

丰富的图示与范例

诠释数据结构

# 图解 数据结构

# —使用 Java

 丰富的图示解释  
复杂的数据结构概念

 以范例程序说明  
数据结构的内涵

 以Java语言实现  
数据结构中的重要理论



提供网络资源下载

胡昭民 编著

清华大学出版社

---

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

**声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。**

图解  
数据结构  
——使用 Java

胡昭民 编著

清华大学出版社  
北京

本书版权登记号：图字 01-2015-2540

本书为荣软科技股份有限公司授权出版发行的中文简体字版本。

## 内 容 简 介

这是一本以 Java 程序语言实战来解说数据结构概念的教材。全书内容浅显易懂，利用大量且丰富的图示与范例，详解复杂的抽象理论，从最基本的的数据结构概念开始说明，再以 Java 工具加以诠释阵列结构、堆栈、链表、队列、排序、查找等重要的概念，引领读者抓住重点轻松进入数据结构的的学习领域。

本书每章重要理论均有范例实现，书中收录了精华的演算法及程序的执行过程，在线阅读或下载附有完整的范例程序源代码，读者可以依照学习进度做练习。除此之外，还有配合各章教学内容的练习题目，以便读者测试自己的学习效果。

本书内容架构完整，逻辑清楚，采用丰富的图例来阐述基本概念及应用，有效提升可读性。以 Java 程序语言实现数据结构中的重要理论，以范例程序说明数据结构的内涵。采用“Eclipse”Java ID 工具，整合编译、执行、测试及除错功能。强调边做边学，结合下载文件，给予最完整的支援。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

### 图书在版编目 (CIP) 数据

图解数据结构：使用 Java / 胡昭民编著. — 北京：清华大学出版社，2015  
ISBN 978-7-302-40299-2

I. ①图… II. ①胡… III. ①数据结构—图解②JAVA 语言—程序设计 IV. ①TP311.12-64②TP312

中国版本图书馆 CIP 数据核字 (2015) 第 113813 号

责任编辑：夏非彼

封面设计：王翔

责任校对：闫秀华

责任印制：刘海龙

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>，<http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969，[c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

质 量 反 馈：010-62772015，[zhiliang@tup.tsinghua.edu.cn](mailto:zhiliang@tup.tsinghua.edu.cn)

印 装 者：北京鑫海金澳胶印有限公司

经 销：全国新华书店

开 本：190mm×260mm 印 张：23.5 字 数：601 千字

版 次：2015 年 8 月第 1 版 印 次：2015 年 8 月第 1 次印刷

印 数：1~3500

定 价：49.00 元

产品编号：063194-01

# 序

数据结构一直是计算机科学领域非常重要的基础课程，它除了是各大专院校信息工程、计算机工程、软件工程、应用数学以及计算机科学等信息相关专业的必修科目外，近年来包括机电、电子或一些商务管理系也列入选修课程。同时，一些信息相关专业的转学考试、研究所考试等，也将数据结构列入必考科目。由此可见，不论从考试的角度，或者研究信息科学的角度，数据结构确实是有志于从事信息工作的专业人员不得不重视的一门基础课程。

要学好数据结构的关键点在于能否找到一本最容易阅读，并将数据结构中各种重要理论、算法等做最详实的解释及举例的图书。市面上以 Java 来实现数据结构理论的书籍比较缺乏，本书是一本如何将数据结构概念用 Java 程序语言实现的重要著作。为了方便学习，书中程序代码都是完整的，可以避免片断学习程序的困扰。另外，本书下载文件中包括提供范例完整的程序代码，方便练习及教学之用。

本书的主要特色在于将比较复杂的理论以图文并茂的形式进行说明，并以最简单的表达方式，将这些数据结构理论加以解释。为了避免在教学及阅读上的不顺畅感，书中的算法尽量不以伪码进行说明，而以 Java 程序语言来展现。同时，书中提到的重要理论，尽量搭配完整的范例程序，便于读者了解以 Java 语言实现这些算法的注意事项。

另外，为了检验读者各章的学习成果，在书中安排了大量的习题，这些题目包含重要考试的考题，以便读者更加灵活应用各种知识。

在附录中提供 Java 的开发环境的简介，本书编辑环境是采用 Eclipse 软件，它是一套 Open Source 的 Java IDE 工具，Eclipse 整合编译、运行、测试及除错功能。

一本好的理论书籍除了内容的完整专业外，更需要清晰易懂的架构安排及表达方式。在仔细阅读本书之后，相信读者会体会笔者的用心，也希望用户能对这门基础学科有更深入、更完整的认识。

本书配套源代码下载地址（注意数字与字母大小写）：<http://pan.baidu.com/s/1bnqCuHD>，若下载有问题，请电子邮件联系 [booksaga@126.com](mailto:booksaga@126.com)，邮件标题为“求代码，图解数据结构：使用 Java”。

作者敬笔

# 目 录

|                                    |    |
|------------------------------------|----|
| 第 1 章 数据结构导论.....                  | 1  |
| 1.1 数据结构简介.....                    | 2  |
| 1.1.1 数据与信息.....                   | 2  |
| 1.1.2 算法.....                      | 3  |
| 1.1.3 算法的条件.....                   | 3  |
| 1.2 认识程序设计.....                    | 5  |
| 1.2.1 程序开发流程.....                  | 5  |
| 1.2.2 数据类型简介.....                  | 6  |
| 1.2.3 结构化程序设计.....                 | 6  |
| 1.2.4 面向对象程序设计.....                | 7  |
| 1.3 算法效能分析.....                    | 9  |
| 1.3.1 时间复杂度.....                   | 9  |
| 1.3.2 Big-oh.....                  | 10 |
| 1.3.3 $\Omega(\text{omega})$ ..... | 11 |
| 1.3.4 $\theta(\text{theta})$ ..... | 12 |
| 1.4 面向对象程序设计与 Java.....            | 12 |
| 1.4.1 类与对象.....                    | 12 |
| 1.4.2 面向对象特性.....                  | 14 |
| 1.4.3 数据封装.....                    | 14 |
| 1.4.4 类继承.....                     | 15 |
| 1.4.5 对象多态.....                    | 17 |
| 1.4.6 抽象类.....                     | 19 |
| 1.4.7 接口.....                      | 20 |
| 本章重点整理.....                        | 22 |
| 本章习题.....                          | 23 |
| 第 2 章 数组结构.....                    | 29 |
| 2.1 线性表.....                       | 30 |
| 2.1.1 线性表定义.....                   | 30 |

|              |                   |           |
|--------------|-------------------|-----------|
| 2.1.2        | 线性表在计算机中的应用 ..... | 30        |
| 2.2          | 认识数组 .....        | 31        |
| 2.2.1        | 一维数组 .....        | 32        |
| 2.2.2        | 二维数组 .....        | 33        |
| 2.2.3        | 三维数组 .....        | 35        |
| 2.2.4        | n 维数组 .....       | 37        |
| 2.2.5        | Arrays 类实现 .....  | 38        |
| 2.3          | 矩阵的简介与运算 .....    | 40        |
| 2.3.1        | 矩阵相加 .....        | 40        |
| 2.3.2        | 矩阵相乘 .....        | 42        |
| 2.3.3        | 转置矩阵 .....        | 45        |
| 2.3.4        | 稀疏矩阵 .....        | 46        |
| 2.3.5        | 上三角形矩阵 .....      | 50        |
| 2.3.6        | 下三角形矩阵 .....      | 55        |
| 2.4          | 数组与多项式 .....      | 60        |
| 2.4.1        | 认识多项式 .....       | 60        |
| 2.4.2        | 多项式的加法 .....      | 60        |
|              | 本章重点整理 .....      | 61        |
|              | 本章习题 .....        | 63        |
| <b>第 3 章</b> | <b>链表 .....</b>   | <b>67</b> |
| 3.1          | 单向链表 .....        | 68        |
| 3.1.1        | 建立单向链表 .....      | 70        |
| 3.1.2        | 单向链表节点的删除 .....   | 74        |
| 3.1.3        | 单向链表的节点插入 .....   | 78        |
| 3.1.4        | 单向链表的反转 .....     | 80        |
| 3.1.5        | 单向链表的串联 .....     | 84        |
| 3.1.6        | 多项式的列表表示法 .....   | 85        |
| 3.2          | 环形链表 .....        | 89        |
| 3.2.1        | 环形链表的定义 .....     | 89        |
| 3.2.2        | 环形链表的节点插入 .....   | 90        |
| 3.2.3        | 环形链表的节点删除 .....   | 90        |
| 3.2.4        | 环形链表的串联 .....     | 93        |
| 3.2.5        | 环形链表表示稀疏矩阵 .....  | 97        |
| 3.3          | 双向链表 .....        | 98        |
| 3.3.1        | 双向链表的定义 .....     | 98        |
| 3.3.2        | 双向链表的节点插入 .....   | 98        |
| 3.3.3        | 双向链表节点删除 .....    | 99        |

|                        |            |
|------------------------|------------|
| 本章重点整理 .....           | 103        |
| 本章习题 .....             | 103        |
| <b>第 4 章 堆栈 .....</b>  | <b>110</b> |
| 4.1 认识堆栈 .....         | 111        |
| 4.1.1 堆栈的运算 .....      | 111        |
| 4.1.2 堆栈的数组实现 .....    | 111        |
| 4.1.3 堆栈的表实现 .....     | 115        |
| 4.2 堆栈的应用 .....        | 118        |
| 4.2.1 汉诺塔问题 .....      | 118        |
| 4.2.2 迷宫问题 .....       | 124        |
| 4.2.3 八皇后问题 .....      | 129        |
| 4.3 算术表达式的求值法 .....    | 132        |
| 4.3.1 中序表示法求值 .....    | 133        |
| 4.3.2 前序表示法求值 .....    | 134        |
| 4.3.3 后序表示法求值 .....    | 135        |
| 4.4 中序法转换为前序法 .....    | 136        |
| 4.4.1 二叉树法 .....       | 136        |
| 4.4.2 括号法 .....        | 136        |
| 4.4.3 堆栈法 .....        | 137        |
| 4.5 前序与后序式转换成中序式 ..... | 143        |
| 4.5.1 括号法 .....        | 143        |
| 4.5.2 堆栈法 .....        | 144        |
| 本章重点整理 .....           | 146        |
| 本章习题 .....             | 147        |
| <b>第 5 章 队列 .....</b>  | <b>155</b> |
| 5.1 认识队列 .....         | 156        |
| 5.1.1 队列的工作运算 .....    | 156        |
| 5.1.2 队列的数组实现 .....    | 156        |
| 5.1.3 以链表实现队列 .....    | 159        |
| 5.2 队列的应用 .....        | 161        |
| 5.2.1 环形队列 .....       | 162        |
| 5.2.2 优先队列 .....       | 165        |
| 5.2.3 双向队列 .....       | 166        |
| 本章重点整理 .....           | 169        |
| 本章习题 .....             | 169        |

