

2023.6.25

Java二级学习

——学习笔记

猎奇人只爱无机盐

2023.7.12~2023. .

目录

[考试题型和分值 2](#_Toc18593)

[1.单选题（40分） 2](#_Toc25236)

[公共基础（10分） 2](#_Toc7895)

[专业知识（30分） 2](#_Toc558)

[2.操作题（60分） 2](#_Toc11386)

[基本操作（如：填空等，18分） 2](#_Toc6799)

[简单应用题（如：改错或窗体应用等，24分） 2](#_Toc2959)

[综合应用题（如：Applet程序设计等，18分） 2](#_Toc16449)

[公共基础 2](#_Toc7882)

[第一章、数据结构与算法 2](#_Toc4681)

[1、概述 2](#_Toc15275)

[2、数据结构 3](#_Toc24660)

[3、算法 8](#_Toc19102)

[4、常见的数据结构 12](#_Toc4838)

[第二章 、程序设计基础 28](#_Toc13515)

[1.程序设计的方法与风格 29](#_Toc28244)

[2.结构化程序设计 29](#_Toc30143)

[3.面向对象方法 30](#_Toc20060)

[第三章 、软件工程基础 31](#_Toc5916)

[1.软件工程基本概念 31](#_Toc10122)

[2.软件生命周期 31](#_Toc20196)

[3.软件设计 32](#_Toc17622)

[4.结构化分析方法 32](#_Toc32043)

[第四章、数据库设计基础 33](#_Toc11037)

[1、数据库系统的基本概念 33](#_Toc27687)

[2、数据库的发展和基本特点 38](#_Toc6491)

[3、数据库的系统内部体系结构 40](#_Toc3579)

[4、数据模型的基本概念 43](#_Toc5080)

[5、数据库设计与原理 52](#_Toc3510)

[Java语言程序设计 55](#_Toc6570)

[第一章 java语言基础 55](#_Toc30037)

[一、java概述 55](#_Toc24366)

[二、JDK环境变量设置 58](#_Toc32105)

[三、Java语言基础 63](#_Toc17787)

[第二章 java面向对象基础 78](#_Toc8319)

[一、基础 78](#_Toc3184)

[二、继承 86](#_Toc1305)

**java计算机二级**

# 考试题型和分值

## 1.单选题（40分）

### 公共基础（10分）

### 专业知识（30分）

## 2.操作题（60分）

### 基本操作（如：填空等，18分）

### 简单应用题（如：改错或窗体应用等，24分）

### 综合应用题（如：Applet程序设计等，18分）

# 公共基础

## 第一章、数据结构与算法

### 1、概述

* 用计算机解决现实生活与生产问题是计算机编程的核心意义
* 数据结构与算法正是为了高效的解决问题引入的两个概念
* 程序=算法+数据结构

• 数据结构是算法实现的基础

• 算法总是要依赖某种数据结构来实现的，算法的操作对象时数据结构

• 算法是编程思想，数据结构是这些思想的基础

### 2、数据结构

* 数据结构是计算机存储、组织数据的方式。
* 数据结构研究的三个方面：

• 1）数据集合中各数据之间固有的关系，即数据的逻辑结构；

• 2）对数据进行处理时，各元素在计算机中的存储关系，即数据的存储结构；

• 3）对各种数据结构的运算

* 数据结构的主要任务

• 是通过分析数据对象的结构特征，包括逻辑结构及数据对象之间的关系，然后把逻辑结构表示成计算机可实现的物理结构，从而便于计算机处理。

• 通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率。

* 2.1、数据结构基本术语

• 1）数据(Data)：

• 是描述客观事物的符号，

• 是计算机中可以操作的对象，

• 是能被计算机识别，并输出给计算机处理的符号集合。

• 数据不仅仅包括整型、实型等数值类型，还包括字符及声音、图像、视频等非数值类型。

• 　2）数据元素（Data Element）：

• 数据的基本单位。

• 一个数据元素由若干数据项构成，在计算机中通常作为整体处理，也被称为记录。

• 　3）数据项（Data Item）：

• 数据项是数据不可分割的最小单位。

• 比如：数据(是一张表)->数据元素(行)->数据项(字段)。

• 　4）数据对象（Data Object）：

• 是性质相同的数据元素的集合，

• 是数据的子集。

• 5）数据类型（Data Type）：

• 是指一组性质相同的值的集合及定义在此集合上的一些操作的总称。

• 　6）数据结构（Data Structure）：

• 是数据的组织形式

• 是数据元素相互之间存在的一种或多种特定关系的数据元素的集合

* 2.2、数据结构的分类

• 2.2.1、逻辑结构

• 数据的逻辑结构，面向的是用户，是从具体问题抽象出来的数学模型，

• 是描述数据元素及其关系的数学特性的。

• 在任何问题中，数据元素都不会是孤立的，在它们之间存在着这样或那样的关系，这种数据元素之间存在的关系称为数据的逻辑结构。

• 分类

• 2.2.1.1、线性结构

• 在该结构中，数据元素除了同属于一个集合外，数据元素之间还存在着一对一的顺序关系。

• 每一个节点都有一个前驱和一个后继,表示为R={d1,d2...dn}，各节点按逻辑关系排列,形成一条'链'。

• 如果一个数据满足下面两个条件：

• 1）有且只有一个根节点；

• 2）每一个节点最多有一个前件，也最多有一个后件

• 则称该数据为线性结构。线性结构又叫线性表。

• 常见的线性结构包括：

• 串（字符串）、队列、链表和栈等。

• 2.2.1.2、非线性结构

• 如果一个数据结构不是线性结构，则称之为非线性结构。

• 树状结构：

• 树形结构中的数据元素之间存在一种一对多的层次关系，每一个元素最多有一个前驱,能够有多个后继,表示为(D,{R}),就像一个树干长了多个树枝具有分支。

• 图状结构：

• 该结构的数据元素之间存在着多对多的任意关系

• 图状结构也称为网状结构。

• 集合结构：

• 在集合结构中，数据元素之间的关系是“属于同一个集合”，表示为R{}。

• 数据元素之间除了同属一个集合外，不存在其他关系。

• 常见的非线性结构包括：

• 广义表，树结构，图结构，多维数组等。

• 2.2.2、存储结构

• 数据结构在计算机中的表示称为存储结构(或物理结构)

