# 树莓派 raspberry pi GPIO python

## 1.RPi.GPIO模块基本使用

### 导入模块

导入 RPi.GPIO 模块:

```
import RPi.GPIO as GPIO
```

通过下面的代码可以检测导入是否成功

```
try:
    import RPi.GPIO as GPIO
except RuntimeError:
    print("Error importing RPi.GPIO!  This is probably because you need superuser privileges.  You can achieve this by using 'sudo' to run your script")
```

### Pin numbering引脚号

RPi.GPIO中有两种引脚的编号方式。第一个是使用电路板编号系统。这指的是在树莓派电路板上P1开头的。使用这个编号系统的优势是,您的硬件将总是工作,不管电路板是哪个版本的。你不需要重新修改代码。

第二个编号系统是BCM数字。这是一个低水平的工作方式,它指的是电路板引脚的位置号。电路板版本变化时脚本程序需要对应修改。

必须指定使用哪种:

```
GPIO.setmode(GPIO.BOARD)
  # or
GPIO.setmode(GPIO.BCM)
```

可以查询使用的哪种编号方法

```
mode = GPIO.getmode()
```

输出: GPIO.BOARD, GPIO.BCM or None

### 设置一个通道

作为输入

```
GPIO.setup(channel, GPIO.IN)
```

(chanel与使用的编号方式对应).

作为输出:

```
GPIO.setup(channel, GPIO.OUT)
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```

### 设置多个通道

```
chan_list = [11,12]    # add as many channels as you want!
GPIO.setup(chan_list, GPIO.OUT)
```

### 输入

读取一个GPIO口的值:

```
GPIO.input(channel)
```

返回：0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

### 输出

设置一个GPIO口的输出值:

```
GPIO.output(channel, state)
```

（State 可以是 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

设置多个通道的输出

```
chan_list = [11,12]                                # also works
with tuples
GPIO.output(chan_list, GPIO.LOW)                   # sets all to
GPIO.LOW
GPIO.output(chan_list, (GPIO.HIGH, GPIO.LOW))    # sets first
HIGH and second LOW
```

### 清理

在任何程序结束时,是一种很好的做法清理任何资源

清理脚本的末尾:

```
GPIO.cleanup()
```

如果想清理特定的通道:

```
GPIO.cleanup(channel)
GPIO.cleanup( (channel1, channel2) )
GPIO.cleanup( [channel1, channel2] )
```

# 2.GPIO 输入

要得到GPIO输入到你的程序的几种方法。第一个和最简单的方法是在时间点上检查输入值。这被称为“轮询”，如果你的程序在错误的时间读取值，可能会错过一个输入。轮询是在循环中进行的。另外一种方法来响应一个GPIO输入是使用“中断”（边沿检测）。边沿是由高到低（下降沿）或低到高（上升沿）的过渡的名称。

## 上拉或者下拉电阻

如果您没有连接到任何硬件的输入引脚，它将'浮动'。换句话说，读取的值是未定义的，因为它没有连接到任何事情，直到你按一个按钮或开关。由于受电干扰，它的值可能会改变。

为了实现这一点，我们使用了上拉或下拉电阻。以这种方式，可以设置输入的默认值。在硬件和软件上有上拉/下拉电阻是可能的。在硬件上，在输入通道和3.3V（上拉）之间的或0V（下拉）使用10K电阻是常用的。通过rpi.gpio模块配置GPIO口可以实现同样的功能:

```
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_UP)
  # or
```

```
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

## 测试输入(轮询)

可以在一个时间点查看输入:

```
if GPIO.input(channel):
    print('Input was HIGH')
else:
    print('Input was LOW')
```

在循环中使用轮询来检测按钮是否按下:

```
while GPIO.input(channel) == GPIO.LOW:
    time.sleep(0.01)  # wait 10 ms to give CPU chance to do other things
```

## 中断与边沿检测

边沿是电信号从低到高（上升沿）或从高到低（下降沿）的改变。这种改是一种事件，为了使程序在运行的过程中检测到按钮按下这样的事件：

- wait_for_edge()
- event_detected()
- 另外一个线程的回调方法

## wait_for_edge()

wait_for_edge()方法不会执行，只到检测到边沿，检测按钮按下也可以写成:

```
GPIO.wait_for_edge(channel, GPIO.RISING)
```

