MapleJS
V1.0
行业硬件使能库

文档版本	01
发布日期	2019-05-10

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 简介

本文档为MapleJS编程框架所支持的行业硬件接口提供使用指导。

在MapleJS开发团队的努力之下，我们的引擎已能支持大量智能家居领域零部件。我们对其中绝大部分硬件能力进行了封装处理，以便于开发者更好地集中业务，提升开发效率。

本文档主要有传感器模块、控制模块、显示模块以及网络模块的硬件接口介绍。开发者可根据本文档调试及使用相关组件。其中网络接口模块相关文档待完善。

如有疑问，可邮件联系MapleJS开发团队，maplejs@huawei.com。

2 传感器模块接口

本子章节主要介绍各类传感器单元的使用方法，主要包括超声波传感器、二氧化碳传感器外传感器、热电偶传感器、陀螺仪加速度传感器、液位传感器、震动传感器以及通用传感器。其中通用传感器部分包含烟雾传感器、煤气传感器、酒精传感器、光敏电阻传感器、火焰传感器、声音传感器、霍尔传感器、土壤湿度传感器以及漫反射红外传感器，因上述传感器初始化等操作均一致,故归类为通用传感器。

2.1 超声波传感器

2.1.1 介绍

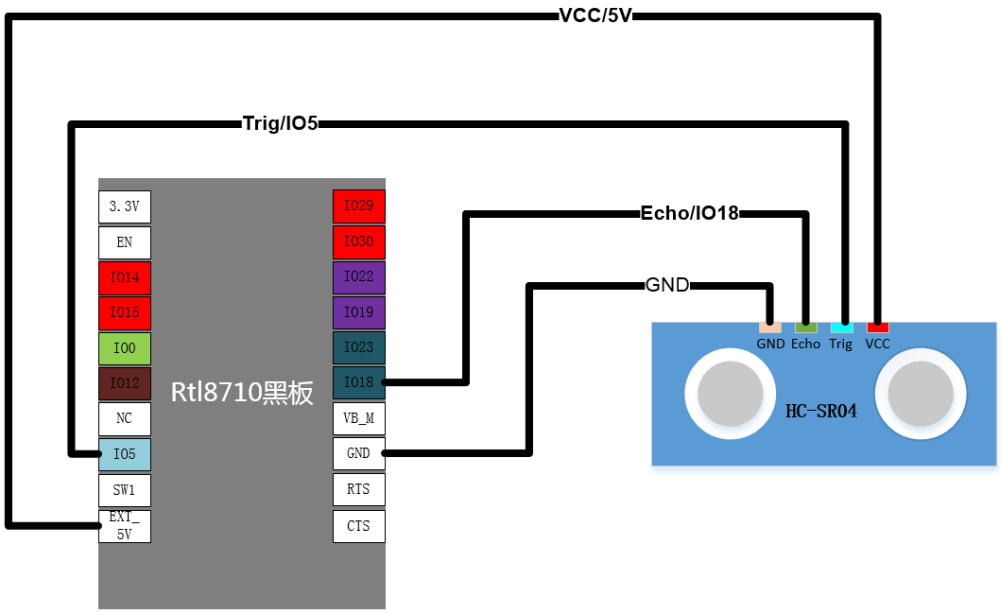
超声波传感器是将超声波信号转换成其他能量信号（通常是电信号）的传感器。超声波是振动频率高于20kHz的机械波。它具有频率高、波长短、绕射现象小，特别是方向性好、能够成为射线而定向传播等特点。超声波对液体、固体的穿透本领很大，尤其是在阳光不透明的固体中。超声波碰到杂质或分界面会产生显著反射形成反射回波，碰到活动物体能产生多普勒效应。超声波传感器广泛应用在工业、国防、生物医学等方面。

2.1.1.1 模块参数

HC-SR04模块性能稳定，测度距离精确，能和国外的SRF05、SRF02等超声波测距模块相媲美。模块高精度，盲区(2cm)超近，最大识别距离为450cm。

使用电压	5V
静态电流	15mA
工作频率	40Hz
最远射程	4m
最近射程	2cm
测量角度	15度
输入触发信号	10us的TTL脉冲
输出回响	输出TTL电平信号，与射程成正比
规格尺寸	45 * 20 * 15mm

2.1.1.2 连线方式



引脚说明

序号 1	模块引脚 VCC	开发板引脚 Ext_5V	说明 正极5V电源
2	GND	GND	负极接地
3	Trig	IO5	接收控制板传来的信号
4	Echo	IO18	将测出的距离结果返回给控制板

2.1.2 模块接口

2.1.2.1 引用模块

```
var ultrasound = require('ultrasound');
```

2.1.2.2 打开模块

```
ultrasound.open(freq_div,Trig);
```

功能描述

超声波传感器使用IO口与开发板相连，因此需要对通信IO初始化。

接口约束

无。

参数列表

- freq_div：分频，分频越小测量的精度越高。当freq_div为1时，测量距离范围是0-27cm，当freq_div为10时，测量范围是0-270cm，当freq_div为100时，测量距离范围是0-2700cm。因为此款超声波传感器的测量距离在2cm到450cm，所以freq_div的值是100。
- trig：number类型，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。

返回值

返回打开对象。

接口示例

```
var ultra = ultrasound.open(1,5);
```

2.1.2.3 获取距离

```
ultrasound.getDistance();
```

功能描述

通过串口读取距离障碍物的距离，单位cm。

接口约束

无。

参数列表

无。

返回值

距离障碍物的距离，单位cm。

接口示例

```
var distance = ultrasound.getDistance();
```

2.1.2.4 关闭模块

```
ultrasound.close();
```

功能描述

关闭ultrasound模块。

参数列表

无。

返回值

无。

接口示例

```
ultrasound.close();
```

2.1.3 约束

i.由于定时器timer4指定18号引脚，所以ECHO引脚只能连接18号引脚。

ii.被测目标必须垂直于超声波传感器。

iii.被测目标表面必须平坦。

iv.测量时在超声波传感器周围没有其他可反射超声波的物体。

v.由于发射功率有限，传感器无法测量10m外的物体。

2.1.4 样例

介绍

用超声波传感器测出到物体之间的距离。

例程

```
var ultrasound = require('ultrasound');
var time = require('timer');
var ultra = ultrasound.open(10,5);
time.setInterval(function(){
    var distance = ultra.getDistance();
    print("distance:",distance,"cm");
},1000);
```

2.2 二氧化碳传感器

2.2.1 介绍

二氧化碳传感器多用于检测空气中二氧化碳的浓度。 在大自然环境里，空气中二氧化碳的正常含量是0.04%（400 PPM)，在大城市里有时候达到500 PPM。室内没有人的情况下，二氧化碳浓度一般在500到700 PPM左右。

当二氧化碳的浓度达到1%(1000 PPM)时，人们会感到沉闷，注意力开始不集中，心悸。如果在不透气的卧室里二氧化碳达到1000 PPM，而我们连续睡觉8个小时，早上起床时我们会感觉没有休息好，不想起床。如果办公室的空气中CO2含量达到1000PPM，员工们的工作效率会下降。

二氧化碳浓度达到1500-2000 PPM时，人们会感到气喘、头痛、眩晕。两个人在密闭的卧室里睡一个晚上，二氧化碳的浓度很容易达到2000 PPM. 办公室的空气中CO2浓度达到2000PPM时，员工们会感觉很困，注意力不集中，精神疲劳。超过了2000PPM后，我们甚至不想继续工作，思考能力明显下降。

5000PPM以上时人体机能严重混乱，使人丧失知觉、神志不清。

2.2.1.1 模块参数

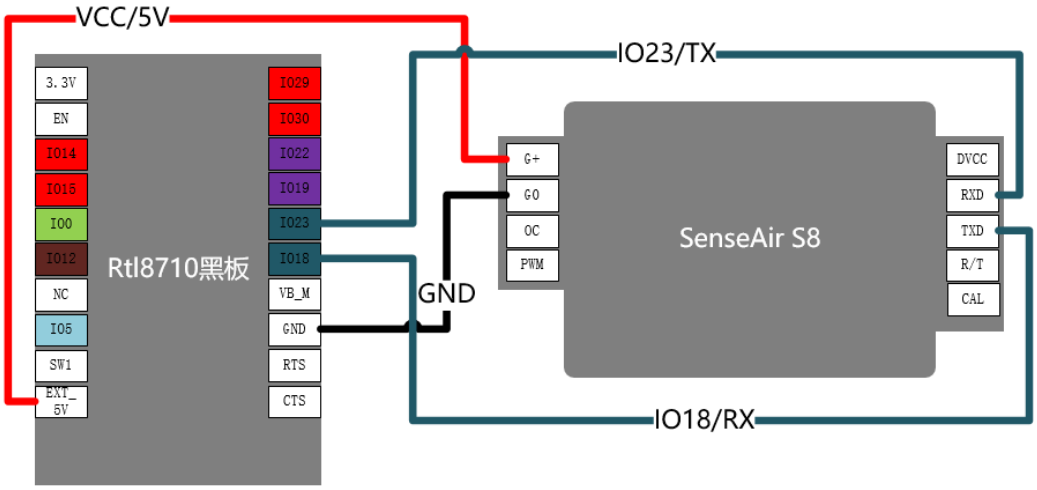
二氧化碳传感器S8是瑞典森尔senseair。

二氧化碳传感器S8参数列表:

测量气体	CO2
工作原理	非色散红外（NDIR）
测量范围	400～2000ppm，高达10000ppm的扩展范围
精度	±70ppm 读数的±3%
数据更新时间	2s
反应时间（t90）	2 分钟
工作温度	0～50℃
工作湿度	0～85%，非冷凝
存储温度	-40～70 ℃
尺寸	33.5 x 20 x 8.5 mm
重量	< 8克
电源	4.5V-5.25V
耗电	300毫安峰值，30mA平均
使用寿命	15年以上

数据通过UART（Modbus协议），方向控制引脚直接连接到RS485 接收器集成电路。报警输出，集电极开路 1000/800 正常状态进行大100毫安。在CO2高，或低功率，或传感器故障晶体管打开。 PWM 输出（1kHz）0到100％占空比为0至2000ppm。

2.2.1.2 连线方式



开发板管脚与模块管脚的连线表：

1	Ext_5V	G+	G+是传感器电源，范围是4.5V-5.25V
2	GND	G0	G0是传感器参考地
3	IO23/TX	RXD	RXD，即UART_RxD，是传感器UART的接收
4	IO18/RX	TXD	TXD，即UART_TxD，是传感器UART的发送

2.2.2 模块接口

2.2.2.1 引用模块

```
var s8 = require('senseair_s8');
```

2.2.2.2 打开模块

```
s8.open(config);
```

功能描述

S8使用串口与开发板相连，因此需要对通信串口初始化。

S8串口的配置参数是9600 bps，8bits，N，1 stopbit，即“9600 8 N 1”。

接口约束

无。

参数列表

- config 为uart端口的配置对象，它包括属性：uartNumber。
  - uartNumber: 内置uart编号，爱联rtl8710为例，值为0或1。 该属性是可选属性，默认为1 (0为调试端口，调试时用于部署脚本，此时，由于io29与io30同usb接口有复用关系，因此不能进行相关操作)。
- 0: 接收引脚为IO29 (即29)，发送引脚为IO30 (即30)。
- 1: 接收引脚为IO18 (即18)，发送引脚为IO23 (即23)。

返回值

无。

接口示例

```
var port = s8.open({portNumber:1});
```

2.2.2.3 读取二氧化碳浓度

```
number port.read();
```

功能描述

S8通过串口读取二氧化碳浓度值。读取浓度单位是ppm 。

接口约束

无。

参数列表

无。

返回值

二氧化碳浓度，取值范围是[0,2000]，正整数。

接口示例

```
var concentration =  port.read();
```

2.2.3 约束

模块的浓度数据2s更新一次，启动时间2分钟，因此注意读取接口的调用时机。

2.2.4 样例

介绍

每隔5000毫秒读取一次二氧化碳浓度。

例程

```
var s8 = require('senseair_s8');
var tim = require("timer");
var port = s8.open({portNumber:1});
tim.setInterval(function() {
    print("浓度："+port.read()+" ppm");
  }, 2000);
print("js execute done!");
```


2.3 红外接收头传感器

2.3.1 介绍

红外接收头传感器用于接收红外遥控器发出的信号。

2.3.1.1 模块参数

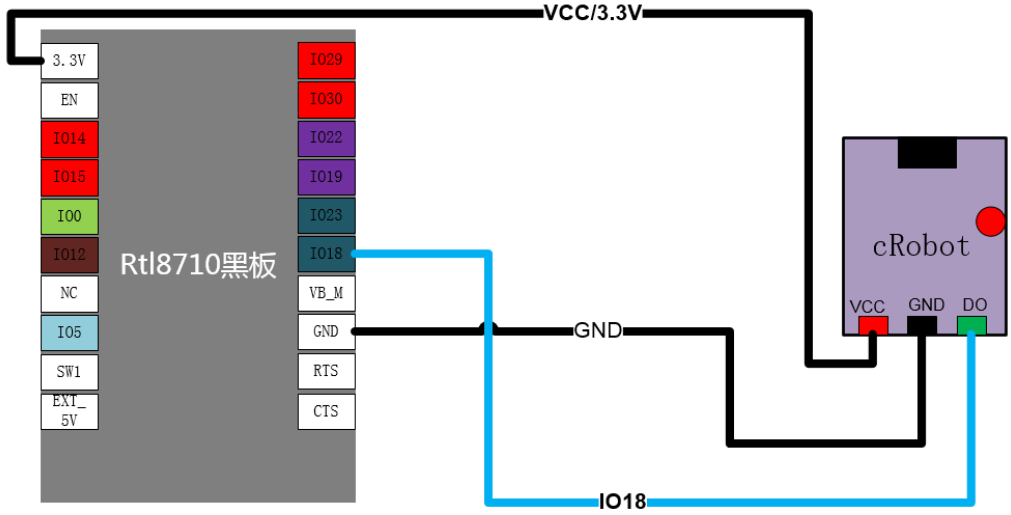
红外接收头传感器参数列表：

工作电压	**2.7-4.5V**
工作电流	1.7-2.7mA
接收频率	37.9kHz
峰值波长	940nm

工作电压	2.7-4.5V
静态输出	高电平
输出低电平	<=0.4V
输出高电平	接近工作电压

红外遥控器发出的信号是一连串的二进制脉冲码。为了使其在无线传输过程中免受其他红外信号的干扰,通常都是先将其调制在特定的载波频率上, 然后再经红外发射二极管发射出去,而红外线接收装置则要滤除其他杂波,只接收该特定频率的信号并将其还原成二进制脉冲码,也就是解调。内置接收管将红外发射管发射出来的光信号转换为微弱的电信号，此信号经由IC内部放大器进行放大，然后通过自动增益控制、带通滤波、解调变、 波形整形后还原为遥控器发射出的原始编码，经由接收头的信号输出脚输入到电器上的编码识别电路。

2.3.1.2 连线方式



引脚说明：

序号	开发板引脚	模块引脚	说明
1	Ext_5V	VCC	3.3-5V电源电压
2	GND	GND	负极接地
3	IO18	DO	数字信号输出

2.3.2 模块接口

2.3.2.1 引用模块

```
var infrared = require('infrared');
```


2.3.2.2 打开模块

```
infrared.open(freq_div);
```

功能描述

红外接收头传感器与开发板相连，因此需要对通信IO口进行初始化。

接口约束

无。

参数列表

freq_div：定时器分频，当freq_div为1时，脉冲宽度取值范围是0~1.6ms，当freq_div为10时，脉冲宽度取值范围是0~16ms等等，当freq_div为100时，脉冲宽度取值范围是0~160ms等等，freq_div越小精度越高，由于该款传感器需要捕捉的脉冲宽度在1.5ms~2.2ms之间，所以freq_div的值为10。

1	0~1.6ms
10	0~16ms
100	0~160ms
...	...

返回值

打开对象。

接口示例

```
var ir = infrared.open(10);
```

2.3.2.3 监听遥控器解码

```
ir.on(func,args);
```

功能描述

监听infrared端口返回的解码值。

接口约束

无。

参数列表

func：回调函数，当监听事件发生时调用该函数；

arg：传递给回调函数的参数，只允许一个参数，如果需要传递多个参数，需要把多个参数打包在一个结构体里。如果没有参数，该接口将默认传递undefined；

序号	返回值	对应键值
1	104	0
2	48	1

3	24	2
4	122	3
5	16	4
6	56	5
7	90	6
8	66	7
9	74	8
10	82	9
11	162	CH-
12	98	CH
13	226	CH+
14	34	|<<
15	2	>>|
16	194	>||
17	224	-
18	168	+
19	144	EQ
20	152	100+
21	176	200+

返回值

无。

接口示例

```
var rcv = ir.on(function(data){});
```

2.3.2.4 关闭模块

```
ir.close();
```

功能描述

关闭infrared模块。

参数列表

无。

返回值

无。

接口示例

```
ir.close();
```

2.3.3 约束

i 由于使用定时器timer4，该模块只能应用18号引脚。

ii 测试距离理论上小于8m。

2.3.4 样例

介绍

红外发射器发射信号，读取红外接收头接收信号并转换为十进制。

例程

```
var infrared = require('infrared');
var time = require('timer');
var ir = infrared.open(10);
ir.on(function(data){
    print("rcv:", data);
});
```

2.4 热电偶传感器

2.4.1 介绍

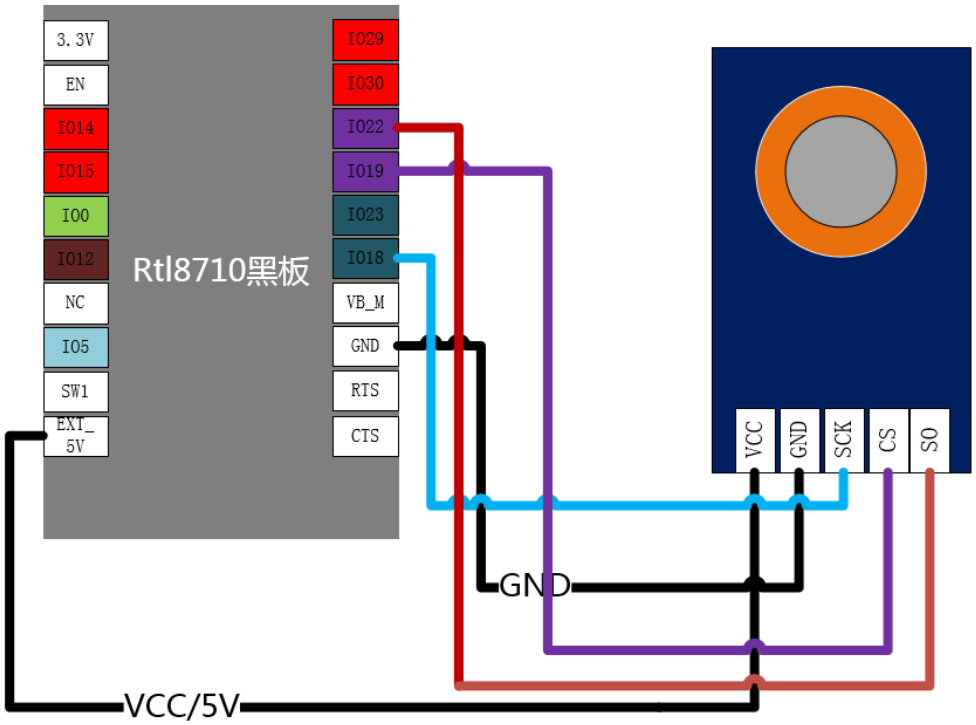
热电偶传感器是工业中使用最为普遍的接触式测温装置。这是因为热电偶具有性能稳定、测温范围大、信号可以远距离传输等特点，并且结构简单、使用方便。热电偶能够将热能直接转换为电信号，并且输出直流电压信号，使得显示、记录和传输都很容易。

2.4.1.1 模块参数

K型热电偶传感器,可以直接测量各种生产中从0℃到1300℃范围的液体蒸汽和气体介质以及固体的表面温度;通常和显示仪表,记录仪表和电子调节器配套使用,是目前用量最大的廉金属热电偶。

MAX6675冷端温度补偿、热电偶数字转换器可进行冷端温度补偿,并将K型热电偶信号转换成数字信号;数据输出为12位分辨率、SPI兼容、只读格式。 转换器温度分辨率为0.25°C,可读取温度达+1024°C,热电偶在0°C至+700°C温度范围内精度为8 LSB。

2.4.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	Ext_5V	VCC	VCC是传感器电源，范围是3.3V-5V
2	GND	GND	传感器接地端
3	IO18	SCK	时钟信号线，通讯数据同步用。时钟信号由通讯主机产生，它决定了SPI的通讯速率
4	IO19	CS	片选信号线，用于选中SPI从设备。当从设备上的SS引脚被置拉低时表明该从设备被主机选中
5	IO22	SO	主机(数据)输入/从设备(数据)输出引脚，即这条信号线上传输从从机到主机的数据

2.4.2 模块接口

2.4.2.1 引用模块

```
var Thermocouple = require('Thermocouple');
```

2.4.2.2 打开模块

```
thermocouple.open(config);
```

功能描述

打开热电偶传感器模块。

接口约束

无。

参数列表

- config:
  - spi : 0 (内置SPI编号,值为0，具体参见SPI模块)。
  - speed : 传输速率,最高到31.25MHz,类型为浮点数number。
  - bits : 数据帧大小，值为4-16的整数(这里必须是16,因为max6675的特性)。有一点需要注意的是，主机和从机数据帧需要一样。

- transmit_mode : 3 (SPI有4种工作模式，值分别为 0、1、2和3。工作模式用于设置时钟极性(CPOL)和时钟相位(CPHA)),具体解释参见SPI模块。

返回值

thermocouple对象。

接口示例

```
var thermocouple = thermocouple.open({spi:0,speed:2000000,bits:16,transmit_mode:3});
```

2.4.2.3 读取温度值

```
thermocouple.read();
```

功能描述

获取温度值。

接口约束

无。

参数列表

无。

返回值

温度值（0～1024摄氏度）。

接口示例

```
var temp = thermocouple.read();
```

2.4.2.4 关闭模块

```
thermocouple.close();
```

功能描述

关闭热电偶传感器模块。

参数列表

无。

返回值

无。

接口示例

```
thermocouple.close();
```

2.4.3 约束

由于max6675寄存器中的数据格式限制，模块的数据帧大小只能为16bits。

2.4.4 样例

介绍

1000毫秒读取一次温度。

例程

```
var thermocouple = require('thermocouple');
var tim = require('timer');
var config = {spi:0,speed:2000000,bits:16,transmit_mode:3};
var thermocouple = thermocouple.open(config);
tim.setInterval(function(){
    print("temp" + thermocouple.read());
},1000);
```

2.5 陀螺仪加速度传感器

2.5.1 介绍

陀螺仪又叫角速度传感器，是不同于加速度传感器（G-sensor）的，可获取物理量是偏转、倾斜时的转动角速度。多用于测试物体的运动状态。

2.5.1.1 模块参数

陀螺仪加速传感器MPU-6050参数列表：

使用芯片	MPU-6050
供电电源	3-5V（内部低压差稳压）
通讯方式	标准I2c通讯协议
芯片内置	16bitAD转换器，16位数据输出
陀螺仪范围	±250 500 1000 2000 °/s
加速范围	±2 ±4 ±8 ±16g
采用	沉金PCB，机器焊接工艺保证质量
引脚间距	2.54mm
温度传感器测量范围	-40~+85度
温度传感器线性误差	±1度
陀螺仪线性误差	0.1°/s
加速度输出频率	最高1000Hz

使用芯片	MPU-6050
陀螺仪最高分辨率	131LSB/（°/s）
DMP姿态解算频率	最高200Hz
陀螺仪输出频率	最高8000Hz

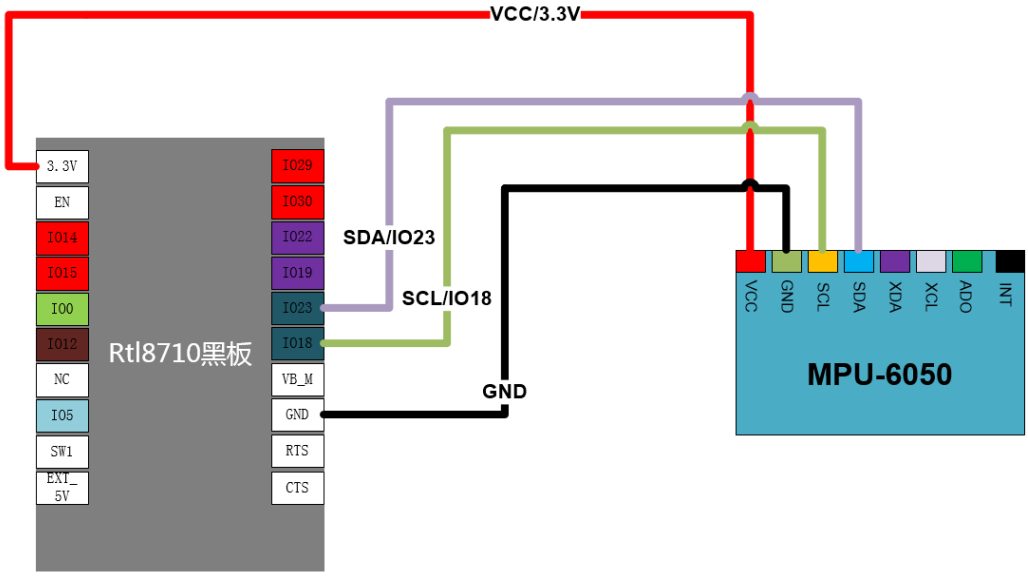
MPU-6050为整合性6轴运动处理组件，相较于多组件方案，免除了组合陀螺仪与加速器时之轴间差的问题，减少了大量的包装空间。MPU-6050整合了3轴陀螺仪，3轴加速器，并含可藉由第二个I2c端口连接其他厂牌的加速器，磁力传感器，或者其他传感器的数位运动处理硬件加速引擎，由主要I2c端口以单一的数据流形式，向应用端输出完整的9轴融合演算技术。

InvenSense的运动处理资料库，可处理运动感测的复杂数据，降低了运动处理运算对操作系统的负荷，并为应运开发提供架构化的API。

MPU-6050的角速度全格感测范围为±250、±500、±1000、±2000°/sec（dps），可准确追踪快速与慢速动作，并且，用户可程式控制的加速器全格感测范围为±2g，±4g，±8g与±16g。产品传输可透过最高至400kHz的I2c或者最高达20MHz的SPI。

MPU-6050内建频率产生器在所有温度范围仅有±1%1频率变化。

2.5.1.2 连线方式



引脚说明：

序号	开发板引脚	模块接口	备注
1	3.3V	VCC	3~5V电源电压
2	GND	GND	负极接地

序号	开发板引脚	模块接口	备注
3	IO23/TX	SCL	连接MCU的i2c接口
4	IO18/RX	SDA	连接MCU的i2c接口
5		XDA	连接外部从机设备
6		XCL	连接外部从机设备
7		ADO	IIC接口的地址控制引脚，该引脚控制IIC地址的最低位
8		INT	输入端

2.5.2 模块接口

2.5.2.1 引用模块

```
var mpu = require('mpu6050');
```

2.5.2.2 打开模块

```
mpu.open(i2c_num,frequency,address);
```

功能描述

mpu6050使用串口与开发板相连，因此需要对通信串口初始化。

接口约束

无。

参数列表

- i2c_num：number类型，当num为0时，SDA引脚为IO23 (即23),SCL引脚为IO18 (即18)。当num为1时，SDA引脚为IO19 (即19),SCL引脚为IO22 (即22)。
- frequency：传输速率，标准模式(0-100kb/s)，快速模式(<400kb/s)。
- address：陀螺仪加速传感器地址。

返回值

mpu对象。

接口示例

```
mpu.open(1,100000,0x68);
```

2.5.2.3 获取 gyroscope 加速度，角速度，温度

```
gyr.getMpuData(tmp);          //以获取温度为例
```

功能描述

读取温度，X、Y、Z轴的加速度和绕X、Y、Z轴的角速度。

接口约束

无。

参数列表

-tmp：mpu寄存器类型，参数如下：

- TEMP：获取当前温度（-40~+85度）；
- ACC_X：获取加速度X轴分量；
- ACC_Y：获取加速度Y轴分量；
- ACC_Z：获取加速度Z轴分量；
- GYR_X：获取绕X轴旋转的角速度；
- GYR_Y：获取绕Y轴旋转的角速度；
- GYR_Z：获取绕Z轴旋转的角速度。

