目录

项目零 初识 Arduino & Mind+

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创设的。

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鸣器吗?
- 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。

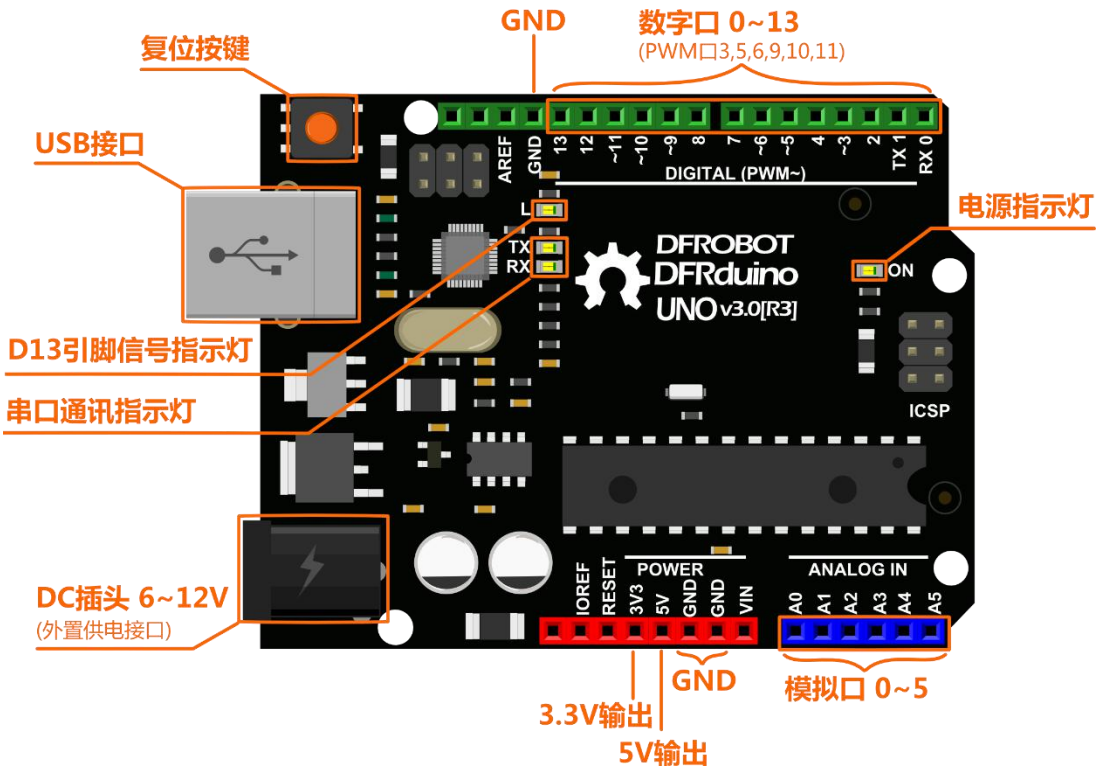
Arduino 名称的由来

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。

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，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。



初窥 Mind+的奥秘

Arduino 板子和电脑间，我们有 USB 线这一硬件构建了物理连接。但仅仅做到这一步就好比是买来了各式各样的硬件、组装好了电脑，但是没有软件无法使用这些硬件。那么要如何建立这两者之间信息层的连接，让我们开始玩转 Arduino 板呢?

答案便是 Mind+！它为两者架起了虚拟的桥梁，从而实现代码的烧录、串口的连接、实时数据流的传输等功能。

本教程的前期项目将引导你，来使用 Mind+进行轻松的图形化编程，有助于更好的理解程序的核心思想和实现步骤。熟悉各类指令后便可小试牛刀、排列组合设计独创的程序，并且逐渐尝试在 Mind+中自己敲打键盘输入代码！有了之前的基础，中后期的项目会脱离图形化编程，逐渐转为纯代码学习，让你在动手输入一行行的代码时，更深刻地感受它的魅力，让编程路上不再步履蹒跚的你，缓缓地放开图形化编程一路上搀扶着你的手，自由奔跑在编程的世界里！

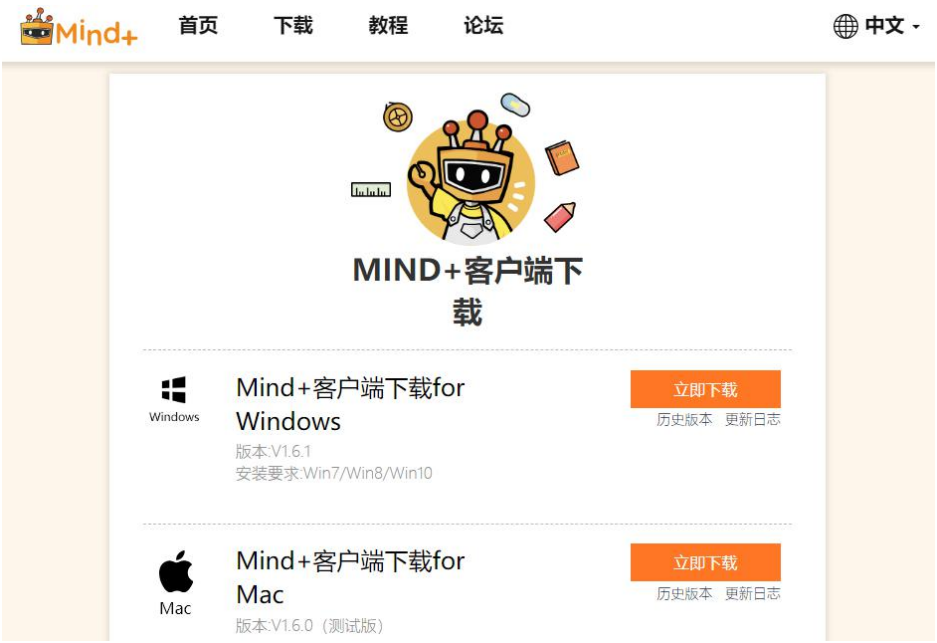
现在让我们快快熟悉 Mind+的世界吧！

初次使用

1.下载 Mind+（下载地址：http://mindplus.cc）

Mind+是一款基于 Scratch3.0 开发的青少年编程软件，支持 arduino、micro:bit、掌控板等各种开源硬件，只需要拖动图形化程序块即可完成编程，还可以使用 python/c/c++等高级编程语言，让大家轻松体验创造的乐趣。

- Mind+客户端下载



*如果在下载以及之后的安装和使用中遇到任何问题可以访问 Mind+的官方网址能够在常见问题和论坛中寻找解决方案，若搜索不到你可以在论坛发帖询问，技术支持会及时地来解决你的问题！http://mindplus.dfrobot.com.cn

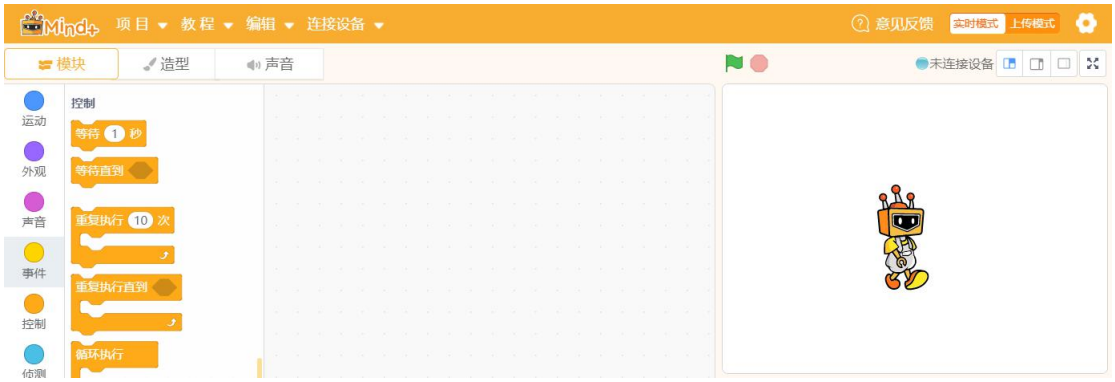
- 下载完成后双击安装：

2. 安装驱动

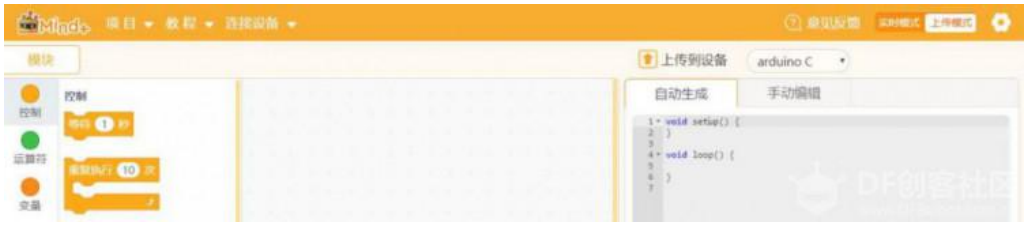
- 安装成功之后打开软件，点击“教程”——“视频教程”按钮打开教程，根据“安装驱动”教程提示进行驱动安装即可。

3. 切换“上传模式”（本系列教程均为“上传模式”下操作）

- 点击右上角“上传模式”按钮，等待切换



- 切换“上传模式”模式成功



4. Mind+界面介绍

- 下载安装成功之后让我们仔细看下 Mind+编程界面。

如果把整个软件比作一个舞台的话，那么不同区域的功能是什么呢?

首先看一下菜单栏：它是用来设置软件的区域，这里就是整个“舞台”的幕后啦，没有菜单栏的帮助，连上台表演的机会都没有。那么“舞台”的幕后都有什么呢?

“项目”菜单可以新建项目、打开项目、保存项目。

“教程”菜单在初步使用时可以在这里找到想要的教程和示例程序。

“连接设备”菜单能检测到连接的设备，并且可以选择连接或是断开设备。

“上传模式/实时模式”按钮切换程序执行的模式。

“设置”按钮用于设置软件主题、语言、学习基本案例，在线或加入交流群进行咨询。

指令区：这里是“舞台”的“道具”区，为了完成各种眼花缭乱的动作，需要很多不同的道具组合。在“扩展”里，可以选择更多额外的道具，支持各种硬件编程。

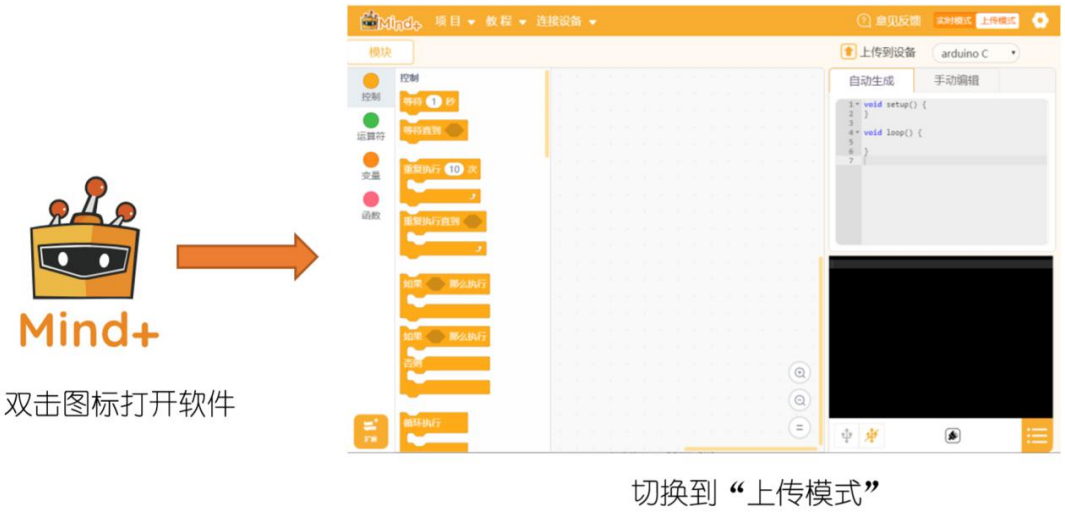
脚本区：这里就是“舞台表演”的核心啦，所有的“表演”都会按照“脚本区”的指令行动，这里是大家都能看得懂的图形化编程。拖拽指令区的指令就能在此编写程序。

代码查看区：如果想弄清楚“脚本区”图形化指令的代码究竟是啥，这里是个好地方。还能够在”手动编辑“中自己通过键盘输入代码。

串口区：想知道“表演”的效果如何，那必须要和“观众”互动啦。这里能显示下载状况，比如可以看到程序有没有成功下载，哪里出错了；程序运行状况；还能显示串口通信数据，也就是说，如果你的 Arduino UNO 板外接了一个声音传感器，那么你就可以看到在这里显示的声音数值大小。这里还有：串口开关、滚屏开关、清除输出、波特率设置、串口输入框、输出格式控制。

下载一个“闪烁（Blink）”程序

STEP1：双击桌上面的图标（如下图），打开 Mind+软件，将模式切换至 “上传模式”

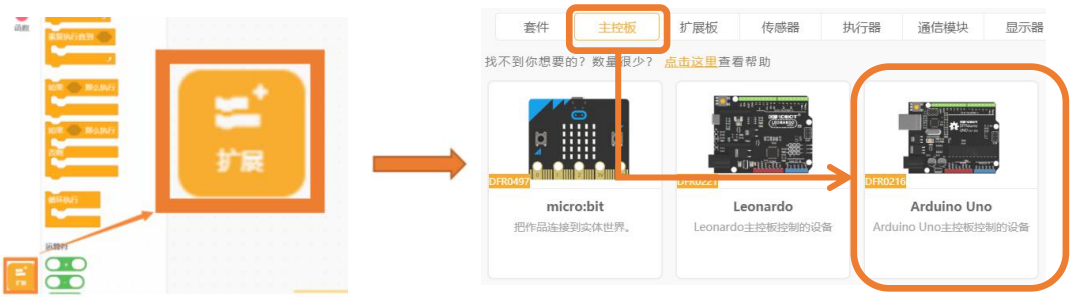


STEP2：用 USB 线将 Arduino 板和电脑连接，然后再点击“连接设备”——“COM7-UNO”

*“COM7”中的 7 会因为设备的关系而出现不同的数字，不影响使用。

*如果这里没有出现 COM 口，请确认 arduino 板电源灯点亮以及驱动安装完毕，若无法解决可联系我们寻求进一步帮助。

STEP3：点击左下角“扩展”，进入后选择主控板——Arduino UNO



点击后返回便能够看见已经加载了 Arduino UNO 模块。

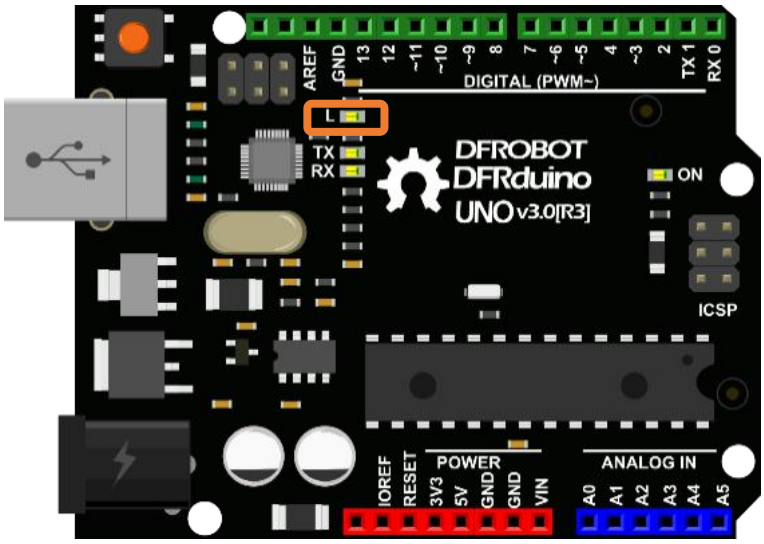
*不要忘记在每次打开软件后都要点击扩展，添加 Arduino UNO 库，否则会出现找不到指令的情况。

STEP4：开始载入程序，点击“教程”中的“示例程序”

单击载入第一个“闪烁”

Mind+已经预置好了程序，单击“上传到设备”，等待程序烧录完毕。

之后便能看见在 arduino 板上“L”旁的 LED 灯在闪烁了！



项目一 LED 闪烁

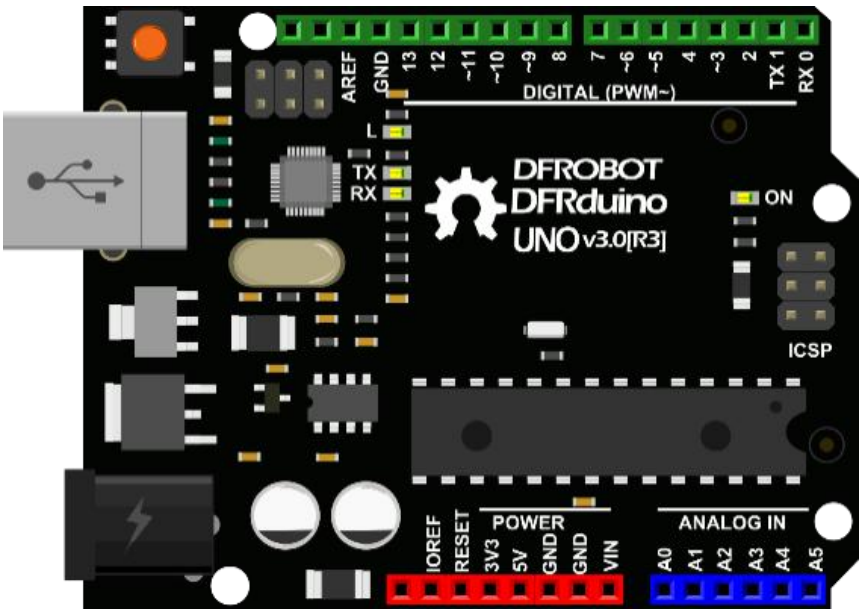
让我开们始吧!

从 LED 开启我们的 Arduino 之旅吧！你将学会像控制按钮输入一样控制 Arduino 的各种输出。在硬件方面，你将学习到有关 LED、按钮、和电阻的内容，包括上拉和下拉电阻的知识，这对于之后的项目非常重要。在这个过程中，你会接触 Mind+图形化编程，编程其实也没你想象的那么困难，对照图形化编程自动生成的 C 代码，你能学习基本的代码知识。让我们从一个最基本的项目，使用 Arduino 控制一个外部 LED 的闪烁。

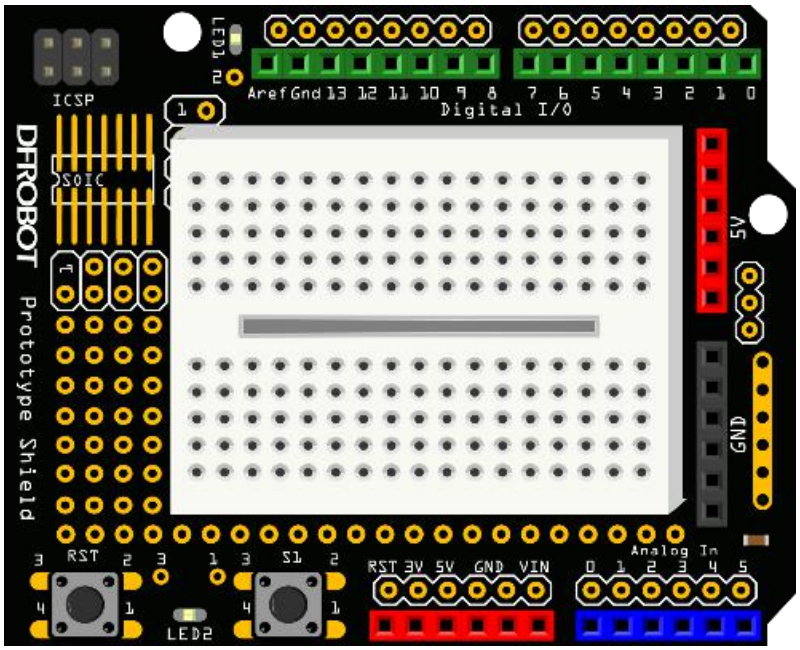
在第一个项目中，我们将重复使用之前的那个测试代码 Blink 程序。有所不同的是，这里我们需要外接一个 LED 到数字引脚，而不是使用焊在开发板上的 LED 13（也就是“L”灯）。便于我们能清晰的认识 LED 的工作原理及一些硬件电路的搭建。

所需元件

- 1× DFduino UNO R3（以及配套 USB 数据线）



- 1× Prototype Shield 原型扩展板+面包板



- 若干 彩色面包线
- 1× 5mm LED 灯 
- 1× 220 欧电阻 

*这里电阻仅为示意图，具体阻值参看包装袋标示。阻值可能根据你使用的 LED 的不同而不同，后面会说明如何计算这个阻值。

硬件连接

首先，从我们的套件中取出 Prototype shield 扩展板和面包板，将面包板背面的双面胶撕下，粘贴到 Prototype shield 扩展板上。再取出 UNO，把贴有面包板 Prototype shield 扩展板插到 UNO 上。取出所需元件，按照图 LED 闪烁 1-1 连接。

*在连接时需要注意图片中的扩展板和实际手中的扩展版可能存在一定的版本差异，接线要对照所用接口下的标号，而非依靠接口的相对位置。

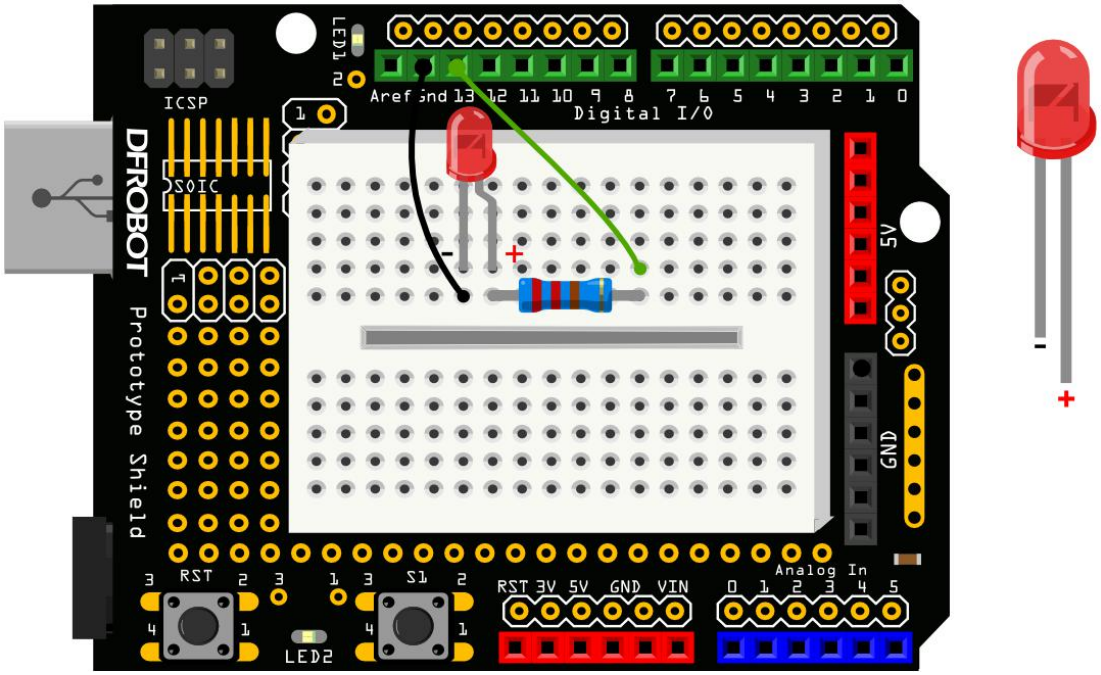


图 LED 闪烁 1-1

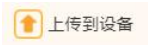
用绿色与黑色的杜邦线连接元件（在 DFRobot 的产品中有如下定义，绿色为数字口，蓝色为模拟口，红色为电源 VCC，黑色为 GND，白色可随意搭配），使用面包板上其他孔也没关系，只要元件和线的连接顺序与上图保持一致即可。

确保 LED 连接是否正确的，LED 长脚为+，即 VCC，短脚为-，即 GND，完成连接后，给 Arduino 接上 USB 数据线、供电，准备下载程序。

图形化编程

打开 Mind+，完成前一课中所学的加载扩展 Arduino UNO 库，并用 USB 线将主板和电脑相连，然后在连接设备复选框中选择主板并连接。之后将左侧指令区拖曳到脚本区。输入样例程序 1-1 所示程序。

样例程序 1-1：

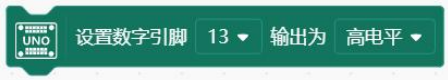
输入完毕后，点击 

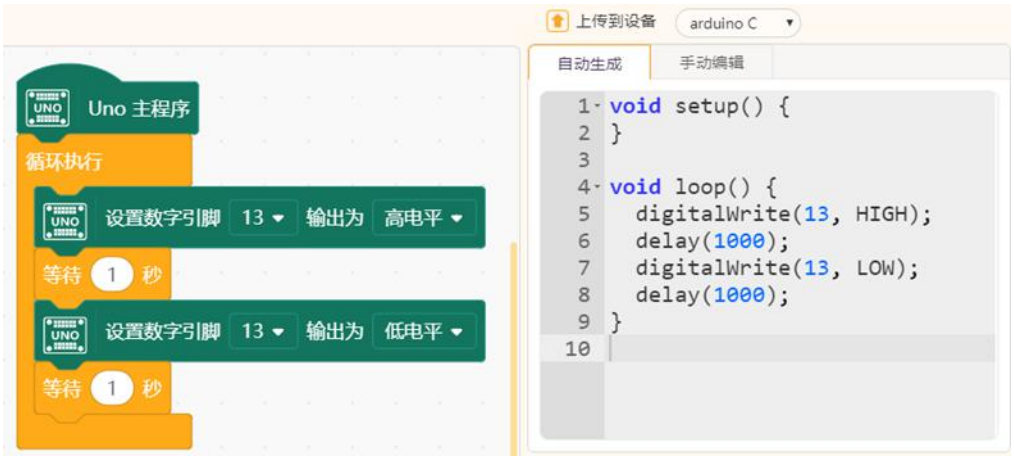
给 Arduino 下载程序。

运行结果：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。

现在让我们来学习一下程序中用到的指令，看看它们是如何工作的。

图形化指令学习

所属模块	指令	功能
Arduino		主程序指令，程序开始执行的地方，指令放在主程序下面才能起作用
控制		循环执行指令中的每条语句都逐次进行，直到最后，然后再从循环执行中的第一条语句再次开始，一直循环下去
Arduino		设置对应引脚为高/低电平，相当于将引脚电压设置为相应的值，HIGH（高电平）为 5V（3.3V 控制板上为 3.3V），LOW（低电平）为 0V
控制		延时等待（输入 0.5 即延时 0.5 秒，最小单位为 1 毫秒，即 0.001 秒）



