#Arduino 语法

setup() 初始化函数

loop() 循环体函数

控制语句类似于C

//if if...else for switch case while do... while break continue return goto

扩展语法类似于C

//;（分号） {}（花括号） //（单行注释） /\* \*/（多行注释） #define #include

算数运算符类似于C

//=（赋值运算符） +（加） -（减） \*（乘） /（除） %（模）

比较运算符类似于C

//==（等于） !=（不等于） <（小于） >（大于） <=（小于等于） >=（大于等于）

布尔运算符类似于C

//&&（与） ||（或） !（非）

指针运算符类似于C

//\* 取消引用运算符 & 引用运算符

位运算符类似于C

& (bitwise and) | (bitwise or) ^ (bitwise xor) ~ (bitwise not) << (bitshift left) >> (bitshift right)

复合运算符类似于C

++ (increment) -- (decrement) += (compound addition) -= (compound subtraction) \*= (compound multiplication) /= (compound division) &= (compound bitwise and) |= (compound bitwise or)

常量

constants 预定义的常量

BOOL true false

引脚电压定义，HIGH和LOW【当读取（read）或写入（write）数字引脚时只有两个可能的值： HIGH 和 LOW 】

HIGH（参考引脚）的含义取决于引脚（pin）的设置，引脚定义为INPUT或OUTPUT时含义有所不同。当一个引脚通过pinMode被设置为INPUT，并通过digitalRead读取（read）时。如果当前引脚的电压大于等于3V，微控制器将会返回为HIGH。 引脚也可以通过pinMode被设置为INPUT，并通过digitalWrite设置为HIGH。输入引脚的值将被一个内在的20K上拉电阻 控制 在HIGH上，除非一个外部电路将其拉低到LOW。 当一个引脚通过pinMode被设置为OUTPUT，并digitalWrite设置为HIGH时，引脚的电压应在5V。在这种状态下，它可以 输出电流 。例如，点亮一个通过一串电阻接地或设置为LOW的OUTPUT属性引脚的LED。

LOW的含义同样取决于引脚设置，引脚定义为INPUT或OUTPUT时含义有所不同。当一个引脚通过pinMode配置为INPUT，通过digitalRead设置为读取（read）时，如果当前引脚的电压小于等于2V，微控制器将返回为LOW。 当一个引脚通过pinMode配置为OUTPUT，并通过digitalWrite设置为LOW时，引脚为0V。在这种状态下，它可以 倒灌 电流。例如，点亮一个通过串联电阻连接到+5V，或到另一个引脚配置为OUTPUT、HIGH的的LED。

数字引脚（Digital pins）定义，INPUT和OUTPUT【数字引脚当作 INPUT 或 OUTPUT都可以 。用pinMode()方法使一个数字引脚从INPUT到OUTPUT变化】

Arduino（Atmega）引脚通过pinMode()配置为 输入（INPUT） 即是将其配置在一个高阻抗的状态。配置为INPUT的引脚可以理解为引脚取样时对电路有极小的需求，即等效于在引脚前串联一个100兆欧姆(Megohms)的电阻。这使得它们非常利于读取传感器，而不是为LED供电。

引脚通过pinMode()配置为 输出（OUTPUT） 即是将其配置在一个低阻抗的状态。

这意味着它们可以为电路提供充足的电流。Atmega引脚可以向其他设备/电路提供（提供正电流positive current）或倒灌（提供负电流negative current）达40毫安（mA）的电流。这使得它们利于给LED供电，而不是读取传感器。输出（OUTPUT）引脚被短路的接地或5V电路上会受到损坏甚至烧毁。Atmega引脚在为继电器或电机供电时，由于电流不足，将需要一些外接电路来实现供电。

整数常量

进制 例子 格式 备注

10（十进制） 123 无

2（二进制） B1111011 前缀'B' 只适用于8位的值（0到255）字符0-1有效

8（八进制） 0173 前缀”0” 字符0-7有效

16（十六进制） 0x7B 前缀”0x” 字符0-9，A-F，A-F有效

小数是十进制数。这是数学常识。如果一个数没有特定的前缀，则默认为十进制。

二进制以2为基底，只有数字0和1是有效的。

'u' or 'U' 指定一个常量为无符号型。（只能表示正数和0） 例如: 33u

'l' or 'L' 指定一个常量为长整型。（表示数的范围更广） 例如: 100000L

'ul' or 'UL' 这个你懂的，就是上面两种类型，称作无符号长整型。 例如：32767ul

浮点常量

浮点数 被转换为 被转换为

10.0 10

2.34E5 2.34 \* 10^5 234000

67E-12 67.0 \* 10^-12 0.000000000067

数据类型类似于C

void boolean char unsigned char byte int unsigned int word

long unsigned long float double string - char array String - object array -（数组）

数据类型转换类似于C

char() byte() int() word() long() float()

word()

