项目零

初识Arduino

00

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。

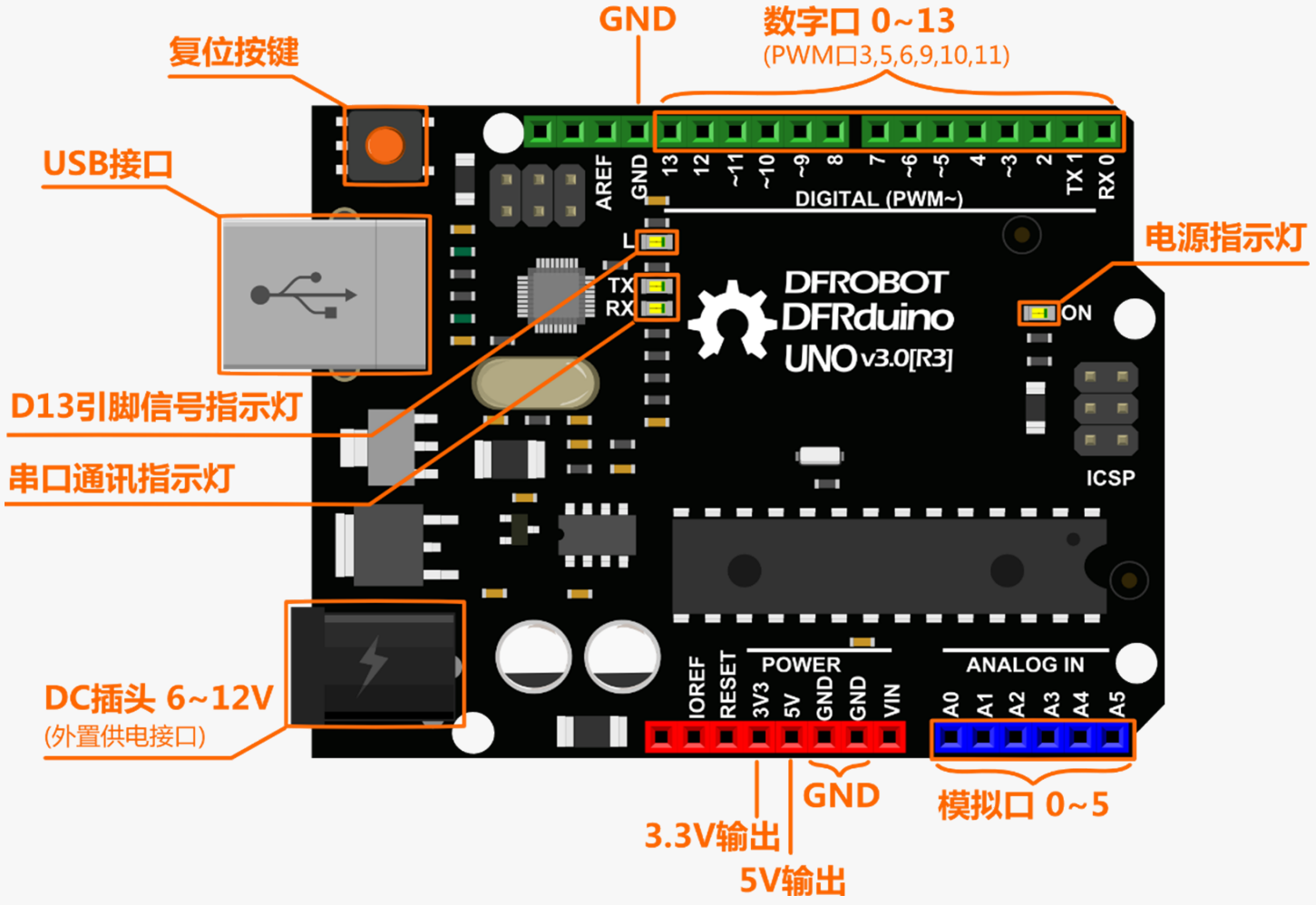
Arduino名称由来

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

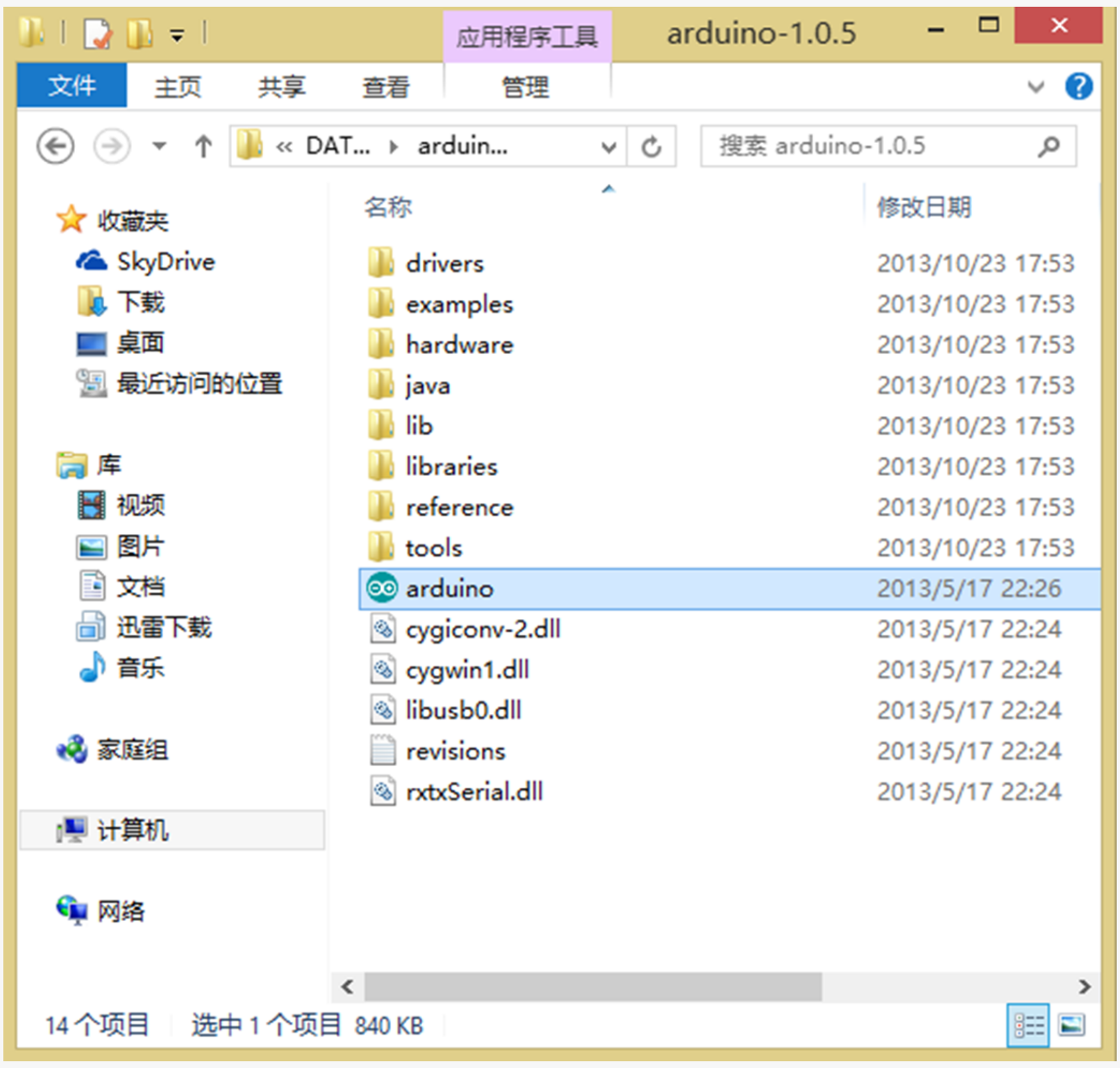
1.下载Arduino IDE

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。

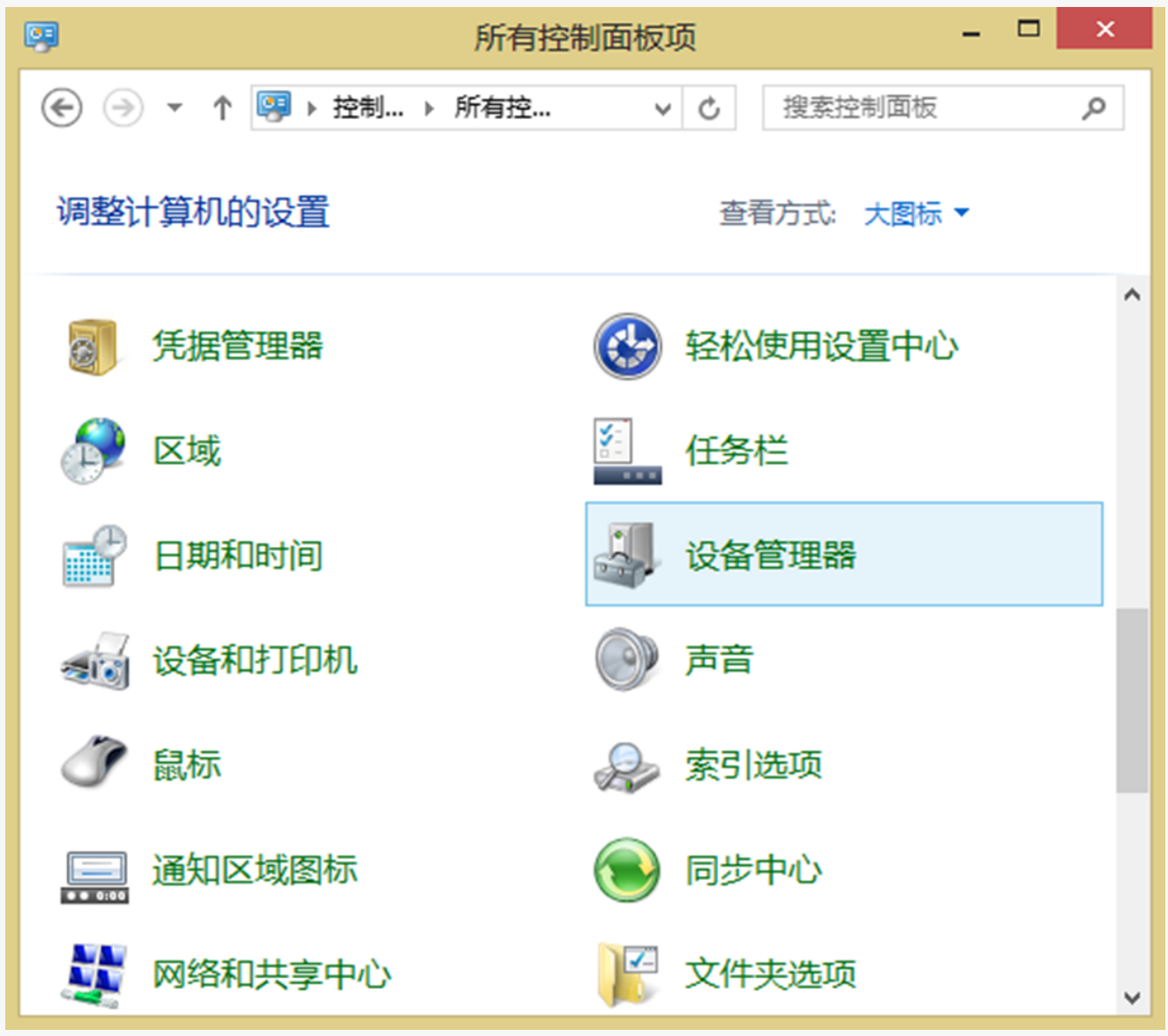
Windows用户，点击下载Windows(ZIP file)，如果Mac，Linux用户则选择相应的系统。

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。

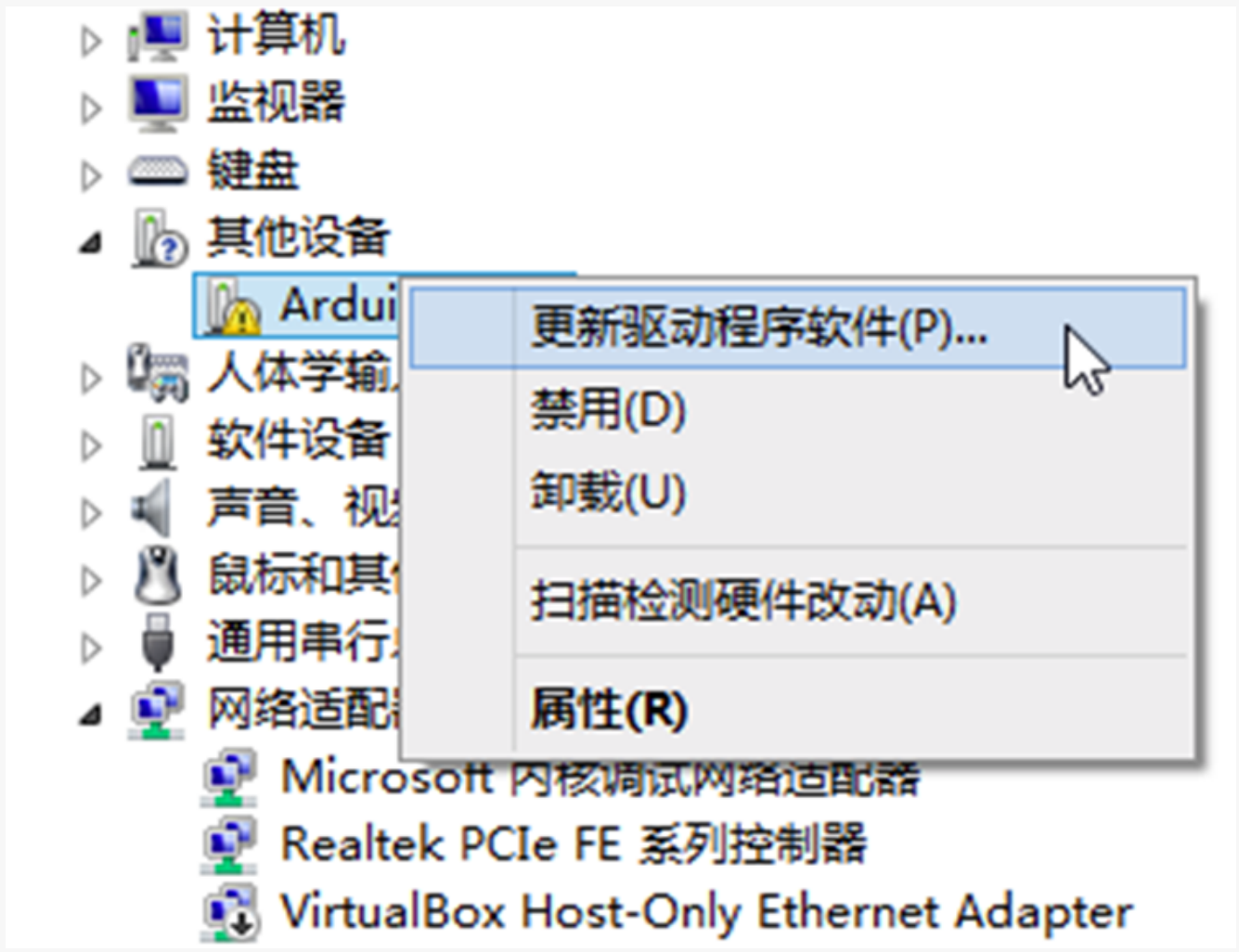


2. 安装驱动

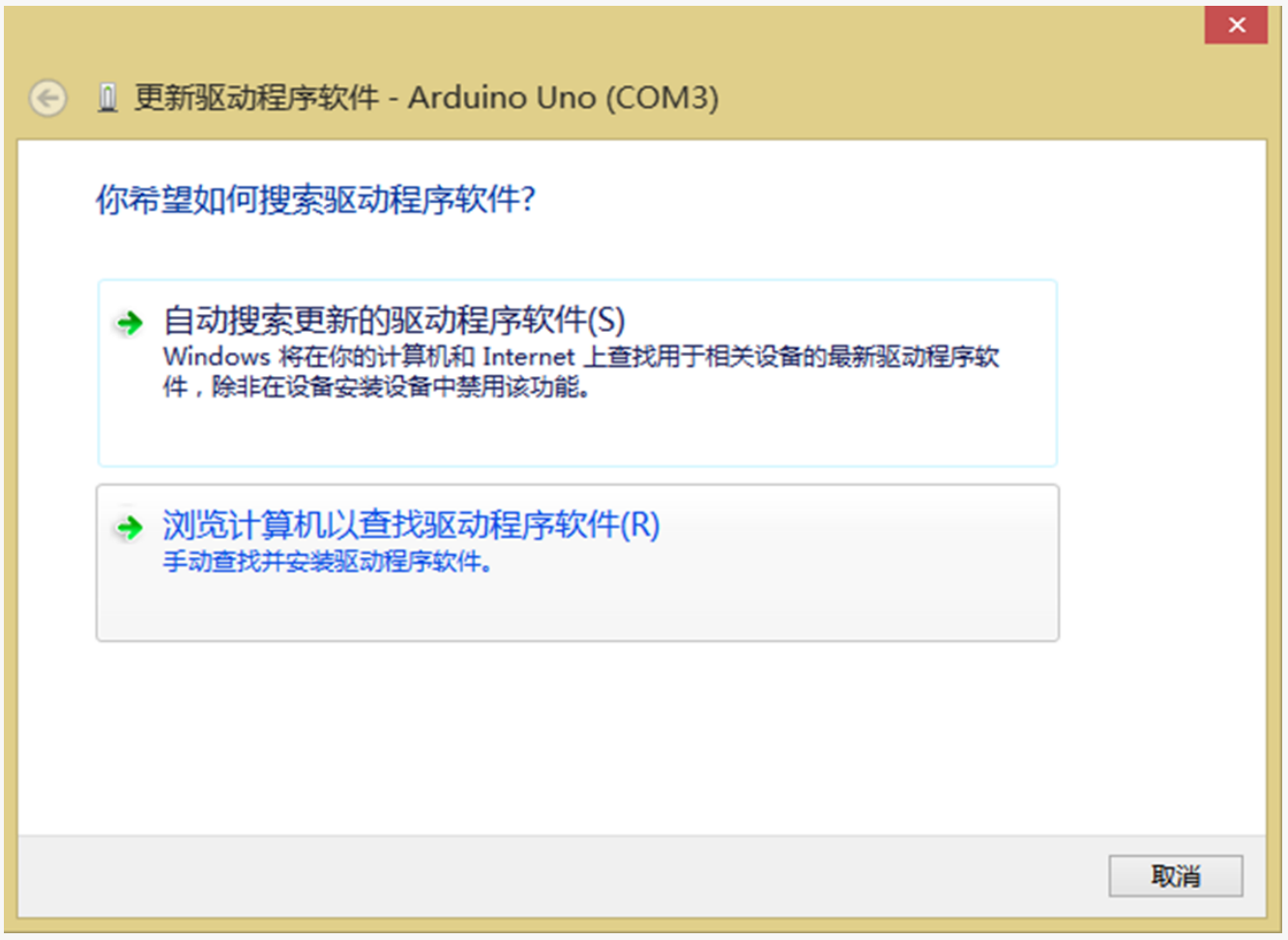
把USB一端插到Arduino UNO上，另一端连到电脑。连接成功后，UNO板的红色电源指示灯ON亮起。然后，打开控制面板，选择设备管理器。



