面向方面程序设计技术

Programming with AOP

凌 晨 （中国科学院研究生院(本部) 北京 100071）
陈芳莉 （解放军国防大学 17 队 北京 100091）

摘要：本文介绍了面向方面编程的基本概念，比较了它与面向过程、面向对象编程的不同，然后结合语言规范 AspectJ 给出了一个面向方面程序的例子。本文还阐述了面向方面技术的优点。

关键词：面向方面编程 方面 关注点 横切关注点 切入点 通知

1 引言

面向对象技术将问题领域中的“名词”转化为软件系统中的“对象”，抽象出对象的类型(类)，通过类的继承体系实现代码的重用。借此，面向对象技术在取代面向过程技术上获得了巨大成功。然而，问题域并不是全部由“名词”组成，比如日志之类的服务，它们不是与特定的对象关联，而是与特定的应用关联。在面向对象编程中，如果创建每个类的超类，然后将日志管理的代码写入这些超类中，那么大量的修改无疑会增大出错的概率，同时这也是一件非常枯燥的事情。面向方面技术应运而生。面向方面的编程提供了一种叫做“方面(aspect)”的模块化单元，其中可包含“横切关注点”，这些横切关注点也是模块化的，允许程序人员把工作重心放在业务逻辑上，而不是对象组织上。

2 三种编程技术体系结构比较

2.1 面向过程技术

好的代码带给程序员好处的最终体现就是代码复用。在面向过程的编程中，如果某些代码日后不能满足需求而需要修改，此时程序员面对是一个个的函数，程序员的接触面也最广：是所有的函数的所有的代码，但同时程序员由于修改了以前的代码而导致程序出现错误的概率也大大增加了。如图 1 所示。

2.2 面向对象技术

面向对象的技术因为使用了“继承”技术得以很好的解决面向过程编程中出现的问题，如图 2 所示。此时程序员面对的是一个个的类，继承基类可以实现原来所有的逻辑，如果需要增加新功能，可以重载基类，但此时重载，程序员要么在基类函数调用前插入一段代码，要么在基类函数全部执行完毕后执行某段新增代码，如果两者都不能够满足要求，程序员就不得不重写整个函数，哪怕这一段新的函数种仅仅增加了一行代码。显然，此时就会回到面向过程编程中遇到的难题，不得不让程序员冒很大的风险。在不重写基类函数的条件下，程序员在面向对象编程中对代码的接触面缩小到了函数的起始点和结束点。代码的安全性提高的同时缩小了程序员接触代码的接触面。

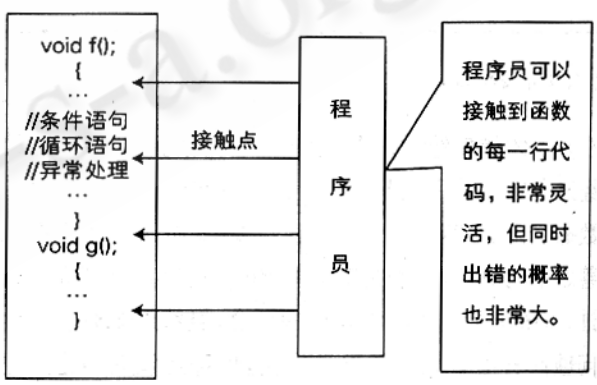

图 1 面向过程技术程序员的接触面

另外，新的需求向面向对象技术提出了挑战。即上面提出的类似日志之类的服务，此时要求在多个类中同时增加类似的代码(函数)，就图 2 体系结构而言，要求程序员同时方便的接触多个类的某些函数。面向对象的这种对函数完全的继承此时就力不从心了，除非愿意冒很大的风险修改多个类，而且这种的修

