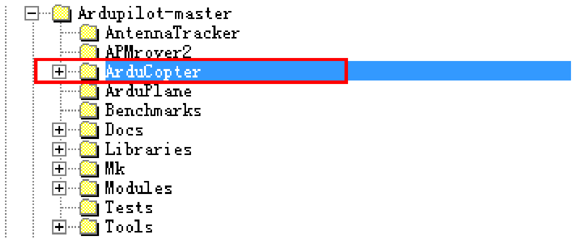
# ardupilot源码分析——任务调度

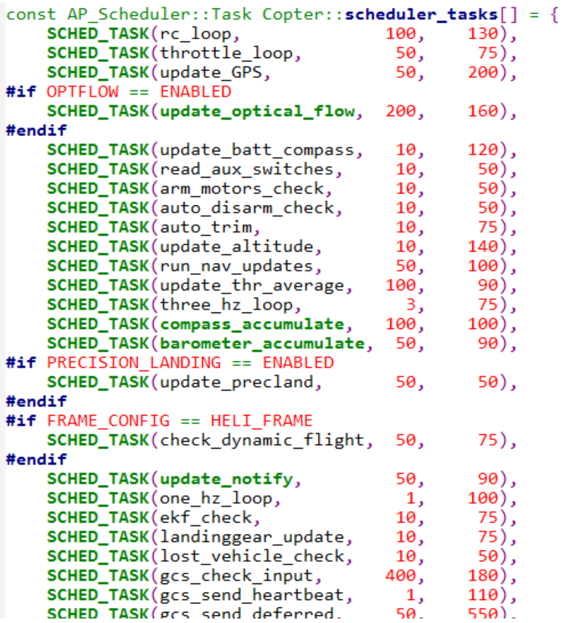
## 简述：

ardupilot的源码当中，使用了函数指针来实现多任务调度，每一个任务都有**函数名称，运行频率，最大运行时间**三个属性。除了主程序运行了一个fast_loop();之外，其他的任务都放在一个任务的数组里面，进行轮询切换。