通过图形化编程，已经完成了 LED 闪烁效果，根据代码区自动生成的 C 代码，让我们一起来学习一下代码语言。

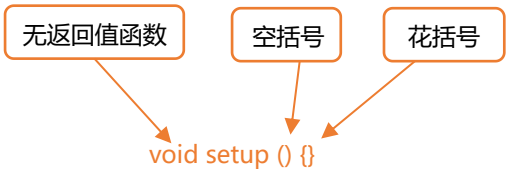
代码学习

本项目的 C 代码如下。接下来我们将依次学习每一行代码的意义

```
1.  void setup () {
2.  }
3.
4.  void loop() {
5.      digitalWrite (13, HIGH);
6.      delay (1000);
7.      digitalWrite (13, LOW);
8.      delay (1000);
9.  }
```

- void setup() {}

这是 setup()函数对应  指令。函数格式如下：



函数内部被花括号括起来的部分将会被依次执行，从“{”开始，“}”结束。两个符号之间的语句都属于这个函数。该括号内的函数通常起着初始化的作用，在 arduino 通电或者重启后，setup 函数只运行一次。

函数

函数通常为具有一个个功能的小模块，通过这些功能的整合，就组成了我们的整段代码，一个完整的功能实现。这些功能块也能被反复运用。这时，就体现函数的好处了。在程序运行过程中，有些功能可能会被重复使用，所以只需程序中调用一下函数名就可以了，无需重复编写。而 setup()和 loop()比较特殊，不能反复调用。

还有一个概念我们需要了解一下，就是函数的返回值，我们可以理解为是一种反馈。在函数中是如何体现有无返回值的呢？就是，函数的声明，比如“ void ”就是函数无返回值的信号，我们之后会经常用到。带返回值的函数，我们先不做说明了，有兴趣的可以去网上了解一下。

然而 setup 和 loop 函数比较特殊，一段代码中只能使用一次。

你是否对函数有了一个简单的概念了呢？不明白也没关系，在我们之后的项目还会涉及到的。

- void loop() {}

loop() 函数，对应 

指令。

* Arduino 程序必须包含 setup () 和 loop () 两个函数，否则不能正常工作。

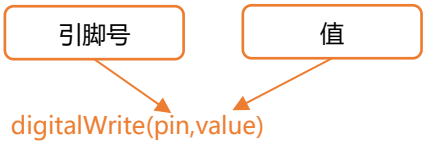
loop () 函数顾名思义，该函数在程序运行过程中不断的循环，loop () 函数中的每条语句都逐次进行，直到函数的最后，然后再从 loop 函数的第一条语句再次开始，三次、四次……一直这样循环下去，直到关闭 Arduino 或者按下重启按钮。

- digitalWrite(13, HIGH)

digitalWrite () 函数对应指令， 

作用是给引脚 13 一个高电平，点亮 LED。

函数格式如下：



- delay(1000)

delay()函数对应指令 

用于延时等待。注意 delay 函数的时间单位为毫秒，1000 毫秒也就是 1 秒。举一反三，如果我们需要延时 0.5 秒，指令对应的代码是什么呢？答案：delay（500）

- digitalWrite(13, LOW);

有了上面的引导，这句话是不是很容易理解了呢？这句话意思为，给引脚 13 一个 0V 的低电平，也就是熄灭 LED。

拓展练习

现在我们知道代码是如何运作的了，让我们来个小小的改动吧！让 LED 保持关闭 5 秒，然后快速闪烁一下（0.25 秒），就像汽车报警器上的 LED 指示灯那样。试着写一下！

答案：

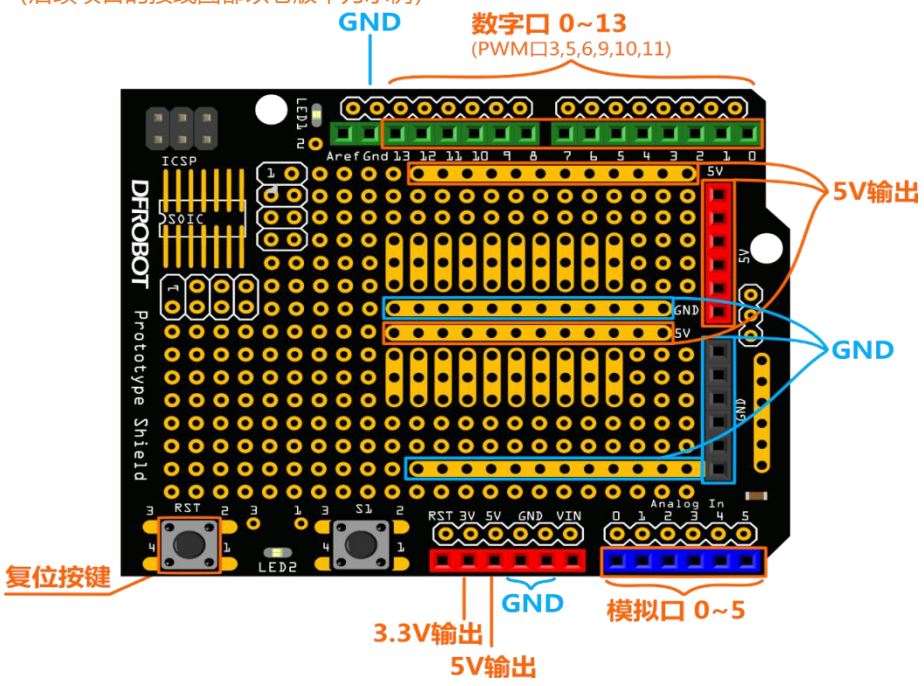
通过改变 LED 亮和暗的时间，可以产生不同的效果，亮暗交替迅速，则感觉动感，亮暗交替放缓，则感觉柔和。外面的灯光效果很多都是基于这样的原理。

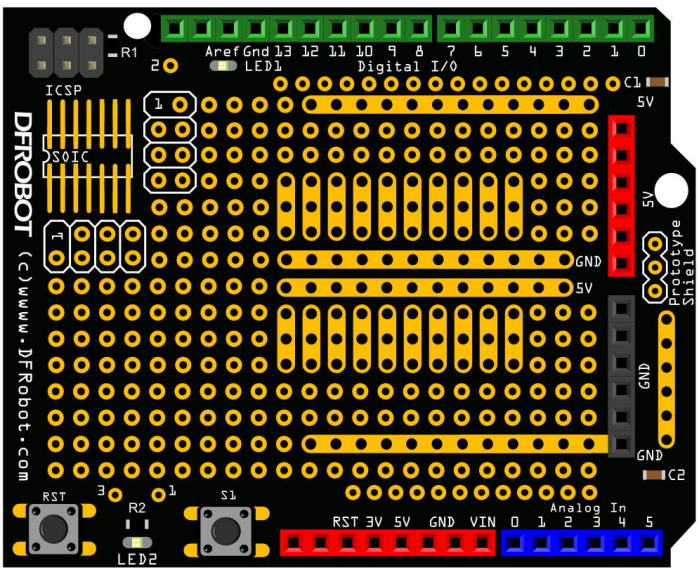
硬件回顾

Prototype Shield 原型扩展板

Arduino UNO 上面的端口资源是非常金贵。尤其是 5V 和 GND 的电源接口在板子上只有 2 到 3 个。因此在搭建多个器件时，需要用到多个 GND 或者 5V 接口，就没有足够的端口资源了。因此必须要一个端口扩展板来充分扩展 Arduino 的资源。

与 Arduino UNO 配合使用的 PrototypeShield 原型扩展板，用来搭建电路原型，可以直接在板子上焊接元件，也可以通过上面的迷你面包板连接电路。面包板与电路板之间通过双面胶连接。来稍微看下这块板子，数字口与模拟口与 UNO 是一一对应的。其次，下图标出的 5V 都是相通的，GND 也一样，都是相通的。（后续项目的接线图都以老版本为示例）

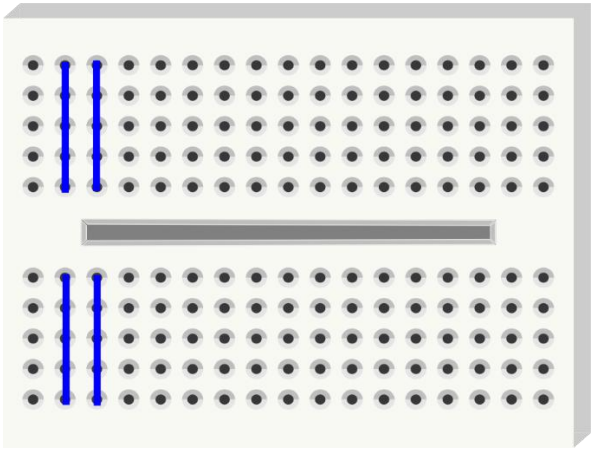




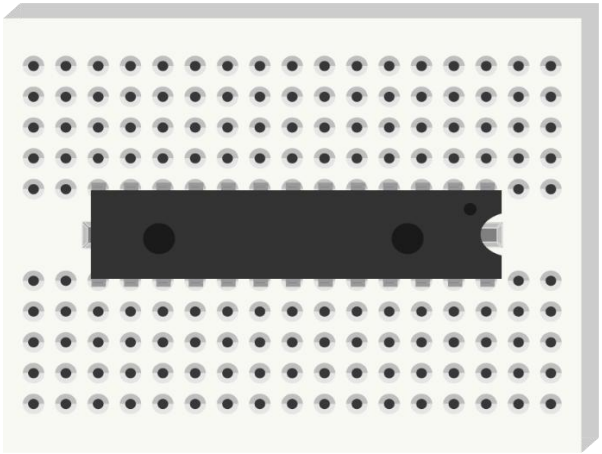
(新版本的 Prototype Shield 原型扩展板)

面包板

面包板是一种可重复使用的非焊接的元件，用于制作电子线路原型或者线路设计。简单的说，面包板是一种电子实验元件，表面是打孔的塑料，底部有金属条，可以实现插上即可导通，无需焊接的作用。面包板该怎么使用？其实很简单，就是把电子元件和跳线插到板子上的洞洞里，具体该怎么插，我们就要从面包板的内部结构上说了。



从上图我们可以看到，面包板分为上下两个部分，蓝线指出的纵向每 5 个孔是相通的。那会有人问，为什么上下两个部分不全通呢？其实面包板中间这个凹槽设计是有讲究的。凹槽两面孔间距刚好是 7.62mm，这个间距正好可以插入标准窄体的 DIP 引脚的 IC。



IC 插上后，因为引脚多，一般很难取下，硬来很容易弄弯引脚，这个槽刚好可以用镊子之类的东西将 IC 慢慢取下。

电阻

下一个要说的元件是电阻。电阻的单位是 Ω。电阻会对电流产生一定的阻力，引起它两端电压的下降。可以将电阻想象成一个水管，它比连接它的管子细一点，当水（电流）流入电阻，因管子变细，水流（电流）虽然从另一端出来，但水流减小了。电阻也是一样的道理，所以电阻可以用来给其他元件减流或减压。

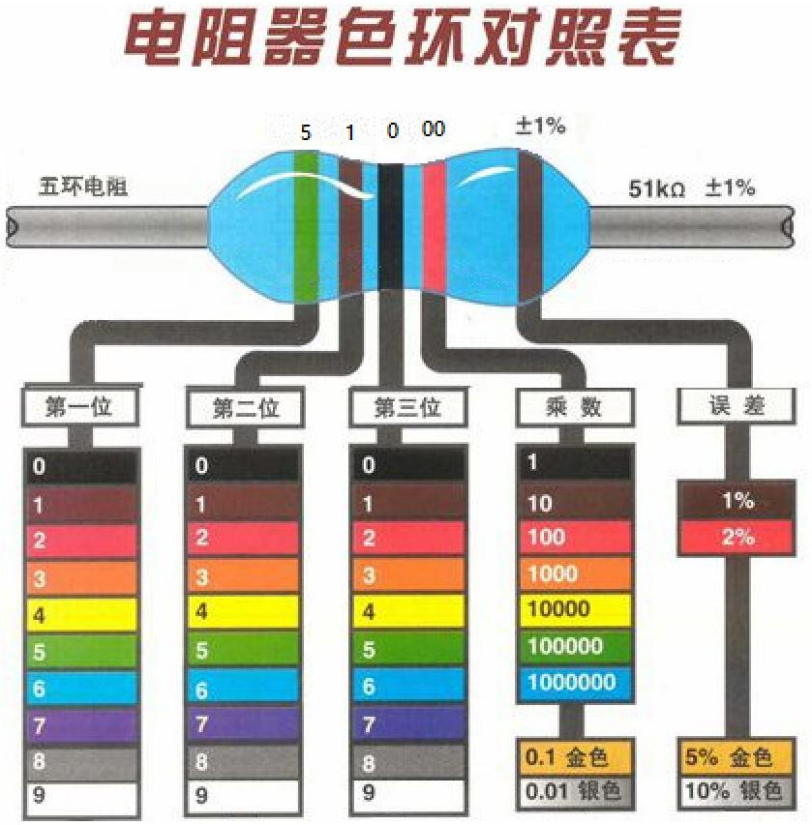
电阻有很多用处，对应名称也不同，上拉电阻，下拉电阻，限流电阻等。我们这里用作限流电阻。在这个例子里，数字引脚 10 输出电压为 5V，输入电流为 40mA（毫安）直流电。普通的 LED 需要 2V 的电压和 35mA 左右的电流。因此如果想以 LED 的最大亮度点亮它，需要一个电阻将电压从 5V 降到 2V，电流从 40mA 减到 35mA。这个电阻起限流的作用。

如果不连电阻会怎样呢？流过 LED 的电流过过大(可以理解为水流过大，水管爆破了！)，会使 LED 烧掉，就会看到一缕青烟并伴随着糊味儿~

这里具体对电阻值选取的计算就不做说明了，只要知道在接 LED 时需要用到一个 100Ω左右的电阻就可以了。大一点也没关系，但不能小于 100Ω。如果电阻值选的过大的话，LED 不会有什么影响，就是会显的比较暗。很容易理解，电阻越大，减流或减压效果更明显了。LED 随电流减小而变暗。

电阻色环读值

我们元器件的包装袋上已经明确标明了各个元件的名称。但不排除有时候不小心标签掉了，可是手头又没有可以测量的工具，那该怎么办呢？有个方法就是从电阻上的色环来读取电阻值。我们这里就不做详细说明了。感兴趣的同学可以参照下图读读看阻值。

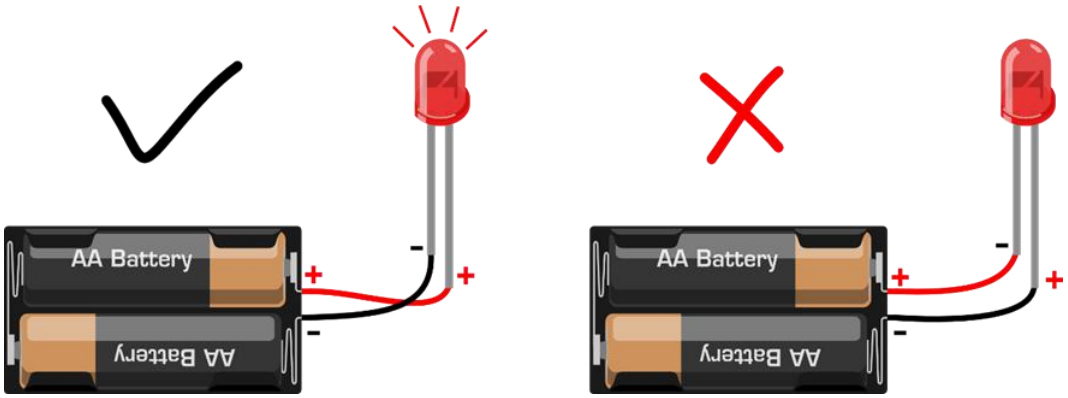


LED

最后要说就是 LED,标准的发光二极管，是二极管中的一种。二极管是一种只允许电流从一个方向流进的电子器件。它就像一个水流系统的中的阀门，但是只允许一个方向通过。如果电流试图改变流动方向，那么二极管就将阻止它这么干。所以，二极管在电路中的作用通常是用来防止电路中意外地将电源与地连接，避免造成损坏其他元件。

LED 也是一种二极管，会发光的二极管。LED 能发出不同颜色和亮度的光线，包括光谱中的紫外线和红外线。（比如我们经常使用的各类遥控器上面的 LED 也是其中一种，与普通的发光二极管长的一样，只是发出的光我们人眼看不到，我们也称之为红外发射管。）

如果仔细观察 LED,你会注意到，LED 引脚长度不同，长引脚为+，短引脚为-。那如果正负接反会怎么样呢？下面这张图就说明问题了，接反就不亮了呗。下图是不是还缺个电阻呀，细心的你发现了吗？



在我们的套件中，还有一种 LED，是 4 个脚的，不要以为我们发错了噢。这个 LED 功能可大着呢，它能呈现不同颜色，也称之为 RGB LED。我们都知道红色、绿色、蓝色是三原色，通过这三种颜色的暗弱变换的组合可以呈现出任何你想要的颜色。把三种颜色放在同一个外壳里就能达到这样的效果。在我们之后的项目中还会介绍到。

现在你知道了各元件的功能及整个项目中软硬件是如何工作的，让我们尝试做其他好玩儿的东西吧！

项目二 S.O.S 求救信号器

本项目将继续使用项目一的搭建的电路，但我们这里将改变一下代码，就能让我们的 LED 变为 S.O.S 求救信号了。这是国际摩尔斯码求救信号。摩尔斯码是一种字符编码，英文的每个字母，都是由横杠和点不同的组合而成。这样的好处是，使用简单的两种状态，就能来传递所有的字母和数字，非常的简便！不得不佩服前人的聪明吧！

我们正好可以通过 LED 开关两种状态来拼出一个个字母。通过长闪烁和短闪烁来表示横杠和点。这个项目中，我们就拼写 S.O.S 这三个字母。

通过查阅摩尔斯码表，我们可以知道，字母“S”用三个点表示，我们这里用短闪烁替代，字母“O”则用三个横杠表示，用长闪烁替代。

图形化编程

在输入指令前我们先来理解样例程序 2-1，先不要急着输入这段代码，只是看一下。

样例程序 2-1：

这一编法正确无疑，然而是不是觉得有点繁琐呢？如果有个 100 个，难不成还重复 100 遍吗？有没有更好的书写程序的方法呢？想必发明编程的人也考虑到这个问题了，所以我们使用一个新的模块来解决这一问题。

连接好主板和电脑，打开 Mind+载入扩展的 Arduino UNO 库，输入样例程序 2-2 所示程序。

样例程序 2-2

输入完毕后，确认正确后，点击 上传到设备 下载代码到 Arduino 中，如果一切顺利的话，我们将看到 LED 闪烁出摩尔斯码 S.O.S 信号，等待 0.5 秒。重复闪烁。给 Arduino 外接电池，整个装到防水的盒子里，就可以用来发 S.O.S 信号了。S.O.S 通常用于航海或者登山。

图形化指令学习

所属模块	指令	功能
控制		可指定次数的循环指令：将指令中包含的程序自下而上循环执行指定次数。

代码学习

本项目的 C 代码如下。接下来我们将依次学习每一行代码的意义

```
1.  // 主程序开始
2.  void setup() {
3.
4.  }
5.  void loop() {
6.      for (int index = 0; index < 3; index++) {
7.          digitalWrite(13, HIGH);
8.          delay(150);
9.          digitalWrite(13, LOW);
10.         delay(100);
11.     }
12.     delay(100);
13.     for (int index = 0; index < 3; index++) {
14.         digitalWrite(13, HIGH);
15.         delay(450);
16.         digitalWrite(13, LOW);
17.         delay(100);
18.     }
19.     delay(100);
20.     for (int index = 0; index < 3; index++) {
21.         digitalWrite(13, HIGH);
22.         delay(150);
23.         digitalWrite(13, LOW);
24.         delay(100);
25.     }
26.     delay(500);
27. }
```

当程序运行后我们可以看到，灯闪了 3 次而不是只闪了 1 次。我们再来看一下，在 loop 主函数中包含了三个独立的代码段，每一段的开头都能发现 for 的字样，这就是我们用来解决重复执行相同代码的好帮手。正式因为使用了 for 循环才能产生这样的效果。

for 循环

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。

来看下我们程序中的 for 循环：

```
1.  for(int index=0;index<3;index++){
2.          ……
3.  }
```

第一步：初始化变量 index=0。

第二步：判断 index 是否小于 3。

第三步：判断第二步成立，for 循环中执行 LED 亮与灭。

第四步：index 自加，变为 1。

（index++这句话表示把 index 的值增加 1，等同于写成 index=index+1，也就是把 index 当前的值变为 index+1，再赋给 index 一遍。0 变为 1，第二轮循环则 1 变 2。）

第五步：回到第二步，此时 index=1，判断是否小于 3。

第六步：重复第三步。

……

直到 index 循环到 3 时，判断 index<3 不成立，自动跳出 for 循环，程序继续往下走。

我们这里需要它循环 3 次，所以设置为 index<3。从 0 开始计算，0 到 2，循环了 3 次。那如果要循环 100 次的话呢？答案：for(int index=0;index<100;index++){}

必须说明的，花括号必须成对出现，如有遗留编译器编译时将不通过。有个小技巧大家可以学一下，在开始写花括号的时候，就先把“{” “}”都写上，之后再在两个括号之间输入代码，这样就不会出现写到最后括号对应不上的情况。在本章的课后作业中，我们就要开始尝试自己敲击键盘，在手动编辑区输入代码。这些小细节还需要大家牢记在心。

我们在写一些判断语句的时候会经常用到一些比较运算符，比如大于，小于等等。下面就说下常用的比较运算符。

比较运算符

“<”称为比较运算符。比较运算符在代码中是用作判断的，比较两个值。我们常用的比较运算符有：

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

特别要说明一下，在之后手动编写代码的时候，等于必须是两个等号。还有像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。

现在知道 for 循环是如何运作吧！我们代码中有 3 个 for 循环：第一个 for 循环 3 次，短闪烁 3 次，代表输出 3 个点，也就是字母“S”。第二个 for 循环同样循环 3 次，长闪烁 3 次，代码输出 3 个横杠，也就是字母“0”。第三个 for 循环又来输出个“S”。

在每 for 循环之间有个小延时 100 毫秒，是 S.O.S 字母之间有个清晰的停顿说明。最后，在回到主函数 loop 重新执行一遍之前，有个相对较长的延时，为 0.5 秒。

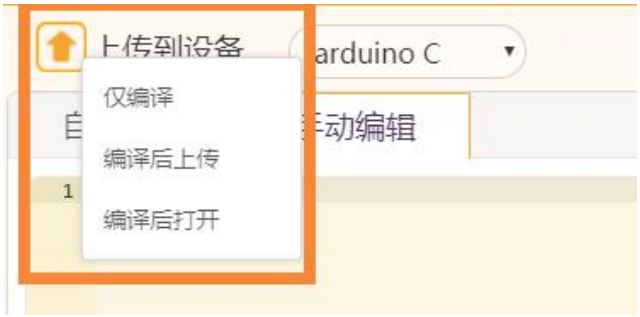
好了，我们 S.O.S 信号源项目就算告一个段落了。有所收获吗？

课后作业

学习了两个项目的基础，让我们来自己动手输入一下本项目“代码学习”中的一行行语句吧！

点击“手动编辑”即可通过键盘输入代码！

输入完代码后，右键点击“上传到设备”旁的上箭头，通过“仅编译”检验代码是否存在格式等问题



*进行图形化编程来修改程序的时候，只会更改“自动生成”中的代码。若此时对话框处于“手动编辑”，点击“上传到设备”，并不能够将图形化编程转化的代码烧录到板中！需要我们再点击“自动生成”进行切换，再上传，才能够更新板子中的程序。

C 语言编程就好比写字，字迹要求横平竖直、排版工整、美观隽秀的同时还要保证一定的书写速度。风格良好的代码亦是如此，语句格式、缩进对齐与输入代码的效率，一并需要。这两者都得积累平时一点一滴的水磨工夫，才能养成良好的习惯，写代码时更得心应手，为项目六及其之后的纯代码学习夯实基础！

*仅编译后出现报错的可能原因（1）：

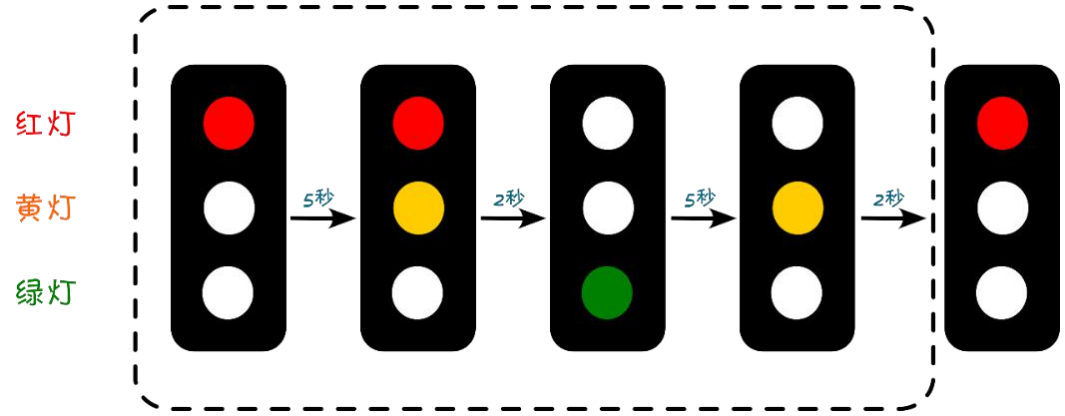
```
1  void setup(){
2    digitalWrite(1, HIGH)
3  }
4
5  void loop(){
6  }
```

```
compile error.
Error: Command failed: "D:\Software-Install\MIND+1.6.0\arduino\hardware\tools\avr\bin\avr-g++" -c -g -Os
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp: In function 'void setup()':
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp:4:1: error: expected ';' before '}' token
 }
 ^
```

出现这一问题的原因是在某一行语句指令的末端，没有加上；或者输入的是中文输入法中的分号

此外再用 Mind+做个课后习题：交通信号灯吧~下图是整个一个运行过程，虚线框的是程序循环的部分。

提示：以上我们是只点亮的一个 LED 灯，现在需要点亮三个 LED 灯。电路连接的原理是和一个灯相同，程序中需要改变的是用三个数字口来分别控制 3 个 LED 灯。自己动手试一下吧！



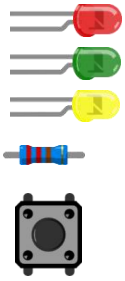
项目三　互动交通信号灯

有没有试着做上面那个课后作业呢？做出来的话，说明你已经基本掌握上面所学的东西了，如果不会也没关系，我相信，看完这个章节，前面那个问题就不攻自破了！我们这回就基于上面这个交通灯来进行一个拓展，增加一种行人按键请求通过马路的功能。当按钮被按下时，Arduino 会自动反应，改变交通灯的状态，让车停下，允许行人通过。

