目　　录

初识 Arduino

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，同时也支持通过串行通信与计算机上运行的软件（如 Processing, MaxMSP）进行交互，进一步明确这些软件与 Arduino 通信的具体方式或用途，以实现更复杂的功能。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

Arduino 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，并找到 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。即使不懂电脑编程，你可以也能用 Arduino 做出酷炫的东西，比如闪烁灯光，也能对马达进行控制，还能通过传感器感受外界环境，并用执行器做出相应的反应。

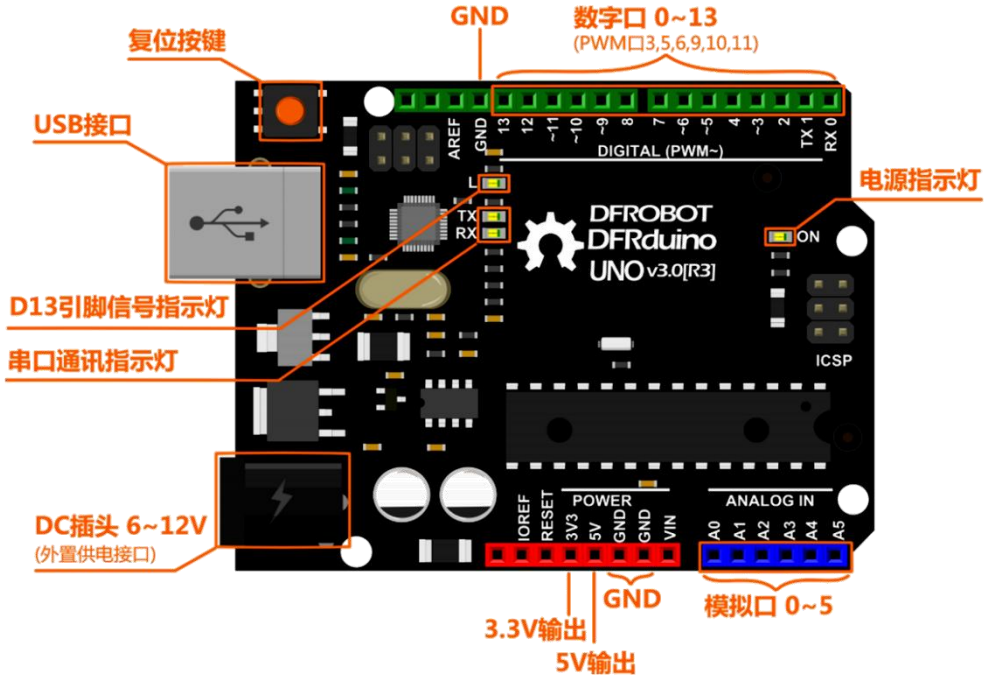
Arduino 名称由来

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。



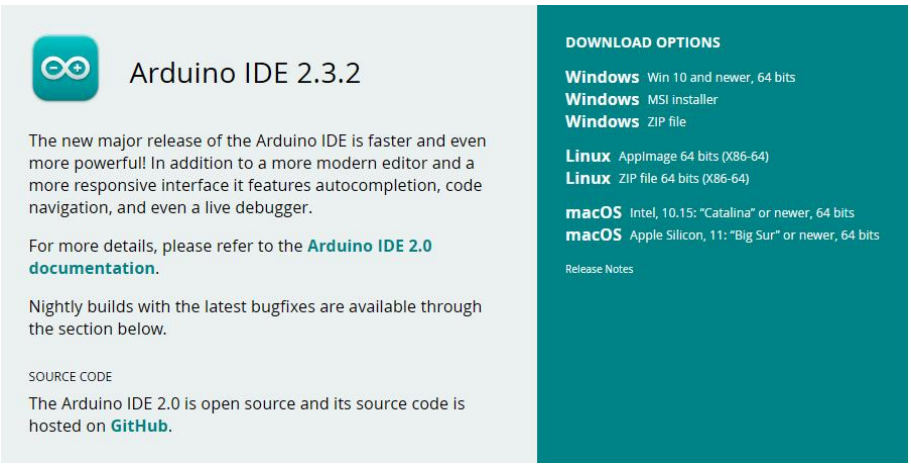
初次使用

1.下载 Arduino IDE

打开网页输入网址

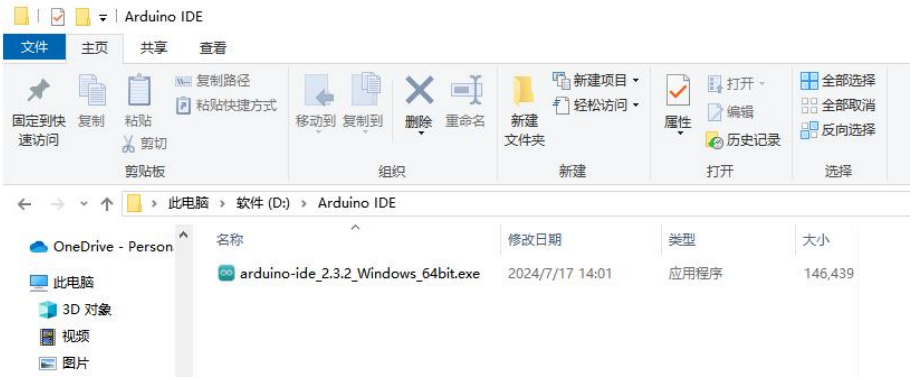
https://www.arduino.cc/en/software

进入到页面后，找到下图显示的部分。



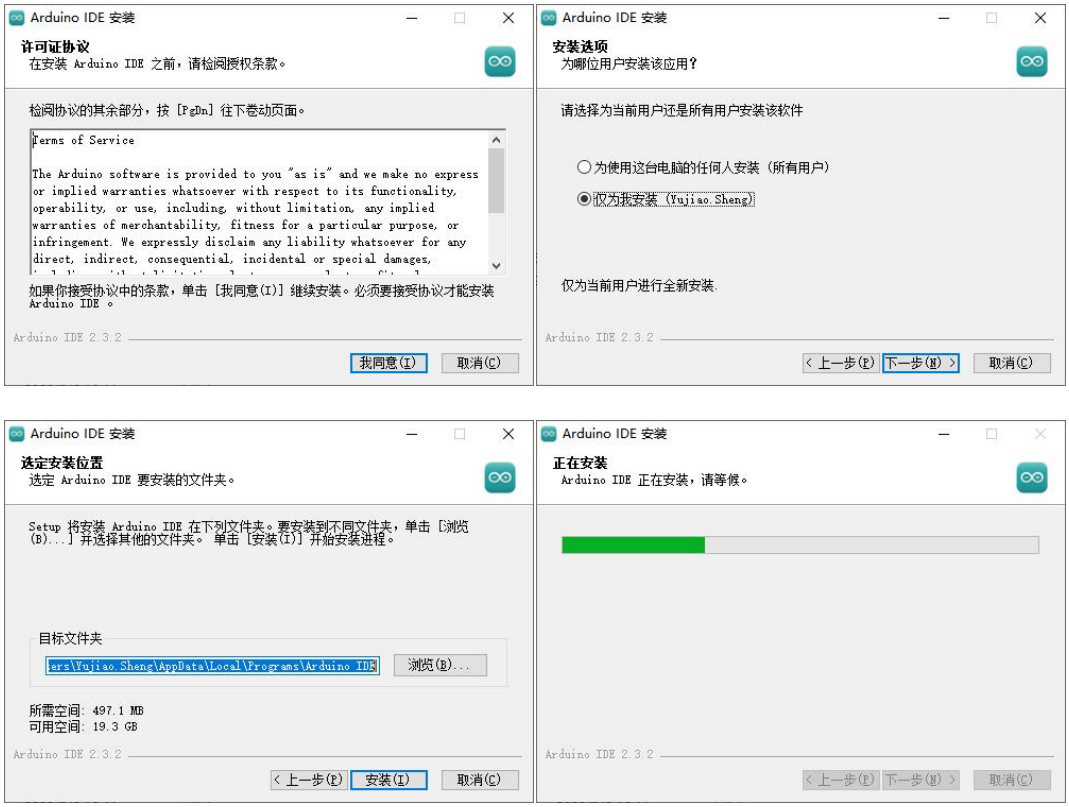
Windows10 以上用户，点击下载“**Windows**”的对应版本。如果 macOS，Linux 用户则选择相应的系统。

要在 Windows 计算机上安装 Arduino IDE 2，只需运行从官网软件页面下载的文件。



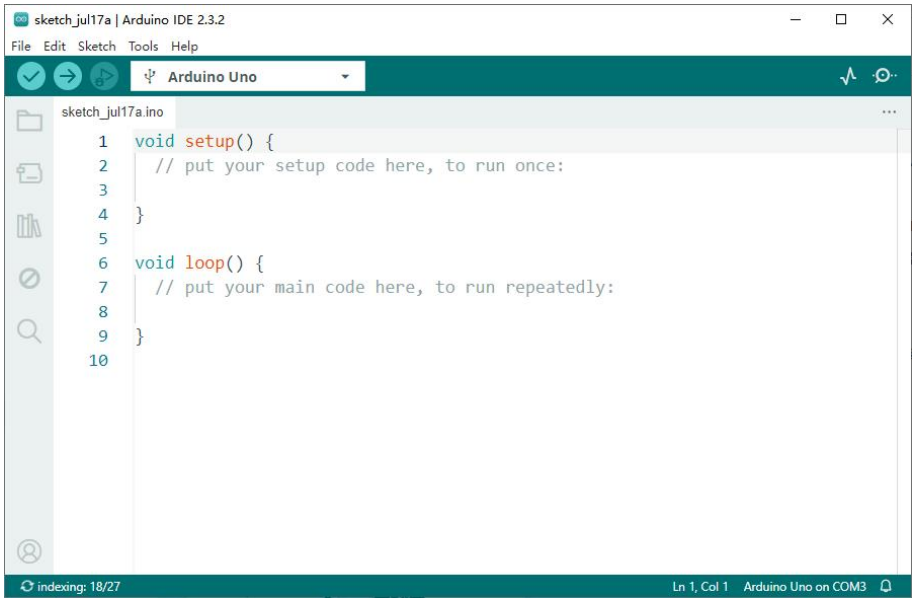
2. 安装 Arduino IDE

按照安装指南中的说明进行操作。安装可能需要几分钟。

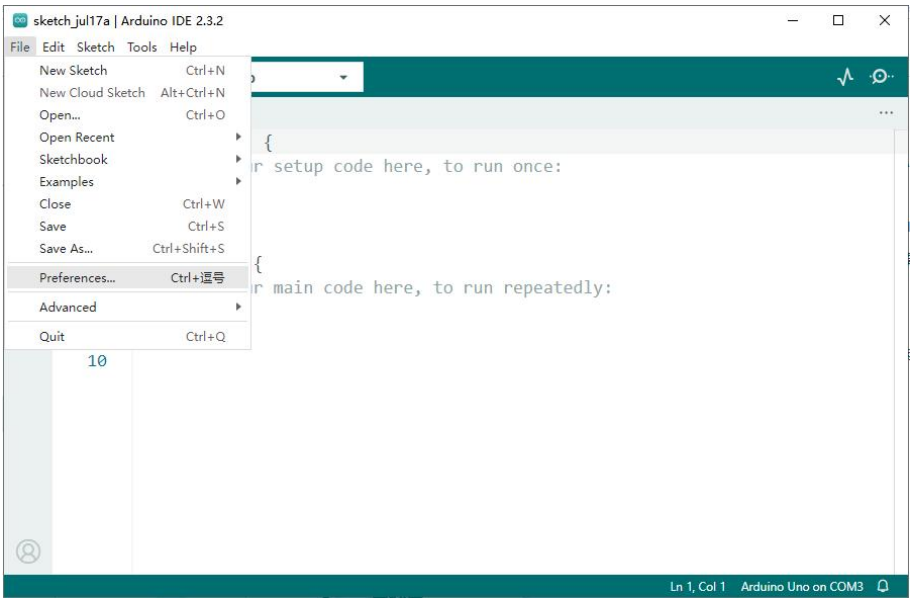


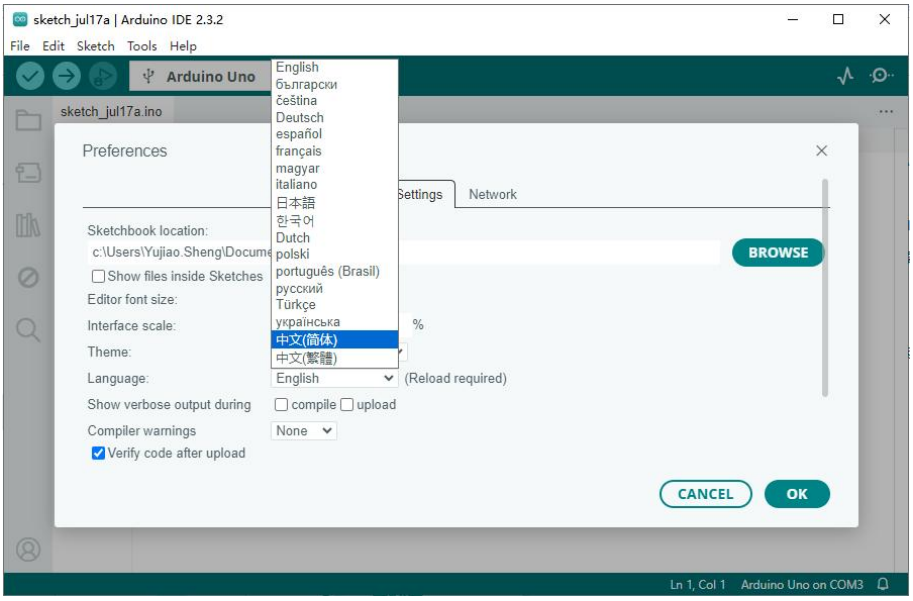
3. 认识 Arduino IDE

打开 Arduino IDE，就会出现 Arduino 的编辑界面。

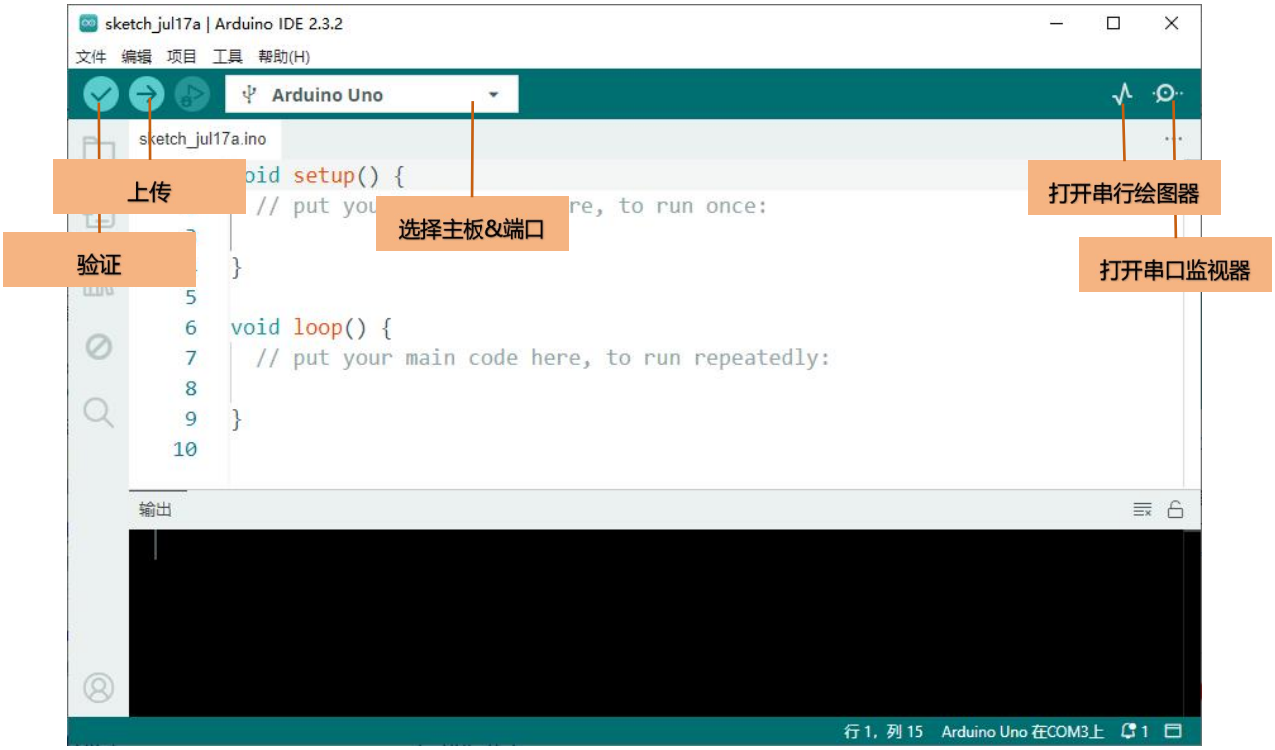


如果英文界面，你不太习惯的话，可以先更改为中文界面。选择菜单栏 File ->Preferences。





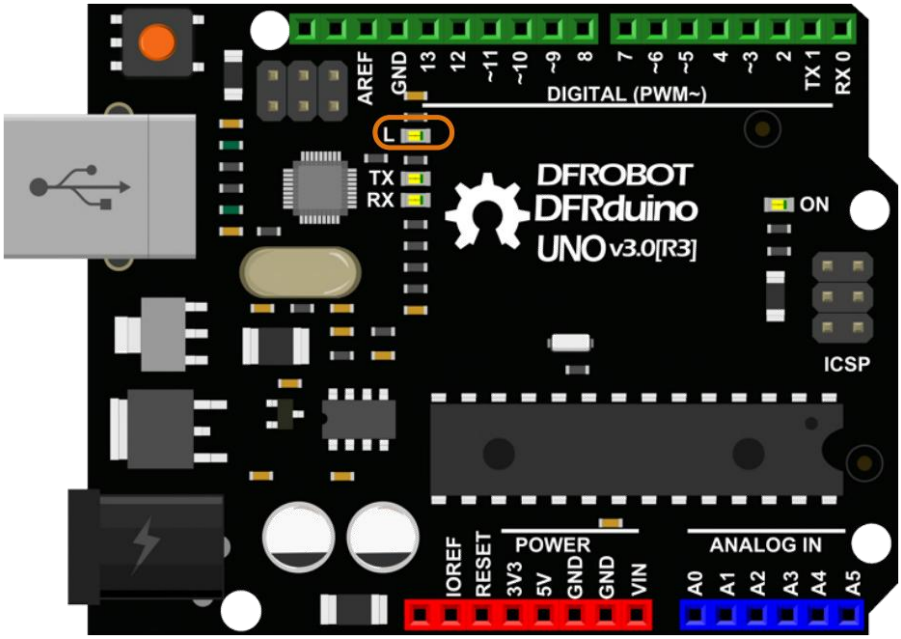
Arduino IDE 是 Arduino 产品的软件编辑环境。简单的说就是用来写代码，上传代码的地方。任何的 Arduino 产品都需要上传代码后才能运作。我们所搭建的硬件电路是辅助代码来完成的，两者是缺一不可的。如同人通过大脑来控制肢体活动是一个道理。如果代码就是大脑的话，外围硬件就是肢体，肢体的活动取决于大脑，所以硬件实现取决于代码。



