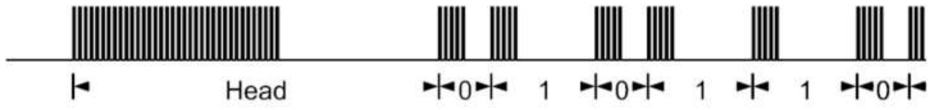
智能系统与控制

树莓派：红外接收

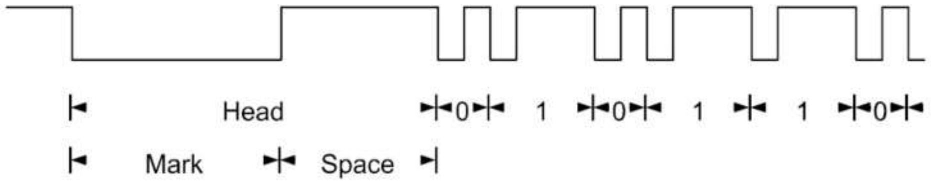
于泓
鲁东大学
信息与电气工程学院
2021.10.19

红外收发的基本原理

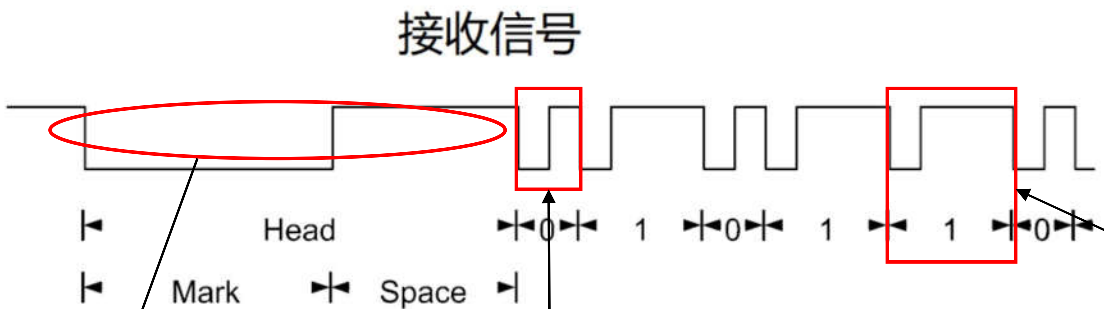
红外遥控器的顶端有一个红外发射头，当有按键按下的时候遥控器内部的编码器就会发射一串这个按键所特有的二进制的 0,1 编码，这些编码经过 38k 载波的调制后通过红外发射头发射出去，



接收信号



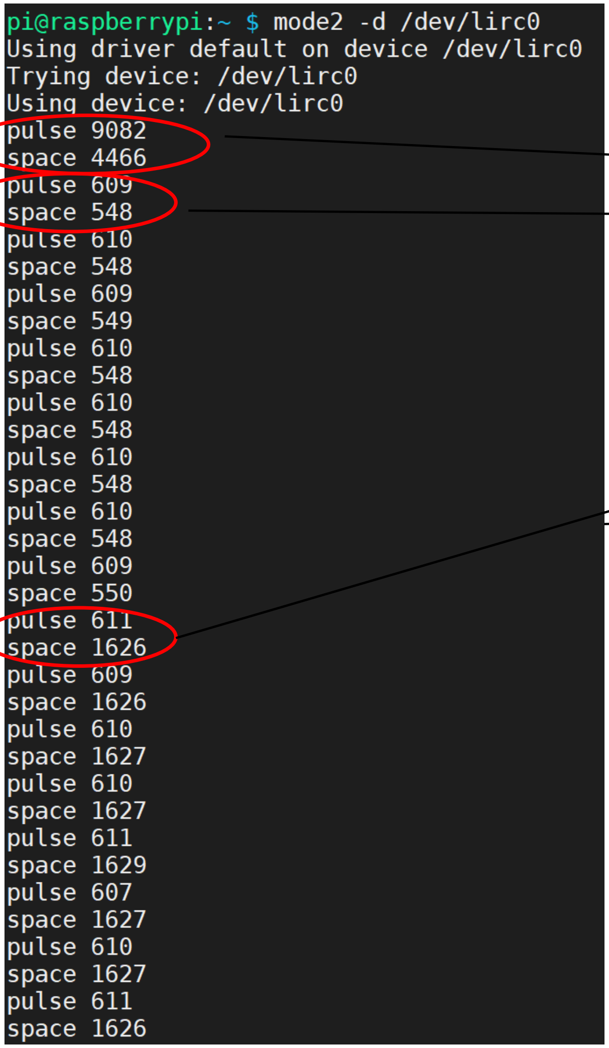
红外接模块的主要部件是一个**黑色的红外接收头**。它是一个 IC 化红外线受光元件。内部电路包括红外监测二极管，放大器，限幅器，带通滤波器，积分电路，比较器等，通过它可以将接受的红外信号滤波整形，提取包络，最终在它的信号输出引脚 DO 处