这个项目中，我们开始要实现 Arduino 的互动了，也会在代码学习中掌握如何创建自己的函数。这次的代码相对长一点，耐下心来，等看完这一章，相信你能收获颇丰！

所需元件

- 2×　5mm LED 灯
- 2×　5mm LED 灯
- 1×　5mm LED 灯
- 6×　220 欧电阻
- 1×　按钮



*我们之后在所需元件中将不再重复罗列以下三样，UNO、扩展板+面包板、面包线。但是，每次都还是需要用到的。

这里 5 个 LED 灯，为什么会用到了 6 个电阻呢？我们知道 5 个电阻是 LED 的限流电阻。还有一个电阻是给按钮的，它叫做下拉电阻（我们后面会解释）。

硬件连接

按图 3-1 的连线图连接你的电路。特别要注意的是，这次连线比较多，注意不要插错。下图中，面包板上标出淡绿色的不是跳线，只是为了说明纵向的孔导通，避免你插错。给 Arduino 上电前认真检查你的接线是否正确。在连线时，保持电源是断开的状态，也就是没有插 USB 线。

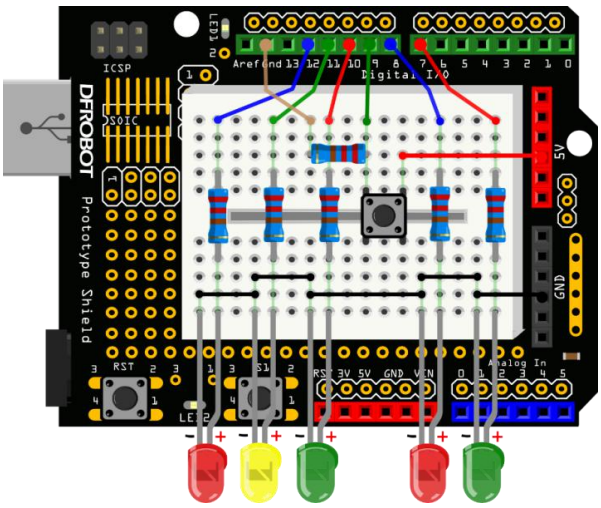


图 3-1　互动交通信号灯连线图

图形化编程

在本项目中，需要用到**变量**和**函数**，因此我们先来学习一下如何在 Mind+中新建变量以及定义函数。

变量的新建

STEP1：点击“新建数字类型变量”。

STEP2：输入合适的变量名。给变量命名时，可以用字母、数字、下划线开头，Mind+里面特别开发了中文变量，输入中文时在自动生成的代码中，会自动转换成汉语拼音字母。

STEP3：新变量 变量 test 已经建立好，以备之后使用。

*因为 Mind+为我们的变量加了前缀以防冲突，所以在自动生成代码中的变量名和自己起的会不同，加上了 mind_n_ 的前缀。

定义新函数

STEP1：点击“自定义模块”。

STEP2：输入合适的函数名。命名规则等同变量，一样可以输入汉字，Mind+会进行转化。

STEP3：此时在指令区出现了 

，我们可以在他的下方进行编程，定义此函数的内容。

STEP4：在程序中调用函数，将 

拖入到程序中，即实现了函数的调用。

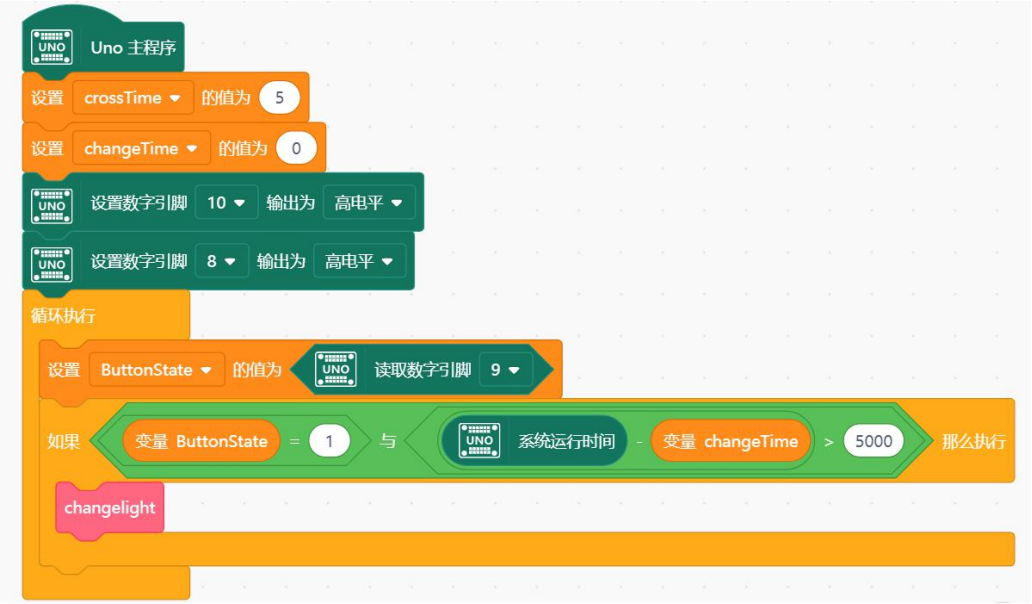
样例程序的输入

连接主板和电脑，打开 Mind+载入扩展库，输入样例程序 3-1 所示程序。

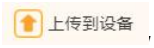
需要建立三个新变量，你有发现是哪三个吗？如在编写的时候遇到困难，可以先行阅读指令学习，了解本项目中所使用的新指令。本项目中涉及的新指令较多，需要静下心来细致地消化和吸收。

样例程序 3-1：


定义 changelight
设置数字引脚 10 输出为 低电平
设置数字引脚 11 输出为 高电平
等待 2 秒
设置数字引脚 11 输出为 低电平
设置数字引脚 12 输出为 高电平
等待 1 秒
设置数字引脚 8 输出为 低电平
设置数字引脚 7 输出为 高电平
等待 变量 crossTime 秒
重复执行 10 次
设置数字引脚 7 输出为 高电平
等待 0.25 秒
设置数字引脚 7 输出为 低电平
等待 0.25 秒
设置数字引脚 8 输出为 高电平
等待 0.5 秒
设置数字引脚 12 输出为 低电平
设置数字引脚 11 输出为 高电平
等待 1 秒
设置数字引脚 11 输出为 低电平
设置数字引脚 10 输出为 高电平
设置 changeTime 的值为 系统运行时间



输入完毕后，检查是否处于 

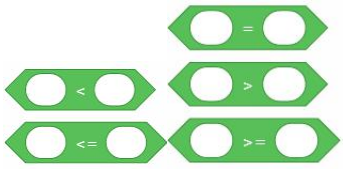
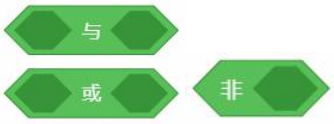
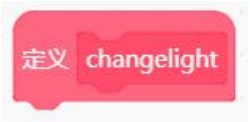
点击 

，给 Arduino 下载程序。

下载完成后，可以尝试按下按钮。看看是什么样的效果？我们可以看到整个变化过程是这样的——开始时，汽车灯为绿灯，行人灯为红灯，代表车行人停。一旦行人，也就是你，按下按钮，请求过马路，那么汽车灯由绿变黄、变红，行人灯就开始由红变绿。在行人通行的过程中，设置了一个过马路的时间 crossTime，一旦到点，行人绿灯开始闪烁，提醒行人快速过马路。闪烁完毕，最终，又回到了开始的状态，汽车灯为绿灯，行人灯为红灯。

图形化指令学习

所属模块	指令	功能
变量		存放可以变化的值，右键点击可以切换其他变量。
变量		可以给变量存入不同的数值，一般将这个过程叫做变量赋值。 点击指令中的变量名，从弹出的下拉窗口中可以选择不同的变量，重新给变量命名或者删除变量。
Arduino		读取数字引脚指令，读取指定引脚收到的值。得到的值为 0 或 1。可以赋值给变量或者作为判断条件。

控制		条件判断指令，用于判断六边形空框内的条件是否成立。 条件成立，则执行指令中包含的程序；条件不成立，则跳过该指令，执行后面的程序。
运算符		算数运算符：加、减、乘、除。 在椭圆形框内可以填入圆头或尖头模块例如变量等，亦可直接输入数值。
运算符		关系运算符：小于、小于等于、等于、大于、大于等于。 在框中放入对应形状的指令或者直接输入数值并进行判断条件是否成立，若成立反馈值为 1，不符成立为零。
运算符		逻辑运算符：与、或、非。 进行逻辑运算的指令，拖入对应形状的指令即可运行。具体解析在代码学习中给出。
Arduino		系统运行时间指令，该指令的效果等同一个变量，记录了从系统通电或重启后所经过的时间。
函数		函数定义指令。 建立新函数：通过点击 自定义模块 ，选择输入的变量类型及个数，并给函数起一个合适的名字。 建立后就可在此指令后编写自己所需要的程序。
函数		函数调用指令。 定义完新函数后，通过该指令来调用。

乍一看指令似乎很复杂，其实理清一下思路并不难，代码学习中将带着你一步步理解每一条指令起到了什么作用，相信你一定会很快理解整个程序是如何运行的了！

代码学习

```
1.  // 动态变量
2.  volatile float mind_n_crossTime, mind_n_changeTime, mind_n_ButtonState;
3.  // 函数声明
4.  void DF_changelight();
5.
```

```
6.
7.  // 主程序开始
8.  void setup() {
9.      mind_n_crossTime = 5;
10.     mind_n_changeTime = 0;
11.     digitalWrite(10, HIGH);
12.     digitalWrite(8, HIGH);
13. }
14. void loop() {
15.     mind_n_ButtonState = digitalRead(9);
16.     if (((mind_n_ButtonState==1) && ((millis()-mind_n_changeTime)>5000))) {
17.         DF_changelight();
18.     }
19. }
20. // 自定义函数
21. void DF_changelight() {
22.     digitalWrite(10, LOW);
23.     digitalWrite(11, HIGH);
24.     delay(2000);
25.     digitalWrite(11, LOW);
26.     digitalWrite(12, HIGH);
27.     delay(1000);
28.     digitalWrite(8, LOW);
29.     digitalWrite(7, HIGH);
30.     delay(mind_n_crossTime * 1000);
31.     for (int index = 0; index < 10; index++) {
32.         digitalWrite(7, HIGH);
33.         delay(250);
34.         digitalWrite(7, LOW);
35.         delay(250);
36.     }
37.     digitalWrite(8, HIGH);
38.     delay(500);
39.     digitalWrite(12, LOW);
40.     digitalWrite(11, HIGH);
41.     delay(1000);
42.     digitalWrite(11, LOW);
43.     digitalWrite(10, HIGH);
44.     mind_n_changeTime = millis();
45. }
```

我们来看第一行代码

```
1.  // 动态变量
```

这是代码中的说明文字，可以叫做注释。

注释

这里的注释是以”//”开始，这个符号所在行之后的文字将不被编译器编译。注释在代码中是非常有用的，它可以帮助你理解代码，如果项目比较复杂，自然而然，代码也会随之非常的长，而此时注释就会发挥很大作用，可以快速帮你回忆起这段代码的功能。同样，当把你的代码分享给别人的时候，别人也会很快理解你的代码。

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。

例如以下文字：

```
/* 在这两个符号之间的文字，

都将被注释掉，编译器自动不进行编译，

注释掉的文字将会呈现灰色 */
```

Mind+将自动把注释的文字颜色变为灰色。

变量的内容较多，我们先来看第三、第四行

```
3.  // 函数声明
4.  void DF_changelight();
```

函数声明

本课图形化编程时，在使用函数前，我们需要点击按钮，新建并命名一个新的函数。在代码编程时也是如此，需要在定义、调用函数前务必先声明！否则程序中的这一函数都无法执行。本课程中大多使用没有输入、输出的函数。它的申明格式如下：

```
void FunctionName ();
```

其他包含输入、输出的函数声明格式，在之后的项目中遇到了会具体分析。

何为变量?

```
2.  volatile float mind_n_crossTime, mind_n_changeTime, mind_n_ButtonState;
```

第二行代码的作用是声明变量，每次使用一个新的变量前，我们都需要去对其声明，对应指令区的：

但变量是什么呢?

我们做个这样的比方，变量好比一个盒子，盒子的空间用来存放东西的，想要放的东西一定要比盒子小，那样才放的下，否则会溢出。变量也是一样，你存储的数据一定要在变量的范围内，否则会出现溢出。

之所以叫变量，是因为程序运行过程中，可以改变它的值。程序中，有时候会对变量值进行数字计算，变量的值也会随之发生变化。在以后的项目中，我们会有深入的了解。

在之前学习如何新建变量中，我们已经知道在给变量起名时，由于 Mind+可以帮你直接转化变量名从而不产生冲突，但是在代码编程中，变量的命名有很多规则，比如在 c 语言中，变量名必须以一个字母开头，之后可以包含字母、数字、下划线。注意 C 语言认为大小写字母是不同的。C 语言中还有一些特定的名称也是不能使用的，比如 main，if，while 等。为了避免这些特定名称作为变量名，所有这些名称在程序中显示为高亮色。

变量这个盒子无限大吗?

之前举例了变量像个盒子，那变量是不是也像盒子一般有不同的大小呢？就像生活中的收纳盒，小的用来装文具，大的可以放下书本，而有的专门用来存放衣物。

答案是有的！变量这个盒子有很多类型和尺寸，具体可以阅读表 3-1。

表 3-1 程序中可能用到的变量数据类型

数据类型	RAM	范围
boolean (布尔型)	1 byte	0 ~ 1 (True 或 False)
char（字符型）	1 byte	-128 ~ 127
unsigned char（无符号字符型）	1 byte	0~255
int（整型）	2 byte	-32768 ~ 32768
unsigned int（无符号整型）	2 byte	0 ~ 65535
long（长整型）	4 byte	-2147483648 ~ 2147483647
unsigned long（无符号长整型）	4 byte	0 ~ 4294967295
float（单精度浮点型）	4 byte	-3.4028235E38 ~ 3.4028235E38
double（双精度浮点型）	4 byte	-3.4028235E38 ~ 3.4028235E38

从上面表格可以看到，变量的类型有很多，不同的数对应不同的变量，int 和 long 是针对整数变量，char 是针对字符型变量，而 float，double 是针对含有小数点的变量。

那又有人问，设置不同大小的盒子干嘛呢？一样大不就行啦，都设置的大一点。理论上没有什么不可以的，可是我们不能忽略一个问题，那就是微控制器的内部存储容量是有限定的。电脑有内存，我们的微控制器同样有内存。像 Arduino UNO 板上的用的主芯片 Atmega328 最大内存是 32K。

在 Mind+中使用图形化编程新建数字类型变量指令时，为简化使用，都采用了 float 型的变量并加入了 volatile 修饰符，这是为了兼顾使用小数、整数及在中断中使用变量，在我们手动编程时则可以根据需要选择变量类型和修饰词。

全局变量和局部变量

我们这里要引用一个新的概念，是全局变量和局部变量。

全局变量，例如在第二行中所声明的变量，它能在整个程序中起作用，但条件是，必须在 setup()、loop()函数外声明。

```
3.  volatile float mind_n_crossTime, mind_n_changeTime, mind_n_ButtonState;
```

局部变量，例如 32 行中的 index

```
32.  for (int index = 0; index < 10; index++) { }
```

这类变量只在自己的代码内起作用。就像我们这里 for 循环中的变量 index，它就是个局部变量。因此在项目二中每个 for 循环中都有一个变量 index，但它们不冲突的原因，就是它们只在自己的循环中执行。

我们接着往下看代码，会在 14 行发现一个新的函数 digitalRead()

```
14.  mind_n_ButtonState = digitalRead(9);
```

- digitalRead()

函数格式如下：

对应指令：

这个函数是用来读取数字引脚状态，HIGH 还是 LOW（其实 HIGH 还有一种表达就是“1”，LOW 是“0”，只是 HIGH/LOW 更直观）。函数需要一个传递参数——pin，这里需要读取的是按键信号，按键所在引脚是数字引脚 9。

并且把读到的信号传递给变量 mind_n_ButtonState，用于后面进行判断。ButtonState 为 HIGH 或者说为 1 时，说明按键被按下了。ButtonState 为 LOW 或者 0，表明按键没被按下。

*本章之前已经讲解了 Mind+在代码中给变量命名时会增加前缀，行文中在提到变量时以命名时输入的变量名代替。

所以，可以直接检查 state 的值来判断按钮是否被按下：

```
15.    if (((mind_n_ButtonState==1) && ((millis()-mind_n_changeTime)>5000)))
16.     {
17.       changelight();
18.    }
```

这里涉及新的语句——if 语句。

- if 语句

它是一种条件判断的语句，判断是否满足括号内的条件，如满足则执行花括号内的语句，如不满足则跳出 if 语句。

if 语句格式如下：

```
if(表达式){
    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。

我们代码中，第一个条件是 ButtonState 变量为 HIGH。如果按键被按下，ButtonState 就会变为 HIGH。第二个条件是 millis()的值减 changeTime 的值大于 5000。这两个条件之间有个“&&”符号。这是一种逻辑运算符，表示的含义是两者同时满足。

在 if 的判断条件中，我们可以发现这(millis()-mind_n_changeTime)>5000 中有一个新的函数。

- 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。变量 changeTime 初始化时，不存储任何数值，只有在 Arduino 运行之后，将 millis()值赋给它，它才开始有数值，并且随着 millis()值变化而变化。通过 millis()函数不断记录时间，判断两次按键之间的时间是不是大于 5 秒，如果在 5 秒之内不予反应。这样做能防止重复按键而导致的运行错误。

逻辑运算符

指令学习中已经提到了逻辑运算符，在 if 语句的判断条件中&&也是其中之一，常用的逻辑运算符有：

- &&　——　逻辑与　（两者同时满足）
- ||　——　逻辑或　（两者其中一个满足）
- !　——　逻辑非　（取反，相反的情况）

输入值 1	输入值 2	返回值
1	1	1
1	0	0
0	1	0
0	0	0

&& 与 真值表

输入值 1	输入值 2	返回值
1	1	1
1	0	1
0	1	1
0	0	0

|| 或 真值表

输入值	返回值
1	0
0	1

！ 非 真值表

表中的 1、0，既能够分别代表着高（1）、低电平（0），也能够代表着判断后的真（1）、假（0）

例如（10 > 8) && (9<5)这一句的返回值是多少呢?

10>8 我们判断后这是正确的，那么这个比较语句的返回值为 1。9<5 显然是假的，返回值为 0。整个语句可以简化为 1 && 0， 再由上方的真值表可以得出，（10 > 8）&&（9<5 ）的返回值为 0。

if 语句内只有一个函数：

- changeLights();

这是一个函数调用的例子。该函数单独写在了 loop()函数之外。我们需要使用的时，直接写出函数名就可以实现调用了。该函数是 void 型且后面括号内为空，所以是无返回值、无传递参数的函数。当函数被调用时，程序也就自动跳到它的函数中运行。运行完之后，再跳回主函数。需要特别注意的：函数调用时，函数名后面的括号不能省，要和所写的函数保持一致。changeLights() 函数内部程序请自行理解，这里就不做说明了。

硬件回顾

按键开关

按键一共有 4 个引脚，图 3-2 分别显示了正面与背面。而图 3-3 则说明了按键的工作原理。一旦按下后，左右两侧就被导通了，而上下两端始终导通。

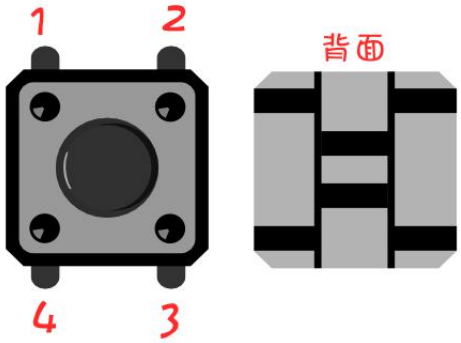


图 3-2 按钮结构图

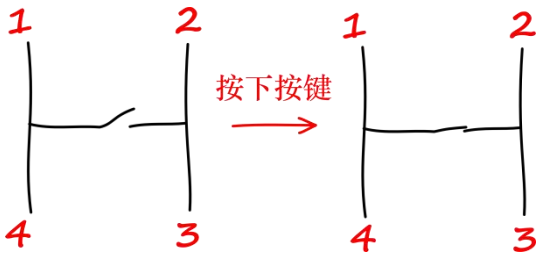


图 3-3 按钮原理图

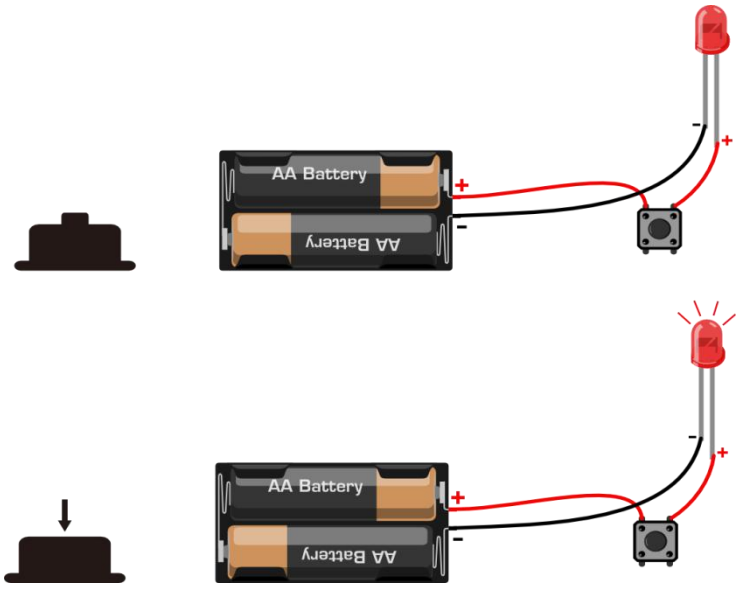


图 3-4 按钮示意图

图 3-4 传达的意思是，按钮就是起到一个通断的作用。在我们这个项目中，按钮控制数字引脚是否接高电平 (接 5V) 。按下的话，数字引脚 9 就能检测到为高电平。否则就是保持一个低电平的状态 (接 GND)。

什么是下拉电阻?

下拉电阻这个名词可能比较抽象，就从字的含义着手， “下拉”我们就理解为把电压往下拉，降低电压。

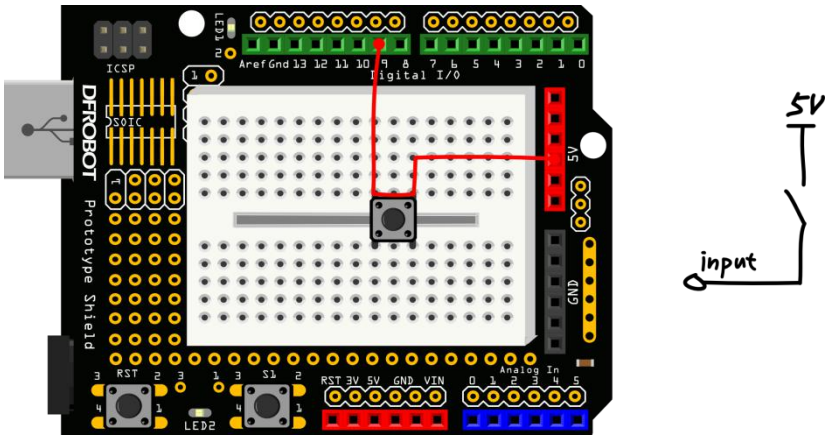


图 3-5 未接下拉电阻

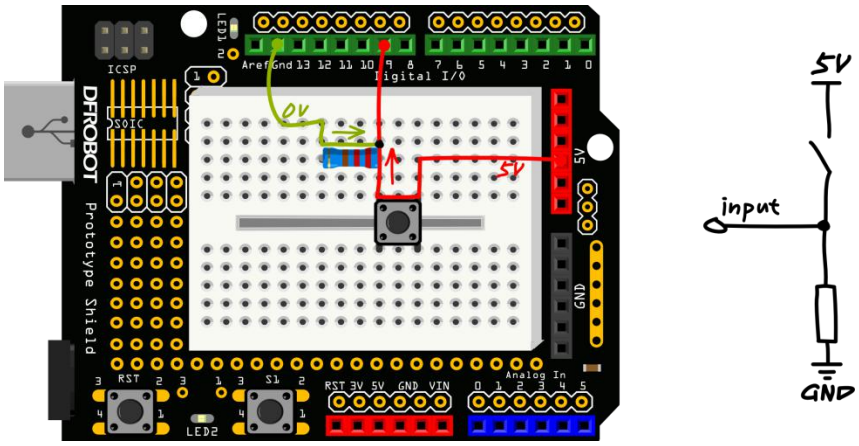


图 3-6 有下拉电阻

按键作为开关。当输入电路状态为 HIGH 的时候，电压要尽可能接近 5V。输入电路状态为 LOW 的时候，电压要尽可能接近 0V。如果不能确保状态接近所需电压，这部分电路就会产生电压浮动。所以，我们在按钮那里接了一个电阻来确保一定达到 LOW，这个电阻就是所谓下拉电阻。

可以从上面两张图看到，第一张是未接下拉电阻的电路，按键没被按下时， input 引脚就处于一个悬空状态。空气会使该引脚电压产生浮动，不能确保是 0V。然而第二张是接了下拉电阻的电路，当没被按下时，输入引脚通过电阻接地，确保为 0V，不会产生电压浮动现象。

课后作业

1. 本项目内容较多，但还是希望同学们能够花点时间自己动手输入代码，不荒废每次练习的机会。

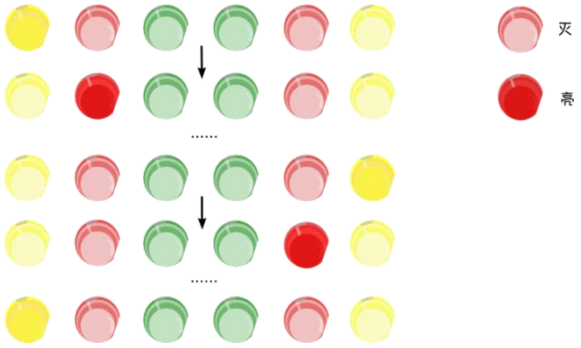
*仅编译后出现报错的可能原因（2）：