返回值

根据所传入的参数返回温度值，或各个轴的加速度和角速度的值。

接口示例

```
var data = gyr.getMpuData(gyr.TEMP);
```

2.5.3 约束

无。

2.5.4 样例

介绍

获取温度；获取X轴加速度；获取Y轴加速度；获取Z轴加速度；获取绕X轴的角速度；获取绕Y轴的角速度；获取绕Z轴的角速度。

例程

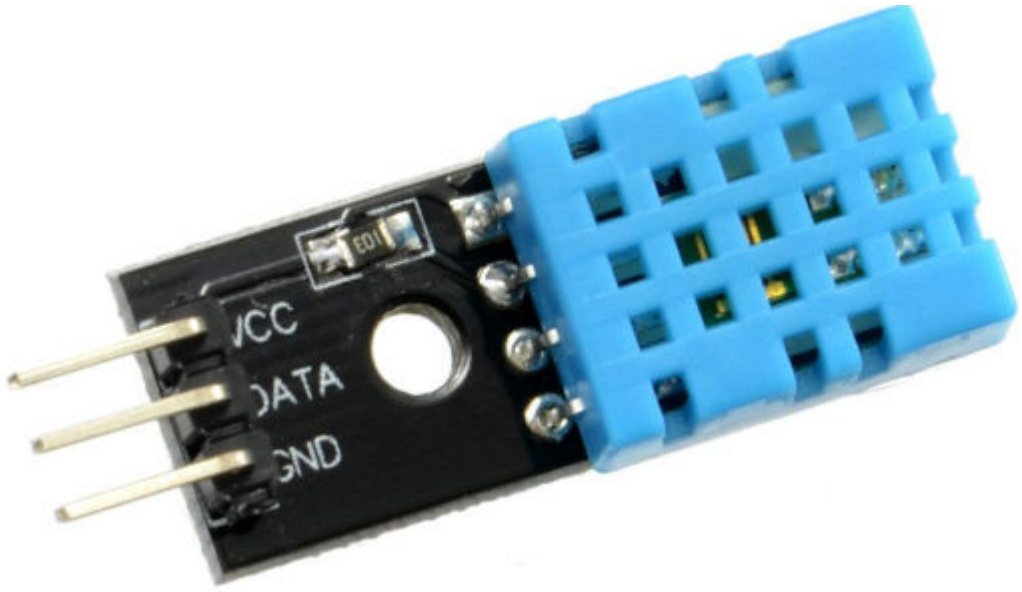
```
var mpu = require('mpu6050');
var gyr = mpu.open(0,100000,0x68);
var temp = gyr.getMpuData(gyr.TEMP);              //获取温度
print("temp=:",temp);
var acc_x = gyr.getMpuData(gyr.ACC_X);            //获取X轴加速度
print("acc_x=:",acc_x);
var acc_y = gyr.getMpuData(gyr.ACC_Y);            //获取Y轴加速度
print("acc_y=:",acc_y);
var acc_z = gyr.getMpuData(gyr.ACC_Z);            //获取Z轴加速度
print("acc_z=:",acc_z);
var gyr_x = gyr.getMpuData(gyr.GYR_X);           //获取绕X轴的角速度
print("gyr_x=:",gyr_x);
var gyr_y = gyr.getMpuData(gyr.GYR_Y);           //获取绕Y轴的角速度
print("gyr_y=:",gyr_y);
var gyr_z = gyr.getMpuData(gyr.GYR_Z);           //获取绕Z轴的角速度
print("gyr_z=:",gyr_z);
```

2.6 温湿度传感器

2.6.1 介绍

2.6.1.1 模块参数

DHT11数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。

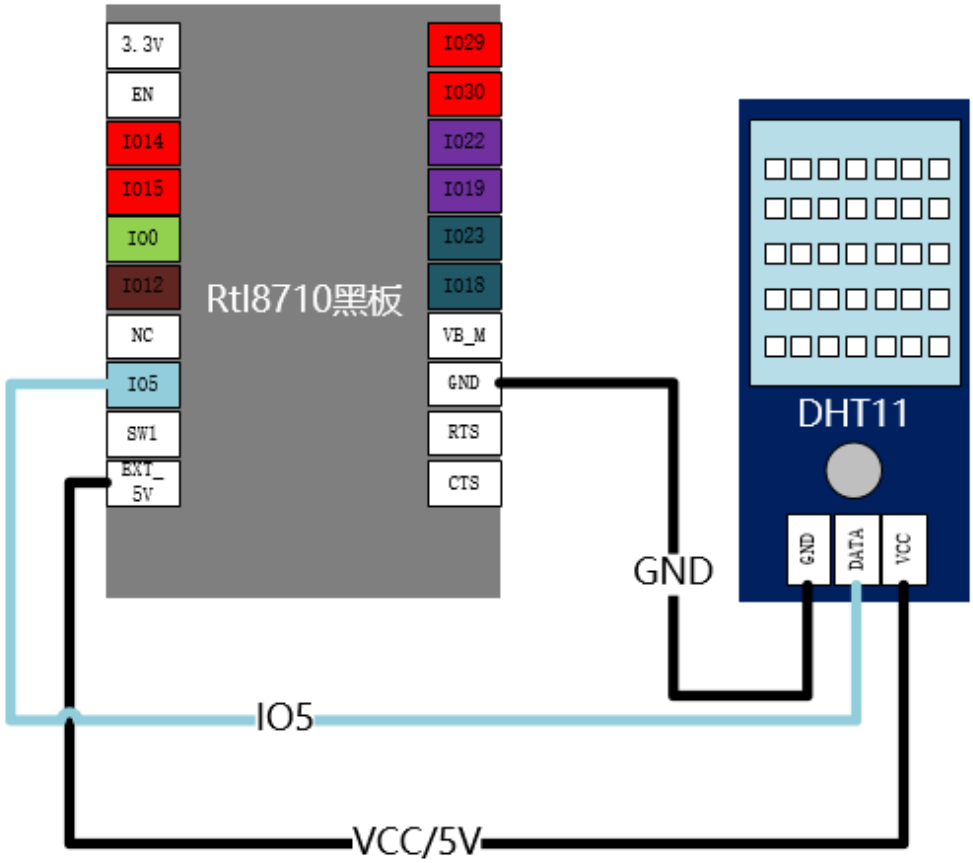


温湿度传感器参数：

参数	数值
接口	单线制串行接口，4针单排引脚封装
湿度精度	±5%RH
温度精度	±2℃
湿度范围	20~90%RH
温度范围	0~50℃

单个数据引脚端口完成输入输出双向传输，其数据包由5Byte（40Bit）组成；一次通讯时间最大3ms,数据分小数部分和整数部分。DHT11是单总线的，采用私有时序。
DHT11应用于暖通空调、测试及检测设备、汽车、数据记录器、消费品自动控制、气象站、家电、湿度调节器、医疗、除湿器等场景。

2.6.1.2 连线方式



开发板与模块管脚连接表：

序号	开发板管脚	模块管脚	备注
1	Ext_5v	VDD	VDD是3.5V-5.5V DC
2	IO5	DATA	DATA是串行数据,单总线
3	GND	GND	GND是接地,电源负极

2.6.2 模块接口

2.6.2.1 引用模块

```
var dht11 = require('dht11');
```

2.6.2.2 初始化模块

```
dth11.init(ionum)
```

功能描述

根据配置初始化DHT11端口。

接口约束

无。

参数列表

- ionum是使用开发板的IO号，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。

返回值

无。

接口示例

```
dth11.init(5);
```

2.6.2.3 读取湿度

```
dht11.getHumidity()
```

功能描述

从DHT11端口读取湿度值。

接口约束

无。

参数列表

无。

返回值

湿度值，单位是%，范围是[0.0,100.0]的正数，可以有1位小数。

接口示例

```
var humi = dht11.getHumidity();
```

2.6.2.4 读取温度

```
dht11.getTemp()
```

功能描述

从DHT11端口读取温度值。

接口约束

无。

参数列表

无。

返回值

温度值，单位摄氏度，范围是[0.0,50.0]的正数，可以有1位小数。

接口示例

```
var temp = dht11.getTemp();
```

2.6.2.5 读取温湿度

```
dht11.getTempHumi()
```

功能描述

从DHT11端口同时读取温度值和湿度值。DHT11总线是一次能输出温度值和湿度值。

接口约束

无。

参数列表

无。

返回值

js对象，包括属性temp和humidity，temp范围是[0.0,50.0]的正数，可以有1位小数，humidity范围是[0.0,100.0]的正数，可以有1位小数。

接口示例

```
var obj =dht.getTempHumi();
print("温度: "+obj.temp+" C");
print("湿度: "+obj.humidity+" %");
```

2.6.3 约束

模块的温湿度读取频次不能高于2s一次。

2.6.4 样例

介绍

温湿度读取并显示。

例程

```
var dht11 = require('dht11');
var tim = require('timer');
dht11.init(5);
tim.setInterval(function(){
  print("humi:"+dht11.getHumidity()+"%");
  tim.setDelay(2000);
  print("temp:"+dht11.getTemp());
  tim.setDelay(2000);
},6000);
print("js execute done!");
```

2.7 颜色传感器

2.7.1 介绍

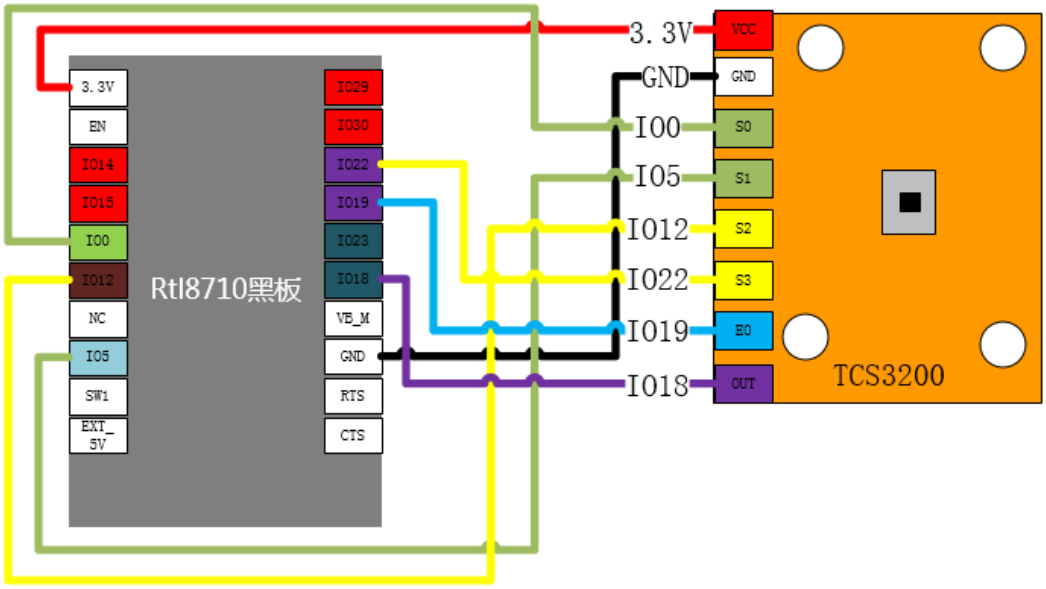
2.7.1.1 模块参数

TCS3200颜色传感器是一款全彩的颜色检测器，包括了一块TAOS TCS3200RGB感应芯片和4个白色LED灯，TCS3200能在一定的范围内检测和测量几乎所有的可见光 TCS3200有大量的光检测器，每个都有红绿蓝和清除4种滤光器 每6种颜色滤光器均匀地按数组分布来清除颜色中偏移位置的颜色分量 内置的振荡器能输出方波，其频率与所选择的光的强度成比例关系。

引脚说明

序号	模块引脚	说明
1	VCC	3~5V电源电压
2	GND	负极接地
3	S0,S1	选择输出比例因子或电源关断模式
4	S2,S3	选择滤波器类型
5	EO	频率输出使能引脚，可控制输出状态
6	OUT	频率输出引脚

2.7.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	3.3V	VCC	电源VCC
2	GND	GND	接地
3	IO0	S0	与S1配合，设置输出频率比例
4	IO5	S1	与S0配合，设置输出频率比例
5	IO12	S2	与S3配合，选择颜色滤波器
6	IO22	S3	与S2配合，选择颜色滤波器
7	IO19	E0	输出使能引脚，低电平使能
8	IO18	OUT	输出

2.7.2 模块接口

2.7.2.1 引用模块

```
var tcs3200 = require('tcs3200');
```


2.7.2.2 打开模块

```
tcs3200.open(a,b,c,d,e,f);
```

功能描述

根据配置打开颜色传感器端口。

接口约束

无。

参数列表

- a,b:对应s0,s1引脚，选择不同的输出比例因子;
- c,d:对应s2,s3引脚，选择不同色光的滤波器;
- e:对应out引脚，输出RGB三原色对应的频率;
- f:对应EO引脚，控制颜色传感器开关;
- a，b，c，d，e，f为number类型，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。

返回值

颜色传感器接口对象。

接口示例

```
var color = tcs3200.open(0, 5, 12, 18, 19, 22);
```

2.7.2.3 设置白平衡

```
colorPin.balance(void)
```

功能描述

模块根据目标颜色设置白平衡，后续测量时将根据白平衡结果基数去计算颜色测量结果。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
color.balance();
```

2.7.2.4 测量目标颜色

```
color.measure(void)
```

功能描述

测量目标颜色，根据白平衡结果基数计算并输出颜色RGB值。

参数列表

无。

返回值

number类型，目标RGB配色值，如白色0xFFFFFF;

接口示例

```
var value = color.measure();
```

2.7.3 约束

无。

2.7.4 样例

介绍

打开颜色传感器，设置白平衡，五秒后测量颜色值并打印输出。

例程

```
var color = require('tcs3200');
var time = require('timer');
var port = color.open(0, 5, 12, 18, 19, 22);
port.balance();
time.setDelay(5000);
print("rgb:" + port.measure);
```

2.8 液位传感器

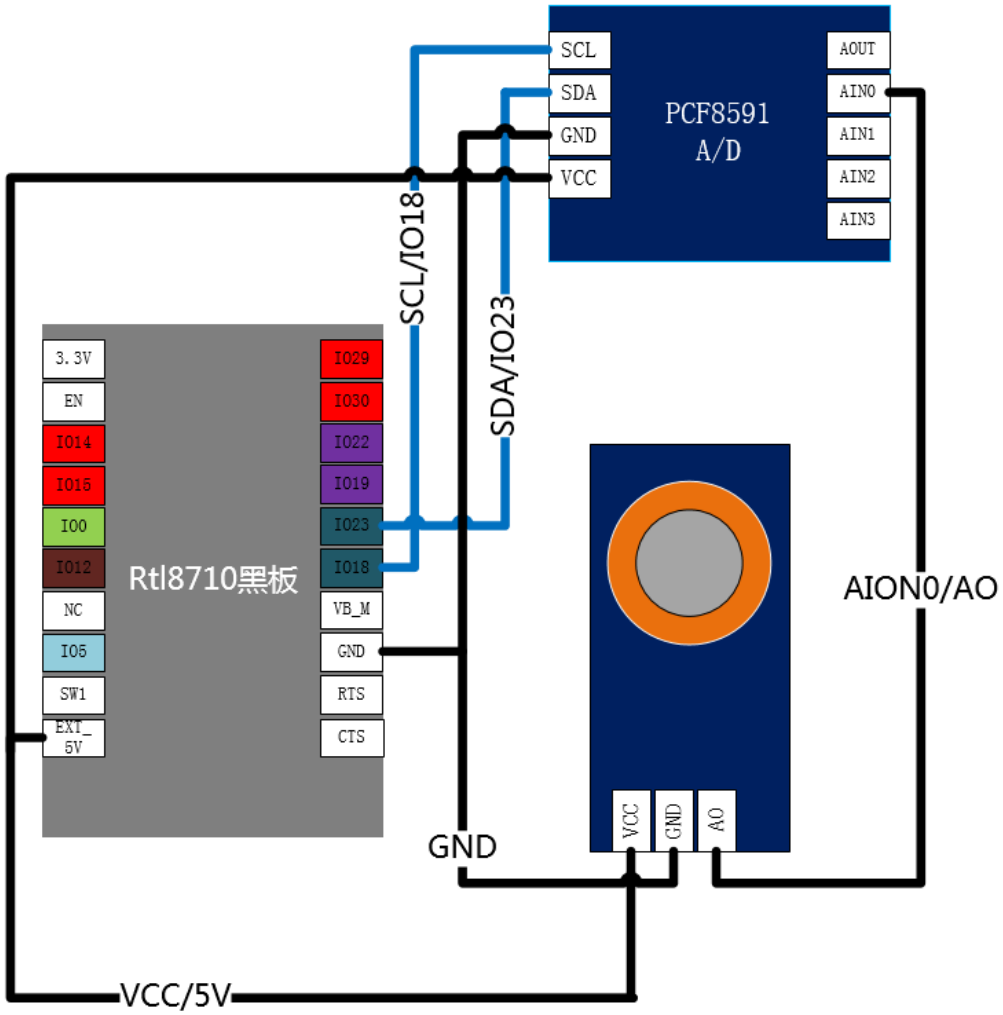
2.8.1 介绍

液位传感器（静压液位计/液位变送器/液位传感器/水位传感器）是一种测量液位的压力传感器。

2.8.1.1 模块参数

静压投入式液位变送器（液位计）是基于所测液体静压与该液体的高度成比例的原理，采用国外先进的隔离型扩散硅敏感元件或陶瓷电容压力敏感传感器，将静压转换为电信号，再经过温度补偿和线性修正，转化成标准电信号。传感器如下图:

2.8.1.2 连线方式



AO模拟量,开发板管脚、AD转换器和模块管脚的接线表:

序号	开发板管脚	AD转换器管脚	模块管脚	说明
1	5V	VCC	VCC	外接 3.3V-5V
2	GND	GND	GND	外接GND
3	IO18/SCL	SCL	AO(连接AD转换器AINO)	传感器模拟量输出接口
4	IO23/SDA	SDA	AO(连接AD转换器AINO)	传感器模拟量输出接口

2.8.2 模块接口

2.8.2.1 引用模块

```
var liquid = require ('liquid');
```

2.8.2.2 打开模块

```
liquid.open (config);
```

功能描述

根据配置打开端口。

接口约束

无。

参数列表

config :

- mode : liquid.AO,模拟量模式。
- i2c : 内置i2c编号,值为0或1;详细解释参见I2C模块。
- speed : 传输速率,标准模式（0-100kb/s),快速模式(<400kb/s)。
- address : 从设备地址,仅支持7-bit address。

返回值

遵循I2C协议的liquid端口对象。

接口示例

```
var i2c_port = liquid.open({mode:liquid.AO,i2c:0,speed:100000,address:0x48});
```

2.8.2.3 读取 AO 模拟量

```
i2c_port.read();
```

功能描述

通过AD转换器，读到传感器输出的模拟值。

接口约束

无。

参数列表

无。

返回值

level 传感器读到的模拟值(值越低，说明水位深度越深，范围（0~255）)。

2.8.2.4 关闭模块

```
i2c_port.close();
```

功能描述

关闭传感器功能。

参数列表

无。

2.8.3 约束

无。

2.8.4 样例

介绍

获取液位模拟量。

例程

```
var liquid = require ('liquid');
var tim = require ("timer");
var config ={mode:liquid.AO,i2c:0,speed:100000,address:0x48};
var i2c_port = liquid.open(config);
tim.setInterval(function(){
    var level = i2c_port.read();
    print("level",level);
},1000);
```

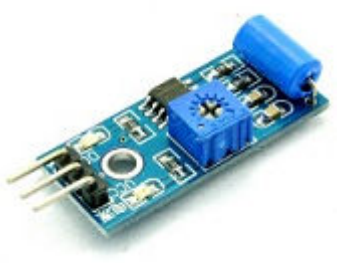
2.9 震动传感器

2.9.1 介绍

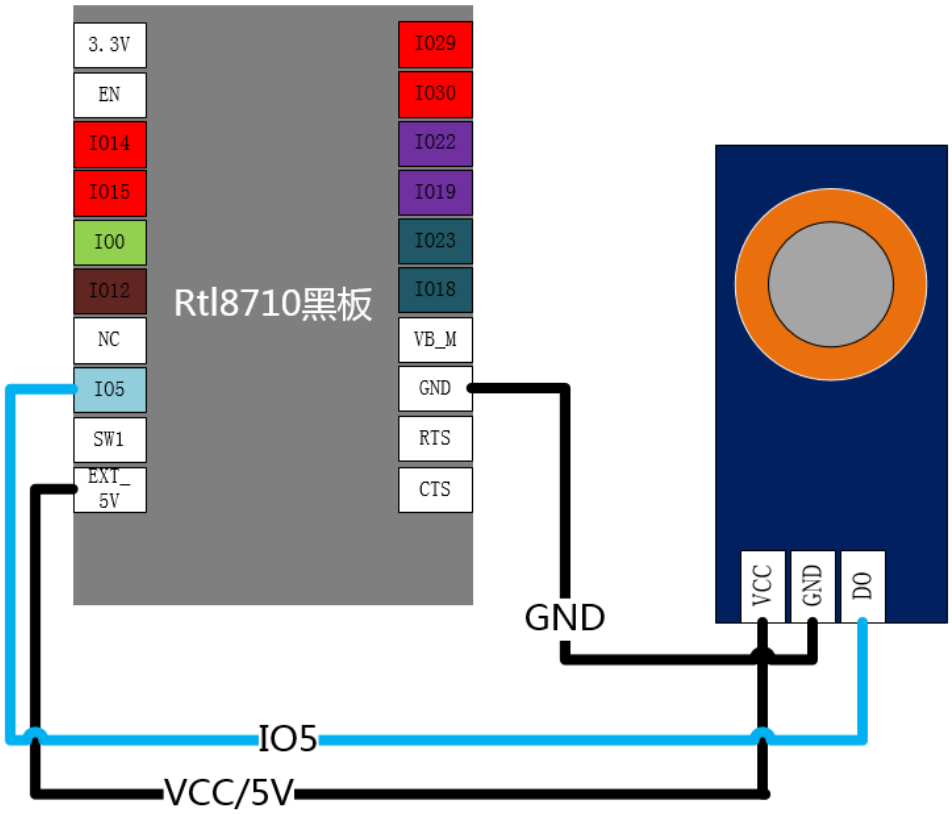
震动传感器，可用于检测物体是否处于震动状态。

2.9.1.1 模块参数

内部含有滚珠开关。滚珠开关，其内部含有导电珠子，器件一旦震动，珠子随之滚动，就能使两端的导针导通;传感器如下图:



2.9.1.2 连线方式



DO数字量,开发板管脚和模块管脚的连线表:

序号	开发板管脚	模块管脚	说明
1	5V	VCC	VCC是传感器电源正极,范围是3.3V-5V
2	GND	GND	GND是传感器参考地
3	IO5	DO	DO是传感器数字量输出接口

2.9.2 模块接口

2.9.2.1 引用模块

```
var sensor = require ('shock');
```

2.9.2.2 打开模块

```
shock.open (config);
```

功能描述

打开震动传感器。

接口约束

无。

参数列表

config:

- pin： 引脚号,以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- mode : 模式,shock.DO。

返回值:

- pin_do : 传感器对象;

2.9.2.3 读取 AO 数字量

```
pin_do.read ();
```

功能描述

读取震动传感器AO数字量。

接口约束

无。

参数列表

无。

返回值

- num : 读到的值
  - 1 (表示传感器正常)。
  - 0（表示传感器被外界条件触发）。

接口示例

```
var level = pin_do.read();
```

2.9.2.4 关闭模块

```
pin_do.close();
```

功能描述

关闭震动传感器端口。

参数列表

无。

返回值

无。

接口示例

```
pin_do.close();
```


2.9.2.5 监听端口

```
pin_do.on (event, func, arg);
```

功能描述

设置震动产生时触发的回调事件。 注 : 可以通过旋转传感器上的十字螺母，以调节震动传感器的领命读。

接口约束

无。

参数列表

- event : 回调事件的事件类型
  - shock.SHCOK_DOWN ： 震动事件。震动传感器检测到震动。
  - shock.SHOCK_UP ： 震动结束事件。震动传感器检测到震动后恢复正常时触发的事件。
- func : 回调函数,监听高低电平转换事件触发时调用该函数 。
- arg : 传递给回调函数的参数,只允许一个参数,如果需要传递多个参数,需要把多个参数打包在一个结构体里;如果没有参数,该接口将默认传递undefined。

返回值

无。

2.9.3 约束

无。

2.9.4 样例

介绍

传感器检测到震动时，触发“打印DO口读到的值”的事件。

例程

```
var shock = require ('shock');
var tim = require ('timer');
var config = {mode:shock.DO,pin:5};
var pin_do = shock.open (config);
pin_do.on(shock.SHOCK_UP,function(){
  var level = pin_do.read();
  print ("sensor status is changing! level is ",level);
});
```

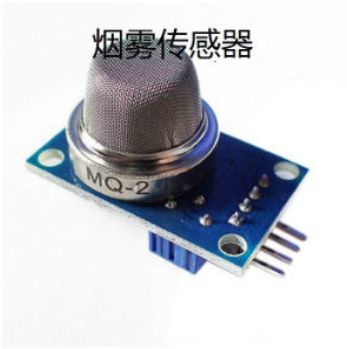
2.10 通用传感器

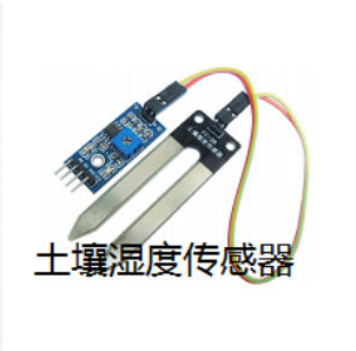
2.10.1 介绍

本节包含烟雾传感器、煤气传感器、酒精传感器、光敏电阻传感器、火焰传感器、声音传感器、霍尔传感器、土壤湿度传感器以及漫反射红外传感器；本模块中所提及传感器初始化等操作均一致,故归类为通用传感器，详细支持型号请参见模块参数。

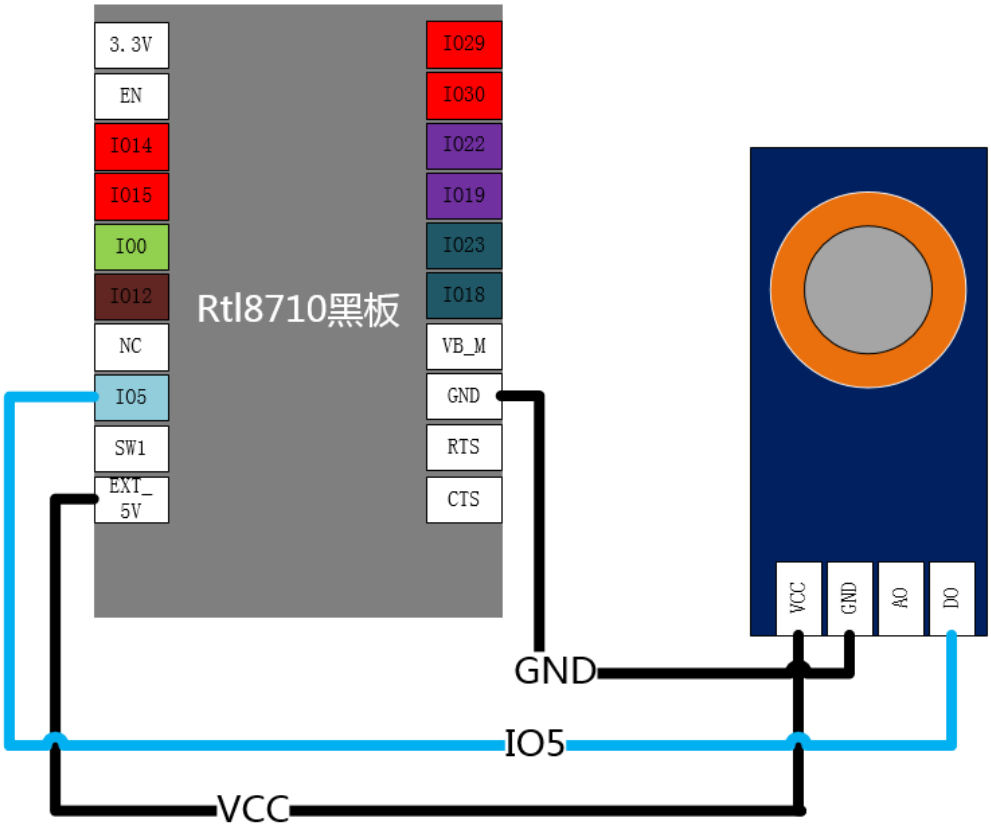
2.10.1.1 模块参数

环境条件达不到设定的阈值,DO输出高电平，当环境条件超过设定的阈值,DO输出低电平；适用传感器有：烟雾传感器MQ-2、煤气传感器MQ-5、酒精传感器MQ-3、光敏电阻传感器、火焰传感器、声音传感器、霍尔传感器FC-03、土壤湿度传感器YL-69以及漫反射红外传感器；相关传感器图片如下：



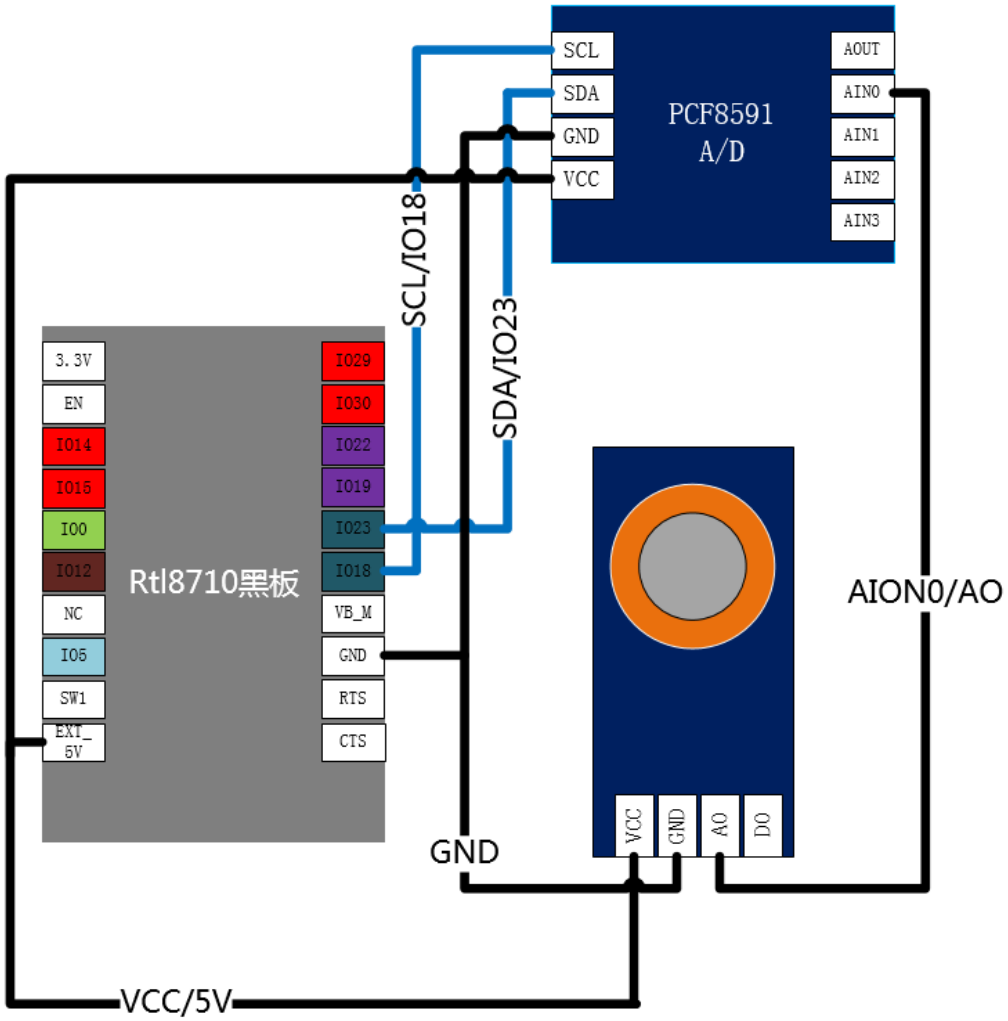


2.10.1.2 连线方式



DO数字量,开发板管脚和模块管脚的连线表:

序号	开发板管脚	模块管脚	说明
1	5V	VCC	VCC是传感器电源正极,范围是3.3V-5V
2	GND	GND	GND是传感器参考地
3	IO5	DO	DO是传感器数字量输出接口



AO模拟量,开发板管脚、AD转换器和模块管脚的接线表:

序号	开发板管脚	AD转换器管脚	模块管脚	说明
1	5V	VCC	VCC	外接 3.3V-5V
2	GND	GND	GND	外接GND
3	IO18/SCL	SCL	AO(连接AD转换器AINO)	传感器模拟量输出接口
4	IO23/SDA	SDA	AO(连接AD转换器AINO)	传感器模拟量输出接口

开发板和传感器直接相连或者开发板通过AD转换器间接和传感器相连。

2.10.2 模块接口

2.10.2.1 引用模块