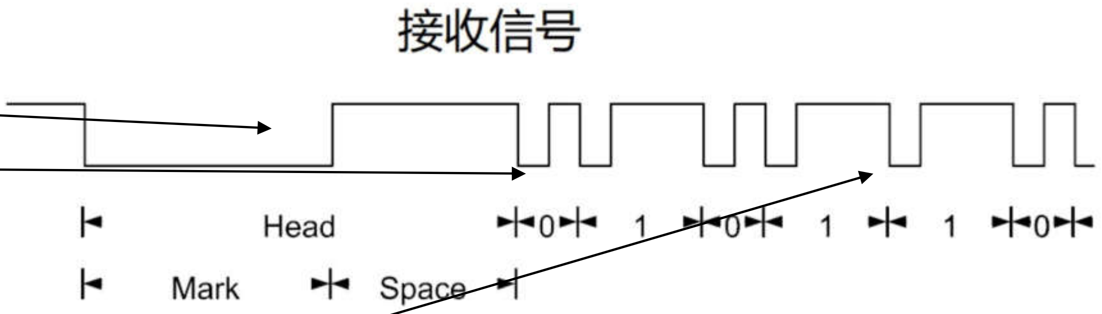
1 信号，
低电平持续时间 560us，高电平持续时间大约为 1680us。

0信号，
低电平与高电平持续时间相同大约为 560us

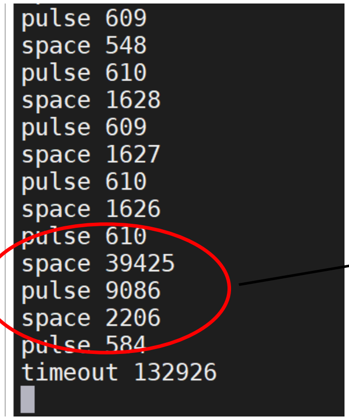
头部信号
先低后高



```
pi@raspberrypi:~ $ mode2 -d /dev/lirc0
Using driver default on device /dev/lirc0
Trying device: /dev/lirc0
Using device: /dev/lirc0
pulse 9082
space 4466
pulse 609
space 548
pulse 610
space 548
pulse 609
space 549
pulse 610
space 548
pulse 610
space 548
pulse 610
space 548
pulse 610
space 548
pulse 609
space 550
pulse 611
space 1626
pulse 609
space 1626
pulse 610
space 1627
pulse 610
space 1627
pulse 611
space 1629
pulse 607
space 1627
pulse 610
space 1627
pulse 611
space 1626
```



每次按键发送32位码



```
pulse 609
space 548
pulse 610
space 1628
pulse 609
space 1627
pulse 610
space 1626
pulse 610
space 39425
pulse 9086
space 2206
pulse 584
timeout 132926
```

