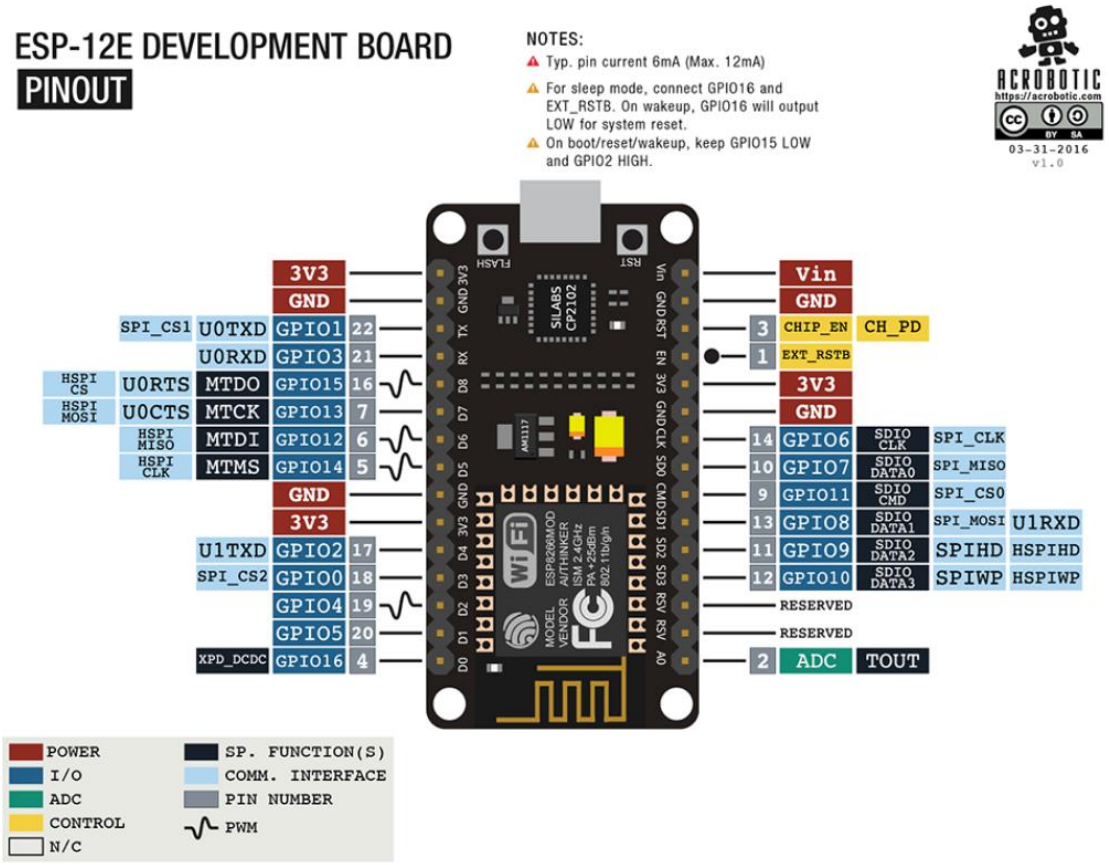
物联网配件参考文档

微信公众号：董哥物联网

源码均采用 ArduinoIDE 的 C++语法

目录

一、ESP8266



输出查看调试

```
int i=0;
float x=3.14;
bool y=true;
String s="hello";
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600); // 启动串口通讯
}
void outstr(String str){
  Serial.println(str);
}
void loop() {
  for(i=0;i<5;i++){
    if(i==3){
      break;
```

```
    }else {
      outstr("i="+String(i));
    }
  }
  outstr("======================");
  delay(1000);//延迟 1000 毫秒=1 秒
}
```

引脚电平设置

```
void setup() {
  //put your setup code here, to run once:
  Serial.begin(9600);//启动串口通讯
  pinMode(D0,OUTPUT);//设置引脚为输出模式
  pinMode(D1,OUTPUT);
  digitalWrite(D0,LOW);//先把此引脚电平置为低电平
  digitalWrite(D1,LOW);//先把此引脚电平置为低电平
}
void outstr(String str){
  Serial.println(str);
}
void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(D0)==LOW){//读取引脚状态是否为低电平
    outstr("D0 is LOW");
    digitalWrite(D0,HIGH);//把此引脚电平置为高电平
    digitalWrite(D1,LOW);//把此引脚电平置为高电平
  }else{
    outstr("D0 is HIGH");
    digitalWrite(D0,LOW);//把此引脚电平置为低电平
    digitalWrite(D1,HIGH);//把此引脚电平置为高电平
  }
  delay(5000);
}
```

wifi 连接

```
#include<ESP8266WiFi.h> //需要导入 WiFi 支持库
void wificonnect(){
 if(WiFi.status() != WL_CONNECTED){
```

```
    // 连接 WiFi 名（此处使用 wifiname 为示例）请将您需要连接的 WiFi 名填入引号中
    // 连接 WiFi 密码（此处使用 password 为示例）请将您需要连接的 WiFi 密码填入引号中
    WiFi.begin("wifiname", "password");          // 开始进行网络连接
    int i = 0;                    // 这一段程序语句用于检查 WiFi 是否连接成功
    while (WiFi.status() != WL_CONNECTED) {   // WiFi.status()函数的返回值是由 NodeMCU 的 WiFi 连接状态所决定的。
                                  // 如果 WiFi 连接成功则返回值为 WL_CONNECTED
      delay(1000);                //适当延时 1 秒
      //WiFi.disconnect();              //清除之前的连接配置,防止路由器重启造成的问题
      outstr(String(i++)); // 此处通过 While 循环让 NodeMCU 每隔一秒钟检查一次 WiFi.status()函数返回值
    }                             // 同时 NodeMCU 将通过串口监视器输出连接时长读秒。
                                  // 这个读秒是通过变量 i 每隔一秒自加 1 来实现的。
    outstr("Connection success!");  // NodeMCU 将通过串口监视器输出"连接成功"信息。
    outstr("IP address:  ");        // 同时还将输出 NodeMCU 的 IP 地址。这一功能是通过调用
    outstr(WiFi.localIP().toString());        // 函数的返回值即 NodeMCU 的 IP 地址。
  }
}
void outstr(String str){
  Serial.println(str);
}
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);              // 启动串口通讯
  wificonnect();
  String s="heelo";
  outstr(String(s.length()));
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

Wifimanage 配置网络