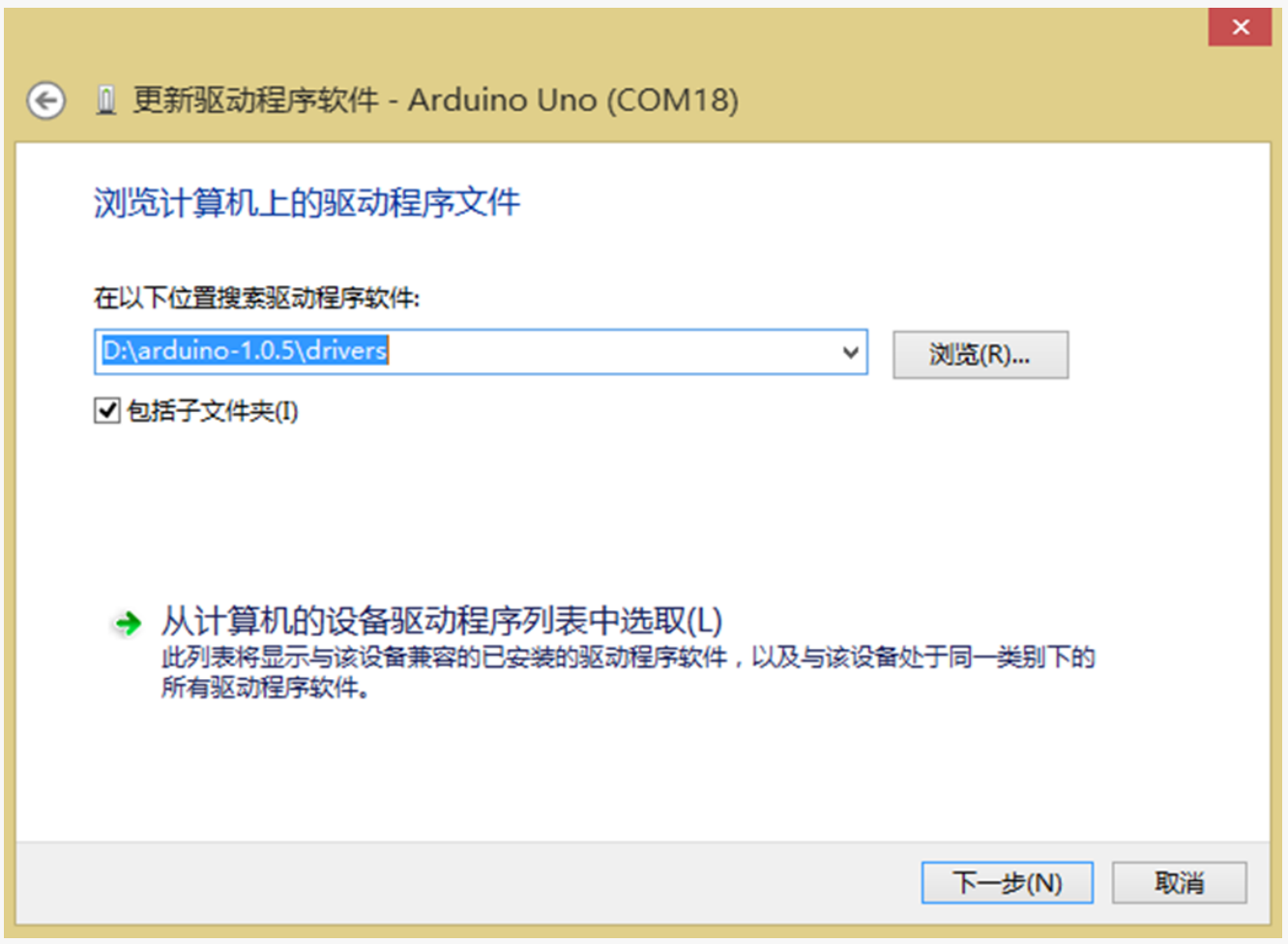
找到其它设备>Arduino-xx，右击选择更新驱动程序软件。



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。

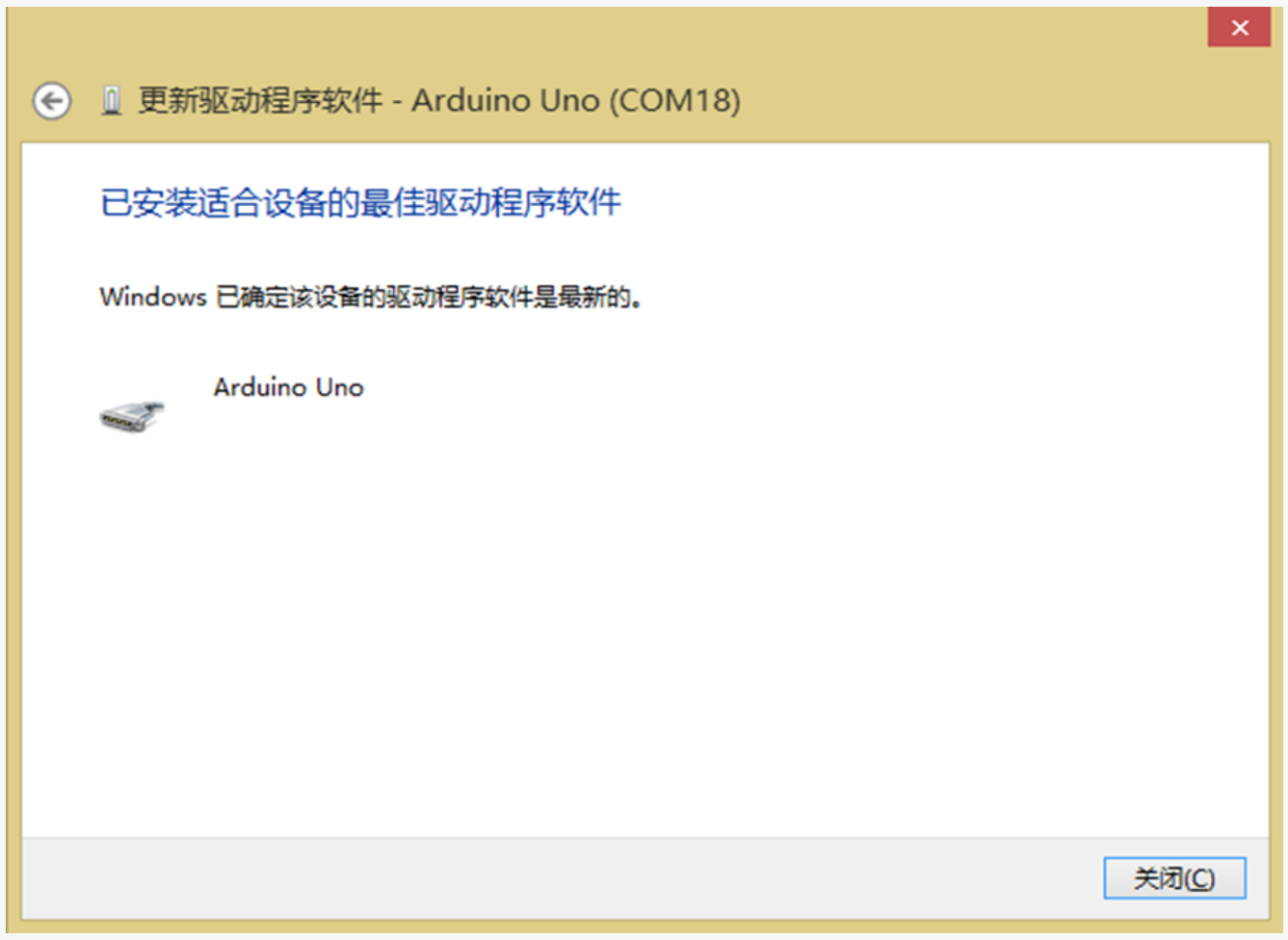


打开到Arduino IDE安装位置，就是上面那个解压文件的位置，选择搜索路径到drivers，点击下一步。

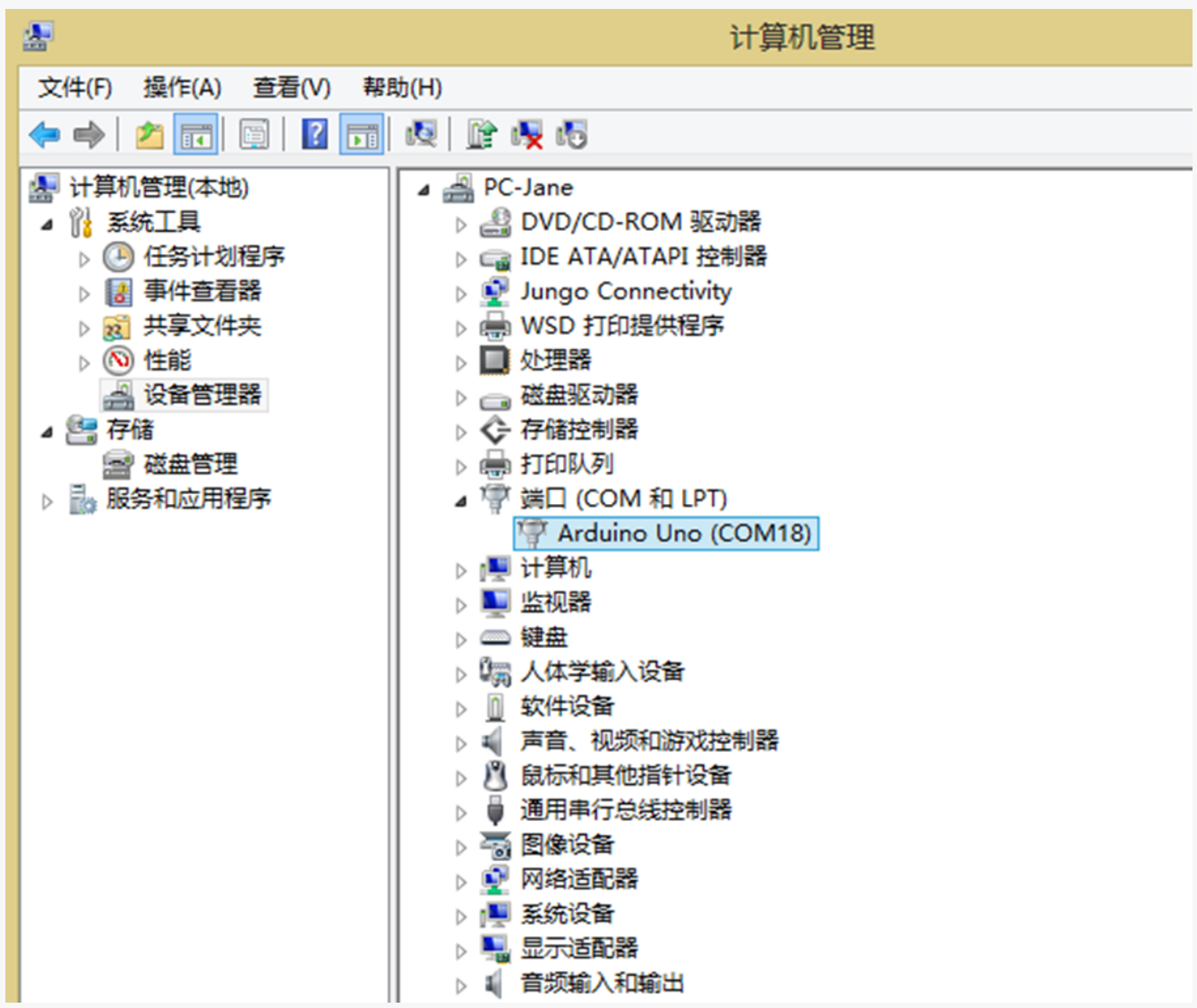


选择始终安装此驱动程序软件，直至完成。

出现下图，说明驱动安装成功。

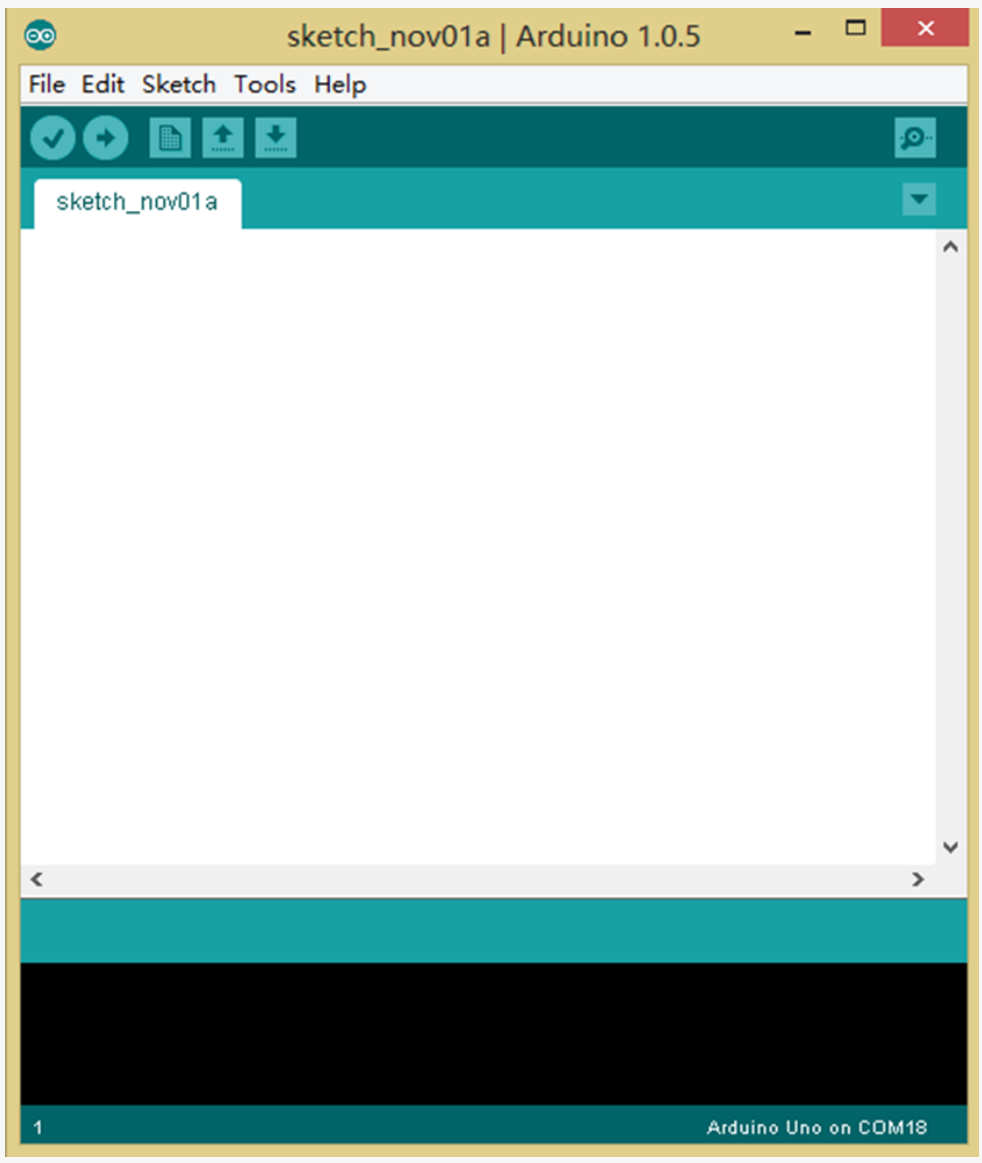


此时，设备管理器端口会显示一个串口号。

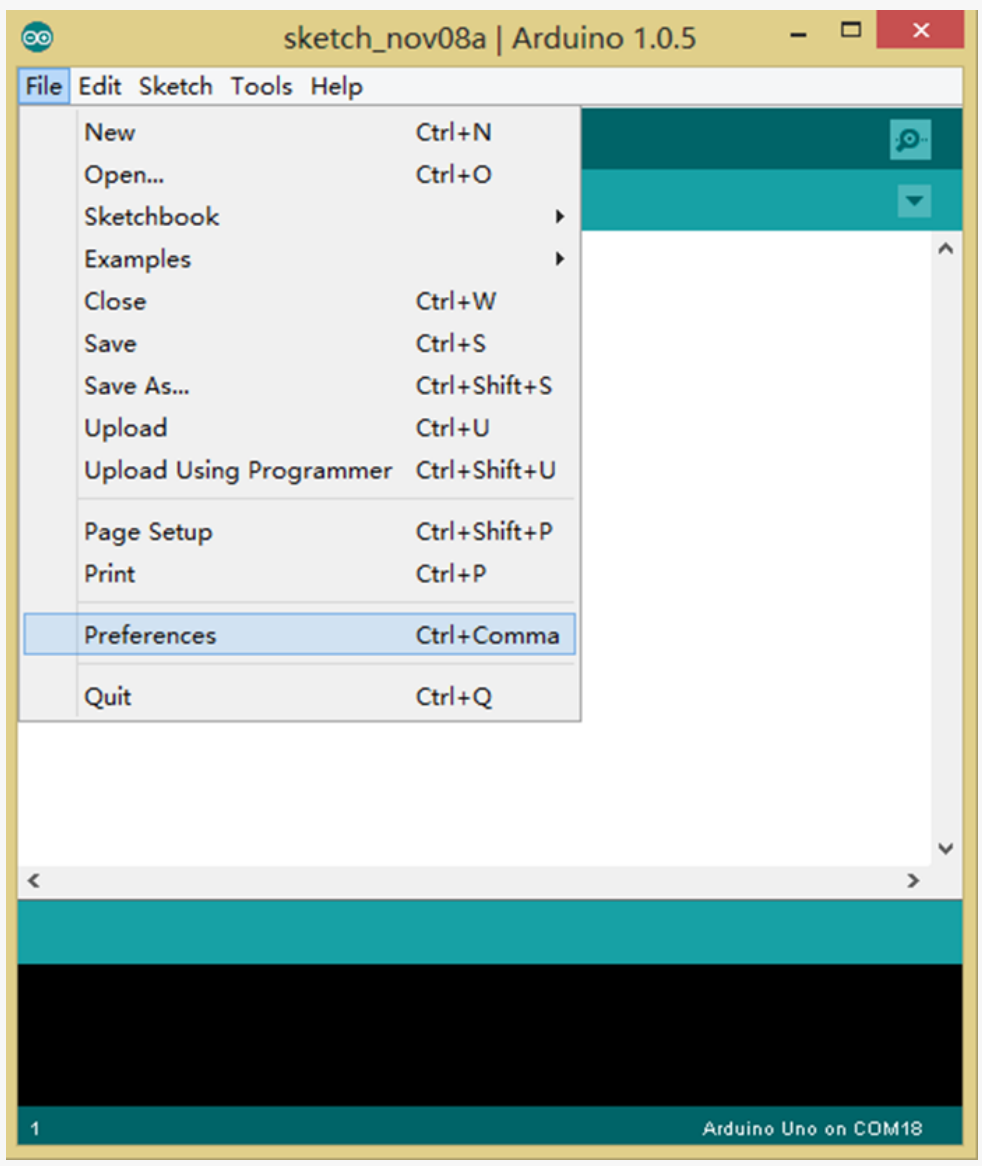


3. 认识Arduino IDE

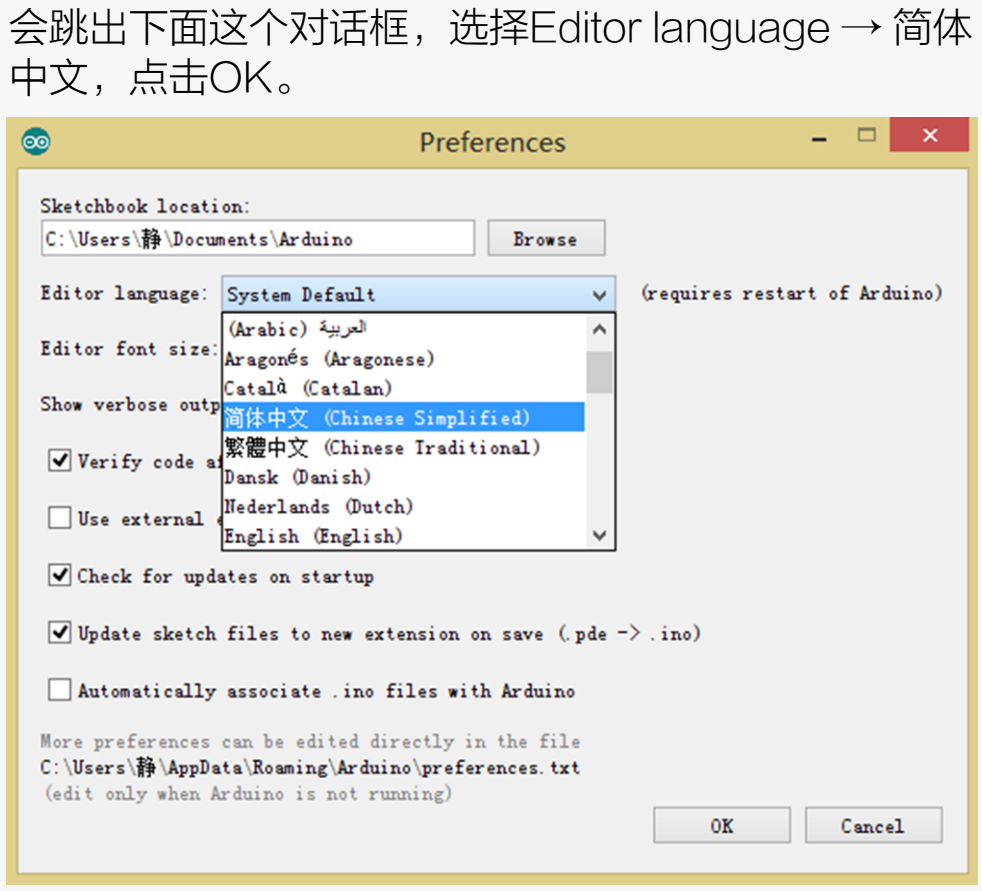
打开Arudino IDE，就会出现Arduino IDE的编辑界面。



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。

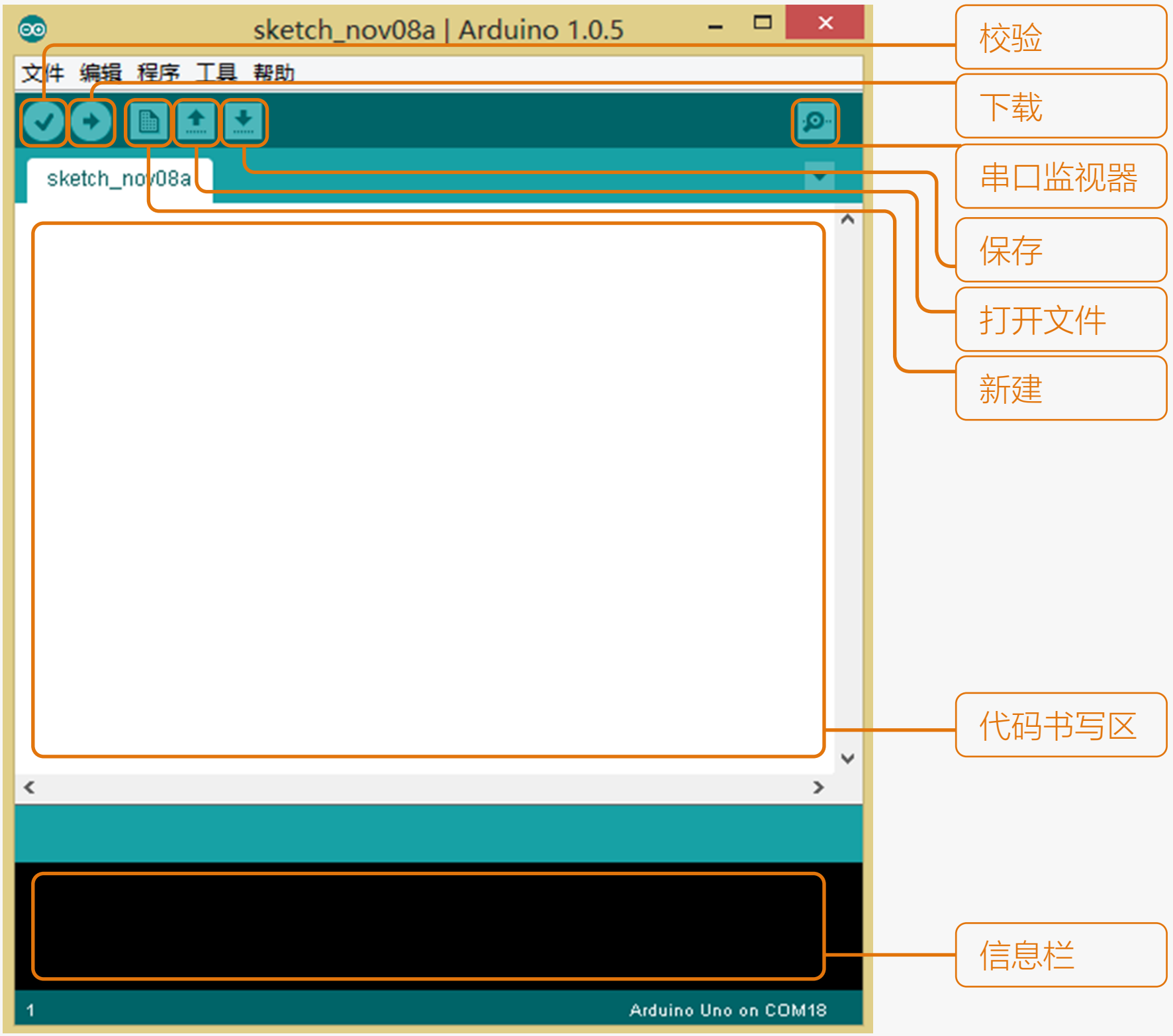


会跳出下面这个对话框，选择Editor language → 简体中文，点击OK。



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。

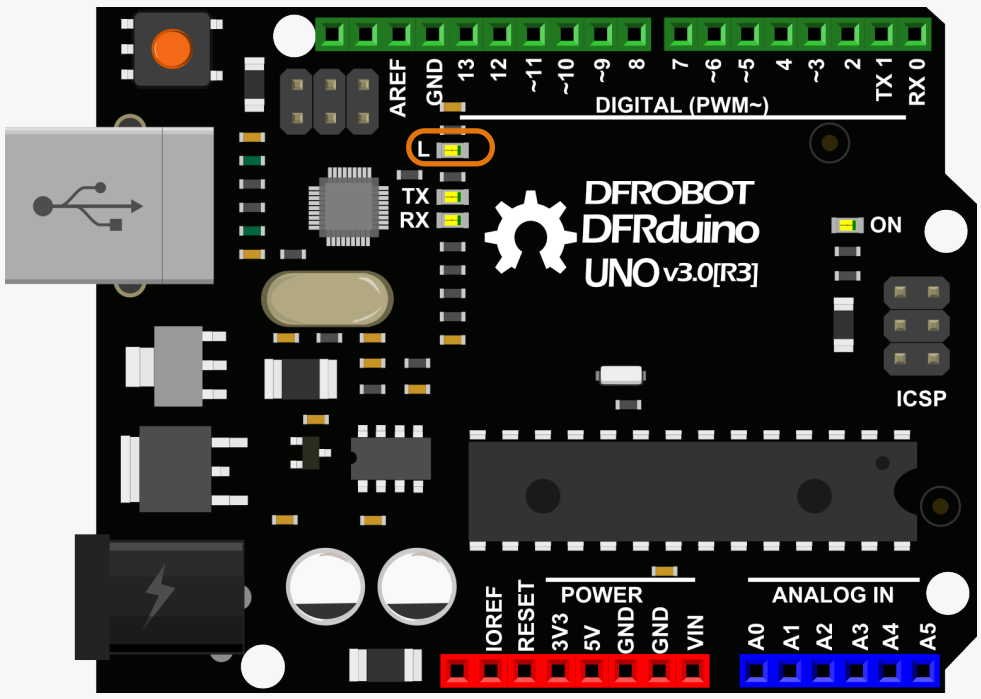


Arduino IDE是Arduino产品的软件编辑环境。简单的说就是用来写代码，下载代码的地方。任何的Arduino产品都需要下载代码后才能运作。我们所搭建的硬件电路是辅助代码来完成的，两者是缺一不可的。如同人通过大脑来控制肢体活动是一个道理。如果代码就是大脑的话，外围硬件就是肢体，肢体的活动取决于大脑，所以硬件实现取决于代码。

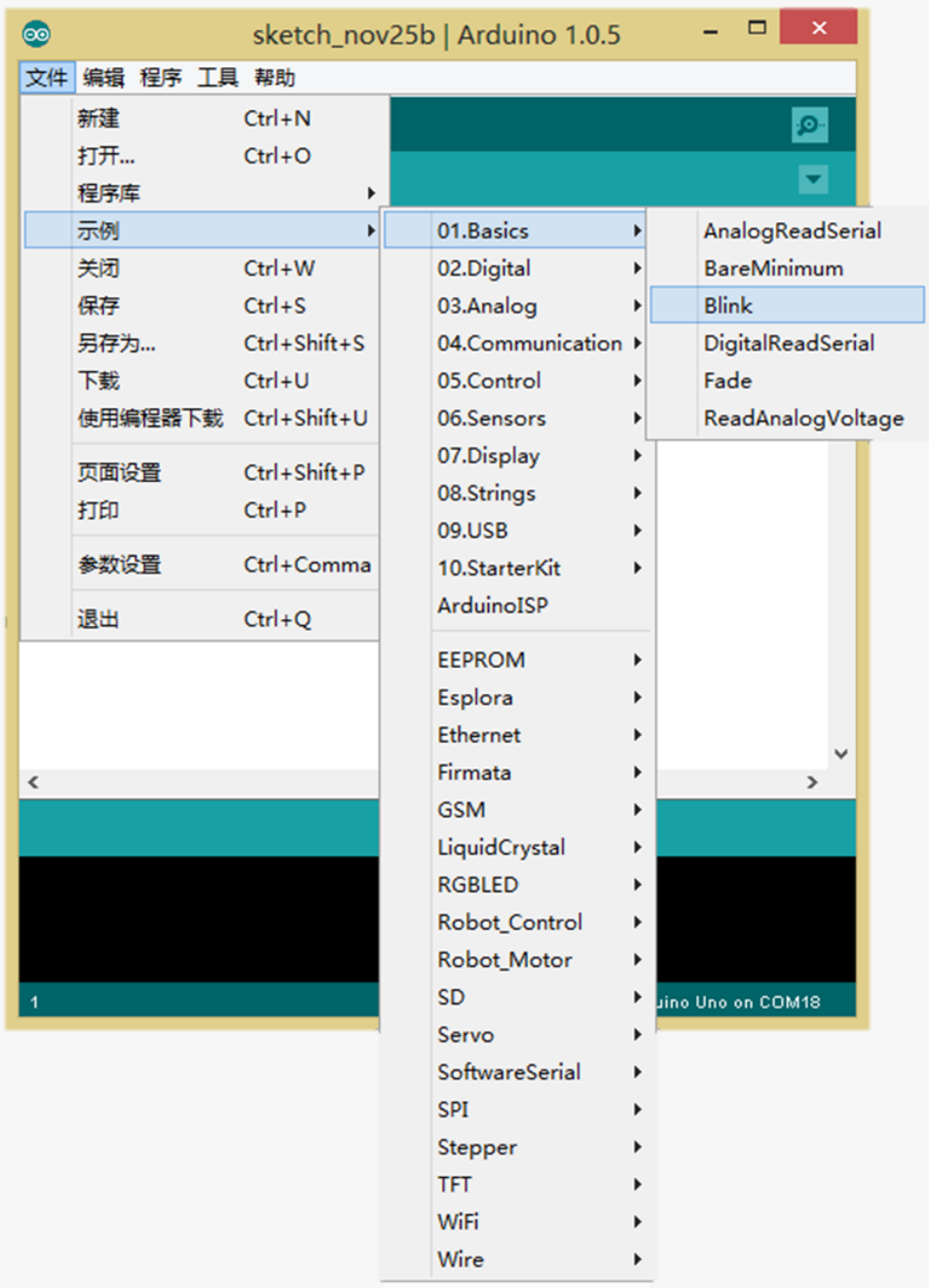
Arduino IDE基本也只需要用到上面标示出来的部分就可以了，上图大部分的白色区域就是代码的编辑区，用来输入代码的。注意，输入代码时，要切换到英文输入法的模式。下面黑色的区域是消息提示区，会显示编译或者下载是否通过。

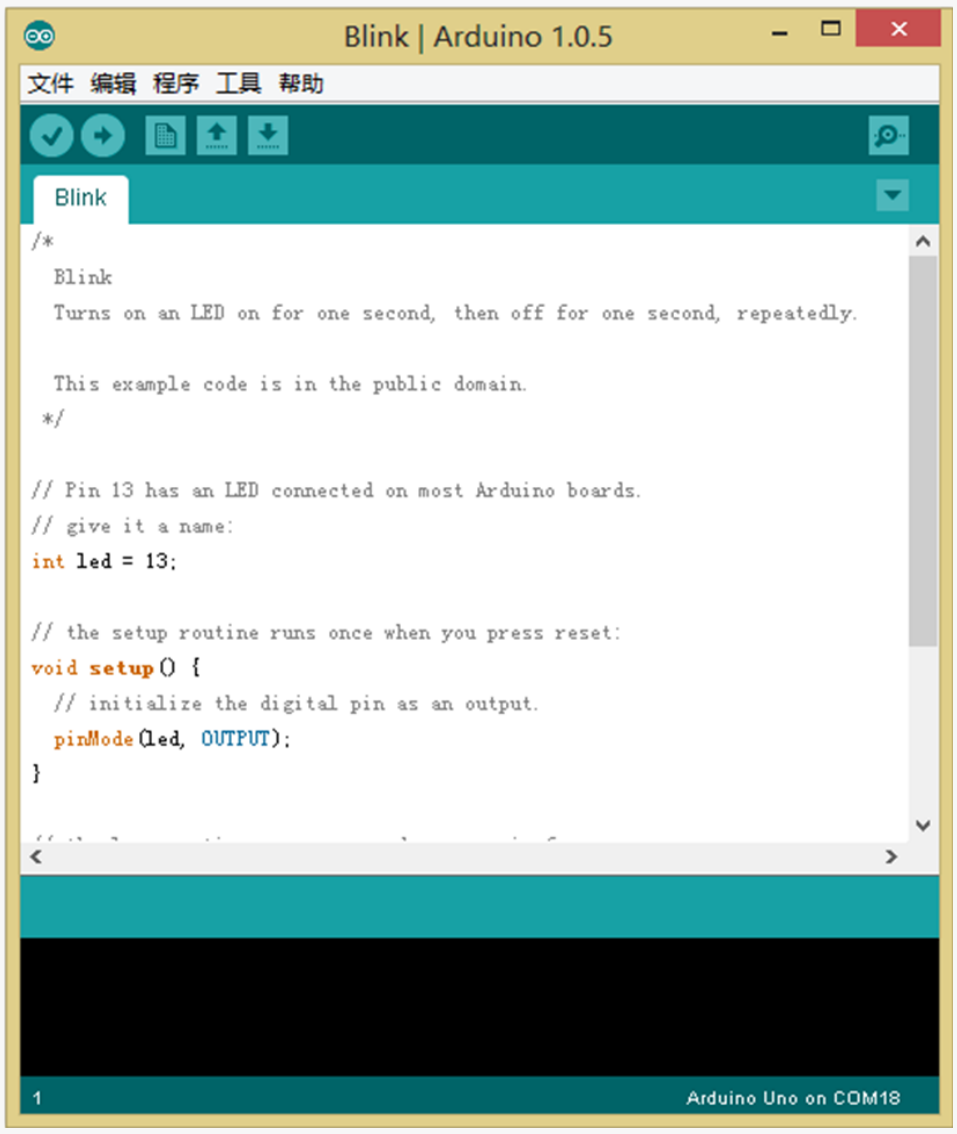
4. 下载一个Blink程序

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。



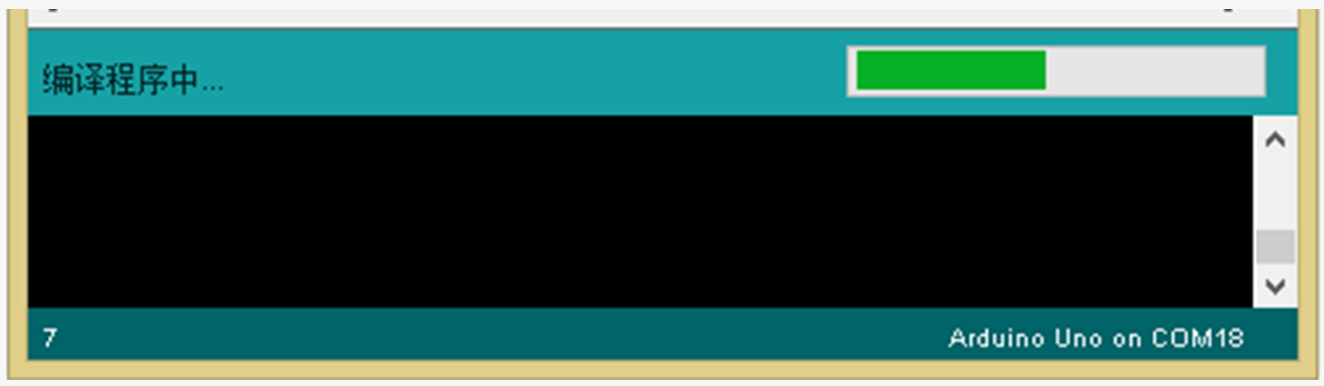
插上USB线，打开Arduino IDE后，找到“Blink”代码。





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。

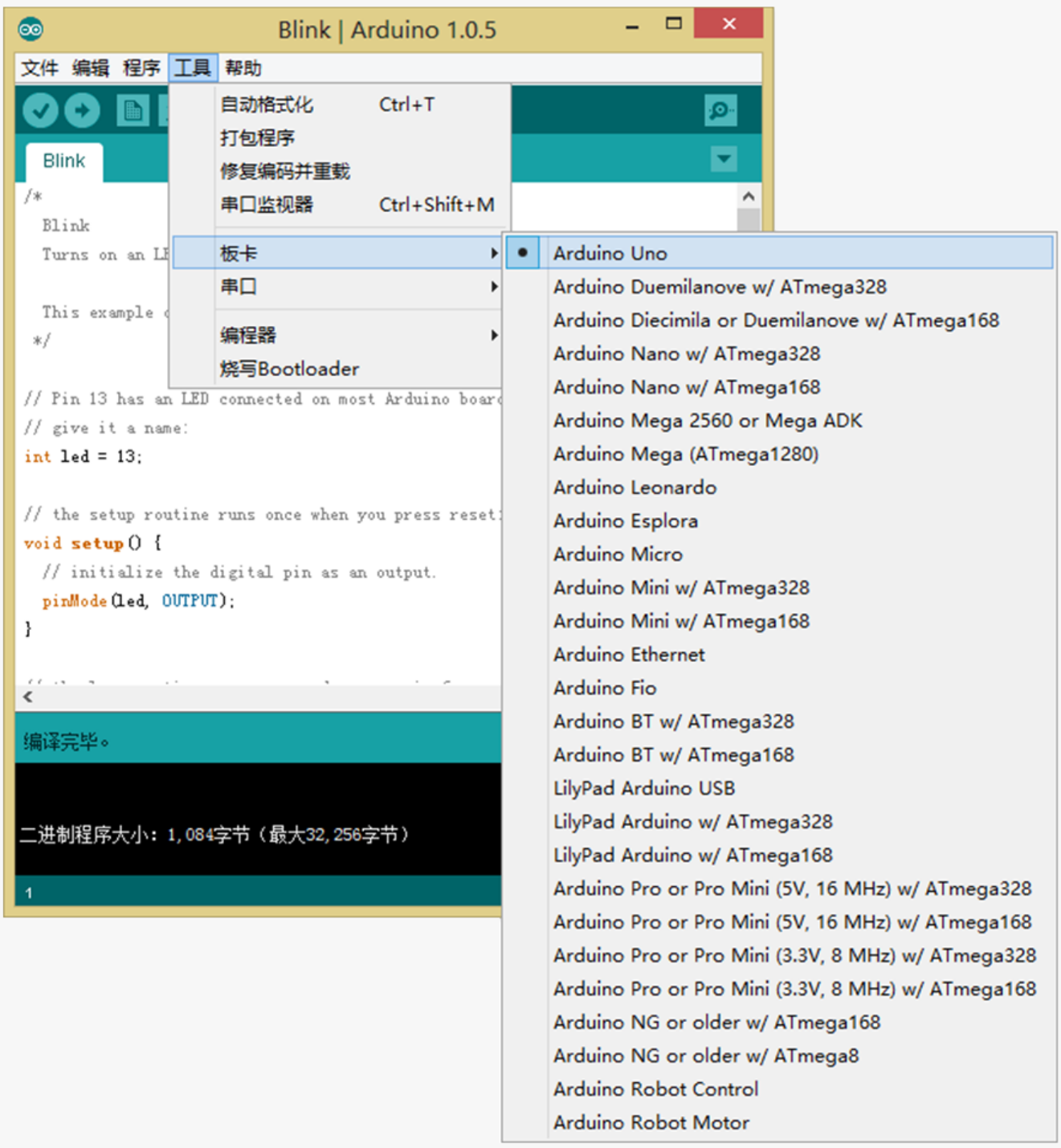


校验完毕!

