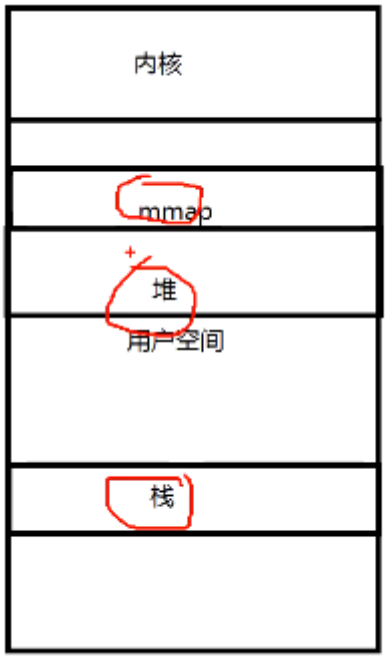
- 3.1.2 内存池的实现与场景分析
  - 内存池的应用场景与性能分析
  - 内存小块分配与管理
  - 内存大块分配与管理
  - 手写内存池，结构体封装与API实现
  - 避免内存泄漏的两种万能方法
  - 定位内存泄漏的3种工具
  - 扩展：nginx内存池实现

内存池：:对一个空白内存维护的过程，其核心时避免频繁的内存分配与释放

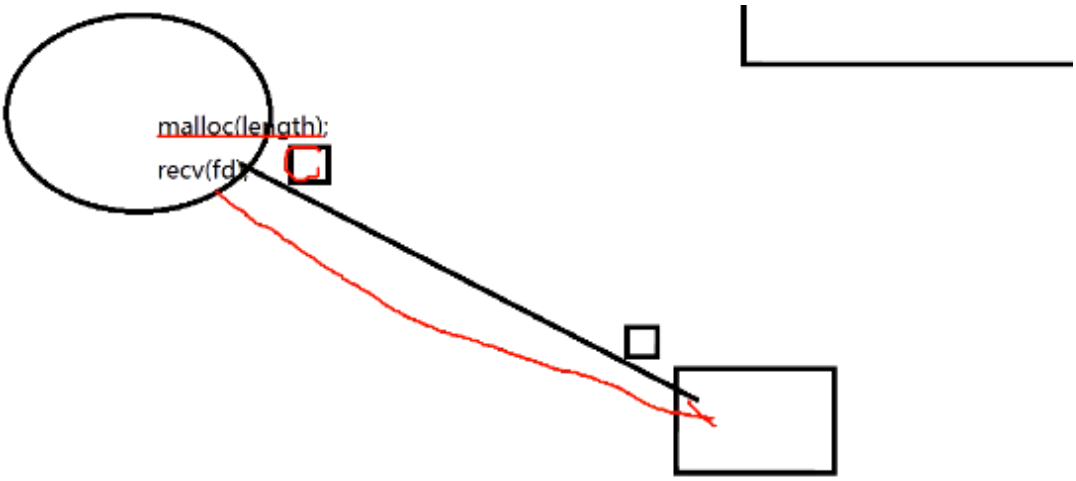
1



内存分为内核空间和用户空间

用户空间：有mmap映射空间　堆 和栈

栈只能定义变量，不能够管理空间，用户只能够管理堆上面的空间，所以内存池实际上是对堆上的空间进行管理的组件



当服务器收到来自客户端发送的数据，如果用一个栈的数组进行保存，没法实现异步的操作，要保证这个数据生命周期要随着这个这个数据的业务(关于数据库，关于redis 等等)业务操作要生存到最后，所以用栈存储数据不太合适，用堆空间存储比较好。那么如果n个客户端都给服务器发送数据，服务器接收数据就会不断地申请堆内存存储收到的数据。这样会造成内存的碎片化，，当我们需要分配一个大块内存的时候，有时候会出现malloc分配失败，返回一个空值，因为堆空间里面没有那么大一块连续的内存。

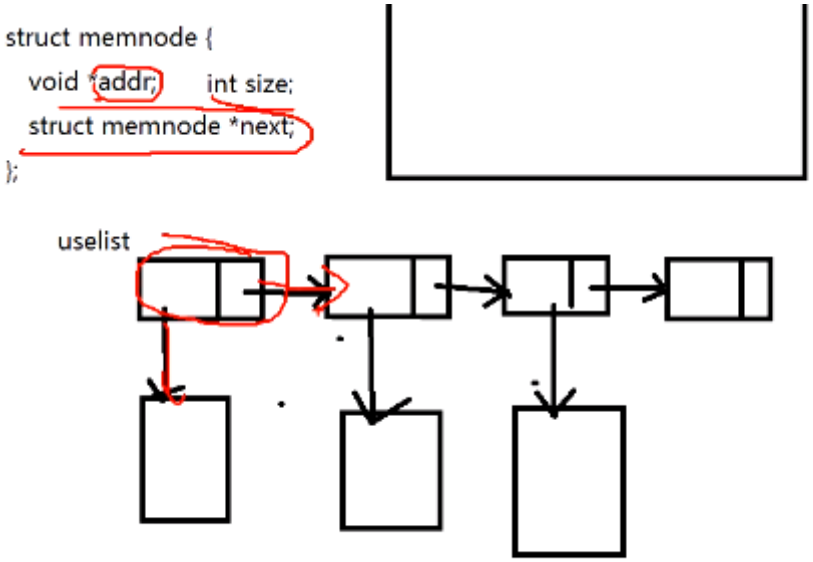
mallc：申请空间后没有置0.，需要memset置0
calloc：在申请空间的同时堆空间置0

