

脑震荡



Arduino 入门学习套件

配套教程26课



Ver 1.0 Rev 3 - 7.28.2017

©2017 脑震荡版权所有

关于脑震荡

脑震荡是关注科技、艺术、制作 (Technology, Art, Making) 的小众媒体。我们倡导“一本正经地玩”，致力于用好玩的方式介绍工具、知识和信息，使更多的人可以释放创造力和动手能力，成为具有独立精神的Maker。

网址：<http://www.naozhendang.com>

知乎专栏：<https://zhuanlan.zhihu.com/naozhendang>

淘宝店铺：<http://naozhendang.taobao.com>



关注公众号



更多套件淘宝扫一扫

目录

01	初识Arduino	3
02	熟悉Arduino硬件	7
03	安装和熟悉Arduino IDE	13
04	面包板的使用	24
05	经典仪式点亮LED	29
06	S.O.S求救信号灯	37
07	用按钮控制LED	41
08	按钮去抖Debounce	47
09	互动交通信号灯	51
10	PWM和呼吸灯	57
11	控制多个LED	63
12	光敏电阻和感光灯	69
13	RGB LED	74
14	库的概念和如何使用扩展库	80
15	玩转LCD显示	87
16	电位器和舵机	92
17	步进电机的控制	100
18	串口监视器的使用	108
19	红外遥控灯	117
20	震动传感器	122
21	温度报警器	127
22	用蜂鸣器制造音乐	133
23	超声波特雷门琴	138
24	8x8LED矩阵怦然心动	145
25	数码管计时器	151
26	学习方法建议和Arduino的未来	158

01 初识Arduino

Arduino是什么？

Arduino是一个比你的台式电脑更能够用来感应和控制现实物理世界的一套工具。它由一个基于简易单片机并且开放源码的计算机平台，和一套为Arduino板编写程序的开发环境组成。

Arduino能通过各种各样的传感器来感知环境，通过控制灯光、马达和其他的装置来反馈、影响环境。板子上的微控制器可以通过Arduino的编程语言来编写程序，编译成二进制文件，烧录进微控制器。对Arduino的编程是利用 Arduino编程语言(基于 Wiring)和Arduino开发环境(基于 Processing)来实现的。基于Arduino的项目，可以只包含Arduino，也可以包含Arduino和其他一些在PC上运行的软件，他们之间进行通信(比如 Flash, Processing, MaxMSP)来实现。

Arduino能做什么？

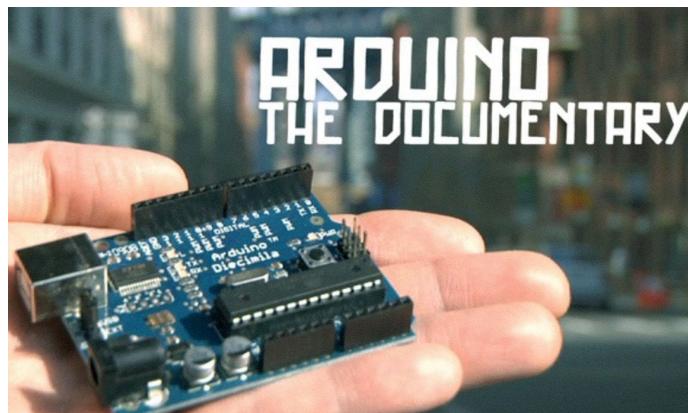
从小梦想做一台遥控小车，但一直没有机会开始?
每天睡觉都懒得去关灯，如何做到够声控关灯?
想要做可以跟观众互动的艺术作品?
参加化妆舞会，想要做一把亮瞎狗眼的绝地武士激光剑?
想要一台能检测心跳的可穿戴设备?
买不起大疆，自己做一台四轴飞行器?
想要学会从零开始DIY一台3D打印机?

还有很多很多很多.....

Arduino的前世今生

Arduino纪录片

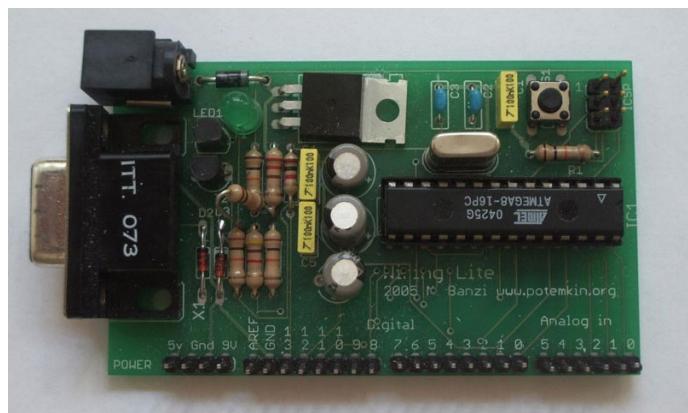
不用长篇累牍的介绍，2010年有一部近半小时的《Arduino The Documentary》纪录片，以访谈形式正能量地呈现了Arduino的诞生和发展。



视频链接：

http://v.youku.com/v_show/id_XMTQ0NDc5MjM0MA==.html

关于Arduino到底是怎么诞生的有很多版本以及师生队友撕逼的细节，这些在此并不是重点。公认的是，Arduino于2004和2005期间诞生于意大利Ivrea小镇一所叫 Interaction Design Institute Ivrea (IDI) 交互设计学院里。初衷是为了给学校的非电子工程背景的设计和艺术类学生提供一套简单、便宜又易用的工具来更好地学习和理解电子技术，并创作。



Arduino一代原型机Wiring Lite

相比当时价格较高的BASIC Stamp和其他平台，Arduino在硬件上极大地集成和简化了电路模块一些繁琐的技术细节，降低成本和提供扩展性，同时在软件上提供了一套面向非电子专业人士的跨平台编程环境，简易的语法和用户界面让初学者很容易上手。很快，Arduino平台在其他学校以及业余电子爱好者中流传开来。软硬件的开源特性吸引了大批开发人员和爱好者为社区添砖加瓦。Arduino核心团队也陆续推出了适合不同情况下使用的各类Arduino变种：例如Mega, Nano, Mini, LilyPad等等。

2011年至今，Arduino伴随着Maker Movement (创客运动)，和Open Hardware Movement (开源硬件运动) 等思潮运动，量变到质变，着实火了。和3D打印，无人机等名词一起成为这些思潮运动的代言标签。这种关系好比当年LSD迷幻剂之于美国60年代末的Counterculture反文化运动。



图片为2015年UK Maker Faire现场

尽管相继有无数的效仿者，有的宣称拥有比Arduino有更强劲的性能，有的则有更小的体积，但至今都还没有一个可以在流行度和社区规模上超越Arduino的平台。如今的Arduino，俨然是一种“经典”，一种精神代表，而不仅仅是一个单片机平台。经典不是拿来超越和打败的，而是用来膜拜的。当很多初学者问为什么大家都推荐先入门Arduino而不是其他平台？答案就是经典。可以说，学习Arduino，不单是掌握一项技能，更可以接触到开源

硬件的经典范式，理解Maker Movement背后的精
神追求。

“ If the only tool you have is a hammer, to
treat everything as if it were a nail. 当你手中唯
一的工具是一把锤子的时候，世界上所有的东西，
在你看来都变成了铁钉。”

正确地认识工具

当然作为工具，Arduino也有它自身的局限性。它
不是万能的，学会了Arduino并不意味着你能实现
所有的想法。一方面，Arduino的确能帮你在短时
间里实现某个想法的初期原型，扩增了使用者的
能力，但另一方面，它也在体积、性能和思维方式
诸多方面限制住使用者的创造力。



图为Arduino Yún

在学习Arduino使用中，不单要知道它的长处也要
了解它的不足。比如Arduino族群中大部分都拥
有有限的计算能力，比较适合作为低成本的感应
端或者执行端；大部分也不是针对处理网络方案
而生的。所以当你的项目需要较强的计算和网络
处理能力时，你可以果断考虑换成类似Raspber-
ry Pi, Intel Edison或是Arduino族群中的Arduino
Due, Arduino Yún。

总而言之，Arduino是一款典型的“低门槛，宽边
界，高天花板”学习平台。非常适合初学者入门，当
你到达一定的水平后又可以无缝地深入学习更底
层更专业的知识或者其他平台。在下一篇教程中，
脑叔将带着大家熟悉Arduino的各款硬件开发板，
以及如何挑选一款适合自己需求的Arduino板。

相关链接

- Arduino官网：<http://arduino.cc>
- 《Arduino The Documentary》记录片的Vimeo链接：<https://vimeo.com/18539129>
- Wiring作者Hernando Barragán关于Arduino旧史的爆料，有兴趣的童鞋可以深入八卦一下：《The Untold History of Arduino》
<http://arduinohistory.github.io/#fn:2>
- Wikipedia上关于IDII的介绍，这是一所值得朝圣的学
校：https://en.wikipedia.org/wiki/InteractionDesignInstitute_Ivrea
- Raspberry Pi官网：<https://www.raspberrypi.org>
- Intel Edison官网：<http://www.intel.com/content/www/us/en/do-it-yourself/edison.html>

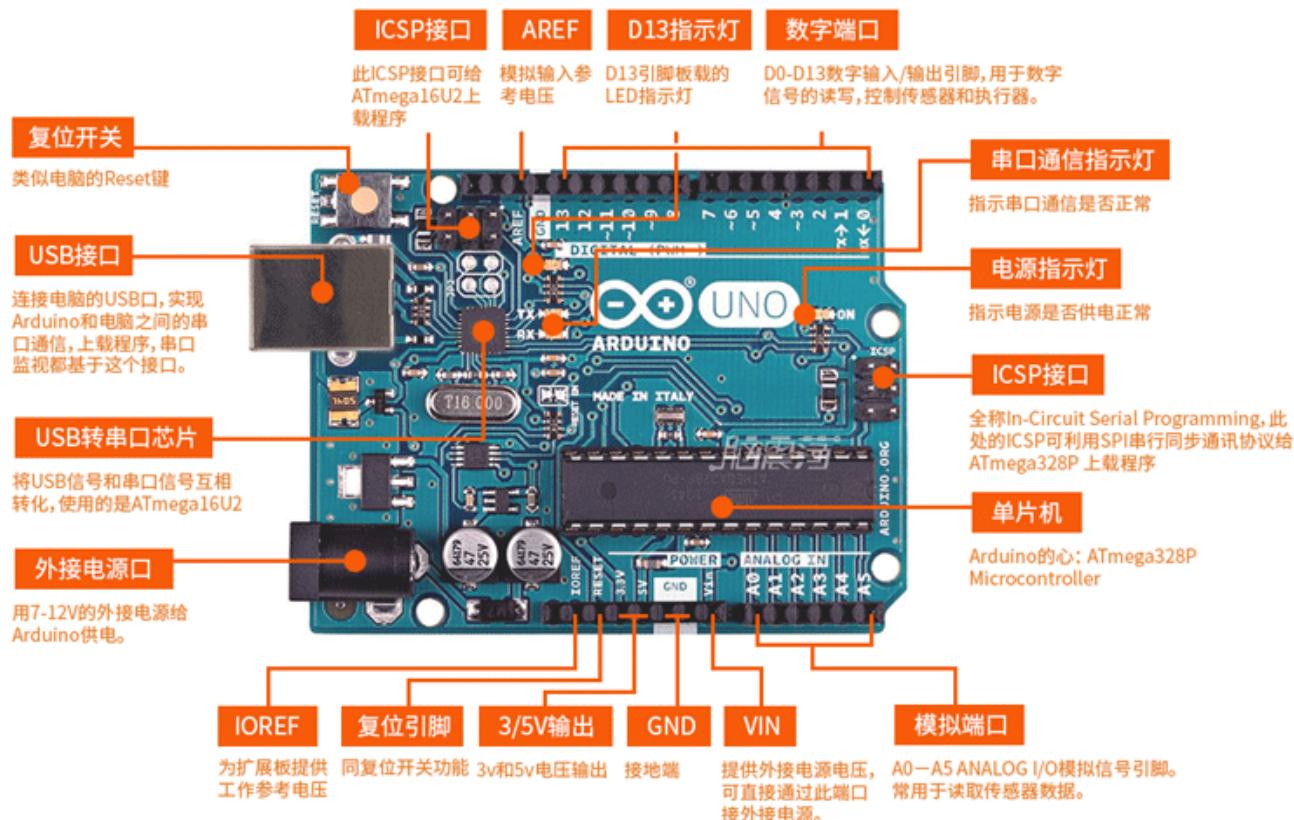
02 熟悉Arduino硬件

UNO开发板的硬件构成

大部分版本的Arduino开发板都是基于Atmel的AVR系列单片机，然后板上集成了一些电源管理，串口通信，然后基本的辅助支持电路。 Arduino UNO是Arduino众多版本中最常见的版本。

Arduino不同版本之间的区别，简而言之就是AVR单片机所用的芯片型号不一样，或者集成了不同的特殊的通信功能模块，抑或者为了减少空间省略了电源模块和一些支持电路。

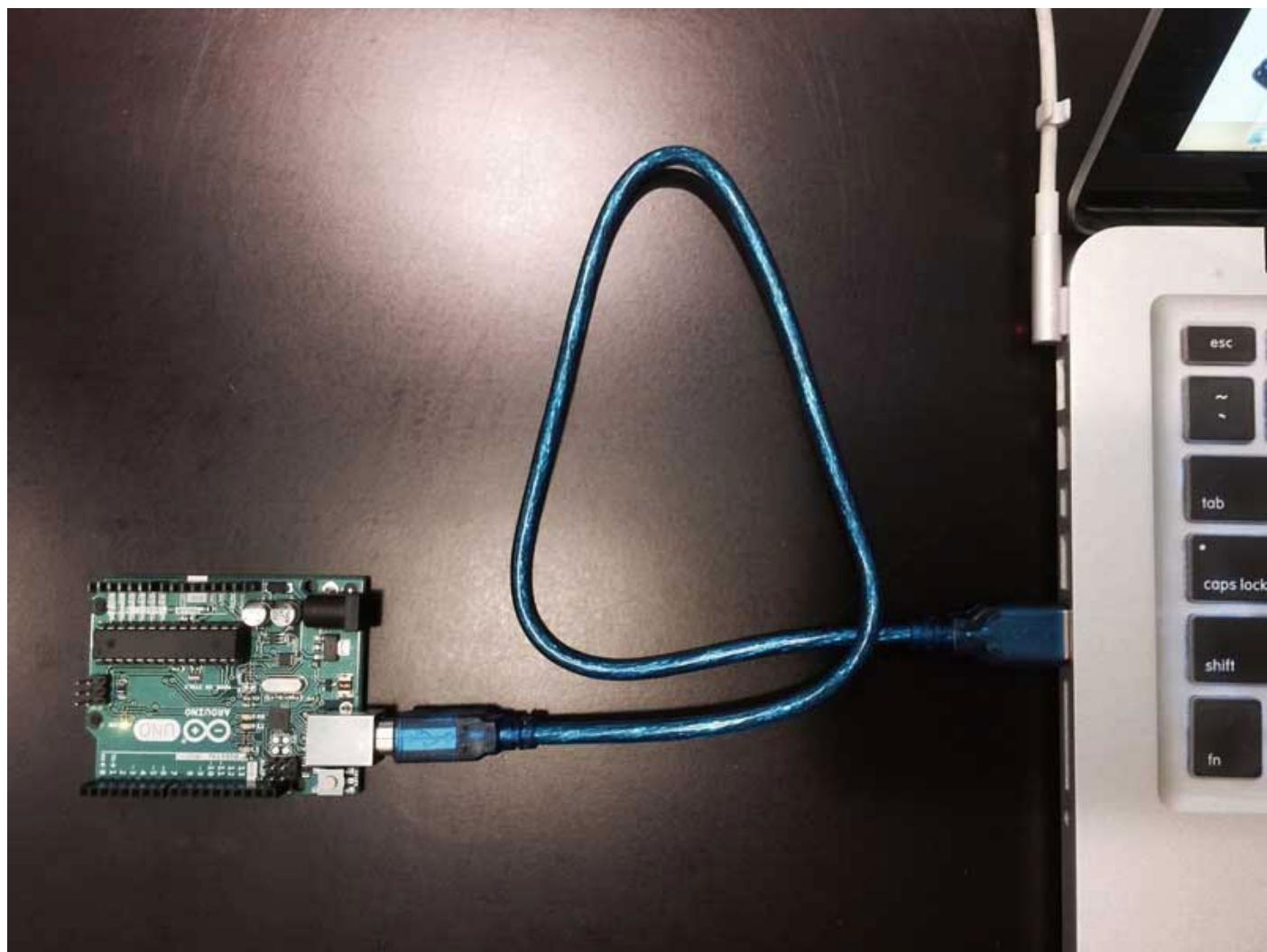
图中标出的数字端口和模拟端口，即为常说的I/O。 数字端口有0~13，模拟端口有0~5。 除了最重要的I/O口外，还有电源部分。UNO可以通过两种方式供电方式，一种通过USB供电，另一种是通过外接6~12V的DC电源。除此之外，还有4个LED灯和复位按键，稍微说下4个LED。ON是电源指示灯，通电就会亮了。L是接在数字口13上的一个LED，在下面一节会有个样例来说明的。TX、RX是串口通讯指示灯，比如我们在下载程序的过程中，这两个灯就会不停闪烁。



如何给Arduino编程

Arduino开发板区别于I/O板的一大特征就是可编程,也就是说Arduino单片机上的具体功能是可以反复更改的。Arduino官方专门为Arduino提供了一套简易化的编程环境(称为Arduino IDE)和编程语言,安装在计算机上,等你编写完特定的程序后,就可以通过USB串口将编译好的程序“上载”或者说“烧录”到Arduino开发板的单片机的存储单元里,然后程序成功被上载后,Arduino就可以脱离计算机独立运行程序了。

Arduino IDE还提供了简易的调试功能,我们将在下一篇教程里,详细讲解IDE的相关知识.

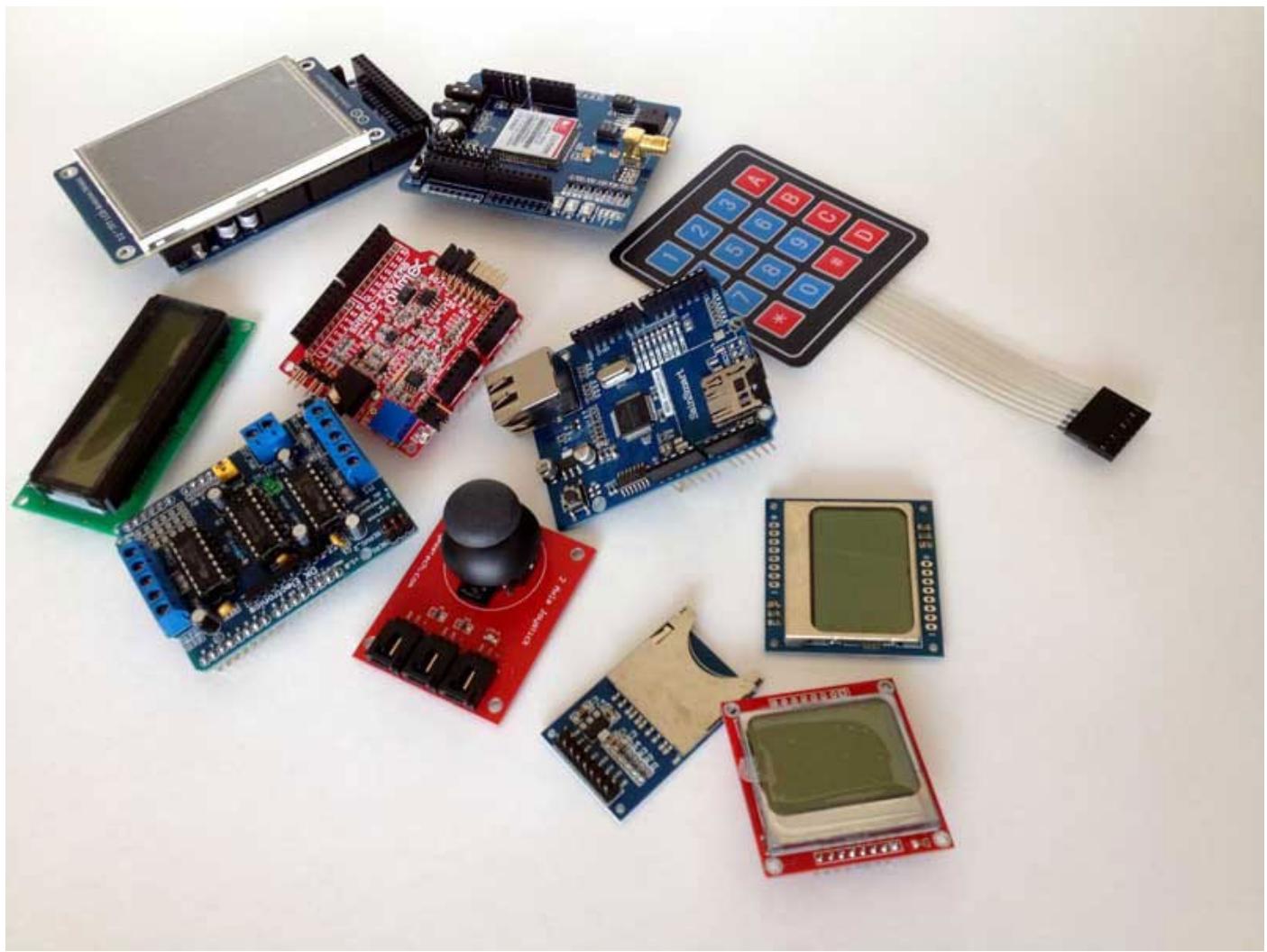


Arduino通过USB与电脑连接

Arduino的扩展模块

综上所述,Arduino开发板只是可以被编程的核心控制模块,就像人的大脑,它并不能单独地完成一个行为,需要有眼睛,鼻子,耳朵这样的传感器,和手脚这样的执行器的配合。市面上有数不胜数的,为Arduino定制的扩展模块:常见的传感器模块类似温度传感器,湿度传感器,光敏传感器,压力传感器等;常见的执行器模块有伺服电机,蜂鸣器,LED等。

Arduino开发板通常提供了足够多的数字/模拟的输入输出接口(PIN),用来外接这些扩展模块。所以,学习使用Arduino很大一部分流程就是在为项目挑选合适的扩展模块,然后再将它们正确的和Arduino开发板连接,利用Arduino IDE编写正确的驱动程序,最后不断调试直至成功。

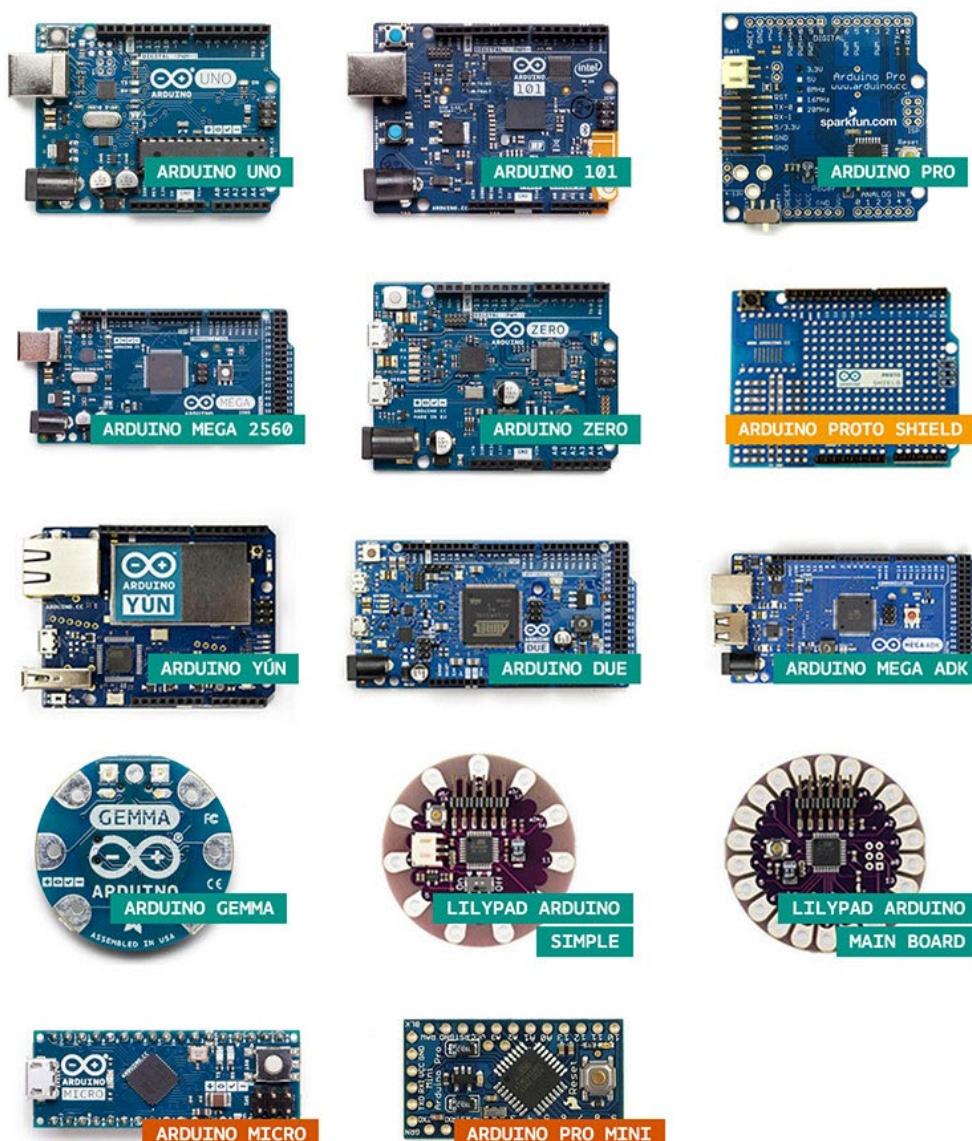


一些Arduino的扩展模块

Arduino的不同版本

访问Arduino官网的产品目录,你就会发现Arduino硬件版本玲琅满目,弱小到如单片的AtTiny85,强大到Arduino Due,每一种都有独有的特性,不同的体积,不同的计算能力: 如果你的项目是可穿戴的,大多数情况下会推荐LilyPad,因为它有适合缝织的焊盘; 加入你想做相关物联网的项目,那么MKR1000和WiFi Shield 101是你需要的,它们都板载了Wifi模块,天生就有网络处理能力... 当然,这里并没有绝对的规则供新手参考,只能慢慢玩转不同的开发板,积累了足够多的经验后,你才能对它们对特缺点了如指掌,然后自然而然地知道该在什么场合下该用哪种Arduino。

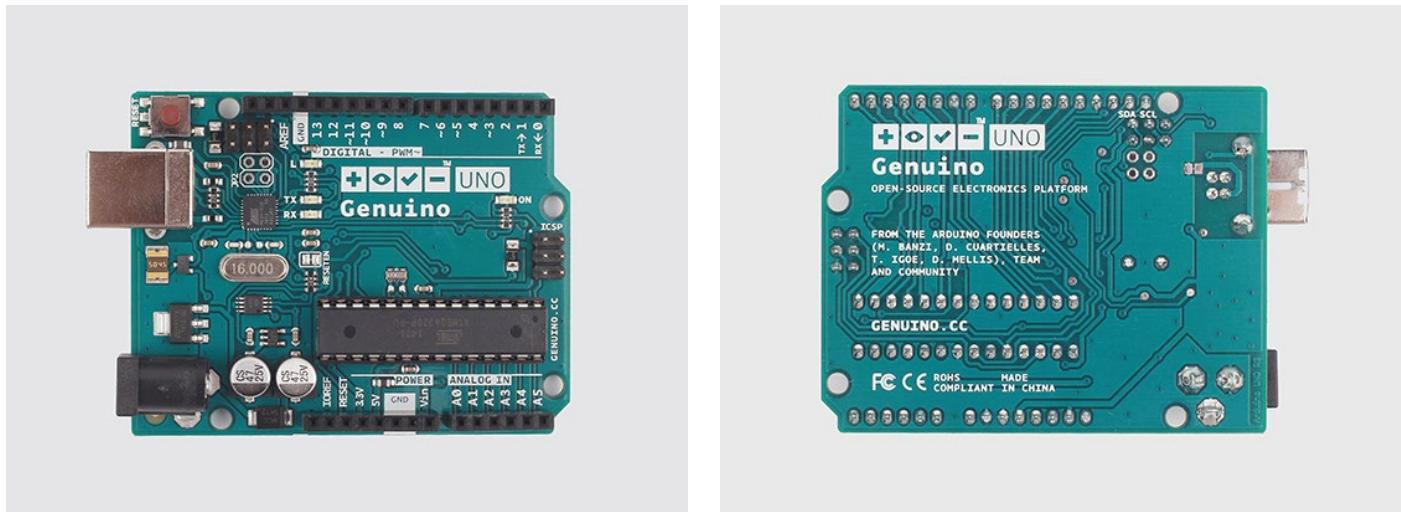
如果你没有清晰的项目需求或者只是想先学习Arduino,那先选择Arduino UNO八成没有错。UNO是目前最流行,也是最主线的版本。



Arduino通过USB与电脑连接

Genuino

Arduino属于开源硬件，顾名思义就是所有的硬件设计源文件都是开源。任何组织和个人在其开源协议下都可以自己制造和售卖Arduino，所以在某宝上你可以找到价格相差巨大的同一版本，并无山寨的Arduino，只有用料和质量的差异。我们首选推荐的当然是官方指定／合作硬件生产商的产品。目前，官方商品在美国用的是Arduino，而在其他国家使用的是Genuino。Genuino在国内的合作方是深圳的Seeed矽递科技。



Genuino

03 安装和熟悉Arduino IDE

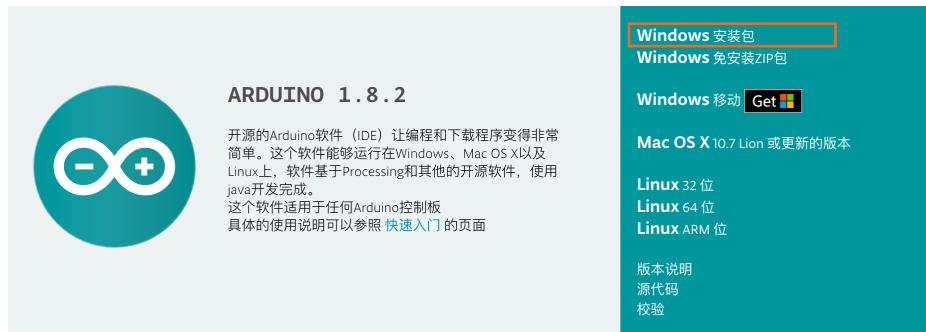
安装Arduino IDE

1. 下载Arduino IDE

在浏览器中访问网址：

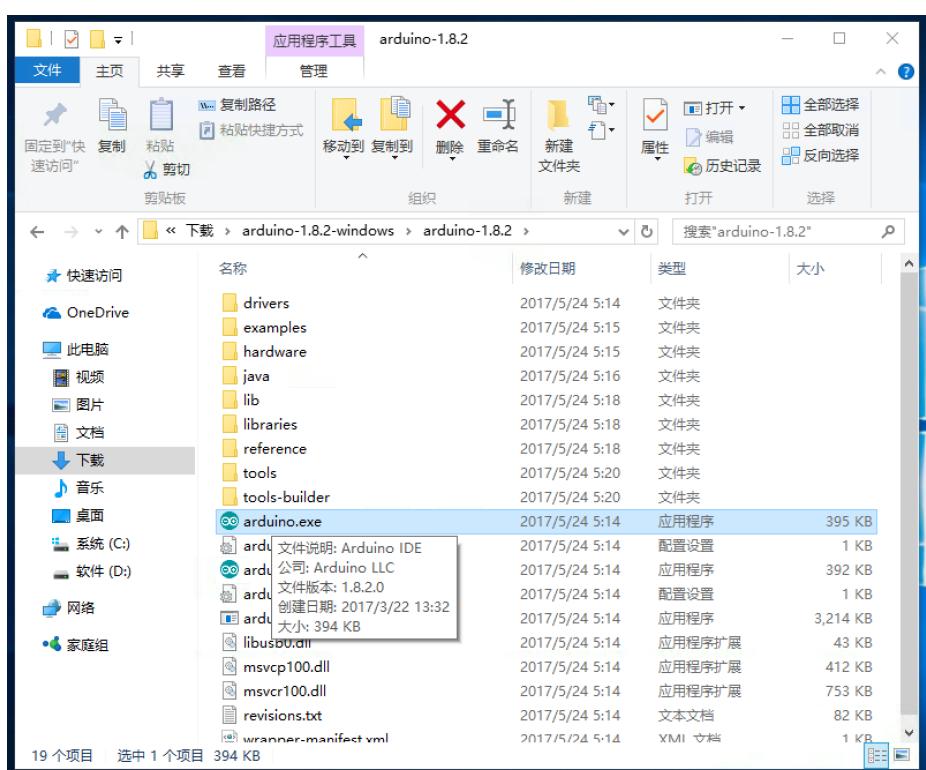
<https://www.arduino.cc/en/Main/Software>

(由于国外下载速度不行，我们
已经把Arduino IDE安装文件同
步到百度盘，可以在“[Arduino
IDE软件](#)”文件夹内找到。



注意这里有Windows Installer
和Windows Zip文件两个选择，
脑叔推荐用Windows Installer
安装，这样电脑会自动安装相关
驱动，但你必须拥电脑的安装程
序权限。如果你选择Zip压缩包
文件，你需要手动安装Arduino
的驱动程序。

如果Mac, Linux用户则选择相
应的系统。



2. 安装驱动

如果你用Windows Installer安装同时你的系统是XP及XP以上至Window 10,只要你连接好Arduino,系统会自动搜索并安装驱动。但如果你下载的是Zip压缩包,或者由于某些原因,你的Arduino无法被系统正确识别,你可以参考以下步骤手动安装程序。

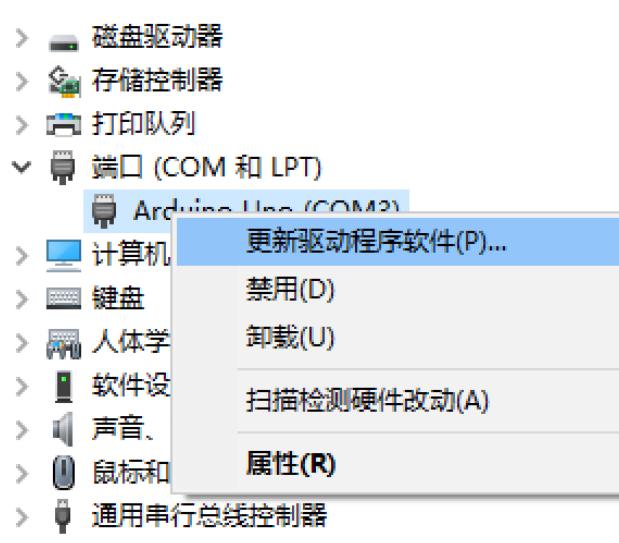
把USB一端插到Arduino UNO上,另一端连到电脑。连接成功后,UNO板的红色电源指示灯ON亮起。