```
#include <ESP8266WiFi.h>
#include <WiFiManager.h>
void setup() {
  Serial.begin(9600);//初始化串口设置
  String  s;
  int i;
  Serial.print(s.length());
  WiFiManager w;//建立 WiFiManager 对象
  w.autoConnect("AutoConnectWifi");//创建名称为：AutoConnectWifi 的
wifi 网络支持配置
  Serial.print("WiFi Connected!");
}

void loop() {

}
```

联网后发送 http 请求

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
//以下请求地址是此开发板从服务器获取控制数据的地址
String url="http://8266iot.cn";//董哥物联网网站
// 发送 HTTP 请求并且将服务器响应通过串口输出
HTTPClient httpClient;//建立 http 连接对象
void outstr(String str){
  Serial.println(str);
}
String httpsendtoserver(String URL){
  String s="";
  httpClient.begin(URL); //使用连接对象开启 http 连接请求
  int httpCode = httpClient.GET();//获取请求返回码
  if (httpCode == HTTP_CODE_OK) {//请求发送成功
    s = httpClient.getString();//获取请求发送后返回的信息
  } else {//请求发送失败
    outstr("shibai");
    outstr(String(httpCode));//输出返回的状态码
    s=String(httpCode);
  }
```

```
  httpClient.end();//关闭 ESP8266 与服务器连接
  return s;
}
void wificonnect(){
 if(WiFi.status() != WL_CONNECTED){
    // 连接 WiFi 名（此处使用 wifiname 为示例）请将您需要连接的 WiFi 名填入引号中
    // 连接 WiFi 密码（此处使用 password 为示例）请将您需要连接的 WiFi 密码填入引号中
    WiFi.begin("wifiname", "password");          // 开始进行网络连接
    int i = 0;                    // 这一段程序语句用于检查 WiFi 是否连接成功
    while (WiFi.status() != WL_CONNECTED) {   // WiFi.status()函数的返回值是由 NodeMCU 的 WiFi 连接状态所决定的。
                                  // 如果 WiFi 连接成功则返回值为 WL_CONNECTED
      delay(1000);                    //适当延时 1 秒
      Serial.println(i++); // 此处通过 While 循环让 NodeMCU 每隔一秒钟检查一次 WiFi.status()函数返回值
    }                          // 同时 NodeMCU 将通过串口监视器输出连接时长读秒。
                               // 这个读秒是通过变量 i 每隔一秒自加 1 来实现的。
    Serial.println("Connection success!");  // NodeMCU 将通过串口监视器输出"连接成功"信息。
    Serial.print("IP address:  ");        // 同时还将输出 NodeMCU 的 IP 地址。这一功能是通过调用
    Serial.println(WiFi.localIP());         // 函数的返回值即 NodeMCU 的 IP 地址。
  }
}
void setup() {
  Serial.begin(9600);
  wificonnect();
}

void loop() {
  if(WiFi.status() == WL_CONNECTED){//连接互联网
    String s=httpsendtoserver(url);

    outstr("return msg:"+ s);
    //outstr(url);
    delay(500);
  }

}
```

字符串引脚处理

```
void dealout(String str){//str="1000500"
  if(str[0]=='0'){//当数据第一位是 0
    digitalWrite(D0,LOW);//将 D0 置为低电平
    outstr("D0=LOW  ",0);
  }else{
    digitalWrite(D0,HIGH);//当数据第一位是 1 则将 D0 置为高电平
    outstr("D0=HIGH  ",0);
  }
  if(str[1]=='0'){
    digitalWrite(D1,LOW);
    outstr("D1=LOW  ",0);
  }else{
    digitalWrite(D1,HIGH);
    outstr("D1=HIGH  ",0);
  }
  if(str[2]=='0'){
    digitalWrite(D2,LOW);
    outstr("D2=LOW  ",0);
  }else{
    digitalWrite(D2,HIGH);
    outstr("D2=HIGH  ",0);
  }
  if(str[3]=='0'){
    digitalWrite(D3,LOW);
    outstr("D3=LOW  ",0);
  }else{
    digitalWrite(D3,HIGH);
    outstr("D3=HIGH  ",1);
  }
}
```

引脚处理和调速

```
void setup() {
  Serial.begin(9600);
```

```
  wificonnect();
  pinMode(D0,OUTPUT);//设置为输出模式
  pinMode(D1,OUTPUT);
  pinMode(D2,OUTPUT);
  pinMode(D3,OUTPUT);
}
void dealout(String str){//str="1000500"
  String data="";
  int i=0;
  if(str.length()<=14){//当前 4 位控制数据+10 位 PWM 控制数据的个数小于 14 个
    Serial.println(str.length());
    int pwmlen=str.length()-4;//3
    Serial.println("pwmlen="+pwmlen);
    int j=10-pwmlen;//7
    while((j--)>0){
      data+="0";//不足 10 位前面添加 0  0000 000
    }
    for(i=0;i<pwmlen;i++){//将 PWM 值添加到 data  0000000500
      data+=str[4+i];
    }
  }else{//大于 14 个只取前 10 个 PWM 值
    for(i=0;i<10;i++){
      data+=str[4+i];           //将 PWM 值添加到 data
      //data=data+str[4+i];
    }
  }
  int PWM =0;
  int k=1000000000;
  for(i=0;i<10;i++){//0000000500
    PWM+=((int)data[i]-48)*k;//53-48
    k/=10;//k=k/10;
  }
  Serial.println(PWM);
  Serial.println(data);
  if(str[0]=='0'){//当数据第一位是 0
    digitalWrite(D0,LOW);//将 D0 置为低电平
    outstr("D0=LOW  ",0);
  }else{
    digitalWrite(D0,HIGH);//当数据第一位是 1 则将 D0 置为高电平
    analogWrite(D0,PWM);//控制电机转速
    outstr("D0=HIGH  ",0);
  }
  if(str[1]=='0'){
    digitalWrite(D1,LOW);
```

```
    outstr("D1=LOW  ",0);
  }else{
    digitalWrite(D1,HIGH);
    analogWrite(D1,PWM);//控制电机转速
    outstr("D1=HIGH  ",0);
  }
  if(str[2]=='0'){
    digitalWrite(D2,LOW);
    outstr("D2=LOW  ",0);
  }else{
    digitalWrite(D2,HIGH);
    outstr("D2=HIGH  ",0);
  }
  if(str[3]=='0'){
    digitalWrite(D3,LOW);
    outstr("D3=LOW  ",0);
  }else{
    digitalWrite(D3,HIGH);
    outstr("D3=HIGH  ",1);
  }
}
```

