

6.3.1 建立 3D 模型

有多种方式可以给数据建模，而我们将使用单个立方体顶点数据代表图形上的每个方块并通过变换矩阵来操作它。星形就是简单地使用一组随机出现的点。我们先来处理块状物体。

每个块都有相同的一组顶点和索引。它们之间的区别就在于大小（伸缩矩阵）和位置（变换矩阵）。每个块还有另外两个矩阵：一个用于旋转（每帧改变一次），另一个用于缩放（启动时设置）。

你需要将所有这些矩阵合并在一起。可以使用 Python 的 `transforms.compose()` 方法。然而，这样每帧都需要处理大量数据。我们提到过 GPU 远比 CPU 强大，因此让 GPU 来处理所有的矩阵操作会更有效。

渲染程序如下：

```
vertex_glsl = array_spec.glsl() + """
uniform mat4 position_matrix;
uniform mat4 eye_matrix;
uniform mat4 scaling_matrix;
uniform mat4 sound_matrix;
uniform float point_size;

varying float red;

void main(void) {
    gl_Position = eye_matrix * position_matrix * scaling_matrix *
                  sound_matrix * vec4(vertex_attrib, 1.0);

    red = (gl_Position[1]+0.9)/2.0;

    gl_PointSize = point_size;
}
"""

fragment_glsl = """
uniform vec4 color;
varying float red;

void main(void) {
    gl_FragColor = vec4(red, 0.0, (1.0-red)/5.0, 1.0);
}
"""
```

这些代码和我们之前见到的非常相似。矩阵乘法不可交换并没有意义^①。这里以某种方式展示了矩阵乘法和顺序是有关系的。顺序并不总是影响结果，但有些时候会。本

① 满足乘法交换律的方阵称为可交换矩阵，即矩阵 A、B 满足 $A*B=B*A$ 。——译者注

例中，确保 `eye_matrix`（用来旋转整个场景）在 `position_matrix`（用来将单个长条放在正确位置）之前非常重要。如果交换了两者顺序，就会变成单个长条围绕这某个点旋转而不是整个场景在旋转。这并不是什么大问题，我们只是让它看起来感觉更好一点。你希望矩阵以什么顺序变形，就可以相应地改变矩阵的乘法顺序（尝试一下，如果失败就再试试）。

你可能注意到这里有一个 `varying` 类型的变量 `red`。它作为 `gl_Position` 向量的纵轴（`y` 轴或者 `1`）并将其转换为颜色值。

你可能还注意到，点的尺寸由一个 `uniform` 变量设置，我们稍后再来处理它。

6.3.2 计算声音强度

每显示一帧，程序将计算当前的声音强度并将其中一个长条设置成相应的声音强度。我们将循环所有的长条以使它们作为一个整体显示出声音强度在最近的 20 帧中的变化。

为此，你要能够计算出声音的强度。使用 `on_frame()` 方法可以做到这一点。

```
scale_factor = 0

if frame_position + 1000 < len(data):
    for i in range(1,500):
        scale_factor1 = scale_factor +
            int.from_bytes(data[frame_position+2*i:
                                frame_position+(2*i)+1],
                            byteorder='little', signed=True)**2
    scale_factor1 = scale_factor1 / 500

self.sound_matrix[self.counter%20] =
    transforms.stretching(1.0,scale_factor1,1.0)

self.program1.uniform.point_size.value =
    float(scale_factor1/4)

self.counter = self.counter+1
```

从声音的某个位置（用音轨开始播放起到当前的时间间隔来表示）起，取其后的 500 帧计算它们的平方和均值可以得到当前位置的声音强度。使用声音强度的平方有两个好处。首先他可以确保每个值都是正数，因为一个负数的平方还是正数；其次音乐每增长一点，它可以使长条的变化更快，在屏幕上的效果更好。这里只是为了使效果看起来更好一点，并不需要科学地计算出正确图形，因此，使用这种方式计算是可以接受的。

使用 500 帧相加的原因很简单，因为相加的帧越多，结果越精确，但是计算会越慢。我们使用 500 是权衡了性能（与显示帧率有关）和音乐模型的精确性。如果你想使之更完善，可以增加一些前面章节描述的功能。

注意我们还设置了星形的尺寸。我们用来缩小比例因子的值（方块是 500，星形是 4），是通过视觉效果——看起来是否好看而不是通过计算来选择的。

视觉效果的完整代码如下所示（参见网站上的 `chapter6-music.py`）。

```
import pygame
from rpigl import glesutils, transforms
from rpigl.gles2 import *
import random
import wave
import time

vertices = [(0.0,0.0,0.0), (0.5,0.0,0.0),
            (0.5,0.5,0.0), (0.0, 0.5,0.0),
            (0.0,0.0,-0.5), (0.5,0.0,-0.5),
            (0.5,0.5,-0.5), (0.0, 0.5,-0.5)]

indices_outer = (0, 1, 2, 3, 0, 4, 5, 1, 5, 6, 2, 6, 7, 3, 7, 4)
vertices_points = []
indices_points = []

for i in range(0,100):
    vertices_points.append(((20 * random.random())-10,
                           (20 * random.random()), (20 * random.random())-10))
    indices_points.append(i)

array_spec = glesutils.ArraySpec("vertex_attrib:3f")

vertex_glsl = array_spec.glsl() + """
uniform mat4 position_matrix;
uniform mat4 eye_matrix;
uniform mat4 scaling_matrix;
uniform mat4 sound_matrix;
uniform float point_size;

varying float red;

void main(void) {
    gl_Position = eye_matrix * position_matrix * scaling_matrix *
    sound_matrix * vec4(vertex_attrib, 1.0);
    red = (gl_Position[1]+0.9)/2.0;
    gl_PointSize = point_size;
}
"""

fragment_glsl = """
```

```

uniform vec4 color;
varying float red;

void main(void) {
    gl_FragColor = vec4(red, 0.0, (1.0-red)/5.0, 1.0);
}
"""

class MyWindow(glesutils.GameWindow):

    def init(self):
        self.angle_x = 5
        self.angle_y = 10
        self.angle_z = 5
        self.counter = 0
        self.vertex_shader = glesutils.VertexShader(vertex_glsl)
        self.fragment_shader = glesutils.FragmentShader(
            fragment_glsl)

        self.program1 = glesutils.Program(self.vertex_shader,
                                           self.fragment_shader)
        self.program1.use()

        glClearDepthf(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)

        glClearColor(0.0, 0.0, 0.0, 1)

        self.program1.uniform.light_dir.value = ((0, 1, -1))

        self.verteces_buffer = array_spec.create_buffer(
            vertex_attrib=vertices)
        self.points_buffer = array_spec.create_buffer(
            vertex_attrib=vertices_points)

        self.elements_outer = glesutils.ElementBuffer(
            indices_outer)
        self.elements_points = glesutils.ElementBuffer(
            indices_points)

        self.blank_matrix = transforms.translation(0.0, 0.0, 0.0)

        self.position_matrix = []
        for i in range(0,20):

            self.position_matrix.append(transforms.translation((i/10)-0.95,
                                                                0.0, 0.0))

        self.sound_matrix = []
        for i in range(0,20):

```

```

        self.sound_matrix.append(transforms.translation(
            0.0, 0.0, 0.0))

self.program1.uniform.scaling_matrix.value =
    transforms.scaling(0.1)
self.program1.uniform.eye_matrix.value =
    transforms.compose(
        transforms.rotation_degrees(self.angle_z, "z"),
        transforms.rotation_degrees(self.angle_x, "y"),
        transforms.rotation_degrees(self.angle_x, "x"),
        transforms.translation(0.0, -0.9, 0.0))
self.counter = 0

def on_frame(self, ftime):
    global start_time
    global data

    self.angle_y = self.angle_y + .5

    self.program1.uniform.eye_matrix.value =
        transforms.compose(
            transforms.rotation_degrees(self.angle_z, "z"),
            transforms.rotation_degrees(self.angle_y, "y"),
            transforms.rotation_degrees(self.angle_x, "x"),
            transforms.translation(0.0, -0.9, 0.0))

    frame_position = int((pygame.time.get_ticks() -
        start_time) * 44.1 * 4)

    scale_factor = 0

    if frame_position + 1000 < len(data):
        for i in range(1,500):
            scale_factor1 = scale_factor +
                int.from_bytes(data[frame_position+2*i:
                    frame_position+(2*i)+1],
                    byteorder='little', signed=True)**2

            scale_factor1 = scale_factor1 / 500

        self.sound_matrix[self.counter%20] =
            transforms.stretching(1.0, scale_factor1, 1.0)

        self.program1.uniform.point_size.value =
            float(scale_factor1/4)

        self.counter = self.counter+1

    self.redraw()

```

```

def draw(self):
    self.program1.uniform.color.value = (1, 1, 1, 1)
    for i in range(0,20):
        self.program1.uniform.position_matrix.value =
            self.position_matrix[i]
        self.program1.uniform.sound_matrix.value =
            self.sound_matrix[i]
        self.verteces_buffer.draw(elements=self.elements_outer,
            mode=GL_LINE_STRIP)
        self.counter = self.counter +1

    self.program1.uniform.position_matrix.value =
        self.blank_matrix
    self.program1.uniform.sound_matrix.value =
        self.blank_matrix
    self.points_buffer.draw(mode=GL_POINTS)

print("starting pygame mixer")
pygame.mixer.pre_init(44100, -16, 2, 2048)
pygame.mixer.init()

music = pygame.mixer.Sound("tell.wav")

print("opening file")
sound_file = wave.open("test.wav", 'rb')

print("getting parameters")
(channels, sample_size, frame_rate, frames, compression_type,
    compression_name) = sound_file.getparams()

print("Number of channels: ", channels)
print("Sample size: ", sample_size)
print("Frame rate: ", frame_rate)
print("Number of Frames: ", frames)
print("Compression type: ", compression_type)
print("Compression name: ", compression_name)

print("readframes")
data = sound_file.readframes(4*frames)
print(len(data))

print("starting audio")
music.play()
start_time = pygame.time.get_ticks()

print("start time: ", time.clock())

MyWindow(200, 200, pygame.RESIZABLE).run()

```

运行结果见图 6-3，有点令人惊喜的 80 年代感觉。

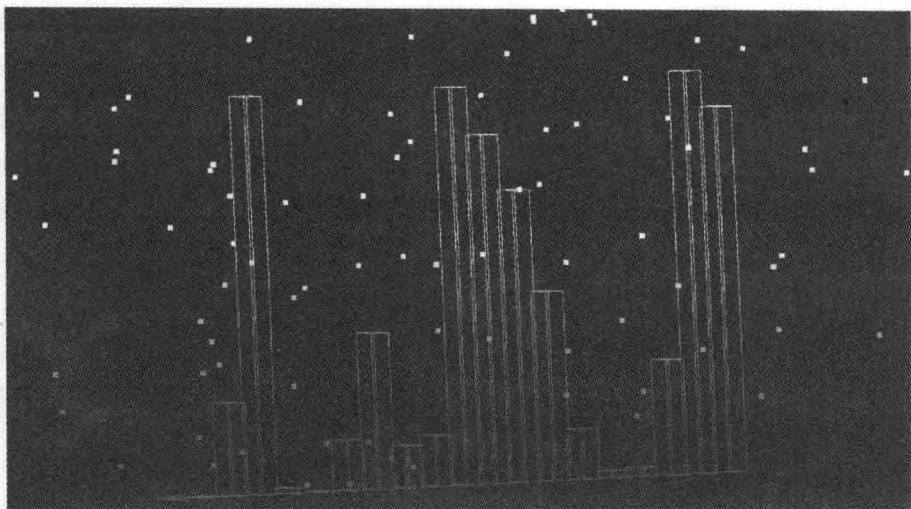


图 6-3 运行时看起来会更好。不要相信我们，自己运行程序看结果

6.4 继续完善

考虑下这些想法，继续完善该项目：

- 将星形设定为固定大小，但改变它们的亮度。可以用另一个 OpenGL 程序来完成，尽管这不是必须的。
- 还可以增加光照，为了更真实，需要用三角形带代替直线。
- 实验下改变方条和星形数量，可以让用户能够轻松配置它们的数量。
- 用图形显示频率图形而不是一直显示音量。为此，你需要使用 SciPi 模块计算音乐的傅里叶变换。注意这项任务需要一定数学知识，很有挑战性，有兴趣的读者可以试试。
- 提供一个更好的接口用来选择播放文件。根据个人喜好，可以选择命令行、文本菜单或者图形界面。这三种例子可以在本书别的部分找到。
- 改变视觉效果。让星星从上落下或者打旋都可以。还可以在旋转的时候缩小或放大它。颜色渲染也不用一直保持不变。要有创造性：3D 世界在等着你发挥想象力来塑造。

6.5 添加纹理

开始时我们说过这是本书中最复杂的一章。虽然它比较难，但也只是介绍了 OpenGL。

还有非常大一部分我们没有覆盖到。如果你觉得有趣并乐于继续深入探索，其中最有趣的一个部分就是纹理。它可以为你的 3D 模型添加很多细节。RPiGL 中提供了几个例子，可以从这些例子入手。在你之前下载的 Python 模块中的 `demos` 子文件夹中可以找到它们。

第一个是 `icosa.py`，它将世界地图设计成一个旋转的球体。图像取自 `world_cube_net.png` 并使用 `glesutils.Texture.from_surface()` 方法加载图像，然后使用 `bind()` 将其应用到着色器中。

第二个是 `bumpedsphere.py`，它工作地有点不一样。还记得你是如何使用法线来计算反射光线总量的么？使用这个文件可以创建一个带纹理的平滑表面。如果你改变法线，使其不再完全垂直于表面，这将会改变此处的光线，仿佛表面上有轻微的隆起。这其实是视觉误差，因为世界上不存在这样的表面，但效果却是平面上出现很多小隆起，并不需要使用巨大的三角形网就可以实现。



提示 关于 OpenGL 网上或者书中有很多有用的信息。但你需要记住并不是所有的 OpenGL 工作方式都一样。树莓派的 OpenGL 版本是 ES 2.0，因此你需要确认所做的工作使用的都是这个版本。

6.6 小结

读完本章，你应该更好地理解以下内容：

- OpenGL 为创建和操纵 3D 图形提供了一种强大的方法。
- 然而，与强大相伴的是复杂性。
- 坐标以向量方式存放，你可以通过将向量和变换矩阵相乘来操作它们。
- 如果有多个变换，你要将这些乘法串在一起。然而，相乘的顺序很重要。
- OpenGL 以一个独立的程序运行，它由两个着色器组成——顶点着色和像素着色。
- 这两个着色器不是用 Python 写的，而是由 GL 着色语言或者 GLSL，有点类似于 C 语言。
- 在 GLSL 中，每行都以分号结尾，代码段由花括号包围起来。
- 在 GLSL 中，uniform 变量由程序设置并且在整个 `draw()` 函数调用期间保持不变。varying 向量由顶点着色器设置，然后 OpenGL 会对其进行插值。
- OpenGL ES 不包含任何光线，因此你需要根据与光源的距离和法线与光源方向的余弦来自己计算。
- 纹理可以给物体增加细节。

Python 与网络

当今世界比以前更紧密地连接在一起，几乎你在计算机上做的每件事都有一定形式的联机部分。树莓派也是一样。只要你使用 Model B 树莓派或 USB 无线适配器，就可以非常简单地把手莓派连上互联网。你可以使用 Midori 浏览器浏览网页，还可以使用电子邮件客户端。这些都是用来消费内容的——从网页上获取信息或者使用其他人提供的服务。然而树莓派的强大之处在于创造。写几行 Python 代码，你就可以从网页上抓取信息或者使用树莓派为全世界提供内容和服务。本章将告诉你如何做到这一点。

7.1 理解主机、端口和套接字

为了和其他计算机通信，你需要知道该把数据送到哪里。可能你仅仅将信息传递给同一个房间内的另一台计算机，也可能你需要将信息发送给远在千里之外的另一方。无论如何，你都需要指定一个地址。网际协议（IP）地址通常被用来定位计算机。有两种 IP 地址，版本 4 和版本 6。在本书写作之时，正在大规模使用版本 4（IPv4），因此你读到的都是关于 IPv4 的。IPv6 地址工作方式和 IPv4 基本类似，因此当大规模使用 IPv6 时，你可以轻松地换到 IPv6 上去。

7.1.1 使用 IP 地址定位计算机

IPv4 地址一般的书写法为 4 个用小数点分开的十进制数。想要查看树莓派的 IP 地

址（不止一个地址），只需打开终端（不是 Python 程序，而是 LXTerminal 程序），然后运行 ifconfig。它将会输出如下内容：

```
eth0  Link encap:Ethernet  HWaddr b8:27:eb:f3:d5:23
      inet addr:192.168.0.13  Bcast:192.168.0.255
        Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:87523 errors:0 dropped:0 overruns:0 frame:0
      TX packets:59811 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:97997131 (93.4 MiB)  TX bytes:12573160 (11.9 MiB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

可以看到，这里有两个网络接口：eth0 和 lo。eth0 表示有线网络连接，lo 表示回环接口，回环到本主机。本例中，以太网连接使用 192.168.0.13 的 IP 地址，而回环口使用 127.0.0.1（通常都是这个地址）。

除了使用 IP 地址来定位计算机外，我们还可以使用域名（例如 www.google.com，或 localhost）。域名比 IP 地址方便记忆。使用域名时，计算机会首先连接到域名服务器并询问该域名对应的 IP 地址，然后访问域名服务器返回的 IP 地址。Localhost 有点特殊，由于它就是 127.0.0.1，代表了本地计算机。

如果把 IP 地址（或者域名）比作现实世界中的楼栋地址，对于同一栋楼来说，你还需要具体到每个人的地址。在计算机网络中，我们使用端口（port）来做到这一点。如果你有一个对外提供服务的软件，它将在特定的端口上监听。客户端就可以连接到该端口上获取信息。有些端口被指定用来提供特定服务。如网络服务通常监听在端口 80 上，SSH 连接使用端口 22。

Python 中的套接字（socket）对象可以用来连接目标主机的目标端口，或者监听在目标端口上等待客户端连接。连接建立之后，就可以使用套接字来收发数据。听起来有点复杂，实际做起来非常简单。理解这个过程最简单的方法就是举例实践。

7.1.2 搭建会话服务器

和本书中大多数程序不同，本程序分为两部分，并且需要同时运行：一个客户端和一个服务器。服务器运行之后处于等待状态，等待连接进来。客户端用来建立连接。下面是

服务器端的代码：

```
import socket

comms_socket = socket.socket()
comms_socket.bind(('localhost', 50000))
comms_socket.listen(10)
connection, address = comms_socket.accept()

while True:
    print(connection.recv(4096).decode("UTF-8"))
    send_data = input("Reply: ")
    connection.send(bytes(send_data, "UTF-8"))
```

看得出来，大部分的功能都来自于套接字模块。对于服务器套接字，你首先要把它绑定到主机的某个端口上。（临时使用时，可以随意选择 50 000 以上的端口号。实际上，你可以使用任意当前没被使用的端口号，但最好不要使用 100 以下的端口号，除非你确定没有人使用它。）listen() 使其进入等待状态，等待连接。当有连接进来时，程序跳转到下面一行：

```
connection, address = comms_socket.accept()
```

它会创建一个新套接字（存储在变量 connection 中）。该套接字和客户端连接在一起。现在你可以通过这个新连接使用 send() 和 recv() 收发数据。数据以字节流而不是字符串的形式存在，因此你需要在字节和 UTF-8（通用字符集转换格式，8 位）之间相互转换。屏幕上显示的信息都是 UTF-8 编码的。^①

下面是客户端代码：

```
import socket

comms_socket = socket.socket()
comms_socket.connect(('localhost', 50000))
while True:
    send_data = input("message: ")
    comms_socket.send(bytes(send_data, "UTF-8"))
    print(comms_socket.recv(4096).decode("UTF-8"))
```

这次，我们不再给套接字绑定主机和端口号，而是直接连接到某个主机和端口。同时也不再创建新的套接字而是使用原有的套接字来收发数据。

除此之外，两个程序非常相似。要发起一个对话，需要两个 Python 会话，最简单的方法是打开两个 LXTerminal 窗口。在第一个窗口中输入 python3 server.py（如果服务器端的代码存储在 server.py 中），在第二个窗口中输入 python3 client.py。

① 有很多种字符格式，有兴趣的读者可以在互联网上搜索相应信息。——译者注

你将看到信息在两个程序之间反复地传送。当然，这显得不是那么像“网络”。它们实际上运行在同一个主机上。这里只是演示了同一主机上两个程序之间的网络通信，但通常所说的网络指的是在两个主机间传递数据。

这两个程序具备所有在主机间通信的基本条件，但还需要一个菜单让用户可以输入他们想要访问的地址。如果一个程序可以同时处理客户端和服务端会更方便。下面这个程序就是基于这两点的改进版本（参见网站上的 `chapter7-chat.py`）：

```
import socket

def server():
    global port
    host = "localhost"

    comms_socket = socket.socket()
    comms_socket.bind((host, port))

    print("Waiting for a chat at ", host, " on port ", port)

    comms_socket.listen(10)
    send_data = ""

    while True:
        connection, address = comms_socket.accept()
        print("opening chat with ", address)
        while send_data != "EXIT":
            print(connection.recv(4096).decode("UTF-8"))
            send_data = input("Reply: ")
            connection.send(bytes(send_data, "UTF-8"))
            send_data = ""
        connection.close()

def client():
    global port
    host = input("Enter the host you want to communicate" +
                " with(leave blank for localhost) ")

    if host == "":
        host = "localhost"

    comms_socket = socket.socket()

    print("Starting a chat with ", host, " on port ", port)
    comms_socket.connect((host, port))
    while True:
        send_data = input("message: ")
        comms_socket.send(bytes(send_data, "UTF-8"))
        print(comms_socket.recv(4096).decode("UTF-8"))

port = int(input("Enter the port you want to communicate on" +
```

```

        " (0 for default)"))
    if port == 0:
        port = 50000
    while True:
        print("Your options are:")
        print("1 - wait for a chat")
        print("2 - initiate a chat")
        print("3 - exit")

        option = int(input("option :"))
        if option == 1:
            server()
        elif option == 2:
            client()
        elif option == 3:
            break
        else:
            print("I don't recognise that option")

```

你应该认识代码中的网络部分，剩下的部分也应该很熟悉了。为了在两个计算机之间通信，需要协商好端口号，设置好监听端的端口号，然后在另一端连接该端口。

该方法还有点小问题。第一点，两个会话端需要交替发送消息，其次你只能在本地网络中和别人通信。实际上，第二点还和你的本地网络是如何搭建的有关。还记得我们说过 IP 地址有点类似于楼栋地址么？多数情况下都是如此，局域网（LAN）类似于城镇。不同的城镇中两个不同的建筑物可以使用同一个地址“商业街 1 号”，同样，在不同的局域网中的两台不同主机也可以有相同的 IP 地址 192.168.1.2。有三个不同的 IP 地址段保留给本地网络使用：

- 10.0.0.0–10.255.255.255
- 72.16.0.0–172.31.255.255
- 192.168.0.0–192.168.255.255

所有这些 IPv4 地址都只能在本地网络中使用（即你只能和本地网络中的计算机通信），这些 IP 之外的地址都是公共地址（即所有因特网中的计算机都可以访问）。

有时从外部因特网也可以连接到本地 IP 地址。这取决于你的因特网提供商或者路由器（参见端口转发 port-forwarding 设置）。受限于本书篇幅和方向，这里不再展开描述相关内容。

7.1.3 “推”向世界

除了与你的 ISP 和路由器斗争外，还有许多其他方法可以避开这两个问题。目前为止，最简单的办法就是使用其他服务来帮你处理消息。推特就是这样的一种服务。它可以在单个计算机和全世界之间传递文本消息。

每个推特用户可以一次发送最长 140 个字符的“推特”。如果这些推特中提到其他推友（名字以 @ 开头），该推特将会出现在它们的连接页面上。如果没有推特账号，为完成本章内容，你需要注册一个。即使已经注册过账号，还是推荐你新注册一个以防将不必要的测试信息推送给所有关注你的人。注册账号是免费的，也很简单。只要登录 www.twitter.com 并根据提示一步步做就可以了。注册完成之后，你也可以浏览到网站的更多内容。

通常人们在网站上使用推特发送消息，但这并不是唯一的方法。除了使用网页接口，推特还为从应用程序收发消息提供了应用程序接口（API）。

推特模块为访问推特 API 提供了简便方法。Raspbian 中默认不包含该模块，因此你需要从 <https://github.com/sizohsix/twitter/tree/master> 中下载（使用右下角的下载 ZIP 包按钮下载）。完成之后，打开 LxTerminal 窗口，解压缩并安装模块：