|                      |     |
|----------------------|-----|
| 第 6 章 树状结构 .....     | 172 |
| 6.1 树 .....          | 173 |
| 6.2 二叉树简介 .....      | 174 |
| 6.2.1 二叉树的定义 .....   | 175 |
| 6.2.2 特殊二叉树简介 .....  | 176 |
| 6.3 二叉树存储方式 .....    | 177 |
| 6.3.1 数组表示法 .....    | 177 |
| 6.3.2 列表表示法 .....    | 179 |
| 6.4 二叉树的遍历 .....     | 181 |
| 6.4.1 中序遍历 .....     | 182 |
| 6.4.2 前序遍历 .....     | 182 |
| 6.4.3 后序遍历 .....     | 183 |
| 6.4.4 二叉树的遍历实现 ..... | 183 |
| 6.4.5 二叉运算树 .....    | 187 |
| 6.5 二叉树的高级研究 .....   | 192 |
| 6.5.1 二叉排序树 .....    | 192 |
| 6.5.2 二叉搜索树 .....    | 197 |
| 6.5.3 线索二叉树 .....    | 200 |
| 6.6 树的二叉树表示法 .....   | 205 |
| 6.6.1 树转换为二叉树 .....  | 205 |
| 6.6.2 树林转换为二叉树 ..... | 209 |
| 6.6.3 树与树林的遍历 .....  | 211 |
| 6.6.4 确定唯一二叉树 .....  | 214 |
| 本章重点整理 .....         | 216 |
| 本章习题 .....           | 217 |
| 第 7 章 图形结构 .....     | 224 |
| 7.1 图论的起源 .....      | 225 |
| 7.2 图形介绍 .....       | 226 |
| 7.3 图形表示法 .....      | 228 |
| 7.3.1 相邻矩阵法 .....    | 228 |
| 7.3.2 相邻表法 .....     | 232 |
| 7.3.3 相邻多元列表法 .....  | 236 |
| 7.3.4 索引表格法 .....    | 237 |
| 7.4 图形的遍历 .....      | 239 |
| 7.4.1 先深后广法 .....    | 240 |
| 7.4.2 先广后深法 .....    | 243 |
| 7.5 生成树 .....        | 246 |

|              |                   |            |
|--------------|-------------------|------------|
| 7.6          | MST 生成树 .....     | 248        |
| 7.6.1        | Prim 算法 .....     | 249        |
| 7.6.2        | Kruskal 算法 .....  | 250        |
| 7.7          | 图形最短路径 .....      | 255        |
| 7.7.1        | 单点对全部顶点 .....     | 256        |
| 7.7.2        | 顶点两两之间的最短距离 ..... | 259        |
| 7.8          | AOV 网络与拓扑排序 ..... | 263        |
| 7.8.1        | AOV 网络简介 .....    | 264        |
| 7.8.2        | 拓扑排序实现 .....      | 264        |
| 7.8.3        | AOE 网络 .....      | 266        |
|              | 本章重点整理 .....      | 268        |
|              | 本章习题 .....        | 269        |
| <b>第 8 章</b> | <b>排序 .....</b>   | <b>277</b> |
| 8.1          | 排序简介 .....        | 278        |
| 8.1.1        | 排序的分类 .....       | 279        |
| 8.1.2        | 排序算法分析 .....      | 279        |
| 8.2          | 内部排序法 .....       | 280        |
| 8.2.1        | 冒泡排序法 .....       | 280        |
| 8.2.2        | 选择排序法 .....       | 285        |
| 8.2.3        | 插入排序法 .....       | 288        |
| 8.2.4        | 希尔排序法 .....       | 290        |
| 8.2.5        | 合并排序法 .....       | 292        |
| 8.2.6        | 快速排序法 .....       | 293        |
| 8.2.7        | 堆积排序法 .....       | 297        |
| 8.2.8        | 基数排序法 .....       | 304        |
| 8.3          | 外部排序法 .....       | 307        |
|              | 本章重点整理 .....      | 316        |
|              | 本章习题 .....        | 317        |
| <b>第 9 章</b> | <b>查找 .....</b>   | <b>323</b> |
| 9.1          | 查找简介 .....        | 324        |
| 9.2          | 常见查找方法 .....      | 324        |
| 9.2.1        | 顺序查找法 .....       | 324        |
| 9.2.2        | 二分查找法 .....       | 326        |
| 9.2.3        | 插值查找法 .....       | 328        |
| 9.2.4        | 斐波那契查找法 .....     | 331        |
| 9.3          | 哈希查找法 .....       | 333        |

|                       |     |
|-----------------------|-----|
| 9.3.1 哈希法简介 .....     | 333 |
| 9.3.2 常见的哈希函数 .....   | 334 |
| 9.3.3 碰撞问题 .....      | 338 |
| 9.3.4 哈希法综合范例 .....   | 342 |
| 本章重点整理 .....          | 345 |
| 本章习题 .....            | 346 |
| <br>                  |     |
| 附录 Java 的开发环境简介 ..... | 353 |

# 第 1 章

# 数据结构导论

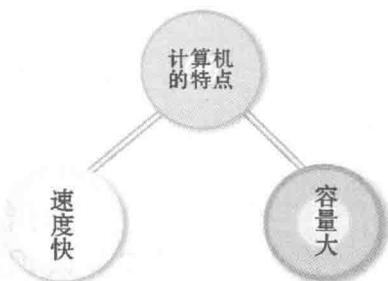
- 1.1 数据结构简介
- 1.2 认识程序设计
- 1.3 算法效能分析
- 1.4 面向对象程序设计与 Java

计算机 (Computer)，或者有人称为计算器 (Calculator)，是一种具备数据处理与计算的电子化设备。它可以接收人类所设计的指令或程序设计语言，经过运算处理后，输出所期待的结果。对于一个有志于从事计算机专业领域的人员来说，数据结构 (Data Structure) 是一门和计算机硬件与软件都相关的学科，其中包含算法 (Algorithm)、数据存储架构、排序、查找、程序设计的概念与哈希函数。

数据结构的研究重点是计算机的程序设计领域，如何将计算机中相关数据的组合，以某种方式组织而成，然后在这样的定义下，就可以探讨各种有意义的操作与关系，以便提高程序的执行效率。

## 1.1 数据结构简介

大家可以将数据结构看成是在数据处理过程中一种分析、组织数据的方法与逻辑，它考虑到了数据间的特性与相互关系。在现代社会中，计算机与信息是息息相关的，因为计算机具有处理速度快与存储容量大两大特点，在数据处理的角色上更为举足轻重，如下图所示。



数据结构无疑就是数据进入计算机内处理的一套完整逻辑，就像程序设计师必须选择一种数据结构来进行数据的新增、修改、删除、存储等操作。因此当我们要求计算机为我们解决问题时，必须以计算机所能接受的模式来确认问题，而安排适当的算法去处理数据，就是数据结构要讨论的重点。

### 1.1.1 数据与信息

谈到数据结构，首先必须了解何谓数据 (Data) 与信息 (Information)。从字面上来看，所谓数据 (Data)，指的就是一种未经处理的原始文字 (Word)、数字 (Number)、符号 (Symbol) 或图形 (Graph) 等，它所表达出来的只是一种没有评估价值的基本元素或项目。例如姓名或我们常看到的课表、通讯录等都可泛称是一种“数据” (Data)。

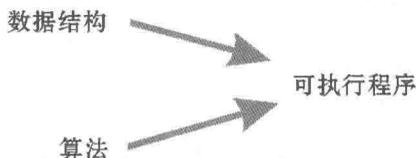
当数据经过处理 (Process)，例如以特定的方式系统地整理、归纳甚至进行分析后，就成为“信息” (Information)。而这样处理的过程就称为“数据处理” (Data Processing)。从严谨的角度来形容“数据处理”，就是用人力或机器设备，对数据进行系统的整理，如记录、排序、合并、整合、计算、统计等，以使原始的数据符合需求，而成为有用的信息。



不过各位可能会有疑问：“那么数据和信息的角色是否绝对一成不变？”。这倒也不一定，同一份文件可能在某种状况下为数据，而在另一种状况下则为信息。例如美伊战争的某场战役死伤人数报告，对你我这些平民百姓而言，当然只是一份不痛不痒的“数据”，不过对于英美联军指挥官而言，这份情报可就是弥足珍贵的“信息”。

### 1.1.2 算法

数据结构与算法是程序设计实践中最基本的内涵。程序能否快速而有效地完成预定的任务，取决于是否选对了数据结构，而程序是否能清楚而正确地把问题解决，则取决于算法。所以各位可以这么认为：“数据结构加上算法等于可执行的程序。”



不过在韦氏辞典中算法却定义为：“在有限步骤内解决数学问题的程序。”如果运用在计算机领域中，我们也可以把算法定义成：“为了解决某项工作或某个问题，所需要有限数目的机械性或重复性指令与计算步骤。”其实日常生活中有许多工作都可以利用算法来描述，例如员工的工作报告、宠物的饲养过程、学生的课程表等。

### 1.1.3 算法的条件

当认识了算法的定义后，我们还要说明算法所必须符合的 5 个条件，如下表所示。

| 算法特性                | 内容与说明                     |
|---------------------|---------------------------|
| 输入 (Input)          | 0 个或多个输入数据，这些输入必须有清楚描述或定义 |
| 输出 (Output)         | 至少会有一个输出结果，不可以没有输出结果      |
| 明确性 (Definiteness)  | 每一个指令或步骤必须是简洁明确的          |
| 有限性 (Finiteness)    | 在有限步骤后一定会结束，不会产生无限循环      |
| 有效性 (Effectiveness) | 步骤清楚且可行，能让用户用纸笔计算而求出答案    |

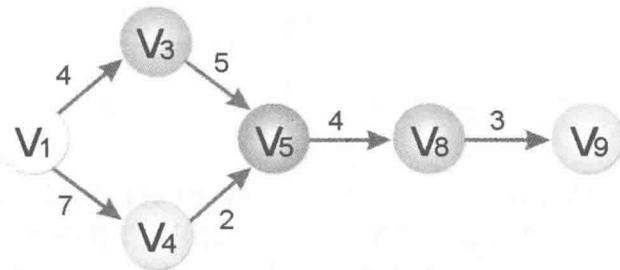
接着各位要来思考：该用什么方法来表达算法最为适当。其实算法的主要目的在于为人们提供阅读了解所执行的工作流程与步骤，只要能清楚表现算法的 5 项特性即可。常用的算法如下。

- 一般文字叙述：中文、英文、数字等。文字叙述法的特色在于使用文字来说明演算步骤。
- 伪语言 (Pseudo-Language)：接近高级程序设计语言的写法，也是一种不能直接放进计算机中执行的语言。一般都需要一种特定的预处理器 (Preprocessor)，或者用手写转换成真正的计算机语言，经常使用的有 SPARKS、Pascal-LIKE 等语言。以下是用 SPARKS 写成的链表反转的算法：

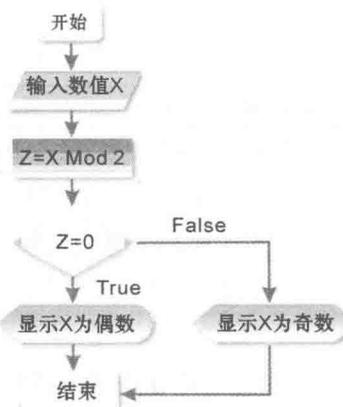
```

Procedure Invert(x)
    P←x;Q←Nil;
    WHILE P≠NIL do
    r←q;q←p;
    p←LINK(p);
    LINK(q)←r;
    END
    x←q;
END
  
```

- 表格或图形：如数组、树形图、矩阵图等。



- 流程图 (Flow Diagram)：是一种通用的图型符号表示法。例如请您输入一个数值，并判断是奇数还是偶数。



- 程序设计语言：目前算法也能够直接以可读性高级语言来表示，例如 Visual Basic、C、C++、Java。

至此，还要特别为各位说明一点，程序（Program）和算法是有区别的。程序中允许无限循环的存在，如一般操作系统中的“作业调度器”（Job Scheduler），在启动后，除非关机或产生例外状况，不然会一直处于执行等待循环。但算法却必须是有限的，这是两者之间的最大不同。

## 1.2 认识程序设计

在数据结构中，所探讨的目标就是将算法朝有效率、可读性高的程序设计方向努力。简单地讲，数据结构与算法必须通过程序（Program）的转换，才能真正由计算机系统来执行。而程序设计的目的就是通过程序的编写与执行来达到用户的需求。或许各位读者认为程序设计的主要目的只是要“执行”出正确的结果，而忽略了执行效率或者日后的维护成本，其实这是不清楚程序设计真正意义的表现。