单温度传感器单引脚

```
#include <OneWire.h>
/*OneWire 库是一个用于与基于 1-Wire 协议的设备进行通信的 Arduino 库。
1-Wire 是一种串行通信协议，
它允许在单个数据线上进行通信，
并且每个 1-Wire 设备都有唯一的 64 位 ROM 代码，可以通过该代码来识别设备。
使用 OneWire 库，可以方便地与各种 1-Wire 设备（如温度传感器、EEPROM 等）进行交互，
从而实现对这些设备的控制和数据采集。*/
#include <DallasTemperature.h>
/*DallasTemperature 库是一个用于与基于 DS18B20 型号的数字温度传感器进行通信的 Arduino 库。
DS18B20 是一种基于 1-Wire 协议的数字温度传感器，它可以通过单个数据线进行通信，
并且具有唯一的 64 位 ROM 代码。DallasTemperature 库与 OneWire 库结合使用，
可以方便地读取 DS18B20 温度传感器的数据，并进行相应的处理和控制。*/
const int sensorPin = D3; // 根据实际情况调整引脚
OneWire oneWire(sensorPin);//创建串口通信的对象，绑定串口通信的引脚
```

```
DallasTemperature sensors(&oneWire);//创建  温度获取传感器对象  ，绑定串口通信对象

void outstr(String str){
  Serial.println(str);
}
void setup() {
  Serial.begin(9600); // 启动串口通讯
  sensors.begin();//初始化温度监测
  sensors.setResolution(12); // 设置传感器分辨率
  /*传感器的分辨率是指传感器可以测量的温度范围内所能检测到的最小温度变化。
  在 DS18B20 数字温度传感器中，分辨率可以通过设置传感器分辨率来控制。
  DS18B20 数字温度传感器有 3 种分辨率：9 位、10 位和 12 位，可以通过编程来设置。
  各分辨率的温度分辨率如下：
  9 位分辨率：0.5°C
  10 位分辨率：0.25°C
  12 位分辨率：0.0625°C
  例如，在使用 10 位分辨率的传感器中，如果传感器测量到的温度值变化了 0.25°C（或更大），
  则传感器将输出一个新的温度值。*/
}
float gettemperature(){
  sensors.requestTemperatures();//请求温度数据
  return sensors.getTempCByIndex(0);
  // float temperature = sensors.getTempCByIndex(0);//通过索引为 0 获取温度数据
  // return temperature;  //返回温度数据
}
void loop() {
  outstr("temperature="+String(gettemperature()));
  delay(1000);
}
```

继电器远程开关灯

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
//以下请求地址是此开发板从服务器获取控制数据的地址
String url="http://8266iot.cn";//董哥物联网网站
String httpsendtoserver(String URL){
  String s="";
```

