给定一个已排序的数组，找到一个元素第一次出现的位置，{1,2,2,3,3,3,4}，输出2

方法：二分查找，对边界情况进行考虑。

HashMap：

首先通过散列函数将数据散列到对应位置，散列冲突的解决方法是，链地址法

当链接数达到8时，存储到红黑树中

TCP介绍

传输层协议，面向连接的，可靠数据传输、流量控制、拥塞控制、三次握手、四次挥手

如何保证可靠数据传输：通过序列号和ACK，以及重传技术，保证数据的可靠性

流量控制：端到端的控制，防止发送端数据发送过快，超过接收端的数据处理能力

拥塞控制：防止过多的数据发送到网络中，使得网络中的链路和路由器不至于过载。拥塞控制是系统的全局的过程，涉及到所有的主机。

拥塞控制原理：控制发送窗口和拥塞窗口，网络没有问题则增大拥塞窗口，否则减小拥塞窗口

慢开始：刚建立链接时，窗口增加缓慢

乘法减小加法增大：在开始和拥塞避免阶段，超时就是的窗口减半，加法增大则缓慢增大窗口

快重传：收到失序报文后立即发出ACK

UDP介绍

无连接的，不可靠传输，速度快，占用资源少

B树和B+树的区别

B树的每个节点都有关键字，B+树的所有关键字都保存在叶子节点，B+树的叶子节点可以构成有序链表，可以按照关键字排序的次序遍历全部记录

B+树的缺点是需要更多的节点，

B+树的内部节点没有存储指向关键字具体信息的指针，因此其内部节点相对B树更小，可以将更多的关键字存放在内存中

而B+树的优点是，查询稳定，且内存中可以存放更多的key，数据存放更加紧密，具有更好的空间局部性，在访问叶子节点关联数据的时候具有更好的缓存命中率

B+树的叶子节点相链，只需要依次线性遍历可以对所有数据进行扫描，而B树需要对每一层遍历

B+树的缺点是，会产生大量的随机IO，因为随着新数据的插入，叶子节点会慢慢分裂，逻辑上连续的节点在物理上会变得不连续，在范围查询时，会产生大量的随机IO

数据库有哪些级别的隔离？

三种问题：脏读、不可重复读、幻读

四种级别：串行化（可避免以上三个问题）、可重复读（避免前两个）、读已提交（第一个）、读未提交（最低级别，无法保证）

聚簇索引：确定表中数据的物理顺序，一个表只能有一个聚簇索引

联合索引：由多个字段组成的索引，只有当联合索引的第一个字段在where条件中出现时才会使用联合索引，否则是顺序扫描

数据库事务的四大特性：原子性（事务要么全部成功，要么全部失败）、一致性（数据库保持一致的状态）、隔离性（多个并发事务不能相互干扰）、持久性（事务提交后，数据改变是永久的）

ARP协议：地址解析协议，根据IP地址获取物理地址的数据链路层协议。主机发送信息将包含目标IP地址的ARP请求广播到同一个局域网内的所有主机，目的MAC全F，源MAC是当前主机的MAC。当另一台主机接收到这个ARP请求包后，通过解析，发现其目的IP地址和自己的IP地址相同，这时该主机将在自己的ARP缓存表中建立一个该ARP请求的IP地址和源MAC的映射。然后该主机将ARP回应包，直接发送给对应的出口，

静态链接和动态链接：共同点，都是共享代码的方式，如果采用静态链接库，则lib被直接包含在exe文件中，动态链接，则可以动态引用和卸载这个与exe独立的dll文件。

静态链接库不能再包含其它链接库，而动态链接则可以

静态库的好处是快，但造成了空间浪费，且更新静态库后，需要重新编译代码，而动态库则更新库后不需要重新编译

MMU是内存管理单元，是CPU中用来管理虚拟存储器的控制线路，负责虚拟地址映射为物理地址，以及为硬件机制的内存访问授权

大多数使用虚拟存储器的系统都使用一种称为分页的技术，虚拟地址被划分成页，物理地址页被进行划分。

HashMap、HashTable、ConcurrentHashMap

HashMap

LRU，Least Recently Used，最近未被访问，则在将来它被访问的可能性也很小。

LRU的实现，用一个数组来存储数据，给每个数据项标记一个访问时间戳，当有新数据项插入时，先将数组中存在的数据时间戳自增，并将新数据项的时间戳置为0并插入到数组中。每次被访问的数据项的时间戳置为0，将时间戳最大的数据项淘汰。

