2019/08/22

1、数组和链表的区别?

数组:

- 在内存中，数组是一块**连续的区域**；
- 数组需要**预留空间，在使用前要先申请占内存的大小，可能会浪费内存空间**;
- **插入数据和删除数据效率低**（插入数据时，这个位置后面的数据在内存中都要向后移。删除数据时，这个数据后面的数据都要往前移动）；
- **随机读取效率很高。**因为数组是连续的，知道每一个数据的内存地址，可以直接找到给地址的数据;
- **不利于扩展，数组定义的空间不够时要重新定义数组。**

链表:

- 在**内存中可以存在任何地方，不要求连续**;
- **每一个数据都保存了下一个数据的内存地址**，通过这个地址找到下一个数据;
- **增加数据和删除数据很容易**;
- **查找数据时效率低，因为不具有随机访问性**，所以访问某个位置的数据都要从第一个数据开始访问，然后根据第一个数据保存的下一个数据的地址找到第二个数据，以此类推；
- **不指定大小，扩展方便。**链表大小不用定义，数据随意增删。

2、数据结构中的堆了解吗？堆排序呢?

- 堆是一种**特殊的树形数据结构**，其每个节点都有一个值，通常提到的堆都是指一颗**完全二叉树**，**根结点的值小于（或大于）两个子节点的值**，同时，根节点的两个子树也分别是一个堆。
- 堆排序：**利用堆的性质**进行的一种选择排序。

```
public class HeadSort {
    public static void buildHeap(int[] arr, int parent, int length) {
        for(int child = 2 * parent + 1; child < length; child = 2 * child + 1) {
            // 让child先指向子节点中最大的节点
            if(child + 1 < length && arr[child] < arr[child + 1]) {
                child++;
            }
            if(arr[parent] < arr[child]) {// 如果发现子节点更大，则进行值的交换
                swap(arr, parent, child);
                // 下面就是非常关键的一步了
                // 如果子节点更换了，那么，以子节点为根的子树会不会受到影响呢?
                // 所以，循环对子节点所在的树继续进行判断
                parent = child;
            } else {
                break;
            }
        }
    }
    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
```

```
    public static void sort(int[] arr){
        for(int parent = arr.length / 2 - 1; parent >= 0; parent--) {//构建最大堆
            buildHeap(arr, parent, arr.length);
        }
        for(int i = arr.length - 1; i > 0; i--) {//把最大的元素扔在最后面
            swap(arr, 0, i);
            buildHeap(arr, 0, i);//将arr中前i - 1个记录重新调整为最大堆
        }
    }
}
```

时间复杂度：最好、最坏、平均 -> O（NlogN）

空间复杂度：O（1）

稳定性：**不稳定**

3、快速排序?

- 快速排序是一种**交换排序**。
- 通过一趟排序将要排序的数据分割成独立的两部分：**分割点左边都是比它小的数，右边都是比它大的数。**然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以**递归**进行，以此达到整个数据变成有序序列。

```
public class QuickSort {
    public static void sort(int[] arr, int left, int right) {
        if(left > right) {
            return;
        } else {
            int base = divide(arr, left, right);
            sort(arr, left, base - 1);
            sort(arr, base + 1, right);
        }
    }
    public static int divide(int[] arr, int left, int right) {//重新找基准数
        int base = arr[left]; //以最左边的数 (left) 为基准
        while (left < right) {
            while (left < right && arr[right] >= base) {// 从数组右端开始，向左遍历，直到找到小于base的数
                right--;
            }
            arr[left] = arr[right];// 找到了比base小的元素，将这个元素放到最左边的位置
            while (left < right && arr[left] <= base) {// 从数组左端开始，向右遍历，直到找到大于base的数
                left++;
            }
            arr[right] = arr[left];// 找到了比base大的元素，将这个元素放到最右边的位置
        }
        // 最后将base放到left位置。此时，left位置的左侧数值应该都比left小,而left位置的右侧数值应该都比left大。
        arr[left] = base;
        return left;

    }
```

```
}
```

时间复杂度：最坏 -> O（N^2）、最好 -> O（NlogN）、平均 -> O（NlogN）

空间复杂度：O（NlogN）

稳定性：**不稳定**

4、快排最坏的情况下说一下，最坏情况下的时间复杂度是多少?