结束

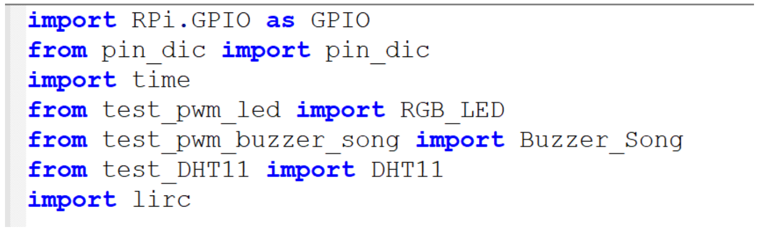
代码实现

RGB_LED： R -- G18
G -- G19
B -- G20

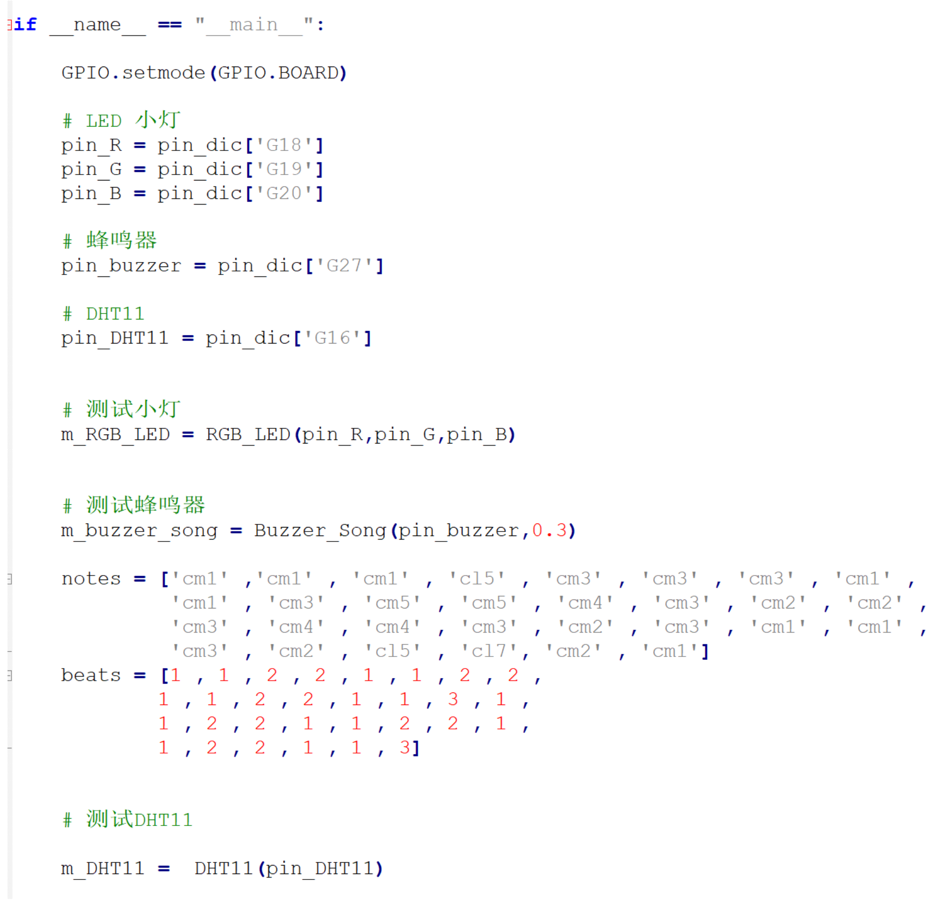
IR ： D0 -- G17

Buzzer： G27

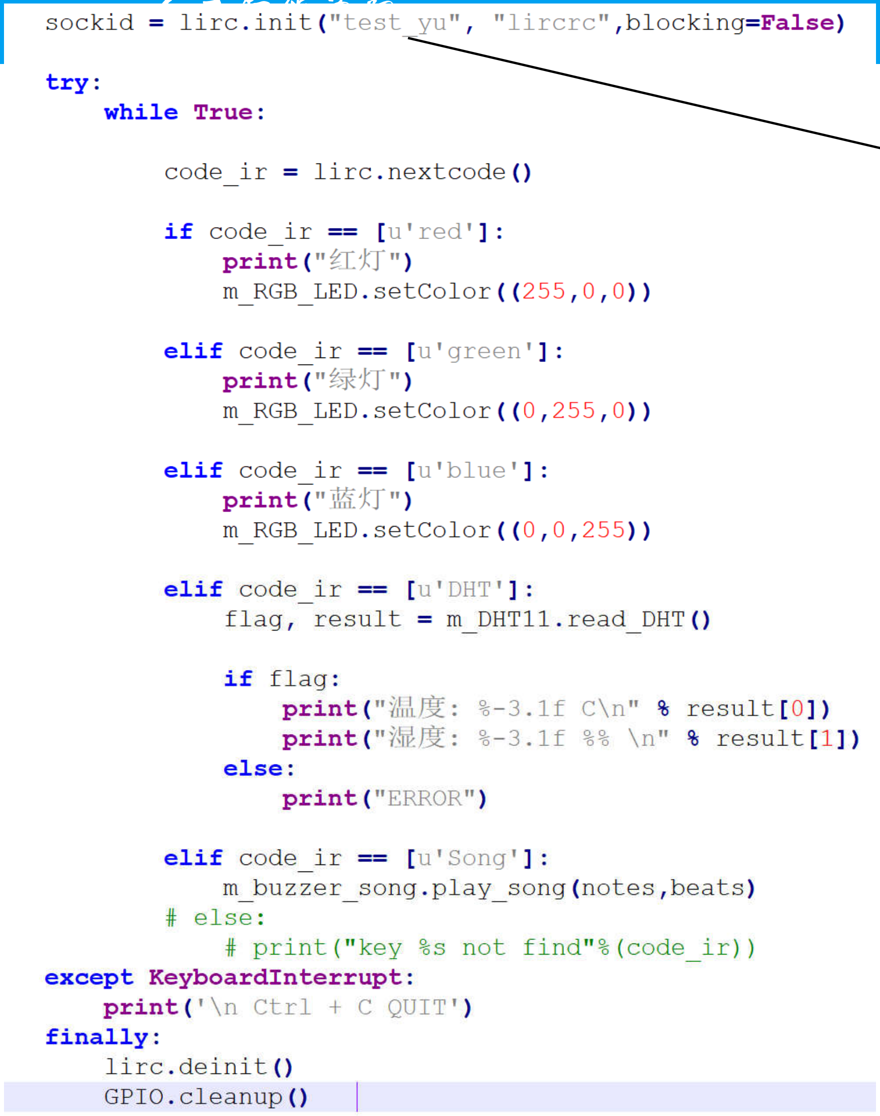
DHT11： G16



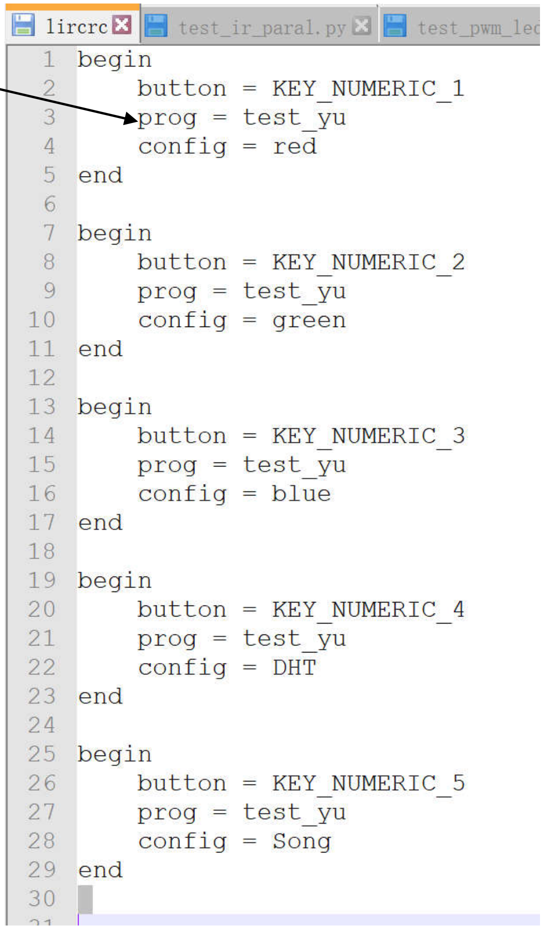
```
import RPi.GPIO as GPIO
from pin_dic import pin_dic
import time
from test_pwm_led import RGB_LED
from test_pwm_buzzer_song import Buzzer_Song
from test_DHT11 import DHT11
import lirc
```



```
if __name__ == "__main__":

    GPIO.setmode(GPIO.BOARD)

    # LED 小灯
    pin_R = pin_dic['G18']
    pin_G = pin_dic['G19']
    pin_B = pin_dic['G20']

    # 蜂鸣器
    pin_buzzer = pin_dic['G27']

    # DHT11
    pin_DHT11 = pin_dic['G16']


    # 测试小灯
    m_RGB_LED = RGB_LED(pin_R,pin_G,pin_B)


    # 测试蜂鸣器
    m_buzzer_song = Buzzer_Song(pin_buzzer,0.3)

    notes = ['cm1' ,'cm1' , 'cm1' , 'cl5' , 'cm3' , 'cm3' , 'cm3' , 'cm1' ,
             'cm1' , 'cm3' , 'cm5' , 'cm5' , 'cm4' , 'cm3' , 'cm2' , 'cm2' ,
             'cm3' , 'cm4' , 'cm4' , 'cm3' , 'cm2' , 'cm3' , 'cm1' , 'cm1' ,
             'cm3' , 'cm2' , 'cl5' , 'cl7', 'cm2' , 'cm1']
    beats = [1 , 1 , 2 , 2 , 1 , 1 , 2 , 2 ,
            1 , 1 , 2 , 2 , 1 , 1 , 3 , 1 ,
            1 , 2 , 2 , 1 , 1 , 2 , 2 , 1 ,
            1 , 2 , 2 , 1 , 1 , 3]


    # 测试DHT11

    m_DHT11 =  DHT11(pin_DHT11)
```



