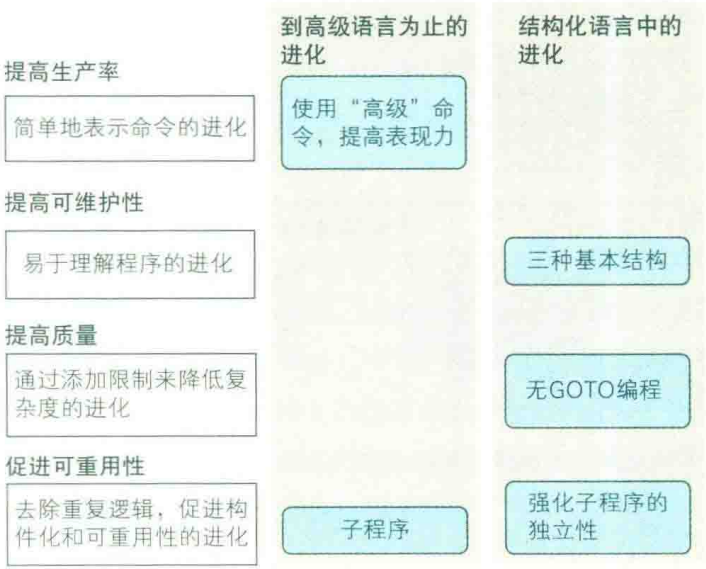


图 3-5　编程语言的进化（之一）

3.9　没有解决全局变量问题和可重用性差的问题

结构化编程成了程序员的常识，直到最近才被面向对象夺了风头。此前在大学课程中或者企业新人培训时，结构化编程是一定会讲的课题。

不过，结构化编程有两个无法解决的问题，那就是全局变量问题和可重用性差的问题（图 3-6）。

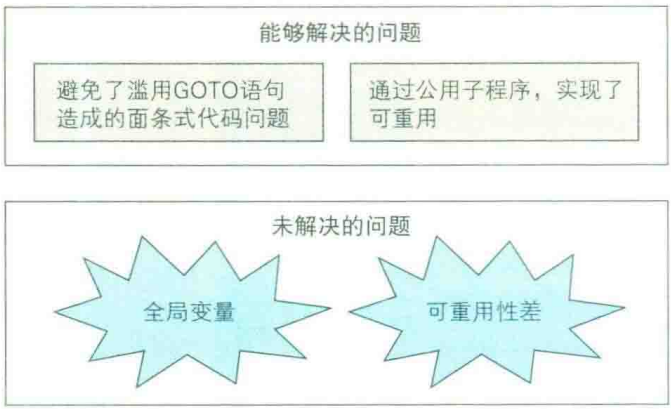


图 3-6　结构化编程中能够解决和未解决的问题

第一个是全局变量问题。结构化语言中导入了局部变量和按值传递结构，可以尽量不使用全局变量来交换信息。不过，局部变量是临时变量，在子程序调用结束时就会消失，因此在子程序运行结束后依然需要保持的信息就只能被存放在子程序的外面，即被保存为全局变量。

滥用 GOTO 语句会严重影响程序的可读性，但这只限于编写该逻辑的部分。而全局变量可以被程序的任何位置使用，所以当因某种情况而需要修改全局变量时，为了查明影响范围，就必须调查所有的逻辑。如果程序很大，那么这种由全局变量引发的问题就会非常严重，这也是结构化语言中很难避免的问题[1]。

另一个问题是可重用性差。结构化语言中可重用的是子程序，而现在已经有了用于编码转换、输入输出处理、数值计算和字符串处理等的通用库，通过重用既有程序就可以在一定程度上实现基本的处理。然而即便如此，从不断增大的应用程序的整体规模来看，这种程度只能说是微不足道的。

① C 语言中通过添加 static 修饰符来限制可以访问全局变量的范围。另外，COBOL 的 WORKING-STORAGE SECTION 的动作与 C 语言的 static 变量一样。关于这些功能与 OOP 的不同之处，我们将在下一章中介绍。

因此需要提高可重用的规模，这已经成为软件开发者的共识，但这一点却很难实现，主要原因就在于作为公用构件创建的只是子程序。

而能够打破该限制的正是 OOP。

正如我们在第 1 章介绍的那样，OOP 早在 1967 年就已经作为 Simula 67 出现了。不过，由于当时计算机硬件性能低下，而 OOP 又过于先进，所以在很长一段时间内，只有一部分研究机构使用 OOP。到了 20 世纪 80 年代，能在具有 GUI（Graphical User Interface，图形用户界面）的工作站上运行的商用语言 Smalltalk 出现，同时 C++ 也作为 C 语言的增强版被设计出来，通过 GUI 库的开发，OOP 的灵活性和可重用性得到证实，慢慢开始崭露头角。随着互联网热潮中 Java 的出现，OOP 逐渐成为主流。

下一章我们将正式开始介绍面向对象编程。

深入学习的参考书籍

[1] 高橋麻奈. やさしい C 第 3 版 [M]. 东京：SB Creative，2007.

