GENUINO 101
使用教程

DFROBOT
DRIVE THE FUTURE

目录

第一话：Arduino 与 Genuino 101

Arduino 平台介绍

Arduino 是一款简单易用的，集硬件，软件功能于一身的开发平台，旨在为智能硬件爱好者，交互艺术设计师以及电子软件工程师，提供简易的开发体验。

Arduino 可以通过各种各样传感器来检测周围环境，并通过电机，LED 灯以及其他发生器来影响周围环境。板上微控制器编程使用 Arduino 编程语言（基于 Wiring）和 Arduino 开发环境（以 Processing 为基础）。Arduino 可以独立运行，也可以与计算机上运行的软件（例如，Flash，Processing，MaxMSP）进行通信。Arduino 开发 IDE 接口基于开放源代码，可以让您免费下载使用，开发出更多令人惊艳的互动作品。

Arduino 是人们连接各种任务的粘合剂。我们可以通过以下的实际案例来了解 Arduino 平台能做什么。

- 您想当咖啡煮好时，咖啡壶就发出“吱吱”声提醒您吗？
- 您想当邮箱有新邮件时，电话会发出警报通知您吗？
- 想要一件闪闪发光的绒毛玩具吗？
- 想要一款具备语音和酒水配送功能的 X 教授蒸汽朋克风格轮椅吗？
- 想要一套按下快捷键就可以进行实验测试蜂音器吗？
- 想为您的儿子自制一个《银河战士》手臂炮吗？
- 想自制一个心率监测器，将每次骑脚踏车的记录存进存储卡吗？
- 想过自制一个能在地面上绘图，能在雪中驰骋的机器人吗？

Arduino 可以让所有这些美妙的项目都触手可及！

Arduino 的诞生

这个最经典的开源硬件项目，诞生于意大利的一间设计学校。Arduino 的核心开发团队成员包括：Massimo Banzi，David Cuartielles，Tom Igoe，Gianluca Martino，David Mellis 和 Nicholas Zambetti。

据说 Massimo Banzi 的学生们经常抱怨找不到便宜好用的微控制器，2005 年冬天，Massimo Banzi 跟朋友 David Cuartielles 讨论了这个问题，David Cuartielles 是一个西班牙籍晶片工程师，当时在这所学校做访问学者。两人决定设计自己的电路板，并引入了 Banzi 的学生 David Mellis 为电路板设计编程语言。两天以后，David Mellis 写出了代码。又过了三天，电路板就完工了。这块电路板被命名为 Arduino。几乎任何人，即使不懂电脑编程，也能用 Arduino 做出很酷的东西，比如对感测器做出回应，闪烁灯光，控制马达。

Arduino 名称的由来

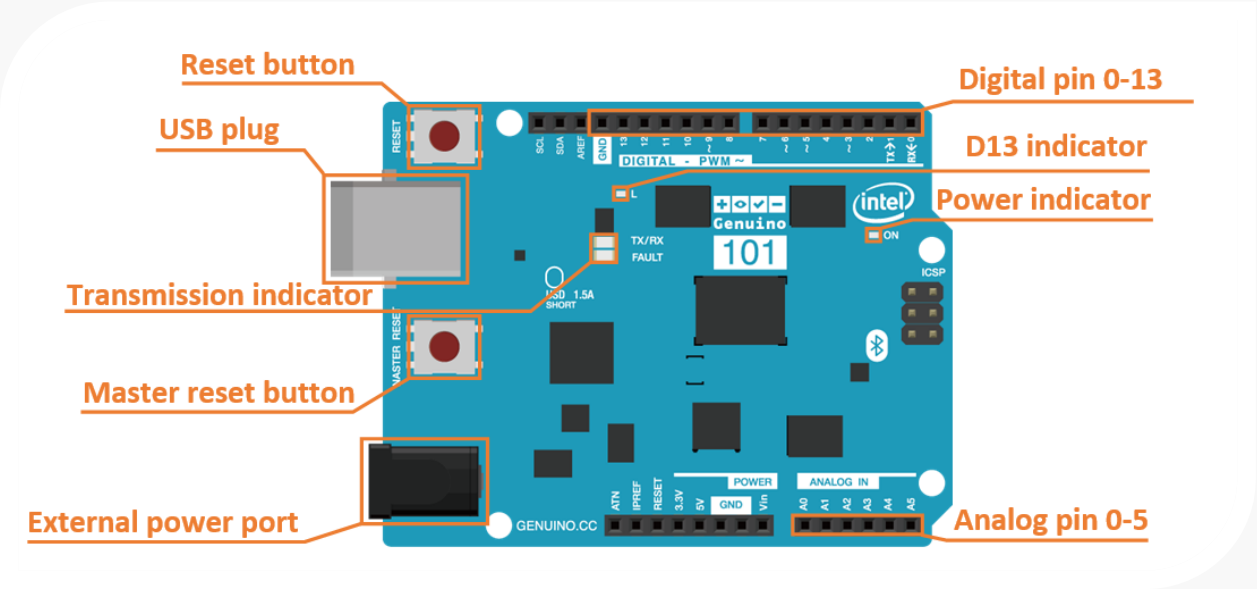
意大利北部一个如诗如画的小镇「Ivrea」，横跨过蓝绿色 Dora Baltea 河，它最著名的事迹是关于一位受压迫的国王。公元 1002 年，国王 Arduino 成为国家的统治者，不幸的是两年后即被德国亨利二世国王给废掉了。今日，在这位无法成为新国王的出生地，cobblestone 街上有家叫「di Re Arduino」的酒吧纪念了这位国王。Massimo Banzi 经常光临这家酒吧，而他将这个电子产品计划命名为 Arduino 以纪念这个地方。

Genuino 101 简介

Genuino101（美国以外地区）和 Arduino101（美国地区）是一款基于英特尔最新发布的 Curie 模组的开发平台。Curie 芯片拥有的两核处理器分别支持 x86 和 ARC 架构。相比经典的 Arduino UNO 主控器，101 的运算速度与功耗都有显著的提升。

除此以外，101 板载了有蓝牙 4.1 模块、六轴加速度计和陀螺仪，能轻松实现蓝牙通信和姿态识别等进阶功能。在设计上，101 保持了 UNO 的简洁性。它与 Arduino UNO 拥有相同的 I/O 口排布，兼容大多数可用于 UNO 的元件和扩展板。与 Arduino 系列其他扩展板相同，Genuino 101 可以使用 Arduino IDE 编程。

101 拥有 14 个数字 I/O 口（包含 4 个 PWM 输出口）和 6 个模拟输入口、USB 接口（可用于程序烧写或串口通信）、一个电源插头、一个 ICSP 接头和 I2C 引脚。



需要注意的是， Genuino 101 的工作电压为 3.3V（但仍然支持 5V 信号输入），这意味着它无法兼容一部分驱动电压为 5V 电子器件。但使用这些器件并不会造成安全问题。

另外，你会注意到 101 同时拥有一个“**RESET**”按钮和一个“**MASTER RESET**”按钮。这是由于 101 在执行程序时，Curie 芯片中其 x86 架构的内核会运行的一个独立的操作系统（ViperOS RTOS），而它的 ARM 核心则用于 I/O 口引脚的信号处理。因此，“**MASTER RESET**”按钮可以用于重启整个操作系统，重置 Arduino 程序，以及蓝牙连接。而“**RESET**”按钮则只会重启 Arduino 程序。

初次使用 Genuino101

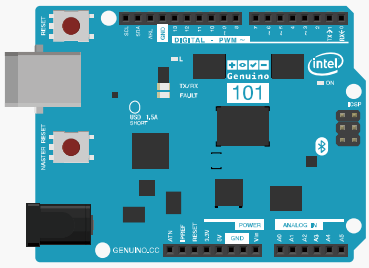
熟悉 Arduino 平台的用户可以直接按照 Genuino101 主页上提供的相关教程学习使用 Genuino101。

Genuino101 主页：https://www.arduino.cc/en/Guide/Arduino101

如果你是第一次接触 Arduino 平台，同样可以通过以下零基础教程学习如何下载安装 Arduino IDE 软件，以及如何为 101 编写程序。

开始之前，请确认你手边有如下物品，除此之外，你还需要一台运行 Windows/Mac OS/Linux 操作系统并且有网络连接的电脑。

所需元件：



Genuino 101 开发板

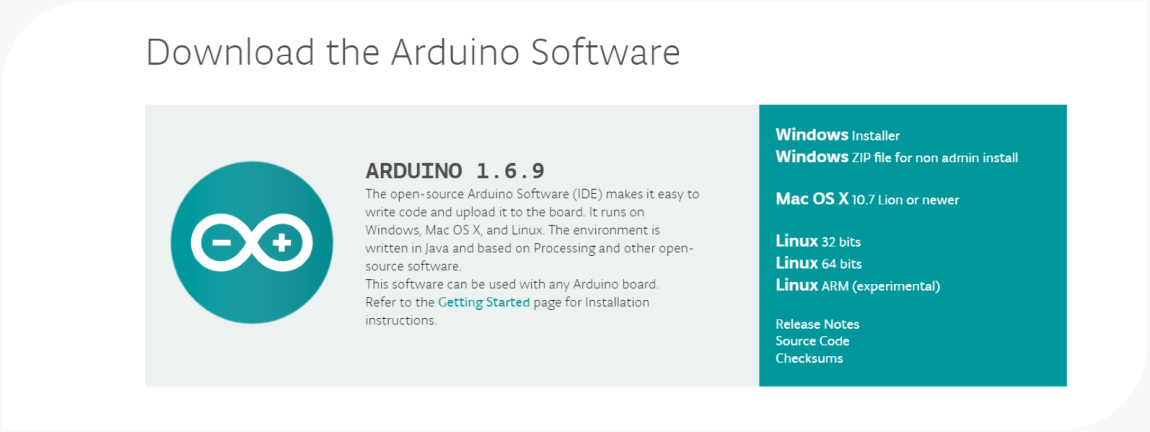
USB A to B 连接线

STEP1：下载 Arduino IDE 软件

以下的步骤说明基于 Windows 操作系统，如果你使用的是其他操作系统，可以将其作为参考。

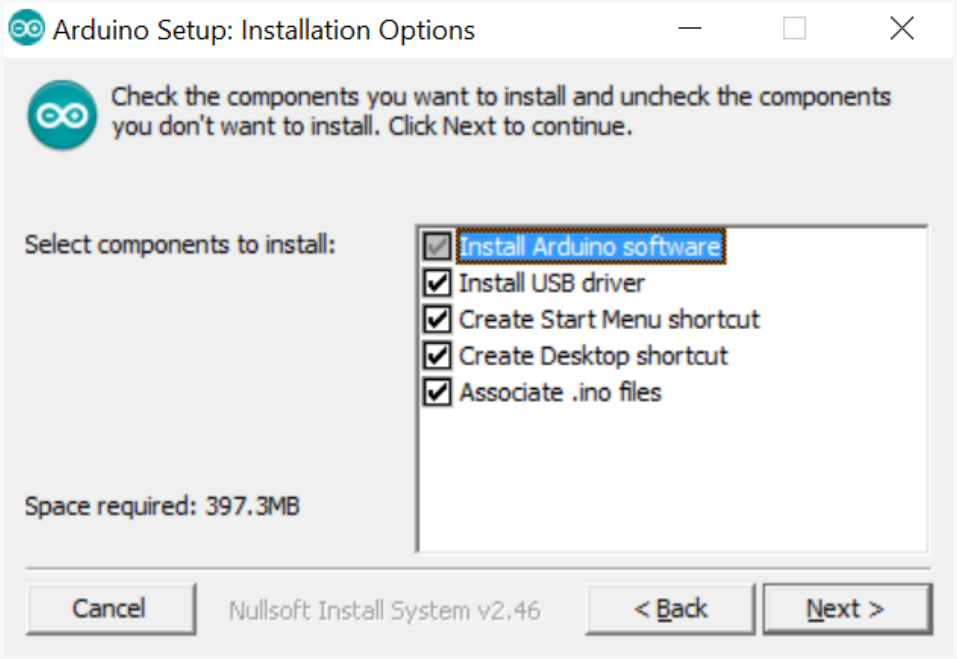
首先，你需要从官网下载最新版本的 Arduino IDE 软件。下载链接：http://arduino.cc/en/Main/Software

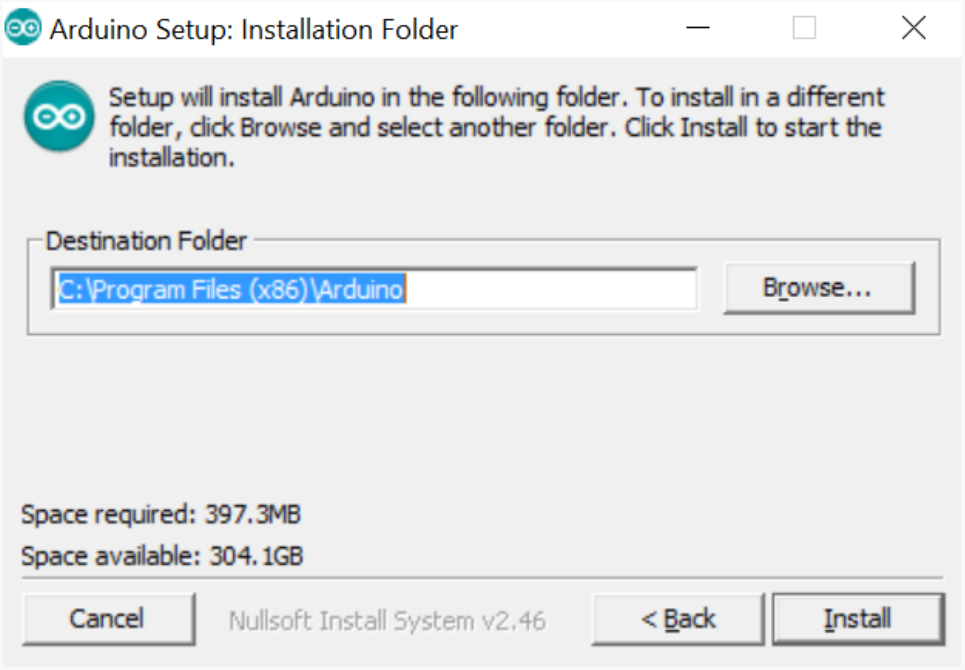
注意: Genuino101 不支持早于 1.6.7 的版本。

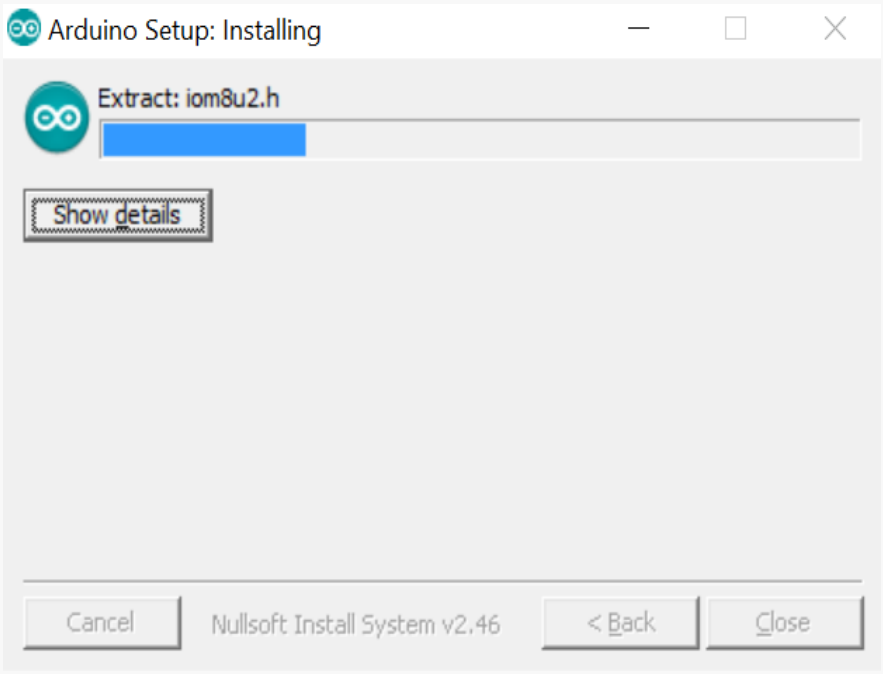


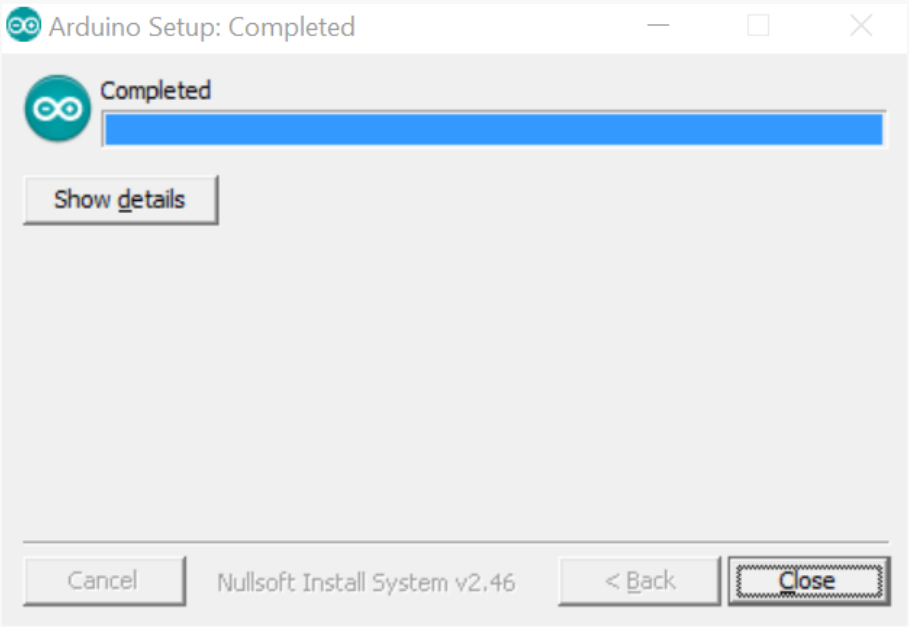
在下载页右侧的列表中选择下载对应的安装包。对于 Windows 系统用户既可以选择下载 **Windows installer**（推荐初次使用者下载），也可以下载 Windows ZIP 安装包（需要手动安装驱动）。

若选择的是 Windows installer，你可以直接执行安装程序，并跟随安装向导完成配置，驱动会在程序安装完成后自动安装。


Arduino Setup: Installation Options
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Install Arduino software
Install USB driver
Create Start Menu shortcut
Create Desktop shortcut
Associate .ino files
Space required: 397.3MB
Cancel
Nullsoft Install System v2.46
< Back
Next >


Arduino Setup: Installation Folder
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Program Files (x86)\Arduino
Browse...
Space required: 397.3MB
Space available: 304.1GB
Cancel
Nullsoft Install System v2.46
< Back
Install

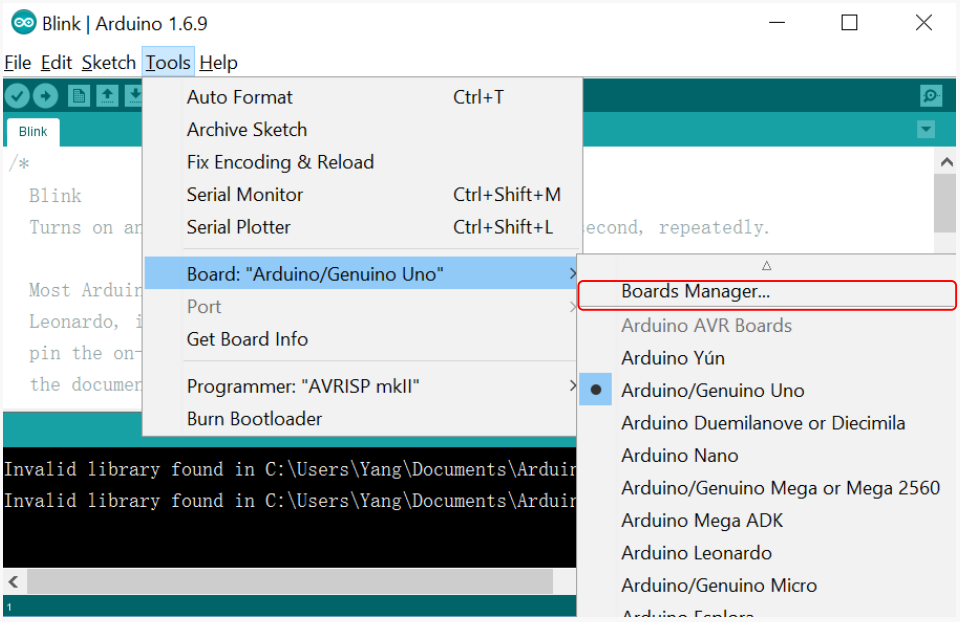

Arduino Setup: Installing
Extract: iom8u2.h
Show details
Cancel
Nullsoft Install System v2.46
< Back
Close


