写，程序就会像代码清单 7.4 中那样。诸位能看出其中的差异吗？

代码清单 7.3　未使用面向对象编程语言的情况（C 语言）

```
/* 玩家 A 确定手势 */
a = GetHand();

/* 玩家 B 确定手势 */
b = GetHand();

/* 判定胜负 */
winner = GetWinner(a, b);
```

代码清单 7.4　使用了面向对象编程语言的情况（C++）

```
// 玩家 A 确定手势
a = PlayerA.GetHand();

// 玩家 B 确定手势
b = PlayerB.GetHand();

// 由裁判判定胜负
winner = Judge.GetWinner(a, b);
```

在 C 语言的代码中，仅仅使用了 GetHand() 和 GetWinner() 这种独立存在的函数。与此相对在 C++ 的代码中，因为函数是隶属于某个类的，所以要使用 PlayerA.GetHand() 这样的语法，表示属于类 PlayerA 的函数 GetHand()。

也就是说用 C++ 等面向对象编程语言编写程序的话，程序可以通过由一个对象去调用另一个对象所拥有的函数这种方式运行起来。这种调用方式被称为对象间的“消息传递”。在面向对象语言中所说的消息传递指的就是调用某个对象所拥有的函数。即便是在现实世界中，我们也是通过对象间的消息传递来开展业务或度过余暇的。在面向对象编程中还可以对对象间的消息传递建立模型。

如果未使用面向对象编程语言，那么可以用流程图表示程序的运行过程。流程图表示的是处理过程的流程，因此通常把非面向对象语

言称为“过程型语言”。而且可以把面向对象编程语言和面向过程型语言，面向对象编程和面向过程编程分别作为一对反义词来使用。

如果使用的是面向对象编程语言，那么可以使用 UML 中的“时序图”和“协作图”表示程序的运行过程。在图 7.5 中对比了流程图和时序图。关于流程图已经没有必要再进行介绍了吧。在时序图中，把用矩形表示的对象横向排列，从上往下表示时间的流逝，用箭头表示对象间的消息传递（即程序上的函数调用）。诸位在这里只需要抓住图中的大意即可。

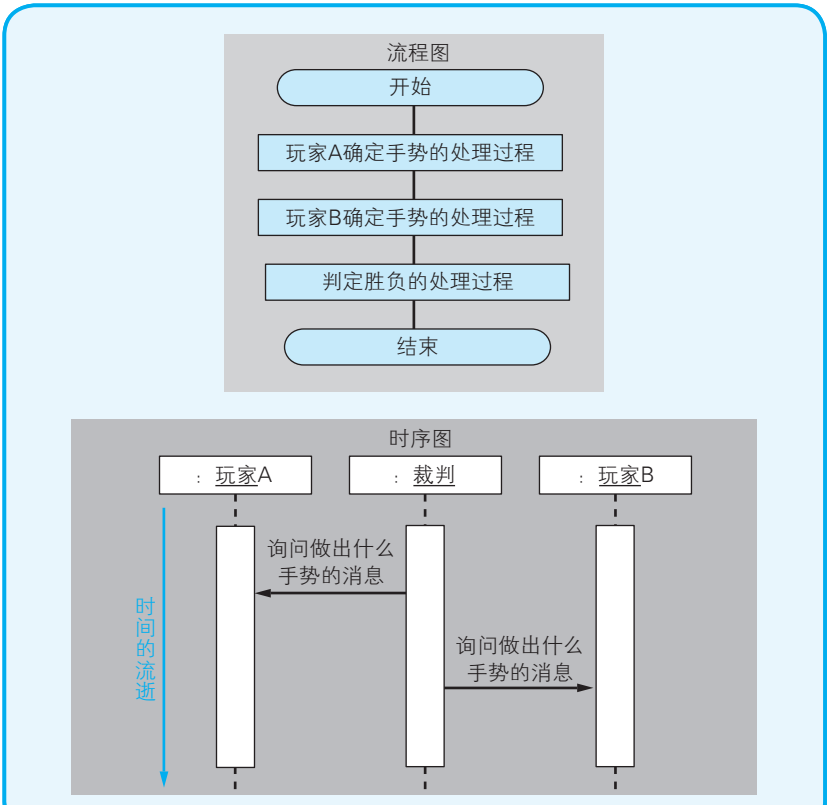

图 7.5 流程图与时序图的对比

沉浸在面向过程编程中的程序员们通常都习惯于用流程图思考程序的运行过程。可是为了实践面向对象编程，就有必要改用时序图来考虑程序的运行过程。

7.9　观点 7：在面向对象编程中使用继承、封装和多态

“继承”（Inheritance）、“封装”（Encapsulation）和“多态”（Polymorphism，也称为多样性或多义性）被称为面向对象编程的三个基本特性。在作为面向对象编程语言的 C++、Java、C# 等语言中，都已具备了能够用程序实现以上三个特性的语法结构。

