计算机网络基础

1. 拓扑分类: 星形/树形/总线/网状/环状 - 局域网/城域网/广域网 - 有线/无线
2. 传输方式:

按传输方向区分 - 单工(单方向传输) 耳机

- 双工(同一时间內只允许一个信号通信) 对讲机

- 全双工(同一时间可来可回) 手机

按传输对象区分 - 单播 一对一

- 多播 一对多

- 广播 对一定范围內所有用户

1. 数据交换
   1. 电路交换Cs 一段数据包完整从头传到尾

缺点 占用信道 优点 速度快

* 1. 报文交换 中间节点

优点 效率略高不占用整个信道 缺点 数据包庞大

* 1. 分组交换Ps 分组传输+目的地拼组

优点 效率最高 缺点 速度慢 分组和重组导致完整性不如Cs

1. 网络协议 - 语法语义时序
2. TCP/UDP的优缺点

TCP的优点: 可靠 稳定(重传机制/流量控制/拥塞控制) 有序 /缺点: 慢 效率低 占用系统资源 容易被攻击

UDP的优点: 无需握手(快) 攻击漏洞少(没有重传机制) / 缺点: 不可靠 容易丢包

1. TCP/UDP 的可靠传输实现

TCP的可靠性实现: ACK应答机制 / 重传机制(超时重传 快速重传 SACK DSACK) / 流量控制(滑动窗口) / 拥塞控制 (维护一个拥塞窗口,设定一个门限) (慢启动 拥塞避免 拥塞解决 快速恢复)

UDP的可靠性实现: UDP本身没有类似于TCP的可靠控制，主要依赖应用层解决可靠性，即在应用层模仿TCP设计ACK应答机制 / 重传机制 简单来讲就是发送方应用程序随机序列号x 接收方应答号x+1 发送方经过一段时间收到ack就删除缓冲区数据 否则就进行重传

1. TCP/UDP的区别

TCP面向连接(三次握手) ，UDP无需握手

TCP通过重传机制、流量控制、拥塞控制提供可靠服务，单论UDP没有可靠性保证，即即便网络堵塞或者接收方缓冲区满也不会降低传输速率，容易丢包，但实时性比较好

TCP是点对点通信，UDP支持广播，多对多

TCP消耗更多的系统资源，UDP消耗更少的系统资源

1. OSI 七层协议

物理层

机械特性 / 电气特性 / 功能特性 / 过程特性

数字信号-编码 / 模拟信号-调制

网络层

ip地址 - 在逻辑上是主机的唯一标记

ipv4和ipv6

三四层的本质区别:

*第三层 点到点 (端口到端口) 第四层 端到端(客户端到服务器)*

*第三层给第四层提供基础 第四层控制第三层*

1. Internet采用哪种网络协议？ TCP / IP协议
   1. IP地址的编码？ 网络号和主机号
      1. CIDR a.b.c.d/x x表示前面x位是网络号
      2. 子网掩码 和IP地址AND运算 得到的就是网络号
   2. *IP和MAC的作用区别：*

IP负责没有直连的两个网络的连接

MAC负责直接连接的两个设备的通信

* 1. IPv6和IPv4
     1. 更多地址
     2. 去掉了IP包头的校验和，传输性能更优
     3. 安全性好

1. DNS解析
   1. 用方便记忆的域名代替不方便记忆的IP地址

数据结构和算法

1. 栈

特点: 一端操作(添加删除) 后入先出(LIFO)

举例: 网页回退 刷盘子 羽毛球盒 求二进制 递归

堆栈溢出的原因 没有回收垃圾资源

1. 队列

特点: 两端操作(头删/读取，尾加) 先入先出(FIFO)

举例: 排队

类型:

双端队列(头尾都可以添加删除元素)

循环队列(头尾相连，成一个环)

1. 二叉树

[1] 遍历方式：前序、中序、后序、层序

[2] 类型：满二叉树、完全二叉树（最后一层集中在最左边），平衡二叉树(AVL树，左右子树的高度差不超过1)，二叉查找树

[3] 平衡二叉树和二叉树的优缺点 平衡二叉树任意节点的左右子树高度差不超过1，使得它的查找效率比较高O(logn)，但是增加和删除节点的成本也很大

普通二叉树如果一直在单一子树上存放数据，就会变得过于臃肿，导致查找效率变差，但普通二叉树的增加和删除节点操作比较简单，不需要旋转二叉树

什么是平衡二叉树：

左右子树都是平衡二叉树，左右子树的高度差不大于1

红黑树和AVL树的区别:

旋转次数：AVL树是严格的平衡二叉树，左右孩子高度差不能大于1，红黑树不严格的平衡，添加或删除节点的旋转次数比较少

应用：AVL树适合搜索更多的场景，红黑树适合添加删除节点更多的场景

B树和B+树的区别：

B树是多叉的搜索树，遵循从左到右元素依次增大的规则

B+树

1.在查询速度上，所有的数据保存在叶子节点，查询更加稳定

2.在排序方面，所有叶子节点都是数据链表，数据紧密性更强

3.在遍历速度方面，只需要遍历叶子结点即可，不需要层层遍历，速度更快

1. 链表

存储方式: 链式存储而不是连续存储（指针域+数据域）

类型：单链表、双链表（两个指针域，向前/向后查找）、循环链表(首尾相接)

比较数组和链表: 数组是静态的（固定长度），链表是动态的（可以增加或删除节点）

数组随机访问 查询O(1) 链表是依次查找 O(n)

数组添加或删除需要修改剩余元素的内存地址 O(n) 链表是链式存储，因此添加和删除 O(1)

1. 哈希表（散列表）

根据key-value形式存储数据的map

哈希表的实现：映射函数的构造

- 除留余数法 选择质数p key = val % p

- 直接定位法 线性函数 ax+b

- 数字分析法 手机的电话号码

- 平方取中法 val \* val 取中间若干位

哈希表的问题：哈希冲突

解决：1) 开放地址法（默认hashtable大小大于数据大小）

- 线性探测 除留余数法 (val+1)%p = key 直到找到空

优点：总能找到空的地址

缺点：大量元素聚集出现，查找效率低

- 平方探查 每次加上 1^2 / 2^2 / 3^2

1. 拉链法
   1. 思想：具有相同key的元素以单链表的形式存储
   2. 优点：没有改变哈希函数的规则
   3. 缺点：大量元素在同一个key链表下 / 查找起来浪费时间
2. 再哈希法
   1. 思想：使用不同的哈希函数，有冲突就换函数
   2. 优点：不容易产生聚集
   3. 缺点：运算慢
3. 公共溢出区
   1. 思想：收容所，收容所有冲突的元素
4. 排序算法
   1. 快速排序
      1. 时间复杂度 O (nlogn)
      2. 思想：
         1. 采用基准值的思想，先选取某一段数组的基准值，把所有小于基准值的数放到其前面，把所有大于基准值的数放到后面（不动），最后交换基准值并返回基准值节点
         2. 采用递归和分而治之的思想，返回的基准值作为切片的中点，将两侧的数组进行递归排序，因为左侧的数组元素都小于基准值，右侧的数组元素都大于基准值，所以最后出来的结果一定是严格升序的
   2. 冒泡排序
      1. 时间复杂度O(n^2)
      2. 思想:
         1. 比较相邻的元素，左侧大于右侧则交换元素，直到遍历至列表尾巴，确定最后一个元素
         2. 重复以上的顺序n-1次，直到确定第二个元素，不再遍历
   3. 选择排序
      1. 时间复杂度O(n^2)
      2. 思想:
         1. 每次遍历时寻找剩余数组的最小元素，和头指针元素交换
         2. 头指针+1 继续重复上述过程直到头指针到n-1位置
   4. 插入排序
      1. 时间复杂度O(n^2)
      2. 思想:
         1. 类似于打扑克
         2. 新遍历的元素向前遍历，如果小于左边元素则交换，直到遇到比他大的或者数组头
         3. 向后寻找新的数组元素，并重复以上步骤
5. BFS和DFS的区别:
   * 1. 遍历方式不同：
        1. BFS是分层逐步搜索，适合地图节点的广度搜索
        2. DFS是深搜+回溯，找到每种可能性并返回，类似于不撞南墙不回头
     2. 实现的方法不同:
        1. BFS是基于队列，先入先出，每层摘出所有队列元素，然后放入新元素
        2. DFS是基于栈，先入后出，类似于递归的思想，其实就是函数的嵌套返回

操作系统

1. 进程之间的通信
   1. 需要操作系统的支持: 因为进程间的地址空间彼此独立，不能访问其他进程的地址空间
   2. 通信方式
      1. 管道
         1. 半双工 比如对讲机
         2. FIFO 类似于循环队列
         3. 会阻塞 - 写满管道时写堵塞 / 管道为空时读阻塞
         4. 数据错乱 - 因为多个进程同时读数据 - 解决方案就是允许多个写进程，但只允许一个读进程
         5. 无名管道
            1. 只能在父子进程里使用
            2. python multiprocessing库里的Pipe
         6. 有名管道
            1. 半双工
            2. 不强制进程的血缘关系
      2. 共享存储
         1. 基于存储区的共享
            1. 操作系统开辟一个共享存储区，两个进程都可以访问这里的数据
            2. 保证进程对数据存储区的访问是互斥的 -> PV操作
            3. 速度快 / 高级通信
         2. 基于数据结构的共享
            1. 只共享特定数据结构的内存
            2. Python multiprocessing库里的Array Value
            3. 速度慢 / 低级通信
      3. 消息传递
         1. 基于格式化的消息传递，消息头和消息体
         2. 分类
            1. 直接通信 指明接收进程ID 消息队列

