#

### 2017

#### 数据结构：

##### 1.向量相对于数组有什么优缺点？【感谢李祥 】

**向量相对于数组的优点：**

(1)从所**在内存中的位置**分析C++中**数组**为内置的数据类型，存放在**栈**中，其内存的分配和释放完全由**系统自动完成**；vector，存放在**堆**中，由STL库中**程序负责同志内存**的分配和释放，使用方便；

(2) 从**大小能否变化**分析，**数组**的大小在始化后就**固定不变**，而**vector**可以通过**puish\_back或pop等操作进行变化**；

(3)从**初始化**分析，数组不能将数组的内容拷贝给其他数组作为初始值，也不能用数组为其他数组赋值；而向量可以；

(4) 随机访问方便，即支持[ ]操作符和vector.at();

(5)**数组**在内存中分配的连续空间，**多次分配释放后会有内存碎片**，而vectors是动态增长的，不是连续的，所以不会出现内存碎片;

(6)vector的迭代器能防止出现类似数组越界;

(7)vector 可以确定长度，节省空间，而数组不能确定长度，必须在定义时定义一个很大的空间留给数组，造成内存的浪费。

**向量相对于数组的缺点：**

(1) 在执行效率方面，数组>vector向量。主要原因是vector的扩容过程要消耗大量的时间，在内部进行插入删除操作效率低;

(2) 只能在vector的最后进行push和pop，不能在vector的头进行push和pop;

(3) 当动态添加的数据超过vector默认分配的大小时要进行整体的重新分配、拷贝与释放；

(4)Java的数4可存储任何类型的数组元素，包括数值类型和所有类类型；Java向量只能存储对象类的实例。

##### 2.二叉树计算叶子节点算法，时间复杂度。（可使用任一程序设计语言或伪代码，建议先用自然语言描述算法）感谢邓晓

定义二叉树结构体如下：

typedef int ElemType;

typedef struct Tnode

{

ElemType data;

struct Tnode \*lchild,\*rchild;

};

【方法一】递归算法思想：

先序遍历二叉树，

1）对于每一个根节点，如果为空，则返回0；如果根节点不为空且左右子树都不存在，则返回1；

2）遍历左子树，将左子树节点作为根节点，参照 步骤1）

3）遍历右子树，返回 步骤1）

int Traverse\_Tree(Tree \*T)

{

if(!T) return 0;

if(!T->lchild && !T->rchild) return 1;

else return Traverse\_Tree(T->lchild) +Traverse\_Tree(T->rchild);

}

时间复杂度：对于N个节点的二叉树，遍历时间复杂度 O(N)。

【方法二】非递归算法

思想：利用栈思想。若根结点不为空，则将树根结点入栈；

1）如果栈不为空，栈顶结点出栈，出栈结点的左右子树都不存在，则叶子结点数加1；如果栈为空则退出。

2）如果出栈结点的右子树不为空，将右子树入栈，

3）如果出栈结点的左子树不为空，将左子树入栈，回到1）。

int Traverse\_Tree(Tree \*T)

{

int top = sum =0;

Tnode \*p;

p=T;

Tnode \*stack[Max\_size];

if(T!=NULL )

{

stack[++top] = p;

while(top >0)

{

p= stack[top--];

if(p->rchild == NULL && p->lchild == NULL) sum++;

if(p->rchild!=NULL) stack[++top] = p->rchild;

if(p->lchild !=NULL) stack[++top] = p->lchild;

}

}

return sum;

}

时间复杂度：O(N)。

##### 3.几乎逆序的数组排序用什么排序算法？写出算法，时间复杂度。【感谢pray】

【分析】：这里只分析了考研大纲里的几个排序



由题设可知，几乎逆序也就是判断最坏情况下该用哪种排序方式。

从时间复杂度上看，很显然堆排序和归并排序最优。

从需要的辅助存储看，堆排序需要常数个辅助单元，辅助空间是O(1)，而归并排序（这里分析的是二路归并）一般情况下，由于要求附加与待排记录等数量的辅助空间，也就是O(n)，所以很少用二路归并。

所以针对题设条件采用堆排序为最优解。

【算法思想】1.根据初始数据利用堆调整算法形成初始堆。

2.利用一系列对象交换和重新调整堆进行排序



【算法】

//最大堆得向下调整算法

FilterDown ( **const int** i, **const int** EndOfHeap ) **{**

**int** current = i**; int** child = 2\*i+1**;**

**Type** temp = heap[i]**;**

**while** ( child <= EndOfHeap ) **{ //**最后位置

**if** ( child +1 < EndOfHeap **&&**heap[child] < heap[child+1] )

child = child+1**; //**选两子女的大者

**if** ( temp >= heap[child] )

**break; //temp**排序码大，不做调整

**else {**

heap[current] = heap[child]**; //**大子女上移

current = child**; //**向下继续调整

child = 2\*child+1**;**

**}**

**}**

heap[current] = temp**; //**回放到合适位置

**}**

HeapSort ( ) **{**

**//**对表**heap[0]**到**heap[*n*-1]**进行排序

**//**使得表中各个对象按其排序码非递减有序

**for** ( **int** i = ( CurrentSize-2 ) / 2**;** i >= 0**;** i-- )

FilterDown ( i, CurrentSize-1 )**; //**初始堆

**for** ( i = CurrentSize-1**;** i >= 1**;** i-- ) **{**

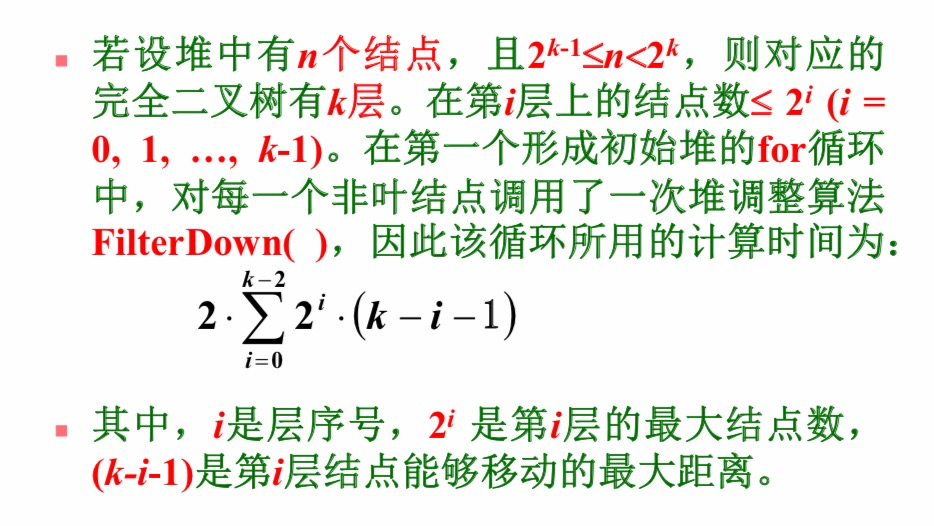
Swap ( heap[0], heap[i] )**; //**交换

FilterDown ( 0, i-1 )**; //**重建最大堆

**}**

**}**

**【堆排序分析】**





##### 4.二叉排序树的2种优化方法，并且介绍这两种方法是怎样优化二叉排序树的【感谢王蜗蜗²⁰¹⁸ 】

答： 红黑树和AVL树两种优化方法。

红黑树提出了为节点增加颜色，红黑是用非严格的平衡来换取增删节点时候旋转次数的降低，任何不平衡都会在三次旋转之内解决，删除节点的rebalance的效率更高，开销更小。

