# 第5章 程序设计基础

程序的本质是指令序列或集合，为了实现特定目标或解决特定问题而设计出来的，能让计算机执行一个或多个操作。本章通过学习与程序设计语言相关的一些知识，帮助读者养成良好的程序设计风格，并探讨不同程序设计语言之间的共性和个性问题。数据结构是计算机科学中的核心学科，也是计算能力的基石，本章通过学习数据结构，了解数据如何在计算机内部存储以及如何有效地访问和操作这些数据。数据库设计是计算机科学中的一个重要领域，它涉及如何创建、组织和管理数据库系统，以满足特定应用的需求，本章通过数据库设计的学习，程序员可以了解如何创建高效、可靠和可扩展的数据库系统，以满足应用程序对数据存储和访问的需求。软件工程是计算机科学的一个分支，它专注于开发、维护和管理高质量软件的过程和方法。本章通过学习软件工程，可以构建出高质量、可靠和易于维护的软件系统，满足用户的需求并持续地进行改进和优化。

**【知识目标】**

- ◆ 了解程序设计语言的演化过程和编程模式的特点。
- ◆ 掌握高级程序设计语言的基本构成及相关语法知识。
- ◆ 了解算法的相关概念与内容。
- ◆ 掌握数据结构的基本概念以及其应用背景。
- ◆ 理解表结构，如数组和链表的基本概念、特性以及应用场景。
- ◆ 掌握图结构的基本概念，理解图的节点和边的概念，了解图在现实世界中的应用，如社交网络、地图导航等。
- ◆ 掌握数据库的设计原则和步骤。
- ◆ 掌握常用的数据库管理系统，掌握结构化查询语言(SQL)。
- ◆ 了解系统分析、系统设计、系统实现及系统测试的基本概念。
- ◆ 熟悉软件项目管理工具及基本概念。
- ◆ 掌握软件生命周期、系统测试等软件工程的必备知识。

**【能力目标】**

- ◆ 能够使用高级程序设计语言进行简单的编程，如编写简单的程序、处理数据等。
- ◆ 能够了解算法的基本概念，并掌握一些常用的算法。
- ◆ 掌握并能够实现基本的查找和排序算法，理解各种算法的优点和局限性，能够在实际问题中做出合理的算法选择。
- ◆ 学习到的数据结构知识可以扩展到更高级的算法进行学习和探索。
- ◆ 初步具备数据分析、数据管理的工作技能。
- ◆ 具备初步的软件逻辑表达和需求表达能力，具备初步的软件测试技能，具备初步的

软件质量工程与软件项目管理意识。

【素质目标】

- ◆ 培养严谨的逻辑思维和批判性思维，具备分析和解决问题的能力。
- ◆ 培养精确严谨的学习态度，数据结构和算法要求步骤清晰、逻辑严谨，有利于培养严谨的工作作风和科学精神。
- ◆ 提升自学能力，数据结构和算法知识体系庞大且不断刷新，需要学生具备持续学习和快速学习新知识的能力。
- ◆ 让学生体会中国科技进步的动态增长，感受到成长中的中国力量。
- ◆ 理解基本的软件工程职业道德和规范。

## 5.1 程序设计语言的演化

程序设计语言是人与计算机交互的工具，如果想要计算机完成指定的工作，就需要使用程序设计语言编写程序让计算机去执行。随着计算机科学技术的发展，程序设计语言经历了机器语言、汇编语言和高级程序设计语言几个阶段。

### 5.1.1 机器语言

在计算机发展的早期，唯一的程序设计语言就是机器语言。每台计算机都有自己的机器语言，机器语言是由“0”和“1”的字符串组成，是一种“低级语言”，也是计算机唯一能识别的语言。虽然使用机器语言编写的程序真实地表示了数据是如何被计算机操纵的，但它存在很明显的缺点。首先，它依赖于计算机，不同类型的计算机具有不同的机器语言。其次，使用机器语言的指令系统是用二进制代码表示的，编程和理解都非常困难。

### 5.1.2 汇编语言

随着计算机的发展，使用符号或助记符的指令和地址代替二进制代码的语言就是汇编语言，也称为符号语言。例如，ADD 代表加法，MOV 代表数据传递等。使用汇编语言，使用者就可以更容易地读懂并理解程序在干什么，纠错及维护就变得更加方便。

使用汇编语言编写的程序称为汇编语言程序，它需要经过汇编系统翻译成机器语言之后才能执行。但汇编语言比机器语言更容易理解和记忆，且能够直接描述计算机硬件的操作，具有很强的灵活性，因此在实时控制、实时监测等领域仍然使用汇编语言。同时，针对计算机特定硬件而编制的汇编语言程序，能准确发挥计算机硬件的功能和特长，程序精炼而质量高，所以至今仍是一种常用而强有力的软件开发工具。

### 5.1.3 高级语言

尽管汇编语言大大提高了编程效率，但仍需要程序员花费大部分精力在硬件上。同时，使用符号语言编程也很枯燥，因为每条机器指令都得单独编码。为了提高编程效率，人们设计出高级程序设计语言。

高级语言有别于机器语言和汇编语言，它独立于计算机的类型，且表达形式更接近于被描述的问题，从而更容易被人掌握、更便于程序的编写。程序员只要使用简单的英文单词、

熟悉的数学表达式以及规定的语句格式，就可以方便地编写程序，而不必考虑计算机操作的具体细节。

高级语言的设计目的就是使程序员摆脱汇编语言烦琐的细节，但其与汇编语言有一个共同点：编写的程序需要转换为机器语言才能被计算机执行，这个转换过程称为解释或编译。

## 5.2 构建程序

程序的开发过程可分成 4 个主要步骤：编辑→编译→链接→执行，如图 5.1 所示。

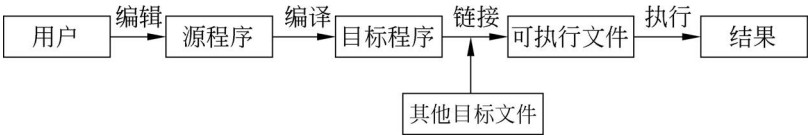



图 5.1　程序开发步骤

### 5.2.1 编辑源程序

编辑源程序是指用户按照一定的程序设计语言规范书写程序代码，并将代码保存到计算机中的过程。源程序可以是以书籍或者磁带或者其他载体的形式出现的，但最为常用的格式是文本文件，程序员可以使用任意一种文本编辑器编辑源程序代码。不过现在的高级程序设计语言开发平台都将程序的编辑、编译、链接、执行等功能进行了集成，极大地方便了用户的使用。

### 5.2.2 编译程序

编译是指把用高级程序设计语言书写的源程序，翻译成计算机能识别的机器语言，源程序经过编译后成为目标程序。编译程序时一般先分析源程序，检查源程序词法、语法和语义的正确性，并将它分解为若干基本成分；然后再根据这些基本成分建立相等价的目标程序部分；最后将目标程序部分综合为目标程序。

### 5.2.3 链接程序

由编译生成的目标程序通常不能立即就能执行，例如，某个源文件中的函数可能引用了另一个源文件中定义的某个符号(如变量或函数等)，或者在程序中可能调用了某个库文件中的函数等。上述这些问题都需使用链接程序来解决，链接程序就是将所有与程序有关的目标文件彼此相连，形成一个能为操作系统执行的统一整体。目标文件经过链接后得到可执行文件，依据目标文件与库函数的链接方式，可将链接分为两大类。

**1. 静态链接**

在静态链接中，库函数的代码将从静态链接库中复制到可执行程序中，这样程序在执行时这些代码将被装入该进程的虚拟地址空间中。静态链接库本质上是一个目标文件的集合，其中的每个文件都含有库中的一个或者一组相关函数的代码。

**2. 动态链接**

在动态链接中，函数的代码放在动态链接库或共享对象的某个目标文件中，此时链接程

序只是在最终的可执行程序中记录共享对象的名字以及少量的登记信息，在执行可执行文件时，动态链接库的全部内容将被映射到运行进程的虚地址空间，并根据可执行程序中记录的信息找到相应的函数代码。

### 5.2.4 程序的执行

源程序经过编辑、编译和链接后生成可执行程序，运行可执行程序就可获得编程处理的结果。与编译、链接不同，运行可执行程序可以脱离语言编译环境。

## 5.3 编程模式

编程模式是使用计算机程序设计语言编写程序来解决具体问题的方式，通常有 4 种编程模式：过程式、面向对象式、函数式和说明式。每种编程模式对应的高级程序设计语言如图 5.2 所示。

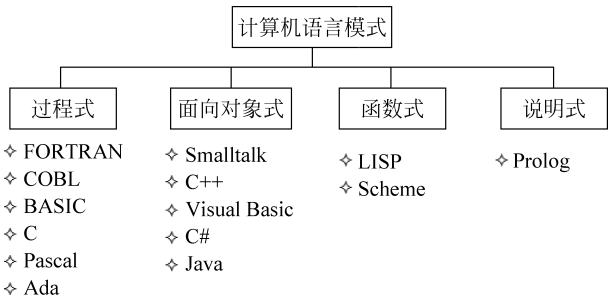



图 5.2 编程模式及语言

### 5.3.1 过程式模式

在过程式模式中存在被动对象及被动对象的活动主体两个概念。其中，被动对象本身不能开始动作，但能通过活动主体接收动作。

在过程式模式下，数据项属于被动对象，存储在计算机内存中；程序属于被动对象的活动主体，对被动对象进行操纵。为了操纵被动对象，活动主体发布动作，称为过程。例如，进行文件打印，先要将文件（被动对象）存储在内存中，然后程序（活动主体）调用 Print 过程完成打印。在过程式模式中，被动对象（文件）和过程（Print）是完全独立的实体。

### 5.3.2 面向对象式模式

过程式模式处理的是被动对象，而面向对象式模式处理的是活动对象，如车、自动门、洗衣机等。通常在活动对象上能执行的所有动作都包含在活动对象自身之中，只需接收合适的外部刺激就可执行其中相应的动作。在面向对象模式中，将文件能执行的所有操作过程（打印、复制、删除等）打包在一起，称为“方法”，此时程序只需向活动对象发出相应请求（打印、复制、删除等），文件就会被打印、复制或删除。

### 5.3.3 函数式模式

在函数式模式中，程序被当作一个函数。例如，求和可认为是具有 $n$ 个输入 1 个输出的

函数。函数式语言具有以下主要功能。

(1) 定义一系列可供任何程序员调用的原始函数。

(2) 允许程序员将若干原始函数进行组合创建新的函数。

例如,在图 5.3 中,定义函数 first,其功能是从一个数据列表中取出第一个元素;再定义函数 rest,其功能是从一个数据列表中取出除第一个元素以外的所有元素。通过函数 rest 和 first 的组合,可以创建一个新函数 third,来完成对数据列表中第三个元素的抽取。

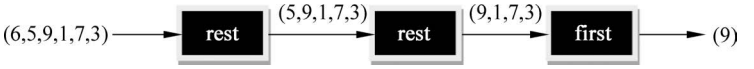



图 5.3 创建新函数 third

### 5.3.4 说明式模式

说明式模式是在规范的逻辑基础上发展而来,它依据逻辑推理原则响应查询,后来发展成为一阶谓词演算。逻辑推理以推导为基础,根据已知正确的事实,运用逻辑推理的可靠准则推导出新的事实。例如,逻辑学有以下著名的推导原则:If (A is B) and (B is C),then (A is C),将此原则应用于事实 1 和事实 2,可以推导出事实 3。

事实 1:Socrates is a human→A is B

事实 2:A human is mortal→B is C

事实 3:Socrates is mortal→A is C

说明式语言也有自身的缺陷,程序员需要掌握大量有关领域的知识及已知的论据,同时还要精通如何逻辑上严谨地定义准则,只有这样才能推导并产生新的论据,这也是说明式程序目前主要局限于人工智能等领域的原因。

## 5.4 高级程序语言概述

### 5.4.1 变量、运算符和表达式

**1. 变量**

变量本质上是存储单元的名称,用于引用计算机内存地址。但对于程序员来说,可将变量理解为一种使用方便的占位符,只需通过变量名就可以使用变量或者查看、修改变量的值,而无须了解变量在计算机内存中的具体地址。

1) 变量声明

大部分程序语言要求变量在使用前必须先声明,声明的主要作用就是告诉计算机该变量及变量类型将在程序中使用,同时要求计算机预留出相应的存储区域。在 C、C++ 和 Java 中,声明字符型、整数型和实数型变量的格式如下。

```
char a;                    //声明字符类型变量 a
int b;                     //声明整数类型变量 b
double c;                  //声明实数类型变量 c
```

2) 变量初始化

变量初始化是为已声明过的变量赋初值,并为后续使用该变量做准备,未初始化的变量

其内存中的值是随机的。变量初始化可以在变量声明后进行，也可以在声明变量的同时进行，如下所示。

```
char a = 'z';
int b = 125;
double c = 123.45;
```

**2. 运算符和表达式**

程序设计中用来表示各种运算的符号称为运算符；参与运算的数据称为运算对象，或称为运算量、操作数；使用运算符将运算对象连接起来的式子称为运算表达式。运算表达式的种类有很多，常见的有算术表达式、关系表达式、逻辑表达式等。例如，2＊4＋2 为运算表达式。

1）算术运算符和算术表达式

算术运算符包括基本算术运算符（＋，－，＊，/，％）和自增运算符（＋＋）、自减运算符（－－）两大类。使用算术运算符和括号将运算对象连接起来的符合某种语言语法规则的式子称为算术表达式。常用的算术运算符如表 5.1 所示。

**表 5.1　算术运算符**

| 运　算　符 | 含　　义 | 示例(x=1) | 运 算 结 果 |
|---|---|---|---|
| ＋ | 加 | 7＋2 | 9 |
| － | 减 | 7－2 | 5 |
| ＊ | 乘 | 7＊2 | 14 |
| / | 除(商) | 7/2；7.0/2/ | 3；3.5 |
| ％ | 求余 | 7％2 | 1 |
| ＋＋ | 自增 | y=＋＋x；y=x＋＋ | y=2,x=2；y=1,x=2 |
| －－ | 自减 | y=－－x；y=x－－ | y=0,x=0；y=1,x=0 |

2）关系运算符和关系表达式

关系运算符主要用于比较两个数据量的大小关系，运算结果是逻辑值（true 或 false）。常用的关系运算符有 6 种，如表 5.2 所示。

**表 5.2　关系运算符**

| 运　算　符 | 定　　义 | 示例(a=2,b=3) | 运 算 结 果 |
|---|---|---|---|
| ＜ | 小于 | a＜4 | 1(真) |
| ＜＝ | 小于或等于 | a＜＝4 | 1(真) |
| ＞ | 大于 | a＞b＋2 | 0(假) |
| ＞＝ | 大于或等于 | a＞＝b＋2 | 0(假) |
| ＝＝ | 等于 | a＝＝5 | 0(假) |
| !＝ | 不等于 | a!＝b | 1(真) |

3）逻辑运算符和逻辑表达式

逻辑运算符主要用于逻辑值（true 或 false）的运算，逻辑运算的结果仍然是逻辑值。逻辑运算符有三种，分别是逻辑与（&&）、逻辑或（||）、逻辑非（!），如表 5.3 所示。

**表 5.3　逻辑运算符**

| 运　算　符 | 定　　义 | 示例(a=2,b=0) | 运 算 结 果 |
|---|---|---|---|
| ! | 非 | !a | 0(假) |
| && | 与 | a&&b | 0(假) |
| \|\| | 或 | a\|\|b | 1(真) |

说明：C 语言中规定，当运算对象为 0 时，即判定其为假；当运算对象为非 0 的任何值(包括负值)时，即判定其为真。

4) 运算符的优先级和结合方向

当一个表达式中存在多个运算符时，运算符被处理的先后次序称为运算符的优先级。C 语言中常见运算符的优先级和结合方向如表 5.4 所示。

**表 5.4　C 语言中常见运算符的优先级和结合方向**

| 优先级 | 运算符 | 名称或含义 | 使用形式 | 结合方向 | 说明 |
|---|---|---|---|---|---|
| 1 | 后置++ | 后置自增运算符 | 变量名++ | 左到右 | |
| | 后置-- | 后置自减运算符 | 变量名-- | | |
| | [ ] | 数组下标 | 数组名[整型表达式] | | |
| | ( ) | 圆括号 | (表达式)/函数名(形参表) | | |
| 2 | - | 负号运算符 | -表达式 | 右到左 | 单目运算符 |
| | (类型) | 强制类型转换 | (数据类型)表达式 | | |
| | 前置++ | 前置自增运算符 | ++变量名 | | 单目运算符 |
| | 前置-- | 前置自减运算符 | --变量名 | | 单目运算符 |
| | * | 取值运算符 | *指针表达式 | | 单目运算符 |
| | & | 取地址运算符 | &左值表达式 | | 单目运算符 |
| | ! | 逻辑非运算符 | !表达式 | | 单目运算符 |
| 3 | / | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| | * | 乘 | 表达式*表达式 | | 双目运算符 |
| | % | 余数(取模) | 整型表达式%整型表达式 | | 双目运算符 |
| 4 | + | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| | - | 减 | 表达式-表达式 | | 双目运算符 |
| 5 | > | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| | >= | 大于或等于 | 表达式>=表达式 | | 双目运算符 |
| | < | 小于 | 表达式<表达式 | | 双目运算符 |
| | <= | 小于或等于 | 表达式<=表达式 | | 双目运算符 |
| 6 | == | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| | != | 不等于 | 表达式!= 表达式 | | 双目运算符 |
| 7 | && | 逻辑与 | 表达式 && 表达式 | 左到右 | 双目运算符 |
| 8 | \|\| | 逻辑或 | 表达式\|\|表达式 | 左到右 | 双目运算符 |
| 9 | ?: | 条件运算符 | 表达式 1?表达式 2: 表达式 3 | 右到左 | 三目运算符 |
| 10 | = | 赋值运算符 | 变量=表达式 | 右到左 | |
| | /= | 除后赋值 | 变量/=表达式 | | |
| | *= | 乘后赋值 | 变量*=表达式 | | |
| | %= | 取模后赋值 | 变量%=表达式 | | |
| | += | 加后赋值 | 变量+=表达式 | | |
| | -= | 减后赋值 | 变量-=表达式 | | |
| 11 | , | 逗号运算符 | 表达式,表达式,… | 左到右 | 从左向右 |

## 5.4.2 数据类型

程序设计语言规定，程序中使用的每个数据必须属于某种类型。高级程序设计语言提供了丰富的数据类型，归纳起来可分为两类：基本数据类型和构造数据类型。C 语言中的数据类型如图 5.4 所示。


图 5.4　C 语言的数据类型

**1. 基本数据类型**

基本数据类型也称为原子类型、标量类型或内建类型，是不能分解为更小数据类型的数据类型。常用的基本数据类型主要包括以下几种。

整数类型：不带小数部分的数字。

实数类型：带小数部分的数字。

字符类型：Unicode 字符集中的符号。

布尔类型：只有两种取值(真或假)的数据类型。

**2. 构造数据类型**

构造数据类型也称为组合数据类型，是由一组元素组合而成，其中每个元素都是基本数据类型或复合数据类型。大部分高级语言都定义了以下几种构造数据类型：数组、结构体、共用体等。

## 5.4.3 赋值语句

赋值是指将一个具体的值赋给已经声明过的变量，大多数高级程序设计语言(如 C、C++)使用“=”来赋值，如以下示例所示。

```
int a,b,c;                    //声明三个整型变量
a = b = c = 3;                //通过赋值符号"="对变量 a,b,c 同时赋值 3
```

### 5.4.4 输入/输出

输入/输出数据是绝大多数程序所必需的，高级程序设计语言通常都具有预定义好的输入/输出函数，用户只需直接调用即可完成输入/输出。

**1. 输入**

输入就是将外部数据送入程序，供程序使用或处理。C 语言中预定义的输入函数有 scanf 和 getchar 等，scanf 用于带格式输入，getchar 用于输入单个字符，并把输入的数据存储到一个变量中。例如：

```
scanf("%d",&a);              //输入一个整数并赋给变量 a,其中,%d 用于控制输入变量的类型
b=getchar();                 //输入一个字符并赋给变量 b
```

**2. 输出**

输出就是将程序处理后的数据送出程序，供其他程序使用或显示、打印等。C 语言中预定义的输出函数有 printf 和 putchar 等，printf 用于带格式输出，putchar 用于输出单个字符。例如：

```
printf("The values is: %d",b);
putchar("z");
```

### 5.4.5 控制结构

在程序运行过程中，多数时候是按先后顺序执行代码，但有时仅选择执行部分代码或者反复执行某一部分代码，这时可以通过不同的程序控制结构来满足要求。基本的程序控制结构有三种：顺序结构，选择结构和循环结构。

**1. 顺序结构**

顺序结构是最基本的结构方式，程序执行时，按照语句的顺序，从上而下一条一条地执行。顺序结构的流程图如图 5.5 所示，先执行 A 操作，再执行 B 操作，两者是顺序执行关系。

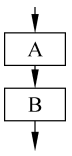



图 5.5 顺序结构

**2. 选择结构**

选择结构也称分支结构，是根据“条件”的真、假来选择执行哪一分支中的语句。选择结构通常可分为三种形式：单分支、双分支和多分支，如图 5.6～图 5.8 所示。

1) 单分支(if)

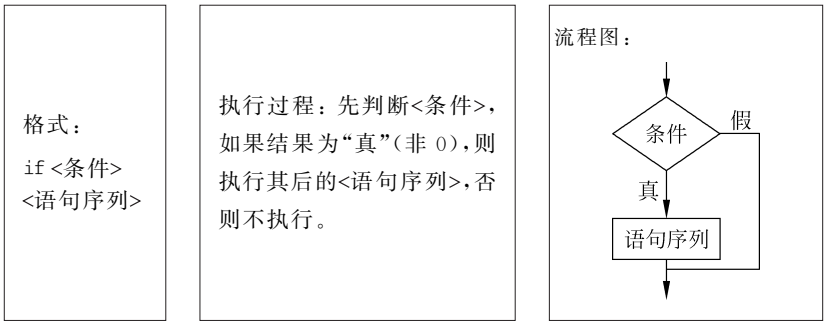



图 5.6 单分支结构

2）双分支(if…else)

格式：

if <条件>
<语句序列 1 >
else
<语句序列 2 >

执行过程：先判断<条件>，如果结果为“真”(非 0)，则执行<语句序列 1 >；如果结果为<假>(0)，则执行<语句序列 2 >。

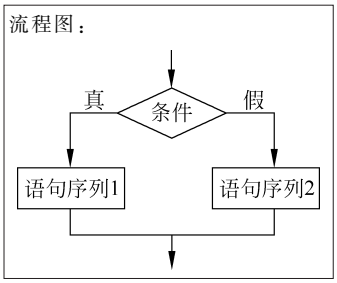



图 5.7　双分支结构

3）多分支(if…else…if)

格式：

if <条件 1 >
<语句序列 1 >
else if <条件 2 >
<语句序列 2 >
else if <条件 3 >
<语句序列 3 >
…
else if <条件 $n$ >
<语句序列 $n$ >
else
<语句序列 $n+1$ >

执行过程：
依次判断<条件>，当出现某个值为真时，则执行其对应的<语句序列>，然后跳到整个 if 语句之外继续执行程序；如果所有的<条件>都为假，则执行<语句序列 $n+1$ >，然后继续执行后续程序。

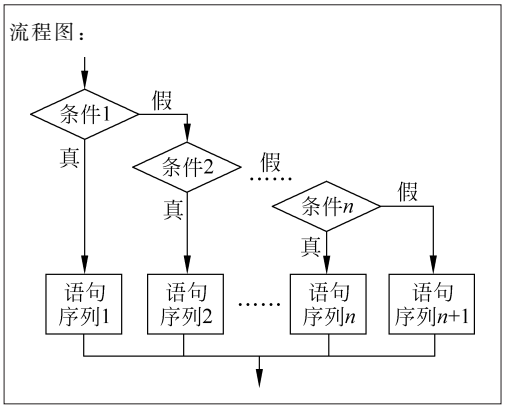



图 5.8　多分支结构

**3. 循环结构**

循环结构可以看成一个条件判断语句和一个向回转向语句的组合。当条件成立时，执行循环结构内的代码；当条件不成立时，跳出循环，执行循环结构后的代码。循环结构主要有以下两种。

(1) 当型循环：先判断条件，当条件为真(非 0)，执行循环体中的语句；当条件为假(0)，跳出循环体，执行循环体后面的语句。因此，当型循环执行循环体的次数为≥0。执行过程如图 5.9 所示。

(2) 直到型循环：先执行循环体中的语句，再判断循环条件是否为真，如果为真，则继续循环；如果为假，则终止循环。因此，直到型循环执行循环体的次数为≥1。执行过程如图 5.10 所示。

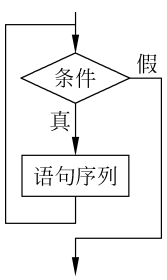



图 5.9　当型循环

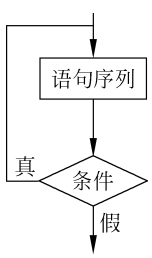



图 5.10　直到型循环

C 语言中常用的循环语句有 while、do…while 和 for。从工作原理上判断，while、for 属于当型循环，do…while 属于直到型循环。



### 5.4.6 注释语句

在程序设计过程中，附加一些说明性信息有助于用户更好地阅读与理解程序。因此，程序设计语言提供了注释语句，即在程序中插入解释性语句，在编译程序时，编译系统对注释语句不做任何处理。对计算机而言，注释的存在与否不会影响程序的执行；但对用户而言，注释是程序的重要组成部分，没有注释将极大地影响程序的阅读和理解。

程序中的注释语句一般有两种形式：一是使用两个特殊的符号在注释开始与注释结尾位置括起来；二是标示出注释的起始位置，标记符号右边的字符全部都属于注释。在 C、C++和 Java 中，使用记号/ * … * /把注释括起来，或者用记号//开始一个注释直至行末。例如：

```
/* This is my first program. */
// This is my first program.
```

## 5.5 常用高级语言

高级语言是一类面向问题或面向对象的语言，它不面向机器也不依赖于具体的计算机，但在不同的平台上会被编译成不同的机器语言。由于高级语言采用自然词汇，并使用与自然语言语法相近的语法体系，所以它的程序设计方法比较接近于人类的思维习惯，编写出的程序更容易阅读和理解。常见的高级程序设计语言可分为两大类：面向过程语言和面向对象语言。

### 5.5.1 C 语言

C 语言是一种面向过程的计算机程序设计语言，它是目前众多计算机语言中公认的优秀的结构程序设计语言之一，由美国贝尔实验室的 Dennis M. Ritchie 于 1972 年推出，1978 年后 C 语言先后被移植到大、中、小及微型计算机上。C 语言可以作为工作系统设计语言，编写系统应用程序，也可以作为应用程序设计语言，编写不依赖计算机硬件的应用程序。C 语言的应用范围广泛，具有丰富的数据类型和较强的数据处理能力，广泛应用于科学研究、日常生活等领域。

C 语言的优点是：简洁紧凑，灵活方便，运算符丰富，数据类型丰富，表达方式灵活，允许直接访问物理地址，对硬件进行操作，生成目标代码质量高，程序执行效率高，可移植性好，表达力强。

### 5.5.2 C++语言

C++语言是一种优秀的面向对象程序设计的语言，它是在 C 语言的基础上发展而来，但比 C 语言更容易学习和掌握。C++语言是由 AT&T 贝尔实验室的 Bjarne Stroustrup 设计和实现，兼备 Simula 语言和 C 语言的特性，1983 年支持面向对象程序设计，1985 年第一次投入商业市场，1987—1989 年支持范型程序设计。

C++语言的主要特征如下。

(1) C++是C语言的超集,比C语言更安全。它既保持了C语言的简洁、高效和接近汇编语言等特点,又克服了C语言的缺点,其编译系统能检查更多的语法错误。同时,绝大多数C语言程序可以不经修改直接在C++环境中运行,用C语言编写的众多库函数也可以用于C++程序中。

(2) C++既支持面向过程的程序设计,又支持面向对象的程序设计。

(3) C++程序在可重用性、可扩充性、可维护性和可靠性等方面都较C语言得到了提高,使其更适合开发大中型的系统软件和应用程序。

(4) C++设计成静态类型,和C同样高效且可移植性强,能给程序设计者更多的选择。

(5) C++避免平台限定或没有普遍用途的特性,不带来额外开销,不需要复杂的程序设计环境。

### 5.5.3 Java语言

Java语言是一种简单的、跨平台的、面向对象的、分布式的、可解释的、健壮的、安全的、结构的、中立的、可移植的、性能优异的、多线程的、动态的语言。Java是由Sun Microsystems公司于1995年5月推出的面向对象程序设计语言。

Java语言的主要特征如下。

(1) Java语言是简单的。Java语言的语法与C语言和C++语言很接近,使得大多数程序员很容易学习和使用Java。另外,Java去掉了C++中很少使用的、难理解的那些特性。

(2) Java语言是面向对象的。Java提供类、接口和继承等原语,只支持类之间的单继承,但支持接口之间的多继承,并支持类和接口之间的实现机制。

(3) Java语言是分布式的。Java建立在扩展TCP/IP网络平台上。库函数提供了用HTTP和FTP传送和接收信息的方法,这使得程序员使用网络上的文件和使用本机文件一样容易。

(4) Java语言是健壮的。Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。

(5) Java语言是安全的。Java丢弃了C++的指针对存储器地址的直接操作,程序运行时,内存由操作系统分配,因此可以避免病毒通过指针侵入系统。Java对程序提供了安全管理器,防止程序的非法访问。