```
var sensor = require ('sensor');
```

2.10.2.2 打开模块

```
sensor.open (config);
```

功能描述

打开并初始化传感器。

功能描述

打开传感器DO端口。

接口约束

无。

参数列表

config:

- pin： 引脚号,以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- mode : 模式,sensor.DO。

返回值:

pin_do : 返回一个sensor中GPIO端口对象。

接口示例

```
var config = {mode:sensor.DO,pin:5};
var pin_do = sensor.open (config);
```

功能描述

打开一个遵循I2C协议的sensor端口,此端口读取通过AD转化的模拟量。

接口约束

无。

参数列表

config_1 :

- mode : sensor.AO,模拟量模式。
- i2c : 内置i2c编号,值为0或1;详细解释参见I2C模块。
- speed : 传输速率,标准模式（0-100kb/s),快速模式(<400kb/s)。
- address : 从设备地址,仅支持7-bit address。

返回值

遵循I2C协议的sensor端口。

接口示例

```
var config_1 = {mode:sensor.AO,i2c:0,speed:100000,address:0x48};
var i2c = sensor.open(config_1);
```


2.10.2.3 从打开端口中获取数据

```
pin_do.read ();
```

功能描述

调用sensor中读功能,根据打开DO端口或者AO端口,分别获取数字量或模拟量数据。

接口约束

无。

参数列表

无。

返回值

- DO方式打开：获得数字量,值域(0或1)。
- AO方式打开：获得模拟量,值域为(0-255)。

接口示例

```
var num = pin_do.read();
```

2.10.2.4 关闭模块

```
pin_do.close();
```

功能描述

关闭传感器端口。

参数列表

无。

返回值

无。

接口示例

```
pin_do.close();
```

2.10.2.5 监听端口

```
pin_do.on (event, func, arg);
```

功能描述

调用sensor模块,打开监听功能。

参数列表

- event : 事件类型
  - sensor.SENSOR_UP(外界条件正常)。
  - sensor.SENSOR_DOWN(外界条件超过阈值,传感器触发)。
- func : 回调函数,当事件event触发时所调用的函数。
- arg : 传递给回调函数的参数,通常只允许一个参数；如需传递多个参数,需要把多个参数打包在一个结构体里；如无参数,该接口将默认传递undefined。

返回值

无。

接口示例

```
pin_do.on(sensor.SENSOR_UP,function(){
  var level = pin_do.read();
  print ("sensor status is changing! level is ",level);
});
```

2.10.3 约束

无。

2.10.4 样例

样例A

介绍

传感器超过阈值（如烟雾传感器浓度超过阈值）时,触发事件。

例程

```
var sensor = require ('sensor');
var tim = require ('timer');
var config = {mode:sensor.DO,pin:5};
var pin_do = sensor.open (config);
pin_do.on(sensor.SENSOR_UP,function(){
  var level = pin_do.read();
  print ("sensor status is changing! level is ",level);
});
```

样例B

介绍

一秒为周期,周期性获取AD转换器读到的模拟量。

例程

```
var sensor = require ('sensor');
var tim = require ("timer");
var config_1 = {mode:sensor.AO,i2c:0,speed:100000,address:0x48};
var i2c = sensor.open(config_1);
tim.setInterval(function(){
    var level = i2c.read();
    print("level",level);
},1000);
```


3 控制模块接口

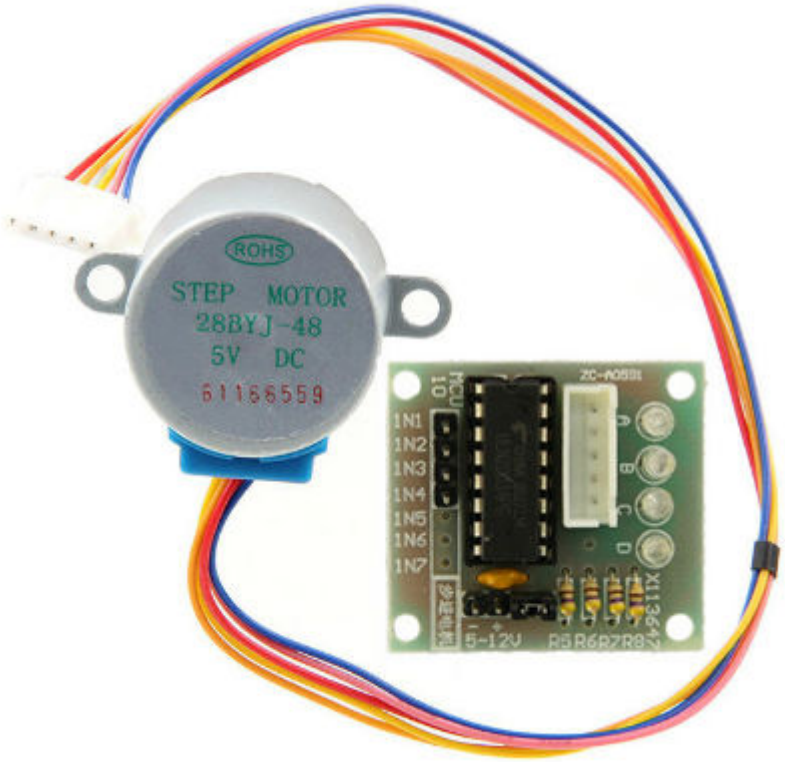
3.1 5V 步进电机

3.1.1 介绍

步进电机是将电脉冲信号转变为角位移或线位移的开环控制电机，是现代数字程序控制系统中的主要执行元件，应用极为广泛。

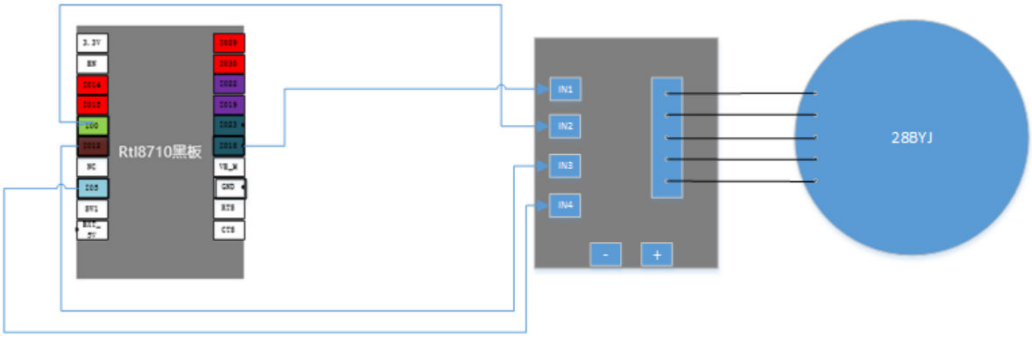
3.1.1.1 模块参数

此模块提供接口给用户控制28BYJ-48步进电机旋转。28BYJ-48步进电机如下图：



3.1.1.2 连线方式

由于板子输出的电流不足以带动28BYJ-48电机，需要ULN2003驱动模块才能驱动。ULN2003驱动板上的接口有防呆口设置，电机接上ULN2003后，开发板与驱动模块引脚对应连线如下图：



序号	开发板管脚	模块引脚	说明
1	IO18	IN1	与电机蓝线短接，用于输出高/低电平控制电机运动
2	IO0	IN2	与电机粉线短接，用于输出高/低电平控制电机运动

3	IO12	IN3	与电机黄线短接，用于输出高/低电平控制电机运动
4	IO5	IN4	与电机橙线短接，用于输出高/低电平控制电机运动
5		+	外接电源正极，5~12V
6		-	外接电源负极

3.1.2 模块接口

3.1.2.1 引用模块

```
var motor= require ('nanostepper');
```

3.1.2.2 打开模块

```
motor.open(pin_array,work_mode);
```

功能描述

motor.open根据参数配置打开引脚并设置相关属性，成功初始化后返回一个motorObj对象。

接口约束

无。

参数列表

- pin_array : 控制引脚数组，数组内引脚顺序按照与IN1~IN4连接的顺序排列，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- work_mode： 步进电机的运行模式，根据设置值不同，以不同方式运行。
  - 0 : 单四拍
  - 1 : 双四拍
  - 2 : 八拍

返回值

motorobj : 电机对象。

接口示例

```
motor.open([18, 0, 12, 5], 0);
```

3.1.2.3 驱动电机旋转

```
myMotor.to(angle,dir);
```

功能描述

控制电机旋转指定角度。

接口约束

无。

参数列表

- angle: 范围(任意整数)，电机旋转的角度。
- dir : 范围(0,1)，电机旋转的方向，设置为0时逆时针移动，设置为1时顺时针移动，缺省值为0。

返回值

无。

接口示例

```
myMotor.to(90,1);
```

3.1.3 约束

无。

3.1.4 样例

介绍

电机以单四拍运行，顺时针移动90°。

连线图

参考4.2.2.1.1模块连线。

例程

```
var motor = require("nanostepper");
var myMotor = motor.open([18, 0, 12, 5], 0);
/* 参数说明:
io18连IN1
io0连IN2
io12连IN3
io5连IN4
电机以单四拍运行。
*/
myMotor.to (90, 1);
/* 电机顺时针移动90°  */
```

3.2 42V 步进电机

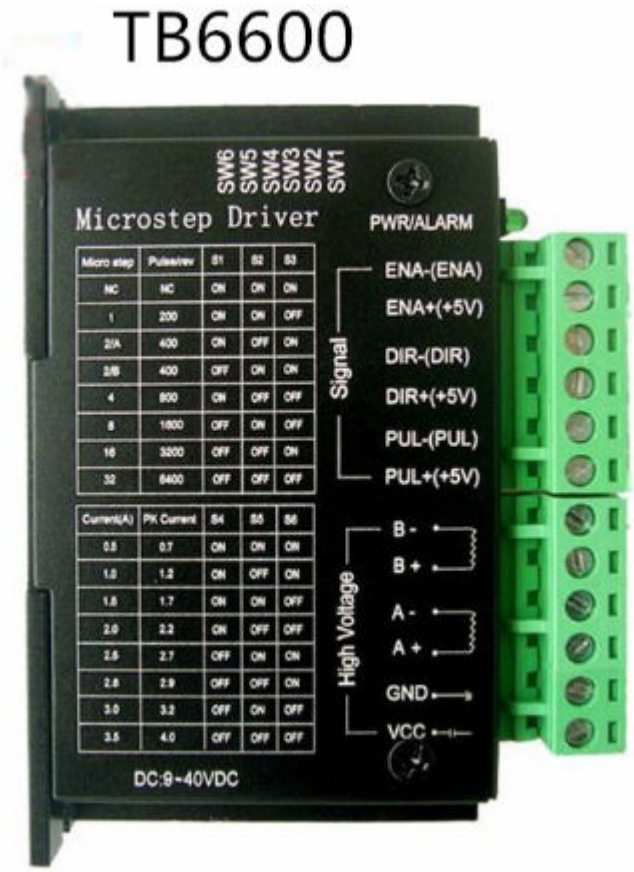
3.2.1 介绍

步进电机是将电脉冲信号转变为角位移或线位移的开环控制电机，是现代数字程序控制系统中的主要执行元件，应用极为广泛。

3.2.1.1 模块参数

此模块提供接口给用户控制SL42STH34-1504A 步进电机旋转, 需结合TB600驱动模块使用，TB6600是两相步进电机驱动。可实现正反转控制，通过 3 位拨码开关，可以选择

7 档细分控制（1、2/A、2/B、4、8、16、32），8 档电流控制（0.5A、1A、1.5A、2A、2.5A、2.8A、3.0A、3.5A）。适合驱动 57、42 型两相、四相混合式步进电机，能达到低振动、低噪声、高速度的驱动效果。电机如下图：



TB6600输入输出

- 信号输入端
  - PUL+ : 脉冲信号输入正
  - PUL- : 脉冲信号输入负
  - DIR+ : 电机正、反转控制正
  - DIR- : 电机正、反转控制负
  - EN+ : 电机脱机控制正
  - EN- : 电机脱机控制负
- 电机绕组连接
  - A+ : 连接电机绕组 A+相
  - A- : 连接电机绕组 A-相
  - B+ : 连接电机绕组 B+相
  - B- : 连接电机绕组 B-相
- 电源电压连接
  - VCC : 电源正端 “+”
  - GND : 电源负端 “-” 注意：DC9-42V,不可以超过此范围，否则将无法正常工作，甚至损坏驱动器。

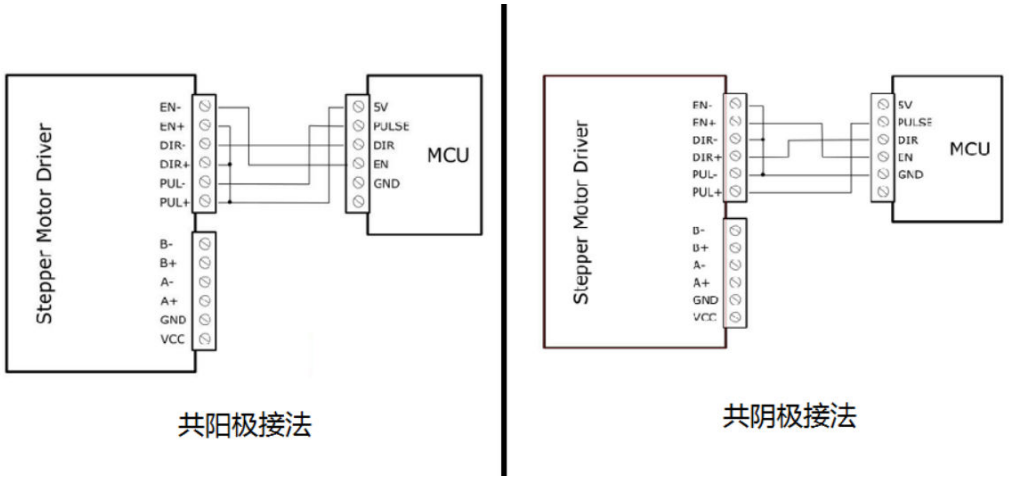
输入端接线说明

输入信号共有三路，它们是 :

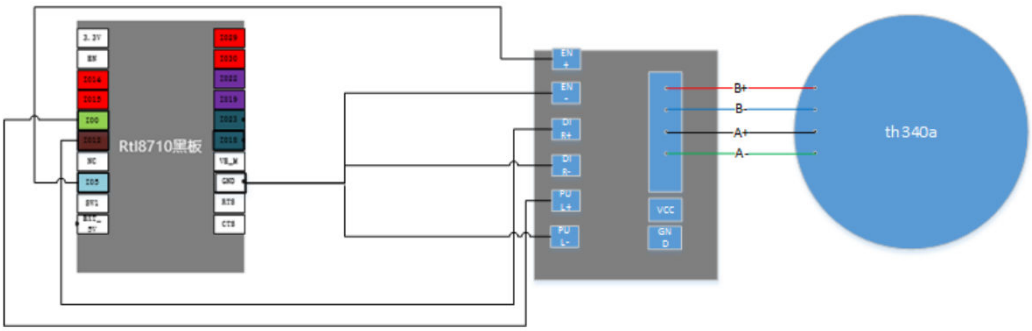
- 步进脉冲信号 PUL+，PUL- 。
- 方向电平信号 DIR+，DIR- 。
- 脱机信号 EN+，EN-。

3.2.1.2 连线方式

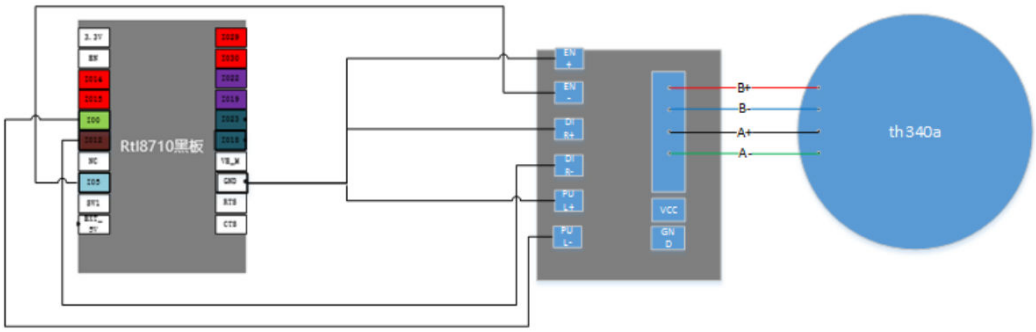
TB6600驱动板连线方式分为两种，用户可根据需要采用共阳极接法或共阴极接法，接法如下图：



共阴极接法



共阳极接法



序号	开发板管脚	TB6600驱动板	步进电机引脚	说明
1	IO5	EN+(共阴)/EN-(共阳)		共阴极接法IO5接EN+，共阳极接法IO5接EN-
2	IO12	DIR+(共阴)/ DIR-(共阳)		共阴极接法IO5接DIR+，共阳极接法IO5接DIR-
3	IO0	PUL+(共阴)/ PUL-(共阳)		共阴极接法IO5接PUL+，共阳极接法IO5接PUL-
4		B+	红线	与电机红线短接，用于输出高/低电平控制电机运动

序号	开发板管脚	TB6600驱动板	步进电机引脚	说明
5		B-	蓝线	与电机蓝线短接，用于输出高/低电平控制电机运动
6		A+	黑线	与电机黑线短接，用于输出高/低电平控制电机运动
7		A-	绿线	与电机绿线短接，用于输出高/低电平控制电机运动
8		GND		外接电源地
9		VCC		外接电源正极(9-12V)

由于SL42STH34-1504A模块的驱动电流为1.5A,需要将TB6600的S4~S6设置为ON、ON、OFF状态。

3.2.2 模块接口

3.2.2.1 引用模块

```
var motor= require ('stepper');
```

3.2.2.2 打开模块

```
motor.open(pin_array,connect_mode,step);
```

功能描述

motor.open根据config配置打开引脚并设置相关属性，成功初始化后返回一个motorObj对象。

接口约束

无。

参数列表

- pin_array : 控制引脚数组，数组内引脚顺序按照EN、DIR、PUL顺序输入，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- connect_mode： 驱动器的连接方式。根据实际连线方式设置该值：
  - 0 : 共阴极连接
  - 1 : 共阳极连接

- step : 细分控制的档位，按照实际S1~S3的打开情况按16进制设置。例如设置成16细分档时，S1~S3的状态分别为OFF、OFF、ON，则应设置该值为0x01(S1对应bit2，S2对应bit1，S3对应bit0)。

返回值

motorObj : 电机对象。

接口示例

```
motor.open([5, 12, 0], 0, 0x0);
/* 参数说明：
采用共阴极连线方式,故connect_mode值为0,
设置成32细分档,S1~S3都为OFF,即step值为0x0,
io5连EN+
io12连DIR+
io0连PUL+
*/
```

3.2.2.3 驱动电机旋转

```
motorObj.to(angle);
```

功能描述

控制电机旋转指定角度。

参数列表

angle: 电机旋转的角度。

返回值

无。

接口示例

```
motorObj.to(90);
```

3.2.3 约束

无。

3.2.4 样例

介绍

发射脉冲驱使电机顺时针移动90°，再次发射脉冲驱使电机逆时针移动90°。

例程

```
var motor = require("stepper");
var myMotor = motor.open([5, 12, 0], 0, 0x0);
myMotor.to (90);
/* 电机顺时针移动90° */
myMotor.to (-90);
/* 电机逆时针移动90° */
```

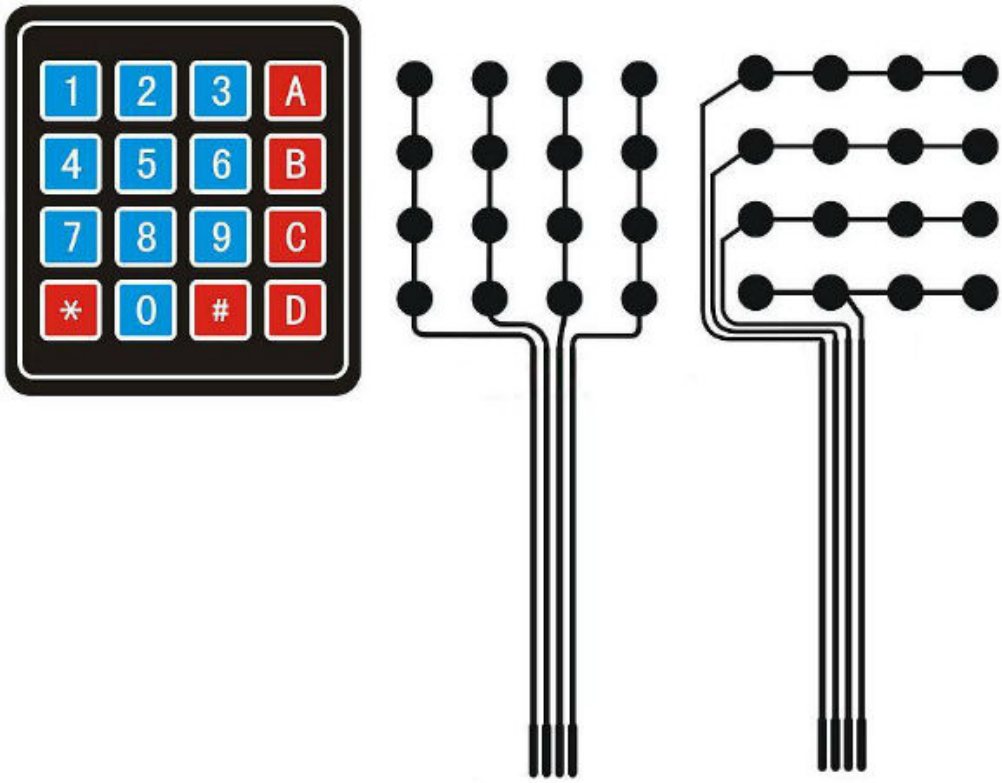
3.3 薄膜键盘

3.3.1 介绍

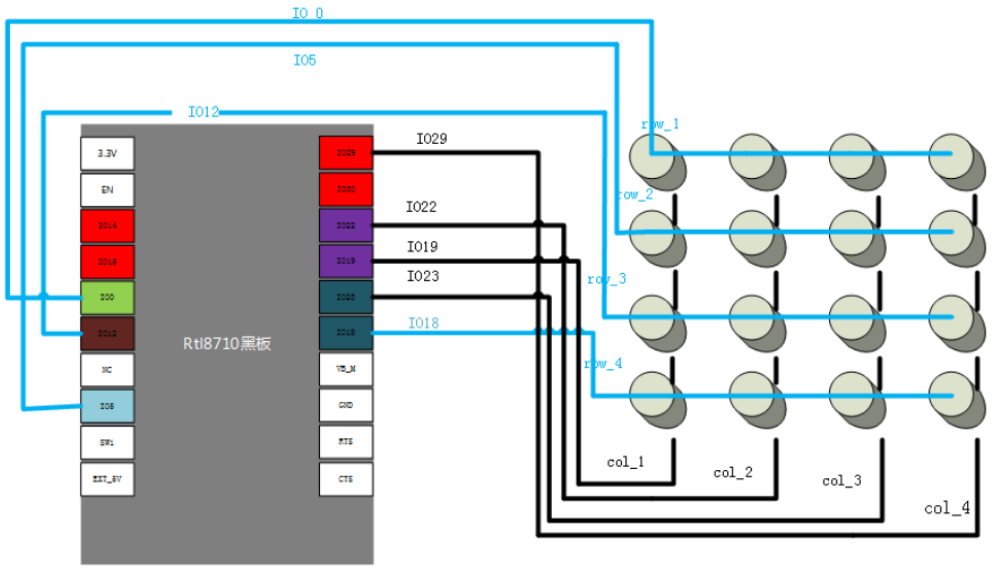
按键式控制输入设备。

3.3.1.1 模块参数

矩阵键盘是单片机外部设备中所使用的排布类似于矩阵的键盘组。在键盘中按键数量较多时，为了减少I/O口的占用，通常用矩阵键盘来替代独立按键。4×4薄膜键盘如下图。（注：此模块支持自定义n*m的矩阵键盘 (n > 0, m > 0 , 0 < n + m < 可使用的IO数)）。



3.3.1.2 连线方式



3.3.2 模块接口

3.3.2.1 引用模块

```
var keyboard = require("keyboard");
```

3.3.2.2 打开模块

```
keyboard.open(rowPinArray,colPinArray);
```

功能描述

keyboard.open打开引脚并设置相关属性，成功初始化后返回一个keyboard对象。

接口约束

无。

参数列表

- rowPinArray: 行IO引脚号数组
- colPinArray： 列IO引脚号数组 注：目前使用的IO口为：以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。 此模块支持自定义row*col的矩阵键盘 (0 < row < 7 , 0 < col < 7 , 2 < row + col < 8 (所有使用的IO数))。

返回值

打开成功时返回一个keyboard对象。

接口示例

```
var rowPin = [0, 5, 12, 18]; //键盘1~4行分别连接IO0,IO5,IO12,IO18
var colPin = [19, 22, 23, 29];//键盘1~4列分别连接IO19,IO22,IO23,IO29
var myKB = keyboard.open(rowPin, colPin);
```


3.3.2.3 设置键值

```
keyboardPin.setValue(valueArray);
```

功能描述

设置矩阵键盘不同键位所对应的值。

接口约束

无。

参数列表

- valueArray: 矩阵键盘对应键位的值。键值为一个字符，如果设置成字符串，则以字符串的第一个字符为键值。键值数组需要和打开时设置的行列数相同，否则设置失败。

返回值

无。

接口示例

```
var rowPin = [0, 5, 12, 18];
var colPin = [19, 22, 23, 29];
var myKB = keyboard.open(rowPin, colPin);
/* 键值个数为 ： rowPin * colPin */
myKB.setValue(['1','2','3','A'],// 键盘列数为4，每个数组需要存4个键值
              ['4','5','6','B'],// 键盘行数为4， 需要要设置4个数组
              ['7','8','9','C'],
              ['*','0','#','D']);
```

3.3.2.4 设置按键事件