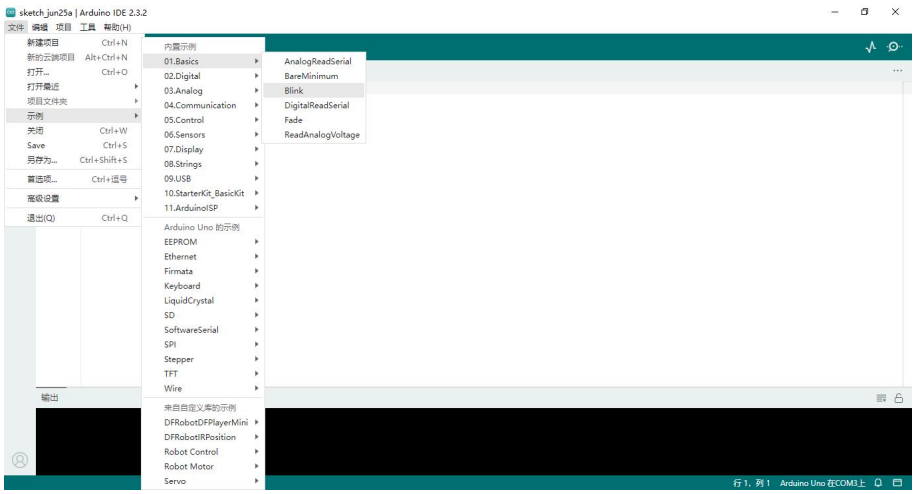
Arduino IDE 基本也只需要用到上面标示出来的部分就可以了，上图大部分的白色区域就是代码的编辑区，用来输入代码的。注意，输入代码时，要切换到英文输入法的模式。下面黑色的区域是消息提示区，会显示验证或者上传是否通过。

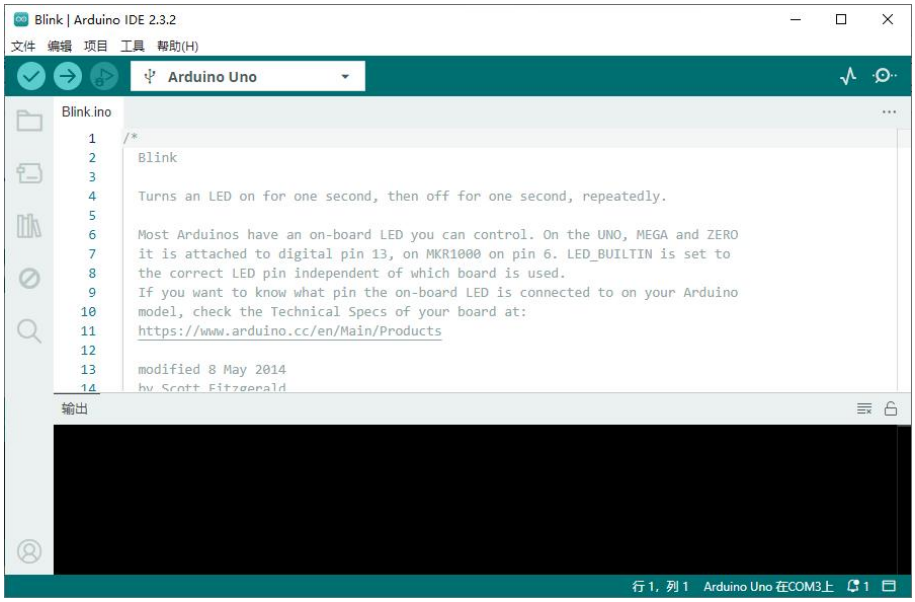
4. 运行一个 Blink 程序

上传一个最简单的代码，既可以帮你熟悉如何上传程序，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。

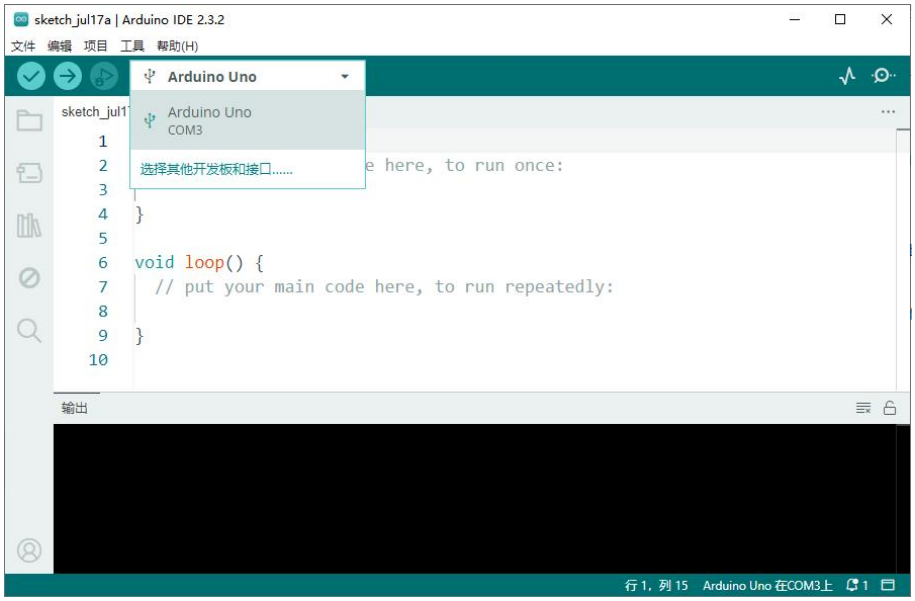


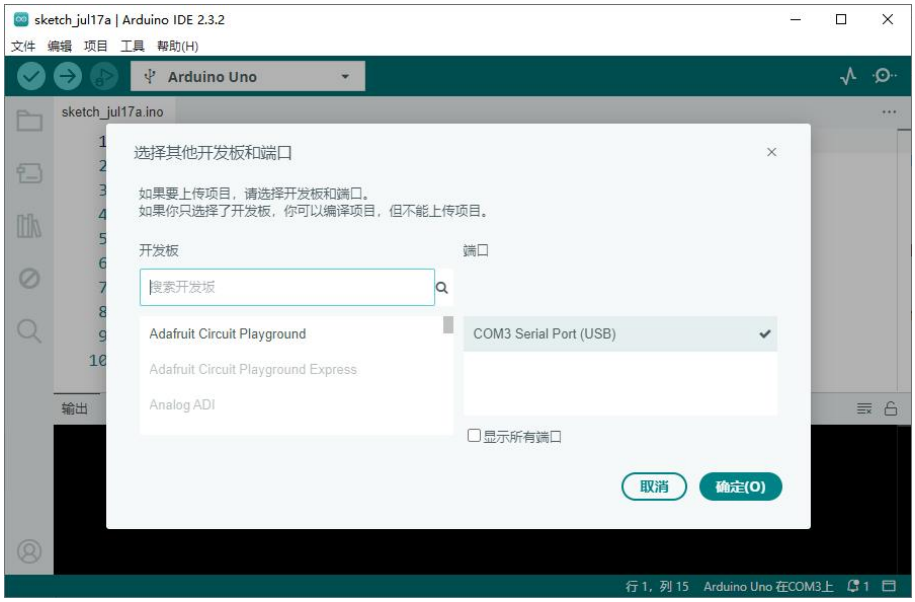
插上 USB 线，打开 Arduino IDE 后，按照下图找到“Blink”代码。

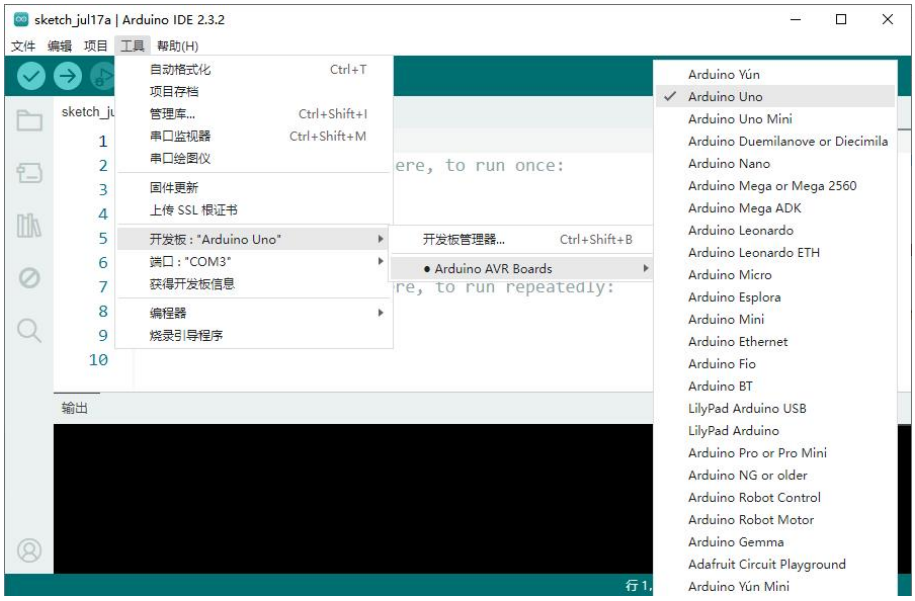




在执行其他操作之前，我们应该选择要上传到哪个主板。在验证和上传按钮旁边，您应该看到一个下拉菜单，在大多数情况下，它会显示连接到您计算机的 Arduino 主板。如果没有自动检测到您的主板，您可以按下拉菜单中的“选择其他开发板和接口……”并按照说明进行操作，或者转到工具栏菜单中的工具 -> 开发板或端口以手动选择开发板和端口。



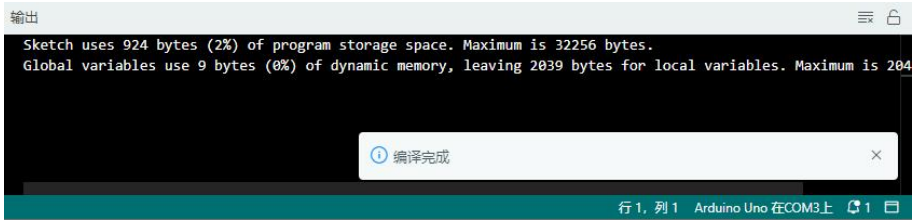




写完一段代码并选择好开发板和端口后，我们都需要编译一下，看看代码有没有错误。点击“验证”。

编译中.....

编译完成！



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，然后再上传到 Arduino 中。

最后，点击“上传”。

上传成功！

你就可以观察到板载 L 灯会像程序编写的一样一直闪烁。

回顾

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:

选择开发板和端口 -> 验证 -> 上传

项目 - SOS 求救信号器

本项目将继续使用【闪烁第一个 LED】所搭建的电路（注意引脚有所更换），只需要对代码进行一定的改动，就能让我们的 LED 发出 SOS 求救信号了。

国际摩斯码是一种独特的字符编码系统，其精妙之处在于它利用简单的两种信号：横杠（长音）和点（短音），通过不同的组合方式来代表英文中的每一个字母以及数字。这种设计极大地简化了信息传递的复杂性，因为仅需两种基本信号形态，就能构建出完整的通信语言。这种创造性的编码方式展现了前人非凡的智慧与创新能力，确实令人钦佩不已。

我们正好可以通过 LED 发光的长短来代表不同的字母。通过长闪烁和短闪烁来分别表示横杠和点。在这个项目中，我们就拼写 SOS 这三个字母。

通过查阅摩斯码表，我们可以知道，字母“S”的摩斯码是三个点，我们这里用三次短闪烁表示。字母“O”则是三个横杠，用三次长闪烁表示。

硬件连接

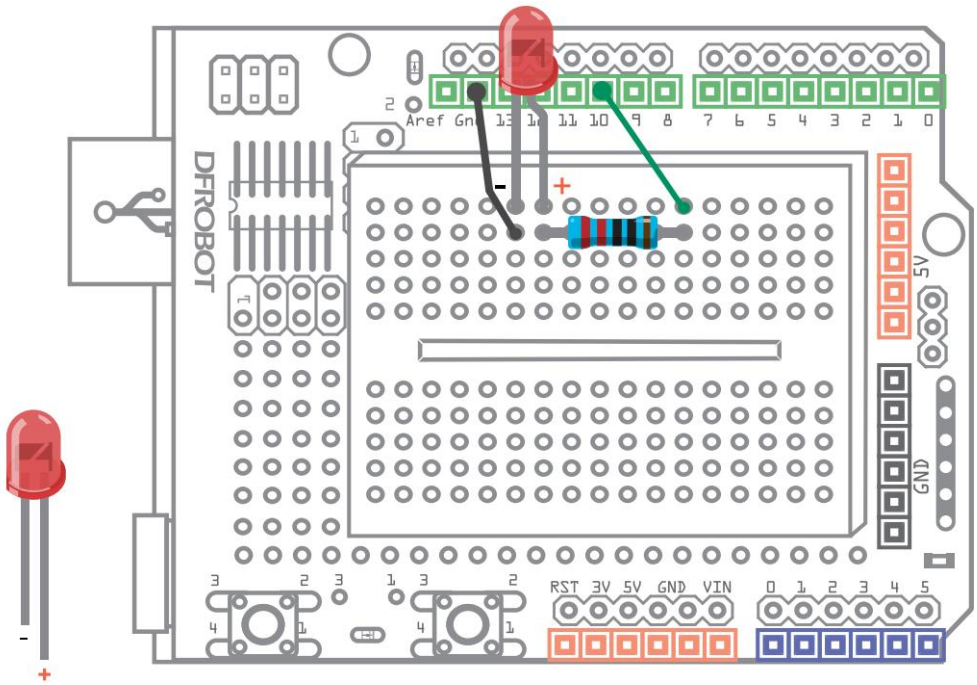


图 1 SOS 求救信号器

示例代码

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

样例代码 1：

```
int ledPin = 1010;
void setup() {
 pinMode(ledPin, OUTPUT);
}
void loop() {
  // 三个短闪烁来表示字母“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(101000); //1000 毫秒延时产生字母之间的间隙

  //三个长闪烁来表示字母“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(1000); //1000 毫秒延时产生字母之间的间隙

//再用三个短闪烁来表示字母“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 SOS 信号前等待 5 秒
}
```

上面的写法固然正确，是不是觉得有点繁琐呢？如果有 100 个字母，难不成还重复 100 遍吗？有没有更好的编写程序的方法呢？想必发明编程的人也考虑到这个问题了，所以我们有了更好的一种写法。

我们先来看一下样例代码 2。

样例代码 2:

```
//项目 -- SOS 求救信号器
int ledPin = 10;
void setup() {
 pinMode(ledPin, OUTPUT);
}
void loop() {
 // 三个短闪烁来表示字母“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
 }
```

```
 //1000 毫秒延时产生字母之间的间隙
 delay(1000);
 //三个长闪烁来表示字母“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
 }
 //1000 毫秒延时产生字母之间的间隙
 delay(1000);
 // 再用三个短闪烁来表示字母“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 SOS 信号前等待 5 秒
 delay(5000);
}
```

在输入代码的时候，注意保持代码层次的清晰，除了美观外，也便于你日后检查代码。编译通过后，上传代码到 Arduino 中，如果一切顺利的话，我们将看

到 LED 闪烁出摩斯码 SOS 信号，等待 5 秒，重复闪烁。

通过使用 Arduino 板外接电池，并将整个装置封装在一个防水盒子里，我们就能创建出一个便携式设备，用于在紧急情况下发送国际通用的求救信号 SOS。这一信号在航海和登山等户外活动中尤为重要，因为它能够快速有效地传达出求助信息。

