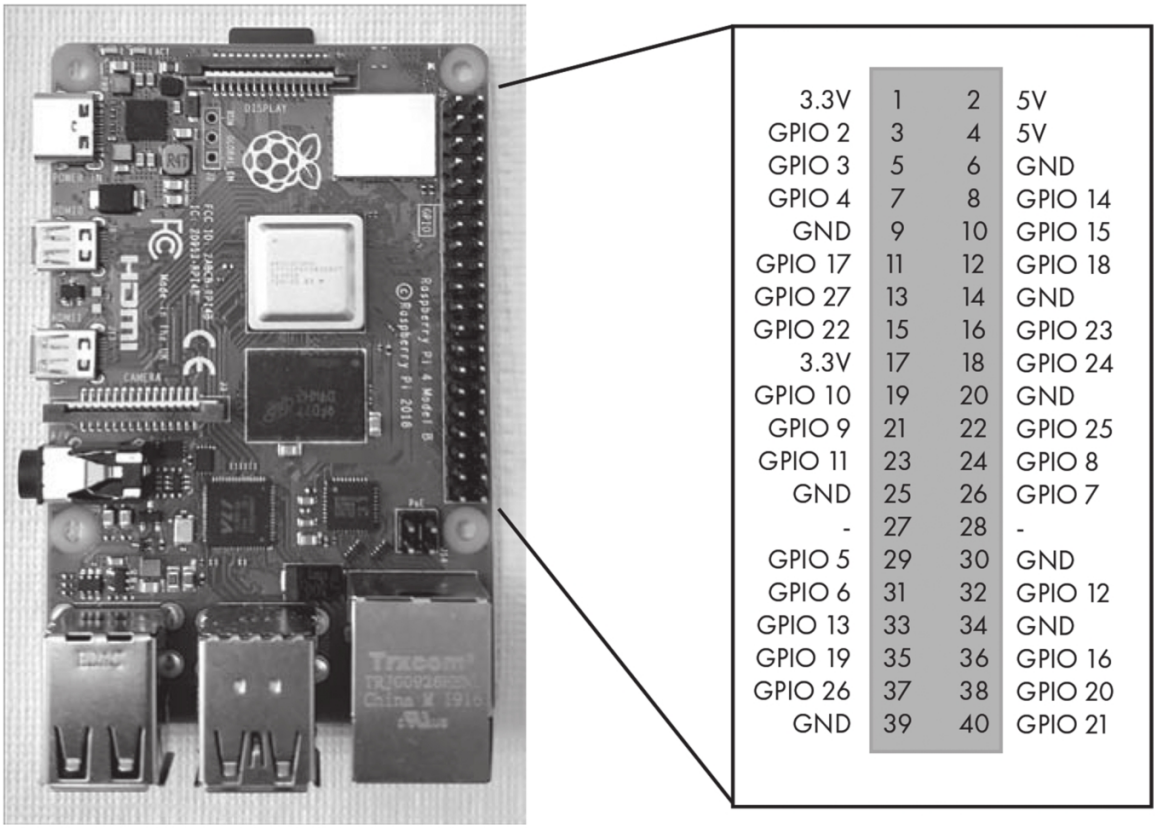

图13-11 Raspberry Pi的GPIO引脚

搭建电路

在编写代码之前，请按图13-12所展示的把电路元件连接到面包板和Raspberry Pi的GPIO引脚。

建议在连接GPIO引脚之前关掉Raspberry Pi的电源。GPIO引脚和面包板之间的连接请使用公对母跨接线。你可以把跨接线的母口连接到GPIO引脚，把公口连接到面包板。

如果使用图13-12所示的引脚编号，VEND LED、VEND按钮、COIN按钮要连接到Raspberry Pi上连续的3个GPIO引脚上。你还需要把一个GND引脚（建议用引脚9）连接到面包板的负电源列，以便把按钮和LED接地。

你可能已经注意到，这个电路的输入开关连线与设计7和设计8中使用的开关连线不同。在那两个设计中，我们把开关的一边连接到5V电源，另

一边连接到下拉电阻/输入引脚。这种电路设计使得开关断开时为低电压，开关闭合时为高电压。这里则正好相反——闭合开关为低电压断开开关为高电压。在内部，当开关断开（或未连接）时，Raspberry Pi会拉高GPIO引脚。