```
keyboardPin.on(even,function);
```

功能描述

设置矩阵键盘键位按下时触发的事件。

接口约束

无。

参数列表

- even : 事件类型
  - KEY_DOWN ： 按键按下事件
- function : even发生时，触发的回调函数。如: function(data){...}，data是一个buffer，表示触发even的键值。

返回值

无。

接口示例

```
myKB.on(keyboard.KEY_DOWN, function(data){
   if (data.toString() == '1')
   {
       print ("pull down key 1");
   }
});
```


3.3.3 约束

由于IO29与IO30复用uart功能，所以在使用该模块时，如果使用IO29连接键盘，并且使用IO30，可能出现键盘功能异常。

3.3.4 样例

介绍

打印矩阵键盘键值，按下按键会打印对应的键值。

例程

```
var keyboard = require("keyboard");
var rowPin = [0, 5, 12, 18];
var colPin = [19, 22, 23, 29];
var myKeyboard = keyboard.open(rowPin, colPin);
myKeyboard.setValue(['1','2','3','A'],
                    ['4','5','6','B'],
                    ['7','8','9','C'],
                    ['*','0','#','D']);
myKB.on(keyboard.KEY_DOWN, function(data){
   if (data.toString() == '1')
   {
       print ("pull down key 1");
   }
});
```

3.4 舵机

3.4.1 介绍

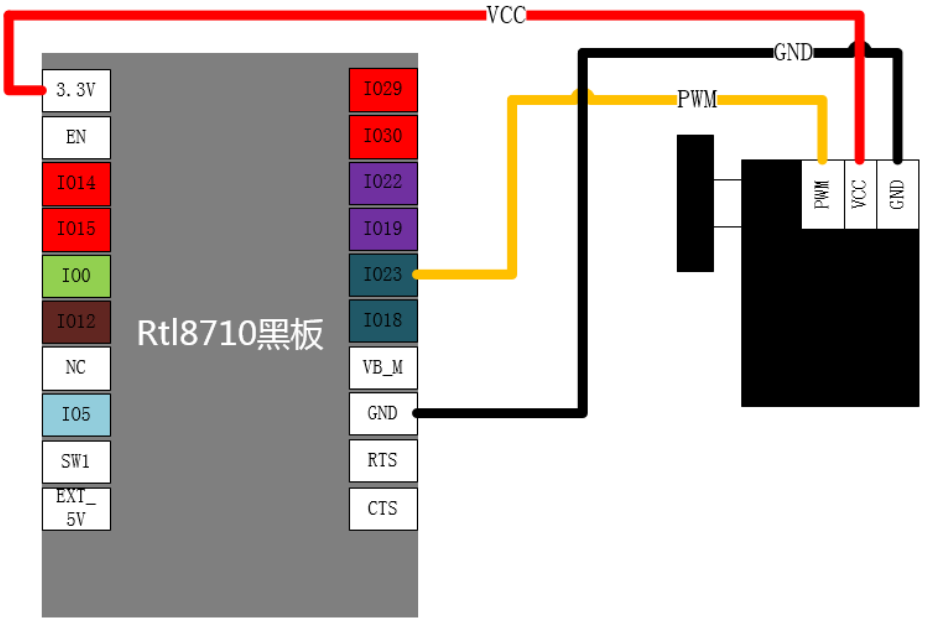
舵机通常包括一个小型直流电机，加上传感器、控制芯片、减速齿轮组，装进一体化外壳。能够通过输入信号（一般是PWM信号，也有的是数字信号）控制旋转角度。因传统上用于飞行器舵面控制而得名。

3.4.1.1 模块参数

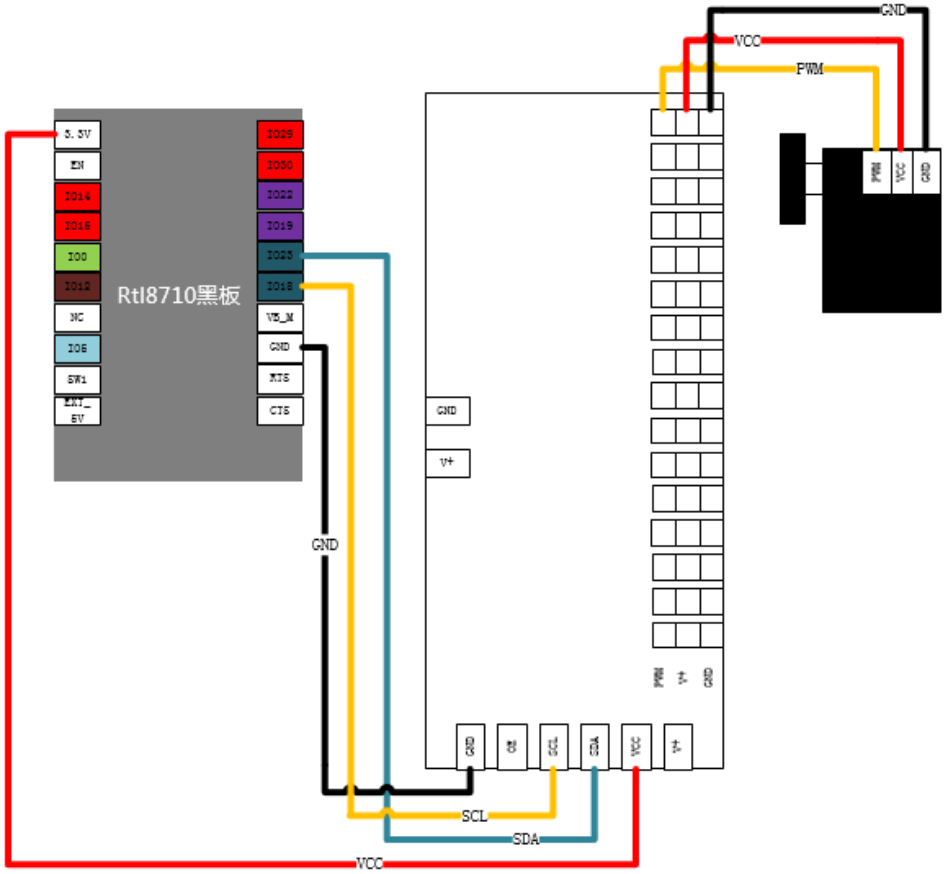
舵机模块提供驱动舵机的接口给用户，适用于工作方式与MG90S、MG996R模块相似的舵机。

该模块可以选用有PWM功能的GPIO口产生输出信号，进行单个舵机控制，也可以采用PCA9685模块转接板进行多个舵机的控制，选用PCA9685转接板进行控制时，开发板和转接板采用I2C通信方式，转接板接受开发板发出的数据并进行处理，产生多个PWM信号对多个舵机进行控制，转接板由另外的5V电源进行供电。

3.4.1.2 连线方式



或者



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	Ext_5V	VCC	电源
2	GND	GND	接地
3	GPIO	PWM	PWM控制信号

或

序号	开发板管脚	模块管脚	说明
1	Ext_5V	VCC	5V电源
2	GND	GND	接地
3	IO23/I2C1_SDA	SDA	SDA，即I2C_SDA，是I2C数据线
4	IO18/I2C1_SCL	SCL	SCL，即I2C_SCL，是I2C时钟线

3.4.2 模块接口

3.4.2.1 引用模块

```
var servo = require("servo");
```

3.4.2.2 打开模块

```
servo.open(config);
```

功能描述

对需要使用的信号输出引脚进行配置打开并设置相关属性。

接口约束

无。

参数列表

在对舵机进行控制的时候，必须要配置的为可以产生PWM引脚pin。

- pin：选用的可以产生PWM信号的引脚。

除了引脚外，其余可以根据选择进行配置的属性有id、type 、rangeMin、rangeMax、invert、startAt、center、controller、 i2cAddr、i2cNumber、model 、minDuty 、maxDuty 。

- id：用户自定义ID号，缺省时默认为空。

- type：舵机的类型，设为“servo.STANDARD”则为标准舵机，可旋转角度为为0~180°；设置为“servo.CONTINUOUS“为连续旋转舵机旋转角度为360°。缺省时为“servo.STANDARD”。目前暂只支持servo.STANDARD类型。
- rangeMin：舵机旋转角度的最小值，缺省时为0。
- rangeMax：舵机旋转角度的最大值。type为servo.STANDARD时，缺省值为180°；type为servo.CONTINUOUS时，缺省值为360°。
- invert： 翻转舵机移动方向，缺省时为false。该值为false时，舵机默认角度变大方向为正方向。（如舵机默认为逆时针旋转为正方向。设置为true后，舵机以顺时针旋转为正方向）。
- startAt：用于初始化舵机的角度，缺省时为rangeMin。创建对象过程中会改变舵机位置至该角度。
- center：如果为true, 则覆盖startAt并将舵机移动到舵机移动范围的中间值。
- controller：设置控制驱动板，缺省时直接通过IO口输出PWM控制；设置为servo.PCA9685时，通过I2C驱动PCA9685输出PWM控制舵机 , 默认addr为 0x40 。
- i2cNumber : 用于设置板子使用的i2c引脚。 取值范围为0或1；缺省时，该值为0。默认speed为1Mb/s。
  - 0: SDA引脚为IO23 (即23),SCL引脚为IO18 (即18)。
  - 1: SDA引脚为IO19 (即19),SCL引脚为IO22 (即22)。
- i2cAddr : 用于设置PCA9685控制板的7位从机地址。缺省时，该值为0x40; 注：当controller缺省时，此属性设置无效。
- model : 设置舵机的类型。servo.MG996R代表MG996R舵机，其移动0~180°对应的高电平时长为600us-2200us， MG995舵机与MG996R相同 , 控制演示如下图所示 ; servo.MG90S代表MG90S舵机，其移动0~180°对应的高电平时长为500us-2500us 。缺省时舵机类型为MG90S。
- minDuty : 用于设置舵机最小角度对应的高电平时长，单位为us。该值必须与maxDuty一起被设置才能生效。
- maxDuty :用于设置舵机最大角度对应的高电平时长，单位为us。该值必须与minDuty一起被设置才能生效。注：minDuty/maxDuty两个参数是预留给用户，用于微调舵机。

返回值

servo对象。

接口示例

使用GPIOK口产生PWM时：

```
var servo = require("servo");
var config = { pin : 5 };
var myServo = servo.open(config);
```

使用PCA9685转接板驱动舵机时：

```
var config = { pin : 5,
               rangeMin : 30,
               rangeMax : 150,
               startAt : 90,
               center : true,
               controller : servo.PCA9685,
               model : servo.MG90S
             };
var myServo = servo.open(config);
```


3.4.2.3 设置角度

```
servo.to(degrees, time, rate);
```

功能描述

将舵机移动到指定角度，单位为度，范围为0-180（或range的范围）。 如果设置了time，舵机将在这段时间内移动到该位置；如果同时设置了rate，则将在time时间内进行rate次移动到degrees角度。 如果指定的角度与当前角度相同，则不发送任何命令。

接口约束

无。

参数列表

- degrees：控制舵机移动的角度位置，范围为0~180°。
- time：控制舵机偏移角度所用的时长，单位为ms。
- rate : 转到目标位置所移动的步数，默认为1，可以根据需要设定，只能为正整数。

返回值

无。

接口示例

```
myServo.to(90);//舵机旋转90°
```

或

```
myServo.to(90, 500);//舵机在500ms内旋转90°
```

或

```
servo.to(90, 500, 10);// 舵机在500ms内用10步旋转90°
```

3.4.2.4 设置到最小角度

```
myServo.min();
```

功能描述

将舵机移动到最小值(rangeMin)。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
var servo = require("servo");
var config = { pin : 5 };
var myServo = servo.open(config);
myServo.min();// 舵机旋转到角度最小值，未设置是最小值为0。
```

或


```
var servo = require("servo");
var config = { pin : 5,
               rangeMin: 45
             };
var myServo = servo.open(config);
myServo.min();// 舵机旋转至最小值为45° 。
```

3.4.2.5 设置到最大角度

```
myServo.max();
```

功能描述

将舵机移动到最大值(rangeMax)。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
var servo = require("servo");
var config = { pin : 5 };
var myServo = servo.open(config);
myServo.max();// 舵机旋转至最大值，默认为180° 。
```

或

```
var servo = require("servo");
var config = { pin : 5,
               rangeMax: 150
             };
var myServo = servo.open(config);
myServo.max();// 舵机旋转至最大值150° °
```

3.4.2.6 设置到中间角度

```
myServo.center();
```

功能描述

将舵机移动到中间 ((rangeMax+rangeMin)/2)。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
var servo = require("servo");
var config = { pin : 5};
```



```
var myServo = servo.open(config);
myServo.center();//默认最小值为0°，最大值为180°，此时中间值为90°。
```

或

```
var servo = require("servo");
var config = { pin : 5,
               rangeMin: 40,
               rangeMax: 80
             };
var myServo = servo.open(config);
myServo.center();// (40 + 80) / 2 = 60，旋转到60°。
```

3.4.2.7 设置到初始角度

```
myServo.home();
```

功能描述

将舵机的角度移动到startAt的值。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
var servo = require("servo");
var config = { pin : 5 };
var myServo = servo.open(config);
myServo.to(90, 500);// 舵机旋转到90°。
myServo.home();// 默认startAt的角度为0°，舵机旋转到0°。
```

或

```
var servo = require("servo");
var config = { pin : 5,
               startAt : 20
             };
var myServo = servo.open(config);
myServo.to(90, 500);//  舵机旋转到90°。
myServo.home();// 设置startAt的角度为20°，舵机旋转到20°。
```

3.4.2.8 停止舵机

```
myServo.stop();
```

功能描述

在无延时情况下，停止移动的舵机。

参数列表

无。

返回值

无。

接口示例

```
var servo = require("servo");
var config = { pin : 5 };
var myServo = servo.open(config);
myServo.to(90);
myServo.stop();/* 在无延时情况下停止舵机，如果在该函数之前有延时，则此函数无效 */
```

3.4.2.9 初始化舵机数组

```
servo.initServos();
```

功能描述

初始化一个舵机数组，用于保存和控制多个舵机对象。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
var myServos = servo.initServos();
```

3.4.2.10 初始化舵机

```
myServos.multOpen (config, ...);
```

功能描述

根据config初始化舵机，并将舵机添加到舵机数组中。

接口约束

无。

参数列表

- config： 舵机的初始化信息，配置信息参照1.2。
- ... : 省略号，可一次性传入多个config，一次性打开多个舵机。

返回值

无。

接口示例

```
var servo = require("servo");
var config1 = {pin : 1, controller : servo.PCA9685 };
var config2 = {pin : 2, controller : servo.PCA9685 };
var config3 = {pin : 3, controller : servo.PCA9685 };
var myServos = servo.initServos();
myServos.multOpen (config1, config2, config3);
```

3.4.2.11 添加对象

```
myServos.addServo (servoPin,  ...);
```


功能描述

向舵机数组中添加舵机对象。

接口约束

无。

参数列表

- servoPin : 已经初始化成功的舵机对象。
- ... : 省略号，可以一次传入多个舵机对象。

返回值

无。

接口示例

```
var servo = require("servo");
var config1 = {pin : 1, controller : servo.PCA9685 };
var config2 = {pin : 2, controller : servo.PCA9685 };
var config3 = {pin : 3, controller : servo.PCA9685 };
var servo1 = servo.open(config1);
var servo2 = servo.open(config2);
var servo3 = servo.open(config3);
var myServos = servo.initServos();
myServos.addServo (servo1, servo2, servo3);
```

3.4.2.12 设置舵机组角度

```
myServos.to (degrees, time, rate);
```

功能描述

与servoPin.to(degrees, time, rate)函数相似，区别在于传入的degrees可以是一个值，也可以是一个数组。

接口约束

无。

参数说明

- degrees：控制舵机移动的位置。
  - 当degrees是一个数值时，所有舵机都会移动到该角度，范围为0~180°；
  - 当degrees是一个数组时，数组中的数值与舵机数组中的舵机一一对应。当degrees数组中的数值个数n小于舵机个数m时，舵机数组只有前n个舵机会移动，剩余舵机不移动；当n大于m时，多余的数值会被忽略，范围为0~180°。
- time：控制舵机偏移角度所用的时长，单位为ms。
- rate : 转到目标位置所移动的步数。

返回值

无。

接口示例

```
var servo = require("servo");
var tim = require ('timer');
var config1 = {pin : 1, controller : servo.PCA9685 };
var config2 = {pin : 2, controller : servo.PCA9685 };
```



```
var config3 = {pin : 3, controller : servo.PCA9685 };
var myServos = servo.initServos();
myServos.multOpen (config1, config2, config3);
myServos.to (90); /* 数组中所有舵机移动到90° */
tim.setDelay(1000);/* 延时1000ms,等待舵机移动到90° */
/* 数组中第一个舵机移动到0°,
 *  数组中第二个舵机移动到180°,
 *  数组中第三个舵机不移动,
 *  所有舵机在1000ms内，分3步移动到指定位置。
 */
myServos.to ([0, 180], 1000, 3);
```

3.4.2.13 设置舵机组动作

```
myServos.setSegment (config);
```

功能描述

设置一连串舵机组合动作，并让舵机按动作移动。

接口约束

无。

参数列表

config：为servos设置舵机动作，主要包括两个个主要属性：keyFrames和moveTimes

- operateMode：表示输入舵机角度的输入模式，有且只有0和1两种输入模式。
  - 0：表示输入舵机角度为绝对值。
  - 1：表示输入舵机的角度为上一个角度的相对值。
- keyFrames : 关键帧数组，数值数组的数组。存放舵机组每个片刻的角度，可存放多个片刻。
- moveTimes : 每个关键帧预留的移动时间。

返回值

无。

接口示例

```
var servo = require("servo");
var config1 = {pin : 1, controller : servo.PCA9685 };
var config2 = {pin : 2, controller : servo.PCA9685 };
var config3 = {pin : 3, controller : servo.PCA9685 };
var myServos = servo.initServos();
var ser1 = servo.open(config1);
var ser2 = servo.open(config2);
var ser3 = servo.open(config3);
myServos.addServo (ser1, ser2, ser3);
var segment = {
    operateMode:0,
    keyFrames:[
       [20,30,40], /* 片刻1：ser1移动至20°，ser2移动到30°， ser3移动到40° */
       [30,40,50], /* 片刻2：ser1移动至30°，ser2移动到40°， ser3移动到50° */
       [40,50,60] /* 片刻3：ser1移动至40°，ser2移动到50°， ser3移动到60° */
   ],
    moveTimes:[1000, 500, 2000]
     /* 在舵机数组从初始状态向片刻1的状态进行改变时，会预留1000ms 的时间，以保证所有舵机可达到预计角度;
    在舵机数组从片刻1的状态向片刻2的状态进行改变时，会预留500ms 的时间，以保证所有舵机可达到预计角度;
    在舵机数组从片刻2的状态向片刻3的状态进行改变时，会预留2000ms 的时间，以保证所有舵机可达到预计角度。*/
}
```



```
myServos.setSegment (segment); /* 将动作片段传递舵机数组，让舵机数组按照设定完成动作 */
```

3.4.3 约束

无。

3.4.4 样例

样例A

介绍

通过 pwm/pca9685控制板使用舵机。

例程

```
var servo = require("servo");
var tim = require ('timer');
var config = { pin : 5 };
  /* 在此配置下，要将板子的IO5连接到舵机的橙色线，3.3v连接到舵机红色线， GND连接到舵机的黑色线 */
  /*
  var config = {
      id: 0,
      pin: 10,
      type: servo.STANDARD,
      rangeMin: 20,
      rangeMax: 150,
      invert: false,      // Invert all specified positions
      startAt: 90,        // Immediately move to a degree
      center: true,
      controller : servo.PCA9685
  };
  //   在此配置下，的连线方式：
  // 1. 要将板子的IO23 连接到pca9685控制板(后简称控制板）的SCK,IO18连接到控制
  // 板的SDA， 3.3v连接控制板的VCC, GND连接控制板的GND，5V连接控制板的V+。
  // 注：控制板尽量单独供电。
  // 2. 将舵机插到控制板的10排。
  */
  var myServo = new servo.open(config);
  /* Servo API */
  // set the servo to the minimum degrees
  // defaults to 0
  myServo.min();
  tim.setDelay(1000);/* 延时1000ms,等待舵机移动到最小角度 */
  // set the servo to the maximum degrees
  // defaults to 180
  myServo.max();
  tim.setDelay(1000);/* 延时1000ms,等待舵机移动到最大角度 */
  // centers the servo to 90°
  myServo.center();
  tim.setDelay(1000);/* 延时1000ms,等待舵机移动到中间角度值 */
  //
  // Moves the servo to position by degrees
  //
  myServo.to(90);
```

样例B

介绍

通过舵机数组控制多个舵机完成自定义动作。

例程

```
  var servo = require("servo");
  var config1 = {pin : 1, controller : servo.PCA9685, startAt: 90 };
  var config2 = {pin : 2, controller : servo.PCA9685, startAt: 90 };
```



```
var config3 = {pin : 3, controller : servo.PCA9685, startAt: 90 };
var myServos = servo.initServos();
var ser1 = servo.open(config1);
var ser2 = servo.open(config2);
var ser3 = servo.open(config3);
/*
 ‘o’的位置代表舵机的位置，下图为舵机和支架构成的简易腿部的示意图。
   o  ①
   |
   o  ②
   |
③ o_
A 腿部直立
  ①   ②
   o - o
       |
    ③ o-
 B 抬起膝盖
动作A->B的过程中，各个舵机的变化：
① 由90° 逆时针旋转45° 到达135° ；
② 由90° 顺时针旋转45° 到达45° ；
③ 90° 没有变化；
*/
myServos.addServo (ser1, ser2, ser3);
var up_leg = {
  operateMode:0,
  keyFrames:[
     [90,90,90], /* 片刻1：三个舵机都为90° ，效果如动作A(腿部直立)  */
     [110,70,90], /* 片刻2：ser1移动至110° ，ser2移动到70° ，ser3不动° ，该片刻是为了防止角度移动过大而设置的过渡帧 */
     [135,45,90] /* 片刻3：三个舵机移动到B动作的预设位置 */
     ],
  moveTimes:[500, 1000, 1000]
  };  /* 将动作片段传递舵机数组，让舵机数组按照设定完成动作 */
myServos.setSegment (up_leg);
```

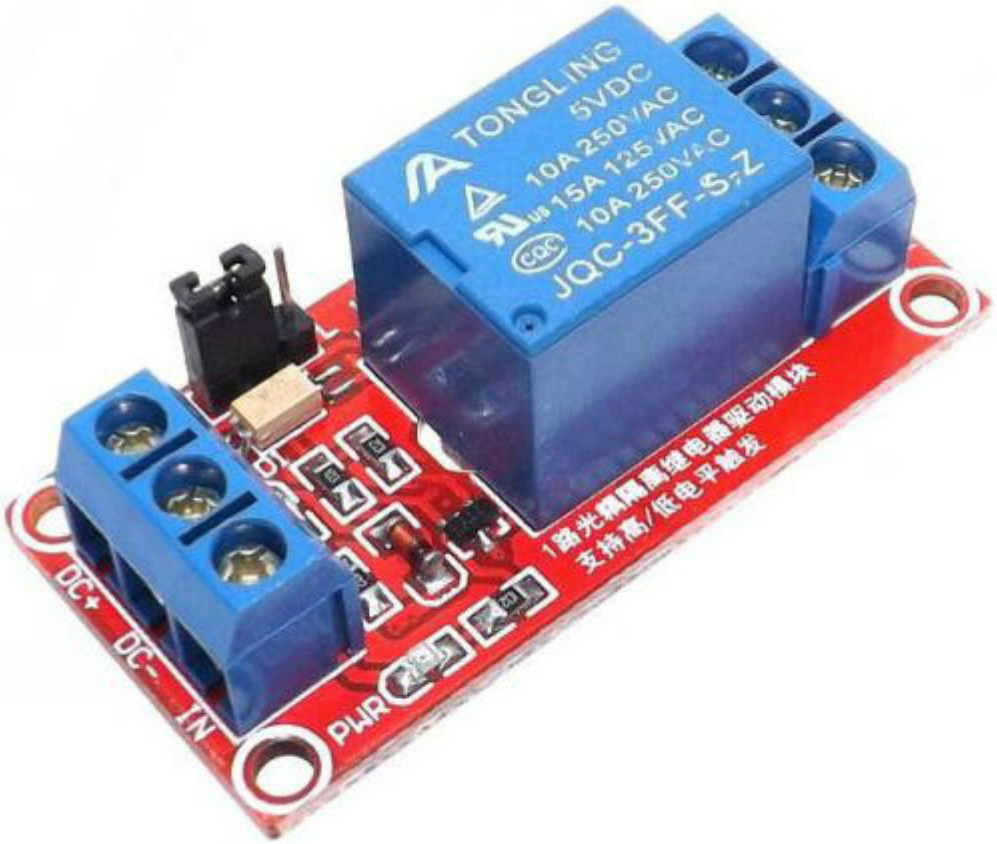
3.5 继电器

3.5.1 介绍

继电器（Relay），也称电驿，是一种电子控制器件，它具有控制系统（又称输入回路）和被控制系统（又称输出回路），通常应用于自动控制电路中，它实际上是用较小的电流去控制较大电流的一种“自动开关”。

3.5.1.1 模块参数

本模块使用的是电磁继电器，如下图所示。

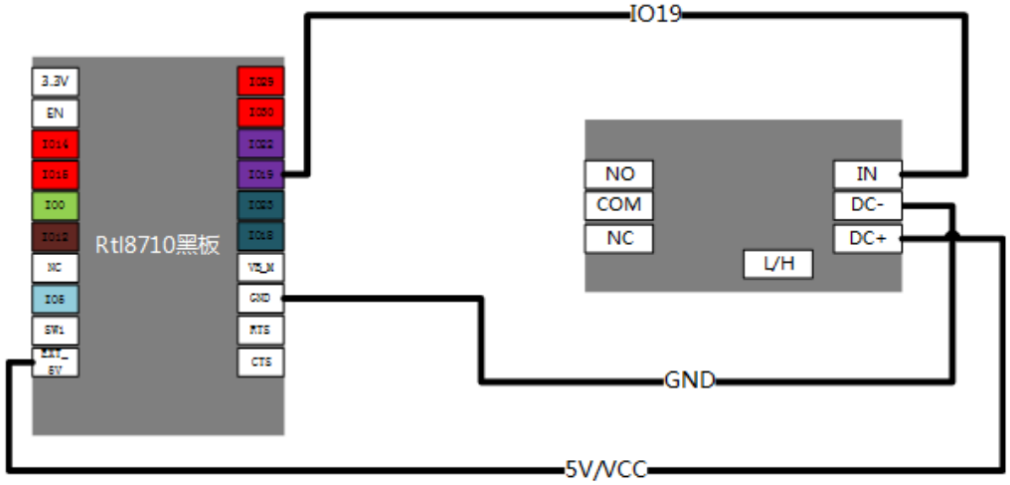


5V继电器参数列表：

参数	数值
电源	< 5V
电流	5 ~190 mA
常开接口最大负载	交流250V/10A ; 直流 30V/10A
尺寸	50 * 25 * 18.5 mm

3.5.1.2 连线方式

开发板管脚与模块管脚连线

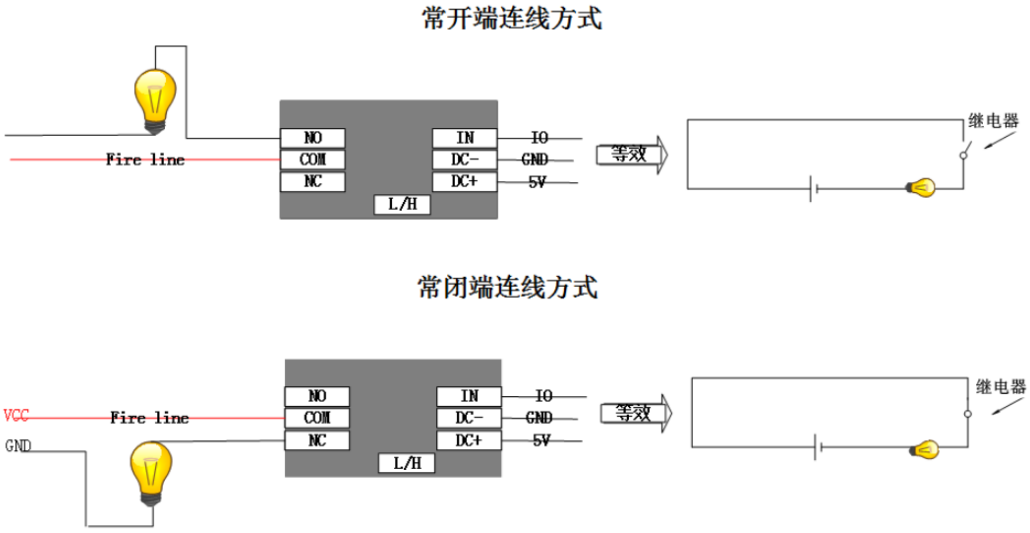


开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	Ext_5V	DC+	DC+是传感器电源，范围是 3.3 < 5V
2	GND	DC-	DC-是传感器参考地
3	IO19	IN	输入，用于继电器使能/关闭目标电路
4	————	NO	继电器常开端
5	————	COM	继电器公共端，连接目标电路的火线
6	————	NC	继电器常闭端
7	————	L/H	继电器上的电平触发选择，需要用跳线帽短接