```
1 void setup(){
2   digitalWrite(1, HIGH);
3 |
4
5 void loop(){
6  }
```

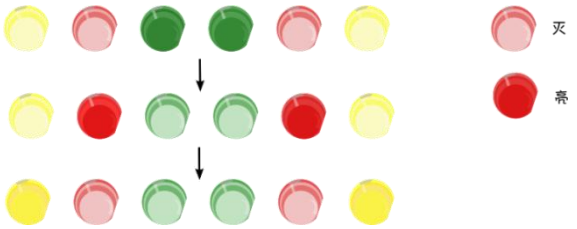
```
compile error.
Error: Command failed: "D:\Software-Install\MIND+1.6.0\arduino\hardware\tools\avr\bin\avr-g++" -c -g -Os
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp: In function 'void setup()':
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp:6:12: error: a function-definition is not a
 void loop(){
           ^
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp:7:1: error: expected '}' at end of input
 }
 ^
```

这一问题经常出现于自己声明函数，在写完函数内容后缺失了 } 这一花括号

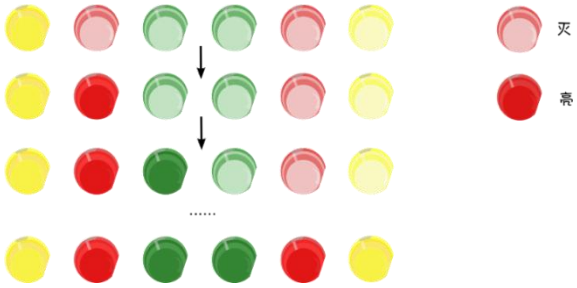
2. 任意颜色 LED 6 个，做一个流水灯的效果，6 盏灯从左至右依次点亮，然后再从右至左依次熄灭



3. 如果上面那个你已经完成了的话，可以尝试一下，先从中间的灯开始亮起，依次向两边扩开。下图是个变换过程的示意图。



4. 再比如，从左至右，依次亮起 1 个，2 个，3 个……



5. 再结合按钮，用按键开关和 LED 互动。

用一个按键，按一下控制灯亮，再按一下控制灯灭。

又或者用两个按键，一个控制灯亮，另一个控制灯灭。

玩儿法有很多，就全靠你的想象了！

项目四　呼吸灯

在前面几章中，我们知道了如何通过程序来控制 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 板上，数字引脚中有六个引脚标有“~”，分别是 3、5、6、9、10、11，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

- 1×　5mm LED 灯 
- 1×　220 欧电阻 

硬件连接

这个项目的硬件连接与项目一、二是完全相同的。如有不明白，可以回看项目一，或参照图 4-1。

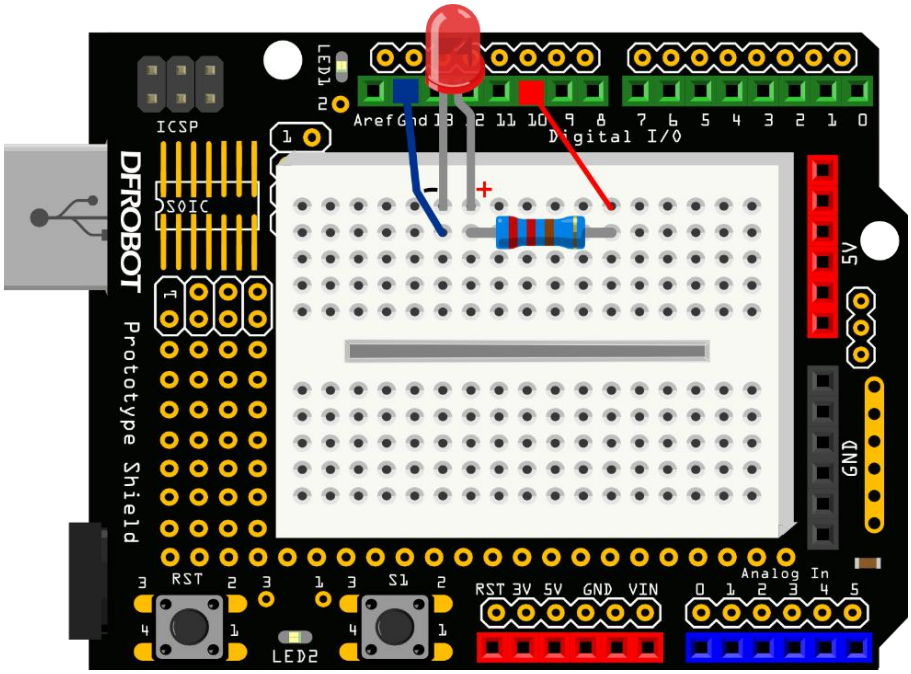
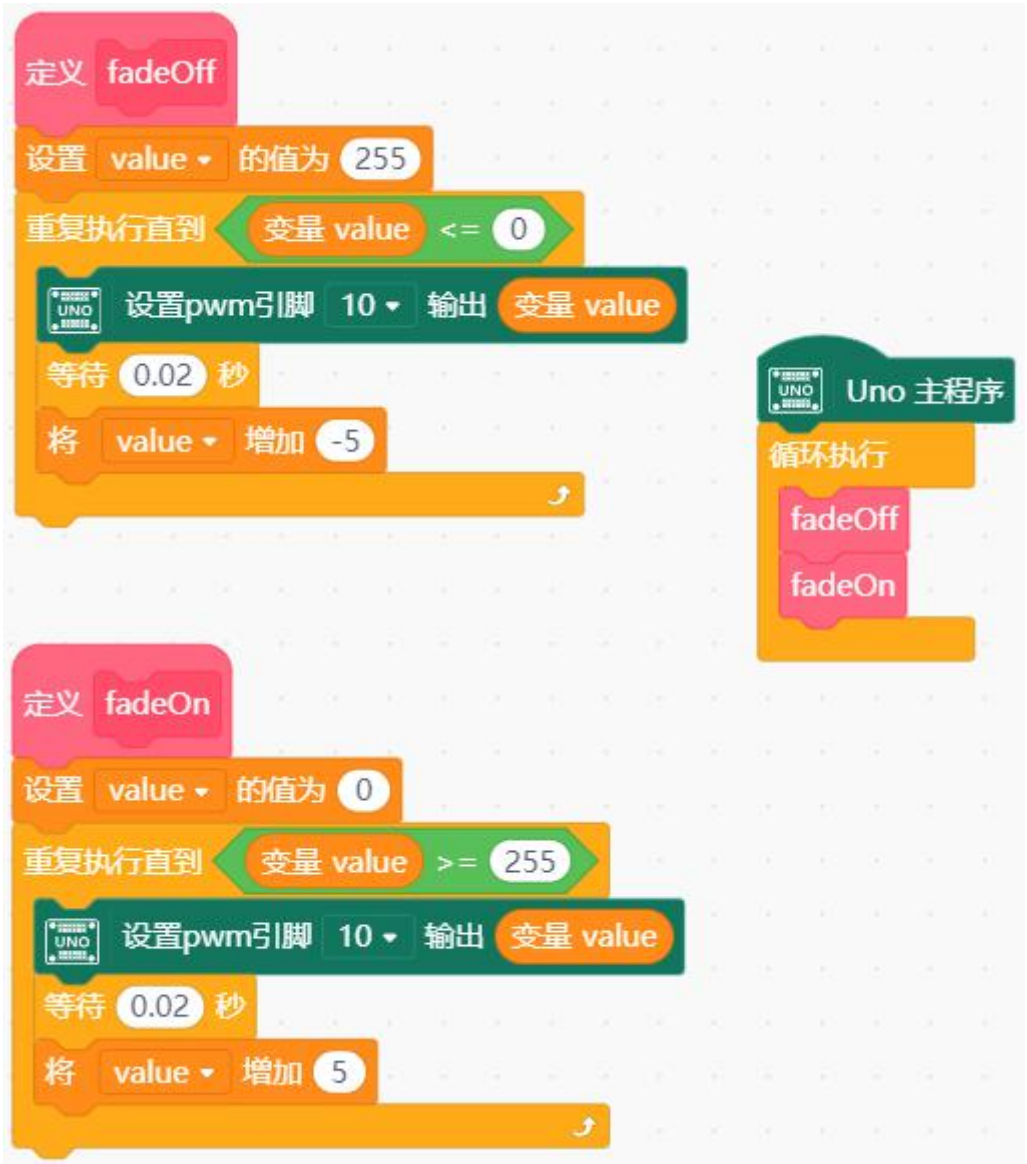


图 4-1 呼吸灯连接示意图

图形化编程

连接主板和电脑，打开 Mind+载入扩展库，输入样例程序 4-1 所示程序。本项目中将用到一个新的循环，指令学习和代码学习中会有详细的讲解。

样例程序 4-1：



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。

图形化指令学习

所属模块	指令	功能
变量	重复执行直到	指定次循环条件的循环指令：将指令中包含的程序自下而上循环执行，直到不满足循环条件，退

		出循环。
Arduino		设置 PWM 引脚输出值指令。 通过 PWM 信号 可以控制亮度（输出值的范围在 0-255）。

代码学习

```
volatile float mind_n_value;
// 函数声明
void DF_fadeOff();
void DF_fadeOn();


// 主程序开始
void setup() {

}
void loop() {
    DF_fadeOff();
    DF_fadeOn();
}


// 自定义函数
void DF_fadeOff() {
    mind_n_value = 255;
    while (!(mind_n_value<=0)) {
        analogWrite(10, mind_n_value);
        delay(20);
        mind_n_value -= 5;
    }
}
void DF_fadeOn() {
    mind_n_value = 0;
    while (!(mind_n_value>=255)) {
        analogWrite(10, mind_n_value);
        delay(20);
        mind_n_value += 5;
    }
}
```

大部分代码我们已经很熟悉了，比如初始化变量声明、引脚设置、以及函数调用。

在主函数中，只有两个调用函数 fadeOff 和 fadeOn。前者是让 LED 逐渐变暗，后者是逐渐变亮，以此来实现呼吸灯的效果。这两个函数的内容基本一致，在数值设置上存在一些差别，故先行学习其中一个。

```
19. void fadeOff() {
20.     mind_n_value = 255;
21.     while (!(mind_n_value<0)) {
22.         analogWrite(10, mind_n_value);
23.         delay(20);
24.         mind_n_value -= 5;
25.     }
26. }
```

fadeOff 函数中出现了一个新的语句 while，while 语句通常在程序用作条件循环。

While（）{}

函数格式如下：

```
while（循环条件）
{
循环体
}
```

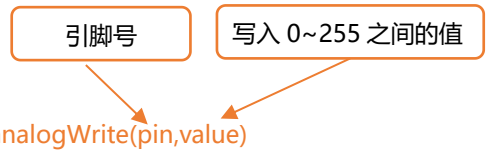
对应指令为： 

while 语句中涉及了一个新函数：

- analogWrite(10 ,mind_n_value);

如何发送一个模拟值到一个数字引脚呢？就要用到该函数，使用这个函数是要具备特定条件的——该数字引脚需具有 PWM 功能。观察一下 Arduino 板，查看数字引脚，你会发现其中 6 个引脚（3、5、6、9、10、11）旁标有“~”，这些引脚不同于其他引脚，因为它们可以输出 PWM 信号。

函数格式如下：



analogWrite(pin,value)

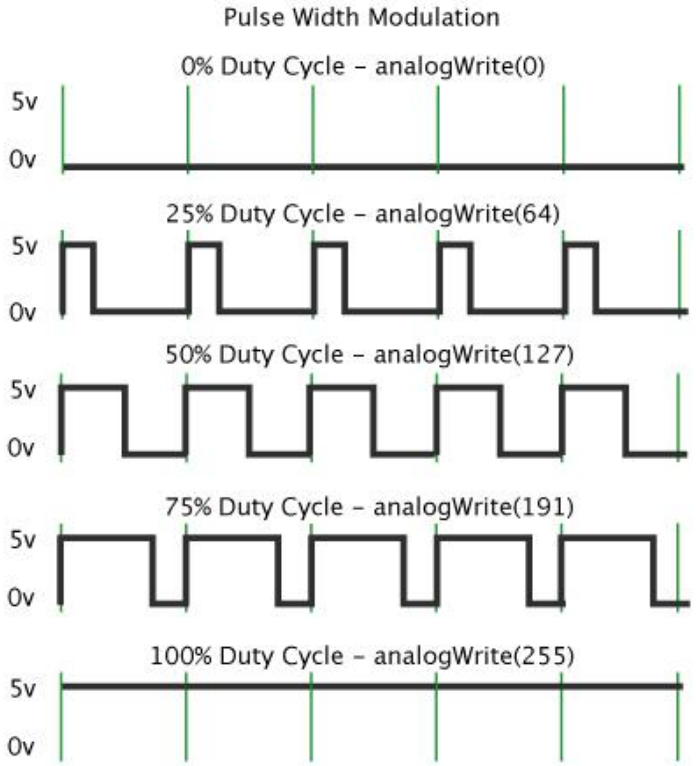
对应指令为： 

analogWrite()函数用于给 PWM 口写入一个 0~255 的模拟值。特别注意的是，analogWrite()函数只能写入具有 PWM 功能的数字引脚。

PWM

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。



上图绿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一个周期时间得到的百分比。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 Arduino 小车时，更能体现 PWM 的好处。

课后作业

1. 同前两个项目，本课的第一条作业依旧是手动输入项目的代码，旨在养成这一习惯、勤加操练，为之后的项目打好必要的输入代码、编译后自行纠错 debug 能力的基础。

*如果遇到其他的编译报错可以登录 DF 创客社区查看或发帖询问。

2. 用 LED 能否做个火焰的效果，通过 PWM 使 LED 产生随机的亮度变化，来模拟一个火焰闪烁的效果。找个浅色罩子盖住效果更佳，可以放在家中作为小夜灯。

主要材料：一个红色 LED、两个黄色 LED 以及 220 欧电阻。在这个实验中，有个函数会比较好用——random()，Mind+中对应的指令为 

。它可以产生一定范围内的随机数。

提示：可以先设定 LED 灯亮度，在其值附近产生一个随机数，比如 random(120)+135，对应指令 

，让其值稳定在 135 附近，产生这种小幅变化，就更具有火焰跳跃感。不妨尝试一下。

random 函数格式为：random（120）它能够从 0-120 随机取值。

具体其他用法可以查看下面链接的编程参考手册，会详细介绍这个函数的用法。之后的讲解中，我们可能有对些函数不进行详细说明，你可以通过这种方法来学习某个新函数。

可以登录：http://wiki.dfrobot.com.cn

在“入门教程”中的“Arduino 编程参考手册”可以学习更多的函数。

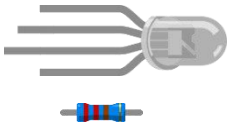
3. 尝试一个稍微有点难度的，通过两个按键，一个按键控制 LED 逐次变亮，另一个按键控制 LED 逐次变暗。

项目五 炫彩 RGB LED

单色 LED 我们就讲到这里了，现在介绍一种新的 LED——RGB LED。之所以叫 RGB，是因为这个 LED 是由红(Red)、绿(Green)和蓝(Blue)三色组成。我们电脑的显示器也是由一个个小的红、绿、蓝点组成的。可以通过调整三个 LED 中每个灯的亮度就能产生不同的颜色。这个项目就是教你通过一个 RGB 小灯随机产生不同的炫彩颜色。我们可以先感性的看一下，按下图连接硬件并输入代码。

所需元件

- 1× 5mm RGB LED 灯
- 3× 220 欧电阻



硬件连接

连接之前，先判别 RGB 是共阴还是共阳，如果不是很清楚，可以先跳到这个项目的硬件部分介绍。连接时，还需注意引脚的顺序，可参照右边的引脚图。

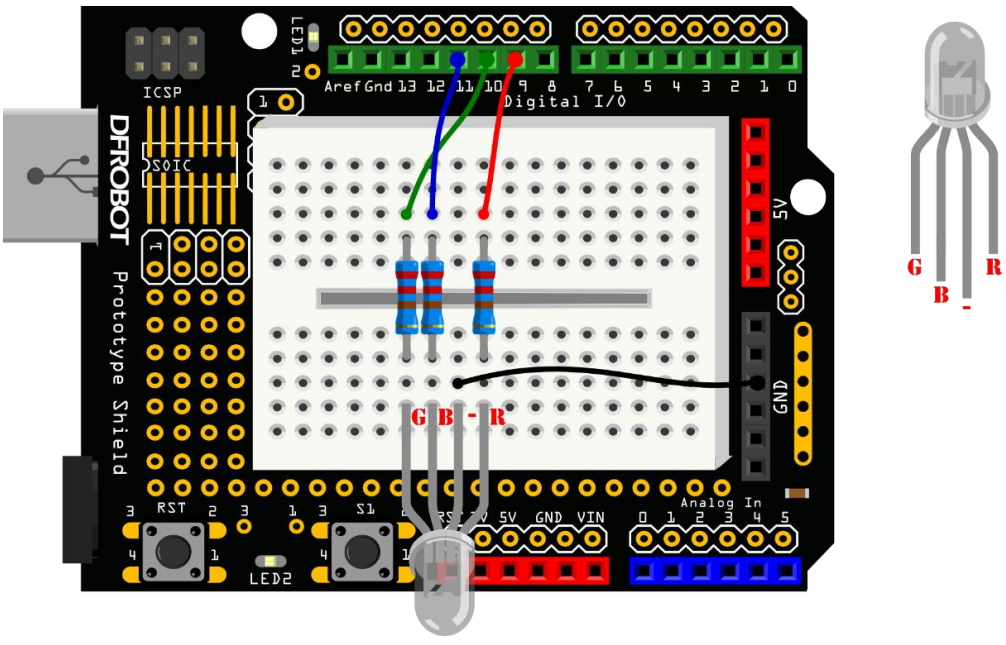


图 5-1 炫彩 RGB LED 连线图

图形化编程

输入样例程序 5-1 所示程序。本项目中需要声明一个可以输入参数的函数，函数的创建步骤之前已经讲解。在新建函数时，如下图，点击框中的“添加输入项”即可命名输入参数的名称。

点住并拖曳“red”这一参数，能够像其他变量一样使用。并且它是个局部变量，仅可以在这一函数中调用。

样例程序 5-1：

代码下载完成后，我们可以看到 LED 颜色呈现随机的变化，不只是单一的一种颜色。

图形化指令学习

所属模块	指令	功能
运算符	在 1 和 10 之间取随机数	随机数指令：用于获得指定范围内的整数数值。
运算符	约束 0 介于(最小值) 0 和(最大值) 100 之间	约束值指令：将值、变量约束到指定范围内。

代码学习

本项目的 C 代码如下。接下来我们将依次学习每一行代码的意义

```
1.  // 函数声明
2.  void colorRGB(float red, float green, float blue);
3.
4.
5.  // 主程序开始
6.  void setup() {
7.      dfrobotRandomSeed();
8.  }
9.  void loop() {
10.     colorRGB((random(0, 255+1)), (random(0, 255+1)), (random(0, 255+1)));
11.     delay(1000);
12. }
13.
14.
15. // 自定义函数
16. void colorRGB(float red, float green, float blue) {
17.     analogWrite(9, (constrain(red, 0, 255)));
18.     analogWrite(10, (constrain(green, 0, 255)));
19.     analogWrite(11, (constrain(blue, 0, 255)));
20. }
```

来分析一下，其实一个 RGB 灯，就是我们前面讲的单色 LED 的结合体，内部集成了三个 LED，也就需要用三个数字 PWM 口来控制。

程序最主要的部分，也就是主函数。主函数中调用了一个自己创建的函数 colorRGB()，函数有三个传递参数，用于写入 Red、Green、Blue 的值，也就是 0~255 的值。

使用函数的好处在于，之后我们想调到某个颜色的时候，只要接给这三个参数赋值就可以了。不需要重复写 analogWrite()函数，使程序变得冗长。

这段函数中，我们比较陌生的就是 constrain()和 random()这两个函数。

我们上一个项目的课后作业部分提到的两个网站。通过那个方法，能否尝试自己来学习一下这两个函数。

- constrain（x, a, b）

constrain（x, a, b） 对应指令为：

constrain()函数需要 3 个参数：x、a 和 b。这里 x 是一个被约束的数，a 是最小值，b 是最大值。如果值小于 a，则返回 a。如果大于 b，则返回 b。

回到我们的程序，red、green、blue 值是被约束数，约束范围在 0~255，也就是我们 PWM 值的范围。它们的值来源于 random()函数随机产生。

- dfrobotRandomSeed();

dfrobotRandomSeed()函数用来设置随机数种子、初始化随机数生成器。利用模拟量引脚（A6、A7）来读取空气中不确定的模拟噪声，用它来做随机种子，可获取到更加理想的随机数。

- random（min,max）

random()函数用于生成一个随机数，min 是随机数的最小值，max 是随机数的最大值，生成[min, max-1]范围的随机数.。在代码中写为“255+1”是因为 Mind+在转化图形指令的时候，为了确保能 random()函数能够取到 255 这个值。关于 random()函数的其他用法，可以参看手册。

硬件回顾

RGB 灯

RGB 灯有 4 个引脚，R、G、B 三个引脚连接到 LED 灯的一端，还有一个引脚是共用的正极（阳）或者共用的阴极（负）。我们这里选用的是共阴 RGB。看下图 5-2，展示了三个 LED 如何华丽蜕变为一个 RGB 的过程，R、G、B 其实就是三个 LED 的正极，把它们的负极拉到一个公共引脚上了，它们公共引脚是负极，所以称之为共阴 RGB。

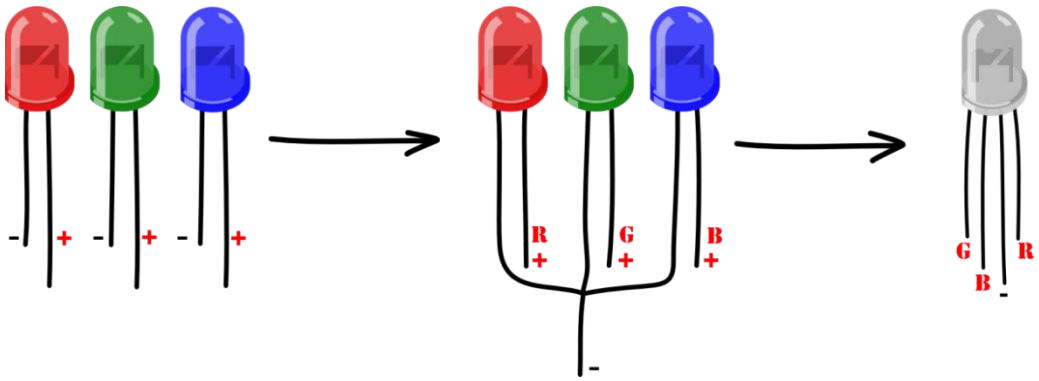


图 5-2 3 个 LED 蜕变为 1 个 RGB 的过程

RGB 灯如何使用呢？如何实现变色呢？

RGB 只是简单的把三个颜色的 LED 灯封装在一个 LED 中。只要当做三个灯使用就可以了。我们都知道红色、绿色、蓝色是三原色，Arduino 通过 PWM 口对三种颜色明暗的调节，也就 analogWrite(value)语句，就能让 LED 调出任何你想要的颜色。

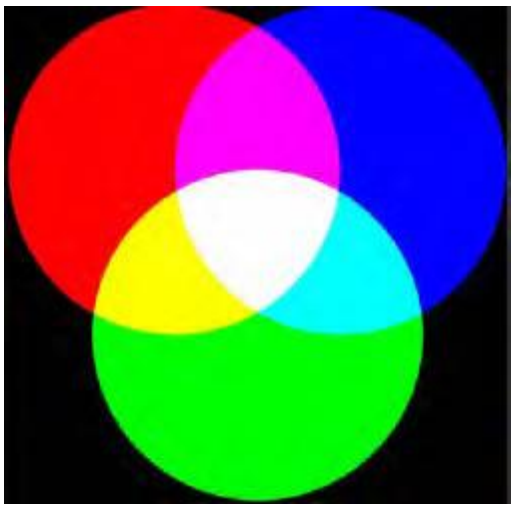


图 5-3 混合 R、G、B 获得不同的颜色

表 5-1 不同 LED 的 PWM 值所组合产生的颜色

红色	绿色	蓝色	颜色
255	0	0	红色
0	255	0	绿色
0	0	255	蓝色
255	255	0	黄色
0	255	255	蓝绿色
255	0	255	紫红色
255	255	255	白色

表 5-1 只是罗列了几种典型的颜色，可调的色彩远多于上表所示的，使用 PWM 可以产生 0~255 之间的全部颜色，共 16777216 种颜色（256×256×256）。不妨可以动手尝试一下，设置三个 LED 的 PWM 值来，随意切换颜色吧！

共阳 RGB 与共阴 RGB 的区别

上面我们还遗留一个问题——共阴与共阳在使用上有什么区别？共阳 RGB 就是把正极拉到一个公共引脚，其他三个端则是负极。下图是可以看出，外表上共阴共阳没有任何区别。

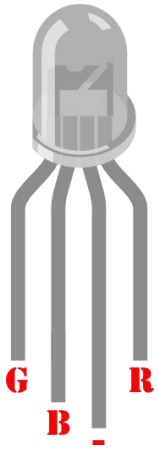


图 5-3 共阴 RGB 示意图

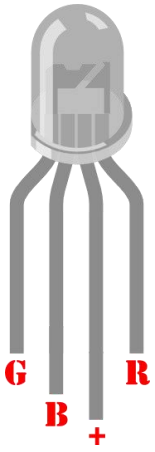


图 5-4 共阳 RGB 示意图

然而在使用上是有区别的，区别分为以下两点：

（1）接线中的改变，共阳的话，共用端需要接 5V，而不是 GND，否则 LED 不能被点亮。

（2）第二点就是，在颜色的调配上，与共阴是完全相反的。

举个例子：共阴 RGB 显示红色为 R-255，G-0，B-0。然而共阳则完全相反，RGB 数值是 R-0,G-255,B-255。

课后作业

1. 手动输入本项目的代码。

2. 基于我们上面的炫彩 RGB 项目，改变代码做一个沿着彩虹色变化的 RGB 灯，而不是我们这样随机产生颜色。这里比较困难的应该是颜色的调制，通过改变 Red、Blue、Green 的值 0~255，组合出一个你想要的颜色。

提示：只要在原有代码基础上做修改就可以了，直接调用 colorRGB()函数，将函数中 3 个参数写入所对应颜色的值即可。

3. 在作业 2 的基础上，能否结合我们上面说的呼吸灯，将彩虹色以呼吸灯渐变形式变化。这样的变换会显得更加柔和。

项目六 报警器

这里我们要接触一个新的电子元件——蜂鸣器，从字面意思就可以知道，这是一个会发声的元件。这次做一个报警器，通过连接蜂鸣器到 Arduino 数字输出引脚，并配合相应的程序就可以产生报警器的声音。其原理是利用正弦波产生不同频率的声音。如果结合一个 LED，配合同样的正弦波产生灯光的话，就是一个完整的报警器了。

所需元件

- 1× 蜂鸣器

硬件连接

按下图连接图连接，注意蜂鸣器引脚正（+），负（-）。标有 + 的引脚接到数字口 8，另一个接 GND。

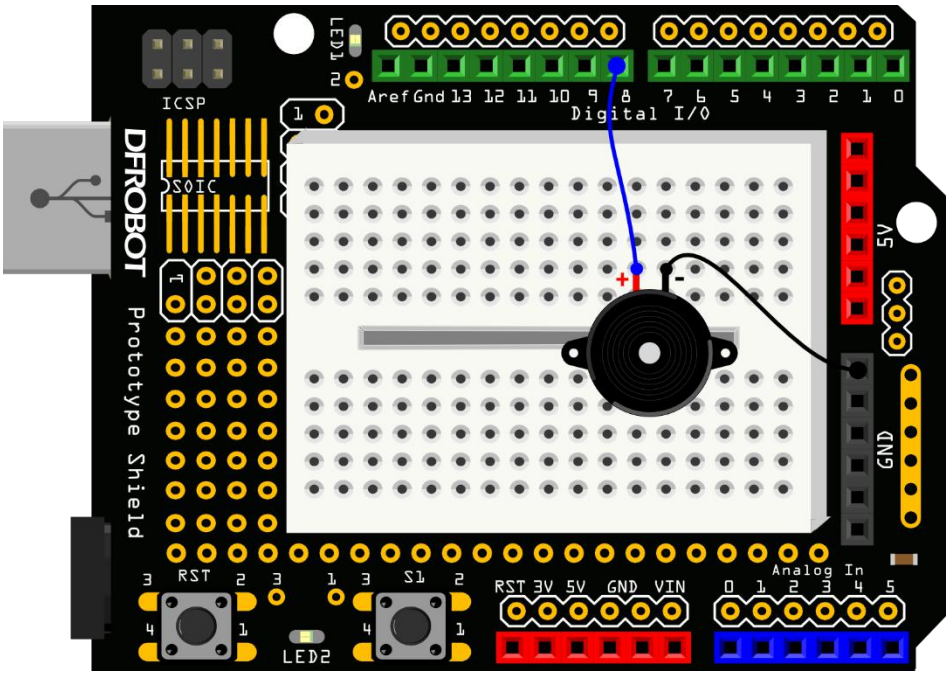


图 6-1 报警器连线图

图形化编程

样例程序 6-1：

下载程序完成后，你会听到一高一低的报警声，如同汽车报警器。

图形化指令学习

所属模块	指令	功能
Arduino	设置引脚 9 喇叭蜂鸣器音调为 1 低 C/C3 节拍为 1/2	控制蜂鸣器输出的音调高低和时间长短。

代码学习

项目代码如下

