还吸收了之前的技术，逐渐成长为强大的技术。

在这种情况下，用一句话来表示面向对象，那就是“让软件开发变轻松的综合技术”。如今，面向对象成为这样一门技术，正是其目标范围大幅扩展的结果。

12.2　时代追上了面向对象

尽管面向对象是一门非常优秀的技术，但它开始渗入软件开发现场也是最近的事情。Java 出现于 1995 年，但当初只有为了表示动态内容而从网络上下载运行的 Applet 受到关注。Java 作为企业基础系统的开发语言真正普及是在 2000 年之后。

不过，最初的面向对象语言 Simula 67 出现于 1967 年，说起来已经被埋没了近 30 年，得益于 Java 的出现，Simula 67 终于重见天日了。

为什么会这么晚呢？用一句话来形容这一情况，那就是“时代追上了面向对象”。

在 1967 年的时候，计算机是一般人很难拥有的非常昂贵的机器。之后虽然硬件性能每年都有显著提高，但是直到 20 世纪 90 年代中期，像 Java 那样采用中间代码方式[1]的运行环境高效运行、具备 GUI 的高性能开发环境高速动作的硬件才开始普及。

在这期间，在软件开发技术领域，高级语言得以普及，结构化编程被提出，各种设计技巧和图形表示也应运而生。在面向对象领域，类库和框架等大规模可重用构件群、作为固定的设计思想的设计模式，以及作为 UML 前身的图形表示等纷纷被提出，并不断改善。这些技术在 20 世纪 90 年代中期之前悄无声息地进行了大幅进化，以 1995 年 Java 的出现为契机，很多技术一下子普及开来。

不过，面向对象编程语言的基本功能与 1967 年出现时相比并未发生改

① 中间代码方式是指在编译阶段将代码转换为不依赖于运行环境的中间代码，并将其在虚拟机上运行的方式。详细介绍请参照第 5 章。

变。虽然出现得很早，但是经过长期沉寂后才终于迎来绽放，正可谓“时代追上了面向对象”。

12.3 面向对象的热潮不会结束

如今，面向对象已成为软件开发领域不可或缺的技术，这种现状会持续到什么时候呢？毕竟在软件领域，新技术时而出现时而消失也是常有的事。

不过，面向对象不会掀起一阵热潮后就凄凉收场，今后 10 年仍会继续占据主导地位。

虽然应用面向对象的产品和技术就像雨后春笋一样不断涌现，但面向对象的根本内容在过去 20 多年基本上没有什么变化。编程语言的结构由 1967 年的 Simula 67 和 20 世纪 70 年代的 Smalltalk 确立，类库和框架等大规模软件构件群在 20 世纪 80 年代出现，设计模式和作为 UML 前身的图形表示在 20 世纪 90 年代初出现。如今的面向对象只不过是对它们进行了提炼和普及而已。在今后 10 年，即使这些技术再进行扩展，其基本结构也不会发生较大改变。

面向对象之后出现的面向切面、面向代理和面向服务等技术也曾受到过人们的关注。

面向切面通过将分散在程序各个部分的共同处理分离出来编写，从而进一步提高软件灵活性。现在已经有了使用 Java 的 AspectJ 实现，但其使用范围有限。

对象作为被动的存在，只有在接收到外部消息时才开始工作，而**面向代理**则打破了这一限制，以能动地进行活动的代理为中心编写软件。不过，“自律的代理”这一概念还未具体实现。

面向服务（Service-Oriented Architecture，SOA）将复杂、庞大的系统变为独立性较高的子系统的松散耦合结构，这一概念非常吸引人，受到了很多关注。不过，由于具体的实现技术只有 Web 服务和企业服务总线（Enterprise Service Bus，ESB），面向对象目前还没有脱离潮词的阶段。

即使超过面向对象的下一代技术普及了，那么它可能也是面向对象的延伸，或者会与面向对象共存。因此，学习面向对象一定不会白学。

也就是说，面向对象是 10 年后仍然通用的技术。在斗转星移的软件领域，这么稳定的技术可以说是一个奇迹。

12.4　将面向对象作为工具熟练掌握

到这里为止，本书以轻松的口吻介绍了面向对象技术。而只阅读本书还不够，大家一定要亲自动手试一试。

为了充分理解类、多态和继承等结构，大家最好使用 Java 或 C# 等进行编程，并使用调试器试着运行一下。关于 UML，最开始不必记住所有的图形，为了理解既有程序，大家可以先使用类图和时序图。