模块与目标电路连线

继电器模块与目标电路的连线方式：



3.5.2 模块接口

3.5.2.1 引用模块

```
var relays = require("relays");
```

3.5.2.2 打开模块

```
relays.open(config);
```

功能描述

relays.open根据config配置打开引脚并设置相关属性，成功初始化后返回一个relays对象。

接口约束

无。

参数列表

- config : 配置参数，包含三个参数值： pin, work_level, connect_mode。
  - pin : gpio引脚编号，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。`
  - work_level : 低电平或高电平触发开关的模式选择。(注，设置需要和板子实际连线方式相同)
- relays.LOW : 跳线与LOW短接，低电平触发。
- relays.HIGH : 跳线与HIGH短接，高电平触发 。
  - connect_mode: 常开/闭端接法的模式设置。(注，设置需要和板子实际连线方式相同)
- relays.NORM_OPEN : 常开端的接法。
- relays.NORM_CLOSED : 常闭端的接法。

返回值

打开成功时返回一个relays对象。

接口示例

```
var config = {pin: 5, workLevel: relays.LOW, connectMode: relays.NORM_OPEN};
/* config需要与实际连线匹配，当前的配置意味着IN->IO5 ，L/H -> L, 常开接法 */
var myrelays = relays.open(config);
```

3.5.2.3 继电器使能目标电路

```
relaysPin.set_work(state);
```

功能描述

通过该接口使能或关闭控制电路。

接口约束

无。

参数列表

- state : 使能或关闭控制电路。
  - relays.OFF :关闭控制电路；
  - relays.ON : 使能控制电路。

返回值

无。

3.5.3 约束

无。

3.5.4 样例

介绍

用于自动控制电路中，它实际上是用较小的电流去控制较大电流的一种“自动开关”。

连线图

参考4.2.2.5.1模块连线，常开电路。

例程

```
var relays = require("relays");
var config = {pin: 5 , workLevel: relays.LOW, connectMode: relays.NORM_OPEN};
var myrelays = relays.open(config);
myrelays.set_state(relays.OFF);
```

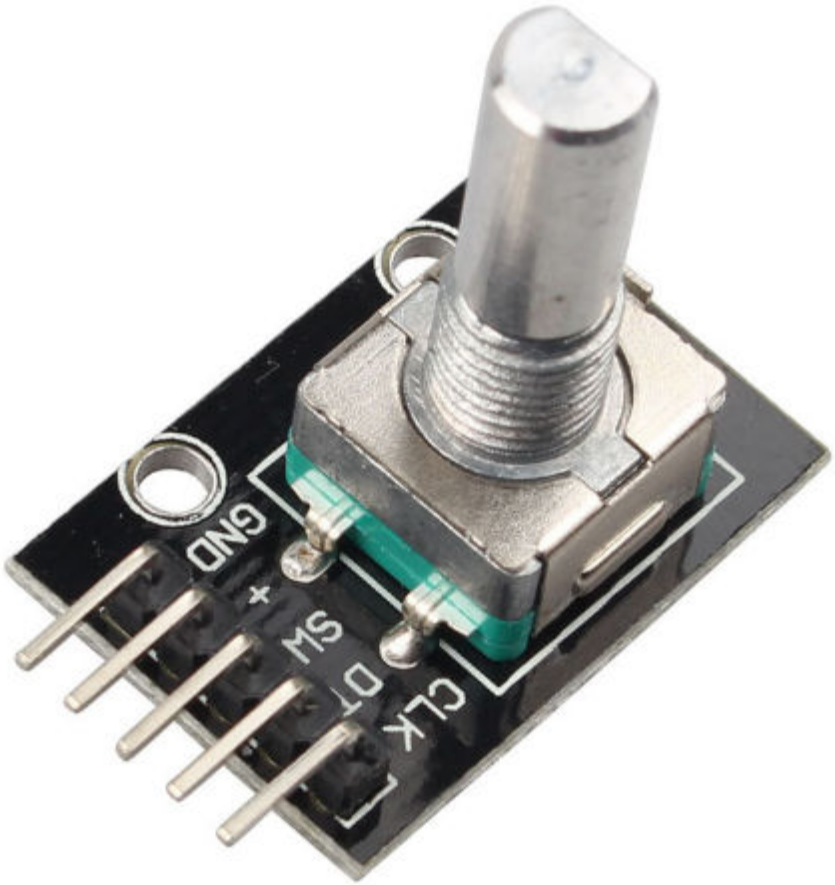
3.6 旋转编码器（ky-040）

3.6.1 介绍

旋转编码器是用来测量转速并配合PWM技术可以实现快速调速的装置，光电式旋转编码器通过光电转换，可将输出轴的角位移、角速度等机械量转换成相应的电脉冲以数字量输出（REP）。

3.6.1.1 模块参数

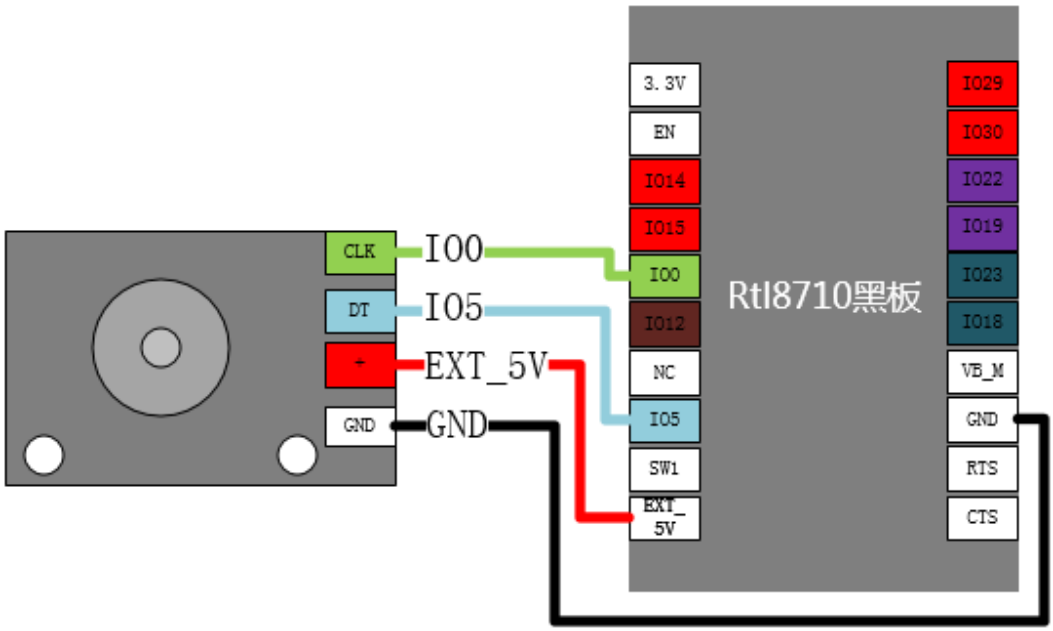
旋转编码器（ky040）可通过旋转可以计数正方向和反方向转动过程中输出脉冲的次数，旋转计数不像电位计，这种转动计数是没有限制的。配合旋转编码器上的按键，可以复位到初始状态，即从0开始计数。



旋转编码器（ky040）参数列表:

参数	数值
工作电压：	5V
一圈脉冲数	20

3.6.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	IO0	CLK	CLK上拉，旋钮逆时针旋转时CLK先输出一段低电平
			旋钮顺时针旋转时CLK后输出一段低电平
2	IO5	DT	DT上拉，旋钮顺时针旋转时DT先输出一段低电平
			旋钮逆时针旋转时DT后输出一段低电平
3	IO12	SW	SW上拉，按键按下时输出下降沿
4	Ext_5V	+	+，5伏电源输入
5	GND	GND	GND，接地

3.6.2 模块接口

3.6.2.1 引用模块

```
var ky040 = require('ky040');
```

3.6.2.2 打开模块

```
ky040.open(config)
```

功能描述

根据配置打开旋钮模块获得模块对象。

接口约束

无。

参数列表

config为一个json值，包含CLK、DT和SW，CLK:连接旋钮编码器CLK的引脚编号、DT:连接旋钮编码器DT的引脚编号、SW:连接旋钮编码器SW的引脚编号，此处的引脚编号只能是非负整数;

返回值

返回旋转编码器模块对象;

接口示例

```
var config = {CLK: 0, DT : 5, SW :12};
var ky040Port = ky040.open(config);
```

3.6.2.3 注册监听事件

```
void ky040Port.on(eventType, func, arg);
```

功能描述

监听旋钮的事件，如旋钮正旋（顺时针）、旋钮逆旋或按键按下，从而调用相应的回调函数。新增回调函数会使旧的回调函数失效。

接口约束

无。

参数列表

- event : 事件类型枚举值，ky040.CW:旋钮顺时针旋转，CWW:旋钮逆时针旋转，BTN_DOWN:按键按下;
- func：回调函数，当监听事件发生时调用该函数;
- arg：传递给回调函数的参数，只允许一个参数，如果需要传递多个参数，需要把多个参数打包在一个结构体里。如果没有参数，该接口将默认传递undefined;

返回值

无。

接口示例

```
myEncoder.on(rotary_encoder.CW, function (){  print ("编码器正旋");
});
```


3.6.3 约束

无。

3.6.4 样例

```
var ky040 = require('ky040');
var config = {CLK:0, DT:5, SW:12};
var ky040Port = ky040.open(config);
ky040Port.on(rotary_encoder.CW, function (num){
  print (num+"编码器正旋");
},1);
ky040Port.on(rotary_encoder.CCW, function (num){
  print (num+"编码器逆旋");
},2);
ky040Port.on(rotary_encoder.BTN_DOWN, function (num){
  print (num+"编码器按键按下");
},3);
```


4 显示模块接口

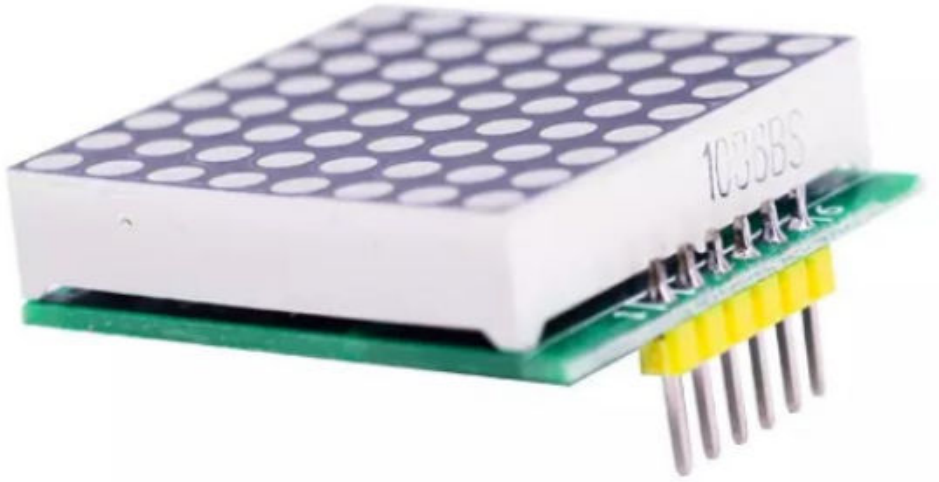
本子章节主要介绍MapleJS所支持的显示模块接口的使用。其中包括8-8点阵显示屏、LCD1602显示屏、oled显示屏、tft显示屏、墨水屏，以及像素软屏。

4.1 8-8 点阵显示屏

4.1.1 介绍

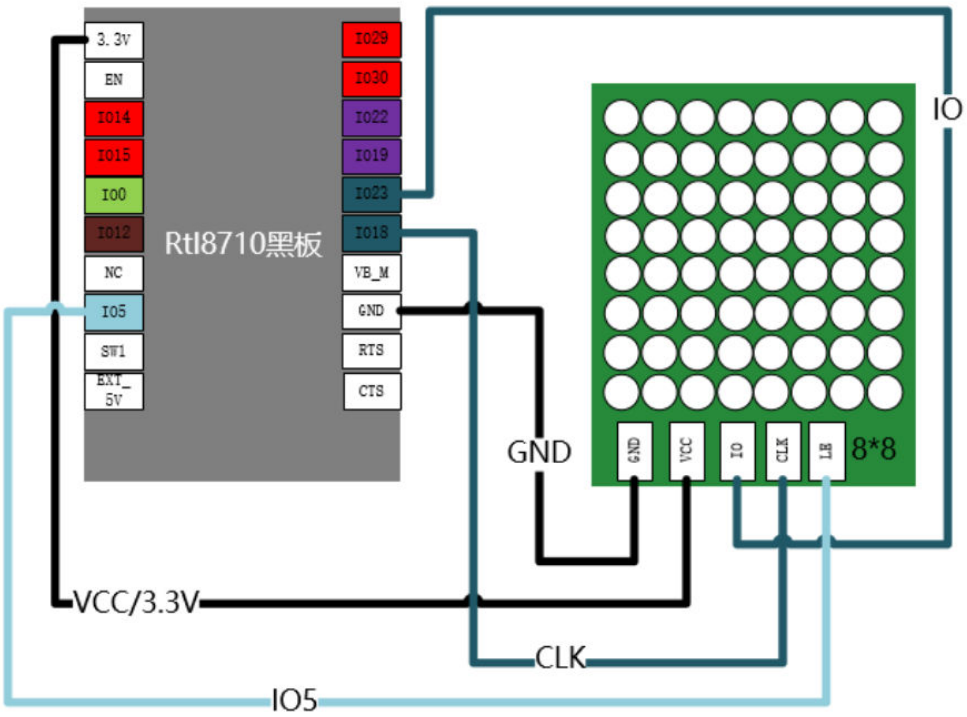
点阵显示屏格式为8乘以8，用来显示8乘以8格式的字符串。

4.1.1.1 模块参数



点阵显示屏我们采用74HC595芯片作为转换驱动芯，采用GPIO口模拟SPI的通信方式进行数据传输，转换芯片将接收到的数据进行处理，对点阵显示屏引脚进行对应操作，从而达到显示信息的目的。

4.1.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	c	VCC	电源
2	GND	GND	接地
3	IO12	CLK	使用GPIO口模拟SPI的时钟线SCLK
4	IO5	OI	使用GPIO口模拟SPI通信的数据输出线MOSI
5	IO0	LE	使用GPIO口作为输出锁存器的时钟线RCK

4.1.2 模块接口

4.1.2.1 引用模块

```
var point = require('lattice');
```

4.1.2.2 打开模块

```
point.open();
```

功能描述

点阵显示屏使用GPIO口与开发板相连，因此需要对通信使用的GPIO进行初始化。

接口约束

无。

参数列表

点阵显示屏需要配置使用的引脚为SCLK，MOSI，RCK。

- SCLK：使用GPIO口模拟SPI通信的时钟引脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- MOSI：使用GPIO口模拟SPI通信的数据输出引脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。
- RCK：以GPIO口作为输出存储器锁存时钟线，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，22，23。0引脚不推荐使用，具体参见gpio模块约束部分。

返回值

点阵显示屏对象。

接口示例

```
var config = {SCLK : 12, MOSI : 5, RCK : 0};
var lattice = point.open(config);
```

4.1.2.3 输入字符串

```
lattice.write("string", times);
```


功能描述

将传入的ASCII字符串进行逐个显示，每个字符扫描指定次数（使用该功能需要提前将相关字库文件部署至开发板中，文件部署步骤请参考相关说明文档）。

接口约束

无。

参数列表

- string：要进行显示的字符串。
- times：每个字符进行扫描的次数，每次扫描大约为8ms，最大值不应超过4294967295(32位)，且只能为正整数。

返回值

无。

接口示例

```
point.write("huawei", 500);
```

4.1.3 约束

因点阵显示屏的像素低，无法进行汉字显示，所以只能进行字符显示，显示屏显示内容无法长久保存。

4.1.4 样例

介绍

将传入的字符串"huawei"进行逐个显示，每个字符扫描指定次数：500。

例程

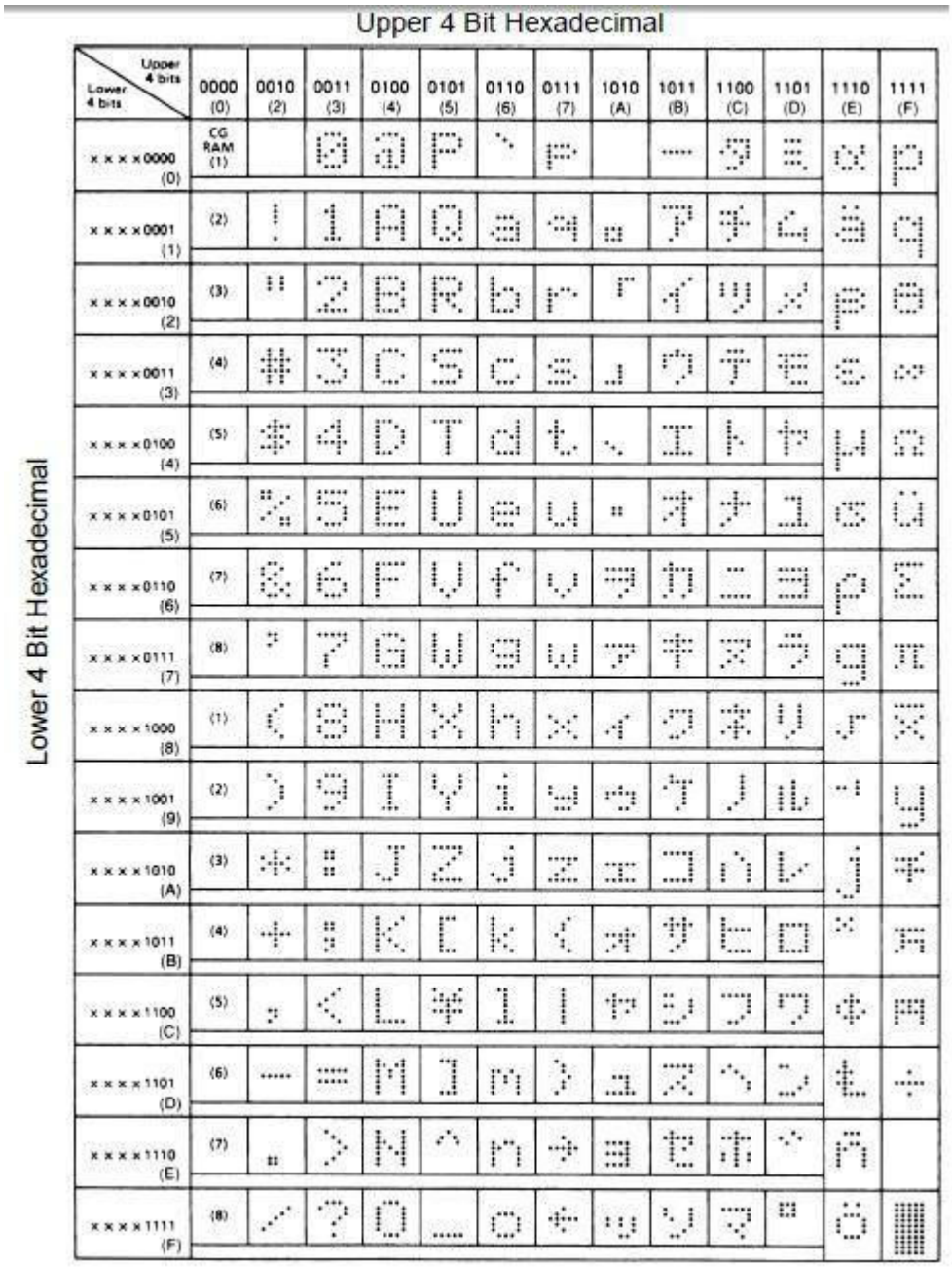
```
var lattice = require('lattice');
var config = {SCLK:0, MOSI:5, RCK:12};
var point = lattice.open(config);
point.write("huawei", 500);
```

4.2 LCD1602 显示屏

4.2.1 介绍

LCD1602是一种工业字符型液晶，能够同时显示16*02即32个字符，支持内置CGROM中的字母、数字、符号等显示，具体可参考下图。

4.2.1.1 模块参数



本模块需加装PCF8574转接板，通过I2C通信协议来驱动屏幕。

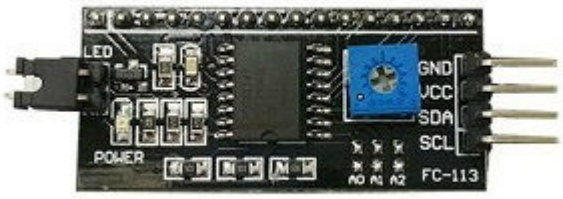
LCD1602参数列表:

参数	数值
显示容量	16×2个字符
芯片工作电压	4.5—5.5V
工作电流	2.0mA(5.0V)
块最佳工作电压	5.0V

参数	数值
字符尺寸	2.95×4.35(W×H)mm

PCF8574是CMOS电路，通过两条双向总线I2C可使大多数MCU实现远程I/O口扩展。该器件包含一个8位准双向口和一个I2C总线接口。

PCF8574功耗较低，输出锁存，具有大电流驱动能力，可直接驱动LED。此外它还拥有一条中断接线(INT)可MCU的中断逻辑相连，通过INT发送中断信号，远端I/O口不必经过总线通信便可通知MCU是否有数据从端口输入。这使得PCF8574可以作为一个单被控器。

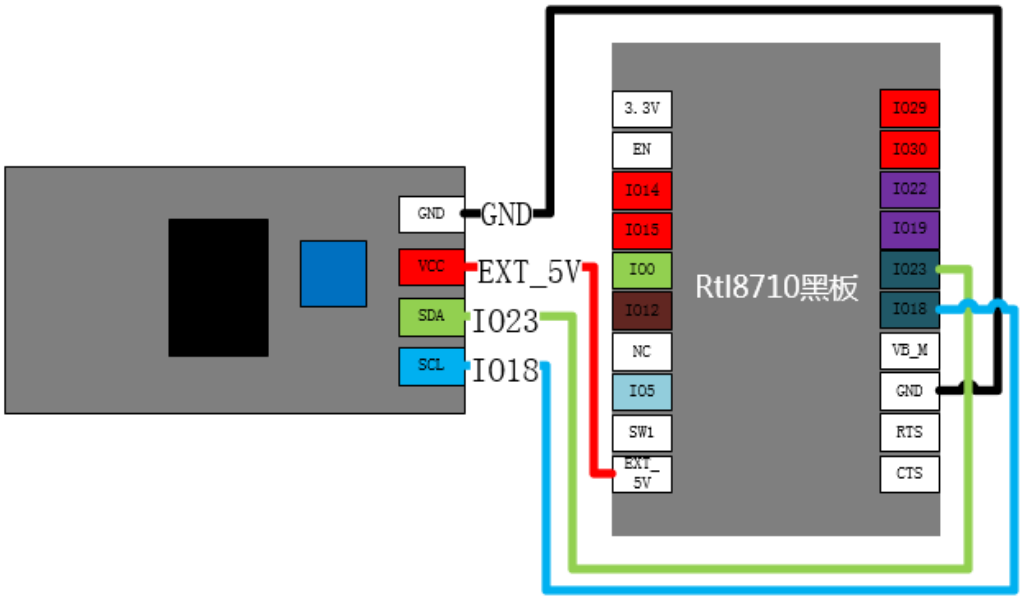


pcf8574

PCF8574T参数列表:

参数	数值
工作温度范围	-40°C to +85°C
操作电压	2.5—6.0V
低备用电流	≤10μA
时钟频率	0.1MHz

4.2.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	Ext_5V	VCC	电源
2	GND	GND	接地
3	IO23/I2C1_SDA	SDA	SDA，即I2C_SDA，是I2C数据线
4	IO18/I2C1_SCL	SCL	SCL，即I2C_SCL，是I2C时钟线

4.2.2 模块接口

4.2.2.1 引用模块

```
var lcd1602 = require('lcd1602');
```

4.2.2.2 打开模块

```
lcd.open(i2c_num);
```

功能描述

打开指定i2c引脚上的lcd1602端口。

接口约束

无。

参数列表

- i2c_num:number类型，开发板的i2c编号，可为0或者1，及其他正整数。

返回值

返回lcd1602对象。

接口示例

```
var lcd = lcd1602.open(0);
```

4.2.2.3 显示字符串

```
void lcdPort.fillText(x,y,string);
```

功能描述

在指定位置显示5*8格式字符串，需要注意1602屏幕只能显示16*2个字符，若显示过多字符将忽略超出一行边界的字符。

接口约束

无。

参数列表

- x:number类型，表示显示字符串起始位置的水平坐标，只能0~15中任意一个整数。
- y:number类型，表示显示字符串起始位置的垂直坐标，只能0~1中任意一个整数。
- string:要显示的字符串。

返回值

无。

接口示例

```
lcd.fillText(0,0,'hello');
```

4.2.2.4 清屏

```
void lcdPort.clear(void);
```

功能描述

对显示屏进行清屏处理。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
lcd.clear();
```


4.2.2.5 控制屏幕背光

```
void lcdPort.backlight(switch);
```

功能描述

打开或关闭显示屏背光。

接口约束

无。

参数列表

- switch:lcd1602开关枚举值
  - lcd1602.ON : 开
  - lcd1602.OFF : 关

返回值

无。

接口示例

```
lcd.backlight(lcd1602.ON);
```

4.2.2.6 关闭模块

```
lcdPort.close(void);
```

功能描述

关闭LCD1602模块。

参数列表

无。

返回值

无。

4.2.3 约束

无。

4.2.4 样例

介绍

使用LCD1602显示屏进行字符串开背光显示。

例程

```
var lcd = require('lcd1602');
var time = require('timer');
var lcdPort= lcd.open(0);
lcdPort.backlight(lcd.ON);
lcdPort.fillText(0, 1, 'Hello');
time.setDelay(3000);
lcdPort.fillText(1, 9, 'world!');
time.setDelay(3000);
lcdPort.clear();
```



```
time.setDelay(3000);
lcdPort.close();
```

4.3 oled 显示屏

4.3.1 介绍

4.3.1.1 模块参数

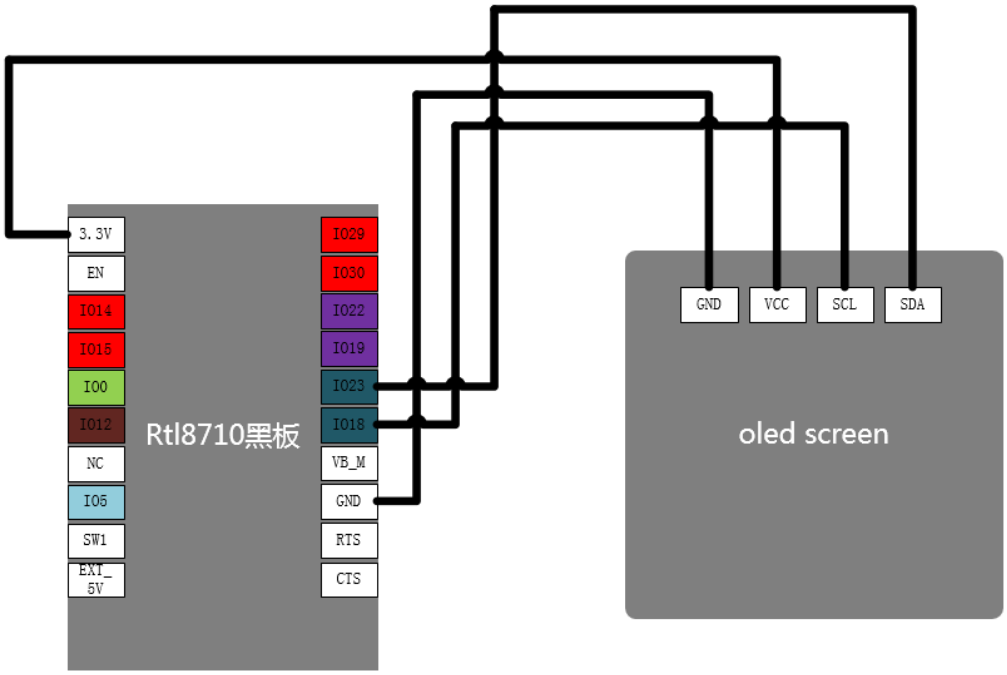
canvas module是AntJS系统基于不同屏幕的专用画图子模块，其接口与HTML5的canvas API保持兼容。本文是canvas module在oled屏上具体实现的指导说明。

oled screen具有功耗低、视角宽、响应速度快显示等优点，经常被用于货架标签、工业仪表等显示应用。

oled screen模块参数列表：

参数	数值
尺寸	0.96寸
分辨率	128*64
可视角度	>160°
功耗	0.04W
输入电压	3.3~5VDC
通信方式	IIC
字库支持	无

4.3.1.2 连线方式



序号	开发板管脚	模块管脚	说明
1	GND	GND	GND是模块的接地引脚
2	3.3V	VCC	VCC是模块的电源引脚
3	SCL	IO18	SCL是模块的IIC时钟线
3	SDA	IO23	SDA是模块的IIC数据线

4.3.2 模块接口

4.3.2.1 引用模块

