深入浅出 OOD（一）

撰文/透明

有物昆成，先天地生。萧呵！谬呵！独立而不改，可以为天地母。吾未知其名，字之曰道。吾强为之名曰大，大曰逝，逝曰远，远曰反。道大，天大，地大，王亦大。

——《道德经》，第二十五章

软件不软

从 60 年代的软件危机，到今天传统软件工程方法处处碰壁的处境，都说明一个问题：软件不软（Software is Hard）[Martin, 95]。说实话，软件是一块硬骨头，真正开发过软件的人都会有此感觉。一个应用总是包含无数错综复杂的细节，而软件开发者则要把所有这些细节都组织起来，使之形成一个可以正常运转的程序，这实在不是一件简单的事。

为什么会这样？举个例子来说：用户可以很轻松地说“我要一个字处理软件”，他觉得“字处理软件”这样一个概念是再清楚不过的了，根本不需要更多的描述；而真正开发一个字处理软件却是一件困难无比的事情——我曾经亲眼看到开发字处理软件的人们是怎样受尽折磨的。为什么表述一个概念很容易，而实现一个概念很困难？因为人们太擅长抽象、太擅长剥离细节问题了。

软件不软，症结就在这里：用户很容易抽象地表达自己的需求，而这种抽象却很难转化为程序代码。软件不软，因为很简单的设想也需要大量的时间去实现；软件不软，因为满足用户的需求和期望实在太困难；软件不软，因为软件太容易让用户产生幻想。

而另一方面，计算机硬件技术却在飞速发展。从几十年前神秘的庞然大物，到现在随身携带的移动芯片；从每秒数千次运算到每秒上百亿次运算。当软件开发者们还在寻找能让软件开发生产力提高一个数量级的“银弹”[Brooks, 95]时，硬件开发的生产力早已提升了百倍千倍。这是为什么？

硬件工程师们能够如此高效，是因为他们都很懒惰。他们永远恪守“不要去重新发明轮子”的古训，他们尽量利用别人的成果。你看到有硬件工程师自己设计拨码开关的吗？你看到有硬件工程师自己设计低通滤波电路的吗？你看到有硬件工程师自己设计计时器的吗？他们有一套非常好的封装技术，他们可以把电路封装在一个接插件里面，只露出接口。别人要用的时候，只管按照接口去用，完全不必操心接插件内部的实现。

而软件工程师们呢？在 STL 成为 C++标准之前（甚至之后），每个 C++程序员都写过自己的排序算法和链表，并认为自己比别人写得更好……真是令人伤心。

OOD 可以让软件稍微“软”一点

软件不软，不过 OOD 可以帮助它稍微“软”一点。OOD 为我们提供了封装某一层面上的功能和复杂性的工具。使用 OOD，我们可以创建黑箱软件模块，将一大堆复杂的东西藏到一个简单的接口背后。然后，软件工程师们就可以使用标准的软件技术把这些黑箱组合起来，形成他们想要的应用程序。

Grady Booch 把这些黑箱称为类属（class category），现在我们则通常把它们称为“组件（component）”。类属是由被称为类（class）的实体组成的，类与类之间通过关联（relationship）结合在一起。一个类可以把大量的细节隐藏起来，只露出一个简单的接口，这正好符合人们喜欢抽象的心理。所以，这是一个非常伟大的概念，因为它给我们提供了封装和复用的基础，让我们可以从问题的角度来看问题，而不是从机器的角度来看问题。

软件的复用最初是从函数库和类库开始的，这两种复用形式实际上都是白箱复用。到 90 年代，开始有人开发并出售真正的黑箱软件模块：框架（framework）和控件（control）。框架和控件往往还受平台和语言的限制，现在软件技术的新潮流是用 SOAP 作为传输介质的 Web Service，它可以使软件模块脱离平台和语言的束缚，实现更高程度的复用。但是想一想，其实 Web Service 也是面向对象，只不过是把类与类之间的关联用 XML 来描述而已[Li, 02]。在过去的十多年里，面向对象技术对软件行业起到了极大的推动作用。在可以预测的将来，它仍将是软件设计的主要技术——至少我看不到有什么技术可以取代它的。