继承指的是通过继承已存在的类所拥有的成员而生成新的类。封装指的是在类所拥有的成员中，隐藏掉那些没有必要展现给该类调用者的成员。多态指的是针对同一种消息，不同的对象可以进行不同的操作。

其实仅仅介绍如何在程序中使用这三个基本特性，就已经需要一本书了。因而有很多人就会被所学到的语法结构和编程技术中涉及的大量知识所束缚，以致不能按照自己的想法编写程序。其实只要沉静下来，不拘泥与语法和技术，转而去关注使用这三个特性所带来的好处，就能顺应着自己的需求恰当地使用面向对象编程语言了。

只要去继承已存在的类，就能高效地生成新的类。如果一个类被多个类所继承，那么只要修正了这个类，就相当于把继承了这个类的所有类都修正了。只要通过封装把外界不关心的成员隐藏起来，类就可以被当作是黑盒，变成了易于使用且便于维护的组件了。而且由于隐藏起来的成员不能被外界所访问，所以也就可以放心地随意修改这

些成员。只要利用了多态，生成对同一个消息可以执行多种操作的一组类，使用这组类的程序员所需要记忆的东西就减少了。总之，无论是哪一点，都是面向对象编程所带来的好处，都可以实现开发效率和可维护性的提升。

稍后将会介绍如何在实际的编程中使用继承。为了对类进行封装，需要在类成员的定义前指定关键词 public（表示该成员对外可见）或是 private（表示该成员对外不可见）。之前的代码清单 7.2 中省略了这些关键词。实现多态可以有多种方法，感兴趣的读者可以去翻阅面向对象语言的教材等相关资料。

7.10 类和对象的区别

前面介绍了有关面向对象的几种观点。诸位读者应该已经了解面向对象编程是怎么一回事了吧。但是请允许笔者再补充一些面向对象编程中必不可少的知识。

首先，要说明一下类和对象的区别。在面向对象编程中，类和对象被看作是不同的概念而予以区别对待。类是对象的定义，而对象是类的实例（Instance）。经常有教材这样说明二者之间的关系：类是做饼干的模具，而用这个模具做出来的饼干就是对象。虽然这是个有趣的比喻，但是如果这样类比的话，就有可能无法看清二者在实际编程中的关系（如图 7.6 所示）。

在之前的代码清单 7.2 所示的程序中，定义了一个类 MyClass。但是我们还无法直接使用类 MyClass 所持有的成员，要想使用就必须在内存上生成该类的副本，这个副本就是对象（如代码清单 7.5 所示）。

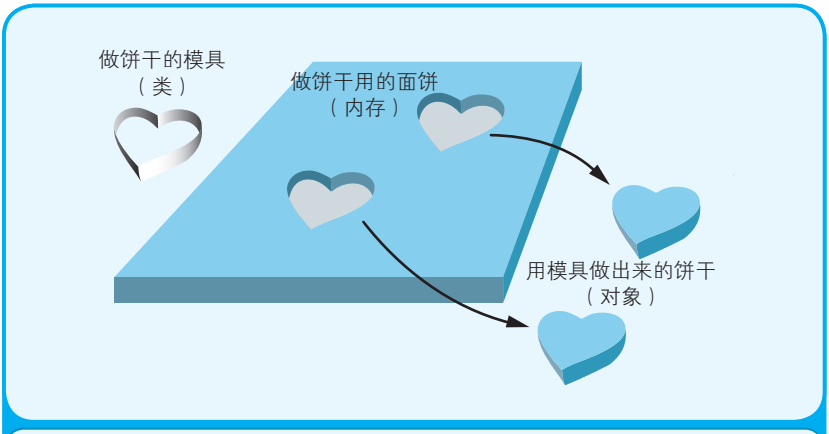

图 7.6　类和对象的关系就像饼干模具和饼干的关系

代码清单 7.5　先创建类的对象然后再使用（C++）

```
MyClass obj;            // 创建对象
obj.Variable = 123;     // 使用对象所持有的变量
obj.Function();         // 使用对象所持有的函数
```