```
unzip twitter-master.zip
cd twitter-master
python3 setup.py build
sudo python3 setup.py install
```

现在已经基本就绪，和推特用户名一样，还需要向推特注册你的应用程序，为你的应用程序取得合适的证书。

登录到推特后，访问 <http://dev.twitter.com>，然后在右上角的用户菜单中选择我的应用程序。在新的页面中点击创建应用程序。在表单中填入应用程序的详细信息。

- 名字在推特中必须是独一无二的，因此不能使用 `test-app` 这个名字。起个有创意的名字或者在名字后面添加几个特殊字符来保证没有人用过就可以了。
- 名字长度不能超过 10 个字符。
- 网站一栏并不一定指的是真实的网站，只是看起来像而已。我们输入 `http://none.example.com`。
- 回调 URL 可以不填。

除此之外，你还需要接受网站的规则并输入验证码。

应用程序页面有一系列选项卡。我们需要改变默认设置中的一项。选中设置（Setting）选项卡，将应用程序类型由只读改为读写。这样你就可以像读信息一样发布状态了。做完之后，按下更新按钮。这就是推特应用程序的设置。

现在你需要创建访问令牌，转到详情（Details）选项卡，单击创建我的访问令牌。完成之后，推特需要一段更新时间，你可能需要刷新页面，来获取你的访问令牌。一旦出现令牌，所有设置都已经完成。现在需要从网页上记下 4 个信息：访问令牌、访问令牌机密、消费密钥和消费机密。这些都是随机字符串，你需要把它们复制粘贴到下列代码中（参见

网站文件 chapter7-twitter.py):

```
import twitter

def print_tweets(tweets):
    for tweet in tweets:
        print('text: ', tweet['text'])
        print('from: ', tweet['user']['screen_name'])

twitter_user = twitter.Twitter(
    auth=twitter.OAuth("ACCESS-TOKEN", "ACCESS-TOKEN-SECRET",
                        "CONSUMER-KEY", "CONSUMER-SECRET"))

status = twitter_user.statuses

home = status.home_timeline()
print("home")
print_tweets(home)

mentions = status.mentions_timeline()
print("mentions")
print_tweets(mentions)

search_string = input("Enter text to search for, " +
                      "or press enter to skip: ")
if search_string != "":
    search = twitter_user.search.tweets(q=search_string)
    print("search")
    print_tweets(search['statuses'])

tweet = input("Enter tweet, or press enter to exit: ")

if tweet != "":
    twitter_user.statuses.update(status=tweet)
```

这个迷你推特客户端一定做不到用户友好。事实上，很难想象有人会用它而不是网页来发推特。然而，有很多不同的方式可以与推特交互，你可能希望把它们添加到你自己的应用程序中。改变程序满足你的需求，做起来应该会比较容易。

7.1.4 使用 JSON 做天气预报

上一个例子中，推特模块提供了所有的基本功能。但并不是每次都能找到这样的模块。有时你需要自己写代码完成和网络服务的交互。幸运的是，有一个发送数据往返的标准格式，它可以让你的程序能够轻松地完成与网络服务的交互。这个标准就是基于 JavaScript 语言的轻量级的数据交换格式 JavaScript Object Notation，通常称为 JSON。开始的时候，它是设计出来与 JavaScript（一种主流的网页编程语言）一起工作的，然而它在

Python 上工作得也很好。

OpenWeatherMap.org 是一个免费提供天气预报数据的网站，你可以把它加到软件中。刚好可以使用 JSON。将你的网页浏览器指向 <http://api.openweathermap.org/data/2.5/forecast/daily?cnt=7&units=meteric&mode=json&q=London> 来感受下 JSON 文档是什么样子。它将返回伦敦未来 7 天的天气预报数据。很难看懂，但是你应该注意到它看起来很像包含更多字典列表（其中还有别的东西）的 Python 字典。使用 `url-lib.request` 模块，Python 可以从因特网上获取这些信息，参见下列代码：

```
import urllib.request

url = http://api.openweathermap.org/data/2.5/forecast/" +
      "daily?cnt=7&units=meteric&mode=json&q=London"
req = urllib.request.Request(url)
print(urllib.request.urlopen(req).read())
```

这段代码将从 OpenWeatherMap.org 中抓取信息并将其显示在屏幕上。然而，数据都是字符串形式。虽然它们看起来像字典和列表，你却不能简单地按照字典和列表来存取它们。你需要构建函数读取整个字符串并将其分割开，但非常幸运，你用不着这么做。`json` 模块可以加载它并返回一个含有该字符串每个部分的字典。例如：

```
import urllib.request, json

url = http://api.openweathermap.org/data/2.5/forecast/
      "daily?cnt=7&units=meteric&mode=json&q=London"
req = urllib.request.Request(url)
forecast_string = urllib.request.urlopen(req).read()
forecast_dict = json.loads(forecast_string.decode("UTF-8"))

print(forecast_dict)
```

现在可以从 `forecast_dict` 数据结构中得到任何想要的信息。在下面的例子中，我们编写了一个简单的天气预报程序，它能够打印出指定城市未来 7 天的天气预报（参见网站上文件 `chapter7-weather.py`）：

```
import urllib.request, json

city = input("Enter City: ")

def getForecast(city) :
    url = http://api.openweathermap.org/data/2.5/forecast/ +
          "daily?cnt=7&units=meteric&mode=json&q="
    url = url + city
    req = urllib.request.Request(url)
    response=urllib.request.urlopen(req)
    return json.loads(response.read().decode("UTF-8"))
```

```

forecast = getForecast(city)

print("Forecast for ", city, forecast['city']['country'])

day_num=1
for day in forecast['list']:
    print("Day : ", day_num)
    print(day['weather'][0]['description'])
    print("Cloud Cover : ", day['clouds'])
    print("Temp Min : ", round(day['temp']['min']-273.15, 1),
          "degrees C")
    print("Temp Max : ", round(day['temp']['max']-273.15, 1),
          "degrees C")
    print("Humidity : ", day['humidity'], "%")
    print("Wind Speed : ", day['speed'], "m/s")
    print()
    day_num = day_num+1

```

注意温度的单位是开尔文，将其减去 273.15 就可以将开尔文转换为摄氏。本例仅使用了 API 返回的部分数据。查看一下天气预报数据结构，看看还有其他哪些有用的信息可以输出。

使用这些基本方法，你可以与任何支持 JSON 的 web API 交互。这个地址列出了常用的服务 www.programmableweb.com/apis/directory/1?sort=mashups，从中可以找到方法，让你的应用程序可以从网上获取任意想要的信息。

7.2 知识测验

至此，你已经见过如何向 API 查询并从中获取信息。是时候检验一下你是否完全理解它的工作过程了。试一下下面的练习，遇到任何不确定的问题都可以翻回去参考前面的例子。

练习 1

使用这个 URL 可以得到一个城市（例如伦敦）当前的天气 <http://api.openweathermap.org/data/2.5/weather?q=London>。使用这个 URL 创建一个应用程序将某个地点的当前天气推特出去。参考答案见本章结尾处。

7.3 走向网站

现在你已经见过如何在两个计算机之间传递数据，如何通过在线 API 上发送和接收数据。所有这些都遗漏了网站，毕竟最流行的访问在线信息的方式就是通过网站。你将在本节中学到如何将树莓派作为主机对外提供网页。

网站有两大部分：HTTP 和 HTML。前者是超文本传输协议（Hypertext Transfer Protocol，网络浏览器和网站之间通信时使用的方法），后者是超文本标记语言（Hypertext Markup Language，用来编写网页的语言）。超文本是一个前卫的名字，任何含有链接的文本都是超文本。有很多模块可以处理 HTTP，但你需要先学习一点 HTML 知识。

现代的 HTML 是一种复杂的语言，可以用来创建功能强大的应用程序，其中可以包括包含各种动作和交互操作。然而基本的语言却非常简单。每个网页都是一个独立的 HTML 文件，每个 HTML 文件都分为两部分，文件头和文件体。文件头包含了页面所需的各种信息，文件体包含了需要在屏幕上显示的内容。HTML 使用标签来描述文档的不同部分。几乎所有的标签都成对出现，包括一个起始标签（如 `<h1>` 表示主标题）和一个结束标签（如 `</h1>`）。标签都写在尖括号内，结束标签以斜线开头。下面的例子中使用了常见的基本标签（参见网站文件 `chapter7-htmleg1.html`）：

```
<!DOCTYPE html>
<head>
An example HTML file</title>
</head>
<body>
<h1>An h1 heading</h1>
<h2>An h2 heading</h2>
<p>
A paragraph of text with a <a href=http://www.raspberrypi.org";link
  <to the Raspberry Pi websites</a>
</p>
<p>
Another paragraph
</p>
</body>
```

除此之外 HTML 还有更多内容，但鉴于本书是一本关于 Python 而不是 HTML 的书，我们不再深入更多细节。了解这些差不多可以满足本章需求。如果你有兴趣了解更多，有很多不错的资源可以学习，这个网站是个不错的开始 <http://w3schools.org>。

保存好文件后可以立即在计算机上打开，它将会在浏览器中打开。这对于检验你的页面是否正确很有用，但这样还不能把它共享给全世界。为了做到这一点，要用到前面提到的 HTTP。在 Python 中，可以非常简单迅速地建立一个 HTTP 服务器。例如：

```
import http.server, os

os.chdir("/home/pi")
httpd = http.server.HTTPServer(('127.0.0.1', 8000),
    http.server.SimpleHTTPRequestHandler)
httpd.serve_forever()
```

它将会在你的树莓派上以 home 目录为根，运行一个使用 8000 端口的网络服务器。

如果你之前关注过端口号，应该还记得 80 是网站服务器的默认端口。我们这里使用 8000 是以防你还运行有其他的网络服务器，如果你喜欢，你也可以把它改成 80。

程序运行之后，可以在树莓派的网络浏览器中输入 `http://localhost:8000/`。`localhost` 通常指向当前运行的主机（它对应的 IP 地址是 127.0.0.1）。`:8000` 表示使用端口 8000，注意这里不是默认端口 80。你将会在浏览器中看到一份十分普通的文件列表。

HTTP 处理目录和文件的方式与电脑很相似。都是从根目录开始。本例中，根目录是 `/home/pi`。在网络浏览器中你可以指定一个目录（名字以 `/` 结尾）或者文件名。如果你进入一个目录，网络浏览器首先检查那里是否存在一个名叫 `index.html` 的文件。如果存在，将会显示文件而不是显示目录内容。把你的 HTML 文件改名成 `index.html`，然后用网络浏览器再次访问 `http://localhost:8000`。这次你将会看到文件内容。

7.3.1 让网站动起来（动态网站）

`http.server` 模块可以快速建立一个服务器用于分享信息，但不能很好地应对可变信息。虽然它提供了多种添加交互体验的方法，但还有更好选择。`tornado` 模块被设计来处理动态内容而不是存储在文件中的内容，非常适合那些需要额外的多功能性的项目。

与前一个基于操作系统文件结构的例子不同，Tornado 创建了自己的虚拟结构，并且你需要创建能够输出合适的 HTML 的类而不是直接使用 HTML 文件。首先你需要打开 LXTerminal 并输入下面一行命令来安装它：

```
sudo apt-get install python3-tornado
```

看看下面这个例子。它创建了一个“Hello World！”的网站。

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("<!DOCTYPE html><head><title>" +
                    "Hello world</title></head>" +
                    "<body>Hello World</body>")

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
    ],)
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

这个例子分为两部分。在第一部分中，我们定义了用于创建网页的类，第二部分（从

if `__name__ == "__main__"`: 开始) 定义并启动网站服务。这部分主要包括:

```
application = tornado.web.Application([
    (r"/", MainHandler),
],)
```

这句代码创建了一个 Tornado 网络应用并设置了它的选项。主要的选项是一个元组列表, 定义了哪个类处理哪个页面。本例中, 只有一个页面, 根 (root), 或者 /, 由 MainHandler 类来处理。这里还可以添加其他选项, 我们在后面的例子中再用。最后两行表示在端口 8888 (这样你就可以同时运行另一个在 8000 上监听的服务 `http.server`) 上监听, 并开始运行。现在运行程序, 在网络浏览器中指向 `http://localhost:8888` 检查下它的工作状态。

所有用于处理网页的类都必须继承自 `tornado.Web.RequestHandler`, 该类提供了所有的基本功能。你只需要添加一个 `get` 方法来调用 `self.write()` (或者后面将会见到的 `self.render()`), 超类将完成剩下的所有事情。

只定义类而不为它创建任何对象看起来有点奇怪。这是因为 Tornado 使用这个类为你创建了对象。

你可以这样使用 Tornado 来处理静态内容, 但不是非常好。所有的 HTML 都包含在 Python 代码中, 对于初学者来说有点混乱。Tornado 真正强大的地方体现在你开始创建可以变化的动态网页的时候。下一个例子将给用户带来一个特殊的问候。你可以使用上个例子的文件, 但需要根据下列代码创建一个新类:

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self, name):
        self.write("<!DOCTYPE html><head>
                    <title>Hello world</title></head>" +
                    "<body>Hello " + name + "</body>")
```

然后修改后半部分代码以使用这个类 (变化部分使用黑体标出):

```
if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r"/hello/(.*)", HelloHandler),
    ],)
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

修改好后保存并运行代码。在网络浏览器中访问 `http://localhost:8888/hello/Ben`, 或者 `http://localhost:8888/hello/Alex`, 或者使用你自己的名字。

7.3.2 使用模板

能更改 HTML 代码显然可以让你创造出比以前更强大的网站。然而, 这样还是将

HTML 包含在 Python 中，还不够令人满意。解决方案就是使用模板。在 HTML 文档中需要有一点 Python 代码，Tornado 需要使用它来构建网页。下面是 HelloHandler 的模板：

```
<!DOCTYPE html>
<head>
<title>Hello</title>
</head>
<body>
Hello {{ name }}
</body>
```

可以看到，需要打印出来的 Python 变量包围在双层花括号中。将其存为 `hello_template.html` 文件并保存在你的 `home` 目录下 (`/home/pi`)，然后更新 `HelloHandler` 类：

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self, name_in):
        self.name=name_in
        self.render("/home/ben/hello-template.html",
                    name=self.name)
```

重新运行程序。将会得到相同的结果，但看得出来，这段代码更整洁，也更容易维护。

7.3.3 使用表格回传数据

你可能已经注意到，有些网站需要通过网络地址获取你的输入信息。一般使用 `page ID`（如你见过的天气 API）来实现。

HTTP 有一个给服务器端回传信息的方法：POST 请求。到目前为止，我们只使用了 Tornado 的 GET 请求（即方法名字）。POST 允许你使用 HTML 表单回传复杂信息。使用下列代码创建一个新的 HTML 文件：

```
<!DOCTYPE html>
<head>
<title>User Information</title>
</head>
<body>
<h1>User Information</h1>
<form action="/hello/" method="post">
Enter your name: <input type="text" name="name">
<input type="submit" value="Sign in">
</form>
</body>
```

将其命名为 `user-info.html` 保存在你的 `home` 目录中。然后按照下列代码改变 `HelloHandler` 类（变化部分以黑体显示）：

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self):
```

```

self.render("/home/ben/user-info.html")

def post(self):
    self.render("hello-template.html", name = self.get_
        argument("name"))

```

最后一段代码也需要根据新变化作出相应修改：

```

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r"/hello/", HelloHandler),
    ])
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

现在运行代码，再用网络浏览器访问 <http://localhost:8888/hello/>。和前面的结果基本一致，所不同的是现在可以在文本框中输入你的名字。

这是因为当你第一次访问 <http://localhost:8888/hello/> 时，浏览器发送了一个 GET 请求，因此 Tornado 渲染了 `user-template.html`。当你单击 Sign In 按钮时，因为表单中包含 `/hello/` 动作和 `POST` 方法，浏览器将发送一个 `POST` 请求（如果动作中不包含服务器名字，浏览器将把请求发送给同一个服务器）。此时 Tornado 通过 `hello-template.html` 中的 `self.get_argument()` 捕捉用户输入数据。`name` 就是你在 `user-info.html` 的文本框中输入的名字。

和变量一样，你可以将一些 Python 代码段放到模板中。这些代码包围在 `{% 和 %}` 中。最常见的代码是：

```

{% set my_var = 0 %}
{% for x in y %} ... {% end %}

```

可以看出，它和正常 Python 有点不同。在声明变量之前需要加上 `set`，也不再使用缩进（在 HTML 中缩进没有意义），`{% end %}` 表示代码段结尾。

举个例子，`hello-template.html` 可以改成：

```

<!DOCTYPE html>
<head>
<title>Hello</title>
</head>
<body>
{% set char_number = 0 %}
{% for letter in name %}
{{ letter }} <br>
{% set char_number = char_number + 1 %}
{% end %}
{{ name }} has {{char_number}} characters
</body>

```

这虽然不是段很有用的代码，但是它能完成的任务比前面的模板多一点。它将换行显

示名字中的每个字母 (
 是 HTML 的换行标签), 并且还计算出名字中的字符数。

练习 2

创建一个新的网络应用, 用户可以输入城市名, 然后得到一个显示该城市未来 7 天天气预报的网页 (数据可以从 openweathermap.org 获取)。参考答案见本章结尾处。

7.4 安全

使用 Tornado 开发出越来越强大的网络应用的同时, 安全问题迟早会变得重要起来。即使你的程序仅在局域网中运行, 还是会有些不希望未授权的人看到的東西。本章的最后一个例子是提供树莓派相关信息的网络应用。

该网络应用在登录系统中使用 cookie。cookie 就是网络应用在浏览器中存储的一些信息。可以使用它来做很多事情, 这里我们使用它来追踪登录会话。用户成功登录后, 我们根据用户名为他们建立一个安全 cookie。cookie 会在退出登录时清除掉。当用户需要查看网络应用时, 程序会检查 cookie, 如果没有 cookie, 将会重定向到登录页面。程序代码打包存放在网站上的 `chapter7-tornado.zip` 中。

```
import tornado.ioloop
import tornado.web

users = {"ben": "mypassword"}

class SysStatusHandler(tornado.web.RequestHandler):
    def get(self):
        if not self.get_secure_cookie("user"):
            self.redirect("/login")
            return
        if self.get_argument("type") == "processes":
            com = ["pstree", ["top", "-bn1"]]
        elif self.get_argument("type") == "system":
            com = ["uname", "-a", ["uptime"]]
        elif self.get_argument("type") == "syslog":
            com = ["tail", "-n100", "/var/log/syslog"]
        elif self.get_argument("type") == "storage":
            com = ["df", "-h", ["free"]]
        elif self.get_argument("type") == "network":
            com = ["ifconfig"]
        else:
            com = ["df", "-h", ["free", ["uname", "-a", ["who",
                ["uptime", ["tail", "/var/log/syslog"], ["pstree",
                    ["top", "-bn1"]]]]]]]
        self.render("sysstatus-template.html", commands=com)
```

```

class LoginHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("login-template.html")

    def post(self):
        print(self.get_argument("name"))
        print(self.get_argument("password"))
        if self.get_argument("name") in users.keys() and users[self.
get_argument("name")] == self.get_argument("password"):
            self.set_secure_cookie("user", self.get_argument("name"))
            self.redirect("/sysstatus?type=system")
        else:
            self.render("login-fail.html")
class LogoutHandler(tornado.web.RequestHandler):
    def get(self):
        self.clear_cookie("user")
        self.render("logout-template.html")

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/login", LoginHandler),
        (r"/sysstatus", SysStatusHandler),
        (r"/logout", LogoutHandler),
    ], cookie_secret="put your own random text here")
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

可以看到，只用一个称为 `type` 的页面参数，`handler` 就可以完成绝大部分的工作（`SysStatusHandler`）。`type` 的值可以是 `processes`、`system`、`syslog`、`storage`、`network` 或者所有这些的总和。然后它发送一个模板中相应的各种 Linux 系统命令的列表。

HTTP GET 请求中，参数在 URL 的问号之后传入，如 `http://localhost:8888/sysstatus?type=network` 用 `network` 给参数 `type` 赋值并传递给 `sysstatus` 页面。

完成繁重工作的模板是 `sysstatus-template.html`：

```

<!DOCTYPE html>
<head><title>Raspberry Pi System Status Checker</title></head>
<body><h1>Raspberry Pi System Status Checker</h1>
<a href="/sysstatus?type=processes">Processes</a>
<a href="/sysstatus?type=system">System</a>
<a href="/sysstatus?type=syslog">System Log</a>
<a href="/sysstatus?type=storage">Storage</a>
<a href="/sysstatus?type=network">Network</a>
<a href="/sysstatus?type=all">Display all</a>
<a href="/logout">Logout</a>

{% import subprocess %}
{% for command in commands %}

```

```

<h2> Command: {{command[0]}} </h2>
<h2> Output: </h2>
<pre>
{% for line in subprocess.check_output(command).splitlines() %}
{{line}}
{% end %}
</pre>
{% end %}

</body>

```

`subprocess` 是一个 Python 模块，使用它可以在底层操作系统上运行命令。在树莓派中，操作系统是 Linux。它的输入是一个列表。列表中第一个条目是可执行命令，所有后面的条目都作为命令的参数。因此，列表 `["df", "-h"]` 将运行命令 `df-h`，该命令返回系统的当前磁盘，它的挂载信息和空余磁盘空间。你可以在 `LXTerminal` 中试一下。实际上，这里提到的所有命令你都可以在 `LXTerminal` 中试试。

`<pre>` 标签用来定义预格式化的文本。被包围在 `<pre>` 标签中的文本通常会保留空格和换行符而不是当做一行处理。

其他的模板都很简单。这个是 `login-fail.html`：

```

<!DOCTYPE html>
<html>
<head>
<title>Login Fail</title>
</head>
<body>
Your login has failed. Please click <a href="login">here</a>
to try again.
</body>

```

这个是 `login-template.html`：

```

<!DOCTYPE html>
<head>
<title>Raspberry Pi System Status Login Form</title>
</head>
<body>
<p>
Please enter your login information</p>
<form action="/login" method="post">
Username: <input type="text" name="name">
<br>
Password: <input type="password" name="password">
<br>
<input type="submit" value="Sign in">
</form>
</body>

```

最后，这个是 `logout-template.html`。

```
<!DOCTYPE html>
<html>
<head>
<title>Login Fail</title>
</head>
<body>
You have logged out. Please click <a href="login">here</a>
if you wish to log in again.
</body>
```

我们这里创建的登录方法提供了一定的安全性，但还远不够安全。如果你创建个处理敏感信息的网络应用，或者允许用户通过控制树莓派但又不信任所有的用户，你需要适当地研究下网络安全。这个话题太过庞大，我们在这里无法展开讨论，但你可以从开放的网站应用程序的项目安全（Open Web Application Security Project，参见 www.owasp.org）开始学习。

7.5 小结

读完本章，你应该能够更好地理解以下内容：

- 使用主机地址和端口号可以在两个计算机之间相互传递数据。
- 主机地址可以是 IP 地址，也可以是域名。
- 你需要使用套接字来连接到其他计算机并通过套接字来发送数据。
- 作为接口程序，API 可以访问网络服务器及其数据。
- 推特 API 提供了一个模块，使用它可以轻松获取和操作 `twitter.com` 上的信息。
- API 使用一种统一的数据格式 JSON，可以方便地在应用程序之间解析和共享数据。
- 网页由 HTML 编写，由 HTTP 对外提供服务。
- `http.server` 可以用来创建非常简单的网络服务器。
- 可以使用 Tornado 创建更复杂的网络应用。
- Tornado 模板使得在 HTML 中处理复杂、变化的信息变得更方便。
- 服务器可以使用 HTML 表单和 POST 请求接收信息。
- cookie 允许你在客户端浏览器存储一些信息，如用户名。

7.6 习题答案

练习 1

这是个将某个地方的天气预报推特出去的示例程序。

