

7.3 死锁的条件？如何解决

死锁：多个进程可以竞争有限数量的资源。

当一个进程申请资源时如果没有可用资源，就会进入“等待状态”。如果所申请的资源被其他进程占有，该等待进程可能再也不能改变状态。称为死锁。

4个条件：

- ①互斥：资源处于非共享模式，一次只有一个进程可以使用
- ②占有并等待。一个进程占有至少一个资源并等待另一个资源，而该资源被其他进程占有。

③ 非抢占：资源不能被抢占。只有持有资源的进程结束后，资源才能被释放。

④ 循环等待。有一组等待进程 $\{P_0, P_1, \dots, P_n\}$
 P_0 等待的资源被 P_1 占有, P_1 等待的资源被 P_2 占有... P_n 等待的资源被 P_0 占有。

如何解决？

破坏(2)：进程在运行前一次性申请所有资源，或者在申请新资源时必须释放已占有资源

破坏(3)：进程可以抢占其他进程的资源

7.1 进程和线程 (process, thread)

以Java为例，一个进程可以有多个线程，多个线程共享进程的堆和方法区资源，但每个线程有自己的程序计数器、虚拟机栈和本地方法线程。线程是进程划分的更小的执行单元，一个进程在执行的过程中可以产生多个线程，各进程保持独立，而线程不一定，同一进程的不同线程可以互相影响。线程执行开销小，但不利于资源管理和保护，而进程相反。

6.30 go 的基本数据类型

数字：整型： int8 int16 int32 int64
1B 2B 4B 8B

uint8 uint16 uint32 uint64

uint = uint32 or 64 int = int32 or 64

取决于操作系统的位数

unsafe.Sizeof(100) // 8 (8B)

-浮点型： float32, float64

复数： complex64 complex128

布尔值 true, false

字符串：双引号，UTF-8

len(string) 返回字节数

`len("a") // len("我") 13`

字符: byte 相当于 uint8, 代表一个 ASCII 字符
rune 相当于 int32 代表一个 UTF-8 字符

遍历字符串: for _, r := range s {} ✓

for i := 0; i < len(s); i++ { }

06.29. 悲观锁和乐观锁

乐观锁和悲观锁，是为了解决并发场景下的数据竞争问题。

乐观锁：认为在操作数据时不会由别的线程操作数据。
不会上锁。在更新的时候判断在此期间是否被修改。
如果没被修改，则执行操作。否则不执行（会自旋，
不停重试）

悲观锁：认为会由别的线程操作数据。直接上锁。

面试：乐观锁：① CAS ② 版本号 悲观锁：加锁

① CAS (compare and snap)
3个操作数：读写的内存位置 (V)，进行比较的预期
值 (A) 打入写入的新值 (B)
如果 V 中的值等于 A，那么写入 B。否则不停重试。
(CAS 是 CPU 支持的原子操作，硬件层面保证)

② 版本号机制。数据中增加一个字段 version。每修改一次，
版本号 +1。访问时同时读取版本号。更新数据时，如果版

越多设置则更新。

并发少，读多：乐观锁

并发多，写多：悲观锁

CAS：ABA问题 $A \rightarrow B \rightarrow A$

高并发下不停重试开销大

功能受限

CAS只能保证单变量操作原子性

06.28 go 中 HashMap 的实现

go 中声明并初始化 HashMap:

map := make(map [String] String)

key类型 value类型

key 的种类可以是：布尔、数字、string、指针、chan、interface、struct。只包含上述类型的数组。

不能是：slice, map, function.

只要能支持 == 和 != 操作 就可以做为 key.

HashMap 基本原理：① hash 函数 ② 如何解决冲突。

go 的实现：① 在 runtime/algos 中定义了各种基础类型的 hash func
及 equal func。得到 hash 值后用与运算得到桶的编号

② 一般有开放地址法和拉链法。go 中

用的是拉链法。

桶的结构 bmap 每个桶可以存 8 个 k-v 对。

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	tophash, 存 hash 的前几位
k_1	k_2	k_3	k_4					
k_5	k_6	k_7	k_8					
v_1	v_2	v_3	v_4					
v_5	v_6	v_7	v_8					

overflow * bmap 溢出桶 (8 个装不下可以再链一个桶)

常规桶的个数为 2^B , - 这是 2 的幂次。hash 值 $(2^B - 1)$ 与运算后定位桶编号 与运算与取模效
果一致, 但更快, 需要与 2 的幂次才可以。

如果 $B \geq 4$, 溢出桶的个数为 2^{B-4} , 否则无溢出桶
平均分配

hmap {

count int

flags uint8

B uint8

noverflow uint8

hash0 uint32

buckets unsafe.Pointer

oldbuckets unsafe.Pointer

nevacuate uintptr (接下来迁移的桶)

*mapextra

}

{ overflow *[]*bmap
oldoverflow *[]*bmap
nextOverflow *bmap

扩容规则

$\frac{\text{count}}{(2^B)} > 6.5 \rightarrow$ 翻倍扩容
Load factor

Load factor $\leq 6.5 \rightarrow$ 等量扩容
no overflow

$B \leq 15, \text{ nooverflow} \geq 2^B$

$B > 15, \text{ nooverflow} \geq 2^{15}$

扩容不是一步到位，是一个扩容，迁移分摊到多次的待表操作中：渐近式扩容，可以避免一次性扩容带来的性能瓶颈。

06.27 红黑树、B树和B+树.