我们接着来分析下代码。

代码回顾

代码 loop()前的部分与项目【闪烁第一个 LED】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，你可以看到与项目【闪烁第一个 LED】中类似的语句用来控制 LED 的打开和关闭，并通过延时 delay()实现长短闪烁功能。然而，有所不同的是这次主函数包含了三个独立的代码段。

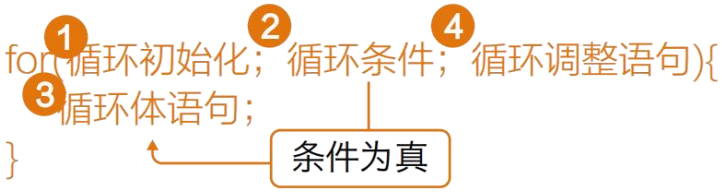
第一段代码段是输出代表字母“S”的三个点：

```
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 开关的控制代码位于一对花括号（也称为大括号）内部，这个区域被定义为一个代码段。花括号都必须成对出现，即开启一个花括号必须有一个相应的闭合花括号，否则编译器在处理代码时会因为语法错误而无法通过编译。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，之后再在两个花括号之间输入代码，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。产生这样的效果是因为使用了 for 循环（用来重复执行循环体中的代码块）。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 →

第二轮：2 → 3 → 4

……

直到 ② 不成立(为假)，则结束执行 for 循环，并继续向下执行程序。

（注意：for 循环首先执行的是初始化部分，然后进行循环条件判断，如果条件为真，则执行循环体，之后才执行循环调整语句的部分。）

来看下我们程序中的 for 循环：

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

第一次循环：

- 声明并初始化变量 x=0。
- 判断 x<3，为真，则执行循环体中的程序。
- x 自加，更新 x 为 1。

第二次循环：

- 此时 x=1。
- 判断 x<3，为真，则执行循环体中的程序。
- x 自加，更新 x 为 2。

……

循环结束:

- 在下一次循环开始前，x 的值被自加到 3。
- 判断 3<3，为假，因此循环结束，不再执行循环体。并继续向下执行。

（x++是 x=x+1 的简写，实现 x 的迭代（自加）功能。）

我们这里需要它循环 3 次，所以设置为 x<3。从 0 开始计算，0，1，2，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。下面就介绍一些常用的比较运算符。

比较运算符

for 循环中的“<”称之为比较运算符。比较运算符在代码中是用作判断的，用于比较两个值。

我们常用的比较运算符有:

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

特别要说明一下，比较运算中等于必须是两个等号（单等号的作用是赋值）。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+、 -、 *、 /（加、减、乘、除）这些常用的算术运算符。

现在知道 for 循环是如何运作的了吧！我们的主函数 loop()中有三个 for 循环，按照下面表格运行，最终完成 SOS 的输出。

第一个 for 循环	短闪烁 3 次，代表输出 3 个点，也就是字母“S”
第二个 for 循环	长闪烁 3 次，代码输出 3 个横杠，也就是字母“O”
第三个 for 循环	短闪烁 3 次，代表输出 3 个点，也就是字母“S”

局部变量和全局变量

必须要注意的，我们这里要引用一个新的概念：局部变量和全局变量。局部变量，这类变量只在自己的代码块内起作用。像是该程序的 for 循环中定义的变量 x 就属于局部变量，这意味着每个 for 循环都拥有自己独立的 x 变量。这些 x 变量之间互不影响，各自在其所属的循环中控制迭代过程，确保了即使在同一程序中存在多个 for 循环，它们之间的变量也不会产生冲突。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。像我们这里的 ledPin，就能在整个程序中起作用。

在构建 SOS 求救信号器项目时，为了确保 SOS 信号中的每个字母（S、O、S）能够被清晰区分，我们在每个 for 循环迭代完成（表示一个字母）后，特意插入了 1000 毫秒的延时。这个延时使得 SOS 字母之间的停顿变得明显，有助于接收方准确识别每个字母。而在整个 SOS 信号序列完成后，即三个字母（S、O、S）均通过 LED 灯的闪烁表示完毕后，我们进一步设置了一个 5 秒的延时。这个较长的延时是在程序准备重新进入主循环 loop()并执行下一轮 SOS 信号之前的一个等待时间，确保了 SOS 信号之间有足够的间隔，避免信号间的混淆。

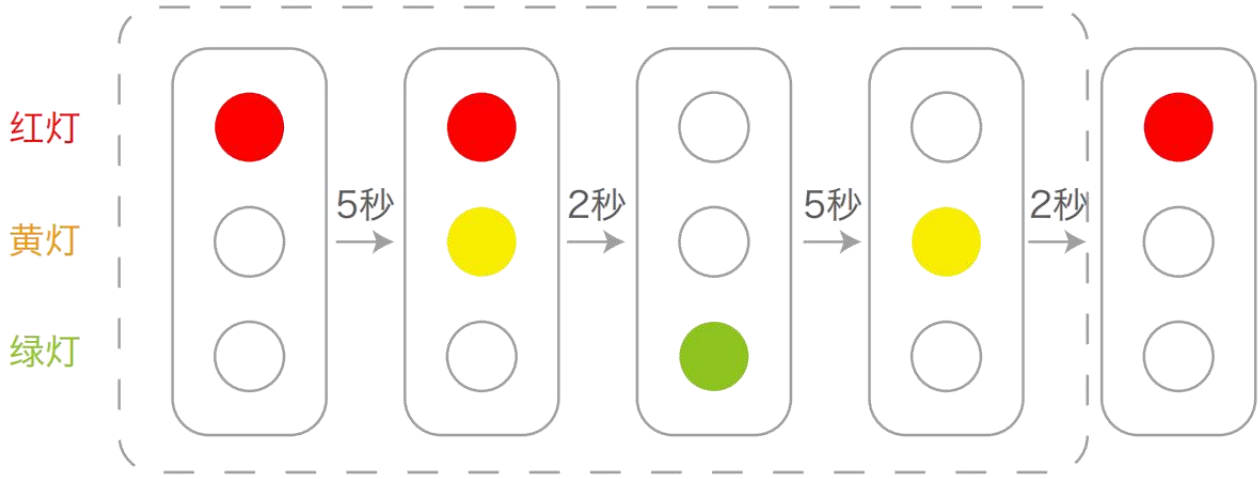
好了，我们 SOS 求救信号器项目就算告一个段落了。有所收获吗?

课后练习

我们以【闪烁第一个 LED】和【SOS 求救信号器】两个项目作为基础，现在尝试自己做个交通信号灯吧，下图是交通信号灯的运行过程，虚线框的是程序的循环部分。自己动手试一下吧!

提示：原先我们只控制了一个 LED 灯，现在要扩展到同时控制三个不同颜色的 LED 灯。电路连接的基本原理保持不变，但我们需要利用三个不同的数字端口

来分别控制这三个 LED 灯。在编程时，关键在于通过调整这些数字端口的高低电平信号，来独立控制每个 LED 灯的亮灭状态。

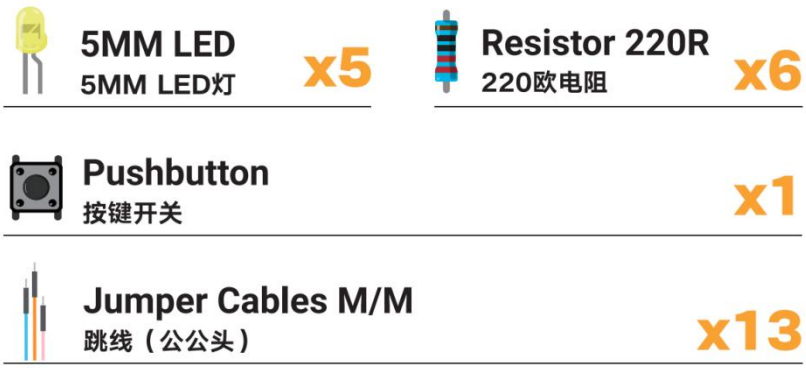


项目 - 交通信号灯

有没有试着做【SOS 求救信号器】的课后作业呢？做出来的话，说明你已经基本掌握之前所学的东西了，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！我们这回就基于课后作业的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。当按键被按下时，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，也会学习在代码中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！

元件清单



*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键的，它叫做下拉电阻（我们后面会解释）。

硬件连接

按图 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。

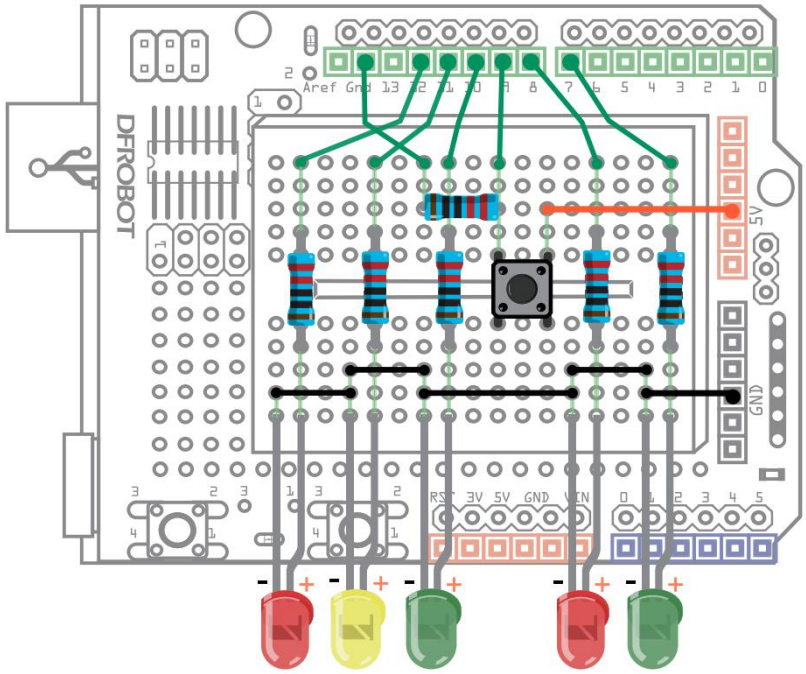


图 1 交通信号灯连接图

示例代码

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

样例代码:

```
// 项目 - 交通信号灯

int carRed = 12;     // 汽车红灯引脚
```

```
int carYellow = 11; // 汽车黄灯引脚
int carGreen = 10;  // 汽车绿灯引脚
int button = 9;     // 按钮引脚
int pedRed = 8;     // 行人红灯引脚
int pedGreen = 7;   // 行人绿灯引脚
int crossTime = 5000; // 允许行人通过的时间（毫秒）
unsigned long changeTime; // 按钮按下后的时间

void setup() {
    // 初始化引脚模式
    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

    // 初始状态设置
    digitalWrite(carGreen, HIGH); // 车行绿灯亮
    digitalWrite(pedRed, HIGH);   // 人行红灯亮
}

void loop() {
```

```
    int state = digitalRead(button);
    // 检测按钮状态及时间间隔
    if (state == HIGH && (millis() - changeTime) > 5000) {
        // 调用变灯函数
        changeLights();
    }
}

void changeLights() {
    // 汽车绿灯变黄灯
    digitalWrite(carGreen, LOW);
    digitalWrite(carYellow, HIGH);
    delay(2000); // 等待 2 秒

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

    // 行人红灯变绿灯
    digitalWrite(pedRed, LOW);
    digitalWrite(pedGreen, HIGH);
```

```
    // 行人绿灯亮，持续 crossTime 毫秒
    delay(crossTime);

    // 闪烁行人绿灯，提示时间快到
    for (int x = 0; x < 10; x++) {
        digitalWrite(pedGreen, HIGH);
        delay(250);
        digitalWrite(pedGreen, LOW);
        delay(250);
    }

    // 行人红灯再次亮起
    digitalWrite(pedRed, HIGH);
    delay(500);

    // 恢复到汽车绿灯亮状态
    digitalWrite(carRed, LOW);
    digitalWrite(carYellow, HIGH);
    delay(1000);
    digitalWrite(carYellow, LOW);
    digitalWrite(carGreen, HIGH);

    // 更新 changeTime
```

```
    changeTime = millis();
}
```

下载完成后，可以尝试按下按键。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。一旦行人按键被按下，请求过马路，那么汽车灯由绿变黄，变红，接着行人灯就由红变绿。在行人通行的过程中，设置了一个过马路的时间 crossTime，一旦到点，行人绿灯开始闪烁，提醒行人快速过马路。闪烁完毕，最终，又回到了开始的状态，汽车灯为绿灯，行人灯为红灯。

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【SOS 求救信号器】的课后作业示例图那样，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 unsigned long。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 数据类型不存储负数，所以存储的范围就变成了 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类型，并且不为负，我们就可以考虑使用 unsigned long 型。你可以用笔计算下，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 和按钮键进行一些设置，在设置时需要注意的是：

```
  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，按键要设置为 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的例子，变量好比用来放东西的盒子，把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（无符号整型）	2 byte	0～65535
long（长整型）	4 byte	-2147483648 ~ 2147483647
unsigned long（无符号长整型）	4 byte	0 ~ 4294967295
float（单精度浮点型）	4 byte	-3.4028235E38 ~ 3.4028235E38
double（双精度浮点型）	8 byte	-1.7976931348623157E+308 ~ 1.7976931348623157E+308