内存池用在什么地方，为什么会用内存池?
1 为了避免频繁的在堆空间中申请空间，造成堆空间内存碎片化。
避免频繁的分配与释放

内存池：在工作中不要自己去实现内存池
1 造了这个轮子也不一定跑得起来
2 要用，但是不要自己去造。
但是内存池的原理一定要懂。

开源的内存池组件：
jemalloc：
tcmalloc：内存回收的机制有一些小问题
使用方法：直接在前面加上一个宏定义 malloc 和free就被hook住了。

早期的内存池的制作思路




```c
struct memnode {
 void *addr;
 int size;
 int flag; //1 used ,0 free
 struct memnode *next;
}
struct memnode *mempool;
void *nmalloc() {
 void *addr = malloc();
  struct memnode *node;
  ADD(mempool,node);
 }
 void nfree(void *addr) {



 }

```

缺点：内存块越分越小，直到最后分配不了

第二个版本的内存池

解决办法：分配固定大小，不管用多大的空间，都分配一个1k的空间，但是这样会造成空间的浪费。

然后改变策略

分配固定的不同块的内存大小。

小块：

1 16个字节

2 32个字节

3 64个字节

4 128 个字节

5 256个字节

6 512 个字节

大块：

如果大于512个大小，需要多少就分配多少。并且不加入内存管理的链表，直接返回给操作系统

小块与大块

整理出内存管理的链表结构

缺点：

```
// malloc
void *nmalloc(int size) {

    void *addr = search(usetable, size);
    if (addr == NULL) {
        addr = malloc(size);
        struct memnode *node ;

        ADD(mempool, node);
    }


}

// free
void nfree(void *addr) {

    truct memnode *node = search_from(addr);
    node->flag = 0;

}
```