```python
sockid = lirc.init("test_yu", "lircrc",blocking=False)

try:
    while True:

        code_ir = lirc.nextcode()

        if code_ir == [u'red']:
            print("红灯")
            m_RGB_LED.setColor((255,0,0))

        elif code_ir == [u'green']:
            print("绿灯")
            m_RGB_LED.setColor((0,255,0))

        elif code_ir == [u'blue']:
            print("蓝灯")
            m_RGB_LED.setColor((0,0,255))

        elif code_ir == [u'DHT']:
            flag, result = m_DHT11.read_DHT()

            if flag:
                print("温度: %-3.1f C\n" % result[0])
                print("湿度: %-3.1f %% \n" % result[1])
            else:
                print("ERROR")

        elif code_ir == [u'Song']:
            m_buzzer_song.play_song(notes,beats)
        # else:
            # print("key %s not find"%(code_ir))
except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')
finally:
    lirc.deinit()
    GPIO.cleanup()
```



```
begin
    button = KEY_NUMERIC_1
    prog = test_yu
    config = red
end

begin
    button = KEY_NUMERIC_2
    prog = test_yu
    config = green
end

begin
    button = KEY_NUMERIC_3
    prog = test_yu
    config = blue
end

begin
    button = KEY_NUMERIC_4
    prog = test_yu
    config = DHT
end

begin
    button = KEY_NUMERIC_5
    prog = test_yu
    config = Song
end
```

