零声教育出品 Mark 老师 QQ：2548898954

定时器应用

- 心跳检测
- 技能冷却
- 倒计时
- 其他需要延时处理的功能

定时器触发方式

> 对于服务端来说，驱动服务端业务逻辑的事件包括网络事件、定时事件、以及信号事件；通常网络事件和定时事件会进行协同处理；定时器触发形式通常有两种：
>
> - 网络事件和定时事件在一个线程中处理；例如：nginx、redis、memcached；
> - 网络事件和定时事件在不同线程中处理；例如：skynet，...；

```
// 网络事件和定时事件在一个线程中处理
while (!quit) {
    int timeout = get_nearest_timer() - now();
    if (timeout < 0) timeout = -1;
    int nevent = epoll_wait(epfd, ev, nev, timeout);
    for (int i=0; i<nevent; i++) {
        // ... 处理网络事件
    }
    // 处理定时事件
    update_timer();
}
```

```
12
13 // 网络事件和定时事件在不同线程中处理
14 void * thread_timer(void *thread_param) {
15     init_timer();
16     while (!quit) {
17         update_timer();
18         sleep(t);
19     }
20     clear_timer();
21     return NULL;
22 }
23 pthread_create(&pid, NULL, thread_timer,
   &thread_param);
```

定时器设计

接口设计

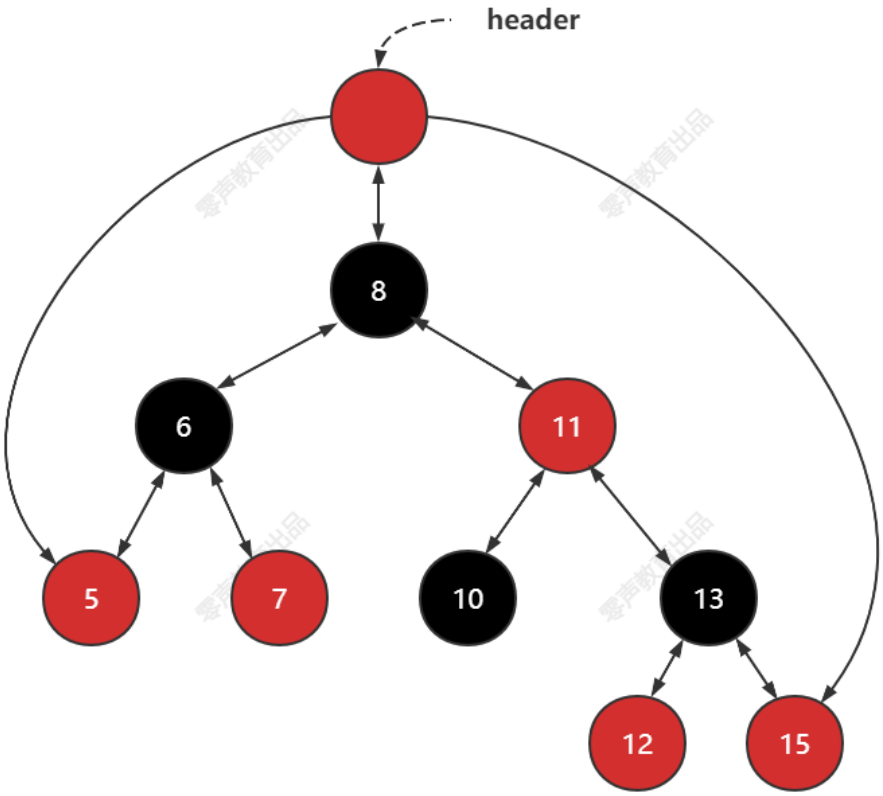
```
1  // 初始化定时器
2  void init_timer();
3  // 添加定时器
4  Node* add_timer(int expire, callback cb);
5  // 删除定时器
6  bool del_timer(Node* node);
7  // 找到最近要触发的定时任务
8  Node* find_nearest_timer();
9  // 更新检测定时器
10 void update_timer();
11 // 清除定时器
12 // void clear_timer();
```