```
  httpClient.begin(URL); //使用连接对象开启 http 连接请求
  int httpCode = httpClient.GET();//获取请求返回码
  if (httpCode == HTTP_CODE_OK) {//请求发送成功
    s = httpClient.getString();//获取请求发送后返回的信息
  } else {//请求发送失败
    outstr("shibai:",0);
    outstr(String(httpCode),1);//输出返回的状态码
    s=String(httpCode);
  }
  httpClient.end();//关闭 ESP8266 与服务器连接
  return s;
}
void setup() {
  Serial.begin(9600);
  wificonnect();
  pinMode(D0,OUTPUT);//设置为输出模式
  //digitalWrite(D4,LOW);
}
void loop() {
  if(WiFi.status() == WL_CONNECTED){//连接互联网
    String s=httpsendtoserver(url);//0000500
    outstr("return msg:"+ s,1);
    dealout(s);
    delay(100);
  }
}
void dealout(String str){
  if(str.length()==1){
    if(str[0]=='1'){//当数据第一位是 0
      digitalWrite(D0,HIGH);//将 D4 置为高电平
      outstr("turn on",1);
    }else{
      digitalWrite(D0,LOW);//
      outstr("turn off",1);
    }
  }else{
    digitalWrite(D0,LOW);//
    outstr("turn off",1);
  }
}
```

二、继电器

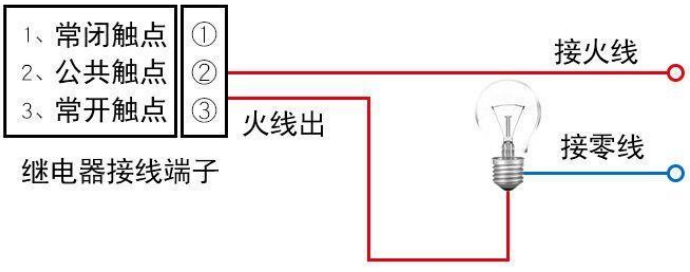
产品参数

1路继电器模块-介绍

1、常开接口限大负载：交流25010A，直流30/10A
2、采用贴片光耦隔离，驱动能力强，性能稳定：触发电流5MA
3、模块工作电压有5V、12V、24V可供选择（请注意连接是否为所需电压）
4、模块可以通过跳线设置高电平或低电平触发
5、容错设置，即使控制线断，继电器也不会动作
6、电源指示灯（绿色），继电器状态指示灯（红色
7、接口设计人性化，接口均可通过接线端子直接连线引出非常方便
8、设有4个固定螺栓孔，孔3.1MM，问距44.5M*20.5MM

