第 1 章　面向对象开发方法概述

一般说来，软件开发都会经历以下生命周期：

- 软件分析：分析问题领域，了解用户的需求。
- 软件设计：确定软件的总体架构，把整个软件系统划分成大大小小的多个子系统，设计每个子系统的具体结构。
- 软件编码：用选定的编程语言来编写程序代码，实现在设计阶段勾画出的软件蓝图。
- 软件测试：测试软件是否能实现特定的功能，以及测试软件的运行性能。
- 软件部署：为用户安装软件系统，帮助用户正确地使用软件。
- 软件维护：修复软件中存在的 Bug，当用户需求发生变化时（增加新的功能，或者修改已有功能的实现方式），修改相应的软件。

为了提高软件开发效率，降低软件开发成本，一个优良的软件系统应该具备以下特点：

- 可重用性：减少软件中的重复代码，避免重复编程。
- 可扩展性：当软件必须增加新的功能时，能够在现有系统结构的基础上，方便地创建新的子系统，不需要改变软件系统现有的结构，也不会影响已经存在的子系统。
- 可维护性：当用户需求发生变化时，只需要修改局部的子系统的少量程序代码，不会牵一发而动全身，修改软件系统中多个子系统的程序代码。

Tips

本章多次使用了系统结构的说法，这里的系统结构是指系统由多个子系统组成，以及子系统由多个更小的子系统组成的结构。

如何才能使软件系统具备以上特点呢？假如能把软件分解成多个小的子系统，每个子系统相对独立，把这些子系统像搭积木一样灵活地组装起来就构成了整个软件系统，这样的软件系统便能获得以上优良特性。软件中的子系统具有以下特点：

- 结构稳定性：软件在设计阶段，在把一个系统划分成更小的子系统时，设计合理，使得系统的结构比较健壮，能够适应用户变化的需求。
- 可扩展性：当软件必须增加新的功能时，可在现有子系统的基础上创建出新的子系统，该子系统继承了原有子系统的一些特性，并且还具有一些新的特性，从而提高软件的可重用性和可扩展性。
- 内聚性：每个子系统只完成特定的功能，不同子系统之间不会有功能的重叠。为了避免子系统之间功能的重叠，每个子系统的粒度在保持功能完整性的前提下都尽可能小，按这种方式构成的系统结构被称为精粒度系统结构。子系统的内聚性会提高软件的可重用性和可维护性。

- 可组合性：若干精粒度的子系统经过组合，就变成了大系统。子系统的可组合性会提高软件的可重用性和可维护性，并且能够简化软件的开发过程。
- 松耦合：子系统之间通过设计良好的接口进行通信，但是尽量保持相互独立，修改一个子系统，不会影响到其他的子系统。当用户需求发生变化时，只需要修改特定子系统的实现方式，从而提高软件的可维护性。

Tips

在精粒度系统结构中，大系统可以分解为多个松耦合的精粒度子系统。与此相反，在粗粒度系统结构中，粗粒度的大系统无法进一步分解，或者只能被分解为有限的几个粗粒度子系统。

图 1-1 显示了一个用积木搭建起来的建筑物系统，有凯旋门，还有小狗狗的家。这些系统中的最小子系统是各种形状的积木（精粒度的子系统），这些积木能够在多个子系统中重用，多个积木能组合成复杂的子系统。当大的系统被拆散或摧毁时，它所包含的小子系统依然有用。

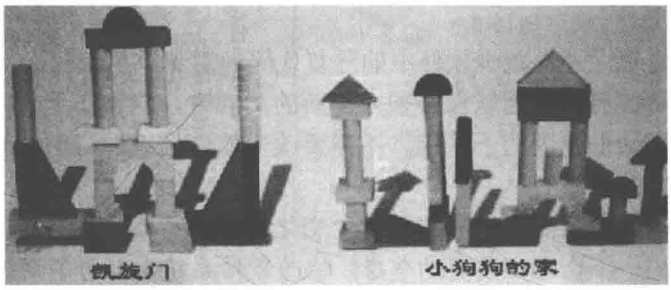

图 1-1　用积木搭建起来的建筑物系统

目前在软件开发领域有两种主流的开发方法：结构化开发和面向对象开发。结构化开发是一种比较传统的开发方法，早期的高级编程语言，如 Basic、C、Fortran 和 Pascal 等，都是支持结构化开发的编程语言。随着软件开发技术的逐步发展，为了近一步提高软件的可重用性、可扩展性和可维护性，面向对象的编程语言及面向对象设计理论应运而生，Java 语言就是一种纯面向对象的编程语言。