先要创建一个个的对象然后才能使用类中定义的成员，对于面向对象语言的初学者而言，他们会认为这样做很麻烦。但是也只能这样做，因为这是面向对象语言就是这样规定的。可是为什么要确立这样的规则呢？原因是即便是在现实世界中，也有类（定义）和对象（实体）的区别。举例来说，假设我们定义了一个表示企业中雇员的类 Employee。如果仅仅是定义完就可以立刻使用类 Employee 中的成员，那么程序中实际上就只能存在一名雇员。而如果规定了要先创建类 Employee 的对象才能使用，那么就可以需要多少就创建多少雇员了（通过在内存上创建出类 Employee 的副本）。在这一点上，稍后将要介绍的具有两个文本框的 Windows 应用程序也是如此，也就是说这个程序创建了两个文本框类的对象。

这样的话，就更能理解“类是做饼干的模具，用模具做出来的饼干

是对象”这句话的含义了吧。有了一个做饼干的模具（类），那么需要多少就能做出多少饼干（对象）。

7.11 类有三种使用方法

前面已经介绍过了，在面向对象编程中程序员可以分工，有的人负责创建类，有的人负责使用类。创建类的程序员需要考虑类的复用性、可维护性、如何对现实世界建模以及易用性等，而且还要把相关的函数和变量汇集到类中。这样的工作称为“定义类”。

而使用类的程序员可以通过三种方法使用类，关于这一点诸位要有所了解。这三种方法分别是：1. 仅调用类所持有的个别成员（函数和变量）；2. 在类的定义中包含其他的类（这种方法被称作组合）；3. 通过继承已存在的类定义出新的类。应该使用哪种方法是由目标类的性质以及程序员的目的决定的。

在诸位平时所见的程序背后，程序员们也是按照上述三种方法使用类的。代码清单 7.6 中列出了一段用 C# 编写的 Windows 应用程序。当用户点击按钮，就会弹出一个消息框，里面显示的是输入到两个文本框中的数字进行加法运算后的结果（如图 7.7 所示）。

诸位在这里不需要深究程序代码的含义，而是要把注意力集中到类的三种使用方法上。在这个程序中，表示整体界面的是以 Form1 为类名的类。类 Form1 继承了类库中的类 System.Windows.Forms.Form。在 C# 中用冒号“:”表示继承。在窗体上，有两个文本框和一个按钮，用程序来表示的话，就是类 Form1 的成员变量分别是以类 System.Windows.Forms.TextBox（文本框类）为数据类型的 textBox1、textBox2，和以类 System.Windows.Forms.Button（按钮类）为数据类型的 button1。像这样类中就包含了其他的类，也可以说是类中引用了其他的类。而

代码中的 Int32.Parse 和 MessageBox.Show，只不过是个别调用了类中的函数。

代码清单 7.6　进行加法运算的 Windows 应用程序（用 C# 编写）

```
public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.TextBox textBox2;
    private System.Windows.Forms.Button button1;
        ...
    private void button1_Click(object sender,
    System.EventArgs e)
    {
        int a, b, ans;
        a = Int32.Parse(textBox1.Text);
        b = Int32.Parse(textBox2.Text);
        ans = a + b;
        MessageBox.Show(ans.ToString());
    }
}
```