AVL高度平衡，因此AVL的Search效率更高，结构上更加直观，时间效能针对读取而言更高；维护稍慢，空间开销较大。

实际应用中，若搜索的次数远远大于插入和删除，那么选择AVL，如果搜索，插入删除次数几乎差不多，应该选择RB。

#### 计算机原理：

##### 5.Amdahl 硬件优化趋势【感谢小马哥】

决定系统性能的关键因素有那么几点：内存资源、CPU资源、磁盘I/O，那么系统的总体性能是取决于当前最慢的磁盘I/O速度，而不是内存和CPU。这种情况下，只有提高磁盘I/O速度才能对系统的整体性能进行优化.

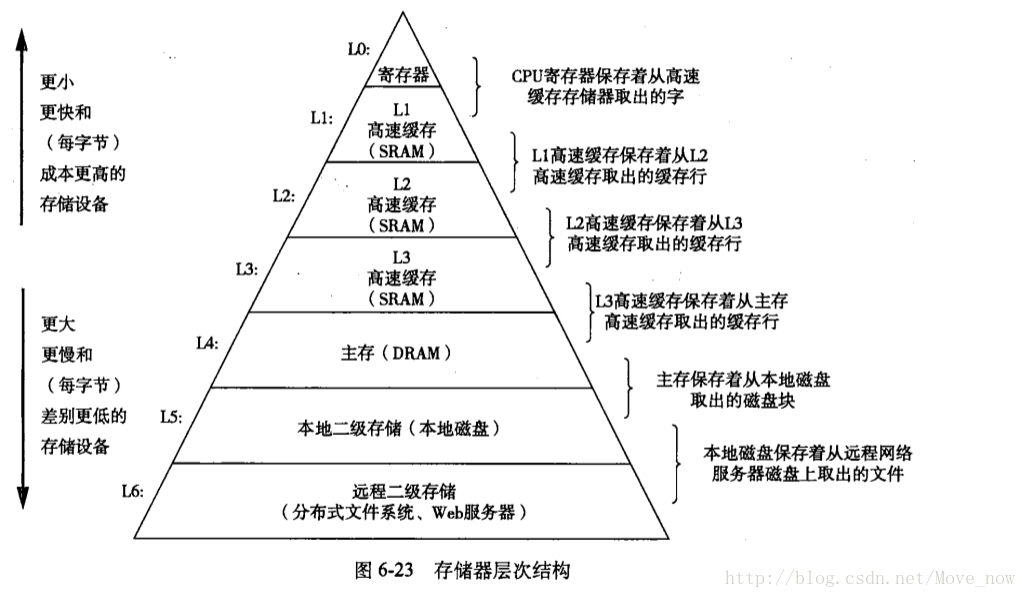
##### 6.流水线是怎样提高性能的，会遇到什么问题，解决方法是什么。【感谢自由守望】

##### 7.软件优化至关重要，软件优化一般有哪些方法？【感谢小马哥】

1. 算法优化，选择适当的数据结构和算法，避免使用糟糕的算法和编码技术。
2. 代码优化，避免限制优化的因素，这样编译器可以产生高效的代码；消除连续的函数调用，消除不必要的存储器引用.
3. 指令优化, 指令优化一般是使用特定的指令集，可快速实现某些运算，同时指令优化的另一个核心思想是打包运算。目前PC上intel指令集有MMX，SSE和SSE2/3/4等，DSP则需要跟具体的型号相关，不同型号支持不同的指令集。intel指令集需要intel编译器才能编译，安装icc后，其中有帮助文档，有所有指令的详细说明。

##### 8.高速缓存【感谢小马哥】

寄存器和内存之间读写速度的差距是很大的，而内存和磁盘之间的读写速度差距也很大。当内存需要传数据给寄存器，假如有这样一个读写速度处于寄存器和内存之间的东西，可以帮忙存储内存要传的数据，然后再把这些u数据传给寄存器，这样不就提高了速度了吗。高速缓存存储器就是这样一个东西，在两端读写速度差距很大时，就插入一个高速缓存存储器，这样的思想就形成了存储器层次结构。



##### 9.性能分析定律【感谢小马哥】

阿姆达尔曾致力于并行处理系统的研究。对于固定负载情况下描述并行处理效果的加速比s，阿姆达尔经过深入研究给出了如下公式：

S=1/(1-a+a/n)

其中，a为并行计算部分所占比例，n为并行处理结点个数。这样，当1-a=0时，(即没有串行，只有并行)最大加速比s=n；当a=0时（即只有串行，没有并行），最小加速比s=1；当n→∞时，极限加速比s→ 1/（1-a），这也就是加速比的上限。例如，若串行代码占整个代码的25%，则并行处理的总体性能不可能超过4。这一公式已被学术界所接受，并被称做“阿姆达尔定律”，也称为“安达尔定理”(Amdahl law)。其实就是Told/Tnew.

##### 10.存储结构是怎样提高性能的，它和局部性的关系是什么。【感谢绾晴】

计算机系统中，一般使用具有层次结构的存储系统来提高性能，主要可分为三个层面：高速缓冲存储器，主存储器和辅存储器

存储器访问存在时间上的局部性和空间上的局部性，存储器的层次结构是依靠存储器访问的局部性实现的存储器的次结构的性能由命中率来衡量，高速缓冲存储器命中率越高，性能就越好

##### 11.虚拟内存的作用，通过什么方式提高虚拟内存的性能。【感谢乔大叶】

虚拟内存提供三个重要的功能, . 第一:它在主存中自动缓存最近使用的存放磁盘上的虚拟地址空间的内容.虚拟内存缓存中的块叫做页 .对磁盘上页的引用会出发缺页, 缺页将控制转移到操作系统中的一个缺页处理程序.缺页处理程序将页面从磁盘复制到主存缓存,如有必要,将写回被驱逐的页.第二:虚拟内存简化了内存管理 ,进而又简化了链接, 在进程间共享数据, 进程的内存分配及程序加载.最后,虚拟内存在每条页表条目中加入保护位 ,从而简化了内存

#### 软件工程：

##### 12.瀑布过程的特点【感谢Ericas 】

瀑布模型（经典生命周期）提出了软件开发的系统化、顺序的方法。其流程从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供一个完整的软件并提供连续的技术支持。

