# Arduino Hardware Serial(UART & USART) Function

## 说明

串行端口用于Arduino和个人电脑或其他设备进行通信。所有Arduino控制器都有至少一个串行端口（也称为UART或者USART）。个人电脑可以通过USB端口与Arduino的引脚0(RX)和引脚1(TX) 进行通信。所以当Arduino的引脚0和引脚1用于串行通信功能时，Arduino的引脚0和引脚1是不能做其他用的。你也可以通过Arduino开发环境软件中的串口监视器来与Arduino 控制器进行串口通信，你只需要点击Arduino IDE软件中的“串口监视器”按钮就可以打开串口监视器。

| BOARD | USB CDC NAME | SERIAL PINS | SERIAL1 PINS | SERIAL2 PINS | SERIAL3 PINS |
|---|---|---|---|---|---|
| Uno, Nano, Mini | | 0(RX), 1(TX) | | | |
| Mega | | 0(RX), 1(TX) | 19(RX), 18(TX) | 17(RX), 16(TX) | 15(RX), 14(TX) |
| Leonardo, Micro, Yún | Serial | | 0(RX), 1(TX) | | |
| Uno WiFi Rev.2 | | Connected to USB | 0(RX), 1(TX) | Connected to NINA | |
| MKR boards | Serial | | 13(RX), 14(TX) | | |
| Zero | SerialUSB (Native USB Port only) | Connected to Programming Port | 0(RX), 1(TX) | | |
| Due | SerialUSB (Native USB Port only) | 0(RX), 1(TX) | 19(RX), 18(TX) | 17(RX), 16(TX) | 15(RX), 14(TX) |
| 101 | Serial | | 0(RX), 1(TX) | | |

# 常见问题

1. 串口只能和一个设备通讯

   串口在arduino中使用时（以UNO为例），和电脑通讯的串口时用0(RX)和1(TX)的，所以不能在arduino 的这两个串口上使用和其它设备的串口通讯。

# 函数

# if(Serial)

## 介绍

指示是否已准备好指定的串口。

在带有本机USB的单板上，if(Serial)表示USB CDC串口连接是否打开。对于所有其他板，和非usb CDC端口，这将始终返回true。

这是在*Arduino IDE 1.0.1*中引入的。

## 语法

```
if (Serial);
```

## 参数

None

## 返回值

如果指定的串口是可用的，则返回true。如果在准备好之前查询Leonardo的USB CDC串行连接，这只会返回false。数据类型:bool。

## 例程

```
void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
}

void loop() {
  //proceed normally
}
```

# available

## 说明

available() 函数可用于检查设备是否接收到数据。该函数将会返回等待读取的数据字节数。(这是已经到达并存储在串行接收缓冲区(最多64字节)中的数据。)

available()函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
stream.available();
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
Serial.available();
wifiClient.available();
```

### 参数

无

### 返回值

等待读取的数据字节数。
返回值数据类型：int

### 例程

```
void setup() {
  // 启动串口通讯
  Serial.begin(9600);
  Serial.println();
}

void loop() {

  if (Serial.available()){                        // 当串口接收到信息后
    Serial.println("Serial Data Available..."); // 通过串口监视器通知用户

    String serialData = Serial.readString();    // 将接收到的信息使用readString()存储于serialData变量
    Serial.print("Received Serial Data: ");     // 然后通过串口监视器输出serialData变量内容
    Serial.println(serialData);                 // 以便查看serialData变量的信息
  }
}
```

## availableForWrite

### 说明

在不阻塞写操作的情况下，获取可在串行缓冲区中写入的字节数(characters)。

### 语法

```
Serial.availableForWrite()
```

### 参数

Serial:串口对象。

### 返回值

可以写入的数据字节数。
返回值数据类型：int

# begin()

## 说明

设置串行数据传输的数据速率，单位为每秒比特(波特)。为了与串行监视器通信，请确保使用屏幕右下角菜单中列出的波特率之一。然而，您可以指定其他速率—例如，通过需要特定波特率的组件在引脚0和1上进行通信。

第二个可选参数配置数据位、奇偶校验位和停止位。默认是8个数据位，没有奇偶校验，一个停止位。

## 语法

```
Serial.begin(speed);
Serial.begin(speed, config);
```

## 参数

Serial:串口对象
speed: in bits per second (baud). Allowed data types: long.
config: sets data, parity, and stop bits. Valid values are:

```
SERIAL_5N1`
`SERIAL_6N1`
`SERIAL_7N1`
`SERIAL_8N1` (the default)
`SERIAL_5N2`
`SERIAL_6N2`
`SERIAL_7N2`
`SERIAL_8N2`
`SERIAL_5E1`: even parity
`SERIAL_6E1`
`SERIAL_7E1`
`SERIAL_8E1`
`SERIAL_5E2`
`SERIAL_6E2`
`SERIAL_7E2`
`SERIAL_8E2`
`SERIAL_5O1`: odd parity
`SERIAL_6O1`
`SERIAL_7O1`
`SERIAL_8O1`
`SERIAL_5O2`
`SERIAL_6O2`
`SERIAL_7O2`
`SERIAL_8O2
```

## 返回值

无

# Serial.end()

## 说明

终止串行通讯，让RX 和 TX引脚用于Arduino的输入（INPUT）或输出(OUTPUT)功能。可调用Serial.begin()重新打开串行通讯。

## 语法

```
Serial.end();
```

## 参数

无

## 返回值

无

# find

## 说明

find函数可用于从设备接收到的数据中寻找指定字符串信息。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
Serial.find(target);
Serial.find(target, length);
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.find(target);
`wifiClient.find(target);
```

## 参数

Serial:串口对象

target: 被查找字符串。允许使用String或char类型。

length: 目标长度。 允许使用: size_t.

## 返回值

返回值类型为bool。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

### 例程

```
void setup() {
  // 启动串口通讯
  Serial.begin(9600);
  Serial.println();
}

void loop() {
  if (Serial.available()){                        // 当串口接收到信息后
    Serial.println("Serial Data Available..."); // 通过串口监视器通知
                                                  // 用户系统开始查找指定信息
    Serial.print("system is trying to find "); Serial.println("^_^");

    // 执行查找并通过串口监视器输出查找结果
    if(Serial.find("^_^")) {
      Serial.print("Great! System found "); Serial.println("^_^");
    } else {
      Serial.print("Sorry System can't find "); Serial.println("^_^");
    }
    Serial.println("");
  }
}
```

## findUntil

### 说明

findUntil函数可用于从设备接收到的数据中寻找指定字符串信息。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。该函数在满足以下任一条件后都会停止函数执行

– 读取到指定终止字符串
– 找到了指定字符串信息
– 达到设定时间（可使用setTimeout来设置）

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
Serial.findUntil(target, terminator)`;
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.findUntil(target, terminator)`
`wifiClient.findUntil(target, terminator)
```

### 参数

Serial:串口对象。

target: 被查找字符串。允许使用cString或char类型。

terminator: 终止字符串。用于设置终止函数执行的字符串信息。设备在读取数据时一旦读取到此终止字符串，将会结束函数执行并返回。

### 返回值

返回值类型为bool。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

## flush

### 说明

flush函数可让开发板在所有待发数据发送完毕前，保持等待状态。

请注意：很多人误认为flush函数具有清除开发板接收缓存区的功能。事实上此函数是没有此功能的。如需了解如何清除开发板接收缓存区内信息的方法，请点击这里进入相应说明页面。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

为了更好的理解flush函数的作用，我们在这里用Serial.flush()作为示例讲解。

当我们通过Serial.print或Serial.println来发送数据时，被发送的字符数据将会存储于开发板的“发送缓存”中。这么做的原因是开发板串行通讯速率不是很高，如果发送数据较多，发送时间会比较长。

在没有使用flush函数的情况下，开发板不会等待所有“发送缓存”中数据都发送完毕再执行后续的程序内容。也就是说，开发板是在后台发送缓存中的数据。程序运行不受影响。

相反的，在使用了flush函数的情况下，开发板是会等待所有“发送缓存”中数据都发送完毕以后，再执行后续的程序内容.

### 语法

```
Serial.flush()`
/*注：此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.flush()`
`wifiClient.flush()
```

### 参数

Serial:串口对象。

### 返回值

无

### 例程

没有使用flush函数的情况下，通过串口监视器显示开发板输出一大串字符的运行效果。

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

  // 记录输出串口信息前的millis时间
  unsigned long millisNoFlushStart = millis();

  // 通过串口输出信息
  Serial.println(F("abcdefghijklmnopqrstuvwxyz"));

  // 记录输出串口信息后的millis时间
```

```
  unsigned long millisNoFlushStop = millis();

  // 通过串口监视器输出没有使用flush函数情况下，输出信息前后的时间差。
  Serial.print(F("NO flush: "));
  Serial.print( millisNoFlushStop - millisNoFlushStart);
  Serial.println(F(" milliseconds."));
}
void loop() {}
```

