

前奏1

# 初识Arduino

---

01



**DFROBOT**  
DRIVE THE FUTURE

# Arduino是什么？

Arduino是一个开放源码电子原型平台，拥有灵活、易用的硬件和软件。Arduino专为设计师，工艺美术人员，业余爱好者，以及对开发互动装置或互动式开发环境感兴趣的人而设的。

Arduino可以接收来自各种传感器的输入信号从而检测出运行环境，并通过控制光源，电机以及其他驱动器来影响其周围环境。板上的微控制器编程使用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。要给Arduino下一个最准确的定义，最好用一些实例来描述。

- ◆ 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都可以为您实现。

# Arduino诞生啦！

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就写出了程式码。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，比如对感测器作出回应，闪烁灯光，还能控制马达。

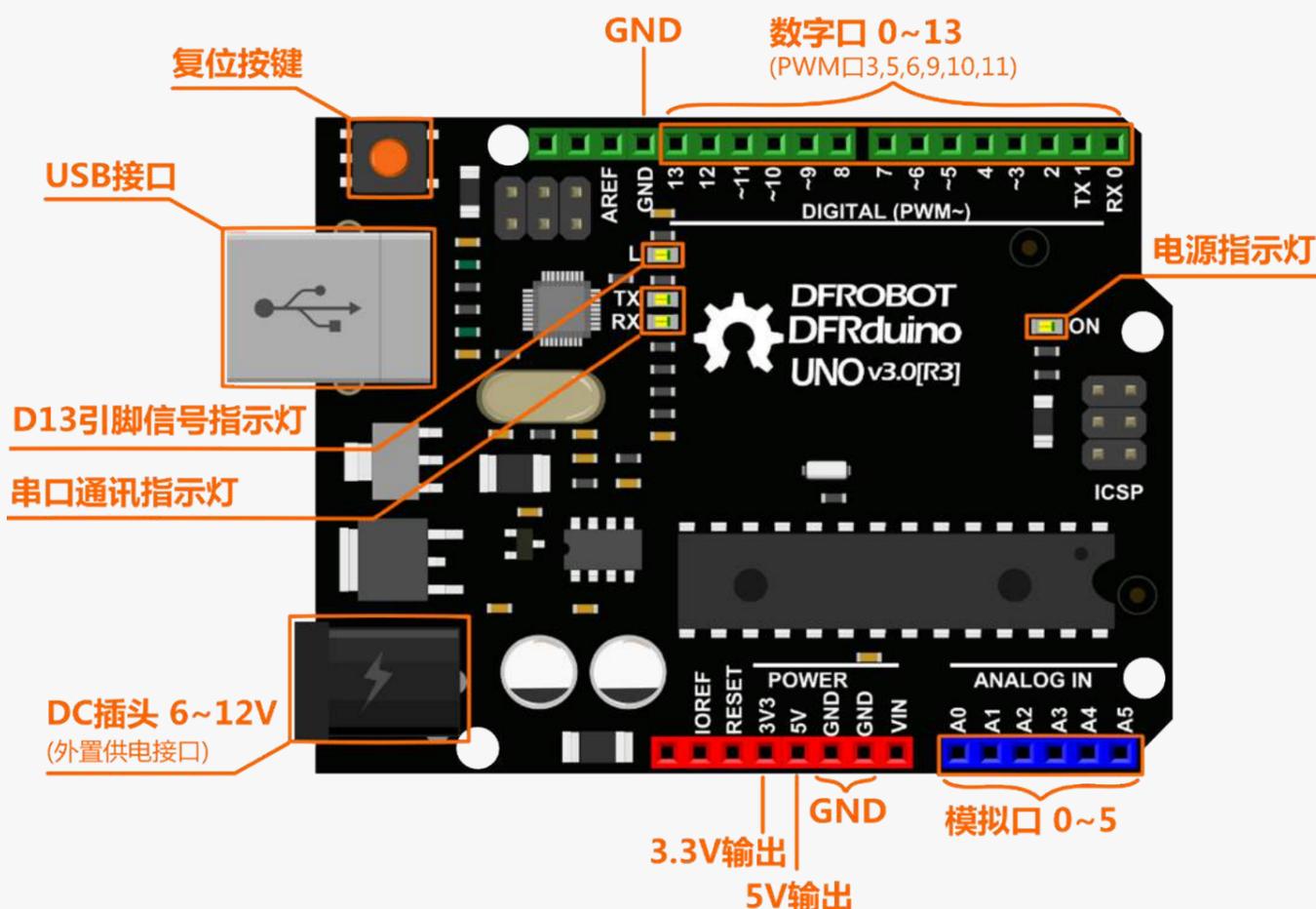
## Arduino名称由来

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Arduin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。

## 认识Arduino UNO

先来简单的看下Arduino UNO。下图中有标识的部分为常用部分。图中标出的数字口和模拟口，即为常说的I/O。数字口有0~13，模拟口有0~5。

除了最重要的I/O口外，还有电源部分。UNO可以通过两种方式供电方式，一种通过USB供电，另一种是通过外接6~12V的DC电源。除此之外，还有4个LED灯和复位按键，稍微说下4个LED。ON是电源指示灯，通电就会亮了。L是接在数字口13上的一个LED，在下面一节会有个样例来说明的。TX、RX是串口通讯指示灯，比如我们在下载程序的过程中，这两个灯就会不停闪烁。



# 初次使用

## 1. 下载Arduino IDE

打开网页输入网址

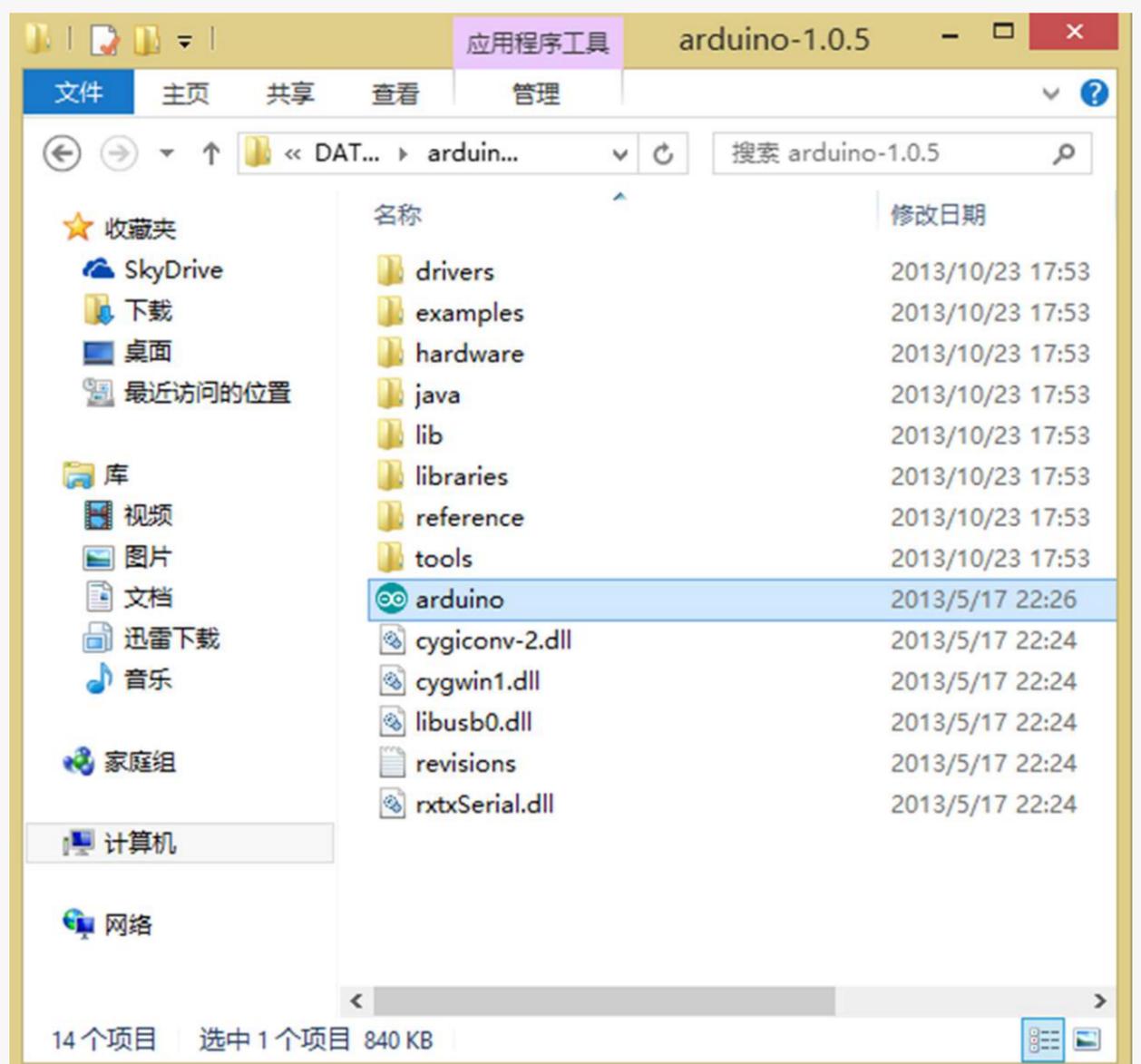
<http://arduino.cc/en/Main/Software>

进入到页面后，找到下图显示部分。



Windows用户，点击下载 [Windows\(ZIP file\)](#)，如果Mac，Linux用户则选择相应的系统。

下载完成后，解压文件，把整个Arduino 1.0.5文件夹放到你电脑熟悉的位置，便于你之后查找。打开Arduino 1.0.5文件夹，就是下图的看到内容。

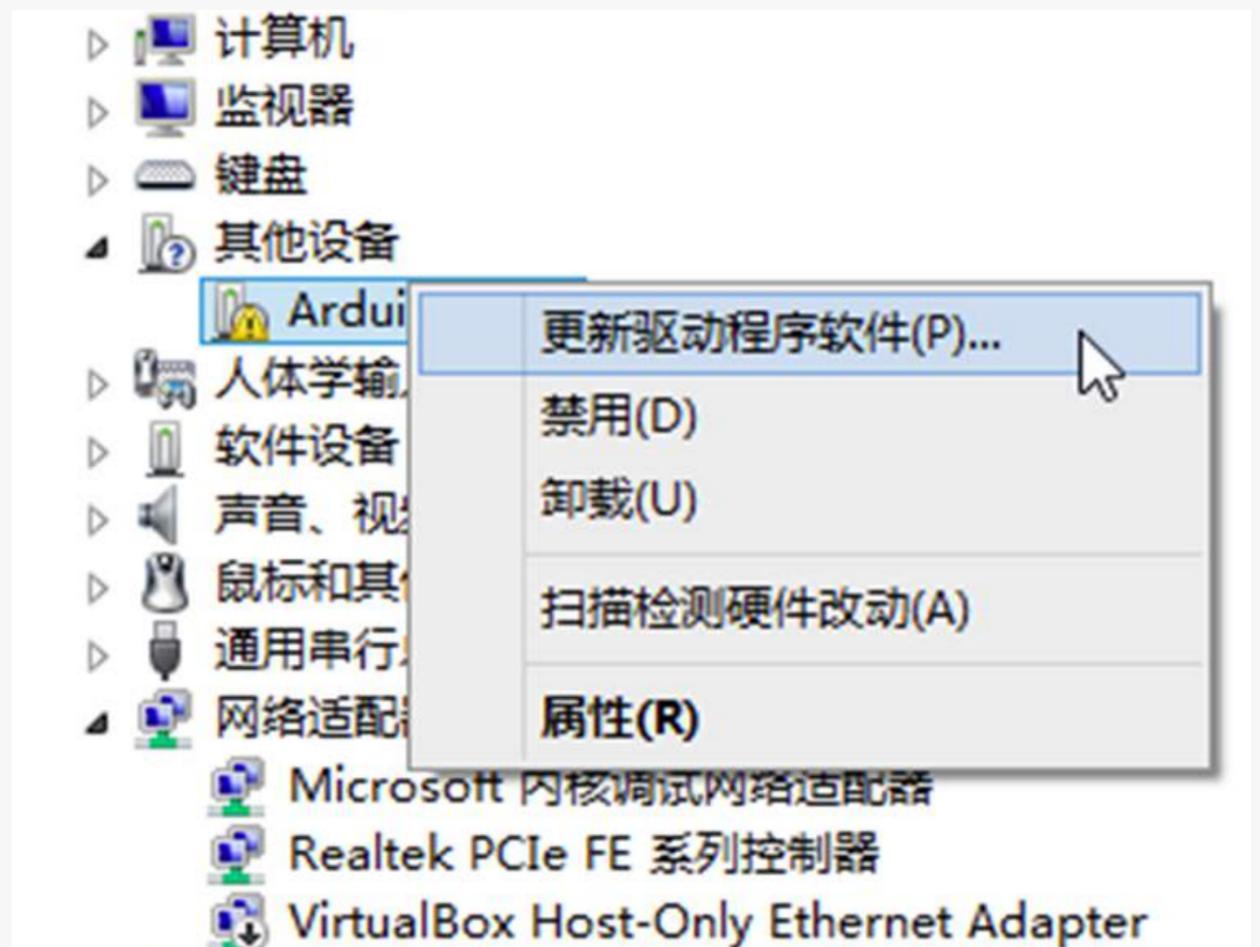


## 2. 安装驱动

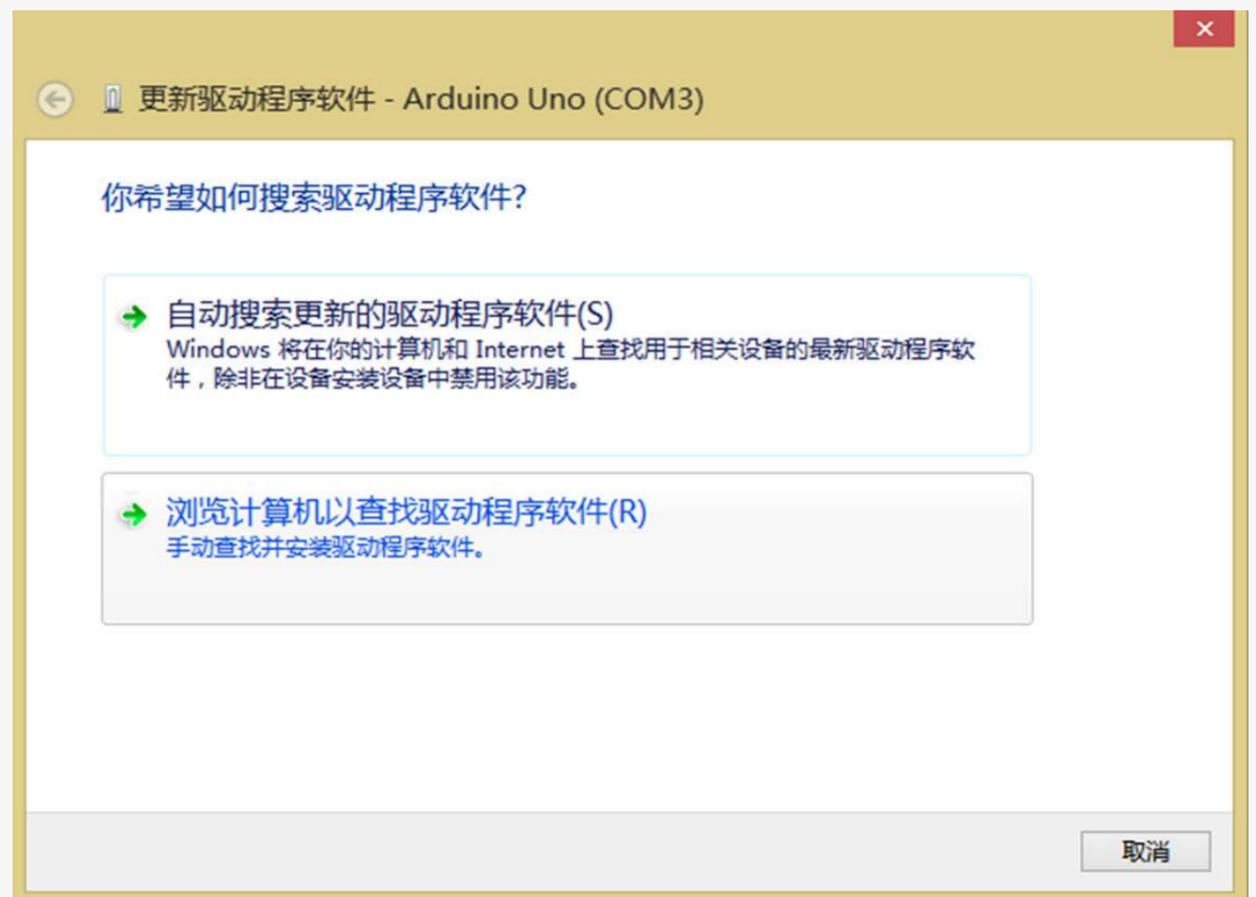
把USB一端插到Arduino UNO上，另一端连到电脑。连接成功后，UNO板的红色电源指示灯ON亮起。然后，打开控制面板，选择设备管理器。



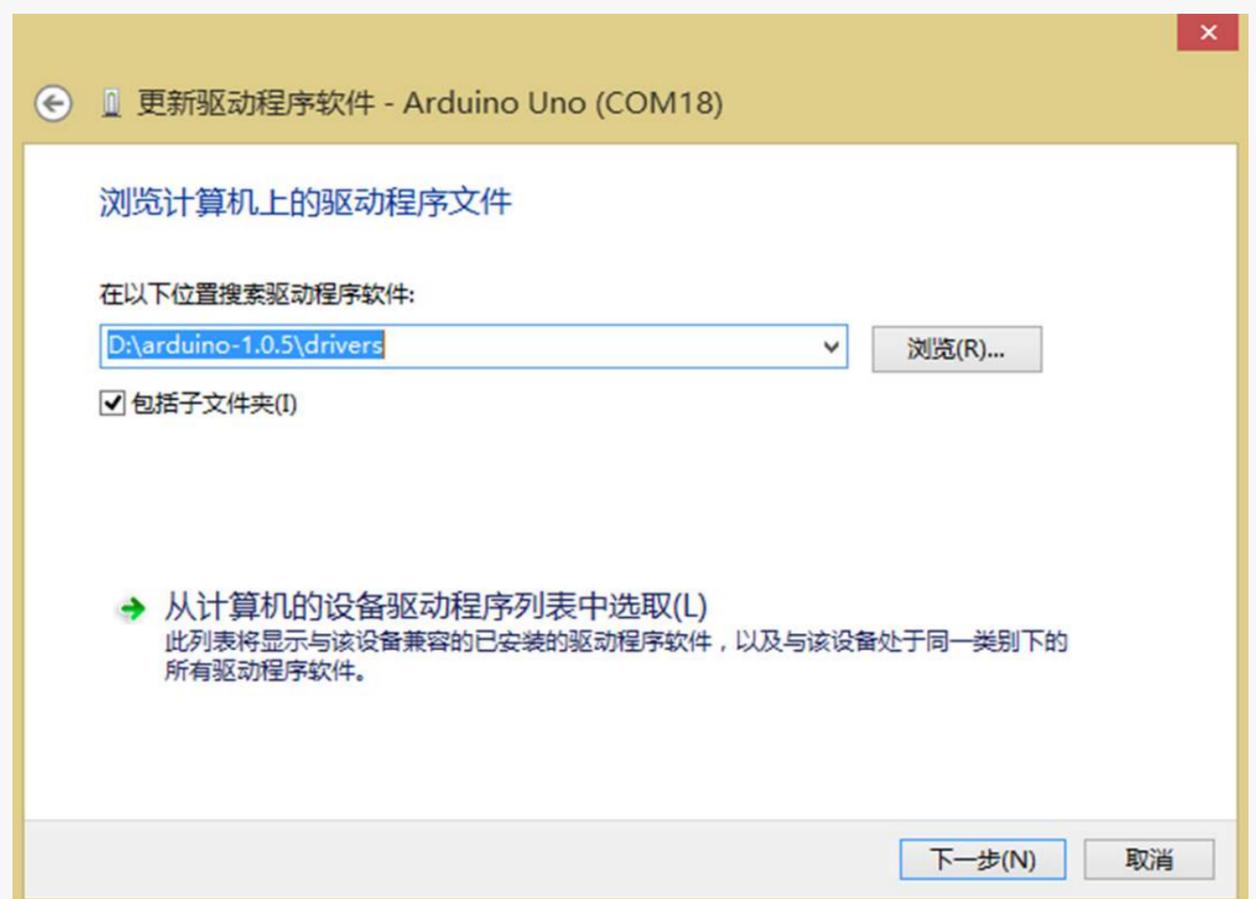
找到其它设备>Arduino-xx，右击选择更新驱动程序软件。



在弹出的对话框中选择下面一项  
--> 手动查找并安装驱动程序软件。



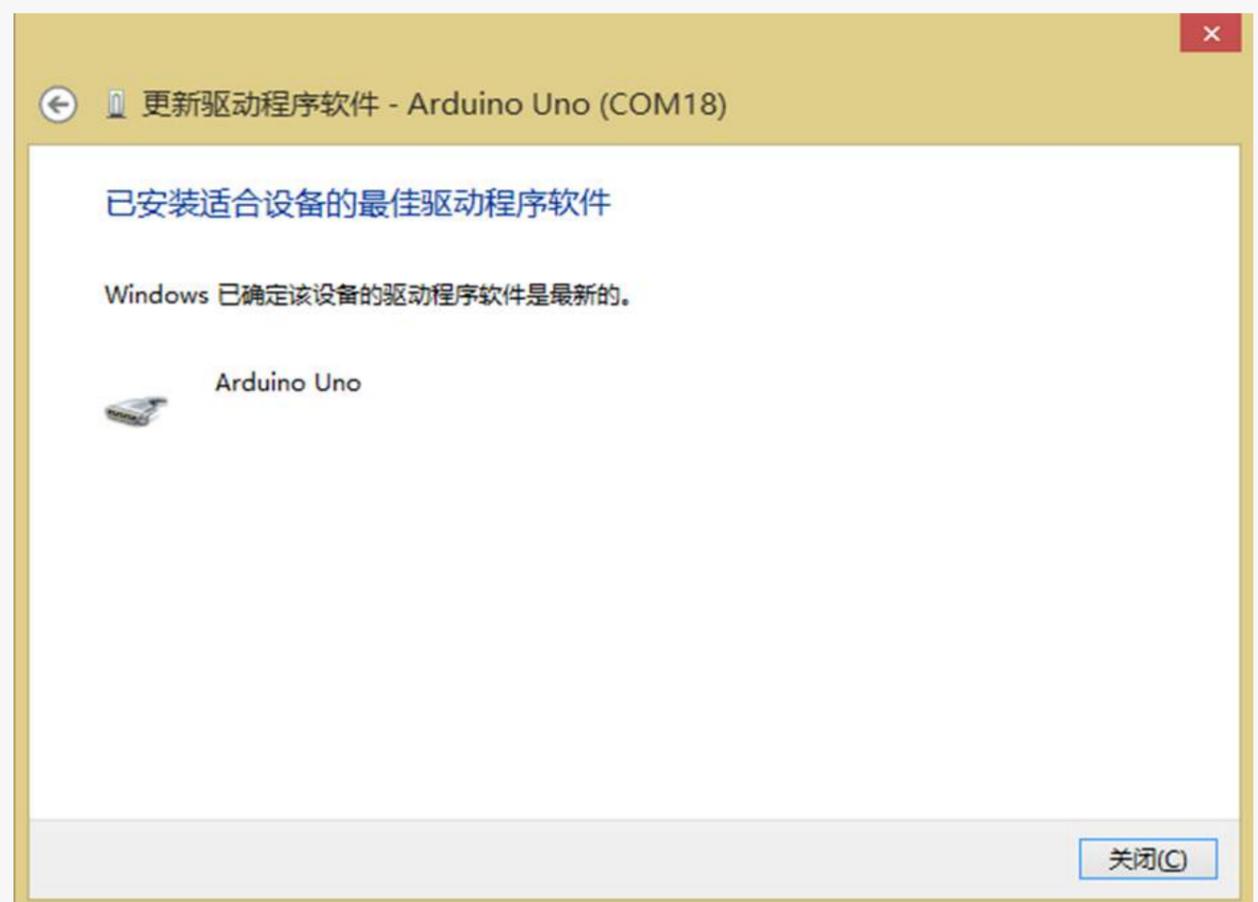
打开到Arduino IDE安装位置，就是上面那个解压文件的位置，选择搜索路径到drivers，点击下一步。