• 存储结构主要分为

• 顺序存储、链式存储、索引存储和散列(哈希)存储。

• 1）顺序存储（线性结构）：

• 数据被存储在一组连续的存储单元中，其数据间的逻辑关系和物理关系是一致的。

• 特点：易查难插（删）。

• 2）链式存储（线性结构）：

• 借助元素地址指针表示数据的逻辑结构,每一个元素都会包含指向下一个元素的指针

• 这种结构须要在节点上附加一个指针项,指出后继节点的位置

• 即每一个节点存储单元包含两个部分:[数据项,指针项]。

• 特点：易插（删）难查。

• 3）索引存储：

• 指存储在数据元素信息的同时还建立附加的索引表。

• 索引表中的所有索引项是有序排列的。

• 特点：查找效率高，但额外增加了空间的开销。

• 4）散列存储(哈希表)：

• 根据元素的关键字通过哈希（或散列）函数直接计算出一个值，并将这个值作为该元素的存储地址。

• 特点：查找速度快。

• 一个完整的存储结构必须能够存储数据元素,以及元素之间的逻辑关系。

• 2.2.3、数据的运算

• 数据的运算是指：施加在数据上的运算包括运算的定义和实现。

• 运算的定义是针对逻辑结构的，指出运算的功能；

• 运算的实现是针对存储结构的，指出运算的具体操作步骤。

• 常见的数据运算有：

• 插入（新增）、删除、修改（更新）、查找（检索）、排序等。

### 3、算法

* 概念：

• 算法，字面意思即计算方法，一个算法就是对特定问题求解步骤的一种描述。

• 算法规定了求解给定类型问题所需的全部处理步骤以及执行顺序,使得问题能在优先时间内机械的求解

• 再具体一点,算法是一段有穷的指令序列，算法必须能使用某种语言描述。

• 注：算法并不等于公式，公式是一种高度精简的算法，算法的形式可以比公式更复杂，解决的问题更加广泛。

* 3.1、算法特性

• 1）有穷性：

• 指算法在执行有限的步骤之后，自动结束而不会出现无限循环，并且每一个步骤在可接受的时间内完成；

• 2）确定性：

• 算法的每一步骤都具有确定的含义，不会出现二义性；

• 3）可行性：

• 算法的每一步都必须是可行的，也就是说，每一步都能够通过执行有限次数完成；

• 4）输入：

• 是指算法具有零个或多个输入；

• 5）输出：

• 是指算法至少有一个或多个输出；

* 3.2、算法的分类

• 1）根据应用分类：

• 按照算法的应用领域，可以分为

• 基本算法

• 数据结构相关算法

• 几何算法

• 图论算法

• 规划算法

• 数值分析算法

• 加密解密算法

• 排序算法

• 查找算法

• 并行算法

• 数值算法

• 等等

• 2）根据确定性分类：

• 确定性算法：

• 有限时间内完成，得到结果唯一；

• 非确定性算法：

• 有限时间内完成，得到结果不唯一，存在多值性。

• 3）根据算法的思路分类：

• 递推算法

• 递归算法

• 穷举算法

• 贪婪算法

• 分治算法

• 动态规划算法

• 迭代算法

• 等

* 3.3、算法的表示

• 一般有三种流程结构：

• 顺序结构

• 分支结构

• 循环结构

* 3.4、算法的性能

• 算法的效率作为判断算法优劣的标准。一个算法的优劣往往通过算法复杂度来衡量

• 分类：

• 时间复杂度

• 时间复杂度是指执行算法所需要的计算工作量

• 即通常所说的算法执行所需要耗费的时间，时间越短，算法越好。

• 算法的时间度量，记作：T（n）=O（f（n））。

• 它表示随问题规模n的增大，算法执行时间的增长率和f（n）的增长率相同。

• 常见的时间复杂度量级有：

•



• 空间复杂度

• 空间复杂度是指执行这个算法所需要的内存空间。

• 是对一个算法在运行过程中临时占用存储空间大小的量度，记做S(n)=O(f(n))。

• 其中，n为问题的规模，f(n)为语句关于n所占存储空间的函数。

• 一般而言，常常以算法执行时间作为算法优劣的主要衡量指标。

### 4、常见的数据结构

* 4.1、数组

• 4.1.1、概述

• 数组是一种聚合数据类型，它是将具有相同类型的若干变量有序地组织在一起的集合。

• 数组可以说是最基本的数据结构，在各种编程语言中都有对应。

• 一个数组可以分解为多个数组元素

• 按照数据元素的类型，数组可以分为

• 整型数组

• 字符型数组

• 浮点型数组

• 指针数组

• 结构数组

• 等

• 数组还可以有一维、二维(比较重要，大题必考)以及多维等表现形式。

• 4.1.2、数组运算

• 遍历：

• 遍历所有元素；

• 插入：

• 将一个或多个元素插入数组；

• 删除：

• 从数组中删除元素；

• 搜索：

• 在数组中搜索元素。您可以按元素的值或索引搜索元素；

• 更新：

• 在给定索引处更新现有元素的值。

• 4.1.3、数组的应用

• 用作构建其他数据结构的基础

• 例如数组列表，堆，哈希表，向量和矩阵；

• 用于不同的排序算法

• 例如插入排序，快速排序，冒泡排序和合并排序。

* 4.2、链表

• 4.2.1、概述

• 链表是一种顺序结构，由相互链接的线性顺序项目序列组成。

• 这种存储结构具有在物理上存在非连续的特点。

• 链表由一系列数据结点构成，每个数据结点包括数据域和指针域两部分。

• 其中，指针域保存了数据结构中下一个元素存放的地址。

• 链表结构中数据元素的逻辑顺序是通过链表中的指针链接次序来实现的。

• 链式存储方式既可用于表示线性结构，也可用于表示非线性结构。

• 4.2.2、链表分类

• 单链表：

• 只能沿正向遍历项目；

• 双链表：

• 可以在前进和后退方向上遍历项目。节点由一个称为上一个的附加指针组成，指向上一个节点；

• 循环链接列表：

• 链接列表，其中头的上一个指针指向尾部，尾号的下一个指针指向头；

• 静态链表：

• 线性存储结构的一种，兼顾顺序表和链表的优点，是顺序表和链表的升级；

• 静态链表的数据全部存储在数组中(顺序表)，但存储的位置是随机的。（因为物理位置连续的，逻辑位置不一定连续）

• 4.2.3、链表操作

• 搜索：

• 通过简单的线性搜索在给定的链表中找到键为k的第一个元素，并返回指向该元素的指针

• 插入：

• 在链表中插入一个元素x。

• 插入可以通过3种不同的方式完成

• 在列表的开头插入

• 在列表的末尾插入

• 在列表的中间插入

• 头结点用来记录链表的基地址，是链表遍历的起点

• 尾结点的后继指针不指向任何结点，而是指向一个空地址NULL。

• 删除：

• 从给定的链表中删除元素x。

• 删除可以通过3种不同方式完成

• 从列表的开头删除

• 从列表的末尾删除

• 从列表的中间删除

• 4.2.4、链表的应用

• 用于编译器设计中的符号表管理。

• 用于在使用Alt Tab（使用循环链表实现）的程序之间进行切换。

• 注：

• 数组无法动态扩容

• 如果申请空间不够，需要申请更大的空间，需要数据复制拷贝进入新的数组，非常耗时；

• 而链表天然支持动态扩容，因为他不要内存空间的连续性

• LinkedHashMap的底层实现就是用的双向链表结构。

• 多线程下Hashmap扩容时出现循环链表，jdk1.8把头插法换成了尾插法。

* 4.3、栈

• 4.3.1、概述

• 栈是一种特殊的线性表，

• 它只能在一个表的一个固定端进行数据结点的插入和删除操作。

• 栈按照"后进先出"或“先进后出”的原则来存储数据

• 先插入的数据将被压入栈底，

• 最后插入的数据在栈顶，

• 读出数据时，从栈顶开始逐个读出。

• 栈中没有数据时，称为空栈。

• 4.3.2、栈的顺序存储及其运算

• 栈的基本运算有3种：

• 入栈

• 入栈运算：在栈顶位置插入一个新元素；

• 退栈

• 退栈运算：取出栈顶元素并赋值给一个指定的变量；

• 读栈顶元素

• 读栈顶元素：将栈顶元素赋值给一个指定的变量；

* 4.4、队列

• 4.4.1、概述

• 队列和栈类似，也是一种特殊的线性表。

• 和栈不同的是，队列只允许在表的一端进行插入操作，而在另一端进行删除操作。

• 一般来说，进行插入操作的一端称为队尾，进行删除操作的一端称为队头。

• 队列中没有元素时，称为空队列。

• 4.4.2、队列操作

• 进队：

• 将元素插入队列的末尾。

• 出队：

• 从队列的开头删除元素。

• 4.4.3、队列的应用

• 用于管理多线程中的线程。

• 用于实施排队系统（例如：优先级队列）。

• 扩展理解：例题

• 循环队列的存储空间为 Q(1:40) ，初始状态为 front=rear=40 。经过一系列正常的入队与退队操作后， front=rear=15 ，此后又退出一个元素，则循环队列中的元素个数为（ ）。

• A、39，或0且产生下溢错误

• B、14

• C、40

• D、15

• 注：当队列front=rear=15时，此时队列可能满了或空了，再退出元素，可能是39，或0且产生下溢错误。

• 循环队列

• 基本概念：

• 队列是一种先进先出（FIFO）的线性表

• 循环队列即为队列的顺序表示和实现，

• 其优点主要是入队和出队的时间复杂度均为O（1）

• 这归功于“循环队列将存储结构的理解由序列型转换为环形”的思维方式

• 基本属性：

• 两个指针（其实就是两个整数型变量，因为在这里有指示作用，所以这里理解为指针）

• front

• 一个指示队头

• rear

• 一个指示队尾

• rear和front互相追赶着，这个追赶过程就是队列添加和删除的过程

• 如果rear追到front说明队列满了

• 如果front追到rear说明队列为空

• 即当rear=front时，队列要么为空要么为满

• 我们把他掰弯，用的是求余，这样两个值就不会跑出最大范围，并且可以实现弯曲的效果

• 所以说对于循环队列我们必须给定最大值MAXQSIZE

• 实现过程：

• 添加：

• （rear+1）%MAXQSIZE

• 在jdk实现中，弯曲的效果是tail=(tail+1)&(elements.length-1)，这里elements.length初始值是16，每次容量扩大会乘以2，这里用到公式：当x=2^n（n为自然数）时，a%x=a&(x-1)

• 删除：

• （front+1）%MAXQSIZE

• 当rear=front时，队列

• 可能是满的

• 当队列添加元素到rear的下一个元素是head的时候，也就是转圈子要碰头了，我们就认为队列满了。

• (Q.rear+1)%MAXQSIZE=Q.front

• 也可能是空的

• 当队列删除元素到front=rear的时候，我们认为队列空了。

• Q.rear==Q.front，不一定为0

* 4.5、树

• 4.5.1、树的概述

• 树是典型的非线性结构，它是包括，2个结点的有穷集合K。

• 在树结构中，有且仅有一个根结点，该结点没有前驱结点。

• 在树结构中的其他结点都有且仅有一个前驱结点，而且可以有两个后继结点，m≥0。

• 4.5.2、树的相关概念

• 父节点：

• 每一个节点都只有一个前件，称为父节点，没有前件的节点只有一个，称为根节点；

• 子节点：

• 每一个节点可以有多个后件，称为该节点的子节点，没有后件的节点，称为叶子节点；

• 度：

• 一个节点所拥有的后件个数称为该节点的度，所有节点中最大的度称为树的度；

• 深度：

• 树中结点的最大层次数称为树的深度或高度；

• 4.5.3、树的种类

• 无序树：

• 树中任意节点的子结点之间没有顺序关系，这种树称为无序树,也称为自由树;