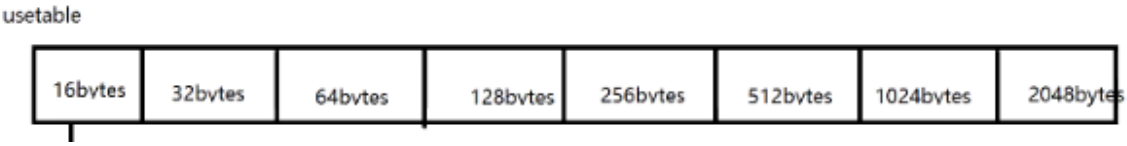
1 内存的地址以链表的方式存储，查找速度慢，申请和释放需要经过两次查找

解决办法：hash rbtree

2 块与块之间存在间隙，没办法合成一个大块，其核心缺点式影响内存的回收。

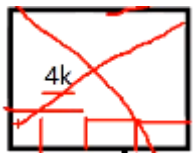
小块的内存回收是一个极其麻烦的事情。

提供一种思路



以16个字节内存为例：

首先一次性分配4k的内存空间挂载到16bytes上面，然后每一次需要一个16个字节空间时，直接从4k的空间中取



，当4k的空间利用完后，欸有剩余空间，就继续再分配一个4k的空间然后以链表的方式和前一个4K的内存块链接起来。

再次使用。

内存池的几种使用场景：

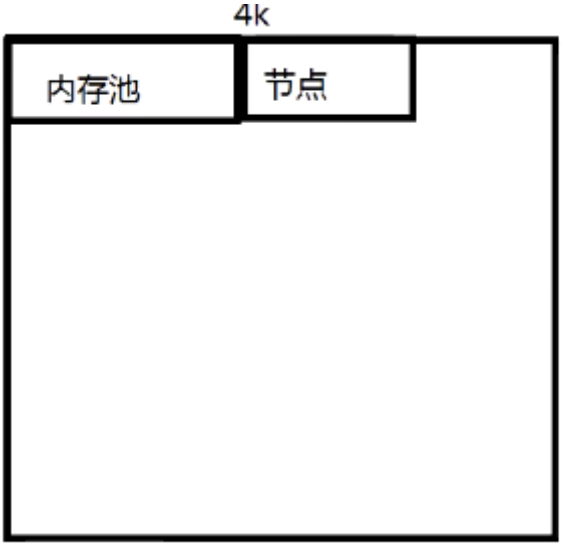
1 全局的内存池 用jemalloc或者tcmalloc 开源封装好的

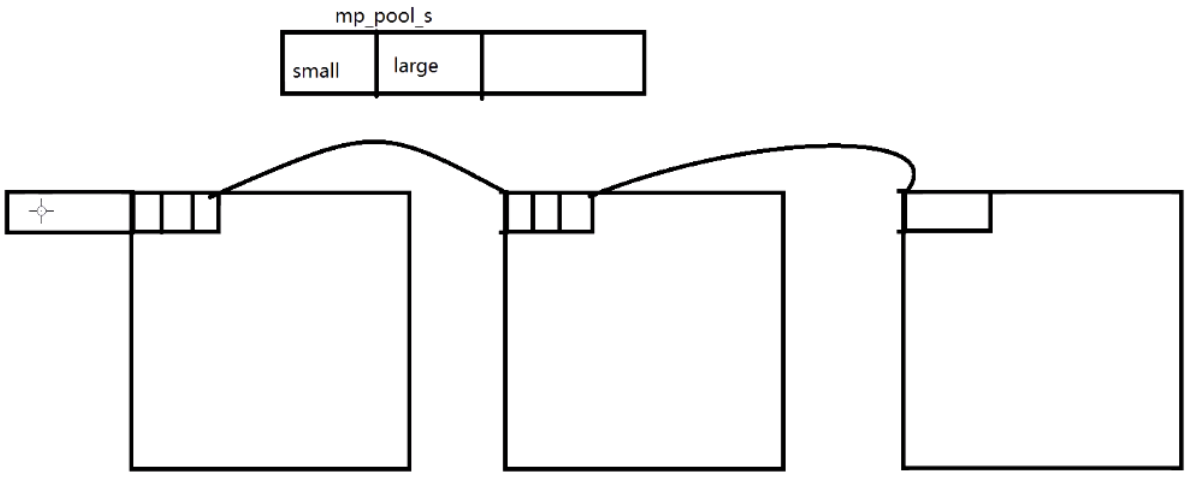
2 一个链接做一个内存池 这种场景，链接的生命周期没有那么长，小块没有必要回收

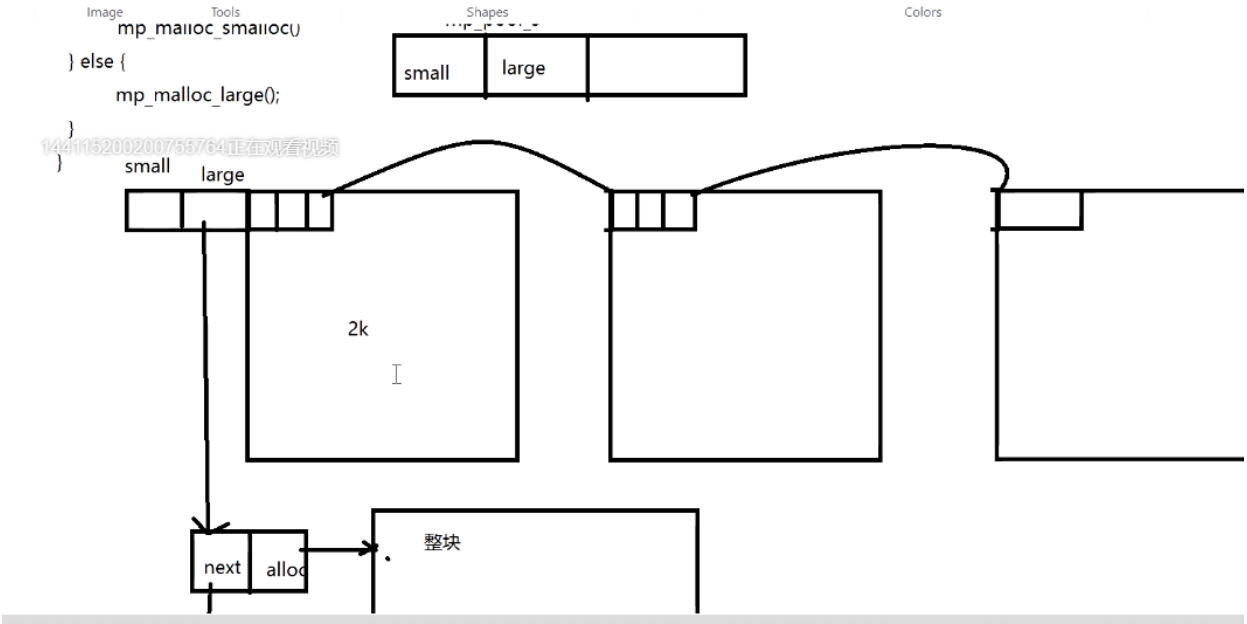
3 每一个消息做一个内存池 ----X 这种方式没有意义。

代码实现：mmpoll.c

一个链接做一个内存池，并没有对小块进行回收。







nginx：一个链接一个内存池

内存池 后期补充：

1 线程池

2 内存池

3 请求池

4 连接池

5 消息队列(消息池) 也是一种池化技术

最容易理解的：线程池 其次 请求池

最不好理解的：内存池 （100个内存池有100个做法，而且还不好理解）

内存池最主要的三个部分：

1 内存块的组织

2 内存的分配

a 分配大块容易理解 直接分配的页

b

3 内存的回收