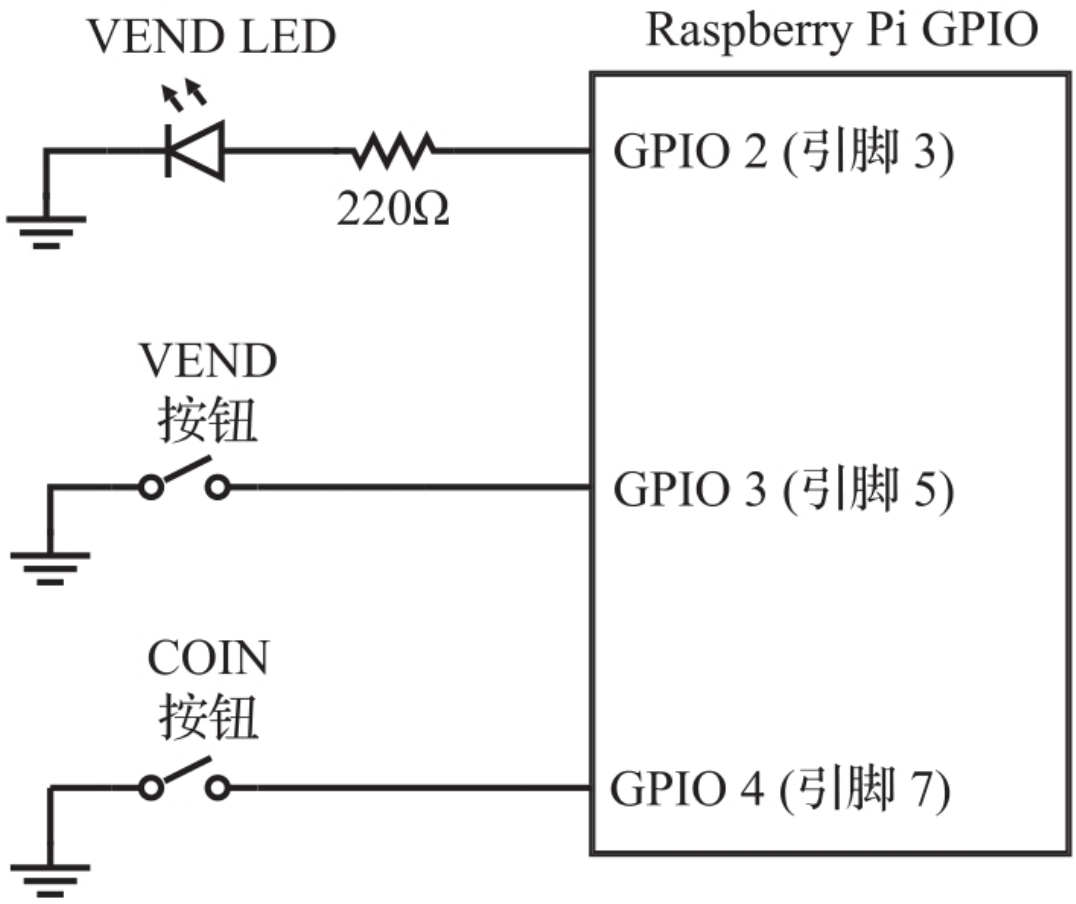

图13-12 Raspberry Pi自动贩卖机电路图

图13-13展示了在面包板上搭建的这个电路。

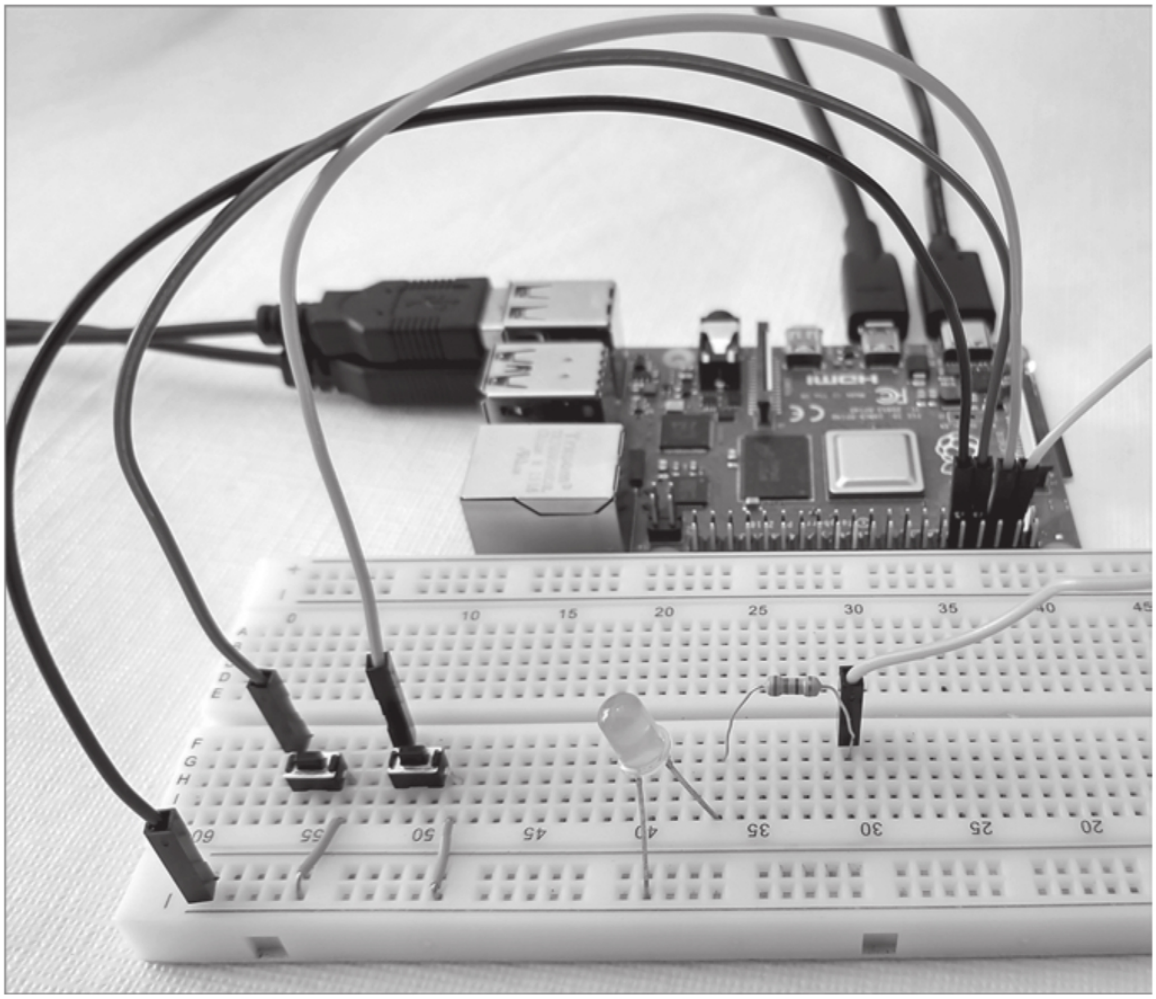

图13-13 面包板上的Raspberry Pi自动贩卖机电路

电路连接好并且已经验证过连接后，接通Raspberry Pi的电源。你可能会看到LED点亮，这没问题，因为你还没有运行任何代码来将LED设置为特定状态。

测试电路

在输入自动贩卖机代码之前，我们编写一个简单的程序来测试一下电路是否连接正确，并感受一下用Python驱动GPIO工作的感觉。为了与GPIO引脚交互，我们将使用GPIO Zero，这是一个可以轻松地与诸如按钮和LED等物理设备一起工作的Python库。使用文本编辑器，在主文件夹根目录中

创建一个名为gpiotest.py的新文件。在文本编辑器中输入如下Python代码。缩进在Python中很重要，所以请确保正确地缩进。

```
from time import sleep❶
from gpiozero import LED, Button❷

button = Button(3)❸
led = LED(2)❹

while True:❺
    led.off()
    button.wait_for_press()
    led.on()
    sleep(1)
```

这个简单的程序导入sleep函数❶以及来自GPIO Zero库的LED和Button类❷。然后，创建一个名为button的变量来表示GPIO 3上的物理按钮❸。同样，创建led变量来表示连接到GPIO 2的LED❹。之后，程序进入无限循环❺：关闭LED，等待按钮按下，然后点亮LED一秒钟，再次进入循环。

保存该文件后，你可以用Python解释器运行它，如下所示：

```
$ python3 gpiotest.py
```

当启动程序时，一开始应该什么都不发生，除了LED可能熄灭（如果之前它是亮的）。如果按下连接到GPIO 3的按钮，LED应该点亮一秒钟，然后熄灭。只要程序运行，你就可以重复该操作。

这个简单的程序不包含任何优雅的退出方式，所以要结束程序，请在键盘上按下<Ctrl+C>。当用这种方式退出程序时，Python解释器会显示最近函数调用的“Traceback”——这是正常的。