• 有序树：

• 树中任意节点的子结点之间有顺序关系，这种树称为有序树；

• 二叉树：

• 每个节点最多含有两个子树的树称为二叉树；

• 完全二叉树

• 满二叉树

• 斜树

• 平衡二叉树

• 霍夫曼树：

• 带权路径最短的二叉树称为哈夫曼树或最优二叉树；

• 4.5.4、二叉树

• 特点：

• 每个结点最多只有两个子结点，可以没有或者只有一个。

• 左子树和右子树是有顺序的，次序不能任意颠倒。

• 性质：

• 在二叉树的第k层上，最多有2k-1个结点；

• 深度为m的二叉树最多有2m-1个结点；

• 具有n个结点的二叉树，其深度至少为log2n+1

• 4.5.5、满二叉树

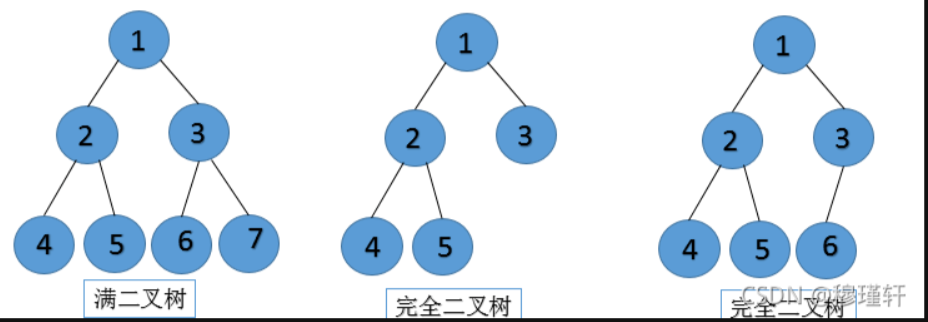
• 除最后一层（叶子节点）外，每一层上的所有结点都有两个子结点。

• 4.5.6、完全二叉树

• 除最后一层外，每一层上的结点数都达到最大值，在最后一层上只缺少右边的若干结点。

• 满二叉树也是完全二叉树，反过来不一定成立。

•



• 4.5.7、平衡二叉树（AVL）

• 它是一 棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树，同时，平衡二叉树必定是二叉查找树。

• 4.5.8、二叉查找树

• 若它的左子树不为空，则左子树上所有节点的值均小于它的根节点的值；

• 若它的右子树不为空，则右子树上所有节点的值均大于它的根节点的值；

• 它的左、右子树也分别是二叉排序树。

• 说明它是一颗有顺序的树，左子树节点的值小于根节点的值，右子树节点的值大于根节点的值。

• 红黑树：

• 红黑树的节点是有颜色概念的，即非红即黑。

• 通过颜色的约束，红黑树维持着二叉搜索树的平衡性。

• 一颗红黑树必须满足以下几点条件：

• 1）根节点必须是黑色。

• 2）任意从根到叶子的路径不包含连续的红色节点。

• 3）任意从根到叶子的路径的黑色节点总数相同。

• 4.5.9、霍夫曼树

• 带权路径最短的二叉树称为哈夫曼树或最优二叉树。

• 4.5.10、B-树

• B-树是一种平衡多路查找树，它在文件系统中很有用。

• 4.5.11、B+树

• B+数是B-树的一种变形。

• 4.5.12、字典树（trie树）

• 字典树是一种以树形结构保存大量字符串。

• 以便于字符串的统计和查找，经常被搜索引擎系统用于文本词频统计。

• 优点：

• 利用字符串的公共前缀来节约存储空间，最大限度地减少无谓的字符串比较，查询效率比哈希表高。

• 4.5.13、后缀树

• 包含一则字符串所有后缀的压缩了的字典树。

* 4.6、堆

• 4.6.1、概述

• 堆(Heap)是一种重要的数据结构，是实现优先队列(Priority Queues)首选的数据结构。

• 由于堆有很多种变体，包括二项式堆、斐波那契堆等，但是这里只考虑最常见的就是二叉堆。

• 特点：

• 根结点的值是所有结点中最小的或者最大的

• 并且根结点的两个子树也是一个堆结构。

• 4.6.2、堆的应用

• 用于实现优先级队列，因为可以根据堆属性对优先级值进行排序。

• 可以在O（log n）时间内使用堆来实现队列功能。

• 用于查找给定数组中k个最小（或最大）的值。

• 用于堆排序算法。

* 4.7、图

• 4.7.1、概述

• 图是另一种非线性数据结构。

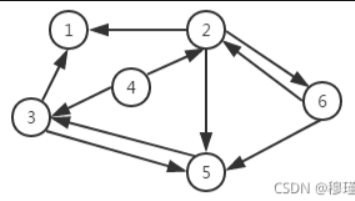
• 在图结构中，数据结点一般称为顶点，而边是顶点的有序偶对。

• 如果两个顶点之间存在一条边，那么就表示这两个顶点具有相邻关系。

• 4.7.2、有向图

• 一个图结构中，边是有方向性的，那么这种图就称为有向图。

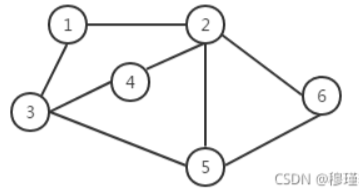
•



• 4.7.3、无向图

• 如果一个图结构中，所有的边都没有方向性，那么这种图便称为无向图。

•



• 4.7.4、图的应用

• 用于表示社交媒体网络。

• 每个用户都是一个顶点，并且在用户连接时会创建一条边。

• 用于表示搜索引擎的网页和链接。

• 互联网上的网页通过超链接相互链接。

• 每页是一个顶点，两页之间的超链接是一条边。

• 用于Google中的页面排名。

• 用于表示GPS中的位置和路线。

• 位置是顶点，连接位置的路线是边。

• 用于计算两个位置之间的最短路径。

* 4.8、哈希表

• 4.8.1、概述

• 散列表源自于散列函数(Hash function)

• 思想：

• 如果在结构中存在关键字和T相等的记录，那么 必定在F(T)的存储位置可以找到该记录，这样就可以不用进行比较操作而直接取得所查记录。

• 4.8.2、哈希表的应用

• 用于实现数据库索引。

• 用于实现关联数组。

• 用于实现"设置"数据结构。

## 第二章 、程序设计基础

### 1.程序设计的方法与风格

* 1.1.源程序文档化

• 符号名的命名

• 程序注释

• 视觉组织

* 1.2.数据说明的方法
* 1.3.语句的结构程序
* 1.4.输入和输出

### 2.结构化程序设计

* 2.1.结构化程序的设计原则

• 自顶向下

• 逐步求精

• 模块化

• 限制使用goto语句

* 2.2.结构化程序的基本结构

• 顺序结构

• 选择结构

• 循环结构

### 3.面向对象方法

* 3.1.对象

• 对象的特征：

• 标识唯一性

• 分类性

• 多态性

• 封装性

• 模块独立性

* 3.2.类和实例

• 类是具有共同属性、共同方法的集合

• 对象则是对其对应类的一个实例

* 3.3.消息

• 消息是实例之间传递的信息

• 它请求对象执行某一处理或回答某一要求的信息

• 它统一了数据流和控制流

* 3.4.继承

• 广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们

* 3.5.多态性

• 对象根据所接收的消息而做出动作，同样的消息被不同的对象接收时可能导致完全不同的行动。

## 第三章 、软件工程基础

### 1.软件工程基本概念

* 1.1.定义与特点

• 研究的主要内容包括：

• 软件开发技术

• 软件工程管理

* 1.2.软件工程三要素：

• 方法

• 工具

• 过程

### 2.软件生命周期

* 2.1.软件定义期

• 问题定义

• 可行性研究

• 需求分析

* 2.2.软件开发期

• 概要设计

• 详细设计

• 实现

• 测试

* 2.3.运行维护期

• 使用和维护

### 3.软件设计

* 3.1.概述
* 3.2.基本原理

• 抽象

• 模块化

• 信息隐藏

• 模块独立性

* 3.3.相关概念

• 内聚性

• 耦合性

### 4.结构化分析方法

* 4.1.概述
* 4.2.常用工具

• 数据流图（DFD）

• 数据字典（DD）

• 判定表

• 判定树

## 第四章、数据库设计基础

### 1、数据库系统的基本概念

* 1.1、数据

• 数据(Data)实际上就是描述事物的符号记录。

• 计算机中的数据一般分为两部分

• 其中一部分与程序仅有短时间的交互关系，随着程序的结束而消亡，它们称为临时性（Transient ）数据，这类数据一般存放于计算机内存中

• 另一部分数据则对系统起着长期持久的作用，它们称为持久性（Persistent ）数据。数据库系统中处理的就是这种持久性数据。

• 软件中的数据是有一定结构的

• 数据有型（Type ）与值（Value）之分

• 数据的型给出了数据表示的类型，如整型、实型、字符型等

• 数据的值给出了符合给定型的值，如整型值15

• .随着应用需求的扩大它包括了将多种相关数据以一定结构方式组合构成特定的数据框架，这样的数据框架称为数据结构（Data structure ）,数据库中在特定条件下称之为数据模式（Data schema ） .

• 在过去的软件系统中是以程序为主体，而数据则以私有形式从属于程序，此时数据在系统中是分散、凌乱的，这也造成了数据管理的混乱，如数据冗余度高，数据一致性差以及数据的安全性差 等多种弊病。

• 近10多年来，数据在软件系统中的地位产生了变化，在数据库系统以及数据库应用系统中数据已占有主体地位，而程序已退居附属地位。

• 在数据库系统中需要对数据进行集中、统一 的管理，以达到数据被多个应用程序共享的目标。

* 1.2、数据库

• 数据库（Database ,简称DB ）是数据的集合，它具有统一的结构形式并存放于统一的存储介质内，是多种应用数据的集成，并可被各个应用程序所共享。

• 数据库存放数据是按数据所提供的数据模式存放的，它能构造复杂的数据结构以建立数据间内在联系与复杂的关系，从而构成数据的全局结构模式。

• 数据库中的数据具有“集成”、“共享”之特点，亦即是数据库集中了各种应用的数据，进行统一的构造与存储，而是他们可以被不同的应用程序所使用。

* 1.3、数据库管理系统

• 数据库管理系统（ Database Management System, 简称 DBMS ）是数据库的机构，它是一种系统软件，负责数据库中的数据组织、数据操纵、数据维护、控制及保护和数据服务等。

• 数据库中的数据是具有海量级的数据，并且其结构复杂，因此需要提供管理工具。

• 数据库管理系统是数据库系统的核心

• 具体功能：

• (1)数据模式定义。

• 数据库管理系统负责为数据库构建模式，也就是为数据库构建其数据框架。

• (2) 数据存取的物理构建。

• 数据库管理系统负责为数据模式的物理存储及构建提供有效的存取方法与手段。

• (3) 数据操纵。

• 数据库管理系统为用户使用数据库中的数据提供方便，它一般提供查询、插入、修改、删除数据的功能。此外，它自身还具有做简单算术运算及统计的能力，而且还可以和某些过程性语言结合，使其具有强大的过程性操作能力。