使用flush函数的情况下，通过串口监视器显示开发板输出一大串字符的运行效果。

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

  // 记录输出串口信息前的millis时间
  unsigned long millisWithFlushStart = millis();

  // 通过串口输出信息
  Serial.println(F("abcdefghijklmnopqrstuvwxyz"));

  // 使用flush函数，让开发板在完成串口输出信息以前"原地等待"
  Serial.flush();

  // 记录输出串口信息后的millis时间
  unsigned long millisWithFlushStop = millis();

  // 通过串口监视器输出没有使用flush函数情况下，输出信息前后的时间差。
  Serial.print(F("WITH flush: "));
  Serial.print( millisWithFlushStop - millisWithFlushStart);
  Serial.println(F(" milliseconds."));
}

void loop() {
}
```

# parseFloat

## 说明

parsefloat()从Serial缓冲区返回第一个有效的浮点数。parseFloat()以第一个不是浮点数的字符结束。如果超时，函数将终止(请参阅Serial.setTimeout())。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
Serial.parseFloat()
Serial.parseFloat(lookahead)
Serial.parseFloat(lookahead, ignore)
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.parseFloat()`
`wifiClient.parseFloat()
```

### 参数

Serial:串口对象。

lookahead:用于在流中前向查找浮点数的模式。允许的数据类型:LookaheadMode。允许的lookahead:

- SKIP_ALL:在扫描流查找浮点数时，忽略减号、小数点或数字以外的所有字符。这是默认模式。
- SKIP_NONE:不跳过任何内容，除非第一个等待的字符有效，否则流不会被接触。
- SKIP_WHITESPACE:只跳过制表符、空格、换行和回车。

ignore:在搜索中跳过指定的字符。例如，用来跳过数千个除数。允许的数据类型:char

### 返回值

在输入信息中找到浮点数值。类型：float

### 例程

```
void setup() {

  // 启动串口通讯
  Serial.begin(9600);
  Serial.println();
}

void loop() {

  if (Serial.available()){              // 当串口接收到信息后
    float serialData = Serial.parseFloat(); // 使用parseFloat查找接收到的信息中的整数
    Serial.print("serialData = ");       // 然后通过串口监视器输出找到的数值
    Serial.println(serialData);
  }
}
```

## parseInt

### 说明

parsefloat()从Serial缓冲区返回第一个有效整数数值。如果超时，函数将终止(请参阅Serial.setTimeout())。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
Serial.parseInt()
Serial.parseInt(lookahead)
Serial.parseInt(lookahead, ignore)
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.parseInt()`
`wifiClient.parseInt()
```

### 参数

Serial:串口对象。

lookahead:用于在流中前向查找浮点数的模式。允许的数据类型:LookaheadMode。允许的lookahead:

- SKIP_ALL:在扫描流查找浮点数时，忽略减号、小数点或数字以外的所有字符。这是默认模式。
- SKIP_NONE:不跳过任何内容，除非第一个等待的字符有效，否则流不会被接触。
- SKIP_WHITESPACE:只跳过制表符、空格、换行和回车。

ignore:在搜索中跳过指定的字符。例如，用来跳过数千个除数。允许的数据类型:char

### 返回值

在输入信息中找到浮点数值。类型：long

### 例程

```
void setup() {
  // 启动串口通讯
  Serial.begin(9600);
  Serial.println();
}

void loop() {

  if (Serial.available()){              // 当串口接收到信息后
    int serialData = Serial.parseInt(); // 使用parseInt查找接收到的信息中的整数
    Serial.print("serialData = ");      // 然后通过串口监视器输出找到的数值
    Serial.println(serialData);
  }

```

## peek

### 说明

返回传入串行数据的下一个字节(字符)，而不将其从内部串行缓冲区中删除。也就是说，连续调用peek()将返回相同的字符，下一次调用read()也将返回相同的字符。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
Serial.peek()`
/*注：此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.peek()`
`wifiClient.peek()
```

### 参数

Serial:串口对象。

### 返回值

设备没有接收到数据时，返回值为-1

设备接收到数据时，返回值为接收到的数据流中的第1个字符。 (数据类型: int)

## print()

### 说明

以人类可读的ASCII码形式向串口发送数据，该函数有多种格式。整数的每一数位将以ASCII码形式发送。浮点数同样以ASCII码形式发送，默认保留小数点后两位。字节型数据将以单个字符形式发送。字符和字符串会以其相应的形式发送。例如：

- Serial.print(78) 发送 “78”
- Serial.print(1.23456) 发送 “1.23”
- Serial.print(‘N’) 发送 “N”
- Serial.print(“Hello world.”) 发送 “Hello world.”