```
var canvas = require("canvas");
```

4.3.2.2 打开模块

```
canvas.open(cfg);
```

功能描述

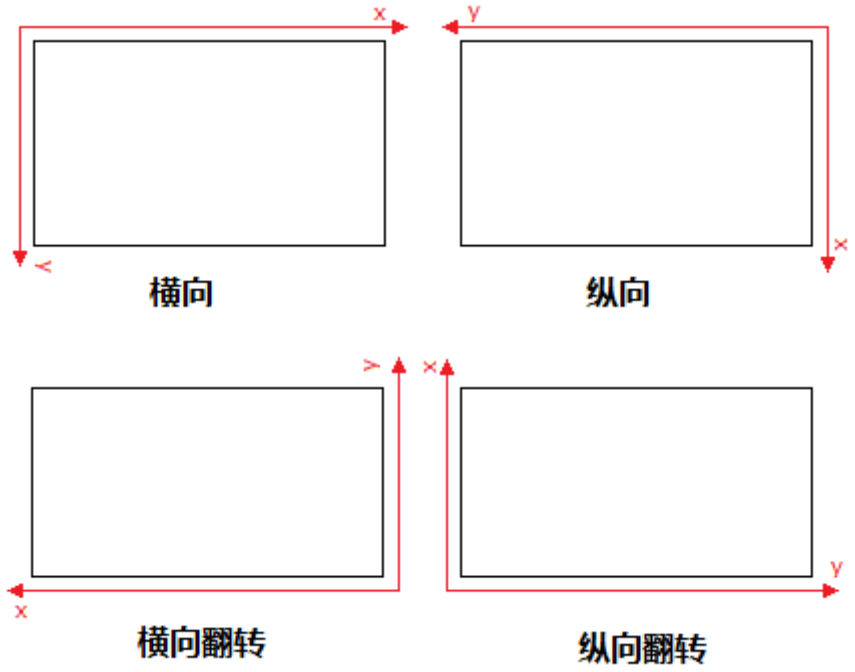
- 按照cfg的配置打开模块。

接口约束

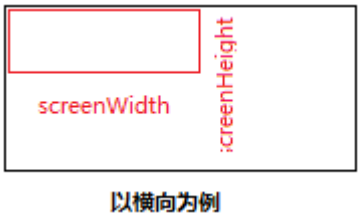
无。

参数列表

- cfg表示打开的配置参数，其中应包含：
  - screenType信息。screenType表示屏幕类型：
- 字符串“ink”表示ink屏；
- 字符串“oled”表示oled屏；
- 字符串“tft”表示tft屏；
- screenType必须指定，不可省略。
  - orientation信息。orientation表示显示方向：
- 取值选项如下：
- 1：横向
- 2：纵向
- 3：横向翻转
- 4：纵向翻转



- orientation必须显示指定，不可省略。
  - screenWidth信息。screenWidth表示自定义屏幕宽度：
- 范围：1 ≤ screenWidth ≤ 128。
- screenWidth可省略，省略时screenWidth将默认设置为128。
  - screenHeight信息。screenHeight表示自定义屏幕高度：
- 范围：1 ≤ screenHeight ≤ 64。
- screenHeight可省略，省略时screenHeight将默认设置为64。



以横向为例

- canvasPort为方法返回的端口句柄。

返回值

无。

接口示例

```
/* 显示指定自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"oled", screenWidth:64, screenHeight:32, orientation:1};
var canvas_port = canvas.open(cfg);
/* 省略自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"oled",orientation:1};
var canvas_port = canvas.open(cfg);
```

4.3.2.3 新建路径

```
canvas_port.beginPath();
```

功能描述

新建一条路径，路径一旦创建成功，图形绘制命令被指向到路径上生成路径。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.beginPath();
```

4.3.2.4 关闭路径

```
canvas_port.closePath();
```

功能描述

关闭路径，之后图形绘制命令又重新指向到上下文中。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.closePath();
```

4.3.2.5 设置画笔起点

```
canvas_port.moveTo(x, y);
```

功能描述

设置画笔的起始点坐标。

接口约束

无。

参数列表

- x为起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.moveto(0,0);
```

4.3.2.6 绘制线段

```
canvas_port.lineTo(x, y);
```

功能描述

绘制线段。

接口约束

无。

参数列表

- x为终点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为终点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.lineto(50,50);
```


4.3.2.7 绘制矩形

```
canvas_port.strokeRect(x1, y1,x2, y2);
```

功能描述

绘制矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeRect(0,0,29,29);
```

4.3.2.8 填充矩形

```
canvas_port.fillRect(x1, y1,x2, y2);
```

功能描述

填充矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillRect(0,0,29,29);
```

4.3.2.9 绘制三角形

```
canvas_port.strokeTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

绘制三角形。

接口约束

无。

参数列表

- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。
- x3为顶点3横坐标：
  - 范围：0≤x3≤(screenWidth-1)。
- y3为顶点3纵坐标：
  - 范围：0≤y3≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeTriangle(10, 10,20, 20, 30, 30);
```

4.3.2.10 填充三角形

```
canvas_port.fillTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

填充三角形。

接口约束

无。

参数列表

- x1为顶点1横坐标：

  - 范围：0 ≤ x1 ≤ (screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0 ≤ y1 ≤ (screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0 ≤ x2 ≤ (screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0 ≤ y2 ≤ (screenHeight-1)。
- x3为顶点3横坐标：
  - 范围：0 ≤ x3 ≤ (screenWidth-1)。
- y3为顶点3纵坐标：
  - 范围：0 ≤ y3 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillTriangle(10, 10, 20, 20, 30, 30);
```

4.3.2.11 绘制多边形

```
canvas_port.strokePolygon(verticesArray);
```

功能描述

绘制多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围0 ≤ x ≤ (screenWidth-1)。
  - 顶点的纵坐标范围0 ≤ y ≤ (screenHeight-1)。
  - 顶点个数3 ≤ n ≤ 10

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.strokePolygon(verticesArray1);
```

4.3.2.12 填充多边形

```
canvas_port.fillPolygon(verticesArray);
```

功能描述

填充多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围0 ≤ x ≤ (screenWidth-1)。
  - 顶点的纵坐标范围0 ≤ y ≤ (screenHeight-1)。
  - 顶点个数3 ≤ n ≤ 10

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.fillPolygon(verticesArray1);
```

4.3.2.13 绘制圆弧

```
canvas_port.arc(x, y, radius, start_angle, end_angle, anti_clockwise);
```

功能描述

绘制圆弧。

接口约束

无。

参数列表

- x为圆心横坐标：
  - 范围：0 ≤ x ≤ 128 。
- y为圆心纵坐标：
  - 范围：0 ≤ y ≤ 64 。
- radius为圆弧半径：
  - 范围：1 ≤ radius ≤ 64。
- start_angle为起始角度：
  - 范围：0 ≤ start_angle ≤ 360。
- end_angle为终止角度：
  - 范围：0 ≤ end_angle ≤ 360。
- anti_clockwise为圆弧方向：
- 0表示顺时针，1表示逆时针。

返回值

无。

接口示例

```
canvas_port.arc(50, 30, 10, 0, 135, 0);
```

4.3.2.14 填充圆形

```
canvas_port.fillCircle(x, y, radius);
```


功能描述

填充圆形。

接口约束

无。

参数列表

- x为圆心横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为圆心纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。
- radius为圆弧半径：
  - 范围：1 ≤ radius ≤ 200。

返回值

无。

接口示例

```
canvas_port.fillCircle(60,30,25);
```

4.3.2.15 绘制椭圆

```
canvas_port.strokeEllipse(x1, x2, y1, y2);
```

功能描述

绘制椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：0 ≤ x1 ≤ (screenWidth-1)。
- x2为最右侧点横坐标：
  - 范围：0 ≤ x2 ≤ (screenWidth-1)。
- y1为最上侧点纵坐标：
  - 范围：0 ≤ y1 ≤ (screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0 ≤ y2 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeEllipse(10,40,5,60);
```


4.3.2.16 填充椭圆

```
canvas_port.fillEllipse(x1, x2, y1, y2);
```

功能描述

填充椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- x2为最右侧点横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y1为最上侧点纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillEllipse(10,40,5,60);
```

4.3.2.17 显示字符

```
canvas_port.fillText(x, y, "textString", "fontString");
```

功能描述

显示字符。（使用该功能需要提前将相关字库文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- x为显示起始点横坐标：
  - 范围：0≤x≤(screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0≤y≤(screenHeight-1)。
- "textString"为待显示字符串。
- "fontString"为字体字符串：
  - "0816"表示英文字体；
  - "1616"表示中文字体。

返回值

无。

接口示例

```
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.fillText(0, 20, "一丁", "1616");
```

4.3.2.18 显示图片

```
canvas_port.drawImage("imageNameString", x, y);
```

功能描述

显示图片。（使用该功能需要提前将相关图片文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- "imageNameString"为图片文件名字符串。
  - "imageNameString"不可包含文件后缀名，例如部署的图片文件为test.pin则填入"test"。
- x为显示起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.drawImage("test", 0, 0);
```

4.3.2.19 显示轮廓

```
canvas_port.stroke();
```

功能描述

显示轮廓。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.stroke();
```


4.3.2.20 清除屏幕

```
canvas_port.clear();
```

功能描述

清除屏幕。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.clear();
```

4.3.2.21 设置线宽属性

```
canvas_port.lineWidth = settingValue;
```

功能描述

设置线宽属性。

接口约束

无。

参数列表

- settingValue为所设置的线宽值。
  - 范围：$1 \leq settingValue \leq 8$
  - 线宽值可省略，省略时线宽值将默认为1。

返回值

无。

接口示例

```
canvas_port.lineWidth = 2;
```

4.3.2.22 设置画笔颜色属性

```
canvas_port.strokeStyle = settingValue;
```

功能描述

设置画笔颜色属性。

接口约束

无。

参数列表

- settingValue为所设置的画笔颜色属性。
  - 取值：0x000000表示画笔为黑色，0xFFFFFF表示画笔为白色。
  - 画笔颜色属性可省略，省略时画笔颜色属性值将默认为0x000000（黑色）。

返回值

无。

接口示例

```
canvas_port.strokeStyle = 0x000000;
```

4.3.3 约束

- 模块接口中所涉及到的数值的属性必须是不小于零的整数，浮点数、负数等均为非法。
- 在显式指定了screenWidth及screenHeight的情况下，Canvas模块将只对指定范围内的画笔内容进行处理显示并自动忽略超出范围部分。
- screenWidth和screenHeight的设定值是相对于lineWidth设为1而言的，当lineWidth被设置为其他合法数值时，screenWidth和screenHeight所实际覆盖的范围会相应缩小，用户需自行计算并确保所绘制图形在其覆盖范围之内。

4.3.4 样例

样例A

介绍

orientation对比。

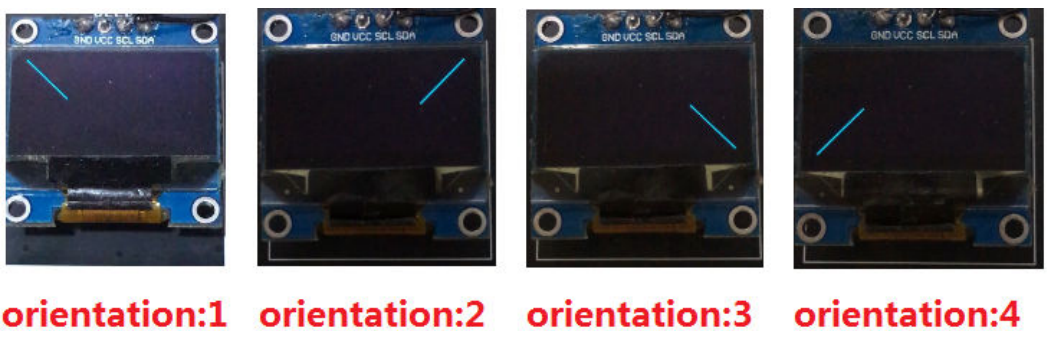
样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 4;
canvas_port.moveto(0,0);
canvas_port.lineto(30,30);
canvas_port.stroke();
canvas_port.closePath();
```

然后依次将代码模板中的cfg修改为：

```
cfg = {screenType:"oled",orientation:2};
cfg = {screenType:"oled",orientation:3};
cfg = {screenType:"oled",orientation:4};
```

样例结果



样例B

介绍

绘制线段。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.moveto(0,5);
canvas_port.lineto(50,5);
canvas_port.lineWidth = 2;
canvas_port.moveto(0,10);
canvas_port.lineto(50,10);
canvas_port.lineWidth = 4;
canvas_port.moveto(0,20);
canvas_port.lineto(50,20);
canvas_port.lineWidth = 8;
canvas_port.moveto(0,30);
canvas_port.lineto(50,30);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例C

介绍

绘制矩形。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.strokeRect(0,0,29,29);
canvas_port.lineWidth = 2;
canvas_port.strokeRect(40,0,69,29);
canvas_port.lineWidth = 4;
canvas_port.strokeRect(80,0,109,29);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例D

介绍

填充矩形。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.fillRect(0,0,29,29);
canvas_port.fillRect(40,0,69,29);
```



```
canvas_port.fillRect(80,0,109,29);
canvas_port.stroke();
canvas_port.closePath();
```

样例E

介绍

绘制圆弧。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.arc(50, 30, 10, 0, 135, 0);
canvas_port.lineWidth = 2;
canvas_port.arc(50, 30, 20, 0, 135, 0);
canvas_port.lineWidth = 4;
canvas_port.arc(50, 30, 30, 0, 135, 0);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例F

介绍

绘制圆形。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.arc(20, 30, 20, 0, 360, 0);
canvas_port.lineWidth = 2;
canvas_port.arc(60, 30, 20, 0, 360, 0);
canvas_port.lineWidth = 4;
canvas_port.arc(100, 30, 20, 0, 360, 0);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例G

介绍

填充圆形。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.fillCircle(60,30,25);
canvas_port.stroke();
canvas_port.closePath();
```


样例H

介绍

显示字符。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.fillText(0, 20, "一丁", "1616");
canvas_port.stroke();
canvas_port.closePath();
```

样例I

介绍

显示图片

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.drawImage("smile", 0, 0);
canvas_port.stroke();
canvas_port.closePath();
```

样例J

介绍

通过设置画笔颜色实现反显。

样例代码

```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0x000000;
canvas_port.fillCircle(60,30,25);
canvas_port.strokeStyle = 0xFFFFFF;
canvas_port.fillCircle(60,30,15);
canvas_port.stroke();
canvas_port.closePath();
```

样例K

介绍

清除屏幕。

样例代码

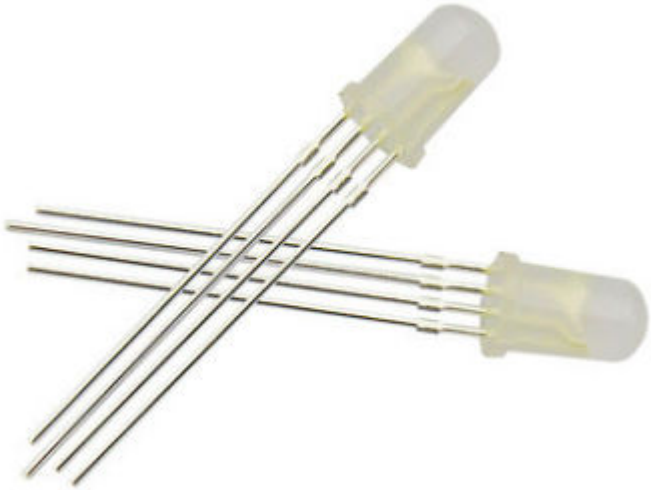
```
canvas = require("canvas");
cfg = {screenType:"oled",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.arc(30, 30, 20, 0, 360, 0);
canvas_port.clear();
canvas_port.stroke();
canvas_port.closePath();
```


4.4 三色灯

4.4.1 介绍

4.4.1.1 模块参数

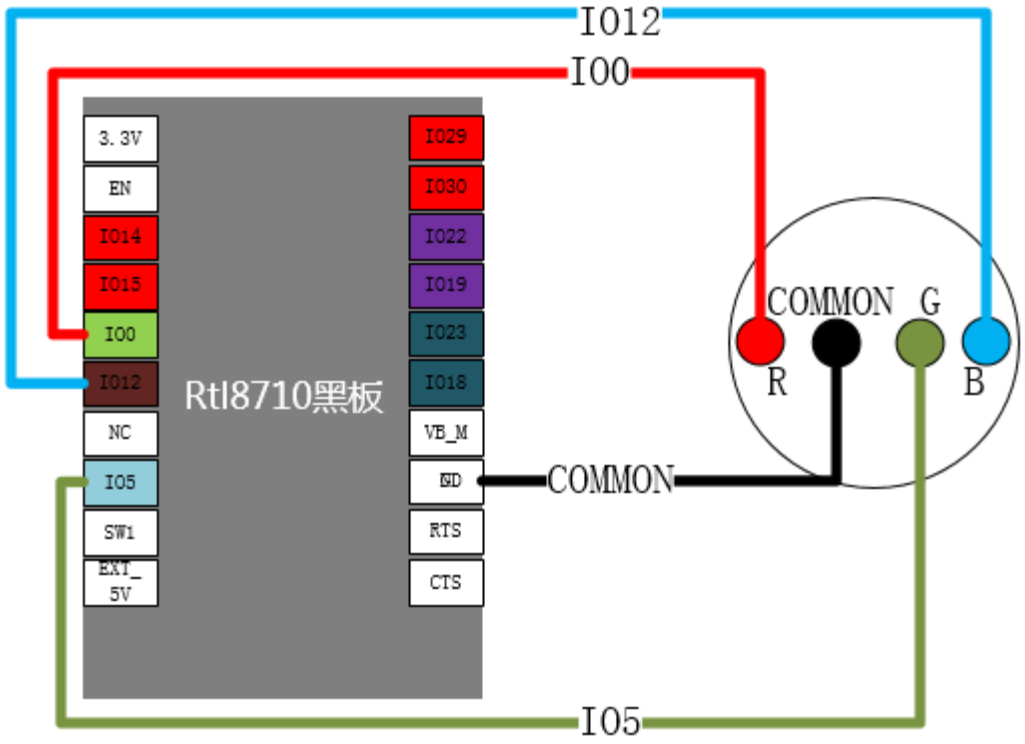
RGB三色LED由三个不同颜色的管芯组成，有共阳、共阴接法 三色LED模块可以通过控制通过PWM脉宽调制实现色彩渐变 realtek有效的PWM管脚为PWM0:IO14，PWM1:IO15， PWM2:IO0， PWM3:IO12， PWM4:IO5 本模块所有工作模式以周期为8ms的pwm脉冲驱动。



三色LED参数列表:

参数	数值
红色波长	1000-1200MCD
绿色波长	3000-5000MCD
蓝色波长	2000-3000MCD
红色电源	1.8-2.0
绿色电源	3.2-3.4
蓝色电源	3.2-3.4

4.4.1.2 连线方式



开发板管脚与模块管脚的连线表：

序号	开发板管脚	模块管脚	说明
1	IO14/PWM0	R	红色管脚
2	GND/VCC	COMMON	公共管脚，共阴或共阳
3	IO15/PWM1	G	绿色管脚
4	IO0/PWM2	B	蓝色管脚

4.4.2 模块接口

4.4.2.1 引用模块

```
var ledrgb = require('ledrgb');
```

4.4.2.2 打开模块

```
ledrgb.open(pin_r, pin_g, pin_b, type);
```

功能描述

根据配置参数打开三色LED端口。

接口约束

无。

参数列表

- pin_r、pin_g、pin_b: number类型，是红、绿、蓝连接的pwm引脚编号，如0、5、12，只能是非负整数，默认为0、5、12;
- type: 三色LED器件极性(共阴、共阳)枚举值，ledrgb.TYPE_CC: 共阴、ledrgb.TYPE_CA: 共阳，默认为共阴类型;

返回值

返回三色LED接口对象;

接口示例

```
var rgb = ledrgb.open(0, 5, 12, rgb.TYPE_CA);
```

4.4.2.3 长亮模式

```
void rgbPin.keep(RGB);
```

功能描述

指定颜色点亮三色灯并保持长亮。

接口约束

无。

参数列表

RGB: RGB配色值,number类型,(0x000000~0xFFFFFF);

返回值

无。

接口示例

```
rgb.keep(0xFFFFFF);
```

4.4.2.4 渐变模式

```
void rgbPin.change(speed);
```

功能描述

红绿蓝循环渐变，需要配合timer模块使用，使用时需注意定时器周期需大于渐变周期，否则将导致事件队列溢出，建议先给较大的定时器周期，估算出渐变周期，再对定时器周期做相应调整。 这里的定时器周期指的是timer模块接口的入参周期值，渐变周期指的是一种颜色全亮到下一种颜色全亮所需的时间。

接口约束

无。

参数列表

speed: 渐变速度，number类型，分10档，[1-10]，数值越大渐变越快;

返回值

无

接口示例

```
rgbPin.change(10);
```

4.4.2.5 呼吸灯模式

```
void rgbPin.breath(speed, RGB);
```

功能描述

以固定颜色进行周期性呼吸，需要配合timer模块使用，使用时需注意定时器周期需大于呼吸周期，否则将导致事件队列溢出，建议先给较大的定时器周期，估算出呼吸周期，再对定时器周期做相应调整。 这里的定时器周期指的是timer模块接口的入参周期值，呼吸周期指的是一次呼吸所需的时间，也就是呼吸灯从灭到全亮再到全灭所需的时间。

接口约束

无。

参数列表

- speed: 呼吸速度，number类型，分10档[1-10]，数值越大,呼吸越快;
- RGB: RGB配色值(0x000000~0xFFFFFF);-

返回值

无。

接口示例

```
rgbPin.breath(5,0xAABBCC);
```

4.4.2.6 关闭三色 LED

```
void rgbPin.close(void);
```

功能描述

熄灭LED并关闭三色LED端口。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
rgb.close();
```

4.4.3 约束

无。

4.4.4 样例

样例A

介绍

打开共阴三色LED，设置rgb三色引脚分别为IO14、IO15、IO0，保持白色显示。

例程

```
var rgb = require('ledrgb');
var rgbPin = rgb.open(14, 15, 0, rgb.TYPE_CC);
rgbPin.keep(0xFFFFFF);
```

样例B

介绍

打开共阳三色LED，设置rgb三色引脚分别为IO0，IO5，IO12，运行循环渐变模式，渐变速度为最快速度10档，周期3秒。

例程

```
var rgb = require('ledrgb');
var time = require('timer');
var rgbPin = rgb.open(0, 5, 12, rgb.TYPE_CA);
var timeID = time.setInterval (function () {
    rgbPin.change(10);
}, 3000);
```

样例C

介绍

打开共阴三色LED，设置rgb三色引脚分别为IO0，IO5，IO12，运行呼吸灯模式，颜色为红色，呼吸速度为5档，周期5秒 。

例程

```
var rgb = require('ledrgb');
var time = require('timer');
var rgbPin = rgb.open(0, 5, 12, rgb.TYPE_CC);
var timeID = time.setInterval (function () {
    rgbPin.breath(5, 0xFF0000);
}, 5000);
```

4.5 tft 显示屏

4.5.1 介绍

4.5.1.1 模块参数

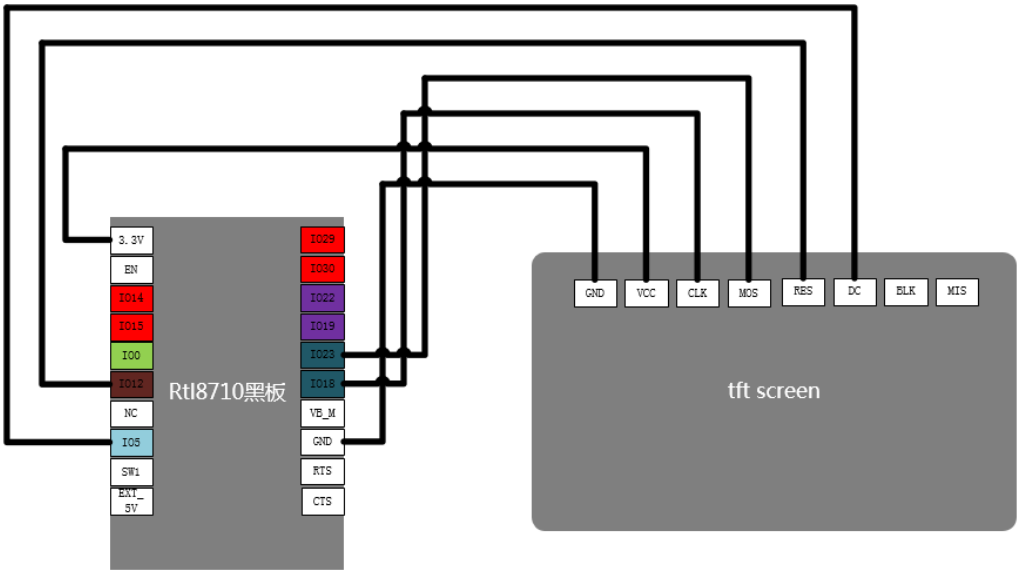
canvas module是AntJS系统基于不同屏幕的专用画图子模块，其接口与HTML5的canvas API保持兼容。本文是canvas module在tft屏上具体实现的指导说明。

tft screen具有功耗低，可视角度大等优点，常用于电子产品等显示应用。

tft screen模块参数列表：

参数	数值
尺寸	3.5寸
控制芯片	ILI9486
分辨率	320 × 480
显示区域	48.96mm × 73.44mm
接口类型	SPI
显示颜色	全彩色
像素点大小	0.153mm × 0.153mm
工作温度	-20 ~ 70℃
工作电压	3.3VDC

4.5.1.2 连线方式



序号	开发板管脚	模块管脚	说明
1	GND	GND	GND是模块的接地引脚
2	3.3V	VCC	VCC是模块的电源引脚
3	IO18	CLK	CLK是模块的SPI通信SCK引脚
4	IO23	MOS	MOS是模块的SPI通信MOSI引脚

5	IO12	RES	RES是模块的复位
6	IO5	DC	DC是模块的寄存器/数据选择引脚

4.5.2 模块接口

4.5.2.1 引用模块

```
var canvas = require("canvas");
```

4.5.2.2 打开模块

```
canvas.open(cfg);
```

功能描述

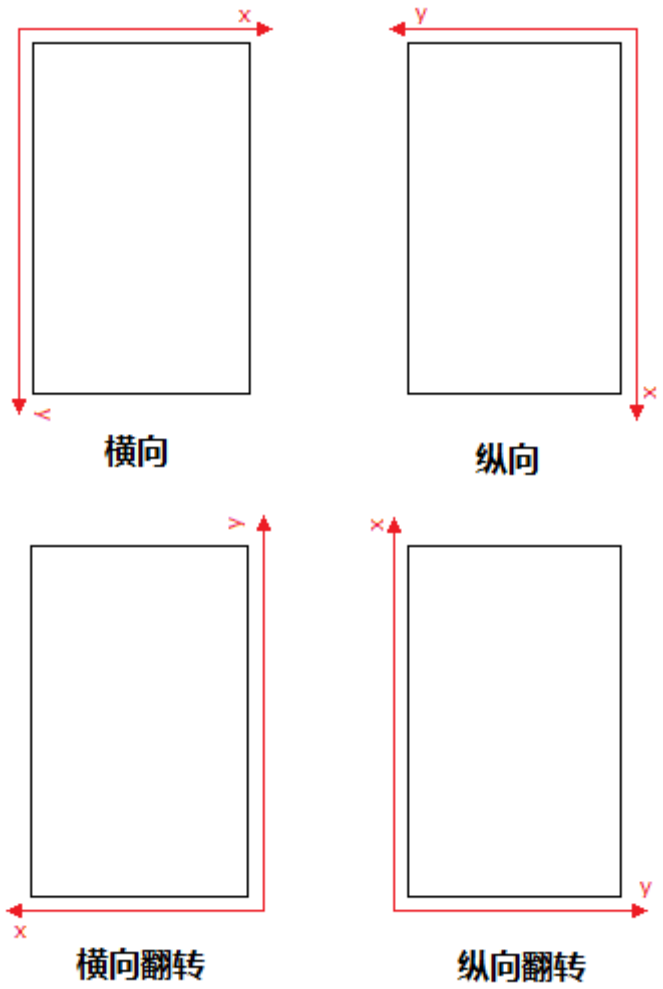
- 按照cfg的配置打开模块。

接口约束

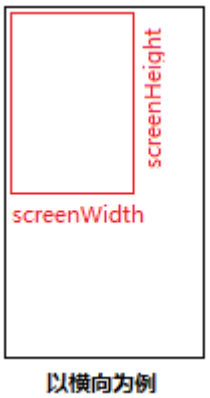
无。

参数列表

- cfg表示打开的配置参数，其中应包含：
  - screenType信息。screenType表示屏幕类型：
- 字符串“ink”表示ink屏；
- 字符串“oled”表示oled屏；
- 字符串“tft”表示tft屏；
- screenType必须指定，不可省略。
  - orientation信息。orientation表示显示方向：
- 取值选项如下：
- 1：横向
- 2：纵向
- 3：横向翻转
- 4：纵向翻转



