AOP是一个多义词，请在下列义项上选择浏览（共5个义项） 添加

- 面向切面编程
- 面向行为编程
- 葡萄酒
- 支付宝开放平台
- AOP的其他含义

AOP（面向切面编程）

在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是OOP的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

AOP在其他领域也有其他含义。[1]

中文名	面向切面编程	属　性	软件开发技术
外文名	Aspect Oriented Programming	衍生范型	函数式编程
简　称	AOP	又　意	葡萄酒

目录

名称含义

Aspect Oriented Programming（AOP）是较为热门的一个话题。AOP，国内大致译作“面向切面编程”。

“面向切面编程”,这样的名字并不是非常容易理解，且容易产生一些误导。笔者不止一次听到类似“OOP/OOD11即将落伍，AOP是新一代软件开发方式”这样的发言。显然，发言者并没有理解AOP的含义。Aspect，没错，的确是“方面”的意思。不过，华语传统语义中的“方面”，大多数情况下指的是一件事情的不同维度、或者说不同角度上的特性，比如我们常说：“这件事情要从几个方面来看待”，往往意思是：需要从不同的角度来看待同一个事物。这里的“方面”，指的是事物的外在特性在不同观察角度下的体现。而在AOP中，Aspect的含义，可能更多的理解为“切面”比较合适。所以笔者更倾向于“面向切面编程”的译法。

可以通过预编译方式和运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术。AOP实际是GoF设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦,提高代码的灵活性和可扩展性，AOP可以说也是这种目标的一种实现。

在Spring中提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务（例如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。

主要功能

日志记录，性能统计，安全控制，事务处理，异常处理等等

主要意图

将日志记录，性能统计，安全控制，事务处理，异常处理等代码从业务逻辑代码中划分出来，通过对这些行为的分离，我们希望可以将它们独立到非指导业务逻辑的方法中，进而改变这些行为的时候不影响业务逻辑的代码。

AOP/OOP

区分

AOP、OOP在字面上虽然非常类似，但却是面向不同领域的两种设计思想。OOP（面向对象编程）针对业务处理过程的实

体及其属性和行为进行抽象封装，以获得更加清晰高效的逻辑单元划分。

而AOP则是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果。这两种设计思想在目标上有着本质的差异。

上面的陈述可能过于理论化，举个简单的例子，对于“雇员”这样一个业务实体进行封装，自然是OOP/OOD的任务，我们可以为其建立一个“Employee”类，并将“雇员”相关的属性和行为封装其中。而用AOP设计思想对“雇员”进行封装将无从谈起。

同样，对于“权限检查”这一动作片断进行划分，则是AOP的目标领域。而通过OOD/OOP对一个动作进行封装，则有点不伦不类。

换而言之，OOD/OOP面向名词领域，AOP面向动词领域。

关系

很多人在初次接触 AOP 的时候可能会说，AOP 能做到的，一个定义良好的 OOP 的接口也一样能够做到，我想这个观点是值得商榷的。AOP和定义良好的 OOP 的接口可以说都是用来解决并且实现需求中的横切问题的方法。但是对于 OOP 中的接口来说，它仍然需要我们在相应的模块中去调用该接口中相关的方法，这是 OOP 所无法避免的，并且一旦接口不得不进行修改的时候，所有事情会变得一团糟；AOP 则不会这样，你只需要修改相应的 Aspect，再重新编织（weave）即可。 当然，AOP 也绝对不会代替 OOP。核心的需求仍然会由 OOP 来加以实现，而 AOP 将会和 OOP 整合起来，以此之长，补彼之短。

应用举例

假设在一个应用系统中，有一个共享的数据必须被并发同时访问，首先，将这个数据封装在数据对象中，称为Data Class，同时，将有多个访问类，专门用于在同一时刻访问这同一个数据对象。

为了完成上述并发访问同一资源的功能，需要引入锁Lock的概念，也就是说，某个时刻，当有一个访问类访问这个数据对象时，这个数据对象必须上锁Locked，用完后就立即解锁unLocked，再供其它访问类访问。

使用传统的编程习惯，我们会创建一个抽象类，所有的访问类继承这个抽象父类，如下：

```
abstract  class  Worker  {
        abstract  void  locked();
        abstract  void  accessDataObject();
        abstract  void  unlocked();
}
```

accessDataObject()方法需要有“锁”状态之类的相关代码。

Java只提供了单继承，因此具体访问类只能继承这个父类，如果具体访问类还要继承其它父类，比如另外一个如Worker的父类，将无法方便实现。

重用被打折扣，具体访问类因为也包含“锁”状态之类的相关代码，只能被重用在相关有“锁”的场