☆☆

[2] 柴田望洋. 明解 C 语言（第 3 版）：入门篇 [M]. 管杰等，译. 北京：人民邮电出版社，2015.

☆☆

新人培训时学习 Java 的工程师可能也需要掌握 C 语言作为计算机的基础知识。C 语言的经典著作是柯尼汉（Kernighan）和里奇（Ritchie）编写的《C 程序设计语言（第 2 版・新版）》，而在已出版的众多入门书中，以上两本是由日本人编写的比较经典的参考书。

[3] 矢沢久雄. 情報はなぜビットなのか——知っておきたいコンピュータと情報処理の基礎知識 [M]. 东京：日経 BP 社，2006.

☆☆

该书引用身边的例子进行讲解，让大家轻松掌握计算机和信息处理的基础知识。该书从比特和字节开始讲起，涉及字符编码、算法、统计、概率、运筹学、关系契约理论和通信协议等，其中还刊登了对计算机发展做出卓越贡献的人物的照片。

专栏

编程往事

COBOL 编译器的鸡和蛋的问题

这是发生在笔者年轻时的事情。有一次前辈问笔者："你知道怎样编写 COBOL 编译器吗？"虽然笔者每天都在使用编译器，但却很少注意到它本身其实也是一种程序。另外，笔者当时根本就没有考虑过编译器是怎样编写的，所以只好回答"不是很清楚"。

这时前辈说："COBOL 编译器也是用 COBOL 编写的哦。""什么？COBOL 竟然能编写编译器？"笔者感到有些意外。在回家的电车上，笔者再次想起这个问题，发现这有点像先有鸡还是先有蛋的问题，让笔者有些混乱。

当然，其实这个问题的答案很简单。正确答案是，最初的 COBOL 编译器是使用 COBOL 以外的其他编程语言（如 FORTRAN 等）编写的。也就是说，最开始的那只小鸡的母亲并不是鸡，而应该是其他鸟类。

＊＊＊

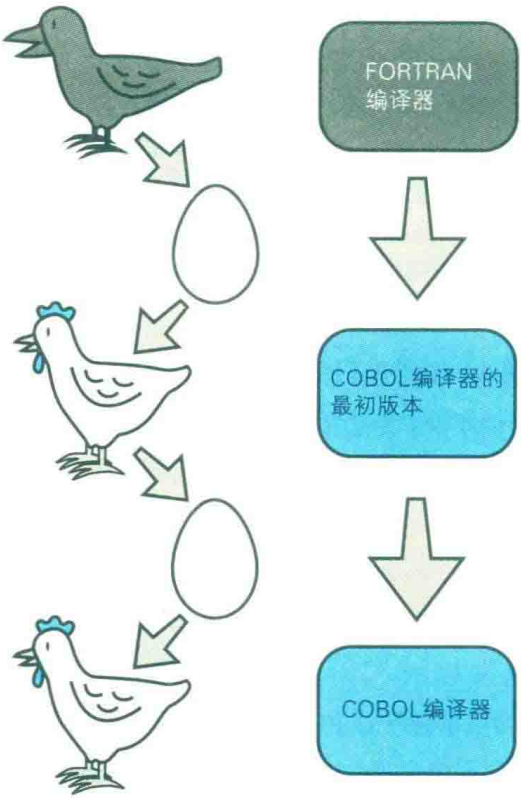
到这儿就清楚了。然而，编写 COBOL 编译器的其他语言的编译器最开始又是怎样编写的呢？世界上最早的高级语言和最早的汇编语言呢？笔者不停地思考着这些问题，总感觉自己有了重大发现。

世界上最早的高级语言的编译器应该是使用当时最新的编程语言——汇编语言编写的，而最早的汇编语言是用机器语言编写的。世上当然不会发生无中生有的事。

也就是说，我们每天使用的编译器凝结着前人在计算机发展过程中倾注的智慧和心血。像这样，利用既有的工具创造出新的工具，这简直就是人类进化的缩略图啊！

＊＊＊

编程语言从机器语言开始，逐渐发展为汇编语言、高级语言和结构化语言，直至目前被广泛使用的 OOP。这其中不仅包含了编程语言语法的进化，而且在使用当前最新的编程语言编写编译器时，其实也利用了前人的