(6) Java语言是体系结构中立的。Java程序在Java平台上被编译为体系结构中立的字节码格式,然后可以在实现这个Java平台的任何系统中运行。这种途径适用于异构的网络环境和软件的分发。

(7) Java语言是可移植的。这种可移植性来源于体系结构中立性,另外,Java还严格规定了各个基本数据类型的长度。Java系统本身也是具有很强的可移植性。Java编译器是用Java实现的,Java的运行环境是用ANSI C实现的。

(8) Java语言是多线程的。Java语言支持多个线程的同时执行,并提供多线程之间的同步机制。

### 5.5.4 Python 语言

Python 是一种高级的、动态类型的编程语言，最初设计用于编写自动化脚本。Python 的简洁性和易读性使得它成为一种非常流行的编程语言，尤其适合初学者。Python 具有强大的标准库和丰富的第三方库，这使得 Python 在各种领域都有广泛的应用，包括 Web 开发、数据科学、人工智能、机器学习、网络爬虫、系统自动化、游戏开发等。

**2024 年 1 月 TIOBE 编程排行榜**

TIOBE 编程社区指数是编程语言流行程度的指标。该索引每月更新一次。评级基于全球熟练工程师的数量、课程和第三方供应商。流行的搜索引擎，如谷歌、必应、雅虎，使用维基百科、亚马逊、YouTube 和百度来计算评分。索引可用于检查编程技能是否仍然是最新的，或者在开始构建新的软件系统时，就应该采用什么编程语言做出战略决策。TIOBE 官方网站公布的 2024 年编程排行榜数据如图 5.11 所示。

| Jan 2024 | Jan 2023 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Python | 13.97% | -2.39% |
| 2 | 2 | | C | 11.44% | -4.81% |
| 3 | 3 | | C++ | 9.96% | -2.95% |
| 4 | 4 | | Java | 7.87% | -4.34% |
| 5 | 5 | | C# | 7.16% | +1.43% |
| 6 | 7 | ↑ | JavaScript | 2.77% | -0.11% |
| 7 | 10 | ↑ | PHP | 1.79% | +0.40% |
| 8 | 6 | ↓ | Visual Basic | 1.60% | -3.04% |
| 9 | 8 | ↓ | SQL | 1.46% | -1.04% |
| 10 | 20 | ⇈ | Scratch | 1.44% | +0.86% |
| 11 | 12 | ↑ | Go | 1.38% | +0.23% |
| 12 | 27 | ⇈ | FOTYTAN | 1.09% | +0.64% |
| 13 | 17 | ⇈ | Delphi/Object Pascal | 1.09% | +0.36% |
| 14 | 15 | ↑ | MATLAB | 0.97% | +0.06% |
| 15 | 9 | ⇊ | Assembly language | 0.92% | -0.68% |
| 16 | 11 | ⇊ | Swift | 0.89% | -0.31% |
| 17 | 25 | ⇈ | Kotlin | 0.85% | +0.37% |
| 18 | 16 | ↓ | Ruby | 0.80% | +0.01% |
| 19 | 18 | ↓ | Rust | 0.79% | +0.18% |
| 20 | 31 | ⇈ | COBOL | 0.78% | +0.45% |

图 5.11　TIOBE 编程排行榜

通过查阅 TIOBE 编程语言排行榜，可以了解全球范围内最流行的编程语言以及未来的趋势，有助于选择适合自己的编程语言和学习方向，从而有针对性地学习和实践，提高自己的就业竞争力，同时培养我们的全球视野和跨文化沟通能力。

## 5.6 算法概述*

计算机系统中的软件都是由某种算法实现的程序模块组成，算法的好坏直接影响软件性能的优劣。因此，算法设计是软件设计的核心问题，设计时必须要解决以下问题：使用何种方法来设计算法，算法需要哪些资源，算法时间复杂度与空间复杂度如何，等等。

Java 语言的主要特征如下。

（1）简洁易读的语法：Python 采用简洁的语法和清晰的代码结构，使得代码易于阅读和理解。

（2）动态类型：Python 是一种动态类型语言，变量的类型在运行时可以自动推断，无须显式声明。

（3）面向对象：Python 支持面向对象编程，可以使用类和对象来组织和管理代码。面向对象的编程范式使得代码更加模块化、可重用和易于维护。

（4）强大的标准库：Python 拥有丰富的标准库，包含各种功能模块，如文件操作、网络通信、数据库连接等。

（5）跨平台：Python 可以在多个操作系统上运行，包括 Windows、Linux、Mac 等。

（6）可扩展性：如果需要一段关键代码运行得更快或者希望某些算法不公开，可以部分程序用 C 或 C++编写，然后在 Python 程序中使用它们。

（7）解释性：用 Python 语言编写的程序不需要编译成二进制代码，而是可以直接从源代码执行程序，在计算机内部，通过 Python 解释器将源代码转换成字节码的中间形式，并将其翻译成计算机使用的机器语言后执行。

（8）可嵌入性：可以把 Python 嵌入 C/C++程序，从而向程序用户提供脚本功能。

（9）可读性强：使用 Python 编写代码时强制缩进可以使代码具有非常好的可读性。

### 5.6.1 算法的概念

算法（Algorithm）是对特定问题求解步骤准确而完整的描述，它的表现形式是计算机指令的有序系列，执行这些指令就可解决特定问题。不同的算法所需的时间、空间不同，其效率也不同；通常使用时间复杂度与空间复杂度来度量一个算法的优劣。一个好的算法应当具有以下 5 个重要特性。

（1）有限性：算法在执行有限步之后必须终止，且每一步都应在有限的时间内完成。这里的“有限”是指实际应用中可以接受的、合理的时间和步骤。

（2）确定性：算法的每一步必须要有确切的含义，不能存在二义性。即在任何条件下，算法只有唯一的一条执行路径，相同的输入只能产生相同的输出。

（3）可行性：算法的每一步都是可执行的，可以通过有限次操作来完成其功能；只要其中有一个操作不可执行，整个算法就不可执行。

（4）输入：一个算法具有 0 个或多个输入。有些算法的输入量是在算法的执行过程中输入；有些算法表面上没有输入，但实际输入量已嵌入算法中。

（5）输出：一个算法具有 1 个或多个输出，输出与输入有着特定的对应关系，是算法对输入进行加工处理后的结果，没有输出的算法是没有意义的。

【例 5.1】 给定两个整数 $m$ 和 $n(m>n)$，求它们的最大公约数。

算法 1：穷举法

(1) $r=n$。

(2) 以 $m$ 除以 $r$，$n$ 除以 $r$，并令所得余数为 $r_1$，$r_2$。

(3) 若 $r_1=r_2=0$，则算法结束，$r$ 为 $m$ 和 $n$ 的最大公约数；否则 $r=r-1$，继续步骤(2)。

(4) 输出结果 $r$。

算法 2：辗转相除法

(1) 以 $m$ 除以 $n$，并令所得余数为 $r(r<n)$。

(2) 若 $r=0$，算法结束，输出结果 $n$；否则继续步骤(3)。

(3) 将 $m$ 置换为 $n$，将 $n$ 置换为 $r$，返回步骤(1)继续进行。

从上面的例子可以看出，同样的问题，可以有多种完全不同的解决方法，每种方法都是一个有穷规则的集合，其中的规则确定了解决最大公约数问题的运算系列。上述两个算法都是执行有穷步之后结束，每个步骤都有确切的含义，且都有输入和输出，并最终得到正确的结果。

## 5.6.2 算法的表示

为了让算法清晰易懂，通常需要一种描述方法来对算法进行表示。算法的描述方法有很多，较为常用的有自然语言法、流程图法、N-S 流程图法、伪代码法等。

**1. 自然语言**

自然语言就是采用人们日常使用的语言，来描述解决问题的方法和步骤；这种描述方法通俗易懂，即使是不熟悉计算机语言的用户也很容易理解程序。

【例 5.2】 使用自然语言描述从 1 开始的连续 $n$ 个自然数求和算法，即 $sum=1+2+\cdots+n$。

(1) 确定 $n$ 的值。

(2) 设等号右边的算式项中的初始值 $i$ 为 1。

(3) 设 sum 的初始值为 0。

(4) 如果 $i\leqslant n$ 时，执行(5)，否则转出执行(8)。

(5) 计算 sum 加上 $i$ 的值后，重新赋值给 sum。

(6) 计算 $i$ 加 1，然后将值重新赋给 $i$。

(7) 转去执行(4)。

(8) 输出 sum 的值，算法结束。

使用自然语言描述算法虽然比较容易掌握，但也存在较大缺陷。例如，自然语言在语法和语义上往往具有多义性，并且比较烦琐，对程序流向等描述不明了、不直观；同时在计算机上难以将自然语言翻译成计算机程序设计语言。

**2. 流程图**

流程图是以特定的图形符号加上说明来表示算法，通常是用一些图框来表示各种操作。用流程图表示算法，直观形象，易于理解，结构清晰，同时不依赖任何具体的计算机和计算机程序设计语言，有利于不同环境的程序设计。

美国国家标准化协会规定了一些常用的流程图符号，如表 5.5 所示。

表 5.5　常用的流程图符号

| 名　称 | 符　号 | 意　义 | 示　例 |
|---|---|---|---|
| 起止符 | | 表示作业的开始或结束 | 开始 |
| 处理符 | | 表示执行或处理某些工作 | sum=sum+$i$ |
| 决策判断符 | | 表示对某一个条件做出判断 | $i$>10 |
| 流程线 | | 表示流程进行的方向 | |
| 输入/输出符 | | 表示数据的输入或结果的输出 | 请输入$i$的值 |
| 连接符 | | 用于流程转接到另一页；避免流程线交叉或过长 | A　A |

**【例 5.3】** 用流程图法描述 sum＝1＋2＋…＋$n$，如图 5.12 所示。

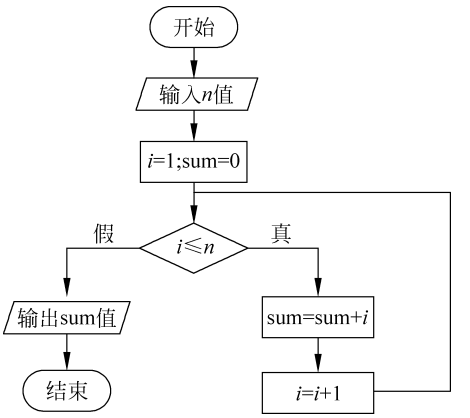



图 5.12　求解 sum＝1＋2＋…＋$n$ 的流程图

### 3. N-S 流程图

N-S 流程图简称 N-S 图，也称为盒图或 CHAPIN 图。N-S 图是在 1973 年由美国学者 I. Nassi 和 B. Shneiderman 提出，并以两人姓氏的首字母来命名。N-S 图是在流程图的基础上完全去掉流程线，并将全部算法写在一个矩形框内，且框内还可以包含其他框的表示形式。N-S 图包括顺序、选择和循环三种基本结构，表示形式分别如图 5.13～图 5.15 所示。


图 5.13　顺序结构

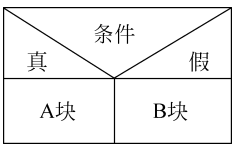



图 5.14　选择结构

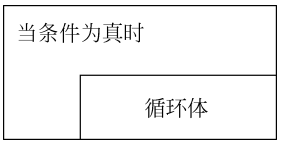



图 5.15　循环结构

【例 5.4】 用 N-S 流程图法描述 sum＝1＋2＋…＋$n$，如图 5.16 所示。

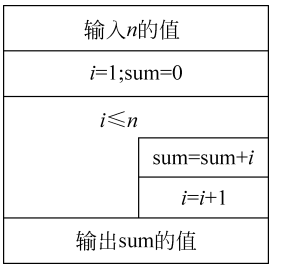



图 5.16 求解 sum＝1＋2＋…＋$n$ 的 N-S 流程图

N-S 流程图形象直观，简单易学，具有良好的可见度，设计意图易于理解，为编程、选择测试用例和维护都带来了便利，是软件设计时常用的算法描述方法之一。

**4. 伪代码**

伪代码是介于自然语言和计算机语言之间的文字和符号，它与一些高级程序设计语言（如 Visual Basic 和 Visual C++）类似，但没有高级程序设计语言所要遵循的严格规则。伪代码通常采用自然语言、数学公式和符号来描述算法的操作步骤，同时采用计算机高级语言的控制结构来描述算法步骤的执行顺序。在程序开发期间，伪代码经常用于“规划”一个程序，然后再转换成某种高级语言程序。

【例 5.5】 用伪代码法来描述 sum＝1＋2＋…＋$n$。

（1）算法开始；

```
输入 n 的值
i←1;
sum←0;
do while i <= n;
  { sum←sum + i;
    i←i + 1;
      }
```

（2）输出 sum 的值。

（3）算法结束。

## 5.6.3 常用算法简介

计算机技术所涉及的算法比较多，常用的算法有枚举法、递推法、递归法、贪心算法、分治法、回溯法等，下面对它们做一简单介绍。

**1. 枚举法（穷举法）**

枚举法，或称为穷举法，其基本思路是：对于要解决的问题，列举出它所有可能的情况，逐个判断哪些是符合问题所要求的条件，从而得到问题的解。例如，要破译一个全部由数字组成的 4 位数密码，因其最多只有 10 000 种组合，故只需按 0000～9999 进行逐个尝试就能找到正确的密码。理论上利用枚举法可以破解任何一种密码，问题只在于如何缩短破译时间。

**2. 递推法**

递推法是按照一定的规律来计算序列中的某个项，通常是通过计算前面的一些项来得出序列中指定项的值。递推法的基本思想是把一个复杂的、庞大的计算过程转换为简单过程的多次重复，算法主要利用计算机速度快和不知疲倦的特点进行实现。

**3. 递归法**

程序直接或间接自己调用自己的方法简称为递归，它通常是把一个大型的、复杂的问题层层转换为一个个与原问题相似的、规模较小的问题来进行求解，递归策略只需少量的程序

就可描述出解题过程所需要的多次重复计算，大大减少了程序的代码量。一般来说，递归需要有边界条件、递归前进段和递归返回段，当边界条件不满足时，递归前进，当边界条件满足时，递归返回。

**4. 贪心算法**

贪心算法是一种对某些求最优解问题的更简单、更迅速的设计技术，算法的特点是一步一步地进行，以当前情况为基础，根据某个优化测度做最优选择，而不考虑各种可能的整体情况，省去了为找最优解要穷尽所有可能而必须耗费的大量时间。贪心算法采用自顶向下，以迭代的方式做出相继的贪心选择，每做一次贪心选择就将所求解问题简化为一个规模更小的子问题，通过每一步贪心选择，就可得到问题的一个最优解。贪心算法的每一步都能获得局部最优解，但由此产生的全局解有时不一定是最优解，所以贪婪法不能回溯。

**5. 分治法**

分治法是把一个复杂的问题分解成两个或更多个相同或相似的子问题，再把子问题分解成更小的子问题，直到最后的子问题可以进行简单的直接求解，合并所有子问题的解就可得到原问题的解。

分治法所能解决的问题一般具有以下几个特征。

(1) 该问题的规模缩小到一定的程度就可以容易地解决。

(2) 该问题可以分解为若干规模较小的相同问题，即该问题具有最优子结构性质。

(3) 利用该问题分解出的子问题的解可以合并为该问题的解。

(4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

**6. 回溯法**

回溯法(探索与回溯法)是一种选优搜索法，按选优条件向前搜索，以达到目标。当搜索到某一步时，发现原先的选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态点称为“回溯点”。回溯法的基本思想是，在包含问题所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度搜索解空间树；当搜索到某一结点时，要先判断该结点是否包含问题的解，如果包含就从该结点出发继续搜索下去，否则就逐层向其祖先结点回溯。若用回溯法求问题的所有解，需要回溯到根，且根结点的所有可行的子树都要进行搜索后才能结束；若使用回溯法求任一个解时，则只需搜索到问题的一个解就可以结束。

## 5.7 数据结构

### 5.7.1 数据结构基本概念

数据结构是计算机科学的核心概念之一，它是数据值在计算机中整齐地储存并组合在一起的一种方式，使得能够有效地访问和操作这些数据。在最基本的层面上，数据结构作为数据的容器，将数据组织成特定的格式，能高效地执行各种操作，包括搜索、插入、删除、排序等。它为程序员提供了强大的工具，使其能够设计出高效、灵活且易于理解的算法。本节将探讨数据结构的基本概念，为初学者提供对这一领域的较为全面认识。

**1. 数据与数据结构**

数据(Data)是由数字、字符等符号构成的描述事物属性的一种表现形式。对于计算机

来说，数据就是可以被计算机识别、处理和传输的符号集合。例如，一个图书馆的书籍数据库中包含的每一本书的信息（如书名、作者、出版年份、出版社、ISBN 等）都是数据。

数据元素（Data Element）又被称为记录项，它是数据的基本单位，也是数据库中存储的最小独立逻辑单元。用户需要获取的各类信息和知识，通常都是通过操作各类数据元素得到的。例如，一个图书馆的书籍数据库中的一本书（包含书名、作者、出版年份、出版社、ISBN 等信息）就是一个数据元素。书的每一项具体信息，如书名、作者，虽然也是数据，但是它们被合在一起才构成了一个完整的数据元素，这个数据元素提供了一本书的全部信息。每一项信息单独看只是数据项，是数据元素中的一部分。此时，每一本书就是一个数据元素，每一个数据元素是独立的，构成了这个图书馆的书籍数据库的主要内容。

数据项（Data Item）是数据的基本单位，也是组成数据元素的最小独立单元。数据项往往表示一种特性或属性，它不可再分割。可以将其视为一种简单的、最基础的信息片断。例如，在上文中提到的图书馆的书籍数据库中，每一本书都是一个数据元素。如图 5.17 所示是一本书的数据元素描述，其中，书名、作者、出版年份、出版社、ISBN 都是数据项。这些数据项都是关于这本书的具体信息，每一项都对应着书的一个具体属性。这些信息组合在一起，组成了一个完整的数据元素，描述了一本书的全部信息。数据项的不同组合可以反映出无数种信息，是构成复杂信息结构的基础。在处理数据时，往往需要对数据项进行处理，例如，查询特定的数据项，修改数据项的值，或者根据数据项的值进行决策等。

| 书名 | 作者 | 出版年份 | 出版社 | ISBN |
|---|---|---|---|---|

图 5.17　一本书的数据元素描述

数据对象（Data Object）是一种具有相同性质的数据元素的集合。例如，图书馆的书籍数据库中可以把所有的书籍按照类型进行分类，如小说、教科书、参考书等，把同一类型的书籍集合起来，形成一个数据对象。这个数据对象就是一个由性质相同的数据元素（同类型的书）所组成的集合。并且，每个数据对象中的每本书都有相同的数据项，如书名、作者、出版年份、出版社、ISBN 等。

数据结构（Data Structure）在计算机科学中，数据结构是将相关联的数据项组织和存储起来以进行高效操作的一种特定方式。数据结构定义了数据项的逻辑关系、存储方式和可能的操作。

**2. 数据结构的三方面**

1）逻辑结构

数据的逻辑结构是指数据对象在逻辑上的组织与构成方式，是从数据的逻辑性质出发的抽象结构。这种结构反映了数据元素之间的逻辑关系，适用于直观性的数据组织分析。通常，逻辑结构可以分为 4 种基本形式：集合、线性结构、树状结构和图结构。

（1）集合。

集合结构是最简单的数据结构，在集合结构中，数据元素之间除了同属于一个集合外，没有其他关系。元素之间的任何关系可以用集合来表示。在实际中，集合结构通常用于描述没有明显内部结构的数据元素集合。例如，可以把一本书中所有的英文单词看作一个集合，这个集合的每个元素就是书中的每一个英文单词，但是单词之间并没有其他明显的先后或上下级关系，如图 5.18 所示。

(2) 线性结构。

线性结构是最常见的数据结构之一,它的特点是数据元素之间存在一对一的线性关系。每个数据元素(除第一个和最后一个外)都有一个前驱和一个后继。线性结构的数据元素可以映射到一个线性序列中,如线性链表、队列、栈和数组等。例如,去超市购物时,通常会制作一份购物清单。在清单中,从上到下列出了要购买的商品,这样的清单就形成了一个线性的关系结构,如图 5.19 所示。

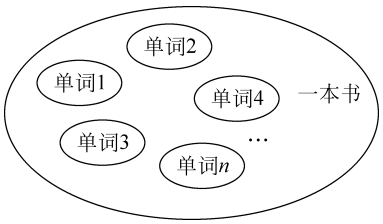



图 5.18 集合结构


图 5.19 线性结构

(3) 树状结构。

树状结构是一种非线性结构,数据元素之间存在一对多的关系。每个数据元素有一个或者多个直接前驱,称为父元素;相应地,也可能有一个或者多个直接后继,称为子元素。树状结构在计算机科学中使用广泛,树状结构在某些目录结构、树状菜单、HTML 的 DOM 结构等多个领域有广泛的应用。如图 5.20 所示,广东理工学院的部门组织结构可以被视为一个树状结构,广东理工学院为根节点。广东理工学院下设的信息技术学院、会计学院、教务处、校团委等为广东理工学院的子节点。这些子节点同时也是各自子树的根节点,信息技术学院又有它自己的子节点基础教研室、软件工程教研室等。会计学院也有它的子节点会计学教研室、财务管理教研室等。教务处的根节点下面有子节点人事科、工资科等。校团委的根节点下面有子节点文体部、宣传部等。每个节点和子节点的关系构成了一个典型的树状结构,清楚地表达了组织内部的层次和关系。这类层级型的数据组织方式普遍存在于人们的生活和工作中。

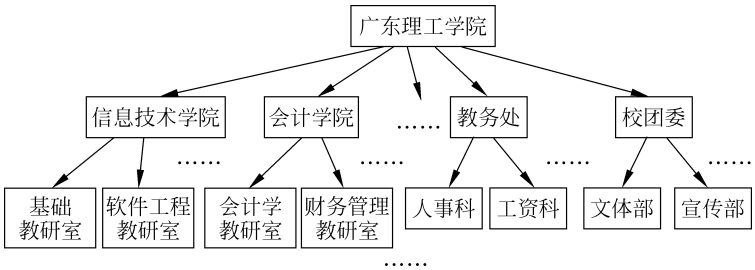



图 5.20 树状结构

(4) 图结构。

图结构是一种更复杂的非线性结构,数据元素之间存在多对多的关系。每个数据元素都可能与任何其他的数据元素相关联。在图结构中,数据元素可以被称为顶点,而数据元素之间的关系可以被表示为边。图结构在电路设计、网络优化等多个领域有广泛的应用。如图 5.21 所示,可以将每个城市当作一个节点,节点是图形结构的基本单位。一个节点可以与任何其他节点直接连接,而不仅仅是父节点或子节点。这就是图结构和树状结构主要的

区别。城市之间的航线可以作为连接这些节点的边。例如,北京(一个节点)和广州(另一个节点)之间的航班就是一条边。边可以是有向的(例如一条单向航线)或者无向的(双向航线)。边还可以有权重,表示两个节点之间的距离、成本或者需要的时间等。

2) 存储结构

存储结构是指数据在内存或磁盘中的存储方式,反映了数据元素之间的物理关系,包括数据的表示和存储以及数据元素间的逻辑关系如何在计算机中体现。存储结构主要有以下4种基本方式:顺序存储、链式存储、索引存储和散列存储。

(1) 顺序存储。

顺序存储结构是一种简单、最常用的数据存储结构。在顺序存储结构中,数据元素之间的存储关系与它们的逻辑关系一致,即逻辑相邻的两个元素在物理位置上也相邻。对于这种结构,可以通过下标直接访问其元素,实现非常快速的数据访问速度。如图5.22所示是包含数据元素 $X_0, X_1, X_2, X_3, \cdots, X_{n-1}$ 的顺序存储结构的示意图,其中,0,1,2,…,$n-1$ 为数据元素在数组中的下标。

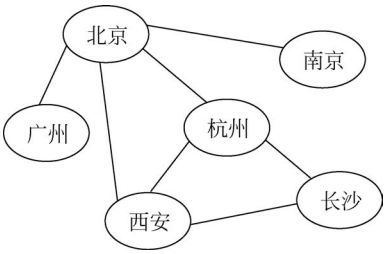



图5.21 图结构

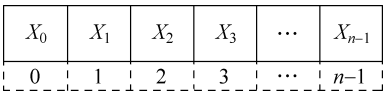



图5.22 顺序存储结构

顺序存储结构主要表现为4个特点:首先,它的存取速度非常快,因为通过数组索引,可以直接并且迅速地访问任何一个特定元素;其次,该结构具有连续的存储空间,但这也导致它无法进行动态的空间调整;然后,因为其结构的简洁性,所以对编程操作非常便捷;最后,由于元素的紧凑存储方式,因此对内存空间利用率非常高。

数组是顺序存储结构的典型代表,广泛应用在各种领域。例如,在绝大多数语言中,字符串就是由字符组成的数组。除此之外,算法如二分查找、快速排序等,都是基于顺序存储结构实现的。但是因为顺序存储方式的存储空间需要预分配,一旦分配后大小不易变动,所以可能会造成存储空间的浪费。另一方面,插入和删除操作需要移动大量元素,效率较低。

(2) 链式存储。

链式存储结构是一种数据结构中被广泛利用的存储模式。与顺序存储结构将数据紧密、顺序地放置在一起不同,链式存储结构允许数据元素在内存中非连续、非顺序地分布。而这些数据元素通过指针链接在一起,形成“链接”的方式构建起它们的逻辑结构。此类结构的主要表现就是各数据元素不必在物理位置上相邻,而是通过指针链接,实现逻辑上的连续。如图5.23所示是包含数据元素 $X_0, X_1, X_2, \cdots, X_{n-1}$ 的链式存储结构的示意图。

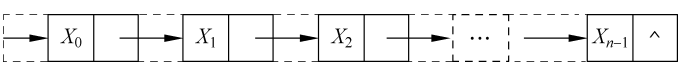



图5.23 链式存储结构

链式存储结构的特点集中于几个关键属性。首先，由于链式存储结构能弹性地按照需求进行存储空间的分配和回收，因此基本上不存在浪费存储空间的问题。其次，由于链式存储结构在插入或删除数据时，仅需更改相关节点的指向，故插入和删除操作具有很高的效率。然而，由于无法进行下标操作，链式存储结构不支持随机存取数据，这也是它的主要限制。

链式存储结构在许多领域都有着广泛的应用。例如，链表数据结构是最典型的使用链式存储结构的例子，包括单链表、双向链表和循环链表等。同时，许多高级数据结构，如各种互联网索引中的B树、B+树，在处理复杂的逻辑关系时，也都会选择使用链式存储结构。对于那些需要进行大量插入和删除操作，且不要求高速随机访问的应用场合，链式存储结构有着无法替代的优势。然而，对于需要快速查询和频繁随机访问的应用场景，链式存储结构则显得劣势明显。

(3) 索引存储。

索引存储结构可以看作数据存储的一种增强方式。通过构建索引，它改变了数据和其物理存储位置之间的直接关系，而将逻辑关系与物理关系解耦。这种结构下的数据元素并不直接显示其逻辑关系，而是依托索引进行检索和访问。每个索引都是一段标识信息和一组地址的组合，可以视为指向数据的指针，从而实现高效的数据查找和访问。

索引存储结构的重要特性主要体现在其增强的查找效率和动态调整的能力。首先，索引对数据的直接引用，使得查找速度大幅提高，尤其在处理大量数据时，索引存储的效率优势更为明显。其次，使用索引存储不必像顺序存储那样预先指定存储空间，而是可以根据需要动态分配，这带来了更大的灵活性。然而，索引存储也有其缺点。例如，索引本身需要额外的存储空间，给存储开销带来了增加。此外，维护索引的代价也不小，特别是在大量插入、删除数据时，索引的动态调整可能会造成效率的下降。

索引存储结构特别适用于数据库管理和文件系统等应用场景。例如，数据库中的B树和B+树都是典型的索引存储结构，通过索引实现对大量数据的高效管理和查询。在文件系统中，文件路径和指针的使用也体现了索引的思想，用于快速定位和读取文件。然而，索引存储也并非适用于所有场景，例如，对存储空间要求极高、数据变动频繁的情况，可能并不适合采用索引存储。

(4) 散列存储。

散列存储结构，或称哈希存储，是一种将数据以特定方式分布在存储空间中的存储模式。在散列存储结构中，通过一个函数(通常称为散列函数或哈希函数)将数据映射到一个较小的、固定的范围，形成一个散列值，这个散列值用来标识数据的存储位置。

散列存储结构的主要特性在于它的快速定位能力和空间利用效率。通过散列函数，可以实现数据的均匀分布，并且查找时间复杂度均摊下来是常数级别的。这使得在查找大量数据时，散列存储结构具有优良的性能。然而，散列存储也面临所谓的“哈希冲突”问题，即不同的输入可能映射到相同的散列值，解决这个问题需要额外的算法或数据结构。

在实际应用中，散列存储结构被广泛用于哈希表、缓存系统等领域。在计算机软件中，无论是编程语言提供的基础数据结构，如Python的字典、Java的HashMap，还是数据库中的索引，如MySQL的哈希索引，都是散列存储的运用。然而，散列存储不适合用于需要保持数据顺序的场景，如需要排序的数据集，或者需要维护元素间的顺序关系的情况。