红外按键+多任务处理

任务1： 利用RGB-LED，循环显示不同的颜色

按键1　开始/停止

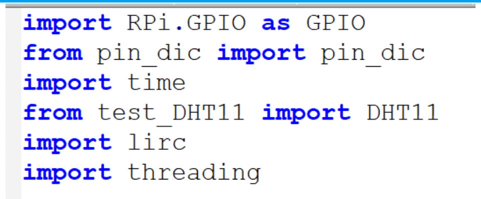
按键2　暂停/回复

任务2： 利用蜂鸣器，循环演奏音乐

按键3　开始/停止

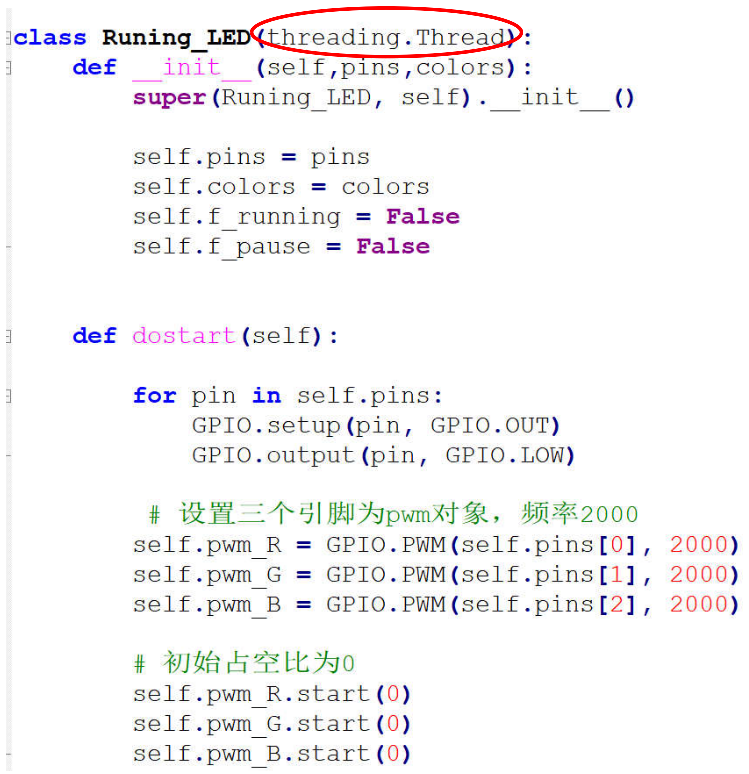
按键4　暂停/回复

任务3： 利用DHT11，显示温度和湿度



```python
import RPi.GPIO as GPIO
from pin_dic import pin_dic
import time
from test_DHT11 import DHT11
import lirc
import threading
```