```

import twitter, urllib.request, json

twitter_user = twitter.Twitter(
    auth=twitter.OAuth("1824182228-
    lHOzvEpNA31LTiUCHkLSiqgW5Pbe7BvJKvKT2H6",
    "AwYIcpfRFUt6F4hMoGfqMGINDiUrW49R1mjVqY6Bts",
    "xStMTHb0HQZaEFLi2bsWA",
    "YgFhiCpvlqLLe5Is5dRAZWNzlT84KnyZCMKfXIwN8"))

city = input("Which city? ")
url = "http://api.openweathermap.org/data/2.5/weather?q="
url = url+city
req = urllib.request.Request(url)
forecast_string = urllib.request.urlopen(req).read()
forecast_dict = json.loads(forecast_string.decode("UTF-8"))

tweet_text = city + " weather is " +
    forecast_dict['weather'][0]['description']

twitter_user.statuses.update(status=tweet_text)

```

练习 2

有很多方法可以做到，我们使用三个文件。第一个是一个名为 city-info.html 的模板，包含以下内容：

```

<!DOCTYPE html>
<head>
<title>Weather</title>
</head>
<body>
<h1>Which City?</h1>
<form action="/weather/" method="post">
Enter a city: <input type="text" name="city">
<input type="submit" value="Submit">
</form>
</body>

```

第二个名为 weather-template 的模板包括如下内容：

```

<!DOCTYPE html>
<head>
<title>Weather</title>
</head>
<body>
{% set day_num = 0 %}
{% for day in forecast %}
<h2>Day : {{str(day_num)}} </h2>
<h3>{{day['weather'][0]['description']}} </h3>
Cloud Cover : {{str(day['clouds'])}} <br>
Temp Min : {{str(round(day['temp']['min']-273.15, 1))}}

```

```

degrees C<br>
Temp Max : {{ str(round(day['temp']['max']-273.15, 1)) }}
degrees C<br>
Humidity : {{str(day['humidity'])}} %<br>
Wind Speed : {{str(day['speed'])}}m/s<br>
{% set day_num = day_num + 1 %}
{% end %}
</body>

```

第三个 Python 文件包括:

```

import tornado.ioloop
import tornado.web
import subprocess
import urllib.request,json

class WeatherHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("/home/ben/city-info.html")

    def post(self):
        url = "http://api.openweathermap.org/data/2.5/forecast/"
            + "daily?cnt=7&units=meteric&mode=json&q="
            + self.get_argument("city")
        req = urllib.request.Request(url)
        response = urllib.request.urlopen(req)
        self.forecast = json.loads(response.read().decode("UTF-8"))
        self.render("weather-template.html",
            forec = self.forecast['list'])

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/weather/", WeatherHandler),
    ],)

    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

我的世界

给不了解的人介绍一下,《我的世界》(mine craft)是一个生存游戏,玩家可以在周围世界里采矿获取物资建造建筑物。游戏的世界由方块(或方格)组成,每个方块都可以被摧毁,摧毁后会露出隐藏的财富。图 8-1 给出了一个典型的我的世界的界面。

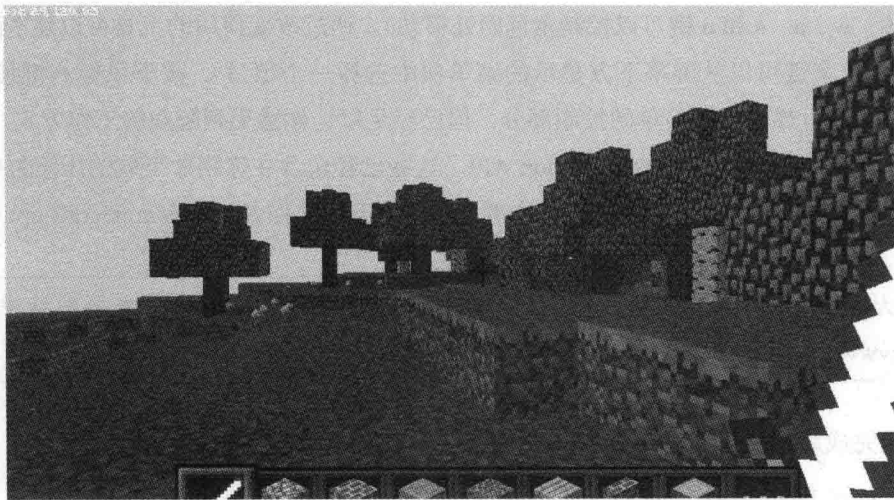


图 8-1 3D 方块成为这个游戏的标志

这是一个相当粗糙的描述,只是介绍个概念让你大致明白它是做什么的。它的完整版本可以运行在大多数计算机上,针对树莓派它也有个特殊版本。树莓派版本的特殊之处在

于你可以用 Python 控制游戏。你不仅可以操作角色本身，还可以操作你身边的整个世界。

8.1 畅游我的世界

这些介绍已经足够了。有些事物必须要亲自看一下才能理解，我的世界就是其中一个。首先，你需要从网上下载软件 <http://pi.minecraft.net>。

要下载的最后一个文件是 `minecraft-pi-0.1.1.tar.gz`。使用下面的命令行在终端中解压缩文件：

```
tar zxvf minecraft-pi-0.1.1.tar.gz
```

现在可以使用下面的命令打开游戏：

```
mcpi/minecraft-pi
```



注意 在玩我的世界时，它会接管鼠标和键盘的控制。如果你想切换回其他非我的世界窗口，只需按 `Alt+Tab` 组合键，游戏就会释放对鼠标和键盘的控制。

8.1.1 控制我的世界

首先要做的是开始新游戏，并尝试在世界中移动（使用鼠标改变朝向，鼠标左键可以选择物体，`w`、`a`、`s` 和 `d` 键可以控制角色四处移动）。通过改变使用的工具可以建造建筑物（按下某个数字键可以从屏幕下方显示的菜单项中选择一个项目），按下鼠标右键使用它。这些只是我的世界中非常简单的控制部分，但已经足以让你感受到游戏的工作方式。

为了控制世界，你需要使用 Python API。这些已经包含在你刚才下载的压缩包中，但这个版本不支持 Python3。因此你需要从本书网站中下载文件 `chapter8-minecraft-api.tar.gz`。



注意 大段代码的例子和其他 API 以及本书中提到的下载都是指从本书网站中下载：
www.wiley.com/go/python-raspberrypi。

下载完成后，使用下面的命令行解压缩：

```
tar zxvf chapter8-minecraft.tar.gz
```

使用 `cd` 命令切换当前目录进入新文件夹。所有本章要创建的程序都应该放在这个文件夹中，这样它们才能使用 Python API：

```
cd mcpi
```

你要写的程序不会运行一个新的我的世界，而是连接到当前在树莓派上运行的我的世界的一个进程上。

认识到这点之后，打开一个新的 Python 解释器。请记住必须转到文件夹 `chapter8-minecraft` 下面。最简单的办法就是打开 LXTerminal 会话，转到 `chapter8-minecraft` 文件夹下面（使用前面的 `cd` 命令），然后执行 `python3`。

下列代码将连接到当前运行的我的世界的一个进程：

```
>>> import minecraft
>>> mc = minecraft.Minecraft.create()
```

现在可以操作一个称为 `mc` 的我的世界对象。这个对象有一个可以移动的 `player` 对象，试试这个命令：

```
>>> mc.player.setPos(10,10,10)
```

你会发现角色移动了。角色移动到坐标 `(10,10,10)` 处，但由于世界是随机生成的，这个点可能在空中，这种情况下，角色将掉落到地板上。或者，他也可能出现在水底或地下。改变坐标将其移动到地面上的某点。坐标中第一个和最后一个数字是角色在地面上的位置（`x` 和 `z`），中间的值（`y`）表示垂直高度。

8.1.2 用 Python 创建我的世界

选好角色位置之后，你可以使用 `setBlock(x,y,z,type)` 方法改变角色周围的世界。`x`、`y`、和 `z` 表示角色周围的世界坐标，和角色坐标的格式一样。`type` 表示方块的 `typeID`。每个不同类型的方块都有一个 `typeID`，你需要使用它们来操纵世界。

例如，当角色在 `(10,15,22)` 这个位置时，你可以用下列命令在他边上建造一个蘑菇：

```
>>> mc.setBlock(11,15,22, 40)
```

表 8-1 给出了一个有用的方块类型列表

表 8-1 一些有用的类型 ID

ID	方块类型	ID	方块类型
0	空气	1	石头
2	草地	3	泥土
4	鹅卵石	5	厚木板
6	树苗	8	(流动的) 水
10	(流动的) 岩浆	18	树叶
22	天青石	40	蘑菇
41	金块	246	发光的黑曜石

构建世界

至此，你已经能够在自己的“我的世界”中工作了。然而，如果希望创建一个可重用的程序，你需要所有东西的起始位置。为了做到这一点，你需要按自己的想法创建世界。下列脚本将世界中心的一个 200×200 方形区域设置成光滑的草地（类型 2），草地上方是空气（类型 0）。

```
import minecraft
mc = minecraft.Minecraft.create()
mc.setBlocks(-100,-1, -100, 100, 0, 100, 2)
mc.setBlocks(-100, 1, -100, 100, 100, 100, 0)
```

代码中使用了 `setBlocks(x1,y1,z1,x2,y2,z2,type)` 方法，它将 `(x1,y1,z1)` 和 `(x2,y2,z2)` 之间的所有方块设置成相应的类型。

绘制画面

现在你已经知道如何设置方块，让我们开始用它来做一些有意义的事情。在完整版的我的世界中，你可以用它来建筑一个超级堡垒作为角色的避难所。然而，树莓派上运行的并不是完整版本。你可以使用这个版本来创作些美术作品，如使用方块画个图。下列代码就是如此（参见网站上的 `chapter8-art.py`）：

```
import minecraft
import sys

mc = minecraft.Minecraft.create()

height = 10
start_x = 0
start_y = 0
canvas = "horizontal"

r = 246
g = 18
b = 22

picture = [
    ["R", "R", "R"],
    ["G", "G", "G"],
    ["B", "B", "B"]
]

if len(sys.argv) > 1:
    with open(sys.argv[1]) as f:
        picture = f.readlines()

pic_x = start_x
for line in picture:
```

```

pic_y = start_y
for block in line:

    if canvas == "horizontal":
        x = pic_y
        y = height
        z = pic_x
    else:
        x = height
        y = pic_y
        z = pic_x
    if block == "R":
        mc.setBlock(x,y,z,r)
    if block == "G":
        mc.setBlock(x,y,z,g)
    if block == "B":
        mc.setBlock(x,y,z,b)

    pic_y = pic_y + 1
    pic_x = pic_x + 1

```

这和第4章中的加载关卡工作方式类似。代码使用 `sys.argv` 来检查用户是否在命令行中指定了文件名。如果指定了，它将使用该文件来绘制图片，否则，使用默认值，画出三条线。

有两个 `for` 循环来遍历每一行的每一个字符。循环中如果遇到 R、G 或 B，就相应地画出一个红色、绿色或者蓝色方块。

然而，你在世界中没有完全的控制权，只能创建一些预定的方块类型，还有并不是所有的方块都可以漂浮在空中。本例中使用型号 246 的方块（发光的黑曜石）表示红色，型号 18（树叶）表示绿色，型号 22（天青石）表示蓝色。当然，你也可以将其换成任何你想要的方块创造出自己的风格。

代码中的坐标有点复杂，因为它有两个模式——垂直和水平——表示了不同绘图方向。正因为如此，他需要处理两组坐标：关于图片的 2D 坐标和关于我的时间的 3D 坐标。`if` 语句根据指定的方向在这两者之间作出选择。

注意，程序将忽略非 R、G 或 B 的所有字符，因此，你可以在图片中包含这些字符，程序将使用空格表示它们。例如，下列文字将创建一个笑脸图标（见图 8-2）：

```

-----
-G---G-
-G---G-
---B---
-R---R-
--RRR--

```

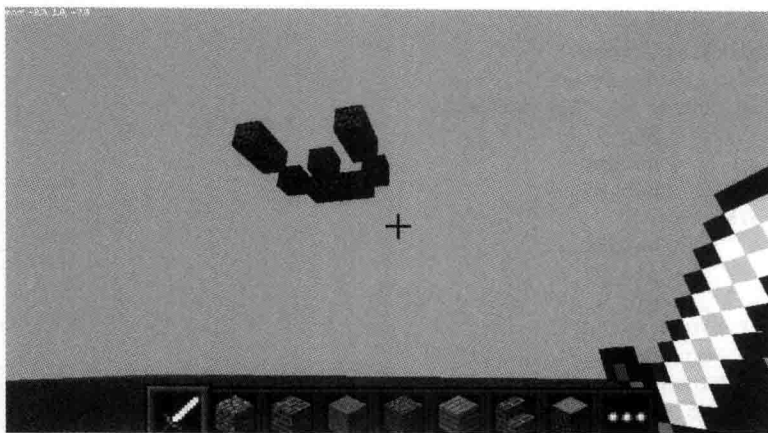


图 8-2 笑脸只是一个开始，你可以使用这个方法在我的世界中画出任意你想要的图形

8.1.3 深入探索

还有很多方法来扩展这个世界。最简单的方法就是将颜色托盘从 3 个扩展到和可供选择的角色和方块类型一样多。稍微复杂的方法是创建一个 3D 图形。这需要你适当地改变下文件结构。我们推荐使用多个 2D 图形组在一起，使用一个特殊行将它们分开。例如，下面这个两层厚的笑脸：

```

-----
-G---G-
-G-B-G-
---B---
-R---R-
--RRR--
*
-----
-----
-G---G-
---B---
-R---R-
--RRR--

```

还可以使用相同的文件结构构建动画，每个组的方块代表一帧。

8.2 制作贪吃蛇游戏

3D 图形的一个主要应用就是制作游戏。我的世界本身就是游戏，你也可以使用它作为游戏引擎构建更多的游戏。在本节中，你将会看到如何制作经典游戏——贪吃蛇。


```

["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "R", "R", "R", "R", "R", "R", "R", "R"]

if len(sys.argv) > 1:
    with open(sys.argv[1]) as f:
        picture = f.readlines()

posn_z = start_z
for line in picture:
    posn_x = start_x
    for block in line:
        x = height
        y = posn_x
        z = posn_z

        if block == "R":
            mc.setBlock(x, y, z, 246)

        posn_x = posn_x + 1
        posn_z = posn_z + 1

snake = [(int((posn_z - start_z)/2), int((posn_x - start_x)/2))]
movement = [-1, 0]

finished = False

grow_in = []
apple_in = apple_freq

while not finished:
    for event in pygame.event.get():
        if event.type == QUIT:
            finished = True

    key_state = pygame.key.get_pressed()
    if key_state[K_LEFT]:
        movement = [-1, 0]
    if key_state[K_RIGHT]:
        movement = [1, 0]
    if key_state[K_UP]:
        movement = [0, 1]
    if key_state[K_DOWN]:

```

```

movement = [0,-1]

next_position = (height, snake[0][1] + movement[1],
                 snake[0][0] + movement[0])
next_block = mc.getBlock(next_position[0],
                          next_position[1], next_position[2])

if next_block == 246 or next_block == 22:
    finished = True

if next_block == 18:
    grow_in.append(len(snake))
    apple_in = apple_freq

snake.insert(0, (next_position[2], next_position[1]))
for block in snake:
    mc.setBlock(height, block[1], block[0], 22)

if 0 not in grow_in:
    remove_block = snake.pop()
    mc.setBlock(height, remove_block[1], remove_block[0], 0)

grow_in2 = []
for number in grow_in:
    if number > -1:
        grow_in2.append(number - 1)
grow_in = grow_in2

if apple_in == 0:
    apple_y = random.randint(1,int(posn_x - start_x)-1)
    apple_x = random.randint(1,int(posn_z - start_z)-1)
    while mc.getBlock(height, apple_y, apple_x) != 0:
        apple_x = random.randint(1,int(posn_z - start_z)-1)
        apple_y = random.randint(1,int(posn_x - start_x)-1)
    mc.setBlock(height, apple_y, apple_x, 18)

apple_in = apple_in - 1

time.sleep(difficulty)

mc.postToChat("Game Over")

```

这里有两个之前没有用过的 Minecraft 方法：getBlock() 和 PostToChat()。这两个方法都很好地自注释了本身。第一个返回指定位置的方块类型，第二个在玩家屏幕上显示字符。

你应该能认出第一部分的代码，因为它和之前的那个画图代码几乎一模一样。因为这里只需要一个颜色来表示限制贪吃蛇移动的迷宫，我们做了一些简化（我们选择了红色，你可以替换成任何一种你喜欢的颜色）。

代码的第二部分（包括 while 循环和它上面的 5 行）包含了游戏的运行机制。贪吃蛇

存储在一个元组的列表中，每个元组包含了一节贪吃蛇的 2D 坐标。这些一起组成了 3D 坐标并显示到屏幕上。

8.2.1 移动贪吃蛇

移动贪吃蛇分为两部分。首先，在开始位置增加一个新的方块。贪吃蛇的最前方的方块坐标加上移动变量就是新方块的位置。其次，使用 `pop()` 将贪吃蛇的最后一个方块从列表中移除。`pop()` 方法将方块从列表中去除并将该方块作为返回值返回。这个坐标的方块将被设置为 0（空气）。

关键的控制部分和第四章中的一样。使用了 `PyGame`。唯一的细微差别就是为了让它工作，就必须有一个 `PyGame` 窗口，并且为了捕获按键动作，键盘焦点必须在这个窗口中。程序运行时，你会看到有个 200×200 的空窗口，并且要想控制贪吃蛇就必须单击该窗口。

8.2.2 增长贪吃蛇

可能代码中最复杂的部分就是当贪吃蛇吞下苹果时如何增长它。这是因为贪吃蛇吃下苹果之后并不是立即变长的。而是当它的整个身体都通过苹果之后才增长。

为了控制这个过程，这里使用了一个叫做 `grow_in` 的列表。每当贪吃蛇吃下一颗苹果，他的长度就会添加到这个列表中。主循环每运行一次，将列表中的每个数都减一，负数丢弃。这就意味着，当整个贪吃蛇通过苹果之后，列表中只剩下 0。列表里有非零数字时，贪吃蛇的末端部分被移除。实现部分参见下面一段代码：

```
if 0 not in grow_in:
    remove_block = snake.pop()
    mc.setBlock(height, remove_block[1], remove_block[0], 0)

    grow_in2 = []
    for number in grow_in:
        if number > -1:
            grow_in2.append(number - 1)
    grow_in = grow_in2
```

8.2.3 添加苹果

剩下要做的就是添加苹果。贪吃蛇吞下前一个苹果之后经过 `apple_freq` 次循环，新苹果会立即出现。苹果出现的位置是随机的，参见下面的代码：

```
if apple_in == 0:
    apple_y = random.randint(1, int(posn_x - start_x) - 1)
    apple_x = random.randint(1, int(posn_z - start_z) - 1)
```

```

while mc.getBlock(height, apple_y, apple_x) != 0:
    apple_x = random.randint(1,int(posn_z - start_z)-1)
    apple_y = random.randint(1,int(posn_x - start_x)-1)
mc.setBlock(height, apple_y, apple_x, 18)

```

这段代码先产生一个随机位置，然后检查下该位置是否已经被占用（如果被占用，产生一个新的随机位置）。apple_in 变量用来统计上个苹果被吃掉之后的循环次数。

从图 8-3 的游戏截图中可以看到游戏的效果，也可以自己运行一下看看。

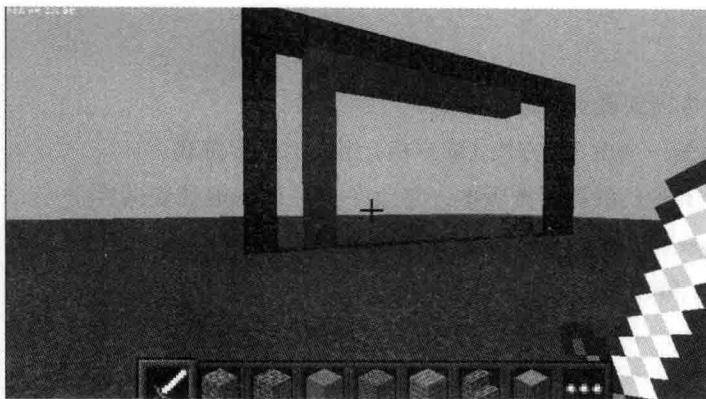


图 8-3 通过改变地图可以改变游戏的难程序。只要注意不要在围墙上留缝隙以防贪吃蛇逃走，否则将无法在我的世界中继续控制贪吃蛇

8.3 深入探索

和第 4 章的游戏相比，这个游戏还没有结束，我们将剩下的工作留给你。有很多进一步完善的方法，包括：

- 增加更多的图形。例如，贪吃蛇挂掉时候添加一个爆炸效果，或者在开始的时候增加一个倒计时。还可以将贪吃蛇图形变得更引人注目点，比如随着长度的增加添加一些其他方块。
- 计分。可以是贪吃蛇的长度。
- 改变难度系数（在循环中使用 `time.sleep()`）。这个可以和分数联系起来。贪吃蛇越长，游戏速度可以越快。
- 惊喜。例如，一旦达到一定长度，贪吃蛇会改变颜色。
- 奖励。短暂地在地图上出现，如果贪吃蛇吃到了，玩家会得到加分。
- 更多关卡。可以设计不同尺寸的地图，或者在地图内部添加一些额外的墙增加游戏难度。

不用把自己限定在完善贪吃蛇上。可以使用这些方法创作新游戏。例如，相同的游戏机制可以用在电子世界争霸赛（Tron）中（改编自同名电影。使用 `apt-get xtron` 安装一下试试看）。

甚至还可以把它像第 4 章中的一样改造成平台，或者其他任何你想要的。唯一束缚你的就是你的想象力。

8.4 小结

读完本章，你需要能更好地理解以下内容：

- 我的世界是一个由 3D 方块（或方格）组成的生存游戏。
- 我的世界有专门的树莓派版本，它允许使用 Python 控制世界。
- 使用 `player.setPos(x,y,z)` 可以移动角色。
- `setBlock()` 和 `SetBlocks()` 可以用来建设世界。

多媒体

我们在前 8 章中，大量介绍了处理数据并把它从树莓派中输出的方法。然而到现在为止还没有介绍过多少向树莓派输入数据的方法。前面提到过的一些按下按钮，以及与鼠标相关的操作指的就是这部分。在本章中，我们将会扭转这种情况。给你的树莓派加上两个附加传感器——话筒和摄像头，让它以一种更自然的方式与外界交互。我们将会分别介绍它们，并告诉你在 Python 程序中如何使用它们。

9.1 使用 PyAudio 让计算机发声

一旦将声音导入计算机，你就可以做各种各样的事情，但这些都要从同一件事情开始——输入声音。要想快速、简单地将声音输入计算机，PyAudio 模块是所有工具中最棒的一个。录好声音之后，你就可以任意操作它，但是如果要在其他程序中使用它，就需要将其转换成其他软件可以识别的格式。WAV 是一种简单无损格式，非常适合简单的音频数据（如果记录了大量数据，WAV 格式的文件会非常巨大）。

另外，你还需要一个 USB 话筒用来收集声音。小巧的树莓派只有声音输出接口。

和平时一样，首先要做的事情就是使用下列命令安装模块：

```
sudo apt-get install libportaudio0 libportaudio2 libportaudiocpp0  
  \portaudio19-dev python3-setuptools python3-pip  
sudo pip-3.2 install pyaudio
```

现在让我们来录音！代码如下（参见网站文件 chapter9-record.py）：

```

import pyaudio
import wave

def record_sound(seconds, chunk_size, sample_rate, filename,
                  channels, format_type):
    p = pyaudio.PyAudio()

    stream = p.open(format=format_type,
                     channels=channels,
                     rate=sample_rate,
                     input=True,
                     frames_per_buffer=chunk_size)

    print("Speak now")

    frames = []

    for i in range(0, int(sample_rate / chunk_size * seconds)):
        data = stream.read(chunk_size)
        frames.append(data)
        if i%int(sample_rate/chunk_size) == 0:
            print(seconds - round(i/(sample_rate/chunk_size)),
                  "seconds remaining")

    print("Finished")

    stream.stop_stream()
    stream.close()
    p.terminate()

    wf = wave.open(filename, 'wb')
    wf.setnchannels(channels)
    wf.setsampwidth(p.get_sample_size(format_type))
    wf.setframerate(sample_rate)
    wf.writeframes(b''.join(frames))
    wf.close()

chunk_size = 1024
sample_rate = 44100
seconds = 15
filename = "output.wav"
channels = 1
format_type = pyaudio.paInt16
record_sound(seconds, chunk_size, sample_rate, filename, channels,
              format_type)