要看枢轴（pivot）的选择策略。在快速排序的早期版本中呢，最左面或者是最右面的那个元素被选为枢轴，那最坏的情况就会在下面的情况下发生啦：

- 数组已经是正序（same order）排过序的（枢轴最右）。
- 数组已经是倒序排过序的（枢轴最左）。
- 所有的元素都相同（1、2的特殊情况）

最坏 -> O（N^2）

快排改进：

- 切换到插入排序：因为快速排序在小数组中也会递归调用自己，对于小数组，插入排序比快速排序的性能更好，因此**在小数组中可以切换到插入排序。**
- 三数取中：最好的情况下是每次都能**取数组的中位数作为切分元素**，但是计算中位数的代价很高。一种折中方法是取 3 个元素，并将大小居中的元素作为切分元素。
- 三向切分：对于**有大量重复元素的数组，可以将数组切分为三部分，分别对应小于、等于和大于切分元素。**三向切分快速排序对于有大量重复元素的随机数组可以在线性时间内完成排序。

5、完全二叉树和满二叉树知道吧？它们有啥区别?

1）满二叉树 ： 节点数是固定的数 。它的最后一层全部是叶子节点，其余各层全部是非叶子节点 。

2）完全二叉树 ： 节点个数是任意的 ，和满二叉树的区别是，他的最后一行可能不是完整的，但绝对是右方的连续部分缺失 。

6、网络编程中的epoll和select知道吗？socket呢?

select/poll/epoll 都是 **I/O 多路复用的具体实现**，select 出现的最早，之后是 poll，再是 epoll。

- select 允许应用程序监视一组文件描述符，**等待一个或者多个描述符成为就绪状态**，从而完成 I/O 操作；
- poll 的功能与 select 类似，也是等待一组描述符中的一个成为就绪状态；
- epoll **只需要将描述符从进程缓冲区向内核缓冲区拷贝一次，并且进程不需要通过轮询来获得事件完成的描述符。** epoll 比 select 和 poll 更加灵活而且没有描述符数量限制。 epoll 对多线程编程更有友好，一个线程调用了 epoll_wait() 另一个线程关闭了同一个描述符也不会产生像 select 和 poll 的不确定情况。

比较：select 和 poll 的功能基本相同，不过在一些实现细节上有所不同。

- **select 会修改描述符，而 poll 不会**；
- select 的描述符类型使用数组实现，FD_SETSIZE 大小默认为 1024， 因此默认只能监听 1024 个描述符。如果要监听更多描述符的话，需要修改 FD_SETSIZE 之后重新编译；而 **poll 没有描述符数量的限制**；
- **poll 提供了更多的事件类型，并且对描述符的重复利用上比 select 高**；
- 如果一个线程对某个描述符调用了 select 或者 poll，另一个线程关闭了该描述符，会导致调用结果不确定；

套接字（**socket**）是一个抽象层，**应用程序可以通过它发送或接收数据，可对其进行像对文件一样的打开、读写和关闭等操作。** 套接字允许应用程序将I/O插入到网络中，并与网络中的其他应用程序进行通信。 **网络套接字是IP地址与端口的组合。**

7、哈希表了解吗？说一下？

- 散列表（Hash table，也叫哈希表），是**根据键（Key）而直接访问在内存存储位置的数据结构。**
- 哈希表充分体现了算法设计领域的经典思想：**空间换时间**，哈希表是时间和空间之间的平衡；

哈希函数的设计：

- 直接地址法
- 除留余数法；
- 平方取中法；

哈希冲突的处理方法：

- **开放地址法**：开放（每个地址对任何元素开放）。**线性探测**（遇到哈希冲突+1）、**平方探测**（遇到冲突 + 1+ 4 + 9 + 16）、**二次哈希**（遇到哈希冲突 + hash2（key））；
- **链地址法**：封闭；
- **再哈希法**：用另外哈希函数找索引；

8、给你一篇英文文章，如何从中找出出现次数前十的单词？

分治法 + HashMap + 优先队列

- 从头到尾遍历文件，从文件中读取遍历到的每一个单词。
- 把遍历到的单词放到hash_map中，并统计这个单词出现的次数。
- 遍历hash_map，将遍历到的单词的出现次数放到优先级队列中。
- 当优先级队列的元素个数超过k个时就把元素级别最低的那个元素从队列中取出，这样始终保持队列的元素是k个。
- 遍历完hash_map，则队列中就剩下了出现次数最多的那k个元素。

9、给你两个字符串，如何检测一个字符串是不是另一个字符串的子序列，子序列是啥你知道吧？（KMP。。。会怎么写代码）

10、线程与进程的区别？

- 进程是**资源分配**的基本单位。
- 线程是**独立调度**的基本单位。一个进程中可以有多个线程，它们共享进程资源。
- **协程是属于线程的。**协程程序是在线程里面跑的，因此协程又称微线程和纤程等。协程的调度切换是用户(程序员)手动切换的,因此更加灵活,因此又叫用户空间线程。原子操作性。由于协程是用户调度的，所以不会出现执行一半的代码片段被强制中断了，因此无需原子操作锁。

