文章编号 1000-5013(2005)04-0428-04

以责任为中心的面向对象观

陈叶旺 余金山

(华侨大学信息科学与工程学院, 福建 泉州 362021)

摘要 面向对象技术的传统的观点只停留在对象层面上, 以对象为中心, 机械地理解面向对象. 狭隘的认识往往导致在软件分析设计阶段存在严重的缺陷, 到后期, 常常不得不对原来的方案做大的调整甚至丢弃. 文中以责任为中心的面向对象观, 突破传统观点的狭隘范围, 更贴近人类的思维方式. 它从宏观上把握着软件开发的各阶段, 从责任的角度重新认识面向对象, 扩展面向对象的内涵.

关键词 面向对象, 面向责任, 抽象概念, 说明规范

中图分类号 TP 311.52 文献标识码 A

面向对象从对结构化方法的批判中诞生, 吸取其基本思想和主要优点, 模拟人类习惯的思维方式. 它按人类认识世界的方法开发软件, 适应了软件规模和数量以惊人的速度急剧膨胀, 提高了软件系统的稳定性、可修改性和重用性[1]. 但传统的观点还存在缺陷. 以责任为中心的面向对象观, 也是从对传统观点的批判中产生的, 它以责任为中心, 超越对象自身层面, 把对面向对象的理解上升到更抽象的层次. 新观点要求从宏观环境出发, 先明确对象责任, 再依据责任确定对象能力. 最后, 根据对象自身来决定实现方式. 新观点能克服传统观念的很多缺陷.

1 传统的面向对象观及缺点

较早前结构化方法的定律是: 程序=(程序)+(数据结构). 面向对象将其进化成: 对象=(算法+数据结构), 程序=(对象+对象+……)[2]. 这强调了算法和数据结构的整体性, 将两者捆绑成一个不可分割的有机体——对象, 程序则是一个个对象有机组合. 这是传统面向对象观的基石, 其核心概念如表 1 所示. 如此, 传统的面向对象就是: 先确定主体数据, 再动作, 将两者封装成独立统一的不可分割的实体, 对外以标准的接口提供服务. 这种从对象自身实现角度上观察问题, 存在很大的局限性.

表 1 核心概念表

概念	传统定义
对象	带数据及施加在数据之上的操作的有机实体. 是"聪明"的数据[3]
类	类是对象的分门别类的抽象描述, 是对象的模板
接口	类中定义为公共的数据和操作集
继承	表达的是对象类之间的相交关系, 它使得某类对象可以得到另外一种类对象的特征和能力
多态	它描述的是同一个消息在不同对象中不同的行为方式
组合	是将其他类对象包含于对象自身内部
封装	封装是数据和行为的隐藏, 不让外界知道具体细节

1.1 概念间关系模糊

传统上认为封装、继承、多态是面向对象的 3 个基本特征. 程序设计是建立类的等级过程, 过分依赖继承, 这是其局限性的突出表现. 继承机制能共享资源, 提高重用度, 本质上是封装的一种手段. 传统的概念往往忽略"组合"的作用, 孤立了封装、继承、多态、组合之间的内在联系. 这几个概念的传统定义也没能说明它们的本质.

收稿日期 2005-01-23

作者简介 陈叶旺(19-), 男, 硕士研究生, 主要从事软件工程的研究. E-mail: viaphone@hqu.edu.cn

1.2　过分强调对象能力

依据传统观点, 寻找对象就是在需求说明中寻找名词, 确定对象行为就是寻觅动词. 然后, 拼凑成一个对象, 突出对象的能力, 而不管它需不需这种能力. 这很难从宏观上所把握对象.

1.3　冗余严重

在结构化设计方法中, 软件开发往往会有大量的数据冗余. 有人认为, 用面向对象的方法能自然而然解决这个问题. 但实际上, 传统思维并没做到, 通常存在类冗余和方法冗余. (1) 类冗余. 开发过程中, 需求经常会有变动. 当需求增加时希望类的数量按线性增长, 但过多依赖继承, 却使得类爆炸性扩张. (2) 方法冗余. 由于过多的继承, 在不同的类中往往包含大量相似的操作和数据.