点击“开始”菜单,打开“控制面板”。

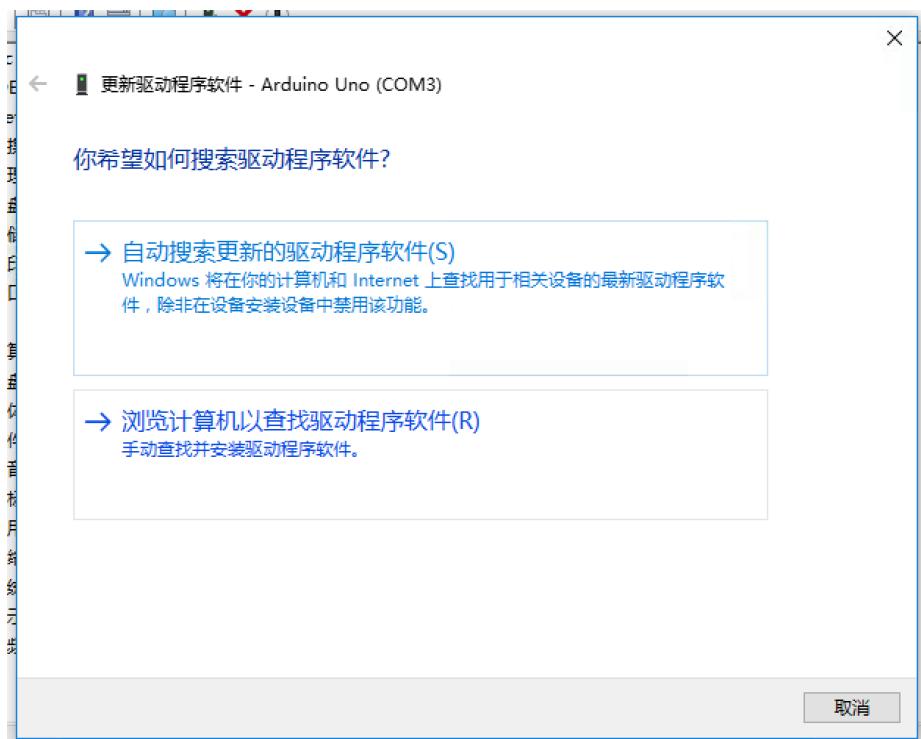
在“控制面板”下,点击“系统和安全”选项,下一步,点击“系统”。“系统”窗口出现后,打开“设备管理器”。



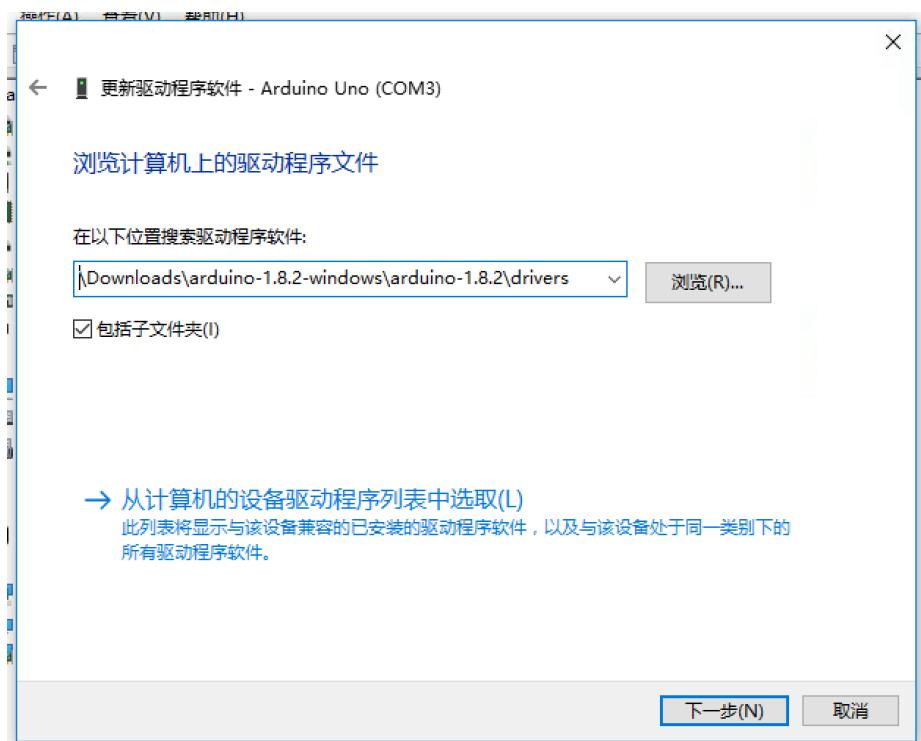
找到“设备管理器”列表中的“端口(COM & LPT)”选项,你应该能看到一个名为“Arduino UNO(COMxx)”的端口。如果没有“端口(COM & LPT)”选项,找找“其它设备”中的“未知设备”右键点击“Arduino UNO(COMxx)”的端口,选择“更新驱动程序”。



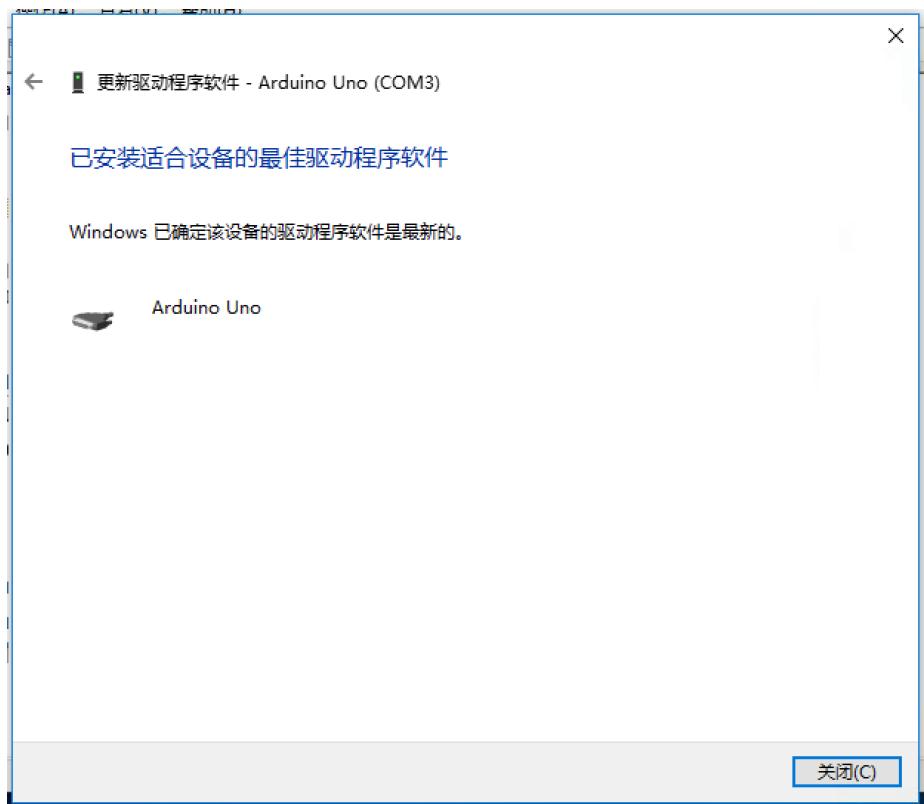
下一步，选择“浏览计算机以查找驱动程序软件”。



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。

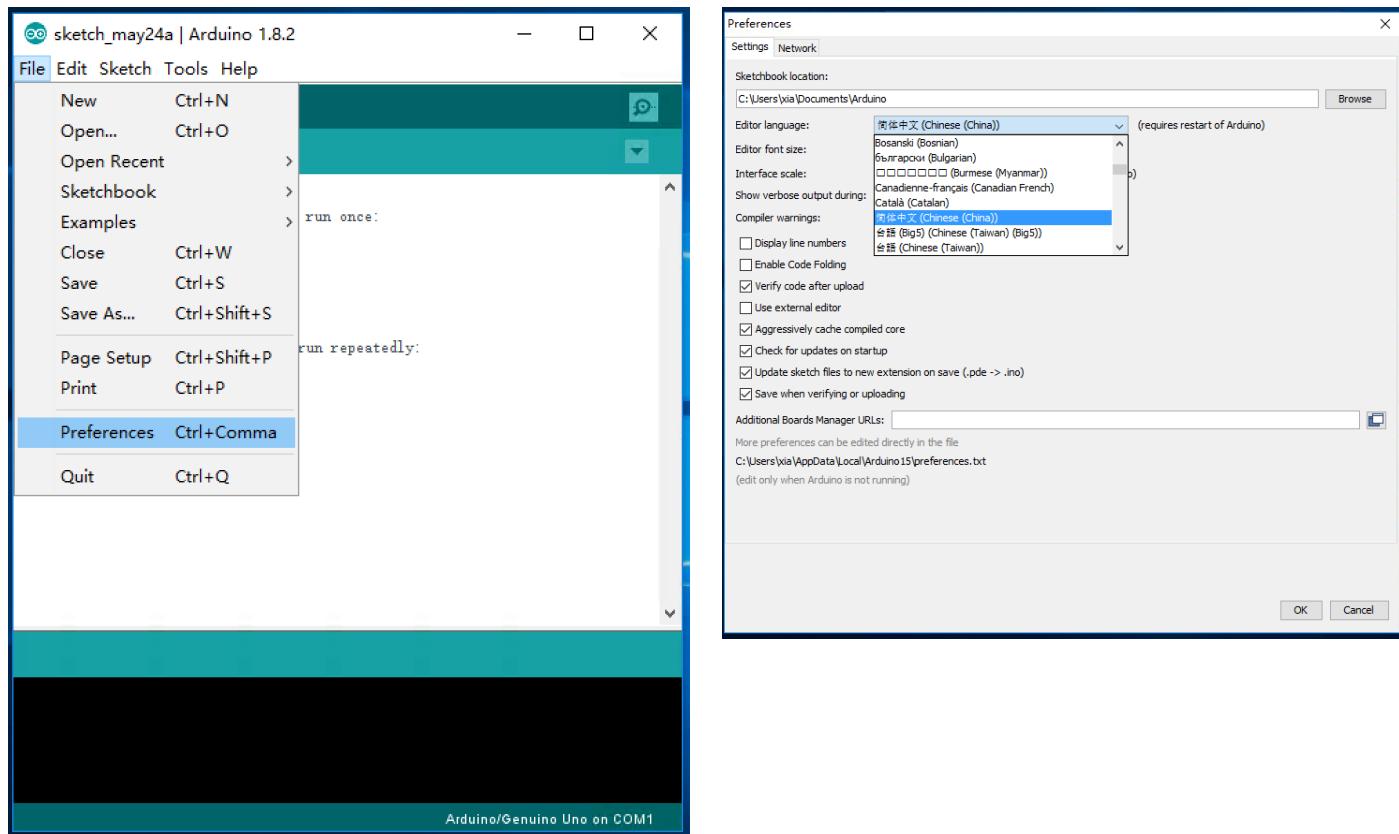


选择始终安装此驱动程序软件，直至完成。出现右图，说明驱动安装成功。

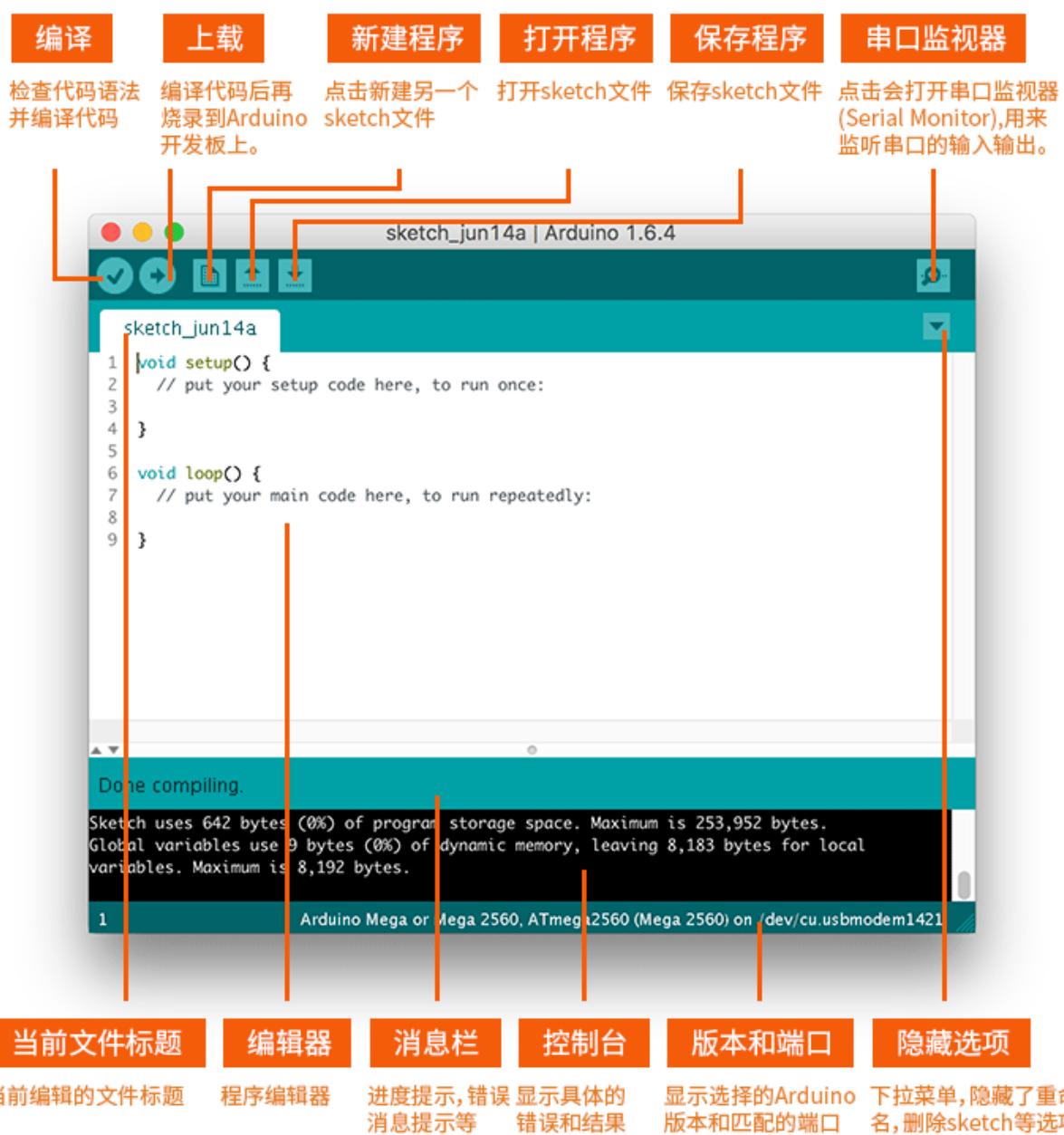


3. 认识Arduino IDE

打开Arudino IDE。选择菜单栏File > Preferences。跳出的对话框里，选择Editor language > 简体中文，点击OK。

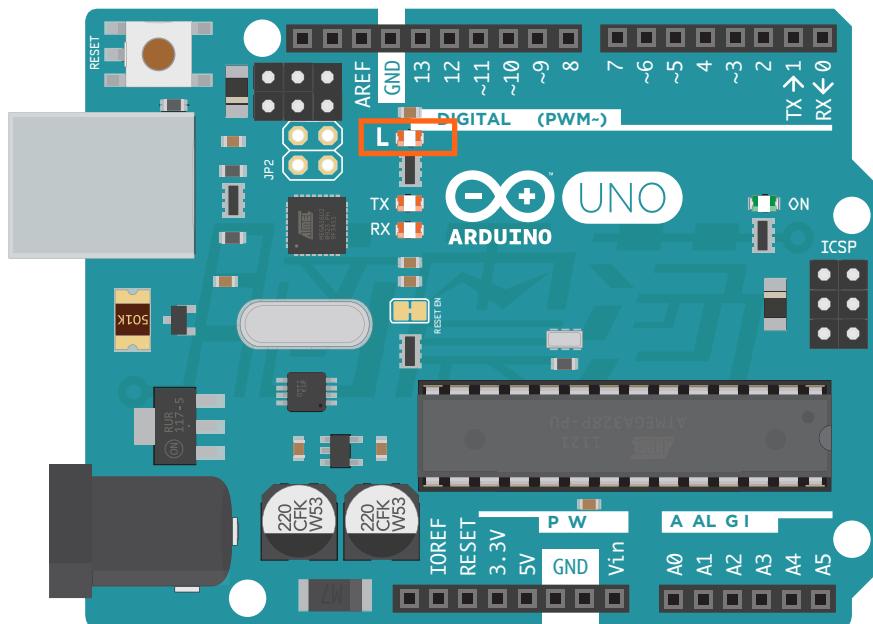


Arduino IDE的界面为了易用性,如图所示大大简化界面包含的元素:

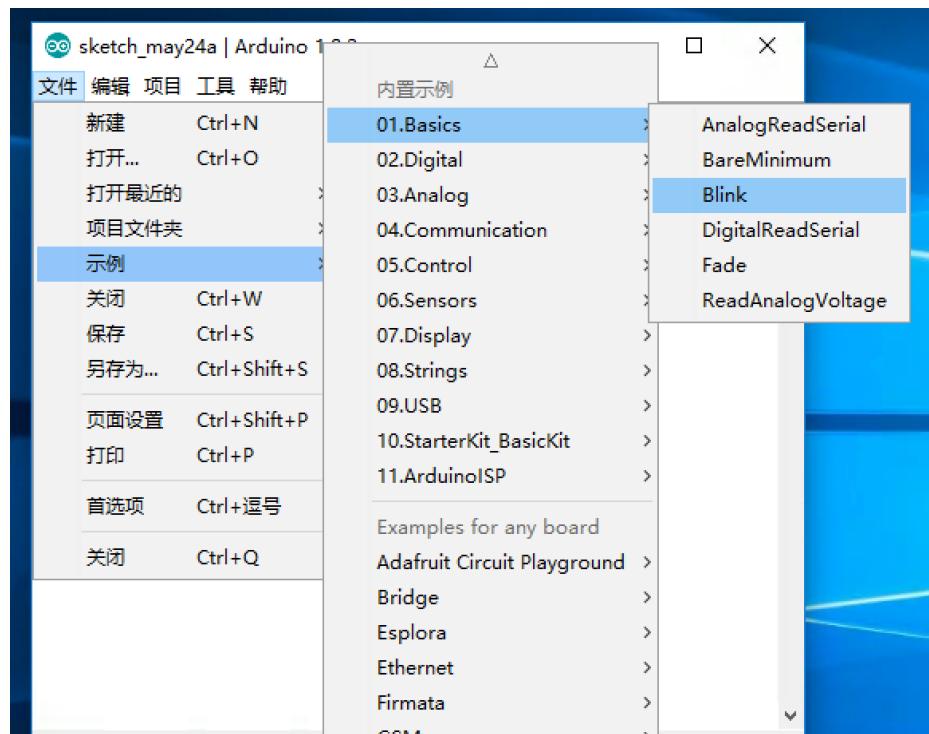


上传程序

让我们试一下上传一个最简单的例子代码，既可以熟悉如何上传程序，同时也测试一下板子是否正常。如果仔细观察UNO板，可以发现13号引脚上标有L的LED。这段测试代码就是让这个LED灯闪烁。



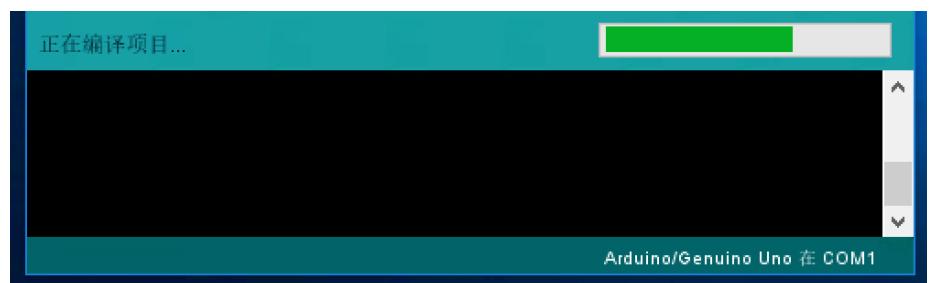
打开Arduino IDE，从“文件 > 示例 > 01.Basics > Blink”打开 Blink 实例。



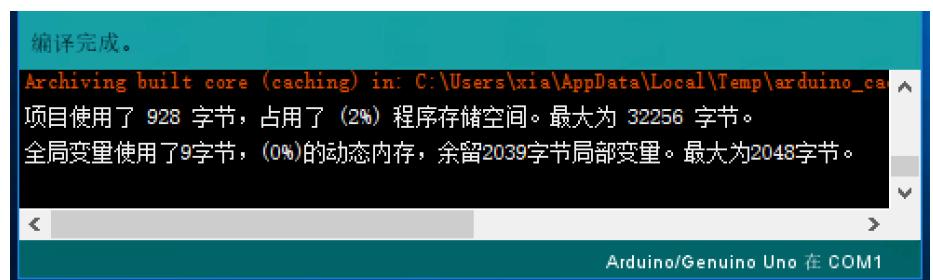
通常，写完一段代码后，我们都需要验证一下，看看代码有没有错误。点击“验证”。



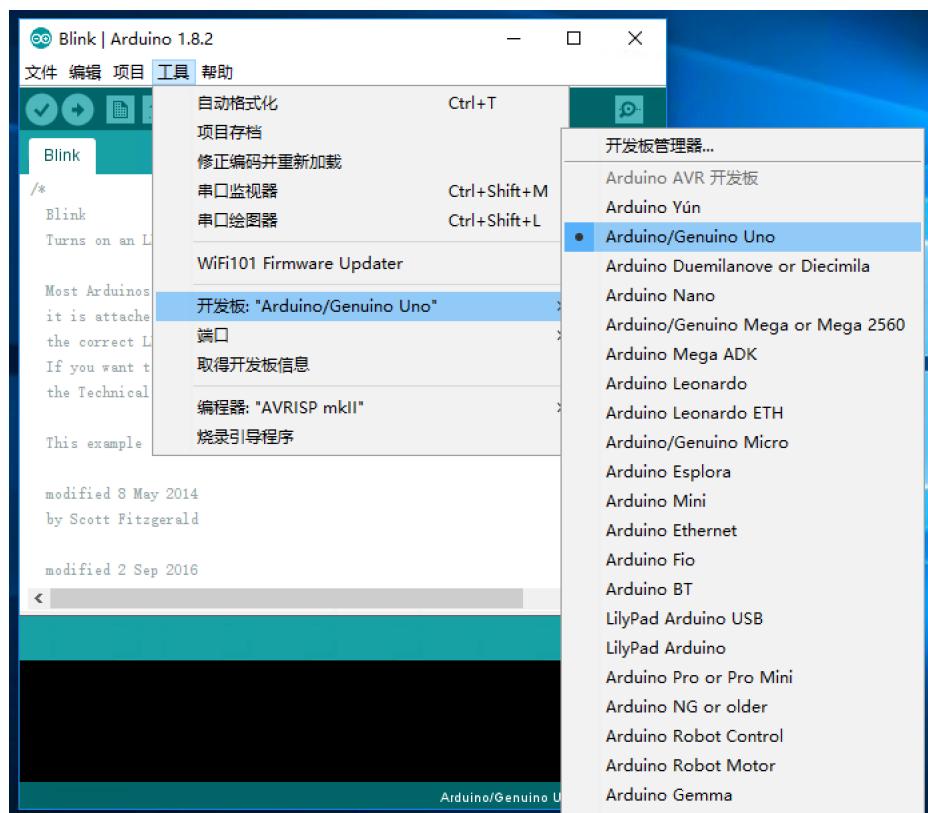
点击“验证”后，会出现进度条，显示正在编译中。



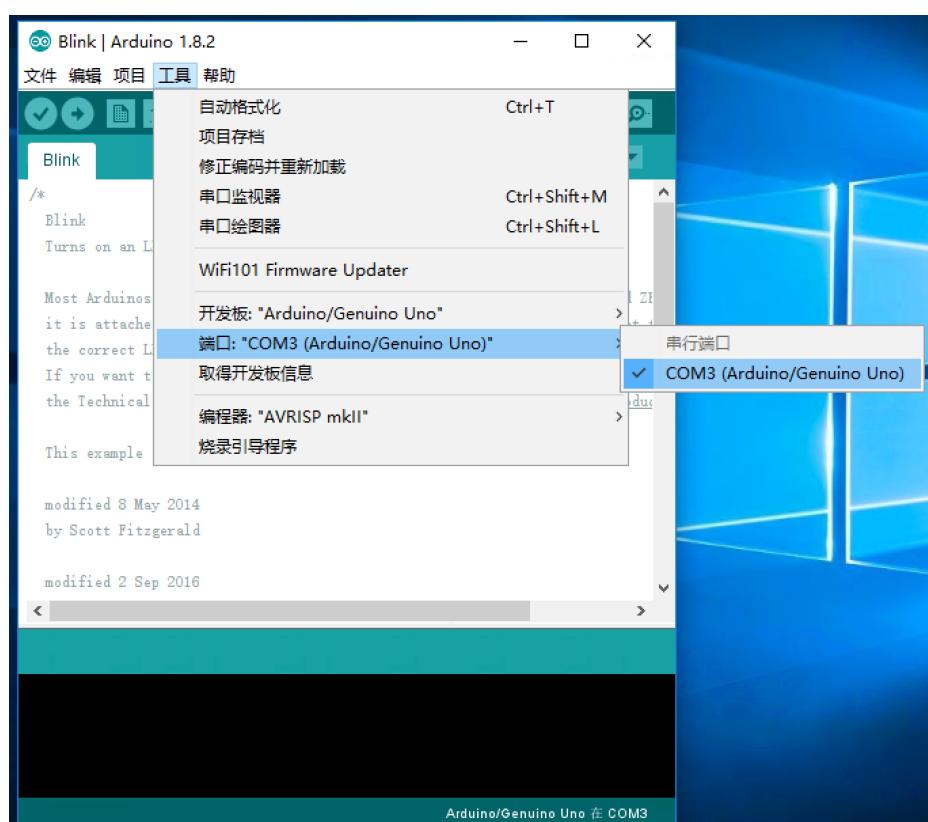
如果显示“编译完成”，说明验证没有错误。如果有错误，消息栏和控制台会显示具体错误信息。输入完代码都需要验证一下，然后再上传到Arduino中。但其实每次点击“上传”都已经包含了“验证”过程。



IDE无法自动得知你正在使用哪一种版本的Arduino，这个过程是手动的。在“工具 > 开发板”下面选择对应的“Arduino/Genuino Uno”(如果你使用的是其它版本，请选择相应的版本)。



除了选择正确的版本外，还要手动匹配正确的串行通信的端口。在“工具 > Serial Port/串口”下面你可以看到可用的串口。一般来说COM1和COM2是被系统占用的，所以说很多时候是COM3或者更大数字的COMx。通常你可以把Arduino和电脑之间的USB线拔掉再插上，如果哪个COM口消失又出现了，那就是Arduino占用的串口。选择中它就可以了。

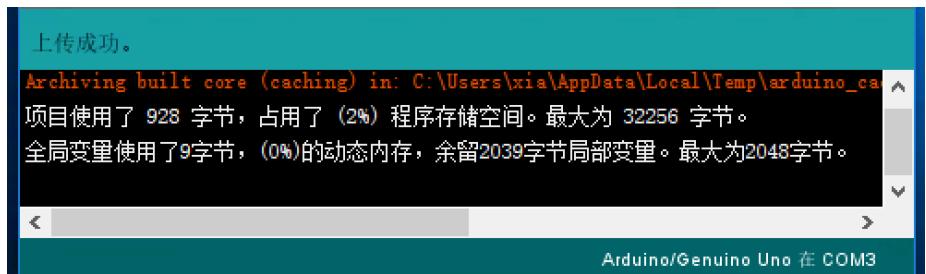


现在，你只要点击IDE的工具栏上面第二个带有箭头的“上传”按钮*。等待几秒钟，你应该能看到Arduino板上标有RX和TX的黄色指示LED在狂闪，说明IDE在烧录程序。

* 英文界面是标有Upload的按钮，所以也有中文叫上传。

如果上传成功，你在状态栏里应该会看到“上传成功”的提示，否则会提醒失败及原因。

上传成功结束后,如果你看到与D13引脚相连的板载LED一闪一闪发着黄光,那么恭喜你,你成功安装了Arduino,而且成功地完成了一次程序上传的实践!



04 面包板的使用

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。

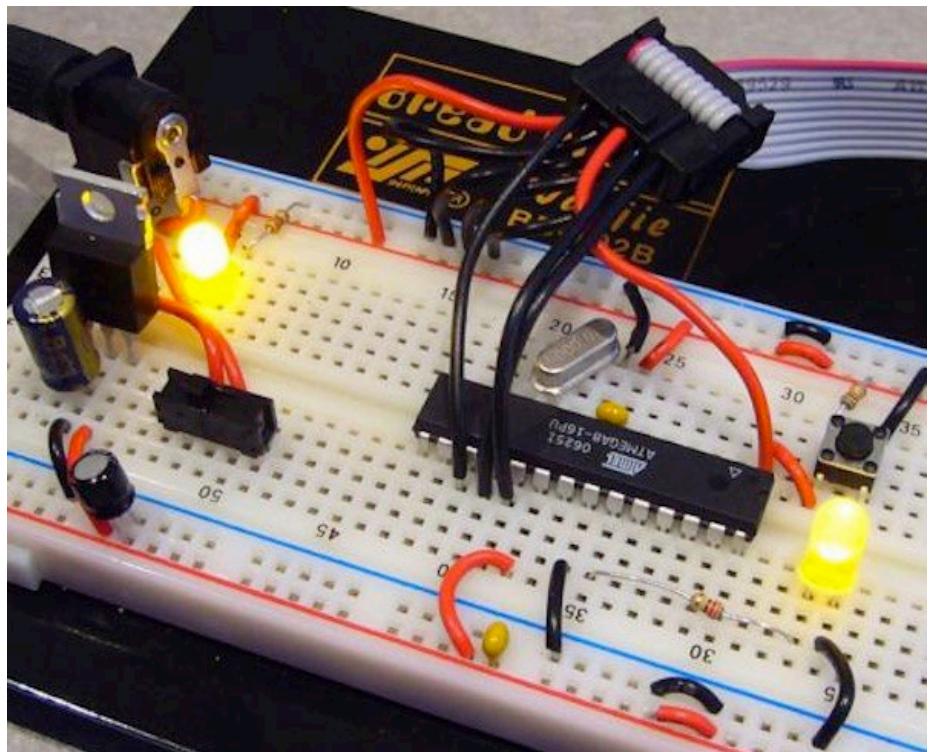
为什么叫面包板

很多年前，电子元件还是相当的大而笨重，电子工程师们就偷偷用了妈妈们做面包时用的板，外加一些图钉，铆钉作为他们搭建电路的平台和连结点。直到电子元件逐年变小变轻，电子工程师发明了更好的工具来搭建电路，这让世上很多母亲非常高兴因为终于她们可以从儿子那要回面包板了。但是电子工程师们仍然沿用着“面包板”的名字，更确切的说，我们今天所称的“面包板”是指无焊面包板。



在面包板上电路板

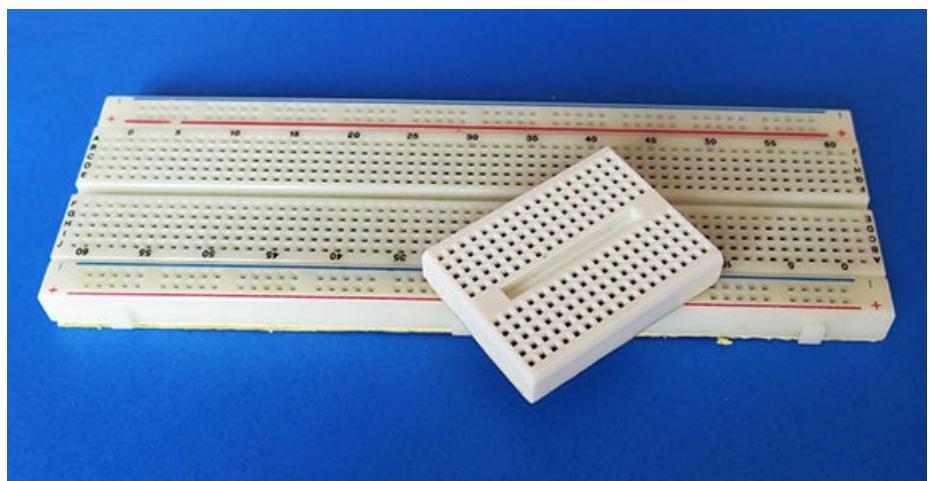
面包板诞生的初衷，就是提供一个临时搭建电路板，尝试调整电路连接又无需焊接的工作基台。



在无焊面包板上搭建的电路

原型开发是一个尝试不同方法，试验结果，修改迭代的过程，直至最后的结果确定。这是面包板为什么最适合原型开发过程中使用的原因。你能在面包板上打架最简易的连接，也能构建最复杂的电路。而且你也可以拼接更多的面包板来构建更多更复杂的连接。

面包板上的所有连接是无须焊接的，如果你连错了直接拔掉重连即可。

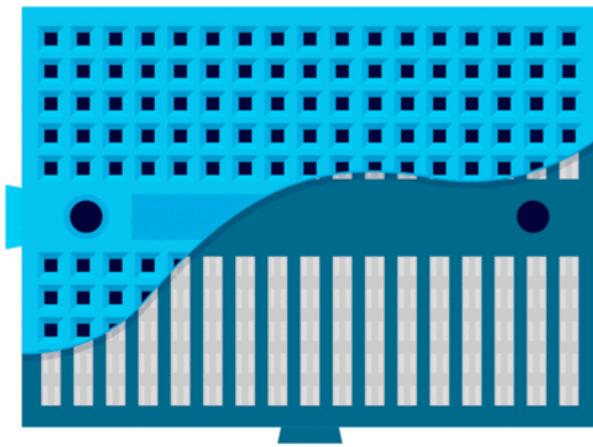


Mini面包板和标准的面包板

面包板的构成

1. 端子片

这是面包板最基础的结构件，外观上就是一个方洞。如果我们把一块面包板暴力打开，我们会发现下面这样的端子片，每件都是由5个小夹头连在一起。而小夹头正好可以卡在塑料件的方格子里。这样我们把跳线或者电子元件的针脚插入方格子就可以被小夹头夹在里面。然后面包板就是由这样的金属件排列而成。所以面包板的一个基本特性就是单位五个插孔之间(如图中的横向)电路是通路，每列(如图中的纵向)插孔之间又是不通路的。

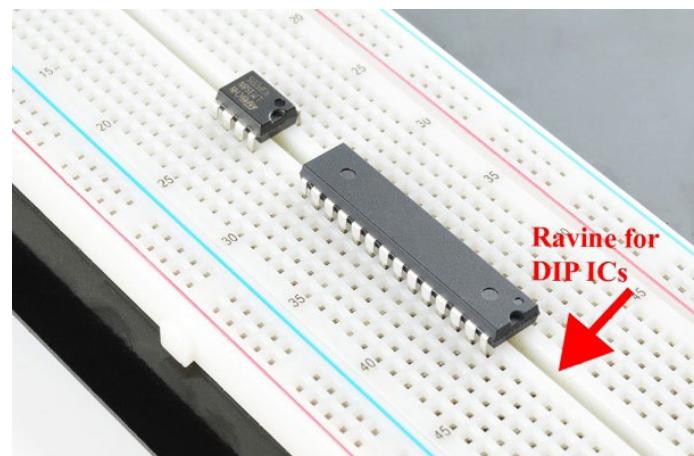


面包板的端子片结构

2. DIP凹槽

面包板中间往往有个大凹槽，这样的设计是有讲究的。

设计的可供性上示意上下两部分是断开的。算上凹槽的宽度，紧挨凹槽两边的插孔的间距正是7.62mm，这个间距正好插入标准窄体的DIP引脚的元件。DIP IC插上后，往往由于引脚过多，很难徒手拔出，如果暴力拔出很容易弄弯针脚，凹槽刚好可以让镊子之类的工具伸到下面，慢慢翘起IC。