把一个值转换为word数据类型的值，或由两个字节创建一个字符。

word(x)

word(h, l)

参数

X：任何类型的值

H：高阶（最左边）字节

L：低序（最右边）字节

修饰符类似于C

static volatile const

辅助工具

sizeof()

数字 I/O

pinMode()

将指定的引脚配置成输出或输入【pinMode(pin, mode) pin:要设置模式的引脚 mode:INPUT或OUTPUT】

例子：

ledPin = 13 // LED连接到数字脚13

void setup()

{

pinMode（ledPin，OUTPUT）; //设置数字脚为输出

}

void loop()

{

digitalWrite（ledPin，HIGH）; //点亮LED

delay(1000); // 等待一秒

digitalWrite(ledPin, LOW); // 灭掉LED

delay(1000); //等待第二个

}

digitalWrite()

给一个数字引脚写入HIGH或者LOW。

如果一个引脚已经使用pinMode()配置为OUTPUT模式，其电压将被设置为相应的值，HIGH为5V（3.3V控制板上为3.3V），LOW为0V。

如果引脚配置为INPUT模式，使用digitalWrite()写入HIGH值，将使内部20K上拉电阻（详见数字引脚教程）。写入LOW将会禁用上拉。上拉电阻可以点亮一个LED让其微微亮，如果LED工作，但是亮度很低，可能是因为这个原因引起的。补救的办法是 使用pinMode()函数设置为输出引脚。

注意：数字13号引脚难以作为数字输入使用，因为大部分的控制板上使用了一颗LED与一个电阻连接到他。如果启动了内部的20K上拉电阻，他的电压将在1.7V左右，而不是正常的5V，因为板载LED串联的电阻把他使他降了下来，这意味着他返回的值总是LOW。如果必须使用数字13号引脚的输入模式，需要使用外部上拉下拉电阻。

digitalRead()

digitalRead（PIN）【pin：你想读取的引脚号（int），返回 HIGH 或 LOW】

例子：

ledPin = 13 // LED连接到13脚

int inPin = 7; // 按钮连接到数字引脚7

int val = 0; //定义变量存以储读值

void setup()

{

pinMode(ledPin, OUTPUT); // 将13脚设置为输出

pinMode(inPin, INPUT); // 将7脚设置为输入

}

void loop()

{

val = digitalRead(inPin); // 读取输入脚

digitalWrite(ledPin, val); //将LED值设置为按钮的值

}

模拟 I/O

analogReference()

analogReference(type)

配置用于模拟输入的基准电压（即输入范围的最大值）。选项有:

DEFAULT：默认5V（Arduino板为5V）或3.3伏特（Arduino板为3.3V）为基准电压。

INTERNAL：在ATmega168和ATmega328上以1.1V为基准电压，以及在ATmega8上以2.56V为基准电压（Arduino Mega无此选项）

INTERNAL1V1：以1.1V为基准电压（此选项仅针对Arduino Mega）

INTERNAL2V56：以2.56V为基准电压（此选项仅针对Arduino Mega）

EXTERNAL：以AREF引脚（0至5V）的电压作为基准电压。

type：使用哪种参考类型（DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, 或者 EXTERNAL）

改变基准电压后，之前从anal??ogRead()读取的数据可能不准确。

不要在AREF引脚上使用使用任何小于0V或超过5V的外部电压。如果你使用AREF引脚上的电压作为基准电压，你在调用analogRead()前必须设置参考类型为EXTERNAL。否则，你将会削短有效的基准电压（内部产生）和AREF引脚，这可能会损坏您Arduino板上的单片机。

另外，您可以在外部基准电压和AREF引脚之间连接一个5K电阻，使你可以在外部和内部基准电压之间切换。请注意，总阻值将会发生改变，因为AREF引脚内部有一个32K电阻。这两个电阻都有分压作用。所以，例如，如果输入2.5V的电压，最终在在AREF引脚上的电压将为2.5 \* 32 /（32 + 5）= 2.2V。

analogRead()

