NJU2013 年计算机科学与技术基础试卷与答案

科目名称：计算机科学与技术基础（算法导论、软件方法、操作系统、数据库）

一、简单描述 Quicksort 算法的原理与过程。概述（即给出结果和理由，不需要严格的数学证明）最坏情况以及平均情况的时间复杂性分析。（15 分）

算法参考网址：

（1）http://www.cnblogs.com/xwdreamer/archive/2011/06/15/2297000.html

（2）http://blog.csdn.net/feliciafay/article/details/6042034

（3）http://www.cnblogs.com/morewindows/archive/2011/08/13/2137415.html

（4）算法演示：http://www.tyut.edu.cn/kecheng1/site01/suanfayanshi/list.asp?id=7

1. Quicksort 算法的原理

Quicksort 算法是由冒泡排序算法改进而得到的，其基本原理是：

（1）在待排序的 n 个元素中任选一个元素作为基准（pivot），然后对数组 a[n]进行分区操作，通过一趟排序之后将数组 a[n]分割为独立的两部分，使得基准左边元素的值都不大于基准值、基准右边元素的值都不小于基准值，这时作为基本的元素排在这两部分的中间，并使得作为基准的元素调整到排序后的正确位置（或称为记录归位）。

（2）完成一趟快速排序之后，可采用递归方法对所得的两部分子数组分别重复上述快速排序，直至每部分内只有一个元素或元素为空为止，这时每个数组元素都将被排在正确的位置，整个数组达到有序。

因此，快速排序算法的核心是分区操作，即调整基准元素的位置以及调整调整返回基准的最终位置以便分治递归。

2. Quicksort 算法的过程

```
//将中间元素最为pivot，参见李春葆《数据结构教程（第3版）》P285
public static void quickSortSwap(int[] arry, int start, int end) {
    int i = start, j = end;
    int mid = arry[(start + end) / 2];
    int tmp = 0;
    if (start < end) {
        while (i != j) {
            while (j > i && arry[j] > mid)
                j--;
            while (i < j && arry[i] < mid)
                i++;
            if (i <= j) {
                tmp = arry[j];
                arry[j] = arry[i];
                arry[i] = tmp;
            }
        }
        quickSortSwap(arry, start, i - 1);
        quickSortSwap(arry, j + 1, end);
    }
}
```

一趟快速排序的算法是：

（1）设置两个变量 start、end，排序开始的时候：start=1，end=N；

（2）以第一个数组元素作为关键数据，赋值给 pivot，即 pivot=arry[1]；

（3）从 end 开始向前搜索，即由后开始向前搜索（end--），找到第一个小于 pivot 的值 arry[end]，并与 arry[start]交换,即 swat(arry,start,end)；

（4）从 start 开始向后搜索，即由前开始向后搜索（start++），找到第一个大于 pivot 的 arry[start]，与 arry[end]交换,即 swat(arry,start,end);

（5）重复第 3、4 步，直到 start=end，这个时候 arry[start]=arry[end]=pivot，而 pivot 的位置就是其在整个数组中正确的位置；

（6）通过递归，将问题规模不断分解。将 pivot 两边分成两个数组继续求新的 pivot，最后解出问题。

```
//将首元素最为pivot
public static void quickSortf(int[] arry, int start, int end) {
    int i = start, j = end;
    int tmp = arry[0];
    if (start < end) {
        tmp = arry[i];
        while (i != j) {
            while (j > i && arry[j] > tmp) {
                j--;
            }
            arry[i] = arry[j];
            print(arry);
            while (i < j && arry[i] < tmp) {
                i++;
            }
            arry[j] = arry[i];
            print(arry);
        }
        arry[i] = tmp;
        quickSortf(arry, start, i - 1);
        quickSortf(arry, i + 1, end);
    }
}
```

3. Quicksort 算法的时间复杂度分析

快速排序的时间主要耗费在划分操作上，对长度为 n 的区间进行划分，共需 n-1 次关键字的比较。

（1）最坏时间复杂度

最坏情况是每次划分选取的基准都是当前无序区间中关键字最小（或最大）的记录，划分的结果是基准左边的子区间为空（或右边的子区间为空），而划分所得的另一个非空的子区间中记录数目，仅仅比划分前的无序区中记录个数减少一个。因此，快速排序必须做 n-1 次划分，第 i 次划分开始时区间长度为 n-i+1，所需的比较次数为 n-i（1≤i≤n-1），故总的比较次数达到最大值：

$$C_{\max}=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}=O(n^2)$$

如果按上面给出的划分算法，每次取当前无序区的第 1 个记录为基准，那么当文件的记录已按递增序(或递减序)排列时，每次划分所取的基准就是当前无序区中关键字最小（或最大）的记录，则快速排序所需的比较次数反而最多。

（2）最好时间复杂度

在最好情况下，每次划分所取的基准都是当前无序区的“中值”记录，划分的结果是基准的左、右两个无序子区间的长度大致相等。设 $C(n)$ 表示对长度为 n 的表进行快速排序所需的比较次数，显然它应该等于长度为 n 的无序区间进行划分所需的比较次数 n-1，加上递归地对划分所得的左、右两个无序子区间

（长度$\le \frac{n}{2}$）进行快速排序所需的比较次数$2C(\frac{n}{2})$，假设表长度$n=2^k$，则$C(n)=n-1+2C(\frac{n}{2})$，展开此递归关系式即可求得总的关键字比较次数：

$$C(n)=n-1+2C(\frac{n}{2})=O(n\log_2 n)$$

设$C(1)$表示对长度为 1 的区间进行快速排序所需的比较次数，可将它看作一个常数。实际上，用递归树来分析最好情况下的比较次数更简单。因为每次划分后左、右子区间长度大致相等，故递归树的高度为$O(\log_2 n)$，而递归树每一层上各结点所对应的划分过程中所需要的关键字比较次数总和不超过 n，故整个排序过程所需要的关键字比较总次数$C(n)=O(n\log_2 n)$。

因为快速排序的记录移动次数不大于比较的次数，所以快速排序的最坏时间复杂度应为$O(n^2)$，最好时间复杂度为$O(n\log_2 n)$。

（3）平均时间复杂度

设$C(n)$表示对长度为 n 的区间进行快速排序所需的平均比较次数，平均情况介于最好与最坏情况之间。对于一个长度为 n 的区间执行一次划分操作，将区间分为三部分，设左右两个子区间分别包括 k 和 n-k-1 个元素。通过两次递归调用分别对这两个子区间进行快速排序所需的时间分别为$C(k)$和$C(n-k-1)$。由于基准元素位于 0 到 n-1 各位置上的可能性相等，则

$$C(n)=n+1+\frac{1}{n}\sum_{k=0}^{n-1}(C(k)+C(n-k-1))=n+1+\frac{2}{n}\sum_{k=0}^{n-1}C(k)$$

最后计算可得快速排序的平均时间复杂度，即：

$$C(n)=O(n\log_2 n)$$

（4）空间复杂度

最小空间复杂度：快速排序算法中一趟使用了 i，j 和 tmp 等 3 个辅助变量，为常量级，若每一趟排序都将记录序列（区间）均匀地分割为两个程度接近的子序列（区间）时，其深度为$O(\log_2 n)$，所需栈空间为$O(\log_2 n)$。