Arduino Setup: Completed
Completed
Show details
Cancel
Nullsoft Install System v2.46
< Back
Close

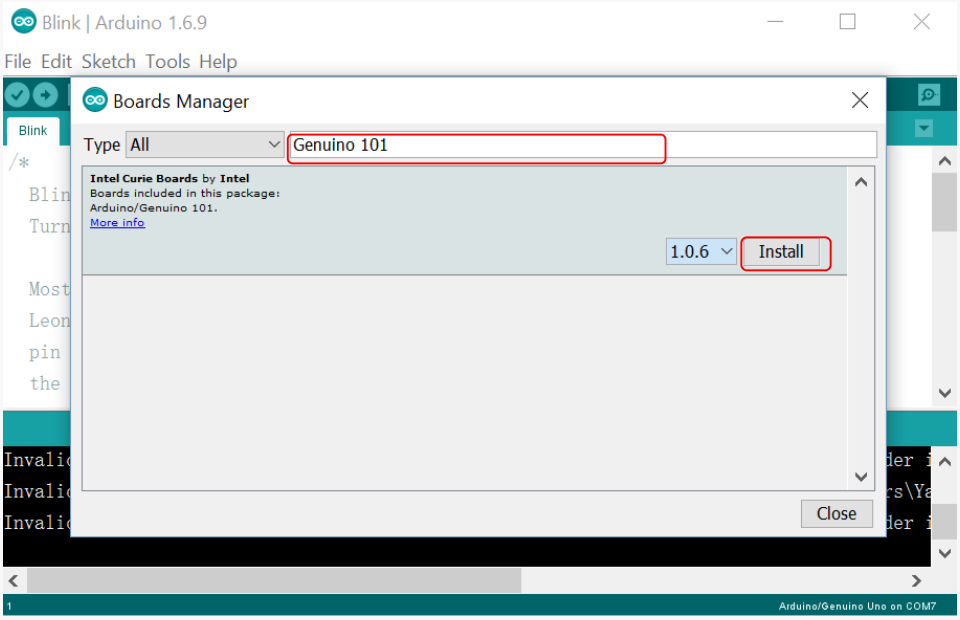
STEP2: 安装 Intel Curie 开发板核心

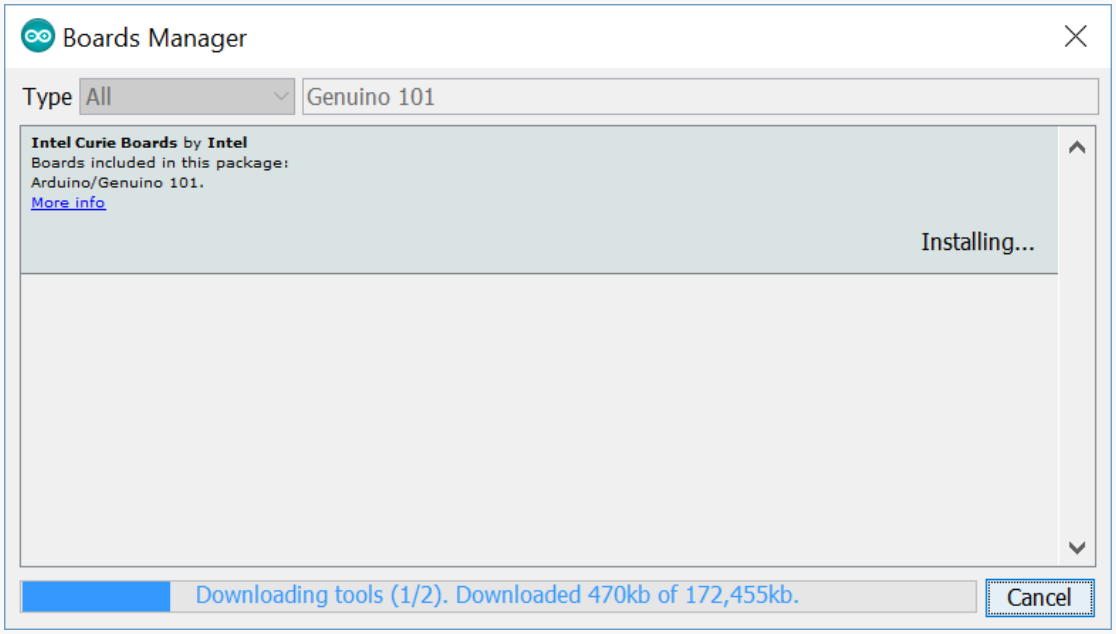
Arduino IDE 安装包中不包含 Intel Curie 开发板核心。首先，要添加 Genuino101 支持，需要在 Arduino 开发板管理器里手动安装 Intel Curie 开发板核心。

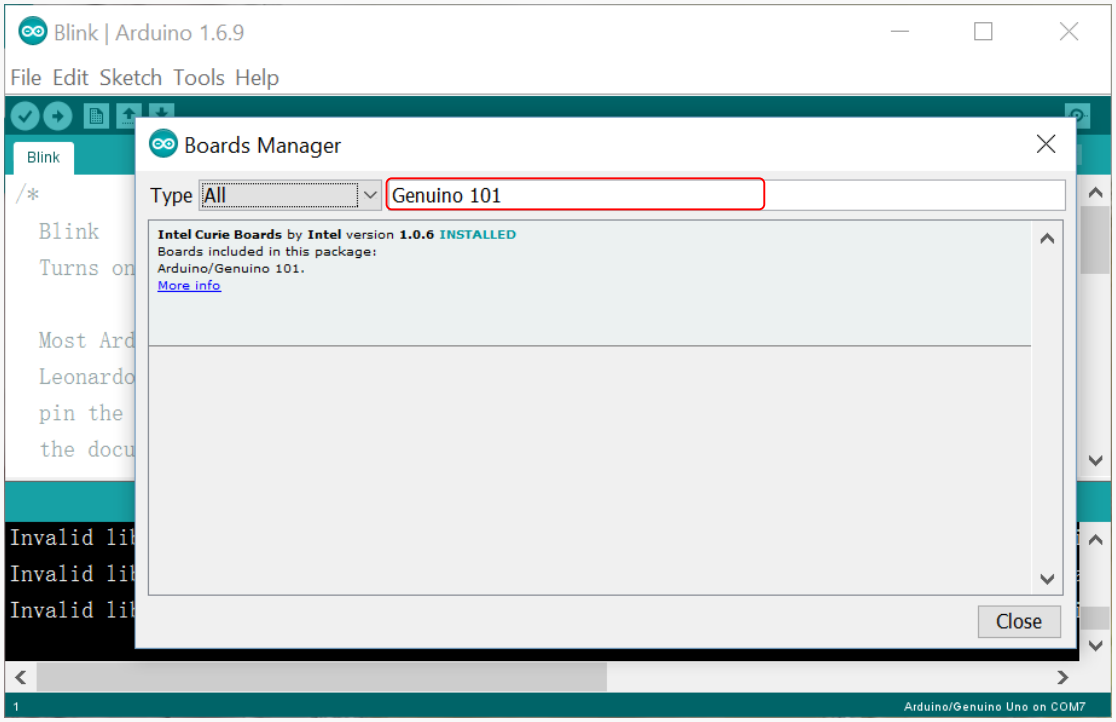
打开**工具栏 -> 工具 -> 板子 -> 开发板管理器**



在新开窗口中，输入"Genuino 101"并等待信息加载完毕。选择版本号 1.0.6（或者最新版本并点击"install"后耐心等待安装完成。整个过程会因网络状况持续 5-10 分钟。安装完成后，开发板信息会被标注为"INSTALLED"。



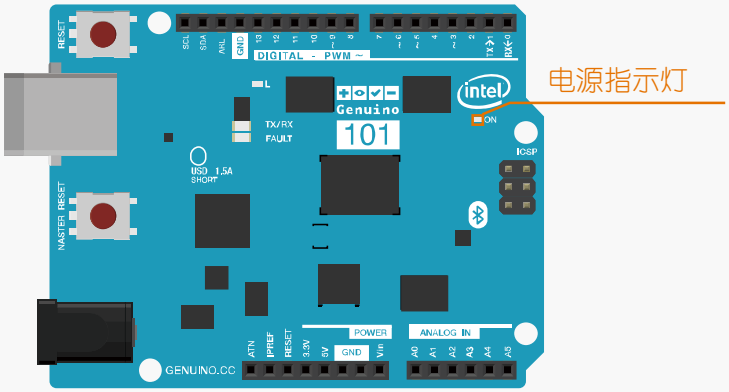




注意：在安装过程中，有些关键进程可能会被防火墙或者杀毒软件拦截，请选择允许更改并添加至白名单。如果 101 在接下来的步骤中运行不正常，你可以尝试卸载并重新安装 Intel Curie 开发板核心。

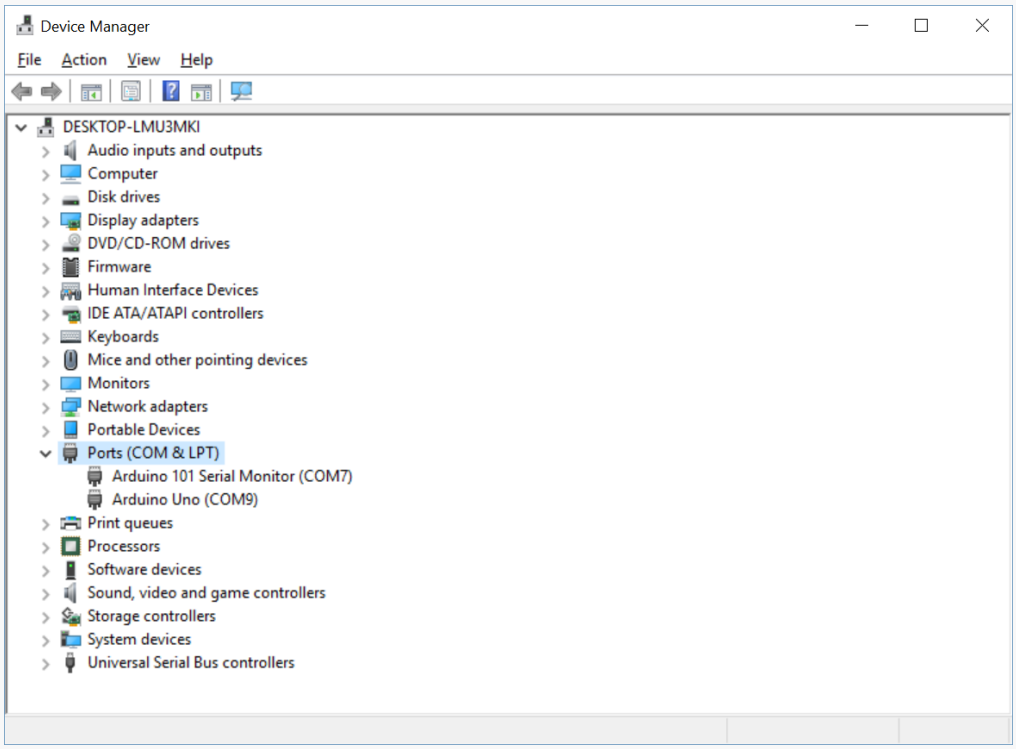
STEP3: 连接 Genuino 101 至电脑

正确安装完成 Arduino IDE 和 Intel Curie 开发板核心后，即可将 Genuino101 通过 USB A to B 数据线连接至电脑。正确连接时 Genuino101 右上角的电源指示灯灯会亮起。



在编程之前，我们需要确保开发板被电脑识别并且找出连接了哪一个 COM 口（用于提供串口通信交互）。可以在接下来的步骤中确认。

首先打开“控制面板”，打开 “设备管理器”，点开“端口（COM&LPT）”。接上 Genuino(Arduino) 101 的端口就会在列表中显示（这里是 COM7）。



STEP4: 在 Arduino IDE 中进行编程

Arduino IDE 软件安装完成后，运行软件打开编程窗口。你可以在这个窗口里编辑并上传代码到 Arduino 开发板上，或是使用内置的串口监视器通过串口与开发板通信。

现在让我们仔细看下 Arduino IDE 界面。

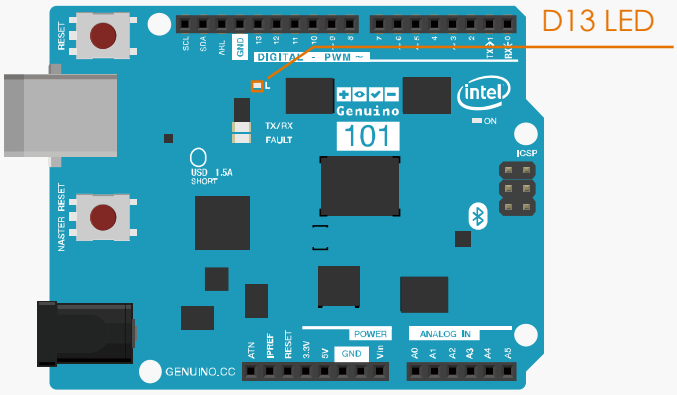
与常规 C 语言程序不同的是，一段用于 Arduino 的程序通常由 void setup() 部分与 void loop() 两部分构成。

"void setup()"用于放置初始化程序的代码，这部分代码在开发板上电后仅运行一次。需要重复运行的代码需要放置在"void loop()"中，的这些代码会一直重复运行，使得开发板时时与外部进行交互。

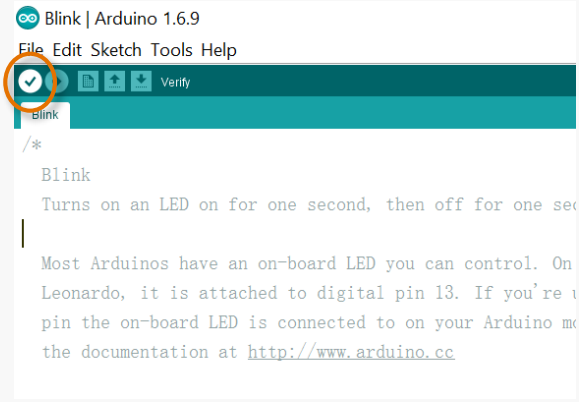
白色区域是编程区域。编程区域以下的黑色部分是信息窗口，用于显示代码上传以及编译的信息。

STEP5: 上传代码至 Genuino 101

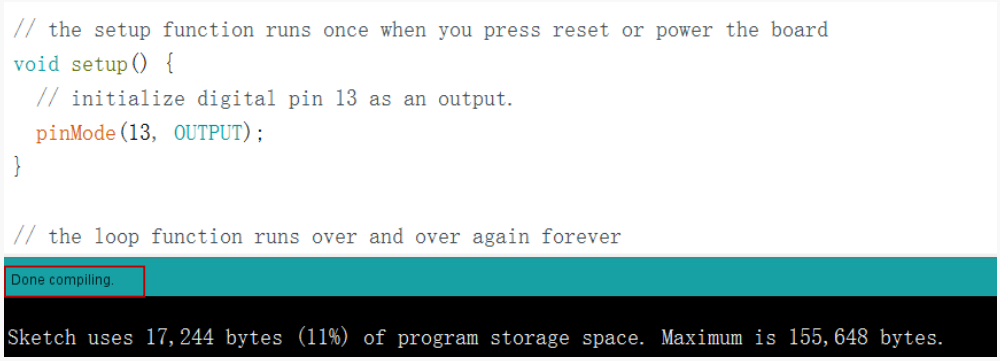
在这一步，我们将演示如何上传示例程序"Blink"到 Genuino101。"Blink"程序的功能是控制 D13 引脚上的 LED 灯间隔一秒闪烁。101 与大部分 Arduino 相同，有一个板载的 D13 LED 信号灯，这意味着在本例中我们不需要其他的元件。LED 状态指示灯可以在数字引脚排针座旁边找到。



在上传之前，你应该首先确认代码中没有错误。点击"编译"确认。



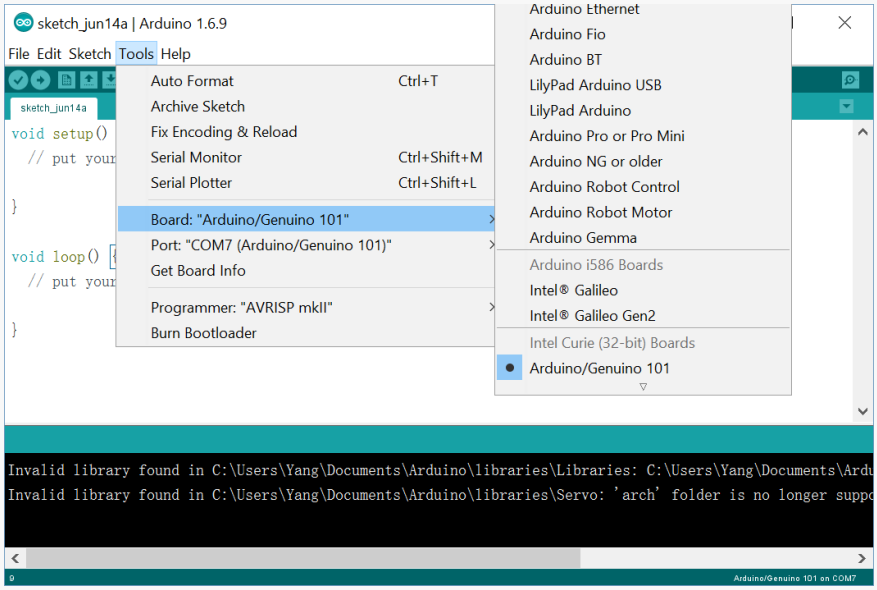
等待几秒钟，如果没有错误的话，一条"Done compiling "信息会在信息窗口显示,表示 "编译成功"。如果出现错误可以返回检查程序是否完整。



编程成功后，我们需要为 101 选择开发板类型和端口。

工具栏>工具>开发板里选择"Arduino/Genuino 101"

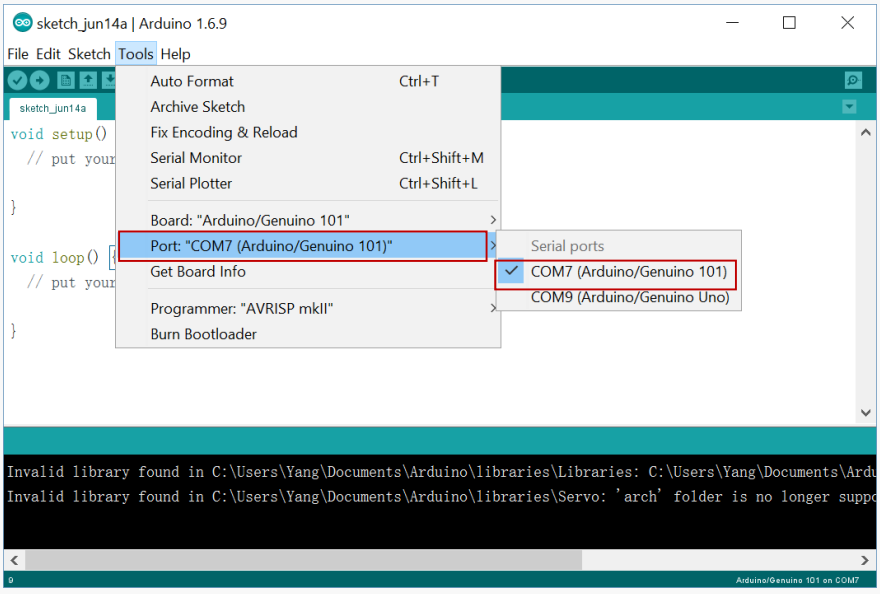
通常在不切换到使用其他型号的开发板时，这个步骤仅需在第一次使用时执行一次即可。



切换到工具栏>工具>端口

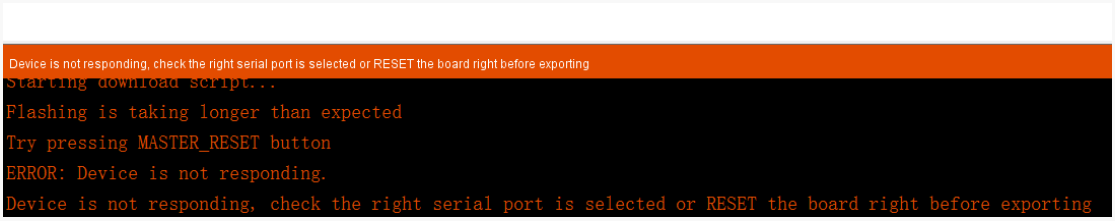
根据 STEP3 中显示的 101 所占用串口序号，我们应该选择"COM 7"作为通信端口。通信端口只有当开发板连接至电脑并被成功识别时才会出现。同一块开发板在插拔后可能会占用不同的端口上，所以我们每次上传前都要重复确认。