- orientation必须显示指定，不可省略。
  - screenWidth信息。screenWidth表示自定义屏幕宽度：
- 范围：1 ≤ screenWidth ≤ 320。
- screenWidth可省略，省略时screenWidth将默认设置为320。
  - screenHeight信息。screenHeight表示自定义屏幕高度：
- 范围：1 ≤ screenHeight ≤ 480。
- screenHeight可省略，省略时screenHeight将默认设置为480。



- canvasPort为方法返回的端口句柄。

返回值

无。

接口示例

```
/* 显示指定自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"tft", screenWidth:64, screenHeight:32, orientation:1};
var canvas_port = canvas.open(cfg);
/* 省略自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"tft", orientation:1};
var canvas_port = canvas.open(cfg);
```

4.5.2.3 新建路径

```
canvas_port.beginPath();
```

功能描述

新建一条路径，路径一旦创建成功，图形绘制命令被指向到路径上生成路径。

接口约束

无。

参数列表

无。

接口示例

```
canvas_port.beginPath();
```

4.5.2.4 关闭路径

```
canvas_port.closePath();
```

功能描述

关闭路径，之后图形绘制命令又重新指向到上下文中。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.closePath();
```

4.5.2.5 设置画笔起点

```
canvas_port.moveTo(x, y);
```

功能描述

设置画笔起始点坐标。

接口约束

无。

参数列表

- x为起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.moveto(0,0);
```

4.5.2.6 绘制线段

```
canvas_port.lineTo(x, y);
```

功能描述

绘制线段。

接口约束

无。

参数列表

- x为终点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为终点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.lineto(100,100);
```

4.5.2.7 绘制矩形

```
canvas_port.strokeRect(x1, y1, x2, y2);
```

功能描述

绘制矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeRect(0,0,29,29);
```

4.5.2.8 填充矩形

```
canvas_port.fillRect(x1, y1, x2, y2);
```

功能描述

填充矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillRect(0,0,29,29);
```

4.5.2.9 绘制三角形

```
canvas_port.strokeTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

绘制三角形。

接口约束

无。

参数列表

- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。
- x3为顶点3横坐标：
  - 范围：0≤x3≤(screenWidth-1)。
- y3为顶点3纵坐标：
  - 范围：0≤y3≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeTriangle(10, 10,20, 20, 30, 30);
```

4.5.2.10 填充三角形

```
canvas_port.fillTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

填充三角形。

接口约束

无。

参数列表*

- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。
- x3为顶点3横坐标：

- 范围：0 ≤ x3 ≤ (screenWidth-1)。

- y3为顶点3纵坐标：
  - 范围：0 ≤ y3 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillTriangle(10, 10,20, 20, 30, 30);
```

4.5.2.11 绘制多边形

```
canvas_port.strokePolygon(verticesArray);
```

功能描述

绘制多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围0 ≤ x ≤ (screenWidth-1)。
  - 顶点的纵坐标范围0 ≤ y ≤ (screenHeight-1)。
  - 顶点个数3 ≤ n ≤ 10

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.strokePolygon(verticesArray1);
```

4.5.2.12 填充多边形

```
canvas_port.fillPolygon(verticesArray);
```

功能描述

填充多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围0 ≤ x ≤ (screenWidth-1)。
  - 顶点的纵坐标范围0 ≤ y ≤ (screenHeight-1)。
  - 顶点个数3 ≤ n ≤ 10

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.fillPolygon(verticesArray1);
```

4.5.2.13 绘制圆弧

```
canvas_port.arc(x, y, radius, start_angle, end_angle, anti_clockwise);
```

功能描述

绘制圆弧。

接口约束

无。

参数列表

- x为圆心横坐标：
  - 范围：0 ≤ x ≤ 320。
- y为圆心纵坐标：
  - 范围：0 ≤ y ≤ 480。
- radius为圆弧半径：
  - 范围：1 ≤ radius ≤ 320。
- start_angle为起始角度：
  - 范围：0 ≤ start_angle ≤ 360。
- end_angle为终止角度：
  - 范围：0 ≤ end_angle ≤ 360。
- anti_clockwise为圆弧方向：
- 0表示顺时针，1表示逆时针。

返回值

无。

接口示例

```
canvas_port.arc(50, 30, 10, 0, 135, 0);
```

4.5.2.14 填充圆形

```
canvas_port.fillCircle(x, y, radius);
```

功能描述

填充圆形。

接口约束

无。

参数列表

- x为圆心横坐标：

- 范围：0 ≤ x ≤ (screenWidth-1)。
- y为圆心纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。
- radius为圆弧半径：
  - 范围：1 ≤ radius ≤ 320。

返回值

无。

接口示例

```
canvas_port.fillCircle(60,30,25);
```

4.5.2.15 绘制椭圆

```
canvas_port.strokeEllipse(x1, x2, y1, y2);
```

功能描述

绘制椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：0 ≤ x1 ≤ (screenWidth-1)。
- x2为最右侧点横坐标：
  - 范围：0 ≤ x2 ≤ (screenWidth-1)。
- y1为最上侧点纵坐标：
  - 范围：0 ≤ y1 ≤ (screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0 ≤ y2 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeEllipse(10,40,5,60);
```

4.5.2.16 填充椭圆

```
canvas_port.fillEllipse(x1, x2, y1, y2);
```

功能描述

填充椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- x2为最右侧点横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y1为最上侧点纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillEllipse(10,40,5,60);
```

4.5.2.17 显示字符

```
canvas_port.fillText(x, y, "textString", "fontString");
```

功能描述

显示字符。（使用该功能需要提前将相关字库文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- x为显示起始点横坐标：
  - 范围：0≤x≤(screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0≤y≤(screenHeight-1)。
- "textString"为待显示字符串。
- "fontString"为字体字符串：
  - "0816"表示英文字体；
  - "1616"表示中文字体。

返回值

无。

接口示例

```
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.fillText(0, 20, "一丁", "1616");
```

4.5.2.18 绘制条形码

```
canvas_port.barcode(x, y, "codeNumberString");
```

功能描述

绘制条形码。

接口约束

无。

参数列表

- x为显示起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。
- "codeNumberString"为待生成条形码的数字字符串。
  - "codeNumberString"的内容应符合条形码规范，是一个12位数字。

返回值

无。

接口示例

```
canvas_port.barcode(50, 50, "692116859353");
```

4.5.2.19 显示图片

```
canvas_port.drawImage("imageNameString", x, y);
```

功能描述

显示图片。（使用该功能需要提前将相关图片文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- "imageNameString"为图片文件名字符串。
  - "imageNameString"不可包含文件后缀名，例如部署的图片文件为test.pin则填入"test"。
- x为显示起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.drawImage("test", 0, 0);
```

4.5.2.20 显示轮廓

```
canvas_port.stroke();
```

功能描述

显示轮廓。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.stroke();
```

4.5.2.21 清除屏幕

```
canvas_port.clear();
```

功能描述

清除屏幕。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.clear();
```

4.5.2.22 设置线宽属性

```
canvas_port.lineWidth = settingValue;
```

功能描述

设置线宽属性。

接口约束

无。

参数列表

- settingValue为所设置的线宽值。
  - 范围：$1 \leq settingValue \leq 8$
  - 线宽值可省略，省略时线宽值将默认为1。

返回值

无。

接口示例


```
canvas_port.lineWidth = 2;
```

4.5.2.23 设置画笔颜色属性

```
canvas_port.strokeStyle = settingValue;
```

功能描述

设置画笔颜色属性。

接口约束

无。

参数列表

- settingValue为所设置的画笔颜色属性。
  - 取值：0x000000 ≤ settingValue ≤ 0xFFFFFF。
  - 画笔颜色属性可省略，省略时画笔颜色属性值将默认为0x000000（黑色）。

返回值

无。

接口示例

```
canvas_port.strokeStyle = 0x000000;
```

4.5.3 约束

1. 模块接口中所涉及到的数值的属性必须是不小于零的整数，浮点数、负数等均为非法。

2. 在显式指定了screenWidth及screenHeight的情况下，Canvas模块将只对指定范围内的画笔内容进行处理显示并自动忽略超出范围部分。

3. screenWidth和screenHeight的设定值是相对于lineWidth设为1而言的，当lineWidth被设置为其他合法数值时，screenWidth和screenHeight所实际覆盖的范围会相应缩小，用户需自行计算并确保所绘制图形在其覆盖范围之内。

4.5.4 样例

样例A

介绍

初始化。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
```

样例B

介绍

orientation对比。

例程

```
canvas = require("canvas");
```

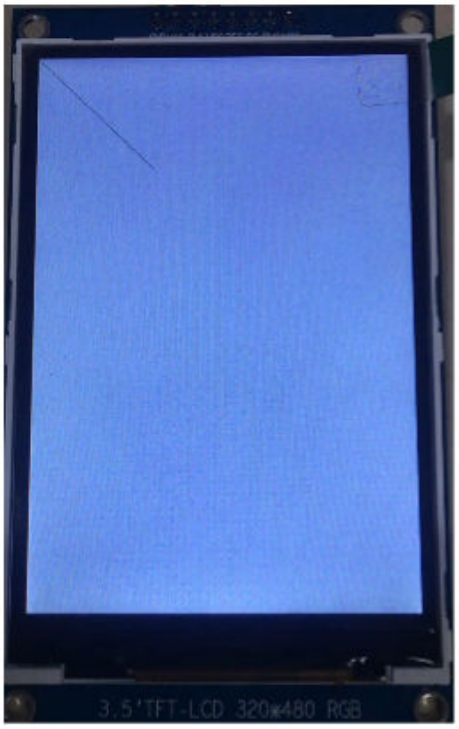


```
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.moveto(0,0);
canvas_port.lineto(100,100);
canvas_port.stroke();
canvas_port.closePath();
```

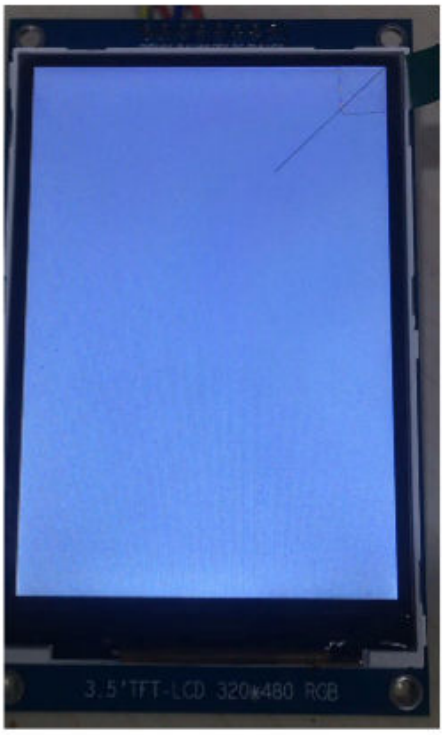
然后依次将代码模板中的cfg修改为：

```
cfg = {screenType:"tft",orientation:2};
cfg = {screenType:"tft",orientation:3};
cfg = {screenType:"tft",orientation:4};
```

样例结果



orientation:1



orientation:2

orientation:3

orientation:4

样例C

介绍

绘制线段。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.strokeStyle = 0x000000;
canvas_port.moveto(0,0);
canvas_port.lineto(50,0);
canvas_port.lineWidth = 2;
canvas_port.strokeStyle = 0xFF0000;
canvas_port.moveto(0,10);
canvas_port.lineto(50,10);
canvas_port.lineWidth = 4;
canvas_port.strokeStyle = 0x00FF00;
canvas_port.moveto(0,20);
canvas_port.lineto(50,20);
canvas_port.lineWidth = 8;
canvas_port.strokeStyle = 0x0000FF;
canvas_port.moveto(0,30);
canvas_port.lineto(50,30);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。 代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例D

介绍

绘制矩形。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.strokeStyle = 0xFF0000;
canvas_port.strokeRect(0,0,29,29);
canvas_port.lineWidth = 2;
canvas_port.strokeStyle = 0x00FF00;
canvas_port.strokeRect(40,0,69,29);
canvas_port.lineWidth = 4;
canvas_port.strokeStyle = 0x0000FF;
canvas_port.strokeRect(80,0,109,29);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。 代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例E

介绍

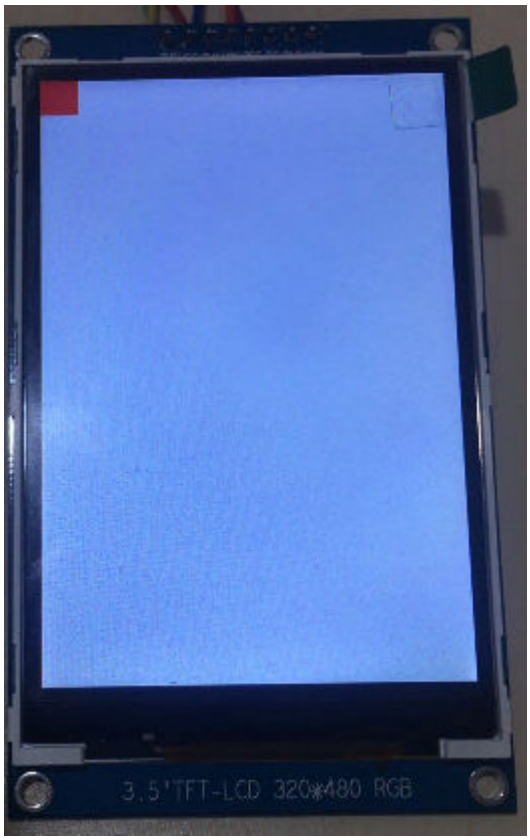
填充矩形。

例程


```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0xFF0000;
canvas_port.fillRect(0,0,29,29);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例结果



样例F

介绍

绘制圆弧。

例程

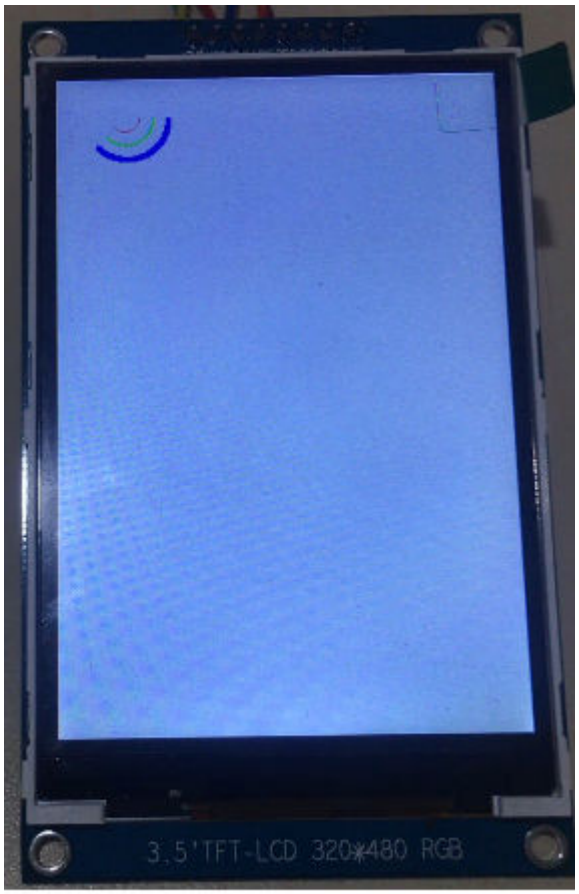
```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.strokeStyle = 0xFF0000;
canvas_port.arc(50, 30, 10, 0, 135, 0);
canvas_port.lineWidth = 2;
canvas_port.strokeStyle = 0x00FF00;
canvas_port.arc(50, 30, 20, 0, 135, 0);
canvas_port.lineWidth = 4;
canvas_port.strokeStyle = 0x0000FF;
canvas_port.arc(50, 30, 30, 0, 135, 0);
```



```
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。 代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例结果



顺时针

样例G

介绍

显示字符。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0xFF0000;
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.strokeStyle = 0x0000FF;
canvas_port.fillText(0, 20, "一丁", "1616");
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例结果

样例H

介绍

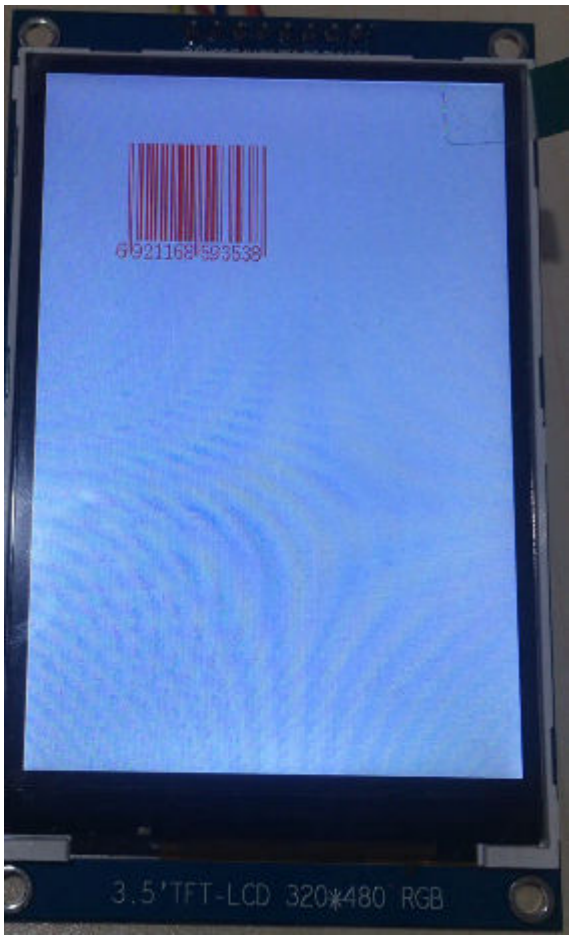
绘制条形码。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0xFF0000;
canvas_port.barcode(50, 50, "692116859353");
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.strokeStyle来调整颜色，颜色的取值范围为0x000000 ≤ strokeStyle ≤ 0xFFFFFF。

样例结果



样例I

介绍

显示图片。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.drawImage("huawei", 0, 0);
canvas_port.stroke();
canvas_port.closePath();
```

样例结果

样例J

介绍

通过设置画笔颜色实现反显。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0xFF0000;
canvas_port.fillCircle(60,30,25);
canvas_port.strokeStyle = 0xFFFFFF;
canvas_port.fillCircle(60,30,15);
canvas_port.stroke();
canvas_port.closePath();
```

样例K

介绍

清除屏幕。

例程

```
canvas = require("canvas");
cfg = {screenType:"tft",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.arc(30, 30, 20, 0, 360, 0);
canvas_port.strokeStyle = 0xFFFFFF; /*清屏前将色彩设置为白色*/
canvas_port.clear();
canvas_port.stroke();
canvas_port.closePath();
```

4.6 墨水屏

4.6.1 介绍

4.6.1.1 模块参数

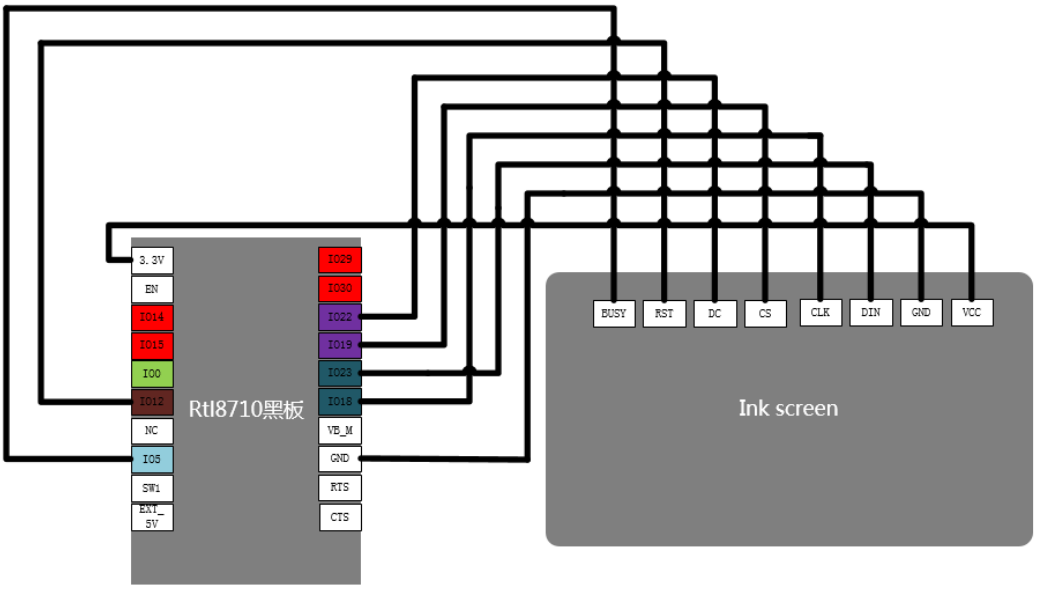
canvas module是AntJS系统基于不同屏幕的专用画图子模块，其接口与HTML5的canvas API保持兼容。本文是canvas module在ink屏上使用的指导说明。

ink screen具有功耗低、视角宽、阳光直射下仍可清晰显示等优点，经常被用于货架标签、工业仪表等显示应用。

ink screen模块参数列表：

参数	数值
工作电压	3.3V
通信接口	SPI
外形尺寸	48mm
显示尺寸	27.6mm × 27.6mm
点距	0.138 × 0.138
分辨率	200 × 200
刷新功耗	26.4mV
待机功耗	<0.17mV
可视角度	>170°

4.6.1.2 连线方式



序号	开发板管脚	模块管脚	说明
1	3.3V	VCC	VCC是模块的电源引脚
2	GND	GND	GND是模块的接地引脚
3	IO23	DIN	DIN是模块的SPI通信MOSI引脚
4	IO18	CLK	CLK是模块的SPI通信SCK引脚
5	IO19	CS	CS是模块的SPI片选引脚
6	IO22	DC	CS是模块的数据/命令控制引脚
7	IO12	RST	RST是模块的复位引脚
8	IO5	BUSY	BUSY是模块的忙状态输出引脚

4.6.2 模块接口

4.6.2.1 引用模块

```
var canvas = require("canvas");
```

4.6.2.2 打开模块

```
var canvasPort = canvas.open(cfg);
```

功能描述

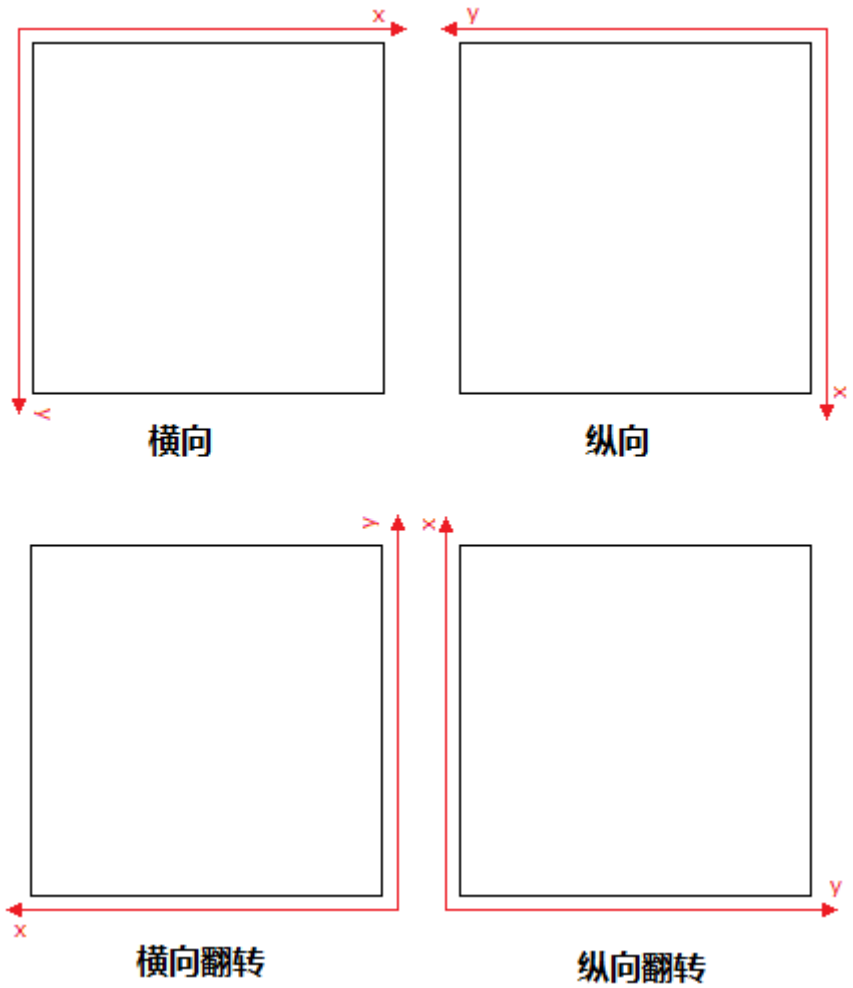
- 按照cfg的配置打开模块。

接口约束

无。

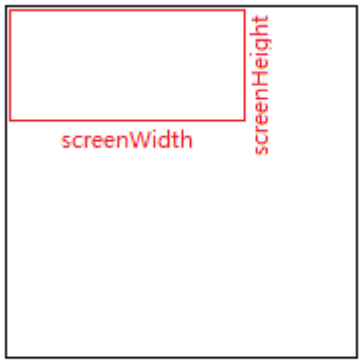
参数列表

- cfg表示打开的配置参数，其中应包含：
  - screenType信息。screenType表示屏幕类型：
- 字符串“ink”表示ink屏；
- 字符串“oled”表示oled屏；
- 字符串“tft”表示tft屏；
- screenType必须显示指定，不可省略。
  - orientation信息。orientation表示显示方向：
- 取值选项如下：
- 1：横向
- 2：纵向
- 3：横向翻转
- 4：纵向翻转



- orientation必须显示指定，不可省略。
  - screenWidth信息。screenWidth表示自定义屏幕宽度：
- 范围：1 ≤ screenWidth ≤ 200。

- screenWidth可省略，省略时screenWidth200。
  - screenHeight信息。screenHeight表示自定义屏幕高度：
- 范围：1 ≤ screenHeight ≤ 200。
- screenHeight可省略，省略时screenHeight将默认设置为200。



以横向为例

返回值

canvasPort为方法返回的端口句柄。

接口示例

```
/* 显示指定自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"ink", screenWidth:64, screenHeight:32, orientation:1};
var canvas_port = canvas.open(cfg);
/* 省略自定义屏幕宽高 */
var canvas = require("canvas");
var cfg = {screenType:"ink",orientation:1};
var canvas_port = canvas.open(cfg);
```

4.6.2.3 新建路径

```
canvas_port.beginPath();
```

功能描述

新建一条路径，路径一旦创建成功，后续的图形绘制命令将被指向该路径。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.beginPath();
```

4.6.2.4 关闭路径

```
canvas_port.closePath();
```


功能描述

关闭路径，之后图形绘制命令又重新指向到上下文中。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.closePath();
```

4.6.2.5 设置画笔起点

```
canvas_port.moveTo(x, y);
```

功能描述

设置画笔的起始点坐标。

接口约束

无。

参数列表

- x为起始点横坐标：
  - 范围：0≤x≤(screenWidth-1)。
- y为起始点纵坐标：
  - 范围：0≤y≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.moveto(0,0);
```

4.6.2.6 绘制线段

```
canvas_port.lineTo(x, y);
```

功能描述

绘制线段。

接口约束

无。

参数列表

- x为终点横坐标：

- 范围：0 ≤ x ≤ (screenWidth-1)。

- y为终点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.lineto(100,100);
```

4.6.2.7 绘制矩形

```
canvas_port.strokeRect(x1, y1,x2, y2);
```

功能描述

绘制矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0 ≤ x1 ≤ (screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0 ≤ y1 ≤ (screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0 ≤ x2 ≤ (screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0 ≤ y2 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeRect(0,0,29,29);
```

4.6.2.8 填充矩形

```
canvas_port.fillRect(x1, y1,x2, y2);
```

功能描述

填充矩形。

接口约束

无。

参数列表

- (x1,y1)和(x2,y2)分别是矩形对角顶点的坐标。
- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillRect(0,0,29,29);
```

4.6.2.9 绘制三角形

```
canvas_port.strokeTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

绘制三角形。

接口约束

无。

参数列表

- x1为顶点1横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。
- x3为顶点3横坐标：
  - 范围：0≤x3≤(screenWidth-1)。
- y3为顶点3纵坐标：
  - 范围：0≤y3≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeTriangle(10, 10,20, 20, 30, 30);
```


4.6.2.10 填充三角形