进程A的发送 - 消息复制到了操作系统内核空间里进程B的PCB的消息队列

进程B的接收 - 消息从操作系统内核空间进程B的消息队列中检查来自A的消息，复制到进程B的地址空间里

python 里 multiprocessing 的Queue

* + - * 1. 间接通信 信箱通信

进程A发送 - 在操作系统内核空间里申请信箱，复制msg

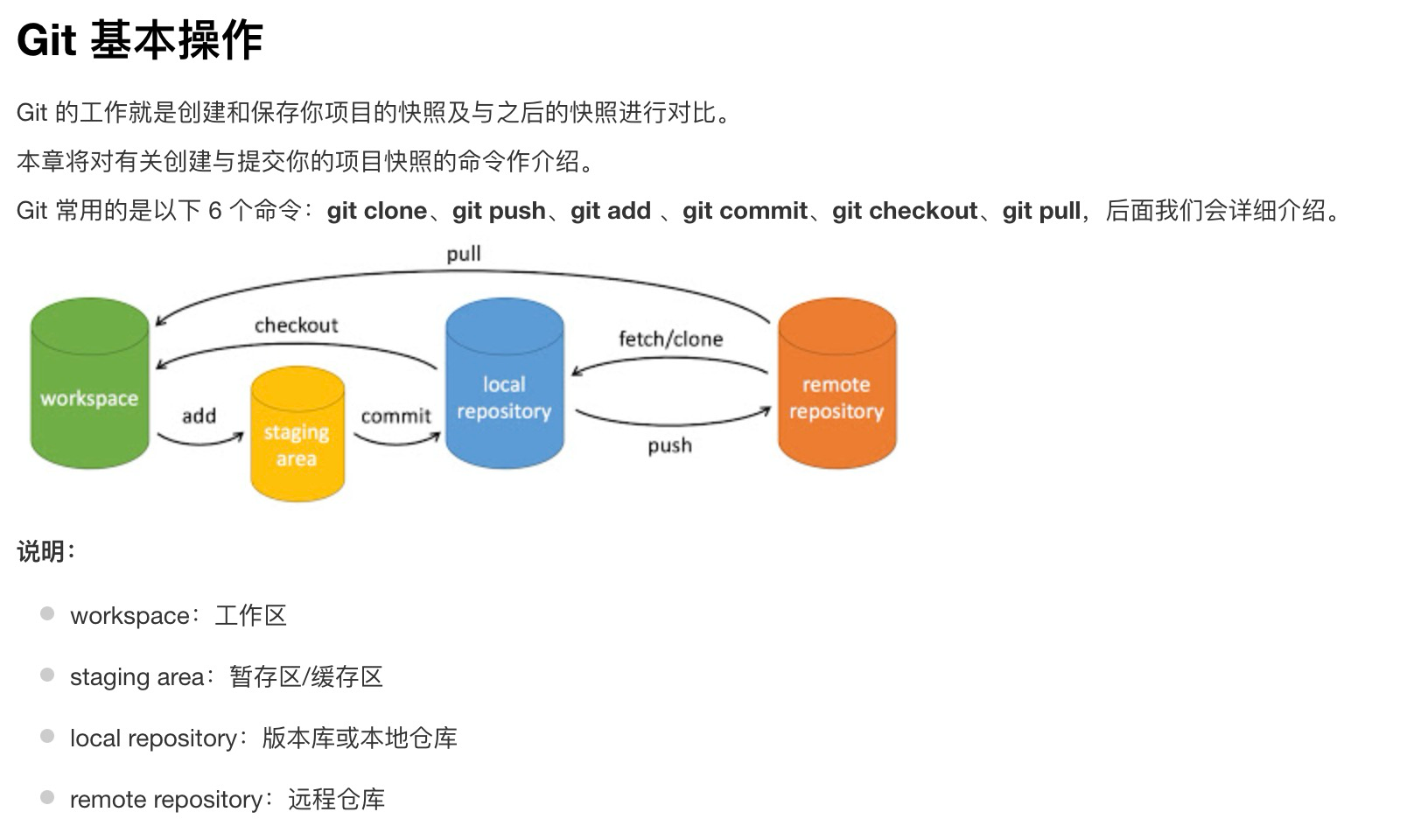
进程B接收 - 指明哪个信箱，再复制消息

1. 进程同步和同步互斥

设计模式

1. 单例模式
   1. 类只有一个实例，新创建的实例可以修改里面的数据 / 但不改变实例存储的地址
   2. 多线程中可能出现地址冲突的问题，可以用锁和上下文管理器解决

Git操作



git fetch 和 git clone

git fetch可以指定分支

git clone 不用 init，更加快速方便。直接拷贝