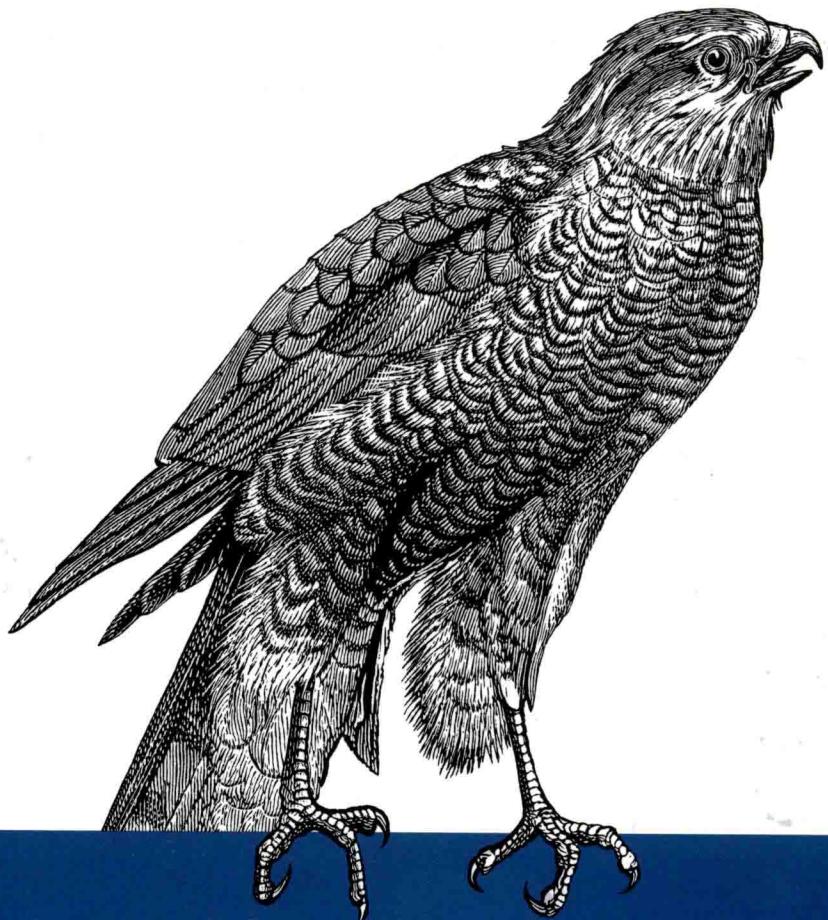


O'REILLY®

异步图书
www.epubit.com.cn

第2版



树莓派开发实战

Raspberry Pi Cookbook

[英] Simon Monk 著

韩波 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

树莓派开发实战（第2版）

[英] Simon Monk 著
韩 波 译

人民邮电出版社
北京

图书在版编目 (C I P) 数据

树莓派开发实战 : 第2版 / (英) 蒙克
(Simon Monk) 著 ; 韩波译. -- 北京 : 人民邮电出版社,
2017.5

ISBN 978-7-115-44720-3

I. ①树… II. ①蒙… ②韩… III. ①软件工具—程序设计 IV. ①TP311. 561

中国版本图书馆CIP数据核字(2017)第035096号

版权声明

Copyright© 2016 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2017.

Authorized translation of the English edition, 2016 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书中文简体版由 O'Reilly Media, Inc. 授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

-
- ◆ 著 [英]Simon Monk
 - 译 韩 波
 - 责任编辑 胡俊英
 - 责任印制 焦志炜
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
 - 邮编 100164 电子邮件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 三河市海波印务有限公司印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 27.5
 - 字数: 560 千字 2017 年 5 月第 1 版
 - 印数: 1 - 2 500 册 2017 年 5 月河北第 1 次印刷
 - 著作权合同登记号 图字: 01-2016-7577 号
-

定价: 89.00 元

读者服务热线: (010) 81055410 印装质量热线: (010) 81055316

反盗版热线: (010) 81055315

广告经营许可证: 京东工商广字第 8052 号

内容提要

树莓派（Raspberry Pi）是一款基于 Linux 系统的、只有一张信用卡大小的卡片式计算机。研发树莓派的最初目的是通过低价硬件和自由软件来推动学校的基础计算机学科教育，但很快树莓派就得到计算机和硬件爱好者的青睐；他们用它学习编程，并创造出各种各样新奇的、风靡一时的软硬件应用。

本书由多产作家 Simon Monk 编写，是将内容进行全面升级之后的第 2 版，囊括了丰富的实践示例，详细讲解了树莓派的配置与管理、网络连接、操作系统及软件。另外，本书还介绍了使用 Python 进行树莓派开发的各项技巧，书中介绍了有关 Python 编程的基础知识、列表与字典、Python 高级特性。同时，本书还介绍了有关机器视觉、硬件基础、控制硬件、电机、数字输入、传感器、显示设备、物联网及 Arduino 等相关知识。

本书适合程序员和计算机软、硬件爱好者，以及对树莓派感兴趣的读者阅读，也适合作为树莓派相关实践课程的指导用书。

第2版前言

自从 2011 年诞生以来，树莓派已经成为基于 Linux 的低成本电脑和嵌入式计算平台这两个领域中的重要角色。同时，也受到了教育工作者和业余爱好者们的一致好评。

自从本书第 1 版问世以来，树莓派的销售量已超过了几百万台，同时还出现了许多新型号的树莓派。某些型号，比如 B+、A+ 和 B+ 型树莓派 2 等，还对该设备的规范进行了改进，出现了具有 4 核处理器的树莓派 2 和树莓派计算模块，即可以把树莓派作为一个插件式电路板用作更大系统的零部件。

针对出现的各种新型树莓派以及 Raspbian 操作系统的各种变化和改进，本书第 2 版进行了全面的升级。

此次改版，本书新增加了两章，其中有一章是关于机器视觉的，另一章是介绍利用树莓派建立物联网项目的。

本书内容经过了精心编排，你既可以像阅读常规图书一样地进行线性阅读，也可以随机查阅各种示例。同时，你还可以通过目录或索引查找自己感兴趣的内容，然后进行跳跃式的阅读。如果你所阅读的内容涉及其他章节，那么会给出相应的提示，以便于你查阅有关内容。这与常见的菜谱不同，它们的做法是首先介绍一系列调味料，然后才会介绍烹饪方法。

树莓派的世界日新月异。因为树莓派有一个活跃的大型社区，所以新型的接口板和软件库会源源不断地涌现。本书除了提供大量特定接口板和软件的使用示例之外，还提供了相应的基础理论，以便帮助读者更透彻地理解如何利用树莓派生态系统中不断发展的各种新技术。

如你所料，本书提供了大量示例代码（大部分为 Python 程序）。这些程序全部都是开源的，并且可以从 GitHub 下载。你可以从本书网站 (<http://www.raspberrypicookbook.com>)

目录

第1章 配置与管理	1
1.0 引言	1
1.1 选择树莓派型号	1
1.2 封装树莓派	3
1.3 选择电源	4
1.4 选择操作系统发行包	6
1.5 通过 NOOBS 刷写 microSD 卡	7
1.6 装配系统	9
1.7 连接 DVI 或 VGA 显示器	10
1.8 使用复合视频显示器/TV	10
1.9 调整显示器中的图像尺寸	12
1.10 优化性能	13
1.11 修改密码	15
1.12 让树莓派开机后直接进入窗口系统	16
1.13 关闭树莓派	17
1.14 安装树莓派摄像头模块	18
1.15 使用蓝牙设备	21
第2章 网络连接	23
2.0 引言	23
2.1 连接有线网络	23
2.2 查看自己的 IP 地址	25
2.3 配置静态 IP 地址	26
2.4 为树莓派配置网络名称	28
2.5 配置无线网络连接	29
2.6 使用控制台线联网	31

2.7	利用 SSH 远程控制树莓派	33
2.8	利用 VNC 远程控制树莓派	35
2.9	利用 RDP 远程控制树莓派	36
2.10	在 Mac 网络中实现文件共享	38
2.11	在 Mac 上共享树莓派的屏幕	39
2.12	将树莓派用作网络存储服务器	41
2.13	网络打印	44
第 3 章	操作系统	47
3.0	引言	47
3.1	通过图形界面移动文件	47
3.2	启动一个终端会话	49
3.3	利用终端浏览文件系统	50
3.4	复制文件或文件夹	52
3.5	重命名文件和文件夹	53
3.6	编辑文件	54
3.7	查看文件内容	56
3.8	不借助编辑器的情况下创建文件	57
3.9	创建目录	57
3.10	删除文件或目录	58
3.11	以超级用户权限执行任务	59
3.12	理解文件权限	60
3.13	修改文件的权限	61
3.14	修改文件的属主	62
3.15	屏幕截图	63
3.16	利用 apt-get 安装软件	64
3.17	删除利用 apt-get 安装的软件	65
3.18	利用 Pip 安装 Python 软件包	65
3.19	通过命令行获取文件	66
3.20	利用 Git 获取源代码	67
3.21	在系统启动时自动运行程序或脚本	67
3.22	让程序或脚本作为服务自动运行	68
3.23	定期自动运行程序或脚本	70
3.24	搜索功能	71
3.25	使用命令行历史记录功能	72
3.26	监视处理器活动	73
3.27	文件压缩	75
3.28	列出已连接的 USB 设备	76

3.29	将输出从命令行重定向到文件	76
3.30	连接文件	77
3.31	使用管道	78
3.32	将输出隐藏到终端	78
3.33	在后台运行程序	79
3.34	创建命令别名	80
3.35	设置日期和时间	80
3.36	查看 SD 卡剩余存储空间	81
第 4 章	软件	83
4.0	引言	83
4.1	搭建媒体中心	83
4.2	安装 Oice 软件	84
4.3	安装其他浏览器	86
4.4	使用树莓派商店	87
4.5	打造网络摄像头服务器	88
4.6	运行老式游戏控制台模拟器	90
4.7	运行树莓派版 Minecraft	92
4.8	运行 Minecraft 服务器	93
4.9	运行 Open Arena	95
4.10	树莓派无线电发射器	96
4.11	运行 GIMP	98
4.12	互联网广播	99
第 5 章	Python 入门	101
5.0	引言	101
5.1	在 Python 2 和 Python 3 之间做出选择	101
5.2	使用 IDLE 编辑 Python 程序	102
5.3	使用 Python 控制台	104
5.4	利用终端运行 Python 程序	105
5.5	变量	106
5.6	显示输出结果	107
5.7	读取用户输入	107
5.8	算术运算	108
5.9	创建字符串	109
5.10	连接（合并）字符串	110
5.11	将数字转换为字符串	111
5.12	将字符串转换为数字	111
5.13	确定字符串的长度	112

5.14 确定某字符串在另一个字符串中的位置	113
5.15 截取部分字符串	113
5.16 使用字符串替换另一个字符串中的内容	114
5.17 字符串的大小写转换	115
5.18 根据条件运行命令	116
5.19 值的比较	117
5.20 逻辑运算符	118
5.21 将指令重复执行特定次数	119
5.22 重复执行指令直到特定条件改变为止	120
5.23 跳出循环语句	120
5.24 定义 Python 函数	121
第 6 章 Python 语言中的列表与字典	123
6.0 引言	123
6.1 创建列表	123
6.2 访问列表元素	124
6.3 确定列表长度	125
6.4 为列表添加元素	125
6.5 删除列表元素	126
6.6 通过解析字符串创建列表	127
6.7 遍历列表	127
6.8 枚举列表	128
6.9 列表排序	129
6.10 分割列表	130
6.11 将函数应用于列表	131
6.12 创建字典	131
6.13 访问字典	133
6.14 删除字典元素	134
6.15 遍历字典	134
第 7 章 Python 高级特性	137
7.0 引言	137
7.1 格式化数字	137
7.2 格式化时间和日期	138
7.3 返回多个值	139
7.4 定义类	140
7.5 定义方法	141
7.6 继承	142
7.7 向文件中写入内容	143

7.8	读文件	144
7.9	序列化 (Pickling)	145
7.10	异常处理	146
7.11	使用模块	147
7.12	随机数	148
7.13	利用 Python 发送 Web 请求	149
7.14	Python 的命令行参数	150
7.15	从 Python 运行 Linux 命令	151
7.16	从 Python 发送电子邮件	152
7.17	利用 Python 编写简单 Web 服务器	153
7.18	同时进行多件事情	154
7.19	让 Python 无所事事	156
7.20	将 Python 应用于树莓派版 Minecraft	156
第 8 章	机器视觉	159
8.0	引言	159
8.1	安装 SimpleCV	159
8.2	为机器视觉配置 USB 摄像头	160
8.3	将树莓派的摄像头模块用于机器视觉	162
8.4	数硬币	163
8.5	人脸检测	166
8.6	运动检测	168
8.7	光学字符识别	170
第 9 章	硬件基础	173
9.0	引言	173
9.1	GPIO 连接器使用说明	173
9.2	使用 GPIO 接口时树莓派的安全保护	176
9.3	配置 I2C	176
9.4	使用 I2C 工具	178
9.5	配置 SPI	179
9.6	安装 PySerial 以便从 Python 访问串口	180
9.7	安装 Minicom 以检测串口	181
9.8	使用带有跳线的面包板	182
9.9	使用树莓派的排线连接面包板	183
9.10	使用树莓派 Squid	184
9.11	使用 Raspberry Squid 按钮	186
9.12	利用两个电阻将 5V 信号转换为 3.3V	187
9.13	利用电平转换模块将 5V 信号转换为 3.3V	189

9.14	利用电池为树莓派供电	190
9.15	利用锂电池为树莓派供电	192
9.16	Sense HAT 入门指南	193
9.17	Explorer HAT Pro 入门指南	194
9.18	RaspiRobot Board 入门指南	196
9.19	使用 Pi Plate 原型板	198
9.20	制作树莓派扩展板（Hardware At Top, HAT）	201
9.21	树莓派的计算模块	204
9.22	Pi Zero	205
第 10 章	控制硬件	207
10.0	引言	207
10.1	连接 LED	207
10.2	让 GPIO 引脚进入安全状态	209
10.3	控制 LED 的亮度	210
10.4	制造蜂鸣声	213
10.5	利用晶体管开关大功率直流设备	214
10.6	使用继电器控制大功率设备的开关	216
10.7	控制高压交流设备	219
10.8	编写用于控制开关的用户界面	220
10.9	编写控制 LED 和电机的 PWM 功率的用户界面	221
10.10	改变 RGB LED 的颜色	223
10.11	使用大量 LED（Charlieplexing 方式）	225
10.12	将模拟仪表用作显示器	228
10.13	中断编程	230
第 11 章	电机	233
11.0	引言	233
11.1	控制伺服电机	233
11.2	精确控制伺服电机	237
11.3	控制多台伺服电机	239
11.4	控制直流电动机的速度	242
11.5	控制直流电机的方向	244
11.6	使用单极步进电机	248
11.7	使用双极步进电机	252
11.8	利用步进电机 HAT 驱动双极步进电机	253
11.9	使用 RaspiRobot 板驱动双极步进电机	256
11.10	打造一款简单的机器人小车	257

第 12 章 数字输入	261
12.0 引言	261
12.1 连接按钮开关	261
12.2 通过按钮开关切换开关状态	264
12.3 使用双位拨动开关或滑动开关	265
12.4 使用三位拨动开关或滑动开关	266
12.5 按钮去抖	269
12.6 使用外部上拉电阻	271
12.7 使用旋转（正交）编码器	272
12.8 使用数字键盘	275
12.9 检测移动	278
12.10 为树莓派添加 GPS	280
12.11 拦截按键	283
12.12 拦截鼠标移动	285
12.13 使用实时时钟模块	286
第 13 章 传感器	291
13.0 引言	291
13.1 使用电阻式传感器	291
13.2 测量亮度	296
13.3 利用热敏电阻测量温度	298
13.4 检测甲烷	301
13.5 测量电压	304
13.6 为测量而降低电压	306
13.7 使用电阻式传感器与 ADC	309
13.8 使用 ADC 测量温度	310
13.9 测量树莓派的 CPU 温度	312
13.10 利用 Sense HAT 测量温度、湿度和气压	313
13.11 利用数字传感器测量温度	315
13.12 利用 MCP3008 模块测量加速度	318
13.13 使用 Sense HAT 的惯性管理单元（IMU）	321
13.14 利用 Sense HAT 寻找磁北	323
13.15 利用簧片开关检测磁铁	323
13.16 利用 Sense HAT 感应磁场	325
13.17 测量距离	325
13.18 电容式触摸传感技术	328
13.19 显示传感器的值	331
13.20 利用 USB 闪存驱动器记录日志	332

第 14 章 显示设备	335
14.0 引言	335
14.1 使用四位 LED 显示设备	335
14.2 在 I2C LED 矩阵上面显示消息	337
14.3 使用 Sense HAT LED 矩形显示器	339
14.4 在 Alphanumeric LCD HAT 上显示消息	341
14.5 在 Alphanumeric LCD 模块上显示消息	343
14.6 使用 OLED 图形显示器	347
14.7 使用可寻址的 RGB LED 灯条	349
第 15 章 物联网	355
15.0 引言	355
15.1 使用 Web 接口控制 GPIO 输出	355
15.2 在网页上面显示传感器读数	360
15.3 使用 IFTTT 发送电子邮件及其他通知	363
15.4 利用 ThingSpeak 发送 Tweets	367
15.5 CheerLights	369
15.6 向 ThingSpeak 发送传感器数据	370
15.7 使用 Dweet 和 IFTTT 响应 Tweet	373
第 16 章 Arduino 与树莓派	377
16.0 引言	377
16.1 通过树莓派对 Arduino 进行编程	378
16.2 利用 Serial Monitor 与 Arduino 进行通信	380
16.3 配置 PyFirmata 以便通过树莓派来控制 Arduino	382
16.4 通过树莓派对 Arduino 的数字输出进行写操作	384
16.5 使用 PyFirmata 与 TTL 串口	386
16.6 使用 PyFirmata 读取 Arduino 的数字输入	388
16.7 利用 PyFirmata 读取 Arduino 的模拟输入	390
16.8 模拟输出（PWM）与 PyFirmata	391
16.9 利用 PyFirmata 控制伺服电机	393
16.10 定制基于 TTL Serial 的 Arduino 通信	395
16.11 通过 I2C 自定义与 Arduino 之间的通信	399
16.12 在树莓派上使用小型 Arduino	402
16.13 aLaMode 板与树莓派入门	403
16.14 在树莓派和 aLaMode 板上使用 Arduino Shield	406
附录 A 配件与供应商	409
附录 B 树莓派引脚	415

配置与管理

1.0 引言

当你购买树莓派的时候，实际上只是购买了一块印制电路板，它甚至连电源和操作系统都没有。

在正式使用树莓派之前，需要先按照本章介绍的内容做好必要的设置。

由于树莓派采用的是标准 USB 接口的键盘和鼠标，大部分设置还是非常简单的，所以你只需重点关注树莓派特有的配置即可。

1.1 选择树莓派型号

面临问题

树莓派具有多种型号，你不知道哪一种适合自己。

解决方案

从通用性的角度来看，你应当选择树莓派 3 或树莓派 2B 型。与单核的 Pi Zero 和 A+型相比，树莓派 3 的内存容量提升 4 倍，并且具有 4 核处理器，所以对于大部分工作而言，它的表现更加出色。B 型的树莓派 3 的一大特色是内置了 Wi-Fi，所以你就无需使用外置的 Wi-Fi 适配器了。

反之，如果你想将树莓派嵌入到一个单用途项目中的话，那么 A 型树莓派或 Pi Zero 将是更好的选择，因为它们更便宜。

进一步探讨

图 1-1 展示了 Pi Zero、A+型和 B 型树莓派 2。

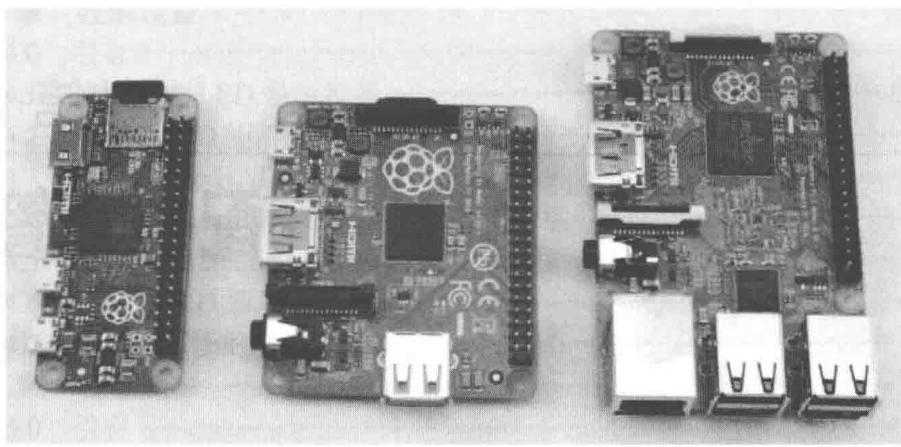


图 1-1 Pi Zero (左)、A+型 (中) 和 B 型 (右) 树莓派 2

如图 1-1 所示，A+型树莓派要比树莓派 2 更加小巧，但是它只有一个 USB 接口，而没有提供 RJ45 以太网接口。不过从外形上看，Pi Zero 最为紧凑，因为它采用的 mini-HDMI 接口和 Micro-USB OTG 接口节约了大量空间。对于 Pi Zero 来说，要想连接键盘、屏幕和鼠标，必须提前准备好支持 USB 和 HDMI 接口的适配器，否则是无法连接标准外围设备的。

表 1-1 为我们总结了树莓派各个型号之间的差异。

表 1-1 不同型号的树莓派的比较

型号	内存容量	USB 接口数量	以太网接口数量	备注
3B	1GB	4	是	内置 Wi-Fi
Zero	512MB	1 (micro 接口)	否	低成本
2B	1GB	4	是	4 核
A+	256MB	1	否	
B+	512MB	4	是	停产
A	256MB	1	否	停产
B rev 2	512MB	2	是	停产
B rev 1	256MB	2	是	停产

如果你拥有的是已经停产的老款树莓派的话也没关系，因为它仍然非常有用。虽然性能方面不如最新的 B 型树莓派 3，但是许多情况下，性能本身是无关紧要的。

在 9.21 节，我们将会介绍树莓派的计算模块，它是一种专门用于嵌入到其他产品内部的树莓派。

参考资料

关于树莓派各种型号的详细介绍，请参考 http://en.wikipedia.org/wiki/Raspberry_Pi。

价格低廉的 Pi Zero 是嵌入到各种电子项目的理想之选，有了它，你就不用担心开销问题了。关于该产品的介绍，请参考 9.22 节。

1.2 封装树莓派

面临问题

你希望给树莓派加一个外壳。

解决方案

除非你的树莓派是作为套件中的一部分购买的，否则它是不带外壳的。这使得树莓派比较容易受伤，因为电路板外面有许多裸露的连接部件，所以把树莓派放到金属上面的时候，非常容易发生短路。

因此，为树莓派购买某种形式的保护装置不失为一项明智之举。如果你将来要用到树莓派的 GPIO 引脚的话，图 1-2 中的 PiBow Coup é 就是一个不错的选择，它不仅设计美观，而且非常实用。

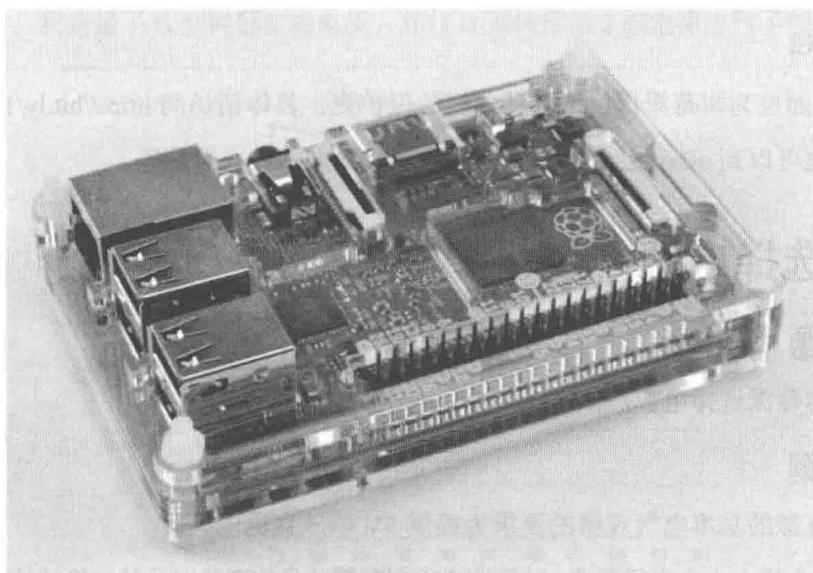


图 1-2 安装在 PiBow Coup é 中的树莓派

进一步探讨

除此之外，还有各种各样的保护装置供你选择，其中包括以下各类。

- 简易的塑料壳，分为两个部分，可通过卡扣组装在一起。
- VESA 壁挂式保护壳（可以附着在显示器或 TV 的背面）。
- 乐高样式的保护壳。
- 3D 打印的保护壳。
- 激光切割而成的卡扣式亚克力保护壳。

保护壳种类繁多，你可以根据自己的喜好购买，与此同时，还必须考虑下面的注意事项。

- 你需要用到 GPIO 接口吗？如果你打算为树莓派连接外部点子设备的话，这个问题就显得非常重了。
- 散热效果如何？如果你打算让树莓派超频运行，或者经常用于播放视频或玩游戏的话，这些情况下树莓派通常会产生大量热量，所以散热就是需要重点考虑的问题了。

此外，你还能找到一些散热器套件，它们通常都会提供许多小巧的带有不干胶的散热器，可以放到树莓派的芯片上用以散热。当你对树莓派的要求比较苛刻，如播放大量视频的时候，这些部件将会派上大用场，但是通常情况下，这些散热器就如同车上的“飞状”条纹一样。

参考资料

Adafruit 上面也为树莓派提供了各种各样的保护壳，具体请访问 <http://bit.ly/1aDT3qm>。

此外，你也可以到 eBay 上寻找其他树莓派商家提供的各式保护壳。

1.3 选择电源

面临问题

你需要为树莓派选择电源。

解决方案

树莓派对电源的基本电气规格的要求为提供 5V 稳压直流电。

至于电源在电流大小方面的要求，这要取决于树莓派的具体型号以及其连接的外围设备。你最好采用一款能够轻松驱动树莓派的电源，因此电流一般不宜小于 700mA。如果你在同一个卖家那里购买树莓派和电源的话，他通常能够告诉你某款电源是否与你的树莓派相匹配。

如果你打算使用无线上网卡或其他大功率的 USB 外设的话，那么最好选用能够提供 1.5A 甚至 2A 电流的电源。

此外需要注意的是，那些非常廉价的电源通常无法提供准确可靠的 5V 电压。

进一步探讨

树莓派的电源和连接器其实与许多智能手机的充电器是完全一样的。只要这些充电器采用的是 micro USB 接口的话，那么其电压几乎都是 5V 的（但是仍需检验一下）。这样的话，唯一的问题就是这些充电器是否能够提供足够的电流。

如果它们无法提供足够的电流的话，那么就会发生下列问题。

- 充电器会过热，甚至有可能引发火灾。
- 充电器本身报废。
- 高负荷运行时（比如当树莓派使用 Wi-Fi 网卡的时候），这会导致电压下降，从而可能引起树莓派重启。

通常来说，我们需要检查电源的相关说明，看看是否至少能够提供 700mA 及以上的电流。如果电源只是标明了功率是多少 W，而没有给出电流是多少 mA，那么将瓦数除以 5 就是 mA 数。也就是说，一个 5V 10W 的电源能够提供 2A（2000mA）的电流。

当我们使用一个最大电流为 2A 的电源的时候，我们用到的电力不会比使用 700mA 电源时的更多。因为树莓派只是按需消耗电流，而不是按照供电器的最大输出电流。

在图 1-3 中，我测量了 B 型树莓派的电流，并与 B 型树莓派 2 的电流进行了相应的比较。

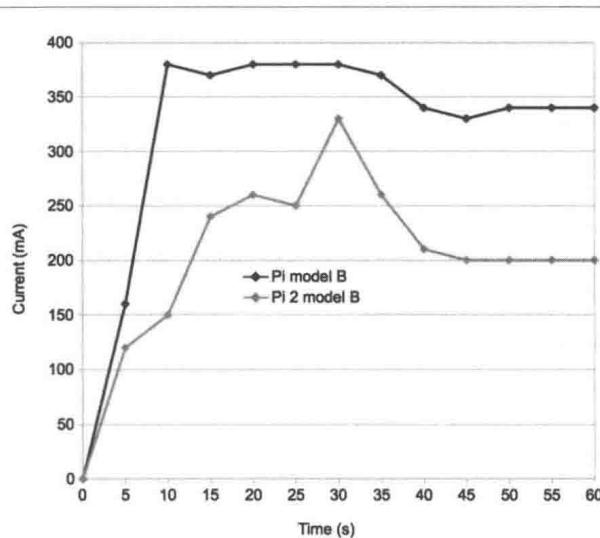


图 1-3 树莓派在引导期间的电流消耗量

虽然新型的树莓派（A+、B+或者树莓派 2）要比旧型号的树莓派更加节能，但是当处理器满负荷或者使用了大量外围设备的时候，它们的电流需求同样也不低。

从图 1-3 所示可以看到，电流很少会高于 500mA。这是因为此时的处理器根本没有做太多的事情。当你开始播放 HD 视频的时候，电流就会显著提高。所以对于电源来说，其额定输出电流最好要高一些，以备不时之需。

参考资料

你可以从 <http://www.pi-supply.com> 购买一个模块，以便在树莓派关机时可以自动切断电源。

1.4 选择操作系统发行包

面临问题

面对众多的树莓派操作系统发行包，你不知道应该选择哪一种。

解决方案

问题的答案取决于你到底想要用树莓派做什么。

如果你想将树莓派作为普通计算机使用，或者用于电子项目中的话，那么你就可以选择树莓派的官方发行包：Raspbian。

如果你打算将树莓派作为媒体中心使用的话，也可以找到许多专门为为此定制的发行包（参见 4.1 节）。

就本书而言，我们基本上以 Raspbian 发行包为主，但是这些代码和命令同样适用于所有基于 Debian 的发行包。

进一步探讨

由于 MicroSD 卡不是很贵，所以你可以多准备几张存储卡，尽量多尝试几种发行包。如果你有此打算的话，最好把自己的文件存储在一个 U 盘上面，这样就无需给每个 microSD 卡复制这些文件了。

请注意，如果你打算按照本书中的例子对 SD 卡进行写操作的话，那么要求你的电脑具有 SD 卡插槽（许多笔记本都有），或者专门购置一台廉价的 USB SD 读卡器。

参考资料

树莓派的官方网站提供了许多树莓派发行包，具体请访问 <http://www.raspberrypi.org/downloads>。

1.5 通过 NOOBS 刷写 microSD 卡

面临问题

你想通过 NOOBS (New Out of the Box Software) 刷写 microSD 卡。

解决方案

NOOBS 是迄今为止为树莓派提供操作系统的最简便的方式。

从 <http://www.raspberrypi.org/downloads> 下载 NOOBS 压缩文件，解压，然后将其放到 microSD 卡中即可。为此，要求计算机装有 SD 卡插槽，或提供 USB 适配器和 SD 转 microSD 适配器。

你下载 NOOBS 压缩文件之后，将其解压缩，并将整个文件夹下面的内容复制到 SD 卡中。需要注意的是如果解压缩的文件夹名为 NOOBS_v1_3_12 或类似名称的话，那么你需要将该文件夹下面的内容复制到 microSD 卡的根目录下面，而不是直接复制这个文件夹。

将存有解压后的 NOOBS 文件的 microSD 卡插入树莓派中，然后启动树莓派。当它启动时，你将看到如图 1-4 所示的窗口。在此窗口中，选择 Raspbian，并单击 Install 按钮。

如果你在 A+ 上面使用 NOOBS 的话，你看到的备选列表会更短一些，因为只会显示适用于该精简平台的发行包。

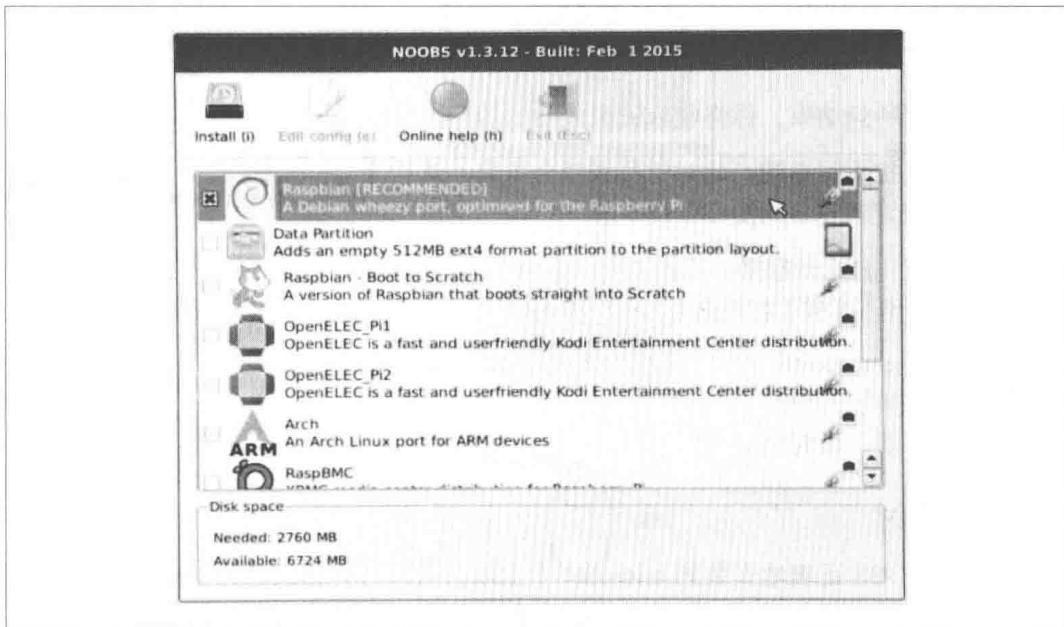


图 1-4 NOOBS 的开始界面

你将收到一条警告消息，指出该 SD 卡将被重写（这正是我们想要做的），然后，选择的发行包会安装到该 SD 卡上面，同时还会显示一个安装进度条以及与该发行包有关的帮助信息（见图 1-5）。

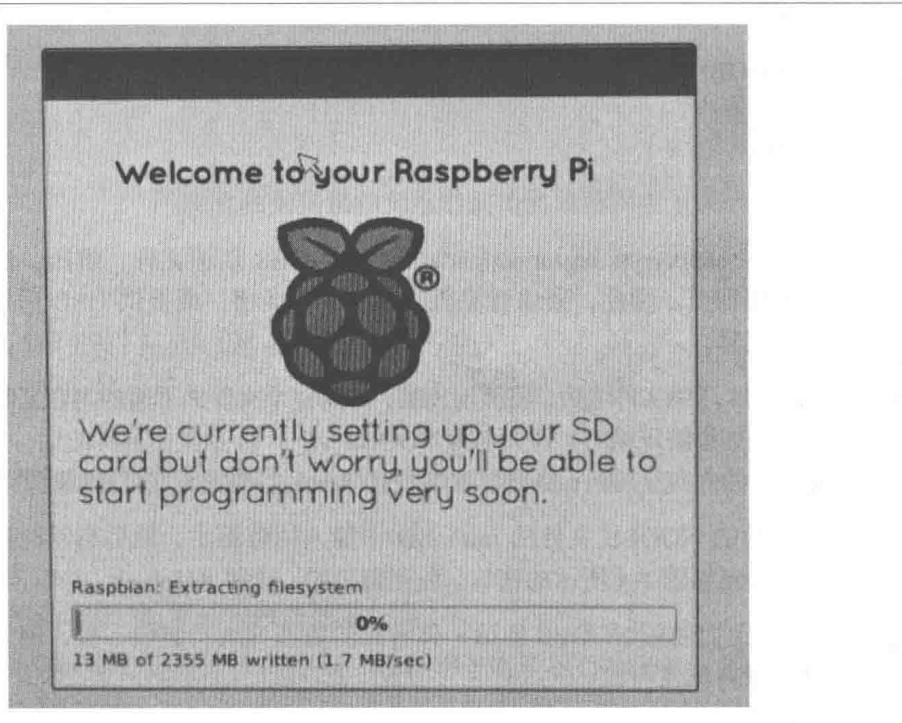


图 1-5 NOOBS 重写 SD 卡

文件复制过程一旦结束，你将收到镜像成功的消息。

届时，你可以按下回车键，此时树莓派就会重启，然后 `raspi_config` 就会自动运行，这样你就可以配置新安装的操作系统了。

树莓派的系统一旦运行起来，你的首要任务就是连接互联网（见 2.1 节和 2.5 节），通过 LXTerminal（见 3.2 节）打开命令行，然后输入下列命令将操作系统更新到最新版本。

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

这个过程将需要一段时间。

进一步探讨

为了能将 NOOBS 正确地安装到 microSD 卡上面，必须将它格式化为 FAT32 格式。通常情况下，市场上出售的 SD 和 microSD 卡都已经提前格式化为 FAT32 格式了。如果你打算使用的是一张旧卡，并且需要格式化成 FAT32 的话，那么只要利用操作系统自

身的工具就可以完成此项任务。MicroSD 存储卡的类型对安装于其上的操作系统的运行速度有非常大的影响，所以挑选的时候，应当尽量选择标有“class 10”的存储卡。

参考资料

通过阅读 <https://www.raspberrypi.org/help/noobs-setup/> 页面，不仅可以了解使用 NOOBS 安装操作系统的详细信息，同时还能找到更多类型的发行包。

1.6 装配系统

面临问题

树莓派所需的一切都已经准备好了，接下来需要将它们装配起来。

解决方案

除非你打算把树莓派嵌入到项目内部，或者想将其用于媒体中心，否则的话，就需要为其安装键盘、鼠标，甚至 Wi-Fi 无线网卡，当然树莓派 3 除外。

图 1-6 展示的是一个典型的树莓派系统。

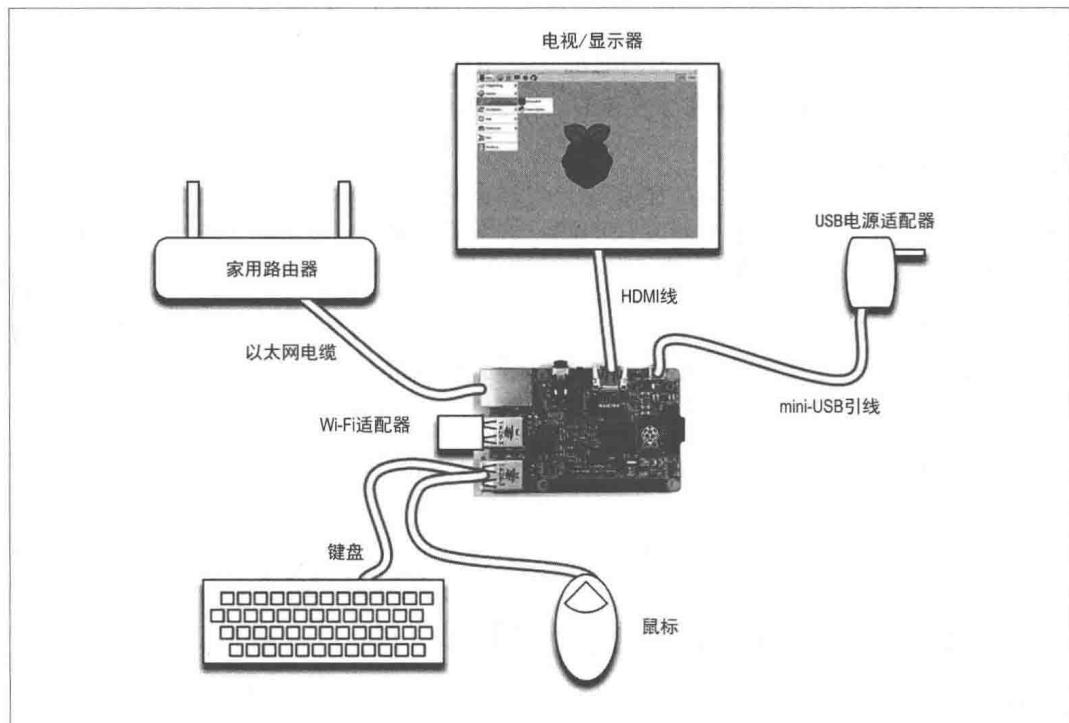


图 1-6 一个典型的树莓派系统

进一步详述

树莓派几乎可以完美搭配所有类型的键盘和鼠标，无论是有线的，还是无线的。当然，无线蓝牙键盘和鼠标是个例外，它们无法用于树莓派。

如果你手头上的是老款树莓派，或者 A 型或 A+型树莓派但是 USB 接口不够用的话，那么你还需购置 USB 集线器。

参考资料

树莓派官方出品的快速入门指南见 <http://bit.ly/1ju8usM>。

1.7 连接 DVI 或 VGA 显示器

面临问题

你的显示器没有 HDMI 连接器，但是你仍然想把它用在树莓派上面。

解决方案

许多人都面临这个问题，不过，对于这些没有 HDMI 接口但是支持 DVI 或者 VGA 输入的显示器来说，你只要购买相应的适配器就可以解决这个问题了。

DVI 适配器不仅用起来最为简单，同时也是最便宜的。如果你以“HDMI male to DVI female converter”作为关键词进行搜索的话，会找到许多低于\$5 的 DVI 适配器。

进一步探讨

VGA 适配器的用法相对来说要更复杂一些，因为它需要相应的电子设备将数字信号转换为模拟信号，所以你要留意一下是否提供了这些转接线。

官方的转换器名为 Pi-View，任何销售树莓派的地方都能买到它。使用 Pi-View 的好处是它已经过了长时间的测试，并且非常适合用于树莓派。当然，在互联网上面还可以找到许多廉价的替代品，不过这些通常都不靠谱。

参考资料

ELinux 上面也介绍了许多挑选转换器的忠告，其地址为 <http://bit.ly/rpi-peripherals>。

1.8 使用复合视频显示器/TV

面临问题

当文本显示在低分辨率的复合视频显示器上面的时候，它们是非常难以辨认的。所以，

你需要调整树莓派的分辨率，以适应较小的屏幕。

解决方案

树莓派提供了两种视频输出：HDMI 和来自音频插孔的复合视频，后者需要专用的线缆。其中，HDMI 的输出质量要更高一些。如果你打算使用复合视频作为主显示器信号的话，那么你可能需要重新斟酌一下。

如果你果真打算使用这种屏幕的话，比如确实需要一个小屏幕的话，那么你必须做出相应的调整，以使视频输出适合屏幕大小。为此，你必须修改 /boot/config.txt。你可以在树莓派上编辑该文件，具体可以在终端会话中利用下列命令完成这些操作。

```
$ sudo nano /boot/config.txt
```

如果文字太小难以看清，而你又没有 HDMI 显示器的话，那么你可以将 SD 卡从树莓派上面取下来，将其插入计算机中。该文件位于 SD 卡的根目录下面，你可以通过计算机上面的文本编辑器来进行修改。

你需要了解自己屏幕的分辨率是多少。对于许多小屏幕来说，它们的分辨率通常是 320×240 像素。在这个文件中，你可以找到如下所示的两行内容。

```
#framebuffer_width=1280  
#framebuffer_height=720
```

将以上两行前面的 # 去掉，并将这两个数字改为你显示器的宽度和高度。在下面的例子中，这两个数字已经被改为 320 和 240。

```
framebuffer_width=320  
framebuffer_height=240
```

保存文件，然后重新启动树莓派。你将会发现，屏幕上面的内容已经变得更加易于阅读了。你也许还会发现，屏幕周围出现了一个大黑边。要想调整黑边，可以参考 1.9 节。

进一步探讨

除此之外，树莓派还可以使用许多廉价的 CCTV 监视器，这时它们看起来非常像仿古的游戏控制台（参见 4.6 节）。

不过，这些监视器的分辨率通常会很低。

参考资料

如果你想阅读其他复合视频显示器的使用指南的话，可以访问 Adafruit 提供的有关介绍，地址为 <http://bit.ly/adafruit-learning>。

另外，如果你使用的是 HDMI 视频输出的话，可以按照本书的 1.7 节和 1.9 节介绍的方法来调整图像。

1.9 调整显示器中的图像尺寸

面临问题

当你第一次将树莓派连接到显示器上面时，可能发现某些文本无法阅读，因为它们跑到屏幕外边去了，或者遇到窗口无法充满整个屏幕的问题。

解决方案

如果遇到文本延伸到屏幕之外的问题，你可以通过 raspi-config 工具关闭 overscan 功能来加以解决。

为此，可以打开一个终端会话，然后通过下列命令来运行 raspi-config 工具。

```
$ sudo raspi-config
```

然后，使用光标键滚动到 Advanced 选项中的 Overscan 处，并将其关闭（见图 1-7）。

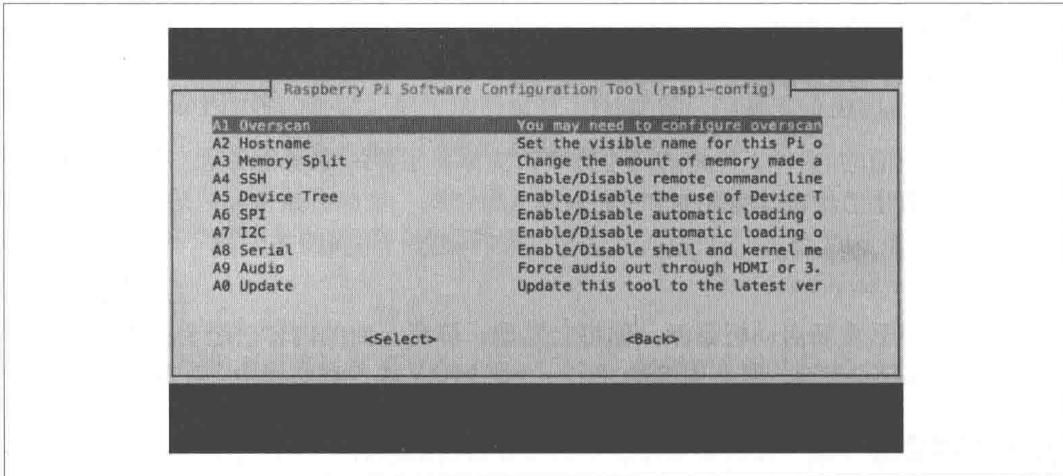


图 1-7 选中 Overscan 选项

如果你的问题是图片周围有大量黑边，那么可以通过编辑文件 /boot/conig.txt 来减少黑边（甚至可以完全消除），具体命令如下所示。

```
$ sudo nano /boot/config.txt
```

找到与 overscan 有关部分，并修改图 1-8 所示的 4 行内容。

要想让这 4 行内容起作用，首先得把每行前面的#符去掉。

之后，可以借助试错法，不断调整设置，直到窗口尽量充满整个显示器为止。需要注意的是这 4 个数字都应该是负数。在调整这些设置时，不妨将 20 作为它们的起始值。

这样能够增大所用屏幕的面积。

```
GNU nano 2.2.6          File: /boot/config.txt
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
#disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280_
#framebuffer_height=720

Get Help  WriteOut  Read File  Prev Page  Cut Text  Cur Pos
Exit  Justify  Where Is  Next Page  LinCut Text  To Spell
```

图 1-8 调整 overscan

进一步探讨

每次修改分辨率之后，都得重启树莓派才能看到修改后的效果，所以整个过程非常烦人。幸运的是这种事情只需要进行一次就行了，并且大部分显示器和 TV 无需调整就能正常工作。

参考资料

关于 raspi-config 工具的详细用法，请参考 http://elinux.org/RPi_raspi-config。

1.10 优化性能

面临问题

你感觉自己的树莓派运行得太慢了，所以，想通过超频来提速。

解决方案

如果你使用的是 4 核处理器的树莓派 2 的话，你不会有太慢的感觉。但是，如果你使用的是单核心处理器的老版树莓派的话，感觉就像是老牛拉破车。

为了提高树莓派的运行速度，你可以使用超频方法。

当然，这会使树莓派的耗电量有所增加，同时也会使它变得更热（参考后面的讨论）。

这里使用的超频方法称为动态超频，因为它会自动检测树莓派的温度，并且一旦温度过高，时钟频率就会自动下调。

为了给树莓派超频，可以在终端输入下列命令来运行 raspi_config 工具。

```
$ sudo raspi-config
```

在菜单中选中 Overclock 选项，这样系统就会弹出如图 1-9 所示的各种选择。

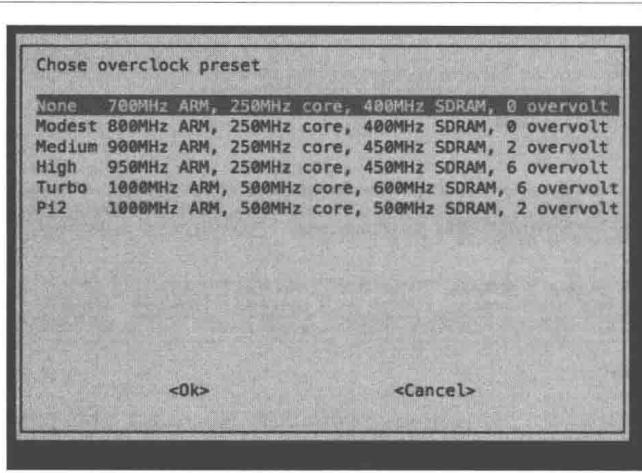


图 1-9 超频选项

你可以从中选择一个选项。如果你的树莓派开始变得不稳定，甚至挂机的话，那么说明需要选择一个更保守的选项，或者重新设为 None 以关闭超频功能。

进一步探讨

利用超频方法，性能会得到明显改善。为了衡量改善程度，我在 15 摄氏度的室温环境下对 B 型树莓派修订版 2 做了一个实验。

我使用下面的 Python 代码作为测试程序。这段代码只针对处理器，而没有涉及计算机的其他方面，例如写 SD 卡、绘图等。不过如果你想测试树莓派的超频效果的话，那么该程序确实能够较好地反映出 CPU 的性能。

```
import time

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

before_time = time.clock()
for i in range(1, 10000):
    factorial(200)
after_time = time.clock()

print(after_time - before_time)
```

测试结果如表 1-2 所示。

表 1-2

超频效果

	速度	电流	温度（摄氏度）
700MHz	15.8 秒	360mA	27
1GHz	10.5 秒	420mA	30

如您所见，性能提升了 33%，但是性能提升的代价是消耗更大的电流，同时带来更高的温度。

为了让树莓派全速运行，你最好提供一个通风良好的外壳。

除此之外，你还可以设法为树莓派提供水冷设备，不过说老实话，这样做非常不明智。

参考资料

关于 raspi-config 工具的详细用法，请参考 http://elinux.org/Rpi_raspi-config。

1.11 修改密码

面临问题

通常情况下，树莓派的默认密码为 Raspberry，你想修改此密码。

解决方案

你可以利用 raspi-config 工具来修改密码。在终端（见 3.2 节）中，你可以通过下列命令来运行 raspi_config 工具。

```
$ sudo raspi-config
```

之后，在菜单中选择 change_pass 菜单项，这时将弹出如图 1-10 所示的提示内容。

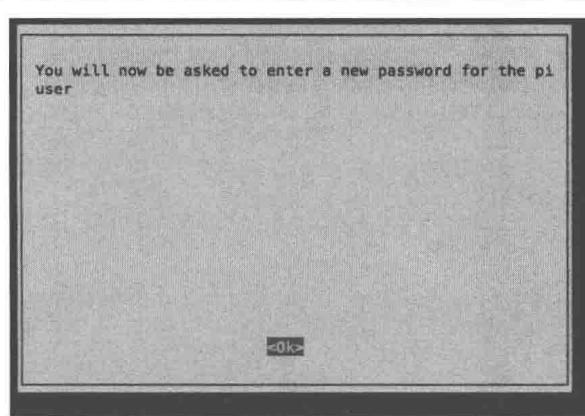


图 1-10 修改密码

修改密码时，我们无需重启树莓派，所做修改就会立即生效。

进一步探讨

你还可以在终端中利用 passwd 命令来修改密码，具体如下所示。

```
$ passwd  
Changing password for pi.  
(current) UNIX password:  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

参考资料

关于 raspi-config 工具的详细用法，请参考 http://elinux.org/Rpi_raspi-config。

1.12 让树莓派开机后直接进入窗口系统

面临问题

每次重启树莓派之后，都不得不登录后以手工方式进入桌面系统。你希望这个过程实现自动化。

解决方案

你可以使用 raspi-config 工具修改启动过程，让树莓派开机后自动登录并打开桌面系统。为此，可以在终端中通过 raspi_config 工具完成相应的设置，具体命令如下所示。

```
$ sudo raspi-config
```

然后，依次选择 Enable Boot to Desktop/Scratch 和 Desktop Log in as user pi 选项，具体如图 1-11 所示。

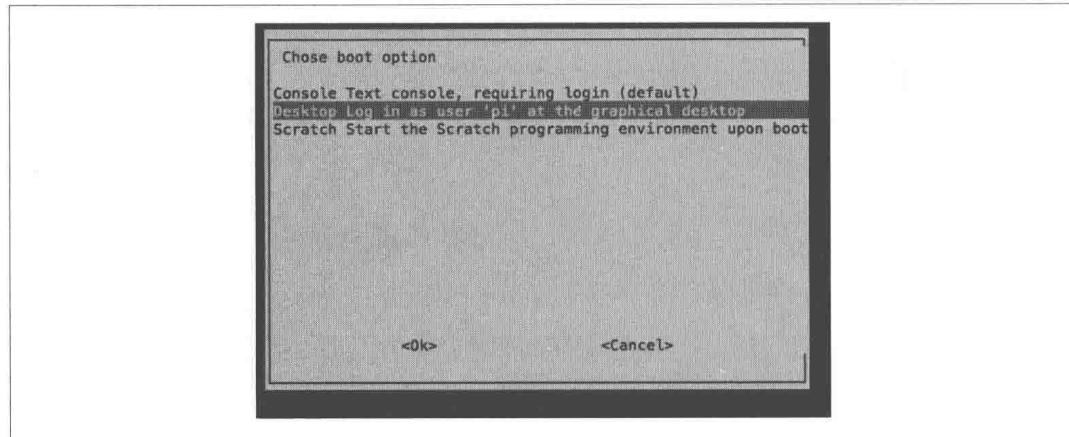


图 1-11 自动启动进入桌面进一步探讨

修改好引导选项之后，系统会提示重新启动树莓派以使修改生效。

很明显，允许树莓派在启动后自动进入桌面环境，这会带来许多安全隐患。但是，树莓派通常只是用作个人电脑使用，而非多人共享，所以便利性会重于一切。

参考资料

关于 raspi-config 工具的详细用法，请参考 http://elinux.org/Rpi_raspi-config。

1.13 关闭树莓派

面临问题

你想关掉自己的树莓派。

解决方案

单击桌面左上角的 Raspberry 菜单，将显示如图 1-12 所示的一些选项。

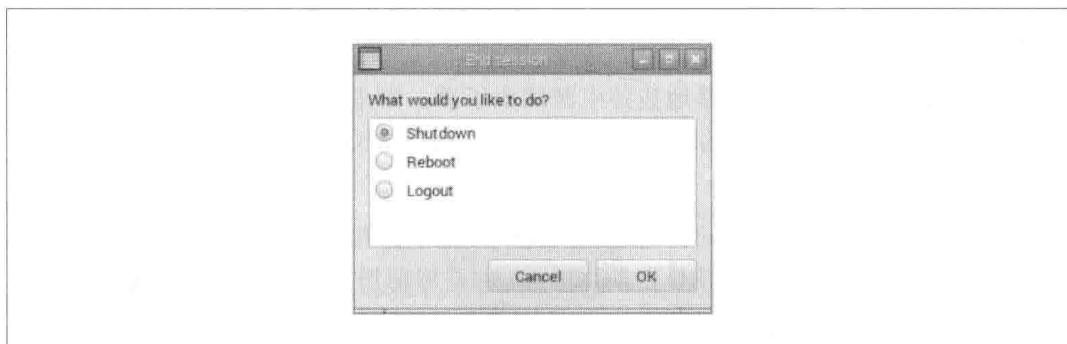


图 1-12 关闭树莓派

Shutdown

关闭树莓派。要想再次启动树莓派的话，需要拔掉电源，然后重新插上才行。

Reboot

重新启动树莓派。

Logout

注销，并显示登录提示符，以便输入登录凭证重新登录。

此外，你也可以在命令行输入下列命令来重启树莓派。

sudo reboot

在安装某些软件之后，通常会用到上面这条命令。当你重新启动时，会看到如图 1-13 所示的消息，它一方面展示了 Linux 系统的多用户特性，同时为所有连接到树莓派的用户提出了警示。



图 1-13 通过终端关闭树莓派

进一步探讨

关闭树莓派时，不要简单粗暴地拔掉电源插头了事，而是最好采用上面介绍的关机方法，因为当你给树莓派断电的时候，可能它正在向 microSD 卡中写入数据。这样的话，就会导致文件受损而无法正常使用。

与大多数电脑的关机方法不同，树莓派的关机实际上并非切断电源。它只是进入了低功耗模式，而树莓派本来就用不了多少电的（但是树莓派自身硬件无法控制其电源）。

参考资料

你可以从 <https://www.pi-supply.com/> 购买一款硬件设备，以实现树莓派关机的同时切断其电源。

1.14 安装树莓派摄像头模块

面临问题

你想使用树莓派的摄像头模块（见图 1-14）。

解决方案

树莓派摄像头模块（见图 1-14）是通过排线连接到树莓派上面的。

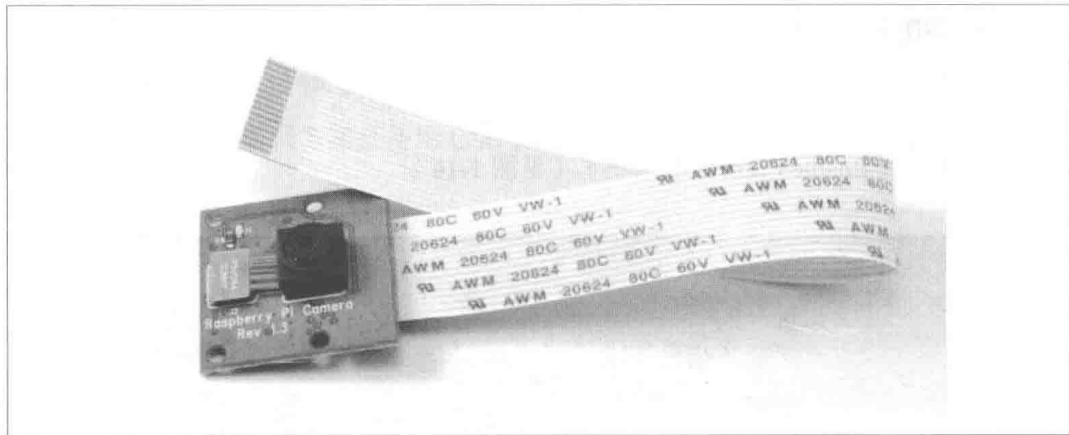


图 1-14 树莓派摄像头模块

这个排线需要连到介于树莓派 2 的音频和 HDMI 接口之间的那个专用连接器上面。对于早先的 B 型树莓派来说，这个连接器位于以太网接口的后面。若要安装该模块，需要拉开连接器两边的扳手，使连接器处于解锁状态，然后将排线插入插槽，注意要让排线连接器的衬垫背对以太网接口。之后，按下两边的扳手，锁住排线（见图 1-15）即可。

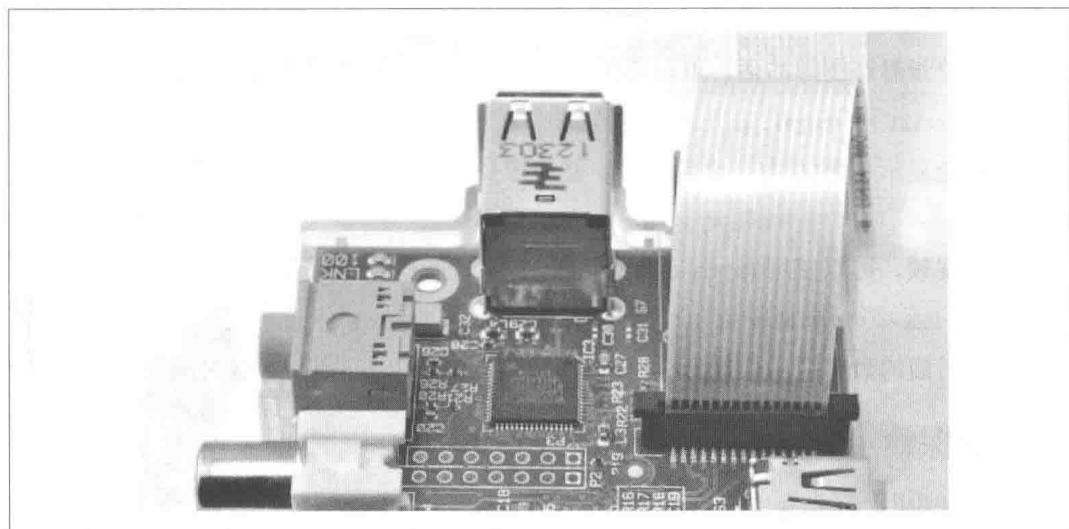


图 1-15 为 B 型树莓派安装摄像头模块

摄像头包装上面的说明文字指出它对静电敏感。

因此，在开始操作之前，需要先释放身上的静电，如触摸 PC 金属机箱等接地设备都可以消除静电。

安装好摄像头硬件之后，还需要进行必要的软件配置才能使用。其中，最简单的配置

方法就是使用 raspi_config。若要运行 raspi-config，可以在终端会话中键入如下所示的命令。

```
$ sudo raspi-config
```

这样你就能看到 Enable Camera 的选项了（见图 1-16）。

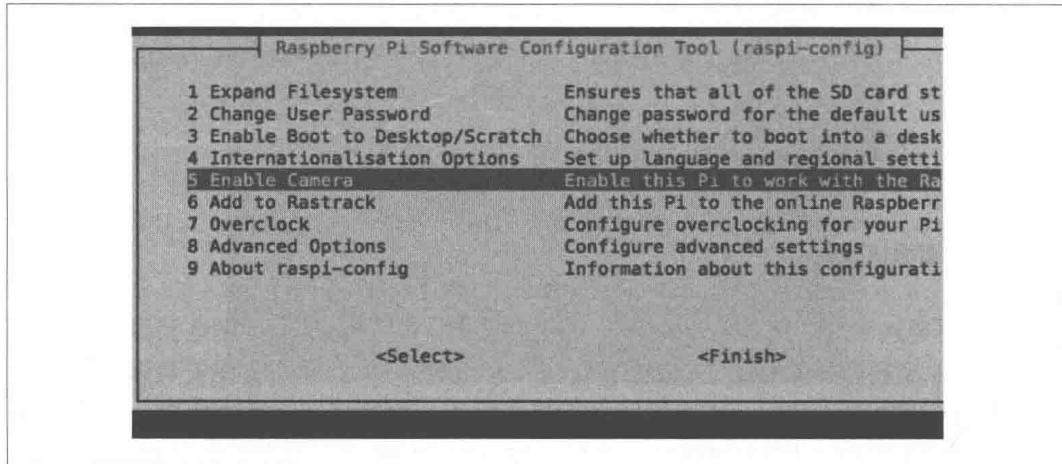


图 1-16 更新后的 raspi-conig 配置工具

有两个分别用来照相和摄像之用的命令，它们分别是 raspistill 和 raspivid。

要想拍摄单张静态图像的话，可以使用 raspistill 命令，具体用法如下所示。

```
$ raspistill -o image1.jpg
```

预览屏幕会显示 5 秒左右，之后拍照，并将照片保存到当前目录下面的 image1.jpg 文件中。

为了录制视频，可以使用 raspivid 命令，具体如下所示。

```
$ raspivid -o video.h264 -t 10000
```

上述命令后面的数字表示录像时间，单位是毫秒，就本例而言，录像时间是 10 秒。

进一步探讨

无论是 raspistill，还是 raspivid 命令，都提供了许多命令选项。无论其中哪个命令，如果只输入命令本身而不带任何参数的话，就会自动显示可用选项的帮助信息。

该摄像头模块能够提供高分辨率的摄像和录像功能。

下面是该摄像头的一些关键参数。

- 500 万像素传感器。
- f/2 定焦镜头。

- 静态分辨率为 1920 像素 × 1080 像素。
- 视频为 1080p, 30 帧/秒。

摄像头模块的一个替代方案是使用 USB 摄像头（请参考 8.2 节）。

参考资料

关于命令 raspistill 和 raspivid 的详细介绍, 请参考 RaspiCam 的说明文档 (<http://bit.ly/cam-doc>)。

1.15 使用蓝牙设备

面临问题

我想在自己的树莓派上面使用蓝牙设备。

解决方案

为树莓派安装 USB 蓝牙适配器, 并安装相应的支持软件。

并非所有的蓝牙适配器都能与树莓派无缝兼容, 但是大部分都是兼容的, 为确保万一, 购买时请选择那些声明能够兼容树莓派的适配器。图 1-17 中的树莓派配备了两个 USB 蓝牙适配器（最近的那个是摄像头）以及一个 USB Wi-Fi 适配器。

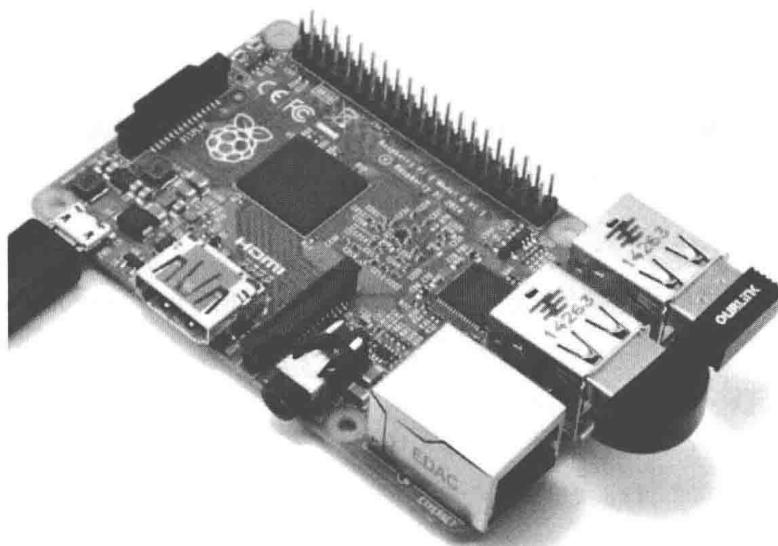


图 1-17 配备了 USB 蓝牙和 Wi-Fi 适配器的树莓派 2

为了安装蓝牙的支持软件，可以使用如下所示的命令。

```
$ sudo apt-get update  
$ sudo apt-get install bluetooth bluez-utils blueman bluez  
$ sudo usermod -G bluetooth -a pi
```

这些命令适用于所有树莓派支持的蓝牙适配器。

安装好必要的软件之后，就可以插入蓝牙适配器了，之后重启树莓派（参考 1.13 节）即可。

这时，你将会在 Raspbian 的 Start 菜单的 Preferences 部分发现一个新工具，名为 Bluetooth Manager。打开这个工具，然后单击 Search，你就可以找到附近的蓝牙设备了（见图 1-18）。当然，附近的蓝牙设备必须设置成对其他设备可见的情况下，你才能找到它们。

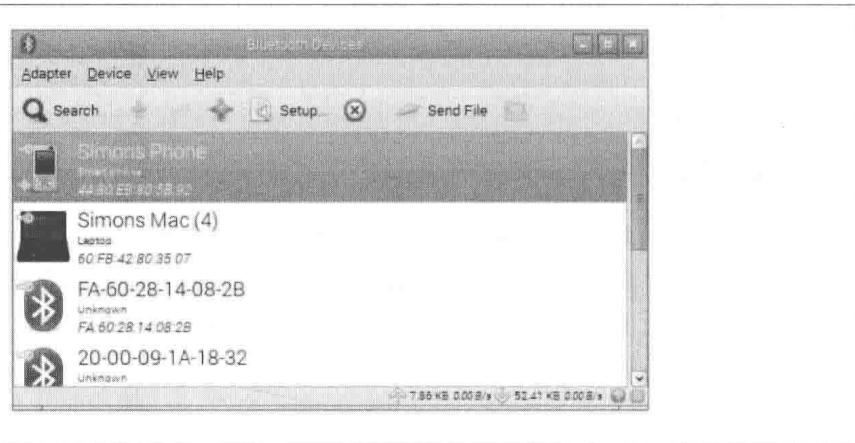


图 1-18 Bluetooth Manager 进一步探讨

通过 Bluetooth Manager，你可以与其他蓝牙设备进行配对，给它们发送文件，以及将自己的树莓派设置成对其他蓝牙设备可见等。

参考资料

关于兼容树莓派的适配器的清单，请参考 http://elinux.org/Rpi_usb_bluetooth_adapters。

网络连接

2.0 引言

树莓派在设计之初，便是要连接到互联网的。互联网通信是它的关键功能之一，这给其他各种用途铺平了道路，例如家庭自动化、Web 服务、网络监控等。

树莓派既可以使用以太网线缆（这种情况至少要求是 B 型树莓派）联网，也可以使用 USB Wi-Fi 无线网卡连接网络。树莓派一旦连接到互联网上面，就意味着你可以通过其他计算机来远程连接树莓派。这对于本身难以接近，或者没有连接键盘、鼠标和显示器的树莓派来说，是非常有用。

接下来，本章将详细介绍树莓派连接互联网以及通过网络远程控制它的各种方法。

2.1 连接有线网络

面临问题

你想通过有线网络将树莓派连接到互联网。

解决方案

首先，如果你的树莓派的型号为 A、A+或 Zero 的话，那么它们自身并没有提供 RJ45 以太网接口。这种情况下，最好使用 USB 无线网卡来连接互联网（见 2.5 节）。

如果你的树莓派是 B 型的，可以将以太网电缆插入 RJ45 接口，同时将线缆另一端接入家用路由器后方的空闲接口中即可。图 2-1 展示的是一款老版的树莓派 1，它的网络 LED 就在音频接口旁边。对于树莓派 2 来说，这些 LED 位于以太网接口自身内部。

树莓派一旦连接到网络上面，它的网络 LED 就会立即开始闪烁。

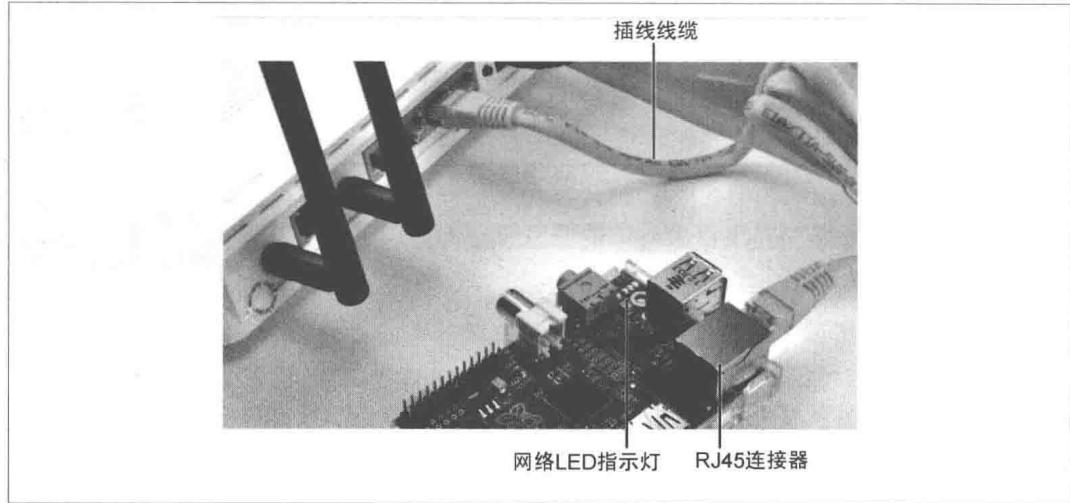


图 2-1 连接树莓派到家用集线器

进一步探讨

对于 Raspbian 系统来说，它被预配置为使用 DHCP (Dynamic Host Configuration Protocol, DHCP) 协议连接网络。只要你的网络支持 DHCP 协议，它就会自动为网络设备分配 IP 地址。

当你把树莓派连接到家用路由器上面的时候，如果树莓派的 LED 没有变亮的话，请检查是否已经将网线插入到集线器的上行链路 RJ45 接口中，或者换一根网线重新尝试。

如果 LED 闪烁，但是树莓派仍然无法使用浏览器上网的话，请通过网络管理控制台检查 DHCP 是否已经开启。为此，你可以查看如图 2-2 所示的有关选项。

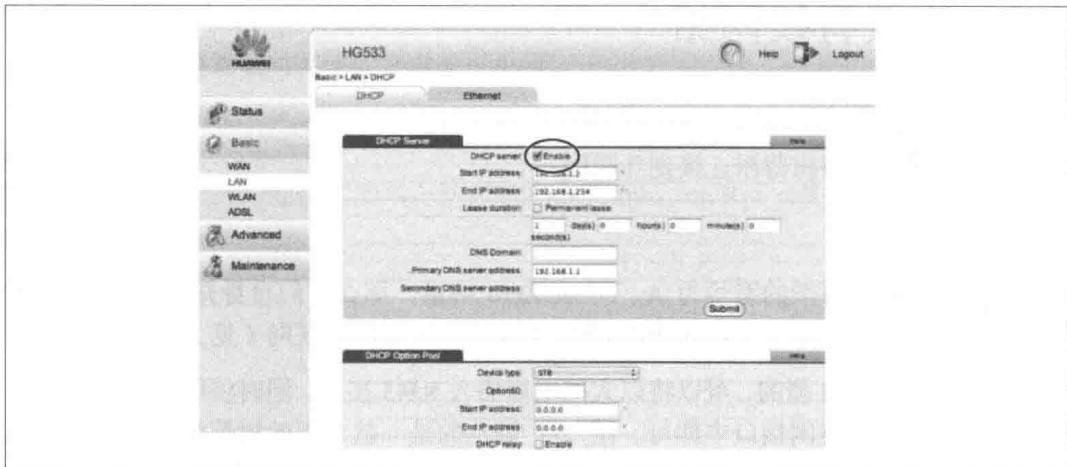


图 2-2 为家用集线器启用 DHCP 协议

参考资料

通过无线网络连接互联网的具体方法，请参考 2.5 节。

2.2 查看自己的 IP 地址

面临问题

你想知道自己树莓派的 IP 地址，以便与它进行通信，无论是将树莓派用作 Web 服务器，与其交换文件，还是利用 SSH（见 2.7 节）或 VNC（见 2.8 节）远程控制树莓派，你都需要知道它的 IP 地址。

一个 IP 地址由 4 部分数字组成，能够在网络中唯一地标示计算机的网络接口。该地址各部分数字之间用点号分隔。

解决方案

要想获悉自己树莓派的 IP 地址，只需在终端窗口中输入下列命令即可。

```
$ hostname -I  
192.168.1.16
```

上面看到的是树莓派在家庭网络中的本地 IP 地址。

进一步探讨

一个树莓派可以拥有多个 IP 地址（例如每个网络连接分配一个 IP 地址）。所以，如果你的树莓派同时连接了有线网络和无线网络的话，那么它就会拥有 2 个 IP 地址。但是，通常情况下树莓派只会使用一种网络连接，要么使用有线连接，要么使用无线连接，而不是同时使用两种连接。要想查看所有的网络连接，可以使用 ifconfig 命令。

```
$ sudo ifconfig  
  
eth0      Link encap:Ethernet HWaddr b8:27:eb:d5:f4:8f  
          inet addr:192.168.1.16 Bcast:192.168.255.255 Mask:255.255.0.0  
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
            RX packets:1114 errors:0 dropped:1 overruns:0 frame:0  
            TX packets:1173 errors:0 dropped:0 overruns:0 carrier:0  
            collisions:0 txqueuelen:1000  
            RX bytes:76957 (75.1 KiB) TX bytes:479753 (468.5 KiB)  
  
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)
```

```
wlan0      Link encap:Ethernet HWaddr 00:0f:53:a0:04:57
          inet addr:192.168.1.13 Bcast:192.168.255.255 Mask:255.255.0.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:38 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:6661 (6.5 KiB) TX bytes:6377 (6.2 KiB)
```

通过观察 ifconfig 的返回结果，你不难发现，当前树莓派同时连接了有线网络（eth0）和无线网络（wlan0），其中前者的 IP 地址为 192.168.1.16，后者的 IP 地址为 192.168.1.13。其中，网络接口 lo 实际上是一个虚拟网络接口，计算机可以借助它跟自己通信。还有一种获悉自己 IP 地址的方法，那就是连接到家用集线器的管理控制台上，然后查看 LAN 页面中的 IP 表。其中，应该会有一个名为 raspberrypi 的设备，旁边就是它的 IP 地址。

参考资料

在 Wikipedia 网站上，有关于 IP 地址的详尽说明，具体请参考 <http://bit.ly/13ZsgoY>。

2.3 配置静态 IP 地址

面临问题

你想给自己的树莓派配置一个静态 IP 地址，这样该地址就不会发生变化了。

解决方案

要给树莓派配置 IP 地址，无论它使用的是有线网络，还是无线网络，你都需要编辑配置文件 /etc/network/interfaces。

如果你想查看自己的 /etc/network/interfaces 文件，可以借助下列命令。

```
$ more /etc/network/interfaces
```

该文件看起来应该如下所示。

```
auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

这相当于说，Raspbian 会留意你的树莓派上的 3 个网络接口，每个接口前面都以单词 iface 开头。

Lo

本地环回接口，该接口可忽略。

eth0

一个使用以太网接口的网络连接。

wlan0

一个使用 USB Wi-Fi 无线网卡或树莓派 3 内置的 Wi-Fi 网卡的网络接口。

对于树莓派上面的每个网络连接，它们都有不同的 IP 地址。就本例来说，你将为以太网接口指定一个静态的 IP 地址。如果你想把 Wi-Fi 接口的 IP 地址也设置成静态的，只需编辑接口文件中的相应条目即可。

要想编辑该文件，可以借助如下所示的命令。

```
$ sudo nano /etc/network/interfaces
```

首先，要确定使用哪个 IP 地址。对这个 IP 地址的要求是：它应该是网络中其他机器尚未使用的地址，同时还要位于你的家用集线器的 IP 地址范围之内。就这里而言，我们将使用 192.168.1.116。

编辑该文件内容，将单词 `dhcp` 改为 `static`，并添加如下所示内容。

```
address 192.168.1.116
netmask 255.255.255.0
gateway 192.168.1.1
```

按照这里的要求修改文件后，静态 IP 地址 192.168.1.116 已经被分配给了接口 `eth0`。

```
auto lo

iface lo inet loopback
iface eth0 inet static
    address 192.168.1.116
    netmask 255.255.255.0
    gateway 192.168.1.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

对于大部分网络来说，网络掩码都应该设为 255.255.255.0，同时将网关设置为家庭路由器本身的 IP 地址。这样的话，该 IP 地址就与你连接路由器的管理控制台所用地址完全一致了。

编辑保存该文件之后，请运行下面的命令来清空所有现存的 DHCP 配置项，然后重启树莓派，以便让所做修改生效。

```
$ sudo rm /var/lib/dhcp/*
$ sudo reboot
```

进一步探讨

内部 IP 地址通常都类似于 192.168.1.116 这样，对于每台电脑来说，只有最后一个数字会有所变化。内部 IP 地址的另一种常用格式是 10.0.0.16。

参考资料

在 Wikipedia 网站上，有关于 IP 地址的详尽说明，具体请参考 <http://bit.ly/13ZsgoY>。

2.4 为树莓派配置网络名称

面临问题

你想给树莓派配置一个网络名称，这样，它的名字就不再是单调的“raspberrypi”了。

解决方案

修改树莓派的网络名称其实非常简单，只需要修改两个文件就可以了。

首先，需要编辑 /etc/hostname 文件。为此，请打开终端窗口，并输入如下所示的命令。

```
$ sudo nano /etc/hostname
```

然后，使用选好的名字替换掉“raspberrypi”即可。需要注意的是你选择的名称应该还是一个单词，不能含有任何标点符号和特殊字符（包括字符_）。

然后，使用编辑器打开文件 /etc/hosts，具体命令如下所示。

```
$ sudo nano /etc/hosts
```

该文件大致如下所示。

```
127.0.0.1      localhost
::1            localhost ip6-localhost ip6-loopback
fe00::0        ip6-localnet
ff00::0        ip6-mcastprefix
ff02::1        ip6-allnodes
ff02::2        ip6-allrouters

127.0.1.1      raspberrypi
```

修改文件最后面的部分，将原名称（“raspberrypi”）改为新名称即可。

重启树莓派之后，如果从其他电脑通过网络观察其名称的话，应当会看到相应的变化。

进一步探讨

修改树莓派的名称有时非常有用，特别是当你将多台树莓派连接到网络上面的时候。

参考资料

关于修改树莓派的 IP 地址的方法，可以参考 2.3 节。

2.5 配置无线网络连接

面临问题

你想通过 USB 无线网卡将树莓派连接到互联网上。

解决方案

如果你的 Raspbian 是最新版本的话，那么配置 Wi-Fi 的任务将易如反掌。你只需插入 USB Wi-Fi 无线网卡，然后在屏幕右上方单击 Network 图标即可（见图 2-3）。之后，你会看到一个无线网络清单。你选择网络后，系统将提示你输入 Pre Shared Key，即密码。

输入密码，稍等片刻，Network 图标就会变为标准的 Wi-Fi 标志，说明无线网络已经连接成功。

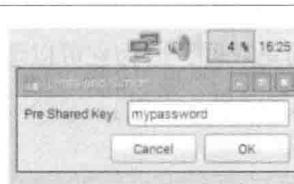


图 2-3 连接 Wi-Fi 网络

如果你使用的是老版的 Raspbian 的话，那么就不得不借助 Wi-Fi Config 工具（桌面上可以找到它的快捷方式）来进行配置了。如果你的 Raspbian 不是最新版本的话，不妨将其更新一下（见 1.4 节）。

如果你使用的是树莓派 3 的话，那么实际上它已经内置了 Wi-Fi 网卡。

如果你使用的是老版树莓派，先在树莓派的 USB 接口上面安装一个兼容的 USB 无线网卡（大部分都是兼容的），然后就可以运行 Wi-Fi 配置工具（见图 2-4）进行配置了。此外，你还可以在树莓派的开始菜单的 Preferences 部分找到 Wi-Fi 配置工具。然后，单击 Scan 按钮搜索接入点。双击你要连接的接入点（你的家用集线器的），并在 PSK 字段输入密码。

最后，单击连接按钮便可接入网络。

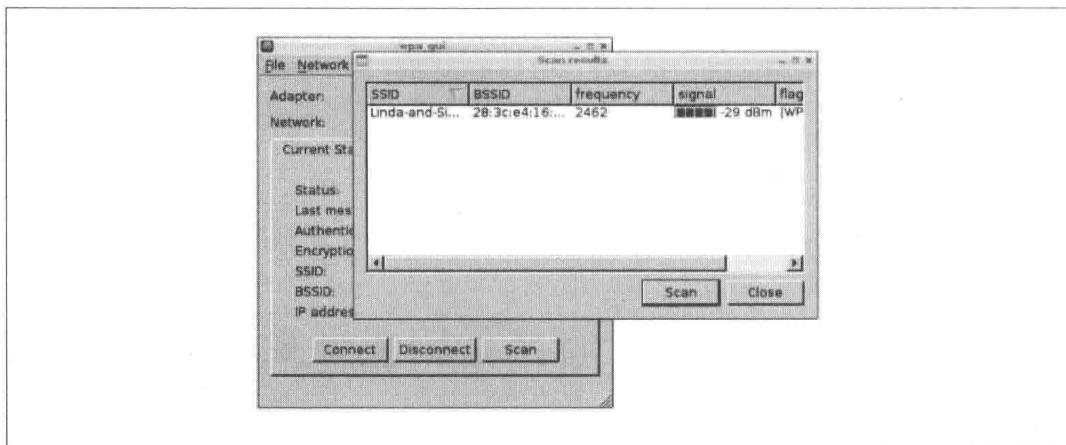


图 2-4 Wi-Fi 配置——正在扫描网络

进一步探讨

USB Wi-Fi 无线网卡比较费电，因此如果遇到树莓派意外重启或无法正确重启的情况，那么很可能需要使用一个更大功率的电源。你可以使用一个能提供 1.5A 或更大电流的电源。

如果你还打算使用鼠标与键盘，可能会遇到 USB 接口不够的情况，这时，可以求助于 USB 集线器。请选择自带电源的集线器，因为它可以缓解树莓派电源的压力。

如果你打算将树莓派用于媒体中心（见 4.1 节），还有一个设置页面，可供你使用 Wi-Fi 将媒体中心连接网络。

此外，你还可以直接使用命令行来设置无线网络连接。为此，首先要编辑 `/etc/network/interfaces` 文件，具体命令如下所示。

```
$ sudo nano /etc/network/interfaces
```

然后，在文件中找到与 `wlan0` 接口有关的部分，并将其修改为：

```
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
```

第一行内容的作用是规定当 USB Wi-Fi 无线网卡插入后，Wi-Fi 连接将自动启动。第二行内容的作用是规定让树莓派通过 DHCP 协议来分配 IP 地址。如果你想要使用静态 IP 地址，那么可以将单词 `dhcp` 换为 `static`，并添加 2.3 节中介绍的分配静态 IP 地址所需的相关内容。

最后一行用来规定客户端文件的位置。实际上，这个文件包含了无线网络的 SSID（网络名称）和 PSK（密码）。所以，接下来就要编辑这个文件了，具体命令如下所示。

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

修改文件时，只要根据你的无线网络设置 ssid 和 pskfoobar 便可。

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
    ssid="My-Network-Name"
    psk="My-password"
    proto=RSN
    key_mgmt=WPA-PSK
    pairwise=TKIP
    auth_alg=OPEN
}
```

为了让对文件所做的修改生效，需要重新启动树莓派。

参考资料

关于有线连接的介绍，请参考 2.1 节。如果需要了解兼容树莓派的 Wi-Fi 网卡的清单，请访问 http://elinux.org/Rpi_verifiedperipherals。关于配置有线网络的详细内容，请参考 2.1 节。

2.6 使用控制台线联网

面临问题

虽然没有网络连接可用，但是，你仍然希望能够从另一台计算机远程访问树莓派。

解决方案

使用控制台线来连接树莓派。

如果你打算以无外设的方式来使用树莓派的话，即不用键盘、鼠标或显示器，那么控制台线将是不二之选。控制台线的外观如图 2-5 所示，可以从 Adafruit (<https://www.adafruit.com/>) 网站购买。

控制台线的连接方式如下。

1. 将红色 (5V) 导线连接到 GPIO 接口左边缘上的 5V 引脚。
2. 将黑色 (GND) 导线连接至上面用到的 5V 引脚左边的 GND 引脚上面。
3. 将白色导线 (RX) 连接至树莓派的 GPIO 14 (TXD) 引脚上，它位于黑色导线左边。
4. 将绿色导线 (RX) 连接到树莓派上白色导线左边的 15 号引脚 (RXD) 上面。

如果你使用了不同的导线，电线颜色可能会有所不同，所以务必查看导线的相关文档，否则，可能会对树莓派造成损害。

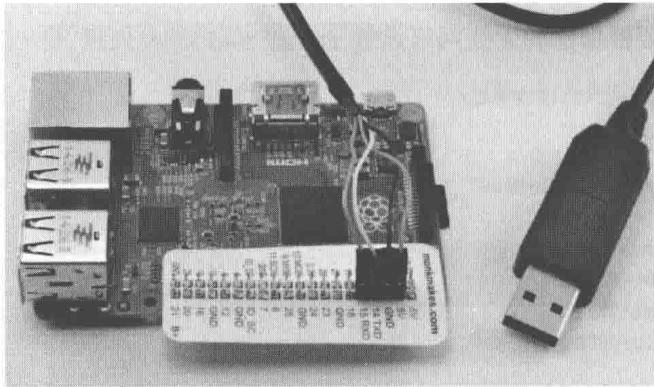


图 2-5 控制台线

需要注意的是 USB 线也能为红色导线提供 5V 电压，对于树莓派自身来说，这就够了；但是，如果连接了很多外设的话，就无法满足要求了。

如果使用的是 Windows 或者 Mac OS X 系统，那么你还需为 USB 线安装相应的驱动程序。对于 Windows 系统来说，相应的驱动程序的下载地址为 <http://bit.ly/1hAX6Mw>；对于 Mac OS Mac OS X 系统来说，相应的驱动程序下载地址为 <http://bit.ly/18oycat>。一般情况下，Linux 用户无需为这些线缆安装任何驱动程序。

要想从 Mac OS X 上连接树莓派，你需要打开终端，并输入如下的命令。

```
$ sudo cu -l /dev/cu.NoZAP-PL2303-00001004 -s 115200
```

具体的设备名称可能会有所不同，但是，如果你在 cu.P 后按 Tab 键，系统将自动补全该命令。在连接上之后，按回车键，就会看到树莓派的登录提示符（见图 2-6）。这里，默认用户名和密码分别是 pi 和 raspberry。

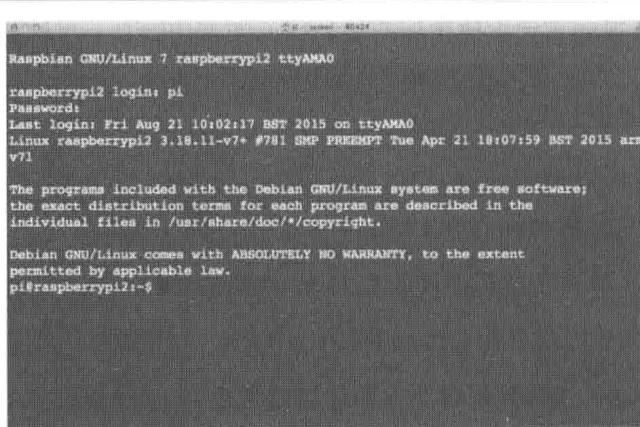


图 2-6 利用控制台线登录

如果你想从 Windows 机器上面连接树莓派的话，你需要下载一款名为 Putty (<http://www.putty.org>) 的终端软件。

当你运行 Putty 软件时，将“Connection type”改为 Serial，并把速率设为 115200。此外，你还需要把“Serial line”设置为控制台线所用的 COM 端口。它可能是 COM7，如果无法正常使用，可以通过 Windows 设备管理器来进行检查。

单击 Open 并按下回车键，终端会话就会启动，并给出登录提示符。

进一步探讨

如果你的树莓派想轻装上阵，控制台线将是不二之选，因为它不仅为树莓派提供了电源，还可以对其进行远程控制。

控制台线在 USB 端有一个芯片，提供了 USB-to-serial 接口。

有时候，为了使用控制台线需要为你的个人电脑安装相应的驱动程序，具体情况取决于你使用的操作系统。这里使用的控制台线是由 Adafruit 供应的（产品码 954）。你还可以使用任何 USB-to-serial 转换器，只要你电脑上面有相应的驱动程序即可。

如果你按照一组 GPIO 接口相应顺序将 4 个插座小心地胶合在一起，那么就可以轻松将这些插座插到正确的地方。

如果你使用了类似 Raspberry Leaf（参见 9.1 节）这样的 GPIO 模板的话，在 GPIO 接口中找到这些正确的位置是非常轻松的。

参考资料

如果你想了解串口控制台的更多知识，请参考 Adafruit 教程，地址为 <http://bit.ly/1bDL4vM>。同时，Adafruit 也出售控制台线。

2.7 利用 SSH 远程控制树莓派

面临问题

你想要从另一台计算机上，通过 SSH 连接一台远程树莓派。

解决方案

要想使用 SSH 连接树莓派，必须首先启用 SSH。对于较新版本的 Raspbian 来说，你可以使用树莓派的 Configuration tool（见图 2-7）来完成这项工作，该软件位于主菜单的 Preferences 中。点选 SSH 的复选框，单击 OK，系统将提示重新启动树莓派。

如果你使用的 Raspbian 版本较旧的话，可以使用 raspi_config 程序。为了启动该程序，只需在终端中键入下列命令即可。

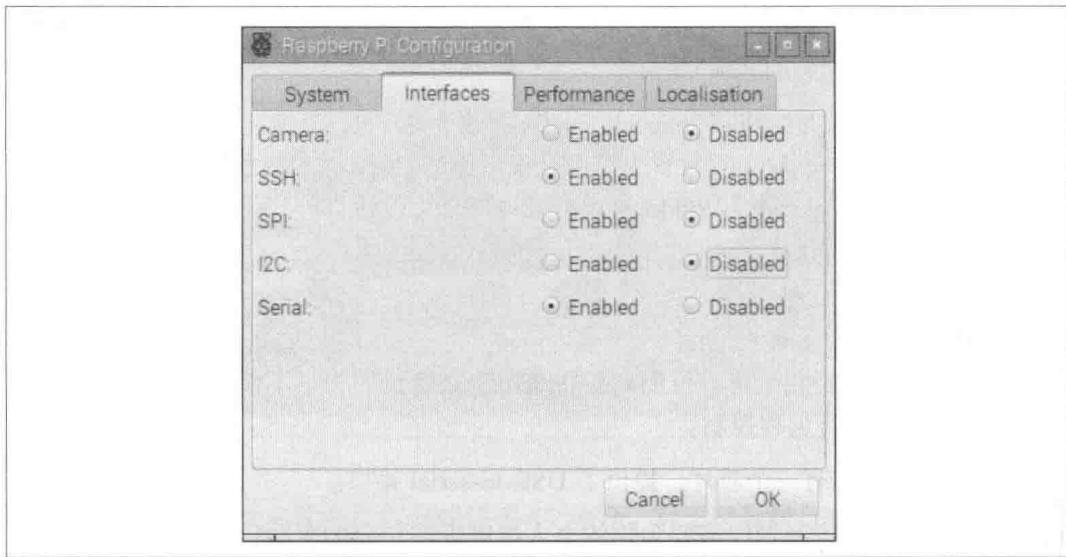


图 2-7 通过树莓派的 Configuration tool 启用 SSH

```
$ sudo raspi-config
```

向下滚动到 SSH 选项，并启用它。



对于较新版本的 Raspbian 来说，SSH 已经自动启用了，所以根本无需修改任何设置。

如果你想从一台安装了 Mac OS X 或 Linux 的计算机上面连接树莓派的话，只需要打开终端窗口，键入下列命令即可。

```
$ ssh 192.168.1.16 -l pi
```

其中，这里的 IP 地址（192.168.1.16）是树莓派的 IP 地址（参见 2.2 节）。这时，将提示输入密码，之后就可以登录到树莓派了（见图 2-8）。

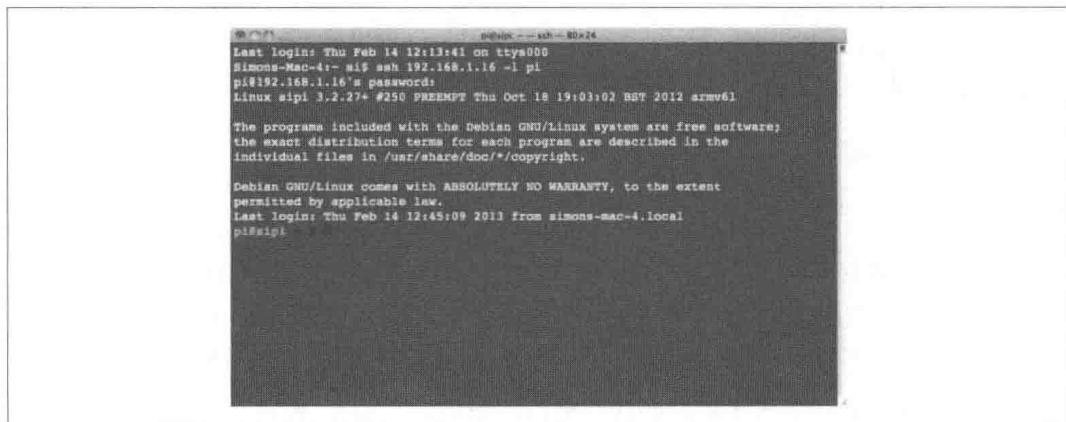


图 2-8 利用 SSH 登录

如果从 Windows 机器远程连接树莓派的话，需要使用 Putty(见 2.6 节) 来启动一个 SSH 会话。

进一步探讨

SSH 是连接远程计算机的一种常见方法，只要是树莓派本身支持的命令，都可以在 SSH 中使用。同时，就像它的名字所暗示的那样，这是一种安全的连接方式，因为通信是加密的。

这种方法唯一的缺点，可能在于它的运行环境是命令行，而非图形环境。如果你想远程访问树莓派的桌面环境的话，就需要使用 VNC (见 2.8 节) 和 RDP (见 2.9 节) 了。

参考资料

还请参考 <http://www.bit.ly/HRnse4> 提供的 Adafruit 教程。

2.8 利用 VNC 远程控制树莓派

面临问题

你需要通过 VNC 从 Mac OS X 或个人电脑 (Windows 或 Linux) 远程访问树莓派完整的 Raspbian 图形桌面。

解决方案

安装 VNC (Virtual Network Connection, VNC) 服务器。

在树莓派上打开终端会话 (或 SSH 会话)，然后运行下列命令。

```
$ sudo apt-get update  
$ sudo apt-get install tightvncserver
```

安装好 VNC 服务器之后，请使用下列命令来运行它。

```
$ vncserver :1
```

第一次运行时，会要求建立密码，也就是说，今后任何远程连接树莓派的人，必须输入该密码后才能获得访问权限。

为了从远程计算机连接树莓派，你需要安装 VNC 客户端。

RealVNC (<http://www.realvnc.com>) 是一个流行之选，可以很好地与 tightvnc 建立连接，并且支持 Windows、Linux 和 Mac OS X 系统。

当你在 Mac OS X 或个人电脑上运行客户端程序的时候，它还会要求输入你想连接的 VNC 服务器的 IP 地址，也就是你的树莓派的 IP 地址。

可以在 IP 地址后面输入 “:1”，用以表示你希望连接的桌面号为 1。

之后，会要求输入密码（见图 2-9）。请记住，这是指安装 `tightvncserver` 后设置的那个密码，并不要求与你的树莓派密码一致。

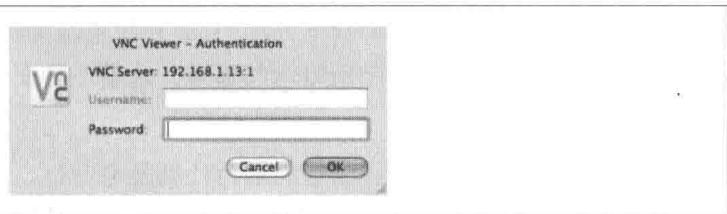


图 2-9 利用 VNC 登录

进一步探讨

虽然大部分工作都可以通过 SSH 进行，但是，有时候访问树莓派图形环境的能力是非常有用的。

如果你希望树莓派每次重启都会自动启动 VNC 服务器的话，可以借助下列命令来实现。

```
$ cd /home/pi  
$ cd .config  
$ mkdir autostart  
$ cd autostart  
$ nano tightvnc.desktop
```

将下列内容粘贴到编辑器窗口中。

```
[Desktop Entry]  
Type=Application  
Name=TightVNC  
Exec=vncserver :1  
StartupNotify=false
```

树莓派一旦设置为自动登录并进入桌面环境的话，VNC 服务器就会在重新启动后自动启动。

参考资料

请参考 <http://bit.ly/172gOL7> 提供的 Adafruit 教程。

请同时参考 2.9 节。

2.9 利用 RDP 远程控制树莓派

面临问题

你想从个人电脑或 Mac OS X 机器中通过 RDP 访问树莓派的 Raspbian 图形桌面。

解决方案

在树莓派上安装 Xrdp 软件，具体命令如下所示。

```
$ sudo apt-get update  
$ sudo apt-get install xrdp
```