• (4)数据的完整性、安全性定义与检查。

• 数据库中的数据具有内在语义上的关联性与一致性，他们构成了数据的完整性，数据的完整性是保证数据库中的数据正确的必要条件，因此必须经常检查以维护数据的正确。

• 数据库中的数据具有共享性，而数据共享可能会引发数据的非法使用，因此必须要对数据正确使用作出必要的规定，并在使用时做检查，这就是数据的安全性。

• 数据完整性与安全性的维护是数据库管理系统的基本功能。

• (5)数据库的并发控制与故障恢复。

• 数据库是一个集成、共享的数据集合体，它能为多个应用程序服务，所以就存在着多个应用程序对数据库的并发操作。

• 在并发操作中如果不加以控制和管理，多个应用程序间就会相互干扰，从而对数据库中的数据造成破坏。

• 因此，数据库管理系统必须对多个应用程序的并发执行做必要的控制一保证数据不受破坏，这就是数据库的并发控制。

• 数据库中的数据一旦遭受破坏，数据库管理系统必须有能力及时进行恢复，这就是数据库的故障恢复。

• (6) 数据的服务。

• 数据库管理系统提供对数据的多种服务功能，如数据拷贝、转存、重组、性能检测、分析等。

•

• 1.3.1、数据语言

• 数据定义语言（ Data Definition Language 简称 DDL ）：

• 该语言负责数据的模式定义与数据的物理存取构建。

• 数据操纵语言（ Data Manipulation Language 简称 DML ）：

• 该语言负责数据的操纵，包括查询及增、删、改等操作。

• 数据控制语言（ Data Control Language 简称 DCL ）：

• 该语言负责数据完整性、安全性定义与检查以及并发控制、故障恢复等功能，包括系统初起程序、文件读写与维护程序、存取路径管理程序、缓冲区管理程序、安全性控制程序、完整性检查程序、并发控制程序、事务管理程序、运行日志管理程序、数据库恢复程序等。

• 1.3.2、使用方式

• 交互式命令(又称自含型或自主型语言)

• 它的语言简单，能在终端上即时操作，它又称为自含型或自主型语言。

• 宿主型语言（一般可嵌入某些宿主语言中）

• 它一般可嵌入某些宿主语言（ Host Language ）中，如 C ， C++ 和 COBOL 等高级过程性语言中。

• 此外，数据库管理系统还为用户提供服务的服务性（ Utility ）程序，包括数据初始装入程序、数据转存程序、性能检测程序、数据库再组织程序、数据转换程序、通信程序等。

• 目前流行的DBMS 均为关系数据库系统，比如 ORACLE 、 Sybase 的 PowerBuilder 及 IBM 的 DB2 、微软的 SQL Server 等，他们均为严格意义上的 DBMS 系统。

• 另外一些小型的数据库，如微软的 Visual Foxpro 和 Access 等，它们只具备数据库管理系统的一些简单功能

* 1.4、其他相关概念

• 数据库管理员：

• 对数据库进行规划、设计、维护、监视等的专业管理人员。

• 数据库系统：

• 由数据库（数据）、数据库管理系统（软件）、数据库管理员（人员）、硬件平台（硬件）、软件平台（软件）五个部分构成的运行实体。

• 数据库应用系统：

• 由数据库系统、应用软件及应用界面三者组成。

### 2、数据库的发展和基本特点

* 2.1、数据库的发展

• 人工管理阶段

• 在 20 世纪 50 年代中期以前，主要用于科学计算，硬件无磁盘，直接存取，软件没有操作系统。

• 20 世纪 50 年代后期到 20 世纪 60 年代中期，进入文件系统阶段。

• 文件系统是数据库发展的初级阶段，它提供了简单的数据共享与数据管理能力。

• 但是它无法提供完整的、统一的、管理和数据共享的能力。

• 由于它的功能简单，因此它附属于操作系统而不成为独立的软件，目前一般将其看成是数据库系统的雏形，而不是真正的数据库系统。

• 20 世纪 60 年代之后，数据管理进入了数据库系统阶段。

• 层次数据库与网状数据库阶段 从 20 世纪 60 年代末期起，真正的数据库系统 ------ 层次数据库与网状数据库开始发展，它们为统一管理与数据共享提供了有力支持，这个时期数据库系统蓬勃发展形成了有名的“数据库时代”。

• 关系数据库系统出现于 20 世纪 70 年代，在 80 年代得到蓬勃发展，并逐渐取代前两种系统，关系数据库系统结构简单，使用方便，逻辑性强物理性少，因此在 80 年代以后一直占据着数据库系统的主导地位。

* 2.2、数据库系统的基本特点

• 数据的集成性：

• (1)在数据库系统中采用统一的数据结构方式，如在关系数据库中采用二维表作为统一结构方式。

• (2) 在数据库系统中按照多个应用的需要组织全局的统一的数据结构（即数据模式），数据模式不仅可以建立全局的数据结构，还可以建立数据间的语义关系从而构成一个内在紧密联系的数据整体。

• (3) 数据库系统中的数据模式是多个应用共同的、全局的数据结构，而没个应用的数据则是全局数据结构中的一部分，称为局部结构（即视图），这种全局与局部的结构模式构成了数据库系统数据集成性的主要特征。

• 数据的高共享性与低冗余性：

• 数据的共享自身又可极大的减少数据冗余性，不仅减少了不必要的存储空间，更为重要的是可以避免数据的不一致性。

• 数据独立性（物理独立性与逻辑独立性）：

• 数据独立性是数据与程序间的互不依赖性，即数据库中的数据独立于应用程序而不依赖于应用程序。

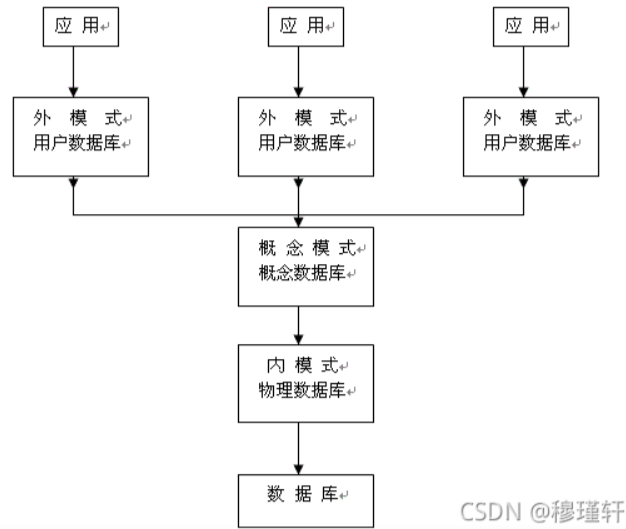
• 也就是说，数据的逻辑结构、存储结构和存取方式的改变不会影响应用程序。

• 数据统一管理与控制：

• 数据库系统不仅为数据提供高度集成环境，同时它还为数据提供统一管理的手段：数据的完整性检查、数据的安全性保护、并发控制。

### 3、数据库的系统内部体系结构

* 数据库系统在其内部具有三级模式及二级映射，三级模式分别是概念模式，内部级模式与外部级模式，二级映射分别是概念级到内部级的映射以及外部级到概念级的映射。



• 3.1、三级模式

• 概念模式（ Conceptual Schema ）：

• 是数据库系统中全局数据逻辑结构的描述，是全体用户（应用）公共数据视图。

• 此种描述是一种抽象的描述，它不设计具体的硬件环境与平台，也与具体的软件环境无关。

• 概念模式主要描述数据的概念记录类型以及它们间的关系，它还包括一些数据间的语义约束，对它的描述可用 DBMS 中的 DDL 语言定义。

• 外模式（ External Schema ）：

• 又称用户模式（ User`s schema ），它是用户的数据视图，也就是用户所见到的数据模式，它由概念模式推导而出。

• 概念模式给出了系统全局的数据描述而外模式则给出每个用户的局部数据描述。

• 一个概念模式可以有若干个外模式，每个用户只关心与它有关的模式，这样不仅可以屏蔽大量无关信息而且有利于数据保护。

• 在一般的 DBMS 中都提供有相关的外模式描述语言（外模式 DDL ）。

• 内模式（ Internal schema ）：

• 又称物理模式（ Physical Schema ），它给出了数据库物理存储结构与物理存取方法，如数据存储文件结构、索引、集簇及 hash 等存取方式与存取路径，内模式的物理性主要体现在操作系统及文件级上，它还未深入到设备级上（如磁盘及磁盘操作）。

• 内模式对一般用户是透明的，但它的设计直接影响数据库的性能。

• DBMS 一般提供相关的内模式描述语言（内模式 DDL ）。

• 这三种数据库中只有物理数据库是真实存在于计算机外存中的，其它两种数据库并不存在于计算机中，而是通过两种映射由物理数据库映射而成。模式的三个级别层次反映了模式的三个不同环境以及它们的不同要求，其中内模式处于最底层，它反映了数据在计算机物理结构中的实际存储形式，概念模式处于中层，它反映了设计者的数据全局逻辑要求，而外模式处于最外层，它反映了拥护对数据的要求。

• 3.2、两级映射

• 两级映射也保证了数据库系统中数据的独立性，亦即数据的物理组织改变与逻辑概念级改变相互独立，使得只要调整映射方式而不必改变用户模式。

• 概念模式到内模式的映射：

• 该映射给出了概念模式中数据的全局逻辑结构到数据的物理存储机构间的对应关系，此种映射一般由 DBMS 实现。

• 外模式到概念模式的映射：

• 概念模式是一个全局模式而外模式是用户的局部模式。

• 一个概念模式中可以定义多个外模式，而每个外模式是概念模式的一个基本视图。

• 外模式到概念模式的映射给出了外模式与概念模式的对应关系，这种映射一般也是由 DBMS 来实现的。

• 扩展理解：例题

• 在数据库的三级模式结构中，外模式有

• A．1个

• B．2个

• C．3个

• D．任意多个

• 注：外模式也称子模式或用户模式，是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。一个数据库可以有多个外模式，内模式只能有一个。

