

让我们开始吧！

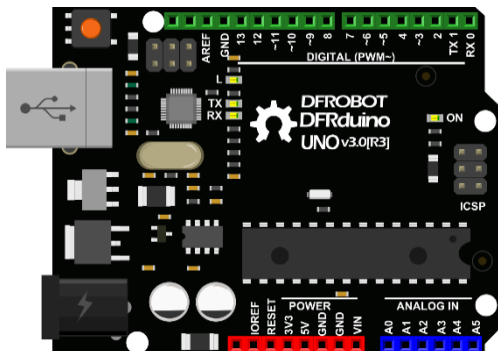
从 LED 开启我们的 Arduino 之旅吧！你将学会像控制按钮输入一样控制 Arduino 的各种输出。在硬件方面，你将学习到有关 LED、按钮、和电阻的内容，包括上拉和下拉电阻的知识，这对于之后的项目非常重要。在这个过程中，你将接触 Arduino 编程，编程其实也没你想象的那么困难。让我们从一个最基本的项目，使用 Arduino 控制一个外部 LED 的闪烁。

项目一 LED 闪烁

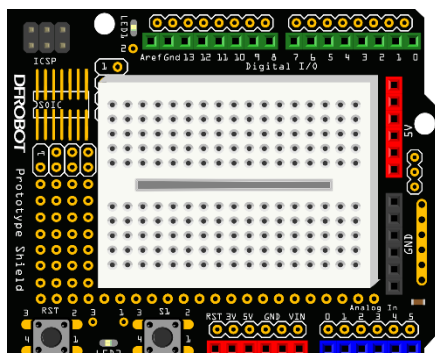
在第一个项目中，我们将重复使用之前的那个测试代码 Blink 程序。有所不同的是，这里我们需要外接一个 LED 到数字引脚，而不是使用焊在开发板上的 LED 13(也就是“L”灯)。便于我们能清晰的认识 LED 的工作原理及一些硬件电路的搭建。




所需元件

- 1× DFduino UNO R3（以及配套 USB 数据线）



- 1× Prototype Shield 原型扩展板+面包板



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

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

硬件连接

首先，从我们的套件中取出 Prototype shield 扩展板和面包板，将面包板背面的双面胶歇下，粘贴到 Prototype shield 扩展板上。再取出 UNO，把贴有面包板 Prototype shield 扩展板插到 UNO 上。取出所有元件，按照图 1 连接。