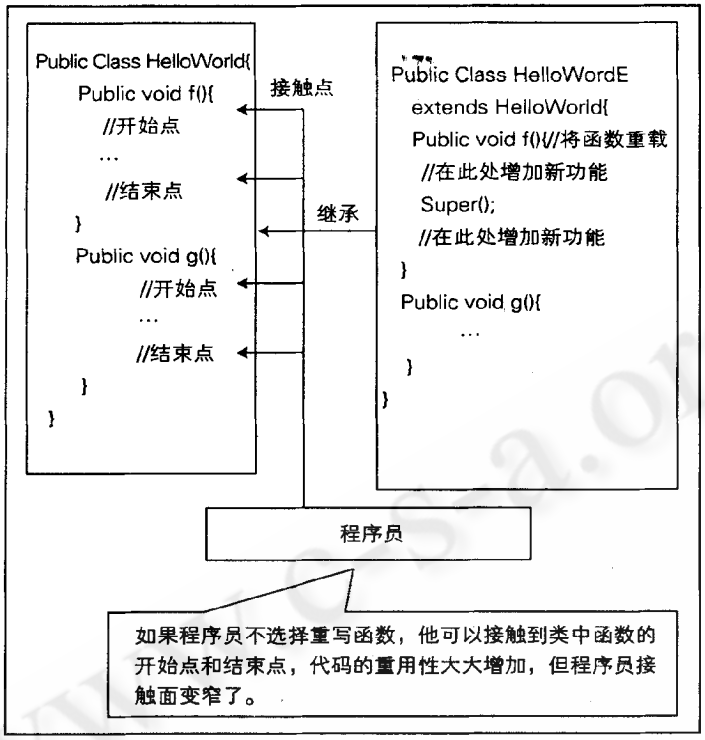

图 2　面向对象技术程序员的接触面

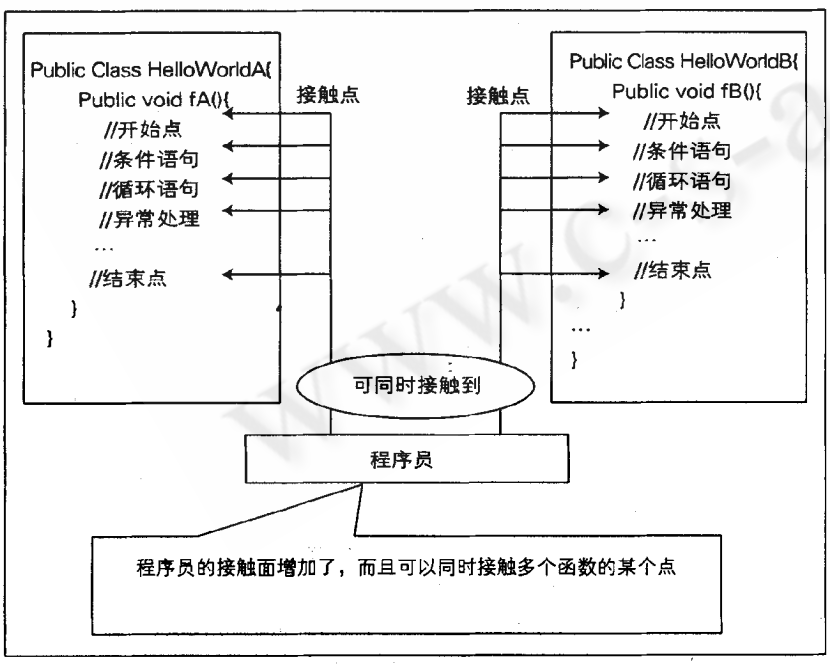

图 3　面向方面技术程序员的接触面

改几乎完全一致。

2.3 面向方面技术

2.3.1 体系结构

如图3 所示，程序员在原来面向对象技术中的接触面增加了很多，灵活性更加大，而且通过方面可以将多个毫不相关的类耦合起来，这就解决上面曾提出的日志问题。面向对象取代面向过程使程序员的对代码接触点缩小到了有限的几个，面向方面技术再次将程序员对代码的接触面拓宽。面向方面技术不是要取代面向对象技术，而是面向对象的很好补充。

2.3.2 关注点

在面向方面编程中，一个关注点就是一个特定的目的，实现某种特定功能的模块，常在多个模块中出现，也叫横切关注点（cross-cutting concerns）。我们可以把一个复杂的系统看作由多个关注点组成，这写关注点可能会包括几个方面，比如日志、调试信息、业务逻辑等，以及编程技术人员从技术上考虑为了方便处理的关注点。关注点的可以让程序人员接触到了不止一个类函数的内部（当然也可以接触到函数的开始点和结束点）。因此，关注点能够将互不相关的类耦合起来。