```

看得出来，有很多属性会影响录制，你可以通过修改它们来改变录制过程。它们中有些是用来控制数据如何采集和保存的（例如，`chunk_size` 和 `sample_rate`），其他的是一些高级属性（如录音的时长，使用的音轨数量）。目前，我们就固定使用这些默认值，除非你很有把握再调整它们。有些属性受硬件限制，试图改变它们会引起 PyAudio 异常。

9.1.1 录音

`record_sound()` 函数完成了所有繁重的工作（把它作为内联代码而不是函数也可以正常工作，这里我们将其定义成函数是为了方便在其他程序中使用）。第一行和第二行创建了一个新的 `PyAudio` 对象，然后使用它来打开流。我们就是从这个流中获取的声音数据。

录音的主要工作由下面的 `for` 循环完成：

```
for i in range(0, int(sample_rate / chunk_size * seconds)):
    data = stream.read(chunk_size)
    frames.append(data)
```

这段代码把数据保存在给定大小的 `chunk` 中。`chunk` 的数据类型是 `byte`，但你不用关心这些。

现在已经得到了以字节串形式保存的声音数据，还需要再做些处理。最简单的操作就是使用 `wave` 模块将其保存为 `WAV` 文件，这也是本程序后半部分所做的工作。

运行程序后会看到一个错误提示。不用担心，这只是 `PyAudio` 的工作方式，用来告诉你程序运行一切正常。（当 `PyAudio` 在设备上检测到音频设备时会抛出这个提示。只要不属于任何形式的 `Python` 异常，一切都会正常运行）。

之后，会在程序运行的目录下得到一个名为 `output.wav` 的文件。这是一个 15 秒长的录音，可以使用任何音频软件回放。

9.1.2 向树莓派讲话

你使用自己录下的声音做任何操作，其中最酷的，不需要任何音乐天分的操作就是语音控制。要做到这一点，你需要找到一种能够从声音中获取谈话内容的方法，这绝对是个有难度的工作。

不幸的是，任何一个模块都无法轻松地完成这项工作。然而，正如你在第 7 章中见到的那样，`Python` 模块并不是在程序中实现附加功能的唯一方法。网络服务是另外一个很好的特性来源，并且可以轻松地加入程序中。

这里使用了 `Google` 提供的一个非常出色的语音转文字的网络应用。不过它还存在一些限制。首先，每段语音长度不能超过 15 秒。其次，音频片段必须以采样率 16 000 的 `FLAC` 格式上传。

对于语音控制系统来说第一个限制不会引起什么问题（听写程序除外）。然而第二个限制有点麻烦，因为 `Python` 还没有标准 `FLAC` 模块。虽然有一些选项，但没有一个能够很好地满足我们的需求。这里我们可以转过来寻找应该一个附加功能的来源（将在下

一章中详细介绍)——Linux 命令行。你可以在 Python 代码中运行并控制普通的 Linux 程序。

实现语音转换成文字(前提是使用 FLAC 格式,格式转换稍后介绍)的代码如下所示:

```
def google_speech_recognition(filename):
    global url

    audio = open(filename, 'rb').read()
    http_header = {'Content-Type': 'audio/x-flac; rate=16000'}
    request = urllib.request.Request(url, data=audio,
                                     headers=http_header)

    response = urllib.request.urlopen(request)
    out = response.read()
    json_data = json.loads(out.decode("utf-8"))

    return json_data
```

读过第 7 章,就会非常熟悉这段代码,因此我们不再详细介绍。这里唯一的新事物就是调用 `urllib.request.Request()` 时的附加参数。参数 `data`,非常明显,是用来传送数据的,本例中就是音频文件。参数 `headers` 用来发送 HTTP 头。`headers` 告诉网络服务谁在请求数据,以及它们在发送什么数据。在这个例子中,我们告诉服务器,在向他发送采样率为 16 000 的 FLAC 格式的音频。它不会直接返回文字,而是返回一个 JSON 编码的数据,其中包括文字和文本正确性的证据(如果语音中存在不同的口音和部分俚语,并不能保证结果完全正确)。如果想知道更多返回内容,可以向下面一样添加一行打印:

```
print(json_data)
```

在第 7 章中我们详细介绍了该结构体。

处理文字比处理语音要简单得多,因此现在可以将这些内容整合到软件中。不过这个例子还没有完成,我们还要进一步扩展它——向它提问,如果幸运,它可以回答你。

9.1.3 向程序提问

给程序添加回答问题的特性很有挑战性,并且编程实现也是一个艰巨的任务。幸运的是我们可以从其他领域得到一点帮助。Wolfram Alpha 是一个试图完成这项工作的网络服务。可以使用网络浏览器访问 www.wolframalpha.com 来体验一下。也许你已经猜到了,它也提供了给应用程序使用的 API。

同样的,如果读过第 7 章,就会很熟悉下面的代码:

```
def wolfram_alpha(speech):
```

```

url_section = urllib.parse.urlencode(dict(
    input=speech,
    appid=wolfram_alpha_app_id,
))

url = http://api.wolframalpha.com/v2/query?' + url_section
response = urllib.request.urlopen(url)

tree = ElementTree.parse(response)
root = tree.getroot()

for node in root.findall('.//pod'):
    print(node.attrib['title'])
    for text_node in node.findall('.//plaintext'):
        if text_node.text:
            print(text_node.text)
    print("*****")

```

你可能不理解后半部分。Wolfram Alpha 的返回结果是 XML 文件。代码中使用 xml 模块中的 ElementTree 解析文件并取出相应信息。由于和本章内容不相关，我们在此不作详细介绍。如果以后你需要了解相关信息，可以在 xml 模块的文档中找到更多信息。



注意 运行这段程序之前，需要先获得一个 Wolfram Alpha 应用 ID。为此，你需要到 <https://developer.wolframalpha.com/portal/signup.html> 注册账号。获得账号之后将其复制到程序中的适当位置。

9.1.4 组合包装

现在可以将这三个功能（录音、转换成文字、回答提问）合并到一起创建一个语音识别程序（我们称为 Pyri，类似于一个基于移动平台的知名应用）。下面给出了代码（参见网站上的 chapter9-pyri.py）。前面讨论过的函数这里不再重复，但是你需要在程序中包含它们。



注意 本书中出现的大段程序都可以在本书网站上下载 www.wiley.com/go/pythonraspi。为了避免不必要的输入错误，你可以从网站上下载代码，复制和粘贴到你的 IDE 或代码编辑器中。下面例子的代码参见 chapter9-pyri.py。

```

import urllib.request
import json
import pyaudio

```

```

import wave
from xml.etree import ElementTree
import subprocess
import time

def record_sound(seconds, chunk_size, sample_rate, filename,
                 channels, format_type):
    #Code give above

def google_speech_recognition(filename):
    #Code given above
def wolfram_alpha(speech):
    #Code give above

chunk_size = 512
sample_rate = 44100
seconds = 5
filename = "output.wav"
channels = 1
format_type = pyaudio.paInt16

url = "https://www.google.com/speech-api/v1/recognize?"
    + "client=chromium&lang=en-US"

wolfram_alpha_app_id = "PUT-YOUR-APPID-HERE"

record_sound(seconds, chunk_size, sample_rate,
             filename, channels, format_type)

subprocess.call(["sox", "output.wav", "-r16k", "-t",
                "wav", "output-16k.wav"])
subprocess.call(["flac", "-5", "-f",
                "output-16k.wav"], stdout=subprocess.PIPE )

time.sleep(2)

try:
    google_data = google_speech_recognition("output-16k.flac")
except urllib.error.HTTPError:
    print("voice recognition failure")
else:
    print(google_data)

if google_data['status'] == 0:
    wolfram_alpha(google_data['hypotheses'][0]['utterance'])
else:
    print("Voice recognition failed. Try again")

```

由于最后的一个 try 语句包含一个 else 子句，你可能不太熟悉。它表示只要 try 语句中的代码不抛出异常，就会执行 else 这个代码段。本例中就意味着只要 Google 请求不抛

出 HTTPError 异常。

运行程序前，需要先确认已经安装好下面这两个模块：

```
sudo apt-get install sox flac
```

然后，有两个系统调用可以将文件转化成合适的格式：

```
subprocess.call(["sox", "output.wav", "-r16k", "-t",  
                "wav", "output-16k.wav"])  
subprocess.call(["flac", "-5", "-f",  
                "output-16k.wav"], stdout=subprocess.PIPE)
```

sox 也被认为是“音频界的瑞士军刀”。它有很多处理音频文件的特性。本例中，它被用来将文件采样率由 44 100 转换为 16 000。由 -r16k 这个参数控制。第二行使用 flac 程序将 WAV 文件转换位 FLAC。

我们将在下一章详细介绍 subprocess.call()，现在你只需要知道它可以让你在 Python 中运行 Linux 命令。

9.1.5 深入探索

如果希望把这个例子继续做下去，可以把它作为一个特性添加你在到第 4 章创建的网络浏览器中。不再只是输出文字，而是用来控制浏览器访问哪个页面。

你也可以做一个简单的声音驱动菜单，这样就可以不用在程序中使用鼠标和键盘。例如，可以显示一组带数字的选项，用户只需要说出自己选择的数字即可选中（有点类似于电话交换台）。

9.2 制作电影

现在你应该知道如何在树莓派上处理音频了，因此我们将进到第二个阶段——视频。

树莓派上可以支持两种摄像头类型——USB 网络摄像头和树莓派基金会的摄像头模块。由于它们的应用不同，这里我们将分别介绍。

9.2.1 使用 USB 网络摄像头

两者之中，Python 对 USB 网络摄像头更好点。在所有的计算机中它的工作方式都一样，因此我们首先介绍它。

说来奇怪，提取图像最简单的方法是在第 5 章中使用的 PyGame 模块。之前，你使用它构建游戏，但是模块包含有图像处理功能（游戏中经常要用到），其中包括从网络摄像头

中提取图像。

如果跟着第 5 章练习过，就应该安装过 PyGame。如果没有装过，现在先要按照下列方法安装模块：

```
sudo apt-get update
sudo apt-get install libsdl-dev libsdl-image1.2-dev \
    libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev \
    libportmidi-dev libavformat-dev libswscale-dev \
    mercurial python3-dev
```

然后获取最新版本的 Pygame：

```
hg clone <span cssStyle="font-family:monospace"><span
    cssStyle="font-family:monospace">https://bitbucket.org/pygame/
    pygame</span></span>
cd pygame
```

这两行将最新版本的 PyGame 下载到 pygame 目录中并将当前目录转到该目录。然后，就可以使用下面命令为 Python3 安装该模块：

```
python3 setup.py build
sudo python3 setup.py install
```

最简单的例子就是从摄像头中获取图像流并将其显示在屏幕上。实际上，它可以把你的树莓派变成一面镜子。下面给出了例子（代码参见网站文件 chapter9-mirror.py）。

```
import pygame
import pygame.camera

pygame.init()
pygame.camera.init()

display = pygame.display.set_mode((640,480),0)
cam = pygame.camera.Camera("/dev/video0", (640,480))
cam.start()

while True:
    image = cam.get_image()
    display.blit(image, (0,0))
    pygame.display.flip()
```

不仅要初始化 PyGame，你还需要初始化模块的摄像头部分。如果你只连接了一个摄像头，它在系统里就是 /dev/video0。如果连接了不止一个摄像头，就需要自己修改这个值。

然后使用模块，就是简单地调用 camera 对象的 get_image() 方法。它会返回一幅图像，你可以像处理 PyGame 图像一样随意使用该图像。本例中，我们只是将其画到主显示器上，然后调用 flip() 更新显示（在第 5 章中介绍过）。

得到图像后，不必立即将其直接显示到屏幕上。可以对它做一些操作。例如，可以使用一些图像效果，然后将其保存到文件中而不是显示出来。

看看下面的代码（参见网站文件 `chapter9-mirror-lines.py`）：

```
import pygame
import pygame.camera

pygame.init()
pygame.camera.init()
from pygame.locals import *

size = (320, 240)

display = pygame.display.set_mode(size, 0)

cam = pygame.camera.Camera("/dev/video0", size)
cam.start()

inverse = pygame.Surface(size, pygame.SRCALPHA)
for i in range (1,50):
    image = cam.get_image()
    lines = pygame.transform.laplacian(image)
    inverse.fill((255,255,255,255))
    inverse.blit(lines, (0,0), None, BLEND_RGB_SUB)
    display.blit(inverse, (0,0))
    pygame.display.flip()

pygame.image.save(inverse, "/home/pi/test.png")
```

这段代码和前面例子的基本结构相同，只是多了一些操作。首先，创建了一个新图像 `lines`。它由原始图像经过拉普拉斯（Laplacian）变换而成，即一种数学处理，可以获得图像中的边缘。返回图像将是一幅由黑色背景和表示图像边缘的白线构成的图像。

接着程序将图像转换成白色背景黑色线条。这不仅需要将图像颜色转换一下，为了做到这一点，你需要创建一幅白色图像，然后从中减去图像（即使用特殊标记 `BLEND_RGB_SU` 调用 `blit()`）。

你可能已经注意到，这段代码没有 `while` 循环，而是使用了 `for` 循环。该循环将执行 50 次然后退出。这将会生成一个几秒钟的视频（如果想要精确时长，可以在程序中加入第 4 章中使用的定时器函数）。循环结束后，将会执行最后一行，将当前转换过的边缘图像保存为 PNG 文件。这让程序看起来像一个定时摄像机。

本程序和上个例子的最后一个区别就是这次使用的图像比上次小。图像尺寸在变量 `size` 中设置。这是因为图像变换非常消耗计算资源，将图像宽高分别缩小一半可以将计算量减少到原来的四分之一（计算量和区域面积成比例）。从而使得视频看起来更连续。试

着改变尺寸，对比下性能变化。

9.2.2 使用 OpenCV 添加计算机图像特性

PyGame 可以很好地为计算机获取图像，但用来处理图像却并不合适。另一个叫 OpenCV 的模块，是专门为处理图像设计的。不幸的是在写作本书时它还不支持 Python3，这种情况在未来也许会改变。由于没有这个模块你就无法实现一些非常有意思的特性，我们将使用 Python2 来介绍这部分内容。但愿支持 Python3 的版本快点发布，这样你就可以在新版本的 Python 中使用它。

首先，从命令行中安装它：

```
sudo apt-get install python-opencv libopencv-core-dev
```

第一个例子和使用 PyGame 的第一个例子相似：一个简单的摄像头查看器。它将让你感受到 OpenCV 库是如何工作的：

```
import cv2

capture = cv2.VideoCapture(0)

while True:
    return_val, frame = capture.read()
    cv2.imshow('Face', frame)

    key = cv2.waitKey(5)
    if key == 113:
        break
```

和使用 PyGame 大致相似。打开摄像头，然后读取一帧并显示出来。最后三行监听用户按下 q 键的动作（q 键对应的键是 113），如果按下就退出。

OpenCV 是一个异常强大的用来处理图像的模块。一个最有趣的特性是目标识别。你的程序可以辨识图像的特定部分，例如眼睛或嘴。

这部分由保存在 XML 文件中的 Haar cascades 完成。它由训练图像集生成，计算机使用训练图像集找出相似的目标。你可以使用运行在 Linux 命令行下的 `opencv_traincascades` 程序训练自己的 Haar cascades。然而，你需要大量的图像（为了获得较好结果一般需要上百张），包括含有和不含有你希望识别物体的图像。关于如何做到，这里有个不错的介绍 http://docs.opencv.org/doc/user_guide/ug_traincascade.html。

我们不会去创建 cascades，OpenCV 在安装时给我们提供了一些例子。下面的例子就是用 OpenCV bundles 来识别眼睛和嘴。代码参见网站文件 `chapter9-objectdetect.py`。

```

import cv2

factor_down = 0.33
factor_up = 3

capture = cv2.VideoCapture(0)
eye_data = cv2.CascadeClassifier('/usr/share/opencv/haarcascades/'
                                + 'haarcascade_eye.xml')
smile_data = cv2.CascadeClassifier('/usr/share/opencv/
    haarcascades/' +
    'haarcascade_mcs_mouth.xml')

while True:

    return_val, frame = capture.read()
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    small_gray_frame = cv2.resize(gray_frame, (0,0),
                                   fx=factor_down, fy=factor_down)

    eyes = eye_data.detectMultiScale(small_gray_frame, 1.3, 5)
    for (eye_x, eye_y, eye_width, eye_height) in eyes:
        cv2.rectangle(frame, (eye_x*factor_up, eye_y*factor_up),
                       (eye_x*factor_up + eye_width*factor_up,
                        eye_y*factor_up+eye_height*factor_up),
                       (255,0,0),2)

    smiles = smile_data.detectMultiScale(small_gray_frame, 1.3, 5)
    for (smile_x, smile_y, smile_width, smile_height) in smiles:
        cv2.rectangle(frame, (smile_x*factor_up,
                               smile_y*factor_up),
                       (smile_x*factor_up + smile_width*factor_up,
                        smile_y*factor_up+smile_height*factor_up),
                       (0,255,0),2)
    cv2.imshow('Face', frame)

    key = cv2.waitKey(5)
    if key == 113:
        break

```

可以看出，加载 Haar cascades 非常简单，只需要使用一个参数来调用 `cv2.CascadeClassifier()`。它通过使用 `detectMultiScale()` 方法创建一个对象，可以使用该对象返回满足 cascade 条件的矩形区域列表。不过这里还有一些其他事情。

在 while 循环中，和上一个 OpenCV 例子一样，程序首先获取一帧图像，然后创建出两幅新图像——`gray_frame` 和 `small_gray_frame`。不要惊讶，它们分别是原始图像的黑白图像和一个缩小版的黑白图像。这样做是由于目标识别非常消耗计算资源，虽然树莓派可以处理完整尺寸的图像，但是运行速度会非常慢。图像越小，目标识别的精度越低。

我们发现将图像缩小到原始图像的三分之一，是个不错的折衷。通过调整 `factor_down` 和 `factor_up`（它们必须互为倒数）的值，找出适合你自己的参数。



提示 如果想得到更好的性能，可以在 `LXTerminal` 中使用 `raspi-config` 超频你的树莓派。

然后 `detectMultiScale()` 返回匹配目标的所有矩形区域坐标，两个 `for` 循环将这些结果全部绘制到图像上。

9.2.3 深入探索

如果你准备花时间创建训练图像集，那么可以将这个简单的例子运用到多个程序中。例如，可以使用它来识别用户而不再需要用户登录，或者也可以在数控小车上添加 `OpenCV` 使其能够识别简单标志。

即使没有建立自己的 `Haar cascades`，你也可以为树莓派添加视觉特性。例如，使用手势 `Haar cascades`，注意观察矩形区域，你将看到矩形区域随着手的挥动而移动。

9.2.4 使用树莓派摄像头模块

`USB 网络摄像头`并不是在树莓派工程中获取图像和视频的唯一选择。事实上，它可能也不是最流行方法。树莓派基金会自己发布了可以插在主板上的摄像头模块。这里列出了摄像头模块相比 `USB 网络摄像头` 的优点和缺点。

优点：

- 比大多数摄像头有更高的静态图像分辨率（5M 像素）。
- 可以很好地支持 `Linux` 命令行。
- 有竞争力的价格（19 英镑，约 31 美元^①）。
- 提供有红外（夜视）版本。

缺点：

- 对于非树莓派专用程序和 `Python` 模块支持不佳。
- 每个板子只支持一个摄像头。
- 线缆很短（15cm），虽然可延长，但是很难。
- 一般需要根据项目需求选择合适的摄像头。
- 摄像头模块和 `OpenCV`。

① 约 200 人民币。——译者注

前一节提到的 OpenCV，原生不支持树莓派基金会的摄像头模块。但是，你可以按照下列网站上的说明修改 OpenCV，使其能够支持摄像头模块：<http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/>。

写作本书时，还没有支持树莓派摄像头模块的 Python 模块。只有一个支持的项目：<http://github.com/ashtons/picom>。但是仍然不够稳定。此时，最好的方法就是在 Python 中使用命令行工具访问摄像头。

第一件事要做的事就是确保摄像头连接到树莓派上并已经启用。打开 LXTerminal 会话并输入：

```
raspistill -o test.png
```

该命令将输出几秒摄像头的预览图像。然后将结果保存到 test.png 中。如果发现任何问题，请翻阅位于 www.raspberrypi.org 的树莓派摄像头设置指南，在继续进行之前先要修复这些问题。

这是一个最基础的使用命令行获取图像的方法。你也可以在 Python 中做同样的事情：

```
>>> import subprocess
>>> subprocess.call(["raspistill", "-o", "test.png"])
```

如果需要，你可以在 Python 中使用该图像，就像使用其他图像一样。例如，可以使用第 4 章中使用的 PyGame 提供的加载图像特性。raspistill 提供了多种选项来控制图像的拍摄方法。在命令行中输入 raspistill 会列出其所有选项。选项范围从简单，例如图像大小到复杂如测光方式或自动白平衡 (AWB)。

举个例子，下列代码用来拍摄一个 200 × 200 像素大小、高对比度、使用素描图像效果的图像。此外，它不在屏幕上预览并有 1 毫秒的延时。

```
raspistill -w 200 -h 200 -co 90 -n -t 1 -ifx sketch -o test.png
```

结果有点像卡通图像。非常适合拿来做缩时摄影卡通视频，这也是我们在下一个例子中要做的（参见网站上文件 [chapter9-timelapse.py](#)）。

```
import subprocess
import time

frames = 10

for i in range(1, frames):
    filename = "test" + format(i, "03d") + ".jpg"
    subprocess.call(["raspistill", "-n", "-t", "1", "-w", "200",
                    "-h", "200", "-co", "90", "-ifx", "sketch",
                    "-e", "jpg", "-e", "jpg", "-o", filename])
```

```
print("taking photo", i)

print("Encoding...")
subprocess.call(["avconv", "-r", "5", "-i", "test%03d.jpg", "-r",
                "24", "-s", "200x200",
                "-vsync", "cfr", "out.mpg"])
```

这段代码中 `rastpistill` 使用了选项 `-e jpg`，将图像保存为 JPEG 而不是 PNG 格式。为以后转成视频提供方便。获取完所有图片之后，使用 `avconv` 将其转换成视频。这里的关键选项是第一个 `-r`。后面的数字表示创建一秒钟视频使用的图像总量。本例中，我们使用 5，但你可以改成任何自己想要的数字（第二个 `-r` 表示最终生成的视频每秒钟的帧数）。`-i test%3d.jpg` 表示选取五个名字像 `test001.jpg` 和 `test002.jpg` 的文件。这个和创建文件的代码相一致：

```
filename = "test" + format(i, "03d") + ".jpg"
```

这两个部分必须一致，这样视频编码器才能够选用刚才创建的图片。你可以将 `test` 换成任何字母，但必须在两个地方同时更改。如果你希望使用大于 999 的图片，则需要添加更多的 0。例如你希望使用 99 999 张图片，就需要将这两行中的 3 改为 5（变量 `frames` 保存了总的图片拍摄数目，实际上拍摄的数量远小于实际这个数，我们只是举个例子）。

`-s 200 × 200` 表示视频的尺寸是 200 × 200 像素。它不必和图像分辨率一致。你可以将 `out.mpg` 改成任何名字，它表示最后生成的视频文件名。

运行代码，将会创建出一个视频。你可以使用任何标准视频播放器如 VLC 来回放结果。

9.2.5 创建直播视频

如果你希望一段预先设定好时长的视频，上面的方法是个不错的选择，但如果希望获得一段不间断地的视频时应该怎么做呢？例如，如果你希望创建一段直播视频使其能够通过互联网观看时该怎么做呢？在互联网上有个传说，第一个网络摄像头就安装在剑桥大学（树莓派的故乡），人们通过它监控咖啡壶，这样它们就可以在适当的时间到茶水间冲咖啡。下面的例子重现了网络摄像头的案例。