## 5.7.2 表结构

表结构是数据结构中的一种基本形式，正确地使用和理解表结构对于理解和掌握数据结构尤为重要。表结构的主要特点是数据元素之间存在一种或多种特定的关系，元素之间的相对位置通常是按照某种规则或者特定需求确定。在实际情况中，从现实生活到计算机领域，表结构无处不在。如在日常生活中，人们会制作名单、清单或者表格来帮助整理和分类信息。在计算机领域，表结构成为保存和操作数据的一个基本形式，如数据库中的数据表、查询结果的展示、电子表格软件等。下面将分别讨论线性表、堆栈、队列等不同类型的表结构，详细介绍它们的定义、操作和具体实现。

**1. 线性表**

线性表是一种最基本且应用广泛的表结构类型，它是由初等数据元素组成的有序的元素序列。线性表的特点在于数据元素之间存在一对一的相互关系，即除首元素与末元素外，每个元素均有且只有一个直接前趋和直接后继。线性表可分为两类：顺序表和链表，如图 5.24 所示。

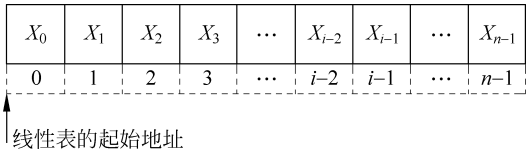



图 5.24　线性表的顺序存储结构

1）顺序表

（1）顺序表的定义。

顺序表是线性表的一种，它采用一组地址连续的存储单元依次存储线性表的元素。同时，它利用元素在表中的位置关系，通过计算得到对应元素的存储地址，因此能够随机地存取链表中的任意元素。

（2）顺序表的地址计算公式。

在顺序表中，元素的物理地址可以通过存储的基站（起始地址）和元素在线性表中的逻辑位置计算得到。假设表元素占用 $C$ 字节的存储空间，线性表的起始地址为 base（即元素 $X[0]$的存储地址），则第 $i$ 个元素的物理地址 $\mathrm{Loc}(X_i)$为

$$\mathrm{Loc}(X_i)=\mathrm{base}+i\times C \quad (0\leqslant i\leqslant n-1) \tag{5.1}$$

**【例 5.6】** 假设有一个整型顺序表，其中，整型数据在内存中占 4B，表中的元素为{10，20，30，40，50}，这些元素在内存中的起始地址（即元素 10 的地址）为 1000。求出元素 40 在内存中的物理地址。

根据顺序表地址计算公式 $\mathrm{Loc}(X_i)=\mathrm{base}+i\times C$ 可知 base 是线性表元素的起始地址，这里 base=1000。$C$ 是每个表元素占用的字节数，已知 $C=4$。$i$ 是表中数据元素为 40 的下标，由于下标是从 0 开始，所以对于数据元素 40 来说，$i=3$。将相关数值代入计算公式得 $\mathrm{Loc}(X_3)=1000+4\times 3=1012$。所以，顺序表中数据元素为 40 的物理地址为 1012。

（3）顺序表的特点。

随机访问：顺序表的最大优点就是能够实现随机存取。可以直接通过元素的序号（或称为下标、索引）快速找到该元素，读取或修改该元素的值。这一点与其他线性表（如链表）

相比，无论是从理论还是实际应用的角度来看，都是顺序表的一个显著优势。

高存储效率：顺序表中的每个元素在内存中都紧密相连，不存在额外的指针字段用于元素之间的链接（相比之下，链表则需要额外的内存空间来存放指针）。因此，顺序表的存储效率非常高，空间利用因子几乎可以达到1。

插入删除复杂：顺序表的插入和删除操作需要进行大量的元素移动。例如，如果在顺序表的起始位置插入一个元素，那么就需要将所有其他元素向后移动一位，以空出第一位来存放新插入的元素。对于包含大量元素的顺序表而言，这样的插入和删除操作可能会耗费较大的时间成本。

存储空间需要提前分配：在创建顺序表的时候，需要预分配一个足够大的连续存储空间来存放元素。但是，如果预分配的空间大于实际使用的空间，这部分多余的空间就会造成浪费。反之，如果所分配的空间过小，而且不能满足后续增加元素的需求，那么就需要重新分配空间，这又会带来额外的时间成本。

（4）顺序表的基本操作。

初始化：这是建立顺序表的第一步，需要分配一个连续的内存空间用来存放元素，同时设定表的初始最大容量和当前长度，通常开始时长度为0。初始化成功后，就会创建一个空的顺序表，并设置好了其基本属性。

查找：查找操作是顺序表最基本的功能之一。通常根据索引值（下标）进行查找，由于内存是连续分配的，因此可以快速定位到具体元素，实现了随机直接访问。

插入：在顺序表的特定位置插入新元素时，需要先将此位置与其后的所有元素移动一位，以空出空间给新的元素，然后在该位置插入新元素，并更新表的长度。如果内存不足以容纳新增元素，可能还需要执行扩容操作。

删除：删除元素的操作需要首先找到需要删除的元素，然后将此元素后的所有元素向前移动一位以填补被删除元素留下的空缺，最后更新表的长度。删除元素不会减少表的容量，除非选择缩减容量，但这并不常见。

获取长度：长度属性存储了顺序表中当前元素个数的信息，通常可以通过一个返回长度的函数来获取这个信息，用于判断表是否为空，或者是否已满等操作。

扩容：当顺序表因存储的元素过多而满时，需要申请一块更大的内存，并把原表的所有元素复制到新的内存中，然后释放原来的内存，同时更新表的最大容量。这个过程被称为扩容。

遍历：遍历是对顺序表中所有元素进行检查或操作的过程，常用于修改表中所有元素，或者找出符合特定条件的元素。

清空：清空操作是将所有的元素从顺序表中移除，让顺序表回到初始化后的状态。通常来说，可以简单地将表的长度设置为0，并不真正去删除每个元素。

2）链表

（1）链表的定义。

链表是一种由结点组成的线性数据结构，但与顺序表不同的是，链表中的元素并不在内存中连续存储，结点与结点之间是通过指针连接在一起的，形成一条链。每个结点包含两个部分：一个是存储数据元素值的数据域，另一个是存储下一个结点地址的指针域，如果一个结点只包含一个指针域，则称此链表为单链表，如图5.25所示。头指针（head）是一个特殊

的指针，其指向链表的开端，是整个链表的访问入口。头结点则指的是链表中的第一个节点，通常这个结点并不存储实际数据，而是作为链表的识别标志和简化链表操作的工具。紧跟在头结点后的第一个存储实际数据的结点称为首结点。相比之下，尾结点是链表中的最后一个结点，其指针部分指向 NULL，标志着链表的结束。

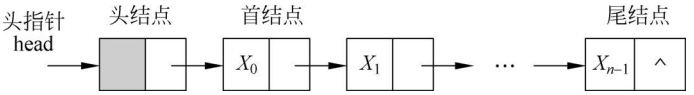



图 5.25　带头结点的单链表存储结构

（2）链表的特点。

动态结构与内存利用：链表是一种动态数据结构，这意味着无须预先知道数据的大小，可以在运行时实时增删元素，从而灵活调整链表的长度。这就为其提供了优于数组这类静态数据结构的优势，它不需要在创建时就指定大小。此外，由于链表中的元素可以在内存中任何地方，能够充分利用计算机内存，实现灵活的内存动态管理。

插入与删除的高效性：在链表中插入或删除一个元素非常高效。当进行插入或删除操作时，只需要调整元素间的链接指向，无须像操作数组那样移动其他元素。这也是链表相比于数组和其他线性数据结构的优势之一。

存取方式的灵活性：利用链表，可以以和元素在内存中的实际排列顺序不同的方式来存取数据。这一点对于数据结构的选择和算法的设计具有重要影响。同时，它也支持在链表中的任意位置插入和删除数据，这在进行大批量数据操作时具有显著优势。

查找效率和空间开销：链表的查找效率比较低，因为对于链表的查找需要从头开始逐个元素进行查找，这样会消耗大量的时间。另外，链表的空间开销比较大，由于每一个结点都需要一个额外的空间来存储下一结点或上一结点的地址，这就相对增加了空间复杂度。

（3）链表的基本操作。

创建链表：创建链表是链表操作的第一步，根据不同需求，可以创建空链表，或者在创建时就初始化几个结点。在创建链表时，通常首先创建一个头结点，来作为整个链表的起始。

插入结点：链表的插入操作可以在链表的任意位置进行，包括头部插入、尾部插入和中间插入。一般情况下，插入结点需要知道插入位置的前一个结点，然后将这个结点的下一结点链接指向新插入的结点，同时将新结点的链接指向原前一个结点所指向的下一结点。头部插入是将新结点变成头结点，原头结点成为新结点的下一结点。尾部插入是将新结点添加到链表尾部，原尾结点的下一结点设置为新结点。中间插入需要找到要插入位置的前一个结点，将其下一结点设置为新结点，新结点的下一结点设置为原下一结点。

删除结点：链表的删除也可以发生在链表的任意位置。在删除结点时，需要找到待删除结点的前一个结点和后一个结点，然后将前一个结点的链接直接指向后一个结点，这样就可以完成结点的删除。

查找结点：链表的查找操作是顺序查找，需要从头结点开始逐个比较，直到找到满足条件的结点或者抵达链表尾部。由于链表无法进行随机访问，查找操作的效率一般较低。

遍历链表：遍历操作是为了访问链表中所有的结点。一般从头结点开始，逐个访问每个结点，直到该结点的链接为 NULL，即到达链表尾部。

**2. 堆栈**

1）栈的定义

栈是一种性质特殊的线性数据结构。它由一系列顺序排列、同类型的数据元素构成，每个元素都可以连接其前一个元素，形成功能上连续的线性结构。栈的显著特点是其先进后出(First In Last Out，FILO)或后进先出(Last In First Out，LIFO)的性质，新的数据元素插入和旧的数据元素的删除都在被称为栈顶(top)的一端进行，而相对固定的一端称为栈底(bottom)。假设栈中的数据元素序列为$\{X_0, X_1, X_2, X_3, \cdots, X_{n-1}\}$，则 $X_0$ 称为栈底元素，$X_{n-1}$ 称为栈顶元素，$n$ 为栈中数据元素的个数(当 $n$ 为 0 时，此时栈为空)。通常，将栈的插入操作称为入栈或者压栈，栈的删除操作称为出栈或者弹栈，如图 5.26 所示。此外，根据存储方式，栈又可区分为静态栈和动态栈，前者通常用数组实现且大小在编译时确定，后者经常使用链表实现且可以在运行时动态调整。在诸如数据缓存、程序调用等场景中，栈发挥了重要作用。

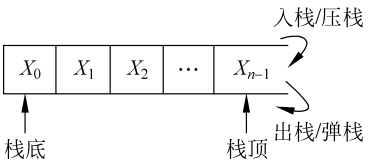


图 5.26　栈结构示意图

2）栈的特点

栈在内存中以线性的方式存放数据元素。根据实现方式的不同，栈的数据元素可以在物理内存中连续存放，这类栈称为顺序栈。另外，数据元素也可以在物理内存中非连续存储，这种非连续存储的栈称为链栈。

后进先出：栈的核心特性是后进先出。这意味着最后放入栈的元素将会被首先取出。这种特性尤其适用于某些需要使用临时存储的场景，如函数的嵌套调用、表达式求值等。

限定访问：栈的访问是受限的。只能在栈顶进行数据的插入和删除操作，不能在栈的其他位置进行数据的操作。这种限定访问的方式有助于更有效地控制和管理数据。

操作限制：栈不允许随机访问，即不能直接通过索引访问栈中的任意元素，只能通过栈顶进行操作。

3）栈的基本操作

入栈操作：添加元素到栈顶，栈的大小会增加一个数据单位。如果栈已满或者没有足够的空间，这个操作可能会无法执行，通常会产生异常或错误信息。

出栈操作(出栈)：移除栈顶的元素，并返回这个元素的值。当执行这个操作时，栈的大小会减少一个数据单位。如果栈已经为空，这个操作则无法执行，然后会有异常或错误信息产生。

取栈顶操作：返回栈顶的元素，但并不移除。这允许用户查看或者访问栈顶的数据，而不改变栈的大小或数据结构。

判断栈空操作：检查栈是否为空。如果栈内没有任何元素，这个操作会返回 true，否则返回 false。这个操作常用于判断栈的状态，以免在空栈上做出入栈或出栈操作导致错误。

**【例 5.7】** 考虑一个栈，其中元素按照以下顺序进行了入栈操作：1，2，3，4，5(1 最先入栈，5 最后入栈)。然后连续进行三次出栈操作。栈顶此时的元素是多少？

根据栈的后进先出 (LIFO) 原则，最后进栈的元素会最先被弹出。因此，连续执行三次弹出操作后，分别将 5、4 和 3 弹出栈，因此栈顶的元素此时为 2。

**3. 队列**

1）队列的定义

队列是一种特殊的线性数据结构，其操作规则为先进先出(First-In-First-Out，FIFO)。元素的插入操作称为入队，并且只允许在队列的一端进行，允许进行插入的一端称为队尾(rear)，而元素的删除操作称为出队，允许进行删除的一端称为队首(front)。假设队列中的数据元素序列为$\{X_0, X_1, X_2, X_3, \cdots, X_{n-1}\}$，则$X_0$称为队首元素，$X_{n-1}$称为队尾元素，$n$为队列中数据元素的个数(当$n$为0时，此时队列为空)，如图5.27所示。

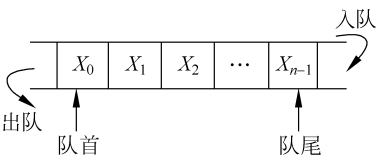



图5.27　队列结构示意图

2）队列的特点

线性存储：队列同样在内存中以线性的方式存放数据元素。根据实现方式的不同，队列的数据元素可以在物理上连续存放，该队列称为顺序队列。另外，数据元素也可以在物理内存中非连续存储，这种非连续存储的队列称为链队列。

先进先出：队列的核心特性是先进先出。这意味着最先放入队列的元素将会被首先取出。这种特性尤其适用于某些需要按照一定顺序处理数据的场景，如打印任务的调度、事件的调度等。

两端访问：与栈不同，队列允许在其两端进行操作。入队操作(插入元素)发生在队尾，而出队操作(删除元素)发生在队首。这种方式保证了队列元素的流动性，符合先进先出的原则。

操作限制：和栈一样，队列同样不支持随机访问，即不能直接通过索引访问队列中的任意元素。队列中插入和删除元素的操作只能分别在队尾和队首进行。这种限制简化了队列内部的操作和管理，有利于维护队列的数据一致性。

3）队列的基本操作

入队操作：在队列的尾部(也称为后端)插入一个新的元素。此操作也会将新元素设置为队列的队尾元素，并更新队列长度。

出队操作：从队列的头部(也称为前端)删除一个元素。此操作会使当前队首的下一元素成为新的队首，并更新队列的长度。

取队首元素操作：返回队首的元素，但不删除它。这样可以查看队首元素。在队列为空时，将返回错误或异常。

求队列长度操作：返回队列中现有元素的数量。反映了队列的大小或容量。

判断队列是否为空操作：返回一个布尔值，判断队列是否有元素。如果队列为空，返回值为true，如果不为空，则返回值为false。

**【例5.8】** 假设你是一个城市的交通管理者，每个路口都有一个停车区域，车辆需要按照到达路口的顺序先进后出，即对应到数据结构中的队列概念。你需要设计一种策略，使得以下的车辆管理按照队列管理运行。

假设有4辆车A、B、C和D顺序到达路口，停车区域只允许一个车辆出一次，也就是说，车辆必须按照其到达路口的顺序出停车区。请回答下列问题：

(1) 请给出上述4辆车从停车区域出去的顺序。

(2) 如果在车辆 B 出停车区后，又新来一辆车 E，请列出此时各车辆的排队情况。

解答：

按照队列的先入先出(FIFO)原则，4 辆车从停车区出去的顺序应该是 A、B、C、D。

在车辆 B 出停车区后，新来一辆车 E，此时车辆的排队情况应为 A 已出队，B 已出队，所以队列中的车辆为 C、D、E。

### 5.7.3 树结构

**1. 树**

1) 树的定义

树(Tree)是 $n(n\geqslant 0)$ 个结点的有限集 $T$，$T$ 为空时称为空树，否则它满足如下两个条件。

(1) 每个元素称为结点(Node)，有且仅有一个特定的称为根(Root)的结点，并且满足该根结点只有后继结点没有前驱结点，如图 5.12 中的 A 结点所示。

(2) 其余的结点可分为 $m(m\geqslant 0)$ 个互不相交的子集，如图 5.28 所示包含三个子集，分别为 $T_1$(B,E,F)，$T_2$(C,G,H)，$T_3$(D,I)，其中每个子集又是一棵树，并称其为子树(Subtree)，这三棵子树的根结点分别是 B、C、D。

结点个数为 0 的树称为空树，一棵树可以只有根但没有子树($m=0$)。一棵单结点的树如图 5.29 所示，只包含一个根结点。

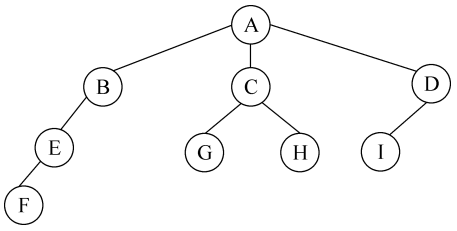



图 5.28 树


图 5.29 单结点树

树是一种层次性结构：子树的根看作树根的下一层元素，一棵树里的元素可以根据这种关系分为一层层的元素。一棵树(除树根外)可能有多棵子树，根据子树的排列顺序是否有意义，可以把树分为有序树和无序树两种。例如，普通的树一般是无序的，二叉搜索树(BST)是有序的。

2) 树的常用术语

(1) 树的结点。

树的结点就是构成树的数据元素，在树状表示法中用圆圈表示，如图 5.29 中只有一个结点，图 5.28 中有 9 个结点。

(2) 叶结点或终端结点。

度为 0 的结点称为叶结点，如图 5.28 中的树的叶子结点为 F、G、H、I。

(3) 非终端结点或分支结点。

度不为 0 的结点为分支结点，如图 5.28 中的分支结点为 A、B、C、D、E。

(4) 双亲结点或父结点。

若一个结点含有子结点，则这个结点称为其子结点的父结点，如图 5.28 所示，A 为 B、

C、D 的双亲结点或者父结点。

(5) 孩子结点或子结点。

一个结点含有的子树的根结点称为该结点的子结点，如图 5.28 所示，B、C、D 为 A 的孩子结点。

(6) 兄弟结点。

具有相同父结点的结点互称为兄弟结点，如图 5.28 所示，B、C、D 为兄弟结点。

(7) 结点的度。

一个结点含有的子树的个数称为该结点的度，如图 5.28 所示，结点 A 的度为 3，结点 B 的度为 1，结点 C 的度为 2，结点 D 的度为 1。

(8) 树的度。

一棵树中，最大的结点的度称为树的度，如图 5.28 所示，最大的结点的度为 3，则树的度为 3。

(9) 路径和路径长度。

从一个祖先结点到其子孙结点的一系列边称为树中的一条路径。显然，从一棵树的根到树中任一个结点都有路径，且路径唯一，路径中边的条数称为路径长度。如图 5.28 所示，结点 G 的路径为 A→C→G，结点 G 的路径长度为 2。

(10) 结点的层次。

从根开始定义起，根为第 0 层，根的子结点为第 1 层，以此类推，如图 5.28 所示，结点 H 的层次数为 2。

(11) 树的高度或深度。

树中结点的最大层次＋1 即为树的深度，如图 5.28 所示，树的深度为 4。

(12) 堂兄弟结点。

双亲在同一层的结点互为堂兄弟，如图 5.28 所示，E、G 结点互为堂兄弟结点。

(13) 结点的祖先。

一个结点的祖先是指从根结点到该结点路径上的所有结点，根结点本身也被视为自己的祖先。

(14) 子孙。

以某结点为根的子树中任一结点都称为该结点的子孙。

(15) 森林。

由 $m(m\geqslant 0)$棵互不相交的树组成的集合称为森林，如图 5.30 所示。

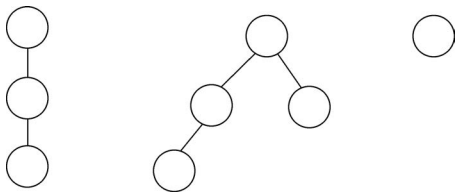


图 5.30　森林

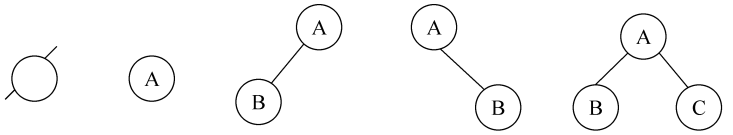
**2. 二叉树**

1) 二叉树的定义

二叉树是由 $n(n\geqslant 0)$个结点的有限集合构成，此集合或者为空集，或者由一个根结点及

两棵互不相交的左右子树组成，并且左右子树都是二叉树。

这也是一个递归定义。二叉树可以是空集合，根可以有空的左子树或空的右子树。二叉树不是树的特殊情况，它们是两个概念。二叉树是每个结点最多有两个子树的树结构。它有 5 种基本形态：二叉树可以是空集；根可以有空的左子树或右子树；或者左、右子树皆为空，如图 5.31 所示。




(a) 空二叉树　(b) 只有根　(c) 根和左子树　(d) 根和右子树　(e) 根和左右子树

图 5.31　二叉树的 5 种形态

所有的结点都只有左子树（左斜树）如图 5.32 所示，或者只有右子树（右斜树）如图 5.33 所示。

2）满二叉树

如果所有的分支结点都存在左子树和右子树，并且所有的叶子结点都在同一层上，则称这棵树为满二叉树，如图 5.34 所示。

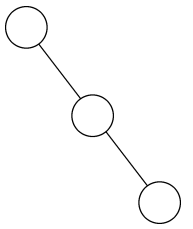


图 5.32　左子树

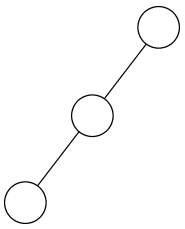


图 5.33　右子树

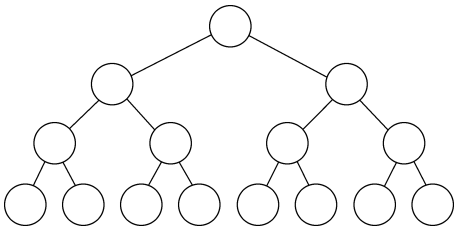


图 5.34　满二叉树

根据满二叉树的定义，得到其特点如下。

（1）所有的叶子结点都在最底层。在任意一层上，如果有叶子结点，那么所有分支结点必然都有两个子结点。

（2）在任意一层上，结点数目是该层最大结点数目。也就是说，如果一棵二叉树的深度为 $h$，那么该二叉树的第 $i$ 层上最多有 $2i-1$ 个结点（$i \geqslant 1$）。

（3）一棵深度为 $h$ 并且含有 $2h-1$ 个结点的二叉树，称为满二叉树。

（4）满二叉树的结点的度数要么为 0（即为叶子结点），要么为 2（即分支结点），不可能存在只有一个子结点的结点。

3）完全二叉树

对一棵具有 $n$ 个结点的二叉树按层序排号，如果编号为 $i$ 的结点与同样深度的满二叉树编号为 $i$ 的结点在二叉树中位置完全相同，就是完全二叉树。满二叉树必须是完全二叉树，反过来不一定成立。

如图 5.35 所示就是一棵完全二叉树。结合完全二叉树定义，得到如下特点。

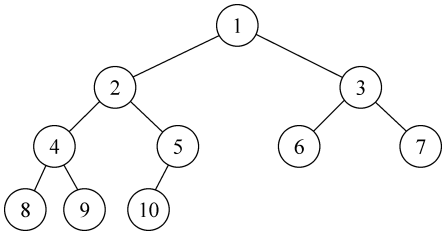



图 5.35　完全二叉树

(1) 在完全二叉树中除最后一层外,所有其他层的结点数量都达到最大。

(2) 在完全二叉树最下面一层,所有结点都连续排列在左侧,按从左至右的顺序分布。

(3) 在完全二叉树的结点分布中,总是遵循从上至下,再从左至右的顺序原则。

(4) 由于完全二叉树的有序性和连续性,完全二叉树在诸如二叉堆和优先队列等多种数据结构和算法中有着广泛应用。

### 5.7.4 图结构

**1. 图**

1) 图的定义

图(Graph)是由顶点(Vertex)集合及边(Edge)集合组成的一种数据结构。边是顶点之间的关系。一个图 $G$ 可以表示为 $G=(V,E)$,其中,$V$ 是顶点的有穷非空集合,$E$ 是边的有穷集合。$E$ 可以为空集,$E$ 为空时,图 $G$ 中没有边。一个简单的图如图 5.36 所示。

其中,顶点集合 $V=\{v_0,v_1,v_3\}$,边集合 $E=\{(v_0,v_1),(v_0,v_2),(v_0,v_3)\}$。思考:顶点 $v_0$ 和 $v_1$ 之间的边是否可以表示为$(v_0,v_1)$?

2) 图的常用术语

(1) 无向图中的边没有方向,每条边用两个顶点的无序对(用圆括号括起来)表示,如图 5.36 所示就是一个无向图,顶点 $v_0$ 和 $v_1$ 之间的边既可表示为$(v_0,v_1)$,也可表示为$(v_1,v_0)$。

(2) 有向图。有向图的边是有方向的,每条边用两个顶点的有序对(用尖括号括起来)表示,如图 5.37 所示的有向图中包含的边有$<v_0,v_1>$,$<v_2,v_0>$,$<v_0,v_3>$。注意,顶点 $v_0$ 和 $v_1$ 之间的边是有方向的,不能写成$<v_1,v_0>$。有向图的边又称为弧(Arc),箭头的尾部称为弧尾,头部称为弧头。

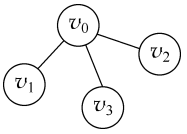



图 5.36 一个简单的无向图

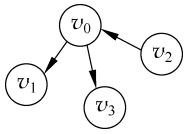
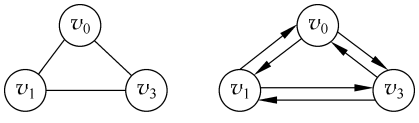



图 5.37 有向图

(3) 完全图。完全图是指图中边的数量达到最大值。$n$ 个顶点的无向完全图具有 $n\times(n-1)/2$ 条边,$n$ 个顶点的有向完全图具有 $n\times(n-1)$条边,如图 5.38 所示。

(4) 带权图。带权图是指图中的边带有权值,带权图又称为网(Network),如图 5.39 所示。




(a) 无向完全图 (b) 有向完全图

图 5.38 无向完全图和有向完全图

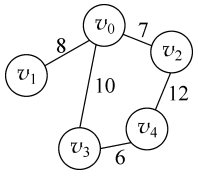



图 5.39 带权图

在不同场景下,带权图可以有不同的含义。例如,顶点可以表示不同的城市,顶点之间的边表示不同城市之间的通信线路,边上的权值表示建立这条通信线路所需花费的代价。

(5) 邻接顶点。若$(v_i,v_j)$是无限图$G$的一条边，则称$v_i$和$v_j$互为邻接顶点(Adjacent Vertex)，又称边$(v_i,v_j)$依附于顶点$v_i$和$v_j$，$v_i$和$v_j$依附于边$(v_i,v_j)$。若$<v_i,v_j>$是有向图$G$的一条边，则称顶点$v_i$邻接到顶点$v_j$，顶点$v_j$邻接自顶点$v_i$，边$<v_i,v_j>$与顶点$v_i$和$v_j$相关联。

(6) 顶点的度。顶点的度(Degree)指的是与顶点关联的边的数量。例如，图5.39中顶点$v_0$的度为3。

有向图中，顶点的度包括入度和出度。以顶点$v_i$为终点的弧的数量称为$v_i$的入度；以顶点$v_i$为起点的弧的数量称为$v_i$的出度。顶点$v_i$的度是入度和出度之和。例如，图5.37中顶点$v_0$的入度为1，出度为2，度为3。

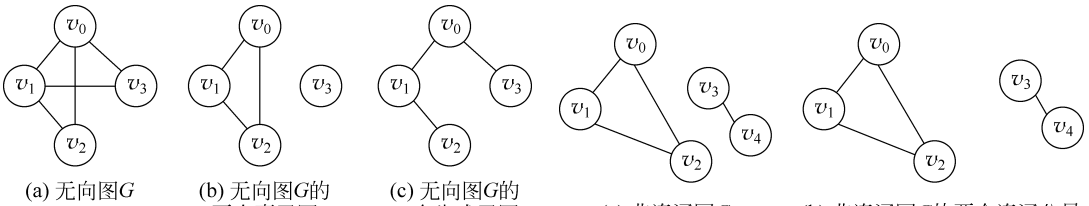
(7) 子图。设图$G=(V,E)$，对于图$G'=(V',E')$，若$V'\subseteq V$且$E'\subseteq E$，则称图$G'$是$G$的子图。若$G'\neq G$，则称$G'$是$G$的真子图。若$G'$是$G$的子图，且$V'=V$，称$G'$是$G$的生成子图。图5.40展示了无向图$G$及其两个真子图和一个生成子图。

(8) 路径。在图$G$中，如果从顶点$v_i$可以到达顶点$v_j$，则称从顶点$v_i$到顶点$v_j$经过的顶点序列为它们之间的路径(Path)。例如，图5.40(b)中顶点$v_0$到$v_2$之间的路径有多条$(v_0,v_2)$、$(v_0,v_1,v_2)$等，顶点$v_0$和$v_3$之间没有路径。若图$G$是有向图，则路径也是有方向的。