红黑树是二叉树，大规模数据存储时红黑树很深，所需 I/O 很多，效率低下，对存储设备寿命的影响也很大。因此，我们需要使用多叉的搜索树，来降低树的高度，提高效率。

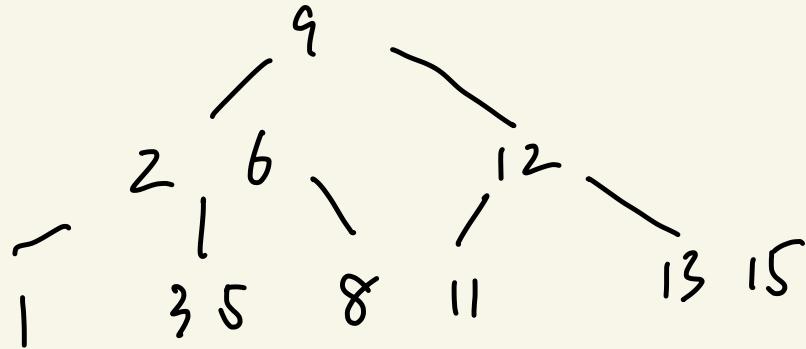
B树：m 叉的搜索树，m 称为 B 树的阶，m 取决于磁盘页的大小。 $m > 2$ 。

~~根结点的孩子树：~~ $[2, m]$

~~非根非叶结点的孩子数：~~ $[m/2, m]$

非叶结点的关键字个数 ($k - vxt$)：孩子数 - 1。

k 个关键字个数把节点拆成 $k + 1$ 段，指向 $k + 1$ 个孩子。

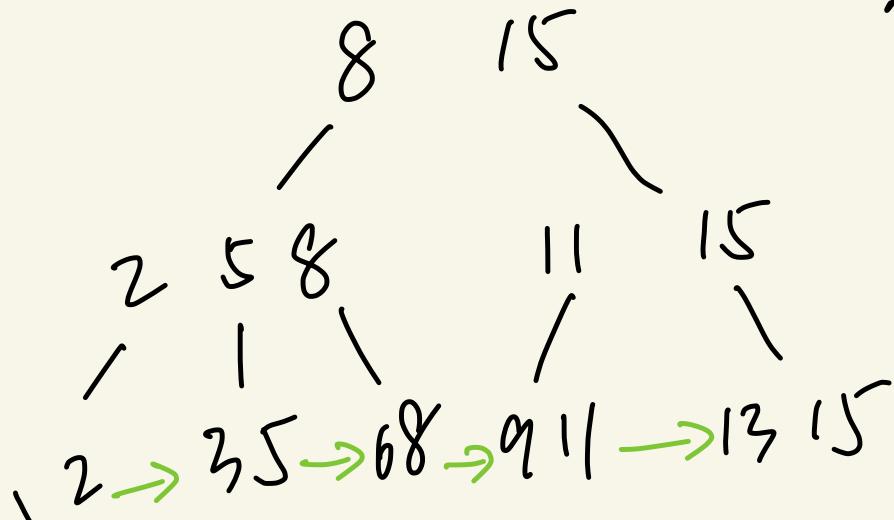


所有叶子节点位于同一层。

与B树不同的点：关键字分布在整棵树
任何关键字只出现一次
搜索可能停止在非叶节点。

B+树：一个节点， k 个关键字，拆成 k 段，指向 k 个孩子
(B 树会差 1)，根结点至少

关键字是子树的最大元素，一些关键字是
出现多次的。



非叶节点不存储 V ，
只作索引

叶节点才存 $k - V$
所有叶节点用链表相连，range-query 方便

3孩子个数，关键字个数：

B：每个节点最多有 m 个孩子；根节点最少 2 个孩子，其他非叶节点最少 $\lceil m/2 \rceil$ 个孩子。非叶节点：关键字个数 = 孩子数 - 1。

B+：3孩子数的情况和B树一样，关键字数 = 孩子数

B+树优点：非叶节点不存储 V 的信息，进一步压缩空间， m 可以进一步变大。range-query 方便（B 树只在磁盘中有序表示）

B 树：可能找不到叶节点就找到更快。（B+树：只能找到叶节点找到 V，但数据完整性能更稳定）。

06.26 Python 内存管理特点

1. 引用计数

Python 内部记录了对象有多少个引用。对象创建时会创建一个引用计数，当引用计数为0，会被

GC.

