



Join GitHub today

Dismiss

GitHub is home to over 36 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Tree: 62e84b27e7 ▾

[liuyanjie.github.io](#) / [source](#) / [_posts](#) / 3-libuv源码分析（四）定时器（Timer）.md[Find file](#)[Copy path](#)

liuyanjie add articles

2c06277 on Apr 24

[1 contributor](#)

78 lines (57 sloc) | 3.26 KB

[Raw](#)[Blame](#)[History](#)

title

date

updated

tags

categories

title	date	updated	tags	categories
libuv源码分析（四） 定时器（Timer）	2019-04-23 15:00:04 UTC	2019-04-24 01:31:28 UTC	libuv node.js eventloop	源码分析

在事件循环中，处理的一个 `handle` 就是计时器，通过 `uv__run_timers`，我们可以找到 `timer.c` 文件，里面包含了 `timer` 的实现。

先来看一下 `uv__run_timers` 的实现源码：

<https://github.com/libuv/libuv/blob/v1.28.0/src/timer.c#L158>

```
void uv__run_timers(uv_loop_t* loop) {
    struct heap_node* heap_node;
    uv_timer_t* handle;

    for (;;) {
        heap_node = heap_min(timer_heap(loop));
        if (heap_node == NULL)
            break;

        handle = container_of(heap_node, uv_timer_t, heap_node);
        if (handle->timeout > loop->time)
            break;

        uv_timer_stop(handle);
        uv_timer_again(handle);
        handle->timer_cb(handle);
    }
}
```

在 libuv 中，timer是按超时时间 `timeout` 存放在最小堆中的，这样，最小的的堆顶就是 `timeout` 最小的那个 timer，也就是最先到达超时时间的那个定时任务。

所以，在检查到期的定时任务时，只需要不断的获取堆顶的元素，并与当前时间比对：

1. 如果没有堆顶元素，则没有任何定时器存在，函数将直接返回。
2. 如果当前时间小于定时任务的超时间，那么堆顶timer未到超时时间，非堆顶的timer更没有达到超时时间，整个uv__run_timers也就会退出。
3. 如果当前时间等于或大于定时任务的超时间，这个timer就是一定达到或超过执行时间的。这时，就可以从timer堆中将其取出，然后调用其回调函数 `handle->timer_cb(handle)` 处理定时任务，然后再次重复获取下一个出现在堆顶的timer，直到情况1或2成立。

以下有两个主要注意的点：

1. 大于或等于实际上包含两种情形，这两种情形对于实际应用程序逻辑来说可能会出现天壤之别。
 - i. 如果当前时间等于定时任务的超时间，就是最理想的状态了，因为定时任务会在定时器到来的时候准时被执行，与预期相符合。
 - ii. 如果当前时间大于定时任务的超时间，则是非理想的状态了，然而这种情形缺是最出现的，因为很难保证当timer的超时时间到来时，程序搞好执行到此。
2. 如果定时任务的回调函数 `handle->timer_cb` 执行时间过长，将会导致整个循环阻塞在此处，从而影响其他定时器的处理，进而也影响到整个时间循环的其他逻辑的处理，因为只有一个线程在处理各类型的回调任务。

Methods

```
int uv_timer_init(uv_loop_t* loop, uv_timer_t* handle)
int uv_timer_start(uv_timer_t* handle, uv_timer_cb cb, uint64_t timeout, uint64_t repeat)
int uv_timer_stop(uv_timer_t* handle)
int uv_timer_again(uv_timer_t* handle)
```

```
void uv_timer_set_repeat(uv_timer_t* handle, uint64_t repeat)
uint64_t uv_timer_get_repeat(const uv_timer_t* handle)
```

uv_timer_t void (uv_timer_cb)(uv_timer_t handle)

```
int uv_timer_init(uv_loop_t* loop, uv_timer_t* handle) {
    uv__handle_init(loop, (uv_handle_t*)handle, UV_TIMER);
    handle->timer_cb = NULL;
    handle->repeat = 0;
    return 0;
}
```

源文件地址：<https://github.com/liuyanjie/knowledge/tree/master/node.js/libuv/4-libuv-timer.md>