链表实现，每次插入新数据时，将新数据插入到链表头部，每次命中，则将数据移到链表头部，那么当链表满时，丢弃链表尾部数据

方法3，利用链表和hashmap，如果命中，则将节点移动到链表头部，如果不存在，则新建一个节点，放到头部。

堆和二叉树有什么区别

堆是一种特殊的完全二叉树

Java中如何实现延时执行

方法一：创建一个线程，里面放入Thread.sleep()方法，只能运行一次

方法二：使用Java API提供的Timer类，调用timer.schedule(task,5000,2000) 5秒后执行，执行时间间隔为2s，

Timer timer = new Timer();

TimerTask task = new TimerTask(){

Public void run(){

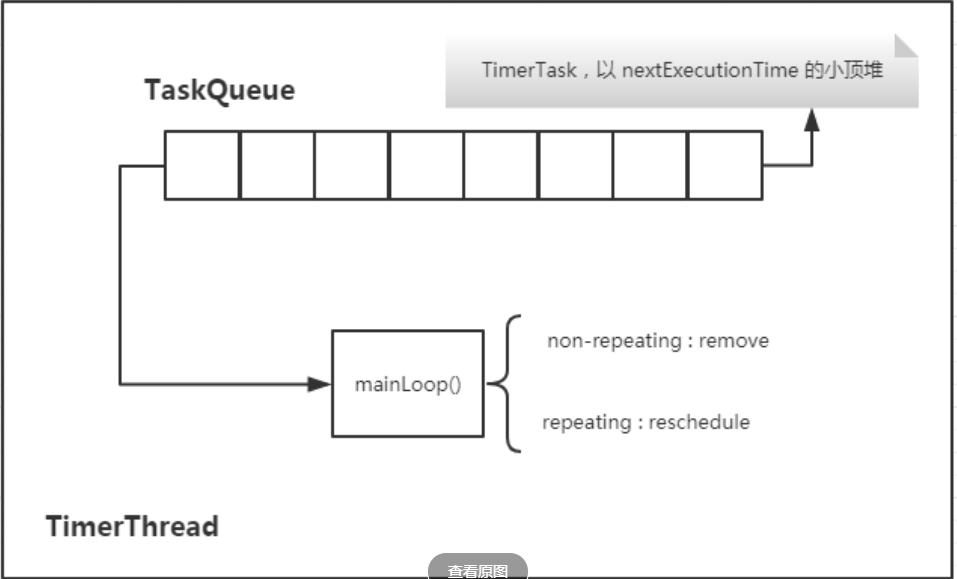
System.out.println(“计时任务”);

}

};

timer.schedule(task,5000,2000);

Java Timer的实现原理



TimerThread:由Timer启动，里面有个无限循环的mainLoop方法，用来取出TaskQueue中最近需要执行的任务，判断是否需要执行。

TaskQueue是由TimerTask组成的小顶堆，其排序是根据TimerTask的nextExecutionTime

Schedule在某次执行任务的开始时间延后了，那么此后每次任务都会延迟

而scheduleAtFixedRate则并不会这样

Epoll和poll、select

操作系统在处理IO的时候，有两个阶段：等待数据传到IO设备、IO设备将数据复制到user space，简化理解：等待数据传到内核空间、内核空间将数据复制到user space

三种模型

阻塞IO模型，在数据达到缓冲区前，进程被阻塞。即进程在调用recvfrom开始到它返回的整段时间都是阻塞的。在应用层数据到达内核、内核复制到用户空间的两个过程都是阻塞的，一起阻塞。

非阻塞IO模型，在应用层数据到内核过程中，recvfrom会轮询查询，如果内核数据还没有准备号，就会返回错误。不断的轮询查询，直到内核中数据准备好了。

多路复用模型IO，为了处理多个IO请求，用一个进程来处理多个fd请求。

Select工作流程

Select中使用的fd\_set结构

当某个程序调用select时，这个程序就被阻塞了，这时候，内核会轮询所有select负责的fd，当找到一个client中的数据准备好了，select就会返回，这个时候程序就会进入系统调用，将数据从内核复制到进程缓存区。虽然服务器进程会被select阻塞，但是select会利用内核不断轮询监听其它客户端的IO操作是否完成