一旦软件安装完毕，它就会自动启动 xrdp 服务，也就是说，每当树莓派重启时，该服务就会自动启动。

如果你的系统是 Windows 7 及其之后的版本，那么实际上已经包含了用于连接树莓派的 RDP 客户端了。它位于启动菜单的 All Programs/Accessories/Remote Desktop Connection 菜单中。对于较旧的 Windows 版本，你可以从 ModMyPi 下载客户端，地址为 <http://bit.ly/1TSIPf3>。

对于 Mac OS X 用户来说，可以从 Microsoft 公司网站上面下载为 Mac OS X 系统提供的 Microsoft RDP 客户端，地址为 <https://www.microsoft.com/en-gb/download/details.aspx?id=18140>。

Linux 机器所需的客户端可以从 <http://www.rdesktop.org/> 下载。

启动 RDP 客户端时，它会询问要连接到哪台计算机，这时输入树莓派的 IP 地址即可。之后，它会要求你提供平常登录树莓派所用的用户名和密码（见图 2-10），也就是说，除非已经修改了默认设置，否则的话用户名为 Pi，密码为 Raspberry。

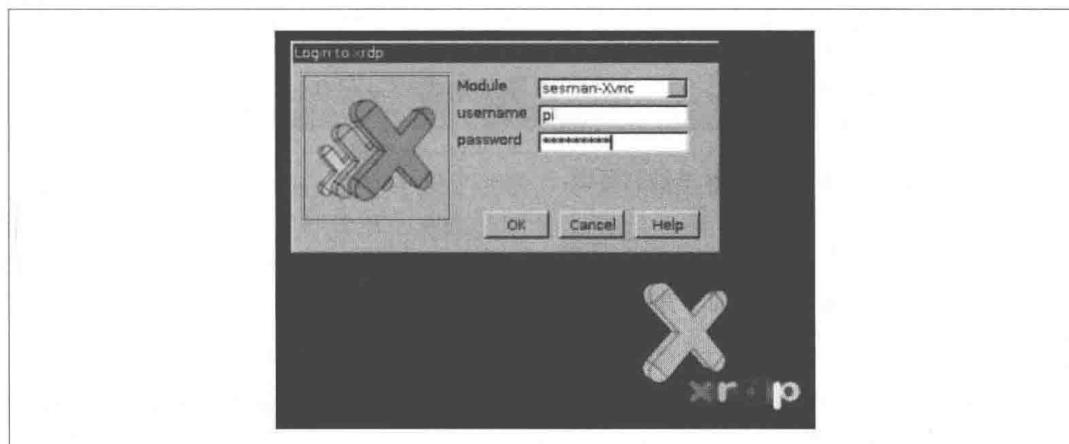


图 2-10 利用 RDP 登录

进一步探讨

RDP 的作用与 VNC 一样，不过更加高效，所以屏幕内容的刷新会更加流畅。它唯一的缺点是对于 Mac OS X 用户来说，它无法与 OS X 的共享屏幕功能搭配使用。

参考资料

请参考 2.8 节。

2.10 在 Mac 网络中实现文件共享

面临问题

你希望树莓派出现在 Mac OS X 的 Finder 列表中，从而可以使用 Finder 来连接树莓派，并浏览其文件系统。

解决方案

Mac OS X 操作系统本身就支持使用 Finder 通过网络浏览文件（见图 2-11）。不过，你必须对树莓派的配置做相应的修改，以便支持 OS X 浏览其文件系统。

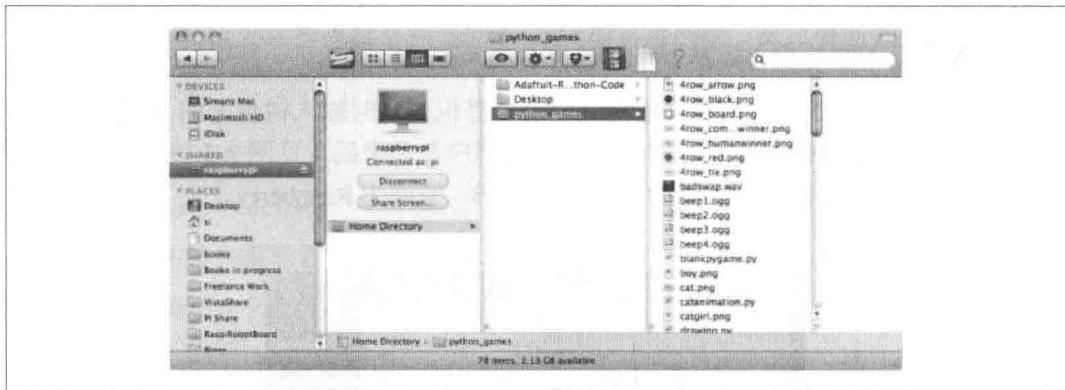


图 2-11 在 Mac OS X 的 Finder 中的树莓派

你需要知道树莓派的 IP 地址（见 2.2 节）。

现在，在树莓派上输入下列命令来安装 netatalk。

```
$ sudo apt-get install netatalk
```

然后，回到 Mac 上，在 Finder 菜单中选择 Go→Connect to Server，然后输入 `afp://192.168.1.16` 作为服务器地址（你要使用自己树莓派的 IP 地址替换这里的地址）。然后，单击 Connect，系统就会提示登录。在出现登录提示符之前，我不得不重启了自己的树莓派。

使用树莓派默认的用户 pi 和相应密码登录后，Finder 就会显示树莓派 home 目录下面的内容。

对于树莓派来说，还有一些配置需要修改。

```
$ sudo apt-get install avahi-daemon  
$ sudo update-rc.d avahi-daemon defaults
```

然后，输入下列命令。

```
$ sudo nano /etc/avahi/services/afpd.service
```

将下列内容粘贴到该文件中。

```
<?xml version="1.0" standalone='no'?><!--*-nxml-*-->  
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">  
<service-group>  
    <name replace-wildcards="yes">%h</name>  
    <service>  
        <type>_afpovertcp._tcp</type>  
        <port>548</port>  
    </service>  
</service-group>
```

为了让它作为守护进程运行，可以使用如下所示的命令：

```
$ sudo /etc/init.d/avahi-daemon restart
```

回到 Mac，这时 Finder 里面应该可以看到自己的树莓派了。

进一步探讨

在树莓派和 Mac 之间轻松传递文件的能力是非常有用的功能。这样的话，即使树莓派没有外接鼠标、键盘和显示器，也能使用其上的文件了。

你还可以在 Mac 上面打开树莓派中的文件，就如同它们就在 Mac 上面一样。它带来的好处就是你能够使用 TextMate 或者自己喜欢的 OS X 文本编辑器来编辑树莓派上面的文件。

如果你的系统是 Windows 或者 Linux 的话，你还可以配置树莓派，让其作为网络附属存储 NAS，从而实现文件共享，具体参见 2.12 节。

参考资料

本节改编自 <http://bit.ly/HRnOBv> 提供的教程，该教程出自 Matt Richardson 和 Shawn Wallace 所著的 *Getting Started with Raspberry Pi* (O'Reilly)。

2.11 在 Mac 上共享树莓派的屏幕

面临问题

你已经建立了 VNC，但是希望共享树莓派的屏幕，就像共享网络中的其他 Mac OS X 设备那样。

解决方案

首先，按照 2.8 节介绍的方法安装 VNC。同时，你还要看完 2.10 节才行。

然后，输入下列命令。

```
$ sudo nano /etc/avahi/services/rfb.service
```

然后，将下列内容粘贴到编辑器中。

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">%h</name>
  <service>
    <type>_rfb._tcp</type>
    <port>5901</port>
  </service>
</service-group>
```

现在，输入下列命令。

```
$ sudo /etc/init.d/avahi-daemon restart
```

这时，你将看到如图 2-12 所示的 Share Screen 选项。当提示输入密码时，请输入为 VNC 所设置的密码，而非树莓派的密码。



图 2-12 利用 Mac OS X 的 Finder 实现树莓派屏幕共享

进一步探讨

利用本节介绍的方法，可以简化分享树莓派屏幕的过程。

如果网络中有多个树莓派的话，就需要分别给它们取不同的名称，以便能够在网络中标识它们（见 2.4 节）。

如果你使用的是 Windows 或 Linux 系统，那么还可以利用 VNC 来连接树莓派（见 2.8 节）。

参考资料

本节改编自 <http://bit.ly/HRnOBv> 提供的教程，该教程出自 Matt Richardson 和 Shawn Wallace 所著的 *Getting Started with Raspberry Pi* (O'Reilly)。

2.12 将树莓派用作网络存储服务器

面临问题

你想要把树莓派用作网络存储服务器，即从网络中的电脑访问树莓派连接的大容量 USB 驱动器。

解决方案

这个问题的解决方案是安装配置 Samba。为此，可以使用如下所示的命令。

```
$ sudo apt-get install samba  
$ sudo apt-get install samba-common-bin
```

现在，请将 USB 硬盘驱动器接入树莓派。该驱动器将自动挂载到/media 文件夹下面。为了检查是否挂载到位，可以使用下列命令进行检查。

```
$ cd /media  
$ ls
```

这时，该驱动器应该被显示出来，其名称为格式化它时所取的那个。树莓派每次重启的时候，都会自动挂载这个驱动器。

现在，你需要配置 Samba，以便该驱动器可以在网络上共享。为此，首先要添加一个 Samba 用户 (pi)。请输入下列命令，并提供一个密码。

```
$ sudo smbpasswd -a pi  
New SMB password:  
Retype new SMB password:  
Added user pi.
```

接下来，需要修改/etc/samba/smb.conf 文件，具体如下所示。

```
$ sudo nano /etc/samba/smb.conf
```

需要寻找的第一行位于文件顶部附近。

```
workgroup = WORKGROUP
```

如果你打算通过 Windows 机器连接时，只需要修改这里就行了。这里应该改成 Windows 工作组的名称。对于 Windows XP 系统来说，默认的工作组名称是 MSHOME；对于后

续的 Windows 版本来说，其名称为 HOME。（但是，还是检查一下自己的 Windows 网络为好。）

下一个需要修改的地方位于该文件下方的 Authentication 部分。

请找到下面这行内容。

```
# security = user
```

删除前面的注释符号#，启用安全保护。

最后，向下滚动到文件尾部，并添加如下所示内容。

```
[USB]
path = /media/NAS
comment = NAS Drive
valid users = pi
writeable = yes
browseable = yes
create mask = 0777
public = yes
```

保存文件，并输入下列命令重启 Samba。

```
$ sudo /etc/init.d/samba restart
```

如果一切正常的话，USB 驱动器现在就能够实现网络共享了。

进一步探讨

如果要从 Mac OS X 连接驱动器的话，只需在 Finder 菜单中选择 Go→connect to Server。然后，在 Server Address 字段中输入 smb://raspberrypi/USB，这时就会出现一个登录对话框，在此，你需要将用户名改为 pi（见图 2-13）。

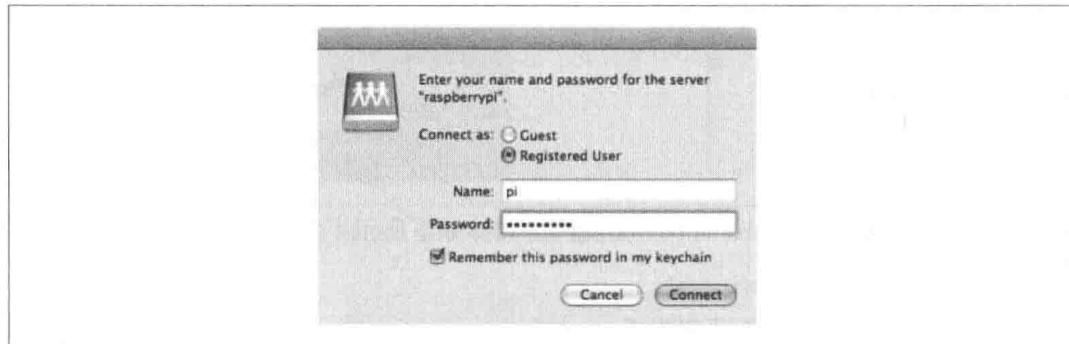


图 2-13 使用 Mac OS X Finder 连接 NAS

如果你想从 Windows 机器上面连接 NAS 的话，见具体过程取决于 Windows 的版本情况。不过，基本过程是一致的，就是需要输入网络地址，该地址应该为\\raspberrypi\\USB

(见图 2-14)。

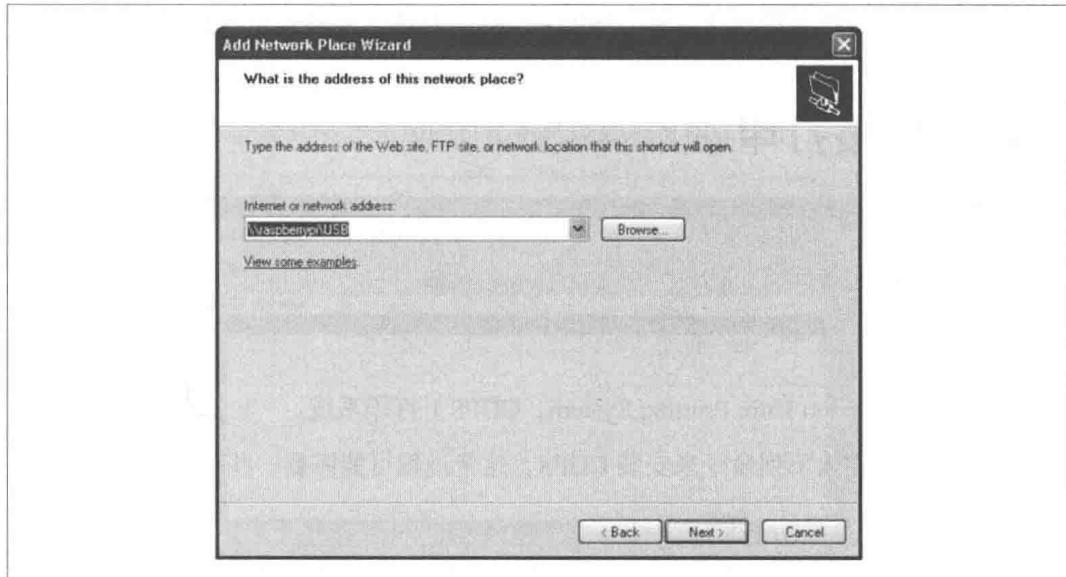


图 2-14 从 Windows 系统连接 NAS

之后，输入正确的用户名和密码，就可以使用 NAS 磁盘了（见图 2-15）。

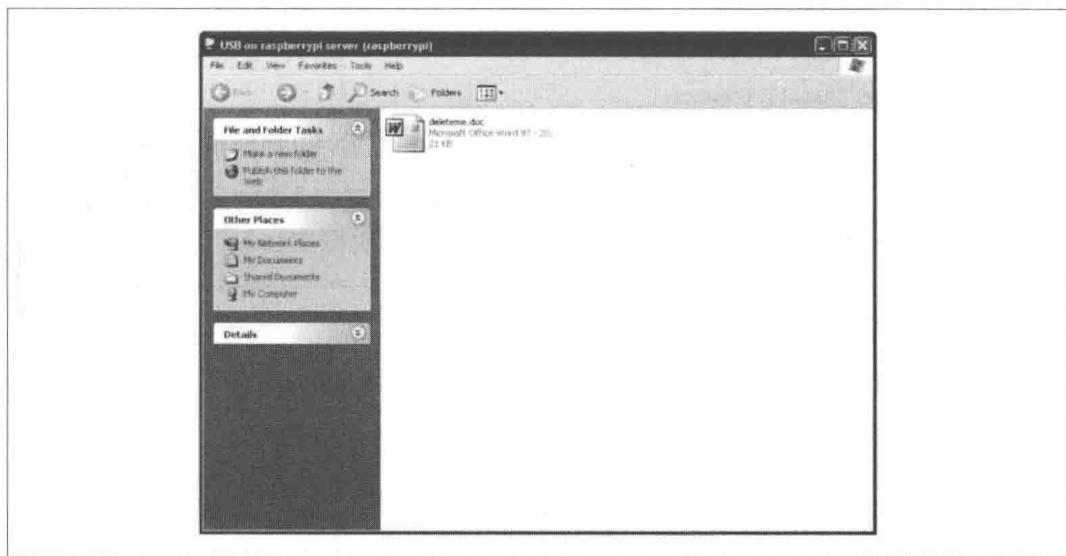


图 2-15 在 Windows 上浏览 NAS

如果你是 Linux 用户，可以使用下列命令来挂载 NAS 驱动器。

```
$ sudo mkdir /pishare  
$ sudo smbmount -o username=pi,password=raspberry //192.168.1.16/USB /pishare
```

参考资料

你也许想要修改树莓派的网络名称，例如 piNAS，具体方法请参考 2.4 节。

2.13 网络打印

面临问题

你想要让树莓派进行网络打印。

解决方案

使用 CUPS (Common Unix Printing System, CUPS) 打印系统。

首先，请在终端输入下列命令来安装 CUPS，这个过程可能需要一些时间。

```
$ sudo apt-get install cups
```

为了使用 CUPS，需要管理员权限，为此，可以使用下列命令提升权限。

```
$ sudo usermod -a -G lpadmin pi
```

CUPS 提供了一个 Web 配置接口。无论是 Midori，还是 Dillo 浏览器，都可以用来浏览 CUPS 的管理页面，但是，Iceweasel 要更好一些。

安装 Iceweasel 浏览器的命令如下所示。

```
$ sudo apt-get install iceweasel
```

你可以从 Start 菜单的 Internet 组中启动 Iceweasel 浏览器，然后打开地址 <http://localhost:631>。

找到 Administration 标签，选择 Add Printer 选项。这时将显示一个打印机清单，列出所有网络上面和直接连接到树莓派的 USB 端口的打印机（见图 2-16）。

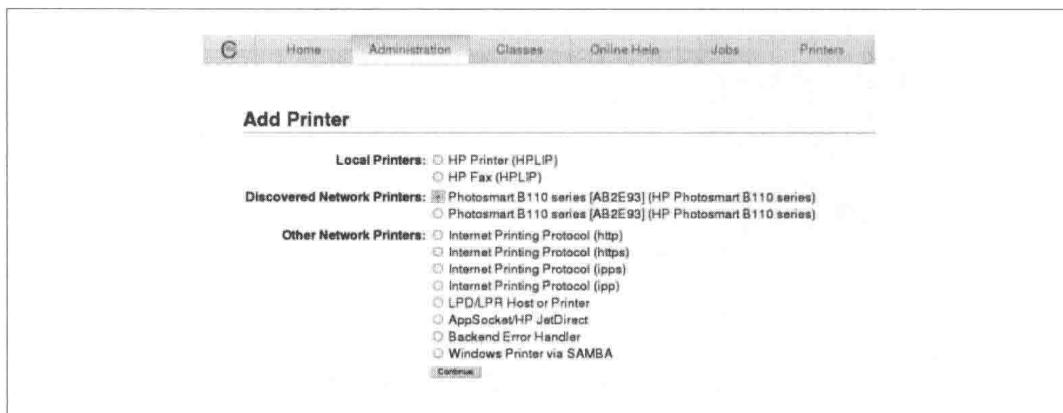


图 2-16 利用 CUPS 查找打印机

然后，按照一系列的对话框的提示就可以完成打印机的配置了。

进一步探讨

完成所有步骤之后，你可以利用 AbiWord 来测试打印机（参见 4.2 节）。输入一些文本并进行打印时，你会看到刚才添加的打印机已经处于可用状态了（见图 2-17）。

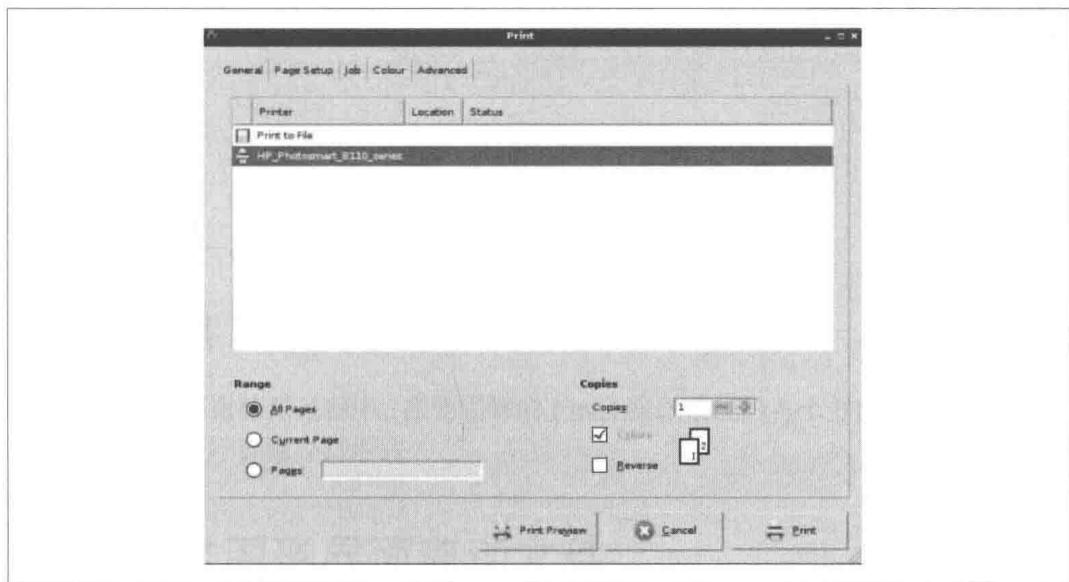


图 2-17 从 AbiWord 进行打印

参考资料

你可以访问 CUPS 的官方网站 (<http://www.cups.org>)。

操作系统

3.0 引言

本章介绍在使用树莓派过程中，需要了解的 Linux 操作系统的各个方面相关知识，并且很多时候都要用到命令行形式。

3.1 通过图形界面移动文件

面临问题

你希望通过图形界面来移动文件，就像在 Mac OS X 或者个人电脑上面那样。

解决方案

使用文件管理器（File Manager）。

你可以在 Accessories 分组的 Start 菜单中找到这个程序（见图 3-1）。

使用 File Manager，你可以将文件或目录从一个目录拖到另一个目录，或使用 Edit 菜单，将一个地方的文件复制粘贴到另一个地方。这里的操作方法与 Windows 的文件管理器或苹果操作系统的 Finder 的用法大体相同。

进一步探讨

File Manager 的左边显示的是已经挂载好的各卷，因此，当你连接一个 USB 闪存驱动器或者外部 USB 驱动器的时候，它们就会显示在这个地方。

中间区域展示的是当前文件夹中的文件，你既可以通过单击工具栏的按钮，也可以在上部的文件路径区域输入具体的位置进行浏览。

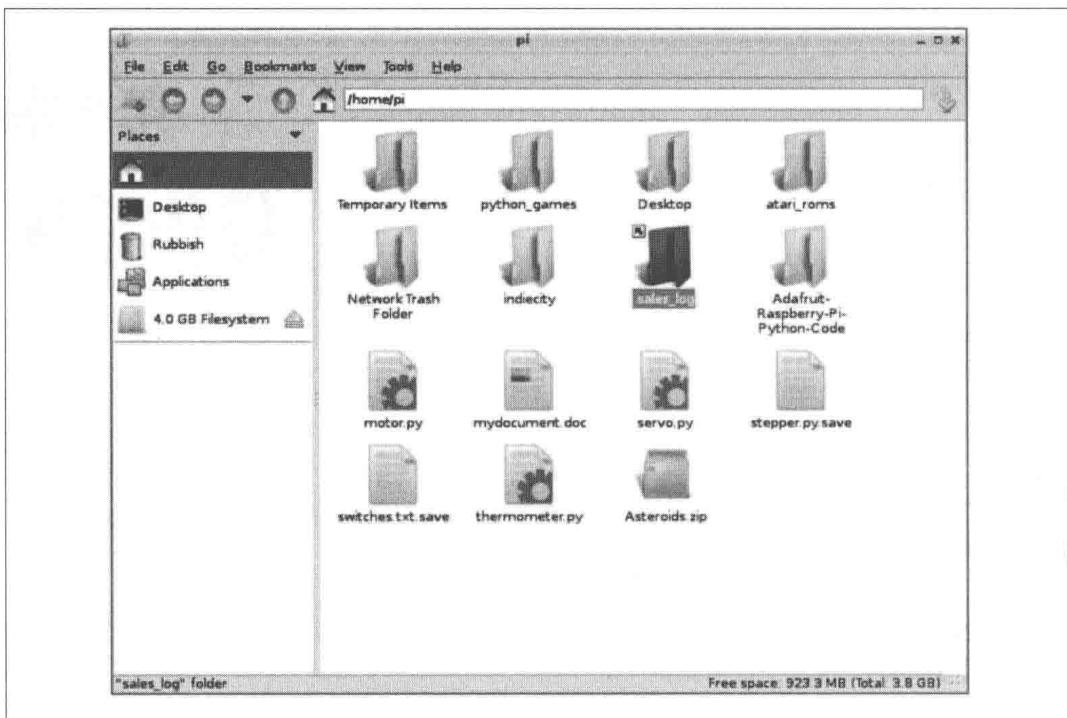


图 3-1 文件管理器

右键单击文件，可以看到当前文件支持的各种操作（见图 3-2）。

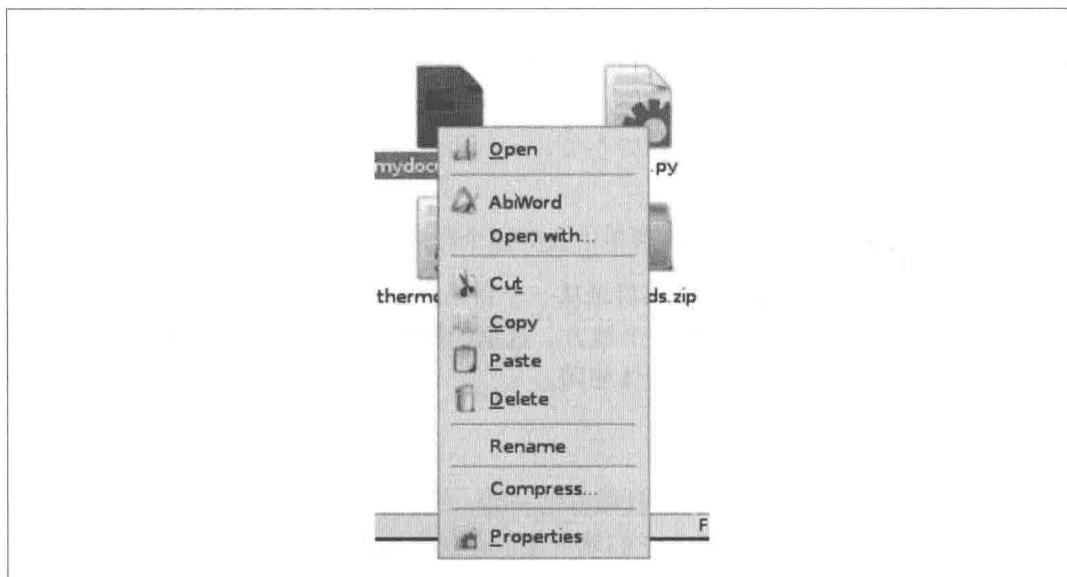


图 3-2 文件管理器的选项

参考资料

请同时参考 3.4 节的内容。

3.2 启动一个终端会话

面临问题

当你使用树莓派的时候，你需要在终端中输入文本命令。

解决方案

单击树莓派桌面顶部的 LX 图标（该图标看起来像一个黑色的计算机显示器），或者从 Start 菜单的 Accessories 分组中选中 Terminal 菜单项（见图 3-3）。



图 3-3 打开 LX 终端

进一步探讨

LX 终端启动时，它将会设置到你当前的主目录（/home/pi）下面。

如果需要的话，可以打开多个终端会话。你可以在不同的目录下打开多个终端，这样做的好处是无需通过 cd 命令频繁地在目录之间来回切换了。

参考资料

在下一节（3.3 节）中，你将了解如何利用终端来浏览目录的结构。

3.3 利用终端浏览文件系统

面临问题

你希望了解如何利用终端来修改目录，以及如何通过终端来浏览文件系统。

解决方案

用来浏览文件系统的主要命令是 cd (change directory)。在 cd 命令的后面，你必须规定你要切换到的目录。这时，你可以使用当前文件夹中的相对路径，或者文件系统内的绝对路径来表示你要到达的目录。

要想了解当前文件夹的位置，你可以借助于命令 pwd (print working directory)。

进一步探讨

你可以通过尝试下面的例子来进行学习。首先，打开一个终端会话，你将看到下面的提示符。

```
pi@raspberrypi ~ $
```

每次执行命令时，你都能够看到这个提示符 (pi@raspberrypi ~ \$)，它的作用是提示你的用户名 (pi) 以及计算机名称 (raspberrypi)。字符~是你主目录 (/home/pi) 的简写形式。因此，无论当前位置如何，你都可以通过下列命令将当前位置变为你的主目录。

```
$ cd ~
```



在本书中，字符\$表示需要输入命令的行的开始位置，但是命令行的返回结果之前是没有任何前缀的，这就跟树莓派屏幕上看到的一样。

你可以通过 pwd 命令来验证一下当前目录是否已经确实变更为目录。

```
$ pwd  
/home/pi
```

如果你想在目录结构中上移一级的话，则可以通过在 cd 命令后面加上一个特殊值..(两个点号) 来实现，具体如下所示。

```
$ cd ..  
$ pwd  
/home
```

也许你已经看出来了，特定文件或目录都是由/分隔开的单词所组成的。因此，整个文件系统的根目录就是/，若要访问/下面的主目录的话，就需要用/home/表示。然后，若要在主目录中定位 pi 文件夹的话，则需要使用/home/pi/。路径中最后面的/是可以忽略的。

路径既可以是绝对路径（以/开头，并给出从根目录起的完整路径），也可以是针对当前工作目录的相对路径，在这种情况下，它们就不会以/开头了。

对于主目录下面的文件，虽然具有完整的读写权限，但是当切换到存放系统文件和应用程序的目录下面的时候，你对某些文件的访问就只剩下读权限了。当然，你可以逾越这一限定（见 3.11 节），但是请务必小心。

你可以利用下面所示的命令来检查根目录的结构。

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

命令 ls (list) 展示了根目录/下面的所有文件和目录。你会发现，home 目录也赫然在列，而你刚才就是从这个目录切换过来的。

现在，你可以通过下面的命令切换到其中一个目录下面。

```
$ cd etc
$ ls
adduser.conf      hosts.deny    polkit-1
alternatives      hp           profile
apm              iceweasel    profile.d
apparmor.d        idmapd.conf  protocols
apt              ifplugd     pulse
asound.conf       init         python
```

这里有些事情要注意，首先，如果列出的文件和文件夹太多的话，无法在屏幕上面一次显示全。这时，可以使用终端窗口的滚动条上下移动。其次，文件和文件夹的颜色也别具深意：文件显示为白色，目录则是蓝色的。

除非你对打字情有独钟，否则，你使用 Tab 键的话会更加轻松。当你开始输入一个文件的名字的时候，可以通过 Tab 键的自动补全功能来完成文件名的输入。举例来说，如果你想切换到 network 目录，可以输入 cd netw，然后按下 Tab 键。由于 netw 已经足以唯一确定出某个文件或目录了，所以按下 Tab 键就会自动补全它。

如果输入的内容尚不足以唯一确定出一个文件或目录名，那么，再次按下 Tab 键就会展示与已经输入的内容相匹配的所有可能的选择项。所以，只输入 net 便按下 Tab 键的话，你会看到如下所示内容。

```
$ cd net
netatalk/ network/
```

你可以在 ls 命令后面提供更多的参数来缩小要展示的内容的范围。切换至目录/etc 下面，并输入下列命令。

```
$ ls f*
fake-hwclock.data fb.modes fstab fuse.conf
```

```
fonts:  
conf.avail conf.d fonts.conf fonts.dtd  
  
fomatic:  
defaultspooler direct filter.conf  
  
fstab.d:  
pi@raspberrypi /etc $
```

字符*是一个通配符。如果在命令 ls 指定参数 f*的话，就会展示所有以 f 开头的文件和目录。有利的是，命令的返回结果首先给出的是/etc 目录下面所有以 f 开头的文件，然后才是所有以 f 开头的文件夹。

通配符的常见用法是列出具有特定扩展名的所有文件（例如，ls *.docx）。就像许多其他操作系统一样，Linux 做法的一个惯例是如果文件名以句点作为前缀，则表示该文件对用户来说应该是不可见的。所以，使用 ls 命令时，是看不到以这种方式命名的文件和文件夹的，除非为 ls 命令指定参数-a。例如：

```
$ cd ~  
$ ls -a  
.  
..  
Adafruit-Raspberry-Pi-Python-Code .dmrc  
.advance .emulationstation .pulse  
.AppleDB .fltk .pulse-cookie  
.AppleDesktop .fontconfig .python_games  
.AppleDouble .gstreamer-0.10 .sales_log  
Asteroids.zip .gvfs .servo.py  
atari_roms indiecity .stella  
.bash_history .local stepper.py.save  
.bash_logout motor.py switches.txt.save  
.bashrc .mozilla .Temporary Items  
.cache mydocument.doc thermometer.py  
.config Network Trash Folder .vnc  
.dbus .profile .Xauthority  
.
```

如您所见，home 目录下面的内容大部分是隐藏文件和文件夹。

参考资料

请同时参考 3.13 节。

3.4 复制文件或文件夹

面临问题

你想通过终端会话复制文件。

解决方案

使用 cp 命令复制文件和目录。

进一步探讨

当然，你可以通过 File Manager 的复制粘贴选项来复制文件（见 3.1 节）。

使用终端会话来复制文件的最简单的例子就是在工作目录复制文件。在 cp 命令后面的第一个参数是待复制的文件，第二个参数是复制后新文件的名称。

例如，下面的例子将新建一个名为 myile.txt 的文件，然后为该文件建立一个副本，并命名为 myile2.txt。至于利用>命令创建文件的更多技巧，请参考 3.8 节。

```
$ echo "hello" > myfile.txt  
$ ls  
myfile.txt  
$ cp myfile.txt myfile2.txt  
$ ls  
myfile.txt  myfile2.txt
```

当然，虽然这里的两个文件的路径都是本地的当前工作目录，但实际上这些路径可以是文件系统中的任意目录，只要你对其具有写权限即可。下面的例子将会把源文件复制到/tmp 目录下面，该目录是用来存放临时文件的。所以，不要将任何重要的文件放到这个目录下面。

```
$ cp myfile.txt /tmp
```

需要注意的是，这里没有指定新文件的名称，而是只给出了新文件所在的目录。这样的话，就会在/tmp 目录下面为 myile.txt 创建一个同名的副本，即 myile.tmp。

有时候，需要复制的文件不止一个，而是要复制整个目录下面的文件和尽可能多的目录。为了复制这种目录，你需要使用-r 选项进行递归。这样就能够复制该目录及其所有的内容。

```
$ cp -r mydirectory mydirectory2
```

无论你想复制文件，还是文件夹，只要你权限不够的话，该命令将会在输出结果中指出来。为此，你要么修改复制操作的目标文件夹的权限（见 3.13 节），要么以超级用户的身份来复制这些文件（见 3.11 节）。

参考资料

请同时参考 3.5 节和 3.10 节。

3.5 重命名文件和文件夹

面临问题

你希望通过终端会话来为文件重命名。

解决方案

利用 mv 命令为文件和目录重命名。

进一步探讨

Mv 命令的用法与 cp 命令非常类似，只不过这里的文件是要进行重命名的，而不是进行复制的。

例如，如果要将一个文件的名称从 my_ile.txt 改为 my_ile.rtf 的话，只需要输入下列命令即可。

```
$ mv my_file.txt my_file.rtf
```

目录的重命名也同样很简单，并且不需要像复制文件时那样添加递归选项-r，因为修改目录的名称就暗含了把该目录下面的所有内容全部放入重命名后的目录下面。

参考资料

请同时参考 3.4 节和 3.10 节。

3.6 编辑文件

面临问题

你想要在命令行下面通过一个编辑器来编辑配置文件。

解决方案

使用编辑器 nano，该编辑器存在于大部分树莓派的发行包中。

进一步探讨

若要使用 nano，只需要简单键入 nano，并在后面指定待编辑的文件的名称或路径。如果文件不存在的话，从编辑器保存文件时会创建该文件。不过，前提条件是你对将要写入文件的目录具有写权限才行。

在你的 home 目录下面，输入命令 nano my_file.txt 就可以编辑或创建文件 nano my_ile.txt。图 3-4 展示了运行中的 nano。

你无法利用鼠标来给光标定位，相反，你可以使用方向键。

在上图屏幕的下方列出了许多命令，你可以在按住 Ctrl 键之后，通过按下其他字母键来访问它们。这里的大多数命令不是很常用。大部分情况下，你可能需要用到的命令有如下几种。



图 3-4 利用 nano 编辑文件

Ctrl-X

退出。nano 在退出之前，会提示你保存文件。

Ctrl-V

下一页。可以看成是鼠标下移。该命令可以让你以每次一屏的方式浏览大型文件。

Ctrl-Y

前一页。

Ctrl-W

搜索，可以用来查找一段文本。

此外，这里还有一些简单的剪切粘贴类型的选项，不过在实际使用过程中，更加简单的方法是通过单击右键，使用弹出菜单中的普通剪贴板（见图 3-5）。



图 3-5 nano 中的剪贴板

使用这个剪贴板还允许你在其他窗口（比如浏览器）之间复制粘贴文本。

如果你想要保存对文件的修改并退出 nano 的话，可以使用 Ctrl-X。然后，输入 Y 以确认保存文件。这样，nano 就会以显示的文件名作为默认名来保存文件。按回车键保存并退出。

如果你想撤销所做的修改，那么可以键入 N，而不是 Y。

参考资料

个人喜好不同，所以不同的人会钟情于不同的浏览器。对于 Linux 来说，还有许多其他类型的编辑器也可以很好地用于树莓派。在 Linux 的世界中，vim(vi improved)拥有大量的追随者。这个编辑器自然也包含在各种流行的树莓派发行版中。但是，对于初学者来说，这个编辑器不太容易掌握。你可以像使用 nano 那样来使用它，只要把命令 nano 替换为 vi 即可。如果你想了解 vim 的更多细节，请参考 http://newbiedoc.sourceforge.net/text_editing/vim.html.en。

3.7 查看文件内容

面临问题

你想查看而非编辑一个小型文件的内容。

解决方案

使用命令 cat 或者命令 more 来查看文件内容。

举例来说：

```
$ more myfile.txt
This file contains
some text
```

进一步探讨

命令 cat 会显示文件的所有内容，即使文件内容较长，一屏放不下的情况下也是如此。

命令 more 一次显示一屏的文本，按空格键后会继续显示下一屏。

参考资料

你还可以使用 cat 命令来串联（连接）多个文件（见 3.30 节）。

与 more 有关的另一个流行命令是 less。这两个命令非常相似，不过 less 命令不仅可以从前往后浏览文件，而且还允许以从后往前的顺序查看文件。

3.8 不借助编辑器的情况下创建文件

面临问题

你想创建一个只有一行内容的文件，但是又不想使用编辑器。

解决方案

利用>和 echo 命令将命令行中的内容重定向到文件中。

举例来说：

```
$ echo "file contents here" > test.txt  
$ more test.txt  
file contents here
```



命令>会覆盖已有的文件，所以使用的时候务必谨慎。

进一步探讨

这种方法对于快速创建文件来说非常有用。

参考资料

在不借助于编辑器的情况下，还有许多命令可以用来查看文件内容，具体可以参考 3.7 节。

为了使用>来捕获其他类型的系统输出，请参考 3.29 节。

3.9 创建目录

面临问题

你想要通过终端新建一个目录。

解决方案

命令 mkdir 可以创建一个新的目录。

进一步探讨

若要新建一个目录，可以使用 mkdir 命令。你可以像下面这样来进行尝试。

```
$ cd ~
```

```
$ mkdir my_directory  
$ cd my_directory  
$ ls
```

如果你想在某个目录下面新建目录，你必须具有这个目录的写权限。

参考资料

关于利用终端浏览文件系统的基础知识，请参考 3.3 节。

3.10 删除文件或目录

面临问题

你想通过终端来删除文件或目录。

解决方案

命令 `rm` (`remove`) 可以用来删除文件或目录及其内容。在使用这个命令的时候，要极其谨慎。

进一步探讨

删除单个文件不仅简单，而且也比较安全。下面通过实例讲解从 `home` 目录下面删除文件 `my_file.txt`，具体命令如下所示。

```
$ cd ~  
$ rm my_file.txt  
$ ls
```

如果你想在某个目录下面执行删除操作的话，你需要具有该目录的写权限。

在删除文件的时候，你还可以使用*通配符。下面的命令将删除所有文件名以 `my_file` 开头的文件。在当前目录下的：

```
$ rm my_file.*
```

此外，你还可以使用下列命令来删除该目录下面的所有文件。

```
$ rm *
```

如果你想以递归的方式来删除一个目录，以及该目录下面所有的内容，即包括其中所有目录的话，那么你可以使用`-r` 选项。

```
$ rm -r mydir
```



在使用终端窗口删除文件的时候，一定记住你无法从回收站恢复这些文件。同时，通常它也不会提供确认选项，而是直接执行删除操作。在与 `sudo` 结合使用的情况下，该命令可能会造成无可挽回的损失（参见 3.11 节）。

参考资料

请参考 3.3 节。

如果担心意外删除文件或者文件夹的话，则你也可以通过为该命令建立别名的方式（参见 3.34 节），强制 rm 命令在执行前先进行确认。

3.11 以超级用户权限执行任务

面临问题

有些命令，如果没有足够的权限是无法使用的。所以，你需要以超级用户的身份来执行这些命令。

解决方案

命令 sudo (substitute user do) 可以让你以超级用户的身份来执行命令。为此，你只需要在这些命令的前面加上 sudo 即可。

进一步探讨

你需要命令行完成的大部分任务，其实都不需要超级用户权限。最常见的需要超级用户权限的任务是安装软件或者编辑配置文件。

例如，当你尝试使用 apt-get update 命令时，可能会收到多条由于权限不足而拒绝执行的消息。

```
$ apt-get update
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?
```

最后一条消息，即“are you root?” 才是真正的拒绝原因。如果你在同一命令前面加上 sudo 的话，它就能正常工作了。

```
$ sudo apt-get update
Get:1 http://mirrordirector.raspbian.org wheezy InRelease [12.5 kB]
Hit http://archive.raspberrypi.org wheezy InRelease
Get:2 http://mirrordirector.raspbian.org wheezy/main Sources [6,241 kB]
Hit http://archive.raspberrypi.org wheezy/main armhf Packages
Ign http://archive.raspberrypi.org wheezy/main Translation-en_GB
Ign http://archive.raspberrypi.org wheezy/main Translation-en
40% [2 Sources 2,504 kB/6,241 kB 40%]
```

如果你有大量命令需要以超级用户权限执行，而你又不希望每次执行时都在它们前面加上前缀 sudo 的话，可以使用下列命令。

```
$ sudo sh
#
```

请注意这里的提示符的变化情况：从\$变为#。这样一来，所有后面的命令都会以超级用户的身份执行。如果你想恢复到普通用户身份的话，可以使用下列命令。

```
# exit  
$
```

参考资料

若想了解更多文件权限方面的内容，请参考 3.12 节。

若要使用 apt-get 命令安装软件的话，请参考 3.16 节。

3.12 理解文件权限

面临问题

当列出文件时，文件名后面会伴有一些奇怪的字符，你想知道这些字符的含义。

解决方案

为了了解文件和目录的权限和所属关系，可以在 ls 命令后面加上-l 选项。

进一步探讨

执行命令 ls -l，你将看到类似下面这样的输出结果。

```
$ ls -l  
total 16  
-rw-r--r-- 1 pi pi 5 Apr 23 15:23 file1.txt  
-rw-r--r-- 1 pi pi 5 Apr 23 15:23 file2.txt  
-rw-r--r-- 1 pi pi 5 Apr 23 15:23 file3.txt  
drwxr-xr-x 2 pi pi 4096 Apr 23 15:23 mydir
```

图 3-6 展示了所列出的信息的各个不同部分，其中第一部分就包含了权限信息。在第二列中，数字 1（图中标示为“Files”）表示包含的文件数量。这个字段只有在列出的条目涉及目录的时候才有意义，对于文件来说，该字段的值只能是 1。接下来的两个条目（都是 pi）表示文件的属主和组。表示文件大小的条目（第 5 列）指示文件的大小，单位为字节。修改日期在每次编辑或修改过后都会随之变化。最后一个条目是文件或目录的实际名称。

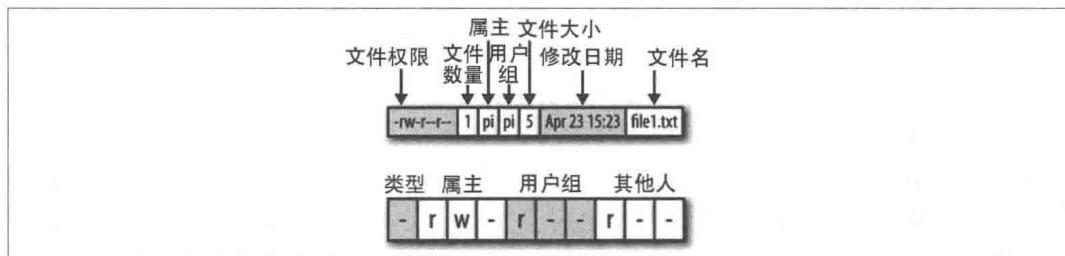


图 3-6 文件权限

文件权限分为 4 段（类型、属主、组和其他人）。

第一段是文件类型，对于目录来说，它的值为字符 d，对于文件来说，它的值是-。

下一段由 3 个字符组成，用来指示该文件的属主的具体权限。每个字符都是一个开关标志，所以，如果属主具有读权限的话，那么第一个字符位置处的值应该是 r，如果它具有写权限的话，那么在第二个字符所在位置的值应该是 w。如果文件属主对其具有执行权限的话，该段的第三个字符为 x（就本例而言，这里是-）。

第三段也有同样的 3 个开关标志，不过这些权限都是针对组内所有用户而言的。用户可以划分到不同的组。所以，就本例而言，该文件同时归属于用户 pi 和组 pi。因此，组 pi 内的其他成员同样具有该文件赋予该组的相关权限。

最后一段规定了其他用户，即除了用户 pi 和组 pi 之外的用户，对该文件所具有的权限。

由于大部分人都是仅以用户 Pi 的身份来使用树莓派的，所以他们最关心的权限都集中在第一段中。

参考资料

若要修改文件的权限，请参考 3.13 节。

3.13 修改文件的权限

面临问题

你需要修改一个文件的权限。

解决方案

利用 chmod 命令修改文件的权限。

进一步探讨

需要修改文件权限的常见情况包括：待编辑的文件被设置为只读文件，或者需要赋予某文件以执行权限以便使其可以作为程序或脚本来运行。

命令 chmod 可以用来为文件添加或删除权限。为此，可以使用两种语法：一种使用 8 进制（以 8 为基数），另一种使用文本。我们下面使用的是文本方式，这种方式更易于理解。

命令 chmod 的第一个参数指定需要做哪些修改，第二个参数指定要对哪些文件或文件夹执行这些操作。第一个参数由权限范围（+、-、= 分别用于添加、删除和设置操作）

和权限类型两部分组成。

例如，下面的代码将会为文件 file2.txt 添加执行（x）权限。

```
$ chmod u+x file2.txt
```

如果我们现在使用 ls 命令展示目录内容，就会发现已经添加了 x 权限。

```
$ ls -l  
total 16  
-rw-r--r-- 1 pi pi 5 Apr 23 15:23 file1.txt  
-rwxr--r-- 1 pi pi 5 Apr 24 08:08 file2.txt  
-rw-r--r-- 1 pi pi 5 Apr 23 15:23 file3.txt  
drwxr-xr-x 2 pi pi 4096 Apr 23 15:23 mydir
```

如果我们要给用户组或其他用户添加执行权限的话，则第一个参数分别使用 g 和 o。如果使用字母 a 的话，则会给所有人添加相应的权限。

参考资料

有关文件权限方面的背景知识，请参考 3.12 节。

修改文件属主方面的知识，请参考 3.14 节。

3.14 修改文件的属主

面临问题

你需要修改文件的属主。

解决方案

命令 chown（change owner）可以用来修改文件或目录的属主。

进一步探讨

就像我们在 3.12 节所看到的那样，任何文件或目录都具有属主及相应的用户组。对于树莓派来说，大多数情况下其用户都是单用户，所以实际上我们无需关心用户组。

偶然的情况下，你会发现系统上安装的某些文件的属主并非用户 pi。如果遇到这种情况，你可以利用 chown 命令来修改文件的属主。

为了修改文件的属主，可以使用命令 chown，后面是新的属主和用户组，两者之间用逗号分隔，最后是文件名。

你可能会发现，修改文件的属主是需要超级用户权限的，这时可以在 chown 命令前面加上 sudo（见 3.11 节）。

```
$ sudo chown root:root file2.txt
$ ls -l
total 16
-rw-r--r-- 1 pi    pi      5 Apr 23 15:23 file1.txt
-rwxr--r-- 1 root  root    5 Apr 24 08:08 file2.txt
-rw-r--r-- 1 pi    pi      5 Apr 23 15:23 file3.txt
drwxr-xr-x 2 pi    pi   4096 Apr 23 15:23 mydir
```

参考资料

有关文件权限方面的背景知识，请参考 3.12 节。

此外，还可以参考 3.13 节来了解修改文件权限的方法。

3.15 屏幕截图

面临问题

你希望截取树莓派屏幕的图像，并将其保存到一个文件中。

解决方案

安装使用讨人喜欢的屏幕截图软件，它的名字是 scrot。

进一步探讨

若要安装 scrot，请在终端下面运行如下所示的命令。

```
$ sudo apt-get install scrot
```

在进行截屏时，最简单的方法就是直接输入命令 scrot。这样就能立即截取主屏上面的图像，并以类似 2013-04-25-080116_1024x768_scrot.png 这样的文件名，将图像保存到当前目录中。

有时候，你需要截取一类特殊对象的快照，例如当前打开的菜单等，但是这些对象所在的窗口一旦失去焦点，这些对象就会随之消失。遇到这种情况的时候，你可以使用-d 选项来规定一个延迟，让它在指定的延迟时间之后进行截图。

```
$ scrot -d 5
```

这个延迟时间是以秒为单位的。

如果截取的是整个屏幕，可以利用像 Gimp（见 4.11 节）这样的图像编辑软件来剪裁它。不过，更简便的做法是截屏时直接利用鼠标选取所需部分，为此可以使用-s 选项。

使用这个选项时，先输入这个命令，然后用鼠标框选所需的屏幕部分即可。

```
$ scrot -s
```

保存截图的文件名将包含该图像的像素大小。

参考资料

命令 scrot 还有许多其他选项，可用于控制其他事项，例如使用多屏，以及改变保存截图的文件的格式等。如果你想了解 scrot 命令的更多选项，可以利用下列命令来查看其手册。

```
$ man scrot
```

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

3.16 利用 apt-get 安装软件

面临问题

你想利用命令行方式来安装软件。

解决方案

从终端会话安装软件的时候，最常用的工具就是 apt-get。

这个命令必须以超级用户身份来运行，其基本格式如下所示。

```
$ sudo apt-get install <name of software>
```

例如，要想安装字处理软件 AbiWord，你可以使用下列命令。

```
$ sudo apt-get install abiword
```

进一步探讨

apt-get 软件包管理器会用到一个可用软件的列表。在你所用的树莓派操作系统发行包中就包含了这个列表，不过这里边的这个很可能已经过时了。所以，当你安装某软件而 apt-get 却告诉你 notfound 时，就需要利用下列命令来更新这个列表了。

```
$ sudo apt-get update
```

因为这个列表以及安装软件包都位于互联网上，所以，只有树莓派连接互联网之后，才能正常进行安装软件。



当你进行更新时，如果遇到类似 E: ProblemWith MergeList /var/lib/dpkg/status 这样的错误信息，则可以尝试如下所示的命令。

```
sudo rm /var/lib/dpkg/status  
sudo touch /var/lib/dpkg/status
```

安装过程经常需要一些时间，因为各种文件需要先下载下来，然后才能进行安装。某些安装软件还会在你的桌面上添加快捷方式，或者在 Start 菜单中添加程序组。

你可以在 apt-get search 命令后面加上搜索字符串，如 abiword，来搜索需要安装的软件。运行该命令后，将会看到一个包含有与你想要安装的软件相匹配的软件的列表。

参考资料

当你不再使用某些程序的时候，可以将其删除以释放空间，具体删除方法请参考 3.17 节。

关于从 GitHub 下载源代码的方法，请参考 3.20 节。

3.17 删除利用 apt-get 安装的软件

面临问题

当你利用 apt-get 安装了大量程序之后，现在想要将其中某些删掉。

解决方案

工具软件 apt-get 有一个删除软件的选项 (remove)，只不过它只能清除利用 apt-get install 命令安装的那些软件包。

例如，如果你想删除 AbiWord 的话，可以使用如下所示的命令。

```
$ sudo apt-get remove abiword
```

进一步探讨

利用上面的方法来删除软件包的时候，并不能清除所有内容，因为这些软件包在安装过程中，同时还会安装其自身所依赖的各种软件包。为了删除这些内容，必须使用 autoremove 选项，具体用法如下所示。

```
$ sudo apt-get autoremove abiword  
$ sudo apt-get clean
```

apt-get 的 clean 选项可以进一步清理无用的软件包安装文件。

参考资料

利用 apt-get 安装软件包的内容，请参考 3.16 节。

3.18 利用 Pip 安装 Python 软件包

面临问题

你希望通过软件包管理器 pip 来安装 Python 库。

解决方案

如果你的 Raspbian 是最新版本的话，那么 pip 已经安装在系统上了，你可以从命令行运行下面的示例命令，该例子取自 8.1 节，其作用是安装 Python 库 svgwrite。

```
$ sudo pip install svgwrite
```

如果你的系统还没有安装 pip 的话，可以通过下面所示的命令进行安装。

```
$ sudo apt-get install python-pip
```

进一步探讨

虽然许多 Python 库可以通过 apt-get（参见 3.16 节）进行安装，但是某些却不行，这时你就得使用 pip 来安装了。

参考资料

若要使用 apt-get 命令安装软件，请参考 3.16 节。

3.19 通过命令行获取文件

面临问题

你希望在不借助浏览器的情况下，从互联网上面下载一个文件。

解决方案

利用命令 wget 从互联网下载文件。

举例来说：

```
$ wget http://www.icrobotics.co.uk/wiki/images/c/c3/Pifm.tar.gz
--2013-06-07 07:35:01--  http://www.icrobotics.co.uk/wiki/images/c/c3/Pifm.tar.gz
Resolving www.icrobotics.co.uk (www.icrobotics.co.uk)... 155.198.3.147
Connecting to www.icrobotics.co.uk (www.icrobotics.co.uk)|155.198.3.147|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5521400 (5.3M) [application/x-gzip]
Saving to: 'Pifm.tar.gz'

100%[=====] 5,521,400   601K/s

2013-06-07 07:35:11 (601 KB/s) - `Pifm.tar.gz' saved [5521400/5521400]
```

如果你的 URL 中含有任何特殊字符，则最好用双引号将其括住。本例中的 URL 取自 4.10 节。

进一步探讨

你可以寻找一些通过 wget 获取文件的软件安装方法的说明，你就会发现通常情况下，

利用命令行下载文件，要比通过浏览器来寻找、下载并将文件复制到所需位置方便得多。

命令 wget 以用于下载之用的 URL 作为其参数，并将文件下载到当前目录下面。它常用来下载某种类型的压缩文件，不过也能用于下载任何网页。

参考资料

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

关于浏览器的选择和使用方面的内容，请参考 4.3 节。

3.20 利用 Git 获取源代码

面临问题

有时候，Python 库及其他软件都是以 Git 仓库的 URL 的形式来提供的。

你需要将它们下载到树莓派上面。

解决方案

为了使用 Git 仓库中的代码，你需要使用 Git clone 命令来获取相应的文件。

进一步探讨

例如，下面的命令将会下载本书的所有示例源码。

```
$ git clone https://github.com/simonmonk/raspberrypi_cookbook_ed2.git
```

参考资料

若要进一步了解 Git 和 Git 托管服务，请分别访问 <http://www.git-scm.com> 和 <http://www.github.com>。

此外，请同时参考 3.16 节。

3.21 在系统启动时自动运行程序或脚本

面临问题

你希望某些程序或脚本，在树莓派开机时自动运行。

解决方案

修改 rc.local 文件，来自动运行你感兴趣的程序。

你可以通过下列命令来编辑/etc/rc.local 文件。

```
$ sudo nano /etc/rc.local
```

在第一段注释代码（以#开头的各行）后面添加如下所示的一行内容。

```
$ /usr/bin/python /home/pi/my_program.py &
```

上面命令行末尾务必加上&，这一点非常重要，其作用是指示在后台运行；否则的话，树莓派就无法引导。

进一步探讨

这种自动运行程序的方法需要小心翼翼地编辑 rc.local，否则可能导致树莓派无法自举。

参考资料

还有一种更加安全的方法可以自动运行程序，我们将在 3.22 节进行介绍。

3.22 让程序或脚本作为服务自动运行

面临问题

你希望让一个脚本或程序在树莓派每次重启时都自动运行。

解决方案

Debian Linux（大部分树莓派的发行包都是从此系统演变而来的）为了在系统启动时自动运行某些命令而采取了一种基于依赖包的机制。

这种情况下，对于在 init.d 文件夹中运行的脚本或程序来说，它们的配置文件的使用和创建就要求颇具技巧了。

进一步探讨

下面通过一个例子来说明如何运行 home 目录下的 Python 脚本。

虽然这个脚本可以做任何事情，但就本例而言，它只是运行了一个简单的 Python 网络服务器，具体情况请参考 7.17 节。

下面是这个过程中所涉及到的具体步骤。

1. 创建一个 init 脚本。
2. 赋予这个 init 脚本执行权限。
3. 通知系统新建了一个 init 脚本。首先，我们要创建这个 init 脚本。这需要在/etc/init.d/

文件夹中完成这项工作。这个脚本可以随意命名，不过就本例来说，我们给它取名为 my_server。

可以利用 nano 来创建这个新文件，具体命令如下所示。

```
$ sudo nano /etc/init.d/my_server
```

将下面的代码粘贴到编辑器窗口中，并保存文件。

```
### BEGIN INIT INFO
# Provides: my_server
# Required-Start: $remote_fs $syslog $network
# Required-Stop: $remote_fs $syslog $network
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Simple Web Server
# Description: Simple Web Server
### END INIT INFO

#!/bin/sh
# /etc/init.d/my_server

export HOME
case "$1" in
    start)
        echo "Starting My Server"
        sudo /usr/bin/python /home/pi/myserver.py 2>&1 &
        ;;
    stop)
        echo "Stopping My Server"
        PID=`ps auxwww | grep myserver.py | head -1 | awk '{print $2}'`
        kill -9 $PID
        ;;
    *)
        echo "Usage: /etc/init.d/my_server {start|stop}"
        exit 1
        ;;
    esac
exit 0
```

对于自动运行脚本而言，这些代码看起来有些长，不过大部分都是些样板式代码。若要运行一个不同的脚本，你需要通读上述代码，修改一下待运行的 Python 文件的名称和描述即可。下一步是赋予文件属主执行权限，具体命令如下所示。

```
$ sudo chmod +x /etc/init.d/my_server
```

现在，该程序已经被设置为一个服务，如果各项测试一切正常的话，便可以将其放入到 autostart 中，使其作为引导序列的一部分，具体命令如下所示。

```
$ /etc/init.d/my_server start  
Starting My Server  
Bottle v0.11.4 server starting up (using WSGIRefServer())...  
Listening on http://192.168.1.16:80/  
Hit Ctrl-C to quit.
```

最后，如果运行正常的话，可以通过下面的命令让系统了解你定义的新服务。

```
$ sudo update-rc.d my_server defaults
```

参考资料

还有一种更加简单的方法可以让程序自动运行，具体请参考 3.21 节。

至于修改文件和文件夹权限的具体介绍，请参考 3.12 节。

3.23 定期自动运行程序或脚本

面临问题

你想要每天运行一次脚本，或定期运行脚本。

解决方案

使用 Linux 的 crontab 命令。

为此，树莓派需要具体的时间与时期，也就是说需要具有网络连接或一个实时时钟，具体请参考 12.13 节。

进一步探讨

命令 crontab 可以让某些事件定期发生。这里的时间周期可以是天或者小时，同时，你还可以定义更加复杂的模式，以便在每周不同的日子里触发不同的事件。这对于需要在半夜进行备份的任务来说，是非常有用的。

你可以通过下列命令来编辑需要调度的事件。

```
$ crontab -e
```

如果你希望运行的程序或脚本需要超级用户权限的话，可以在所有的 crontab 命令前面加上前缀 sudo（见 3.11 节）。

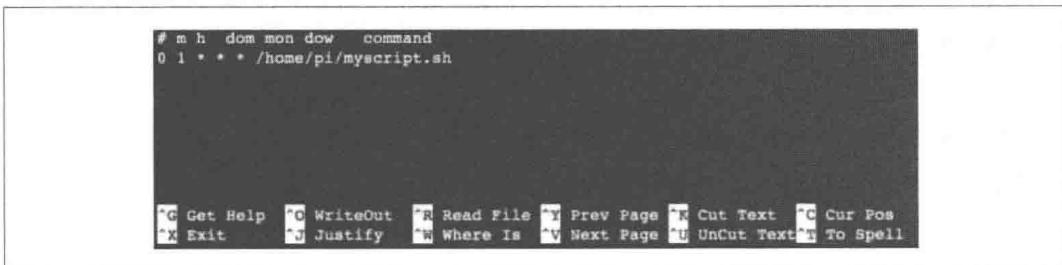
注释行指出了 crontab 行的格式。这些数字依次表示分钟、小时、每月中的几号、月、星期几，然后是你想要运行的命令。

第一行（以#开头）就是注释行，它能够提示 crontab 行的具体格式。

如果在数字位置上出现了*，它表示“每”；如果该位置出现的是数字的话，该脚本就

只有在每月的分/时/天/才会被执行。

所以，如果需要在每天凌晨 1 点运行脚本的话，需要添加图 3-7 中所示的一行内容。



```
# m h dom mon dow    command
0 1 * * * /home/pi/myscript.sh
```

图 3-7 编辑 crontab

只用一个数字的话，你可以利用从星期几到星期几的方式指定运行的时间范围，例如：

```
0 1 * * 1-5 /home/pi/myscript.sh
```

如果你的脚本需要从特定的目录运行的话，可以使用分号（;）来间隔多个命令，具体如下所示。

```
0 1 * * * cd /home/pi; python mypythoncode.py
```

参考资料

如果你想查看 crontab 的详细文档，可以使用如下所示的命令。

```
$ man crontab
```

3.24 搜索功能

面临问题

你知道一个文件就在系统的某个地方，具体不太清楚，所以你想设法找到它。

解决方案

利用 Linux 的 find 命令。

进一步探讨

find 命令会从指定的目录开始搜索，如果找到了相应的文件，就会显示其位置。例如：

```
$ find /home/pi -name gemgem.py
/home/pi/python_games/gemgem.py
```

你可以从目录树中更高的目录开始搜索，甚至可以从整个文件系统的根目录（/）开始。不过这样做的话，会耗费更多的时间，同时还会收到出错信息。

你可以将这些出错信息重定向到其他地方，方法是在命令行后面添加 2>/dev/null。

如果要在整个文件系统中搜索文件的话，可以使用如下所示的命令。

```
$ find / -name gemgem.py 2>/dev/null  
/home/pi/python_games/gemgem.py
```

在 find 命令中，你同样可以使用通配符，具体如下所示。

```
$ find /home/pi -name match*  
/home/pi/python_games/match4.wav  
/home/pi/python_games/match2.wav  
/home/pi/python_games/match1.wav  
/home/pi/python_games/match3.wav  
/home/pi/python_games/match0.wav  
/home/pi/python_games/match5.wav
```

参考资料

实际上，find 命令还有许多其他高级搜索功能。如果你想查阅完整的 find 手册文档的话，可以使用下面所示的命令。

```
$ man find
```

3.25 使用命令行历史记录功能

面临问题

你希望能够在命令行中重复执行某些命令，而无需再次输入它们。

解决方案

使用向上和向下箭头键从命令历史记录中选择之前用过的命令，或者使用 history 命令与 grep 命令来查找之前用过的命令。

进一步探讨

你可以通过点按上箭头键来访问上一条运行过的命令。再次点按此键的话，就会访问上一条命令之前的那条命令等。如果你越过了想要使用的命令，可以使用下箭头键往回退。

如果你想取消对选中命令的运行操作的话，可以使用组合键 Ctrl-C。

随着时间的推移，命令历史记录会变得很大，以至于找不到你以前使用过的命令。要

想找到很久以前用过的命令，你可以求助于命令 history。

```
$ history
1 sudo nano /etc/init.d/my_server
2 sudo chmod +x /etc/init.d/my_server
3 /etc/init.d/my_server start
4 cp /media/4954-5EF7/sales_log/server.py myserver.py
5 /etc/init.d/my_server start
6 sudo apt-get update
7 sudo apt-get install bottle
8 sudo apt-get install python-bottle
```

上面列出了所有的命令历史记录，由于这些内容太多，可能很难找到你想要的命令。为了弥补这一点，你可以使用管道方式将 history 命令的结果输出给 grep 命令，使其只显示与搜索字符串相匹配的命令。所以，如果要查找所有执行过的 apt-get 命令（见 3.16 节），可以使用如下所示的命令。

```
$ history | grep apt-get
6 sudo apt-get update
7 sudo apt-get install bottle
8 sudo apt-get install python-bottle
55 history | grep apt-get
```

这里每条历史命令前面都有一个编号，所以，找到所需的命令后，可以通过在！后面加上其历史编号的方法来运行它，具体如下所示。

```
$ !6
sudo apt-get update
Hit http://mirrordirector.raspbian.org wheezy InRelease
Hit http://mirrordirector.raspbian.org wheezy/main armhf Packages
Hit http://mirrordirector.raspbian.org wheezy/contrib armhf Packages
....
```

参考资料

如果你想查找的是文件，而不是命令，请参见 3.24 节。

3.26 监视处理器活动

面临问题

树莓派有时可能运行有点慢，所以你想看看是哪些进程霸占了处理器。

解决方案

使用任务管理器（Task Manager）实用程序，你可以在系统工具程序组的开始菜单中找

到它（见图 3-8）。

通过任务管理器，我们可以查看 CPU 和内存的使用情况。你还可以右键单击某个进程，然后从弹出菜单中选择相应选项以将其停止。

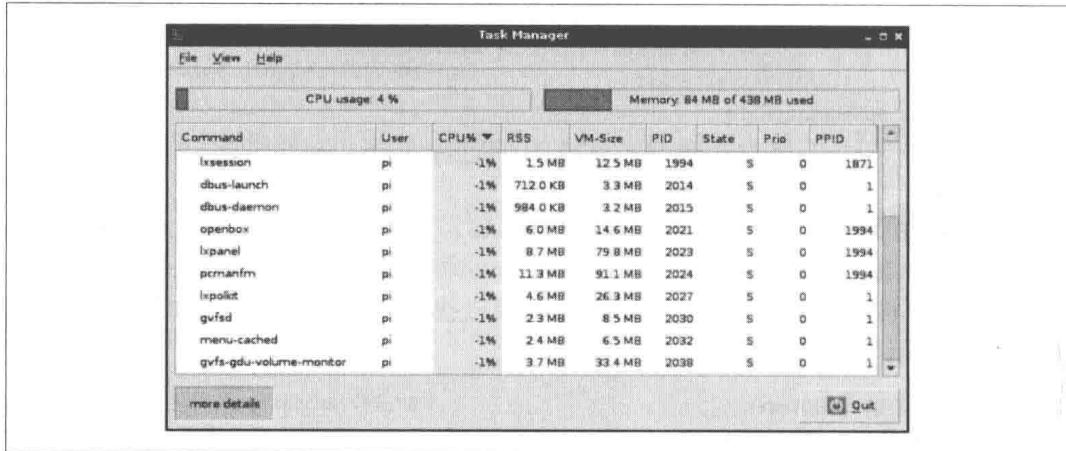


图 3-8 任务管理器

窗口顶部的图形显示了 CPU 和内存的总体使用率。进程列表位于下方，你可以在此了解每个进程占用 CPU 的情况。

进一步探讨

如果你喜欢从命令行完成这类事情，则可以使用 Linux 系统的 top 命令来显示处理器和内存的使用情况，以及哪些进程占用了大量的资源（见图 3-9）。然后可以使用 kill 命令杀死进程。不过，这需要具备超级用户权限。

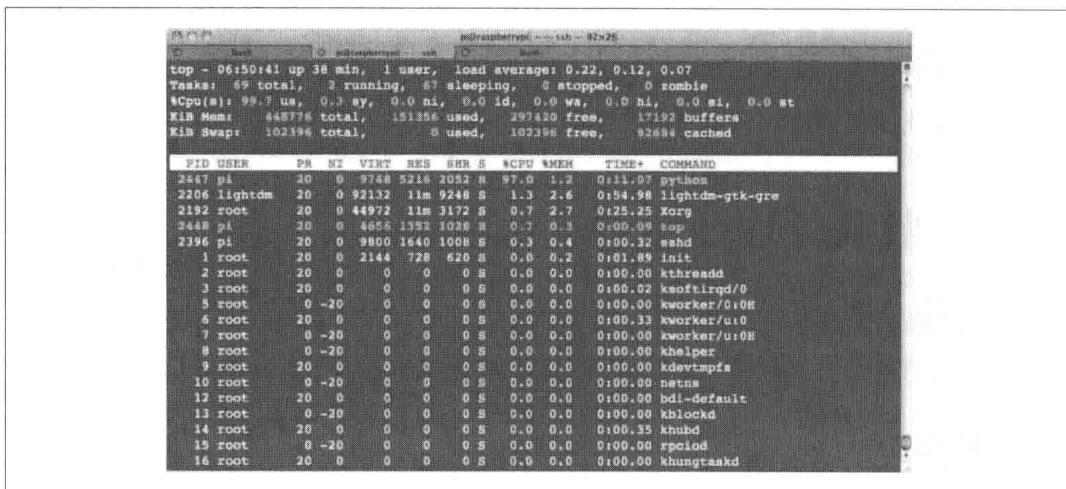


图 3-9 使用 top 命令查看资源使用情况

在本例中，你可以看到，位于上面的那个进程是一个占用了 97%CPU 资源的 Python 程序。第一列显示其进程 ID (2447)。如果要终止这个进程的话，可以输入以下命令：

```
$ kill 2447
```

但这样做的时候，很可能会杀死一些重要的操作系统进程，遇到这种情况，可以关闭树莓派然后重启，一切就会恢复正常了。

有时，有些正在运行的进程在使用 top 后无法立即看到。如果遇到这种情况，可以使用 ps 命令并将结果通过管道（见 3.31 节）传递给 grep 命令来查找所有正在运行的进程，这样不仅可以搜索结果，而且还能将找到的内容高亮显示。

例如，为了找出霸占 CPU 的 Python 进程的进程 ID，我们可以运行以下命令：

```
$ ps -ef | grep "python"  
pi      2447 2397 99 07:01 pts/0    00:00:02 python speed.py  
pi      2456 2397  0 07:01 pts/0    00:00:00 grep --color=auto python
```

就本例而言，Python 程序 speed.py 的进程 ID 为 2447。列表中的第二个条目是 ps 命令本身的进程。

kill 命令的变种是 killall 命令。这个命令要谨慎使用，因为它会终止所有匹配其参数的进程。下面的命令将终止树莓派上运行的所有 Python 程序：

```
$ sudo killall python
```

参考资料

另请参见 top、ps、grep、kill 和 killall 的命令手册。你可以先输入 man，后跟相应的命令名来查看其帮助信息，具体如下所示：

```
$ man top
```

3.27 文件压缩

面临问题

你下载了压缩文件并想将其解压缩。

解决方案

你可以使用 tar 或 gunzip 命令，具体视文件类型而定。

进一步探讨

如果要解压缩的文件仅具有.gz 扩展名，则可以通过下列命令来解压缩：

```
$ gunzip myfile.gz
```

你还经常会遇到一些称为 tarball 的文件，其中包含的目录已经被 Linux 的 tar 命令压缩过，然后又用 gzip 压缩到名称类似 myfile.tar.gz 的文件中。

你可以使用 tar 命令从 tarball 中提取原始文件和文件夹：

```
$ tar -xzf myfile.tar.gz
```

参考资料

用户手册中提供了 tar 的详细信息，你可以通过命令 man tar 来访问这些信息。

3.28 列出已连接的 USB 设备

面临问题

你已插入 USB 设备，并希望弄清楚 Linux 是否能够识别它。

解决方案

使用 lsusb 命令。该命令能够列出已经连接到树莓派 USB 端口的所有设备：

```
$ lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 15d9:0a41 Trust International B.V. MI-2540D [Optical mouse]
```

进一步探讨

这个命令可以告诉你设备是否已连接，但不能保证设备可以正常工作。这可能需要为硬件安装驱动程序或做进一步配置。

参考资料

有关附加外部网络摄像头时使用 lsusb 的示例，请参阅 4.5 节。

3.29 将输出从命令行重定向到文件

面临问题

你希望快速创建一个包含某些文本的文件，或将目录列表记录到文件中。

解决方案

使用>命令将本应该显示在命令行中的输出重定向。

例如，要将目录列表复制到名为 myfiles.txt 的文件中，请执行以下操作：

```
$ ls > myfiles.txt  
$ more myfiles.txt  
Desktop  
indiecity  
master.zip  
mcpi
```

进一步探讨

可以在任何产生输出的 Linux 命令上使用>命令，甚至包括 Python 程序。

你还可以使用与之相反的<命令来重定向用户输入，尽管它不像>那么有用。

参考资料

若要使用 cat 将多个文件连接在一起的话，请参考 3.30 节。

3.30 连接文件

面临问题

你有多个文本文件，并且要将它们合并成一个大文件。

解决方案

使用 cat 命令将多个文件连接到一个输出文件中。

例如：

```
$ cat file1.txt file2.txt file3.txt > full_file.txt
```

进一步探讨

连接文件是 cat 命令的真正目的。你可以提供尽可能多的文件名，它们都将写入所指向的文件中。如果不重定向输出的话，那么输出仅显示在“终端”窗口中。如果它们是些大型文件的话，则可能需要耗费一些时间！

参考资料

另请参考 3.7 节，在那里 cat 用于显示文件的内容。

3.31 使用管道

面临问题

你想使用一个 Linux 命令的输出作为另一个命令的输入。

解决方案

使用 pipe 命令（即键盘上竖线（|））将一个命令的输出传递给另一个命令。

例如：

```
$ ls -l *.py | grep Jun  
-rw-r--r-- 1 pi pi 226 Jun 7 06:49 speed.py
```

本例将找到所有扩展名为 py 并且在目录列表中含有 Jun 的文件，表明它们在 6 月最后一次修改。

进一步探讨

乍一看，它与重定向命令>（见 3.29 节）非常类似。两者的区别是，+>_命令不能重定向到另一个程序，而是只能重定向到文件。

只要愿意，我们可以将尽可能多的程序连接起来，如下所示，但是这种方式并不常见。

```
$ command1 | command2 | command3
```

参考资料

有关使用 grep 查找进程的示例，请参阅 3.26 节。关于通过管道和 grep 搜索命令历史记录的示例，请参考 3.25 节。

3.32 将输出隐藏到终端

面临问题

你想运行一个命令，但是又不想让输出信息填满屏幕。

解决方案

使用>将输出重定向到/dev/null。

例如：

```
$ ls > /dev/null
```

进一步探讨

这个例子只是用来展示语法，别无他用。一个更常见的使用场景是，所运行的程序代码中，开发人员留下了很多跟踪消息，而你又不想看到这些消息。下面的示例隐藏了 find 命令的多余输出（参见 3.34 节）。

```
$ find / -name gemgem.py 2>/dev/null  
/home/pi/python_games/gemgem.py
```

参考资料

有关重定向至标准输出的详细信息，请参阅 3.29 节。

3.33 在后台运行程序

面临问题

你想在运行一个程序的同时还进行一些其他任务。

解决方案

使用&命令在后台运行程序或命令。

例如：

```
$ python speed.py &  
[1] 2528  
$ ls
```

这样的话，就不用等待程序运行完成了，命令行仅显示进程 ID（第二个数字），并立即允许继续运行其他命令。然后，您可以使用此进程 ID 来终止后台进程（详见 3.26 节）。

要使后台进程回到前台，请使用 fg 命令：

```
$ fg  
python speed.py
```

该命令将报告正在运行的命令或程序，然后等待它完成。

进一步探讨

后台进程的输出仍将显示在终端中。

在后台运行进程的替代方法是打开多个终端窗口。

参考资料

有关进程管理的详细介绍，请参阅 3.26 节。

3.34 创建命令别名

面临问题

你想为经常使用的命令创建别名。

解决方案

使用 nano (详见 3.6 节) 编辑文件 ~/.bashrc，在文件的末尾添加所需内容，具体如下所示：

```
alias l='ls -a'
```

上述命令将创建一个名为 l 的别名，当输入该别名时，将被视为输入了命令 ls-a。

使用 Ctrl-X 和 Ctrl-Y 保存并退出文件，然后使用新的别名更新终端，键入以下命令：

```
$ source .bashrc
```

进一步探讨

许多 Linux 用户喜欢通过下列方式为 rm 设置别名，以便它在得到确认后才进行删除操作。

```
$ alias rm='rm -i'
```

这不是一个坏方法，但如果你使用别人的系统，请别忘了人家可能没有设置这个别名！

参考资料

有关 rm 的详细信息，请参阅 3.10 节。

3.35 设置日期和时间

面临问题

你希望在 Raspberry Pi 上手动设置日期和时间，因为它没有 Internet 连接。

解决方案

可以使用 Linux 系统的 date 命令。

日期和时间的格式为 MMDDHHMMYYYY，其中 MM 是月份，DD 是月中的日，HH 和 MM 分别是小时和分钟，YYYY 是年份。

例如：

```
$ sudo date 010203042013  
Wed Jan 2 03:04:00 UTC 2013
```

进一步探讨

如果树莓派已经连入 Internet，那么当它启动时，将使用 Internet 时间服务器自动设置自己的时间。

在使用 date 命令的时候，还可以通过仅仅输入 date 本身来显示 UTC 时间：

```
$ date  
Wed Jan 2 03:08:14 UTC 2013
```

参考资料

如果你希望树莓派即便在没有网络的情况下也能保持正确的时间的话，则可以使用实时时钟（RTC）模块（具体参考 12.13 节）。

3.36 查看 SD 卡剩余存储空间

面临问题

你想知道 SD 卡上有多少可用空间。

解决方案

可以使用 Linux 系统的 df 命令：

```
$ df -h  
Filesystem      Size  Used Avail Use% Mounted on  
rootfs          3.6G  1.7G  1.9G  48% /  
/dev/root        3.6G  1.7G  1.9G  48% /  
devtmpfs         180M    0  180M   0% /dev  
tmpfs           38M  236K  38M   1% /run  
tmpfs           5.0M    0  5.0M   0% /run/lock  
tmpfs            75M    0   75M   0% /run/shm  
/dev/mmcblk0p1   56M   19M   38M  34% /boot
```

进一步探讨

通过输出结果的第一行可以看出，该存储卡的存储空间为 3.6 GB，其中已经使用了 1.7 GB。

当磁盘空间被耗尽的时候，可能会出现意外的错误行为，例如错误消息指出文件无法写入。

参考资料

你可以通过 man df 命令查看 df 的用户手册。

软件

4.0 引言

本章介绍树莓派上面各种现成软件的使用方法。

本章中的某些内容会把树莓派变成一个单一用途的工具，而其他内容则介绍树莓派上面个别软件的用法。

4.1 搭建媒体中心

面临问题

你想把自己的树莓派打造成一个超级媒体中心。

解决方案

为了将树莓派打造成一个媒体中心，你可能需要拥有树莓派 3 或者 B 型树莓派 2 那样的优异性能，毕竟媒体播放是一个处理器密集型的工作。

在安装 NOOBS 的过程中，你就可以把树莓派配置成为一个媒体中心（见 1.5 节）。除了选择安装 Raspbian 之外，对于树莓派 1 来说，可以安装 OpenELEC_Pi1，对于树莓派 2 来说，可以安装 OpenELEC_Pi2。

OpenELEC 是专门针对树莓派用作媒体中心而优化过的一个操作系统。它提供了 Kodi 媒体中心软件，该软件是基于开源项目 XBMC 的，该项目最初就是用于将 Xbox 游戏控制台转换为媒体中心的。后来，这个项目被移植到多种平台上面，其中就包括树莓派（见图 4-1）。

树莓派可以完美播放全高清视频以及流媒体音乐、MP3 文件和互联网广播。

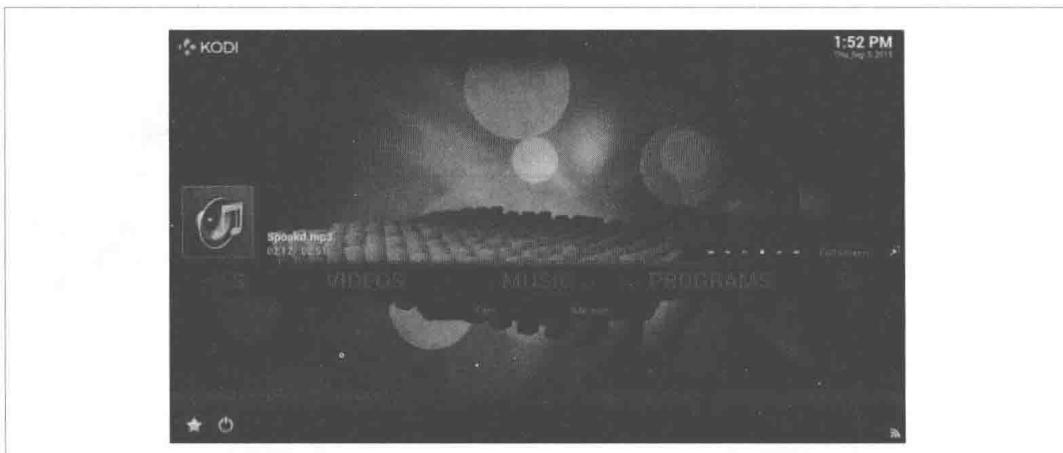


图 4-1 用于媒体中心的树莓派

进一步探讨

Kodi 是一款非常强大的软件，具有丰富的功能。要想检查该软件是否正常工作，最简单的方法恐怕就是将一些视频或音乐文件放到一个 USB 驱动器或 USB 外置硬盘上，并将其连接到树莓派。这样的话，你就能够通过 Kodi 来播放它们了。

树莓派很可能会放置到 TV 旁边，实际上，TV 的 USB 端口完全可以为树莓派提供足够的电流来供其运行。在这种情况下，你就不必使用单独的电源了。

此外，无线键盘和鼠标也是不错的选择，如果你同时购买两者的话，集线器可以共享一个 USB 端口，这样就免得到处都是导线了。这种情况下，使用内置触控板的迷你键盘也是非常有用的。

通常情况下，有线网络的性能要优于 Wi-Fi 连接，但是，如果树莓派不在以太网接口附近的话，用起来就不太方便了。这种情况下，你可以配置 XBMC 实用无线网卡来上网。

Kodi 的配置非常简单，该软件的完整使用指南请参考 <http://kodi.wiki/>。

参考资料

你可以为树莓派添加一个红外线控制器来远程控制 XBMC (<http://bit.ly/17W0Ghl>)。

4.2 安装 Office 软件

面临问题

你需要在树莓派上面打开文字处理软件和电子表格文档。

解决方案

说到底，树莓派就是一台 Linux 计算机，所以有许多办公软件可用，安装好之后就可以处理电子表格和文字处理软件文档了。

树莓派的办公软件可以从网上下载，所以，你需要一个互联网连接。

在安装任何新软件之前，最好打开终端，并输入如下所示的命令。

```
$ sudo apt-get update
```

若要安装 AbiWord 文字处理软件，可以使用如下所示命令。

```
$ sudo apt-get install abiword
```

这时，系统将要求你输入 Y 以确认安装，大概 1 分钟左右，该软件就会安装完毕。如果你查看 Start 菜单，就会发现一个名为 Office 的新组，其中就有 AbiWord（见图 4-2）。

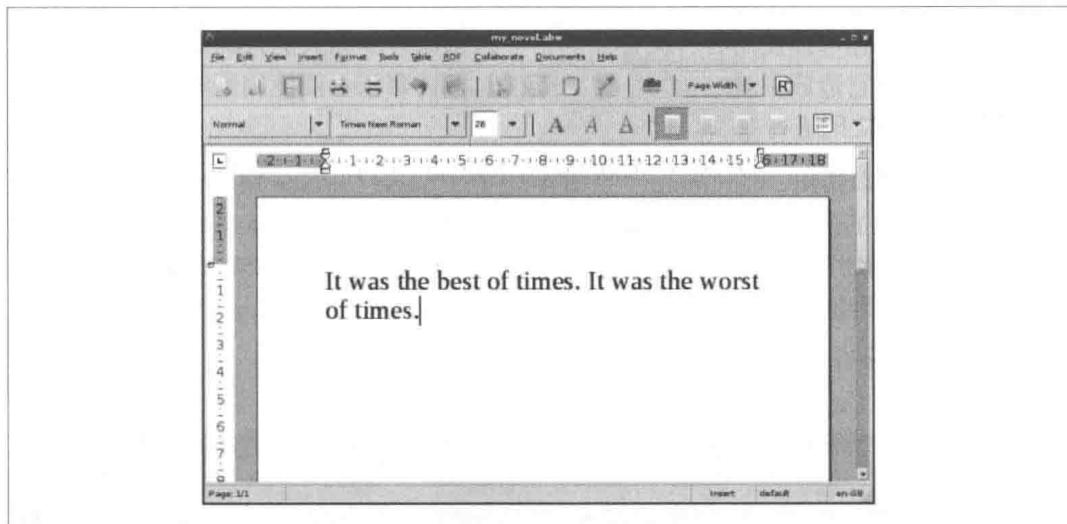


图 4-2 树莓派上的 AbiWord

AbiWord 能够打开.doc、.docx 以及其他常用的字处理文档格式。

如果你需要使用电子表格的话，Gnumeric 软件会是个不错的选择。你可以通过下列命令来安装该软件。

```
$ sudo apt-get install gnumeric
```

进一步探讨

如果你发现这些办公应用运行速度太慢的话，可以尝试对树莓派进行超频来提高速度（参考 1.10 节）。树莓派 3 或 2 在运行这些办公软件的时候，通常要比之前的树莓派快得多。

参考资料

人们正在努力将其他办公软件移植到树莓派上面，比如 Libre Office (Open Office 软件的一个分支)。所以，你可以经常上网关注办公软件移植方面的新闻。

关于 apt-get 的使用方法，可以参考 3.16 节。

4.3 安装其他浏览器

面临问题

你想使用 Midori 之外的浏览器。

解决方案

在树莓派上面，你可以使用各种浏览器。由于树莓派并非强大的计算机，所以现代浏览器和网页会对它造成很大的负担。也就是说，在树莓派上面使用浏览器的时候，需要在功能和性能之间进行权衡。

从名字上面就可以猜出，Chromium (见图 4-3) 是 Google Chrome 用户非常熟悉的一款浏览器。它提供了全部的功能，但是当你上下滚动一个处于忙碌状态的页面时，会明显变慢。你可以使用下列命令来安装 Chromium，安装之后将会在 Start 菜单的 Internet 部分添加一个新链接。

```
$ sudo apt-get install chromium-browser
```

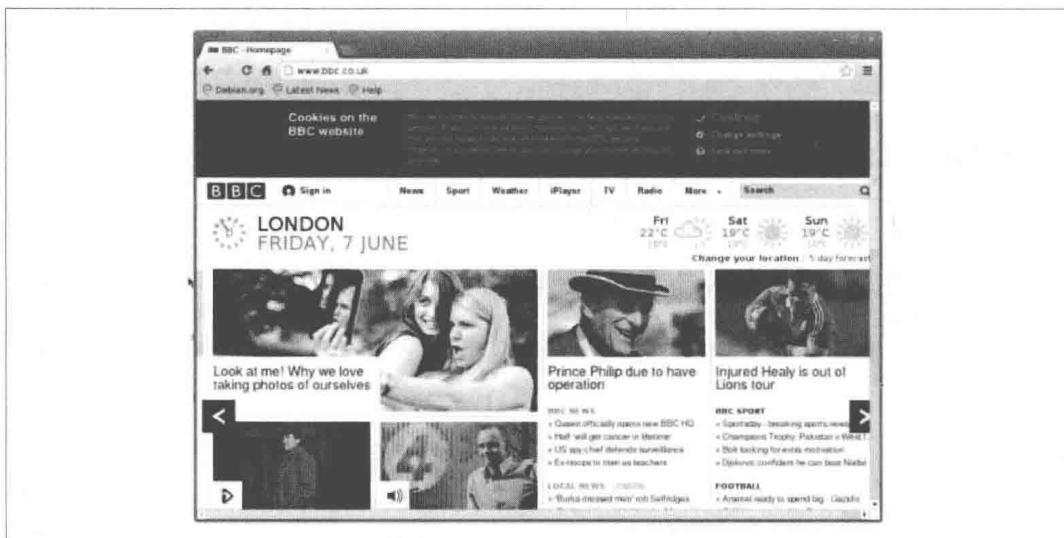


图 4-3 Chromium 浏览器

Midori 浏览器的另一个常见的替代品是 Iceweasel (见图 4-4)。这个基于 Firefox 的浏览器要比 Chromium 快得多，因为它会使用网站的移动版本（如果有的话），而移动版本的页面通常是更加简洁的 HTML。你可以通过下列命令来下载安装 Iceweasel 浏览器。

```
$ sudo apt-get install iceweasel
```

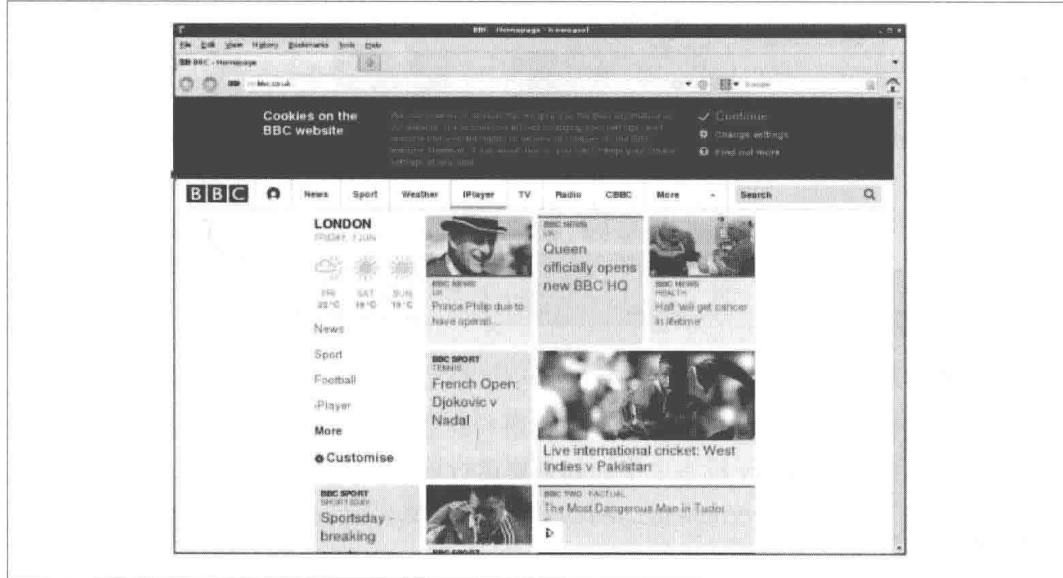


图 4-4 Iceweasel 浏览器

进一步探讨

对于大部分网站来说，浏览器需要具备足够的计算能力才能顺畅浏览，所以，为了避免难以忍受的龟速感，你需要一个树莓派 3 或树莓派 2。

参考资料

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

4.4 使用树莓派商店

面临问题

你想要使用树莓派商店来安装软件和游戏。

解决方案

树莓派的树莓派商店（见图 4-5），犹如苹果的 App 商店或者谷歌的 Play 商店，你可以

从中下载、安装和运行各种免费和收费的应用程序。

你可以在 Start 菜单的 Internet 部分找到树莓派商店应用的快捷方式。

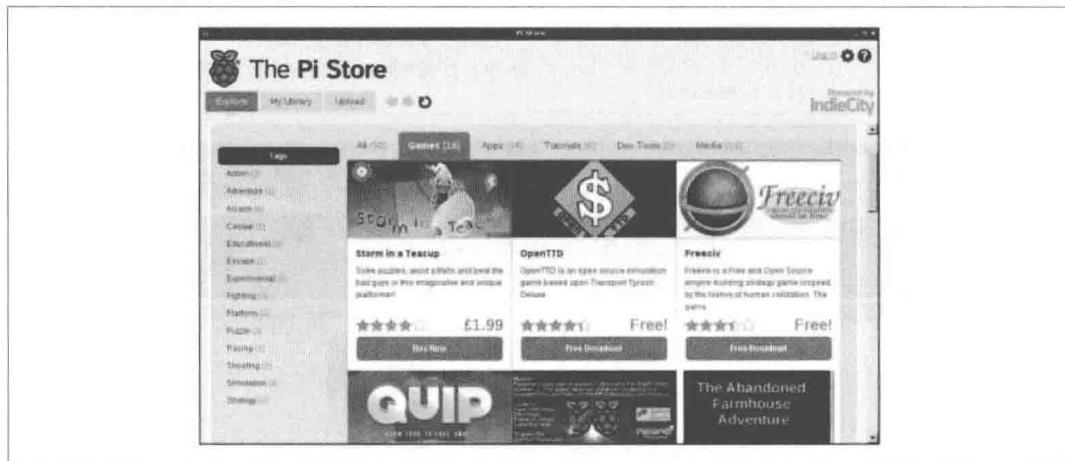


图 4-5 树莓派商店

当你第一次尝试下载应用程序时，会要求你进行注册。注册之后，应用程序就会下载下来，并出现在 My Library 标签之中。如果要运行该程序的话，只需要双击即可。

进一步探讨

利用树莓派商店，你可以方便地浏览树莓派的各种有趣应用。随着时间的推移，将会有越来越多的应用入驻该商店。

参考资料

你可以浏览树莓派商店的官方网站 (<http://Store.raspberrypi.com/>)。

4.5 打造网络摄像头服务器

面临问题

你想把树莓派打造成一台网络摄像头服务器。

解决方案

下载 motion 软件。利用它，你可以对带有 USB 网络摄像头的树莓派进行相应的配置，以便可以通过网页查看摄像头。

要想安装该软件，可以在终端中输入下列命令。

```
$ sudo apt-get install motion
```

将 USB 网络摄像头插入树莓派，然后利用 lsusb 命令查看该摄像头的连接情况。

```
$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 3538:0059 Power Quotient International Co., Ltd
Bus 001 Device 006: ID ebla:299f eMPIA Technology, Inc.
```

如果没有发现明显与网络摄像头有关的内容，请断开它，并重新运行该命令，然后观察是否有内容从列表中消失。就本例而言，列表中的最后一项内容就是网络摄像头。

现在，需要修改几处配置。首先，你需要编辑文件 /etc/motion/motion.conf，所需命令如下所示。

```
$ sudo nano /etc/motion/motion.conf
```

这是一个大型的配置文件，在其顶部附近，可以找到“daemon off；”，你需要将其改为“daemon on”。

另一处需要修改的地方位于该文件较为靠下的部分，具体来说，你需要将“webcam_localhost =on”改为“webcam_localhost =off”。

此外，还有一个文件也需要进行修改。为此，需要键入下列命令。

```
$ sudo nano /etc/default/motion
```

将“start_motion_daemon=no”改为“start_motion_daemon=yes”。

为了运行 Web 服务，需要键入下列命令：

```
$ sudo service motion start
```

现在，你可以打开浏览器来观看网络摄像头了。为此，你需要知道树莓派的 IP 地址（见 2.2 节）。

从同一网络的另一台计算机上面打开浏览器，导航至 URL <http://192.168.1.16:8081/>。现实中，你需要把这里的 URL 改为自己树莓派的 IP 地址，不过 URL 尾部的“:8081”端口数不要修改。

如果一切顺利的话，你将会看到类似于图 4-6 所示的内容。



图 4-6 树莓派网络摄像头

进一步探讨

软件 motion 的功能实际上非常强大，它提供了许多其他的设置，可以用来调节网络摄像头的工作情况。

默认时，网络摄像头只能供你的网络内部使用。如果你想将网络摄像头供互联网观看，那么需要对你的家用集线器设置端口转发。这就要求你登录集线器的管理控制台，找到端口转发选项，并在树莓派的 IP 地址的 8081 端口上面启用该功能。

这样一来，你就能够使用 ISP 分配的外部 IP 地址来观看该网络摄像头了。该地址通常可以在管理控制台的第一页中找到。不过需要注意的是除非你从 ISP 那里购买了静态 IP 地址，否则的话，你的家用集线器调制调解器每次重启后，该 IP 地址很可能也会随之改变。

另外，像 No-ip (<http://www.noip.com>) 之类的服务也可以提供静态 DNS，所以，你可以注册一个域名，并且每当你的 IP 地址变化之后，该域名都能自动映射到该地址上面。

参考资料

要想阅读 motion 的完整文档，请访问其网站，地址为 <http://bit.ly/1dyfqqA>。

关于与树莓派兼容的网络摄像头清单，请访问 http://elinux.org/RPi_USB_Webcams。

树莓派还有一个插件式网络摄像头模块（见 1.14 节）。在本书写作期间，该模块还无法兼容 motion，不过当你阅读本书时，也许已经能够保持兼容了。

关于网络摄像头在机器视觉工程中的应用，请参考第 8 章。

关于 apt-get 的使用方法，可以参考 3.16 节。

4.6 运行老式游戏控制台模拟器

面临问题

你想在树莓派上面使用游戏模拟器，把它变成一个老式游戏机。

解决方案

从 20 世纪 80 年代开始，诞生了一批老式游戏机的模拟器。其中最流行的一个模拟器就是 Stella (<http://stella.sourceforge.net/>)，它是用来模拟 Atari 2600（见图 4-7）的。



虽然这些游戏都老掉渣了，已经不值什么了，但是，它们仍然是有版权的。要想在类似 Stella 这些模拟器上面玩这些游戏，需要相应的 ROM 镜像文件，虽然可以轻易从互联网上下载到，但是别忘了，它们并不属于你。所以，请恪守法律。

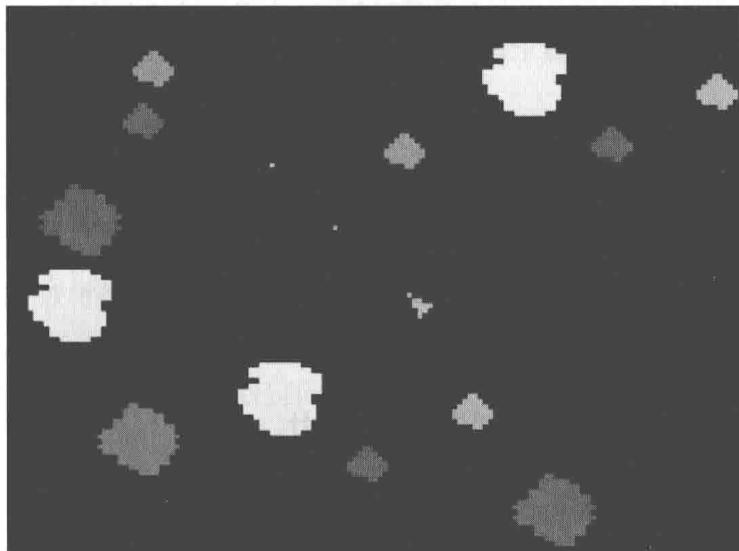


图 4-7 Stella Atari 2600 模拟器中的小行星

要想安装 Stella，只要在终端窗口输入下列命令即可。