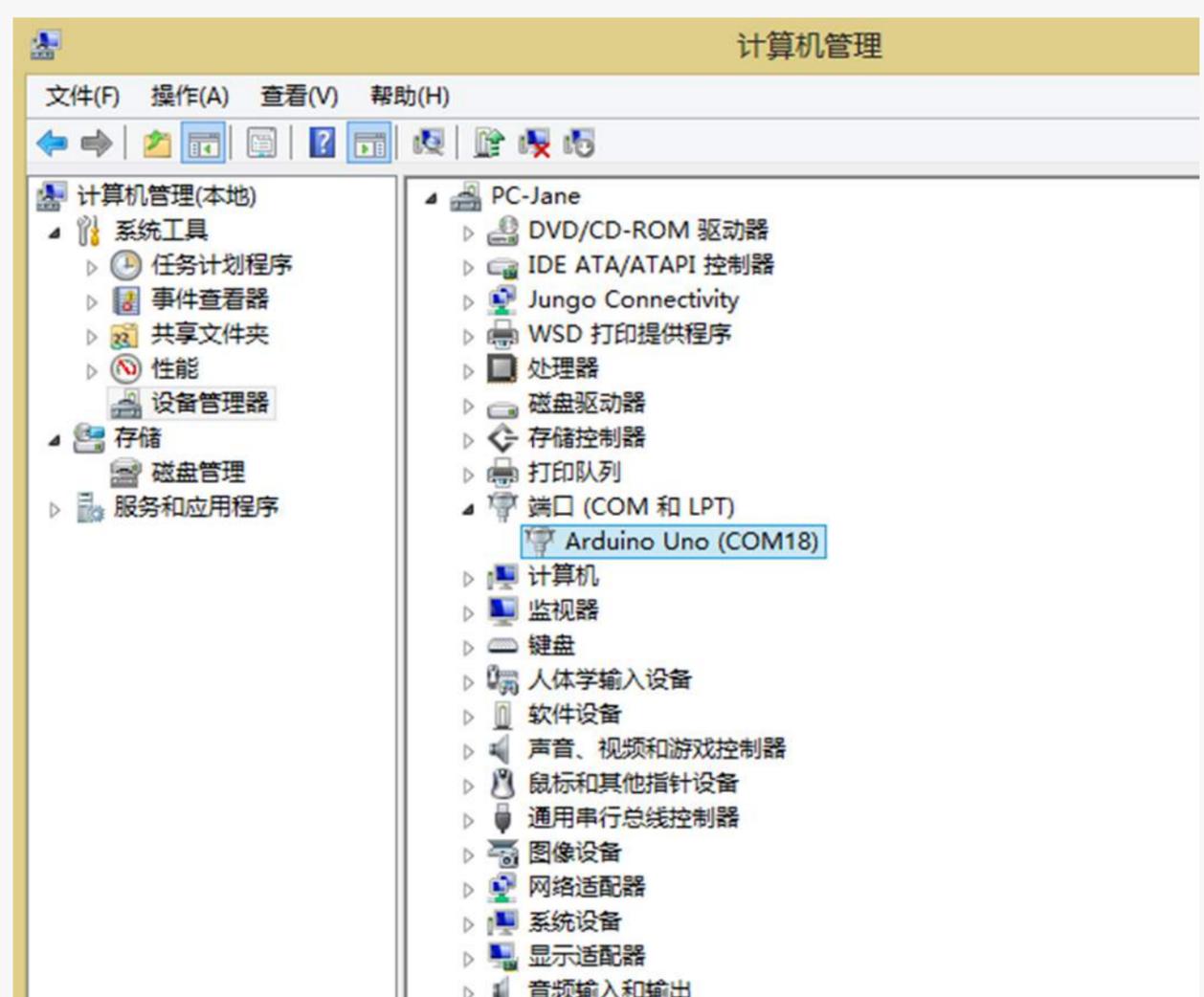
选择始终安装此驱动程序软件，直至完成。



出现下图，说明驱动安装成功。

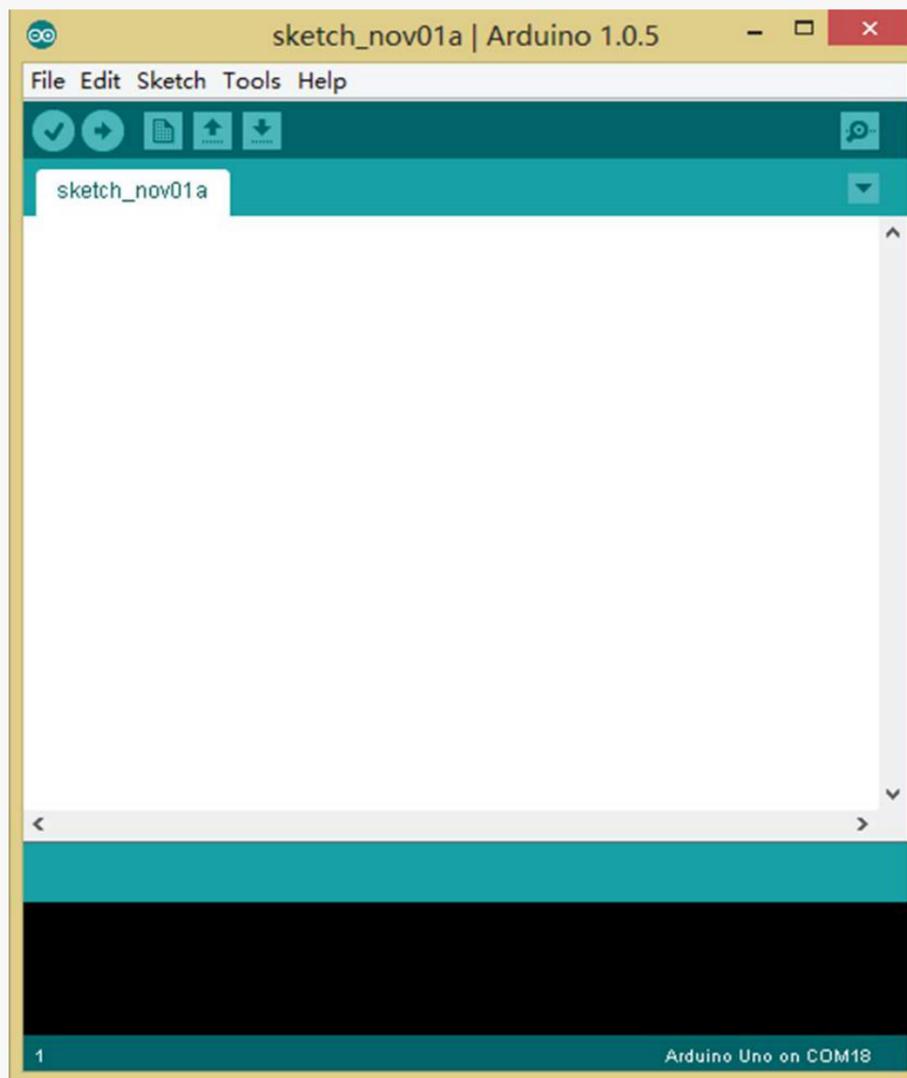


此时，设备管理器端口会显示一个串口号。

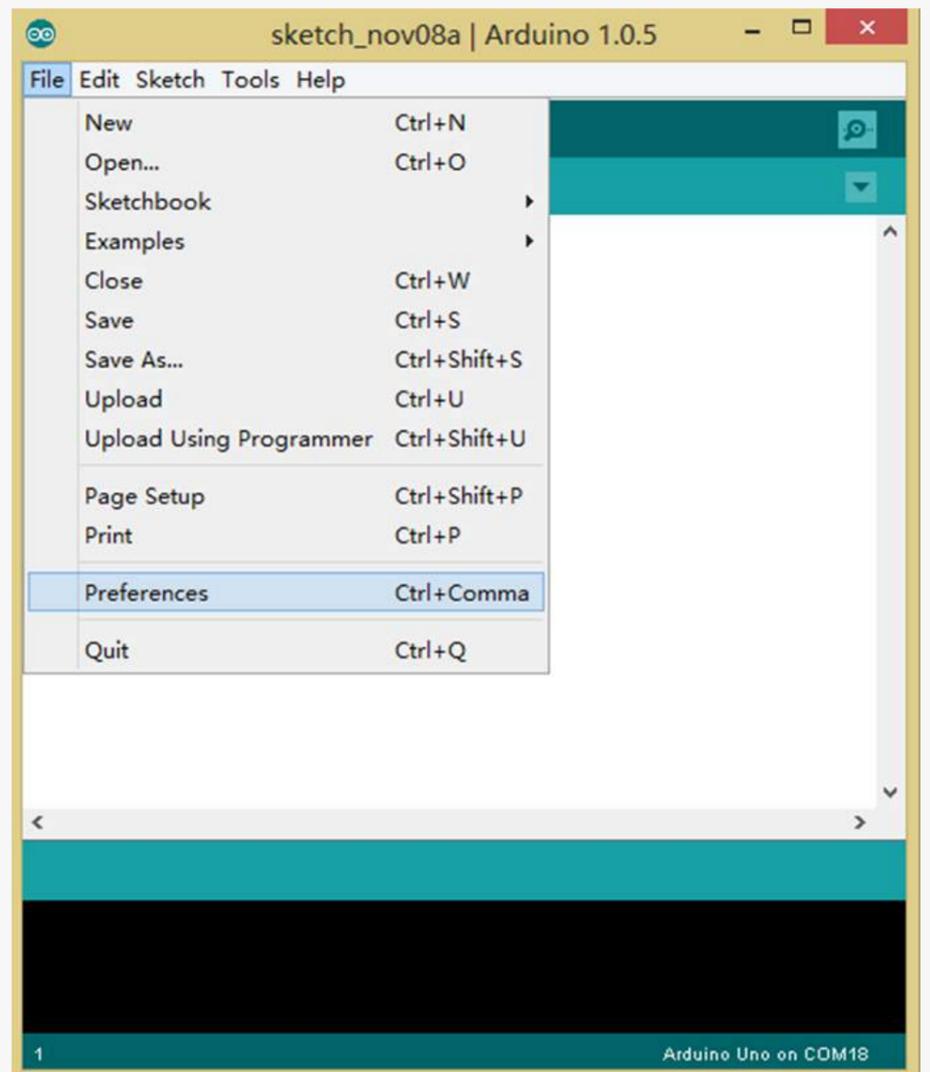


### 3. 认识Arduino IDE

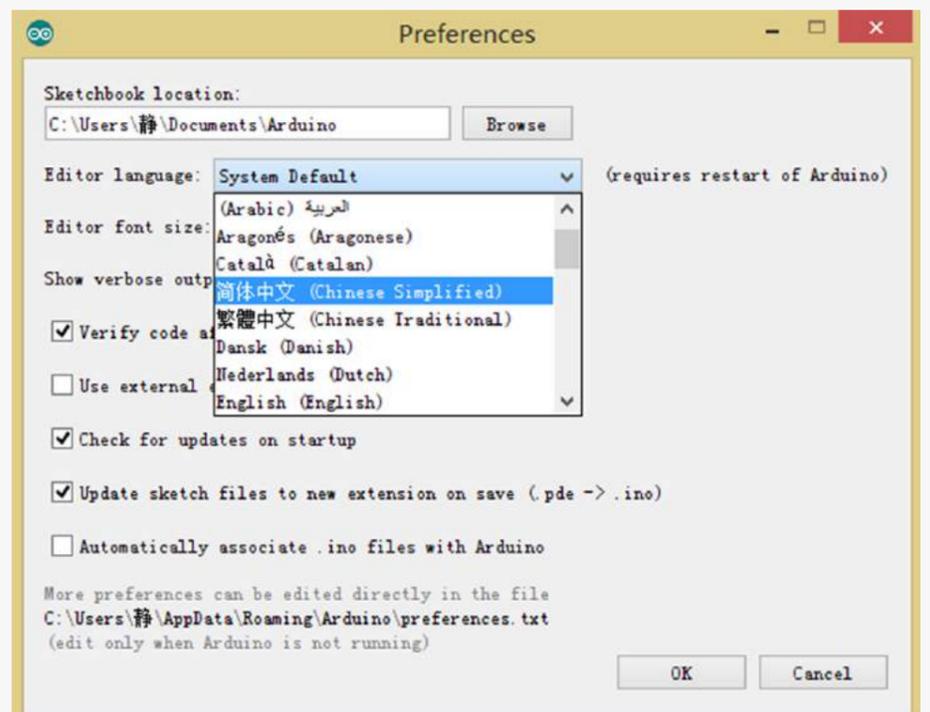
打开Arduino IDE，就会出现Arduino IDE的编辑界面。



如果英文界面，你不太习惯的话，可以先更改为中文界面。选择菜单栏File → Preferences。

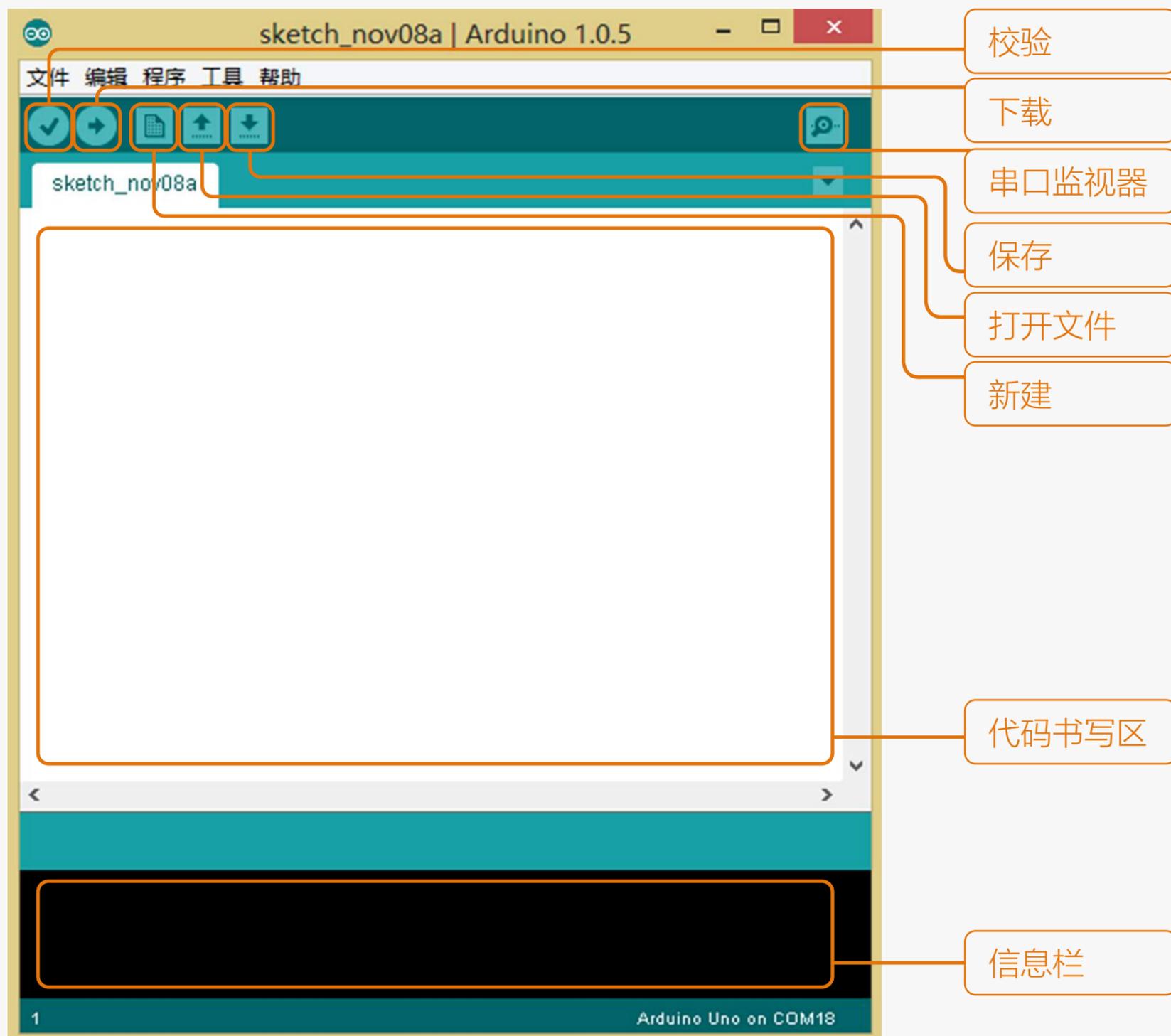


会跳出下面这个对话框，选择Editor language → 简体中文，点击OK。



关闭Arduino IDE，重新打开，就是中文界面了！

先简单认识看一下Arduino的这个编译器，以后可是要经常和它打交道的。

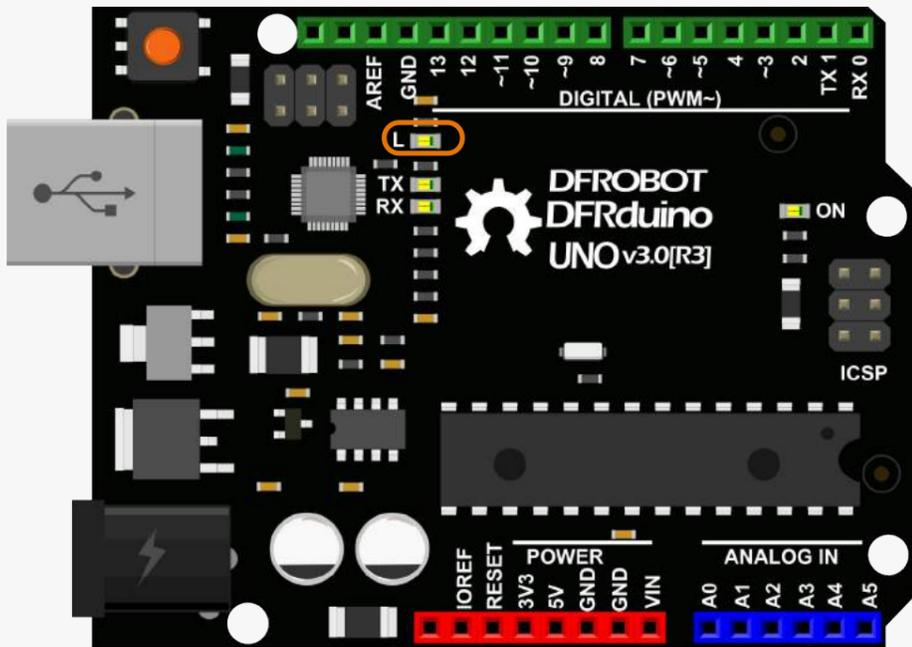


Arduino IDE是Arduino产品的软件编辑环境。简单的说就是用来写代码，下载代码的地方。任何的Arduino产品都需要下载代码后才能运作。我们所搭建的硬件电路是辅助代码来完成的，两者是缺一不可的。如同人通过大脑来控制肢体活动是一个道理。如果代码就是大脑的话，外围硬件就是肢体，肢体的活动取决于大脑，所以硬件实现取决于代码。

