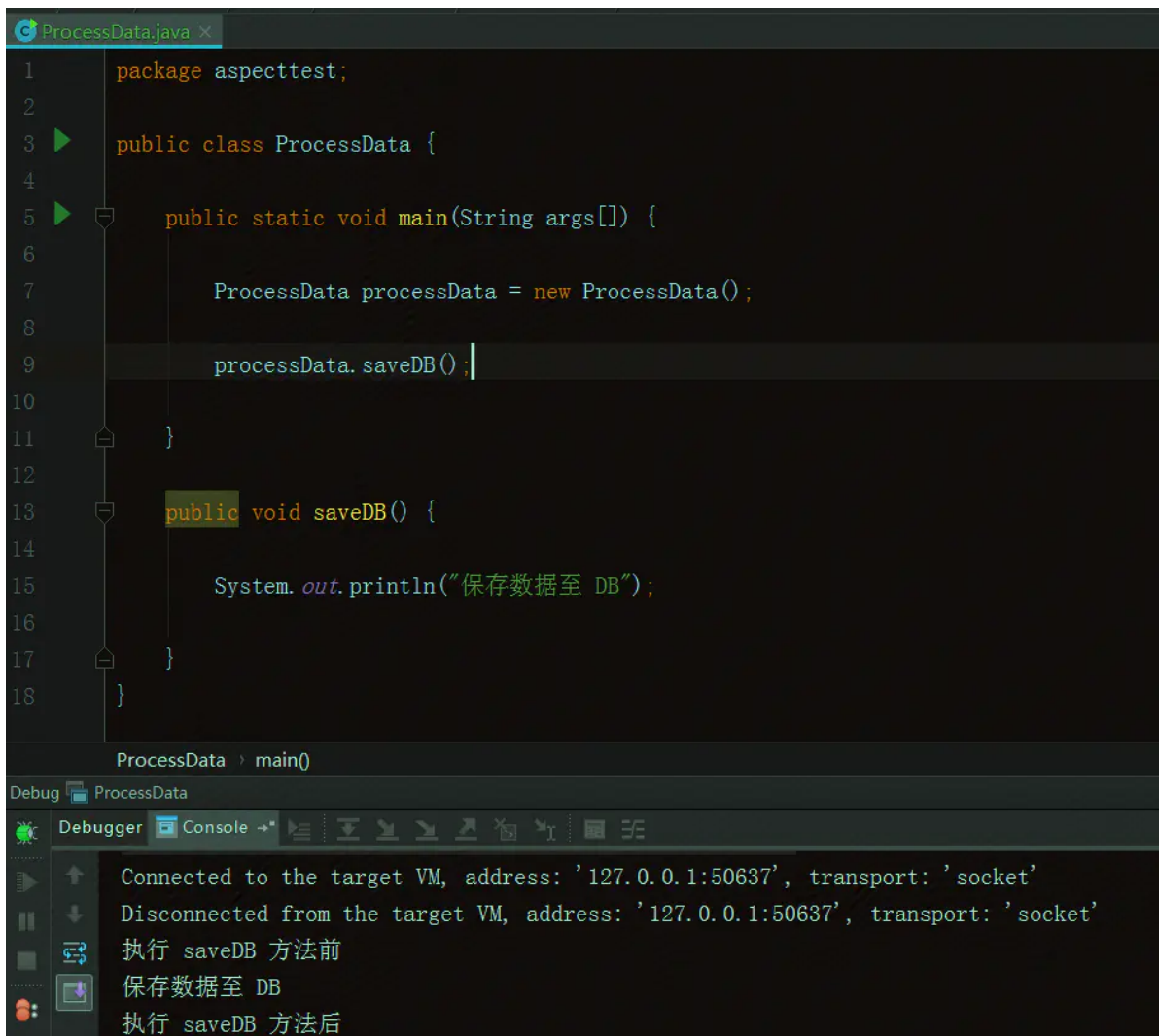
```
LogAspect.aj x
1 package aspecttest;
2
3 public aspect LogAspect {
4
5     /**
6      * 定义切点, 日志记录切点
7      */
8     pointcut recordLog():call(* aspecttest.ProcessData.saveDB());
9
10    /**
11     * 定义前置通知!
12     */
13    before():recordLog(){
14        System.out.println("执行 saveDB 方法前");
15    }
16
17    /**
18     * 定义后置通知
19     */
20    after():recordLog(){
21        System.out.println("执行 saveDB 方法后");
22    }
23
24 }
25
```