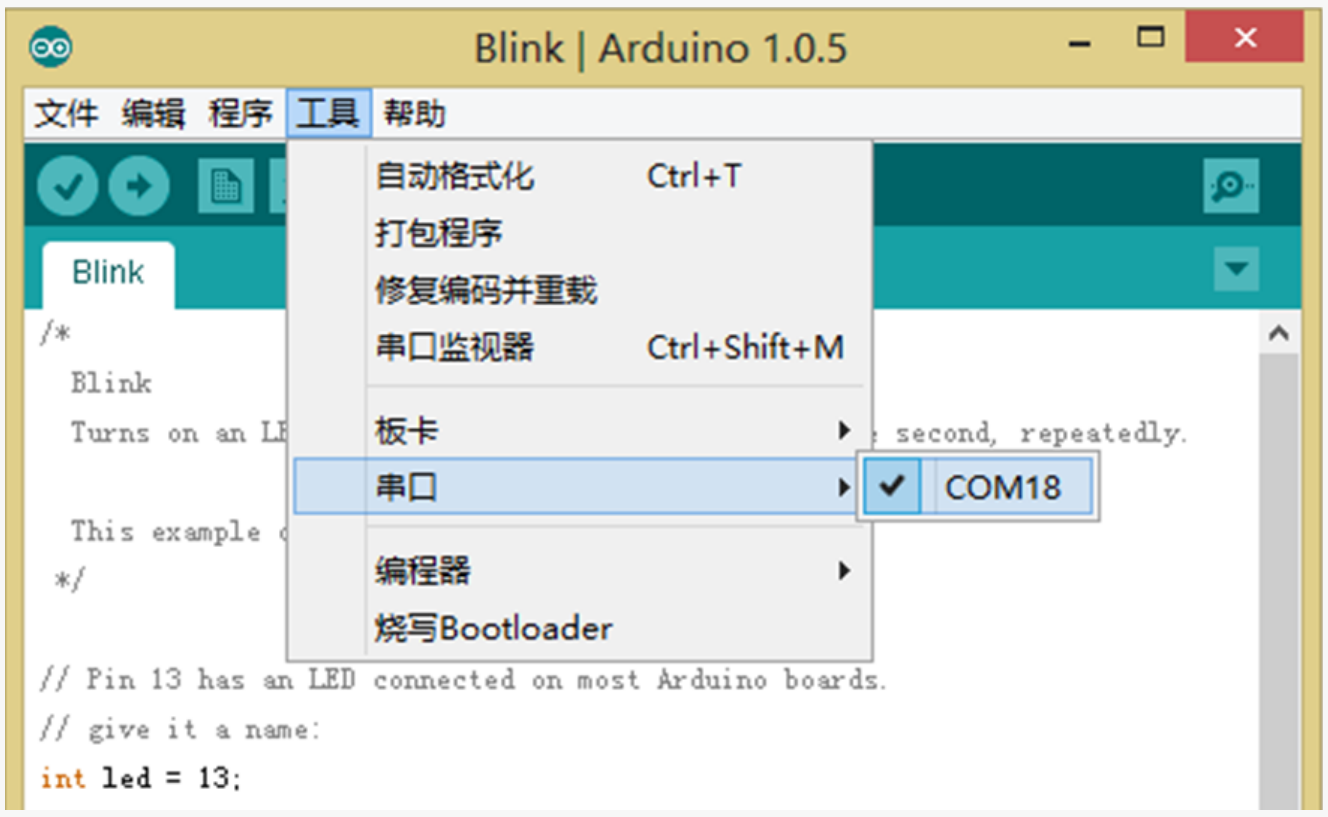
由于是样例代码，所以校验不会有错误，不过在以后写代码的过程中，输入完代码，都需要校验一下，然后再下载到Arduino中。

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!

项目一

LED闪烁

01

DFROBOT®
DRIVE THE FUTURE

让我们开始吧！

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，你将接触Arudino编程，编程其实也没你想象的那么困难。

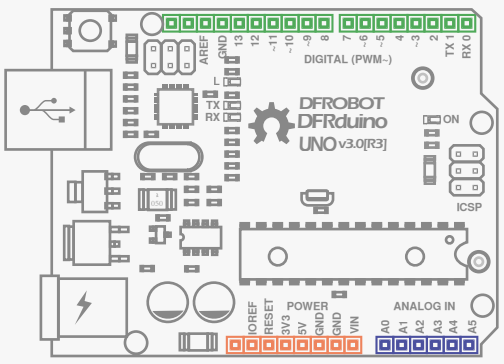
让我们从一个最基本的项目，使用Arduino控制一个外部LED的闪烁。

项目一 LED闪烁

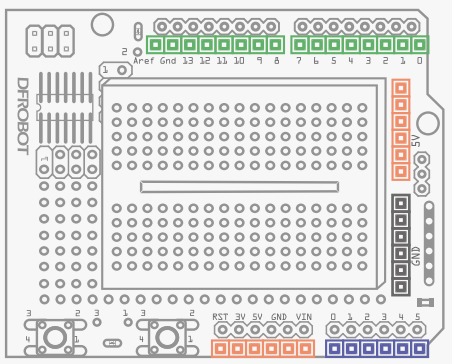
在第一个项目中，我们将重复使用之前的那个测试代码Blink程序。

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



x1 **DFduino UNO R3**
（及配套USB数据线）



x1 **Prototype Shield**
原型扩展板+面包板

x2 **Jumper Cables**
M/M
跳线(公公头)

x1 **Resistor 220R**
220欧电阻

x1 **5MM LED**
5MM LED灯

*这里仅为示意图，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歇下，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。取出所有元件，按照图1连接。

用绿色与红色的面包线连接，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+，短脚为-，完成连接后，给Arduino接上USB数据线，供电，准备下载程序。

DF中定义：

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

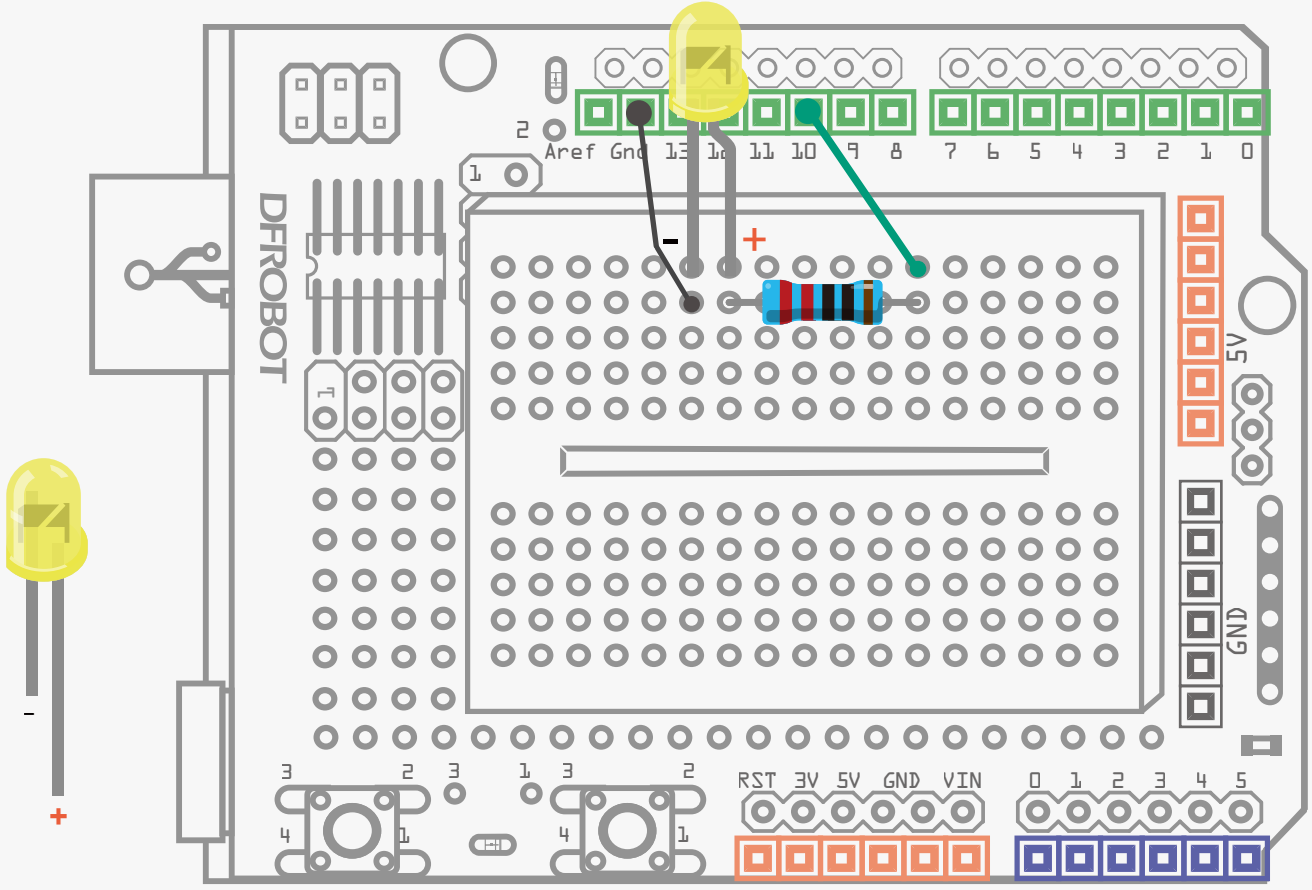


图1-1 LED闪烁连线图

输入代码

打开Arduino IDE，在编辑框中输入样例代码1-1所示代码。（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。）

样例代码 1-1 :

```
//项目一 —— LED 闪烁
/*
    描述：LED每隔一秒交替亮灭一次
*/
int ledPin = 10;
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和硬件，看看它们是如何工作的。

代码回顾

代码的第一行如下所示：

```
//项目一 —— LED 闪烁
```

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。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。

又如以下文字：

```
/* 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 */
```

这是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。

注释接下来的一行是：

```
int ledPin = 10;
```

变量类型 变量名

这就是所谓的变量声明，变量是用来存储数据的。这个例子，我们用的类型是int型或者说是整型，可以表示一个在-32768到32767之间的数。变量的类型，是由你存储的内容来决定的。这里我们存储的10这个整数。ledPin是变量名。（变量名其实就是这个变量的一个名字，代表这个值。当然，也可以不叫ledPin，按你的喜好来取），变量名的选取最好根据变量的功能来定。ledPin这里说明，这个变量表示LED和Arduino的数字引脚10相连。

在声明的最后用一个“;”来表示这句语句的结束。分号必不可少！必须切换到英文输入法中的分号。

何为变量？

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。

在给变量起名字时，还需要强调的一点。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橙色。

接下来是setup()函数：

在这个程序里有两个函数，一个叫做setup，它的目的主要的loop函数运行之前为程序做必要的设置。在Arduino中程序运行时将首先调用 setup() 函数。用于初始化变量、设置针脚的输出/输入类型、配置串口等等。每次 Arduino 上电或重启后，setup 函数只运行一次。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。

setup()函数内只有一条语句，那就是pinMode函数。

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

这个函数是用来设置Arduino数字引脚的模式的，只用于数字引脚定义是输入(INPUT)还是输出(OUTPUT)。在函数的括号内包含两个参数，引脚号以及引脚的模式。

函数格式如下：

pinMode就是一个函数的调用，只是这个函数已经在Arduino软件内部编写好了，所以我们也只需直接调用就可以了。在函数的括号内包含两个参数，就是需设定引脚号及引脚的模式，引脚号是ledPin，在我们程序的第一句话就声明过了，ledPin代表10，之后用到ledPin的地方，都可以理解为10的代名词。这条语句能试着理解了吗？这条语句想告诉Arduino，数字引脚10被设置为OUTPUT模式。

如果让你设置数字引脚2为输入模式，你会吗？
答案：pinMode(2，INPUT);

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，并且后面的括号内为空，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。

INPUT与OUTPUT的区别

我是这么理解的，INPUT是输入的信号，是外部给控制器的信号，根据外部环境变化才给到控制器信号。比如像我们之后会用到的按钮，它就是典型的 INPUT模式，它需要我们按下按键后，控制器才能接收到外部给它的指令。

OUTPUT是输出信号，你需要让控制器能反应出某些特征，向外界发出信号，典型的就是LED，它闪烁的过程就是向外部发出信号的过程。又比如我们后面会用到的蜂鸣器，一个会发出声音的玩意儿，发声的过程就是向外界发出信号的过程，所以它也是OUTPUT。

我们接着往下看，程序现在进行到我们的主函数loop()：

```
void loop() {
      digitalWrite(ledPin,HIGH);
      delay(1000);
      digitalWrite(ledPin,LOW);
      delay(1000);
}
```

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

在 setup() 函数中初始化和定义了变量后，就开始执行 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。

在这个项目中，我们希望LED灯亮，保持1秒，然后关闭，保持1秒，然后一直重复上面的动作。那么在Arduino的语句中，该怎么实现呢?

先看loop()函数内的第一条语句，这里我们涉及到了另外一个函数就是digitalWrite()。

```
digitalWrite(ledPin,HIGH);
```

函数格式如下：

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

引脚号　值 HIGH/LOW

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。我们这里就是给引脚10（ledPin）一个5V的高电平，点亮了引脚10这个LED。我们这里强调了，pinMode()被设置为OUTPUT时，才用到digitalWrite()。这是为什么呢？看一下下面这段话。

pinMode() digitalWrite() digitalRead() 的关系

前面说了pinMode()中的INPUT与OUTPUT设置是有讲究的，是由器件本身的功能决定的。然而，前面设置INPUT和OUTPUT与之后程序需要如何执行也有着紧密关系的。既然pin是OUTPUT的话，那势必是控制器Arduino要给外界信号，所以需要Arduino给引脚先写入信号——digitalWrite()。我们这里还没用到digitalRead()，就先说了吧！如果pin是INPUT的话，是外界给了控制器Arduino信号，所以需要Arduino读取引脚信号——digitalRead()。
对于初学者来说，可以先学着用，再慢慢弄明白里面的原由。pinMode()设置为OUTPUT,对应使用digitalWrite()。INPUT对应使用digitalRead()。下表是一张对应表：

比如：LED、蜂鸣器	比如：按键控制
pinMode(pin,OUTPUT)	pinMode(pin,INPUT)
digitalWrite(pin,HIGH/LOW)	digitalRead(pin)

接着的一句语句：

```
delay(1000);
```

接着看下一句是：

```
digitalWrite(ledPin,LOW);
```

delay()函数，用于延时等待。等待1000毫秒（1000毫秒也就是1秒，以此类推吧！）。我们举一反三一下，如果我们需要延时2秒呢？答案：delay(2000);

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，为引脚10一个0V的低电平，也就是熄灭LED。

然后再延时1秒。之后回到loop()函数开始部分，循环运行。

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（250毫秒），就像汽车报警器上的LED指示灯那样。试着写一下：

答案：

```
void loop() {
    digitalWrite(ledPin,HIGH);
    delay(250);
    digitalWrite(ledPin,LOW);
    delay(5000);
}
```

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。外面的灯光效果都是基于这样的原理。让我们再来看下硬件。看看硬件又是如何工作的。

硬件回顾

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。

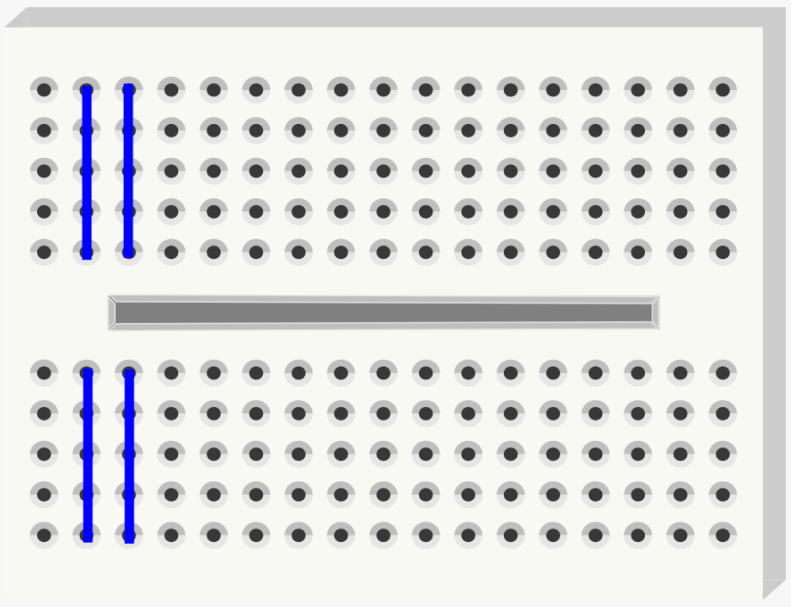


图1-2 面包板内部导通图

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。

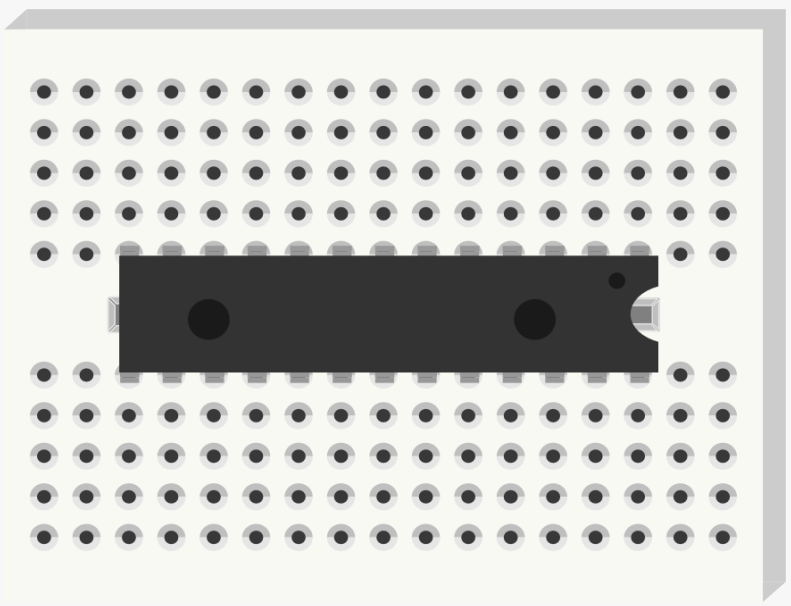


图1-3 插入DIP芯片后

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。感兴趣的可以读读看阻值。

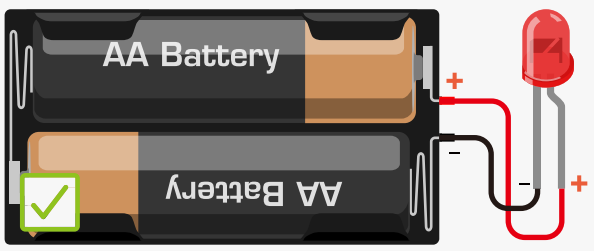
提供一个五色坏电阻阻值在线计算器：
http://www.21ic.com/tools/component/201003/54192.htm

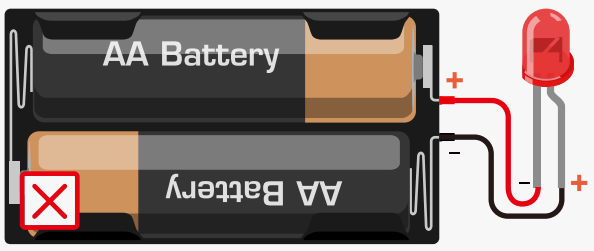
LED

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。）

LED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

02

本项目将继续使用项目1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。这是国际莫尔斯码求救信号。莫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点和横杠。我们这个项目中，我们就拼写S.O.S这三个字母。

通过查阅莫尔斯码表，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。

有了前一个项目的基础，不难理解下面样例代码2-1。但先不要急着输入这段代码，只是看一下。

样例代码2-1：

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

                                    // 三个快闪烁来表示字母“S”
                                          digitalWrite(ledPin,HIGH);
                                          delay(150);
                                          digitalWrite(ledPin,LOW);
                                          delay(100);

                                          digitalWrite(ledPin,HIGH);
                                          delay(150);
                                          digitalWrite(ledPin,LOW);
                                          delay(100);

                                          digitalWrite(ledPin,HIGH);
                                          delay(150);
                                          digitalWrite(ledPin,LOW);
                                          delay(100);

                                          delay(100);    //100毫秒延时产生字母之间的间隙
```

```
//三个短闪烁来表示字母“O”
    digitalWrite(ledPin,HIGH);
    delay(400);
    digitalWrite(ledPin,LOW);
    delay(100);

    digitalWrite(ledPin,HIGH);
    delay(400);
    digitalWrite(ledPin,LOW);
    delay(100);

    digitalWrite(ledPin,HIGH);
    delay(400);
    digitalWrite(ledPin,LOW);
    delay(100);

    delay(100);    //100毫秒延时产生字母之间的间隙

//再用三个快闪烁来表示字母“S”
    digitalWrite(ledPin,HIGH);
    delay(150);
    digitalWrite(ledPin,LOW);
    delay(100);

    digitalWrite(ledPin,HIGH);
    delay(150);
    digitalWrite(ledPin,LOW);
    delay(100);

    digitalWrite(ledPin,HIGH);
    delay(150);
    digitalWrite(ledPin,LOW);
    delay(100);

    delay(5000);    // 在重复S.0.S信号前等待5秒
}
```

输入代码

上面的写法固然正确，可是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，所以有了我们更好的一种写法。
我们先来看一下样例代码2-2。

样例代码2-2：

```
//项目二 -- S.O.S信号
int ledPin = 10;
void setup() {
      pinMode(ledPin, OUTPUT);
}
void loop() {
       // 三个快闪烁来表示字母“S”
       for(int x=0;x<3;x++){
          digitalWrite(ledPin,HIGH);          //设置LED 为开
          delay(150);                         //延时150毫秒
          digitalWrite(ledPin,LOW);           //设置LED 为关
          delay(100);                         //延时100毫秒
       }
 //100毫秒延时产生字母之间的间隙
 delay(100);

//三个短闪烁来表示字母“O”
for(int x=0;x<3;x++){
     digitalWrite(ledPin,HIGH);               //设置LED 为开
     delay(400);                              //延时400毫秒
     digitalWrite(ledPin,LOW);                //设置LED 为关
     delay(100);                              //延时100毫秒
}

//100毫秒延时产生字母之间的间隙
delay(100);
```

```
    // 再用三个快闪烁来表示字母“S”
    for(int x=0;x<3;x++){
          digitalWrite(ledPin,HIGH);          //设置LED 为开
          delay(150);                         //延时150毫秒
          digitalWrite(ledPin,LOW);           //设置LED 为关
          delay(100);                         //延时100毫秒
     }

    // 在重复S.0.S信号前等待5秒
    delay(5000);
}
```

在输入代码的时候，注意保持代码的一个层次感，除了美观外，也便于你日后检查代码。确认正确后，下载代码到Arduino中，如果一切顺利的话，我们将看到LED闪烁出莫尔斯码S.O.S信号，等待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。我们接着来分析下代码。

代码回顾

代码的第一部分与上个项目是完全一样的。也是初始化一个变量，设置数字引脚10的模式为输出模式。在主函数loop()中，你可以看到与上一个项目中类似的语句用来控制LED的开和关，并保持一段时间。然而，这次不同的是，主函数包含了三个独立的代码段。

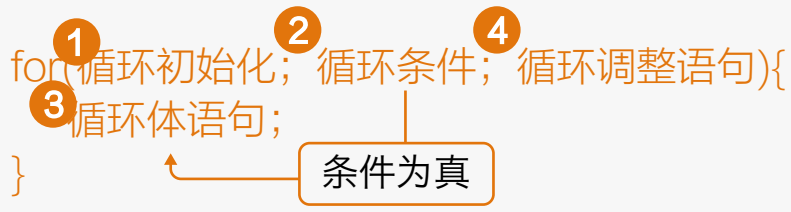
第一段代码是输出是三个点：

```
for(int x=0;x<3;x++){
      digitalWrite(ledPin,HIGH);        //设置LED 为开
      delay(150);                       //延时150毫秒
      digitalWrite(ledPin,LOW);         //设置LED 为关
      delay(100);                       //延时100毫秒
 }
```

LED开关的语句是包含在一对花括号内的，因此为一组代码段。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。

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。产生这样效果的是因为使用了for循环。for语句通常在程序中用作循环使用。我们来看一下：

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：

```
for(int x=0;x<3;x++){
      ……
}
```

第一步：初始化变量x=0。
第二步：判断x是否小于3。
第三步：判断第二步成立，for循环中执行LED开与关。
第四步：x自加，变为2。
（x++这句话表示把x的值增加1，等同于写成x=x+1，也就是把x当前的值变为x+1，再赋给x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x=2，判断是否小于3。
第六步：重复第三步。
……
直到x循环到3时，判断x<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x<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 x=0;x<100;x++){}

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。

比较运行符

“<”称之为比较运算符。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，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，长闪烁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，短闪烁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”。

必须要注意的，我们这里要引用一个新的概念，是局部变量和全局变量。局部变量，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x，它就是个局部变量，所以说，虽然每个for循环中都有一个变量x，但它们不冲突就是这样原因，它们只在自己的循环中执行。还有一种变量叫全局变量，不同之处是，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。就像我们这里的ledPin，能在整个程序中起作用。

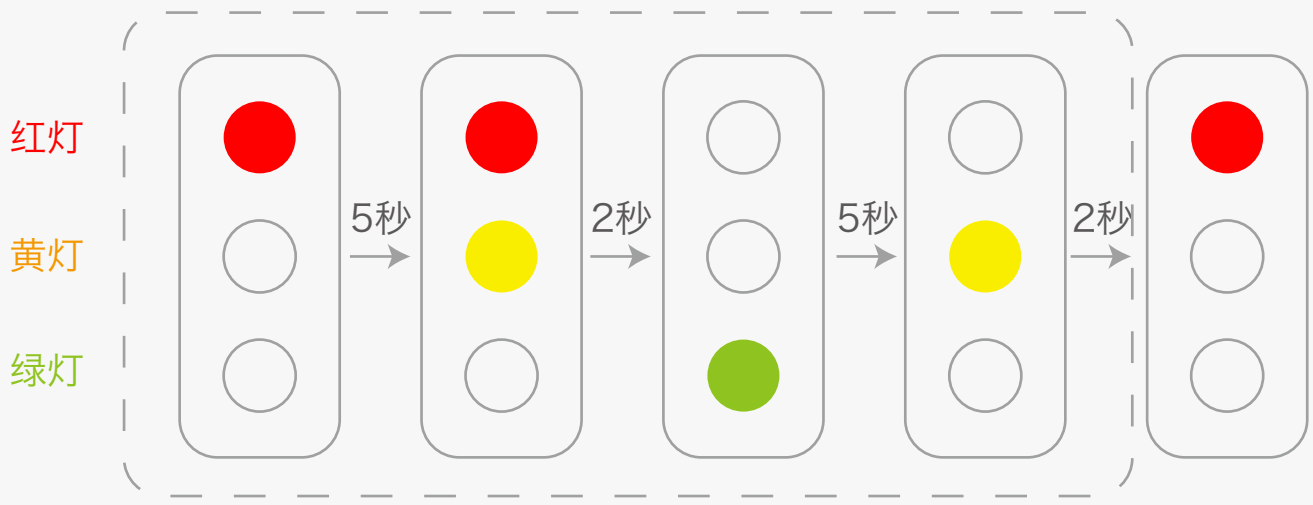
在每for循环之间有个小延时100毫秒，是S.O.S字母之间有个清晰的停顿说明。最后，在回到主函数loop重新执行一遍之前，有个相对较长的延时，为5秒。

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

我们学习了两个项目的基础，现在做个课后习题吧~做个交通信号灯，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