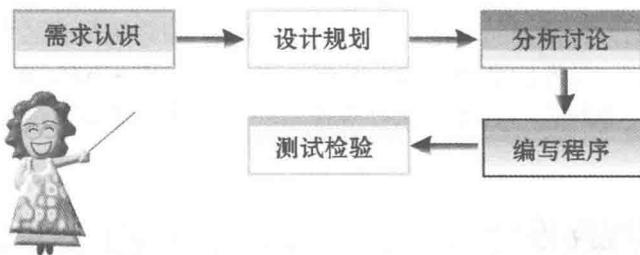
### 1.2.1 程序开发流程

至于程序设计时必须利用何种程序设计语言，通常可根据主客观环境的需要确定，并无特别规定。一般评判程序设计语言好坏的四项原则如下。

- 可读性（Readability）高：阅读与理解都相当容易。
- 平均成本低：成本考虑不局限于编码的成本，还包括执行、编译、维护、学习、调试与日后更新等成本。
- 可靠度高：所编写出来的程序代码稳定性高，不容易产生边际错误（Side Effect）。
- 可编写性高：对于针对需求所编写的程序相对容易。

对于程序设计领域的学习方向而言，无疑就是以有效率、可读性高的程序设计成果为目标。一个程序的产生过程，则可分为以下 5 个设计步骤。

- 步骤 1：需求认识（Requirements）：了解程序所要解决的问题是什么，有哪些输入及输出等。
- 步骤 2：设计规划（Design and Plan）：根据需求选择适合的数据结构，并以任何的表示方式写一个算法以解决问题。
- 步骤 3：分析讨论（Analysis and Discussion）：思考其他可能适合的算法及数据结构，最后再选出最适当的目标。
- 步骤 4：编写程序（Coding）：把分析的结论写成初步的程序代码。
- 步骤 5：测试检验（Verification）：最后必须确认程序的输出是否符合需求，这个步骤需要执行程序并进行许多的相关测试。



程序设计的 5 大步骤

## 1.2.2 数据类型简介

当您进行程序设计时，除了算法的设计外，首先必须挑选一种程序设计语言。要了解程序设计语言的重点，除了语法及语义外，最重要的就是数据类型（Data Type）。所谓数据类型就是程序设计语言的变量（variable）所能表示的数据种类。又因为存储层次上的不同可分为 3 种。

### ▲ 基本数据类型（atomic data type）

基本数据类型或称为物理数据类型（physical data type），也就是一个基本的数据实体，例如一般程序设计语言中的整数、实数、字符等。基本上，每种语言都拥有略微不同的基本数据类型，例如 C 语言的基本数据类型为整数（int）、字符（char）、单精度浮点数（float）与双精度浮点数（double）。

### ▲ 结构型数据类型（structure data type）

结构型数据类型或称为虚拟数据类型（virtual data type），比物理数据类型更高级，是指一个数据结构包含其他的数据类型，例如字符串（string）、集合（set）、数组（array）。

### ▲ 抽象数据类型（Abstract Data Type, ADT）

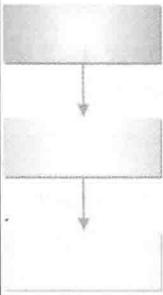
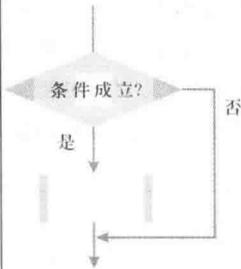
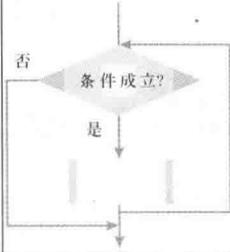
抽象数据类型比结构型数据类型更高级，ADT 是指定义一些结构型数据类型所具备的数学运算关系，用户无须考虑 ADT 的制作细节，只要知道如何使用即可。也就是说，只针对数据的运算，而非数据本身的性质，例如某个数据对象可以插入一个列表，或在列表中增删，而不需关心这个对象的类型是字符串、整数、实数还是逻辑值。通常出现在面向对象程序设计语言（OOP）中的堆栈（stack）或队列（queue）就是一种很典型的 ADT 模式。

## 1.2.3 结构化程序设计

在传统程序设计的方法中，主要以“由下而上”与“由上而下”方法为主。所谓“由下而上”是指程序设计师将整个程序需求中最容易的部分先编写，再逐步扩大来完成整个程序。而“由上而下”则是将整个程序需求从上而下、由大到小逐步分解成较小的单元，或称为“模块”（Module），这样使得程序设计师可针对各模块分别开发，不但可减轻设计者负担、可读性较高，也便于日后维护。而结构化程序设计的核心精神，就是“由上而下设计”与“模块化设计”。例如在 Pascal 语言中，这些模块称为“过程”（Procedure），而 Java 中称为“函数”（Function）。

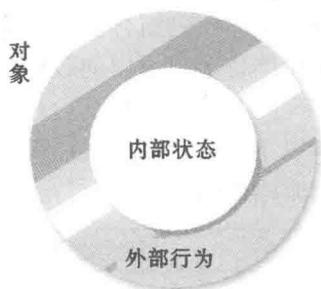
每一个模块会完成特定的功能，主程序则组合每个模块后，完成最后要求的功能。不过一旦主程序要求的功能变动时，则可能许多模块内的数据与算法都需要同步变动，而这也是面向过程的设计无法有效使用程序代码的主要原因。

通常“结构化程序设计”具有以下三种控制流程，对于一个结构化程序，不管其结构如何复杂，都可利用以下的基本控制流程来加以表达。

| 流程结构名称                    | 概念示意图                                                                               |
|---------------------------|-------------------------------------------------------------------------------------|
| 顺序结构：逐步编写程序语句             |    |
| 选择结构：根据某些条件做逻辑判断          |   |
| 重复结构：根据某些条件决定是否重复执行某些程序语句 |  |

#### 1.2.4 面向对象程序设计

“面向对象程序设计”（Object-Oriented Programming, OOP）是近年来相当流行的一种新兴程序设计理念。它主要让我们在设计程序时，能以一种更生活化、可读性更高的设计概念来进行，并且所开发出来的程序也较容易扩充、修改及维护。例如在现实生活中充满了形形色色的物体，每个物体都可视为一种对象。我们可以通过对象的外部行为（behavior）及内部状态（state）模式，来进行详细的描述。行为代表此对象对外所显示出来的运行方法，状态则代表对象内部各种特征的目前状况，如下图所示。



如果要使用程序设计语言方式来描述一个对象，就必须进行所谓的抽象化动作（abstraction）。也就是利用程序代码来记录此对象的属性、方法与事件，如下表所示。

| 名称 | 特色与说明                                                                                              |
|----|----------------------------------------------------------------------------------------------------|
| 属性 | 属性（attribute）是指对象的静态外观描述，例如一辆车子的颜色、大小等，或是对抽象的内在于（如车子引擎的马力、排气数等）描述，就类似于 Java 程序中的类成员数据（member data） |
| 方法 | 方法（method）是指对象中的动态响应方式，例如车子可以开动、停止，就是一种行为模式，用来代表一个对象的功能，也就是 Java 中的类成员方法（member method）            |
| 事件 | 事件（event）是指对象可以针对外部事件做出各种反应，譬如车子没油时，引擎就会停止，当然对象也可以主动地发出事件信息。例如 Java 程序中的窗口组件就可以对事件做出反应与处理          |

面向对象程序设计还具备以下三种特性。



面向对象程序设计的三种特性

### ▲ 封装（Encapsulation）

封装就是利用“类”来实现“抽象数据类型”（ADT）。所谓“抽象”，就是将代表事物特征的数据隐藏起来，并定义一些方法来作为操作这些数据的接口，让用户只能接触到这些方法，而无法直接使用数据，也符合了信息隐藏的意义，而这种自定义的数据类型就称为“抽象数据类型”。

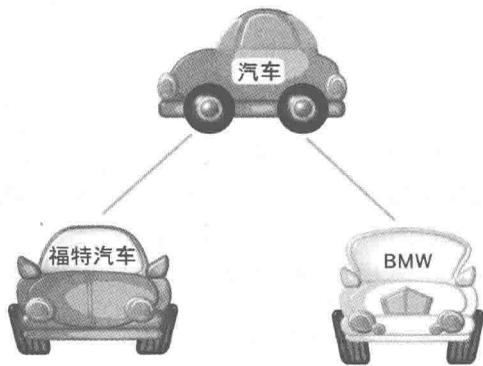
每个类都有其数据成员与函数成员，我们可将其数据成员定义为私有的（private），而将来运算或操作数据的函数成员定义为公有的（public）或受保护的（protected）来实现信息隐藏的功能，这就是“封装”（encapsulation）的作用。

### ▲ 继承（Inheritance）

“继承”接近现实生活中的遗传，例如你的父母生下你，那么你一定会遗传到父母的某

些特征，当面向对象技术以这种生活实例去定义其功能时，则称为“继承”。

在继承关系中，被继承者称为“基类”或“父类”，而继承者则称为“派生类”或“子类”。继承允许我们去定义一个新的类来继承既存的类，进而使用或修改继承而来的方法，并可在子类中加入新的数据成员与函数成员。



<遗传关系图：子类（福特汽车与 BMW）与父类（汽车）>

### ▲ 多态 (Polymorphism)

“多态”是面向对象设计的重要特性，也称为“同名异式”。“多态”的功能可让软件在开发和维护时，达到充分的延伸性。简单地说，多态最直接的定义就是让具有继承关系的不同类对象，可以调用相同名称的成员函数，并产生不同的反应结果。

## 1.3 算法效能分析

对一个程序（或算法）效能的评估，经常是从时间与空间两种因素来进行考虑。时间方面是指程序的运行时间，称为“时间复杂度”（Time Complexity）。空间方面则是此程序在计算机内存所占的空间大小，称为“空间复杂度”（Space Complexity）。



提示

所谓“空间复杂度”是一种以概量精神来衡量所需要的内存空间。而这些所需要的内存空间，通常可以分为“固定空间内存”（包括基本程序代码、常数、变量等）与“变动空间内存”（随程序或进行时而改变大小的使用空间，例如引用类型变量）。

由于计算机硬件发展的日新月异及涉及所使用计算机的不同，所以纯粹从程序（或算法）的效率角度来看，应该以算法的运行时间为主要评估与分析的依据。

### 1.3.1 时间复杂度

程序设计师可以就某个算法的执行步骤计数来衡量运行时间，但是同样是两行指令：

```
a=a+1
```

与

```
a=a+0.3/0.7*10005
```

由于涉及变量存储类型与表达式的复杂度，所以真正绝对精确的运行时间一定不相同。不过话说回来，如此大费周章地去考虑程序的运行时间往往寸步难行，而且毫无意义。这时可以利用一种“概量”的概念来衡量运行时间，我们称为“时间复杂度”（time complexity），其详细定义如下：

在一个完全理想状态下的计算机中，我们定义  $T(n)$  来表示程序执行所要花费的时间，其中  $n$  代表数据输入量。当然程序的运行时间（Worse Case Executing Time）或最大运行时间是时间复杂度的衡量标准，一般以 Big-oh 表示。由于分析算法的时间复杂度必须考虑它的成长比率（Rate of Growth），往往是一种函数，而时间复杂度本身也是一种“渐近表示”（Asymptotic Notation）。

### 1.3.2 Big-oh

$O(f(n))$  可视为某算法在计算机中所需运行时间不会超过某一常数倍的  $f(n)$ ，也就是说当某算法的运行时间  $T(n)$  的时间复杂度（time complexity）为  $O(f(n))$ （读成 big-oh of  $f(n)$  或 order is  $f(n)$ ），意思是存在两个常数  $c$  与  $n_0$ ，则若  $n \geq n_0$ ，则  $T(n) \leq cf(n)$ ， $f(n)$  又称为运行时间的成长率。请各位看以下范例，以了解时间复杂度的意义。