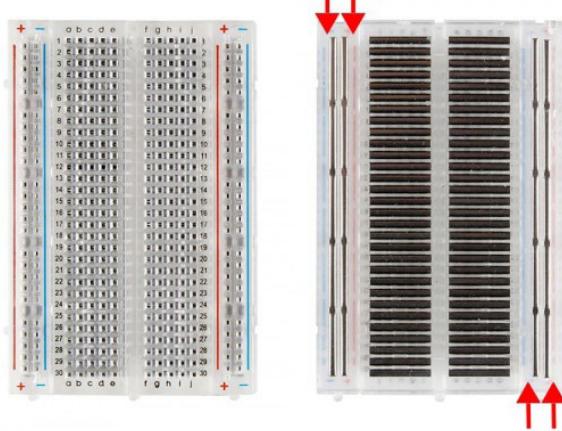


面包板中间的凹槽

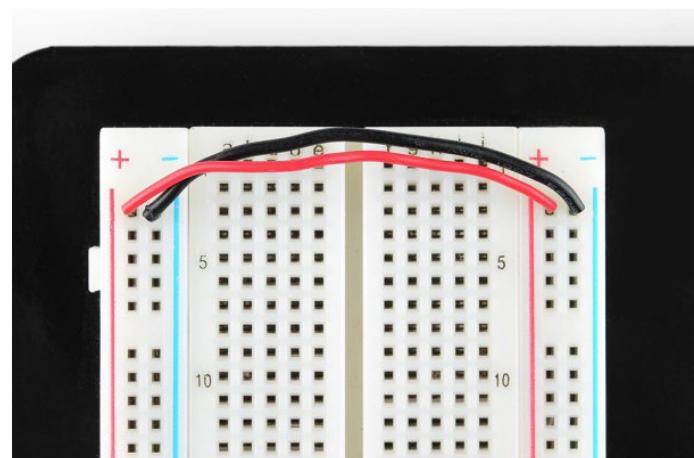
3. 电源通道

如图所示，标准的大面包板两侧，往往各有两列插孔，通常有红色线的“+”，蓝色或黑色线的“-”。这是专门给整板提供电源插孔的电源通道。

特别值得一提的是两边的电源通道之间是没有通路的。所以如果你希望两边都有电源的话，最好使用跳线把两边的通道对应连起来，如图。



面包板两侧的电源通道



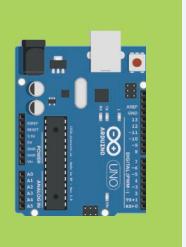
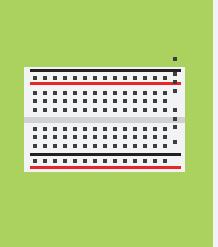
跳线连接两侧电源通道

05 经典仪式点亮LED

Blink

在这一篇教程里，我们要来完成经典的“Arduino点亮LED仪式”。点亮LED对于学习Arduino而言，好比佛教的剃度，基督教的洗礼，是新手必须经历的一个练习。其实在《入门3：安装和熟悉Arduino IDE》中我们便提及了Blink实例，但由于过于经典，我们单独拎出来作为一篇教程详细讲解。

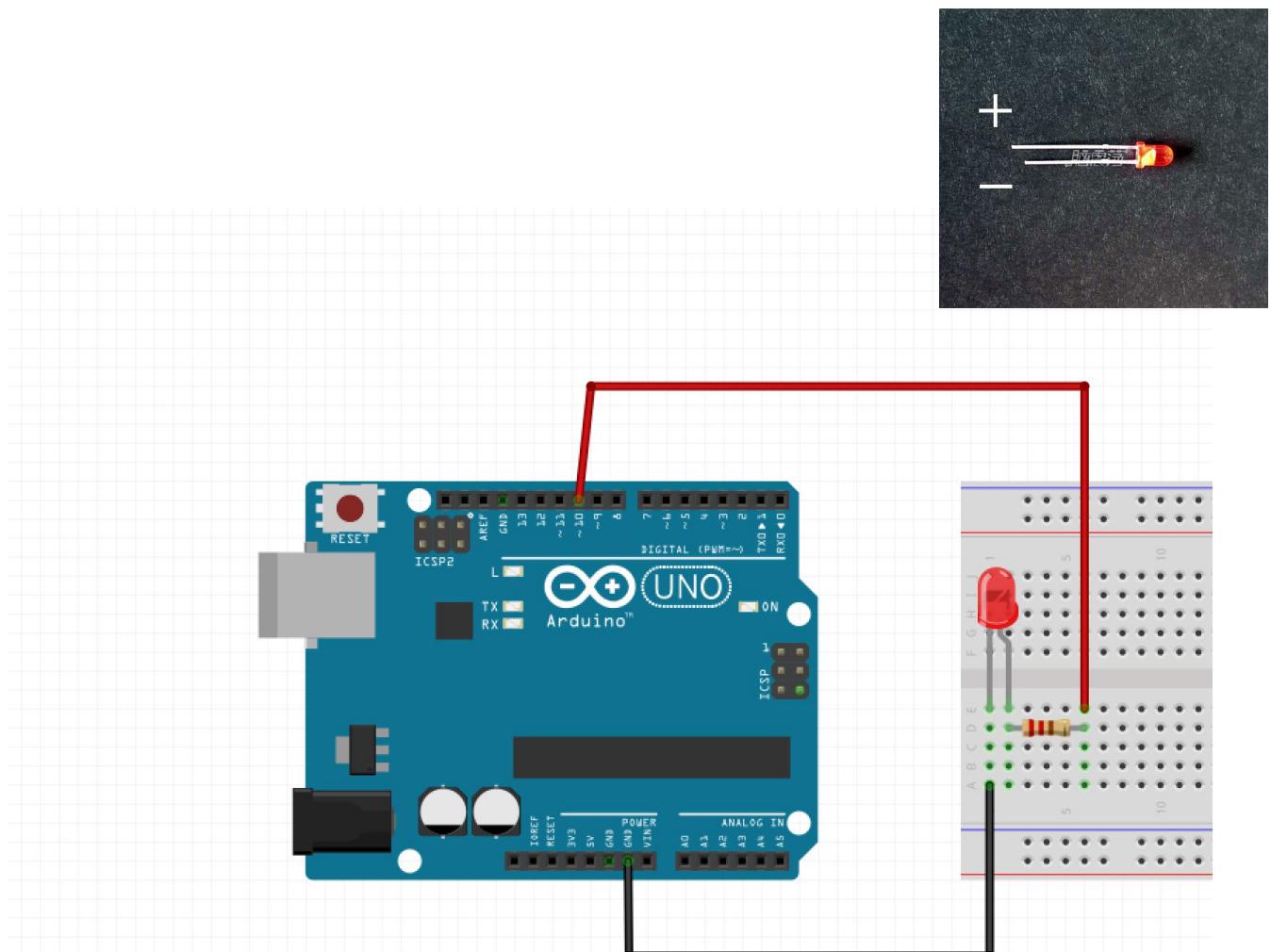
所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1
	220欧姆电阻 220Ω Resistors	x1		红色LED Red LED	x1

硬件连接

取出所有元件，按照图示连接。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。

具体连接时，要注意LED是有正负之分的，长脚为正(+)，短脚为负(-)。完成连接后，给 Arduino接上USB数据线。Arduino和电脑USB端口连接后可以获得电源，电脑USB端口供电的电压通常是5V，所以不需要额外通过电源插口给Arduino供电。



输入代码

打开Arduino IDE，一般系统会自动生产一个新的文件。如果没有，通过“文件 > 新建”创建新的Arduino文件。Arduino文件后缀名为.ino，一般被称为sketch(草稿)文件。

虽然你可以直接双击我们提供的项目示例文件。但脑叔还是建议初学者自己一行一行输入代码，有助于学习编程。

```
1 //入门5示例：点亮闪烁LED
2 /*
3     描述：LED每隔一秒交替亮灭一次
4     by www.naozhendang.com
5 */
6 int ledPin = 10;
7
8 void setup() {
9     pinMode(ledPin, OUTPUT);
10}
11 void loop() {
12     digitalWrite(ledPin, HIGH);
13     delay(1000);
14     digitalWrite(ledPin, LOW);
15     delay(1000);
16 }
```

输入完毕后，点击IDE的“验证(Verify)”，查看输入代码是否通过编译。如果显示没有错误，单击“上传(UpLoad)”，给Arduino上载代码。

当上载烧录成功后，状态栏会提示“上传成功(Done uploading)”，然后文本控制窗口中会显示上载程序的占用的大小。, 你就可以看到红色LED有规律的一闪一闪。



理解代码

```
1 //入门5示例：点亮闪烁LED
```

```
2 /*  
3  * 描述：LED每隔一秒交替亮灭一次  
4  * by www.naozhendang.com  
5 */
```

```
6 int ledPin = 10;
```

```
int ledPin = 10;  
变量类型 变量名
```

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，很多人容易忽略注释，初学者应该培养好的习惯。第一学会看注释，注释的作用是提供信息，注释能帮助你更快得看懂代码。第二要养成写注释的习惯，即使你的代码的读者只有你自己，也别忘了写注释说明。

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。大段说明适合这种注释。IDE将自动把注释的文字颜色变为灰色。

声明一个名叫“ledPin”的整数变量。变量是用来存储数据的。这个例子，我们用的类型是int型(整型)，可以表示一个在-32768到32767之间的数。变量的类型，是由你要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，看上去比较容易理解。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。**注意：分号必不可少！而且必须切换到英文输入法中的分号。**

如何理解变量

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，那样才放得下，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。

```

8 void setup() {
9
10 }

void setup() {
}

```

无返回值函数 空括号

在这个程序里有两个函数，一个叫做setup，顾名思义它的目的主要是loop函数运行之前为程序做必要的初始设置。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。

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

setup()函数内只有一条语句，那就是pinMode函数。

pinMode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。

pinMode(pin, mode)

引脚

模式INPUT / OUTPUT

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，怎么写呢？

很简单：pinMode(2, INPUT);

函数

函数就是一个能实现某种具体功能的模块，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。这些功能块还能被反复运用。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。

setup()和loop()在Arduino程序中比较特殊，不能反复调用。函数的返回值，就是函数执行完毕的反馈。你可以创建一个函数的时候就告诉IDE这个函数是否有返回，比如“void”就表示函数无返回值，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。

INPUT和OUTPUT

这里的INPUT和OUTPUT是相对于Arduino/单片机而言。

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。

```
11 void loop() {  
12   digitalWrite(ledPin,HIGH);  
13   delay(1000);  
14   digitalWrite(ledPin,LOW);  
15   delay(1000);  
16 }
```

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。

| 12 | digitalWrite(ledPin,HIGH);

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);


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()。

pinMode() digitalWrite() digitalRead() 的关系

pinMode() 的INPUT和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

pinMode(pin,OUTPUT)

digitalWrite(pin,HIGH/LOW)

比如:按钮

pinMode(pin,INPUT)

digitalRead(pin)

| 13 | `delay(1000);`

delay函数是一个延时函数，就是告诉Arduino维持现状啥也不干，要给出这个延时的具体长度，这里是1000，注意单位是毫秒(ms)，1000ms也就是1秒。要延时2s就是delay(2000)。

| 14 | `digitalWrite(ledPin, LOW);`

这句话的意思就把ledPin(即10)引脚电平拉低，就是熄灭LED。再延时1s。点亮LED—维持1s—熄灭LED—维持1s，再不断重复着过程。我们看到的就是D13上的板载LED间隔一秒地闪烁。

如果你想让LED闪的更快点，怎么办呢？没错，你只要修改delay函数里的时间值就可以了。减小这个延时的值，就表示等待时间更短，也就是闪的更快。反之，闪的更慢。

06 S.O.S求救信号灯

摩斯码

本项目将继续延用入门1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求救信号灯。

摩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，非常简便！摩斯码在早期无线电上举足轻重，是每个无线电通讯者所须必知的。

我们正好可以通过LED开关两种状态来拼出一个字母。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，我们这里用快闪烁替代，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，不难理解入门6示例1的代码。

```
1 //入门6示例1：S.O.S求救信号灯
2 /*
3     描述：分别用短闪烁和长闪烁表示S和O字母
4     by www.naozhendang.com
5 */
6 int ledPin = 10;
7
8 void setup() {
9     pinMode(ledPin, OUTPUT);
10 }
11 void loop() {
12     // 三个快闪烁来表示字母“S”
13     digitalWrite(ledPin,HIGH);
14     delay(150);
15     digitalWrite(ledPin,LOW);
16     delay(100);
17     digitalWrite(ledPin,HIGH);
18     delay(150);
19     digitalWrite(ledPin,LOW);
20     delay(100);
21     digitalWrite(ledPin,HIGH);
22     delay(150);
23     digitalWrite(ledPin,LOW);
24     delay(100);
25
26     delay(100); //100毫秒延时
27
28     //三个短闪烁来表示字母“O”
29     digitalWrite(ledPin,HIGH);
30     delay(400);
31     digitalWrite(ledPin,LOW);
32     delay(100);
33     digitalWrite(ledPin,HIGH);
34     delay(400);
35     digitalWrite(ledPin,LOW);
36     delay(100);
37     digitalWrite(ledPin,HIGH);
38     delay(400);
39     digitalWrite(ledPin,LOW);
40     delay(100);
41
42     delay(100); //100毫秒延时
43
44     //再用三个快闪烁来表示字母“S”
45     digitalWrite(ledPin,HIGH);
46     delay(150);
47     digitalWrite(ledPin,LOW);
48     delay(100);
49     digitalWrite(ledPin,HIGH);
50     delay(150);
51     digitalWrite(ledPin,LOW);
52     delay(100);
53     digitalWrite(ledPin,HIGH);
54     delay(150);
55     digitalWrite(ledPin,LOW);
56     delay(100);
57
58     delay(5000); // 在重复S.O.S信号前等待5秒
59 }
```

示例1的代码非常简单明了，也没有任何错误。唯一的问题就是烦琐。如果有100个不同的字母，难道要写100遍么？那怎么办呢？这个时候就要学会使用for循环。

使用了for循环后，整体代码就缩短了，更容易理解。写代码的时候也要认真写注释。验证正确后，上传代码到Arduino中，如果一切顺利的话，我们将看到LED闪烁出摩尔斯码S.O.S信号，等待5秒。重复闪烁。给Arduino 9V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。

```
1 //入门6示例2：S.O.S求救信号灯实现2
2 /*
3     描述：分别用短闪烁和长闪烁表示S和O字母，使用for循环
4     by www.naozhendang.com
5 */
6 int ledPin = 10;
7
8 void setup() {
9     pinMode(ledPin, OUTPUT);
10 }
11 void loop() {
12     // 三个快闪烁来表示字母“S”
13     for(int i=0;i<3;i++){
14         digitalWrite(ledPin,HIGH); //设置LED为开
15         delay(150); //延时150毫秒
16         digitalWrite(ledPin,LOW); //设置LED为关
17         delay(100); //延时100毫秒
18     }
19     delay(100); //100毫秒延时
20
21     //三个短闪烁来表示字母“O”
22     for(int i=0;i<3;i++){
23         digitalWrite(ledPin,HIGH); //设置LED为开
24         delay(400); //延时150毫秒
25         digitalWrite(ledPin,LOW); //设置LED为关
26         delay(100); //延时100毫秒
27     }
28     delay(100); //100毫秒延时
29
30     //再用三个快闪烁来表示字母“S”
31     for(int i=0;i<3;i++){
32         digitalWrite(ledPin,HIGH); //设置LED为开
33         delay(150); //延时150毫秒
34         digitalWrite(ledPin,LOW); //设置LED为关
35         delay(100); //延时100毫秒
36     }
37     delay(5000); // 在重复S.O.S信号前等待5秒
38 }
39
40
41 }
```

语法学习

```
13 for(int i=0;i<3;i++){  
14     digitalWrite(cledPin,HIGH);  
15     delay(150);  
16     digitalWrite(cledPin,LOW);  
17     delay(100);  
18 }
```

for语句用于重复执行被花括号包围的语句块。一个增量计数器通常被用来递增和终止循环。for语句对于任何需要重复的操作是非常有用的。常常用于与数组联合使用以收集数据/引脚。for循环的头部有三个部分：

```
for (初始化部分; 条件判断; 数据递增部分){  
    循环语句块  
}
```

初始化部分被第一个执行，且只执行一次。每次通过这个循环，条件判断部分将被测试；如果为真，语句块和数据递增部分就会被执行，然后条件判断部分就会被再次测试，当条件测试为假时，结束循环。

比较运算符

“<”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。

07 用按钮控制LED

按键开关

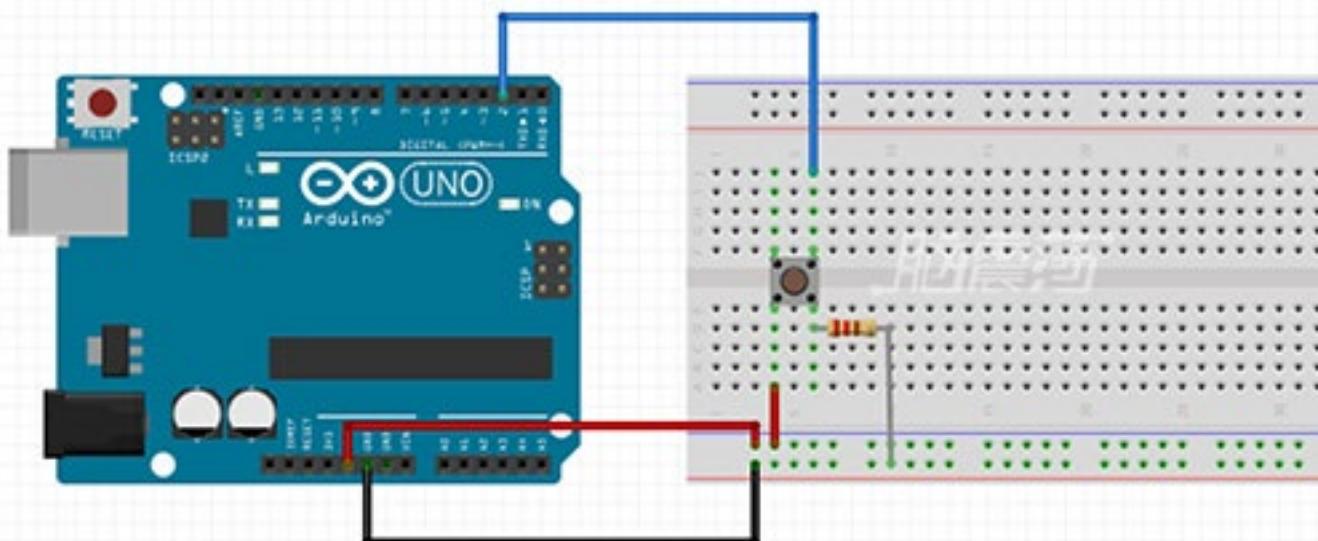
直接两篇入门教程，我们都是用代码直接控制LED。那么如何用一个按键开关来控制LED呢？本篇教程里我们就来学习一下如何用按钮控制板载的LED，并且再一次温习一下digitalRead()和digitalWrite()。

所需元件：



硬件连接

首先将开关的两端分别连接5V和D2号数字引脚，再将10K电阻两端连接D2和GND。至于LED，我们使用的是D13数字引脚的自带LED(标有“L”),当然你也可以加个LED,长脚接D13,短脚接GND。



代码

你可以直接双击我们提供的L7.ino文件打开示例代码。脑叔更推荐大家还是自己尝试着一行行写出来。

```
1 //入门7示例1：用按钮开关控制LED
2 /*
3     描述：按钮被按下时LED会亮
4     by www.naozhendang.com
5 */
6 //const常量声明后不能更改
7 const int buttonPin = 2;      //按钮连接的引脚为D2
8 const int ledPin = 13;        //LED连接的引脚为D13
9
10 //声明一个整型变量来储存按钮的状态
11 //默认为关(0), 开就是(1)
12 int buttonState = 0;
13
14 void setup() {
15     pinMode(ledPin, OUTPUT);   //设置ledPin为输出模式
16     pinMode(buttonPin, INPUT); //设置buttonPin为输入模式
17 }
18
19 void loop() {
20     buttonState = digitalRead(buttonPin); //获取按钮引脚的状态
21
22     if (buttonState == HIGH) {    //如果按钮引脚值为HIGH
23         digitalWrite(ledPin, HIGH); //LED亮
24     } else {
25         digitalWrite(ledPin, LOW); //LED灭
26     }
27 }
```

验证代码，没有问题后上传即可。如果正确，按下按钮以后，Arduino UNO板载的LED就会亮起，松掉按钮，LED就会灭掉。

语法学习

const常量

```
6 //const常量声明后不能更改  
7 const int buttonPin = 2;  
8 const int ledPin = 13;
```

const关键字代表常量。它是一个变量限定符，用于修改变量的性质，使其变为只读状态。如果在声明变量前加上const，该变量虽然像任何相同类型的其他变量一样使用，但再也不能改变这个变量的值。如果尝试给一个const变量赋值，编译时IDE将会报错。

变量的范围

我们之前介绍过变量的概念。变量就是储存数值的小盒子，加上const就像不允许你换放在盒子里的东西，不加的话就可以随时换内容。不同的东西要放在不同盒子里。比如整数就要放在对应的整型变量盒子里。而且每种盒子所能容下的大小是有限制的。

有人会问问，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。

下表列出了程序中常见的变量数据类型和它们的范围：

变量类型	RAM	范围
boolean	1 byte	0/1, true/false
char	1 byte	-128 到 127
unsigned char	1 byte	0 到 255
int	2 byte	-32768 到 32767
unsigned int	2 byte	0 到 65535
long	4 byte	-2147483648 到 2147483647
unsigned long	4 byte	0 到 4294967295
float	4 byte	-3.4028235E38 到 3.4028235E38
double	4 byte	-3.4028235E38 到 3.4028235E38

```
digitalRead(buttonPin)
```

digitalRead函数是用来读取数字引脚状态，HIGH还是LOW(HIGH=1, LOW=0)。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2，由于前面做了声明，所以这里用buttonPin。并且把读到的信号传递给变量buttonState，用于后面进行判断。buttonState为 HIGH/1时，说明按键被按下了。buttonState为LOW/0，表明按键没被按下。

```
if (表达式){  
    执行语句;  
}
```

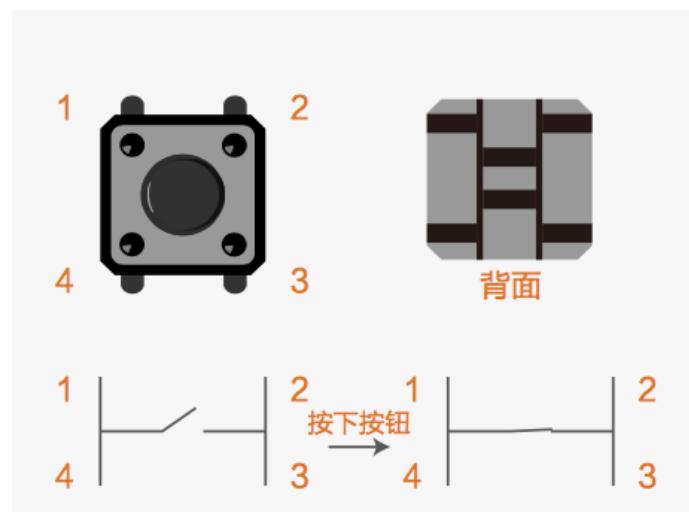
表达式是指我们的判断条件，通常为一些关系式或逻辑式，也可是直接表示某一数值。如果if表达式条件为真则执行if中的语句。表达式条件为假，则跳出if语句。

08 按钮去抖Debounce

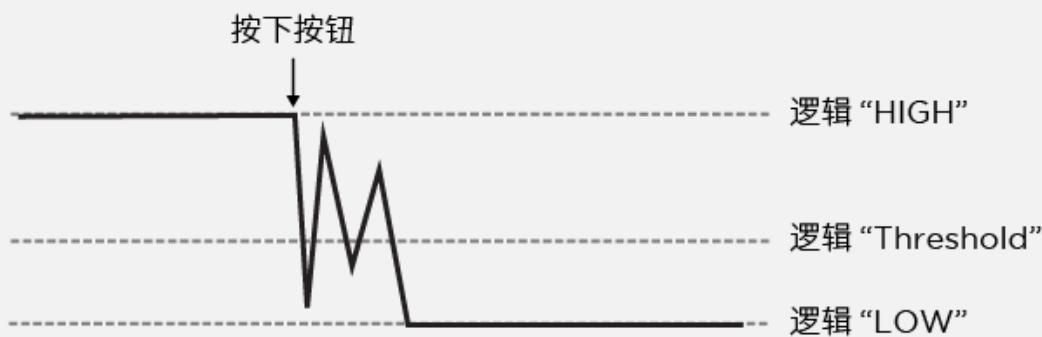
什么是去抖(Debounce)

前篇我们学习一下如何用按钮控制板载的LED，这篇教程中我们学一下按钮去抖(debounce)问题，什么是debounce以及怎么debounce。

按键一共有4个引脚，如右图一旦按下后，左右两侧就被导通了，而上下两端始终导通。

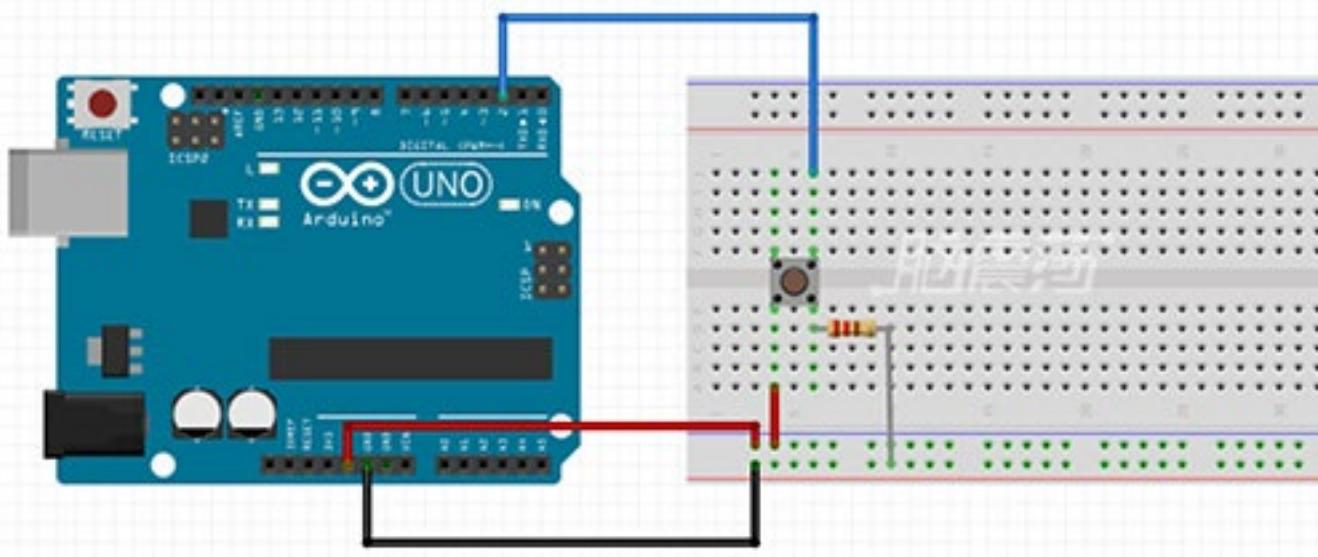


开关按钮在按下时，表面上只按了一下，但信号的传导并不是单纯的由1直接跳到0。这是由于机械触点断开/闭合时会有抖动，信号如图示会在HIGH和LOW之间抖动(bounce)。这种抖动对人来说是感觉不到的，但对计算机来说，则是完全可以感应到的。当按钮被按下时，单片机读到的信号可能会是111110110110000000...，而不是理想中的111111111000000000...，如此一来，虽然我们只按了一下按钮，可能会被电路误读成按了好几下按钮，而给人的感觉就是有时候按钮会不灵。Debounce的目的就是为了要除去信号在高低电位之间弹跳所造成的不正确输入。而今天Debounce的主要原理就是在较短时间里对输入进行再次确认来过滤抖动。



硬件连接

硬件的连接跟上一个教程一样。



代码

同样,这个例子的代码可以基于《入门7. 用按钮控制LED》示例代码修改而成。

验证代码,没有问题后上传即可。与《入门7. 用按钮控制LED》不同的是,这里我们按下按钮,LED的状态会改变(原来亮的话会灭掉,原来灭的话会亮起来)。

变量debounceDelay,代表debounce的时间间隔,如果这个值太大的话,正常按下的动作也会被识别成抖动过滤掉;但太小的话,降噪效果就会不明显,初始赋值50ms左右。

```
1 //入门8示例1: 按钮去抖Debounce
2 /*
3  * 描述:按钮被按下时LED会改变状态。原来亮的话会灭掉,原来灭的话会亮起来
4  * by www.naozhendang.com
5 */
6 //const常量声明后不能更改
7 const int buttonPin = 2;      //按钮连接的引脚为D2
8 const int ledPin = 13;        //LED连接的引脚为D13
9
10 int ledState = HIGH;         // 声明一个整型变量来储存LED的状态
11 int buttonState;             // 声明一个整型变量来储存按钮的状态
12 int lastButtonState = LOW;   // 声明一个整型变量来储存LED上一次的状态
13
14 // 用户储蓄时间的变量, 所以用long变量。
15 long lastDebounceTime = 0;   // 声明一个long型变量来储存上一次去抖的时间
16 long debounceDelay = 50;     // 声明一个long型变量来储存判断去抖的时间差
17
18 void setup() {
19   pinMode(ledPin, OUTPUT);    //设置ledPin为输出模式
20   pinMode(buttonPin, INPUT); //设置buttonPin为输入模式
21
22   digitalWrite(ledPin, ledState); //初始化LED的状态为HIGH, 即亮
23 }
24
25 void loop() {
26   //获取按钮引脚的状态,并存储到名为reading到整型变量里
27   int reading = digitalRead(buttonPin);
28
29   //下面要我们要检测按钮是否被按下
30   //如果被按下, 我们就开始计时
31   //直到超过指定的时间, 按钮状态稳定了再读取, 这样可以过滤抖动误差
32
33   //如果按钮跟上一次状态不一样, 即按钮状态改变了
34   if (reading != lastButtonState) {
35     lastDebounceTime = millis(); //记录当前的时间戳, 重新计时
36   }
37
38   //如果当前时间和开始时的时间差已经超过了设定的判断去抖的时间差,
39   if ((millis() - lastDebounceTime) > debounceDelay) {
40
41     //如果按钮当前状态改变
42     if (reading != buttonState) {
43       buttonState = reading; //储存状态
44
45       if (buttonState == HIGH) { //如果按钮按下
46         ledState = !ledState; //改变ledState的值
47       }
48     }
49   }
50
51   digitalWrite(ledPin, ledState); //改变LED的状态
52
53   //为下次循环做准备
54   //本次的按钮状态就是下次循环中上一次按钮的状态。
55   lastButtonState = reading;
56 }
```

millis()

millis()返回的并不是当前的时间,而是arduino从启动到当前所经过的毫秒数,但是它也是有最大限制的,大约记录42天后溢出,溢出就会归零。如果将当前的时间记录下来(时间戳),记为timeStamp,然后在之后某一时间进行一次(millis()-timeStamp)计算,就可以得出两次读数的时间差。一旦算出超过预定值,就可以执行后续语句什么的。在Arduino程序中,这种方法是最常见的计时方法。

09 互动交通信号灯

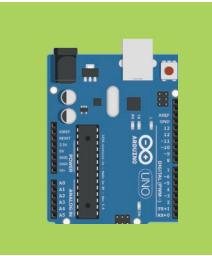
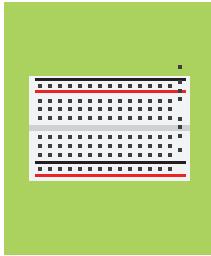
互动交通信号灯

到目前为止,我们学习了基本的LED控制, 使用模拟引脚读取按钮的状态, 并以此来控制LED, 以及按钮的去抖降噪。本篇教程里, 我来升级一下难度, 来模拟一个现实生活中的具体问题。需求是这样的:

我们设计互动的交通信号系统, 实现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。

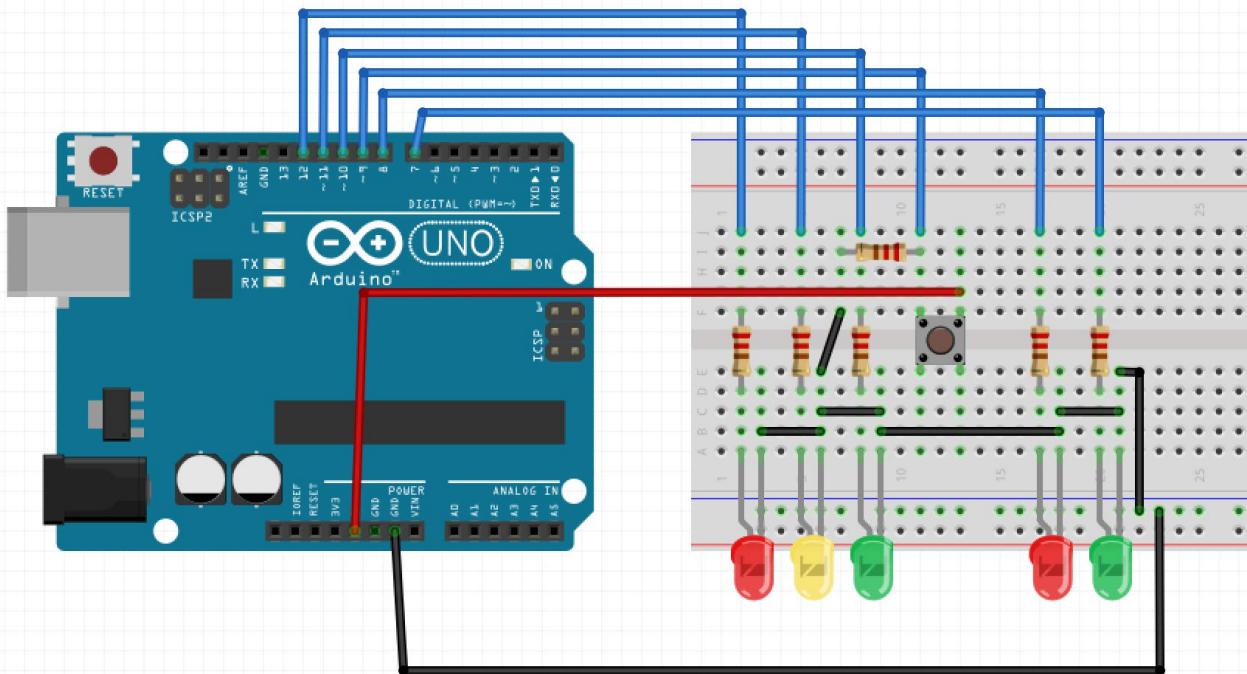
在这个项目中, 我们将要运用到之前所学的知识, 控制多个LED并学会自己创建函数, 难度显然提高了不少。

所需元件:

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1	
	220欧姆电阻 220Ω Resistors	x6		按钮开关 Push Buttons	x1	
	红色LED Red LED	x2	绿色LED Green LED	x2	黄色LED Yellow LED	x1

硬件连接

按照下图连接好LED、电阻和按钮。这次连线比较多，注意不要插错。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。



fritzing

代码

上传完成后，按下按钮看看效果是不是我们想要的。

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。

整段代码看起来很长很复杂，其实你把思路理清了，就一点也不难了。

```
1 //入门9示例：互动交通信号灯
2 /*
3  * 描述：实现行人按键请求通过马路的功能。
4  * 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。
5  * by www.naozhendang.com
6 */
7 int carRed = 12; //设置汽车灯
8 int carYellow = 11;
9 int carGreen = 10;
10 int button = 9; //按钮引脚
11 int pedRed = 8; //设置行人灯
12 int pedGreen = 7;
13 int crossTime = 5000; //允许行人通过的时间
14 unsigned long changeTime; //按钮按下后的时间
15
16 void setup() {
17     //所有LED设置为输出模式
18     pinMode(carRed, OUTPUT);
19     pinMode(carYellow, OUTPUT);
20     pinMode(carGreen, OUTPUT);
21     pinMode(pedRed, OUTPUT);
22     pinMode(pedGreen, OUTPUT);
23     pinMode(button, INPUT); //按钮设置为输入模式
24     digitalWrite(carGreen, HIGH); //车行
25     digitalWrite(pedRed, LOW); //人停
26 }
27
28 void loop() {
29     int state = digitalRead(button);
30     //按钮去抖
31     //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
32     if(state == HIGH && (millis() - changeTime) > 5000){
33         //调用变灯函数
34         changeLights();
35     }
36 }
37
38 void changeLights() {
39     digitalWrite(carGreen, LOW); //汽车绿灯灭
40     digitalWrite(carYellow, HIGH); //汽车黄灯亮
41     delay(2000); //等待2秒
42
43     digitalWrite(carYellow, LOW); //汽车黄灯灭
44     digitalWrite(carRed, HIGH); //汽车红灯亮
45     delay(1000); //为安全考虑等待1秒
46
47     digitalWrite(pedRed, LOW); //行人红灯灭
48     digitalWrite(pedGreen, HIGH); //行人绿灯亮
49
50     delay(crossTime); //等待一个通过时间
51
52     //闪烁行人灯绿灯，提示可过马路时间快到
53     for (int x=0; x<10; x++) {
54         digitalWrite(pedGreen, HIGH);
55         delay(250);
56         digitalWrite(pedGreen, LOW);
57         delay(250);
58     }
59
60     digitalWrite(pedRed, HIGH); //行人红灯亮
61     delay(500);
62
63     digitalWrite(carRed, LOW); //汽车红灯灭
64     digitalWrite(carYellow, HIGH); //汽车黄灯亮
65     delay(1000);
66     digitalWrite(carYellow, LOW); //汽车黄灯灭
67     digitalWrite(carGreen, HIGH); //汽车绿灯亮
68
69     changeTime = millis(); //记录自上一次灯变化的时间
70     //返回到主函数循环中
71 }
```

语法学习

```
unsigned long changeTime;
```

这是一个新的变量类型。我们之前介绍过各种变量的范围，
unsigned long数据类型，则不存储负数，存储的范围就从0到
4294967295.