构造一个线程对象



```python
class Runing_LED(threading.Thread):
    def __init__(self,pins,colors):
        super(Runing_LED, self).__init__()

        self.pins = pins
        self.colors = colors
        self.f_running = False
        self.f_pause = False


    def dostart(self):

        for pin in self.pins:
            GPIO.setup(pin, GPIO.OUT)
            GPIO.output(pin, GPIO.LOW)

         # 设置三个引脚为pwm对象，频率2000
        self.pwm_R = GPIO.PWM(self.pins[0], 2000)
        self.pwm_G = GPIO.PWM(self.pins[1], 2000)
        self.pwm_B = GPIO.PWM(self.pins[2], 2000)

        # 初始占空比为0
        self.pwm_R.start(0)
        self.pwm_G.start(0)
        self.pwm_B.start(0)
```

```
def dostop(self):
    self.pwm_R.stop()
    self.pwm_G.stop()
    self.pwm_B.stop()
    for pin in self.pins:
        GPIO.output(pin, GPIO.HIGH)

def color2ratio(self,x,min_color,max_color,min_ratio,max_ratio):
    return (x - min_color) * (max_ratio - min_ratio) / (max_color - min_color) + min_ratio

def setColor(self,col):
    R_val,G_val,B_val = col

    R =self.color2ratio(R_val, 0, 255, 0, 100)
    G =self.color2ratio(G_val, 0, 255, 0, 100)
    B =self.color2ratio(B_val, 0, 255, 0, 100)

    # 改变占空比
    self.pwm_R.ChangeDutyCycle(R)
    self.pwm_G.ChangeDutyCycle(G)
    self.pwm_B.ChangeDutyCycle(B)


 def stop(self):
     self.f_running = False

 def pause(self):
     self.f_pause = True

 def resume(self):
     self.f_pause = False
```

```
def run(self):

    self.dostart()
    self.f_running = True

    while True:
        for col in self.colors:

            if not self.f_running:
                self.dostop()
                self.f_running = False
                return

            while self.f_pause:
                if not self.f_running:
                    self.dostop()
                    return
                else:
                    pass

            # 设置颜色
            self.setColor(col)
            # 延时
            time.sleep(1)
```