COM 选择完成后，开发板的信息和端口就会在窗口右下角显示。

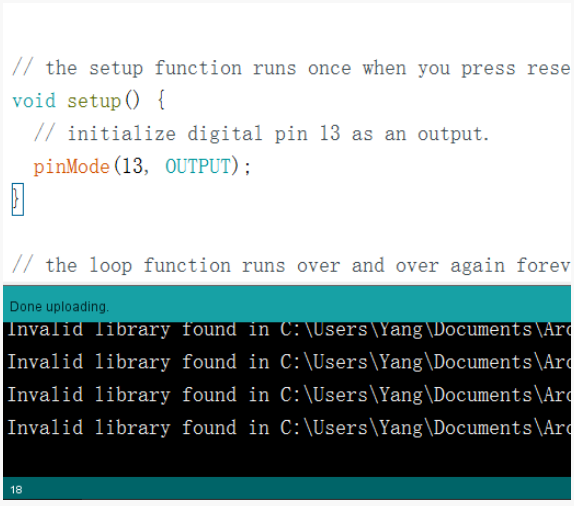


最后，点击“上传”烧写代码到 101 中。

由于 101 特殊的运行机制，有时会出现代码无法成功上传的情况。此时以下信息会在消息窗口显示。要解决这个问题，可以尝试按下板子上的"Master Reset"重启 101。



成功上传后，“Done uploading”消息会出现在信息窗口。此时 101 板载的 D13 的 LED 灯会开始闪烁。



简而言之，为 101 上传代码可以分为以下三个步骤：

- 编译代码；
- 选择开发板型号和端口；
- 上传！

以上就是使用 101 的一些基本的方法。如果在使用中存在任何疑问或者建议，欢迎访问我们的论坛联系我们。

论坛链接: http://www.dfrobot.com.cn/community/forum.php

记得来逛 DFRobot 的社区看看更多的教程和精彩的项目哦。我们同样希望你能够把你自己的项目或者想法发在论坛上分享。欢迎成为我们的一员！

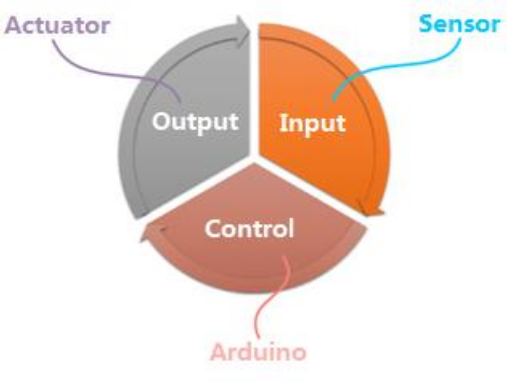
第二话：是什么让我们的机器“活”过来的？

交互设备

在接下来的几个章节中， 我们将建立一些交互套件项目，从按下按钮可以点亮 LED 灯开始，到显示温度传感器的测量示数。这些套件同样可以被分类为“交互式设备”，为了帮助你更好地理解他们的机械原理，我们需要首先弄明白它们的构造。

最简单的交互设备也要由 3 个部分组成：

- 用于命令接受或数据收集的输入单元
- 用于数据处理或信号处理的控制单元
- 用于发送数据或执行动作的输出单元



那么这些单元都意味着什么呢？举我们自身为例，我们通过眼睛、耳朵、鼻子和皮肤收集光线、声音、味道和力量形式的信息，这些信息会进入我们的大脑来决定需要采取什么反应。最终我们根据这些信息采取实际行动来改变周身物理环境。具体来说，例如你的朋友对你说“你好”，你回应说“你好”，你的耳朵扮演着输入元件、大脑扮演控制模块，嘴巴扮演输出单元这样的角色。

同样，对 Geniuno101 项目来说，我们使用多种传感器作为输入单元，101 作为控制单元，执行机构作为输出单元。

输入单元——传感器

传感器（又被称为变换器）是一种可以察觉或探测到环境特性（比如光、温度、湿度等）并且将它们转化为信号数据的物理元件。在本教程中，我们将使用按钮、声音传感器和温度传感器等传感器。

控制单元——Genuino 101

101 将在这扮演控制单元，使用信号引脚建立输入单元和输出单元的联系，并且在计算处理模块中处理数据。

输出单元——执行机构

执行机构是负责移动或者控制一个系统或者机械结构的设备。它把电能转化为运动，声音和光。在本教程中，我们使用类似 LED 灯、蜂鸣器和舵机等执行机构。

IO 引脚

IO 引脚是可以提供输入输出等信号传输的信号引脚，其他元件通过它们连接在开发板上。Genuino 101 有三种 IO 引脚。

- 数字量引脚
- 模拟量引脚
- 协议引脚（数字）

数字引脚既可以发送数字信号（其中一些支持脉冲调制信号），也可以发送接收数字信号，这意味着它同时支持传感器和执行机构。然而，模拟量引脚只能接受模拟量信号，所以只能支持模拟量传感器。

协议引脚也是一种数字引脚，I2C，串口和 SPI 都是经常用到的数字引脚。

代码和硬件之间的关系

上面提到的输入单元，控制单元和输出单元都是硬件，类比我们人类，硬件是我们的身体。然而，大脑更加重要因为它产生想法并控制我们的每一个行为。在这里代码就像我们的大脑一样。对我们人类来说，身体和大脑都是不可或缺的。

数字信号和模拟信号

输入单元、Arduino 控制器和输出单元使用信号通信，而信号又被代码所处理。输入单元和控制器是如何互相通信的呢？要回答这些问题，我们首先需要理解两个概念：数字信号和模拟信号。

数字信号和模拟信号的不同点：

数字信号：

数字信号有两种状态："高电平"和"低电平"：

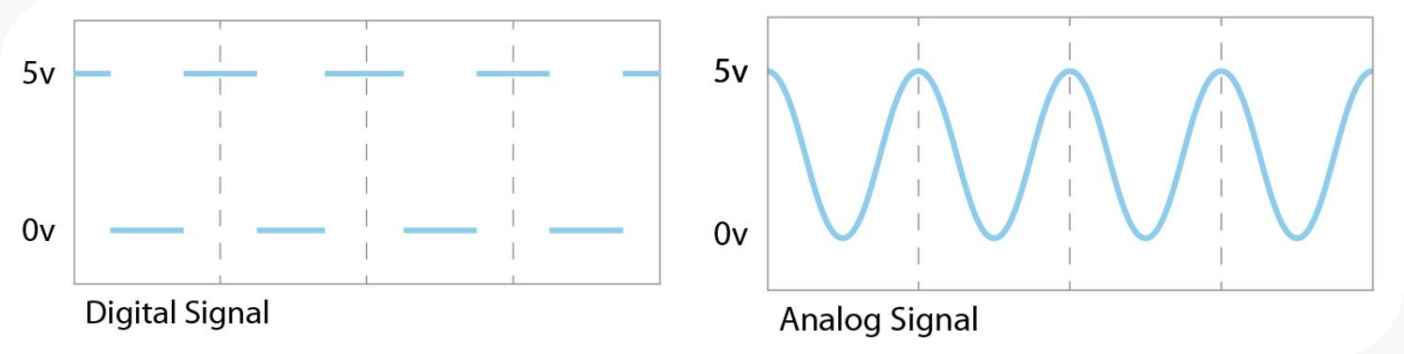
"高电平"是 5V 的电信号，代表"1"（或者开）。

"低电平"是 0V 的电信号，代表"0"（或者关）。

模拟信号：

模拟信号可以在一定范围内有不同的量值。

在 Arduino 的模拟信号引脚上，0V 到 5V 之间的所有值都可以对应到 0 到 1023 中的某个数。比如，0 对应 0V；1023 对应 5V 而 512 则对应 2.5V。



DFRobot 套件中的“数字”和“模拟”

要使用一个元件，我们应当始终清楚它支持的是何种信号类型。在我们的套件中，你可以按如下方法区分数字量元件和模拟量元件。

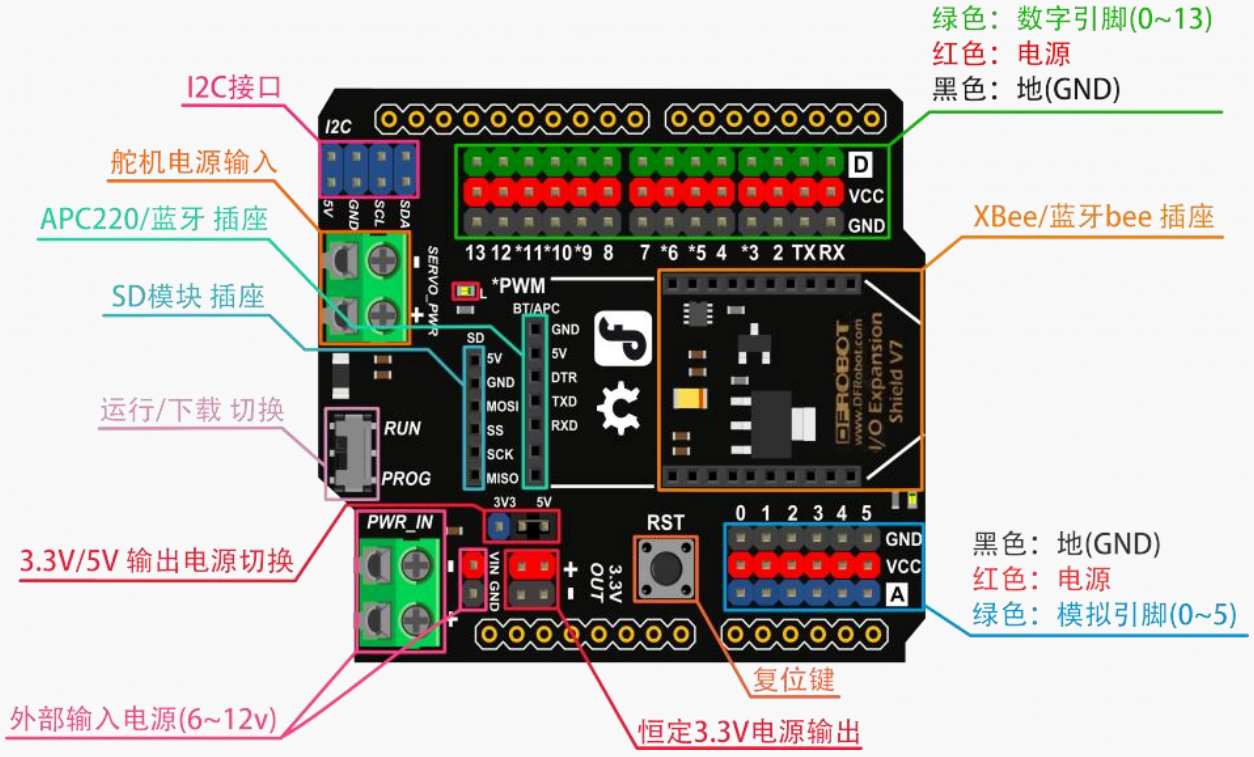
数字量模块在角落里会标着字母“D”；
而模拟量模块会在角落里标上字母“A”

数字量模块的数据线被标为绿色；
而模拟量模块的数据线则被标为蓝色。

IO 扩展板

IO（输入和输出）扩展板是 Arduino 项目最受欢迎的外接硬件之一。一个标准的 IO 扩展板装有信号和电源扩展排针，扩展按钮和开关，扩展电源接口和其他模块接口。与面包板不同的是，模块可以直接连接在 IO 扩展板上，这使得回路连接更为快速而高效。

DFRobot 开发的 IO 扩展板兼容大部分的 UNO 开发板，即意味着它同样适用于 101。让我们先来看看扩展板长什么样吧。



在使用扩展板之前，我们首先需要弄明白板上的这些标志是什么意思。

板子右边的字母“D”和“A”分别代表“数字”和“模拟”。

同样，不同种类的引脚也被标为不同的颜色：

绿色 = 数字信号引脚

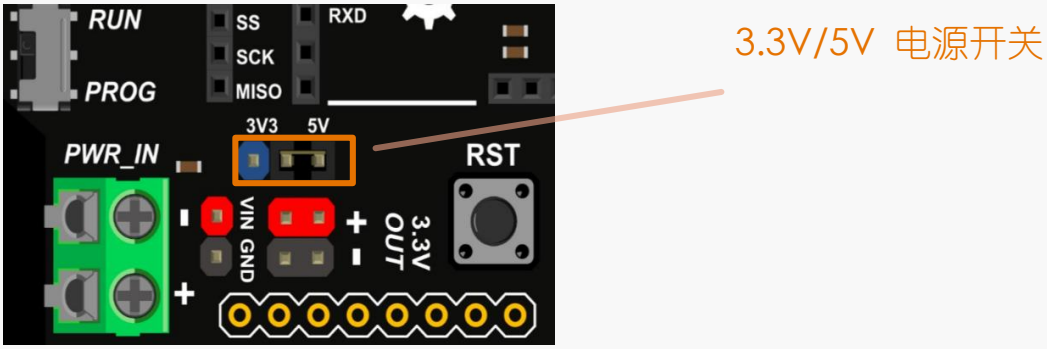
蓝色 = 模拟信号引脚

红色 = VCC (电源)

黑色 = GND (接地)

板子上的 VCC 引脚同时支持 5V 和 3V 的电源输入。可以使用跳线帽切换连接 5V 还是 3V 的引脚来切换电压。

由于教程中的元件都是 3.3V/5V 兼容的，我们就将跳线帽连接在 5V 引脚这边。需要注意，可能您采购的元件在 5V 电压下会烧坏，所以要谨慎选择！教程样例全部选用的是 5V 供电，请放心使用。



同样地，你可以在板子上找到一个扩展的 D13 LED 灯和一个 Reset 按钮。

你可以在我们的 wiki 上找到有关扩展板的更多信息。

第三话：数字信号和模拟信号

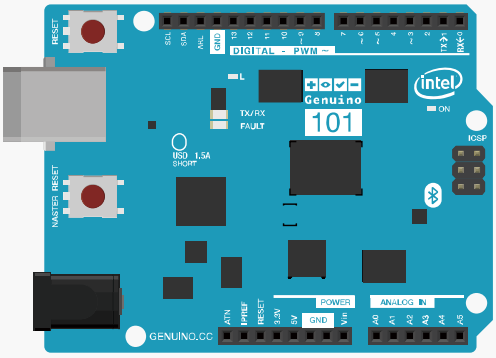
在我们了解不同种类信号的机理之后，本章节中我们将使用 Arduino IDE 中的串口监视器来探究在真实的电路中它们是怎么回事。

数字信号

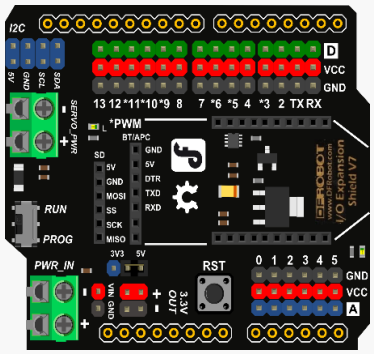
我们将使用一个数字信号按钮模块来演示什么是数字信号。你的套件中已包含这个模块。

所需元件:

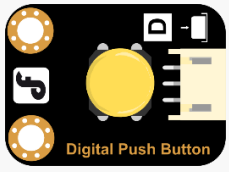
- 1 × Genuino 101 开发板 (包括 A to B USB 连线)



- 1 × I/O 传感器扩展板 V7.1



- 1 × 数字信号按钮模块



硬件连接

取出传感器扩展板并把它对齐到 101 上面一一确保引脚都是正确排布之后压紧。

取出数字信号按钮模块把它连接在数字引脚 2 上。确保电源引脚、接地引脚和信号引脚都是正确连接的，否则你可能会烧坏元件。可以参考下图连线：

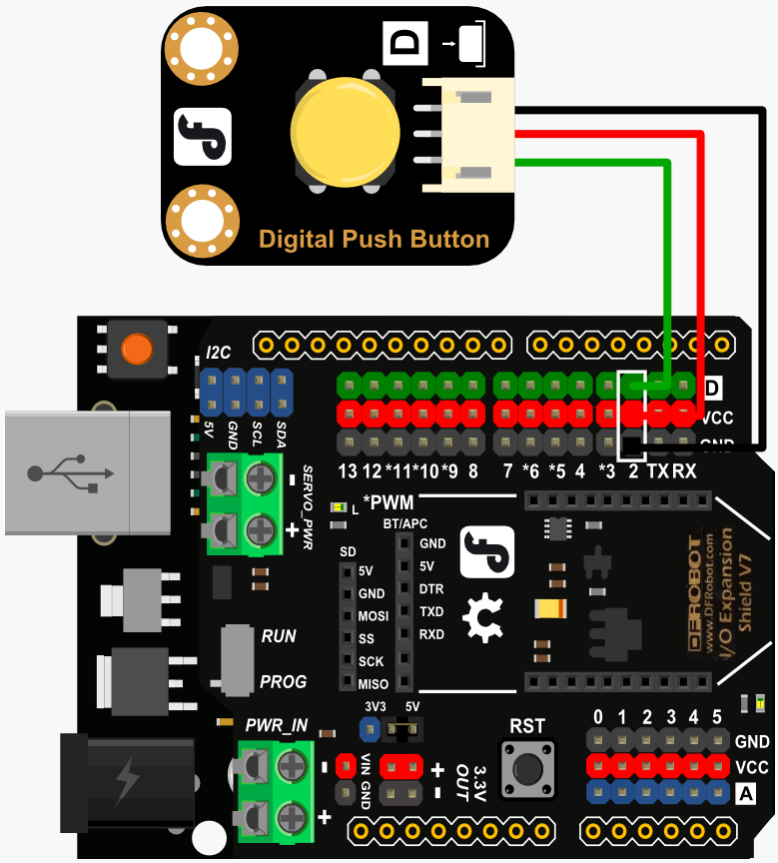


图 2-1 数字引脚连接

元件连好后接上 USB 线给 101 上电。我们现在可以准备上传程序了。

输入代码

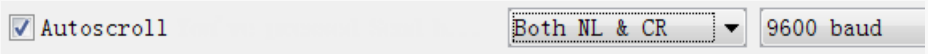
打开 Arduino IDE 并选择：“文件”>“示例”>“0.1.Basics”>“DigitalReadSerial”。以下代码会显示在 IDE 中：

```
int pushButton = 2;              //connect to digital pin 2

void setup() {                    // initial function
  Serial.begin(9600);             // set up baud rate of the serial port
  pinMode(pushButton, INPUT);     // set the button to be in the output mode
}
void loop() {                                 //main function
  int buttonState = digitalRead(pushButton);  // record statistics about the
status of digital pin 2

  Serial.println(buttonState);      // Serial port print statistics about the
status of digital pin 2
  delay(1);                              // delay 1ms
}
```

点击上传，Arduino IDE 将会编译代码并上传到 101 中。

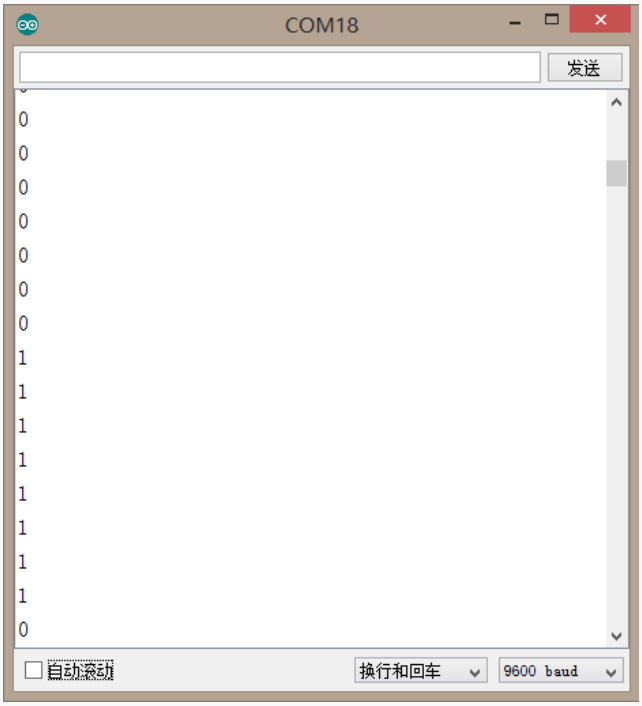
一旦上传完成后，打开工具栏>“工具”>“串口监视器”，打开串口监视器。

波特率是串口通信时数据传输的速度。然而 101 通过一个虚拟串口与 PC 端通信而不是使用硬件串口，所以你可以设置任意波特率都可以正常运行。



串口监视器显示 101 开发板发送的信息。我们将使用窗口监视器观察按钮的状态。你将会看到接连不断的“0”，这代表按钮没有被按下（off 状态）。

如果你按下了按钮，你将看到只要按着按钮，“1”就会出现在监视器中，这代表着按钮被按下（on 状态）。“0”和“1”或者说“off”和“on”是数字信号的两个值。在我们的 Arduino 开发中它有很多用途。