试想如果我们使用一个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。

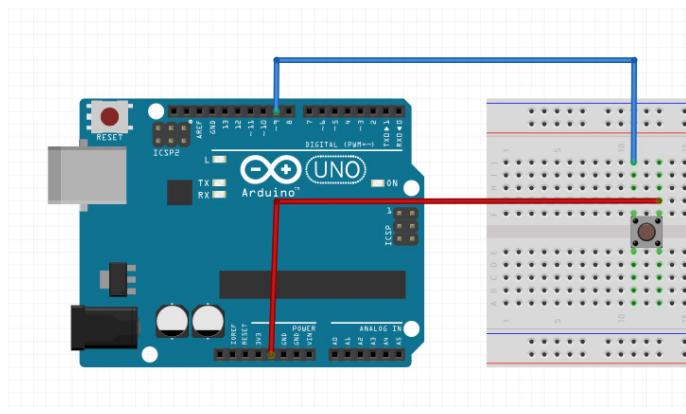
```
void changeLights() {  
    ...  
};
```

这段代码创建了一个名叫changeLights()的函数，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。

```
changeLights();
```

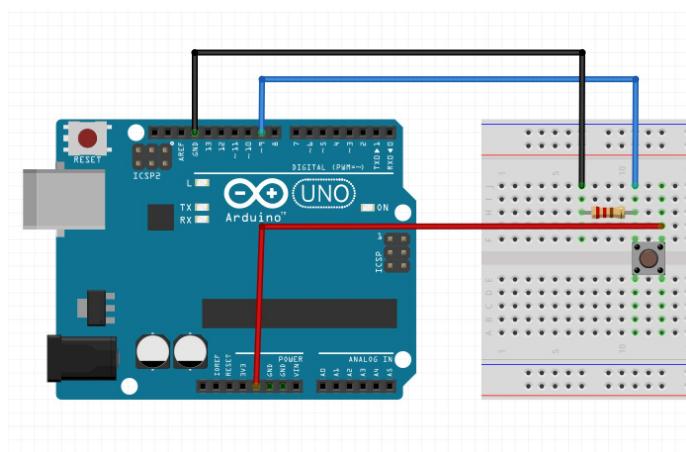
这样写就可以调用函数了。需要特别注意的是：函数调用时，函数名后面的括号不能省，要和所写的函数保持一致。changeLights()函数内部就不做说明了。

下拉电阻的概念

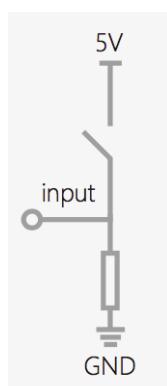
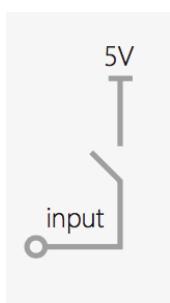


从字的含义着手，“下拉”可以理解为把电压往下拉，降低电压。

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。



可以从左边两张图对比看到，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。

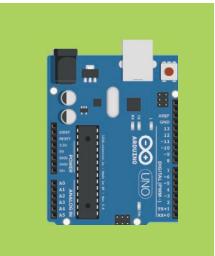
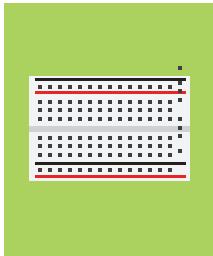
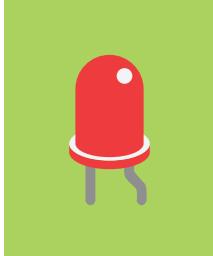


10 PWM和呼吸灯

如何控制LED的亮度？

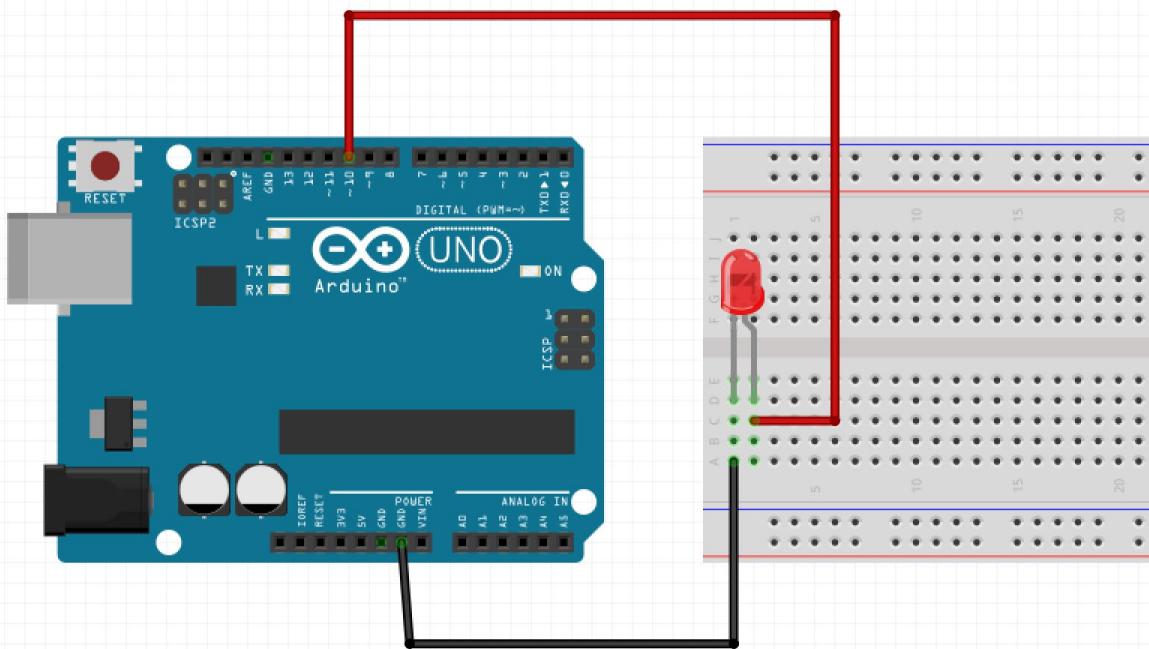
在前几篇入门教程中，我们学会了用digitalWrite(HIGH/LOW) 函数控制LED的亮和灭。那么你可能就会问了，除了这两种二元状态，就没有中间状态了么？如果我想控制LED的亮度呢？如何从亮慢慢地熄灭呢？在本篇教程中，我们就来学习一个非常重要的概念-- PWM。使用它就可以轻易地实现LED渐亮渐灭的呼吸效果。

所需元件：

	Arduino UNO 控制板	x1		面包板	x1
	220欧姆电阻	x1		红色LED	x1

硬件连接

硬件的连接和《入门5-经典仪式点亮LED》的电路是一样。这里我们仔细看一下Arduino板子上的几个引脚。会发现有些引脚旁边有“~”波浪线的标记。这个就是PWM的特殊标记，如果引脚带有这个标记就说明它有PWM功能。



fritzing

代码

代码上传完成后，我们可以看到LED会有个逐渐由亮到灭再到亮的一个缓慢过程，而不是直接的亮和灭，如同早起苹果电脑的呼吸灯一般，感觉在呼吸。

```
1 //入门10示例：PWM呼吸灯
2 /*
3     描述：利用PWM功能实现LED的渐亮渐灭
4     by www.naozhendang.com
5 */
6 int ledPin = 10;
7
8 void setup() {
9     pinMode(ledPin,OUTPUT);
10}
11
12 void loop(){
13     fadeOn(1000,5);
14     fadeOff(1000,5);
15}
16
17 void fadeOn(unsigned int time,int increament){
18     for (byte value = 0 ; value < 255; value+=increament){
19         analogWrite(ledPin, value);
20         delay(time/(255/5));
21     }
22}
23
24 void fadeOff(unsigned int time,int decreament){
25     for (byte value = 255; value >0; value-=decreament){
26         analogWrite(ledPin, value);
27         delay(time/(255/5));
28     }
29}
```

语法学习

```
24 void fadeOff(unsigned int time,int decreament){  
25   for (byte value = 255; value >0; value-=decreament){  
26     analogWrite(ledPin, value);  
27     delay(time/(255/5));  
28   }  
29 }
```

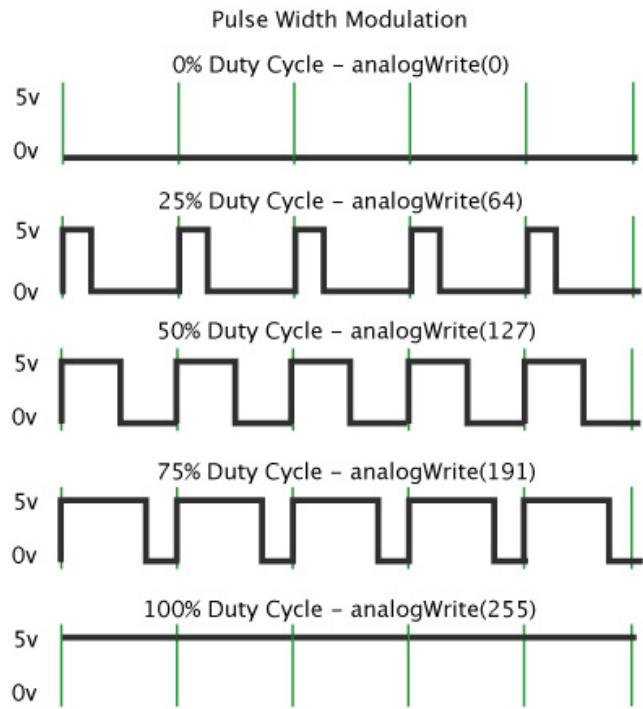
在这里，我们创建了一个叫fadeOff的无返回值函数。这个函数需要输入两个值，一个是无符号整型时间，另一个是整型的增量。fadeOff函数内部，执行了一个for循环。循环条件是value<255，变量的增量由 decreament决定。

analogWrite(ledPin, value);



这里又出现了一个新的函数analogWrite。是不是跟之前的digitalWrite很像。没错，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。

关于PWM



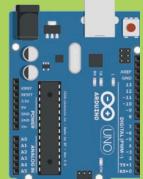
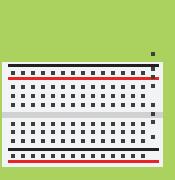
PWM, 全称叫Pulse Width Modulation, 一般翻译为脉宽调制 / 脉冲宽度调制”。简单来说, Arduino 只能产生5V和0V电压, 如果想要2.5V或者其他值的电压就要靠PWM, PWM通过单片机的数字输出的不同脉冲宽度, 来得到“模拟输出”需要的值。

11 控制多个LED

如何控制超多个LED？

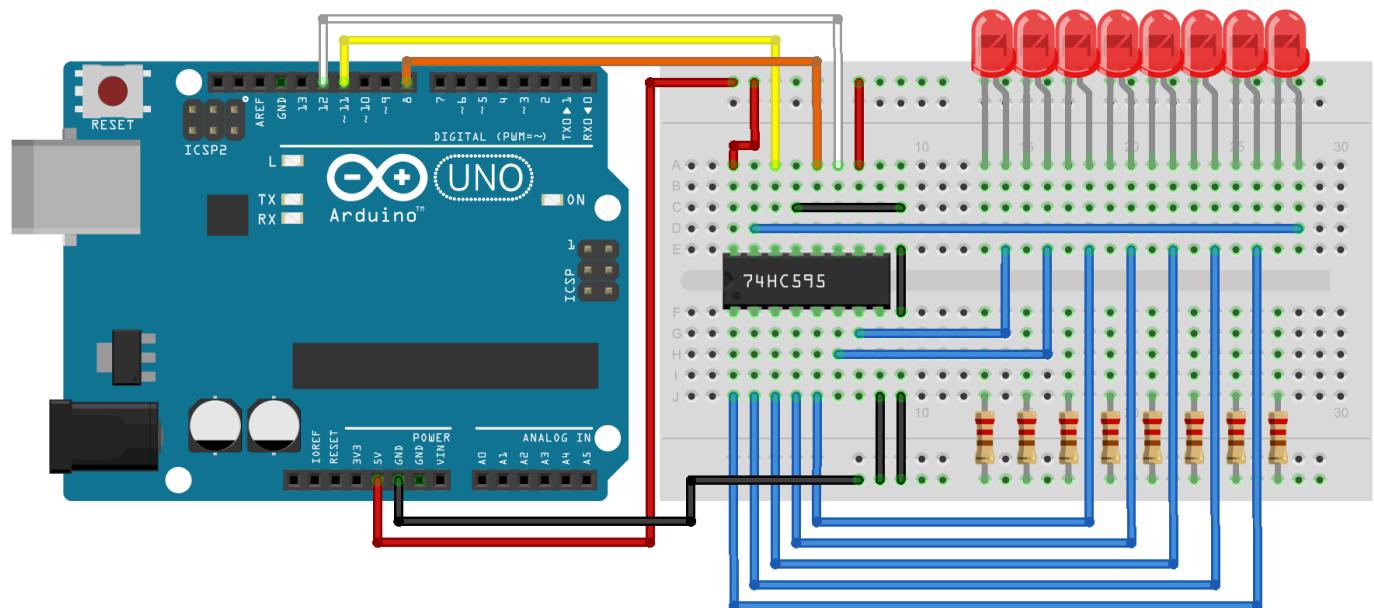
在《入门6-S.O.S求救信号灯》中，我们用到了5个LED，那如果我们要控制8个呢？我们数一下Arduino的数字引脚，从0~13个，够了。那如果我们要控制13个LED甚至更多LED呢？本篇，我们就来学习一下，如何用74HC595芯片来节省Arduino的I/O引脚。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1
	220欧姆电阻 220Ω Resistors	x8		红色LED Red LED	x8
	74HC595芯片 74HC595 Shift Register	x1			

硬件连接

如图用面包线把LED，74HC595芯片和Arduino连接好。注意再三检查连接是否正确。我们之后再来解释74CH595的原理。



代码

上传完程序到Arduino后，我们可以看到8个LED闪烁的美妙场景。这样我们成功用3个数字I/O口而不是8个I/O，这样节约了资源，提高了效率。

```
1 //入门J11示例：74HC595控制多个LED
2 /*
3     描述：只用三个Arduino引脚就可以控制8颗LED。
4     by www.naozhendang.com
5 */
6 //接 74HC595 的 ST_CP (pin 12,latch pin)
7 int latchPin = 8;
8 //接 74HC595 的 SH_CP (pin 11, clock pin)
9 int clockPin = 12;
10 //接 74HC595 的 DS (pin 14)
11 int dataPin = 11;
12
13 void setup() {
14     //将latchPin, clockPin, dataPin 设置为输出模式
15     pinMode(latchPin, OUTPUT);
16     pinMode(clockPin, OUTPUT);
17     pinMode(dataPin, OUTPUT);
18 }
19
20 void loop() {
21     //这个循环的意思是让a这个变量+1一直加到到256，每次循环都进行下面的活动
22     for (int a = 0; a < 256; a++) {
23         //数据shiftOut开始前，要把latchPin拉成低电位
24         digitalWrite(latchPin, LOW);
25
26         shiftOut(dataPin, clockPin, MSBFIRST, a);
27         /*
28             * 这个就是用MSBFIRST参数让0-7个针脚以高电平输出 (LSBFIRST 低电平) 是dataPin的参数,
29             * clockPin的参数是变量a, 前面我们说了这个变量会一次从1+1+到256, 是个十进制数,
30             * 输入到芯片后会产生8个二进制数, 达到开关的作用
31             */
32
33         //数据shiftOut完毕后，要把latchPin拉回成高电位
34         digitalWrite(latchPin, HIGH);
35         |
36         //延时500s
37         delay(500);
38     }
39 }
```

语法学习

```
digitalWrite(latchPin, LOW);  
shiftOut(dataPin, clockPin, MSBFIRST, val);  
digitalWrite(latchPin, HIGH);
```

这是使用74HC595的一般步骤。首先需要把latchPin拉低，这样才能shiftOut数据。

shiftOut函数是一个无返回值函数。需要四个输入值：

dataPin: 数据输出引脚, 数据的每一位将逐次输出。引脚模式需要设置成输出。

clockPin: 时钟输出引脚, 为数据输出提供时钟, 引脚模式需要设置成输出。

bitOrder: 数据位移顺序选择位, 该参数为byte类型, 有两种类型可选择, 分别是高位先入MSBFIRST和低位先入LSBFIRST。

val: 所要输出的数据值, 该数据值将以byte形式输出。

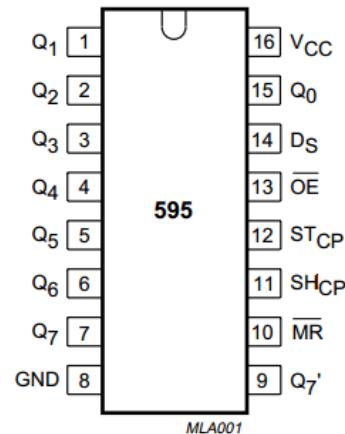
最后记得把latchPin拉高。

74HC595移位寄存器

74HC595 移位寄存器 (Shift Register) 是一颗八位 (8-bit) 串行输入 (serial-in)、串行或者并行输出 (serial/parallel-out) 的移位寄存器。使用这颗芯片，只需要占用控制器比如Arduino上的三个I/O 口，就可以同时控制8个输出。

你也可以把多个74HC595串联起来，就可以扩充出更多的引脚。比如2颗 74HC595 可以控制16个输出，4颗就是32个输出，以此类推。

74HC595常常被用来控制LED阵列，节省I/O口。



74HC595 芯片自身共有16个引脚，下面的图表是这些引脚的说明：

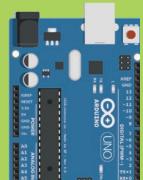
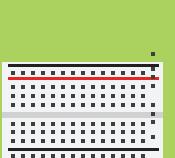
引脚编号	名称	说明
1~7, 15	Q0 ~ Q7	输出引脚
8	GND	接地
9	Q7'	串行数据输出
10	MR	主复位 (低电平时有效)
11	SH_CP	数据输入时钟线
12	ST_CP	输出存储器锁存时钟线
13	OE	允许输出 (低电平时有效)
14	DS	串行数据输入
16	Vcc	电源

12 光敏电阻和感光灯

光敏电阻和感光灯

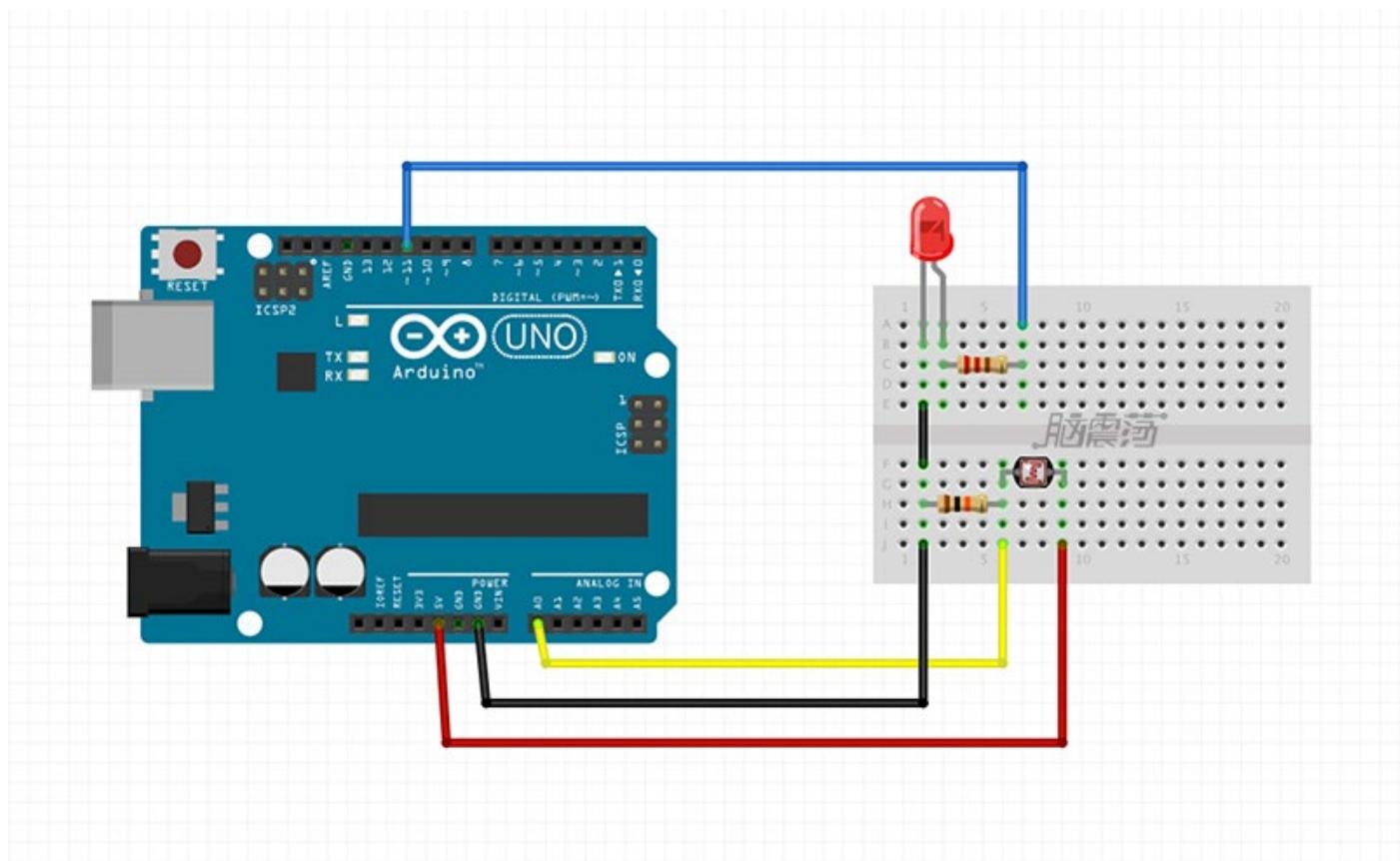
本篇教程里我们要做的是一种感光灯。它能随着光线明暗而选择是否亮灯。比如晚上睡觉的时候，感光灯感觉到周围环境变暗了，就自动亮起。到了白天，天亮后，感光灯自动关闭。 听上去是不是有点智能的味道，其实很简单。我们只要用光敏电阻就能实现了。光敏电阻，顾名思义就是对光敏感的电阻。它的阻值会随着光线的不同发生极大的改变，黑暗环境下，光敏电阻具有非常高阻值的电阻。光线越强，电阻值反而越低。这样我们就可以通过测量阻值来反应光线强度。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1
	220欧姆电阻 220Ω Resistors	x1		10K欧姆电阻 10KΩ Resistors	x1
	红色LED Red LED	x1		光敏电阻 Photo Cell Light Sensor	x1

硬件连接

如图连接好各个元件。注意光敏二极管和LED一样也是有正负极的，长脚为正极(+)，短脚为负极(-)。还需注意的与光敏二极管相连的电阻是 $10k\Omega$ ，而不是 220Ω 。



代码

上传完代码后，LED灯会亮起，用手机后置摄像头的闪光灯对着光敏电阻，这时你会发现LED灯神奇般地自动熄灭。但是，一旦慢慢拿走闪光灯，LED灯又会慢慢地亮起。

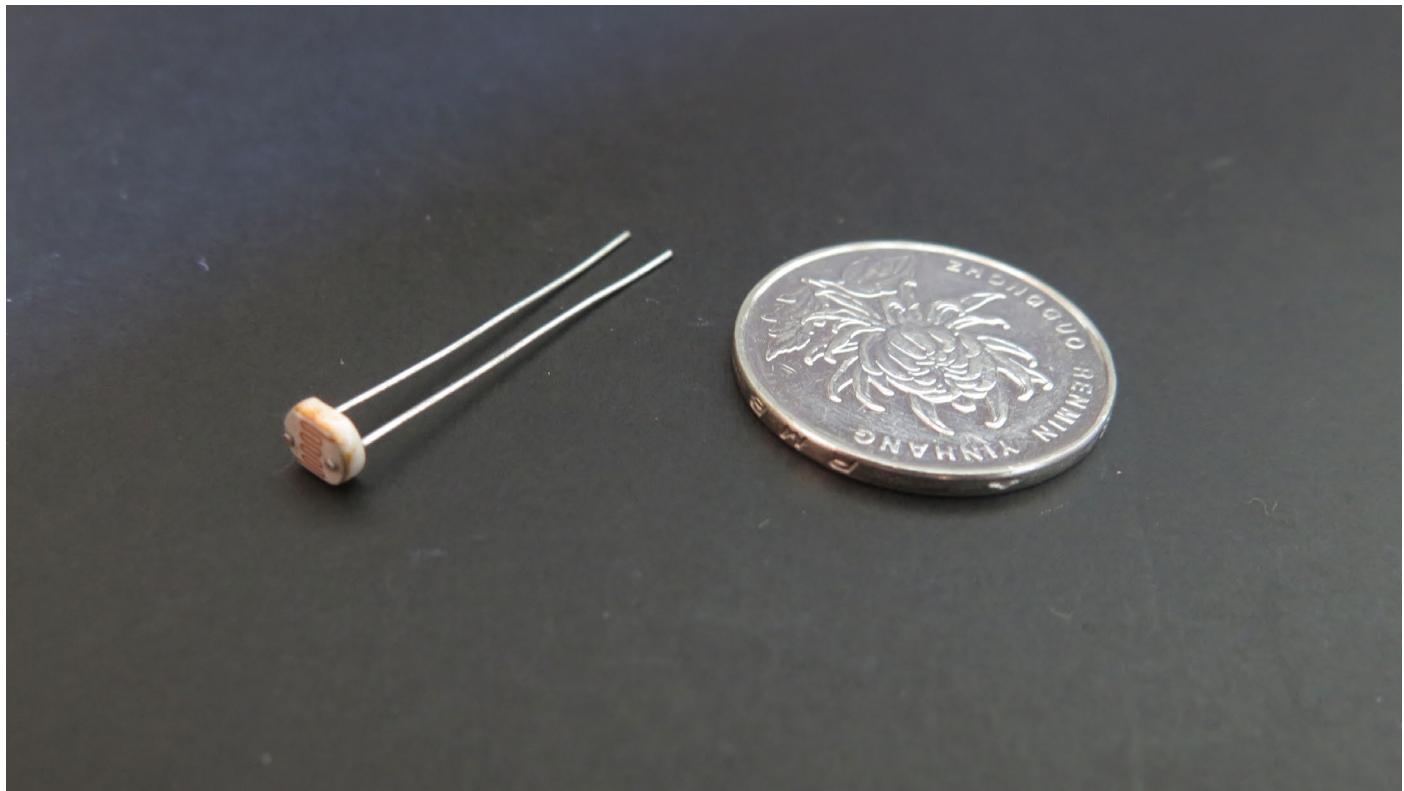
```
1 //入门12示例：光敏电阻和感光等
2 /*
3     描述：使用光敏电阻来感知光照强度，从而控制LED的亮度
4     by www.naozhendang.com
5 */
6 int LED = 11;                                //设置LED灯为数字引脚11
7 int val = 0;                                  //设置模拟引脚0读取光敏二极管的电压值
8 int LEDbrightness;                           //LED的亮度值
9
10 void setup(){                                 //串口波特率设置为9600
11     Serial.begin(9600);
12 }
13
14 void loop(){ {
15     val = analogRead(0);                      //读取电压值0~1023
16     Serial.println(val);                     //可以通过串口查看电压值
17
18     //由于光照越强，光敏电阻阻值越低，读取的电压会越低
19     //我们要用这个这个值来调整LED的亮度，先做一个反向
20     val = 1023 - val;
21     //用map函数将范围映射到0~255之间
22     LEDbrightness = map(val, 0, 1023, 0, 255);
23     analogWrite(LED, LEDbrightness); //调整LED的亮度
24
25     delay(100);
26 }
```

光敏电阻

光敏电阻，英文叫法很多，有Photocells, CdS Cells, Photoresistors或者Light Dependent Resistors (LDR)。顾名思义是一种对光敏感的电阻，常用来检测光。体积小，便宜，功耗低而且容易使用，所以常常可以在玩具，电子产品和家用电器中见到它们的身影。

光敏电阻本质上是一种电阻，其工作原理比较简单：当光照在弯弯曲曲的头部面上时其电阻值(单位欧 Ω)会随之改变。它们非常便宜，而且很多大小规格，但它们的缺点是非常不准确。即使同批生产出来的光敏电阻之间也常常会略有差异。它们的误差范围比较大，有时超过50%甚至更多。因此，切记它们不能用于测量准确的光照量(单位流明或者坎德拉)。只能用于判断基本的光线变化。比如“光线是亮了还是暗了？”，“面前是否有障碍物(挡住光)了？”这样的应用场景，光敏电阻就是你的不二之选！

尽管多数光敏电阻工作原理一样，但规格上都有点区别。所以在使用它们之前，请仔细查阅datasheet或者询问清楚。比如搞清楚最大电压，有光照时的阻值(光电阻)，没有光照时的阻值(暗电阻)等参数特性。

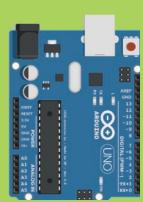
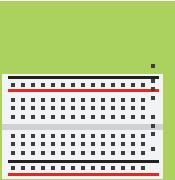


13 RGB LED

RGB LED

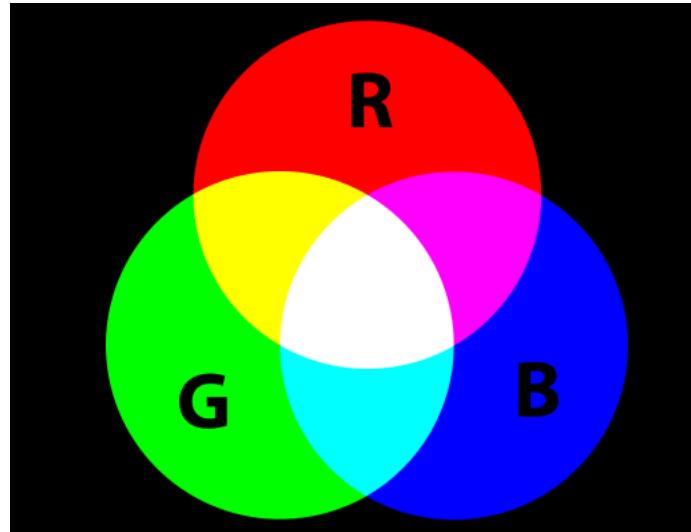
在之前我们接触地LED都是红、绿、黄等单色等LED，这篇教程里我们为大家介绍一种新等LED叫RGB LED。RGB LED可以通过Red、Green和Blue三原色等随意组合产生各种颜色。这个项目里我们就较大家如何控制RGB产生随机或者特定的颜色。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1
	220欧姆电阻 220Ω Resistors	x3		RGB三色LED RGB LED	x1

共阴/共阳RGB LED

RGB是RED(红)、GREEN(绿)、BLUE(蓝)色光三原色(又称加法三原色)的简称。之所以称为三原色，因为根据色光原理，你可以用红、绿、蓝三种光按不同比例混合出任意其他种颜色的光。比如把这三种色光等量混合可以调出白光。RGB LED可以说是色彩缤纷的电子显像世界的基础。比如说你面前的电脑屏幕或者手机屏幕就是由成千上万细小的RGB LED组成。



RGB LED市面上常见的有两种。一种叫共阳RGB，顾名思义就是共享一个正极(Common Anode)。另一种就叫共阴RGB，共享一个负极 (Common Cathode)。两者在使用时要区别：

- 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。共阴的话，反之，共用端为GND。
- 在颜色的调配上，共阳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。