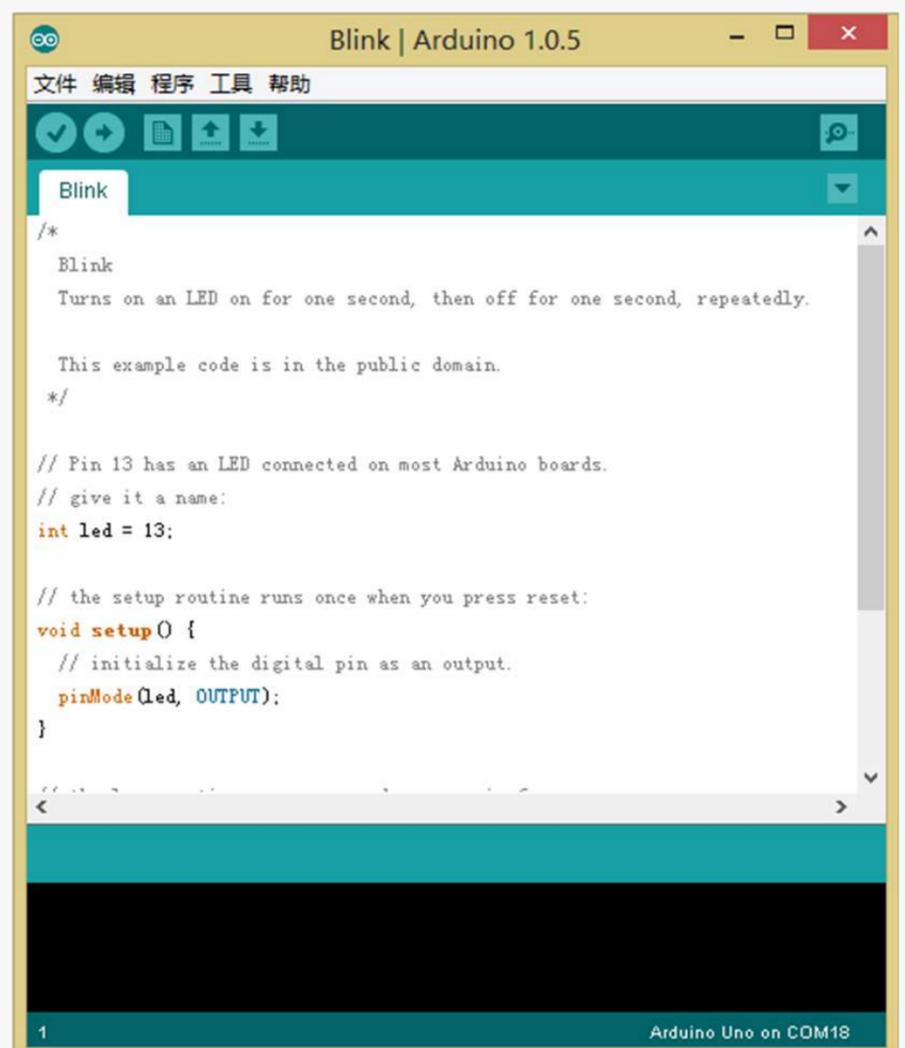
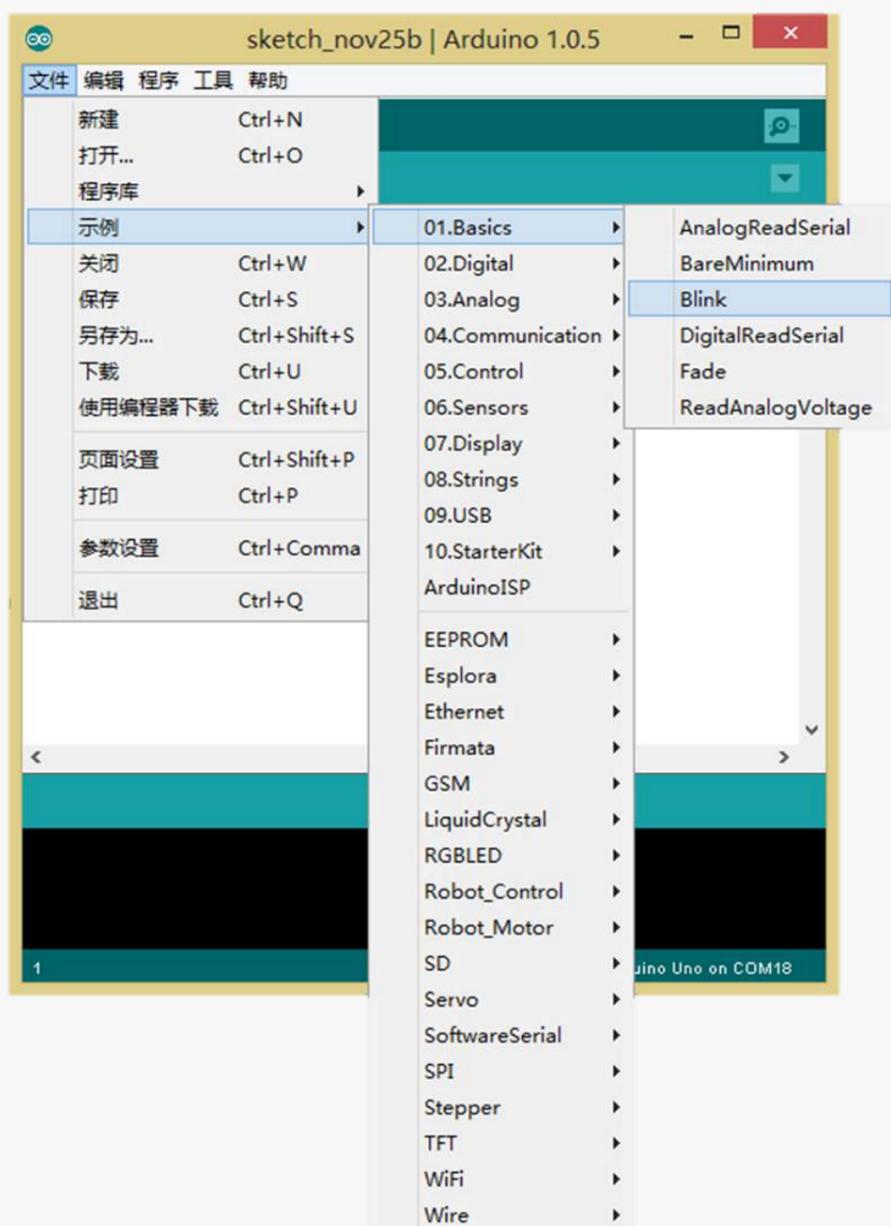
Arduino IDE基本也只需要用到上面标示出来的部分就可以了，上图大部分的白色区域就是代码的编辑区，用来输入代码的。注意，输入代码时，要切换到英文输入法的模式。下面黑色的区域是消息提示区，会显示编译或者下载是否通过。

## 4. 下载一个Blink程序

下载一个最简单的代码，既可以帮你熟悉如何下载程序，同时也测试下板子好坏。UNO板上标有L的LED。这段测试代码就是让这个LED灯闪烁。



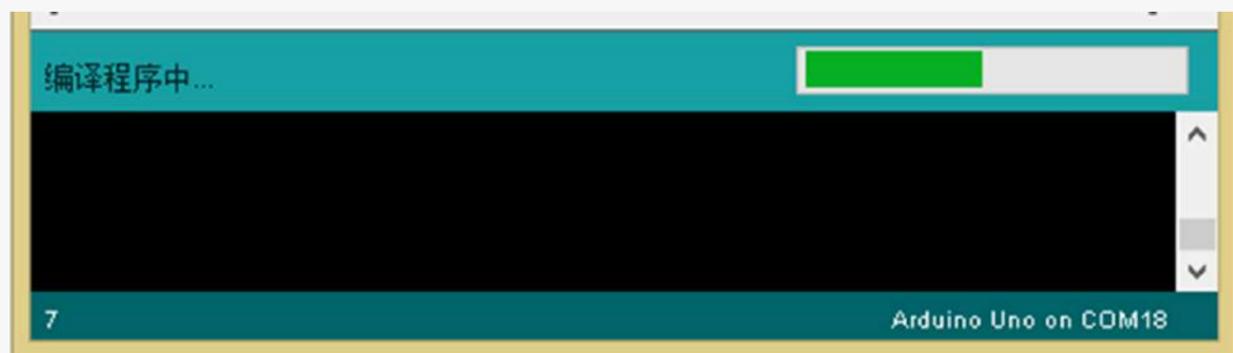
插上USB线，打开Arduino IDE后，找到“Blink”代码。



通常，写完一段代码后，我们都需要校验一下，看看代码有没有错误。点击“校验”。



下图显示了正在校验中。



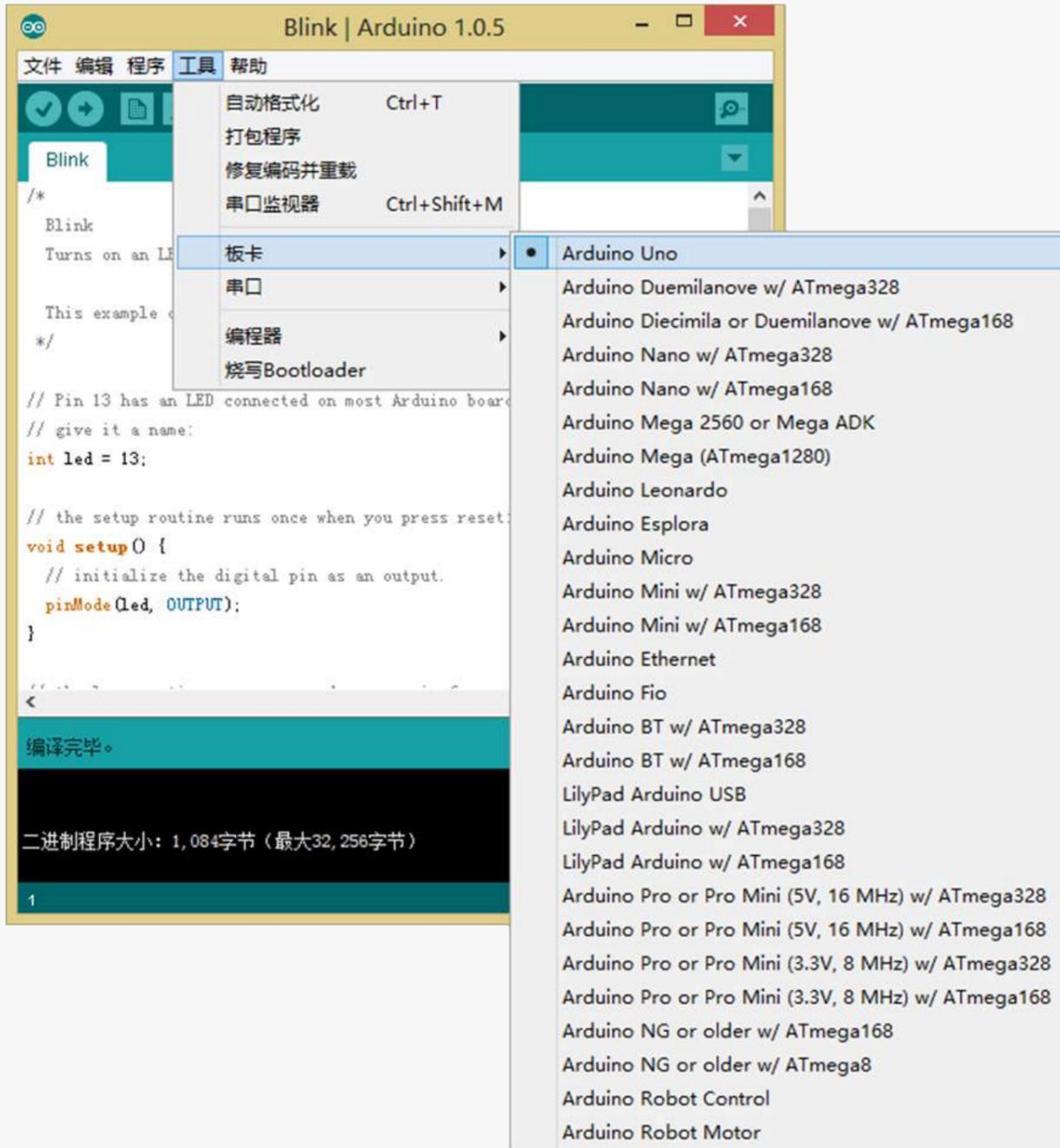
校验完毕!



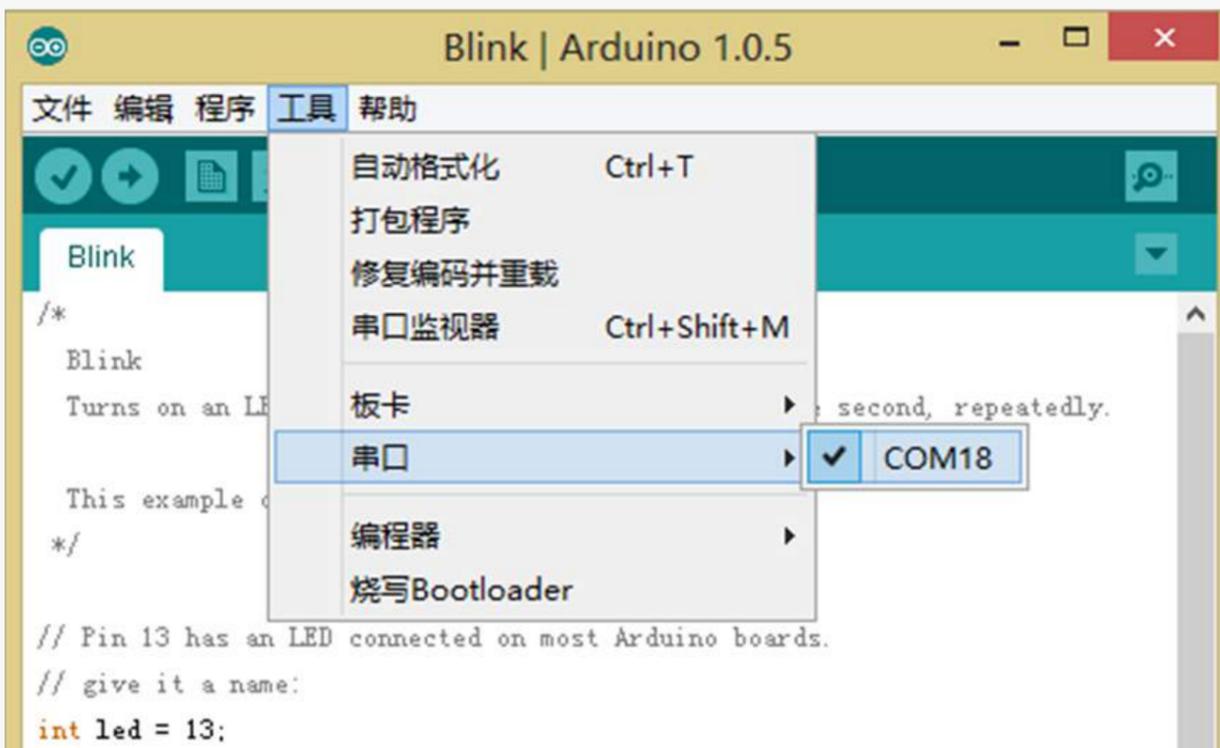
由于是样例代码，所以校验不会有错误，不过在以后写代码的过程中，输入完代码，都需要校验一下，然后再下载到Arduino中。

在下载程序之前，我们还要先告诉Arduino IDE板子型号以及相应的串口。

选择所用的板卡Board → Arduino UNO。



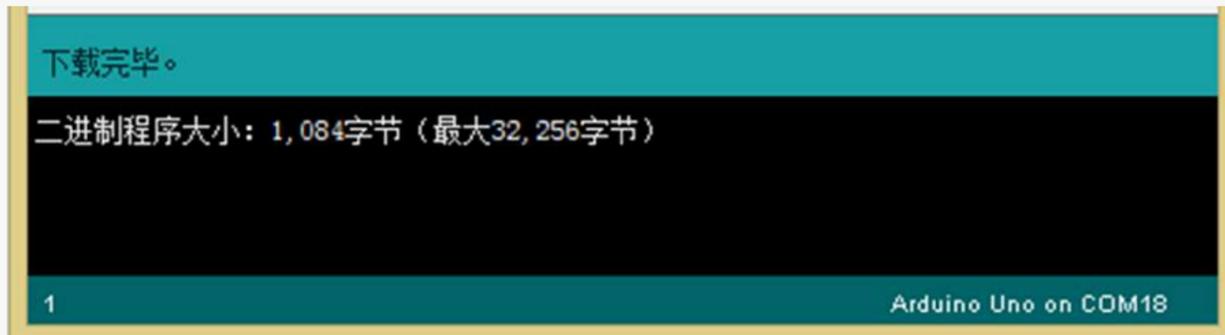
选择当前的串口——COM口。



最后，点击“下载”。



下载完毕!



以上就是给Arduino下载程序一个blink程序的整个过程。

以后程序下载就照着这个步骤做就可以了，再理一下思路，分为三步走：

校验 → 选择boards和com → 下载！

前奏2

# 是什么让 东西“活”起来了

---

02



**DFROBOT**  
DRIVE THE FUTURE

# 简单的自动控制装置 需要具备哪些元素

我们用Arduino做的小制作都可以称为是一个简单的自动控制装置。一个简单自动控制的装置，通常会有三个元素，输入、控制和输出。输入设备来搜集信号，控制器对接收到的信号进行处理、最后再由输出设备输出信号。我们以人来说，五感就是输入信号，把信号送到大脑，大脑再做出反应，输出的就是人的行为。



而在Arduino的世界里也同样有输入、控制与输出。Arduino的五感是通过各式各样的传感器来实现的。Arduino控制器好比是人的大脑，来反应和处理信号。最后输出主要有声、光（Led）、动（直流电机、舵机）等表现形式。

做个简单的比喻吧！有个人叫你，你随即就回答：“听到了”。这里，你的耳朵就是输入设备，你的大脑就是控制设备，嘴巴就是你的输出设备。那整个过程我们如何通过Arduino来实现呢？

最简单的，通过一个声音传感器，一听到有声音，Arduino就会接受到一个信号，然后，Arduino就让蜂鸣器“吱”一声表示回答。来分析下，这里，声音传感器就是输入设备，Arduino就是控制设备，最后蜂鸣器就是输出设备。

## 思考

能否识别出套件中哪些可做输入设备，哪些可做输出设备？

## 输入设备-传感器

传感器是一种物理装置或生物器官，能够探测、感受外界的信号、物理条件（如光、热、湿度）或化学组成（如烟雾），并将探知的信息传递给其他装置或器官。传感器的作用是将一种能量转换成另一种能量形式，所以不少学者也用“换能器 - Transducer”来称谓“传感器 - Sensor”。

### 传感器的接口

传感器接口分为三种，先了解下，分为：

- 数字接口
- 模拟接口
- 协议接口（数字）

协议接口也是数字接口的一种，常用的有I2C，Serial，SPI。

## 控制设备-Arduino

不用多说，控制设备就是Arduino的控制器。我们这里选用的是Arduino UNO。前面说了控制器好比人的大脑的作用，用来处理事情。

## 输出设备 - 执行器

执行器也有很多种，最常见的是“动”。好比人的动作，任何动作我们需要借助电机来完成。有了电机才能让东西“动”起来。其他的还有“声音”，“光”表现形式。蜂鸣器和喇叭就可以实现声音的输出。

## 代码 与硬件之间的关系

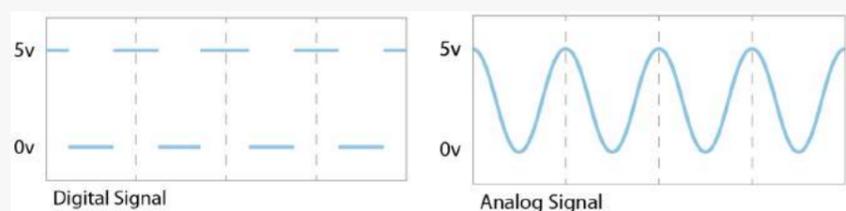
前面说的输入设备、控制器、输出设备都是指硬件。固然重要，就像人的躯体。那人的思想是不是更重要，思想才是控制人行为的根源。大脑其实就是思想的载体，两者缺一不可的。有没有联想到Arduino中了？代码的作用就是思想的作用。虽然我们有控制器，但它不知道怎么去做，需要我们告诉它，而我们告诉它的方式就是通过代码。知道代码的重要性了吗？

## 数字信号 与模拟信号的区别

说下模拟信号与数字信号的区别：

数字(Digital Signal): 只有2个值(0V和5V)。运用在Arduino中，就是高(HIGH)或者低(LOW)，“HIGH”是“1”，对应为5V。“LOW”是“0”，对应为0V。

模拟(Analog Signal): 在一定范围内，有无限值。在Arduino中模拟口中，已经将0V到5V之间的值映射为0~1023范围内的值。比如，0对应为0V，1023对应为5V，512对应为2.5V。



## 电子世界 的“数字”与“模拟”

前面说了，输入设备需要采集信号，再把这个信息给到Arduino，Arduino再给信号输出设备。三个设备之间通过信号联系在了一起。代码是处理这些信号的。下面了解下电子世界的信号是怎么样的？输入设备与控制器是以什么形式“交流”的呢？同样控制器又是怎么与输出设备“沟通”的呢？这里我们需要知道电子世界的两种“语言”——数字信号与模拟信号。

