志中刊登了标题中含有面向对象编程的专栏，那么仅凭这一点，杂志的销路就好不了。

虽然现状如此，但是还是让笔者讲解一下面向对象编程吧。因为在未来的开发环境中，将成为主流的不是 Java 就是 .NET[①]，而无论选择哪个，面向对象编程的知识都是不可或缺的。这使得在这之前还对其敬而远之的程序员们也不得不迎头赶上了，因为他们已经没有退路了。

的确，精通面向对象编程需要花费大量时间。所以请诸位先通过阅读本章，掌握一些基础知识，至少能够说出面向对象是什么。然后再为实践面向对象编程而开始踏踏实实的深层学习吧。

7.2　对 OOP 的多种理解方法

在计算机术语辞典等资料中，常常对面向对象编程做出了如下定义。

面向对象编程是一种基于以下思路的程序设计方法：将关注点置于对象（Object）本身，对象的构成要素包含对象的行为及操作[②]，以此为基础进行编程。这种方法使程序易于复用，软件的生产效率因而得以提升。其中所使用的主要编程技巧有继承、封装、多态三种。

这段话足以作为对术语的解释说明，但是仅凭这段话我们还是无法理解面向对象编程的概念。

① 原作写于 2003 年，所以当时的情况和当时对未来的展望可能和今天的状况多少有些出入。——译者注

② 在 C 语言中，结构体是数据的集合，它将数据捆绑在一起，使得我们可以将这些数据看作是一个整体。而对结构体中的数据进行操作的函数却写在了结构体的外部。然而在面向对象编程中，将表示事物行为的函数也放入了这个整体，这就形成了对象的概念，使得这个整体既能描述属性，又能描述行为。——译者注

“面向对象编程是什么?”如果去问十名程序员，恐怕得到的答案也会是十种。就此打个可能稍微有点特别的比方吧。有几个人去摸一只刺猬，但他们看不到刺猬的全身。有的人摸到了刺猬的后背，就会说“摸起来扎手，所以是像刷子一样的东西”；而有的人摸到了刺猬的尾巴，就会说“摸起来又细又长，所以是像绳子一样的东西”（如图 7.1 所示）。同样的道理，随着程序员看问题角度的不同，对面向对象编程的理解也会是仁者见仁、智者见智。

图 7.1 面向对象编程是什么?

那么到底哪种理解方法才是正确的呢？其实无论是哪种方法，只要能够通过实际的编程将其付诸实践，那么这种方法就是正确的。诸位也可以用自己的理解方法去实践面向对象编程。虽然是这么说，但

如果仅仅学到了片面的理解方法，也是无法看到面向对象编程的全貌的，会感到对其概念的理解是模模糊糊的。因此，下面我们就把各种各样的理解方法和观点综合起来，以此来探究面向对象编程的全貌吧。

7.3　观点 1：面向对象编程通过把组件拼装到一起构建程序

在面向对象编程中，使用了一种称为“类”的要素，通过把若干个类组装到一起构建一个完整的程序。从这一点来看，可以说类就是程序的组件（Component）。面向对象编程的关键在于能否灵活地运用类。

首先讲解一下类的概念。在第 1 章中讲解过，无论使用哪种开发方法，编写出来的程序其内容最终都会表现为数值的罗列，其中的每个数值要么表示“指令”，要么表示作为指令操作对象的“数据”。程序最终就是指令与数据的集合。

在使用古老的 C 语言或 BASIC 等语言编程时（它们不是面向对象的编程语言，即不是用于表达面向对象编程思想的语言），用“函数”表示指令，用“变量”表示数据。对于 C 语言或是 BASIC 的程序员而言，程序就是函数和数据的集合。在代码清单 7.1 中，用 FunctionX 的形式为函数命名，用 VariableX 的形式为变量命名。

代码清单 7.1　程序是函数和变量的集合（C 语言）

```
int Variable1;                          } 变量
int Variable2;
int Variable3;
…
void Function1() { 处理过程 }           } 函数
void Function2() { 处理过程 }
void Function3() { 处理过程 }
…
```

在大型程序中需要用到大量的函数和变量。假设要用非面向对象的编程方法编写一个由 10000 个函数和 20000 个变量构成的程序，那么结果就很容易是代码凌乱不堪，开发效率低到令人吃惊，维护起来也十分困难。