如果程序没有按预期工作，请仔细检查输入的代码并参考附录B排除电路故障。

自动贩卖机程序

在设计7和设计8中，自动贩卖机的逻辑由一个SR锁存器、一个AND门和一个电容器控制。现在，你可以用Raspberry Pi上的一个程序来代替它

们。这个新的设计也摆脱了COIN LED。在前面的设计中，如果投入硬币，则COIN LED就会点亮。在新的设计中，程序将输出投币数。每投入一枚硬币，投币数应增加1，每发生一次售出操作，投币数应减少1。

设备需求如下：

❑按下COIN按钮投币数加1。

❑按下VEND按钮模拟出售商品。如果投币数大于0，VEND LED将短暂亮起，投币数减1。如果投币数等于0，则按下VEND按钮时没有反应。

❑每按下一个可操作的按钮，不论是COIN还是VEND，都会使得程序输出当前投币数。

使用文本编辑器，在主文件夹根目录中创建一个名为vending.py的新文件。在文本编辑器中输入如下Python代码：

```
from time import sleep❶
from gpiozero import LED, Button

vend_led = LED(2)❷
vend_button = Button(3)
coin_button = Button(4)
coin_count = 0❸
vend_count = 0

def print_credits():❹
    print('Credits: {0}'.format(coin_count - vend_count))
```

```
def coin_button_pressed():❺
    global coin_count
    coin_count += 1
    print_credits()

def vend_button_pressed():❻
    global vend_count
    if coin_count > vend_count:
        vend_count += 1
        print_credits()
        vend_led.on()
        sleep(0.3)
        vend_led.off()

coin_button.when_pressed = coin_button_pressed❼
vend_button.when_pressed = vend_button_pressed

input('Press Enter to exit the program.\n')❽
```

首先，代码导入sleep函数、LED类和Button类❶，它们都会在后面的程序中用到。然后，声明3个变量来表示连接到GPIO引脚的物理组件——GPIO 2上的vend_led、GPIO 3上的vend_button，以及GPIO 4上的coin_button❷。变量coin_count被声明用于跟踪COIN按钮被按下的次数，变量vend_count被声明用于跟踪自动售出操作发生的次数❸。这两个变量用于计算投币数。

函数print_credits❹输出可用投币数，即coin_count和vend_count的差值。

函数coin_button_pressed❺是按下COIN按钮时运行的代码。它将coin_count加1，并输出投币数。global coin_count语句允许在函数coin_button_pressed内修改全局变量coin_count。

函数vend_button_pressed❻是按下VEND按钮时运行的代码。如果投币数满足要求（coin_count>vend_count），则该函数将vend_count加1，输出投币数，并点亮LED 0.3秒。

设置coin_button.when_pressed=coin_button_pressed❼把coin_button_pressed函数与GPIO 4上的coin_button关联起来，以便在按钮按下时运行该函数。同样，把vend_button_pressed与vend_button关联起来。

最后，我们调用input函数❽。这个函数在屏幕上输出一条消息，并等待用户按下回车键。这是一种简单的保持程序运行的方法。没有这行代码，程序会到达其终点，并在用户有机会与按钮交互之前停止运行。

保存该文件后，你可以用Python解释器运行它，如下所示：

```
$ python3 vending.py
```

启动程序时，你应该立即看到Press Enter to exit the program.显示在终端窗口。此时，请尝试按下COIN按钮，这个按钮连接到GPIO 4。你应该看到程序输出Credits:1。接下来，请尝试按下VEND按钮。LED应该短暂地亮起，程序应输出Credits:0。尝试再次按下VEND按钮——无事发生。尝试多次按下COIN和VEND，确保程序按预期工作。完成程序测试后，按回车键结束程序。

如你所见，Raspberry Pi或类似的设备可以用软件复制与我们之前用硬件实现的相同的逻辑。但是，基于软件的解决方案更容易修改。改变几行代码就可以添加新的功能，无须增加新的芯片，也无须重新布线。Raspberry Pi实际上对于我们在这里想要做的事情来说有点大材小用了；同样的事情也可以用性能较差的设备以更低的成本来完成，但其中的原理是一样的。