电子世界的数字与模拟与我们平常说的数字与模拟不同。这里的数字，并不是代表的阿拉伯数字的意思。这里的模拟，也不是我们日常认为的真实事物的虚拟。这里需要你颠覆对数字与模拟原有的概念，电子世界将给你一个的新的诠释。不要问为什么，因为这已经是约定俗成的东西了。

## DFRobot中的 “数字”与“模拟”

DFRobot套件中，我们有两种方法可以区分传感器为数字还是模拟。

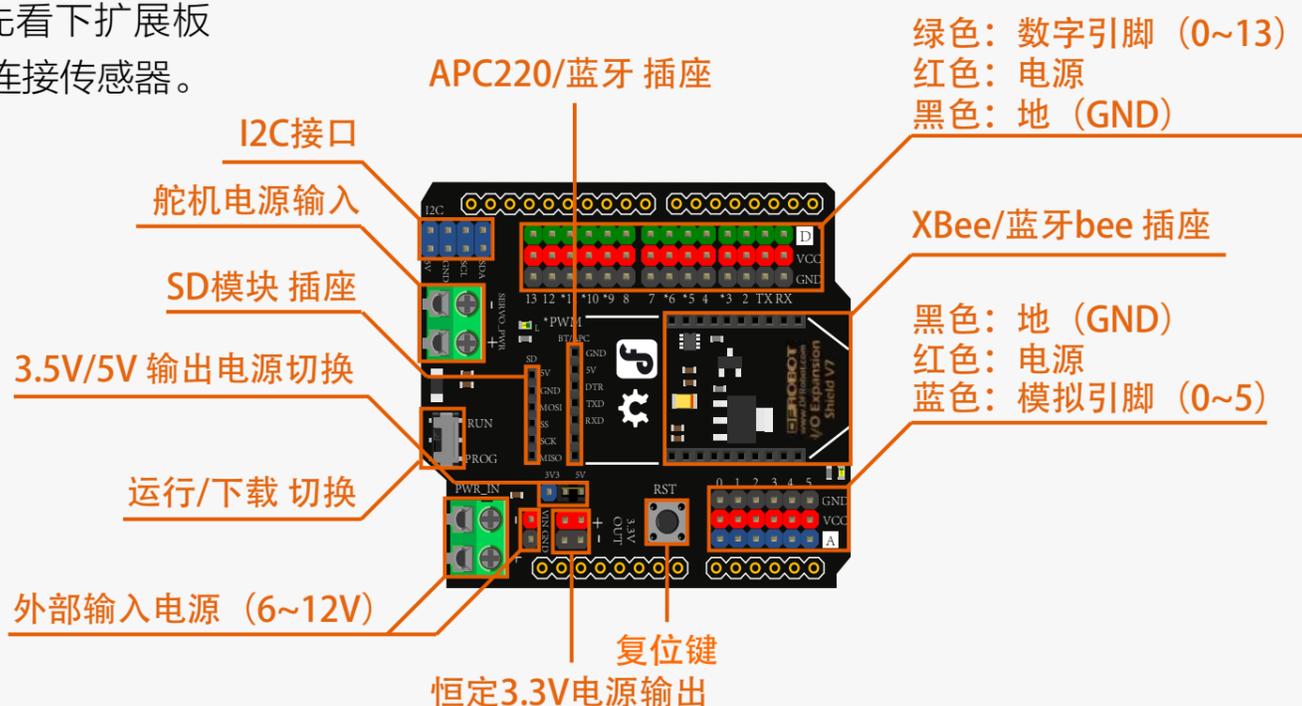
(1) 绿色线为数字信号的传感器，蓝色线为模拟信号的传感器。

(2) 板子上会印有“D”或者“A”的字样，“D”代表“数字”，“A”代表“模拟”。

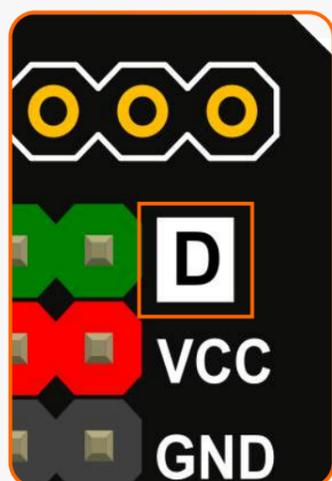


# IO 传感器扩展板 V7.1

顺便来介绍下IO传感器扩展板，先看下扩展板的功能图。我们这里主要就是用来连接传感器。



前面说了DF的传感器会有“D”和“A”的字样。扩展板上也同样有对应的“D”与“A”的字样，对应插上就可以了。



而IO扩展板的最大的好处之一，就是相对于控制板上的仅限的几个电源接口，扩展板大大增加了电源接口和GND接口，不用担心，如果连接多个传感器时，会出现电源接口不够用的情况。

在板子上，数字引脚和模拟引脚下面都会有对应一排“红色”排阵，以及一排“黑色”排阵。这就是扩展出来的电源接口。红色排阵是与电源相连的，黑色排阵对应与GND相通。

**特别说明下DF中的颜色区分：**

绿色：数字信号(Digital Signal)

蓝色：模拟信号(Analog Signal)

红色：电源

黑色：GND

由于这里V7扩展板用到的功能不多，所以就不一一介绍了，感兴趣的可以查看IO 传感器扩展板 V7.1的产品资料库。

这一节主要了解，是什么让东西“活”起来了，整个过程是怎么样的？不仅是需要我们的硬件设备，还需要我们的软件来驱使它来工作。从下一篇开始，我们动手玩了。

前奏3

从串口中认识

“数字”与“模拟”

---

03



**DFROBOT**  
DRIVE THE FUTURE

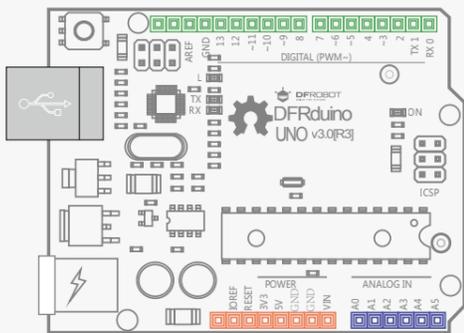
# 让我们开始吧！

前面我们只是对Arduino是如何工作的有了一定的了解,知道了首先需要搭建一个“身体”，也就是整个硬件设备。然后需要“思想”，也就是代码去控制它的大脑(Arduino)。“身体”如何工作的，信号就是他们的“血液”。信号分为两种——数字信号与模拟信号。这一节，我们可以更直观的看到数字信号与模拟信号的区别。

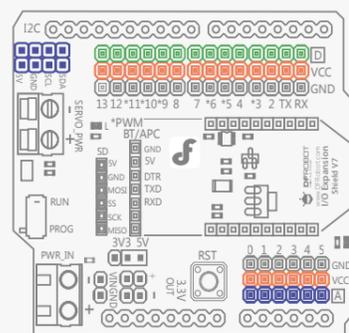
# 数字信号

我们选用一个数字信号传感器来作为例子——数字按钮模块。

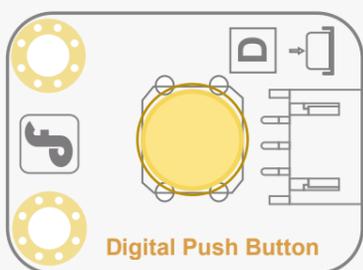
## 所需元件



**x1**  
DFRduino UNO R3  
(以及配套USB数据线)



**x1**  
IO Expansion Shield  
IO 传感器扩展板 V7.1

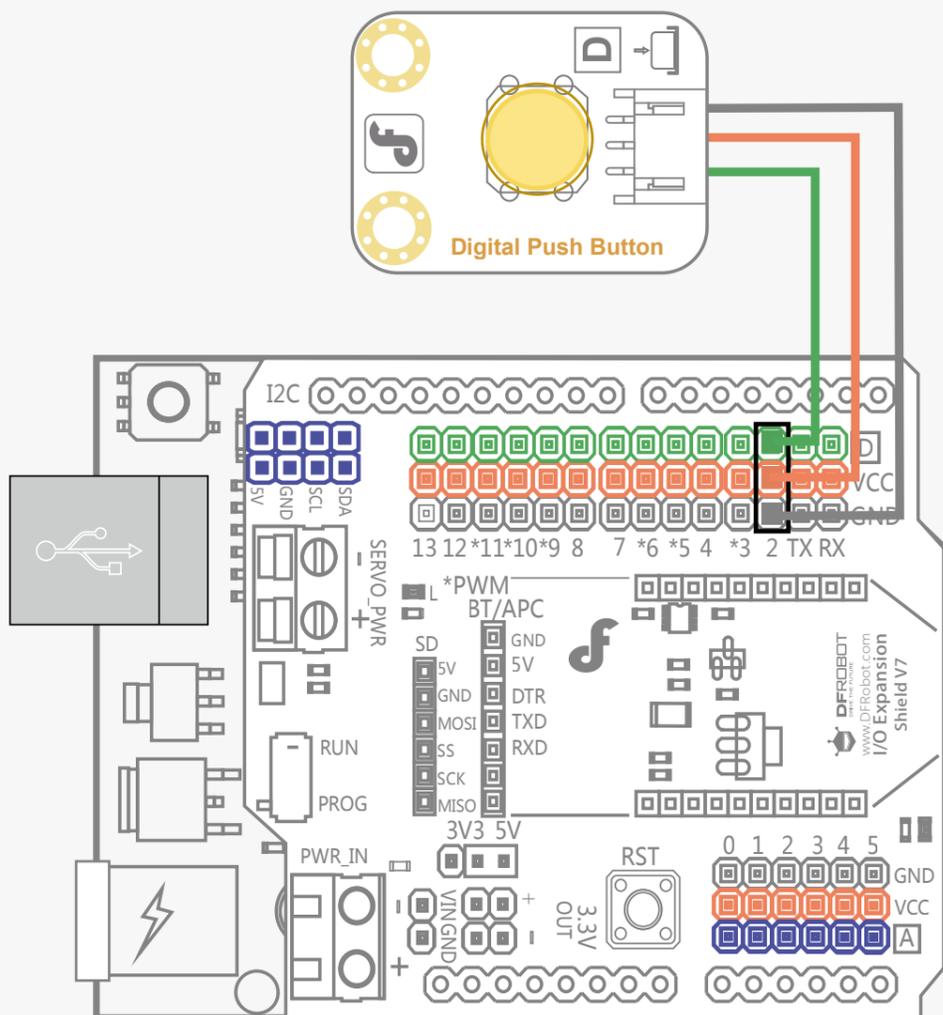


**x1**  
Digital Push Button  
数字大按钮模块

## 硬件连接

首先，从我们的套件中取出IO 传感器扩展板V7.1，把扩展板直接插到UNO上，注意UNO与扩展板的上下引脚一一对应，不要错位。找到数字大按钮模块，直接连接到数字引脚2，需要注意传感器的线序与扩展板上对应。下图为连接的示意图。

完成连接后，给Arduino接上USB数据线，供电，准备下载程序。



## 串口监视器效果

选择菜单中的文件(File) -- 示例(Examples) -- 01 Basics - DigitalReadSerial 代码。

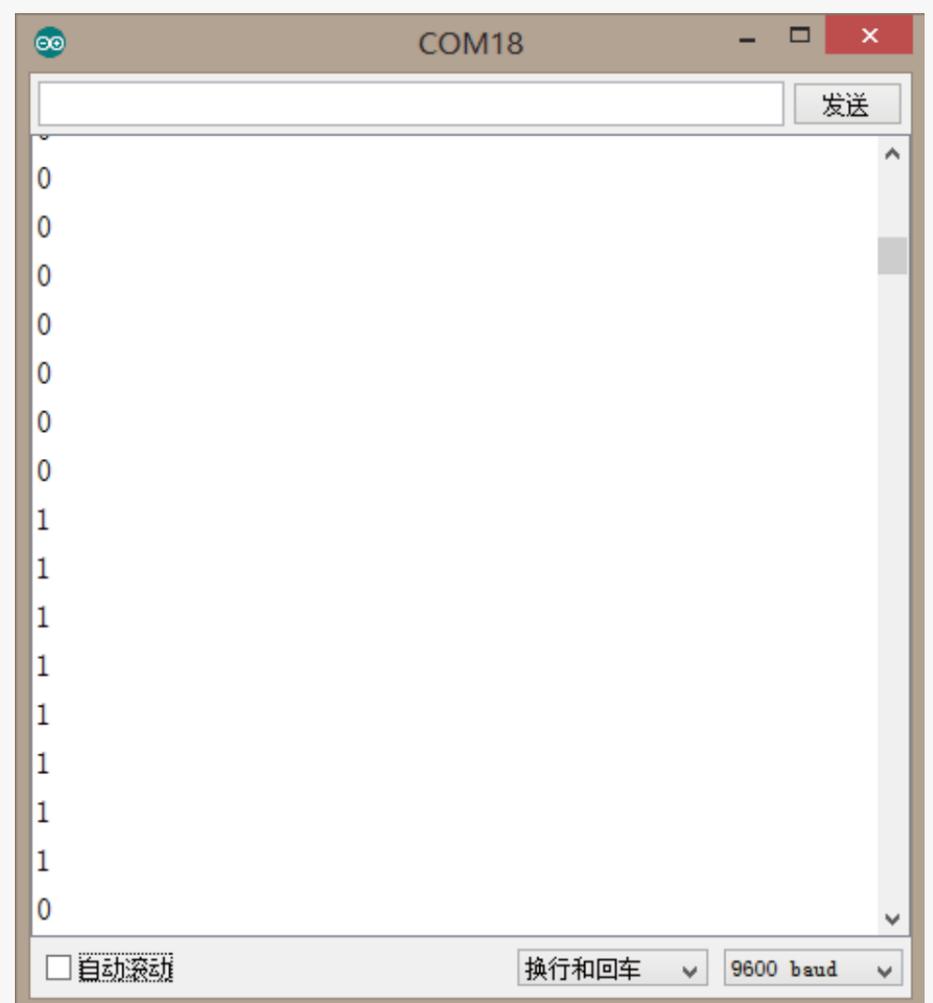
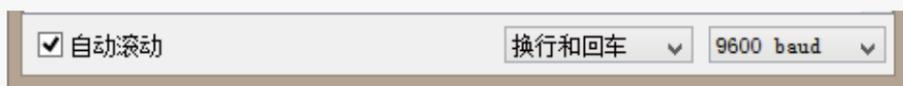
```
int pushButton = 2; //连接到数字引脚2
void setup() //初始化函数
{
  Serial.begin(9600); //设置串口波特率
  pinMode(pushButton, INPUT); //设置按键为输入模式
}
void loop() { // 主函数
  int buttonState = digitalRead(pushButton); //读取数字引脚2的状态
  Serial.println(buttonState); //串口打印出引脚2的状态
  delay(1); //延时1ms
}
```

单击“下载 (Upload)”，给Arduino下载代码。成功下载完程序后，打开Arduino IDE的串口监视器

可以直接从串口读取按钮的状态。按钮没按下的时候，串口显示为“0”，一旦被按下，串口显示为“1”。



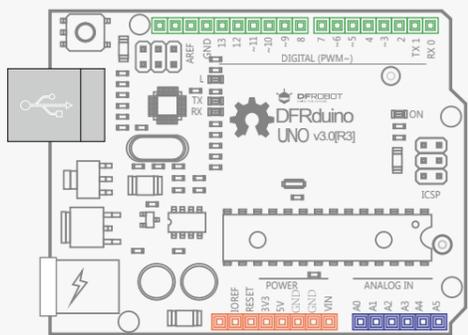
设置串口监视器的波特率为9600。



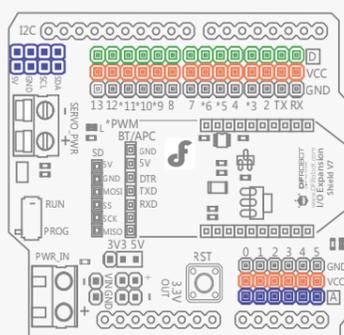
# 模拟信号

我们选用一个模拟量的传感器来作为例子——模拟角度传感器。

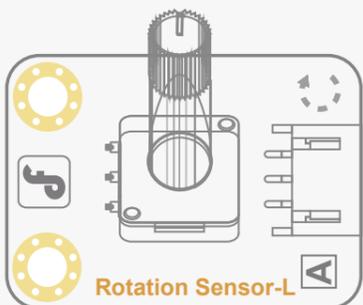
## 所需元件



**x1**  
DFRduino UNO R3  
(以及配套USB数据线)



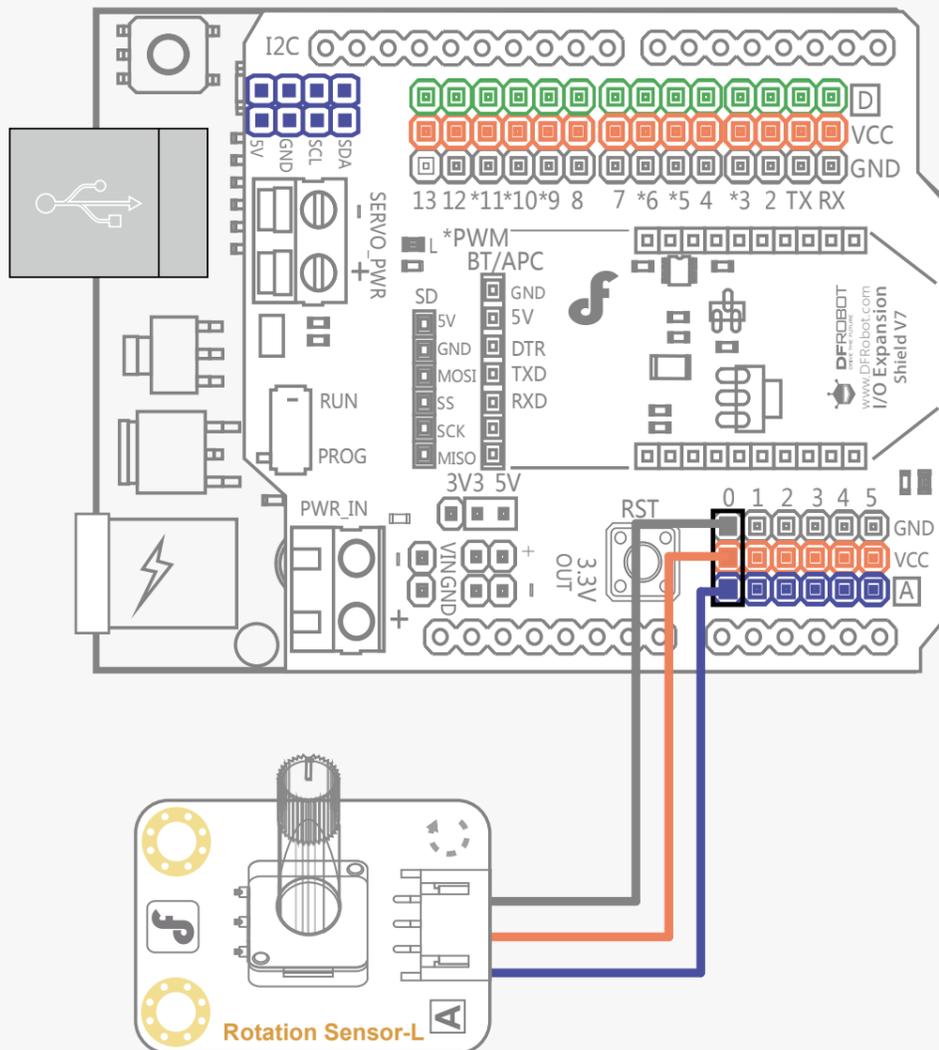
**x1**  
IO Expansion Shield  
IO 传感器扩展板



**x1**  
Analog Rotation Sensor  
模拟角度传感器

# 硬件连接

拔下前面使用的按键，换成模拟角度传感器，直接连接到扩展板的模拟口0。完成连接后，给Arduino接上USB数据线，供电，准备下载程序。



# 输入代码

打开Arduino IDE，选择菜单中的文件(File) -- 示例(Examples)  
-- 01 Basics - AnalogReadSerial 代码。代码如下：