成就。

因此，在 OOP 之后出现的编程语言当然是用 OOP 编写的。大家一定都很期待看到接下来会出现什么样的编程语言吧。

不过，无论出现了多么优秀的编程语言，最伟大的还是当初发明汇编语言，并使用机器语言编写编译器的先驱者吧。

本章的关键词

类、实例、实例变量、方法、多态、继承、包、异常、垃圾回收

第4章 面向对象编程技术：去除冗余、进行整理

热身问答

在阅读正文之前，请挑战一下下面的问题来热热身吧。

问题

美国计算机协会（ACM）每年都会将“图灵奖”授予在计算机科学领域做出突出贡献的个人，请问“图灵”这一名称的由来是什么？

A. 开发了最早的计算机的项目名称

B. 最早研究计算机科学理论的人的名字

C. 最早发明出来的计算机的名称

D. 最初放置计算机的地方的名称

B. 最早研究计算机科学理论的人的名字

解 析

图灵奖每年由美国计算机协会颁发，授予在计算机科学领域做出突出贡献的个人，因此也被称为计算机界的诺贝尔奖，非常有权威性。在 2000 年之后，为面向对象的发展做出贡献的人们也获得过这一奖项。

该奖项的名称取自于艾伦・图灵（Alan Turing）。图灵是一位数学家，被称为“计算机科学之父”。他设计的“图灵机”将计算与算法的概念形式化，为现在实际应用的计算机奠定了逻辑基础。

另外，在第二次世界大战期间，为了计算导弹的弹道轨迹，以美国为中心的多个国家对计算机进行了研究和开发。虽然有不少人认为在 1946 年公布于世的电子数字积分计算机（Electronic Numerical Integrator and Computer，ENIAC）是世界上第一台计算机，但关于这一点目前还存在争论。现在我们普遍认为 1937 年完成的试验样机——阿塔纳索夫 - 贝瑞计算机（Atanasoff-Berry Computer，ABC）是世界上第一台计算机。

本章重点

本章将介绍面向对象编程（Object Oriented Programming，OOP）的基本结构。

首先，我们将再次介绍一下之前提到的类、多态和继承这三种结构，并讨论实例变量与传统的全局变量和局部变量的区别，以及类类型的作用。然后，我们将简单地介绍一下进化的 OOP 结构，即包、异常和垃圾回收的相关内容。

这里我们将尽量避免拿现实世界的事物来打比方，而是单纯地将 OOP 作为一种编程架构，来看一下它与之前的语言相比具有哪些优势。一些熟悉 OOP 的读者可能会认为没有必要专门花篇幅来介绍这些理所当然的内容，但笔者只是想借此机会和大家一起思考一下 OOP 取得了哪些进步。

4.1 OOP 具有结构化语言所没有的三种结构

OOP 具有之前的编程语言所没有的三种优良结构，分别是类、多态和继承。在 OOP 刚开始普及的 20 世纪 90 年代，它们经常被称为 OOP 的三大要素[1]。

在上一章的结尾，我们介绍了结构化语言无法解决的两个问题，一个是全局变量问题，另一个是可重用性差的问题。

而 OOP 的这三种结构正好可以解决这两个问题。

① OOP 具有不使用全局变量的结构。
② OOP 具有除公用子程序之外的可重用结构。

稍微延伸一下，也可以说 OOP 的三种结构是“去除程序的冗余、进行整理的结构”。

① 一般认为 OOP 的三大要素分别是封装、多态和继承。不过，由于类不只具有封装的作用，所以本书中改为使用类。

打个比方，那些让人无从着手、难以理解的程序就像是一个乱七八糟的房间。由于无法马上在这样的房间里找到需要的东西，所以我们很有可能会再次购买，或者即使将房间里的某一处整理干净，周围也依然是乱作一团。如果要保持房间整洁，平时就要多加注意，此外还需要使用清理不必要物品（去除冗余）的吸尘器和规整必要物品（整理）的收纳架（图 4-1）。

图 4–1　为了保持房间整洁，需要吸尘器和收纳架

OOP 的三种结构为程序员提供了去除冗余逻辑、进行整理的功能结构。

类结构将紧密关联的子程序（函数）和全局变量汇总在一起，创建大粒度的软件构件。通过该结构，我们能够将之前分散的子程序和变量加以整理。多态和继承能够将公用子程序无法很好地处理的重复代码进行整合，彻底消除源代码的冗余。如果我们能使用这些结构开发出通用性较高的功能，就可以实现多个应用程序之间的大规模重用了。