从指定的模拟引脚读取数据值。 Arduino板包含一个6通道（Mini和Nano有8个通道，Mega有16个通道），10位模拟数字转换器。这意味着它将0至5伏特之间的输入电压映射到0至1023之间的整数值。这将产生读数之间的关系：5伏特/ 1024单位，或0.0049伏特（4.9 mV）每单位。输入范围和精度可以使用analogReference()改变。 它需要大约100微秒（0.0001）来读取模拟输入，所以最大的阅读速度是每秒10000次。

analogRead（PIN）

引脚：从输入引脚（大部分板子从0到5，Mini和Nano从0到7，Mega从0到15）读取数值，返回从0到1023的整数值

例子：

int analogPin = 3; //电位器（中间的引脚）连接到模拟输入引脚3

//另外两个引脚分别接地和+5 V

int val = 0; //定义变量来存储读取的数值

void setup()

{

serial.begin（9600）; //设置波特率（9600）

}

void loop()

{

val = analogRead（analogPin）; //从输入引脚读取数值

serial.println（val）; //显示读取的数值

}

analogWrite() - PWM

analogWrite（pin,value）

从一个引脚输出模拟值（PWM）。可用于让LED以不同的亮度点亮或驱动电机以不同的速度旋转。analogWrite()输出结束后，该引脚将产生一个稳定的特殊占空比方波，直到下次调用analogWrite()（或在同一引脚调用digitalRead()或digitalWrite()）。PWM信号的频率大约是490赫兹。

在大多数arduino板（ATmega168或ATmega328），只有引脚3，5，6，9，10和11可以实现该功能。在aduino Mega上，引脚2到13可以实现该功能。老的Arduino板（ATmega8）的只有引脚9、10、11可以使用analogWrite()。在使用analogWrite()前，你不需要调用pinMode()来设置引脚为输出引脚。

analogWrite函数与模拟引脚、analogRead函数没有直接关系。

pin：用于输入数值的引脚。

value：占空比：0（完全关闭）到255（完全打开）之间。

例子：

int ledPin = 9; // LED连接到数字引脚9

int analogPin = 3; //电位器连接到模拟引脚3

int val = 0; //定义变量存以储读值

void setup()

{

pinMode（ledPin,OUTPUT）; //设置引脚为输出引脚

}

void loop()

{

val = analogRead（analogPin）; //从输入引脚读取数值

analogWrite（ledPin，val / 4）; // 以val / 4的数值点亮LED（因为analogRead读取的数值从0到1023，而analogWrite输出的数值从0到255）

}

高级 I/O

tone()

在一个引脚上产生一个特定频率的方波（50%占空比）。持续时间可以设定，否则波形会一直产生直到调用noTone()函数。该引脚可以连接压电蜂鸣器或其他喇叭播放声音。

在同一时刻只能产生一个声音。如果一个引脚已经在播放音乐，那调用tone()将不会有任何效果。如果音乐在同一个引脚上播放，它会自动调整频率。

使用tone()函数会与3脚和11脚的PWM产生干扰（Mega板除外）。

注意：如果你要在多个引脚上产生不同的音调，你要在对下一个引脚使用tone()函数前对此引脚调用noTone()函数。

tone(pin, frequency)

tone(pin, frequency, duration)

pin：要产生声音的引脚

frequency: 产生声音的频率，单位Hz，类型unsigned int

duration：声音持续的时间，单位毫秒（可选），类型unsigned long

noTone()

停止由tone()产生的方波。如果没有使用tone()将不会有效果。

noTone(pin)

pin: 所要停止产生声音的引脚

shiftOut()

将一个数据的一个字节一位一位的移出。从最高有效位（最左边）或最低有效位（最右边）开始。依次向数据脚写入每一位，之后时钟脚被拉高或拉低，指示刚才的数据有效。

注意：如果你所连接的设备时钟类型为上升沿，你要确定在调用shiftOut()前时钟脚为低电平，如调用digitalWrite(clockPin, LOW)。

注意：这是一个软件实现；Arduino提供了一个硬件实现的SPI库，它速度更快但只在特定脚有效。

shiftOut(dataPin, clockPin, bitOrder, value)

dataPin：输出每一位数据的引脚(int)

clockPin：时钟脚，当dataPin有值时此引脚电平变化(int)

bitOrder：输出位的顺序，最高位优先或最低位优先

value: 要移位输出的数据(byte)

dataPin和clockPin要用pinMode()配置为输出。 shiftOut目前只能输出1个字节（8位），所以如果输出值大于255需要分两步。

//最高有效位优先串行输出

int data= 500;