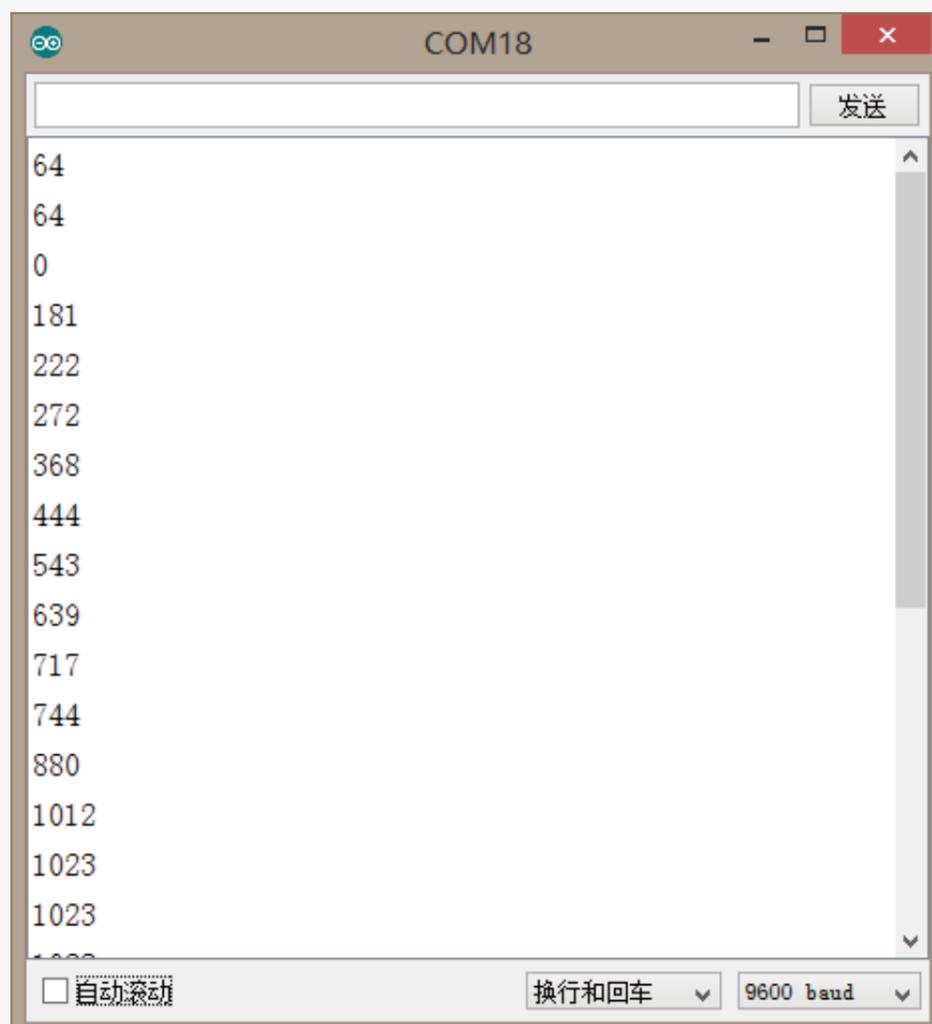
```
void setup() {                                //初始化函数
  Serial.begin(9600);                          //设置串口波特率
}

void loop() {                                  // 主函数
  int sensorValue = analogRead(A0);           //读取模拟引脚0的状态
  Serial.println(sensorValue);                //串口打印出引脚0的状态
  delay(1);                                    //延时1ms
}
```

同样，单击“下载 ( UpLoad )”，给Arduino下载代码。成功下载完程序后，打开Arduino IDE的串口监视器。

并且设置波特率为9600。

试着旋转电位器，可以看到0~1023之间的值。

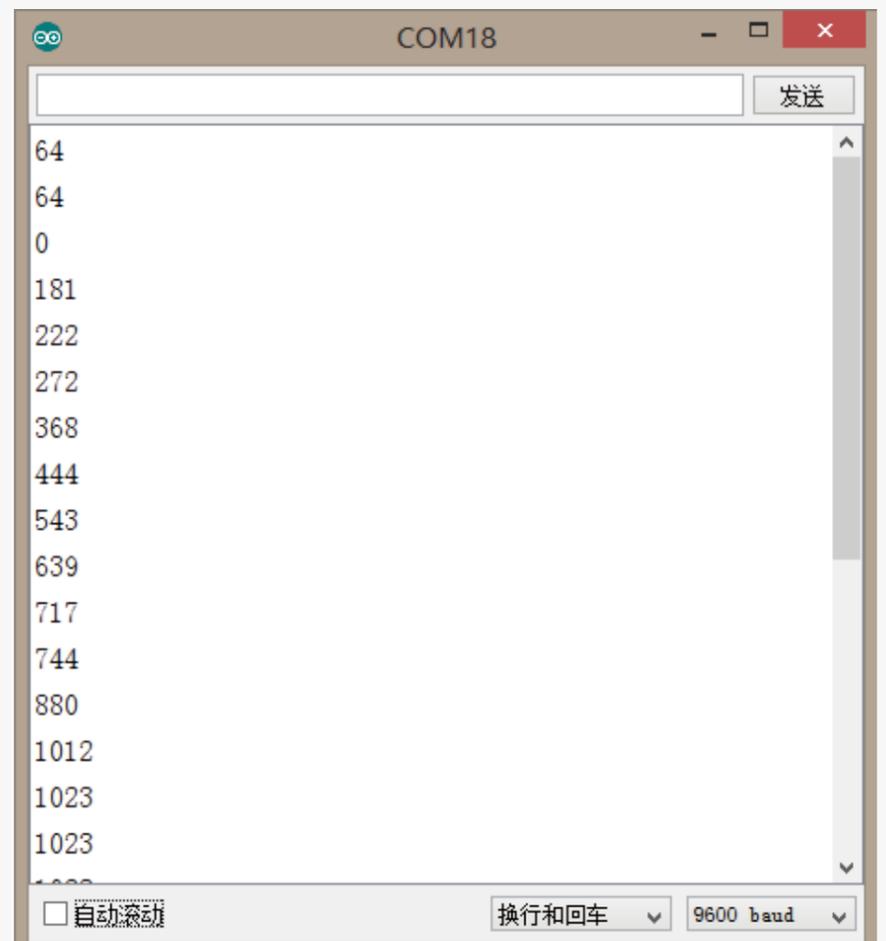
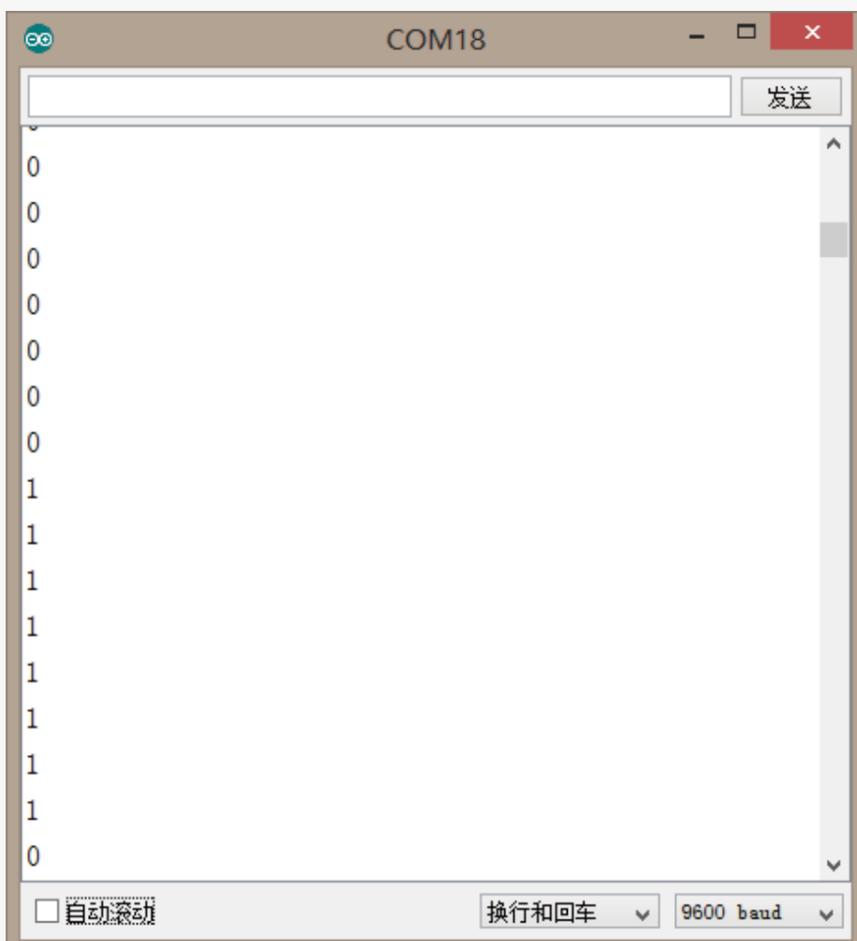


# 两者比较分析

## 串口监视器

串口监视器可以实现Arduino与电脑之间进行互动。可以显示Arduino发送到PC端的数据，还可以让电脑发送数据给Arduino。

从串口监视器可以明显的看出，模拟与数字的鲜明的区别。数字口输出的只有0或者1，而模拟可以输出0~1023之间的任何值。



## 代码区别

从代码可以看出，数字引脚和模拟引脚读数的方式是不同的。数字口使用digitalRead()来读取引脚状态值。而模拟口是通过analogRead()来读取引脚状态值的。其实，最简单的从英文的字面意思应该也能明白这句语句的意思了。不明白没有太大关系，我们之后几节会做详细说明。

数字：

```
int buttonState = digitalRead(pushButton); //读取数字引脚2的状态
```

模拟：

```
int sensorValue = analogRead(A0); //读取模拟引脚0的状态
```

## 动手试一试

可以尝试使用套件中的其他数字传感器和模拟传感器，从串口监视器看看输出效果，是否与前面我们所做的相符。

# 项目一 点亮一盏灯

---

04



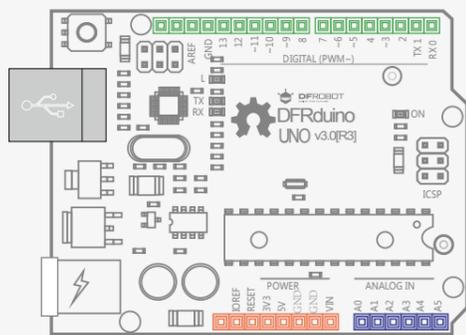
**DFROBOT**  
DRIVE THE FUTURE

# 项目一 点亮一盏灯

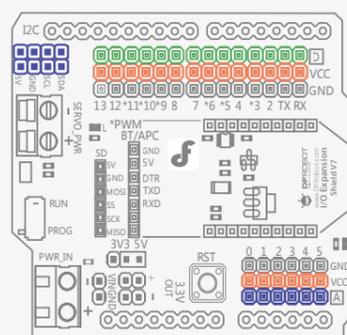
在前面几篇前奏中，我们已经对Arduino有了简单的了解，整个装置工作是依赖于哪些部分。也了解了电子世界最重要两个量，数字量与模拟量。接下来我们就正式开始做东西了，第一个要做的必须是最经典的，最经典的莫过于“blink”。

其实，前面在一开始驱动安装的时候就用过这段代码了，区别在于这里将不使用板子上的LED13(也就是“L”灯)，而是在数字引脚13连接一个LED。

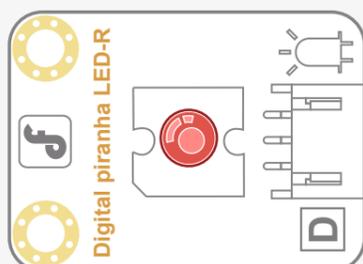
## 所需元件



**x1**  
DFRduino UNO R3  
(以及配套USB数据线)

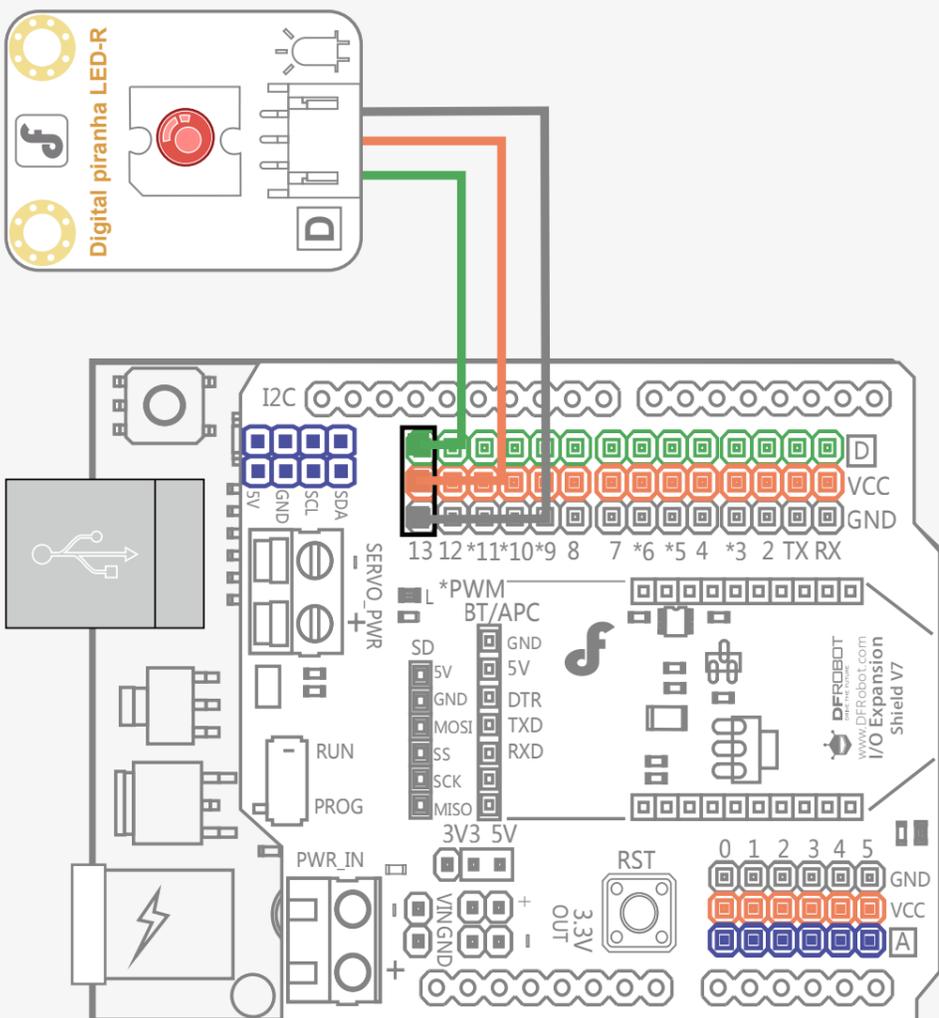


**x1**  
IO Expansion Shield  
IO 传感器扩展板



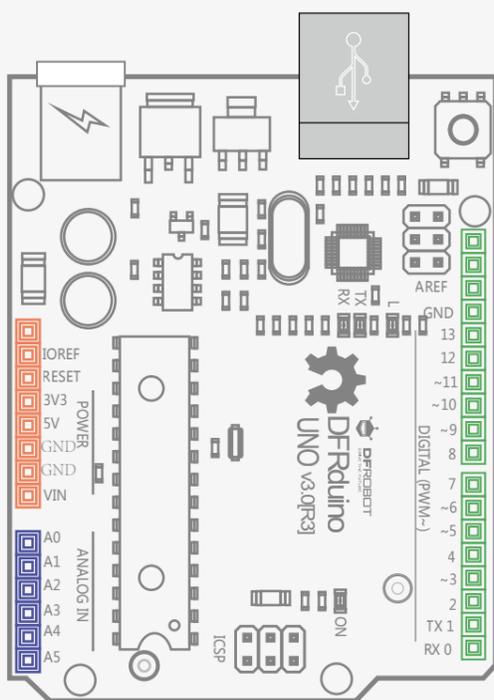
**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块

# 硬件连接

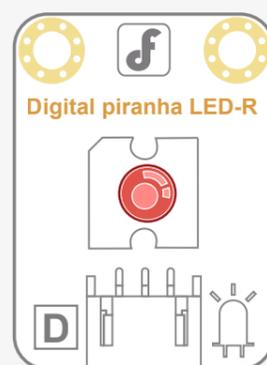


## 硬件分析（数字输出）

我们从前面几章说的输入输出的角度来看。整个装置只有两个部分，控制与输出。Arduino就是控制设备，LED发光模块就是输出设备。对的，这个整个装置是没有输入设备的。有了这么一分析，我们再看代码就不那么难理解了。



控制设备



输出设备

## 输入代码

输入代码也是一种学习编程的过程，虽然提供代码的压缩包，但还是建议初学者自己输入代码，亲身体验一下。打开Arduino IDE，在编辑框中输入样例代码。

```
//项目一 —— LED 闪烁
/*
描述：LED每隔一秒交替亮灭一次
*/
int ledPin = 13;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin,HIGH);
    delay(1000);
    digitalWrite(ledPin,LOW);
    delay(1000);
}
```

输入完毕后，点击IDE的“校验（Verify）”，查看输入代码是否通过编译。如果显示没有错误，单击“下载（Upload）”，给Arduino下载代码。以上每一步都完成了的话，你应该可以看到面包板上的红色LED每隔一秒交替亮灭一次。

现在让我们来回顾一下代码，看看它们是如何工作的。

## 代码回顾

先说下Arduino代码必须具备的两个组成部分：

```
void setup() {  
  // 写入setup代码，只运行一次：  
}  
  
void loop() {  
  // 写入main代码，重复运行：  
}
```

Arduino代码必须包含setup()和loop()这两个函数。setup英文中是“设置”的意思。所以setup()函数是用于一些初始化设置的，只在代码一开始时，运行一次。loop是“循环”的意思，只要Arduino不掉电，loop就会不停的重复运行。

由于LED是输出设备，所以不难看出，在setup()函数中先初始化LED为输出模式。

函数格式如下：

```
pinMode(pin,mode)
```

这个函数是用来设置Arduino数字引脚的模式，只用于数字引脚定义是输入(INPUT)还是输出(OUTPUT)。

pin指数字引脚号，mode指引脚模式(OUTPUT/INPUT)。

回头看下代码中，

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

这句话的意思就是，将ledPin设置为输出模式，中间的逗号可不能省。那ledPin是什么呢？

看下代码的第一句话：

```
int ledPin = 13;
```

我们在一开始的时候给13号引脚起了个名字叫做ledPin，所以ledPin就代表了13号引脚。前面的int可不能少！int代表了ledPin是个整数。

明白了这两句话的意思了，如果我们现在需要换个引脚，LED不连接到13号引脚，连接到10引脚，可以怎么写：

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

只需把pin换成对应的引脚号就行了。

再看下loop()函数，loop函数中就只用到了一个函数digitalWrite()。

函数格式如下：

`digitalWrite(pin,value)`

这个函数的意义是：引脚pin在pinMode()的中被设置为OUTPUT模式时，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。

`digitalWrite(ledPin,HIGH);`

//LED被点亮

`digitalWrite(ledPin,LOW);`

//LED被熄灭

代码中的，ledPin同样指引脚。写入HIGH时，引脚13就被至高，LED被点亮。写入LOW时，引脚13就被拉低，LED被熄灭。

亮与灭直接还有句语句：

`delay(1000);`

delay是延时的意思。括号中写入的是毫秒(ms)。所以，delay(1000)就是延时1s的意思。最后实现的就是LED亮一秒，灭一秒，一直无限循环。

细心的朋友可能注意到，代码开始部分有段带“//”和“/\*...\*/”的文字：

//项目一 —— LED 闪烁

/\*

描述：LED每隔一秒交替亮灭一次

\*/

这是代码中的说明文字，可以叫做注释。是以“//”开始，这个符号所在行之后的文字将不被编译器编译。

还有另外一种写注释的方式，用“/\*...\*/”，这个符号的作用是可以注释多行，这也是与上一种注释方式的区别之处。在/\*和\*/中间的所有内容都将被编译器忽略，不进行编译。IDE将自动把注释的文字颜色变为灰色。

## 趣味练习

(1)能否试试变换LED的亮灭速度，让LED保持关闭5秒，然后快速闪烁一下（250毫秒），就像汽车报警器上的LED指示灯那样。

(2)通过改变LED开和关的时间，可以产生不同的效果，开关时间短，则感觉动感，开关时间长，则感觉柔和。

# 项目二 感应灯

---

05

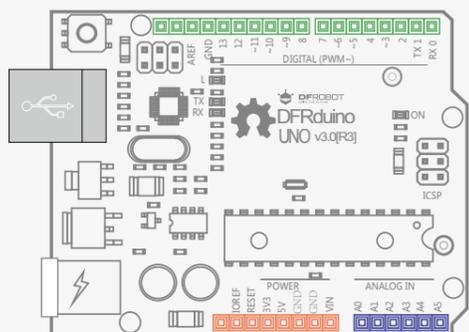


**DFROBOT**  
DRIVE THE FUTURE

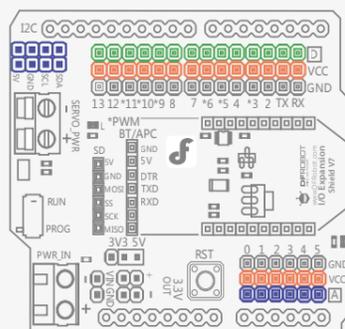
# 项目二 感应灯

这节课要做的是个感应灯，当有人经过的时候，LED灯就会自动亮起，人一旦走了，LED又自动关闭了。这里用到的传感器是人体红外热释电运动传感器。它是一种能检测人或动物身体发射的红外线的传感器。拿它来做整人玩具应该是个不错的选择！