最大空间复杂度：递归树的高度为$O(n)$，所需栈空间为$O(n)$。

平均空间复杂度：$O(\log_2 n)$。

二、简述什么是近似算法，什么是随机算法，如何评价这两类算法的优劣？（10 分）

1. 近似算法

近似算法是计算机科学中算法研究的一个重要方向。所谓“近似”，就是指结果不一定是最优的，但

是也在可以承受的范围内，而且可以比精确求解消耗更少的资源。这里的资源是计算复杂性理论中的标准，可以是时间，空间或者询问次数等。在计算复杂性理论中的某些假设下，比如最著名的$P \neq NP$假设下，对于一些可已被证明为 NP 完全的优化问题，无法在多项式时间内精确求到最优解，然而在现实或理论研究中，这类问题都有广泛的应用，在精确解无法得到的情况下，转而依靠高效的近似算法求可以接受的近似解。

许多组合优化问题都是 NP-难的。因此在 P≠NP 的情况下，这些 NP-难问题没有多项式时间的算法，如何解决这些问题，已成为当前研究的重要问题。前一章我们已提出解决这一问题的途径之一，是求其 NP-难问题的一个“较好”的可行解，或者说求一个“近似”最优解．何为“近似”最优解？

在计算复杂性理论中的某些假设下，比如最著名的 P≠NP 假设下，对于一些可已被证明为 NP 完全的优化问题，无法在多项式时间内精确求到最优解，然而在现实或理论研究中，这类问题都有广泛的应用，在精确解无法得到的情况下，转而依靠高效的近似算法求可以接受的近似解。

定义：设 D 是一个最优化问题，A 是一个算法，若把 A 用于 D 的任何一个实例 I，都能在 I 的多项式时间内得到 I 的可行解，则称算法 A 为问题 D 的一个近似算法，其中 I 表示实例 I 的规模或输入长度。

定义 8．1．1．　设 D 是一个最优化问题，A 是一个算法，若把 A 用于 D 的任何一个实例 I，都能在 $|I|$ 的多项式时间内得到 I 的可行解，则称算法 A 为问题 D 的一个近似算法，其中 $|I|$ 表示实例 I 的规模或输入长度，进而，设实例 I 的最优值为 $OP(I)$，而算法 A 所得到实例 I 的可行解之值为 $A(I)$，则称算法 A 解实例 I 的性能比为 $R_A(I)$，而算法 A 解问题 D 的性能比为 $R_A(D)$，同时称 D 有 R_A—近似解．其中

$$R_A(I)=\begin{cases}\dfrac{A(I)}{OP(I)}, & \text{若D为最小化问题.}\\ \dfrac{OP(I)}{A(I)}, & \text{若D为最大化问题.}\end{cases}$$

$R_A(D)=\inf\{r\geq|R_A(I)\leq r,\ I\in D\}$

解同一个问题的近似算法可能有多个，比如 A_1 和 A_2 是解问题 D 的两个近似算法，若 $R_{A1}(D)<R_{A2}(D)$，则称算法 A_1 的性能好于算法 A_2，显然当 $R_A(D)=1$ 时，则算法 A 是解 D 的精确算法．

近似算法的性能

若一个最优化问题的最优值为 c*，求解该问题的一个近似算法求得的近似最优解相应的目标函数值为 c，则将该近似算法的性能比定义为

$$\eta = \max\left\{\frac{c}{c^*},\frac{c^*}{c}\right\}$$

在通常情况下，该性能比是问题输入规模 n 的一个函数 ρ(n)，即

$$\max\left\{\frac{c}{c^*},\frac{c^*}{c}\right\}\leqslant \rho\ (n)$$

该近似算法的相对误差定义为：

$$\lambda=\left|\frac{c-c^*}{c^*}\right|$$

若对问题的输入规模 n，有一函数 ε(n)使得

$$\left|\frac{c-c^*}{c^*}\right| \leqslant \varepsilon(n)$$

则称 ε(n)为该近似算法的相对误差界。近似算法的性能比 ρ(n)与相对误差界 ε(n)之间显然有如下关系：

$$\varepsilon(n) \leqslant \rho(n)-1$$

2. 随机算法

随机算法（randomized algorithm），是在算法中使用了随机函数，且随机函数的返回值直接或者间接地影响了算法的执行流程或执行结果。就是将算法的某一步或某几步置于运气的控制之下，即该算法在运行的过程中的某一步或某几步涉及一个随机决策，或者说其中的一个决策依赖于某种随机事件。

随机算法，又叫概率算法，随机算法是算法本身包含了随机数生成器的算法。根据《算法导论（中文第二版）》描述，在进行算法分析的时，有时可以在获得了一定输入分布信息之后对输入的分布进行一定的假定，在此基础上进行平均情况分析得到算法的时间复杂度。然而有时候无法获得输入分布的信息，这时可以在算法本身增加一定的随机性，继而实现对算法进行平均情况分析。通过设计随机算法有效地避免较多的较坏情况输入的出现，从而提高算法的平均情况下的性能。

有一些问题，用确定性的算法在可以忍受的时间内无法计算出结果，但可以用随机算法在可以很短的时间内得到基本可信的结果。举一个简单的例子，计算平面左边上区域 A 是否包含着区域 B。用确定性算法的话，计算两个不规则区域的包含问题是很复杂的。我们考虑用随机算法，随机在 B 中选择一个检测点 P，然后检测点 P 是否在区域 A 内。若不在区域 A 内，则得到不包含的结果。若在区域 A 内，则随机选择另外一个节点进行同样的检测，一共选址 K 个检测点。若 K 个点均在 A 内，则得到包含的结果。该随机算法的准确率随着 K 的增大而增大。随机算法并不要求对每一输入都计算出计算结果，但可以将错误率缩小到可以忽略的程度。

解问题 P 的随机算法定义为：设 I 是问题 P 的一个实例，用算法解 I 的某些时刻，随机选取 $b \in I$，由 b 来决定算法的下一步动作。

优点：

（1）执行时间和空间，小于同一问题的已知最好的确定性算法；

（2）实现比较简单，容易理解。

随机算法分为两类：

（1）拉斯维加斯（Las Vegas）算法，要么给出正确解，要么无解。无解时，需要再次调用该算法。要么给出问题的正确答案，要么得不到答案。反复求解多次，可使失效的概率任意小。

（2）蒙特卡罗（Monte Carlo）算法，总能给出解，但可能是正确解也可能是错误解。多次调用该算法可降低错误率。总能得到问题的答案，偶然产生不正确的答案。重复运行，每一次都进行随机选择，可使不正确答案的概率变得任意小。

至于用哪种算法取决于具体的应用。对于一些不允许出错的应用则采用 Las Vegas 算法，但若是允许小概率错误的话，Monte Carlo 算法更加省时。

随机算法的优点：对于某一给定的问题，随机算法所需的时间与空间复杂性，往往比当前已知的、最好的确定性算法要好；到目前为止设计出来的各种随机算法，无论是从理解上还是实现上，都是极为简单的；随机算法避免了去构造最坏情况的例子。