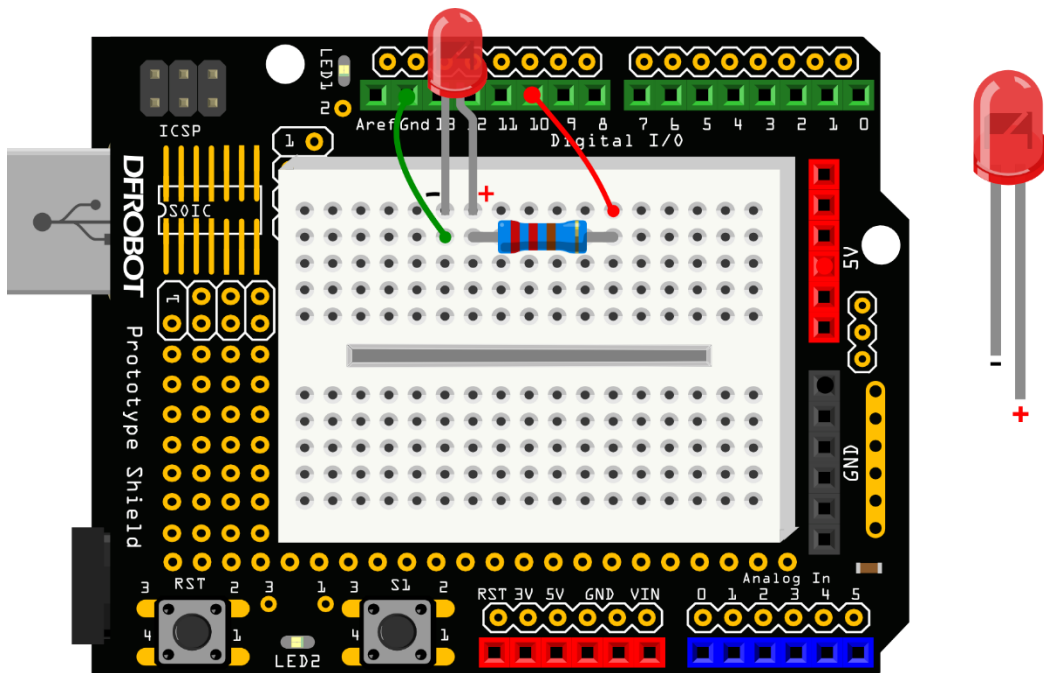


图 1-1 LED 闪烁连线图

用绿色与红色的面包线连接（DF 中定义，绿色为数字口，蓝色为模拟口，红色为电源 VCC，黑色为 GND，白色可随意搭配），使用面包板上其他孔也没关系，只要元件和线的连接顺序与上图保持一致即可。

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

输入代码

打开 Arduino IDE，在编辑框中输入样例代码 1-1 所示代码。（输入代码也是一种学习编程的过程，虽然提供代码的压缩包，但还是建议初学者自己输入代码，亲身体验一下。）

样例代码 1-1:

```
//项目一 —— LED 闪烁
/*
  描述：LED 每隔一秒交替亮灭一次
*/
int ledPin = 10;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

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

现在让我们来回顾一下代码和硬件，看看它们是如何工作的。

代码回顾

代码的第一行如下所示：

```
//项目一 —— LED 闪烁
```

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

还有另外一种写注释的方式，用“/*...*/”，这个符号的作用是可以注释多行，这也是与上一种注释方式的区分之处。在/*和*/中间的所有内容都将被编译器忽略，不进行编译。

例如以下文字：

```
/* 在这两个符号之间的文字，  
都将被注释掉，编译器自动不进行编译，  
注释掉的文字将会呈现灰色 */
```

IDE 将自动把注释的文字颜色变为灰色。

注释接下来的一行是：