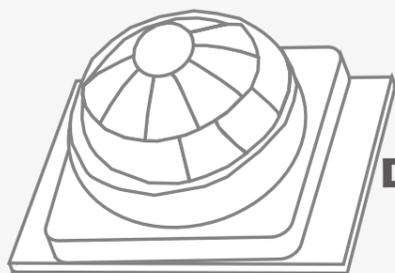
## 所需元件



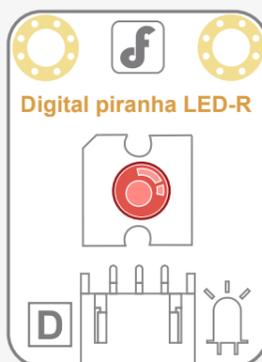
**x1**  
DFRduino UNO R3



**x1**  
IO Expansion Shield  
IO 传感器扩展板 V7.1



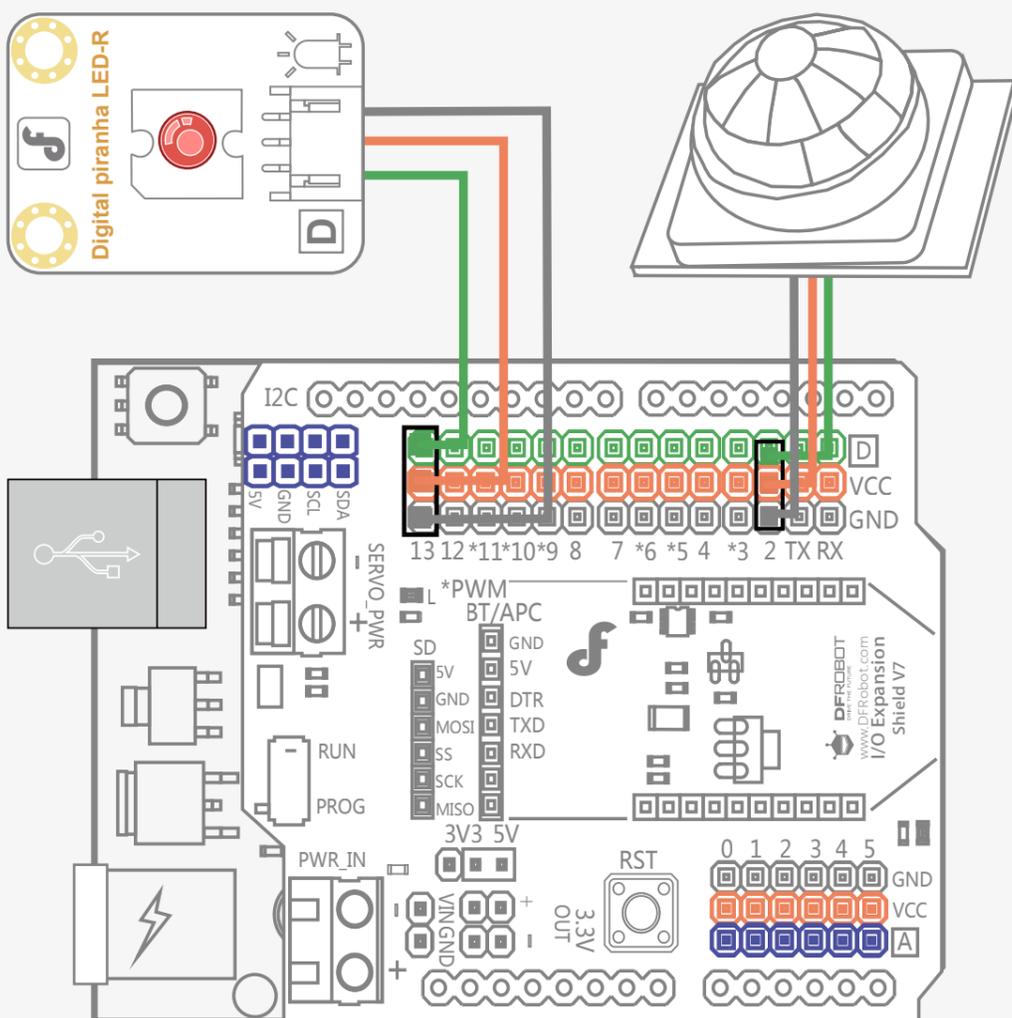
**x1**  
Digital Infrared Motion Sensor  
人体红外热释电运动传感器



**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块

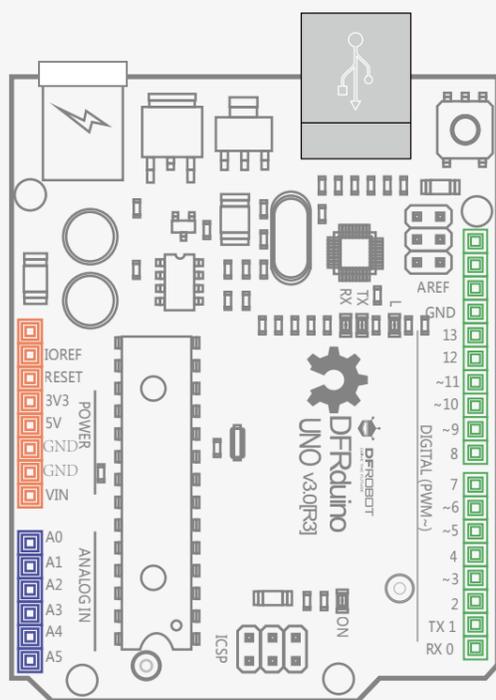
# 硬件连接

人体红外热释电运动传感器  
连接数字引脚2  
数字食人鱼红色LED发光模块  
连接数字引脚13

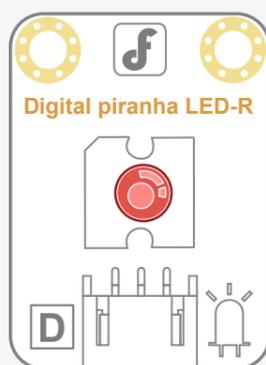


## 硬件分析 ( 数字输入—数字输出 )

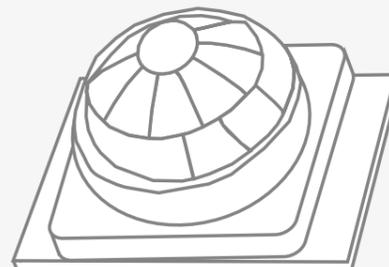
整个装置分为三个部分，输入，控制与输出。人体红外热释电运动传感器为输入设备，Arduino就是控制设备，LED发光模块就是输出设备。又由于人体红外热释电运动传感器为数字量的传感器，所以接数字口。LED输出信号也是数字量，同样接数字口。



控制设备



输出设备



输入设备

## 输入代码

---

```
//项目二 —— 感应灯
int sensorPin = 2;           //传感器连接到数字2
int ledPin = 13;            //LED连接到数字13
int sensorState = 0;        //变量sensorState用于存储传感器状态

void setup() {
  pinMode(ledPin, OUTPUT);  //LED为输出设备
  pinMode(sensorPin, INPUT); //传感器为输入设备
}

void loop(){
  sensorState = digitalRead(sensorPin); //读取传感器的值
  if (sensorState == HIGH) {           //如果为高, LED亮
    digitalWrite(ledPin, HIGH);
  }
  else {                                //否则, LED灭
    digitalWrite(ledPin, LOW);
  }
}
```

下载完成后, 可以试着人走开, 等待一段时间, 看看LED是否会关掉。随后再试着靠近, LED是不是会自动亮起。

## 代码回顾

---

还是由输入输出着手，传感器是输入(INPUT)，LED是输出(OUTPUT)。所以在初始化中设置为：

```
pinMode(ledPin, OUTPUT);      //LED为输出设备
pinMode(sensorPin, INPUT);    //传感器为输入设备
```

有了输入设备，我们需要读取输入设备的值，才能进行之后的判断，所以loop函数一开始就是读取传感器的值。

读取数字传感器状态的函数是——digitalRead()。

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

函数格式如下：

### digitalRead(pin)

这个函数是用来读取数字引脚状态，HIGH还是LOW。人体红外热释电传感器有人或者动物走动时，读到HIGH，否则读到LOW。代码的后半段就是对判断出来的值来执行相应动作。（HIGH代表1，LOW代表0）。

数字传感器只会读到两个值(HIGH和LOW)。这里要用到新的一个语句——if语句。

if语句格式如下：

```
(1) if(表达式){
    语句;
}

(2) if(表达式){
    语句;
}else{
    语句;
}
```

表达式是指我们的判断条件，通常为一些关系式或逻辑式，也可是直接表示某一数值。如果if表达式条件为真，则执行if中的语句。表达式条件为假，则跳出if语句。格式(1)多用于一种判断中，格式(2)多用于两种判断的情况。

这里只有两种情况，传感器有人读到的是高，否则就是低。所以用的if...else语句。

```
if (sensorState == HIGH) {
    ... //如果为高, LED亮
}
else {
    ... //否则, LED灭
}
```

“==”是一种比较运算符，用于判断两个数值是否相等，记得是“**双等号**”！而“=”是赋值的意思。把等号右边的值赋给左边。

我们常用的运算符有：

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

特别说明下，小于等于和大于等于，<和=之间不能留有空格，否则编译不通过。

当然，除了比较运算符外，程序也可以用的+、-、\*、/（加、减、乘、除）这些常用的算术运算符。

## 趣味练习

(1) 喜欢去鬼屋玩的小伙伴们，一定会喜欢这个，给LED做个“面目狰狞”的壳儿，放在一个阴冷黑暗的小屋，再配点刺激的音乐，应该还是挺带感的。当然灯光效果也少不了，可以换成开关切换频率较快的模式。

(2) 文艺青年可以拿这个人体红外热释电传感器做个漂亮的装饰灯。详见教程：

<http://www.dfrobot.com.cn/community/forum.php?mod=viewthread&tid=1983&highlight=IQ%E7%81%AF>

项目三

# Mini台灯

---

06

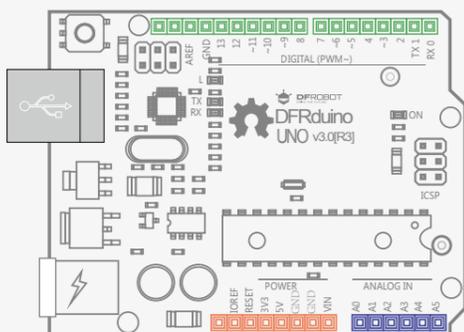


**DFROBOT**  
DRIVE THE FUTURE

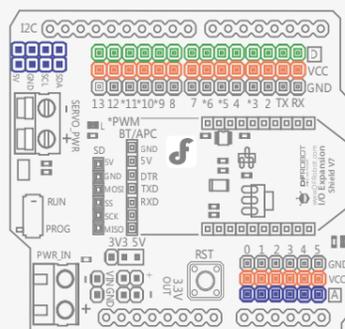
# 项目三 Mini台灯

台灯，是我们再常见不过的东西。“啪”一下开，“啪”一下关。Mini台灯的功能就和台灯类似，按钮就像是LED的开关，每按一下，就会切换LED的状态。做完之后再给小灯来个壳儿，一定很Q。

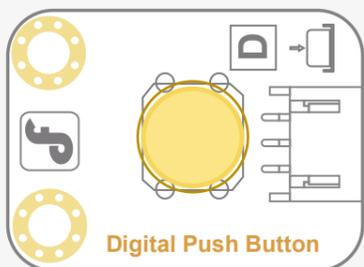
## 所需元件



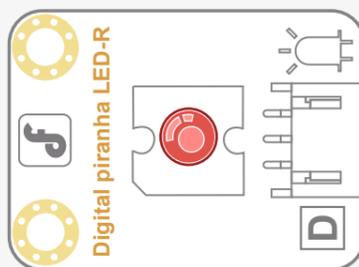
**x1**  
DFRduino UNO R3



**x1**  
IO Expansion Shield  
IO 传感器扩展板 V7.1



**x1**  
Digital Push Button  
数字大按钮模块

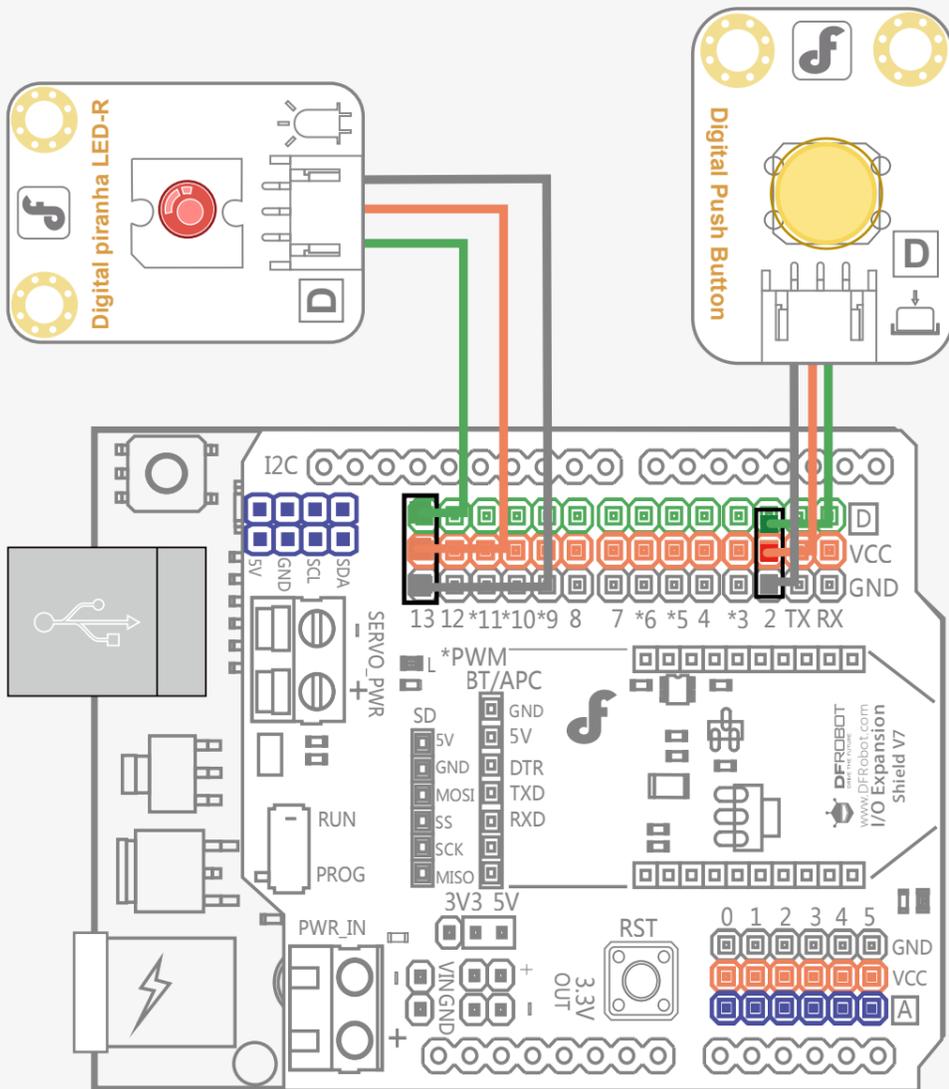


**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块

# 硬件连接

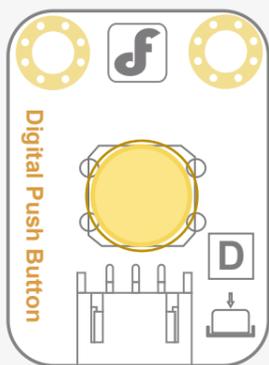
数字大按钮  
连接数字引脚2

数字食人鱼红色LED发光模块连接  
数字引脚13

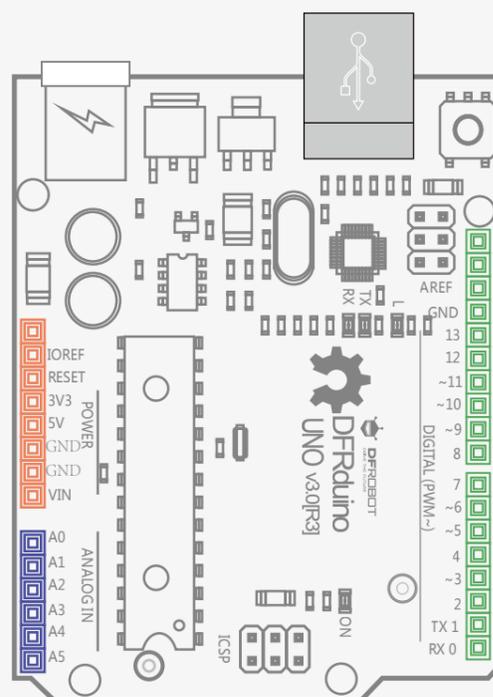


## 硬件分析 ( 数字输入—数字输出 )

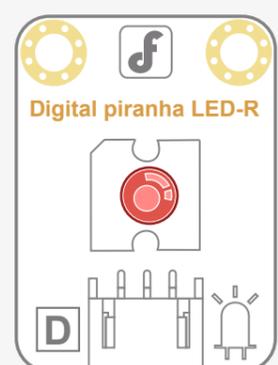
很明显，大按钮是输入设备，LED是输出设备。和前面感应灯类似，也是一个数字输入控制一个数字输出。只是形式与代码有所不同。



输入设备



控制设备



输出设备

## 输入代码

```
//项目三 —— 小台灯
int buttonPin = 2;           //按钮连接到数字2
int ledPin = 13;            //LED连接到数字13

int ledState = HIGH;        // ledState记录LED状态
int buttonState;           // buttonState记录按键状态
int lastButtonState = LOW;  // lastbuttonState记录按键前一个状态

long lastDebounceTime = 0;
long debounceDelay = 50;   //去除抖动时间

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

  digitalWrite(ledPin, ledState);
}

void loop() {
  int reading = digitalRead(buttonPin); //reading用来存储buttonPin的数据
  if (reading != lastButtonState) {    // 一旦检测到数据发生变化，记录当前时间
    lastDebounceTime = millis();
  }
  // 等待50ms，再进行一次判断，是否和当前button状态相同
  // 如果和当前状态不相同，改变button状态
  // 同时，如果button状态为高（也就是被按下），那么就改变led的状态
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

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

  digitalWrite(ledPin, ledState);      // 改变button前一个状态值
  lastButtonState = reading;
}
```

下载完代码，按下按钮，灯点亮。再按下按钮，灯熄灭。是不是很像个小台灯？

## 代码回顾

由硬件分析可以看出，按键是输入设备，LED是输出设备。

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

通过digitalWrite()读取按键的状态:

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

按键在由低变高或者由高变低时，都会有个抖动的过程，时间非常的短，如下图所示:



为了避免由于抖动产生的错误信号，所以我们代码中有一个去抖的过程。去抖的方法很简单，就是等到数据发生变化时，隔一段时间再检测一次。

一旦检测到读取的数据发生变化，通过millis()函数记下时间:

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

millis()是一个函数，该函数是Arduino语言自有的函数，它返回值是一个时间，Arduino开始运行到执行到当前的时间，也称之为机器时间，就像一个隐形时钟，从控制器开始运行的那一刻起开始计时，以毫秒为单位。

再等待50ms，再进行一次判断，是否和当前button状态相同。如果和当前状态不相同，改变button状态。同时，如果button状态为高（也就是被按下），那么就改变LED的状态。

```
if ((millis() - lastDebounceTime) > de-  
bounceDelay) {  
    if (reading != buttonState) {  
        buttonState = reading;  
  
        if (buttonState == HIGH) {  
            ledState = !led-  
State;  
        }  
    }  
}
```

### 趣味练习

灯光门铃

现在越来越多年轻人回家就塞上耳机，即使在家都听不见门铃声，那就自制一个灯光门铃，有人来了，灯就开始狂闪，提醒里面的人，门口有人在按门铃了。这样的门铃也同样适用于那些耳朵不好的老人，又或者是那些聋哑人士。

# 项目四

# 声控灯

---

07

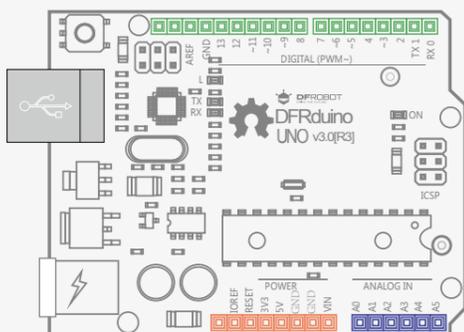


**DFROBOT**  
DRIVE THE FUTURE

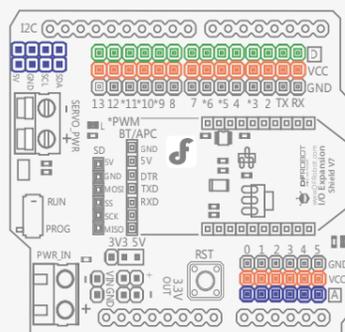
# 项目四 声控灯

小时候有没有对走廊的声控灯很感兴趣呢？会不会拼命的跺脚只为让那盏灯点亮。这节课我们就做个这样的声控灯。只要你轻轻拍下手，灯就自动亮起来了，没了声音，灯就又自动关了。这里用到的是个声音传感器，我们可以利用这个传感器做出更多互动作品，通过声音触发来控制更多好玩儿的东西，比如说做个发光鼓等等。