3. 算法评价

三、在模型驱动的软件体系结构中，平台无关模型（Platform Independent

Models，PIM）和平台相关模型（Platform Specific Models，PSM）分别描述软件的哪些方面？软件运行时依赖的网络拓扑结构信息应该在PIM还是PSM中描述？（15分）

对象管理组织(Object Management Group, OMG)提出了基于模型驱动的体系结构(Model Driven Architecture, MDA)技术，将软件系统建立在各种模型的基础上，通过模型的变换来驱动系统的开发，设计轻便的、可操作的应用程序。

模型驱动体系(MDA)是对象管理组织定义的一个软件开发框架，为软件系统设计提供了一条新途径。在 MDA 中，MDA 将模型区分为平台无关模型(Platform Independent Model, PIM)和平台相关模型(Platform Specific Model, PSM)，模型不再仅仅是描绘系统、辅助沟通的工具，而是软件开发的核心和主干。它的核心思想是抽象出与实现技术无关、完整描述业务功能的平台独立模型(PIM)，针对不同实现技术制定多个映射规则，通过这些映射规则及辅助工具将 PIM 转换成与具体实现技术相关的平台相关模型(PSM)，最后，将 PSM 转换成代码。

PIM 与 PSM 这两种模型是 MDA 体系结构中对于一个系统的不同视角的模型描述，它们之间是抽象和求精的关系。与具体实现技术无关，PIM 可以被多种实现技术复用，当技术平台发生变迁时，PIM 不必做改动；PIM 可以更加精确地体现系统的本质特征，对跨平台互操作问题进行建模非常容易，因为使用与平台无关的通用术语，语义表达会更加清晰。PIM 与 PSM 两个模型之间通过模型映像机制相互映像，从而保证了模型的可追溯性，这也体现了 MDA 软件开发过程是一个模型至顶向下、逐步求精的过程。

MDA 将软件系统的模型分离为平台无关模型 PIM 和平台相关模型 PSM，同时又能通过映射规则将它们统一起来，以这样的方式试图去摆脱需求变更所带来的困境。在将模型自动转换成代码的研究中，平台相关模型 PSM (Platform Specific Model)是针对特定平台的模型，在 MDA 框架中，首先使用平台无关的建模语言来搭建平台无关的模型 PIM 然后根据特定平台和实现语言的映射规则，将 PIM 转换以生成平台相关的模型 PSM，最终生成应用程序代码和测试框架。

1. 平台无关模型（Platform Independent Models，PIM）

平台无关模型(PIM)是**模型驱动架构(MDA)**的核心，在基于 MDA 的软件开发过程中，平台无关模型(PIM)扮演着核心角色，MDA 通过建立 PIM 来生成实际执行系统，也通过修改 PIM 来修改执行系统。

模型驱动架构(MDA)是对象管理组织(OMG)为解决软件开发危机提出的软件开发架构，其基本思想是将模型作为软件开发的核心产品，严格区分系统的功能规约与实现细节。MDA 的典型开发过程如图 1 所示，分为 3 步：首先对应用领域建模生成 PIM，然后将 PIM 转换为一个或多个 PSM，最后将 PSM 转换成代码。其中，PIM 在描述纯粹关注技术的业务逻辑中扮演了中心角色，PIM 建模是至关重要的一步，关系着 MDA 开发的成败。

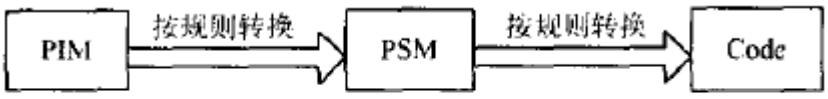
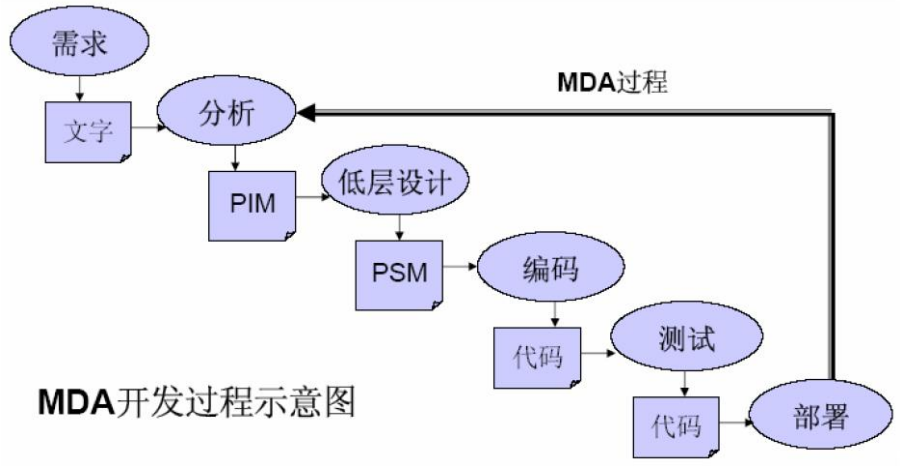


图 1　MDA 开发过程



MDA 的核心是 PIM，它是一个软件系统功能和结构的形式化规范，与具体实现技术和硬件环境无关，所以 PIM 的建模语言也应是平台无关的。

平台无关模型（Platform Independent Model）类似于系统分析模型，它处于中间抽象层次，关注系统的整个架构实现，但却忽略掉与平台相关的部分。平台独立模型可以转换成多个平台相关模型；平台独立模式(PIM)是一种高阶抽象的模式，该模式与开发技术独立。PIM 是系统分析与设计结果的重要产出，主要根据需求塑模(RM)的结果从如何支援企业运作的观点描述一个软件系统，并不涉及描述系统开发与运作之平台。

2. 平台相关模型（Platform Specific Models，PSM）

四、什么叫做软件体系结构？（10 分）

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组组合连接起来。软件体系结构包括定义能满足所有技术和业务要求的结构化解决方案，同时优化性能、安全性和可管理性等常见的质量特性。它需要根据各种因素做出一系列决策，而每项决策都会对软件的质量、性能、可维护性和全面成功产生相当大的影响。

这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本上得到保持。

软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。软件体系结构不仅指定了系统的组织结构和拓扑结构，并且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理。

一个程序和计算机系统该软件体系结构是指系统的一个或者多个结构。结构中包括软件的构件，构件的外部可见属性以及他们之间的相互关系。体系结构并非可运行软件，而是一种表达，使软件工程师能够（1）分析设计在满足规定需求方面的有效性（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案（3）降低与软件构造相关的风险。