```
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_Tone DFTone;

// 主程序开始
void setup() {

}
void loop() {
    DFTone.play(8, 131, 250);
    DFTone.play(8, 523, 250);
}
```

Loop 中只有一个新的函数，DFTone.Play( )。对应指令 

函数格式如下：

如果我们希望蜂鸣器能够发出更加多样，和不同时长的声音，就会发现，单单依靠 Mind+的图形化编程已经无法满足预期的要求了，“播放音符”指令后对应的节拍表示发音持续时间，在 Mind+中可以理解为 1 拍=1 秒。如图，Mind+的发音时长只有几个固定的选项，无法再减小发声时间。

。

Mind+的图形指令虽然使用便捷，但是一定程度上牺牲了代码的灵活性，也限制了使用者天马行空的创造力。所以从现在开始，我们将图形化编程这根拐杖放在一旁，只用代码编程完成后续课程的项目，快来感受它的魅力吧！

代码编程

在手动编辑区输入样例代码 6-2

```
//项目六 报警器
float sinVal;
int toneVal;
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_Tone DFTone;

void setup(){
    pinMode(8, OUTPUT);
}

void loop(){
    for(int x=0; x<180; x++){
```

```
            //将 sin 函数角度转化为弧度
            sinVal = (sin(x*(3.1415/180)));
            //用 sin 函数值产生声音的频率
            toneVal = 2000+(int(sinVal*1000));
            //给引脚 8 一个蜂鸣器声音输出信号.
            DFTone.play(8, toneVal,2);
        }
}
```

下载完代码后，我们能够听见蜂鸣器发出由低到高、再由高到低的警报声。不同于之前图形化编程发出较为简单的声音，使用代码能够让输出更具有变化和多样性。

代码学习

首先，定义两个变量：

```
float sinVal;
int toneVal;
```

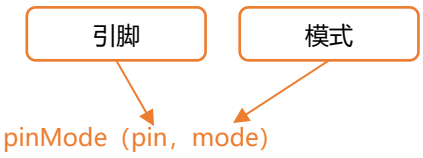
浮点型变量 sinVal 用来存储正弦值，这是由于正弦值有小数。正弦波呈现一个波浪形的变化，变化比较均匀，所以我们选用正弦波的变化来作为我们声音频率的变换，toneVal 从 sinVal 变量中获得数值，并把它转换为所需要的频率。

```
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_Tone DFTone;
```

对于库和对象，将在项目十中做详细解释，那时有了几次使用的基础将更容易理解。

```
void setup(){
    pinMode(8, OUTPUT);
}
```

setup 中的函数 pinMode（）的格式如下：



这个函数的作用是，来规定某个引脚是作为输入，或是输出，mode 的值分别对应 INPUT（输入）和 OUTPUT（输出）。为什么在 Mind+中没有这条指令，是因为它已经被封装在了其他指令中，比如 digitalwrite（pin），当程序运行数字信号输出指令的时候，其实已经使用了 pinMod(OUT)指令，但是这一过程没有在代码中体现。

这里用的是 sin()函数，一个数学函数，可以算出一个角度的正弦值，这个函数采用弧度单位。因为我

们不想让函数值出现负数，所以设置 for 循环在 0~179 之间，也就是 0~180 度之间。

```
    for(int x=0; x<180; x++){
```

函数 sin()用的弧度单位，不是角度单位。要通过公式将角度转为弧度：

```
        sinVal = (sin(x*(3.1415/180)));
```

之后，将这个值转变成相应的报警声音的频率：

```
        toneVal = 2000+(int(sinVal*1000));
```

这里有个知识点——浮点型值转换为整型。

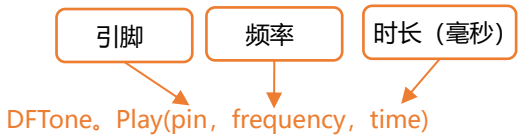
sinVal 是个浮点型变量，也就是含小数点的值，而我们不希望频率出现小数点的，所以需要有一个浮点值转换为整型值的过程，下方语句完成了这件事：

```
        int(sinVal*1000)
```

把 sinVal 乘以 1000，转换为整型后再加上 2000 赋值给变量 toneVal，现在 toneVal 就是一个适合声音频率了。

之后，我们用 tone()函数把生成的这个频率给我们的蜂鸣器。

```
        DFTone.play(8, toneVal,2);
```



硬件回顾

蜂鸣器

蜂鸣器其实就是一种会发声的电子元件。蜂鸣器主要分为压电式蜂鸣器和电磁式蜂鸣器两种类型。

压电式蜂鸣器和电磁式蜂鸣器区别

压电式蜂鸣器是以压电陶瓷的压电效应，来带动金属片的振动而发声。当受到外力导致压电材料发生形变时压电材料会产生电荷。电磁式的蜂鸣器，则是利用通电导体会产生磁场的特性，通电时将金属振动膜吸下，不通电时依振动膜的弹力弹回。不太明白也没太大关系，不影响我们使用。

压电式蜂鸣器需要比较高的电压才能有足够的音压,一般建议为 9V 以上。电磁式蜂鸣器用 1.5V 就可以发出 85dB 以上的音压了，唯消耗电流会大大的高于压电式蜂鸣器。所以还是建议初学者使用电磁式蜂鸣器。

有源蜂鸣器和无源蜂鸣器区别

无论是压电式蜂鸣器还是电磁式蜂鸣器，都有有源蜂鸣器和无源蜂鸣器两种区分。

有源蜂鸣器和无源蜂鸣器的根本区别是输入信号的要求不一样。这里的“源”不是指电源，而是指振荡源，有源蜂鸣器内部带振荡源，说白了就是只要一通电就会响，适合做一些单一的提示音。而无源内部不带震荡源，所以如果仅用直流信号无法使其响，必须用 2K-5K 的方波去驱动它，但是无源蜂鸣器比有源蜂鸣器音效更好，适合需要多种音调的应用。

从外观上看，有源无源的区别在于，有源蜂鸣器有长短脚，也就是所谓正负极，长脚为正极，短脚为负极。而无源蜂鸣器则没有正负极，两个引脚长度相同。

在套件中，我们为初学者选用的蜂鸣器类型是电磁式无源蜂鸣器。当然，如果喜欢探究，不妨考虑买一个有源蜂鸣器玩玩，直观感受一下两者的区别。

蜂鸣器的应用有很多，我们都可以就蜂鸣器做一些好玩的东西，比如常见的会结合蜂鸣器的有红外传感器，超声波传感器，用于监测物体靠近报警。温度传感器，测到温度过高报警。气体传感器，有气体泄漏报警等等。除了报警，还可以用来作为乐器，通过不同频率，调成乐谱的不同调式，是不是很 amazing 啊?

课后作业

1. 在手动编辑中，输入本项目的代码。

2. 结合红色 LED 灯做一个完整的报警器。

   提示：可以让 LED 也随着 sin 函数变化，使声音与灯光节奏保持一致。

3. 结合项目三中介绍的按钮，做个简易门铃的效果，每次按下按钮，蜂鸣器发出提示音。

项目七 温度报警器

在上一节中，我们认识了一个发声元件——蜂鸣器，也做了一个简单的小报警器。是不是还不过瘾呢？这次我们要做一个更实际的应用——温度报警器。当温度到达我们设定的限定值时，报警器就会响。我们可以用于厨房温度检测报警等等，各种需要检测温度的场合。这个项目中，除了要用到蜂鸣器外，还需要一个 LM35 温度传感器。

我们这里将头一回接触传感器，传感器是什么？简单的从字面上的理解就是，一种能感知周围环境，并把感知到的信号转换为电信号的感应元件。感应元件再把电信号传递给控制器。就好比人的各个感官，感知周围环境后，再信息传递给大脑是一样的道理。

所需元件

- 1× 蜂鸣器

- 1× LM35 温度传感器

硬件连接

蜂鸣器和项目六的接法相同。在接 LM35 温度传感器时，注意三个引脚的位置，有 LM35 字样的一面面向自己，从左至右依次接 5V、Analog 0、GND，如我们下图所示。

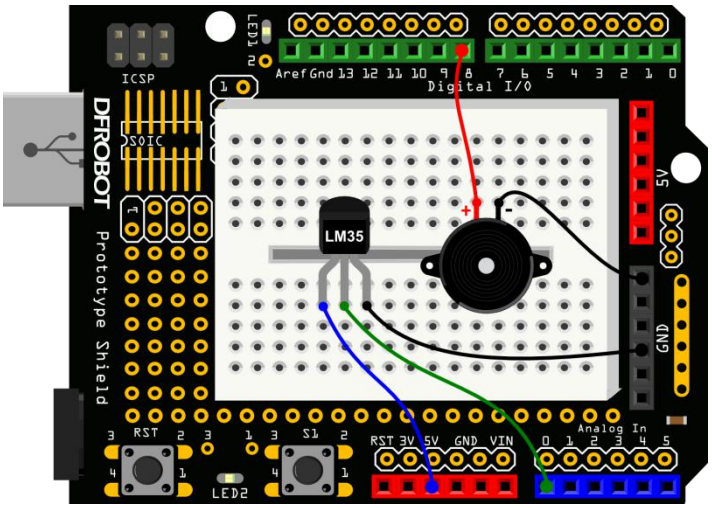


图 7-1 温度报警器连线图

代码编程

从这个项目开始，我们将彻底转为代码编程。有了之前项目手动输入代码的练习，我们能够很快适应纯代码的编程。

相信拥有探索精神的你已经发现如何选择代码风格以及怎样调整字体的大小，通过右键单击跳出的菜单里有诸多操作。

样例程序 7-1：

```
//项目七 – 温度报警器
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_Tone DFTone;

float sinVal;
int toneVal;
unsigned long tepTimer ;

void setup(){
    pinMode(8, OUTPUT);         // 蜂鸣器引脚设置
    Serial.begin(9600);         //设置波特率为 9600 bps
}

void loop(){
    int val;              //用于存储 LM35 读到的值
    double data;          //用于存储已转换的温度值
    val=analogRead(0);    //LM35 连到模拟口，并从模拟口读值
    data = (double) val * (5/10.24);   // 得到电压值，通过公式换成温度

    if(data>27){          // 如果温度大于 27，蜂鸣器响
          for(int x=0; x<180; x++){
            //将 sin 函数角度转化为弧度
            sinVal = (sin(x*(3.1412/180)));
            //用 sin 函数值产生声音的频率
            toneVal = 2000+(int(sinVal*1000));
```

```
            //给引脚 8 一个蜂鸣器发声信号
            DFTone.play(8,toneVal,2);
          }
      } else {           // 如果温度小于 27，关闭蜂鸣器
            DFTone.play(8,0,2);        //关闭蜂鸣器
      }

    if(millis() - tepTimer > 500){      // 每 500ms，串口输出一次温度值
            tepTimer = millis();
            Serial.print("temperature: ");      // 串口输出“温度”
            Serial.print(data);                 // 串口输出温度值
            Serial.println("℃");                // 串口输出温度单位
       }
}
```

成功下载完程序后，右键单击框选处，并设置串口监视器的波特率为 9600。

单击框选处，打开 Mind+的串口。

然后就可以从串口监视器中，看到 LM35 监测到的温度值输出在屏幕上。

代码学习

这段代码与我们项目六的大部分内容是相同的，代码中的大部分语法在前几项目中已经说过了，现在看起来是不是有点头绪了呢?

程序开始载入了两个库并创建对象，这两者将会在第十课中做详细讲解。

```
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_LM35 LM35;
#include <DFRobot_Libraries.h>
// 创建对象
DFRobot_Tone DFTone;
```

随后设置了三个变量:

```
float sinVal;
int toneVal;
unsigned long tepTimer ;
```

第一、二个变量就不说了，项目六中代码回顾中已作解释，第三个变量 tepTimer，是一个无符号的长整型（unsigned long）用于存放机器时间，便于定时在串口输出温度值，由于机器运行时间较长，所以选用一个长整型，又由于时间不为负，则选用无符号长整型，对于变量类型不明确的，可以再回看下项目三相关解释。

setup()函数的第一句，我们想必已经很熟了，设置蜂鸣器为输出模式，有人可能会问为什么 LM35 不用设置呢？LM35 是个模拟量，模拟量不需要设置引脚模式。pinMode 只用于数字引脚。

第二句是我们即将学习的新内容：串口，其中的相关语句。

```
    Serial.begin(9600);          //设置波特率为 9600 bps
```

Arduino 的通信伙伴——串口

串口是 Arduino 和外界进行通信的一个简单的方法。每个 Arduino 都至少有一个串口，UNO 分别与数字引脚 0(RX)和数字引脚 1(TX)相连。所以如果要用到串口通信的，数字 0 和 1 不能用于输入输出功能。

Arduino 下载程序也是通过串口来完成的。所以，当下载程序的过程中，USB 将占用了数字引脚 0(RX)和数字引脚 1(TX)。此时，RX 和 TX 引脚不能接任何东西,否则会产生冲突。可以下载完之后，再接上。

在以后的使用过程中，需要注意，特别是一些无线通信模块，通常会用到 TX、RX。所以下载程序时，需将模块先取下。避免造成程序下载不进去的情况。

串口可用的函数也有好多，可用查看语法手册。我们这里就先介绍几个常用的:

```
Serial.begin(9600);          //设置波特率为 9600 bps
```

例如代码中的这个函数，它用于初始化串口波特率，也就是数据传输的速率，是使用串口必不可少的函数。直接输入相应设定的数值就可以了，如果不是一些特定的无线模块对波特率有特殊要求的话，波特率设置只需和串口监视器保持一致即可。

再到 loop()函数内部，开始部分又声明了两个变量 val 和 data，注释中已对这两个变量进行说明了，这两个变量与前面声明的两个变量不同的是，这两个是局部变量，只在 loop()函数内部起作用。关于全局变量和局部变量的区别，可以参看一下项目二中说明。

```
val=analogRead(0);  //LM35 连到模拟口，并从模拟口读值
```

这里用到了一个新函数——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。

我们这里读到的是温度的电压值，是以 0~1023 的方式输出。而我们 LM35 温度传感器每 10mV 对应 1 摄氏度。

```
data = (double) val * (5/10.24);  // 得到电压值，通过公式换成温度
```

从传感器中读到的电压值,它的范围在 0~1023,将该值分成 1024 份,再把结果乘以 5,映射到 0~5V,因为每度 10mV，需要再乘以 100 得到一个 double 型温度值，最后赋给 data 变量。

后面进入一个 if 语句，对温度值进行判断。这里的 if 语句与之前讲的有所不同。if...else 用于对两种情况进行判断的时候。

if...else 语句格式:

```
if(表达式){
    语句 1;
} else{
    语句 2;
}
```

表达式结果为真时，执行语句 1，放弃语句 2 的执行，接着跳过 if 语句，执行 if 语句的下一条语句；如果表达式结果为假时，执行语句 2，放弃语句 1 的执行，接着跳过 if 语句，执行 if 语句的下一条语句。无论如何，对于一次条件的判断，语句 1 和语句 2 只能有一个被执行，不能同时被执行。

回到我们的代码, if 中的语句就省略不说了，不明白的可以回看项目六：

```
    if(data>27){
          for(int x=0; x<180; x++){
              ...
          }
     } else {
           ...
     }
```

进入 if 判断，对 data 也就是温度值进行判断，如果大于 27，进入 if 前半段，蜂鸣器鸣响。否则，进入 else 后的语句，关闭蜂鸣器。

除了不断检测温度进行报警，我们还需要代码在串口实时显示温度。这里又用到 millis()函数（项目三中有说明），利用固定的机器时间，每隔 500ms 定时向串口发出数据。

那串口收到数据后，如何在串口监视器上显示呢？就要用到下面的两句语句：

```
            Serial.print(data);                 // 串口输出温度值
            Serial.println("C");                // 串口输出温度单位
```

print()的解释是，以我们可读的 ASCII 形式从串口输出。

这条命令有多种形式：

（1）数字则是以位形式输出（例 1）

（2）浮点型数据输出时只保留小数点后两位（例 2）

（3）字符和字符串则原样输出，字符需要加单引号（例 3），字符串需要加双引号（例 4）。

例如：

（1）Serial.print(78); 输出“78”

（2）Serial.print(1.23456); 输出“1.23”

（3）Serial.print（‘N’）; 输出“N”

（4）Serial.print（“Hello world.”）; 输出“Hello world.”

不仅有我们上面这种形式输出，还可以以进制形式输出，可以参看语法手册。

println()与 print()区别就是，println()比 print()多了回车换行，其他完全相同。

串口监视器输出还有一条语句比较常见的是 Serial.write(),它不是以 ASCII 形式输出，而是以字节形式输出，感兴趣的可以查看语法手册。

代码中，可能有一处会不太明白：

```
Serial.print(data);                    // 串口输出温度值
```

有人会问，data 不是字符串吗？怎么输出是数字呢？不要忘了，这是我们前面定义的变量，它其实就是代表数字，输出当然就是数字啦！

硬件回顾

LM35

LM35 是一种常见的温度传感器，使用简便，不需要额外的校准处理就可以达到+ 1/4℃的准确率。

我们看一下 LM35 引脚示意图，Vs 接入电源，Vout 是电压输出，GND 接地。

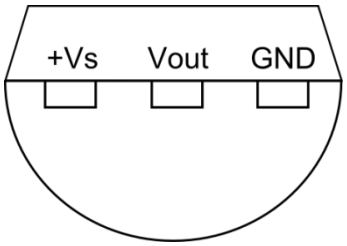


图 7-2 LM35 引脚示意图（从下往上看，引脚朝向身体）

计算公式：

Vout = 10mV/℃ * T℃（温度范围在+2℃~40℃）

这个公式哪里来的呢？如果我们换做其他的温度传感器该怎么改换算呢？可以网上搜索 ALLDATASHEET 网站来查芯片的使用说明书（也叫做 datasheet）。 Datasheet 会提供出厂芯片所有的性能参数，以及一些简单典型电路的搭建也会告诉你。以后碰到其他的传感器，不同的芯片就能通过这个方法来得到计算公式。

我们试一下搜索 LM35，下图公式就是截取自 LM35 的 datasheet 中。图 7-3 显示的就是 LM35 的计算公式。

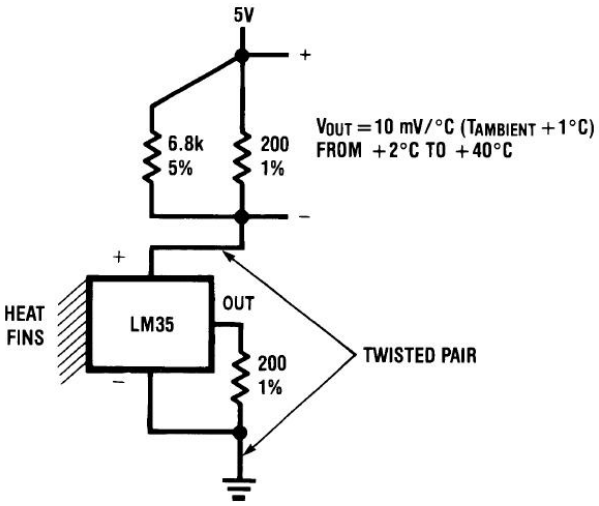


图 7-3 LM35 计算公式

课后作业

1. 在手动编辑中，输入本项目代码。

2. 将我们上面的温度报警器再结合 LED 灯。在不同的温度范围设置不同颜色灯，并伴随不同频率的声音。

比如：温度小于 10 或者大于 35，亮红灯，蜂鸣器发出比较急促的声音。

温度在 25~35 之间，亮黄灯，蜂鸣器伴随相对缓和的声音。

温度在 10~25 之间，亮绿灯，关闭蜂鸣器。

温度报警器可以用于植物种植等对环境温度有要求的一些东西。发挥你的想象，看看还能玩出什么好玩的东西?

项目八 震动传感器

震动传感器，我们从名字中应该就可以判断，传感器能够检测震动中的物体。我们用什么来做震动传感器呢？那就是滚珠开关。滚珠开关，其内部含有导电珠子，器件一旦震动，珠子随之滚动，就能使两端的导针导通。

通过这个原理，我们可以做一些小玩具结合起来。最常见的，比如我们看到一些小孩子穿的一闪一闪的小鞋子！走动的过程，就能使内部珠子滚动。

只要传感器检测到东西震动，就会有信号输出。这里，我们想通过滚珠开关做个简单的震动传感器，并把震动传感器和 LED 的结合，当传感器检测到物体震动时，LED 亮起，停止震动时，LED 关闭。

所需元件

- 1× 滚珠开关 SW200D 
- 1× 5mm LED 灯 
- 1× 220 欧电阻 

硬件连接

从滚珠开关这个名字，我们可以把它和什么联想在一起呢？就是按键开关，滚珠开关和我们项目三中介绍的按钮在硬件连接是完全相同的，原理也相似。只是使用方法不同而已。可以把下图对应项目三的一起看，你会发现很多相似之处。滚珠开关也需要一个下拉电阻，LED 需要一个限流电阻。

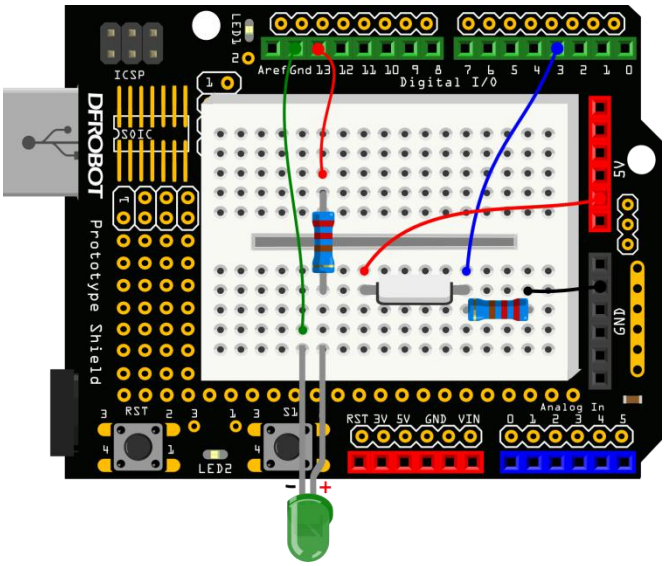


图 8-1 震动传感器连线图

代码编程

输入样例代码 8-1

```
//项目八 – 震动传感器
int SensorLED = 13;              //定义 LED 为数字引脚 13
int SensorINPUT = 3;             //连接震动开关到中断 1，也就是数字引脚 3
Volatile unsigned char state = 0;
void blink();                    //使用任何函数前，必须先声明

void setup() {
    pinMode(SensorLED, OUTPUT);          //LED 为输出模式
    pinMode(SensorINPUT, INPUT);         //震动开关为输入模式