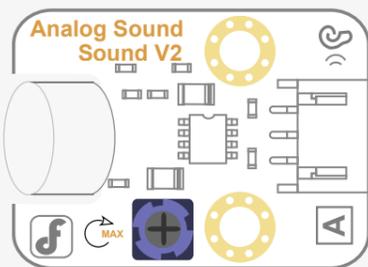
## 所需元件



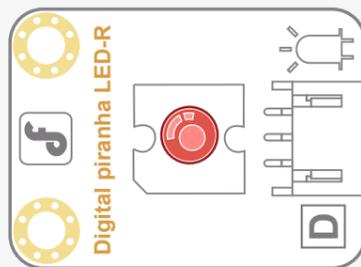
**x1**  
DFRduino UNO R3



**x1**  
IO Expansion Shield  
IO 传感器扩展板 V7.1



**x1**  
Analog Sound Sensor  
模拟声音传感器

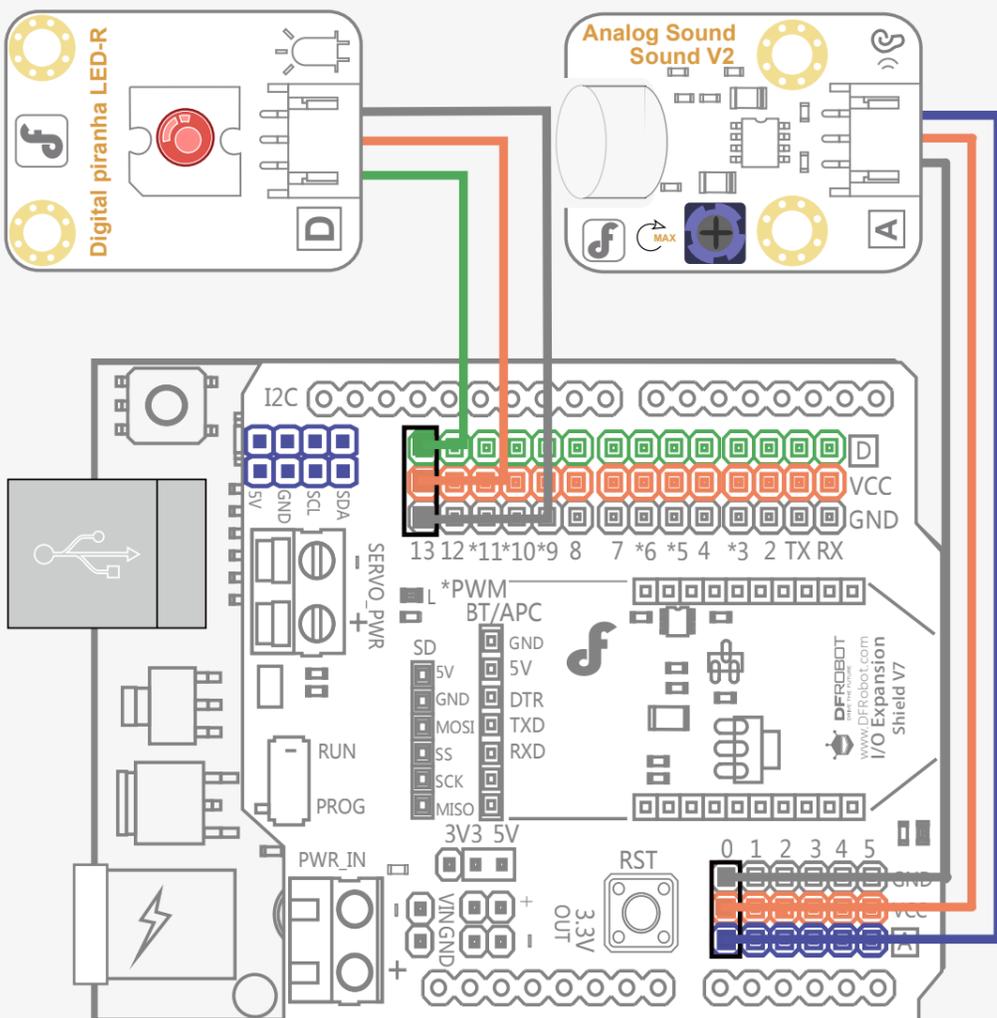


**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块

# 硬件连接

模拟声音传感器  
连接模拟引脚0

数字食人鱼红色LED发光模块  
连接数字引脚13

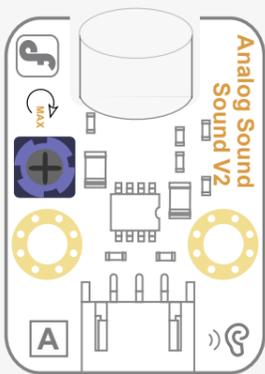


## 硬件分析

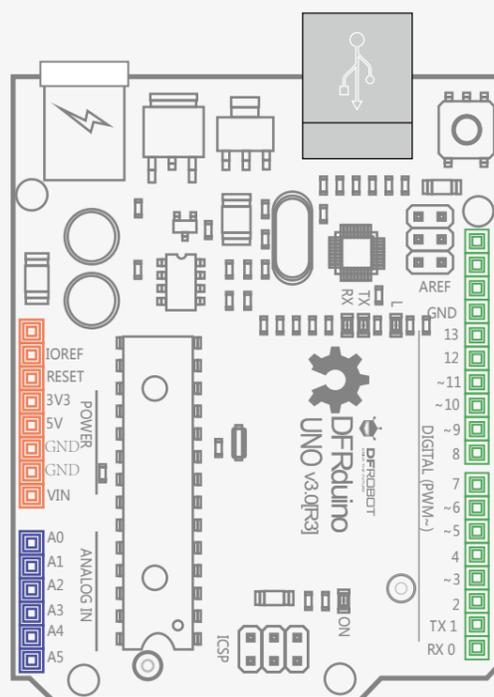
### ( 数字输入—数字输出 )

前面几次我们接触的都是数字传感器，这次我们要尝试使用模拟传感器了，还记得在一开始说的数字与模拟的区别吗？(串口中认识“数字”与“模拟”一节)。数字，只有两个值(0/1)。模拟，是线性的，理论上的无限值(0~1023)。

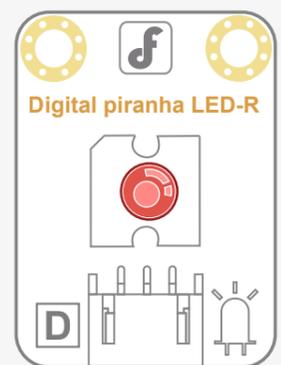
所以这里是个，模拟输入，数字输出的模式。



输入设备



控制设备



输出设备

## 输入代码

输入代码也是一种学习编程的过程，虽然提供代码的压缩包，但还是建议初学者自己输入代码，亲身体会一下。打开Arduino IDE，在编辑框中输入样例代码。

---

```
//项目四 —— 声控灯
int soundPin = 0;           //声音传感器接到模拟0
int ledPin = 13;           //LED接到数字13

void setup() {
  pinMode(ledPin, OUTPUT);
  // Serial.begin(9600);    //用于调试
}

void loop(){
  int soundState = analogRead(soundPin); //读取传感器的值
  // Serial.println(soundState); //串口打印声音传感器的值

  //如果声音值大于10，亮灯，并持续10s，否则关灯
  if (soundState > 10) {
    digitalWrite(ledPin, HIGH);
    delay(10000);
  }else{
    digitalWrite(ledPin, LOW);
  }
}
```

对着话筒拍下手，或者说句话，试试灯能不能点亮？

## 代码回顾

---

在`setup()`中只设置了LED为输出，为什么没有设置声音传感器输入模式？这是因为模拟口都是输入设置，所以不需要设置了。

声音传感器是输入设备，所以需要读取对应模拟口0的值。与读取数字口函数`digitalRead(pin)`类似，所以模拟口读取函数是：

### `analogRead(pin)`

这个函数用于从模拟引脚读值，`pin`是指连接的模拟引脚。Arduino的模拟引脚连接到一个了10位A/D转换，输入0~5V的电压对应读到0~1023的数值，每个读到的数值对应的都是一个电压值。比如 $512 = 2.5V$ 。

最后是一个if判断，判断是否到达你预设的值。

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

需要修改预设值的话，可以打开串口监视器，看看你需要的声音强度的值在什么范围，然后做相应调整就可以了。

# 项目五 呼吸灯

---

08

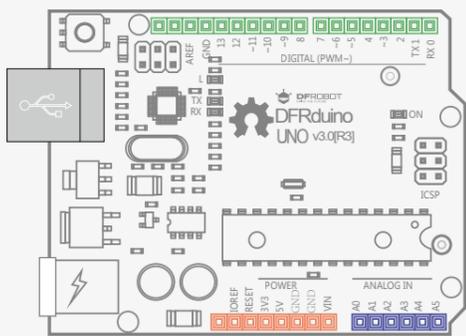


**DFROBOT**  
DRIVE THE FUTURE

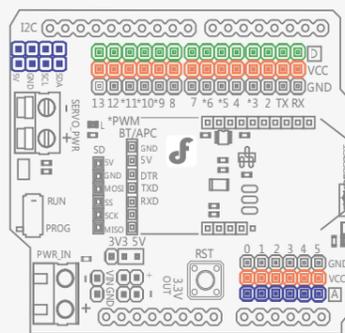
# 项目五 呼吸灯

在前面几章中，我们知道了如何控制LED亮灭。但Arduino还有个很强大的功能通过程序来控制LED的明亮度。Arduino UNO数字引脚中有六个引脚标有“~”，这个符号就说明该口具有PWM功能。我们动手做一下，在做的过程中体会PWM的神奇力量！下面就介绍一个呼吸灯，所谓呼吸灯，就是让灯有一个由亮到暗，再到亮的逐渐变化的过程，感觉像是在均匀的呼吸。

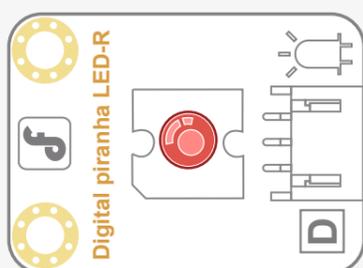
## 所需元件



**x1**  
DFRduino UNO R3



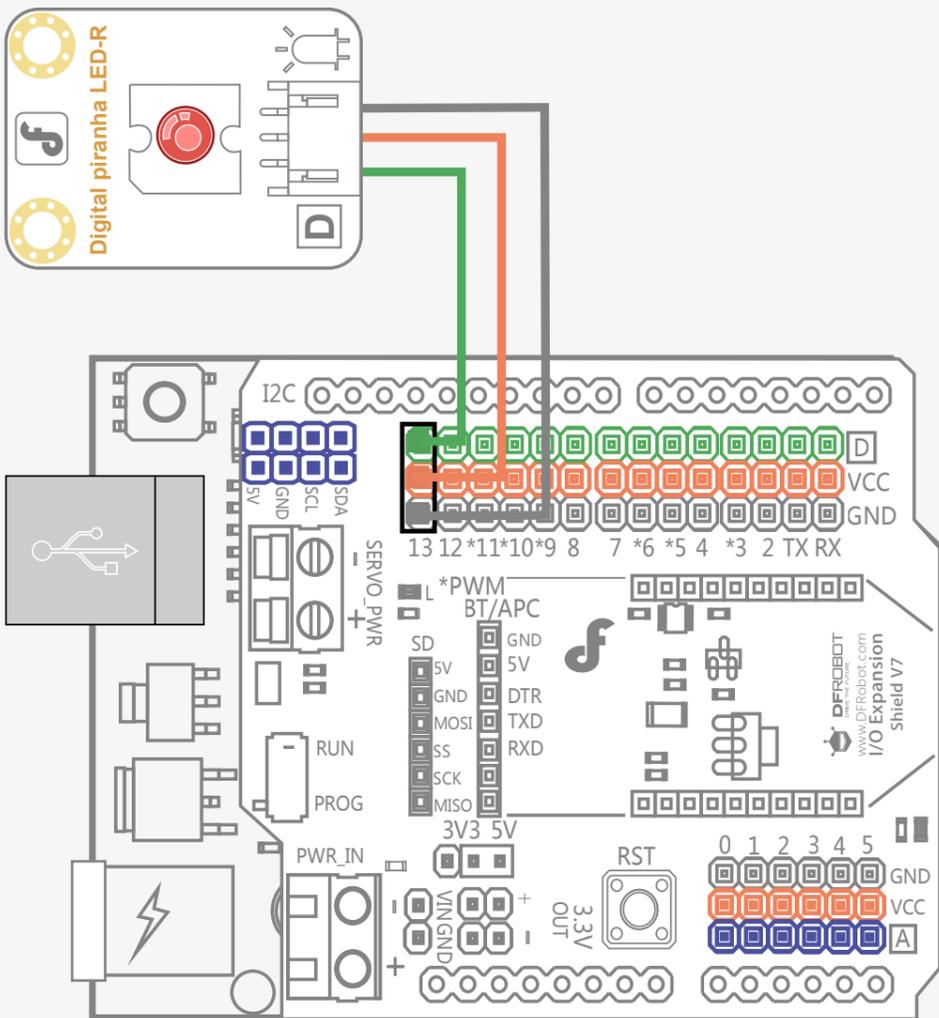
**x1**  
IO Expansion Shield  
IO 传感器扩展板



**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块

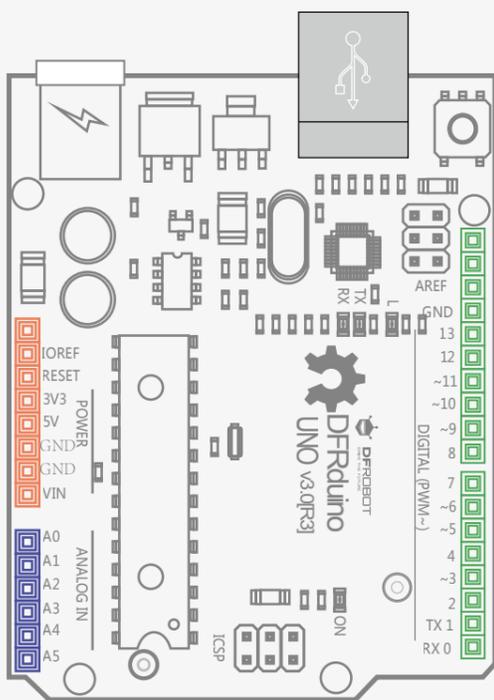
# 硬件连接

数字食人鱼红色LED发光模块  
连接数字引脚10

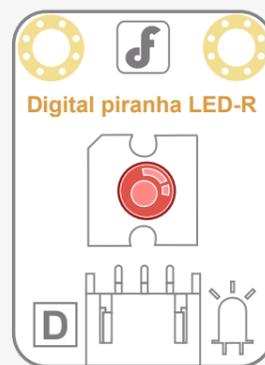


## 硬件分析（模拟输出）

和项目一（点亮一盏灯）类似的装置，同样没有输入设备，只有一个输出设备，但又有所不同。项目一LED是作为数字输出，而这里我们是作为模拟输出。代码部分会说明。



控制设备



输出设备

## 输入代码

---

```
//项目五 - 呼吸灯
int ledPin = 10;

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);
    }
}
```

代码下载完成后，我们可以看到LED会有个逐渐由亮到灭的一个缓慢过程，而不是直接的亮灭，如同呼吸一般，均匀变化。

## 代码回顾

当我们需要重复执行某句话时，我们可以使用for语句。  
for语句格式如下：



for循环顺序如下：

第一轮：1 2 3 4

第二轮：2 3 4

...

直到2不成立，for循环结束。

知道了这么个顺序之后，回到代码中：

```

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

```

这两个for语句实现了value的值不断由0增加到255，随之在从255减到0，在增加到255……,无限循环下去。

再看下for里面，涉及一个新函数analogWrite()。我们知道数字口只有0和1两个状态，那如何发送一个模拟值到一个数字引脚呢？就要用到该函数。观察一下Arduino板，查看数字引脚，你会发现其中6个引脚旁标有“~”，这些引脚不同于其他引脚，它们可以输出PWM信号。

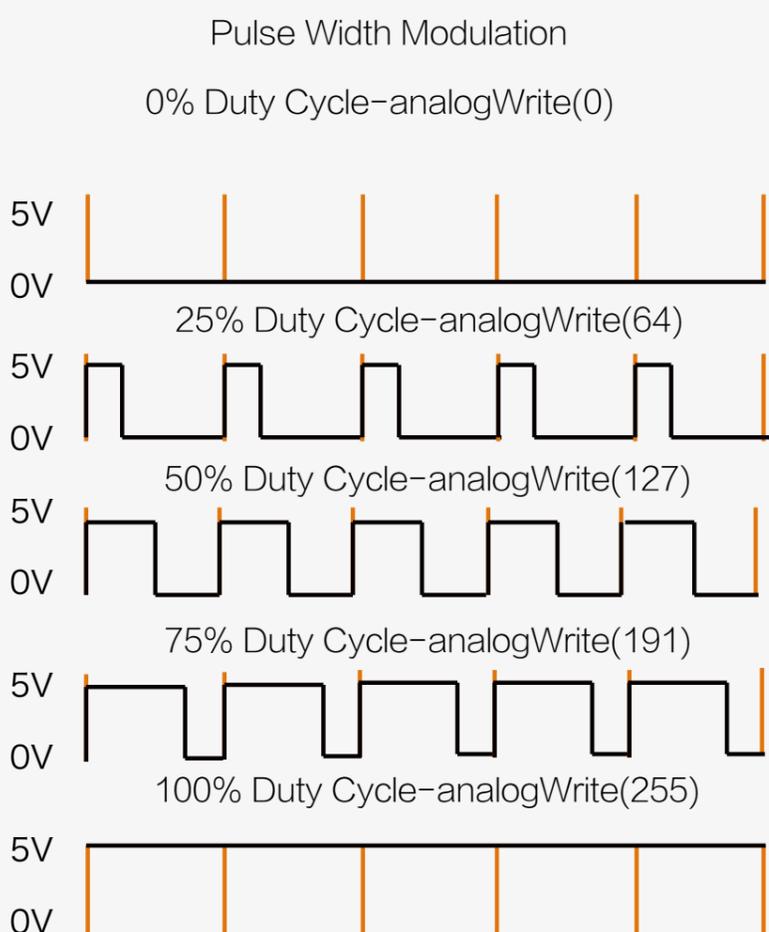
函数格式如下：

**analogWrite(pin,value)**

analogWrite()函数用于给PWM口写入一个0~255的模拟值。所以，value是在0~255之间的值。特别注意的是，analogWrite()函数只能写入具有PWM功能的数字引脚，也就是3, 5, 6, 9, 10, 11引脚。

PWM是一项通过数字方法来获得模拟量的技术。数字控制来形成一个方波，方波信号只有开关两种状态（也就是我们数字引脚的高低）。通过控制开与关所持续时间的比值就能模拟到一个0到5V之间变化的电压。开（学术上称为高电平）所占用的时间就叫做脉冲宽度，所以PWM也叫做脉冲宽度调制。

通过下面五个方波来更形象的了解一下PWM。



上图橘色竖线代表方波的一个周期。每个analogWrite(value)中写入的value都能对应一个百分比，这个百分比也称为占空比(Duty Cycle)，指的是一个周期内高电平持续时间比上低电平持续时间得到的百分比。图中，从上往下，第一个方波，占空比为0%，对应的value为0。LED亮度最低，也就是灭的状态。高电平持续时间越长，也就越亮。所以，最后一个占空比为100%的对应value是255，LED最亮。50%就是最亮的一半了，25%则相对更暗。

PWM比较多的用于调节LED灯的亮度。或者是电机的转动速度，电机带动的车轮速度也就能很容易控制了，在玩一些Ar-duino小车时，更能体现PWM的好处。

这一节介绍结束了！同样的硬件连接，通过软件的变化，可以呈现出完全不同的效果，是不是觉得Arduino很酷炫！

项目六

# 灯光调节器

---

09



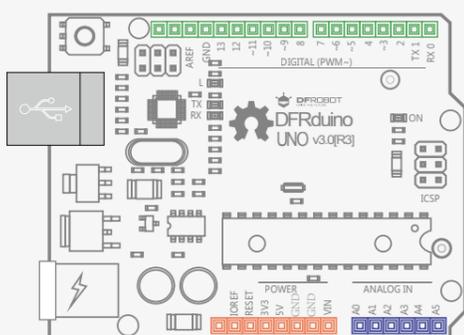
**DFROBOT**  
DRIVE THE FUTURE

# 项目六 灯光调节器

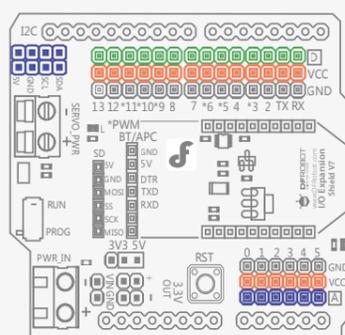
所谓灯光调节器，就是可以自由控制灯的亮度，我们这里通过一个模拟角度传感器来LED灯的亮度。随着旋转角度的变化，LED亮度也发生相应改变。角度越大，LED灯也就越亮，相反，角度越小，LED灯也就越暗。这里只是用了小小的LED来做演示效果，如果想运用到我们的生活之中的话，也是同样的原理。那就先做个小型的灯光调节器吧！

模拟角度传感器还能用到很多地方,比如我们后面会接触的舵机,可以通过这个传感器来控制转动角度,又或者以后有机会接触直流电机的小伙伴,可以尝试下用角度传感器来控制转速等等,用处很多!