切点：执行 saveBD 时进行横切操作

通知：执行 saveDB 前打印日志

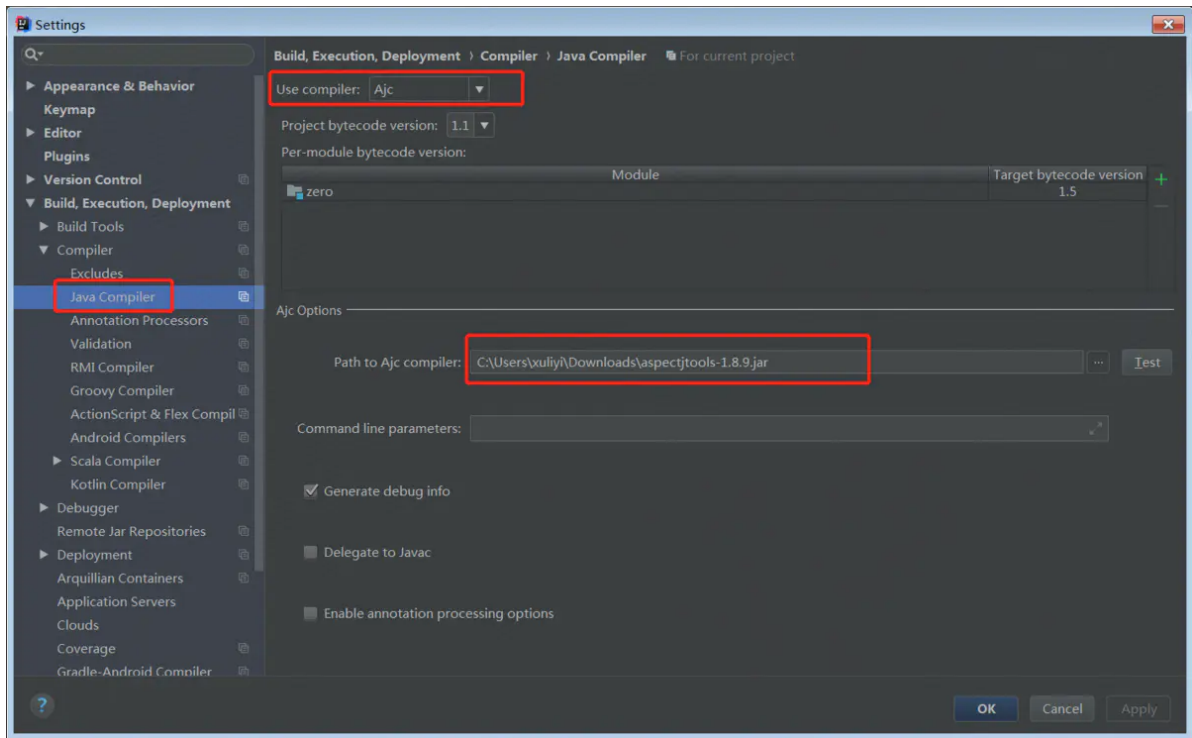
通知：执行 saveDB 后打印日志

图四



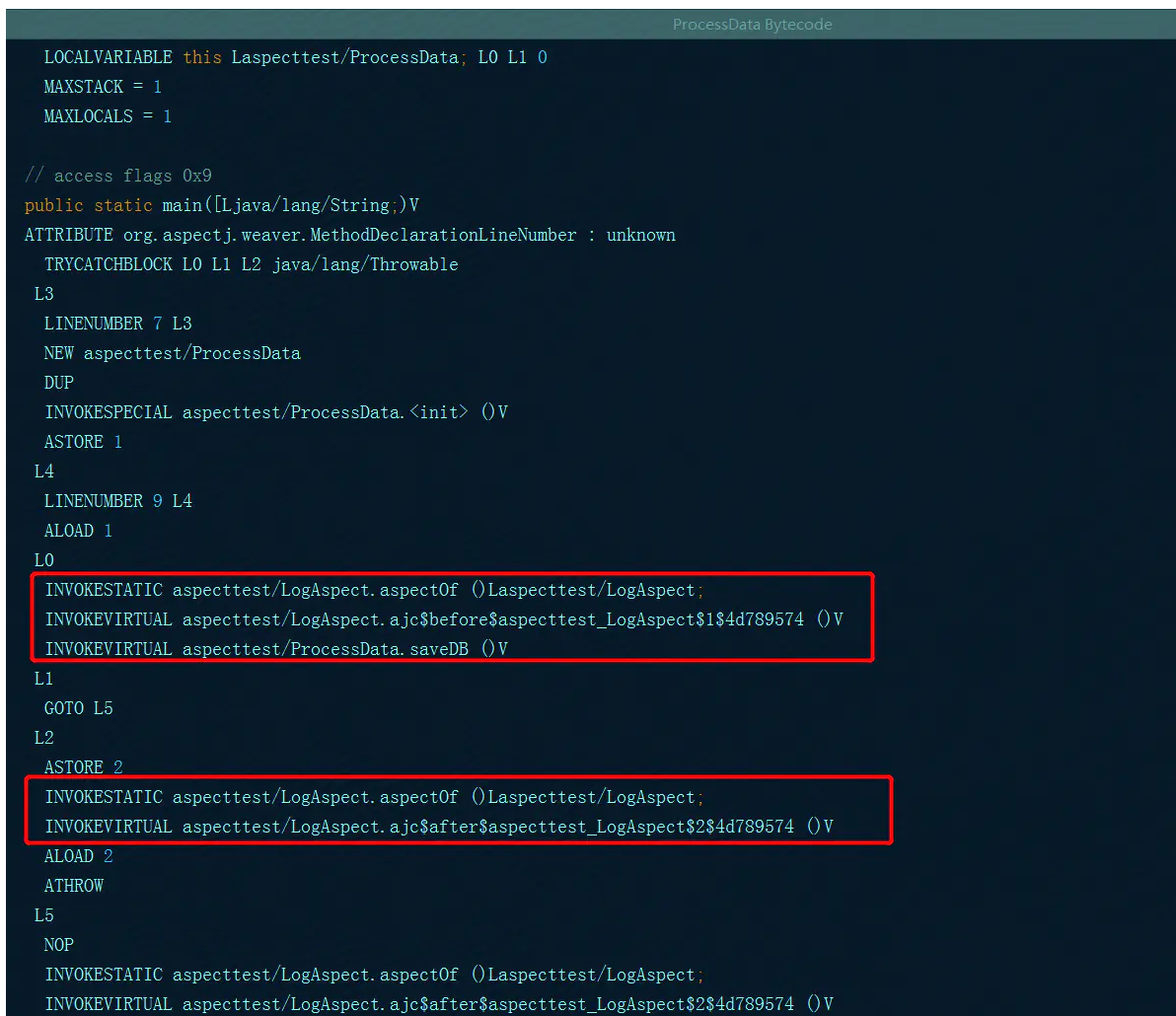
图五

如上图所示，图四将非业务逻辑代码抽取出来，进行切面处理。图五只进行业务逻辑处理。打印结果和不使用AOP的结果是一致的。需要注意的点是，AspectJ 框架的编译器不是 `javac`，而是 `Ajc`，`Ajc` 可以认为是 `javac` 的增强版本，在使用 Aspect 进行编码测试 AOP 时，需要手动的修改编译器，IDEA 修改方法：File->setting->Build,Execution,Development->Compiler->Java Compiler use compiler 改为 `Ajc`。如下图所示：