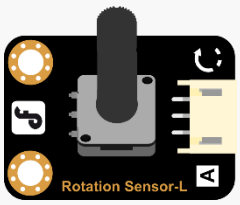


模拟信号

在这一章节中我们将继续探索模拟信号。我们将使用一款使用模拟信号的传感器：模拟角度传感器。由于 Genuino 101 工作电压是 3.3V，所以电位器和环境光传感器会有部分数据移除，使用时请注意！

所需元件:

1 × 模拟角度传感器-L



在我们大部分实验中我们都将使用 101 和 I/O 扩展，为了节省时间在每次的材料列表里就不再多提。

硬件连接

把模拟角度传感器连接在模拟信号引脚 0 上，像上一个实验一样确保信号引脚、接地引脚和电源引脚连接正确。连接 USB 线到 101 并上电，我们现在可以上传代码了。

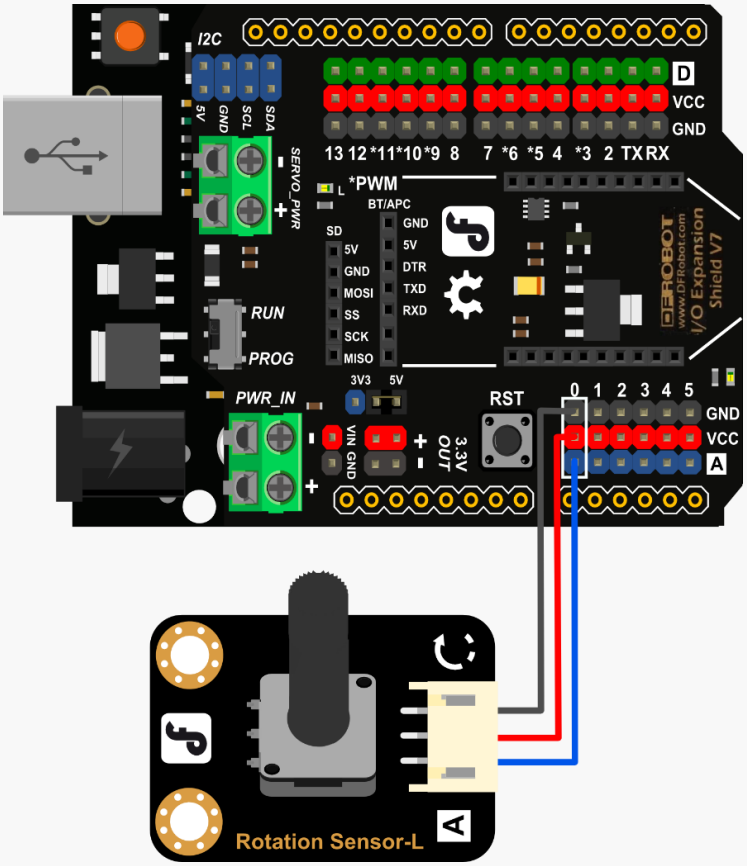


图 2-2 模拟引脚连接

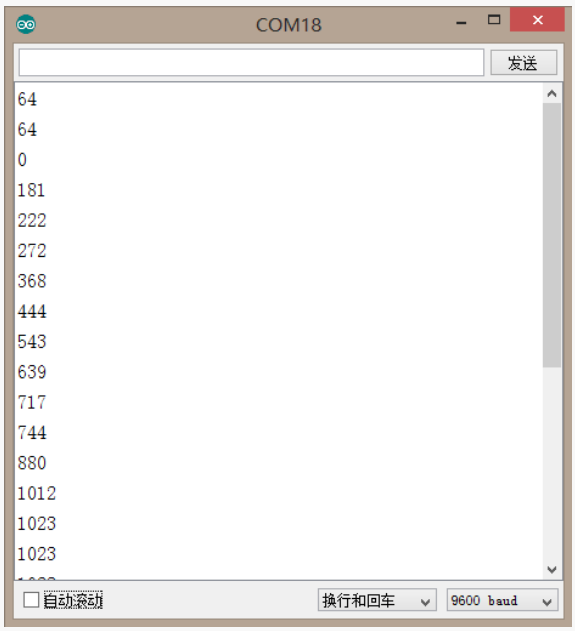
输入代码

在 Arduino IDE 中，打开工具栏> “文件” > “示例” > “01.Basics” > “AnalogReadSerial”，会打开新的程序窗口，显示如下：

```
void setup() {                        //Initial setup
  Serial.begin(9600);                 //Set port baud rate
}

void loop() {                         //Main function
  int sensorValue = analogRead(A0);    // Read status of analog pin 0.
  Serial.println(sensorValue);        // Print status of pin 0 in Serial
Monitor
  delay(1);                           //Delay 1ms
}
```

点击 “上传”，Arduino IDE 会编译程序并上传到 101 中。上传完成后，打开工具栏>"工具"> “串口监视器”。设置串口波特率为 9600。

如下，串口监视器中会显示一列数据。试着旋转模拟角度传感器，串口监视器上将会显示 0 到 1023 中的某个数，大小取决于模拟角度传感器的轴旋转了多少。0 意味着没有旋转而 1023 代表完全旋转。



比较数字信号和模拟信号

串口监视器

我们使用窗口监视器读取 101 返回的值，我们同样可以使用串口监视器发送数据给 101，并且在运行程序时灵活传输数据。

数字信号和模拟信号之间的差别很明显地在串口监视器中显示出来。

代码差异

如果仔细地检查代码，你可以看到我们使用了不同命令读取数字量引脚和模拟量引脚上的值：

在读取数字量引脚时，我们使用命令：“digitalRead()”

在读取模拟量引脚时，我们使用命令：“analogRead()”

数字量引脚：

```
int buttonState = digitalRead(pushButton);
// read values from digital pin2
```

模拟量引脚：

```
int sensorValue = analogRead(A0);
//read values from analog pin0
```

我们将在接下来的几章具体解释这些命令。

牛刀小试

DFRobot 套件中的传感器都跟上面例子中的连接方式相似，只要确保电源引脚、接地引脚和信号引脚连接都是正确的即可。连接完成后，你可以打开串口监视器观察它们输出的值，快来试试吧！

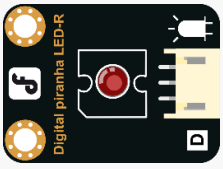
项目一 LED 灯闪烁

在对 Arduino 平台和 Genuino101 有了基础的了解，并弄明白数字信号和模拟信号之后，我们就可以来搭建第一个项目——使 LED 灯闪烁啦。

在最开始的章节中，我们上传了一个闪烁程序来测试板子上的 LED 状态灯。现在，我们将使用一个外接的 LED 模块代替板子上的 LED 状态灯，并让它根据我们的要求闪烁。

所需元件

1× 数字食人鱼红色 LED 发光模块

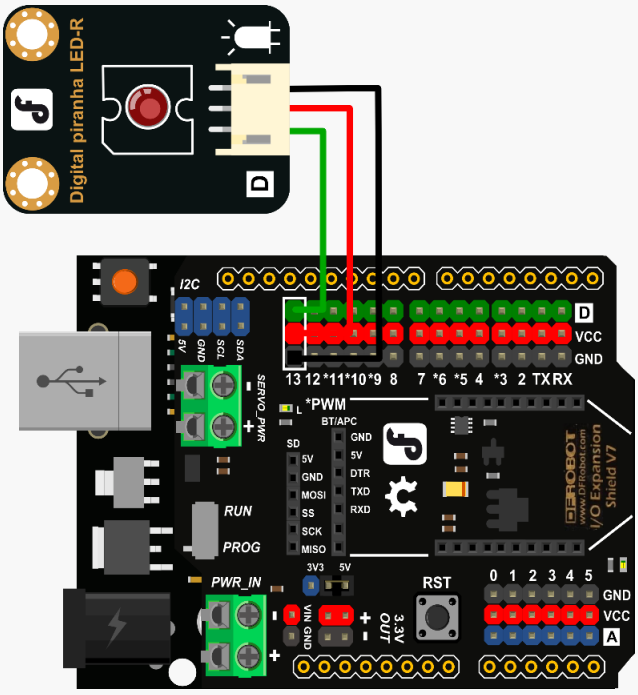


注意：由于接下来的都需要使用，I/O 扩展板不再列入所需物品列表。为了方便你可以把 I/O 扩展板留在 101 上。

硬件连接

把数字食人鱼红色 LED 发光模块接在 Arduino 板子的 13 号数字引脚上，确保电源引脚、接地引脚和信号引脚都连接正确，否则可能会烧坏元件。

元件连接好后，使用 USB 线连接 101 板子和电脑。



输入代码

打开 Arduino IDE。尽管你可以通过“文件”>"示例"> “01.Basic” > “Blink” 打开代码，我们还是建议你自己手动输入代码熟悉下。

代码如下 Sample Code 1-1:

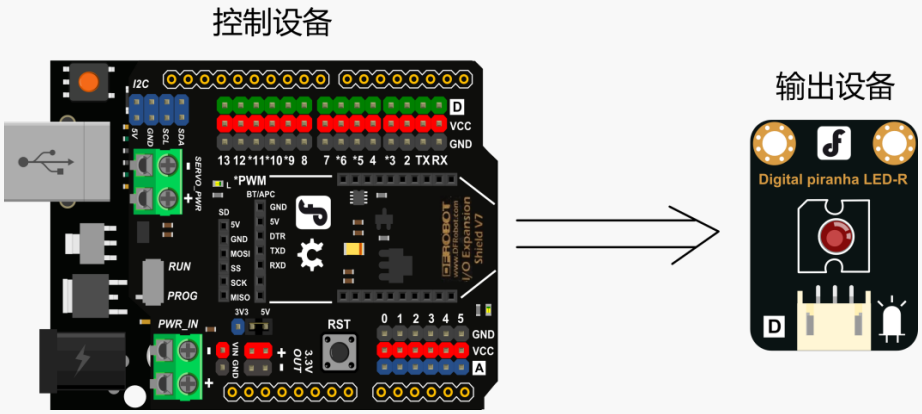
```
// Item One —— LED Blink

/*
Description: LED light on first; LED light off one second later; loop the above
action
*/

int ledPin = 13;
void setup() {
        pinMode(ledPin, OUTPUT);
}
void loop() {
        digitalWrite(ledPin,HIGH);
        delay(1000);
        digitalWrite(ledPin,LOW);
        delay(1000);
}
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 101。上传完成后你就可以看到外接的 LED 灯开始闪烁了！

现在让我们来分析代码看看它是如何工作的。

硬件分析（数字量输出）

在之前的章节中，我们提到了输入和输出，而这个实验中整个设备由两个部分组成：控制模块和输出模块，101 开发板是控制模块而数字食人鱼红色 LED 发光模块充当输出单元，同时这个系统中没有输入单元。



代码分析

Arduino 的代码由两个部分组成：

```
"void setup() {}"
```

当设备初始化时这部分代码将被执行并只运行一次，这部分代码被用来声明设备里的输入和输出。

```
"void loop() {}"
```

这部分代码在"void setup{}"之后执行，并且初始化之后，这部分代码会一直重复运行下去（直到设备断电）。所有的关键程序步骤都包含在这部分执行。

具体分析这部分代码:

```
pinMode(ledPin, OUTPUT);
```

上面这部分代码把 ledPin 设置为输出模式，括号中的逗号是代码的语法，用来分开参数。

ledPin 又是什么呢？

回到代码的第一行，如下：

```
int ledPin = 13;
```

这意味着我们把 13 引脚命名为“ledPin”，因此 ledPin 代表 101 板子上的 13 号数字信号引脚。“int”代表整数，指“ledPIn”是一个整数。

如果把 LED 模块接到 10 号引脚而不是 13 引脚会怎样呢？

```
pinMode(10, OUTPUT);
```

如果这样的话，我们就需要把代码中的数字 13 改成 10。

让我们来看看“loop()”函数，这里面只包含一个函数 digitalWrite()。

函数“digitalWrite()”的语法如下：

```
digitalWrite(pin,value)
```

在我们设置 pinMode()为 OUTPUT 输出后，电压要么是高电平（5V 或者 3.3V）要么是低电平（0V）。高电平意味着开而低电平意味着关。

```
digitalWrite(ledPin,HIGH);   // LED on
digitalWrite(ledPin,LOW);    // LED off
```

上面语句中，在 digitalWrite()中写入 HIGH，13 引脚将输出高电平，LED 灯会被点亮。当 digitalWrite()中写入 LOW 时，13 引脚将输出低电平，LED 灯会熄灭。

最后别忘了下面这一句：

```
delay(1000);
```

括号中的数字代表延时的时间是多少毫秒。1 秒钟就是 1000 毫秒，所以上面的语句代表延长 1 秒钟时间。因此 LED 等会持续点亮 1 秒钟接着熄灭 1 秒钟，如此循环。

你会注意到代码开头的“//”和“/*…*/”：

```
// Item One—LED Blink
/*
Description: LED on and off alternately every other second
*/
```

所有写在“//”后面的和写在“/*…*/”之间的都不会被编译，我们使用这个功能来给程序写简单的注释。“//”只会注释掉一行语句而“/*…*/”则可以注释掉多行语句。Arduino IDE 会使被注释掉的语句变灰来提示你。

牛刀小试

试着让 LED 灯闪烁地更快或者更慢。例如，使 LED 灯熄灭 5 秒钟然后按照 250 毫秒间隔快速闪烁。

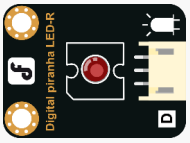
项目二 感应灯

在这一部分，我们将使用人体红外热释电传感器实现如下功能：

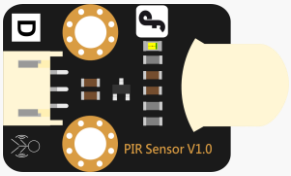
- 当传感器检测到人靠近时，LED 灯会点亮；
- 没有检测到人的动作时，LED 灯会熄灭。

所需物品

1 × 数字食人鱼红色 LED 发光模块



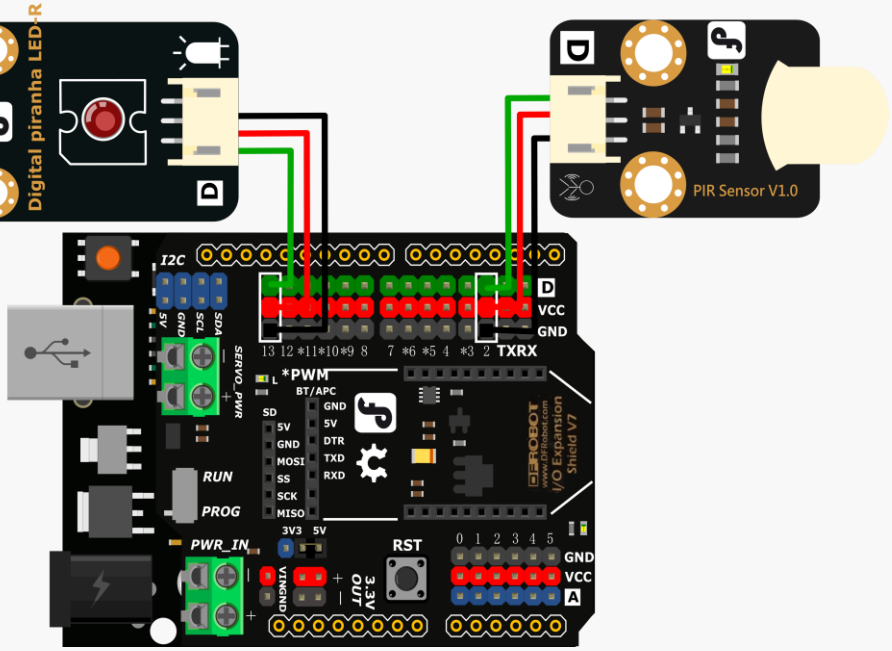
1 × 人体红外热释电运动传感器



硬件连接

我们需要连接以下部分：

- 连接红外线传感器到开发板上的数字引脚 2；
- 连接数字食人鱼红色 LED 发光模块到数字引脚 13；



输入代码

示例程序 2-1:

```
// Item Two—Sensor Light
int sensorPin = 2;      // IR sensor to be attached to digital pin NO.2
int ledPin =  13;   // Digital Piranha LED-R to be attached to digital pin NO.13
int sensorState = 0;      // variable sensorState for storing statistics about
the status of the sensor

void setup() {
  pinMode(ledPin, OUTPUT);         // LED is the output unit
  pinMode(sensorPin, INPUT);      // Sensor is the input unit
}
void loop(){
  sensorState = digitalRead(sensorPin);    // Read statistics collected from the
sensor
  if (sensorState == HIGH) {       // If it is HIGH, LED light will be turned on
    digitalWrite(ledPin, HIGH);
  }
  else {                           // If it is LOW, LED light will be turned off
    digitalWrite(ledPin, LOW);
  }
}
```

使用上述示例代码来实现我们所需要的功能。

你可以将其复制并粘贴到 Arduino IDE 中，但我们建议手动输入一遍已获得更好的理解。

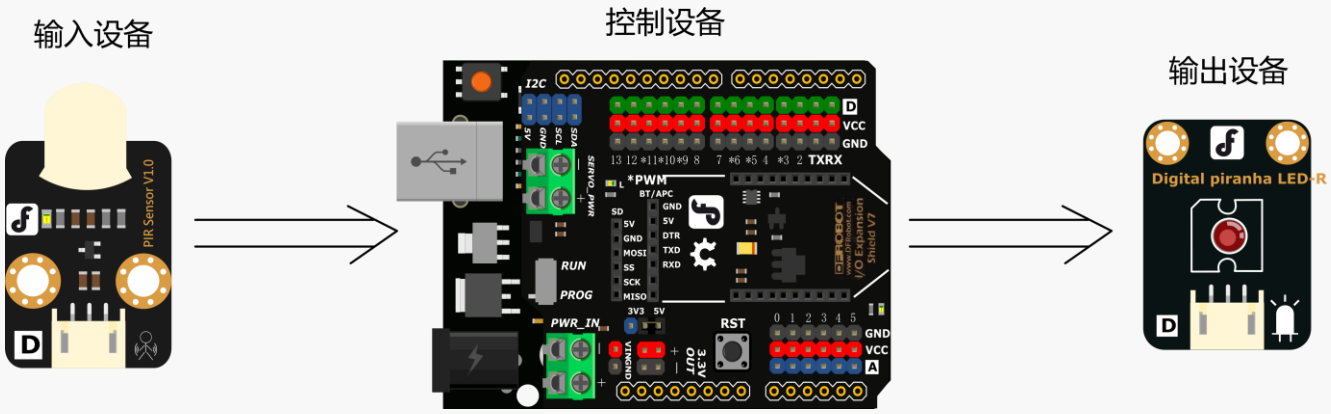
输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

之后就可以检测程序是否运作啦，当传感器检测到人体靠近时，LED 会发光；而没有人体靠近时，LED 会熄灭。

硬件分析（数字量输入 – 数字量输出）

整个装置由三个部分组成：输入单元，控制单元和输出单元。

人体红外热释电运动传感器作为输入单元，Arduino 开发板作为控制单元而数字食人鱼红色 LED 发光模块则作为输出单元。红外传感器和 LED 灯使用数字信号，所以它们都接在数字信号引脚上。



代码分析

把红外传感器设置为输入单元，LED 灯设置为输出单元。可以在程序的初始化部分设置。.