上面，我向读者介绍了一些背景知识，也稍微介绍了 OOD 的好处。下面，我将回答几个常见的问题，希望能借这几个问题让读者看清 OOD 的轮廓。

什么是 OOD？

面向对象设计（Object-Oriented Design，OOD）是一种软件设计方法，是一种工程化规范。这是毫无疑问的。按照 Bjarne Stroustrup 的说法，面向对象的编程范式（paradigm）是[Stroustrup, 97]：

- 决定你要的类；
- 给每个类提供完整的一组操作；
- 明确地使用继承来表现共同点。

由这个定义，我们可以看出：OOD 就是“根据需求决定所需的类、类的操作以及类之间关联的过程”。

OOD 的目标是管理程序内部各部分的相互依赖。为了达到这个目标，OOD 要求将程序分成块，每个块的规模应该小到可以管理的程度，然后分别将各个块隐藏在接口（interface）的后面，让它们只通过接口相互交流。比如说，如果用 OOD 的方法来设计一个服务器-客户端（client-server）应用，那么服务器和客户端之间不应该有直接的依赖，而是应该让服务器的接口和客户端的接口相互依赖。

这种依赖关系的转换使得系统的各部分具有了可复用性。还是拿上面那个例子来说，客户端就不必依赖于特定的服务器，所以就可以复用到其他的环境下。如果要复用某一个程序块，只要实现必须的接口就行了。

OOD 是一种解决软件问题的设计范式（paradigm），一种抽象的范式。使用 OOD 这种设计范式，我们可以用对象（object）来表现问题领域（problem domain）的实体，每个对象都有相应的状态和行为。我们刚才说到：OOD 是一种抽象的范式。抽象可以分成很多层次，从非常概括的到非常特殊的都有，而对象可能处于任何一个抽象层次上。另外，彼此不同但又互有关联的对象可以共同构成抽象：只要这些对象之间有相似性，就可以把它们当成同一类的对象来处理。

OOD 到底从哪儿来？

有很多人都认为：OOD 是对结构化设计（Structured Design，SD）的扩展，其实这是不对的。OOD 的软件设计观念和 SD 完全不同。SD 注重的是数据结构和处理数据结构的过程。而在 OOD 中，过程和数据结构都被对象隐藏起来，两者几乎是互不相关的。不过，追根溯源，OOD 和 SD 有着非常深的渊源。

1967 年前后，OOD 和 SD 的概念几乎同时诞生，它们分别以不同的方式来表现数据结构和算法。当时，围绕着这两个概念，很多科学家写了大量的论文。其中，由 Dijkstra 和 Hoare 两人所写的一些论文讲到了“恰当的程序控制结构”这个话题，声称 goto 语句是有害的，应该用顺序、循环、分支这三种控制结构来构成整个程序流程。这些概念发展构成了结构化程序设计方法；而由 Ole-Johan Dahl 所写的另一些论文则主要讨论编程语言中的单位划分，其中的一种程序单位就是类，它已经拥有了面向对象程序设计的主要特征。

这两种概念立刻就分道扬镳了。在结构化这边的历史大家都很熟悉：NATO 会议采纳了 Dijkstra 的思想，整个软件产业都同意 goto 语句的确是有害的，结构化方法、瀑布模型从 70 年代开始大行其道。同时，无数的科学家和软件工程师也帮助结构化方法不断发展完善，其中有很多今天足以使我们振聋发聩的名字，例如 Constantine、Yourdon、DeMarco 和 Dijkstra。有很长一段时间，整个世界都相信：结构化方法就是拯救软件工业的“银弹”。当然，时间最后证明了一切。

而此时，面向对象则在研究和教育领域缓慢发展。结构化程序设计几乎可以应用于任何编程语言之上，而面向对象程序设计则需要语言的支持[1]，这也妨碍了面向对象技术的发展。实际上，在 60 年代后期，支持面向对象特性的语言只有 Simula-67 这一种。到 70 年代，施乐帕洛阿尔托研究中心（PARC）的 Alan Key 等人又发明了另一种基于面向对象方法的语言，那就是大名鼎鼎的 Smalltalk。但是，直到 80 年代中期，Smalltalk 和另外几种面向对象语言仍然只停留在实验室里。