```
$ sudo apt-get install stella
```

一旦安装好，你就会在 Start 菜单的 Games 组中看到新安装的程序。但是，现在先不要运行它，因为你还需要获得该游戏的 ROM 镜像。

如果你是在美国以外，并实际拥有这个游戏，那么你有权取得该 ROM 镜像的多个副本，以用于备份，但不是所有国家都允许这样做的。此外，你还可以找到一些已经被放弃版权限制的游戏的 ROM 镜像。

在获得了想要玩的游戏的 ROM 镜像之后，创建一个名为 roms 的文件夹，并将相应的镜像保存到这个文件夹中。然后，启动 Stella。

为了启动游戏，只需单击镜像文件即可。默认情况下，光标键会被映射为游戏摇杆，而空格键则用于开火射击。你可以修改模拟器的许多设置，例如，你必定非常希望修改视频设置，将显示模式改为全屏模式。

此外，你还可以通过 Input Settings，利用 Emul Events 标签将控制器按钮映射到键盘上面。

进一步探讨

树莓派本身的资源并不富裕，但是模拟器会耗费惊人的资源，因此，你会发现你将需要使用树莓派 3 或树莓派 2。

如果你搜索网络的话，就会发现许多人已经做好了这些基本的设置，并添加了复古的 USB 控制器，比如易于获取并且价格低廉的 Nintendo Retrolink USB Super SNES Classic controller，从而把树莓派和显示器打造成了一个街机风格的大型游戏中心。

参考资料

对于树莓派来说，还有许多种可以使用的控制台模拟器，尽管这些模拟器的成熟度和稳定性各有不同。一个值得考虑的模拟器是 Mame (<http://www.mamedev.org/>)，它能够模拟许多不同的游戏平台。

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

4.7 运行树莓派版 Minecraft

面临问题

你想在树莓派上面运行流行游戏 Minecraft。

解决方案

Minecraft 的初创者 Mojang 已经将其移植到了树莓派平台上面，在最新版的 Raspbian 发行包中，已经预装了树莓派版的 Minecraft 游戏（见图 4-8）。



图 4-8 树莓派上面的 Minecraft

进一步探讨

为了让 Minecraft 适应树莓派，开发人员对该游戏的图形代码部分做了部分剪切。这就意味着你只能够直接在连接了键盘、鼠标和显示器的树莓派上面玩游戏。但是，你无

法通过远程的 VNC 连接来玩这款游戏。

树莓派版的 Minecraft 基于该游戏的移动版本，同时精简了某些功能，其中包括著名的 Redstone。

参考资料

要想了解 Minecraft 在树莓派上的移植情况，请参考 <http://pi.minecraft.net/>。

除此之外，你还可以在树莓派上面运行完整版本的 Minecraft 服务器（见 4.8 节），不过，如果在树莓派 3 或树莓派 2 上面运行该游戏的话，效果要更好一些。

树莓派版的 Minecraft 还提供了一个 Python 编程接口，在 7.20 节，你将学习如何使用 SSH 连接向 Python 发送命令来执行自动建造命令。

4.8 运行 Minecraft 服务器

面临问题

你想搭建一个低成本、低能耗的 Minecraft 服务器供自己和朋友们玩游戏。

解决方案

首先，你需要安装 Java 运行时环境，具体命令如下所示。

```
$ sudo apt-get update  
$ sudo apt-get install openjdk-7-jdk
```

然后，创建一个存放服务器代码的目录，并通过下列命令来获取 spigot Minecraft 服务器。

```
$ mkdir mcserver  
$ cd mcserver  
$ wget http://getspigot.org/spigot18/spigot_server.jar
```

启动该服务器的命令如下所示。

```
$ java -Xms512M -Xmx800M -jar spigot_server.jar nogui
```

该服务器很快就会报出下列出错信息。

```
[15:51:26 WARN]: Failed to load eula.txt  
[15:51:26 INFO]: You need to agree to the EULA in order to run  
the server. Go to eula.txt for more info.  
[15:51:26 INFO]: Stopping server
```

这些信息表明你需要接受相关的许可证协议条款，为此，你需要编辑服务器在刚才安装期间为你创建的 eula.txt 文件。这时，可以使用 nano 打开 eula.txt 文件，具体命令如下所示。

```
$ nano eula.txt
```

将文件中的“eula =false”改为“eula =true”，保存文件，然后运行，最后再次启动该服务器。它需要花费几分钟的时间来创建一个世界，并启动该服务器。请不要担心，当你下一次运行它的时候，就会快多了。

最后，该服务器就会运行起来，为了检查它的运行状况，你可以尝试在网络中另一台运行 Minecraft 的计算机上面连接该服务器。

启动 Minecraft，然后依次单击 Multiplayer 和 Direct Connect。输入你的树莓派的 IP 地址（见图 4-9），然后单击 Connect。



图 4-9 连接 Minecra 服务器

一旦连接成功，你跟网络上的其他伙伴就可以加入该服务器开始游戏了（见图 4-10）。



图 4-10 在树莓派 Minecra 服务器上玩游戏

你也许想要将自己在游戏中的身份变成 operator，为此可以使用下列命令。

```
> op YourName
```

这里，YourName 是你在游戏中的昵称。

进一步探讨

树莓派 2 具有 4 核处理器和 1GB 内存，足以运行一个像样的 Minecraft 服务器。其中，参数-Xms 512M 和-Xmx 800M 用来设置 Java 用于该服务器的最小和最大内存。

通过 LXTerminal 或者 SSH 窗口，你可以实时了解什么人连接到了该服务器上面，同时你也可以通过这些窗口来执行各种服务器管理命令。如果你输入 help 命令的话，就会得到一个详细的命令清单。

当你想要停止该服务器的时候，可以输入命令 stop。

```
[17:41:10 INFO]: UUID of player SimonMonk is 4e3ela27-4e7e-4bfb-b331-a30af5bd49ec
[17:41:11 INFO]: SimonMonk[/192.168.1.5:58831] logged in with entity id
382 at ([world]257.23244892748414, 67.0, 185.3728463324922)
>stop
[17:41:20 INFO]: Stopping the server
[17:41:20 INFO]: Stopping server
[17:41:20 INFO]: Saving players
[17:41:20 INFO]: SimonMonk lost connection: Server closed
[17:41:20 INFO]: SimonMonk left the game.
[17:41:21 INFO]: Saving worlds
[17:41:21 INFO]: Saving chunks for level 'world'/Overworld
[17:41:24 INFO]: Saving chunks for level 'world_nether'/Nether
[17:41:24 INFO]: Saving chunks for level 'world_the_end'/The End
>pi@Pi2headless ~ /mcserver $
```

为了允许互联网上的朋友也可以到你的服务器上玩，你需要通过自己的路由器管理控制台来设置端口转发。

参考资料

你不必直接连到树莓派来启动 Minecraft 服务器，即使通过 SSH，照样也能够很好地启动该服务器。

要想了解更多关于运行自己 Minecraft 服务器的信息，请参考 http://uk.ign.com/wikis/minecraft/Admin_and_Server_Commands。

此外，请参考树莓派版 Minecraft 的相关内容（见 4.7 节）。

4.9 运行 Open Arena

面临问题

你想要在树莓派上面运行 Quake 派生的 Open Arena。

解决方案

从树莓派商店下载并运行 Open Arena（见图 4-11）。

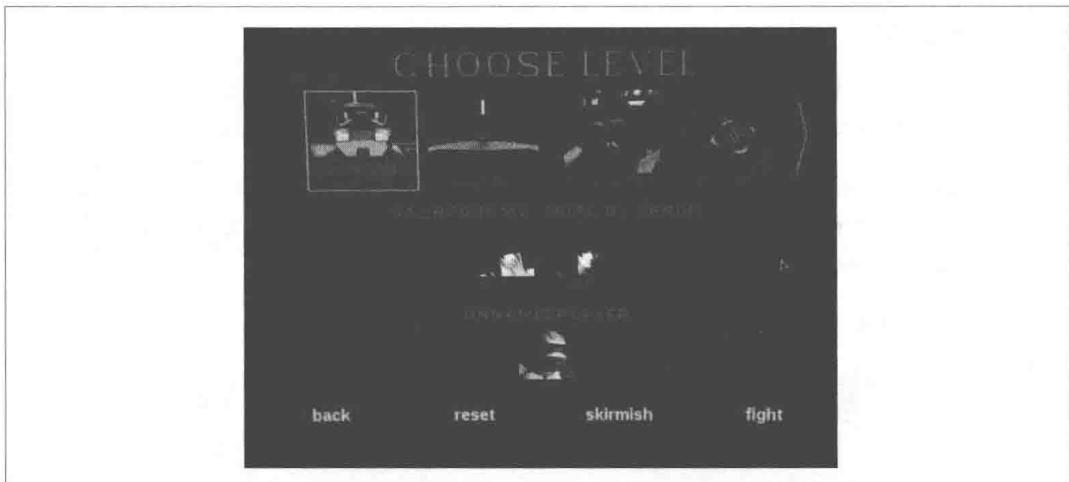


图 4-11 树莓派上的 Open Arena

进一步探讨

毫无悬念，你会在树莓派商店的 Games 部分找到 Open Arena。由于该游戏相当暴力，且血腥，所以它提供了一个简化版。

参考资料

关于 Open Arena 的更多内容，请参考 <http://www.openarena.ws/>。

关于树莓派商店的详细介绍，请参考 4.4 节。

4.10 树莓派无线电发射器



请务必观看在 <http://razzpisampler.oreilly.com> 上与本节有关的视频。

面临问题

你想把树莓派打造成一台大功率调频发射器，以便将无线电信号发送给普通的 FM 收音机（见图 4-12）。

解决方案

伦敦帝国学院的一伙聪明人已经创造了一些 C 代码，并用 Python 进行了封装，你可以藉此达成上面的目标。在下载的播放样本曲中，甚至包括来自《星球大战》的主题曲。

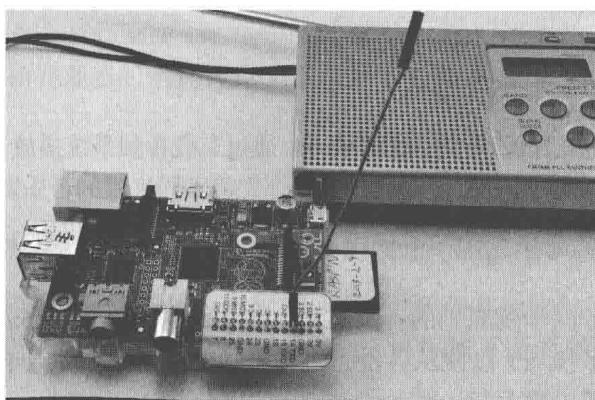


图 4-12 将树莓派打造成调频发射器

你唯一要做的就是用一根短导线连接到 GPIO 的 4 号引脚上面，为此，使用一根 female-to-male 接头的导线即可。事实上，如果收音机就在树莓派旁边的话，连天线之类的信号增强装置都用不着。

首先，你需要利用下列命令来安装 pifm 库。

```
$ mkdir pifm  
$ cd pifm  
$ wget http://www.icrobotics.co.uk/wiki/images/c/c3/Pifm.tar.gz  
$ tar -xzf Pifm.tar.gz
```

然后，找到一个 FM 收音机，并将频率调到 103.0MHz。如果该频率已经被其他发送装置占用的话，可以另选一个频率，并将其记下来。

现在，运行下列命令（如果你改变了频率的话，请修改最后面的参数，即 103.0）。

```
sudo ./pifm sound.wav 103.0
```

如果一切正常的话，你就应当听到激动人心的《星球大战》主题曲了。

进一步探讨

务必谨记，这个项目在你的国家可能是非法的。它的输出功率比 MP3 播放器的调频发射器要大。

你可以使用它来播放其他.wav 文件，不过这些文件必须是 16-bit 44.1kHz 的。

该代码还包括一个 Python 库，你可以将它用于自己的 Python 程序中。因此，你可以为它编写一个用户界面来选择和播放歌曲。

下面的代码展示了如何使用这个 Python 接口。

```
pi@raspberrypi ~ /pifm $ sudo python  
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
```

```
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import PiFm
>>> PiFm.play_sound("sound.wav")
```

你可以把树莓派放到自己的汽车内，这将是一个通过车载音频系统播放音频的极佳方式——特别是当你使用最新版本的 pifm 的时候，这允许您将不同来源的音乐发送给调频发射器。

参考资料

本节内容基于伦敦帝国学院原创的一篇文章，该文章的地址为 <http://bit.ly/18AcT5u>。这篇文章还提供了许多技巧，以便从各种音源（包括 USN 麦克风）向 pifm 输出音乐，从而为各种可能打开了一扇大门。

4.11 运行 GIMP

面临问题

你想要编辑图像。

解决方案

下载运行 GNU 图像处理程序（GIMP，见图 4-13）。

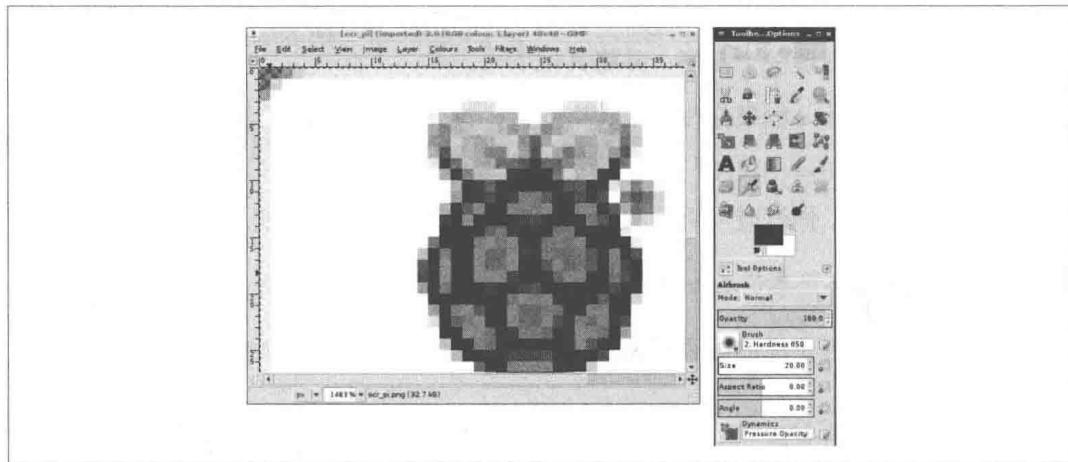


图 4-13 树莓派上面的 GIMP

为了安装 GIMP，请打开终端会话，并输入如下所示的命令。

```
$ sudo apt-get install gimp
```

一旦安装好了 GIMP，你就会在 Start 菜单的 Graphics 顶部发现一个新的菜单项，名为 GNU Image Manipulation Program。

进一步探讨

尽管 GIMP 是内存和处理器的消耗大户，但是对于 B 型树莓派来说，运行 GIMP 是完全吃得消的。

参考资料

要想进一步了解 GIMP 的用法，请访问 GIMP 的网站 (<http://www.GIMP.org/>)。

GIMP 是一款功能丰富的高级图像编辑程序，所以，要想掌握它的话，必须下功夫学习一番。为此，你可以访问 GIMP 网站 Documentation 标签下面的在线手册进一步学习。

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

4.12 互联网广播

面临问题

你想在树莓派上面收听互联网广播。

解决方案

安装 VLC 媒体播放器，具体命令如下所示。

```
sudo apt-get install vlc
```

一旦安装好该软件，就会在 Start 菜单的 Sound Video 部分找到 VLC。

运行该程序，并在 Media 菜单中选择 Open Network Stream 选项。

这时，将会出现一个对话框(见图 4-14)，你可以在此输入想要收听的互联网广播电台的 URL。

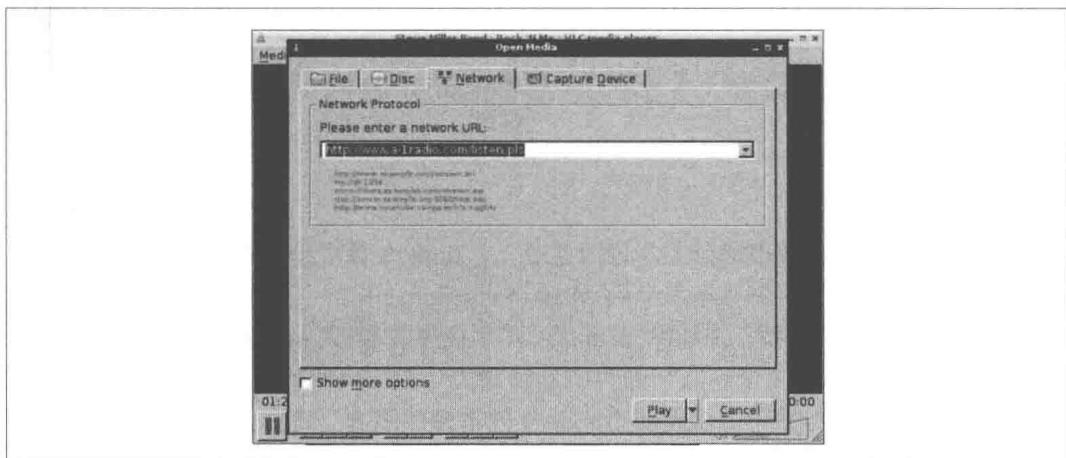


图 4-14 树莓派上的 VLC

此外，你还需要在树莓派的音频接口中插入耳机或扩音器。

进一步探讨

你也可以使用下列命令来运行 VLC。

```
$ vlc http://www.a-lradio.com/listen.pls -I dummy
```

尽管 VLC 有可能产生一系列出错信息，但是并不妨碍正常播放广播。

参考资料

本节内容主要引用自 <http://www.jan-holst.dk/pi-radio/pi-radio.html> 上的一篇教程，Jan Holst 在该教程中进行了深入的讲解，并为该工程添加了许多收音机样式的控件。

对于英国读者来说，可以从 <http://forum.chumby.com/viewtopic.php?id=7054> 页面上找到 BBC 电台的在线清单。

第 5 章

Python 入门

5.0 引言

对于树莓派来说，有许多编程语言可用，其中最流行的就是 Python 了。事实上，树莓派名字中的 Pi 就是受到单词 Python 的启发而取的。

本章提供了大量的示例代码，来帮助你掌握树莓派的编程技术。

5.1 在 Python 2 和 Python 3 之间做出选择

面临问题

你需要使用 Python，但是具体使用哪个版本却拿不定主意。

解决方案

两个版本都用。先使用 Python 3，直到遇到只有版本 2 才能完美解决的问题时，再回到 Python 2 的怀抱。

进一步探讨

虽然 Python 的最新的版本是 Python 3，并且该版本已经发布好几年了，但是许多人仍在坚守 Python 2。实际上，在 Raspbian 发行包中，两个版本都提供了：版本 2 名为 Python，而版本 3 则名为 Python 3。本书中的示例，除非特别指明，否则都是利用 Python 3 编写的。对于绝大部分示例代码来说，无需修改即可同时运行于 Python 2 和 Python 3 环境中。就 Python 社区而言，弃用旧版本的 Python 2 的阻力在于 Python 3 所引入的某些特性无法与版本 2 相兼容。也就是说，在为 Python 2 开发的巨量第三方库中，许多都无

法在 Python 3 下正常使用。

我的策略是在有可能的情况下，尽量使用 Python 3 编写代码，只有遇到兼容性问题的时候，万不得已才重新回到 Python 2 的怀抱。

参考资料

有关 Python 2 与 Python 3 的辩论文章，请参考 Python wiki (http://wiki.Python.org/moin/Python_2orPython)。

5.2 使用 IDLE 编辑 Python 程序

面临问题

你不知道应该使用什么来编写 Python 程序。

解决方案

在各种常见的树莓派发行包中，通常都同时提供了 Python 2 和 Python 3 两个版本的 IDLE Python 开发工具。如果你使用的是 Raspbian 的话，可以在 Start 菜单的 Programming 区找到名为 Python 2 和 Python 3 的两个版本的 IDLE 的快捷方式（见图 5-1）。



图 5-1 利用 IDLE 启动 Python

进一步探讨

IDLE 和 IDLE 3 外观看起来没什么区别，只是它们使用的 Python 的版本不同而已，所以不妨打开 IDLE 3（见图 5-2）。

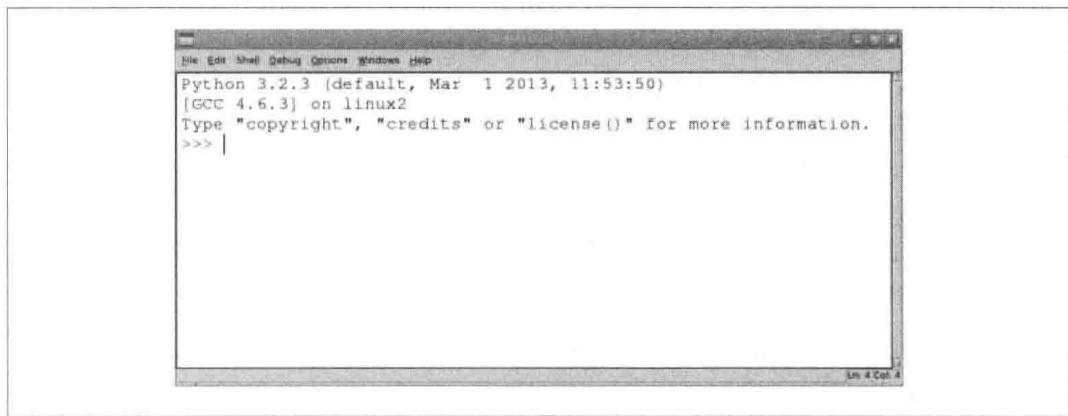


图 5-2 IDLE Python 控制台

打开的窗口名为 Python Shell。这是一个交互区域，你可以在此输入 Python 命令，然后马上就会看到返回的结果。所以，你可以尝试在该 shell 的>>>提示符后面输入下列命令。

```
>>> 2 + 2  
4  
>>>
```

Python 会计算表示式 $2+2$ 的值，并返回结果 4。

Python Shell 非常适合随性的尝试，不过要想编写程序的话，你就需要使用一个编辑器了。为了使用 IDLE 打开一个新文件进行编辑，可以从 File 菜单中选择 New Window 选项（见图 5-3）。



图 5-3 IDLE 编辑器

现在，你就可以在编辑器窗口中输入自己的程序代码了。作为一个尝试，你可以将下

列文本粘贴进该编辑器中，将文件保存为 count.py，然后，从 Run 菜单中选择 Run Module 选项。这样，你就可以从 Python 控制台中看到程序的运行结果了。

```
for i in range(1, 10):
    print(i)
```

作为一门编程语言，Python 的与众不同之处在于缩进是该语言的一个基本组成部分。许多基于 C 的编程语言都是利用{和}来界定代码块的，而 Python 则使用缩进级别来划分代码块。因此，对于上面的代码来说，Python 会将 print 作为循环的一部分而重复执行，因为它缩进了 4 个空格。

本书约定：使用 4 个空格作为一个缩进级别。



当你刚接触 Python 时，可能经常会遇到 IndentationError:unexpected indent 之类的错误提示，这些提示说明某些地方的缩进出现问题了。如果一切看起来都很正常的话，最好仔细检查各个缩进中是否存在制表符。要知道，Python 是会对制表符区别对待的。

此外，你还需注意观察 IDLE 是如何通过不同的颜色来高亮显示程序结构的。

参考资料

在本章中的许多示例都是通过 IDLE 来编辑 Python 代码的。

跟使用 IDLE 编辑运行 Python 文件类似，你也可以使用 nano 编辑文件（见 3.6 节），然后通过终端会话来运行它们（见 5.4 节）。

5.3 使用 Python 控制台

面临问题

你想要输入 Python 命令，而非编写完整的程序。

解决方案

你可以在 IDLE（见 5.2 节）或者终端会话中使用 Python 控制台。

进一步探讨

为了在终端窗口中启动一个 Python 2 控制台，只需输入命令 Python 即可；对于 Python 3 控制台，则需要输入命令 Python 3。

如果出现提示符>>>，则表明你可以输入 Python 命令了。如果你需要输入多行命令，那么控制台将会自动产生一个由 3 个点号指示的后续行。对于这些行来说，你仍然需要使用 4 个空格进行缩进，具体如下所示。

```
>>> for i in range(1, 10):
...     print(i)
...
1
2
3
4
5
6
7
8
9
>>>
```

在最后一条命令之后，你需要按两次回车键，控制台才能识别出这是缩进块的结束位置，继而运行这些代码。

此外，Python 控制台还提供了命令历史记录，你可以通过上下键来向前或向后选择运行过的命令。

参考资料

如果你想输入多行代码的话，最好还是使用 IDLE（见 5.2 节），将这些代码放到一个文件中进行编辑和运行。

5.4 利用终端运行 Python 程序

面临问题

虽然从 IDLE 内运行程序是个不错的方法，但是有时候你想从一个终端窗口来运行 Python 程序。

解决方案

在终端中使用 `python` 或者 `python 3` 命令，后面加上需要运行的程序文件的名称即可。

进一步探讨

为了从命令行运行一个 Python 2 程序，可以使用如下所示的代码。

```
$ python myprogram.py
```

如果你想要使用 Python 3 来运行程序的话，则需要将命令 `Python` 改为 `Python 3`。你要运行的 Python 程序应该位于扩展名为 `.py` 的文件中。虽然你可以用普通用户的身份来运行大部分 Python 程序，但是，对于某些程序，特别是使用了 GPIO 端口的程序来说，你必须具有超级用户权限才能够运行。如果你的程序要求具有超级用户权限的话，可以在命令前面加上前缀 `sudo`。

```
$ sudo python myprogram.py
```

在前面的这些例子中，命令中必须包含 Python 才能运行程序，不过，你还可以在 Python 程序的开头部分添加一行内容，这样 Linux 就能够知道它是一个 Python 程序了。这一特殊行通常称为 shebang (hash-bang 的缩写) 行，下面是由单行代码组成的一个示例程序，其中第一行就是 shebang 行。

```
#!/usr/bin/python  
print("I'm a program, I can run all by myself")
```

为了能够从命令行直接运行这个示例程序，你必须首先通过下列命令赋予该文件写权限（请参考 3.13 节）。

```
$ chmod +x test.py
```

其中，参数+x 表示添加执行权限。

现在，你只需要单条命令就能够执行这个名为 test.py 的 Python 程序了。

```
$ ./test.py  
I'm a program, I can run all by myself  
$
```

注意，对于上面这条命令来说，最前边的./是搜索该文件所必需的。

参考资料

在 3.23 节中介绍了如何通过定时事件来运行一个 Python 程序的方法。

要想在引导期间自动运行程序，请参考 3.22 节。

5.5 变量

面临问题

你想要给一个值取名。

解决方案

利用=给一个值指定名称。

进一步探讨

对于 Python 来说，你无需声明变量的类型，直接给它赋值即可，具体如下所示。

```
a = 123  
b = 12.34  
c = "Hello"  
d = 'Hello'
```

```
e = True
```

你可以使用单引号或双引号来定义字符串常量。在 Python 中，逻辑常量是 True 和 False，这里是大小写敏感的。

按照惯例，变量名以小写字母开头，如果其中包含多个单词的话，可以使用下画线将它们连接起来。给变量取一个描述性的名称，永远都是一个好主意。

像 x、total 和 number_of_chars，都是合法的变量名称。

参考资料

赋予变量的值，也可以是列表（见 6.1 节）或者字典（见 6.12 节）。

关于变量算术运算的介绍，详情请参考 5.8 节。

5.6 显示输出结果

面临问题

你想要查看变量的值。

解决方案

使用 print 命令。你可以在 Python 控制台（见 5.3 节）中尝试下面的示例代码。

```
>>> x = 10
>>> print(x)
10
>>>
```

进一步探讨

对于 Python 2 来说，你可以使用不带括号的 print 命令。但是，对于 Python 3 来说，这是不允许的，所以为了兼容这两个版本起见，请在待输出值的两边加上括号。

参考资料

要想读取用户的输入，请参考 5.7 节。

5.7 读取用户输入

面临问题

你想提示用户输入一个值。

解决方案

使用 `input` (Python 3) 或者 `raw_input` (Python 2) 命令。你可以在 Python 3 控制台 (见 5.3 节) 下面尝试下列代码。

```
>>> x = input("Enter Value:")
Enter Value:23
>>> print(x)
23
>>>
```

进一步探讨

使用 Python 2 时，必须将上面例子中的 `input` 替换为 `raw_input`。

实际上，Python 2 也有一个 `input` 函数，不过它的作用是验证输入，并试图将其转换为适当类型的 Python 值，而 `raw_input` 的作用跟 Python 3 中的 `input` 函数完全一致，都是返回一个字符串。

参考资料

关于 Python 2 的 `input` 函数的更多内容，请参考 <http://docs.python.org/2/library/functions.html#input>。

5.8 算术运算

面临问题

你想使用 Python 进行算术运算。

解决方案

使用运算符`+`、`-`、`*`和`/`。

进一步探讨

对于算术运算来说，最常用的运算符是`+`、`-`、`*`和`/`，分别对应于加法、减法、乘法和除法运算。

此外，你还可以像下例那样通过小括号对表达式的各个部分进行分组，在这个示例中，给出一个摄氏温度，它会把它转换为华氏度数。

```
>>> tempC = input("Enter temp in C: ")
Enter temp in C: 20
>>> tempF = (int(tempC) * 9) / 5 + 32
```

```
>>> print(tempF)
68.0
>>>
```

其他算术运算符还包括%（取模运算）和**（幂运算）。举例来说，为了求2的8次方，可以使用下列运算表达式。

```
>>> 2 ** 8
256
```

参考资料

关于input命令的用法，请参考5.7节；将输入的字符串转换为数字的方法，请参考5.12节。此外，Math库提供了许多常用的数学函数，你可以直接拿来使用，该库的地址为<http://docs.python.org/3.0/library/math.html>。

5.9 创建字符串

面临问题

你需要创建字符串变量。

解决方案

要想创建一个新的字符串，可以使用赋值运算符和字符串常量。你可以使用单引号或双引号来括住字符串，但是前后要匹配。

举例来说：

```
>>> s = "abc def"
>>> print(s)
abc def
>>>
```

进一步探讨

如果字符串本身包含单引号或双引号的话，那么可以选择字符串内没有用到的那种类型的引号来括住这个字符串。例如：

```
>>> s = "Isn't it warm?"
>>> print(s)
Isn't it warm?
>>>
```

有时候，字符串内部会包含一些特殊字符，如制表符和换行符等。这时候，可以使用转义字符来满足这一需求。为了包含一个制表符，可以使用\t，对于换行符，可以使用

\n，具体如下例所示。

```
>>> s = "name\tage\nMatt\t14"
>>> print(s)
name      age
Matt      14
>>>
```

参考资料

要想了解所有的转义字符，请参考 Python 参考手册 (<http://bit.ly/17Xxuqf>)。

5.10 连接（合并）字符串

面临问题

你想把多个字符串合并到一起。

解决方案

使用+（连结）运算符。

举例来说：

```
>>> s1 = "abc"
>>> s2 = "def"
>>> s = s1 + s2
>>> print(s)
abcdef
>>>
```

进一步探讨

对于许多编程语言来说，你可以连接一串的值，这些值中既可以有字符串类型，也可以有其他类型，比如数值类型；这种情况下，连接期间会自动将数字转换为字符串。但是，对于 Python 来说情况有所不同，如果你尝试输入下列命令的话，会得到一个错误。

```
>>> "abc" + 23
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

在连接它们之前，你需要将待连接的各个部分都转换为一个字符串，具体如下例所示。

```
>>> "abc" + str(23)
'abc23'
>>>
```

参考资料

至于利用 str 函数将数字转换为字符串的具体方法，请参考 5.11 节。

5.11 将数字转换为字符串

面临问题

你想把一个数字转换为一个字符串。

解决方案

你可以使用 Python 提供的 str 函数，具体如下所示。

```
>>> str(123)
'123'
>>>
```

进一步探讨

之所以要把数字转换为字符串，通常是因为只有如此，你才能够将它与别的字符串合并到一起（见 5.10 节）。

参考资料

关于将字符串转换为数字的逆运算，请参考 5.12 节。

5.12 将字符串转换为数字

面临问题

你想把一个字符串转换为一个数字。

解决方案

你可以使用 Python 提供的 int 或者 float 函数。

举例来说，为了将字符串 -123 变成一个数字，可以使用下列代码。

```
>>> int("-123")
-123
>>>
```

这个函数可以处理所有整数，无论符号是正，还是负。

要想把一个字符串转换为一个浮点数，那就不能使用 int 函数了，而应该使用 float 函数。

```
>>> float("00123.45")
123.45
>>>
```

进一步探讨

无论 int 函数，还是 float 函数，它们不仅都能正确处理数字前面部分的前导零，而且还能够容忍数字前后多余的空格符以及其他空白字符。

此外，int 函数还可以将一个表示非十进制数字的字符串转换为数字，为此，只需将基数作为第二个参数传递给这个函数即可。

在下面的例子中，会将表示二进制数字 1001 的字符串转换为一个数字。

```
>>> int("1001", 2)
9
>>>
```

在下面第二个示例中，会将十六进制数 AFF0 转换为一个整数。

```
>>> int("AFF0", 16)
45040
>>>
```

参考资料

关于将数字转换为字符串的逆运算，请参考 5.11 节。

5.13 确定字符串的长度

面临问题

你想知道字符串中包含了多少个字符。

解决方案

你可以使用 Python 提供的 len 函数。

进一步探讨

举例来说，为了确定字符串 abcdef 的长度，你可以使用下列代码。

```
>>> len("abcdef")
6
>>>
```

参考资料

此外，`len` 命令还可以用于列表（见 6.3 节）。

5.14 确定某字符串在另一个字符串中的位置

面临问题

你需要获得一个字符串在另一个字符串中的位置。

解决方案

你可以使用 Python 提供的 `find` 函数。

举例来说，为了确定字符串 `def` 在字符串 `abcdef ghi` 中的位置，你可以使用下列代码。

```
>>> s = "abcdefghijklm"
>>> s.find("def")
3
>>>
```

请注意，字符串的位置是从 0 开始计数的，也就是说，位置 3 表示的是字符串中的第 4 个字符。

进一步探讨

如果待查找的字符串实际上并不存在于正在搜索的字符串中的话，那么 `find` 函数的返回值为 -1。

参考资料

函数 `replace` 可以用来查找并替换字符串中所有的匹配项（见 5.16 节）。

5.15 截取部分字符串

面临问题

你想截取特定字符之间的部分字符串。

解决方案

你可以使用 Python 的 `[:]` 表达式。

举例来说，如果你想截取字符串 `abcdefghijklm` 中介于第 2 个字符到第 5 个字符之间的部分的话，可以使用下列代码。

```
>>> s = "abcdefghi"  
>>> s[1:5]  
'bcde'  
>>>
```

需要注意的是字符的位置是从 0 开始计数的，所以，位置 1 表示字符串中的第 2 个字符，位置 5 表示字符串中的第 6 个字符，不过，由于位置的范围并不包含最右边的数字，所以这里并不包括字母 f。

进一步探讨

实际上，[:] 的作用是非常强大的，其中的两个参数，任何一个都可以忽略掉，这时候字符串的开始和结束位置会视情况而定。举例来说：

```
>>> s[:5]  
'abcde'  
>>>
```

同时：

```
>>> s = "abcdefghi"  
>>> s[3:]  
'defghi'  
>>>
```

同时，你还可以使用负数来表示从字符串末尾开始反向计数。这种表示方法在某些情况下是非常有用的，比如当你想要获取某个文件名后面 3 个字母组成的扩展名的时候，具体如下例所示。

```
>>> "myfile.txt"[-3:]  
'txt'
```

参考资料

与本节介绍的分割字符串不同，在 5.10 节中，我们介绍了将字符串合并到一起的方法。在 6.10 节中也会用到同样的语法，不过它不是针对字符串，而是针对列表的。

5.16 使用字符串替换另一个字符串中的内容

面临问题

你想利用一个字符串替换另一个字符串中的所有匹配项。

解决方案

你可以使用 replace 函数。

举例来说，为了利用 `times` 替换所有的 `X`，可以使用如下所示的代码。

```
>>> s = "It was the best of X. It was the worst of X"  
>>> s.replace("X", "times")  
'It was the best of times. It was the worst of times'  
>>>
```

进一步探讨

待查找的字符串必须是严格匹配的，也就是说，不仅对大小写敏感，同时还要考虑空格。

参考资料

如果只需查找字符串，而无需进行替换的话，请参考 5.14 节。

5.17 字符串的大小写转换

面临问题

你想把字符串中的所有字符全部转换成大写字母或小写字母。

解决方案

你可以根据情况使用 `upper` 函数或者 `lower` 函数。

举例来说，为了将 `aBcDe` 转换为大写字母，可以使用如下所示的代码。

```
>>> "aBcDe".upper()  
'ABCDE'  
>>>
```

为了将这个字符串转换为小写字母，你可以使用下列代码。

```
>>> "aBcDe".lower()  
'abcde'  
>>>
```

进一步探讨

就像大部分处理字符串函数所做的那样，`upper` 和 `lower` 函数也不会实际修改字符串，而是返回一个修改后的字符串副本。

举例来说，下面的示例代码将会返回字符串 `s` 的副本，需要注意的是原始字符串本身并没有发生任何变化。

```
>>> s = "aBcDe"  
>>> s.upper()
```

```
'ABCDE'  
>>> s  
'aBcDe'  
>>>
```

如果你想把 s 的值全部转换为大写的话，可以使用如下所示的代码。

```
>>> s = "aBcDe"  
>>> s = s.upper()  
>>> s  
'ABCDE'  
>>>
```

参考资料

关于如何替换字符串中的文本的方法，请参考 5.16 节。

5.18 根据条件运行命令

面临问题

你希望只有当某些条件成立时才运行某些 Python 命令。

解决方案

你可以使用 Python 的 if 命令。

在下面的例子中，只有当 x 取值大于 100 的时候，才会输出消息 x is big。

```
>>> x = 101  
>>> if x > 100:  
...     print("x is big")  
...  
x is big
```

进一步探讨

在关键字 if 后面是一个条件表达式。这个条件表达式通常无非就是比较两个值，并返回 True 或者 False。如果返回值为 True，那么，后面缩进的各行代码就会被执行。

通常情况下，人们都是希望在条件为 True 的情况下执行某些操作，而在条件为 False 的情况下执行另外一些操作。如果遇到这种情况的话，可以将 else 命令与 if 命令联合起来使用，具体如下例所示。

```
x = 101  
if x > 100:  
    print("x is big")  
else:
```

```
print("x is small")  
print("This will always print")
```

你还可以利用一长串的 `elif` 语句将一组条件连接到一起。只要其中任何一个条件得到满足的话，就会执行对应的代码块，并且不再判断后续的条件分支。举例来说：

```
x = 90  
if x > 100:  
    print("x is big")  
elif x < 10:  
    print("x is small")  
else:  
    print("x is medium")
```

这段代码的输出结果是 `x is medium`。

参考资料

关于不同类型比较操作的详细信息，请参考 5.19 节。

5.19 值的比较

面临问题

你想对值进行比较。

解决方案

你可以使用下列比较运算符：`<`、`>`、`<=`、`>=`、`==`或者`!=`。

进一步探讨

在 5.18 节中，已经用过`<`（小于）和`>`（大于）运算符了。下面，我们将列出所有的比较运算符。

`<` 小于

`>` 大于

`<=` 小于等于

`>=` 大于等于

`==` 等于

`!=` 不等于

虽然有些人更喜欢使用`<>`，而非`!=`，但是两者的作用是完全一样的。

你可以在 Python 控制台（见 5.3 节）下面测试这些命令，如下所示。

```
>>> 1 != 2
True
>>> 1 != 1
False
>>> 10 >= 10
True
>>> 10 >= 11
False
>>> 10 == 10
True
>>>
```

在进行比较运算时，一个常见的错误是使用了`=`（赋值），而非`==`（双等号）。这个错误通常比较隐蔽，因为如果比较运算的一边是个变量的话，那么这是完全合法的并且会被正确执行的，但是，却无法得到预期的结果。

就如同比较数字一样，你也可以使用这些操作符来比较字符串，具体用法如下所示。

```
>>> 'aa' < 'ab'
True
>>> 'aaa' < 'aa'
False
```

对字符串进行比较是按照字典方式进行的，即按照它们在字典中出现的顺序进行比较。

当然，这种方式并非十分完美，因为对于每个字母来说，通常认为大写字母要小于小写字母。每个字母都有一个值，即它的 ASCII 编码 (<https://en.wikipedia.org/wiki/ASCII>)。

参考资料

请同时参考 5.18 节和 5.20 节的内容。

5.20 逻辑运算符

面临问题

你需要使用 if 语句描述一个复杂的条件。

解决方案

使用下列逻辑运算符：`and`、`or` 和 `not`。

进一步探讨

举例来说，你想要检查变量 `x` 的值是否介于 10 ~ 20 之间。为此，你可以使用 `and` 运算符。

```
>>> x = 17
```

```
>>> if x >= 10 and x <= 20:  
...     print('x is in the middle')  
...  
x is in the middle
```

如果需要的话，你可以将任意多个 and 和 or 语句组合起来使用，同时，你还可以利用括号对复杂的表达式进行分组。

参考资料

请同时参考 5.18 节和 5.19 节。

5.21 将指令重复执行特定次数

面临问题

你需要将某些程序代码重复执行特定的次数。

解决方案

你可以使用 Python 的 for 命令，并指定迭代次数。

举例来说，如果要将一个命令重复执行 10 次，可以使用如下所示的代码。

```
>>> for i in range(1, 11):  
...     print(i)  
...  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
>>>
```

进一步探讨

这里，range 命令的第二个参数并不包括在范围之内，也就是说，如果要想计算到 10 的话，该参数的值必须指定为 11。

参考资料

对于终止循环的条件要远比这里重复执行特定次数更为复杂的情况，请参考 5.22 节。

如果你想对列表或字典逐元素执行某些命令的话，请分别参考 6.7 节或 6.15 节。

5.22 重复执行指令直到特定条件改变为止

面临问题

你需要重复执行某些程序代码，直到某些条件发生变化为止。

解决方案

你可以使用 Python 的 while 语句。该语句会重复执行嵌套在其内部的语句，直到它的条件不再成立为止。在下面的例子中，while 语句内嵌的代码会一直循环执行下去，直到用户输入 X 后才会退出循环。

```
>>> answer = ''  
>>> while answer != 'X':  
...     answer = input('Enter command:')  
...  
Enter command:A  
Enter command:B  
Enter command:X  
>>>
```

进一步探讨

需要注意的是前面的示例中所用的 input 命令只适用于 Python 3。如果要在 Python 2 中运行该代码的话，可以使用 raw_input 函数替换掉 input 函数。

参考资料

如果你只是想让某些命令执行特定的次数的话，可以参考 5.21 节。

如果你想对列表或字典逐元素执行某些命令的话，那么请分别参考 6.7 节或 6.15 节。

5.23 跳出循环语句

面临问题

在执行循环语句的时候，如果遇到某些情况，你将需要退出循环过程。

解决方案

你可以使用 Python 的 break 语句来退出 while 循环或 for 循环。

下面例子中的代码的行为与 5.22 节中的完全一致。

```
>>> while True:  
...     answer = input('Enter command:')
```

```
...     if answer == 'X':  
...         break  
...  
Enter command:A  
Enter command:B  
Enter command:X  
>>>
```

进一步探讨

需要注意的是这个示例中所用的 `input` 命令只适用于 Python 3。如果要在 Python 2 中运行该代码的话，可以使用 `raw_input` 函数替换掉 `input` 函数。这个示例代码的行为与 5.22 节中的代码的行为完全一致。不过，就本例而言，`while` 循环的条件为 `True`，所以该循环将一直进行下去，直到用户输入 `X` 后，才会通过 `break` 语句退出循环。

参考资料

此外，你还可以通过 `while` 语句本身的条件来退出循环，具体请参考 5.18 节。

5.24 定义 Python 函数

面临问题

你想避免在程序中一遍又一遍地重复同样的代码。

解决方案

创建一个函数，将多行代码组织在一起，以便可以从多个不同的地方来调用它。

下面的示例代码展示了如何在 Python 中创建和调用函数。

```
def count_to_10():  
    for i in range(1, 11):  
        print(i)  
  
count_to_10()
```

在这个例子中，我们利用 `def` 命令定义了一个函数，每当调用该函数时，它会打印输出一个介于 1 到 10 之间的数字。

```
count_to_10()
```

进一步探讨

函数的命名惯例与变量完全一致，具体请参考 5.5 节。也就是说，函数名应该以小写字母开头，如果名称由多个单词组成的话，可以利用下画线来分隔单词。

上面的示例函数不是太灵活，因为它只能计算到 10。如果想要提高该函数的灵活性以便可以计算到任意数字的话，可以在函数中添加一个参数来表示最大值，具体如下所示。

```
def count_to_n(n):
    for i in range(1, n + 1):
        print(i)

count_to_n(5)
```

参数名 `n` 需要放到小括号中，之后，`range` 命令会用到这个参数，但不是直接使用它，而是在加 1 之后。

使用一个参数来表示想要计数到的数字的时候，如果通常要计数到 10，而有时候又要计数到其他数字的话，那么你就不得不每次都规定一个数字。实际上，你可以为参数指定一个默认值，这样就能够两者兼顾了，具体如下所示。

```
def count_to_n(n=10):
    for i in range(1, n + 1):
        print(i)

count_to_n()
```

这样的话，每次调用该函数时，这个参数的值都是 10，除非指定了其他数字。

如果你的函数需要多个参数的话，比如需要计算两个数字之间的数字个数的话，那么可以使用逗号来分隔参数。

```
def count(from_num=1, to_num=10):
    for i in range(from_num, to_num + 1):
        print(i)

count()
count(5)
count(5, 10)
```

上面这些示例代码中的所有函数都没有返回任何值，它们只是进行了某些计算。

如果你想让函数返回值的话，则需要使用 `return` 命令。

下列函数以字符串作为其参数，并在该字符串后面追加单词 `please`。

```
def make_polite(sentence):
    return sentence + " please"

print(make_polite("Pass the cheese"))
```

当函数返回值时，你可以将结果赋值给一个变量，或者像本例中那样，直接打印出结果。

参考资料

如果想要从一个函数中返回多个值的话，可以参见 7.3 节。

Python 语言中的列表与字典

6.0 引言

通过第 5 章的学习，我们已经了解了 Python 语言的基本知识。在这一章中，我们将学习两种关键的 Python 数据结构：列表和字典。

6.1 创建列表

面临问题

你想利用一个变量来存放一系列的值，而非单个值。

解决方案

你可以使用列表。在 Python 语言中，列表是按顺序存放的一组值的集合，所以，你可以通过位置来访问这些值。

你可以使用[和]来创建列表，两个方括号之间可以放入该列表的初始值。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>>
```

与类似于 C 之类的语言不同，它们对数组的定义非常严格，而 Python 在声明列表的时候，根本无需指定列表的长度。只要你喜欢，你可以随时改变列表中元素的数量。

进一步探讨

就像你在上面的例子中看到的那样，列表中的元素的类型不要求完全一致，当然，大部分情况下它们通常都是同类型的。

如果你想创建一个空列表，以便在将来需要时添加元素的话，你可以使用下列方式来新建空列表。

```
>>> a = []
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.2 访问列表元素

面临问题

你需要找到列表中的特定元素，或者修改它们。

解决方案

你可以使用`[]`表达式，通过指定元素在列表中的位置来访问它们，例如：

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1]
'Fred'
```

进一步探讨

列表中的位置（下标）是从 0 开始计数的，即第一个元素的下标为 0。

就像使用`[]`表达式从列表中读取元素值那样，你也可以使用它来修改指定位置的元素的值，例如：

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1] = 777
>>> a
[34, 777, 12, False, 72.3]
```

如果你试图利用一个很大的下标来修改（或就这里来说，读取）一个元素的话，就会收到一个“Index out of range” 错误消息。

```
>>> a[50] = 777
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.3 确定列表长度

面临问题

你需要知道列表中含有多少个元素。

解决方案

你可以使用 Python 提供的 len 函数，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> len(a)
5
```

进一步探讨

此外，len 命令还可用于字符串（见 5.13 节）。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.4 为列表添加元素

面临问题

你希望为列表添加元素。

解决方案

为此，你可以使用 Python 提供的 append、insert 或者 extend 函数。

要想在列表末尾添加单个元素的话，你可以使用 append 函数。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.append("new")
>>> a
[34, 'Fred', 12, False, 72.3, 'new']
```

进一步探讨

有时候，你想要做的并非在列表尾部添加新元素，而是要把新元素添加到指定的位置。

为此，你可以使用 insert 命令。该命令的第一个参数是希望插入元素的位置的索引，第二个参数是希望插入的元素，具体如下所示。

```
>>> a.insert(2, "new2")
>>> a
[34, 'Fred', 'new2', 12, False, 72.3]
```

需要注意的是新插入元素之后，其后面元素的下标会依次加 1。

`append` 和 `insert` 函数每次只能向列表中添加一个元素。而对于 `extend` 函数来说，它可以将一个列表中的所有元素一次性添加到另一个列表的末尾。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> b = [74, 75]
>>> a.extend(b)
>>> a
[34, 'Fred', 12, False, 72.3, 74, 75]
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.5 删 除 列 表 元 素

面 临 问 题

你需要从列表中删除元素。

解 决 方 案

你可以使用 Python 提供的 `pop` 函数。

当使用 `pop` 命令时，如果没有参数的话，它会删除列表中的最后一个元素。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop()
72.3
>>> a
[34, 'Fred', 12, False]
```

进 一 步 探 讨

请注意，`pop` 函数会返回从列表中删除的元素的值。

要想删除列表中指定位置的元素，而非最后一个元素的话，可以给 `pop` 命令指定待删除元素的位置，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop(0)
34
```

如果你使用了一个列表尾部之外的位置下标的话，将会收到一个“Index out of range”异常消息。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.6 通过解析字符串创建列表

面临问题

你需要将一个包含由特定字符隔开的单词的字符串转换成一个字符串数组，并且让该数组中的每个字符串都是一个单词。

解决方案

你可以使用 Python 提供的字符串函数 `split`。

在使用 `split` 命令的时候，如果不带参数的话，它会把字符串中的单词分割为数组的单个元素，具体如下所示。

```
>>> "abc def ghi".split()  
['abc', 'def', 'ghi']
```

你可以使用含参数的 `split` 函数来拆分字符串，使用该参数作为一个分割器，具体如下所示：

```
>>> "abc--de--ghi".split('--')  
['abc', 'de', 'ghi']
```

进一步探讨

当你需要从文件导入数据的时候，`split` 命令是非常有用的。同时，该命令还有一个可选参数，这个参数通常是一个字符串，用来指定分割字符串时所用的分隔符。所以，如果你想使用逗号作为分隔符的话，可以像下面这样来分割字符串。

```
>>> "abc,def,ghi".split(',')  
['abc', 'def', 'ghi']
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.7 遍历列表

面临问题

你需要逐个对列表中的所有元素执行某些操作。

解决方案

你可以使用 Python 语言提供的 for 命令，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for x in a:
...     print(x)
...
34
Fred
12
False
72.3
>>>
```

进一步探讨

在关键字 for 之后，需要紧跟一个变量名（本例为 x）。这个变量称为循环变量，它的取值将遍历关键字 in 之后的列表中的每一个元素。

For 语句下面的缩进将会对列表中的每个元素都执行一次。

每循环一次，x 的值就会被设置为对应位置的元素的值。实际上，你可以使用 x 输出相应的值，具体见上例所示。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.8 枚举列表

面临问题

你需要对列表中的每个元素逐次执行某些代码，同时还需要知道各个元素的位置下标。

解决方案

你可以使用 Python 语言提供的 for 命令与 enumerate 命令。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for (i, x) in enumerate(a):
...     print(i, x)
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
```

```
(4, 72.3)
```

```
>>>
```

进一步探讨

在枚举每个值的时候，通常都需要知道它们在列表中的相应的位置。一个替代方法是简单使用一个下标变量进行计数，然后通过[]语法来访问相应的值，如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for i in range(len(a)):
...     print(i, a[i])
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
(4, 72.3)
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

要想在不知道每个元素的下标位置的情况下遍历列表的话，请阅读 6.7 节。

6.9 列表排序

面临问题

你需要对列表中的元素进行排序。

解决方案

你可以使用 Python 语言提供的 sort 命令。

```
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> a.sort()
>>> a
['best', 'it', 'of', 'the', 'times', 'was']
```

进一步探讨

当对一个列表进行排序的时候，你实际上是对列表本身进行了相应的修改，而非返回了原列表排序之后的一个副本。这就是说，如果你还需要原始列表的话，那么在对其进行排序之前，需要先用标准程序库中的 copy 命令生成原始列表的副本。

```
>>> import copy
```

```
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> b = copy.copy(a)
>>> b.sort()
>>> a
['it', 'was', 'the', 'best', 'of', 'times']
>>> b
['best', 'it', 'of', 'the', 'times', 'was']
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.10 分割列表

面临问题

你需要利用原列表的部分内容来生成一个子列表。

解决方案

为此，你可以使用 Python 语言的[:] 结构。下面的例子将返回一个列表，其内容由原来列表中下标位置 1 到下标位置 2（位于：之后的数字将会被排除在外）之间的元素所组成。

```
>>> l = ["a", "b", "c", "d"]
>>> l[1:3]
['b', 'c']
```

需要注意的是字符的位置是从 0 开始计数的，所以，位置 1 表示字符串中的第 2 个字符，位置 5 表示字符串中的第 6 个字符，不过，由于位置的范围并不包含最右边的数字，所以这里并不包括字母 d。

进一步探讨

实际上，[:] 的作用是非常强大的，其中的两个参数，任何一个都可以忽略掉，这时候列表的开始和结束位置会视情况而定。举例来说：

```
>>> l = ["a", "b", "c", "d"]
>>> l[:3]
['a', 'b', 'c']
>>> l[3:]
['d']
>>>
```

此外，你还可以使用负数来表示从字符串的末尾开始反向计数。下面的例子将会返回列表中最后两个元素。

```
>>> l[-2:]
```

```
['c', 'd']
```

顺便提一句，上面的例子中，`l[:-2]`将返回`['a','b']`。

参考资料

在 6.1 节 ~ 6.11 节中的示例中，都会涉及列表的应用。

你还可以参考 5.15 节，其中的语法同样适用于字符串。

6.11 将函数应用于列表

面临问题

你需要对列表中的每个元素都应用某个函数，并收集相应的结果。

解决方案

你可以使用 Python 语言的一个特性，该特性名为推导式（comprehensions）。

下面的例子会将列表的每个字符串元素转换为大写形式，并返回长度与原来相同的一个新列表，只是所有字符串都已经变成大写了。

```
>>> l = ["abc", "def", "ghi", "ijk"]
>>> [x.upper() for x in l]
['ABC', 'DEF', 'GHI', 'IJK']
```

尽管这种方式有些乱，但是仍然没有理由来拒绝联合使用这种语句，将一个推导式嵌入到另一个语句中。

进一步探讨

这是使用推导式的一种非常简洁的方法。整个表达式都要求放入方括号（[]）之中。推导式的第一个元素是需要施加于列表各元素的代码。推导式的其余部分看上去更像是一个列表遍历命令（见 6.7 节）。循环变量紧跟在关键字 `for` 后面，关键字 `in` 之后是需要用到的列表。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.12 创建字典

面临问题

你需要创建一个查找表，其中的值和键已经关联在一起。

解决方案

你可以使用 Python 的字典。

当你需要按照顺序访问一组元素，或者总是知道想要使用元素的下标时，使用数组是你的不二之选。字典可以代替列表来存放一组数据，不过两者之间的组织方式区别较大。

phone_numbers	
Key: Simon	Value: 01234 567899
Key: Jane	Value: 01234 666666
Key: Pete	Value: 01234 777555
Key: Linda	Value: 01234 887788

图 6-1 字典的组织方式

图 6-1 所示 Python 的字典使用的是键/值对的方式，也就是说你可以使用键来高效地访问值，而不必搜索整个字典。

为了创建一个字典，可以使用 {} 表达式。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
```

进一步探讨

在上面的例子中，字典的键都是些字符串，但是这不是硬性要求；这些键也可以是数字，事实上任何数据类型都是允许的，不过字符串是最常用的一种形式。

字典的值也可以是任何类型的数据，包括其他字典或者列表。下面的例子中，我们创建了一个字典 (a)，并且随后将其作为了第二个字典 (b) 的值。

```
>>> a = {'key1':'value1', 'key2':2}
>>> a
{'key2': 2, 'key1': 'value1'}
>>> b = {'b_key1':a}
>>> b
{'b_key1': {'key2': 2, 'key1': 'value1'}}
```

当你显示字典的内容时，你可能会发现，字典中的各个元素的顺序与当初创建并利用某些内容来初始化该字典时的顺序并不一定一致。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers
{'Jane': '01234 666666', 'Simon': '01234 567899'}
```

与列表不同的是字典并没有按顺序存放元素的要求。正是由于这种内在的表示方式，导致字典元素的顺序无论怎么看都是随机的。

导致顺序随机的原因在于这种基本数据结构实际上是一种哈希表。哈希表利用哈希函

数来决定将值存放到哪里，哈希函数能够为任意对象产生一个对应的数字。

要想了解哈希表的更多知识，请访问 Wikipedia(http://en.wikipedia.org/wiki/Hash_table)。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及字典的应用。

6.13 访问字典

面临问题

你需要查找和修改字典元素。

解决方案

你可以使用 Python 的`[]`表达式。在括号中，你需要指定待访问的元素的键，具体如下所示。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Simon']
'01234 567899'
>>> phone_numbers['Jane']
'01234 666666'
```

进一步探讨

这个查找只能在单个方向上进行，即从键到值。

如果你使用的键并不存在于该字典中的话，那么就会收到一个 key error，具体如下例所示。

```
{'b_key1': {'key2': 2, 'key1': 'value1'}}
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Phil']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Phil'
>>>
```

就像使用`[]`表达式从字典中读取值一样，你还可以使用它来添加新的值，或者覆盖现有的值。

下面的例子展示了如何向字典中新加一个元素，该元素的键和值分别为 Pete 和 01234777555。

```
>>> phone_numbers['Pete'] = '01234 777555'
>>> phone_numbers['Pete']
'01234 777555'
```

如果该键尚未被这个字典所用的话，就会自动添加一个新元素。如果该键已经存在的话，那么无论之前该键对应的值是什么，都会被新值所覆盖掉。

这一点与从字典中读取值有所不同，那种情况下，使用未知的键会导致出错。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

关于错误处理的详细情况，请参考 7.10 节。

6.14 删除字典元素

面临问题

你需要从字典中删除元素。

解决方案

你可以使用 `pop` 命令，同时指定要删除的元素的键。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}  
>>> phone_numbers.pop('Jane')  
'01234 666666'  
>>> phone_numbers  
{'Simon': '01234 567899'}
```

进一步探讨

命令 `pop` 将会返回从字典中删除的元素的值。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

6.15 遍历字典

面临问题

你想对字典中的所有元素依次执行某种操作。

解决方案

你可以使用 `for` 命令来遍历字典的键。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name in phone_numbers:
...     print(name)
...
Jane
Simon
```

进一步探讨

还有其他几种方法，也可以用来遍历字典。

如果你需要同时访问键和值的话，下面的方法将非常有用。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name, num in phone_numbers.items():
...     print(name + " " + num)
...
Jane 01234 666666
Simon 01234 567899
```

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

涉及 for 命令的用法的小节，包括 5.21 节、6.7 节和 6.11 节。

Python 高级特性

7.0 引言

在本章中，我们将介绍 Python 语言的一些高级概念，特别是面向对象的 Python、文件的读写、异常处理、使用模块和因特网编程。

7.1 格式化数字

面临问题

你想把数字格式化为指定的小数位。

解决方案

你可以对数字应用格式串，具体如下所示。

```
>>> x = 1.2345678
>>> "x={:.2f}".format(x)
'x=1.23'
```

进一步探讨

用于格式化的字符串可以由常规文本和标志组成，并且需要用{和}进行分隔。函数 format 的参数（可以有任意多个）会根据格式说明符来取代标志。

在前面的示例中，格式说明符为:.2f，表示数字被指定小数点后面保留两位数，同时其类型为浮点型 f。

如果你想把数字格式化为总长度为 7 位（不够用空格补齐）的话，那么就需要在小数

点前面再加上一个数字，具体如下所示。

```
>>> "x={:7.2f}".format(x)
'x=    1.23'
>>>
```

就本例而言，由于该数字长度只有 3 位，所以会在 1 之前填充 4 个空格。如果你想使用前导零的方式进行补齐的话，可以使用下列代码。

```
>>> "x={:07.2f}".format(x)
'x=0001.23'
>>>
```

下面是一个稍微复杂的例子，这里要使用摄氏度数和华氏度数来表示温度，具体代码如下所示。

```
>>> c = 20.5
>>> "Temperature {:5.2f} deg C, {:5.2f} deg F.".format(c, c * 9 / 5 + 32)
'Temperature 20.50 deg C, 68.90 deg F.'
>>>
```

参考资料

Python 语言中的格式化技术有一套完整的格式化语言（<http://bit.ly/cwzPp5>）。

7.2 格式化时间和日期

面临问题

你想把一个日期转化为字符串，并且按照特定的方式进行格式化。

解决方案

你可以将格式串应用到日期对象上面。

举例来说：

```
>>> from datetime import datetime
>>> d = datetime.now()
>>> "{:%Y-%m-%d %H:%M:%S}".format(d)
'2015-12-09 16:00:45'
>>>
```

进一步探讨

Python 格式化语言为格式化日期提供了许多特殊符号。

%Y、%m 和%d 分别对应于年、月和日。

其他常用于格式化日期的符号还有%B 和%Y，前者用于提供月份的全称，后者提供 4

位数字表示的年份，具体如下所示。

```
>>> "{:%d %B %Y}".format(d)
'09 December 2015'
```

参考资料

关于数字格式化的内容，请参考 7.1 节。

Python 语言中的格式化技术，涉及到一套完整的格式化语言。这方面的完整说明请参考 <http://strftime.org/>。

7.3 返回多个值

面临问题

你想编写一个返回多个值的函数。

解决方案

你可以返回 Python 元组，并使用多变量赋值语法。元组是一种 Python 数据结构，与列表比较相似，只不过元组是用小括号而非方括号括起来的。此外，两者的大小都是固定的。

例如，你可以建立一个函数，来将温度的绝对温标转换为华氏温标和摄氏温标。你可以让这个函数同时返回两种温标表示的温度，为此可以使用逗号将多个返回值隔开，具体如下所示。

```
>>> def calculate_temperatures(kelvin):
...     celsius = kelvin - 273
...     fahrenheit = celsius * 9 / 5 + 32
...     return celsius, fahrenheit
...
>>> c, f = calculate_temperatures(340)
>>>
>>> print(c)
67
>>> print(f)
152.6
```

调用该函数时，你只需在=前面提供跟返回值数量一致的变量即可，并且每个返回值都会按照同样的位置赋给相应的变量。

进一步探讨

当只有一两个值需要返回时，上面介绍的这种返回多个值的方式是比较合适的。但是，如果数据非常复杂的话，那么你会发现，使用 Python 面向对象特性来定义一个存放数据的类

将是更加优雅的一种解决方案。这样的话，你可以返回一个类的实例，而非元组。

参考资料

至于类的定义方法，请参考 7.4 节。

7.4 定义类

面临问题

你需要将相关数据和函数整合到一个类中。

解决方案

你可以定义一个类，并根据需要提供相应的成员变量。

下面的代码定义了一个类，用来表示通讯录记录。

```
class Person:  
    '''This class represents a person object'''  
  
    def __init__(self, name, tel):  
        self.name = name  
        self.tel = tel
```

在上面的类定义中的第一行，使用了三引号来指明这是一个文档性质的字符串。这个字符串通常是用来介绍该类的用途的。当然，这个字符串是完全可选的，但是为类添加了文档字符串之后，可以帮助人们了解这个类是做什么的。尤其是当定义的类是供别人使用的时候，这个字符串将格外有用。

与普通注释字符串不同的是虽然注释字符串本身不是有效代码，但是却与类紧密相联，所以，无论何时，你都可以通过下列命令来读取类的文档字符串。

```
Person.__doc__
```

在类定义的内部是构造函数，每当为该类新建一个实例的时候，都会自动调用这个函数。类与模板比较类似，因此在定义名为 Person 的类的时候，我们并没有创建任何实际的 Person 对象，直到像下面这样生成实例为止。

```
def __init__(self, name, tel):  
    self.name = name  
    self.tel = tel
```

构造函数的命名，必须像上面那样，在单词 init 前后分别加上一个下画线。

进一步探讨

Python 与大部分面向对象编程语言的一个不同之处在于在类内部定义的所有方法，

都必须包含特殊变量 `self` 来作为其参数。在这种情况下，变量 `self` 就是对新建实例的引用。

从概念上讲，变量 `self` 与 Java 等其他编程语言中的特殊变量 `this` 是等价的。

这个方法中的代码会将提供给它的参数传递给成员变量。成员变量无需提前声明，但是，必须带有前缀 `self.`。

所以，下面的代码：

```
self.name = name
```

将会生成一个名为 `name` 的变量，它可以供类 `Person` 的所有成员访问，同时，该变量是利用传递给创建实例的调用的值来初始化的，具体如下所示。

```
p = Person("Simon", "1234567")
```

这样，我们就可以利用下面的代码来查看新建的 `Person` 对象 `p` 的名称是否为“`Simon`”了。

```
>>> p.name  
Simon
```

对于复杂的程序来说，可以将每个类单独放到一个文件中，并且以该类的名称给文件命名，这是一种非常好的做法。此外，这种做法也使得将类转换为模块更加容易（见 7.11 节）。

参考资料

关于定义方法的内容，请参考 7.5 节。

7.5 定义方法

面临问题

你需要为类添加方法。

解决方案

下面的示例代码展示了如何在类的定义中包含一个方法。

```
class Person:  
    '''This class represents a person object'''  
    def __init__(self, first_name, surname, tel):  
        self.first_name = first_name  
        self.surname = surname  
        self.tel = tel  
  
    def full_name(self):  
        return self.first_name + " " + self.surname
```

方法 `full_name` 的作用是把一个人的姓和名连起来，并以空格作为间隔。

进一步探讨

实际上，你可以把方法看作是绑定到特定类上的一些函数，它们既可以使用这个类的成员变量，也可以不使用这个类的成员变量。所以，正如函数那样，你不仅可以在方法内部写入任何所需代码，也可以从一个方法中调用其他方法。

如果一个方法调用了同一个类中的其他方法，那么调用时必须在被调用方法之前添加前缀 `self`。

参考资料

关于定义类的内容，请参考 7.4 节。

7.6 继承

面临问题

你需要为一个现有的类制作一个特殊版本。

解决方案

你可以利用继承为现有的类创建一个子类，并为其添加新的成员变量和方法。

在默认的情况下，你新建的所有类都是 `object` 的子类。要想改变这种状况，在定义类的时候，可以在类名称之后的括号中加入想要使用的超类。下面的示例代码定义了一个 `Person` 的子类 (`Employee`)，并添加了一个新的成员变量 (`salary`) 和另外一个方法 (`give_raise`)。

```
class Employee(Person):

    def __init__(self, first_name, surname, tel, salary):
        super().__init__(first_name, surname, tel)
        self.salary = salary
    def give_raise(self, amount):
        self.salary = self.salary + amount
```

需要注意的是上面的示例代码适用于 Python 3。对于 Python 2 来说，不允许通过这种方式来使用 `super`，而应该采用下列所示的方式。

```
class Employee(Person):

    def __init__(self, first_name, surname, tel, salary):
        Person.__init__(self, first_name, surname, tel)
        self.salary = salary

    def give_raise(self, amount):
```

```
self.salary = self.salary + amount
```

进一步探讨

在上面的两个示例中，子类的初始化方法都是首先调用父类（超类）的初始化方法，然后才添加成员变量。这样做的好处在于无需在新的子类中重复初始化代码。

参考资料

关于定义类的内容，请参考 7.4 节。

Python 的继承机制实际上非常强大，并且支持多重继承，也就是说一个子类可以继承自多个超类。要想了解继承的更多内容，请参考 Python 的官方文档 (<http://bit.ly/17Y2c2k>)。

7.7 向文件中写入内容

面临问题

你想向一个文件中写入某些内容。

解决方案

你可以分别使用 `open`、`write` 和 `close` 函数来打开一个文件，并向其中写入某些数据，然后关闭该文件。

```
>>> f = open('test.txt', 'w')
>>> f.write('This file is not empty!')
>>> f.close()
```

进一步探讨

文件一旦打开，在其被关闭之前，你可以随意向其中写入内容，次数不限。需要注意的是使用 `close` 函数关闭文件是非常重要的，因为，虽然每次写操作都应该即时更新文件，但是写入的内容也可能被缓存在内存中，所以这些数据还是有可能会丢失的。此外，如果不关闭文件的话，会导致该文件被锁住，以及其他程序无法打开该文件。

方法 `open` 需要两个参数，第一个参数是需要进行写操作的文件的所在路径。

这个路径既可以是基于当前工作目录的相对路径，也可以是从/目录开始的绝对路径。

第二个参数是文件的打开模式。如果要覆写现有的文件，或者文件不存在时按照指定的文件名创建文件的话，可以使用参数 `w`。

表 7-1 给出了文件模式字符的完整列表。你可以利用+来组合各种模式字符。

所以，如果要以二进制读的模式来打开一个文件的话，你可以使用下列代码。

```
>>> f = open('test.txt', 'r+b')
```

表 7-1

文件模式模式说明

R	读
W	写
A	追加——将内容追加到现有文件末尾，而非覆盖现有文件
B	二进制模式
T	文本模式（默认）
+	r+w 组合模式的缩写

二进制模式允许你对二进制数据流进行读写操作，二进制数据流包括图像等数据，但是文本不属此列。

参考资料

若要读取文件的内容，请参考 7.8 节。若想进一步了解异常处理，请参考 7.10 节。

7.8 读文件

面临问题

你想把文件的内容读入到一个字符串变量中。

解决方案

为了读取文件的内容，你需要使用文件操作有关的 `open`、`read` 和 `close` 方法。下面的示例代码会把文件的所有内容读取到变量 `s` 中。

```
f = open('test.txt')
s = f.read()
f.close()
```

进一步探讨

此外，对于文本文件来说，你还可以使用 `readline` 方法以每次一行的方式来读取文件。

对于上例来说，如果文件不存在，或由于某种原因而处于不可读状态的话，系统就会抛出一个异常。为了处理这种情况，你可以在代码中加入 `try/except` 结构来处理异常，具体如下所示。

```
try:
    f = open('test.txt')
```

```
s = f.read()
f.close()
except IOError:
    print("Cannot open the file")
```

参考资料

关于文件写操作以及文件的各种打开模式，请参考 7.7 节。

关于异常处理的详细介绍，请参考 7.10 节。

7.9 序列化 (Pickling)

面临问题

你想把一个数据结构的所有内容全部保存到一个文件中，以便在程序下次运行时可以将其读取出来。

解决方案

你可以使用 Python 的序列化功能，按照某种格式将数据结构转存到文件中，并且，这种格式能在今后自动将转存的内容以原来的形式恢复到内存中。

在下面的示例代码中，我们将会把一个复杂的列表结构保存到一个文件中，具体如下所示。

```
>>> import pickle
>>> mylist = ['some text', 123, [4, 5, True]]
>>> f = open('mylist.pickle', 'w')
>>> pickle.dump(mylist, f)
>>> f.close()
```

为了将一个文件的内容反序列化到一个新的列表中，可以使用下列代码。

```
>>> f = open('mylist.pickle')
>>> other_array = pickle.load(f)
>>> f.close()
>>> other_array
['some text', 123, [4, 5, True]]
```

进一步探讨

序列化技术几乎能够完美适用于任何你能够想到的数据结构上面，而非仅限于列表上。被序列化的内容，通常都会以文本的形式保存到文件中，尽管这些文本的可读性非常强，但是通常情况下，你根本无需查看或编辑这些文本文件。

参考资料

关于文件写操作以及文件的各种打开模式，请参考 7.7 节。

7.10 异常处理

面临问题

如果程序运行期间出现某些错误，你不仅希望能够捕获这些错误，并且还能够以对用户更加友好的形式来展示这些错误信息。

解决方案

你可以使用 Python 提供的 try/except 结构。

下面的示例代码取自 7.8 节，它能够捕获文件打开期间的所有问题。

```
try:  
    f = open('test.txt')  
    s = f.read()  
    f.close()  
except IOError:  
    print("Cannot open the file")
```

由于你把可能容易出错的命令都封装到了 try/except 结构中，所以，发生任何错误时，你能够在任何错误信息被显示之前就提前捕获它们，这样的话，你就有机会利用自己的方式来处理它们了。就本例来说，你就可以利用更加友好的消息“Cannot open the file”来通知出错情况了。

进一步探讨

一个常见的运行时异常的情况（文件访问期间除外），是当你访问列表时下标出现越界的时候。例如，如果一个列表只有 3 个元素，当你试图访问第 4 个元素的时候，就会出现这种情形：

```
>>> list = [1, 2, 3]  
>>> list[4]  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: list index out of range
```

错误和异常是以层级结构的形式来组织的，并且你可以根据需要来捕获特殊或一般的异常。

Exception 类非常接近层次树的顶部，并且几乎可以用来捕获所有的异常。此外，你还可以使用单独的 except 段来捕获不同类型的异常，并使用不同的方式进行处理。如

果你不规定任何异常类型的话，就会捕获所有类型的异常。

另外，Python 还允许在错误处理中使用 `else` 和 `finally` 子句。

```
list = [1, 2, 3]
try:
    list[8]
except:
    print("out of range")
else:
    print("in range")
finally:
    print("always do this")
```

如果没有发生异常，`else` 子句就会被执行，但是无论是否发生异常，`finally` 子句都是要被执行的。

每当发生异常时，你都可以通过异常对象获取有关的详细情况，需要注意的是只有使用了 `as` 关键字之后才能如此，具体如下所示。

```
>>> list = [1, 2, 3]
>>> try:
...     list[8]
... except Exception as e:
...     print("out of range")
...     print(e)
...
out of range
list index out of range
>>>
```

这不仅允许你使用自己的方式来处理错误，同时还保留了原始的错误消息。

参考资料

关于 Python 异常类的层次结构的详细内容，你可以参考 Python 的相关文档 (<http://bit.ly/1cqqHV7>)。

7.11 使用模块

面临问题

你希望在自己的程序中使用 Python 的模块。

解决方案

你可以使用 `import` 命令。

```
import random
```

进一步探讨

对于 Python 语言来说，有非常多的模块（有时候称为库）可供选用。并且，大部分的模块都是作为标准程序库的一部分而包含在 Python 中了，至于剩下的其他模块，你仍然可以通过下载来安装到 Python 中。

标准 Python 库中的模块提供了随机数、数据库访问、各种互联网协议、对象序列化等功能。

模块如此之多的一个后果是有可能发生冲突，例如，两个模块是否会包含同名的函数等。

为了避免这些冲突，当导入模块时，你可以规定模块的哪些部分是可以供外部访问的。

例如，如果你只是使用如下所示的命令：

```
import random
```

这样是不会出现任何冲突的，因为你只有在被访问的任何函数或变量前面加上前缀 random 的情况下，才能够访问它们。（顺便提一句，下一节将会介绍 random 包。）

反之，如果你使用了下列命令的话，那么无需使用任何前缀，就可以访问模块内的所有内容。除非你对所用模块内的所有函数了然于胸，否则产生冲突的可能性就会非常大。

```
from random import *
```

在这两种极端情况之间，你还可以选择显式规定自己程序所需的模块中的组件，以便在无需使用前缀的情况下也可以访问它们。

举例来说：

```
>>> from random import randint  
>>> print(randint(1, 6))  
2  
>>>
```

第三种选择是通过关键字 as 为模块提供一个更简洁或更有意义的名称，然后利用这个名称来引用该模块。

```
>>> import random as R  
>>> R.randint(1, 6)
```

参考资料

关于 Python 所提供模块的权威清单，请参考 [http://docs.Python.org/2/library/。](http://docs.Python.org/2/library/)

7.12 随机数

面临问题

你需要在指定范围内生成一个随机数。

解决方案

你可以使用 random 库。

```
>>> import random  
>>> random.randint(1, 6)  
2  
>>> random.randint(1, 6)  
6  
>>> random.randint(1, 6)  
5
```

生成的随机数的大小会介于两个参数之间（其中包括这两个参数）——就本例来说，我们是在模拟掷骰子。

进一步探讨

这里生成的数字并非真正意义上的随机数，而是大家所熟悉的伪随机数序列。也就是说，它们是一个很长的数列，当长度足够大的时候，如果从中取数字的话，就会表现出统计意义上的随机分布。对于游戏来说，这种特性已经足够好了，但是如果你想生成彩票号码的话，则需要求助于专门的定制化硬件了。电脑不擅于随机，因为随机的確不是电脑的天性。

随机数通常用于从列表中随机选择元素。为此，你可以先生成一个位置下标，然后使用这个下标访问元素。此外，random 模块还专门为这提供了一个函数。你可以尝试下列代码。

```
>>> import random  
>>> random.choice(['a', 'b', 'c'])  
'a'  
>>> random.choice(['a', 'b', 'c'])  
'b'  
>>> random.choice(['a', 'b', 'c'])  
'a'
```

参考资料

若要进一步了解 random 包，请访问其官方文档 (<http://docs.python.org/2/library/random.html>)。

7.13 利用 Python 发送 Web 请求

面临问题

你需要使用 Python 将网页内容读取到一个字符串中。

解决方案

Python 提供了一个扩展库，可以用来发送 HTTP 请求。

下面的 Python 2 代码会将 Google 主页内容读取到字符串变量 `contents` 中。

```
import urllib2
contents = urllib2.urlopen("https://www.google.com/").read()
print(contents)
```

进一步探讨

如果你使用的是 Python 3 而非 Python 2 的话，那么你还需要将代码修改为下面的样子。

```
import urllib.request
response = urllib.request.urlopen('http://python.org/')
contents = response.read()
print(contents)
```

在读取 HTML 之后，你很可能想要搜索它，并从中提取真正需要的文本部分。为此，你可以使用相应的字符串处理函数（参见 5.14 节和 5.15 节）。

参考资料

利用 Python 与因特网交互的相关内容，请参考第 15 章。

7.14 Python 的命令行参数

面临问题

你想从命令行运行一个 Python 程序，并向其传递某些参数。

解决方案

你可以导入 `sys` 模块，并使用其 `argv` 属性，具体如下例所示。这样的话，会返回一个数组，数组的第一个元素就是程序的名称。该数组的其他元素，则是命令行中在程序名之后输入的所有参数（需要用空格分隔）。

```
import sys

for i, value in enumerate(sys.argv):
    print("arg: %d %s" % (i, value))
```

从命令行运行该程序，并在后面带上某些参数，结果如下所示。

```
$ python cmd_line.py a b c
arg: 0 cmd_line.py
arg: 1 a
```

```
arg: 2 b  
arg: 3 c
```

进一步探讨

在命令行中指定参数的能力对于在开机期间（见 3.21 节）或在指定时间（见 3.23 节）自动运行 Python 程序来说是非常有用的。

参考资料

关于从命令行运行 Python 的基本知识，请参考 5.4 节。

如果想输出 argv 的话，可以使用列表枚举（见 6.8 节）。

7.15 从 Python 运行 Linux 命令

面临问题

你想从自己的 Python 程序中运行一个 Linux 命令或程序。

解决方案

你可以使用 system 命令。

举例来说，如果想要删除一个名为 myile.txt 的文件（该文件位于 Python 的启动目录中）的话，可以使用下列代码。

```
import os  
os.system("rm myfile.txt")
```

进一步探讨

有时，你不仅要像上例那样“摸黑”执行命令，而且还得捕获命令的响应。比如，假设你想使用“hostname”命令来查看树莓派的 IP 地址（参见 2.2 节）。在这种情况下，你可以利用 subprocess 库中的 check_output 函数。

```
import subprocess  
ip = subprocess.check_output(['hostname', '-I'])
```

这里，变量 ip 将用来存放树莓派的 IP 地址。与 system 不同的是，check_output 要求将命令本身及其所有参数都作为数组的单独元素来提供。

参考资料

关于 OS 库的相关文档，请访问 <http://www.pythonforbeginners.com/OS/pythons-OS-module>。

有关 subprocess 库的详细信息，请访问 <https://docs.python.org/3/library/subprocess.html>。

在 14.4 节中，你将会看到利用 subprocess 库在液晶屏幕上显示树莓派的 IP 地址、主机名和时间的示例代码。

7.16 从 Python 发送电子邮件

面临问题

你想从一个 Python 程序中发送电子邮件。

解决方案

Python 提供了一个支持简单邮件传输协议（SMTP）的库，你可以利用它来发送电子邮件。

```
import smtplib

GMAIL_USER = 'your_name@gmail.com'
GMAIL_PASS = 'your_password'
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

def send_email(recipient, subject, text):
    smtpserver = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
    smtpserver.ehlo()
    smtpserver.starttls()
    smtpserver.ehlo()
    smtpserver.login(GMAIL_USER, GMAIL_PASS)
    header = 'To:' + recipient + '\n' + 'From: ' + GMAIL_USER
    header = header + '\n' + 'Subject:' + subject + '\n'
    msg = header + '\n' + text + '\n\n'
    smtpserver.sendmail(GMAIL_USER, recipient, msg)
    smtpserver.close()

send_email('destination_email_address', 'sub', 'this is text')
```

要想使用上面的代码向指定的地址发送电子邮件，首先要根据自己的电子邮件登录凭证来修改 GMAIL_USER 和 GMAIL_PASS。如果使用的不是 Gmail 的话，那么你还需要修改 SMTP_SERVER 的值，同时 SMTP_PORT 的值也可能需要修改。

此外，你还需要修改最后一行中电子邮件的目的地。

进一步探讨

Send_email 将 smtplib 库的用法简化为单个函数，以便于在自己的项目中重复使用。

从 Python 发送电子邮件的能力为各种项目开启了机会之门。例如，你可以使用 PIR 之类的传感器，以便在侦测到移动发生时发送电子邮件。

参考资料

关于使用 IFTTT Web 服务发送电子邮件的示例代码，请参考 15.3 节。

关于利用树莓派发送 HTTP 请求的内容，请参考 7.13 节。

关于 `smtplib` 的详细介绍，请参考 <http://docs.python.org/2/library/smtplib.html>。

更多与互联网有关的示例代码，请参考第 15 章。

7.17 利用 Python 编写简单 Web 服务器

面临问题

你需要创建一个简单的 Python Web 服务器，但是又不想运行完整的 Web 服务器栈。

解决方案

你可以使用 Python 库 `bottle` 来运行一个纯 Python 的 Web 服务器来响应 HTTP 请求。

为了安装 `bottle`，可以使用下列命令。

```
sudo apt-get install python-bottle
```

下面的 Python 程序（在图书资源中名为 `bottle_test`，资源地址 <http://www.raspberrypicookbook.com>）提供了显示树莓派时间的简单功能。图 7-1 展示了从网络任意位置上的浏览器连接到树莓派时所显示的页面内容。

```
from bottle import route, run, template
from datetime import datetime

@route('/')
def index(name='time'):
    dt = datetime.now()
    time = "{:%Y-%m-%d %H:%M:%S}".format(dt)
    return template('<b>Pi thinks the date/time is: {{t}}</b>', t=time)

run(host='0.0.0.0', port=80)
```

为了运行该程序，你需要作为超级用户来执行它。

```
$ sudo python bottle_test.py
```

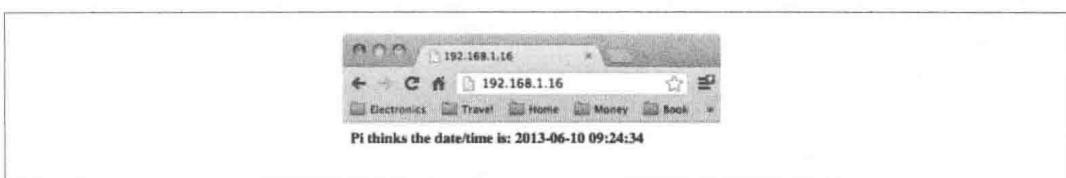


图 7-1 浏览 Python bottle Web 服务器

这个示例代码还需要做进一步的解释。

在 import 命令后，命令@route 会把 URL 路径/与其后的处理函数关联起来。

该处理函数将格式化日期与时间，然后返回供浏览器渲染的 HTML 字符串。就本例来说，它使用了一个模板，该模板随后会被相应的值替换掉。

最后面的 run 所在行的代码才是启动该 Web 服务进程的实际代码。由于 80 端口是 Web 服务的默认端口，所以，如果你希望使用其他端口的话，就需要在服务器地址之后利用：添加相应的端口号。

进一步探讨

只要你喜欢的话，你可以在程序中定义任意数量的路由和处理程序。

对于小而简单的 Web 服务器项目来说，bottle 是理想之选，同时，由于它是用 Python 语言编写的，所以非常易于编写一个处理函数来控制硬件，使其通过浏览器页面作为接口来响应用户的请求。在第 15 章中，你还会看到其他利用 bottle 的示例代码。

参考资料

如果希望进一步了解 bottle 的话，可以参考该项目的相关文档 (<http://bottlepy.org/docs/dev/>)。

关于利用 Python 格式化日期和时间的详细内容，请参考 7.2 节。

更多与互联网有关的示例代码，请参考第 15 章。

7.18 同时进行多件事情

面临问题

当 Python 程序忙于某件事情的时候，你希望还能同时进行其他事情。

解决方案

你可以使用 Python 的 thread 库。

下面的代码将建立一个线程，并且该线程运行时将中断主线程的计算。该示例代码可以从 *Raspberry Pi Cookbook* 网站的下载区获取，网站具体地址为 <http://www.raspberrypicookbook.com/>。

```
import thread, time, random

def annoy(message):
    while True:
        time.sleep(random.randint(1, 3))
```

```
print(message)

thread.start_new_thread(annoy, ('BOO !!!,))

x = 0
while True:
    print(x)
    x += 1
    time.sleep(1)
```

控制台上将输出类似于下面的内容。

```
$ python thread_test.py
0
1
BOO !!
2
BOO !!
3
4
5
BOO !!
6
7
8
```

当你使用 Python 的 `thread` 库让一个新线程运行起来时，必须指定让哪个函数作为该线程来运行。在本例中，这个函数名为 `annoy`，它包含一个循环语句，在一个介于 1 到 3 秒的随机间隔之后打印输出一则消息，并且该循环将一直进行下去。

为了启动该线程，需要调用 `thread` 模块中的 `start_new_thread` 函数。

这个函数需要用到两个参数：第一个参数是需要运行的函数的名称（本例为 `annoy`），第二个参数是一个元组，存放传递给该函数的所有参数（本例为'BOO!!'）。

你会发现，当主线程正在忙着计数的时候，每隔几秒就会被作为 `annoy` 函数运行的线程所打断。

进一步探讨

像上面这些线程，有时候也被称为轻量级进程，因为从效果上看，类似于同时运行多个程序或进程。不过，对于线程来说，其优点在于在同一个程序中运行的多个线程可以访问相同的变量，并且当程序的主线程退出时，在主线程中启动的任何进程也照常如此。

参考资料

有一篇关于 Python 线程入门的好教材，地址为 http://www.tutorialspoint.com/Python/python_multithreading.htm。

7.19 让 Python 无所事事

面临问题

你希望让 Python 消磨一段时间。比如，当向终端发送消息的这段延迟时间内，你可能会想让 Python 消磨时间。

解决方案

你可以使用 time 库中的 sleep 函数，具体如下面的示例代码所示。该代码可以从本书的网站 (<http://www.raspberrypicookbook.com>) 上面下载，程序名称为 sleep_test.py。

```
import time

x = 0
while True:
    print(x)
    time.sleep(1)
    x += 1
```

在打印下一个数字之前，该程序的主循环会延迟几秒时间。

进一步探讨

函数 time.sleep 会以秒数作为其参数，如果你想减少延迟时间的话，可以使用小数。例如，为了推迟 1 毫秒，你可以使用 time.sleep (0.001)。

对于任何持续时间不确定或者只持续零点几秒的循环来说，为其设置一个较小的延迟是个不错的主意，因为当 sleep 被调用的时候，处理器就会被释放出来供其他进程使用。

参考资料

关于 time.sleep 降低 Python 程序的 CPU 负载机制的有趣论述，请参考 <http://raspberrypi.stackexchange.com/questions/8077/how-can-i-lower-the-usage-of-CPU-For-this-Python-program>。

7.20 将 Python 应用于树莓派版 Minecraft

面临问题

你已经对 Python 有了深入了解，现在，你想将其应用于 Minecraft 上面。

解决方案

利用树莓派版 Minecraft 提供的 Python 接口，你可以在该服务器运行期间通过 Python

语言与其进行交互。

虽然可以在 LXTerminal 会话和 Minecraft 游戏之间来回切换，但是每次窗口失去焦点时游戏都会暂停，因此最好还是从另一台计算机上使用 SSH 连接树莓派（见 2.7 节）。这种做法的额外好处是你可以在游戏中实时观察 Python 脚本的运行情况。

下面的示例代码可以在当前位置创建楼梯（见图 7-2），具体代码可以从 *Raspberry Pi Cookbook* 的网站（<http://www.raspberrypicookbook.com/>）的下载区下载。

```
from mcpi import minecraft, block  
mc = minecraft.Minecraft.create()  
  
mc.postToChat("Lets Build a Staircase!")  
  
x, y, z = mc.player.getPos()  
  
for xy in range(1, 50):  
    mc.setBlock(x + xy, y + xy, z, block.STONE)
```

上面的 Python 库是 Raspbian 预安装的，如果你的系统中没有该库的话，可以更新自己的系统（见 1.5 节）。

导入该库之后，变量 mc 将被赋予一个 Minecraft 类的新实例。

方法 postToChat 会向玩家的屏幕发送一则消息，告诉玩家即将建造一个楼梯。

变量 x、y 和 z 将绑定到玩家的位置上，随后，for 循环每次将 x 和 y（y 是高度）的值增加 1 的时候，都会调用 setBlock 方法来创建一级楼梯（见图 7-2）。

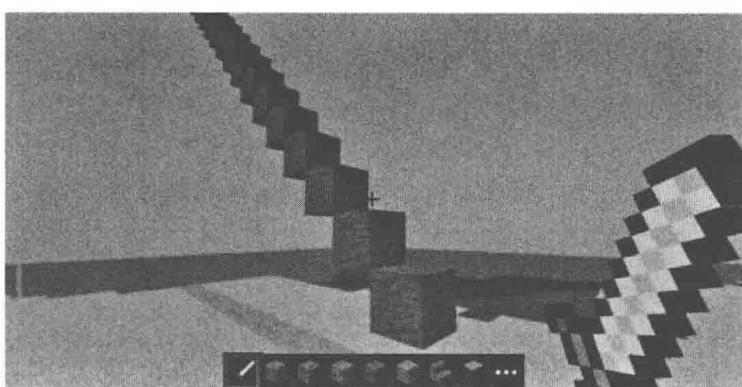


图 7-2 利用 Python 在树莓派版 Minecraft 中建造楼梯

进一步探讨

该 mcpi 库不仅可以用来进行聊天、发现其他玩家位置和摆放建筑零件，而且还提供了

许多其他方法，以便让你可以：

- 发现指定坐标处的建筑组件的 ID；
- 找出谁在玩和瞬移它们；
- 确定玩家位置；
- 确定玩家面向的方向。

对于这个类，尚未有权威的文档，只有一篇比较有用的博客文章（<http://www.stufaboutcode.com/2013/04/minecraft-pi-edition-api-tutorial.html>）。

参考资料

要想了解树莓派版 Minecraft 的更多信息，请参考 4.7 节。

关于在树莓派上面运行 Minecraft 服务器的方法，请参考 4.8 节。

机器视觉

8.0 引言

机器视觉（Computer vision, CV）可以让树莓派睁眼看世界。从实用的角度看，这就意味着你的树莓派能够分析图像，寻找感兴趣的物品，甚至识别面部和文本。

如果你连接一台照相机来提供图像的话，这一切都能成为可能。

8.1 安装 SimpleCV

面临问题

你想在树莓派上面安装机器视觉软件 SimpleCV。

解决方案

为了安装 SimpleCV，首先要使用下列命令来安装必须的软件包。

```
$ sudo apt-get update  
$ sudo apt-get install ipython python-opencv python-scipy  
$ sudo apt-get install python-numpy python-setuptools python-pip  
$ sudo pip install svgwrite
```

然后，利用下列命令来安装 SimpleCV 本身。

```
$ sudo pip install https://github.com/sightmachine/SimpleCV/zipball/master
```

安装完成之后，你可以通过下列命令来检查是否一切正常。

```
$ simplecv  
+-----+  
SimpleCV 1.3.0 [interactive shell] - http://simplecv.org
```

```
+-----+
Commands:
"exit()" or press "Ctrl+ D" to exit the shell
"clear()" to clear the shell screen
"tutorial()" to begin the SimpleCV interactive tutorial
"example()" gives a list of examples you can run
"forums()" will launch a web browser for the help forums
"walkthrough()" will launch a web browser with a walkthrough
```

这将打开 SimpleCV 控制台。它实际上是一个 Python 控制台，只是为 SimpleCV 提供了额外的功能而已。

进一步探讨

SimpleCV 是机器视觉软件 OpenCV 的 Python 封装。SimpleCV，顾名思义，就是 OpenCV 的简化应用。如果你想释放 OpenCV 的全部威力，请浏览 <http://opencv.org/>。

机器视觉需要消耗大量的处理器和内存资源，因此，尽管 SimpleCV 和 OpenCV 可以在老版的树莓派上面运行，但是对于树莓派 3 或 2 之前的版本来说，其运行速度将会慢得让人无法忍受。

参考资料

关于 OpenCV 的详细介绍，请参考 <http://opencv.org/>。

关于 SimpleCV 项目的主页，请访问 <http://simplecv.org>。

在本章中，直到 8.4 节才首次应用 SimpleCV，你可以阅读该节以获取 SimpleCV 入门的详细知识。

8.2 为机器视觉配置 USB 摄像头

面临问题

你想设置一个 USB 摄像头以供机器视觉之用。

解决方案

你可以使用一个与树莓派兼容的 USB 摄像头（参见 http://elinux.org/RPi_USB_Webcams）。请选择一款优质的摄像头，如果你的项目要求摄像头靠近物体的话，请选择可以手动调焦的摄像头。为了能够真正近距离拍摄物体，廉价的 USB 内窥镜将会派上大用场。

有时候你可能希望为自己的 CV 项目建立一个照明良好的区域，当然是否需要要根据自己的 CV 项目的具体情况而定。图 8-1 展示了一个由半透明的塑料储物箱做成的简单灯箱，并且是同时从侧面和顶部均匀给光的。摄像头连接固定在灯箱顶部，这种布局将

会在 8.4 节中用到。

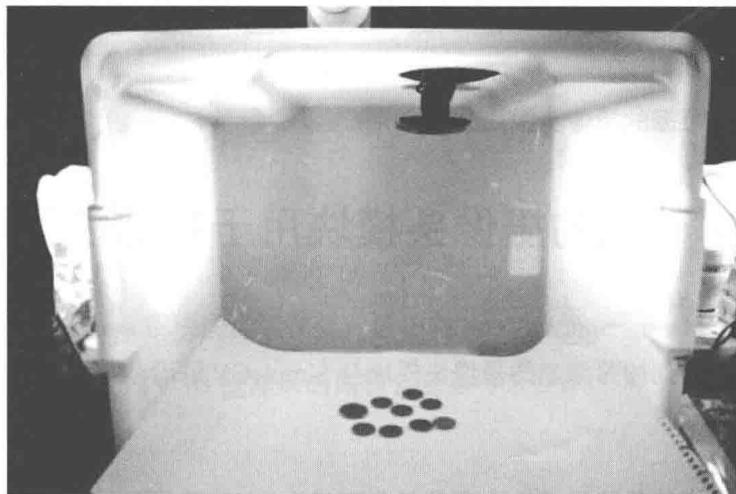


图 8-1 利用自制“摄影灯箱”提供均匀照明

此外，你还可以购买专门用于摄影的商品化的摄影灯箱，它们通常更加好用。

为了使得照明系统明亮而均匀，你可能需要费些时间来反复试验。此外，阴影也会是一个棘手的问题。

进一步探讨

你可以通过 SimpleCV 控制台来测试自己的 USB 摄像头。启动 SimpleCV，然后输入下面粗体显示的命令。

```
SimpleCV:1> c = Camera()
VIDIOC_QUERYMENU: Invalid argument

SimpleCV:2> i = c.getImage()
SimpleCV:3> i
SimpleCV:3: <SimpleCV.Image Object size:(640, 480), filename: (None),
at memory location: (0x2381af8)>
SimpleCV:4> i.show()
```

你不用理会无效的参数消息。

当你执行 `i.show()` 命令时，会打开另外一个窗口，并且会在其中显示摄像头刚才捕获的图像。

虽然树莓派的摄像头模块（见 8.3 节）可以与 SimpleCV 联合使用，但是该模块到树莓派的导线太短了，所以最好还是使用一款高品质的摄像头。

参考资料

要想联合使用树莓派的摄像头模块和 SimpleCV，请参考 8.3 节。

8.3 将树莓派的摄像头模块用于机器视觉

面临问题

你希望将直接连接到树莓派上的摄像头模块与 SimpleCV 联合使用。

解决方案

树莓派的摄像头模块无法自动作为摄像设备而显露出来。为了让该模块与 SimpleCV 协同工作，最简单的方法就是借助 Python 的 picamera 模块，通过该摄像头捕获图像。

下面的代码片段会利用 picamera 来捕获一张图像，并将其保存到一个临时文件中，然后作为适合 SimpleCV 使用的 image 对象进行加载。

```
import picamera
from SimpleCV import *

def get_camera_image():
    with picamera.PiCamera() as camera:
        camera.capture('tmp.jpg')
    return Image('tmp.jpg')
```

在 8.4 节中的程序假定使用 USB 摄像头。该程序的第二个版本会将上面的函数用于树莓派的摄像头模块，具体代码位于文件 coin_count_pi_cam.py 中。

进一步探讨

像这样每当捕获一张图像都要写文件的做法貌似效率不高，事实上也确实如此，不过对于树莓派来说，使用 SimpleCV 对该图像所做的任何处理的耗时，很可能都会比加载和保存图像的时间要长得多。

你可以通过 picamera 模块来设置摄像头的各种功能，最常用的有控制解析度、自动曝光和白平衡。这样做好处在于可以使机器视觉的效果更加易于保持一致性。

摄像头的最佳解析度是 2592×1944 像素，不过这个解析度会严重降低图像处理的速度，所以你可能想要将解析度设置在 1024×768 像素附近，并且关闭自动白平衡。为此，你可以使用 get_camera_image 函数来修改相应的设置，具体代码如下所示。

```
import picamera
```

```
from SimpleCV import *

def get_camera_image():
    with picamera.PiCamera() as camera:
        camera.resolution = (1024, 768)
        camera.awb_mode = 'off'
        camera.capture('tmp.jpg')
    return Image('tmp.jpg')
```

参考资料

关于树莓派摄像头的安装方法，请参考 1.14 节。

关于 Python 的 picamera 模块的详细信息，请访问 <http://picamera.readthedocs.org/>。

关于通过 SimpleCV 使用 USB 摄像头的内容，请参考 8.2 节。

8.4 数硬币

面临问题

你想要利用机器视觉来统计摄像头下面硬币的数目。

解决方案

利用 SimpleCV 及其 findCircle 函数，你可以实时计算放到摄像头下面的硬币的数目。

对于这个机器视觉应用来说，你需要提供良好的灯光照明，以及位置固定的摄像头。我使用的配置如图 8-1 所示。

在开始编写可以计算树莓派看到的硬币数量的程序之前，你得先通过 SimpleCV 控制台来进行必要的实验，从而获得正确识别圆形所需的参数。为此，先利用命令 simplecv 来启动 SimpleCV，并通过下面给出的命令来启动摄像头，捕获图像，然后在单独的窗口中显示出来。