面向对象不是一个空泛的理论，而是可以立马使用的实用技术。如果大家自己动手去使用，就会发现各个技术其实并没有那么难。

另外，以笔者的经验来看，在使用面向对象的技术时，应该注意使用面向对象本身不是我们的目的。

我们在第 4 章最后提到了“决心决定 OOP 的生死”，这并不是仅针对编程来说的，设计模式、UML、建模和迭代式开发流程等都是用于轻松编写高质量软件的工具，但仅使用这些技术，并不一定能提高软件的可维护性和可重用性。我们的目的是编写出质量高、可维护性强、易于重用的软件，这一点很重要。

另外，这些技术也都非常有趣，大家在了解之后可能就会想方设法地去使用，但是，如果在不合适的地方使用了设计模式，或者使用 UML 绘制了大量无用的成果，反而会导致系统难以理解和维护，这样就本末倒置了。特别是在评审设计和代码时，如果反复出现“这好像不是面向对象的设计”的发言，那就需要注意了。这样的话，面向对象就不再是工具，而成了目的。在这种情况下，大家一定要再次确认一下本来的目的是什么。

12.5 享受需要动脑的软件开发

软件开发是一项需要动脑的有趣的工作。

从零开始编写程序后，立即将其在计算机上运行并确认结果，这真的是乐趣无穷。在和团队成员一起展开工作的情况下，系统完工时的集体荣誉感也很有魅力。

面向对象就是让这种软件开发工作更加有趣的技术。使用类、多态和继承的编程可以给人带来愉悦感，设计模式会让人产生解谜一样的乐趣。使用 UML 的建模是一项从无到有一边画图一边整理需求的工作，有着类似于作曲或者绘画的创造性。通过实践敏捷开发流程，团队也变得活跃起来。

到这里，面向对象的学习旅程就要结束了。很多人说面向对象很难，但事实绝不是这样。面向对象是让我们更轻松地进行软件开发的工具，是凝聚了前人智慧的技术窍门集。

另外，面向对象不仅方便，还会刺激人们的好奇心，是一门非常有趣的技术。熟练掌握该技术之后，需要开动脑筋的软件开发就会变得更加轻松。如果本书能够给大家提供哪怕一丁点帮助，笔者也会非常开心。

深入学习的参考书籍

这里为大家介绍一些有助于实际动手学习面向对象的参考书籍。

[1] 牛尾刚. オブジェクト脳のつくり方——Java · UML · EJB をマスターするための究極の基礎講座 [M]. 东京：翔泳社，2003.

☆☆

为了快速掌握面向对象，相比在最开始就学习大量理论，在动手实践的过程中逐渐培养感觉更加重要。为了帮助读者切身体验多态结构，书中进行了“总经理命令，起立！”式的演练。该书以新颖的视角介绍面向对象，是应该常备身边的一本书。

[2] Kathy Sierra，Bert Bates. Head First Java（中文版）[M]. 杨尊一，译. 北京：中国电力出版社，2007.

☆☆

可以说这本书是上面的参考书籍 [1] 的海外版。除了 OOP 的基本结构之外，该书还全面涵盖了 Java 的规格说明，如异常、垃圾回收、Swing、文件处理、线程、泛型和 RMI 等。除该书之外，Head First 系列还出版了许多书目，由此也可以看出，欧美人也在认真地学习面向对象等技术。

第13章 函数式语言是怎样工作的

本章的关键词

函数、表达式、头等函数、部分应用、高阶函数、函数组合、副作用、延迟求值、模式匹配、类型推断

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

下面哪项是纯函数式语言 Haskell 不支持的结构（可多选）？

A. 返回值是 `void` 型的函数定义

B. 使用 `return` 语句从函数返回

C. 改写局部变量

D. 使用 `for` 语句的循环处理

A. 返回值是 void 型的函数定义

B. 使用 return 语句从函数返回

C. 改写局部变量

D. 使用 for 语句的循环处理

解 析

函数式语言的结构与被称为“命令式语言”的传统编程语言有很大不同，特别是属于函数式语言中的纯函数式语言的 Haskell 不支持 A~D 的结构。

首先，Haskell 不能定义返回值为 void 型的函数，这是因为它的一个原则是函数必须有返回值。另外，由于函数的返回值最后将转化为求值表达式，所以并不存在显式的 return 语句。不管是局部变量还是全局变量，变量的内容都不可以修改。循环处理使用模式匹配和递归，因此并不提供 for 语句或者 while 语句等基本语法。