无权图中，路径长度指路径上边的数量。带权图中，路径长度指路径上各边权值之和。

简单路径是指路径上没有重复的顶点，例如，图5.40(b)中顶点$v_0$到$v_2$之间的路径$(v_0,v_1,v_0,v_2)$就不是简单路径。回路是指起点和终点相同且长度大于1的路径，回路又称为环。例如，图5.24(b)中路径$(v_0,v_1,v_2,v_0)$是一个回路。

(9) 连通性。在无向图中，若顶点$v_i$到$v_j$之间有路径，则称$v_i$和$v_j$是连通的。若无向图$G$中任意两个顶点都是连通的，则称该图为连通图。非连通图的极大连通子图称为该图的一个连通分量。例如，图5.40(a)和图5.40(c)都是连通图，如图5.41所示为非连通图$G$和它的连通分量。




(a) 无向图$G$　(b) 无向图$G$的两个真子图　(c) 无向图$G$的一个生成子图

图5.40　无向图$G$的真子图和生成子图

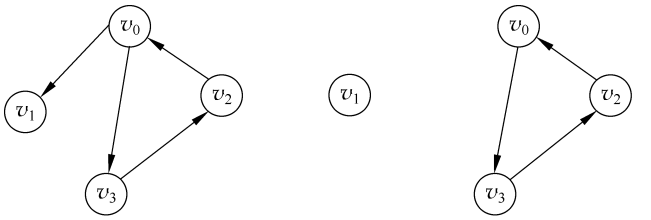
(a) 非连通图$G$　(b) 非连通图$G$的两个连通分量

图5.41　非连通图$G$和它的连通分量

在有向图中，若每一对顶点之间都是连通的，则称该图为强连通图(Strongly Connected Graph)。非强连通图的极大强连通子图称为该图的强连通分量，例如，如图5.42所示为非强连通图($v_1$到$v_0$之间没有路径)和它的强连通分量。

(10) 生成树和生成森林。生成树是一种特殊的生成子图，它包括图中的全部顶点，但是只有构成一棵树所需的$n-1$条边。例如，图5.40(c)是图5.40(a)的一个生成树。

对于非连通图，每个连通分量可形成一棵生成树，这些生成树组成了该图的生成森林。




(a) 非强连通图$G$　　(b) 非强连通图$G$的两个强连通分量

图 5.42　非强连通图 $G$ 和它的强连通分量

**2. 图的存储**

1）邻接矩阵

图的邻接矩阵表示指的是使用顺序存储结构存储图的顶点信息，采用邻接矩阵存储图的边信息。邻接矩阵是表示图中各顶点之间邻接关系的矩阵。根据边是否带权值，邻接矩阵有不同的含义。

(1) 无权图的邻接矩阵。

设图 $G$ 有 $n$ 个顶点，顶点集合 $\boldsymbol{V}$ 可表示为 $\boldsymbol{V}=(v_0,v_1,\cdots,v_{n-1})$，边集 $E$ 可用一个邻接矩阵 $\boldsymbol{A}_{n\times n}$ 表示，$\boldsymbol{A}_{n\times n}$ 的元素 $a_{ij}$ 表示图 $G$ 中顶点 $v_i$ 到 $v_j$ 之间的邻接关系，若 $v_i$ 和 $v_j$ 之间有边，则 $a_{ij}=1$，否则 $a_{ij}=0$。$\boldsymbol{A}_{n\times n}=[a_{ij}](0\leqslant i,j<n)$的定义如下。

$$a_{ij}=\begin{cases}1, & \text{若 } v_i \text{ 到 } v_j \text{ 之间有边}\\ 0, & \text{若 } v_i \text{ 到 } v_j \text{ 之间没有边}\end{cases}$$

无向图 $G_1$ 和有向图 $G_2$ 的邻接矩阵如图 5.43 和 5.44 所示。

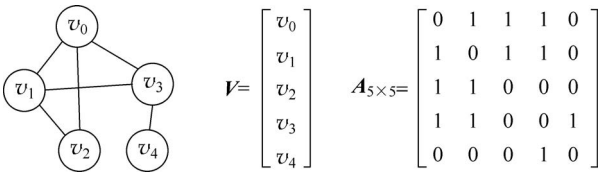



图 5.43　无向图 $G_1$ 和它的邻接矩阵表示

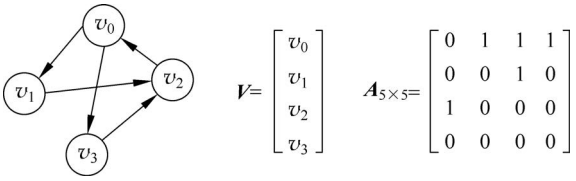



图 5.44　有向图 $G_2$ 和它的邻接矩阵表示

无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定对称。

(2) 带权图的邻接矩阵。

带权图的邻接矩阵 $\boldsymbol{A}_{n\times n}=[a_{ij}](0\leqslant i,j<n)$的定义如下。

$$a_{ij}=\begin{cases}w_{ij}, & \text{若 } v_i \text{ 到 } v_j \text{ 之间有边}\\ \infty, & \text{若 } v_i \text{ 到 } v_j \text{ 之间没有边}\end{cases}$$

其中，$w_{ij}(w_{ij}>0)$表示 $v_i$ 到 $v_j$ 之间边的权值，$\infty$表示 $v_i$ 到 $v_j$ 之间没有边。

带权有向图 $G_3$ 的邻接矩阵如图 5.45 所示。

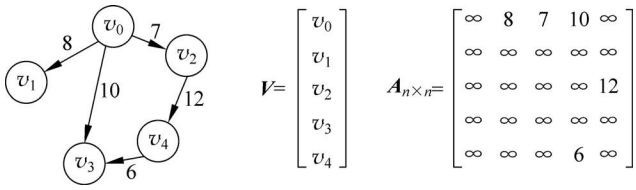



图 5.45　带权有向图 $G_3$ 的邻接矩阵表示

用邻接矩阵表示图，很容易判断任意两个顶点之间是否有边，同时也很容易求出各个顶点的度。无向图的邻接矩阵第 $i$ 行或第 $i$ 列的非零元素的个数为顶点 $v_i$ 的度；有向图的邻接矩阵第 $i$ 行非零元素的个数为顶点 $v_i$ 的出度，第 $i$ 列非零元素的个数为顶点 $v_i$ 的入度。

2）邻接表

图的邻接表（Adjacency List）表示，指采用顺序存储图的顶点集合，采用邻接表存储图的边集合。无向图和有向图的邻接表不同。

邻接表采用链式存储结构存储图，由一个顺序存储的顶点表和多个链式存储的边表组成。边表个数和图的顶点数相同。顶点表由顶点结点组成，每个顶点结点又由数据域和指针域组成，其中，数据域 data 存放顶点值，指针域 firstArc 指向边表中的第一个边结点。边表由边结点组成，每个边结点由 adjVex、value、nextArc 几个域组成，其中，value 存放边的信息，如权值；adjVex 存放与该节点邻接的顶点在顶点表中的位置。nextArc 指向下一个边结点。

如图 5.46 所示为带权无向图 $G_4$ 的邻接表表示。

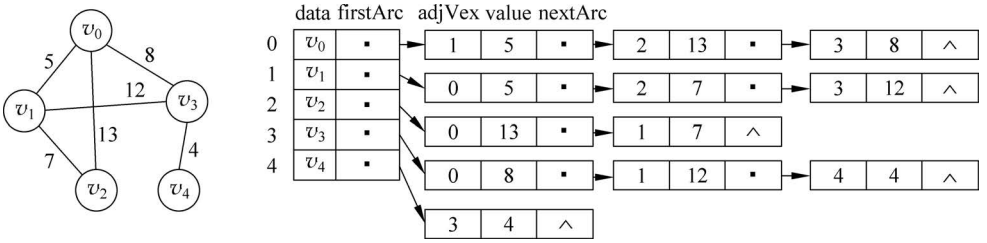



图 5.46　带权无向图 $G_4$ 的邻接表表示

（1）无权图的邻接表。

对于无向图而言，邻接表的表示方法也可以快速地得到每个顶点的度（只需查看顶点对应边链表中边结点的数量）。无向图的邻接表中每条边会出现两次。

如图 5.47 所示为带权无向图 $G_5$ 的邻接表表示。

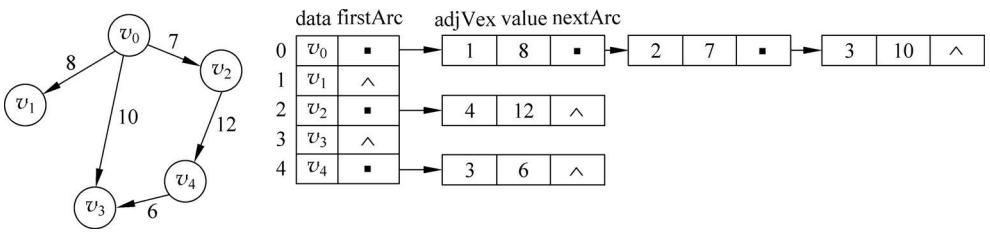



图 5.47　带权无向图 $G_5$ 的邻接表表示

(2) 带权图的邻接表。

带权有向图的邻接表表示方法不会重复存储边的信息,但是比较难计算一个顶点的度。顶点对应边链表的边结点数为该顶点的出度,要想计算顶点的入度则需要查看整张表,这时可以配合逆邻接表得到顶点的入度,如图 5.48 所示。

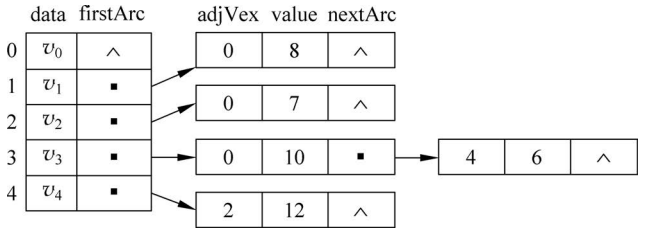



图 5.48　带权无向图 $G_5$ 的逆邻接表表示

在逆邻接表中,每个顶点对应的边链表中存储的是指向它的边,这样就可以快速计算一个顶点的入度,但是会耗费额外的存储空间。

## 5.7.5　查找与排序

### 1. 查找

1) 查找的基本概念

查找:在查找表中查询满足某种查询条件的记录而进行的过程即为查找。若能在指定的查找表中找到满足给定条件的记录,则称为查找成功,并返回该记录在查找表中的位置。若在指定的查找表中找不到满足给定条件的记录,则称查找不成功,则返回-1。

查找表:查找表是一种将同一类型的记录按照某种逻辑关系组织在一起的数据结构。这些记录可以按照线性表、树状等结构组织起来。线性表中的每一行即为一条记录。树状结构中每个结点就是一条记录。每条记录中可以包含一个或多个数据项,而人们往往会把用作查找依据的数据项称为关键字项,数据项的值称为关键字。

例如,可以将表 5.6 看成一个查找表,每一行数据就是一条记录,代表着一位学生的成绩信息。每条记录又包括学号、姓名、大学英语等 6 个数据项,这些数据项都可以作为关键字项来进行查找。

**表 5.6　学生成绩表**

| 学　号 | 姓　名 | 大学英语 | 高等数学 | 数据结构与算法 | 总　分 |
|---|---|---|---|---|---|
| 104 | 王一 | 60 | 81 | 78 | 219 |
| 102 | 钱二 | 95 | 76 | 82 | 253 |
| 101 | 王一 | 90 | 78 | 72 | 240 |
| 103 | 张三 | 94 | 76 | 80 | 250 |
| 106 | 李四 | 95 | 73 | 71 | 239 |
| 109 | 王五 | 85 | 83 | 80 | 248 |

若某个数据项的值能够唯一标识一条记录,则称这个数据项的值为该记录的主关键字。不同的记录,其主关键字不同。换句话说,以主关键字作为查询条件,那么查找结果一定是唯一的,例如,以“学号=101”为查询条件在表 5.1 中进行查找,只会得到一条记录的查询结果,则 101 就是这条记录的主关键字。

若某个数据项的值能够标识多条记录，则称这个数据项的值为次关键字。假如以“姓名＝王一”为查询条件在表 5.1 中进行查找，查找结果不唯一，则称“王一”为次关键字。

简单起见，本章所涉及的关键字均指主关键字，并假设主关键字的数据类型均为整型。

因此查找某条记录的操作实际上可以转换成在查找表中查找某条记录的关键字等于待查找值的过程。只要能在查找表中找到某条记录的关键字与待查找值相等，就等同于找到了这条记录，从而可以忽略记录中的其他数据项，这样就简化了查找操作。

为简化查询操作，可以将一条记录看成一个记录结点，记录结点包括两部分：关键字部分和除关键字以外的数据项部分。可将表 5.1 中数据转换成如图 5.49 所示。

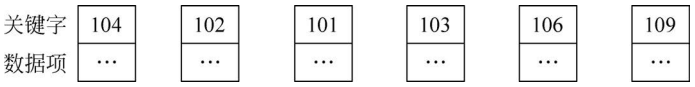
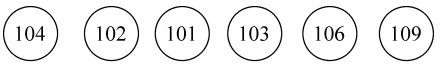
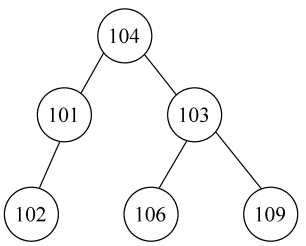



图 5.49　学生成绩表简化图

将图 5.49 中的每条记录的关键字组织成线性结构排成一行，如图 5.50(a)所示；组织成树状结构，如图 5.50(b)所示。




(a) 线性结构查找表




(b) 树状结构查找表

图 5.50　线性结构与树状结构查找表

因此为了提高查找效率，在实际应用中往往会将若干记录根据具体要求按照一对一、一对多等关系组织起来形成线平均查找长度

平均查找长度：查找算法的关键操作是关键字与待查找值的比较，通常把查找过程中关键字与待查找值之间执行的平均比较次数的期望值称为平均查找长度(Average Search Length，ASL)。平均查找长度是衡量一个查找算法效率的评价依据。

对于一个含有 $n$ 条记录的查找表，查找成功时的平均查找长度为

$$\mathrm{ASL}=\sum_{i=1}^{n}P_iC_i \tag{5.2}$$

其中，$n$ 是查找表中记录的总条数；$P_i$ 是查找第 $i$ 条记录的概率。若不特别声明，认为每条记录的查找概率相等，即 $P_1=P_2=\cdots=P_i=\cdots=P_n=1/n$；$C_i$ 是找到第 $i$ 条记录的关键字与待查找值相等时，第 $i$ 条记录的关键字与待查找值所需要进行的比较次数。比较次数 $C_i$ 取决于待查找值在表中的位置。

2) 顺序查找

顺序查找是在线性表上进行的一种查找方法，因此又叫作线性查找。顺序查找适用于存储结构为顺序存储或链式存储的线性表，而且对顺序表或者链表中记录的关键字的排列次序无任何要求，属于无序查找算法。本节仅讨论以顺序表为存储结构的查找过程。

基本思想：从顺序表的一端开始向另一端顺序扫描，依次将每条记录的关键字与待查找值进行比较，若某个记录的关键字等于待查找值，则查找成功，并返回该记录在顺序表中的位置；若整个顺序表中的记录都与待查找值比较完毕，但仍找不到关键字与待查找值相等的记录，则查找失败，并返回－1。

**【例 5.9】** 已知 11 条记录($n$＝11)的关键字序列为{10，20，30，40，45，60，70，75，80，

90,100},要求采用顺序查找的方法完成以下功能。

(1) 查找关键字为 75 的记录,若找到,则输出该记录在表中的位置以及比较次数,否则给出查找失败的提示信息以及比较次数。

(2) 查找关键字为 65 的记录,若找到,则输出该记录在表中的位置以及比较次数,否则给出查找失败的提示信息以及比较次数。

分析过程:

(1) 查找关键字为 75 的记录。

顺序查找的实现通常通过遍历序列来完成,从序列的开始到序列的结束,依次将序列中的每个元素与期望找到的关键字进行比较,直到找到一个与关键字对应的元素或到达序列的尾部为止。在这个过程中,进行比较的次数与找到的元素在序列中的位置是紧密联系的。

在这个情况下,序列为{10,20,30,40,45,60,70,75,80,90,100},期望找到的关键字为 75。从序列的开始处,即数字 10 开始比较,每次比较的结果都与 75 不符,直至第 8 个数字 75,此时找到了与关键字相符的元素。因此,关键字 75 在序列中的位置是 8,对应的比较次数也是 8,因为在找到关键字 75 之前,需要对序列前面的 7 个元素以及数字 75 进行比较。整个查找的过程如图 5.51 所示。

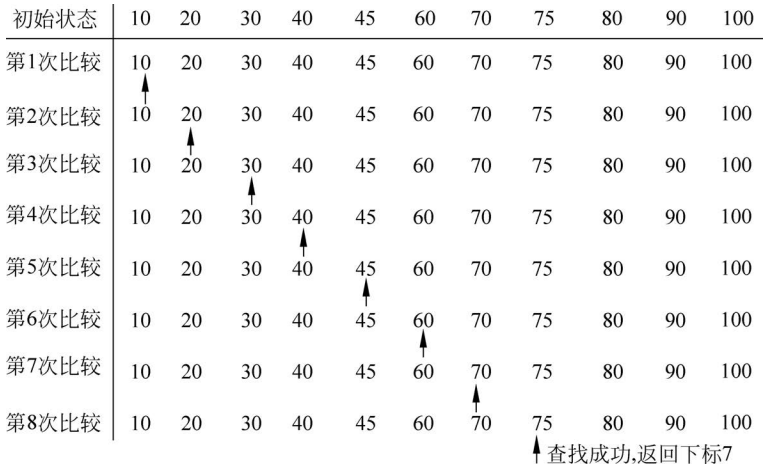



图 5.51 顺序查找数据元素为 75 的元素

(2) 查找关键字为 65 的记录。

同样地,还是需要遍历整个序列,依次将序列中的每个元素与期望找到的关键字进行比较,直到序列尾部。在这个情况下,仍然是序列{10,20,30,40,45,60,70,75,80,90,100},但期望找到的关键字是 65。

从序列的开始处数字 10 开始比较,但直至序列结束数字 100,都还没有找到与关键字 65 相等的元素。这表示在序列中不存在与关键字 65 对应的元素,因此会输出查找失败的提示信息。

由于在查找过程中需要遍历整个序列,在序列结束后还没有找到所期望的关键字,所以整个查找过程的比较次数为 11,也就是序列中元素的数量。因此,比较次数为 11。整个查找的过程如图 5.52 所示。

结合上述过程可知,要想成功找到第 $i$ 条记录($0 \leqslant i \leqslant n-1$),需要与待查找关键字的比

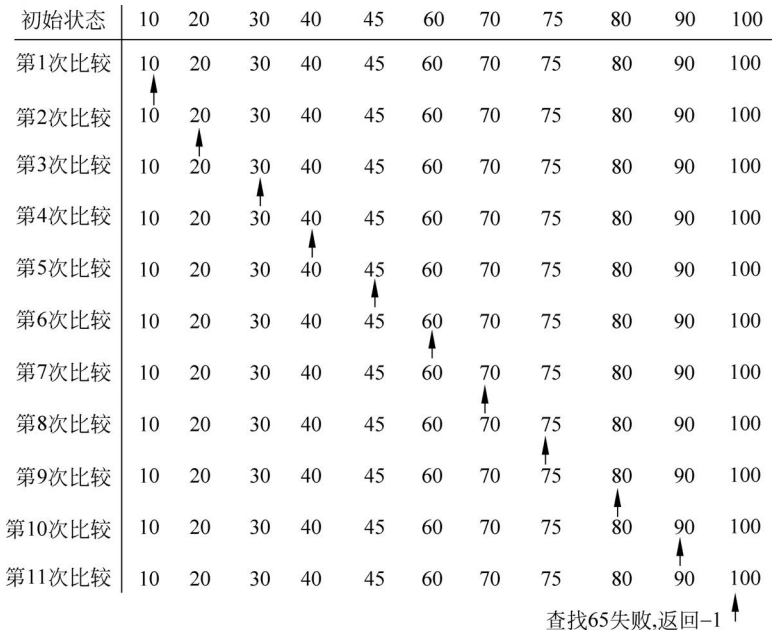



图 5.52　顺序查找数据元素为 65 的元素

较次数为 $i+1$，假设每条记录都能被找到，则每条记录的查找概率相等均为 $1/n$，则其成功时的平均查找长度为总的成功查找次数除以总的记录数：

$$\mathrm{ASL}=\frac{1+2+3+\cdots+n}{n}=\frac{1}{n}\sum_{i=0}^{n-1}(i+1)=\frac{n+1}{2} \tag{5.3}$$

查找失败时，待查找关键字一共需要与 $n$ 条记录的关键字比较 $n$ 次。因此不管是查找成功还是查找失败，顺序查找的时间复杂度均为 $O(n)$。可见，当 $n$ 较大时，查找效率很低，所以顺序查找算法虽然简单，但特别不适用于 $n$ 特别大的查找表。

虽然顺序查找的效率不高，但在下列两种情况下只能采用顺序查找：首先如果顺序表为无序表，那么只能用顺序查找；其次采用链式存储结构的线性表，则只能采用顺序查找。

3）二分查找

二分查找又叫折半查找，其查找效率比顺序查找算法效率高，但是它仅适用于顺序表，并且要求顺序表中各记录要按照其关键字进行升序排序或降序排序，属于有序查找算法。

基本思想：首先将查找范围为整个有序顺序表中的中间记录的关键字与待查找值进行比较，若相等，则查找成功，并返回该记录在顺序表中的位置；若不相等，则以中间记录为分界点，将查找表分成左右两个查找区域；若中间记录的关键字大于待查找值，则将查找范围缩小到左查找区域中；若中间记录的关键字小于待查找值，则将查找范围缩小到右查找区域中；然后再在左查找区域或者右查找区域中重复上述操作，直到找到关键字等于待查找值的记录（查找成功）为止或者查找区域不存在（查找失败）为止。

为实现二分查找算法，需要借助以下三个变量。

（1）变量 low，表示当前待查找区域的开始位置，初始值为 0。

（2）变量 high，表示当前待查找区域的结束位置，初始值为 $n-1$，$n$ 为待查记录的总个数。

(3) 变量 mid,表示当前待查找区域中中间记录关键字的位置。为避免发生整型溢出问题,mid=low+(high−low)/2。

**【例 5.10】** 有序表 $r$ 中 11 条记录的关键字序列为{10,20,30,40,45,60,70,75,80,90,100},使用二分查找的方法查找数据元素 75 和 65,给出分析过程。

当待查找值分别为 75 和 65 时进行二分查找的分析过程如下。

(1) 待查找值为 75 时的二分查找过程,如图 5.53 所示。

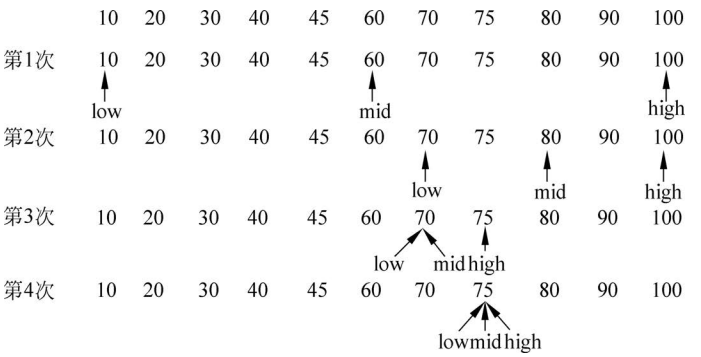



图 5.53 查找 75 的过程(4 次比较后查找成功)

① 初始时 low=0,high=10,low≤high,故待查找区域不为空且待查找区域关键字序列为{10 20 30 40 45 60 70 75 80 90 100},此时求得 mid=5,$r$[mid].key=60,此时发生第 1 次比较且此时 $r$[mid].key<searchValue,故查找范围缩小到 60 的后半段区域,修改 low=mid+1=6。

② low=6,high=10,low≤high,故待查找区域不为空且待查找区域关键字序列为{70 75 80 90 100},此时求得 mid=8,$r$[mid].key=80,此时发生第 2 次比较且此时 $r$[mid].key>searchValue,故查找范围缩小到关键字 80 的前半段区域,修改 high=mid−1=7。

③ low=6,high=7,low≤high,故待查找区域不为空且待查找区域关键字序列为{70 75},此时求得 mid=6,$r$[mid].key=70,此时发生第 3 次比较且此时 $r$[mid].key<searchValue,故查找范围缩小到 70 的后半段区域,修改 low=mid+1=7。

④ low=7,high=7,low≤high,故待查找区域不为空且待查找区域关键字序列为{75},此时求得 mid=7,$r$[mid].key=75,此时发生第 4 次比较且此时 $r$[mid].key==searchValue,查找结束。

因此查找成功,查找次数为 4,该记录在查找表中的下标为 7。

(2) 待查找值为 65 时的二分查找过程,如图 5.54 所示。

① 初始时 low=0,high=10,low≤high,故待查找区域不为空且待查找区域关键字序列为{10 20 30 40 45 60 70 75 80 90 100},此时求得 mid=5,$r$[mid].key=60,此时发生第 1 次比较且此时 $r$[mid].key<searchValue,故查找范围缩小到 60 的后半段区域,修改 low=mid+1=6。

② low=6,high=10,low≤high,故待查找区域不为空且待查找区域关键字序列为{70 75 80 90 100},此时求得 mid=8,$r$[mid].key=80,此时发生第 2 次比较且此时 $r$[mid].key>searchValue,故查找范围缩小到关键字 80 的前半段区域,修改 high=mid−1=7。

③ low=6,high=7,low≤high,故待查找区域不为空且待查找区域关键字序列为

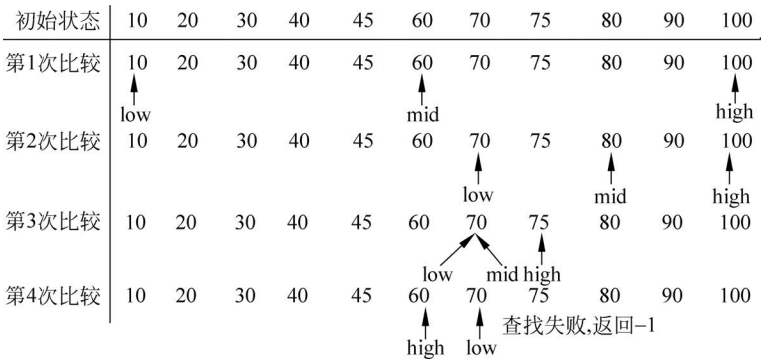



图 5.54　查找 65 的过程

{70 75},此时求得 mid＝6,r[mid].key＝70,此时发生第 3 次比较且此时 r[mid].key＞searchValue,故查找范围缩小到关键字 70 的前半段区域,修改 high＝mid－1＝5。

④ low＝6,high＝5,low＞high,故待查找区域不存在,关键字与待查找值没有发生比较,查找结束。

查找失败,比较次数为 3。

**2. 排序**

1) 排序的定义

排序是计算机程序设计中的一种重要操作,其功能是对一个数据元素集合或序列重新排列成一个按数据元素某个项值有序的序列。作为排序依据的数据项称为“排序码”,也即数据元素的关键字。为了便于查找,通常希望计算机中的数据表是按关键字有序的。如有序表的折半查找,查找效率较高。还有,二叉排序树、B－树和 B＋树的构造过程就是一个排序过程。若关键字是主关键字,则对于任意待排序序列,经排序后得到的结果是唯一的。

若对任意的数据元素序列,使用某个排序方法对它按关键字进行排序,若相同关键字元素间的位置关系在排序前与排序后保持一致,称此排序方法是稳定的,而不能保持一致的排序方法则称为不稳定的。

2) 排序的分类

排序涉及将数据元素按照某种特定的顺序进行排列。根据排序过程中数据是否全部存入内存,可以分为内排序和外排序两大类。

(1) 内排序。

内排序是指待排序的数据能够完整地放入计算机内存中进行排序的过程。由于内存访问速度远快于外部存储设备,内排序通常具有更高的效率。内排序适用于数据量相对较小,能够完全加载到内存中的情况。在进行内排序时,可以利用各种高效的排序算法,如冒泡排序、选择排序、快速排序、归并排序、堆排序等,来快速完成排序任务。本章主要介绍冒泡排序与选择排序,其他的内排序算法会在数据结构课程中详细讲解。

① 冒泡排序。

冒泡排序是从序列中的第一个元素开始,依次对相邻的两个元素进行比较,如果前一个元素大于后一个元素则交换它们的位置;如果前一个元素小于或等于后一个元素,则不交换它们。这一比较和交换的操作一直持续到最后一个还未排好序的元素为止。

冒泡排序的原理是通过不断交换相邻的两个元素来达到排序的目的。具体步骤如下。

在常数为 $n$ 的序列{ $a[0],a[1],a[2],\cdots,a[n-1]$}中，从 $a[0]$起，依次比较相邻的两个数，若邻接元素不符合次序要求，则对它们进行交换。本次操作后，数组中的最大元素被排到数组第 $n-1$ 的位置。

在剩下未排序的 $n-1$ 个数{ $a[0],a[1],a[2],\cdots,a[n-2]$}中，从 $a[0]$起，依次比较相邻的两个数，若邻接元素不符合次序要求，则对它们进行交换。本次操作后，$a[0]$～$a[n-2]$中的最大元素排到数组第 $n-2$ 的位置。

在剩下未排序的 $n-k$ 个数{$a[0],a[1],a[2],\cdots,a[n-i]$}中，从 $a[0]$起，依次比较相邻的两个数，若邻接元素不符合次序要求，则对它们进行交换。本次操作后，$a[0]$～$a[n-i]$中的最大元素排到数组第 $n-i$ 的位置。