**优点：**

1， 强调开发的阶段性，各阶段具有顺序性和依赖性

2， 强调早期调研和需求分析，推迟编码实现的观点

3， 提供了一个模板，这个模板使得分析、设计、编码、测试和支持的方法可以在该模板下有一个共同的指导

**缺点：**

1， 文档驱动，用户无法及时了解产品的情况

2， 依赖早期调研和需求分析，很难适应在很多项目开始阶段必然存在的不确定性。

3， 流程单一，必须要完成前一阶段的任务，才能进行下一阶段，开发过程中的成功经验无法用于本产品。

4， 测试在后期引入，对于系统存在的重大缺陷，如果在可执行程序评审之前没有被发现，将可能造成重大损失

5， 组织庞大，人员闲置。

**适用范围：**

需求确定，工作能够采用线性的方式完成的软件。

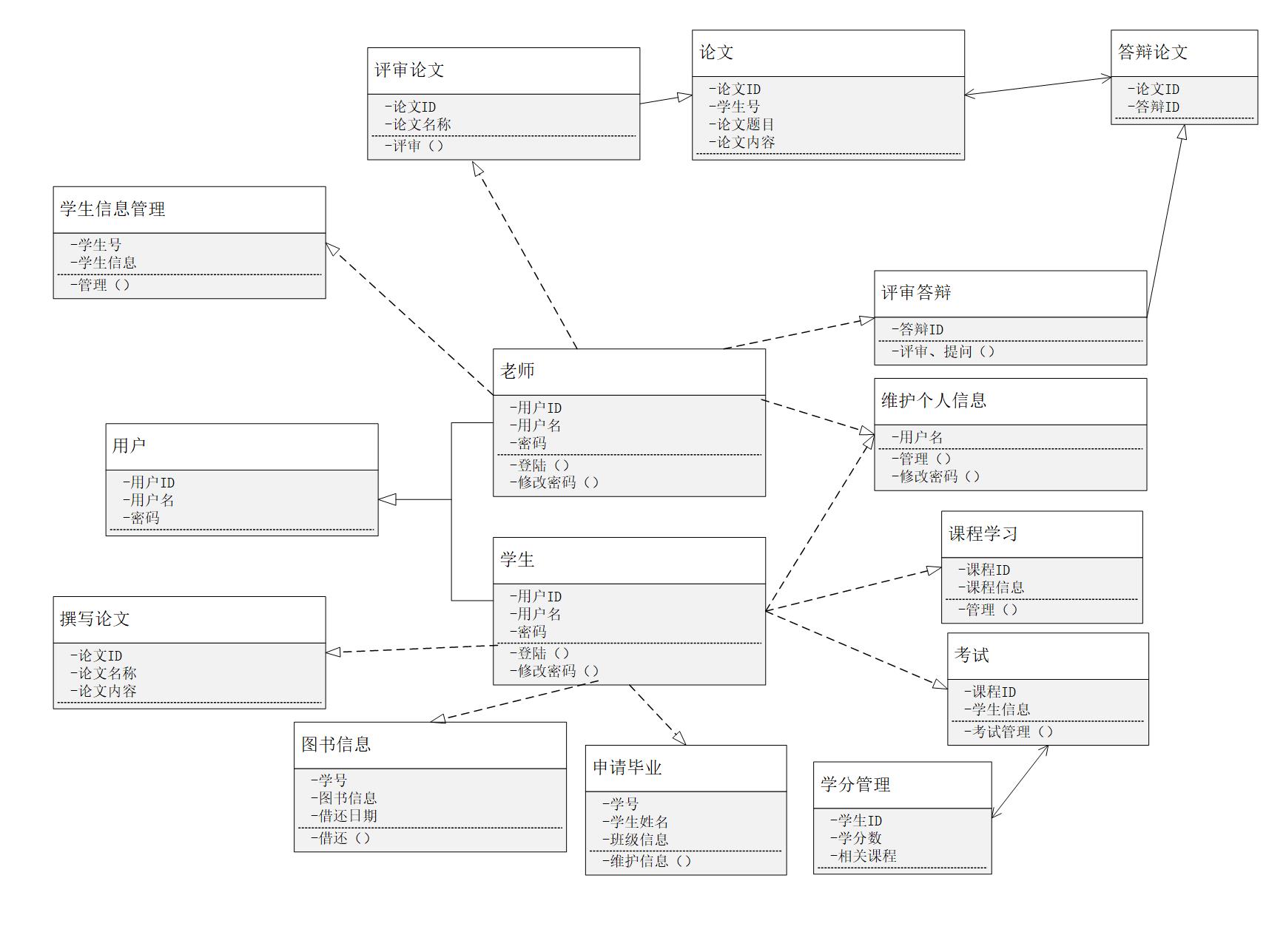
##### 13.开闭原则【感谢SkY 完成】

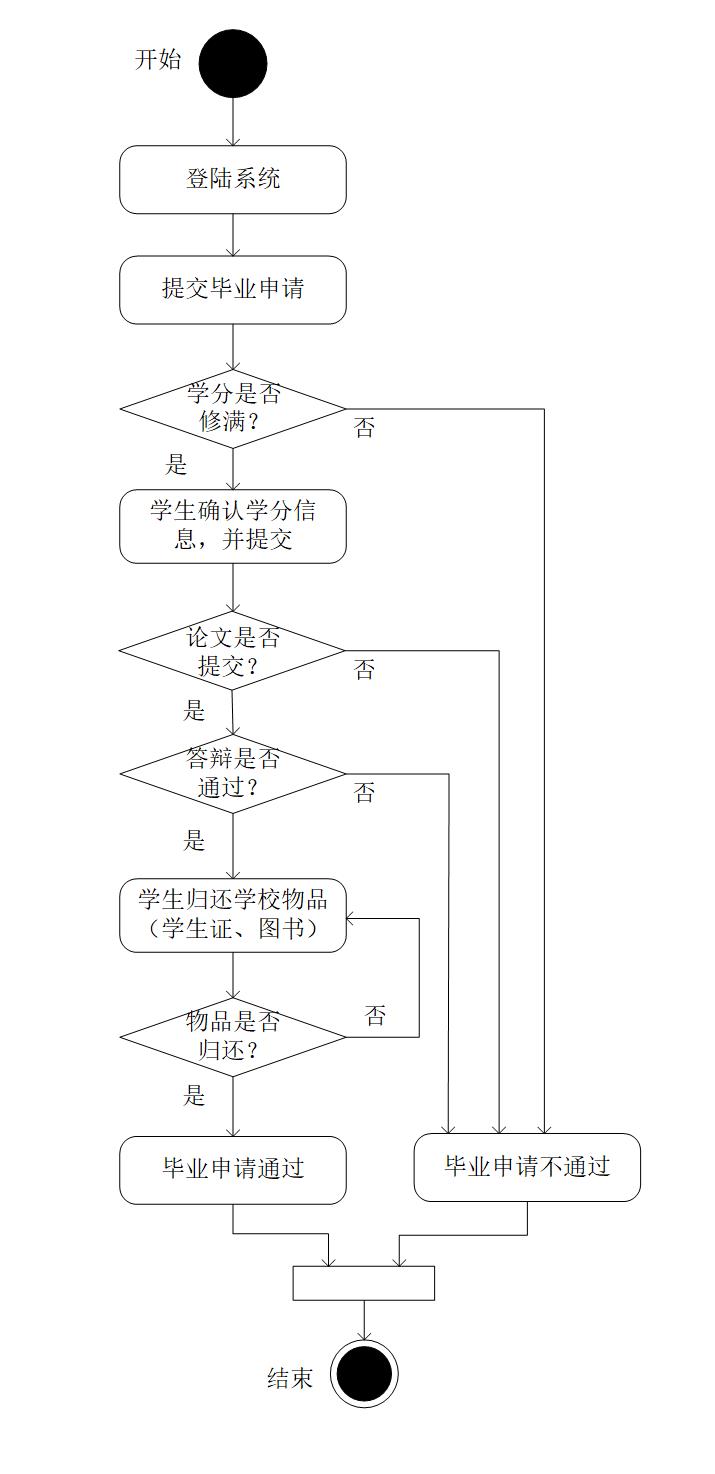
**开闭原则**是面向对象设计中“可复用设计”的基石，是面向对象设计中最重要的原则之一。软件实体对扩展是开放的，但对修改是关闭的，即在不修改一个软件实体的基础上去扩建其功能。实现开闭原则的关键就在于“抽象”，它的好处是可复用性、可维护性。