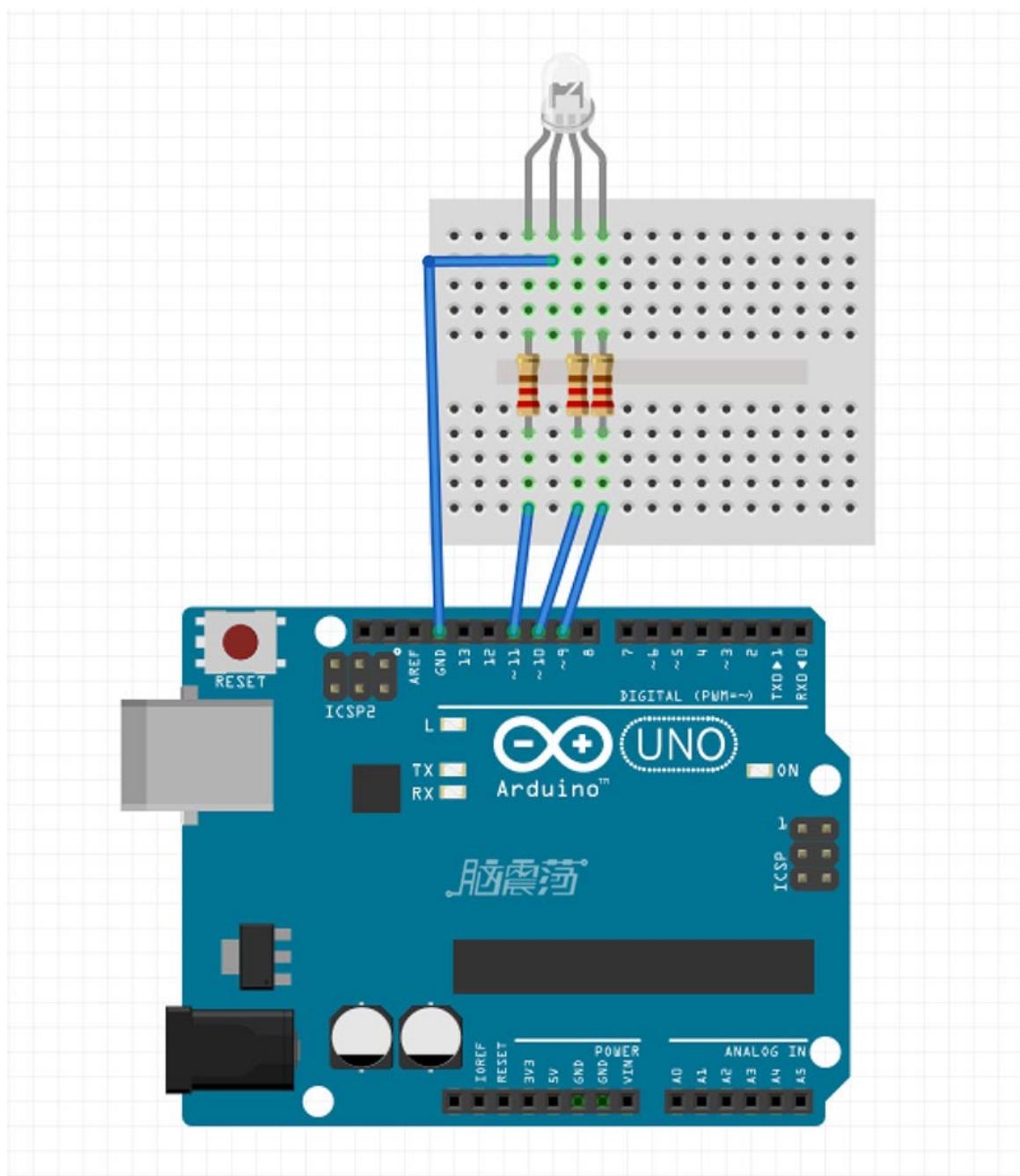


R	G	B	颜色
255	0	0	红色
0	255	0	绿色
0	0	255	蓝色
255	255	0	黄色
0	255	255	蓝绿色
255	0	255	紫红色
255	255	255	白色

不同LED的PWM值所组合产生的颜色

硬件连接

连接之前,先判别RGB是共阴还是共阳的。我们采用的是共阴RGB LED。如图,连接好线路。



代码

这段代码上传后，RGB LED会发出随机变化的颜色。

```
1 //入门13示例：RGB LED
2 /*
3     描述：RGB随机变化
4     by www.naozhendang.com
5 */
6 int redPin = 11;
7 int greenPin = 10;
8 int bluePin = 9;
9
10 //如果使用的是共阳RGB，可以反注释下面一行代码
11 // #define COMMON_ANODE
12
13 void setup()
14 {
15     pinMode(redPin, OUTPUT);
16     pinMode(greenPin, OUTPUT);
17     pinMode(bluePin, OUTPUT);
18 }
19
20 void loop()
21 {
22     //随机产生R, G, B三个色值
23     setColor(random(0,255), random(0,255), random(0,255));
24     delay(1000);
25 }
26
27 void setColor(int red, int green, int blue){
28     #ifdef COMMON_ANODE
29         red = 255 - red;
30         green = 255 - green;
31         blue = 255 - blue;
32     #endif
33     analogWrite(redPin, red);
34     analogWrite(greenPin, green);
35     analogWrite(bluePin, blue);
36 }
```

语法学习

random(min, max);



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。

注意: random()产生的随机数可能包括最小值, 但是一定不包括最大值。

14 库的概念和如何使用扩展库

什么是库

之前的几篇我们接触了LED，按钮等基本的元器件。接下去我们会接触到更多更高级的传感器或者执行器。但在晋阶前，我们有必要了解一下库的概念，以及如何使用各种库。

库(Library)在编程中是个基础的概念，简单地说，在编程时，有些特定的功能模块，函数，常量等常常被重复利用。为了方便二次开发和利用，将这些功能模块抽象打包，并提供特定的接口。这样的功能黑匣子，我们就可以称为“库(Library)”。

Arduino库也是类似，主要是一些传感器或者执行器的驱动文件和一些常用的功能函数。对于初学者而言，我们可以不用关心这些功能或者驱动具体是如何实现的，我们只要学会使用库里提供的函数就可以了。

一般来说，Arduino库有两种：

标准库

标准库是指Arduino IDE自带的一些常用功能，还有Examples里各种实例的支持库，比如伺服电机(Servo Motors)和LCD显示屏的驱动库。

标准库被预装在Arduino安装文件夹的“Libraries”子文件夹中。如果你在同一台电脑上安装了多版本的Arduino IDE，那么每个IDE不会共享标准库，每个IDE自带标准库。所以大多数情况下，最好不要轻易改动IDE自带的标准库，也不要在保准库文件夹中安装自定义的库。

扩展库

互联网上有各式各样传感器或者执行器的驱动库，你可以在Arduino Playground, Github或者其它地方直接下载到。

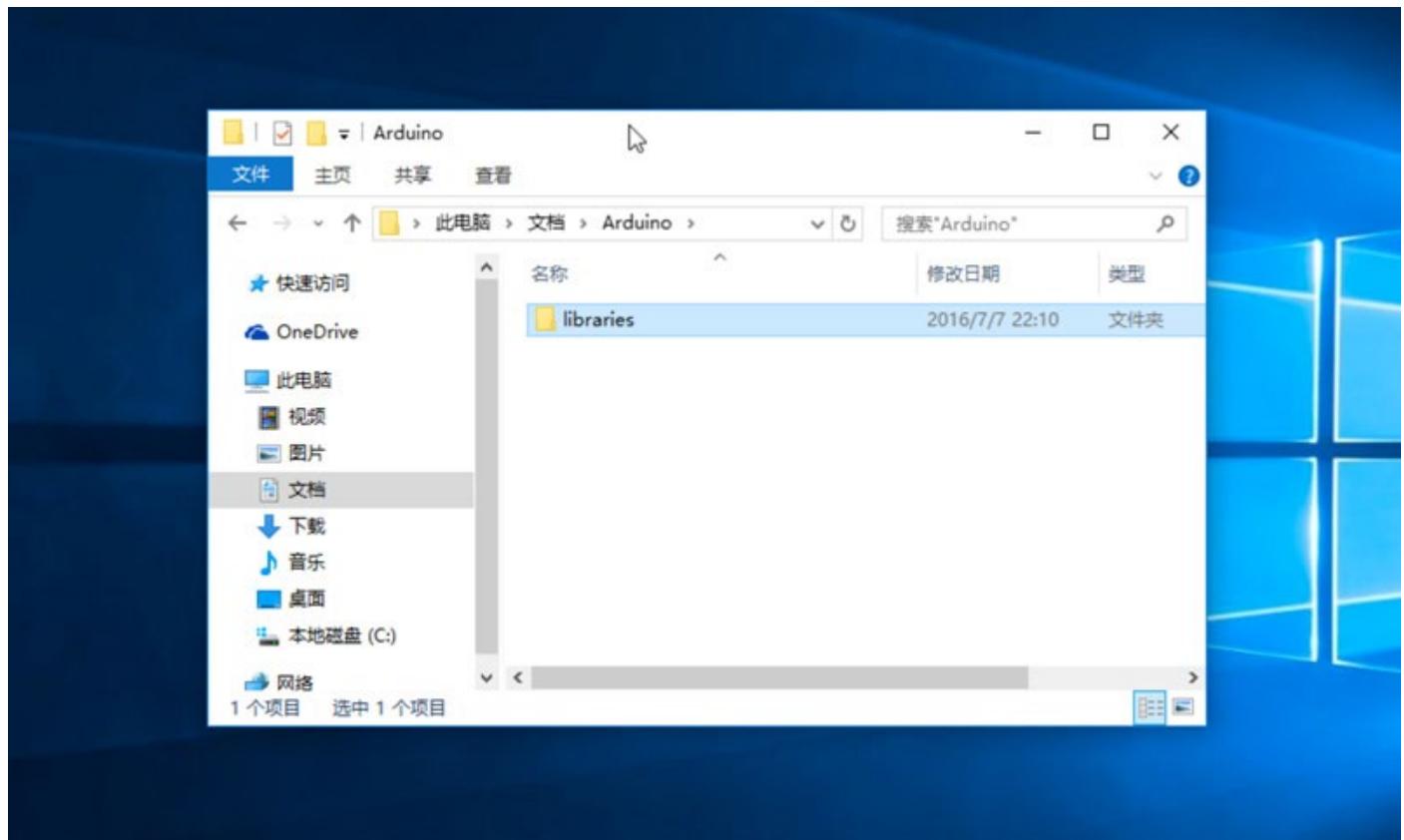
这些扩展库应该被安装在Sketchbook的Libraries文件夹里，这样它们可以被所有版本的IDE共享。当你安装新的IDE时，也不需要一一重新安装这些扩展库。

如何在Windows上安装扩展库

1. 找到Sketchbook和Libraries文件夹

把扩展库安装在正确的目录下至关重要，不然IDE编译代码时就会找不到相关的扩展库而报错。

第一步要做的就是找到Arduino的Sketchbook目录文件夹，这是Arduino IDE默认的保存sketch(即.ino)文件的地方。1.0.2版本后的Arduino IDE在安装时会自动在系统“Documents(我的文档)”里创建名为“Arduino”的文件夹(注意，不叫sketchbook哦)，里面还应有一个“libraries”子文件夹。



如果您想自定义sketchbook的文件夹,可以打开Arduino IDE,点击“文件>首选项(File>Preferences)”。然后你可以浏览选择自定义的文件夹目录。自定义目录后,你还需要到该目录里创建一个名为“libraries”的子文件夹。这个文件夹就是安装扩展库的地方。



2. 下载并安装扩展库

这里我们以Adafruit的SleepyDog库作为例子说明。我们在Github上找到库文件：

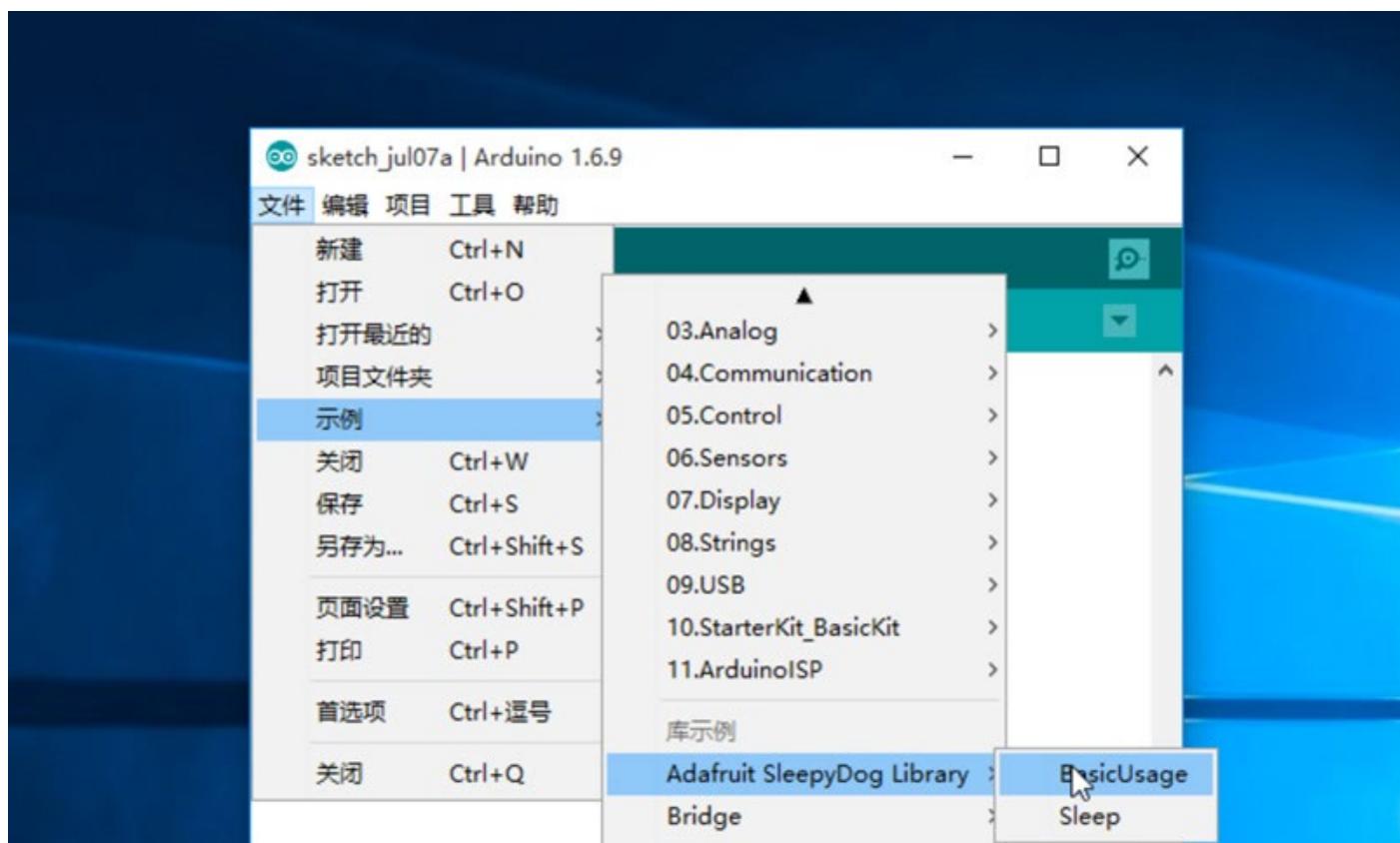
https://github.com/adafruit/Adafruit_SleepyDog

点击下载并且解压。然后将解压出来的名为“Adafruit_SleepyDog-master”的文件夹移至“Documents>Arduino>libraries”里，并重命名为“Adafruit_SleepyDog”。

The screenshot shows the GitHub repository page for Adafruit_SleepyDog. At the top, it displays the repository name 'adafruit / Adafruit_SleepyDog' and metrics: 20 stars, 9 forks, and 24 issues. Below this is a navigation bar with links for 'Code', 'Issues 2', 'Pull requests 0', 'Wiki', 'Pulse', and 'Graphs'. A summary section shows 15 commits, 1 branch, 2 releases, and 3 contributors. A dropdown menu indicates the branch is 'master'. There are buttons for 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download' (which is highlighted in green). The main content area lists repository files: '.github' (Add GitHub issue template), 'examples' (Wait for serial monitor in example (for native USB boards)), 'utility' (disable USB on '32u4 for ~200uA savings), and 'Adafruit_SleepyDog.cpp' (Add sleep and change name.). On the right, there's a 'Clone with HTTPS' section with a link to the repository URL and options to 'Open in Desktop' or 'Download ZIP'. A timestamp 'a year ago' is visible at the bottom right.

3. 重启Arduino IDE

关闭并重新打开Arduino IDE, 从“文件>示例(File>Examples)”找到“Adafruit SleepyDog Library”下面的“BasicUsage”实例。然后编译该实例, 若IDE没有报错, 编译成功的话那说明扩展库安装成功了。



4. 扩展库的使用

```
#include <Adafruit_SleepyDog.h>
```

扩展库的使用一般是引用库的头文件，比如Adafruit_SleepyDog这个扩展库：

但具体的用法，需要大家仔细查阅扩展库的文档。

如何在其他系统安装扩展库？

如果您的电脑系统是Mac或者是Linux系统，请访问脑震荡的在线教程参考如何安装库：

<http://naozhendang.com/t/learning-arduino-10-libraries>

15 玩转LCD显示

玩转LCD显示

前一篇，我们了解了什么是Arduino库，如何安装库。今天我们一方面来学习一下用LCD模块显示特定内容，另一方面实践一下标准库的使用。

所需元件：



LCD模块

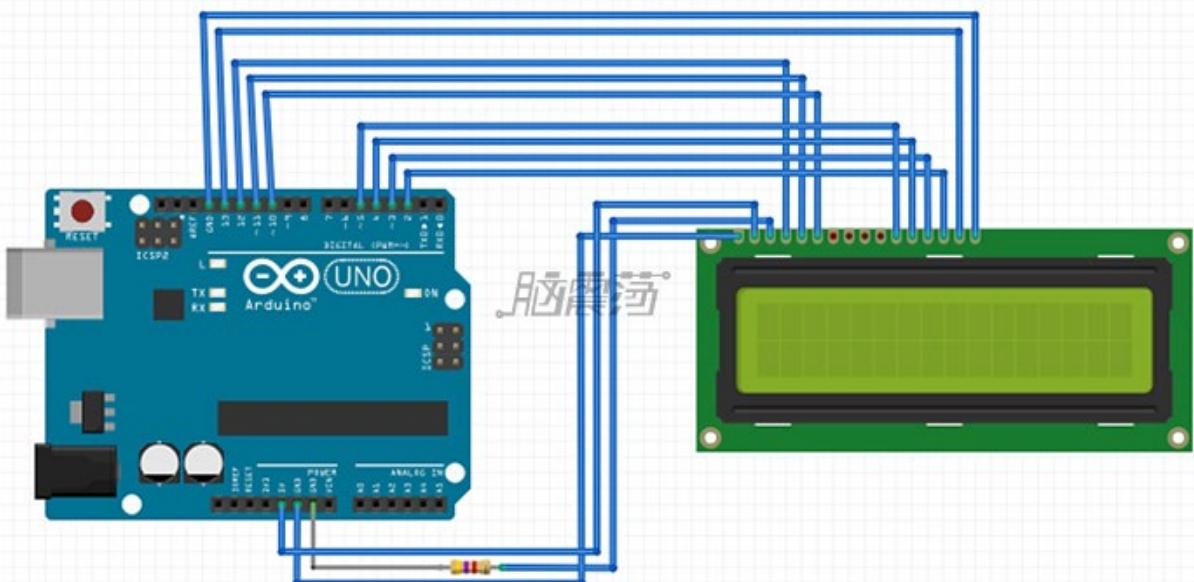
LCD模块种类超多，大小通常以“行数x字符”数来表示，比如2x16, 2x8, 4x20等等。大部分LCD模块都是由背光+液晶模块组成，利用Arduino驱动也很方便。接口方式有很多种：4位或者8位并行，3-wire，串口，I2C和SPI，LCD模块的价格也跟接口的难易程度成反比，也就是说采用4位或者8位并行的是最便宜的。市面上比较便宜的基本采用Hitachi HD44780芯片(或者兼容)方案，从外观上看，板上有16个针脚的就是这类LCD，16个pin有些是一排的，有些是双排的。今天用到的是国产的一块HJ1602A蓝色2x16的点阵LCD，采用4位并行接口，显示内容为2行，每行显示16个字符，每个字符大小为5×8点阵。

下面是16个针脚对应的Arduino的针脚以及它们的作用。

LCD模块针脚	Arduino针脚	作用
1 (GND)	GND	供电(-)
2 (VDD)	5v	供电(+)
3 (VO/Contrast)	GND	显示对比度控制, 接地表示最大对比
4 (RS)	12	寄存器选择
5 (RW)	11	读写模式选择
6 (E/enable)	10	允许寄存器写入
11 (DB4)	5	读写数据
12 (DB5)	4	读写数据
13 (DB6)	3	读写数据
14 (DB7)	2	读写数据
15 (BL1/Backlight1)	13	背光供电(+)
16 (BL2/Backlight2)	GND	背光供电(-)

硬件连接

按照图示连接好电路。



代码

这段代码上传后，LCD屏上会显示Hello World的字样。你可以试着修改代码，显示其他内容。

这里我们要到的是一个叫LiquidCrystal的库，这也是4位/8位并行接口LCD常用到的库，封装了很多常用的LCD功能。

这个实验也是LCD的最基本操作，复杂的咱以后慢慢学。

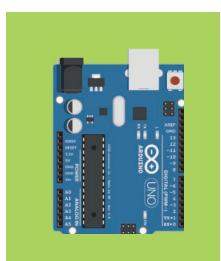
```
1 //入门15示例：玩转LCD显示
2 /*
3     描述:用1602LCD显示内容
4     by www.naozhendang.com
5 */
6 #include <LiquidCrystal.h> //引入LCD控制库
7
8 // rs (LCD 4号引脚) 连接Arduino D12引脚
9 // rw (LCD 5号引脚) 连接Arduino D11引脚
10 // enable (LCD 6号引脚) 连接Arduino D10引脚
11 // LCD 15号引脚连接Arduino D13引脚
12 // LCD d4, d5, d6, d7 分别连接Arduino 5, 4, 3, 2引脚
13
14 LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
15
16 int backLight = 13;      // D13引脚是用来控制LCD的背光的
17
18 void setup()
19 {
20     pinMode(backLight, OUTPUT);
21     digitalWrite(backLight, HIGH); // 打开LCD的背光功能
22     lcd.begin(16,2);           // 几列几行. 比如16x2LCD就是2行16列,
23     lcd.clear();               // 清屏
24     lcd.setCursor(0,0);        // 把光标移到0列0行
25     lcd.print("Hello World!"); // 显示内容
26     lcd.setCursor(0,1);        // 把光标移到0列1行(即第二行)
27     lcd.print("naozhendang.com"); // 显示内容
28 }
29
30 void loop()
31 {
32 }
```

16 电位器和舵机

电位器和舵机

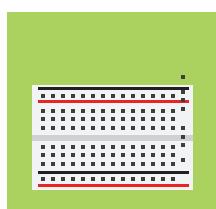
舵机是常用电机的一种，在航模，机器人关节里是非常重要的执行器。电位器也是一样，其实本质上就是我们初中物理学过的滑动变阻器。本篇教程里，我们就来深入了解一下这两种电子元件。

所需元件：



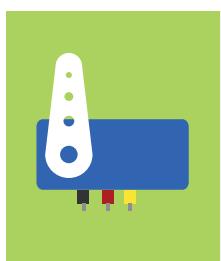
Arduino UNO
控制板 x1

Arduino Uno R3



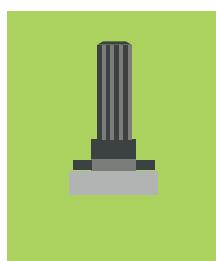
面包板 x1

Breadboard



9g 舵机(含配件) x1

9g Micro Servo
(w/ servo arms & screw)



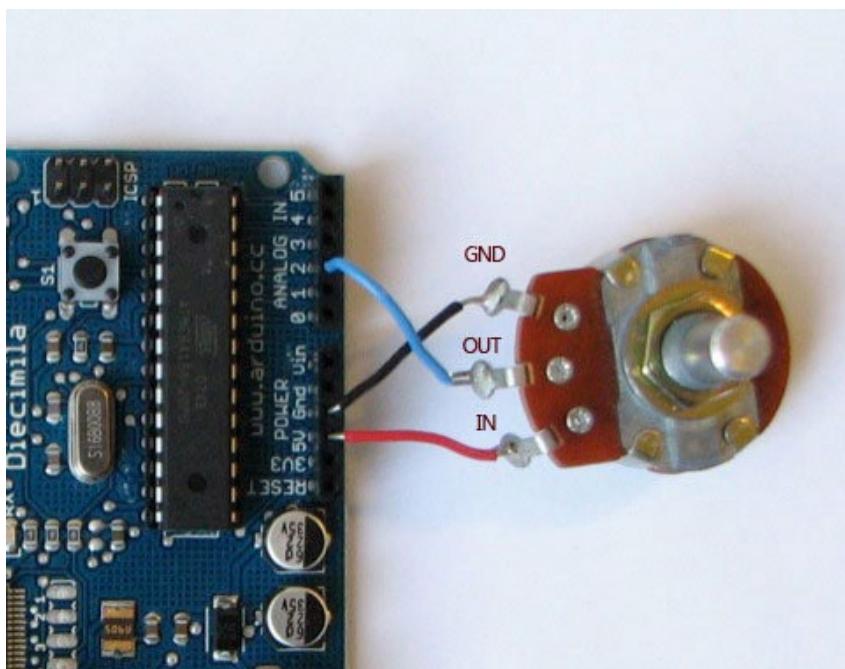
10K 电位器 x1

10K Log Potentiometer

电位器

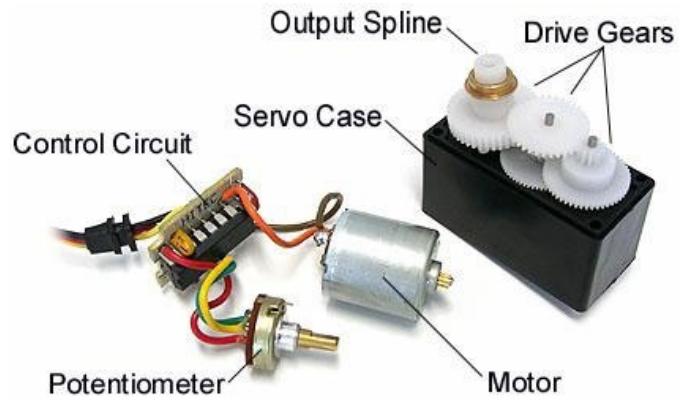
电位器(英文:Potentiometer, 通俗上也简称 Pot, 少数直译成电位计), 中文通常又称为可变电阻器(VR, Variable Resistor)或简称可变电阻, 是一种具有三个端子, 其中有两个固定接点与一个滑动接点, 可经由滑动而改变滑动端与两个固定端间电阻值的电子零件, 使用时可形成不同的分压比率, 改变滑动点的电位, 因而得名。至于只有两个端子的可变电阻器(或已将滑动端与其中一个固定端保持连接, 对外实际只有两个有效端子的)并不称为电位器, 只能称为可变电阻。(两个端子的可变电阻英文称 rheostat 或 variable resistor)。

电位器最常见的用途是各式音响声源设备里的音量控制或电子设备里的各式准位与功率等的控制, 也可以做为位置或角度的传感器, 或者是作为钨丝灯泡调光器或电热丝功率调节器的控制元件等。但某些用途, 如前述例子中的后者, 通常将电位器接成两端子可变电阻(rheostat)形式来使用。常见的碳膜或陶瓷膜电位器可以透过铜箔或铜片与印刷膜接触旋转或滑动产生于输出、输入端的不同电阻。较大功率的电位器则是使用线绕式。电位器有时会合并附带其他功能, 例如某些音量控制用的电位器附开关, 可兼作音量与电源开关的功能, 此时通常是在音量最小的一端附带关闭电源。

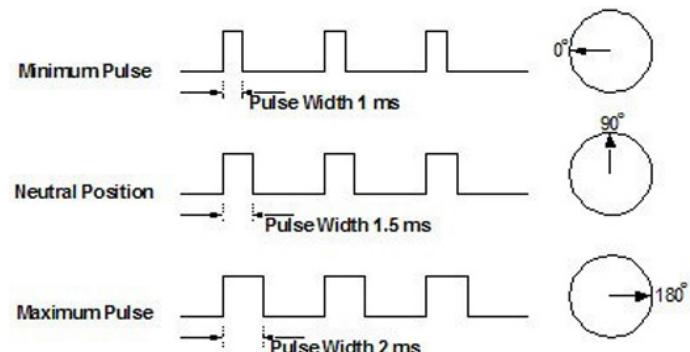


舵机

舵机(Servo)作为基本的输出执行构件,在微机电系统和航模被广泛应用。舵机是一种位置(角度)伺服的驱动器,适用于那些需要角度不断变化并可以保持的控制系统。最常见的舵机也称为伺服电机。拆开舵机,我们可以看到舵机基本是由齿轮组,直流电机,电位器和控制电路组成的。

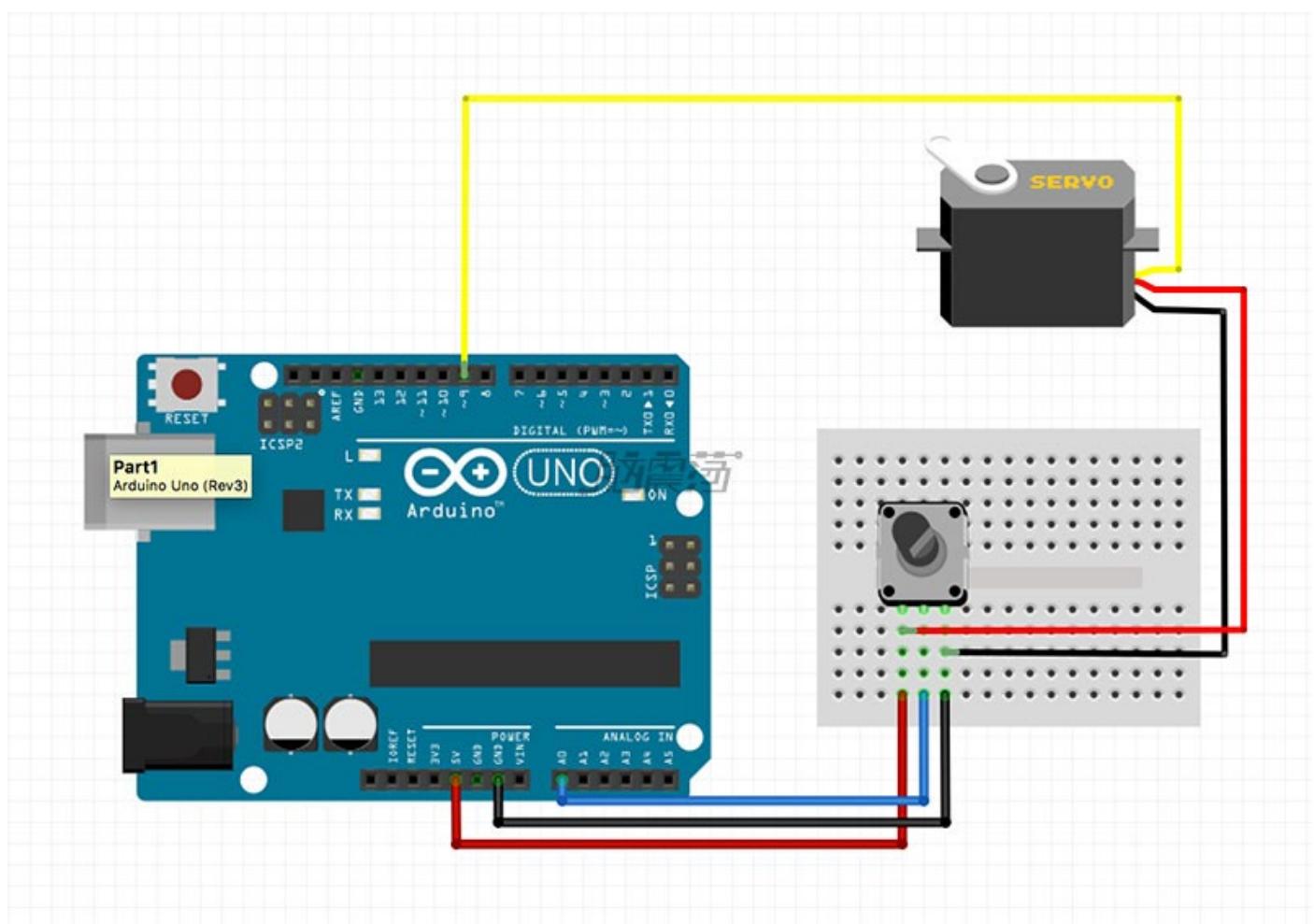


舵机的位置改变是靠控制PWM(脉冲宽度调制,在《入门10-PWM和呼吸灯》中我们介绍过)的占空比来实现的。一般使用的PWM周期是20ms,占空比0.5~2.5ms的正脉冲宽度和舵机的转角相对应(不同牌子的舵机对应的标准有差异)。



硬件连接

按照图示连接好电路。注意舵机有三根接线：黑色/棕色：GND 红色：正极 (~4.8-6V). 黄色/橙色/白色：信号线 (3-5V)



示例1

上传运行就可以了，舵机左右0-180度的摆动。

```
1 //入门16示例1：舵机
2 /*
3     描述：舵机左右摆动
4     by www.naozhendang.com
5 */
6 #include <Servo.h>      // 声明调用Servo.h库
7 Servo myservo;           // 创建一个舵机对象
8 int pos = 0;              // 变量pos用来存储舵机位置
9
10 void setup() {
11     myservo.attach(9);   // 将引脚9上的舵机与声明的舵机对象连接起来
12 }
13
14 void loop() {
15     for(pos = 0; pos < 180; pos += 1){ // 舵机从0°转到180°，每次增加1°
16         myservo.write(pos);           // 给舵机写入角度
17         delay(15);                  // 延时15ms让舵机转到指定位置
18     }
19
20     for(pos = 180; pos>=1; pos-=1) { // 舵机从180°回到0°，每次减小1°
21         myservo.write(pos);           // 写角度到舵机
22         delay(15);                  // 延时15ms让舵机转到指定位置
23     }
24 }
```

示例2

扭动电位器，舵机会随着电位器的转动而转动。

```
1 //八| J16示例2：电位器控制舵机
2 /*
3     描述：使用电位器控制舵机的位置
4     by www.naozhendang.com
5 */
6 #include <Servo.h>           // 声明调用Servo.h库
7 Servo myservo;               // 创建一个舵机对象
8
9 int potpin = 0;              // 连接到模拟口0
10 int val;                    // 变量val用来存储从模拟口0读到的值
11
12 void setup() {
13     myservo.attach(9);        // 将引脚9上的舵机与声明的舵机对象连接起来
14 }
15
16 void loop() {
17     val = analogRead(potpin);   // 从模拟口0读值，并通过val记录
18     val = map(val, 0, 1023, 0, 179); // 通过map函数进行数值转换
19     myservo.write(val);        // 给舵机写入角度
20     delay(15);                // 延时15ms让舵机转到指定位置
21 }
```

语法学习

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

在库相关的章节里，我们已经提到了Arduino IDE自带了一些库，叫标准库。Servo.h就是其中之一。你可以把它理解为一个黑匣子。它里面有很多控制舵机的函数。你在Arduino文件的头部添加这样一个语句就可以调用这个库里的函数。

```
Servo myservo;
```

引用了Servo.h，你还需要声明一个Servo的对象，对象的名字叫myservo。你可能同时控制多个舵机，声明一个对象来对应一个舵机。

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

我们还要告诉Arduino，这个myservo的对象在哪一个引脚上，使用attach(pin)函数。attach函数是已经定义在Servo.h里的，不需要重新定义。