在剩下未排序的两个数{ $a[0],a[1]$}中，比较这两个数，若不符合次序要求，则对它们进行交换。本次操作后，$a[0]$～$a[1]$中的最大元素排到数组第 1 的位置。

例如，假设排序表的关键字序列为{8,3,9,10,1,4}，按照冒泡排序算法进行排序。整个冒泡排序的过程如图 5.55 所示。

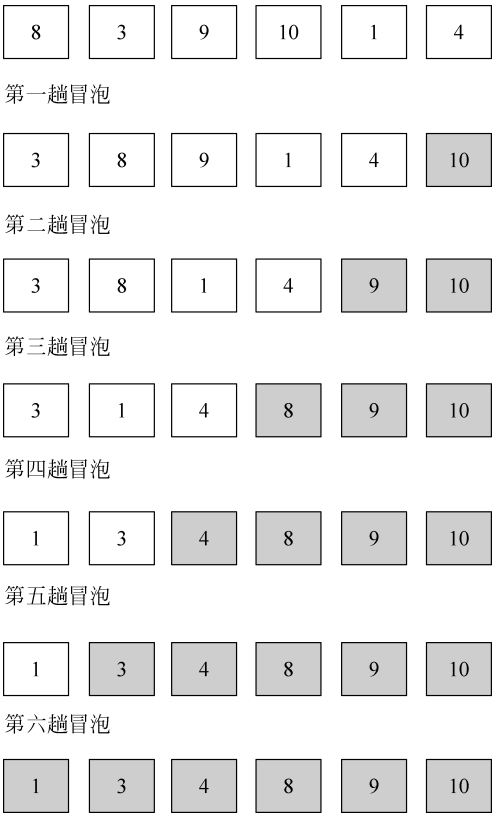



图 5.55　冒泡排序过程

冒泡排序是通过把小的元素往前调或者把大的元素往后调。比较是相邻的两个元素比较，交换也发生在这两个元素之间。若两个元素相等，则不会发生交换，所以相同元素的前后顺序并没有改变，因此冒泡排序是一种稳定排序算法。冒泡排序的时间复杂度较高，当数据量较大时，排序效率较低。在数据完全逆序的情况下，性能最差。冒泡排序虽然是一种稳

定的排序算法，但在实际应用中由于其效率问题，往往被其他更高效的排序算法所替代。

② 选择排序。

选择排序是直接从待排序序列中按规则找出最小或最大的元素放到有序序列中，直到待排序序列中的元素被选择完为止，从而达到排序的目的。

选择排序算法的原理是通过每次从未排序部分中选择最小或最大的元素，将其与未排序部分的第一个元素交换位置。这样一轮选择下来，最小或最大的元素就被正确地放到了已排序部分的末尾。通过多次选择，直到整个数列变得有序为止。具体步骤如下。

- 在常数为 $n$ 的序列{$a[0],a[1],a[2],\cdots,a[n-1]$}中选取最小值，与 arr[0]交换。
- 从序列{$a[1],a[2],\cdots,a[n-1]$}中选取最小值，与 arr[1]交换。
- 从序列{$a[2],\cdots,a[n-1]$}中选取最小值，与 arr[2]交换。
- 第 $i$ 次从序列{$a[i-1],\cdots,a[n-1]$}中选取最小值，与 arr[i−1]交换，以此类推。
- 总共通过比较 $n-1$ 次，最终得到一个有序序列。

例如，假设排序表的关键字序列为{25,4,3,18,36,2,22}，按照选择排序算法进行排序，整个选择排序的全过程如图 5.56 所示。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 初始序列 | 25 | 4 | 3 | 18 | 36 | 2 | 22 |
| 第一趟 | 2 | 4 | 3 | 18 | 36 | 25 | 22 |
| 第二趟 | 2 | 3 | 4 | 18 | 36 | 25 | 22 |
| 第三趟 | 2 | 3 | 4 | 18 | 36 | 25 | 22 |
| 第四趟 | 2 | 3 | 4 | 18 | 36 | 25 | 22 |
| 第五趟 | 2 | 3 | 4 | 18 | 22 | 25 | 36 |
| 第六趟 | 2 | 3 | 4 | 18 | 22 | 25 | 36 |
| 有序序列 | 2 | 3 | 4 | 18 | 22 | 25 | 36 |

图 5.56　选择排序过程

选择排序算法简单易懂，实现起来较为容易。适用于小规模数据的排序问题；在数据已经部分有序的情况下，性能不会受到太大影响；与冒泡排序相比，选择排序的交换次数较少。但是选择排序的时间复杂度较高，当数据量较大时，排序效率较低。在数据完全逆序的情况下，性能同样较差。选择排序是一种不稳定的排序算法，即在排序过程中可能会改变相等元素之间的相对位置，与更高效的排序算法相比，选择排序在实际应用中的使用较少。

(2) 外排序。

与内排序相对应的是外排序，主要用于处理数据量非常大、无法一次性加载到内存中的情况。在这种情况下，需要将数据分成多个小块，分别对这些小块进行排序，并将排序结果存储在外部存储设备(如硬盘)上。然后，通过多路归并等技术将这些已排序的小块合并成一个完整的有序结果。由于外部存储设备的访问速度相对较慢，因此外排序通常比内排序更加复杂和耗时。在实际应用中，通常需要综合考虑数据规模、内存大小和排序要求等因素来选择合适的排序方法。

## 5.8　数据库设计基础

数据库系统是为适应数据处理的需要而发展起来的一种较为理想的数据处理系统，也是一个为实际可运行的应用系统提供数据的软件系统，是存储介质、处理对象和管理系统的集合体。

### 5.8.1 数据库概述

**1. 数据库的基本概念**

数据库(Database)是按照数据结构来组织、存储和管理数据的集合。1963 年,美国 Honeywell 公司揭开了数据库技术的序幕;随着信息技术和市场的发展,特别是 20 世纪 90 年代以后,数据管理不再仅仅是存储和管理数据,而转变成用户所需要的各种数据管理的方式。数据库有很多种类型,从最简单的存储各种数据的表格,到能够进行海量数据存储的大型数据库系统,都在各个方面得到了广泛的应用。

在信息化社会,充分有效地管理和利用各类信息资源,是进行科学研究和决策管理的前提条件。数据库技术是管理信息系统、办公自动化系统、决策支持系统等各类信息系统的核心部分,是进行科学研究和决策管理的重要技术手段。数据库技术涉及许多基本概念,主要包括信息、数据、数据处理、数据库、数据库管理系统以及数据库系统等。

1) 信息

信息是现实世界中各种事物的存在方式、运动形态以及它们之间的相互联系等诸要素在人脑中的反映,是通过人脑抽象后形成的概念。

2) 数据

数据是反映客观事物属性的记录,是信息的具体表现形式,数据经过加工后就成为信息。在计算机系统中,字母、数字、语音、图形、图像等都是数据。

3) 数据处理

数据处理(Data Processing)是对数据的采集、存储、检索、加工、变换和传输。根据处理设备的结构和工作方式,以及数据的时间、空间分布方式不同,数据有不同的处理方式。不同的处理方式要求不同的硬件和软件支持,且有各自的特点,具体应用时应根据问题的实际环境选择合适的处理方式。

4) 数据库

数据库(Database)是指以一定方式存储在一起、能为多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

5) 数据库管理系统

数据库管理系统(Database Management System,DBMS),是一种针对对象数据库,为管理数据库而设计的大型计算机软件管理系统。具有代表性的数据库管理系统有 Oracle、Microsoft SQL Server、Access、MySQL 及 PostgreSQL 等。通常数据库管理员会使用数据库管理系统来创建和维护数据库系统。

6) 数据库系统

数据库系统(Database System,DBS),通常由软件、数据库和数据管理员组成。软件主要包括操作系统、各种宿主语言、实用程序以及数据库管理系统。数据库管理系统统一管理数据库,数据的插入、修改和检索均要通过数据库管理系统进行。数据管理员负责创建、监控和维护整个数据库,使数据能被任何有权使用的用户有效使用。数据库管理员一般是由业务水平较高、资历较深的人员担任。

**2. 数据管理技术的发展**

数据管理技术是对数据进行分类、组织、编码、输入、输出、存储、检索和维护的技术。数

据管理技术的发展大致划分为以下几个阶段：人工管理阶段、文件系统阶段、数据库系统阶段和高级数据库阶段。

1）人工管理阶段

20 世纪 50 年代以前，计算机主要用于数值计算。当时的计算机软硬件都不完善，外存只有纸带、卡片和磁带，没有直接存取设备；软件方面，没有操作系统以及管理数据的软件，程序员在程序中不仅要规定数据的逻辑结构，还要设计其物理结构，包括存储结构、存取方法、输入/输出等；数据方面，数据量小，数据无结构，由用户直接管理，且数据间缺乏逻辑组织，数据依赖于特定的应用程序，缺乏独立性。当数据的物理组织或存储设备改变时，用户程序就必须重新编制。

2）文件系统阶段

20 世纪 50 年代后期到 20 世纪 60 年代中期，出现了磁鼓、磁盘等数据存储设备，新的数据处理系统迅速发展起来。这种数据处理系统是把计算机中的数据组织成相互独立的数据文件，系统可以按照文件的名称对其进行访问，对文件中的记录进行存取，并可以实现对文件的修改、插入和删除，这就是文件系统。文件系统实现了记录内的结构化，即给出了记录内各种数据间的关系。但是，文件从整体来看却是无结构的，其数据面向特定的应用程序，因此数据共享性、独立性差，且冗余度大，管理和维护的代价也很大。

3）数据库系统阶段

20 世纪 60 年代后期，随着计算机在数据管理领域的广泛应用，出现了数据库这样的数据管理技术。数据库的特点是数据不再只针对某一特定应用，而是面向全组织，具有整体的结构性，共享性高、冗余度小、具有一定的程序与数据间的独立性，并且实现了对数据的统一控制。

4）高级数据库阶段

计算机硬件、数据和数据库应用，这三者推动着数据库技术与系统的发展。计算机硬件平台的发展仍然实践着摩尔定律，数据的复杂度和数据量都在迅速增长，数据库应用迅速向深度、广度扩展；尤其是互联网的出现，极大地改变了数据库的应用环境，向数据库领域提出了前所未有的技术挑战。这些因素的变化推动着数据库技术的进步，出现了一批新的数据库技术，如 Web 数据库技术、并行数据库技术、数据仓库与联机分析技术、数据挖掘与商务智能技术、内容管理技术、海量数据管理技术等。

**3. 数据模型**

数据模型(Data Model)是现实世界数据特征的抽象和数学形式框架，也是数据库系统中用以提供信息表示和操作手段的形式构架。数据模型的三要素分别是数据结构、数据操作和完整性约束条件。

(1) 数据结构：是所研究的对象类型的集合，是对象和对象间数据的类型、内容、性质以及数据间的联系的表达和实现，是系统静态特征的描述。数据结构是数据模型的基础，数据操作和数据约束都基本建立在数据结构之上。

(2) 数据操作：数据库中对象的实例允许执行的操作集合，主要包括检索和更新(插入、删除、修改)两类操作。数据模型必须定义这些操作的确切含义、操作符合、操作规则以及实现操作的语言。数据操作是对系统动态特征的描述。

(3) 完整性约束条件：是一组完整性规则的集合，主要用于描述数据结构内数据间的

语法、语义联系，数据间的制约和依存关系，以及数据动态变化的规则，以保证数据的正确、有效和相容。

**4. 数据模型分类**

目前，应用在数据库系统中的数据模型主要有层次模型、网状模型和关系模型，它们之间的根本区别在于数据之间联系的表示方式不同（即记录之间的联系方式不同）。层次模型、网状模型和关系模型分别以“树状结构”“图结构”和“二维表（关系）”表示数据之间的联系。

1）层次模型

层次模型是数据库系统最早使用的一种模型，它的数据结构是一棵“有向树”，根结点在最上端，层次最高，子结点在下，逐层排列。层次模型中的数据是一对多关系结构，采用关键字来访问每一层的每一部分。

最具影响的层次模型的 DBS 是 20 世纪 60 年代末 IBM 公司推出的 IMS 层次模型数据库系统。

2）网状模型

网状模型以网状结构（图结构）表示实体与实体之间的联系，网中的每个结点表示一个记录，联系用连接指令或指针来表示。网状模型可以表示多个从属关系的联系，也可以表示数据间的交叉关系，即数据间的横向关系与纵向关系，是层次模型的扩展。

3）关系模型

关系模型是以二维表结构（也称关系）来表示实体与实体之间的联系，以关系代数理论为基础。在关系模型中，数据结构是一个二维表框架组成的集合，操作的对象和结果也都是二维表。关系模式是目前最流行的数据库模型。

## 5.8.2 关系数据库

**1. 关系数据库的基本概念**

关系数据库，是建立在关系数据模型基础上的数据库，它借助关系代数的概念和方法来处理数据。关系数据模型形式上被组织成一张二维表格，表格中的数据能以不同的方式进行存取。每个表格（也称为一个关系）都包含列和行，列表示数据种类，行表示数据实体。创建一个关系本质上就是定义数据列的数据类型、取值范围和数据约束等。

目前主流的关系数据库有 Oracle、DB2、SQL Server、Sybase、MySQL 等。

**2. 关系数据库的规范化***

关系数据库是当前应用最广的数据库，关系数据库设计的核心问题是关系模型的设计。对于规模较小的数据库，可以比较轻松地设计数据库中的表结构；然而，对于关系结构复杂的大型数据库，其关系模型表结构也更为庞杂，如果表结构定义不合理，就可能产生数据冗余、数据不一致和不完整。因此，有必要对数据库表结构进行规范化，以减少数据冗余，提高数据库的存储效率、数据完整性和可扩展性。

关系规范化的基本思想就是消除关系中的数据冗余和不合适的数据依赖，解决数据插入、更新和删除时发生的异常现象。

构造数据库必须遵循一定的规则，在关系数据库中，这种规则就称为范式。范式的概念最早由 E. F. Codd 提出，从 1971 年起，Codd 相继提出了关系的三级规范化形式，即第一范

式(1NF)、第二范式(2NF)、第三范式(3NF)；1974 年，Codd 和 Boyce 共同提出了一个新的范式，即 Boyce-Codd 范式，简称 BC 范式；1976 年，Fagin 提出了第四范式，后来又有人提出了第五范式。

1) 第一范式(1NF)

如果关系模式 $R$，其所有的属性均为简单属性，即每个属性都是不可再分的，则称 $R$ 属于第一范式，记作 $R \in 1NF$。1NF 是关系中最基本的规范形式。

例如，表 5.7 中属性值可再分，表 5.8 中属性可再分，它们都不满足 1NF。

**表 5.7　学生信息表(1)**

| 姓　　名 | 联 系 电 话 | | 年　　龄 |
|---|---|---|---|
| 李　丽 | 13612345678 | | 20 |
| 李小明 | 13988776655 | 010-1234567 | 21 |

**表 5.8　学生信息表(2)**

| 姓　　名 | 联 系 电 话 | | 年　　龄 |
|---|---|---|---|
| | 手　　机 | 座　　机 | |
| 李　丽 | 13612345678 | 020-9876543 | 20 |
| 李小明 | 13988776655 | 010-1234567 | 21 |

每个规范化的关系都属于 1NF；对于不满足 1NF 的关系，只需去除属性的组合项就能将其转换为属于 1NF 的关系。不满足 1NF 的数据库，不是关系数据库。

2) 第二范式(2NF)

如果关系模式 $R \in 1NF$，且每一个非主键属性都完全函数依赖于 $R$ 的主键，则称 $R$ 属于第二范式，记作 $R \in 2NF$。假设有一教务管理实例，要求学生上课指定一个老师、一本教材、一个教室、一个时间，则如表 5.9 所示的学生上课关系模式就不满足 2NF。

**表 5.9　学生上课表(1)**

| 学生 | 课程名称 | 教师 | 教师职称 | 教材名称 | 教室 | 上课时间 |
|---|---|---|---|---|---|---|
| 李小丽 | 编程基础 | 陈明 | 副教授 | 《C 语言程序设计》 | 101 | 14：30 |

具体分析如下。

一个学生上一门课，一定是特定某个老师上课，即“教师”完全函数依赖于(学生，课程)。

一个学生上一门课，教师特定了，则教师的职称也确定了，即“教师职称”完全函数依赖于(学生，课程)。

一个学生上一门课，一定在特定某个教室，即“教室”完全函数依赖于(学生，课程)。

一个学生上一门课，一定在特定时间，即“上课时间”完全函数依赖于(学生，课程)。

一门课程，一定指定了某个教材，即“教材”完全函数依赖于(课程)。

因此，表中的非主键属性(教师、教师职称、教材、教室、上课时间)不是完全函数依赖于主键(学生，课程)，只是其中的部分(教师、教师职称、教室、上课时间)函数依赖于主键(学生，课程)，故不满足 2NF。

不满足 2NF 的关系模式，在使用中将产生插入异常、删除异常和修改异常，分别示例如下。

(1) 假设要在教学计划中新增加一门课程“微积分”,教材是《微积分学》,但由于学生还没选课,而学生又是主属性不能为空,故在插入课程和教材数据时将产生错误,即产生插入异常。

(2) 假设删除表 5.9 中的记录,则记录中的“教材名称”(《C 语言程序设计》)也会被删除。此时,如果想要查看“编程基础”这门课程所使用的教材,将产生错误,即产生删除异常。

(3) 假如编程基础的教材进行了更换,更换为《C++程序设计》,若有 10 000 名学生选了该课程,则这 10 000 条记录都需进行修改,修改工作量巨大,产生修改异常。

为满足 2NF 要求,可通过关系模式的投影分解进行转换。投影分解的基本原则就是“一事一地”,即让一个关系只描述一个实体或者实体间的联系,如果多于一个实体或联系,则进行投影分解。因此,如表 5.9 所示的关系模式可以投影分解为两个关系模式,且投影分解后的关系模式都满足 2NF,如表 5.10 和表 5.11 所示。

**表 5.10　学生上课表(2)**

| 学生 | 课程名称 | 教师 | 教师职称 | 教室 | 上课时间 |
|---|---|---|---|---|---|
| 李小丽 | 编程基础 | 陈明 | 副教授 | 101 | 14：30 |

**表 5.11　课程教材表**

| 课程名称 | 教材名称 |
|---|---|
| 编程基础 | 《C 语言程序设计》 |

3) 第三范式(3NF)

如果关系模式 $R \in 2NF$,且每个非主属性都不传递依赖于 $R$ 的主键,则称 $R$ 属于第三范式,记作 $R \in 3NF$。投影分解后的如表 5.10 所示学生上课表虽然满足 2NF,但并不满足 3NF,原因如下:“教师”完全函数依赖于(学生,课程),“教师职称”完全函数依赖于“教师”。因此,“教师职称”也完全函数依赖于(学生,课程),但这种函数依赖不是直接函数依赖,而是传递函数依赖。

有传递依赖的关系模式,在使用时同样存在插入异常、修改异常和删除异常,示例分别如下。

(1) 新来一名教师,但还未分配教学课程,插入职称时,产生插入异常。

(2) 教师的职称发生了改变,若关于该教师的记录有 $N$ 条,则需修改 $N$ 次,产生修改异常。

(3) 删除一名教师的上课记录,教师的职称同时被删除,产生删除异常。

因此,如表 5.10 所示的关系模式可投影分解为两个关系模式,且投影分解后的关系模式都满足 3NF,如表 5.12 和表 5.13 所示。

**表 5.12　学生上课表(3)**

| 学生 | 课程名称 | 教师 | 教室 | 上课时间 |
|---|---|---|---|---|
| 李小丽 | 编程基础 | 陈明 | 101 | 14：30 |

**表 5.13　教师职称表**

| 教师 | 教师职称 |
|---|---|
| 陈明 | 副教授 |

4) BC 范式(BCNF)

如果关系模式 $R \in 1NF$,且每个属性都不部分依赖于 $R$ 的主键也不传递依赖于 $R$ 的主键,则称 $R$ 属于 BC 范式,记作 $R \in BCNF$。

相对于 3NF,BCNF 的要求更加严格,3NF 只是要求 $R$ 为 2NF 且非主键属性不传递依赖于 $R$ 的主键,而 BCNF 则要求 $R$ 的每个属性都达到要求。因此,若 $R$ 满足 BCNF,则一定满足 3NF;反之则不一定成立。

一般来说，一个关系模式规范到 3NF 或 BCNF 就能满足设计要求，在 BC 范式以上还有第四范式、第五范式，但在绝大多数应用中并不需要规范化到这种程度。在某些情况下，过多的范式化甚至会对数据库的逻辑可读性和使用效率造成阻碍，在数据库中存在一定程度的冗余并不一定都是坏事情。

**3. 关系数据库的设计步骤**

数据库设计包括以下 6 个主要步骤。

(1) 需求分析：了解用户的数据需求、处理需求、安全性及完整性要求。

(2) 概念结构设计：通过数据抽象，设计系统概念模型，一般为 E-R 模型。

(3) 逻辑结构设计：设计系统的模式和外模式，对于关系模型主要是基本表和视图。

(4) 物理结构设计：设计数据的存储结构和存取方法，如索引的设计。

(5) 系统实施：组织数据入库、编制应用程序、试运行。

(6) 运行维护：系统投入运行，长期的维护工作。

## 5.8.3 结构化查询语言

**1. SQL 概述**

结构化查询语言(Structured Query Language，SQL)，是一种有特殊目的的编程语言，也是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统。

结构化查询语言是高级的非过程化编程语言，允许用户在高层数据结构上工作。它不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以对于不同底层结构的数据库系统，可以使用相同的结构化查询语言作为数据输入与管理的接口。结构化查询语言语句可以嵌套，这使它具有极大的灵活性和强大的功能。

结构化查询语言包含 6 部分：数据查询语言(DQL)、数据操纵语言(DML)、事务处理语言(TPL)、数据控制语言(DCL)、数据定义语言(DDL)、指针控制语言(CCL)。

**1. 数据操纵语言**

数据操纵语言(Data Manipulation Language，DML)的语句包括动词 INSERT、UPDATE 和 DELETE，分别用于添加、修改和删除表中的行。

**2. 数据控制语言**

数据控制语言(Data Control Language，DCL)是用来设置或者更改数据库用户或角色权限的语句，这些语句包括 GRANT、DENY、REVOKE 等语句，在默认状态下，只有 sysadmin、dbcreator、db_owner 或 db_securityadmin 等角色的成员才有权利执行数据控制语言。

**3. 数据定义语言**

数据定义语言(Data Definition Language，DDL)的语句包括动词 CREATE 和 DROP。例如，在数据库中创建新表(CREATE TABLE)或删除表(DROP TABLE)等。

## 5.8.4 常用数据库系统

**1. Oracle**

Oracle 数据库是甲骨文公司开发的一款关系数据库管理系统，是一种大型数据库系统，在数据库领域一直处于领先地位。Oracle 数据库系统是目前世界上最流行的关系数据

库管理系统，系统效率高、可靠性好、使用方便、功能强大，适用于各类大、中、小、微型计算机环境，能适应高吞吐量的数据库解决方案。主要应用于商业和政府部门。

**2. DB2**

DB2 数据库是美国 IBM 公司开发的一套关系型数据库管理系统，主要应用于大型应用系统，具有较好的可伸缩性，可支持从大型计算机到单用户环境，能应用于所有常见的服务器操作系统平台。DB2 提供了高层次的数据利用性、完整性、安全性、可恢复性，以及小规模到大规模应用程序的执行能力，具有与平台无关的基本功能和 SQL 命令。

DB2 以拥有一个非常完备的查询优化器而著称，其外部连接改善了查询性能，并支持多任务并行查询。同时，具有很好的网络支持能力，每个子系统可以连接十几万个分布式用户，可同时激活上千个活动线程，对大型分布式应用系统尤为适用。

**3. SQL Server**

SQL Server 是 Microsoft 公司推出的关系型数据库管理系统，它最初是由 Microsoft、Sybase 和 Ashton-Tate 三家公司共同开发，于 1988 年推出第一个 OS/2 版本，随后相继推出 SQL Server 7.0、SQL Server 2000、SQL Server 2005、SQL Server 2008、SQL Server 2012、SQL Server 2016、SQL Server 2019 等。

下面以 SQL Server 2012 为例，简单介绍数据库的创建、分离和附加，以及数据表的创建，数据的插入和更新等操作。

1）启动 SQL Server Management Studio

操作步骤：

（1）单击“开始”→“所有程序”→Microsoft SQL Server 2012→SQL Server Management Studio，如图 5.57 所示。


图 5.57　SQL Server Management Studio 登录

（2）在“连接到服务器”操作界面中需要指定注册服务器类型、服务器名称、身份验证类型。

在如图 5.57 所示的“连接到服务器”操作界面中设置好连接服务器的类型、名称、身份证后，单击“连接”按钮进入 Microsoft SQL Server 2012 工作界面。Microsoft SQL Server 2012 工作界面是一个标准的 Windows 界面，由标题栏、菜单栏、工具条、树窗口组成，如图 5.58 所示。

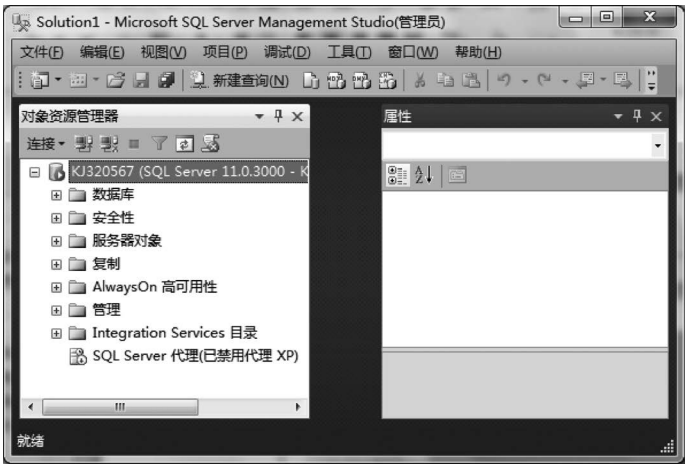



图 5.58　SQL Server Management Studio 的工作界面

2）SQL Server 内置系统数据库

启动 SQL Server Management Studio 连接数据库引擎后，展开“数据库”→“系统数据库”文件夹，可以看到 master、model、msdb、tempdb 4 个内置系统数据库。

master 数据库记录 SQL Server 系统的所有系统级别信息，如图 5.59 所示，它记录所有的登录账户和系统配置设置。master 还记录了所有其他数据库的数据库文件的位置以及 SQL Server 的初始化信息。如果 master 数据库不可以用，则 SQL Server 无法启动。

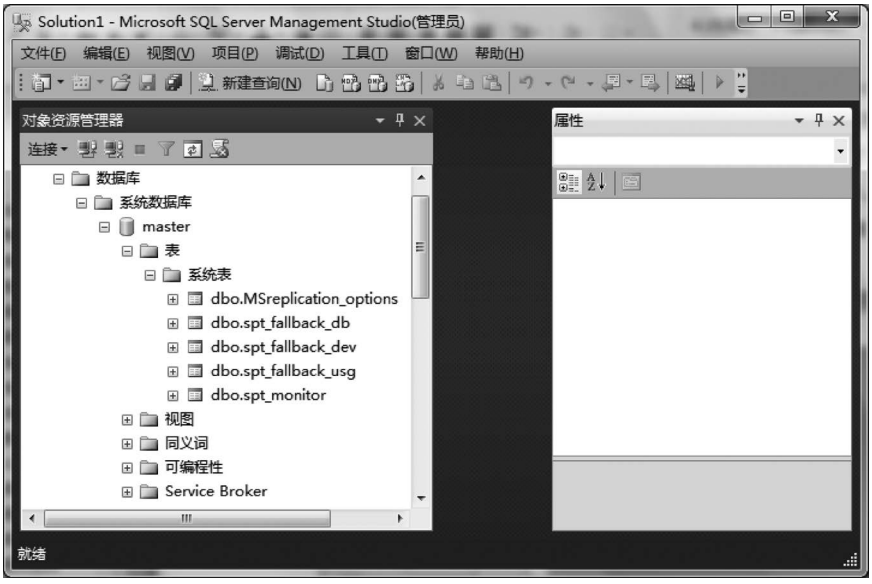



图 5.59　系统数据库 master

model 数据库是模板数据库。当发出 CREATE DATABASE 语句时，新数据库部分通过复制 model 数据库中的内容进行创建，剩余部分由空白页填充。由于 SQL Server 每次启动时都是要创建 tempdb 数据库，故 model 数据库必须一直存在于 SQL Server 系统中。

msdb 数据库是代理服务数据库，给 SQL Server 代理提供必要的信息来运行作业，为其报警、任务调度和记录操作员的操作提供存储空间，是 SQL Server 中重要的数据库之一。

tempdb 是一个临时数据库，为所有的临时表、临时存储过程及其他临时操作提供存储空间。tempdb 数据库由整个系统的所有数据库使用，不管用户使用哪个数据库，它们所建立的所有临时表和存储过程都存储在 tempdb 上。SQL Server 每次启动时，tempdb 数据库被重新建立。当用户与 SQL Server 断开连接时，其临时表和存储过程自动被删除，并且在系统关闭后没有活动连接，因此 tempd 中不会有什么内容从一个 SQL Server 会话保存到另一个会话。由于临时存放中间结果，因此无法备份和恢复 tempdb 数据库。