1.4　紧耦合性

传统的观点认为, 方法隶属于对象, 是对象的功能体现, 对象间的消息传递是不同实体之间的交互协作, 因而对象之间是松耦合. 然而, 这并不是事实. 因为传统观点常常会导致冗余, 使得对象之间功能相互交错, 大大增加了对象之间的耦合性, 因而对某个类的更动时, 常常是牵一发而动全身 .

1.5　低内聚性

由于只从对象自身角度上看问题, 很难把握核心功能, 其重点常被分散于不同的对象中. 一旦核心功能变动, 这些对象不得不都作调整. 这些缺点原本就存在于结构化方法中, 面向对象能解决这些问题, 但没有理解其精神, 问题依然得不到解决. 应该指出的是, 仅用类、继承和对象来架构软件的方法不是面向对象.

2　以责任为中心的面向对象观

人们考虑问题, 往往从具体问题层面和抽象的概念层面两个不同角度出发. 如表达颜色时, 用“红、黄、蓝 ……” 来表示. 颜色是抽象的, 而“红色的、黄色的、蓝色的杯子” 却是很具体的东西, 能被直接理解. 当要形容其他物体时, 只要将名词换掉就行. 传统面向对象观的最大缺点是, 总从如何实现角度出发, 非要将“红色的杯子” 当作一个名词, “黄色的杯子” 又是一个名词. 这种做法很不明智, 不是人们认识世界的方法, 也不是面向对象. 因此, 有必要从更高的抽象层次来重新认识面向对象, 以责任为中心的对象观就是属于这种抽象层次. 下面我们以这种新的观点来对面向对象的核心概念进行详细讨论.

2.1　对象

对象是有明确责任的实体, 其属性方法与责任不可分割. 不是对象责任域范围内的行为和属性数据就应该让其他对象承担. 以前面的例子来看, 就应将颜色和杯子分开, 这两者是有不同责任的实体. 用颜色自身属性来说明物体, 是它的责任和义务. 颜色不只是专为杯子对象服务, 也可用来形容其他东西. 可以看出, 颜色是抽象的实体, 独立于杯子而存在, 当要决定具体颜色时, 就属于实现方式范畴. 所以, 应把抽象概念和实现方式区分开. 这才是面向对象的精髓, 而传统的观点恰恰混淆了两者的界线.

2.2　类

类是同类型对象的抽象描述, 而对象是有明确责任的实体. 那么, 在类中所描述的属性、行为都与对象的责任息息相关. 从责任角度上看, 类是用属性和行为来对有其自身责任的对象实体的抽象. 实体类对对象的行为有具体的规定, 抽象类则是说明性地描述, 而没有付诸实现. 具体实现交由派生类去完成 . 抽象类只从宏观上规定了任务, 成为对象责任的抽象说明规范.

2.3　继承

继承是类与类之间一种特殊的关系, 子类继承父类的资源, 继承简化了代码重用, 但在继承资源的同时, 也继承了相应的责任和义务. 继承是“是一种” 关系, 即子类对象可以认为“是一种” 父类对象[4]. 比如, 小学生、中学生、大学生都继承于学生类. 小学生、中学生可以说是一种学生, 大学生也是一种学生, 他们公共的东西在学生类中. 本质上说, 这些公共的东西是他们都要承担了责任, 那就是“学习”. 应该注意的是, 滥用继承是相当危险的, 而过多的继承层次往往是致命的. 其缺点有如下 3 点. (1) 将一部分实现细节暴露给子类, 破坏了封装性. (2) 父类做更改时, 子类也不得不会随之调整 . (3) 从父类继承的实现方式, 不能在运行期间进行改变.