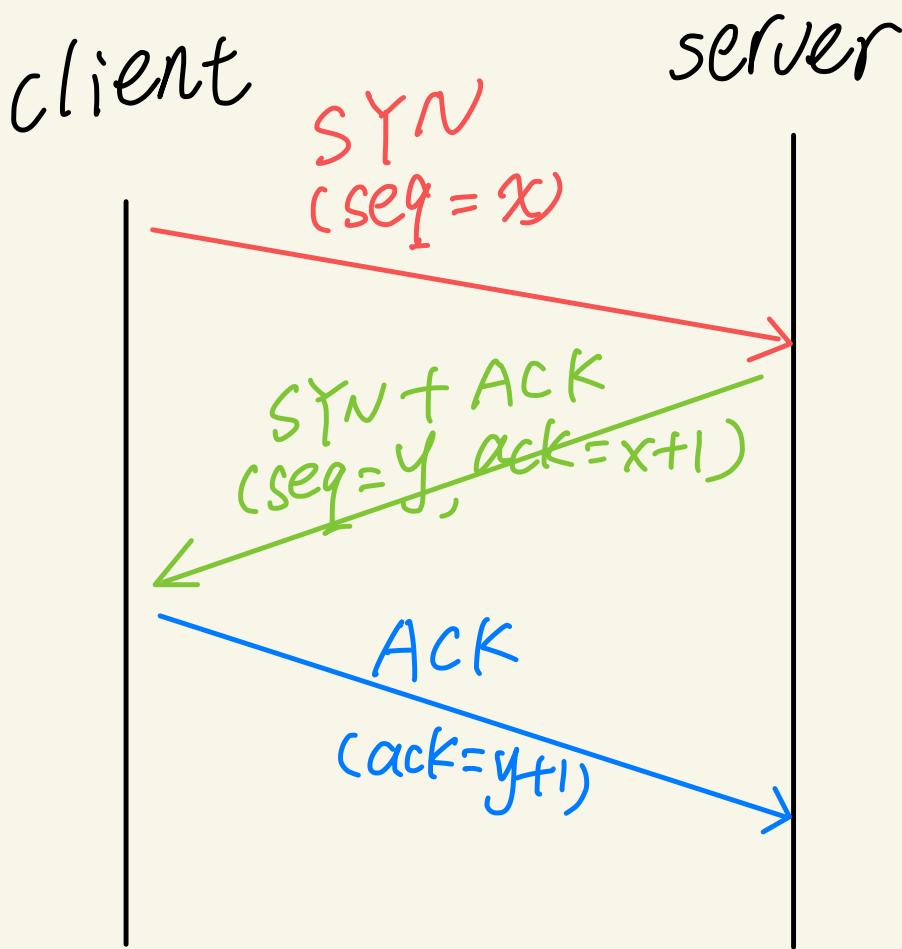
- ① 分配新名称/放入容器，引用计数增加
- ② del / 引用超出作用域 / 重新赋值 引用计数减少
- ③ sys.getrefcount() 获取引用计数

2. GC：检查引用计数为0的对象 清除内存空间

3. 内存池：很多小内存申请释放频繁，会影响效率。引入内存池机制，用于管理小块内存。能够减少内存碎片。

- ① GC 将内存返还给内存池而非操作系统
- ② 小于 256B 的对象使用 PyMalloc 分配器，大的使用系统 malloc
- ③ Int, Float, List 等之间不共享内存池

06.23 TCP 3-way handshake process



TCP uses the 3-way handshake process to establish a secure and reliable communication between the client and server.

① Client sends the SYN to the server with SYN Flag as 1, means client wants to establish a secure connection with the server. $\text{seq} = X$, which is a random number, assigned to the first bit of the data.

② Server receives the SYN from client, sends SYN+ACK to the client, with SYN and ACK Flag as 1. ack number is set as $x+1$, tell client that the server has successfully receives the previous SYN from the client. Another seq is set as y which is a random number, assigned to the first bit of the data.

③ Client receives the Start ACK from the server, send an ACK to the server, with ack number set as $y+1$.

After the 3-way handshake the TCP connection is established.

06.22

OOP features,

06.21 Describe what happens when you click on a URL in a browser

- ① You type the URL, for example, www.leetcode.com into the browser.
- ② The browser, will contact the DNS (domain name System), to convert the human-readable URL into the numerical IP address.
- ③ With the IP address, the browser can send an HTTP request to the server.
- ④ If the server receives the HTTP request from the browser, it will parse

the request, and send an HTTP response to the browser.

- ⑤ After receiving the response, the browser will render the HTML encoded in the response. If there are additional resources embedded in the HTML, steps 3~5 will be repeated.

4 layer cache:

browser, OS, router, ISP

recursively request the IP address

Root DNS

Top-level .com

Second-level

microsoft.com

Third-level

download.microsoft.com

TCP connection , 3-way handshake