数据结构设计

> 对定时任务的组织本质是要处理对定时任务优先级的处理；由此产生两类数据结构；

- 按触发时间进行顺序组织
  - 要求数据结构有序（红黑树、跳表），或者相对有序（最小堆）；
  - 能快速查找最近触发的定时任务；
  - **需要考虑怎么处理相同时间触发的定时任务**；
- 按执行顺序进行组织
  - 时间轮

红黑树



最小堆

满二叉树：所有的层节点数都是该层所能容纳节点的最大数量（满足 2^n）

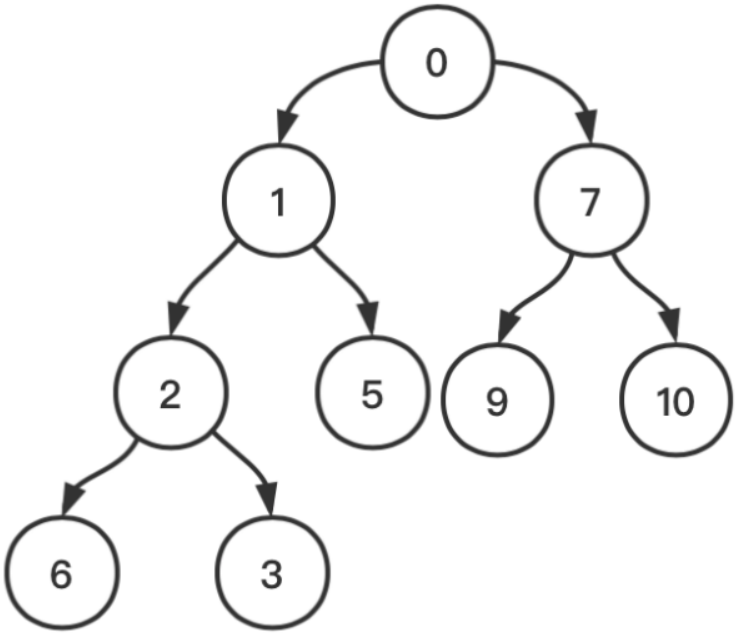
完全二叉树：若二叉树的深度为 h，去除了 h 层的节点，就是一个满二叉树；且 h 层都集中在最左侧排列；

最小堆：

- 是一颗完全二叉树；

> - 某一个节点的值总是小于等于它的子节点的值；
> - 堆中任意一个节点的子树都是最小堆；

0	1	7	2	5	9	10	6	3



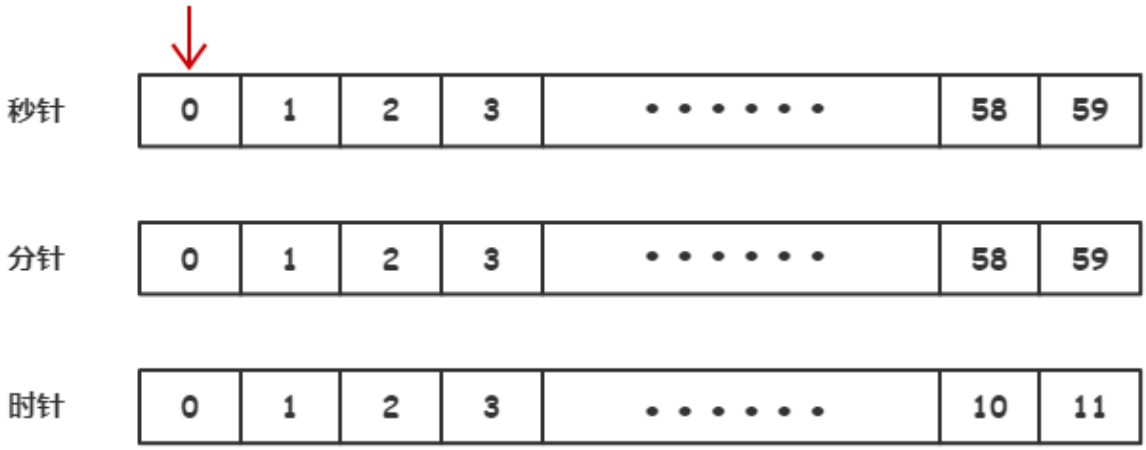
最小堆添加节点

> 为了满足完全二叉树的定义，往二叉树最高层沿着最左侧添加一个节点；然后考虑是否能**上升**操作；如果此时添加值为 4 的节点，4 节点是 5 节点的左子树；4 比 5 小，4 和 5 需要交换值；