##### 14.敏捷宣言是什么【感谢idea 】

答:1).个体与交互胜过过程与工具;2).可以工作的软件胜过面面俱到的文档;3).客户协作胜过合同谈判 ;4).响应变化胜过遵循计划;

##### 15.一个场景（学生毕业申请系统），画出UML图、画出流程图0、画出流程图1【感谢晓超 】





##### 16.结合传感器说明简述软件测试的作用。【感谢侯隽感谢Alex 】

##### 17.是不是用例越多越好？为什么说明原因。【感谢旗旗是个小太阳】

答案：不是。

优点：

（1）按照详细测试用例的步骤执行的过程，不仅可以帮助他们了解测试对象的功能与业务知识，也可以帮助他们了解测试设计技术与方法。

（2）更好的一致性。由于设计的测试用例提供了详细了步骤，每个测试人员按照这个步骤可以得到一直的测试结果，因此保证测试一致性。

（3）有助于测试用例的自动化。因为详细的测试用例提供了详细的步骤和期望的结果，因此将它们转化为自动化测试用例会相对比较简单。

缺点：

（1）设计成本高：测试人员需要花费大量的时间投入到测试用例的编写上面。同时测试用例文档的页数越多，被完整阅读的可能性就越少。

（2）效果差：好的测试用例设计是从无穷的测试中选择合理测试输入、测试组合、测试数据等，以相对有限的测试用例数目尽量达到理想的覆盖率。而详细的测试用例设计很难完全定义这些组合和场景，实践中需要测试设计不断迭代和更新。

（3）维护成本高：测试用例的输入参考，例如：需求文档是经常变更的，这就会导致测试用例越详细，其维护的工作量更大。

##### 18.白盒测试和黑盒测试在用例设计上的区别。【感谢z 】

白盒测试侧重于程序结构测试，以程序的内部逻辑为基础设计测试用例。白盒测试一般使用逻辑覆盖法和基本路径法进行设计。

白盒测试优点：

1. 能仔细考虑软件的实现
2. 能检测代码的每条分支和路径
3. 能测出隐藏在代码中的错误

白盒测试缺点：

1. 昂贵
2. 不验证需求规格的正确性
3. 无法检测数据敏感性错误

黑盒测试又称功能测试，把程序看做一个无法打开的黑盒子，在完全不考虑程序内部结构的情况下，按照需求规格说明书进行测试查看每个功能是否能正常使用。黑盒测试常用方法有等价类划分、边界分析、错误推测。

黑盒测试优点：

1. 测试时不需要了解实现的具体细节
2. 从用户的角度进行测试，容易理解和接受
3. 有助于暴露需求规格不一致或有歧义的问题

黑盒测试缺点：

1. 不能覆盖系统所有程序路径
2. 代码得不到测试
3. 自动化复用效率低

### 2018

#### 数据结构

##### 栈用单链表和数组哪个更好，说理由。【感谢小春】

答：（补充向量）

向量：向量支持随机读取和更改 查找一般 末端删除快 随机删除慢，它是 “class”. 可作为顺序容器，类似数组，但大小可动态变化。与数组比较，它浪费存储空间，用起来麻烦

链表：只有头结点有名字，不支持随机读取和更改 查找慢 元素较多时删除时间大约是查找时间，适用于插入或删除频繁，存储空间需求不定的情况，存储空间不连续，动态分配内存；

数组：，数组是“变量，大小不可改变，数组无法动态定做，或者要做额外的善后工作，数组无法整体复制，向量可以，数组无法扩容。存储效果高，存取数度快；

##### 给了个LNode的类，里面是链表的定义，来实现栈的pop函数功能【感谢pretty 小鳗鱼】

答：

typedef struct LNode

{

int data;

struct LNode \*next;

}LNode;

int pop(LNode \*lst, int&x)

{

LNode \*p;

if(lst->next == NULL)

return 0;

p=list->next;

x=p->data;

lst->next=p->next;

free(p);

return 1;

}

##### 希尔排序，说明为什么会不稳定？【感谢Pray】

**希尔排序简单介绍：**希尔排序属于插入排序的一种，是直接插入排序的更高效改进版本。是基于直接插入排序以下两点性质提出的改进：

①.直接插入排序在对几乎已经排好序的数据操作时，效率高，可以达到线性的效率。

②.因为每次只能移动一位，直接插入排序一般是低效的。

其实说到底还是直接插入排序插入不够理想，根据插入排序的时间复杂度，最好的情况可以达到线性的程度，一般情况下，却不是这样，每次插入一个需要移动大量数据，我们希望在执行直接插入前能够尽量保持有序。希尔排序改进了移动步长，可以实现大跨度移动，减少移动时间。所以也叫缩小增量插入排序。

**希尔排序算法思想：**将整个无序数据分割成若干小的子序列分别进行插入排序。即先取一个 小于n的整数d1作为一个增量，把全部数据分成d1个组，所有距离为d1的倍数的记录放在同一个组里，在各组内先进行插入排序，然后，取第二个增量d2（<d1），重复上述分组和排序，直到所取得dt=1，即将所有的记录放在一个组中进行直接插入排序。

**算法实现：**

//由大到小(这里增量选择数组长度的一半，每次减半，直到gap=1，但是gap=1必须排序，所以判断条件为gap>=1)

public static void shellSort(int[] arr){

int size = arr.length;

//这层for是分组

for(int gap = size/2;gap>=1;gap/=2){

//这层for是对数组进行插入排序

for(int i= gap; i<size;i++){

int temp = arr[i];

int k = i-gap;

while(k>=0 && arr[k]<temp){

arr[k+gap] = arr[k];

k=k-gap;

}

arr[k+gap]=temp;

}

}

}

**算法分析：**希尔排序的最好与最坏时间复杂度同直接插入排序，最坏情况为O(n^2),当n在特定范围时，平均时间复杂度O(n^1.3)(ps.这里的1.3跟选取的增量大小有关)，空间复杂度为O(1)，由于在最后一次直接排序前，经过分组排序，所有可能重复的元素的相对位置会发生交换，所以希尔排序是不稳定的。