IoT自动贩卖机

假设自动贩卖机的操作员想通过互联网远程查看机器状态。由于你使用Raspberry Pi实现自动贩卖机的逻辑，因此可以更进一步，把它变成一个IoT自动贩卖机！你可以在程序中添加一个简单的网络服务器，允许从网络浏览器连接到该设备的IP地址，查看投币了多少次，售出操作发生了多少次。

Python让这变得相对容易，因为它含有简单的网络服务器库http.server。你只需要构造一些包含想要发送数据的HTML，并为传入的GET请求编写一个处理程序。你还需要在程序开始时启动网络服务器。

使用文本编辑器在主文件夹根目录中编辑现有的vending.py文件。首先插入如下import语句作为文件的第一行（保持所有已有代码不变，只是向下移动一行）：

```
from http.server import BaseHTTPRequestHandler, HTTPServer
```

然后，删除文件末尾的整个input行，并在文件结尾的地方添加如下代码：

```
HTML_CONTENT = """
<!DOCTYPE html>
<html>
  <head><title>Vending Info</title></head>
  <body>
    <h1>Vending Info</h1>
    <p>Total Coins Inserted: {0}</p>
    <p>Total Vending Operations: {1}</p>
  </body>
</html>
"""

class WebHandler(BaseHTTPRequestHandler):❶
    def do_GET(self):❷
        self.send_response(200)❸
        self.send_header('Content-type', 'text/html')❹
        self.end_headers()
        response_body = HTML_CONTENT.format(coin_count, vend_count).encode()❺
        self.wfile.write(response_body)


print('Press CTRL-C to exit program.')
server = HTTPServer(('', 8080), WebHandler)❻
try:❼
    server.serve_forever()❽
except KeyboardInterrupt:
    pass
finally:
    server.server_close()❾
```

HTML_CONTENT是一个多行字符串，它定义了程序通过网络发送的HTML代码。这个HTML代码块表示一个简单的网页，其中包括了一个<title>、一个<h1>标题和两个<p>段落以描述自动贩卖机的状态。这些段落中的特定值表示为占位符{0}和{1}。这些值由程序在运行时填充。由于这是HTML，因此字符串中的空格和换行并不重要。

WebHandler类❶描述了网络服务器如何处理传入的HTTP请求。它继承自BaseHTTPRequestHandler类，这意味着它具有与BaseHTTPRequestHandler相同的方法和字段。但是，这只是为你提供了一个通用的HTTP请求处理程序，你仍然需要指定你的程序应如何响应特定的HTTP请求。在本设计中，程序只需要响应HTTP GET请求，所以代码定义了do_GET方法❷。当GET请求到达服务器时，调用这个方法，它会回复如下内容：

❑表示成功的状态码200❸；

❑Content-type:text/html，告诉浏览器期待的响应是HTML❹；

❑前面定义的HTML字符串，但两个占位符已经被替换为coin_count和vend_count的值❺。

网络服务器的实例是用HTTPServer类创建的❻。这里指定的服务器名称可以是任何名称，HTTP服务器在端口8080（",8080）上监听。这里也是你指定对进入的HTTP请求使用WebHandler类的地方。

网络服务器用server.serve_forever()启动❽。它被放置在try/except/finally代码块中❼，这样服务器可以持续运行，直到发生KeyboardInterrupt异常（由<Ctrl+C>生成）。当发生这种情况时，调用server.server_close()进行清理并结束程序❾。

保存该文件后，你可以用Python解释器运行它，如下所示：

```
$ python3 vending.py
```

当按下COIN或VEND按钮时，这个程序应该像之前一样运行。不过，现在你还可以从网络浏览器连接到该设备并查看关于自动贩卖机的数据。为此，你需要一个与Raspberry Pi在同一本地网络的设备，除非你的Raspberry Pi用公网IP地址直接连接到互联网，在这种情况下，互联网上的任何设备应该都可以与之连接。如果你没有其他设备，那么你可以启动Raspberry Pi自身的网络浏览器，让Raspberry Pi同时充当客户端和服务器。

你需要找到Raspberry Pi的IP地址。如果你想了解查找IP地址的详细信息，请参阅设计30，不过，你应该使用以下命令：

```
$ ifconfig
```

有了Raspberry Pi的IP地址后，在作为客户端的设备上打开网络浏览器。在地址栏输入http://IP:8080，用Raspberry Pi的IP地址替换其中的“IP”。最终结果应该是类似http://192.168.1.41:8080这样的。把它输入浏览器的地址栏后，你应该看到网页加载了投币数和售出操作数。每次请求该页面时，都应该看到Python程序把关于请求的信息输出到终端。网页加载后，它不会自动重载，所以如果你又按下了COIN或VEND按钮，并想查看最新值的话，需要刷新网络浏览器的网页。要停止这个程序，请在键盘上按下<Ctrl+C>。