//移位输出高字节

shiftOut(dataPin, clock, MSBFIRST, (data >> 8));

//移位输出低字节

shiftOut(data, clock, MSBFIRST, data);

//最低有效位优先串行输出

data = 500;

//移位输出低字节

shiftOut(dataPin, clock, LSBFIRST, data);

//移位输出高字节

shiftOut(dataPin, clock, LSBFIRST, (data >> 8));

例子：

相应电路，查看tutorial on controlling a 74HC595 shift register

//引脚连接到74HC595的ST\_CP

int latchPin = 8;

//引脚连接到74HC595的SH\_CP

int clockPin = 12;

// //引脚连接到74HC595的DS

int dataPin = 11;

void setup() {

//设置引脚为输出

pinMode(latchPin, OUTPUT);

pinMode(clockPin, OUTPUT);

pinMode(dataPin, OUTPUT);

}

void loop() {

//向上计数程序

for(J = 0; J <256; J + +){

//传输数据的时候将latchPin拉低

digitalWrite(latchpin, LOW);

shiftOut(dataPin,clockPin,LSBFIRST,J);

//之后将latchPin拉高以告诉芯片

//它不需要再接受信息了

digitalWrite(latchpin, HIGH);

delay(1000);

}

}

shiftIn()

将一个数据的一个字节一位一位的移入。从最高有效位（最左边）或最低有效位（最右边）开始。对于每个位，先拉高时钟电平，再从数据传输线中读取一位，再将时钟线拉低。

注意：这是一个软件实现；Arduino提供了一个硬件实现的SPI库，它速度更快但只在特定脚有效。

shiftIn(dataPin,clockPin,bitOrder)

dataPin：输出每一位数据的引脚(int)

clockPin：时钟脚，当dataPin有值时此引脚电平变化(int)

bitOrder：输出位的顺序，最高位优先或最低位优先

pulseIn()

读取一个引脚的脉冲（HIGH或LOW）。例如，如果value是HIGH，pulseIn()会等待引脚变为HIGH，开始计时，再等待引脚变为LOW并停止计时。返回脉冲的长度，单位微秒。如果在指定的时间内无脉冲函数返回。

此函数的计时功能由经验决定，长时间的脉冲计时可能会出错。计时范围从10微秒至3分钟。（1秒=1000毫秒=1000000微秒）

pulseIn(pin, value)

pulseIn(pin, value, timeout)

pin:你要进行脉冲计时的引脚号（int）。

value:要读取的脉冲类型，HIGH或LOW（int）。

timeout (可选）：指定脉冲计数的等待时间，单位为微秒，默认值是1秒（unsigned long）

返回：脉冲长度（微秒），如果等待超时返回0（unsigned long）

例子：

int pin = 7;

unsigned long duration;

void setup()

{

pinMode(pin, INPUT);

}

void loop()

{

duration = pulseIn(pin, HIGH);;

}

时间

millis()

返回Arduino开发板从运行当前程序开始的毫秒数。这个数字将在约50天后溢出（归零）

例子：

unsigned long time;

void setup(){

Serial.begin(9600)；

}

void loop(){

serial.print("Time:");

time = millis();

//打印从程序开始到现在的时间

serial.println(time);

//等待一秒钟，以免发送大量的数据

delay(1000);

}

参数 millis 是一个无符号长整数，试图和其他数据类型（如整型数）做数学运算可能会产生错误

micros()

返回 Arduino 开发板从运行当前程序开始的微秒数。这个数字将在约70分钟后溢出（归零）。在 16MHz 的 Arduino 开发板上（比如 Duemilanove 和 Nano），这个函数的分辨率为四微秒（即返回值总是四的倍数）。在 8MHz 的 Arduino 开发板上（比如 LilyPad），这个函数的分辨率为八微秒。

注意 ：每毫秒是1,000微秒，每秒是1,000,000微秒。

例子：

unsigned long time;

void setup(){

Serial.begin(9600);

}

void loop(){

Serial.print（“Time:”）;

time = micros();

//打印从程序开始的时间

Serial.println(time);

//等待一秒钟，以免发送大量的数据

delay(1000);

}

delay()

使程序暂停设定的时间（单位毫秒）。（一秒等于1000毫秒）

参数：ms：暂停的毫秒数（unsigned long）

例子：

ledPin = 13 / / LED连接到数字13脚

void setup()

{

pinMode(ledPin, OUTPUT); // 设置引脚为输出

}

void loop()