##### 哈希冲突的2种解决方法，一种在表内，一种在表外【感谢berry】

选用哈希函数计算哈希值时，可能不同的 key 会得到相同的结果，这就是冲突。

常用的主要有两种方法解决冲突：

1. 表外——链接法

链接法解决冲突的做法是： 将所有关键字为同义词的结点链接在同一个单链表中。

若选定的散列表长度为 m，则可将散列表定义为一个由 m 个头指针组成的指针数组 T[0..m-1] 。

凡是散列地址为 i 的结点，均插入到以 T[i] 为头指针的单链表中。 T 中各分量的初值均应为空指针。

装填因子定义为：α= 填入表中的元素个数 / 散列表的长度。在链接法中，装填因子 α 可以大于 1，但一般均取 α ≤ 1。

在这种方法中，Hash表每个单元中存放的不再是记录本身，而是相同同义词单链表的表头指针。



1. 表内——开放定址法

当冲突发生时，使用某种探查(亦称探测)技术在散列表中寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到。

按照形成探查序列的方法不同，可将开放定址法区分为线性探查法、二次探查法、双重散列法等。

a.线性探查法

hi=(h(key)+i) ％ m ，0 ≤ i ≤ m-1

基本思想是： 探查时从地址 d 开始，首先探查 T[d]，然后依次探查 T[d+1]，…，直到 T[m-1]，此后又循环到 T[0]，T[1]，…，直到探查到 有空余地址 或者到 T[d-1]为止。

b.二次探查法

hi=(h(key)+i\*i) ％ m，0 ≤ i ≤ m-1

基本思想是： 探查时从地址 d 开始，首先探查 T[d]，然后依次探查 T[d+1^2]，T[d+2^2]，T[d+3^2],…，等，直到探查到 有空余地址 或者到 T[d-1]为止。

缺点是无法探查到整个散列空间。

c.双重散列法

hi=(h(key)+i\*h1(key)) ％ m，0 ≤ i ≤ m-1

基本思想是： 探查时从地址 d 开始，首先探查 T[d]，然后依次探查 T[d+h1(d)], T[d + 2\*h1(d)]，…，等。

该方法使用了两个散列函数 h(key) 和 h1(key)，故也称为双散列函数探查法。

定义 h1(key) 的方法较多，但无论采用什么方法定义，都必须使 h1(key) 的值和 m 互素，才能使发生冲突的同义词地址均匀地分布在整个表中，否则可能造成同义词地址的循环计算。

该方法是开放定址法中最好的方法之一。

##### 5.哈夫曼树，哈夫曼编码的算法，压缩率【感谢North】

哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的 路径长度（若根结点为0层，叶结点到根结点的路径长度为叶结点的层数）。树的带权路径长度记为WPL= (W1\*L1+W2\*L2+W3\*L3+...+Wn\*Ln)。

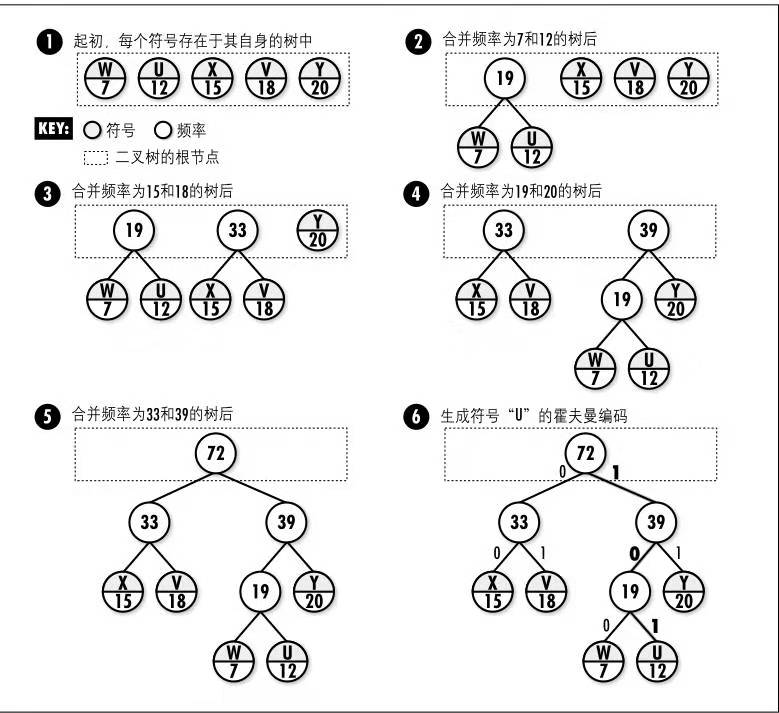
**哈夫曼编码是哈夫曼树的一个应用。哈夫曼编码**构造哈夫曼树：

构建的过程往往从叶子结点向上进行。首先，将每个符号和频率放到它自身的树中（步骤1）。然后，将两个频率最小的根结点的树合并，并将其频率之和存放到树的新根结点中（步骤2）。这个过程反复持续下去，直到最后只剩下一棵树（这棵树就是霍夫曼树，步骤5）。霍夫曼树的根结点包含数据中符号的总个数，它的叶子结点包含原始的符号和符号的频率。

建立一棵霍夫曼数是数据压缩和解压缩的一部分。用霍夫曼树压缩数据，给定一个具体的符号，从树的根开始，然后沿着符号的叶向叶子结点追踪。在向下追踪的过程中，当向左分支移动时，向当前编码末尾追加0；当向右移动时，向当前编码末尾追加1。

解压缩用霍夫曼树编码的数据，要一位一位地读取压缩数据。从树的根开始，当在数据中遇到0时，向树的左分支移动；当遇到1时，向右分支移动。一旦到达一个叶子结点，就找到了符号。接着从头开始，重复上述过程，直到所有的压缩数据都确定。

**压缩率：**通常用1减去压缩数据大小除以原始数据大小的值。这个值称为数据的压缩率。



#### 计算机系统

##### am 化方法【感谢姜 】

阿姆达尔定律是计算机系统设计的重要定量原理之一，于1967年由IBM360系列机的主要设计者阿姆达尔首先提出。该定律是指：系统中对某一部件采用更快执行方式所能获得的系统性能改进程度，取决于这种执行方式被使用的频率，或所占总执行时间的比例。阿姆达尔定律实际上定义了采取增强（加速）某部分功能处理的措施后可获得的性能改进或执行时间的加速比。简单来说是通过更快的处理器来获得加速是由慢的系统组件所限制。

阿姆达尔定理主要观点：要想显著加速整个系统，必须提升全系统大部分的速度。

决定系统性能的关键因素有那么几点：内存资源、CPU资源、磁盘I/O，那么系统的总体性能是取决于当前最慢的磁盘I/O速度，而不是内存和CPU。这种情况下，只有提高磁盘I/O速度才能对系统的整体性能进行优化。

常用的优化方法没找到

##### memery + disk和memery+cache两种存储系统的设计差异【感谢Jo 】

基于所查资料，题目应为"disk cache"与"memery cache"的设计差异。

MemoryCache，是将资源缓存到内存中，等待下次访问时不需要重新下载资源，而直接从内存中获取。

DiskCache，是将资源缓存到磁盘中，等待下次访问时不需要重新下载资源，而直接从磁盘中获取。

两者最大的区别在于，当退出进程时，内存中的数据会被清空，而磁盘的数据不会，所以，当下次再进入该进程时，该进程仍可以从DiskCache中获得数据，而MemoryCache则不行。