为了通过因特网发送数据，你需要某种形式的服务器。有一些服务器可以做到，但是从熟悉的角度看，我们将继续使用第 7 章中见过的 Tornado 服务器。

下列代码用来创建直播视频。运行之前，你需要首先使用命令 `mkdir/home/pi/images`

创建目录 `/home/pi/images`。

```
import tornado.ioloop
import tornado.web
import subprocess
import time

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        subprocess.call(["raspistill", "-w", "200", "-h", "200",
                        "-e", "jpg", "-n", "-t", "1", "-o",
                        "/home/pi/images/live.jpg"])

        time.sleep(2)
        self.write('<!DOCTYPE html><head>' +
                  '<META HTTP-EQUIV="refresh" ' +
                  'CONTENT="5"></head><body>' +
                  '</body>')

class ImageHandler(tornado.web.StaticFileHandler):
    def set_extra_headers(self, path):
        self.set_header('Cache-Control',
                        'no-store, no-cache, must-revalidate,' +
                        ' max-age=0')

application = tornado.web.Application([
    (r"/", MainHandler),
    (r"/images/(.*)", ImageHandler, {"path": "/home/pi/images"})])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

如果不熟悉这段，就翻到第7章去看下 Tornado 这一节。不过这里还是有一些新东西。

其实，这里所做的一切都是在等待用户请求含有实时视频流（来自根目录，你可以根据自己需要来改动）的页面，然后开始拍摄新图像。打开网页后，网页内包含有拍摄的图像。为了获得实时视频流而不是实时图片，网页中包含有标签 `<META HTTP-EQUIV="refresh" CONTENT="5">`，用来告诉网络浏览器每5秒刷新一次页面。视频质量不高，但足以让你看清壶中咖啡数量的变化。

类 `ImageHandler` 继承自 `tornado.web.StaticFileHandler`。这样我们就可以修改缓存的默认设置。默认情况下，Tornado 为了提高效率不会释放之前保存的图像。当网络浏览器请求一张 Tornado 已经发送过的图像，它会告诉浏览器使用之前发送的图像就可以了。由于浏览器会一直请求图像 `live.jpg`，这样它就永远只能得到第一张图片。添加 `set_extra_headers`，就是告诉 Tornado 不使用缓存，每次都重新读取文件。

现在可以工作了，但还不够理想。例如每次有人请求页面时它都会重新获取一张图片，如果很多人不断请求页面，会造成拥塞，因为摄像头每次只能拍摄一张照片。如果只

是监视咖啡壶，这样做没有问题。但如果用作保安摄像头，就会遇到麻烦。

为了避开这个问题，我们需要将拍照部分分割成独立的一部分，而另一部分用来提供服务。下列代码会连续地拍照片，这些照片可以用来合成视频，同时，使用最新的图像来更新 live.jpg。参见网站文件 chapter9-live-take-pics.py：

```
import subprocess
import time
file_number = 0
dir = "/home/pi/images/"

while True:
    file_name = dir + format(file_number, "05d") + ".jpg"
    file_number = file_number + 1
    subprocess.call(["raspistill", "-w", "400", "-h", "400", "-e",
                    "jpg", "-n", "-t", "1", "-o", file_name])
    subprocess.call(["cp", "-f", file_name, dir + "live.jpg"])
    time.sleep(2)
```

下面一段会代码自动运行上面的脚本，并和上个例子一样显示最新一幅图像。参见网站文件 chapter9-live-image-server.py。

```
import tornado.ioloop
import tornado.web
import subprocess

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write('<!DOCTYPE html><head>' +
                  '<META HTTP-EQUIV="refresh"' +
                  'CONTENT="5"></head><body>' +
                  '</body>')

class ImageHandler(tornado.web.StaticFileHandler):
    def set_extra_headers(self, path):
        self.set_header('Cache-Control',
                        'no-store, no-cache, must-revalidate, max-age=0')

application = tornado.web.Application([
    (r"/", MainHandler),
    (r"/images/(.*)", ImageHandler, {"path": "/home/pi/images"})])

if __name__ == "__main__":
    subprocess.Popen(["python3", "chapter 9-live-take-pics.py"])
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

这里使用 subprocess.Popen() 来创建一个新进程，使两个程序同时运行（将在下一章中详细介绍）。

9.2.6 深入探索

不过这里还有一些问题。这段代码将会不停运行，直到图像把 SD 卡占满为止。此时，树莓派将会崩溃。如果你希望进一步探索，可以试着做一下。例如，每隔一定时间（可以是每天，或者每 20 000 张图像），将图片压缩成视频，然后删除图像。另外，也可以将视频传到网上。

为了更安全，可以将视频传到异地的中央服务器上。当树莓派丢失或损坏后，仍然能够得到这些视频。

如果你是在公共网络服务器上做开发，可以实现一些安全方面的特性，只允许授权用户在线查看视频。详情请参阅第 7 章相应小节。

9.3 小结

读完本章，你应该更好地理解以下内容：

- PyAudio 是一个不错的模块，可以为 Python 程序添加声音。
- wave 模块可以输出 WAVE 格式的文件。
- 并不是所有声音处理任务都有相应的 Python 工具，但没有问题，可以使用 Linux 命令行工具如 sox 和 flac 来弥补这个缺陷。
- 可以使用网络服务如 Google 的语言识别来增加更多的音频特性。
- 为树莓派添加摄像头有两个选择：USB 网络摄像头和树莓派基金会的摄像头模块。
- USB 网络摄像头能很好地兼容标准 Python 工具如 PyGame 和 OpenCV。
- 摄像头模块也在 OpenCV 中工作，但需要使用自己的命令行工具来捕获图像和视频。
- 使用 OpenCV 可以添加机器视觉特性如目标识别。
- 网络服务器如 Tornado 可以用来在互联网上提供图像流服务。

脚 本

为了保证计算机正确合理地运行，需要执行很多不同的任务。例如，为防止数据丢失，你需要定期备份数据。还有些任务可以根据你的需求来完成某件事情。你可能需要给音乐、图片排序。本章主要关注如何利用 Python 自动化这些常规工作，尽可能地减少操作计算机，让生活变得更简单。

这需要很多和底层操作系统的交互，因此首先要学习一点 Linux 知识。

10.1 从 Linux 命令行开始

现在你可能已经熟练掌握了 Raspbian 的常规操作。Raspbian 作为操作系统有几个明显特征。首先，它使用了 Linux 内核。内核开机后一直运行，为系统提供了底层支持，用来管理硬件和内存，控制程序如何运行。其次，它为使用 Raspbian 提供了命令和大量工具，这些都是基于文本模式的。在本章中将详细介绍这部分内容。Raspbian 的另一个特性就是提供了桌面环境，之前我们接触过，但在本章中我们很少使用它。默认的 GUI 是 LXDE，基于 LXDE 有很多图形界面程序。

如果经常使用 Windows，你首先面临的重大变化可能就是文件系统。树莓派上找不到 C 盘（或者类似的其他盘符）。所有东西都是基于一个从 /（即根，root）开始的层次结构。开始学习 Python 脚本时，先要东西放在文件系统的什么位置。打开 LXTerminal（树莓派中提供命令行工具的窗口）并输入以下内容：

```
cd /
ls
```

第一个命令将目录更改到 / (根目录), 第二个命令列出当前目录下的所有内容。运行后可以得到类似的结果 (见图 10-1):

```
bin    dev    home    lost+found  mnt    proc    run    selinux  sys    usr
boot   etc    lib     media       opt    root    sbin   srv      tmp    var
```

这些是根目录下的目录 (如果根目录下有文件, 也会在此列出, 但现在这里没有任何文件)。如果使用其他 Linux 系统, 你会在根目录 / 下发现相同的子目录。这里的大多数目录你都不需要使用。Raspbian 会帮你管理好它们, 并保持更新。需要使用的有: 用户 home 目录 /home, 可移动设备如 USB 记忆棒的目录 /media, 和保存系统基本的应用程序集合 /etc。

认识到这个文件系统和保存在硬盘上的文件系统不同非常重要。它和 Windows 的做法不同。例如, /home/pi 存在 SD 卡上。然而如果将名字为 MyStick 的 USB 记忆棒插入树莓派, 你将会在 /media/MyStick 中找到它。

/sys 和 /proc 在任何磁盘上都不存在。它们是虚拟文件系统, 看起来和正常的目录和文件类似, 但只是系统用来显示系统的一种方式。例如, 在 LXTerminal 中输入:

```
cat /proc/cpuinfo
```

cat 是用来输出一个或多个文件内容的命令。/proc/cpuinfo 中包含了 CPU 的技术细节。你将得到类似输出:

```
pi@raspberrypi /proc $ cat /proc/cpuinfo
Processor: ARMv6-compatible processor rev 7 (v6l)
BogoMIPS: 697.95
Features: swp half thumb fastmult vfp edsp java tls
CPU implementer: 0x41
CPU architecture: 7
CPU variant: 0x0
CPU part: 0xb76
CPU revision: 7
```

```
Hardware: BCM2708
```

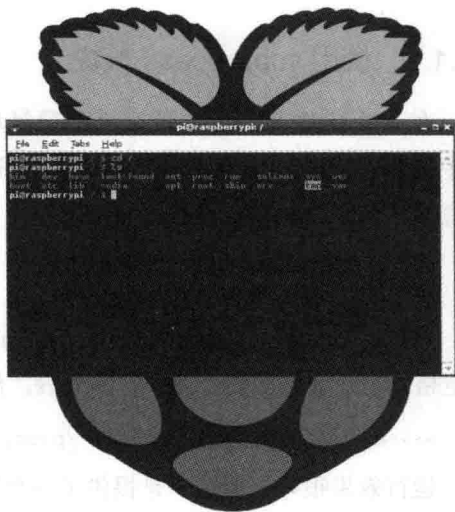


图 10-1 通过 LXTerminal 可以和强大的 Linux 命令行换件交互

```
Revision: 000d
Serial: 0000000074f3d523
```

LXTerminal 使用了 Bash 环境。这是个强大的环境，并提供自己的编程语言。如果希望学习更多关于 Raspbian 以及通用 Linux 的内容，可以从 Bash 开始，可用的资源有很多（例如 www.linuxcommand.org）。然而这是一本关于 Python 的书，因为我们将不再深入下去，有兴趣的读者可以自学。

10.1.1 使用 subprocess 模块

在 Python 中与底层系统交互的最简单的方法就是使用 subprocess 模块。我们在第 7 章中使用过它，你应该还有点印象。打开 Python 解释器（如果还在使用 LXTerminal，可以直接输入 python3），并输入以下内容：

```
>>> import subprocess
>>> subprocess.call("ls")
```

可以看出，你可以通过 subprocess.call() 在底层系统中运行任何命令。如果命令中含有空格，则需要将其划分成字符串列表。例如，命令 cat/proc/cpuinfo 需要变成：

```
>>> subprocess.call(["cat", "/proc/cpuinfo"])
```

运行效果很好，但这只是提供了一个更繁琐的执行命令的方式。毕竟，你可以简单地通过 LXTerminal 来执行相同的命令。正如我们在开始时说的那样，本章的目标是自动运行普通任务，为了做到这一点，你需要分析命令输出，然后才能操纵它们。

例如，cat/proc/cpuinfo 返回很多信息，大多数你可能都不需要关注。下面的程序剔除了除计算机的处理器类型之外的所有信息。

```
import subprocess

p = subprocess.Popen(["cat", "/proc/cpuinfo"],
                      stdout=subprocess.PIPE)

text = p.stdout.read().decode()

for line in text.splitlines():
    if line[:9] == "Processor":
        print(line)
```

这里使用了 subprocess.Popen() 而不是 subprocess.call()。它创建了一个新对象，我们可以使用该对象来获取想要的信息，从而实现更多的控制功能。

运行在 Linux 机器上的命令，通常有两个输出：stdout 和 stderr。所有正常输出都输出到 stdout（或者标准输出（standard out））。例如当运行 Python 程序时，所有的 print() 语句

都将输出到 `stdout`。系统将错误信息传送到 `stderr`（或者标准错误（standard error））中。如果命令在 `LXTerminal` 中执行，这 `stdout` 和 `stderr` 都会显示到屏幕上，但如果使用脚本来运行程序，将这两者分开会更好。比如，如果运行很多命令，可以将所有出错信息放在一个地方。这样有错误发生时可以立即看到而不用再去检查所有的输出信息。

参数 `stdout=subprocess.PIPE` 告诉 Python 将 `stdout` 输出到我们创建的对象中而不是到屏幕上。由于你没有告诉它将 `stderr` 输出到哪里，默认情况下，它将输出到屏幕。因此，如果在上述程序中使用下面这行：

```
p = subprocess.Popen(["cat", "/proc/cpuinfozz"],
                      stdout=subprocess.PIPE)
```

即使没有在任何的 `print()` 语句中打印出错信息，它仍然会被打印到屏幕上，

如果需要，你也可以捕获 `stderr`。例如，在下面的程序中，将把用户输入的文件名对应的文件内容。

```
import subprocess

f_name = input("Enter a filename: ")

p = subprocess.Popen(["cat", f_name ], stdout=subprocess.PIPE,
                      stderr=subprocess.PIPE)

text = p.stdout.read().decode()
error = p.stderr.read().decode()

print(text)

if len(error) > 0:
    print("*****ERROR*****")
    print(error)
```

10.1.2 命令行标签

Linux 命令通常带有选项。这些选项跟在主命令之后，为命令提供附加信息。例如，在 `LXTerminal` 中，执行下列命令：

```
ls
ls -a
ls --all
```

`ls` 列出了当前目录下的内容。使用选项 `-a`，可以列出目录中的所有东西（通常 `ls` 会忽略以 `‘.’` 开头的文件和目录）。`--all` 和 `-a` 一样。许多命令的同一个选项都有两个版本——以 `-` 开头的短格式和以 `--` 开头的长格式。使用标签 `-h` 或者 `-help` 运行 `ls`，可以获得更多帮助信息。

很快你就会看到，选项后面还可以带值。

如果你在开发脚本，应该尽最大可能遵守这些约定。optparse 模块可以帮助我们。上一个例子的文件名可以改成从选项输入而不是提示用户输入。看看下面的例子：

```
import subprocess
from optparse import OptionParser

parser = OptionParser()

parser.add_option("-f", "-file", dest="filename",
                  help="The file to display")

options, arguments = parser.parse_args()

if options.filename:
    p = subprocess.Popen(["cat", options.filename ],
                          stdout=subprocess.PIPE,
                          stderr=subprocess.PIPE)

    text = p.stdout.read().decode()
    error = p.stderr.read().decode()
else:
    test = ""
    error = "Filename not given"
if len(error) > 0:
    print("*****ERROR*****")
    print(error)
else:
    print(text)
```

可以看到，这里创建了 OptionParser 对象。本例中只有一个选项，你可以多调用几次 parser.add_option() 来增加更多选项。其中前两个参数表示选项的短格式和长格式。dest="filename" 表示该选项的值保存在 options 返回的 filename 属性。

注意 parser 将自动添加 -h 和 --help 选项，在 add_option() 调用中使用 help= 参数建立帮助文本。

如果你将上述代码保存为 print-file.py，可以在 LXTerminal 中 cd 到文件所在目录，输入 python3 print-file.py -f /proc/cpuinfo 来运行它，也可以通过 python3 print-file.py --help 查看帮助。

10.1.3 正则表达式

不过这里只是将 cat 命令封装到 Python 中和直接执行该命令的效果相同，并没有增加任何东西。

现在我们来添加一个特性，根据用户指定来显示文件的某几行。

Python（和许多其他编程语言）有一个称为正则表达式的特性。它的名字有点奇怪（通

常缩写为 `regex`)，可以用来匹配指定的文本。

正则表达式通过特殊字符来工作。最常见的特殊字符是 `*`。表示匹配前面的字符零至多次（任意数量）。例如，`do*g` 将匹配 `dg`、`dog`、`doog` 等。字符 `+` 表示匹配前面的字符一至多次。例如，`do+g` 将匹配到 `dog`、`doog`、`dooog` 等，但是没有 `dg`。而 `do?g` 将只匹配 `dg` 或者 `dog`。

句号将用来匹配除换行符外的任意字符，因此，`.*` 将匹配任意行，而 `.+` 将匹配所有非空行。

还可以将字符集合在一起，例如 `d[io]g` 将匹配 `dig` 和 `dog`，`d[io]*g` 将匹配 `dg`、`dog`、`dig`、`doog`、`dioioioig` 等其他类似的字符串。

稍后你将看到正则表达式的更多特性，现在让我们开始学习如何使用它们。下面的代码来自网站文件 `chapter10-regex.py`。

```
import subprocess
from optparse import OptionParser
import re

parser = OptionParser()

parser.add_option("-f", "-file", dest="filename",
                  help="The file to display")

parser.add_option("-r", "-regex", dest="regex",
                  help="The regular expression to search for")
options, arguments = parser.parse_args()
if options.filename:
    p = subprocess.Popen(["cat", options.filename ],
                          stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    text = p.stdout.read().decode()
    error = p.stderr.read().decode()
else:
    test = ""
    error = "Filename not given"

if len(error) > 0:
    print("*****ERROR*****")
    print(error)
else:
    for line in text.splitlines():
        if not options.regex or (options.regex
                                and re.search(options.regex, line)):
            print(line)
```

代码使用了 `re` 模块提供的正则表达式。使用正则表达式有两种主要方式：`re.match()` 和 `re.search()`。第一个方法只检测 RE 是不是在 `string` 的开始位置匹配，第二种方法则会扫

描整个字符串查找匹配。上述程序使用了后者，因为我们希望能够尽可能简单地匹配到指定的那几行。

if 语句中的条件有点复杂：

```
if not options.regex or (options.regex
    and re.search(options.regex, line)):
```

它用来处理用户没有输入 -r 或者 -regex 选项的情况。如果没有 regex 选项或者有该选项并且匹配到相应的行，就输出该行。

运行这段代码，将得到与第一个例子相同的结果：

```
python3 chapter.py -f /proc/cpuinfo -r Processor
```

Linux 系统将注册的各种事件保存在 log 文件中，对于诊断问题十分有用。但 log 文件会变得很大，使用起来不方便。它们都存放在 /var/log 文件夹中。syslog 保存了系统的各种功能的信息。例如，如果你遇到了 USB 驱动的问题，运行下列命令将列出树莓派中注册新的 USB 外围设备的所有事件（无论是 hub 还是设备）：

```
python3 chapter10-regex.py -f /var/log/syslog -r"USB.*found"
```

这里不需要将正则表达式写在括号中，但有时 Linux shell 在传值给 Python 前会处理一些特殊字符，使用双引号将阻止这些操作，因此在命令行中使用正则表达式时使用双引号是个好习惯。

回到正则表达式的特性：

- `[^abc]` 匹配任何除 a、b 和 c 之外的任意字符，因此 `d[^o]g` 匹配 dig、dxg 和 dag 但不包括 dog。
- `[a-c]` 匹配从 a 到 c 的字符，因此 `d[a-j]g` 匹配 dag、dbg 和 dig，但不包括 dkg 或 dog。
- `{a,b}` 匹配前面的字符 a 至 b 次。因此 `do{2,4}g` 匹配 doog、dooog 和 dooooog，但不包括 dog 或 dooooog。

此外，还有一些由斜线开头，有特殊含义的特殊字母。表 10-1 列出了这些特殊字母。

表 10-1 正则表达式中的转义字符

代码	说明
<code>\n</code>	换行符
<code>\t</code>	制表符，Tab
<code>\d</code>	任意数字
<code>\D</code>	任意非数字的字符
<code>\s</code>	任意的空白符
<code>\w</code>	字母或数字或下划线或汉字
<code>\W</code>	不是字母、数字、下划线、汉字的字符

显然，在匹配 \ 字符时会遇到问题。为此，可以使用转义符 \ 来匹配特殊字符（例如 .，* 或者 +）。因此，\\w 匹配字符 \w，而 \w 匹配任意字母数字字符组。

由于斜线需要使用转义符，在 Python 中遇到字符串时会产生新的问题。例如：

```
>>> print("\\")
\
>>> print("\\\\")
\\
```



注意 这个例子中没有问题，因为你已经将字符串传递给 Python。但如果你在 Python 中创建用于匹配正则表达式的字符串，则需要记住在正则表达式中使用的每个斜线字符前多加一个斜线（用作转义符）。

10.2 知识测验

我们用很短的篇幅介绍了大量关于正则表达式的内容。为了确保你能够完全理解，请看下面的练习。

将下列字符保存到文件中：

```
aaa
a10
10
Hello
Hel1lo
Helloo
```

下列正则表达式分别能够匹配出哪几行？试着回答一下，然后运行上一个例子查看结果（你需要使用 Leafpad 创建文件或者从网站上下载 chapter10-regex-test）。例如，运行下列命令可以得到第一个答案：

```
python3 chapter10-regex.py -f chapter10-regex-test -r"."
```

记得在程序中使用 `re.search()` 而不是 `re.match()`

- .
- \d
- \D\d
- 1{3,4}
- e*
- e+
- [Ha]

```
■ \d{2,3}
```

```
■ 1?
```

10.3 脚本中的网络

在第 7 章中已经介绍过网络，此处不再赘述。这里我们将介绍一个在脚本中使用网络的方法。最常见的事情就是在两个计算机之间复制文件。有个叫 Fabric 的模块非常好用，但在本书写作之时，它还不支持 Python3。情况可能会改变，但也不会很快。

由于缺乏可用模块，你需要构建 Python 代码来获取和复制所有东西。这也确实可行，但需要花费太长时间。还有一个 Linux 命令行程序 scp（secure copy，安全复制）可以做到这点，并且你也已经见过如何运行命令行程序。

尽管这是一个很小的程序，在正常运行 scp 时，它需要询问你的密码。这在 Python 脚本中会产生问题，因为你不能简单地让它回答问题。然而，scp 允许你使用证书来登录。这样当你以合法用户身份登录到一个机器上之后，不需要再使用密码就可以直接登录。

scp 可以用于在不同的 Linux 计算机（如树莓派）之间复制信息，因此可以用于两个树莓派，或者树莓派和 Linux 服务器。其他（非 Linux）服务器也可能支持 scp，但需要让你的系统管理员安装这个服务。

首先要做的是登录到你需要传送信息的机器上，并运行在命令行中，如 LXTerminal，运行下列命令。如果你只有一台 Linux 机器，你可以在单个机器上做这个实验，因此在本机上运行该命令。如果你是远程登录，可以通过 ssh 来操作。

```
ssh-keygen -t rsa
```

它将会在你的 home 目录下的 .ssh 文件夹中创建两个文件，id_rsa 和 id_rsa.pub。其中包含了公钥和私钥。id_rsa.pub 是公钥，需要将其复制到你要登录的机器上去。可以使用 USB 记忆棒，云存储（如 Dropbox 或 Google Drive），甚至是 email，但由于我们在介绍 scp，你也可以使用它。scp 的命令格式如下：

```
scp location1 location2
```

即将文件从 location1 复制到 location2。如果其中一个位置是正常文件路径，如 /home/pi/.ssh/id_rsa.pub，scp 将把它当做本地路径。如果路径是这个格式：user@machine:/home/pi，scp 将使用 user 登录远程主机。machine 可以是 IP 地址或主机名。对于树莓派，将使用 IP 地址。如果你希望使用用户名 pi 将 id_rsa.pub 复制到 IP 地址为 192.168.0.10 的主机，则命令如下：

```
scp /home/pi/.ssh/id_rsa.pub pi@192.168.0.10:/home/pi
```

如果你是在单个主机上操作，则不需要移动文件。但还是试一下 `scp`，可以使用主机 `localhost`。因此复制命令如下：

```
scp /home/pi/.ssh/id_rsa.pub pi@localhost:/home/pi
```

其次你需要登录刚才复制文件的远程主机，使用下面的命令将文件内容复制到 `authorized_keys` 中：

```
cat /home/pi/id_rsa.pub >> /home/pi/.ssh/authorized_keys
```

看起来有点烦琐，但现在不使用密码就可以将文件从本机复制到其他机器上了（相反方向不可以）。输入下面的命令试一下：

```
touch test
scp test user@machine:/home/pi/
```