### 4、数据模型的基本概念

* 4.1、数据模型的基本概念

• 是数据特征的抽象，从抽象层次上描述了系统的静态特征、动态行为和约束条件，为数据库系统的信息表与操作提供一个抽象的框架。

• 数据模型所描述的内容有三个部分

• 4.1.1、数据结构

• 数据模型中的数据结构主要描述数据的类型、内容、性质以及数据间的联系等。

• 数据结构是数据模型的基础，数据操作与约束均建立在数据结构上。

• 不同数据结构有不同的操作与约束，因此，一般数据模型的分类均以数据结构的不同而分。

• 4.1.2、数据操作

• 数据模型中的数据操作主要描述在相应数据结构上的操作类型与操作方式。

• 4.1.3、数据约束

• 数据模型中的数据约束主要描述数据结构内数据同的语法、语义联系，它们之间的制约与依存关系，以及数据动态变化的规则，以保证数据的正确、有效与相容。

* 4.2、数据模型

• 数据模型按不同的应用层次分成三种类型

• 概念数据模型（ conceptual data model ）

• 概念数据模型简称概念模型，它是一种面向客观世界、面向用户的模型；

• 它与具体的数据库管理系统无关，与具体的计算机平台无关。

• 概念模型着重于对客观世界复杂事物的结构描述及它们之间的内在联系的刻画。

• 概念模型是整个数据模型的基础。

• 目前，较为有名的概念模型有 E-R 模型、扩充的 E-R 模型、面向对象模型及谓词模型等。

• 4.2.1、 E-R 模型

• 实体：

• 现实世界中的事物；

• 属性：

• 事物的特性；

• 联系：

• 现实世界中事物间的关系。

• 实体集的关系有一对一、一对多、多对多的联系。

• E-R模型三个基本概念之间的联接关系：

• 实体是概念世界中的基本单位，属性有属性域，每个实体可取属性域内的值。

• 一个实体的所有属性值叫元组。

• E-R模型的图示法：

• （1）实体集表示法；

• 实体集表示法（矩形表示实体）：

• 如，实体集学生（student）和课程（course）

desc

• （2）属性表法；

• 属性表法（椭圆形表示属性）：

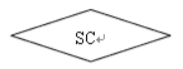
• 如，学生有属性：学号（S#）、姓名（Sn）及年龄（Sa）

desc

• （3）联系表示法

• 联系表示法（菱形表示联系）：

• 如，学生与课程的类型SC



• 逻辑数据模型（ logic data model ）

• 逻辑数据模型又称数据模型，它是一种面向数据库系统的模型，该模型着重于在数据库系统一级的实现。

• 概念模型只有在转换成数据模型后才能在数据库中得以表示。

• 目前，逻辑数据模型也有很多种，较为成熟并先后被人们大量使用过的有；层次模型、网状模型、关系模型、面向对象模型等。

• 4.2.2、层次模型

• 层次模型的基本结构是树形结构

• 具有以下特点：

• 每棵树有且仅有一个无双亲结点，称为根；

• 树中除根外所有结点有且仅有一个双亲。

• 4.2.3、关系模型

• 关系的数据结构关系模型采用二维表来表示，简称表。

• 二维表由表框架 (Frame) 及表的元组—（ Tuple ）组成。

• 表框架由 n 个命名的属性（ Attribute ）组成， n 称为属性元数 (Arity) 。

• 每个属性有一个取值范围称为值域（ Domain ）。

• 表框架对应了关系的模式，即类型的概念。

• 在表框架中按行可以存放数据，每行数据称为元组，实际上，一个元组是由 n 个元组分量所组成，每个元组分量是表框架中每个属性的投影值。

• 一个表框架可以存放 m 个元组， m 称为表的基数（ Cardinality ）。

• 一个 n 元表框架及框架内 m 个元组构成了一个完整的二维表。

• 在二维表中凡能唯一标识元组的最小属性称为键或码。

• 从所有侯选健中选取一个作为用户使用的键称主键。

• 表A中的某属性是某表B的键，则称该属性集为A的外键或外码。

• 关系中的数据约束：

• 实体完整性约束：

• 约束关系的主键中属性值不能为空值；

• 参照完全性约束：

• 是关系之间的基本约束；

• 用户定义的完整性约束：

• 它反映了具体应用中数据的语义要求。

• 物理数据模型（ physical data model ）

• 物理数据模型又称物理模型，它是一种面向计算机物理表示的模型，此模型给出了数据模型在计算机上物理结构的表示。

* 4.3、关系代数

• 关系数据库系统的特点之一是它建立在数据理论的基础之上，有很多数据理论可以表示关系模型的数据操作，其中最为著名的是关系代数与关系演算。

• 关系模型的基本运算：

• 1）插入

• 2）删除

• 3)修改

• 4）查询（包括投影、选择、笛卡尔积运算）

• 五个基本操作：

• 并(∪)

• 差(-)

• 笛卡尔积(×)

• 投影(π)

• 选择(σ)

• 四个组合操作：

• 交(∩)

• 联接(等值联接)

• 自然联接(RS)

• 除法(÷)

• 4.3.1、关系代数表达式

• 建立一个学生选课的关系数据库，它由下面三个关系模式组成：

• 学生关系S(SNO,SNAME,AGE,SEX)

• 学生关系S（学生学号，姓名，年龄，性别）

• 学习关系SC(SNO,CNO,GRADE)

• 学习关系SC（学生学号，课程号，成绩）

• 课程关系C(CNO,CNAME,TEACHER)

• 程关系C（课程号，课程名称，老师）

• 选择(SELECT)运算：

• 查询表中年龄小于20的学生

• SQL方式：

• select \* from S where AGE< 20

• 关系代数方式：

• δAGE<20(S)

• 投影(PROJECTION)运算：

• 投影操作是从列的角度进行运算，查询学生的姓名和年龄，即求S关系上学生姓名和年龄两个属性上的投影

• SQL方式：

• select S.SNAME,S.AGE form S

• 关系代数方式：

• π SNAME,AGE(S)

• 连接(JOIN)运算：

• 连接运算也称为θ连接，从两个关系（R、S）中的广义笛卡尔积中选取属性间满足一定条件的元组形成一个新的连接，θ不为“=”的连接称为非等值连接， θ为“=”的连接称为等值连接。检索学习课程号为C2的学生学号与姓名

• SQL方式：

• select SC.SNO,S.SNAME from SC,S where SC.SNO=S.SNO and SC.CNO='C2'

• 关系代数方式：

• π SNO,SNAME (σCNO='C2'(SSC))

• 其他运算：

• 检索选修课程号为C2或C4的学生学号

• SQL方式：

• select SNO FROM SC where CNO='C2' or CNO='C4'

• 关系代数方式：

• π SNO(σ CNO='C2'∨CNO='C4'(SC))

• 检索至少选修课程号为C2或C4的学生学号

• SQL方式：

• select SA.SNO from SC AS SA,SC AS SB where SA.SNO=SB.SNO and SA.CNO='C2' and SB.CNO='C4'

• 关系代数方式：

• π 1(σ1=4∧2='C2'∧5='C4'（SC×SC）)

•



### 5、数据库设计与原理

* 5.1、数据库设计概述

• 在数据库应用系统中一个核心问题就是设计一个能满足用户要求，性能良好的数据库，这就是数据库设计（Database design）

• 数据库设计的基本任务是根据用户对象的信息需求、处理需求和数据库的支持环境（包括硬件、操作系统与DBMS）设计出数据模式。

• 在数据库设计中有两种方法，

• 一种是以信息需求为主，兼顾处理需求，称为面向数据的方法（data-orented approach）;

• 另一种方法是以处理需求为主，兼顾信息需求，称为面向过程的方法（process-oriented）。

• 数据库设计目前一般采用生命周期（life cycle）法，即将整个数据库应用系统的开发分解成目标独立的若干阶段。

• 它们是：需求分析阶段、概念设计阶段、逻辑设计阶段、物理设计阶段、编码阶段、测试阶段、运行阶段、进一步修改阶段。

* 5.1.1、数据库概念设计

• 数据库概念设计的方法有以下两种：

• 1） 集中式模式设计法这是一种统一的模式设计方法，

• 她根据需求由一个统一机构或人员设计一个综合的全局模式。

• 这种方法设计简单方便，它强调统一与一致，适用于小型或并不复杂的部门或单位，而对大型的或语义关联复杂的单位则并不适合。

• 2） 视图集成设计法这种方法是将一个单位分解成若干个部分，先对每个部分作局部模式设计，建立各个部分的视图，然后以各视图为基础进行集成

• 在集成过程中可能会出现一些冲突，这是由于视图设计的分散性形成的不一致所造成的，因此需对视图作修正，最终形成全局模式。

• 视图集成设计法是一种由分散到集中的方法，它的设计过程复杂但它能较好地反映需求，适合于大型与复杂的单位，避免设计的粗糙与不周到，目前此种方法使用比较多。

• 视图设计一般有三种设计次序：

• 自顶向下

• 由底向上

• 由内向外

• 视图集成的几种冲突：

• 命名冲突

• 概念冲突

• 域冲突

• 约束冲突

* 5.1.2、数据库的逻辑设计

• 数据库的逻辑设计主要工作是将E-R图转换成指定RDBMS中的关系模式。

• 首先，从E-R图到关系模式的转换是比较直接的，实体与联系都可以表示成关系，E-R图中属性也可以转换成关系的属性。实体集也可以转换成关系。

• 逻辑设计的另一个重要内容是关系视图的设计，它又称为外模式设计。

• 关系视图是在关系模式基础上所设计的直接面向操作用户的视图，它可以根据用户需求随时创建，一般RDBMS 均提供关系视图的功能。

• 关系视图的主要作用：

• 1）提供数据逻辑独立性；

• 2）能适应用户对数据的不同需求；

• 3）有一定数据保密功能。

* 5.1.3、数据库的物理设计

• 数据库的物理设计的主要目标是对数据库内部物理结构作调整并选择合理的存取路径，以提高数据库访问速度及有效利用存储空间，

• 在现代关系数据库中已大量屏蔽了内部物理机构，因此留给用户参与物理设计的余地并不多，一般的RDBMS中留给用户参与物理设计的内容大致有如下几种：

• 索引设计

• 集簇设计

• 分区设计

# Java语言程序设计

## 第一章 java语言基础

### 一、java概述

* 1、编程语言的发展

• 第一代语言：

• 机器语言

• 第二代语言：

• 汇编语言、单片机MOV A B

• 第三代语言：

• C所有高级语言的基础、面向过程的语言，C++面向过程/对象语言，Java完全面向对象的跨平台语言，.net跨语言的面向对象平台。

* 2、Java发展史

• Java语言出现于1991年，JDK1.0版本正式发布于1995年

• Java的摇篮：SUN Stanford University NetWork

• Java之父：James Gosling(詹姆斯•高斯林)