Webkit资源分成两类，一类是主资源，比如HTML页面，或者下载项，一类是派生资源，比如HTML页面中内嵌的图片或者脚本链接。DiskCache与MemoryCache相似之处是只能存储一些派生类资源文件。

MemoryCache对应的类为CachedResource，用于保存原始数据（比如CSS，JS等），以及解码过的图片数据。

DiskCache存储形式为一个index.dat文件，记录存储数据的url，然后再分别存储该url的response信息和content内容。

##### 3.流水线设计的优点，为什么能提高性能，可能会影响的因素，怎么解决处理这些影响因素 感谢【高晨晨chen】

流水线技术：将指令执行过程划分为不同阶段，占用不同的资源，使多条指令同时执行。

优点：程序的执行时间缩短，各功能部件的利用率明显提高。

缺点：需要付出硬件上的开销。控制过程也比顺序执行复杂。

为什么提高性能：多条指令可以并行处理，提高了吞吐量，提高资源利用率。

影响流水线的因素：

1、结构相关（结构冲突）

后一指令用到前一指令的相同结构，如存储器。

2、数据相关（数据冲突）

后一指令用到前一指令的相关数据。

3、控制相关（控制冲突）

当流水线遇到转移指令或其他改变PC值的指令而造成断流时，会引发控制相关。

比如条件判断语句或跳转语句。

解决办法：

1、结构相关：

（1）后一相关指令暂停一个周期

（2）资源重复配置：如将存储器拆分为数据存储器和指令存储器。

2、数据相关：

（1）把遇到数据相关的指令及其后续指令都暂停一至几个时钟周期，直到数据相关问题消失后再继续执行，可分为硬件阻塞（stall）和软件插入“NOP”两种方式。

（2）数据旁路技术。通过将ALU等部件的输出直接作为下一指令的输入，提前获得数据。

（3）编译优化：通过编译器调整指令顺序来解决数据冲突。把后面用不到数据相关的指令，插入前面，等到数据相关的指令完成后，继续执行下一个指令。

3、控制相关：

解决办法：

（1）分支预测，尽早判别转移是否发生，尽早预测转移分支。

（2）预取两个方向指令。

（3）加快和提前形成条件码。

（4）提高转移方向的猜准率。（这个好像用的最多）

##### 缺失率 【感谢North】

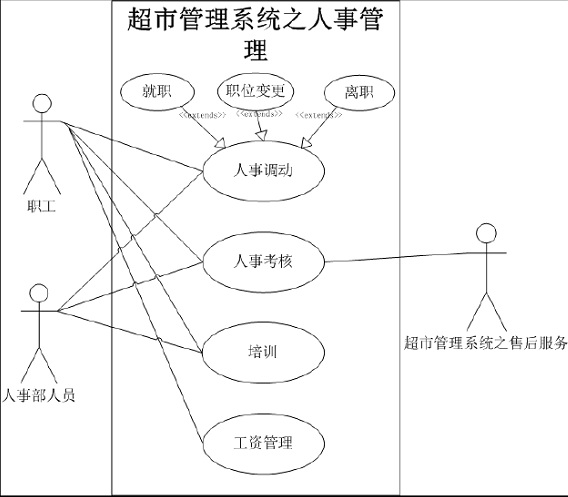
Cache和主存都被分为若干大小相等的块，Cache块还可以称作Cache行。每块由若干字节组成，块的长度称为块长。由于Cache的容量远远小于主存的容量，所以Cache中的块数要远少于主存中的块数。仅仅保存主存中**最活跃的**若干块的副本。

当CPU发出读请求时，如果欲访问地址在Cache中命中，就将此地址转换成Cache地址，直接对Cache进行读操作。与主存无关。如果Cache不命中，则仍需访问主存，并把此字所在的块一并从主存调入Cache内。 若此时Cache已满，则需要根据某种替换算法，用新的主存块替换掉原来Cache中原来的块的信息。有些时候，也可以同时访问Cache和主存，如果命中Cache，终止主存访问。访问主存慢，所以命中后就停止访问主存。CPU与Cache之间数据交换以字为单位，Cache与主存之间以Cache块为单位。