五、解释题（10 分）

1. 内核

内核是操作系统最基本的部分，是为众多应用程序提供对计算机硬件的安全访问的一部分软件，这种访问是有限的，并且内核决定一个程序在什么时候对某部分硬件操作多长时间。

2. 进程

进程是具有独立功能的程序在一定数据集合上的一次执行过程，是操作系统进行资源分配和调度的基本单位，是一个程序及其数据在处理器上顺序执行所发生的活动。

进程的基本特征

（1）结构性

进程包含有描述进程信息的数据结构和运行在进程上的程序。操作系统用进程控制块描述和记录进程的动态变化过程；进程的数据结构包含进程控制块、程序块和代码块。

（2）动态性

进程是程序在数据集合上的一次执行过程，具有生命周期，由创建而产生，由调度而运行，由结束而消亡，是一个动态推进的过程。

（3）并发性

在同一段时间内，若干个进程可以共享一个处理器。进程的并发性能够改进系统的资源利用率，提高计算机的效率。

（4）独立性

在操作系统管理上，进程是一个独立的资源分配单位，进程可以在创建时获取资源，也可以在运行过程中获取资源。操作系统为进程分配各种资源，如处理器和内存地址空间等。

（5）异步性

在计算机环境中，处理器的数量总是小于进程的数量，多个进程被强制分享同一个处理器，进程以交替方式被处理器执行。进程的这种执行方式为异步性。

进程与程序的关系

（1）进程是一个动态的概念，强调的是程序的一次“执行”过程；程序则是一组有序指令的集合，在多道程序设计环境下，它不涉及“执行”，是一个静态的概念。

（2）不同进程可执行同一个程序。由进程的定义可知，区分进程的条件一是所执行的程序，二是数据集合。即使多个进程执行相同的一个程序，只要它们运行在不同的数据集合上，它们就是不同的进程。

（3）每个进程都有自己的生命期。进程的本质是程序的一次执行过程，当系统要完成某项工作时，它就“创建”一个进程，以便执行事先编写好的、完成该工作的那段程序。程序执行完毕、完成预定的任务后，系统就“撤消”这个进程，收回它所占用的资源。一个进程创建后，系统就感知到它的存在；一个进程撤消后，系统就无法再感知到它。于是从创建到撤消，这个时间段就是一个进程的“生命期”。

（4）进程之间具有并发性。在一个系统中，同时会存在多个进程，与它们对应的多个程序同时在系统中运行，轮流占用 CPU 和各种资源。这正是多道程序设计的初衷，说明这些进程在系统中并发执行着。

（5）进程间会相互制约。由于进程是系统中资源分配和运行调度的单位，因此在对资源共享和竞争中，必然会相互制约，影响了各自向前推进的速度。

线程

在进程的基础上需要提出新的并发机制，需要将进程作为资源分配的单位和调度单位分离开来，让进程只作为资源分配的单位，而用新的概念—线程，作为调度的基本单位。

线程是操作系统进程中能够独立执行的实体，是进程的组成部分，是处理器调度的基本单位。

在一个进程中允许多个并发执行的线程，同一进程的线程共享进程获得的内存空间和资源，但不独立拥有进程资源。

线程是操作系统中的基本调度和分派单位，线程具有唯一的标志符和线程控制块。线程控制块中包含有线程的一切私有信息。

线程与进程的区别

- 进程是拥有资源的基本单位，线程则是程序执行的基本单位。
- 不同进程的地址空间相互独立，同一进程的各线程共享同一个地址空间。

- 不同进程间的通信，须使操作系统提供的进程通信机制。同一进程各线程间的通信，可直接通过访问共享的进程地址空间实现。
- 不同进程间的调度切换，系统要花费很大开销；同一进程的线程间切换，无须转换地址空间。
- 多个进程间可并发执行，多个线程间也可并发执行。

进程和线程都具有并发性、异步性、结构性；不同的是进程具有独立性，是独立的资源分配和调度单位，而线程具有共享性，所有属于同一进程的线程共享进程的资源；

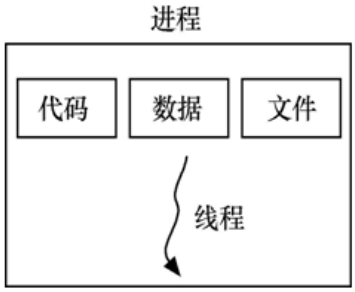


图1-1 单线程进程

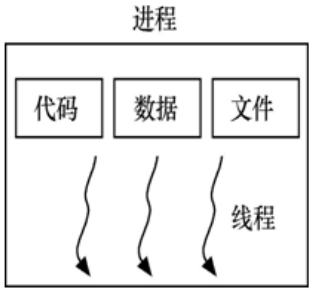


图1-2 多线程进程

作业（Job）：是将命令、程序和数据按照预先确定的次序结合在一起，并提交给系统的一个组织单位。

3. 页表

页表（page table）是系统为保证进程的正确运行而建立的页面映像表。分页转换功能由驻留在内存中的表来描述，该表称为页表，存放在物理地址空间中。用来将虚拟地址空间映射到物理地址空间的数据结构称为页表。

4. 信号量

信号量(Semaphore)，有时被称为信号灯，是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被并发调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。

信号量(Semaphore)-- 相当一个信号灯，程序里是一个非负整数，表示状态。可以用来保护两个或多个关键代码段，这些关键代码段不能并发调用。在进入一个关键代码段之前，线程必须获取一个信号量。

5. 页面替换

在地址映射过程中，若在页面中发现所要访问的页面不再内存中，则产生缺页中断。当发生缺页中断时操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。而用来选择淘汰哪一页的规则叫做页面置换算法。

六、简答题（15 分）

1. 程序员编程可利用操作系统的什么接口，可用哪些功能（3 种以上）？

操作系统为用户提供两个接口。一个是命令接口，用户利用这些操作命令来组织和控制作业的执行或管理计算机系统；另一个是程序接口，编程人员使用它们来请求操作系统服务。

1．命令接口

使用操作命令进行作业控制的主要方式有两种：脱机方式和联机方式。脱机方式是指用户将对作业的控制要求以作业控制说明书的方式提交给系统，由系统按照作业说明书的规定控制作业的执行。在作业执行过程中，用户无法干涉作业，只能等待作业执行结束之后才能根据结果信息了解作业的执行情况。

2. 程序接口

程序接口由一组系统调用命令（简称系统调用）组成，用户通过在程序中使用这些系统调用命令来请求操作系统提供的服务。

（1）系统调用

所谓系统调用就是用户在程序中调用操作系统所提供的一些子功能。具体地讲，系统调用就是通过系统调用命令中断现行程序，而转去执行相应的子程序，以完成特定的系统功能。系统调用功能完成后，控制又返回到系统调用命令的逻辑后继指令，被中断的程序将继续执行下去。

实际上，系统调用命令不仅可以供用户程序使用，系统程序也要使用系统调用来实现其功能。

对于不同操作系统而言，其所提供的系统调用命令条数、格式以及所执行的功能等都不尽相同，即使是同一个操作系统，其不同版本所提供的系统调用命令条数也会有所增减。通常，一个操作系统提供的系统调用命令有几十乃至上百条之多，它们各自有一个惟一的编号或助记符。这些系统调用按功能大致可以分为设备管理、文件管理、进程控制、进程通信、存储管理几大类。

系统调用命令是为了扩充机器指令，增强系统功能，方便用户使用而提供的。因此，在一些计算机系统中，把系统调用命令称为广义指令。广义指令与机器指令在性质上是不同的，机器指令是用硬件线路直接实现的，而广义指令则是由操作系统提供的一个或多个子程序模块实现的。