{

digitalWrite(ledPin, HIGH); // 点亮LED

delay(1000); // 等待1秒

digitalWrite(ledPin, LOW); // 灭掉LED

delay(1000); // 等待一秒

}

虽然创建一个使用delay()的闪烁LED很简单，并且许多例子将很短的delay用于消除开关抖动，delay()确实拥有很多显著的缺点。在delay函数使用的过程中，读取传感器值、计算、引脚操作均无法执行，因此，它所带来的后果就是使其他大多数活动暂停。其他操作定时的方法请参加millis()函数和它下面的例子。大多数熟练的程序员通常避免超过10毫秒的delay(),除非arduino程序非常简单。

但某些操作在delay()执行时任然能够运行，因为delay函数不会使中断失效。通信端口RX接收到得数据会被记录，PWM(analogWrite)值和引脚状态会保持，中断也会按设定的执行。

delayMicroseconds()

使程序暂停指定的一段时间（单位微秒）。一毫秒等于一千微秒，一秒等于1000000微秒。 目前，能够产生的最大的延时准确值是16383。这可能会在未来的Arduino版本中改变。对于超过几千微秒的延迟，你应该使用delay()代替。

例子：

int outPin = 8; // digital pin 8

void setup()

{

pinMode（outPin，OUTPUT）; //设置为输出的数字管脚

}

void loop()

{

digitalWrite（outPin，HIGH）; //设置引脚高电平

delayMicroseconds(50); // 暂停50微秒

digitalWrite(outPin, LOW); // 设置引脚低电平

delayMicroseconds(50); // 暂停50微秒

}

数学运算

min() max() abs()

constrain(x,a,b)【将一个数约束在一个范围内】

map(value, fromLow, fromHigh, toLow, toHigh)

value:需要映射的值

fromLow:当前范围值的下限

fromHigh:当前范围值的上限

toLow:目标范围值的下限

toHigh:目标范围值的上限

例子：

void setup(){}

void loop()

{

int val = analogRead(0);

val = map(val, 0, 1023, 0, 255);

analogWrite(9, val);

}

数学实现

long map(long x, long in\_min, long in\_max, long out\_min, long out\_max)

{

return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min;

}

pow(base, exponent) sqrt(x)

三角函数

sin() cos() tan()

随机数

randomSeed()【随机数种子】 random()【random(max),random(min, max)】

位操作

lowByte() 取一个变量（例如一个字）的低位（最右边）字节。

highByte() 提取一个字节的高位（最左边的），或一个更长的字节的第二低位。

bitRead() 读取一个数的位。bitRead(x, n) X：想要被读取的数 N：被读取的位，0是最重要（最右边）的位 该位的值（0或1）

bitWrite() 在位上写入数字变量 bitWrite(x, n, b) X：要写入的数值变量 N：要写入的数值变量的位，从0开始是最低（最右边）的位 B：写入位的数值（0或1）

bitSet() 为一个数字变量设置一个位 bitSet(x, n) X：想要设置的数字变量 N：想要设置的位，0是最重要（最右边）的位

bitClear() 清除一个数值型数值的指定位(将此位设置成 0) bitClear(x, n) X：指定要清除位的数值 N：指定要清除位的位置，从0开始，0 表示最右端位

bit() 计算指定位的值（0位是1，1位是2，2位4，以此类推） bit(n) 需要计算的位

设置中断函数

attachInterrupt()

attachInterrupt(interrupt, function, mode)

当发生外部中断时，调用一个指定函数。当中断发生时，该函数会取代正在执行的程序。大多数的Arduino板有两个外部中断：0（数字引脚2）和1（数字引脚3）。

arduino Mege有四个外部中断：数字2（引脚21），3（20针），4（引脚19），5（引脚18）

interrupt：中断引脚数

function：中断发生时调用的函数，此函数必须不带参数和不返回任何值。该函数有时被称为中断服务程序。

mode：定义何时发生中断以下四个contstants预定有效值：

LOW 当引脚为低电平时，触发中断

CHANGE 当引脚电平发生改变时，触发中断

RISING 当引脚由低电平变为高电平时，触发中断

FALLING 当引脚由高电平变为低电平时，触发中断.