从上面表格可以看到，变量的类型有很多，不同的类型数对应不同的变量，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的，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给对应的——pin，pin 代表对应的引脚号。这里需要读取的是按键信号，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 来代替。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 的值来判断按键是否被按下：

```
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，是 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（单位为毫秒）。

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，大于五秒则执行 if 花括号内的语句，如果在 5 秒之内则不执行。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 2 分别显示了正面与背面。而图 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。

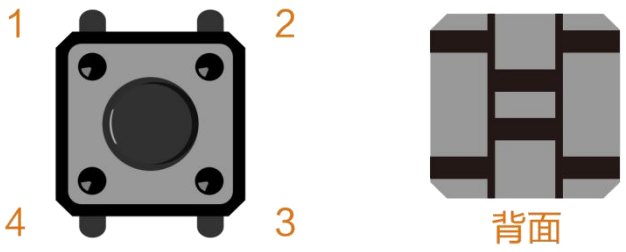


图 2 按键结构图

图 3 按键原理图

在图 4 中，按键仅就是起到一个通断的作用。但在本项目中，按键要能被读取状态，所以除了要将按键的 1、2（或者 3、4；1、4；2、3）引脚连入电路中，还需要接入信号口。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。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)。

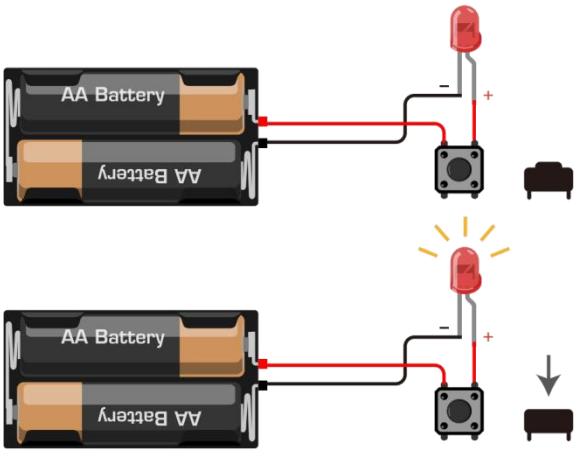


图 4 按键示意图

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，当输入电路状态为高电平（high）的时候，电压要尽可能接近 5V；输入电路状态为低电平（low）的时候，电压要尽可能接近 0V。为了避免状态偏离所需电压造成的电压浮动现象如果不能确保

状态接近所需电压，这部分电路就会产生电压浮动。所以，在按键电路中，我们将一个电阻在按钮串联在输入（input）引脚和接地（ground）之间，将按键未被按下的输入状态固定在低电平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。

可以从下面两张图看到，图 5 五是未接下拉电阻的电路，按键未没被按下时，input 输入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。然而图 6 六是接了下拉电阻的电路，当按键未没被按下时，输入引脚通过电阻接地，确保为 0V，不会产生电压浮动现象。

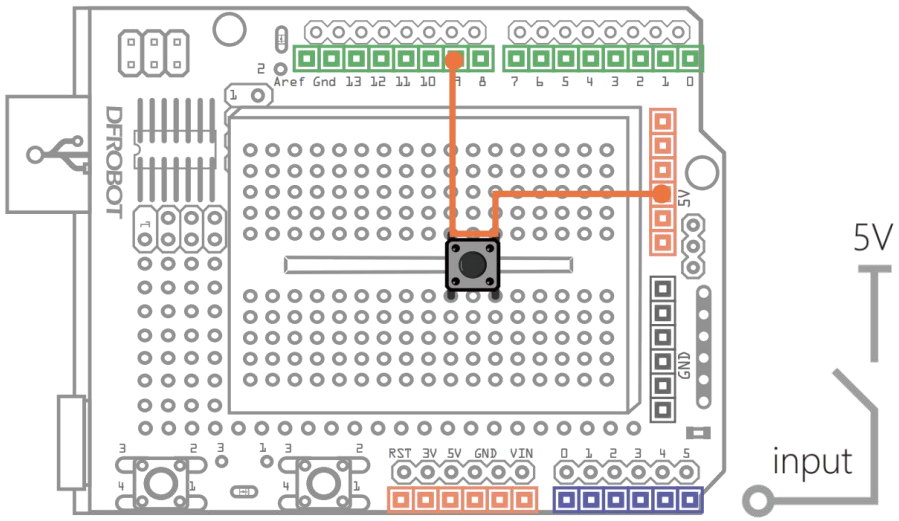


图 55 未接下拉电阻

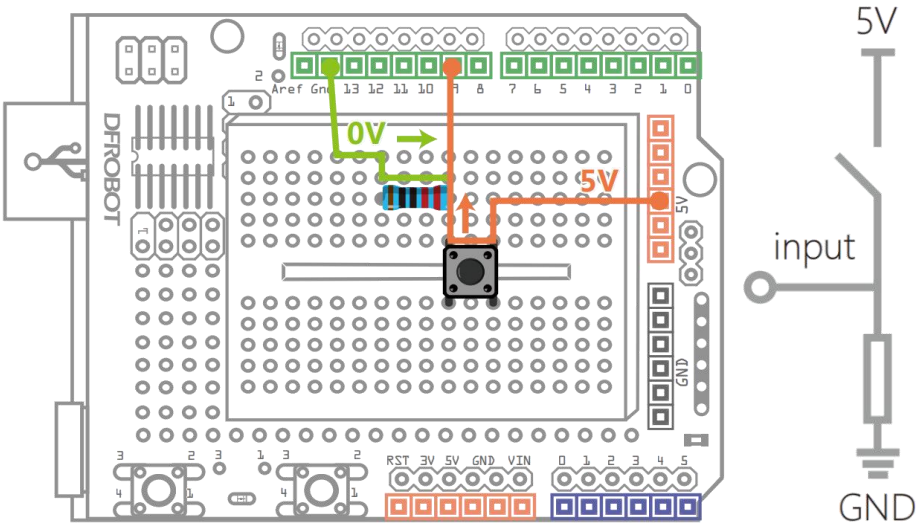
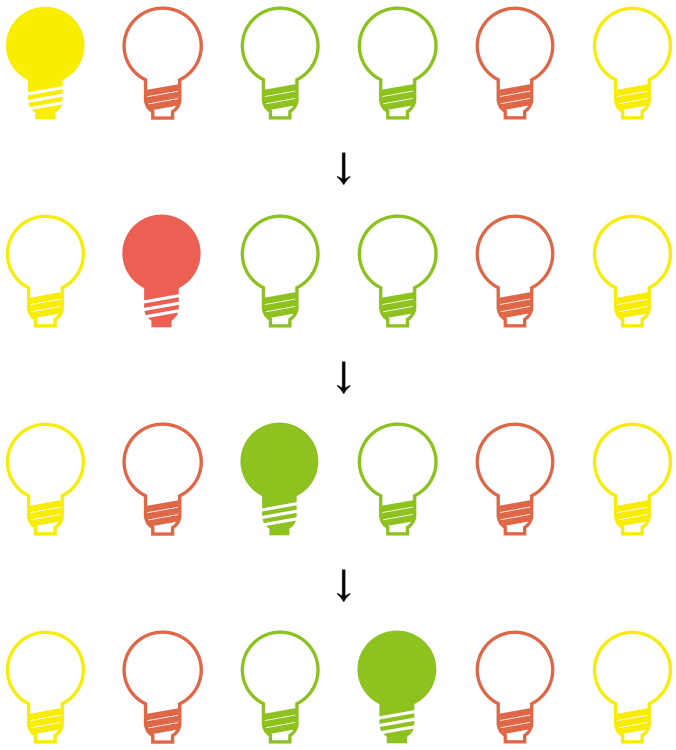
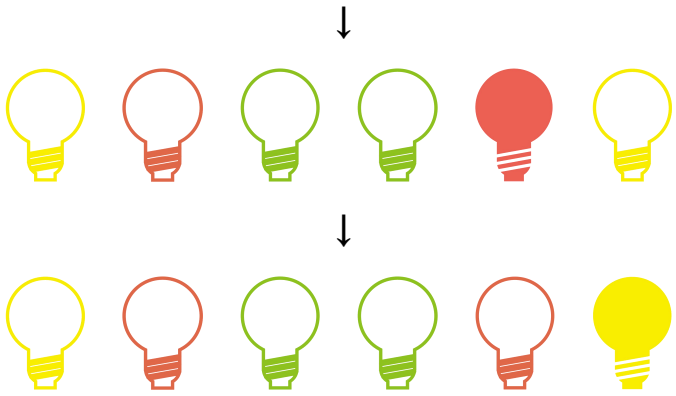


图 66 有下拉电阻

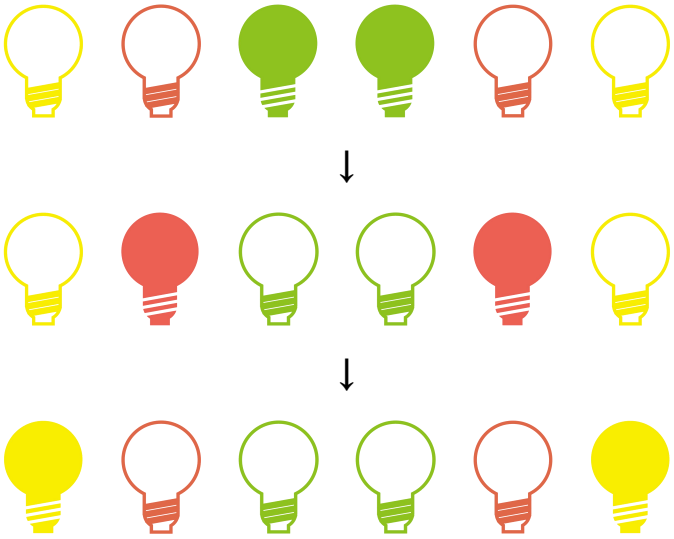
课后练习

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，重复循环这个过程。

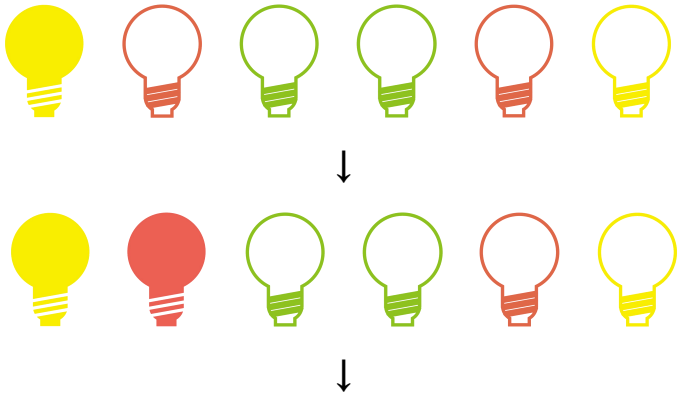


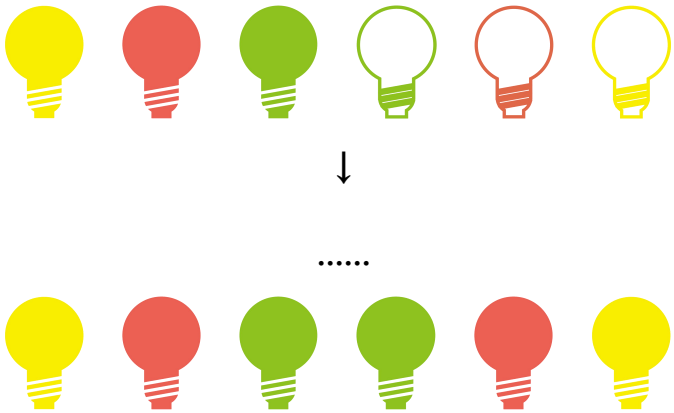


2、如果上面那个你已经完成了的话，可以尝试一下，先从中间的灯开始亮起，依次向两边扩开。下图是个变换过程的示意图。



3、再比如，从左至右，依次亮起 1 个，2 个，3 个……

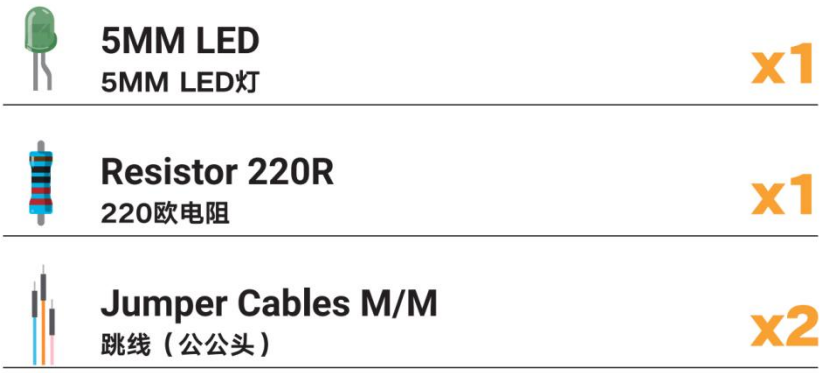




项目 - 呼吸灯

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，就是通过程序来控制 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。我们动手做一个呼吸灯，在做的过程中体会 PWM 的神奇力量！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。

元件清单



硬件连接

这个项目的硬件连接与项目【闪烁第一个 LED】类似。但需要注意连接的引脚有所区别，请参考下面的连线图进行连接。

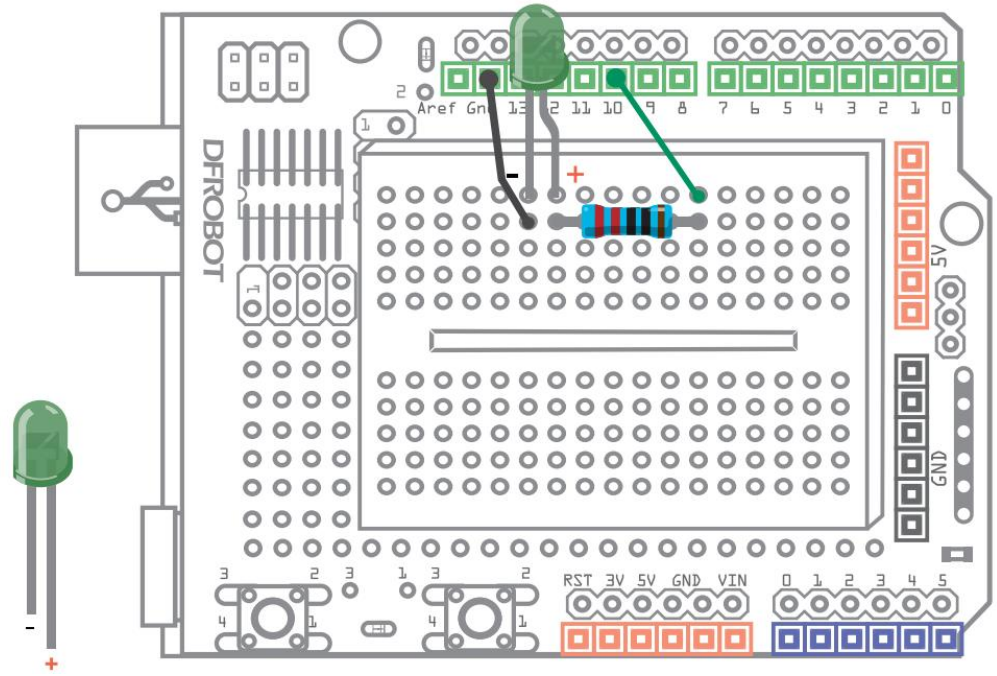


图 1 呼吸灯连线图

示例代码

样例代码:

```
//项目 - 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));
  }
}
```

代码上传完成后，我们可以看到 LED 会有个逐渐由暗到亮的一个缓慢过程，如同呼吸一般，均匀变化，而不是直接的亮灭。

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));
        }
}
```

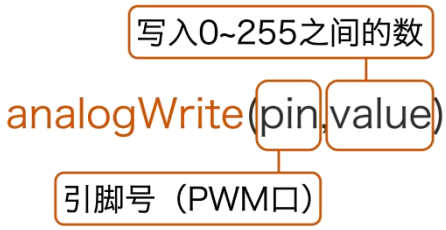
fadeOn()函数有两个传递参数，从参数名称中就可以简单看出，int time 指的是时间，int increament 指的是增量。函数中包含了一个 for 循环，循环条件是 value<255，变量的增量由 increament 决定。

for 语句中涉及了一个新函数：

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

要想将一个模拟值发送到数字引脚，我们需要使用到这个函数，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：

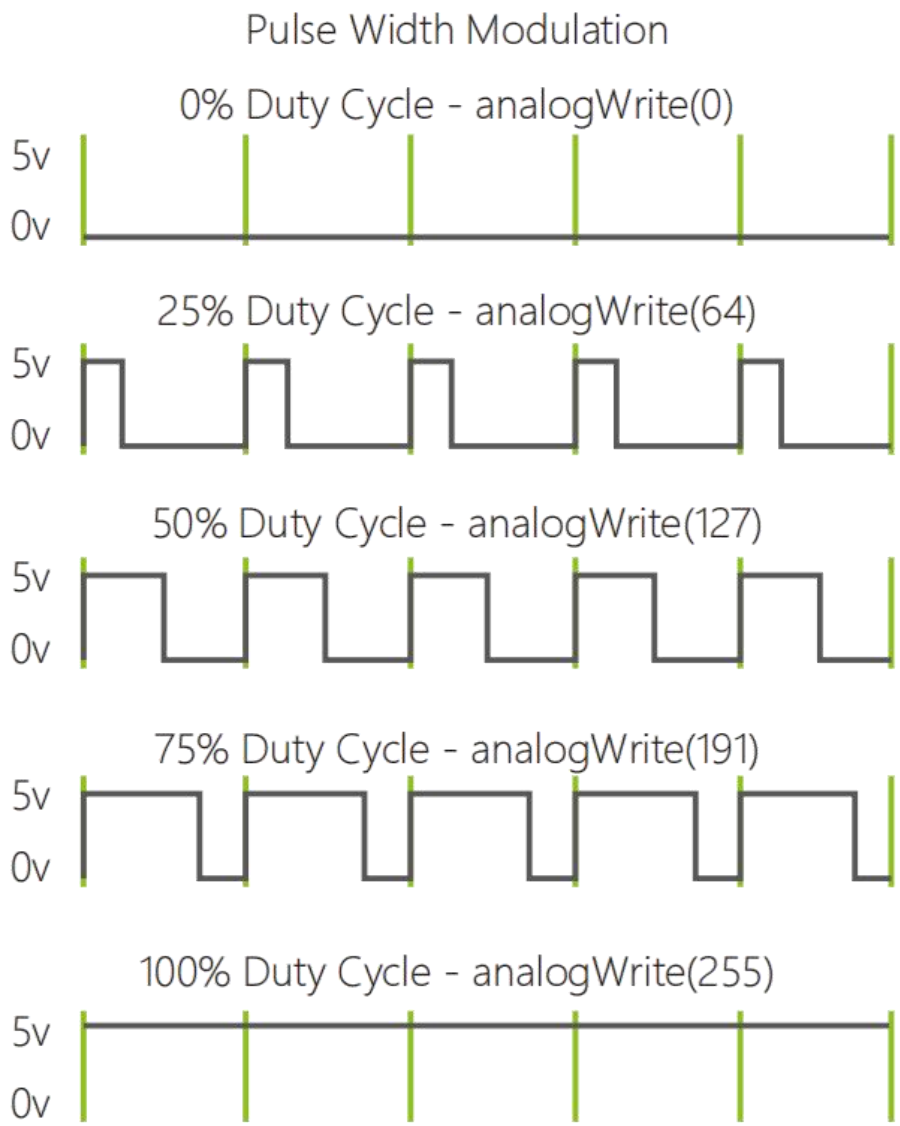


analogWrite()函数用于给 PWM 口写入一个 0~255 的模拟值。特别注意的是，analogWrite()函数只能对具有 PWM 功能的数字引脚进行模拟写入。

PWM

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

通过下面五个方波来更形象的了解一下 PWM。



上图绿色竖线代表方波的一个周期。每个 analogWrite(value)中写入的 value 都能对应一个百分比，这个百分比也称为占空比(Duty Cycle)，指的是一个周期内高电平持续时间比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%则更暗一些。

PWM 比较多的用于调节 LED 灯的亮度，或者是电机的转动速度。在玩一些 Arduino 小车时，更能体现 PWM 的好处，通过控制电机转动的速度来控制车速，还能通过控制左右电机的速度来控制转向。

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！

课后练习

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。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 以及三个 220 欧电阻。在这个练习中，推荐使用 random()函数来实现火焰不规则变化的效果。

random()函数的功能是产生一定范围内的随机数。

提示：可以先给 LED 设定一个基础亮度，在基础亮度值附近产生一个随机数，比如 random(-60, 60)+135，让其值稳定在 135 附近，并产生小幅变化，就更具有火焰跳跃感。不妨写程序来尝试一下。

具体用法可以查看下面链接的编程参考手册，里面会详细介绍这个函数的用法。之后的学习中，如果遇到了新的函数或者想尝试使用没有讲解过的函数，你可以通过编程参考手册来学习某个新函数。