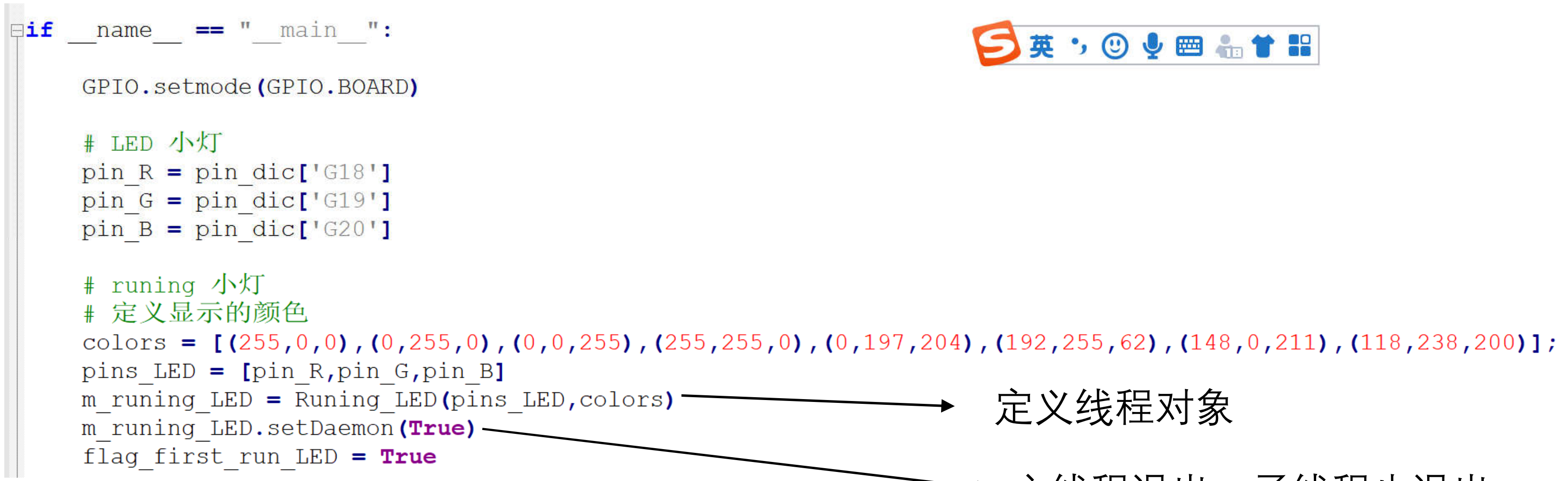
线程对象中必须写入的函数用来负责线程启动后执行的程序

判断是否结束

判断是否暂停

```python
class Runing_Song(threading.Thread):
    def __init__(self,pin,notes,beats):
        super(Runing_Song, self).__init__()

        # 设 置 蜂 鸣 器 引 脚 模 式
        self.pin_buzzer = pin
        GPIO.setup(self.pin_buzzer,GPIO.OUT)

        self.note2freq = {"cl1":131,"cl2":147 ,'cl3':165 ,"cl4":175 ,"cl5":196 ,"cl6":211 ,"cl7":248,
                          "cm1":262,"cm2":294 ,'cm3':330 ,"cm4":350 ,"cm5":393 ,"cm6":441 ,"cm7":495,
                          "ch1":525,"ch2":589 ,'ch3':661 ,"ch4":700 ,"ch5":786 ,"ch6":882 ,"ch7":990
                          }
        self.delay_beat = 0.3

        self.f_running = False
        self.f_pause = False


    def dostart(self):
        # 创 建PWM对 象 初 始 频 率 440 占 空 比 50%
        self.Buzzer = GPIO.PWM( pin_buzzer , 440)
        self.Buzzer.start(50)

    def dostop(self):
        self.Buzzer.stop()
        GPIO.output(self.pin_buzzer, GPIO.LOW)

    def stop(self):
        self.f_running = False

    def pause(self):
        self.f_pause = True

    def resume(self):
        self.f_pause = False
```

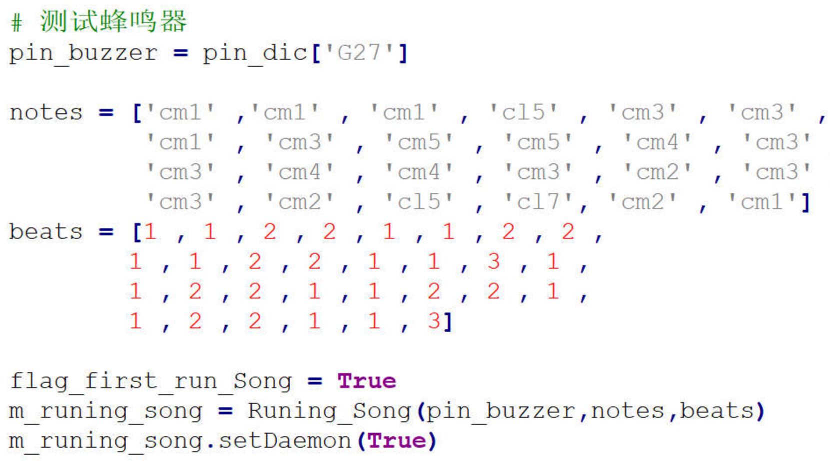
```python
    def run(self):
        self.dostart()
        self.f_running = True

        while True:
            for note,beat in zip(notes,beats):

                if not self.f_running:
                    self.dostop()
                    self.f_running = False
                    return

                while self.f_pause:
                    if not self.f_running:
                        self.dostop()
                        return
                    else:
                        self.Buzzer.ChangeFrequency(0.1)
                        time.sleep(0.1)

                self.Buzzer.ChangeFrequency(self.note2freq[note])
                time.sleep(self.delay_beat*beat)
```



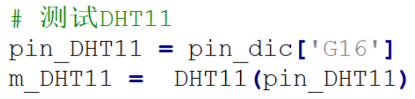
```python
if __name__ == "__main__":

    GPIO.setmode(GPIO.BOARD)

    # LED 小灯
    pin_R = pin_dic['G18']
    pin_G = pin_dic['G19']
    pin_B = pin_dic['G20']

    # runing 小灯
    # 定义显示的颜色
    colors = [(255,0,0),(0,255,0),(0,0,255),(255,255,0),(0,197,204),(192,255,62),(148,0,211),(118,238,200)];
    pins_LED = [pin_R,pin_G,pin_B]
    m_runing_LED = Runing_LED(pins_LED,colors)
    m_runing_LED.setDaemon(True)
    flag_first_run_LED = True
```