于是一种新的编程方法就被发明出来了，即把程序中有关联的函数和变量汇集到一起编成组。这里的组就是类。在 C++、Java、C# 等面向对象编程语言中，语法上是支持类的定义的。在代码清单 7.2 中，就定义了一个以 MyClass 为名称的类。因为程序的构成要素中只有函数和变量，所以把它们分门别类组织起来的类也理所当然地成了程序的组件。通常把汇集到类中的函数和变量统称为类的“成员”（Member）。

为了使 C 语言支持面向对象编程，人们扩充了它的语法，开发出了 C++ 语言。而通过改良 C++ 又开发出了 Java 和 C#。在本章中，将会分别介绍用 C 语言、C++、Java 和 C# 编写的示例程序。诸位在阅读时只需抓住其大意即可，不必深究每个程序的具体内容。

代码清单 7.2　定义类 MyClass，将函数和变量组织到一起（C++）

```
class MyClass ──────────────────── 类名
{
    int Variable1;
    int Variable2;                          ┐
    …                                       │ 类的成员
    void Function1() { 处理过程 }           │ （变量和函数）
    void Function2() { 处理过程 }           │
    …                                       ┘
};
```

7.4　观点 2：面向对象编程能够提升程序的开发效率和可维护性

在使用面向对象编程语言开发时，并不是所有的类都必须由程序员亲自编写。大部分的类都已内置于面向对象编程语言中了，这些类可以为来自各个领域的程序员所使用。通常将像这样的一组类（一组组件）称作“类库”。通过利用类库可以提升编程的效率。还有一些类原本是为开发其他程序而编写的，如果可以把这些现成的类拿过来使用，那么程序的开发效率就更高了。

所谓企业级的程序，指的是对可维护性有较高要求的程序。可维护性体现在当程序投入使用后对已有功能的修改和新功能的扩充上。如果所维护的程序是用一组类组装起来的话，那么维护工作就轻松了。之所以这样说，是因为作为维护对象的函数和变量，已经被汇聚到名为类的各个组中了。举例来说，假设我们已经编写出了一个用于员工薪资管理的程序。随着薪资计算规则的变更，程序也要进行修改，那么需要修改的函数和变量就应该已经集中在一个类中了，比如一个叫作 CalculationClass 的类（如图 7.2 所示）。也就是说，维护时没有必要去检查所有的类，只需修改类 CalculationClass 就可以了。关于可维护性，在第 12 章中还会继续介绍。

“我是创造类的人，你是使用类的人”——在实际应用面向对象编程时要带着这个感觉。开发小组中的全体成员没有必要都对程序中的方方面面有所了解，而是组中有些人只负责制作组件（类），有些人只负责使用组件。当然也会有需要同时做这两种工作的情况。另外，还可以把一部分组件的开发任务委托给合作公司，或者买来商业组件使用。

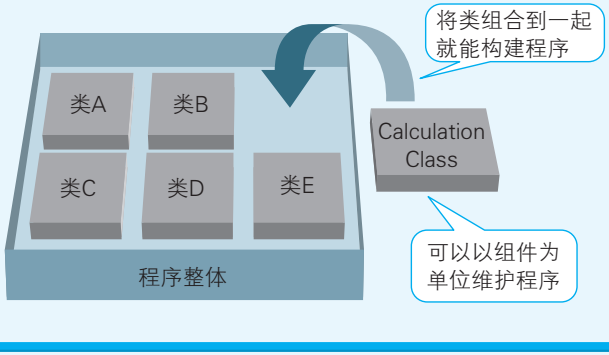

图 7.2 只要把组件组合起来就可以高效地开发、轻松地维护

对于创造类的程序员，他们考虑的是程序的开发效率和可维护性，并决定应该将什么抽象为类。如果一个类的修改导致其他的类就也要跟着修改，这样的设计是不行的。必须把组件设计成即使是坏了（有缺陷了）也能轻松地替换，就像在汽车或家电等工业制品中所使用的组件那样。