• JDK1.0、JDK1.1、JDK1.2、JDK1.3、JDK1.4、JDK5.0、JDK6.0、JDK7.0、JDK8.0

• JDK1.4之后-Sun将JDK1.5改名为J2SE5.0，J2SE5.0较以前的J2SE版本有着很大的改过

* 3、Java技术架构体系

• ava代表一个技术体系、根据应用方向的不同分为SE/EE/ME三部分

• a) J2SE：桌面应用程序

• Java SE：（Java Platform,Standard Edition)为创建和运行java程序提供了最基本的环境，是java技术的核心和基础，可以重点学习：集合类、文件操作、I/O流、线程、序列化和CUI窗体编程等内容。

• b) J2EE：web应用程序

• Java EE：（Java Platform，Enterprise Edition)为基于服务器的分布式企业应用提供开发和运行环境，可以重点学习jsp、servlet、jdbc、ejb以及ssh等框架。

• c) J2ME：手机应用程序

• Java ME：（Java Platform，Micro Edition)为嵌入式应用提供开发和运行环境。如手机程序和PDA程序。可以重点学习配置和简表。

* 4、android与java语言有什么关系

• Android：是由google于2007年11月5日宣布的基于linux平台开源的手机操作系统。

• Android-SDK：Android应用程序开（Android SDK引用了大部分的Java SDK）

• Java平台和java语言：

• Java平台：由三部分组成

• 核心的java api（包、框架及类库）

• java字节码（编译且可执行的形式）

• java虚拟机（jvm执行字节码的运行机制）

• Java语言只不过是java平台中的一小部分，其他语言同样可以实现java原则的功能。

• 例如：Groovy（jvm的一个替代语言，它的语法与java语言相似，以精简有趣的方式在java平台处理事务并把类似Python Ruby等强大的功能带到了java世界）、JRuby及JPython等这些语言同样可以编写出运行在jvm上执行的字节码。

* 5、Java特点

• a) Java是一种面向对象的编程语言

• b) Java语言是一种编译解释型语言

• c) 因为Java语言是基于Java虚拟机运行，所以Java语言可以实现跨平台

• d) Java的GC垃圾回收器

• 计算机高级编程语言按其程序的执行方式可以分为：编译型语言和解释型语言。

• 编译型：使用专门的编译器，针对特定操作系统将查询代码一次性翻译成计算机能识别的机器指令。C、c++都属于编译型。

• 解释型：使用专门的解释器，将源程序代码逐条的解释成特定平台的机器指令，解释一句执行语句，类似于同声翻译。ASP、PHP属于解释型。

• Java既不是编译型也不是解释型语言，它是介于编译型和解释型之间的一个结合体。

• 首先采用通用的java编译器将java源程序编译成为与平台无关的字节码文件（.calss文件），然后由虚拟机（jvm）对字节码文件解释。

• 执行。

• 一次编译、多处执行。

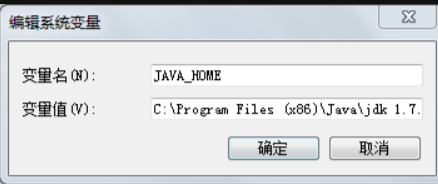
• Jvm是通过jdk中的java.exe来实现的。

### 二、JDK环境变量设置

* 1、设置步骤

• a)在系统变量里点击新建，变量名填写JAVA\_HOME，变量值填写JDK的安装路径

• 比如：“C:\Program Files\Java\jdk 1.7.0\_67”了。



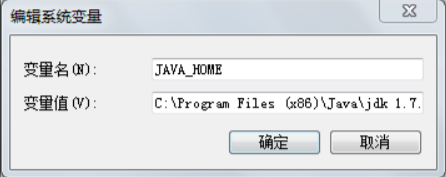
• b)在系统变量里点击新建变量名填写CLASSPATH，变量值填写“.;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME\lib\tools.jar ”。

• 注意不要忘记前面的点和中间的分号。



• c)在系统变量里找到Path变量，这是系统自带的，不用新建。后加上“;%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin”。

• 注意前面的分号。



• d)cmd中运行java和javac,没出错就配置成功了。

• 例如：



* 2、jdk jre 之间的区别

• Jre：

• 是java runtime environment 的简称，即java运行时环境。主要有java虚拟机，java平台核心类和若干支持文件组成。Jre不包括开发工具、编译器、调试器以及其他工具。

• Jre由classLoader（类加载器）负责查找和加载程序引用到基础类库和其他类库。基础类库会自动到rt.jar中加载，操作系统通过path环境变量来操作jre并确定基础类库问价rt.jar的位置。其他类库classLoader在环境变量classPath指定路径中搜索。

• Jdk：

• 是java development kit 的简称，即java开发工具包，提供了java的开发环境和运行环境。Jdk中除了包含jre的全部内容以外，还包括开发者用以编译、调试和运行的java的工具：

• 常见的工具：

• Javac.exe 编译器，用于将java源程序转成字节码

• Jar.exe 打包工具

• Javadoc.exe 文档生成器

• Jdb.exe debugger，差错工具

• Java.exe 运行编译后的java程序

• Javah.exe 可调用java的过程，c过程

• Javap.exe java反汇编器

• ......

• 如果只需要运行java程序/Applet程序（是一种特殊的java程序，本身不能单独运行，需要嵌入在html中，借助浏览器/appletviewer解释执行），下载安装jre环境即可。如果要自行开发，需要下载jdk，djk中附带有jre环境。

• java -version 查看的是系统安装的JDK，它找的路径是path中：C:/WINDOWS/system64/java.exe；

• javac -version 查看的是环境变量设置的JDK版本；

* 3、cmd命令运行java程序

• 一个java程序，由一个或多个类组成。一个程序只有一个main()主入口方法。

• //例如：

• public class Test{

• public static void main(String[] args){

• System.out.println("hello world!");

• }

• }

• 把Test.java 在D盘，而Test.class在C盘。而D盘中没有Test.class然后在cmd中：

• D:\>set classpath=c: //设置class开始查找的路径，echo %classpath%查看

• D:\>java Test

• hello world!

• //带包的,如果当前目录在D盘，知道.java文件在哪,如果放在D:\盘

• package com.cn;

• public class Test{

• public static void main(String[] args){

• System.out.println("hello world!");

• }

• }

• 方式一：从当前目录开始

• D:\>javac -d . Test.java //-d打包，则生成的class,会放在D:\com\cn目录

• D:\>java com.cn.Test //即可运行

• 方式二：指定目录开始

• D:\>javac -d D:\myclass Test.java //-d打包，则生成的class,会放在

• D:myclass\com\cn目录

• D:\>set classpath=D:\myclass

• D:\>java com.cn.Test //即可运行

* 解释：

• 在window中“ .”指的是当前路径，linux中也是,也就是说从当前目录开始查找class文件

• （classPath：告诉jvm要使用或者执行的class放在什么路径上）

• （dos中用echo %path%可以查看环境变量配置）

• % % 取JAVA\_HOME=....的这个值，不写就是字符串了

• Path任何路径下都可以识别java/javac命令

• dt.jar是关于运行环境的类库，主要是swing包

• tools.jar是关于一些工具类的类库

• rt.jar包含了jdk的基础类库

• Jdk目录下的src.zip文件：

• 是java提供的API类的源代码压缩文件。

### 三、Java语言基础

* 1、java中的标识符

• 标识符是用于标识某一事物的符号，java程序中的类名、方法名和变量名都属于标识符。

• 规则：

• a)可用英文字母、下划线、$符合数字构成但不能以数字开头（其他特殊字符均不能识别）

• b)Java是大小写敏感的，标识符也不例外

• c)避开java语言中默认的关键字或保留字

• d)标识符没有长度限制

• 习惯：

• 类名：每个单词首字母大写，其他小写

• 接口：每个单词首字母大写，其他小写

• 方法：以小写字母开头，多个单词，从第二个开始首字母大写

• 变量：以小写字母开头，多个单词，从第二个开始首字母大写

• 常量：全部大写，多个单词以\_连接

• 包名：全部小写

* 2、java中的关键字

• 定义：被java语言赋予了特殊含义的单词

• 特点：关键字中所有的字母都是小写

• 用于定义数据类型的关键字

• void

• byte

• short

• int

• long

• float

• double

• char

• boolean

• 用于定义数据类型值的关键字

• true

• false

• null

• 用于定义流程控制的关键字

• if

• else

• switch

• case

• default

• while

• do

• for

• break

• continue

• return

• 用于定义访问权限的关键字权限

• private

• default

• protected

• public

• 定义类、接口、函数、变量修饰符、类与类之间关系的关键字

• class

• interface

• abstract

• static

• final

• synchronized

• extends

• implement

• 用于定义建立实例及引用实例、判断实例的关键字

• new

• this

• super

• instanceof

• 用于异常处理的关键字

• try

• catch

• finally

• throw

• throws

• 用于包的关键字

• package

• import

• 其他关键字

• native

• strictfp

• transient

• volatitle

• assert

• 保留字:

• goto

• const

* 3、注释

• Java中的注释格式：

• •单行注释：

• 格式： // 注释文字

• •多行注释：

• 格式： /\* 注释文字 \*/

• •文档注释：

• 格式：/\*\* 注释文字 \*/

* 4、java中的数据类型

•



• 注：整数默认：int 小数默认：double

• long a=8888l(L);

• float b=8888.88f(F)

• char类型采用的是Unicode编码，无论是字母还是中文，在java中都是占用两个字节。

• 解决double和float精度不准，使用BigDecimal类。前者多用于计算科学/工程计算。后者多用于商业计算。

• JDK5.0中其实引用类型中还有一种：就是枚举。枚举就是将新的类型做一个限定。

• 例如：

• public enum Sex{

• MALE("男"),FEMALE("女");

• private final String value;

• //构造器默认也只能是private, 从而保证构造函数只能在内部使用

• Sex(String value) {

• this.value = value;

• }

• public String getValue() {

• return value;