（2）系统调用的执行过程

虽然系统调用命令的具体格式因系统而异，但是，从用户程序进入系统调用的步骤及其执行过程大体上是相同的。用户程序进入系统调用是通过执行一条“调用指令”（在有些操作系统中称为访管指令或软中断指令）实现的，当用户程序执行到调用指令时，就中断用户程序的执行，转去执行实现系统调用功能的处理程序。

3．图形用户接口

随着大屏幕高分辨率图形显示设备和多种交互式输入/输出设备（如鼠标、触摸屏等）的出现，图形用户接口于 20 世纪 80 年代后期出现并迅速推广。图形用户接口的目标是通过对出现在屏幕上的对象直接进行操作，以控制和操纵程序的运行。例如，用键盘或鼠标对菜单中的各种操作进行选择，使命令程序执行用户选定的操作；用户也可以通过滑动滚动条上的滑动块在列表框中的选择项上滚动，以使所要的选择项出现在屏幕上，并用鼠标选取的方式来选择操作对象（如文件）；用户还可以用鼠标拖动屏幕上的对象（如某图形或图标）使其移动位置或旋转、放大和缩小。这种图形用户接口大大减少或免除了用户的记忆工作量，其操作方式从原来的记忆并键入改为选择并点取，极大地方便了用户，受到普遍欢迎。目前图形用户接口是最为常见的人机接口形式，可以认为图形接口是命令接口的图形化。

OS 向用户提供了三种接口类型。第一是命令接口，用户利用这些操作命令来组织和控制作业的执行或管理计算机系统。根据对作业控制方式的不同，又将命令结构分为联机命令接口和脱机命令接口。第二种是程序接口，它通常由一组系统调用组成，编程人员使用他们来请求操作系统的服务。第三种是图形用户接口，它可以看成是命令接口的图形化，即通过图形化的界面以更加友好的方式向用户提供服务。

程序员编程可利用操作系统的程序接口。

一个 OS 的功能通常通过它提供的系统调用体现出来。OS 提供的系统调用有以下几类：

（1） **进程控制类系统调用**。这类调用主要用于对进程的控制，如创建和终止进程的系统调用，获得和设置进程属性的系统调用等。

（2） 文件操作类系统调用。这类系统调用用来对文件进行操作的系统调用数目较多，其中包括创建文件、删除文件，打开文件，关闭文件、读文件、写文件、建立目录、移动文件的读/写指针，改变文件的属性等系统调用。

（3） 进程通信类系统调用。这类系统调用被用来在进程间传递消息和信号，其中包括消息系统中的建立连接，接受连接，关闭连接，发送消息，接受消息等系统调用，以及共享存储区通信中的建立共享存储区，与共享存储区建立连接、读取共享存储区，写共享存储区等调用等。

（4） **系统维护类系统调用。**这类调用用来实现对系统的日常维护，其中包括设置和获得系统的挡墙日期和时间、或的进程和子进程所使用的 CPU 时间、设置文件访问和修改的时间，了解内存的使用情况和操作系统的版本号等系统调用。

2. 说出对进程死锁的解决方法（3 种）

答案参考网址：http://www.cnblogs.com/ching/archive/2012/02/01/2334659.html

所谓**死锁**是指多个进程因竞争资源而造成的一种僵局，若无外力作用，这些进程都将无法向前推进。死锁是计算机系统和进程所处的一种状态。**死锁产生的原因有以下两点：**

（1）系统资源不足。例如在上面的例子中，若系统中有两台打印机或者两台输入设备供进程 P1 和 P2 使用，当其中任一个进程提出设备请求后，立即可得到满足，显然不会出现死锁状态。这就是说，如果系统中有足够的资源可供每个进程使用，死锁就不会发生。所以说，产生死锁的根本原因是可供多个进程共享的系统资源不足。

（2）进程推进顺序不当。例如在上面的例子中，若进程 P2 在 P1 提出使用输入设备的要求之前，已经完成了对输入设备和打印机的使用，并且释放了它们；或者是在进程 P2 提出使用打印机的请求之前，进程 P1 已经完成了对输入设备和打印机的使用，并且释放了它们，则均不会发生死锁。

操作系统出现死锁的必要条件包括以下四个：

（1） 互斥条件：一个资源每次只能被一个进程使用，但进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一个进程所占有。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。但进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放。

（3） 非剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。

（4） 循环等待（circular waiting）条件：若干进程之间形成一种头尾相接的循环等待资源关系。存在一个两个或多个进程的循环链，链中每一个进程等待被链中下一个进程占有的资源。即在一定条件下，若干进程进入了相互无休止地等待所需资源的状态。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立。

只要破坏这四个必要条件中的任意一个条件，死锁就不会发生。

一般地，解决死锁的方法分为死锁的预防，避免，检测与恢复三种（注意：死锁的检测与恢复是一个方法）。我们将在下面分别加以介绍。

死锁的预防是保证系统不进入死锁状态的一种策略。它的基本思想是要求进程申请资源时遵循某种协议，从而打破产生死锁的四个必要条件中的一个或几个，保证系统不会进入死锁状态。

〈1〉打破互斥条件。即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等，这是由资源本身的属性所决定的。所以，这种办法并无实用价值。

〈2〉打破不可抢占条件。即允许进程强行从占有者那里夺取某些资源。就是说，当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须 释放所占有的全部资源，以后再重新申请。它所释放的资源可以分配给其它进程。这就相当于该进程占有的资源被隐蔽地强占了。这种预防死锁的方法实现起来困 难，会降低系统性能。

〈3〉打破占有且申请条件。可以实行资源预先分配策略。即进程在运行前一次性地向系统申请它所需要的全部资源。如果某个进程所需的全部资源得不到满足，则 不分配任何资源，此进程暂不运行。只有当系统能够满足当前进程的全部资源需求时，才一次性地将所申请的资源全部分配给该进程。由于运行的进程已占有了它所 需的全部资源，所以不会发生占有资源又申请资源的现象，因此不会发生死锁。但是，这种策略也有如下缺点：

（1）在许多情况下，一个进程在执行之前不可能知道它所需要的全部资源。这是由于进程在执行时是动态的，不可预测的；

（2）资源利用率低。无论所分资源何时用到，一个进程只有在占有所需的全部资源后才能执行。即使有些资源最后才被该进程用到一次，但该进程在生存期间却一直占有它们，造成长期占着不用的状况。这显然是一种极大的资源浪费；

（3）降低了进程的并发性。因为资源有限，又加上存在浪费，能分配到所需全部资源的进程个数就必然少了。

〈4〉打破循环等待条件， 实行资源有序分配策略。采用这种策略，即把资源事先分类编号，按号分配，使进程在申请，占用资源时不会形成环路。所有进程对资源的请求必须严格按资源序号 递增的顺序提出。进程占用了小号资源，才能申请大号资源，就不会产生环路，从而预防了死锁。这种策略与前面的