库里面还有很多其他的函数，调用格式都类似：
对象名.函数名();

```
myservo.write(pos);
```

write(pos)函数也是Servo.h封装的一个函数。用于控制舵机的角度，pos是position的缩写，0-180之间的数值。

```
map(value, fromLow, from-  
High, 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。

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

* 两个范围中的“下限”可以比“上限”更大或者更小，比如：

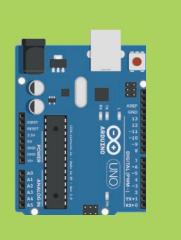
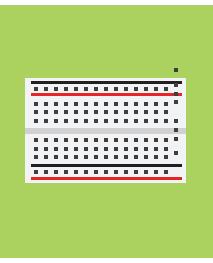
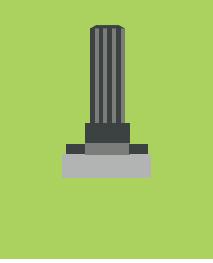
```
y = map(x, 1, 50, 50, -100);
```

17 步进电机的控制

步进电机

前一篇实验了如何用电位器控制舵机。本篇我们继续了解另一种常见的电机—步进电机。

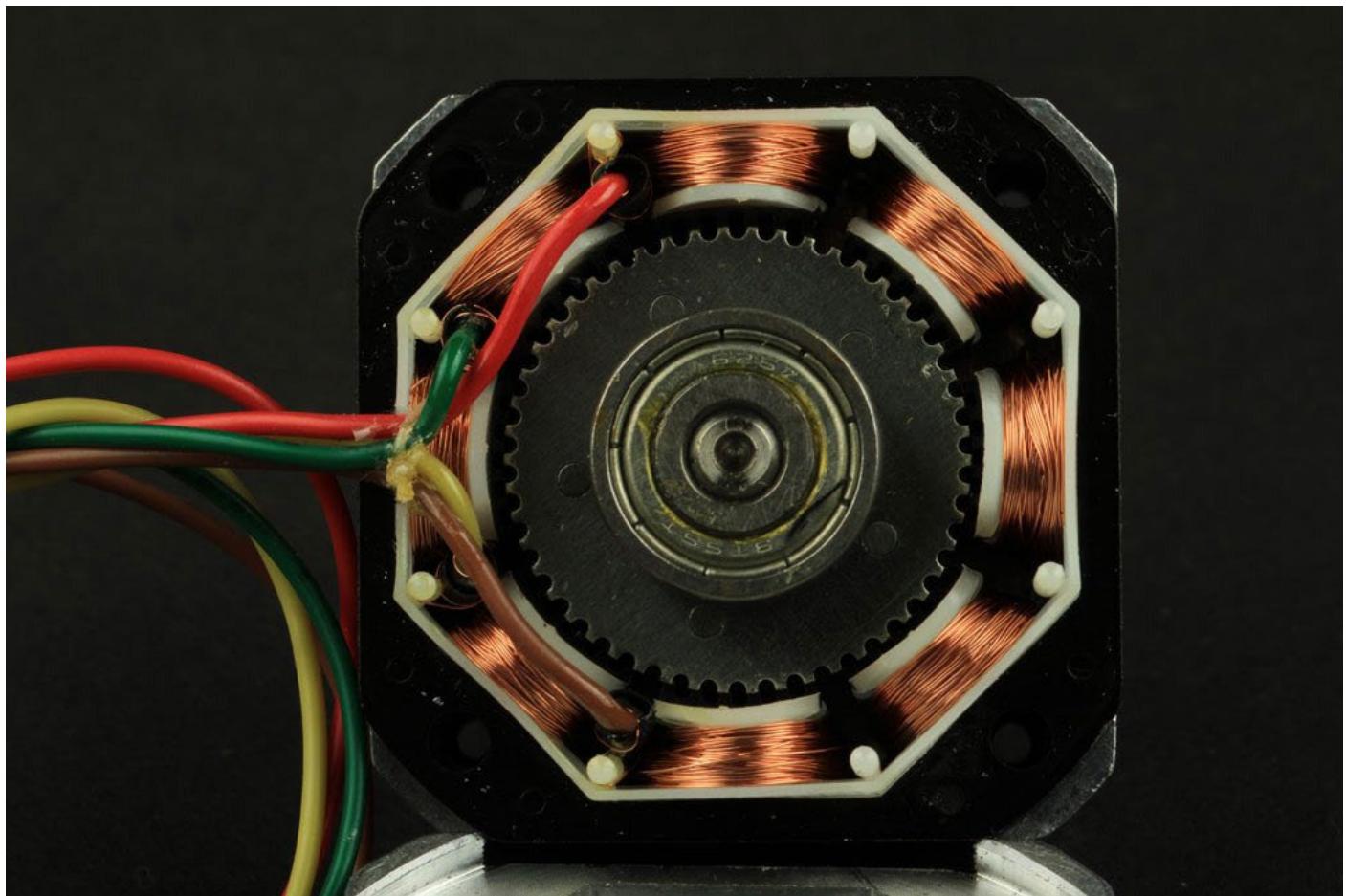
所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		ULN2003 步进电机驱动板 Stepper Motor Driver Board	x1
	面包板 Breadboard	x1		28BYJ-48 步进电机 (5V) 28BYJ-48 Stepper Motor	x1
	10K电位器 10K Log Potentiometer	x1			

什么是步进电机

步进电机(Stepper/Step/Stepping Motor),主要是依靠定子线圈序列通电,顺次在不同的角度形成磁场,推拉定子旋转。接触步进电机时会有很多容易混淆的概念。比如单极性、双极性、两相八线、四相八线等等。主要是由于线圈的接法不同,我们先简单地辨析一下:

按照电机驱动架构可分为单极性 (unipolar) 和双极性 (bipolar) 步进电机。所谓的极性,就是电流通过线圈绕组产生磁场的极性,单极性就是只有一个磁极,双极就是有两个磁极。四相,八相是指步进电机的相数,即步进电机内部的线圈组数。电机的相数不同,步进电机接收到每个脉冲信号的角度也不同。通过不同的极性,不同的相数,线圈接法会得到不同的电机性能。



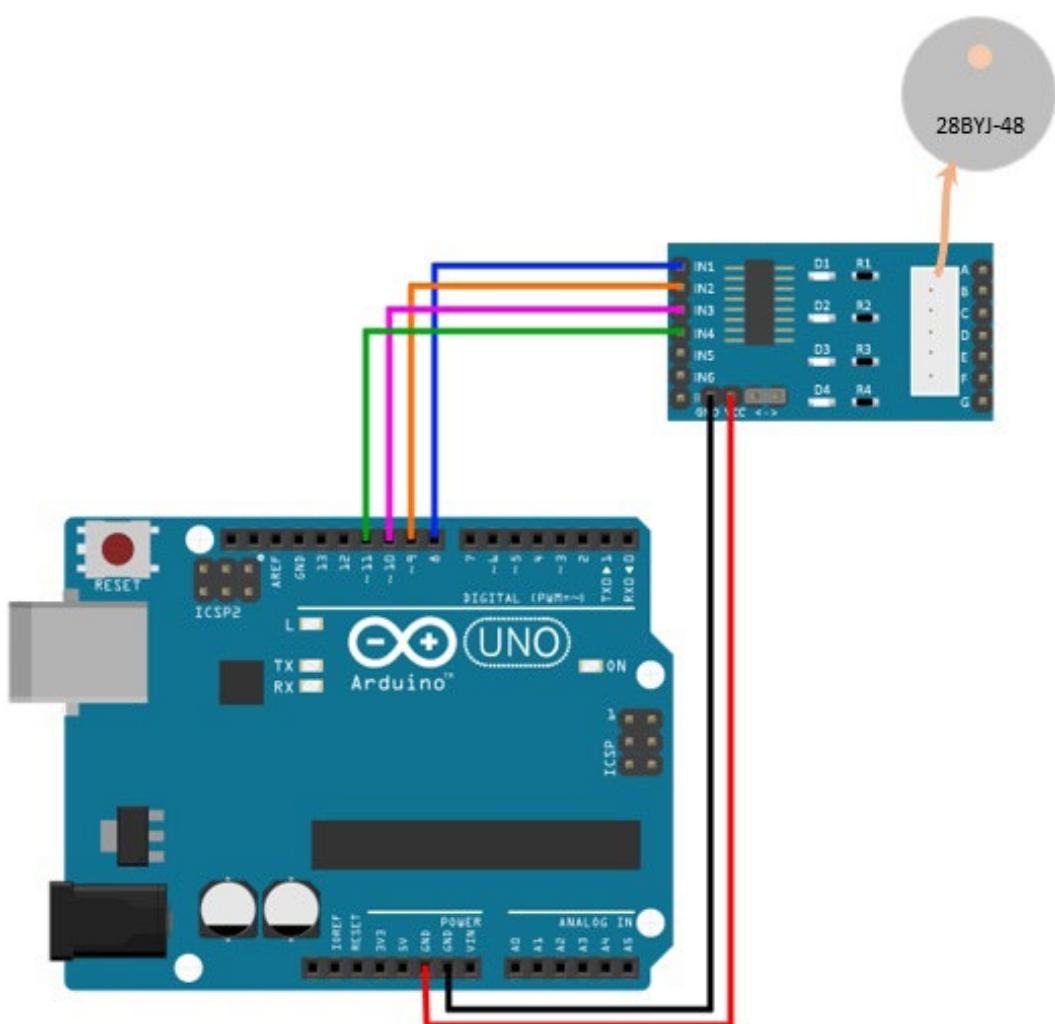
步进电机和伺服电机、舵机的区别

伺服电机不算是一种电机，而通常是包含电机、传感器和控制器的电机系统。舵机是个俗称，适用于航模上，其实是一种低端的但最常见的伺服电机系统，价格低廉但精度较低。



硬件连接

步进电机需要步进电机驱动板驱动，按照图示连接好电路。



代码

上传代码，和舵机示例2类似，当你转动电位器时，步进电机也会转动一定角度。

```
1 //入门17示例：步进电机
2 /*
3     描述：使用电位器控制步进电机
4     by www.naozhendang.com
5 */
6 #include <Stepper.h>
7 #define STEPS 100          //设置步进电机的一周的步数
8 Stepper stepper(STEPS, 8, 9, 10, 11); //创建步进电机的对象实例
9
10 int previous = 0;
11
12 void setup(){
13     stepper.setSpeed(30); // 设置步进电机的转速为30PRM
14 }
15
16 void loop(){
17     int val = analogRead(0); //获取电位器的读数
18
19     stepper.step(val - previous); //步进电机前进到目标位置
20
21     previous = val; // 储存当前读数，为下次计算做准备
22 }
```

语法学习

```
#include <Stepper.h>
```

Stepper.h是Arduino IDE自带的控制步进电机的标准库。

```
Stepper(steps, pin1, pin2)
```

```
Stepper(steps, pin1, pin2, pin3, pin4)
```

Stepper函数是用来创建步进电机实例的，共有两种以上用法。steps代表电机转一圈所用的步数。这个一般是步进电机出厂时就固定的。也可以通过步距角计算得出：

$$\text{转一圈的步数} = 360 / \text{步距角}$$

pin1,pin2,pin3,pin4是连接至步进电机的引脚。根据电机的引线数确定，pin3, pin4是可选的。

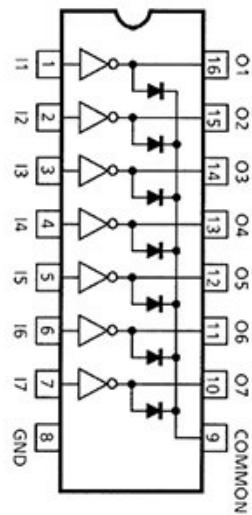
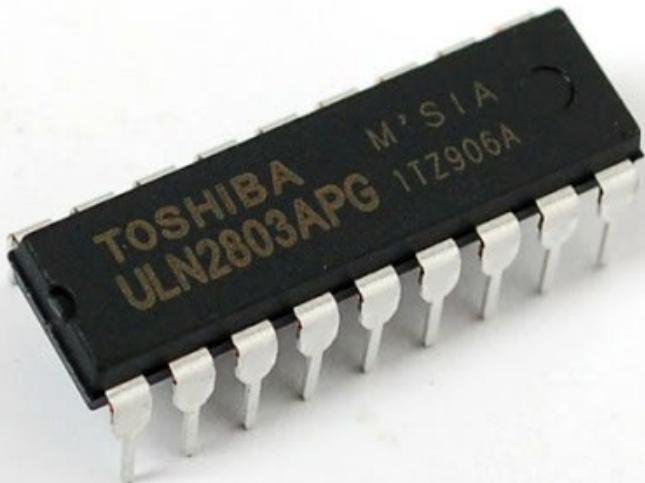
```
stepper.setSpeed(rpms);
```

setSpeed(rpms)函数是用来设置步进电机的速度，即每分钟该转的步数。

ULN2003芯片

步进电机往往需要特定的驱动系统来提供并控制电流脉冲。本教程中我们实验的步进电机属于功率较小的，我们使用的驱动电路使用的核心芯片就是ULN2003。ULN2003是由7个硅NPN复合晶体管组成，其主要作用就是把小电流变成大电流，ULN2003可以承受较高的工作电压和电流。

ULN2003一共有16个引脚，根据datasheet，有七个输入(IN)和对应的七个输出(OUT)，pin8接地，pin9接12V或者5V。



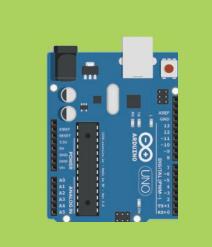
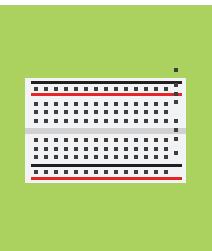
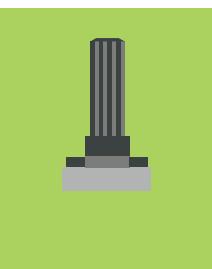
18 串口监视器的使用

串口监视器的使用

在《入门17：步进电机的控制》里，我们初步了解了步进电机的基础，以及如何通过Arduino和ULN2003芯片驱动步进电机。本篇教程在上一篇的电路上改动一下代码，来学习一下串口监视器的主要两种方式。

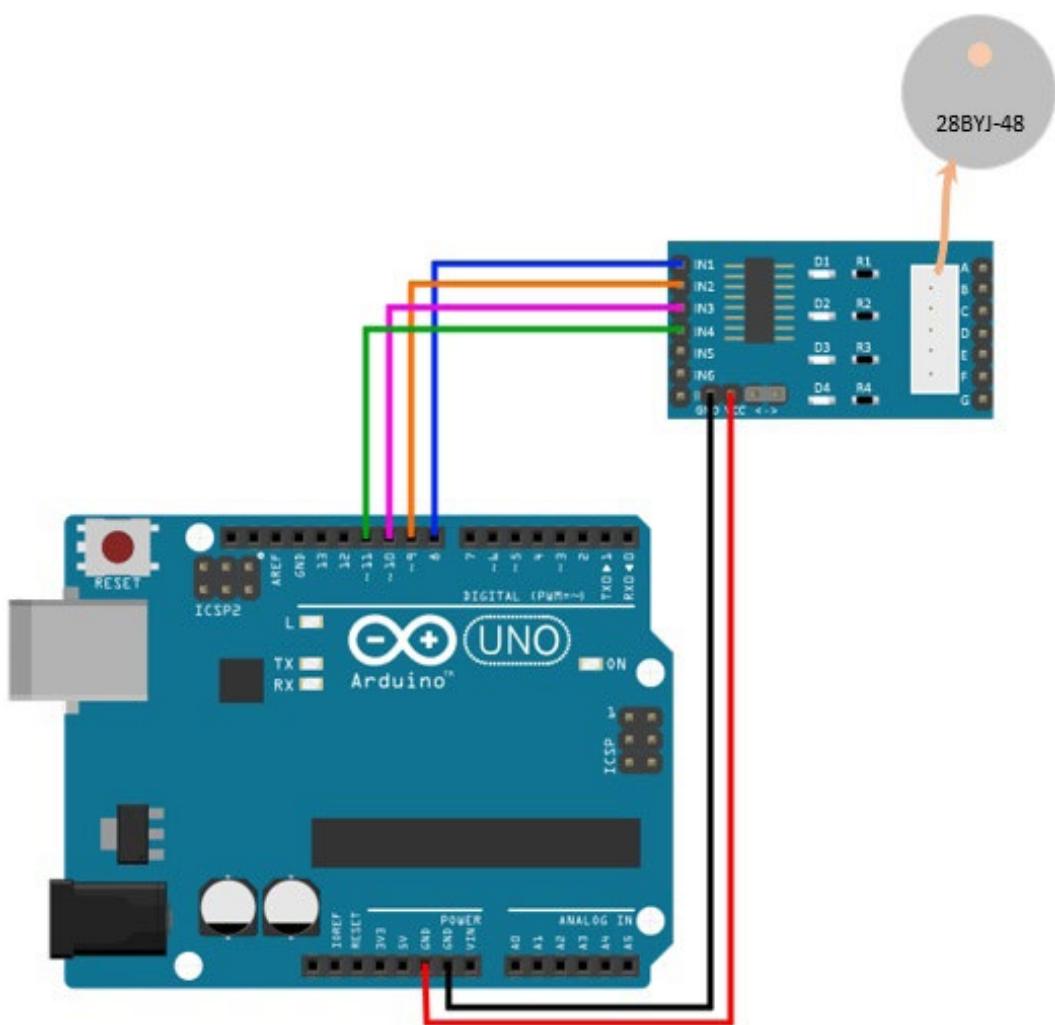
串口监视器(Serial Monitor)，顾名思义是用来监视串口通信的小工具。不要小看串口监视器，用好了，它就是帮你解决大多数烦恼的Swiss Army Knife(瑞士军刀)。Arduino和电脑是通过串口连接的，所有的数据通信都通过这个通道，所以串口监视器就像是这个通道上安装的安全摄像头一样。然你可以查看通信数据。Arduino IDE没有像其它高级IDE提供比较全面的debug工具，所以合理利用好串口监视器是Arduino代码debug的主要途径。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		ULN2003 步进电机驱动板 Stepper Motor Driver Board	x1
	面包板 Breadboard	x1		28BYJ-48 步进电机 (5V) 28BYJ-48 Stepper Motor	x1
	10K电位器 10K Log Potentiometer	x1			

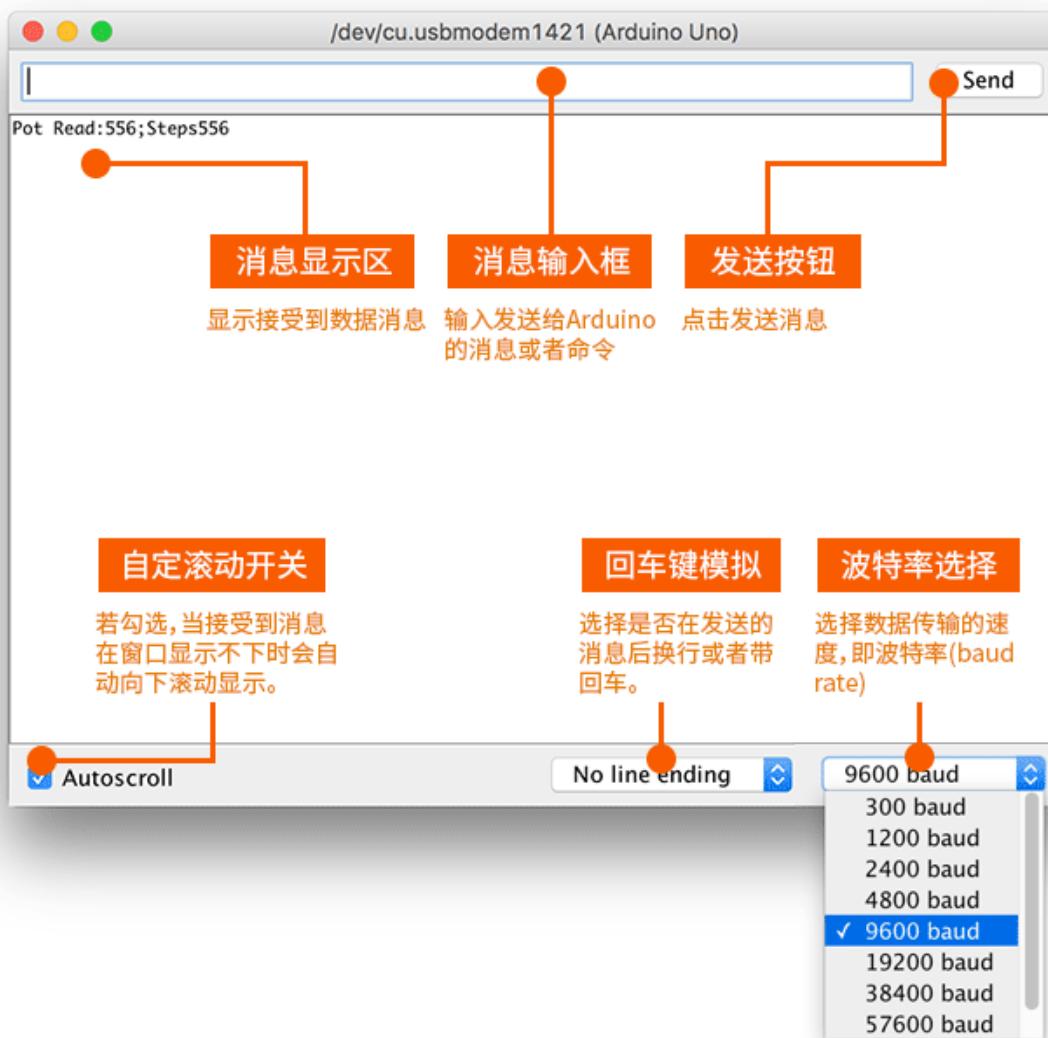
硬件连接

在本教程中，我们在《入门17：步进电机的控制》的代码基础上添加一些控制串口监视器的代码来学习。电路连接和《入门17：步进电机的控制》和一样。



串口监视器界面

首先，我们来熟悉一下如何打开串口监视器的UI，以及各个界面元素的作用。



如何打开串口监视器

串口监视器的入口在Arduino的工具栏右侧，带有一个搜索图标的按钮。点击这个按钮就会跳出串口监视器的窗口界面。



波特率(baud rate)

串口监视器窗口界面各选项都比较容易理解，唯独右下方的下拉菜单中的300baud~115200baud这个选项需要解释一下。这些选项是指波特率(baud rate)，即串口通信中每秒传输的数据位数。9600baud就是每秒能传输9600位(bit)。选择的波特率必须与Arduino代码中Serial.begin()函数定义的波特率一致，串口监视器才会显示正确的信息，不然会出现乱码。

示例1：串口监视器输出信息

这段代码的作用时，在串口监视器里输出两个数据，val代表A0引脚读取的电位器的值，val-previous代表步进电机应该移动的步数。

我们保存代码并上传成功后，我们打开串口监视器，选择对应的波特率，就能看到串口监视器里不断地打印出数据消息。这些数据就是Arduino运行时把数据发送给串口监视器显示的，这样我们就能动态地监测某些传感器和执行器实现的功能是否正常，或者程序当前执行到哪一步了，遇到了什么样的问题等等。

```
1 //入门18示例1：串口监视器
2 /*
3     描述：使用串口监视器输出消息
4     by www.naozhendang.com
5 */
6 #include <Stepper.h>
7 #define STEPS 100           //设置步进电机的一周的步数
8 Stepper stepper(STEPS, 8, 9, 10, 11); //创建步进电机的对象实例
9
10 int previous = 0;
11
12 void setup(){
13     stepper.setSpeed(30); // 设置步进电机的转速为30PRM
14     Serial.begin(9600); // 设置波特率为9600bps
15 }
16
17 void loop(){
18     int val = analogRead(0); //获取电位器的读数
19
20     stepper.step(val - previous); //步进电机前进到目标位置
21     Serial.print("Pot Read:");
22     Serial.print(val);
23     Serial.print(";Steps");
24     Serial.println(val - previous);
25
26     previous = val; // 储存当前读数，为下次计算做准备
27 }
```

示例1说明

`Serial.begin(9600);`

使用串口监视器必须首先使用`Serial.begin()`在`setup()`函数内定义波特率。

`Serial.println(val)`

`Serial.println(val,format)`

在你需要输出信息的地方使用`Serial.print()`或者`Serial.println()`, 二者的差别就是`Serial.println()`输出后会自动加上一个换行符”\n”。`Serial.println()`函数有两种用法。`val`是打印的值, 可以是任意数据类型。`format`是输出的数据格式。比如:

```
Serial.print(78, BIN) //输出 “1001110”
Serial.print(78, OCT) //输出 “116”
Serial.print(78, DEC) //输出 “78”
Serial.print(78, HEX) //输出 “4E”
Serial.print(1.23456, 0) //输出 “1”
Serial.print(1.23456, 2) //输出 “1.23”
Serial.print(1.23456, 4) //输出 “1.2346”
Serial.print(‘N’) //输出 “N”
Serial.print(“Hello world.”) //输出 “Hello world.”
```

示例2：串口监视器输入命令

可以看到串口监视器界面的顶部有一个输入框和“发送”按钮，没错，串口监视器不但能接受Arduino程序发送过来的数据，而且还能向Arduino程序发送命令。

我们再次修改代码。我们要实现的是直接通过串口监视器控制步进电机的移动的步数。

上传代码。然后在串口监视器的文本框里输入任意的整数，然后步进电机就会前进相应的步数。

```
1 //入门18示例2：串口监视器输入命令
2 /*
3     描述：使用串口监视器接受命令控制步进电机
4     by www.naozhendang.com
5 */
6 #include <Stepper.h>
7 #define STEPS 100           //设置步进电机的一周的步数
8 Stepper stepper(STEPS, 8, 9, 10, 11); //创建步进电机的对象实例
9
10 int previous = 0;
11
12 void setup(){
13     stepper.setSpeed(30); // 设置步进电机的转速为30PRM
14     Serial.begin(9600); // 设置波特率为9600bps
15 }
16
17 void loop(){
18     if (Serial.available())
19     {
20         int steps = Serial.parseInt();
21         stepper.step(steps);
22     }
23 }
```

示例2说明

```
if (Serial.available())
```

if (Serial.available())的判断的作用时告诉Arduino, 如果Arduino串口收到命令, 执行判断内部的代码, 否则什么也不做。

```
int steps = Serial.parseInt();
```

Serial.parseInt()的作用就是读取串口接收到的整数值。

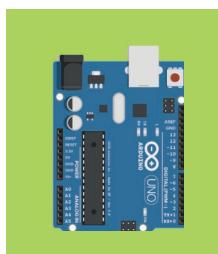
19 红外遥控灯

红外遥控灯

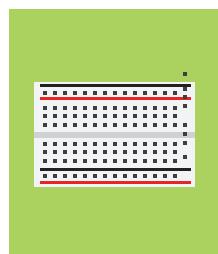
这节我们会接触一个新的元件——红外接收管。

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。

所需元件：



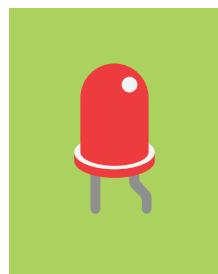
**Arduino UNO
控制板** ×1
Arduino Uno R3



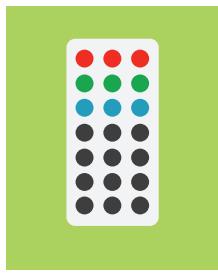
面包板 ×1
Breadboard



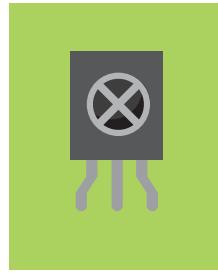
220欧姆电阻 ×1
220Ω Resistors



红色LED ×1
Red LED



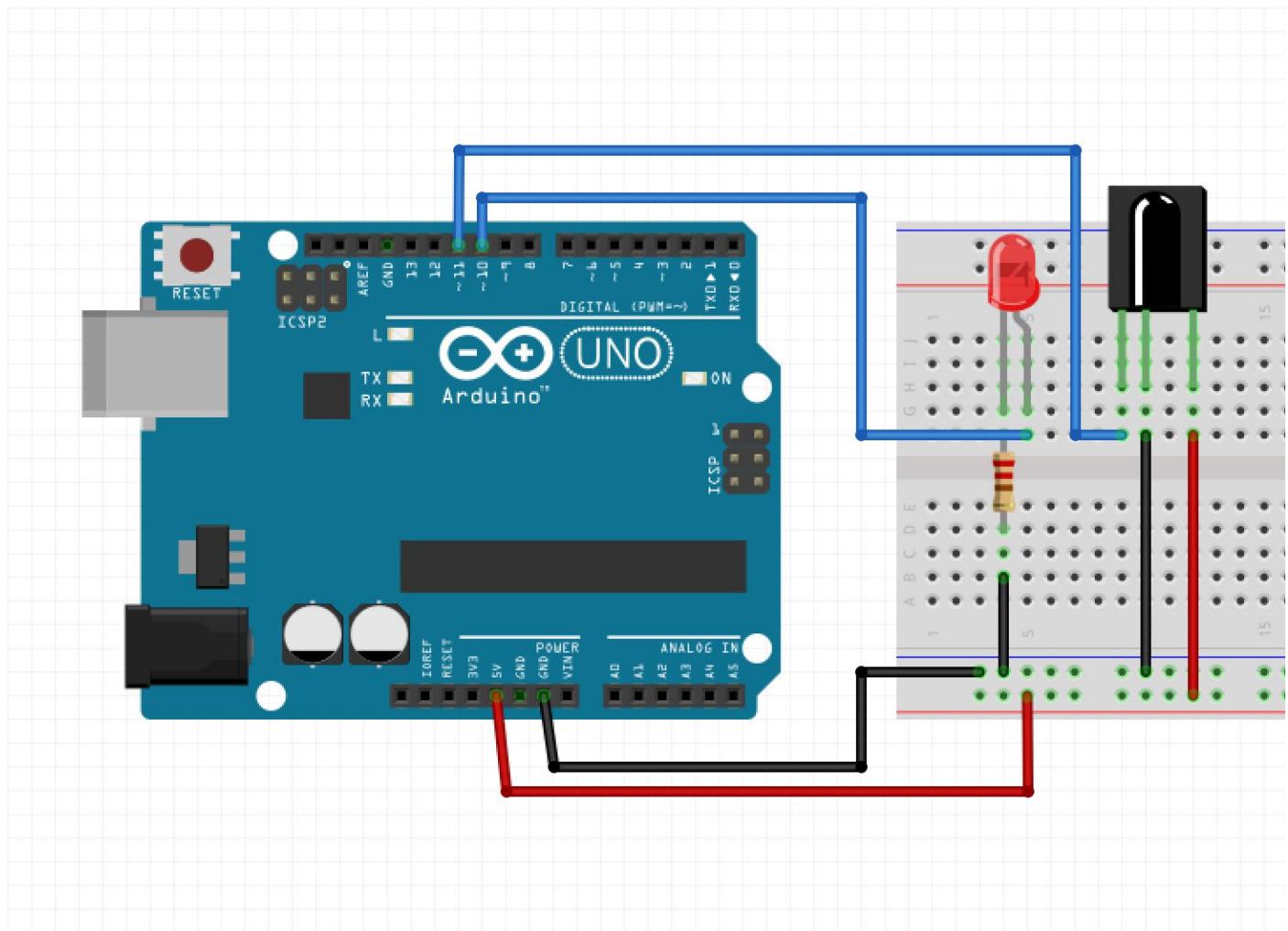
Mini红外遥控器 ×1
Mini IR Remote Controller



**HX1838一体化
红外接收头** ×1
HX1838 IR Receiver

硬件连接

红外接收头有三个针脚(Vout, GND和5V), Vout引脚连接Arduino的D11引脚。LED和220欧电阻串联接在Arduino的D10引脚上。



代码

上传代码，成功后。按下遥控器上的电源键，就可以控制LED的亮和灭了。

```
1 //入门19示例：红外遥控灯
2 /*
3     描述：用遥控器红外遥控LED的亮灭
4     by www.naozhendang.com
5 */
6 #include <IRremote.h>           //调用IRremote.h库
7 int RECV_PIN = 11;               //定义RECV_PIN变量为11
8 int ledPin = 10;                // LED - digital 10
9 boolean ledState = LOW;         // ledstate用来存储LED的状态
10 IRrecv irrecv(RECV_PIN);       //设置RECV_PIN（也就是11引脚）为红外接收端
11 decode_results results;        //定义results变量为红外结果存放位置
12
13 void setup(){                  //串口波特率设为9600
14     Serial.begin(9600);          //启动红外解码
15     irrecv.enableIRIn();          // 设置LED为输出状态
16     pinMode(ledPin,OUTPUT);
17 }
18
19 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
20     if (irrecv.decode(&results)) {
21         //接收到的数据以16进制的方式在串口输出
22         Serial.println(results.value, HEX);
23
24         //一旦接收到电源键的代码， LED翻转状态， HIGH变LOW， 或者LOW变HIGH
25         if(results.value == 0xFD00FF){
26             ledState = !ledState;           //取反
27             digitalWrite(ledPin,ledState); //改变LED相应状态
28         }
29     }
30     irrecv.resume(); // 继续等待接收下一组信号
31 }
32 }
```

语法学习

ledState = !ledState

“!”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。

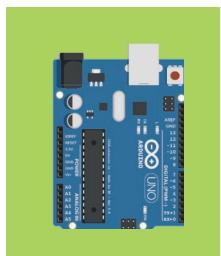
20 震动传感器

震动传感器

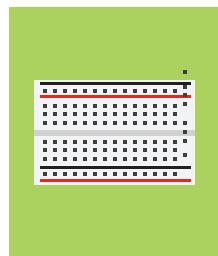
这节我们会介绍一种新的传感器 -- 滚珠开关

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。 所以滚珠开关常常被用来检测震动。在本次试验中，我们将学习如何用滚珠开关来控制LED。

所需元件：



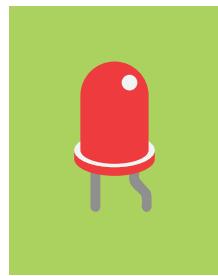
**Arduino UNO
控制板** x1
Arduino Uno R3



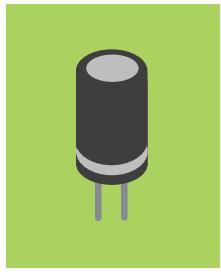
面包板 x1
Breadboard



220欧姆电阻 x2
220Ω Resistors



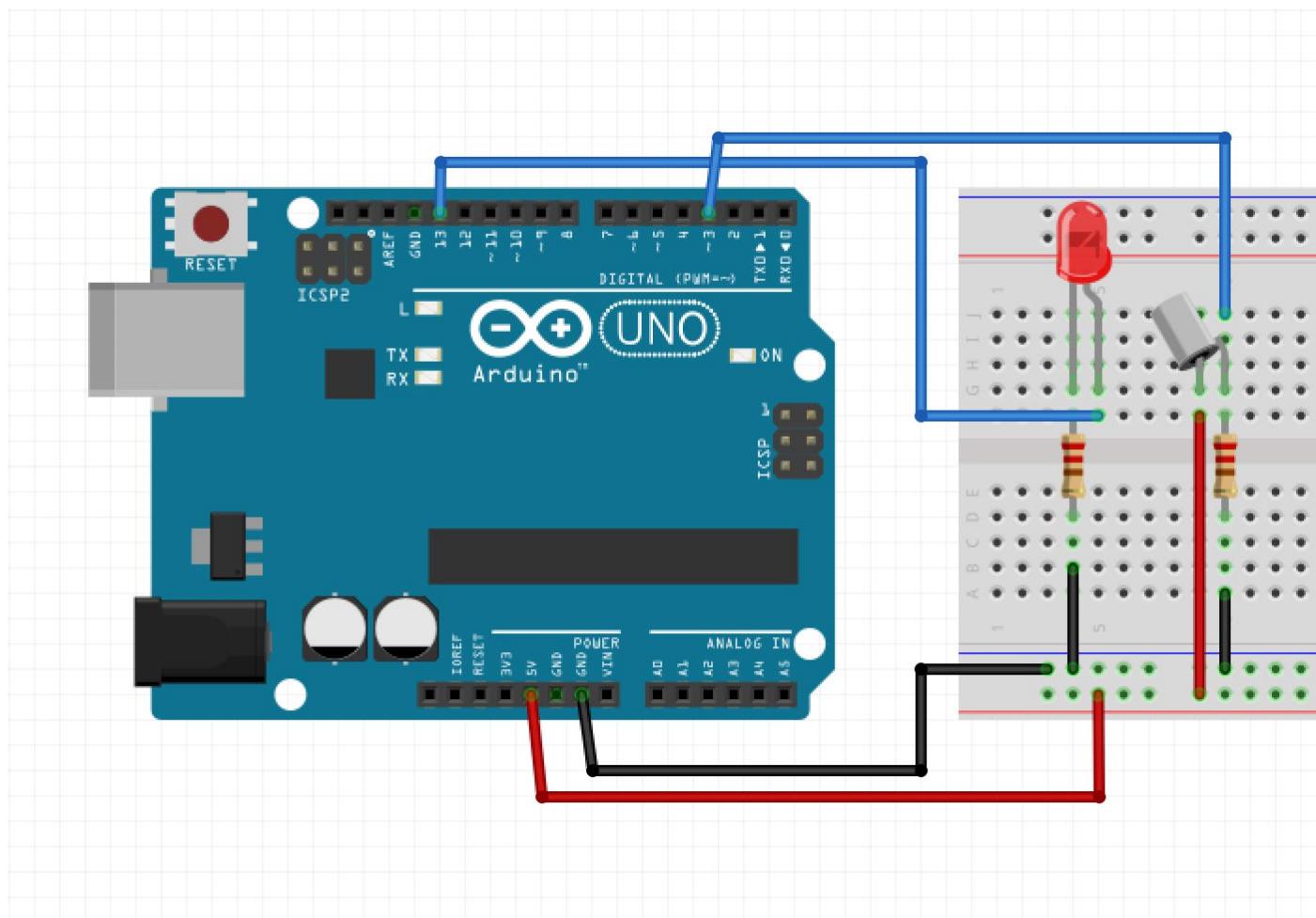
红色LED x1
Red LED



**SW-520D
双珠滚珠开关** x1
SW-520D Tilt Sensor

硬件连接

滚珠开关可以理解为按钮开关，两者原理差不多。只是使用方法稍有不同而已。可以把下图对应《入门7》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。



代码

上传代码后，当我们晃动滚珠开关时，LED就会亮起。停止晃动时，LED就会熄灭。