03

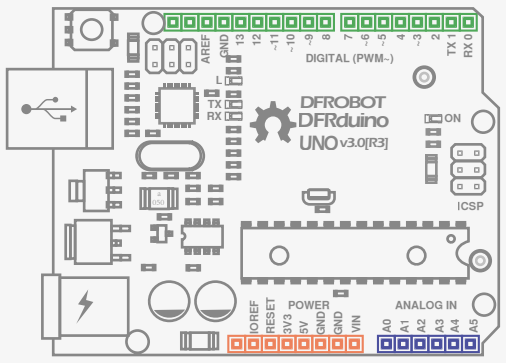
DFROBOT®
DRIVE THE FUTURE

项目三 互动交通信号灯

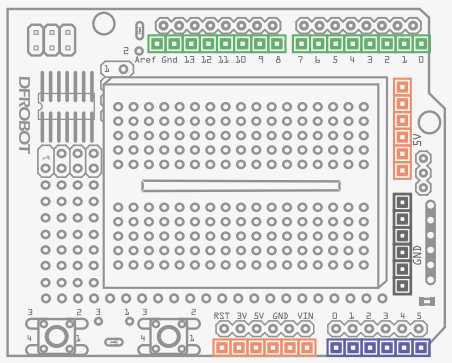
有没有试着做上面那个课后作业呢？做出来的话，说明你已经基本掌握上面所学的东西了，如果不会也没关系，我相信，看完这个章节，前面那个问题就不攻自破了！我们这回就基于上面这个交通灯来进行一个拓展，增加一种行人按键请求通过马路的功能。当按钮被按下时，Arduino会自动反应，改变交通灯的状态，让车停下，允许行人通过。

这个项目中，我们开始要实现Arduino的互动了，也会在代码学习到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，相信你能收获不少！

所需元件



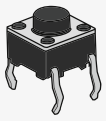
DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x13

Resistor 220R
220欧电阻 x6



Pushbutton
按键开关 x1

5MM LED
5MM LED灯 x2

5MM LED
5MM LED灯 x2

5MM LED
5MM LED灯 x1

*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。

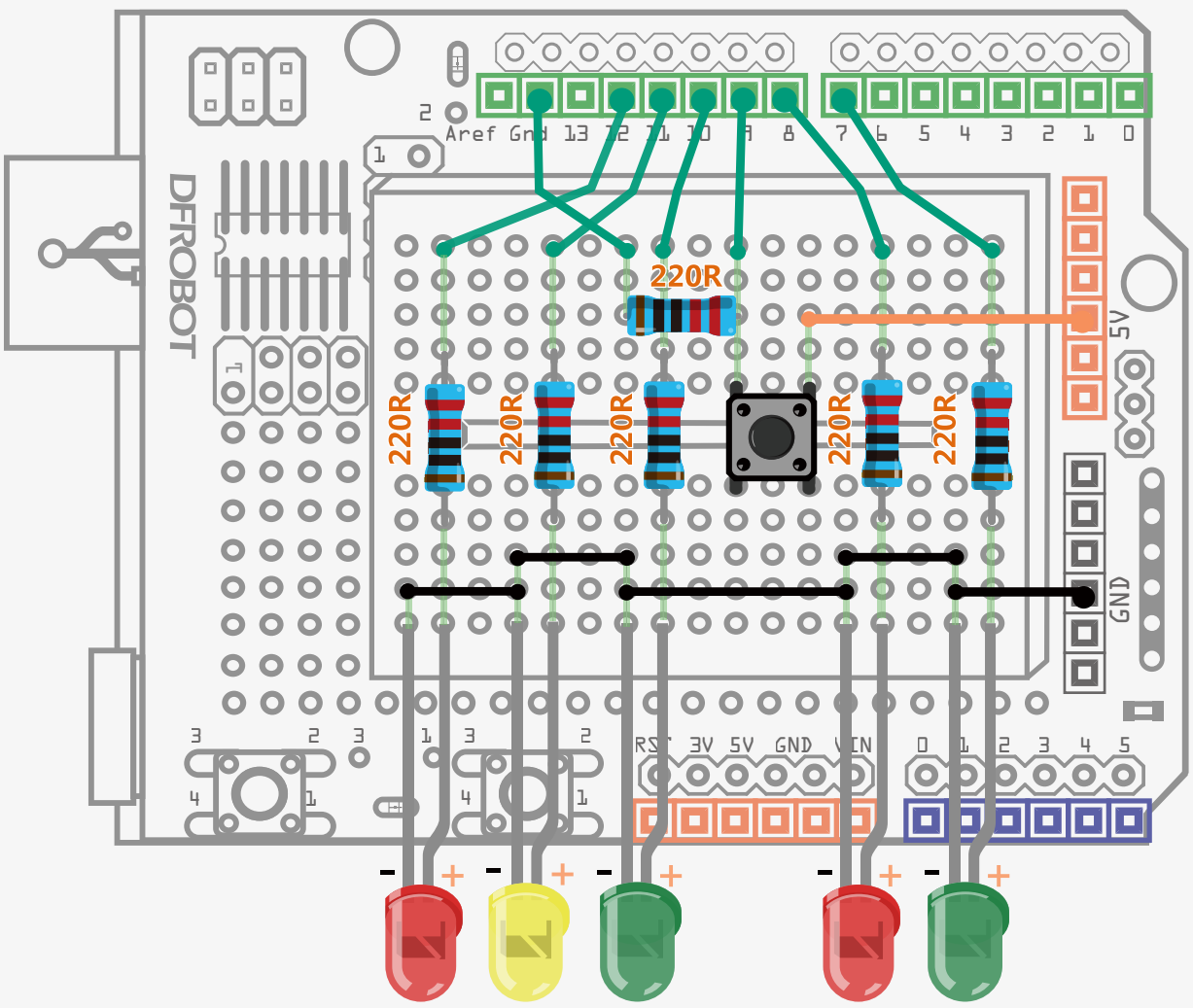


图3-1 LED闪烁连线图

输入代码

输入下面的样例代码3-1，这段代码引自《beginning-arduino》一书。

样例代码 3-1 :

```
//项目三 -- 互动交通信号灯
int carRed = 12;          //设置汽车灯
int carYellow = 11;
int carGreen = 10;
int button = 9;           //按钮引脚
int pedRed = 8;           //设置行人灯
int pedGreen = 7;
int crossTime = 5000;                    //允许行人通过的时间
unsigned long changeTime;                //按钮按下后的时间

void setup() {
        //所有LED设置为输入模式
        pinMode(carRed, OUTPUT);
        pinMode(carYellow, OUTPUT);
        pinMode(carGreen, OUTPUT);
        pinMode(pedRed, OUTPUT);
        pinMode(pedGreen, OUTPUT);
        pinMode(button, INPUT);          //按钮设置为输入模式
        digitalWrite(carGreen, HIGH);  //开始时，汽车灯绿灯
        digitalWrite(pedRed, LOW);     //行人灯为红灯
}

void loop() {
        int state = digitalRead(button);
        //检测按钮是否被按下，并且是否距上次按下后有5秒的等待时间
        if(state == HIGH && (millis() - changeTime)> 5000){
       //调用变灯函数
       changeLights();
       }
}
```

```
void changeLights() {
        digitalWrite(carGreen, LOW);       //汽车绿灯灭
        digitalWrite(carYellow, HIGH);     //汽车黄灯亮
        delay(2000);                       //等待2秒

        digitalWrite(carYellow, LOW);      //汽车黄灯灭
        digitalWrite(carRed, HIGH);        //汽车红灯亮
        delay(1000);                       //为安全考虑等待1秒

        digitalWrite(pedRed, LOW);         //行人红灯灭
        digitalWrite(pedGreen, HIGH);      //行人绿灯亮

        delay(crossTime);                  //等待一个通过时间

        //闪烁行人灯绿灯，提示可过马路时间快到
        for (int x=0; x<10; x++) {
                digitalWrite(pedGreen, HIGH);
                delay(250);
                digitalWrite(pedGreen, LOW);
                delay(250);
        }
        digitalWrite(pedRed, HIGH);        //行人红灯亮
        delay(500);

        digitalWrite(carRed, LOW);         //汽车红灯灭
        digitalWrite(carYellow, HIGH);     //汽车黄灯亮
        delay(1000);
        digitalWrite(carYellow, LOW);      //汽车黄灯灭
        digitalWrite(carGreen, HIGH);      //汽车绿灯亮

        changeTime = millis();             //记录自上一次灯变化的时间
        //返回到主函数循环中
}
```

下载完成后，可以尝试按下按钮。看看是个什么的效果？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，那么行人灯就开始由红变绿，汽车灯由绿变黄，变红。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。

整段代码看起来很复杂，其实理清一下思路并不难。如果你还是没有办法理不清里面变化关系的话，可以试着画一个示意图，像项目2的课后作业那样，这样一来可能会方便你理解程序。

代码回顾

通过前面两个项目，你应该能够理解这个代码的大部分内容。代码开始是一串的变量的声明，在声明中，出现了一个新名词。这里就解释一下这个新名词：

```
unsigned long changeTime;
```

这是一个新的变量类型。我们之前，只创建过int整型变量，它可以存放一个-32768到32767之间的整数。这次要创建的是一个long的变量类型，它可以存放一个-2147483648到2147483647之间的整数。而unsigned long数据类型，则不存储负数，所以存储的范围就从0到4294967295.

如果我们使用一个int型的话，信号灯状态变化的时间，只能存储最大32秒（32768毫秒约为32秒），一旦出现变量溢出就会造成程序运行出现错误，所以为了避免这样的情况，要选用能存储更大数的一个变量，并且不为负，我们就可以考虑使用unsigned long型。你可以用笔算下，这个变量最大能存储的数，时间可达49天。

随即进入setup()函数，对LED和按钮进行一些设置，在设置时，需要注意到的是：

```
pinMode(button, INPUT);
```

pinMode()函数我们已经很熟悉了，在项目一的时候就介绍过，只是和LED有所不同的是，按钮要设置为INPUT。

在setup()函数中，先给定行人灯和汽车灯的一个初始状态：

```
digitalWrite(carGreen, HIGH); //开始时，汽车灯绿灯
digitalWrite(pedRed, LOW);   //行人灯为红灯
```

进入到的主程序中的第一句，就是来检测button（引脚9）的状态的：

```
int state = digitalRead(button);
```

变量这个盒子无限大吗？

有人会问为什么有些变量类型可以存储很大的数，而有些变量类型不行呢？这是由变量类型所占的存储空间决定的。就拿我们前面讲变量的时候举过得例子，变量好比用来放东西的盒子，那不同类型的变量想象成不同大小的盒子，int的盒子比unsigned long的盒子小，所以放的东西当然少啦！这样解释是不是比较容易理解这个概念呢?

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。所以，我们要尽量的少用存储空间，能不用则不用。

下表列出了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（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（单精度浮点型）	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。

此时，一个新函数出现了——digitalRead()!

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，这里需要读取是按键信号，按键所在引脚是数字引脚9，由于前面做了声明，所以这里用button。
并且把读到的信号传递给变量state，用于后面进行判断。state为 HIGH或者说为1时，说明按键被按下了。state为LOW或者0，表明按键没被按下。

所以，可以直接检查state的值来判断按钮是否被按下：

```
if(state == HIGH && (millis() - changeTime)> 5000) {
    //调用变灯函数
    changeLights();
}
```

这里涉及新的语句-- if语句。

```
if(表达式){
    语句;
}
```

if语句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。
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state变量为HIGH。如果按键被按下，state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。

```
(millis() - changeTime)> 5000)
```

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。这样做的目的是，防止重复按键而导致的运行错误。

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，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() 函数内部就不做说明了。

逻辑运算符

前面说到的&&是一个逻辑运算符，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）

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。

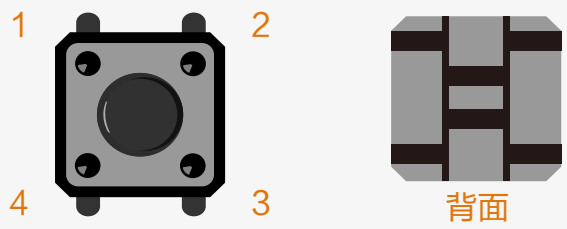


图 3-2 按钮结构图

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（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）。

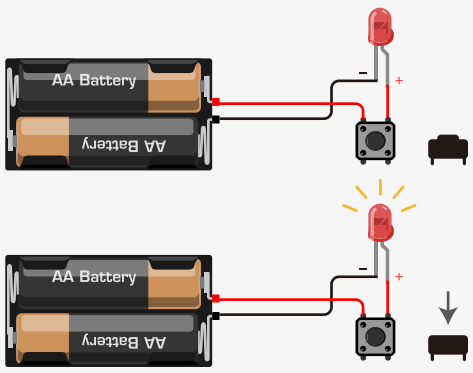


图 3-4 按钮示意图

什么是下拉电阻?

下拉电阻这个名词可能比较抽象，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。

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。

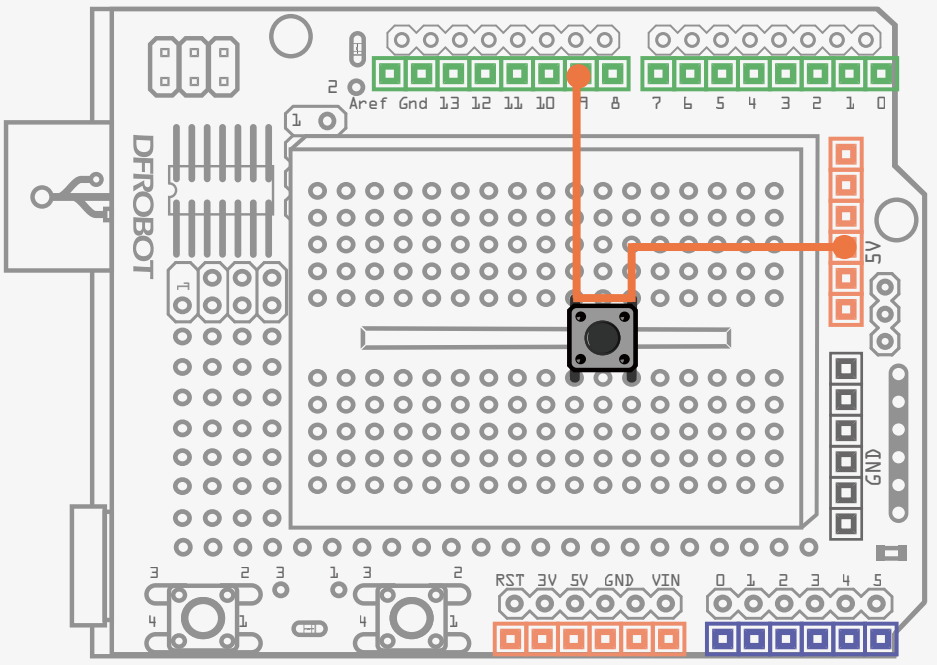
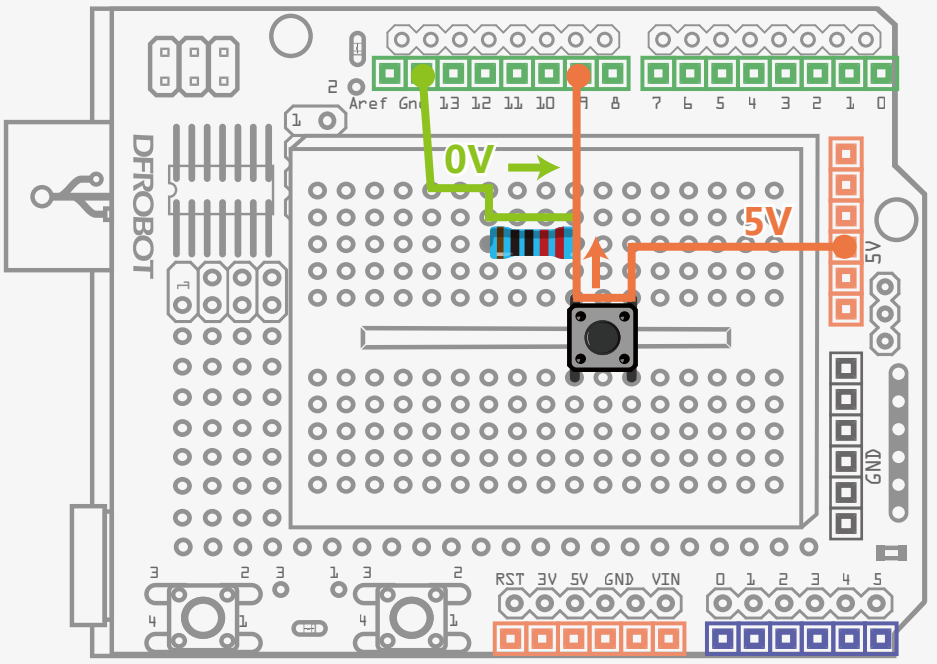


图 3-5 未接下拉电阻



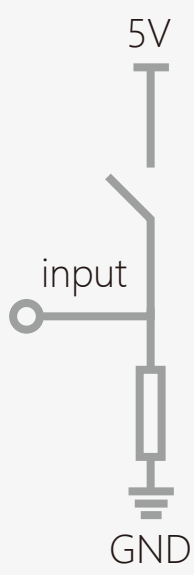
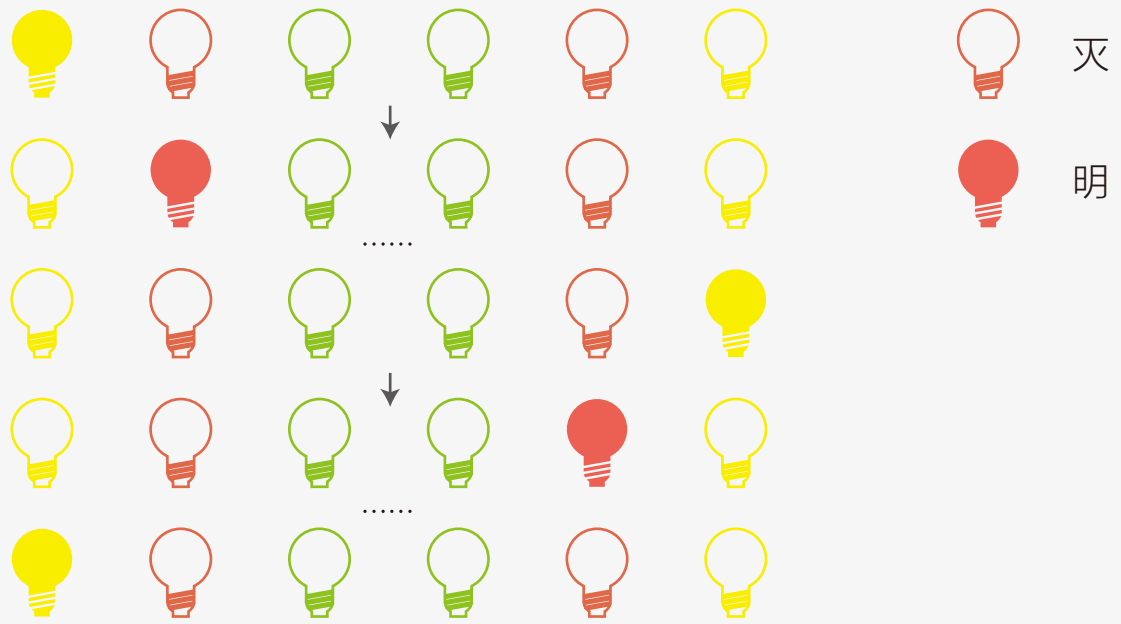


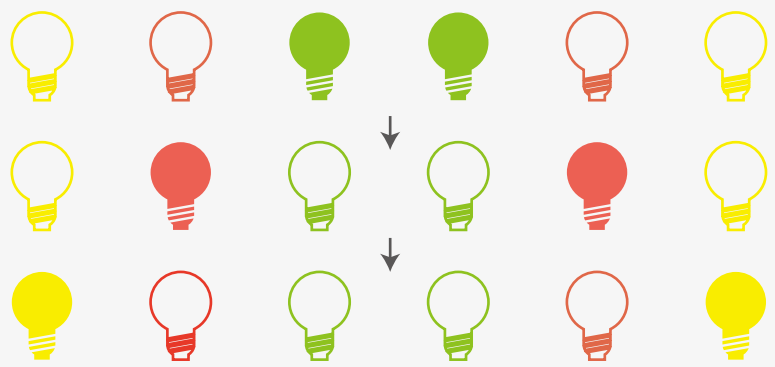
图 3-6 有下拉电阻

课后作业

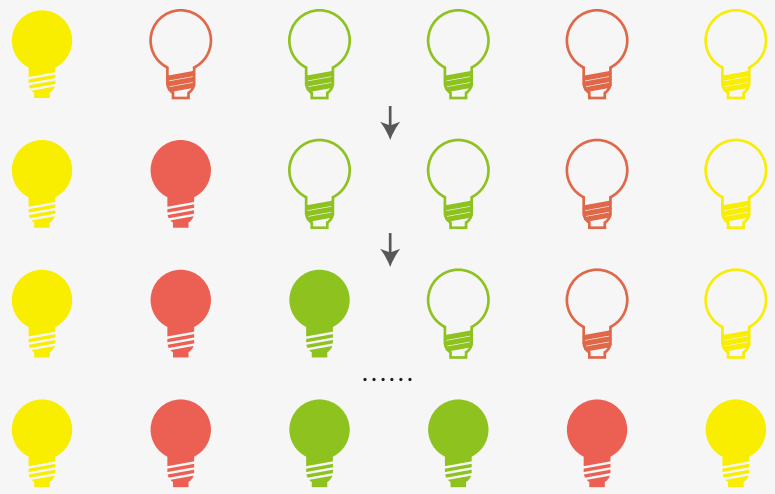
1、选择任意颜色LED 6个，做一个流水灯的效果，6盏灯从左至右依次点亮，然后再从右至左依次熄灭。



2、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。



3、再比如，从左至右，依次亮起1个，2个，3个……



4、再结合按钮，用按键开关和LED互动。（提供供参考教程）

○1 用一个按键，按一下控制灯亮，再按一下控制灯灭。

http://www.dfrobot.com.cn/index.php?route=sns/tutorials/detail&tid=1175

○2又或者用两个按键，一个控制灯亮，另一个控制灯灭。

http://learn.adafruit.com/adafruit-arduino-lesson-6-digital-inputs?view=all

玩儿法有很多，就全靠你的想象了！

项目四

呼吸灯

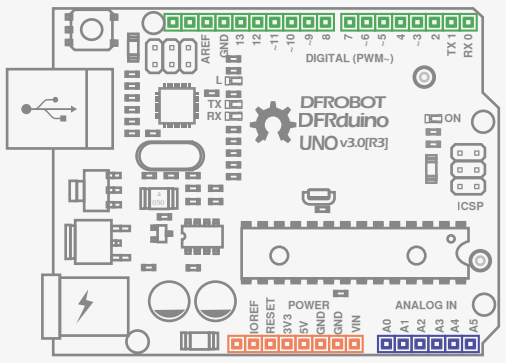
04

DFROBOT®
DRIVE THE FUTURE

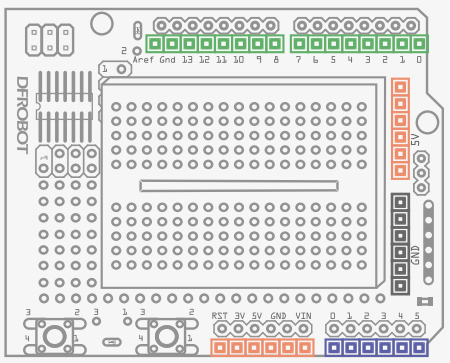
项目四 呼吸灯

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



DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x2

Resistor 220R
220欧电阻 x1

5MM LED
5MM LED灯 x1

硬件连接

这个项目的硬件连接与项目一是完全相同的。
如有不明白可以回看项目一。

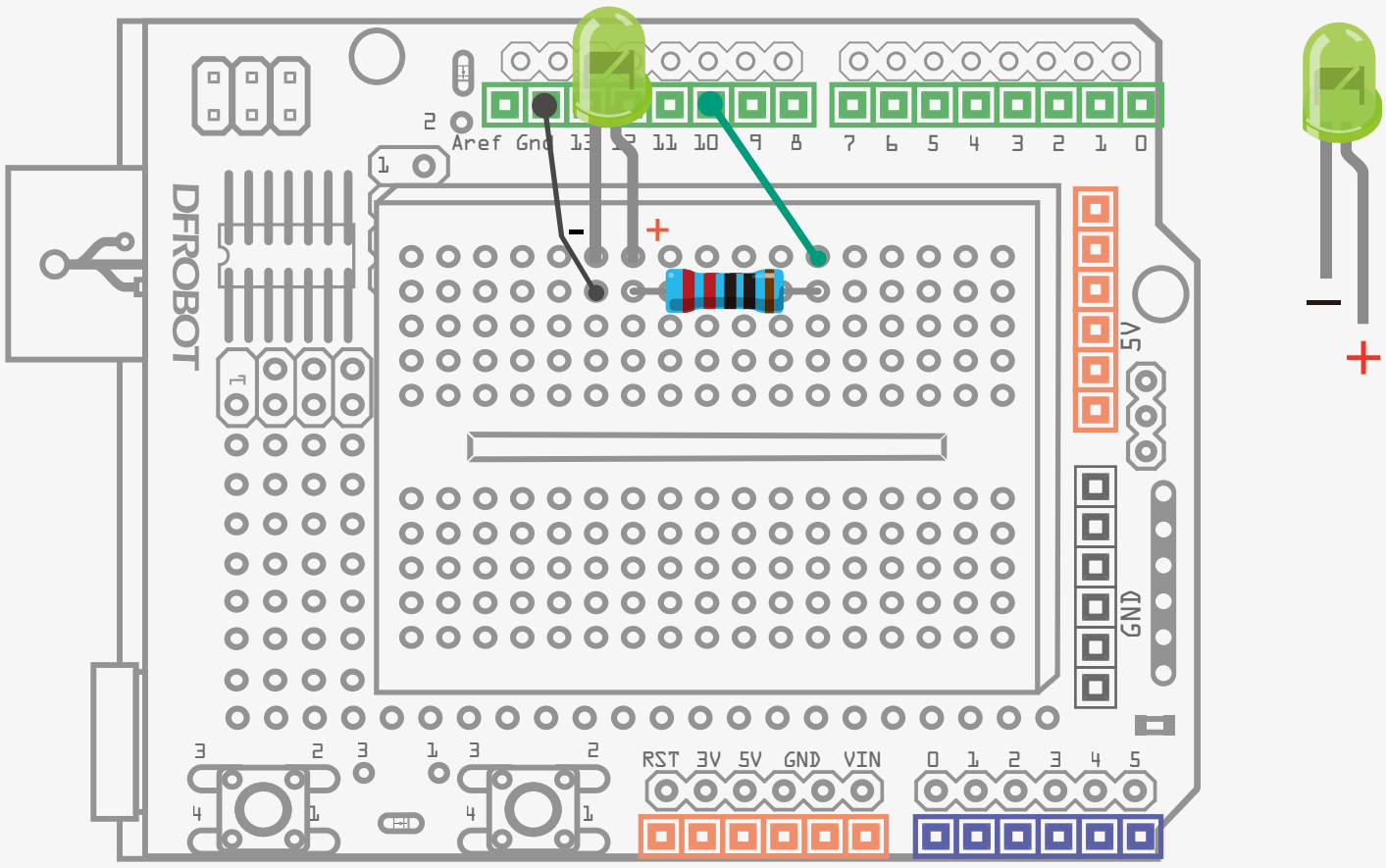


图 4-1 呼吸灯连线图

输入代码

样例代码4-1 :

```
//项目四 - 呼吸灯
int ledPin = 10;

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

void loop(){
      fadeOn(1000,5);
      fadeOff(1000,5);
}

void fadeOn(unsigned int time,int increament){
       for (byte value = 0 ; value < 255; value+=increament){
          analogWrite(ledPin, value);
          delay(time/(255/5));
          }
}

void fadeOff(unsigned int time,int decreament){
       for (byte value = 255; value >0; value-=decreament){
          analogWrite(ledPin, value);
          delay(time/(255/5));
          }
}
```

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

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、for循环、以及函数调用。

在主函数中，只有两个调用函数，先看其中一个就能明白了。

```
void fadeOn(unsigned int time,int increament){

    for (byte value = 0 ; value < 255; value+=increament){

      analogWrite(ledPin, value);
      delay(time/(255/5));
      }
}
```

fadeOn()函数有两个传递参数，从参数名称中就可以简单看出，int time指的是时间，int increament指的是增量。函数中包含了一个for循环，循环条件是value<255，变量的增量由 increament决定。

for语句中涉及了一个新函数，

```
analogWrite(ledPin, value)
```

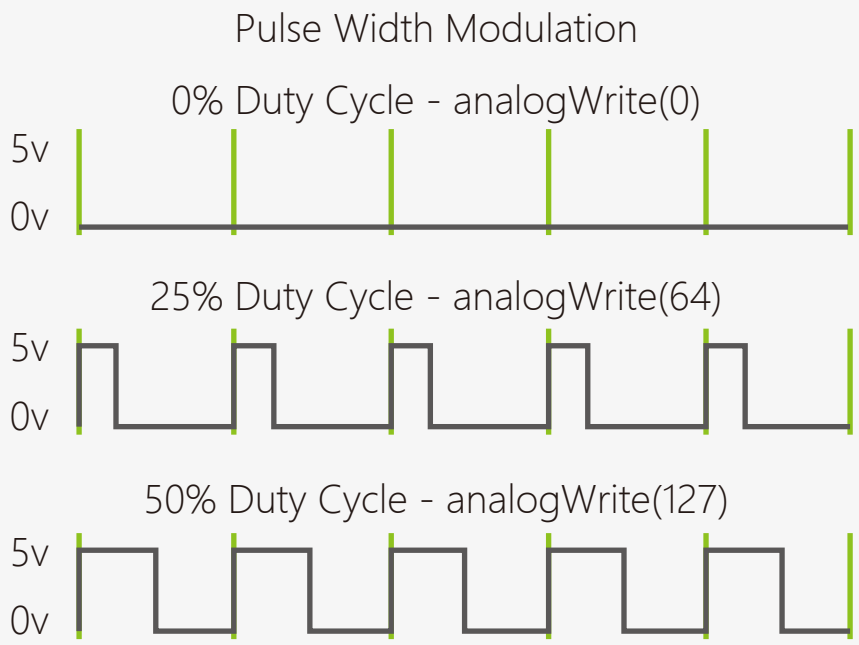
如何发送一个模拟值到一个数字引脚呢？就要用到该函数，使用这个函数是要具备特定条件的——该数字引脚需具有PWM功能。观察一下Arduino板，查看数字引脚，你会发现其中6个引脚（3、5、6、9、10、11）旁标有“~”，这些引脚不同于其他引脚，因为它们可以输出PWM信号。