本章首先简要介绍结构化的软件开发过程，然后介绍了面向对象的软件开发过程，对面向对象的一些核心思想和概念进行了阐述。本章列举了不少形象的例子，来帮助读者理解面向对象的开发思想，并且以一个画板（Panel）软件系统的例子贯穿整个章节，这个例子分别按照结构化开发方式和面向对象开发方式实现，从而鲜明地对比这两种开发方式对软件的可维护性、可扩展性和可重用性的影响。

本章的思想性和理论性比较强，如果读者已经有一定的面向对象的开发经验，阅读本章时会很顺利。如果读者没有面向对象的开发经验，初次阅读时会感觉比较抽象，在这种情况下，初次阅读时只需了解一些基本概念与核心思想即可，等到阅读完全书后，再来回顾和领悟本章内容。

本章在介绍范例时，展示了部分 Java 程序代码，如果读者对 Java 编程没有任何基

础，那么在初次阅读本章时不必深入探究这些 Java 程序代码，可在阅读完本书后再回顾它们。

1.1　结构化的软件开发方法简介

1978 年，E.Yourdon 和 L.L.Constan-tine 提出了结构化开发方法，即 SASD 方法。1979 年，Tom DeMarco 对此方法做了进一步的完善。SASD 方法是 20 世纪 80 年代使用最广泛的软件开发方法。它首先用结构化分析（SA，Structure Analysis）对软件进行需求分析，然后用结构化设计（Structure Design，SD）方法进行总体设计，最后是结构化编程（Structure Programming，SP）。这种开发方法使得开发步骤明确，SA、SD 和 SP 相辅相成，一气呵成。

结构化开发方法主要是按照功能来划分软件结构的，它把软件系统的功能看作是根据给定的输入数据，进行相应的运算，然后输出结果，如图 1-2 所示。

图 1-2　结构化开发中的软件功能

进行结构化设计时，首先考虑整个软件系统的功能，然后按照模块划分的一些基本原则（比如内聚性和松耦合）等，对功能进行分解，把整个软件系统划分成多个模块，每个模块实现特定的子功能。为了提高软件的内聚性，在模块中还会把功能分解到更小的子模块中。在完成所有的模块设计后，把这些模块拼装起来，就构成了整个软件系统。软件系统可看作是多个子系统的集合，每个子系统都是具有输入/输出功能的模块，如图 1-3 所示。

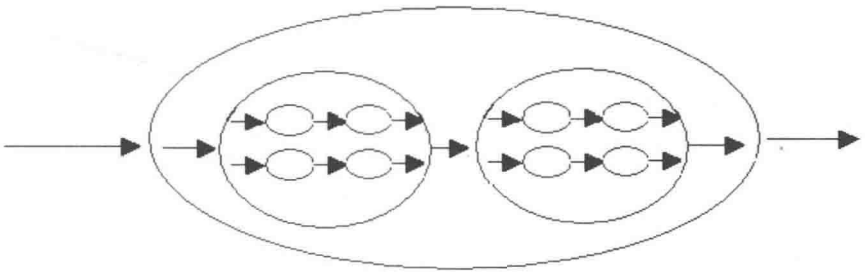

图 1-3　按照功能划分成多个子系统的软件系统结构

结构化设计属于自顶向下的设计，在设计阶段就不得不考虑如何实现系统的功能，因为分解功能的过程其实就是实现功能的过程。结构化设计的局限性在于不能灵活地适应用户不断变化的需求。当用户需求发生变化，比如要求修改现有软件功能的实现方式或者要求追加新的功能时，就需要自顶向下地修改模块的结构，有时候甚至整个软件系统的设计被完全推翻。

在进行结构化编程时，程序的主体是方法，方法是最小的功能模块，每个方法都是具有输入/输出功能的子系统，方法的输入数据来自于方法参数、全局变量和常量，方法的输出数据包括方法返回值，以及指针类型的方法参数。一组相关的方法组合成大的功能模块。

Tips

结构化编程刚出现的时候，主要通过编写实现各种功能的函数（也称作过程）来实现，故也称为“面向过程编程（Procedure Oriented Programming）”。这与本章 1.2 节中介绍的面向对象编程的说法比较对应。