其中 `user`，`machine` 和 `/home/pi` 可以根据自己的环境做适当变化。第一行只是简单地创建一个名为 `test` 的文件。如果一切顺利，就不再需要输入密码了。

所有这些设置完成之后，你就可以使用 `subprocess.call()` 在两个机器之间复制文件。例如：

```
subprocess.call(["scp", "file1.py", "pi@localhost:/home/pi"])
```

10.4 组合包装

在本章开始的时候，我们说过 `Python` 可以使计算机常规操作变得更简单。这里已经演示了很多不错的 `Python` 工具，但还没有满足这个目标。现在我们就开始。本节中，我们将创建一个 `Python` 程序，可以帮助你备份常用文件，以防数据丢失。SD 卡损坏后，你也可以恢复自己的数据。

我们所使用的工具大部分都在本章中介绍过。另外还需要一个新的 `os` 模块。它提供了访问操作系统的功能。下面这段代码（后面有解释）可以从网站上下载 `chapter10-backup.py`：

```
import os
import tarfile
from optparse import OptionParser
from time import localtime
import datetime
import subprocess
import re
```

```

parser = OptionParser()

parser.add_option("-f", "--file", dest="filename",
                  help="filename to write backup to (if no option  
is give, backup will be used)", metavar="FILE")
parser.add_option("-p", "--path", dest="path",
                  help="path to backup (if no option is give, ~  
will be used)")
parser.add_option("-v", "--verbose", action="store_true",
                  dest="verbose", default=False,
                  help="print status messages to stdout")
parser.add_option("-i", "--images", action="store_true",
                  dest="images", default=False,
                  help="backup image files")
parser.add_option("-c", "--code", action="store_true", dest="code",
                  default=False,
                  help="backup code files")
parser.add_option("-d", "--documents", action="store_true",
                  dest="documents", default=False,
                  help="backup document files")
parser.add_option("-a", "--all", action="store_true", dest="all",
                  default=False,
                  help="backup all filetypes (this overrides  
c, d & i)")
parser.add_option("-m", "--mtime", dest="mtime", default=False,
                  help="backup files modified less than this many  
days ago")
parser.add_option("-r", "--regex", dest="regex",
                  help="only back up filenames that match this  
regex")
parser.add_option("-s", "--server", dest="server", default=False,
                  help="copy backup file to this remote point  
(should be an scp location)")

options, arguments = parser.parse_args()

if options.filename:
    backup_file_name = options.filename + '.tar.gz'
else:
    backup_file_name = "backup.tar.gz"

backup_tar = tarfile.open(backup_file_name, "w:gz")

file_types = {"code":[".py"],
              "image":[".jpeg", ".jpg", ".png", ".gif"],
              "document":[".doc", "docx", ".odt", ".rtf"]}
backup_types = []
all_types = False

if options.images:
    backup_types.extend(file_types["image"])

```

```

if options.code:
    backup_types.extend(file_types["code"])
if options.documents:
    backup_types.extend(file_types["document"])

if len(backup_types) == 0 or options.all:
    all_types = True
if options.mtime:
    try:
        mtime_option = int(options.mtime)
    except ValueError:
        print("mtime option is not a valid integer.",
              "Ignoring option")
        mtime_option = -1
    else:
        mtime_option = -1

if options.path:
    if os.path.isdir(options.path):
        directory = options.path
    else:
        print("Directory not found. Using ~")
        directory = os.getenv("HOME")
else:
    directory = os.getenv("HOME")

for root, dirs, files in os.walk(directory):
    for file_name in files:
        if not options.regex or re.match(options.regex, file_name):
            name, extension = os.path.splitext(os.path.join(root,
                                                              file_name))

            if (extension in backup_types) or all_types:
                modified_days = (datetime.datetime.now() -
                                datetime.datetime.fromtimestamp(
                                    os.path.getmtime(
                                        os.path.join(root,
                                                    file_name))))
                .days
                if mtime_option < 0 or modified_days <
                    mtime_option:
                    if options.verbose:
                        print("Adding ",
                              os.path.join(root, file_name),
                              "last modified", modified_days,
                              "days ago")
                    backup_tar.add(os.path.join(root, file_name))

if options.server:
    subprocess.call(["scp", backup_file_name, options.server])

```

通过各种 `parser.add_option()` 可以传递不同的信息，你可以根据自己需要改变这些代码。它的基本功能就是将文件复制到 `tar.gz` 文件中，即 Linux 系统中流行的一种压缩文件。

改文件随后会自动复制到位于另一个服务器上的安全位置。如果源文件遭到破坏，就可以通过备份恢复它。

运行 `python3 chapter10-backup.py-help`，将会看到如下输出，即帮助信息：

```
Usage: chapter10-backup.py [options]

Options:
  -h, --help            show this help message and exit
  -f FILE, --file=FILE  filename to write backup to (if no option
                        is given, backup will be used)
  -p PATH, --path=PATH  path to backup (if no option is given, ~
                        will be used)
  -v, --verbose          print status messages to stdout
  -i, --images           backup image files
  -c, --code            backup code files
  -d, --documents       backup document files
  -a, --all             backup all filetypes (this overrides c, d &
                        i)
  -m MTIME, --mtime=MTIME
                        backup files modified less than this many
                        days ago
  -r REGEX, --regex=REGEX
                        only back up filenames that match this
                        regex
  -s SERVER, --server=SERVER
                        copy backup file to this remote point
                        (should be an scp location)
```

基本使用方法如下：

```
python3 chapter10-backup.py -f backup.tar.gz -p /home/pi -s \
pi@192.168.0.10:/home/pi/backups/
```

这些参数表示程序将搜索 `/home/pi` 及其子文件夹中的文件。程序中使用了 `os.walk()` 函数来完成此项操作。它将返回一个目录和文件的集合，可以使用 `for` 循环来处理，即下面这两行代码：

```
for root, dirs, files in os.walk(directory):
    for file_name in files:
```

第一行用来遍历目录，三个返回元素分别表示每次遍历的路径名（`root`）、目录列表（`dirs`）和文件列表（`files`）。由于本程序只关心文件，这里只有一个内层循环来遍历文件。`os.walk()` 会自动遍历所有子文件夹，不需要亲自动手。

还有些选项可以用来限定文件的选择范围。最基本的一个是根据文件类型，`-i`、`-c` 和 `-d` 分别用来限定图像，代码和文档。`-a` 覆盖了这三个选项，并选择所有类型文件（默认值）。这些选择并不是完美的，它只是根据文件类型字典来工作：

```
file_types = {"code":[".py"],
              "image":[".jpeg", ".jpg", ".png", ".gif"],
              "document":[".doc", ".docx", ".odt", ".rtf"]}
```

如果选择了相应的文件类型，则会备份其中列出的所有扩展名的文件。下面这行代码用来得到文件的扩展名：

```
name, extension = os.path.splitext(root, file_name)
```

函数 `os.path.splitext()`（注意这里是 `split-ext` 而不是 `split-text`）将文件名分为两部分，分开返回两个值。第一个是主文件名（即 `name` 变量），剩下的一个是扩展名。（存储在 `extension` 变量中）。剩下要做的就是检查扩展名是否包含在 `backup_types` 中。这里我们将所有指定的文件类型都集中到了 `backup_types` 里。指定文件类型之前，先确保该集合包含了所有需要备份的文件类型。

`-r` 或 `--regex` 可以用来指定正则表达式，只有匹配的文件名才会备份。下面这行给出了代码：

```
if not options.regex or re.match(options.regex, file_name):
```

注意这里使用了 `re.match()` 而不是 `re.search()`。这意味着整个文件名必须匹配该正则表达式。例如，正则表达式 `"*\.py"` 将会匹配所有扩展名为 `.py` 的文件（和使用 `-c` 选项效果相同）。

最后一个选项是 `-m` 或 `--mtime`，即修改时间（`modified time`）的简写。换句话说，它将备份所有最近修改时间小于指定天数的文件。代码实现比较复杂：

```
modified_days = (datetime.datetime.now() -
                  datetime.datetime.fromtimestamp(
                      os.path.getmtime(
                          os.path.join(root, file_name)))) .days
if mtime_option < 0 or modified_days < mtime_option:
```

其中主要工作都是由 `os.path.getmtime()` 来完成的。这里需要使用绝对路径（即 `/home/pi/filename` 而不只是 `filename`），它将返回文件的最后修改时间。

`os.path.join()` 需要两个参数：路径和文件名，将其合成为前面函数所需的参数（不能简单地将两个字符串合并起来，因为路径有时以 `/` 结尾，有时不是）。

`os.path.getmtime()` 返回的时间并不是标准时间，而是自 1970 年 1 月 1 日以来经过的秒数（这是 Unix 系统标准，在 Linux 系统中也是如此）。因此为了计算出文件的最后修改时间到现在过了几天，我们首先需要使用 `datetime.datetime.fromtimestamp()` 将其转换为 Python 的 `datetime`，然后再减去当前时间。

完成这些之后，剩下的就是比较计算结果是否小于选项中指定的天数。将（`mtime_option<0`）作为 if 中的起始子句是因为如果程序在处理用户输入时出错或者用户没有使用 `mtime` 选项，默认指定 `mtime_option` 是 `-1`。

这些都是限定文件选择范围的选项。确定之后，剩下的事情就是将其加入 `tar` 文件。这些由 `tarfile` 模块完成，它提供了一个简单的处理归档文件的方法。你只需要像下面的代码一样，在开始的时候打开文件就够了：

```
backup_tar = tarfile.open(backup_file_name, "w:gz")
```

第二个参数（`w:gz`）表示以写方式打开文件，并且指定这是个 `gzipped`（即，压缩）文件。下面一行代码用来将指定的文件添加到归档中：

```
backup_tar.add(os.path.join(root, file_name))
```

如果指定了相应选项，你应该认识使用 `scp` 复制文件到远程服务器的代码。

这段程序只是展示了一个让计算机保持有序脚本。它可以用来完成烦琐的备份工作，然而，你还是得记得去运行它来创建备份。这里 Python 帮不了你，但是有个叫做 `crontab` 的 Linux 特性可以在指定时间运行程序，换句话说就是类似使用者的时程表。该程序所有选项都在命令行中设置，你需要做的就是决定需要运行什么，并设置 `chrontab` 在指定时间开始运行。

`crontab` 有两个主要选项：`-l` 列出当前设置的需要运行的程序，`-e` 打开文本编辑器来什么时间运行什么程序。每个任务占一行，由 5 个用空格隔开的数字或星号开始，命令跟在后面。这 5 个数字分别代表程序需要运行的分钟、小时，一个月中的第几天、月份和一个星期中的第几天，星号表示任意。比如下面的例子（由于排版原因，将其看做一行）：

```
0 0 1 * * python3 /home/pi/chapter10-backup.py -s
pi@192.168.0.10:/home/pi/backups
```

它将会在每个月第一天的午夜备份你的 `home` 目录。下面的例子将会在每天中午运行：

```
12 0 * * * python3 /home/pi/chapter10-backup.py -s
pi@192.168.0.10:/home/pi/backups
```

10.5 在 Python 中操作文件

在本章中，已经看到过很多操作文件的方法。然而你还没有真正在 Python 中打开任何文件来读写数据（除了 `tar` 归档，这是个特殊案例）。这一小节，你将会看到如何将信息存入文本文件，然后再从中读出。

实际上非常简单。你需要做的就是调用 `open()` 函数。例如，下面代码将打开 `myfile.txt` 文件，并把它的一行都打印出来：

```
file_a = open("myfile.txt", encoding="utf-8")

for line in file_a:
    print(line.strip())

file_a.close()
```

`open()` 函数创建了 `file` 对象。它可以接受多个参数。最基本的就是文件名，编码格式也是一个很重要的参数，用来告诉 Python 文件的格式。大多数文本文件都使用 `utf-8` 格式，创建自己的文件时也需要使用该格式。



注意 不能使用 `file` 作为变量名，以为它在 Python 已经被用作其他用途。

打开文件之后，可以使用 `for` 循环来遍历它。这里唯一有点不常见的是我们在 `line` 后面使用了 `.strip()`。这是因为文件中的每一行都包含有换行符，被打印出来时 `print` 函数又自己添加了新的换行符，因此没有这个函数，打印出来的每两行之间都会多出一个空行。

每打开一个文件，都需要关闭它。这里给出一个自动关闭文件的例子：

```
with open("myfile.txt", encoding="utf-8") as file_a:
    for line in file_a:
        print(line.strip())
```

这两段代码所做的工作都一样。`with` 语句在代码段结束之后会自动关闭文件。如果你容易忘记关闭文件，这将会非常有用。

写文件也很简单。只需要在 `open()` 的参数中加入 `mode="w"` 后就可以写文件了。看看下面的代码：

```
with open("myfile.txt", mode="w", encoding="utf-8") as file_a:
    for letter in "abcde":
        file_a.write(letter + "\n")

with open("myfile.txt", encoding="utf-8") as file_a:
    for line in file_a:
        print(line.strip())
```

它将重写 `myfile.txt`。如果希望在文件末尾添加内容，可以使用 `mode="a"`（附加，`append`）。它将保持文件原有内容不变，将新内容添加到文件末尾。

10.6 小结

读完本章，你应该更好地理解以下内容：

- 树莓派上运行的是一个 Linux 版本的系统。
- Linux 的文件系统和 Windows 不同，是基于根目录，/ 的文件系统。
- Linux 具有完整的文本模式。
- 使用 subprocess 模块可以在文本模式下运行命令。
- Linux 命令可以输出到 stdout 和 stderr。
- 写 Python 脚本时，如果可能，从命令行的标记中获取全部输入非常有用。
- 正则表达式可以用来匹配文本模式。
- scp 可以用来在不同计算机间复制文件。
- os 模块提供了很多函数来帮助你 and 操作系统间交互（远超过我们这里的覆盖范围，查看 Python 文档可以获得更多信息）。
- open() 函数可以为读写打开文件。

硬件接口

和大多数计算机不同，树莓派具有一组通用输入输出（GPIO），通过它们可以与外界交互。SD 卡边上突出的一组金属针就是 GPIO 口。你可以将它们当做可编程开关控制其他事务，或者用它们来从外界接收信息。简单来说，你可以使用它们对树莓派进行各种各样的扩展。它们的应用范围很广，数字艺术家使用它们来创建交互式显示，机器人建造者使用它们来提升自己的作品。想象一下，使用树莓派的 GPIO 和一些组件几乎可以做任何事情。

11.1 硬件设置选择

在开始构建电路之前，你首先需要知道如何连接树莓派 GPIO 口。由于你不能直接将电线连接到 GPIO 口中（实际上你可以直接将它们焊在一起，但我们并不推荐这么做），下一节中我们将介绍连接它们的一些方法。

11.1.1 母转公接头

这应该是最简单的选择。这种接头一端可以连接到 GPIO 口上，另一端可以连到面包板上。这是访问 GPIO 口的最简便方法。本书给出了该方法的图片。可以只连接一部分管脚，如果一次访问大量管脚则有可能造成混乱。参见图 11-1。

Pi Cobbler (树莓派 GPIO 扩展板)

Pi Cobbler (GPIO 扩展板) 可以简单地讲 GPIO 管脚连接到面包板上 (参见图 11-2)。和使用转接线相比, 它并没有提供任何新特性, 只是看起来更整洁, 也不容易混淆管脚。

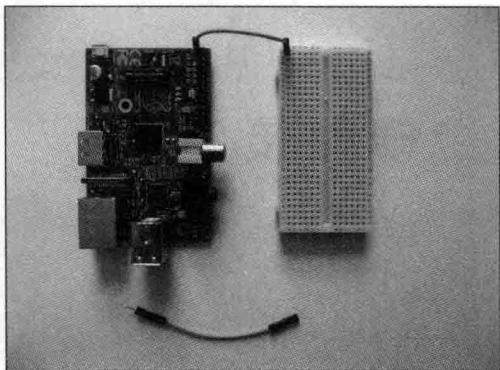


图 11-1 转接头的母头可以连接到 GPIO 管脚, 公头可以插入面包板

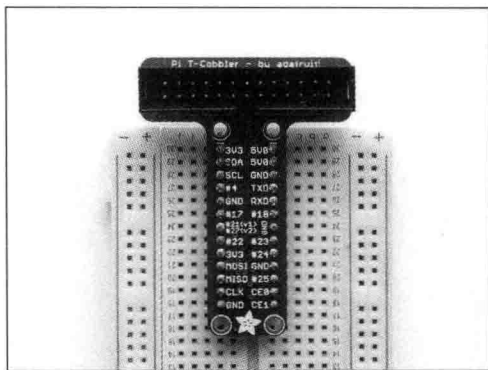


图 11-2 Pi Cobbler 将 GPIO 端口全部连接到面包板上, 避免了连线缠绕和混淆 (Adafruit Industries)

11.1.2 无焊面包板

无论你选择哪种方式, 都需要使用面包板。它提供了一种简单的方式, 可以快速地各个组件的电路连接在一起, 完成之后可以轻松拆除。面包板有不同的尺寸, 但基本排列都类似。典型的面包板的长边有两条平行接口, 可以用来提供正负电源。它们中间是两排插孔, 中间留有间隙。一般将与长边垂直的每 5 个孔板用一条金属条连接。板子中央的一条凹槽, 是针对需要集成电路、芯片试验而设计的 (见图 11-3)。图中, 我们用了一个小无焊面包板, 它没有正负电源槽。你可以使用任何尺寸的面包板。如果对电子感兴趣, 很有必要购买一个全尺寸的面包板, 在大部分项目中都会用到它。

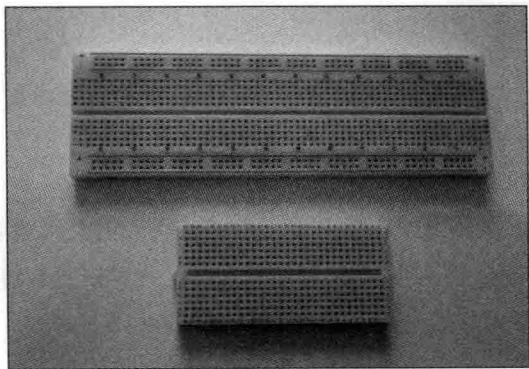


图 11-3 使用无焊面包板可以轻松搭建电路原型, 拆除之后还可以使用相同的原件构建新的电路

你可以将原件直接插入孔中, 使用公对公连接线或者小段单芯线缆将各个元件连接起来。

11.1.3 成品板和万能板

在无焊面包板上完成电路原型后，你可能希望将它们做成成品，将所有元件永久固定在一起，远比无焊面包板耐用。这有点超出了本章范围，但如果希望进一步做得更完善，很快你就会发现自己需要使用成品板。

另一个选择是使用万能板。它上面有很多洞，你可以在上面焊接。各个洞之间没有连接，可以根据需要将其焊接在一起。

11.1.4 PCB 加工

最高等级的选择是制作自己的 PCB（印刷电路板，Printed Circuit Boards）。包括商业印刷在内，有许多方法可以做到这一点。这是整个设计全部完成之后的最终步骤。如果你希望完成这一步，Fritzing 里有些很好的资源（参见 <http://fritzing.org>）。它们提供软件帮助你设计板子，并提供打印服务。

11.2 辅助工具

不需要任何工具就可以搭建简单的电路（只要有了面包板和公对公跳线），但有了这些工具可以更轻松地搭建电路。

11.2.1 剪线/剥线器

从字面上很容易理解它们的用途。如果计划使用单芯电线来连接电路，就需要这个工具（通常这两个功能会集成到一个工具里）。然而如果已经给面包板上用了跳线，就不需要这个工具了。

11.2.2 万用表

万用表用来测量多种参数，包括电压、电流和电阻。它可以帮助定位问题电路。没有万用表，在遇到电路问题时，很难判断连线是否能够导通，或者原件是否坏掉。它还可以简单地判断电阻大小（本章稍后介绍）。

11.2.3 电烙铁

电烙铁可以将两个元件焊接在一起或将元件和电路板焊接在一起。在使用成品电路板或购买了需要焊接到树莓派上的附件时需要使用电烙铁。本章不会讲到焊接，如果需要这

么做，这个网站提供了一个很好的名为“轻松焊接”的指南：http://mightyohm.com/files/soldercomic/FullSolderComic_EN.pdf。

图 11-4 展示了这些工具。



图 11-4 搭建硬件所需工具集。不过对于本章讲到的工程，它们不是必需品

11.3 本章所需的硬件

我们尽最大努力保证本章所需硬件尽可能地简单、易得，并且尽可能便宜。为了完成本章的工程，至少需要下列硬件：

- 发光二极管 (LED)
- 电阻：220 Ω 、1.1k Ω 和 6.2k Ω
- 无焊面包板 (任意尺寸)
- 面包板跳线 (或者单芯电线和剪线钳)
- 一种连接树莓派和面包板的方式——母转公跳线或者 Pi Cobber 均可
- MCP3008 芯片
- 按键开关
- 光敏电阻 (LDR)

在后面的小节中将会用到这些硬件。

11.3.1 第一个电路

在深入电路细节之前，让我们先创建一个简单的电路。你需要一个是面包板，一个 LED，一个 220 Ω 电阻，并将一些 GPIO 口连接到面包板上 (使用母转公跳线或 Pi Cobbler)。

有了这个电路,就可以使用 Python 控制 LED (一种光源) 的开关。可以将 LED 看做是小灯泡,但需要注意两点:首先,它的效率很高,正常使用时不会太热,其次,它是单向导通的。也就是说,它的两个引脚一个是正极,一个是负极,需要分别连接到电源正负极上。发光二极管管体内部金属极较小的是正极,大的片状的是负极[⊖]。

电阻在电路中起到限流作用,避免电流过大。通常情况下,在电路中至少需要一个最小为 220Ω 的电阻,否则就可能烧毁树莓派或者其他元件。其中的细节我们稍后再介绍。

在这个电路中,不一定要使用 220Ω 的电阻, 220Ω 至 470Ω 中间的任意阻值都可以(试一下大阻值的电阻,LED 将会变暗)。

根据电阻上的色环可以直接读出电阻值。通常有四到五个色环(220Ω 的通常有四个,分别是红、红、黑,然后是银色或金色)。同样的,稍后我们将解释它的含义。

如图 11-5 所示,将电阻和 LED 连接到面包板上。注意需要将 LED 的正极(内部金属极小的是正极)和电阻连在一起。

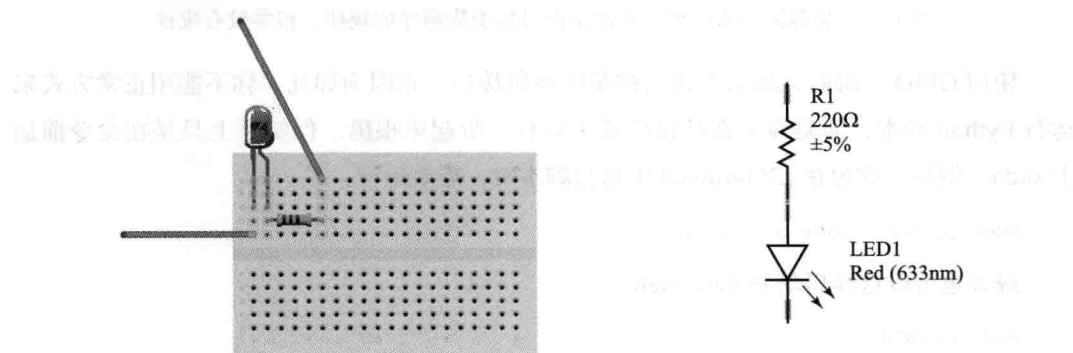


图 11-5 电路的两种图示。左边是物理连接图,右边是原理图

为了确保电路能工作,将电阻上的另一根线和树莓派的 3.3V 引脚相连(参见图 11-6),LED 的另一个引脚和接地引脚相连。如果连接正常,将会点亮 LED。

这次只是将树莓派用做电源。为了能够在 Python 中控制电路,先要安装 RPi.GPIO 模块。首先使用下列命令(在 LXTerminal 中,而不是 Python 中执行)安装了 pip (辅助访问模块的工具):

```
sudo apt-get install python3-pip
```