```
pinMode(ledPin, OUTPUT);          // Define LED as the output unit
pinMode(sensorPin, INPUT);       // Define sensor as the input unit
```

在 loop()函数里写入"digitalRead()"函数读取数字引脚处电压值。

```
sensorState = digitalRead(sensorPin);
```

函数格式如下：

```
digitalRead(pin)
```

这个函数是用来读取数字引脚出的状态，即“高电平”或者“低电平”。当人体红外热释电运动传感器检测到人体靠近时，引脚状态为高电平。（高电平即 1，低电平即 0）.

另一部分代码则根据传感器检测到的状态决定需要响应的动作。

数字式的传感器只能读出两个值（高或者低，即 1 或者 0），因此我们使用 if 判断语句执行。

“if”语句通常和比较运算符用在一起，比如"=="（相等），"!="（不等），"<"（小于）或者">"（大于）。这使得程序在计算出表达式的值后可以进行逻辑运算，通常当达到某个特定条件就做某事时我们会使用“if”语句做判断。

我们就来看下 if 语句在我们代码中的应用：

```
  if (sensorState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else{
    digitalWrite(ledPin, LOW);
  }
}
```

在上述代码中，if 语句判断传感器的状态是高还是低，如果是高电平，就会给 LED 灯高电平使其发光。

当 if 语句中条件不成立时可以使用“else”语句来执行另一种可选的输出——如果传感器输出为低电平，LED 灯就熄灭。

下面给出了有关判断语句更具体的介绍。

- x == y (x 等于 y)
- x != y (x 不等于 y)
- x < y (x 小于 y)
- x > y (x 大于 y)
- x <= y (x 小于等于 y)
- x >= y (x 大于等于 y)

在代码中 Arduino 会根据如上语句进行数学计算并做出逻辑判断。

牛刀小试:

(1) 你可以使用闪烁 LED 灯试着做一个幽灵面具。你可以改变 LED 灯闪烁的时间间隔，使他们为了不同效果闪得更快或者更慢，这在废弃的房子里使用效果一定很棒！

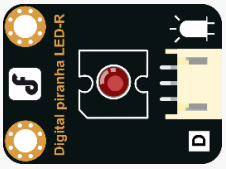
(2) 你可以使用红外传感器制作一个装饰用的灯，点击查看教程。

项目三　迷你台灯

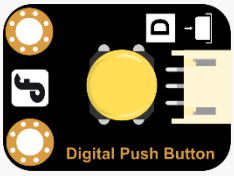
台灯是我们的日常用品，只需按下开关它就能给我们带来光明。让我们自己来做一个迷你台灯吧。

所需元件

1 × 数字食人鱼红色 LED 发光模块



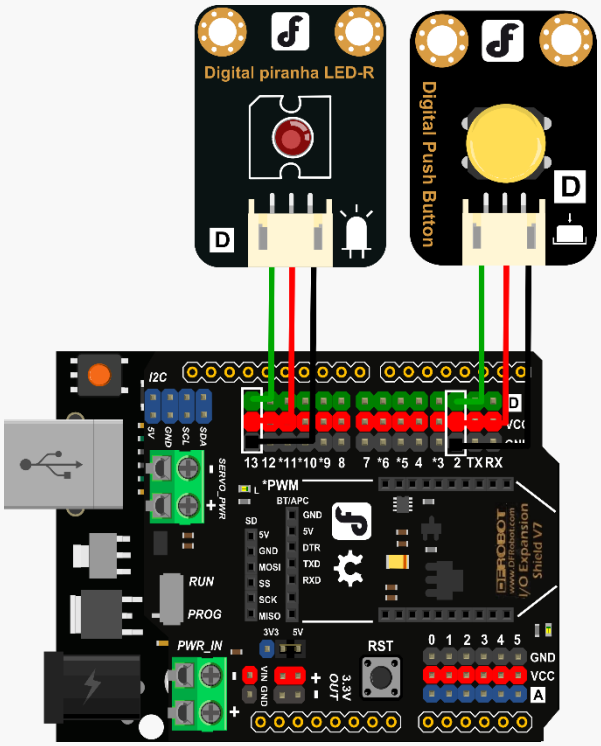
1 × 数字大按钮模块



硬件连接

把数字大按钮模块接到数字引脚 2 上；

把数字食人鱼红色 LED 发光模块接到数字引脚 13 上。



输入代码

示例程序 3-1:

```
// Item Three—Mini Lamp
int buttonPin = 2;              // button to be connected to digital pin No.2
int ledPin = 13;               // LED to be connected to digital pin No.13
int ledState = HIGH;            // ledState records the state of LED light
int buttonState;                //buttonState records the stat of the button
int lastButtonState = LOW;      // lastbuttonState records the previous status
of the button
long lastDebounceTime = 0;
long debounceDelay = 50;        // eliminate the oscillation time

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
}

void loop() {
    //reading stores buttonPin's data
    int reading = digitalRead(buttonPin);
   //record the time when data changes
  if (reading != lastButtonState) {
       lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
           ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
   //Change the button's previous state value
  lastButtonState = reading;
}
```

使用上述示例代码来实现我们所需要的功能。

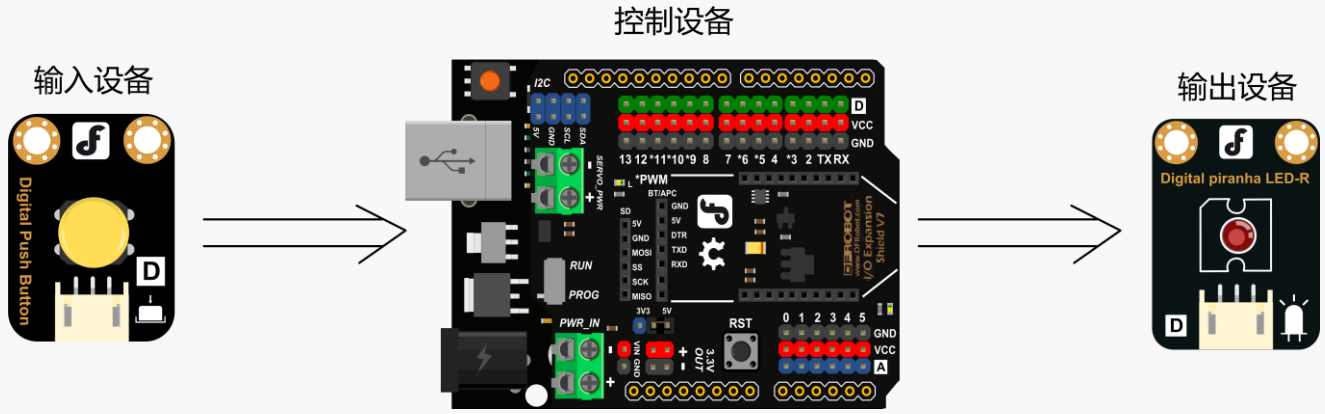
你可以将其复制并粘贴到 Arduino IDE 中，但我们建议手动输入一遍已获得更好的理解。

输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

代码上传完成后，按下按钮，LED 灯会点亮。当再次按下按钮后，LED 灯会熄灭。

硬件分析（数字量输入 – 数字量输出）

在整个装置中，数字大按钮模块是输入单元而 LED 灯是输出单元，就像我们上次实验一样，我们有一个数字输入和一个数字输出。



代码分析

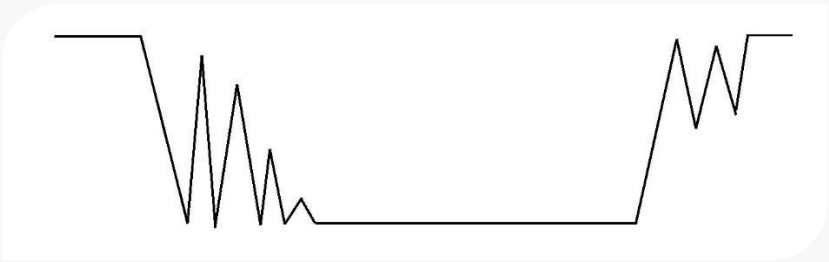
在初始化程序中设置按钮为输入设备而 LED 灯为输出设备：

```
pinMode(buttonPin, INPUT);
pinMode(ledPin, OUTPUT);
```

使用 digitalWrite() 函数读取按钮的状态值：

```
int reading = digitalRead(buttonPin);
```

在按钮在高电平和低电平之间切换时会产生一段信号震荡，就像下图所示：



为了避免上述的信号误差干扰，我们需要在信号切换的时候等待电路稳定后再读取按钮状态。

函数 millis()可以记录收集到的状态数据发生改变时的时间值。

```
if (reading != lastButtonState) {
   lastDebounceTime = millis();
}
```

“函数 millis()” 可以测量持续时间. 其测量单位为毫秒（1 秒 = 1000 毫秒）。当 101 开发板启动时，该函数开始计时。

按钮状态发生改编后，等待 50ms 至电路稳定，检查按钮信号状态是否和上次记录状态相同。如果不同，就改变按钮记录状态，同时，如果按钮状态为高电平（意味着按钮被按下），就改变 LED 灯的状态。

```
if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      if (buttonState == HIGH) {
           ledState = !ledState;
      }
    }
}
```

牛刀小试

带迷你灯的门铃

现而今，人们经常带着耳机听音乐，因此很多时候他们在家带着耳机而门铃响了却不能听到。

带迷你灯的门铃则可以解决这个问题。如果门铃按钮被按下了，台灯同时也会被点亮。

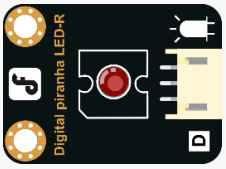
这个创意同样适合帮助有听力障碍的老年人。

项目四　声控 LED

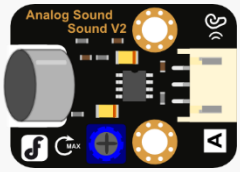
在这一章节我们将做一个声控 LED，你可以拍手控制 LED 闪烁。要实现这个功能，我们需要一个声音传感器。除了这个实验之外，我们还可以使用声音传感器做各种各样声控设备。

所需元件

1 × 数字食人鱼红色 LED 发光模块



1 × 模拟声音传感器

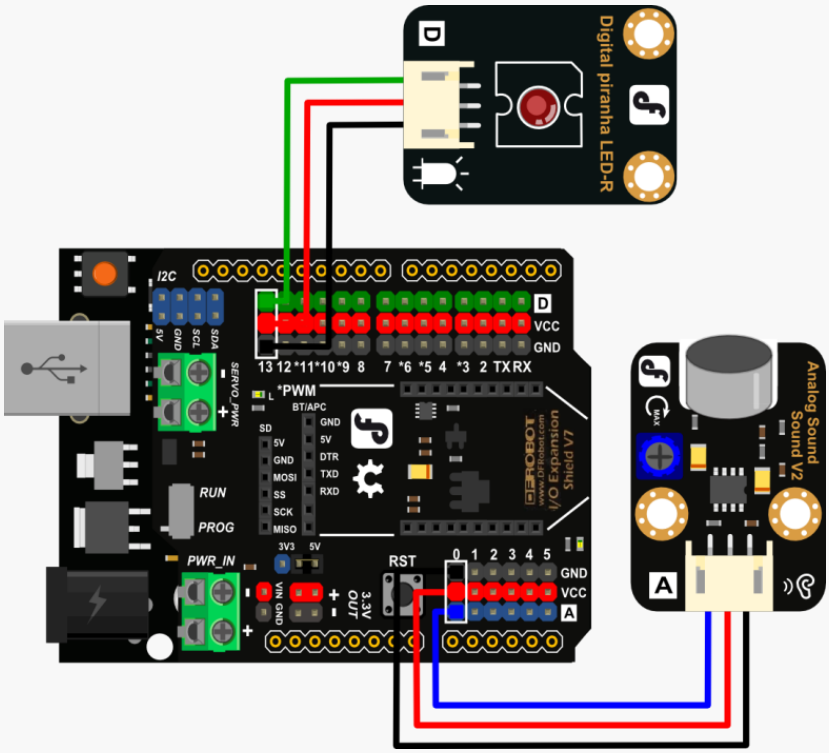


硬件连接

把模拟声音传感器接在模拟信号引脚 0 口上；

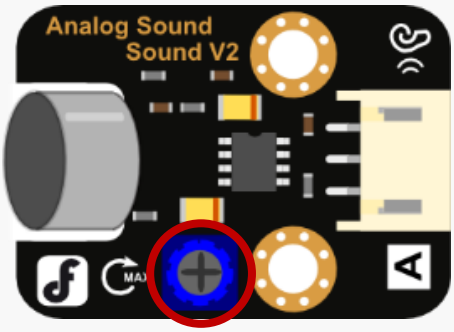
把数字食人鱼红色 LED 发光模块接在数字引脚 13 口上。

参考下面的接线图：



声音传感器使用

有时候第一次使用设备时可能不能很好地运作，这是因为设备的灵敏度被设置地太高或者太低，使得 LED 灯一直点亮或者一直熄灭。这时候可以通过调整板子上的电位器调整灵敏度直到设备正常工作。如下：



输入代码

示例程序 4-1:

```
// Item Four—Sound Control Light
int soundPin = 0;       // Analog sound sensor is to be attached to analog pin 0
int ledPin =  13;       // Digital Piranha LED-R is to be attached to digital
pin 13.
void setup() {
pinMode(ledPin, OUTPUT);
// Serial.begin(9600);    // For monitoring
}
void loop(){
int soundState = analogRead(soundPin);  // Read sound sensor's value
// Serial.println(soundState);      // serial port print the sound sensor's
value

// if the sound sensor's value is greater than 10, the light will be on for
10s. Otherwise, the light will be turned off
if (soundState > 10) {
     digitalWrite(ledPin, HIGH);
     delay(10000);
  }else{
     digitalWrite(ledPin, LOW);
  }
}
```

输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

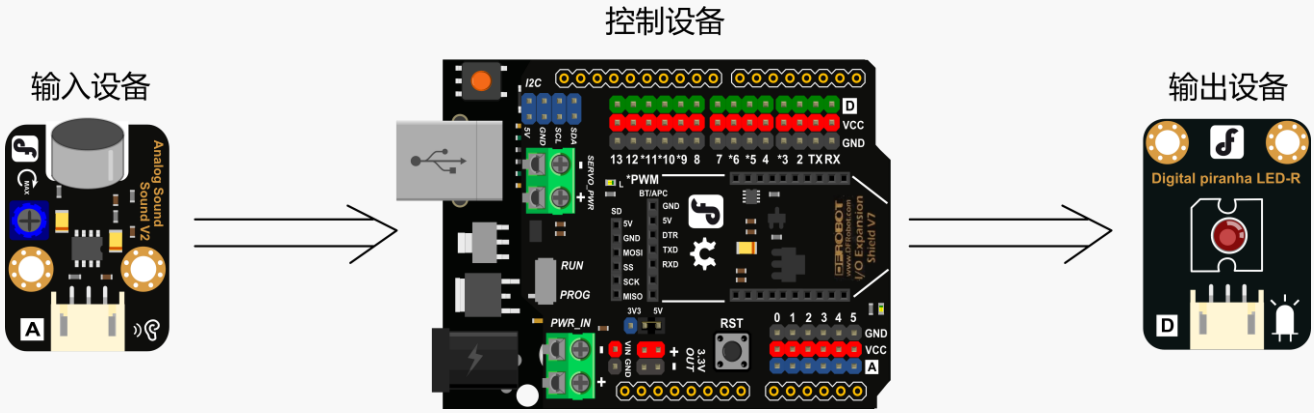
程序上传完成后，试着拍手并观察。此时 LED 应该会点亮，而没有声音时，LED 灯就会熄灭。

硬件分析（模拟量输入 – 数字量输出）

在前面章节中我们讨论了数字信号和模拟信号，模拟声音传感器，就如名称所示，是一个模拟量的传感器。

数字信号只有两个值：0 和 1；
模拟信号有很多值，从 0 到 1023 之间都可。

更多的信息可以在第二章教程中找到。



代码分析

在 setup()程序中 LED 被设置为输出模式，但是为什么不把声音传感器设置为输入呢？那是因为所有的模拟量引脚都是输入所以我们不需要设置。

声音传感器是一个输入单元，它的状态值可以从模拟量引脚 0 口读取。读取函数如下：

```
analogRead(pin)
```

这个函数的作用是读取模拟量引脚的值。Arduino 的模拟量引脚跟一个 A/D 转换器相连，所以 0 到 5V 的电压信号可以被对应到数字 0 到 1023。每个收集到的数字都代表一个电压，例如，512 = 2.5V。

最终，使用 if 语句判断读取结果是不是大于某个值来判断有没有明显的声音。

```
if (soundState > 10) {
      ...
}else{
      ...
}
```

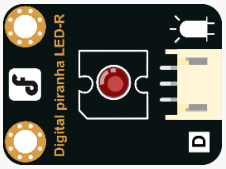
你可以打开串口监视器，测试一下最能代表你所需要声音的临界值，之后可以在代码中相应地修改。

项目五 呼吸灯

在这一章节中，从 LED 灯简单闪烁更进一步，我们将学习如何调整 LED 灯的亮度并添加渐变效果。
我们也会引入 PWM（脉冲宽度调制）的概念，并解释它是如何通过数字信号引脚来输出模拟信号的。

所需元件

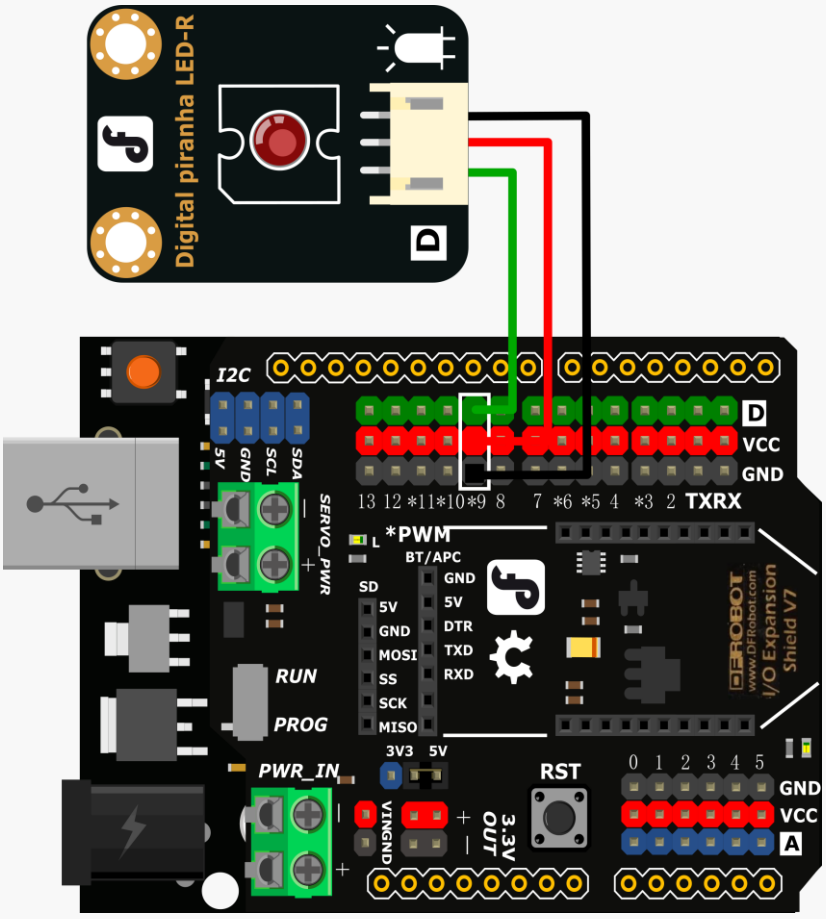
1× 数字食人鱼红色 LED 发光模块



硬件连接

把数字食人鱼红色 LED 发光模块接到数字量引脚 9 口上。注意 Genuino 101 只有 4 个 PWM 口，分别是 3,5,6,9。后面会详细介绍 PWM。

参考以下的接线图：



输入代码

示例程序 5-1:

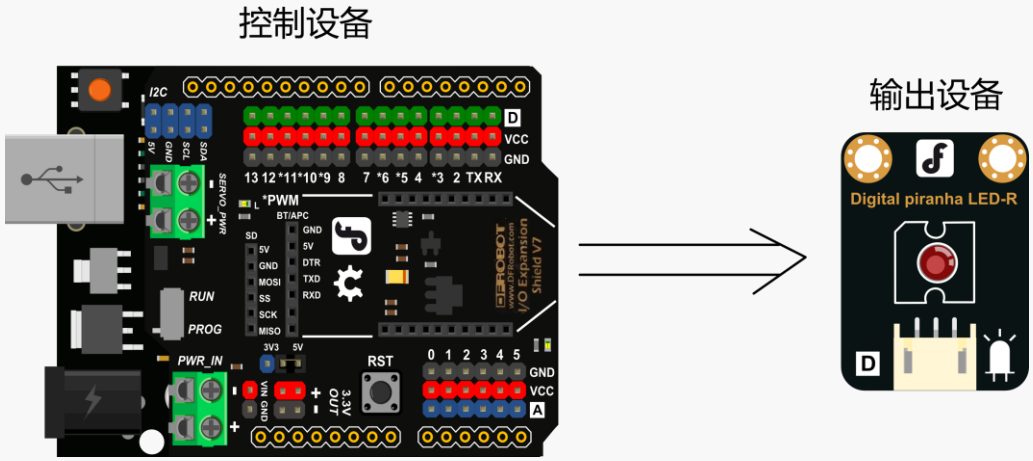
```
int ledPin = 9;

void setup() {
     pinMode(ledPin,OUTPUT);
}

void loop(){
  for (int value = 0 ; value < 255; value=value+1){
     analogWrite(ledPin, value);
     delay(5);
  }
  for (int value = 255; value >0; value=value-1){
     analogWrite(ledPin, value);
     delay(5);
  }
}
```

输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

当代码上传完成后，你可以看到 LED 灯渐渐地变亮又变暗，如此往复。

硬件分析（模拟量输出）

装置很像我们在项目 1 中闪烁小灯时的装置，同样只有一个输出单元。本项目中的不同之处就在于我们使用了 LED 作为模拟量输出而不是数字量输出。了解更多可以参考代码模块。



代码分析

我们使用 for 语句重复某些代码，其结构如下：