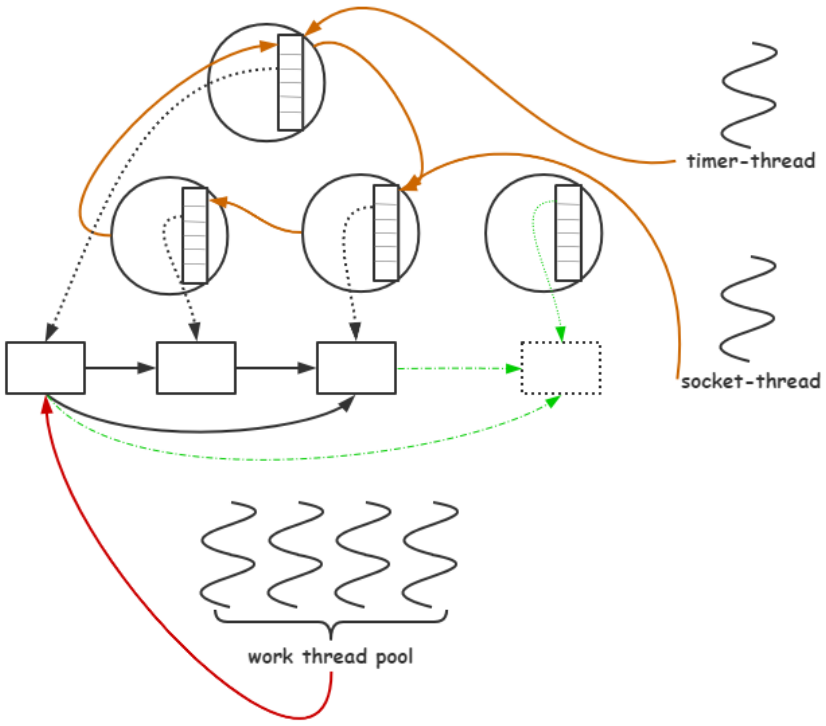
最小堆删除节点

> 删除操作需要先查找是否包含这个节点；确定存在后，交换最后一个节点，先考虑能否执行下降操作，否则执行上升操作；最后删除最后一个节点；
>
> 例如：删除 1 号节点，则需要下沉操作；删除 9 号节点，则需要上升操作；

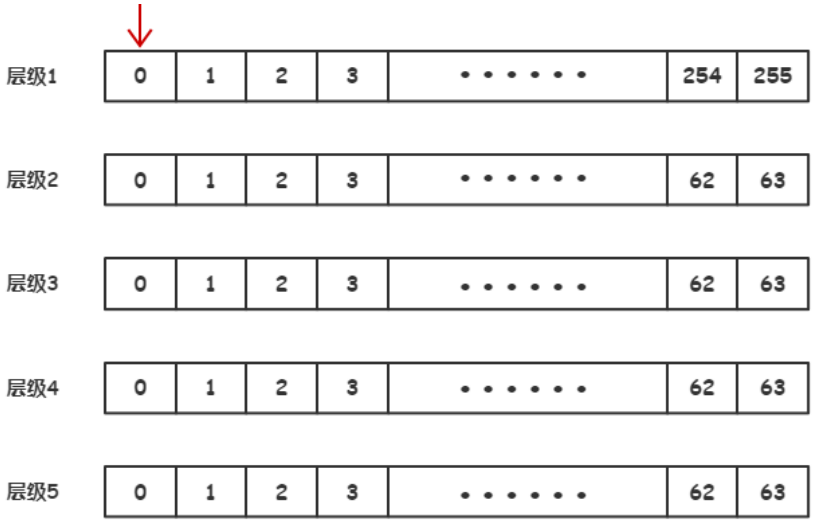
时间轮


秒针
0
1
2
3
58
59
分针
0
1
2
3
58
59
时针
0
1
2
3
10
11

运行图


timer-thread
socket-thread
work thread pool

原理图


层级1
0
1
2
3
254
255
层级2
0
1
2
3
62
63
层级3
0
1
2
3
62
63
层级4
0
1
2
3
62
63
层级5
0
1
2
3
62
63