2.3.3 AOP 的好处

AOP 可帮助我们解决代码混乱和代码分散所带来的问题，同时还有一些其他好处：

（1）系统容易扩展。由于方面可以让程序员接触到多个函数的多个接触面，所以在原来的基础上很容易通过加入新的方面来扩展原来的功能。

（2）升级更加容易。使用 AOP，设计师可以推迟为将来的需求作决定，实现当前的紧要需求，而把日后更新的需求使用方面与以前的需求耦合起来。

（3）更好的代码复用。新的

编程技术的最终目的都是为了代码复用，AOP 把每个方面实现独立为模块，模块之间的耦合是松散的，更容易理解和维护，代码的冗余也更小，也更容易复用。

3 通过 AspectJ 更好的了解 AOP

AspectJ 是施乐公司帕洛阿尔托研究中心的一个免费产品，它是一个语言规范，也是一个 AOP 语言实现。AspectJ 语言构造是从 Java 语言扩展而来的，所有合法的 Java 程序都是合法的 AspectJ 程序。AspectJ 有自己的独特语言构造。

3.1 方面

方面是 AspectJ 的模块单元，将切入点和通知集中到一起，其作用象 Jave 中的类。方面可以标记其自身和任何关注点为抽象的。抽象的关注点与抽象类相似，允许你把细节实现推迟到派生方面里。

3.2 连接点

程序执行过程中的点，比如调用某个类里的特定方法的地方。

3.3 切入点

用来指定、组合所需连接点并搜集连接点上特定上下文信息的语言构造。

3.4 通知

在符合特定条件的情况下执行的代码。例如，一个通知可能会在执行到一个连接点之前在数据库中写入一条纪录。

3.5 织入

现在的面向方面技术还没有独立的编译器，只能借助于某种已有的语言扩展而来。AspectJ 是从 Java 语言扩展而来的，它通过特定的编译器将面向方面特有的语法转为符合 Java 字节码规范的. class 文件，这就是所谓的织入。

3.6 AspectJ 程序例子

有了以上的概念，给出一个 AspectJ 程序的例子。

下面类中有一个字符串输出的方法：

```
//HelloWorld. java
public class HelloWorld {
  public static void ShowMessage( String MStr) {
    //执行某些操作
    System. out. println( MStr) ;
  }
  public static void ShowMessageS( String MStr1,
MStr2) {
    //执行某些操作
    System. out. println ( MStr1 + " + " +
MStr2) ;
  }
}
以下开始使用面向方面的技术
//LogAspect. java
public aspect LogAspect {
  pointcut CallLog( ) :
  call( public static void HelloWorld. ShowMessage
*( . . ) ) ;
  before( ) : CallLog( ) {
    //开始写日志
    System. out. println( "Log Enter") ;
  }
  after( ) : CallLog( ) {
    //开始写日志
    Sytem. out. println( "Log Exit") ;
  }
}
```

在 LogAspect. java 文件中，声明了一个 LogAspect 方面，其中定义了一个 CallLog( ) 切入点，记住切入点仅仅是 AspectJ 中的一个语言构造，它说明了所关注的函数为类 HelloWorld 中所有以 ShowMessage 开头的函数，然后此方面定义了两个通知，一个是在 CallLog 切入点前调用进入日志功能，一个是在 CallLog 切入点后调用日志退出功能。

4 结束语

面向方面技术能解决许多其它编程技术不容易解决的问题，现在越来越多的人开始了解并使用面向方面的技术。这种技术也会越来越完善，相信不久的将来会在编译器的内核中增加对面向方面技术的支持。

参考文献

1 Michael J. Yuan and Norman Richards, Lightweight Aspect – Oriented Programming, 2003.

2 Thinking in Java Second Edition, Bruce Eckel, 2002.

3 Ramnivas 通过 AspectJ 更好的了解 AOP，电子工业出版社，2002。