到 90 年代，OOD 突然就风靡了整个软件行业，这绝对是软件开发史上的一次革命。不过，登高才能望远，新事物总是站在旧事物的基础之上的。70 年代和 80 年代的设计方法揭示出许多有价值的概念，谁都不能也不敢忽视它们，OOD 也一样。

OOD 和传统方法有什么区别？

还记得结构化设计方法吗？程序被划分成许多个模块，这些模块被组织成一个树型结构。这棵树的根就是主模块，叶子就是工具模块和最低级的功能模块。同时，这棵树也表示调用结构：每个模块都调用自己的直接下级模块，并被自己的直接上级模块调用。

那么，哪个模块负责收集应用程序最重要的那些策略？当然是最顶端的那些。在底下的那些模块只管实现最小的细节，最顶端的模块关心规模最大的问题。所以，在这个体系结构中越靠上，概念的抽象层次就越高，也越接近问题领域；体系结构中位置越低，概念就越接近细节，与问题领域的关系就越少，而与解决方案领域的关系就越多。

但是，由于上方的模块需要调用下方的模块，所以这些上方的模块就依赖于下方的细节。换句话说，与问题领域相关的抽象要依赖于与问题领域无关的细节！这也就是说，当实现细节发生变化时，抽象也会受到影响。而且，如果我们想复用某一个抽象的话，就必须把它依赖的细节都一起拖过去。

而在 OOD 中，我们希望倒转这种依赖关系：我们创建的抽象不依赖于任何细节，而细节则高度依赖于上面的抽象。这种依赖关系的倒转正是 OOD 和传统技术之间根本的差异，也正是 OOD 思想的精华所在。

OOD 能给我带来什么？

问这个问题的人，脑子里通常是在想“ OOD 能解决所有的设计问题吗？ ”没有银弹。

---

[1] 我听见有人说“ 我可以用 C 语言实现面向对象 ”，不过我希望你不会说“ 我也喜欢这样做 ”。很明显，如果没有语言特性的支持，面向对象方法将寸步难行。其实结构化设计也是一样，不过几乎所有的编程语言都提供了对三种基本流程结构的支持，所以基本不会遇到这个问题。

OOD 也不是解决一切设计问题、避免软件危机、捍卫世界和平……的银弹。OOD 只是一种技术。但是，它是一种优秀的技术，它可以很好地解决目前的大多数软件设计问题——当然，这要求设计者有足够的能力。

OOD 可能会让你头疼，因为要学会它、掌握它是很困难的；OOD 甚至会让你失望，因为它也并不成熟、并不完美。OOD 也会给你带来欣喜，它让你可以专注于设计，而不必操心那些细枝末节；OOD 也会使你成为一个更好的设计师，它能提供给你很好的工具，让你能开发出更坚固、更可维护、更可复用的软件。

C++是一种“真正的”面向对象编程语言吗？

最后这个问题和我们今天的主题关系不大，不过这是一个由来已久的问题，而且以后也不一定会有合适的时机说明这个问题，所以今天一起回答了。

很多人都觉得 C++缺少“真正的”面向对象语言所必须的一些特性，例如垃圾收集（garbage collection）、多分配（multiple-dispatch）之类的。但是，缺少这些特性并不影响开发者们用 C++实现面向对象的设计思路。

在我看来，任何语言，只要它直接支持面向对象设计的实现，那它就是“真正的”面向对象语言。用这个标准来评价，C++是完全符合的：它直接支持继承、多态、封装和抽象，而这些才是面向对象最重要的。而 VB 和 C 这样的语言不能直接支持这些（尽管可以用特殊的技巧来实现），所以不是“真正的”面向对象语言。

参考书目

- [Brooks, 95] Frederick Brooks, *The Mythical Man-Month*, Addison-Wesley, 1995.
- [Li, 02] 李维，《Delphi 6/Kylix 2 SOAP/Web Service 程序设计篇》，机械工业出版社，2002 年。
- [Martin, 95] Robert Martin, *Designing Object-Oriented C++ Applications Using the Booch Method*, Prentice-Hall, 1995.
- [Stroustrup, 97] Bjarne Stroustrup, *The C++ Programming Language (Special Edition)*, Addison-Wesley, 1997.