## 代码分析：

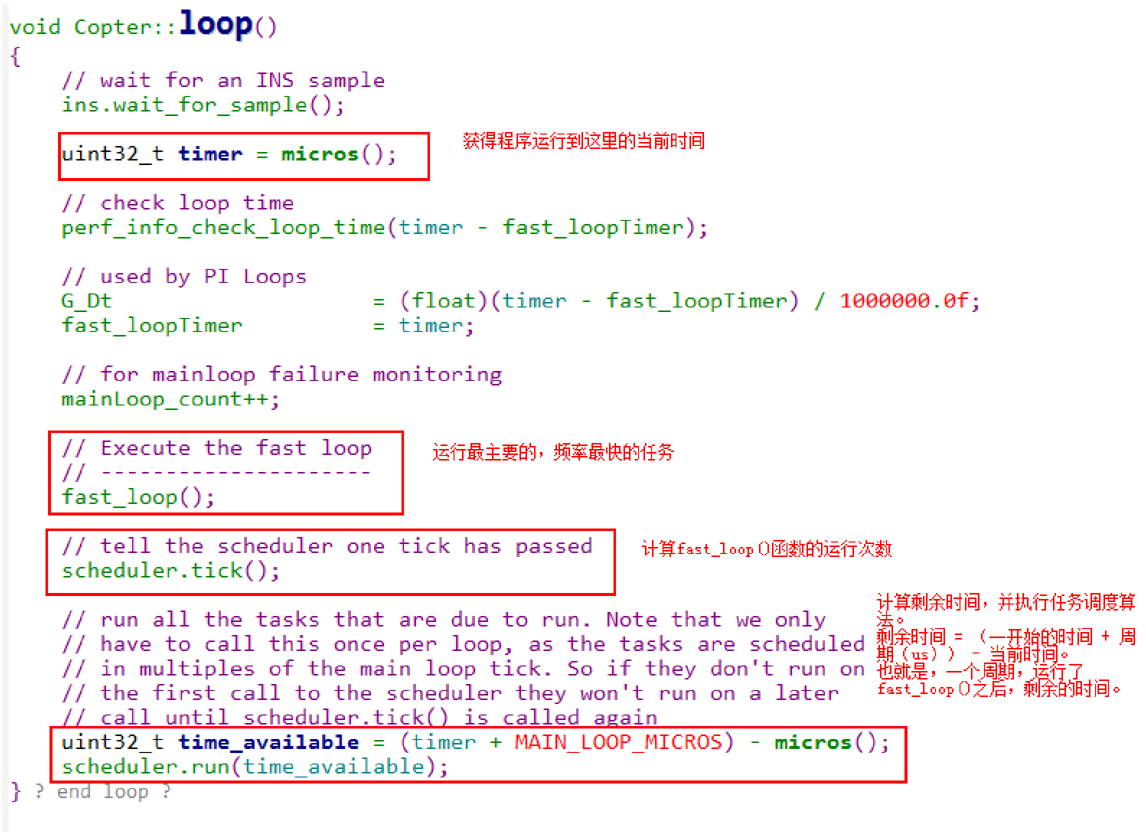


- **选择arducopter文件的ArduCopter.cpp代码。**



```
const AP_Scheduler::Task Copter::scheduler_tasks[] = {
    SCHED_TASK(rc_loop,              100,    130),
    SCHED_TASK(throttle_loop,         50,     75),
    SCHED_TASK(update_GPS,            50,    200),
#if OPTFLOW == ENABLED
    SCHED_TASK(update_optical_flow,  200,    160),
#endif
    SCHED_TASK(update_batt_compass,   10,    120),
    SCHED_TASK(read_aux_switches,     10,     50),
    SCHED_TASK(arm_motors_check,      10,     50),
    SCHED_TASK(auto_disarm_check,     10,     50),
    SCHED_TASK(auto_trim,             10,     75),
    SCHED_TASK(update_altitude,       10,    140),
    SCHED_TASK(run_nav_updates,       50,    100),
    SCHED_TASK(update_thr_average,   100,     90),
    SCHED_TASK(three_hz_loop,          3,     75),
    SCHED_TASK(compass_accumulate,   100,    100),
    SCHED_TASK(barometer_accumulate,  50,     90),
#if PRECISION_LANDING == ENABLED
    SCHED_TASK(update_precland,       50,     50),
#endif
#if FRAME_CONFIG == HELI_FRAME
    SCHED_TASK(check_dynamic_flight,  50,     75),
#endif
    SCHED_TASK(update_notify,         50,     90),
    SCHED_TASK(one_hz_loop,            1,    100),
    SCHED_TASK(ekf_check,             10,     75),
    SCHED_TASK(landinggear_update,    10,     75),
    SCHED_TASK(lost_vehicle_check,    10,     50),
    SCHED_TASK(gcs_check_input,      400,    180),
    SCHED_TASK(gcs_send_heartbeat,     1,    110),
    SCHED_TASK(gcs_send_deferred,     50,    550),
```

- **在这个数组里面，装着不同的任务（任务名称，运行频率（**Hz**），最大允许运行的时间）**



```
void Copter::loop()
{
    // wait for an INS sample
    ins.wait_for_sample();

    uint32_t timer = micros();

    // check loop time
    perf_info_check_loop_time(timer - fast_loopTimer);

    // used by PI Loops
    G_Dt                        = (float)(timer - fast_loopTimer) / 1000000.0f;
    fast_loopTimer              = timer;

    // for mainloop failure monitoring
    mainLoop_count++;

    // Execute the fast loop
    // ---------------------
    fast_loop();

    // tell the scheduler one tick has passed
    scheduler.tick();

    // run all the tasks that are due to run. Note that we only
    // have to call this once per loop, as the tasks are scheduled
    // in multiples of the main loop tick. So if they don't run on
    // the first call to the scheduler they won't run on a later
    // call until scheduler.tick() is called again
    uint32_t time_available = (timer + MAIN_LOOP_MICROS) - micros();
    scheduler.run(time_available);
} ? end loop ?
```