```
canvas_port.fillTriangle(x1, y1,x2, y2, x3, y3);
```

功能描述

填充三角形。

接口约束

无。

参数列表

- x1为顶点1横坐标：
  - 范围：0 ≤ x1 ≤ (screenWidth-1)。
- y1为顶点1纵坐标：
  - 范围：0 ≤ y1 ≤ (screenHeight-1)。
- x2为顶点2横坐标：
  - 范围：0 ≤ x2 ≤ (screenWidth-1)。
- y2为顶点2纵坐标：
  - 范围：0 ≤ y2 ≤ (screenHeight-1)。
- x3为顶点3横坐标：
  - 范围：0 ≤ x3 ≤ (screenWidth-1)。
- y3为顶点3纵坐标：
  - 范围：0 ≤ y3 ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillTriangle(10, 10,20, 20, 30, 30);
```

4.6.2.11 绘制多边形

```
canvas_port.strokePolygon(verticesArray);
```

功能描述

绘制多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围0 ≤ x ≤ (screenWidth-1)。
  - 顶点的纵坐标范围0 ≤ y ≤ (screenHeight-1)。
  - 顶点个数3 ≤ n ≤ 10

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.strokePolygon(verticesArray1);
```

4.6.2.12 填充多边形

```
canvas_port.fillPolygon(verticesArray);
```

功能描述

填充多边形。

接口约束

无。

参数列表

- verticesArray为多边形顶点坐标array。
  - 顶点的横坐标范围$0 \le x \le (screenWidth-1)$。
  - 顶点的纵坐标范围$0 \le y \le (screenHeight-1)$。
  - 顶点个数$3 \le n \le 10$

返回值

无。

接口示例

```
var verticesArray1 = [[10,20], [20,10], [40,10], [30,20]];
canvas_port.fillPolygon(verticesArray1);
```

4.6.2.13 绘制圆弧

```
canvas_port.arc(x, y, radius, start_angle, end_angle, anti_clockwise);
```

功能描述

绘制圆弧。

接口约束

无。

参数列表

- x为圆心横坐标：
  - 范围：$0 \le x \le 199$。
- y为圆心纵坐标：
  - 范围：$0 \le y \le 199$。
- radius为圆弧半径：
  - 范围：$1 \le radius \le 200$。
- start_angle为起始角度：
  - 范围：$0 \le start\_angle \le 360$。
- end_angle为终止角度：
  - 范围：$0 \le end\_angle \le 360$。

- anti_clockwise为圆弧方向：
- 0表示顺时针，1表示逆时针。

返回值

无。

接口示例

```
canvas_port.arc(50, 30, 10, 0, 135, 0);
```

4.6.2.14 填充圆形

```
canvas_port.fillCircle(x, y, radius);
```

功能描述

填充圆形。

接口约束

无。

参数列表

- x为圆心横坐标：
  - 范围：$0 \leq x \leq 199$。
- y为圆心纵坐标：
  - 范围：$0 \leq y \leq 199$。
- radius为圆弧半径：
  - 范围：$1 \leq radius \leq 200$。

返回值

无。

接口示例

```
canvas_port.fillCircle(60,30,25);
```

4.6.2.15 绘制椭圆

```
canvas_port.strokeEllipse(x1, x2, y1, y2);
```

功能描述

绘制椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：$0 \leq x1 \leq (screenWidth-1)$。
- x2为最右侧点横坐标：
  - 范围：$0 \leq x2 \leq (screenWidth-1)$。

- y1为最上侧点纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.strokeEllipse(10,40,5,60);
```

4.6.2.16 填充椭圆

```
canvas_port.fillEllipse(x1, x2, y1, y2);
```

功能描述

填充椭圆。

接口约束

无。

参数列表

- x1为最左侧点横坐标：
  - 范围：0≤x1≤(screenWidth-1)。
- x2为最右侧点横坐标：
  - 范围：0≤x2≤(screenWidth-1)。
- y1为最上侧点纵坐标：
  - 范围：0≤y1≤(screenHeight-1)。
- y2为最下侧点纵坐标：
  - 范围：0≤y2≤(screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.fillEllipse(10,40,5,60);
```

4.6.2.17 显示字符

```
canvas_port.fillText(x, y, "textString", "fontString");
```

功能描述

显示字符。（使用该功能需要提前将相关字库文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- x为显示起始点横坐标：
  - 范围：0≤x≤(screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0≤y≤(screenHeight-1)。
- "textString"为待显示字符串。
- "fontString"为字体字符串：
  - "0816"表示英文字体；
  - "1616"表示中文字体。

返回值

无。

接口示例

```
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.fillText(0, 20, "一丁", "1616");
```

4.6.2.18 绘制条形码

```
canvas_port.barcode(x, y, "codeNumberString");
```

功能描述

绘制条形码。

接口约束

无。

参数列表

- x为显示起始点横坐标：
  - 范围：0≤x≤(screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0≤y≤(screenHeight-1)。
- "codeNumberString"为待生成条形码的数字字符串。
  - "codeNumberString"的内容应符合条形码规范，是一个12位数字。

返回值

无。

接口示例

```
canvas_port.barcode(50, 50, "692116859353");
```

4.6.2.19 显示图片

```
canvas_port.drawImage("imageNameString", x, y);
```

功能描述

显示图片。（使用该功能需要提前将相关图片文件部署至开发板中，文件部署步骤请参考相关说明文档。）

接口约束

无。

参数列表

- "imageNameString"为图片文件名字符串。
  - "imageNameString"不可包含文件后缀名，例如部署的图片文件为test.pin则填入"test"。
- x为显示起始点横坐标：
  - 范围：0 ≤ x ≤ (screenWidth-1)。
- y为显示起始点纵坐标：
  - 范围：0 ≤ y ≤ (screenHeight-1)。

返回值

无。

接口示例

```
canvas_port.drawImage("test", 0, 0);
```

4.6.2.20 显示图形

```
canvas_port.stroke();
```

功能描述

显示图形。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.stroke();
```

4.6.2.21 清除屏幕

```
canvas_port.clear();
```

功能描述

清除屏幕。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
canvas_port.clear();
```

4.6.2.22 设置线宽属性

```
canvas_port.lineWidth = settingValue;
```

功能描述

设置线宽属性。

接口约束

无。

参数列表

- settingValue为所设置的线宽值。
  - 范围：1 ≤ settingValue ≤ 8
  - 线宽值可省略，省略时线宽值将默认为1。

返回值

无。

接口示例

```
canvas_port.lineWidth = 2;
```

4.6.2.23 设置画笔颜色属性

```
canvas_port.strokeStyle = settingValue;
```

功能描述

设置画笔颜色属性。

接口约束

无。

参数列表

- settingValue为所设置的画笔颜色属性。
  - 取值：0x000000表示画笔为黑色，0xFFFFFF表示画笔为白色。
  - 画笔颜色属性可省略，省略时画笔颜色属性值将默认为0x000000（黑色）。

返回值

无。

接口示例

```
canvas_port.strokeStyle = 0x000000;
```

4.6.3 约束

1. 模块接口中所涉及到的数值的属性必须是不小于零的整数，浮点数、负数等均为非法。

2. 在显式指定了screenWidth及screenHeight的情况下，Canvas模块将只对指定范围内的画笔内容进行处理显示并自动忽略超出范围部分。

3. screenWidth和screenHeight的设定值是相对于lineWidth设为1而言的，当lineWidth被设置为其他合法数值时，screenWidth和screenHeight所实际覆盖的范围会相应缩小，用户需自行计算并确保所绘制图形在其覆盖范围之内。

4.6.4 样例

样例A

介绍

初始化。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
```

样例B

介绍

orientation对比。

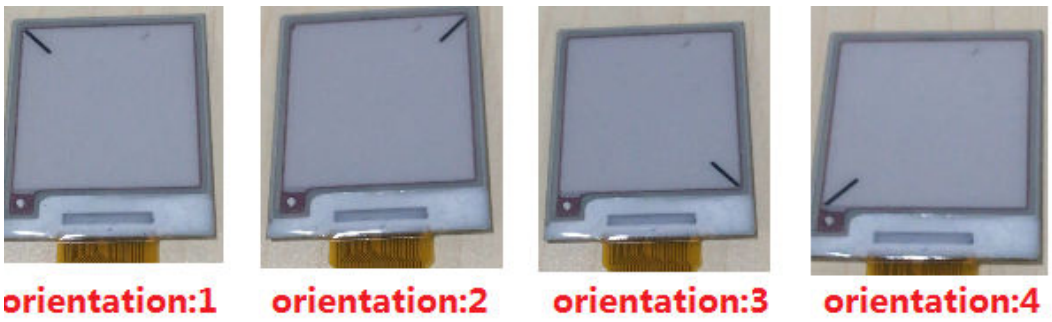
例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 4;
canvas_port.moveto(0,0);
canvas_port.lineto(30,30);
canvas_port.stroke();
canvas_port.closePath();
```

然后依次将代码模板中的cfg修改为：

```
cfg = {screenType:"ink",orientation:2};
cfg = {screenType:"ink",orientation:3};
cfg = {screenType:"ink",orientation:4};
```

样例结果



样例C

介绍

绘制线段。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.moveto(0,5);
canvas_port.lineto(50,5);
canvas_port.lineWidth = 2;
canvas_port.moveto(0,10);
canvas_port.lineto(50,10);
canvas_port.lineWidth = 4;
canvas_port.moveto(0,20);
canvas_port.lineto(50,20);
canvas_port.lineWidth = 8;
canvas_port.moveto(0,30);
canvas_port.lineto(50,30);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例结果

样例D

介绍

绘制矩形。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.strokeRect(0,0,29,29);
canvas_port.lineWidth = 2;
canvas_port.strokeRect(40,0,69,29);
canvas_port.lineWidth = 4;
canvas_port.strokeRect(80,0,109,29);
canvas_port.stroke();
canvas_port.closePath();
```


代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例结果

样例E

介绍

填充矩形。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.fillRect(0,0,29,29);
canvas_port.fillRect(40,0,69,29);
canvas_port.fillRect(80,0,109,29);
canvas_port.stroke();
canvas_port.closePath();
```

样例结果

样例F

介绍

绘制圆弧。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.lineWidth = 1;
canvas_port.arc(50, 30, 10, 0, 135, 0);
canvas_port.lineWidth = 2;
canvas_port.arc(50, 30, 20, 0, 135, 0);
canvas_port.lineWidth = 4;
canvas_port.arc(50, 30, 30, 0, 135, 0);
canvas_port.stroke();
canvas_port.closePath();
```

代码中可通过设置canvas_port.lineWidth来调整线宽，线宽的取值范围为1 ≤ lineWidth ≤ 8。

样例结果

样例G

介绍

显示字符。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.fillText(0, 0, "Hello World!", "0816");
canvas_port.fillText(0, 20, "一丁", "1616");
canvas_port.stroke();
canvas_port.closePath();
```

样例结果

样例H

介绍

绘制条形码。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.barcode(50, 50, "692116859353");
canvas_port.stroke();
canvas_port.closePath();
```

样例结果

样例I

介绍

显示图片。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.drawImage("huawei", 0, 0);
canvas_port.stroke();
canvas_port.closePath();
```

样例结果

样例J

介绍

通过设置画笔颜色实现反显。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
canvas_port.strokeStyle = 0x000000;
canvas_port.fillCircle(60,30,25);
canvas_port.strokeStyle = 0xFFFFFF;
canvas_port.fillCircle(60,30,10);
canvas_port.stroke();
canvas_port.closePath();
```

样例K

介绍

清除屏幕。

例程

```
canvas = require("canvas");
cfg = {screenType:"ink",orientation:1};
canvas_port = canvas.open(cfg);
canvas_port.beginPath();
```



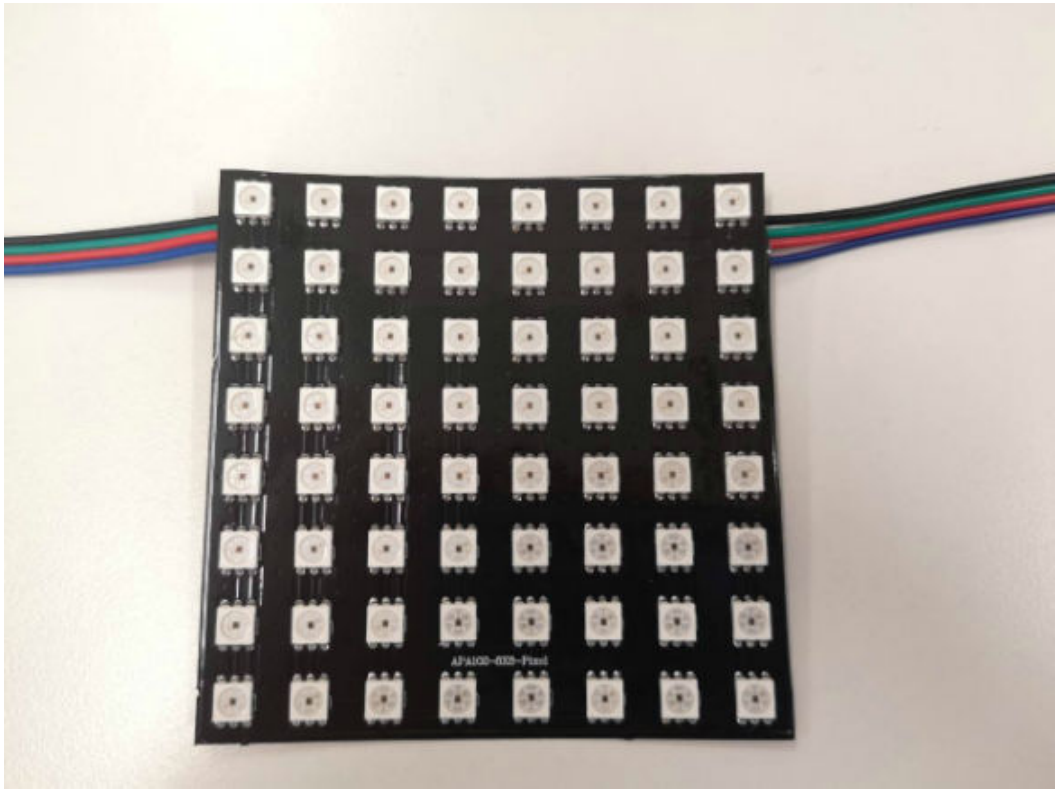
```
canvas_port.arc(30, 30, 20, 0, 360, 0);
canvas_port.clear();
canvas_port.stroke();
canvas_port.closePath();
```

4.7 像素软屏

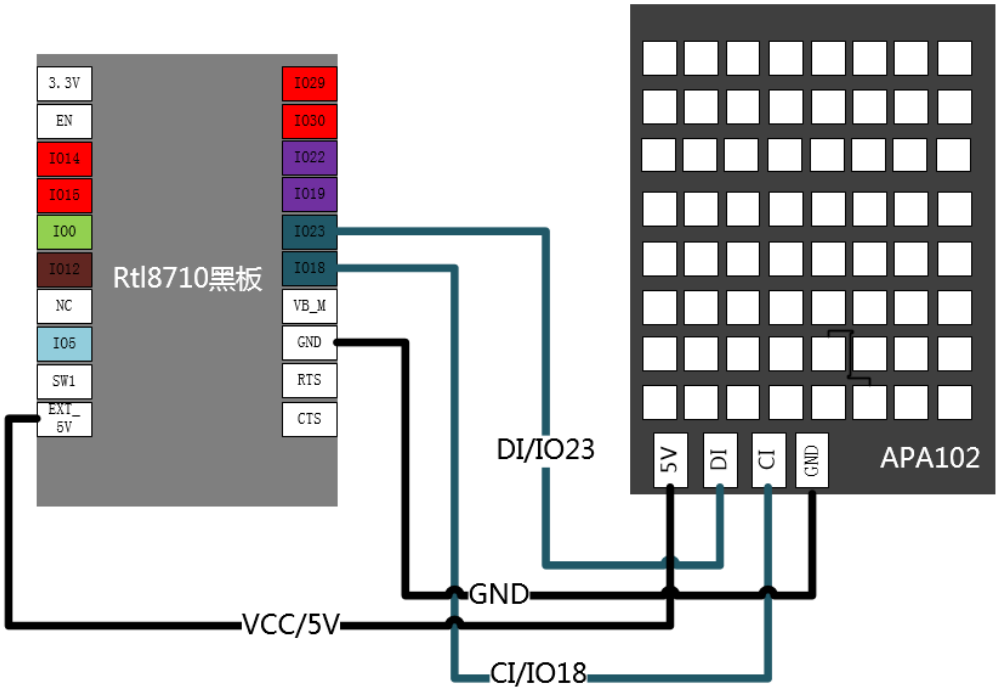
4.7.1 介绍

4.7.1.1 模块参数

APA102是一种RGB全彩LED，该模块采用的是APA102灯珠集成的软屏，参考图片如下：



4.7.1.2 连线方式



开发板与模块引脚连接表：

序号	开发板管脚	模块管脚	备注
1	EXT_5V	5V	灯珠需要5v的供电电压
2	IO23	DI	DI是数据输入线
3	IO18	CI	CI是时钟输入线
4	GND	GND	接地

4.7.2 模块接口

4.7.2.1 引用模块

```
var apa102 = require('apa102');
```

4.7.2.2 初始化模块

```
apa102.init(ionum1, ionum2);
```

功能描述

根据配置初始化apa102端口。

接口约束

无。

参数列表

- ionum1是clock_port，使用开发板的IO号，取值范围是[0, 5, 12, 18, 19, 22, 23, 29, 30]，且只能取正整数，建议不要使用IO0。
- ionum2是data_port，使用开发板的IO号，取值范围是[0, 5, 12, 18, 19, 22, 23, 29, 30]，且只能取正整数，建议不要使用IO0。

返回值

初始化成功则返回0，不成功则返回报错信息。

接口示例

```
apa102.init(18,23);
```

4.7.2.3 运行模式

```
apa102.runMode(light_mode, num_leds, value);
```

功能描述

设置APA102软屏的亮灯模式，以及软屏的灯珠个数，以及灯珠闪亮的颜色。

接口约束

无。

参数列表

- light_mode : 目前设置了三种灯亮的方式“LED_FLOW”（流水灯模式），“LED_BREATH”（呼吸灯模式),“LED_COLOR”（全亮模式）。
- num_leds : 目前使用的软屏是8*8的规格，共64个灯珠。(根据软屏上灯珠的个数确定)
- value ： RGB颜色值或者十六进制颜色码。

返回值

设置成功返回0，失败返回错误信息。

接口示例

```
apa.runMode(apa.LED_FLOW,64,0xf05261);
```

4.7.2.4 设置亮度

```
apa102.setBright(bright_val);
```

功能描述

设置灯珠的亮度。

接口约束

无。

参数列表

- bright_val : 亮度的范围是0~31，默认值为5。

返回值

设置成功返回0，失败返回错误信息。

接口示例

```
apa102.setBright(10);
```

4.7.3 约束

无。

4.7.4 样例

介绍

显示流水灯。

例程

```
var apa=require('apa102');
var tim=require('timer');
apa.init(18,23);//IO18和IO23，采用的是爱联第一代的板子(绿板子)
tim.setInterval(function() {
    apa.runMode(apa.LED_FLOW,64,0xf05261);
}, 50);
/*
    mode:
        apa.LED_FLOW   流水灯
        apa.LED_BREATH 呼吸灯
        apa.LED_COLOR 全亮
    num_leds:1~64
    color:
        red:0xf05261
        blue:0x1d64b1
        pink:0xb11da2
        yellow:0xf0d752
    还提供了apa.setBright(int val)接口，val的范围是0~31的亮度，不设置的话val默认为5
*/
apa.runMode(apa.LED_FLOW,64,0xf05261);
print("js execute done");
```

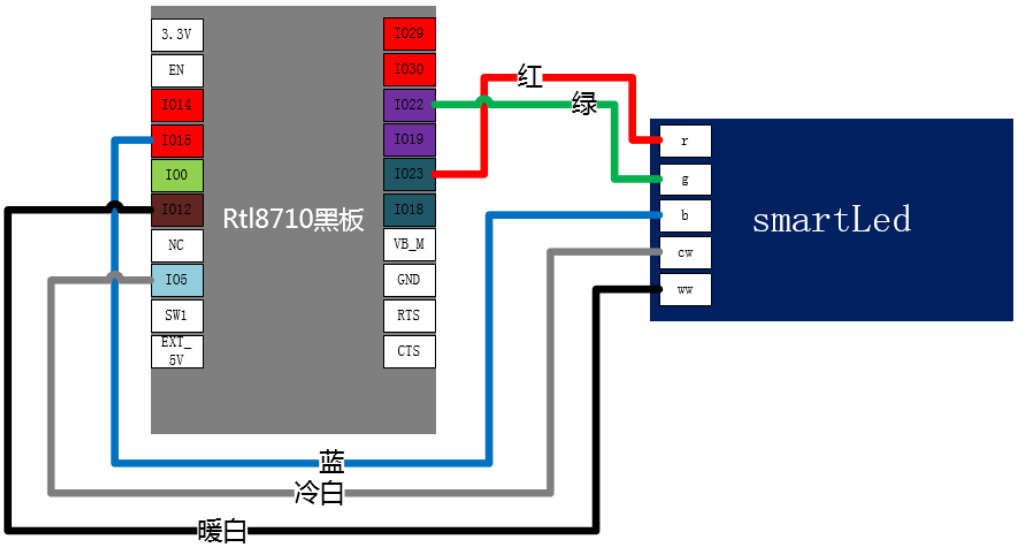
4.8 智能灯

4.8.1 介绍

4.8.1.1 模块参数

smartLed模块可使用5路PWM波去控制智能灯模块的红、绿、蓝、冷白和暖白五种颜色。

4.8.1.2 连线方式



开发板与模块管脚连接表：

序号	开发板管脚	模块管脚	说明
1	IO5	cw	冷白
2	IO12	ww	暖白
3	IO15	b	蓝
4	IO22	g	绿
5	IO23	r	红

4.8.2 模块接口

4.8.2.1 引用模块

```
var smartLed = require('smartLed')
```

4.8.2.2 打开 smartLed 端口

```
smartLed.open(config);
```

功能描述

打开smartLed的端口

接口约束

无。

参数列表

```
- config：打开操作的配置信息，包括如下属性：
  - connection：led的连接方式，module.CA表示共阳型连接，module.CC表示共阴型连接，默认为module.CA；
```



```
  - period：脉冲周期，类型为整数number（number>=0），单位是us，默认脉冲周期为1ms，即为1000(us)；
  - cwPin：色温型冷白驱动管脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，
22，23。0引脚不推荐使用，具体参见gpio模块约束部分,默认为5；
  - wwPin：色温型暖白驱动管脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，
22，23。0引脚不推荐使用，具体参见gpio模块约束部分,默认为5；
  - rPin：色彩型红色驱动管脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，
22，23。0引脚不推荐使用，具体参见gpio模块约束部分,默认为5；
  - gPin：色彩型绿色驱动管脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，
22，23。0引脚不推荐使用，具体参见gpio模块约束部分,默认为5；
  - bPin：色彩型蓝色驱动管脚，以爱联模组RTL8710开发板为例，可用引脚号为0，5，12，14，15，18，19，
22，23。0引脚不推荐使用，具体参见gpio模块约束部分,默认为5
```

返回值

smartLed端口。

接口示例

```
var config = {
  connection:module.CA,
  pwmFreq:1,
  cwPin:5,
  wwPin:12,
  rPin:23,
  gPin:22,
  bPin:15
};
var port = smartLed.open(config);
```

4.8.2.3 点亮

```
void switchOn();
```

功能描述

点亮智能灯。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
port.switchOn();
```

4.8.2.4 熄灭

```
void switchOff();
```

功能描述

熄灭智能灯。

接口约束

无。

参数列表

无。

返回值

无。

接口示例

```
port.switchOff();
```

4.8.2.5 亮度调节

```
void brightSet(brightLevel);
```

功能描述

调节当前智能灯的亮度。

接口约束

无。

参数列表

brightLevel：亮度值，其取值范围为[0,255]。

返回值

无。

接口示例

```
port.brightSet(128);
```

4.8.2.6 色温调节

```
void cctSet(cctVal);
或
void cctSet(cwLevel, wwLevel);
```

功能描述

设置智能灯的色温，支持两种方式接口。

接口约束

无。

参数列表

- cctVal：色温值，其取值范围为[2700,6500]；
- cwVal：冷白值，其取值范围为[0,255]；
- wwVal：暖白值，其取值范围为[0,255]；

返回值

无。

接口示例

```
port.cctSet(4000);port.cctSet(50, 100);
```


4.8.2.7 色彩调节

```
void colorSet(colorVal);
或
void colorSet(rLevel, gLevel, bLevel);
```

功能描述

设置智能灯的色彩，支持两种方式的接口。

接口约束

无。

参数列表

- colorVal：颜色设置值，其取值范围为[0x000000,0xFFFFFF]，第一个字节表示红色值，第二个字节表示绿色值，第三个字节表示蓝色值；
- rLevel：红色值，其取值范围为[0,255]；
- gLevel：绿色值，其取值范围为[0,255]；
- bLevel：蓝色值，其取值范围为[0,255]；

返回值

无。

接口示例

```
port.colorSet(0x0000FF);
port.colorSet(0, 0, 255);
```

4.8.2.8 静态模式

```
void staticMode(cctVal, colorVal, brightLevel)
或
void staticMode(cwValue, wwValue, colorVal, brightLevel)
```

功能描述

设置智能灯为静态模式，同时输入冷暖白和RGB及亮度值。

接口约束

无。

参数列表

- cctVal：色温值，其取值范围为[2700,6500]；
- cwVal：冷白值，其取值范围为[0,255]；
- wwVal：暖白值，其取值范围为[0,255]；
- colorVal：颜色设置值，其取值范围为[0x000000,0xFFFFFF]，第一个字节表示红色值，第二个字节表示绿色值，第三个字节表示蓝色值；
- brightLevel：亮度值，其取值范围为[0,255]；

返回值

无。

接口示例

```
port.staticMode(2700, 0, 50);
```



```
port.staticMode(120, 200, 0, 50);
```

4.8.2.9 持续周期性变化接口

```
port.on(module.TIMER, period, duration, function);
```

功能描述

运用该接口可实现闪烁模式、呼吸模式及渐变模式。

接口约束

无。

参数列表

- module.TIMER:事件回调函数的类型，目前只支持module.TIMER，表示持续周期性变化接口。
- period:调用时间间隔，单位为ms。
- duration:总共持续时间，-1为无穷大，单位为ms，是period的倍数。
- function：为类似function(count){}形式的函数，其中count为当前计数，从1开始，每隔period时间，将调用一次。

返回值

无。

接口示例

```
//闪烁间隔为1s，总共闪烁1分钟
port.on(module.TIMER, 1000, 60000, function(count){
    if (count % 2 == 0){
        port.brightSet(0);
    }
    else{
        port.brightSet(255);
    }
});
//20ms刷新一次呼吸灯亮度，最亮到下次最亮的周期为1s，保持一直呼吸
port.on(module.TIMER, 20, -1, function(count){
    var data = count % 50; // 呼吸周期为1s，20ms刷新一次，那么50次为一个周期
    var right = data > 25 ? data - 25 : 25 - data; // 亮度为 (|x - 25|) / 25
    port.brightSet(right / 25 * 255);})
//20ms刷新一次渐变颜色，共持续1分钟
port.on(module.TIMER, 20, 60000, function(count){
    // 伪代码，target和origin是目标颜色和起始颜色
    var curRGB = count * (target - origin) / (60000 / 20) + origin;
    port.colorSet(curRGB);
})
```

4.8.2.10 pwm 通道设置操作

```
void pwmChannelSet(channel, dutyCycle);
```

功能描述

体统原生操作5路PWM的接口。

接口约束

无。

参数列表

- channel pwm通道，如下：
- module.CW 表示冷白pwm通道；
- module.WW 表示暖白pwm通道；
- module.R 表示红色pwm通道；
- module.G 表示绿色pwm通道；
- module.B 表示蓝色pwm通道；
- dutyCycle：占空比，取值范围为[0, 1]。

返回值

无。

接口示例

```
port.pwmChannelSet(module.CW, 0.5);
```

4.8.3 约束

无。

4.8.4 样例

介绍

smart LED使用。

例程

```
smartled = require('smartLed');
var config = {'connection':smartled.CA, 'period':10};
led = smartled.open(config);
led.switchOn();
led.brightSet(255);
led.brightSet(0);
led.cctSet(2700);
led.cctSet(6500);
led.cctSet(255, 200);
led.colorSet(120, 255, 240);
led.colorSet(0xffccee);
led.colorSet(0);
led.colorSet(0xffffff);
led.staticMode(2800, 0xffffff, 255);
led.staticMode(255, 255, 0xffffff, 255);
led.pwmChannelSet(smartled.WW, 0.5);
led.pwmChannelSet(smartled.R, 0.5);
led.pwmChannelSet(smartled.G, 0.5);
led.pwmChannelSet(smartled.B, 0.5);
led.pwmChannelSet(smartled.CW, 1);
led.on(smartled.TIMER, 100, 50000, function(count){
    print(count);
    led.colorSet(0);
});
timer.setInterval(function(){
    led.on (smartled.TIMER, 100, 10000, function(count){
        print("new:", count);
        led.colorSet(100);
    });
}, 5000);
```


5 网络模块接口

本模块文档待完善。