策略相比，资源的利用率和系统吞吐量都有很大 提高，但是也存在以下缺点：

（1）限制了进程对资源的请求，同时给系统中所有资源合理编号也是件困难事，并增加了系统开销；

（2）为了遵循按编号申请的次序，暂不使用的资源也需要提前申请，从而增加了进程对资源的占用时间。

死锁的避免

上面我们讲到的死锁预防是排除死锁的静态策略，它使产生死锁的四个必要条件不能同时具备，从而对进程申请资源的活动加以限制，以保证死锁不会发生。下面我 们介绍排除死锁的动态策略--死锁的避免，它不限制进程有关申请资源的命令，而是对进程所发出的每一个申请资源命令加以动态地检查，并根据检查结果决定是 否进行资源分配。就是说，在资源分配过程中若预测有发生死锁的可能性，则加以避免。这种方法的关键是确定资源分配的安全性。

1、安全序列

我们首先引入安全序列的定义：所谓系统是安全的，是指系统中的所有进程能够按照某一种次序分配资源，并且依次地运行完毕，这种进程序列 {P1，P2，...，Pn}就是安全序列。如果存在这样一个安全序列，则系统是安全的；如果系统不存在这样一个安全序列，则系统是不安全的。

安全序列{P1，P2，...，Pn}是这样组成的：若对于每一个进程 Pi，它需要的附加资源可以被系统中当前可用资源加上所有进程 Pj 当前占有资源之和所满足，则{P1，P2，...，Pn}为一个安全序列，这时系统处于安全状态，不会进入死锁状态。

虽然存在安全序列时一定不会有死锁发生，但是系统进入不安全状态（四个死锁的必要条件同时发生）也未必会产生死锁。当然，产生死锁后，系统一定处于不安全状态。

死锁的检测和恢复

一般来说，由于操作系统有并发，共享以及随机性等特点，通过预防和避免的手段达到排除死锁的目的是很困难的。这需要较大的系统开销，而且不能充分利用资 源。为此，一种简便的方法是系统为进程分配资源时，不采取任何限制性措施，但是提供了检测和解脱死锁的手段：能发现死锁并从死锁状态中恢复出来。因此，在 实际的操作系统中往往采用死锁的检测与恢复方法来排除死锁。

死锁检测与恢复是指系统设有专门的机构，当死锁发生时，该机构能够检测到死锁发生的位置和原因，并能通过外力破坏死锁发生的必要条件，从而使得并发进程从死锁状态中恢复出来。

2.死锁的恢复

一旦在死锁检测时发现了死锁，就要消除死锁，使系统从死锁状态中恢复过来。

（1）最简单，最常用的方法就是进行系统的重新启动，不过这种方法代价很大，它意味着在这之前所有的进程已经完成的计算工作都将付之东流，包括参与死锁的那些进程，以及未参与死锁的进程。

（2）撤消进程，剥夺资源。终止参与死锁的进程，收回它们占有的资源，从而解除死锁。这时又分两种情况：一次性撤消参与死锁的全部进程，剥夺全部资源；或 者逐步撤消参与死锁的进程，逐步收回死锁进程占有的资源。一般来说，选择逐步撤消的进程时要按照一定的原则进行，目的是撤消那些代价最小的进程，如按进程的优先级确定进程的代价；考虑进程运行时的代价和与此进程相关的外部作业的代价等因素。

此外，还有进程回退策略，即让参与死锁的进程回退到没有发生死锁前某一点处，并由此点处继续执行，以求再次执行时不再发生死锁。虽然这是个较理想的办法，但是操作起来系统开销极大，要有堆栈这样的机构记录进程的每一步变化，以便今后的回退，有时这是无法做到的。

死锁是指两个或多个进程因无休止地互相等待永远不会成立的条件而形成的一种永久阻塞状态。我们知道，死锁大多是因并发进程共享临界资源而引起的，而且是一种与时间相关的错误现象，一般具有不可重现性，因此，需要认真地研究并予以解决。

1、处理死锁的方法有以下三类：

（1）死锁检测和恢复（deadlock detection and recovery）：

死锁检测（deadlock detection）即探查和识别死锁的方法。这种策略并不采取任何动作来使死锁不出现，而是系统事件触发执行一个检测算法。，也即在系统运行过程中，及时地探查和识别死锁的存在，并识别出处于死锁之中的进程和资源等。死锁恢复（deadlock recovery）是指当检测并识别出系统中出现处于

死锁之中的一组进程时，如何使系统回复到正常状态并继续执行下去。死锁恢复常采用下述两种方法。

1)撤消进程即当发现死锁时，就撤消（夭折）一些处于死锁状态的进程，并收回它的占用的资源，以解除死锁，使其它进程能继续运行，或者在提供检查点（checkpoint）信息情况下回退（rolled back）到一个较早的状态。这里有一个开销问题，即撤消哪个（些）进程比较“划算”。

2)挂起进程即当发现死锁时，挂起一些进程，抢占它们占用的资源，使得处于死锁之中的其它进程继续执行。待以后条件满足后，再恢复被挂起的进程。

常利用资源分配图、进程等待图来协助这种检测。

（2）死锁预防（deadlock prevention）：

死锁预防（deadlock prevention）是在系统运行之前，事先考虑防止死锁发生的对策，即在最初设计各种资源调度算法时，就没法防止在系统运行过程中可能产生的死锁。

Havender[1968]提出了若干死锁预防方法。他认为，只要设法破坏产生死锁的任一必要条件，死锁就不会发生。为此，可采用如下策略：

1)每个进程必须一次性地请求它所需要的所有资源。若系统无法满足这一要求，则它不能执行。这是一种预分资源方法，它破坏了产生死锁的第三个必要条件。

2)一个已占有资源的进程若要再申请新资源，它必须先释放已占资源。若随后它还需要它们，则需要重新提出申请。换言之，一个进程在使用某资源过程中可以放弃该资源，从而破坏了产生死锁的第二个必要条件。

3)将系统中所有资源顺序编号，规定进程只能依次申请资源，这就是说，一个进程只有在前面的申请满足后，才能提出对其后面序号的资源的请求。这是一种有序使用资源法，它破坏了产生死锁的第四个必要条件。

（3）死锁避免（deadlock avoidence）：

死锁避免（deadlock avoidence）是在系统运行过程中注意避免死锁的发生。这就要求每当申请一个资源时，系统都应根据一定的算法判断是否认可这次申请，使得在今后一段时间内系统不会出现死锁。这面方最著名的算法首推 Dijkstra[1965]提出的银行家（banker）算法。

3. 简述设备驱动程序的作用和**实现方法**。

英文名为“Device Driver”，全称为“设备驱动程序”是一种可以使计算机和设备通信的特殊程序，可以 说相当于硬件的接口，操作系统只有通过这个接口，才能控制硬件设备的工作，假如某设备的驱动程序未能正确安装，便不能正常工作。

设备驱动程序（英语：device driver），简称驱动程序（driver），是一个允许高级（High level）电脑软件（computer software）与硬件（hardware）交互的程序，这种程序创建了一个硬件与硬件，或硬件与软件沟通的接口，经由主板上的总线（bus）或其它沟通子系统（subsystem）与硬件形成连接的机制，这样的机制使得硬件设备（device）上的数据交换成为可能。