```
SimpleCV:1> c = Camera()
SimpleCV:2> i = c.getImage()
SimpleCV:3> i.show()
```

上面的代码将会显示硬币的图像，具体见图 8-2。

圆检测要求对图像进行反转处理，或者使用黑色背景也行。

你可以通过下面所示的命令来实现图像的反转，并显示结果。

```
SimpleCV:4> i2 = i.invert()
SimpleCV:5> i2.show()
```

上面的命令将建立 i 反转后的副本，具体如图 8-3 所示。

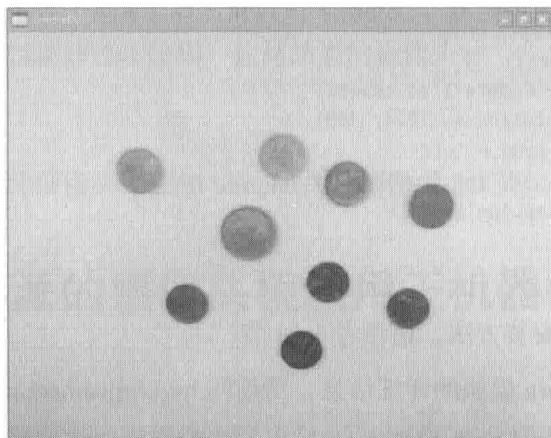


图 8-2 一些硬币的基本图像

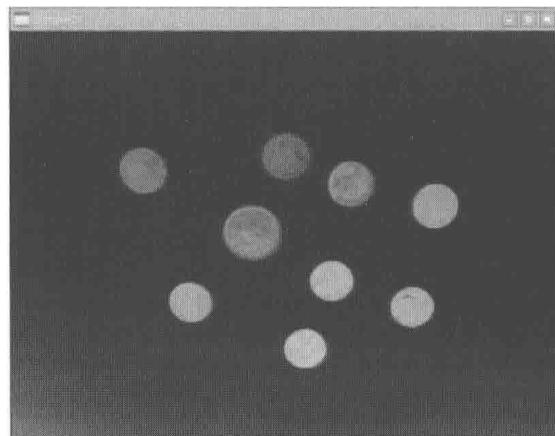


图 8-3 硬币反转后的图像

到目前为止，图像已经准备好了，接下来要做的是利用让 SimpleCV 通过 `findCircle` 命令在图像中寻找圆形。这个命令需要 3 个参数，并且为了防止出现错误辨识的情况，你需要对这些参数进行相应的调整。这些参数分别介绍如下。

canny

这是边缘检测的阀值。按照机器视觉的术语来讲，边缘就是图像像素颜色发生重大变化处的分隔线。这个参数的默认值是 100，并且当降低这个值的时候，会检测到更多的边缘。当然，这不会导致检测到更多的圆形，因为这些额外的边缘存在，有可能会破坏本来很圆的形状。就硬币来说，这些边缘既可以是硬币的文字方面的，也可能是图像方面的。

thresh

找到边缘之后，圆检测需要确定边缘表示圆所需的长度。当降低这个值的时候，会检测到更多的圆形。

distance

这个参数用来设置相邻圆形之间的间隔距离（以像素为单位）。

为了寻找圆形，可以使用下列所示的命令。

```
SimpleCV:6> coins = i2.findCircle(canny=100, thresh=70, distance=15)
SimpleCV:7> coins
SimpleCV.Features.Detection.Circle at (237,297),
SimpleCV.Features.Detection.Circle at (307,323),
SimpleCV.Features.Detection.Circle at (373,305),
SimpleCV.Features.Detection.Circle at (305,261),
SimpleCV.Features.Detection.Circle at (385,253),
SimpleCV.Features.Detection.Circle at (243,231),
SimpleCV.Features.Detection.Circle at (307,383),
SimpleCV.Features.Detection.Circle at (407,371),
SimpleCV.Features.Detection.Circle at (235,373)]
```

如果一个硬币也没找到的话，可以尝试降低参数 canny 和 thresh 的取值。如果找的硬币过多的话，可以增加 thresh 的值。你可以通过下列命令，将硬币外圈叠加到原始图像上面，从而检查 SimpleCV 是否真正找到了这些硬币。

```
SimpleCV:8> coins.draw(width=4)
SimpleCV:9> coins.show()
```

这将显示叠加在实际硬币上的圆圈（见图 8-4）。

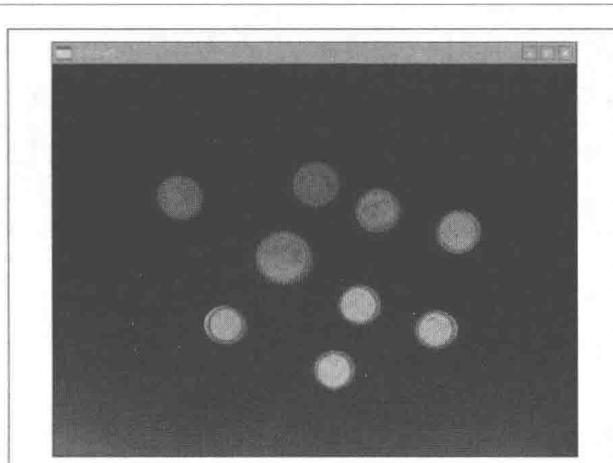


图 8-4 发现硬币

你可以到处移动硬币，或者加减硬币数量，然后重新拍照，重复上述过程，以便确认识别过程的可靠性。此外，你还可以调整各个参数，直到达到令人满意的效果为止。

我们可以把上面在 SimpleCV 控制台中使用的命令打包到一个 Python 程序中去，让它（以树莓派最快的速度）打印出检测到的硬币数量。这个程序可以从本书相关资源中 (<http://www.raspberrypicookbook.com>) 下载，相应的文件名为 coin_count.py。

```
from SimpleCV import *
c = Camera()
while True:
    i = c.getImage().invert()
    coins = i.findCircle(canny=100, thresh=70, distance=1)
    print(len(coins))
```

就像你看到的那样，在导入 SimpleCV 库后，程序中的命令跟在控制台中输入的命令完全一样。唯一的区别在于它不是显示硬币，而是通过 len 函数显示硬币数量。

```
$ sudo python count_coins.py
9
9
9
10
10
```

你可以通过四处移动硬币，或通过添加硬币来检查该项目的工作情况是否良好。

进一步探讨

当我使用 B+ 版本的树莓派的时候，在库加载和相机设置而导致最初的延迟之后，每秒会出现大约“计数”两次。如果使用的是树莓派 2 的话，计数的次数会增加到每秒 5 次左右。

虽然我们没有将这里的东西应用到自动售货机的打算，但是进一步利用硬币的直径来确定币值并加总桌面上的硬币币值本身就是一个非常有趣的项目。

你可以使用 diameter 方法来确定某个硬币的直径，具体代码如下所示。

```
SimpleCV:10> coins[0].diameter()
SimpleCV:11> 60
```

参考资料

关于 SimpleCV 的安装信息，请参考 8.1 节。

关于配置摄像头的相关内容，请参考 8.2 节。

8.5 人脸检测

面临问题

你希望找出人脸在照片或摄像头图像中的坐标位置。

解决方案

你可以使用 SimpleCV 中的 Haar-like 特征检测功能来分析图像，并识别其中的人脸。

如果你还没有做过这类实验的话，请先安装 SimpleCV（参见 8.1 节）。首先，你需要打开 SimpleCV 控制台，并加载一幅含有人脸的图像。实际上，你可以从本书的下载资源中 (<http://www.raspberrypicookbook.com>) 找到合适的人脸图像文件，该文件名为 faces.jpg。

然后，运行下列命令：

```
SimpleCV:1> i=Image("faces.jpg")
SimpleCV:2> faces = i.findHaarFeatures('face.xml', min_neighbors=5)
SimpleCV:3> faces.draw(width=4)
SimpleCV:4> i.show()
```

这样就会打开一个图像浏览窗口，其中的人脸已经使用矩形做了标注，如图 8-5 所示。



图 8-5 检测人脸

进一步探讨

像使用摄像头进行交互一样，你还可以将现成的文件加载到 SimpleCV 中。在上面的例子中，图像 `i` 是从文件 `faces.jpg` 中载入的。方法 `findHaarFeatures` 有一个强制性的文件，它描述待搜索的特征的类型。这些特征被称作 `haar` 特征，并且由一个 XML 文件给出具体的描述。

这些文件是由 SimpleCV 来预加载的，不过你可以通过搜索互联网来了解 `haar` 文件的详细说明。

本例用到的第二个参数（min_neighbors）的作用是调节haar函数，随着min_neighbors数值的降低，假阳性的检测结果就会随之增多。如果观察假阳性结果的话，会发现图片上通常都有面部的正面器官（嘴巴、鼻子和眼睛）。

另外还有许多内置的haar特征，你可以通过下列命令列出它们。

```
SimpleCV:5> i.listHaarFeature()
SimpleCV:4> fullbody.xml', 'face4.xml', 'face.xml',
'upper_body.xml', 'right_ear.xml', 'eye.xml', 'lower_body.xml',
'two_eyes_small.xml', 'nose.xml', 'face2.xml', 'lefteye.xml',
'right_eye.xml', 'two_eyes_big.xml', 'face3.xml', 'mouth.xml',
'glasses.xml', 'profile.xml', 'left_ear.xml', 'left_eye2.xml',
'upper_body2.xml', 'right_eye2.xml', 'face_cv2.xml'
```

就像你所看到的，它们都是与身体部位有关的。

检测haar特征的时候，通常需要几秒时间，即使在树莓派3上面也是如此。

参考资料

关于SimpleCV的安装信息，请参考8.1节。

关于配置摄像头的相关内容，请参考8.2节。

此外，你可以在<https://github.com/Itseez/opencv/tree/master/data/haarcascades>页面找到许多有趣的Haar文件。

8.6 运动检测

面临问题

你想利用连接到树莓派上面的摄像头检测其视野内的移动物体。

解决方案

你可以使用SimpleCV来检测源自摄像头的连续帧之间的变化情况。

下面的程序代码会将每次捕获到的图像与之前的图像进行比较。然后，它会检查差值图像中的所有斑块（颜色相近的区域），如果找到大于MIN_BLOB_SIZE的，就会打印输出一则消息，声明检测到移动现象。

```
from SimpleCV import *
MIN_BLOB_SIZE = 1000
c = Camera()
old_image = c.getImage()
```

```
while True:  
    new_image = c.getImage()  
    diff = new_image - old_image  
    blobs = diff.findBlobs(minsize=MIN_BLOB_SIZE)  
    if blobs :  
        print("Movement detected")  
    old_image = new_image
```

进一步探讨

图像的连续帧大致如图 8-6 和图 8-7 所示。当第二幅图像减去第一幅之后，将得到类似于如图 8-8 所示的图像。然后，对其进行斑块检测（blob detection），将得到如图 8-9 所示的由轮廓线勾勒出的斑块图。

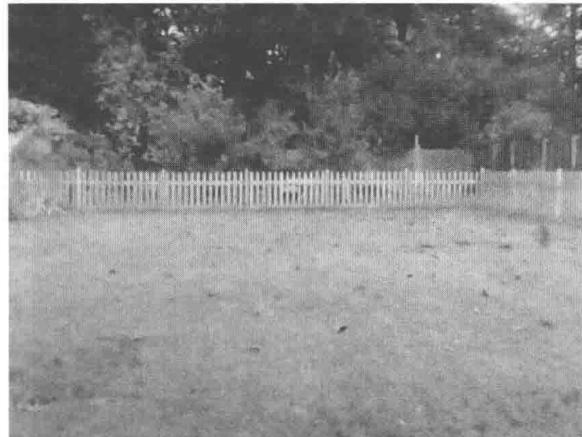


图 8-6 运动检测帧 1



图 8-7 运动检测帧 2



图 8-8 运动检测，差值图像



图 8-9 运动检测斑块

如果使用树莓派 2 的话，上面的运动检测程序每秒可以处理 5 帧。

参考资料

关于 SimpleCV 的安装方法，请参考 8.1 节。关于配置摄像头的相关内容，请参考 8.2 节。

检测运动的另一种方法是使用被动红外（PIR）传感器，具体参考 12.9 节。

8.7 光学字符识别

面临问题

你想把包含文字的图像转换为真正的文本。

解决方案

你可以使用光学字符识别（OCR）软件 tesseract 从图像中提取文本。

要想安装 tesseract，可以使用如下所示的命令。

```
$ sudo apt-get install python-distutils-extra tesseract-ocr tesseract-ocr-eng  
libopencv-dev libtesseract-dev libleptonica-dev python-all-dev swig  
libcv-dev python-opencv python-numpy python-setuptools  
build-essential subversion  
$ sudo apt-get install tesseract-ocr-eng tesseract-ocr-dev libleptonica-dev  
python-all-dev swig libcv-dev  
$ sudo svn checkout  
http://python-tesseract.googlecode.com/svn/python-tesseract-0.7.4/  
$ cd python-tesseract-0.7.4  
$ sudo python setup.py build  
$ sudo python setup.py install
```

为了试用 tesseract 软件，你需要提供一张含有文字的图像文件。为此，你可以从本书提供的下载资源中下载一个名为 ocr_example.png 的文件，地址 <http://www.raspberrypicookbook.com>。

要想把图像转换成文本，可以使用如下所示的代码。

```
$ tesseract ocr_example.png found  
Tesseract Open Source OCR Engine v3.02 with Leptonica  
$ more found.txt  
The quick brown fox jumped over  
the lazy dogs back.  
$
```

进一步探讨

库 tesseract 适用于大部分的图像类型，包括 PDF、PNG 和 JPG 文件。

参考资料

若要进一步了解 tesseract 库，请访问 <https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract>。

硬件基础

9.0 引言

本章将为大家介绍树莓派通用输入/输出（GPIO）接口基本的配置和使用方法。借助这个 GPIO 接口，你就可以把各种感兴趣的电子设备连接到自己的树莓派上面了。

9.1 GPIO 连接器使用说明



请务必在 <http://razzpisampler.oreilly.com> 上观看本节对应的视频。

面临问题

你需要将电子设备连接到该 GPIO 接口上面，但是，在此之前需要先了解各个引脚的作用。

解决方案

实际上，树莓派的 GPIO 连接器已有 3 个版本，其中原始树莓派的两个版本有 26 个引脚，用于“+”型树莓派的版本有 40 个引脚。

图 9-1 展示了 B 型原始树莓派修订版 1 和修订版 2 的 GPIO 的引出线。对于这些修订版本，有一种快速区分的方法，即第一次修订后的声频插座是黑色的，而第二次修订后的声频插座则是蓝色的。

GPIO 接口在第一次修订和第二次修订之间的变化主要有 3 处。这些变化我们已经在图 9-1 中圈出。首先，I2C 端口被换掉了。SDA 和 SCL 两个引脚仍然在原处，但使用了不

同的内部 I2C 接口。这意味着如果将这两个引脚用作 GPIO 而非 I2C 的话，那么在修订版 2 上，你需要把它当成 2 号和 3 号引脚使用。同时，在修订版 2 中，GPIO 的 21 号引脚已经被 27 号引脚取代。在接口的顶部，提供了 3.3V 和 5V 电源，但是 GPIO 的所有输入和输出都使用 3.3V。所有标注了数字的引脚都可以用作 GPIO。在数字之后标注了其他名称的引脚，说明它们还具有其他特殊的用途，例如，14TXD 和 15RXD 分别是串口的发送和接收引脚。SDA 和 SCL 引脚用于 I2C 接口，而 MOSI、MISO 和 SCKL 用于 SPI 接口。

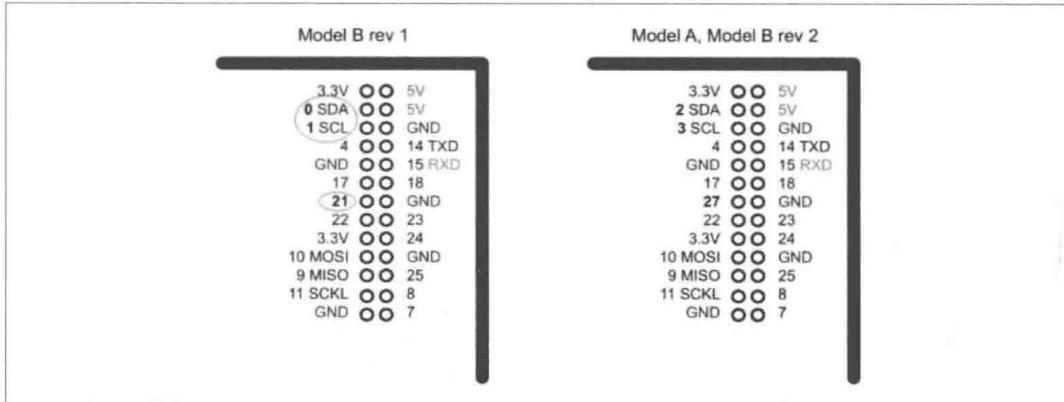


图 9-1 GPIO 引脚 (26 针)

图 9-2 展示了当前 40 个引脚的布局，对于所有带有 40 个引脚的树莓派来说，它们的布局都是相同的。

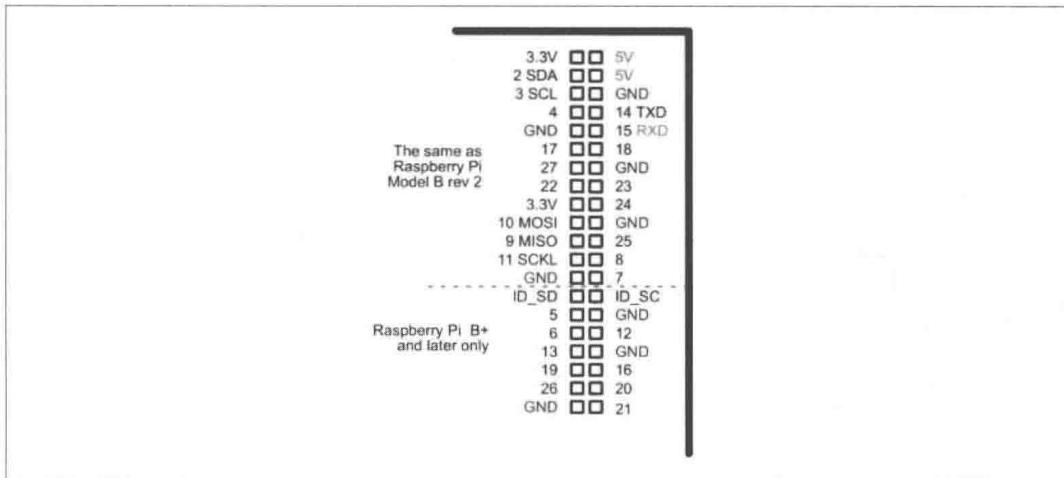


图 9-2 GPIO 引脚 (40 个引脚)

由于顶部的 26 个引脚的布局与第二次修订后的 B 型原始树莓派完全一致，这使得带有 40 个引脚的树莓派能够与之前为 26 引脚树莓派设计的硬件和软件完全兼容，只不过多了 3 个接地和 9 个 GPIO 引脚而已。引脚 ID_SD 和 ID_SC 用于跟特定的串行存储器芯

片进行通信，它们可以放在兼容 HAT（The Hardware At top, HAT）标准的接口板上，从而使树莓派能够识别该电路板（详见下文）。

进一步探讨

即使你知道哪个引脚在树莓派的那个位置，不过在通过计数的方式进行定位的时候，仍然很容易出错。一种更好的引脚定位方法，是使用 GPIO 模板，例如图 9-3 所示的 Raspberry Leaf 模板。

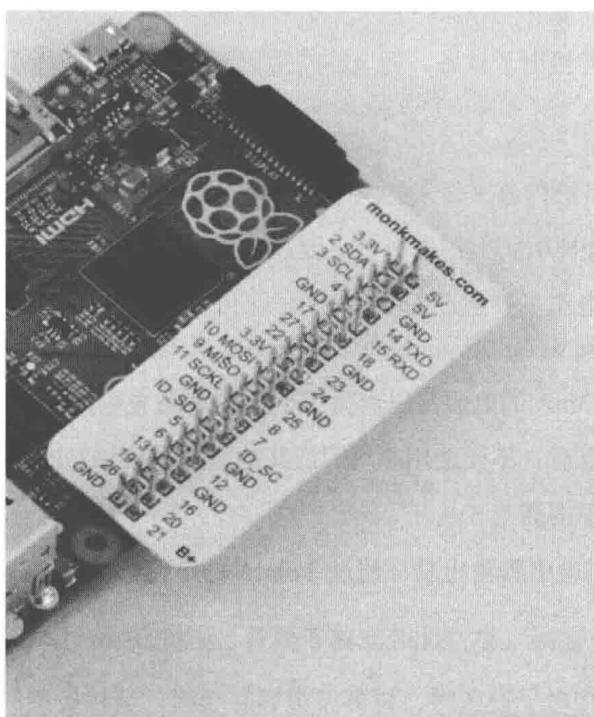


图 9-3 Raspberry Leaf 模板参考资料

有了这种纸质模板，我们对 GPIO 各个引脚就一目了然了。其他的 GPIO 模板还有 Pi GPIO Reference Board。

HAT 标准是树莓派 2、B+ 和 A+ 的一种接口标准。虽然这个标准并不反对直接使用 GPIO 引脚，但是，符合 HAT 标准的接口板可以声称自己为 HAT，它们与常规树莓派的接口板的不同之处在于 HAT 接口板必须带有用以识别 HAT 身份的电可擦可编程只读存储器（EEPROM）芯片，从而使树莓派可以自动安装必要的软件。在本书写作过程中，HAT 还没有达到这种高级的水平，但是这个想法确实不错。引脚 ID_SD 和 ID_SC 被用来跟 HAT 的 EEPROM 进行通信。

树莓派的 GPIO 接口仅提供了数字输入和输出引脚，这一点与其他类似的卡片机不同，因为它们通常都提供了模拟输入。不过，你可以通过使用一个单独的 ADC 芯片（13.5 节）或采用电阻式传感器（13.1 节）来弥补这一不足之处。

关于 HAT 方面的例子，请参考 9.16 节中 Sense HAT 的相关内容。

9.2 使用 GPIO 接口时树莓派的安全保护

面临问题

你希望在树莓派上连接电子设备，但又不希望因意外情况而使树莓派受损。

解决方案

为了降低 GPIO 接口给树莓派带来的风险，请遵循以下简单的规则。

- 施加在 GPIO 任何引脚上的电压都不要超过 3.3V。
- 每个输出上的电流不要超过 3mA。对于早期的 26 引脚树莓派，其总输出电流必须低于 50mA；对于 40 引脚的树莓派，其总输出电流必须低于 100mA。
- 使用 LED 时，3mA 的电流足以点亮一个正确串联了 470Ω 电阻的红色 LED。
- 当给树莓派通电后，不要使用螺丝刀或其他金属物接触 GPIO 接口。
- 树莓派的电源不得高于 5V。
- 不要从 5V 的供电引脚输出总额超过 250mA 的电流。

进一步探讨

毫无疑问，在有外接电子设备时，树莓派很容易受损。虽然新型的树莓派的健壮性有所提高，但是其脆弱性仍然很显著。在树莓派通电前，务必谨慎并进行必要的检查，否则一不小心树莓派就可能被“报销”了。

参考资料

请阅读这篇非常棒的关于树莓派 GPIO 输出能力的文章 (<http://bit.ly/1aMQ0fh>)。

9.3 配置 I2C

面临问题

你希望让一台 I2C 设备与树莓派搭伙，但是不知道具体该怎么做。

解决方案

对于最新版的 Raspbian 来说，只要在 Preferences 下面的主菜单中找到 Raspberry Pi Configuration 工具，就能轻松启用 I2C 设备了，具体如图 9-4 所示。你只需选中相应的 I2C 框，并单击 OK 按钮就行了。这时，系统会提示您重新启动。

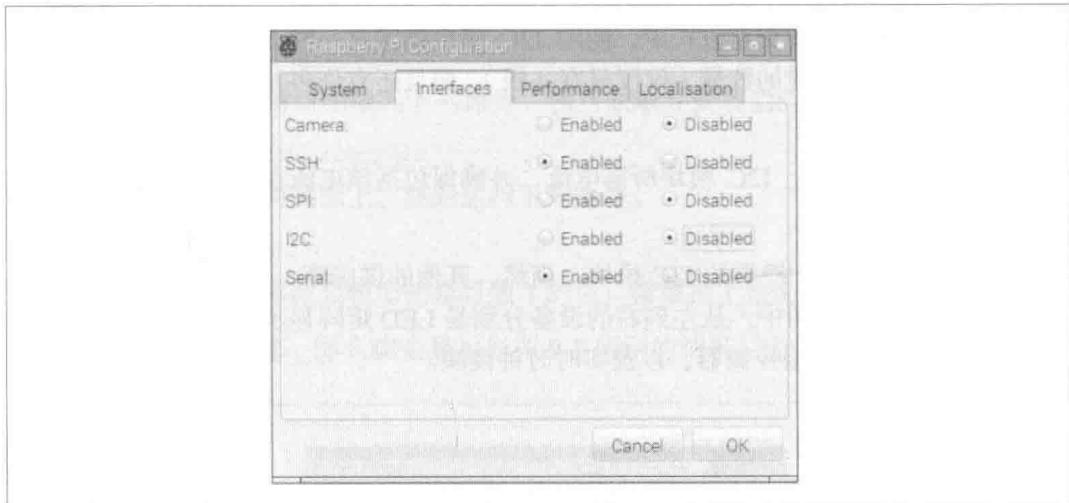


图 9-4 利用树莓派配置工具启用 I2C

在较旧版本的 Raspbian 系统中，可以利用 raspi-config 工具完成上述工作。

打开 raspi-config 工具的命令如下所示。

```
$ sudo raspi-config
```

然后，从菜单中选择 Advanced 选项，并向下滚动到 I2C 项，如图 9-5 所示。

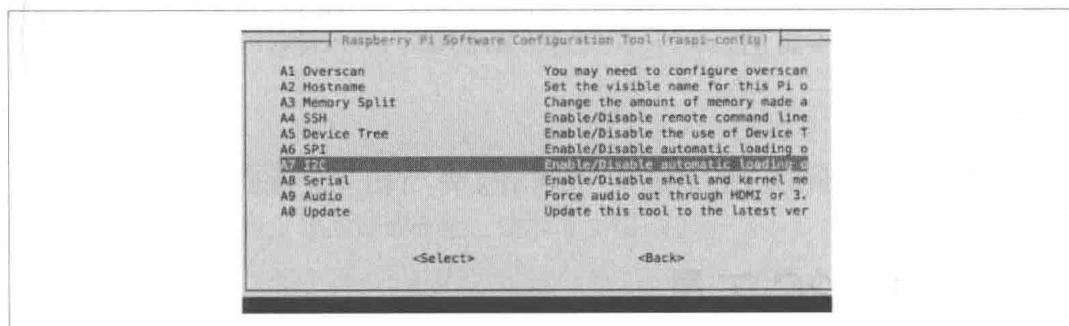


图 9-5 利用工具 raspi-conig 启用 I2C

这时，会询问“Would you like the ARM I2C interface to be enabled? ”，回复“*Yes*”。然后，再次询问是否希望在启动时加载 I2C 模块，继续回复“*Yes*”。

此时，你或许还想安装 Python 的 I2C 库，为此键入下列命令即可。

```
$ sudo apt-get install python-smbus
```

为了使这些修改生效，你需要重新引导树莓派。

进一步探讨

实际上，在与树莓派的交互方式中，使用 I2C 模块确实是一种上上之选。该方式不仅可以减少连接所需线缆的数量（仅仅只有 4 根），而且还有许多非常简洁的 I2C 模块可供选择。

但是，不要忘了统计上 I2C 模块所需电流，并确保包括该电流在内的总电流不得超过 9.2 节中所规定的上限。

图 9-6 展示了 Adafruit 提供的 I2C 模块。当然，其他的供应商，例如 SparkFun 等，也提供 I2C 设备。在该图中，从左到右的设备分别是 LED 矩阵显示、四位七段数码管显示、16 通道 PWM/伺服控制器，以及实时时钟模块。

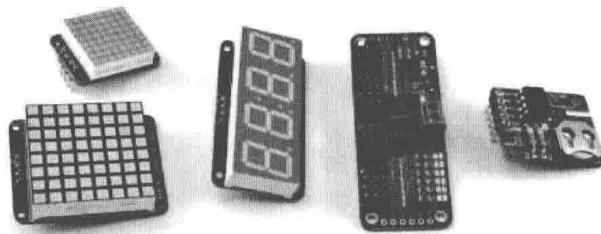


图 9-6 I2C 模块

其他 I2C 模块包括 FM 无线电发射模块、超声波测距模块、OLED 显示模块以及各种类型的传感器。

参考资料

在本书中，还有一些涉及 I2C 的章节，包括 11.3 节、14.1 节、14.2 节和 14.4 节。

9.4 使用 I2C 工具

面临问题

你为树莓派连接了一个 I2C 设备，然后，你希望知道该设备是否连接妥当并获得其 I2C 地址。

解决方案

安装使用 i2c-tools。



注意：在较新版本的发行包中，你会发现 i2c-tools 已经安装好了。

在树莓派的 Terminal 窗口中输入下列命令，便可获取并安装 i2c-tools。

```
$ sudo apt-get install i2c-tools
```

先将 I2C 设备连接到树莓派上，然后运行下列命令。

```
$ sudo i2cdetect -y 1
```

请注意，如果你的树莓派是较旧的修订版 1 的话，需要把上述命令中的 1 改为 0。

如果 I2C 处于可用状态，那么就会显示如图 9-7 所示的结果。这表明使用了两个 I2C 地址，即 0x68 和 0x70。

A screenshot of a terminal window titled "pi@raspberrypi: ~" running on a Raspberry Pi. The window displays the command "pi@raspberrypi: ~\$ sudo i2cdetect -y 1" followed by the output of the i2cdetect command. The output shows two I2C addresses: 0x68 and 0x70. The terminal window has a dark background with white text and a light gray border.

图 9-7 I2C 工具

进一步探讨

I2cDetect 是一个常用诊断工具，尤其是在初次使用一个新 I2C 设备的时候。

参考资料

在本书中，还有一些涉及 I2C 的章节，包括 11.3 节、14.1 节、14.2 节和 14.4 节。

关于利用 apt-get 进行安装的详细介绍，请参考 3.16 节。

9.5 配置 SPI

面临问题

你的树莓派需要用到一个串行外设接口（SPI）总线。

解决方案

默认情况下，Raspbian 的配置并未启用树莓派的 SPI 接口。若要启用该接口，具体步骤与 9.3 节中介绍的基本相似。对于较新版本的 Raspbian，可以使用主菜单下面 Preferences 菜单项中的 Raspberry Pi Configuration 工具；对于较旧版本的 Raspbian，可以使用 raspi-config 工具，具体命令如下所示。

```
$ sudo raspi-config
```

然后，依次选择 Advanced 和 SPI，并且在选择 Yes 之后，树莓派就会重新启动。重启之后，就可以使用 SPI 了。

进一步探讨

启用了 SPI，树莓派便可以与模数转换器（ADC）及端口扩展芯片等外围设备进行串行数据传输了。

你可能会遇到这样一些情况，即虽然连接到 SPI 设备，却并未使用 SPI 接口，而是使用 bit banging 技术，这种情况下通常使用 RPi.GPIO 库来跟用于 SPI 接口的 4 个 GPIO 引脚进行交互。

参考资料

我们在 13.5 节中将会用到 SPI 模数转换芯片。

9.6 安装 PySerial 以便从 Python 访问串口

面临问题

你希望在树莓派上通过 Python 使用串行端口(Rx 和 Tx 引脚)。

解决方案

安装 PySerial 库：

```
$ sudo apt-get install python-serial
```

进一步探讨

库的用法非常简单。首先，通过下列语法建立连接。

```
ser = serial.Serial(DEVICE, BAUD)
```

其中，上面的 DEVICE 表示连接到串口(/dev/ttyAMA0)的设备，而 BAUD 为波特率，注意这里是数字，而非字符串。例如：

```
ser = serial.Serial('/dev/ttyAMA0', 9600)
```

一旦建立连接，你就可以通过以下方式来串行发送数据了。

```
ser.write('some text')
```

通常情况下，响应的监听是由读取和打印构成的循环语句来完成的，例如：

```
while True:  
    print(ser.read())
```

参考资料

在第 12.10 节中，你会用到这里介绍的技术将硬件连接到串口。

9.7 安装 Minicom 以检测串口

面临问题

你希望从终端会话中发送和接收串行命令。

解决方案

安装 Minicom：

```
$ sudo apt-get install minicom
```

安装好 Minicom 之后，你只要键入下列命令，就可以与连接到 GPIO 接口的 RXD 和 TXD 引脚上的串行设备进行串行通信了。

```
$ minicom -b 9600 -o -D /dev/ttyAMA0
```

-b 之后的参数是波特率，-D 后面的参数是串口。请记住，这里的波特率要与通信设备的波特率保持一致。

该命令将开启一个 Minicom 会话。不过，你首先要启用 local echo 功能，因为这样才能看到输入的命令。为此，按下组合键 Ctrl-A，然后再按 Z 键，这时将显示如图 9-8 所示的命令列表，按 E 键启用 local echo 功能。

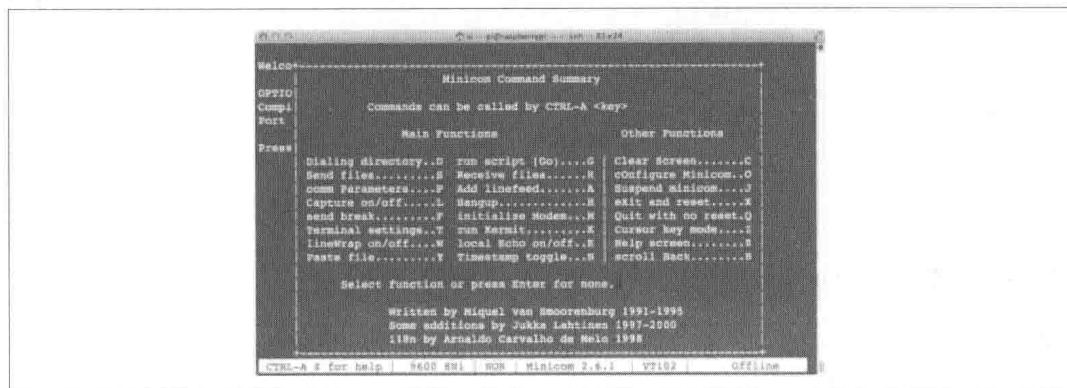


图 9-8 Minicom 命令

此后，你输入的所有内容都会发送到串行设备，同时，任何来自该设备的内容都将显示出来。

进一步探讨

Minicom 是一款非常棒的工具，特别适用于检查来自串行设备的消息，从而确保其运行正常。

参考资料

关于 Minicom 的文档，请参考 <http://linux.die.net/man/1/minicom>。

如果你希望编写处理串行通信的 Python 程序，不妨参考 9.6 节中相应的 Python 串口库。

9.8 使用带有跳线的面包板

面临问题

你希望使用树莓派和免焊面包板制作电子设备原型。

解决方案

使用 male-to-female 跳线和类似 Raspberry Leaf 这样的 GPIO 引脚标签模板（见图 9-9）。

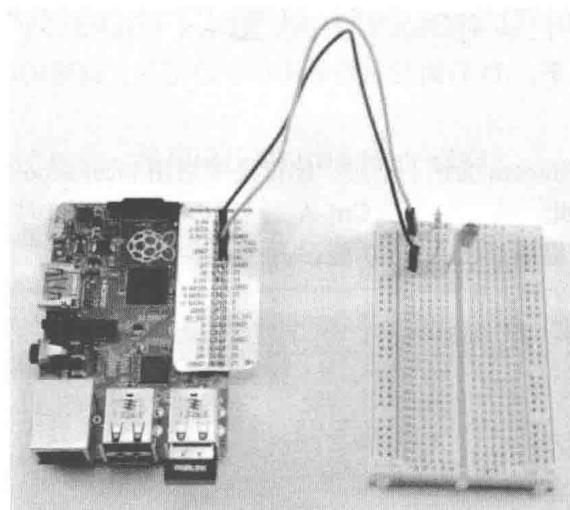


图 9-9 使用 male-to-female 跳线连接树莓派和面包板

进一步探讨

有时候，我们直接在树莓派电路板上识别各个引脚比较费劲，为此，可以打印一个

类似 Raspberry Leaf 这样的模板套在引脚上，这样识别起来就轻松多了。

除此之外，在选择连接面包板不同部分的 male-to-male 跳线时，这种方法也非常奏效。

当其他元件都不需要使用面包板时，可以直接使用 Female-to-Female 跳线将带有公头引脚的模块连到树莓派上面。

一个获取面包板、Raspberry Leaf 和跳线的好办法，就是购买以面包板为主的入门级套件，例如 MonkMakes 的树莓派电子入门套件（见附录 A）。

参考资料

本节介绍的方法在连接数比较小的时候非常管用。但是，在需要建立更多连接时，最好使用树莓派的排线（参见 9.9 节）。

9.9 使用树莓派的排线连接面包板

面临问题

你希望使用树莓派和免焊面包板制作一些电子设备原型。

解决方案

使用 Adafruit 树莓派的排线，这是一种由小型印制电路板（PCB）和可以安装到免焊的面包板上与双列直插（DIL）芯片相似的引脚组成的设备。该 PCB 的所有引脚都提供了标签，同时还提供了一个插槽。提供的带状电缆的作用是连接排线和树莓派（参见图 9-10）。

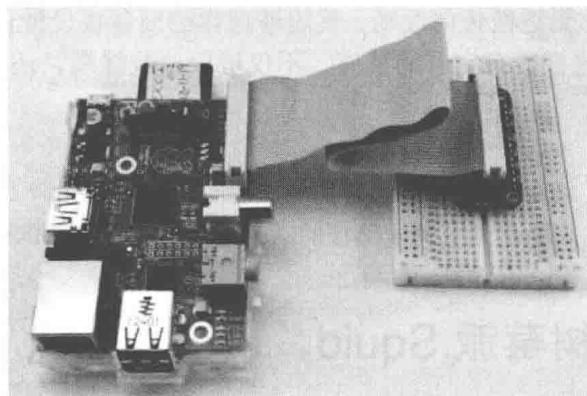


图 9-10 利用排线连接树莓派和面包板

图 9-10 中所示的树莓派的排线是该产品的 26 引脚版本。此外，还有为新型树莓派设计

的 40 引脚版，但是它给其他部件留下的空间较小，所以，如果 GPIO 连接器顶部的 26 个引脚就足够你的项目之用的话，即使对于具有 40 个引脚的树莓派，有时还是使用 26 引脚的排线为好。

进一步探讨

排线的一大优势是可以先将元件在面包板上面组配妥当之后再插入带状电缆。

要确保排线的红色边缘面向 SD 存储卡的边缘，只有这样，才能将线缆插入树莓派的插槽中。

一旦建好面包板原型之后，就可以着手建立该原型的焊接版本了，这时，使用 Adafruit Perma-Proto 板是一个不错的选择，具体如图 9-11 所示。

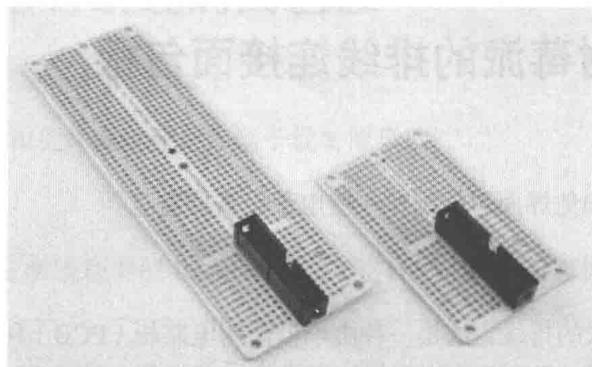


图 9-11 Adafruit Perma-Proto 板

这些电路板都是一些现成的印刷电路板 (PCB)，其连线和板孔的布局与面包板完全一致。这样的话，你无需做任何改动，就能够把你的面包板设计迁移到 Perma-Proto 板上面了。这些电路板带有完善的插座，不仅可以接收线缆，还可以插入树莓派的排线。

参考资料

如果你想制作直接插入树莓派上的电路板的话，请参考 9.19 节和 9.20 节。

9.10 使用树莓派 Squid

面临问题

你想将 RGB LED 连接到树莓派上面，而不必在面包板上面建立任何东西。

解决方案

使用 Raspberry Squid RGB LED（见图 9-12）。

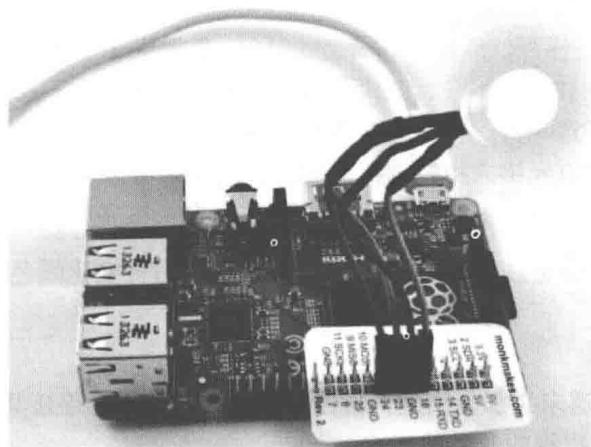


图 9-12 Raspberry Squid

Raspberry Squid 是一种内置了串联电阻和 female 头引线的 RGB LED，因此，可以直接插到树莓派的 GPIO 引脚上面。Squid 的引线都是通过颜色来进行编码的。黑色导线连接 GPIO 的 GND 引脚，红绿蓝导线连接用于红绿蓝颜色通道的相关 GPIO 引脚。红绿蓝输出设备既可以是简单的数字输出设备，也可以是 PWM 输出设备（见 10.3 节），以便混合成不同的颜色。

你可以打造自己的 Squid，具体方法可参考 <https://github.com/simonmonk/squid>，此外，你也可以直接从 <https://www.monkmakes.com/> 购买现成的 Squid。

Squid 有一个 Python 库，安装方法如下所示。

```
$ git clone https://github.com/simonmonk/squid.git  
$ cd squid  
$ sudo python setup.py install
```

在示例文件夹中，有一个名为 test_led.py 的脚本，使用 Raspberry Squid 时所需了解的东西，几乎都可以从这里找到。

```
from squid import *  
import time  
  
rgb = Squid(18, 23, 24)  
rgb.set_color(RED)  
time.sleep(2)  
rgb.set_color(GREEN)  
time.sleep(2)
```

```
rgb.set_color(BLUE)
time.sleep(2)
rgb.set_color(WHITE)
time.sleep(2)
rgb.set_color(WHITE, 300)
time.sleep(2)
```

导入该 squid 库之后，你可以新建一个 Squid 对象，来提供 3 个用于红绿蓝通道的引脚，就本例而言，它们是 18、23 和 24。之后，你就可以通过使用 set_color 方法来设置颜色了，该方法可以将任意一种颜色名称作为其参数，例如 RED、GREEN、WHITE 等。或者，也可以使用三值元组来指定红绿蓝通道，每个值的取值范围介于 0 ~ 100 之间。方法 set_color 的第二个参数是可选的，其作用是指定总体亮度。

在设置好颜色之后，time.sleep(2) 的作用是指定两秒之后再变化颜色。

进一步探讨

Squid 的 3 个颜色通道未必都会用到，当你希望将对应的引脚连接其他电子设备时，只需要看看对应 GPIO 引脚的开关状态即可。

参考资料

关于 Squid 库的详细资料，请参考 <https://github.com/simonmonk/squid>。

关于 Squid 按钮的详细介绍，请参考 9.11 节。

在 10.10 节中，介绍了一个控制 RGB LED（基于 Squid 或者面包板）的示例项目。

9.11 使用 Raspberry Squid 按钮

面临问题

你希望不借助面包板就能够为树莓派连接一个按钮开关。

解决方案

使用 Squid 按钮。

Squid 按钮（见图 9-13）是一种带有 female 头导线的按钮开关，可以直接插到树莓派的 GPIO 连接器上使用。此外，Squid 按钮还带有一个低值电阻器，以便在该按钮不小心连接到数字输出而非数字输入上面时限制电流之用。

Squid 按钮非常适合 Squid RGB LED，并且可以共享 Python 库（请参见 9.10 节介绍的安装方法）。

Squid 按钮也可以直接通过 RPi.GPIO 库来使用，不过 Squid 库提供了按键防抖功能（具

体参见 <https://github.com/simonmonk/squid>) 来防止意外情况下的重复按压。在 Squid 库的示例文件夹中的 test_button.py 文件详细说明了该按钮的使用方法。

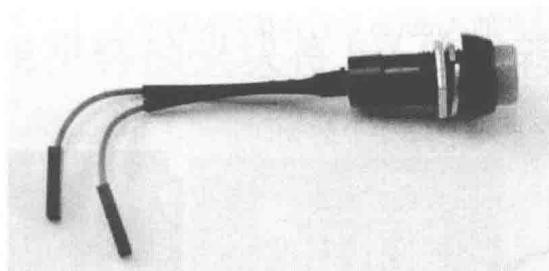


图 9-13 Squid 按钮

```
from button import *
import time

b = Button(7)

while True:
    if b.is_pressed():
        print(time.time())
```

上面代码中的数字（这里是 7）表示该按钮所连 GPIO 引脚的编号。

另一个引脚被连接到 GND。

进一步探讨

Squid 按钮在使用数字输入的测试性项目中非常有用，另外，由于该按钮适于板上装配，所以它还可以用于其他各种永久性项目中。

参考资料

关于 squid 库的详细介绍，请参考 <https://github.com/simonmonk/squid>。关于各种开关的详细信息，请参考 12.1 节、12.2 节、12.3 节、12.4 节、12.5 节和 12.6 节。

至于 Squid RGB LED 的详细介绍，请参考 9.10 节。

9.12 利用两个电阻将 5V 信号转换为 3.3V

面临问题

树莓派的工作电压为 3.3V，但是你想在不损坏树莓派的情况下将某模块的 5V 输出连接到树莓派的 GPIO 引脚上面。

解决方案

将一对电阻器用作分压器来降低输出电压。

图 9-14 展示了如何将 5V 的 Arduino 串行连接到树莓派。

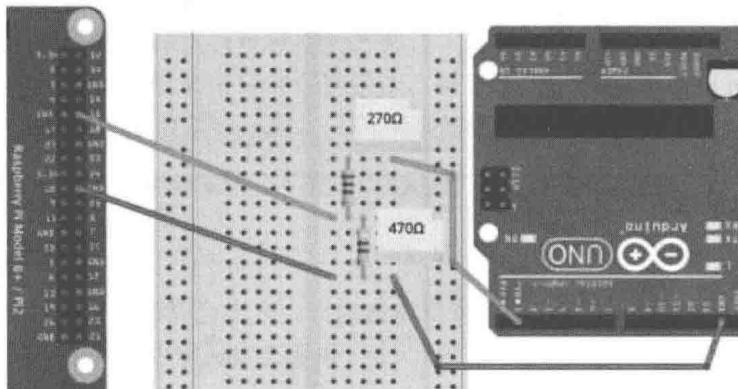


图 9-14 利用电阻将 5V 信号转换为 3.3V

为了完成这个实验，你需要：

- 270 Ω 电阻（附录 A 中的“电阻器和电容器”部分）；
- 470 Ω 电阻（附录 A 中的“电阻器和电容器”部分）。

树莓派的 TXD 信号的输出电压为 3.3V，它可以直接连接到 Arduino 的 5V 输入上，这不会有任何问题。Arduino 模块会把所有高于 2.5V 的电压作为高压看待。

但是，当你需要把 Arduino 模块的 5V 输出连接到树莓派的 RXD 引脚的时候，问题就来了。该输出肯定不能直接连到 RXD 输入上，因为 5V 信号会损坏树莓派。这时，我们可以像图 9-14 所示那样借助于两个电阻来降低电压。

进一步探讨

这里使用的电阻将会消耗掉 6mA 的电流。与树莓派所使用的高达 500mA 的电流相比，这点电流的消耗影响不大。

如果你想将分压器消耗的电流最小化，那么可以使用更大的电阻，按比例缩放，例如 27k Ω 和 47k Ω ，这样的话，就只会消耗 60 μ A 的电流。

参考资料

有关连接树莓派和 Arduino 的详细介绍，请参考第 16 章。

如果你有多个信号需要在 3.3V 和 5V 之间转换，那么最佳选择可能是使用多通道电平转换模块，具体请参考 9.13 节。

9.13 利用电平转换模块将 5V 信号转换为 3.3V

面临问题

树莓派工作电压是 3.3V，但是你想在不损坏树莓派的情况下，将多个 5V 数字引脚连接到树莓派的 GPIO 引脚上面。

解决方案

使用图 9-15 所示的双向电平转换模块。

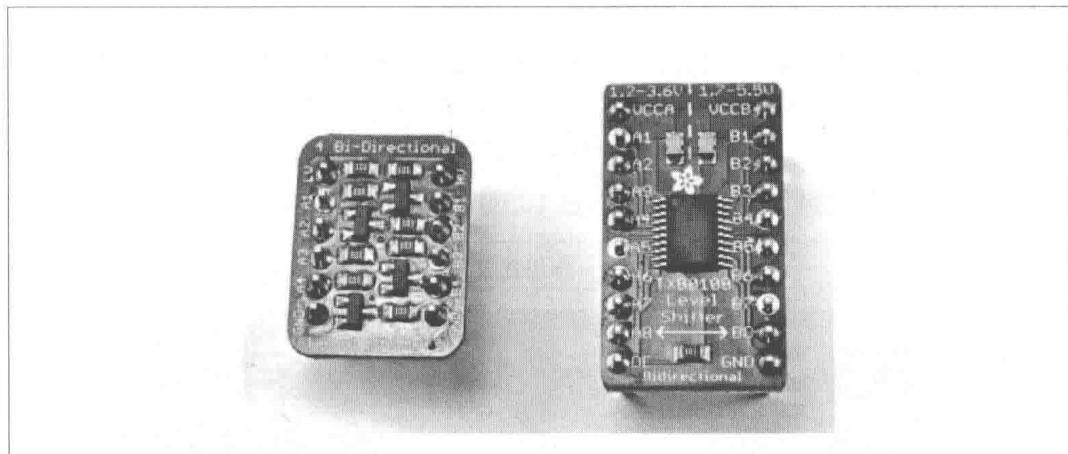


图 9-15 电平转换模块进一步探讨

这些模块非常便于使用。模块的一边有一个特定电压的电源，以及许多工作在此电压下面的输入输出通道。该模块的另一边有许多引脚，其中一个是具有另外一个电压的电源引脚，并且一边的输入输出都会自动转换为另一边的电压。

这类电平转换器包括数量不同的通道，图 9-15 所示的模块分别有 4 个和 8 个通道。

你可以从附录 A 中找到这些电平变换器的来源。

参考资料

请参考 9.12 节，尤其是只有一两个电平需要转换时。

通常情况下，5V 的逻辑输入接收 3.3V 的输出是不会有问题的，但是在某些情况下，例如采用 LED 条（见 14.7 节）时就不行了，在这种情况下，需要使用一个晶体管或上

述中的一个模块来提高逻辑电平。

9.14 利用电池为树莓派供电

面临问题

你想要将树莓派连到机器人上面，并且使用碱性电池为其供电。

解决方案

对于使用树莓派的项目，通常都要求使用 5V 电压、最大 600mA 的电流（参见 1.3 节）。这里，对于 5V 电压的要求是非常严格的。你不应该使用低于或高于 5V 的电源为树莓派供电。现实中，如果想要用高于 5V（例如 9V）的电池给树莓派供电的话，你必须使用调压器将其降低到 5V。

由于树莓派的供电要求相对较高，所以对于小型的 9V 电池，甚至是 7 号电池组来说，都不适合用来给它供电。最适合的电源是由 6 节 5 号充电电池与一个调压器组成的套件。

图 9-16 展示了使用调压器和电池组通过 GPIO 连接器上的 5V 引脚为树莓派供电的装置。

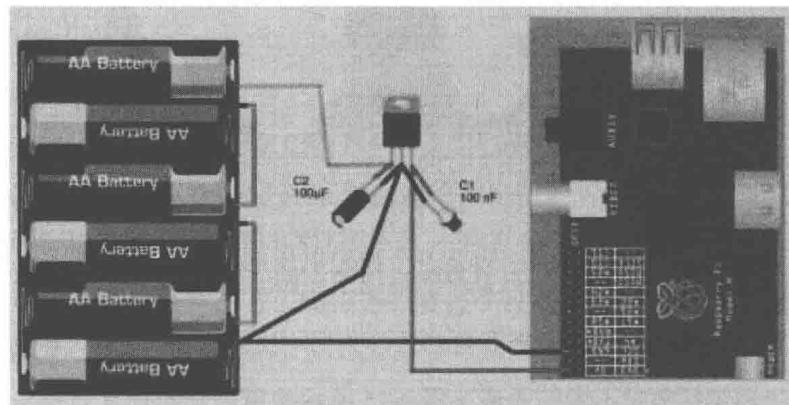


图 9-16 利用 AA 电池为树莓派供电

7805 调压器容易发热，如果它过热的话，其热熔断路器将会发挥作用，致使电压下降，从而可能导致树莓派重启。在集成电路板上面附加一个散热器可以帮助解决这个问题。

另一个解决方案，也是一个不容易发热的方案，是使用 RasPiRobot Board V3（参见 9.18 节），具体如图 9-17 所示。

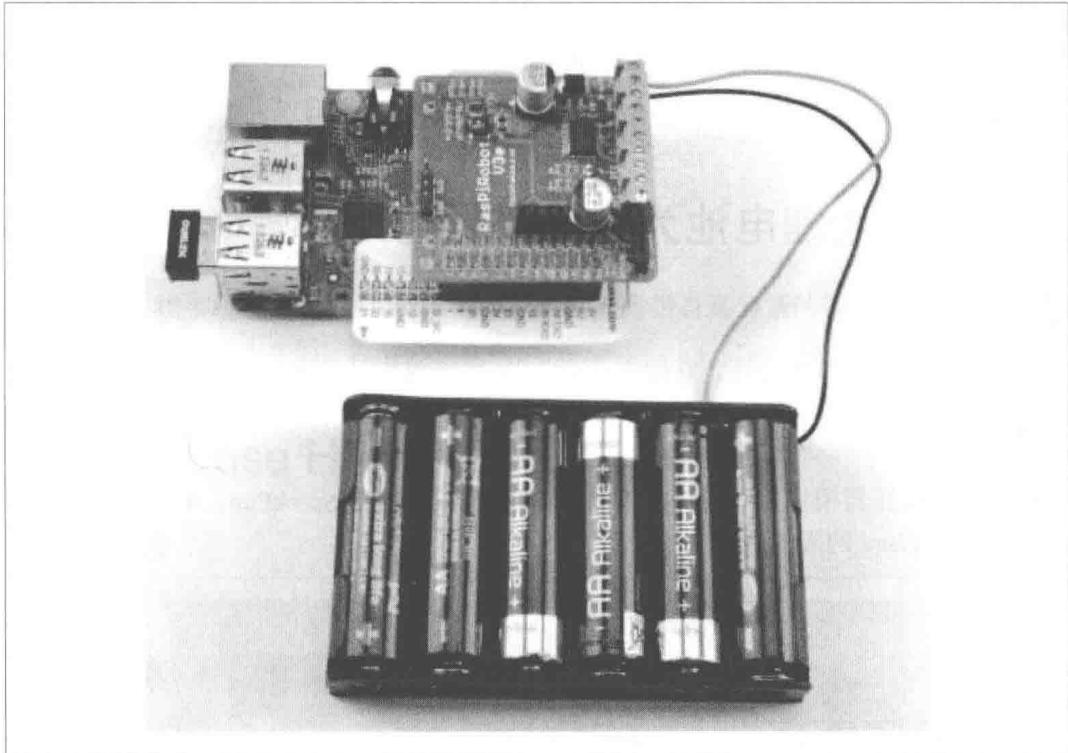


图 9-17 利用 RasPiRobot Board V3 为树莓派供电

作为电机驱动装置，RasPiRobot Board V3 具有一个内置的开关模式电源，可以接收 7V 到 12V 之间的输入电压，同时可以为树莓派提供 5V 电源。由于 RasPiRobot Board V3 是一个高效的开关电源，而非类似 7805 这样的线性调压器，所以正常使用树莓派的情况下，它是不会变得很热的。

进一步探讨

7805 要求的输入电压至少要比 5V 高 2V。你也可以购买一个低压差（LDO）的稳压器，例如 LM2940。LM2940 的输出引脚与 7805 相同，但是输入电压只要求比输出的 5V 高 0.5 即可。但是不要忘了，号称 1.5V 的电池实际上过不了多久就会变成 1.2V 了。所以，由 4 节电池组成的套件撑不了多长时间就会电压不足，最好使用 6 节电池。上面的例子中使用的是一个 9V 的电池组，但是如果你想把树莓派安装到 12V 供电的汽车或房车的话，这个方案仍然奏效。同时，你还需要一个使用直流电（DC）的小显示器。这种设备很容易找到，因为它们经常用于闭路监视系统。

参考资料

另一种利用电池组给树莓派供电的方法是使用一个现成的手机充电器，只要这个充电

器能用电池就行。同时，还要确保这个充电器能够提供 600mA 及更高的电流。

9.15 节展示了如何使用锂电池组为树莓派供电。

至于 RasPiRobot Board V3 的详细用法，请参考 9.18 节。

9.15 利用锂电池为树莓派供电

面临问题

你想将树莓派与机器人连接，并使用一个 3.7V 锂电池为其供电。

解决方案

使用升压型稳压器模块（见图 9-18）。下面展示的模块取自 SparkFun，不过类似设计的产品如果从 eBay 购买的话会更加便宜。

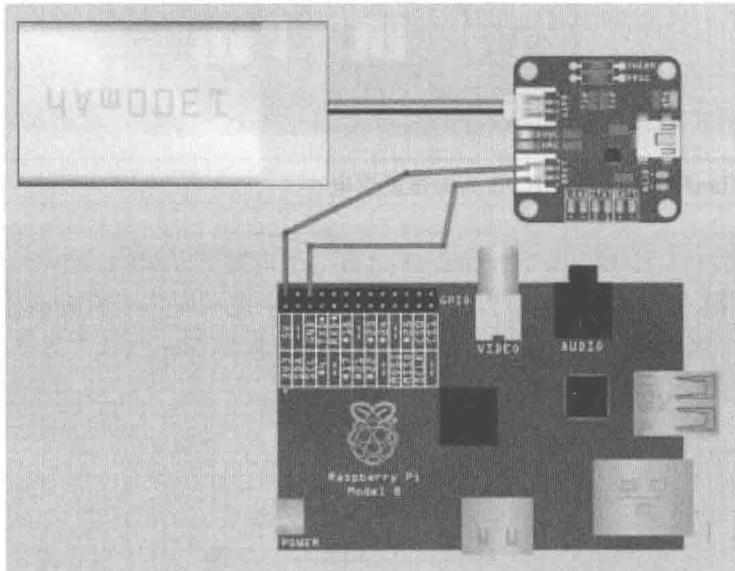


图 9-18 利用 3.7V 锂电池为树莓派供电

但是，只要是从 eBay 便宜采购的物件，通常需要进行全面深入的测试之后才可以投入使用，这个模块也不例外。它们未必总是像广告中宣称的那样，并且质量不一。

这类模块的好处是一方面可以用作给树莓派提供 5V 电压的稳压器，同时还能够为其充电电路提供供给电源的 USB 接口。所以，如果你将树莓派的电源适配器插到该充电器的插座中的话，在给树莓派供电的同时还能够为电池充电，这样你拔下 USB 电源时，只要电池的电量充足，就能继续为树莓派供电。

如果使用一块 1300mA 的锂电池的话，大约能够提供两三个小时的电量。

进一步探讨

如果你打算利用其他方式处理供电问题，可以直接使用一个升压转换模块，其虽然没有充电功能，但是更便宜。

参考资料

你还可以去找一些现成的 USB 锂电池组，它们通常带有高容量的锂电池，为树莓派的供电时间可长达若干小时。

9.16 Sense HAT 入门指南

面临问题

你想了解树莓派 Sense HAT 的使用方法。

解决方案

对于树莓派而言，Sense HAT（见图 9-19）是一个非常有用的接口板，只不过这个电路板的命名多少有点让人摸不着头脑。是的，它上面的确含有传感器，不仅能够测量温度，还能测量相对湿度和大气压力（见 13.10 节）。此外，它还为导航类的项目提供了加速度计、陀螺仪（见 13.13 节）和磁力仪（见 13.14 节）。不仅如此，它还有一个 8x8 全彩 LED 矩阵显示器（见 14.3 节）。

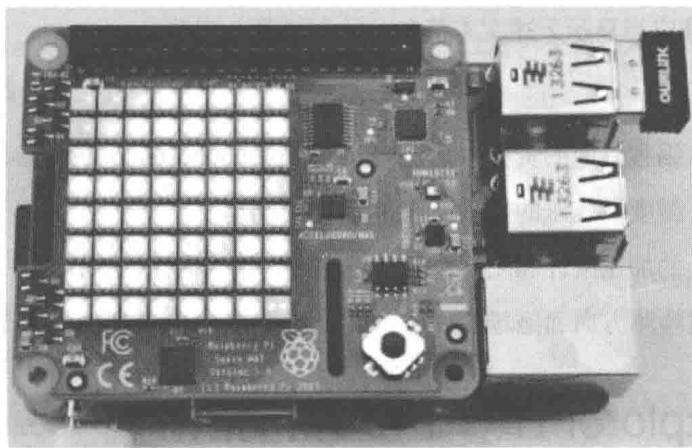


图 9-19 树莓派 Sense HAT

Sense HAT 要求树莓派具有 40 个引脚的 GPIO 接口，因此，接口只有 26 个引脚的旧版

树莓派就无法使用 Sense HAT 了。

在使用 Sense HAT 之前，还需要先安装一些必要的软件。虽然这个过程并不复杂，但比较耗时。

请注意，当阅读到这里的时候，你也许发现 Raspbian 已经安装了 Sense HAT 需要的所有软件。如果是这样的话，运行下面这些命令也不会引起任何问题，因为安装程序会告诉你已无需安装任何东西了。

为了安装 Sense HAT 的 Python 库，请键入下列命令。

```
$ sudo apt-get install sense-hat
```

这个安装程序将会自动启用 I2C，所以你就无需像平常那样来设置 I2C 了（参见 9.3 节）。

由于 Sense HAT 需要用到一个名为 Python Image Library（Python Image Library，PIL）的图形库，所以需要通过下列命令安装该库。

```
$ sudo pip-3.2 install pillow
```

这个安装过程需要一段时间，完成之后，如果 I2C 还未启用的话，请重新启动系统。

进一步探讨

在这本书中，还会有许多地方要用到 Sense HAT，不过就目前来说，只需通过打开一个 Python 控制台来检查它是否工作正常就可以了，具体命令如下所示。

```
$ sudo python
```

然后，在 Python 控制台中输入下列命令。

```
>>> from sense_hat import SenseHat  
>>> hat = SenseHat()  
>>> hat.show_message('The Raspberry Pi Cookbook!')
```

这时，LED 矩阵应当显示上述文本消息，并在屏幕上滚动。

参考资料

关于 Sense HAT 编程方面的资料，请访问 <https://pythonhosted.org/sense-hat/api/>。

关于测量温度、湿度和气压的方法，请参考 13.10 节。

至于 Sense HAT 的加速度计和陀螺仪的使用方法，请参考 13.13 节。

至于利用磁力仪检测方向和磁场的方法，请分别参考 13.14 节和 13.16 节。

9.17 Explorer HAT Pro 入门指南

面临问题

你想了解 Pimoroni Explorer HAT Pro 的入门知识。

解决方案

将 HAT 插到树莓派上面，并安装 Python 库 Explorer HAT Pro。

图 9-20 显示了树莓派 B+ 上面的一个 Pimoroni Explorer HAT Pro。需要注意的是这个 HAT 只能用于具有 40 个引脚的树莓派上面。

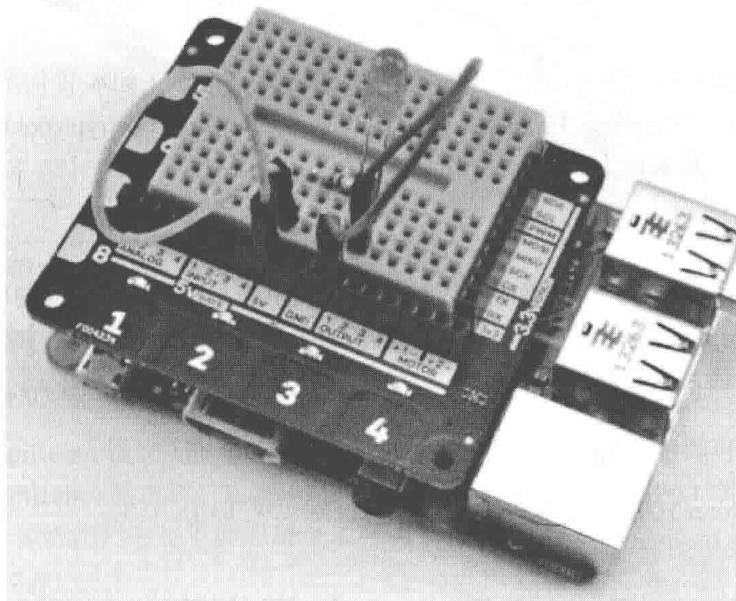


图 9-20 Pimoroni Explorer HAT Pro

Explorer HAT Pro 提供了一些常用的输入输出选件，以及一个可以连接小型面包板的地方，其功能包括：

- 4 个 LED；
- 4 个缓冲输入；
- 4 个缓冲输出（高达 500mA）；
- 4 个模拟输入；
- 2 个低功率马达（最大 200mA）；
- 4 个电容性触垫；
- 4 个电容性鳄鱼夹垫。

若要为 Explorer HAT Pro 安装所需 Python 库，请参考下列命令。

```
$ sudo pip install explorerhat
```

安装好这个程序库之后，你就可以着手进行一项小试验：让其自带的红色 LED 闪烁。

```
import explorerhat, time

while True:
    explorerhat.light.red.on()
    time.sleep(0.5)
    explorerhat.light.red.off()
    time.sleep(0.5)
```

打开一个编辑器（ nano 或 IDLE ），并把下面的代码粘贴进去。正如本书中所有的示例程序一样，你可以从“ Raspberry Pi Cookbook ”网站（ <http://www.raspberrypicookbook.com/> ）上面下载该程序，其名称为 explorer_blink.py 。

进一步探讨

Explorer HAT Pro 提供了 4 个缓冲输入与输出——也就是说，这些输入与输出并非与树莓派直接相连，而是连接到 Explorer HAT Pro 的芯片上面。这就意味着如果在连接的时候意外出错的话，损坏的将是 Explorer HAT Pro ，而非树莓派。

参考资料

你可以将 Explorer HAT 用于电容性触觉感应技术（见 13.18 节）。

9.18 RaspiRobot Board 入门指南

面临问题

你想了解如何使用 RaspiRobot 板。

解决方案

图 9-21 展示的是第 3 版的 RaspiRobot 板。这个电路板带有一个双电机控制器，可用于两个直流电机或一个步进电机。同时，它还可以通过内置的开关式稳压器为树莓派提供 5V 电源。此外，这个电路板还提供了两个开关输入以及两个高电流（ 2A ）输出，并且能够轻松连接 HC-SR04 测距仪和树莓派的 I2C 接口。

第三版的 RaspiRobot 板有自己的 Python 库，你可以通过下列命令来下载并安装该 Python 库。

```
$ git clone https://github.com/simonmonk/raspirobotboard3.git
$ cd raspirobotboard3/python
$ sudo python setup.py install
```

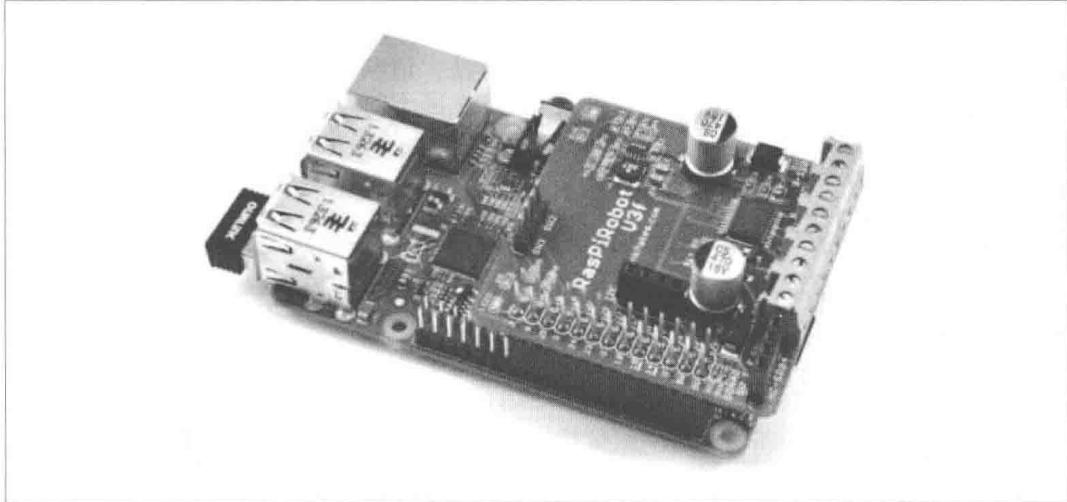


图 9-21 RaspiRobot 板 V3

进一步探讨

将 RasPiRobot V3 安装到树莓派上，然后给树莓派加电。

在尝试与 RaspiRobot 板有关的命令的时候，只需使用 Python 控制台即可，无需连接外部电源或 RaspiRobot 板电机。RaspiRobot 板的代码库使用的是 GPIO 端口，因此，你需要作为一个超级用户通过下面的命令来启动 Python。

```
$ sudo python
```

在第一次加电时，你会发现 RaspiRobot 板上面的两个 LED 都会变亮。接下来，你可以输入下列命令，当这个程序库初始化之后，两个 LED 就会熄灭。

```
>>> from raspirobotboard import *
>>> rr = RRB3()
```

LED 的打开和关闭可以通过下列命令来完成。

```
>>> rr.set_led1(1)
>>> rr.set_led1(0)
```

RaspiRobot 板有两对引脚，它们是用来连接开关的。这两对引脚在 RaspiRobot 板上面分别标为 SW1 和 SW2。你可以通过下面的命令来测试 SW1 的开关状态。

```
>>> print(rr.sw1_closed())
False
```

通过螺丝刀连接 SW1 的两个引脚使其短路，然后再次运行上面的命令，就会返回 True。

此外，还提供了设置两个集电极开路输出（set_oc1 和 set_oc2）和电机控制命令（forward、reverse、left、right 和 stop）的相关命令。

完整的命令介绍资料，请参考 <http://www.raspirobot.com>。

参考资料

关于如何使用这块电路板来制作运动机器人的方法，请参考 11.10 节。

若要使用 RaspiRobot 板控制双极步进电机的话，请参见 11.9 节。

9.19 使用 Pi Plate 原型板

面临问题

你希望了解如何使用 Pi Plate 原型板。

解决方案

Pi Plate（见图 9-22）的类型与 RaspiRobot 板（见 9.18 节）不同，前者是一种原型板，而后者属于接口板。换句话说，它没有提供任何电子元件——因为它的设计初衷是让你在原型区域焊接自己的元件。

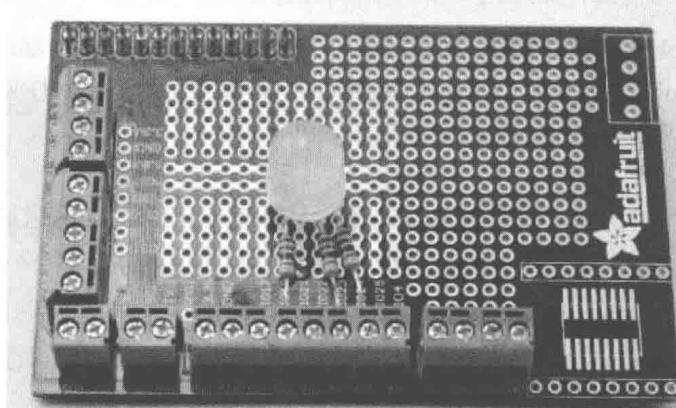


图 9-22 Pi Plate

这个电路板提供了一个区域，可供具有 16 个引脚的表面贴装芯片来使用，另外还有 4 个排成一行的板孔，由于间距更大，所以即使焊接螺旋式接线端子也能容得下。

这块电路板的两端提供有已经连接到所有 GPIO 引脚的螺旋式接线端子。你可以忽略整个原型区域，只使用螺旋式接线端子来将导线连接到 GPIO 引脚上面。

进一步探讨

该电路板提供了间距为标准 0.1 英寸的板孔网格，这个间距适用于绝大部分穿孔组件，

包括 DIL IC。你可以通过将穿孔组件的引线从上端插入孔中，然后在底端焊接的方式来连接各种组件。

连接板孔的线路在该电路板的上面清晰可见，同时还可以看到这个电路板被分成多个区域，其中包括预留给 DIL IC 的带有中央电源母线的区域、通用原型区域和为表面贴装芯片及额外螺旋式接线端子准备的区域。

将各个元件焊接好之后，你还需要额外的导线来连接它们。我们既可以在该电路板的上面连接它们，也可以在下面连接，如果设计复杂的话，还可以两面同时连接。

无论如何，最后在焊接之前就规划好各个器件之间的布局。

在下文中，你将根据图 9-23 所示的布局，将一个 RGB LED 安装到 Pi Plate 上面。这里是 10.10 节的一个翻版，唯一不同在于后者的设计是建立在免焊的面包板的基础之上的。

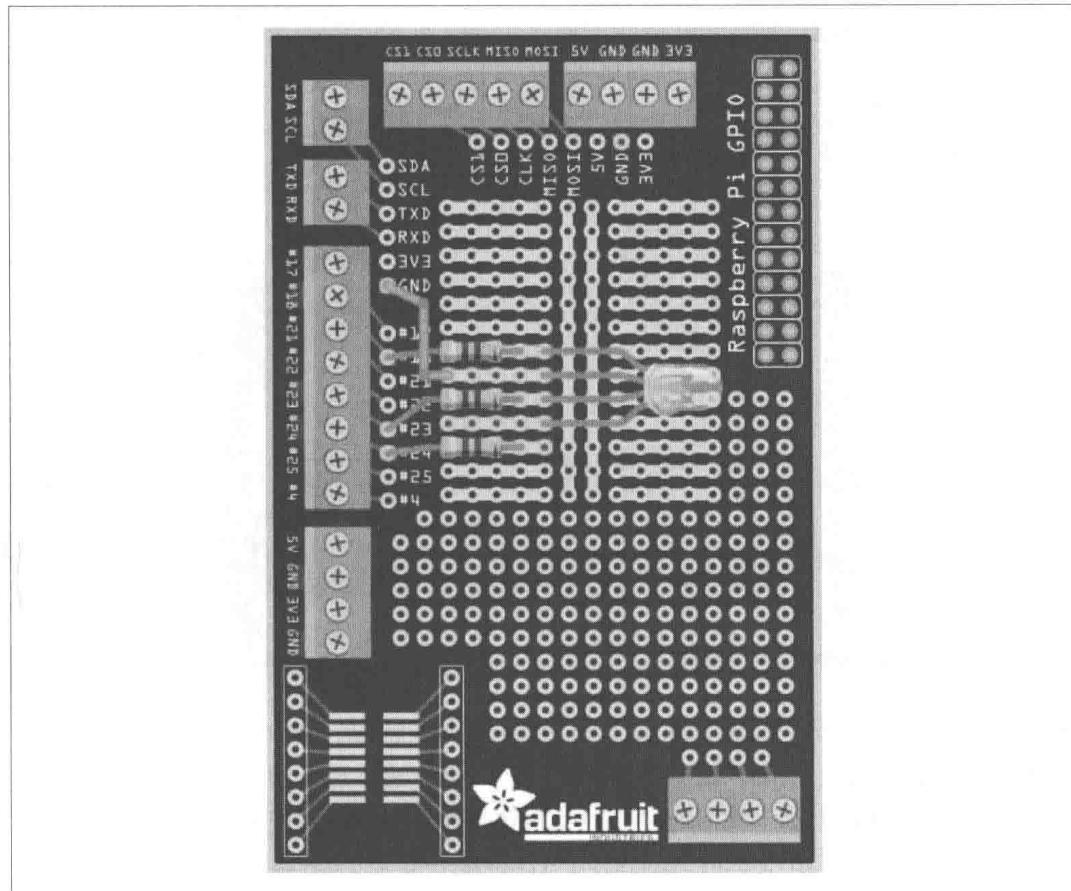


图 9-23 Pi Plate 上面的 RGB LED 的版面布局

首先，我们要焊接上电阻。将引线弯好，然后插入电路板适当的板孔中。然后，把电路板翻过来，用铬铁压住引线露出的板孔之外的部分，停留一会，直到焊锡淌满导线

周围为止（见图 9-24）。

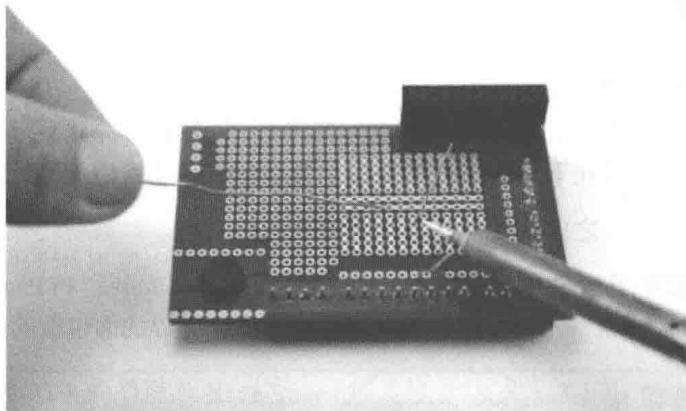


图 9-24 在 Pi Plate

上焊接电阻焊接好两端之后，剪去多余导线，然后继续以同样的方法焊接上另外两个电阻（见图 9-25）。

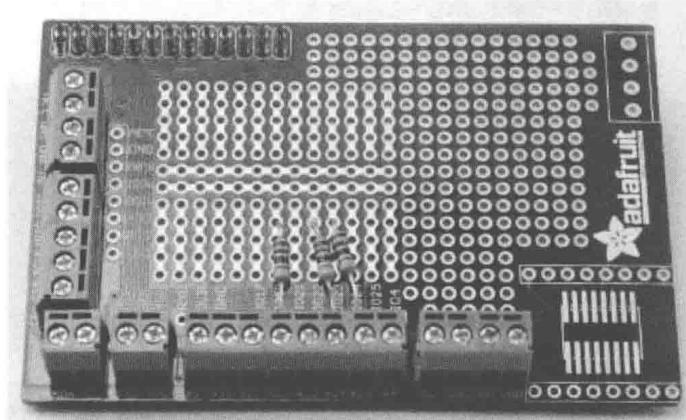


图 9-25 已焊接到 Pi Plate 上面的电阻

接下来，要焊接的是 LED，请务必使用正确的焊接方法。最长的导线是公用的阴极，同时，它也是 LED 上唯一无需连接任何电阻就直连到线接板孔中的引线。有时候，你会看到 LED 的长引线并非正极引线，但是这种情况非常少见。这通常发生在红外线 LED 上面。所以，如果你拿不准的话，请参考 LED 的数据表，或访问供应商的信息页面。

你还需要像图 9-26 所示那样，在 Pi Plate 上将一段较短的导线从相应行焊接到 GND 上面。

当焊接好电路板之后，它应当如图 9-27 所示。

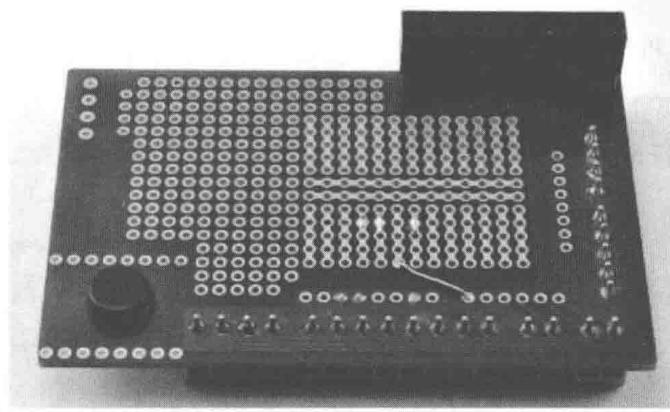


图 9-26 焊接连接 Pi Plate 的导线

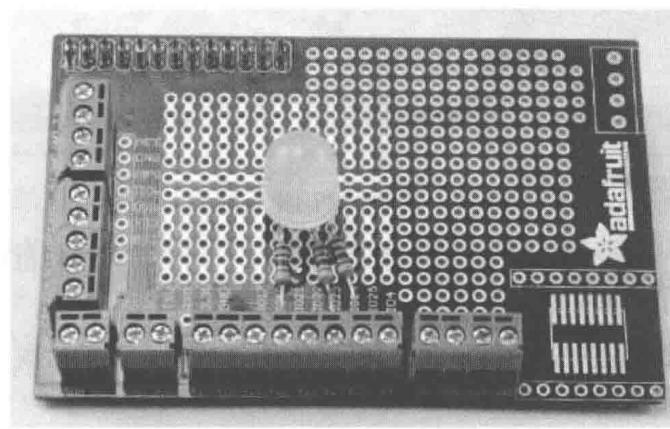


图 9-27 Pi Plate 上焊接好的 RGB LED

你可以利用 10.10 节提供的 Python 程序来试一试该 LED 是否工作正常。

参考资料

对于这个产品来说，Adafruit 站点上面有更加详细的介绍，具体请参考 <https://www.adafruit.com/products/801> 页面。

9.20 制作树莓派扩展板(Hardware At Top , HAT)

面临问题

你想要制作一款符合 HAT 标准的树莓派接口板原型。

解决方案

使用 Perma-Proto Pi HAT (见图 9-28)。

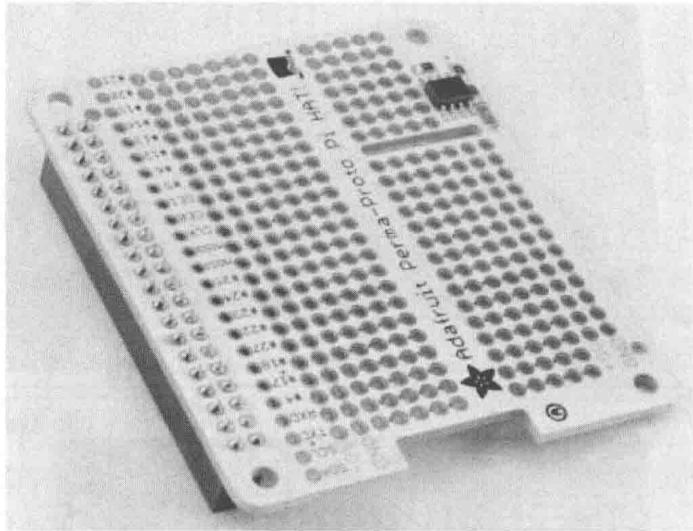


图 9-28 Perma-Proto Pi HAT

随着提供了 40 个引脚的 GPIO 接口的树莓派的到来，为树莓派的规范附加电路板的新标准也应运而生了，该标准名为 HAT (Hardware At Top, HAT)。

当然，你不必死板地遵守这个标准，尤其是在制作一次性产品的时候，但是，如果你想设计一款用于销售的产品的话，那么你最好遵守 HAT 标准。

HAT 标准不仅定义了 PCB 的大小和形状，同时还要求 PCB 要焊接有电可擦除只读存储器 (EEPROM) 芯片。这个芯片被连接到 GPIO 接头的 id_sd 和 id_sc 引脚，并且未来将允许该芯片对树莓派进行某些配置，甚至在利用附加的 HAT 启动树莓派的时候自动加载软件。

该板的原型区域是由分布在板子两侧的电源轨和两排插孔组成，其中这些插孔的布局与面包板类似，每 5 个孔为一组。

如果你不打算对 EEPROM 进行编程的话，那么你就可以到此为止了。但是，如果你想为 HAT 的 EEPROM 添加自定义的信息的话，那么请继续阅读下面的讨论部分。

进一步探讨

虽然 HAT 是一个意义深远的标准，不过，直到写作本书时为止，Raspbian 仍未使用任何已写入到 EEPROM 中的信息。这种情况将来会有所改观，届时将带来激动人心的事

情，即 HAT 可以自动处理很多事情，比如只要检测到树莓派就启用 I2C 并根据硬件安装合适的 Python 库。

要想把数据写入 EEPROM，你首先得启用处于隐匿状态的 I2C 端口，这就要用到 ID_SD 和 ID_SC 引脚，这两个引脚可以用来读写 EEPROM。为此，你需要编辑/boot/conig.txt 文件，加入下面一行内容。如果该文件中早已存在该行内容，但是被注释起来的话，只要解除注释即可。

```
dtoverlay=i2c-vc-on
```

完成上述任务之后，重启树莓派，这时就能够通过相应的 I2C 工具（见 9.4 节）检测到连接到 I2C 总线上面的 I2C EEPROM 了。

```
$ i2cdetect -y 0
 0 1 2 3 4 5 6 7 8 9 a b c d e f
00: --- - - - - - - - - - - - -
10: -- - - - - - - - - - - -
20: -- - - - - - - - - - - -
30: -- - - - - - - - - - - -
40: -- - - - - - - - - - - -
50: 50 - - - - - - - - - - - -
60: -- - - - - - - - - - - -
70: -- - - - - - - - - - - -
```

从 i2cdetect 命令结果中你可以看到该 EEPROM 的 I2C 地址为 50。注意，这里使用的选项是 -y 0，而不是 -y 1，因为引脚 2 和 3 上面的不是常规 I2C 总线，而是该 HAT EEPROM 特有的 I2C 总线。为了读写 EEPROM，你需要下载相应的工具，具体命令如下所示。

```
$ git clone https://github.com/raspberrypi/hats.git
$ cd hats/eepromutils
$ make
```

写 EEPROM 需要 3 个步骤。首先，你必须编辑文件 EEPROM_SETTINGS.TXT。至少需要把 product_id、product_version、vendor 和 product 字段改为自己公司的名称和产品名称。注意，这个文件中还有许多其他的选项，它们自身都带有很好的注释说明。这些选项的功能包括指定后备电源、所使用的 GPIO 引脚等。

然后，编辑该文件，并通过下面的命令将这个文件从文本格式转换为适合写入 EEPROM (ROM_ile.eep) 的格式。

```
$ ./eeprommake eeprom_settings.txt rom_file.eep
Opening file eeprom_settings.txt for read
UUID=7aa8b587-9c11-4177-bf14-00e601c5025e
Done reading
Writing out...
Done.
```

最后，把 rom_ile.eep 复制到 EEPROM 上面，具体命令如下所示。

```
sudo ./eepflash.sh -w -f=rom_file.eep -t=24c32
This will disable the camera so you will need to REBOOT after this...
This will attempt to write to i2c address 0x50. Make sure there is...
This script comes with ABSOLUTELY no warranty. Continue only if you...
Do you wish to continue? (yes/no): yes
Writing...
0+1 records in
0+1 records out
127 bytes (127 B) copied, 2.52071 s, 0.1 kB/s
Done.
pi@raspberrypi ~/$
```

一旦写入完成，你就可以使用下面的命令读取 ROM。

```
$ sudo ./eepflash.sh -r -f=read_back.eep -t=24c32
$ ./eepdump read_back.eep read_back.txt
$ more read_back.txt
```

参考资料

你可以从 <https://github.com/raspberrypi/hats> 页面找到树莓派 HAT 的设计指南。

目前，市场上已经有许多现成的 HAT，其中包括步进电机（见 11.8 节）、电容式传感器触摸式传感器（见 13.18 节）和来自 Adafruit 的 16 通道 PWM（见 11.1 节），以及 Pimoroni Explorer HAT Pro（见 9.16 节）。

9.21 树莓派的计算模块

面临问题

你想利用树莓派来创建一款产品，但是又不想包含一个完整的树莓派在里面。

解决方案

使用树莓派的计算模块。

树莓派的计算模块，即图 9-29 中的左边的元件，右边是其 IO 板，板载了树莓派的 SoC 处理器和 512MB 内存，由于采用了 SODIMM 内存条的接口和造型，所以完全可以在更新笔记本内存时顺便把它插进去。

该硬件基于原始的树莓派，不过没有采用原来的 SD 卡槽，而是直接在电路板上面提供了 4GB 闪存模块。

该电路板的设计思想是产品开发人员可以设计自己带有 SODIMM 插座的 PCB，从而将树莓派的计算模块插入即可。与这个计算模块配套的 IO 板，非常适合使用该计算板来构建你的设计原型。

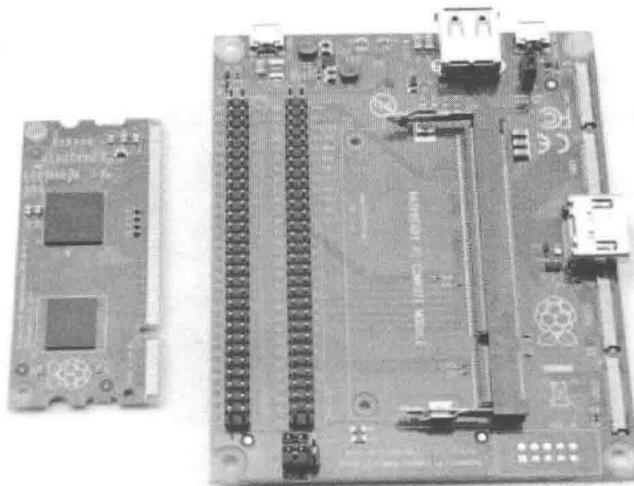


图 9-29 树莓派的计算模块及其 IO 板

进一步探讨

这个电路板先于 Raspberry Pi Zero(见 9.22 节) 发行, 不过到了写作本书这个时点来看, 它好像有点多余, 因为它不仅比 Raspberry Pi Zero 贵很多, 而且性能也不是很好, 只是尺寸略微小了一点而已。此外, 它还缺少 Pi Zero 提供的各种接口, 但是, 当你开发的产品不想定制自己的 PCB 的时候, 这些接口可能正是你所需要的。

尽管该计算模块已经为几个为数不多的产品所采用, 但是, 除非硬件换代或者价格大幅下调, 否则该计算模块的将来会有很大的不确定性。

参考资料

关于树莓派计算模块的完整信息, 请参阅 <https://www.raspberrypi.org/blog/raspberry-pi-compute-module-new-product/>。

9.22 Pi Zero

面临问题

你想了解更多的 Pi Zero 知识, 以及将其用于电子项目的方式。

解决方案

由于 Pi Zero 非常小巧, 并且价格极低, 这些使得它非常适合嵌入到各种电子项目之中。

图 9-30 展示了一款 Raspberry Pi Zero。

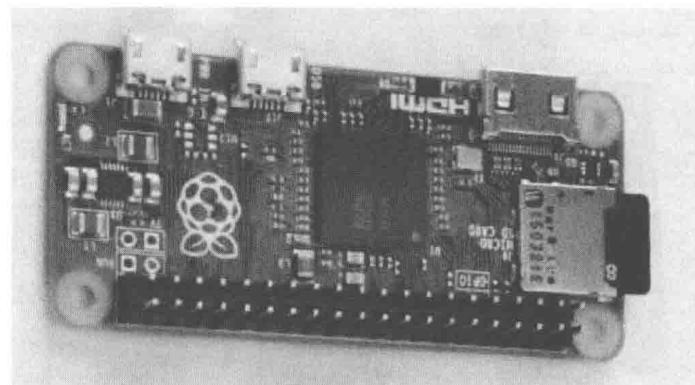


图 9-30 Raspberry Pi Zero

由于 Pi Zero 没有提供接口引脚，所以你的首要任务很可能就是给它焊上引脚。相应的引脚可以从 Pi Zero 的各种入门套件中找到，例如 Pi Hut。

此外，你还可以使用容易买到的单行排阵，只要并排焊接两个就可以了。

进一步探讨

由于 Pi Zero 只有一个 USB 接口和一个微型 USB OTG (on the go, OTG) 接口，所以要想对 Pi Zero 进行配置的话，你还需要一个 USB 适配器和 USB 集线器，因为只有这样才能插入无线 USB 适配器、键盘和鼠标。

此外，你可以使用 2.6 节介绍的控制台线，通过编辑 2.5 节介绍的/etc/network/interfaces 来配置 Wi-Fi。一旦配置好 Wi-Fi，你就可以通过 SSH (见 2.7 节) 让 Pi Zero 无线上网了。

参考资料

对于各种树莓派之间的比较，请参考 1.1 节。

控制硬件

10.0 引言

在本章中，你将学习如何通过树莓派的 GPIO 接口来控制电子设备。

本章的大部分示例都会用到免焊面包板以及 male-to-female 和 male-to-male 跳线(见 9.8 节)。为了与老版的 26 引脚的树莓派保持兼容，这里所有的面包板示例都仅仅使用了其顶部的 26 个引脚，因为这些引脚可以通用于两种 GPIO 布局 (见 9.1 节)。

10.1 连接 LED



请务必在 <http://razzpisampler.oreilly.com> 上观看本节相关的视频。

面临问题

你希望了解将 LED 连接到树莓派的方法。

解决方案

你可以将 LED 连接到一个 GPIO 引脚上面，不过需要利用一个 470Ω 或 $1k\Omega$ 的串联电阻来限制电流。为了进行这项实验，你需要：

- 面包板和跳线；
- 470Ω 电阻；

- LED。

图 10-1 展示了如何利用免焊面包板和 male-to-female 跳线给 LED 布线的方法。

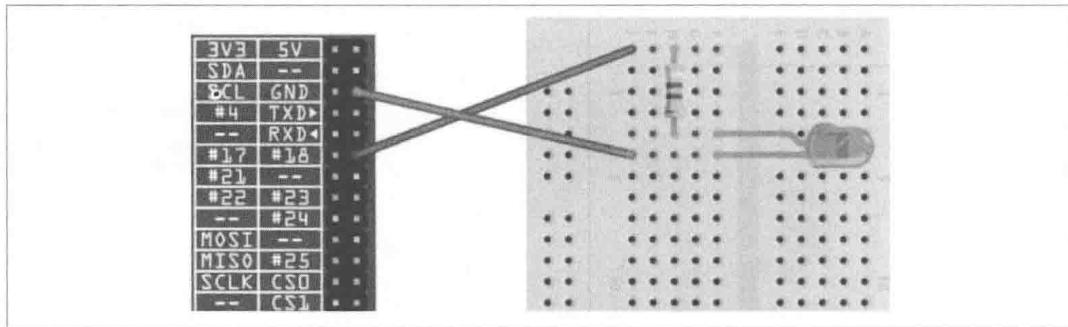


图 10-1 将 LED 连接到树莓派上面

连接好 LED 之后，我们需要能够从 Python 发送命令来控制其开关状态。

为此，你需要从控制台以超级用户身份登录 Python 控制台，并输入下列命令。

```
$ sudo python
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(18, GPIO.OUT)
>>> GPIO.output(18, True)
>>> GPIO.output(18, False)
```

这些命令是用来打开和关闭 LED 的。

进一步探讨

LED 是一种常用、廉价和高效的光源，但是其使用方式需要格外注意。如果你将其直接连接到高于 1.7V 的电压电源（如 GPIO 输出）上面的话，会有一个非常大的电流通过它。这个电流通常大到足以损毁 LED，甚至供电设备——如果供电设备是树莓派的话，情况就更不妙了。

通常情况下，你应该始终为 LED 配备一个串联电阻，因为介于 LED 和电压电源之间的串联电阻能够将流经该 LED 的电流限制在特定的数值上，从而对 LED 和为其供电的 GPIO 引脚提供保护。

树莓派的 GPIO 引脚只能提供 3mA 或 16mA 左右的电流（取决于电路板及使用的引脚的数量），具体参考 9.2 节。对于 LED 来说，通常只要电流大于 1mA 就足以驱动它们发光，只不过电流越大，亮度会越高。当你根据 LED 的类型来选择串联电阻的时候，可以参考表 10-1，同时，该表也给出了从 GPIO 引脚给出的电流的近似值。

表 10-1 为 LED 和 3.3V 的 GPIO 引脚选择串联电阻

LED 类型	电阻	电流 (mA)
红色	470Ω	3.5
红色	1kΩ	1.5

续表

LED 类型	电阻	电流 (mA)
橘色、黄色和绿色	470Ω	2
橘色、黄色和绿色	1kΩ	1
蓝色、白色	100Ω	3
蓝色、白色	270Ω	1

如你所见，在任何情况下，使用 470Ω 的电阻都是安全的。如果你使用的是蓝色或白色 LED 的话，即使电阻值大为降低，也不会危及树莓派的安全。

如果你想把利用 Python 控制台进行的实验进一步扩展为通过程序代码令 LED 闪烁的话，为此可以把下面的代码粘贴到 IDLE (见 5.2 节) 或 nano (见 3.6 节) 编辑器中，并将文件保存为 led_blink.py。此外，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的下载资源中找到这个程序。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

while (True):
    GPIO.output(18, True)
    time.sleep(0.5)
    GPIO.output(18, False)
    time.sleep(0.5)
```

有一点要记清楚，运行该程序时，由于 RPi.GPIO 库要求具有超级用户权限，所以，必须使用下列命令。

```
$ sudo python led_blink.py
```

参考资料

在 <http://led.linear1.org/1led.wiz> 上面，有一个方便的串联电阻计算器。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

10.2 让 GPIO 引脚进入安全状态

面临问题

你希望每当程序退出后，将所有 GPIO 引脚都设置为输入端，以便降低由于 GPIO 接头短路而损坏树莓派的机会。

解决方案

为此，你可以使用 try: finally: 结构和 GPIO.cleanup 方法。

下面的代码源自 10.1 节，但是为了安全退出而对原来的代码进行了重写，具体如下所示。

存放该代码的文件名为 led_blink_safe.py。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

try:
    while (True):
        GPIO.output(18, True)
        time.sleep(0.5)
        GPIO.output(18, False)
        time.sleep(0.5)
finally:
    print("Cleaning Up!")
    GPIO.cleanup()
```

这样一来，当你利用 Ctrl-C 组合键关闭它时，这个程序会在退出之前调用 GPIO.cleanup 方法。

进一步探讨

如果没有调用 cleanup 方法或者树莓派没有重启的话，那么程序结束运行之后，被设置为输出端的引脚仍然会保持输出状态。当你为新项目连接导线的时候，如果没有注意到这个问题的话，可能会将一个 GPIO 输出连接到一个不恰当的电源导轨或 GPIO 引进而意外短路。

发生这种短路的典型情况是当你连接按钮开关的时候，将已经配置为输出端的一个 GPIO 引脚和 HIGH 引脚连接到了 GND 引脚上面，就像 12.1 节中介绍的那样。

总而言之，为安全起见，你要么在调换硬件时倍加小心，要么像前面那样使用 GPIO.cleanup 方法，要么重新启动树莓派。无论如何，在断电的情况下为树莓派连接新硬件永远不失为上上之策。

参考资料

关于 Python 的异常处理内容，请参考 7.10 节。

10.3 控制 LED 的亮度

面临问题

你希望利用 Python 程序来改变 LED 的亮度。

解决方案

RPi.GPIO 库提供了一个脉冲宽度调制（PWM）功能，可用来控制 LED 的功率和亮度。

要想使用该功能，需要按照 10.2 节中的说明连接 LED，然后运行下面的测试程序（`led_brightness.py`）。

```
import RPi.GPIO as GPIO

led_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

pwm_led = GPIO.PWM(led_pin, 500)
pwm_led.start(100)

while True:
    duty_s = raw_input("Enter Brightness (0 to 100):")
    duty = int(duty_s)
    pwm_led.ChangeDutyCycle(duty)
```

当你使用的是 Python 3 而非 Python 2 的时候，需要将其中的 `raw_input` 改为 `input`。

通过运行这个 Python 程序，你就可以输入一个介于 0 到 100 之间的数字来改变亮度了。