    //低电平变高电平的过程中，触发中断 1，调用 blink 函数
    attachInterrupt(1, blink, RISING);
}

void loop(){
    if(state!=0){                                // 如果 state 不是 0 时
          state = 0;                             // state 值赋为 0
          digitalWrite(SensorLED,HIGH);          // 亮灯
          delay(500);                            //延时 500ms
    }
    else{
          digitalWrite(SensorLED,LOW);  //否则，关灯
    }
}

void blink(){          //中断函数 blink()
    state++;           //一旦中断触发，state 就不断自加
}
```

当我们晃动板子时，LED 灯也会随之亮，一旦停止晃动，LED 灯又恢复到熄灭的状态。

代码学习

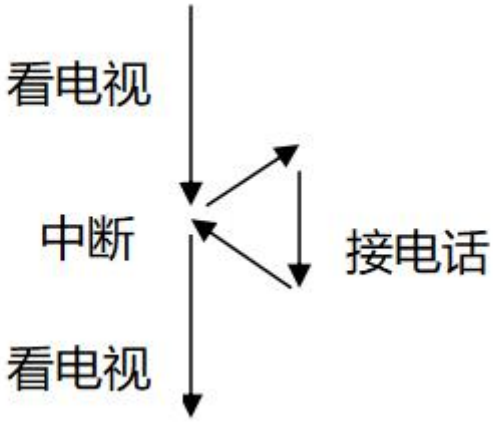
代码虽不长，但还是不太容易理解的。先大致说下代码的运行过程。

在没有任何打扰的情况下，程序在不断运行着…，让 LED 一直处于关闭。突然，被人打扰了（也就是晃动板子），就跳到中断函数 blink()中（当然进入中断也是要条件的，我们后面说）。此时，state 不断自加，产生连锁反应，主函数中 if 函数检测到 state 不为 0 了，那么就让 LED 亮起了，同时又重新让 state 为 0，等待下一次中断。如果没有中断的话，LED 又恢复到关闭的状态。

简单说了下程序的运行过程，重复的知识点就不做说明了。就重点说下中断函数 attachInterrupt()。

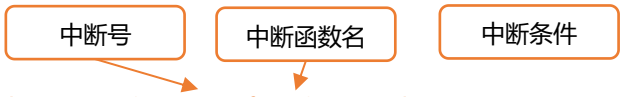
中断

什么是中断？打个比方吧，比如你在家好好的看电视，突然家里电话铃响了，那么你不得不停下看电视，先去接电话，等接完电话后，你又可以继续看电视啦！在整个过程中，接电话就是一个中断过程，电话铃响就是中断的标志，或者说是中断条件。



现在知道中断是什么意思了，再回到 attachInterrupt()函数，它是一个当外部发生中断时，才被唤醒的函数。区别于其他函数，它依附于中断引脚才发生。Arduino UNO 板有两个外部中断引脚：数字引脚 2（中断 0）和数字引脚 3（中断 1）。中断 0 与中断 1 是中断号，在函数中需要用到。不同板子，中断号对应引脚可能不同，可以查阅 Arduino 官方编程语法手册。

attachInterrupt() 需要三个传递参数：



attachInterrupt(interrupt, function, mode)

interrupt：中断号 0 或者 1。

如果选择 0 的话，连接到数字引脚 2 上，选择 1 的话，连接到数字引脚 3 上。

function：调用的中断函数名。

写中断函数时，需要特别说明以下三点：

我们写中断函数的时候，该函数不能含有参数和返回值。也就是说，要是一个无返回值的函数。

中断函数中不要使用 delay()和 millis()函数，因为数值不会继续变化。

中断函数中不要读取串口，串口收到的数据可能会丢失。

mode：中断的条件。

只有特定的以下四种情况：

LOW：当引脚为低电平时，触发中断。

CHANGE：当引脚电平发生改变时，触发中断。

RISING：当引脚由低电平变为高电平时，触发中断。

FALLING：当引脚由高电平变为低电平时，触发中断。

知道了 attachInterrupt()函数的用法，回归到我们的代码中：

attachInterrupt(1, blink, RISING);

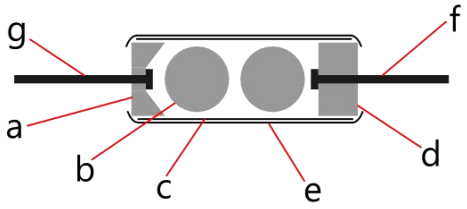
对应上面说明看：1，指中断号 1。所以滚珠开关接到数字引脚 3；blink 是我们下面要调用的中断函数；RISING，指引脚 3 在由低变为高的一瞬间，中断触发。

为什么要选 RISING 呢？由于硬件我们还没提到，我们就先滚珠开关想象成按键。在按键没按下的时，是断开的，引脚 3 处于低的状态。一旦被按下，就和 5V 导通，变为高。这个过程是引脚由低电平变高电平的过程，所以选择 RISING 模式。

硬件回顾

滚珠开关

滚珠开关，也叫做珠子开关，震动开关等等。虽然叫法不同，不过原理是相同的。就是通过珠子滚动接触导针的原理来控制电路的通断。看下结构图就明白了。



a. 青铜盖
b. 青铜珠子
c. 青铜管
d. PC胶座
e. 热缩管
f. 青铜导针
g. 磷铜弹簧夹

图 8-2 滚珠开关内部结构图

滚珠开关内部两个珠子，通过珠子滚动接触导针的原理来控制电路的接通或者断开。传感器震动或者晃动时，珠子就会接触导针，从而导通。还需要注意的一点是，由于滚珠开关的内部构造，滚珠开关只有一头是导通的，金色导针一端是导通的，银色导针一端是不导通的。这也就是为什么，往金色一端倾斜，灯会点亮，而偏向银色一端倾斜时，灯不会被点亮的原因。

项目九 感光灯

这个项目中将介绍一个新元件——光敏电阻。从名字可以看出，这个器件是依赖光作用的。在黑暗的环境中，光敏电阻具有非常高阻值的电阻。光线越强，电阻值反而越低。通过读取这个电阻值，就可以检查光线的亮暗了。我们这里选用的是光敏二极管，光敏二极管其实就是光敏电阻中的一种，只是它还具有正负极性。

我们这次做的这个非常好玩，叫做感光灯。它能随着光线明暗而选择是否亮灯。这个光感灯非常适合用做夜晚使用的小夜灯。晚上睡觉的时候，家中灯关掉后，感光灯感觉到周围环境变暗了，就自动亮起。到了白天，天亮后，感光灯就又恢复到关闭的状态了。

所需元件

- 1× 光敏二极管 
- 1× 5mm LED 灯 
- 1× 220 欧电阻 
- 1× 10k 欧电阻 

硬件连接

LED 灯还是和以往一样的接法。而光敏二极管是有正负极的，和 LED 一样，也是遵循长脚（+），短脚（-）的原则。还需注意的与光敏二极管相连的电阻是 **10k**，而不是 220Ω。

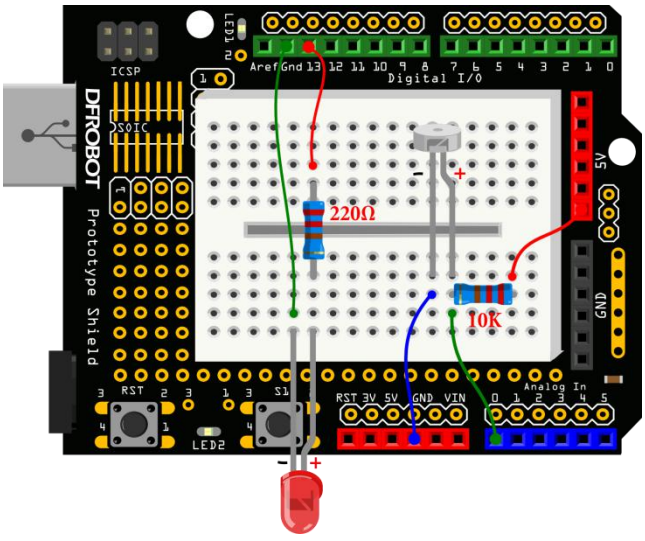


图 9-1 感光灯的接线图

代码编程

输入样例代码 9-1：

```
//项目九 – 感光灯
int LED = 13;                          //设置 LED 灯为数字引脚 13
int val = 0;                           //设置模拟引脚 0 读取光敏二极管的电压值

void setup(){
     pinMode(LED,OUTPUT);              // LED 为输出模式
     Serial.begin(9600);               // 串口波特率设置为 9600
}

void loop(){
     val = analogRead(0);              // 读取电压值 0~1023
     Serial.println(val);              // 串口查看电压值的变化
     if(val<1000){                     // 一旦小于设定的值，LED 灯关闭
          digitalWrite(LED,LOW);
     }else{                            // 否则 LED 亮起
          digitalWrite(LED,HIGH);
     }
     delay(10);                        // 延时 10ms
}
```

下载完代码后，LED 灯会亮起，这时，你需要拿一个手电筒照你的光敏二极管（用手机后置摄像头的闪光灯应该也可以），这时你会发现 LED 灯神奇般的自动熄灭。但是，一旦你的手电筒移开，LED 灯又再次亮起。

代码学习

这段代码想必你一定能看的懂了吧？我就简单说一下，可能不明白的地方。

我们之前在项目七中讲 LM35 温度传感器的时候，也用到了用模拟口读值。强调了模拟量不需要输入输出模式。这里，也是同样用模拟口用来读取光敏二极管的模拟值。

一旦有光照射，读出的模拟值就会减小，这里设定的上限值是 1000。这个值可以按你需要的亮度来选取。选取方法：先把整个装置放在你想让 LED 关闭的一个环境下，然后打开串口，查看串口显示的值，把这个值替换掉代码中的 1000。

从串口监视器读值，是调试代码一种很好的方法。

硬件回顾

光敏二极管

这里接触了一种新元件——光敏器件。这类器件都是将光信号变成电信号的特殊电子元件。元件内部有特殊的光导材料，外部用塑料或者玻璃封装。光线照射在这类光导材料上时，光敏器件的电阻值就会迅速变小。光敏元件有很多，光敏电阻，光敏二极管，光敏三极管等等。不过原理是差不多的。我们这里选用的是光敏二极管。

光敏二极管其实是光敏电阻中的一种。所谓二极管，就是有正负极的，所以在连线的时候也要注意正负极。

光敏电阻在黑暗的环境中，具有非常高阻值的电阻。光线越强，电阻值反而越低。随着两端电阻值的减小，电压也就相应减小（从模拟口读到的值也就变小，模拟口 0~1023 的值对应是 0~5V 的电压值）。

那电压为什么会减小呢？那就要用到我们初中学的物理知识——分压原理。让我们看一个典型的分压电路，看看它是如何工作的。

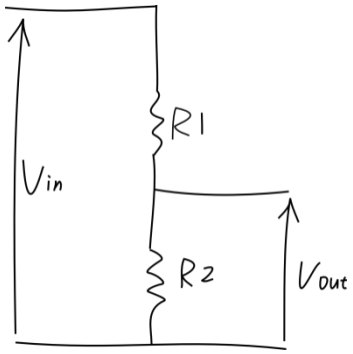


图 9-2 分压电路图

输入电压 Vin（我们这里也就是 5V），连在两个电阻上，只测量通过电阻 R2 的电压 Vout，其电压将小于输入电压。计算 R2 两端的 Vout 电压公式是：

$$V_{out} = \frac{R2}{R1+R2} \times V_{in}$$

图 9-3 分压公式

在我们这项目中，R1 代表的就是 10k 电阻，R2 代表的就是光敏二极管。本来 R2 在黑暗中，值很大很大，所以 Vout 也就很大，接近 5V。一旦有光线照射的话，R2 的值就会迅速减小，所以 Vout 也就随之减小了，读取的电压值就小。通过上面这个公式可以看出， R1 选取不能太小，最好在 1k~10k 左右，否则比值变化不明显。

项目十 舵机初动

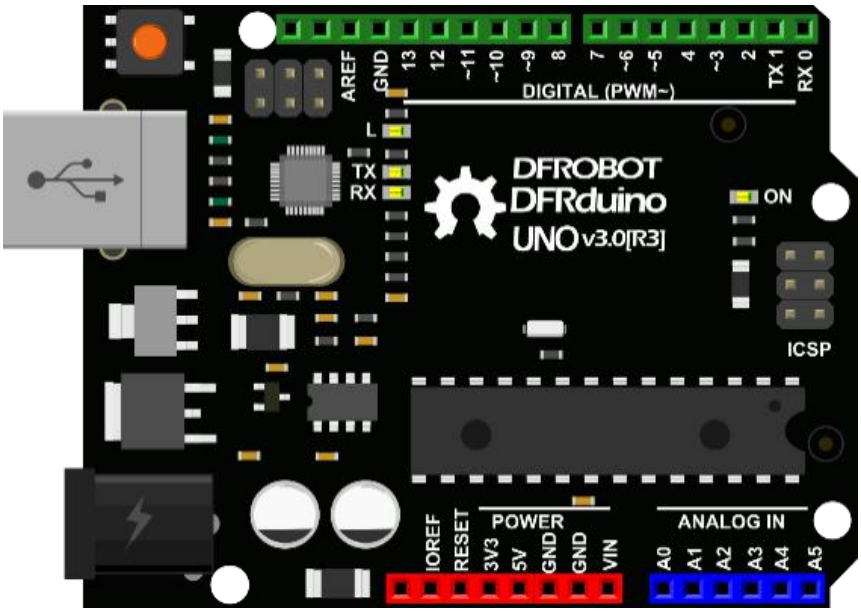
本项目要接触到舵机。舵机是一种电机，它使用一个反馈系统来控制电机的位置。可以很好掌握电机角度。大多数舵机是可以最大旋转 180°的。也有一些能转更大角度，甚至 360°。舵机比较多的用于对角度有要求的场合，比如摄像头，智能小车前置探测器，需要在某个范围内进行监测的移动平台。又或者把舵机放到玩具，让玩具动起来。还可以用多个舵机，做个小型机器人，舵机就可以作为机器人的关节部分。所以，舵机的用处很多。

Ardruino 也提供了<Servo.h>库，让我们使用舵机变得更方便了。

先从简单入手，套件这个 9G 小舵机是 180°的，我们就让它在 0~180°之间来回转动。

所需元件

- 1× DFduino UNO R3（以及配套 USB 数据线）



- 1× Micro Servo 9g

硬件连接

这个项目的连线很简单，只需按图 10-1 所示连接舵机三根线就可以了，连的时候注意线序，舵机引出三根线。一根是红色，连到+5V 上。一根棕色（有些是黑的），连到 GND。还有一根是黄色或者橘色，连到数字引脚 9。

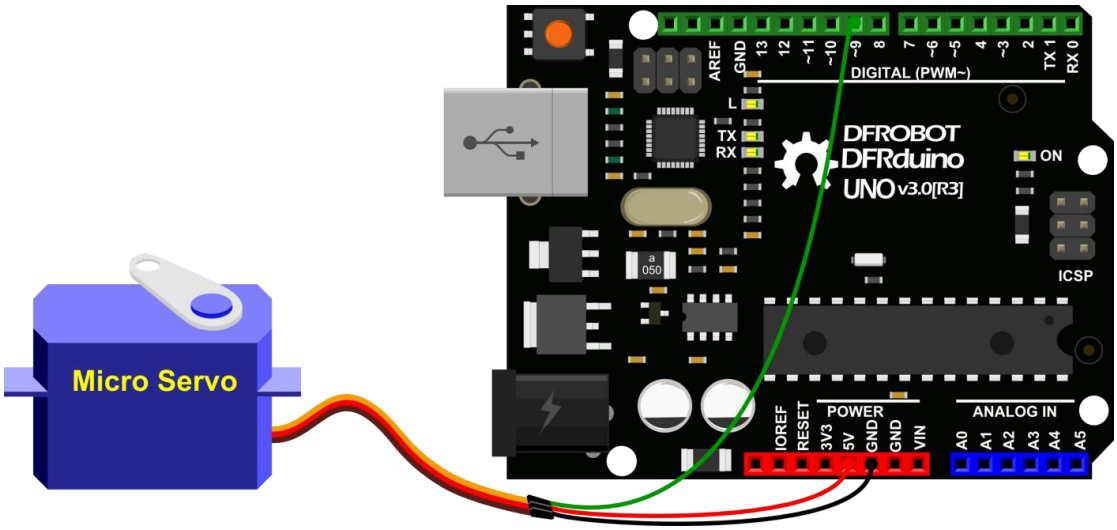


图 10-1 舵机连线图

代码编程

样例程序 10-1 ：

```
//项目十 - 舵机
#include <DFRobot_Servo.h>              // 声明调用 Servo.h 库
Servo myservo;                          // 创建一个舵机对象
int pos = 0;                            // 变量 pos 用来存储舵机位置
void setup() {
    myservo.attach(9);                  // 将引脚 9 上的舵机与声明的舵机对象连接起来
}

void loop() {
    for(pos = 0; pos < 180; pos += 1){  // 舵机从 0°转到 180°，每次增加 1°
          myservo.angle(pos);           // 给舵机写入角度
          delay(15);                    // 延时 15ms 让舵机转到指定位置
    }
    for(pos = 180; pos>=1; pos-=1) {    //舵机从 180°转回 0°，每次减小 1°
          myservo.angle(pos);           // 写角度到舵机
          delay(15);                    // 延时 15ms 让舵机转到指定位置
    }
}
```

代码的第一行是我们在本项目中需要学习关于库的知识。这行指令我们未曾接触过，但是在 Mind+中我们能够通过以下步骤来熟悉和舵机有关的代码。

STEP1：进入扩展，点击执行器

STEP2：在界面右上角的搜索中输入“舵机”

STEP3：选择项目中所要求的 0-180°舵机

STEP4：回到主界面，将新加载舵机的指令拖入“手动编辑”的输入框中，即可看见它对应的代码

所有的指令都可以通过拖进手动编辑区，对应的 c 代码会出现。在之后遇到新指令的代码学习，或是一时忘记指令对应的指令，都可以通过这一方式来查看和温故代码。并且这些代码都可以直接复制到手动编辑区中，但别忘记根据连接图来修改相关参数！

下载代码，下载成功后我们可以看到舵机 0~180°来回转动。

图形化指令学习

所属模块	指令	功能
执行器	设置 11 引脚伺服舵机为 90 度	设置舵机角度指令

代码学习

代码的开始先调用<DFRobot_Servo.h>库

```
1. #include <DFRobot_Servo.h>          // 声明调用 Servo.h 库
```

这库已经在 Mind+中，打开 Mind+1.6.1\Arduino\libraries\DFRobot_Servo\DFRobot_Servo.h，这就是 Servo 库所在位置。

库是什么?

我们怎么理解库呢？和我们前面讲到的函数意义是差不多的。函数通常按一个个功能来划分的，就像一个个小的储物柜，函数名好比储物柜标签名。我们使用的时候，直接看标签就好了，方便我们使用。那库是什么呢？库则是把多个函数封装打包起来，好比大的储物柜，里面含有一个个小的储物柜。不知道这样说，你是不是能理解库和函数的关系？

对象

同样，大储物柜也需要一个标签，这标签的学术名叫做“**对象**”。所以这里叫创建一个对象。就是我们接下来的这句语句：

```
Servo myservo;          // 创建一个舵机对象
```

库中函数的调用

setup()函数中有一条语句：

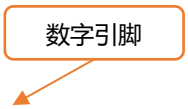
```
myservo.attach(9);
```

这里就开始调用 Servo 库中的函数了，和我们以前函数调用有点区别。这里，我们需要先指明这是哪个库中的函数。所以，先指出对象名，再指出函数名。每次要用到储物柜的东西就要先指明这个标签。这样程序才知道要去哪里找东西。

库函数调用格式如下：

对象名.函数名();

不要忘了中间的“.”！ myservo 是我们前面设的标签（对象），然后调用的函数是：



attach（pin）；

attach(pin)函数有一个传递参数——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。

进入主函数，有两个 for 循环，第一段是从 0 开始，循环到 180，每次增加 1 度。第二个 for 循环则是从 180 开始，每次减小 1 度，一直减到 0。在回到上面那个循环中……

for 循环中又调用了一个 Servo 库中的函数 myservo.angle(pos)，我们可以不用管函数内部复杂的程序，只要先会使用就可以了。

myservo.angle(pos);

和上面那个函数调用一样，先要指明是哪个库。该函数的传递参数就是角度，单位为“°”。

如果还想了解 DFRobot_Servo 库中还有哪些好用的函数的话，可以到 Mind+官网中的教程和论坛查看，里面会有相关介绍的。

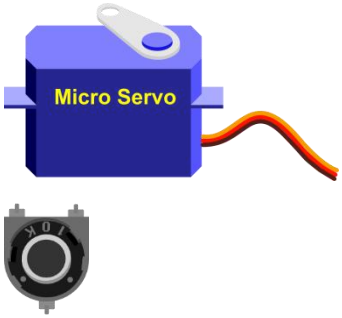
我们这里对舵机的硬件部分就不做详细说明了，先学会简单的使用即可。如果还想了解更多的话，可以借助我们的网络资源。

DF 创客社区：www.dfrobot.com.cn

项目十一 可控舵机

在前面一个项目中，我们知道了如何让舵机动起来，这里将进一步的通过外部信号来让舵机随着输入的改变来相应改变角度，方便做一些可控的转动装置。我们这里通过一个可变电阻——电位器，来控制舵机。当然你也可以通过其他的模拟量或者数字量来控制舵机。模拟量的话，比如改造一下前面的感光灯，变成一个会动的感光灯。数字量的话，比如通过一个按钮，倾斜开关等等，一旦触发开关，就让舵机转动，可以有很多玩法。再给舵机加个外壳，让它更具生命力。

所需元件



- 1× Micro Servo 9g
- 1× 10K 电位器

硬件连接

与前面一节不同处在于多了一个电位器，电位器相当于一个可变阻值的电阻，两个引脚的一边分别接 5V 与 GND，而另一边只有单独一个引脚的接模拟口 0，用于做输入信号。

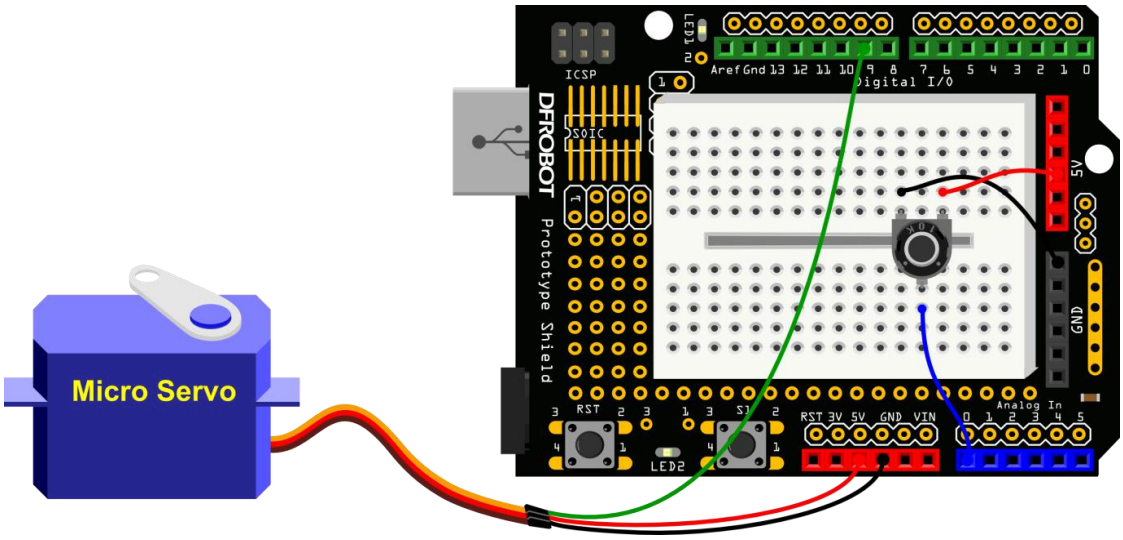


图 11-1 可控舵机连线图

代码编程

样例程序 11-1

```
//项目十一 - 可控舵机
#include <DFRobot_Servo.h>              //声明调用 Servo.h 库
Servo myservo;                          //创建一个舵机对象

int potpin = 0;                         //连接到模拟口 0
int val;                                //变量 val 用来存储从模拟口 0 读到的值

void setup() {
    myservo.attach(9);                  //将引脚 9 上的舵机与声明的舵机对象连接起来
}

void loop() {
    val = analogRead(potpin);           //从模拟口 0 读值，并通过 val 记录
    val = map(val, 0, 1023, 0, 180);    //通过 map 函数进行数值转换
    myservo.angle(val);                 //给舵机写入角度
    delay(15);                          //延时 15ms 让舵机转到指定位置
}
```

下载代码，成功后，旋转电位器，看看舵机是不是随着电位器转动。

代码学习

代码的开始部分还是需要调用<DFRobot_Servo.h>库，并创建相应的对象。同时，需要一个模拟口用来读取电位器的值，我们这里用变量 potPin 代表模拟口 0。

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。在 Mind+ 中对应指令： 

。输入代码时，如果不那么熟悉，可以通过拖动到指令区，进行查看。

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);

```
  val = map(val, 0, 1023, 0, 180);      //通过 map 函数进行数值转换
```

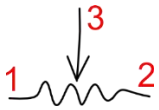
所以，回到代码中，我们是想将模拟口读到的 0~1023 的值，转换为舵机的 0~180°。

硬件回顾

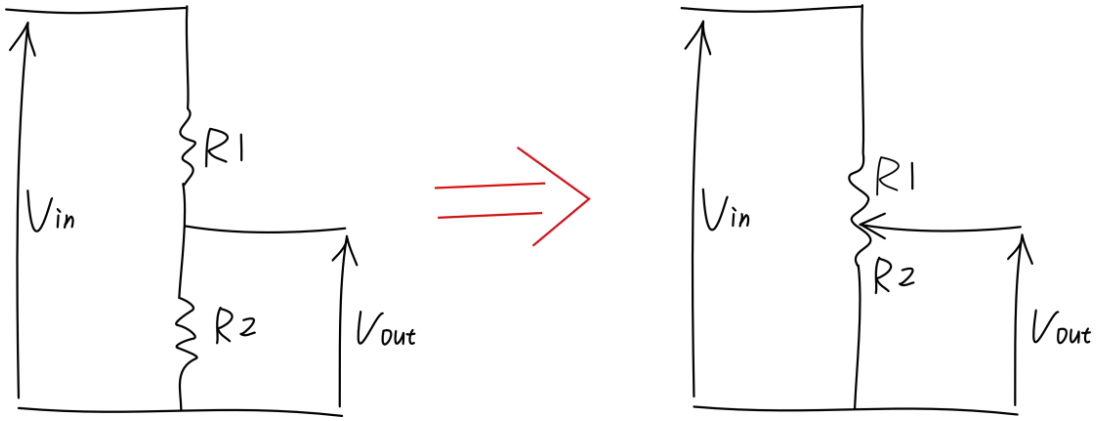
电位器

电位器可以理解为个电阻，只是这个电阻阻值可变。我们这里可调节的范围是 0~10KΩ。电阻两端接电源，通过中间引脚调节阻值，随着电阻值的改变而带动电压变化。我们用模拟口 0 读取到这个变化中的电压值，并转换为对应的舵机的角度值。这就是整个的控制过程。

电位器在电路上的表示的图标为下图，分别对应器件上的 3 个引脚。



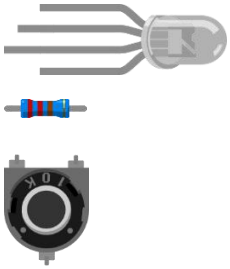
简单的看下原理，不知道还记不记得在第九个项目中讲到的分压原理。电位器用的同样是分压原理。我们可以理解为，电位器被拆分为上下两个电阻 R1 和 R2，随着转动电位器，上下阻值发生变化，从而对应的输出电压就不同。我们可以想象成切蛋糕，分到的蛋糕越多（电阻），吃下去的能量（电压 Vout）也就越大。电压值大小的变化可以直接通过模拟口读到的值（0~1023）反应出来。



项目十二 彩灯调光台

在项目五的时候，我们已经接触过 RGB LED 了，可以实现变色，这回儿我们需要加入互动元素进去。通过三个电位器来任意变换对应的 R、G、B，组合成任何你想要的颜色，在家做个心情灯吧，随心情任意切换。

所需元件



- 1× 5mm RGB LED 灯
- 1× 220 欧电阻
- 3× 10K 电位器

硬件连接

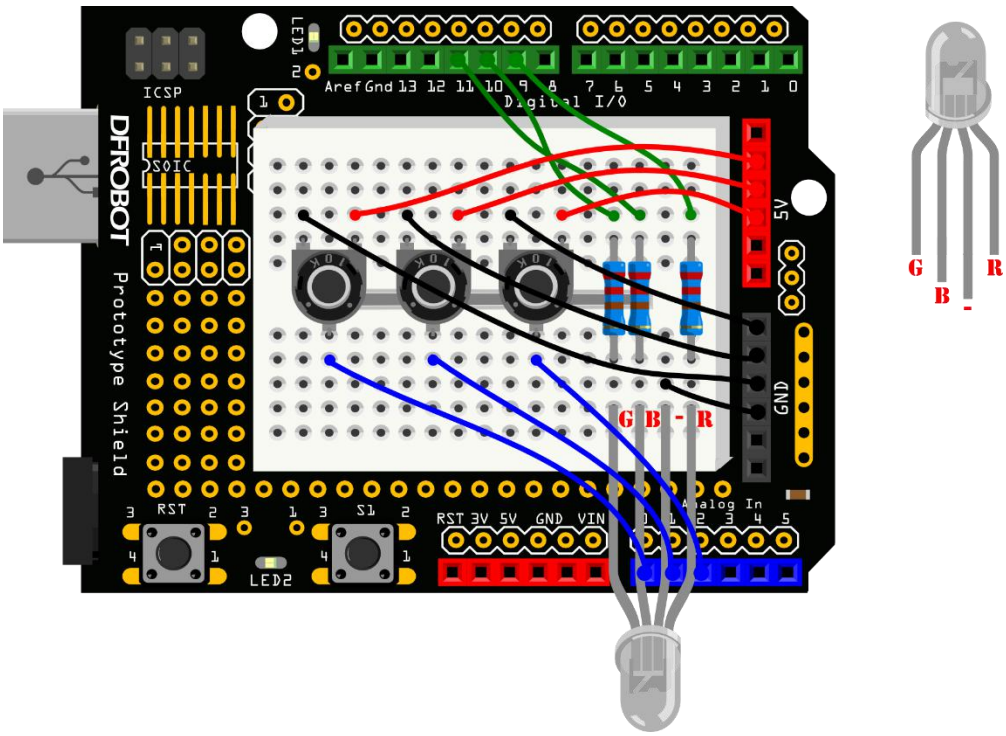


图 12-1 彩灯调光台连线图

代码编程

样例程序 12-1