本章重点

本章将为大家介绍函数式语言的概要和基本结构。

函数式语言作为面向对象的下一代开发技术而受到了许多技术人员的关注，但由于其基本思想与传统编程语言存在很大不同，所以对熟悉 C 语言和 Java 语言的人来说，函数式语言中的很多结构都难以理解。本章将针对函数式语言特有的结构和术语，使用最简单的示例代码进行说明。不过，因为是编程语言的结构，所以请感兴趣的读者在读完本章之后，一定要下载 Haskell 或者 Scala 等开发环境，实际动手操作一下试试。

13.1 面向对象的“下一代”开发技术

在企业系统开发领域，2000 年之后，Java 和 C# 等面向对象语言成为主流。然而，不少研究人员和技术人员对此并不满足，他们针对面向对象的下一代技术，提出了各种各样的开发方法和解决方案，包括面向组件、面向代理、面向切面和面向服务等。不过，其中许多技术因为达不到概念要求或者应用范围有限等，并未取得像面向对象那么大的成功。

由于函数式语言是编程语言的一种，而编程语言可以说是软件开发中最重要的技术，所以函数式语言不会成为只有概念流行的潮词。回顾过去的大趋势，结构化方法和面向对象都是以编程语言为核心的技术，鉴于这一点，函数式语言或许可以说是下一代开发技术的有力候补。

不过，由于函数式语言朴素的概念没有引起系统用户和客户企业的兴趣，所以在市场和销售的场合中基本上看不到其身影。

话虽如此，近年来，该技术也开始受到许多技术人员的关注，相关图书出版和社区活动等也开始盛行。今后，如果实际的应用程序案例增加，对象领域扩展到上流工程和开发流程等，那么它可能也会被用作意为“大幅提高生产率的开发技术”的潮词。就像随着互联网的普及而出现的 Java

一样，如果出现符合时代潮流的技术，就可能迅速普及。

即使不会立即使用函数式语言，通过掌握该技术，也能加深对编程的理解。建议大家务必借此机会掌握它。

13.2　函数式语言的 7 个特征

下面我们来介绍一下函数式语言的结构。

虽然统称为函数式语言，但其实存在各种语言，它们的具体结构和语法都各不相同。为了介绍函数式语言与传统语言的不同，这里将以仅具有函数式语言性质的编程语言——**纯函数式语言**[1] 为例，对其 7 个常见特征依次进行说明。

特征 1：使用函数编写程序
特征 2：所有表达式都返回值
特征 3：将函数作为值进行处理
特征 4：可以灵活组合函数和参数
特征 5：没有副作用
特征 6：使用分类和递归来编写循环处理
特征 7：编译器自动进行类型推断[2]

虽然示例代码使用的是 Haskell 和 Java，但之后的讲解都是函数式语言的一般性质。另外，为了增强可读性，我们将最低限度地使用代码进行介绍。

13.3　特征 1：使用函数编写程序

如果用一句话来表示函数式语言，那就是“使用函数编写程序的结构”。

Java 和 C# 等面向对象语言中使用汇总子程序和变量的类来编写程序，

① 函数式语言中包含仅具有函数式语言结构的“纯函数式语言”和同时具有传统语言结构的“非纯函数式语言”。关于函数式语言的分类，我们将在后文中详细介绍。

② 类型推断是强类型语言中常备的结构。

而函数式语言则使用函数来编写程序。

> 函数式语言中使用函数来编写程序。

看到这里，有些读者可能会感到不可思议。使用函数进行编程的结构，与面向对象之前的 C 语言是一样的。另外，在面向对象编程中，编写不属于类的独立函数，通常会被认为是不妥当的做法。因此，函数在面向对象的下一代开发技术中承担起主要作用，可能确实会让人感到奇怪。

不过，虽说是“函数”，但是在传统语言和函数式语言中的定义有很重要的区别。

在传统编程语言中，函数通常指一连串的步骤。该词原本是指返回返回值这一过程，在有的编程语言中，将持有返回值的过程称为“函数”，将不持有返回值的过程称为“过程”。而在广泛普及的 C 语言中，不管有没有返回值，都称为函数。现在，“函数”一词通常作为子程序或过程的同义词使用。

函数式语言中的函数结构与数学中的函数基本相同。

$$y=f(x)$$

该式的意思是，函数 f 将 x 值转换为 y。与此相同，函数式语言中的函数也是指将参数中指定的值转换为返回值的结构。因此，函数式语言中的函数必须持有参数和返回值。