```
pi@raspberrypi ~ $ sudo python led_brightness.py
Enter Brightness (0 to 100):0
Enter Brightness (0 to 100):20
Enter Brightness (0 to 100):10
Enter Brightness (0 to 100):5
Enter Brightness (0 to 100):1
Enter Brightness (0 to 100):90
```

为了退出该程序，只需要使用组合键 `Ctrl-C`。

进一步探讨

PWM 是一种非常高明的技术，它使你可以在保持每秒总体脉冲数（以 Hz 为单位）不变的情况下改变脉冲的长度。图 10-2 展示了 PWM 的基本原理。

在高频情况下，测定的 PWM 频率会与作为参数提供的频率略有出入。当然，这种情况有可能在推出新版的 RPi.GPIO PWM 功能之后有所改善。

你可以通过修改下面的代码来改变 PWM 的频率。

```
pwm_led = GPIO.PWM(led_pin, 500)
```

由于上面的数值是以 Hz 为单位的，所以就本例而言，其频率为 500Hz。

表 10-2 对 GPIO.PWM 的第二个参数规定的频率与利用示波器在引脚上实测的频率进行了相应的比较。

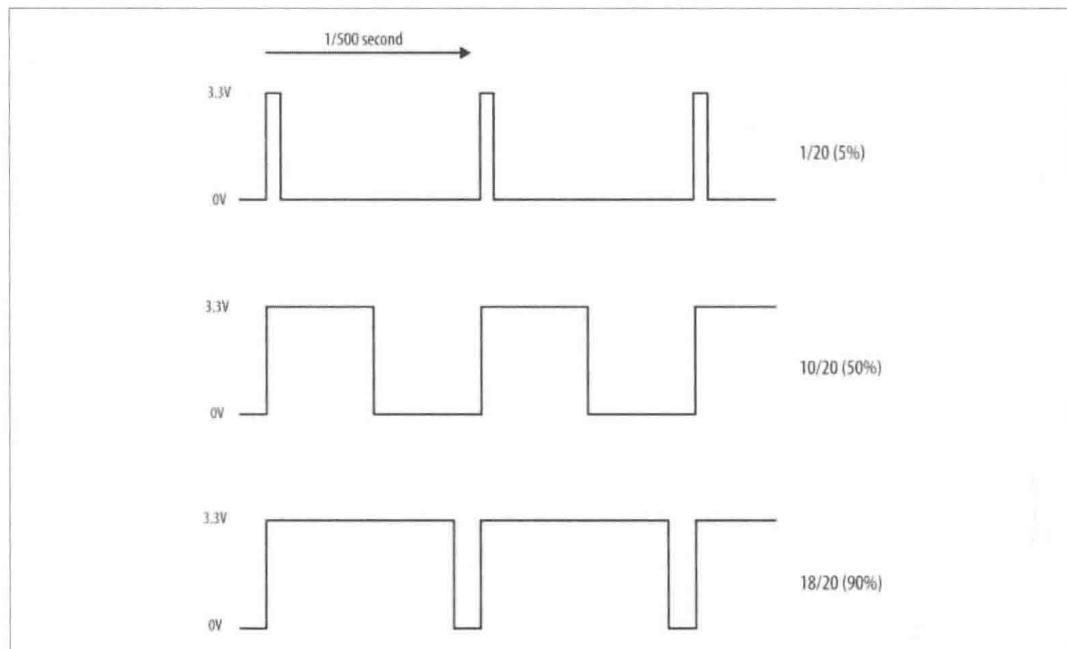


图 10-2 脉冲宽度调制

表 10-2 要求的频率与实际的频率

要求的频率	测量到的频率
50 Hz	50 Hz
100 Hz	98.7 Hz
200 Hz	195 Hz
500 Hz	470 Hz
1 kHz	890 Hz
10 kHz	4.4 kHz

此外，我还发现随着频率的上升，其稳定性会逐渐降低。这就意味着这个 PWM 功能无法适用于音频，但是对于控制 LED 亮度或电机转速来说已经足够了。

参考资料

如果想要进一步了解 PWM，请参阅 Wikipedia (<http://bit.ly/1iobPt8>)。

在 10.10 节中，我们会使用 PWM 来改变 RGB LED 的颜色，而在 11.4 节中，我们会利用 PWM 来控制直流电机的速度。

在树莓派上面使用面包板和跳线的更多内容，请参考 9.8 节。此外，你也可以通过滑块控件来控制 LED 的亮度，具体见 10.9 节。除此之外，还有一种方法可以控制 RGB LED

的颜色，即使用 Squid RGB LED 库，具体见 9.10 节。

10.4 制造蜂鸣声

面临问题

你想利用树莓派制造蜂鸣声。

解决方案

你可以使用压电蜂鸣器，该蜂鸣器需要连接到一个 GPIO 引脚上。

大部分小型的压电蜂鸣器都非常适合图 10-3 给出的布局方式。

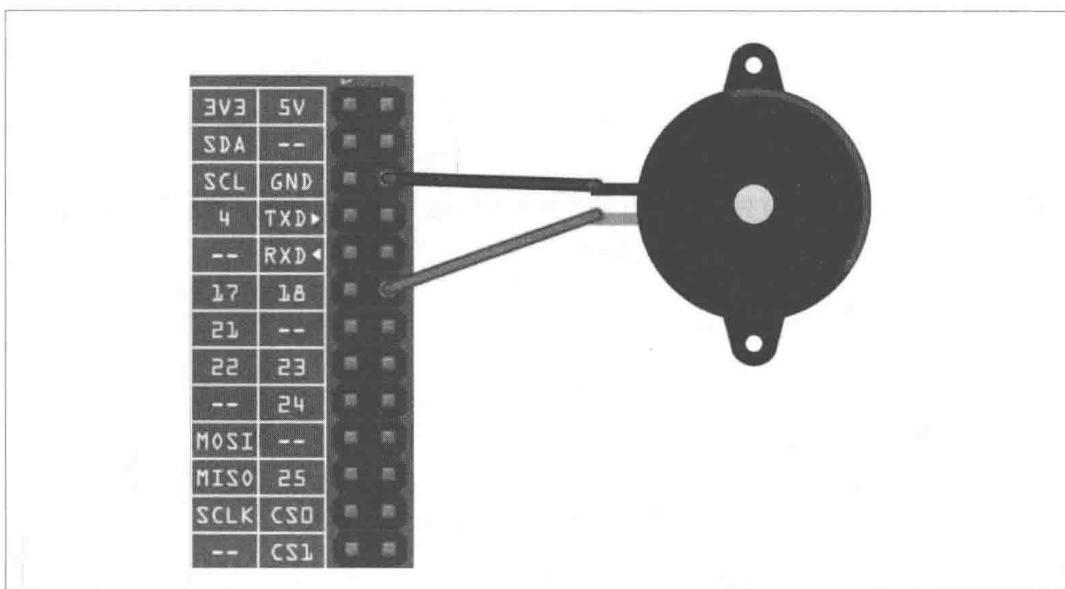


图 10-3 为树莓派连接压电蜂鸣器

我这里使用的是 Adafruit 提供的元件。

你可以通过 female-to-female 接头把蜂鸣器的引脚直接连接到树莓派上面。

这些蜂鸣器所需的电流非常小。但是，如果你想使用一个较大的蜂鸣器或者想确保安全的话，可以在 GPIO 引脚和蜂鸣器的导线之间加入一个 470Ω 的电阻。

请将下面的代码粘贴到 IDLE（见 5.2 节）或 nano（见 3.6 节）的编辑器中，并将文件保存为 `buzzer.py`。此外，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的下载资源中找到这个程序。

```
import RPi.GPIO as GPIO
```

```
import time

buzzer_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer_pin, GPIO.OUT)

def buzz(pitch, duration):
    period = 1.0 / pitch
    delay = period / 2
    cycles = int(duration * pitch)
    for i in range(cycles):
        GPIO.output(buzzer_pin, True)
        time.sleep(delay)
        GPIO.output(buzzer_pin, False)
        time.sleep(delay)

while True:
    pitch_s = raw_input("Enter Pitch (200 to 2000): ")
    pitch = float(pitch_s)
    duration_s = raw_input("Enter Duration (seconds): ")
    duration = float(duration_s)
    buzz(pitch, duration)
```

当你运行该程序时，它首先会要求你输入一个以 Hz 为单位的音调，然后就会发出几秒钟的蜂鸣声。

```
$ sudo python buzzer.py
Enter Pitch (2000 to 10000): 2000
Enter Duration (seconds): 20
```

进一步探讨

压电蜂鸣器不支持广泛的频率范围，同时音质也不太好。但是，其音调还是可以进行某些程度的调整的。但是，由代码生成的频率都是非常接近的。

该程序的工作原理，就是以较短的时间延迟来切换 GPIO 的 18 号引脚的开关状态。延迟的大小是通过音调计算得到的。音调（频率）越高，所需的延迟就越短。

参考资料

关于压电蜂鸣器的参数列表，请访问 <http://bit.ly/Iwkv2R>。

10.5 利用晶体管开关大功率直流设备

面临问题

你想控制一个大功率、低电压的直流设备（例如 12V LED 模块）的电源。

解决方案

这些大功率 LED 如果直接连接到 GPIO 引脚上面的话，点亮时会消耗大量的电流。

此外，它们要求使用的是 12V 电压，而非 3.3V。为了控制这种大功率负载，你需要使用晶体管。

在本例中，你用到的大功率晶体管名为 MOSFET (metal-oxide-semiconductor field-effect, MOSFET) 晶体管，虽然它们的价格还不到 1 美元，但是却可以承载高达 30 安培的电流，这已经是大功率 LED 所需电流的许多倍了。这里使用的 MOSFET 型号为 FQP30N06L。

图 10-4 展示了 MOSFET 在面包板上面的连接方式。请务必正确区分 LED 模块电源线的正负极。

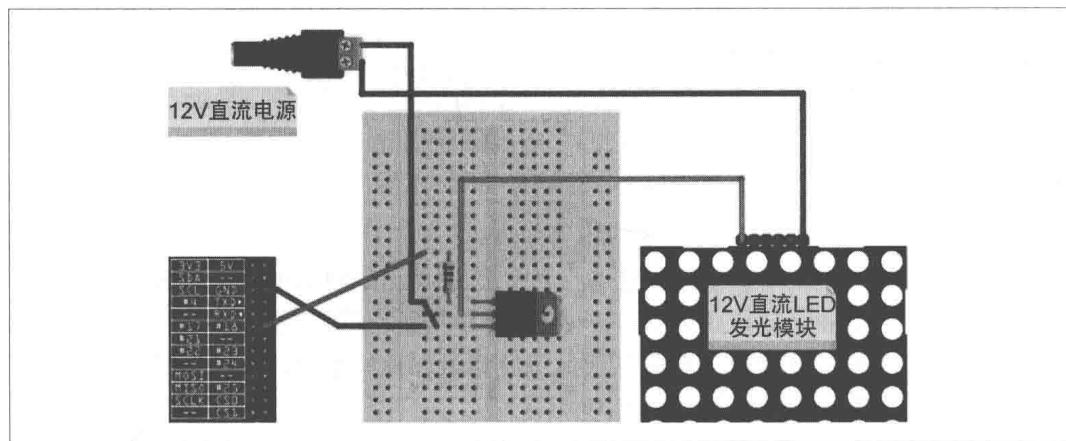


图 10-4 利用 MOSFET 控制大电流

为了进行这项实验，你需要：

- 面包板和跳线；
- $1\text{k}\Omega$ 电阻；
- FQP30N06L N-Channel MOSFET 或者 TIP120 达林顿晶体管；
- 12V 电源适配器；
- 12V 直流 LED 模块。

控制 LED 面板开关的 Python 代码与控制单个低功率 LED 的代码（见 10.1 节）完全一致，虽然后者没有使用 MOSFET。

此外，你也可以使用 PWM 与 MOSFET 控制 LED 模块的亮度（见 10.3 节）。

进一步探讨

只要你需要使用 GPIO 接口为大功率设备供电，就都得使用电池或者外接电源适配器，因为 GPIO 接口只能提供相对较低的电流（见 9.2 节）。就本例来说，你可以使用一个 12V 直流电源适配器来给 LED 面板供电。挑选电源适配器的时候，请选择具有足够承受功率能力的。因此，如果 LED 模块的功率是 5W 的话，那么至少需要一个 12V 5W 的电源（当然 6W 的会更好）。如果电源标识的是最大电流而非功率的话，那么可以通过最大电流乘以电压来计算它的功率。所以，一个 500mA 12V 的电源能够提供的功率为 6W。

为了防止 MOSFET 的开关切换引起的电流峰值导致 GPIO 引脚过载，必须使用一个电阻。由于 MOSFET 控制 LED 面板的负极，所以电源正极可以直接连接到 LED 面板的正极，并且将 LED 的负极连接到 MOSFET 的漏极。MOSFET 的源极连接 GND，MOSFET 的栅极控制从漏极到栅极的电流。当栅极的电压超过 2V 的时候，MOSFET 将会开启，从而使电流通过其自身及 LED 模块。

这里使用的 MOSFET 是 FQP30N06L，最后的字母 L 表示这是一个逻辑电平 MOSFET，即其栅阈值电压适用于 3.3V 的数字输出。当然，如果这里使用非 L 版本的 MOSFET 的话，也可能工作得很好，但是，它们无法确保栅临界电压位于 2V 到 4V 这个规定范围之内。所以，如果你运气不佳，MOSFET 正好位于 4V 端的话，它就无法正常切换开关了。

如果你不想使用 MOSFET 的话，还有一种替代方案，即使用电源达林顿晶体管，如 TIP120。由于它提供了与 FQP30N06L 一致的输出引脚，所以之前的面包板布局也无需改动。这里的电路同样适用于控制其他低电压直流设备的电源，但是电机和继电器除外，因为它们需要某些特殊处理（见 10.6 节）。

参考资料

如果想查看 MOSFET 的参数表，请访问 <http://bit.ly/18J3bxT>。

如果你希望创建一个图形用户界面来控制 LED 模块，请参考 10.8 节关于简单开关控制和 10.9 节关于使用滑块对亮度实现可变控制的内容。

10.6 使用继电器控制大功率设备的开关

面临问题

你希望开关一个不适合 MOSFET 控制的设备的电源。

解决方案

你可以使用继电器和小型晶体管。

图 10-5 展示了晶体管和继电器在面包板上面的连接方式，请务必确保以正确的方法安装晶体管和二极管。二极管的一端有一个条纹，而这里使用的晶体管一面是平面，另一面是曲面。

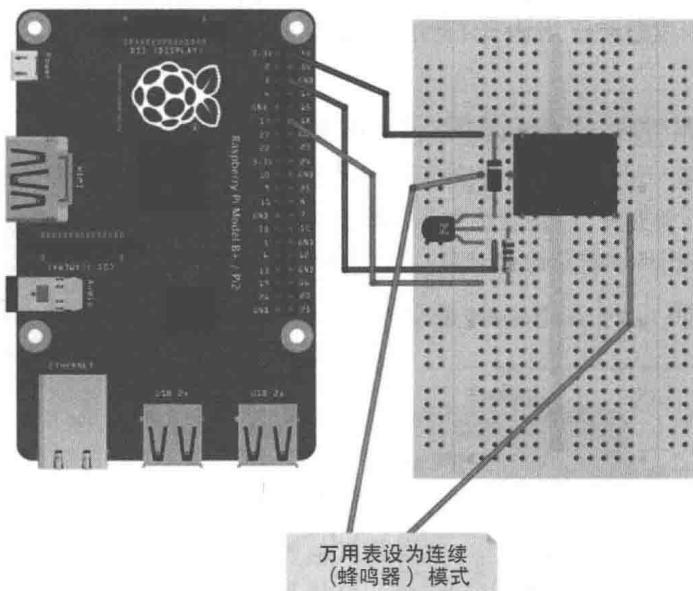


图 10-5 使用继电器和树莓派

为了进行这项实验，你需要：

- 面包板和跳线；
- $1k\Omega$ 电阻；
- 2N3904 晶体管；
- 1N4001 二极管；
- 5V 继电器；
- 万用表。

你可以直接使用 10.1 节中的 LED 闪烁程序。如果一切正常的话，你能听到每次开关时发出的滴嗒声。但是，继电器是一种低速的机械设备，所以不要企图与 PWM 一起使用，因为这样会对继电器造成损坏。

进一步探讨

继电器历史悠久，自从电子产品面市初期起，它们就随之出现了，同时，它们具有用

法简单的优点，此外，凡是需要使用开关的情形，通常都可以使用继电器，例如当你需要控制交流电的时候，或者在未知控制设备开关的布线的情况下。

如果在使用过程中超过了继电器触点的额定参数的话，就会缩短继电器的使用寿命。这种情况下，可能会出现电弧，从而导致触点最终被融合在一起。当然，还可能由于继电器过热而引发危险。当你有不明白的地方的时候，请仔细查看继电器触点的相关说明。

图 10-6 展示了典型继电器的电路图、引脚布局和外形。

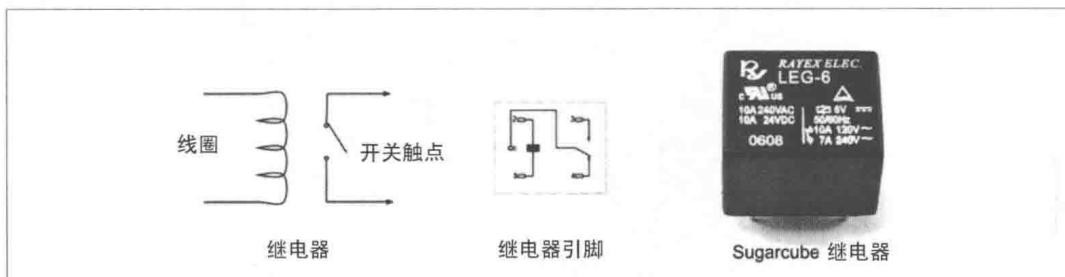


图 10-6 继电器工作原理

继电器实质上就是一个开关，当电磁铁将其触点吸合时，开关就会处于接通状态。由于电磁铁和开关之间没有任何电器连接，这就避免了驱动继电器线圈的电路与开关端的高压相接触，从而起到保护作用。

继电器的缺点是它们的动作太慢，并且动作几十万次之后通常就报废了。也就是说，它们只适用于慢速开关情形，而不适用于需要像 PWM 这样的快速开关的情形。

继电器的触点需要使用 50mA 左右的电流才能够闭合。由于树莓派的 GPIO 引脚只能提供 3mA 的电流，所以，你需要使用一个小型的晶体管来作为开关。你无需使用 10.5 节中那样的大功率 MOSFET，相反，你只需要使用一个小型晶体管就可以了。这些晶体管有 3 个连接，基极（中间引线）通过一个 1kΩ 限流电阻连接到 GPIO 引脚上面。发射极与 GND 相连，而集电极则连接到继电器的一端上。继电器的另一端连接到 GPIO 接口的 5V 输出上面。二极管的作用在于抑制晶体管迅速开关继电器触点电源而导致的高压脉冲电流。



虽然继电器可用于开关 110V 或 240V 的交流电，但是这个电压是非常危险的，所以不能将其用于面包板。如果你希望控制高压电的开关，请参考 10.7 节相关内容。

参考资料

关于使用 MOSFET 控制直流电开关的内容，请参考 10.5 节。

10.7 控制高压交流设备

面临问题

你希望使用树莓派控制 110V 或 240V 交流电的开关。

解决方案

你可以使用 PowerSwitch Tail II (参见图 10-7)。有了这种简便的设备，使得从树莓派控制交流设备的开关变得易如反掌。该设备的一端是交流电插槽，另一端是插头，类似一根延长电缆，唯一的区别在于位于该导线中间的控制盒有 3 个螺丝端子。通过将端子 2 连接 GND，将端子 1 连接到一个 GPIO 引脚，这个设备就能够用作电器的开关了。

你可以原封不动地使用 10.1 节中的 Python 代码来操作 PowerSwitch Tail，如图 10-7 所示。

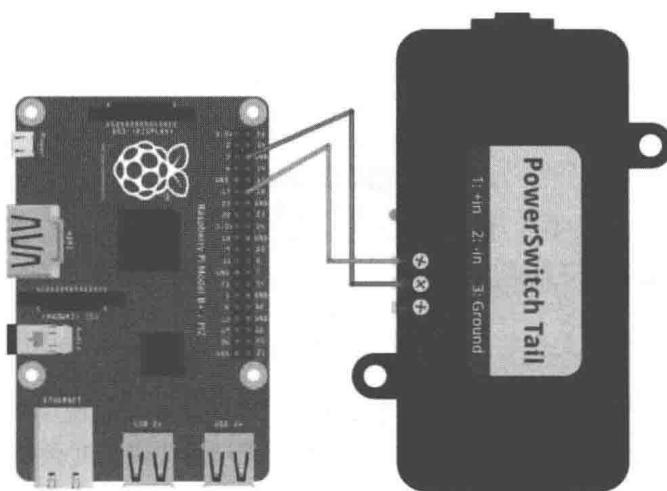


图 10-7 在树莓派上使用 PowerSwitch Tail

进一步探讨

PowerSwitch Tail 用到了一个继电器，为了控制该继电器的开关，又使用了一个名为 opto-isolator 的元件，该元件在 photo-TRIAC (一个高压光敏开关) 上有一个发光 LED。当这个 LED 发亮时，photo-TRIAC 就会处于导通状态，从而为继电器的触点通电。

由于 opto-isolator 内部的 LED 有一个限流电阻，所以利用 3.3V 的 GPIO 引脚供电时，实际上仅有 3mA 的电流通过 LED。

当然，你还可以从亚马逊或 eBay 上面找到类似 PowerSwitch Tail 这样的设备，并且还可能会更加便宜。

参考资料

关于使用大功率 MOSFET 控制直流电开关的内容，请参考 10.5 节；关于在面包板上使用继电器的内容，请参考 10.6 节。

240V 版本的 PowerSwitch Tail 套装产品现已面市，地址是 <http://bit.ly/1byrZtl>。

10.8 编写用于控制开关的用户界面

面临问题

你希望编写一个运行于树莓派上面的应用程序，并通过该程序的开关按钮来控制其他对象的开关。

解决方案

你可以使用用户界面框架 Tkinter 来编写一个 Python 程序，通过使用复选框来控制 GPIO 引脚的开关状态（见图 10-8）。

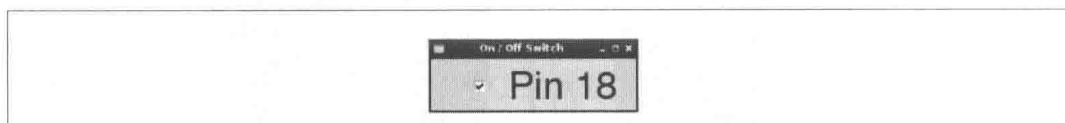


图 10-8 一个控制开关的用户界面

下面，你需要将 LED 或类似的其他输出设备连接到 GPIO 的 18 号引脚。

刚开始的时候，最好从使用 LED（见 10.1 节）开始下手。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 `gui_switch.py`。

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.check_var = BooleanVar()
        check = Checkbutton(frame, text='Pin 18',
```

```
        command=self.update,
        variable=self.check_var, onvalue=True, offvalue=False)
check.grid(row=1)

def update(self):
    GPIO.output(18, self.check_var.get())

root = Tk()
root.wm_title('On / Off Switch')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()
```

需要注意的是对于 RPi.GPIO 来说，需要具有超级用户权限才能够访问 GPIO 硬件，所以，你需要使用 sudo 命令来运行该程序，具体如下所示。

```
$ sudo python gui_switch.py
```



对于 Python 3 来说，Tkinter 库已经更名为 tkinter，注意后者中的小写字母 t。

进一步探讨

上面的示例代码定义了一个名为 App 的类，程序的大部分代码都位于这个类中。它的初始化函数创建了一个名为 check_var 的成员变量，该变量用来存放一个 BooleanVar 事例，并将其作为 variable 变量提供给复选框。这样能够确保每次复选框被选中的时候，该变量中的值都会发生变化。每当发生这种变化时，选项 command 就会运行 update 命令。

函数 update 只是简单地把 check_var 中的值写到 GPIO 输出上面。

参考资料

你也可以使用这个程序来控制 LED（见图 10-8）、大功率直流设备（见 10.5 节）、继电器（见 10.6 节）或高压交流设备（见 10.7 节）。

10.9 编写控制 LED 和电机的 PWM 功率的用户界面

面临问题

你想编写一款运行于树莓派的应用程序，通过一个滑块来控制使用 PWM 的设备的功率。

解决方案

你可以使用用户界面框架 Tkinter 来编写一个 Python 程序，利用滑块让 PWM 的占空系

数在百分之 0 到百分之 100 之间变动（见图 10-9）。

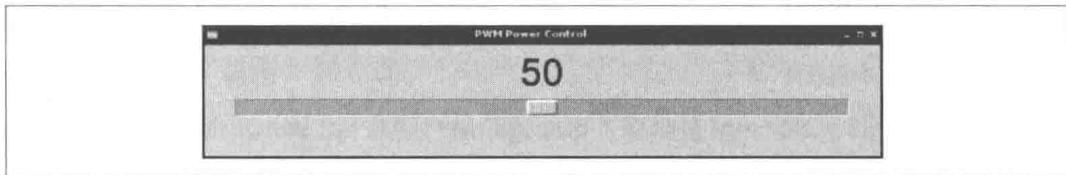


图 10-9 控制 PWM 功率的用户界面

为此，你需要将 LED 或其他类型的输出设备连接到 GPIO 的 18 号引脚上面来响应 PWM 信号。当然，为简单起见，从 LED（见 10.1 节）开始下手最为轻松。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 `gui_slider.py`。

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
pwm = GPIO.PWM(18, 500)
pwm.start(100)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        scale = Scale(frame, from_=0, to=100,
                      orient=HORIZONTAL, command=self.update)
        scale.grid(row=0)

    def update(self, duty):
        pwm.ChangeDutyCycle(float(duty))

root = Tk()
root.wm_title('PWM Power Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()
```

需要注意的是对于 RPi.GPIO 来说，需要具有超级用户权限才能够访问 GPIO 硬件，所以，你需要使用 `sudo` 命令来运行该程序。

```
$ sudo python gui_slider.py
```

进一步探讨

上面的示例代码定义了一个名为 `App` 的类，程序的大部分代码都位于这个类中。每当滑块的

值发生变化时，选项 command 就会运行 update 命令。这样就会更新输出引脚的占空系数。

参考资料

你还可以使用这个程序来控制 LED（见 10.1 节）、直流电机（见 11.4 节）或者大功率直流设备（见 10.5 节）。

10.10 改变 RGB LED 的颜色

面临问题

你想控制 RGB LED 的颜色。

解决方案

你可以使用 PWM 来控制每个红、绿、蓝通道的功率。

为了进行这项实验，你需要：

- 面包板和跳线；
- 3 个 470Ω 电阻；
- RGB 共阴极 LED；
- Perma-Proto 板（见 9.9 节）或者 Pi Plate（见 9.19 节），供制作更加持久的项目（可选）。

图 10-10 展示了 RGB LED 在面包板上面的连接方式。请确保 LED 连接方式的正确性，最长的导线应该是面包板顶部的第二根导线。这种连接称为共阴极，这是因为对于该 LED 案例而言，其内部的红色、绿色和蓝色 LED 的负极（阴极）是连接在一起的，这样做可以减少封装所需的引脚数量。

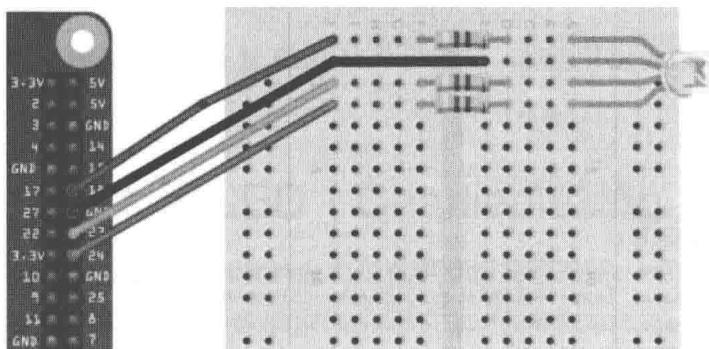


图 10-10 在树莓派上使用 RGB LED

如果你不想使用面包板的话，可以使用 Raspberry Squid（见 9.10 节）来替换它。

我们要编写的程序会使用 3 个滑块来控制 LED（见图 10-11）的红色（Red）、绿色（Green）和蓝色（Blue）通道。

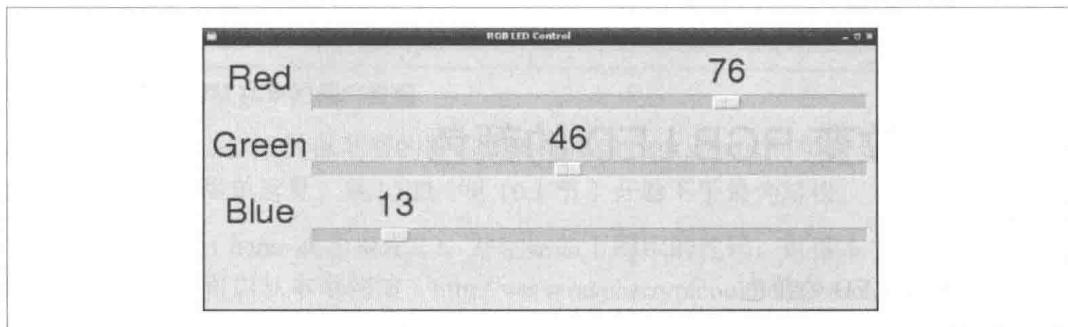


图 10-11 通过用户界面控制 RGB LED

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 gui_sliderRGB.py

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(24, GPIO.OUT)

pwmRed = GPIO.PWM(18, 500)
pwmRed.start(100)

pwmGreen = GPIO.PWM(23, 500)
pwmGreen.start(100)

pwmBlue = GPIO.PWM(24, 500)
pwmBlue.start(100)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        Label(frame, text='Red').grid(row=0, column=0)
        Label(frame, text='Green').grid(row=1, column=0)
        Label(frame, text='Blue').grid(row=2, column=0)
        scaleRed = Scale(frame, from_=0, to=100,
                         orient=HORIZONTAL, command=self.updateRed)
        scaleRed.grid(row=0, column=1)
```

```
scaleGreen = Scale(frame, from_=0, to=100,
                   orient=HORIZONTAL, command=self.updateGreen)
scaleGreen.grid(row=1, column=1)
scaleBlue = Scale(frame, from_=0, to=100,
                  orient=HORIZONTAL, command=self.updateBlue)
scaleBlue.grid(row=2, column=1)

def updateRed(self, duty):
    pwmRed.ChangeDutyCycle(float(duty))

def updateGreen(self, duty):
    pwmGreen.ChangeDutyCycle(float(duty))

def updateBlue(self, duty):
    pwmBlue.ChangeDutyCycle(float(duty))

root = Tk()
root.wm_title('RGB LED Control')
app = App(root)
root.geometry("200x150+0+0")
root.mainloop()
```

进一步探讨

上述代码的运作方式与 10.9 节中控制单个 PWM 通道的代码类似。但是，就本例而言，你需要 3 个 PWM 通道以及 3 个滑块，因为每个颜色都对应于一个滑块。这里所用的 RGB LED 的类型是共阴极的。实际上，即使你的 RGB LED 是共阳极类型的，也照常可以使用，只是要将共阳极连接到 GPIO 接口的 3.3V 引脚上。此外，你还会发现滑块会变成反向的，即 100 变成了关闭，而 0 则变为打开。

当你为这个项目挑选 LED 时，最好选带有漫反射标志的，因为这种 LED 的颜色混合效果更好。

参考资料

如果你只需控制一个 PWM 通道的话，请参考 10.9 节。

除此之外，还有一种方法可以控制 RGB LED 的颜色，即使用 Squid RGB LED 库，具体见 9.10 节。

10.11 使用大量 LED (Charlieplexing 方式)

请务必在 <http://razzpisampler.oreilly.com> 上观看本节相关的视频。

面临问题

你希望使用尽可能少的 GPIO 引脚来控制大量的 LED。

解决方案

为此，你可以利用一种称为 Charlieplexing 的技术来达到目的。该技术的名称源自其发明者——Maxim 公司的 Charlie Allen，这项技术充分利用了 GPIO 引脚在程序运行过程中可以从输出变为输入的特性。当一个引脚变为输入时，流经它的电流就低到不足以点亮 LED 或影响其他被设为输出且连接到 LED 的引脚。

图 10-12 展示了通过 3 个引脚控制 6 个 LED 的原理图。

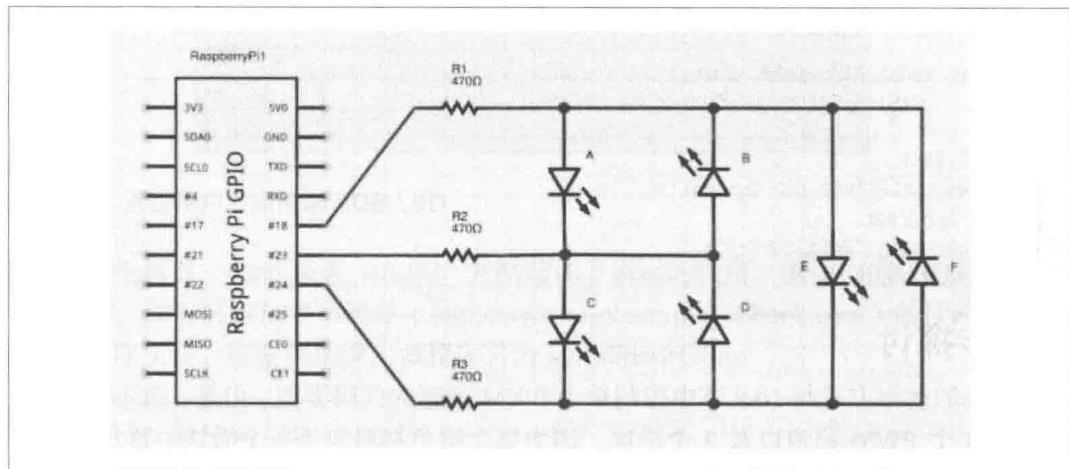


图 10-12 Charlieplexing 技术

图 10-13 展示了连接 LED 和电阻的面包板的布局。

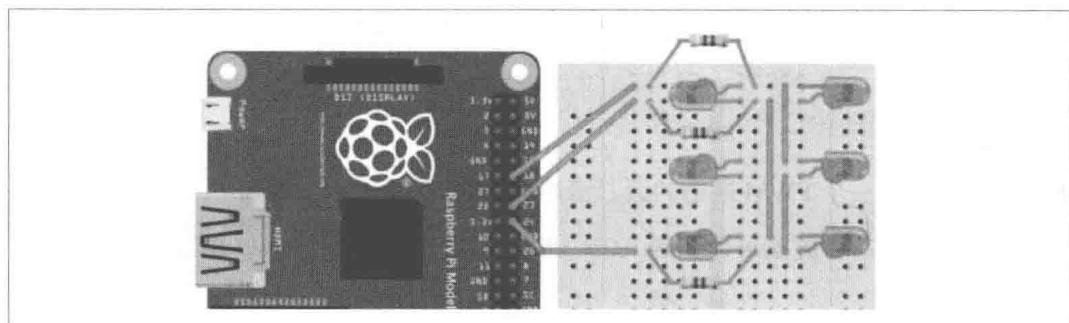


图 10-13 Charlieplexing 技术所用面包板的布局

为了进行这项实验，你需要：

- 面包板和跳线；
- 3 个 470Ω 电阻；
- 6 个 LED。

打开一个编辑器（ nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 charlieplexing.py。

这个示例代码运行时，会要求你输入一个介于 0 到 5 之间的一个数字，然后就会根据这个数字点亮 6 个 LED 中相应的那个。

```
import RPi.GPIO as GPIO

pins = [18, 23, 24]

pin_led_states = [
    [1, 0, -1], # A
    [0, 1, -1], # B
    [-1, 1, 0], # C
    [-1, 0, 1], # D
    [1, -1, 0], # E
    [0, -1, 1] # F
]

GPIO.setmode(GPIO.BCM)

def set_pin(pin_index, pin_state):
    if pin_state == -1:
        GPIO.setup(pins[pin_index], GPIO.IN)
    else:
        GPIO.setup(pins[pin_index], GPIO.OUT)
        GPIO.output(pins[pin_index], pin_state)

def light_led(led_number):
    for pin_index, pin_state in enumerate(pin_led_states[led_number]):
        set_pin(pin_index, pin_state)

set_pin(0, -1)
set_pin(1, -1)
set_pin(2, -1)

while True:
    x = int(raw_input("Pin (0 to 5):"))
    light_led(x)
```

进一步探讨

为了理解 Charlieplexing 的工作原理，让我们假设你希望点亮图 10-12 的 LED A。对于 LED 来说，只有它的正极是高电平，负极是低电平的时候，它才会发光。如果电压不符合上述条件，那么它是不会发光的。为了点亮 LED A，你需要将它的一根引线（通过一个电阻）连接到 GPIO 的 18 号引脚来提供高电平，并将 LED A 的另一根引线通过一个电阻连接到 GPIO 的 23 号引脚来提供低电平。但是，你还必须同时确保 GPIO 的 24 号引脚被设置为输入，否则的话，LED C 或 D 也会被点亮，具体取决于 GPIO 的 24 号引脚电平的高低情况。

数组 pin_led_states 用来保存每个 LED 对应的 GPIO 设置。如果值为 0，表示该引脚是

低电平；如果值为 1，则表示高电平；如果值为 -1，则表示该引脚被设置为输入。

每个 GPIO 引脚所控制的 LED 的编号可以通过下列公式计算得到。

$$\text{LEDs} = n^2 - n$$

通过使用 4 个引脚，你就可以控制 16、4 或 12 个 LED；如果使用 10 个引脚的话，你就可以控制多达 90 个 LED 了。

在本例中，你每次只点亮 1 个 LED。如果想每次点亮多个 LED，那么就需要运行一个刷新循环，从而将期望的 LED 状态保存到数组中并刷新显示，并在进行下一个之前打开需要点亮的 LED。当然，这个过程必须足够快，唯有如此才能使多个 LED 最终看起来仿佛是同时点亮的。

当你想同时点亮多个 LED 的时候，LED 的数量越多，LED 的点亮次数会越少，同时 LED 会越暗。

参考资料

如果想详细了解 Charlieplexing 技术，请参考 Wikipedia (<https://en.wikipedia.org/wiki/Charlieplexing>)。

10.12 将模拟仪表用作显示器

面临问题

你希望给树莓派连接一台模拟电压表。

解决方案

我们假设你有一个 5V 电压表，你可以使用一个 PWM 输出来直接驱动该电压表，为此只需将电压表的负极接地，正极连接 GPIO 引脚即可（见图 10-14）。如果你的电压表是常用的 5V 类型的话，那么通常最高只能显示 3.3V。

如果你想使用全范围 5V 电压表的话，需要用一个晶体管来作为 PWM 信号开关，以及一个 $1\text{k}\Omega$ 电阻来限制晶体管基极的电流。为了进行这项实验，你需要：

- 5V 电压表；
- 面包板和跳线；
- 2 个 $1\text{k}\Omega$ 电阻；
- 2N3904 晶体管。

该实验中面包板的布局如图 10-15 所示。

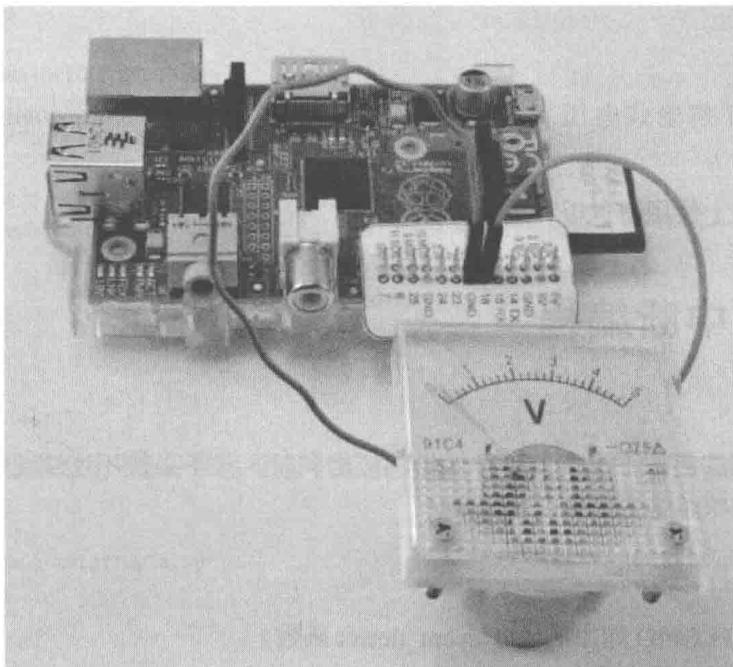


图 10-14 直接将电压表连接至 GPIO 引脚

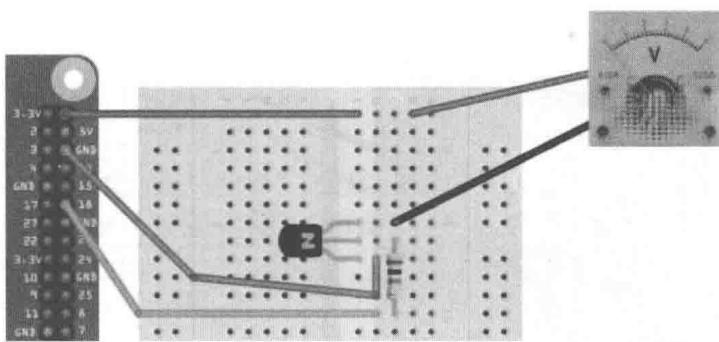


图 10-15 在 3.3V GPIO 上使用 5V 电压表

进一步探讨

为了测试该电压表，可以使用 10.9 节中控制 LED 亮度的程序。

你可能已经注意到了，指针在刻度盘两端读数的时候，通常都是稳定的，但是读数在其他地方的时候，指针往往会发生轻微抖动。之所以发生这种现象，主要是由于 PWM 信号的生成方式的副作用导致的。为了获得更加稳定的读数结果，你可以使用外用

PWM 硬件，比如 11.3 节中所用的 16 通道模块等。

参考资料

若想进一步了解老式电压表工作原理，请访问 Wikipedia (<https://en.wikipedia.org/wiki/Voltmeter>)。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

10.13 中断编程

面临问题

你希望对类似按钮按下之类的事件做出响应的同时，又不必使用连续轮询的方式来检查输入引脚的状态是否发生变化。

解决方案

你可以使用 RPi.GPIO 库中的 add_event_detect 函数。

下面的示例将会展示按钮被按下的时候，如何附加一个由此触发的中断服务例程。

使用导线将开关连接至面包板，具体如图 10-16 所示。此外，你也可以使用 Squid Button (见 9.11 节)。

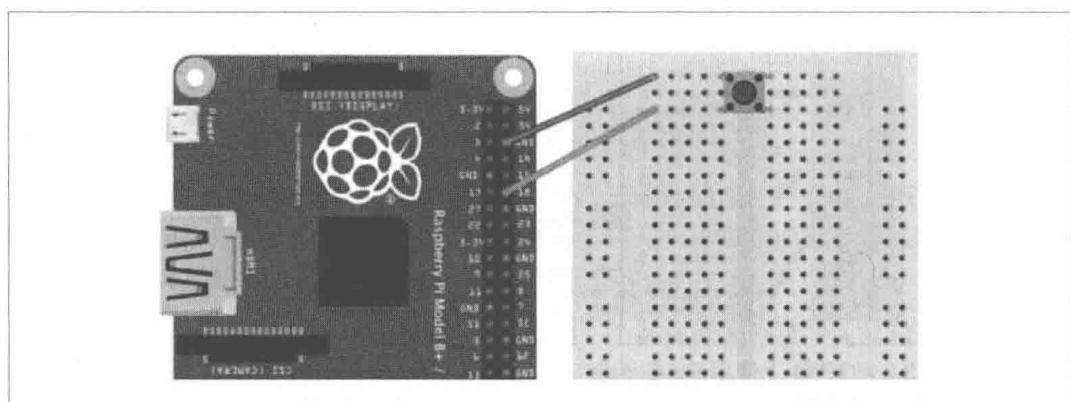


图 10-16 将开关连接至 GPIO 输入来演示中断

打开一个编辑器 (nano 或者 IDLE)，并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，就本例来说，该程序名为 interrupts.py。

这个示例程序的代码会连续更新秒数，并在按钮被按下时显示消息。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
def my_callback(channel):
    print('You pressed the button')

GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(18, GPIO.FALLING, callback=my_callback)

i = 0
while True:
    i = i + 1
    print(i)
    time.sleep(1)
```

你可以试着以超级用户身份来运行这个程序。当你按下按钮的时候，应该会看到如下所示的输出。

```
$ sudo python interrupts.py
1
2
3
You pressed the button
4
You pressed the button
5
You pressed the button
You pressed the button
6
```

进一步探讨

你只需利用一个循环不停地进行检查，自然就能在按钮被按下或者 GPIO 输入发生变化的时候检测到这些事件，示例代码如下所示。

```
while True:
    if GPIO.input(18) == False:
        # put the code to be actioned here
    time.sleep(0.1)
```

这种做法的不足之处在于你在检测按钮是否被按下的过程中无法做其他事情。第二个缺点是如果按钮下按速度非常快的话，那么你还没有来得及用 GPIO.input 记录下来，事件的发生和消失就都过去了。这种机制叫作轮询。

中断的工作原理不同于轮询。中断允许你将函数关联到一个引脚上面，这样以来，当输入上的电压从低变高，或者从高变低的时候，就会触发这个函数运行。

你可以通过上面的示例程序来了解这个工作原理。首先，定义一个名为 my_callback 函数

数，该函数只有一个参数。该参数的作用是规定触发中断的输入，从而允许你使用同一个处理函数来处理许多中断。

```
def my_callback(channel):
    print('You pressed the button')
```

就本例而言，这个回调函数只是显示了一则消息。

实际进行连接工作的代码如下所示。

```
GPIO.add_event_detect(18, GPIO.FALLING, callback=my_callback)
```

第一个参数指定了引脚(18 号)。第二个参数可以是 GPIO.FALLING 或者 GPIO.RISING。如果被设为 FALLING 的话，那么只有当 GPIO 引脚的电压从高到低时才会调用该函数。本例就是这种情况，开关将被内部上拉电阻的输入电压拉低。反之，如果第二个参数被设为 RISING，那么只有当输入从低变高时才会调用函数 (当开关被释放时)。

当事件处理函数运行的时候，不会阻止主计数循环，因为它实际上是运行在自己单独的线程中的。

当开关按下时，通常会发生抖动。这就意味着开关不能一次性地从打开状态转换到关闭状态，而是在两种状态之间切换若干次，即抖动，这样的话，当实际上只按下一次的时候，看起来却像是在瞬间按下了多次。

如果你不停地按下按钮的话，就可能看到这种现象对输出的影响了，即按下一次按钮却输出了多个消息。

这个库实际上有一个防抖选项，其原理是在规定时间内不允许重复触发中断。为了利用该功能，只要在调用 add_event_detect 函数时额外加入一个可选参数 bouncetime 即可。参数 bouncetime 的值是以毫秒为单位的。

```
GPIO.add_event_detect(18, GPIO.FALLING, callback=my_callback, bouncetime=100)
```

参考资料

关于树莓派上各种开关的使用详解，请参考 12.1 节。

11.0 引言

在本章中，我们将介绍如何将各种不同类型的电机应用于树莓派。

11.1 控制伺服电机



请务必在 <http://razzpisampler.oreilly.com> 上观看本节相关的视频。

面临问题

你想使用树莓派来控制伺服电机的转角。

解决方案

你可以使用 PWM 控制伺服电机脉冲的宽度来控制其位置。

尽管这种方式是可行的，但是生成的 PWM 并非完全平稳，所以伺服电机会产生一些抖动。一种替代方案是利用 11.2 节中介绍的设备驱动软件 ServoBlaster 来生成更加稳定的脉冲定时。

如果你的树莓派是老式的树莓派 1，那么你还需为伺服电机提供一个单独的 5V 电源，因为负载电流的峰值会导致树莓派崩溃或过载。如果你使用的是 B+ 或更新版本的树莓派，基于板载调压功能的改进，我们可以直接利用 GPIO 端口的 5V 引脚来给小型的伺服电机供电。

图 11-1 展示了一个可以与树莓派 B+ 轻松搭配的小型的 9g 伺服电机。

通常情况下，伺服电机的 5V 引线是红色的，地线是棕色的，而控制线是橘色的。5V 导线和接地线需要连接至 GPIO 接头的 5V 和 GND 引脚上面，而控制线则需要连接到 18 号引脚上面，为此，可以使用 female-to-male 的引线完成连接。

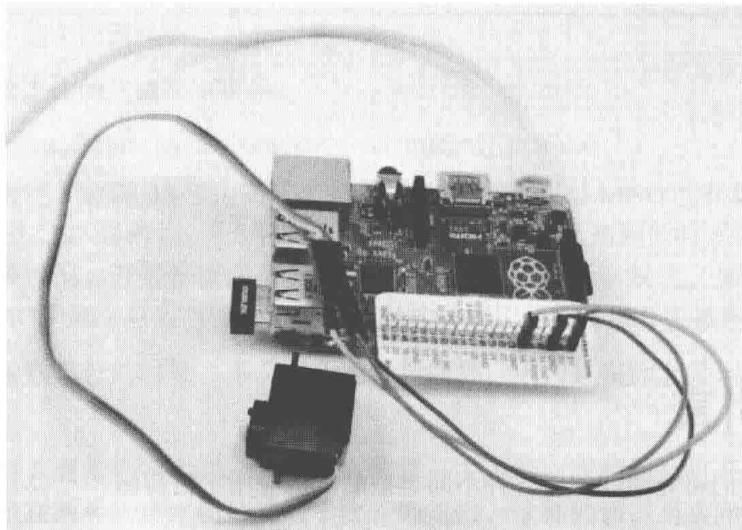


图 11-1 将小型伺服电机直接连接到树莓派 B+

如果你打算使用独立电源，最好将所有导线都布置到一个面包板上面。

为了完成本次实验，你需要：

- 5V 伺服电机；
- 面包板和跳线；
- $1k\Omega$ 电阻；
- 5V 1A 电源或者 4.8V 电池组。

该实验中面包板的布局如图 11-2 所示。

这里的 $1k\Omega$ 电阻不是必不可少的，但是它确实可以防止 GPIO 引脚在伺服电机发生故障时，控制信号中的瞬间高强电流对齐造成的损害。

如果你愿意的话，也可以使用电池组来给伺服电机供电。一个可以容纳 4 节 5 号电池的电池仓和充电电池大约可以提供 4.8V 的电压，这对于一个伺服电机来说已经足够了。使用 4 节碱性 5 号电池可以提供 6V 电压，这对于多数伺服电机来说也很好了，但是你最好还是检查一下自己伺服电机的参数手册，以确保它确实可以工作在 6V 电压下。

设置伺服电机角度的用户界面是基于 `gui_slider.py` 程序的，该程序原本是用于控制 LED 亮度的（见 10.9 节）。但是，你可以进一步修改它：使用滑块来设置一个介于 0 ~ 180

度之间的角度（见图 11-3）。

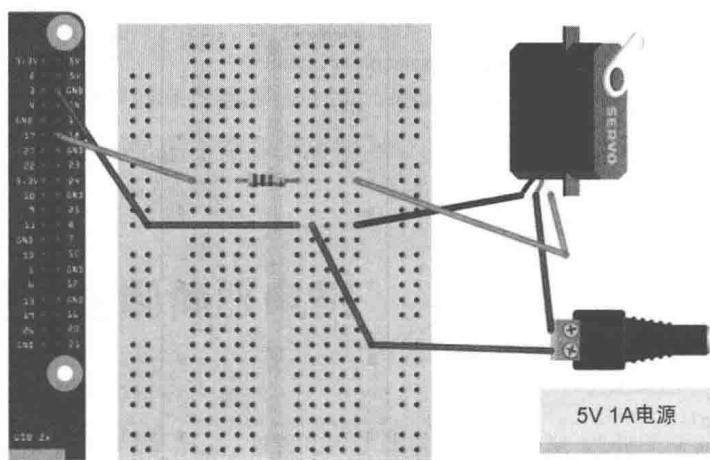


图 11-2 控制伺服电机

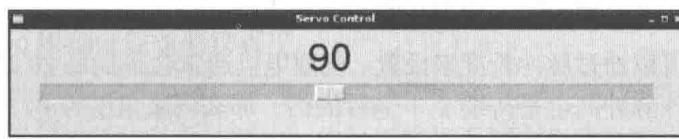


图 11-3 控制伺服电机的用户界面

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 servo.py 。

需要注意的是由于这个程序使用了一个图形用户界面，所以你无法从 SSH 来运行它。

你必须从树莓派的窗口环境中来运行它，或者通过 VNC （见 2.8 节）或 RDP （见 2.9 节）来远程控制该软件。此外，你还需要通过超级用户身份来运行该程序，所以，你需要使用命令 sudo python servo.py 。

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
pwm = GPIO.PWM(18, 100)
pwm.start(5)

class App:
```

```

def __init__(self, master):
    frame = Frame(master)
    frame.pack()
    scale = Scale(frame, from_=0, to=180,
                 orient=HORIZONTAL, command=self.update)
    scale.grid(row=0)

    def update(self, angle):
        duty = float(angle) / 10.0 + 2.5
        pwm.ChangeDutyCycle(duty)

root = Tk()
root.wm_title('Servo Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()

```

进一步探讨

伺服电机常用于远程控制车辆和机器人。大部分伺服电机都不是连续运转的，也就是说，它们不是转一圈，而是转大约 180 度。

伺服电机的位置可以通过脉冲长度来设置。伺服电机通常会预期每 20 毫秒内至少收到一个脉冲。如果该脉冲高电平持续了 1 毫秒的话，那么伺服角度为 0；如果持续了 1.5 毫秒，那么伺服电机会位于中心位置；如果持续 2 毫秒的话，伺服角度为 180 度（见图 11-4）。

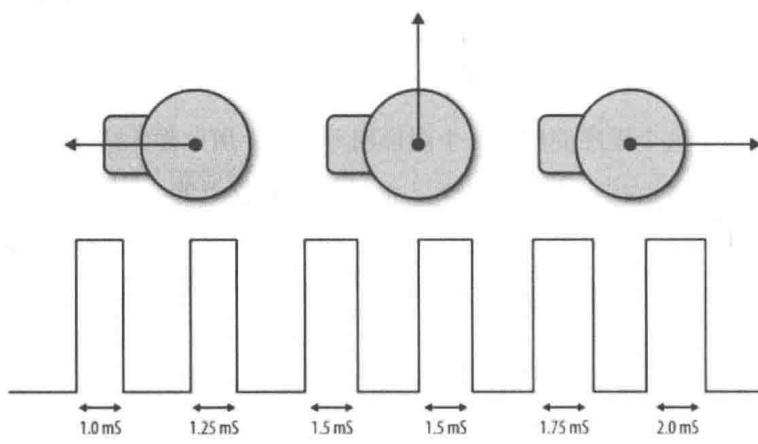


图 11-4 伺服电机

这个示例程序将 PWM 频率设置为 100Hz，也就是每 10 毫秒向伺服电机发送一个脉冲。伺服角度将转换为 0 ~ 100 之间的一个占空系数。

参考资料

如果你需要控制多个伺服电机，或者需要更高的稳定性和精确性的话，那么你可以使用一个专用的伺服电机控制模块，具体参考 11.3 节。

Adafruit 还提供了另一种伺服控制方法 (<http://bit.ly/17FVspx>)。

关于还有一种使用 ServoBlaster 设备驱动软件生成更加稳定的脉冲定时的替代方案，具体参见 11.2 节。

11.2 精确控制伺服电机

面临问题

对于你的伺服电机应用程序来说，RPi.GPIO 库提供的 PWM 函数无法满足其精密性或稳定性要求。

解决方案

你可以使用 Servo Blaster 设备驱动程序。

Servo Blaster 软件是由 Richard Hurst 发布的它使用树莓派的 CPU 硬件来生成脉冲，并且与 RPi.GPIO 库相比其计时更加精确。你可以使用下列命令来安装这个软件，之后，需要重新启动树莓派。

```
git clone git://github.com/richardghirst/PiBits.git
cd PiBits/ServoBlaster/user
sudo make
sudo make install
```

这个程序取自 11.1 节，你可以通过修改让它使用 Servo Blaster 代码。修改后的程序可以从文件 servo_blaster.py 中找到，这里假设伺服控制引脚被连接至 GPIO18 号引脚。

```
from Tkinter import *
import os
import time

servo_min = 500 # uS
servo_max = 2500 # uS

servo = 2 # GPIO 18

def map(value, from_low, from_high, to_low, to_high):
    from_range = from_high - from_low
    to_range = to_high - to_low
    scale_factor = float(from_range) / float(to_range)
    return to_low + (value / scale_factor)
```

```

def set_angle(angle):
    pulse = int(map(angle, 0, 180, servo_min, servo_max))
    command = "echo {}={}us > /dev/servoblaster".format(servo, pulse)
    os.system(command)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        scale = Scale(frame, from_=0, to=180,
                      orient=HORIZONTAL, command=self.update)
        scale.grid(row=0)

    def update(self, angle):
        set_angle(float(angle))

root = Tk()
root.wm_title('Servo Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()

```

用户界面的代码与 11.1 节中的几乎没有变化，不同之处在于 `set_angle` 函数。这个函数首先利用函数 `map` 通过常量 `servo_min` 和 `servo_max` 将角度转换为脉冲持续时间。之后，它构造了一个类似于命令行方式下面运行的命令。该命令行的格式为：开头是一个 `echo` 命令，后面是需要控制的伺服电机的数量，然后是等号，最后是以毫秒为单位的脉冲持续时间。这个命令的字符串部分指向设备/`/dev/servoblaster`。然后，伺服电机就会随之调整它的角度。



停用 ServoBlaster

当 ServoBlaster（更准确地说是 `servo.d`）运行的时候，你无法使用伺服电机的引脚做任何事情，同时树莓派上面的音频也将无法使用。所以，当你由于某种原因需要使用这些引脚的时候，可以使用下列所示的命令停用 ServoBlaster，然后重新启动树莓派。

```
$ sudo update-rc.d servoblaster disable
$ sudo reboot
```

你的树莓派重启之后，ServoBlaster 就无法再控制这些引脚了。当然，你可以通过下列命令随时重新运行 ServoBlaster。

```
$ sudo update-rc.d servoblaster enable
$ sudo reboot
```

进一步探讨

实际上，ServoBlaster 驱动程序确实非常强大，通过适当的配置之后，你可以使用近乎所有的 GPIO 引脚来控制伺服电机。在默认设置情况下，它将 8 个 GPIO 引脚定义为伺服电机控制引脚。每个引脚都有一个通道编号，具体见表 11-1。

表 11-1 用于 ServoBlaster 的伺服通道默认引脚

伺服通道	GPIO 引脚
0	4
1	17
2	18
3	27
4	22
5	23
6	24
7	25

连接过多的伺服电机会导致跳线变成一团乱麻。如果使用类似 MonkMakes Servo Six 这样的电路板的话，就能极大地简化伺服电机与树莓派之间的布线。

参考资料

关于 Servo Blaster 的完整介绍，请访问 <https://github.com/richardghirst/PiBits/tree/master/ServoBlaster>。

如果你无需 Servo Blaster 的精确定时功能，可以选择使用 RPi.GPIO 库，它也能为伺服电机生成脉冲，具体参见 11.1 节。

11.3 控制多台伺服电机

面临问题

你需要在保持高精度的情况下控制多台伺服电机。

解决方案

尽管 11.2 节中的 Servo Blaster 代码可以精确地控制多个伺服电机，但是它仍然无法解决所有这些伺服电机导线之间的布线难题。为此，你可以使用一个面包板，但是效果仍不理想，因为这些导线很可能会变松。

为了让导线的连接更加轻松，你可以使用如图 11-5 所示的伺服电机 HAT。

这个 HAT 允许你使用树莓派的 I2C 接口来控制多达 16 伺服电机或者 PWM 通道。这些伺服电机只需直接插进 HAT 即可使用。

该模块的逻辑电路是通过树莓派的 3.3V 连接来供电的。这样就可以与伺服电机的电源完全隔离，后者是由外部 5V 电源适配器来供电的。

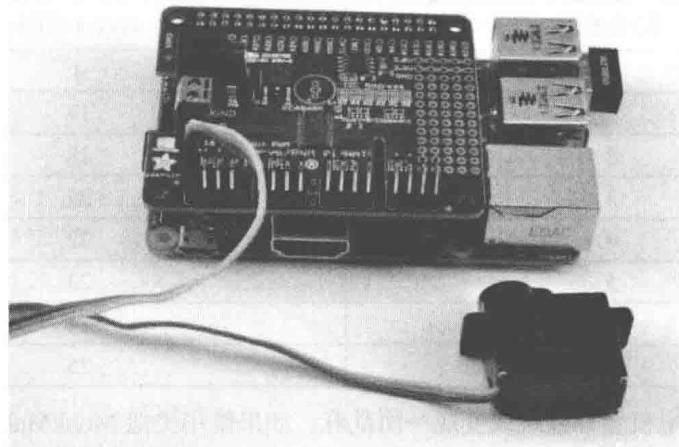


图 11-5 Adafruit 的伺服电机 HAT

如果你喜欢的话，也可以利用电池组而非电源来给伺服电机供电。一个可以容纳 4 节 5 号电池的电池仓和充电电池大约可以提供 4.8V 的电压，这对于大部分伺服电机来说已经足够了。使用 4 节碱性 5 号电池可以提供 6V 电压，这对于多数伺服电机来说也很好了，但是你最好还是检查一下自己伺服电机的参数手册，以确保它确实可以工作在 6V 电压下。

设计人员已经对连接伺服电机的引脚接头做了相当合理的安排，所以你可以直接将伺服电机安装到这些引脚上面。在安装的过程中，务必保证安装方式的正确性。

为了使用该模块的 Adafruit 软件，你需要在树莓派上面配置 I2C（见 9.3 节）。

实际上，Adafruit 库并没有提供安装脚本，而只是一个含有若干文件的目录。所以，你在使用它的时候，需要切换到其下载目录下面，否则你的程序就无法找到它们。

为了下载树莓派所需的所有 Adafruit 软件库，可以使用如下所示的命令。

```
$ git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git  
$ cd Adafruit-Raspberry-Pi-Python-Code  
$ cd Adafruit_PWM_Servo_Driver
```

最后两行命令的作用是将当前目录设置为包含 PWM 所用代码及 Adafruit 提供的示例程序所在目录，你可以使用下面的命令来运行示例程序。

```
$ sudo python Servo_Example.py
```

下面是另一个示例代码，它修改自 10.2 节中的代码，有了它，你就可以使用滑块把伺服电机的转角设置为 0 ~ 180 度之间的一个角度。这个程序文件必须保存到 Adafruit_PWM_Servo_Driver 目录下面。滑块将同时改变通道 0 和通道 1 上的伺服位置，所以，两个伺服电机都会位于该滑块设置的位置。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 servo_module.py 。需要注意的是由于该程序使用了图形用户界面，所以你无法从 SSH 运行它。你必须从树莓派的窗口环境中来运行它，或者通过 VNC （见 2.8 节）或 RDP （见 2.9 节）来远程控制该软件。

```
from Tkinter import *
from Adafruit_PWM_Servo_Driver import PWM
import time

pwm = PWM(0x40)
pwm.setPWMFreq(50)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        scale = Scale(frame, from_=0, to=180,
                      orient=HORIZONTAL, command=self.update)
        scale.grid(row=0)

    def update(self, angle):
        pulse_len = int(float(angle) * 500.0 / 180.0) + 110
        pwm.setPWM(0, 0, pulse_len)
        pwm.setPWM(1, 0, pulse_len)

root = Tk()
root.wm_title('Servo Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()
```

进一步探讨

在 import 命令后面的第一行新建了一个 PWM 实例，它使用 I2C 地址作为其参数，就本例来说，其值为 0×40 。该模块提供了焊盘连接，所以当 I2C 地址与其他使用中的 I2C 设备发生冲突或者你想使用多个此类模块的时候，可以修改 I2C 地址。下一行代码将 PWM 频率设为 50Hz，这样就能每 20 毫秒更新一次脉冲了。

真正将 PWM 设置到某个特定通道的代码为：

```
pwm.setPWM(0, 0, pulse_len)
```

第一个参数是需要修改占空系数的 PWM 通道。每个 PWM 周期被分为 4096 个时间片段，第二个参数用来指定脉冲从哪个时间片段开始。这个参数应该始终设为 0 。第三个参数用来规定脉冲应当在第几个时间片段结束。下面代码中的常量 500.0 和 110 是历经多次实验优化出来的，能够尽可能让伺服电机的转角接近 180 度。

```
pulse_len = int(float(angle) * 500.0 / 180.0) + 110
```

为这个模块选择电源的时候，不要忘了一个标准的远程控制伺服电机在运作过程中可以轻易消耗掉 400mA 电流，而在有负载的情况下，所消耗的电流将会更多。所以，如果你打算同时驱动多台大型伺服电机的话，必须配备一个大的电源适配器。

参考资料

如果你的树莓派位于伺服电机的右侧，那么伺服电机 HAT 将是你的不二之选。但是，如果伺服电机与你的树莓派所在位置较远的话，那么可以选择 Adafruit 的另一款伺服电机 module（产品序列号为 815），它提供了与伺服电机 HAT 相同的伺服电机控制器硬件，但是只有 4 个引脚用于连接该电路板的 I2C 接口与树莓派的 I2C 接口。

关于 Adafruit 为树莓派提供的代码库的相关文档，请访问 <http://bit.ly/1iomEey>。

11.4 控制直流电动机的速度

面临问题

你需要使用树莓派控制直流电机的速度。

解决方案

你可以使用 10.5 节中同样的设计。不过，最好给电机添加一个二极管，以防止电压尖峰损坏晶体管，甚至树莓派。为此，可以选择 1N4001 二极管，它的一端有一个条纹，以便于正确连接。

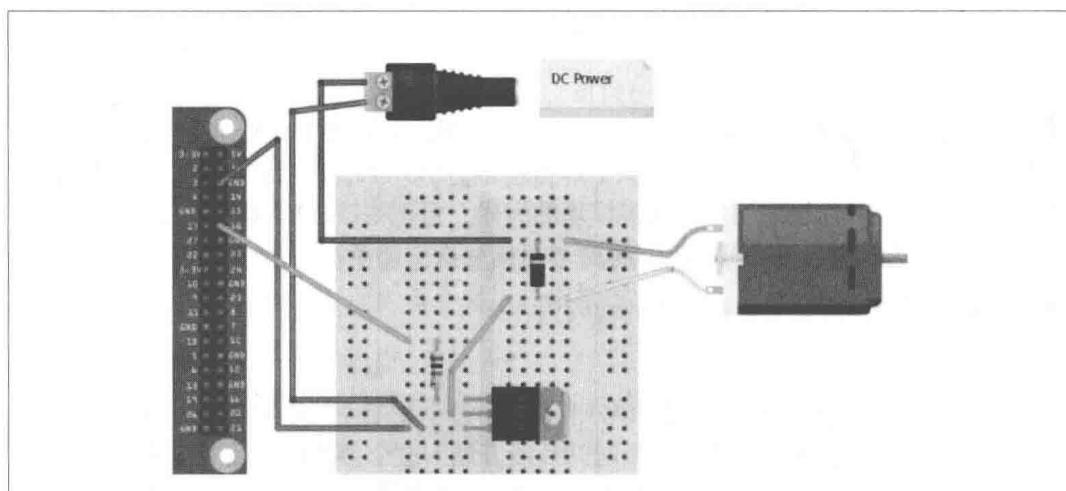


图 11-6 控制高功率电机

你需要用到下列元件：

- 3V 到 12V 的直流电机；
- 面包板和跳线；
- $1k\Omega$ 电阻；
- MOSFET 晶体管 FQP30N06L；
- 1N4001 二极管；
- 与电机电压匹配的电源。

为了控制电机的速度，你可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，该程序名为 `gui_sliderRGB.py`。需要注意的是由于该程序使用了图形用户界面，所以你无法从 SSH 运行它。你必须从树莓派的窗口环境中来运行它，或者通过 VNC (见 2.8 节) 或 RDP (见 2.9 节) 来远程控制该软件。

进一步探讨

如果你只打算使用低功率直流电机 (小于 200mA)，你可以使用一个更小 (且更廉价) 的晶体管 (如 2N3904)。使用 2N3904 的面包板布局如图 11-7 所示。

你也许使用了 GPIO 接口引出的 5V 导线给小型电机供电，并且侥幸成功。但是，如果你发现树莓派崩溃后，那么就应该使用外部电源了，具体如图 11-6 所示。

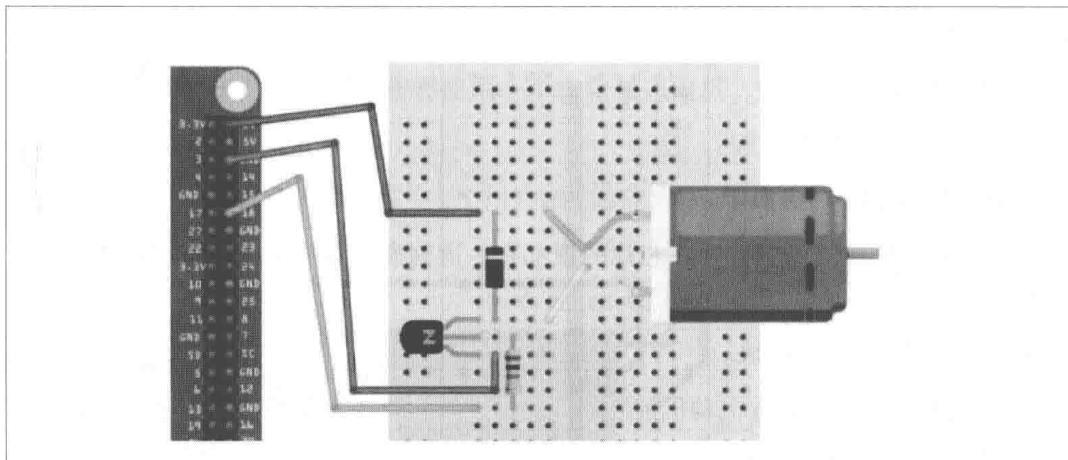


图 11-7 控制低功率电机

参考资料

本节中的设计只是用来控制电机的速度，它无法控制电机的方向。关于控制电机方向

的相关内容，请参见 11.5 节。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

11.5 控制直流电机的方向

面临问题

你想控制小型直流电机的速度和方向。

解决方案

你可以使用一个 H-Bridge 芯片或模块。

要想控制一个电机，有两种方案可供选择。第一种是“DIY 方式”，需要用到免焊面包板和 L293D 芯片。第二种设计是使用 SparkFun 现成的 H-Bridge 模块，通过跳线接头，直接将其连接到树莓派。

无论是 L293D，还是 SparkFun 模块，都可以在无需借助任何外部硬件的情况下驱动两个电机。下面我们讨论控制直流电机的另外一些选项。

选项 1：L293D 芯片与面包板。

如果选择了 L293D 方式，那么你将需要：

- 3V 到 12V 的直流电机；
- 面包板和跳线；
- L293D 芯片；
- 与电机电压匹配的电源。

面包板的布局如图 11-8 所示。

请确保芯片面向正确的方向：在芯片的顶部有一道刻痕，刻痕所在的一端应该位于面包板的顶部。

选项 2：电动机控制器模块。

如果你打算使用 SparkFun 电动机控制器或类似的电动机控制器模块，那么你将需要：

- 3V 到 12V 的直流电机；
- 跳线；
- 引脚插头；
- SparkFun 电机驱动 1A dual；

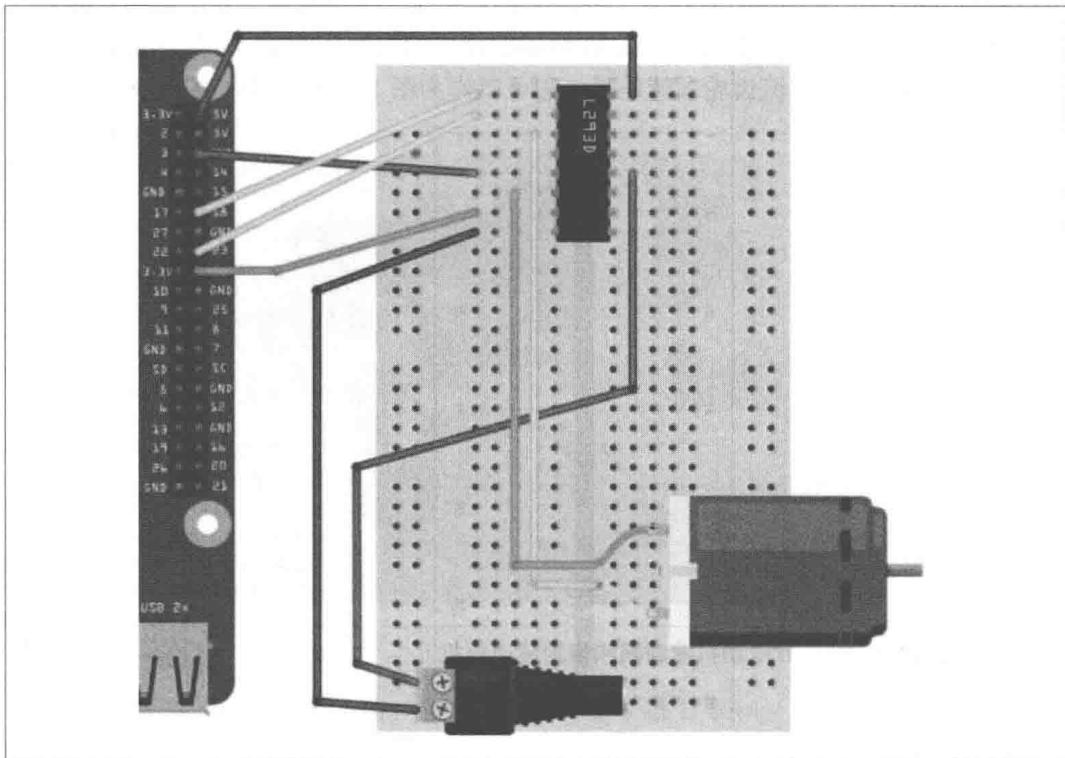


图 11-8 使用 L293D 芯片控制电机

- 与电机电压匹配的电源。

具体布局如图 11-9 所示。需要注意的是图中展示的电机只是直流电机自身。如果你的项目需要驱动轮子的话，通常需要使用一个齿轮减速电机，它组合使用电机和变速箱以降低 RPM，同时增加扭矩。

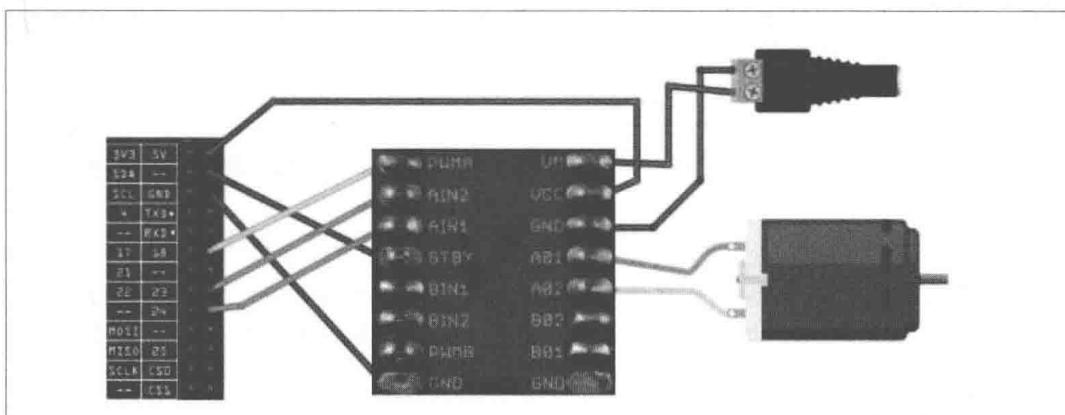


图 11-9 连接 SparkFun 电机控制器

电机控制器模块本身不带引脚插头，所以你需要提前给电路板添加相应的引脚插头。在进行连接的时候，可以使用 female-to-female 接头的引线。图 11-10 展示的模块已经使用导线连接好小型直流齿轮减速电机，它可以作为图 11-9 所示布局的布线图。

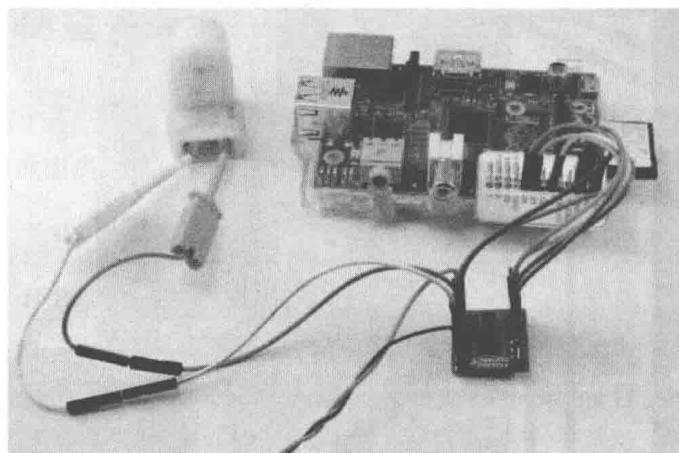


图 11-10 使用 SparkFun 电动机控制器和齿轮减速电机

软件

无论哪种硬件选项，都可以使用同一个程序来控制电机。为此你可以输入字母 f 或者 r，后面跟上一个介于 0~9 之间的数字。这样的话，电机就会按照数字所代表的速度向前或向后运动。

```
$ sudo python motor_control.py
Command, f/r 0..9, E.g. f5 :f5
Command, f/r 0..9, E.g. f5 :f1
Command, f/r 0..9, E.g. f5 :f2
Command, f/r 0..9, E.g. f5 :r2
```

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 `motor_control.py`。这个程序是在命令行下面使用的，所以你可以从 SSH 运行它。

如果你希望在 Python 3 下面使用这个程序的话，只需要将命令 `raw_input` 改为 `input` 即可。

```
import RPi.GPIO as GPIO
import time

enable_pin = 18
in1_pin = 23
in2_pin = 24
GPIO.setmode(GPIO.BCM)
GPIO.setup(enable_pin, GPIO.OUT)
GPIO.setup(in1_pin, GPIO.OUT)
```

```

GPIO.setup(in2_pin, GPIO.OUT)
pwm = GPIO.PWM(enable_pin, 500)
pwm.start(0)

def clockwise():
    GPIO.output(in1_pin, True)
    GPIO.output(in2_pin, False)

def counter_clockwise():
    GPIO.output(in1_pin, False)
    GPIO.output(in2_pin, True)

while True:
    cmd = raw_input("Command, f/r 0..9, E.g. f5 :")
    direction = cmd[0]
    if direction == "f":
        clockwise()
    else:
        counter_clockwise()
    speed = int(cmd[1]) * 10
    pwm.ChangeDutyCycle(speed)

```

进一步探讨

在考察该程序的运作机制之前，你首先需要详细了解 H-Bridge 的工作原理。

图 11-11 给出了它的工作原理图，注意这里使用的是开关，而不是晶体管或芯片。

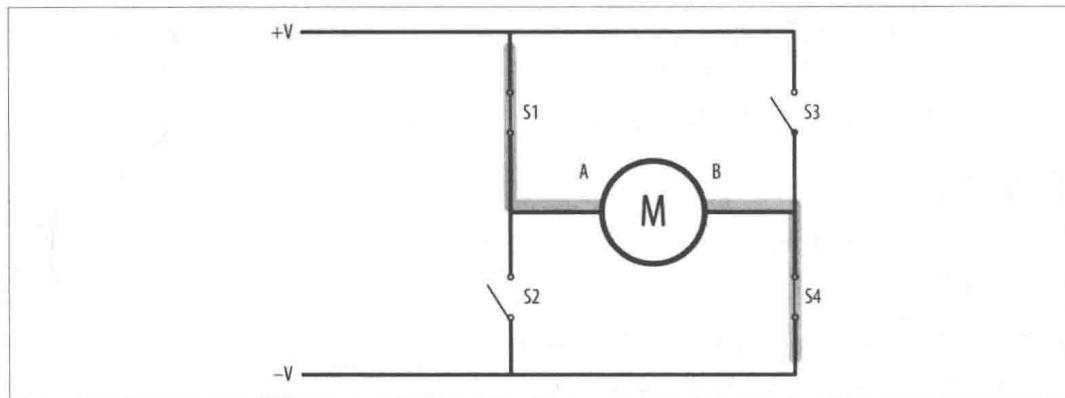


图 11-11 H-Bridge

通过逆转电机的磁极，H-Bridge 就能反转电机的转动方向。

在图 11-11 中，S1 和 S4 是闭合的，而 S2 和 S3 则是打开的。这样就允许电流通过电机，同时让 A 端为正极，B 端为负极。如果我们反转开关，此时 S2 和 S3 关闭，同时 S1 和 S4 打开，那么 B 端将变成正极，A 端将变为负极，从而使电机反向转动。

不过，你也许已经发现这个电路是非常危险的。如果由于某种原因导致 S1 和 S2 同时

闭合，那么电源的正负极就会直接相连从而形成短路。此外，如果 S3 和 S4 同时闭合的话，同样也会引起短路。

你虽然可以单独使用晶体管来制作 H-Bridge，但是不如直接使用类似 L293D 这样的 H-Bridge 来得简单。这个芯片内部实际上包含了两个 H-Bridges，也就是说，你可以使用该芯片来控制两个电机。

L293 有两个电机控制通道，每个通道都提供了 3 个控制引脚。Enable 引脚的作用是启用或禁用整个通道。在这个示例代码中，它被连接到 PWM 输出来控制电机的转速。其他两个引脚（In1 和 In2）的作用是控制电机的转动方向。你可以通过函数 `clockwise` 和 `counter_clockwise` 来了解这两个引脚的用法。

```
def clockwise():
    GPIO.output(in1_pin, True)
    GPIO.output(in2_pin, False)

def counter_clockwise():
    GPIO.output(in1_pin, False)
    GPIO.output(in2_pin, True)
```

如果 IN1 是高电平，而 IN2 是低电平，那么电机将在一个方向转动。如果这两个引脚的电平反转，那么电机就会向相反的方向转动。

除了在面包板上面使用 L293D 之外，你还可以从网店上面入手极为廉价的 PCB 上面含有 L293D 的模块，使用螺旋式接线端子直接将电机和插头引脚连接到树莓派的 GPIO 接口上面。如果你需要一个高功率的电动机控制器模块的话，那么你会发现这些模块的工作原理是一样的，只是使用的电流更大，甚至达到 20A 或更高。Pololu (<https://www.pololu.com>) 上面有花样繁多的这类电动机控制器板。

参考资料

Adafruit 的步进电机 HAT（见 11.8 节）和 RaspiRobot 板（见 11.9 节）也可以用来控制直流电动机的速度和方向。

另外，你还可以从 <http://bit.ly/18c4GKm> 查看 L293D 的参数手册，以及从 <http://bit.ly/ILHVkJ> 浏览 SparkFun 电机驱动模块产品页面。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

11.6 使用单极步进电机

面临问题

你希望使用树莓派驱动一个 5 条导线的单极步进电机。

解决方案

你可以使用 ULN2803 Darlington 驱动面包板上面的芯片。

在电机技术世界中，步进电机介于主流电机与伺服电机之间。就像常见的直流电机那样，步进电机可以连续旋转，同时你还可以通过在任意方向上每次移动一段距离的方法来精确地控制电机的位置。

为了进行本节中的实验，你需要：

- 5V 5 引脚单极步进电机；
- ULN2803 Darlington 驱动 IC；
- 面包板和跳线。

图 11-12 展示了使用 ULN2803 时的布线图。需要注意的是该芯片可以用来驱动两个步进电机。为了能够驱动第二个步进电机，需要将 GPIO 接口上面的另外 4 个控制引脚连接至 ULN2803 的 5 ~ 8 号引脚上面，同时将第二个步进电机的 4 个引脚连接至 ULN2803 上面的 11 ~ 14 号引脚。

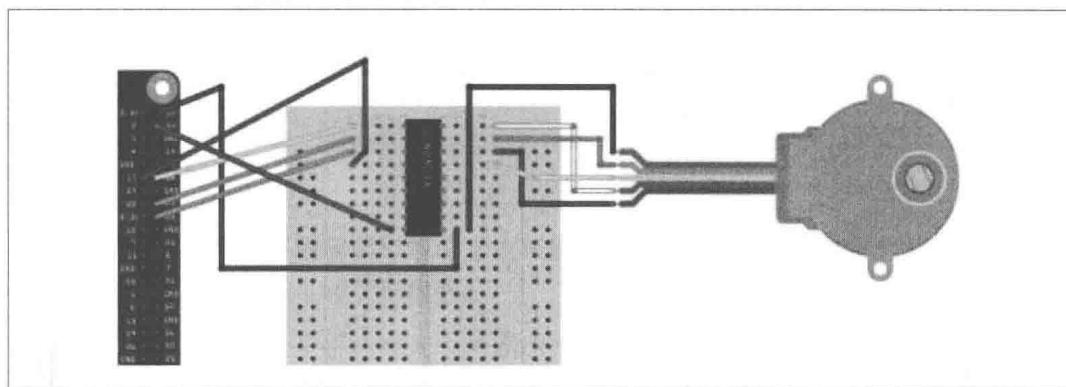


图 11-12 使用 ULN2803 控制单极步进电机

GPIO 接口提供的 5V 供电可用于小型的步进电机。

如果出现树莓派崩溃问题或者需要使用更大的步进电机的时候，那么可以使用一个单独的电源给电机供电（ULN2803 的 10 号引脚）。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 `stepper.py`。这个程序是在命令行下面使用的，所以你可以从 SSH 运行它。

如果你希望在 Python 3 下面使用这个程序的话，只需要将命令 `raw_input` 改为 `input` 即可。

```

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

coil_A_1_pin = 18
coil_A_2_pin = 23
coil_B_1_pin = 24
coil_B_2_pin = 17

GPIO.setup(coil_A_1_pin, GPIO.OUT)
GPIO.setup(coil_A_2_pin, GPIO.OUT)
GPIO.setup(coil_B_1_pin, GPIO.OUT)
GPIO.setup(coil_B_2_pin, GPIO.OUT)

forward_seq = ['1010', '0110', '0101', '1001']
reverse_seq = list(forward_seq) # to copy the list
reverse_seq.reverse()

def forward(delay, steps):
    for i in range(steps):
        for step in forward_seq:
            set_step(step)
            time.sleep(delay)

def backwards(delay, steps):
    for i in range(steps):
        for step in reverse_seq:
            set_step(step)
            time.sleep(delay)

def set_step(step):
    GPIO.output(coil_A_1_pin, step[0] == '1')
    GPIO.output(coil_A_2_pin, step[1] == '1')
    GPIO.output(coil_B_1_pin, step[2] == '1')
    GPIO.output(coil_B_2_pin, step[3] == '1')

while True:
    set_step('0000')
    delay = raw_input("Delay between steps (milliseconds)?")
    steps = raw_input("How many steps forward? ")
    forward(int(delay) / 1000.0, int(steps))
    set_step('0000')
    steps = raw_input("How many steps backwards? ")
    backwards(int(delay) / 1000.0, int(steps))

```

当你运行这个程序的时候，它会要求你输入每个步进之间的延迟时间。这个延迟可以取 2 或更大的数字。然后，还会提示输入每个方向上的步进数。

```

$ sudo python stepper.py
Delay between steps (milliseconds)?2
How many steps forward? 100

```

```
How many steps backwards? 100
Delay between steps (milliseconds)?10
How many steps forward? 50
How many steps backwards? 50
Delay between steps (milliseconds)?
```

进一步探讨

步进电动机使用一个齿形转子和电磁铁，以使电机每次可以转动一个步进角度（见图 11-13）。需要注意的是导线的颜色会有所变化。

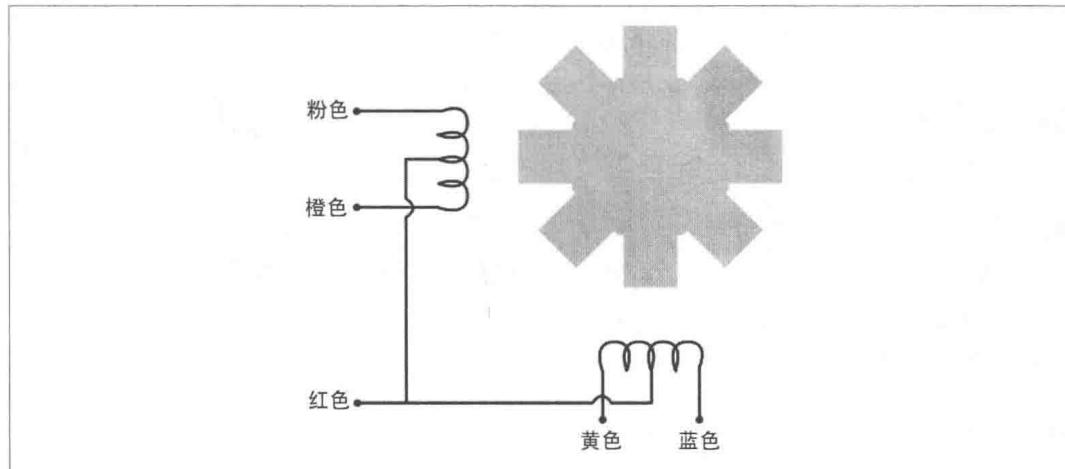


图 11-13 步进电机

通过一定的顺序给线圈通电就可以驱动电机旋转。能够在 360 度范围内旋转的步进电机的步进数，实际上就是转子的齿数。

在示例代码中，使用一个字符串列表来表示一个单独步进中的 4 个通电阶段。

```
forward_seq = ['1010', '0110', '0101', '1001']
```

电机相反方向的旋转序列正好就是前进序列的反转。

你可以在自己的程序中使用 forward 和 backward 函数来驱动电机向前步进或向后步进。这两个函数的第一个参数表示在一个步进序列中的各个部分之间的延迟时间，这里以毫秒为单位。这个参数的最小值要视你使用的电机而定，如果取值太小的话，电机就不会转动。一般情况下，可以使用 2 毫秒，或更大的延迟。第二个参数是步进数。

```
def forward(delay, steps):
    for i in range(steps):
        for step in forward_seq:
            set_step(step)
            time.sleep(delay)
```

函数 `forward` 中嵌套了两个 `for` 循环。外层的循环根据步进数进行循环，内部的循环根据电机转动序列进行迭代，对序列中的每个元素调用 `setStep`。

```
def set_step(step):
    GPIO.output(coil_A_1_pin, step[0] == '1')
    GPIO.output(coil_A_2_pin, step[1] == '1')
    GPIO.output(coil_B_1_pin, step[2] == '1')
    GPIO.output(coil_B_2_pin, step[3] == '1')
```

函数 `set_step` 会将每个控制引脚设置为高电平或低电平，具体取决于参数指定的模式。

主循环会将在正向和反向之间的步进设为 0000，以便在电机没有实际移动的时候，将所有的输出都设置为 0。否则，某个线圈可能还在通电，从而导致电机不必要的电源浪费。

参考资料

如果你使用的是 4 线双极步进电机，请参考 11.7 节。

关于步进电机的进一步信息，例如各种类型和工作原理等，可以参考 Wikipedia (http://en.wikipedia.org/wiki/Stepper_motor)，这里有非常不错的电机驱动模式的动画演示。

关于伺服电机更详细的用法介绍，请参考 11.1 节；至于控制直流电机的方法，请参考 7.16 节和 7.18 节。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

11.7 使用双极步进电机

面临问题

你希望使用树莓派驱动 4 导线双极步进电机。

解决方案

你可以使用一个 L293D H-Bridge 驱动芯片。为了驱动双极步进电机，你需要使用一个 H-Bridge，其原因正如单词 `bipolar` 所暗示的那样，通过绕组的电流方向需要反转，有点像在两个方向上面驱动一个直流电机（见 11.5 节）。

为了进行本节中的实验，你需要：

- 12V 4 引脚双极步进电机；
- L293D H-Bridge IC；
- 面包板和跳线。

这里使用的电机电压是 12V 的，比前面的单极步进电机的要大一些。所以，电机本身

的电源要从外部供电，而非从树莓派供电。布线请参考图 11-14。

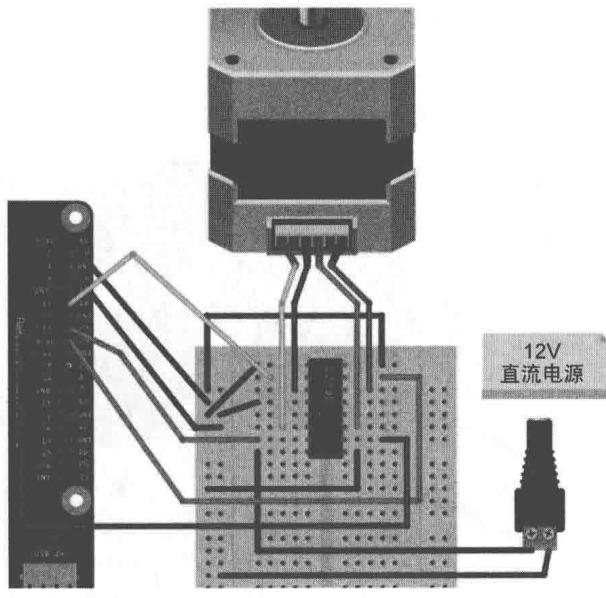


图 11-14 使用 L293D 控制双极步进电机

进一步探讨

你可以原封不动地使用 `stepper.py` 程序来控制这个步进电机（见 11.6 节）。该设计同时利用了 L293D 的两个 H-Bridge，所以，你需要为每个待控制的电机提供一个这样的芯片。

参考资料

如果你使用的是 5 线单极步进电机，那么请参考 11.6 节。

关于步进电机的进一步信息，例如各种类型和工作原理等，可以参考 Wikipedia (http://en.wikipedia.org/wiki/Stepper_motor)，这里有非常不错的电机驱动模式的动画演示。关于伺服电机更详细的用法介绍，请参考 11.1 节；至于控制直流电机的方法，请参考 7.16 节和 7.18 节。

关于在树莓派上使用面包板和跳线的详细介绍，请参考 9.8 节。

此外，你还可以使用 RaspiRobot 板来驱动步进电机（见 11.9 节）。

11.8 利用步进电机 HAT 驱动双极步进电机

面临问题

你想利用单个接口板来控制多个双极步进电机。

解决方案

你可以使用 Adafruit 的步进电机 HAT (见图 11-15)。

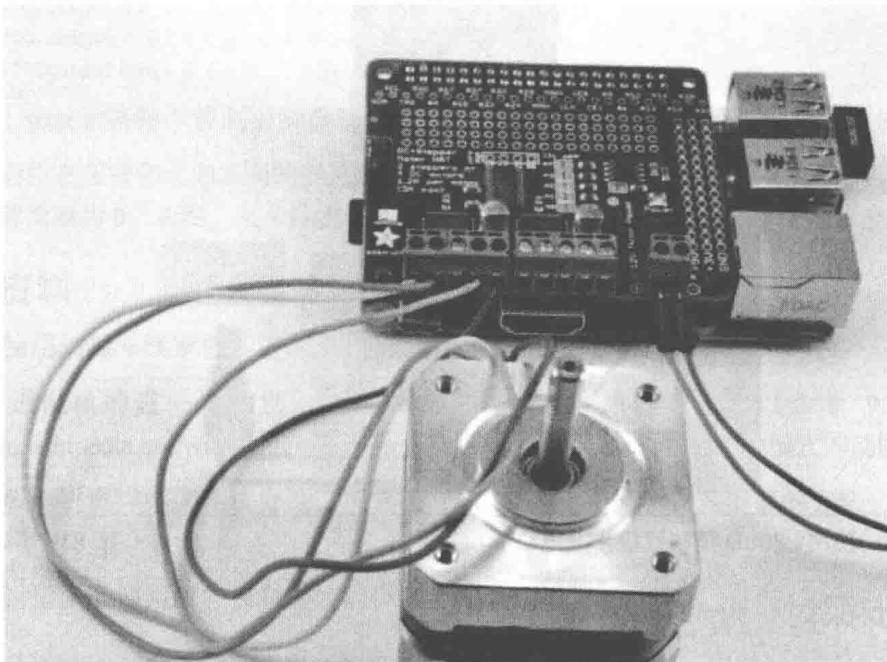


图 11-15 使用 Adafruit 的步进电机 HAT 控制双极步进电机

这个电路板可以驱动两个双极步进电机。图 11-15 展示的是带有一个双极步进电机的电路板，其中一个线圈连接到 M1 端子，另一个线圈连接到 M2 端子。电机是单独由图 11-15 中右边的螺旋式接线端子来供电的。这个 HAT 使用了 I2C，所以，请务必确保已经启用了 I2C (见 9.3 节)。

为了让这个 HAT 更加便于使用，Adafruit 提供了一个相应的 Python 库。要想安装这个库，可以使用如下所示的命令。

```
$ git clone https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library.git  
$ sudo apt-get install python-dev  
$ cd Adafruit-Motor-HAT-Python-Library/  
$ sudo python setup.py install
```

这个库还提供了一些示例代码，所以，你可以切换到相应的目录，并通过下列命令来运行一个简单的示例。

```
$ cd examples  
$ sudo python StepperTest.py
```



I2C Busses

如果你按照 9.20 节介绍的方法创建了自己的 HAT 并启用了 I2C 总线 0 的话，那么就需要返回头来修改/boot/conig.txt，因为 Adafruit 会自动检测所用的 I2C 总线，如果总线 0 已经启用的话，它会检测到一个错误。

在/boot/conig.txt 中，删除或注释掉下面所示的命令行。

```
dtparam=i2c_vc=on
```

完成上述工作之后，需要重新启动树莓派。

进一步探讨

当你运行程序时，电机就会开始运转，并且该程序会围绕 4 种不同的步进模式进行循环。你可以尝试下面的例子来控制步进电机。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 stepper_hat.py 。由于这个程序以命令行方式运行，所以你可以通过 SSH 来运行它。

```
from Adafruit_MotorHAT import Adafruit_MotorHAT
import time

HAT = Adafruit_MotorHAT
stepper_hat = Adafruit_MotorHAT()

stepper = stepper_hat.getStepper(200, 1) # 200 steps/rev, port 1 (M1, M2)

try:
    while True:
        speed = input("Enter stepper speed (rpm) ")
        stepper.setSpeed(speed)
        steps_forward = input("Steps forward ")
        stepper.step(steps_forward, HAT.FORWARD, HAT.SINGLE)
        steps_forward = input("Steps reverse ")
        stepper.step(steps_forward, HAT.BACKWARD, HAT.SINGLE)

finally:
    print("cleaning up")
    stepper_hat.getMotor(1).run(HAT.RELEASE)
```

在导入 Adafruit_MotorHAT 模块之后，将其重新命名为 HAT，并且在后续的代码中用 HAT 来指代这个冗长的模块名，从而使代码变得更加简洁。

这里使用了 try/finally 语句块确保使用 Ctrl-C 命令来关闭程序的时候会给线圈断电。

参考资料

关于 HAT 标准的讨论以及打造自己的 HAT 的方法，请参考 9.20 节。

关于 HAT 的用法以及相应库的介绍, 请参考 <https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/>。

要想使用 L293D 来控制步进电机, 请参考 11.7 节; 关于 RaspiRobot 板的介绍, 请参考 11.9 节。

11.9 使用 RaspiRobot 板驱动双极步进电机



请务必在 <http://razzpisampler.oreilly.com> 上观看本节相关的视频。

面临问题

你想利用同时为树莓派和电机供电的电源来控制一个双极步进电机。

解决方案

你可以使用第 3 版的 RaspiRobot 板。

RaspiRobot 板直接使用它的螺旋式接线端子作为电机电源, 并将同样的电源降至 5V 后驱动树莓派。所以在本例中, 12V 的电源将同时为 12V 的步进电机和树莓派供电。