**范例** ▶ 1.3.1: 假如运行时间  $T(n) = 3n^3 + 2n^2 + 5n$ ，求时间复杂度。

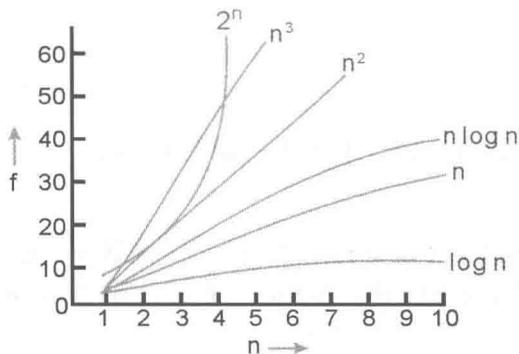
**解答** ▶

首先得找出常数  $c$  与  $n_0$ ，我们可以找到当  $n_0 = 0$  时， $c = 10$ ，则当  $n \geq n_0$  时， $3n^3 + 2n^2 + 5n \leq 10n^3$ ，因此得知时间复杂度为  $O(n^3)$ 。

#### ▲ 常见 Big-oh

事实上，时间复杂度只是执行次数的一个概略的量度层级，并非真实的执行次数。而 Big-oh 则是一种用来表示最坏运行时间的表现方式，它也是最常用于描述时间复杂度的渐近式表示法。常见的 Big-oh 有下列几种。

| Big-oh          | 特色与说明                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------------|
| $O(1)$          | 称为常数时间（constant time），表示算法的运行时间是一个常数倍                                                            |
| $O(n)$          | 称为线性时间（linear time），执行的时间会随数据集合的大小而线性增长                                                          |
| $O(\log_2 n)$   | 称为次线性时间（sub-linear time），成长速度比线性时间还慢，而比常数时间还快                                                    |
| $O(n^2)$        | 称为平方时间（quadratic time），算法的运行时间会成二次方的增长                                                           |
| $O(n^3)$        | 称为立方时间（cubic time），算法的运行时间会成三次方的增长                                                               |
| $O(2^n)$        | 称为指数时间（exponential time），算法的运行时间会成 2 的 $n$ 次方增长。例如解决 Nonpolynomial Problem 问题算法的时间复杂度即为 $O(2^n)$ |
| $O(n \log_2 n)$ | 称为线性乘对数时间，介于线性及二次方增长的中间行为模式                                                                      |



对于  $n \geq 16$  时, 时间复杂度的优劣比较如下:

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

**范例** ▶ 1.3.2: 确定下面的时间复杂度 ( $f(n)$  表示执行次数)。

- (a)  $f(n) = n^2 \log n + \log n$
- (b)  $f(n) = 8 \log \log n$
- (c)  $f(n) = \log n^2$
- (d)  $f(n) = 4 \log \log n$
- (e)  $f(n) = n/100 + 1000/n^2$
- (f)  $f(n) = n!$

**解答** ▶

- (a)  $f(n) = (n^2 + 1) \log n = O(n^2 \log n)$
- (b)  $f(n) = 8 \log \log n = O(\log \log n)$
- (c)  $f(n) = \log n^2 = 2 \log n = O(\log n)$
- (d)  $f(n) = 4 \log \log n = O(\log \log n)$
- (e)  $f(n) = n/100 + 1000/n^2 \leq n/100$  (当  $n \geq 1000$  时)  $= O(n)$
- (f)  $f(n) = n! = 1 * 2 * 3 * 4 * 5 * \dots * n \leq n * n * n * \dots * n \leq n^n$  ( $n \geq 1$  时)  $= O(n^n)$

### 1.3.3 $\Omega$ (omega)

$\Omega$  也是一种时间复杂度的渐近表示法, 如果说 Big-oh 是运行时间量度的最坏情况, 那么  $\Omega$  就是运行时间量度的最好状况。以下是  $\Omega$  的定义:

对  $f(n) = \Omega(g(n))$  (读作 big-omega of  $g(n)$ ), 意思是存在常数  $c$  和  $n_0$ , 对所有的  $n$  值而言,  $n \geq n_0$  时,  $f(n) \geq cg(n)$  均成立。例如  $f(n) = 5n + 6$ , 存在  $c = 5$ ,  $n_0 = 1$ , 对所有  $n \geq 1$  时,  $5n + 5 \geq 5n$ , 因此对于  $f(n) = \Omega(n)$  而言,  $n$  就是成长的最大函数。

**范例** ▶ 1.3.3:  $f(n) = 6n^2 + 3n + 2$ , 请利用  $\Omega$  来表示  $f(n)$  的时间复杂度。

**解答** ▶

$f(n) = 6n^2 + 3n + 2$ , 存在  $c = 6$ ,  $n_0 \geq 1$ , 对所有的  $n \geq n_0$ , 使得  $n^2 + 3n + 2 \geq 6n^2$ , 所以  $f(n) =$

$\Omega(n^2)$ 

### 1.3.4 $\theta$ (theta)

$\theta$  是一种比 Big-O 与  $\Omega$  更精确的时间复杂度渐近表示法。

其定义如下：

$f(n) = \theta(g(n))$  (读作 big-theta of  $g(n)$ )，意思是存在常数  $c_1$ 、 $c_2$ 、 $n_0$ ，对所有的  $n \geq n_0$  时， $c_1 g(n) \leq f(n) \leq c_2 g(n)$  均成立。换句话说，当  $f(n) = \theta(g(n))$  时，就表示  $g(n)$  可代表  $f(n)$  的上限与下限。

以  $f(n) = n^2 + 2n$  为例，当  $n \geq 0$  时， $n^2 + 2n \leq 3n^2$ ，可得  $f(n) = O(n)$ 。同理， $n \geq 0$  时， $n^2 + 2n \geq 3n^2$ ，可得  $f(n) = \Omega(n)$ ，所以  $f(n) = n^2 + 2n = \theta(n^2)$ 。

## 1.4 面向对象程序设计 with Java

Java 是一种纯面向对象程序设计 (Object-Oriented Programming, OOP) 语言，程序中所有相关的程序运算或执行操作，都是利用由类所产生的对象来控制。跟 C 语言等其他语言相比，虽然历史较短，但在短时间内急速成长，现在已经被用于许多环境下了。通常在程序执行时，必须要有执行的平台。所谓的平台是一种包括硬件及软件的执行环境，例如操作系统 Windows、Linux 就是一种执行平台，而 Java 的执行平台不局限于硬件执行环境，能达到跨平台的执行效果。

### 1.4.1 类与对象

面向对象程序中最主要的单元就是对象 (Object)。通常对象并不会凭空产生，它必须有一个可以依据的原型 (Prototype)，而这个原型就是一般在面向对象程序设计中所说的“类” (Class)。

在原型规划阶段，或称之为类的设计 (design) 阶段，首先必须考虑到将来产生的对象、所包含的数据。当对象的原型规划完成后，就可以实际地产生出一个可用的对象，通常称这个过程为对象的“实现” (implementation) 阶段。在 Java 中，对象的实现方式如下：

```
类名称对象(变量)名称 = new 构造函数();
```

- new: 按照类构造函数所代表的引用类型，分配内存空间，以建立该类的实体对象。
- 构造函数 (constructor): 用来建立该类的对象，并在建立的同时设定初始值。

通常 Java 声明基本数据类型的变量 (例如整数) 时，会自动分配内存空间，但是类的类型变量在声明时，则必须以 new 指令来分配内存空间，但是这个分配动作并不会为所建立的对象给定初值，如果想要在建立对象的同时给定初值，就必须借助构造函数，这也正是构造函数的主要功能。

使用构造函数时除了必须与类同名外，还不能有任何的返回值。每个类可以有一个以上的构造函数，这些构造函数可以有不同的参数个数或数据类型，即构造函数可以重载

(Overload) 定义。我们将利用下面的程序范例来说明类的设计与对象的声明。

**范例程序** CH01\_01.java

```
01 // ===== Program Description =====
02 // 程序名称: CH01_01.java
03 // 程序目的: 类与对象
04 // =====
05
06 //声明类
07 public class CH01_01
08 { //成员数据
09     private int carLength, engCC, maxSpeed;
10     private String modelName;
11     //构造函数
12     public CH01_01(String name)
13     {
14         carLength = 423;
15         engCC = 3000;
16         maxSpeed = 250;
17         modelName = name;
18     }
19     //类方法
20     public void ShowData()
21     {
22         System.out.println(modelName + "基本数据");
23         System.out.println("车身长度: " + carLength);
24         System.out.println("汽缸 CC 数: " + engCC);
25         System.out.println("最高车速: " + maxSpeed);
26     }
27     public void SetSpeed(int setSpeed)
28     {
29         System.out.println("\n 使用定速器");
30         maxSpeed = setSpeed;
31         System.out.println("定速设定为: " + setSpeed);
32         System.out.println("目前最高车速为: " + maxSpeed);
33     }
34     //主程序
35     public static void main(String args[])
36     {
37         //实现对象
38         CH01_01 BMW318 = new CH01_01("BMW 318i");
39         //调用类方法
40         BMW318.ShowData();
41         BMW318.SetSpeed(160);
42     }
43 }
```

## 执行结果

```

<terminated> CH01_01 [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (2015年4月15日 下午11:11)
BMW 318i基本数据
车身长度: 423
汽缸CC数: 3000
最高车速: 250

使用定速器
定速设定为: 160
目前最高车速为: 160

```

## 1.4.2 面向对象特性

对于 Java 的面向对象，最主要有三种特性，分述如下。

| 名称 | 特色与说明                                                              |
|----|--------------------------------------------------------------------|
| 封装 | 使用类来制作抽象数据类型（Abstract Data Type, ADT），也就是利用类将数据和用来处理数据的方法包装起来      |
| 继承 | 继承是指所谓的派生类能完全地使用基类的成员数据与成员方法                                       |
| 多态 | 多态或称为“同名异式”（Polymorphism），就是让具有继承关系的不同类对象，可以调用相同名称的成员方法，并产生不同的反应结果 |

## 1.4.3 数据封装

所谓的对象数据封装（encapsulation）动作，就是将静态属性数值与动态行为方法包裹于此对象所“引用”（reference）到的类中。主要的目的是避免对象范围以外的程序有任何改动或破坏内部数据的可能。

不过在一般的 Java 程序里，为了应对各种不同性质对象的产生，通常会声明许多不同类型的类。如果某类中的成员数据属于不可变动的数值，那就必须明确告知程序这些数据的访问权限，以避免其他类对象去破坏该数据的完整性。

另外在 Java 语言中，所有的类、数据与方法都可以定义访问权限。在 Java 中利用三种内置关键词 `private`、`protected` 与 `public` 来设定，分别说明如下。

| 关键词名称                  | 特色与说明                                                            |
|------------------------|------------------------------------------------------------------|
| <code>private</code>   | 以 <code>private</code> （私有的）所声明的数据或方法，仅能被同一类中的成员所使用              |
| <code>protected</code> | 以 <code>protected</code> 所声明的数据或方法，只可以在同一个类，或是派生类或同一封装中，作有限的访问动作 |
| <code>public</code>    | 以 <code>public</code> 所声明的各种数据或方法，可以被所有外部程序、类或对象使用               |

定义访问权限的语法，必须在声明成员数据、方法或类之前加入关键词，例如下面的程序声明片段所示：

```

Private int userPassword          //声明整数类型变量 userPassword 为私有数据成员
Protected getPassword()         //声明类方法 getPassword 为受保护成员方法
public class checkPassword       //声明类 checkPassword 为公共类

```

#### 1.4.4 类继承

继承类似于遗传的概念，当面向对象技术以这种生活实例去定义其功能时，则称为继承（inheritance）。这种继承模式在 Java 平台下的声明语法如下：

访问修饰符 class 派生类名称 extends 基类名称

当访问修饰符省略时，表示使用默认的访问方式。它代表的意义为：相同“套件”（package）中的所有类都能访问此类。访问修饰符的关键词如下所示：

| 关键词           | 特色说明             |
|---------------|------------------|
| public（公开的）   | 代表所有类皆可访问此类      |
| abstract（抽象的） | 代表此类不能被实例化（建立对象） |
| final（最终的）    | 代表此类无法再被重新定义（继承） |