> 函数式语言中的函数必须持有参数和返回值。

函数式语言中使用将参数转换为返回值的函数来编写应用程序，因此，整体结构就变成由一系列函数不断对值进行转换的网络结构（图 13-1）。

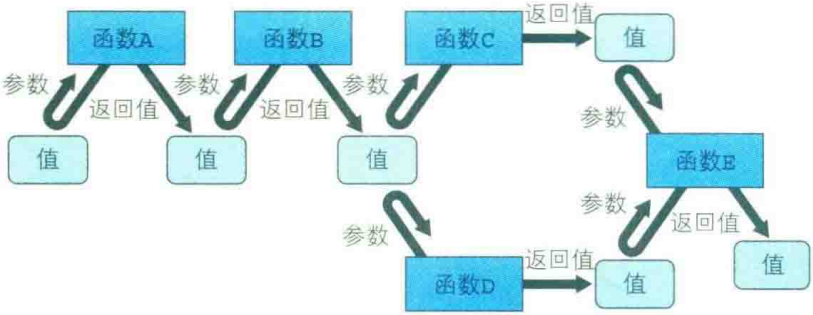

图 13–1　使用函数网络构成应用程序

函数式语言的这一特征还表现在函数调用的表述上。在传统编程语言中，函数调用一般表述为“指定参数调用函数”，而在函数式语言中，则表述为“对参数应用函数”。这反映了函数式语言中的函数纯粹是转换参数来获得返回值的结构。

> 在函数式语言中，函数调用表述为“对参数应用函数”。

13.4　特征 2：所有表达式都返回值

在传统编程语言和函数式语言中，程序的基本构成元素的名称也不一样。前者称为**命令语句**（statement），而后者则称为**表达式**（expression）。下面我们就来介绍一下两者的区别。

正如第 3 章中介绍的那样，编程语言是按照“机器语言→汇编语言→高级语言→结构化语言→面向对象语言”的顺序进化的。这基本上是将计算机能够直接运行的机器语言命令替换为人类容易理解的符号和语句，将多个机器语言命令汇总为高级命令的过程。因此，即使是 Java 等面向对象语言，其中的各个命令语句也都相当于对机器语言的命令语句进行了抽象。

代码清单13.1 命令式语言的代码示例

```
①    int amount;                  // 声明变量
②    amount = price * count;      // 将计算结果存储到变量中
③    transaction.commit();        // 调用过程
```

上面是 Java 的代码示例，分别表示为各个变量分配内存区域（①）、将计算结果存储到内存区域中（②）和调用过程（③），都表现为让计算机执行什么样的处理。

这种由命令语句构成的编程语言，包括汇编语言、高级语言、结构化语言及面向对象语言在内，都称为**命令式语言**[1]。

> 由命令语句构成的编程语言称为命令式语言。

而函数式语言中的情形则稍有不同。在函数式语言中，程序的构成元素是表达式，而非命令语句。我们在前面介绍过，在函数式语言中，函数必须返回返回值。同样，表达式也必须返回值。

代码清单13.2 函数式语言（Haskell）中的表达式示例

```
④  max a b
⑤  x + 3
⑥  7
⑦  "abc"
```

在以上 Haskell 的代码示例中，④～⑦分别表示函数应用（④）[2]、算术计算（⑤）[3] 和数据（⑥⑦），每个表达式都会返回值。也就是说，④是返回函数应用结果的表达式，⑤是返回计算结果的表达式，⑥和⑦分别是返回

① 命令式语言也称为过程式语言。

② ④的表达式表示对 a 和 b 这两个参数应用 max 函数。

③ 实际上，⑤的表达式表示对 x 和 3 这两个参数应用 + 函数。

7 和 "abc" 的值本身的表达式。请注意，这里的 7 和 "abc" 等数据也是表达式。在函数式语言中，不管是数据还是函数，表达式一定会返回值。

在命令式语言和函数式语言中，表示程序运行的术语也不一样。前者表述为**执行**（execute）命令，而后者则与数学术语一样，表述为对表达式进行**求值**（evaluate）。这是因为函数式语言是将数学的函数和算式表示为程序的语言。

函数式语言是由表达式构成的。如果对表达式进行求值，就会返回值。

这种不同可以说是命令式语言和函数式语言的本质区别。在命令式语言中，通过按顺序执行在内存中展开的命令，各个命令读写内存区域中存储的数据来实现程序[1]。而在函数式语言中，则是通过依次对各个表达式进行求值，再对使用所得到的值的其他表达式进行求值，从而实现程序。