如果你使用的是 RaspiRobot 板的早期版本 (版本 1 或版本 2), 那么当使用 USB 连接给 RaspiRobot 板供电的时候, 就不能再同时用它来给树莓派供电。

但是对于这个电路板的第 3 版本来说, 就不存在这种问题。

将步进电机和电源连接到 RaspiRobot 板上面, 具体如图 11-16 所示。Adafruit 12V 步进电机的导线, 按照从左到右的顺序, 其颜色分别为: 黄色、红色、灰色和绿色。

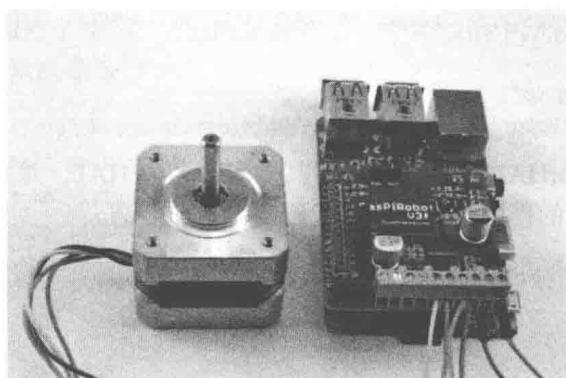


图 11-16 使用 RaspiRobot 板控制双极步进电机

为了运行这个程序，你需要首先安装用于 RaspiRobot V3 的代码库，具体命令如下所示。

```
$ git clone https://github.com/simonmonk/raspirobotboard3.git  
$ cd raspirobotboard3/python  
$ sudo python setup.py install
```

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 `stepper_rrb.py`。这个程序是在命令行下面使用的，所以你可以从 SSH 运行它。

如果你使用的是 Python 3，那么只要将 `raw_input` 命令改为 `input` 即可。

```
from rrb3 import *  
import time  
  
rr = RRB3(12.0, 12.0) # battery, motor  
  
try:  
    while True:  
        delay = raw_input("Delay between steps (milliseconds)?")  
        steps = raw_input("How many steps forward? ")  
        rr.step_forward(int(delay) / 1000.0, int(steps))  
        steps = raw_input("How many steps backwards? ")  
        rr.step_reverse(int(delay) / 1000.0, int(steps))  
  
finally:  
    GPIO.cleanup()
```

进一步探讨

你可能已经发现了，“Delay between steps” 有一个最小值，当低于这个值的时候，电机只会抖，但是不转。

参考资料

关于 RaspiRobot 板及其他使用该板的项目的完整文档，请访问 RaspiRobot 网站（ <http://www.raspirobot.com> ）。

要想使用面包板上的 L293D 驱动步进电机，请参考 11.7 节。

11.10 打造一款简单的机器人小车

面临问题

你希望将树莓派用作一个机器人小车的控制器。

解决方案

你可以把 RaspiRobot 板 V3 用作树莓派的接口板来控制两个电机和机器人底盘套件，

如 Magician chassis。

为了进行本节中的实验，你需要：

- RaspiRobot 板 V3；
- 诸如 Magician chassis 之类的减速机底盘；
- 6 节 5 号电池仓；
- USB WiFi 适配器。

此外，你也可以直接购买提供了打造这种小车所需全部元件的套装，比如 MonkMakes RaspiRobot rover Kit。

为了制作机器人小车，第一步就是组装底盘。大部分廉价的减速机底盘都提供了可容纳 4 节 5 号电池的电池仓，但是要想给树莓派供电的话，实际上需要的是一个可容纳 6 节电池的电池仓。所以，当你浏览底盘说明书中关于电池仓的部分时，请使用一个 6 节 5 号电池的电池箱。

布线如图 11-17 所示。

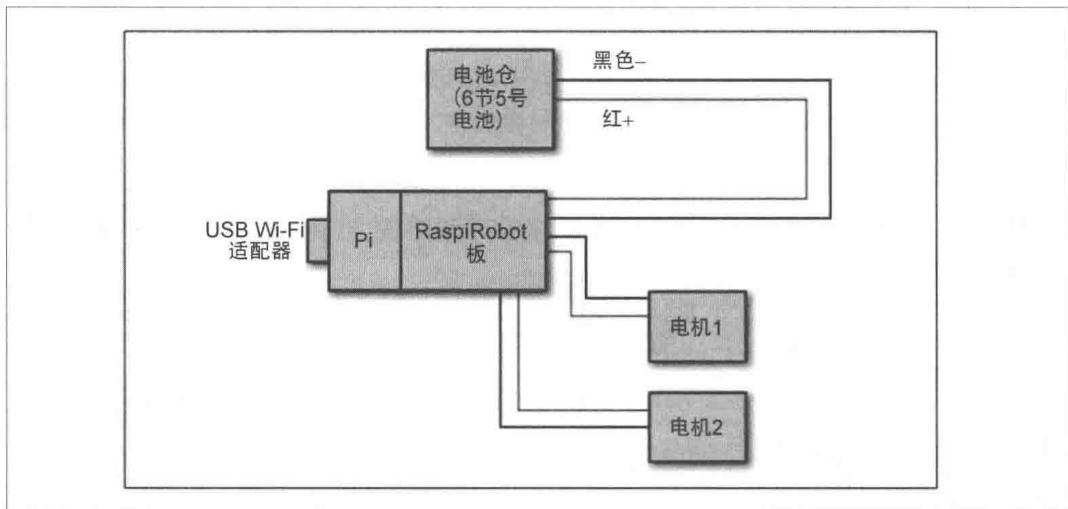


图 11-17 机器人小车布线图

电池组给 RaspiRobot 板供电，而后者又给树莓派提供 5V 电源，所以，实际上只需要一个电源就可以了。

制作完成后的小车如图 11-18 所示，车身前端安装了一个测距仪，以用来测量距离。

为了驱动这个机器人小车，你需要使用一个控制程序，以便可以通过笔记本或者通过 SSH 连接到你的树莓派的计算机上的方向键来操纵小车。为此，你可以按照 2.5 节和 2.7 节介绍的方法配置树莓派来使用 Wi-Fi 和 SSH。

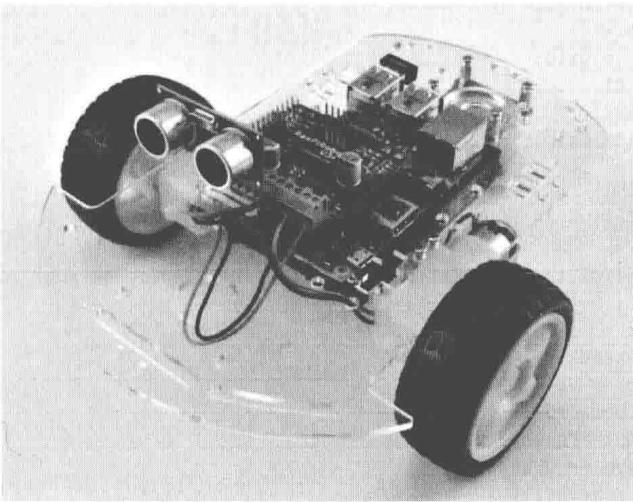


图 11-18 完成后的机器人

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 rover.py 。

```
from rrb3 import *
import sys
import tty
import termios

rr = RRB3(9.0, 6.0) # battery, motor

UP = 0
DOWN = 1
RIGHT = 2
LEFT = 3
print("Use the arrow keys to move the robot")
print("Press Ctrl-C to quit the program")

# These functions allow the program to read your keyboard
def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    if ch == '0x03':
        raise KeyboardInterrupt
    return ch

def readkey(getchar_fn=None):
```

```

getchar = getchar_fn or readchar
c1 = getchar()
if ord(c1) != 0x1b:
    return c1
c2 = getchar()
if ord(c2) != 0x5b:
    return c1
c3 = getchar()
return ord(c3) - 65 # 0=Up, 1=Down, 2=Right, 3=Left arrows

# This will control the movement of your robot and display on your screen
try:
    while True:
        keyp = readkey()
        if keyp == UP:
            rr.forward(1)
            print 'forward'
        elif keyp == DOWN:
            rr.reverse(1)
            print 'backward'
        elif keyp == RIGHT:
            rr.right(1)
            print 'clockwise'
        elif keyp == LEFT:
            rr.left(1)
            print 'anti clockwise'
        elif keyp == LEFT:
            rr.left(1)
            print 'anti clockwise'
        elif ord(keyp) == 3:
            break

except KeyboardInterrupt:
    GPIO.cleanup()

```

为了获得中断按键的功能，这个程序使用了 termios 库以及 readchar 和 readkey 这两个函数。

在导入命令之后，新建了一个 RRB3 的实例。它需要用到两个参数，即电池组电压和电机电压（就本例而言分别是 9V 和 6V）。如果你的底盘的电机使用了不同的电压，那么请自行修改第二个参数。

主循环的作用只是检查按键事件，然后向 RRB3 库发送相应的向前、向后、向左和向右的指令。

进一步探讨

如果你继续为这个小车添加更多的外设的话，它会变得更加好玩。例如，如果为小车加入一个摄像头并配置一个网络流媒体的话，它就变成一个移动监控器了（见 4.5 节）。

RRB3 库还支持 HC-SR04 测距仪，该设备可以安装到 RaspiRobot 板 V3 的插座中。你可以使用这个测距仪来检查障碍物，同时，RRB3 库也为它提供了相应的示例程序。关于 RaspiRobot 板和 RRB3 库的更多资料，请访问 GitHub (<https://github.com/simonmonk/raspirobotboard3>)。

数字输入

12.0 引言

在本章中，你将学习使用数字输入的有关内容，例如开关和键盘。同时，本章也会介绍某些提供了数字输出的模块，这些输出可以连接树莓派的 GPIO 输入。

大部分的示例都会用到免焊面包板和跳线（见 9.8 节）。

12.1 连接按钮开关



请务必观看在 <http://razzpisampler.oreilly.com> 上与本节有关的视频。

面临问题

你希望在树莓派上连接一个开关，当按下它时，就会运行某些 Python 代码。

解决方案

将一个开关连接到 GPIO 引脚，并在你的 Python 程序中使用 RPi.GPIO 库侦测按钮按下动作。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 轻触按钮开关。

图 12-1 展示了如何使用面包板和跳线来连接轻触按钮开关。

除了使用面包板和轻触按钮开关之外，你还可以使用 Squid Button（见图 12-2）。由于这种按钮开关的一端焊有母头的引线，所以可以直接连接到 GPIO 接口上面（见 9.11 节）。

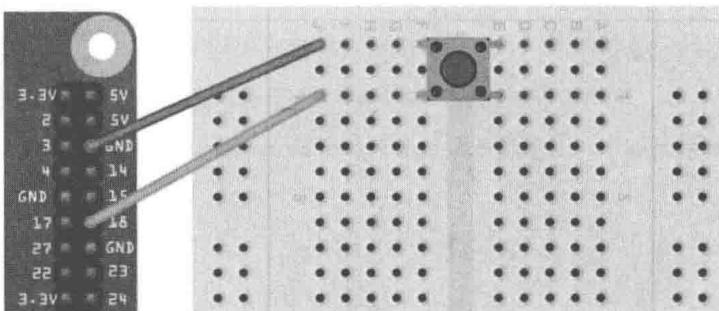


图 12-1 为树莓派连接按钮开关

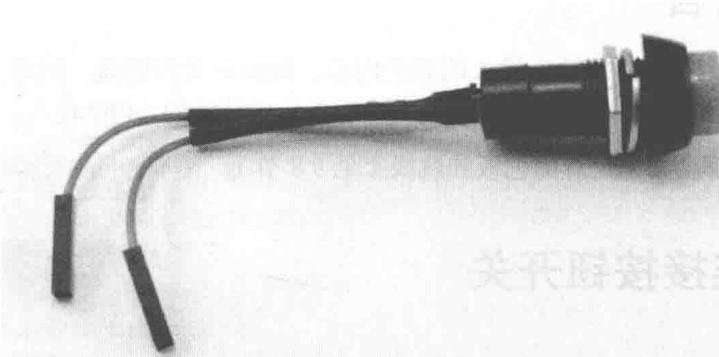


图 12-2 Squid Button

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 switch.py 。

当按钮按下时，这个示例代码将会显示一则消息。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)
while True:
    input_state = GPIO.input(18)
    if input_state == False:
        print('Button Pressed')
        time.sleep(0.2)
```

为了运行这个程序，你需要具有超级用户权限，所以必须使用如下所示的命令。

```
pi@raspberrypi ~ $ sudo python switch.py  
Button Pressed  
Button Pressed  
Button Pressed  
Button Pressed
```

进一步探讨

你也许已经注意到了开关的连接方式，即当开关按下时，它会将配置为输入的 18 号引脚连接至 GND。这个输入引脚通常在 GPIO.setup 中通过可选参数 pull_up_down=GPIO.PUD_UP 配置为上拉至 3.3V。这就意味着如果在按钮被按下的时候使用 GPIO.input 读取输入值，将会返回 False。这有点违反直觉。

每个 GPIO 引脚都有一个可软件配置的上拉电阻和下拉电阻。当将一个 GPIO 引脚用作输入的时候，你可以在 GPIO.setup 中使用可选参数 pull_up_down 配置这些电阻，以启用某个电阻或者一个电阻也不启用。如果忽略这个参数的话，那就不会启用任何电阻。这将导致输入悬空 (input floating)，即它的值不是稳定的，而是随着受电噪声的影响程度而在高电平和低电平之间漂移。

如果将其设为 GPIO.PUD_UP，就会启用上拉电阻；如果将其设为 GPIO.PUD_DOWN，将会启用下拉电阻。你也许希望按钮开关只有两个连接，即只能表示打开和关闭。虽然某些轻触按钮开关确实只有两个连接，但是大部分都是 4 个连接。图 12-3 显示这些连接是如何安排的。

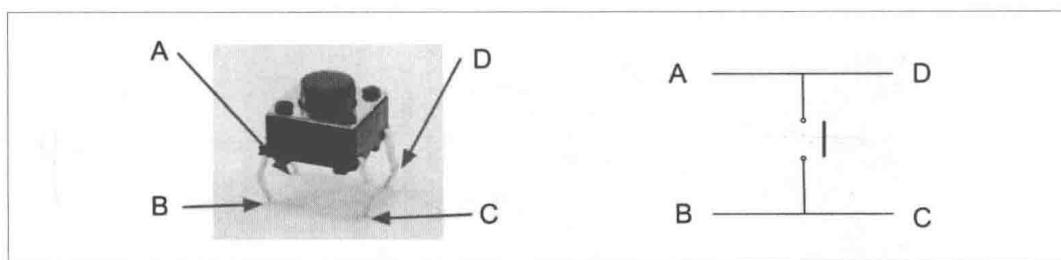


图 12-3 轻触按钮开关

实际上，只有两个是真正的电气连接，因为在这个开关内部，引脚 B 和引脚 C 连在一起，引脚 A 和引脚 D 连在一起。

参考资料

关于在树莓派上使用面包板和跳线的详细介绍，请参见 9.8 节。

关于利用开关触发中断的内容，请参见 10.13 节。

为了消除开关的抖动，请参见 12.5 节。

若要使用外部上拉电阻和下拉电阻，请参见 12.6 节。

12.2 通过按钮开关切换开关状态

面临问题

你希望通过开关按钮来打开或关闭某些元件，即每次按下开关的时候，将切换开关状态。

解决方案

你可以把按钮的最后一次状态记录下来，并在每次按下按钮时反转这个值。

下面的示例代码会在你按下开关时切换 LED 的开关状态。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 轻触按钮开关；
- LED；
- 470Ω 电阻。

图 12-4 展示了通过面包板和跳线连接轻触按钮开关与 LED 的方法。

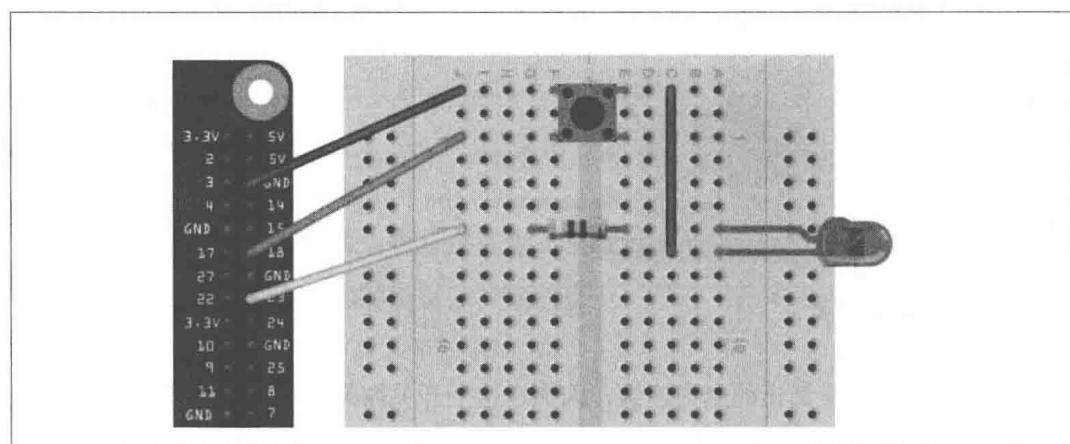


图 12-4 将按钮开关和 LED 连接至树莓派

除了连接树莓派到面包板的 male-to-female 跳线之外，你还需要一个 male-to-male 跳线或实心焊丝。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例

代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 switch_on_of.py。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

switch_pin = 18
led_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(led_pin, GPIO.OUT)

led_state = False
old_input_state = True # pulled-up

while True:
    new_input_state = GPIO.input(switch_pin)
    if new_input_state == False and old_input_state == True:
        led_state = not led_state
    old_input_state = new_input_state
    GPIO.output(led_pin, led_state)
```

进一步探讨

变量 led_state 用来保存 LED 的当前状态（True 表示打开，False 表示关闭）。每当按钮按下时，下面这行代码就会运行。

```
led_state = not led_state
```

命令 not 的作用是对 led_state 的值进行取反，因此如果 led_state 为 True，那么它就将变成 False，反之亦然。

变量 old_input_state 用来记录按钮的状态，所以只有输入状态从 True（开关未被按下）变为 False（按钮被按下）时才会定义为按钮被按下。

参考资料

你会发现，有时候按下按钮时，并未切换 LED 的开关状态，这是开关抖动引起的。你可以在 12.5 节中发现其他避免开关抖动的方法。

12.3 使用双位拨动开关或滑动开关

面临问题

你希望可以将双位拨动开关或滑动开关连接到树莓派，并能够通过 Python 程序获取开关状态。

解决方案

你可以像使用轻触按钮开关（见 12.1 节）一样来使用这两种开关：只需将中央和一端的触点连接即可（见图 12-5）。

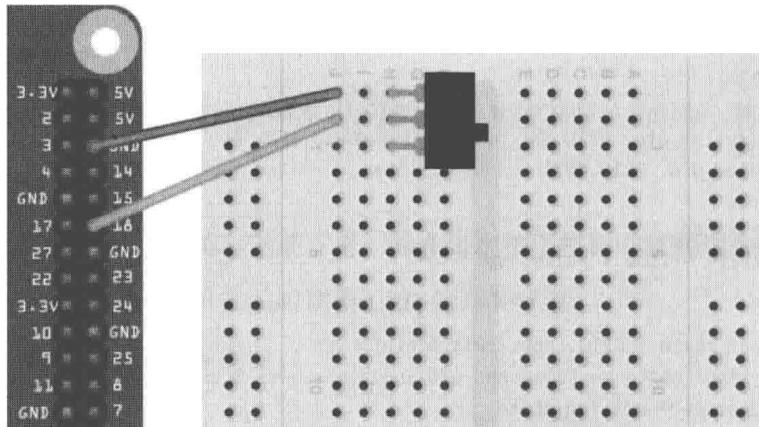


图 12-5 将滑动开关连接到树莓派

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 微型拨动开关或滑动开关。

这种布局完全可以使用 12.1 节中的代码。

进一步探讨

这类滑动开关是非常有用的，因为你可以一眼看出它们当前的状态，而无需 LED 之类的额外指示装置。不过，这类开关不如轻触按钮开关耐用，并且价格也更贵，而轻触按钮开关在消费电子领域已经越来越流行，因为它们外面可以安装一个更加美观的塑料按钮。

参考资料

为了使用带有中间位置的三位开关，可以参见 12.4 节。

12.4 使用三位拨动开关或滑动开关

面临问题

你希望可以将三位拨动开关（中间位置关闭）连接到树莓派，并能够通过 Python 程序

获取开关状态。

解决方案

你可以按照图 12-6 所示的方式将开关连接到 GPIO 的两个引脚上面，并通过 Python 程序使用 RPi.GPIO 库来检测开关的状态。

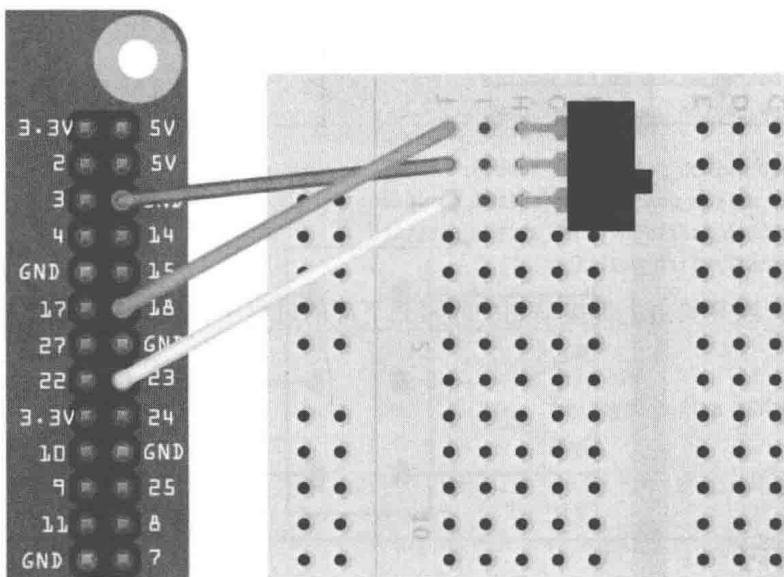


图 12-6 连接三位开关到树莓派

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 微型中间关闭三位拨动开关。

开关的公共（中间的）连接要接地，位于开关两端的连接要与启用了内部上拉电阻的 GPIO 引脚相连。打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，就本例来说，该程序名为 switch_2_pos.py。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

top_input = 18
bottom_input = 23
GPIO.setup(top_input, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```
GPIO.setup(bottom_input, GPIO.IN, pull_up_down=GPIO.PUD_UP)

switch_position = "unknown"

while True:

    top_state = GPIO.input(top_input)
    bottom_state = GPIO.input(bottom_input)
    new_switch_position = "unknown"
    if top_state == False:
        new_switch_position = "up"
    elif bottom_state == False:
        new_switch_position = "down"
    else:
        new_switch_position = "center"
    if new_switch_position != switch_position:
        switch_position = new_switch_position
        print(switch_position)
```

运行该程序，当开关由上部拨动到中间再到下部的过程中，开关位置每次改变时都会被输出。

```
$ sudo python switch_3_pos.py
up
center
down
```

进一步探讨

这个程序设置了两个启用了上拉电阻的输入，变量 `switch_position` 用于记录开关的当前状态。在循环内部，会读取两个 GPIO 输入，`if`、`elif` 和 `else` 结构的 3 个条件可以确定开关的位置，并将相应的值赋给变量 `new_switch_position`。如果这个值与之前的值不同，那么就会打印输出开关的状态。

将来你会发现各种不同类型的拨动开关，其中一些类型为 DPDT、SPDT、SPST 或者 SPST、点放等。这些字母的含义如下所示。

- D：双
- S：单
- P：刀
- T：掷

DPDT 表示双刀双掷开关。这里的刀表示可以通过一个机械杆控制的独立开关触点的数目。所以，一个双刀开关可以独立控制两个器件的开和关。一个单掷开关只能打开或关闭一个独立触点（或两个触点，如果是双刀的话）。不过，一个双掷开关可以将一个共用触点连接到另外两个触点中的一个上面。

图 12-7 展示了常见的开关类型。

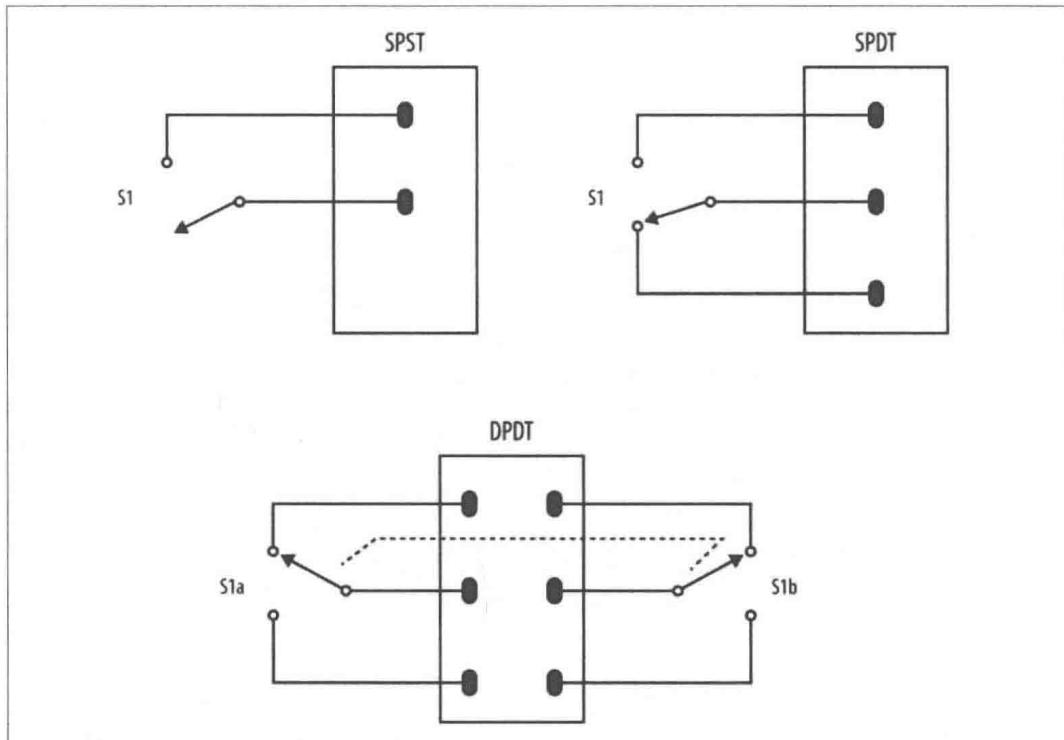


图 12-7 拨动开关的类型参考资料

如果想进一步了解 if 语句的工作机制，请参见 5.18 节。关于最简单的开关示例，请参考 12.1 节。

12.5 按钮去抖

面临问题

有时候按下开关的按钮时，你所预期的动作会不止发生一次，因为开关触点会发抖。在这种情况下，你可能希望编写代码来消除开关抖动的影响。

解决方案

这个问题有多种解决方案。如果你想考察这些方案的话，可以按照 12.2 节中的介绍搭建一个面包板配置。

这个示例程序的原始代码中没有任何去抖功能，代码如下所示。

```
import RPi.GPIO as GPIO
```

```

import time

GPIO.setmode(GPIO.BCM)

switch_pin = 18
led_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(led_pin, GPIO.OUT)

led_state = False
old_input_state = True # pulled-up

while True:
    new_input_state = GPIO.input(switch_pin)
    if new_input_state == False and old_input_state == True:
        led_state = not led_state
    old_input_state = new_input_state
    GPIO.output(led_pin, led_state)

```

问题在于如果开关触点发生抖动，则就像连续多次切换开关。如果抖动发生了奇数次，最后结果看起来还好。但是，如果抖动发生了偶数次，则会出现本来是要打开 LED，最后却变成了关闭它的情况。

在开关被按下之后的短时间内，所有的变化都应该忽略不计，直到开关抖动结束为止。一种快速而简单的去抖方法是通过添加 `time.sleep` 命令，在侦测到按钮按下之后休眠一小会，比如 0.2 秒。严格讲，这个延迟可能有些大了。你会发现，这个延迟可以进一步减小。

```

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

switch_pin = 18
led_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(led_pin, GPIO.OUT)

led_state = False
old_input_state = True # pulled-up

while True:
    new_input_state = GPIO.input(switch_pin)
    if new_input_state == False and old_input_state == True:
        led_state = not led_state
        time.sleep(0.2)
    old_input_state = new_input_state
    GPIO.output(led_pin, led_state)

```

进一步探讨

虽然这种解决方案在大多数情况下都是可行的，但是，你可以进一步考虑使用支持中断的开关引脚（见 10.13 节）。

大部分开关都会发生抖动现象，并且某些开关上面的抖动现象会非常严重，具体如图 12-8 中的示波器描述所示。

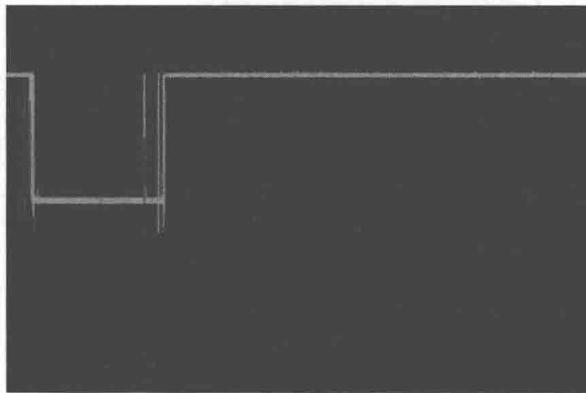


图 12-8 廉价开关的触点抖动

你可以看到，无论是开关闭合时，还是开关释放时，都会发生抖动现象。当然，大部分开关的表现不会像这个这么糟糕。

参考资料

关于连接按钮的基础知识，请参考 12.1 节。

Squid Button 库也提供了按钮消抖功能，它使用的方法与本节介绍的非常类似。关于 Squid Button 的详细介绍，请参考 9.11 节。

12.6 使用外部上拉电阻

面临问题

你想使用一根长导线来连接树莓派和开关，但是这样的话，会导致从输入引脚收到一些错误读数。

解决方案

内部上拉电阻实际上是非常弱的（大约 $40k\Omega$ ）。如果你的开关使用了较长的导线或者工作于电噪声环境中的话，在数字输入上面可能出现假触发现象。为了解决这个问题，

你需要关闭内部的上拉电阻和下拉电阻，然后使用一个外部的上拉电阻。

图 12-9 展示了外部上拉电阻的使用方法。

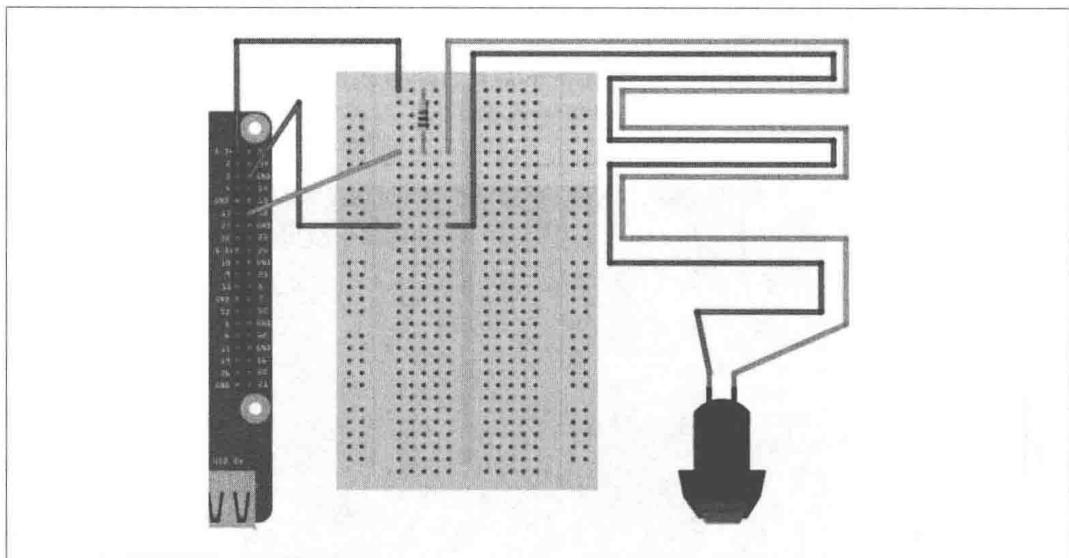


图 12-9 使用外部上拉电阻

要想测试这个硬件的工作情况，你可以使用 12.1 节中的 switch.py 程序。

进一步探讨

电阻的阻值越低，开关的连接距离就能越远。但是，当你按下按钮时，一个 3.3V 的电流会从电阻到地。一个 100Ω 电阻会消耗掉 $3.3V/100\Omega=33mA$ 的电流。这个电流在 3.3V 提供 50mA 或树莓派 1 的安全限制之内，所以，如果你使用的是老版树莓派的话，就不要使用阻值更低的电阻了。如果你使用的是较新的 GPIO 提供 40 个引脚的树莓派的话，甚至可以将电阻的阻值降低至 47Ω 。

在任何情况下，一个 $1k\Omega$ 电阻都能提供一个足够远的允许连接距离，并且不会引起任何问题。

参考资料

关于连接按钮的基础知识，请参考 12.1 节。

12.7 使用旋转（正交）编码器

面临问题

你希望使用旋转编码器检测旋转。

解决方案

你可以使用一个旋转器(正交编码器),并将其连接到两个GPIO引脚上面,具体如图12-10所示。

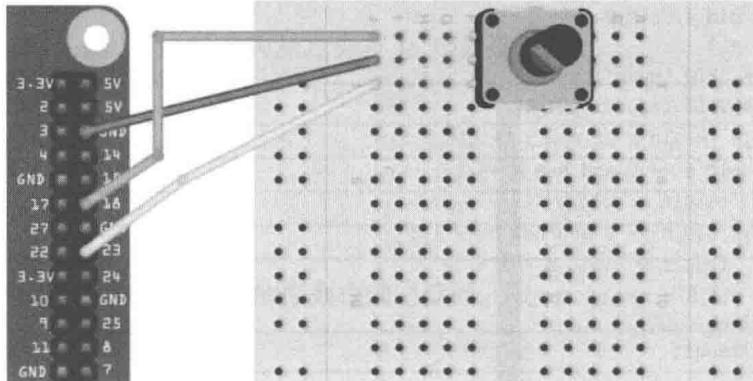


图 12-10 连接旋转编码器

为了进行本节中的实验,你需要:

- 面包板和跳线;
- 旋转编码器。

这种类型的编码器被称为正交编码器,其行为类似于一对开关。并且,两个开关的开合序列与旋转编码器的转轴的转动一样,最终决定了旋转的方向。

对于上面展示的旋转编码器,中央导线是公共连接线,另外两边还有导线A和导线B。当然,并非所有的旋转编码器都采用这种布局,所以具体使用过程中,你要仔细检查所用编码器数据手册中关于引脚的具体介绍。此外,由于许多旋转编码器还包含一个按钮开关,并且开关具有一对单独的触点,这使得人们更加容易混淆。

打开一个编辑器(nano或者IDLE),并粘贴如下所示的代码。就像本书中的其他示例代码一样,你也可以从本书网站(<http://www.raspberrypicookbook.com>)的代码区下载相应的程序文件,就本例来说,该程序名为rotary_encoder.py。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

input_A = 18
input_B = 23

GPIO.setup(input_A, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```

GPIO.setup(input_B, GPIO.IN, pull_up_down=GPIO.PUD_UP)

old_a = True
old_b = True

def get_encoder_turn():
    # return -1, 0, or +1
    global old_a, old_b
    result = 0
    new_a = GPIO.input(input_A)
    new_b = GPIO.input(input_B)
    if new_a != old_a or new_b != old_b :
        if old_a == 0 and new_a == 1 :
            result = (old_b * 2 - 1)
        elif old_b == 0 and new_b == 1 :
            result = -(old_a * 2 - 1)
        old_a, old_b = new_a, new_b
    time.sleep(0.001)
    return result

x = 0

while True:
    change = get_encoder_turn()
    if change != 0 :
        x = x + change
        print(x)

```

当旋转编码器顺时针旋转时，这个测试程序会递增，每次加 1；当旋转编码器逆时针旋转时，它会递减，每次减 1。

```

pi@raspberrypi ~ $ sudo python rotary_encoder.py
1
2
3
4
5
6
7
8
9
10
9
8
7
6
5
4

```

进一步探讨

图 12-11 展示了从两个触点（A 和 B）获取的脉冲序列。你从中可以发现，模式本身以 4 个脉冲步进为周期进行重复，由此得名正交编码器。

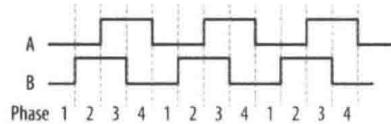


图 12-11 正交编码器工作原理

当顺时针旋转时（图 12-11 所示为从左到右），序列是：

相位	A	B
1	0	0
2	0	1
3	1	1
4	1	0

当向相反方向转动的时候，相位的序列也随之反转。

相位	A	B
4	1	0
3	1	1
2	0	1
1	0	0

上面的 Python 程序在函数 `get_encoder_turn` 中实现了用于检测旋转方向的算法。当顺时针方向转动时，该函数返回 1；当逆时针方向转动时，该函数返回 -1；当没有发生转动的时候，该函数返回 0。这个程序使用了两个全局变量：`old_a` 和 `old_b`，分别用来保存开关 A 和 B 的前一个状态。通过与新的读取值进行比较后，就可以检测（使用了非常巧妙的逻辑）到编码器的转向了。

为了保证下一次采样不会发生在前一次采样后很短的时间内，我们将休眠周期设为 1 毫秒；否则的话，两次采样之间的过渡时间过短会导致错误的读数。

无论你以多么快的速度转动这个旋转编码器的把手，该测试程序都能可靠地工作。但是，应该尽量避免在循环中执行耗时的操作，否则的话，就可能丢失一些旋转步进。

参考资料

此外，你也可以使用可变电阻和阶跃响应法（见 13.1 节）或者使用数模转换器（见 13.5 节）来检测把手的旋转位置。

12.8 使用数字键盘

面临问题

你想通过数字键盘与树莓派进行交互。

解决方案

数字键盘是以行和列的形式布局的，在行和列的每个交叉处都有一个按键开关。为了弄清哪个按键被按下，首先要将所有行和列连接到树莓派的 GPIO 引脚上面。因此，对于一个 4×3 的数字键盘来说，需要用到 4+3 个引脚。通过逐次扫描每列（将其设为输出高电平）并读取每行输入的值，就可以检测到哪个键被按下了（如果有的话）。

需要注意的是不同的数字键盘的引出线的区别会很大。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 4×3 数字键盘；
- 7 个公接头引脚。

图 12-12 展示了使用 SparkFun 数字键盘的项目的布线图。该数字键盘没有提供插头引脚，所以必须将其焊接到数字键盘上面。

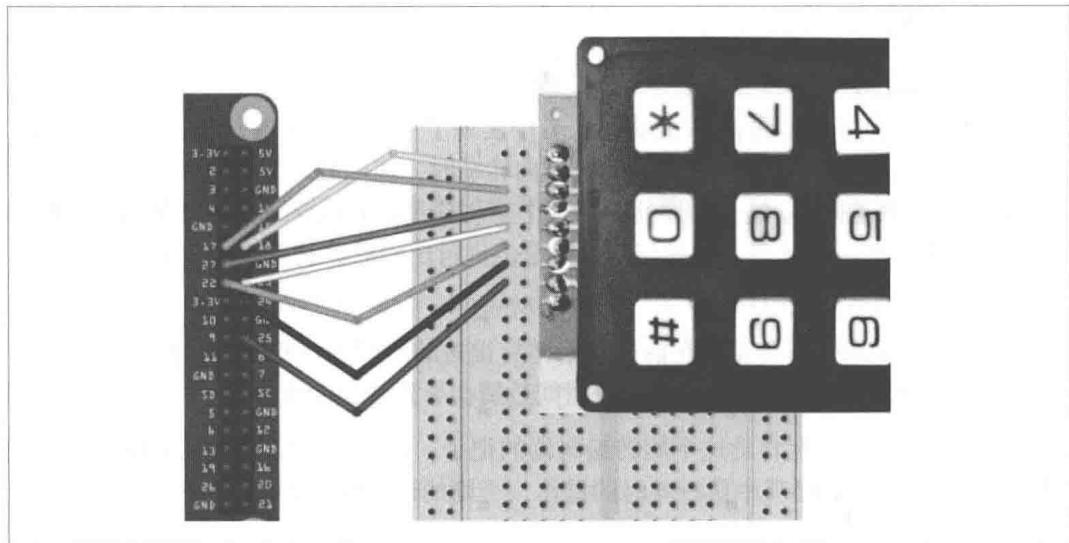


图 12-12 数字键盘布线图

打开一个编辑器（ nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，就本例来说，该程序名为 keypad.py。



在运行该程序之前，请确保对于你使用的数字键盘来说，行引脚和列引脚都是正确的，此外，如果需要的话，可以修改变量 rows 和 cols 的值。否则的话，当按下一个键的时候，可能导致 GPIO 的一个高电平输出与另一个低电平的输出短路。这就可能损坏树莓派。

这里定义的行和列的布局，对于数字键盘来说是正确的。第一行连接到 GPIO 的 17 号引脚，第二行连接到 25 号引脚，其他依此类推。行和列与数字键盘接口的导线连接如图 12-13 所示。

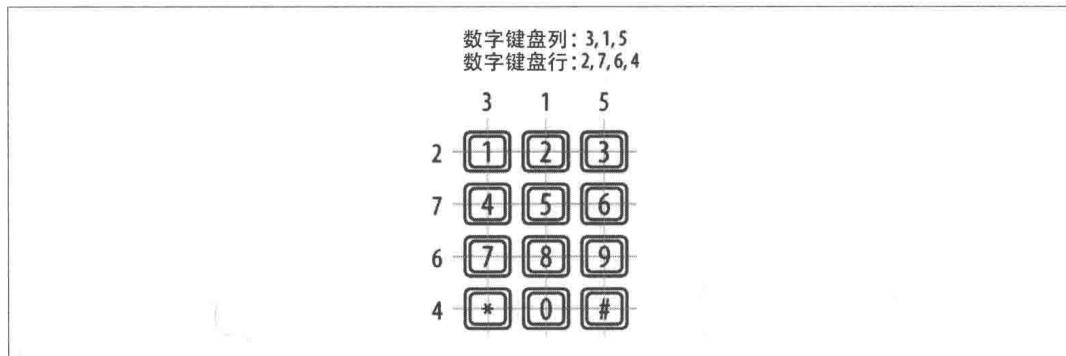


图 12-13 数字键盘引脚的连接

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rows = [17, 25, 24, 23]
cols = [27, 18, 22]
keys = [
    ['1', '2', '3'],
    ['4', '5', '6'],
    ['7', '8', '9'],
    ['*', '0', '#']]

for row_pin in rows:
    GPIO.setup(row_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

for col_pin in cols:
    GPIO.setup(col_pin, GPIO.OUT)

def get_key():
    key = 0
    for col_num, col_pin in enumerate(cols):
        GPIO.output(col_pin, 1)
        for row_num, row_pin in enumerate(rows):
            if GPIO.input(row_pin):
                key = keys[row_num][col_num]
        GPIO.output(col_pin, 0)
    return key

while True:
    key = get_key()
```

```
if key :  
    print(key)  
    time.sleep(0.3)
```

要想运行该程序，你需要具有超级用户身份，因为它将访问 GPIO。当你逐次按下各个键时，可以通过程序了解相应的跟踪信息。

```
pi@raspberrypi ~ $ sudo python keypad.py  
1  
2  
3  
4  
5  
6  
7  
8  
9  
*  
0  
#
```

进一步探讨

变量 `keys` 用于存放行列位置与键名之间的映射。你可以根据自己的数字键盘对其进行定制。

因为需要初始化为输入和输出的引脚很多，所以可以使用循环语句来初始化所有行引脚和列引脚。

所有实际动作都是在函数 `get_key` 中进行的。它会依次启用每一列，方法是将其设置为高电平。然后，通过一个内部循环来依次测试每一行。如果某一行为高电平，那么会在 `keys` 数组中寻找与这一行和这一列对应的键名。如果没有检测的任何按键动作，那么就会返回默认值 `key(0)`。

主循环 `while` 只是获取键的值，并打印输出该值。命令 `sleep` 只是用于降低输出速度。

参考资料

除了使用数字键盘以外，你还可以使用 USB 数字键盘。这样的话，你只需要按照 12.11 节描述的方法来捕捉击键就行了。

12.9 检测移动

面临问题

你希望在检测到移动发生时，通过 Python 触发某些动作。

解决方案

你可以使用 PIR（被动红外）运动检测模块。

为了进行本节中的实验，你需要：

- Female-to-Female（母头转母头）跳线；
- PIR 运动检测模块。

图 12-14 展示了传感器模块的连接方法。这个模块预期使用 5V 电源，并输出 3.3V 电压，所以可以完美适用于树莓派。

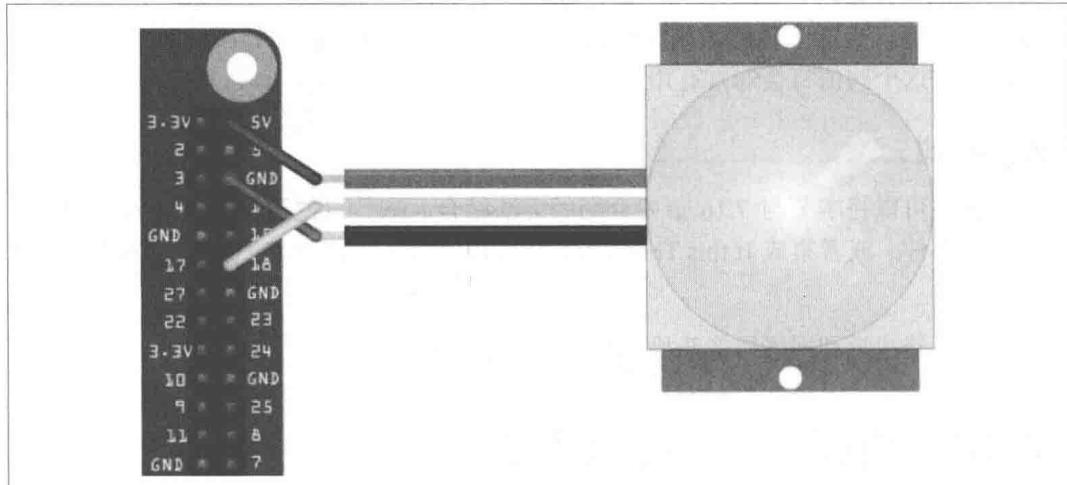


图 12-14 连接 PIR 运动检测模块



请确保所用的 PIR 模块具有 3.3V 输出。如果其输出为 5V 的话，则需要使用两个电阻将其降为 3.3V（具体参考 13.6 节）。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 pir.py。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.IN)
while True:
    input_state = GPIO.input(18)
    if input_state == True:
```

```
print('Motion Detected')
time.sleep(1)
```

该程序只是简单输出 GPIO 的 18 号输入的状态。

```
$ sudo python pir.py
Motion Detected
Motion Detected
```

进一步探讨

一旦触发，PIR 传感器的输出就会保持一段时间的高电平。你可以使用该模块的电路板上面的调谐电位器来调整这个时间。另一个调谐电位器（如果有的话）可以用来设置亮度级的阀值。当这个传感器用于控制照明，比如只有在黑夜里检测到移动才打开照明的时候，这个阀值将会非常有用。

参考资料

此外，你还可以将本节与 7.16 节中介绍的方法结合起来，以便检测到有人入侵的时候发送电子邮件，或者集成 If this Then That (IFTTT) 服务提供更加完善的通知方式（见 15.3 节）。

为了利用机器视觉和网络摄像头检测移动，请参考 8.6 节。

12.10 为树莓派添加 GPS

面临问题

你希望给移动树莓派连接一个串行 GPS 模块，然后通过 Python 来访问相应的数据。

解决方案

你可以使用一个 3.3V 的串行 GPS 模块，直接连接到树莓派的 RXD 连接上面。

图 12-15 展示了该模块的连接方式。树莓派的 RXD 与 GPS 模块的 Tx 相连。然后，就只剩下连接 GND 和 5V 了，所以，我们仅需使用 3 个 Female-to-Female (母头转母头) 接头即可。

需要注意的是 GPS 的消息需要解码之后才能使用。不过，好在已经有一套软件非常适合此项工作。为此，需要利用下列命令来安装这些软件包。

```
$ sudo apt-get install gpsd
$ sudo apt-get install gpsd-clients
```

在这些软件包中，最重要的一个就是 gpsd。它能够从串行连接或 USB 连接以及其他数据源来读取 GPS 数据，然后在端口 2748 上提供一个本地 Web 服务，将数据提供给客

户程序使用。

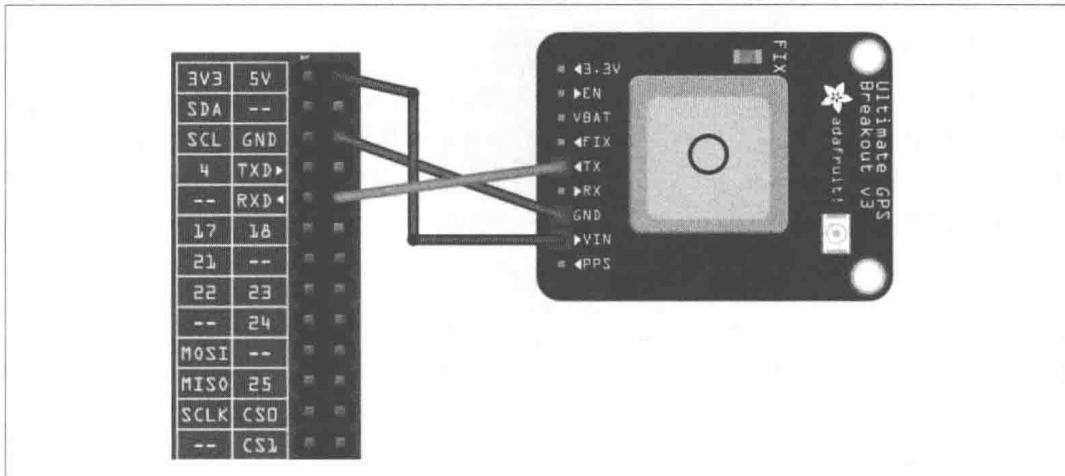


图 12-15 将 GPS 连接至树莓派

为了启动 gspd 服务，可以使用下列命令。

```
$ sudo gspd /dev/ttyAMA0
```

为了检查它是否工作正常，可以使用如下所示的命令。

```
$ cgps -s
```

上面的-s 是可选项，作用是禁止显示原始数据（见图 12-16）。

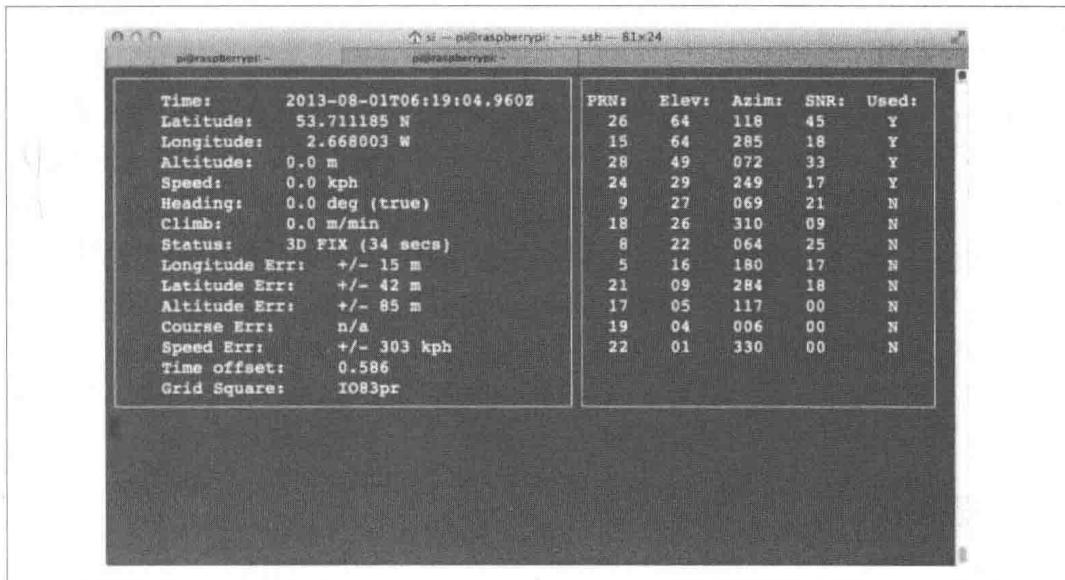


图 12-16 利用 cgps 测试 GPS

如你所料，上面安装的第三个软件包（python-gps）是一个 Python 库，有了它，你就可以通过一种简单正确的方式来访问 GPS 数据了。下面，将介绍如何利用 python_gps 和一个简短的测试程序来显示维度、经度和时间。

打开一个编辑器(nano 或者 IDLE)并粘贴如下代码。请不要把这个文件命名为 gps.py，因为这会与 Python 的 gps 库发生冲突。就像本书中的其他示例代码一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，就本例来说，该程序名为 gps_test.py。

```
from gps import *
session = gps()
session.stream(WATCH_ENABLE|WATCH_NEWSTYLE)

while True:
    report = session.next()
    if report.keys()[0] == 'epx' :
        lat = float(report['lat'])
        lon = float(report['lon'])
        print("lat=%f\lon=%f\time=%s" % (lat, lon, report['time']))
        time.sleep(0.5)
```

运行该程序，你将看到类似下面这样的踪迹信息。

```
$ python gps_test.py
lat=53.710257  lon=-2.664245  time=2013-08-01T08:06:24.960Z
lat=53.710258  lon=-2.664252  time=2013-08-01T08:06:25.960Z
lat=53.710258  lon=-2.664252  time=2013-08-01T08:06:25.960Z
lat=53.710248  lon=-2.664243  time=2013-08-01T08:06:26.960Z
lat=53.710248  lon=-2.664243  time=2013-08-01T08:06:26.960Z
lat=53.710248  lon=-2.664250  time=2013-08-01T08:06:27.960Z
```

进一步探讨

上面的程序创建了一个会话，然后为了读取数据而建立了相应的数据流。GPS 将以不同的格式重复发送消息。你可以通过命令 if 选取自己需要的消息，即包含了位置信息的那些消息。消息部分被存储在一个字典中，你可以从中访问和显示它们。

你除了可以通过 Python 使用 GPS 数据之外，也可以通过 xgps 工具来显示 GPS 数据（见图 12-17）。为此，只需输入下列命令即可。

```
$ xgps
```

这个工具要求使用显示器，所以，你应该直接在树莓派上面运行它，或者通过 VNC（见 2.8 节）或 RDP（见 2.9 节）来运行它。

参考资料

你可以利用同样的方式来使用 USB GPS 模块 (<http://bit.ly/1edSyHX>)。

关于 gpsd 的详细介绍，请访问 <http://bit.ly/1fP61q4>。

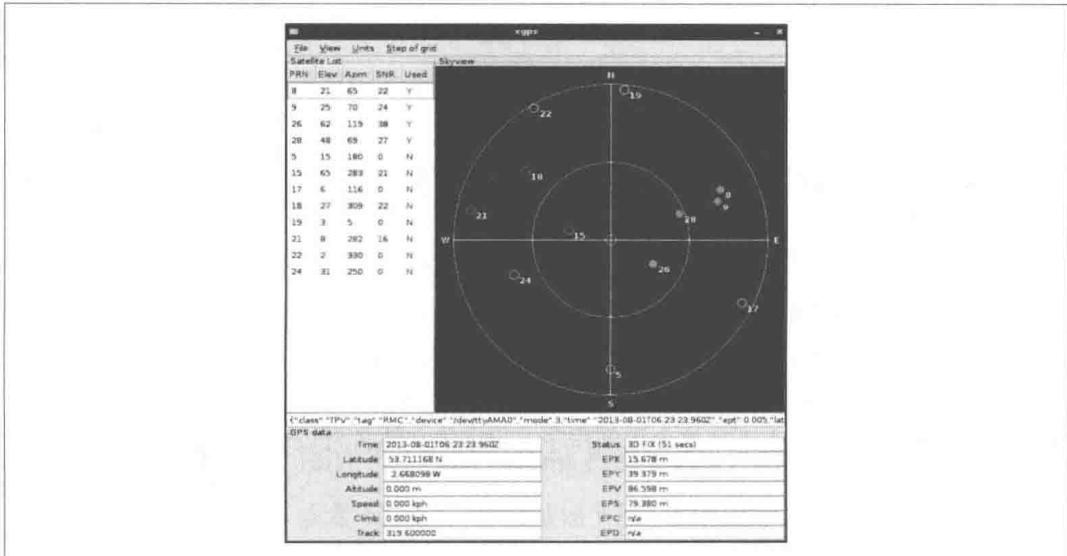


图 12-17 利用 xgps 浏览 GPS 数据

12.11 拦截按键

面临问题

你需要拦截 USB 数字键盘或数字键板上的某个按键。

解决方案

对于这个问题，至少有两种解决方法，最简单的方法是使用 `sys.stdin.read` 函数。较之于其他方法，该方法的优点在于无需图形用户界面，所以使用这种方法的程序可以从 SSH 会话执行。

打开一个编辑器（`nano` 或者 `IDLE`），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 `keys_sys.py`。

```
import sys, tty, termios

def read_ch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch
```

```
while True:  
    ch = read_ch()  
    if ch == 'x':  
        break  
    print("key is: " + ch)
```

另一种替代方法是使用 Pygame。Pygame 是为编写游戏软件而设计的 Python 库，同时，它还可以用来检测按键。所以，你可以用它检测到按键后执行某些动作。由于 Python 3 中没有包含 Pygame，所以本节中的示例必须使用 Python 2，而非 Python 3。

下面的示例程序为我们展示了如何在每次按键时通过 Pygame 打印输出一则消息。不过，该程序只能在窗口系统下面运行，所以你需要通过 VNC（见 2.8 节）、RDP（见 2.9 节）或者直接在树莓派上面运行它。你可以从本书的网站 (<http://www.raspberrypicookbook.com>) 下载这个示例程序，程序的文件名为 keys_pygame.py)。

```
import pygame  
import sys  
from pygame.locals import *  
  
pygame.init()  
screen = pygame.display.set_mode((640, 480))  
pygame.mouse.set_visible(0)  
  
while True:  
    for event in pygame.event.get():  
        if event.type == QUIT:  
            sys.exit()  
        if event.type == KEYDOWN:  
            print("Code: " + str(event.key) + " Char: " + chr(event.key))
```

它会打开一个空的 Pygame 窗口，并且只有窗口被选中的时候，按键才会被截获。该程序将在运行它的终端窗口中输出相应结果。

如果你按下一个箭头键或 Shift 键，由于这些键没有对应的 ASCII 值，所以该程序会抛出一个错误。

```
$python keys_pygame.py  
Code: 97 Char: a  
Code: 98 Char: b  
Code: 99 Char: c  
Code: 120 Char: x  
Code: 13 Char:
```

就本例而言，无法使用 Ctrl-C 来停止该程序的运行。要想关闭这个程序，需要单击 PyGame 窗口上面的 X 标记。

进一步探讨

当你使用 Pygame 方式的时候，其他键会被定义相应的常数值，从而允许你使用键盘上面的光标及其他非 ASCII 键（比如向上箭头键和 Home 键）。利用其他方式的时候，这是无法实现的。

参考资料

拦截键盘事件可以代替使用矩阵键盘（见 12.8 节）。

12.12 拦截鼠标移动

面临问题

你希望利用 Python 检测鼠标移动。

解决方案

这个问题的解决方法与 12.11 节中使用 Pygame 拦截键盘事件的方法非常类似。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 mouse_pygame.py。

```
import pygame
import sys
from pygame.locals import *

pygame.init()
screen = pygame.display.set_mode((640, 480))
pygame.mouse.set_visible(0)

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            sys.exit()
        if event.type == MOUSEMOTION:
            print("Mouse: (%d, %d)" % event.pos)
```

每当鼠标在 Pygame 窗口中移动的时候，就会触发 MOUSEMOTION 事件。你可以根据该事件的 pos 值来获得坐标数据。当然，这个坐标是相对于窗口左上角的绝对坐标。

```
Mouse: (262, 285)
Mouse: (262, 283)
Mouse: (262, 281)
Mouse: (262, 280)
Mouse: (262, 278)
Mouse: (262, 274)
Mouse: (262, 270)
Mouse: (260, 261)
Mouse: (258, 252)
Mouse: (256, 241)
Mouse: (254, 232)
```

进一步探讨

除此之外，其他可以拦截的事件还有 MOUSEBUTTONDOWN 和 MOUSEBUTTONUP。它们可以用来检测鼠标左键按下和释放的时间。

参考资料

在 Pygame 文档中，关于 mouse 的介绍请访问其官方网站 (<http://www.pygame.org/docs/ref/mouse.html>)。

12.13 使用实时时钟模块

面临问题

你希望树莓派能够记住时间，即使没有联网的情况下也能如此。

解决方案

你可以使用一个实时时钟（RTC）模块。

DS1307 是一种非常常见的 RTC 芯片。它提供了一个 I2C 接口，并且作为一个现成的模块，自身含有芯片、维持精确计时的石英和可以容纳一节 3V 锂电池的电池仓。为了进行本节中的实验，你需要：

- 一个 DS1307 或兼容 RTC 模块；
- Female-to-Female（母头转母头）跳线。



你使用的 RTC 模块必须兼容 3.3V。这就是说，该模块的 I2C 接口要么根本没有上拉电阻，要么上拉电压到 3.3V，而非 5V。如果这里使用 Adafruit 模块的话，在焊接到该模块时，不要包含那两个电阻。如果你有现成的模块，那么务必去掉所有上拉电阻。

如果 RTC 模块为套装形式的话，请先组装起来，不要忘了摒弃上拉电阻，然后将该模块连接到树莓派，具体如图 12-18 所示。

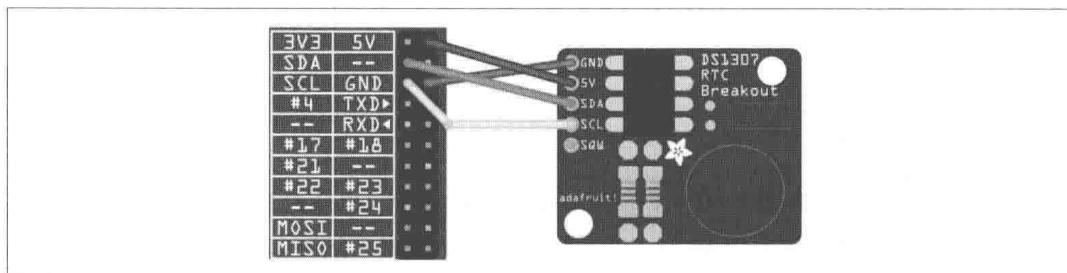


图 12-18 连接 RTC 模块

因为 DS1307 是一种 I2C 模块，所以，必须将你的树莓派设置为在 I2C 模式下工作（见 9.3 节）。为此，你可以使用 I2C 工具检测该设备是否可见（见 9.4 节）。

```
$ sudo i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          - - - - - - - - - - - - - - - -
10:          - - - - - - - - - - - - - - - -
20:          - - - - - - - - - - - - - - - -
30:          - - - - - - - - - - - - - - - -
40:          - - - - - - - - - - - - - - - -
50:          - - - - - - - - - - - - - - - -
60:          - - - - - - - - - - - - - - - -
70:          - - - - - - - -
```

上面表中的 68 表示 RTC 模块被连接到了地址为 68（十六进制）的 I2C 总线上面。

如果你打算使用原生 B 型树莓派第一版的话，那么需要在上面的命令行的选项 y 之后使用 0。第一版的电路板的特别之处在于它有一个黑色的音频接口。

接下来，你需要运行下列命令，这样 RTC 就可以为程序 hwclock 所用了。

```
$ sudo modprobe rtc-ds1307
$ sudo bash
$ echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
```

再次重申，如果你使用的是第一版的树莓派的话，必须将 i2c-1 改为 i2c-0。

现在，你已经可以使用下列命令来访问 RTC 了。

```
$ sudo hwclock -r
Sat 01 Jan 2000 00:08:08 UTC -0.293998 seconds
```

就像你看到的那样，当前尚未设置时钟。

为了设置 RTC 模块的时间，首先要确保树莓派上面的时间是正确的。如果树莓派已经连接互联网的话，时间的设置将是自动完成的。为此，你可以使用 date 命令进行检查。

```
$ date
Tue Aug 20 06:42:47 UTC 2013
```

如果时间不正确的话，你也可以使用 date 命令以手动方式进行设置（见 3.35 节）。为了将树莓派的系统时间传递给 RTC 模块，可以使用如下所示的命令。

```
$ sudo hwclock -w
```

此后，你就可以使用-r 选项来读取时间了，具体命令如下所示。

```
$ sudo hwclock -r
Wed 02 Jan 2013 03:11:43 UTC -0.179786 seconds
```

RTC 具有正确时间本身没有多大意义，它的真正用途在于系统重启的时候用它给 Linux

设置正确的系统时间。为此，你需要更改几处设置。

首先，编辑文件/etc/modules（使用命令 sudo nano /etc/modules），在模块列表末尾添加 rtc-ds1307。如果在设置 I2C、SPI 及其他选项时已经添加了一些模块的话，那么文件看起来应该如下所示。

```
# /etc/modules: kernel modules to load at boot time.  
#  
# This file contains the names of kernel modules that should be loaded  
# at boot time, one per line. Lines beginning with "#" are ignored.  
# Parameters can be specified after the module name.  
  
snd-bcm2835  
i2c-bcm2708  
i2c-dev  
spidev  
rtc-ds1307
```

然后，你需要在启动期间自动运行两个命令来设置系统时间。为此，可以使用命令 sudo nano /etc/rc.local 来编辑文件/etc/rc.local，并在最后一行即 exit 0 前面插入如下所示两行命令。如果你使用的是树莓派第一版的话，务必将 i2c -1 改为 i2c -0。

```
$ echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device  
$ sudo hwclock -s
```

完成上述工作后，文件将变成下面的样子。

```
#  
# In order to enable or disable this script, just change the execution  
# bits.  
#  
# By default, this script does nothing.  
  
# Print the IP address  
_IP=$(hostname -I) || true  
if [ "$_IP" ]; then  
    printf "My IP address is %s\n" "$_IP"  
fi  
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device  
sudo hwclock -s  
exit 0
```

接下来，当你重启的时候，树莓派将根据 RTC 来设置其系统时间。

不过，如果有可用的互联网连接的情况下，会优先采用互联网方式来设置系统时间。

进一步探讨

RTC 对于树莓派来说并非必须可少的，因为联网的树莓派会自动通过时间服务器来

设置自己的时间。但是，你的树莓派无法保证时刻联网，所以，最好还是选用 RTC 硬件。

参考资料

AB Electronics 提供了一种简洁的 RTC，它可以直接插进 GPIO 接口（<http://bit.ly/1hShxWi>）。这种 RTC 如图 12-19 所示。

本节内容基于 Adafruit (<http://bit.ly/IcKvQ9>) 上面的一篇指南文字。

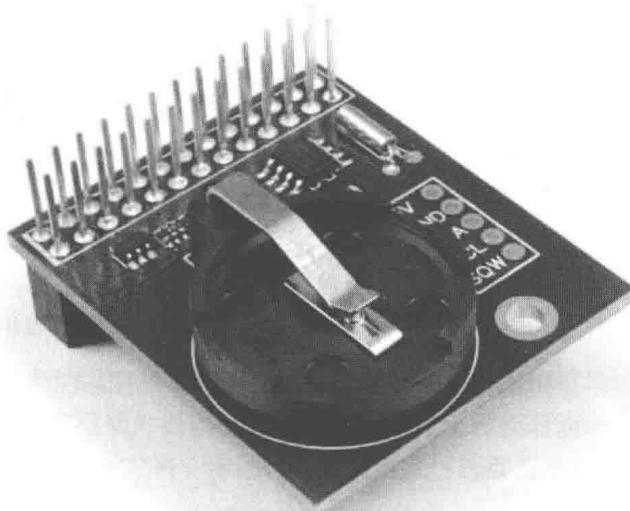


图 12-19 AB Electronics 的 RTC 模块

传感器

13.0 引言

在本章中，我们将会通过许多实例来介绍各种传感器的使用方法。在这些传感器的帮助下，树莓派就可以测量温度、光线等。

与诸如 Arduino 之类的电路板不同的是树莓派本身并没有提供模拟输入。这就意味着在使用传感器的时候，需要额外提供相应的模数（ADC）转换器硬件。幸运的是，这件事可以轻而易举地完成。除此之外，你还可以将电阻式传感器与电容和多个电阻配合使用。

本章中的大部分例子都需要用到免焊面包板和 Female-to-Female（母头转母头）跳线（见 9.8 节）。

13.1 使用电阻式传感器



请务必观看在 <http://razzpisampler.oreilly.com> 上与本节有关的视频。

面临问题

你想要给树莓派连接一个可变电阻器并测量阻值，然后利用 Python 程序确定出可变电阻器的把手位置。

解决方案

你只需要借助一个电容器、几个电阻和两个 GPIO 引脚就能在树莓派上测量电阻值了。

在本例中，你能够通过测量小型可变电阻器从滑块触点到电位器的一端之间的电阻值来计算把手的位置。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- $10k\Omega$ 调谐电位器；
- 2 个 $1k\Omega$ 电阻；
- $330nF$ 电容器。

图 13-1 展示了各个元件在面包板上面的布局情况。

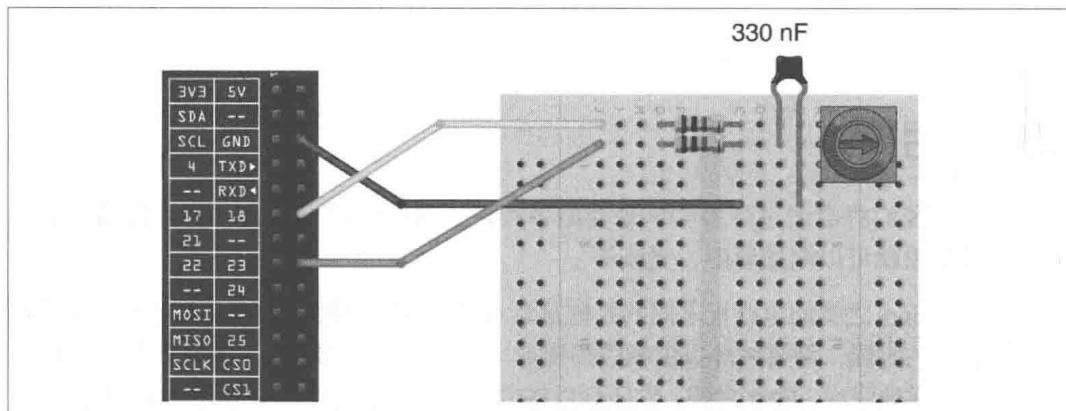


图 13-1 在树莓派上测量阻值

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 pot_step.py 。

```
import RPi.GPIO as GPIO
import time, math

C = 0.36 # uF
R1 = 1000 # Ohms

GPIO.setmode(GPIO.BCM)
# Pin a charges the capacitor through a fixed 1k resistor
# and the pot in series
# Pin b discharges the capacitor through a fixed 1k resistor
a_pin = 18
b_pin = 23

# Discharge the capacitor, leaving it ready to start filling up
def discharge():
    GPIO.setup(a_pin, GPIO.IN)
    GPIO.setup(b_pin, GPIO.OUT)
```

```

GPIO.output(b_pin, False)
time.sleep(0.1)

# Return the time taken (uS) for the voltage on the capacitor
# to count as a digital input HIGH
# which is 1.65V or higher
def charge_time():
    GPIO.setup(b_pin, GPIO.IN)
    GPIO.setup(a_pin, GPIO.OUT)
    GPIO.output(a_pin, True)
    t1 = time.time()
    while not GPIO.input(b_pin):
        pass
    t2 = time.time()
    return (t2 - t1) * 1000000

# Take an analog reading as the time taken to charge after
# first discharging the capacitor
def analog_read():
    discharge()
    t = charge_time()
    discharge()
    return t

# Convert the time taken to charge the capacitor into a value of resistance
# To reduce errors, do it 100 times and take the average
def read_resistance():
    n = 10
    total = 0;
    for i in range(1, n):
        total = total + analog_read()
    t = total / float(n)
    T = t * 0.632 * 3.3
    r = (T / C) - R1
    return r

try:
    while True:
        print(read_resistance())
        time.sleep(0.5)
finally:
    GPIO.cleanup()

```

当你运行程序时，将会看到类似下面的输出结果。

```

$ sudo python pot_step.py
10049.2157936
10105.1559448
10158.6098671
11331.0049375
10162.6154582
10156.8142573
9501.27855937
8216.17444356

```

当你旋转调谐电位器的旋钮的时候，其读数也会随之变化。读取的阻值的变化范围为 0 到 10000Ω ，不过实际上会存在一定的误差。如果想要获得更加精确的读数的话，可以调整程序顶部的常量 C 的值，同时有一点需要牢记，就是放入另一个电容的时候，即使该电容标称的值是相同的，也要修改 C 的值。

进一步探讨

要想了解该程序的工作机制，需要先了解如何利用阶跃响应技术测量可变电阻器的阻值。

图 13-2 展示了本示例的原理图。

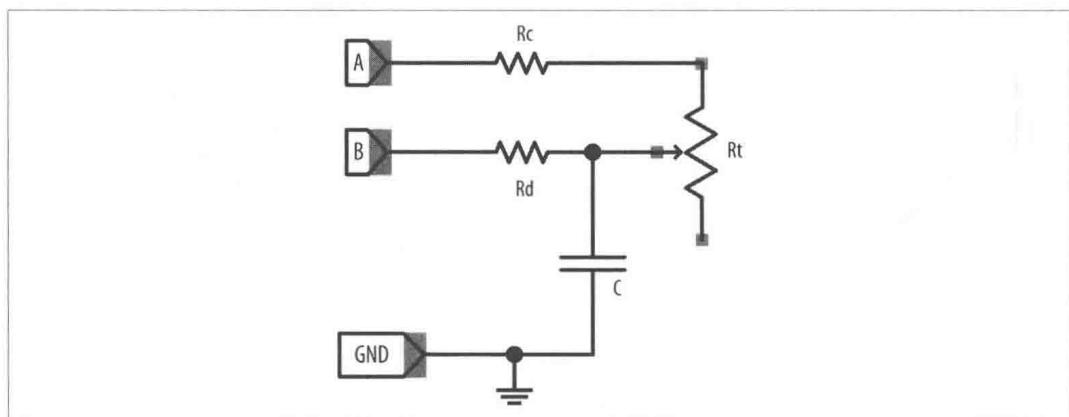


图 13-2 利用阶跃响应测量电阻

该机制之所以名为阶跃响应，是因为它是通过检测输出从低电平变为高电平而引起阶跃变化时电路的响应来工作的。

你可以将电容器视为一个电子的容器，并且随着不断填充电荷，电容器的电压会不断升高。你无法直接测量电容器的电压，因为树莓派没有 ADC 转换器。但是，你可以计算电容器通过充电使其电压超过 1.65V 或形成一个高电平数字输入所需时间。电容器的充电速度取决于可变电阻器 (Rt) 的阻值。阻值越小，电容器的充电速度和电压上升的速度就越快。

为了能够获得准确的读数，你必须在每次读取之前清空电容器。在图 13-2 中，线路 A 用于通过 R_c 和 R_t 给电容器充电，线路 B 则用于通过 R_d 给电容放电（清空）。电阻 R_c 和 R_d 的作用是当电容器充电和放电的时候，防止有过大的电流损坏树莓派的相关 GPIO 引脚。

读取数值的相关步骤是首先通过 R_d 对电容进行放电，然后通过 R_c 和 R_t 对其充电。

为了进行放电，需要将连接 A (GPIO 18) 设置为输入，从而有效断开 R_c 和 R_t 与

电路的连接。之后，将连接 B (GPIO 23) 设为输出低电平，并保持 100 毫秒以清空电容。

现在，电容已经清空，你可以通过将连接 B 设为输入（有效地断开它），并启用连接 A 为 3.3V 高电平，这样就可以开始充电了。此时，电容器 C 开始通过 R_C 和 R_t 来进行充电。在连接 B 的电压从低升高至 1.65V 左右之前，while 循环什么也不干。

当连接 B 的电压升到 1.65V 时，它就会返回时间。

图 13-3 展示了这种充放电组合中电阻和电容器的电压在高低电平之间的切换方式。

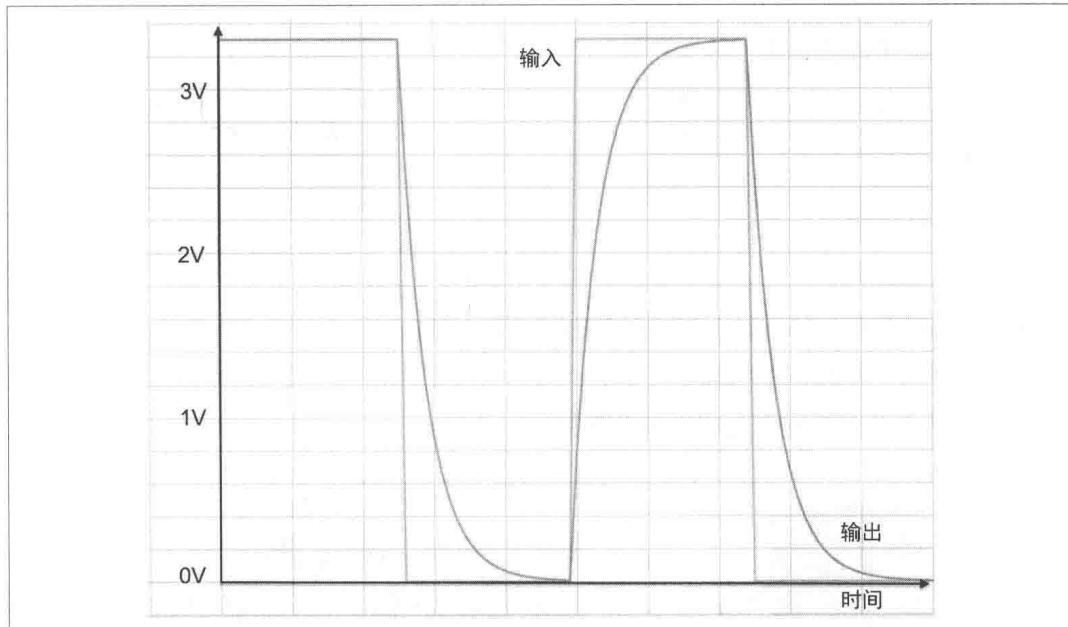


图 13-3 电容器的充电和放电参考资料

就像上面看到的那样，电容器的电压起初迅速增加，但是电容器充满电之后就会减小。幸运的是该曲线中我们所关注的部分会以近乎直线的形式一直上升，直至电容电压达到 1.65V 为止，也就是说，电容器的电压升到这里所花时间大致与 R_t 的阻值和旋钮的位置成正比。

这种方法并不是非常精确，但是成本极低，并且易于使用。误差主要是由于电容器的高位值的精度只有 10%。

阶跃响应适用于所有种类的电阻式光传感器（见 13.2 节）、温度传感器（见 13.3 节），甚至气体传感器（见 13.4 节）。

关于调谐电位器位置精确测量的详细介绍，请参考 13.5 节中联合使用 ADC 转换器和电位器的测量方法。

13.2 测量亮度

面临问题

你想通过树莓派和光敏电阻器来测量光强。

解决方案

你可以使用 13.1 节中介绍的方法和代码，将调谐电位器换为光敏电阻器即可。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 光敏电阻器；
- 2 个 $1\text{k}\Omega$ 电阻；
- 330nF 电容器。

Monk Makes 提供的 Electronics Starter Kit for Raspberry Pi 中包含了所有上述元件（见 474 页的“成型设备”部分）。

图 13-4 展示了各元件在面包板上面的布局。

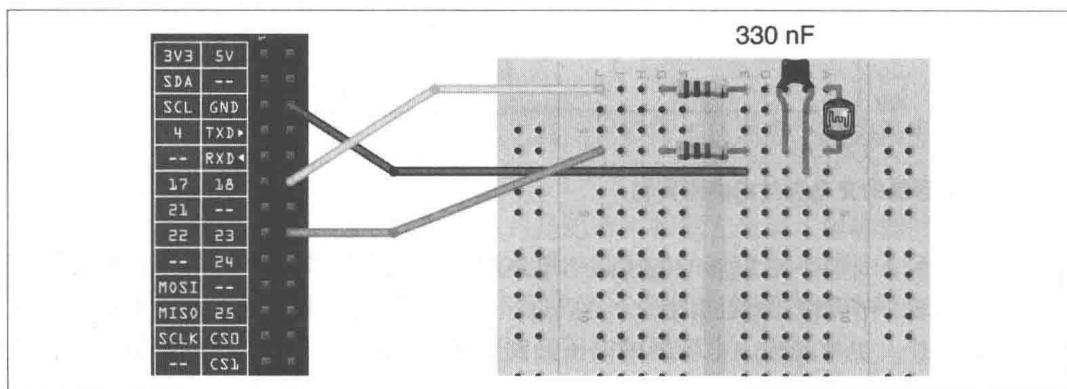


图 13-4 利用树莓派测量光强

使用 13.1 节中的 `pot_step.py` 程序的时候，你会发现输出内容会随着你的手在光敏电阻器上方的移动，即遮挡住光线的多少，而发生变化。

这种解决方案提供了较为可靠的亮度读数。作为电阻传感器（见 13.1 节）的一般解决方案的改进，这种方案可以在绝不会损坏树莓派 GPIO 引脚的情况下测量 0Ω 的电阻。由于光敏电阻器的阻值永远不会下降到零，所以你可以进一步简化电路：去掉两个 $1\text{k}\Omega$

电阻器和一个 GPIO 引脚，简化后的结果如图 13-5 所示。

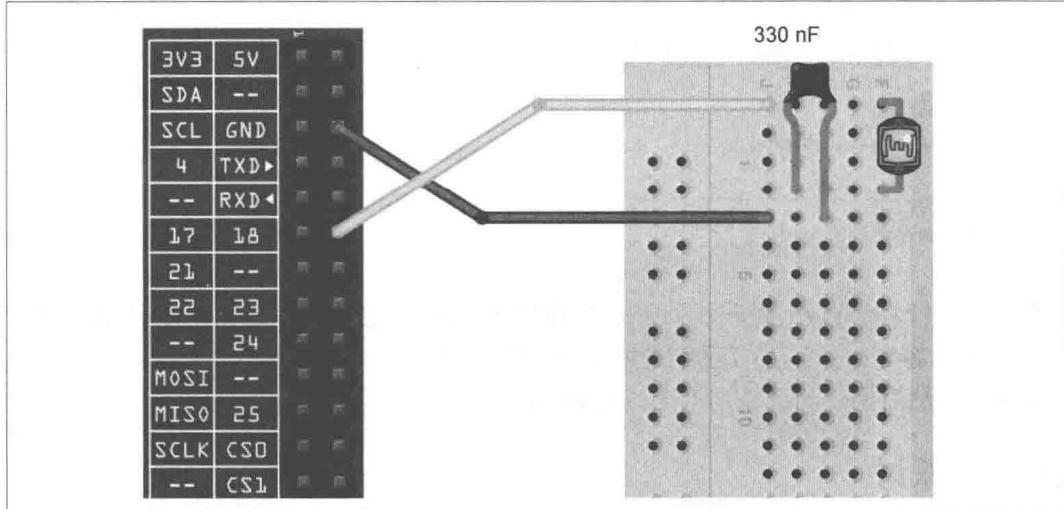


图 13-5 简化后的光传感器

在这个删减版中，同一个 GPIO 引脚被用于电容器的充放电以及检查电压是否高于输入阈值 HIGH。相应的程序代码位于文件 photoresistor_step.py 中。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

pin = 18

def discharge():
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, False)
    time.sleep(0.1)

# Return the time taken for the capacitor to go HIGH
# which is 1.65V or higher
def charge_time():
    t1 = time.time()
    GPIO.setup(pin, GPIO.IN)
    while not GPIO.input(pin):
        # charge for 1ms
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, True)
        time.sleep(0.001)
        # set to input to test again
        GPIO.setup(pin, GPIO.IN)
        time.sleep(0.001)
```

```
t2 = time.time()
return (t2 - t1) * 1000000

# Take an analog reading as the time taken to charge C
def analog_read():
    discharge()
    return charge_time()

while True:
    print(analog_read())
    time.sleep(0.5)
```

在只使用一个引脚的情况下，通过光敏电阻给电容放电之后，`charge_time` 函数通过光敏电阻为电容充电仅需 1 毫秒。它将该引脚设置为输入，并且测试其电压是否达到阈值 HIGH，然后重复测试直到电压达到 HIGH 为止。

进一步探讨

光敏电阻器是这样一种电阻器，其阻值会随着通过透明窗口进入其内的光线的多少而变化。光线越亮，阻值越小。一般地，阻值会在 $1\text{k}\Omega$ （强光）到 $100\text{k}\Omega$ （完全黑暗）之间变动。

实际上，传感器充其量不过给出了一个大致的亮度水平。

参考资料

你还可以使用 ADC 和光敏电阻器来测量光的强度，具体见 13.5 节。

13.3 利用热敏电阻测量温度

面临问题

你想利用热敏电阻测量温度。

解决方案

热敏电阻是一种阻值会随着温度而变化的电阻。利用 13.1 节中介绍的阶跃响应方法可以测量出热敏电阻的阻值，从而计算出温度。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- $1\text{k}\Omega$ 热敏电阻；
- 2 个 $1\text{k}\Omega$ 电阻；

- 330nF 电容器。

Monk Makes 提供的 Electronics Starter Kit for Raspberry Pi 中包含了所有上述元件。获取热敏电阻时，要确保了解其 Beta 和 R₀ (25°C 时的阻值) 的值，同时它是负温度系数 (NTC) 设备。

图 13-6 展示了本节所用面包板的布局。

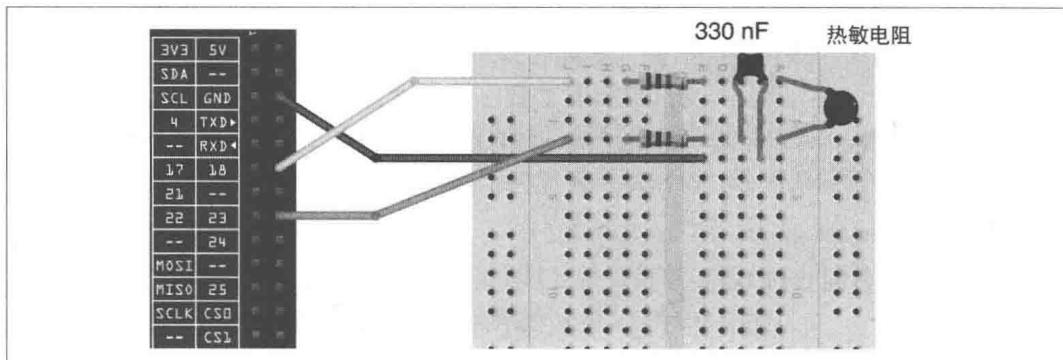


图 13-6 使用热敏电阻的面包板的布局

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 `thermistor.py` 。

```
import RPi.GPIO as GPIO
import time, math

C = 0.36 # uF
R1 = 1000 # Ohms
B = 3800.0 # The thermistor constant
R0 = 1100.0 # The resistance of the thermistor at 25C

GPIO.setmode(GPIO.BCM)

# Pin a charges the capacitor through a fixed 1k resistor
# and the thermistor in series
# Pin b discharges the capacitor through a fixed 1k resistor
a_pin = 18
b_pin = 23

# Discharge the capacitor, leaving it ready to start filling up
def discharge():
    GPIO.setup(a_pin, GPIO.IN)
    GPIO.setup(b_pin, GPIO.OUT)
    GPIO.output(b_pin, False)
    time.sleep(0.1)
```

```

# Return the time taken (uS) for C to charge
# which is 1.65V or higher
def charge_time():
    GPIO.setup(b_pin, GPIO.IN)
    GPIO.setup(a_pin, GPIO.OUT)
    GPIO.output(a_pin, True)
    t1 = time.time()
    while not GPIO.input(b_pin):
        pass
    t2 = time.time()
    return (t2 - t1) * 1000000

# Take an analog reading from charge time
def analog_read():
    discharge()
    t = charge_time()
    discharge()
    return t

# Convert the time taken to charge the capacitor into a value of resistance
# To reduce errors, do it 10 times and take the average.
def read_resistance():
    n = 10
    total = 0;
    for i in range(1, n):
        total = total + analog_read()
    t = total / float(n)
    T = t * 0.632 * 3.3
    r = (T / C) - R1
    return r

def read_temp_c():
    R = read_resistance()
    t0 = 273.15      # 0 deg C in K
    t25 = t0 + 25.0 # 25 deg C in K
    # Steinhart-Hart equation - Google it
    inv_T = 1/t25 + 1/B * math.log(R/R0)
    T = (1/inv_T - t0)
    return T

try:
    while True:
        print(read_temp_c())
        time.sleep(0.5)
finally:
    GPIO.cleanup()

```

当你运行该程序时，将会看到一系列以摄氏度为单位的温度值。为了将其转换为华氏度，可以使用公式 $T_f = T_c \times 9 / 5 + 32$ 。

```
$ sudo python thermistor.py  
18.3040458984  
17.8302664759  
17.3917856854  
17.9286173793
```

进一步探讨

该项目中的大部分代码与 13.1 节中的完全一致。新加入的部分就是函数 `read_temp_c`，它的作用是将阻值转换为以摄氏度为单位的温度。通过热敏电阻的阻值计算温度的时候，需要利用 Steinhart-Hart 方程进行一些非常复杂的数学运算。这个方程需要知道热敏电阻的两个值：在 25°C（称为 T0 或 T25）的阻值，以及用于热敏电阻的一个常量，称为 Beta 或 B。如果你使用了不同类型的热敏电阻的话，还需要将这些值插入到程序顶部的变量 B 和 R0 中。

需要注意的是电容器通常只有 10% 的准确性，并且热敏电阻的 R0 同样如此，所以要想通过热敏电阻获得有用的读数，你必须调整变量 C 和 R0 的值。

首先调整变量 C 的值，使用一个 $1\text{k}\Omega$ 精度为 1% 的电阻替换 13.1 节中的电位器。

然后，调整 R0 来获得与准确的温度表一致的读数。

参考资料

要想利用 TMP36 来测量温度，请参考 13.8 节。

要想使用数字温度传感器（DS18B20）来测量温度，请参考 13.11 节。

要想使用 Sense HAT 来测量温度，请参考 13.10 节。

13.4 检测甲烷

面临问题

你需要使用甲烷传感器来测量甲烷气体的浓度。

解决方案

你可以使用廉价的电阻式气体传感器，将其连接到树莓派上面，来检测甲烷等气体。为此，你可以使用 13.1 节中使用的阶跃响应方法。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 甲烷传感器；

- 2 个 $1\text{k}\Omega$ 电阻；
- 330nF 电容器。

该传感器包含了一个要求使用 5V 最大 150mA 电流的加热元件。只要树莓派的电源能够提供额外的 150mA 电流，完全可以利用树莓派来供电。

该传感器模块的引脚很粗，所以无法插入到面包板的插孔中。为了解决这个问题，可以在每个引脚上面焊接一个短的实心焊丝（见图 13-7）。另一种解决办法是购买 SparkFun 的气体传感器分线板 (<https://www.sparkfun.com/products/8891>)。

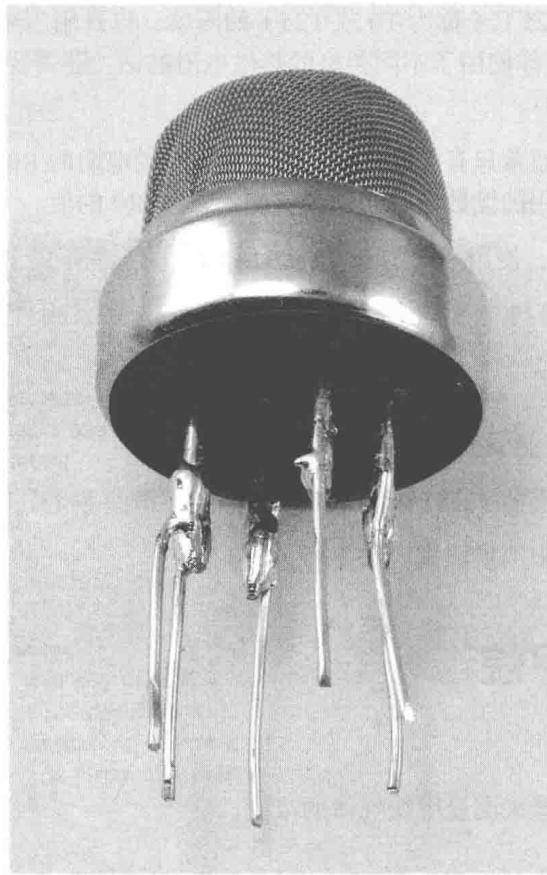


图 13-7 为气体传感器焊接导线

如果你使用了 SparkFun 的分线板，你可以按照图 13-8 所示那样连接面包板；如果你为气体传感器焊接了导线的话，可以像图 13-9 所示那样进行连接。

需要注意的是图 13-9 所示的直接连接方式使用了与分线板相同的符号，而不是传感器自身的符号，不过你若仔细进行检查就会发现，连接的是传感器的 6 个引脚，而非分

线板的 4 个引脚。

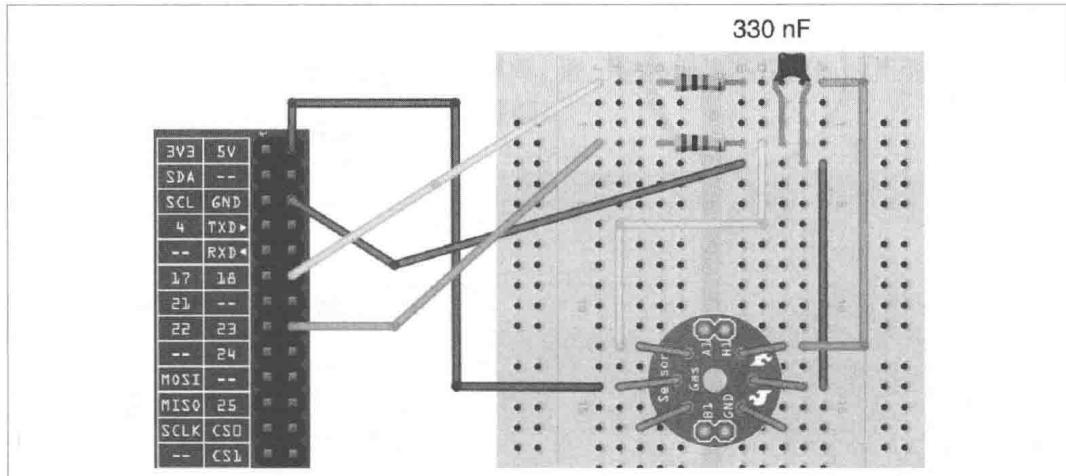


图 13-8 将气体传感器连接到树莓派（分线板）

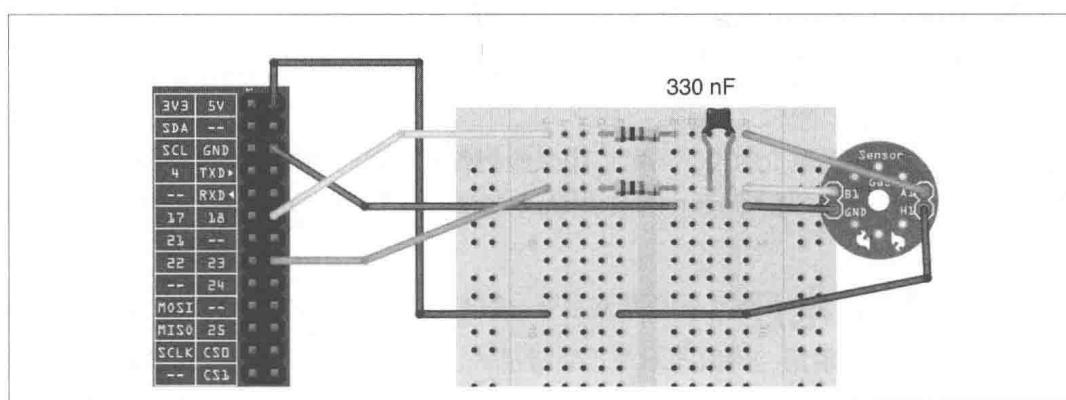


图 13-9 将气体传感器连接至树莓派（直接连接）

你可以原封不动地使用 13.1 节中的程序，通过向甲烷传感器吹气来进行测试。当你向其吹气的时候，你将看到传感器的读数会下降。

进一步探讨

甲烷气体传感器显而易见的用途是用于搞怪的放屁检测项目，更严肃的用途是检测天然气的泄漏。比如，假设有一个家庭监控项目，使用各种各样的传感器来监控家居。这样，它就可以在你度假的时候，如果出现情况，就会向你发送邮件，通知你家里可能发生爆炸。如果没有情况，就不会发送邮件。

图 13-10 中的这些类型的传感器使用了一个加热元件对特定气体敏感的触媒浸渍过的电阻表面进行加热。当存在某种气体时，触媒层的阻值就会发生变化。

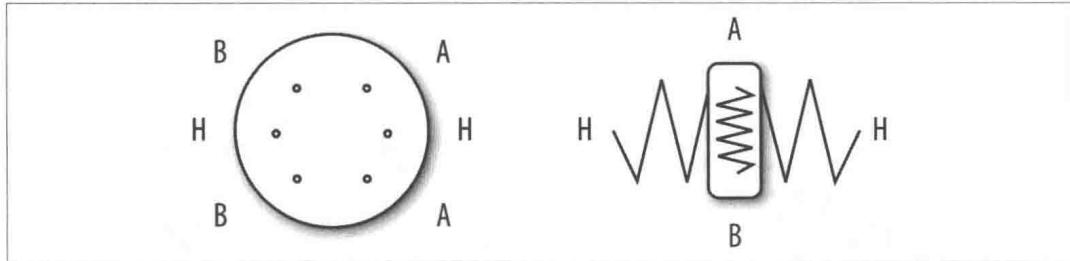


图 13-10 甲烷气体传感器

无论是加热器，还是敏感表面，从电子学的角度看都是电阻。因此，两者可以通过任意方式进行连接。

虽然这种特殊的气体传感器对甲烷最敏感，但是也能检测小范围内的其他气体。这就是对着它吹气的时候，虽然健康的人不会呼出甲烷，但是其读数依然会变的原因。对着该元件吹气时的冷却效应也会产生影响。

参考资料

本文所用传感器的参数手册，请访问 <http://bit.ly/1gYupsu> 页面。它给出了该传感器的各类敏感气体的所有信息。

除此之外，还有许多廉价的传感器可以用来检测各种不同的气体。关于 SparkFun 提供的传感器列表，请参考 SparkFun 网站 (<https://www.sparkfun.com/categories/146>)。

13.5 测量电压

面临问题

你想要测量模拟电压。

解决方案

树莓派的 GPIO 接口仅提供了数字输入。如果你想测量电压的话，需要使用一个单独的数模转换器（ADC）。

你可以使用 MCP3008 八通道 ADC 芯片。该芯片实际上提供了 8 个模拟输入，因此你可以每个通道上面连接一个传感器（也就是说，最多可以连接 8 个传感器），然后通过树莓派的 SPI 接口连接到该芯片。

为了进行本节中的实验，你需要：

- 面包板和跳线；

- MCP3008 八通道 ADC IC；
- 10k Ω 调谐电位器。

图 13-11 展示了用于该芯片的面包板的布局。请一定确保该芯片朝向的正确性，即芯片封装上面的小缺口应该朝向该面包板的顶端。

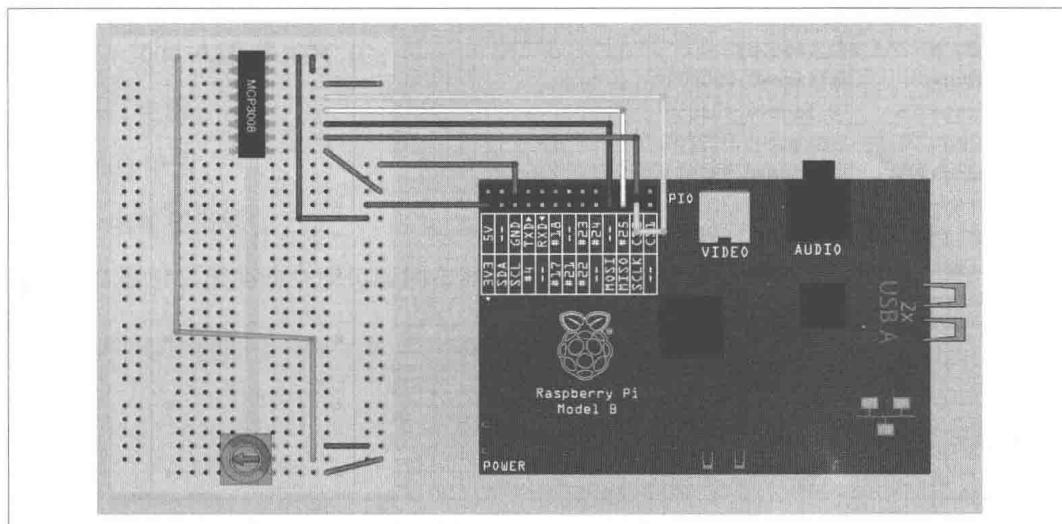


图 13-11 在树莓派上面使用 MCP3008 ADC IC

可变电阻器的一端连接到 3.3V，另一端接地，这就允许中间的连接可以设置为 0 到 3.3V 之间的任意电压。

在实验该程序之前，一定确保启用了 SPI，并且安装好了 Python 库 SPI（见 9.5 节）。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 adc_test.py。