区别：

- 拥有资源： **进程是资源分配的基本单位，但是线程不拥有资源**，线程可以访问隶属进程的资源。
- 调度： **线程是独立调度的基本单位，在同一进程中，线程的切换不会引起进程切换，从一个进程中的线程切换到另一个进程中的线程时，会引起进程切换。**
- 系统开销： **由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，所付出的开销远大于创建或撤销线程时的开销。**类似地，在进行进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。
- 通信方面： **线程间可以通过直接读写同一进程中的数据进行通信，但是进程通信需要借助 IPC。** 进程间通信（**IPC**，Inter-Process Communication）。

11、进程间通信的方式有哪几种？用过哪些方式？

进程通信：**进程间传输信息。**

- 管道：只支持**半双工通信**（单向交替传输）；只能在父子进程或者兄弟进程中使用。

- FIFO：也称为命名管道，去除了管道只能在父子进程中使用的限制。
- 消息队列：消息队列**可以独立于读写进程存在，**从而避免了 FIFO 中同步管道的打开和关闭时可能产生的困难；**避免了 FIFO 的同步阻塞问题**，不需要进程自己提供同步方法；**读进程可以根据消息类型有选择地接收消息**，而不像 FIFO 那样只能默认地接收。
- 信号量：它是一个**计数器**，用于**为多个进程提供对共享数据对象的访问。**
- 共享存储：允许多个进程共享一个给定的存储区。因为数据不需要在进程之间复制，所以这是最快的一种 IPC 。
- 套接字：与其它通信机制不同的是，它**可用于不同机器间的进程通信。**

12、进程同步?

进程同步：**控制多个进程按一定顺序执行**；

- 临界区：对临界资源进行访问的那段代码称为临界区。
- 同步与互斥：同步，多个进程因为合作产生的直接制约关系，使得进程有一定的先后执行关系。互斥，多个进程在同一时刻只有一个进程能进入临界区。
- 信号量：信号量（Semaphore）是一个整型变量，可以对其执行 down 和 up 操作，也就是常见的 P 和 V 操作。
- 管程：使用信号量机制实现的生产者消费者问题需要客户端代码做很多控制，而**管程把控制的代码独立出来，不仅不容易出错，也使得客户端代码调用更容易。**

13、孤儿进程和僵尸进程是什么?

孤儿进程：**一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。**孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：**一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。**

14、tcp/udp ?

- 用户数据报协议 UDP（User Datagram Protocol）是**无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。**
- 传输控制协议 TCP（Transmission Control Protocol）是**面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。**

15、测试方法知道哪些?

1）单元测试：完成最小的软件设计单元（模块）的验证工作；

2）集成测试：通过测试发现与模块接口有关的问题。

3）系统测试：是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。

4）回归测试：回归测试是指在发生修改之后重新测试先前的测试用例以保证修改的正确性。

5）验收测试：验收测试是指系统开发生命周期方法论的一个阶段，这时相关的用户或独立测试人员根据测试计划和结果对系统进行测试和接收。

黑盒测试： 也称**功能测试或数据驱动测试**，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

常用的黑盒测试方法有：**等价类划分法；边界值分析法**；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

白盒测试： 也称为**结构测试或逻辑驱动测试**，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了一些与数据相关的错误。

白盒测试中的逻辑覆盖包括**语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。**

16、linux命令?

- **删除非空目录：rm -rf file目录**
- 查看文件：cat
- **改变文件读、写、执行等属性 chmod**
- **改变文件读、写、执行等属性 chmod**
- 管道：批处理命令连接执行，使用 |
- FIND： 文件查找
- Grep：文本搜索，**-c 统计文件中包含文本的次数**
- 查询进程：ps -ef ，查询归属于用户xuexue的进程： ps -ef | grep xuexue
- 列出所有端口(包括监听和未监听的)： netstat -a
- 性能监控：
  - 监控CPU： sar -u
  - 查询内存： sar -r 1 2
  - 查询硬盘使用： df -h
- 查看当前目录所占空间大小： du -sh

17、说下sql调优？

查询性能优化：

- 使用Explain进行分析，Explain 用来分析 SELECT 查询语句，开发人员可以通过分析 Explain 结果来优化查询语句。
- 优化数据访问：

  1）减少请求的数据量：

  - 只返回必要的列：最好不要使用 SELECT * 语句。
  - 只返回必要的行：使用 LIMIT 语句来限制返回的数据。
  - 缓存重复查询的数据：使用缓存可以避免在数据库中进行查询，特别在要查询的数据经常被重复查询时，缓存带来的查询性能提升将会是非常明显的。