当派生类继承基类所有的成员数据与方法后，并不代表派生类就可以直接去访问基类中的所有成员。有关派生类的访问动作，必须依据基类成员的访问修饰符来进行判断。

##### 范例程序 CH01\_02.java

```

01 // ===== Program Description =====
02 // 程序名称： CH01_02.java
03 // 程序目的： 类继承
04 // =====
05
06 //基类
07 class BMW_Serial
08 { //成员数据
09     private int carLength, engCC, maxSpeed;
10     public String modelName;
11     //类方法
12     public void ShowData()
13     {
14         carLength = 423;
15         engCC = 3000;
16         maxSpeed = 250;
17         System.out.println(modelName + "基本数据");
18         System.out.println("车身长度: " + carLength);
19         System.out.println("汽缸 CC 数: " + engCC);
20         System.out.println("最高车速: " + maxSpeed);
21     }
22 }
23 //派生类
24 public class CH01_02 extends BMW_Serial
25 { //构造函数
26     public CH01_02(String name)
27     {

```

```

28     modelName = name;
29     }
30     //主程序区块
31     public static void main(String args[])
32     {
33         //实现对象
34         CH01_02 BMW318= new CH01_02("BMW 318i");
35         BMW318.ShowData();
36     }
37 }

```

### 执行结果



```

<terminated> CH01_02 [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (2015年4月16日 上午7:
BMW 318i基本数据
车身长度: 423
汽缸CC数: 3000
最高车速: 250

```

派生类还可以依据本身需求对所继承的各种方法重新定义（overriding，或称为重写）。此方法和基类的某个 `public` 或 `protected` 方法同名，并且自变量个数、数据类型与方法返回值类型完全相同。例如下面的程序片段所示：

```

class BMW_Motocycle extends BMW_Serial //以继承方式声明类
{
:
public ShowData() //重新定义类成员方法
{
    System.out.println("这是利用 BMW_Motocycle 类重新定义的 ShowData 方法");
}
}

```

另外派生类中也可以根据实际需要，声明一个和基类具有相同的名称，但是具有不同自变量状态的方法（例如自变量个数不同、自变量数据类型不同等）。这种做法称为此方法的“重载”（overloading）。请看以下程序片段及说明：

```

class BMW_Motocycle extends BMW_Serial //以继承方式声明类
{
:
public ShowData() //重新定义类方法
{
    System.out.println("这是利用 BMW_Motocycle 类重新定义的 ShowData 方法");
}
}

```

```

public ShowData(String modelName, int price) //重载类方法
{
    System.out.println("BMW_Motocycle 类重载的 ShowData 方法");
}
}

```

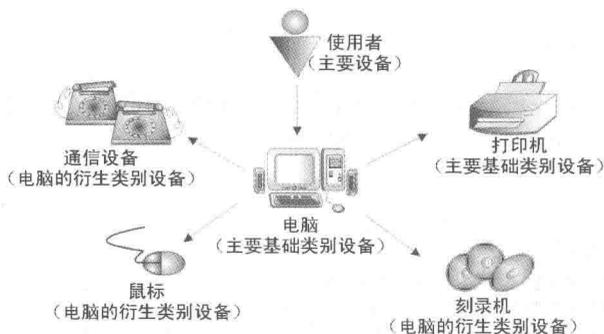
在 Java 的程序结构里，两个名称相同但是拥有不同自变量的方法被视为两种不同方法的存在。最后还要提醒各位一个重要的概念，即当使用派生类构造函数建立对象时，Java 编译程序会先去调用基类的构造函数，然后才执行派生类的构造函数。

### 1.4.5 对象多态

“多态”（polymorphism）利用类继承架构的基础，先建立一个内容为“null”的基类对象，让用户可以将它转变成为各种派生类对象，进而加以控制所有派生类的“同名异式”方法。

通常一个大型的应用程序会由许多的对象共同组成，我们不能奢望在没有任何媒介的情形下，对象与对象之间会自动地达成协调处理。在面向对象的概念中，可以利用“消息”（Message）传递的手法，来完成两个或多个对象间的互动与沟通。

在程序设计过程中，所传递的对象属于何种类，必须在程序运行时（run time）才可以确定。就如同一台计算机上连接了许多的相关设备，等用户“需要使用”的时候，才能确定到底是要用哪项设备，并调用该设备的哪种功能。但是可以确定的是用户能够通过计算机，对所有接口设备下达执行工作命令。



也就是说，我们可以不需要去担心如何跟各种“可能对象”进行信息沟通，直接利用多态机制对基类的对象传递必要的消息即可。以上例来说，用户可以直接命令（传递消息）计算机执行打印、刻录、读取或扫描的动作。下面范例利用多态的方式，来控制各种可能的派生对象。

#### 范例程序 CH01\_03.java

```

01 // ===== Program Description =====
02 // 程序名称: CH01_03.java
03 // 程序目的: 对象多态
04 // =====
05

```

```
06 //基类
07 class myComputer
08 { //类方法
09     public void Run(){};
10 }
11 //派生类一
12 class myScanner extends myComputer
13 { //成员数据
14     private String paperScan;
15     //构造函数
16     public myScanner(String inData){this.paperScan = new String(inData);}
17     //重新定义类方法
18     public void Run(){System.out.println("使用扫描仪扫描" + paperScan + "完成");}
19 }
20 //派生类二
21 class myPrinter extends myComputer
22 { //成员数据
23     private String paperPrint;
24     //构造函数
25     public myPrinter(String inData){this.paperPrint = new String(inData);}
26     //重新定义类方法
27     public void Run(){System.out.println("使用打印机打印" + paperPrint + "完成");}
28 }
29 //主要类
30 public class CH01_03
31 { //成员数据
32     private String inputData;
33     //构造函数
34     public CH01_03()
35     {
36         System.out.println("用户想要先扫描文件 A!!再将文件 A 打印!!");
37         System.out.println("它的流程为: ");
38         inputData = "文件 A";
39     }
40     public static void main(String args[])
41     { //实现对象
42         CH01_03 computerUser = new CH01_03();
43         myComputer MyComputer;
44         myScanner MyScanner = new myScanner("对象 A");
45         myPrinter MyPrinter = new myPrinter("对象 A");
46         System.out.println("将对象"计算机"转变成对象"扫描仪"执行 Run () 方法!!");
47         //实现多态
48         MyComputer = MyScanner;
49         MyComputer.Run();
50         System.out.println("将对象"计算机"转变成对象"打印机"执行 Run () 方法!!");
51         //实现多态
52         MyComputer = MyPrinter;
53         MyComputer.Run();
54     }
55 }
```

## 执行结果



## 1.4.6 抽象类

抽象类（abstract class）至少包含一个完整方法及一到多个抽象方法的“基类”，而所谓抽象方法则是指使用保留字“abstract”来声明，并且不加入任何程序语句的成员方法。它的声明语法如下：

```
abstract class 类名称
{
...
    abstract 返回值的数据类型成员方法(自变量列);
...
}
```

因为抽象基类中含有一到多个抽象方法（abstract method），所以无法直接用来产生对象。为了能顺利地建立各种所需的派生类，用户必须在派生类中定义基类所有的抽象方法。

## 范例程序 CH01\_04.java

```
01 // ===== Program Description =====
02 // 程序名称: CH01_04.java
03 // 程序目的: 抽象类
04 // =====
05
06 //抽象类
07 abstract class autoMobile
08 { //抽象方法
09     abstract public void setData();
10     abstract public void showData();
11 }
12 //派生类
13 class BENZ_Serial extends autoMobile
14 { //成员数据
15     private int carLength, engCC, maxSpeed;
16     //构造函数
17     public BENZ_Serial(String modelName)
18     {
```

```

19     System.out.println("BENZ 系列: " + modelName + "基本数据");
20     }
21     //重新定义抽象方法
22     public void setData()
23     {
24         carLength = 400;
25         engCC = 3200;
26         maxSpeed = 280;
27     }
28     public void showData()
29     {
30         System.out.println("车身长度: " + carLength);
31         System.out.println("汽缸 CC 数: " + engCC);
32         System.out.println("最高车速: " + maxSpeed);
33     }
34 }
35 //主要类
36 public class CH01_04
37 {
38     public static void main(String args[])
39     { //实现抽象类对象
40         autoMobile myCar = null;
41         //实现派生类对象
42         BENZ_Serial SLK2000 = new BENZ_Serial("SLK2000");
43         //实现多态
44         myCar = SLK2000;
45         myCar.setData();
46         myCar.showData();
47     }
48 }

```

### 执行结果

```

<terminated> CH01_04 [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (2015年4月16日 上午7:
BENZ系列: SLK2000基本数据
车身长度: 400
汽缸CC数: 3200
最高车速: 280

```

## 1.4.7 接口

接口 (interfaces) 与抽象类相似, 它们之间最大的差异在于抽象类因为 Java 在类继承上的限制, 一个派生类仅能继承单一基类, 而接口则可以让用户编写出内含多种接口的实现类。或者您可以把这种类型想象成另类“多重类继承”的呈现。另一个差异点在于抽象类至少包含一个完整方法, 而接口所包含的都是抽象方法。

在语法的声明方式上, 接口声明所使用的关键词也不同于类的声明。有关接口的声明语

法如下:

```
interface 接口名称
{
...
返回值的数据类型成员方法();
...
}
```

下面范例说明了接口的实现方式:

#### 范例程序 CH01\_05.java

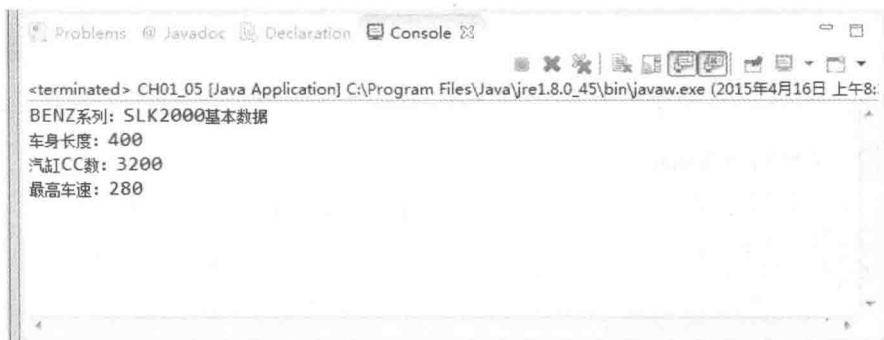
```
01 // ===== Program Description =====
02 // 程序名称: CH01_05.java
03 // 程序目的: 接口操作
04 // =====
05
06 //声明接口一
07 interface autoMobile_setData
08 { //成员方法
09     void setData();
10 }
11 //声明接口二
12 interface autoMobile_showData
13 { //成员方法
14     void showData();
15 }
16 //接口实现类
17 class CH01_05 implements autoMobile_setData, autoMobile_showData
18 { //成员数据
19     int carLength, engCC, maxSpeed;
20     //构造函数
21     public CH01_05(String modelName)
22     {
23         System.out.println("BENZ 系列: " + modelName + "基本数据");
24     }
25     //重新定义抽象方法
26     public void setData()
27     {
28         carLength = 400;
29         engCC = 3200;
30         maxSpeed = 280;
31     }
32     public void showData()
33     {
34         System.out.println("车身长度: " + carLength);
35         System.out.println("汽缸 CC 数: " + engCC);
36         System.out.println("最高车速: " + maxSpeed);
37     }
38     //主程序区块
39     public static void main(String args[])
40     {
41         CH01_05 SLK2000 = new CH01_05("SLK2000");
```

```

42     SLK2000.setData();
43     SLK2000.showData();
44 }
45 }