函数格式如下：

analogWrite()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Arduino小车时，更能体现PWM的好处。

这一章介绍结束了！同样的硬件连接，通过软件的变化，可以呈现出完全不一样的效果，是不是觉得Arduino很神奇！

课后作业

1、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。找个用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()。random()可是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，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。

具体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。

点击查看： DFRobot 中文版 Arduino编程参考手册(位置：入门教程 - Arduino编程参考手册)
http://wiki.dfrobot.com.cn/index.php/%E9%A6%96%E9%A1%B5#.E5.85.A5.E9.97.A8.E6.95.99.E7.A8.8B
点击查看： Arduino 官方编程参考手册
http://arduino.cc/en/Reference/HomePage

2、再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。

可以参考程序: http://www.geek-workshop.com/thread-1054-1-1.html

项目五

炫彩RGB LED

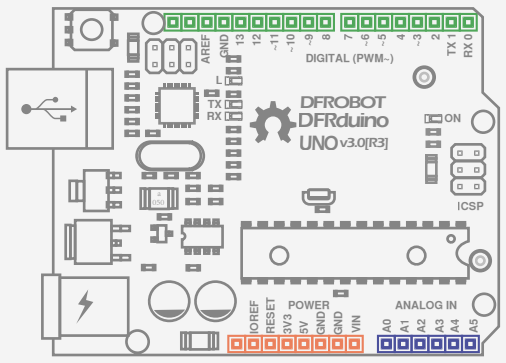
05

DFROBOT®
DRIVE THE FUTURE

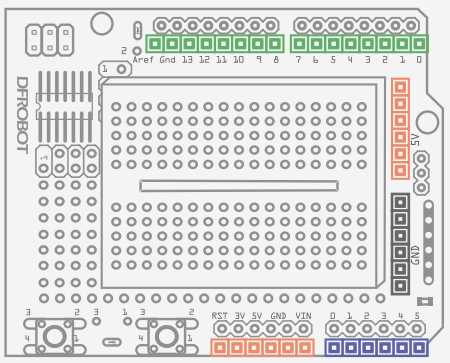
项目五 炫彩RGB LED

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



DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x4

Resistor 220R
220欧电阻 x3

5mm RGB LED
5mm RGB LED灯 x1

硬件连接

连接之前，先判别RGB是共阴还是共阳，如果不是很清楚的，可以先跳到这个项目的硬件部分介绍。连接时，还需注意一点，引脚的顺序，可参照右边的引脚图。

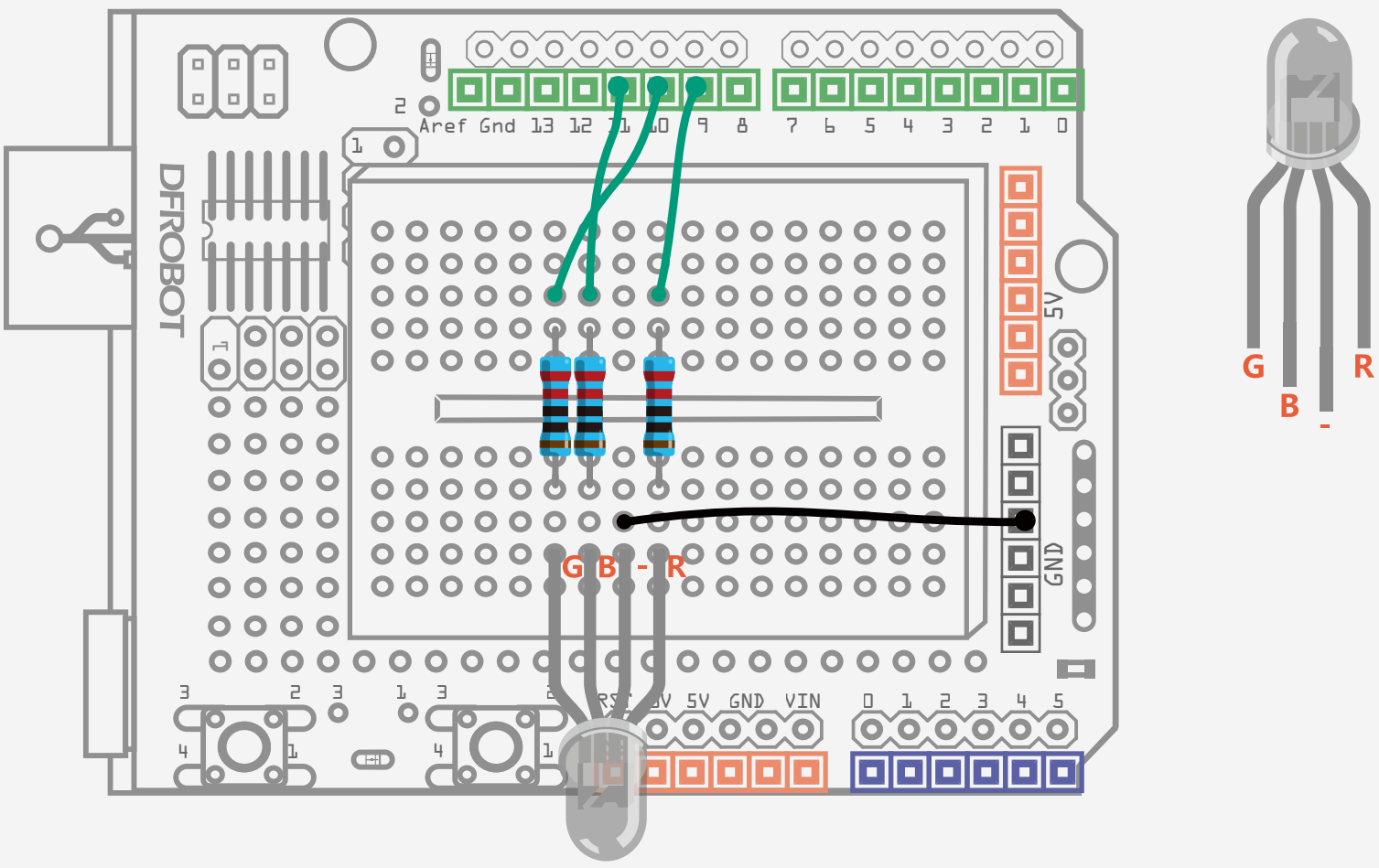


图 5-1 炫彩RGB LED连线图

输入代码

样例代码5-1 :

```
//项目五 - 炫彩RGB灯
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup(){
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    pinMode(bluePin, OUTPUT);
}

void loop(){
     //R:0-255 G:0-255 B:0-255
     colorRGB(random(0,255),random(0,255),random(0,255));
    delay(1000);
}

void colorRGB(int red, int green, int blue){
        analogWrite(redPin,constrain(red,0,255));
        analogWrite(greenPin,constrain(green,0,255));
        analogWrite(bluePin,constrain(blue,0,255));
}
```

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。

代码回顾

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。在我们程序开头部分可以看到定义了三个引脚，并设置为输出模式。

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，只有直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。

函数格式如下：

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。

函数格式如下：

random()函数用于生成一个随机数，min随机数的最小值，max随机数的最大值。random()函数还有其他用法，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。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。

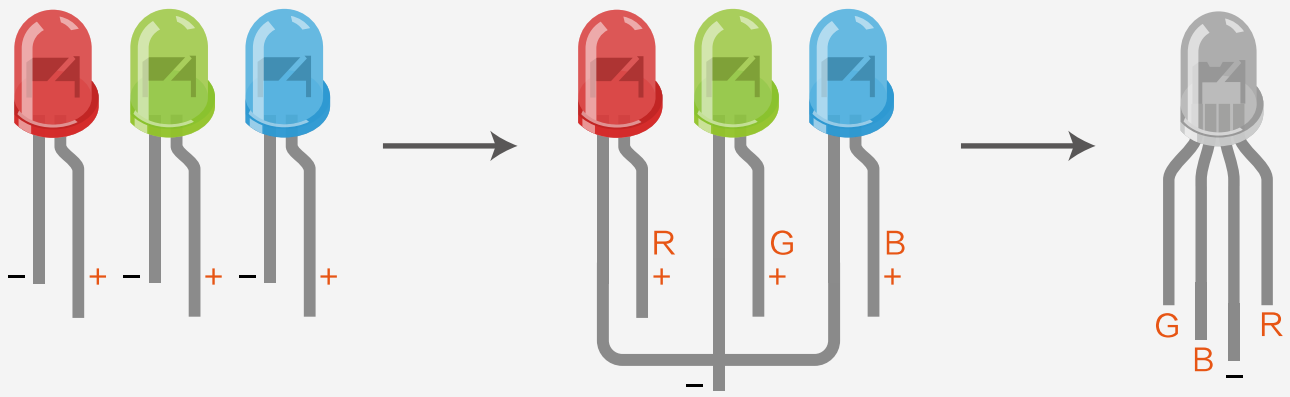


图 5-2 3个LED蜕变为1个RGB的过程

RGB灯如何使用？如何实现变色？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，即analogWrite(value)语句，就能让LED调出任何你想要的颜色。

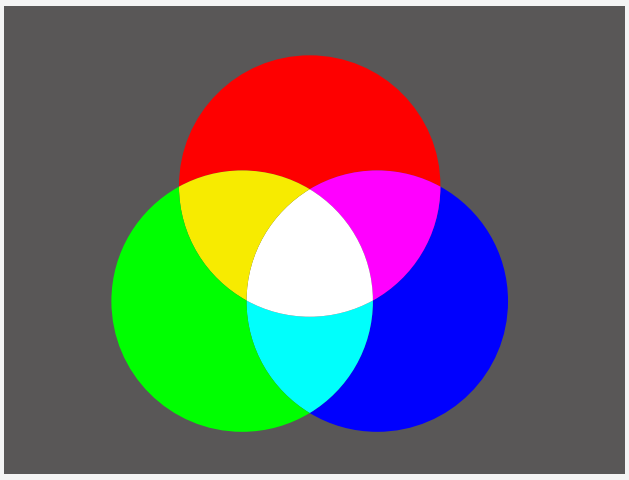


图 5-3 混合R、G、B获得不同的颜色

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！

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 不同LED的PWM值所组合产生的颜色

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。

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。

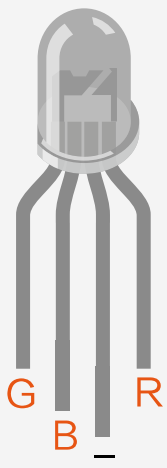


图 5-4 共阴RGB示意图

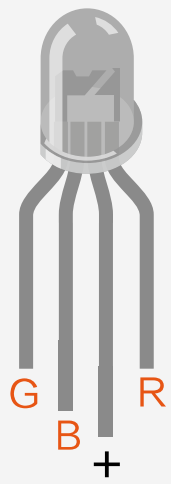


图 5-5 共阳RGB示意图

课后作业

1、基于我们上面的炫彩RGB项目，改变代码能都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。

2、在作业1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。

3、Arduino是个开源的平台，从网上寻找一些别人已经写好了的库，不需要自己从头写，难度也比较大，所以我们只需调用别人写好的库，来达到我们想要的效果就可以了。

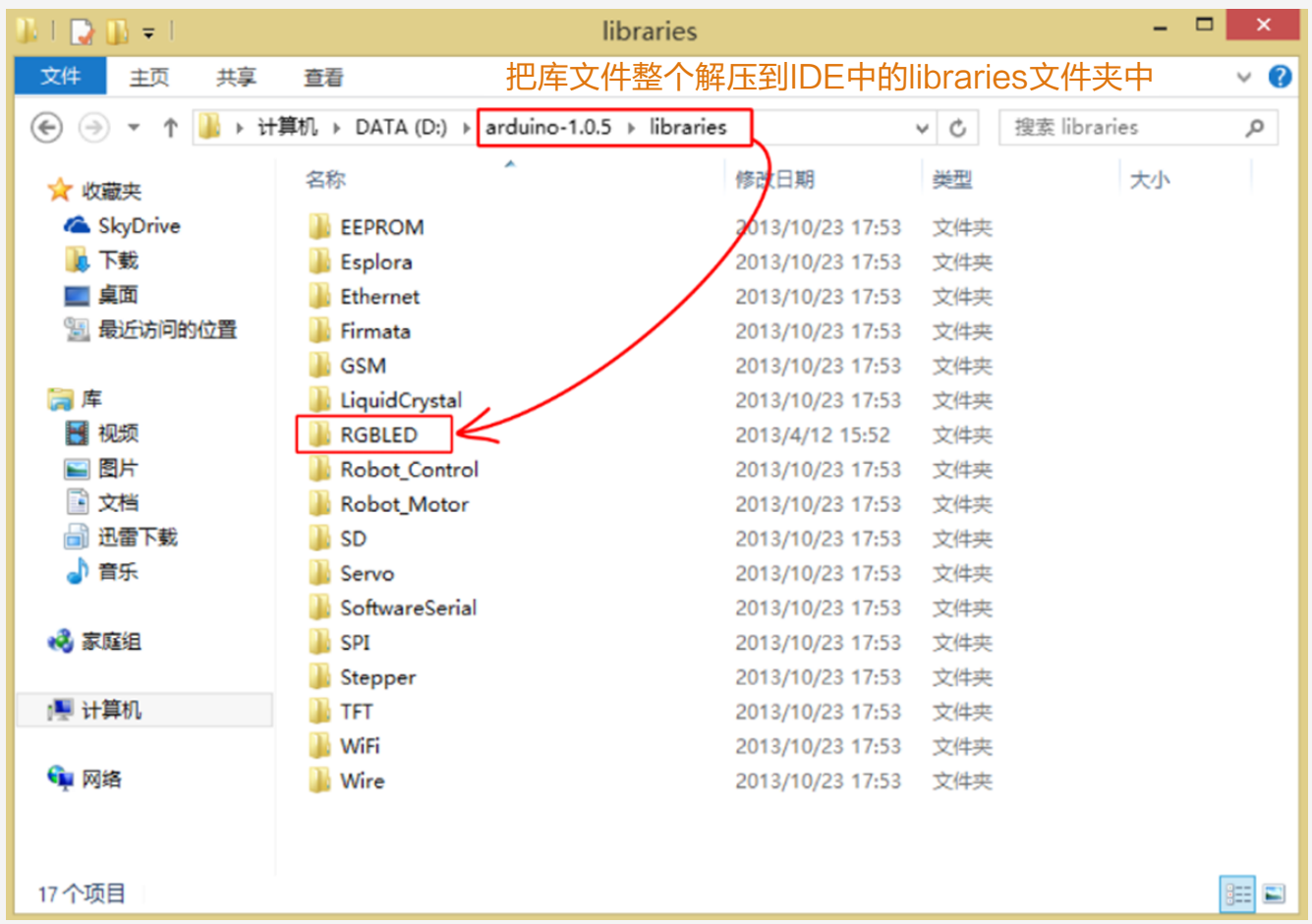
下面就提供一个DFRobot的RGB LED库文件。你可以尝试直接运行样例代码
http://www.dfrobot.com.cn/index.php?route=product/product&filter_name=RGB&product_id=756

4、RGB灯和按钮结合，用三个按钮分别控制R、G、B的颜色。随意变幻出你想要的颜色。
可以参看教程：
http://learn.adafruit.com/adafruit-arduino-lesson-7-make-an-rgb-led-fader?view=all

如何加载库？

先把库文件从网站上下载下来，整个压缩包解压到 Arduino IDE 的libraries文件夹中。

需要注意的是，库文件夹下要直接显示*.cpp和*.h文件，绝对不可以把这些库文件再套到二级以上目录，这样子就会导致IDE无法识别。



接下来直接用 Arduino IDE 运行Example里面的程序就可以了，注意不同代码对应的引脚不同。在代码中改为你连接的引脚就行了。

项目六

报警器

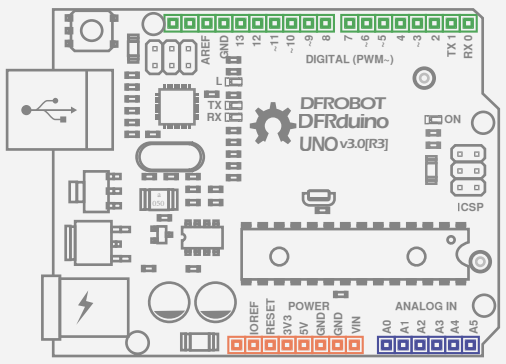
06

DFROBOT®
DRIVE THE FUTURE

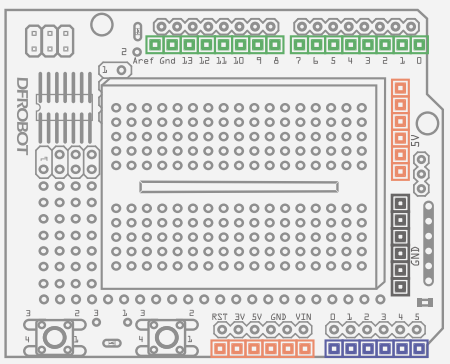
项目六 报警器

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

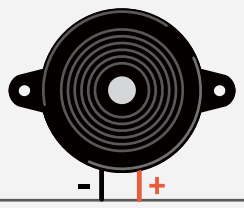


DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x2



Buzzer
蜂鸣器 x1

硬件连接

按下图连接图连接，注意蜂鸣器长脚为正（+），短脚为（-）。短脚接到GND，长脚接到数字口8。

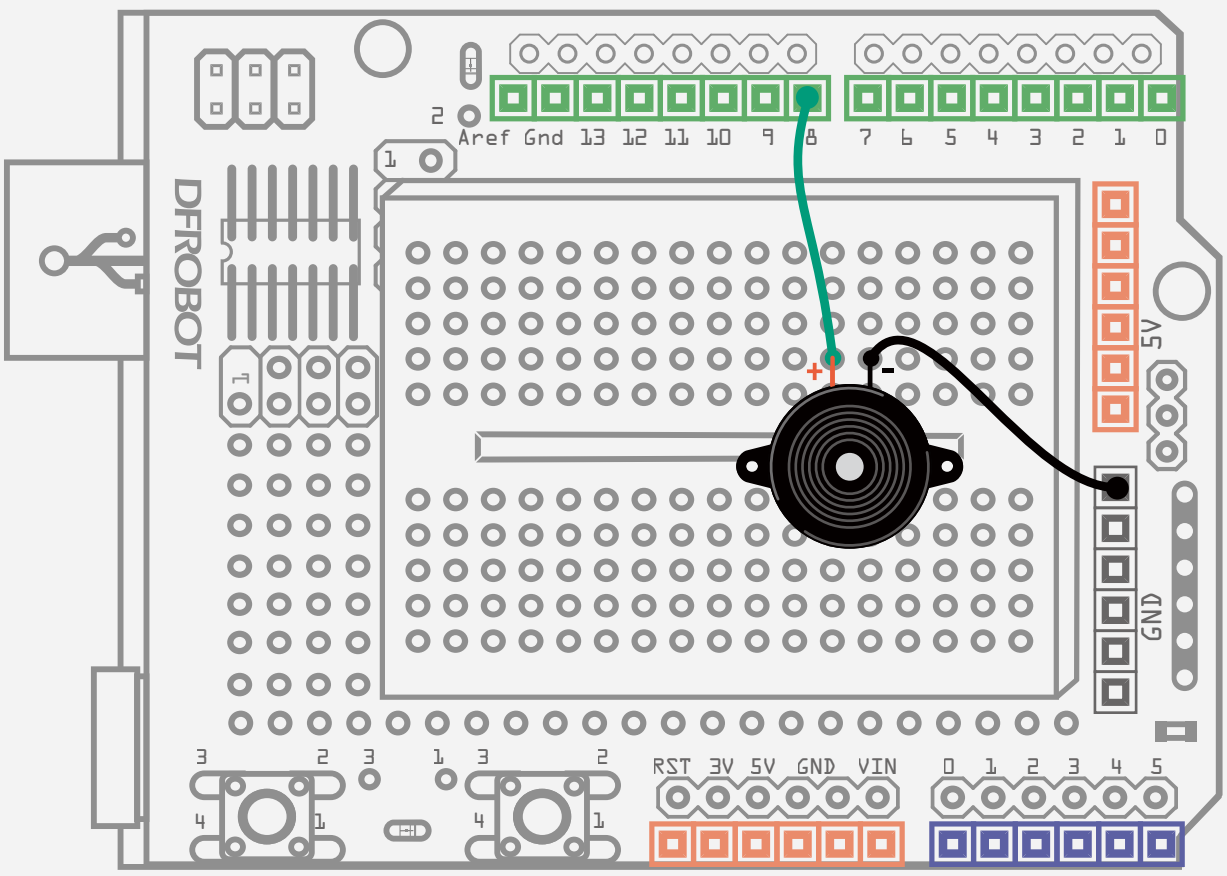


图 6-1 报警器连线图

输入代码

输入下面的样例代码6-1，这段代码引自《beginning-arduino》一书。

样例代码6-1 :

```
//项目六 报警器
float sinVal;
int toneVal;

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*(3.1412/180)));
        //用sin函数值产生声音的频率
        toneVal = 2000+(int(sinVal*1000));
        //给引脚8一个
        tone(8, toneVal);
        delay(2);
    }
}
```

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。

代码回顾

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，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。

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3.1412/180)将角度转为弧度：

```
sinVal=(sin(x*(3.1412/180)));
```

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得过程，也就是下面这句语句就完成了这件事：

```
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。

```
tone(8, toneVal);
```

下面我们来介绍一下tone相关的三个函数

1、tone(pin,frequency)

Pin都是指连接到蜂鸣器的数字引脚，frequency是以Hz为单位的频率值。

2、tone(pin,frequency, duration)

第二个函数，有个duration参数，它是以毫秒为单位，表示声音长度的参数。像第一个函数，如果没有指定duration，声音将一直持续直到输出一个不同频率的声音产生。

3、noTone(pin)

noTone(pin)函数，结束该指定引脚上产生的声音。

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。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。

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。

所以，对于初学者来说有源蜂鸣器会更容易上手一点。在套件中，我们为初学者选用的蜂鸣器类型是电磁式有源蜂鸣器。当然，如果有源蜂鸣器玩的够熟练的话，不妨考虑买一个无源蜂鸣器玩玩，可以演奏出不用的音乐效果。

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、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。

2、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

07

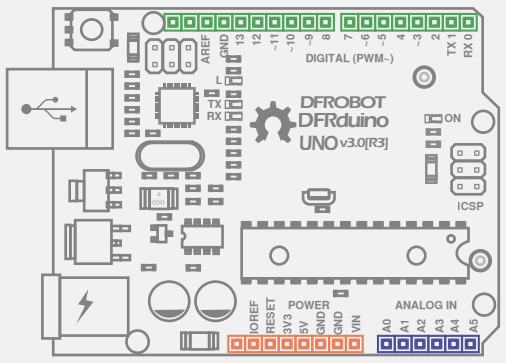
DFROBOT®
DRIVE THE FUTURE

项目七 温度报警器

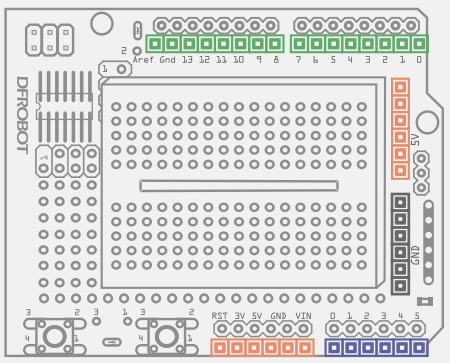
在上一节中，我们认识了一个发声元件——蜂鸣器，也做了一个简单的小报警器。是不是还不过瘾呢？这次我们要做一个更实际的应用——温度报警器。当温度到达我们设定的限定值时，报警器就会响。我们可以用于厨房温度检测报警等等，各种需要检测温度的场合。这个项目中，除了要用到蜂鸣器外，还需要一个LM35温度传感器。

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

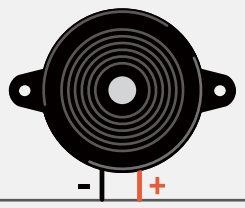


DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x5



Buzzer
蜂鸣器 x1

Tem. Sensor
温度传感器 x1

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。

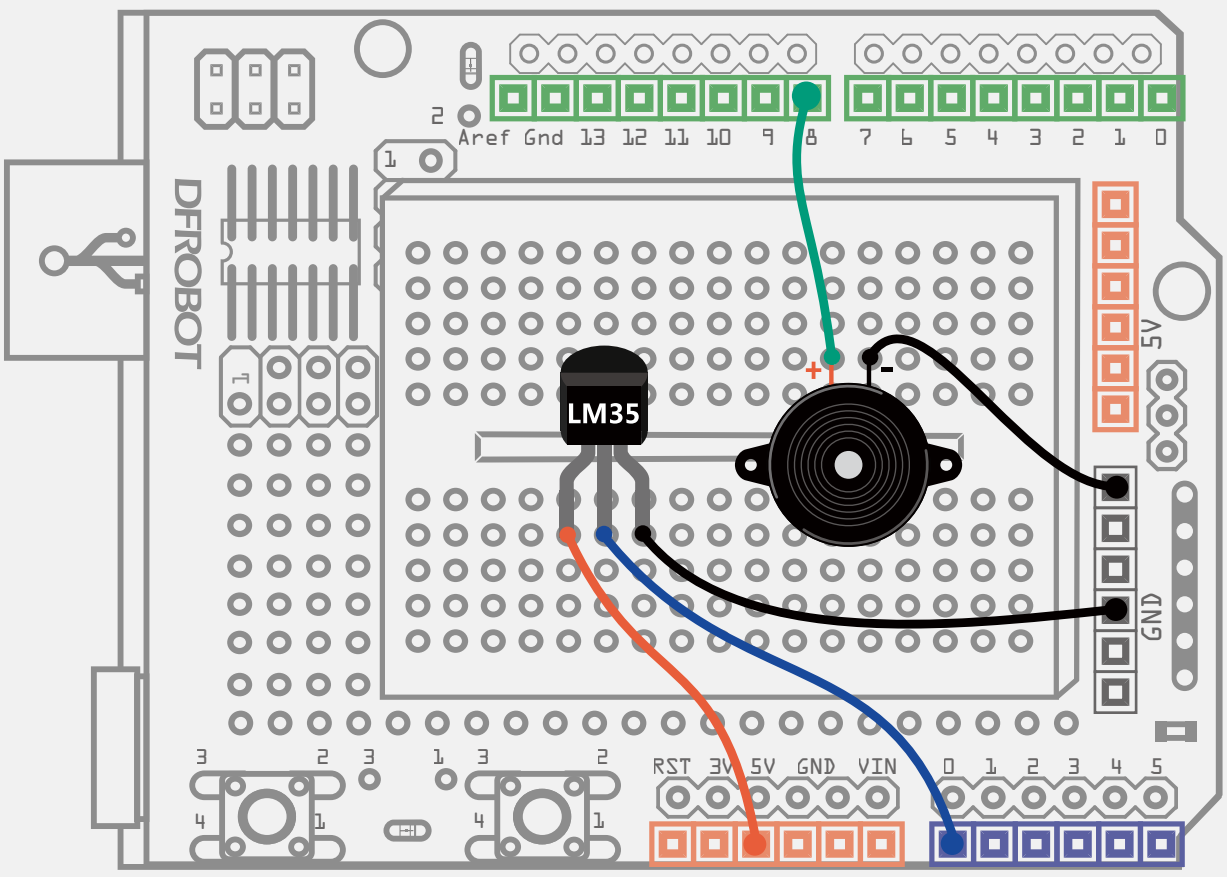


图 7-1 温度报警器连线图

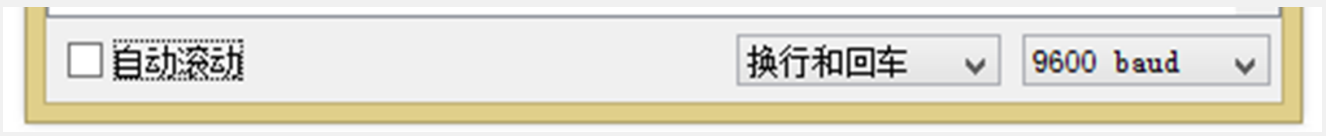
输入代码

样例代码7-1 :

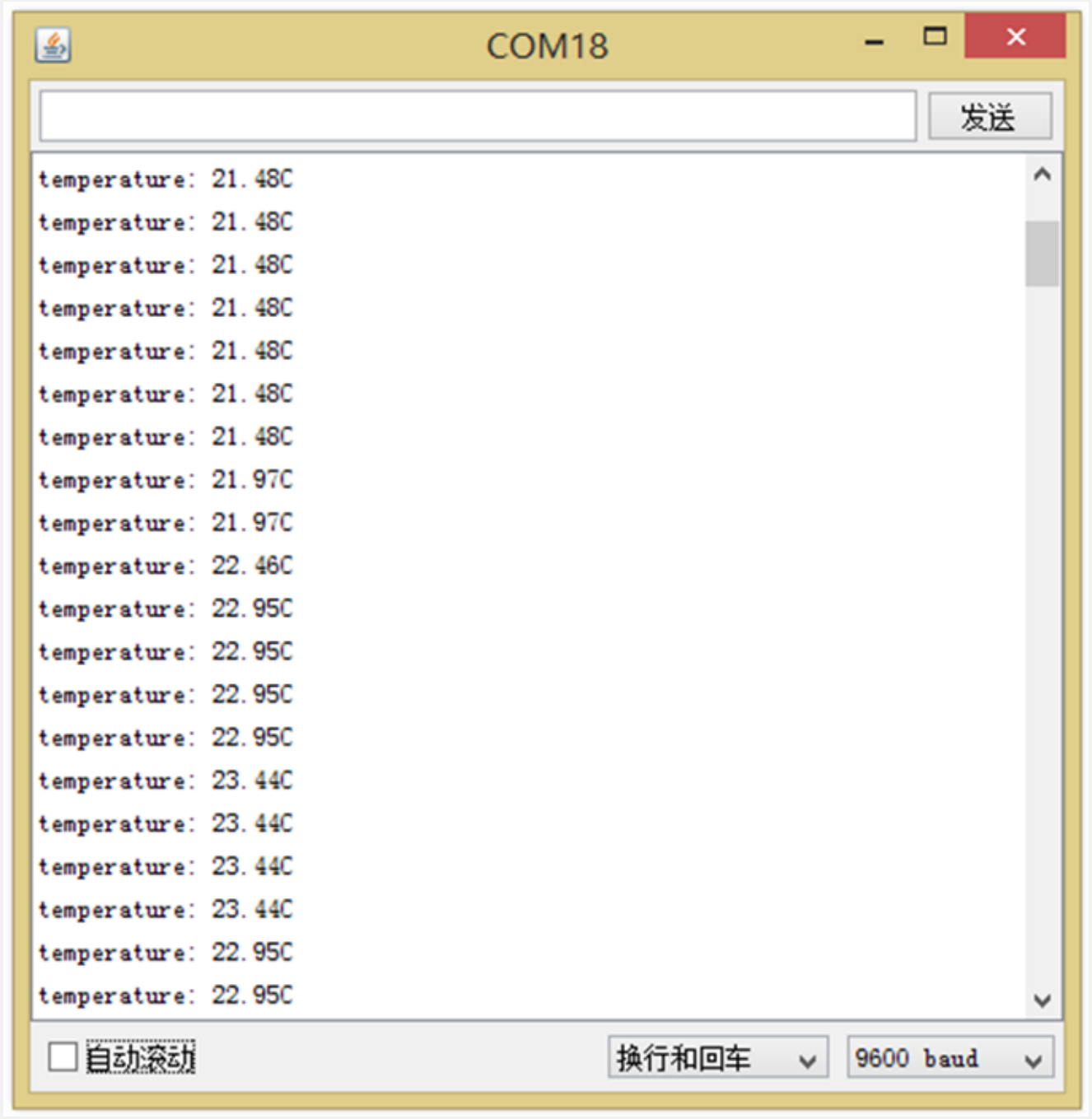
```
//项目七 - 温度报警器
float sinVal;
int toneVal;
unsigned long tepTimer ;