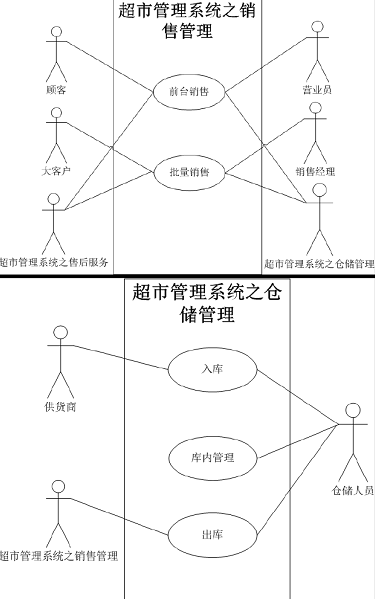
缺失率＝读和写不命中数量／读和写的总数量

#### 软件工程

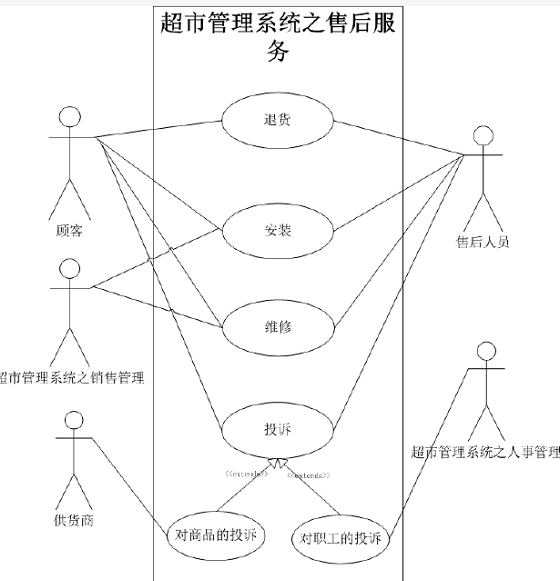
##### 超市系统的用例图【感谢小春】



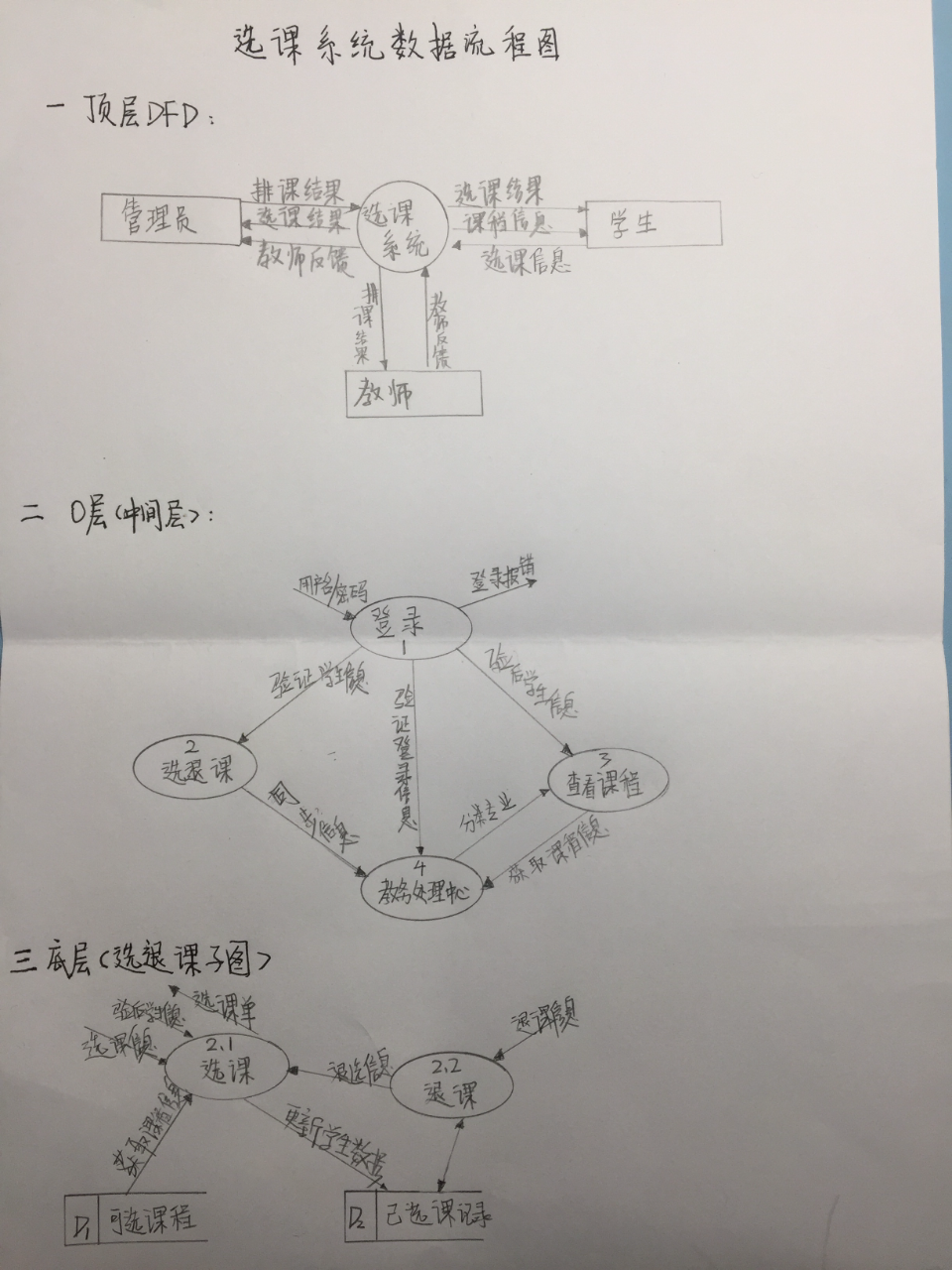
-



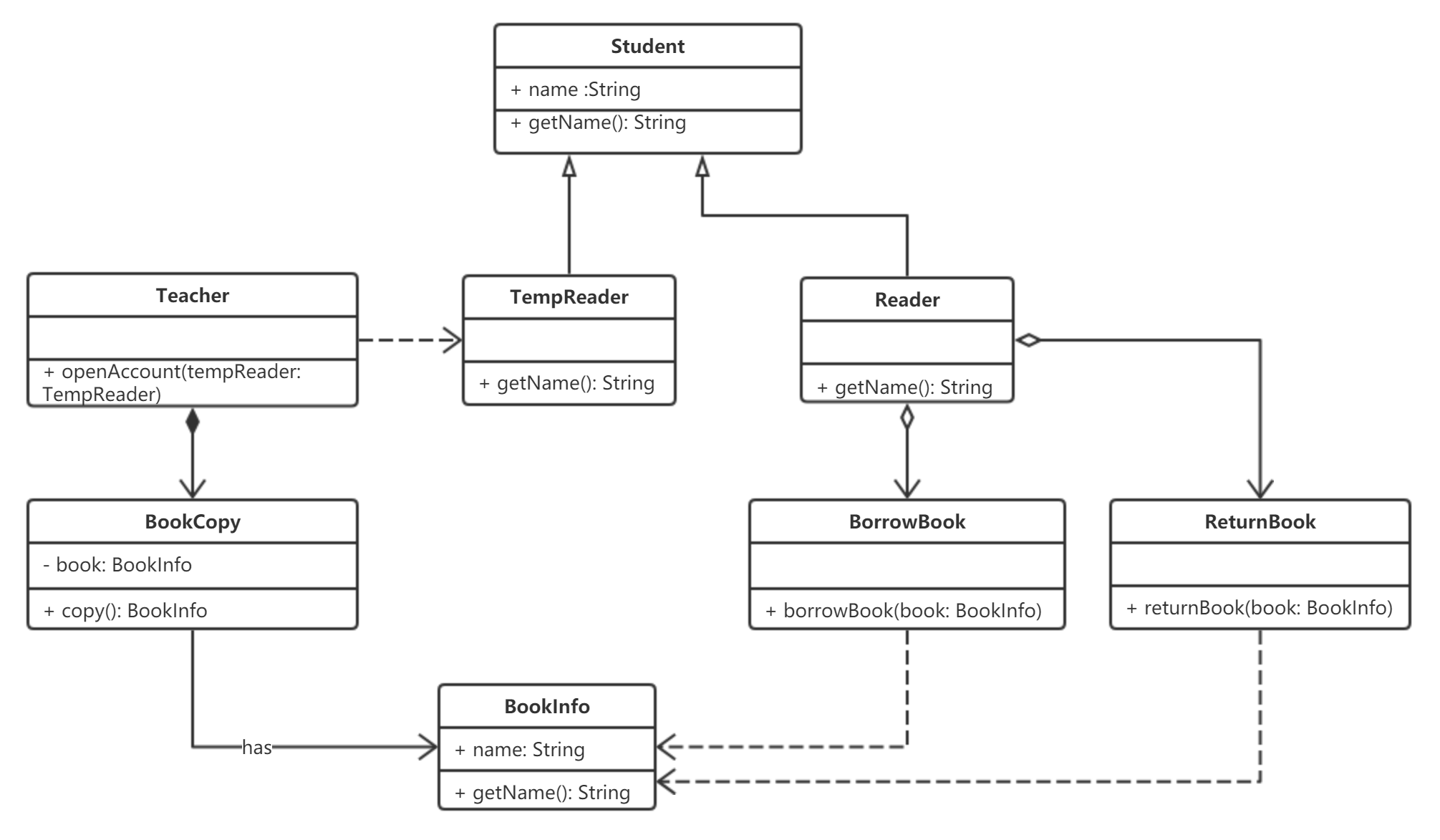
-



##### 选课系统的数据流程图【感谢小春】



##### 3.图书馆的类图，关于读者，老师，学生，临时读者，图书信息，图书拷贝，借书还书记录。感谢关



##### 4.面向对象设计选择，依赖倒置原则，接口隔离原则。【感谢绾晴】

面相对象原则有开闭原则，依赖倒置原则，单一责任原则，里氏代换原则，复合复用原则，迪米特法则，接口隔离原则，我们在面相对象设计中要遵循这些原则，才能增强程序的可扩展性，迭代性，可维护性