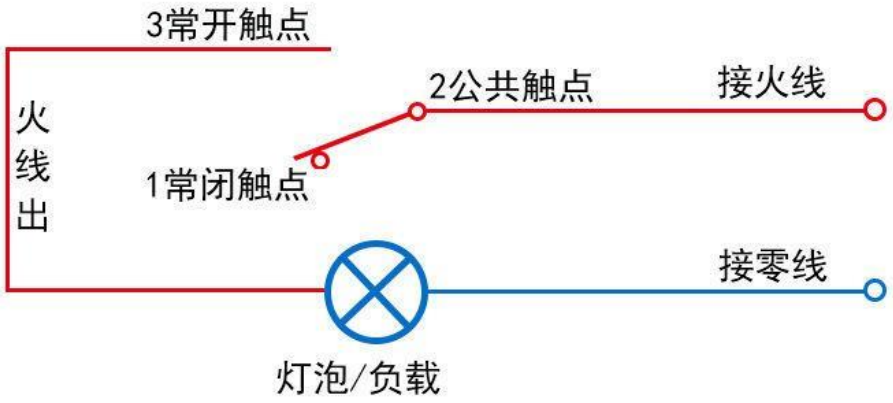
手工测量，会有细微误差

继电器接法示意图



继电器接线端子

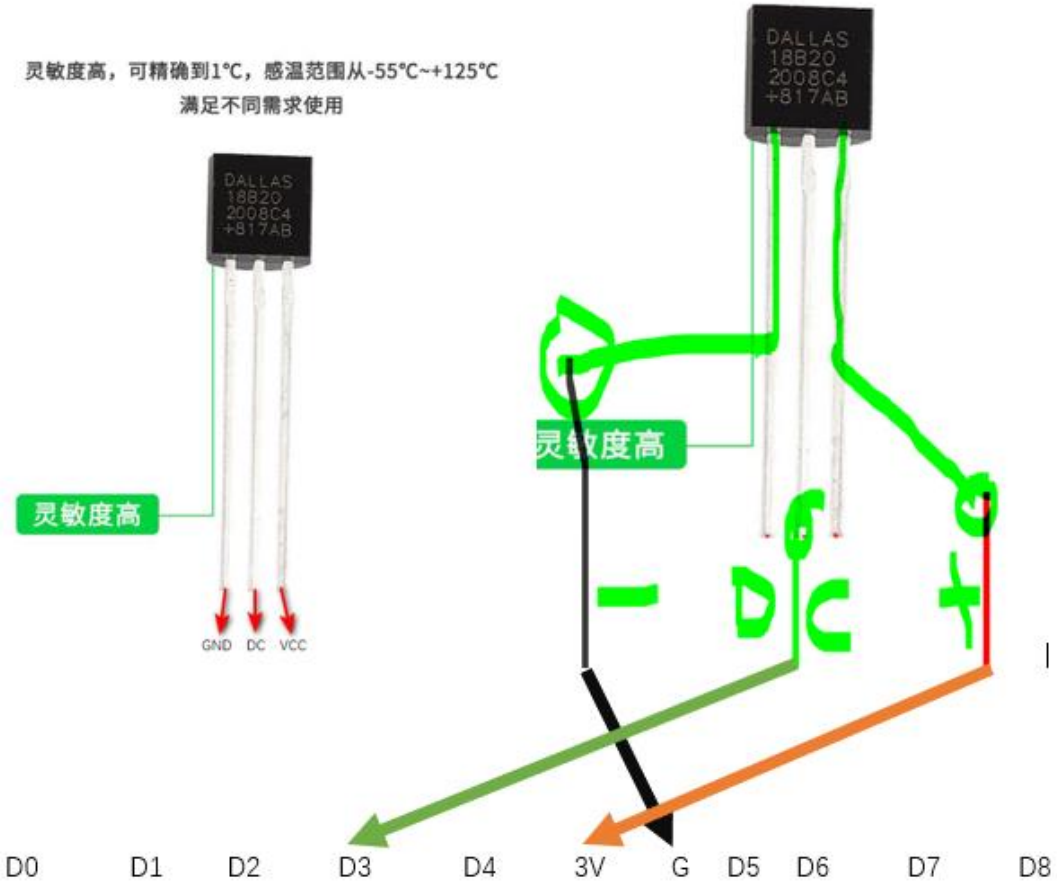
继电器接法电路原理图



1路继电器模块一接口说明

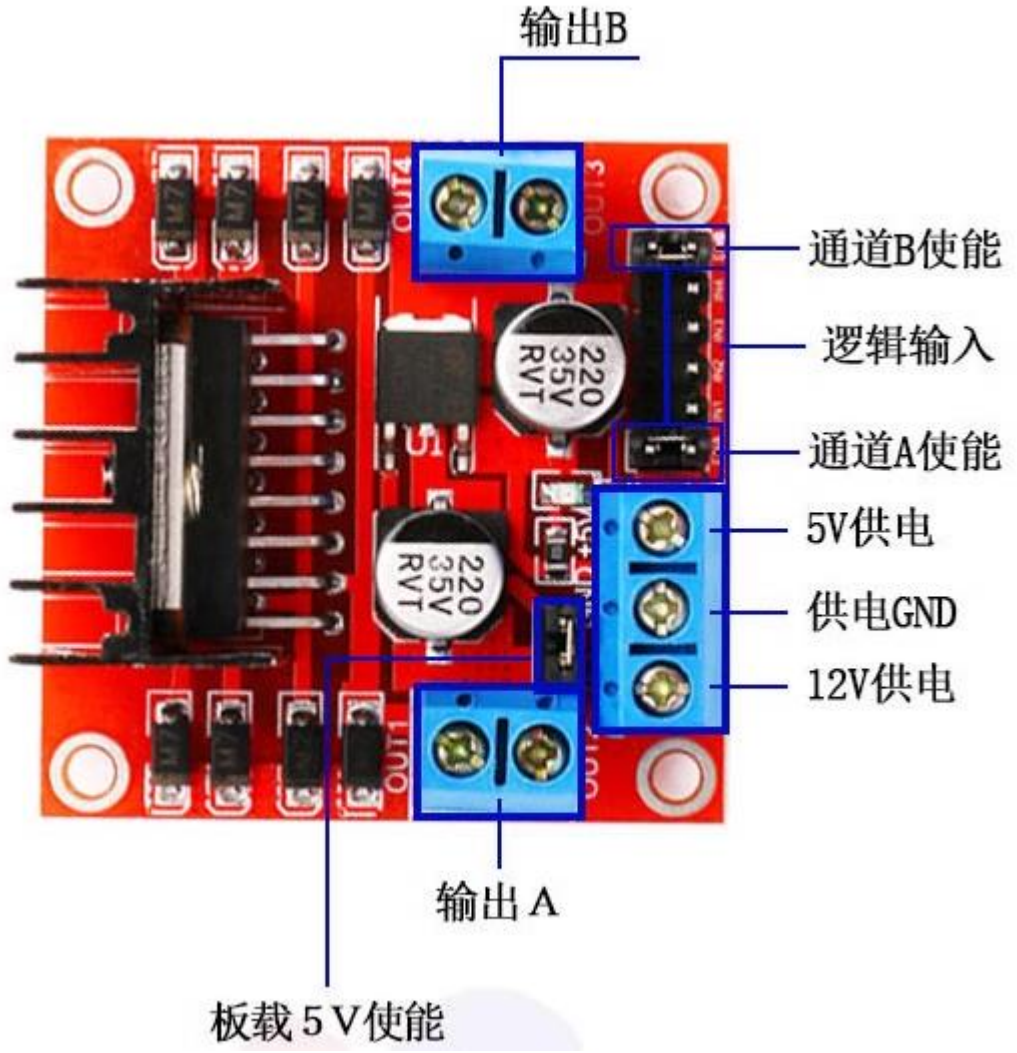
一、输入部分
1. DC+:接电源正极(电压按继电器要求,5V、12V和24V选择)
2、DC-:接电源负极
3、N:可以高或低电平控制继电器吸合
二、继电器输出端
1、NO:继电器常开接口,继电器吸合前悬空,吸合后与COM短接
2、COM继电器公用接口
3、NC:继电器常闭接口,继电器吸合前与COM短接,吸合后悬空
三、高低电平触发选择端
1、跳线与LOW短接时为低电平触发
2、跳线与HGH短接时为高电平触发

三、温度传感器（DS18B20）



建议温度传感器接 D3 或者 D4 引脚，其他引脚读取的时候会出问题

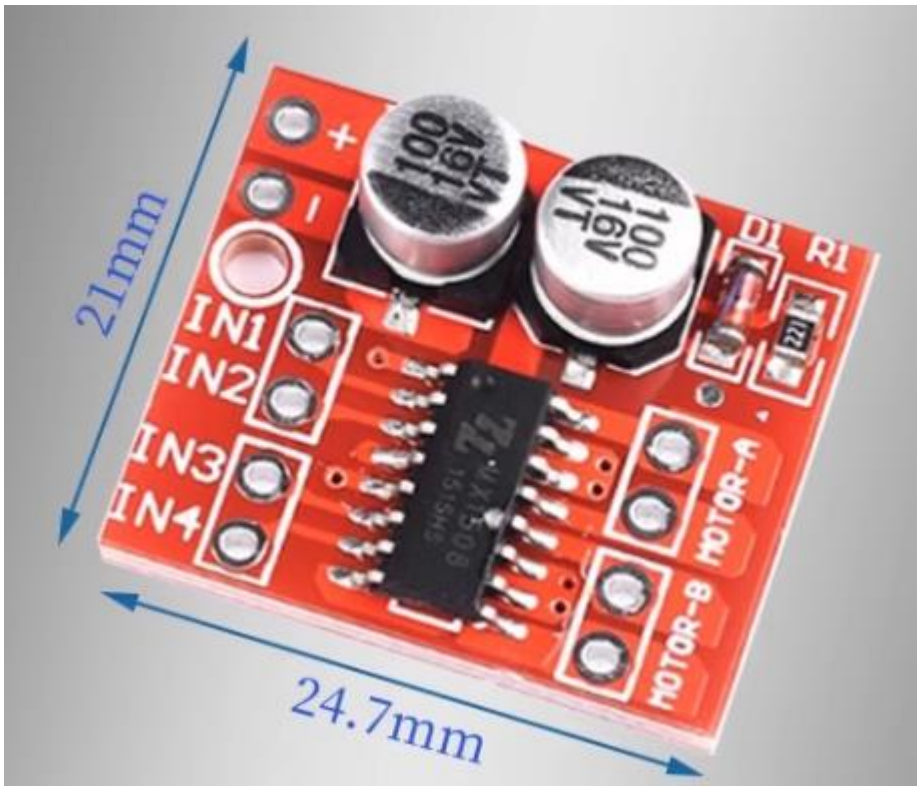
四、L298N-二路直流电机控制模块

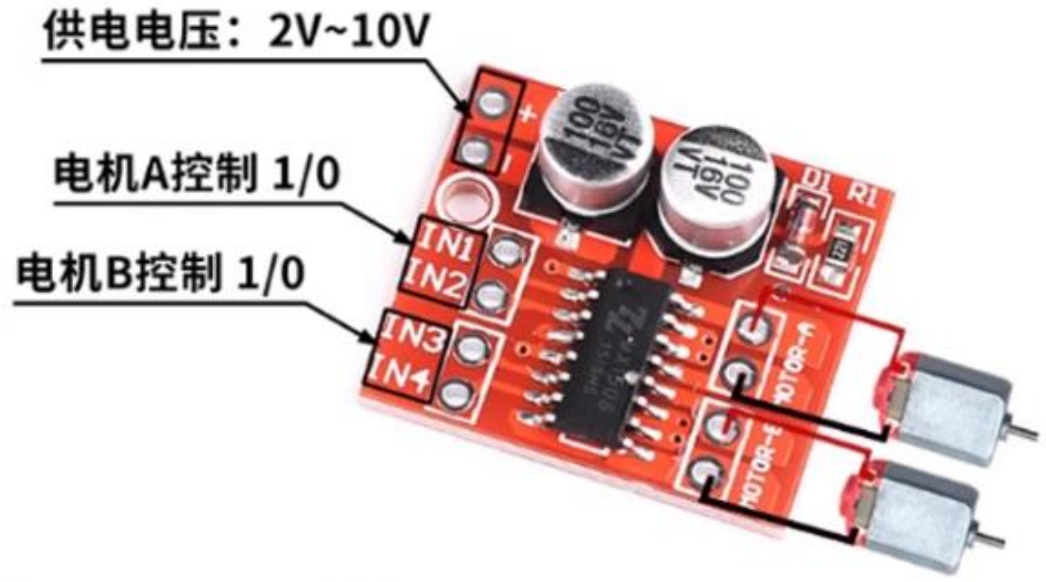


主控芯片：L298N
工作模式：H 桥驱动（双路）
逻辑电压：5V
逻辑电流：0-36mA
驱动电压：5-35V,如需要板内取电，则供电范围 7V-35V
驱动电流：2A(MAX 单桥)
存储温度：-20℃到+135℃
功率：≤25W
控制信号输入电压范围：低电平：一 0.3V≤Vin≤1.5V
高电平：2.3V≤Vin≤Vss
使能信号输入电压范围：电平：一 0.3≤Vin≤1.5V(控制信号无效)
高电平：2.3V≤Vin≤Vss(控制信号有效)
外围尺寸：43*43*27mm

当驱动电压（板子背面标识为 12V 输入，实际可以接受的输入范围是 7-12V)为 7V-12V 的时候，即 12V 电机驱动端子接通驱动电源时，插上跳线帽使用板载的 78M05 供给芯片的逻辑电源，指示灯亮，可以不用再外接逻辑电源；当使用板载 5V 供电之后，接口中的+5V 供电端子不要输入电压，但是可以引出 5V 电压供外部使用。(这种即为常规应用!)
当驱动电压高于 12V,小于等于 24V(芯片手册中提出可以支持到 35V,但是按照经验一般 L298 保守应用 MAX 电压支持到 24V 已经很了不起！)时，比如要驱动额定电压为 18V 的电机。首先必须拔除板载 5V 输出使能的跳线帽，指示灯熄灭，不使用板载的 78M05 供给芯片的逻辑电源，然后在 5V 输出端口外部接入 5V 电压对 L298N 内部逻辑电路供电。(这种是高压驱动的非常规应用！)