```
import spidev, time

spi = spidev.SpiDev()
spi.open(0, 0)

def analog_read(channel):
    r = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((r[1]&3) << 8) + r[2]
    return adc_out

while True:
    reading = analog_read(0)
    voltage = reading * 3.3 / 1024
    print("Reading=%d\tVoltage=%f" % (reading, voltage))
```

```
time.sleep(1)
```

对于这个程序来说，我们感兴趣的部分位于 `analog_read` 函数中。该函数需要一个介于 0 到 7 之间的数字作为参数，用以指示应该从芯片左下部的 8 个模拟输入中的哪一个读取数值。

然后，通过位操作给相应的通道设置一个请求，并将比特数据发送给 MCP3008，该芯片会读取最终数据。

```
$ sudo python adc_test.py
Reading=0      Voltage=0.000000
Reading=126    Voltage=0.406055
Reading=221    Voltage=0.712207
Reading=305    Voltage=0.982910
Reading=431    Voltage=1.388965
Reading=527    Voltage=1.698340
Reading=724    Voltage=2.333203
Reading=927    Voltage=2.987402
Reading=1020   Voltage=3.287109
Reading=1022   Voltage=3.293555
```

进一步探讨

MCP3008 是 10 比特的 ADC，所以当你每次读取的时候，将会返回一个介于 0 到 1023 之间的一个数字。该测试程序将会把它转化为电压值，方法是将读数乘以电压范围 (3.3V)，然后除以 1024。你可以联合使用下面用到 MCP3008 的示例，从而实现从最多 8 个传感器上读取数值的效果。

此外，你还可以使用电阻式传感器和 MCP3008，将两者与一个定值电阻结合起来构成一个分压器（见 13.6 节和 13.7 节）。

参考资料

如果你只对侦测旋钮的转动感兴趣的话，可以使用旋转编码器来代替电位器（见 12.7 节）。

此外，你也可以使用 ADC 芯片而非阶跃响应方式（见 13.1 节）来检测电位器的位置（见 13.1 节）。

要想查看 MCP3008 的参数手册，请访问 <http://bit.ly/I1lSqn>。

此外，Pimoroni 的 Explorer HAT Pro 也具有 ADC（见 9.17 节）。

13.6 为测量而降低电压

面临问题

你想测量一个电压，但是该电压高于 MCP3008 所能承受的 3.3V（见 13.5 节）。

解决方案

你可以使用两个电阻作为分压器，把电压降低到合适的范围之内。

为了进行下面的实验，你将需要：

- 面包板和跳线；
- MCP3008 八通道 ADC IC；
- 10k Ω 电阻；
- 3.3k Ω 电阻；
- 9V 电池和夹线。

图 13-12 展示了该实验中面包板的布局。该配置将用于测量电池的电压。

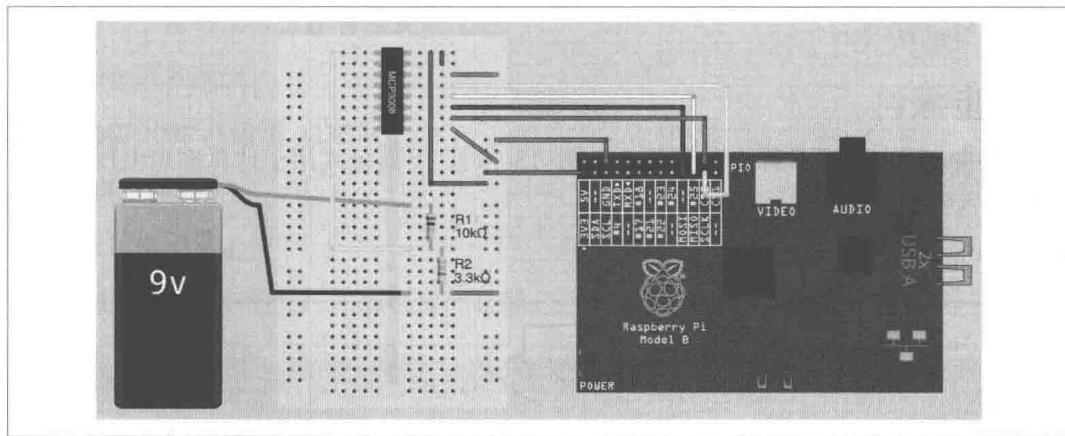


图 13-12 降低模拟输入的电压

绝不要利用本示例来测量高压交流电，以及其他任何类型的交流电。本例仅适用于低压直流电。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 `adc_scaled.py` 。

```
import spidev  
  
R1 = 10000.0  
R2 = 3300.0  
  
spi = spidev.SpiDev()  
spi.open(0,0)  
  
def analog_read(channel):
```

```

r = spi.xfer2([1, (8 + channel) << 4, 0])
adc_out = ((r[1]&3) << 8) + r[2]
return adc_out

reading = analog_read(0)
voltage_adc = reading * 3.3 / 1024
voltage_actual = voltage_adc / (R2 / (R1 + R2))
print("Battery Voltage=" + str(voltage_actual))

```

该程序与 13.5 节中的程序非常类似。主要区别在于使用了两个不同组织的电阻进行缩放。这两个电阻的阻值分别保存在变量 R1 和 R2 中。当你运行该程序的时候，将会显示电池的电压。

```

$ sudo python adc_scaled.py
Battery Voltage=8.62421875

```

在连接任何高于 9V 的元件之前，请仔细阅读“进一步探讨”部分的内容，否则就有可能损坏 MCP3008。

进一步探讨

本节中的电阻布局通常称为分压器（见图 13-13）。下面公式用于计算输出电压，需要提供输入电压和两个电阻的值。

$$V_{out} = V_{in} \times R_2 / (R_1 + R_2)$$

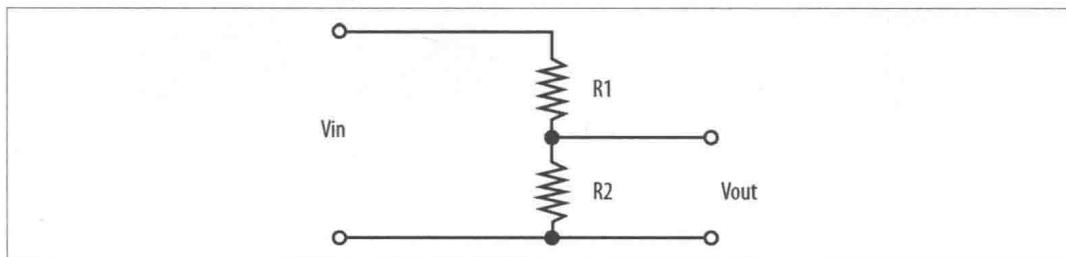


图 13-13 分压器

也就是说，如果 R1 和 R2 的值相同的话（例如 $1k\Omega$ ），那么 V_{out} 就是 V_{in} 的二分之一。当你选择 R1 和 R2 的时候，还需要考虑通过 R1 和 R2 的电流。该电流为 $V_{in}/(R_1+R_2)$ 。在前面的例子中，R1 是 $10k\Omega$ ，而 R2 是 $3.3k\Omega$ ，所以，电流为 $9V/13.3k\Omega = 0.68ma$ 。虽然这个电流很小，但仍足以耗尽电池的电量，所以不要总是将其保持连接状态。

参考资料

为了避免数学计算，你可以使用一个在线阻值计算器 (<http://bit.ly/17EtrJI>)。

当使用电阻式传感器与 ADC 时（见 13.7 节），分压器也可以用来将电阻值转换为电压值。

13.7 使用电阻式传感器与 ADC

面临问题

你有一个电阻式传感器，并且希望将它与 MCP3008 ADC 芯片一起使用。

解决方案

你可以使用带有一个固定阻值电阻的分压器和电阻式传感器，将传感器的电阻值转换为可以用 ADC 测量的电压。

作为一个示例，你可以将 13.2 节中使用阶跃响应技术的光传感器的项目改为使用 MCP3008。

为了进行下面的实验，你将需要：

- 面包板和跳线；
- MCP3008 八通道 ADC IC；
- $10k\Omega$ 电阻；
- 光敏电阻器。

图 13-14 展示了本例使用的面包板的布局。

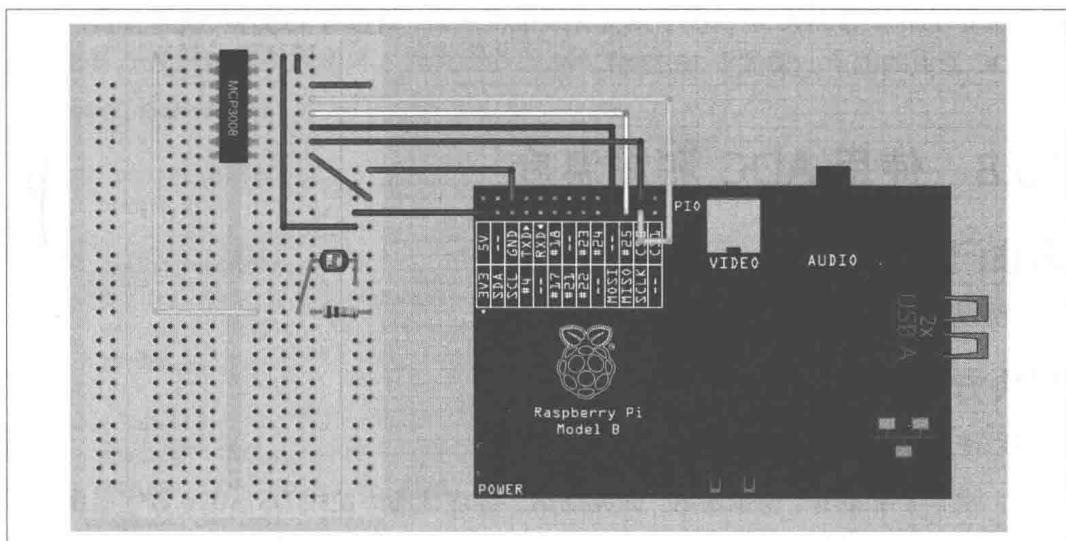


图 13-14 使用光敏电阻器和 ADC

你可以原封不动地使用 13.5 节中的程序代码 (adc_test.py)。当你用手遮住光传感器的

时候，读数就会随之改变。同时，你还需要在树莓派上面设置 SPI，如果尚未设置的话，请参考 9.5 节的相关介绍。

```
$ sudo python adc_test.py
```

```
Reading=341    Voltage=1.098926
Reading=342    Voltage=1.102148
Reading=227    Voltage=0.731543
Reading=81     Voltage=0.261035
Reading=86     Voltage=0.277148
```

这些读数可能会有较大的不同，主要取决于你的光敏电阻器，但是重点在于数字会随着亮度的变化而变化。

进一步探讨

定值电阻器的选择并不是非常重要。如果阻值太高或太低的话，你会发现读数的范围会非常小。你可以在电阻传感器的最大值和最小值之间选择一个电阻值。你可能需要先做几次试验，才能决定哪个电阻适用于你关心的读数范围。如果拿不准的话，可以从 $10\text{k}\Omega$ 开始着手，看看效果如何。

你几乎可以将光敏电阻器换成任何电阻式的传感器。因此，举例来说，你可以使用 13.4 节中的气体传感器。

参考资料

如果想要在不借助 ADC 的情况下测量光线强度的话，请参考 13.2 节。关于一次使用多个 ADC 通道的例子，请参考 13.12 节。

13.8 使用 ADC 测量温度

面临问题

你想使用 TMP36 和一个模数转换器来测量温度。

解决方案

你可以使用 MCP3008 ADC 芯片。

可是，除非你需要多个模拟通道，否则的话，你应该考虑使用 DS18B20 数字温度传感器，因为它更加准确，并且不需要单独的 ADC 芯片（见 13.11 节）。

为了进行下面的实验，你需要：

- 面包板和跳线；

- MCP3008 八通道 ADC IC；
- TMP36 温度传感器。

图 13-15 展示了本例使用的面包板的布局。

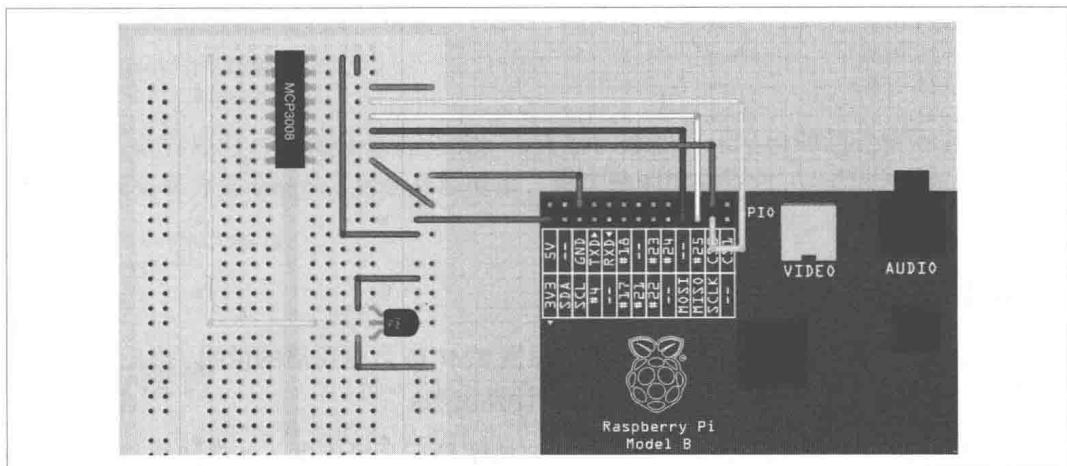


图 13-15 使用 TMP36 与 ADC

请务必确保 TMP36 的朝向正确无误。其封装的一面是平坦的，而另一面是弯曲的。

你需要在树莓派上面设置 SPI，如果还没有的话，请参考 9.5 节的相关内容。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 adc_tmp.py 。

```
import spidev, time

spi = spidev.SpiDev()
spi.open(0,0)

def analog_read(channel):
    r = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((r[1]&3) << 8) + r[2]
    return adc_out

while True:
    reading = analog_read(0)
    voltage = reading * 3.3 / 1024
    temp_c = voltage * 100 - 50
    temp_f = temp_c * 9.0 / 5.0 + 32
    print("Temp C=%f\tTemp f=%f" % (temp_c, temp_f))
    time.sleep(1)
```

该程序是基于 13.8 节中的示例代码的，只是多了一点关于摄氏温度和华氏温度的数学计算。

```
$ sudo python adc_tmp36.py
Temp C=19.287109      Temp f=66.716797
Temp C=18.642578      Temp f=65.556641
Temp C=18.964844      Temp f=66.136719
Temp C=20.253906      Temp f=68.457031
Temp C=20.898438      Temp f=69.617188
Temp C=20.576172      Temp f=69.037109
Temp C=21.865234      Temp f=71.357422
Temp C=23.154297      Temp f=73.677734
Temp C=23.476562      Temp f=74.257812
Temp C=23.476562      Temp f=74.257812
Temp C=24.121094      Temp f=75.417969
Temp C=24.443359      Temp f=75.998047
Temp C=25.087891      Temp f=77.158203
```

进一步探讨

TMP36 会输出一个与稳定成正比的电压。根据 TMP36 数据手册的说明，摄氏温度可以通过电压（以 V 为单位）乘以 100 减去 50 计算出来。

TMP36 非常适用于计算近似温度，不过额定精度只有 2 摄氏度。如果使用长导线来连接它的话，精度将会更差。从某种程度上讲，你可以校正单个设备，但是要提高精度，需要使用一个 DS18B20（见 13.11 节），它在-10 到+85 摄氏度范围内的标称精度为 0.5%。由于其是一个数字设备，所以在使用长导线连接的情况下也不会损失精度。

参考资料

关于 TMP36 的参数手册，请访问 <http://bit.ly/1b3Mflm>。

若想利用热敏电阻测量温度，请参考 13.3 节。

要想使用数字温度传感器（DS18B20）来测量温度，请参考 13.11 节。

要想使用 Sense HAT 来测量温度，请参考 13.10 节。

13.9 测量树莓派的 CPU 温度

面临问题

你想要知道树莓派的 CPU 目前到底有多么热。

解决方案

你可以使用 os 库访问 Broadcom 芯片内置的温度传感器。

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载

相应的程序文件，就本例来说，该程序名为 `cpu_temp.py`。

```
import os, time

while True:
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp = dev.read()
    print(cpu_temp)
    time.sleep(1)
```

当你运行此程序时，它就会报告温度。请注意，这里打印输出的消息实际上是一个字符串，开头部分是 `temp=`，后面是温度值，最后是温度单位'C'。

```
$ python cpu_temp.py
temp=33.6'C
temp=33.6'C
```

进一步探讨

如果你想利用数字而非字符串来表示温度的话，可以去掉多余的文本，并将数字转换为浮点数。对此，有一个专门的示例程序，名为 `cpu_temp_float.py`。

```
import os, time

while True:
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp_s = dev.read()[5:-3] # top and tail string
    cpu_temp = float(cpu_temp_s)
    print(cpu_temp)
    time.sleep(1)
```

参考资料

关于截取字符串的详细介绍，请参考 5.15 节。

若想利用热敏电阻测量温度，请参考 13.3 节。

要想利用 TMP36 来测量温度，请参考 13.8 节。

关于利用数字温度传感器（DS18B20）测量温度的说明，请参考 13.11 节。

要想使用 Sense HAT 来测量温度，请参考 13.10 节。

13.10 利用 Sense HAT 测量温度、湿度和气压

面临问题

你想测量温度、湿度和气压，但是又不想使用 3 个独立的传感器。

解决方案

你可以使用树莓派的 Sense HAT (见图 13-16)。这样，你不仅获得了 3 种传感器，同时还有一个类似显示器的东西。

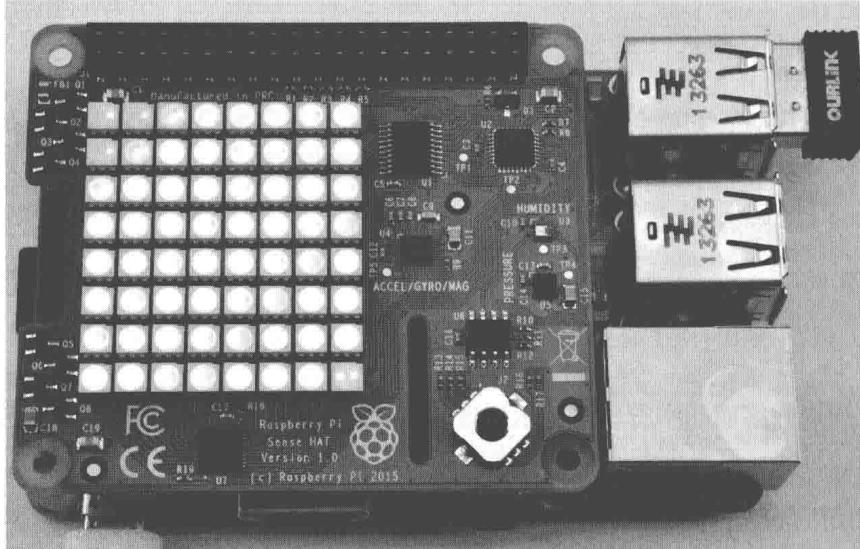


图 13-16 Sense HAT

从 9.16 节开始，就需要安装 Sense HAT 所需的程序库了。

打开一个编辑器 (nano 或者 IDLE)，并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件，就本例来说，该程序名为 `sense_hat_thp.py`。

```
from sense_hat import SenseHat
import time

hat = SenseHat()

while True:
    t = hat.get_temperature()
    h = hat.get_humidity()
    p = hat.get_pressure()
    print('Temp C:{:.2f} Hum:{:.0f} Pres:{:.0f}'.format(t, h, p))
    time.sleep(1)
```

当你运行该程序的时候，终端就会输出类似下面这样的内容。

```
$ sudo python sense_hat_thp.py
Temp C:27.71 Hum:56 Pres:1005
Temp C:27.60 Hum:55 Pres:1005
```

其中，温度以摄氏度为单位，湿度是相对湿度百分比，气压以毫巴为单位。

进一步探讨

你会发现，Sense HAT 的温度读数会偏高，这是因为温度传感器内置在湿度传感器中，并且位于 Sense HAT PCB 电路板上。虽然 Sense HAT 的发热量很小（除非你用了显示器），但位于该 Sense HAT 下面的树莓派却会变热，从而增加了该 HAT 的温度。避免这个问题的最好的方法是使用一个 40 路的带状电缆让 Sense HAT 远离树莓派。此外，你还可以尝试利用树莓派的温度读数来修正该读数，具体请参考 <http://bit.ly/1ofeewf> 上面的讨论。就个人而言，我觉得这些校正尝试可能仅仅适用于发帖用户自身的特定情况，但是不太可能产生可靠的结果。

就像从湿度传感器中读取温度一样，压力传感器也提供了一个内置的温度传感器，你可以通过下面的方法来读取。

```
t = hat.get_temperature_from_pressure()
```

目前，尚未有文档明确指出该读数是否比使用湿度传感器更准确，但根据我的实验情况看，它报告的温度通常会比湿度传感器的读数低 1 摄氏度左右。

参考资料

关于 Sense HAT 的入门知识，请参考 9.16 节。

Sense HAT 的编程指南请参考 <https://pythonhosted.org/sense-hat/api/>。

此外，Sense HAT 还为导航类型项目提供了一个加速度计、陀螺仪（见 13.13 节）和磁力仪（见 13.14 节）。同时，它还提供了一个 8×8 全彩 LED 矩形显示设备（见 14.3 节）。

13.11 利用数字传感器测量温度

面临问题

你想利用精确的数字传感器来测量温度。

解决方案

你可以使用 DS18B20 数字温度传感器。这个设备比 13.8 节中使用的 TMP36 更加精确，并且使用数字接口，所以它无需 ADC 芯片。

尽管该芯片的接口称为单线，但是它仅仅是针对数据引脚而言的。

所以，你至少还需要另外一根导线来连接单线设备。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- DS18B20 温度传感器；
- $4.7\text{k}\Omega$ 电阻器。

请按照图 13-17 展示的方式将各个元件安装到面包板上面。请务必确保 DS18B20 朝向的正确性。对于最新版的 Raspbian 来说，它已经支持 DS18B20 所用的单线接口，但是，你必须启用它才行。

```
$ dtoverlay=w1-gpio
```

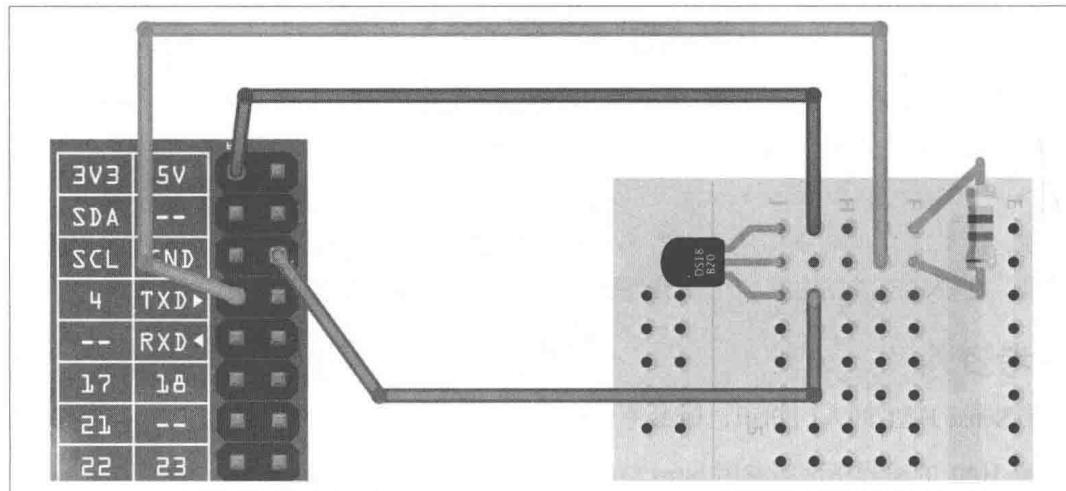


图 13-17 将一个 DS18B20 连接到树莓派

打开一个编辑器（ nano 或 IDLE ），并粘贴如下所示的代码。就像本书中的其他代码示例一样，你可以在本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件。就本例而言，该程序名为 temp_DS18B20.py 。

```
import glob, time

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
    s = lines[0] + '\n' + lines[1]
    temp_index = s.find('t=')
    if temp_index != -1:
        temp_string = s[temp_index+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_f
```

```

lines = read_temp_raw()
equals_pos = lines[1].find('t=')
if equals_pos != -1:
    temp_string = lines[1][equals_pos+2:]
    temp_c = float(temp_string) / 1000.0
    temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f

while True:
    print("temp C=%f\ttemp F=%f" % read_temp())
    time.sleep(1)

```

当程序运行时，每隔 1 秒就会同时使用摄氏温度和华氏温度报告一次温度值。

```

$ python temp_DS18B20.py
temp C=25.187000 temp F=77.336600
temp C=25.125000      temp F=77.225000
temp C=25.062000      temp F=77.111600
temp C=26.312000      temp F=79.361600
temp C=27.875000      temp F=82.175000
temp C=28.875000      temp F=83.975000

```

进一步探讨

乍一看，这个程序好像有点奇怪。PSI8B20 使用了一个类似文件一样的接口。设备的文件接口部是位于/sys/bus/wl/devices 目录下面，并且文件路径名称总是以 28 起始，但是文件路径的其余部分则会因传感器而异。

这里的代码假设只有一个传感器，并寻找第一个以 28 开头的文件来，要使用多个传感器，请在方括号内使用不同的索引值。

在该文件夹中会有一个名为 w1_slave 的文件，为了查找温度读数，程序会打开并读取该文件。

传感器返回的字符串文本如下所示：

```

81 01 4b 46 7f ff 0f 10 71 : crc=71 YES
81 01 4b 46 7f ff 0f 10 71 t=24062

```

代码的其余部分，会从这个消息中提取温度部分。温度部分位于 t=后面，其单位是千分之一摄氏度。

函数 read_temp 会计算温度，并返回以摄氏温度和华氏温度的读数。

除了 DS18B20 的基本芯片版本外，你还可以购买封装在坚固耐用的防水封装版本。

参考资料

要了解记录读数的方法，请参考 13.20 节。

本节内容主要来自于 Adafruit 教程 (<http://bit.ly/1ioBYbh>)。

关于 DS18B20，请查看其说明书 (<http://bit.ly/17EveOI>)。

关于使用热敏电阻测量温度的介绍，请参考 13.3 节。

关于使用 TMP36 测量温度的方法，请参考 13.8 节。

关于使用 Sense HAT 测量温度的方法，请参考 13.10 节。

13.12 利用 MCP3008 模块测量加速度

面临问题

你想给树莓派连接一个三轴加速度计。

解决方案

你可以使用一个模拟加速度计和 MCP3008 ADC 芯片来测量 X、Y 和 Z 轴的模拟输出。

为了进行下面的实验，你将需要：

- 面包板和跳线；
- MCP3008 八通道 ADC IC；
- ADXL335 三轴加速度计。

图 13-18 展示了本示例所用的面包板的布局。本示例使用三通道 ADC 来测量 X、Y 和 Z 的加速力。

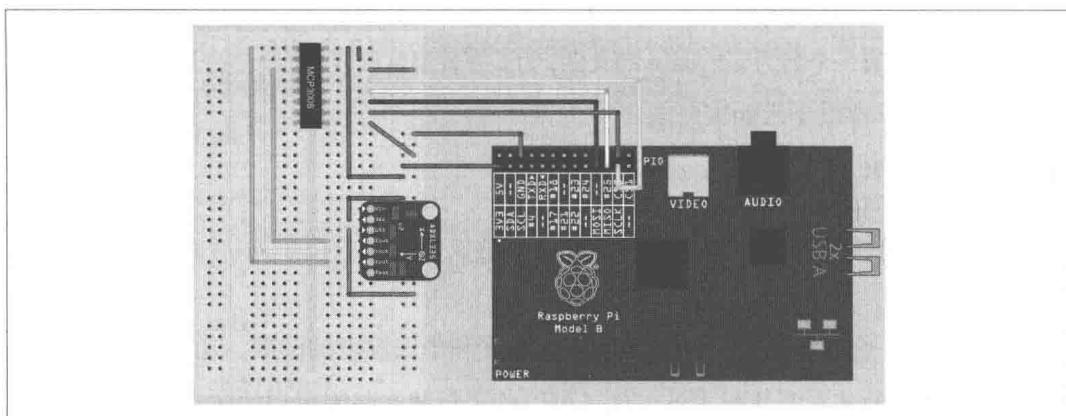


图 13-18 使用三轴加速度计

你需要在树莓派上面设置 SPI，所以，如果尚未设置的话，请参考 9.5 节中的相关介绍。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 `adc_accelerometer.py` 。

```
import spidev, time

spi = spidev.SpiDev()
spi.open(0,0)

def analog_read(channel):
    r = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((r[1]&3) << 8) + r[2]
    return adc_out

while True:
    x = analog_read(0)
    y = analog_read(1)
    z = analog_read(2)
    print("X=%d\ty=%d\tz=%d" % (x, y, z))
    time.sleep(1)
```

该程序只是简单读取 3 个力，并将其打印输出。

```
$ sudo python adc_accelerometer.py
X=508  Y=503  Z=626
X=508  Y=504  Z=624
X=506  Y=505  Z=627
X=423  Y=517  Z=579
X=411  Y=513  Z=548
X=532  Y=510  Z=623
X=609  Y=518  Z=495
X=607  Y=521  Z=496
X=610  Y=513  Z=499
```

前 3 个数值是加速度计在水平状态下的读数。接下来的 3 个数值是将整个面包板倾斜的时候的读数。你可以看到， X 的读数变小了。如果将面包板向另一边倾斜的话，就会导致 X 的读数变大。

进一步探讨

加速度计最常见的用途就是检测倾斜。之所以能够检查倾斜，是因为 Z 轴加速力会受到重力的主宰（见图 13-19）。

当加速度计在一个方向上倾斜的时候，垂直向下的重力就会作用在加速度计的其他轴上面。

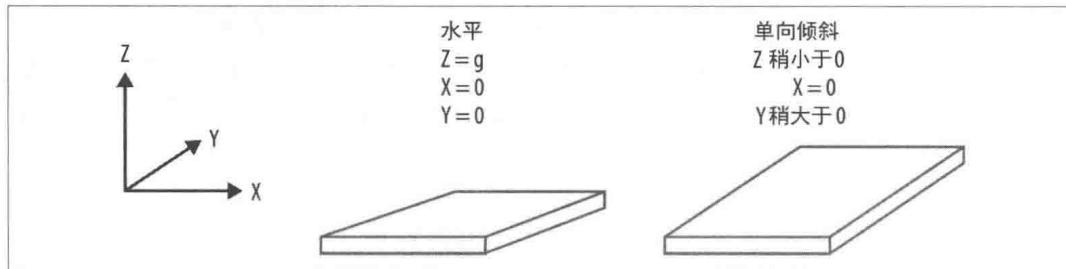


图 13-19 利用加速度计检测倾斜

我们可以利用这个原理来检测何时倾斜超过了指定的阀值。下面的程序 (tilt.py) 将对此进行阐释。

```
import spidev, time

spi = spidev.SpiDev()
spi.open(0,0)

def analog_read(channel):
    r = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((r[1]&3) << 8) + r[2]
    return adc_out

while True:
    x = analog_read(0)
    y = analog_read(1)
    z = analog_read(2)
    if x < 450:
        print("Left")
    elif x > 550:
        print("Right")
    elif y < 450:
        print("Back")
    elif y > 550:
        print("Forward")
    time.sleep(0.2)
```

当你运行该程序的时候，就会看到方向消息。你可以用这个代码来控制一个漫游机器人或连接了网络摄像头的电动云台。

```
$ sudo python tilt.py
Left
Left
Right
Forward
Forward
Back
Back
```

参考资料

关于该模块所用的加速度计芯片的参数手册，请参考 <http://bit.ly/19zQzZq>。

此外，还有许多其他类型的模拟加速度计模块，你可能发现它们的读数并不完全一致。请确保模拟输出不要超过 3.3V。

Sense HAT 也包含了一个加速度计（见 13.13 节）。

13.13 使用 Sense HAT 的惯性管理单元 (IMU)

面临问题

你希望通过树莓派获得比 13.12 节中加速度计提供的读数更加精确的方位信息。

解决方案

你可以使用 Sense HAT 的惯性管理单元 (IMU)。这个单元不仅包含了类似 13.12 节中使用的三轴加速度计，而且还提供了三轴陀螺仪和磁力仪。通过将这 3 个不同传感器的读数组合起来，你就能够获得 Sense HAT 更加准确的方位信息，通常用俯仰角、滚转角和偏航角来表示，具体如图 13-20 所示。

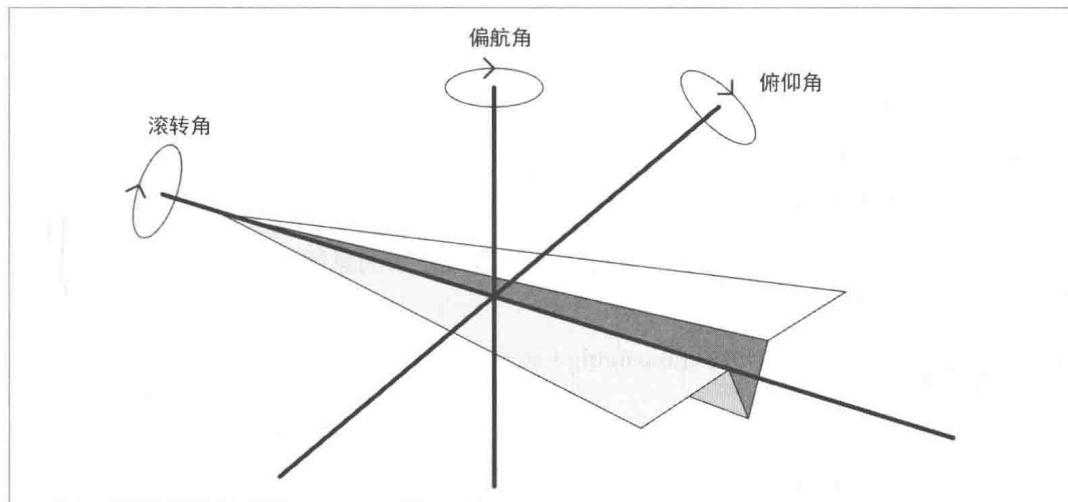


图 13-20 俯仰角、滚转角和偏航角

俯仰角、滚转角和偏航角是来自航空领域的 3 个术语，它们都是相对于飞机的飞行而言的。俯仰角是相对于水平面的夹角，滚转角是绕飞机的飞行轴线的旋转角度（设想一个机翼向上，而另一个向下），偏航角是针对水平轴的左右夹角（想想改变方

向的情形)。

打开一个编辑器(nano 或者 IDLE), 并粘贴如下所示的代码。就像本书中的其他示例代码一样, 你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件, 就本例来说, 该程序名为 sense_hat_orientation.py。

```
from sense_hat import SenseHat

sense = SenseHat()

sense.set_imu_config(True, True, True)

while True:
    o = sense.get_orientation()
    print("p: {:.0f}, r: {:.0f}, y: {:.0f}".format(o['pitch'], o['roll'],
o['yaw']))
```

函数 set_imu_config 函数规定在罗盘、陀螺仪和加速度计(按照此顺序)中使用哪些仪器来测量方位。如果将 3 种仪器都设为 True, 则意味着它们将都参与测量。

当你运行该程序的时候, 将看到类似下面这样的输出内容。

```
$ sudo python sense_hat_orientation.py
p: 1, r: 317, y: 168
p: 1, r: 318, y: 169
```

如果你朝着 USB 端口的方向倾斜 Sense HAT 和树莓派的话, 就会看到俯仰角的数值将随之增加。

进一步探讨

一个加速度计可以测量静止物体的受力, 因此可以通过测量重力(Z 轴)在 X 轴和 Y 轴上面分力的大小来计算倾斜程度。

陀螺仪的工作原理有所不同, 它是通过科里奥利效应来测量绕运动方向转动的同时来回摇摆的运动物体的受力的。

参考资料

关于 Sense HAT 的 IMU 的详细信息, 请参考 [https://www.raspberrypi.org/ learning/astro-pi-guide/sensors/movement.md](https://www.raspberrypi.org/learning/astro-pi-guide/sensors/movement.md)。

要想测量温度、湿度和气压, 请参考 13.10 节。

此外, Sense HAT 的 IMU 还可以用来制作罗盘, 以及用来感应磁场(见 13.16 节)。

想了解更多关于陀螺仪和科里奥利效应的内容, 请访问 https://en.wikipedia.org/wiki/coriolis_efect。

13.14 利用 Sense HAT 寻找磁北

面临问题

你想利用 Sense HAT 来检测磁北。

解决方案

你可以通过 Sense HAT 中内置的 3 轴磁力仪对应的 Python 库来寻找磁北。

首先，你需要按照 9.16 节介绍的方法来安装 Sense HAT 库。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 sense_hat_compass.py 。

```
from sense_hat import SenseHat

sense = SenseHat()

while True:
    bearing = sense.get_compass()
    print('Bearing: {:.0f} to North'.format(bearing))
```

当你运行这个程序的时候，将看到一系列的方位读数。

```
$ sudo python sense_hat_compass.py
Bearing: 138 to North
Bearing: 138 to North
```

进一步探讨

罗盘会对其附近的所有磁场都非常敏感，所以你可能很难得到准确的方位。

参考资料

关于 Sense HAT 的相关文档，请访问 <https://github.com/RPi-Distro/python-sense-hat> 和 <http://pythonhosted.org/sense-hat/api/> 。

要想利用 Sense HAT 检测磁铁，请参考 13.16 节。

13.15 利用簧片开关检测磁铁

面临问题

你想检测是否存在磁铁。

解决方案

你可以使用一个簧片开关（见图 13-21）。它的工作方式与普通开关类似，只不过只有当靠近磁铁时才会被激发。

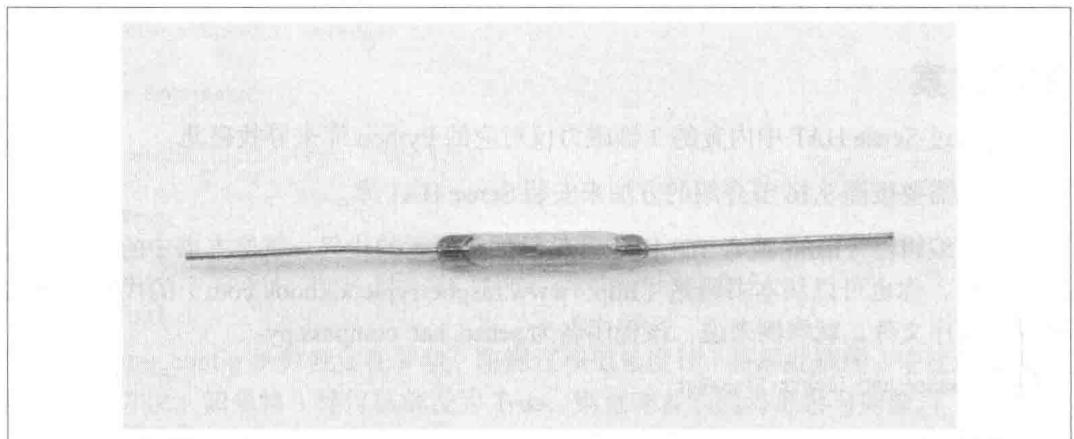


图 13-21 簧片开关

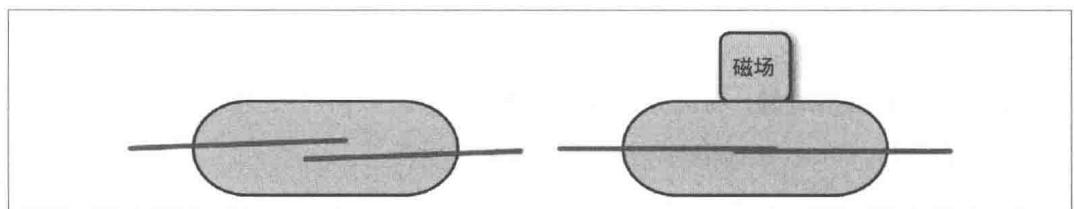


图 13-22 簧片开关的工作原理

在玻璃管中封装了两个簧片触点。当磁铁靠近这个簧片开关时，簧片就会吸合在一起，从而接通电路。

在第 12 章中从 12.1 节开始介绍的所有普通开关的示例代码，都可以用于本节。

进一步探讨

簧片开关是一种没有多大技术含量的磁铁检测方法。该方法自 20 世纪 30 年代就已经出现了，所以非常可靠。它们还用于某些安防系统中，即把塑料封装的簧片开关放到门框中，然后在门本身放入塑料封装的固定磁铁。当门打开时，簧片开关触点就会分开，从而触发报警。

参考资料

关于利用 Sense HAT 的磁力仪检测磁铁的方法，请参考 13.16 节。

13.16 利用 Sense HAT 感应磁场

面临问题

你想通过 Python 程序，利用 Sense Hat 内置的磁力仪来检测磁场。

解决方案

你可以通过 Sense HAT 的 Python 库来使用它的磁力仪。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 sense_hat_magnet.py 。

```
from sense_hat import SenseHat
import time

hat = SenseHat()
fill = (255, 0, 0)

while True:
    reading = int(hat.get_compass_raw()['z'])
    if reading > 200:
        hat.clear(fill)
        time.sleep(0.2)
    else:
        hat.clear()
```

当磁铁靠近 Sense HAT 时， LED 就会全部变红，持续时间为五分之一秒。

进一步探讨

使用哪个轴上的罗盘数据并不重要，因为存在固定磁铁的时候，它们都会受到极大干扰。

参考资料

要想利用簧片开关检测磁铁，可以参考 13.15 节。

关于 Sense HAT 显示器的其他使用方法，请参考 14.3 节。

13.17 测量距离

面临问题

你想利用超声波测距仪测量距离。

解决方案

你可以使用一个廉价的 SR-04 测距仪。这类设备需要 2 个 GPIO 引脚：一个引脚用来发射超声波脉冲，另一个引脚用来监视回波返回所用的时间。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- SR-04 测距仪（eBay）；
- 470Ω 电阻器；
- 270Ω 电阻器。

请按照图 13-23 所示将元件安装到面包板上面。为了将测距仪回波输出从 5V 降到 3.3V，电阻是必不可少的（见 9.12 节）。

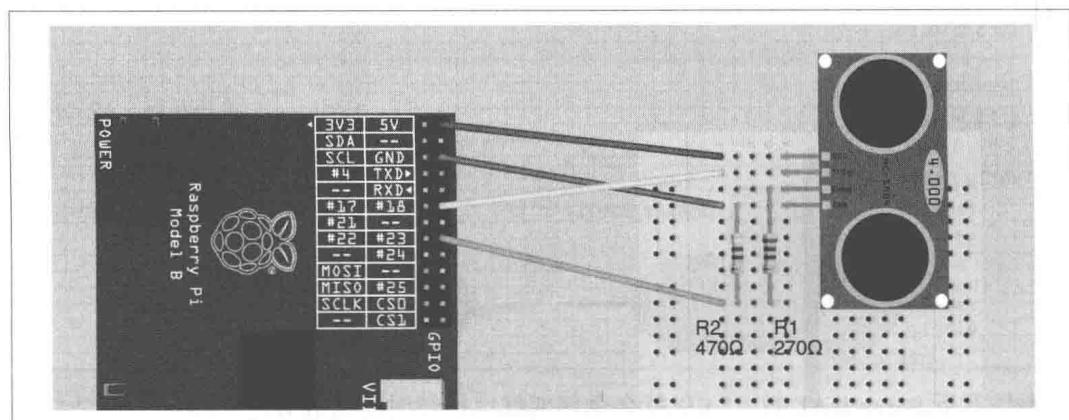


图 13-23 将 SR-04 测距仪连接至树莓派

打开一个编辑器（nano 或者 IDLE），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 ranger.py。

```
import RPi.GPIO as GPIO
import time

trigger_pin = 18
echo_pin = 23

GPIO.setmode(GPIO.BCM)
GPIO.setup(trigger_pin, GPIO.OUT)
GPIO.setup(echo_pin, GPIO.IN)

def send_trigger_pulse():
    GPIO.output(trigger_pin, True)
    time.sleep(0.00001)
    GPIO.output(trigger_pin, False)
```

```

GPIO.output(trigger_pin, True)
time.sleep(0.0001)
GPIO.output(trigger_pin, False)

def wait_for_echo(value, timeout):
    count = timeout
    while GPIO.input(echo_pin) != value and count > 0:
        count = count - 1

def get_distance():
    send_trigger_pulse()
    wait_for_echo(True, 10000)
    start = time.time()
    wait_for_echo(False, 10000)
    finish = time.time()
    pulse_len = finish - start
    distance_cm = pulse_len / 0.000058
    distance_in = distance_cm / 2.5
    return (distance_cm, distance_in)

while True:
    print("cm=%f\tinches=%f" % get_distance())
    time.sleep(1)

```

程序的用法将在“进一步探讨”中介绍。当程序运行时，它会每秒返回一次距离，分别以厘米和英寸为单位。当你使用手或其他障碍物挡住它时，读数就会发生相应的变化。

```

sudo python ranger.py
cm=154.741879  inches=61.896752
cm=155.670889  inches=62.268356
cm=154.865199  inches=61.946080
cm=12.948595   inches=5.179438
cm=14.087249   inches=5.634900
cm=13.741954   inches=5.496781
cm=20.775302   inches=8.310121
cm=20.224473   inches=8.089789

```

进一步探讨

尽管可选用的超声波测距仪的类型很多，但是这里使用的这种类型更简单易用，同时也更加廉价。它的工作机制是先发送超声波脉冲，然后测量接收到回波所用的时间。该设备前面的一个圆形超声换能器是发射装置，另一个是接收装置。这个过程是由树莓派进行控制的。这种类型的设备与其他更加昂贵的模块相比，不同之处在于更加昂贵的版本自身带有微控制器，该控制器能够完成所有所需的时间测量，并提供一个 I2C 或串行接口来返回最终的读数。

当你在树莓派上使用这些类型的传感器的时候，测距仪的输入 trig（触发器）要连接到一个 GPIO 输出上，同时测距仪的输出 echo 要连接到树莓派的 GPIO 输入上面，但是

首先要把电压从 5V 降到 3.3V。

图 13-24 展示了该传感器在工作时的示波器跟踪图。顶部（红色）的追踪图连接到 trig 引脚，而底部的（黄色）追踪图连接到 echo。你可以看到，最初 trig 引脚产生了一个高电平短波，然后，在 echo 引脚变为高电平短波之前有一个短暂延迟。之后，它会在高电平上面保持一段时间，时间长短与到传感器的距离成正比。

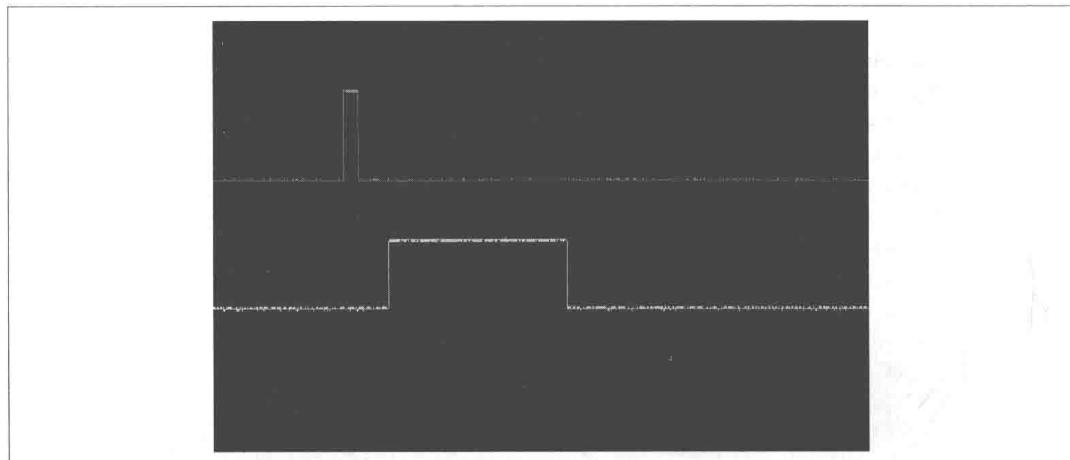


图 13-24 触发和回波的示波器跟踪图

用于这个传感器的代码首先会生成一个触发脉冲（使用函数 `send_trig_ger_pulse`）。它必须等到 echo 引脚变为高电平之后，才计算 echo 引脚保持高电平的时间。

之后，我们就可以使用回波脉冲持续时间和音速来计算距离了。

函数 `wait_for_echo` 会一直等待，直到 echo 引脚变为高电平或低电平为止，这取决于它的第一个参数。该函数的第二个参数提供了一个超时，避免由于某种原因导致 echo 引脚无法变为所等待的状态，这样循环就不会无限挂起了。

这种测距方法不是十分准确，因为温度、压力和相对湿度都会影响音速，从而影响距离读数的准确性。

参考资料

要想查看超声波测距仪的参数手册，请访问 <http://bit.ly/183GLgz>。

13.18 电容式触摸传感技术

面临问题

你希望给树莓派添加触摸式接口。

解决方案

你可以使用 Adafruit Capacitative Touch HAT (见图 13-25)。

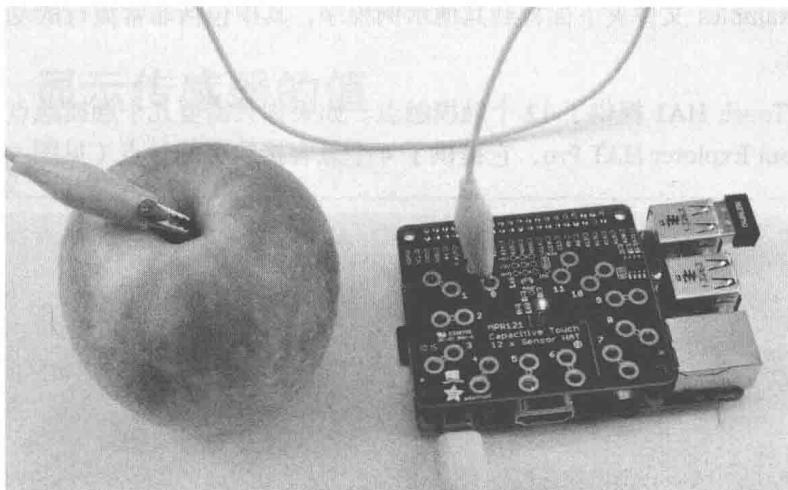


图 13-25 连接到苹果上面的 Adafruit Capacitative Touch HAT

触摸传感器不仅非常好玩儿，并且在教育方面用途极为广泛。你可以附加到任何东西上面，只要它们稍微导电即可，例如水果等。一个流行的项目是利用接线夹将各种水果和植物连接到该电路板的传感器接线端上面，从而打造一个水果键盘。这样，当你触摸不同的水果的时候，就会发出不同的声音。

Adafruit 的 Touch HAT 需要使用树莓派的 I2C 接口。此外，你还需要安装 SPI 软件，如果尚未安装的话，请参考 9.3 节。

为了安装这个 HAT 相应的 Python 库，可以使用如下所示的命令。

```
$ cd /home/pi  
$ git clone https://github.com/adafruit/Adafruit_Python_MPR121.git  
$ cd Adafruit_Python_MPR121/  
$ sudo python setup.py install
```

为了测试这个 Touch HAT，可以运行该库提供的示例代码 simpletest.py，具体命令如下所示。

```
$ cd examples  
$ sudo python simpletest.py  
Adafruit MPR121 Capacitive Touch Sensor Test  
Press Ctrl-C to quit.  
0 touched!  
0 released!
```

你可以直接触摸连接垫，或者通过接线夹将这些连接垫连接到水果上面，具体如图

13-25 所示。

进一步探讨

你可以从 examples 文件夹下面找到其他示例程序，其中包括非常流行的基于水果的音乐键盘示例。

Adafruit 的 Touch HAT 提供了 12 个触摸触点。如果你只需要几个触摸触点的话，可以使用 Pimoroni Explorer HAT Pro，它提供了 4 个兼容接线夹的触点（见图 13-26）。

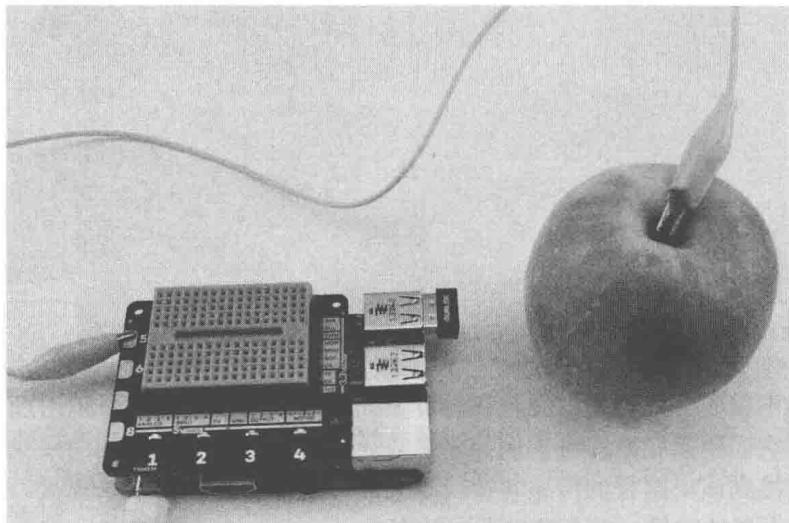


图 13-26 连接水果的 Explorer HAT Pro

要想使用 Explorer HAT Pro 的触摸触点，首先要按照 9.17 节中介绍的方法安装该 HAT 的库。

除了在一端提供了用于接线夹的 4 个接线端之外，它还提供了 4 个按钮开关，分别标有 1 到 4，供这个触摸接口使用。

为了尝试这个 Explorer HAT Pro，请打开 Python 终端，然后输入下列命令来检查触摸。

```
>>> import explorerhat  
Explorer HAT Pro detected...  
>>> explorerhat.touch.five.is_pressed()  
True  
>>> explorerhat.touch.five.is_pressed()  
False  
>>>
```

当你握住苹果的时候运行函数 `is_pressed()`，结果为 `True`；否则，结果为 `False`。

参考资料

关于 Adafruit Touch HAT 的文档, 请访问 <https://www.adafruit.com/products/2340>; 关于 Explorer HAT Pro 的文档, 请访问 <https://github.com/pimoroni/explorer-HAT>。

13.19 显示传感器的值

面临问题

你已经在树莓派上面连接了一个传感器, 但是希望在屏幕上用大数字显示读数。

解决方案

你可以使用 Tkinter 库打开一个窗口, 并将读数以 label 形式写入窗口中, 并且显示的时候使用较大字体 (见图 13-27)。

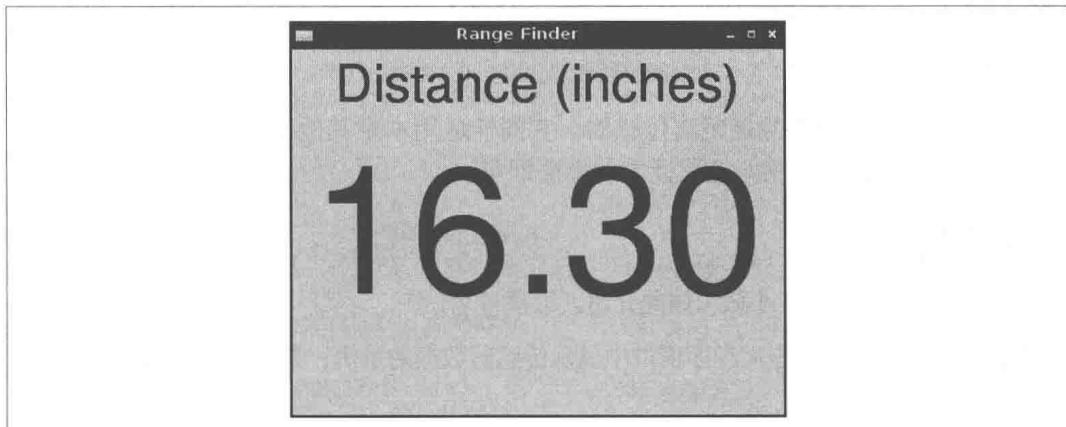


图 13-27 利用 Tkinter 显示传感器的读数

本示例使用的是来自 13.17 节中的超声波测距仪的数据。所以, 如果你想尝试这个实验的话, 需要首先阅读 13.17 节。

为了测试测距仪, 请打开一个编辑器 (nano 或者 IDLE), 并粘贴如下所示的代码。就像本书中的其他示例代码一样, 你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载相应的程序文件, 就本例来说, 该程序名为 `gui_sensor_reading.py`。使用超声波传感器测量距离的代码与 13.17 节中的是一样的, 因此这里只给出与显示读数相关的代码:

```
class App:  
  
    def __init__(self, master):  
        self.master = master
```

```
frame = Frame(master)
frame.pack()
label = Label(frame, text='Distance (inches)', font=("Helvetica", 32))
label.grid(row=0)
self.reading_label = Label(frame, text='12.34', font=("Helvetica", 110))
self.reading_label.grid(row=1)
self.update_reading()

def update_reading(self):
    cm, inch = get_distance()
    reading_str = "{:.2f}".format(inch)
    self.reading_label.configure(text=reading_str)
    self.master.after(500, self.update_reading)

root = Tk()
root.wm_title('Range Finder')
app = App(root)
root.geometry("400x300+0+0")
root.mainloop()
```

进一步探讨

虽然这个示例代码使用的是距离传感器，它同样适用本章其他传感器的相关示例。你只需要修改标签和从传感器获取读取的方法即可。

参考资料

有关将数字格式化为指定小数位数的介绍，请参考 7.1 节。

关于在 Web 浏览器而非应用程序窗口中显示传感器数据的示例，请参考 15.2 节。

13.20 利用 USB 闪存驱动器记录日志

面临问题

你想将传感器产生的测量数据记录到 USB 闪存驱动器上面。

解决方案

你可以编写一个 Python 程序将数据写到 USB 闪存驱动器上面。通过以 CSV 格式写文件，你可以将文件直接导入电子表格软件中，包括树莓派上面的 Gnumeric（见 4.2 节）。

下面的示例程序将会把从 DS18B20 中读取的温度记录到日志中。因此，如果你想试用该程序的话，请先阅读 13.11 节中的相关内容。打开一个编辑器（nano 或者 IDLE），

并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（<http://www.raspberrypicookbook.com>）的代码区下载相应的程序文件，就本例来说，该程序名为 temp_log.py。

```
import os, glob, time, datetime

log_period = 600 # seconds

logging_folder = glob.glob('/media/*')[0]
dt = datetime.datetime.now()
file_name = "temp_log_{:%Y_%m_%d}.csv".format(dt)
logging_file = logging_folder + '/' + file_name

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f

def log_temp():
    temp_c, temp_f = read_temp()
    dt = datetime.datetime.now()
    f = open(logging_file, 'a')
    f.write('\n{:H}:{:M}:{:S}'.format(dt))
    f.write(str(temp_c))
    f.close()

print("Logging to: " + logging_file)
while True:
```

```
log_temp()  
time.sleep(log_period)
```

这个程序被设置为每 10 分钟（600 秒）记录一次温度。你可以通过修改 log_period 的值来改变设置。

进一步探讨

当你将一个 USB 闪存驱动器插入树莓派时，它会自动将该驱动器安装到/media 下面。如果树莓派上面连接了多个可移动驱动器的话，那么该程序将使用在/media 下面找到的第一个文件夹。日志文件的名称由当前日期构成。

如果使用 Open Office 之类的电子表格来打开文件的话，那么你就可以直接编辑它。你的电子表格软件会让你指定数据的分隔符，例如可以使用逗号。

图 13-28 展示了使用本节代码捕获的数据集合，并且结果文件已经使用树莓派上运行的 Gnumeric 电子表格软件打开了。

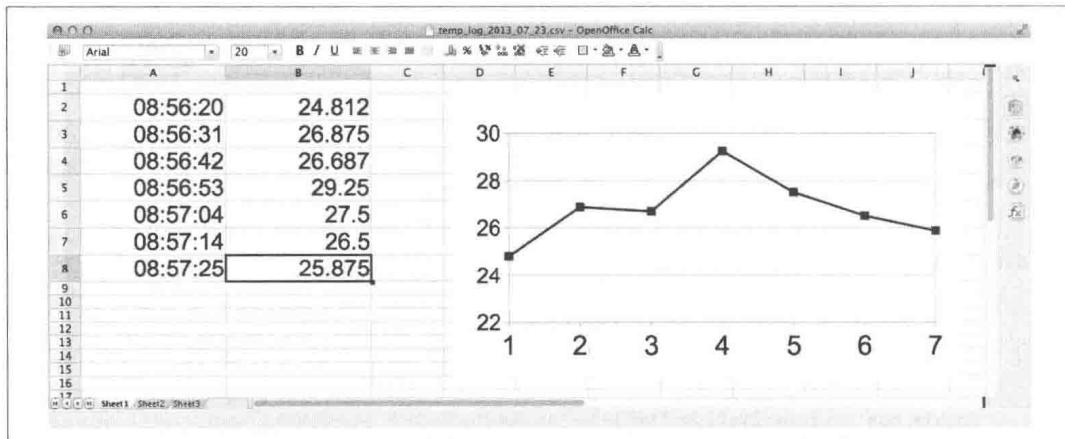


图 13-28 图表数据

参考资料

稍做修改，这个程序就可以适用于本章使用的其他传感器。

关于将传感器数据记录到 Web 服务上面的例子，请参考 15.6 节。

显示设备

14.0 引言

虽然树莓派可以选择显示器和 TV 作为显示设备,但是最好还是使用更加小巧和专用的显示设备。在本章中,我们将考察可供树莓派选用的各种显示设备。

其中,某些示例需要用到免焊面包板和 Female-to-Female (母头转母头) 跳线(见 9.8 节)。

14.1 使用四位 LED 显示设备



请务必观看在 <http://razzpisampler.oreilly.com> 上与本节有关的视频。

面临问题

你想使用一个老式的七段发光二极管显示器来显示一个四位数字。

解决方案

你可以使用一个 I2C LED 模块,如图 14-1 中展示的那样,使用 Female-to-Female (母头转母头) 跳线连接到树莓派上面。

为了进行本节中的试验,你需要:

- 4 个 Female-to-Female (母头转母头) 跳线;
- Adafruit 背负 I2C 的 4×7 段 LED;

树莓派与该模块之间的连接如下所示。

- 该显示设备上面的 VCC (+) 连接到树莓派 GPIO 接口上面的 5V 引脚。

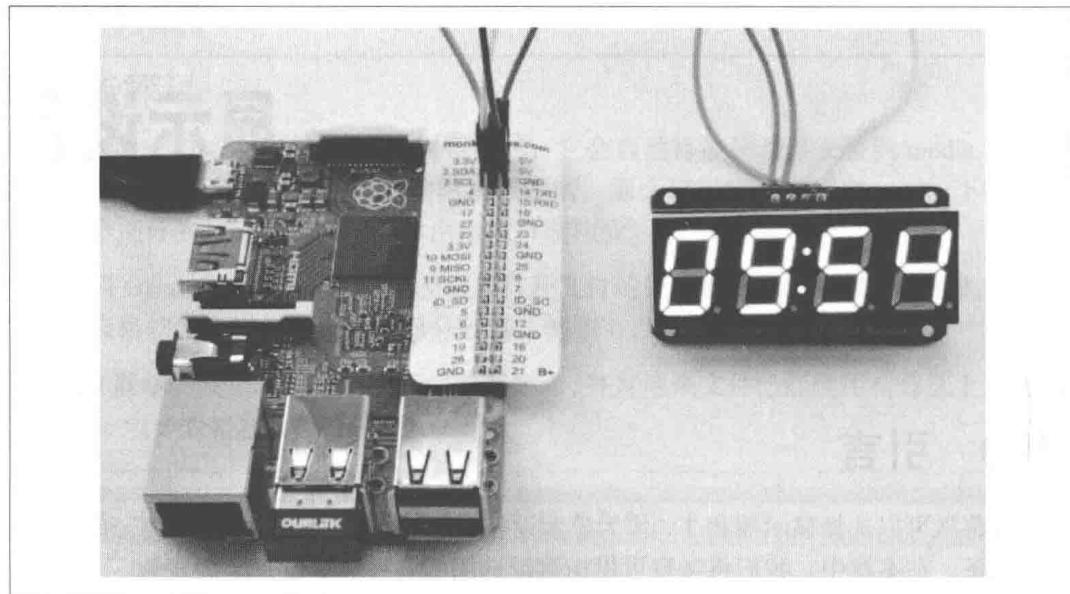


图 14-1 连接了七段 LED 显示设备的树莓派

- 该显示设备上面的 GND (-) 连接到树莓派 GPIO 接口的 GND 引脚。
- 该显示设备上面的 SDA (D) 连接到树莓派 GPIO 接口的 GPIO 2 (SDA) 上面。
- 该显示设备上面的 SCL (C) 连接到树莓派 GPIO 接口的 GPIO 3 (SCL) 上面。

注意，Adafruit 还提供一个超大尺寸的 LED 显示屏。你可以使用上面的方式将其连接到树莓派上面，但是这个大尺寸的显示屏需要两个正极电源引脚：一个用于逻辑 (`V_IO`)，一个用于显示 (`5V`)。为此，你只需另外使用一根 Female-to-Female (母头转母头) 跳线将额外的引脚连接至 GPIO 接口的第二个 `5V` 引脚即可。这是因为由于该显示设备尺寸较大，所以需要更多的电流供发光二极管显示器使用。幸运的是树莓派可以提供足够的电流供它使用。

为了让本节中的示例正常运行，你需要首先按照 9.3 节中的介绍设置树莓派的 I2C。

Adafruit 为该显示设备提供了一个相应的 Python 库。但是，它不是作为一个严格意义上的库来安装的，所以要想使用该库的话，需要首先下载相应的文件夹结构。

```
$ git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

通过下列命令切换到 Adafruit 代码所在目录。

```
$ cd Adafruit-Raspberry-Pi-Python-Code  
$ cd Adafruit_LEDBackpack
```

在这个文件夹中，你可以找到一个显示时间的示例程序。为了运行这个程序，可以使

用下列代码。

```
$ sudo python ex_7segment_clock.py
```

进一步探讨

如果你利用 nano 打开这个示例文件 segment_clock.py 的话，将会看到如下所示的关键命令。

```
from Adafruit_7Segment import SevenSegment
```

上面命令的作用是将库代码导入到你的程序中，然后，你需要利用下面的一行代码来创建一个 SevenSegment 的实例。作为参数提供的地址实际上是 I2C 的地址(见 9.4 节)。

每个 I2C 从属设备都有一个地址数字。这个 LED 电路板的背面有 3 对焊盘，所以如果想改变地址的话，可以通过焊接方式将它们桥接起来。如果你想通过一个树莓派操作多个 I2C 设备的话，这是非常必要的。

```
segment = SevenSegment(address=0x70)
```

要想实际设置某个数字的内容，可以使用如下所示的命令。

```
segment.writeDigit(0, int(hour / 10))
```

第一个参数 (0) 是数字的位置，需要注意的是这些位置分别是 0、1、3 和 4。

位置 2 是预留给该显示设备中间的两个点的。

第二个参数是需要显示的数字本身。

参考资料

你可以从 <http://bit.ly/HQBE6W> 站点找到 Adafruit 库的详细介绍。

14.2 在 I2C LED 矩阵上面显示消息

面临问题

你希望控制彩色 LED 矩形显示设备上的像素。

解决方案

你可以使用一个 I2C LED 模块，如图 14-2 中展示的那样，使用 Female-to-Female (母头转母头) 跳线连接到树莓派上面。

为了进行本节中的试验，你需要：

- 4 个 Female-to-Female (母头转母头) 跳线；

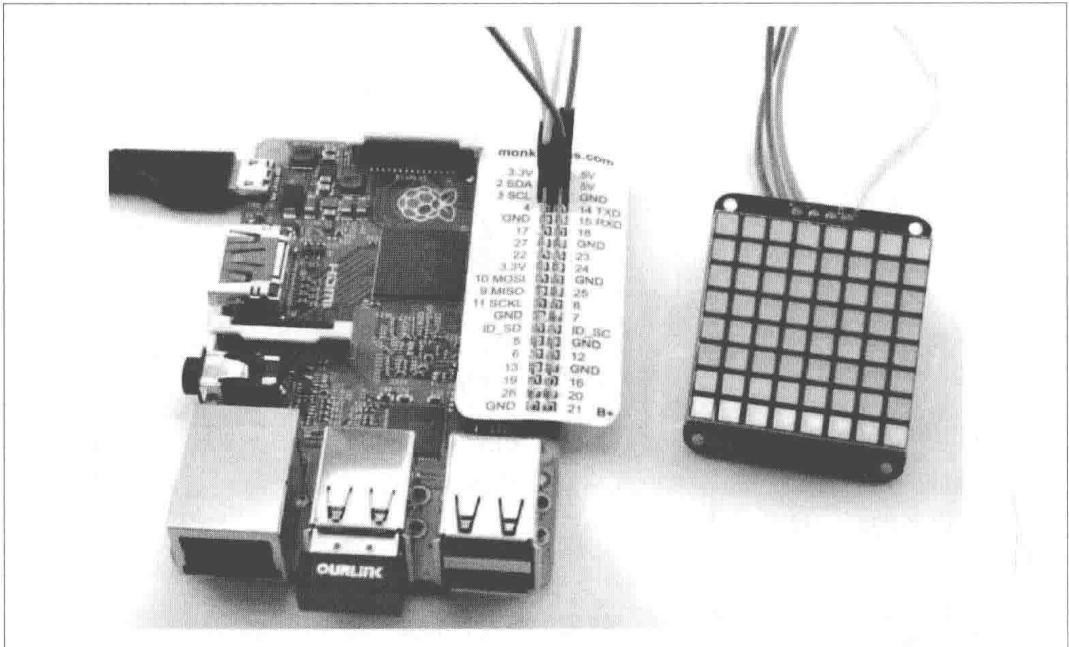


图 14-2 连接树莓派的 LED 矩阵显示设备

- Adafruit 背负 I2C 的双色 LED 方形像素矩阵显示设备。

树莓派与该模块之间的连接如下所示。

- 该显示设备上面的 VCC (+) 连接到树莓派 GPIO 接口上面的 5V 引脚。
- 该显示设备上面的 GND (-) 连接到树莓派 GPIO 接口的 GND 引脚。
- 该显示设备上面的 SDA (D) 连接到树莓派 GPIO 接口的 GPIO 2 (SDA) 上面。
- 该显示设备上面的 SCL (C) 连接到树莓派 GPIO 接口的 GPIO 3 (SCL) 上面。

为了让本节中的示例正常运行，你需要首先按照 9.3 节中的介绍设置树莓派的 I2C。

Adafruit 为该显示设备提供了一个相应的 Python 库。但是，它不是作为一个严格意义上的库来安装的，所以要想使用该库的话，需要首先下载相应的文件夹结构。

```
$ git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

通过下列命令切换到 Adafruit 代码所在目录。

```
$ cd Adafruit-Raspberry-Pi-Python-Code  
$ cd Adafruit_LEDBackpack
```

在这个文件夹下面，你可以找到一个使用滚动数字显示时间的测试程序。

要想运行该程序，可以使用如下所示的命令。

```
$ sudo python ex_8x8_color_pixels.py
```

进一步探讨

这个程序会按照顺序循环操作每个像素的所有颜色。下面给出的代码，已经删除了某些注释和不必要的导入代码。

```
import time
from Adafruit_8x8 import ColorEightByEight

grid = ColorEightByEight(address=0x70)

iter = 0

# Continually update the 8x8 display one pixel at a time
while(True):
    iter += 1

    for x in range(0, 8):
        for y in range(0, 8):
            grid.setPixel(x, y, iter % 4)
            time.sleep(0.02)
```

在下面这行代码中，I2C 地址（见 9.4 节）是以参数的形式来提供的。

```
grid = ColorEightByEight(address=0x70)
```

每个 I2C 从属设备都有一个地址数字。这个 LED 电路板的背面有 3 对焊盘，所以如果想改变地址的话，可以通过焊接方式将它们桥接起来。如果你希望通过单个树莓派来操作多个具有相同基地址的 I2C 设备的话，这是非常有必要的。

每循环一次，变量 `iter` 的值就会加 1。命令 `grid.setPixel` 以 `x` 和 `y` 坐标作为其前两个参数。最后一个参数是给像素设置的颜色。该参数是一个介于 0 到 3 之间的数字（0 表示关闭，1 表示绿色，2 表示红色，而 3 则表示橘色）。

变量 `iter` 用于生成一个介于 0 到 3 之间的数字，这个数字是通过%运算符得到的，该运算符就是所谓的取模运算符，就本例来说就是除以 4 之后得到的余数。

参考资料

关于该产品的更多介绍，请参考 <http://www.adafruit.com/products/902>。

14.3 使用 Sense HAT LED 矩形显示器

面临问题

你想通过 Sense HAT 的显示器来展示消息和图像。

解决方案

首先，请按照 9.16 节介绍的方法来安装 Sense HAT 所需的软件，然后就可以使用相应

的库命令来显示文本了。

程序 `sense_hat_clock.py` 展示的是以滚动消息的形式重复显示日期与时间。正如本书中的所有示例程序一样，你可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码下载区下载它们。

```
from sense_hat import SenseHat
from datetime import datetime
import time

hat = SenseHat()
time_color = (0, 255, 0)
date_color = (255, 0, 0)

while True:
    now = datetime.now()
    date_message = '{:%d %B %Y}'.format(now)
    time_message = '{:%H:%M:%S}'.format(now)

    hat.show_message(date_message, text_colour=date_color)
    hat.show_message(time_message, text_colour=time_color)
```

进一步探讨

由于上面定义了两种颜色，所以消息的日期与时间部分将以不同的颜色来显示。然后，这些颜色将用作 `show_message` 的可选参数。

`Show_message` 的其他可选参数包括以下几个。

- `scroll_speed` 每个滚动步骤之间的时间延迟，并非滚动速度。所以这个值越大，滚动就会越慢。
- `back_colour` 设置背景色。需要注意的是这里的“`back_colour`”使用的是英式拼写，即带有字母“u”。该显示器的用途非常广泛，而非仅限于显示滚动的文本。就最简单的来说，你可以使用 `set_pixel` 设置特定的像素，利用 `set_rotation` 设置显示的方向，同时还可以通过 `load_image` 显示图像（虽然有点小）。下面的示例代码取自 `sense_hat_taster.py`，它很好地展示了这些函数的用法。就像本书中的所有示例程序一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码下载区来下载该程序。

显示的图像的大小只能是 8×8 像素的，并且可以采用诸如.jpg 之类的常用图像格式，图像的位深度将会自动处理。

```
from sense_hat import SenseHat
import time

hat = SenseHat()

red = (255, 0, 0)
```

```
hat.load_image('small_image.png')
time.sleep(1)
hat.set_rotation(90)
time.sleep(1)
hat.set_rotation(180)
time.sleep(1)
hat.set_rotation(270)
time.sleep(1)

hat.clear()
hat.set_rotation(0)
for xy in range(0, 8):
    hat.set_pixel(xy, xy, red)
    hat.set_pixel(xy, 7-xy, red)
```

图 14-3 展示了正在显示一幅粗糙图像的 Sense HAT。

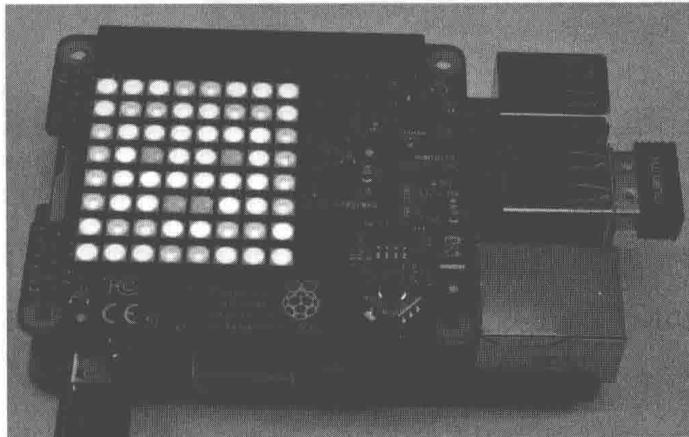


图 14-3 正在显示“图像”的 Sense HAT

参考资料

关于 Sense HAT 的完整文档，请参考 <https://pythonhosted.org/sense-hat/api/>。

关于格式化时间和日期的详细介绍，请参考 7.2 节。

其他用到 Sense HAT 的示例，请参考 9.16 节、13.10 节、13.13 节、13.14 节和 13.16 节。

14.4 在 Alphanumeric LCD HAT 上显示消息

面临问题

你想将几行文本优雅地显示到一个 LCD 显示设备上面。

解决方案

你可以像图 14-4 所示那样，在树莓派上面连接一个 Displayotron Pimoroni HAT。



图 14-4 Displayotron LCD HAT

这个 HAT 要求同时启用 I2C 和 SPI，如果尚未启用的话，具体请参考 9.3 节和 9.5 节。

然后，从 GitHub 下载这个 HAT 对应的库代码，并通过下列代码进行安装。

```
$ git clone https://github.com/pimoroni/dot3k.git  
$ cd dot3k/python/library  
$ sudo python setup.py install
```

举例来说，下面的程序（`displayotron_ip.py`）可以查找树莓派的主机名和 IP 地址，并连同时间一起显示出来。如果一切正常的话，那么这个 LED 的背光就会变绿，但是如果网络连接有问题的话，背光就会变成红色。

就像本书中的所有示例程序一样，你也可以从本书网站（<http://www.raspberrypiCookbook.com/>）的代码下载区下载该程序。

```
import dothat.lcd as lcd  
import dothat.backlight as backlight  
import time  
from datetime import datetime  
import subprocess  
  
while True:  
    lcd.clear()  
    backlight.rgb(0, 255, 0)  
    try:
```

```
hostname = subprocess.check_output(['hostname']).split()[0]
ip = subprocess.check_output(['hostname', '-I']).split()[0]
t = '{:%H:%M:%S}'.format(datetime.now())
lcd.write(hostname)
lcd.set_cursor_position(0, 1)
lcd.write(ip)
lcd.set_cursor_position(0, 2)
lcd.write(t)
except:
    backlight.rgb(255, 0, 0)
time.sleep(1)
```

进一步探讨

这个测试程序将会导入所需的程序库，包括 subprocess 库（见 7.15 节），这些库被用来寻找树莓派的 IP 地址（见 2.2 节）及其主机名。

在这个库中，主要的方法包括如下几个。

- lcd.clear 清除显示的所有文本。
- lcd.set_cursor_position 设置新文本的写入位置，该方法以行和列作为参数指定具体位置。
- lcd.write 将作为参数提供给它的文本显示到当前光标位置。
- backlight.rgb 设置背光（0 到 255）的红、绿和蓝值。

参考资料

关于该 HAT 的更多信息，请参考 Pimoroni 相应的产品页面 (<https://shop.pimoroni.com/products/display-o-tron-hat>)。

如果你想直接给树莓派连接一个廉价的 LCD 模块，而非使用 HAT 的话，请参考 14.5 节。

14.5 在 Alphanumeric LCD 模块上显示消息

面临问题

你希望使用字母数字式的 LCD 显示设备来显示文字，而不是使用现成的显示器 HAT。

解决方案

你可以使用一个 HD44780-compatible LCD 模块，并将其连接到 GPIO 接口上面。

很明显，这比使用封装好的显示设备要复杂得多，但是，如果正在规划的项目非常复杂，并且使用一个元件作为其显示设备，同时还需要访问其他 GPIO 引脚的话，那么了

解直接使用 LCD 模块的方法将是非常有用的。

你可以找到许多廉价的 LCD 模块，例如图 14-5 中连接到树莓派上面的模块。这些模块都带有 16 个引脚，不过好在并非所有的引脚都需要连接到 GPIO 引脚上面。

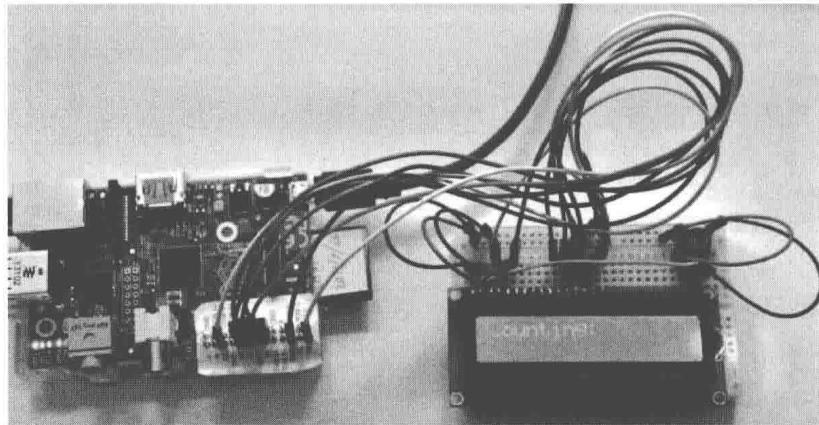


图 14-5 连接到树莓派上面的一个 16×2 LCD 显示设备

这类模块的尺寸各异，并且每个尺寸都规定了可以显示的行数，以及每行可以显示的字符数。所以，举例来说，本节使用的模块的大小是 16×2 ，因为它可以显示两行文本，每行可以显示 16 个字符。

其他常见的尺寸有 8×1 、 16×1 、 16×2 、 20×2 和 20×4 。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 16×2 HD44780-compatible LCD 模块；
- 单行 16 接头引脚；
- $10k\Omega$ 调谐电位器。

图 14-6 展示了连接该显示器的面包板布局。该 LCD 模块通常不提供插头引脚，所以你需要将这些引脚焊接到恰当的位置。

调谐电位器可以用来控制显示设备的对比度。需要注意的是当项目无法正常工作的时候，可以尝试将调谐电位器的旋钮在整个可移动范围内移动。这可能是由于对比度设置为关闭，所以导致显示的字符根本无法看到。

Adafruit 提供的树莓派示例代码可以从 GitHub (<http://bit.ly/1gHetIt>) 上面下载，其中包括一个通过 HD44780 驱动程序来控制 LCD 显示设备的程序库。

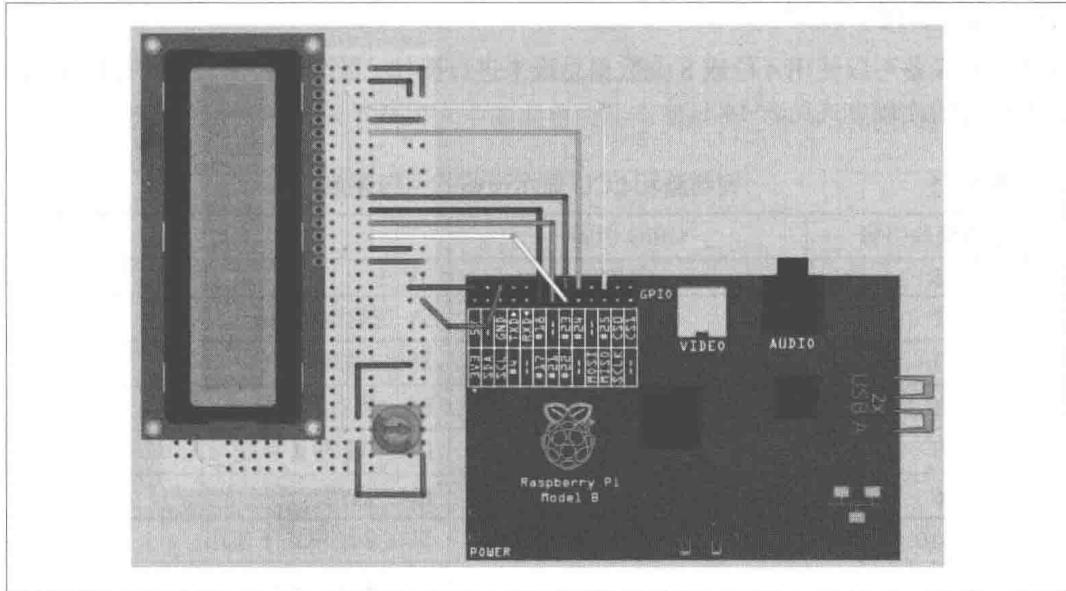


图 14-6 将 LCD 显示设备连接至树莓派

在安装该示例程序之前，需要按照 9.2 节中的介绍先把 RPi.GPIO 库安装好。

由于这个 Adafruit 库不是作为一个严格意义上的库来安装的，所以要想使用该库的话，需要首先下载相应的文件夹结构。

```
$ git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

然后，将目录切换至存放 Adafruit 代码的目录下面，具体命令如下所示。

```
$ cd Adafruit-Raspberry-Pi-Python-Code  
$ cd Adafruit_CharLCD
```

这里有一个测试程序，专门用来显示树莓派的时间和 IP 地址。

但是，如果你使用的是较新的第二版 B 型树莓派的话，那么你首先需要编辑文件 Adafruit_CharLCD.py。

```
$ nano Adafruit_CharLCD.py
```

找到下面这行。

```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 21, 22], GPIO = None):
```

将数字 21 替换为 27，使它看起来像下面这行一样，然后保存文件并退出。

```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 27, 22], GPIO = None):
```

现在，你就可以通过下面的命令来运行这个示例程序了。

```
$ sudo python Adafruit_CharLCD_IPclock_example.py
```

进一步探讨

这些显示设备可以使用 4 位或 8 位数据总线来进行操作，此外，还需要 3 个控制引脚。这些引脚的连接方式见表 14-1。

表 14-1 树莓派和 LCD 显示设备的引脚连接

LCD 模块引脚	GPIO 引脚	备注
1	GND	0V
2	+5V	5V 逻辑电源电压
3	无连接	对比度控制电压
4	25	RS：寄存器选择
5	GND	RW：读/写（总是写入）
6	24	EN：启用
7-10	无连接	只在 8 位模式下使用
11	23	D4：4 号数据线
12	17	D5：5 号数据线
13	21	D6：6 号数据线
14	22	D7：7 号数据线
15	+5V	LED 背光
16	GND	LED 背光

Adafruit 库文件 Adafruit_CharLCD.py 的用途是设置数据引脚的值，并将其发送到显示模块。它提供了如下所示的函数，供你在编程的时候引用。

home()

移至左上角。

clear()

将显示的文本全部清除。

setCursor(column, row)

设置文本书写位置的光标位置。

cursor()

打开光标显示。

noCursor()

关闭光标显示（默认）。

message(text)

在当前光标位置写入文本。

下面这个迷你示例程序展示了利用这个库来显示一个简单消息是多么的容易。

```
from Adafruit_CharLCD import Adafruit_CharLCD
from time import sleep

lcd = Adafruit_CharLCD()
lcd.begin(16,2)

i = 0

while True:
    lcd.clear()
    lcd.message('Counting: ' + str(i))
    sleep(1)
    i = i + 1
```

参考资料

本节内容借鉴了 Adafruit 学习网站 (<http://bit.ly/1kf4xWC>) 上面的一篇 Adafruit 教程。此外，Adafruit 还销售兼容本节内容的一款插件式电路板，该电路板上面附带了一个 LCD 显示设备。为便利起见，可以使用 HAT 上面现成的字母数字式 LCD 显示设备，具体请参考 2.2 节。

14.6 使用 OLED 图形显示器

面临问题

你想给树莓派连接一个图形 OLED（有机 LED）显示器。

解决方案

你可以使用基于 SSD1306 驱动芯片的 OLED 显示器，这需要用到 I2C 接口（见图 14-7）。

为了进行本节中的试验，你需要：

- 4 个 Female-to-Female（母头转母头）跳线；
- I2C OLED 显示器， 128×64 像素。

这些显示器中，有的只有 4 个引脚，其他的则有 8 个引脚。这些 8 个引脚的显示器（包括 Adafruit 的显示器）的引脚之所以多，是因为它们同时提供了 I2C 和 SPI 接口。而 4 引脚的显示器则只有 I2C 接口。

树莓派与该模块之间的连接如下所示。

- 该显示设备上面的 VCC 连接到树莓派 GPIO 接口上面的 5V 引脚。

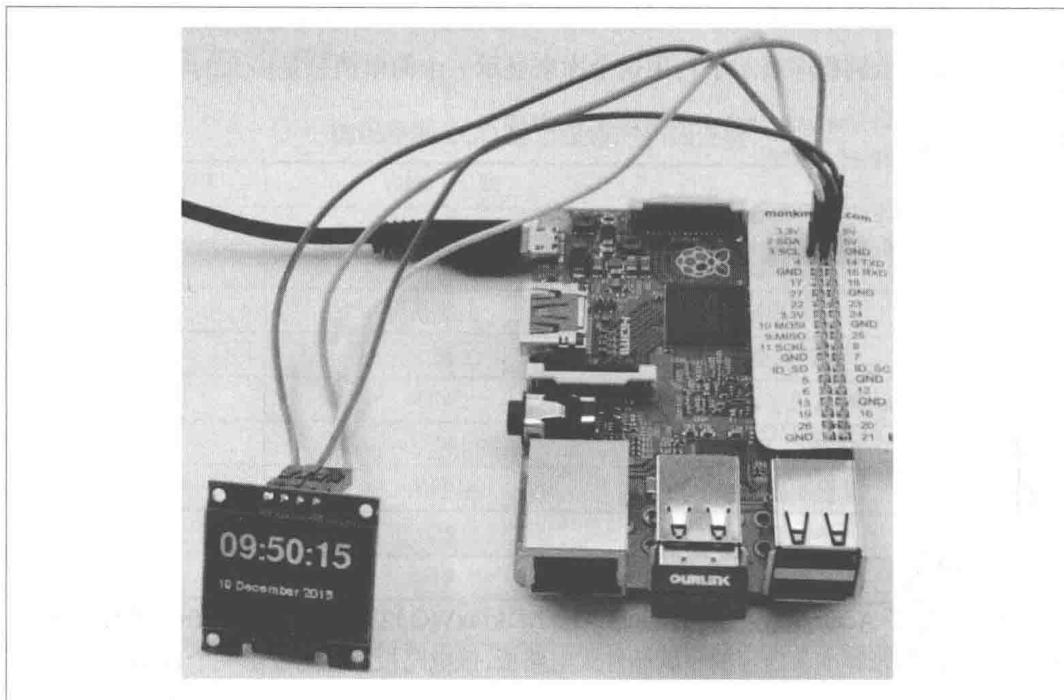


图 14-7 I2C OLED 显示器

- 该显示设备上面的 GND 连接到树莓派 GPIO 接口的 GND 引脚。
- 该显示设备上面的 SDA 连接到树莓派 GPIO 接口的 GPIO 2 (SDA) 上面。
- 该显示设备上面的 SCL 连接到树莓派 GPIO 接口的 GPIO 3 (SCL) 上面。

为了让本节中的示例正常运行，你需要首先按照 9.3 节中的介绍设置树莓派的 I2C。

Adafruit 为这些显示器提供了相应的库，具体安装方法如下所示。

```
$ git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git
$ cd Adafruit_Python_SSD1306
$ sudo python setup.py install
```

这个库需要使用 Python 图像库 (PIL)，其安装命令如下所示。

```
$ sudo pip install pillow
```

示例程序 `old_clock.py` 可以在 OLED 显示器上面显示时间与时期。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，并从代码下载区下载该程序。

```
from oled.device import ssd1306
from oled.render import canvas
```

```
from PIL import ImageFont
import time
from datetime import datetime

# Set up display
device = ssd1306(port=1, address=0x3C)
font_file = '/usr/share/fonts/truetype/freefont/FreeSansBold.ttf'
small_font = ImageFont.truetype('FreeSans.ttf', 12, filename=font_file)
large_font = ImageFont.truetype('FreeSans.ttf', 33, filename=font_file)

# Display a message on 3 lines, first line big font
def display_message(top_line, line_2):
    global device
    with canvas(device) as draw:
        draw.text((0, 0), top_line, font=large_font, fill=255)
        draw.text((0, 50), line_2, font=small_font, fill=255)

while True:
    now = datetime.now()
    date_message = '{:%d %B %Y}'.format(now)
    time_message = '{:%H:%M:%S}'.format(now)
    display_message(time_message, date_message)
    time.sleep(0.1)
```

每一个 I2C 从属设备都有一个地址，该地址可以通过下面代码进行设置。

```
device = ssd1306(port=1, address=0x3C)
```

对于许多廉价的 I2C 模块来说，它们的地址通常都是固定的 3C（十六进制数），但是有时也会有所不同，所以你应该仔细查看设备附带的文档，或者利用 I2C 工具（见 9.4 节）列出连接到总线上面的所有 I2C 设备，这样就能够从显示设备上面看到它们的地址了。

进一步探讨

小型的有机 LED (OLED) 显示设备不但便宜、省电，并且虽然外形小巧，但是却具有非常高的分辨率。在许多消费产品中，它们正有取代 LCD 显示器的趋势。

参考资料

本节内容适用于 4 引脚的 I2C 接口。如果你想使用 SPI 接口的话，请参考 Adafruit 网站上的一篇相关教程，地址为 <https://learn.adafruit.com/ssd1306-oled-displays-with-raspberry-pi-and-beaglebone-black>。

14.7 使用可寻址的 RGB LED 灯条

面临问题

你想给树莓派连接一个 RGB LED 灯条 (Neopixels)。

解决方案

你可以在树莓派上面使用基于 WS2812 RGB LED 芯片的 LED 灯条。

这些 LED 灯条的用法其实非常简单，你只要直接将其连接到树莓派，并通过树莓派来给这些 LED 供电即可。当然，在正常情况下这种方法能够很好地工作，但是阅读本节的“进一步探讨”部分后你就会发现，这种方法有许多潜在的缺陷，不过别担心，我们同时提供了相应的解决方案，能够保证你无忧地使用 LED 灯条。



不要使用树莓派的 3.3V 引脚给 LED 供电

虽然利用 GPIO 接口的 3.3V 电源引脚为 LED 供电看上去非常有诱惑力，但是请不要这样做——它只能提供弱电流（见 9.2 节）。如果使用这个引脚供电的话，很容易损坏树莓派。

在图 14-8 中使用的 LED 灯条是从灯条盘卷中剪下来的。就本例来说，该灯条只有 10 个 LED。

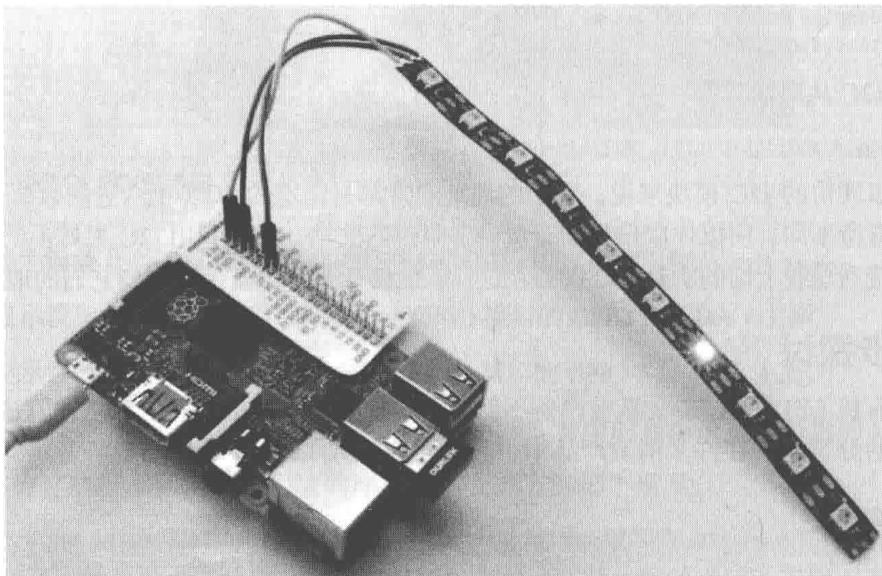


图 14-8 具有 10 个 LED 的灯条

因为每个 LED 都会消耗 60mA 的电流，所以，如果没有为 LED 灯条提供单独的电源的话，10 就是 LED 数目的一个比较合理的上限（请参考“进一步探讨”部分）。

为了将该灯条连接到树莓派上面，可以将带有塞孔的跳线的一端剪下来，并将导线端焊接到该 LED 灯条的 3 个连接上面：GND、DI（数据输入）和 5V 连接。然后，将这些跳线分别连接到 GPIO 接口的 GND、GPIO18 和 5V 引脚上面即可。

需要注意的是该 LED 灯条上面印有一行箭头（见图 14-9）。当你给这个 LED 灯条焊接引线的时候，请确保从左侧箭头的切头处开始。

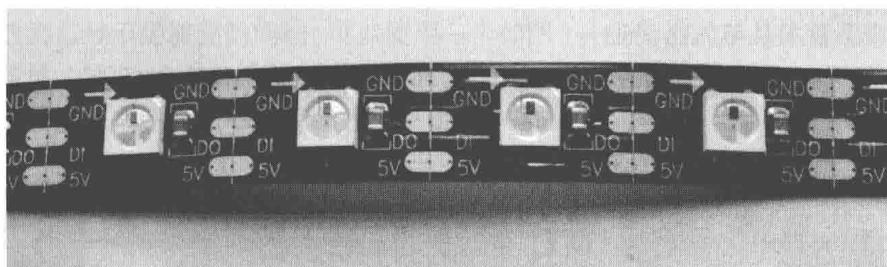


图 14-9 LED 灯条的特写

截止到写作本书时，寻找适用于所有树莓派型号的 WS2812 代码库还不是一件容易的事情。最初的代码库是由 Jeremy Garff 开发的 (https://github.com/jgarf/rpi_ws281x)，后来 Richard Hurst 提供了支持树莓派 2 的代码库 (https://github.com/richardghirst/rpi_ws281x)。Adafruit 也参与了该库的开发。但是，Raspbian 切换到 Linux 的 4.1.6 内核后，有些事情就停了下来。截止写作本书时，Pimoroni Unicorn HAT（它使用了 Richard Hurst 的代码库的一个改进版）是最为可靠的一个版本。由于跟商业产品有关，所以每当 Raspbian 和树莓派电路板发生改进时，这个版本最有可能得到及时的维护。为了安装这个库，可以使用如下所示的代码。

```
$ git clone https://github.com/pimoroni/unicorn-hat.git  
$ cd unicorn-hat/python/rpi-ws281x  
$ make  
$ sudo python setup.py install
```