```
//项目十二 –彩灯调光台
int redPin = 9;                    // R – digital 9
int greenPin = 10;                 // G – digital 10
int bluePin = 11;                  // B – digital 11
int potRedPin = 0;                 // 电位器 1 – analog 0
int potGreenPin = 1;               // 电位器 2 – analog 1
int potBluePin = 2;                // 电位器 3 – analog 2
void colorRGB(int, int, int);      // 函数声明
void colorRGB(int red, int green, int blue){     //该函数用于显示颜色
   analogWrite(redPin,constrain(red,0,255));
   analogWrite(greenPin,constrain(green,0,255));
   analogWrite(bluePin,constrain(blue,0,255));
}

void setup(){
   pinMode(redPin,OUTPUT);
   pinMode(greenPin,OUTPUT);
   pinMode(bluePin,OUTPUT);
   Serial.begin(9600);                    // 初始化串口
}

void loop(){
   int potRed = analogRead(potRedPin);       // potRed 存储模拟口 0 读到的值
   int potGreen = analogRead(potGreenPin);   // potGreen 存储模拟口 1 读到的值
   int potBlue = analogRead(potBluePin);     // potBlue 存储模拟口 2 读到的值

   int val1 = map(potRed,0,1023,0,255);      //通过 map 函数转换为 0~255 的值
   int val2 = map(potGreen,0,1023,0,255);
   int val3 = map(potBlue,0,1023,0,255);
   //串口依次输出 Red，Green，Blue 对应值
   Serial.print("Red:");
   Serial.print(val1);
   Serial.print("Green:");
   Serial.print(val2);
   Serial.print("Blue:");
   Serial.println(val3);
   colorRGB(val1,val2,val3);                 // 让 RGB LED 呈现对应颜色
}
```

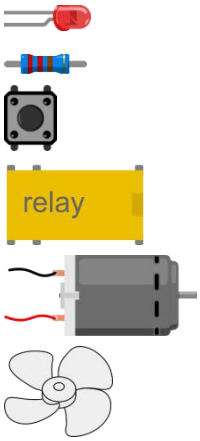
下载代码，旋转三个电位器，可以变化出不同的颜色。

项目十三 自制风扇

这次，我们会做一个小风扇。同时会接触两件新元件——继电器、直流电机。继电器，我们可以理解为是用较小的电流去控制较大电流的一种“自动开关”。在这里，继电器是用来控制电机转动的。

所需元件

- 1× 5mm LED 灯
- 2× 220 欧电阻
- 1× 按钮
- 1× 继电器 HRS1H-S -DC5V
- 1× 小电机
- 1× 风扇叶片



硬件连接

按下图进行连线，按钮接线与项目三类似，连接到数字 2。按钮一端连接 5V，另一端连接 GND，并用一个 220Ω的电阻作为下拉电阻，以防引脚悬空干扰。继电器有 6 个引脚，分别标有序号。1、2 引脚为继电器的输入信号，分别接 Arduino 的数字引脚和 GND。3、4、5、6 为继电器输出的控制引脚，这里只使用 4、6 两个引脚。我们把继电器想成一个开关，开关也只要用到两个引脚。

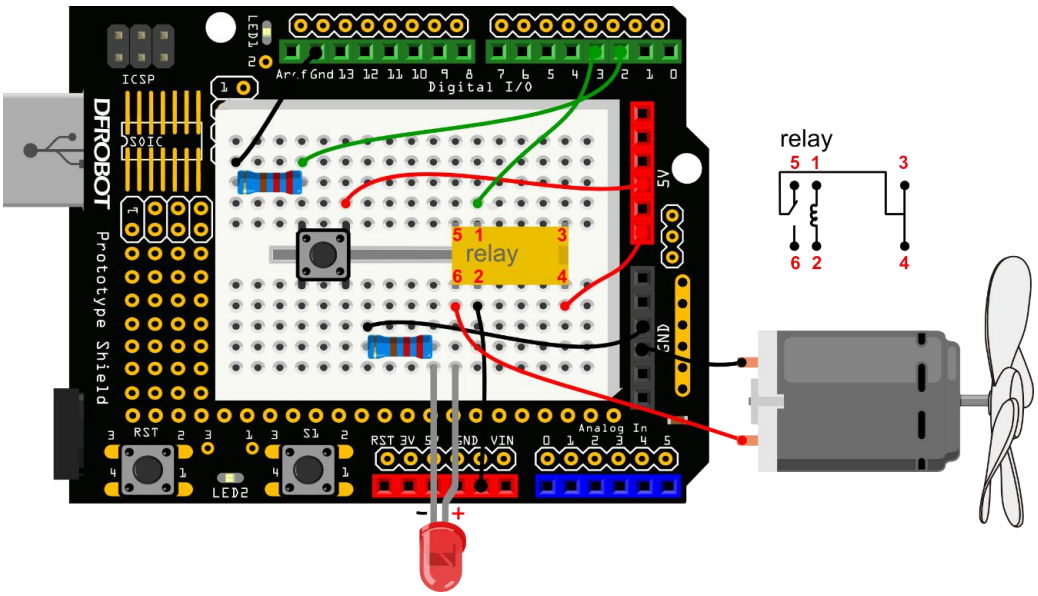


图 13-1 自制风扇连线图

代码编程

样例程序 13-1

```
//项目十三 – Arduino 控制风扇转动
int buttonPin = 2;                          // button 连接到数字 2
int relayPin = 3;                           // 继电器连接到数字 3
int relayState = HIGH;                      // 继电器初始状态为 HIGH
int buttonState;                            // 记录 button 当前状态值
int lastButtonState = LOW;                  // 记录 button 前一个状态值
long lastDebounceTime = 0;
long debounceDelay = 50;                    //去除抖动时间

void setup() {
    pinMode(buttonPin, INPUT);
    pinMode(relayPin, OUTPUT);
    digitalWrite(relayPin, relayState);     // 设置继电器的初始状态
}
void loop() {
    int reading = digitalRead(buttonPin);   //reading 用来存储 buttonPin 的数据

    // 一旦检测到数据发生变化，记录当前时间
    if (reading != lastButtonState) {
          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），那么就改变继电器的状态
    if ((millis() - lastDebounceTime) > debounceDelay) {
          if (reading != buttonState) {
               buttonState = reading;

               if (buttonState == HIGH) {
                    relayState = !relayState;
               }
          }
    }
    digitalWrite(relayPin, relayState);

    // 改变 button 前一个状态值
    lastButtonState = reading;
}
```

通过按键，可以控制电机和 LED 的开和关。

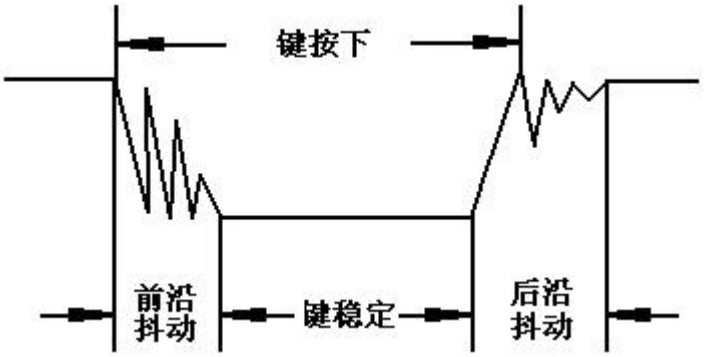
代码学习

代码的大部分内容，基本应该没有什么难度了，主要说下按键去抖问题。代码中：

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

reading 有变化之后，不是立马就采取相应的行动，而是先“按兵不动”，先看看这个信号是不是“错误信号”，所以再等待一阵，（也就是通过 millis 来实现这个等待过程的），发现确实是前方发过来的正确信号，然后执行相关动作。

之所以这么做的原因是，按键在被按下时，会有个抖动的过程，而不是立马由低变高，或者由高变低。所以这个过程中，可能会产生错误信号，我们通过程序中的这种方法，来解决硬件上的这个问题。



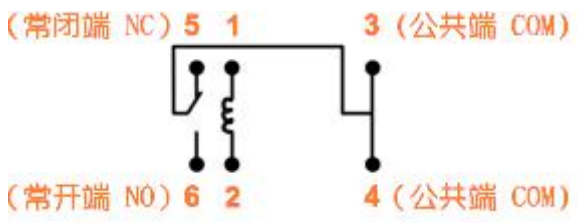
按键抖动 示意图

硬件回顾

继电器

我们可以把继电器理解为一个“开关”，实际上是用比较小的电流去控制较大电流的“开关”。这里只是为了让初学者了解继电器工作原理，所以没有使用较大的电源器件，还是选用是需要 5V 就能驱动的直流电机。

我们来看下继电器的内部构造：



这款继电器一共有 6 个引脚。1,2 引脚是用来接 Arduino 数字引脚和 GND。通过数字引脚来驱动继电器。1，2 两端为线圈两端。Arduino 给 HIGH 后，线圈中就有电流，线圈就会产生磁性（就像磁铁一样），吸合中间的触片（能听到“哒”一声），常开端（NO）就与公共端导通。相反，如果 Arduino 给 LOW，线圈中没有电流，常闭端（NC）就与公共端导通。

所以，电路中我们接了 4,6 引脚用于控制电机和 LED 的通断，（当然也可以用引脚 3,6）。

直流电机、直流减速电机与舵机的区别

普通直流电机是我们接触比较多的电机。一般只有两个引脚，上电就能转，正负极反接则反向转动。如你所见，它做着周而复始的圆周运动，无法进行角度的控制，不过可以通过电机驱动板，可以对转速进行控制，不过由于普通电机转速过快，所以，一般不直接用在智能小车上。

直流减速电机是在普通电机加上了减速箱，这样便降低了转速，使得普通电机有的更广泛的使用空间，比如可以用于智能小车上。同样也可以通过 PWM 来进行调速。

舵机也是一种电机，它使用一个反馈系统来控制电机的位置，可以用来控制角度。所以，舵机经常用来控制一些机器人手臂关节的转动。

项目十四 红外遥控灯

这节我们会接触一个新的元件——红外接收管。所谓红外接收管，也就是接收红外光的电子器件。红外接收管，看着离我们很遥远的感觉！其实不然，它就在我们身边。比如我们电视机，空调这些家电，其实它们都需要用到红外接收管。我们都知道遥控器发射出来的都是红外光，电视机上势必要有红外接收管，才能接收到遥控器发过来的红外信号。

我们这次就用红外接收管做个遥控灯，通过遥控器的红色电源键来控制 LED 的开关。

在开始遥控灯之前，我们先来个预热实验，通过串口来了解下如何使用红外接收管和遥控器。

预热实验：

所需元件

- 1× 红外接收管

- 1× Mini 遥控器

硬件连接

看着是不是很高兴，这应该是我们看到最容易的连线了，只需要连接三根线就可以了，注意一下正负就可以了（图中表明部分）。红外接收管 Vout 输出接到数字引脚 3。

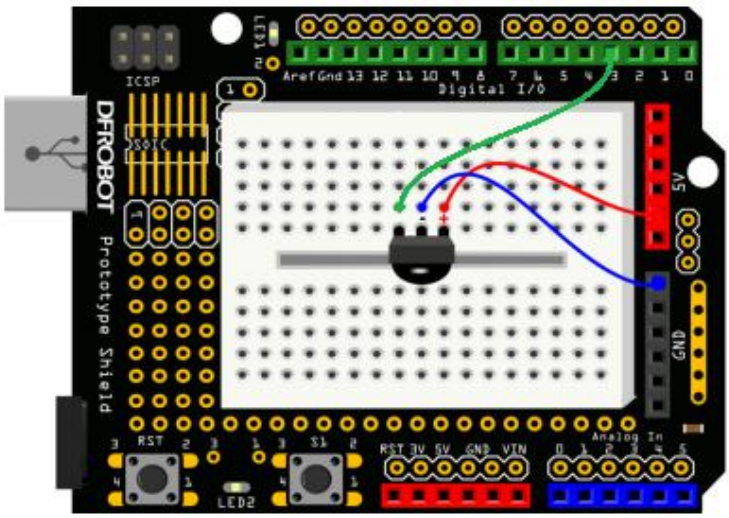


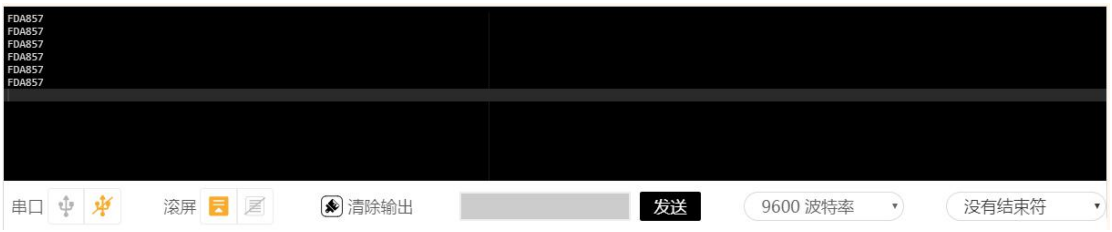
图 14-1 红外接收管连线图

代码编程

样例程序 14-1

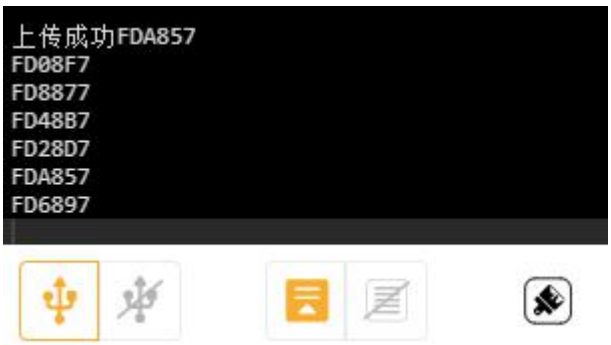
```
#include <DFRobot_IRremote.h>          //载入红外库
String result;                         // 创建变量
IRremote_Receive remoteReceive_3;      // 创建对象

// 主程序开始
void setup() {
    Serial.begin(9600);                // 设置串口波特率
    remoteReceive_3.begin(3);          //启动红外解码
}
void loop() {
    result = (remoteReceive_3.getIrCommand());      //将红外读取值存入变量中
    if ((result!="0")) {                            //判断接受值是否不为零
        Serial.println(result);                     //串口输出有效读取值
    }
}
```

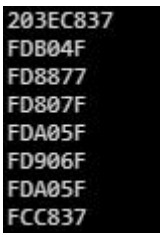
下载完成后，在串口区，设置波特率为 9600，与代码中 Serial.begin(9600)相匹配，并打开串口。这些操作在项目七中已有教学，若还不熟悉，可以复习下之前的章节。



*我们还可以将代码区和串口区的对话框拉宽，隐藏在按钮中的波特率就会显现出来。

设置完后，用 Mini 遥控器的按钮对着红外接收管的方向，任意按个按钮，我们都能在串口监视器上看到相对应的代码。如下图所示，按数字“0”，接收到对应 16 进制的代码是 FD30CF。每个按钮都有一个特定的 16 进制的代码。





像图中的出现“203EC837”和“FCC837”都是因为传输时被干扰或因为遥控器没有对准接收管的原因而产生的错误信号。

*此外在 Mind+中，库规定红外接收管只能连接 2、3 引脚。如遇问题可以在官方文档的常见问题中查找，或是论坛发帖询问。

上面这段代码有一个知识点需要着重讲解

```
result = (remoteReceive_3.getIrCommand());  //将红外读取值存入变量中
```

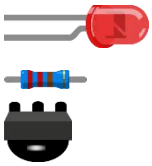
Mind+中 remoteReceive_3.getIrCommand()这一指令在读取一次红外接收管后，会将数据以字符串的类型储存在寄存器中，且在被使用过后，寄存器中就会清除这一指令储存的数据，所以我们无法多次使用这数据，故而需要建立一个新的字符串变量来存放和反复使用它。

由于红外解码较为复杂，其中具体如何实现就暂时不做讲解。所幸的是，高手把这些难的工作已经做好了，提供给我们这个 DFRobot_IRremote 库，我们只需要会用就可以了，先不需要弄明白函数内部如何工作的。要用的时候，把代码原样搬过来就好了。照猫画虎，先用起来再说~

预热完之后，我们言归正传，开始制作遥控灯。

所需元件

- 1× 5mm LED 灯
- 1× 220 欧电阻
- 1× 红外接收管
- 1× Mini 遥控器



硬件连接

其实就是在原有的基础上，加了个 LED 和电阻，LED 使用的是数字引脚 10。红外接收管仍然接的是数字引脚 3。

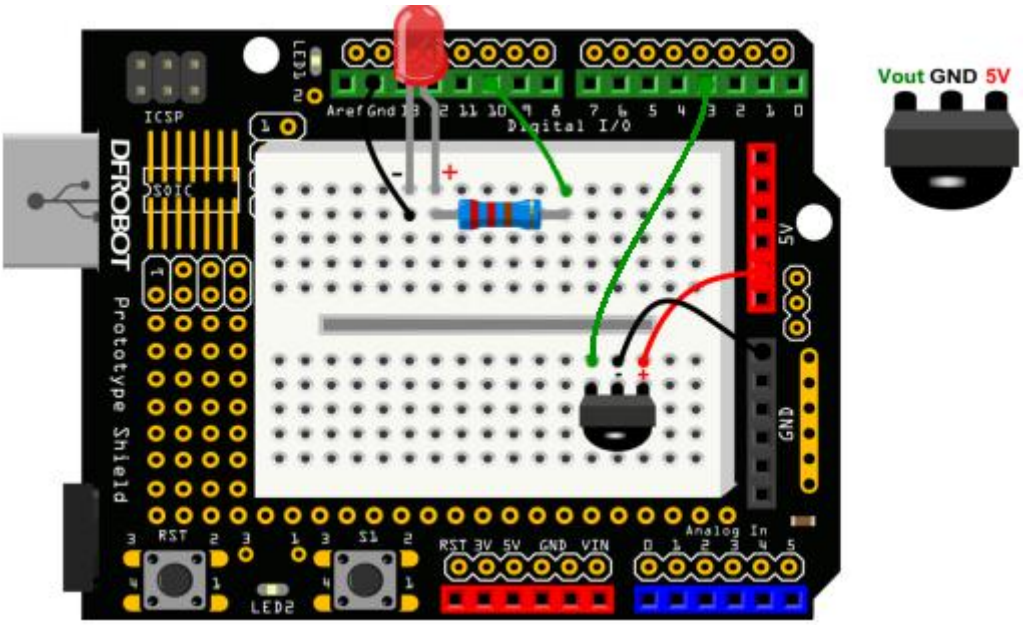


图 14-2 红外接收管连线图

代码编程

样例程序 14-2

```
#include <DFRobot_IRremote.h>

String result;                                  // 创建寄存变量
IRremote_Receive remoteReceive_3;               // 创建对象
int RECV_PIN = 3;
int ledPin = 10;                                // LED – digital 10
boolean ledState = LOW;                         // ledstate 用来存储 LED 的状态

// 主程序开始
void setup() {
    Serial.begin(9600);
    remoteReceive_3.begin(RECV_PIN);
    pinMode(ledPin,OUTPUT);                     // 设置 LED 为输出状态
}
void loop() {
    result = (remoteReceive_3.getIrCommand());
    if ((result!="0")) {
          Serial.println(result);
          //一旦接收到电源键的代码, LED 翻转状态，HIGH 变 LOW，或者 LOW 变 HIGH
          if(result == "FD00FF"){
          ledState = !ledState;                 //取反
          digitalWrite(ledPin,ledState);        //改变 LED 相应状态
          }
    }
}
```

代码学习

程序一开始还是对红外接收管的一些常规定义，按原样搬过来就可以了。

```
#include <DFRobot_IRremote.h>

String result;                                  // 创建寄存变量
IRremote_Receive remoteReceive_3;               // 创建对象
int RECV_PIN = 3;
int ledPin = 10;                                // LED – digital 10
boolean ledState = LOW;                         // ledstate 用来存储 LED 的状态
```

在这里，我们多定义了一个变量 ledState，通过名字应该就可以看出来含义了，用来存储 LED 的状态的，由于 LED 状态就两种（1 或者 0）,所以我们使用 boolean 变量类型，(可回看项目三中，表 3-1 列

举出的数据类型)。

setup()函数中，对使用串口，启动红外解码，数字引脚模式进行设置。

到了主函数 loop()，一开始还是先判断是否接收到红外码，并把接收到的数据存储在变量 result 中。

```
  if ((result!="0")) {
```

一旦接收到数据后，程序就要做两件事。第一件事，判断是否接收到了电源键的红外码。

```
if(result == "FD00FF"){
```

第二件事，就是让 LED 改变状态。

```
ledState = !ledState;                    //取反
digitalWrite(ledPin,ledState);           //改变 LED 相应状态
```

这里可能对“！”比较陌生，“！”是一个逻辑非的符号，“取反”的意思。我们知道“！=”代表的是不等于的意思，也就是相反。这里可以类推为，!ledState 是 ledState 相反的一个状态。“！”只能用于只有两种状态的变量中，也就是 boolean 型变量。

课后作业

1. 通过这个遥控项目，再结合上一个项目的风扇，能不能再给遥控器增加一个功能，既可控灯，还可控风扇。

2. DIY 一个你的遥控作品吧！比如简单的会动的小人，结合我们前面的舵机，通过遥控器上不同的按键，让舵机转动不同的角度，感觉随你的控制转动，发挥你的想象做出更多 Arduino 作品吧！

项目十五 红外遥控数码管

数码管，常见的用来显示数字的，比如像计算器。在本实验之前我们先来了解一下数码管是如何工作的。数码管，其实也算是 LED 中的一种。数码管的每一段，都是一个独立的 LED，通过数字引脚来控制相应段的亮灭就能达到显示数字的效果。下面让我们通过实验的方式来感受一下数码管的神奇之处吧！

所需元件

- 1× 八段数码管
- 8× 220 欧电阻

硬件连接

按下图连线图连接，注意数码管各段所对应的引脚。右边引脚说明图上为什么画这么几个箭头呢？个人觉得，这样看起来更方便。可以给你作为参考。我们从上面一排看，红色箭头的方向，从右往左，b→a→f→g 的顺序正好对应下面红色箭头逆时针顺序 b→a→f→g。蓝色箭头也是表达的同样的意思。

我还特意在连接图上，对数码管所连接的引脚做了标示。这样就能更清楚的知道哪个引脚控制哪一段了。这 8 个电阻同样是起限流的作用。

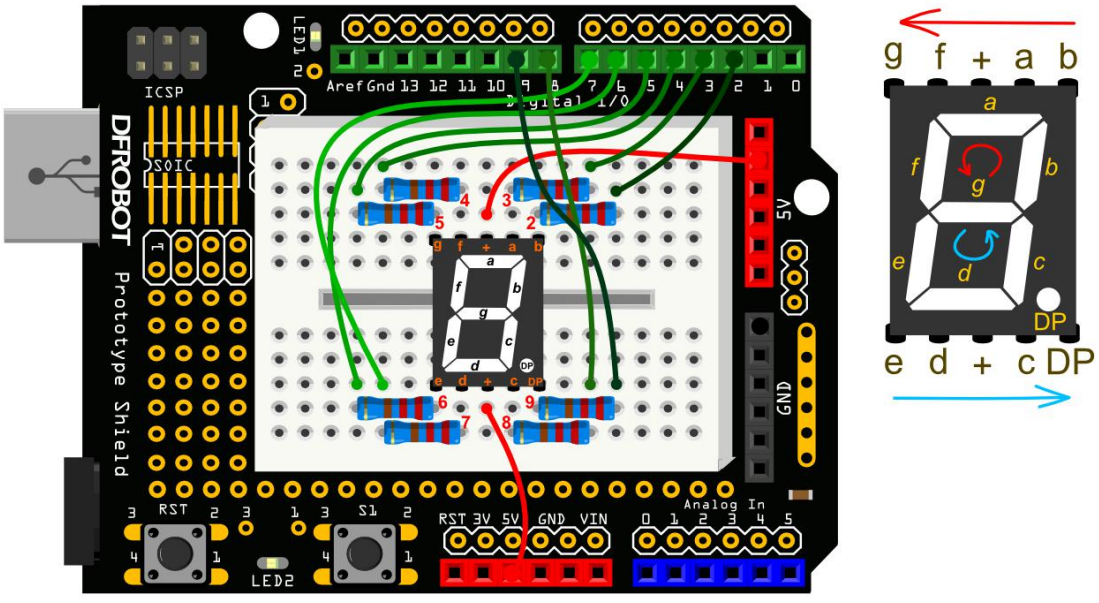


图 15-1 数码管显示连线图

代码编程

样例代码 15-1：

```
 //项目 15 – 数码管显示
void setup(){
    for(int pin = 2 ; pin <= 9 ; pin++){        // 设置数字引脚 2~9 为输出模式
          pinMode(pin, OUTPUT);
          digitalWrite(pin, HIGH);
    }
}

void loop() {
    // 显示数字 0
    int n0[8]={0,0,0,1,0,0,0,1};
    //数字引脚 2~9 依次按数组 n0[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n0[pin-2]);
    }
    delay(500);