5V 使能即一个电平为 5V 的控制信号，当此信号输入有效时，且电机驱动模块中电源供电正常时，电机驱动模块输出电流。否则即使电源供电正常，电机上也无电流。L298N 使能端 (高电平有效，常态下用跳线帽接于 VCC),可通过这两个端口实现 PWM 调速（使用 PWM 调速时取下跳线帽)ENA 和 ENB 接上 PWM 信号，IN1、IN2、IN3、INA 正常接上高低电平使电机正转、反转或停转。

L298N 供电的 5V 如果是用另外电源供电的话，(即不是和单片机的电源共用)，那么需要将单片机的 GND 和模块上的 GND 连接在一起，只有这样单片机上过来的逻辑信号才有个参考 0 点。板载 5V 稳压芯片的输入引脚和电机供电驱动接线端子导通的。





2 路直流电机驱动模块

电机驱动模块非常适合在电池供电的智能小车，玩具小车，机器人等上面使用，供电电压 2V~10V,可同时驱动两个直流电机或者 1 个 4 线 2 相式步进电机，可实现正反转和调速的功能，每路电流能到 1.5A 持续电流，峰值电流可达 2.5A,有热保护，不用怕电堵转烧坏，温度下降后自动恢复。(目前市面上的智能小车电和电流都在此范围内)。

内置低导通内阻 MOS 开关管，发热极小，无需散热片，体积小，省电是您电池供电的理想选择。(L298N 内部为晶体管开关效率低发热高，需散热，体积笨重，市面上的 L298 极容易烧坏都没有采用原装芯片，本产品能够完美替代。)体积小，质量轻，低待机电流，是您模型车载的理想选择。

产品参数

供电电压：2V~10V

信号端输入电压：1.8~7V

单路工作电流与：1.5A

峰值电流：可达 2.5A

低待机电流：小于 0.1uA 内置带迟滞效应的过热保护电路(TSD)

无需担心电机堵转

产品尺寸：24.7*21*5mm(长宽)，超小体积，适合组装和车载

安装孔直径：2mm

直流电机驱动逻辑真值表:

直流电机	旋转方式	IN1	IN2	IN3	IN4
MOTOR-A	正转(调速)	1/PWM	0		
	反转(调速)	0	1/PWM		
	待机	0	0		
	刹车	1	1		
MOTOR-B	正转(调速)			1/PWM	0
	反转(调速)			0	1/PWM
	停止			0	0
	刹车			1	1

注：“1”代表高电平；“0”代表低电平；“PWM”代表脉宽调制液，调节占空比改变转速。IN1，IN2控制直流电机A；IN3，IN4控制直流电机B；两路完全独立的。输入端1Nx有防共态导通功能，悬空时等效于为低电平输入。

五、人体红外运动检测模块（SR602）

工作参数：
热释电人体红外传感器模块，型号：SR-602
感应距离：最长5米;推荐0-3.5M。
输出：高电平，H = 3.3V，L = 0V
电源DC：3.3V-15V
静态电流：20uA