不过类、多态和继承的名称比较特别，虽然这些结构在之前的编程语言中没有过，需要另起名称，但是这样的名称往往会让人感觉面向对象很难。

然而实际上这三种结构是非常接地气的，它们可以说是编程语言领域具有历史性意义的重大发明。也正因为如此，最初设计了这些结构、开发

出最早的面向对象语言 Simula 67 的两名挪威科学家在 2001 年获得了计算机领域的诺贝尔奖——**图灵奖**。另外，开发了 Smalltalk、提出面向对象概念的艾伦·凯也在两年后获得了图灵奖。

下面就让我们一起来揭开这项重大发明的神秘面纱吧。

4.2 OOP 的结构会根据编程语言的不同而略有差异

Java、Ruby、C#、Visual Basic.NET、Objective-C、C++ 和 Smalltalk 等语言都属于 OOP，但它们的功能和语法却不尽相同。接下来我们将使用简单的 Java 示例代码来介绍 OOP 的基本功能，关于各编程语言的详细结构，请大家自行查看其语言规范。

本章以具有打开或关闭文件、读取一个字符等简单功能的文件访问处理作为示例。虽然示例代码使用 Java，但是为了方便起见，有时也会使用 Java 不支持的语法，所以书中有些代码是无法被 Java 编译器编译的，还请大家理解。

4.3 三大要素之一：类具有的三种功能

首先来介绍一下三大要素中的第一个要素——**类**。

这里，我们将类的功能总结为汇总、隐藏和“创建很多个”。

> 类的功能是汇总、隐藏和“创建很多个”。
>
> ① “汇总”子程序和变量。
>
> ② “隐藏”只在类内部使用的变量和子程序。
>
> ③ 从一个类“创建很多个”实例。

类结构本身并不难，但它却能给编程带来很多积极的效果，可以说是一种非常强大的结构。接下来将依次对类的上述三个功能进行介绍。

4.4　类的功能之一：汇总

代码清单 4.1 中定义了 openFile（打开文件）、closeFile（关闭文件）和 readFile（读取一个字符）这三个子程序及一个全局变量[1]。下面我们使用类的功能来逐步修改该程序。

代码清单4.1　采用结构化编程的文件访问处理

```
// 存储正在访问的文件编号的全局变量
int fileNO;

// 打开文件的子程序
//（通过参数接收路径名）
void openFile(String pathName){ /* 省略逻辑处理 */ }

// 关闭文件的子程序
void closeFile() { /* 省略逻辑处理 */ }

// 从文件读取一个字符的子程序
char readFile() { /* 省略逻辑处理 */ }
```

首先来看一下汇总功能。

类能汇总变量和子程序。这里所说的变量是指 C 和 COBOL 等语言中的全局变量。OOP 中将由类汇总的子程序称为**方法**，将全局变量称为**实例变量**（又称为“属性”“字段”），之后我们会根据情况使用这些术语。

下面就让我们使用类来汇总代码清单 4.1。在 Java 中，创建类时会在开头声明类名，并用大括号将汇总范围括起来。由于这里是将读取文本文件的子程序群和变量汇总到一起，所以我们将类命名为 TextFileReader，如代码清单 4.2 所示。

① 为了方便起见，这里定义了独立的全局变量和子程序，但实际上，Java 语法中并不允许定义不属于类的变量和子程序。

代码清单4.2 使用类进行汇总

```
class TextFileReader {
  // 存储正在访问的文件编号的变量
  int fileNO;

  // 打开文件
  //（通过参数接收路径名）
  void open(String pathName) { /* 省略逻辑处理 */ }

  // 关闭文件
  void close() { /* 省略逻辑处理 */ }

  // 从文件读取一个字符
  char read() { /* 省略逻辑处理 */ }
}
```

看到这里，估计有不少读者认为这只不过是将子程序和变量汇总在一起而已。实际上确实如此，但汇总和整理操作本身就是有价值的。打个比方，在收拾乱七八糟的屋子时，与其只准备一个大箱子，不如准备多个箱子分别存放衣服、CD、杂志、文具和小物件等，这样会更方便拿取物品，两者是同样的道理。

由于代码清单 4.2 的示例非常小，所以大家可能感受不到汇总的效果。请大家想象一下企业基础系统中使用的大规模应用程序，其代码往往有几十万到几百万行。如果使用 C 语言平均为每个子程序编写 50 到 100 行代码，那么子程序就有几千到几万个。如果使用 OOP 平均在每个类中汇总 10 个子程序，那么类的总数就比子程序的总数减少 1 位，为几百到几千个（图 4-2）。当然，即便如此数量依然庞大，但所有构件的数量削减 1 位的效果是不容小觑的（另外，关于为了进一步整理大规模的软件而将多个类进行分组的“包”功能，我们将在后文中进行介绍）。