    // 显示数字 1
    int n1[8]={0,1,1,1,1,1,0,1};
    // 数字引脚 2~9 依次按数组 n1[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n1[pin-2]);
    }
    delay(500);

    // 显示数字 2
    int n2[8]={0,0,1,0,0,0,1,1};
    // 数字引脚 2~9 依次按数组 n2[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n2[pin-2]);
    }
    delay(500);

    // 显示数字 3
    int n3[8]={0,0,1,0,1,0,0,1};
    // 数字引脚 2~9 依次按数组 n3[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n3[pin-2]);
    }
    delay(500);
```

```
    // 显示数字 4
    int n4[8]={0,1,0,0,1,1,0,1};
    // 数字引脚 2~9 依次按数组 n4[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n4[pin-2]);
    }
    delay(500);

    // 显示数字 5
    int n5[8]={1,0,0,0,1,0,0,1};
    // 数字引脚 2~9 依次按数组 n5[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n5[pin-2]);
    }
    delay(500);

    // 显示数字 6
    int n6[8]={1,0,0,0,0,0,0,1};
    // 数字引脚 2~9 依次按数组 n6[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n6[pin-2]);
    }
    delay(500);

    // 显示数字 7
    int n7[8]={0,0,1,1,1,1,0,1};
    // 数字引脚 2~9 依次按数组 n7[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n7[pin-2]);
    }
    delay(500);

    // 显示数字 8
    int n8[8]={0,0,0,0,0,0,0,1};
    // 数字引脚 2~9 依次按数组 n8[8]中的数据显示
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n8[pin-2]);
    }
    delay(500);

    // 显示数字 9
    int n9[8]={0,0,0,0,1,1,0,1};
    // 数字引脚 2~9 依次按数组 n9[8]中的数据显示
```

```
    for(int pin = 2; pin <= 9 ; pin++){
          digitalWrite(pin,n9[pin-2]);
    }
    delay(500);
}
```

完成下载后，数码管就会循环显示 0~9 的数字。由于要看懂代码的话，首先需要了解数码管的构造，所以我们这回先说硬件部分。

硬件回顾

数码管

数码管其实就是一个前面介绍的 LED 的组合体，这个组合体包含 8 个 led，所以也称之为八段数码管。说白了就八个灯。哪八段？不用多说了吧！a 到 g 以及小数点 DP。其实用法和前面说的 LED 也是一样的，每段都是一个发光二极管，分别用 8 个数字口来控制它们的亮灭，通过不同段的显示，就能组成 0~9 的数字。比如，我们让 b、a、f、e、d、c 亮起的话，就能显示一个数字“0”了。

下图 15-2 是引脚说明图，不陌生了吧！在前面硬件连接的时候，已经看到过一次了。

这里，b→a→f→g→e→d→c→DP 分别连接到 Arduino 数字引脚 2~9。

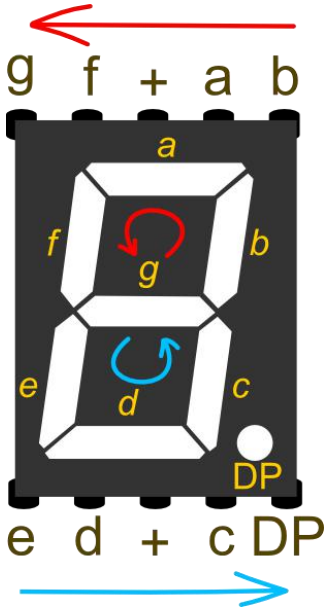


图 15-2 引脚说明图

数码管的共阴共阳在使用上有什么区别

共阳数码管，它们公共端接 5V，那在代码中，控制另一端的数字引脚为 LOW，这样才能让数码管点亮。如果是共阴数码管，公共端接 GND，在代码中中，控制另一端数字引脚为 HIGH，才让数码管点亮。

所以，共阴共阳只是在代码上要稍作修改。我们这里选用的是共阳数码管。了解了硬件，我们来看看软件部分。

代码学习

硬件部分我们已经说过，数码管需要接到 8 个数字引脚，所以在一开始，需要定义 8 个数字引脚作为输出。这次我们用一个 for 循环来完成这 8 个数字引脚的设置。数码管 b、a、f、g、e、d、c、DP 分别和 Arduino 数字引脚 2~9 对应。

```
 for(int pin = 2 ; pin <= 9 ; pin++){        // 设置数字引脚 2~9 为输出模式
        pinMode(pin, OUTPUT);
        digitalWrite(pin, HIGH);
 }
```

从引脚 2 开始，一直循环到引脚 9，都设为 OUTPUT 模式，初始化为 HIGH。前面说过，共阳的话，设置 HIGH，不被点亮，所以开始先不点亮数码管。（当然，你一个一个引脚分开设置输出模式也是不会错的，只是会让代码显得很冗长。）

好了，到了主函数，要分别显示 0~9 的数字。是不是觉得代码大部分都是相似的。所以，我们只要看明白如何显示数字 0，那整段代码就都迎刃而解了。

数组

```
int n0[8]={0,0,0,1,0,0,0,1};
```

这里我们要引入一个数组的概念。数组是一个变量的集合，可以通过索引号来找到数组中的元素。在我们的程序中，声明了一个 int 型的数组。并取名为 n0。之后用 8 个数值来初始化这个数组。那如何获得数组中的元素呢？你只需要简单的指出这个元素的索引号。数组是从 0 开始索引的，这意味着数组中的第一个元素的索引号为 0 而不是 1，因此数组中的 8 个元素的索引号是 0~7。在这里元素 4，对应索引号为 3（n0[3]），值为 1。元素 8（索引号 7，n0[7]）的值为 1。

声明中 n0[8]的方括号中的 8 代表有 8 个元素。

定义完数组后，进入又一个 for 循环。

```
for(int pin = 2; pin <= 9 ; pin++){
     digitalWrite(pin,n0[pin-2]);
}
```

这个 for 循环是给 2~9 引脚写入状态值，也就是 HIGH 还是 LOW，digitalWrite 函数中写入 HIGH 的另一种形式就是写入“1”，LOW 则可以写为“0”。我们通过数组索引的方式给 2~9 引脚赋值。

比如当 pin=2，代入 n0[pin-2]中，对应为 n0[0]，n0[0]意思是获得数组的第一个元素，为 0。完成了引脚 2 置低（LOW）。我们前面说了，共阳的数码管，置低（LOW）的话，是被点亮，所以，b 端被点亮了。循环到 pin=3，a 段被点亮,。循环到 pin=4，f 段被点亮，依次类推……

整个循环过程如下：

pin=2 → n0[0] =0 → digitalWrite(2,0) → b 段点亮

pin=3 → n0[1] =0 → digitalWrite(3,0) → a 段点亮

pin=4 → n0[2] =0 → digitalWrite(4,0) → f 段点亮

pin=5 → n0[3] =1 → digitalWrite(5,1) → g 段不点亮

pin=6 → n0[4] =0 → digitalWrite(6,0) → e 段点亮

pin=7 → n0[5] =0 → digitalWrite(7,0) → d 段点亮

pin=8 → n0[6] =0 → digitalWrite(8,0) → c 段点亮

pin=9 → n0[7] =1 → digitalWrite(9,1) → DP 段不点亮

这样就完成了显示数字“0”了。同样用数组的方法显示数字 1~9。自己动手画一下，哪几段亮，哪几段不亮就一目了然了。

如果代码 1 弄明白后，我们要教大家一种更简单的方法。

代码编程 2

样例程序 15-2

```
//项目 15 – 数码管数字显示
int number[10][8] =
{
  {0,0,0,1,0,0,0,1},    //显示 0
  {0,1,1,1,1,1,0,1},    //显示 1
  {0,0,1,0,0,0,1,1},    //显示 2
  {0,0,1,0,1,0,0,1},    //显示 3
  {0,1,0,0,1,1,0,1},    //显示 4
  {1,0,0,0,1,0,0,1},    //显示 5
  {1,0,0,0,0,0,0,1},    //显示 6
  {0,0,1,1,1,1,0,1},    //显示 7
  {0,0,0,0,0,0,0,1},    //显示 8
  {0,0,0,0,1,1,0,1}     //显示 9
};
void numberShow(int i){          //该该函数用来显示数字
    for(int pin = 2; pin <= 9 ; pin++){
       digitalWrite(pin, number[i][pin - 2]);
    }
}
```

```
void setup(){
  for(int pin = 2 ; pin <= 9 ; pin++){        // 设置数字引脚 2~9 为输出模式
      pinMode(pin, OUTPUT);
      digitalWrite(pin, HIGH);
  }
}

void loop() {
    for(int j = 0; j <= 9 ; j++){
        numberShow(j);        //调用 numberShow()函数，显示 0~9
        delay(500);
    }
}
```

代码学习

对比一下代码 1，能发现明显的区别在哪里了吗？代码 1 中，我们创建了 10 个一维数组，代码 2 只需要创建一个二维数组就全部搞定了。不要被什么一维、二维数组的名字给吓唬到，其实用法一样的。

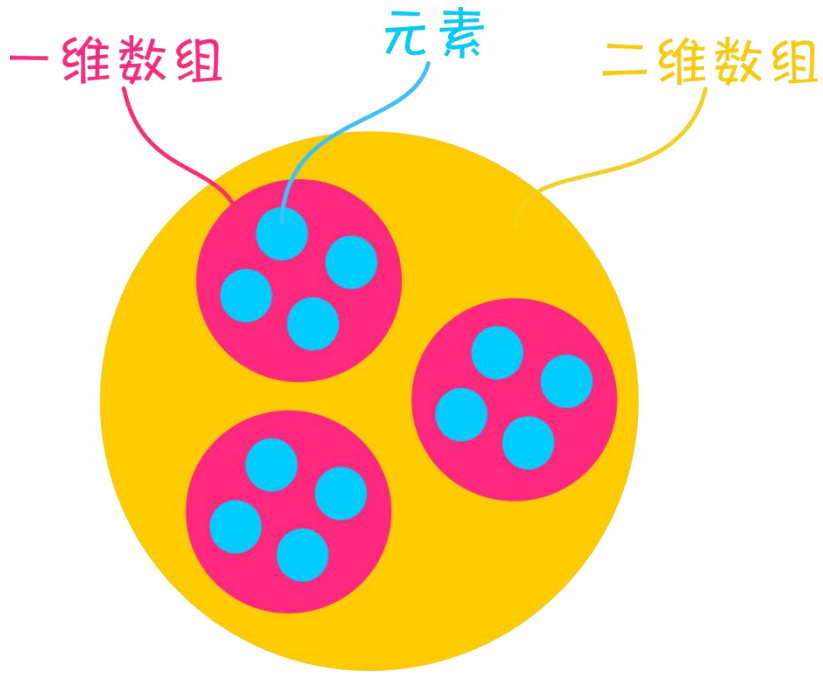


图 15-3 数组关系图

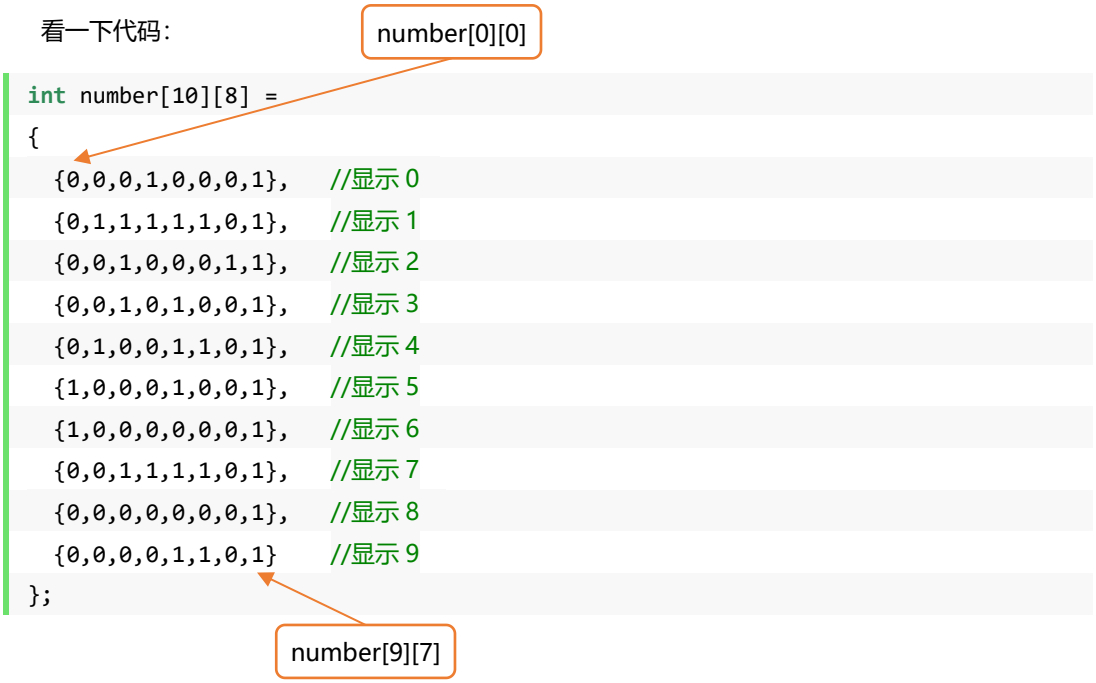
通过上面这个图，元素、一维数组、二维数组之间的关系就一目了然了。一维数组由元素组成，而二维数组则是由一个个一维数组组成的，关系就是那么简单。

代码 1 中，分别用 10 个数组，每个数组有 8 个元素，每个元素依次对应到数码管 b~DP 引脚状态值，这样就能在数码管上反映为 0~9 的数字显示。

而我们现在则是把前面散开的 10 个一维数组整合到一起，变为一个二维数组。同样通过索引的方式

来找到这些元素。但，还是不要忘了索引号也是从 0 开始的！前面的方括号写入的是只是元素个数。

看一下代码：




```
int number[10][8] =
{
  {0,0,0,1,0,0,0,1},    //显示 0
  {0,1,1,1,1,1,0,1},    //显示 1
  {0,0,1,0,0,0,1,1},    //显示 2
  {0,0,1,0,1,0,0,1},    //显示 3
  {0,1,0,0,1,1,0,1},    //显示 4
  {1,0,0,0,1,0,0,1},    //显示 5
  {1,0,0,0,0,0,0,1},    //显示 6
  {0,0,1,1,1,1,0,1},    //显示 7
  {0,0,0,0,0,0,0,1},    //显示 8
  {0,0,0,0,1,1,0,1}     //显示 9
};
```

这就是一个二维数组。索引号从 0 开始，如果让你找 number[0][0]，能找到是哪个数吗？就是二维数组中第 1 行的第 1 个数，为 0。number[9][7]也就是第 10 行的第 8 个数，为 1。

```
void numberShow(int i){          //该该函数用来显示数字
    for(int pin = 2; pin <= 9 ; pin++){
       digitalWrite(pin, number[i][pin - 2]);
    }
}


void loop() {
    for(int j = 0; j <= 9 ; j++){
        numberShow(j);        //调用 numberShow()函数，显示 0~9
        delay(500);
    }
}
```

上面这两段代码我们整合在一起看，loop()主函数中，for 循环让变量 j 在 0~9 循环，j 每赋一次值，numberShow()函数就要运行一次。

numberShow()函数整个运行过程如下：

程序一开始 j=0，numberShow(j)为 numberShow(0),跳回到上面的 numberShow()函数，i 现在的

值就为 0 了，pin 初始值为 2，所以 digitalWrite()现在值为 digitalWrite(2,number[0][0])，回到数组 number[10][8]中找到 number[0][0]对应的值，为 0。此时，digitalWrite(2,0)，代表引脚 2 被置 LOW，引脚 2 对应的 b 段点亮（共阳数码管置 LOW 才被点亮）。之后再是循环 pin=3，pin=4，……，一直到 pin=9 整个 for 循环才结束，也代表数组的第一行的 8 个元素全被运行了一遍，最终显示一个数字“0”。

回顾一下代码 1 是如何显示一个数字“0”的：

```
// 显示数字 0
int n0[8]={0,0,0,1,0,0,0,1};
//数字引脚 2~9 依次按数组 n0[8]中的数据显示
for(int pin = 2; pin <= 9 ; pin++){
   digitalWrite(pin,n0[pin-2]);
    }
```

原理是一样的，通过给引脚 2~9 循环赋值，控制数码管 b~DP 段亮灭，就能显示出一个我们想要的数字。

numberShow(0)循环完后，再次回到 loop()中的 for 函数：

j=1 numberShow(1) i=1 number[1][pin-2] 显示数字 1
j=2 numberShow(2) i=2 number[2][pin-2] 显示数字 2
j=3 numberShow(3) i=3 number[3][pin-2] 显示数字 3
……
j=9 numberShow(9) i=9 number[9][pin-2] 显示数字 9

好了，这就是整段代码的分析过程，好好体会一下一维数组和二维数组的区别，以及整段代码如何巧妙运行的。

了解了数码管和红外接收管各自的工作原理后，我们需要把这两者结合起来，看看红外接收管和数码管结合能迸发出怎样的火花呢？想到了吗？遥控数码管！Arduino 控制器把红外接收管从 Mini 遥控器那儿接到的信号，经处理传达给数码管。让 Mini 遥控器上 0~9 对应在数码管上显示 0~9，除此之外，还有递减，递增的功能。

所需元件

- 1× 八段数码管
- 1× 红外接收管
- 1× Mini 遥控器
- 8× 220 欧电阻

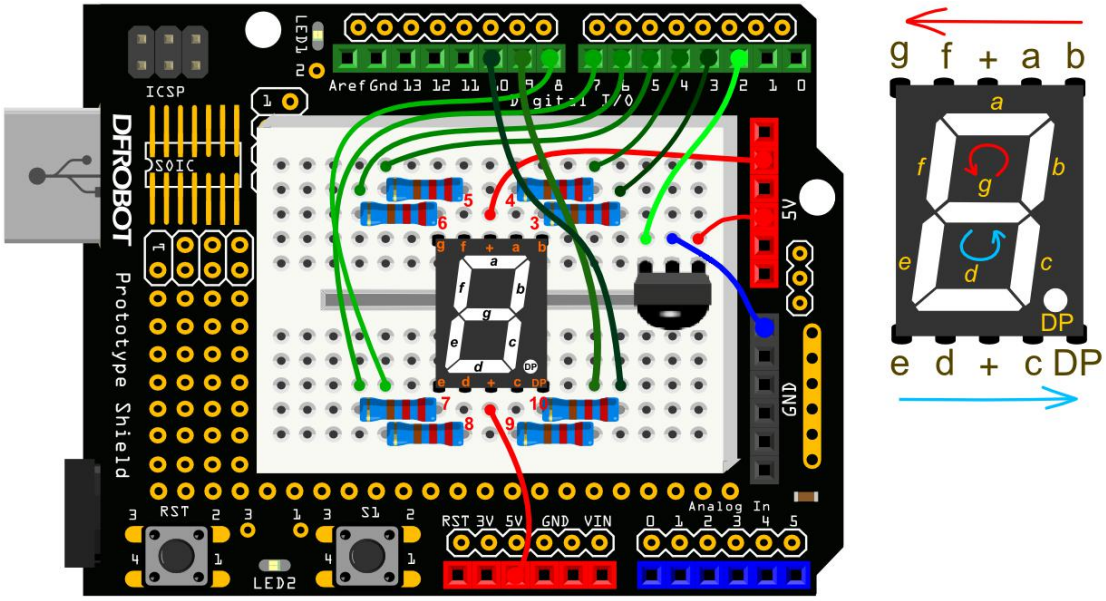


图 15-4 红外遥控数码管连线图

代码编程

在输入代码的过程中，结合前面的项目十四和十五，看看整段代码是如何把这两者整合的，程序中是

如何处理把红外接收管接到的信号，再转变为数码管的显示的。

样例程序 15-3

```
//项目十五 - 红外遥控数码管
#include <DFRobot_IRremote.h>       //调用 IRremote.h 库
int RECV_PIN = 2;                   //定义 RECV_PIN 变量为 2
IRremote_Receive remoteReceive_2;   //声明对象

String results;                     //定义 results 变量为红外结果存放位置
int currentNumber = 0;              //该变量用于存放当前数字

String codes[12]=                   //该数组用来存放红外遥控器发出的红外码
{
   "FD30CF","FD08F7",                // 0 ,1
   "FD8877","FD48B7",                // 2 ,3
   "FD28D7","FDA857",                // 4 ,5
   "FD6897","FD18E7",                // 6 ,7
   "FD9867","FD58A7",                // 8 ,9
   "FD20DF","FD609F",                // + ,-
};

int number[10][8] =             //该数组用来存放数码管显示的数字
{
  {0,0,0,1,0,0,0,1},//0
  {0,1,1,1,1,1,0,1},//1
  {0,0,1,0,0,0,1,1},//2
  {0,0,1,0,1,0,0,1},//3
  {0,1,0,0,1,1,0,1},//4
  {1,0,0,0,1,0,0,1},//5
  {1,0,0,0,0,0,0,1},//6
  {0,0,1,1,1,1,0,1},//7
  {0,0,0,0,0,0,0,1},//8
  {0,0,0,0,1,1,0,1} //9
};

void numberShow(int i) {                    //该函数用来让数码管显示数字
  for(int pin = 3; pin <= 10 ; pin++){
     digitalWrite(pin, number[i][pin - 3]);
  }
}

void setup(){
    Serial.begin(9600);                     //设置波特率为 9600
    remoteReceive_2.begin(RECV_PIN);        //启动红外解码
```

```
    for(int pin = 3 ; pin <= 10 ; pin++){    //设置数字引脚 2~9 为输出模式
          pinMode(pin, OUTPUT);
          digitalWrite(pin, HIGH);
    }
}

void loop() {
    //判断是否接收到解码数据,把接收到的数据存储在变量 results 中
    results = (remoteReceive_2.getIrCommand());
    if (results !="0") {
      for(int i = 0; i <= 11; i++){
         //判断是否接收到 0~9 按键的红外码
         if(results == codes[i]&& i <= 9){
               numberShow(i);       //在数码管上对应显示 0~9
               currentNumber = i;  //把当前显示的值赋给变量 currentNumber
               Serial.println(i);
               break;
         }

         // 判断是否接收到递减的红外码，并且当前值不为 0
         else if(results == codes[10]&& currentNumber != 0){
               currentNumber--;                //当前值递减
               numberShow(currentNumber);      //数码管显示递减后的值
               Serial.println(currentNumber); //串口输出递减后的值
               break;
         }

         //判断是否接收到递增的红外码，并且当前值不为 9
         else if(results == codes[11]&& currentNumber != 9){
               currentNumber++;                //当前值递增
               numberShow(currentNumber);      //数码管显示递增后的值
               Serial.println(currentNumber); //串口输出递增后的值
               break;
         }
      }

   Serial.println(results);  //串口监视器查看红外码

  }
}
```

图 15-5 遥控数按键说明

下载完代码后，尝试按下上图 15-5 指出部分的按钮，看看是数码管是个怎样的变化。

代码学习

```
int currentNumber = 0;      //该变量用于存放当前数字
```

在这里，我们多定义了一个变量 currentNumber，通过名字应该就可以看出来含义了吧。这个变量的作用是，用来存储当前的数字，数字递增递减是能找到对应的参照点。

同样用数组的方式来存放这些红外码。long 是变量的类型，如果你还想用遥控器上的其他按钮来控制做一些其他事情的话，把红外码替换掉就好了。

```
String codes[12]=              //该数组用来存放红外遥控器发出的红外码
{
   ”FD30CF”,”FD08F7”,              // 0 ,1
   ”FD8877”,”FD48B7”,              // 2 ,3
   ”FD28D7”,”FDA857”,              // 4 ,5
   ”FD6897”,”FD18E7”,              // 6 ,7
   ”FD9867”,”FD58A7”,              // 8 ,9
   ”FD20DF”,”FD609F”,              // + ,-
};
```

紧接着是，一个二维数组 number[10][8]的定义。我们在数码管那一章节已有说明了，通过调用数组的元素，把这些元素的值依次赋给数码管显示段的控制引脚，并在 numberShow()函数中得以实现数码管数字显示。

setup()函数中，仍然是波特率设置，启动红外解码，数字引脚模式设置等，这些常规设置。

到了主函数 loop()，一开始还是先判断是否接收到红外码，并把接收到的数据存储在变量 results 中。

```
if (results !="0") {
```

一旦接收到数据后，程序就要做两件事。第一件事，判断是哪个红外码，也就能对应找到是哪个按键按下的。第二件事，找到对应按扭后，让数码管干什么事？让我们接着看看程序是如何完成这两件事的。

第一件事：

会有三种情况需要判断，第一种情况，按遥控器 0~9 时，数码管显示数字 0~9。第二种情况，每按下“后退”键，数字在原有基础上递减一位，直到减到 0 为止。第三种情况，每按下“前进”键，数字在原有基础上增一位，直到增到 9 为止。

对这三种情况进行判断，这里呢，同样用到了 if 语句，与以往有所不同的，我们选择用 if...else if。if...else 和 if...else if 的区别在哪儿？区别在于 else if 后面需要接判断表达式，else 不需要判断表达式。然而，不管是 else 还是 else if 都是依附于 if 语句存在的，不能独立使用。

回到代码中，这就是以下三种情况：

```
if(results == codes[i]&& i <= 9){

if(results == codes[10]&& currentNumber != 0){

if(results == codes[11]&& currentNumber != 9){
```

第一个 if 判断的是第一种情况，显示数字 0~9。判断条件就是接收到的数据 results.value 的值是不是数组中 codes[0]~codes[9]的红外码。

第二个 if 判断的是第二种情况，是否接到“后退”键指令，也就是 code[10]= 0xFD20DF，并且当前显示数字不为 0。

第三个 if 判断的是第三种情况，是否接到“前进”键指令，也就是 code[11]= 0xFDA857，并且当前显示数字不为 9。

还有一个问题——如何找到数组中的元素呢？所以，就需要在 if 判断前设置一个 for 循环，让变量 i 一直在 0~11 之间循环。

第一件事判断是哪个红外码完成后，开始执行第二件事。就是每个 if 语句后，都有相应的执行代码，其中有一处需要进行讲解。

break 语句

break 语句用于跳出循环。在判断是否为数字 0-9、“后退”键、或“前进”键，且执行完相应的显示后，使用 break 语句，能够跳出当前循环，不再向下执行代码。这样做是为了实现，在一次 for 循环中只执行一次数码管显示的刷新，排除干扰造成的误显示，并且能够提升程序执行的效率。

整段代码就讲完了，这段代码应该是所以项目中最复杂的代码，可以一开始不能完全看明白，不过没关系，实践出真知，通过一遍遍不断的尝试，相信你总有一天能明白的。

课后作业

通过这个遥控项目，DIY 一个你的遥控作品吧！比如简单的会动的小人，结合我们前面的舵机，通过遥控器上不同的按键，让舵机转动不同的角度，感觉随你的控制转动，发挥你的想象做出更多 Arduino 作品吧！