七、关系代数与 SQL（第 1 小题 4 分，第 2 小题 6 分，共 10 分）

设有一个课程教学管理数据库系统，其关系模式如下：

课程 C（课程编号 cno，课程名称 cname，开课系别名称 dname，学时数 hours）

教师 T（教师编号 tno，教师姓名 tname，工作系别名称 dname）

授课 P（教师编号 tno，课程编号 cno，年份 year）

其中：教师可以跨系承担其他系的授课任务。

1. 请用**关系代数**表示下述查询操作

（1）查询在“计算机”系开设的所有课程的列表，结果返回课程编号和课程名称；

$$\Pi_{cno,cname}(\sigma_{dname='计算机'}(c))$$

（2）查询没有承担过授课任务的教师的编号和姓名。

$$\Pi_{tno,tname}(\mathrm{t})-\Pi_{tno,tname}(t \triangleright\triangleleft p)$$

2. 请用 **SQL 语言**表示下列查询操作

（1）统计查询每一位教师的累计授课时数，结果返回教师的教师编号及其累计授课时数；

```
select tno,SUM(hours) from c,p x where c.cno=p.cno group by tno
```

（2）查询没有为自己工作过的院系上课的教师姓名。

相关子查询

```
select distinct tname from t where tno not in (select tno from c where c.dname=t.dname)
```

[第（2）题的答案需要核实!!!]

八、在关系数据库管理系统中，用于实现数据完整性保护的措施有哪些？（7 分）

数据库的完整性是指数据的正确性、一致性和相容性，确保数据库中的数据可以成功和正确地更新，防止错误的数据进入数据库造成无效操作。

数据的完整性包括 4 个方面的内容：实体完整性、域完整性、参照完整性和用户定义完整性。

可以通过各种约束、默认值、规则等数据库对象来保证数据的完整性，保护数据库完整性的措施主要包括：

1）**约束（Constraint）**：是数据库系统提供的自动保持数据库完整性的一种机制，它定义了可输入表或表的单个列中的数据的限制条件。使用约束优先于使用触发器、规则和默认值。约束独立于表结构，可以在不改变表结构的基础上，添加或删除约束。当表被删除时，表所带的所有约束定义也随之被删除。

主键（Primary Key，简称 PK）约束：表的一列或几列的组合的值在表中唯一地指定一行记录，这样的一列或多列称为表的主键，通过它可强制表的实体完整性。

- 主键不能为空，且不同两行的键值不能相同。
- 表本身并不要求一定要有主键，但最好给表定义主键。
- 在规范化的表中，每行中的所有数据值都完全依赖于主键。

外键（Foreign Key，简称 FK）约束：外键约束定义了表与表之间的关系。

- 通过将一个表中一列或多列添加到另一个表中，创建两个表之间的连接，这个列就成为第二个表的外键，即外键是用于建立和加强两个表数据之间的连接的一列或多列，通过它可以强制参照完整性。
- 外键约束的主要目的是控制存储在外键表中的数据。

级联操作（）：根据主键表中数据的修改而对外键表中数据相应地做相同的修改。SQL Server 提供了两种级联操作：级联删除和级联修改

- 级联删除：当主键表中某行被删除时，外键表中所有相关行将被删除。
- 级联修改：当主键表中某行的键值被修改时，外键表中所有相关行的该外键值也将被自动修改为新值。

唯一性约束（Unique）：唯一性约束指定一个或多个列的组合的值具有唯一性，以防止在列中输入重复的值，为表中的一列或者多列提供实体完整性。

- 唯一性约束指定的列可以有 NULL 属性。主键也强制执行唯一性，但主键不允许空值，故主键约束强度大于唯一约束。因此主键列不能再设定唯一性约束。
- 一个表可以定义多个唯一性约束。

检查约束（Check）：对输入列或整个表中的值设置检查条件，以限制输入值，保证数据库的数据完整性。

- 当对具有检查约束列进行插入或修改时，SQL Server 将用该检查约束的逻辑表达式对新值进行检查，只有满足条件(逻辑表达式返回 TRUE)的值才能填入该列，否则报错。
- 可以为每列指定多个 CHECK 约束。

2）默认值（Default）

- 通过定义列的默认值或使用数据库的默认值对象绑定表的列，以确保在没有为某列指定数据时，来指定列的值。
- 默认值可以是常量，也可以是表达式，还可以为 NULL 值。

3）规则（Rule）：规则是数据库中对存储在表的列或用户定义数据类型中的值的规定和限制，是单独存储的独立的数据库对象。

- 规则与其作用的表或用户定义数据类型是相互独立的。
- 规则和约束可以同时使用，表的列可以有一个规则及多个约束。
- 规则与检查约束在功能上相似，但在使用上有所区别。
- 规则的操作包括创建、查看、绑定、松绑和删除等。

4）属性值（域）约束：包括非空值约束、基于属性的检查子句、域约束子句。每个属性都必须对应于一个所有可能的取值构成的域，声明一个属性属于某种具体的域就相当于约束它可以取的值。域约束是完整性约束的基本形式，每当有新数据项插入到数据库中，系统就能方便地进行域约束检测。

5）全局约束：包括基于元组的检查子句、断言

一个关系中给定属性集上的取值也在另一个关系的某一属性集的取值中出现（参照完整性）。当参照完整性约束在数据修改操作时被违反，通常的处理是拒绝执行导致完整性被破坏的操作。

断言：就是一个谓词，它表达了用户希望数据库某时刻满足的一条件，域约束和参照完整性约束是断言的特殊形式。

（4）域完整性

域完整性是关于一个给定列的有效入口，是指数据库表中的列必须满足某种特定的数据约束。域完整性通常用有效性检查来实现的，也可以通过限制数据类型、格式或者可能的取值范围来实现。

（5）触发器（Trigger）

触发器是一条语句，只要指定的事件发生，它会自动被系统执行，是一类靠事件驱动的特殊过程。触发器是定义在表上的一类由时间驱动的特殊过程，类似于约束，但比约束更加灵活，具有更精细和强大的数据控制能力。要设置触发器机制，须满足以下两个条件：

- 指明在什么条件下执行触发器。它被分解为一个引起触发器被检测的事件和一个触发器执行必须满足的条件。
- 指明触发器执行的动作。

因此，一个触发器用来定义一个条件，以及在该条件为真时需要执行的动作。通常，触发器的条件以断言的形式定义，动作以过程的形式定义。一旦把一个触发器输入数据库，只要指定的事件发生，相应的条件满足，数据库系统就会自动执行它。

数据完整性实施途径

数据完整性类型	实施途径	数据完整性类型	实施途径
实体完整性	Primary key	参照完整性	Foreign key
	Unique key		check
	Unique index		triggers
	Identity column		procedure
域完整性	default	用户定义完整性	rule、triggers、procedure、create table 中全部列级和表级约束
	check		
	Foreign key		
	Data type		
	rule		

九、什么是列存储数据库？与传统基于行存储的关系数据库系统相比，列存储数据库有哪些方面的优势？（8 分）

列存储数据库是对应并区别于行存储数据库的一个概念。行数据库是数据按记录存储的关系型数据库，其中每一条数据记录的所有属性都存储在一起，如果要查询一条记录的一个属性值，需要先读取整条记录的数据。列存储数据库是按数据库记录的列来组织和存储数据，数据库中每个表由一组页链的集合组成，每条页链对应表中的一个存储列，而该页链中每一页存储的是该列的一个或多个值。