• }

• public static void main(String[] args) {

• System.out.println(Sex.MALE.getValue()); //男

• }

• //这就是枚举，这里是将Sex所有可能的取值都列举到这个枚举中，如果一个属性定义为枚举类型，只能到这个枚举中区寻找他的值！

* 5、java类型转换

• 1）自动转换（也叫隐式类型转换）

• a)转换双方的类型必须兼容:如：int 和long 兼容 但是和boolean不兼容

• b)目标类型比原类型范围要大

• 2）强制转换（也叫显示类型转换）

• 语法：（欲转的类型）变量名

• 常用数字转字符：

• 方式一：String.valueOf(int a)

• 方式二：Integer.toString(int a)

• 方式三：String str = num + "";

* 6、运算符

• 1)运算符类型

• 算数运算符

• + - \* / % ++ --

• 关系运算符

• > >= < <= == !=

• 逻辑运算符

• ! & | ^ && (短路与) | |（短路或）

• 位运算符

• ~（取反）&（按位与）| ^ >>(右移) <<(左移) >>>(无符号移)

• 其他运算符

• = += -= xy:z instanceof new

• 2)运算符优先级

• 1、分隔符

• . [] () , ;

• 2单目运算符

• +（正） -（负） ~（取反） ！ ++ --

• 3、创建对象（类型转换）

• New 强制转换

• 4、乘/除

• \* / %

• 5、加/减

• + -

• 6、位移

• << >> >>>

• 7、关系比较

• < > <= >= instanceof

• 8、等于/不等于

• == !=

• 9、按位与

• &

• 10、按位异或

• ^

• 11、按位或

• |

• 12、条件与

• &&

• 13、条件或

• | |

• 14、三目

• :

• 15、赋值

• = += -= <<=

• 3) += 和 + 的区别

• x+= y 和 x = x + y 平时是可以互换通用的。但是 += 除了实现了 + 的功能之外还进行了一次强制类型转换

• 例如:

• short s = 3;

• s += 1; 编译通过

• s = s + 1； 编译不通过

• 因为s+1表达式计算结果为int型，而左边是short型。

• 4) & 和 &&的区别

• &被称为逻辑与，该运算在计算表达式结果时，表达式的每一个操作都要参与计算，然后再得出整个表达式结果。

• &&被称为短路与，表达式从左往右计算时，如果发现一个false值，就放弃运算直接将false结果返回。

• &&效率高于&

• 5）i++ 和++i区别

• 如果i++ 和 ++i 作为一条单独的语句执行是等价的。好比在for循环中for（int i=0;i<10;i++）与for（int i=0;i<10;++i）是等价的

• 如果i++和++i作为表达式的一部分，就会有差别，++i表示先将变量+1然后再参与计算，而i++是先参加运算再+1

• 6）其他运算符

• a) <<:左移几位其实就是该数据乘以的几次幂

• b) >>:右移几位，其实就是该数据除以2的几次幂

• 例如：

• 最有效率的方式算出2乘以8等于几？

• 解：2<<3 （2\*2³=16）

• c) ^ ：异或（两个是否相同）

• 例如：

• true ^ true=flase

* 7、break和continue

• break:

• 跳出

• break作用的范围：

• 要么是switch语句，要么是循环语句。

• 记住：当break语句单独存在时，下面不要定义其他语句，因为执行不到。

• break跳出当前循环

• 例：

• for(int i=0;i<5;i++){

• System.out.println("abc"+i);

• break;

• System.out.println("def"+i); //编译无法通过

• }

• continue：

• 继续

• 作用的范围：

• 循环结构。

• 作用：

• 结束本次循环，继续下次循环。

• 如果continue单独存在时，下面不要有任何语句，因为执行不到。

• break语句：

• 应用范围：选择结构和循环结构。

• continue语句：

• 应用于循环结构。

* 8、数组

• 1）一维数组

• 数组的声明方式：

• a)int[] arr = new int [长度]；arr[0]=1;arr[1]=2......

• b)int[] arr ={1,2,3};

• c)int[] arr new int[]{1,2,3};

• 数组的拷贝：

• System.arraycopy(源数组，起始位置，目标数组，存放目标数组位置，复制长度)；

• 数组的排序：

• a)Arrays.sort(数组对象)； //升序排

• b)冒泡排序（重要）

• 原理：

• 一次比较两个，如果他们的顺序错了就把他们位置交换。

• 代码：

• for (int i = 0; i < arr.length - 1; i++) {

• for (int j = 0; j < arr.length - 1 - i; j++) {

• if (arr[j] > arr[j + 1]) { //挨个比，小到大排

• int tem = arr[j];

• arr[j] = arr[j + 1];

• arr[j + 1] = tem;

• }

• }

• }

• c）选择排序（重要）

• 原理:

• 每一趟比较从待排序的数据中选出最大（或最小）的一个元素，按顺序放在已经排好的数列后，直到待排序的数据排完。

• 代码：

• for (int i = 0; i < arr.length - 1; i++) {

• for (int j = i + 1; j < arr.length; j++) {

• if (arr[i] > arr[j]) { //每次挑出最大的，从小到大排

• int tem = arr[i];

• arr[i] = arr[j];

• arr[j] = tem;

• }

• }

• }

• d）插入排序

• 原理：

• 将要排序的数组分成两部分，每次从后面的数组部分取出索引最小的数组元素插入到前面的数组部分的适当位置中，通常把数组的第一个元素作为一组，后面的为一组。

• e）快速排序：

• 应用到了递归。

• f）归并排序、希尔排序、堆排序、基数排序

• 注：以上都属于内部排序。

• （2）二维数组

• 二维数组的定义方式：

• int[][] erw = new int[行数][列数];

• 注:定义二维数组必须指定其行数，列数可以指定，可以不指定。

• 数组名后直接加上length(如arr.length)，所指的是有几行(Row)；

• 指定索引后加上length(如arr[0].length)，指的是该行所拥有的元素，也就是列数目。

* 9、for each循环

• 又称强化的for循环，java5新增，在遍历数组、集合的时候，foreach拥有不错的性能。

• 其基本语法如下：

• for（元素类型T element：循环对象）{语句或块 //循环体}

• 注：

• foreach虽然能遍历数组或者集合，但是只能用来遍历，无法在遍历的过程中对数组或者集合进行修改，而for循环可以在遍历的过程中对源数组或者集合进行修改。

• foreach循环通过iterator实现，使用foreach循环的对象必须实现Iterable接口，所以取到的element没有下标，无法对其进行修改。

## 第二章 java面向对象基础

### 一、基础

* 1、类和对象

• 类：是对一类相同事物的抽象描述。

• 对象：对象是类的一个具体实例。

• 注：类的声明仅创建了一个模板，没有创建实际的对象。

• 类定义格式：

• [修饰符] class <类名> {

• 属性定义：[修饰符] 类型 <属性名> [=初始值]

• 方法定义：<修饰符> <返回类型> <方法名>（[参数列表]）{}

• 构造方法：

• }

* 2、面向对象与面向过程的区别

• 对于面向过程思想，强调的是过程（动作）。 C语言

• 冰淇淋装进冰箱

• 打开冰箱—>存储冰淇淋—>关上冰箱

• 面向对象

• 对于面向对象思想，强调的是对象（实体）。

• 冰箱打开—>冰箱存储—>冰箱关闭

• 特点：

• a)面向对象就是一种常见的思想，符合人们的思考的习惯。

• b)面向对象的出现，将复杂的问题简单化。

• c)面向对象的出现，让曾经在过程中的执行者，变成了对象中的指挥者。

* 3、成员变量和局部变量

• 成员变量：

• 是指在类中的变量，也就是指属性，

• 其作用域是在整个类中有效。

• 成员变量在定义时可以不指定初始值，系统会按其默认原则初始化（但是成员变量被final修饰时为常量,系统不会为它初始化）

• 局部变量：

• 一般是指在方法体内部定义的变量

• 其作用域是在方法块内部有效。

• 局部变量在使用是必须先初始化，否则不能通过编译。

• 注：

• public、protected、private、static等修饰符可用于修饰成员变量，但是不能修饰局部变量，两者都可以被final修饰。

• 成员变量存储在堆内存中，局部变量存储在栈内存中。

• 在类中声明的变量是成员变量，也叫全局变量，放在堆中的。

• 在类中声明的变量即可是基本类型的变量，也可是引用类型的变量：

• 当声明的是基本类型的变量其变量名及其只时放在堆类存中的；

• 当为引用类型时，其声明的变量仍然会存储一个内存地址值，该内存地址值指向所引用的对象。

• 在方法中声明的变，即该变量是局部变量，

• 每当程序调用方法时，系统都会为该方法建立一个方法栈，其所在方法中声明的变量就放在（方法）栈中，当方法结束系统会释放方法栈，其对应在该方法中声明的变量随着栈的销毁而结束，这就局部变量只能在方法中有效的原因。

• Java栈上的所有数据都是此线程私有的。

• 在方法中生明的变量可以是基本类型的变量，也可以是引用类型的变量：

• 当声明是基本类型的变量的时，其变量名及值（变量名及值是两个概念）是放在方法栈中；

• 当声明的是引用变量时，所声明的变量是放在方法的栈中，该变量所指向的对象是放在堆内存中。

* 4、方法

• 静态方法：

• 没有this隐式参数（非静态方法中都会有一个隐式参数this）。

• 因为静态方法不能操作对象。

• 虽然操作静态变量或者静态方法，都会使类初始化，但是静态的属性和方法是类的属性和方法，而非静态的属性和方法是对象的属性和方法。

• 静态方法的用途：

• 静态工厂模式

• 构造方法的名字必须与类名相同。这一特性的优点是符合Java语言的规范，

• 缺点是类的所有重载的构造方法的名字都相同，不能从名字上区分每个重载方法，不容易根据其名字分辨其功能。

• 静态工厂方法的方法名可以是任意的。

• new每次都会创建对象，而静态工厂是否创建对象，则取决于里面如何实现的。

• 例如：

• //返回符合中国标准的日历

• Calendar cal=Calendar.getInstance(Locale.CHINA);