```

### 执行结果



接口实现在应用上的便利，不只在多重继承，因为接口可以被视为一种类的延伸，所以与抽象类相同，可以利用继承的模式轻易地将各种不同接口的成员方法加以结合，形成一个新的接口形态。除此之外，接口中所有的数据成员不需经过额外声明，都会被自动定义成 `static` 与 `final` 类型，所以接口经常被用来定义程序中所需要的各种常数。

### 本章重点整理

- ❶ 数据结构 (Data Structure) 是一门和计算机硬件与软件都相关的学科。其中包含算法 (Algorithm)、数据存储架构、排序、查找、程序设计概念与哈希函数。
- ❷ 所谓数据 (Data)，指的就是一种未经处理的原始文字 (Word)、数字 (Number)、符号 (Symbol) 或图形 (Graph) 等，它所表达出来的只是一种没有评估价值的基本元素或项目。
- ❸ 当数据经过处理 (Process)，例如以特定的方式系统地整理、归纳甚至分析后，就成为“信息” (Information)。
- ❹ “数据处理”，就是用人力或机器设备，对数据进行系统的整理，如记录、排序、合并、整合、计算、统计等，以使原始的数据符合需求，而成为有用的信息。
- ❺ 我们可以把算法定义成：“为了解决某一个工作或问题，所需要有限数目的机械性或重复性指令与计算步骤。”
- ❻ 算法必须符合的 5 个条件：输入、输出、有限、有效、明确。
- ❼ 常见的算法：一般文字叙述、伪语言 (Pseudo-Language)、表格或图形、流程图、程序设计语言。
- ❽ 数据类型因为存储层次上的不同可分为三种：基本数据类型 (Atomic Data Type)、结构型数据类型 (Structure Data Type)、抽象数据类型 (Abstract Data Type, ADT)。
- ❾ 结构化程序设计的核心精神，就是“由上而下设计”与“模块化设计”。

- ❶ 属性 (attribute) 是指对象的静态外观描述, 例如一辆车子的颜色、大小等。
- ❷ 方法 (method) 是指对象中的动态响应方式, 例如车子可以开动、停止。
- ❸ 事件 (event) 是指对象可以针对外部事件做出各种反应, 譬如车子没油时, 引擎就会停止, 当然对象也可以主动地发出事件信息。
- ❹ 面向对象程序设计还具备以下三种特性: 封装性、继承性、多态性。
- ❺ 所谓“抽象化”, 就是将代表事物特征的数据隐藏起来, 并定义一些方法来作为操作这些数据的接口, 让用户只能接触到这些方法, 而无法直接使用数据。
- ❻ 在继承关系中, 被继承者称为“基类”或“父类”, 而继承者则称为“派生类”或“子类”。
- ❼ 算法效能评估因素: “时间复杂度” (Time Complexity) 和 “空间复杂度” (Space Complexity)。
- ❽  $O(f(n))$  可视为某算法在计算机中所需运行时间不会超过某一常数倍的  $f(n)$ , 也就是说某算法的运行时间  $T(n)$  的时间复杂度 (time complexity) 为  $O(f(n))$  (读成 big-oh of  $f(n)$  或 order is  $f(n)$ )。
- ❾ 在 Java 语言中, 所有的类、数据与方法都可以定义访问权限。在 Java 中利用三种内置关键词 `private`、`protected` 与 `public` 来设定。
- ❿ 抽象类 (abstract class) 至少包含一个完整方法及一至多个抽象方法的“基类”, 而所谓抽象方法则是指使用保留字 “abstract” 来声明, 并且不加入任何内容叙述的成员方法。

## 本章习题

1. 请比较数据与信息两者间的差异。

答: 所谓数据 (Data), 指的就是一种未经处理的原始文字 (Word)、数字 (Number)、符号 (Symbol) 或图形 (Graph) 等, 它所表达出来的只是一种没有评估价值的基本元素或项目。当数据经过处理 (Process), 例如以特定的方式系统地整理、归纳甚至进行分析后, 就成为“信息” (Information)。而这样处理的过程就称为“数据处理” (Data Processing)。

2. 算法必须符合哪 5 项条件?

答:

| 算法特性                | 内容与说明                      |
|---------------------|----------------------------|
| 输入 (Input)          | 0 个或多个输入数据, 这些输入必须有清楚描述或定义 |
| 输出 (Output)         | 至少会有一个输出结果, 不可以没有输出结果      |
| 明确性 (Definiteness)  | 每一个指令或步骤必须是简洁明确而不含糊的       |
| 有限性 (Finiteness)    | 在有限步骤后一定会结束, 不会产生无限循环      |
| 有效性 (Effectiveness) | 步骤清楚且可行, 能让用户用纸笔计算而求出答案    |

3. 何谓伪语言 (Pseudo-Language) ?

答:

这是接近高级程序设计语言的写法, 也是一种不能直接放进计算机中执行的语言。一般都需要一种特定的预处理器 (Preprocessor), 或者用手写转换成真正的计算机语言, 经常使用的有 SPARKS、Pascal-LIKE 等。

4. 简述面向对象程序设计的三项特性。

答: 封装 (Encapsulation)、继承 (Inheritance)、多态 (Polymorphism)。

5. 请计算下列程序的 sum 值。

```
Procedure AAA(n)
  sum ← 0
  x ← 2
while x < n do
  x ← 2 * x
  sum ← sum + 1
end
print sum
end
```

答:

当  $n=2$  时, while 循环不会执行,  $sum=0$

$n=4$  时,  $sum=1$

$n=8$  时,  $sum=2$

$n=16$  时,  $sum=3$

因此, 如果是  $n$  型  $2^i$  式, 则  $sum=i-1$ , 如果输入是  $n$ , 则  $sum$  的值是  $\log_2 n - 1$ 。

6. 请问下列程序片段的循环部分实际执行次数与时间复杂度。

```
for i=1 to n
for j=i to n
for k=j to n
{ end of k Loop }
{ end of j Loop }
{ end of i Loop }
```

答:

我们可利用数学式来计算, 公式如下:

$$\begin{aligned}
& \sum_{i=1}^n \sum_{j=1}^n \sum_{k=j}^n 1 = \sum_{i=1}^n \sum_{j=1}^n (n-j+1) \\
&= \sum_{i=1}^n \left( \sum_{j=1}^n n - \sum_{j=1}^n j + \sum_{j=1}^n 1 \right) \\
&= \sum_{i=1}^n \left( \frac{2n(n-i+1)}{2} - \frac{(n+i)(n-i+1)}{2} \right) + (n-i+1) \\
&= \sum_{i=1}^n \left( \frac{(n-i+1)}{2} \right) (n-i+2) \\
&= \frac{1}{2} \sum_{i=1}^n (n^2 + 3n + 2 + i^2 - 2ni - 3i) \\
&= \frac{1}{2} \left( n^3 + 3n^2 + 2n + \frac{n(n+1)(2n+1)}{6} - n^3 - n^2 - \frac{3n^2 + 3n}{2} \right) \\
&= \frac{1}{2} \left( \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right) \\
&= \frac{n(n+1)(n+2)}{6}
\end{aligned}$$

这个  $\frac{n(n+1)(n+2)}{6}$  就是实际循环执行次数, 且我们知道必定存在  $c, \frac{n(n+1)(n+2)}{6} n_0$  使得

$\leq cn^3$ , 当  $n \geq n_0$  时, 时间复杂度为  $O(n^3)$ 。

7. 请回答下列问题:

- ①何谓基本型数据类型 (Atomic Data Type) ?
  - ②何谓结构型 (Structure) 数据类型?
  - ③请以任一种语言为例, 说明它所提供的数据类型, 哪两种为基本型? 哪一种为结构型?
- 答:

①基本数据类型 (Atomic Data Type): 表示最底层且只包含单的数据值。基本上, 每一种程序设计语言都包含不同的基本数据类型, 例如 C 语言的基本数据类型为整数 (integer)、字符 (character)、单精度浮点数 (fixed-point number) 和双精度浮点数 (double-point number)。

②结构型数据类型 (Structured Data Type): 数据类型的一种定义, 为一个数据结构, 包含其他的数据对象, 例如数组 (array)、字符串 (string)、列表 (list)、集合 (set)、文件 (file) 等。

③例如 C 语言中, 整数 (integer)、字符 (character) 是基本数据类型而数组 (array) 是一种结构型数据类型。

8. 解释下列名词:

- ①  $O(n)$  (Big-Oh of  $n$ )
- ② 切割征服 (Divide and Conquer)
- ③ 抽象数据类型 (Abstract Data Type)
- ④ 尾部递归 (Tail Recursion)

答:

①  $O(f(n))$  可视为某算法在计算机中所需运行时间不会超过某一常数倍的  $f(n)$ , 也就是说

某算法的运行时间  $T(n)$  的时间复杂度 (time complexity) 为  $O(f(n))$  (读成 big-oh of  $f(n)$  或 order is  $f(n)$ )。就是存在两个常数  $c$  与  $n_0$ , 若  $n \geq n_0$ , 则  $T(n) \leq cf(n)$ ,  $f(n)$  又称为运行时间的成长率 (rate of growth)。

② 算法的设计方法之一。如快速排序法 (quick sort), 方法是将问题分成两半, 如此递归进行, 直到小到可以解决问题为主。再将结果用递归方法两两合并, 直到完成。

③ 比结构型数据类型更高级, 是指定义一些结构型数据类型所具备的数学运算关系。也就是说, 用户无须考虑 ADT 的制作细节, 只要知道如何使用即可, 例如堆栈 (stack) 或队列 (queue) 就是一种很典型的 ADT 模式。

④ 利用重复控制结构 (迭代法) 的技巧, 使得具有两个以上递归程序的算法中, 第二个以上的递归, 调用该算法能够省略代码。

9. 试证明若  $f(n) = a_m n^m + \dots + a_1 n + a_0$ , 则  $f(n) = O(n^m)$ 。

答:

$$\begin{aligned} f(n) &\leq \sum_{i=1}^m |a_i| n^i \\ &\leq n^m \sum_{i=0}^m |a_i| n^{i-m} \\ &\leq n^m \sum_{i=0}^m |a_i|, \text{ for } n \geq n \end{aligned}$$

另外我们可以把  $\sum_{i=0}^m |a_i|$  视为常数  $C \Rightarrow f(n) = O(n^m)$

10. 请写一个算法来求取函数  $f(n)$ ,  $f(n)$  的定义如下:

$$f(n) = \begin{cases} n^n & \text{if } n \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

答:

```

Procedure FUNCTION(n)
  if n ≤ 0 then return(0)
  p ← n; q ← n-1
  while q > 0 do
    p ← q * n
    q ← q-1
  end
  return(p)
end

```

11. 请证明  $\sum_{1 \leq i \leq n} i = O(n^2)$

解答:

$$\sum_{1 \leq i \leq n} i = 1+2+3+\dots+n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

又可以找到常数  $n_0=0$ 、 $c=1$ ，当  $n \geq n_0$ ， $\frac{n^2+n}{2} \leq n^2$ ，因此得知时间复杂度为  $O(n^2)$

12. 考虑下列  $x \leftarrow x+1$  的执行次数。

(1)

```

:
x ← x+1
:

```

(2)

```

for i ← 1 to n do
:
x ← x+1
:
end

```

(3)

```

for i ← 1 to n do
:
for j ← 1 to m do
:
x ← x+1
:
End
:
end

```

解答:

(1) 1次; (2)n次; (3) $n*m$ 次。

13. 求下列算法中  $x \leftarrow x+1$  的执行次数及时间复杂度。

```

for i ← 1 to n do
j ← 1
for k ← j+1 to n do
x ← x+1
end
end
end

```

解答:

有关  $x \leftarrow x+1$  这行指令的执行次数，因为  $j \leftarrow i$ ，且  $k \leftarrow j+1$ ，所以可用以下数学式表示，其执行次数为：

$$\sum_{i=1}^n \sum_{k=i+1}^n 1 = \sum_{i=1}^n (n-i) = \sum_{i=1}^n n - \sum_{i=1}^n i = n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2} \quad (\text{次})$$