```
for(init; condition; update){
    statement(s);
}
```

具体解释：

1. “init”语句首先执行，并且只执行一次；
2. 判断“condition”条件是否为真。如果不为真，则停止循环。
3. 执行“statement”语句。
4. 执行“update”语句，回到第二步继续。

回到代码上来：

```
  for (int value = 0; value < 255; value=value+1){
          ...
  }
  for (int value = 255; value >0; value=value-1){
          ...
  }
```

两个 for 循环分别使 value 的值从 0 增加到 255 和从 255 减小到 1。

现在来看看函数 analogWrite()。

数字量引脚只有两个值：0 或 1。那我们如何通过数字量引脚发送模拟量呢？下面就该 PWM 登场啦。101 开发板上有四个数字量引脚被标上了“~”，这代表它们可以发送 PWM 信号。

analogWrite()函数格式如下：

```
analogWrite(pin,value)
```

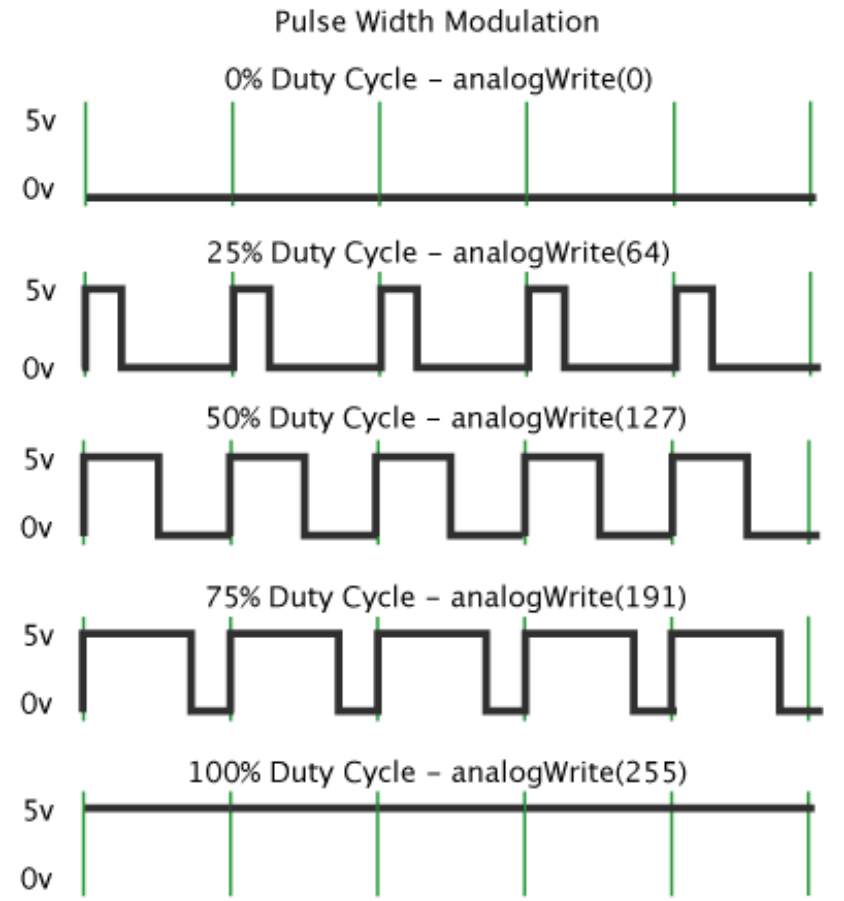
函数 analogWrite()可以给 PWM 引脚发送 0 到 255 之间的模拟量，同时只能用于 PWM 引脚上，在开发板上是引脚 3，5，6，9。

什么是 PWM？

PWM 代表脉冲宽度调制，简单地说，PWM 通过在数字量引脚快速切换高低电平得到模拟信号。我们可以使用 PWM 模拟任意 0V 到 VCC 之间的电压。

如果使用示波器观察 PWM 信号的话，你可以看到一系列方波，方形顶部代表高电平，底部代表低电平。方波的宽度越大，就代表高电平脉宽越宽—即脉冲宽度调制名称的由来。

通过以下 5 个方波来更深入地理解 PWM 吧，我们以 5V 为例说明：



图上的绿色垂直线条代表时间间隔，黑色垂直线代表电脉冲。

每个“analogWrite(value)”中的 value 值都可以对应到一个特定的百分比，也叫做占空比。占空比代表着脉冲从低到高和从高到低的时间长度。

第一个图中方波的占空比为 0%，对应的 value 值也为 0。如果占空比为 100%，那么 value 值则为 255，像第五个图一样。占空比越大，输出电压越高，LED 灯就会越亮。

手动设置 PWM

为了帮助你更好的理解，我们可以使用下面的程序手动设置占空比。在每个循环段内设置高电平型号的时间长度就可以做到这一点，当循环段的总时段足够短时，也就意味着 LED 闪烁地足够快，我们就认为 LED 灯被点亮。

实例代码:

```
int ledPin = 10;
void setup() {
     pinMode(ledPin,OUTPUT);
}

void loop(){
  for (int duration = 0 ; duration < 20; duration=duration+1){
     digitalWrite(ledPin, HIGH);
     delay(duration);
     digitalWrite(ledPin, LOW);
     delay(20-duration);
}
  for (int duration = 20 ; duration > 0; duration=duration-1){
     digitalWrite(ledPin, HIGH);
     delay(duration);
     digitalWrite(ledPin, LOW);
     delay(20-duration);
  }
}
```

PWM 可以用在调整 LED 灯的亮度，或者调整直流电机的速度上。你还能想到其他的用法吗？

使用相同的硬件连接但是不同的代码，完全可以得到 LED 灯的不同效果。试着调整下代码，看看还能不能得到其他的玩法，发挥你的想象力吧！

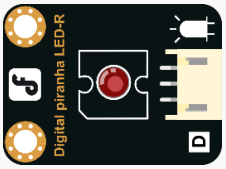
项目六 灯光调节器

我们可以使用灯光调节器来控制灯光的亮度。在本章节中，我们将使用模拟角度传感器实现这一功能。灯光会随着旋钮旋转角度变化而从暗变亮或从亮变暗，旋转过的角度越大，LED 灯就越亮，反之亦然。

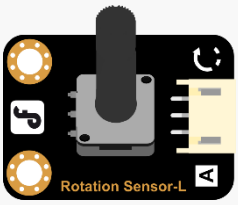
角度传感器同样可以用来控制舵机轴的角度，以及直流电机的旋转速度。除此之外它还有很多用处，开动脑筋去发现吧。

所需元件

1 × 数字食人鱼红色 LED 发光模块



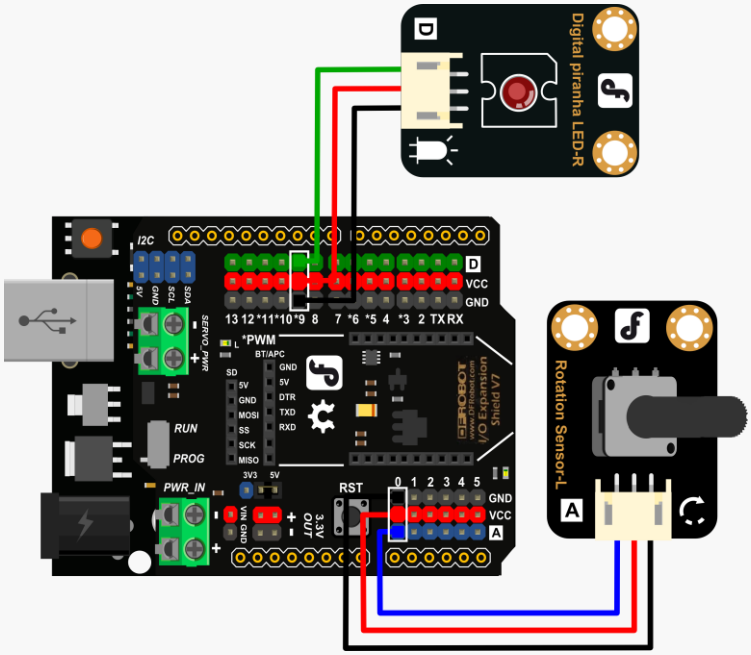
1 × 模拟角度传感器



硬件连接

把模拟角度传感器接在模拟量引脚 0 上；

把数字食人鱼红色 LED 发光模块接在数字量引脚 9 上。



输入代码

示例程序 6-1:

```
// Item Six—Light -regulator
int potPin = 0; // Analog Rotation Sensor shall be attached to analog pin 0
int ledPin = 9; // LED shall be attached to digital pin 9
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  int sensorValue = analogRead(potPin); // collect value at analog pin 0
  // A value between 0 and 1023 to mapped to a value between 0 and 255
  int outputValue = map(sensorValue, 0, 1023, 0, 255);
  analogWrite(ledPin, outputValue); // write corresponding value for LED
  delay(2);
}
```

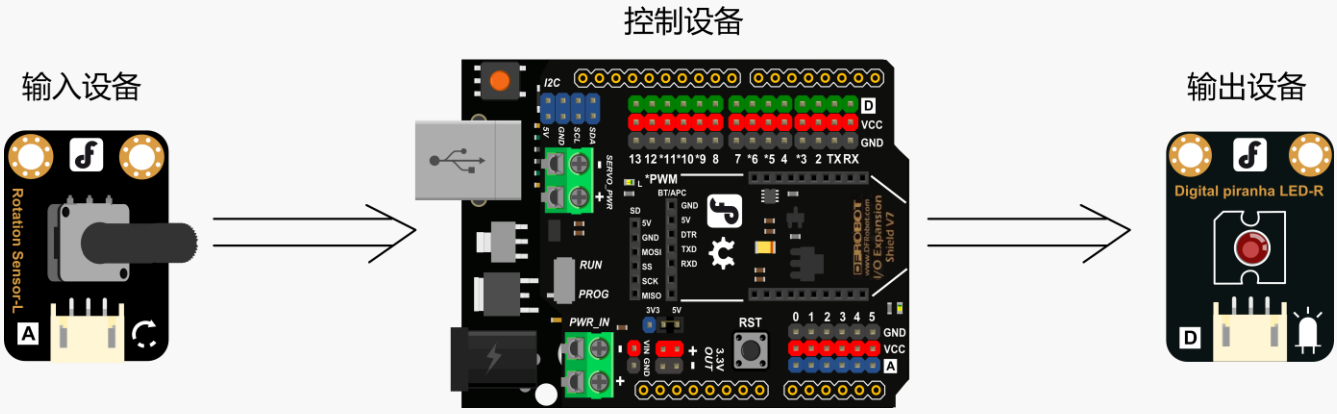
输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

慢慢旋转角度传感器上电位器的轴并观察，LED 灯的亮度会随着你的旋转而变化。

硬件连接（模拟量输入 – 模拟量输出）

在上一章节中我们学会了如何使用 PWM 在数字量引脚输出模拟信号，在本章节中，我们将使用模拟量的输入控制该模拟量的输出。

在装置中，角度传感器是输入单元，LED 灯是输出设备，它们两者都使用模拟信号。



代码分析

在本程序中，我们将使用 map 函数，该函数可以把某个范围内的值对应到另一个范围中去，让我们来仔细看看它的应用吧：

函数语法如下：

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

该程序把“fromLow”到“fromHigh”中的一个数值对应到“toLow”和“toHigh”中去。

Map 函数中参数含义：

value： 需要对应的数值

fromLow：对应前的范围的下限

fromHigh：对应前的范围的上限

toLow：对应后的范围的下限

toHigh：对应前的范围的上限

map 函数的一个优点是两个范围的下限都可以比它们的上限大：

```
y = map(x, 1, 50, 50, 1);
```

同时 map 函数中的数值可以是一个负值：

```
y = map(x, 1, 50, 50, -100);
```

让我们再来分析下代码：

```
int outputValue = map(sensorValue, 0, 1023, 0, 255);
```

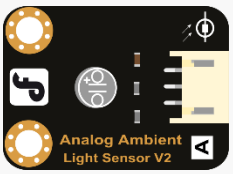
上述代码用来把从模拟量引脚接收到的模拟数值（0~1023）转化到 PWM 引脚可以用的范围（0~255）。

项目七 夜光宝盒

夜光宝盒，听着名字是不是很好玩，实际也是这么好玩儿！我们要做的这个盒子，在白天是闭合的，一旦进入了深夜，就开始慢慢张开，灯光也会慢慢变亮，好似一颗“夜明珠”，一旦到了白天，有慢慢合上了！哈哈…先来大致说下原理吧！要完成这个功能，我们需要使用一个新的元件——舵机，它可以进行精确的角度控制。另外，我们需要一个模拟环境光线传感器来检测盒子外面的光线强度。当检测到夜晚来临时，舵机就会启动来打开盒子，而盒子内部的 LED 灯也会渐渐发亮。

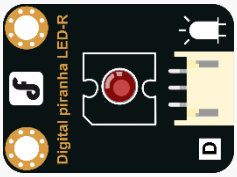
所需元件

1 × 模拟环境光线传感器 V2



1 × TowerPro SG50 舵机

1 × 数字食人鱼红色 LED 发光模块



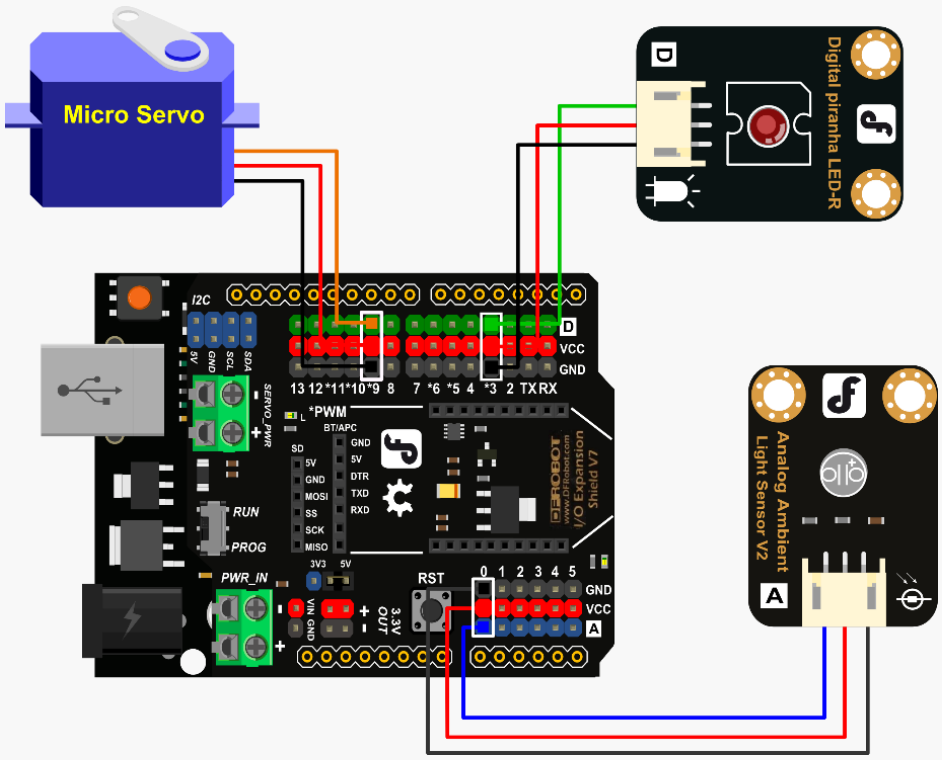
硬件连接

把 TowerPro SG50 舵机接在数字信号引脚 9 号口；

把模拟环境光线传感器 V2 接在模拟信号引脚 0 号口；

把数字食人鱼红色 LED 发光模块接在数字信号引脚 3 号口；

舵机是一个很常见的旋转输出机构，被广泛地运用于机器人控制领域。与其他常规电机不同的是，舵机在轴上装有电位器来监控角度位置，并且拥有内部的控制回路，这使得舵机可以精确移动到指定角度。



代码输入

示例程序 7-1:

```
#include <Servo.h>
Servo myservo;
int LED = 3;   //Set the LED light to be digital pin 3
int val = 0;   // val stores analog ambient light sensor's value
int pos = 0;
int light =0;

void setup(){
    pinMode(LED,OUTPUT);         // LED is set to be in the output mode
    Serial.begin(9600);          //Baud rate of the serial port is set to be 9600
    myservo.attach(9);           // attach the servo to digital pin 9
    myservo.write(0);            // initial angle is 0
}

void loop(){
  val = analogRead(0);         //read the analog ambient light sensor's value
  Serial.println(val);         //Check sensor value in serial port
  if(val<40){  // when val is less than the pre-set value, increase the angle
         pos = pos +2;
         if(pos >= 90){       //The angle should not be more than 90
             pos = 90;
         }
         myservo.write(pos);      //write angle of the servo
         delay(100);
         //As the angle expands, the LED becomes brighter
```

```
        light = map(pos,0,90,0,255);
        analogWrite(LED,light);   //write the brightness value
    }else{
        pos = pos -2;    //decrease 2°
        if(pos <= 0){
           pos = 0;      //until 0°
        }
        myservo.write(pos);     //write angle of the servo
        delay(100);
        //As the angle shrinks, the LED light becomes darker
        light = map(pos,0,90,0,255);
        analogWrite(LED,light);  //write the brightness value
    }
}
```

使用上述示例代码来实现我们所需要的功能。

输入完成后，点击“编译”检查有没有语法错误。编译成功后便上传代码至 Arduino。

把舵机固定在盒子的连接处，灯塞在盒子里面，传感器当然是要露在外面的，需要检测环境光。安装完成后，把盒子置于暗处，看下盒子会不会自动打开。

如果对示例程序中的函数有疑问的话，可以参考之前的章节。

项目八 蓝牙连接

随着教程的深入，你可能会想要制作一些需要无线通信的设备，比如移动气象站或者随身心率监测器。而 101 主控器上板载的蓝牙 4.1 模块正赋予了 Genuino 101 与其他蓝牙设备（包括电脑、手机和其他蓝牙配件）的通信能力。

在本章节中，我们将通过蓝牙适配器上的串口建立 Genuino101 和电脑端的蓝牙连接，并通过串口监视器在 101 和 USB 蓝牙适配器之间收发数据。

所需元件

1x USB BLE-link 无线适配器

STEP1：蓝牙连接

在电脑 USB 口上插入 USB BLE-link。

USB BLE-link 是什么？

在开始之前，我们需要了解 USB BLE-link 是什么？USB BLE-link 是一款蓝牙 4.0 的 USB 无线适配器，蓝牙配对完成后，它会从 101 接受蓝牙数据并发送到电脑的 COM 端口。需要注意本套件中的 USB BLE-link 内部固件仅支持 101 开发板。

为了简便使用，USB BLE-link 和 101 的配对方式被设置为靠近配对。下面将详细介绍配对步骤。

设置蓝牙连接

在使用蓝牙功能之前，我们先学习如何配置蓝牙连接并显示连接状态。

首先，将以下程序上传到 Genuino101 中。

蓝牙通信功能需要库“BLESerial.h”的支持，该函数库可以在 DFRobot 官网产品页面下下载。

示例代码：

```
#include "BLESerial.h"

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(115200);    // initialize serial communication
  BLESerial.setName("Bluno101");
  BLESerial.begin();
  //Serial.println("Bluetooth device active, waiting for connection...");
  while(!BLESerial);
}
```