• 例如：

• 我们看名字就知道其功能

• NumberFormat cf = NumberFormat.getCurrencyInstance(); //货币格式

• NumberFormat pf = NumberFormat.getPercentInstance(); //百分比

• 注：

• 方法什么时候设计为静态，什么时候为非静态？

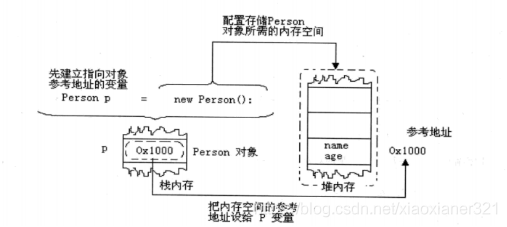
• 对象是类的一个实例，各个对象结构相同，只是属性不同。

• 而静态方法是对象无法调用的。

• 所以，静态方法适合那些工具类中的工具方法，这些类只是用来实现一些功能，也不需要产生对象，通过设置对象的属性来得到各个不同的个体。

* 5、对象的构造

•



• 构造方法所完成的主要工作就是帮助新创建的对象赋初值。

• 如果程序中没有声明构造方法，那么系统会自动声明一个无参构造方法。

• 如果有声明构造方法，那么就不会创建默认的构造方法了。

• 注：

• 构造方法可以是私有，外部无法通过其构造方法产生实例对象。

• 构造方法中可以有return语句。

• 其实void方法后面也可以写return。

* 6、static关键字

• (1)static的特点：

• a)static是一个修饰符，用于修饰成员。

• b)static修饰的成员被所有的对象所共享。

• c)static优先于对象存在，因为static的成员随着类的加载就已经存在了。

• d)static修饰的成员多了一种调用的方式，直接可以通过类名调用。

• e)static修饰的数据是共享的，对象中存储的是特有数据。

• (2)成员变量和静态变量的区别：

• a)两个变量的生命周期不同

• 成员变量随着对象的创建而存在，随着对象的被回收而释放。

• 静态变量随着类的加载而存在，随着类的消失而消失。

• b)调用方式不同。

• 成员变量只能被对象调用。

• 静态变量可以被对象调用，还可以被类名调用。

• c)别名不同

• 成员变量（实例变量）

• 静态变量（类变量）

• d)数据存储位置不同。

• 成员变量数据存储到堆内存的对象中，所以也叫对象的特有数据。

• 静态变量数据存储在方法区（共享数据区）的静态区，所以也叫对象的共享数据。一个对象修该静态属性值后，会影响其他对象。

• e)静态使用的注意事项：

• 静态方法只能访问静态成员。（非静态既可以访问静态，也可访问非静态）。

• 静态方法中不可以使用this或者super关键字。

• 注：main()为程序的主方法，该方法前添加static修饰，是与类相关的。Java解释器运行时将会寻找该方法，因为解释器调用该方法是，还没来得及创建当前类的对象，因此不能定义成与对象相关。这也是为什么主方法是静态的。调用静态方法会使得类被加载。

* 7、单例设计模式

• (1)设计模式：对问题行之有效的解决方式，其实它是一种设计思想。

• 单例设计模式：

• a)解决的问题：就是可以保证一个类在内存中的对象唯一性。

• b)必须对于多个程序使用同一个配置信息对象时，就需要保证对象的唯一性。

• 不允许其他程序用new创建该类对象;在该类中创建一个本类实例;对外提供一个方法让其他程序可以获取该对象。

• (2)步骤

• a)私有化该类的构造函数。

• b)通过new在本类中创建一个本类静态对象。

• c)定义一个共有静态方法，将创建的对象返回。

• (3)饿汉式和懒汉式

• // a)饿汉式

• class Single{ // 类一加载，对象就已经存在了。

• private Single() {//私有化构造}

• private static final Single s = new Single();

• public static Single getInstance() {

• return s; //返回对象的引用，且引用地址不可修改

• }

• }

• // b)懒汉式

• class Single1{

• // 类加载进来，没有对象，只能调用getInstance方法时，才会创建对象。延迟加载

• private static Single s = null;

• private Single() {//私有化构造

• }

• public static Single getInstance() {

• if (s == null) //如果存在就不会创建新的

• s = new Single();

• return s;

• }

• }

### 二、继承

* 1、继承概述

• (1)继承的好处

• a)提高了代码的复用性。

• b)让类与类之间产生关系，给第三个特征多态提供了前提。

• (2)java中支持单继承，不直接支持多继承。

• 单继承：

• 一个子类只能有一个直接父类。

• 因为多个父类中可能有相同成员，调用会产生不确定性。

• 同时子类能继承父类中所有的的公有属性和公有方法，而隐式的继承继承了私有属性。

• （当我们去调用子类的构造器时，会默认先实例化父类的无参构造器。如果父类没有无参构造器，只有有参构造器，则子类中构造器中必须显示的指定父类构造器。当调用父类构造器时，父类中不管是共有属性方法，还是私有属性方法都会被加载到内存当中。只是子类对象访问不到父类的私有属性和方法。）

• 多层继承：

• java支持多层继承。例如：C继承B，B继承A 就会出现继承体系。

• 注：

• 类的方法可以被继承，但是类的构造器不能被继承。

• 从构造本身的结构来看，要与类名相同，这一点上就无法实现继承。

• 从本质上看构造方法不属于类的成员。

• 因此不能被子类继承。

* 2、访问控制

• 在Java中，可以在类、类的属性以及类的方法前面加上一个修饰符，来对类进行一些访问上的控制。

•



* 3、继承详解

• 1）实现继承的格式

• [修饰符] class 子类名 extends 父类

• 2）子类继承父类的成员变量

• 当子类继承了某个类之后，便可以使用父类中的成员变量具体的原则如下：

• 对于子类可以继承的父类成员变量，如果在子类中出现了同名称的成员变量，则会发生隐藏现象，即子类的成员变量会屏蔽掉父类的同名成员变量。

• 如果要在子类中访问父类中同名成员变量，需要使用super关键字来进行引用。而对于私有变量，子类是无权限访问的。

• 例如：

• public class Parent {

• private String sy = "我是父类私有的属性";

• public String gy = "我是父类共有的属性";

• public static String jt = "我是父类静态的属性";

• public final int i = 3; //父类常量

• public static final int A = 4; //父类静态常量

• }

• class Child extends Parent{

• private String sy = "我是子类私有的属性";

• public String gy = "我是子类共有的属性";

• public static String jt = "我是子类静态的属性";

• public final int i = 4; //子类常量

• public static final int A = 5; //子类静态常量

• private String getParent(){

• String message = "父类："+super.gy+" 子类："+this.gy; //(this可省略)得到父类和子类属性

• return message;

• }

• public static void main(String[] args) {

• Parent p = new Child();//p是父类引用,向上转型

• System.out.println(p.gy);//我是父类共有的属性

• System.out.println(p.jt);//我是父类静态的属性

• System.out.println(p.i); //3

• System.out.println(p.A); //4

• //System.out.println(p.sy); //访问受限，访问不到父类私有属性,private只能在同类中

• //System.out.println(p.getParent()); //父类对象中并不存在该方法

• System.out.println(new Child().getParent());//父类：我是父类共有的属性 子类：我是子类共有的属性 可以通过super访问到父类的属性，说明继承了父类的属性

• }

• }

• (3)子类继承父类的方法

• 对于子类可以继承的父类成员方法，如果在子类中出现了同名称的成员方法，则称为覆盖，即子类的成员方法会覆盖掉父类的同名成员方法。如果要在子类中访问父类中同名成员方法，需要使用super关键字来进行引用。

• 父类声明为final的方法不能被覆盖；

• 父类声明为static的方法不能被覆盖，只会被隐藏；

• 注意：

• 隐藏和覆盖是不同的。

• 隐藏是针对成员变量和静态方法的，而覆盖是针对普通方法的。

• 例如：

• public class Parent {

• public void jc(){

• System.out.println("我是父类的方法");

• }

• public static void jt(){

• System.out.println("我是父类的静态方法");

• }

• public final void zz(){

• System.out.println("我是父类的final方法");

• }

• private void sy(){

• System.out.println("我是父类私有的方法");

• }

• }

• class Child extends Parent{

• public void jc(){

• System.out.println("我是子类的方法");

• }

• public static void jt(){

• System.out.println("我是子类的静态方法");

• }

• /\*public final void zz(){//编译无法通过

• System.out.println("我是父类的final方法");//Cannot override the final method from

• }\*/

• private void sy(){

• System.out.println("我是子类私有的方法");

• }

• public void getMessage(){

• super.jc(); //继承了父类的方法

• super.jt(); //继承了父类的方法

• }

• public static void main(String[] args) {

• Parent p = new Child();

• p.jc(); //我是子类的方法 ---父类方法被覆盖

• p.jt(); //我是父类的静态方法 ---没有被覆盖

• //p.sy(); //访问不到

• new Child().sy(); //我是子类私有的方法

• new Child().getMessage();//我是父类的方法//我是父类的静态方法

• }

• }

* 4、this关键字解析

• Java为了解决变量名冲突和不确定性的问题，而引入了关键字this。This代表当前类的一个实例（该类的对象）

• 具体使用情况：

• 返回调用当前方法的对象的引用 return this

• 在构造方法中调用当前类中的其它构造方法

• 当前方法参数名和成员变量名相同时，用this区分参数名和成员变量名

• 例如：

public class A{

String name;

public A test(){

return this;//这里的this就代表这个类的对象

}

public A (String name){

this();//这里调用了无参构造

this.name = name;

}

}

• return this; 对于 A a1 = new A(); this 代表a1

• A a2 = new A(); this 代表a2

• this（）代表无参构造

• this(参数)代表有参构造

• this.属性 代表成员变量名

• //注：不能在构造函数之外的任何函数中调用构造函数，在一个构造函数中只能调用一个构造函数，且必须处于起始位置。

• 注：由于this是指向当前类的实例的引用，因此只能够在实例方法的定义体内使用它，在类方法中，用关键字static声明的方法，不能使用this