```
1 //入门20示例：震动传感器
2 /*
3     描述：用滚珠开关控制LED
4     by www.naozhendang.com
5 */
6 int SensorLED = 13;           //定义LED为引脚D13
7 int SensorINPUT = 3;          //连接震动开关到中断1，也就是数字引脚3
8 unsigned char state = 0;
9
10 void setup(){
11     pinMode(SensorLED, OUTPUT);      //LED为输出模式
12     pinMode(SensorINPUT, INPUT);    //震动开关为输入模式
13
14     //低电平变高电平的过程中，触发中断1，调用blink函数
15     attachInterrupt(1, blink, RISING);
16 }
17
18 void loop(){
19     if(state!=0){                  // 如果state不是0时
20         state = 0;                // state值赋为0
21         digitalWrite(SensorLED,HIGH); // 亮灯
22         delay(500);              // 延时500ms
23     } else {
24         digitalWrite(SensorLED,LOW); // 否则，关灯
25     }
26 }
27
28 void blink(){                  //中断函数blink()
29     state++;
30 }
```

语法学习

中断的概念

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。

attachInterrupt(interrupt, function, mode)

The diagram shows the parameters of the attachInterrupt() function. The first parameter 'interrupt' is underlined with a red line. The second parameter 'function' is underlined with a red line. The third parameter 'mode' is underlined with a red line. Below these three parameters, there are three red vertical lines pointing down to the labels: '中断' (Interrupt) under 'interrupt', '中断触发条件' (Interrupt trigger condition) under 'function', and '函数' (Function) under 'mode'. To the right of the 'mode' label, the word '中断号' (Interrupt number) is written vertically.

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。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:中断触发的条件。只有特定的以下四种情况:

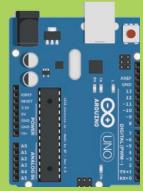
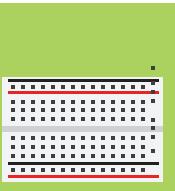
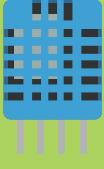
- 1、LOW 当引脚为低电平时,触发中断。
- 2、CHANGE 当引脚电平发生改变时,触发中断。
- 3、RISING 当引脚由低电平变为高电平时,触发中断。
- 4、FALLING 当引脚由高电平变为低电平时,触发中断。

21 温度报警器

温度报警器

我们继续为大家介绍常见的元件，本篇要介绍的是 DHT11温湿度和蜂鸣器两种电子元件。DHT11温湿度传感器，看名字就知道这个传感器可以检测环境中的温度和湿度两个值。而蜂鸣器是一种可以发声的执行器。我们实验就打算用DHT11检测环境温度，当温度超过我们设定的值后，就控制蜂鸣器发出报警声，完成一个简单的温度报警器。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1
	面包板 Breadboard	x1
	DHT11温湿度模块 DHT11 Temperature-Humidity Sensor	x1
	5V有源蜂鸣器 5V External Drive Buzzer	x1

DHT11温湿度传感器

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。常应用于暖通空调、汽车、消费品、湿度调节器、除湿器、医疗、自动控制等领域，具有极高的可靠性与卓越的长期稳定性。

传感器包括一个电阻式感湿元件和一个NTC测温元件，为 4 针单排引脚封装。连接也比较方便。



有源和无源蜂鸣器的区别

蜂鸣器是一种电子发声元器件，可以发出“beep beep”的声音。采用直流电压供电，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中作发声器件。

首先大家要了解有源和无源这里的“源”不是指电源，而是指震荡源。也就是说，有源蜂鸣器内部带震荡源，所以只要一通电就会叫。而无源内部不带震荡源，所以如果用直流信号无法令其鸣叫。必须用2K~5K的方波去驱动它。有源蜂鸣器往往比无源的贵，就是因为里面多个震荡电路。

有源蜂鸣器是一通电就响的，两个脚分别是正负极，两个脚一长一短，短脚为负极，长脚为正极接VCC。本实验用的是有源蜂鸣器，要注意正负极。

而无源蜂鸣器的是需要波形信号驱动的，两个脚没有正负极。



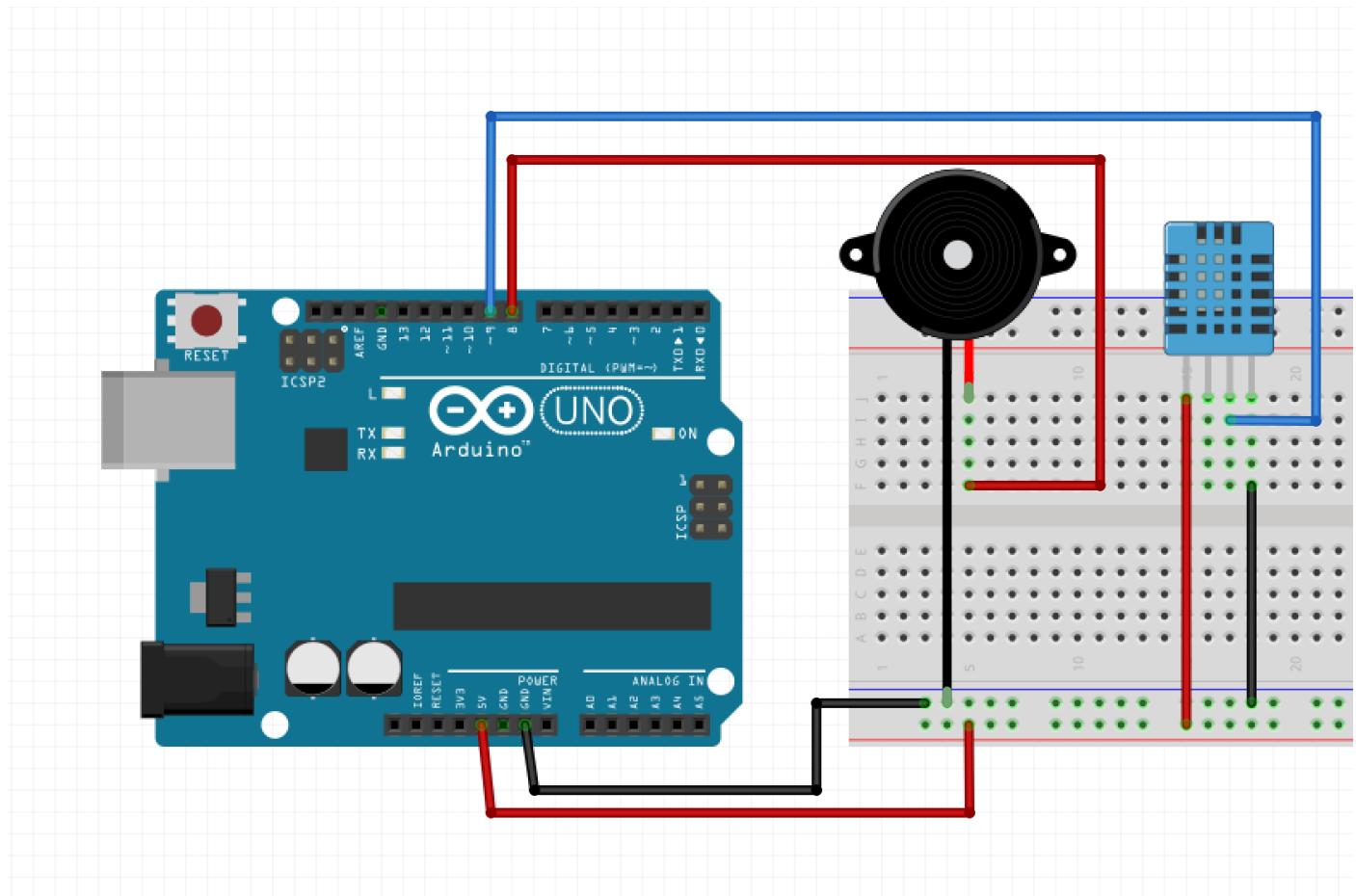
有源蜂鸣器



无源蜂鸣器

硬件连接

本实验使用的是有源蜂鸣器，注意正负极。



代码

1. 安装扩展库

首先，我们需要安装驱动DHT11温湿度的扩展库，它并不是Arduino IDE自带的，所以我们需要自己安装。在示例文件夹里的“DHT-sensor-library”复制到“Documents>Arduino>libraries”里。然后重启Arduino IDE

2. 上传代码

上传完程序后，打开Arduino IDE的串口监视器。将波特率设置为9600。然后就可以看到串口监视器里不断地输出程序了。

用手将DHT11传感器捂热，当温度超过27摄氏度时，蜂鸣器就开始叫了。

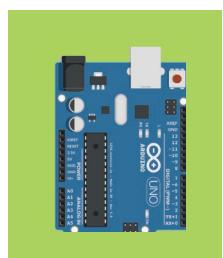
```
1 //入门21示例：温度报警器
2 /*
3     描述：用DHT11温湿度传感器和蜂鸣器实现温度报警器
4     by www.naozhendang.com
5 */
6 #include "DHT.h"
7 DHT dht(9, DHT11); //进行传感器初始设置，9为传感器使用的引脚
8
9 float sinVal;
10 int toneVal;
11
12 void setup() {
13     pinMode(8, OUTPUT);          // 蜂鸣器引脚设置
14     Serial.begin(9600); //设置波特率为9600
15     dht.begin(); //DHT开始工作
16 }
17
18 void loop() {
19     // 两次检测之间，要等几秒钟，这个传感器有点慢。
20     delay(500);
21     // 读温度或湿度要用250毫秒
22     float t = dht.readTemperature(); //读温度，默认为摄氏度
23     Serial.print("Temperature: ");
24     Serial.print(t);
25     Serial.println(" °C");
26
27     if(t>27){           // 如果温度大于27，蜂鸣器响
28         for(int x=0; x<180; x++){
29             //将sin函数角度转化为弧度
30             sinVal = (sin(x*(3.1412/180)));
31             //用sin函数值产生声音的频率
32             toneVal = 2000+(int(sinVal*1000));
33             //给引脚8一个
34             tone(8, toneVal);
35             delay(2);
36         }
37     } else {           // 如果温度小于27，关闭蜂鸣器
38         noTone(8);        //关闭蜂鸣器
39     }
40 }
```

22 用蜂鸣器制造音乐

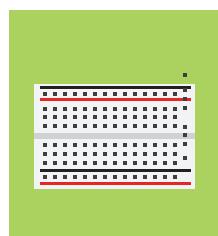
用蜂鸣器制造音乐

在上一篇温度警报器实验中，为大家介绍和实践了蜂鸣器，但我们仅仅是控制它叫起来。你知道么，蜂鸣器还可以有不同的叫声，可以让它制作简单的8bit音乐哦。在本次试验里，让我们来试试用有源蜂鸣器制作音乐吧。

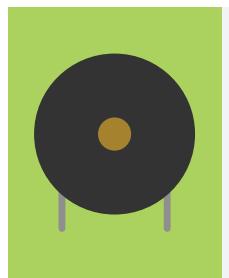
所需元件：



**Arduino UNO
控制板** x1
Arduino Uno R3



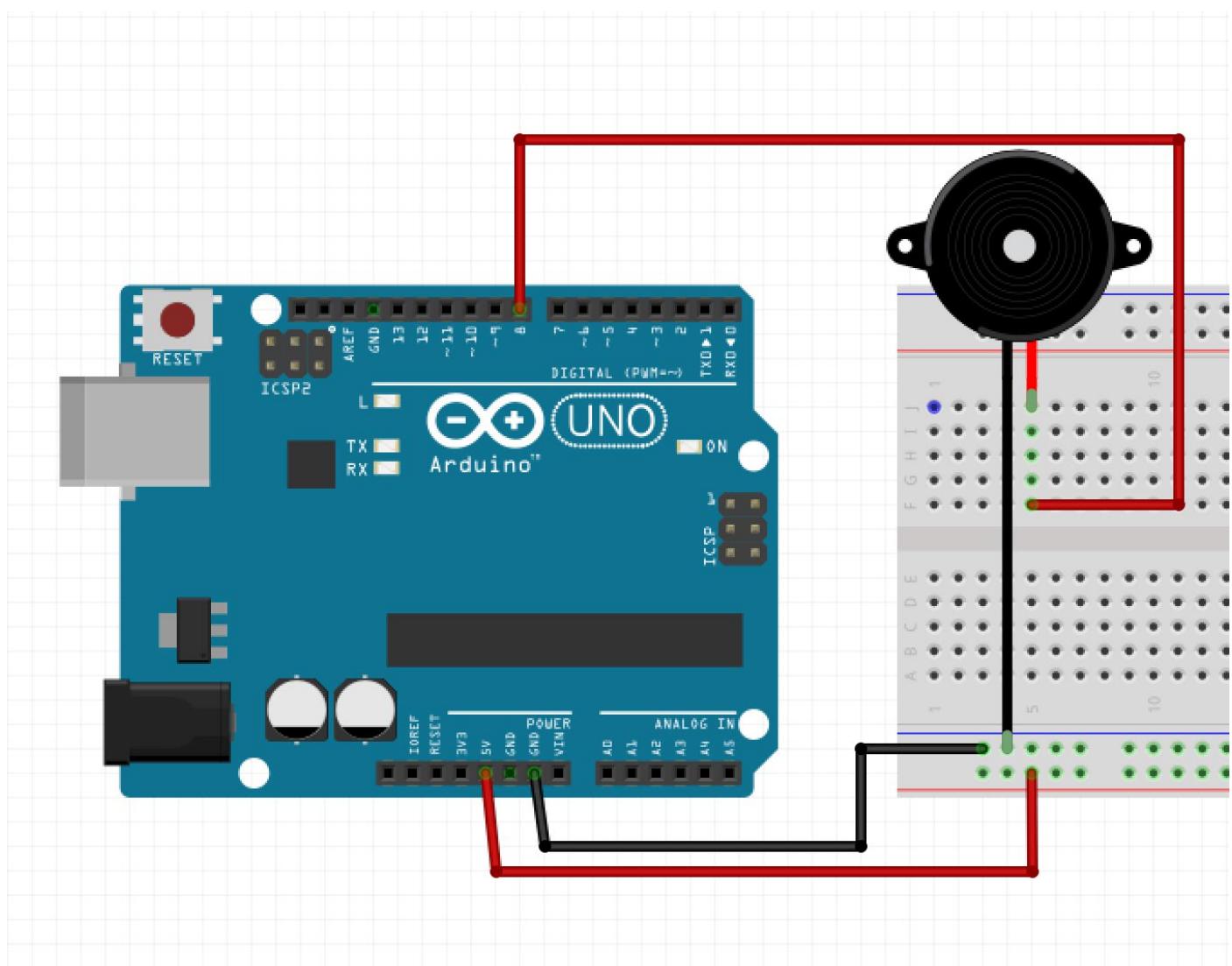
面包板 x1
Breadboard



5V有源蜂鸣器 x1
5V External Drive Buzzer

硬件连接

本实验使用的还是是有源蜂鸣器，注意正负极。使用的数字引脚还是D8。



代码

上传完程序后，按一下Arduino板上的Reset键，音乐就会播放一次。

```
1 //入门22示例：用蜂鸣器制作音乐
2 /*
3     描述：使用蜂鸣器和tone()函数制作简单的8位音乐
4     by www.naozhendang.com
5 */
6 #include "pitches.h" //音符库
7
8 // 音符数组，具体音符的定义在音符库里
9 int melody[] = {
10    NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
11};
12
13 // 节奏长度数组，4= 1 / 4拍， 8= 1 / 8拍
14 int noteDurations[] = {
15    4, 8, 8, 4, 4, 4, 4, 4
16};
17
18 void setup() {
19    // 循环整个音符数组
20    for (int thisNote = 0; thisNote < 8; thisNote++) {
21
22        // 计算每个音符持续的时间
23        // 1000ms/拍数 1 / 4拍 = 1000 / 4, 1 / 拍= 1000/8
24        int noteDuration = 1000 / noteDurations[thisNote];
25        tone(8, melody[thisNote], noteDuration);
26
27        // 音符之间的延时，一般音符持续的时间*1.3即可
28        int pauseBetweenNotes = noteDuration * 1.30;
29        delay(pauseBetweenNotes);
30        // 停止
31        noTone(8);
32    }
33}
34
35 void loop() {
36 }
```

语法学习

tone(pin, frequency)

tone(pin, frequency, duration)



蜂鸣器可以使用Arduino自带的tone()函数发出声音,它主要有两种用法。

如果使用tone(pin, frequency), Arduino会向指定pin发送制定频率的方波,执行noTone()函数来停止。tone(pin, frequency, duration)方法多了一个参数,代表发送方波持续的时间,到时自动停止发送信号,就不需要noTone()函数。

tone()函数对pin是有限制的, pin3-pin13, 其他端口无法使用。

#include "pitches.h"

程序的头部,引入了名为“pitches.h”,它并不是一个Arduino自带的标准库。而是一个自定义的头文件。你注意到Arduino IDE里的工具栏下面有生成了2个标签,第二个标签就是 pitches.h的内容,你可以在里面查看和编辑。它的实际位置是放在和.ino文件同一个文件夹下的。

pitches.h里定义的是不同音符对应的频率数值。

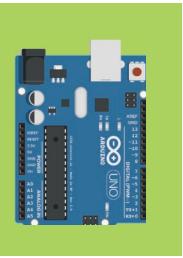
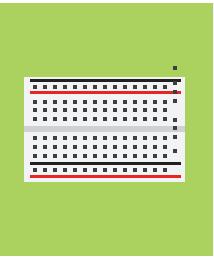
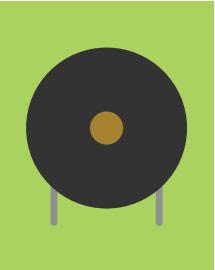
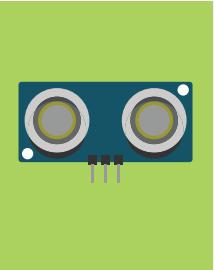
```
L22 pitches.h
1 //*****
2 * Public Constants
3 *****/
4
5 #define NOTE_B0 31
6 #define NOTE_C1 33
7 #define NOTE_CS1 35
8 #define NOTE_D1 37
9 #define NOTE_DS1 39
10 #define NOTE_E1 41
11 #define NOTE_F1 44
12 #define NOTE_FS1 46
13 #define NOTE_G1 49
14 #define NOTE_GS1 52
15 #define NOTE_A1 55
16 #define NOTE_AS1 58
17 #define NOTE_B1 62
18 #define NOTE_C2 65
```

23 超声波特累门琴

超声波特累门琴

在前两篇教程里，我们学会了如何用蜂鸣器发出特定的声音。本章节里，我们来制作一款由超声波驱动的特雷门琴。

所需元件：

	Arduino UNO 控制板 Arduino Uno R3	x1		面包板 Breadboard	x1
	5V有源蜂鸣器 5V External Drive Buzzer	x1		HC-SR04 超声波模块 HC-SR04 Ultrasonic Sensor	x1

什么是特雷门琴？

特雷门琴是世界第一件电子乐器。特雷门琴生产于1919年，由前苏联物理学家利夫·特尔门(Lev Teremin)教授发明，艺名雷奥·特雷门(Leon Theremin)。同年已经由一位女演奏家作出公开演奏，尤甚者连爱因斯坦都曾参观，至今依然是世上唯一不需要身体接触的电子乐器。

Theremin的原理是利用两个感应人体与大地的分布电容的LC振荡器工作单元分别产生震荡的频率与大小变化而工作。圆形天线是用来调节音量的，手越靠近，声音越小。垂直的天线用来调节频率，手越靠近，音调越高。演奏的关键是了解双手的位置与所发出的音符之间的关系。

当然在本实验中的特雷门琴的原理稍许不同，我们使用的是超声波来感应手的距离的。



SR04 超声波模块

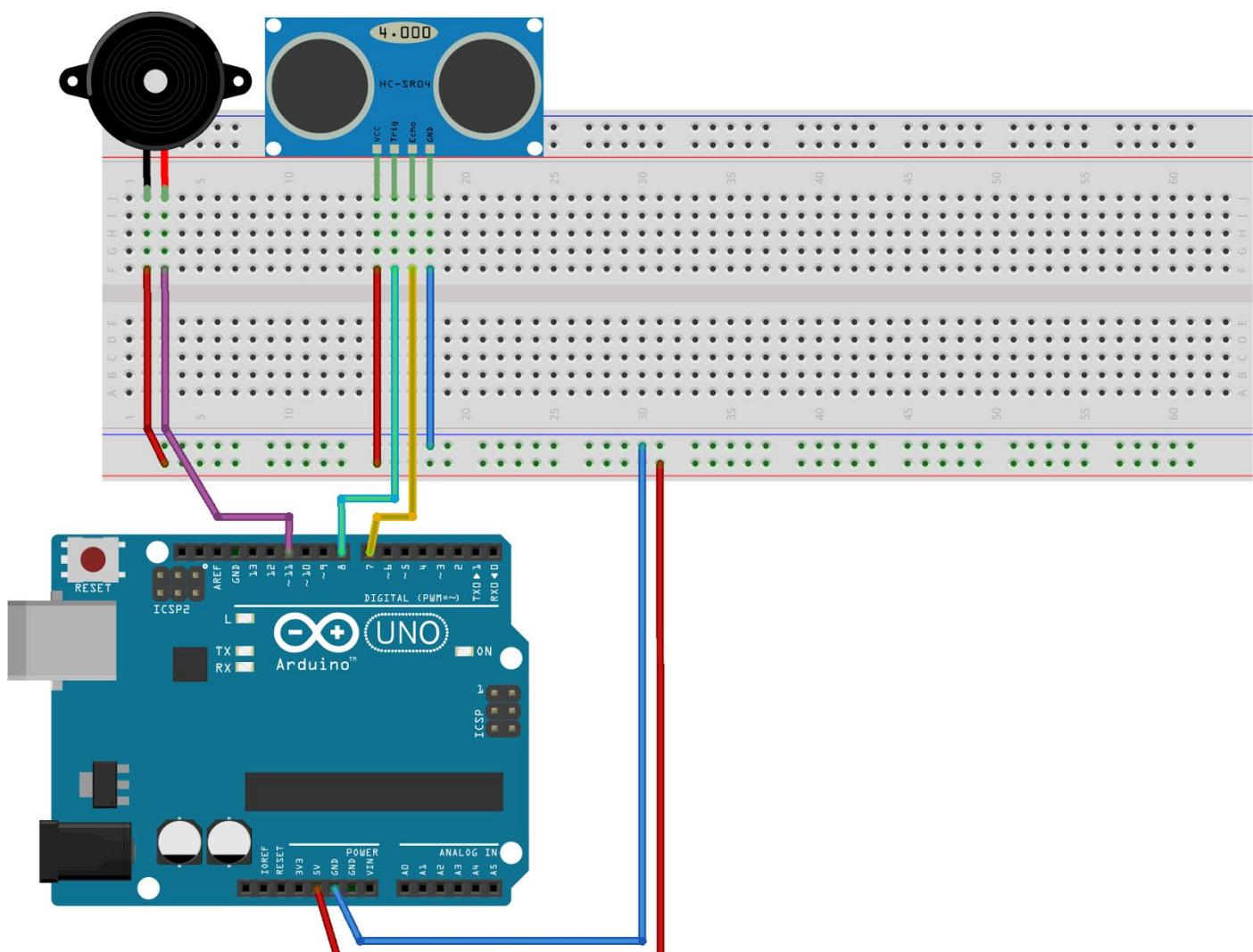
科学家们将每秒钟振动的次数称为声音的频率，它的单位是赫兹(Hz)。我们人类耳朵能听到的声波频率为20Hz~20000Hz。当声波的振动频率小于20Hz或大于 20000Hz时，我们便听不到了。因此，我们把频率高于20000赫兹的声波称为“超声波”。由于超声波指向性强，能量消耗缓慢，在介质中传播的距离较远，因而超声波经常用于距离的测量。

超声波测距原理：超声波发射器向某一方向发射超声波，在发射时刻的同时开始计时，超声波在空气中传播，途中碰到障碍物就立即返回来，超声波接收器收到反射波就立即停止计时。超声波在空气中的传播速度为340m/s，根据计时器记录的时间t，就可以计算出发射点距障碍物的距离(s)，即： $s=340t/2$ 。这就是所谓的时间差测距法。



硬件连接

如图连接好电路。



fritzing

代码

上传完程序后，把手放在超声波传感器的前面，体会一下最大距离和最小距离。当你在超声波设定的范围内移动手时，就会使蜂鸣器发出不同频率的声音，这是隔空演奏的特雷门琴啦。

```
1 //入门23示例：超声波特雷门琴
2 /*
3     描述：使用蜂鸣器、超声波传感器制作特雷门琴
4     by www.naozhendang.com
5 */
6 // 设定SR04连接的Arduino引脚
7 #define echoPin 7
8 #define trigPin 8
9
10 #define LEDPin 13 //使用板载LED
11
12 int buzzer = 11;//蜂鸣器引脚
13 int buzztone = 0;
14
15 //设置蜂鸣器的频率范围
16 int maxbuzztone = 4978;
17 int minbuzztone = 1;
18
19 //设置超声波范围的最大值和最小值
20 int maximumRange = 100;
21 int minimumRange = 0;
22 long duration, distance;
23
24 void setup() {
25     Serial.begin (9600); // 设置波特率
26     // 初始化串口通信及连接SR04的引脚
27     pinMode(trigPin, OUTPUT);
28     pinMode(echoPin, INPUT);
29
30     pinMode(LEDPin, OUTPUT); // 板载LED作为提示灯
31     pinMode(buzzer, OUTPUT); // 初始化蜂鸣器
32 }
33
34 void loop() {
35     // 产生一个10us的高脉冲去触发TrigPin
36     digitalWrite(trigPin, LOW);
37     delayMicroseconds(2);
38
39     digitalWrite(trigPin, HIGH);
40     delayMicroseconds(10);
41
42     digitalWrite(trigPin, LOW);
43     duration = pulseIn(echoPin, HIGH);
44
45     // 检测脉冲宽度，并计算出距离
46     distance = duration/58.2;
47
48     if (distance >= maximumRange || distance <= minimumRange){
49         // 提示超出范围
50         Serial.println("-1");
51         digitalWrite(LEDPin, HIGH);
52
53         noTone(buzzer);
54     } else {
55         //范围内，输出距离已经用蜂鸣器输出对应的频率
56         Serial.println(distance);
57         digitalWrite(LEDPin, LOW);
58
59         buzztone = map(distance, minimumRange, maximumRange, minbuzztone, maxbuzztone);
60         buzztone = maxbuzztone - buzztone;
61         Serial.println(buzztone);
62         tone(buzzer,buzztone);
63     }
64
65     //延时50ms
66     delay(25);
67 }
```

语法学习

delayMicroseconds(2)

使程序暂停指定的一段时间(单位:微秒)。一秒等于1000000微秒。目前,能够产生的最大的延时准确值是16383。这可能会在未来的Arduino版本中改变。对于超过几千微秒的延迟,你应该使用delay()代替。

duration=pulseIn(echoPin,
HIGH);

pulseIn(pin, value)

pulseIn(pin, value, timeout)

pulseIn函数用于读取引脚脉冲的时间长度,脉冲可以是HIGH或LOW。如果是HIGH,函数将先等引脚变为高电平,然后开始计时,一直到变为低电平为止。返回脉冲持续的时间长短,单位为ms。如果超时还没有读到的话,将返回0。

参数:

pin:你要进行脉冲计时的引脚号(int)。

value:要读取的脉冲类型,HIGH或LOW(int)。

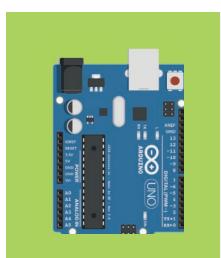
timeout (可选):指定脉冲计数的等待时间,单位为微秒,默认值是1秒(unsigned long)

24 8x8LED矩阵怦然心动

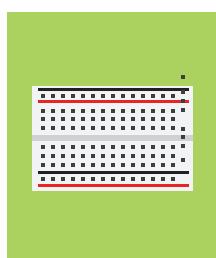
8x8LED矩阵怦然心动

在本篇里，我们将介绍新的显示元件--8x8LED点阵屏。8x8LED点阵屏共有64颗LED，是非常实用的像素屏，今天我们就来用它显示一颗跳动的心，可谓是表白利器哦。

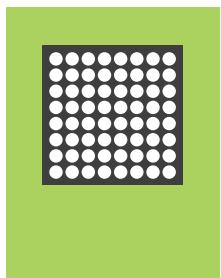
所需元件：



Arduino UNO
控制板
x1
Arduino Uno R3



面包板
Breadboard
x1



8x8 LED点阵模块
x1
8x8 LED Matrix

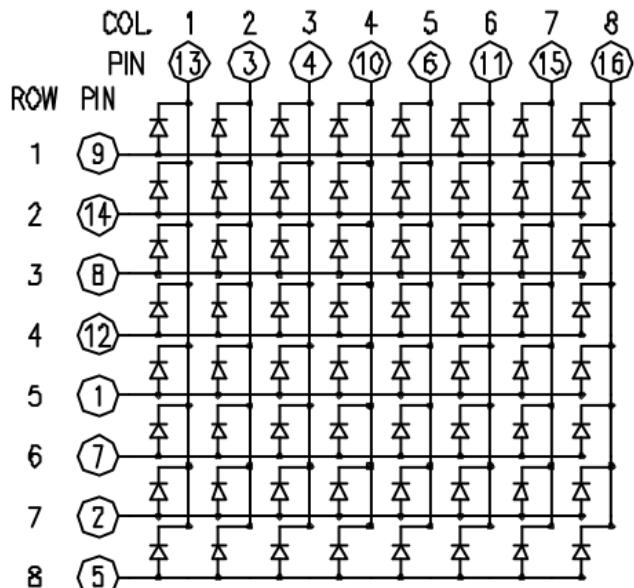
8x8LED点阵

8x8LED和之前我们接触过的RGB LED一样,有共阴和共阳的区别,但这里共阴共阳的概念有点不一样。

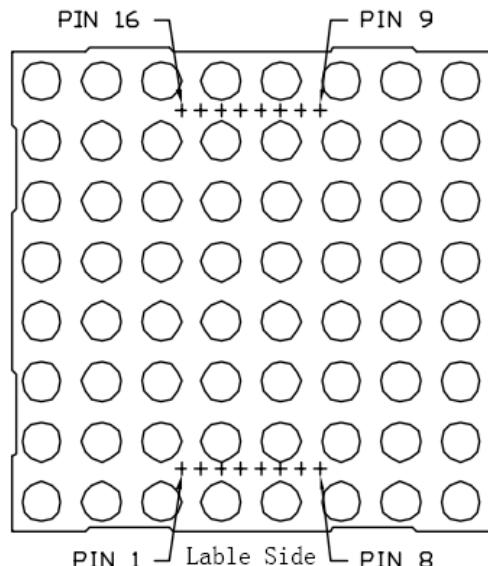
1. 共阳就是LED正极全部接在一起,而负极则不是,共阴相反但也是类似。这种说法是错误的哦!(RGB LED的共阳和共阴是这样的),但点阵8*8点阵共阴和共阳都是接在一起的。

2. LED点阵所谓共阳和共阴只是引脚脚排列名称相同(也就是说无论共阳还是共阴管脚编号一致),但行和列的极性则相反。既然这样,那么反过来接共阳不就是变成了共阴吗?理论上是的,但要注意行和列相交的位置发生了变化。

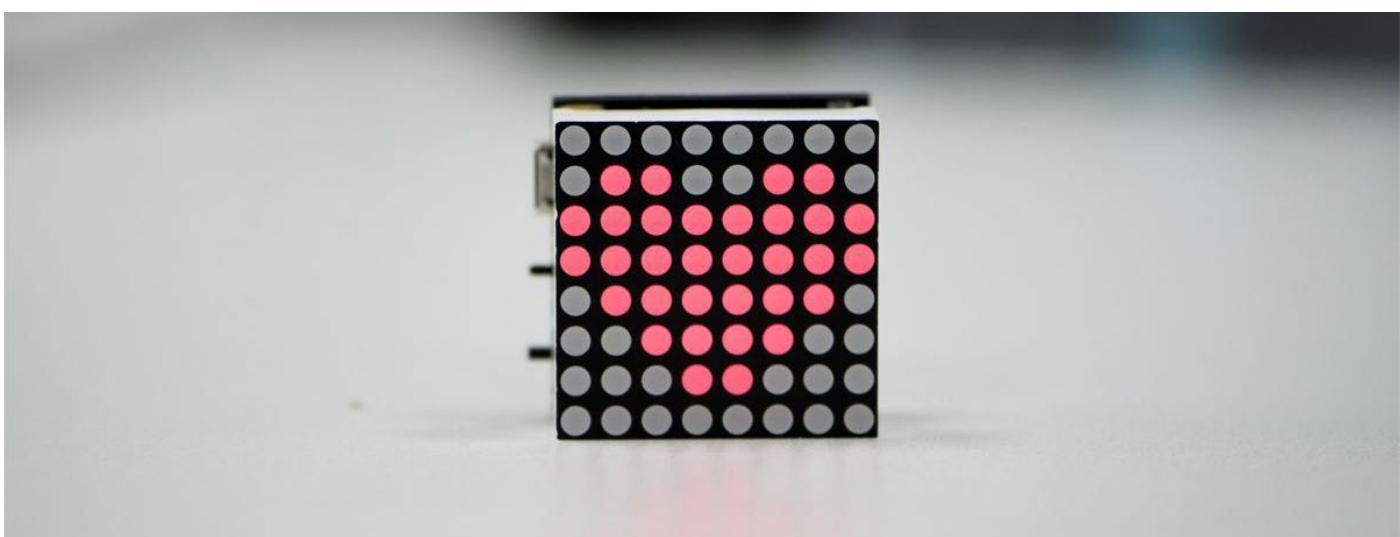
3. 8*8点阵的引脚排列不是顺序的!W



共阳LED点阵

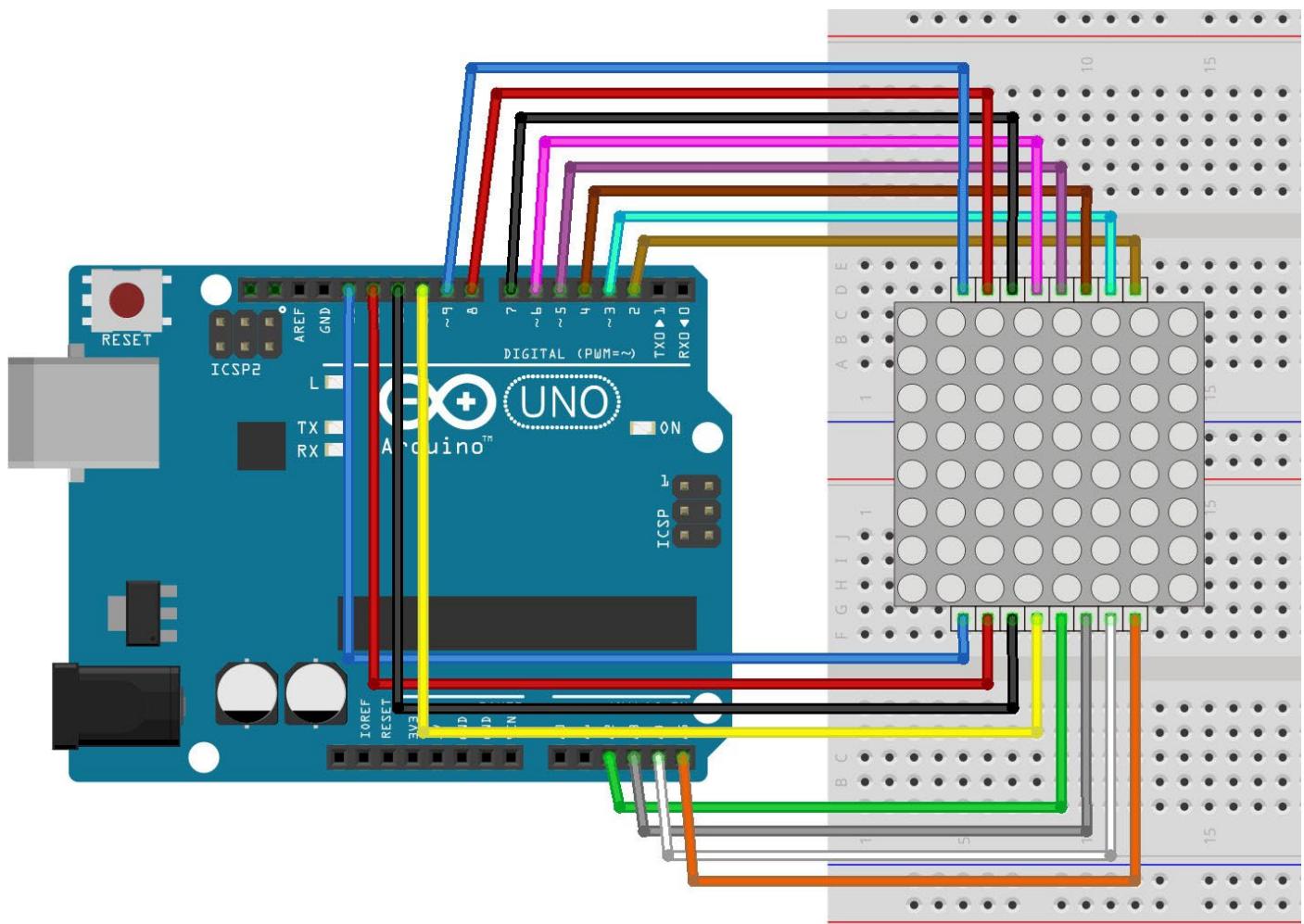


PIN引脚



硬件连接

这个实验中，我们没有用额外的驱动芯片(比如74HC595)来驱动LED点阵，而是直接用Arduino来驱动，几乎会占用所有的I/O，所以连线较多，注意再三检查。



代码

上传完程序后，你就可以看见LED点阵上出现了跳动的心。