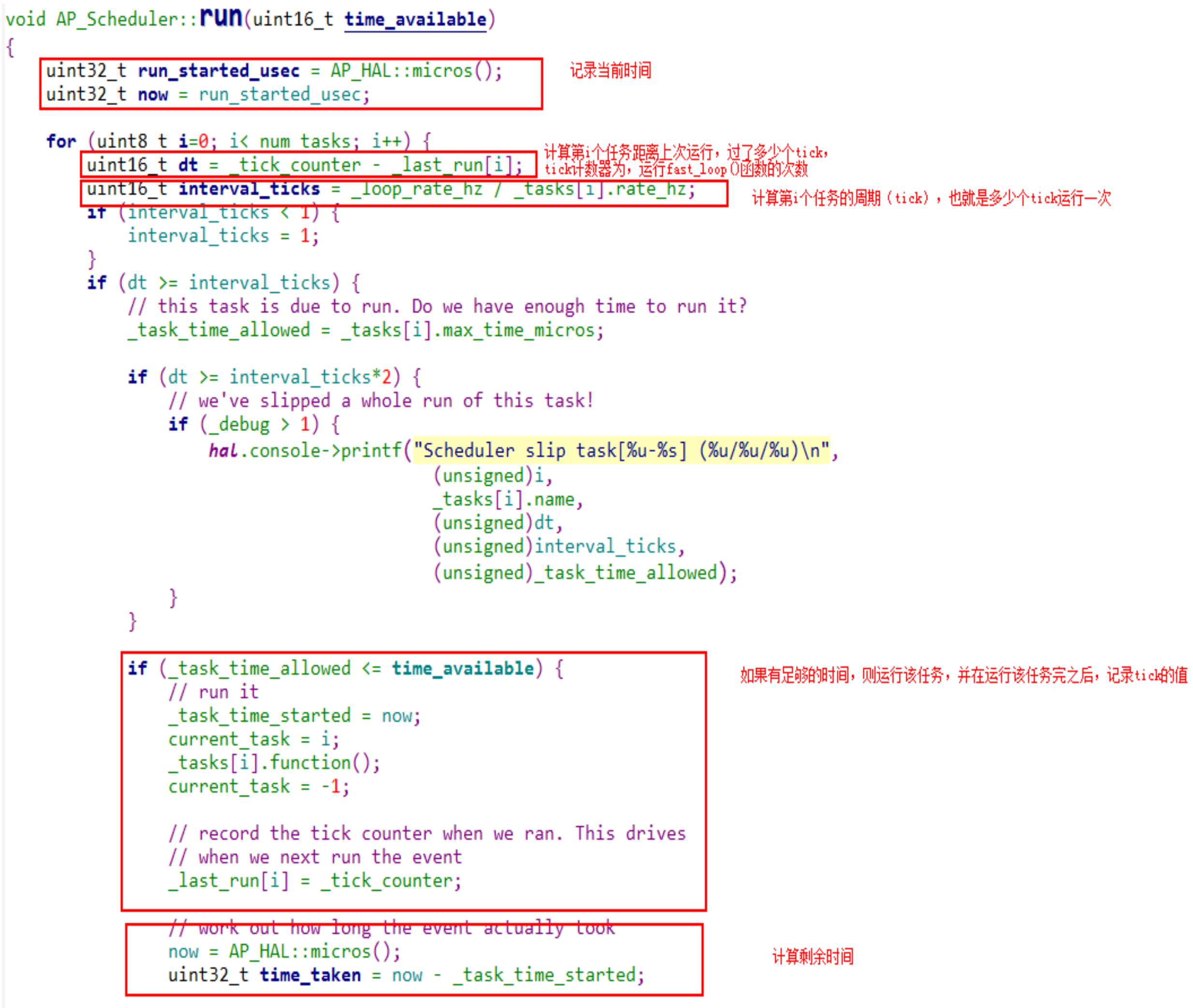
获得程序运行到这里的当前时间

运行最主要的，频率最快的任务

计算fast_loop()函数的运行次数

计算剩余时间，并执行任务调度算法。
剩余时间 = （一开始的时间 + 周期（us））- 当前时间。
也就是，一个周期，运行了fast_loop()之后，剩余的时间。

- **在**loop();**函数里面，运行了一个函数，**fast_loop();**，并计算剩余时间，传递给调度算法执行任务调度。在**run()**函数里面完成任务调度**



```
void AP_Scheduler::run(uint16_t time_available)
{
    uint32_t run_started_usec = AP_HAL::micros();
    uint32_t now = run_started_usec;

    for (uint8_t i=0; i< num_tasks; i++) {
        uint16_t dt = _tick_counter - _last_run[i];
        uint16_t interval_ticks = _loop_rate_hz / _tasks[i].rate_hz;
        if (interval_ticks < 1) {
            interval_ticks = 1;
        }
        if (dt >= interval_ticks) {
            // this task is due to run. Do we have enough time to run it?
            _task_time_allowed = _tasks[i].max_time_micros;

            if (dt >= interval_ticks*2) {
                // we've slipped a whole run of this task!
                if (_debug > 1) {
                    hal.console->printf("Scheduler slip task[%u-%s] (%u/%u/%u)\n",
                                        (unsigned)i,
                                        _tasks[i].name,
                                        (unsigned)dt,
                                        (unsigned)interval_ticks,
                                        (unsigned)_task_time_allowed);
                }
            }

            if (_task_time_allowed <= time_available) {
                // run it
                _task_time_started = now;
                current_task = i;
                _tasks[i].function();
                current_task = -1;

                // record the tick counter when we ran. This drives
                // when we next run the event
                _last_run[i] = _tick_counter;

                // work out how long the event actually took
                now = AP_HAL::micros();
                uint32_t time_taken = now - _task_time_started;
```