3）新建数据库

SQL Server 数据库是由数据文件和事务日志文件组成，一个数据库至少应包含一个数据文件(.MDF)和一个事务日志文件(.LDF)。数据文件包含数据和对象，例如，表、索引、存储过程和视图等，日志文件包含恢复数据库中所有事务所需的信息。为了便于分配和管理，可以将数据文件集合起来，放入文件组中。创建用户数据库(如 xscjgl_DB)的操作步骤如下。

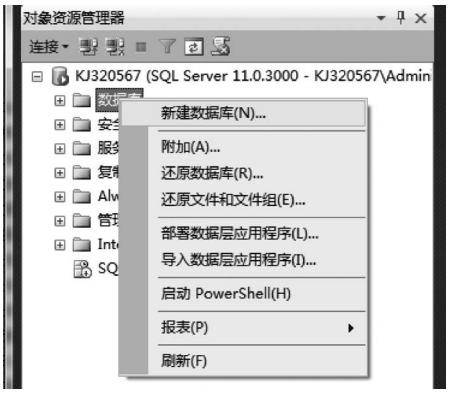



图 5.60　新建数据库

(1) 选中将要使用的数据库服务器，用鼠标右键单击数据库，在弹出的快捷菜单中选择“新建数据库”，如图 5.60 所示。

(2) 打开“新建数据库”窗口的“常规”选择页，在“数据库名称”文本框中输入数据库的名称(如 xscjgl_DB)，设置数据文件和日志文件的名称、位置、文件大小和增长方式等信息，单击“添加”按钮还可添加次要数据文件，如图 5.61 所示。

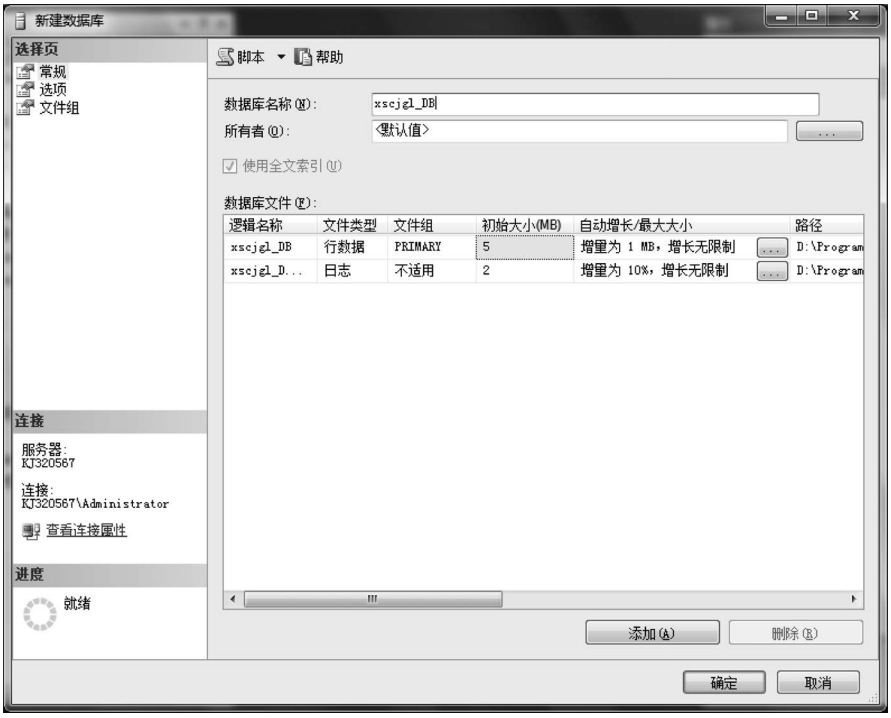



图 5.61　新建数据库设置

(3) 单击“确定”按钮,完成数据库的创建,如图 5.62 所示。

4) 分离数据库

分离数据库就是将某个数据库从 SQL Server 数据库列表中删除,使其不再被 SQL Server 管理和使用,但该数据库的数据文件(.MDF)和对应的日志文件(.LDF)完好无损。分离成功后,可以将该数据文件(.MDF)和对应的日志文件(.LDF)复制到其他磁盘中作为备份。分离数据库(如 xscjgl_DB)的操作步骤如下。

(1) 在对象资源管理器中展开“服务器”节点,在数据库对象下找到需要分离的数据库(如 xscjgl_DB),右键单击该数据库,在弹出的快捷菜单中选择“任务”→“分离”选项,如图 5.63 所示。


图 5.62 新建后的数据库

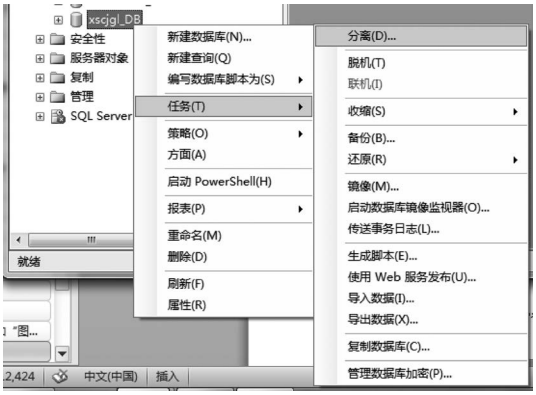



图 5.63 分离数据库操作

(2) 在弹出的“分离数据库”窗口中显示要分离的数据库,单击“确定”按钮,完成数据库的分离操作,如图 5.64 所示。

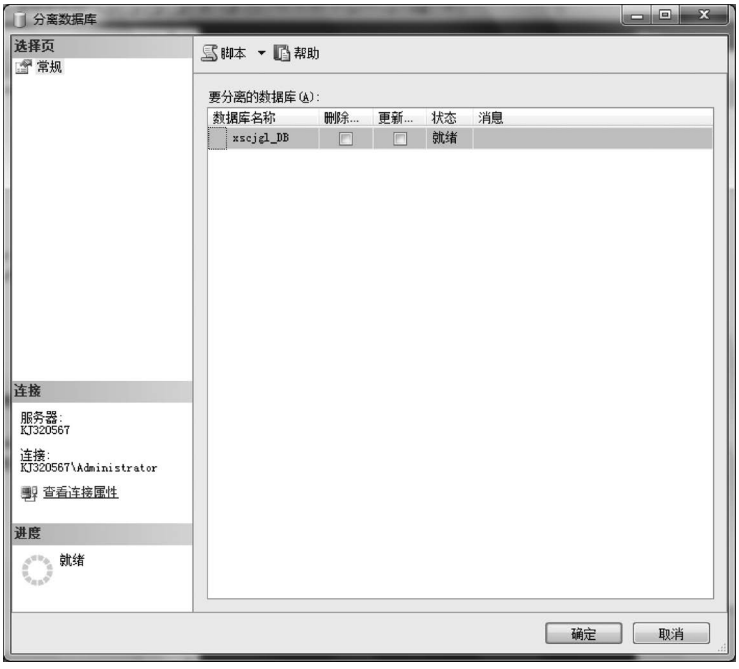



图 5.64 分离数据库窗口

5）附加数据库

附加数据库就是将计算机中的数据库文件(.MDF)和对应的日志文件(.LDF)添加到某个 SQL Server 数据库服务器中，并由该服务器来管理和使用数据库。附加用户数据库(如 xscjgl_DB)的操作步骤如下。

(1) 在对象资源管理器中展开“服务器”节点，右键单击“数据库”，在弹出的快捷菜单中选择“附加”选项，如图 5.65 所示。


图 5.65　附加数据库

(2) 在弹出的“附加数据库”窗口中单击“添加”按钮，如图 5.66 所示，然后在计算机中选择需要附加的数据库文件，单击“确定”按钮，即可完成数据库的附加操作，如图 5.67 所示。


图 5.66　“附加数据库”窗口

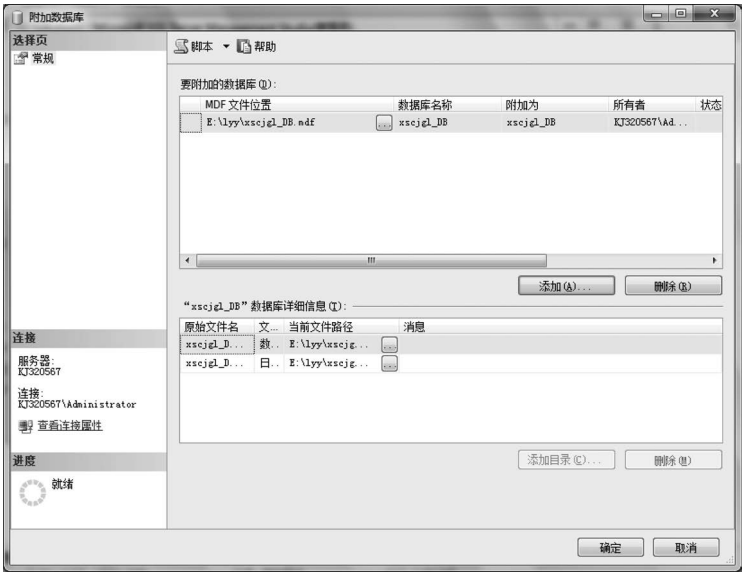



图 5.67　附加数据库操作

6）创建数据表

展开对象资源管理器，在数据库 xscjgl_DB 中创建学生信息表，表结构及约束条件如表 5.14 所示。

**表 5.14　学生信息表**

| 字段名称(列名) | 字段含义 | 数据类型及长度 | 允许为空 | 默认值 | 主键 |
|---|---|---|---|---|---|
| StudNo | 学号 | varchar(15) | | | Y |
| StudName | 姓名 | varchar(20) | | | |
| StudSex | 性别 | char(2) | | “男” | |
| StudBirthday | 出生日期 | datetime | Y | | |
| StudAddress | 家庭住址 | varchar(100) | Y | | |

在对象资源管理器中，创建数据表的操作步骤如下。

(1) 展开数据库 xscjgl_DB→“表”→单击鼠标右键选择“新建表”，如图 5.68 所示。

(2) 在打开的设计表操作界面中输入数据表的字段名称(列名)，并设置数据类型、字段长度及字段约束(主键、默认值、是否为空)。设计的“学生表”字段信息如图 5.69 所示。

(3) 数据表字段信息设计完毕后，单击“保存”按钮，并输入数据表名称，即可完成数据表的创建。展开对象资源管理器可查看创建后的数据表。创建完毕后的学生表信息如图 5.70 所示。


图 5.68　新建表


图 5.69　设计学生信息表


图 5.70　创建完毕的学生信息表

7）添加记录

数据表创建完毕后，利用对象资源管理器，就可完成数据的录入，即添加记录。添加记录时，在数据库中选中需要录入数据的数据表，然后右击选择“编辑前 200 行”，在弹出的表格编辑窗口中录入相关数据即可。学生信息表中需添加的数据如表 5.15 所示，数据添加完毕后的学生信息表如图 5.71 所示。

表 5.15　学生信息表中需添加的数据

| StudNo | StudName | StudSex | StudBirthday | StudAddress |
|---|---|---|---|---|
| A001 | 张三 | 男 | 1998-07-12 | 广东广州 |
| A002 | 李四 | 女 | 1999-11-09 | 福建厦门 |
| A003 | 王五 | 男 | 1998-01-24 | 北京昌平 |
| A004 | 赵六 | 男 | 1997-06-18 | 广西南宁 |


图 5.71　添加数据后的学生信息表

8）更新记录

在 SQL Server 2012 中，各类操作既可利用对象资源管理器来完成，也可使用 T-SQL 语句来完成。例如，在学生信息表中，使用 UPDATE 语句将学号为 A001 的学生姓名更新为“张小明”。操作步骤如下。

（1）右击数据库⊟ xscj_DB→单击“新建查询”→在查询窗口中输入以下代码。

```
UPDATE 学生信息表 SET StudName = '张小明'  WHERE StudNo = 'A001'
```

（2）单击“分析”按钮 ✓，检查语法无误后→单击 ! 执行(X) 按钮，在“消息”窗口中显示：1 行受影响，结果如图 5.72 所示。

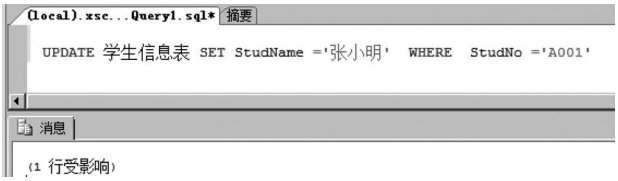



图 5.72　更新记录

9）删除记录

在学生信息表中，利用 DELETE 语句删除学号号为 A004 的记录，操作步骤如下。

（1）右击数据库→单击“新建查询”→在查询窗口中输入以下代码。

```
DELETE FROM 学生信息表 WHERE StudNo = 'A004'
```

（2）单击“分析”按钮 ✓，检查语法无误后→单击 ! 执行(X) 按钮，在“消息”窗口中显示：1 行受影响，结果如图 5.73 所示。

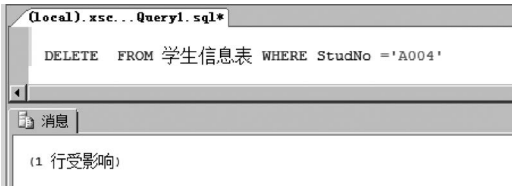



图 5.73　删除记录

10）数据查询

（1）查询所有记录。

在学生信息表中，查询所有学生的基本信息，操作步骤如下。

① 右击数据库 →单击“新建查询”→在查询窗口中输入以下代码。

```
SELECT * FROM 学生信息表
```

② 单击“分析”按钮 ✓，语法检查无误后→单击 ! 执行(X) 按钮，查询结果如图 5.74 所示。

（2）带条件查询。

在学生信息表中查询性别为“男”的学生信息。操作步骤如下。

① 右击数据库→单击“新建查询”→在查询窗口中输入以下代码。

```
SELECT * FROM 学生信息表 WHERE StudSex = '男'
```

② 单击“分析”按钮 ✓，检查语法无误后→单击 ! 执行(X) 按钮，查询结果如图 5.75 所示。


图 5.74　查询所有记录

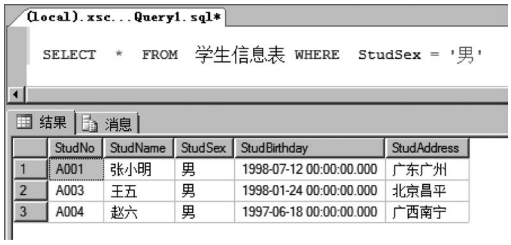



图 5.75　带条件查询结果

**4. MySQL**

MySQL 是瑞典 MySQL AB 公司开发的一个关系型数据库管理系统，目前属于 Oracle 旗下产品。MySQL 是流行的关系型数据库管理系统，在 Web 应用方面 MySQL 是最好的关系数据库管理系统应用软件之一。

MySQL 是一种关联数据库管理系统，它将数据保存在不同的表中，提高了数据的存取速度和灵活性。MySQL 所使用的 SQL 是用于访问数据库的最常用标准化语言。MySQL 软件采用双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

## 5.8.5　数据库技术的新发展

将数据库技术应用到特定的领域中，出现了工程数据库、统计数据库、空间数据库、并行数据库、多媒体数据库、主动数据库、移动数据库等多种数据库，使数据库领域中新的技术层出不穷。

**1. 数据库技术发展现状**

数据库技术与多学科技术的有机结合是当前数据库技术发展的重要特征。数据库技术的发展现状归纳为以下 4 方面。

1）面向对象方法和技术正逐步融入数据库

人们将面向对象的方法引入数据库领域，形成了面向对象数据库管理系统（OODBMS）。

它实际上是数据库技术(DB)和面向对象技术(OO)结合的产物。OODBMS 首先是一个数据库系统,即系统具备数据库系统的处理能力;其次又是一个面向对象的系统,即包含对象的概念、方法和技术。与传统的数据库相比,OODBMS 在复杂系统的模拟、表达和处理能力等方面具有优势,不足之处是理论技术还相当不成熟、不够完善。但随着数据库技术和面向对象技术的不断发展和完善,OODBMS 必将得到广泛应用。

2) 网络技术与数据库技术的融合

分布式数据库系统是数据库技术与计算机网络技术相结合的产物。分布式数据库就有局部数据库和全局数据库的概念,它具有以下优点:既能对数据进行全局管理,又能使各节点自主管理本节点数据;数据具有独立性且分布透明;增大了数据的容量;提高了数据的可靠性与可用度;改善了系统的性能和并行处理能力。当然也具有以下缺点:花在通信部分的系统开销较大;复杂的存取结构;数据的安全性和保密性较难处理。不过这些缺点正随着计算机其他技术的发展逐步得到解决。

3) 多媒体技术进入数据库领域

随着多媒体技术的发展,无论是 PC 还是在网络上都充斥着各种多媒体信息,如声音、图像、视频、超文本信息等,当这些信息增加时,就需要数据库来组织和管理这些信息。多媒体数据库是计算机技术、影像技术和通信技术相结合的产物,具有类型复杂、信息量大、实时性、分布性和交互性等特点。

4) 人工智能与数据库技术的结合

人工智能是研究计算机模拟人的大脑思维和模拟人的活动的一门科学,因此逻辑推理和判断是其最主要的特征,但对于信息检索则效率很低。数据库技术是数据处理方面的技术,对于数据的存储、管理、检索有其独特的优势,但对于逻辑推理却无能为力。造成这种局面的原因是,过去这两方面的研究视野均局限于本领域,人工智能只追求逻辑推理正确无误,不注意空间和时间的限制,因而研制的语言和专家系统效率低是必然的。而数据库开发者拼命争取时间和空间,但他们只考虑数据库实际存放的数据,而不考虑库中虽无但通过推理可得出的数据。智能数据库系统是人工智能与数据库技术相结合的产物。它具有两种技术的优点而避免了它们的缺点,是一种新型的数据库系统。

**2. 当前数据库系统存在的不足**

目前,商品化的数据库管理系统以关系型数据库为主导产品,技术比较成熟。面向对象的数据库管理系统虽然技术先进,数据库易于开发、维护,但尚未有成熟的产品。国际国内的主导关系型数据库管理系统有 Oracle、Sybase、Informix 和 Ingres。这些产品都支持多平台,如 UNIX、VMS、Windows,但支持的程度不一样。IBM 的 DB2 也是成熟的关系型数据库。但是,DB2 是内嵌于 IBM 的 AS/400 系列机中,只支持 OS/400 操作系统。

1) 关系数据库系统的缺点

(1) 数据类型表达能力差。从下一代应用软件的发展角度来看,关系数据库的根本缺陷在于缺乏直接构造与这些应用有关的信息的类型表达能力,缺乏这种能力将产生以下有害的影响。例如,大多数 RDBMS 产品所采用的简单类型在重构复杂数据的过程中将会出现性能问题;数据库设计过程中的额外复杂性;RDBMS 产品和编程语言在数据类型方面的不协调。大多数现代的 RDBMS 产品已成熟地用于商务和财政方面,而这些领域不要求很高和很复杂的数据模型。虽然这些产品多多少少克服了一些以上所述的缺点,但从理论

上看，关系数据模型不直接支持复杂的数据类型，这是由于第一范式的要求，所有的数据必须转换为简单的类型，如整数、实数、双精度数和字符串。

对于工程应用来说，这种不能支持复杂数据类型的典型结果就是需要额外地分解数据结构工作，这些被分解的结构不能直接表示应用数据，且从基本成分重构时也非常烦琐和费时间。

(2) 复杂查询功能差。关系数据库系统的某些优点也同时是它的不足之处。虽然 SQL 为数据查询提供了很好的定义方法，但当用于复杂信息的查询时可能是非常烦琐的。此外，在工程应用时规范化的过程通常会产生大量的简单表。在这种环境下，由存取信息产生的查询必须处理大量的表和复杂的码联系，以及连接运算。

除非这些查询以固定的例行程序方式提供，否则用户就必须对 SQL 非常熟悉，以便适当地浏览数据库，查出所需的信息。然而，一旦查询方式按固定例行程序方式进行，用户最终就进行应用软件的常规维护。但应用或人机接口软件的变化，又可能要求经常修改例行的查询，数据库结构的变化，也可能导致例行查询程序的应用或人机接口软件的失效。由于这些原因，关系数据库系统的维护开销可能很大。

由于关系数据库不能提供足够的构造能力及性能方面的原因，在进行较复杂的数据库设计过程中，不可能将许多工程问题直接分解成一些简单的部分。由于缺乏直接指针存取方法，所以查询有关的信息需要花费时间。

(3) 支持长事务能力差。由于 RDBMS 记录锁机制的颗粒度限制，对于支持多种记录类型的大段数据的登记和检查来说，简单的记录级的锁机制是不够的，但基于键值关系的较复杂的锁机制来说却很难推广也难以实现。

(4) 环境应变能力差。在要求系统频繁改变的环境下，关系系统的成本高且修改困难。在工程应用中支持“模式演变”的功能是很重要的，而 RDBMS 不容易支持这种功能。另外，关系数据库和编程语言所提供的数据类型的不一致，使得从一个环境转换到另一个环境时需要多至 30％的附加代码。

2) 面向对象数据库系统的缺点

(1) 技术还不成熟。面向对象数据库技术的根本缺点是这项技术还不成熟，还不广为人知。与许多新技术一样，风险就在于应用。从事面向对象数据库产品和编程环境销售活动的公司还不能令人信服，因为这些公司的历史还相当短暂。ODBMS 如今还存在着标准化问题，由于缺乏标准化，许多不同的 ODBMS 之间不能通用。此外，是否修改 SQL 以适应面向对象的程序，还是用新的对象查询语言来代替它，目前还没有解决，这些因素表明，随着标准化的出现，ODBMS 还会变化。

(2) 面向对象技术需要一定的训练时间。有面向对象系统开发经验的公司的专业人员认为，要成功地开发这种系统的关键是正规的训练，训练之所以重要，是由于面向对象数据库的开发是从关系数据库和功能分解方法转化而来的，人们还需要学习一套新的开发方法使之与现有技术相结合。此外，面向对象系统开发的有关原理才刚开始形成雏形，还需一段时间在可靠性、成本等方面完善。

(3) 理论还需完善。从正规的计算机科学方面看，还需要设计出坚实的演算或理论方法来支持 ODBMS 的产品。此外，既不存在一套数据库设计方法学，也没有关于面向对象分析的一套清晰的概念模型，怎样设计独立于物理存储的信息还不明确。

面向对象数据库和关系数据库系统之间的争论，不同于 20 世纪 70 年代关系数据库和

网状数据库的争论，那时的争论是在同一主要领域（即商业事务应用）中究竟是谁代替谁的问题。现在是肯定关系数据库系统基本适合商业事务处理的前提下，对非传统的应用，特别是工程中的应用，用面向对象数据库来补充不足的问题。面向对象数据库系统将成为下一代数据库的典型代表，并和关系数据库系统并存。它将在不同的应用领域支持不同的应用需求。

**3. 数据库技术的发展方向和新型数据库**

经过 40 多年的发展，数据库技术已经得到了极大的完善，尤其是关系型数据库管理系统。随着数据库技术不断向新的应用领域的渗透，新技术的不断涌现，数据库技术将在以下几方面得到更大的发展。

1）对象-关系数据库

关系数据库几乎是当前数据库系统的标准，关系语言与常规语言一起几乎可完成任意的数据库操作，但其简洁的建模能力、有限的数据类型、程序设计中数据结构的制约等不足，阻碍了关系型数据库更广泛的应用。面向对象方法起源于程序设计语言，它本身就是以现实世界的实体对象为基本元素来描述复杂的客观世界，但功能不如数据库灵活。因此将面向对象的建模能力和关系数据库的功能进行有机结合，是数据库技术的一个发展方向。

2）数据仓库与数据挖掘

数据仓库技术是从数据库技术发展而来的，是面向主题的、稳定的、综合的、随时间变化的数据集合。创建数据仓库的主要目标是：使各种各样的数据源数据对于如执行官、经理、分析家这些人，能够帮助他们做出符合发展规律的决策。随着商业竞争愈来愈激烈，数据仓库、数据发掘技术的应用会越来越普遍，其产品会更加成熟。

3）实时数据库技术

实时数据库管理系统（RTDBMS）是数据库系统发展的一个分支，它适用于处理不断更新的快速变化的数据，以及具有时间限制的事务处理。实时数据库技术是实时系统和数据库技术相结合的产物，利用数据库技术来解决实时系统中的数据管理问题，同时利用实时技术为实时数据库提供时间驱动调度和资源分配算法。我们相信它必将对传统数据库系统发展有巨大的推动作用，从而推动数据库技术在现代信息社会中更广泛的应用。

4）Web 数据库

基于 Web 的数据库应用系统，是将数据库和 Web 技术结合，通过浏览器访问数据库，并实现动态的信息服务系统。利用扩展技术和一些相应的软件将数据库和 Web 结合起来，在 Web 上提供用户访问和修改数据库的接口，用户就能通过浏览器在任何地方访问这些数据库。

当今社会对数据库技术有着广泛的应用需求，这必将对数据库技术起到极大的推动作用。另外，数据库技术与新出现的各种技术的相互结合、相互渗透，必将数据库技术引向更广泛的应用领域。

## 5.9 软件工程基础

软件工程是一门应用工程学的概念、原理、技术和方法来构建和维护有效的、实用的和高质量的软件的学科，它涉及软件计划、需求分析、设计、编码、测试和维护的开发过程、方法和工具等诸多方面的原理及应用知识，是一门用来指导计算机软件开发和维护工作的理论学科。软件工程的目的是提高软件生产率、提高软件质量、降低软件成本，简而言之，是用科

学的方法以较低的成本研制具有较高质量的软件。

## 5.9.1 软件生命周期

**1. 软件生命周期的概念**

软件的生命周期，也称为软件的生存周期，它是按开发软件的规模和复杂程度，从时间上把软件开发的整个过程（从计划开发开始到软件报废为止的整个阶段）进行分解，形成相对独立的几个阶段，每个阶段又分解成几个具体的任务，然后按规定顺序依次完成阶段的任务并规定一套标准的文档作为各个阶段的开发成果，最后生产出高质量的软件。根据我国现行信息技术国家标准规定（国标名：系统与软件工程软件生存周期过程，国标号 GB/T 8566—2022），将软件生命周期过程分为 4 个过程组：协定过程组、技术管理过程组、技术过程组和组织的项目使能过程组。软件生命周期过程如图 5.76 所示。

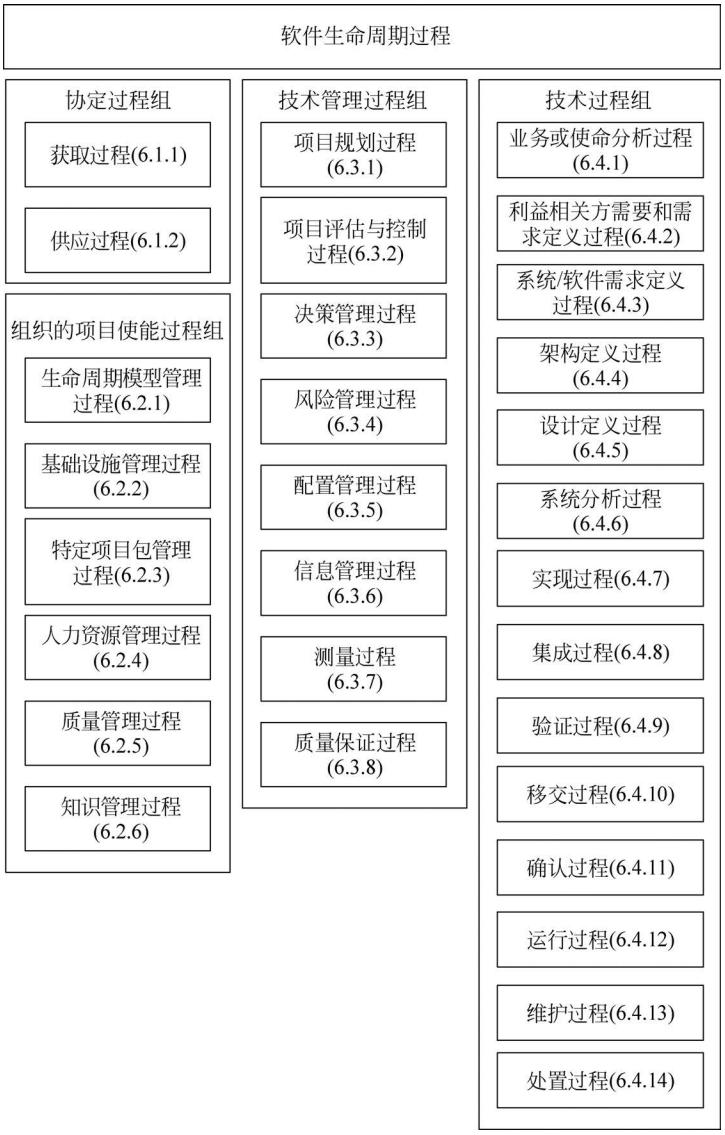



图 5.76　软件生命周期过程

**2. 软件工程的基本原理**

中大型软件开发过程本身就是一项复杂的工程活动，需要投入大量的人力、物力和财力，就像建筑工程一样，必须按照工程方法学来标准化管理，一般来讲，应该遵循以下 7 条基本原理。

1）用分阶段的生存周期计划严格管理

统计发现，不成功的软件项目有 50％以上是因计划不周而造成的，可见把制作完善的开发计划作为首条基本原理是吸取了前人的经验教训而提出来的。

在软件开发与维护的整个生存周期中，需要完成许多性质各异的工作。本条基本原理意味着，应该把软件生存周期划分成若干阶段，并相应地制定出切实可行的计划，在项目实施过程中严格按照计划对软件的开发与维护工作进行管理。Boehm 认为，在软件的整个生存周期中应该制订至少 6 类计划，它们是项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划。每一阶段，项目不同层次人员都必须严格按照项目计划在规定的时间周期内管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划，如有偏离，需及时纠正。