当中断函数发生时，delya()和millis()的数值将不会继续变化。当中断发生时，串口收到的数据可能会丢失。你应该声明一个变量来在未发生中断时储存变量。

在单片机自动化程序中当突发事件发生时，中断是非常有用的，它可以帮助解决时序问题。一个使用中断的任务可能会读一个旋转编码器，监视用户的输入。

如果你想以确保程序始终抓住一个旋转编码器的脉冲，从来不缺少一个脉冲，它将使写一个程序做任何事情都要非常棘手，因为该计划将需要不断轮询的传感器线编码器，为了赶上脉冲发生时。其他传感器也是如此，如试图读取一个声音传感器正试图赶上一按，或红外线槽传感器（照片灭弧室），试图抓住一个硬币下降。在所有这些情况下，使用一个中断可以释放的微控制器来完成其他一些工作。

例子：

int pin = 13;

volatile int state = LOW;

void setup()

{

pinMode(pin, OUTPUT);

attachInterrupt(0, blink, CHANGE);

}

void loop()

{

digitalWrite(pin, state);

}

void blink()

{

state = !state;

}

detachInterrupt()

关闭给定的中断

detachInterrupt(interrupt)

interrupt: 中断禁用的数(0或者1)

开关中断

interrupts()

重新启用中断（使用noInterrupts()命令后将被禁用）。中断允许一些重要任务在后台运行，默认状态是启用的。禁用中断后一些函数可能无法工作，并传入信息可能会被忽略。中断会稍微打乱代码的时间，但是在关键部分可以禁用中断

noInterrupts()

禁止中断（重新使能中断interrupts()）。中断允许在后台运行一些重要任务，默认使能中断。禁止中断时部分函数会无法工作，通信中接收到的信息也可能会丢失。

中断会稍影响计时代码，在某些特定的代码中也会失效

例子：

void setup()

void loop()

{

noInterrupts（）;

//关键的、时间敏感的代码放在这

interrupts();

//其他代码放在这

}

通讯

Serial

用于Arduino控制板和一台计算机或其他设备之间的通信。所有的Arduino控制板有至少一个串口(又称作为UART或USART)。它通过0(RX)和1(TX)数字引脚经过串口转换芯片连接计算机USB端口与计算机进行通信。因此，如果你使用这些功能的同时你不能使用引脚0和1作为输入或输出。

您可以使用Arduino IDE内置的串口监视器与Arduino板通信。点击工具栏上的串口监视器按钮，调用begin()函数（选择相同的波特率）。

Arduino Mega 有三个额外的串口：Serial 1使用19(RX)和18(TX)，Serial 2使用17(RX)和16(TX)，Serial3使用15(RX)和14(TX)。 若要使用这三个引脚与您的个人电脑通信，你需要一个额外的USB转串口适配器，因为这三个引脚没有连接到Mega上的USB转串口适配器。若要用它们来与外部的TTL串口设备进行通信，将TX引脚连接到您的设备的RX引脚，将RX引脚连接到您的设备的TX引脚，将GND连接到您的设备的GND。（不要直接将这些引脚直接连接到RS232串口;他们的工作电压在+/- 12V，可能会损坏您的Arduino控制板。）Arduino Leonardo板使用Serial 1通过0(RX)和1(TX)与viaRS-232通信，。Serial预留给使用Mouse and Keyboard libarariies的USB CDC通信 。更多信息，请参考Leonardo 开始使用页和硬件页。

函数：

if (Serial)

表示指定的串口是否准备好

在Leonardo上，if(Serial)表示不论有无USB CDC，串行连接都是开放的。对于所有其他的情况，包括Leonardo上的if(Serial1)，将一直返回true。 这来自于 Arduino 1.0.1 版本的介绍

对于所有的arduino板: if (Serial)

Arduino Leonardo 特有: if (Serial1)

Arduino Mega 特有:

if (Serial1)

if (Serial2)

if (Serial3)

available()

获取从串口读取有效的字节数（字符）。这是已经传输到，并存储在串行接收缓冲区（能够存储64个字节）的数据。 available()继承了 Stream类

Serial.available()

仅适用于Arduino Mega ：

Serial1.available()

Serial2.available()

Serial3.available()

例子：

incomingByte = 0; //传入的串行数据

void setup() {

Serial.begin(9600); // 打开串行端口，设置传输波特率为9600 bps

}

void loop() {

//只有当你接收到数据时才会发送数据，：

if (Serial.available() > 0) {

//读取传入的字节：

incomingByte = Serial.read();

//显示你得到的数据：

Serial.print("I received: ");

Serial.println(incomingByte, DEC);

}

}

begin()