下面举例说明结构化开发的过程。如图 1-4 所示显示了一个按照功能划分的画板（Panel）系统的结构。

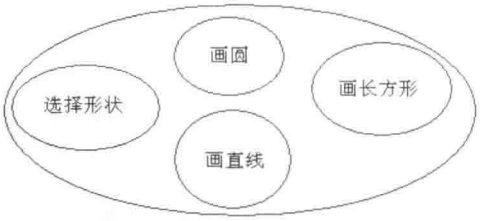

图 1-4　按照功能划分的 Panel 系统的结构

从图 1-4 可以看出，Panel 系统包括 4 个功能模块：选择形状模块、画长方形模块、画圆模块和画直线模块。如图 1-5 所示为选择形状模块的数据流图（Data Flow Diagram，DFD）。

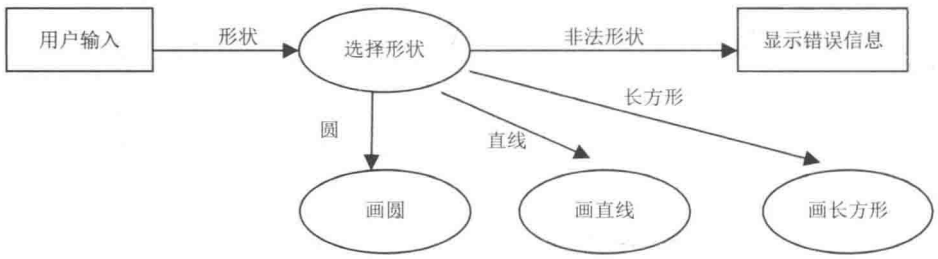

图 1-5　选择形状模块的数据流图

以上数据流图中包括 3 种形状的符号：

- 椭圆：表示处理过程，即功能。输入数据在此进行处理产生输出数据。
- 矩形：数据输入的源点或数据输出的终点。
- 带箭头的直线：数据流的方向。

从图 1-5 看出，用户输入特定形状类性，选择形状模块对此进行判断，如果用户输入的是圆，那么调用画圆模块，以此类推。如果用户输入的不是圆、长方形和直线，就显示错误信息。例程 1-1 是用 C 语言编写的 Panel 系统的结构化源程序，程序中的各个方法（除 main()方法以外）分别实现 4 个功能模块。此外还定义了一些常量，表示

形状名称，这些常量位于系统的全局范围内。

例程 1-1　panel.c

```
#include <stdio.h>

/** 定义常量 */
#define CIRCLE 1
#define RECTANGLE 2
#define LINE 3

/** 画圆模块 */
void drawCircle (){...}          //省略显示实现细节

/** 画长方形模块 */
void drawRectangle (){...}       //省略显示实现细节

/** 画直线模块 */
void drawLine (){...}            //省略显示实现细节

/** 选择形状模块 */
void selectShape(){
  int shape;
  scanf("%d ",&shape);           //接收用户输入的形状
  switch(shape){
    case CIRCLE :
      drawShape();
      break;
    case RECTANGLE :
      drawRectangle();
      break;
    case LINE :
      drawLine();
      break;
    default :
      printf("输入的形状不存在");
      break;
  }
}

/** 程序入口方法 */
void main(){
  selectShape();
}
```

假定需求发生变化，要求增加一个画三角形功能，那么需要对系统做多处改动：

（1）在整个系统范围内，增加一个常量：

```
#define TRIANGLE 4
```

（2）在整个系统范围内增加一个新的画三角形模块，即增加以下方法：

```
drawTriangle(){…};
```

（3）在选择形状模块 selectShape()内增加以下逻辑：

```
case TRIANGLE :
  drawTriangle();
  break;
```

由此可见，结构化开发方法制约了软件的可维护和可扩展性，模块之间的松耦合性不高，修改或增加一个模块，会影响到其他的模块。导致这种缺陷的根本原因在于：

- 自顶向下地按照功能来划分软件模块，而软件的功能不是一成不变的，会随着用户需求的变化而改变，这使得软件在设计阶段就难以设计出稳定的系统结构。
- 软件系统中最小的子系统是方法。方法和一部分与之相关的数据分离，全局变量数据和常量数据分散在系统的各个角落，这削弱了各个系统之间的相对独立性，从而影响了软件的可维护性。