2）坚持进行阶段评审

大量经验表明，软件的质量保证工作不能等到某一阶段如编码阶段结束之后再进行。这样说至少有两个理由：第一，大部分错误是在设计阶段造成的，或是因需求分析不够正确而造成设计失误，例如，据 Boehm 等人的统计，设计错误占软件错误的 63％，编码错误仅占 37％；第二，错误发现与改正得越晚，修正软件所需付出的代价也越高。因此，在每个阶段都应该进行严格的评审和测试工作，以便尽早发现软件开发过程每一阶段所犯的错误，是一条必须遵循的重要原则。

3）实行严格的产品控制

在软件开发过程中不应随意更改软件需求，因为每改变一项需求意味着设计图纸需要更改，因此往往需要付出更高的代价，但是在软件开发过程中改变需求又是难以避免的，由于外部主观、客观条件的变化，相应地用户需求的改变则是必然的，显然不能硬性禁止客户变更需求，而只能依靠科学的产品控制方法来适应这种客观状况。也就是说，当需求改变时，为了确保软件各个配置成分的一致性，必须实行严格的产品控制，其中最主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分（即每一开发阶段产生的文档资料或程序代码）。基准配置管理也称为变动控制，一切有关修改软件的建议，特别是涉及对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。绝对不能谁想修改软件（包括尚在开发过程中的软件）就随意进行修改。

4）采用现代程序设计方法

从软件工程概念产生开始，人们一直把主要精力用于研究各种新的程序设计方法。20 世纪 60 年代末提出的结构化程序设计方法，已经成为绝大多数人公认的经典的程序设计方法，以后在此基础之上又进一步发展出各种结构分析（SA）与结构设计（SD）方法。实践表明，采用先进的设计方法既可提高软件开发的效率，降低软件开发风险，又可以提高软件维护的效率。

5）结果应能清楚地审查

软件产品不同于一般的实物产品，它是看不见摸不着的逻辑产品。软件开发人员（或开发工程组）的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评估和管理。为了提高软件开发过程的可见性，更好地进行管理，应根据开发项目计划的总目标及完成期限，明确开发组织及团队成员的责任和产品标准，从而使得所得到的结果能够清楚地审查。

6）开发小组的人员应该少而精

这条基本原理的含义是，开发小组的成员素质应该好而精，人数不宜过多。开发小组成员的素质是影响软件产品质量和开发效率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍甚至几十倍，而且素质高的人员所开发的软件错误数量明显少于素质低的人员所开发的软件错误数量。此外，随着开发小组成员数目的增加，人力资源成本也急剧增加。当开发小组人员数为 $N$ 时，可能的通信路径有 $N \times (N-1)/2$ 条，可见随着人数 $N$ 的增大，人力成本将急剧增加。因此，建立少而精的开发团队是软件工程的一条基本原理。

7）不断改进软件工程实践的必要性

遵循上述 6 条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产，但是，仅遵循以上 6 条原理并不能保证软件开发与维护过程的技术方法与时俱进，因此 Boehm 提出应把不断改进软件工程实践的必要性作为软件工程的第 7 条基本原理。遵循这条原理，软件工程不仅要积极主动地采纳新的软件技术方法，而且要不断总结经验，例如，积极收集项目进度和资源耗费数据、积极收集出错类型和问题报告数据等。这些数据不仅可以用来评价新开发技术应用的成效，而且可以用来指导我们必须着重开发哪些软件工具和应该优先采用哪些技术。

**3. 软件工程的三要素**

前面介绍过软件的生命周期，不难发现，软件工程包括技术和管理两方面的内容，是技术与管理紧密结合所形成的工程学科。通常把在软件生命周期全过程中使用的一整套技术方法的集合称为方法学。软件工程方法学包含三个要素：方法、工具和过程。其中，方法是完成软件开发的各项任务的技术方法，是回答“怎样做”的问题；工具是为运用方法而提供的自动的或半自动的软件工程支撑环境；过程是为了获得高质量的软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

目前，最广泛应用的软件工程方法学分别是传统方法学（面向过程方法学）和面向对象方法学。

1）传统方法学

传统方法学也称面向过程方法学，它采用结构化技术（结构化分析、结构化设计和结构化实现）来完成软件开发的各项任务，并使用适当的软件工具或软件工程来支持结构化技术的运用。这种方法学把软件生命周期的全过程依次划分为若干阶段，然后顺序地完成每个阶段的任务。采用这种方法开发软件，从对问题的抽象逻辑分析开始，逐阶段地按顺序进行开发。前一个阶段任务的完成是启动后一个阶段工作的前提和基础，而后一阶段任务的完成通常是使前一阶段提出的问题解决方法更进一步具体化，加入了更多的实现细节。每个

阶段的开始和结束都有严格的标准，对于任何两个相邻的阶段而言，前一阶段的结束标准就是后一阶段的开始标准。在每个阶段结束之前都必须进行正式严格的技术审查和管理复审，从技术和管理两方面对这个阶段的开发成果进行检查，检查通过之后这个阶段才算结束；如果检查不通过，则必须进行必要的返工，并且返工后还要再经过审查。审查的一条主要标准就是每个阶段都应该交出最新的、和所开发的软件完全一致的、高质量的文档资料，从而保证在软件开发工程结束时有一个完整准确的软件配置交付使用。文档是项目成员之间通信的工具，它们清楚准确地说明了到这个时候为止，关于该项工程已经知道了什么，同时确立了下一步工作的基础。此外，文档也起备忘录的作用，如果文档不完整，那么一定是某些工作忘记做了，在进入生存周期的下一阶段之前，必须补足这些遗漏的细节。在完成生存周期每个阶段的任务时，应该采用适合该阶段任务特点的系统化的技术方法——结构分析或结构设计技术。

把软件生存周期划分成若干阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度；在软件生存周期的每个阶段都采用科学的管理技术和良好的技术方法，而且在每个阶段结束之前都从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发工程的全过程以一种有条不紊的方式进行，保证了软件的质量，特别是提高了软件的可维护性。总之，采用软件工程方法论可以大大提高软件开发的成功率，软件开发的生产效率也能明显提高。

2）面向对象方法学

当软件规模变得庞大，或者对软件的需求是模糊的，或者软件的需求会随时间变化较大的时候，使用传统方法学开发软件往往是错误的。此外，使用传统方法学开发出的软件维护起来也是十分困难的。

结构化技术只能获得有限成功的一个重要原因是因为这种技术要么就面向行为（即对数据的操作），要么就面向数据，还没有既面向数据又面向行为的结构化技术。众所周知，软件系统本质上是信息处理系统，离开了操作便无法更改数据，而脱离了数据的操作是毫无意义的。数据和对数据的处理原本是密切相关的，把数据和操作人为地分离成两个独立的部分，自然会增加软件开发与维护的难度。与传统方法相反，面向对象方法把数据和行为看成同等重要，它是一种以数据为主线，把数据和对数据的操作紧密地结合起来的方法。

概括地说，面向对象方法学具有下述 4 个要点。

（1）把对象（Object）作为融合了数据和行为（施加在数据上的操作）的软件构件。面向对象程序是由对象组成的，程序中任何元素都是对象，复杂对象由比较简单的对象组合而成。也就是说，用对象分解取代了传统方法的功能分解。

（2）把所有对象都划分成类（Class）。每个类都定义了一组数据和一组操作，类是对具有相同数据和相同操作的一组相似对象的定义。数据用于表示对象的静态属性，是对象的状态信息，而施加于数据之上的操作用于实现对象的动态行为。

（3）按照父类（或称为基类）与子类（或称为派生类）的关系，把若干相关类组成一个层次结构的系统（也称为类等级）。在类等级中，下层派生类自动拥有上层基类中定义的数据和操作，这种现象称为继承。

（4）对象彼此间仅能通过发送消息互相联系。对象与传统数据有本质区别，它不是被

动地等待外界对它施加操作，相反地，它是数据处理的主体，对象所有私有信息都被封装在该对象内，不能从外界直接访问。

**4. 软件工程师的基本职责、基本素质与技能**

软件工程师是一种从事软件开发和软件工程实践的专业人员。他们负责设计、开发、测试和维护软件系统，以满足用户需求和解决问题。软件工程师在整个软件开发生命周期中发挥关键作用，包括需求分析、系统设计、编码、测试、部署和维护等阶段。

1）软件工程师的基本职责

（1）需求分析：理解和收集软件系统的需求，并将其转换为可行的设计方案。

（2）系统设计与架构：根据需求分析的结果，设计软件系统的整体架构和组件，确保系统的可靠性、可扩展性和可维护性。

（3）编码和实现：使用编程语言和工具，将系统设计转换为可执行的代码，负责实现各个模块和功能。

（4）软件测试：编写和执行测试用例，确保软件系统的功能和质量，识别和修复软件缺陷。

（5）版本控制和协作：使用版本控制工具（如 Git）管理代码库，与团队成员协作开发，解决冲突和合并代码。

（6）故障排除和维护：监测和解决软件系统中的错误和故障，维护系统的正常运行。

（7）文档编写：编写软件开发文档、用户手册和技术文档，以便其他人能够理解和使用软件系统。

（8）技术研究和学习：不断学习新的技术和工具，跟踪行业的发展趋势，以保持自身的竞争力和创新能力。

（9）项目管理：参与软件项目的规划和组织，确保项目按时交付，并有效地管理资源和时间。

（10）沟通与协作：与团队成员、用户和其他利益相关者进行有效的沟通和协作，解释技术细节和理解需求。

2）软件工程师的基本素质与技能

软件工程师需要具备扎实的编程技能、系统设计能力、软件测试经验和良好的沟通协作能力。同时，持续学习和不断提升自身的技术和知识也是软件工程师的重要素质。软件工程师应具备的基本素质和技能如下。

（1）编程能力：软件工程师应具备扎实的编程技能，能够使用至少一种编程语言进行软件开发，理解算法和数据结构，并能够解决复杂的编程问题。

（2）系统设计与分析：软件工程师应具备良好的系统设计和分析能力，能够理解用户需求，并将其转换为可行的软件设计方案。

（3）软件开发方法论：熟悉软件开发的不同方法论和流程，如敏捷开发、瀑布模型等，并能够根据项目需求选择合适的方法论。

（4）数据库知识：了解数据库的基本原理和常用操作，能够设计和管理数据库，进行数据的存储和检索。

（5）软件测试与调试：具备良好的测试和调试技能，能够编写有效的测试用例，进行软件缺陷的识别和修复。

(6) 沟通与协作能力：良好的沟通和协作能力对于软件工程师非常重要。能够与团队成员、用户和其他利益相关者有效地沟通，并理解他们的需求和反馈。

(7) 持续学习与创新意识：软件行业发展迅速，软件工程师需要持续学习新的技术和工具，并保持对新技术的探索和创新意识。

(8) 问题解决能力：能够独立思考和解决问题，分析和解决软件开发过程中的挑战和障碍。

(9) 质量意识：关注软件的质量和可靠性，并遵循最佳实践和标准(如国标、行业标准等)，以确保开发出高质量的软件产品。

(10) 项目管理：了解基本的项目管理原理和方法，能够合理规划和组织软件项目，并有效地分配资源和时间。

## 5.9.2 系统分析

生命周期基本过程中的开发过程始于分析阶段，这个阶段的结果是生成了软件的规格说明文档，但软件规格说明文档只说明了软件要做什么的问题，而并没有说明如何去做的方法。分析阶段可以使用两种独立的方法，它们依赖于实现阶段使用的是过程式编程语言还是面向对象语言。本节简单地讨论这两种方法。

**1. 面向过程分析**

如果实现阶段使用过程式语言，那么面向过程分析就是分析阶段使用的方法。这种情况下的规格说明可以使用多种建模工具，但这里只讨论其中少数几个。

1) 数据流图

数据流图是结构化分析方法中使用的工具，它以图形的方式描绘数据在系统中流动和处理的过程，由于它只反映系统必须完成的逻辑功能，所以它只是一种功能模型。在面向过程开发方法中，数据流图是需求分析阶段产生的结果，它们使用 4 种符号，如图 5.77 所示，图 5.78 是数据流图的例子，它反映了旅行社录入订票单数据后，系统对订票单的处理过程，系统是如何将订票单数据加工成机票信息和账单信息，以及数据在系统当中的中间变化和流动过程。

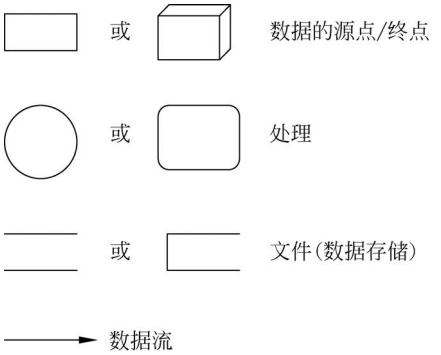



图 5.77 数据流图的基本符号

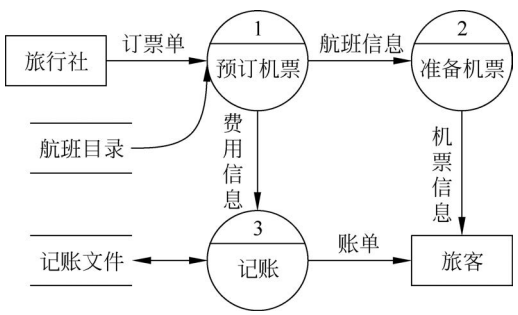



图 5.78 数据流图的例子

2) 实体-关系图

分析阶段使用的另一个建模工具是实体-关系图，即 E-R 图。E-R 图提供了表示实体、属性和联系的方法，用来描述现实世界的概念模型。E-R 图用矩形表示实体，矩形框内写明实体名；用椭圆表示实体的属性，并用无向边将其与相应的实体连接起来；用菱形表示实

体之间的联系，在菱形框内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型(1∶1，1∶$n$ 或 $m$∶$n$)，如图 5.79 所示。图 5.80 是反映教师、课程、学生三个实体之间的关系图的例子。教师与课程之间是一对多的关系，学生与课程之间是多对多的关系。

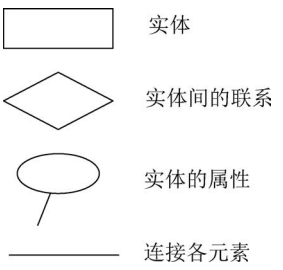



图 5.79　E-R 图的基本元素

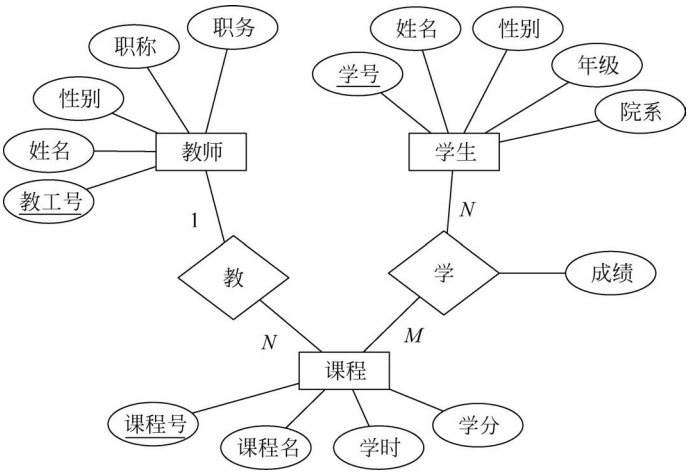



图 5.80　E-R 图的实例

3）程序流程图

程序流程图是程序分析中最基本、最重要的分析技术，它是进行程序流程分析过程中最基本的工具。它运用工序图示符号对生产现场的整个制造过程做详细的记录，以便对零部件、产品在整个制造过程中的生产、加工、检验、存储等环节做详细的研究与分析，特别适用于生产过程中的成本控制、效益分析等用途。如图 5.81 所示为程序流程图的一个实例。

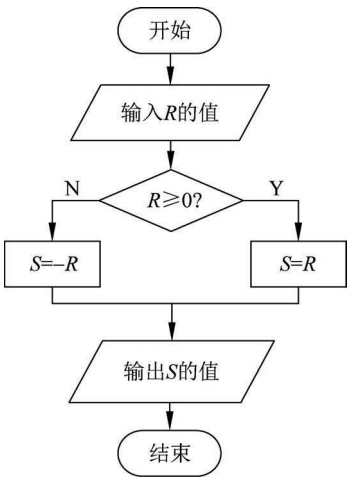



图 5.81　程序流程图的实例

**2. 面向对象分析**

如果实现过程使用的是面向对象语言，那么分析过程就是面向对象分析过程。这种情况下的规格说明文档可以使用多种工具，但这里只讨论其中少数几种。

1）用例图

用例图给出了系统的用户视图：它显示了用户与系统间的交互。用例图使用 4 种组件：系统、用例、动作者和关系。系统(用矩形表示)执行功能。系统中的行动由用例显示，它用圆角的矩形表示。动作者是使用系统的某人或某事。虽然动作者用线条人物来表示，但它们并不需要表示人类。

图 5.82 显示老式电梯的用例图。这个图中的系统是电梯，唯一的动作者是电梯的使用者。这里有两个用例：按电梯按钮(在每层的大厅)和在电梯内按楼层按钮。电梯在每层只有一个按钮，它发送电梯移到该层的指令。

2）类图

类图是显示了模型的静态结构，特别是模型中存在的类、类的内部结构以及它们与其他

类的关系等。

类图由许多说明性的模型元素组成，例如，类、包和它们之间的关系，这些元素和它们的内容互相连接。类图可以组织在包中，仅显示特定包中的相关内容，不显示暂时性信息。类图是最常用的 UML 图，显示出类、接口以及它们之间的静态结构和关系；它用于描述系统的结构化设计。类图最基本的元素是类或接口。图 5.83 显示了电梯问题的一个类图。

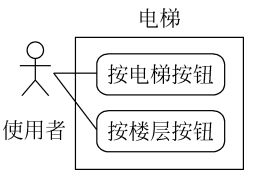



图 5.82　用例图的实例

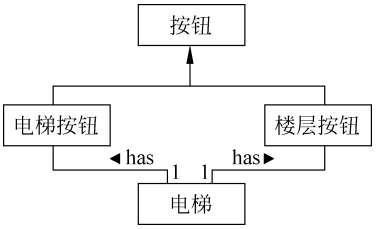



图 5.83　类图的实例

3）状态图

状态图是描述一个实体基于事件反应的动态行为，显示了该实体如何根据当前所处的状态对不同的事件做出反应，通常我们创建一个 UML 状态图是为了研究类、角色、子系统或组件的复杂行为。

## 5.9.3　系统设计

设计阶段定义系统如何完成在分析阶段所定义的需求。在设计阶段，系统所有的组成部分都被定义。系统设计内容主要包括：确定设计方针和方法；将系统分解为若干子系统；确定各子系统的目标、功能及其相互关系；决定对子系统的管理体制和控制方式；对各子系统进行技术设计和评价；对全系统进行技术设计和评价等。

**1. 面向过程设计**

在面向过程设计中，既要有设计过程，也要有设计数据。我们讨论一类注重过程的设计方法。在面向过程设计中，整个系统被分解成一组过程或模块。

1）结构图

在面向过程设计中，说明模块间关系的常用工具是结构图。结构图是指以模块的调用关系为线索，用自上而下的连线表示调用关系并注明参数传递的方向和内容，从宏观上反映软件层次结构的图形。图 5.84 是画结构图的基本符号，图 5.85 是结构图的一个例子。

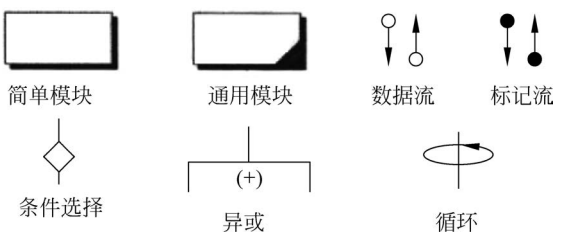



图 5.84　画结构图的基本符号

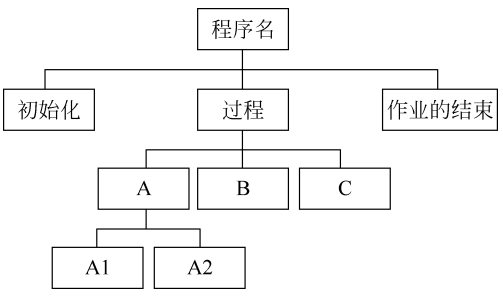



图 5.85　结构图的例子

2）模块化

模块化是指解决一个复杂问题时自顶向下逐层把系统划分成若干子模块的过程，通过模块的不同组合得到不同品种、不同规格的产品。模块化意味着将大项目分解成较小的部分，以便能够容易理解和处理，换言之，模块化意味着将大程序分解成能相互通信的小程序。模块化是一种将复杂系统分解成多个更好管理的子模块的方式，当系统被分解成模块时，主要关心两点：耦合性和内聚性。

3）耦合性

耦合性(Coupling)，也叫耦合度，耦合是对模块间互相绑定紧密程度的度量。耦合的强弱取决于模块间接口的复杂性、调用模块的方式以及通过界面传送数据的多少。模块间的耦合度是指模块之间的依赖关系，包括控制关系、调用关系、数据传递关系。模块间联系越多，其耦合性越强，同时表明其独立性越差。软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分模块的一个准则就是高内聚低耦合。越紧耦合的模块，它们的独立性越差。既然目标是为了让模块尽可能地独立，那么需要让它们松散耦合。至少有三条理由能说明在系统设计过程中应尽可能地追求模块间的松散耦合。

（1）松散耦合的模块更可能被重用。

（2）松散耦合的模块不容易在相关模块中产生错误。

（3）当系统需要修改时，松散耦合的模块允许我们只修改需要改变的模块，而不会影响到不需要改变的模块。

注意：软件系统中模块间的耦合必须最小化。

4）内聚性

模块化的另一个问题是内聚性。内聚性，又称块内联系，是指模块之间的功能强度的度量，即一个模块内部各个元素彼此结合的紧密程度的度量。模块中组成元素结合得越紧密，模块的内聚性就越高，模块的独立性也就越高。理想的内聚性要求模块的功能应明确、单一，即一个模块只做一件事情。

注意：软件系统模块间的内聚必须最大化。

**2. 面向对象设计**

在面向对象设计中，设计阶段通过详细描述类的细节（列出类的属性和方法）来继续。图 5.86 显示了在老式电梯设计中使用的 4 个类的细节。

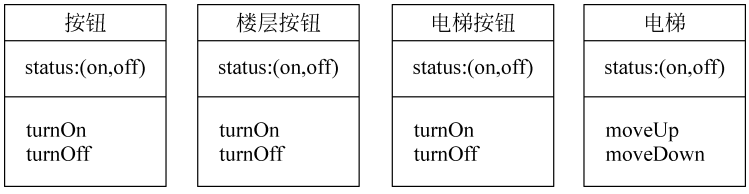



图 5.86　类图的例子

## 5.9.4　系统实现

在瀑布模型中，设计阶段完成之后，实现阶段就可以开始了。在这个阶段，程序员为面向过程设计中的模块编写程序或者编写程序单元，实现面向对象设计中的类。在每种情况中，都有一些我们需要提及的问题。

**1. 语言选择**

在面向过程开发中，工程团队需要从面向过程语言中选择一种或一组语言。虽然有些语言(像 C++)被看成既是面向过程的，又是面向对象的语言，但通常实现使用纯过程语言，如 C 语言。在面向对象的情况下，C++和 Java 的使用都很普遍。

**2. 软件质量**

软件质量是软件工程项目所关心的一个非常重要的问题，高质量的软件系统是一个能够满足用户需求、符合组织操作标准和能够高效运行在为其开发的硬件上的软件。但是，如果需要取得高质量的软件系统，那么必须定义软件质量的一些属性。

软件质量可以划分成三个广义的度量：可操作性、可维护性和可迁移性。每个度量还可以展开，如图 5.87 所示。

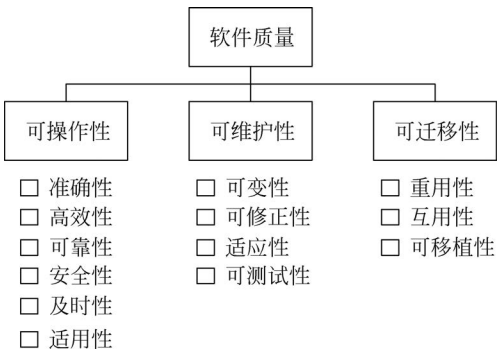



图 5.87　质量因素

1) 可操作性

可操作性涉及系统的基本操作。可操作性有多种度量方法：准确性、高效性、可靠性、安全性、及时性和适用性。

(1) 不准确的系统比没有系统更糟糕。任何被开发的系统都必须经过系统测试工程师和用户检测。准确性能够通过诸如故障平均时间、每千行代码错误数，以及用户请求变更数等这样的测量指标来度量。

(2) 高效性大体上是个主观的术语。在一些实例中，用户为开发方提出一些条件，开发的系统必须满足特定的性能指标，例如，实时响应必须在 1s 之内接收到，成功率为 95%。

(3) 可靠性实际上综合了其他各种因素。如果用户指望系统完成工作并对其有信心，这时它就是最可靠的。另外一些度量直接说明了系统的可靠性，最显著的是故障平均时间。

(4) 一个系统的安全是以未经授权的人得到系统数据的难易程度为参照的。尽管这是一个主观的领域，但仍然有可能利用核查的清单来帮助评估系统来评估系统的安全性。例如，系统有并且需要密码来验证用户吗?

(5) 在软件工程中及时性意味着几件不同的事情，如系统及时传递它的输出了吗? 对于在线系统，响应时间满足用户的需求了吗?