```
1 //入门24示例：LED矩阵怦然心动
2 /*
3     描述：使用8x8LED矩阵制作表白神器
4     by www.naozhendang.com
5 */
6 int R[] = {2,7,A5,5,13,A4,12,A2}; //行
7 int C[] = {6,11,10,3,A3,4,8,9}; //列
8
9 unsigned char biglove[8][8] = {      //大“心型”的数据
10    0,0,0,0,0,0,0,
11    0,1,1,0,0,1,1,0,
12    1,1,1,1,1,1,1,1,
13    1,1,1,1,1,1,1,1,
14    1,1,1,1,1,1,1,1,
15    0,1,1,1,1,1,1,0,
16    0,0,1,1,1,1,0,0,
17    0,0,0,1,1,0,0,0,
18 };
19
20 unsigned char smalllove[8][8] = {      //小“心型”的数据
21    0,0,0,0,0,0,0,
22    0,0,0,0,0,0,0,
23    0,0,1,0,0,1,0,0,
24    0,1,1,1,1,1,1,0,
25    0,1,1,1,1,1,1,0,
26    0,0,1,1,1,1,0,0,
27    0,0,0,1,1,0,0,0,
28    0,0,0,0,0,0,0,0,
29 };
30
31 void setup(){
32     //循环定义行列PIN 为输出模式
33     for(int i = 0;i<8;i++){
34         pinMode(R[i],OUTPUT);
35         pinMode(C[i],OUTPUT);
36     }
37 }
38
39 void loop(){
40     for(int i = 0 ; i < 100 ; i++){      //循环显示100次
41         Display(biglove);           //显示大“心形”
42     }
43     for(int i = 0 ; i < 50 ; i++){      //循环显示50次
44         Display(smalllove);          //显示小“心形”
45     }
46 }
47
48 void Display(unsigned char dat[8][8]) { //显示函数
49     for(int c = 0; c<8;c++){
50         digitalWrite(C[c],LOW); //选通第c列
51
52         //循环
53         for(int r = 0;r<8;r++){
54             digitalWrite(R[r],dat[r][c]);
55         }
56         delay(1);
57         Clear(); //清空显示去除余晖
58     }
59 }
60
61 void Clear(){                         //清空显示
62     for(int i = 0;i<8;i++){
63         digitalWrite(R[i],LOW);
64         digitalWrite(C[i],HIGH);
65     }
66 }
```

语法学习

二维数组

二维数组是多维数组的最简单形式。在本质上，是一个一维数组的列表。声明一个 x 行 y 列的二维整型数组：

```
unsigned char biglove[8][8] = { type arrayName [ x ][ y ];  
0,0,0,0,0,0,0,  
0,1,1,0,0,1,1,0,  
1,1,1,1,1,1,1,1,  
1,1,1,1,1,1,1,1,  
1,1,1,1,1,1,1,1,  
0,1,1,1,1,1,1,0,  
0,0,1,1,1,1,0,0,  
0,0,0,1,1,0,0,0,  
};
```

type可以是任意有效的Arduino数据类型，arrayName是二维数组的名称。一个二维数组可以被认为是一个带有 x 行和 y 列的表格。所以数组中的每个元素是使用形式为 $a[i, j]$ 的元素名称来标识的，其中 a 是数组名称， i 和 j 是唯一标识 a 中每个元素的下标。

左图的二维数组等同于下面的声明，因为内部嵌套的括号是可选的。

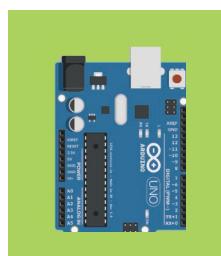
```
unsigned char biglove[8][8] = {  
{0,0,0,0,0,0,0,0},  
{0,1,1,0,0,1,1,0},  
{1,1,1,1,1,1,1,1},  
{1,1,1,1,1,1,1,1},  
{1,1,1,1,1,1,1,1},  
{0,1,1,1,1,1,1,0},  
{0,0,1,1,1,1,0,0},  
{0,0,0,1,1,0,0,0}  
};
```

25 数码管计时器

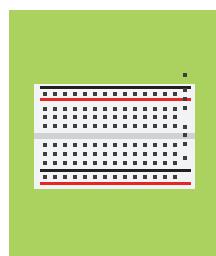
数码管计时器

到目前为止，相信您已经对Arduino怎样运作，如果编写程序读取传感器、控制驱动器有一定的知识了。在这个教程里，继续为大家介绍一种显示元件-- 数码管。我们将用数码管和按钮做一个计时器。

所需元件：



**Arduino UNO
控制板** ×1
Arduino Uno R3



面包板 ×1
Breadboard



4位共阴数码管 ×1
4-Digit 7-Segment Display



按钮开关 ×1
Push Buttons



1K欧姆电阻 ×4
1KΩ Resistors

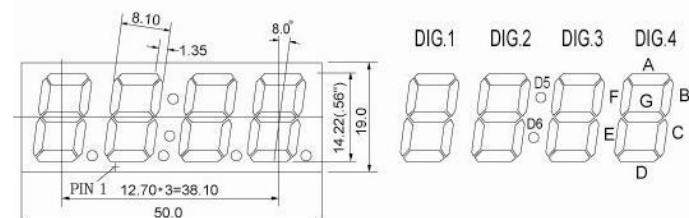


220欧姆电阻 ×1
220Ω Resistors

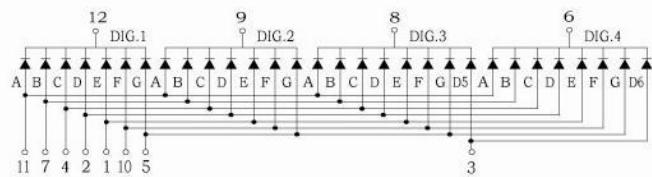
LED数码管

LED数码管 (LED Segment Displays) 由多个发光二极管封装在一起组成“8”字型的器件，引线已在内部连接完成，只需引出它们的各个笔划，公共电极。数码管实际上是由七个发光管组成8字形构成的，加上小数点就是8个。这些段分别由字母a,b,c,d,e,f,g,dp来表示。(具体对应的引脚，以数码管的datasheet为准)。

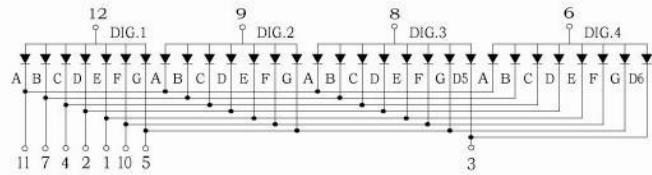
数码管跟RGB LED类似，有共阴共阳之分。套件提供的是共阴的数码管(4位带时钟，不显示小数点)。



SR*40563

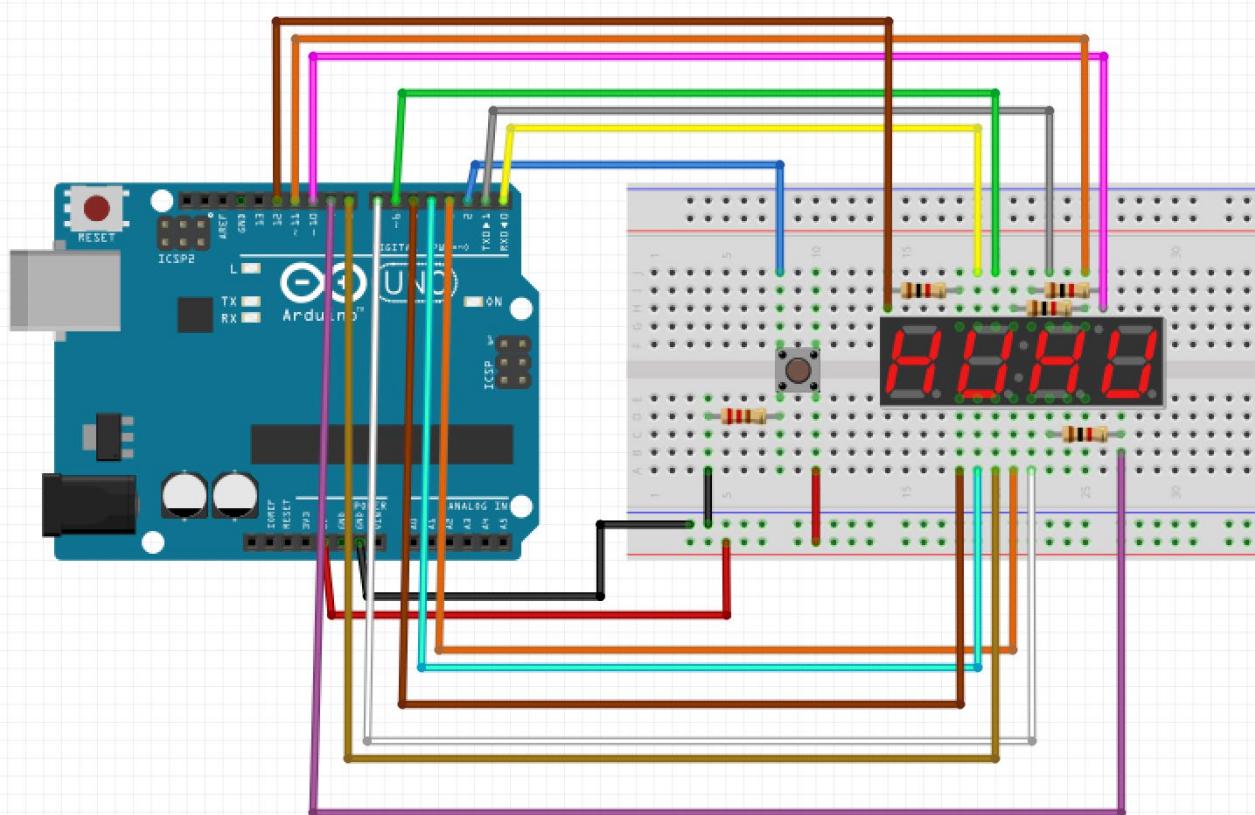


SR*30563



硬件连接

按照图示连接好硬件。按钮的接法和《入门8-按钮去抖Debounce》中的一样。4位数码管引脚11,7,4,2,1,10,5,3分别接到Arduino的引脚0,1,3,4,5,6,7,8。4位数码管引脚12,9,8,6分别接到Arduino的引脚12,11,10,9上。限流电阻有两种接法:一种是在d1到d4总共接4颗。这种接法好处是需求电阻比较少 但是会产生每一位上显示不同数字亮度会不一样1最亮8最暗。另外一种接法就是在其他8个引脚上接这种接法亮度显示均匀但是用电阻较多。下面的连法采用的是第一种。



fritzing

代码

上传完程序，按下按钮，就是有计时器效果，再按一下计时器停止。

```
1 //入门25示例：数码管计时器
2 /*
3     描述：用4位数码管做一个计时器，按钮按下归零开始计时，再按一下停止。
4     by www.naozhendang.com
5 */
6 #include <TimerOne.h>
7
8 long t = 0;
9 int count = 0;
10 int countState = LOW; //计时状态
11
12 //注意：套件提供的4位数码管为共阴带时钟，但4位数字右下角的小数点是不显示。
13 //阳极接口：8字段对应的a,b,c,d,e,f,g,dot
14 //数码管对应针脚分别为11,7,4,2,1,10,5,3
15 byte segs[8] = {0,1,3,4,5,6,7,8};
16 //阴极接口
17 //数码管对应针脚分别为12,9,8,6
18 byte pin[4]={12,11,10,9};
19 |
20 //显示8位字符形状的数据组
21 byte seven_seg_digits[10][8] = {
22     //a,b,c,d,e,f,g,dot
23     { 1,1,1,1,1,1,0,1}, // = 0
24     { 0,1,1,0,0,0,0,1}, // = 1
25     { 1,1,0,1,1,0,1,1}, // = 2
26     { 1,1,1,0,0,1,1,1}, // = 3
27     { 0,1,1,0,0,1,1,1}, // = 4
28     { 1,0,1,1,0,1,1,1}, // = 5
29     { 1,0,1,1,1,1,1,1}, // = 6
30     { 1,1,1,0,0,0,0,1}, // = 7
31     { 1,1,1,1,1,1,1,1}, // = 8
32     { 1,1,1,1,0,1,1,1} // = 9
33 };
34
35 const int buttonPin = 2; //按钮连接的引脚为D2
36 int buttonState; // 声明一个整型变量来储存按钮的状态
37 int lastButtonState = LOW; // 声明一个整型变量来储存LED上一次的状态
38
39 // 用户储蓄时间的变量，所以用long变量。
40 long lastDebounceTime = 0; // 声明一个long型变量来储存上一次去抖的时间
41 long debounceDelay = 50; // 声明一个long型变量来储存判断去抖的时间差
42
43 void setup() {
44     pinMode(buttonPin, INPUT);
45
46     for(int i=0;i<sizeof(segs);i++){
47         pinMode(segs[i], OUTPUT);
48     }
49
50     for(int i=0;i<sizeof(pin);i++){
51         pinMode(pin[i], OUTPUT);
52     }
53
54     clearLEDs(); //清屏
55
56     //全部归零
57     for(int j=0;j<sizeof(segs);j++){
58         digitalWrite(segs[j],seven_seg_digits[0][j]);
59     }
60
61     //设置计时器
62     Timer1.initialize(100000);
63     Timer1.attachInterrupt( add );
64     Timer1.stop();
65 }
66
```

```
67 void loop() {
68     drawTimer();
69
70     //获取按钮引脚的状态，并存储到名为reading到整型变量里
71     int reading = digitalRead(buttonPin);
72
73     //下面我们要检测按钮是否被按下
74     //如果被按下，我们就开始计时
75     //直到超过指定的时间，按钮状态稳定了再读取，这样可以过滤抖动误差
76
77     //如果按钮跟上一次状态不一样，即按钮状态改变了
78     if (reading != lastButtonState) {
79         lastDebounceTime = millis(); //记录当前的时间戳，重新计时
80     }
81
82     //如果当前时间和开始时的时间差已经超过了设定的判断去抖的时间差,
83     if ((millis() - lastDebounceTime) > debounceDelay) {
84
85         //如果按钮当前状态改变
86         if (reading != buttonState) {
87             buttonState = reading; //储存状态
88
89             if (buttonState == HIGH) { //如果按钮按下
90                 if(countState){
91                     //停止计时
92                     Timer1.stop();
93                     countState = LOW;
94                 }else{
95                     //重新开始计时
96                     t = 0;
97                     Timer1.resume();
98                     //Timer1.restart();
99                     countState |= HIGH;
100                }
101            }
102        }
103    }
104
105    //为下次循环做准备
106    //本次的按钮状态就是下次循环中上一次按钮的状态。
107    lastButtonState = reading;
108}
109
110 void drawTimer(){ //绘制时间
111     clearLEDs(); //清屏
112     setDigit(0); //选择第1位
113     setNum((t/60)/10); // 设置第1位数字
114     delayMicroseconds(50); //delay 5ms
115
116     clearLEDs(); //清屏
117     setDigit(1); //选择第2位
118     setNum((t/60)%10); // 设置第2位数字
119     delayMicroseconds(50); //delay 5ms
120
121     clearLEDs(); //清屏
122     setDigit(2); //选择第3位
123     setNum((t%60)/10); // 设置第3位数字
124     delayMicroseconds(50); //delay 5ms
125
126     clearLEDs(); //清屏
127     setDigit(3); //选择第4位
128     setNum(t%10); // 设置第4位数字
129     delayMicroseconds(50); //delay 5ms
130 }
131
132 void setDigit(int x) { //设置4位中的1位
133     for(int i=0;i<sizeof(pin);i++){
134         digitalWrite(pin[i], HIGH);
135     }
136
137     switch(x)
138     {
139         case 0:
140             digitalWrite(pin[0], LOW); //Light d1 up
141             break;
142         case 1:
143             digitalWrite(pin[1], LOW); //Light d2 up
144             break;
145         case 2:
146             digitalWrite(pin[2], LOW); //Light d3 up
147             break:
148         default:
149             digitalWrite(pin[3], LOW); //Light d4 up
150             break;
151     }
152 }
153
154 void setNum(int x){ //设置数字
155     for(int j=0;j<sizeof(segs);j++){
156         digitalWrite(segs[j], seven_seg_digits[x][j]);
157     }
158 }
159
160 void clearLEDs(){ //重置清屏
161     for(int i=0;i<sizeof(segs);i++){
162         digitalWrite(segs[i], LOW);
163     }
164 }
165
166 void add(){
167     count++;
168     if(count == 10){
169         count = 0;
170         t++;
171         if(t == 3600){
172             t = 0;
173         }
174     }
175 }
```

语法学习

```
Timer1.initialize(100000);  
Timer1.attachInterrupt( add );
```

你可能很好奇millis()、delay()等函数是怎么知道具体的时间差的。其实Arduino UNO(确切地说是 atmega328芯片)里一共有三个定时器，除了定时中断，它们还可以控制引脚pwm输出。

```
Timer1.stop();  
Timer1.resume();
```

其中timer0已经被用来实现delay()、delayMicroseconds、millis()。有些情况下，我们要用到定时器timer0相关的函数，但同时又要定时接收外来信号。我们就要启用timer1和timer2。

要使用timer1我们需要在头部引入TimerOne的头文件：

```
#include <TimerOne.h>
```

Timer1.initialize(100000); 用来初始化Timer1，括号里的时间单位为微秒，1秒=1000 000微秒。

```
Timer1.attachInterrupt( add );
```

这个就是设置时间到了就要干什么事情，括号里面就是函数名。例如，现在设置每隔0.1s然后就去做add这个函数！具体要做什么，就在add函数里定义。

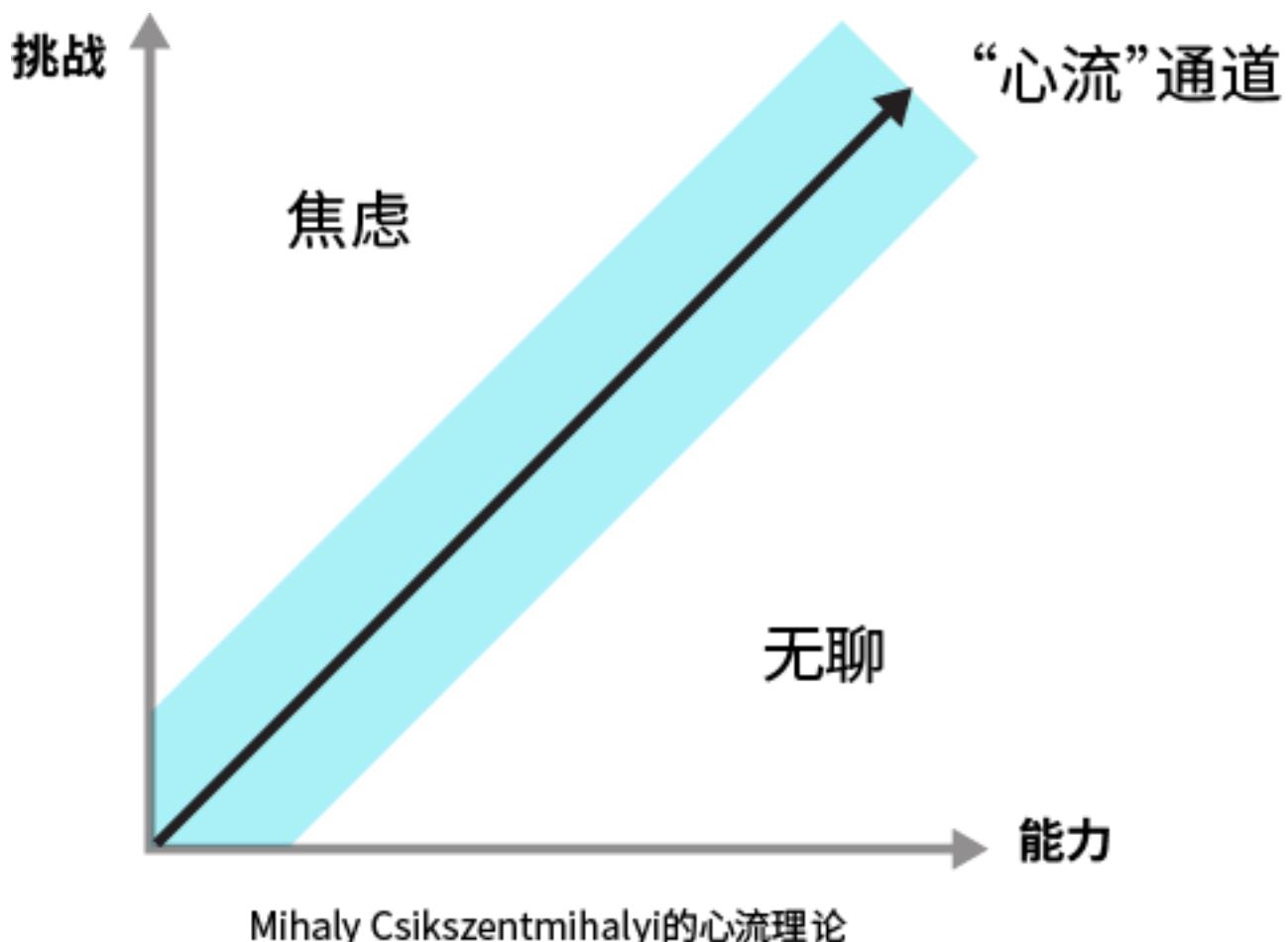
Timer1.stop(); 很好理解，暂停Timer1计时。

Timer1.resume(); 重启Timer1计时。

26 学习方法建议和 Arduino的未来

Arduino的“方式”

有些人学Arduino觉得很难上手，或者有些人玩了一阵子觉得Arduino过于简单，这些情况都有。很多时候是学习的方法不得法，也有可能是思维方式的模式。心理学家米哈里·契克森米哈(Mihaly Csikszentmihalyi)有著名的心流理论。所谓的心流(flow)，可以简单说成一种“如有神助的状态”，当我们的感觉和行动都处于最舒适最有效率的一种状态。米哈里·契克森米哈这种状态主要由人要完成的任务的难度和人的自身能力之间关系决定。举例说，当你觉得入门Arduino感觉比较吃力的时候，很有可能你要完成实验或者教程难度过大，你现有的相关知识储备不够或者解决问题的能力不够；反之，你也许是专业的程序员或者电子工程师，会觉得学这些基础知识太无聊了，这说明你遇到的挑战不够，这也为很多鄙视Arduino的原因。Arduino预设的用户是那些没有太多电子科学基础的设计师和艺术家。所以脑叔给初学Arduino者第一个建议就是选择和自己能力匹配难度的教程，当然如果你已经是专家级别的，甚至你可以应该质疑是否学习Arduino这个平台，工具而已，适合自己的才是最好的。



玩

“玩”Arduino，这是很重要的态度。很多人一上来就是学院派的作风，先把谭老师的C基础过几遍，再研究模电基础。这种教科书般的方法会让玩弄Arduino的乐趣荡然无存。不要过于担心犯错，烧坏几个元件是常事，写错代码也不会引起爆炸。最重要的是会“作乐”，不夸张地说这是学习过程中支撑你学下去，给你成就感的源泉。

边做边学，在错误中学习

理论固然重要，但在Arduino学习过程中更重要的一点就是动手。只有自己尝试着跟着教程一步步做，你才知道各种坑，而且只有在一个个坑中栽倒过，你才会吃一堑长一智。这需要大量的时间和精力投入，没有什么捷径。

Tinkering

Tinker这一词，脑叔也不知道对应中文哪个词会比较恰当。Tinkering在英文中是修补，有点瞎捣鼓的意思。这也是Arduino区别与传统硬件开发思想的一个精髓。传统的硬件开发讲究的是效率，非常线性，有明确的产品开发目标，然后就是从A到B的开发。对比下Arduino玩家一般在初期并没有什么成型的想法，一遍做，一遍探索是非常常见的过程，这过程注重的是想法的多样性而不是开发效率。



废物利用

如今人们随意丢弃很多身边的电子垃圾：旧打印机，旧收音机，手机，电脑硬盘等等。而对于那些电子爱好者和穷苦的hacker来说这些东西可都是财富。你可以重复利用很多旧电器上面的电子元件和模块。把可以重复利用的东西拆下来归类，让它们在以后的项目中再次发光发热。这不但可以节省一大笔开支，而且对于环境保护也尽了绵薄之力。

社区协作

最后一点，要利用好社区的力量。这里的社区有线上的社区，比如Arduino.cc的官方社区，里面沉积了大量的知识、问题还有经验，但线上社区的缺陷是知识点零散，形式单一，比如大都是以论坛和wiki的形式呈现。这也是为什么脑震荡希望通过更高质量，更科学的梳理来帮助大家。还有一种线下的社区，比如各地的makerspace，有共享的空间和设备，更重要的是有共同爱好的人，定期组织聚会，一起学习，互相帮助比一个人更有效率更有趣。



Arduino的未来

Arduino作为当下最热的Maker原型平台，它未来将如何发展？常言道，物极必反，一方面Arduino内部利益的分裂，另一方面面对众多的模仿者和竞争者，Arduino是否能在这样前有狼，后有虎的形势下继续高歌猛进？作为是否值得投入时间和精力学习？那么最后，让我们来聊聊Arduino平台的未来，当然这些都是脑叔自己的解读，若您有独到见解，也欢迎深入讨论。

随着Arduino越来越多的普及，Arduino核心团队也马不停蹄尝试更好地商业化Arduino，无论是开源软件，还是开源硬件，这条路似乎是条必经之路，也是条鲜又人走好的蜀道。先例有Linux届的Ubuntu，源自Reprap的Makerbot等。Arduino核心团队无论从品牌和供应链都比从前更加有控制欲了。比如重新设计了Arduino的品牌形象还有产品包装，强调了官方出品。

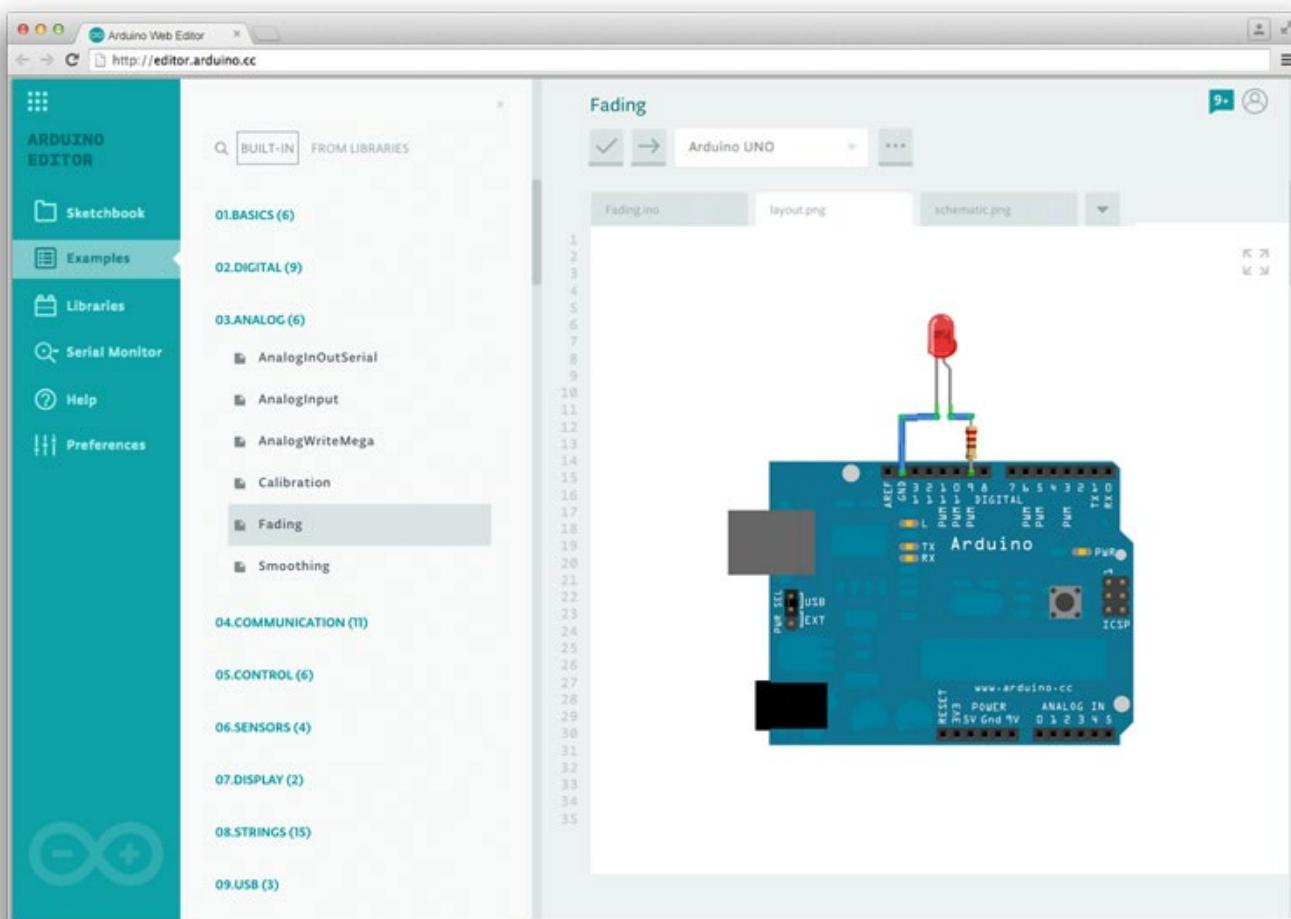


在供应链上，原来的Arduino是“Made in Italy”，是由意大利本地的一家制造商承包的。但核心团队为了Arduino更好地国际化，在美国和Adafruit合作，继续沿用Arduino TM这个品牌，在非美国地区，则新创了“Genuino”的品牌，比如，在中国就和深圳的SeeedStudio合作，由其生产分销官方的Genuino开发板。这也是前段时间“Arduino vs. Arduino”事件的根本原因 — 国际化带来的利益矛盾。对于我们来说当然是件好事，我们可以在Arduino开发板鱼龙混杂的今天买到质量靠谱的官方版本了。



再谈到Arduino硬件和Arduino IDE的开发，在物联网和云概念普及到街头大妈的今天，我们也可以看到Arduino核心团队在这方面的布局。譬如集成了BLE蓝牙4.0的Arduino 101，集成了WIFI模块的Arduino YÚN，拥有更强计算能力的Arduino Due。但比起Particle，WIFI神器ESP8266，Pi等竞争对手，价格和定位都过于尴尬。在IDE的开发上，之前我们介绍过，Arduino官方也在尝试在线的云IDE，所有的代码编写，编译和烧录都在云端处理，本地不用任何软件。这个概念也不新，同类的竞争者如CodeBender已经初具规模。脑叔认为云IDE并不能完全取代本地IDE，相对免安装，云端编译等便捷的方面，对网络的依赖度和效率上，本地IDE有着无可比拟的优势，Arduino核心团队还应该把重心放回本地IDE上。

Arduino之于开源硬件，就如Apple之于手机界，霸业已成，需要超越的不是别人而只是自己。



Arduino在线编辑器