依赖倒置原则：高层模块不应该依赖底层模块，两个都应该依赖与抽象；抽象不应该依赖于细节，细节应该依赖于抽象

例如：我们平常使用的工厂模式

把数据转换成xml格式，我们应该定义个数据转换抽象类，再定义个继承抽象类的实现类，实现转换xml格式方法，当我们要新增一个转换成xls文件时，我们只需要继承抽象类实现一个换xls格式的类即可。

接口隔离原则：

接口隔离原则是指使用多个专门接口，而不使用单一总接口。每一个接口应该承担一种相对独立的角色，不多不少，不干不该干的事，该干的事都要干。使用接口隔离原则拆分接口时，首先必须满足单一职责原则，将一组相关的操作定义在一个接口中，且在满足高内聚的前提下，接口中的方法越少越好

例如做支付宝和微信支付接口，应该定义两个类

## 分层数据流模型

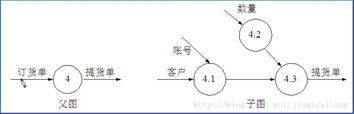
### 基本图形符号



### 设计原则

**(1) 父图与子图的平衡原则**

子图的输入输出数据流同父图相应加工的输入输出数据流必须一致, 此即父图与子图的平衡.



**(2) 数据守恒原则**

对任何一个加工来说, 其所有输出数据流中的数据必须能从该加工的输入数据流中直接获得, 或者说是通过该加工能产生的数据.

1.外部实体与外部实体之间不存在数据流

2.外部实体与数据存储之间不存在数据流

3.数据存储与数据存储之间不存在数据流

(3) 守恒加工原则

对同一个加工来说, 输入与输出的名字必须不相同, 即使它们的组成成分相同.

1.对于每个加工, 必须既有输入数据流, 又有输出数据流.

2.数据流与加工有关, 且必须经过加工.

对于加工, 输入是A, 输出还是A, 也违反了数据守恒原则, 输入与输出一样, 加工没有作用. 人不可能吃的什么, 出来的是同一样的. 

对于加工, 只有输入, 没有输出, 违反了数据守恒原则. 人不可能只吃饭, 不大小便. 

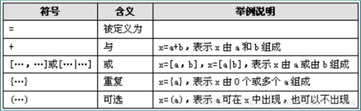
对于加工, 只有输出, 没有输入, 违反了数据守恒原则. 人不可能一直大小便, 但不吃饭. 

数据流经过加工之后, 数据流的关系, 如图：

[]

### 数据字典

数据流图描述了系统的分解。但没有对图中各成分进行说明。数据字典是对数据流图中出现的所有被命名的图形元素在数据字典中作为一个词条加以定义，使 切的解释。



在数据字典中有4种类型的条目：

1、数据项条目：通常为数据项的值类型，允许的取值范围等

2、数据流条目：给出某个数据流的定义，列出该数据流的各组成数据项。

3、文件条目：对文件的定义，列出期组成的数据项

4、加工条目：对每个不能再分解的加工做说明，包括加工的激发条件，加工的逻辑，优先级等等。

根据层级数据流图分为顶层数据流图、中层数据流图和底层数据流图. 除顶层数据流图外, 其他数据流图从零开始编号. 顶层数据流图只含有一个加工表示整个系统; 输出数据流和输入数据流为系统的输入数据和输出数据, 表明系统的范围, 以及与外部环境的数据交换关系. 中层数据流图是对父层数据流图中某个加工进行细化, 而它的某个加工也可以再次细化, 形成子图; 中间层次的多少, 一般视系统的复杂程度而定. 流程图0、画出流程图1又叫 顶层数据流图、一层数据流图;

它是将提供给用户的业务流程图(“物理模型”)进行功能建模, 转化成开发人员能够理解的一系列“逻辑模型”图, 即以图形化的方法描绘数据在系统中的流动和处理的过程, 这些图都应该用规范的DFD描述.

对于一个软件系统, 其数据流图可能有许多层, 每一层又有许多张图. 为了区分不同的加工和不同的DFD子图,应该对每张图进行编号, 以便于管理.

● 顶层图只有一张, 图中的加工也只有一个, 所以不必为其编号.

● 0层图只有一张, 图中的加工号分别是0.1、0.2、…, 或者1, 2 .

● 子图就是父图中被分解的加工号.

● 子图中的加工号是由图号、圆点和序号组成, 如: 1.12, 1.3 等等.

6

# 2019

## 数据结构与算法(60')

1. 为何单向链表只能在尾部入，头部出？如果要使得尾部和头部都能以O(1)进行插入和删除操作，需要做什么改造？

2. Dijkstra算法填空

3. 写一个高效的构建二叉堆的算法，分析其复杂度，并给出结果

4. 写一个算法，统计二叉树中有2个非空子树的节点，并分析复杂度

5. 写一个算法，从输入的数字序列中，提取最小的k个元素，并达到O(N)的复杂度，如果达不到O(N)的复杂度，会酌情给分

## 软件工程(60')

6. 分4小问，4×4分=16分

* 简述CMMI
* 简述需求获取的途径
* 简述抽象和逐步求精的关系
* 怎样判断用例的好坏？

7. 23分

* 给定一个需求，画出状态机图：台灯有灭-弱光-强光状态，有接通电源和使用内置充电电池两种能量来源，只有一个按钮，按按钮会使得在 灭-弱光-强光 三个状态中切换，如果没有接通电源也没有足够的电量，按按钮没反应；如果是亮的状态，却没有接通电源，当内置电池电量用完的时候会灭掉
* 给定一个考试管理系统，给出顶层和0层图，0层分解为考试报名系统和成绩管理系统

8. (15') 给定一个CBubbleSorter类中的Sort方法，问

* 什么是内聚性？这个方法符合内聚性吗？为什么？怎样提高这个方法的内聚性？
* 如果这个类是继承自CSorter类，画一个类图来表示这两个的关系
* 根据上面的类图，怎样设计才能方便地切换为其他的排序算法？你的改动体现了什么设计规则？

9. 2×3=6分

* 白盒测试的覆盖标准，至少列出三个
* 有人说：“测试就是检查程序的正确性”，对吗？解释下

## ICS(30')

10. 三问，3×10=30分

* 流水线的分段划分方法非常影响流水线的性能，简述一下怎么个影响法；阐述下基于分段的流水线处理器中，什么是加速比；简述下怎么分段才能使加速比趋向于理想中的加速比？
* 流水线的停顿非常影响性能，一般有什么原因会出现停顿？一般用什么策略解决停顿问题？
* 现代CPU中，分层存储结构如何影响流水线的性能？存储层级结构是基于什么原理设计的？为降低存储对CPU的影响，应该采取什么措施进行优化？