Poll工作原理

和select非常相似，但是poll使用的是pollfd结构，poll是链式的，没有最大连接数的限制，poll有一个特点是水平出发，也就是通知程序fd就绪后，这次没有处理，下次poll的时候会再次通知同一个fd已经就绪

Select的缺点

fd\_size为32位整数大小，最多能处理1024个fd，每次要判断哪些event发生，采用轮询机制，成本很高。

每次调用select需要从user space把FD\_SET复制到内核。

需要轮询每个fd，线性时间

Epoll技术

Epoll没有fd数量限制，每个epoll监听一个fd，不需要将fd\_set从用户空间复制到内核

Epoll是被动触发方式，给fd注册了对应的事件，当数据准备好后，就会把就绪的fd加入到就绪队列，epoll\_wait在就绪队列里查看有没有就绪的fd，有就唤醒队列里的等待者，然后调用回调函数。

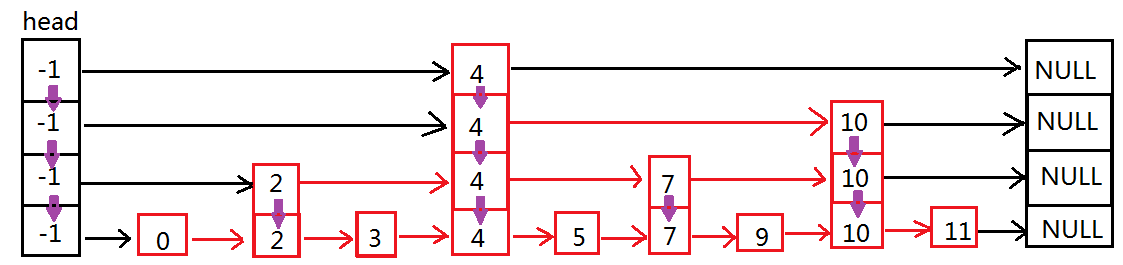
Epoll在执行epoll\_create会在内核的高速cache中建立一棵红黑树，以及就绪链表(该链表存储已经就绪的文件描述符)。

调用epoll\_ctl函数添加文件描述符会在红黑树上增加响应的节点。

在执行epoll\_ctl的add操作时，不仅将文件描述符放到红黑树中，而且要注册回调函数。

epoll\_wait只需要观察就绪链表中有无数据即可，将就绪链表中的数据返回给数组，并返回就绪数量。返回文件描述符是通过mmap让内核和用户空间共享同一块内存。

跳表的优势，与红黑树对比



跳表性质：1由多层组成2每层是一个有序链表3最底层链表包含所有元素4如果一个元素在第i层链表中出现，则它在第i-1层中也会出现5上层节点可以跳转到下层。

跳表在有序链表上增加了多级索引，通过索引来实现快速查找

查找过程：首先在最高级索引上查找最后一个小于当前元素的位置，然后再跳到次高级索引继续查找，直到跳到最底层为止

理想情况下，查找的时间复杂度为O(log n)

如何决定一个节点有多少级索引，用随机数产生一个0和1，如果是1的化则level++，直到产生0为止。当数据足够大的时候，level的取值会趋向正态分布

跳表的效率与红黑树和AVL树不相上下，胜在实现起来简单。调表更新的时候需要改动的地方很少，而红黑树和AVL树需要改动的地方很多。

在多线程的情况下，红黑树和AVL树在维持平衡时，需要锁住很多资源，越是在靠近根节点的地方越容易产生竞争。但是跳表的操作更加局部性一点，需要锁住的资源很少。

Bloom过滤器

判断一个元素key是否在某个集合中

首先需要k个hash函数，每个函数可以把key散列成1-n的整数

初始化时，需要一个长度为n的比特数组，每个比特位初始化为0

将key加入到集合时，用k个哈希函数计算出k个散列值，并将数组中对应位置1

判断key是否在集合中时，用k个哈希函数计算出k个散列值，并查询数组中对应的比特位，如果比特位都是1，则认为在集合中。

优点：不需要存储key，节省空间，且效率很高

缺点：算法判断key在集合中时，有一定概率key其实不在集合中，无法删除

应用：在需要后端查询时，先判断key是否在后端存在，在Bigtable中使用了bloom过滤器来查找不存在的行或者列

如何防止恶意刷票

1、IP限制2、Cookies验证3、MAC地址限制4、验证码注册5、行为记录

如何避免爬虫对数据库造成过大的负载

封锁爬虫IP地址、通过User-Agent信息来排除访问量最大的User-Agent、网站流量统计系统和日志分析识别爬虫

输入url到页面展示过程中，发生了什么？

输入网址，在本地网页缓存中判断是否存在，先在内存中、然后磁盘中

域名解析，DNS解析，从域名到DNS的过程，该过程由DNS服务器完成。首先判断在本机DNS缓存中是否存在该域名，如果存在则获取对应的IP地址，否则，访问本地DNS服务器，然后再访问权威域名服务器，最后到根域名服务器。