此指令也可以通过附加参数来指定数据的格式。这个允许的值为：BIN (binary二进制), OCT (octal八进制), DEC (decimal十进制), HEX (hexadecimal十六进制)。对于浮点数，该参数可以指定小数点的位数。例如：

- Serial.print(78, BIN) 发送 “1001110”
- Serial.print(78, OCT) 发送 “116”
- Serial.print(78, DEC) 发送 “78”
- Serial.print(78, HEX) 发送 “4E”
- Serial.println(1.23456, 0) 发送 “1”
- Serial.println(1.23456, 2) 发送 “1.23”
- Serial.println(1.23456, 4) 发送 “1.2346”

你可以用F()把待发送的字符串包装到flash存储器。例如：
Serial.print(F(“Hello World”))

要发送单个字节数据，请使用Serial.write()。

### 语法

```
Serial.print(val);
Serial.print(val, format);
```

## 参数

Serial:串口对象。

val: 要发送的数据（任何数据类型）

format: 指定数字的基数（用于整型数）或者小数的位数（用于浮点数）。

## 返回值

size_t (long): print()返回发送的字节数（可丢弃该返回值）。

## 例程

```
int x = 0;    // 变量

void setup() {
  Serial.begin(9600);      // 打开串口通讯
}

void loop() {
  // print labels
  Serial.println("NO");       // 打印文字标志
  Serial.print("Format");       // 打印文字标志
  Serial.print("\t");

  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++){

    //通过不同格式显示
    Serial.print(x);       // 输出ASCII编码的十进制数字。与"DEC"相同
    Serial.print("\t");

    Serial.print(x, DEC);  // 输出ASCII编码的十进制数字。
    Serial.print("\t");

    Serial.print(x, HEX);  // 输出ASCII编码的十六进制数字。
    Serial.print("\t");

    Serial.print(x, OCT);  // 输出ASCII编码的八进制数字
    Serial.print("\t");

    Serial.println(x, BIN);  // 输出ASCII编码的二进制数字，然后换行
    //
    delay(200);            //延迟200毫秒
  }c
  Serial.println("");      // 再次换行
```

```
45 }
```

### 运行结果

程序运行后，串口监视器显示结果如下：

```
NO
FORMAT  DEC     HEX     OCT     BIN
0       0       0       0       0
1       1       1       1       1
2       2       2       2       10
3       3       3       3       11
4       4       4       4       100
5       5       5       5       101
6       6       6       6       110
7       7       7       7       111
8       8       8       10      1000
9       9       9       11      1001
10      10      A       12      1010
11      11      B       13      1011
12      12      C       14      1100
```

## println()

返回 串口通讯指令目录页

### 说明

将数据以人类可读的ASCII文本形式输出到串口，后跟一个回车字符(ASCII 13，或'\r')和一个换行字符(ASCII 10，或'\n')。该命令采用与Serial.print()相同的形式。

### 语法

```
1 Serial.println(val);
2 Serial.println(val, format);
```

### 参数

Serial:串口对象。

val: 要发送的数据（任何数据类型）

format: 指定数字的数据形式或小数的位数（用于浮点数）。

### 返回值

size_t (long): println()返回发送的字节数（可丢弃该返回值）。

# read

## 说明

read() 函数可用于从设备接收到数据中读取一个字节的数据。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
Serial.read()`
/*此处Serial为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如*/
`Serial.read()`
`wifiClient.read()
```

## 参数

Serial:串口对象。

## 返回值

设备没有接收到数据时，返回值为-1
设备接收到数据时，返回值为接收到的数据流中的1个字符。(数据类型：int)

# readBytes

## 说明

readBytes函数可用于从设备接收的数据中读取信息。读取到的数据信息将存放在缓存变量中。该函数在读取到指定字节数的信息或者达到设定时间后都会停止函数执行并返回。该设定时间可使用setTimeout来设置。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
stream.readBytes(buffer, length)`
/*注：此处stream为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：
`Serial.readBytes(buffer, length)`
`wifiClient.readBytes(buffer, length)*/
```

## 参数

buffer: 缓存变量/数组。用于存储读取到的信息。c允许使用char或者byte类型的变量或数组。
length: 读取字节数量。readBytes函数在读取到length所指定的字节数量后就会停止运行。允许使用int类型。