然后使用这个命令获得需要的库(还是在 LXTerminal 中):

```
sudo pip-3.2 install RPi.GPIO
```

[⊖] 适用于直插式 LED,也可以使用万用表判断正负极。——译者注

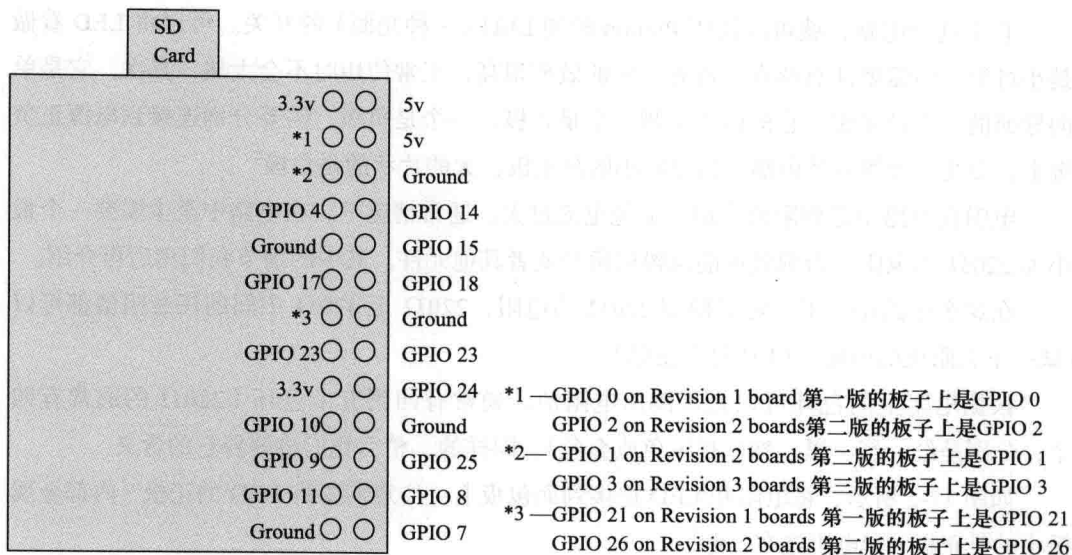


图 11-6 树莓派引脚排列。不用试图寻找引脚顺序的规律，根本没有规律

使用 GPIO 引脚时，就需要访问树莓派底层接口。正因为如此，你不能用正常方式来运行 Python 脚本，而是需要在特权模式下运行。听起来很酷，但实际上只是在命令前加上 `sudo`。例如，你想在 `LXTerminal` 中运行脚本时，需要运行：

```
sudo python3 your-script.py
```

或者也可以这样启动 Python shell：

```
sudo python3
```

或者你可以在特权模式下在 `LXTerminal` 中运行如下代码启动 `IDLE3`：

```
sudo idle3
```

安装完 `RPi.GPIO` 之后，只需要将电路连接到一个 GPIO 管脚。断开和 3.3v 引脚相连的线，并将其连接到第 22 角上。完成之后，打开一个 Python 会话（不要忘记 `sudo`！），并输入下列命令：

```
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(22, GPIO.OUT)
>>> GPIO.output(22, True)
>>> GPIO.output(22, False)
>>> GPIO.output(22, True)
>>> GPIO.output(22, False)
```

可以看到，将引脚置为 `True` 将点亮 LED，置为 `False` 将熄灭 LED。电路结构参见图 11-7。

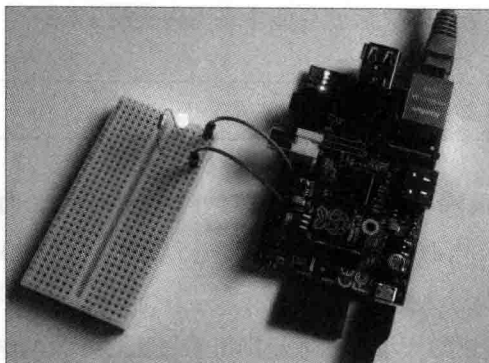


图 11-7 在小型无焊面包板上连接完成的电路

关于电路

基本上，一个电路包含三种要素——电源、负载和导电回路。电路通常都包含有这三个部分。如果只是将负载连接到电源上，而没有构成回路，负载将不会工作。如果直接将电源和地连接起来构成回路，将会发生短路，即产生强大的电流，并损坏电源（参见 11.3.2 节）。

电路结构有简有繁，从简单的电路（例如我们之前搭建的）到复杂系统（如计算机），都遵守同样的原理。

在本章中，限于篇幅我们只讨论如何使用树莓派的输入输出口，如果对电路有兴趣，可以参阅其他资料。

如果希望继续深入了解电路，Penguin Tutor 网站中提供了一个很好的教程，可以帮助你进一步理解它：www.penguintutor.com/electronics/。

11.3.2 保护树莓派

通常，很少出现由于编程导致的硬件物理损坏。编程可能损坏文件（这种情况同样也很少见），但通常情况下，无论将事情搞得多糟，重装系统一般都可以恢复。然而，一旦在 GPIO 口上连接了其他硬件，就有可能直接给 CPU 供电，也就可能损坏它。有两种电学特性可能会引起问题，电压和电流。

关于电压，规则简单也很重要。永远不能将 3.3V 电压连接到 GPIO 口上。这一点非常重要，也是我们要特别注意的地方。有很多电子器件都设计成使用 5V 电压，因为其他的处理器使用这个电压。然而，一但将其直接连接到树莓派上，将给板子带来无可挽回的

损失。结果就是变砖，因为损坏之后就只能拿来当砖头用了。



注意 永远不能在 GPIO 口上连接大于 3.3V 的电压。

注意到，树莓派有 5v 端口。它只是用来对外界提供电源，而不能用于自身。如果需要在 GPIO 口连接使用 5v 电源的设备，就需要一个逻辑电平转换器。大概几英镑就可以买到，用来将 5v 信号转换成 3.3v，反之亦然。

电压可以看做电力能量的大小，而电流则可看做流过导线的电力总量。欧姆定律将这两个量联系起来，即：

$$\text{电压} = \text{电流} \times \text{电阻}$$

或者，换一种形式：

$$\text{电流} = \text{电压} / \text{电阻}$$

电流的单位是安培 (A)，电压的单位是伏特 (V)，电阻的单位是欧姆 (Ω)。在前面的电路中，电压是 3.3V，电阻是 220Ω ，因此：

$$\text{电流} = 3.3 / 220 = 0.015\text{A} \quad \text{或者} \quad 15\text{mA}$$

树莓派的任何一个引脚最大只能接受 16mA 的电流，加到所有 GPIO 引脚的电流总和不能超过 50mA。这意味着同时只能点亮三个 LED（或者使用更大的电阻限制通过每个 LED 的电流），也就意味着除非和大于或等于 220 欧姆的电阻相连，否则不能直接将 GPIO 引脚接入电路。如果无法确定电阻大小，从安全角度考虑，推荐使用较大的电阻。

从技术方面看，上述说法并非完全正确，因为 LED 和其他原件有点不同。LED 两端会产生一个小的电压降。然而，在理解 LED 两端电压衰落的原理之前，还是先遵守上面的规则好了。

如果需要使用更大的电流，或者只是想保护树莓派免受瞬间大电流冲击，可以使用包含输入输出端口缓冲的扩展板，如 PiFace，或者使用集成缓冲电路（集成电路或者芯片）来保护 GPIO 接口。

11.3.3 电源限制

树莓派能够提供的总电流受限于它能接受的总电流。尤其是当树莓派上连接了很多像光电鼠标和 USB 存储器的其他设备时，有些供电口会尽可能地提供更多的电源。

如果在使用 GPIO 口时发现树莓派变得不稳定或者开始自动关机，电源不足可能是主要原因。

为了解决问题，可以升级电源适配器以获得更大电流，或者减少树莓派的输出总电流，如在 `raspi-config` 中减少超频或者移除不必要的外围设备。

11.3.4 获得输入

在上一个例子中，我们使用 GPIO 口来开关 LED，用到的是 GPIO 的输出功能。现在，我们来看下输入功能。继续使用全面的电路，不用拆除，但是需要添加一个按键。

按键开关是个很简单的原件。打开时（即没有按下）它的两个引脚是断开的，因此没有电流可以通过。按下时，它将会接通两个引脚，和电线一样开始导电。

和之前看到的电路一样，我们需要一个电阻来防止电路中出现过大的电流。为了更安全，这里我们只用了 $1.1\text{k}\Omega$ 的电阻。

然而，如果你只是将电源连接到电阻上，再连接按键，然后是 GPIO，当开关按下时，仍然不会构成电路。如果没有电路，GPIO 口也就不能感应到开或关（如果愿意，也称作 True 或 False）。当开关打开时我们需要一个电路将 GPIO 口拉低到地电平（即 False）。这就是著名的下拉电阻，因为在没有任何电路连接到 GPIO 口上时它将 GPIO 口拉低到 0V 。它需要远大于其他电阻以保证开关闭合时有足够的电势输入 GPIO 口。我们使用了 $6.2\text{k}\Omega$ 欧姆的电阻，当然 $10\text{k}\Omega$ 欧姆也可以。

图 11-8 给出了电路连接图。注意 GPIO 和地之间的电阻相对较大。如图所示，按键应该位于 3.3V 电源和 GPIO4 之间。

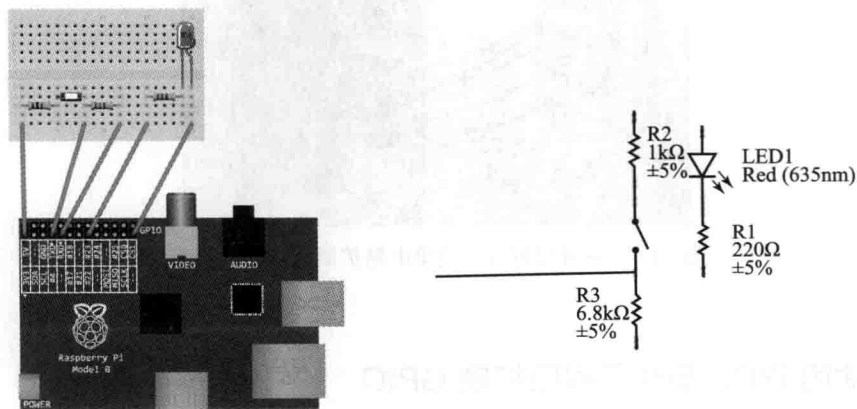


图 11-8 电路图，可以帮助你理解电路

设置好之后，下面的代码将创建一个简单的反应游戏。它将等待一个随机时间，然后点亮 LED。LED 点亮之后你需要迅速按下按键，然后它将给出你的反应时间。参见网站文件 `chapter11-reaction.py`。

图 11-9 给出了游戏运行的效果。

```
import RPi.GPIO as GPIO
import time
import random
from datetime import datetime

GPIO.setmode(GPIO.BCM)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(4, GPIO.IN)

GPIO.output(22, False)
random.seed()

while True:
    time.sleep(random.random()*10)
    start = datetime.now()
    GPIO.output(22, True)
    while not GPIO.input(4):
        pass
    print("Your reaction time: ",
          (datetime.now() - start).total_seconds())
    print("Get ready to try again.")
    GPIO.output(22, False)
```

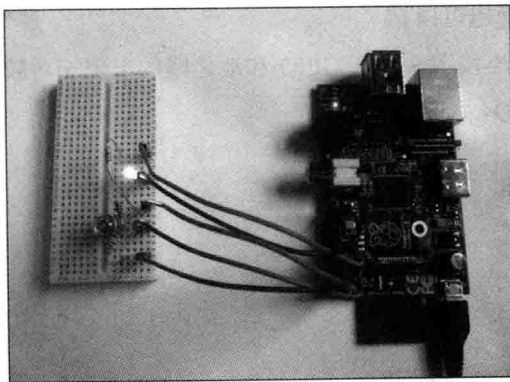


图 11-9 你可以将这个简单电路扩展成一个大工程

11.4 使用 I2C、SPI 和串口扩展 GPIO

两个计算机需要建立连接是，可以使用以太网。这是个系列标准，定义了很多事情，

如使用的物理线缆、计算机的网络寻址方式。只要计算机兼容以太网，它们就可以相互通信。

除此之外，还有一些为小尺寸硬件，如不同的芯片，制定的通信协议。在本章中我们只要介绍三种主要的协议——SPI、I2C 和串口。

11.4.1 SPI 通信协议

SPI 或者串行外设接口（Serial Peripheral Interface）使用 4 根线在两个或多个设备间来提供双向通信——一个主设备（通常是树莓派）和一个或多个从设备（典型代表就是芯片）。四根线包括用来保持设备同步的时钟线，主设备输出从设备输入线（MOSI），主设备输入从设备输出线（MISO），和从设备选择线^①。

下面我们开始将树莓派和从设备的相应引脚连接起来。

SPI 可以提供很多的扩展选则，例如模拟输出。

树莓派有很多输入和输出口，但这些都是数字口。也就是说，它们只能读出开或者关。有些时候已经够用了，如按键和控制 LED，但有时你需要量化地读写具体数值。例如，你可能需要从传感器，如光线或者温度传感器中读入数据。这些数据处于一个范围内，而不是开或关的数值。

这些（位于某个范围内的）值被称为模拟量（相对于数字量）。为了读取它们，你需要一个模拟数字转换器（ADC）。本章中，我们使用 MCP3008，它提供了 8 个模拟通道，和树莓派之间使用 SPI 通信。

图 11-10 给出了 MCP3008 的引脚说明，图 11-11 给出了电路连接示意图。



图 11-10 自上至下，芯片的左边 8 个接口分别表示 0~7 的模拟输入，右边 8 个接口用于芯片控制。在塑料封装的上面有一个半圆形缺口，表示芯片上端，在对照图片时需要对准该缺口

① SS，也称作片选线 CS。——译者注

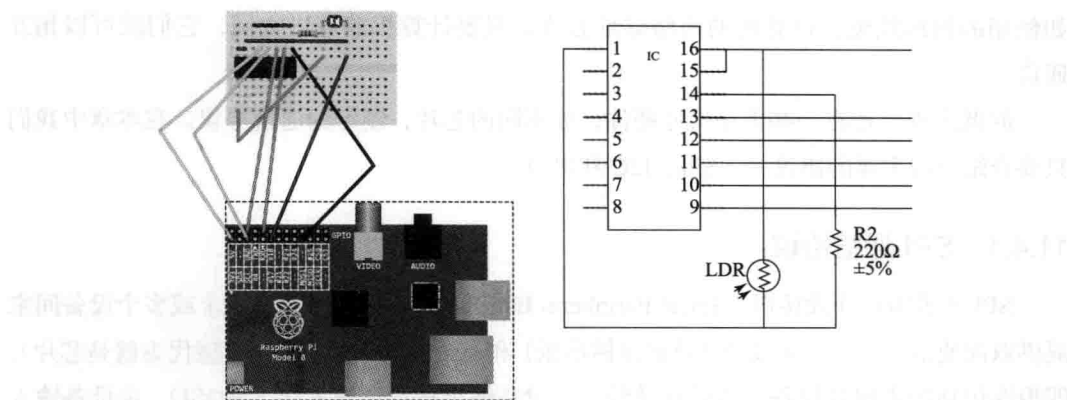


图 11-11 电路连线。改变 MCP3008 连接的 GPIO 口时需要在程序中修改相应的值

代码如下所示，参见网站文件 `chapter11-spiadc.py`。图 11-12 给出了连接好的电路。

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

# read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
def readadc(adcnun, clockpin, mosipin, misopin, cspin):
    if ((adcnun > 7) or (adcnun < 0)):
        return -1
    GPIO.output(cspin, True)

    GPIO.output(clockpin, False) # start clock low
    GPIO.output(cspin, False)    # bring CS low

    commandout = adcnun
    commandout |= 0x18 # start bit + single-ended bit
    commandout <= 3    # we only need to send 5 bits here
    for i in range(5):
        if (commandout & 0x80):
            GPIO.output(mosipin, True)
        else:
            GPIO.output(mosipin, False)
        commandout <= 1
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)

    adcout = 0
    # read in one empty bit, one null bit and 10 ADC bits
    for i in range(12):
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)
        adcout <= 1
        if (GPIO.input(misopin)):
            adcout |= 0x1
```

```

GPIO.output(cspin, True)

    adcout >= 1      # first bit is 'null' so drop it
    return adcout

SPICLK = 4
SPIMISO = 17
SPIMOSI = 18
SPICS = 23

# set up the SPI interface pins
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.set'up(SPICS, GPIO.OUT)

ldr_adc = 0;
last_read = 0
tolerance = 5

while True:
    # we'll assume that the light didn't change
    input_changed = False
    # read the analog pin
    ldr_value = readadc(ldr_adc, SPICLK,
                        SPIMOSI, SPIMISO, SPICS)
    ldr_movement = abs(ldr_value - last_read)

    if ( ldr_movement > tolerance ):
        input_changed = True

    if ( input_changed ):
        print('Light = ', int(ldr_value))
        last_read = ldr_value

    # hang out and do nothing for a half second
    time.sleep(0.5)

```

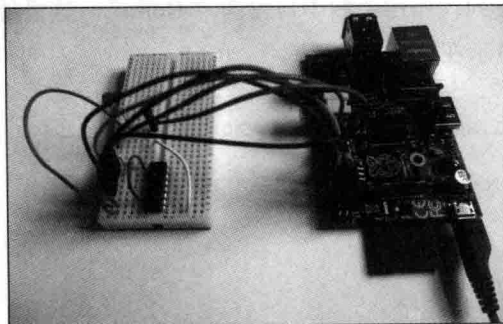


图 11-12 使用 MCP3008 将可变电阻 (LDR) 的模拟信号转换成数字信号输出到树莓派

不用太关心 `readadc()` 函数是如何工作的。它能操作单个字节并将其通过 MOSI 和 MISO 线传送和接收。如果希望增加几路输入，可以使用这个函数来读取 ADC 的其他端口。RPi.GPIO 库计划支持 SPI，但本书写作时，这部分还没有实现。可以通过下列网站查询项目进度 <http://code.google.com/p/raspberry-gpio-python/>。

主循环使用 `readadc()` 函数来获得与 ADC 相连的 LDR 值，并（如果数值变化超过门限）将其输出到屏幕上。

11.4.2 I2C 通信协议

内置集成电路（I2C 或者 I²C，发音为“I 方 C”）是个比 SPI 还要强大的协议。它仍然使用 4 根线，但其中一个电源线，另一根是底线，因此只有两根数据线。I2C 总线上最多可以连接 127 个设备，它具有寻址能力，而不是像 SPI 那样使用简单的从设备选择（片选）信号。

Quick2wire 制作了一系列的 I2C 板子用来扩展树莓派的功能。虽然它们可以简化处理过程，但是要使用 I2C 并不需要这些板子。它们还开发出了 Python 模块用于 I2C 通信，即使不适用它们的板子也可以使用该模块。模块可以从下面网站上找到 <https://github.com/quick2wire/quick2wire-python-api>。

和 SPI 类似，RPi.GPIO 有计划支持 I2C。请查看前面给出的网站获得最新信息。

11.4.3 串口通信协议

前面两种协议通常用于在设备间传输二进制数据。串口通信，则着手于另外一个方面，通常用于传输文本信息（当然这两者之间也有例外）。

`pyserial` 模块用于支持串口通信，可以通过下列命令获得该模块：

```
sudo pip install pyserial
```

`pyserial` 模块安装完成后，就可以创建串口连接，然后使用 `write()` 和 `read()` 方法来发送和接收数据。例如，下列代码用于向串口发送“Hello”。这里使用了 Ciseco PiLite，将会把接收到的字符滚动显示在屏幕上。

```
>>> import serial
>>> pilite = serial.Serial("/dev/ttyAMA0", baudrate=9600)
>>> pilite.open()
>>> pilite.write(bytes("Hello", "utf-8"))
```

11.5 深入研究

如果希望继续深入研究这个例子，尝试一下随后给出的想法。

11.5.1 Arduino

如果业余喜欢电子制作，最有用的工具套件可能就是 Arduino。它拥有比树莓派略大的微控制器板。其中含有更多的 GPIO 口（具体数目和模型有关），并且还有模拟输入，有些情况下还有模拟输出。

当然，Arduino 最有用的可能还是可以插入主板的扩展板（称为 Shield）。使用它们可以用少量的连线快速搭建出强大的硬件。

树莓派的 GPIO 和 Arduino 之间可以使用 I2C 或者 SPI 连接起来。不过大多数 Arduino 模型使用 5v 电压，连接时需要使用逻辑电平转换器。它们也可以通过 USB 使用串行连接通信。

有人认为连接到树莓派上之后，Arduino 的功能很难发挥出来。事实上，几乎所有在 Arduino 上完成的工作都可以在树莓派上完成。然而，大量已存的 Arduino 硬件和代码使得它们成为对树莓派非常有用的伙伴。

然而这些代码使用的是 C++ 而不是 Python，这可能是对本书读者最大的不利条件。该语言的格式和 Python 不同，但其基本理论是一致的。虽然我们在这里没有打算介绍它，但如果深入阅读过相关书籍，理解基本的 C++ 应该不成问题。

11.5.2 PiFace

PiFace 是一个可以直接插入树莓派 GPIO 口的扩展板，对于扩展 GPIO 非常有用。不仅可以缓冲输入输出（保护树莓派并提供 500mA 电流），还提供一组中继用于驱动更耗电的元件比如马达。

它的尺寸和树莓派相同，可以直接叠在树莓派上方。用做简单的机器人是个很好的选择。个人购买该元件并不便宜，大约 25 英镑（折合 41 美元），这和它的易用性和用于支持的工程有关。图 11-13 展示了 A 型树莓派连接上 PiFace 之后的效果

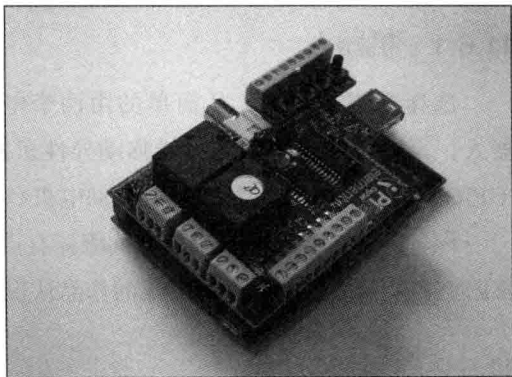


图 11-13 使用 PiFace 的输出缓冲可以控制更多东西而不用担心树莓派的小电流限制

11.5.3 Gertboard

Gertboard 是扩展树莓派 GPIO 口的巨人。它在一个板子上塞满了所有你想要的东西。

它拥有类似 ATmega 芯片的微控制器可以执行板外程序，马达控制器、数模转换器、模数转换器、按键开和一打 LED。当然，这些都需要付出成本，不只是价格（约合 60 英镑 /98 美金），还包括尺寸。由于尺寸太大，很多项目并不适合使用该扩展模块（例如机器人）。

对于学习电子和控制来说，这块板子很合适。一旦有了好主意，可以立即搭建电路实践自己的工程。

这块板子由树莓派基金会硬件最高领导人 Gert Van Loo 设计，完全值得信任，因为设计它的是真正从内到外（正确地）了解树莓派的人。

11.5.4 Wireless Inventor's Kit

到现在为止，我们所做的一切都需要线缆来连接各个组件。对于大多数项目来说都没问题。然而，有些项目需要更自由点。Ciseco 集成了 Wireless Inventor's Kit，其中包含了使用树莓派做简单无线通信需要的所有东西。

它使用的无线系统工作在比 WiFi 网络低的频率，可以用作传感器和树莓派间的无线传输，或者使用树莓派无线控制电路。

11.6 尝试一些流行工程

使用树莓派和现成的组件几乎可以做成任何东西，为了帮助你起步，接下来小节给出了一些想法。

11.6.1 机器人

选择范围非常大，从简单的由两个电机驱动的轮式设备，到复杂的拟人行走的机器人，再到像蛇或者蜘蛛一样移动异性机器人。开始制作机器人先要学习如何控制电机（GPIO 口提供的电能不足以直接驱动电机）。PiFace 有一对继电器，可以从那里开始。

你可能还需要一些伺服器。伺服器有点类似于电机，但可以按角度转动。还可以在树莓派上安装很多传感器来帮助你的作品认识周围的世界。

11.6.2 家庭自动化

想象一下可以使用智能手机控制暖气和灯光的世界会是什么样子。这不再是个不可实现梦想——使用树莓派、一些电路和 Python 就可以完成。第 7 章展示了如何通过互联网控制 Python 程序，再结合本章学到的知识，你就可以梦想变为现实。

11.6.3 防盗报警器

这个也许没有家庭自动化那么吸引人，但它能保证你的安全。使用树莓派摄像头和一些像被动红外（PIR）移动传感器之类的东西，你可以建造自己的 Fort Knox[⊖]。

11.6.4 数字艺术

艺术不再是在脏暗的房间里无生气地绘画，也不再局限于雕塑、诗歌或者舞蹈。它可以是任何你需要的形式。其中一个新兴形式就是数字艺术。例如，添加变幻的光线，或者压力感应平板电脑，当人们触摸它时会产生反馈。发散一下思维，然后使用树莓派为你的想象植入生命。

11.7 小结

读完本章，你应该更好地理解以下内容：

- 树莓派角落上突出的接口是通用输入输出（GPIO），在 Python 中可以对其编程。
- RPi.GPIO 库为开关 GPIO 口或者从中读取数值提供了一个简单的接口。
- 使用 GPIO 时需要格外小心，因为输入电压或电流过大会不可逆地损坏树莓派。
- 可以使用 SPI、I2C 和串口添加设备扩展树莓派，
- MCP3008 芯片就是一个例子，它提供了八路模数转换通道，可以使用 SPI 读取数据。
- 树莓派几乎可以控制任意复杂的电路，唯一的限制就是你的想象力。

⊖ 美国北部军用地。——译者注

测试与调试

很多时候代码运行的结果和预期的不一样。检查输出，走读代码，但有时还是找不出异常输出的原因。这是编程中遇到的最让人恼火的事情。

12.1 通过打印变量调查故障

有很多方法可以找出究竟发生了什么问题，但最简单的方法就是使用 `print()` 语句。使用该语句将每个变量值打印出来，通常可以找出底层发生的问题。

下面一段代码建立了一个简单的菜单系统。运行时没有报错，但无论输入什么，它都提示 “Unkown choice”。（参见网站文件 `chapter12-debug.py`。）