1.2 面向对象的软件开发方法简介

面向对象的开发方法把软件系统看成是各种对象的集合，对象就是最小的子系统，一组相关的对象能够组合成更复杂的子系统。面向对象的开发方法具有以下优点：

- 把软件系统看成是各种对象的集合，这更接近人类认识世界的自然思维方式。
- 软件需求的变动往往是功能的变动，而功能的执行者——对象一般不会有大的变化。这使得按照对象设计出来的系统结构比较稳定。
- 对象包括属性（数据）和行为（方法），对象把数据及方法的具体实现方式一起封装起来，这使得方法和与之相关的数据不再分离，提高了每个子系统的相对独立性，从而提高了软件的可维护性。
- 支持封装、抽象、继承和多态等各种特征，提高了软件的可重用性、可维护性和可扩展性。这些特征将在后面的章节中详述。

Tips

广义地讲，面向对象编程是结构化编程的一种改进实现方式。传统的面向过程的结构化编程的最小子系统是功能模块，而面向对象编程的最小子系统是对象。

1.2.1 对象模型

在面向对象的分析和设计阶段，致力于建立模拟问题领域的对象模型。建立对象模型既包括自底向上的抽象过程，也包括自顶向下的分解过程。

（1）自底向上的抽象。

建立对象模型的第一步是从问题领域的陈述入手。分析需求的过程与对象模型的形成过程一致，开发人员与用户的交谈是从用户熟悉的问题领域中的事物（具体实例）开始的，这就使用户与开发人员之间有了共同语言，使得开发人员能彻底搞清用户需求，然后再建立正确的对象模型。开发人员需要进行以下自底向上的抽象思维：

- 把问题领域中的事物抽象为具有特定属性和行为的对象。比如一个模拟动物园的程序中，存在各种小动物对象，比如各种小猫、小狗等。
- 把具有相同属性和行为的对象抽象为类。比如尽管各种小猫、小狗等对象各自不同，但是它们具有相同的属性和行为，所以可以将各种小猫对象抽象为小猫类，而各种小狗对象抽象为小狗类。
- 当多个类之间存在一些共性（具有相同属性和行为）时，把这些共性抽象到父类中。比如在前面的例子中，小猫类和小狗类又可以进一步归于哺乳动物类。

在自底向上的抽象过程中，为使子类能更合理地继承父类的属性和行为，可能需要自顶向下的修改，从而使整个类体系更加合理。由于这种类体系的构造是从具体到抽象，再从抽象到具体的，符合人类的思维规律，因此能更快、更方便地完成任务。这与自顶向下的结构化开发方法构成鲜明的对照。在结构化开发方法中，构造系统模型是最困难的一步，因为自顶向下的“顶”（即系统功能）是一个空中楼阁，缺乏坚实稳定的基础，而且功能分解有相当大的任意性，因此需要开发人员有丰富的软件开发经验。而在面向对象建模中，这一工作可由一般开发人员较快地完成。

（2）自顶向下的分解。

在建立对象模型的过程中，也包括自顶向下的分解。例如对于计算机系统，首先识别出主机对象、显示器对象、键盘对象和打印机对象等。接着对这些对象再进一步分解，例如主机对象由处理器对象、内存对象、硬盘对象和主板对象等组成。系统的进一步分解因有具体的对象为依据，所以分解过程比较明确，而且也相对容易。所以面向对象建模也具有自顶向下开发方法的优点，既能有效地控制系统的复杂性，又同时避免了结构化开发方法中功能分解的困难和不确定性。

1.2.2　UML：可视化建模语言

面向对象的分析与设计方法，在 20 世纪 80 年代末至 90 年代中发展到一个高潮。但是，诸多流派在思想和术语上有很多不同的提法，对术语和概念的运用也各不相同，统一是继续发展的必然趋势。需要用一种统一的符号来描述在软件分析和设计阶段勾画出来的对象模型，统一建模语言（Unified Modeling Language，UML）应运而生。UML 是一种定义良好、易于表达、功能强大且普遍适用的可视化建模语言。它吸取了诸多流派的优点，而且有进一步的发展，最终成为大众所共同接受的标准建模语言。

1.2.3　Rational Rose：可视化建模工具

Rational Rose 是 Rational 公司开发的一种可视化建模工具，之后归属于 IBM 公司。它采用 UML 语言来构建对象模型，是分析和设计面向对象软件系统的强有力的工具。如图 1-6 所示为 Rational Rose 的界面。本书的许多 UML 图都是用 Rational Rose 来绘制的。但本书并没有介绍 Rational Rose 工具本身的用法，读者可以参考其他相关的书籍来进一步了解它的用法。