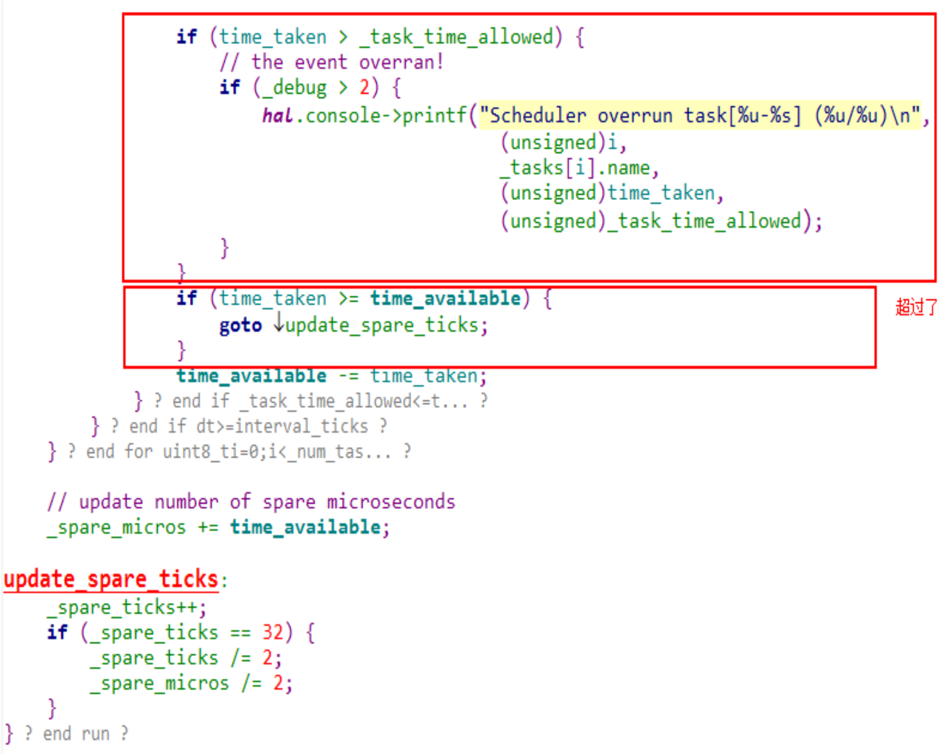
记录当前时间

计算第i个任务距离上次运行，过了多少个tick，
tick计数器为，运行fast_loop()函数的次数

计算第i个任务的周期（tick），也就是多少个tick运行一次

如果有足够的时间，则运行该任务，并在运行该任务完之后，记录tick的值

计算剩余时间



```
                if (time_taken > _task_time_allowed) {
                    // the event overran!
                    if (_debug > 2) {
                        hal.console->printf("Scheduler overrun task[%u-%s] (%u/%u)\n",
                                              (unsigned)i,
                                              _tasks[i].name,
                                              (unsigned)time_taken,
                                              (unsigned)_task_time_allowed);
                    }
                }
                if (time_taken >= time_available) {
                    goto ↓update_spare_ticks;
                }
                time_available -= time_taken;
            } ? end if _task_time_allowed<=t... ?
        } ? end if dt>=interval_ticks ?
    } ? end for uint8_ti=0;i<_num_tas... ?

    // update number of spare microseconds
    _spare_micros += time_available;

update_spare_ticks:
    _spare_ticks++;
    if (_spare_ticks == 32) {
        _spare_ticks /= 2;
        _spare_micros /= 2;
    }
} ? end run ?
```

如果任务运行时间，大于最大允许的时间，则打印该任务的相关信息

超过了剩余的时间，则立刻跳出调度函数

- **任务调度函数里面的逻辑如上图。**