  2）减少服务器端扫描的行数：最有效的方式是使用索引来覆盖查询。
- 重构查询方式：
  - 切分大查询：一个大查询如果一次性执行的话，可能一次锁住很多数据、占满整个事务日志、耗尽系统资源、阻塞很多小的但重要的查询。
  - 分解大连接查询：将一个大连接查询分解成对每一个表进行一次单表查询，然后在应用程序中进行关联。

18、计算机网络，说下拥塞控制？大端小端了解吗？

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而拥塞控制是为了**降低整个网络的拥塞程度。**

TCP 主要通过四个算法来进行拥塞控制：慢开始、拥塞避免、快重传、快恢复。

慢开始与拥塞避免：

发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...

注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。

如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。

快重传与快恢复：

在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。

在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。

在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。

慢开始和快恢复的快慢指的是 cwnd 的设定值，而不是 cwnd 的增长速率。慢开始 cwnd 设定为 1，而快恢复 cwnd 设定为 ssthresh。

大端小端：大小端是面向多字节类型定义的，比如2字节、4字节、8字节整型、长整型、浮点型等，单字节的字符串一般不用考虑。 **大端（Big-Endian）就是高字节（MSB）在前，内存存储体现上，数据的高位更加靠近低地址。小端(Little-Endian)就是低字节（LSB）在前，内存存储体现上，数据的低位更加靠近低地址。**

19、cpu调度算法？批处理和交互系统各自有哪些算法？

1）批处理系统：批处理系统**没有太多的用户操作**，在该系统中，调度算法目标是**保证吞吐量和周转时间**（从提交到终止的时间）。

- **先来先服务 first-come first-serverd（FCFS）：** 非抢占式的调度算法，按照**请求的顺序**进行调度 。
- **短作业优先 shortest job first（SJF）** ：非抢占式的调度算法，按**估计运行时间最短的顺序**进行调度。
- **最短剩余时间优先 shortest remaining time next（SRTN）** ：最短作业优先的抢占式版本，**按剩余运行时间的顺序进行调度。** 当一个新的作业到达时，其整个运行时间与当前进程的剩余时间作比较。如果新的进程需要的时间更少，则挂起当前进程，运行新的进程。否则新的进程等待。

2）交互系统：交互式系统**有大量的用户交互操作**，在该系统中调度算法的目标是**快速地进行响应。**

- **时间片轮转** ： 将所有就绪进程按 FCFS 的原则排成一个队列，每次调度时，把 CPU 时间分配给队首进程，该进程可以执行一个时间片。当时间片用完时，由计时器发出时钟中断，调度程序便停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 时间分配给队首的进程。
- **优先级调度** ： 为每个进程分配一个优先级，按优先级进行调度。
- **多级反馈队列** ： 一个进程需要执行 100 个时间片，如果采用时间片轮转调度算法，那么需要交换 100 次。

多级队列是为这种需要连续执行多个时间片的进程考虑，它设置了多个队列，每个队列时间片大小都不同，例如 1,2,4,8,...。进程在第一个队列没执行完，就会被移到下一个队列。这种方式下，之前的进程只需要交换 7 次。

20、request？get和post？

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
缓存	能被缓存	不能缓存
编码方式	只能进行url编码	支持多种编码方式
是否保留在浏览历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	发送数据，GET 方法向 URL 添加数据，但URL的长度是受限制的。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	安全性较差，因为参数直接暴露在url中	因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。
传参方式	get参数通过url传递	post放在request body中。

21、请你说一说http和https区别？

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，**HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。**

- **https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。**
- **http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。**
- **http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。**
- **http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。**

22、软连接和硬连接？

- 硬链接： 与普通文件没什么不同，`inode` 都指向同一个文件在硬盘中的区块
- 软链接： 保存了其代表的文件的绝对路径，是另外一种文件，在硬盘上有独立的区块，访问时替换自身路径。

23、git怎么提交代码？

- 执行指令进行初始化，会在原始文件夹中生成一个隐藏的文件夹.git （ git init）
- 执行指令将文件添加到本地仓库： （ git add . //添加当前文件夹下的所有文件）
- git commit -m "layout" //引号中的内容为对该文件的描述
- 关联git仓库：新建一个repository时会出现下面的代码，直接复制即可：

git remote add origin <https://github.com/xuexuehan/first_spring_mybatis.git

- git push -u origin master -f

24、git pull和fetch区别？

git fetch和git pull都可以将远端仓库更新至本地， 与git pull相比git fetch相当于是从远程获取最新版本到本地，但不会自动merge。如果需要有选择的合并git fetch是更好的选择。效果相同时git pull将更为快捷。