将串行数据传输速率设置为位/秒（波特）。与计算机进行通信时，可以使用这些波特率：300，1200，2400，4800，9600，14400，19200，28800，38400，57600或115200。当然，您也可以指定其他波特率 - 例如，引脚0和1和一个元件进行通信，它需要一个特定的波特率

Serial.begin(speed) 仅适用于Arduino Mega : Serial1.begin(speed) Serial2.begin(speed) Serial3.begin(speed)

speed: 位/秒 (波特) - long

end()

停用串行通信，使RX和TX引脚用于一般输入和输出。要重新使用串行通信, 需要 Serial.begin()语句

Serial.end()

仅适用于Arduino Mega: Serial1.end() Serial2.end() Serial3.end()

find()

Serial.find() 从串行缓冲器中读取数据，直到发现给定长度的目标字符串。如果找到目标字符串，该函数返回true，如果超时则返回false

Serial.flush() 继承了 Stream 类

Serial.find(target)

target : 要搜索的字符串（字符）

findUntil()

Serial.findUntil()从串行缓冲区读取数据，直到找到一个给定的长度或字符串终止位。

如果目标字符串被发现，该函数返回true，如果超时则返回false。

Serial.findUntil()继承了Stream类。

Serial.findUntil(target, terminal)

target : 要搜索的字符串(char) terminal : 在搜索中的字符串终止位 (char)

flush()

等待超出的串行数据完成传输。（在1.0及以上的版本中，flush()语句的功能不再是丢弃所有进入缓存器的串行数据。）

flush()继承了Stream类

Serial.flush()

仅 Arduino Mega 可以使用的语法：

Serial1.flush()

Serial2.flush()

Serial3.flush()

parseFloat()

查找传入的串行数据流中的下一个有效的浮点数。 parseFloat()继承了Stream类

Serial.parseFloat()

仅适用于Arduino Mega：

Serial1.parseFloat()

Serial2.parseFloat()

Serial3.parseFloat()

parseInt()

查找传入的串行数据流中的下一个有效的整数。 parseInt()继承了Stream类

Serial.parseInt()

仅适用于Arduino Mega：

Serial1.parseInt()

Serial2.parseInt()

Serial3.parseInt()

peek()

返回传入的串行数据的下一个字节（字符），而不是进入内部串行缓冲器调取。也就是说，连续调用 peek()将返回相同的字符，与调用read()方法相同。peek()继承自 Stream类

Serial.peek()

仅适用于Arduino Mega :

Serial1.peek()

Serial2.peek()

Serial3.peek()

print()

以人们可读的ASCII文本形式打印数据到串口输出。此命令可以采取多种形式。每个数字的打印输出使用的是ASCII字符。浮点型同样打印输出的是ASCII字符，保留到小数点后两位。Bytes型则打印输出单个字符。字符和字符串原样打印输出。Serial.print()打印输出数据不换行，Serial.println()打印输出数据自动换行处理。例如

Serial.print(78)输出为“78”

Serial.print(1.23456)输出为“1.23”

Serial.print(“N”)输出为“N”

Serial.print(“Hello world.”)输出为“Hello world.”

也可以自己定义输出为几进制（格式）；可以是BIN（二进制，或以2为基数），OCT（八进制，或以8为基数），DEC（十进制，或以10为基数），HEX（十六进制，或以16为基数）。对于浮点型数字，可以指定输出的小数数位。例如

Serial.print(78,BIN)输出为“1001110”

Serial.print(78,OCT)输出为“116”

Serial.print(78,DEC)输出为“78”

Serial.print(78,HEX)输出为“4E”

Serial.println(1.23456,0)输出为“1”

Serial.println(1.23456,2)输出为“1.23”

Serial.println(1.23456,4)输出为“1.2346”

你可以通过基于闪存的字符串来进行打印输出，将数据放入F()中，再放入Serial.print()。例如 Serial.print(F(“Hello world”)) 若要发送一个字节，则使用 Serial.write()

Serial.print(val)

Serial.print(val，格式)

val：打印输出的值 - 任何数据类型

格式：指定进制（整数数据类型）或小数位数（浮点类型）

例子：

int x = 0; // 定义一个变量并赋值

void setup() {

Serial.begin(9600); // 打开串口传输，并设置波特率为9600

}

void loop() {

//打印标签

Serial.print("NO FORMAT"); // 打印一个标签

Serial.print("\t"); // 打印一个转义字符

Serial.print("DEC");

Serial.print("\t");

Serial.print("HEX");

Serial.print("\t");

Serial.print("OCT");