```
int ledPin = 10;
```

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

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

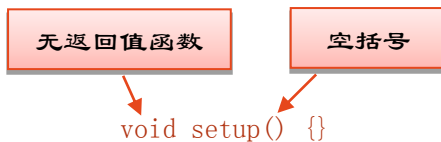
何为变量？

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

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

在给变量起名字时，还需要强调的一点。在 C 语言中，**变量名必须以一个字母开头**，之后可以包含字母、数字、下划线。注意 C 语言认为大小写字母是不同的。**C 语言中还有一些特定的名称也是不能使用的，比如 main, if, while 等。为了避免这些特定名称作为变量名，所有这些名称在程序中显示为橙色。**

接下来是 setup() 函数：



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

函数内部被花括号括起来的部分将会被依次执行，从“{”开始，“}”结束。两个符号之间的语句都属于这个函数。

函数

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

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

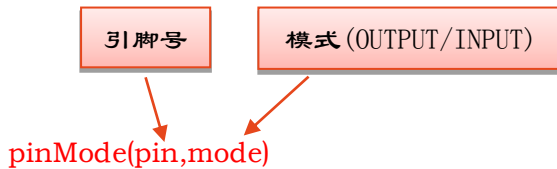
然而 setup 和 loop 函数比较特殊，一段代码中只能使用一次。

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

setup()函数内只有一条语句，那就是 pinMode 函数。

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

函数格式如下：



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

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

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

INPUT 与 OUTPUT 的区别

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

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

我们接着往下看，程序现在进行到我们的主函数 loop()：

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

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

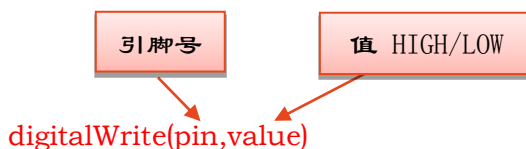
在 `setup()` 函数中初始化和定义了变量后，就开始执行 `loop()` 函数。顾名思义，该函数在程序运行过程中不断的循环，`loop()` 函数中的每条语句都逐次进行，直到函数的最后，然后再从 `loop` 函数的第一条语句再次开始，三次、四次……一直这样循环下去，直到关闭 Arduino 或者按下重启按钮。

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

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

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

函数格式如下：



这个函数的意义是：引脚 `pin` 在 `pinMode()` 的中被设置为 **OUTPUT** 模式时，其电压将被设置为相应的值，**HIGH** 为 5V（3.3V 控制板上为 3.3V），**LOW** 为 0V。我们这里就是给引脚 10（`ledPin`）一个 5V 的高电平，点亮了引脚 10 这个 LED。

我们这里强调了，`pinMode()` 被设置为 **OUTPUT** 时，才用到 `digitalWrite()`。这是为什么呢？看一下下面这段话：

`pinMode()` 与 `digitalWrite()`、`digitalRead()` 的关系