在功能升级后，旧组件能够被新组件所替换的设计也是必不可少的。因此，创造者和使用者之间就需要事先商定类的使用规范。请诸位记住，对于类的使用者而言“类看起来是什么样子的”这种关于规范的描述通常被称为“接口”（Interface）。例如只要把接口告诉合作公司，就可以要求他们编写类，编写出的类也就自然能够与程序中的其他部分严丝合缝地拼装起来。在面向对象语言中，也提供了用于定义接口的语法。

7.5　观点 3：面向对象编程是适用于大型程序的开发方法

通过之前的介绍，诸位应该也理解了为什么说面向对象编程适用于编写大型程序。假设一个程序需要 10000 个函数和 20000 个变量，如果把这个程序用 100 个类组织起来，那么平均一个类里就只有 100 个函数和 200 个变量了。程序的复杂度也就降到了原来的 1%。而如果使用了稍后将会讲解的封装这种编程技巧（即将函数和变量放入黑盒，使其对外界不可见），还可以更进一步降低复杂度。

在讲解面向对象编程的书籍和杂志文章中，由于受到篇幅的限制，往往无法刊登大篇幅的示例程序。而通过短小的程序恐怕又无法把面向对象编程的优点传达出来。当然本书也不例外。所以就要请诸位读者一边假想着自己在开发一个大型程序，一边阅读本书的解说。

为了拉近计算机和人的距离，使计算机成为更容易使用的机器，围绕着计算机的各种技术都在不断发展。在人的直觉中，大件物品都是由组件组装起来的。因此可以说面向对象编程方法把同样的直觉带给了计算机，创造了一种顺应人类思维习惯的先进的开发方法。

7.6　观点 4：面向对象编程就是在为现实世界建模

程序可以在计算机上实现现实世界中的业务和娱乐活动。计算机本身并没有特定的用途，而是程序赋予了计算机各种各样的用途。在面向对象编程中，可以通过“这个是由什么样的对象构成的呢?”这样的观点来分析即将转换成程序的现实世界。这种分析过程叫作“建模”。可以说建模对于开发者而言，反映的是他们的世界观，也就是在他们的眼中现实世界看起来是什么样子的。

在实际建模的过程中，要进行“组件化”和“省略化”这两步。所谓组件化，就是将可看作是由若干种对象构成的集合的现实世界分割成组件。因为并不需要把现实世界 100% 地搬入到程序中，所以就可以忽略掉其中的一部分事物。举例来说，假设要为巨型喷射式客机建模，那么就可以从飞机上抽象归类出机身、主翼、尾翼、引擎、轮子和座席等组件（如图 7.3 所示）。而像是卫生间这样的组件，不需要的话就可以省略。“建模”这个词也可以理解为是制作塑料模型。虽然巨型喷射式客机的塑料模型有很多零件，但是其中应该会省略掉卫生间吧，因为这对于塑料模型来说不是必需的。

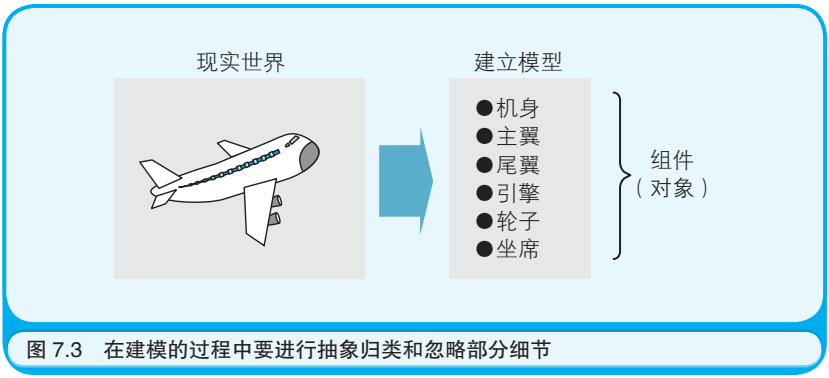

图 7.3　在建模的过程中要进行抽象归类和忽略部分细节

7.7　观点 5：面向对象编程可以借助 UML 设计程序

可以说建模就是在为面向对象编程做设计。为了把对现实世界建模的结果以图形的形式表示出来，还经常使用被称作 UML（Unified Modeling Language，统一建模语言）的表记方法。UML 是通过统一历史上曾经出现的各种各样的表记方法而发明出来的，事实上 UML 已经成为了建模表记方法中的世界标准。