(6) 适用性是另一个很主观的领域。适用性的最好的度量方法在于用户，例如，看他们是如何正在使用这个系统，用户访谈也能够发现系统适用性上的问题。

2) 可维护性

可维护性以保持系统正常运行并及时更新为参照。很多系统需要经常修改，这不是因为它们不能很好地运行而是因为外部因素的改变。例如，一个公司的工资单系统就不得不经常修改以满足政府法律和规则的改变。

可变性是一个主观因素，但是一个有经验的项目领导可以估计出多长时间需要发生一次改变请求。如果持续时间太长，则可能是因为系统很难改变。

可修正性的一种度量是恢复正常的平均时间，也就是当程序发生故障后使程序恢复运行所花费的时间。虽然它是反应性定义，但目前还没有方法来预测从故障中改正程序需要花多少时间。

用户经常要求系统进行变动，适应性是一个定性的属性，它用来度量系统变动的难易程度。如果一个系统需要完全重写程序才能进行改变的话，那它就没有灵活性。

你可能会认为可测试性是一个很主观的领域，但测试工程师有包含各种因素的检测清单来评估系统的可测试性。

3）可迁移性

可迁移性是指把数据或系统从一个平台移动到另一个平台并重用代码的能力。在很多情况下，这不是一个很重要的因素。另外，如果编写具有通用性的软件，那么可迁移性就很关键了。

（1）如果编写的模块可以在其他系统中使用，那么它具有很好的重用性。好的程序员建立函数库以便在解决类似的问题时能够重用这些函数。

（2）互用性是发送数据给其他系统的能力。在当今高度集成的系统中，这是非常需要的属性。事实上，它已经变得如此重要以致操作系统现在都支持系统之间传递数据的能力，例如，在文字处理软件和电子数据表之间传递数据。

（3）可移植性是一种把软件从一个硬件平台转移到另一个硬件平台的能力。

## 5.9.5 系统测试

测试阶段的目标就是发现错误，这就意味着良好的测试策略能够发现更多错误。软件测试是在规定的条件下对程序进行操作，以便发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程。软件测试有两种基本的测试方法：白盒测试和黑盒测试，如图 5.88 所示。

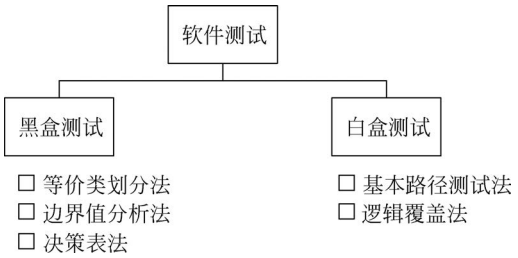



图 5.88 软件测试

**1. 黑盒测试**

黑盒测试（也称功能测试）是指在不知道程序的内部也不知道程序是如何工作的情况下进行的测试。换言之，程序就像看不见内部的黑盒。黑盒测试按照软件应该完成的功能来测试软件，如它的输入和输出。下面介绍几种黑盒测试方法。

1）等价类划分法

在进行功能测试的时候，很多情况下，一组数据的测试效果等同于这一组数据所属类别的测试效果，那么，在测试的时候就可以大大节省测试的时间。例如，对百度文库的页码输入框进行测试，假设有一篇文章的最大页数为 17，则可将输入值分为三个类别，第一类为[1,17]的闭合区间，第二类为大于 17 的区间，第三类为小于 1 的区间。如果在每个类别设计一组数据进行功能测试，如在[1,17]区间设计一组数据为 10，在大于 17 的区间设计一组数据为 20，在小于 1 的区间设计一组数据为 0，那么，10、20 和 0 这三组数据的测试效果就能够达到这三类所有数据的测试效果，这种测试方法即为等价类划分法。

2）边界值分析法

当遇到边界时，错误经常发生。例如，一个模块定义它的输入值必须大于或等于 100，那这个模块用边界值 100 来测试就非常重要。如果模块在边界值出错，那有可能就是模块代码中的有些条件设置错误，例如，$x>=100$ 被写成 $x>100$。

3）决策表法

决策表法是利用决策表来设计测试用例的一种测试方法，决策表又称为判断表，是一种呈表格状的图形工具，适用于判断条件较多、各条件又相互组合、有多种决策方案的情况。例如，关于行李托运收费的模块，其业务逻辑描述如下。

如果行李不超过 30kg，那么可以免费托运；如果行李超过 30kg，那么对头等舱乘客超过部分每千克收费 4 元，对普通舱乘客超重部分每千克收费 6 元；如果乘客是残疾人，那么，收费减半。

根据该业务逻辑描述，可以为其建立一张决策表（$W$ 表示重量），如表 5.16 所示。

**表 5.16　决策表示例**

| 条件组合 | | 1 | 2 | 3 | 4 | 5/6/7/8 |
|---|---|---|---|---|---|---|
| 条件 | $W>30$kg | √ | √ | √ | √ | |
| | 头等舱乘客 | √ | √ | | | -- |
| | 残疾乘客 | √ | | √ | | -- |
| 行为 | $(W-30)\times 2$ | √ | | | | |
| | $(W-30)\times 4$ | | √ | | | |
| | $(W-30)\times 3$ | | | √ | | |
| | $(W-30)\times 6$ | | | | √ | |
| | 免费 | | | | | √ |

从以上决策表可知，只要输入 $W$ 的值、乘客的舱位类型以及乘客的健康情况，则该模块应输出相应的行李托运收费结果。因此，可以根据决策表中的 5 种条件组合设计 5 组测试数据，就可以对该模块实行功能测试。例如：

测试数据 1：$W=40$kg，乘客的舱位类型＝头等舱，健康情况＝残疾（对应条件组合 1）

测试数据 2：$W=40$kg，乘客的舱位类型＝头等舱，健康情况＝健康（对应条件组合 2）

测试数据 3：$W=40$kg，乘客的舱位类型＝普通舱，健康情况＝残疾（对应条件组合 3）

测试数据 4：$W=40$kg，乘客的舱位类型＝普通舱，健康情况＝健康（对应条件组合 4）

测试数据 5：$W=20$kg，乘客的舱位类型＝头等舱或普通舱，健康情况＝残疾或健康（对应条件组合 5/6/7/8）

思考：每一组测试数据对应的输出结果是什么？

**2. 白盒测试**

白盒测试（又称结构测试、玻璃盒测试）是基于了解软件内部结构的一种测试方法。测试的目标是检查软件所有的部分是否全部设计出来。白盒测试假定测试者知道有关软件的一切，在这种情况下，程序就像玻璃盒子，其中的每件事情都是可见的。白盒测试是全面了解程序内部逻辑结构、对所有逻辑路径进行测试的一种测试方法，一般由软件工程师或一个专门的团队来完成。使用白盒测试方法需要保证至少满足下面 4 条标准。

（1）每个模块中的所有独立的路径至少被测试过一次。

（2）所有的判断结构（两路的或多路的）每个分支都被测试。

（3）每个循环被测试。

（4）所有的数据结构都被测试。

白盒测试有多种测试方法，这里只讨论其中的两种：基本路径测试法和逻辑覆盖法。

1）基本路径测试法

基本路经测试是由 Tom McCabe 提出的。该方法创建一组测试用例，使得软件中的每条语句至少被执行一次。基本路径测试法需要找出被测程序中的所有路径的集合，当程序不是很复杂的时候，或许很容易找出程序中的所有可能路径，但当程序变得非常复杂的时候，找出程序的所有可能路径就变得十分困难，因此，在基本路径测试法的基础上，人们提出了一种更容易找出被测程序的所有路径集合的子集的方法，即独立路径测试方法。

独立路径测试法是在程序控制流图的基础上，通过分析程序的环路复杂性，导出独立路径集合，从而设计测试用例的方法。

图 5.89 是一个独立路径测试的例子，假定系统只由一个程序构成，程序只有一个单循环，左边为程序的控制流图，右边列出了它的程序环路复杂性和独立路径数。

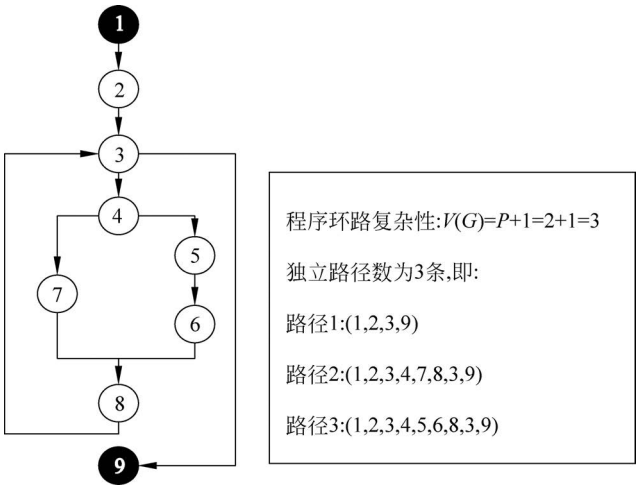



图 5.89　独立路径测试的例子

2）逻辑覆盖法

逻辑覆盖法又细分为多种测试方法：语句覆盖、判断覆盖、条件覆盖、判断/条件覆盖和条件组合覆盖。

（1）语句覆盖：是指设计若干测试用例，程序运行时每个可执行语句至少被执行一次。

（2）判断覆盖：是指设计若干测试用例，执行被测试程序时，程序中每个判断条件的真值分支和假值分支至少被执行一遍。判断覆盖又称为分支覆盖。

（3）条件覆盖：是指设计若干测试用例，执行被测试程序时，程序中每个判断条件中的每个判断式的真值和假值至少被执行一遍。

（4）判断/条件覆盖：是指设计若干测试用例，执行被测试程序时，程序中每个判断条件的真假值分支至少被执行一遍，并且每个判断条件的内部判断式的真假值分支也要被执行一遍。

（5）条件组合覆盖：是指设计若干测试用例，执行被测试程序时，程序中每个判断条件的的内部判断式的各种真假组合可能都至少被执行一遍。

### 5.9.6 软件文档

软件的正确使用与有效维护离不开文档。软件文档编制贯穿于软件产品开发的各个阶段，是提高软件产品开发效率、规范软件产品开发过程、保证软件产品质量的关键，国家发布的一系列软件文档标准有力地促进了软件产品开发的系统化、规范化和标准化。软件通常有三种独立的文档：用户文档、系统文档和技术文档。注意：文档是一个持续的过程。如果软件在发布之后有问题，也必须写文档。如果软件被修改，那么所有的修改也要写进文档。只有当软件过时被抛弃后，编写文档才停止。

**1. 用户文档**

为了能够正确使用软件，用户手册对用户来说是必不可少的。它告诉用户如何一步一步地使用软件系统。它通常包含一个教程，用来指导用户熟悉软件系统的各项特性。

一个好的用户手册能够成为一个功能强大的营销工具。用户文档在营销中的重要性再强调也不过分。手册应该面向新手和专业用户。配有好的用户文档必定有利于软件的销售。

**2. 系统文档**

系统文档定义软件本身。撰写系统文档的目的是让原始开发人员之外的人能够维护和修改软件系统。系统文档在系统开发的所有 4 个阶段都应该存在。

在分析阶段，收集的信息应该仔细地用文档记录。另外，系统分析员应该定义信息来源。需求和选用的方法必须基于它们的推论来清楚表述。

在设计阶段，最终版本中用到的工具必须记录在文档中。例如，如果结构图修改了多次，那么最终的版本要用完整的注释记录在案。

在实现阶段，代码的每个模块都应记录在文档中。另外，代码应该使用注释和描述头尽可能详细地说明代码的功能用途。

最后，开发人员必须仔细地形成测试阶段的文档。对最终产品使用的每种测试，连同它的结果都要记录在文档中。甚至令人不快的最坏结果和产生它们的数据也要记录在案。

**3. 技术文档**

技术文档描述了软件系统的安装和服务；安装文档描述了系统软件如何安装在每台计算机上，如服务器和客户端；服务文档则描述了系统应该如何维护和更新。

### 5.9.7 软件项目管理

软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利地完成并且对人员、产品、过程和项目进行分析和管理的活动。软件项目管理的根本目的是让软件项目(尤其是大型项目)的整个软件生命周期都能在管理者的控制之下，以预定成本按期、按质地完成软件交付给用户使用。而研究软件项目管理是为了从已有的成功或失败的案例中总结出能够指导今后开发的通用原则、方法，指导实践活动。

软件项目管理的提出是在 20 世纪 70 年代中期的美国，当时美国国防部专门研究了软件开发不能按时提交，预算超支和质量达不到用户要求的原因，结果发现 70%的项目是因为管理不善而引起的，而非技术原因。于是软件开发者开始逐渐重视起软件开发中的各项管理。到了 20 世纪 90 年代中期，软件研发项目管理不善的问题仍然存在。据美国软件工

程实施现状的调查，软件研发的情况仍然很难预测，大约只有10%的项目能够在预定的费用和进度下交付。1995年，据统计，美国共取消了810亿美元的商业软件项目，其中31%的项目未做完就被取消，53%的软件项目进度通常要延长50%的时间，只有9%的软件项目能够及时交付并且费用也控制在预算之内。

软件项目管理和其他的项目管理相比有相当的特殊性。首先，软件是纯知识产品，其开发进度和质量很难估计和度量，生产效率也难以预测和保证。其次，软件系统的复杂性也导致了开发过程中各种风险难以预见和控制。Windows这样的操作系统有1500万行以上的代码，同时有数千个程序员在进行开发，项目经理都有上百个。这样庞大的系统如果没有很好的管理，其软件质量是难以想象的。

**1. 软件项目管理的内容**

软件项目管理的内容主要包括如下几方面：人员组织与管理、计划管理、软件度量、风险管理、软件质量保证、软件过程能力评估、软件配置管理等。

这几方面都是贯穿、交织于整个软件开发过程中的，其中，人员组织与管理把注意力集中在项目组人员的构成、优化；软件度量是用量化的方法评测软件开发中的要素（如费用、生产率、进度和产品质量等）是否符合期望值，包括过程度量和产品度量两个方面；软件项目计划主要包括工作量、成本、开发时间的估计，并根据估计值制定和调整项目组的工作；风险管理预测未来可能出现的各种危害到软件产品质量的潜在因素并由此采取措施进行预防；质量保证是保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动；软件过程能力评估是对软件开发能力的高低进行衡量；软件配置管理是针对开发过程中人员、工具的配置、使用等提出的管理策略。下面就部分内容展开讨论。

**2. 软件项目管理的原则**

从软件工程的角度讲，软件开发主要分为6个阶段：需求分析阶段、概要设计阶段、详细设计阶段、编码阶段、测试阶段、安装及维护阶段。不论是作坊式开发，还是团队协作开发，这6个阶段都是不可缺少的。根据公司实际情况，公司在进行软件项目管理时，重点将软件配置管理、项目跟踪和控制管理、软件风险管理及项目策划活动管理4方面内容导入软件开发的整个阶段。在20世纪80年代初，著名软件工程专家B. W. Boehm总结出了软件开发时需遵循的7条基本原则，同样，在进行软件项目管理时，也应该遵循这7条原则。它们是：

（1）用分阶段的生命周期计划严格管理。

（2）坚持进行阶段评审。

（3）实行严格的产品控制。

（4）采用现代程序设计技术。

（5）结果应能够清楚地审查。

（6）开发小组的人员应该少而精。

（7）承认不断改进软件工程实践的必要性。

**3. 人员组织与管理**

对于整个项目来讲，软件开发过程中的开发人员是最大的资源。对人员的配置、调度安排贯穿整个软件过程，人员的组织管理是否得当，是影响软件项目质量的决定性因素。

首先在软件开发的一开始，要合理地配置人员，根据项目的工作量、所需要的专业技能，

再参考各个人员的能力、性格、经验，组织一个高效、和谐的开发小组。一般来说，一个开发小组人数为5～10人最为合适，如果项目规模很大，可以采取层级式结构，配置若干这样的开发小组。

在选择人员的问题上，要结合实际情况来决定是否选入一个开发组员。并不是一群高水平的程序员在一起就一定可以组成一个成功的小组。作为考察标准，技术水平、与本项目相关的技能和开发经验，以及团队工作能力都是很重要的因素。一个一天能写一万行代码但却不能与同事融洽沟通的程序员，未必适合一个对组员之间通信要求很高的项目。还应该考虑分成几个小的开发项目组，小组中有页面美工、后台服务程序、数据库几个部分，应该合理地组织各项工作的人员匹配。例如，对于一个中型农技110网站，对数据采集量要求较高，一个人员匹配方案可以是两个美工、两个后台服务程序编写人员、三个数据采集整理人员。

可以用如下公式来对候选人员能力进行评分，达到一定分数的则可以考虑进入开发组，但这个公式不包含对人员数量配比的考虑。

$$\text{Score} = \sum_{i=1}^{8} w_i \times C_i$$

$C_i$ 是对项目组人员各项能力的评估。其值的含义如表5.17所示。

**表5.17　$C_i$取值含义**

| $C_i$ 的取值 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 含义 | 该人此项能力很差，完全没有相关经验 | 有一定此项能力，或曾从事过少量相关工作 | 此项能力较好，或有较多相关项目经验 | 能力优秀，有丰富的同类项目开发经验 |

$W_i$ 是权重值，对应 $C_i$ 描述的能力在本项目中的重要性，其值的含义如表5.18所示。

**表5.18　$W_i$取值含义**

| $W_i$ 的取值 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 含义 | 本项目中不要求此项能力，或此项目能力对目前的候选人来说都是认定满足的 | 本项目对此项能力有一定要求，但不作为普遍要求 | 此项能力在本项目中比较重要，要求所有人员都要达到一定的水准 | 此项能力在本项目中非常重要，所有人员都必须达到比较好的水准 |

在决定一个开发组的开发人员数量时，除了考虑候选人素质以外，还要综合考虑项目规模、工期、预算、开发环境等因素的影响。在组建开发组时，还应充分估计到开发过程中的人员风险。由于工作环境、待遇、工作强度、公司的整体工作安排和其他无法预知的因素，一个项目尤其是开发周期较长的项目几乎无可避免地要面临人员的流入流出。如果不在项目初期对可能出现的人员风险进行充分的估计，做必要的准备，一旦风险转化为现实，将有可能给整个项目开发造成巨大的损失。以较低的代价进行及早的预防是降低这种人员风险的基本策略。具体来说，可以从以下几方面对人员风险进行控制。

(1) 保证开发组中全职人员的比例，且项目核心部分的工作应该尽量由全职人员来担任，以减少兼职人员对项目组人员不稳定性的影响。

(2) 建立良好的文档管理机制，包括项目组进度文档、个人进度文档、版本控制文档、整体技术文档、个人技术文档、源代码管理等。一旦出现人员的变动，如某个组员因病退出，替

补的组员能够根据完整的文档尽早接手工作。

(3) 加强项目组内技术交流,如定期召开技术交流会,或者根据组内分工情况建立项目组内部的开发小组,使开发小组内的成员能够相互熟悉对方的工作和进度,能够在必要的时候替对方工作。

(4) 对于项目经理,可以从一开始就指派一个副经理在项目中协同项目经理管理项目开发工作,如果项目经理退出开发组,副经理可以很快接手。但是只建议在项目经理这样的高度重要的岗位采用这种冗余复制的策略来预防人员风险,否则将大大增加项目成本。

(5) 为项目开发提供尽可能好的开发环境,包括工作环境、待遇、工作进度安排等,同时一个优秀的项目经理应该能够在项目组内营造一种良好的人际关系和工作氛围。良好的开发环境对于稳定项目组人员以及提高生产效率都有不可忽视的作用。

**4. 计划管理**

软件项目计划是一个软件项目进入系统实施的启动阶段,主要进行的工作包括:确定详细的项目实施范围、定义递交的工作成果、评估实施过程中主要的风险、制定项目实施的时间计划、成本和预算计划、人力资源计划等。

软件项目管理过程是以项目计划活动为起点,而第一项计划活动就是估算:估算需要多长时间、估算需要多少工作量,以及估算需要多少人员。此外,还必须估算所需要的资源(硬件及软件)和可能涉及的风险。

为了估算软件项目的工作量和完成期限,首先需要预测软件的规模。度量软件规模的常用方法有直接的方法——LOC(代码行),间接的方法——FP(功能点)。这两种方法各有优缺点,应该根据软件项目的特点选择适用的软件规模度量方法。

根据项目的规模可以估算出完成项目所需的工作量,可以使用一种或多种技术进行估算,这些技术主要分为两大类:分解和经验建模。分解技术需要划分出主要的软件功能,接着估算实现每一个功能所需的程序规模的人/月数。经验技术的使用是根据经验导出的公式来预测工作量和时间,可以使用自动工具来实现某一特定的经验模型。

精确的项目估算一般至少会用到上述技术中的两种。通过比较和协调使用不同技术导出的估算值,可能得到更精确的估算。软件项目估算永远不会是一门精确的科学,但将良好的历史数据与系统化的技术结合起来能够提高估算的精确度。

当对软件项目给予较高期望时,一般都要进行风险评估。在标识、分析和管理风险上花费的时间和人力可以从多方面得到回报:更加平稳的项目进展过程;更高的跟踪和控制项目的能力;由于在问题发生之前已经做了周密计划而产生的信心。

对于一个项目管理者,他的目标是定义所有的项目任务,识别出关键任务,跟踪关键任务的进展情况,以保证能够及时发现拖延进度的情况。为此,项目管理者必须制定一个足够详细的进度表,以便监督项目进度并控制整个项目。

常用的制定进度计划的工具主要有 Gantt 图和工程网络两种。Gantt 图具有悠久历史、直观简明、容易学习、容易绘制等优点。但是,它不能明显地表示各项任务彼此间的依赖关系,也不能明显地表示关键路径和关键任务,进度计划中的关键部分也不明确。因此,在管理大型软件项目时,仅用 Gantt 图是不够的,不仅难于做出既能节省资源又能保证进度的计划,而且容易发生差错。

工程网络不仅能描绘任务分解情况及每项作业的开始时间和结束时间,而且能清楚地

表示各个作业彼此间的依赖关系。从工程网络图中容易识别出关键路径和关键任务。因此，工程网络图是制定进度计划的强有力的工具。通常，联合使用 Gantt 图和工程网络这两种工具来制定和管理进度计划，使它们互相补充、取长补短。

进度安排是软件项目计划的首要任务，而项目计划则是软件项目管理的首要组成部分。与估算方法和风险分析相结合，进度安排将为项目管理者建立起一张计划图。

**5. 软件过程能力评估**

软件过程能力描述了一个开发组织是否具有开发高质量软件产品的能力，现行的国际标准主要有两个：ISO9000.3 和 CMM。

ISO9000.3 是 ISO9000 质量体系认证中关于计算机软件质量管理和质量保证标准部分。它从管理职责、质量体系、合同评审、设计控制、文件和资料控制、采购、顾客提供产品的控制、产品标识和可追溯性、过程控制、检验和试验、检验/测量和试验设备的控制、检验和试验状态、不合格品的控制、纠正和预防措施、搬运/储存/包装/防护和交付、质量记录的控制、内部质量审核、培训、服务、统计系统等 20 个方面对软件质量进行了要求。

CMM(能力成熟度模型)是美国卡耐基·梅隆大学软件工程研究所(CMU/SEI)于 1987 年提出的评估和指导软件研发项目管理的一系列方法，用 5 个不断进化的层次来描述软件过程能力。现在的 CMM 是 2.0 版本。

ISO9000 和 CMM 的共同点是二者都强调了软件产品的质量，所不同的是，ISO9000 强调的是衡量的准则，但没有告诉软件开发人员如何达到好的目标、如何避免差错。CMM 则提供了一整套完善的软件研发项目管理的方法，它可以告诉软件开发组织，如果要在原有的水平上提高一个等级，应该关注哪些问题，而这正是改进软件过程的工作。

CMM 描述了 5 个级别的软件过程成熟度(初始级、可重复级、已定义级、已定量管理级、优化级)，成熟度反映了软件过程能力的大小。

初始级的特点是软件机构缺乏对软件过程的有效管理，软件过程是无序的，有时甚至是混乱的，对过程几乎没有定义，其软件项目的成功来源于偶尔的个人英雄主义而非群体行为，因此它不是可重复的；可重复级的特点是软件机构的项目计划和跟踪稳定，项目过程可控，项目的成功是可重复的；已定义级的特点在于软件过程已被提升成标准化过程，从而更加具有稳定性、可重复性和可控性；已定量管理级的软件机构中软件过程和软件产品都有定量的目标，并被定量地管理，因而其软件过程能力是可预测的，其生产的软件产品是高质量的；优化级的特点是过程的量化反馈和先进的新思想、新技术促进过程不断改进，技术和过程的改进被作为常规的业务活动加以计划和管理。

CMM 是科学评价一个软件企业开发能力的标准，但要达到较高的级别也非常困难，根据 1995 年美国所做的软件产业成熟度的调查，在美国的软件产业中，CMM 成熟度等级为初始级的占 70%，为可重复级的占 15%，为定义级的所占比例小于 10%，为管理级的所占比例小于 5%，为优化级的所占比例小于 1%。而国内企业的水平就更加令人担忧，到目前为止，只有东软一家达到优化级，少数几家能够达到可定义级。尽快改变这种局面，科学化、规范化、高效地进行软件开发活动，整体上提高我国软件行业的水平，是国内软件企业的当务之急，也是专业人员应该为自己制定的目标。如果有一天也能指挥一个数千人的庞大开发队伍，操作 Windows 这样巨型规模的软件项目，并生产出高质量的产品，才有理由宣称自己的软件项目管理能力达到了一个“自主、自足”的水平。

# 小　结

- 程序设计是利用程序解决特定问题的过程，它以某种程序设计语言为工具，给出这种语言下的程序。程序设计过程一般要经过分析、设计、编码、测试和调试等不同阶段。
- 程序设计语言的演化：机器语言→汇编语言→高级语言。机器语言和汇编语言简称为低级语言，主要是针对机器而言，由低级语言编写的程序，计算机能直接识别与运行。而利用高级语言编写的程序独立于计算机类型，具有接近于人类的语言描述问题的特点，易于理解与接受，但不能直接运行于机器上，需通过编译才能让计算机识别。利用高级语言进行构建程序需经过 4 个步骤：编辑→编译→链接→执行。
- 高级程序语言主要分为面向对象与面向过程的程序设计语言，主要由变量、运算符、表达式、各种控制结构组成，同时有适当的注释语句。常见的高级语言有 C 语言、C++语言、Java 语言等。
- 算法是一个有穷规则的集合，其中的规则确定了一个解决某一类型问题的运算序列。算法有 5 个重要特性：有限性、确定性、输入、输出、可行性。算法的描述方法有很多，有自然语言、流程图、N-S 流程图、伪代码等。
- 数据结构有表结构、树结构和图结构这三种主要类型。列表是有序数据的集合，主要有数组和链表两种形式；树结构是一种层次结构，在其中每个元素称为结点，结点间存在父子的关系；图结构是一种复杂度更高的结构，它允许结点间存在任意的关系，没有固定的层次。数据处理操作有查找和排序两种基本操作，查找是寻找特定的数据元素，而排序则是将一组数据按照特定的顺序进行排列。
- 数据库(Database)是按照数据结构来组织、存储和管理数据的仓库；数据管理技术的发展大致划分为以下几个阶段：人工管理阶段、文件系统阶段、数据库系统阶段和高级数据库阶段；数据模型的三要素分别是：数据结构、数据操作和完整性约束条件。
- 关系数据库的三个范式：第一范式，即关系中每个属性都是不可再分的简单项；第二范式，如果关系模式 $R \in 1NF$，且每个非主属性都完全函数依赖于 $R$ 的每个关系键，则称 $R$ 属于第二范式；第三范式，如果关系 $R \in 2NF$，且每个非主属性都不传递依赖于 $R$ 的每个关系键，则称 $R$ 属于第三范式。
- 6 个设计步骤：需求分析、概念设计、逻辑结构设计、物理结构设计、系统实施、运行维护。
- 结构化查询语言包含 6 部分：数据查询语言、数据操纵语言、事务处理语言、数据控制语言(DCL)、数据定义语言、指针控制语言。常用数据库系统有 Oracle、DB2、SQL Server、MySQL。
- 软件的生存周期分为 5 个基本过程、9 个支持过程和 7 个组织过程，其中，开发过程一般又分为 6 个阶段：需求分析阶段、概要设计阶段、详细设计阶段、编码阶段、测试阶段、安装及维护阶段。
- 系统测试有两种测试方法：白盒测试和黑盒测试。白盒测试是全面了解程序内部

逻辑结构的情况下对所有逻辑路径进行的测试，黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。

- 软件文档的编制应贯穿于软件产品开发的每一阶段。
- 软件项目管理的内容主要包括如下几方面：人员组织与管理、计划管理、软件度量、风险管理、软件质量保证、软件过程能力评估、软件配置管理等。软件开发时需遵循7条基本原则。

# 习　　题

**一、单项选择题**

1. 在各类程序设计语言中，相比较而言，(　　)程序的执行效率最高。

A. 汇编语言　　B. 面向对象的语言

C. 面向过程的语言　　D. 机器语言

2. 下列关于程序设计语言的说法正确的是(　　)。

A. 高级语言程序的执行速度比低级语言程序快

B. 高级语言就是自然语言

C. 高级语言与机器无关

D. 计算机可以直接识别和执行高级语言编写的源程序

3. 计算机硬件唯一能直接理解的语言是(　　)。

A. 机器语言　　B. 汇编语言　　C. 高级语言　　D. 面向过程语言

4. 结构化程序设计方法的三种基本结构是(　　)。

A. 程序、返回、处理　　B. 输入、输出、处理

C. 顺序、选择、循环　　D. I/O、转移、循环

5. 以下不是面向对象思想中的主要特征的是(　　)。

A. 多态　　B. 继承　　C. 封装　　D. 垃圾回收

6. 数据结构的主要任务是(　　)?

A. 仅用于存储数据　　B. 简单地设计用户接口

C. 优化数据查找速度　　D. 在计算机中有效地组织和操作数据

7. 下面哪一项不属于表结构的类型?(　　)

A. 数组　　B. 链表　　C. 树　　D. 双向链表

8. 下面哪一种树结构中，每个节点最多只能有两个子节点?(　　)

A. 二叉树　　B. B树　　C. 平衡树　　D. 多叉树

9. 在图结构中，基本组成元素主要有哪两种?(　　)

A. 节点和分支　　B. 边和顶点　　C. 表和树　　D. 树和数组

10. 在以下查找与排序算法中，哪种算法适用于已排序的数据集合?(　　)

A. 全部查找　　B. 深度优先搜索　　C. 二分查找　　D. 冒泡排序

11. 在数据管理技术的发展过程中，经历了人工管理阶段、文件系统阶段和数据库系统阶段。在这几个阶段中，数据独立性最高的是(　　)阶段。

A. 数据库系统　　B. 文件系统　　C. 人工管理　　D. 数据项管理

12. 数据库的三级体系结构即子模式，模式与内模式是对(　　)的三个抽象级别。

A. 信息世界　　B. 数据库系统　　C. 数据　　D. 数据库管理系统

13. SQL Server 是(　　)数据库。

A. 关系型　　B. 非关系型　　C. 键值型　　D. 以上都不是

14. 在关系型数据库中，数据是以什么形式组织的？(　　)

A. 树状结构　　B. 网状结构　　C. 表格形式　　D. 图结构

15. 在 E-R 模型中，包含以下基本成分(　　)。

A. 数据、对象、实体　　B. 控制、联系、对象

C. 实体、联系、属性　　D. 实体、属性、操作

16. 软件测试的目的是(　　)。

A. 证明软件的正确性　　B. 找出软件系统中存在的所有错误

C. 证明软件系统中存在错误　　D. 尽可能多地发现软件系统中的错误

17. 使用白盒测试方法时，确定测试数据应根据(　　)和指定的覆盖标准。

A. 程序的内部逻辑　　B. 程序的复杂程度

C. 该软件的编辑人员　　D. 程序的功能

18. 下面几种白箱测试技术，哪种是最强的覆盖准则？(　　)

A. 语句覆盖　　B. 条件覆盖　　C. 判定覆盖　　D. 条件组合覆盖

19. 有关计算机程序功能、设计、编制、使用的文字或图形资料称为(　　)。

A. 软件　　B. 文档　　C. 程序　　D. 数据

20. 好的项目目标应该是(　　)。

A. 全面的而不是具体的　　B. 切合实际，可达到的

C. 非常复杂的　　D. 模糊的

**二、简答题**

1. 简述计算机程序设计语言的分类和各自的特点。

2. 常用的算法描述方法有哪几种？分别用不同的描述方法描述结构化程序的三种基本结构。

3. 软件生存周期可以分为哪几个阶段？每个阶段的提交物是什么？

4. 什么是数据流图？其中的基本符号各表示什么含义？

5. 软件测试包括哪些步骤？

6. 什么是黑盒测试法？常用的黑盒测试方法有哪些？

**三、程序编程**

1. 编写程序，使用一种程序设计语言，由键盘输入三个整数，按照从小到大的顺序输出。

2. 使用一种程序设计语言，从键盘输入 $h$ 值，输出 $h$ 行用 * 号组成的等腰三角形。例如，输入 $h=4$，输出的图形如下。

```
   *
  ***
 *****
*******
```

3. 编写程序计算中国古代数学家张丘健在他的《算经》中提出了著名的“百钱百鸡问题”：鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。翁、母、雏各几何？