TCP连接，三次握手

服务器收到HTTP请求，返回一个HTTP响应，包括响应头和响应体

客户端收到响应后，解析css、js

堆和栈的区别

分配方式的区别

栈（操作系统）：由操作系统自动分配释放，存放函数的参数值，局部变量的值，其操作方式类似于数据结构的栈

堆（操作系统）：一般由程序员分配释放，若程序员未释放，则在程序结束时，会被操作系统回收哦，分配方式类似于链表。

缓存方式的区别

栈一般使用的是一级缓存，通常是被调用时处于存储空间中，调用完立即释放

堆存放在二级缓存中，生命周期由虚拟机的垃圾回收算法决定

效率的区别

栈由系统自动分配、速度快，程序员无法控制

堆由new分配的内存，速度较慢，而且容易产生碎片，但是使用方便

存储内容的区别

栈中，参数是从右往左入栈，然后是函数中的局部变量入栈，静态变量不入栈

当调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向函数的返回地址，也就是主函数中下一条指令的地址。

堆：一般在堆的头部用一个字节存放堆的大小。

REST是一种软件架构风格，针对网络应用的设计和开发方式，可以降低开发复杂性，提高系统可伸缩性

MVC：model、view、control的缩写，软件设计典范，业务逻辑、数据、界面分离的方法组织代码

非递归前序遍历二叉树：先访问根节点，再分别访问左右子树。对于任意节点P，先访问节点P，并将P入栈，判断P的左孩子是否为空。若为空，则取出栈顶节点并进行出栈操作，并将栈顶的右孩子置为当前节点P，循环至1；若不为空，则将P的左孩子置为当前节点。直到P为空且栈为空，遍历结束。

最大连续子数组的解法

一：暴力解法，任意两个连续区间求和，找到最大的，这个的时间复杂度是n^3

二：分治解法，分成左右两个部分，递归求解左边最大连续子数组和右边最大连续子数组，然后求跨两个数组的连续子数组

合并过程：求中点右边的最大值和最大位置，中点左边的最大值和最大位置，然后将两个最大和加在一起

时间是nlogn的，空间是n的

三：在线算法

Sum不断累加，如果sum小于0，则使sum=0

因为如果sum小于0，那么后面的再加的数也只会让子数组变小。

时间为n，空间为n

最大的k个数的解法

快速排序法，用二分的快排，找到第k大的位置，时间复杂度是O(nlogk)

堆法O(nlogk)

堆解决k个数的方法是基于比较的方法中最优的

最大子方阵问题

B[i,j] = min{B[i-1,j],B[i,j-1],B[i-1,j-1]}+1 if A[i][j] = 1;

LSM tree原理是将对数据的增量修改保存在内存中，达到指定的大小限制后，就将这些修改操作批量写入到磁盘中，不过读取的时候比较麻烦，需要合并磁盘中历史数据和内存中

LSM 树的原理是把一棵大树拆分成N棵小树，它首先写入内存中，随着小树越来越大，内存中的小树会flush到磁盘中，磁盘中的树可以定期的做merge操作，合并成一棵大树，以优化读性能。

内核什么时候回收内存：一定期对进程的内存进行周期性检查，以保证内存的正常可用性、二是当内存分配时没有空闲空间满足要求时，触发内存回收。

进程间通信5种方式

1. 管道，本质文件，一种特殊的文件，但不属于任何文件系统，并且只存在于内存种，且只有父子进程能够通讯
2. 命名管道FIFO，使用文件来传输数据，速度慢
3. 消息队列：消息链接表，存放在内核中，消息队列通过ID来标识，受到系统容量的限制
4. 信号量：本质是一个计数器，用于进程间的同步和互斥
5. 共享内存：速度快，但需要保持同步
6. 套接字：可以用于不同进程间的通信

死锁的条件：互斥、请求与保持、不可剥夺、循环等待

对资源申请进行检查，并根据结果判断是否可能发生死锁

按下Ctrl+C时，产生了SIGINT信号