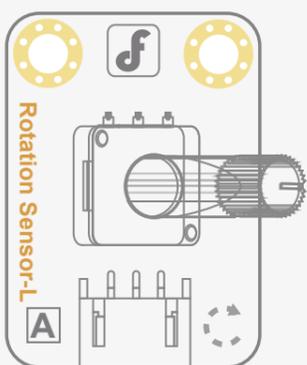
## 所需元件



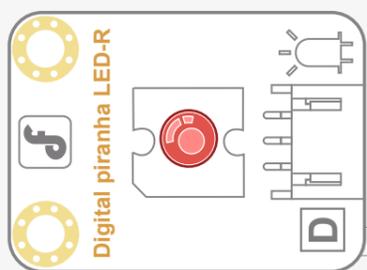
**x1**  
DFRduino UNO R3



**x1**  
IO Expansion Shield  
IO 传感器扩展板



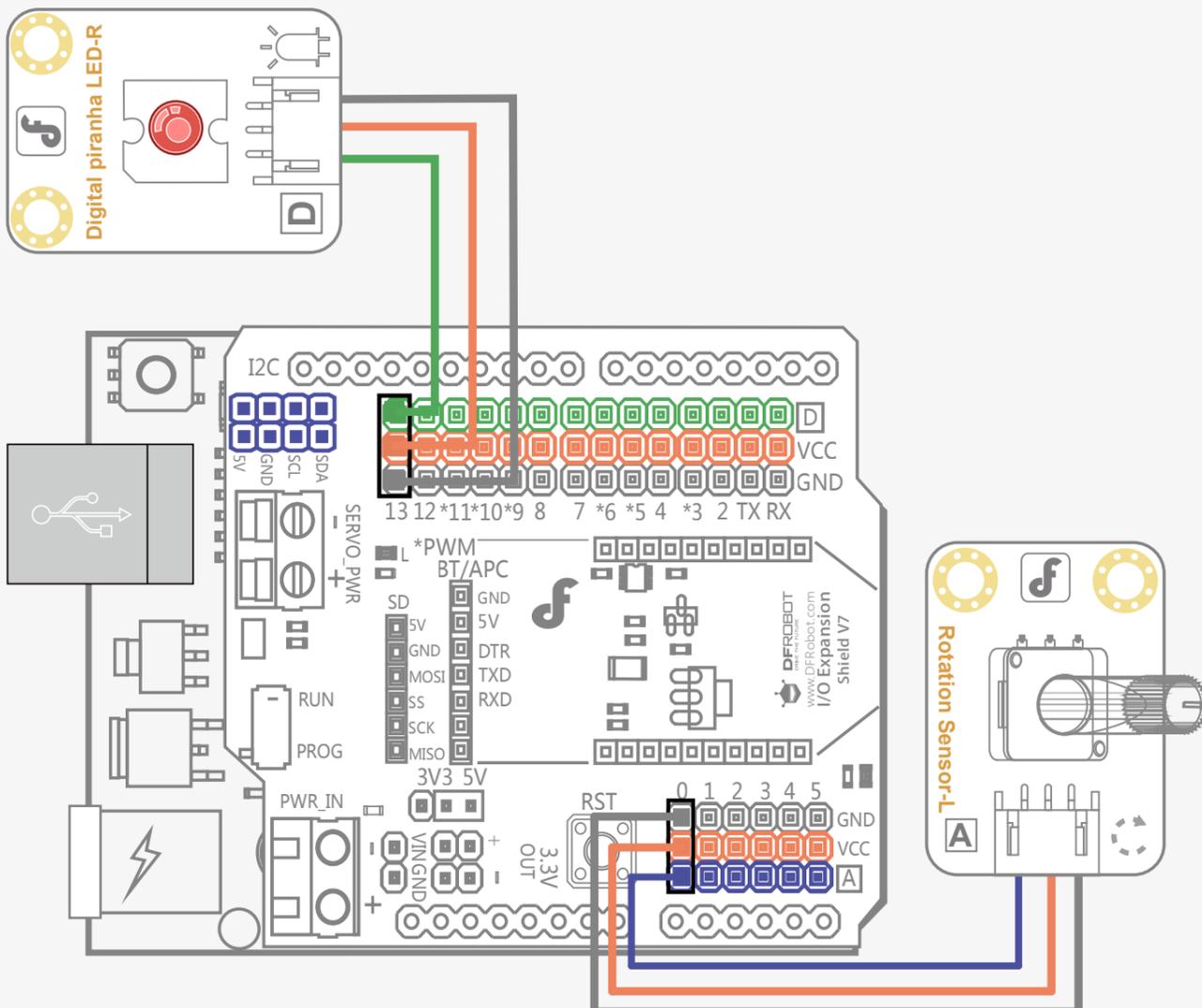
**x1**  
Analog Rotation Sensor  
模拟角度传感器



**x1**  
Digital piranha LED-R  
数字食人鱼红色LED发光模块

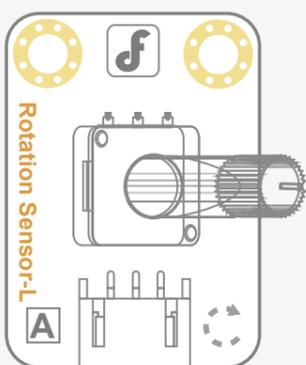
# 硬件连接

模拟角度传感器 连接模拟引脚0  
数字食人鱼红色LED发光模块 连接数字引脚10

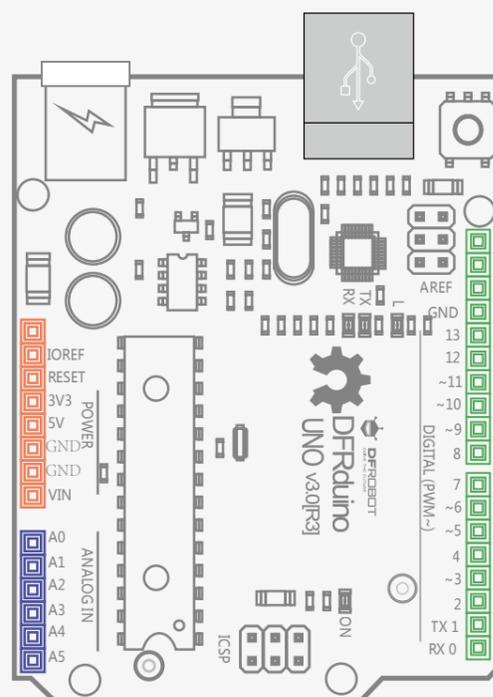


## 硬件分析 (模拟输入—模拟输出)

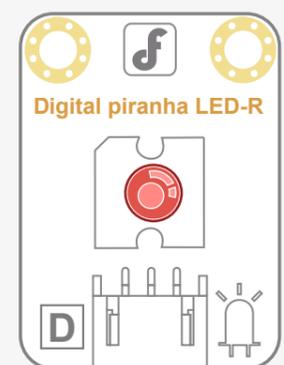
在呼吸灯一节，我们已经学会了如何用数字引脚的PWM口来做模拟输出。这一节将加入互动元素，通过模拟输入来控制模拟输出。



输入设备



控制设备



输出设备

## 输入代码

```
//项目六 —— 灯光调节器
int potPin = 0;           //电位器连接到模拟0
int ledPin = 10;         //LED连接到数字10

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

void loop() {
    int sensorValue = analogRead(potPin); //读取模拟口0的值
    //通过map()把0~1023的值转换为0~255
    int outputValue = map(sensorValue, 0, 1023, 0, 255);
    analogWrite(ledPin, outputValue);     //给LED写入对应值
    delay(2);
}
```

缓慢旋转电位器，仔细观察LED的亮度是否发生变化。

## 代码回顾

---

这里主要讲下map函数。

函数格式如下：

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

map函数的作用是将一个数从一个范围映射到另外一个范围。也就是说，会将 fromLow 到 fromHigh 之间的值映射到 toLow 在 toHigh 之间的值。

map函数参数含义：

**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

map的神奇之处还在于，两个范围中的“下限”可以比“上限”更大或者更小，因此map()函数可以用来翻转数值的范围，可以这么写：

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

这个函数同样可以处理负数，请看下面这个例子：

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

回到代码中，

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

我们是想将模拟口读到的0~1023的值，转换为PWM口的0~255。

# 项目七

# 芝麻开门

---

10

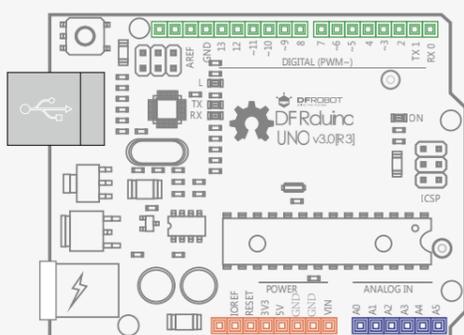


**DFROBOT**  
DRIVE THE FUTURE

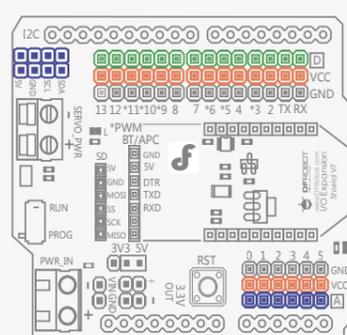
# 项目七 芝麻开门

所谓芝麻开门，就是这门不是轻易能打开的，存在着某些玄关需要你去破解。我们这里设计的这个门，是通过不断晃动手中的传感器才能开。有人会说，传感器我知道是那个数字震动传感器，那门呢？你没看错就是舵机，它能控制角度。所以，所谓的门的转动，是靠舵机来完成的。做个来看下效果就知道了~

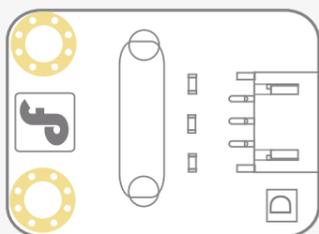
## 所需元件



**x1**  
DFRduino UNO R3



**x1**  
IO Expansion Shield  
IO 传感器扩展板 V7.1



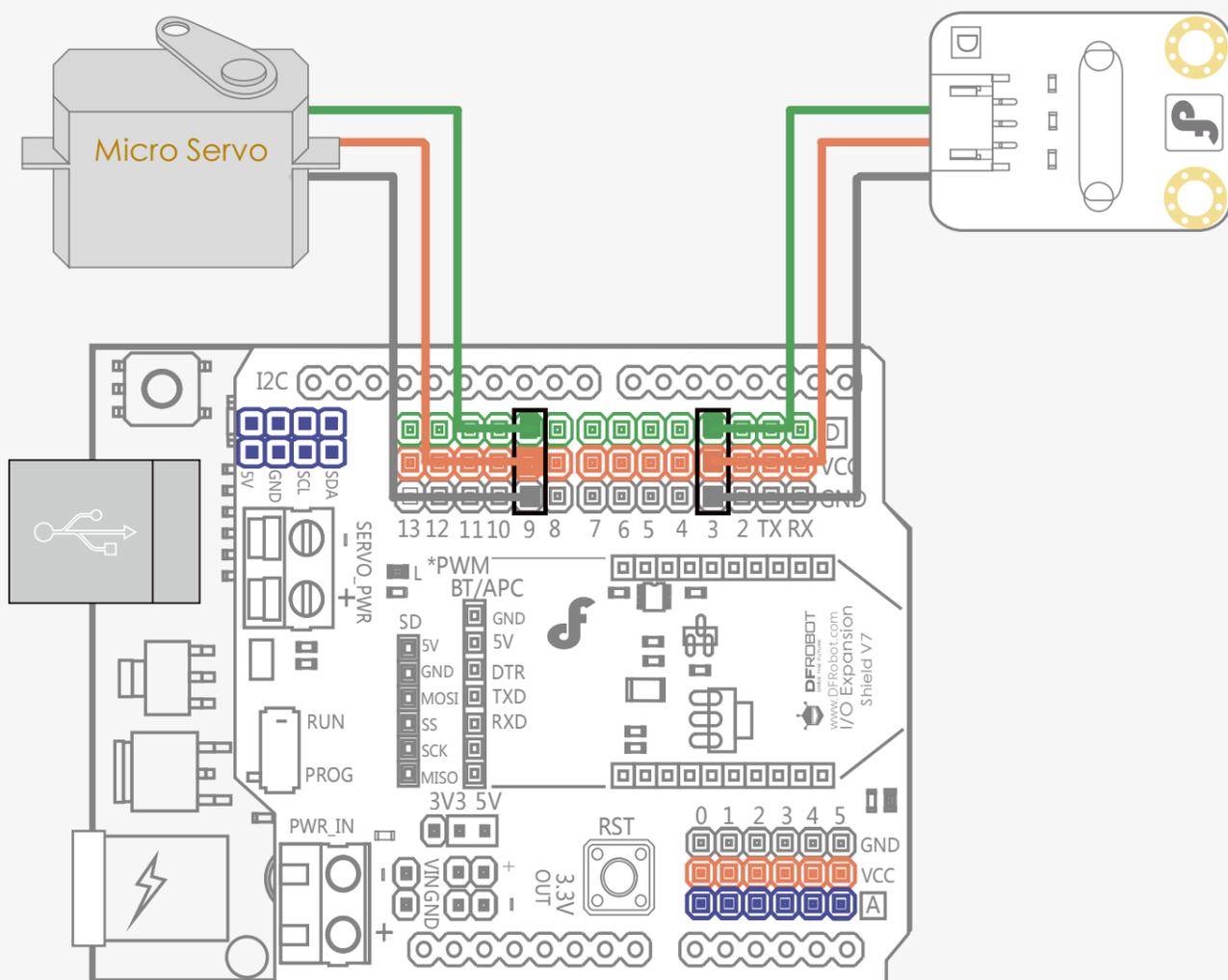
**x1**  
Digital Tilt Sensor  
数字钢球倾角传感器



**x1**  
5g micro servo  
TowerPro SG50舵机

# 硬件连接

TowerPro SG50 连接数字引脚9  
数字钢球倾角传感器 连接数字引脚3



## 输入代码

```
#include <Servo.h>

int sensorPin = 3;           //震动传感器 - Digital 3
Servo myservo;
int pos = 0;

void setup() {
  Serial.begin(9600);
  pinMode(sensorPin, INPUT);
  myservo.attach(9);        //舵机 - Digital 9
}

void loop() {
  int sensorState = digitalRead(sensorPin); //读取震动传感器的状态
  Serial.println(sensorState);
  if(!sensorState){        //一旦状态发生变化，舵机加2°，直到加到180°
    pos = pos + 2;
    if(pos >=180){
      pos = 180;
    }
    myservo.write(pos);    //写入舵机的角度
    Serial.println(pos);   //串口同时输出角度值
    delay(100);
  } else{                  //状态不发生变化，舵机减2°，直到减到0°
    pos = pos - 2;
    if(pos <=0){
      pos = 0;
    }
    myservo.write(pos);
    Serial.println(pos);
    delay(100);
  }
  delay(1);
}
```

不断的晃动震动传感器，可以看到舵机的角度会随之变大。停止晃动后，舵机角度又开始慢慢减小。好比一扇门慢慢打开，合上。

## 代码回顾

---

代码的开始先调用<Servo.h>库

```
#include <Servo.h>
```

这个库已经在Arduino IDE中了，可以打开Arduino-1.0.5/ libraries/ Servo/ Servo.h，这就是Servo库所在位置。

如果要在代码中用库中函数，是不能直接调用的，需要给库找个中介，让“他”建立代码和库中间的关系：

```
Servo myservo;
```

这里的myservo起到的就是这个作用。建立联系，之后调用库中的函数的话，就要按照下面这个模式进行：

```
myservo.函数名();
```

中间那“.”不要漏了！

如何定义舵机是接到那个引脚呢？用到就是attach()函数了。

```
attach(pin);
```

attach(pin)函数有一个参数——pin，任意一个数字引脚（不建议使用数字0,1）。我们这里选择数字引脚9。

```
myservo.attach(9);
```

知道了如何定义一个舵机之后，如何把对应的角度写进去呢？

```
write(pos);
```

pos就是写入的角度值。

# 项目八

# 夜光宝盒

---

11

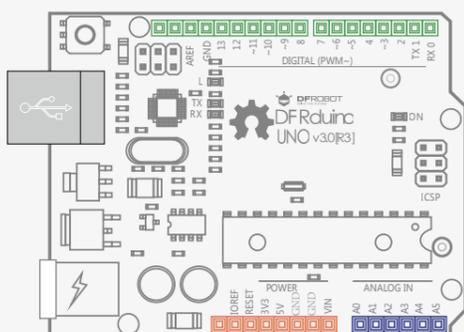


**DFROBOT**  
DRIVE THE FUTURE

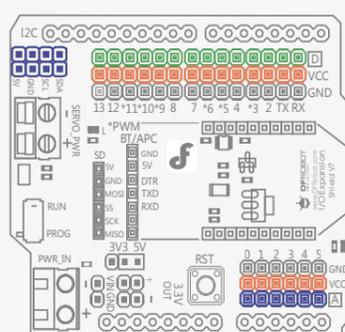
# 项目八 夜光宝盒

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！通过一个模拟环境光传感器，来检测环境光线强弱，随着亮度的不同，输出值不同。到了晚上的设定值，就转动舵机角度，LED同时慢慢变亮。

## 所需元件



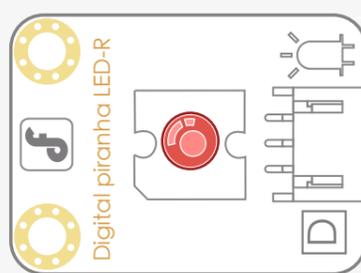
**x1**  
DFRduino UNO R3



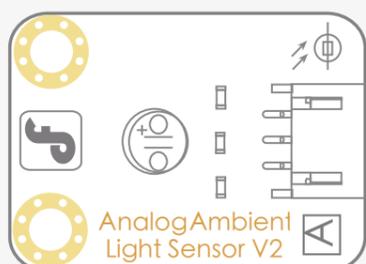
**x1**  
IO Expansion Shield  
IO 传感器扩展板



**x1**  
5g micro servo  
TowerPro SG50舵机



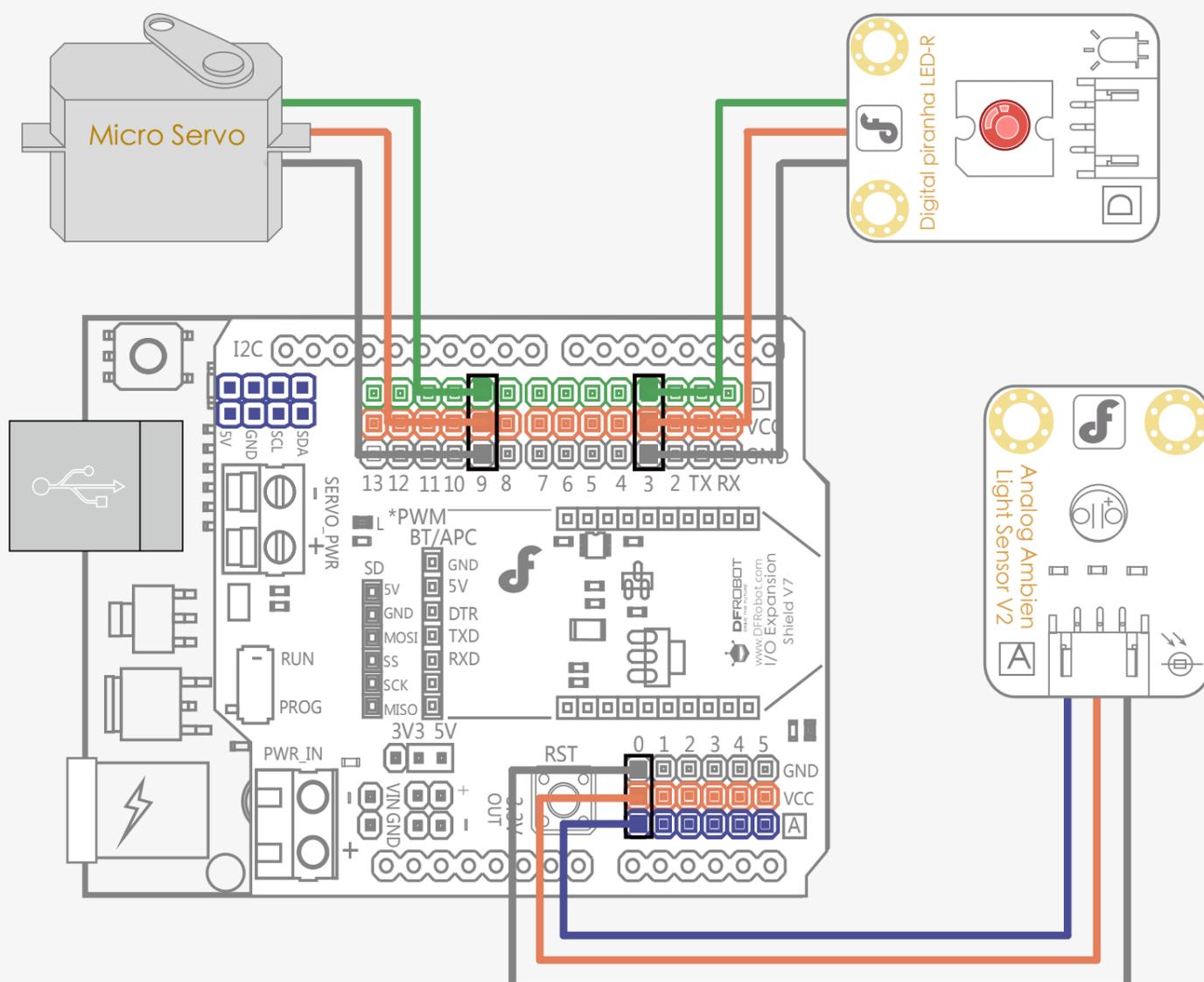
**x1**  
Digital piranha LED light  
数字食人鱼红色LED发光模块



**x1**  
Analog Ambient Light Sensor  
模拟环境光线传感器

## 硬件连接

TowerPro SG50 连接数字引脚9  
模拟环境光线传感器 连接模拟引脚0  
数字食人鱼红色LED发光模块 连接数字引脚3



## 代码输入

```
#include <Servo.h>
Servo myservo;
int LED = 3;           //设置LED灯为数字引脚3
int val = 0;          //val存储环境光传感器的值
int pos = 0;
int light = 0;

void setup(){
  pinMode(LED,OUTPUT); //LED为输出模式
  Serial.begin(9600);  //串口波特率设置为9600
  myservo.attach(9);   //舵机接到数字口9
  myservo.write(0);    //初始角度为0
}

void loop(){
  val = analogRead(0); // 读取传感器的值
  Serial.println(val); // 串口查看电压值的变化
  if(val<40){          // 一旦小于设定的值，增加角度
    pos = pos +2;
    if(pos >= 90){     //转到了90° 后，就保持90°
      pos = 90;
    }
    myservo.write(pos); //写入舵机的角度
    delay(100);
    light = map(pos,0,90,0,255); //随角度增大，LED亮度增大
    analogWrite(LED,light); //写入亮度值
  }else{
    pos = pos -2;     //减2°
    if(pos <= 0){
      pos = 0;        //减到0° 为止
    }
    myservo.write(pos); //写入舵机的角度
    delay(100);
    light = map(pos,0,90,0,255); //随角度减小，LED亮度减小
    analogWrite(LED,light); //写入亮度值
  }
}
```

---

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。

代码部分，注释已经非常清楚了，涉及函数在前几章也都了解过了，所以就多做说明了。

---

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](http://www.dfrobot.com.cn)