2.4　接口

按传统的观点,接口是类中定义为公共的数据和操作集,是在给外界透露信息.目前的面向对象对象语言,一般都将接口与类中分离开,对接口进行单独定义.与纯抽象类相似,它只说明性地定义了一组行为规范,没有付诸具体实现,而是让遵守接口的对象去完成.接口像一份合同,规定了对象所必须提供的服务.与继承不同,接口是声明继承,而不是实现继承,更具弹性.(1) 为类对象架构模型减肥,有效阻止由于过多继承所引发的类的爆炸性增长.(2) 接口能避免继承缺点和陷阱,使其得到简化.(3) 在对象交互时,允许双方互不知对方类型及继承架构的情形下彼此沟通,避免了耦合性.(4) 接口机制符合软件服务的观念,只要客户端遵照服务接口规格,就可以使用特定的服务,服务的背后就与客户端无关,从而拥有跨平台的能力.例如在IT界,对于生产同类产品的不同产商来说,他们之间可能互不相知.但只要都遵守公共标准,透过这份标准,就能知道对方能做什么,需要什么,同类产品也就可以相互替换.因此,接口是对象责任的抽象说明规范,是提供的服务的契约.与抽象类一样,接口属于具体实现和抽象概念之间的过渡层,即说明规范层.

2.5 面向对象的三层次

以责任为中心面向对象观,可以从3个层面看待对象.(1) 抽象概念层.对象是有一组明确责任的实体.(2) 说明规范层.对象是一组能被其他对象调用行为规范集合.(3) 具体实现层.对象是算法+数据结构,是具体的代码.责任是抽象的,是人的意识,说明规范是责任的描述,是要让外界能理解的责任说明,具体实现则是对象责任的最终完成.例如,在Windows操作系统中有鼠标指针(MouseCursor)对象.很显然它的责任有,在屏幕上获取位置、移动、变换图标样式等.为了说明责任,就要定义一组相应的行为规范,如表2所示.

这些都不涉及具体实现,只描述了它的功能,据此可以定义一个抽象基类或接口,如

```
Class MouseCursor {
Virtual Point getLocation ();
Virtual Void moveTo ();
Virtual Boolean changeIcon (); }
```

表2 抽象说明

责任	行为规范(接口)
在屏幕上获取位置	getLocation()
移动	moveTo()
变换图标样式	changeIcon()

当要具体实现它时,就与平台有关,如表3所示.Windows下的实现方法就和Linux不同,在Windows下实现,就定义WinMouseCursor;在Linux下实现,就定义XWinMouseCursor.如此就将MouseCursor和具体平台脱离,实现了松耦合与可延展性.3个层次的关系是抽象概念+说明规范=接口或抽象类,以及说明规范+具体实现=实体类.

表3 具体实现

Windows	Linux
Class WinMouseCursor::Public MouseCursor	Class XWinMouseCursor::Public MouseCursor
{ Point getLocation (){//具体代码};	{ Point getLocation (){//具体代码};
Void moveTo (){//…};	Void moveTo (){//…};
Boolean changeIcon (){// …}; }	Boolean changeIcon (){// …}; }

2.6 多态

多态性是在继承关系下的一种表象.传统的观念认为:多态性是相同标志的函数或运算符在不同的场合的不同行为方式.这些不同行为方式所能完成的任务,是在抽象概念层和说明规范层中定义的,它属于具体实现层所考虑的内容.准确地说,多态性是多种不同实现方式,其实质是用不同的方法来履行相同的义务.多态使得不同的对象,以不同的方式应对变化的外部世界,面向对象最为灵活的地方就在于多态,同时它却有极强的欺骗性.不能相信多态,在没确定对象具体身份前,不知道它会怎么做.

2.7 组合

组合是将其他对象包含于自身内部,从而获得新功能,亦可通过组合将责任委托给其他对象,也称之为"聚合".包容者和被容者之间是一种"同生共死"的关系,是一个完整的统一体.所以,组合实际上是在招兵买马与委托责任.组合的威力要比继承大,在选择用继承还是组合时,应优先考虑组合而不是继承[5].相对于继承,它有如下3个优点.(1) 捍卫了封装性,实现的细节隐藏于被包容的对象内部,没有透露给它的容器.(2) 在发挥继承的作用的同时,能减少类的层次,阻止类的爆炸性增长,有效减少冗

余. (3) 具体的实现方式可以在运行期内更改, 更为灵活.