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
          tone(8, toneVal);
          delay(2);
        }
    } else {                             // 如果温度小于27，关闭蜂鸣器
        noTone(8);                       //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("C");           // 串口输出温度单位
      }
}
```

成功下载完程序后，打开Arduino IDE的串口监视器。

设置串口监视器的波特率为9600。



就可以直接从串口中读取温度值，并尝试升高周围环境温度，或者用手直接接触LM35使其升温，串口可以很直观的看到温度有明显的变化。



蜂鸣器工作的条件是，一旦检测到环境温度大于27度，蜂鸣器鸣响，环境温度小于27度，则关闭蜂鸣器。

代码回顾

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。

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);
```

这个函数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。我们这里就只是用于串口监视器。

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);
```

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);
```

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。

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，当下载程序的时候，USB将占用了数字引脚0(RX)和数字引脚1(TX)。此时，在下载程序的过程中，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。

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(val);
Serial.println(val);
```

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);
```

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。

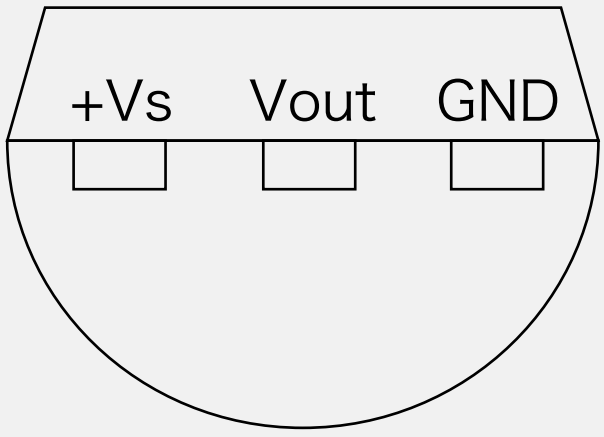


图 7-2 LM35引脚示意图

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？这里提供一个网址，专门用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。
ALLDATASHEET:
http://www.alldatasheet.com/

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

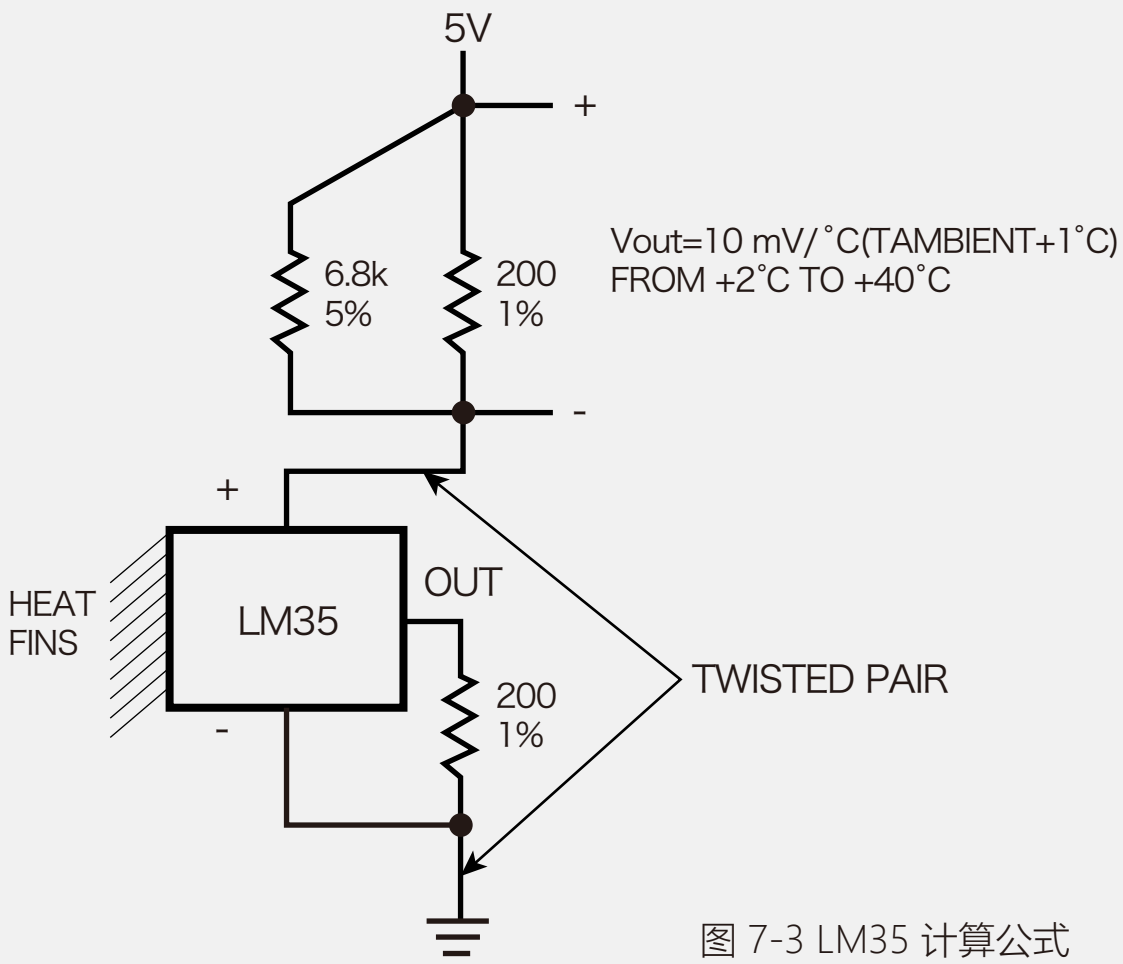


图 7-3 LM35 计算公式

课后作业

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，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

08

DFROBOT®
DRIVE THE FUTURE

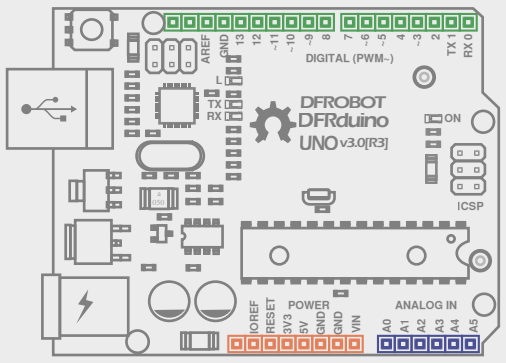
项目八 震动传感器

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。

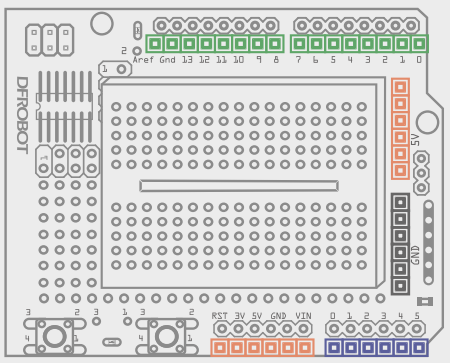
通过这个原理，我们可以做一些小玩具结合起来。最常见的，比如我们看到一些小孩子穿的一闪一闪的小鞋子！走动的过程，就能使内部珠子滚动。

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



x1 **DFduino UNO R3**
（及配套USB数据线）



x1 **Prototype Shield**
原型扩展板+面包板

x5 **Jumper Cables**
M/M
跳线(公公头)

x1 **5MM LED**
5MM LED灯

x2 **Resistor 220R**
220欧电阻

x1 **Tilt Switch Sensor**
倾斜开关

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？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。

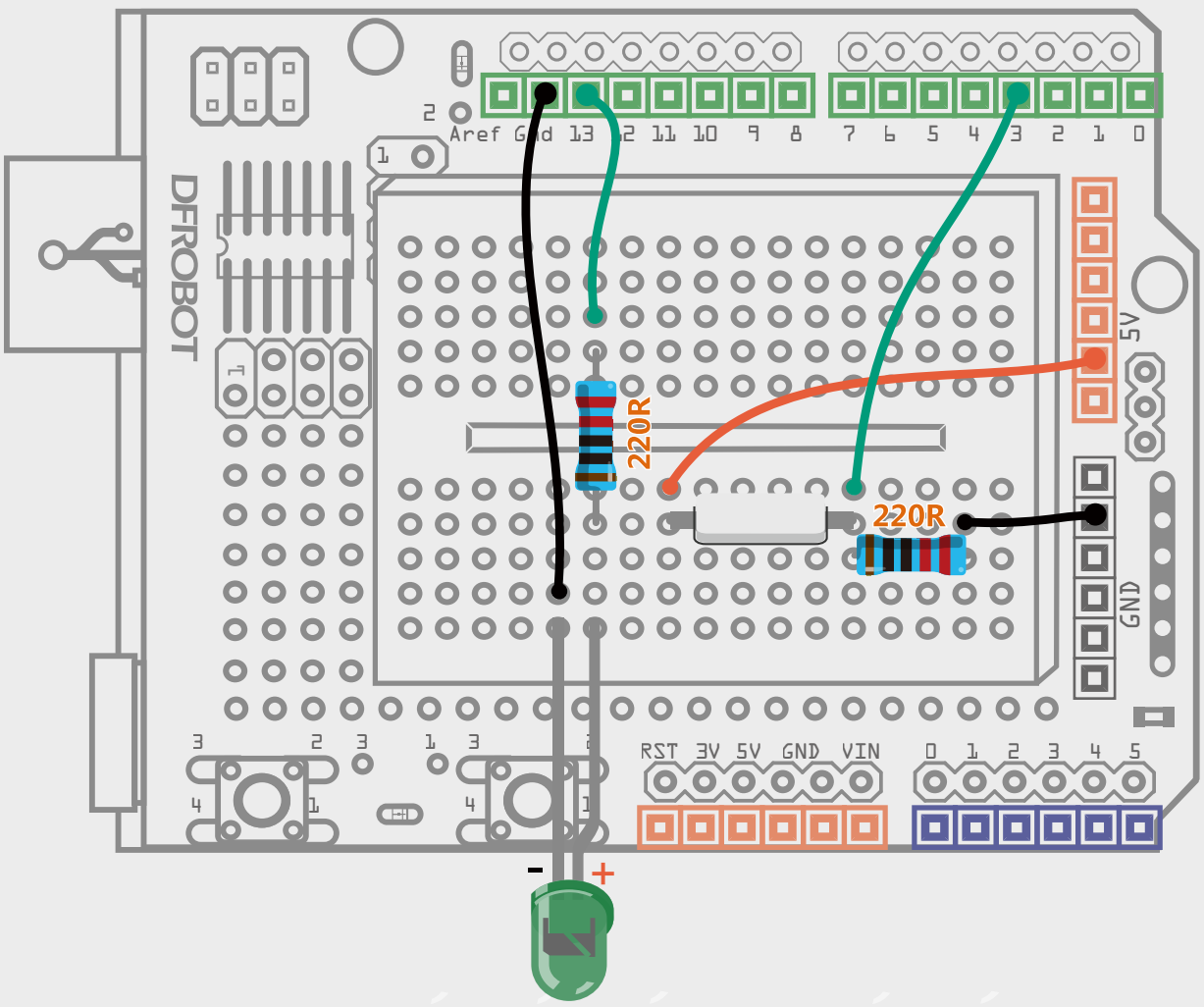


图 8-1 震动传感器连线图

输入代码

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，即数字引脚3
unsigned char state = 0;

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。

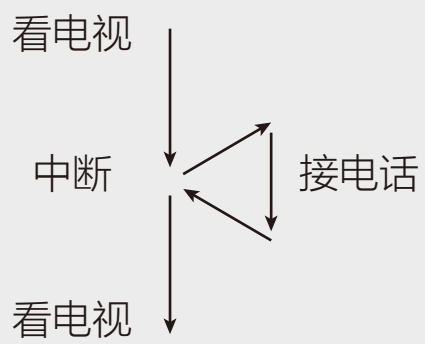
代码回顾

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，连锁反应的，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有恢复到关闭的状态。

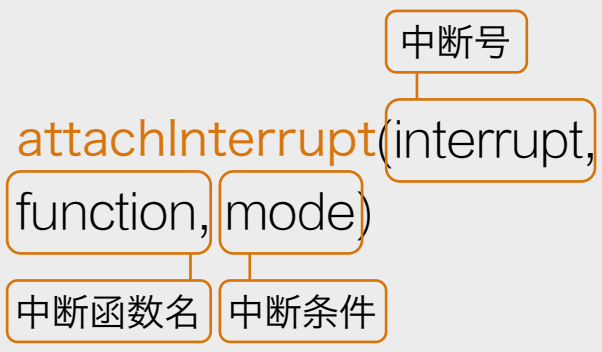
重复的知识点不再赘述，重点说下中断函数attachInterrupt()。

什么是中断？打个比方吧，比如你在家看电视，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。大多数板子都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（http://arduino.cc/en/Reference/AttachInterrupt）。

attachInterrupt()需要三个传递参数：



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：

1、我们在写中断函数的时候，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。

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

3、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：

1、LOW 当引脚为低电平时，触发中断。

2、CHANGE　当引脚电平发生改变时，触发中断。

3、RISING 当引脚由低电平变为高电平时，触发中断。

4、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);
```

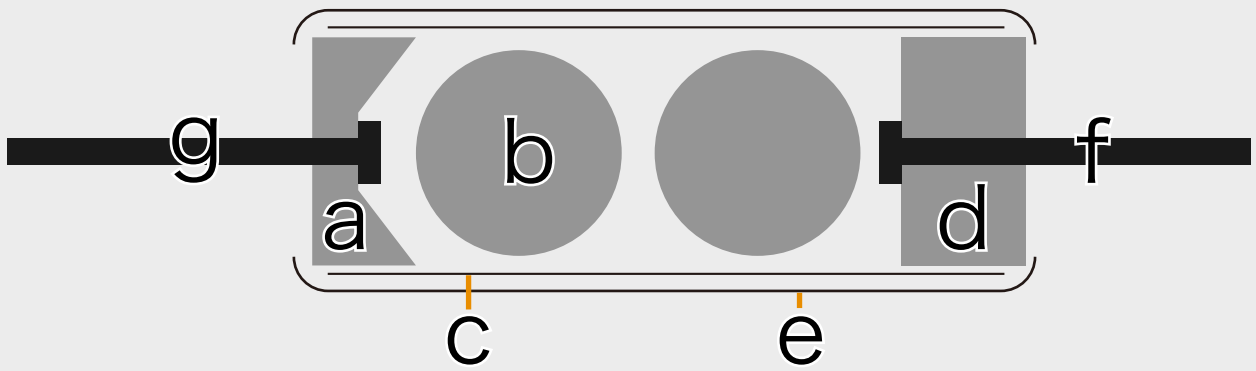
对应上面说明看。1，指中断号1。所以滚珠开关接到数字引脚3。blink是我们下面要调用的中断函数。RISING，指引脚3在由低变为高的一瞬间，中断触发。

为什么要选RISING呢？由于硬件我们还没提到，我们就把滚珠开关想象成按键。在按键没按下时，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

09

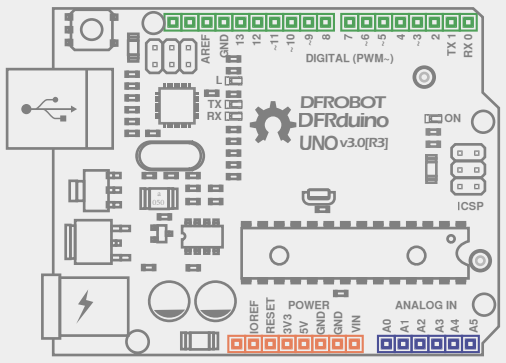
DFROBOT®
DRIVE THE FUTURE

项目九 感光灯

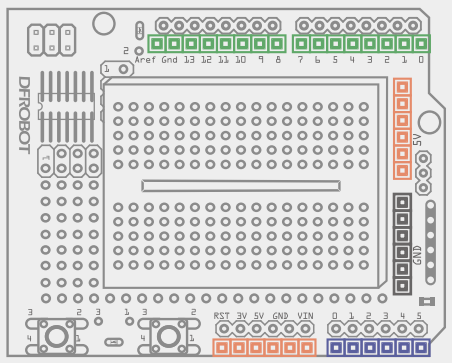
这个项目中将介绍一个新元件——光敏电阻。从名字可以看出，这个器件是依赖光作用的。在黑暗的环境中，光敏电阻具有非常高阻值的电阻。光线越强，电阻值反而越低。通过读取这个电阻值，就可以检查光线的亮暗了。我们这里选用的是光敏二极管，光敏二极管其实就是光敏电阻中的一种，只是它还具有正负极性。

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



DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x5

5MM LED
5MM LED灯 x1

Ambient Light
Sensor
光敏电阻 x1

Resistor 220R
220欧电阻 x1

Resistor 10K
10K电阻 x1

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。

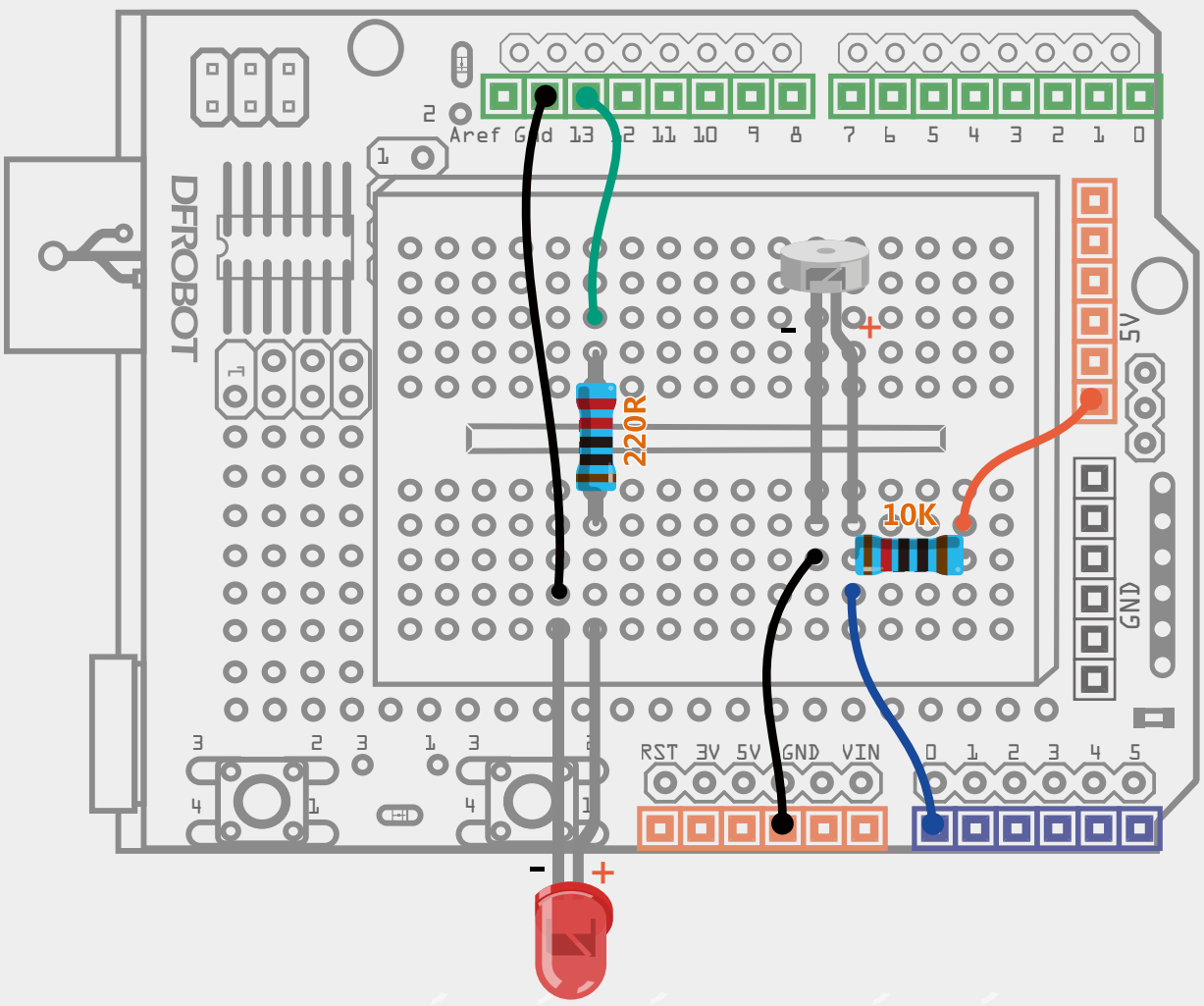


图 9-1 光感灯的接线图

输入代码

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。

代码回顾

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）

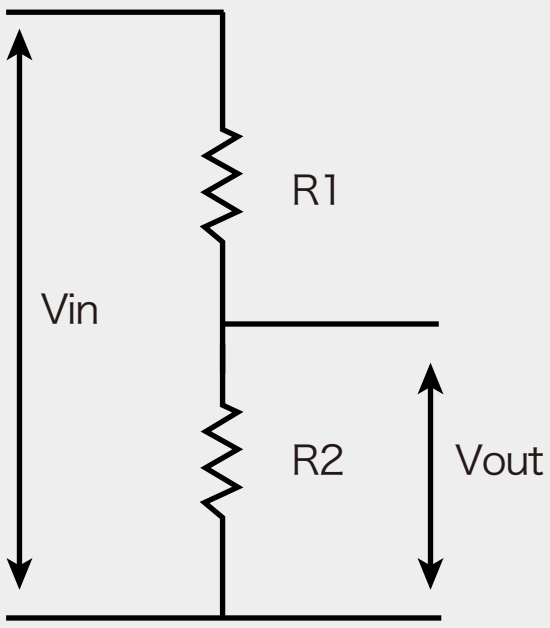


图 9-2 分压电路图

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电压公式如右图所示。（图9-3）

$$Vout = \frac{R2}{R1+R2} \times Vin$$

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

10

DFROBOT®
DRIVE THE FUTURE

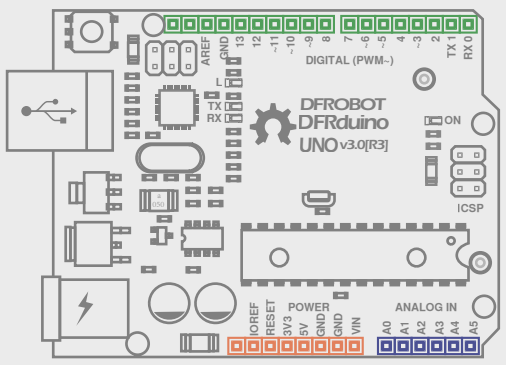
项目十 舵机初动

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。

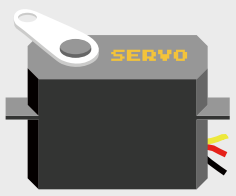
先从简单入手，套件这个9G小舵机是180°的，我们就让它在0~180°之间来回转动。

所需元件



DFduino UNO R3
（及配套USB数据线）
x1

Jumper Cables
M/M
跳线(公公头)
x3



Servo
舵机
x1

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。

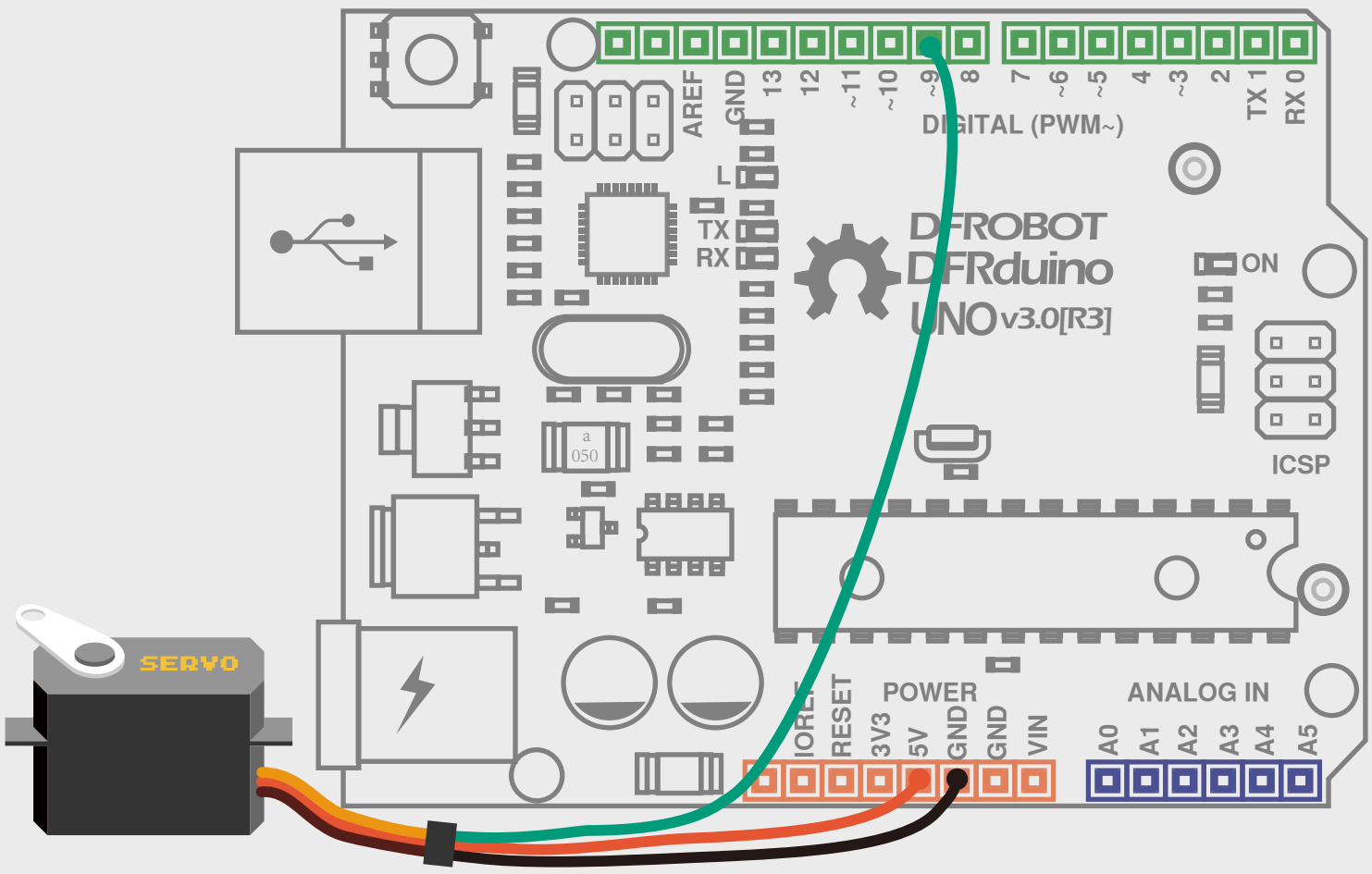


图 10-1 舵机连线图

输入代码

样例代码10-1

