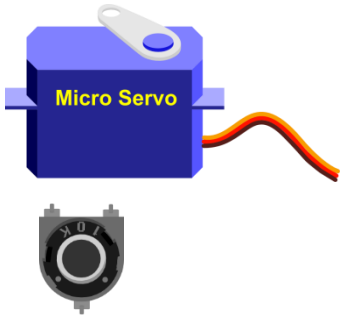
项目十一 可控舵机

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

所需元件



- 1× Micro Servo 9g
- 1× 10K 电位器

硬件连接

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

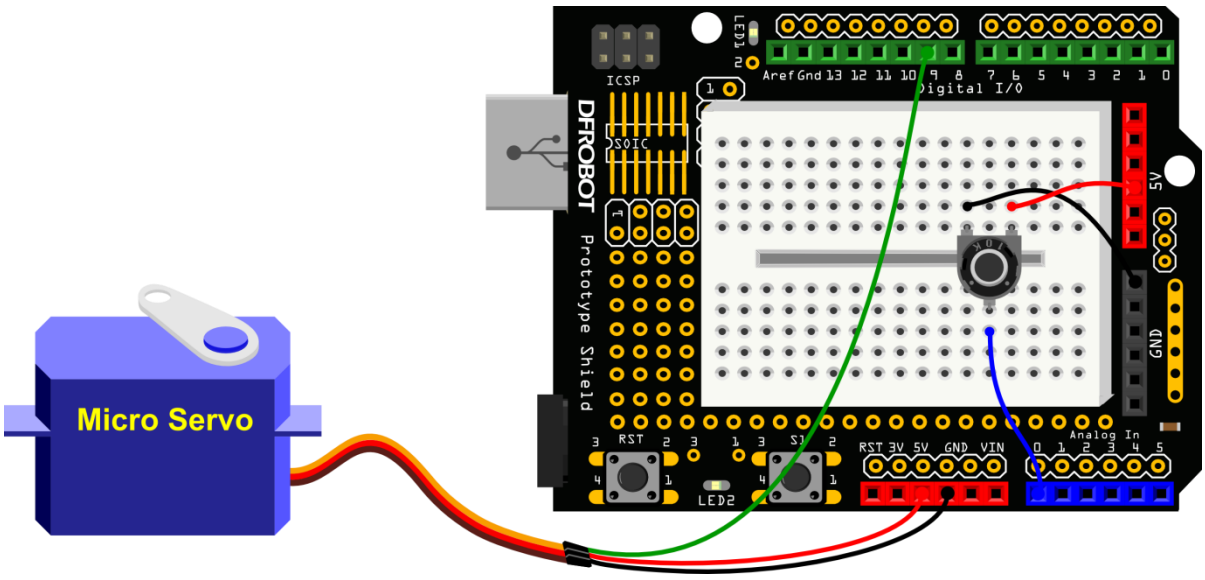


图 11-1 可控舵机连线图

输入代码

样例代码 11-1 ：

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

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

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

void loop() {
  val = analogRead(potpin);           //从模拟口 0 读值，并通过 val 记录
  val = map(val, 0, 1023, 0, 179);    //通过 map 函数进行数值转换
  myservo.write(val);                 // 给舵机写入角度
  delay(15);                          // 延时 15ms 让舵机转到指定位置
}
```

下载代码，成功后，旋转电位器，看看舵机是不是随着电位器转动。

代码回顾

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

这里主要讲下 map 函数。

函数格式如下：

map(value, fromLow, fromHigh, toLow, toHigh)

map 函数的作用是将一个数从一个范围映射到另外一个范围。也就是说，会将 fromLow 到 fromHigh 之间的值映射到 toLow 在 toHigh 之间的值。

map 函数参数含义：

value：需要映射的值

fromLow：当前范围值的下限

fromHigh：当前范围值的上限

toLow：目标范围值的下限

toHigh：目标范围值的上限

map 的神奇之处还在于，两个范围中的“下限”可以比“上限”更大或者更小，因此 map()函数可以用来翻转数值的范围，可以这么写：

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

这个函数同样可以处理负数，请看下面这个例子：

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

```
val = map(val, 0, 1023, 0, 179);
```

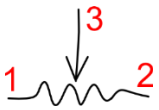
所以，回到代码中，我们是想将模拟口读到的 0~1023 的值，转换为舵机的 0~180° 。

硬件回顾

电位器

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

电位器在电路上的表示的图标为下图，分别对应器件上的 3 个引脚。



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