这种检测边沿 GPIO.RISING, GPIO.FALLING or GPIO.BOTH的优势是使用CPU的资源很少

可以加一个timeout参数:

```
# wait for up to 5 seconds for a rising edge (timeout is in milliseconds)
channel = GPIO.wait_for_edge(channel, GPIO_RISING, timeout=5000)
if channel is None:
    print('Timeout occurred')
else:
    print('Edge detected on channel', channel)
```

## event_detected() function

event_detected()方法是用在循环事件中，在Pygame或PyQt这种存在一个主循环监听GUI时非常有用.

```
GPIO.add_event_detect(channel, GPIO.RISING)  # add rising edge detection on a channel
do_something()
if GPIO.event_detected(channel):
    print('Button pressed')
```

## 线程回调

RPi.GPIO运行第二个线程来处理回调函数:

```
def my_callback(channel):
    print('This is a edge event callback function!')
    print('Edge detected on channel %s'%channel)
    print('This is run in a different thread to your main
program')

GPIO.add_event_detect(channel, GPIO.RISING,
callback=my_callback)  # add rising edge detection on a
channel
...the rest of your program...
```

多个回调函数:

```
def my_callback_one(channel):
    print('Callback one')

def my_callback_two(channel):
    print('Callback two')

GPIO.add_event_detect(channel, GPIO.RISING)
GPIO.add_event_callback(channel, my_callback_one)
GPIO.add_event_callback(channel, my_callback_two)
```

这里的回调函数是按顺序运行的，不是同事运行的，因为只有一个进程（process）在运行

## 开关去抖

开关在按下的过程中回调函数会执行多次，这是因为开关按下的过程中由于抖动产生多个边沿的原因，下面的方法可以解决这个问题:

- 增加一个0.1uF的电容与开关串接
- 软件去抖
- 结合上述两种方法

在程序中增加一个bouncetime参数可以现实去抖:

```
# add rising edge detection on a channel, ignoring further
edges for 200ms for switch bounce handling
GPIO.add_event_detect(channel, GPIO.RISING,
callback=my_callback, bouncetime=200)
```

or

```
GPIO.add_event_callback(channel, my_callback, bouncetime=200)
```

## 移除事件监听

不再需要监听时:

```
GPIO.remove_event_detect(channel)
```

# 3.GPIO 输出

## 设置RPi.GPIO

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)
```

## 设置为高电平输出:

```
GPIO.output(12, GPIO.HIGH)
 # or
GPIO.output(12, 1)
 # or
GPIO.output(12, True)
```

## 设置低电平输出:

```
GPIO.output(12, GPIO.LOW)
 # or
GPIO.output(12, 0)
 # or
GPIO.output(12, False)
```

## 同时设置多个通道的输出:

```
chan_list = (11,12)
GPIO.output(chan_list, GPIO.LOW) # all LOW
GPIO.output(chan_list, (GPIO.HIGH,GPIO.LOW))  # first LOW,
second HIGH
```

## 清空

```
GPIO.cleanup()
```

input()方法可以读取目前通道的输出:

```
GPIO.output(12, not GPIO.input(12))
```

# 4.PWM（脉冲宽度调制）

## 创建PWM 实例

```
p = GPIO.PWM(channel, frequency)
```

## 启动PWM

```
p.start(dc)   # where dc is the duty cycle (0.0 <= dc <=
100.0)
```

### 改变频率

```
p.ChangeFrequency(freq)   # where freq is the new frequency in Hz
```

### 改变占空比

```
p.ChangeDutyCycle(dc)  # where 0.0 <= dc <= 100.0
```

### 停止PWM

```
p.stop()
```

变量P超出范围时PWM也会停止.

LED每两秒闪烁一次:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 0.5)
p.start(1)
input('Press return to stop:')   # use raw_input for Python 2
p.stop()
GPIO.cleanup()
```

An example to brighten/dim an LED:

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 50)  # channel=12 frequency=50Hz
p.start(0)
try:
    while 1:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

# gpio_function(channel)

显示一个GPIO口的功能：

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
func = GPIO.gpio_function(pin)
```

will return a value from:
GPIO.IN, GPIO.OUT, GPIO.SPI, GPIO.I2C, GPIO.HARD_PWM, GPIO.SERIAL, GPIO.UNKNOWN