```
void loop() {
  while (BLESerial.operator bool())
  {
    digitalWrite(13, HIGH);
  }
  digitalWrite(13, LOW);
}
```

以上代码用于初始化 101 的蓝牙功能，并启动搜索附近蓝牙设备状态（即 USB BLE-link）。完成蓝牙连接后，D13 引脚的 LED 状态灯会点亮，反之将保持熄灭状态。

蓝牙接近配对

代码上传成功后，关闭身边所有其他蓝牙设备，在电脑 USB 端口插入蓝牙适配器。把 101 的蓝牙天线靠近蓝牙适配器（小于 20mm），等待两台设备上的蓝牙 LED 指示灯点亮（适配器上的蓝色 LED 指示灯可以在针孔附近找到）。当两个指示灯同时亮起并且保持点亮时，就代表蓝牙已经连接。配对完成后，即可将 101 移开， 蓝牙会保持配对状态。。

代码分析

我们使用库 BLESerial 来实现蓝牙数据传输。我们可以使用以下函数初始化蓝牙连接。

`BLESerial.begin()`

初始化 BLESerial 函数。

`BLESerial`

检查 BLESerial 是否被初始化，初始化完成后返回值为 1。

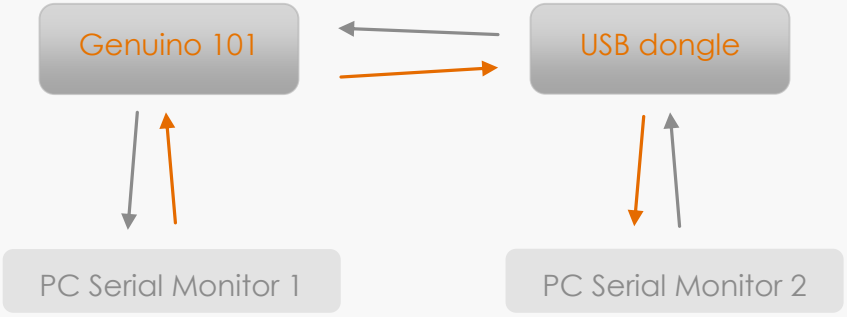
`BLESerial.operator bool()`

检查 101 开发板是否与适配器配对，配对完成后，返回值 1，否则返回值 0。

我们在 while 循环里添加了“digitalWrite(13, HIGH)”使蓝牙连接完成后 LED 保持常亮。

STEP2：蓝牙收发数据

我们将会在之前的基础上添加一些代码，使 101 开发板和 USB 蓝牙适配器能够互相发送数据。成功运行后，我们发送给蓝牙适配器的数据都将显示在 Genuino 101 的串口监视器上，反之亦然。



示例代码：

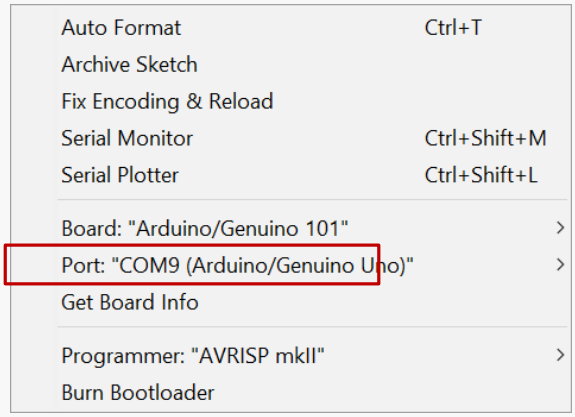
```
#include "BLESerial.h"

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(115200);    // initialize serial communication
  BLESerial.setName("Bluno101");
  BLESerial.begin();
  //Serial.println("Bluetooth device active, waiting for connections...");
  while(!BLESerial);
}

void loop() {
  while (BLESerial.operator bool()) {
    digitalWrite(13, HIGH);
    while(Serial.available()){
        BLESerial.write(Serial.read());
    }
    while(BLESerial.available()){
        Serial.write(BLESerial.read());
    }
  }
  digitalWrite(13, LOW);
}
```

设置串口通信

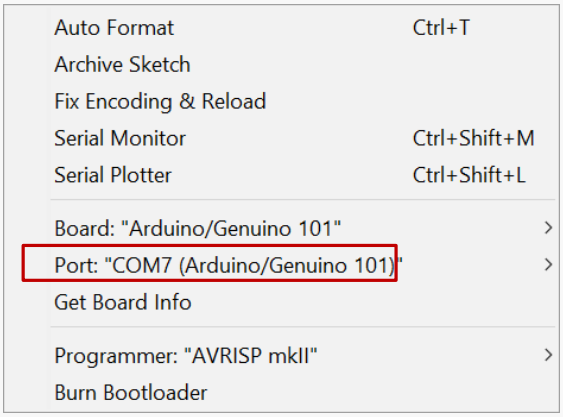
上传成功并配对完成（蓝牙指示灯保持常亮）后，打开两个串口监视器来测试程序。 要同时监控两个串口，我们必须分两次打开 Arduino.exe，并且窗口中分别选择对应的串口。

同时运行的两个 Arduino IDE 进程

之后，在两个 Arduino IDE 进程中分别选择对应 COM 端口。本例中 101 开发板为 COM7，USB 蓝牙适配器为 COM9。

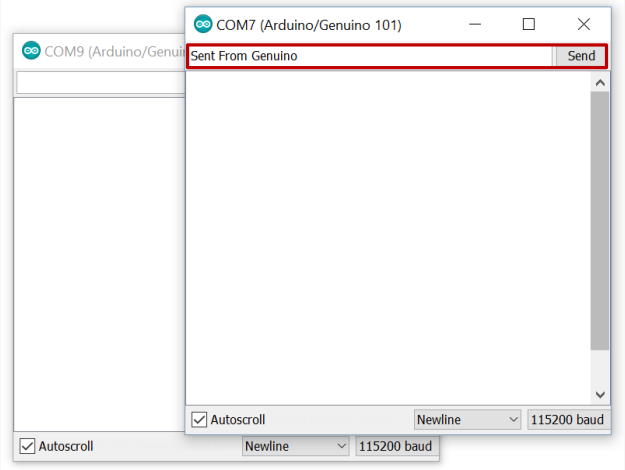


USB 蓝牙适配器在 COM9

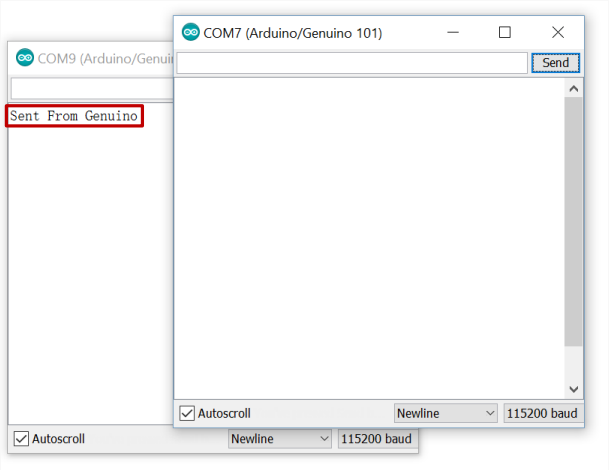


Genuino 101 在 COM7

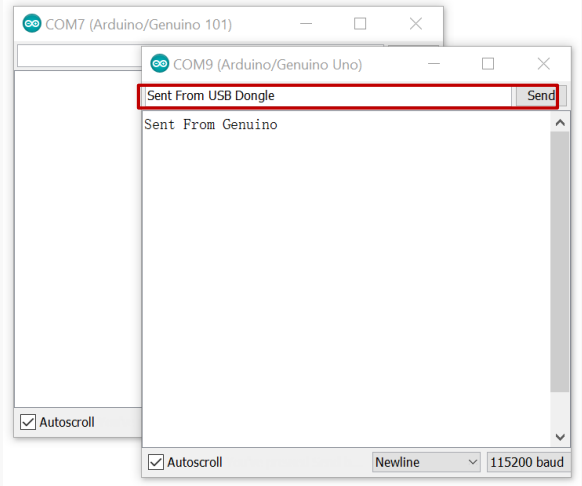
选择完成后打开串口监视器，并在其中一个输入栏中输入信息并点击"发送"。输入的我信息看到另一个监视器上显示。



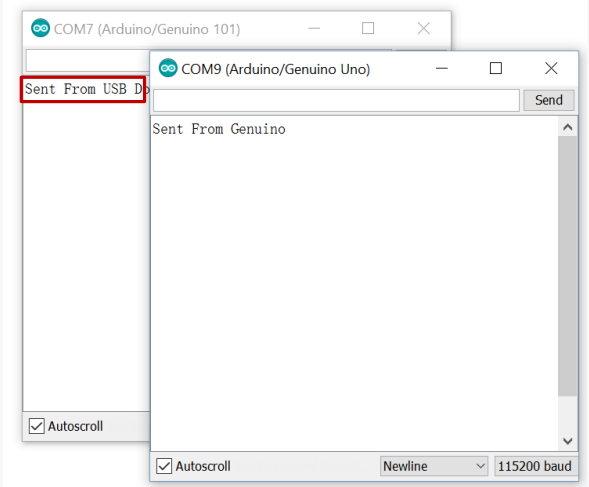
通过 Genuino 101 串口监视器发送字符串 (COM7)



通过 USB 适配器串口监视器接收字符串 (COM9)



通过 USB 适配器串口监视器发送字符串 (COM7)



通过 Genuino 101 串口监视器接收字符串 (COM9)

代码分析

数据传输函数的原理如下。检测到蓝牙连接后，101 开发板开始监测串口，如果串口有数据输入，就发送至蓝牙适配器并显示在其的串口监视器上。101 同时检测蓝牙串口。当接收到无线数据时就会显示在 101 的串口监视器上。

```
BLESerial.available()
```

该函数用于检测虚拟蓝牙串口有没有接收到数据，有的话就返回 1，否则返回 0。

```
BLESerial.write()
```

该函数通过蓝牙串口向蓝牙适配器发送数据。

```
BLESerial.read()
```

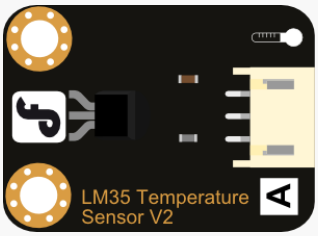
该函数通过蓝牙串口从 USB 蓝牙适配器读取数据，返回值即所读取的数据。

项目九 无线气象站

在学习了如何设置蓝牙连接后，我们可以尝试搭建一些简单的无线互动装置。在本章节中，我们将使用 101 搭建一个可以通过蓝牙传输时时温度的无线气象站。

所需元件

1 × 模拟 LM35 线性温度传感器

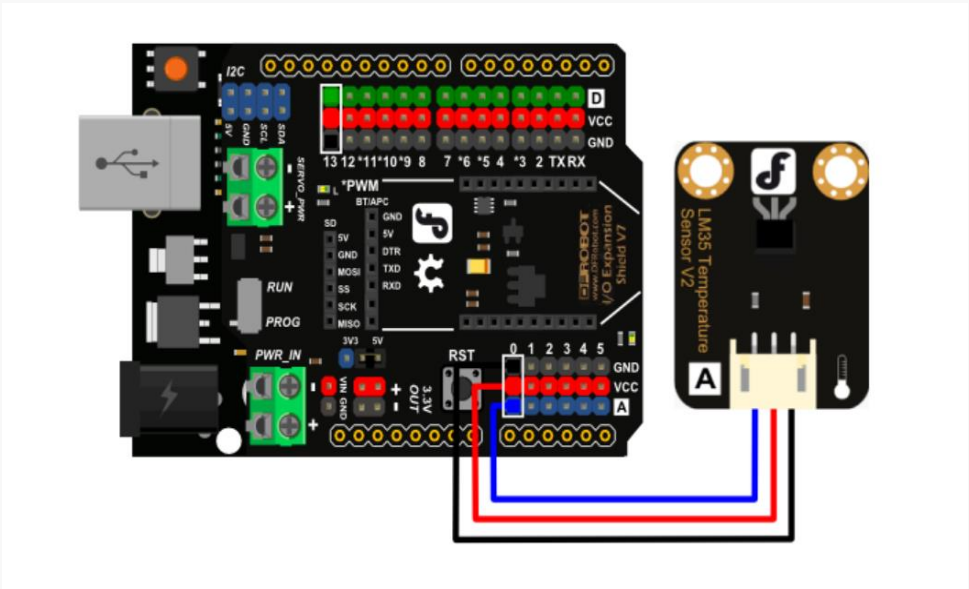


1x USB BLE-link 无线适配器

硬件连接

把 LM35 线性温度模拟值传感器接在 I/O 扩展板的模拟量引脚 0 号口上。

把 USB BLE-link 接在电脑 USB 端口上。



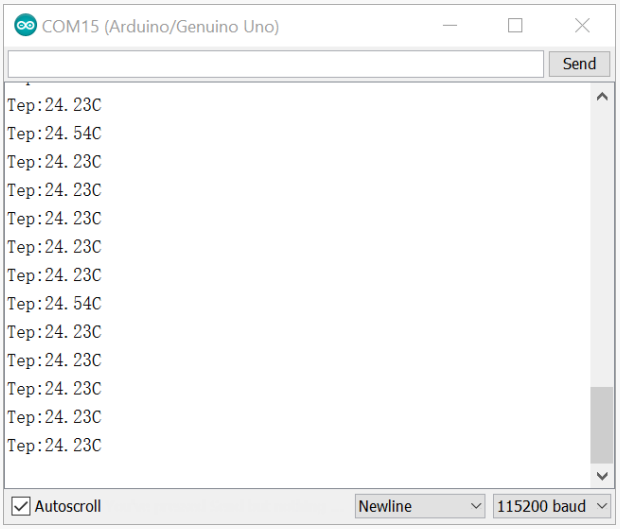
输入代码

```
#include "BLESerial.h"

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(115200);    // initialize serial communication
  BLESerial.setName("Bluno101");
  BLESerial.begin();
  //Serial.println("Bluetooth device active, waiting for connections...");
  while(!BLESerial);
}

void loop() {
  while (BLESerial.operator bool())
  {
    digitalWrite(13, HIGH);
    uint16_t val;
    double dat;
    val=analogRead(0);//Connect LM35 on Analog 0
    dat = (double) (val*（3.3/1024)*100);
    BLESerial.print("Tep:"); //Display the temperature on Serial monitor
    BLESerial.print(dat);
    BLESerial.println("C");
    delay(500);
  }
  digitalWrite(13, LOW);
}
```

设置连接

在 Arduino IDE 中，选择开发板类型为 Arduino/Genuino 101 和 COM 端口（本例中为 COM7），并上传程序。完成后，把 101 开发板和 USB 蓝牙适配器放在一起等待蓝牙指示灯亮起（正如我们在上一章中所做的那样）。

连接完成后，把 COM 端口号切换到 USB 蓝牙适配器接的端口上（本例中为 COM9），并打开串口监视器，环境温度的信息就会显示在监视器里。



从串口监视器读取的温度值

代码分析

本代码中我们没有改变蓝牙初始化部分，并添加了一些温度检测的代码。这些代码使 101 开发板通过蓝牙给 USB 蓝牙适配器发送数据。我们使用以下函数在 USB 蓝牙适配器的串口监视器上显示数据。

`BLESerial.print()`
在串口监视器里显示一个变量或字符串。

`BLESerial.println()`
在串口监视器里显示一个变量或字符串，然后换行。

项目十 芝麻开门

Genuino101 的另一个有趣的功能就在于它板载的六轴加速计和陀螺仪。本章节中，我们将教你如何读取加速机和陀螺仪的数据并把它们转化为实时姿态，并通过姿态来控制电机。

我们需要使用英特尔提供的库“CurieIMU.h”从加速计和陀螺仪读取数据，该函数库包含在开发板核心库里，所以我们不需要手动加载。

读取加速计数据

加速度计的测量值由重力加速度与自身加速度两部分组成。当开发板静止平置时，则会在竖直方向上得到一个正的定值，标示 101 仅受到竖直向下的重力加速度。当开发板处于运动状态时，测得的结果这是重力和实际加速度的矢量和。

输入代码

示例程序来自于“CurieIMU.h”库，可以在工具栏>文件>示例>CurieIMU>Accelerometer 里找到。

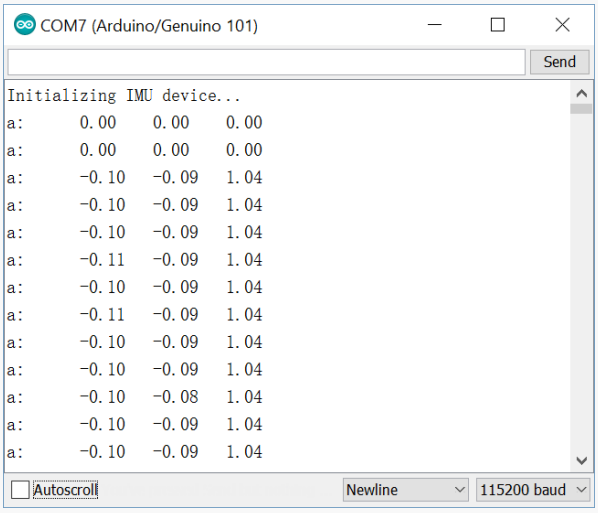
同样你可以复制以下代码使用。

```
#include "CurieIMU.h"

void setup() {
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open

  // initialize device
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();

  // Set the accelerometer range to 2G
  CurieIMU.setAccelerometerRange(2);
}

void loop() {
  int axRaw, ayRaw, azRaw;         // raw accelerometer values
  float ax, ay, az;

  // read raw accelerometer measurements from device
  CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);

  // convert the raw accelerometer data to G's
  ax = convertRawAcceleration(axRaw);
  ay = convertRawAcceleration(ayRaw);
  az = convertRawAcceleration(azRaw);

  // display tab-separated accelerometer x/y/z values
```

```
  Serial.print("a:\t");
  Serial.print(ax);
  Serial.print("\t");
  Serial.print(ay);
  Serial.print("\t");
  Serial.print(az);
  Serial.println();
}

float convertRawAcceleration(int aRaw) {
  // since we are using 2G range
  // -2g maps to a raw value of -32768
  // +2g maps to a raw value of 32767

  float a = (aRaw * 2.0) / 32768.0;

  return a;
}
```

串口读取运动状态

程序上传完成后，等待几秒钟直至 101 开发板完全启动，打开串口监视器。当开发板静止朝上放在水平桌面上时，得到的数据大致如下所示。测量单位为 g = 9.8m/s2.



代码分析

```
CurieIMU.begin();
```

初始化 CurieIMU 函数。

```
CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);
```

从装置读取未经转化的测量结果并把它们存入变量“axRaw”,” ayRaw” and “azRaw”。

```
ax = convertRawAcceleration(axRaw);
ay = convertRawAcceleration(ayRaw);
az = convertRawAcceleration(azRaw);
```

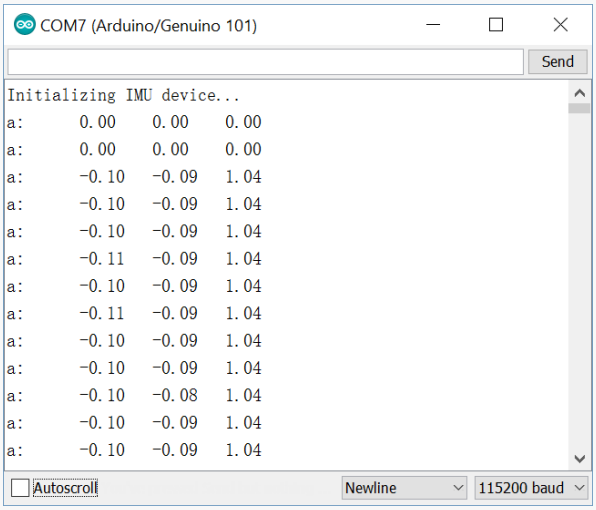
该函数把得到的数据除以常数(2.0* / 32768.0 in our case)，转化为单位为 g = 9.8m/s2 的标准化数据。

附加函数

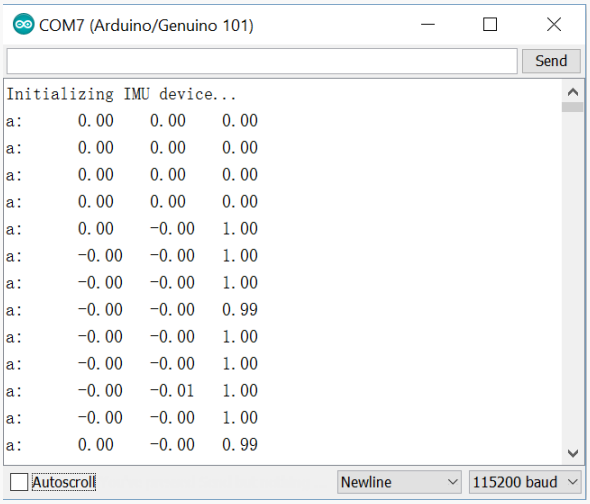
由于板子很难在静止时保持绝对水平，或者我们想要定义板子的其他方向为上方，我们就需要先设置好板子的上方向并设置好每个方向的偏置。可以在“CurieIMU.begin();”后添加如下代码来实现这一功能。