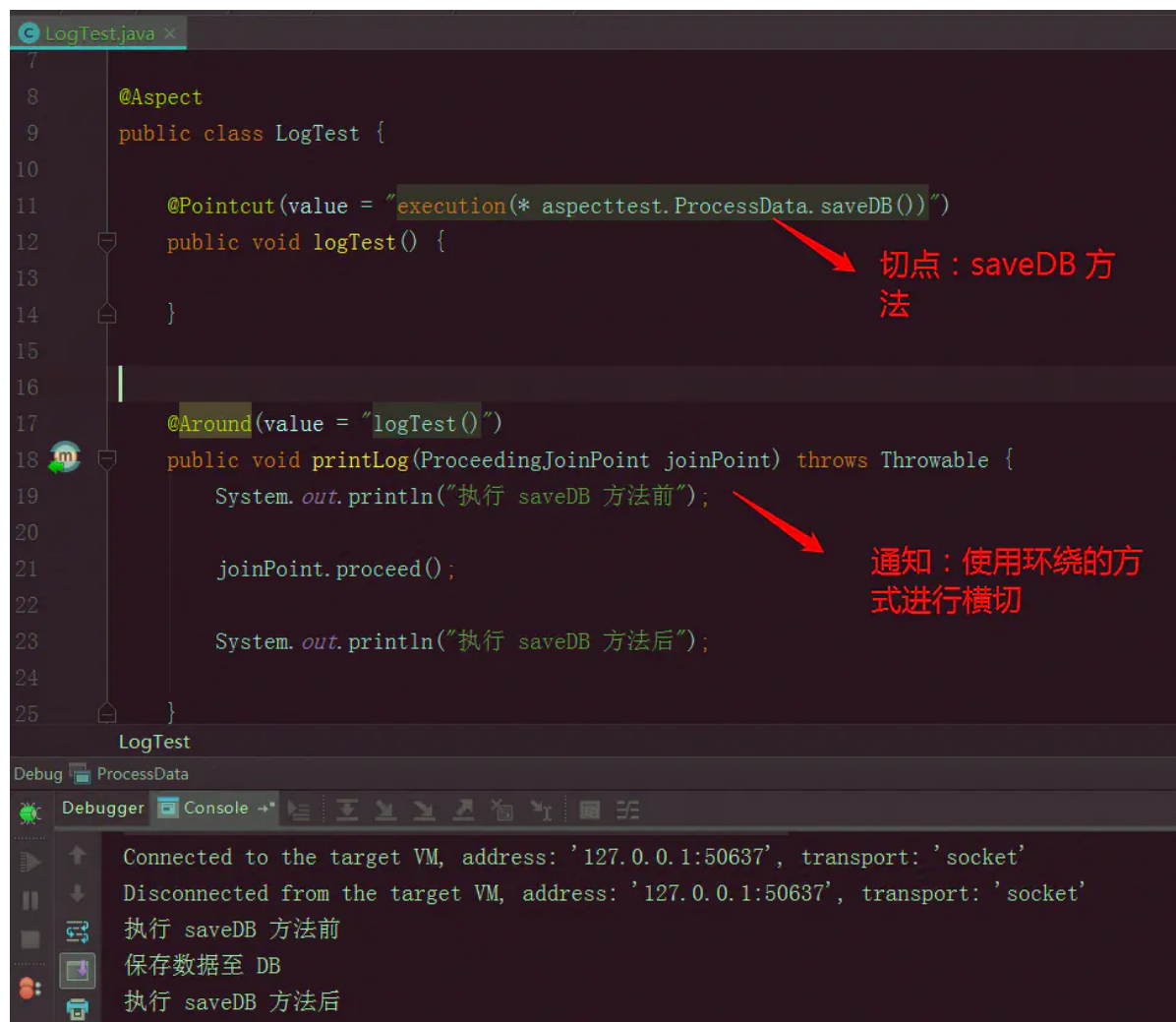
图六

使用 AspectJ 框架进行 AOP 之后，反编译代码，业务逻辑代码被非业务逻辑代码静态织入了，如下图所示：



图七

2、使用注解方式使用 AspectJ 框架



图八

需要注意的是，使用 `@Aspect` 注解进行横切的时候，也是需要配置编译器，使用的是 `Ajc`。

下一节将讲述的是Spring AOP 中是如何兼容 Aspect 框架的