Serial.print("\t");

Serial.print("BIN");

Serial.print("\t");

for(x=0; x< 64; x++){ // 打印ASCII码表的一部分, 修改它的格式得到需要的内容

//打印多种格式：

Serial.print(x); // 以十进制格式将x打印输出 - 与 "DEC"相同

Serial.print("\t"); // 横向跳格

Serial.print(x, DEC); // 以十进制格式将x打印输出

Serial.print("\t"); // 横向跳格

Serial.print(x, HEX); // 以十六进制格式打印输出

Serial.print("\t"); // 横向跳格

Serial.print(x, OCT); // 以八进制格式打印输出

Serial.print("\t"); // 横向跳格

Serial.println(x, BIN); // 以二进制格式打印输出

// 然后用 "println"打印一个回车

delay(200); // 延时200ms

}

Serial.println(""); // 打印一个空字符，并自动换行

}

println()

打印数据到串行端口，输出人们可识别的ASCII码文本并回车 (ASCII 13, 或 '\r') 及换行(ASCII 10, 或 '\n')。此命令采用的形式与Serial.print ()相同

Serial.println(val)

Serial.println(val, format)

val: 打印的内容 - 任何数据类型都可以

format: 指定基数（整数数据类型）或小数位数（浮点类型）

例子：

int analogValue = 0; // 定义一个变量来保存模拟值

void setup() {

//设置串口波特率为9600 bps：

Serial.begin(9600);

}

void loop() {

//读取引脚0的模拟输入：

analogValue = analogRead（0）;

//打印g各种格式：

Serial.println(analogValue); //打印ASCII编码的十进制

Serial.println(analogValue, DEC); //打印ASCII编码的十进制

Serial.println(analogValue, HEX); //打印ASCII编码的十六进制

Serial.println(analogValue, OCT); //打印ASCII编码的八进制

Serial.println(analogValue, BIN); //打印一个ASCII编码的二进制

// 延时10毫秒：

delay(10);

}

read()

读取传入的串口的数据。read() 继承自 Stream 类

serial.read()

Arduino Mega独有:

erial1.read()

serial2.read()

serial3.read()

例子：

int incomingByte = 0; // 传入的串行数据

void setup() {

Serial.begin(9600); // 打开串口，设置数据传输速率9600

}

void loop() {

// 当你接收数据时发送数据

if (Serial.available() > 0) {

// 读取传入的数据:

incomingByte = Serial.read();

//打印你得到的：

Serial.print("I received: ");

Serial.println(incomingByte, DEC);

}

}

readBytes()

Serial.readBytes()从串口读字符到一个缓冲区。如果预设的长度读取完毕或者时间到了 (参见 Serial.setTimeout())，函数将终止.

Serial.readBytes()返回放置在缓冲区的字符数。返回0意味着没有发现有效的数据。

Serial.readBytes()继承自 Stream 类

Serial.readBytes(buffer, length)

buffer：用来存储字节（char[]或byte[]）的缓冲区

length：读取的字节数（int）

readBytesUntil()

Serial.readBytesUntil()将字符从串行缓冲区读取到一个数组。如果检测到终止字符，或预设的读取长度读取完毕，或者时间到了 (参见 Serial.setTimeout())函数将终止。

Serial.readBytesUntil()返回读入数组的字符数。返回0意味着没有发现有效的数据。

Serial.readBytesUntil()继承自 Stream类

Serial.readBytesUntil(character, buffer, length)

character ：要搜索的字符（char）

buffer ：缓冲区来存储字节（char[]或byte[]）

length:读的字节数（int）

setTimeout()

Serial.setTimeout()设置使用Serial.readBytesUntil() 或Serial.readBytes()时等待串口数据的最大毫秒值. 默认为1000毫秒。

Serial.setTimeout()继承自Stream 类

Serial.setTimeout(time)

time ：以毫秒为单位的超时时间（long）

write()

写入二级制数据到串口。发送的数据以一个字节或者一系列的字节为单位。如果写入的数字为字符，需使用print()命令进行代替

Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

Arduino Mega还支持：Serial1，Serial2，Serial3 （替代Serial）

val: 以单个字节形式发的值

str: 以一串字节的形式发送的字符串

buf: 以一串字节的形式发送的数组

len: 数组的长度

返回结果：

byte

write() 将返回写入的字节数，但是否使用这个数字是可选的

SerialEvent()

Stream

USB（仅适用于 Leonardo 和 Due）

键盘

鼠标