```
CurieIMU.autoCalibrateAccelerometerOffset(X_AXIS, 0);
CurieIMU.autoCalibrateAccelerometerOffset(Y_AXIS, 0);
CurieIMU.autoCalibrateAccelerometerOffset(Z_AXIS, 1);
```

在这里我们设置 Z 轴（垂直于开发板向下的方向）的值为 1 而其余的为 0，这就意味着板子在校准过程中是正面向上放置的。校准后得到的数据如下所示。



校准前测量结果



校准后测量结果

读取陀螺仪数据

陀螺仪读取三个正交方向的角速度，代表了物体转动得多快。

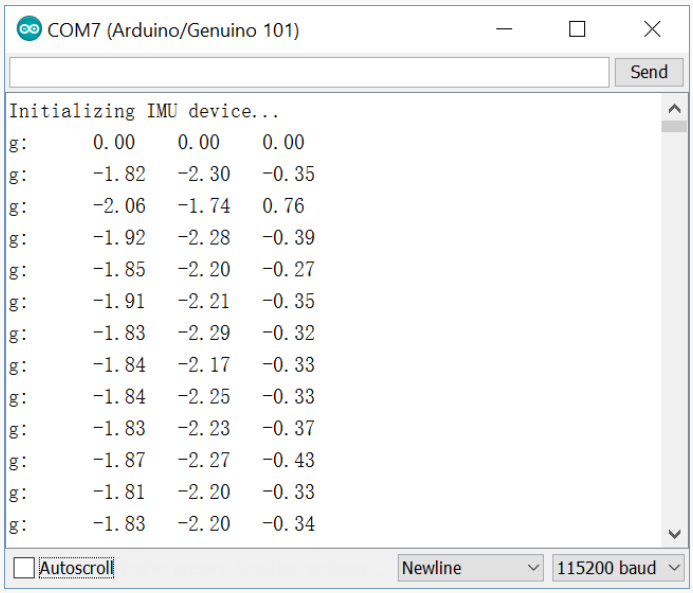
输入代码

示例程序来自于“CurieIMU.h”库，可以在工具栏>文件>示例>CurieIMU>Gyro 里找到。

```
#include "CurieIMU.h"

void setup() {
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open

  // initialize device
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();

  // Set the accelerometer range to 250 degrees/second
  CurieIMU.setGyroRange(250);
}

void loop() {
  int gxRaw, gyRaw, gzRaw;         // raw gyro values
  float gx, gy, gz;
  // read raw gyro measurements from device
  CurieIMU.readGyro(gxRaw, gyRaw, gzRaw);
  // convert the raw gyro data to degrees/second
  gx = convertRawGyro(gxRaw);
  gy = convertRawGyro(gyRaw);
  gz = convertRawGyro(gzRaw);
  // display tab-separated gyro x/y/z values
  Serial.print("g:\t");
  Serial.print(gx);
  Serial.print("\t");
  Serial.print(gy);
  Serial.print("\t");
  Serial.print(gz);
  Serial.println();
}
float convertRawGyro(int gRaw) {
  // since we are using 250 degrees/seconds range
  // -250 maps to a raw value of -32768
  // +250 maps to a raw value of 32767
  float g = (gRaw * 250.0) / 32768.0;
  return g;
}
```

串口读取测量值

程序上传完成后，等待几秒直至 101 开发板完全启动后打开串口监视器。因为角速度与重力无关，所以你可以以任意方向放置开发板。测量得到的数据大致如下所示，测量单位为 角度/秒。



代码分析

```
CurieIMU.readGyro(gxRaw, gyRaw, gzRaw);
```

该函数从陀螺仪读取未经处理的三个方向的角速度值，并存放在变量“gxRaw”,” gyRaw” and “gzRaw”里。

```
gx = convertRawGyro(gxRaw);
```

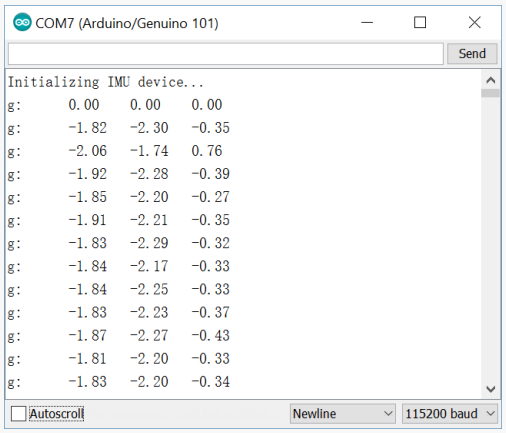
该函数把直接测量值转化为单位为度/s 的可读数据。

附加函数

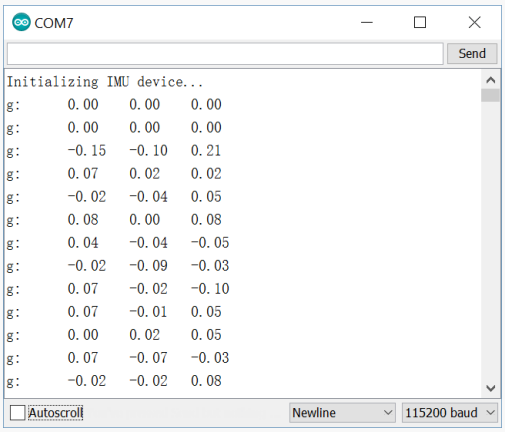
函数库同样设有陀螺仪的校准函数。与加速计不同的是，101 开发板校正过程中可以以任意方向静置，并且不需要额外定义参照方向。在“CurieIMU.begin();”后添加以下代码即可完成校准。

```
CurieIMU.autoCalibrateGyroOffset();
```

校准后的数据应该如下所示：



校准前测量结果



校准后测量结果

姿态控制舵机

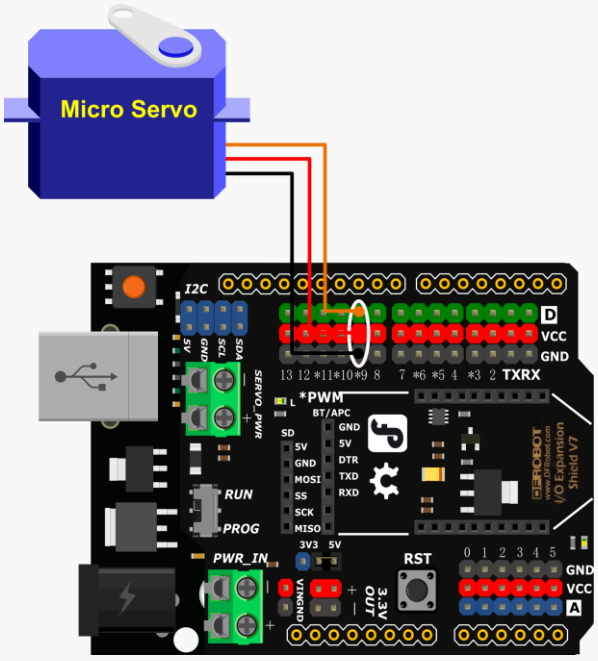
在这个步骤中我们将会学习通过 101 的空间姿态来控制舵机。当 101 向左倾斜时，舵机会转至一个方向；而向右倾斜时舵机会转至另一个方向。

所需元件

1 × TowerPro SG50 舵机

硬件连接

把 TowerPro SG50 舵机接在数字信号引脚 9 号口上。



使用原理

当 101 开发板只在重力作用下（没有其余加速度），轴上测得的加速度只会在 -g 和 g 之间。轴偏离水平面越多，重力的分量在该轴上就越大。因此，通过把轴上的测量结果对应到舵机的角度位置，就可以很容易地根据板子姿态控制舵机。

输入代码

```
#include "CurieIMU.h"
#include <Servo.h>
int pos;
Servo myservo;  // create servo object to control a servo

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  // initialize device
  CurieIMU.begin();
    CurieIMU.autoCalibrateAccelerometerOffset(X_AXIS, 0);
    CurieIMU.autoCalibrateAccelerometerOffset(Y_AXIS, 0);
    CurieIMU.autoCalibrateAccelerometerOffset(Z_AXIS, 1);
  // Set the accelerometer range to 2G
  CurieIMU.setAccelerometerRange(2);
}

void loop() {
  int axRaw, ayRaw, azRaw;         // raw accelerometer values
  float ax, ay, az;
  // read raw accelerometer measurements from device
  CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);

  // convert the raw accelerometer data to G's
  ax = convertRawAcceleration(axRaw);
  ay = convertRawAcceleration(ayRaw);
  az = convertRawAcceleration(azRaw);

   pos = 90+ax*90;
   constrain (pos,0,180);
   myservo.write(pos);
}

float convertRawAcceleration(int aRaw) {
  // since we are using 2G range
  // -2g maps to a raw value of -32768
  // +2g maps to a raw value of 32767

  float a = (aRaw * 2.0) / 32768.0;
  return a;
}
```

代码分析

```
#include <Servo.h>
```
在驱动舵机之前，我们需要先加载函数库“Servo.h”，而该函数库已经包含在 Arduino IDE 预置库中。

```
Servo myservo;
```
定义 “myservo” 为舵机类型对象，从而使用“Servo.h”中的函数库。

```
myservo.attach(9);
```
定义舵机接在哪一个引脚上（本例中为 D9）。

```
pos = 90+ax*90;
```
把变量“ax”（数据范围为-1 到 1）的数值映射到变量“pos”（数据范围为 0 到 360）。

```
constrain (pos,0,180);
```
限制 pos 范围为 0 到 180，以免超出范围。

```
write(pos);
```
使舵机转动到某个角度。你可以用 0 到 180 之间的任意整数，这个数字分别对应着舵机在 0 度到 180 度之间的角度位置。

项目十一 电子水平仪

在本章节中，学习如何把加速度计所测得的数据通过串口导入至 Processing 图形交互软件，从而画出一个可以显示 101 开发板实时姿态的 3D 模型。

搭建电子水平仪

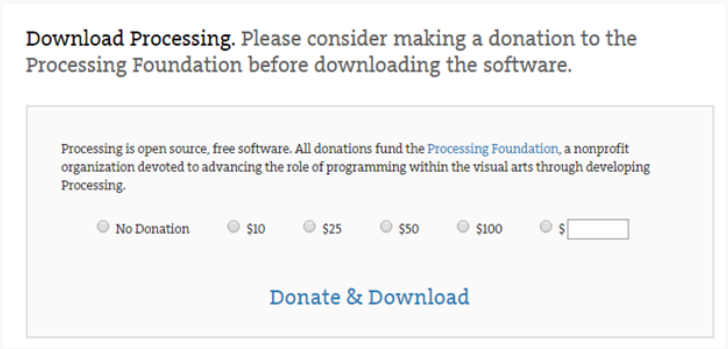
本章节中，我们将介绍一款可以将实时数据转化为 3D 视图的有线通信软件“Processing”。由于篇幅限制，我们将只介绍如何输入数据并画图。当然，我们也建议对可视化交互项目感兴趣的读者深入学习这款软件。

下载 Processing

点击下面链接根据引导下载。

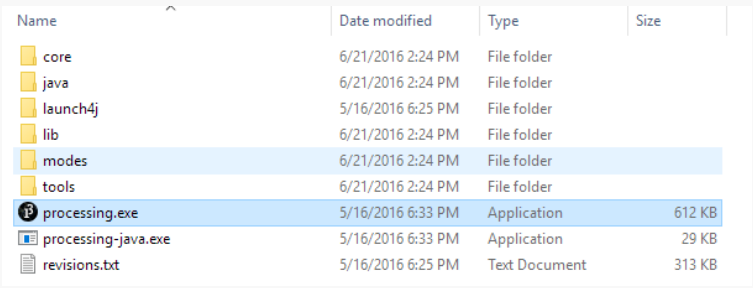
https://processing.org/download/

与 Arduino IDE 一样，Processing 是一款免费的开源软件，你可以点击“捐助并下载”捐助任意金额进入下载界面。下载适用于你操作系统的版本，在下面的教程中将使用 64 位 Windows 作示范。



STEP 1 下载 Processing

STEP 2 解压软件



STEP 3 打开 Processing

输入代码

开始前，我们要给 101 上传程序使它持续给串口发送姿态数据。并且，我们需要给 Processing 编程来接收数据并转化为 3D 图形。

Arduino IDE 代码：

```
#include "CurieIMU.h"
#include "math.h"
int16_t ax, ay, az;         // accelerometer values
int16_t gx, gy, gz;         // gyrometer values

const int ledPin = 13;      // activity LED pin
boolean blinkState = false; // state of the LED

void setup() {
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open

  // initialize device
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();

  // verify connection
  Serial.println("Testing device connections...");
  if (CurieIMU.testConnection()) {
    Serial.println("CurieIMU connection successful");
  } else {
    Serial.println("CurieIMU connection failed");
  }

  // use the code below to calibrate accel/gyro offset values
  Serial.println("Internal sensor offsets BEFORE calibration...");
  Serial.print(CurieIMU.getXAccelOffset());
  Serial.print("\t"); // -76
  Serial.print(CurieIMU.getYAccelOffset());
  Serial.print("\t"); // -235
  Serial.print(CurieIMU.getZAccelOffset());
  Serial.print("\t"); // 168
  Serial.print(CurieIMU.getXGyroOffset());
  Serial.print("\t"); // 0
  Serial.print(CurieIMU.getYGyroOffset());
  Serial.print("\t"); // 0
  Serial.println(CurieIMU.getZGyroOffset());
```

```
  Serial.println("About to calibrate. Make sure your board is stable and
upright");
  delay(5000);

  // The board must be resting in a horizontal position for
  // the following calibration procedure to work correctly!
  Serial.print("Starting Gyroscope calibration...");
  CurieIMU.autoCalibrateGyroOffset();
  Serial.println(" Done");
  Serial.print("Starting Acceleration calibration...");
  CurieIMU.autoCalibrateXAccelOffset(0);
  CurieIMU.autoCalibrateYAccelOffset(0);
  CurieIMU.autoCalibrateZAccelOffset(1);
  Serial.println(" Done");

  Serial.println("Internal sensor offsets AFTER calibration...");
  Serial.print(CurieIMU.getXAccelOffset());
  Serial.print("\t"); // -76
  Serial.print(CurieIMU.getYAccelOffset());
  Serial.print("\t"); // -2359
  Serial.print(CurieIMU.getZAccelOffset());
  Serial.print("\t"); // 1688
  Serial.print(CurieIMU.getXGyroOffset());
  Serial.print("\t"); // 0
  Serial.print(CurieIMU.getYGyroOffset());
  Serial.print("\t"); // 0
  Serial.println(CurieIMU.getZGyroOffset());

  Serial.println("Enabling Gyroscope/Acceleration offset compensation");
  CurieIMU.setGyroOffsetEnabled(true);
  CurieIMU.setAccelOffsetEnabled(true);

  // configure Arduino LED for activity indicator
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read raw accel/gyro measurements from device
  CurieIMU.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  double unitx, unity;
  unitx= double(ax)/sqrt(ax*ax+ay*ay+az*az);
  unity= double(ay)/sqrt(ax*ax+ay*ay+az*az);
  double pitch, yaw,roll;

  yaw = 0;
  pitch = -asin(unity/sqrt(1-unitx*unitx)) ;
  roll = atan(unitx/sqrt(1-unitx*unitx));
```

```
  Serial.print(float(yaw));
  Serial.print(","); // print comma so values can be parsed
  Serial.print(float(pitch));
  Serial.print(","); // print comma so values can be parsed
  Serial.println(float(roll));
  delay (10);

}
```

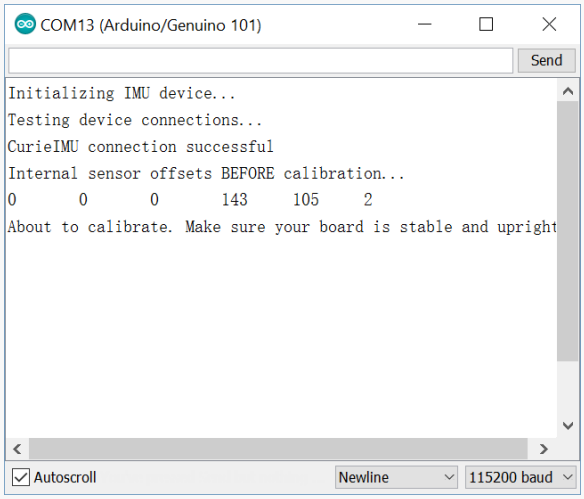
Processing 代码：

```
import processing.serial.*;
Serial myPort;

int newLine = 13; // new line character in ASCII
float yaw;
float pitch;
float roll;
String message;
String [] ypr = new String [3];

void setup()
{
  size(600, 500, P3D);

  /*Set my serial port to same as Arduino, baud rate 9600*/
  myPort = new Serial(this, Serial.list()[0], 9600); // if you have only ONE
COM port active
  //myPort = new Serial(this, "COM5", 9600);  // if you know the 101 COM port

  textSize(16); // set text size
  textMode(SHAPE); // set text mode to shape
}

void draw()
{
  serialEvent();  // read and parse incoming serial message
  background(255); // set background to white

  translate(width/2, height/2); // set position to centre

  pushMatrix(); // begin object

  rotateX(pitch); // RotateX pitch value
  rotateY(-yaw); // yaw
  rotateZ(-roll); // roll
```

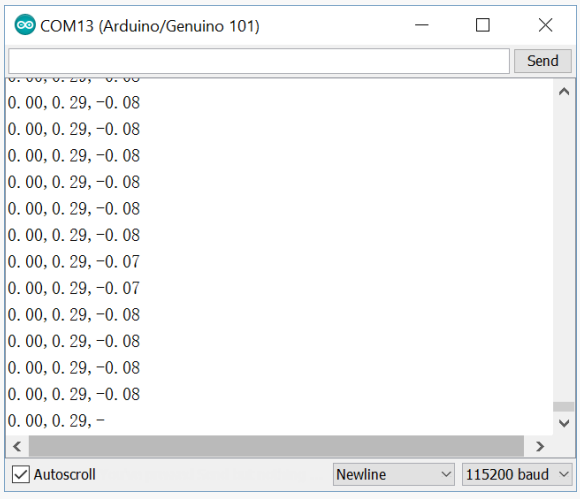
```
  drawArduino(); // function to draw rough Arduino shape

  popMatrix(); // end of object

  // Print values to console
  print(pitch);
  print("\t");
  print(roll);
  print("\t");
  print(-yaw);
  println("\t");

}

void serialEvent()
{
  message = myPort.readStringUntil(newLine); // read from port until new line
(ASCII code 13)
  if (message != null) {
    ypr = split(message, ","); // split message by commas and store in String
array
    yaw = float(ypr[0]); // convert to float yaw
    pitch = float(ypr[1]); // convert to float pitch
    roll = float(ypr[2]); // convert to float roll
  }
}
void drawArduino() {
  /* function contains shape(s) that are rotated with the IMU */
  stroke(0, 90, 90); // set outline colour to darker teal
  fill(0, 130, 130); // set fill colour to lighter teal
  box(300, 10, 200); // draw Arduino board base shape

  stroke(0); // set outline colour to black
  fill(80); // set fill colour to dark grey

  translate(60, -10, 90); // set position to edge of Arduino box
  box(170, 20, 10); // draw pin header as box

  translate(-20, 0, -180); // set position to other edge of Arduino box
  box(210, 20, 10); // draw other pin header as box
}
```

指导步骤

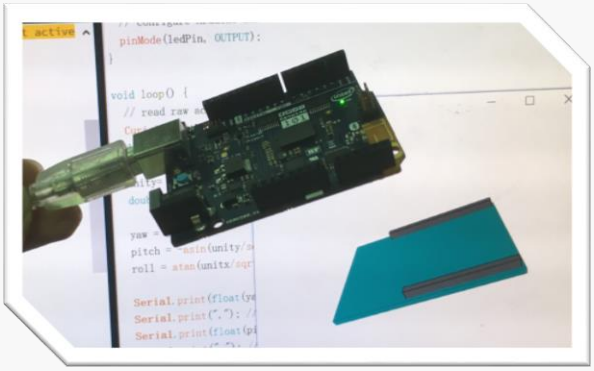
在 Arduino IDE 将程序上传到 101 开发板中，等待几秒钟直至系统完全启动。选择 101 板子的端口打开串口监视器。串口监视器第一次打开时，101 会首先运行校准程序。确保板子平放在水平面上，整个过程大概会持续 10 秒。



校准完成后，101 开发板会向串口发送姿态信息，每一行分别代表偏航角（yaw）、俯仰角（pitch）和翻滚角（roll），之间以逗号隔开。因为我们将只使用加速计的测量数据，我们无法知道偏航角（板子的朝向，即 Z 轴转动角度）的大小，只能知道俯仰角和翻滚角（X，Y 轴的转动角度，可以知道板子朝哪个方向偏和偏离角度大小）。



检查串口监视器可以接收到数据后，关闭串口监视器并运行复制在 Processing 中的程序（记得首先关掉一个串口监视器才能打开另一个，不然会报错），此时一个 101 开发板的 3D 模型会出现在弹出的窗口中。现在，试试倾斜你的 101 开发板，看看 3D 模型会不会跟着转吧。



希望你的 Arduino 之旅不会因此而停止，用你的奇思妙想，玩出更多新颖有创意的作品。如果你愿意与我们分享的话，也可以直接登陆我们的论坛，让我们的社区论坛记录下你的点点滴滴！

欢迎登陆 DFRobot 创客社区!

DFRobot 创客社区: www.dfrobot.com.cn