```
//项目十 - 舵机
#include <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.write(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.write(pos);             //写角度到舵机
          delay(15);                      //延时15ms让舵机转到指定位置
  }
}
```

下载代码，下载成功后我们可以看到舵机0~180°来回转动。

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。

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？

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;
```

变量pos就不说了，用来存放角度值。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);
```

pin：数字引脚

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write(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.write(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Servo库中还有哪些好用的函数的话，可以参看下面的网址，里面会有相关介绍的。

Servo库：
http://arduino.cc/en/reference/servo
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

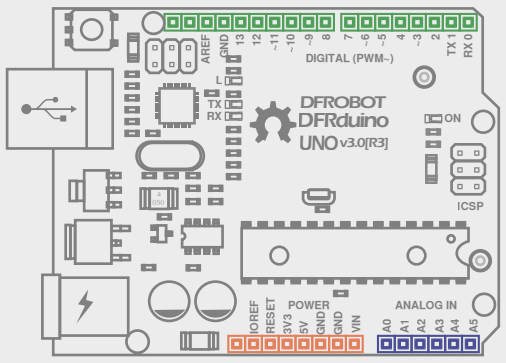
11

DFROBOT®
DRIVE THE FUTURE

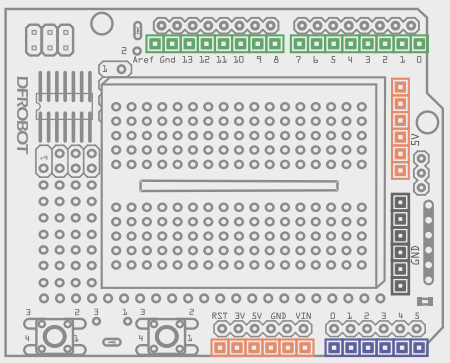
项目十一 可控舵机

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，可以有很多玩儿法。再给舵机加个外壳，让它更具生命力。

所需元件



x1 **DFduino UNO R3**
（及配套USB数据线）

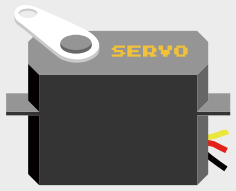


x1 **Prototype Shield**
原型扩展板+面包板

Jumper Cables
M/M
跳线(公公头) x3

Jumper Cables
F/M
跳线(公母头) x3

10K
Potentiometer x1
10K电位器



Servo x1
舵机

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。

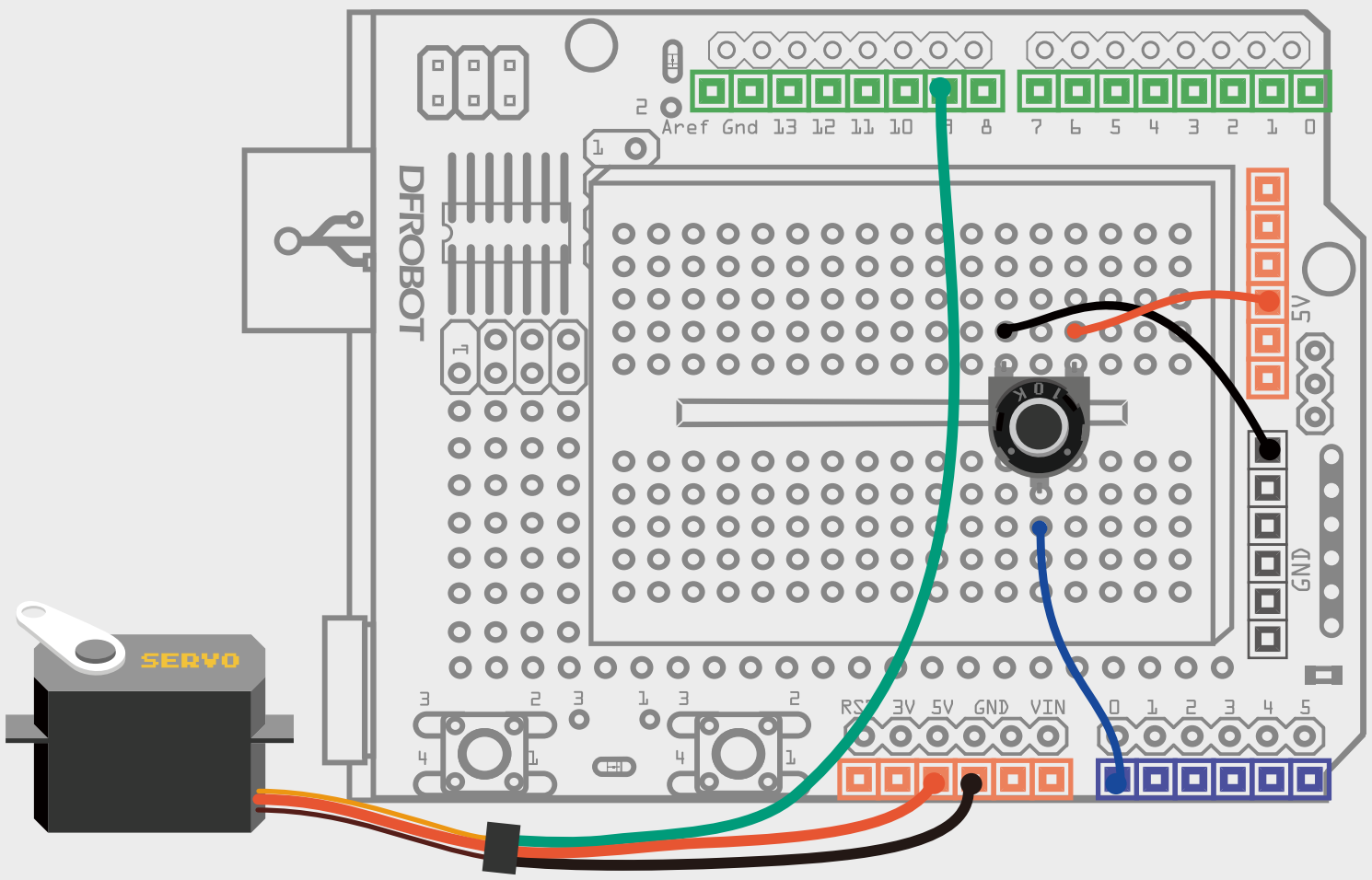


图 11-1可控舵机连线图

输入代码

样例代码11-1

```
//项目十一  可控舵机
#include <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, 179);  //通过map函数进行数值转换
  myservo.write(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。

代码回顾

代码的开始部分还是需要调用<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。

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, 179);
```

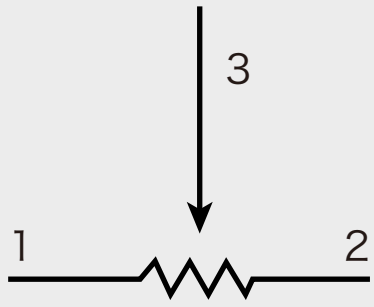
所以，回到代码中，我们是想将模拟口读到的0~1023的值，转换为舵机的0~180° 。

硬件回顾

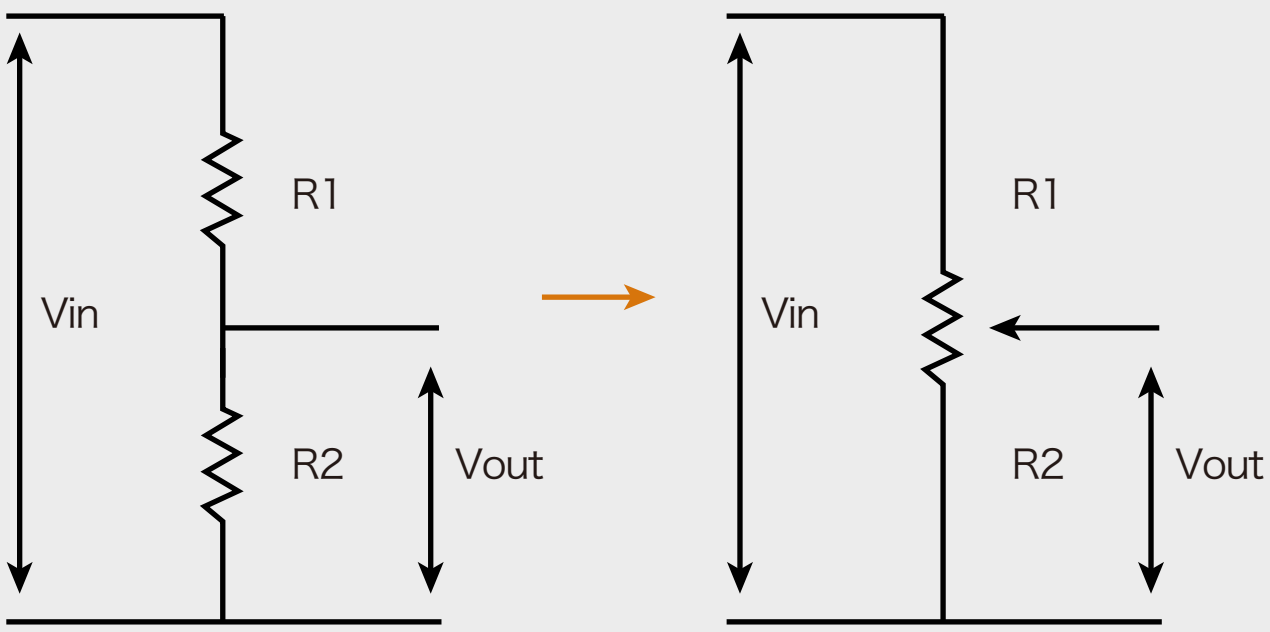
电位器

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右图，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

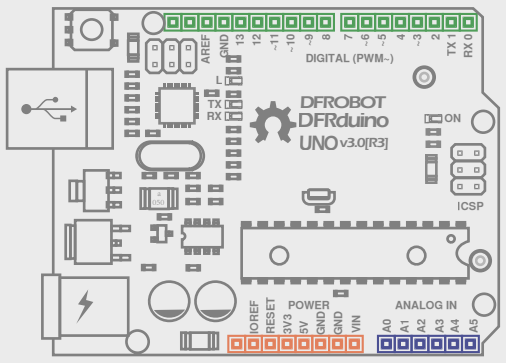
12

DFROBOT®
DRIVE THE FUTURE

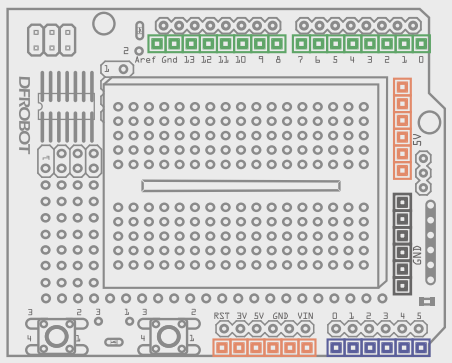
项目十二 彩灯调光台

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



DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x13

Resistor 220R
220欧电阻 x3

10K
Potentiometer
10K电位器 x3

5mm RGB LED
5mm RGB LED灯 x1

硬件连接

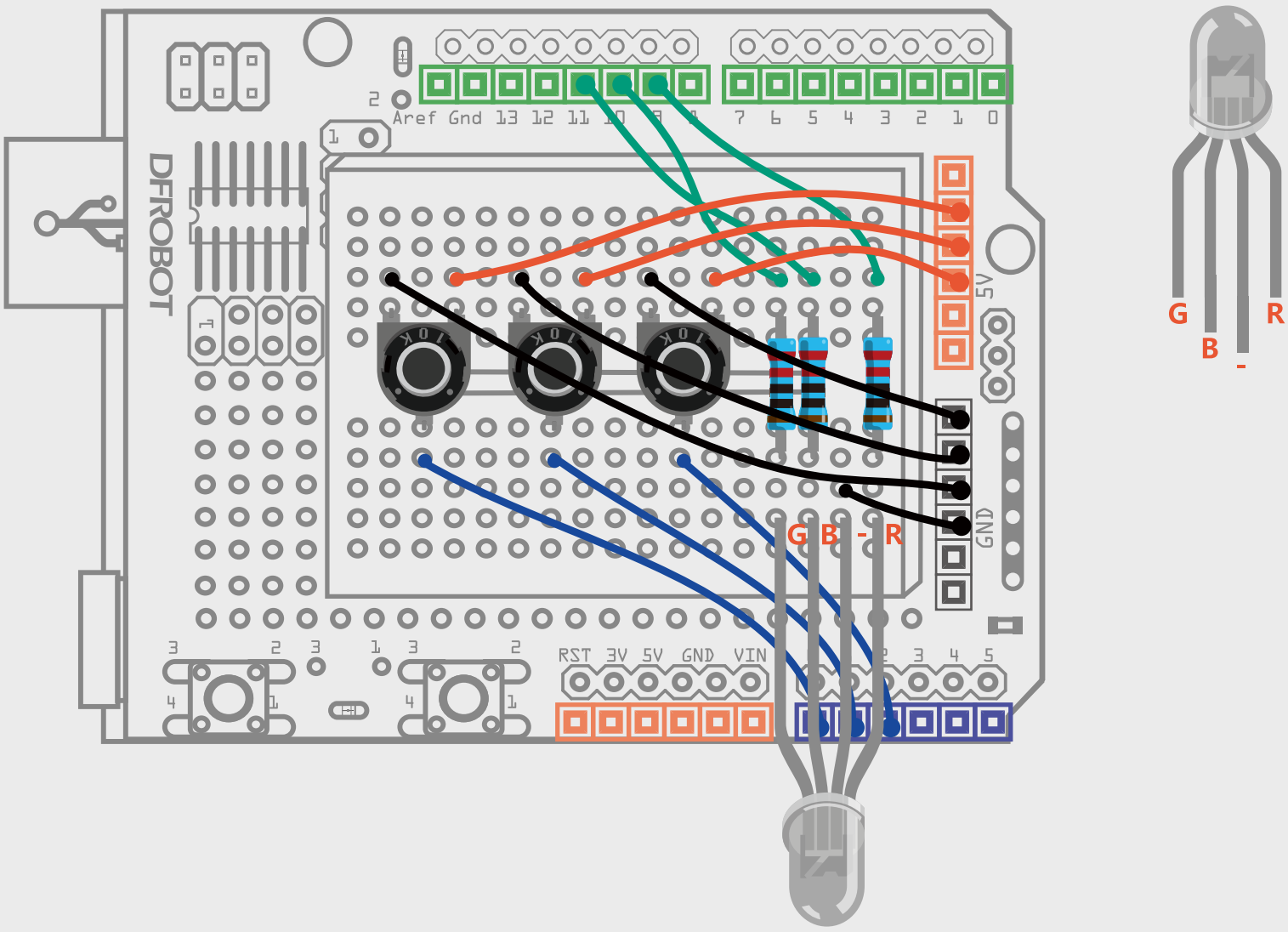


图 12-1 彩灯调光台连线图

输入代码

样例代码12-1

```
//项目十二 - 互动彩灯
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 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
}

//该函数用于显示颜色
void colorRGB(int red, int green, int blue){
  analogWrite(redPin,constrain(red,0,255));
  analogWrite(greenPin,constrain(green,0,255));
  analogWrite(bluePin,constrain(blue,0,255));
}
```

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。

项目十三

自制风扇

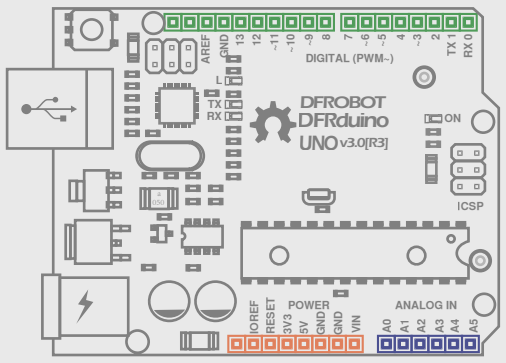
13

DFROBOT®
DRIVE THE FUTURE

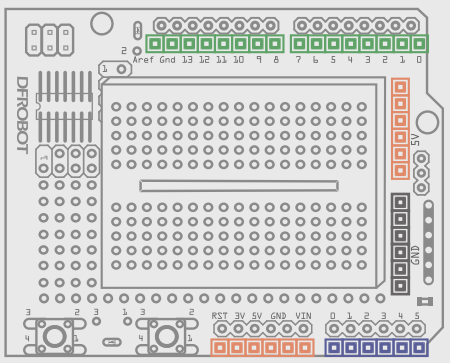
项目十三 自制风扇

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



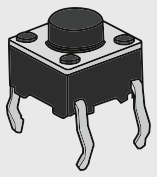
DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables M/M
跳线(公公头) x9

5MM LED
5MM LED灯 x1

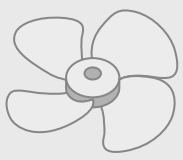


Pushbutton
按键开关 x1

Resistor 220R
220欧电阻 x2

Relay
小型继电器 x1

130 Motor
130小马达 x1



Fan
透明风扇叶 x1

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。

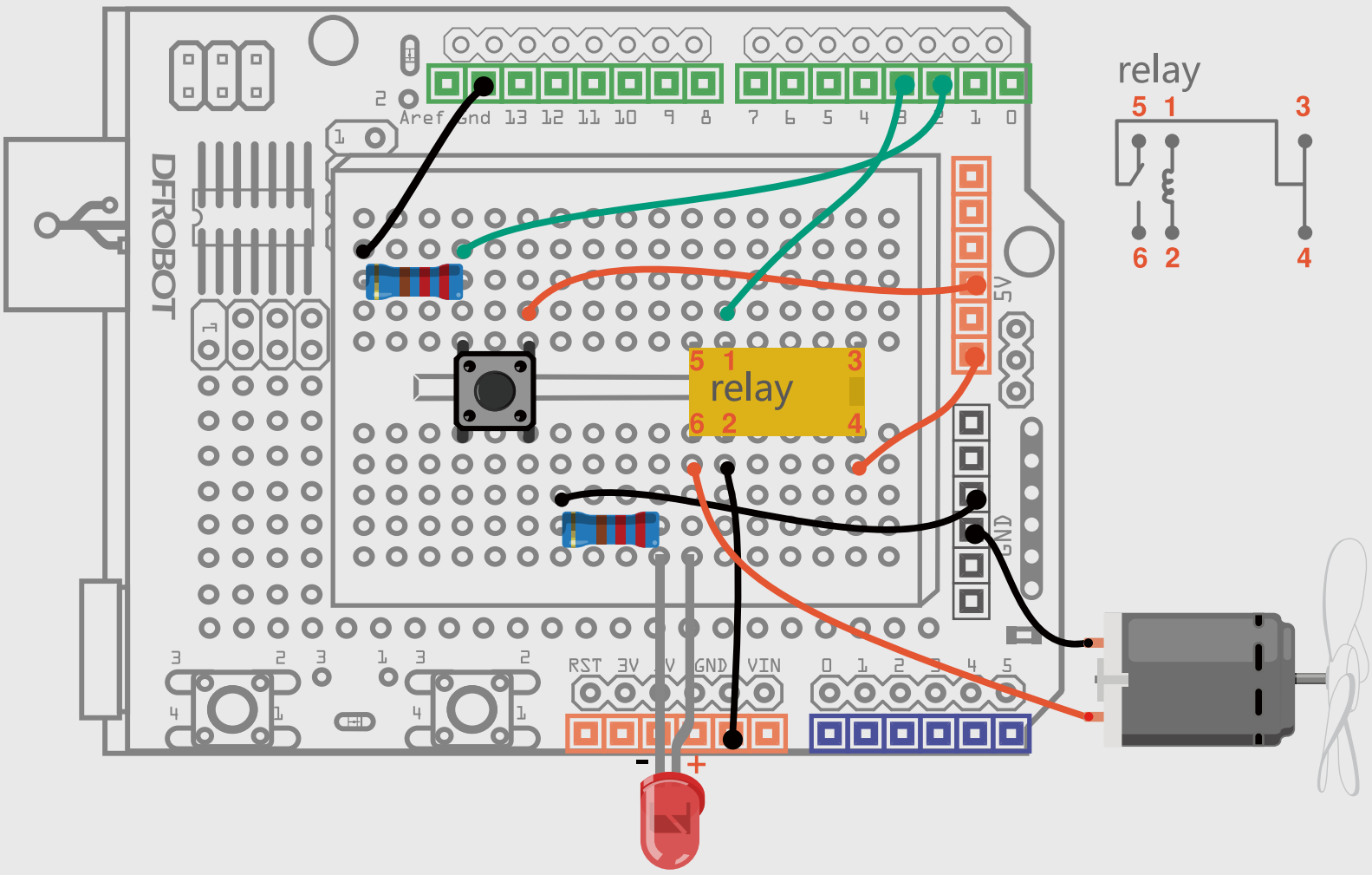


图 13-1 自制风扇的接线图

输入代码

样例代码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。

代码回顾

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) {
      ……
    }
  }
```

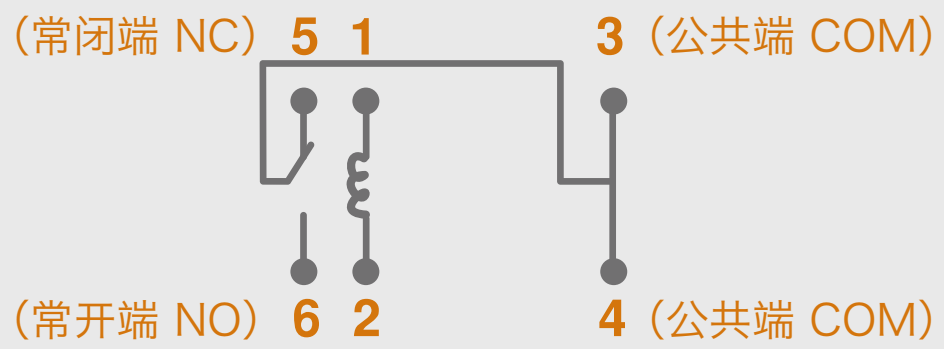
reading有变化之后，不是立马就采取相应的行动，而是先“按兵不动”，先看看这个信号是不是“错误信号”，所以再等待一阵，（也就是通过millis来实现这个等待过程的），发现确实是前方发过来的正确信号，然后执行相关动作。

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。

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

14

DFROBOT®
DRIVE THE FUTURE

项目十四 红外遥控灯

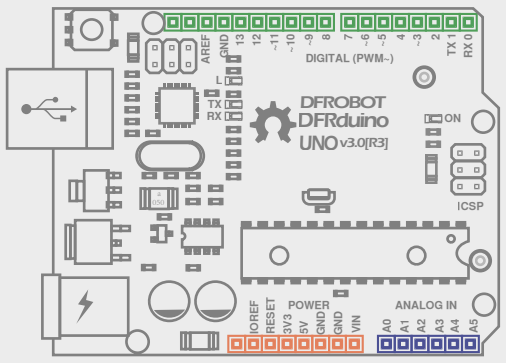
这节我们会接触一个新的元件——红外接收管。所谓红外接收管，也就是接收红外光的电子器件。红外接收管，看着离我们很遥远的感觉！其实不然，它就在我们身边。比如我们电视机，空调这些家电，其实它们都需要用到红外接收管。我们都知道遥控器发射出来的都是红外光，电视机上势必要有红外接收管，才能接收到遥控器发过来的红外信号。

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。

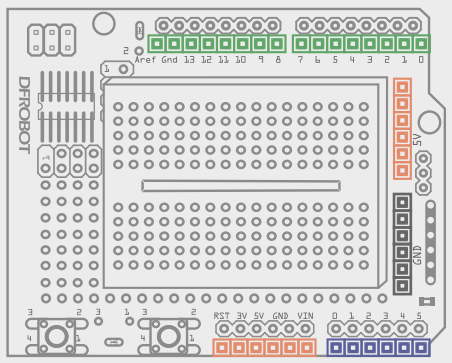
在开始遥控灯之前，我们先来个预热实验，通过串口来了解下如何使用红外接收管和遥控器。

预热实验：

所需元件

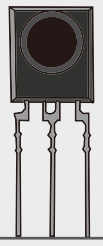


DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x3



IR Receiver
红外接收管 x1

IR Remote
controller
迷你遥控器 x1

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11。

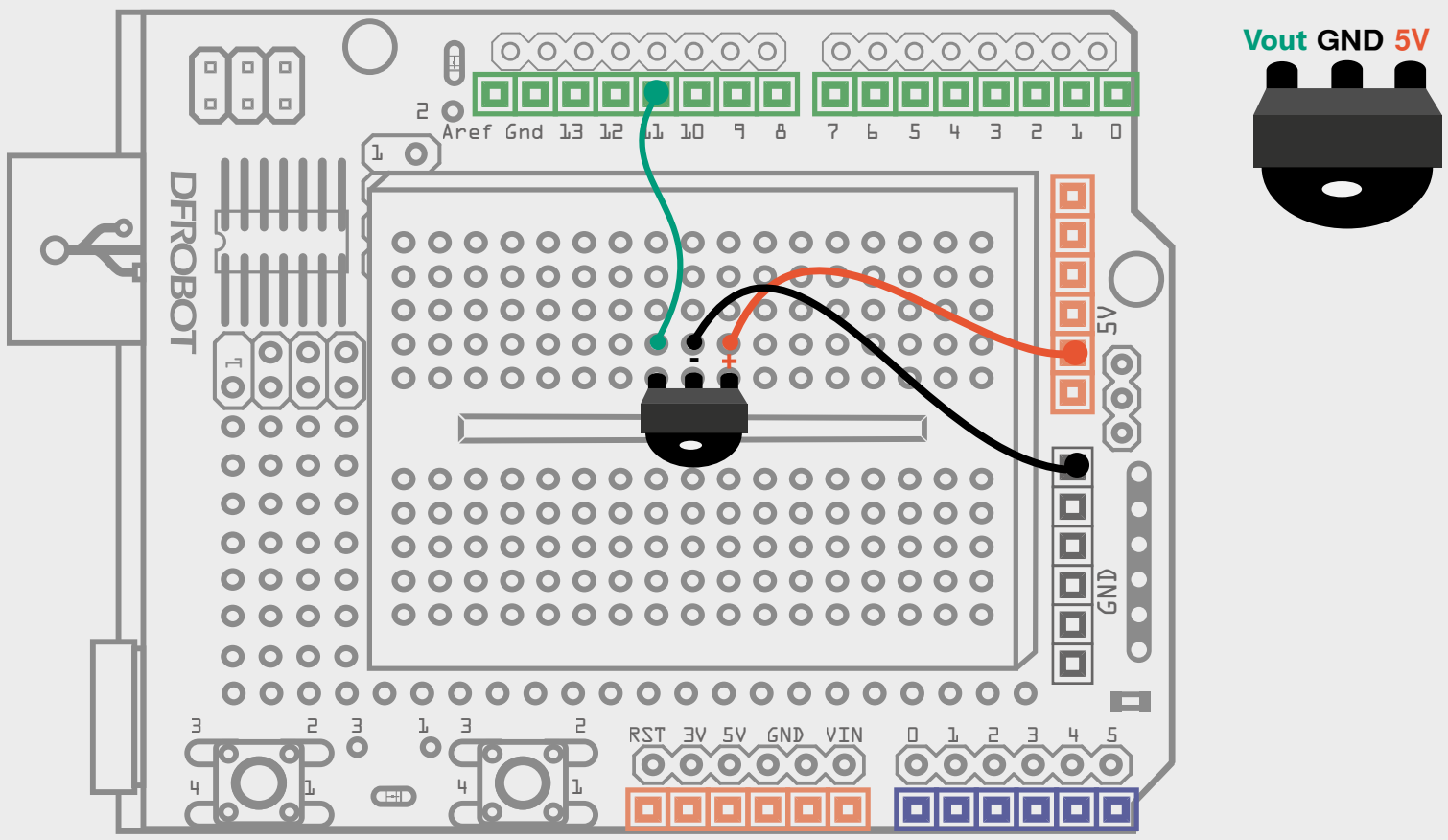


图14-1 红外接收管连线图

输入代码

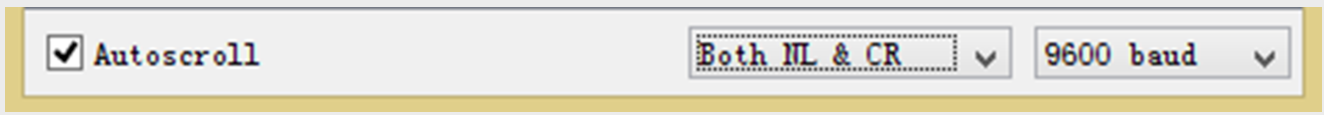
这段代码，你可以不用自己手动输入，我们提供现成的IRremote库，在我们的教程代码文件夹中的_14_1中，把整个库的压缩包解压到Arduino IDE安装位置Arduino 1.0.5/ libraries文件夹中，直接运行Example中的IRrecvDemo代码即可。如果还是不是很明白如何加载库，可以回看一下项目五课后作业部分，对如何加库做了详细说明。