在列存储体系中，关系表依据列分开存储。假设现有表T，T 共有N 条属性，T_i 为第i 条属性的属性名称，$i\in[1,N]$。在每个T_i上做投影操作，ΠT_i的值与新生成的id号重组，依次读入存储数据块。如图1 所示，原始表中有3列属性ID、Name 和Sex，对各列做投影操作后，将获取的列值与自动生成的ID号进行重组，存入磁盘空间。

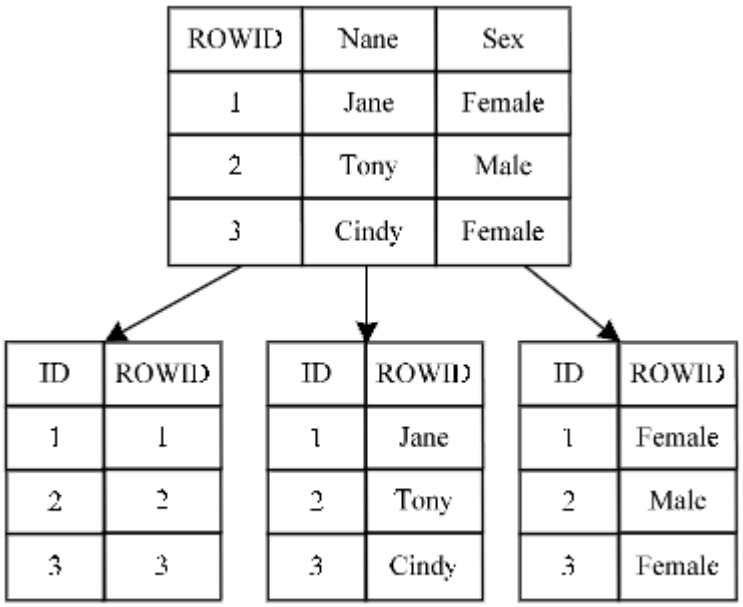


图 1　列存储结构

列存储（Decomposition Storage Model，缩写为 DSM），相对于行存储 (N-ary storage model，NSM)的主要区别在于：DSM 将所有记录中相同字段的数据聚合存储，而 NSM 将每条记录的所有字段的数据聚合

存储。

传统的关系数据库采用 N 元存储模型（NSM，N-ary Storage Model），而对于大多数的查询来说，常常只使用一条记录的一部分数据。因此，为了减少 IO 的消耗，提出了“分解存储模型”（DSM，Decomposition Storage Model）。DSM 将关系垂直分为 n 个子关系，属性仅当需要时才加以存取访问。对于涉及多个属性的查询来说需要额外的开销用于连接子关系。

PAX（Partition Attribute Across）是记录在页面中的混合布局方式，结合了 NSM 和 DSM 的优点，避免了对主存不需要的访问。PAX 首先将尽可能多的关系记录采用 NSM 方式加以存储。在每个页面内，使用按属性和 minipage 进行类似于 DSM 的存储。在顺序扫描时，PAX 充分利用了缓存的资源。同时，所有的记录都位于相同的页面。对于记录的重构操作，仅仅在 minipage 之间进行，并不涉及跨页的操作。

列存储模式极大提高了分析型查询的效率，列存储数据库系统 Sybase IQ 可用于 OLAP、数据仓库、在线分析、数据挖掘等查询密集型应用，。列存储数据库的应用价值来自它对复杂查询的灵活快速以及压缩所带来的存储优势，这使其在数据仓库和商务智能方面具有良好的发展前景，且复杂数据查询效率高、读磁盘少、存储空间少。具体来说，列存储数据库的优势体现在以下几个方面：

（1）列数据库存储方式带来的技术优势

列数据库按列存储的结构，便于在列上对数据进行轻量级的压缩，列上多个相同的值只需要存储一份。压缩能够大量地降低存储成本。

按列存储和压缩的特点，也为列数据库在查询方面带来先天优势，因为若能将更多的数据压缩在一起，则在每次读取时就可以获得更多的数据。很显然，每次读取操作获取更多的数据就意味着更快的处理速度。同时，列数据库的存储特点有利于迅速查询所需要的列。行存储虽然可以比较轻松地添加修改记录，但是会增加许多不必要的读取; 而列存储只需要读取相关数据，并且可以从多个入口写入数据。

每个字段的数据聚集存储，在查询只需要少数几个字段的时候，能大大减少读取的数据量，查询密集型应用的特点之一就是查询一般只关心少数几个字段，而相对应的，行存储模型（NSM）中每次必须读取整条记录；列存储数据库是一个字段的数据聚集存储，容易为这种聚集存储设计更好的压缩/解压算法。

（2）列数据库在数据管理方面的便捷优势

在数据管理方面，行数据库采用的是稠密索引，即当数据库文件中的记录不按照关键码的顺序排列时(比如按照加入的顺序排列) ，需要对每一个记录建立一个索引项。这有两方面的缺点：一是增加所用的存储空间，二是增加数据更新时的代价。正是由于这两方面的问题，在基于行存储的关系型数据库中， 为表的所有列都建立索引就不太现实。这样就出现了下面的问题：如果一个查询语句是基于一个未加索引的列查询， 系统就不得不做全表扫描，导致数据库的查询效率不高。因此，考虑在哪些列加索引并根据应用的需求适时调整，就成为数据库管理员的一项繁重工作。而列数据库因为各条记录在磁盘中是按照关键码值压缩顺序存放的，所以采用的是稀疏索引，即把连续的若干记录分成组(块) ，对一组(块) 记录建立一个索引项。列数据库可以为所有列建立稀疏索引，事实上也都是这么做的。这是因为每个列的值都已被压缩顺序存储，索引只需建立到数据块级，因此索引的存储空间很小，维护费用很低，使得可以以很小的代价给所有的列建立索引。当查询通过索引定位到某一数据块后，就可以使用二分法查找。这样， 数据查询在任何情况下都不会导致全表扫描，从而提高了数据库的查询性能。在列数据库中，后台程序默认地自动为所有的列维护稀疏索引，因此为数据库管理员卸下了建立、管理和维护索引的繁重工作。

列数据库因为在每一列上都采用了轻量的稀疏索引，在插入删除数据时，利用这些索引可以把空洞尽量减小，免除了数据库管理员的大量导入、导出工作。

（3）列数据库在数据挖掘领域的应用优势

数据挖掘应用有着数据量大、主要针对少数列进行复杂查询操作的特点。在列存储模式下，对于列的 DML (Data Manipulation Language，数据操纵语言命令) 操作，仅仅是对列所对应的数据库页链进行数据扫描，不会导致对全表的数据访问，可以有效降低 DML 操作的 I/O 数量。由于数据按列存储，为调整数据

模型所做的新增列、删除列操作不再会遇到数据碎片问题。这是因为在列存储中，由于数据存储以列为单元，列删除或者新增操作是将列所对应的数据库页链从表的页链集合中去除或者新增，不涉及其他列对应的数据库页链。此外，列数据库中的数据按列压缩的特点也同样能够减少挖掘时的 I/O 量。