2. 8 封装

从字面上理解, 封装就是将东西包起来, 不让外界知道实际内容. 因而, 传统观点认为, 封装是数据和行为集合放在一起, 形成一个统一的不可分割的实体, 这种理解并不全面. 再例如, 有公司为一些厂商做代理, 向某校大学生做产品推销, 而不论他们是专科生、本科生还是研究生, 如图 1 所示. 若一个本科生购买产品, 对于代理公司来说, 只要知道他是该校大学生就行, 学生的类别被隐藏于“大学生”这个层面上. 从而认为, 继承是一种隐藏. 若要学生出示证件, 他们出示的证件肯定不同, 这就是多态的表现. 可以说, 在这里具体的行为方式被隐藏了. 所以, 多态是另一种隐藏. 从大学生的角度上看, 代理公司是卖产品商家, 能提供产品就行, 但实际上这些产品是其他厂商生产, 学生们不知道, 厂商也被隐藏了. 那么, 组合也是一种隐藏. 这例子说明, 封装是任何形式的信息隐藏, 使得问题得到简化, 是面向对象符合自然规律之所在. 面向对象的最大威力就在于封装.

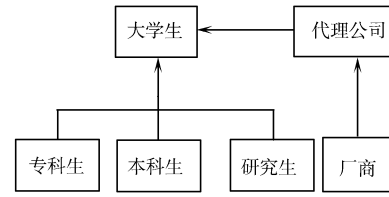

图 1　代理销售

3　结论

相对于传统的观点, 以责任为中心的面向对象观是: 先确定责任, 再封装实现方式, 对外提供标准的接口以履行义务. 其精髓在于抽象和实现的分离, 威力在于封装, 灵活性在于多态. 其优点有如下 3 点. (1) 意义深刻, 关系清晰. 在获得新的内涵的同时, 概念之间关系变得清晰. 封装才是面向对象的威力, 组合、继承、多态都是封装的手段. 组合优于继承, 接口优于继承, 过多继承不可取. (2) 对象责任明确. 以责任区分对象, 按需求进行封装, 从宏观上就把握住方向, 具体的实现细节又可以做到灵活多变. (3) 减少冗余, 实现松耦合、高内聚. 本文从责任的角度, 以更高、更广的视野重新认识面向对象, 对几个核心概念重新定义, 扩展它们的内涵、理顺关系、指明其深刻意义, 把对面向对象的理解提升到新高度. 面向对象还处于不断的发展过程中, 还存在不足, 有待更进一步的发展.

参　考　文　献

1　余金山. 软件工程中以几个热点问题[J]. 华侨大学学报(自然科学版) , 2004, 25(1) : 5~ 9

2　钱　能. C++ 程序设计教程[M]. 北京: 清华大学出版社, 1999. 234~ 235

3　Shalloway A, Trott J R. Design patterns explained: A new perspective on object-oriented design [M]. 北京: 中国电力出版社, 2003. 15~ 16

4　迈耶斯. Effective C++ (中文版)[M]. 侯　捷译. 北京: 中国电力出版社, 2003. 153~ 211

5　Gamma E, Helm R, Johnson R, et al. Design patterns: Elements of reusable object-oriented software[M]. 北京: 机械工业出版社, 2003. 18~ 20

Responsibility-Centered Outlook of Object-Oriented

Cheng Yewang　　Yu Jinshan

(College of Information Science and Engineering, Huaqiao University, 362021, Quanzhou, China)

Abstract　Conventional viewpoint of object-oriented stays at the aspect of objectand takes object as center. It comprehends object-oriented mechanically. The narrowrecognition often results in serious defect in design and analysis, so that theoriginal plan has to be revised greatly, even abandoned. The object-oriented outlook with the responsibility-center outlook of breaks through its limitation. It is closer to the way in which human being thinking of. It graspsmacroscopically all stages of software development . This paper discussesobject-oriented from the perspective of responsibility, and then extend itsconnotation.

Keywords　object-oriented, responsibility-oriented abstract concept, specification, implementation