这段代码来自IRremote库中examples中的 IRrecvDemo

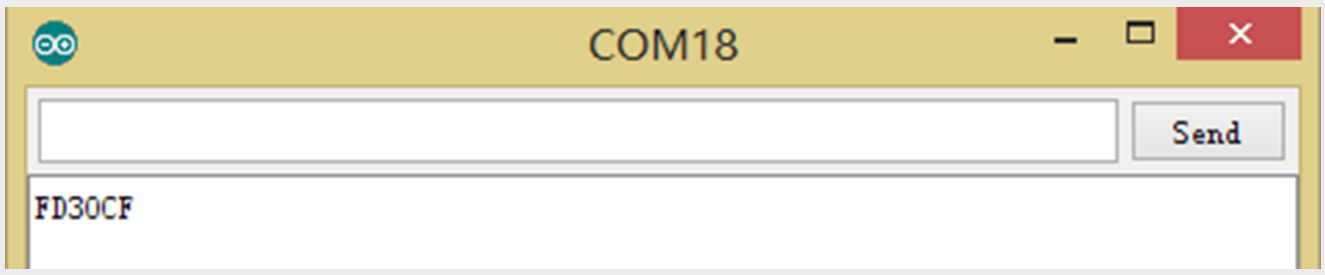
样例代码14-1

```
//项目十四 - 红外接收管
#include <IRremote.h>          //调用IRremote.h库
int RECV_PIN = 11;             //定义RECV_PIN变量为11
IRrecv irrecv(RECV_PIN);       //设置RECV_PIN（也就是11引脚）为红外接收端
decode_results results;        //定义results变量为红外结果存放位置

void setup(){
  Serial.begin(9600);          //串口波特率设为9600
  irrecv.enableIRIn();         //启动红外解码
}

void loop() {
    //是否接收到解码数据,把接收到的数据存储在变量results中
    if (irrecv.decode(&results)) {
        //接收到的数据以16进制的方式在串口输出
        Serial.println(results.value, HEX);
        irrecv.resume();  // 继续等待接收下一组信号
  }
}
```

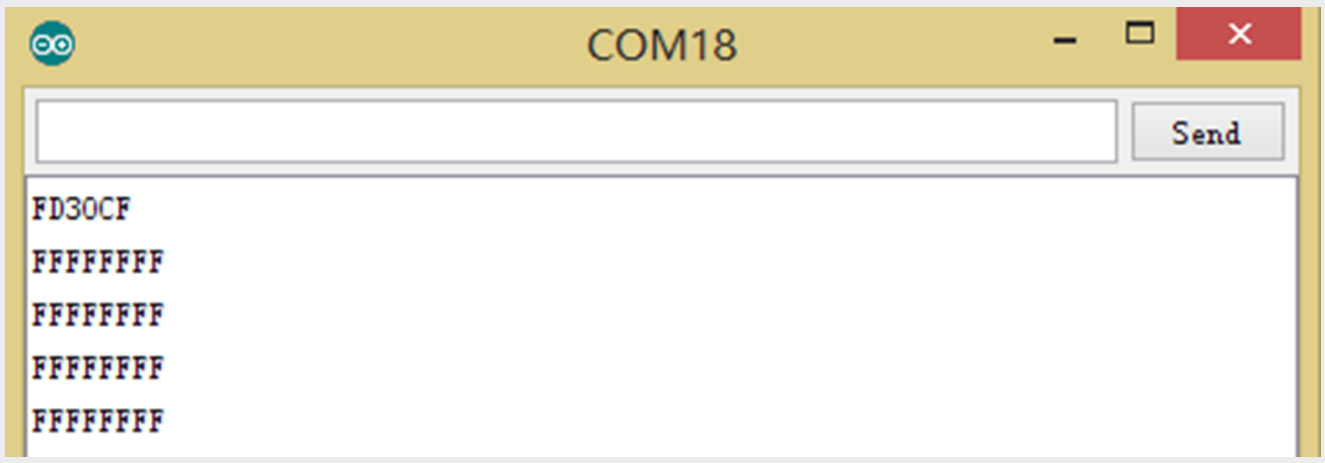
下载完成后，打开Arduino IDE的串口监视器（Serial Monitor），设置波特率baud为9600，与代码中Serial.begin(9600)相匹配。



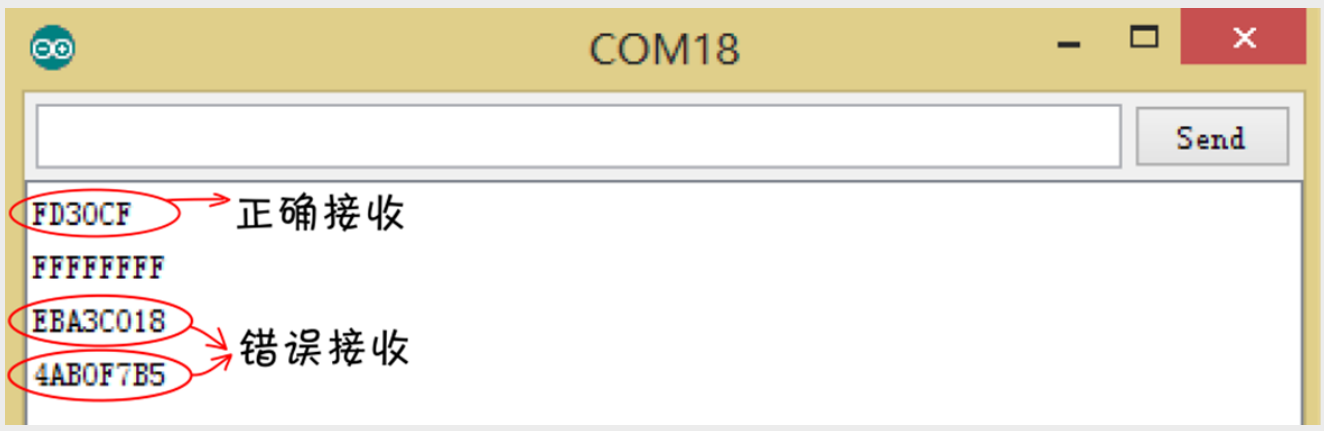
设置完后，用Mini遥控器的按钮对着红外接收管的方向，任意按个按钮，我们都能在串口监视器上看到相对应的代码。如下图所示，按数字“0”，接收到对应16进制的代码是FD30CF。每个按钮都有一个特定的16进制的代码。



如果按住常按一个键不放就是出现“FFFFFFFF”。



在串口中，正确接收的话，应该收到以FD-开头的六位数。如果遥控器没有对准红外接收管的话，可能会接收到错误的代码。如我们下图所示：

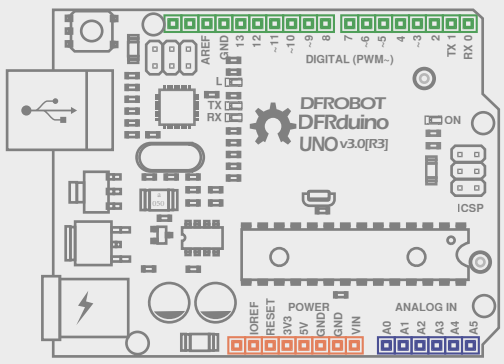


上面这段代码我们没有像以前一样一步一步做详细说明，原因就是由于红外解码较为复杂，所幸的是，高手把这些难的工作已经做好了，提供给我们这个IRremote库，我们只需要会用就可以了，先不需要弄明白函数内部如何工作的。要用的时候，把代码原样搬过来就好了。照猫画虎，先用起来再说~

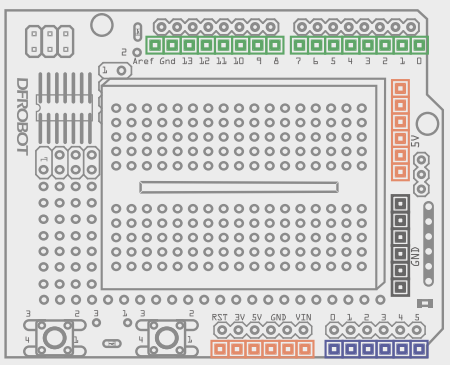
预热完之后，我们言归正传，开始制作遥控灯。

红外遥控灯

所需元件

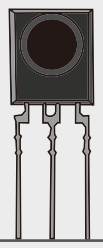


DFduino UNO R3
（及配套USB数据线） x1



Prototype Shield
原型扩展板+面包板 x1

Jumper Cables
M/M
跳线(公公头) x3



IR Receiver
红外接收管 x1

IR Remote
controller
迷你遥控器 x1

5MM LED
5MM LED灯 x1

Resistor 220R
220欧电阻 x1

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11。

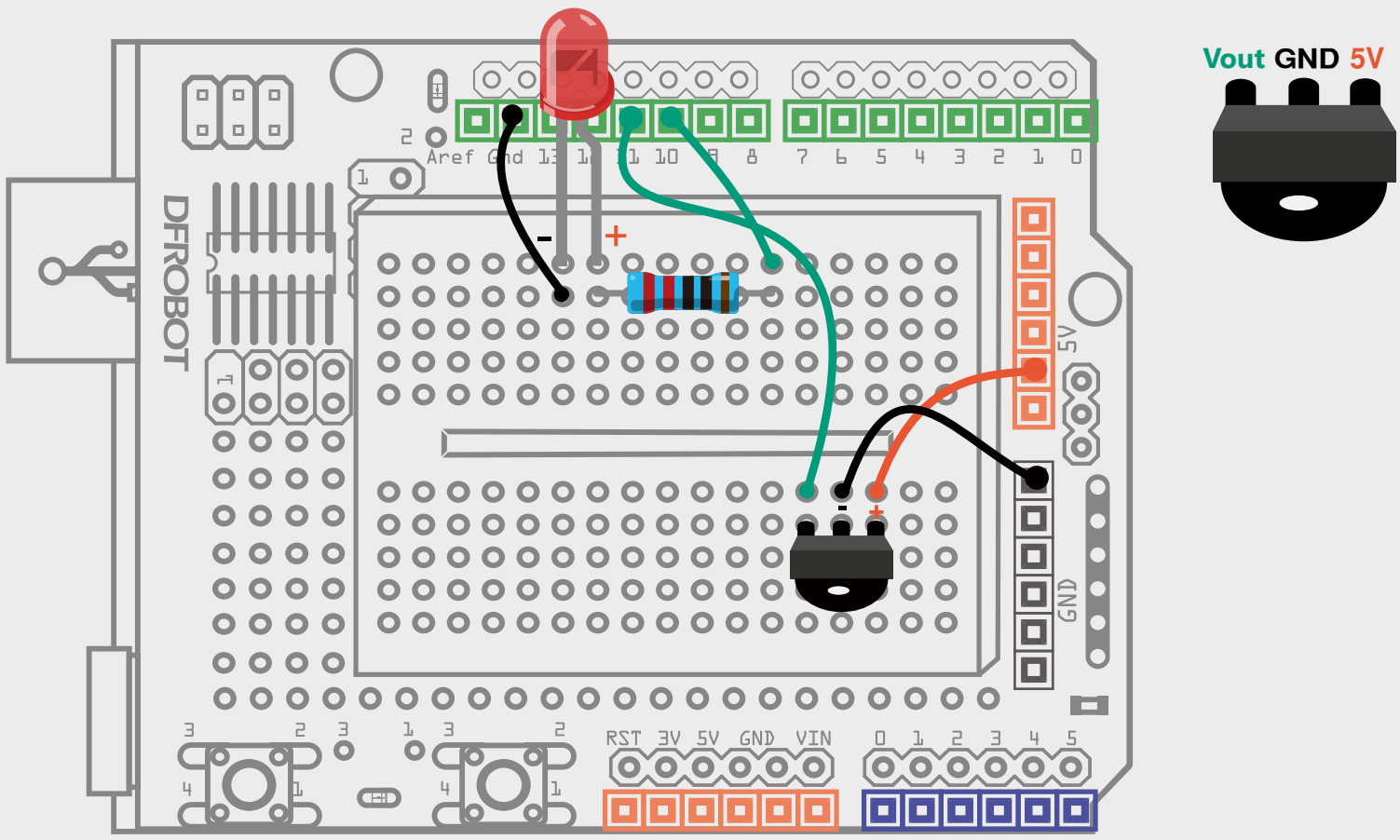


图14-2 红外遥控灯连线图

输入代码

这里不建议一步一步输入代码，可以在原有的代码上进行修改，观察下相对前一段代码增加了哪些内容。

样例代码14-2

```
#include <IRremote.h>
int RECV_PIN = 11;
int ledPin = 10;                    // LED  -  digital 10
boolean ledState = LOW;             // ledstate用来存储LED的状态
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  pinMode(ledPin,OUTPUT);           // 设置LED为输出状态
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);

    //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(results.value == 0xFD00FF){
        ledState = !ledState;           //取反
        digitalWrite(ledPin,ledState);  //改变LED相应状态
     }
    irrecv.resume();
  }
}
```

代码回顾

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 <IRremote.h>        //调用IRremote.h库
int RECV_PIN = 11;           //定义RECV_PIN变量为11
IRrecv irrecv(RECV_PIN);     //设置RECV_PIN(即引脚11)为红外接收端
decode_results results;      //定义results变量为红外结果存放位置

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results中。

```
if (irrecv.decode(&results))
```

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(results.value == 0xFD00FF)
```

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。

最后，继续等待下一组信号。

```
irrecv.resume();
```

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

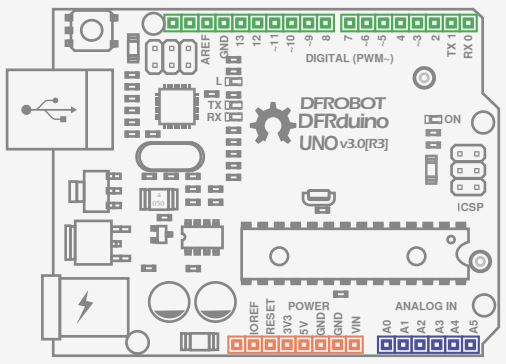
15

DFROBOT®
DRIVE THE FUTURE

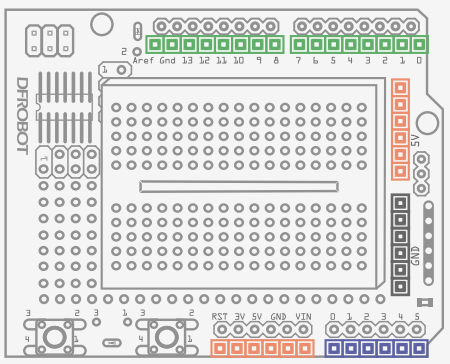
项目十五 红外遥控数码管

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



x1 **DFduino UNO R3**
（及配套USB数据线）



x1 **Prototype Shield**
原型扩展板+面包板

x10 **Jumper Cables**
M/M
跳线(公公头)

x8 **Resistor 220R**
220欧电阻

x1 **8-Segment LED**
八段数码管

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。

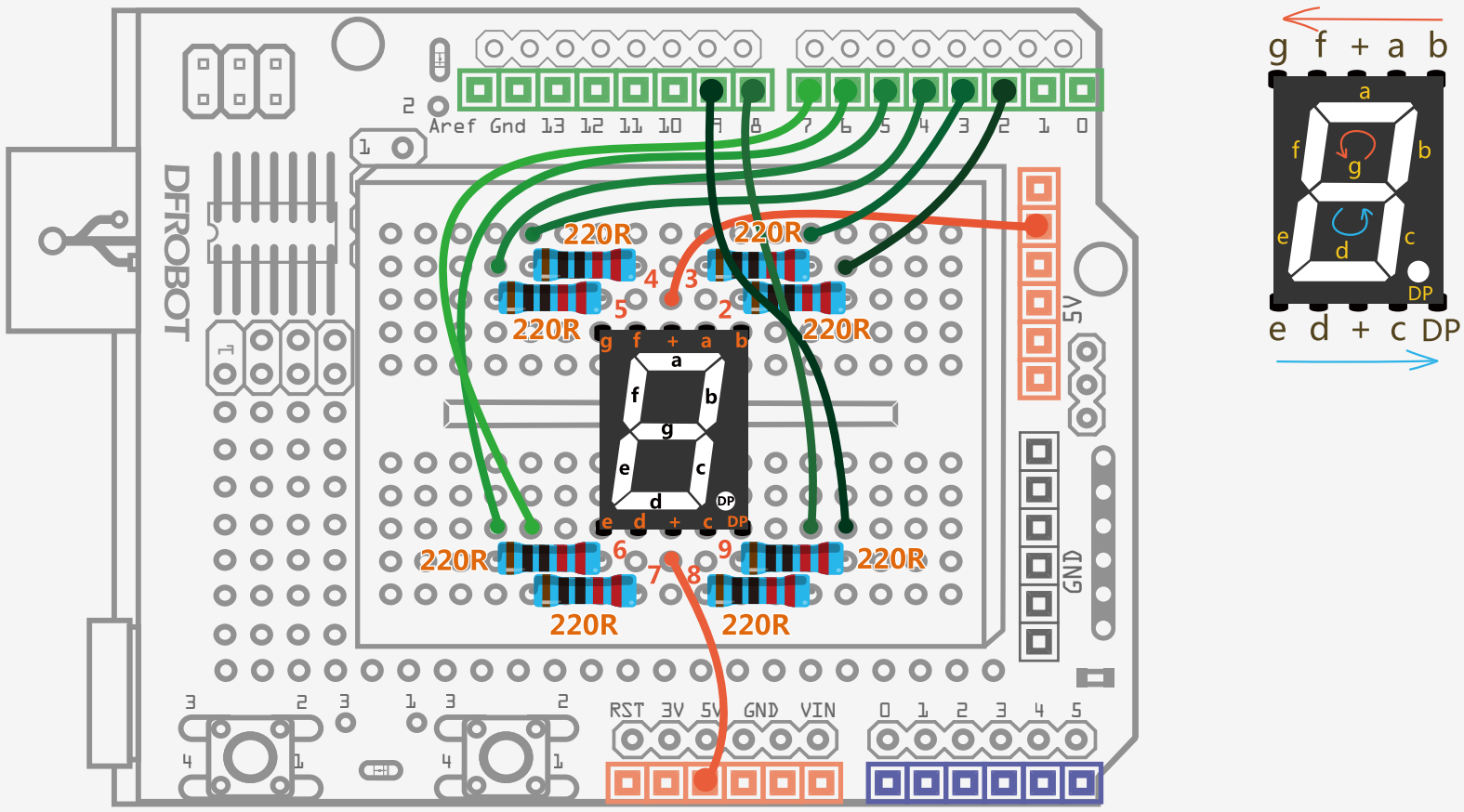


图15-1 数码管显示连线图

输入代码

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

按键开关

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。

右图15-2是个引脚说明图，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。

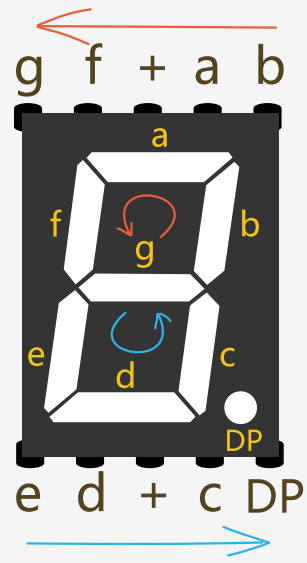


图15-2 引脚说明图

数码管一共有10个引脚。a~DP　这8个引脚接到数字口，那还有两个引脚呢？这是公共端，LED只有一端是不能被点亮的。我们在RGB灯那章讲到过共阴共阳的问题，数码管也存在共阴共阳问题。所谓共阳就是公共端接+5V，共阴则是公共端接GND。

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。硬件有了了解，我们来看看软件部分。

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。

代码回顾

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++){
      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。

```
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，那就是先输入代码2。

输入代码 2

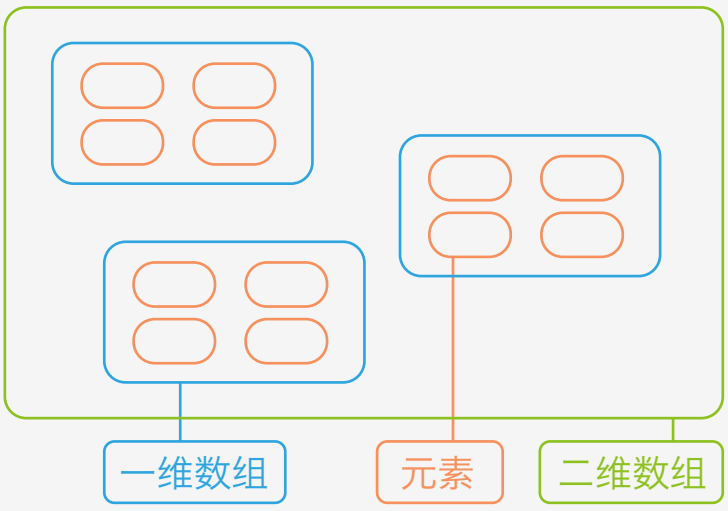
我们这里要教大家实现这个数码管0~9循环的代码另一种写法，上面我们说到了数组，通过创建10个数组用于显示0~9。这里同样也还是用数组写，区别在于代码1中其实准确的说应该叫一维数组，我们这里用到二维数据。这样一来，可以让代码看起来更简洁。动手输一下下面这段代码吧，看看是不是同样的效果！

样例代码15-2

```
//项目11 - 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);
    }
}
```

代码回顾 2

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。



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：

number[0][0]

```
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
};
```

number[9][7]

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

……

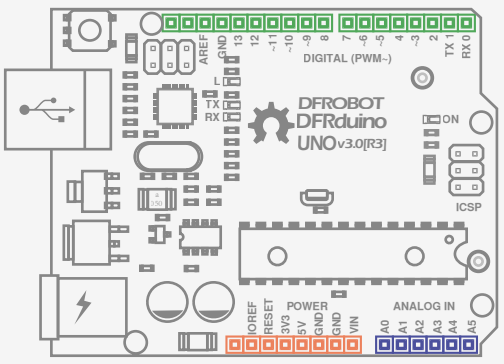
j=3 → numberShow(9) → i=9 → number[9][pin-2] → 显示数字9

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。

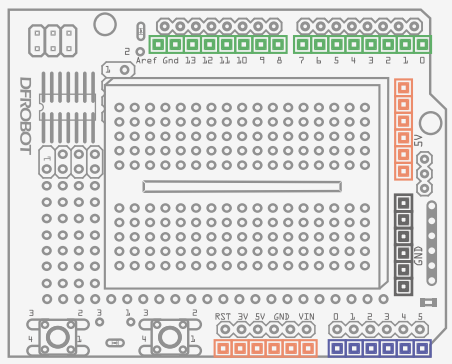
所需材料

所需元件



x1

DFduino UNO R3
（及配套USB数据线）



x1

Prototype Shield
原型扩展板+面包板

Jumper Cables
M/M
跳线(公公头)

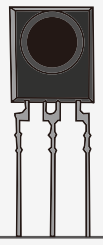
x13

Resistor 220R
220欧电阻

x8

8-Segment LED
八段数码管

x1



IR Receiver
红外接收管

x1

IR Remote
controller
迷你遥控器

x1

硬件连接

你会发现在硬件连接上同样就是把项目十四和十五结合在一起，并没有什么太大变化，如果在连接数码管时候有些不明白，可以回看一下项目十五。

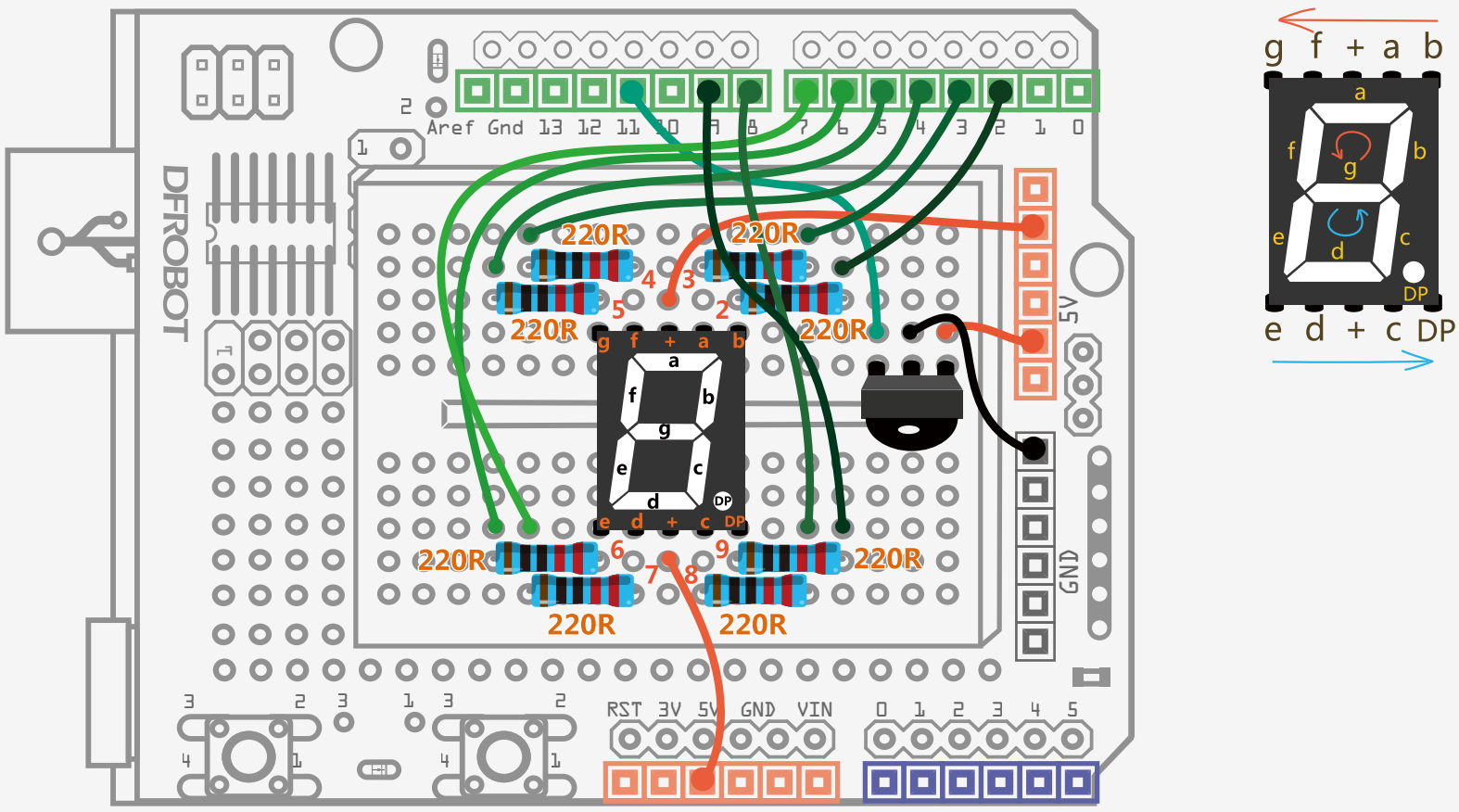


图15-3 红外遥控数码管连线图

输入代码

样例代码13-1

```
//项目十三 - 红外遥控数码管
#include <IRremote.h>        //调用IRremote.h库
int RECV_PIN = 11;           //定义RECV_PIN变量为11
IRrecv irrecv(RECV_PIN);     //设置RECV_PIN（即11引脚）为红外接收端
decode_results 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

long codes[12]=              //该数组用来存放红外遥控器发出的红外码
{
   0xFD30CF,0xFD08F7,     // 0 ,1
   0xFD8877,0xFD48B7,     // 2 ,3
   0xFD28D7,0xFDA857,     // 4 ,5
   0xFD6897,0xFD18E7,     // 6 ,7
   0xFD9867,0xFD58A7,     // 8 ,9
   0xFD20DF,0xFD609F,     // + ,-
};

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 = 2; pin <= 9 ; pin++){
     digitalWrite(pin, number[i][pin - 2]);
  }
}

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

  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() {
  //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
  if (irrecv.decode(&results)) {
      for(int i = 0; i <= 11; i++){
          //判断是否接收到0~9按键的红外码
          if(results.value == 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.value == 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.value == 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.value, HEX);          //串口监视器查看红外码
    irrecv.resume();                                    //等待接收下一个信号
  }
}
```

图15-4 遥控数按键说明

下载完代码后，尝试按下上图15-4指出部分的按钮，看看是数码管是个怎样的变化。

代码回顾

程序一开始还是对红外接收管的一些常规定义，按项目十二搬过来即可。

```
#include <IRremote.h>       //调用IRremote.h库
int RECV_PIN = 11;          //定义RECV_PIN变量为11
IRrecv irrecv(RECV_PIN);    //设置RECV_PIN(即引脚11)为红外接收端
decode_results 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
```

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，0x-表示是16进制。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。

```
long codes[12]=             //该数组用来存放红外遥控器发出的红外码
{
  0xFD30CF,0xFD08F7,              // 0 ,1
  0xFD8877,0xFD48B7,              // 2 ,3
  0xFD28D7,0xFDA857,              // 4 ,5
  0xFD6897,0xFD18E7,              // 6 ,7
  0xFD9867,0xFD58A7,              // 8 ,9
  0xFD20DF,0xFD609F,              // 前进 ,后退
};
```

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 (irrecv.decode(&results))
```

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.value == codes[i]&& i <= 9)
if (results.value == codes[10]&& currentNumber != 0)
if (results.value == 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。就不一一详细说了。

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！