此模块灵敏度高，响应速度快，静态功耗小，体积小巧，易于安装，已装好透镜，已焊接好排针，配上电源无需调试。可用于人体感应灯，报警器，安防，门禁按需摄像，农林矿业自动设备等 。

要点说明：

1.本产品输出的高电平时间可调，2.5秒到1小时，出厂时设定的输出时间是2.5秒，如需更改，可更改一个贴片电阻。典型延时时间对应的电阻值请阅读参考下文相关表格

2.封锁时间，2秒，不可调节

3.出厂定义为可重复触发，无法变更

4.本模块的供电电压是3.3V到15V，极限电压是2.8V到18V

5.模块的输出时序是：上电后输出高电平2秒，然后变为低电平，进入待机状态。若更改延时时间，模块上电后输出高电平的时间会相应增长，可理解为上电后进入正常工作状态的启动时间会增长

6.装上光敏元件后，白天不工作，晚上工作；不装光敏元件，全天工作(默认是没有装光敏元件的)

7.此模块很灵敏，注意安装位置，尽量避开热源辐射源(出风口/阳光直射)

8.本模块灵敏度可调，需要更改一个贴片电阻

9.使用接220V普通烙铁焊接时，需拔下电源后才可进行焊接，建议使用24V低压烙铁

10.如后面接单片机，接灯，请阅读下文的参考电路图

热释电人体红外传感器模块

产品型号：SR-602

感应距离：0-3.5米(峰值5米)

输出：高电平H=3.3V,L=0V

供电直流：3.3V-15V

静态电流：20uA

功能及接口说明

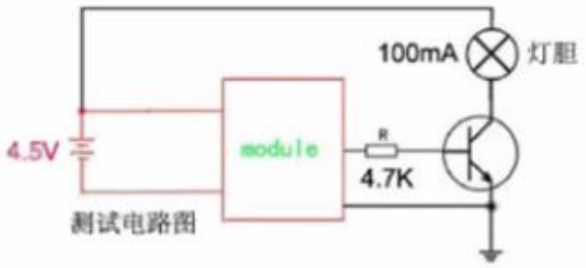
延时时间调节说明

延时曲电阻RL阻值(欧姆) (1%精度)	延时时间(单位秒)
0	2.5
22K	5
39K	8
56K	11
75K	19
91K	35
120K	56
13K	67
154K	130
174K	265
196K	390
221K	522
246K	1050
267K	2096
294K	3150
316M	4200

本模块默认延时为2.5S，上述表格为实验实测得数据，有一定的误差，仅为调试时间范围的参考。若需更改延时时间，请结合实际使用的情况，对具体电路测试后选用适合的参数。

测试及应用参考电路

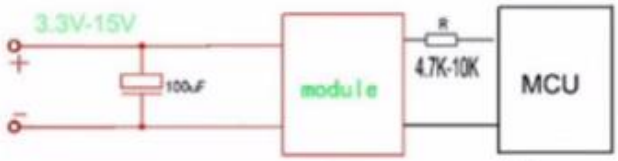
直流电源控制灯泡



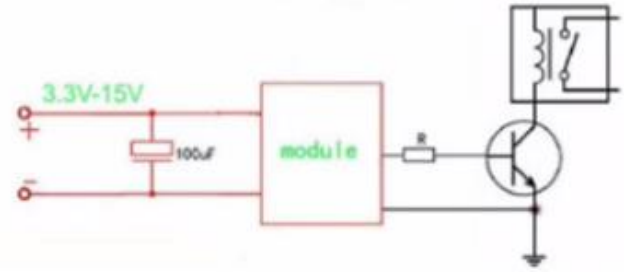
测试电路图

应用简化原理

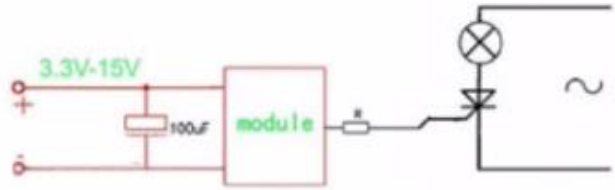
输出接单片机



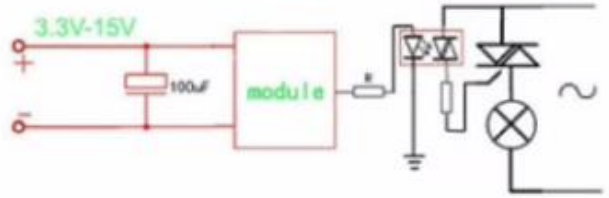
输出通过三极管
控制继电器



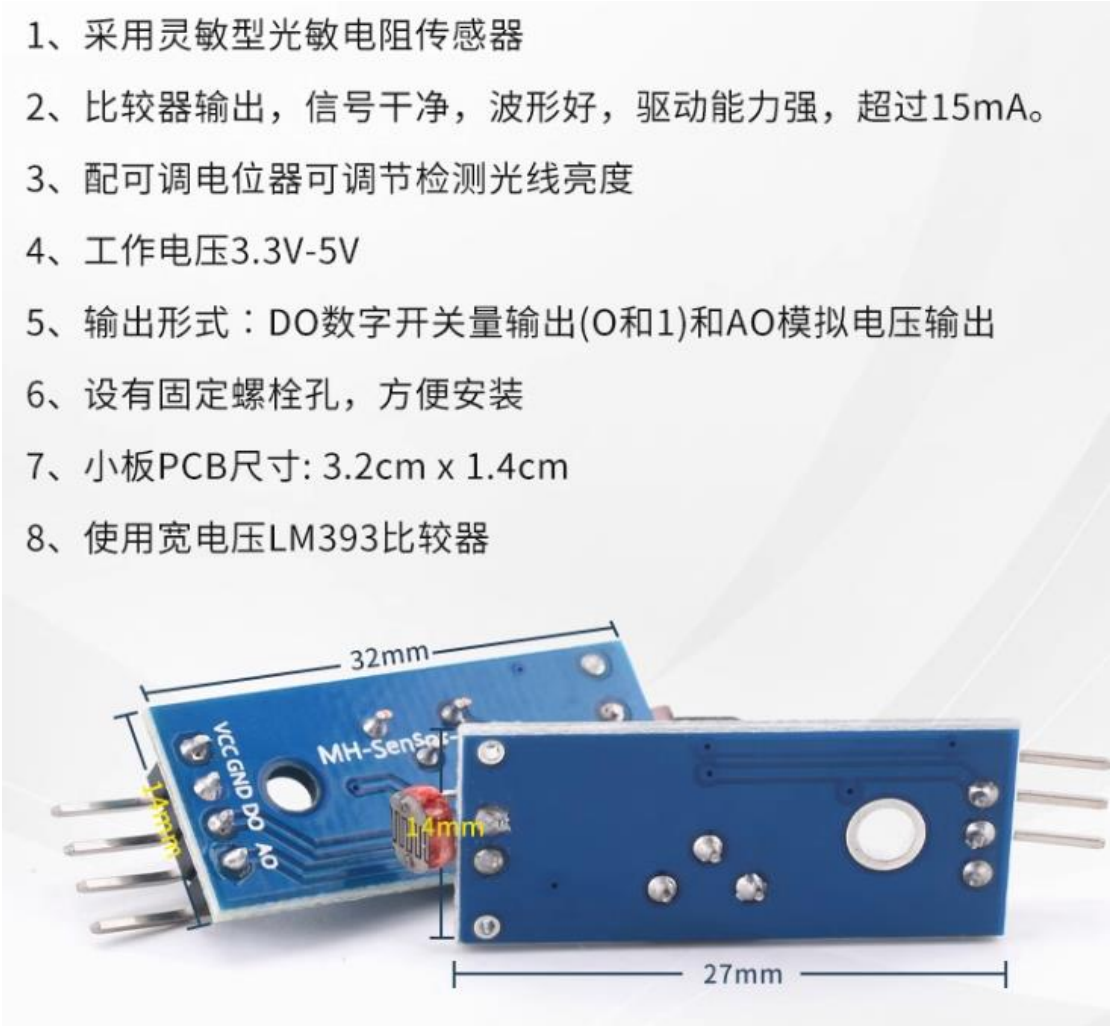
输出通过可控硅
控制交流灯电路



输出通过光耦
控制交流灯电路



六、光敏传感器模块



工作原理