在使用命令式语言的情况下，在程序运行时，我们需要十分注意计算机按什么顺序执行命令，各个命令如何读写内存。而在使用函数式语言的情况下，相比计算机的动作，在程序运行时，我们更要注意应用函数时值是怎样转换的。两种语言的不同之处体现在命令式语言是过程式的，而函数式语言是声明式的。

熟悉 Java 和 C 语言等命令式语言的人在初次阅读使用函数式语言编写的程序时，会有一个困惑，那就是尽管函数必须有返回值，但却没有 return 语句[2]。这是因为，函数式语言必须返回值，所以规定最后求得的表达式的值就是函数的返回值。

> 函数的返回值就是最后求得的表达式的值。因此，函数式语言中通常并没有 return 语句。

在函数式语言中，不管是函数还是数据，都被作为“返回值的表达式”处理，因此，它们的区别比较模糊。实际上，在许多函数式语言中，将值

① 现在实用的计算机的运行原理以发明者冯·诺依曼的姓名命名，称为“诺依曼计算机”。

② 在 Ruby 中，在方法内最终求得的值就是返回值，因此可以省略 return 语句。

存储到变量中的语法与函数定义的语法类似。Haskell 的代码示例如下所示。

代码清单13.3 Haskell中的变量定义与函数定义的代码示例

```
① value       = 1
② increment x = x + 1
③ increment   = λx -> x + 1
```

最开始的①是将 1 存储到变量 value 中。②中定义了一个参数为 x 的 increment 函数，主体代码为 x + 1[1]。③的表达式是使用**λ表达式（lambda 表达式）**的特殊写法来写②的函数定义，λ x -> x + 1 表示一个表达式[2]。像这样，在 Haskell 中，将值存储到变量中与函数定义都使用“=”。请特别注意①和③中在名称后面都紧跟着“=”。数据存储（①）和函数定义（③）都是将表达式存储到变量中，这一点是相同的。

在 Haskell 中，即使在执行函数的情况下，也会像下面这样不加括号。例如，对参数 8 应用 increment 函数的表达式如下所示。

代码清单13.4 Haskell中的函数应用的代码示例

```
increment 8
```

在 Java 和 C 语言等命令式语言中，通常都在过程调用中加上括号 ()，以在语法上明确区分数据和过程。而在很多函数式语言中，数据和函数的写法是一样的。

这是因为在函数式语言中，虽然函数和数据存在是否需要参数的区别，

① 在②和③中，虽然并未标明变量 x 为数值型，但根据 Haskell 中的类型推断，x 会被作为数值型处理。关于类型推断，我们将在特征 7 中介绍。

② 像代码清单 13.3 中的③的表达式那样，使用字符 λ 写的函数称为 λ 表达式。λ 表达式是 20 世纪 30 年代提出的数学概念，函数式语言就是用 λ 表达式表示的编程语言。使用 λ 表达式，我们可以定义没有名称的函数，并将其称为匿名函数。

但是它们都被看作表达式。也就是说，函数是随着参数返回值的表达式，数据是直接返回值的表达式。

> 在函数式语言中，函数和数据都被看作返回值的表达式。

13.5　特征 3：将函数作为值进行处理

如前所述，在函数式语言中，函数必须返回值，除此之外，其本身也可以作为值进行处理。也就是说，我们可以将函数本身存储到变量中，或者包含在集合中，还可以将其指定为其他函数的参数或返回值。这种函数称为**头等函数**（first-class function）。

> 在函数式语言中，我们可以将函数作为值进行处理。

函数可以接收其他函数作为参数的结构对编程来说具有十分重要的意义。这是因为在接收函数作为参数的一端可以执行该函数。使用该结构，在整体处理相同的情况下，如果想将一部分处理根据情况进行切换，就可以简洁地编写该处理（图 13-2）。

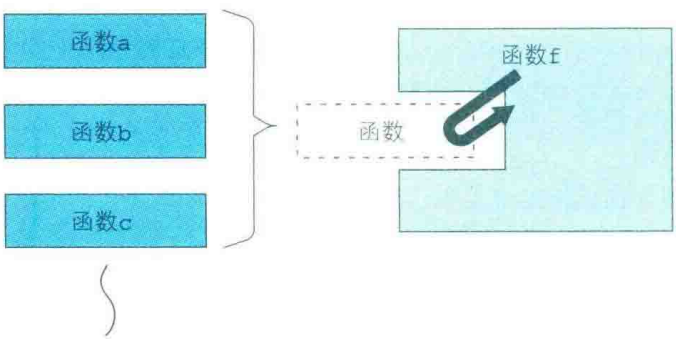

图 13–2　可以执行作为参数接收的函数