### 返回值

buffer(缓存变量)中存储的字节数。数据类型：size_t

## readBytesUntil

### 说明

readBytesUntil() 函数可用于从设备接收到数据中读取信息。读取到的数据信息将存放在缓存变量中。该函数在满足以下任一条件后都会停止函数执行并且返回。

– 读取到指定终止字符
– 读取到指定字节数的信息
– 达到设定时间（可使用setTimeout来设置）

当函数读取到终止字符后，会立即停止函数执行。此时buffer（缓存变量/数组）中所存储的信息为设备读取到终止字符前的字符内容。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
stream.readBytesUntil(character, buffer, length)`
/*注：此处stream为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.readBytesUntil(character, buffer, length)`
`wifiClient.readBytesUntil(character, buffer, length)
```

### 参数

character: 终止字符。用于设置终止函数执行的字符信息。设备在读取数据时一旦读取到此终止字符，将会结束函数执行。允许使用char类型。
buffer: 缓存变量/数组。用于存储读取到的信息。允许使用char或者byte类型的变量或数组。
length: 读取字节数量。readBytes函数在读取到length所指定的字节数量后就会停止运行。允许使用int类型。

### 返回值

buffer(缓存变量)中存储的字节数。数据类型：size_t

### 例程

```
char terminateChar =  'T';      // 建立终止字符
const int bufferLength = 10;    // 定义缓存大小为10个字节
char serialBuffer[bufferLength];// 建立字符数组用于缓存

void setup() {
  // 启动串口通讯
  Serial.begin(9600);
  Serial.println();
}

void loop() {
  if (Serial.available()){             // 当串口接收到信息后
```

```
    Serial.readBytesUntil(terminateChar, serialBuffer, bufferLength);// 将接收到的信息使用read读取

    for(int i=0; i<bufferLength; i++){  // 然后通过串口监视器输出readBytesUntil
      Serial.print(serialBuffer[i]);    // 函数所读取的信息
    }

    Serial.println("");
  }
}
```

# write()

## 说明

写二进制数据到串口，数据是一个字节一个字节地发送的，若以字符形式发送数字请使用print()代替。

## 语法

```
Serial.write(val);
Serial.write(str)Serial.write(buf, len);
```

## 参数

val: 作为单个字节发送的数据
str: 由一系列字节组成的字符串
buf: 同一系列字节组成的数组
len: 要发送的数组的长度

## 返回值

write()会返回发送的字节数。

## 例程

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

void loop(){
  Serial.write(45);
}
```

# readString

## 说明

readString() 函数可用于从设备接收到数据中读取数据信息。读取到的信息将以字符串格式返回。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
stream.readString();
/*注：此处stream为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.readString()`
`wifiClient.readString()
```

### 参数

无

### 返回值

接收到的数据，类型为字符串。

## readStringUntil

### 说明

readStringUntil函数可用于从设备接收到的数据中读取信息。读取到的数据信息将以字符串形式返回。该函数在满足以下任一条件后都会停止函数执行并返回。

– 读取到指定终止字符
– 达到设定时间（可使用setTimeout来设置）

当函数读取到终止字符后，会立即停止函数执行。此时函数所返回的字符串为"终止字符"前的所有字符信息。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

### 语法

```
Stream.readStringUntil(terminator)`
/*注：此处stream为概念对象名称。在实际使用过程中，需要根据实际使用的stream子类对象名称进行替换。如：*/
`Serial.readStringUntil(terminator)`
`wifiClient.readStringUntil(terminator)
```

### 参数

terminator: 终止字符。用于设置终止函数执行的字符信息。设备在读取数据时一旦读取到此终止字符，将会结束函数执行。允许使用char类型。

### 返回值

接收到的数据，类型为字符串

# setTimeout()

## 说明

serial . settimeout()设置等待串行数据的最大毫秒数。默认值是1000毫秒。

本函数属于Stream类。该函数可被Stream类的子类所使用，如（Serial, WiFiClient, File 等）。

## 语法

```
Serial.setTimeout(time);
```

## 参数

Serial:串口对象

time:超时时间，单位为毫秒。允许的数据类型: long

## 返回值

无

## 注意

使用' *Serial*.setTimeout() '设置的超时值的串行函数:

# serialEvent()

## 说明

当数据可用时调用。使用Serial.read()来捕获该数据。

## 语法

```
void serialEvent() {
  //statements
}
```

## 参数

statement:任何有效的语句

## 返回值

无

## 注意

`serialEvent()` doesn't work on the Leonardo, Micro, or Yún.

`serialEvent()` and `serialEvent1()` don't work on the Arduino SAMD Boards

`serialEvent()`, `serialEvent1()`, `serialEvent2()`, and `serialEvent3()` don't work on the Arduino Due.