光敏电阻是用硫化隔或硒化隔等半导体材料制成的电阻器，其工作原理是基于内光电效应。随着光照强度的升高，电阻值迅速降低，由于光照产生的载流子都参与导电，在外加电场的作用下作漂移运动，电子奔向电源的正极，空穴奔向电源的负极，从而使光敏电阻器的阻值迅速下降。其在无光照时几乎呈高阻状态，暗电阻很大。

、

使用说明

1、光敏电阻模块对环境光线最敏感，一般用来检测周围环境的光线的亮度，触发单片机或继电器模块等；

2、模块在环境光线亮度达不到设定阈值时，DO 端输出高电平，当外界环境光线亮度超过设定阈值时，DO 端输出低电平；

3、DO 输出端可以与单片机直接相连，通过单片机来检测高低电平由此来检测环境的光线亮度改变；

4、DO 输出端可以直接驱动本店继电器模块，由此可以组成一个光控开关

5、小板模拟量输出 AO 可以和 AD 模块相连，通过 AD 转换，可以获得环境光强更精准的数值；

光敏电阻传感器模块 4针

接口说明:

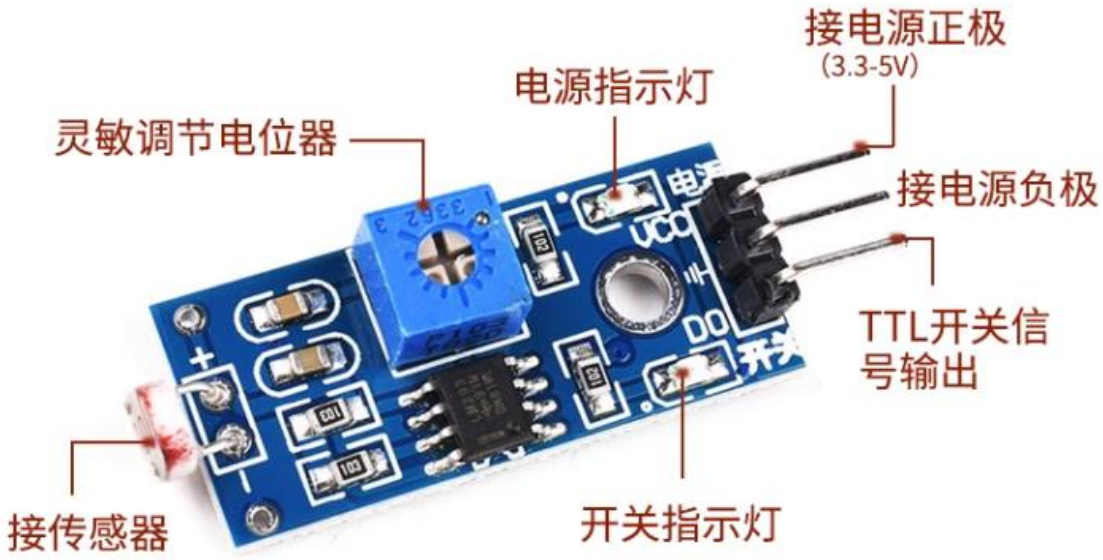
1、 VCC 接电源正极(3.3-5V)

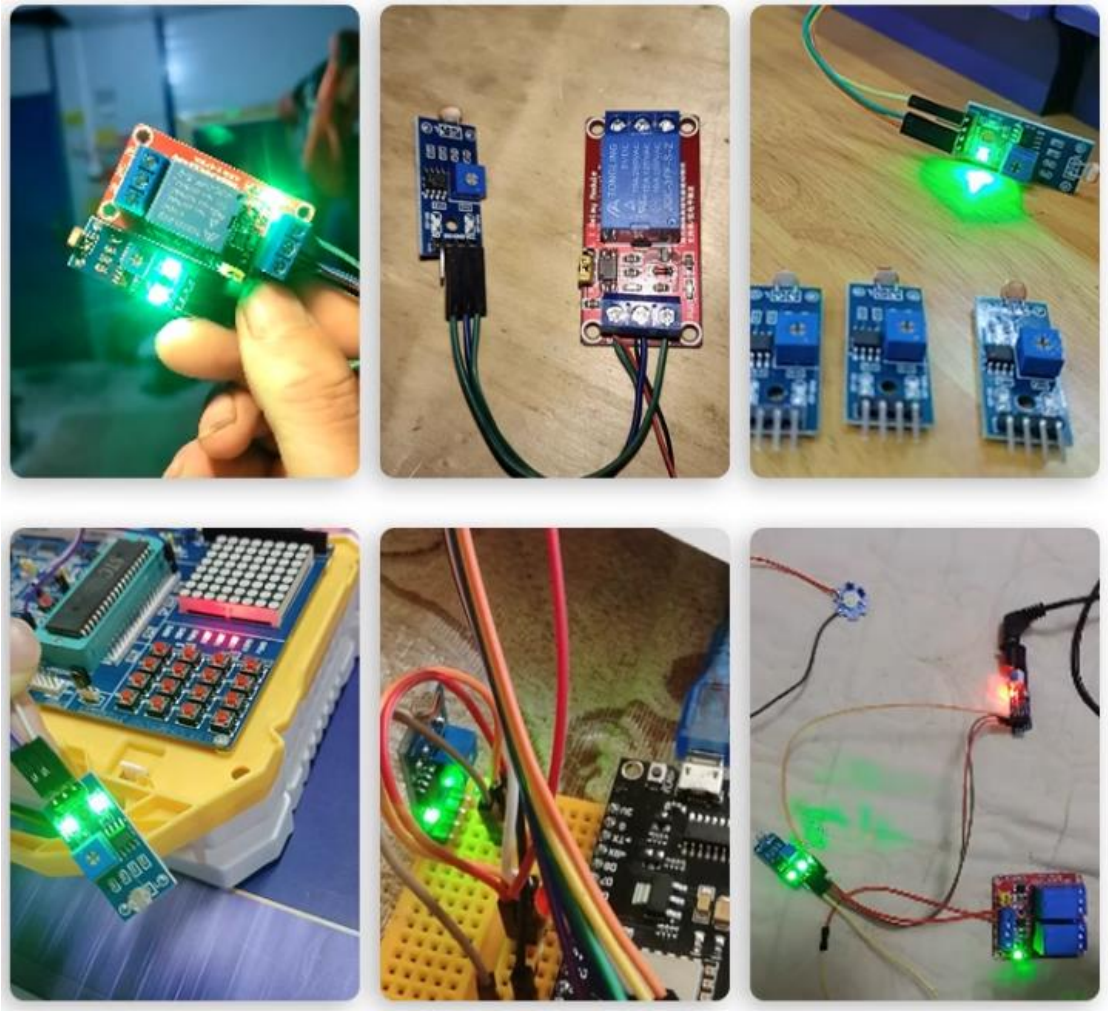
2、GND接电源负极

3、DO TTL数字信号输出

4、AO小板模拟信号输出

光敏电阻传感器模块 3针





七、双向有刷电机电调模块

相比于 L298N 或者二路直流电机控制模块，电调的输出功率更加稳定，动力更足

工作电压：4V--9V（建议2S锂电）

输入：Li Po 2S/Ni-MH/Ni-Cd 4-7电池

额定电流（A）：20A

驱动频率：PWM 2kHz

恒流：20A，25A＜10s脉冲

UBEC输出：电压5.7V，电流1A

注意：如有舵机电流或功率过大的建议加装独立BEC供电，以免造成供电不足

重量：8g

尺寸：L25mm,W28mm,H7mm

适合1/16,1/18，1/24遥控攀爬车军卡半大脚车改装，遥控船，

重量轻的特点也适合mini-Z蚊车改装

功能：前进、倒车、刹车

该ESC可搭配130/180/260/280/380刷电机

调整：电调是定中位区间的，新控需微调一下油门，让电调灯不亮，保持是中位，控的油门微调是TH TRIM

默认为JST头，带开关。

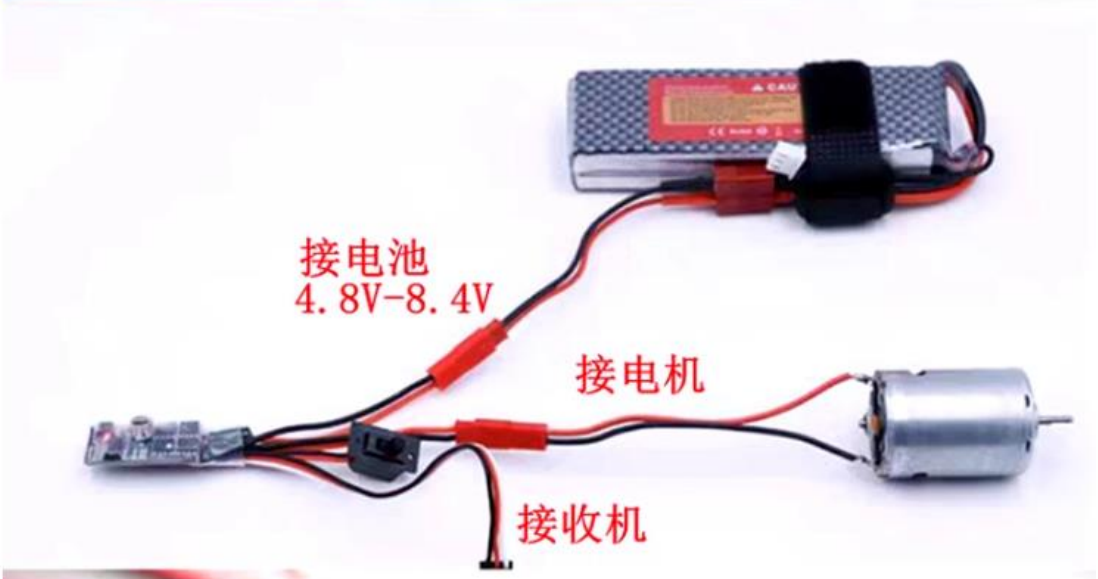
接线：

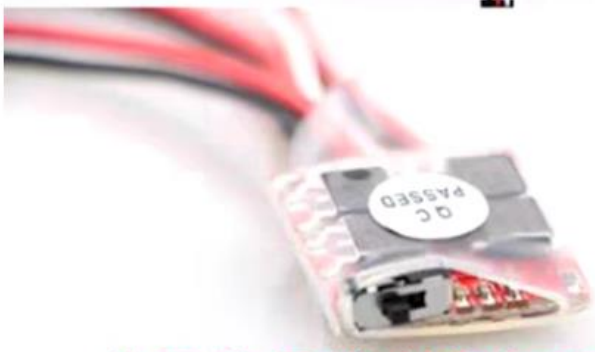
电池 （一定要分红正黑负）

马达（不分正负）

黑红白三线-->接收机的ch2 （一般是黑色靠外，白色靠贴纸）

接线方法





此开关是带刹车与无刹车转换的

注意：电调通电前务必使遥控信号为中间位置，
如有无启动提示音或无法启动，请尝试慢慢微调
摇杆中点位置，直至提示音响起，
建议插摇杆回中通道，例如1，2，4通道，