回忆一下第12章，网站不是静态的，就是动态的。在第12章的设计中，你能运行的网站是静态的，它提供提前准备好的内容。本章的自动贩卖机网站是动态的。当请求到来时，它生成HTML响应。具体来说，它在响应前更新HTML内容中的投币数和售出操作数。

作为附加挑战，请尝试修改程序，使其在网页上也显示“投币数”的值。这个值应该与输出到终端的最新投币数匹配。

附录

附录A 参考答案

这里给出了本书中练习题的答案。有些题目没有唯一的正确答案，对于这些问题，本书给出了一个示例答案。如果你在查看本部分的答案之前就自己想出了答案，那么你将从这些练习题中获得最大的收获！

练习1-1：略

练习1-2：

答案：

10（二进制）=2（十进制）

111（二进制）=7（十进制）

1010（二进制）=10（十进制）

练习1-3：

答案：

3（十进制）=11（二进制）

8（十进制）=1000（二进制）

14（十进制）=1110（二进制）

练习1-4：

答案：

10（二进制）=2（十六进制）

11110000（二进制）=F0（十六进制）

练习1-5:

答案:

1A（十六进制）=0001 1010（二进制）

C3A0（十六进制）=1100 0011 1010 0000（二进制）

练习2-1:

答案:

用8位数表示A～D的方案见表A-1。

表A-1 用字节表示A～D的自定义系统

字符	二进制
A	00000001
B	00000010
C	00000011
D	00000100

DAD用本系统表示应该为00000100 00000001 00000100（为清晰起见，添加了空格）。写成十六进制为：0x040104。

练习2-2:

（1）答案:

文本 Hello

二进制 01001000 01100101 01101100 01101100 01101111

十六进制 0x48656C6C6F

文本 5 cats

二进
制 00110101 00100000 01100011 01100001 01110100 01110011

十六进制 0x352063617473

请注意，对“5 cats”编码，得到字符“5”的二进制表示为0b00110101。这和数字5是不一样的，数字5的编码为0b101。这里的字符代表数字5的符号（5），而数字代表的是数量。同样是对“5 cats”的编码，注意，即使是空格符（你可能认为是空的）也需要用一个字节表示。

（2）答案：

二进
制 01000011 01101111 01100110 01100110 01100101 01100101

文本 Coffee

二进制 01010011 01101000 01101111 01110000

文本 Shop

（3）答案：

十六进制 43 6C 61 72 69 6E 65 74

文本 Clarinet

练习2-3：

答案：

表A-2 表示灰度的自定义系统

颜色	二进制
黑色	00
深灰色	01
浅灰色	10
白色	11

如果我们用2位系统，那么2位数的4个不同数值为00、01、10和11。这四个二进制数都可以映射到一种颜色：黑色、白色、深灰色和浅灰色——具体的映射关系由你决定，因为是你在设计系统。表A-2给出了一个示例答案，这里的正确答案不止一个。

练习2-4：

（1）答案：

假设图像总是4×4像素的，那么我们需要表示16个像素，每个像素一种颜色。使用前面表A-2定义的灰度表示系统，我们需要2位来表示每个像素的颜色。要表示完整图像的16个像素，且每个像素2位，总共需要16×2=32位。

当用二进制编码数据时，我们应该按什么顺序来表示这16个像素呢？虽然有点武断，但是在本例中，我们按照从左到右、从上到下的顺序排列数据，如图A-1所示。

使用图A-1展示的方法，当我们用二进制编码数据时，开始的2位代表像素1的颜色，其后2位代表像素2的颜色，以此类推。然后，我们用前面练习中定义的颜色代码来定义每个像素的颜色。例如，如果像素1是白色，像素2是黑色，像素3是深灰色，那么图像数据的前6位就为110001。

（2）答案：

这里，我将通过（1）给出的示例方法来写出图2-1中花朵图像的二进制表示，以说明如何解决这个问题。为了更直观地展示，图A-2在花朵图像上覆盖了已编号的图像网格。

现在，我们已经给网格中的每个像素分配了一个数字，我们可以参考表A-2对每个方块应用一个2位的数值。最终结果是一个二进制序列：1111110111101101111110101100101。它表示灰度花朵图像。

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16