定义线程对象

主线程退出，子线程也退出

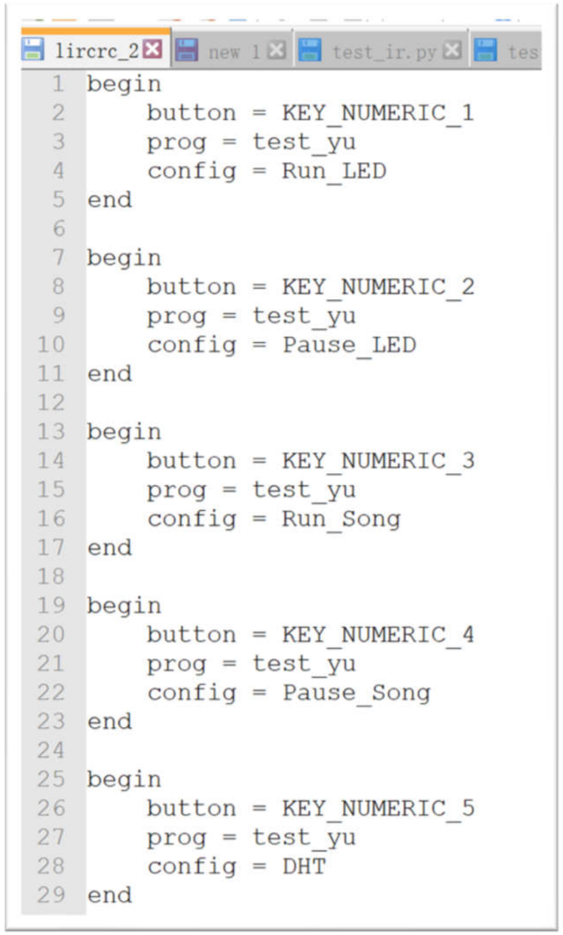


```python
    # 测试蜂鸣器
    pin_buzzer = pin_dic['G27']

    notes = ['cm1' ,'cm1' , 'cm1' , 'cl5' , 'cm3' , 'cm3' , 'cm3' , 'cm1' ,
             'cm1' , 'cm3' , 'cm5' , 'cm5' , 'cm4' , 'cm3' , 'cm2' , 'cm2' ,
             'cm3' , 'cm4' , 'cm4' , 'cm3' , 'cm2' , 'cm3' , 'cm1' , 'cm1' ,
             'cm3' , 'cm2' , 'cl5' , 'cl7', 'cm2' , 'cm1']
    beats = [1 , 1 , 2 , 2 , 1 , 1 , 2 , 2 ,
             1 , 1 , 2 , 2 , 1 , 1 , 3 , 1 ,
             1 , 2 , 2 , 1 , 1 , 2 , 2 , 1 ,
             1 , 2 , 2 , 1 , 1 , 3]

    flag_first_run_Song = True
    m_runing_song = Runing_Song(pin_buzzer,notes,beats)
    m_runing_song.setDaemon(True)
```



```python
# 测试DHT11
pin_DHT11 = pin_dic['G16']
m_DHT11 =  DHT11(pin_DHT11)
```

```
# 红外接收器初始化
sockid = lirc.init("test_yu", "lircrc_2",blocking=False)

try:
    while True:

        code_ir = lirc.nextcode()

        if code_ir == [u'DHT']:
            flag, result = m_DHT11.read_DHT()

            if flag:
                print("温度: %-3.1f C\n" % result[0])
                print("湿度: %-3.1f %% \n" % result[1])
            else:
                print("ERROR")
```



```
begin
    button = KEY_NUMERIC_1
    prog = test_yu
    config = Run_LED
end

begin
    button = KEY_NUMERIC_2
    prog = test_yu
    config = Pause_LED
end

begin
    button = KEY_NUMERIC_3
    prog = test_yu
    config = Run_Song
end

begin
    button = KEY_NUMERIC_4
    prog = test_yu
    config = Pause_Song
end

begin
    button = KEY_NUMERIC_5
    prog = test_yu
    config = DHT
end
```

```
elif code_ir == [u'Run_LED']:

    if  m_runing_LED.f_running:
        print("stop runing LED")
        m_runing_LED.stop()

    else:
        print("start runing LED")

        if flag_first_run_LED:
            m_runing_LED.start()
            flag_first_run_LED = False
        else:
            m_runing_LED = Runing_LED(pins_LED,colors)
            m_runing_LED.setDaemon(True)
            m_runing_LED.start()

elif code_ir == [u'Pause_LED']:

    if m_runing_LED.f_pause:
        print("Resume Runing LED")
        m_runing_LED.resume()
    else:
        print("Pause Runing LED")
        m_runing_LED.pause()
```

首次启动

再次启动重定义线程

```
elif code_ir == [u'Run_Song']:

    if  m_runing_song.f_running:
        print("stop Song")
        m_runing_song.stop()

    else:
        print("start Song")
        if flag_first_run_Song:
            m_runing_song.start()
            flag_first_run_Song = False
        else:
            m_runing_song = Runing_Song(pin_buzzer,notes,beats)
            m_runing_song.setDaemon(True)
            m_runing_song.start()

elif code_ir == [u'Pause_Song']:

    if m_runing_song.f_pause:
        print("Resume Song")
        m_runing_song.resume()
    else:
        print("Pause Song")
        m_runing_song.pause()


except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')
finally:
    lirc.deinit()
    GPIO.cleanup()
```