通过继承使用

通过组合使用

调用类的成员

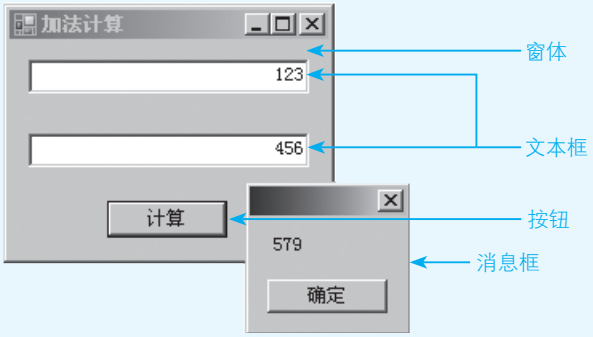

图 7.7　Windows 应用程序的运行结果

7.12 在 Java 和 .NET 中有关 OOP 的知识不能少

在本章的最后，笔者来解释一下为什么说程序员已经到了无法逃避面向对象编程的地步了。在未来的开发环境中，将成为的主流的不是 Java 就是 .NET。Java 和 .NET 其实是位于操作系统（Windows 或 Linux 等）之上，旨在通过隐藏操作系统的复杂性从而提升开发效率的程序集，这样的程序集也被称作“框架”（Framework）。框架由两部分构成，一部分是负责安全执行程序的“执行引擎”，另一部分是作为程序组件集合的“类库”（如图 7.8 所示）。

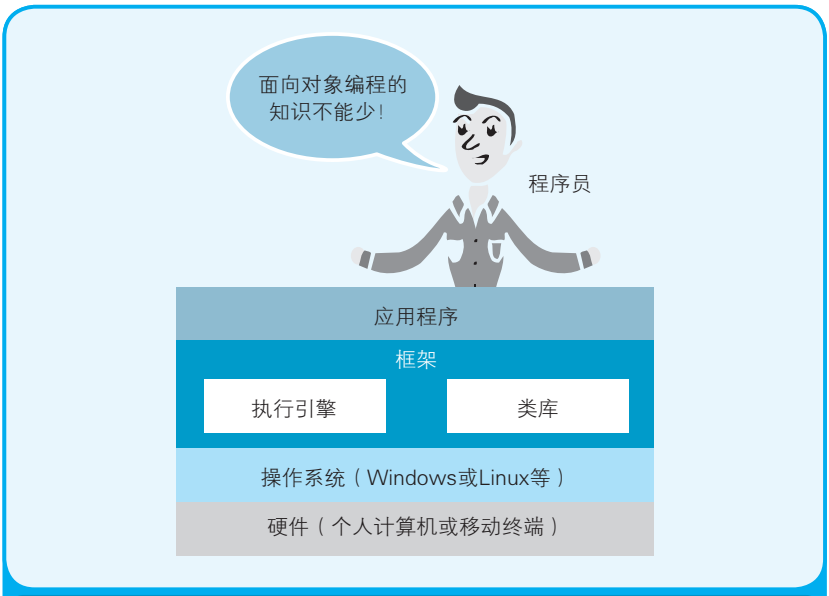

图 7.8 框架提供了高效的开发环境

无论是使用 Java 还是 .NET，都需要依赖类库进行面向对象编程。在 Java 中，使用的是与框架同名的 Java 语言。而在 .NET 中，使用的

是 .NET 框架支持的 C#、Visual Basic.NET、Visual C++、NetCOBOL 等语言进行开发。上述的每种语言都是面向对象语言。其中 Visual Basic.NET 和 NetCOBOL 是在古老的 Visual Basic 和 COBOL 语言中增加了面向对象的特性（类的定义、继承、封装和多态等）而诞生的新语言。至今还对面向对象编程敬而远之的程序员们，你们已经不得不迎头赶上了。不要再觉得麻烦什么的了，以享受技术进步的心情开始学习面向对象编程吧！

☆　　☆　　☆

通过综合整理面向对象的各种理解方法，相信诸位已经能看到面向对象的全貌了。但这里还有一点希望诸位注意，那就是请不要把面向对象当成是一门学问。程序员是工程师，工程是一种亲身参与的活动而不是一门学问。请诸位把面向对象编程作为一种能提升编程效率、写出易于维护的代码的编程方法，在适当的场合实践面向对象编程，而不要被它各种各样的概念以及所谓的编程技巧所束缚。

面向对象编程就是通过把组件拼装到一起进行编程的方法——笔者曾经明确下过这样的结论，也是以此为理念进行实践的。但是也许有人会摆出学者的那一套理论：“你还没有明白面向对象编程的理念，你这个是面向组件编程！”如果真有人这样说，笔者就会反问他：“这么说你正在实践面向对象编程吗?”

在接下来的第 8 章，笔者将一改编程的话题，开始讲解数据库。敬请期待！

第 8 章
一用就会的数据库

热身问答

在阅读本章内容前，让我们先回答下面的几个问题来热热身吧。

初级问题

数据库术语中的“表”是什么意思？

中级问题

DBMS 是什么的简称？

高级问题

键和索引的区别是什么？