```
choices = {1:"Start", 2:"Edit", 3:"Quit"}
for key, value in choices.items():

    print("Press ", key, " to ", value)

user_input = input("Enter choice: ")

if user_input in choices.values():

    print("You chose", choices[user_input])
```

```
else:
```

```
    print("Unknown choice")
```

你可能已经看出问题在哪里，但如果还没有发现问题，怎样才能最好地找到它呢？现在的问题是当 `user_input` 有效时，`if` 语句没有正确地判断出来，因此我们添加一些 `print()` 语句来看看究竟发生了什么：

```
choices = {1:"Start", 2:"Edit", 3:"Quit"}
```

```
for key, value in choices.items():
```

```
    print("Press ", key, " to ", value)
```

```
user_input = input("Enter choice: ")
```

```
print("user_input: ", user_input)
```

```
print("choices: ", choices)
```

```
print("choices.values(): ", choices.values())
```

```
if user_input in choices.values():
```

```
    print("You chose", choices[user_input])
```

```
else:
```

```
    print("Unknown choice")
```

问题马上就出现了：`choices.values()` 应该换成 `choices.keys()`。代码用错了 `choices` 字典中的字段。修改代码然后运行。问题解决之后，程序应该能够正常运行。

然而，它还是不工作。一定还有其他错误。再看一下 `print` 语句的输出：

```
user_input: 1
```

```
choices: {1: 'Start', 2: 'Edit', 3: 'Quit'}
```

```
choices.values(): dict_values(['Start', 'Edit', 'Quit'])
```

你能看出为什么失败了吗？除了变量的值，第二个关键问题是值的数据类型，因此我们来扩展 `print` 语句，在其中加入更多细节：

```

print("user_input: ", user_input)

print("choices: ", choices)

print("choices.values(): ", choices.values())

print("type(user_input): ", type(user_input))

for key in choices.keys():

    print("type(key): ", type(key), "key: ", key)

```

如果你运行这些代码，应该会输出：

```

Press 1 to Start

Press 2 to Edit

Press 3 to Quit

Enter choice: 1

user_input: 1

choices: {1: 'Start', 2: 'Edit', 3: 'Quit'}

choices.values(): dict_values(['Start', 'Edit', 'Quit'])

type(user_input): <class 'str'>
type(key): <class 'int'> key: 1

type(key): <class 'int'> key: 2

type(key): <class 'int'> key: 3

Unknown choice

```

现在可以看出问题在于 `user_input` 是一个字符串，但是 `choices` 中的 `keys` 是整数。最简单的解决办法就是改变 `choices`，将其中的 `keys` 变成字符串类型：

```
choices = {"1": "Start", "2": "Edit", "3": "Quit"}
```

现在程序的基本逻辑已经工作正常；然而，它会输出一些用户不希望看到的文字。显然，删除 `print()` 语句可以解决问题，但是将来可能还会用到它们。可以将它们留下来，使用一个标记来开关它们，例如：

```
debug = True
```

```
choices = {"1": "Start", "2": "Edit", "3": "Quit"}
```

```

for key, value in choices.items():
    print("Press ", key, " to ", value)

user_input = input("Enter choice: ")

if debug:
    print("DEBUG user_input: ", user_input)

    print("DEBUG choices: ", choices)

    print("DEBUG choices.values(): ", choices.values())
    print("DEBUG type(user_input): ", type(user_input))

    for key in choices.keys():
        print("DEBUG type(key): ", type(key), "key: ", key)

if user_input in choices.keys():
    print("You chose", choices[user_input])
else:
    print("Unknown choice")

```

将来遇到再任何问题，只需要将 `debug` 变量改为 `True`。打印出来的文字前面带的 `DEBUG` 可以明确地将调试输出和正常输出区分开。

12.2 通过测试发现故障

调试是一个修正程序存在的问题的过程。这是个很有挑战性的工作，而发现第一处问题尤为困难。听起来有点不可思议，但是事实就是这样。随着程序不断变大，它的应用场景也开始增加，应用场景增加后，遇到问题的地方也会增加。

想象一下，例如，一个字处理程序拥有样式、页面布局、文件格式、布局管理等功能，每个部分都可能存在潜在的问题。因此对于开发者来说，保证每个功能都能够正常运行非常重要。问题也可能隐藏在各模块交界处。例如，可能只有在某个排列下使用了某个字体才有问题。

12.2.1 使用单元测试检查代码片段

单元测试是最基本的测试程序，是指对代码中小片段进行检查和验证。通常用来验证单个方法以保证功能正常。

本质上来说，所有的单元测试都是使用一组特定的输入来运行一段代码，然后检查输出是否正确。

例如，一个将输入的字符串转换成大写字母并返回的函数，可以这样实现并测试：

```
def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.isupper():
            output_string = output_string + character
        else:
            output_string = output_string + chr(ord(character)-32)
    return output_string

print(capitalise("helloWorld"))
```

程序能够正常工作，是因为 Python 使用和数字相同的存储方式来存放 UTF-8 的字符，大写字母在 ASCII 表中比小写字母靠前 32 个位置。

在这个例子中，我们使用一个简单的测试用例，将结果打印到屏幕上，接受人工检查。我们也可以像下面的例子一样使用 unittest 模块让 Python 替我们检查结果。

```
import unittest

def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.isupper():
            output_string = output_string + character
```

```

else:
    output_string = output_string + chr(ord(character)-32)
return output_string

class Tests(unittest.TestCase):
    def test_1(self):
        self.assertEqual("HELLOWORLD", capitalise("helloWorld"))

if __name__ == '__main__':
    unittest.main()

```

这段代码和前面的代码差不多。运行之后，它将会检查字符串“helloWorld”是否变成“HELLOWORLD”。在这个水平上看，它和将结果输出到屏幕上没什么区别。

运行 `unittest.main()` 时，Python 将运行子类 `unittest.TestCase` 中的所有以 `test_` 开头的方法。在这个例子中，它将运行 `test_1`。使用单元测试的优势在于可以将一组测试组合起来运行，能够很快检查出程序是否正常工作。

你可以再添加一个测试用例来检查字符串“hello world”是否变成大写的“HELLO WORLD”：

```

def test_2(self):
    self.assertEqual("HELLO WORLD", capitalise("Hello World"))

```

运行之后会得到下列输出：

```
FAIL: test_2 (__main__.Tests)
```

```
-----
Traceback (most recent call last):
```

```
File "capitalise.py", line 17, in test_2
```

```
    self.assertEqual("HELLO WORLD", capitalise("Hello World"))
```

```
AssertionError: 'HELLO WORLD' != 'HELLO\x00WORLD'
```

```
- HELLO WORLD
```

```
?
```

```
+ HELLOWORLD
```

```
?
```

看起来测试失败了。可以看到对空格的处理出错了。回头看下代码，我们发现问题出在我们从字符串内所有的字符包括非小写字母的 UTF-8 值中减了 32。因为空格不是小写字母，本不应该处理，但仍然被减了 32。

设计测试用例时应当尝试使用大范围的有效输入。例如：

```
class Tests(unittest.TestCase):
```

```
    def test_1(self):
```

```
        self.assertEqual("HELLOWORLD", capitalise("hello World"))
```

```
    def test_2(self):
```

```
        self.assertEqual("HELLO WORLD", capitalise("Hello World"))
```

```
    def test_3(self):
```

```
        self.assertEqual('!"f$%^&*()_+==',
                           capitalise('!"f$%^&*()_+=='))
```

```
    def test_4(self):
```

```
        self.assertEqual("1234567890", capitalise("1234567890"))
```

```
    def test_5(self):
```

```
        self.assertEqual("HELLO WORLD", capitalise("HELLO WORLD"))
```

```
    def test_6(self):
```

```
        self.assertEqual("`~#~'@;:,.<>/?",
                           capitalise("`~#~'@;:,.<>/?"))
```

运行之后会发现更多错误。问题就出在程序的处理逻辑中。程序中不处理大写字母，而处理了所有非大写字母。然而，实际上你期望的是只处理小写字母，不处理其他字符。

如果按照下列代码修改 `capitalise` 函数，它就可以正常工作了：

```
def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.islower():
            output_string = output_string + chr(ord(character)-32)
        else:
            output_string = output_string + character
    return output_string
```

再次运行程序会发现所有的测试都通过了。

默认情况下，只有当测试失败时 `unittest` 才会给出详细结果。否则，它只会返回测试通过。大多数情况下它都能够满足你的需求，但它还提供了一个选项用来输出更多信息。有两种方式可以设置这个选项。如果在命令行中运行脚本，可以添加 `-v` 选项来获得更多输出。例如，你已经将程序保存为 `capitalise.py`，可以用下面的命令运行它以获得更多信息：

```
python3 capitalise.py -v
```

同样，你也可以在程序中添加该选项。例如可以将下列代码：

```
if __name__ == '__main__':
    unittest.main()
```

改为：

```
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

在继续下面内容之前，我们需要指出，这里的 `capitalise()` 函数只是用来举例的。如果需要将字符转换成大写，应该使用 `string` 类中的 `upper()` 函数。例如：

```
>>> 'hello world'.upper()
```

12.2.2 获得更多断言

所有这些测试都调用了 `self.assertEqual()`。这行语句告诉单元测试模块期望的测试结果是什么。也就是这两个值相等的时候测试才能通过，如果不相同，则测试失败。它在很

多测试用例中都能用到，但你可能还希望检查一些其他东西。除此之外，还有很多不同的 `assert` 方法可以在测试中。

下面这些用于检查各种结构体是否相同：

```
■ assertSequencesEqual(sequence1, sequence2)
■ assertListEqual(list1, list2)
■ assertTupleEqual(tuple1, tuple2)
■ assertSetEqual(set1, set2)
■ assertDictEqual(dict1, dict2)
```

在这些结构体中，可能你还希望检查某个值是否属于该结构体而不仅仅是比较两个结构体是否相同。下列方法用于检查某个值是否属于结构体：

```
■ assertIn(value, structure)
■ assertNotIn(value, structure)
```

字符串是一个特殊的结构体，它有自己的方法：

```
■ assertMultiLineEqual(string1, string2)
```

使用下列方法可以检查除相等之外的其他逻辑：

```
■ assertNotEqual(value1, value2)
■ assertGreater(value1, value2)
■ assertGreaterEqual(value1, value2)
■ assertLess(value1, value2)
■ assertLessEqual(value1, value2)
```

还有一些方法允许适当的误差：

```
■ assertAlmostEqual(value1, value2)
■ assertNotAlmostEqual(value1, value2)
```

这两个语句用于检测两个值相差是否超过 0.000001。它们在测试浮点数程序时很有用，因为它们可以接受适当的舍入误差。

下面的语句可以用来测试任何可以缩减成 `True` 或 `False` 东西：

```
■ assertTrue(value)
■ assertFalse(value)
```

无论你想检查什么东西，每个 `test_` 函数都应该调用一个 `assert` 方法，以判断测试用例是否成功。

你可以在许多方面使用单元测试。有一种称为测试驱动开发的开发方式，它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。然而，大多数程序员都在开发完成之后编写测试用例来检测代码是否能够正常工作。

12.2.3 使用测试集进行回归测试

编写程序通常都不是一蹴而就的。并不是坐下来，写一段代码然后就可以停下来去做别的事情了。通常都是实现了某些特性之后交给客户使用，然后在下一个版本中修改错误并添加新的特性。

在添加新特性时可能破坏原来的东西，因此不仅需要测试新添加的功能，还需要保证旧功能可以工作。测试旧代码称为回归测试。拥有一个合适的测试集可以让回归测试变得很轻松。

为了保证在原来可以工作的代码中不引入错误，修改完成之后需要重新进行测试。然而，随着程序越变越大，所需的测试也会越来越多。最终，你会发现不可能一次运行所有的测试。这时可以将测试分成不同的测试集。一次只需运行几个相关的测试即可。

继续使用上一个例子，你可以将程序改成：

```
if __name__ == '__main__':

    letters_suite = unittest.TestSuite()
    symbols_suite = unittest.TestSuite()

    letters_suite.addTest(Tests("test_1"))

    symbols_suite.addTest(Tests("test_2"))
    symbols_suite.addTest(Tests("test_3"))
    symbols_suite.addTest(Tests("test_4"))
    symbols_suite.addTest(Tests("test_5"))
    symbols_suite.addTest(Tests("test_6"))

    all_suite = unittest.TestSuite()
    all_suite.addTest(letters_suite)
    all_suite.addTest(symbols_suite)

    unittest.TextTestRunner(verbosity=2).run(all_suite)
```

这段代码和前面的代码所做的工作一样，即运行所有的测试用例。然而它将测试用例分成不同的测试集。`letters_suite` 用来运行检查字符的是测试，`symbols_suite` 用来运行所有检查符号的测试，`all_suite` 运行这两个测试集。你可以使用最后一行来运行这三个测试集。

使用上述代码，你会发现可以轻易地建立一个简单的测试菜单，以保证所有功能都能正常运行。

12.2.4 测试整个程序包

将单元测试自动化了之后可以快速方便地检查所有功能是否正常运行。然而，它并不能够覆盖所有内容。即使看起来所有东西都能正常工作，当你将它们整合起来之后还是能发现问题。

在商业软件开发中，单元测试结束之后，代码将被传递给质量保证组以确保各部分能够正常工作。这个小组将从软件使用角度给出一系列测试用例。这里例子用于检查程序的各个方面并使用一组输入来测试程序，确保它能正常工作。这项工作又是通过测试人员手工模拟用户操作进行测试，又是使用专用的测试软件模拟鼠标和键盘输入。

当然，现在你还不可能有一个质量保证小组来帮助你保证软件质量，但从专业方法来看，你可以借鉴一些内容。你需要一个系统化的方法。在开始测试前，先列出程序的所有功能，包括测试输入和期望输出。然后对照列表一项项执行，以确保程序工作正常。

代码每改动一次就执行一遍测试有点过分，但你需要周期性执行它，或者至少在每次的大版本发布时测试一遍。

12.2.5 保证软件可用性

程序完成之后，你对所有内容都十分了解。你知道如何与之交互，如果获得最好的结果，以及如何使用各种选项。然而，你的用户并不知道这些内容。你的软件需要帮助它们理解这些内容，并为用户提供足够的信息，让用户知道如何使用它。毕竟，如果用户不知道如何使用它们，你的特性有多强大都没用。

用户测试就是指用户的使用性测试，用于检测软件实现是否符合自己预期的要求。理想情况下，你可以将用户集中到一个房间里，在用户面前摆上软件，并请他们执行某些任务，然后观察输出结果。然而，这在现实世界中是不可能的。有时你能够说服朋友或者相关人员来帮助你，但开发的程序越多，你会发现能找到的志愿者越少。唯一的解决办法就是聆听使用软件的人的声音，确保能够收到反馈。

12.3 究竟需要多少测试

关于软件缺陷，有条谚语：“没有证据并不是表示没有问题”。基本上无论你做了多少测试，都不能够证明程序中没有缺陷。事实上，几乎不可能写出没有缺陷的程序。测试的目的并不是保证软件没有缺陷，而是用来保证软件足够好。“足够好”在不同的项目中有不同的含义。越重要的软件，越需要测试，但所有的软件都至少应该有一点测试。实现新功能时并不需要多花哨，重要的是要保证足够的测试而不是留下一堆问题。因此，花时间写单元测试并保证每个部分正常工作是值得的。毕竟当程序出错时，损坏的是你的数据。

12.4 小结

读完本章，你应该更好地理解以下内容：

- 调试是在代码中发现并解决问题的过程。
- 正确地使用 `print()` 语句能够帮助你找出问题所在。
- 使用某种方法开关 `print()` 语句有时会很有用，在发现更多问题的时候可以重用这些语句。
- 测试是在代码中找出问题的过程。
- 单元测试是最基本的测试形式，使用 `unittest` 模块可以将其自动化。
- 测试用例可以分成不同的测试集，可以方便你测试程序的某一方面。
- 开发新特性时可能会引入新的问题，因此改过代码之后最好做一下回归测试。
- 单元测试并不能发现所有的问题，因此需要进行系统级别的测试。
- 可用性问题也是缺陷，因此你应该倾听用户反馈，以确保将程序使用方法正确地传达给它们。

至此，本书即将结束。但愿你现在已经有足够的知识和自信来创建自己的程序。如果感觉自己还没有完全掌握 Python 的特性也不用担心，很少有人能全部掌握。如果遇到不懂的问题，可以随时翻阅本书或者 Python 文档 <http://docs.python.org/3/>。

但愿你已经认识到编程并不复杂，如果将一个问题分成多个小步骤，编程就会变得很简单。最主要的是记住编程一个愉快的过程！找到自己感兴趣的领域并实践一下。尽管树莓派尺寸不大，却很少有它做不到的事情。

推荐阅读



Arduino高级开发权威指南 (原书第2版)

作者: Steven F. Barrett ISBN: 978-7-111-45246-1 定价: 59.00元



例说XBee无线模块开发

作者: Jonathan A. Titus ISBN: 978-7-111-45681-0 定价: 59.00元



Arduino与LabVIEW开发实战

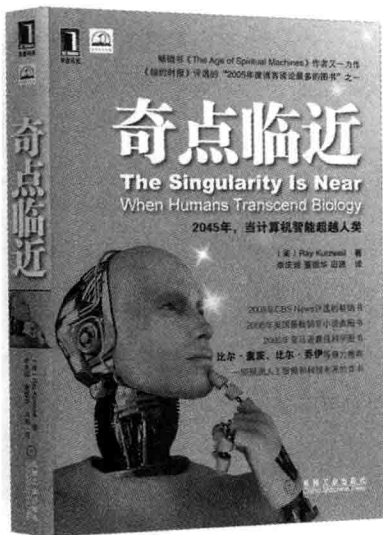
作者: 沈金鑫 ISBN: 978-7-111-45839-5 定价: 59.00元



Arduino开发实战指南: STM32篇

作者: 姚汉 ISBN: 978-7-111-44582-1 定价: 59.00元

推荐阅读



奇点临近

畅销书《The Age of Spiritual Machines》作者又一力作
《纽约时报》评选的“2005年度博客谈论最多的图书”之一

2005年CBS News评选的畅销书

2005年美国最畅销非小说类图书

2005年亚马逊最佳科学图书

比尔·盖茨、比尔·乔伊等鼎力推荐

一部预测人工智能和科技未来的奇书

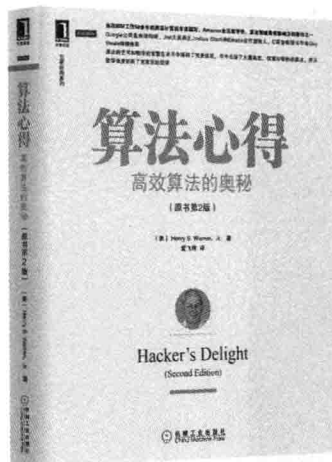
“阅读本书，你将惊叹于人类发展进程中下一个意义深远的飞跃，它从根本上改变了人类的生活、工作以及感知世界的方式。库兹韦尔的奇点是一个壮举，以不可思议的想象力和雄辩论述了即将发生的颠覆性事件，它将像电和计算机一样从根本上改变我们的观念。”

—— 迪安·卡门，物理学家

“本书对科技发展持乐观的态度，值得阅读并引人深思。对于那些像我这样对“承诺与风险的平衡”这一问题的看法与库兹韦尔不同的人来说，本书进一步明确了需要通过对话的方式来解决由于科技加速发展而引发的诸多问题。”

—— 比尔·乔伊，SUN公司创始人，前首席科学家

推荐阅读



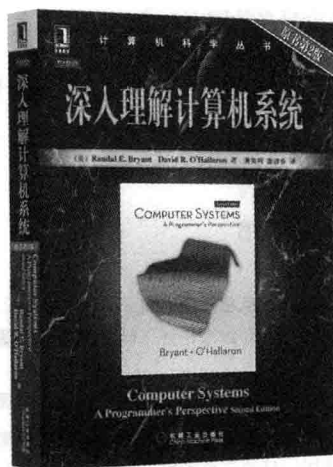
算法心得：高效算法的奥秘（原书第2版）

作者：Henry S. Warren ISBN: 978-7-111-45356-7 定价：89.00元



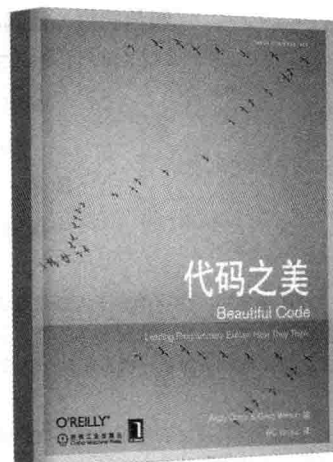
算法导论（原书第3版）

作者：Thomas H. Cormen 等 ISBN: 978-7-111-40701-0 定价：128.00元



深入理解计算机系统（原书第2版）

作者：Randal E. Bryant 等 ISBN: 978-7-111-32133-0 定价：99.00元



代码之美

作者：Grey Wilson ISBN: 978-7-111-25133-0 定价：99.00元

本书由树莓派基金会资深软件开发工程师亲笔撰写，是学习在树莓派上编程的必备手册。即使你没有任何编程经验，也可以畅游树莓派的世界。本书覆盖了初学编程者和第一次做Python开发所需的基础知识，书中首先对Python编程做了基本介绍，并给出了通用的Python代码，然后逐步介绍了：

- 配置并开始Python编程
- 使用变量、循环和函数
- 学习3D图形编程
- 使用PyGame
- 编程“我的世界”游戏
- 编写Python脚本
- 理解传感器和GPIO

本书深入浅出地介绍每条命令，并辅以生动的例子和源代码，可以帮助你学习Python编程所需的各种知识和技能，是开启Python编程之旅的必备指南。

读者可以登录华章网站（www.hzbook.com）或访问<http://www.wiley.com/go/pythonraspi>下载本书源代码。

WILEY

www.wiley.com

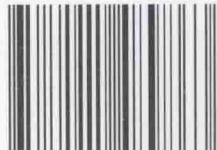


投稿热线: (010) 88379604
客服热线: (010) 88378991 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn

上架指导: 计算机/程序设计

ISBN 978-7-111-48986-3



9 787111 489863 >

定价: 59.00元

PDF电子书说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 **QQ: 461573687**, 或者 **QQ: 2404062482**。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！