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

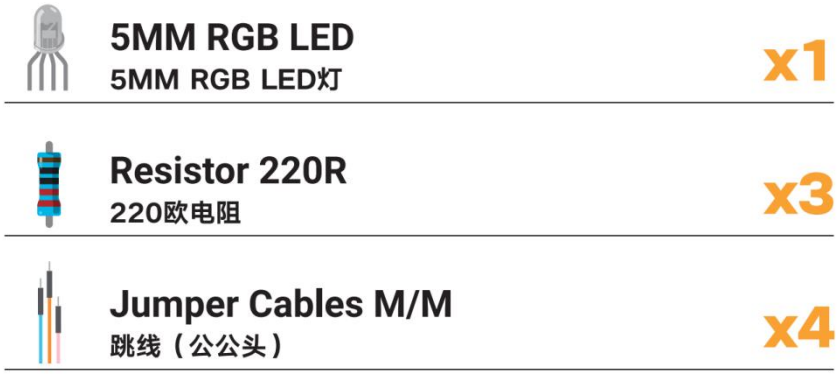
2、再尝试一个稍微有点难度的，通过两个按键，一个按键控制 LED 逐次变亮，另一个按键控制 LED 逐次变暗。

项目 - 炫彩 LED

这个项目介绍一种新的 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。找出所需元件，按下图连接硬件并输入代码。

元件清单



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，可参照左边的引

脚图。

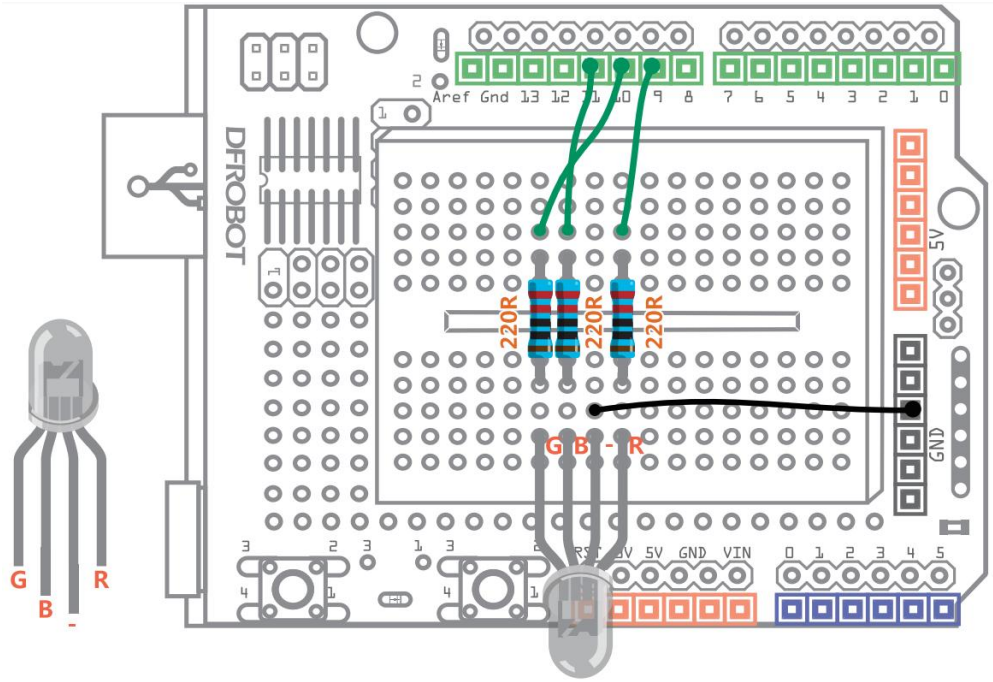


图 1 炫彩 LED 硬件连接图

示例代码

样例代码:

```
//项目 – 炫彩 RGB

int redPin = 9;

int greenPin = 10;

int bluePin = 11;


void setup(){

    pinMode(redPin, OUTPUT);
```

```
    pinMode(greenPin, OUTPUT);
    pinMode(bluePin, OUTPUT);

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

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 是随机数的最小值，max 是随机数的最大值。如果想知道 random()函数的其他用法，可以参看手册。

https://wiki.dfrobot.com.cn/Arduino%E7%BC%96%E7%A8%8B%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C#random

硬件回顾

RGB 灯

RGB 灯有 4 个引脚，R、G、B 三个引脚连接到信号传输口上，还有一个引脚是共用的正极（阳）或者共用的负极（阴）。我们这里选用的是共阴 RGB。图 2 展示了三个 LED 如何华丽蜕变为一个 RGB LED 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。

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，只需要将这三个颜色的 LED 分别进行控制即可。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 RGB

LED 调出任何你想要的颜色。

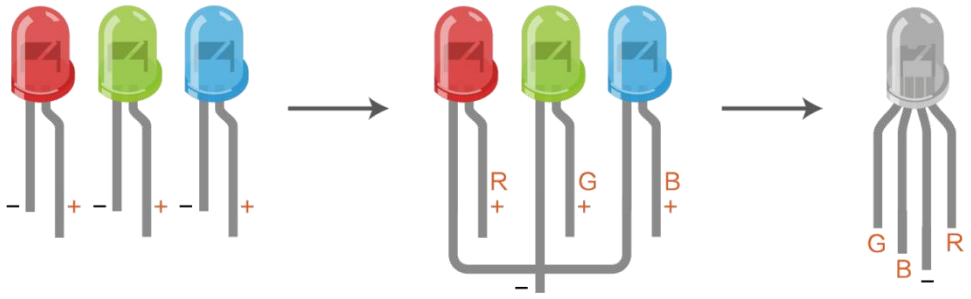


图 2 三个 LED 蜕变成一个 RGB 的过程

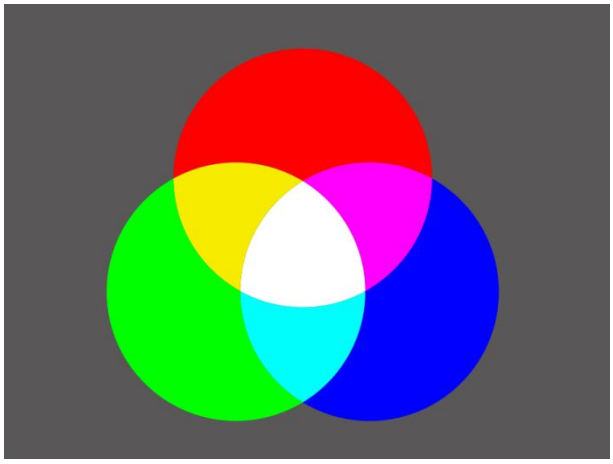


图 3 混合 RGB 产生不同颜色

表 1 只是展示了几种典型的混合颜色，可调的色彩远多于下表所示的，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 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上，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。

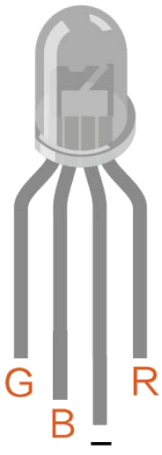


图 4 共阴 RGB 示意图

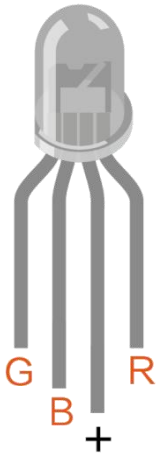


图 5 共阳 RGB 示意图

如何判断共阴还是共阳

其实非常简单，甚至都不需要编写程序来验证。前面讲过 RGB LED 其实是三个颜色的 LED。共阴和共阳的电路图如下：

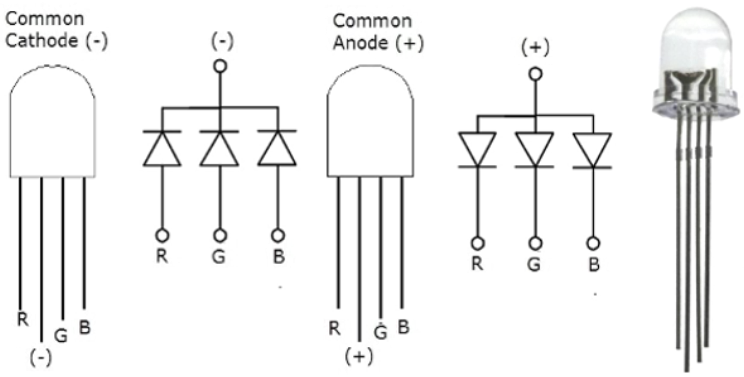


图 6 RGB LED 共阴共阳对比图

测试方法如下：

1. 在 R、G、B 三个引脚各串联一个 220Ω电阻。
2. 首先将公共端接入负极（接地）。
3. 逐一尝试将 R、G、B 引脚接入正极（如 3V 电源）。
4. 观察 LED 是否亮起对应颜色：若某引脚接入正极时亮起相应颜色（红、绿、蓝），则该 RGB LED 为共阴极；若均不亮，则判定为共阳极。
5. 为确保准确性，可按相同方法测试共阳极情况（即公共端接正极，逐一测试 R、G、B 引脚接地）

课后练习

1、基于我们上面的炫彩 RGB 项目，改变程序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、Green、Blue 的值 0~255，组合出你想要的颜色。

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 Arduino IDE 中就可以加载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。

比如我们今天使用的 RGB LED，就已经有现成的用户库了。

下面就提供一个加载 RGB LED 库和使用库的案例。

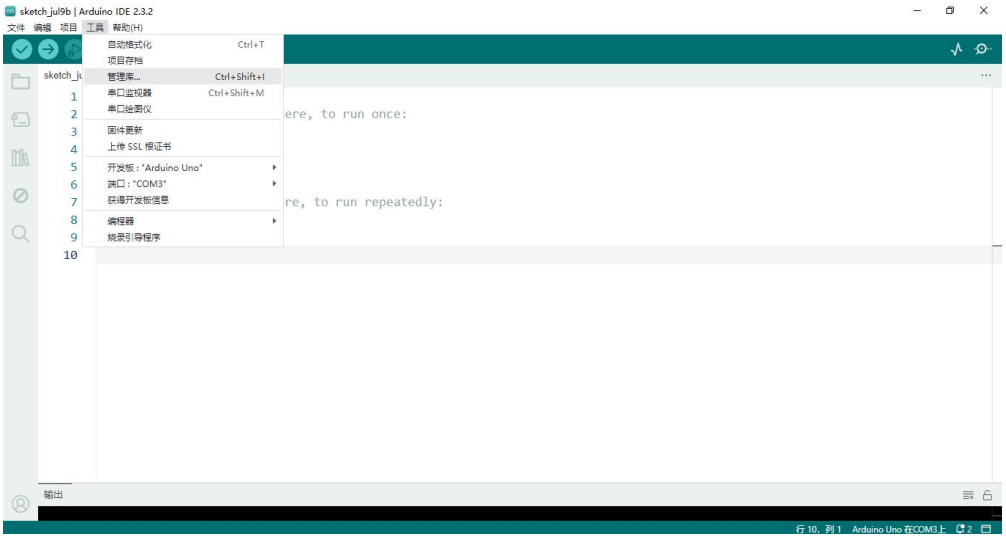


图 7 工具->管理库

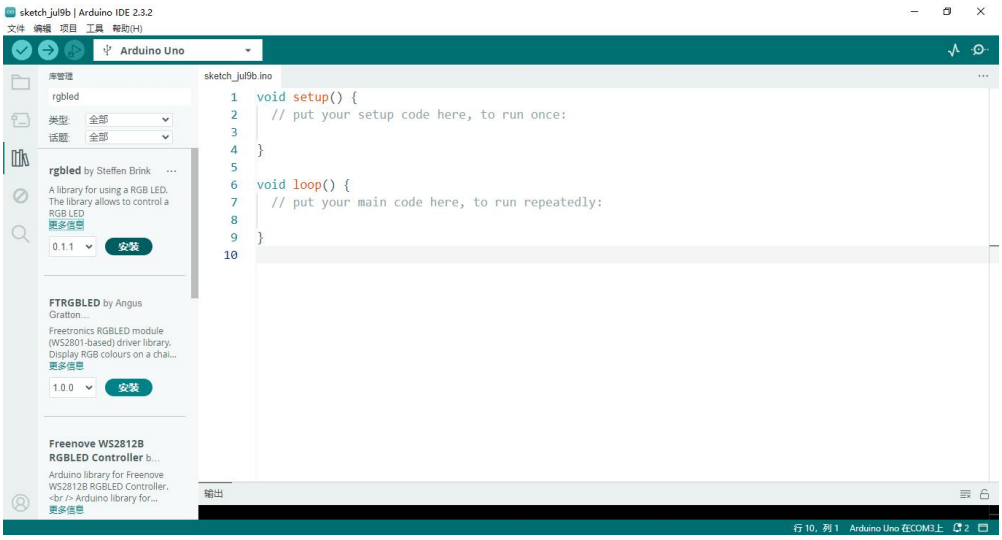


图 8 搜索 rgbled->安装

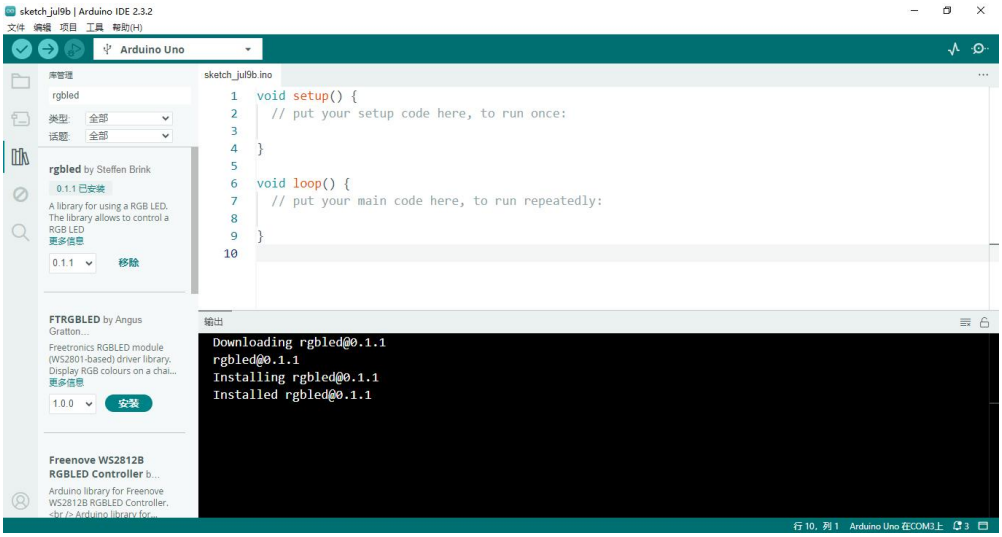


图 9 安装成功

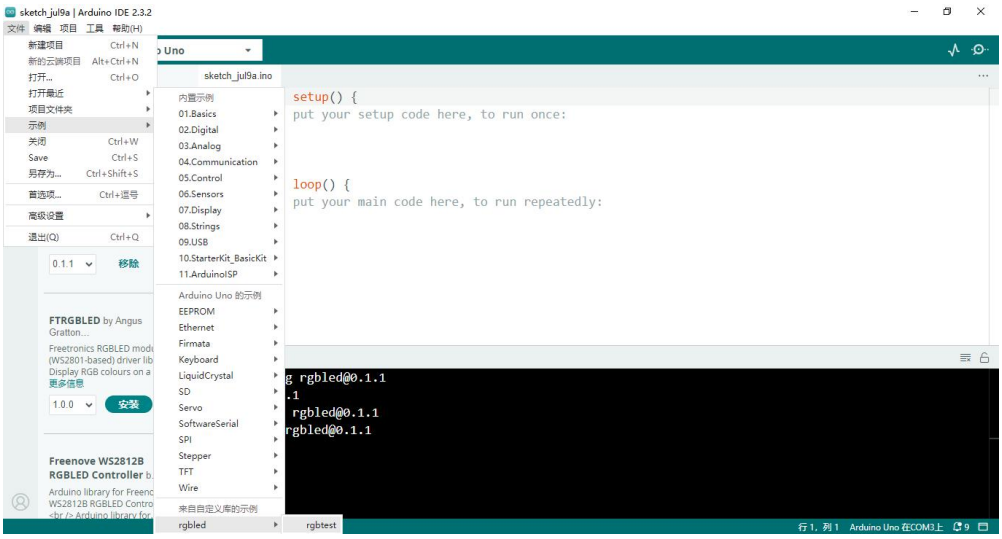


图 10 文件->示例->rgbled

图 11 验证和上传

项目 - 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。如果再结合一个 LED，配合同样的正弦波产生灯光的话，就是一个完整的声光报警器了。

元件清单

Buzzer
蜂鸣器 x1

Jumper Cables M/M
跳线（公公头） x2

硬件连接

按下图连接图连接，注意本项目蜂鸣器为无源蜂鸣器无正负之分。蜂鸣器的两个引脚分别接到 GND 和数字口 8。

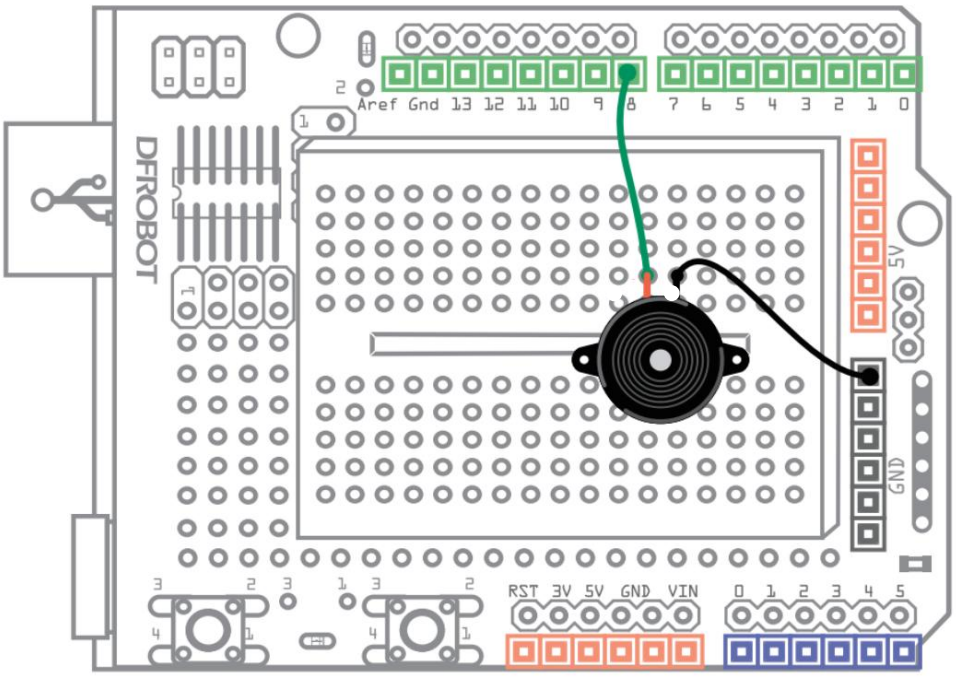


图 1 报警器连线图

示例代码

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

样例代码:

```
//项目 - 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)));
        //将 sin 函数角度转化为弧度
        toneVal = 2000+(int(sinVal*1000));
        //用 sin 函数值产生声音的频率
        tone(8, toneVal);
        //将 toneVal 给引脚 8
        delay(2);
    }
}
```

下载程序完成后，你会听到平滑的一高一低的报警声，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。如图 2，当弧度位于区间 0~π，也就是角度位于区间 0~180 度时，sin()值为正。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≤x < 180。

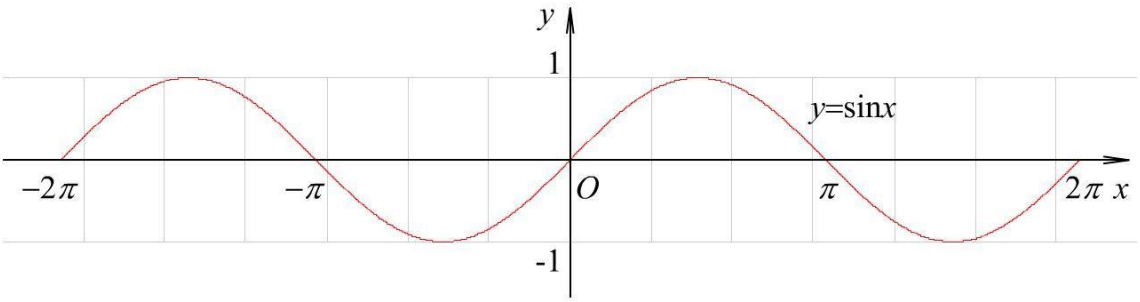


图 2 sin()函数

```
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)));
```

这样 sinVal 的值将会根据角度的变化产生像波浪一样的起伏，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，也就是下面这条语句就完成了这件事：

```
int toneVal;
```

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，但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。而无源内部不带振荡源，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。

在套件中，我们为初学者选用的蜂鸣器类型是无源蜂鸣器，可以演奏出不同的音乐效果。

蜂鸣器的应用有很多，我们也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?

课后练习

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【交通信号灯】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 - 温度报警器

在项目【报警器】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贴近生活的应用——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各种需要检测温度的场合。这个项目中，除了要用到蜂鸣器，还需要一个 LM35 温度传感器。

我们这里是第一次接触传感器，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，再将信息传递给大脑是一样的道理。

元件清单

Tem.Sensor
温度传感器 x1

Buzzer
蜂鸣器 x1

Jumper Cables M/M
跳线（公公头） x5

硬件连接

这个项目中的蜂鸣器和项目【报警器】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 和 GND，如下图所示。

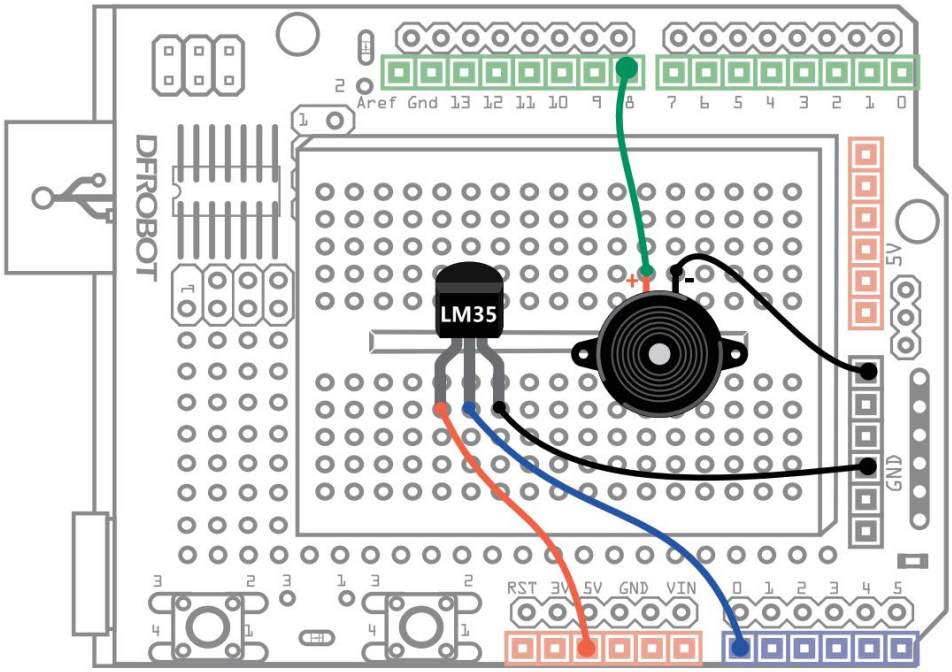


图 1 温度报警器连线图

代码示例

样例代码:

```
//项目 - 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 * (5/1024)*100; // 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。

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【报警器】的大部分内容是相同的，代码中的大部分语法在前几个项目中已经说过了，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【报警器】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【交通信号灯】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 传输的是个模拟量，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【SOS 求救信号器】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 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 * (5/1024)*100;
```

val：这是从温度传感器读取的模拟值，它介于 0 到 1023 之间。

(5/1024)：这个部分是一个比例因子。因为 1024 温度传感器能输出的最大数字值加 1（从 0 开始计数），所以 5/1024 表示每增加一个单位的模拟值所增加的电压值。val * (5/1024)代表现在温度对应的电压值，单位为 V（伏特）。

100：乘以 100 是为了将结果转换为温度。由于每 10mV（即 0.01V）对应一摄氏度，所以转换后的电压为 val(5/1024)/0.01（即*100），就能得到温度。

Arduino 的通信伙伴——串口

串口是 Arduino 与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 的串口分别与数字引脚 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){
        语句一;
    }
    } else {
          语句二;
    }
```

表达式结果为真时，执行语句一，放弃语句二的执行，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，执行语句二，放弃语句一的执行，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【报警器】：

```
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 ± 0.25℃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。其引脚分布如下图所示：

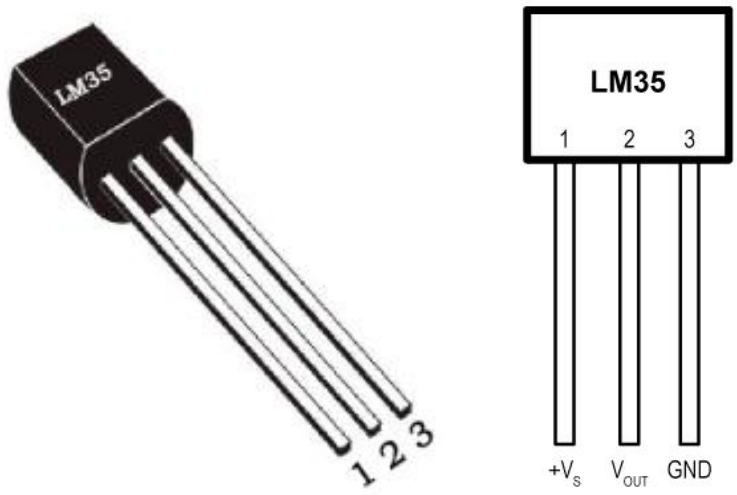


图 2 LM35 引脚示意图

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。

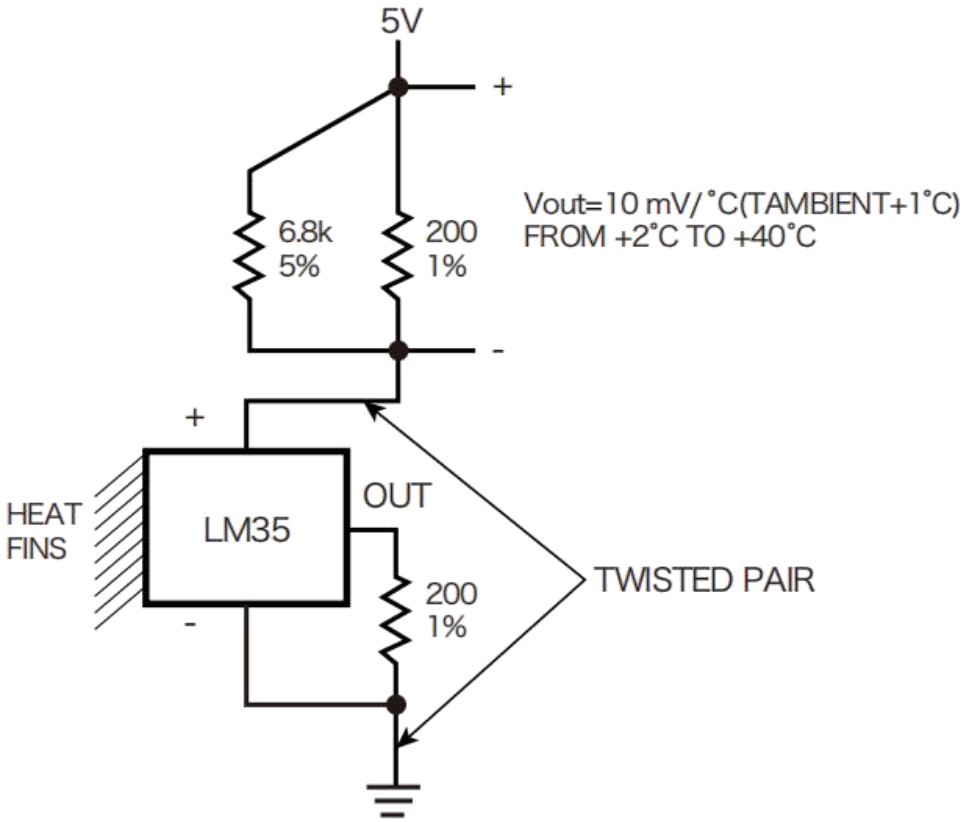


图 3 LM35 计算公式

课后练习

温度报警器可以用于监控对环境温度有要求的一些东西，比如植物。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。

发挥你的想象，看看还能玩出什么好玩的东西?

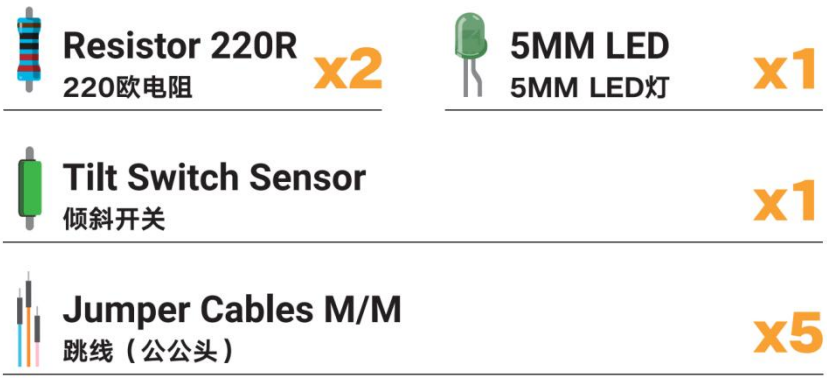
项目 - 震动探测

震动传探测，我们从名字中应该就可以判断，该装置能够检测物体是否在震动。我们用什么来做震动探测呢？那就是倾斜开关。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，我们想通过倾斜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 熄灭。

元件清单



硬件连接

从倾斜开关这个名字，我们可以把它和什么联想在一起？就是按键开关，倾斜开关和我们项目【交通信号灯】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。可以把下图与项目【交通信号灯】一起看，你会发现很多相似之处。倾斜开关也需要一个下拉电阻，LED 需要一个限流电阻。

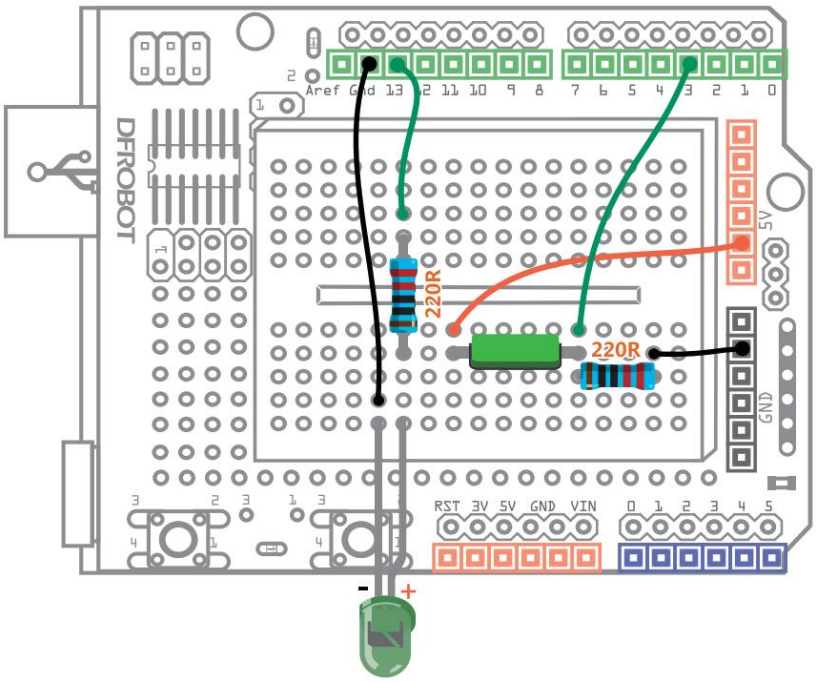


图 1 震动探测连线图

示例代码

样例代码:

```
//项目 – 震动探测

int SensorLED = 13;          //定义 LED 为数字引脚 13

int SensorINPUT = 3;         //连接倾斜开关到中断 1，也就是数字引脚 3

unsigned char state = 0;
```

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

  //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。

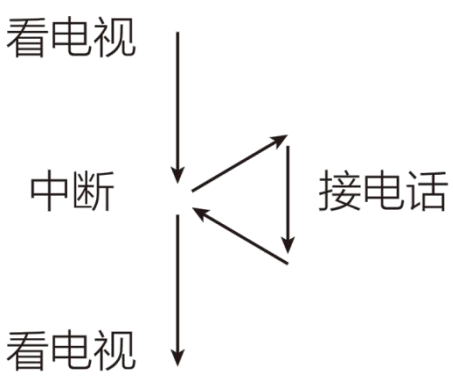
代码回顾

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。

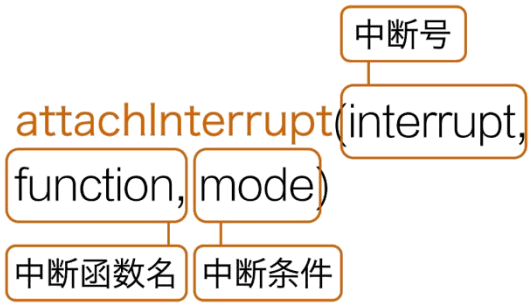
在没有任何打扰的情况下，程序在不断运行着……，让 LED 一直处于关闭。突然，被人打扰了（也就是晃动板子），就跳到中断函数 blink()中（当然进入中断也是要条件的，我们后面说）。此时，state 不断自加导致主函数中 if 函数检测到 state 不为 0 了，那么就让 LED 亮起了，同时又重新让 state 为 0，等待下一次中断。未发生中断，LED 则恢复关闭状态。

重复的知识点不再赘述，下面重点讲解中断函数 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吧，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，网址如下：http://arduino.cc/en/Reference/AttachInterrupt。



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，要写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。所以倾斜开关接到数字引脚 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，我们就把倾斜开关想象成按键。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。

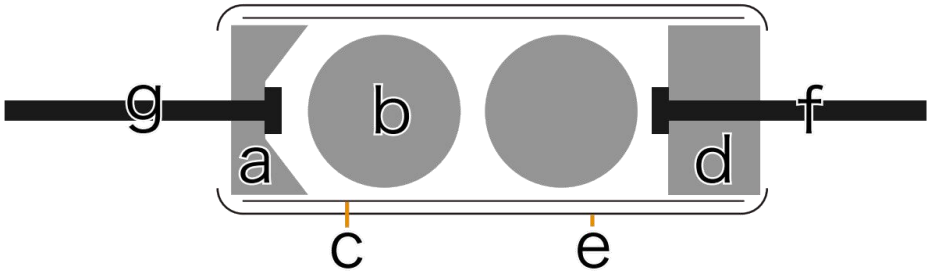
硬件回顾

倾斜开关

倾斜开关，也叫做滚珠开关，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。倾斜开关内部两个珠子，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，由于倾斜开关的内部构造，倾斜开关只有一头

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。



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

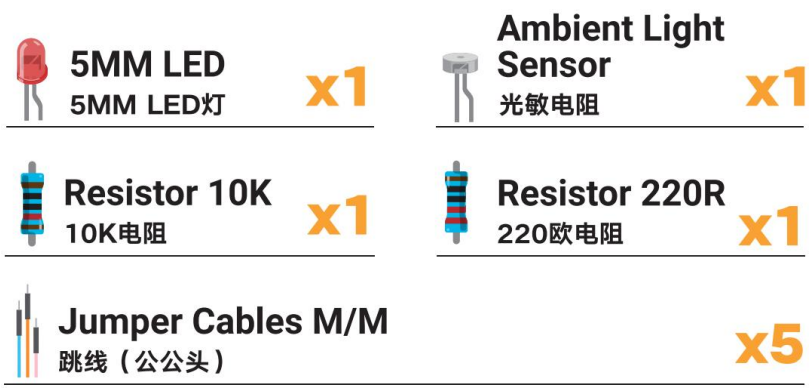
图 2 倾斜开关内部结构图

项目 - 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，这个元件是依赖光作用的。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。

元件清单



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（在下方连接图中左侧为负极，右侧为正极）。还需注意的与光敏二极管相连的电阻是 10k，而不是 220Ω。

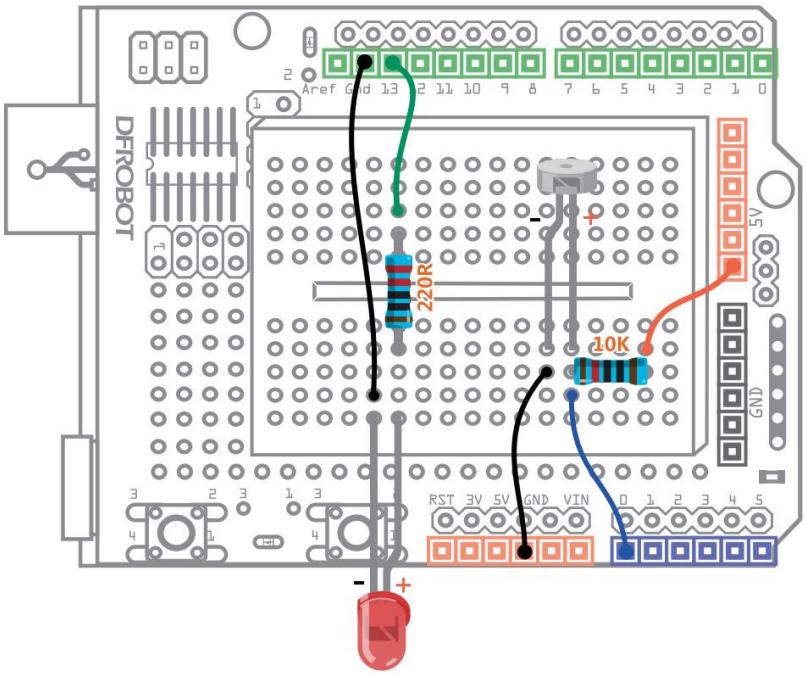


图 1 感光灯连线图

示例代码

样例代码：

```
//项目 – 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【温度报警器】中讲 LM35 温度传感器的时候，也用到了模拟口读取。强调了模拟量不需要设置 pinMode()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 2）

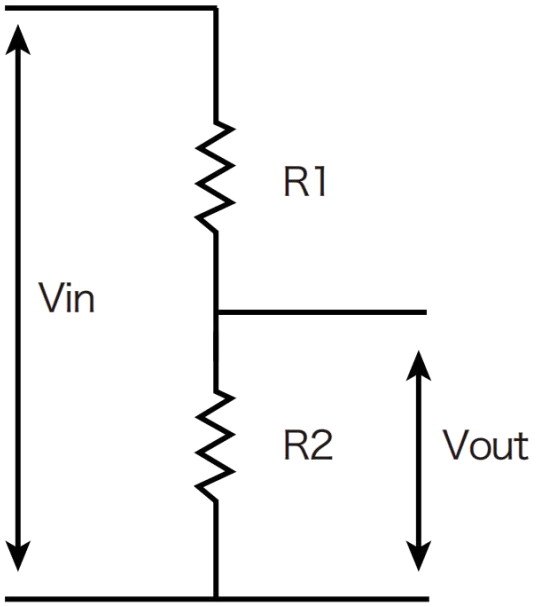


图 2 分压电路图

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 电压公式如下图所示。（图 3）

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

图 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 - 舵机初动

本项目要接触到舵机。舵机是一种电机，通过反馈系统来控制其位置，从而能够精确地掌握电机的角度。大多数舵机最大旋转角度是 180°。也有一些能转到 270°，甚至 360°。舵机常被用于对角度有精确要求的场合，如摄像头、智能小车的前置探测器，以及需要在特定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。

Arduino 也提供了<Servo.h>库，让我们使用舵机变得更方便了。

先从简单程序入手，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。

元件清单

Servo
舵机 x1

Jumper Cables M/M
跳线（公公头） x3

硬件连接

这个项目的连线很简单，只需按图 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。

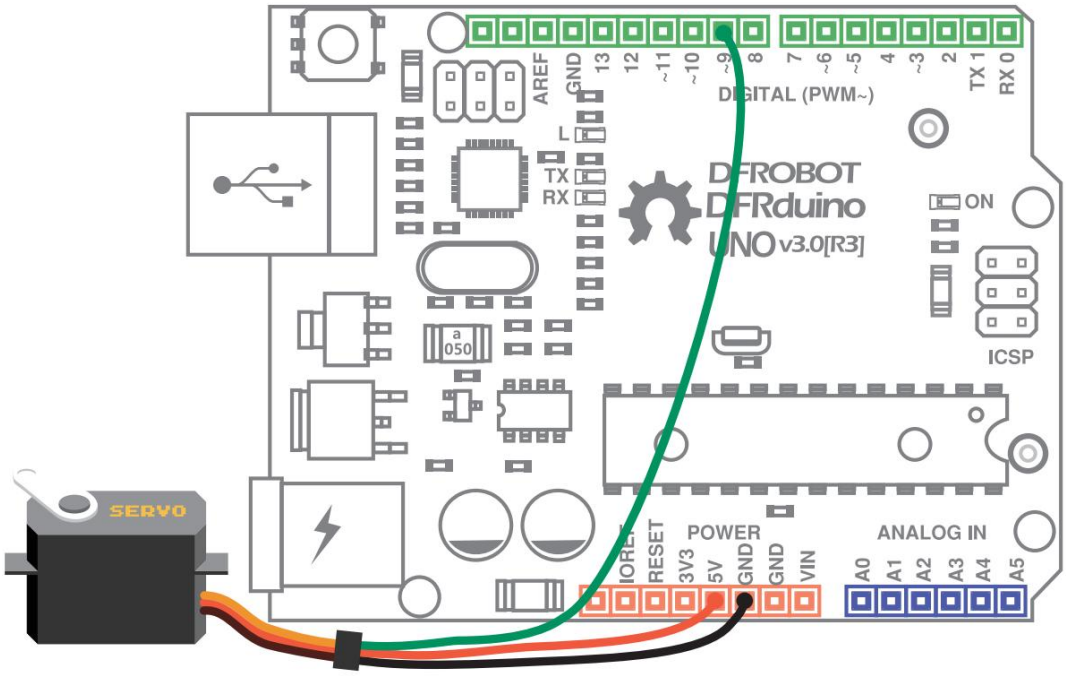


图 1 舵机初动连线图

代码示例

样例代码:

```
//项目 - 舵机初动

#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°回到 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

代码的开头首先调用了<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;
```

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)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。再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)，变量 pos 用来存放角度值。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。该函数的功能是将舵机转动到对应的角度，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

我们无需深入了解函数内部的复杂逻辑，只需学会如何正确使用即可。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 - 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。

元件清单

10K Potentimeter
10K 电位器 **x1**

Servo
舵机 **x1**

Jumper Cables M/M
跳线（公公头） **x6**

硬件连接

与项目【舵机初动】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。

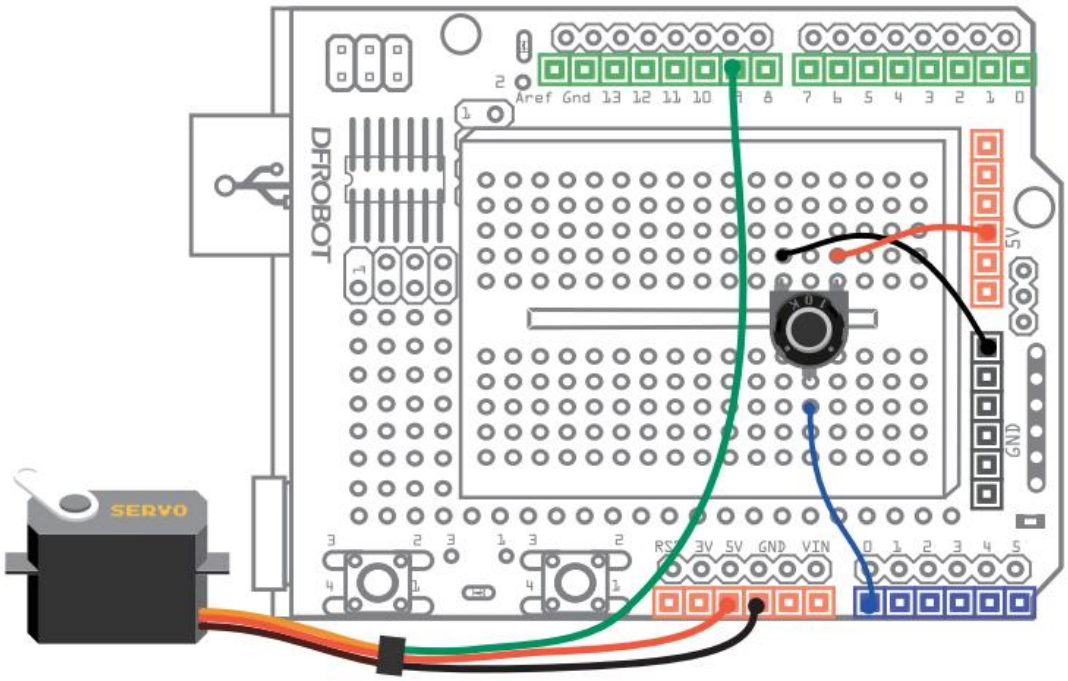


图 1 可控舵机连线图

代码示例

```
//项目 - 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:

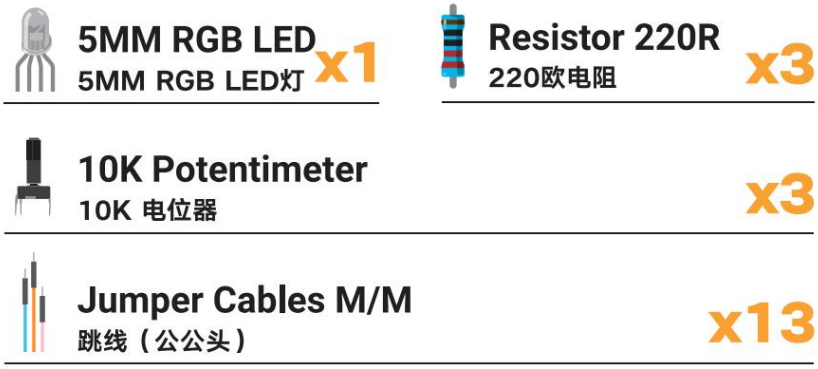
```
y = map(x, 1, 50, 50, -100);
val = map(val, 0, 1023, 0, 179);
```

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°。

项目 - 彩灯调光台

在项目【炫彩 LED】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。

元件清单



硬件连接

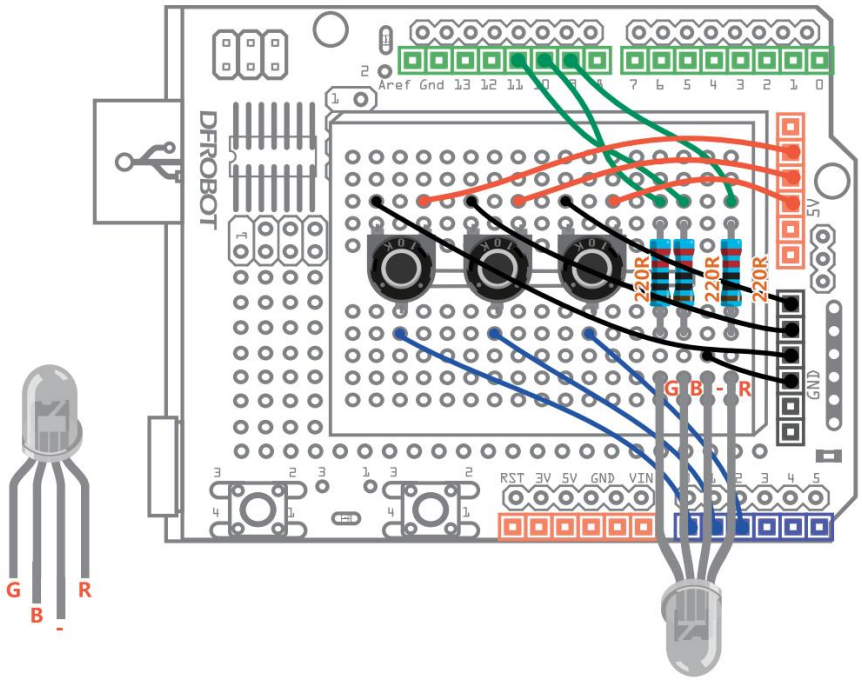


图 1 彩灯调光台连线图

代码示例

样例代码:

```
//项目 -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));
}
```

上传代码，成功后，通过旋转三个电位器， 看看 RGB 灯是否会随之变化。

项目 - 自制风扇

这次，我们会做一个用按钮控制小风扇。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。

元件清单

硬件连接

按下图进行连线，按钮接线与项目【交通信号灯】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。

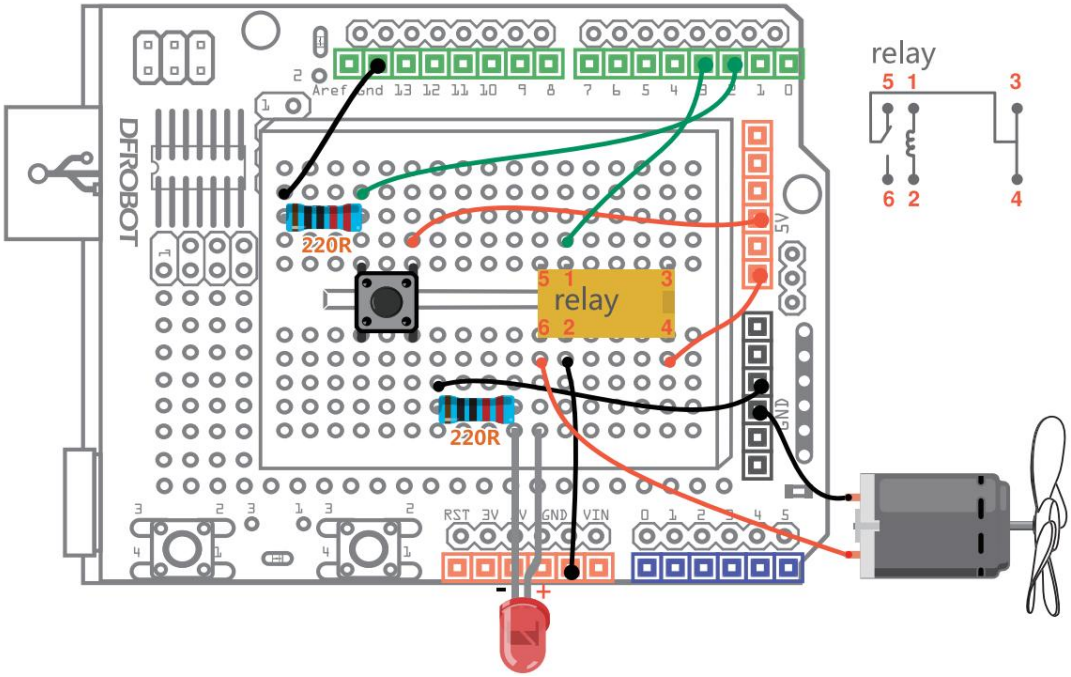


图 1 自制风扇连线图

代码示例

Sample code:

```
//项目 – 自制风扇

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;
}
```

代码回顾

代码的大部分内容，基本应该没有什么难度了，主要说下按键去抖的代码进行说明。

去抖代码如下：

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