在 UML 中，规定了九种图（见表 7.1）。之所以有这么多种，是为了从各种各样的角度表示对现实世界建模的结果。例如用例图是从用户的角度，即用户使用程序的方式出发表示建模结果的一种图。而类图等出发的角度则是程序。

表 7.1　UML 中规定的九种图

名称	主要用途
用例图（Use Case Diagram）	表示用户使用程序的方式
类图（Class Diagram）	表示类以及多个类之间的关系
对象图（Object Diagram）	表示对象
时序图（Sequence Diagram）	从时间上关注并表示多个对象间的交互
协作图（Collaboration Diagram）	从合作关系上关注并表示多个对象间的交互
状态图（Statechart Diagram）	表示对象状态的变化
活动图（Activity Diagram）	表示处理的流程等
组件图（Component Diagram）	表示文件以及多个文件之间的关系
配置图（Deployment Diagram）	表示计算机或程序的部署配置方法

UML 仅仅规定了建模的表记方法，并不专门用于面向对象编程。因此公司的组织架构图和业务流程图等也可以使用 UML 表记。

“这儿可有九种图呢，记忆起来很吃力啊”——也许会有人这么想吧。但是可以换一种积极的想法来看待它。既然 UML 被广泛地应用于绘制面向对象编程的设计图，那么只要了解了 UML 中仅有的这九种图的作用，就可以从宏观的角度把握并理解面向对象编程思想了。怎么样，如果这样想的话，就应该会对学习 UML 跃跃欲试了吧。

图 7.4 中有一个 UML 类图的示例。图中所画的类表示的正是前面代码清单 7.2 中的类 MyClass。将一个矩形分为上中下三栏，在上面的一栏中写入类名，中间的一栏中列出变量（在 UML 中称为“属性”），在下面的一栏中列出函数（在 UML 中称为“行为”或是“操作”）。

在进行面向对象编程的设计时，要在一开始就把所需要的类确定下来，然后再在每个类中列举出该类应该具有的函数和变量，而不要到了后面才把零散的函数和变量组织到类中。也就是说，要一边观察作为程序参照物的现实世界，一边思考待解决的问题是由哪些事物（类）构成的。正因为在设计时要去关注对象，这种编程方法才被称为面向对象编程（Object Oriented Programming，其中的 Oriented 就是关注的意思）。而在那些传统的开发方法中，进行设计则是要先考虑程序应该由什么样的功能和数据来构成，然后立即确定与之相应的函数和变量。与此相对在面向对象编程的设计中，因为一上来就要确定有哪些类，从而构成程序的函数和变量就必然会被组织到类中。

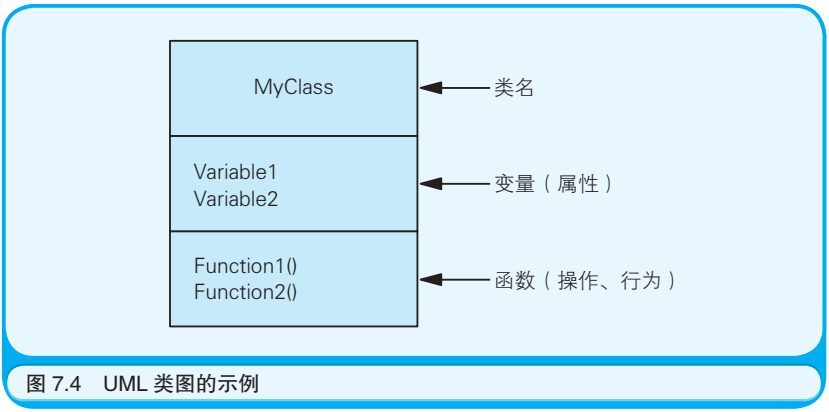

图 7.4 UML 类图的示例

7.8 观点 6：面向对象编程通过在对象间传递消息驱动程序

假设要编写这样一个程序，玩家 A 和玩家 B 玩剪刀石头布，由裁判判定输赢。如果使用作为非面向对象编程语言的 C 语言编写，程序就会像代码清单 7.3 中那样；如果使用作为面向对象编程语言的 C++ 编