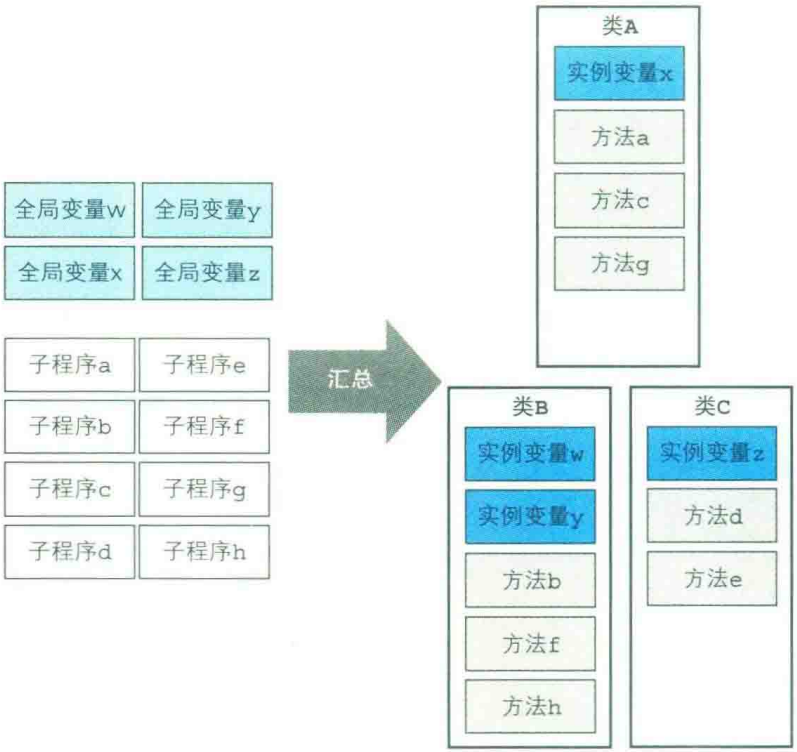


图 4–2　类的功能之一：汇总

汇总的效果并不只是减少整体构件的数量。我们再来比较一下代码清单 4.1 和代码清单 4.2。其实在汇总到类中时我们还略微进行了修改，大家注意到了吗？

没错，子程序的名称改变了。在代码清单 4.1 中，子程序的名称是 openFile、closeFile 和 readFile，都带有 File。而在代码清单 4.2 中去掉了 File，子程序的名称分别为 open、close 和 read。

后一种命名方式显然更轻松。在没有类的结构化语言中，所有子程序都必须命名为不同的名称，而类中存储的元素名称只要在类中不重复就可以。在代码清单 4.2 中，我们将类命名为 TextFileReader，声明该类是用于读取文件的，因此就无须在各个方法的名称中加上 File 了。这样一来，即使其他类中也有 open、read 和 close 等同名的方法，也不会发

生什么问题。举例来说，一个家庭中所有成员的名字应该都不相同，但是和姓氏不同的邻居家的家庭成员重名则没有关系（类名冲突可以使用包进行回避，我们将在本章后半部分介绍包的相关内容）。

为汇总后的类起合适的名称也便于查找子程序。虽然这个步骤看起来并不起眼，但它却是促进可重用的重要功能之一。无论编写的子程序的质量多高，如果因为数量太多而难以查找和调用，那么也是没有任何意义的。反之，如果编写的子程序便于查找，那么对其进行重用的机会也会增加。

我们来总结一下汇总功能。

<类的功能之一：汇总>

能够将紧密联系的（多个）子程序和（多个）全局变量汇总到一个类中。

优点如下。

- 构件的数量会减少
- 方法（子程序）的命名变得轻松
- 方法（子程序）变得容易查找

4.5 类的功能之二：隐藏

接下来我们看一下隐藏功能。

在代码清单 4.2 中，子程序和全局变量都汇总到了类中，但是在这种状态下，从类的外部仍然可以访问 `fileNO` 变量[1]。`TextFileReader` 类的 `open`、`read` 和 `close` 方法会访问 `fileNO` 变量，但其他处理则无须访问，因此最好限定为只有这三个方法能访问该变量。这样一来，当

① 准确来说，根据 Java 语法，代码清单 4.2 中的 `fileNO` 变量和三个方法只可以被同一个包中的类自由访问。