```
      if (reading != buttonState) {
        ...
      }
    }
```

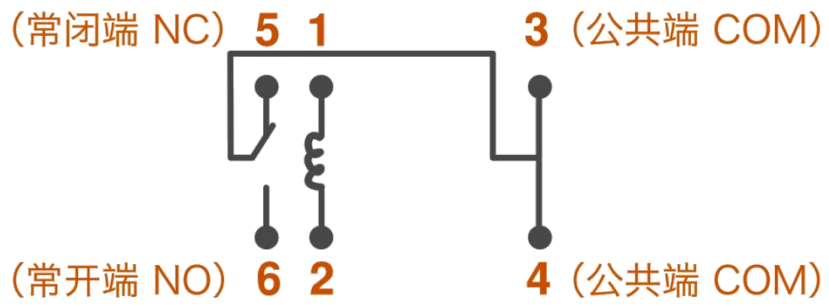
reading 有变化之后，不是立马就采取相应的行动，而是先“按兵不动”，先看看这个信号是不是“错误信号”，所以再等待一阵，（也就是通过 millis 来实现这个等待过程的），发现确实是前方发过来的正确信号，然后执行相关动作。这么做的原因是，按键在被按下时，会有个抖动的过程，而不是立马由低变高，或者由高变低。所以这个过程中，可能会产生错误信号，我们通过程序中的这种方法，来解决硬件上的这个问题。

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，而是选用需要 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。

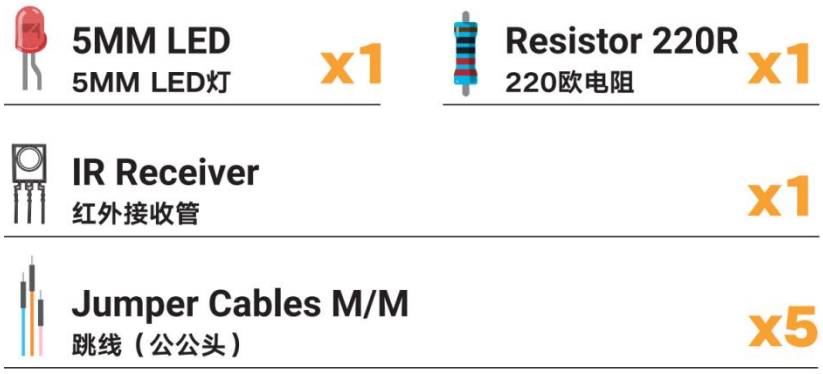
舵机也是一种电机，它使用一个反馈系统来控制电机的位置，可以用来控制角度。所以，舵机经常用来控制一些机器人手臂关节的转动。

项目 - 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。比如我们的电视机，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。

元件清单



硬件连接

红外接收模块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。在这个基础上，加一个 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（LED 灯在该项目的后半部分会用上）。

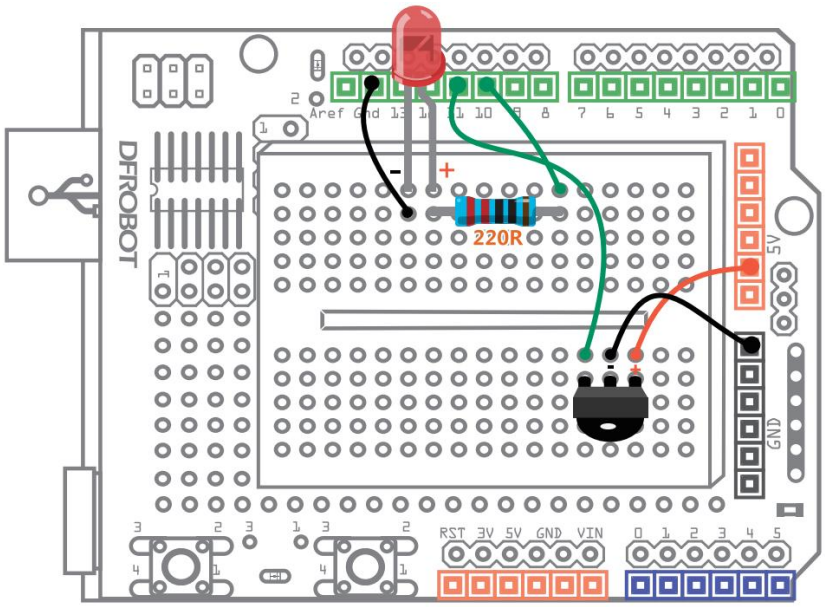
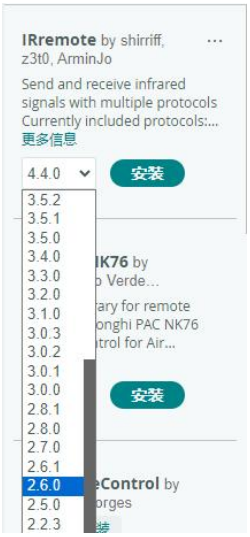


图 1 红外遥控接收器连线图

代码示例

对于遥控红外模块，我们提供现成的 IRremote 库，直接在 Arduino IDE 中安装 IRremote 库（需使用该库的 2.6.0 版本，如图 2），输入样例代码 1 运行即可。如果还是不是很明白如何加载库，可以回看一下项目【炫彩 LED】课后作业部分，对如何加载库做了详细说明。

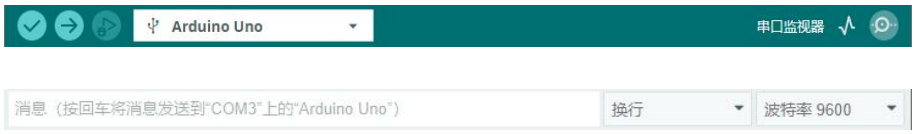


样例代码 1:

```
//项目 - 红外遥控灯测试代码

#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。

如果长按一个键不放就会出现“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。

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，观察下相对样例代码 1 增加了哪些内容。

样例代码 2:

```
#include <IRremote.h>              //调用 IRremote.h 库
int RECV_PIN = 11;                 //定义 RECV_PIN 变量为 11
int ledPin = 10;                   // LED - digital 10
boolean ledState = LOW;            // ledstate 用来存储 LED 的状态
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
  pinMode(ledPin,OUTPUT);       // 设置 LED 为输出状态
}

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);

      //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();  // 继续等待接收下一组信号
   }
}
```

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
int ledPin = 10;               // LED - digital 10
boolean ledState = LOW;        // ledstate 用来存储 LED 的状态
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
```

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();
```

课后练习

1、通过这个遥控项目，再结合项目【自制风扇】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 - 红外遥控数码管

数码管，常见的用来显示数字的，像计算器屏幕上显示的数字就是一种数码管。在本项目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。现在，让我们通过完成项目的方式来感受一下数码管的神奇之处吧！

元件清单

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。下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。

我还特意在连线图上，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。红外接收模块连线如图所示，如果有不清楚的地方请参考项目【红外遥控灯】。

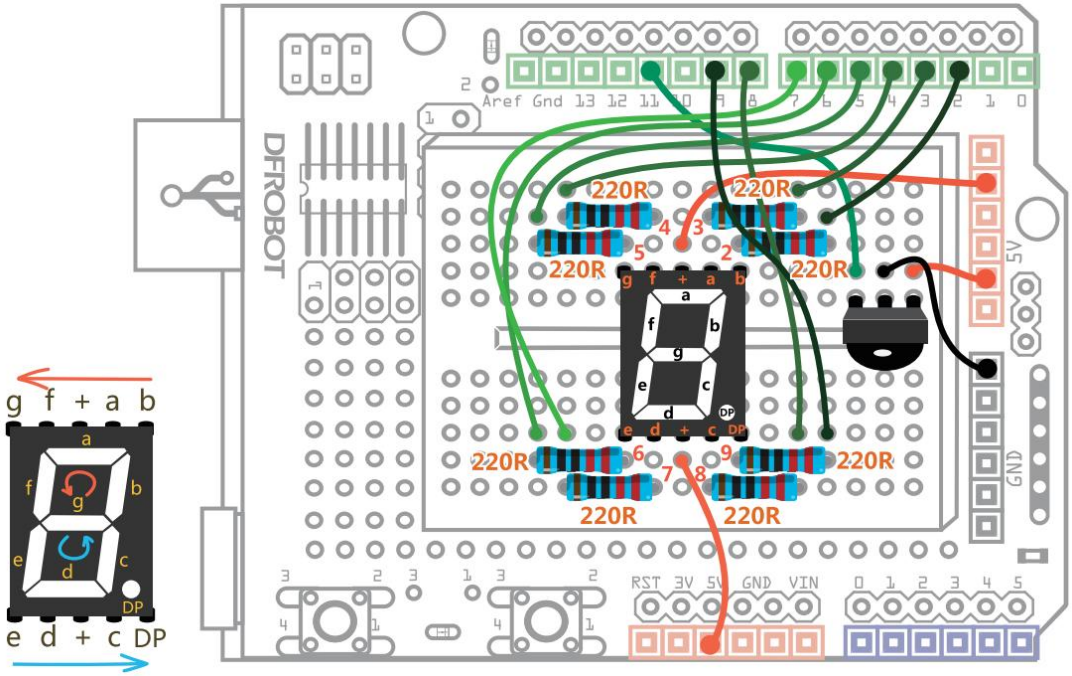


图 1 红外遥控数码管连线图

示例代码_1

样例代码 1:

```
//项目 - 红外遥控数码管_1

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

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

    // 显示数字 2
```

```
int n2[8]={0,0,1,0,0,0,1,1};
for(int pin = 2; pin <= 9 ; pin++){
// 数字引脚 2~9 依次按数组 n2[8]中的数据显示
   digitalWrite(pin,n2[pin-2]);
}
delay(500);
// 显示数字 3
int n3[8]={0,0,1,0,1,0,0,1};
for(int pin = 2; pin <= 9 ; pin++){
// 数字引脚 2~9 依次按数组 n3[8]中的数据显示
   digitalWrite(pin,n3[pin-2]);
}
delay(500);

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

// 显示数字 5
```

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

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

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

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

    // 显示数字 9
    int n9[8]={0,0,0,0,1,0,0,1};
    for(int pin = 2; pin <= 9 ; pin++){
    //  数字引脚 2~9 依次按数组 n9[9]中的数据显示
       digitalWrite(pin,n9[pin-2]);
    }
    delay(500);
}
```

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

硬件回顾

数码管其实就是一个前面介绍的 LED 的组合体，这个组合体包含 8 个 LED，所

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

下面图 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。

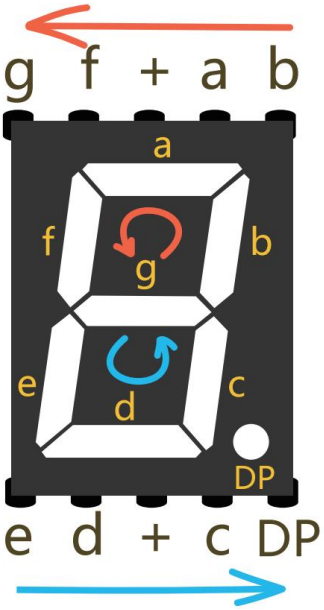


图 2 引脚连接说明图

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？另外两个引脚是公共端，LED 只有一端是不能被点亮的。我们在项目【炫彩 LED】中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。现在你对

硬件有所了解了，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，控制另一端数字引脚为 HIGH，才让数码管点亮。

代码回顾_1

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

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

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

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

```
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++){
// 数字引脚 2~9 依次按数组 n0[8]中的数据显示
      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。

如果样例代码 1 弄明白后，我们要教大家一种更简单的方法，继续看下方的样例代码 2。

示例代码_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！

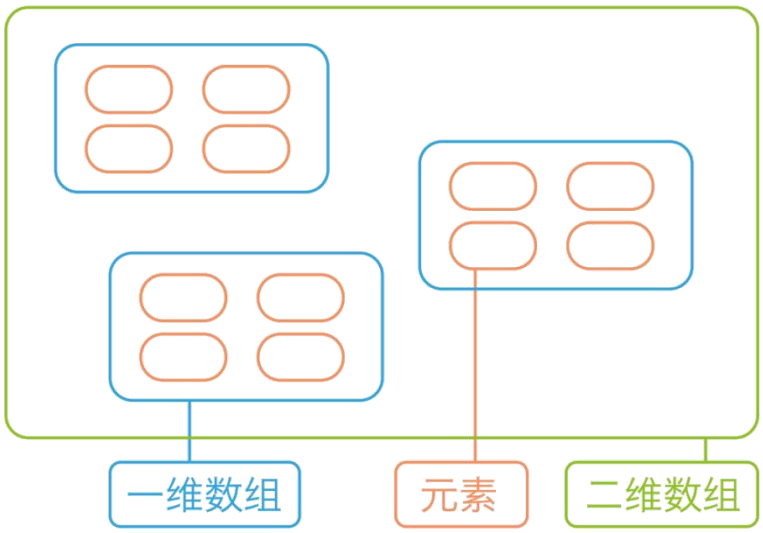
样例代码 2:

```
 //项目 - 红外遥控数码管_2
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
  {0,0,0,0,1,0,0,1}    //显示 9
};
```

```
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 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

对比一下样例代码 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。



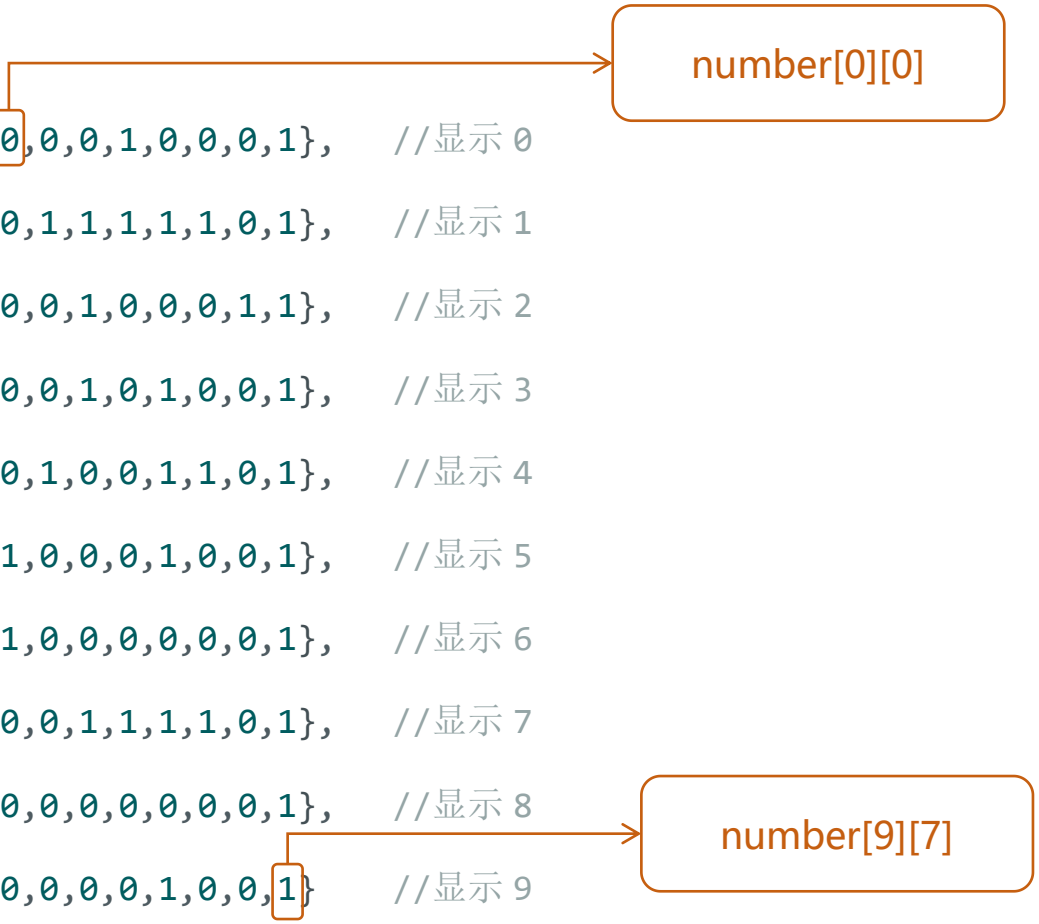
通过上面这个图，元素、一维数组、二维数组之间的关系就一目了然了。一维数组由元素组成，而二维数组则是由一个个一维数组组成的，关系就是那么简单。

样例代码 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只是元素个数。

看一下代码：

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



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

```
void numberShow(int i){          //该函数用来显示数字
    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”。

回顾一下样例代码 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};
for(int pin = 2; pin <= 9 ; pin++){
// 数字引脚 2~9 依次按数组 n0[8]中的数据显示
   digitalWrite(pin,n0[pin-2]);
}
```

原理是一样的，通过给引脚 2~9 循环赋值，控制数码管 b~DP 段亮灭，就能显示出一个我们想要的数字。numberShow(0)循环完后，再次回到 loop()中的 for 函数：

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

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

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

……

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

好了，这就是整段代码的分析过程，好好体会一下一维数组和二维数组的区别，以及整段代码如何巧妙运行的。了解了数码管和红外接收管各自的工作原理后，我们需要把这两者结合起来，看看红外接收管和数码管结合能迸发出怎样的火花呢？想到了吗？遥控数码管！Arduino 控制器把红外接收管从 Mini 遥控器那儿接到的信号，经处理传达给数码管。让 Mini 遥控器上 0~9 对应在数码管上显示 0~9，除此之外，还有递减，递增的功能。

示例代码_3

样例代码 3:

```
//项目 - 红外遥控数码管_3
#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
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,               // 前进 ,后退
};

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
    {0,0,0,0,1,0,0,1} //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
  }
}
```

上传完代码后，尝试按下图 3 指出部分的按键，看看是数码管是个怎样的变化。

图 3 遥控器按键说明

代码回顾_3

程序一开始还是对红外接收管的一些常规定义，按项目【红外遥控灯】搬过来即可（注意：IRremote 库选择 2.6.0 版本）。

```
#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
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，数字递增递减时能找到

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键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。我们在代码回顾_2 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，找到对应按键后，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，这段代码应该是所有项目中最复杂的代码，可能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。

课后练习

学习了如何使用遥控功能，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，发挥你的想象做出更多 Arduino 作品吧！