前面说了 `pinMode()` 中的 **INPUT** 与 **OUTPUT** 设置是有讲究的，是由器件本身的功能决定的。然而，前面设置 **INPUT** 和 **OUTPUT** 与之后程序需要如何执行也有着紧密关系的。既然 `pin` 是 **OUTPUT** 的话，那势必是控制器 Arduino 要给外界信号，所以需要 Arduino 给引脚先写入信号——`digitalWrite()`。我们这里还没用到 `digitalRead()`，就先说了吧！如果 `pin` 是 **INPUT** 的话，是外界给了控制器 Arduino 信号，所以需要 Arduino 读取引脚信号——`digitalRead()`。

对于初学者来说，可以先学着用，再慢慢弄明白里面的原由。`pinMode()` 设置为 **OUTPUT**，对应使用 `digitalWrite()`。**INPUT** 对应使用 `digitalRead()`。下表是一张对应表：

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

接着的一句语句：

单位：毫秒

`delay(1000);`

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

接着看下一句是：

`digitalWrite(ledPin, LOW);`

有了上面的引导，这句话是不是很容易理解了呢？这句话意思为，为引脚 10 一个 0V 的低电平，也就是熄灭 LED。

然后再延时 1 秒。之后回到 `loop()` 函数开始部分，循环运行。

现在我们知道代码是如何运作的了，让我们来个小小的改动吧！让 LED 保持关闭 5 秒，然后快速闪烁一下（250 毫秒），就像汽车报警器上的 LED 指示灯那样。试着写一下：

答案：

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

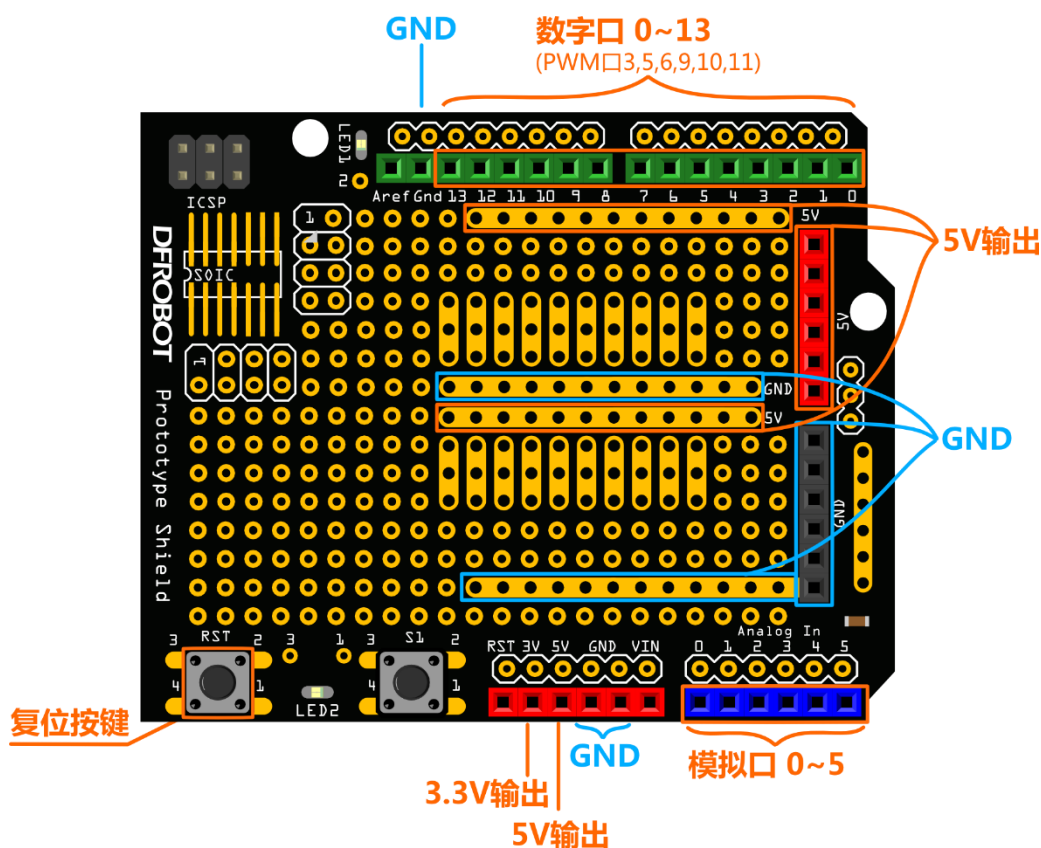
通过改变 LED 开和关的时间，可以产生不同的效果，开关时间短，则感觉动感，开关时间长，则感觉柔和。外面的灯光效果都是基于这样的原理。让我们再来看下硬件。看看硬件又是如何工作的。

硬件回顾

Prototype Shield 原型扩展板

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

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



面包板

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

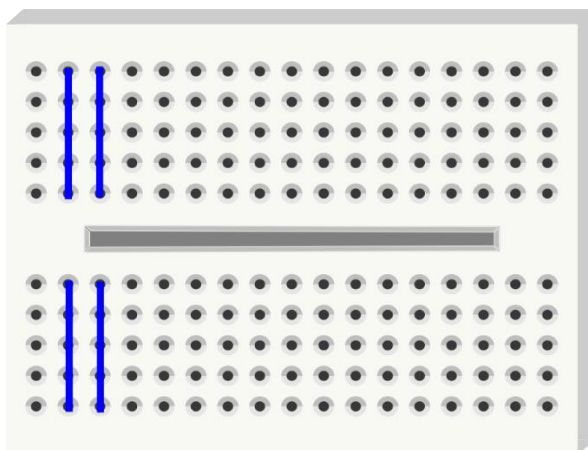


图 1-2 面包板内部导通图

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

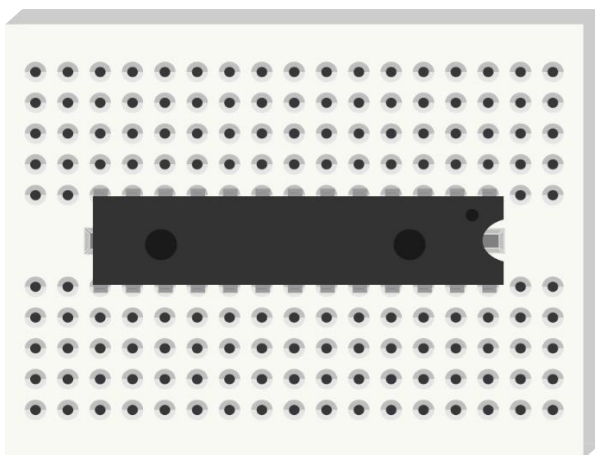


图 1-3 插入 DIP 芯片后

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

电阻

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

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

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

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

电阻色环读值

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

提供一个五色环电阻阻值在线计算器：

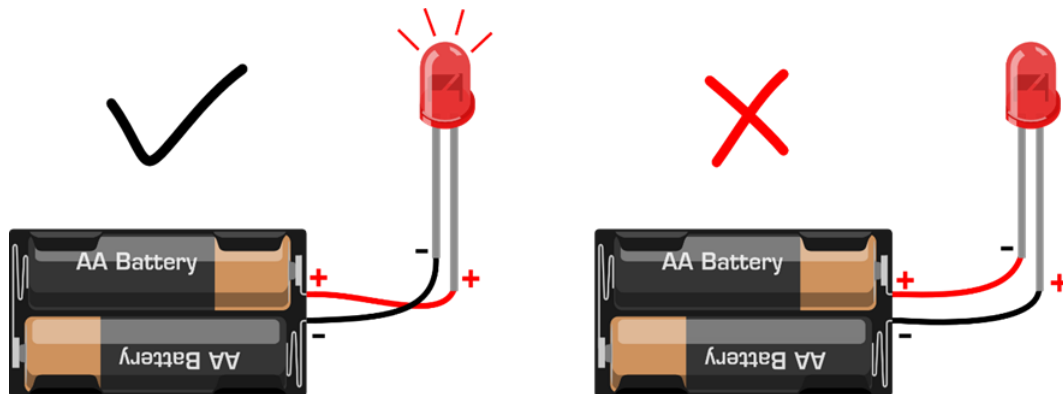
<http://www.21ic.com/tools/component/201003/54192.htm>

LED

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

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

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



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

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