而时间复杂度为  $O(n^2)$ 。

14. 请决定以下程序片的运行时间：

```
k=100000
while k<>5 do
    k=k DIV 10
end
```

解答：

因为  $k=k \text{ DIV } 10$ ，所以一直到  $k=0$  时，都不会出现  $k=5$  的情况，整个循环为无限循环，运行时间为无限长。

# 第2章

# 数组结构

- 2.1 线性表
- 2.2 认识数组
- 2.3 矩阵的简介与运算
- 2.4 数组与多项式

几乎所有的程序设计语言中，都包含数组（Array）数据结构。一个数组元素可以表示成一个索引和名称，并且存储在相邻的计算机内存中，属于一种典型的线性表，当多个同质性的数据需要处理时，都可以使用数组方式存放数据。

## 2.1 线性表

尚未开始正式介绍数组结构之前，我们先来认识线性表（Linear List）。线性表或称为有序表（Ordered List），是数学概念应用在计算机科学中一种相当基本的数据结构。简单地说，线性表是  $n$  个元素的有限序列（ $n \geq 0$ ），如 26 个英文字母的字母表：A, B, C, D, E, ..., Z 就是一个线性表。

### 2.1.1 线性表定义

基本上，线性表数据元素可以是任何一种类型，不过对于同一线性表的每一个元素都必须属于同一类型，例如 10 个阿拉伯数字的数列：(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) 就是一个线性表，而且序列中元素的数据类型是数字。而有序列表的定义，可以形容如下：

- 有序列表可以是空集合，或者可写成  $(a_1, a_2, a_3, \dots, a_{n-1}, a_n)$ 。
- 存在唯一的第一个元素  $a_1$  与存在唯一的最后一个元素  $a_n$ 。
- 除了第一个元素  $a_1$  外，每一个元素都有唯一的前驱（predecessor），例如  $a_i$  的前驱为  $a_{i-1}$ 。
- 除了最后一个元素  $a_n$  外，每一个元素都有唯一的后继（successor），例如  $a_{i+1}$  是  $a_i$  的后继者。

至于有序列表中的每一元素与相邻元素间还会存在某种关系，例如以下 8 种常见的运算方式：

- 计算列表的长度  $n$ 。
- 取出列表中的第  $i$  项元素来加以修正， $1 \leq i \leq n$ 。
- 插入一个新元素到第  $i$  项， $1 \leq i \leq n$ ，并使得原来的第  $i, i+1, \dots, n$  项，后移变成  $i+1, i+2, \dots, n+1$  项。
- 删除第  $i$  项的元素， $1 \leq i \leq n$ ，并使得第  $i+1, i+2, \dots, n$  项，前移变成第  $i, i+1, \dots, n-1$  项。
- 从右到左或从左到右读取列表中各个元素的值。
- 在第  $i$  项存入新值，并取代旧值， $1 \leq i \leq n$ 。
- 复制列表。
- 合并列表。

### 2.1.2 线性表在计算机中的应用

线性表也可应用在计算机中的数据结构，基本上按照内存存储的方式，可分为以下两种。

### ▲ 静态数据结构 (static data structure)

静态数据结构或称为“密集表”(dense list)，它是一种将有序列表的数据使用连续分配空间(contiguous allocation)来存储的。例如数组类型就是一种典型的静态数据结构。优点是设计时相当简单，读取与修改列表中任一元素的时间都固定。缺点则是删除或加入数据时，需要移动大量的数据。另外静态数据结构的内存分配是在编译时，就必须分配给相关的变量。因此数组在建立初期，必须事先声明最大可能的固定存储空间，容易造成内存的浪费。

### ▲ 动态数据结构 (dynamic data structure)

动态数据结构又称为“链接列表”(linked list, 简称链表)，它是一种将线性表的数据使用不连续存储空间来存储。例如指针类型就是一种典型的动态数据结构。优点是数据的插入或删除都相当方便，不需要移动大量数据。另外动态数据结构的内存分配在执行时才发生，所以不需事先声明，能够充分节省内存。缺点就是设计数据结构时较为麻烦，另外在查找数据时，也无法像静态数据一般可随机读取，必须顺序找到该数据为止。



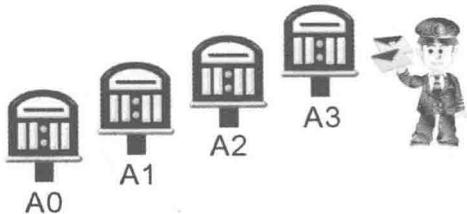
提示

何谓指针变量与动态存储器分配？

“指针变量”(pointer variable)是指内含值为指到内存存储位置的一种数据类型的变量。而“动态存储器分配”(dynamic memory allocation)是指变量存储区分配的过程是在运行(runtime)时，通过操作系统提供可用的内存空间。

## 2.2 认识数组

“数组”(Array)结构其实就是一排紧密相邻的可数内存，并提供一个能够直接访问单一数据内容的计算方法。各位其实可以想象一下自家门前的信箱，每个信箱都有住址，其中路名就是名称，而信箱号码就是索引。邮差可以按照传递信件上的住址，把信件直接投递到指定的信箱中，这就好比程序设计语言中数组的名称是表示一块紧密相邻内存的起始位置，而数组的索引功能则用来表示从此内存起始位置的第几个区块。



在不同的程序设计语言中，数组结构类型的声明也有所差异，但通常必须包含下列5种属性：

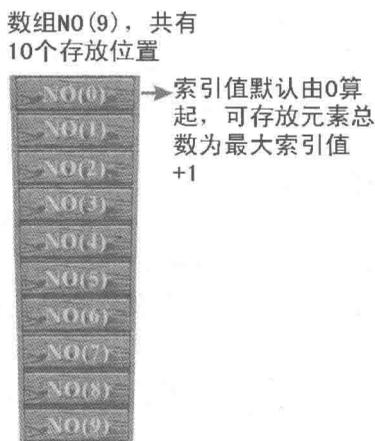
- ❑ 起始地址：表示数组名(或数组第一个元素)所在内存中的起始地址。
- ❑ 维度(dimension)：代表此数组为几维数组，如一维数组、二维数组、三维数组等。
- ❑ 索引上下限：指元素在此数组中，内存所存储位置的上标与下标。
- ❑ 数组元素个数：是索引上限与索引下限的差+1。

- 数组类型：声明此数组的类型，它决定数组元素在内存所占用的大小。

对于任何程序设计语言中的数组表示法（Representation of Arrays），只要符合具备有数组 5 种属性与计算机内存足够的理想情况下，都可能容许  $n$  维数组的存在，接下来我们将更深入地为您逐步介绍各种不同维数数组的详细定义。

## 2.2.1 一维数组

假设  $A$  是一维数组（One-dimension Array）名称，它含有  $n$  个元素，亦即  $A$  是  $n$  个连续内存（各个元素为  $A[0], A[1], \dots, A[n-1]$ ）的集合，并且每个元素的内容为  $a_0, a_1, \dots, a_{n-1}$ 。例如下面是  $NO(9)$  数组的立体示意图：



例如在 Java 语言中一维数组声明方式如下：

```
数据类型 [ ] 变量名称 = new 数据类型 [长度];
```

当 Java 数组声明时会在内存中分配一段暂存空间，如下图所示：



空间的大小以声明的数据类型及数组数量为依据，例如声明  $\text{int}$  类型，数组数量为 10，则数组占内存容量为  $4 \times 10 = 40$  (Byte)。

以上的例子只是 Java 语言中的一维数组声明模式。以下将为您介绍程序设计语言中一维数组的通用表示法。

如果数组  $A$  声明为  $A(l:u_1)$ ，表示  $A$  为含  $n$  个元素的一维数组，其中  $l$  为下标， $u_1$  为上标。则数组元素  $A(1), A(2), A(3), \dots, A(n)$ ， $\alpha$  为此  $A$  数组在内存中的起始位置， $d$  为每一个数组元素所占用的空间，那么数组元素与内存地址有下列关系：

```
 $A(1), A(2), A(3), \dots, A(u_1)$ 
```

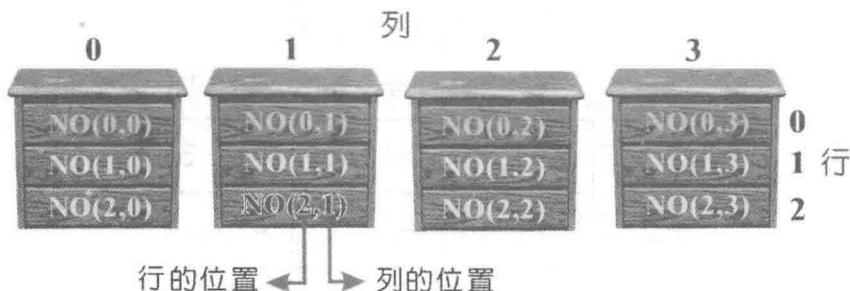
$$a \quad a+1*d \quad a+2*d \quad \dots \quad a+(u_1-1)*d$$

$$\Rightarrow \text{Loc}(A(i)) = a + (i-1)*d \quad (\text{Loc}(A(i)) \text{ 表示 } A(i) \text{ 所在的地址})$$

## 2.2.2 二维数组

二维数组 (Two-dimension Array) 可视为一维数组的扩展, 只不过须将二维转换为一维数组。例如一个含有  $m*n$  个元素的二维数组  $A$ ,  $m$  代表行数,  $n$  代表列数, 请看下面的  $\text{NO}(2,3)$  的二维数组立体示意图。

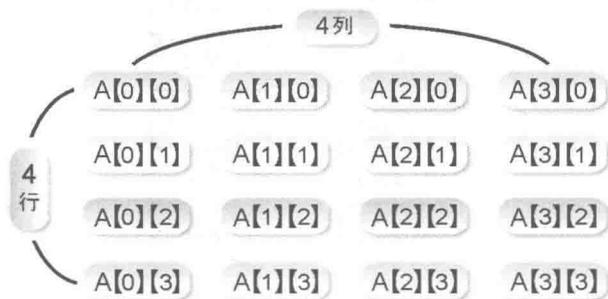
二维数组  $\text{NO}(2, 3)$ , 共有  $(2+1) \times (3+1) = 12$  个存放位置



例如在 Java 语言中二维数组声明方式如下:

```
数据类型 [ ] [ ] 变量名称 = new 数据类型 [第一维长度] [第二维长度];
```

$A[3][3]$  数组中各个元素在直观平面上的排列方式如下:



以上我们将二维数组以矩阵图形表示, 是希望读者能直观且清楚地看到二维数组的内容。当然在实际的计算机内存中是无法以矩阵方式存储的, 必须以线性方式, 视为一维数组的扩展来处理。通常按照不同的语言, 又可区分为两种方式。

### ▲ 以行为主 (Row-major)

例如 Java、C/C++、Pascal 语言的数组存放方式。存储顺序为  $A(1,1), A(1,2), \dots, A(1,n), A(2,1), A(2,2), \dots, A(m-1,n), A(m,n)$ 。假设  $\alpha$  为数组  $A$  在内存中的起始地址,  $d$  为单位空间, 那么数组元素  $A(i,j)$  与内存地址有下列关系:

$$\text{Loc}(A(i, j)) = \alpha + n*(i-1)*d + (j-1)*d$$

### ▲ 以列为主 (Column-major)

例如 Fortran 语言的数组存放方式。存储顺序为  $A(1,1)$ ,  $A(2,1)$ ,  $A(3,1)$ ,  $\dots$ ,  $A(m,1)$ ,  $A(1,2)$ ,  $A(2,2)$ ,  $\dots$ ,  $A(m,n)$ 。假设  $\alpha$  为数组  $A$  在内存中的起始地址,  $d$  为单位空间, 那么数组元素  $A(i,j)$  与内存地址有下列关系:

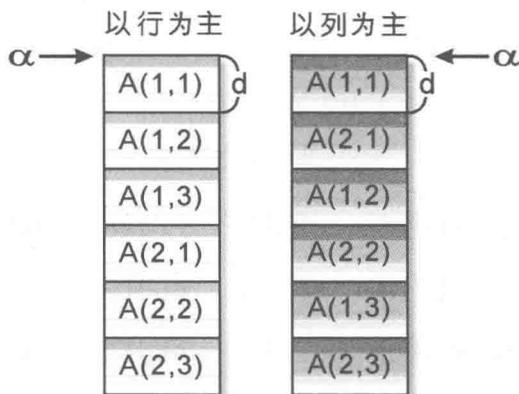
$$\text{Loc}(A(i,j)) = \alpha + (i-1) * d + m * (j-1) * d$$

了解以上的公式后, 我们在此以下图为例进行说明。如果声明数组  $A(1:2,1:3)$ , 则表示法如下:

$A(1:2:1:3)$

|     | 第1列      | 第2列      | 第3列      |
|-----|----------|----------|----------|
| 第1行 | $A(1,1)$ | $A(1,2)$ | $A(1,3)$ |
| 第2行 | $A(2,1)$ | $A(2,2)$ | $A(2,3)$ |

以下是在内存中的实际排列方式:



#### 范例 ▶ 2.2.1

若  $A(3,3)$  在位置 121,  $A(6,4)$  在位置 159, 则  $A(4,5)$  的位置为何? (单位空间  $d=1$ )

#### 解答 ▶

由  $\text{Loc}(A(3,3))=121$ ,  $\text{Loc}(A(6,4))=159$ , 得知数组  $A$  的分配是“以列为主”的方式, 所以起始地址为  $\alpha$ , 单位空间为 1, 则对数组  $A(1:m,1:n)$  有如下推导:

$$\begin{aligned} & \Rightarrow \alpha + (3-1) * 1 + m * (3-1) * 1 \\ & = \alpha + 2 * (1+m) = 121 \Rightarrow \alpha + 2 + 2m = 121 \dots \textcircled{1} \\ & \alpha + (6-1) * 1 + (4-1) * m \\ & = \alpha + 3m + 5 = 159 \Rightarrow \alpha + 3m + 5 = 159 \dots \textcircled{2} \end{aligned}$$

由①, ②式可得 $\alpha=49$ ,  $m=35 \Rightarrow \text{Loc}(A(4,5))=49+4*35+3=192(\#)$

以上计算数组元素地址的方法, 都是以  $A(m,n)$  或写成  $A(1:m,1:n)$  的方式来表示, 这两种方式称为简单表示法, 且  $m$  与  $n$  的起始值一定都是 1, 这里要介绍另一种“注标表示法”。也就是我们可以把数组  $A$  声明成  $A(l_1:u_1, l_2:u_2)$ , 且对任意  $A(i,j)$ , 有  $u_1 \geq i \geq l_1$ ,  $u_2 \geq j \geq l_2$ 。此数组共有  $(u_1-l_1+1)$  行,  $(u_2-l_2+1)$  列。那么地址计算公式和上面以简单表示法有些不同 (因为简单表示法,  $A(m,n)$  可视为  $A(1:m,1:n)$ )。

假设  $\alpha$  仍为起始地址, 而且  $m=(u_1-l_1+1), n=(u_2-l_2+1)$ , 则有下列公式:

#### ▲ 以行为主 (Row-Major)

$$\text{Loc } A(i, j) = \alpha + ((i-l_1+1)-1) * n * d + ((j-l_2+1)-1) * d$$

#### ▲ 以列为主 (Column-Major)

$$\text{Loc } A(i, j) = \alpha + ((j-l_2+1)-1) * m * d + ((i-l_1+1)-1) * d$$

#### 范例 ▶ 2.2.2

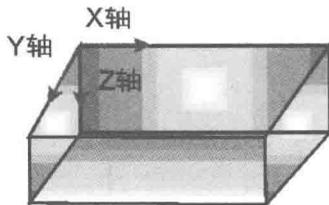
$A(-3:5, -4:2)$  的起始地址  $A(-3, -4)=100$ , 以 row-major 排列, 请问  $\text{Loc}(A(1,1))=?$

#### 解答 ▶

假设  $A$  数组有  $m$  行与  $n$  列, 以 row-major 排列, 且  $\alpha=\text{Loc}(A(-3,-4))=100$ 、 $m=5-(-3)+1=9$  (列)、 $n=2-(-4)+1=7$  (行),  $A(1,1)=100+((1-(-3)+1)-1)*7*1+((1-(-4)+1)-1)*1=133$ 。

### 2.2.3 三维数组

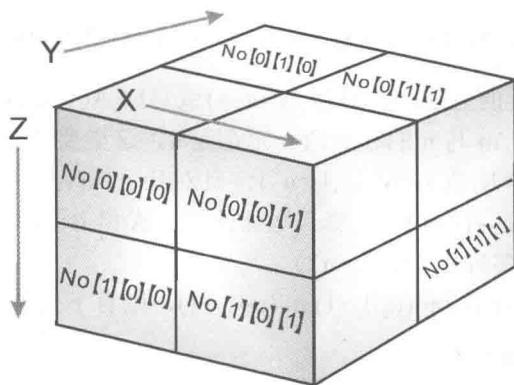
接下来让我们来看看三维数组 (Three-dimension Array), 基本上三维数组的表示法和二维数组一样, 皆视为一维数组的扩展, 如果数组为三维数组, 则可以看作是一个立方体, 如下图所示。



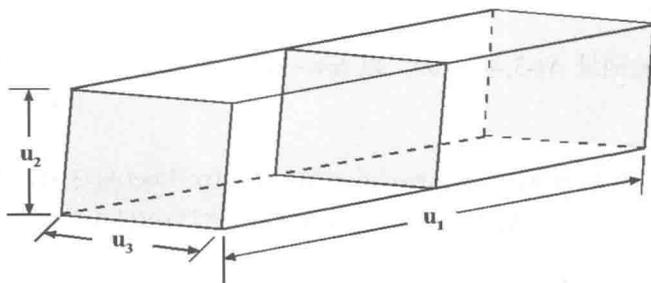
例如在 Java 语言中三维数组声明方式如下:

```
数据类型 [ ] [ ] [ ] 变量名称 = new 数据类型 [第一维长度] [第二维长度] [第三维长度];
```

例如数组  $\text{No}[2][2][2]$  共有 8 个元素, 可以使用立体图形表示如下:

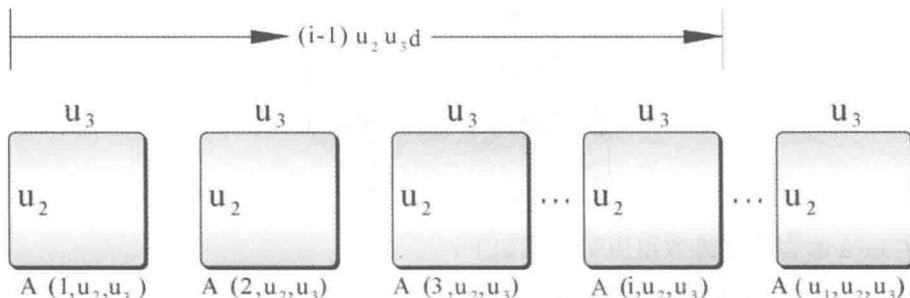


基本上，三维数组若以线性的方式来处理，一样可分为“以行为主”和“以列为主”两种方式。如果数组 A 声明为  $A(l_1:u_1, l_2:u_2, l_3:u_3)$ ，表示 A 为一个含有  $u_1 * u_2 * u_3$  个元素的三维数组。我们可以把  $A(i,j,k)$  元素想象成空间上的立方体。



▲ 以行为主 (Row-Major)

我们可以将数组 A 视为  $u_1$  个  $u_2 * u_3$  的二维列阵，再将每个数组视为有  $u_2$  个一维数组，每个一维数组可包含  $u_3$  个元素。另外每个元素有  $d$  个单位空间，且  $\alpha$  为数组起始地址。



在转换公式时，只要知道我们最终是要看看  $A(i,j,k)$  在一直线排列的第几个，所以很简单地可以得到以下地址计算公式：

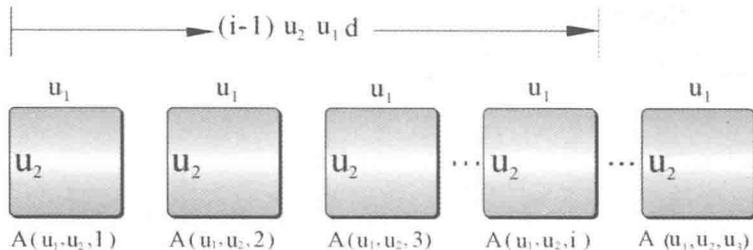
$$\text{Loc}(A(i, j, k)) = \alpha + (i-1) u_2 u_3 d + (j-1) u_3 d + (k-1) d$$

若数组 A 声明为  $A(l_1:u_1, l_2:u_2, l_3:u_3)$  模式，则：

```
a= u1- l1+1, b= u2- l2+1, c= u3- l3+1;
Loc(A(i, j, k))=a+(i-l1)bcd+(j-l2)cd+(k-l3)d
```

### ▲ 以列为主 (Column-Major)

将数组 A 视为  $u_3$  个  $u_2 * u_1$  的二维数组，再将每个二维数组视为  $u_2$  个一维数组，每一数组含有  $u_1$  个元素。每个元素有  $d$  个单位空间，且  $\alpha$  为起始地址：



$\Rightarrow \text{Loc}(A(i, j, k)) = \alpha + (k-1)u_2u_1d + (j-1)u_1d + (i-1)d$

若数组声明为  $A(l_1:u_1, l_2:u_2, l_3:u_3)$  模式，则：

```
a= u1- l1+1, b= u2- l2+1, c= u3- l3+1;
Loc(A(i, j, k))=a+(k-l3)abd+(j-l2)ad+(i-l1)d
```

### 范例 ▶ 2.2.3

$A(6,4,2)$  以 Row-major 方式排列，若  $\alpha=300$ ，且  $d=1$ ，求  $A(4,4,1)$  的地址。

### 解答 ▶

直接代入三维数组，以行为主的公式即可，请参考下面的表达式：

$\text{Loc}(A(4,4,1)) = 300 + (4-1) * 4 * 2 * 1 + (4-1) * 2 * 1 + (1-1) * 1 = 300 + 24 + 6 = 330$

## 2.2.4 n 维数组

有了一维、二维、三维数组，当然也可能有四维、五维或者更多维数的数组。不过因为受限于计算机内存，通常程序设计语言中的数组声明都会有维数的限制。在此，我们将三维以上的数组归纳为  $n$  维数组。例如，在 Java 语言中  $n$  维数组声明方式如下：

```
数据类型[] [].. [] 变量名称=new 数据类型[第一维长度][第二维长度]...[第n维长度];
```

假设数组 A 声明为  $A(l_1:u_1, l_2:u_2, l_3:u_3, \dots, l_n:u_n)$ ，则可将数组视为  $u_1$  个  $n-1$  维数组，每个  $n-1$  维数组中有  $u_2$  个  $n-2$  维数组，每个  $n-2$  维数组中，有  $u_3$  个  $n-3$  维数组……有  $u_{n-1}$  个一维数组，在每个一维数组中有  $u_n$  个元素。

如果  $\alpha$  为起始地址 ( $\alpha = \text{Loc}(A(1,1,1, \dots, 1))$ )， $d$  为单位空间，则数组 A 元素中的内存分配公式有如下两种方式。

### ▲ 以行为主 (row-major)

```
Loc(A(i1, i2, i3, …, in)) = a + (i1-1)u2u3u4…und
+ (i2-1)u3u4…und
```

---

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

**声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。**