下面的示例程序 (`led_strip.py`) 将会使 LED 沿着灯条的方向依次变红。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，并从代码下载区下载该程序。

```
import time  
from neopixel import *  
  
# LED strip configuration:  
LED_COUNT      = 10      # Number of LED pixels.  
LED_PIN        = 18      # GPIO pin connected to the pixels (must support PWM!).  
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)  
LED_DMA        = 5       # DMA channel to use for generating signal (try 5)  
LED_BRIGHTNESS = 255    # Set to 0 for darkest and 255 for brightest  
LED_INVERT     = False   # True to invert the signal  
  
RED = Color(255, 0, 0)  
NO_COLOR = Color(0, 0, 0)
```

```

strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS)
strip.begin()

def clear():
    for i in range(0, LED_COUNT):
        strip.setPixelColor(i, NO_COLOR)
    strip.show()

i = 0
while True:
    clear()
    strip.setPixelColor(i, RED)
    strip.show()
    time.sleep(1)
    i += 1
    if i >= LED_COUNT:
        i = 0

```

在 LED strip configuration 部分中的各个参数，可以根据不同的 LED 配置进行相应的修改。如果你的灯条具有不同数量的 LED，那么可以修改 LED_COUNT。而其余的常数则无需改变。

灯条上面的每个 LED，都可以通过 setPixelColor 方法来为其设置单独的颜色。这个方法的第一个参数，是需要设置颜色的 LED 的下标位置（从 0 开始）。第二个参数是要设置的颜色。实际上，直到这个 show 方法被调用之后，LED 的颜色才会发生改变。

进一步探讨

灯条上面的每个 LED 的最大电流为 60mA 左右，但是，只有当所有 3 个颜色通道（红、绿和蓝）都达到最大亮度（255）时，才会消耗这么多的电流。因此，如果你计划使用大量 LED 的话，那么需要使用一个独立的 5V 电源，才能给灯条上面的 LED 提供足够的电流。图 14-10 展示了连接独立电源的方法。如果使用一个母头 DC 插孔到螺丝端子的适配器的话，可以使得外部电源到面包板的连接工作更加轻松。

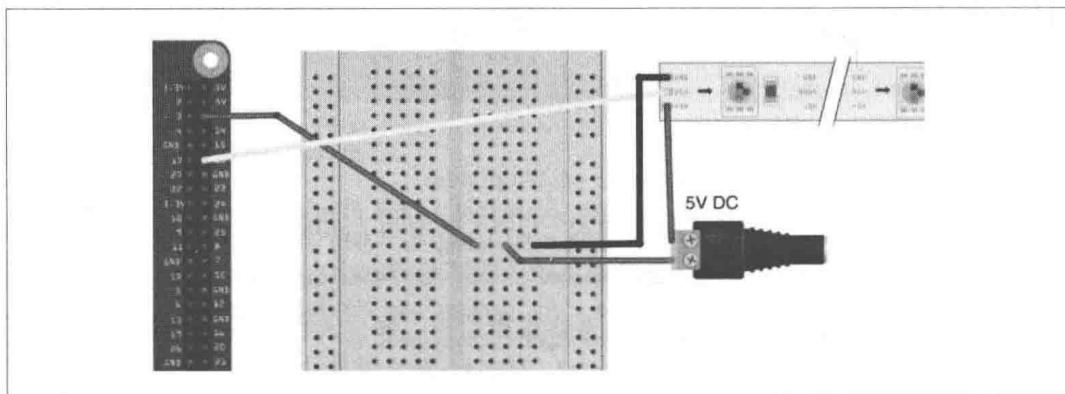


图 14-10 利用外部电源为 LED 灯条供电

根据 WS2812 的参数手册, 用于 LED 的 5V 电源的输入信号的逻辑高电平至少要有 4V。因此, 当使用树莓派的 3.3v GPIO 输出给它供电的时候, 很难保证 LED 能够可靠工作。话虽如此, 我却从未遇到过任何麻烦。

如果你遇到信号不够强的问题, 可以使用一个如图 14-11 所示的单晶体管逻辑电平转换器。这里, 2N7000 是一个比较合适的晶体管 (注意, 在这种情况下 2N3904 无法正常工作, 因为它不能迅速地响应反转脉冲)。

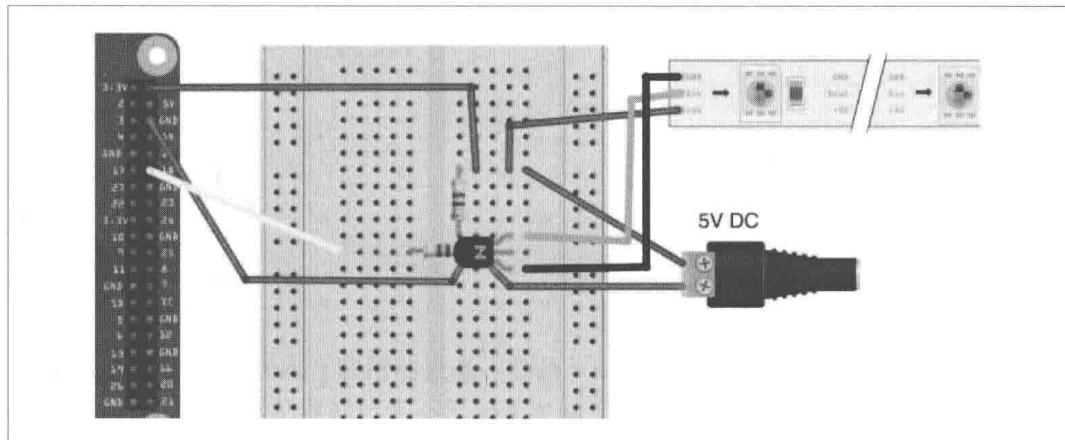


图 14-11 使用逻辑电平变换器

虽然这个晶体管能够将 3.3V 的信号转换为 5V, 但是它有一个副作用, 就是会反转信号, 即如果 GPIO 引脚的输出是 HIGH 的话, 进入 LED 灯条的信号将会变为 LOW, 反之亦然。这就意味着你需要改变下面这行代码, 来修改程序的逻辑。

```
LED_INVERT = False
```

改为:

```
LED_INVERT = True
```

参考资料

除了可以使用晶体管将信号电平从 3.3V 提高到 5V 之外, 你还可以使用电平转换模块 (见 9.13 节)。

Unicorn HAT (本节使用的就是它的程序库) 具有 64 个 WS2812 LED, 它们以 8×8 的网格形式排列。关于这个显示设备的更多介绍, 请访问 <https://shop.pimoroni.com/products/unicorn-hat>。

15.0 引言

物联网（IoT）是由接入互联网的设备（物品）组成的网络，当前呈迅猛发展之势。换句话说，目前使用浏览器的已经不仅限于电脑了，因为已经有越来越多的物理家电、可穿戴和移动设备也加入到这个队伍中来了。使用浏览器的大军覆盖了所有种类的家庭自动化技术：智能家电和照明、安防系统、通过互联网操作的宠物喂给装置，以及各种实用且有趣的项目等。

在本章中，我们将介绍树莓派在物联网中的某些具体应用。

15.1 使用 Web 接口控制 GPIO 输出

面临问题

你想利用树莓派的 Web 接口来控制 GPIO 输出。

解决方案

你可以使用 Python 的 Web 服务器库 bottle（见 7.17 节）创建一个 HTML Web 接口，用来控制 GPIO 端口。

为了进行本节中的实验，你需要：

- 面包板和跳线；
- 3 个 $1\text{k}\Omega$ 电阻器；
- 3 个 LED；

- 轻触按钮开关。

该示例所用的面包板如图 15-1 所示。

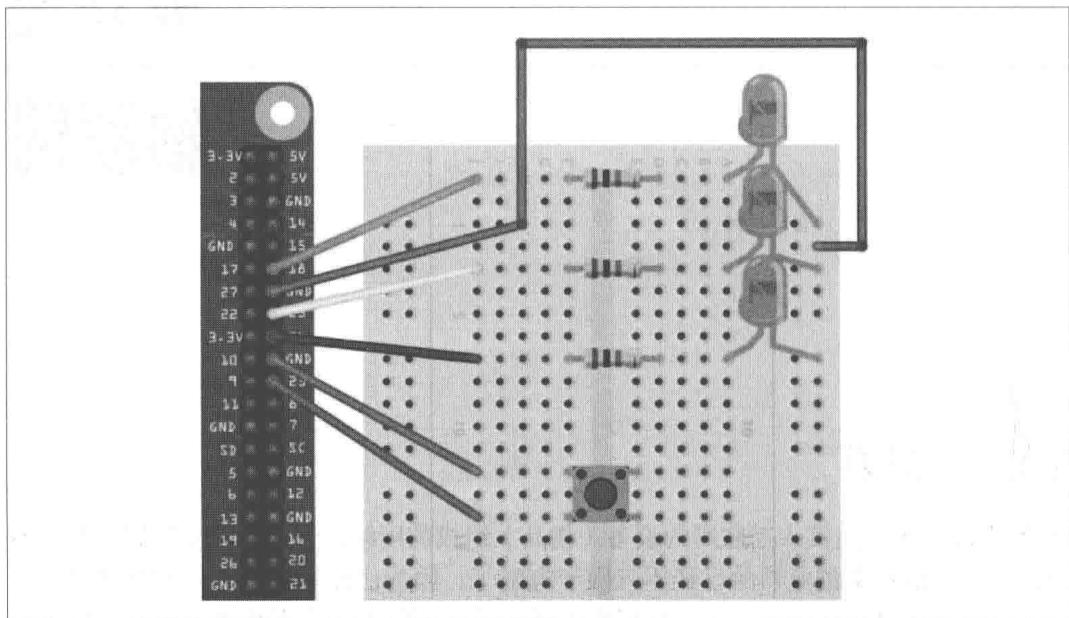


图 15-1 利用网页控制 GPIO 输出时所用的面包板布局

如果你不想使用面包板的话，可以选择 Raspberry Squid 和 Squid Button（具体参考 9.10 节和 9.11 节）。这两个替代品可以直接插到树莓派的 GPIO 引脚上面，具体如图 15-2 所示。

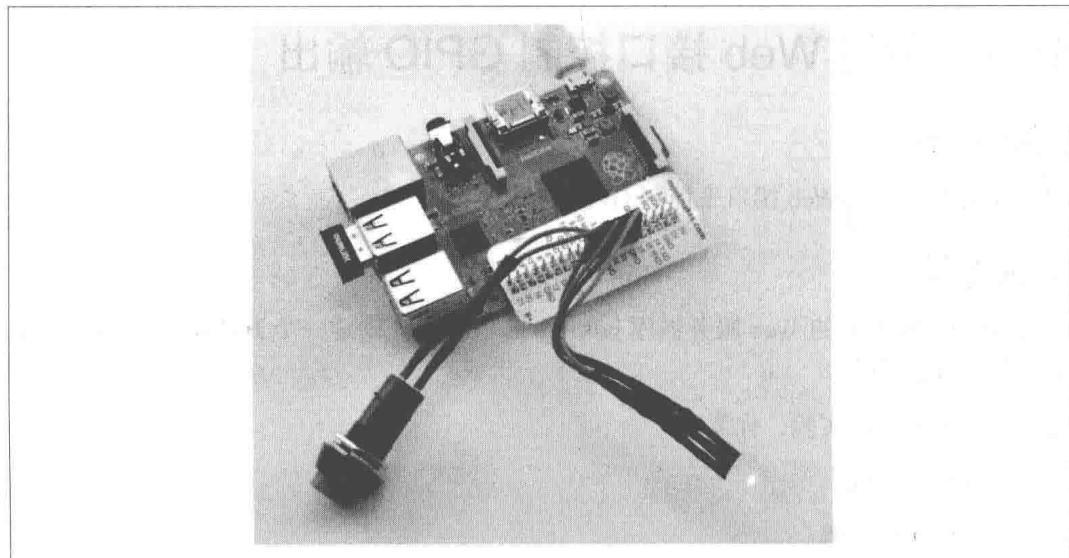


图 15-2 Raspberry Squid 和 Squid Button

关于 bottle 库安装方法的详细介绍，请参考 7.17 节。

打开一个编辑器（ nano 或者 IDLE ），并粘贴如下所示的代码。就像本书中的其他示例代码一样，你也可以从本书网站（ <http://www.raspberrypicookbook.com> ）的代码区下载相应的程序文件，就本例来说，该程序名为 web_control.py 。此外，你可以找到一个名为 web_control_test.py 的程序。这个程序只是对硬件进行简单测试，即让 LED 依次闪光，并显示开关的状态，但是，它没有用到任何 Web 服务器。

下面代码来自 web_control.py 程序。

```
from bottle import route, run
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
led_pins = [18, 23, 24]
led_states = [0, 0, 0]
switch_pin = 25

GPIO.setup(led_pins[0], GPIO.OUT)
GPIO.setup(led_pins[1], GPIO.OUT)
GPIO.setup(led_pins[2], GPIO.OUT)
GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def switch_status():
    state = GPIO.input(switch_pin)
    if state:
        return 'Up'
    else:
        return 'Down'

def html_for_led(led):
    l = str(led)
    result = " <input type='button' onClick='changed(" + l +
")' value='LED " + l + "'/>"
    return result

def update_leds():
    for i, value in enumerate(led_states):
        GPIO.output(led_pins[i], value)

@route('/')
@route('/<led>')
def index(led):
    if led >= '0' and led <= '9':
        led_num = int(led)
        led_states[led_num] = not led_states[led_num]
        update_leds()
        response = "<script>"
        response += "function changed(led)"
        response += "{"
        response += "    window.location.href='/"
        response += led
        response += "'}"
        return response
    else:
        return "Error: LED number must be between 0 and 9."
```

```
response += "</script>"  
  
response += '<h1>GPIO Control</h1>'  
response += '<h2>Button=' + switch_status() + '</h2>'  
response += '<h2>LEDs</h2>'  
response += html_for_led(0)  
response += html_for_led(1)  
response += html_for_led(2)  
return response  
  
try:  
    run(host='0.0.0.0', port=80)  
finally:  
    print('\nCleaning up')  
    GPIO.cleanup()
```

你必须以超级用户身份来运行该程序。

```
sudo python web_control.py
```

如果该程序启动无误的话，将会显示如下所示内容。

```
Bottle server starting up (using WSGIRefServer())...  
Listening on http://0.0.0.0:80/  
Hit Ctrl-C to quit.
```

现在，请你从网络上的任意一台机器，甚至从树莓派本身上面打开浏览器窗口，并输入树莓派的 IP 地址。这时，将看到如图 15-3 所示的 Web 接口。

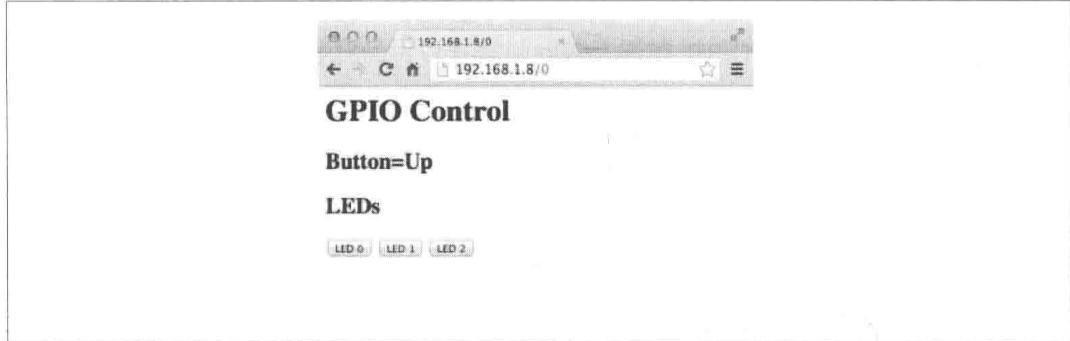


图 15-3 用于 GPIO 的 Web 接口

在屏幕下方有 3 个 LED 按钮，如果你单击其中一个的话，就会发现这样能够切换对应 LED 的开关状态。

此外，如果你按下按钮，那么当你重新加载该网页的时候，就会发现 Button 旁边的文字将变成 Down，而不是原来的 Up。

进一步探讨

为了理解这个程序的工作机制，我们需要先弄清楚 Web 接口的工作原理。所有 Web 接

口都依赖于某处的服务器（就本例而言，这里的 Web 接口就是树莓派上的一个程序）来响应网络浏览器的请求。

当服务器收到一个请求时，它会分析请求中提供的信息，然后据此利用超级文本标志语言进行响应。

如果 Web 请求是指向根页面 (`http://192.168.1.8/`) 的话，那么 `led` 就取默认值 `n`。但是，如果我们在浏览器中输入 `http://192.168.1.8/2` 的话，那么这个 URL 最后面的数字 `2` 就是给参数 `led` 指定的值。

然后，根据参数 `led` 的值就能判断出需要切换开关状态的是 LED `2`。

为了能够访问这个实现 LED 开关的 URL，我们需要进行相应的处理，以便在 LED `2` 的按钮被按下时，这个页面会根据 URL 末尾的附加参数进行重载。这里需要用到一个技巧，即在返回给浏览器的 HTML 中添加一个 JavaScript 函数。当浏览器运行这个函数时，会促使该页面利用相应的附加参数来执行重载操作。

这一切意味着我们所面临的情况非常令人费解：先用 Python 程序生成 JavaScript 代码，然后让浏览器来执行生成的 JavaScript 代码。下面展示的是用来生成这个 JavaScript 函数的代码。

```
response = "<script>\nresponse += "function changed(led) {\nresponse += "    window.location.href='/' + led\nresponse += "}\nresponse += "</script>"
```

你不用亲自为每个按钮生成相应的 HTML，这个工作可由 `html_for_led` 函数完成。

```
def html_for_led(led):\n    l = str(led)\n    result = " <input type='button' onClick='changed(" + l + ")'\n    value='LED " + l + "'/>"\n    return result
```

上面的代码会用 3 次，因为每个按钮都要用一次，然后就可以把按钮的按下动作与 `changed` 函数联系起来了。这个函数也需要以参数的形式来提供 LED 的编号。

实际修改 GPIO 输出的代码位于 `update_leds` 函数中。

每当服务器收到一个包含需要切换开关状态的 LED 编号的请求的时候，就会调用这个函数。

```
def update_leds():\n    for i, value in enumerate(led_states):\n        GPIO.output(led_pins[i], value)
```

该函数只是简单遍历一个状态数组，依次将每个输出设置为其当前值。

下面这行代码来自 `index` 函数，它会反转由参数 `led` 指定的 LED 的状态数组中的值。

```
led_states[led_num] = not led_states[led_num]
```

报告按钮状态的过程则要简单得多了。

这只需读取输入的状态，然后生成报告按钮位置的 HTML 即可，这一切都由函数 `switch_status` 来完成。

```
def switch_status():
    state = GPIO.input(switch_pin)
    if state:
        return 'Up'
    else:
        return 'Down'
```

参考资料

有关 `bottle` 用法的详细介绍，请参考相应的文档，地址为 <http://bottlepy.org/docs/dev/>。

15.2 在网页上面显示传感器读数

面临问题

你希望在网页上面显示树莓派的传感器读数，并且希望该页面能够自动更新。

解决方案

你可以使用 `bottle` 网络服务器和一些奇特的 JavaScript 代码来自动更新显示内容。

图 15-4 中展示的例子可以显示树莓派内置传感器检测到的 CPU 温度。

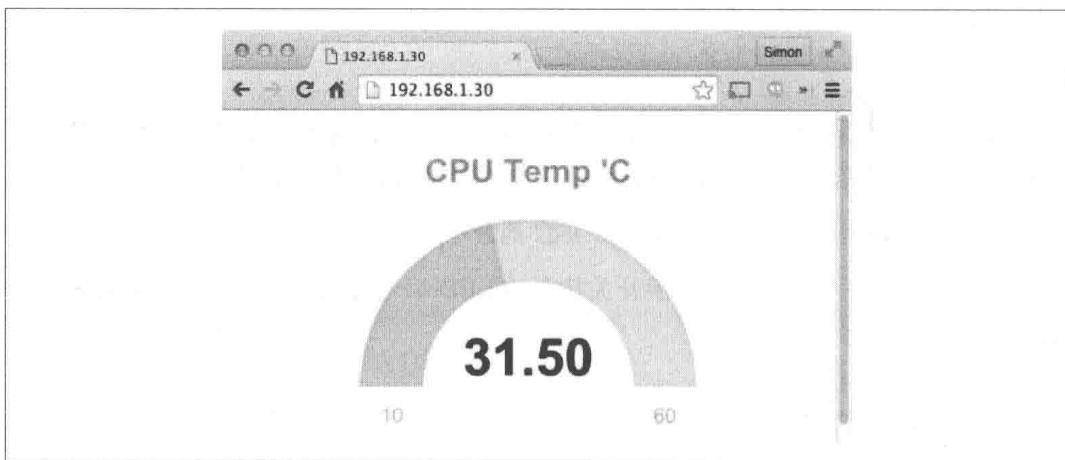


图 15-4 显示树莓派的 CPU 温度

关于 bottle 库的安装方法, 请参考 7.17 节。

本示例含有 4 个文件, 它们都位于文件夹 web_sensor 下面。

web_sensor.py

包含用于 bottle 服务器的 Python 代码。

main.html

存放在浏览器中显示的网页。

justgage.1.0.1.min.js

一个显示温度计的第三方 JavaScript 库。

raphael.2.1.0.min.js

一个 justgage 库所需的代码库。

与本书中的所有示例程序一样, 你也可以通过本书网站 (<http://www.raspberrypiCookbook.com/>) 的代码下载区下载该示例程序。

若想运行该程序, 首先将目录切换到 web_sensor 下面, 然后通过下列命令来运行该 Python 程序。

```
$ sudo python web_sensor.py
```

然后, 从树莓派上面或者从树莓派所在网络的任意计算机上面打开一个浏览器, 并将树莓派的 IP 地址输入到浏览器的地址栏中, 这时就会看到如图 15-4 所示的页面了。

进一步探讨

主程序 web_sensor.py 实际上十分简单。

```
import os, time
from bottle import route, run, template

def cpu_temp():
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp = dev.read()[5:-3]
    return cpu_temp

@route('/temp')
def temp():
    return cpu_temp()

@route('/')
def index():
    return template('main.html')

@route('/raphael')
def index():
```

```
return template('raphael.2.1.0.min.js')

@route('/justgage')
def index():
    return template('justgage.1.0.1.min.js')

run(host='0.0.0.0', port=80)
```

函数 `cpu_temp` 的作用是读取树莓派 CPU 的温度，详情请参考 13.9 节。

然后，为网络服务器 `bottle` 定义了 4 个路径。第一个路径（`/temp`）返回一个字符串，其中包含了以摄氏度为单位的温度。根路径（`/`）将返回页面（`main.HTML`）的主 HTML 模板。另外两个路径提供对于 `raphael` 库和 `justgage` 库的副本的访问。

文件 `main.html` 主要用来保存渲染用户界面的 JavaScript 代码。

```
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
<script src="raphael"></script>
<script src="justgage"></script>

<script>
function callback(tempStr, status) {
if (status == "success") {
    temp = parseFloat(tempStr).toFixed(2);
    g.refresh(temp);
    setTimeout(getReading, 1000);
}
else {
    alert("There was a problem");
}
}

function getReading(){
    $.get('/temp', callback);
}
</script>
</head>

<body>
<div id="gauge" class="200x160px"></div>

<script>
var g = new JustGage({
    id: "gauge",
    value: 0,
    min: 10,
    max: 60,
    title: "CPU Temp 'C"
});
getReading();
</script>
```

```
</body>
</html>
```

其中，JQuery、Raphael 和 JustGage 这 3 个代码库都会被导入（JQuery 是从 <https://developers.google.com/speed/libraries/#jquery> 导入的，而另外两个则是从本地副本导入的）。为了将树莓派中的数据读入到浏览器窗口中，需要两步来完成。首先，需要调用 getReading 函数。这个函数会向 web_sensor.py 发送一个带有 /temp 路径的 Web 请求，并规定当这个 Web 请求完成后运行名为 callback 的函数。函数 callback 会更新 JustGage 的显示内容，并将再次调用 getReading 的间隔时间设置为 1 秒。

参考资料

关于利用 Tkinter 在应用程序而非网页内显示传感器读数的示例，请参考 13.19 节。

在显示传感器的值方面，代码库 JustGage 提供了所有常见的选项，详情请访问 <http://justgage.com/>。

15.3 使用 IFTTT 发送电子邮件及其他通知

面临问题

你想使用一种灵活的方式，让树莓派利用电子邮件、Facebook 状态、Twitter 或 Slack 来发送通知。

解决方案

让你的树莓派给 IFTTT 的 Maker 频道发送请求来触发各种可配置的通知。

下面我们通过一个示例来解释这个流程，例如当树莓派的 CPU 温度超过阀值时向你发送电子邮件。

为了使用这个流程，你首先需要创建一个 IFTTT 账户，为此，可以访问 www.ifttt.com 网站，并完成相应的注册。

下一步是创建一个新的 IFTTT 流程。非常让人不解的是 IFTTT 使用单词“流程”来表示规则，例如当我从树莓派收到一个 Web 请求的时候，就发送一封电子邮件。接下来，单击 Create a Recipe 按钮，这时会提示你先输入流程的 IF 部分，然后再输入 THAT 部分。

就本例来说，IF THIS 部分（触发器）是从树莓派收到一个 Web 请求，所以单击 THIS，在搜索字段中输入 Maker 找到这个 Maker 频道。然后，选中这个 Maker 频道，并且在提示你“Choose a trigger”时选择相应的选项（Receive a web Request）。这将打开如图 15-5 所示的窗口。

在 Event Name 字段中输入文本 `cpu_too_hot`，并单击 Create Trigger 按钮。

现在，你将进入该流程的 THAT 部分，即动作部分，你需要选择一个动作频道。虽然这里的选项很多，但是对于本例而言，你需要使用的是 Email 频道，所以在搜索字段中输入 Email，并选择 Email 频道。这时，你会看到多个与电子邮件相关的频道，其中包括 Gmail。请选择 email 频道，即使你打算使用 Gmail 账号发送邮件。



图 15-5 “Receive a web request” 触发器窗口

在选择好 Email 频道之后，请选择动作 Send Me an Email，这时将会显示如图 15-6 所示的窗口。

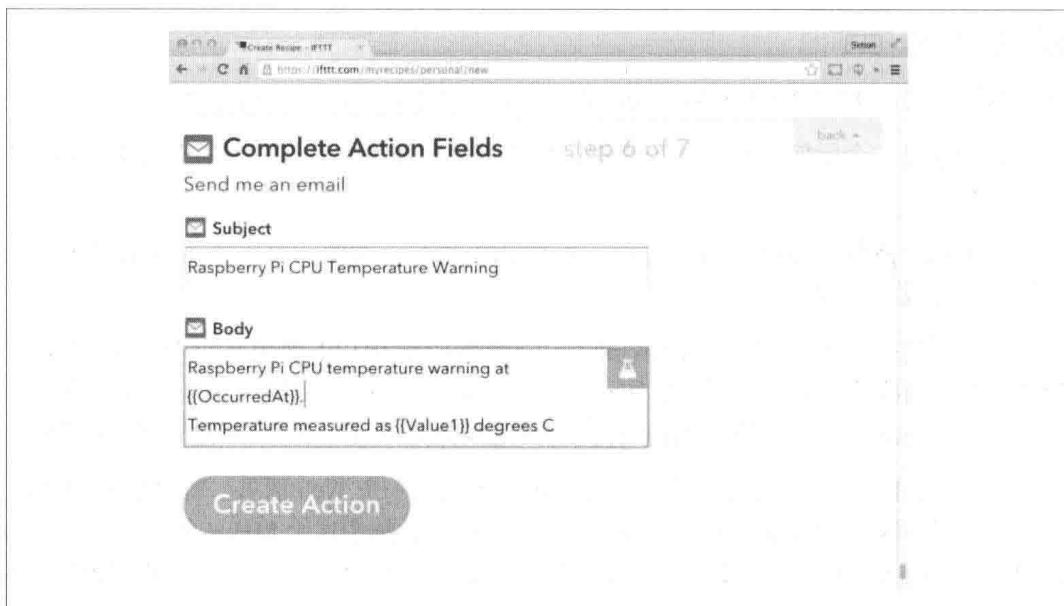


图 15-6 配置 Send Me an Email 动作

请你修改相应文本，使其与图 15-6 所示内容保持一致。需要注意的是 OccurredAt 和 Value1 这两个特殊值都要用{{和}}括住。这两个值被称为组成要素，并且是取自 Web 请求中的变量值，最终会代入电子邮件的主题和正文中。

最后，单击 Create Action 以及 Create Recipe，流程的创建工作便结束了。

至此，该流程已经生效了，所以只要树莓派给它发送了 Web 请求，它就会开始发送电子邮件。

这里用来发送 Web 请求的 Python 程序可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码下载区找到，其名称为 ifttt_cpu_temp.py。

```
import time, os, urllib, urllib2

MAX_TEMP = 37.0
MIN_T_BETWEEN_WARNINGS = 60 # Minutes

EVENT = 'cpu_too_hot'
BASE_URL = 'https://maker.ifttt.com/trigger/'
KEY = 'cyR3vPNF1P9K32W4NZB9cd'
def send_notification(temp):
    data = urllib.urlencode({'value1' : str(temp)})
    url = BASE_URL + EVENT + '/with/key/' + KEY
    response = urllib2.urlopen(url=url, data=data)
    print(response.read())

def cpu_temp():
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp = dev.read()[5:-3]
    return float(cpu_temp)

while True:
    temp = cpu_temp()
    print("CPU Temp (C): " + str(temp))
    if temp > MAX_TEMP:
        print("CPU TOO HOT!")
        send_notification(temp)
        print("No more notifications for: " + str(MIN_T_BETWEEN_WARNINGS) + " mins")
        time.sleep(MIN_T_BETWEEN_WARNINGS * 60)
    time.sleep(1)
```

在运行这个程序之前，你需要先取得 Maker Action 频道的访问密码，方法是选择 IFTTT 页面上方的 Channels，搜索 Maker，然后连接该频道，这时就能在页面底部看到自己的密码了（见图 15-7）。

将这个密码粘贴到 ifttt_cpu_temp.py 内以 KEY=开头的行中，这样就可以通过下列命令来运行该程序了。

```
$ sudo python ifttt_cpu_temp.py
```

现在，你可以设法来提高 CPU 的温度了，比如播放视频，或者给树莓派多加上一层外包装。一旦事件被触发，你就会收到一封电子邮件，具体如图 15-8 所示。你要注意观察这些值是如何代入电子邮件中去的。

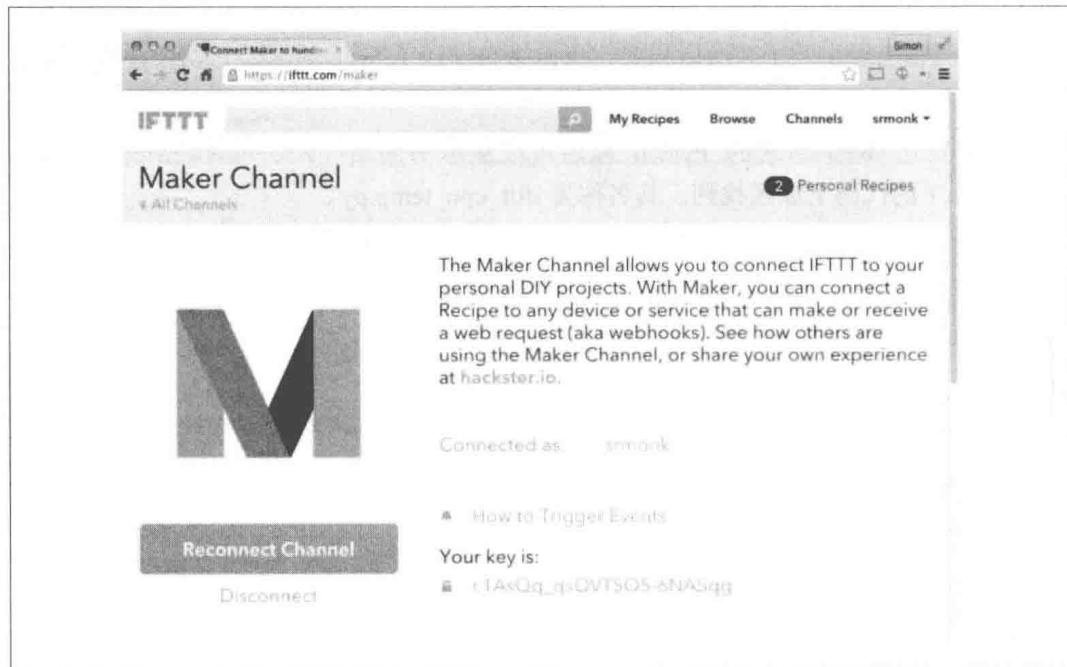


图 15-7 获取 IFTTT 密码

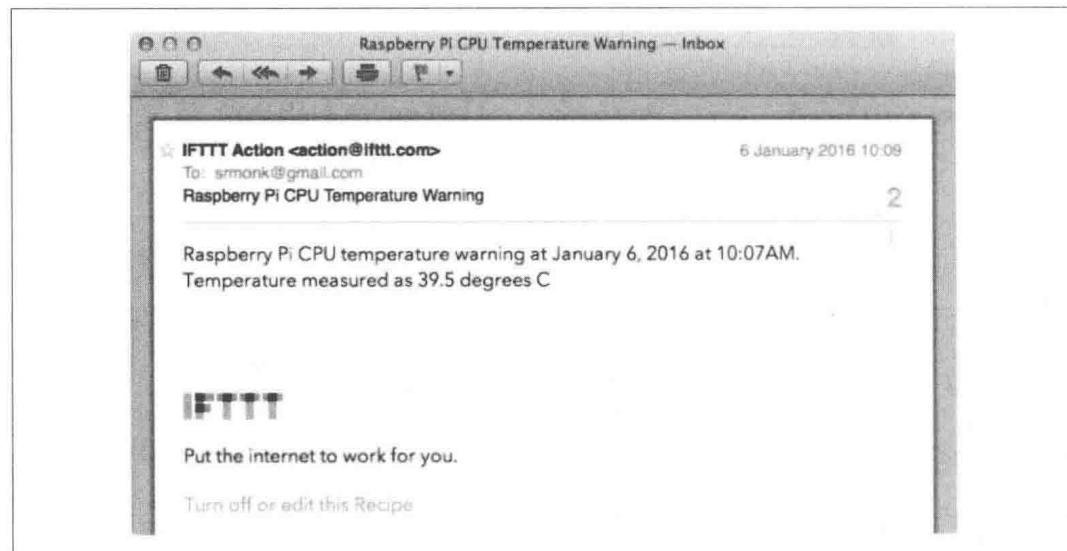


图 15-8 通知邮件

进一步探讨

对于这个程序来说，大部分动作都是在发送电子邮件的 `send_notification` 函数中进行的。它首先构造了一个 URL，其中包含了密码和请求参数 `value1`（存放温度），然后使用 Python 的 `urllib2` 向 IFTTT 发送 Web 请求。

主循环会不断地比较温度和 `MAX_TEMP` 的大小，如果 CPU 温度超过了 `MAX_TEMP`，就会发送 Web 请求，然后按照 `MIN_T_BETWEEN_WARNINGS` 的规定进入一个长时间的休眠，以防止你的邮箱被通知邮件塞满。

当然，除了使用 IFTTT 之外，你还可以利用 7.16 节介绍的方法直接发送电子邮件。但是，通过 IFTTT 发送消息的话，你不仅可以发送电子邮件通知，还可以使用 IFTTT 提供的所有动作频道，并且根本就不需要编写任何代码。

参考资料

关于直接使用 Python 发送电子邮件的方法，请参考 7.16 节。

测量 CPU 温度的代码的具体介绍，请参考 13.9 节。

15.4 利用 ThingSpeak 发送 Tweets

面临问题

你想利用树莓派自动发送 tweet，例如，为了让人们反感而不断将自己树莓派的温度发给他们。

解决方案

你可以直接使用 15.3 节介绍的方法，只要将 Action Channel 改为 Twitter 即可。但是，除此之外，你还可以使用 ThingSpeak 服务。

ThingSpeak (<https://thingspeak.com>) 类似于 IFTTT，但是它是直接面向 IoT 项目的。它允许创建利用 Web 请求存储和检索数据的频道，并提供了包括 ThingTweet 在内的许多动作，所以你可以把它看成是针对 Twitter 的 Web 服务封装。这些功能要比 Twitter API 更易于使用，因为后者要求你的应用登录 Twitter。

首先，请访问 <https://thingspeak.com> 并进行注册。然后，选择动作 ThingTweet(见图 15-9)。这时，将要求你登录 Twitter。

这里用来触发 tweet 的 Python 程序可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码下载区找到，其名称为 `send_tweet.py`。

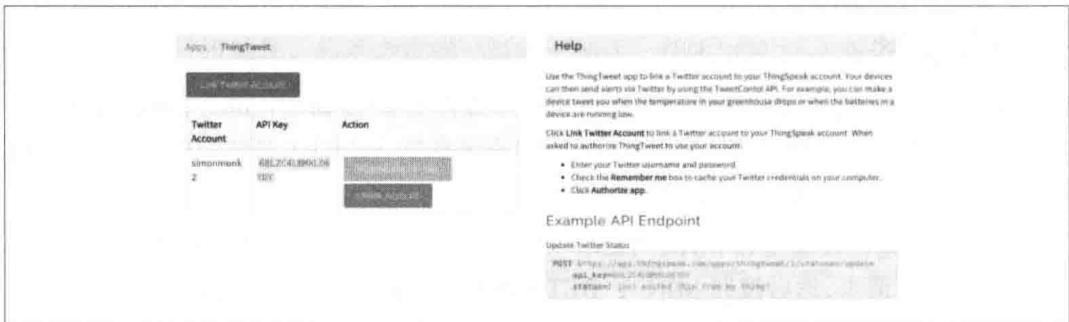


图 15-9 ThingTweet 动作

```
import time, os, urllib, urllib2

MAX_TEMP = 37.0
MIN_T_BETWEEN_WARNINGS = 60 # Minutes

BASE_URL = 'https://api.thingspeak.com/apps/thingtweet/1/statuses/update/'
KEY = '68LZC4LBMXL06YDY'

def send_notification(temp):
    status = 'Raspberry Pi getting hot. CPU temp=' + temp
    data = urllib.urlencode({'api_key' : KEY, 'status': status})
    response = urllib2.urlopen(url=BASE_URL, data=data)
    print(response.read())

def cpu_temp():
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp = dev.read()[5:-3]
    return cpu_temp

while True:
    temp = cpu_temp()
    print("CPU Temp (C): " + str(temp))
    if temp > MAX_TEMP:
        print("CPU TOO HOT!")
        send_notification(temp)
        print("No more notifications for: " + str(MIN_T_BETWEEN_WARNINGS) +
" mins")
        time.sleep(MIN_T_BETWEEN_WARNINGS * 60)
    time.sleep(1)
```

就像 15.3 节那样，在运行该程序之前，你需要将图 15-9 中的密码粘贴到相应的代码中。然后，你就可以按照 15.3 节中介绍的方法来运行和测试这个程序了。

进一步探讨

这里的代码与 15.3 节中的非常类似，主要不同之处在于 `send_notification` 函数，它的作用是构建 tweet，然后发送 Web 请求，其中以参数状态作为消息。

参考资料

关于 ThingSpeak 服务的完整文档, 请访问页面 <https://uk.mathworks.com/help/thingspeak/>。

在 15.5 节中, 你将会掌握流行的 CheerLights, 它也是利用 ThingSpeak 实现的; 而在 15.6 节中, 你将学习如何使用 ThingSpeak 来收集传感器数据。

15.5 CheerLights

面临问题

你想给自己的树莓派连接一个 RGB LED, 从而参与到流行的 CheerLights 项目中去。

Cheerlights 是一种 Web 服务, 当人们向@CheerLights 发送包含颜色名称的 tweet 时, 它就会把最新的颜色记录下来。在这个世界上, 已经有很多人参与了 Cheerlight 项目: 他们利用 Web 服务请求最新的颜色, 然后让灯光显示相应的颜色。所以, 当有人发送 tweet 时, 所有人的灯光都会变色。

解决方案

你可以给自己的树莓派连接一个 Raspberry Squid RGB LED (见图 15-10), 然后运行名为 cheerlights.py 的测试程序, 该程序可以从本书网站 (<http://www.raspberrypicookbook.com/>) 上面下载。

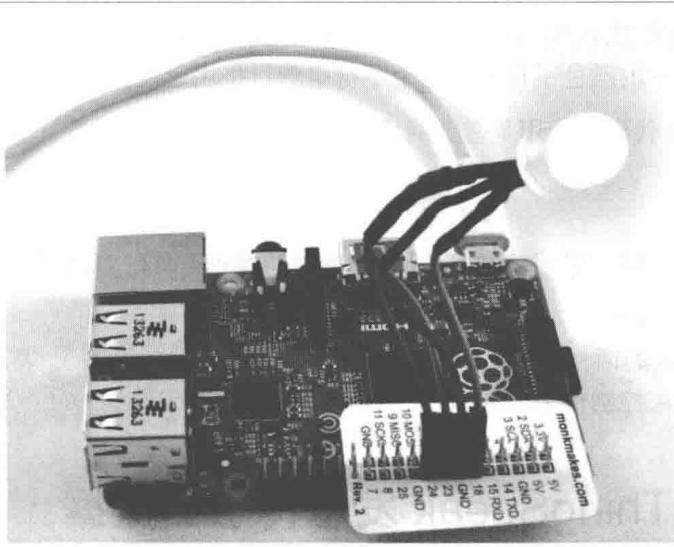


图 15-10 Cheerlight

```
from squid import *
import urllib, time
```

```
squid = Squid(18, 23, 24)
cheerlights_url = "http://api.thingspeak.com/channels/1417/field/2/last.txt"

try:
    while True:
        try:
            cheerlights = urllib.urlopen(cheerlights_url)
            c = cheerlights.read()
            cheerlights.close()
            print(c)
            squid.set_color_rgb(c)
        except:
            print('Error')
            time.sleep(2)

    finally:
        GPIO.cleanup()
```

运行这个程序之前，你需要安装 Squid 库（见 9.10 节）。

你的 LED 将立即把自己设置为最新的颜色。但是，如果不久之后有人发送了 tweet，它的颜色就可能随之改变；如果没变的话，你可以自己发送一条消息，比如“@cheerlights red”，这时你的 LED 以及参与该项目的世界各地的 LED 都会随之改变。对于 CheerLights 来说，合法的颜色名称包括 red（红色）、blue（蓝色）、cyan（青色）、white（白色）、oldlace（浅米色）、purple（紫色）、magenta（洋红色）、yellow（黄色）、orange（橘色）和 pink（粉红色）。

进一步探讨

上面的代码只是向 ThingSpeak 发送一个 Web 请求，然后 ThingSpeak 会返回一个 6 位的十六进制数表示的颜色字符串，这个字符串随后将用于设置 LED 的颜色。

上面的 try/except 代码的作用是避免网络临时故障时程序发生崩溃。

参考资料

CheerLights 使用 ThingSpeak 来保存频道中的最新颜色，而在 15.6 节中，频道则用于记录传感器的数据。

如果你没有 Squid 的话，可以在面包板上使用 RGB（见 10.10 节），甚至可以利用 14.7 节中介绍的方法来控制整个 LED 灯条。

15.6 向 ThingSpeak 发送传感器数据

面临问题

你想把传感器数据记录到 ThingSpeak 上面。

解决方案

登录到 ThingSpeak，然后从 Channels 的下拉菜单中选择 My Channels。然后，按照图 15-11 所示的提示填写窗口顶部的相应内容就可以新建一个频道了。

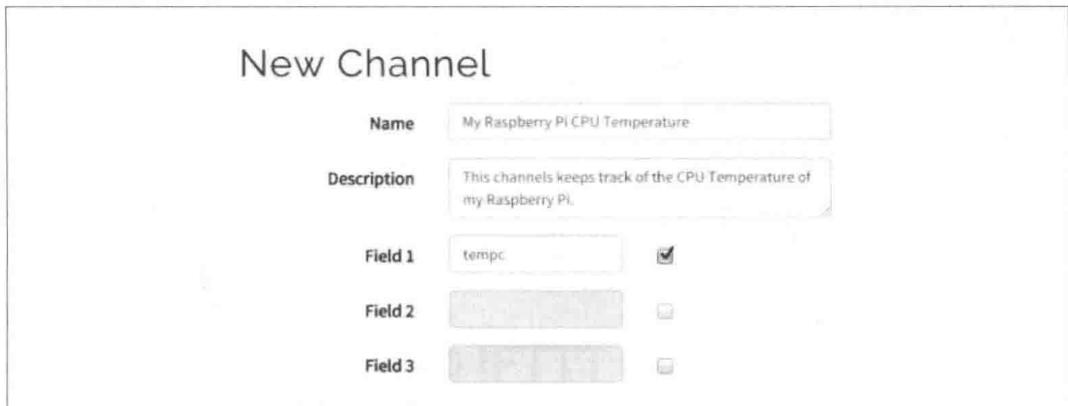


图 15-11 在 ThingSpeak 中创建频道

这个窗口中的其余内容不必填写，编辑完成之后，单击页面底部的 Save Channel 按钮。单击 Data Import/Export 选项卡，就能看到可供使用的 Web 请求，以及刚才所建通道的定制信息总结（见图 15-12）。



图 15-12 向 ThingSpeak 频道提供数据

为了给这个频道发送数据，必须发送相应的 Web 请求。同时，发送 Web 请求的 Python 程序可以从本书网站 (<http://www.raspberrypicookbook.com>) 上面下载，该程序名为

thingspeak_data.py。

```
import time, os, urllib, urllib2

PERIOD = 60 # Seconds

BASE_URL = 'https://api.thingspeak.com/update.json'
KEY = 'DYHHDDKKLU8OV58T'

def send_data(temp):
    data = urllib.urlencode({'api_key' : KEY, 'field1': temp})
    response = urllib2.urlopen(url=BASE_URL, data=data)
    print(response.read())

def cpu_temp():
    dev = os.popen('/opt/vc/bin/vcgencmd measure_temp')
    cpu_temp = dev.read()[5:-3]
    return cpu_temp

while True:
    temp = cpu_temp()
    print("CPU Temp (C): " + str(temp))
    send_data(temp)
    time.sleep(PERIOD)
```

然后，运行该程序。这样你就可以在 Private View 选项卡的 ThingSpeak 频道页面中看到如图 15-13 所示的图形。

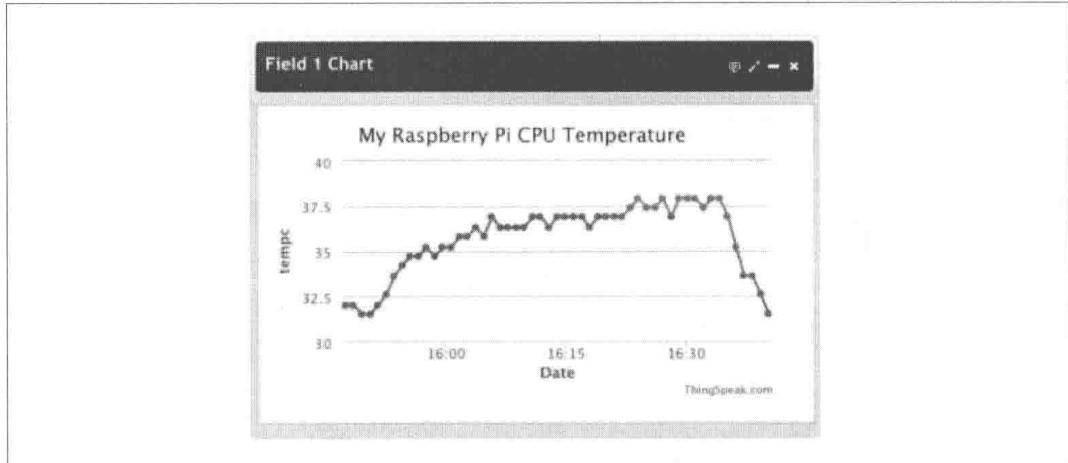


图 15-13 绘制传感器数据

对于这个图形来说，每当有新的读数到达时都会进行更新，读数的更新频率为每分钟一次。

进一步探讨

变量 PERIOD 的作用是确定发送温度的时间间隔，它的单位是秒。

函数 send_data 用来构建 Web 请求，并通过参数 field1 来提供温度。

如果你的数据是公益性的，例如准确的环境数据，那么你可能希望公开这个频道，以便所有人都能利用这些数据。但是就树莓派的 CPU 温度来说，其好像不属于这种情况。

参考资料

关于将传感器数据导出到电子表格的示例，请参考 13.20 节。

关于读取 CPU 温度的代码的相关解释，请参考 13.9 节。

15.7 使用 Dweet 和 IFTTT 响应 Tweet

面临问题

你想让树莓派执行某些操作，以响应 tweet 中特定的 # 标签和@标签。

实际上，15.5 节介绍的方法就可以完成这个事情，只不过它的效率太低了，因为它是通过不断轮询 Web 请求的方式来检测颜色是否发生改变的。

解决方案

在监视 tweet 的时候，有一种不依赖于轮询的高效机制，即使用 IFTTT（见 15.3 节）去辨认感兴趣的推特，然后向名为 Dweet 的服务发送 Web 请求。Dweet 会向运行在树莓派上面的 Python 程序推送通知，如图 15-14 所示。

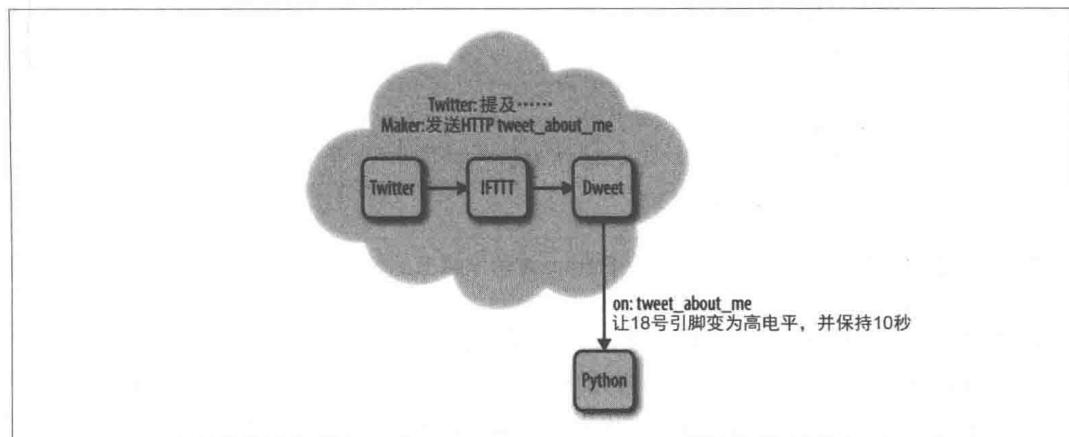


图 15-14 联合使用 IFTTT、Dweet 和 Python

举例来说，每次 Twitter 上提及你的用户名的时候，你就让连接到面包板的 LED 或 Raspberry Squid 的 LED 闪烁 10 秒钟。

就硬件而言，本节对所用的电子元件的要求并不高，只要当 GPIO 18 引脚变成高电平时它们能够引发我们的注意就行了。它既可以是 Raspberry Squid（见 9.10 节）的一个通道，也可以是连接到面包板的单个 LED（见 10.1 节），为灵活性起见，甚至可以是一个继电器（见 10.6 节）。如果选用继电器的话，你可以创建类似 Bubblino 这样的项目（<http://bubblino.com/>）。

第一步是登录 IFTTT（见 15.3 节），然后创建一个新的流程。

然后，选择一个名为 New Mention of You 的动作频道，并单击 Create Trigger。对于这个流程的动作频道，请选择 Maker，然后选择动作“Make a Web Request”，并按照图 15-15 所示的提示填写相关字段。

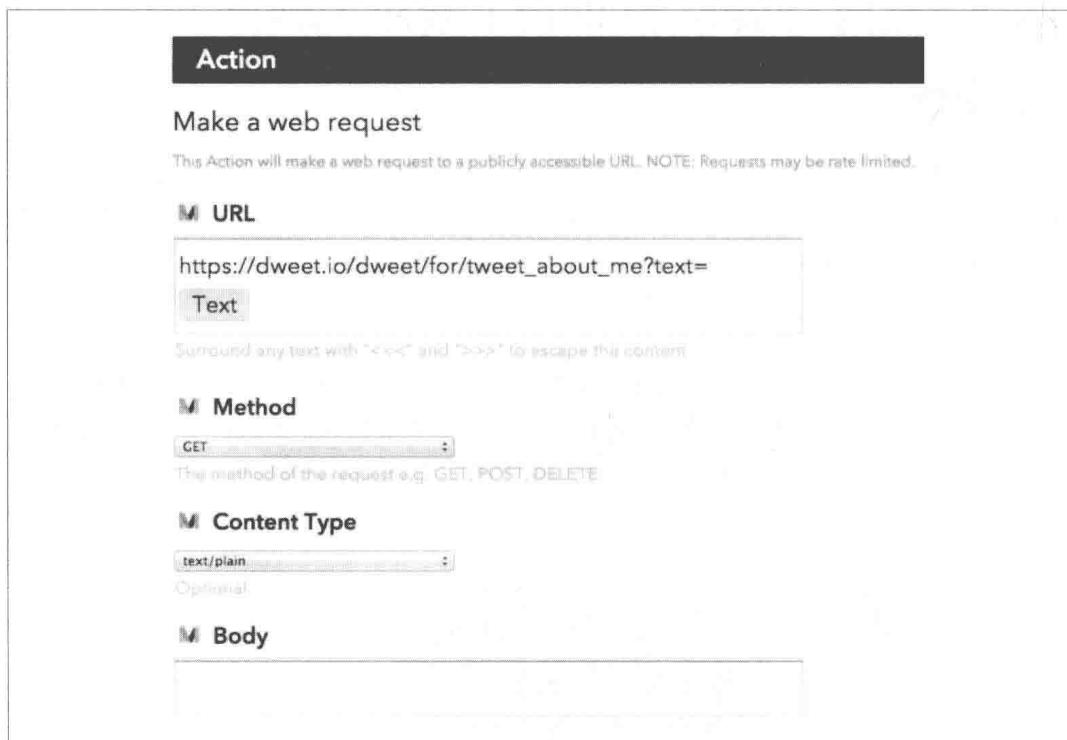


图 15-15 在 IFTTT 中填写 Send HTTP Request 动作频道的相关内容

该网址包含了一个带有文本组成要素的请求参数，它用于存放 tweet 的正文。当然，除非要将其打印到控制台，否则是不会用到它的，但是对于更加高级的项目来说，你还可以将消息显示到 LCD 屏幕上面，所以掌握如何将来自 tweet 的数据传递给 Python 程序还是非常重要的。

然后，单击 Create Recipe，以使这个 IFTTT 流程生效。

Web 服务 dweet.io 类似于 IoT 世界中的 Twitter，你可以通过它提供的 Web 接口来发表和侦听 dweet。

使用 Dweet 的时候，它不要求用户进行注册，所以也就无需登录了。你只要有一个向 Dweet 发送消息的东西（本例为 IFTTT），同时有另一个东西（树莓派的 Python 程序）等待你关心的事情发生后的通知。就本例来说，连接着两个东西的令牌为 tweet_about_me。由于这个令牌不是非常独特，所以如果几个人同时尝试本书中的这个示例的话，那么他们将会得到对方的消息。要想避免这个问题，就需要使用更独特的令牌（例如，给这个消息添加一个由数字和字母组成的随机字符串）。

为了从 Python 程序访问 Dweet，你需要安装 dweepy 库，具体命令如下所示。

```
$ git clone git://github.com/paddycarey/dweepy.git  
$ cd dweepy  
$ sudo python setup.py install
```

用于这个流程的程序可以从本书网站 (<http://www.raspberrypicookbook.com/>) 下载，具体名称为 twitter_trigger.py。

```
import time  
import dweepy  
import RPi.GPIO as GPIO  
  
KEY = 'tweet_about_me'  
OUTPUT_PIN = 18  
OUTPUT_DURATION = 10  
GPIO.setmode(GPIO.BCM)  
GPIO.setup(OUTPUT_PIN, GPIO.OUT)  
  
while True:  
    try:  
        for dweet in dweepy.listen_for_dweets_from(KEY):  
            print('Tweet: ' + dweet['content']['text'])  
            GPIO.output(OUTPUT_PIN, True)  
            time.sleep(OUTPUT_DURATION)  
            GPIO.output(OUTPUT_PIN, False)  
    except Exception:  
        pass
```

进一步探讨

该程序使用 listen_for_dweets_from 方法维护到 dweet.io 的连接，侦听该服务器推送来的所有报文，最后通过来自 IFTTT 的 dweet 来响应 tweet。语句块 try/except 的作用是

确保在遇到通信中断的时候，该程序会马上重启侦听进程。

参考资料

在 15.5 节中还介绍了一个与本示例类似的项目，但是它使用了一种不同的方法。

Arduino 与树莓派

16.0 引言

虽然树莓派非常适合于需要网络连接、图形用户界面以及低压 GPIO 输出的项目，但是，与 Arduino（见图 16-1）之类的单片机电路板相比，其劣势在于缺乏模拟输入。幸运的是通过与树莓派相连，Arduino 就能够与外围电子元件进行交互了，真可谓两全其美。

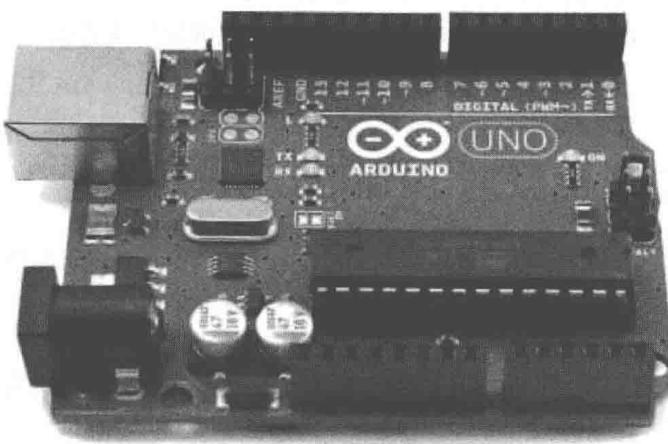


图 16-1 Arduino Uno 电路板

Arduino 电路板不仅在外观上与树莓派相似，而且它们本质上也都是一台微型电脑。不

过，与树莓派相比，Arduino 还是有许多显著不同之处的。

- 没有键盘、鼠标和屏幕接口。
- 只有 2KB RAM 和 32KB 的闪存可用于存储程序。
- 其处理器主频只有 16MHz，而树莓派的主频则高达 700MHz。

看到这里，你可能会问，既然这种单片机功能如此之弱，为什么不直接使用树莓派呢？

答案是较之于树莓派，Arduino 电路板（最常见的是 Arduino Uno）在与外部电子设备的交互方面更具优势。

例如，Arduino 具有：

- 14 个输入/输出引脚，类似于树莓派的 GPIO 引脚。并且，每个引脚可以提供高达 40mA 电流，而原装树莓派的则只有 3mA。这样，它无需借助其他电子元件就能给更多的设备供电。
- 6 个模拟输入。这使得连接模拟传感器更加容易（见 16.7 节）。
- 6 个 PWM 输出。这些输出都是通过硬件定时的，所以能够产生比树莓派更加准确的 PWM 信号，从而更好地控制伺服电机。
- 各种插件式 Shield 能够满足所有设备的需求，这些设备包括从电机控制到 LCD 显示器等。

在许多方面，采用树莓派与 Arduino 来共同处理所有底层设备是一个非常棒的组合，因为它能够发挥两块电路板的优势。如果使用带有 GPIO 接口并含有类似于 aLaMode（见 16.13 节）这样的兼容 Arduino 硬件的接口板的话，就可以将树莓派与 Arduino 的功效发挥到极致。

16.1 通过树莓派对 Arduino 进行编程

注意：务必访问网站 <http://razzpisampler.oreilly.com> 观看与本节相关的内容。

面临问题

你希望在树莓派上面运行 Arduino IDE，以便为 Arduino 编写和加载程序。

解决方案

Arduino IDE 可以在树莓派上面运行，虽然有些慢，不过还能够使用。

在使用 Arduino IDE 的时候，你一定很想利用树莓派 2 的速度优势。为了安装 Arduino IDE，可以输入如下所示的命令。

```
$ sudo apt-get update  
$ sudo apt-get install arduino
```

安装之后，你会在程序菜单中发现一个 Electronics 分组（见图 16-2）。



图 16-2 运行在树莓派上面的 Arduino IDE

要想通过 Arduino IDE 进行编程，需要利用 USB 线连接到树莓派上面。

在配置树莓派使用 Arduino IDE 的时候，最简单的办法是运行 Kevin Osborn 提供的一个脚本，该脚本能对串行端口和 Arduino 进行必要的配置，使它们可以正常运行起来。使用这种方式的一个好处是它会同时配置好 aLaMode 电路板（见 16.13 节）。

为了下载和运行这个脚本，你需要执行如下所示的命令。

```
$ https://github.com/wyolum/alamode/raw/master/bundles/alamode-setup.tar.gz  
$ tar -xvf alamode-setup.tar.gz  
$ cd alamode-setup  
$ sudo ./setup
```

现在你可以将 Arduino 连接到树莓派上面。从 Tools 菜单中选择 Board，并将电路板类型设置为 Arduino Uno。然后，从 Serial Port 选项中选择 /dev/ttyACM0。为了上载令 Arduino 闪烁指示灯的测试程序，请选择 File 菜单，然后单击“Examples, Basic”，最后单击 Blink。单击工具栏上的向右箭头键，就可以开始编译和上载过程。如果一切正常的话，你应该在 IDE 窗口底部的状态区中看到“Done Uploading”的消息。



如果你已经插入 Arduino，但是列出的设备中没有发现 ttyACM0 的话，请尝试重新启动 Arduino IDE。如果情况依旧的话，那么就得重新启动树莓派了。在重启并再次打开 Arduino IDE 的时候，要确保 Arduino 一直处于连接状态。

进一步探讨

为了高效地使用 Arduino 和树莓派，你需要学习一些 Arduino 的编程技术。这时候，你可能会发现“*programming Arduino :Getting Started with Sketches*”（McGraw -Hill /Tab Books）能够给你带来真正的帮助。

但是，如果借助于 PyFirmata 项目的话，你就能够在无需编写任何 Arduino 端代码的情况下使用 Arduino 了。

参考资料

Arduino IDE 的配置脚本来自于 Bald Wisdom 的博客 (<http://bit.ly/IFZ5zK>)。

16.2 利用 Serial Monitor 与 Arduino 进行通信

面临问题

你希望显示一条发送自 Arduino 的消息。

解决方案

Arduino IDE 提供了一个名为 serial monitor 的功能，不仅可以通过 USB 线缆发送文本消息到 Arduino，还可以查看从 Arduino 发来的消息。

要想尝试这个功能，你首先要编写一个简短的 Arduino 程序（在 Arduino 的世界里，程序被称为 sketch）。这个程序的功能非常简单，每秒一次不停发送一条消息。

这个 Arduino sketch 的代码如下所示。如同本书中的其他示例程序一样，你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 下载它，该程序位于 ArduinoHello 文件夹下面。你只需要访问本书链接，然后单击 Code 即可。

你可以通过菜单 File→New 来新建一个 sketch，并将下列文本粘贴进去，然后将其上载到 Arduino 中。

```
void setup()
{
    Serial.begin(9600);
}
void loop()
```

```
{  
    Serial.println("Hello Raspberry Pi");  
    delay(1000);  
}
```

一旦该 sketch 上载至 Arduino，它就会通过串口发送消息“Hello Raspberry Pi”。为了看到这则消息，需要通过单击工具栏右侧的放大镜图标来打开 serial serial monitor(见图 16-3)，否则你是看不到它的。

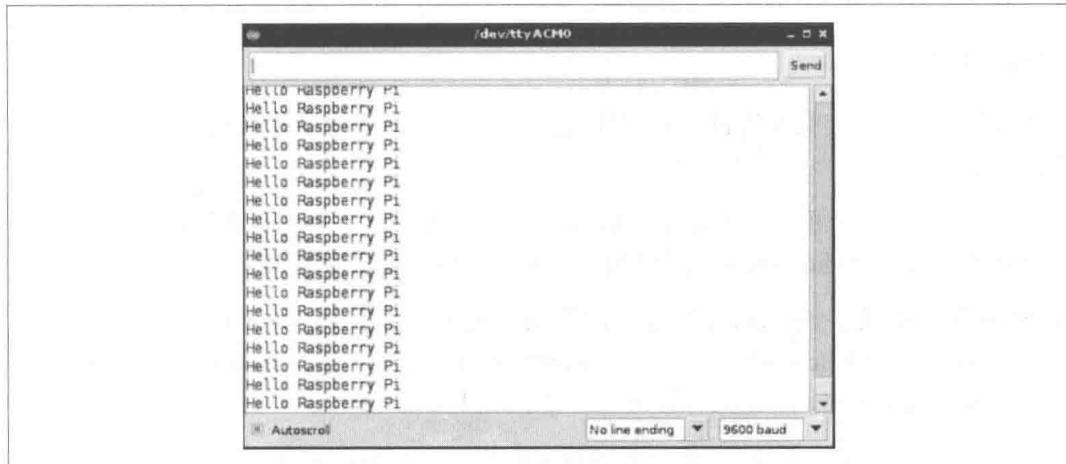


图 16-3 利用 serial monitor 查看消息

这个 serial monitor 的右下角有一个下拉列表，你可以在此选择波特率（通信速度）。如果波特率没有被设置为 9600 的话，请将它改为这个数值。

进一步探讨

本章后面的几节（16.10 节和 16.11 节）将会介绍如何编写自己的代码来跟树莓派上面的 Python 程序进行通信，这样就无需运行 Arduino IDE 了。

还有一个更加通用的方法，那就是使用所谓的 PyFirmata，在它的帮助下你就不必在树莓派上进行任何的编码工作了。关于 PyFirmata 的详细介绍，请参考 16.3 节。

参考资料

实际上，Arduino 非常易于学习。下面是部分图书和在线资源的链接，它们都是些不错的入门读物。

- Programming Arduino: Getting Started with Sketches (<http://amzn.to/1NmngE8>) (Tab Books) by Simon Monk
- The official Arduino Getting Started Guide (<http://bit.ly/IGAqel>)
- The Adafruit Arduino lesson series (<http://bit.ly/1dTdI1h>)

- Arduino Cookbook (O'Reilly) by Michael Margolis

16.3 配置 PyFirmata 以便通过树莓派来控制 Arduino

面临问题

你希望可以将 Arduino 用作树莓派的接口板。

解决方案

你可以将 Arduino 连接到树莓派的 USB 接口上，这样就能通过计算机为其提供电源并与之通信了。

然后，在 Arduino 上面安装 Firmata sketch，同时在树莓派上面安装 PyFirmata。这样做的话就要求安装 Arduino IDE，所以如果尚未安装的话，请参考 16.1 节的介绍。

Arduino IDE 提供了 Firmata，所以为了将 Firmata 安装到你的 Arduino 电路板上面，你只需上载一个 sketch 即可，这个 sketch 可以通过 File→Examples→Firmata→StandardFirmata 找到。

安装好 Firmata 之后，Arduino 就开始等待来自树莓派的通信了。

现在，你需要安装的是 PyFirmata，它是该连接的另一方。这需要用到 PySerial 库，为此，你可以参考 9.6 节介绍的安装方法。

现在，只要输入下列命令就可以下载和安装 PyFirmata 了。

```
$ git clone https://github.com/tino/pyFirmata.git  
$ cd pyFirmata  
$ sudo python setup.py install
```

你可以在 Python 控制台中尝试运行 PyFirmata。输入下列命令后，就会打开 Arduino 13 号引脚上（标记为 L）内置的 LED 了，并且随后将其关闭。

```
$ sudo python  
Python 2.7.3 (default, Jan 13 2013, 11:20:46)  
[GCC 4.6.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
  
>>> import pyfirmata  
>>> board = pyfirmata.Arduino('/dev/ttyACM0')  
>>> pin13 = board.get_pin('d:13:o')  
>>> pin13.write(1)  
>>> pin13.write(0)  
>>> board.exit()
```

进一步探讨

上面的代码首先导入了 PyFirmata 库，然后建立了一个名为 board 的 Arduino 实例，并

且以 USB 接口 (`/dev/ttyACM0`) 作为其参数。之后，你获得了一个 Arduino 引脚（本例为 13）的引用，并将其设为数字输出。字母 d 表示数字的，13 表示引脚编号，o 表示输出。

为了将这个输出引脚设置为高电平，你可以使用 `write(1)`；如果要将其设置为低电平，可以使用 `write(0)`。此外，你也可以使用 `True` 和 `False` 来代替 1 和 0。

图 16-4 展示了一个电路板两侧都带有一行连接的 Arduino。

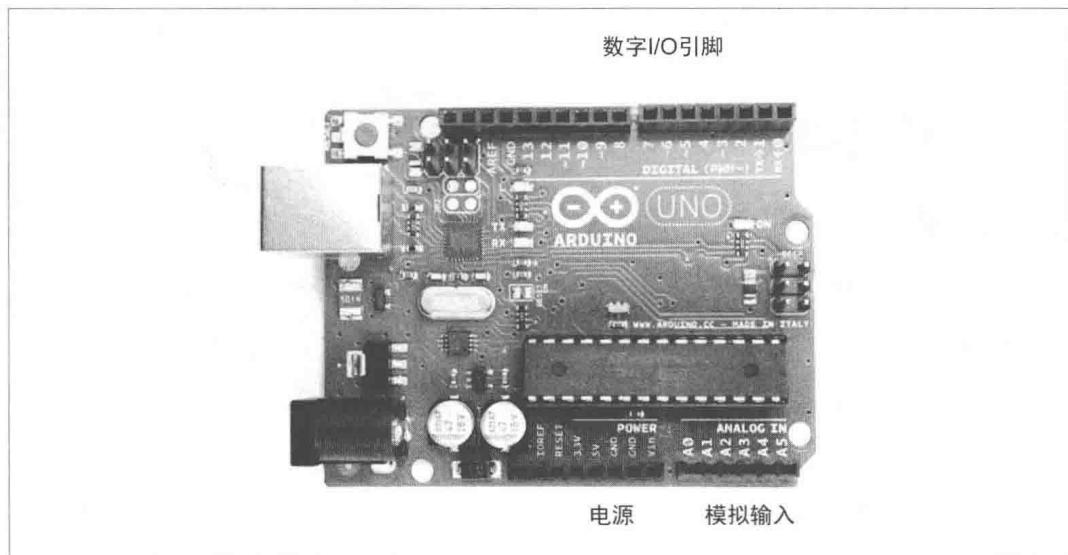


图 16-4 Arduino Uno 上的 I/O 引脚

其中，图上部标记为 0 到 13 的引脚可以用于数字输入或输出。这些引脚中的一部分也可以用于其他用途。引脚 0 和引脚 1 可以用于串行接口，同时，在使用 USB 端口的时候，也会用到这两个引脚。连接到板载 LED 的引脚 13 被标记为 L。数字 I/O 引脚 3、5、6、9、10 和 11 的边上都标记了~符号，这表示它们可以用于 PWM 输出（见 10.3 节）。

在电路板的另一面也有一组接口，可以提供 5V 和 3.3V 电源，以及标记为 A0 到 A5 的 6 个模拟输入。

虽然 Arduino Uno 本身需要 50mA 左右的电流，但是树莓派本身使用的电流是它的 10 左右，所以通过树莓派的 USB 连接来给 Arduino 供电是完全可能的。但是，当你开始给 Arduino 连接大量外部设备的时候，电流的消耗就会急剧上升，这时，你就需要使用 DC 筒式接口作为 Arduino 自身的电源适配器。它可以提供 7V 到 12V 的直流电。

在使用 Firmata 的时候，唯一的缺点就是所有指令都来自于树莓派，而 Arduino 独立运

行时的能力并没有充分利用。对于高级的项目来说，你可能不会编写代码让 Arduino 接收来自树莓派的命令或向树莓派发送消息，而是让它干其他的事情。

参考资料

下面的几节将会考察通过 PyFirmata 使用 Arduino 的全部引脚特性：16.4 节、16.5 节、16.6 节、16.7 节和 16.8 节。

你可以在 PyFirmata 的 GitHub 页面 (<http://bit.ly/Iucv2r>) 上找到 PyFirmata 的官方文档。

16.4 通过树莓派对 Arduino 的数字输出进行写操作

面临问题

你想在树莓派上面通过 Python 控制 Arduino 的数字输出。

解决方案

在 16.3 节中，介绍了让 Arduino 电路板内置的 LED（标记为 L）闪烁的方法。下面，将介绍如何连接外部的 LED，并编写一个 Python 程序使其闪烁。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- 270Ω 电阻器；
- LED。

除了使用面包板和 LED 外，你还可以插入 Squid RGB LED 的一个彩色通道（见 9.10 节）。

请连接面包板，并按照图 16-5 所示将元件固定到 Arduino 上面。

如果你尚未配置 PyFirmata 的话，具体配置方法请参考 16.3 节。

下面的 Python 脚本将会令 LED 以 1Hz 的频率进行闪烁。打开一个编辑器(nano 或者 IDLE)并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代码下载区下载相应的程序。本节的示例程序名为 ardu_lash.py。

```
import pyfirmata  
import time  
  
board = pyfirmata.Arduino('/dev/ttyACM0')
```

```
led_pin = board.get_pin('d:10:o')

while True:
    led_pin.write(1)
    time.sleep(0.5)
    led_pin.write(0)
    time.sleep(0.5)
```

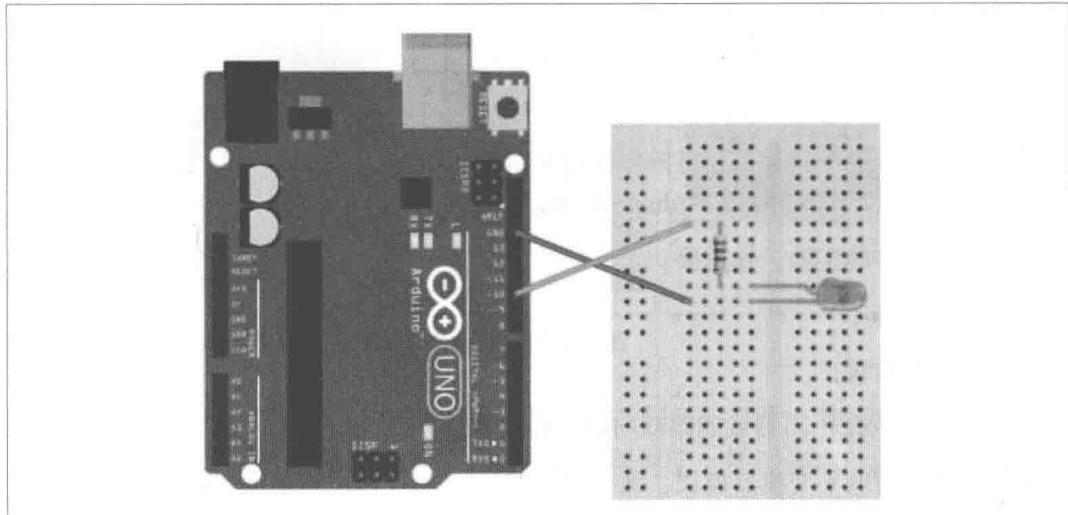


图 16-5 Arduino 和 LED 布线图

进一步探讨

本示例与 10.1 节中连接 LED 到树莓派的例子非常相似。然而，需要注意的是由于 Arduino 的输出可以提供比树莓派的输出大得多的电流，所以，你可以使用一个阻值较小的电阻，使得 LED 更亮一些。Arduino 的输出也是 5V，而不是 3.3V。

如果你希望使用 10.8 节中那样的用户界面来控制 LED 的话（见图 16-6），这也非常简单，只要修改一下代码就可以了。你可以找到一个已经修改好的代码，名为 ardu_gui_switch.py，你可以从本书网站上面下载它。需要牢记的是这个程序无法从 SSH 命令行中运行。你必须能够访问树莓派的图形环境，这样的话，你才能看到这个用户界面。

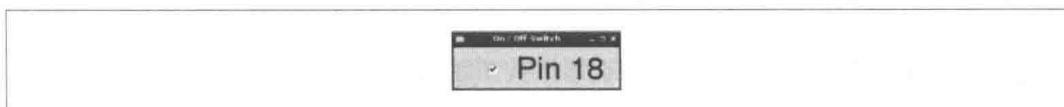


图 16-6 用于开关其他设备的用户界面

参考资料

关于直接利用树莓派控制 LED 的开关的例子，请参考 10.1 节。

16.5 使用 PyFirmata 与 TTL 串口

面临问题

你希望通过串行连接（GPIO 接口上面的 RXD 和 TXD 引脚）而非 USB 来使用 PyFirmata。

解决方案

你可以使用一个电平转换器，将树莓派的 RXD 引脚连接至 Arduino 的 Tx 引脚，同时，将树莓派的 TXD 引脚连接至 Arduino 的 Rx 引脚。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- 270Ω 和 470Ω 电阻器或四路双向电平转换器。

如果你打算使用电平变换器模块，那么可以按照图 16-7 所示连接面包板。

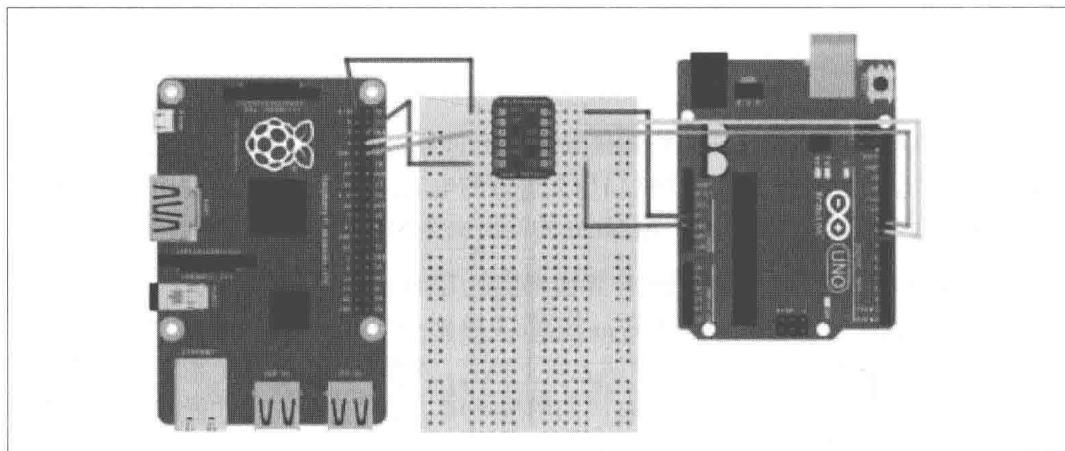


图 16-7 利用电平转换模块与 Arduino 进行串行通信的布线图

反之，如果你打算使用一对电阻，那么可以按照图 16-8 所示方式连接面包板。

Arduino 的 Rx 输入可以直接使用来自树莓派的 TXD 引脚的 3.3V 输出，但是，来自 Arduino 的 Tx 引脚的 5V 电压必须降低为适用于树莓派的 3V 电压。

此外，你还需要配置 PyFirmata，具体请参考 16.3 节。该项目的 Arduino 端仍然与 16.4 节中的严格保持一致，即使用 USB，而非使用串行连接。唯一需要修改的地方来自在

树莓派上运行的一个 Python 程序，为此，只需将设备名称从 /dev/ttyACM0 改为 /dev/ttyAMA0 即可，这是因为串行端口的名称不同于 USB 接口。

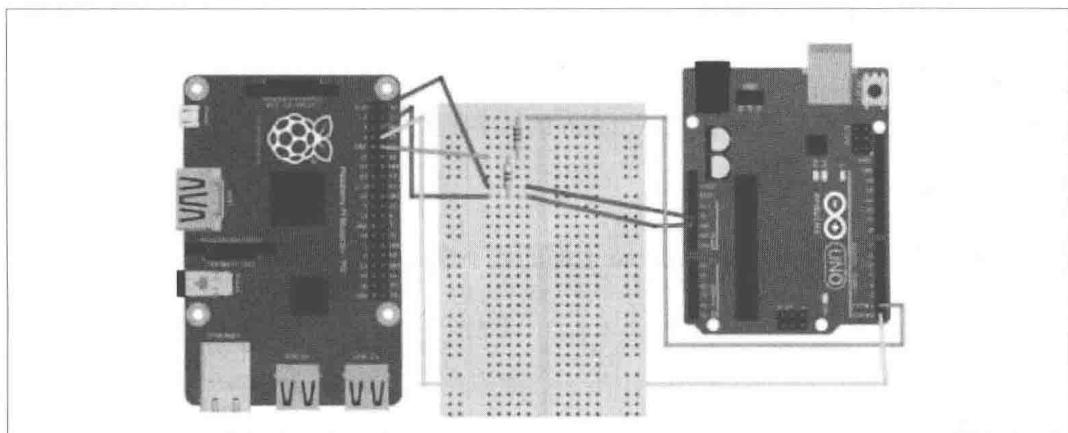


图 16-8 将一对电阻器用于 Arduino 串行通信的布线图

下面的 Python 脚本将会令 LED 以 1Hz 的频率进行闪烁。打开一个编辑器（ nano 或者 IDLE ）并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站（ <http://www.raspberrypicookbook.com> ），从代码下载区下载相应的程序。本节的示例程序名为 ardu_lash_ser.py 。

```
import pyfirmata
import time

board = pyfirmata.Arduino('/dev/ttyAMA0')
led_pin = board.get_pin('d:13:o')

while True:
    led_pin.write(1)
    time.sleep(0.5)
    led_pin.write(0)
    time.sleep(0.5)
```

进一步探讨

这里的电平转换器是必不可少的，因为树莓派的串行端口连接、RXD 和 TXD 必须使用 3.3V 电平，而 Arduino Uno 使用的是 5V 电压。工作电压为 5V 的 Arduino 可以使用 3.3V 的信号，但是反过来却不行，因为将 5V 信号连接到 3.3V 的 RXD 引脚上面的时候，很可能会对树莓派造成损坏。

参考资料

你也可以轻松修改其他使用 PyFirmata 的示例（从 16.5 节到 16.8 节中都有这种示例），由 USB 连接改为串行连接，只需将 Python 程序中的设备名修改一下即可。

16.6 使用 PyFirmata 读取 Arduino 的数字输入

面临问题

你想在树莓派上通过 Python 读取 Arduino 的数字输入。

解决方案

你可以使用 PyFirmata 来读取 Arduino 上面的数字输入。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- $1k\Omega$ 电阻器；
- 轻触按钮开关。

请连接面包板，并按照图 16-9 所示将元件固定到 Arduino 上面。

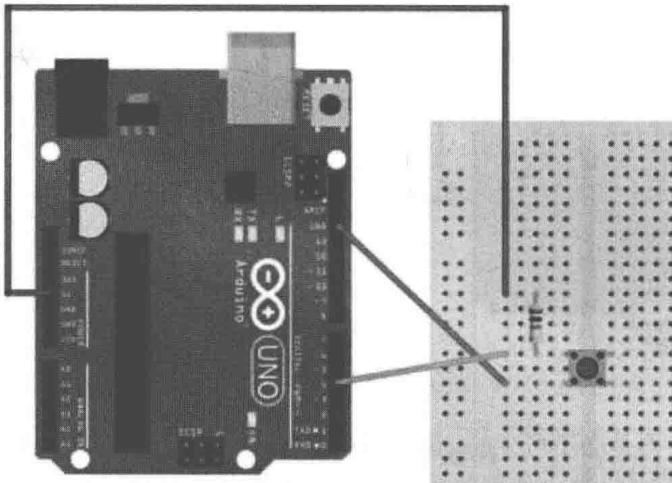


图 16-9 Arduino 与按钮开关的布线图

如果你尚未配置 PyFirmata 的话，具体配置方法请参考 16.3 节。

下面的 Python 脚本的作用是每当开关被按下的时候，它都会输出一个消息。这与程序 switch.py 非常相似。打开一个编辑器（nano 或者 IDLE）并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代

码下载区下载相应的程序。本节的示例程序名为 ardu_switch.py。你只需要访问本书链接，然后单击 Code 即可。

```
import pyfirmata
import time

board = pyfirmata.Arduino('/dev/ttyACM0')
switch_pin = board.get_pin('d:4:i')
it = pyfirmata.util.Iterator(board)
it.start()
switch_pin.enable_reporting()

while True:
    input_state = switch_pin.read()
    if input_state == False:
        print('Button Pressed')
        time.sleep(0.2)
```

当你运行上面的程序的时候，前一两秒没有任何动静，因为这时 Firmata sketch 正在启动并建立与树莓派的连接。但是，一旦它启动后，每当按钮被按下时，你就会看到一则消息。

```
$ sudo python ardu_switch.py
Button Pressed
Button Pressed
Button Pressed
```

进一步探讨

PyFirmata 利用 Iterator 的概念来监视 Arduino 的输入引脚。之所以这样做，与 Firmata 的实现密不可分。也就是说，你不能简单地按需读取 Arduino 的输入引脚的值，而是必须建立一个单独的 Iterator 线程，并通过下面的命令来管理开关的读数。

```
it = pyfirmata.util.Iterator(board)
it.start()
```

此后，你还必须使用下列命令来启用报告所关心的引脚上的信号的功能。

```
switch_pin.enable_reporting()
```

这种机制的副作用在于当你按下 Ctrl-C 退出该程序的时候，它无法正确退出。要想终止 Iterator 线程，只能打开另外一个终端窗口或 SSH 会话来终止进程（见 3.26 节），除此之外再也没有更好的方法了。

如果唯一正在运行的 Python 进程就是这个程序的话，那么你可以利用下列命令来终止它。

```
$ sudo killall python
```

如果直接断开 Arduino 与树莓派的连接的话，就会断开通信连接，也会导致该 Python 程序退出。

参考资料

本节示例与 12.1 节中将开关直接连接到树莓派的例子非常相似，并且如果你只有一个开关的话，那么类似这种使用 Arduino 的方法并不会带来任何实际益处。

16.7 利用 PyFirmata 读取 Arduino 的模拟输入

面临问题

你想在树莓派上通过 Python 读取 Arduino 的模拟输入。

解决方案

你可以使用 PyFirmata 来读取 Arduino 的模拟输入。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- 10k Ω 调谐电位器。

请连接面包板，并按照图 16-10 所示将元件连接到 Arduino 上面。

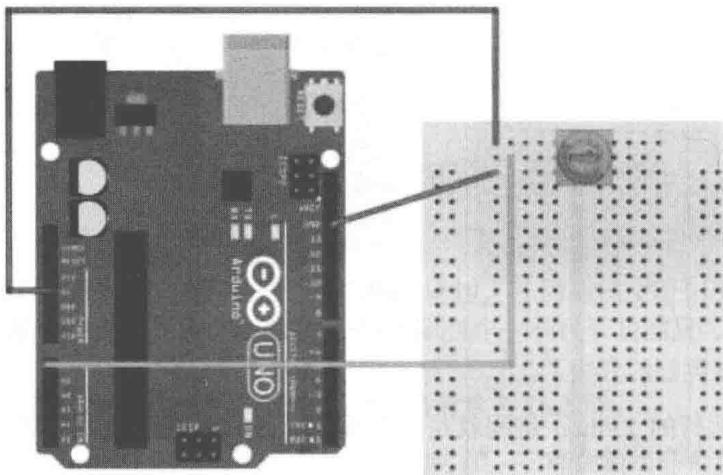


图 16-10 Arduino 与调谐电位器的布线图

如果你尚未配置 PyFirmata 的话，则具体配置方法请参考 16.3 节。

下面的 Python 脚本（`ardu_adc.py`）将显示来自模拟输入的原始读数以及模拟输入的电压值。这个程序与 `adc_test.py` 非常类似。

打开一个编辑器（`nano` 或者 `IDLE`）并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代码下载区下载相应的程序。本节的示例程序名为 `ardu_adc.py`。

```
import pyfirmata
import time

board = pyfirmata.Arduino('/dev/ttyACM0')
analog_pin = board.get_pin('a:0:i')
it = pyfirmata.util.Iterator(board)
it.start()
analog_pin.enable_reporting()

while True:
    reading = analog_pin.read()
    if reading != None:
        voltage = reading * 5.0
        print("Reading=%f\tVoltage=%f" % (reading, voltage))
    time.sleep(1)
```

模拟读数是介于 0.0 到 1.0 之间的值。

```
$ sudo python ardu_adc.py
Reading=0.000000      Voltage=0.000000
Reading=0.165200      Voltage=0.826000
Reading=0.784000      Voltage=3.920000
Reading=1.000000      Voltage=5.000000
```

进一步探讨

这个程序与 16.6 节中的非常类似。它必须使用一个 `Iterator`，所以在终止程序时也会遇到相同的问题。此外，这里的 `if` 语句也是必需的，因为如果第一次读取操作发生在实际读取模拟输入之前的话，那么读取到的就是 `None`，而非一个数字。在使用了 `if` 语句之后，这个程序就能高效地忽略所有空的读数了。

参考资料

为了使用数字输入，请参考 16.6 节。

16.8 模拟输出（PWM）与 PyFirmata

面临问题

你想利用 Arduino 的 PWM 来控制 LED 的亮度。

解决方案

你可以使用 PyFirmata 向 Arduino 发送命令，使其在某个输出上面生成一个 PWM 信号。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- 270Ω 电阻器；
- LED。

请连接面包板，并按照图 16-11 所示将元件连接到 Arduino 上面。

如果你尚未配置 PyFirmata 的话，具体配置方法请参考 16.3 节。

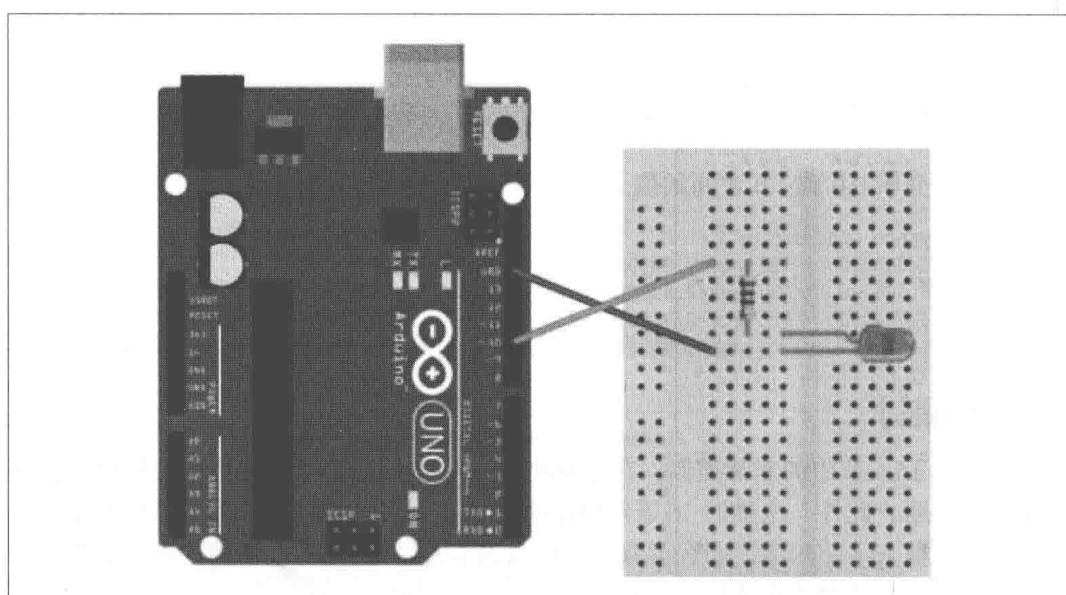


图 16-11 利用 Arduino 的 PWM 控制 LED 的布线图

下面的 Python 脚本（`ardu_pwm.py`）会先要求你输入 PWM 功率值，然后据此设置相应的 LED 亮度。打开一个编辑器（nano 或者 IDLE）并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代码下载区下载相应的程序。本节的示例程序名为 `ardu_pwm.py`。

```
import pyfirmata

board = pyfirmata.Arduino('/dev/ttyACM0')
led_pin = board.get_pin('d:10:p')
```

```
while True:  
    duty_s = raw_input("Enter Brightness (0 to 100):")  
    duty = int(duty_s)  
    led_pin.write(duty / 100.0)
```

如果输入的值是 100，那么 LED 就会达到最大亮度。当这个数字减小的时候，亮度也会随之降低。

```
$ sudo python ardu_pwm.py  
Enter Brightness (0 to 100):100  
Enter Brightness (0 to 100):50  
Enter Brightness (0 to 100):10  
Enter Brightness (0 to 100):5  
Enter Brightness (0 to 100):
```

进一步探讨

本节里的 sketch 实际上非常简单。你可以通过下列命令将其输出定义为 PWM 输出。

```
led_pin = board.get_pin('d:10:p')
```

其中，p 表示 PWM。请不要忘了，这个命令只能用于标记有~符号的 Arduino 引脚。

我们还可以修改滑块控件（见图 16-12），以便可以通过 PyFirmata 进行操作。你可以下载这个 sketch，其名称为 ardu_gui_slider。



图 16-12 用于控制亮度的用户界面

参考资料

尽管 Arduino 可以输出 40mA 电流，这大约是 GPIO 引脚电流的 10 倍，但是仍不足以直接驱动电机或大功率的 LED 模块。如果想要驱动这类模块的话，你将需要使用 11.4 节中介绍的电路，并将树莓派 GPIO 引脚改为 Arduino 输出引脚。

16.9 利用 PyFirmata 控制伺服电机

面临问题

你想利用 Arduino 控制伺服电机的转角。

解决方案

你可以使用 PyFirmata 向 Arduino 发送命令，以便生成控制伺服电机转角所需的脉冲。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- $1k\Omega$ 电阻器；
- LED。

请按照图 16-13 所示的方式连接面包板。

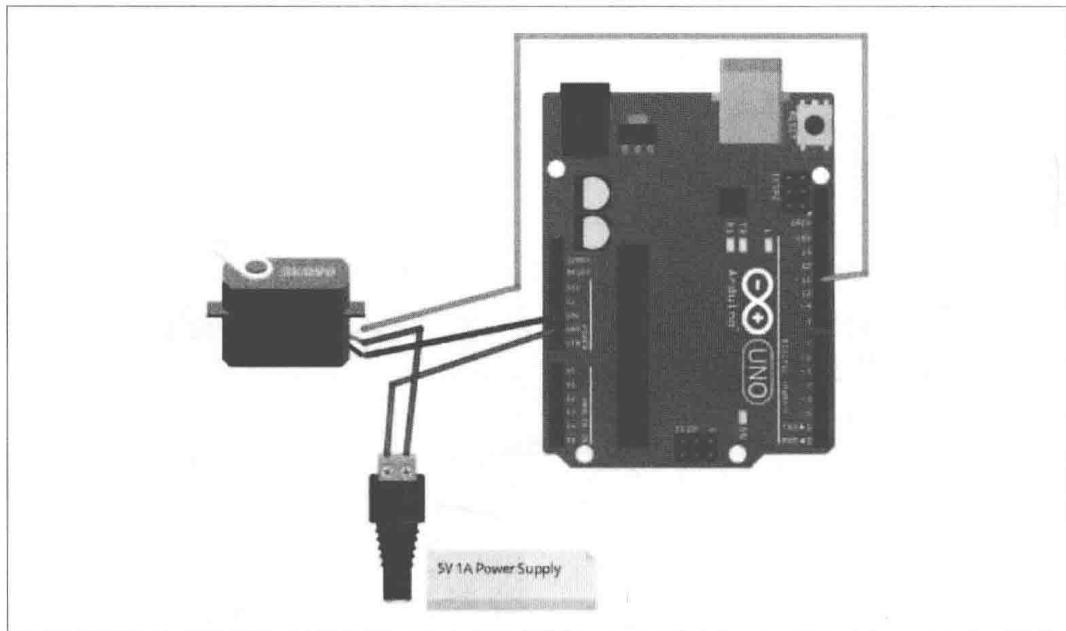


图 16-13 Arduino 控制伺服电机所需布线图

如果你尚未配置 PyFirmata，请参考 16.3 节介绍的方法进行配置。下面的 Python 脚本（`ardu_servo.py`）会要求你输入伺服电机的转角值，然后会据此设置伺服电机的机臂。

打开一个编辑器(nano 或者 IDLE)并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代码下载区下载相应的程序。本节的示例程序名为 `ardu_servo.py`。

```
import pyfirmata

board = pyfirmata.Arduino('/dev/ttyACM0')
servo_pin = board.get_pin('d:11:s')

while True:
    angle_s = raw_input("Enter Angle (0 to 180):")
```

```
angle = int(angle_s)
servo_pin.write(angle)
```

当输入的值为 0 的时候，伺服电机将位于其行程的一端；如果将这个值改为 180，它会停到另一端；当这个值为 90 的时候，它会停到中间。

```
$ sudo python ardu_servo.py
Enter Angle (0 to 180):0
Enter Angle (0 to 180):180
Enter Angle (0 to 180):90
```

进一步探讨

本节中的 sketch 实际上非常简单。你可以通过下面的命令将其输出定义为伺服电机的输出。

```
led_pin = board.get_pin('d:11:s')
```

其中，`s` 表示伺服电机。这个命令可以用于 Arduino 任意的数字引脚上面。

如果你熟悉 11.1 节的话，你就会发现，当使用 Arduino 控制伺服电机的时候，不会出现抖动现象。

参考资料

关于单纯利用树莓派控制伺服电机的解决方案，请参考 11.1 节、11.2 节和 11.3 节。

16.10 定制基于 TTL Serial 的 Arduino 通信

面临问题

你希望在不借助 PyFirmata 的情况下，利用两个设备的串行接口与 Arduino 交换数据。

解决方案

你可以使用一个电平转换器或一对电阻器将树莓派的 RXD 引脚连接至 Arduino 的 Tx 引脚，同时，将树莓派的 TXD 引脚连接至 Arduino 的 Rx 引脚。

为了进行本节中的实验，你需要：

- Arduino Uno；
- 面包板和跳线；
- 270Ω 和 470Ω 电阻器，或者四路双向电平转换器；
- $10k\Omega$ 调谐电位器。

如果你打算使用电平转换模块的话，可以按照图 16-14 所示来连接面包板。

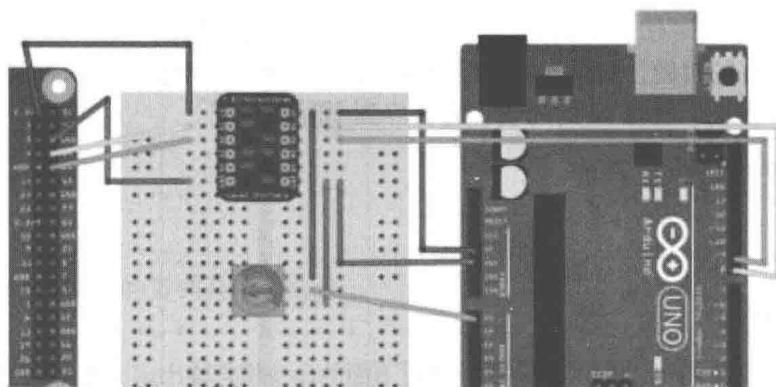


图 16-14 Arduino 串行通信布线图

反之，如果你打算使用一对电阻的话，那么可以按照图 16-15 所示方式连接面包板。

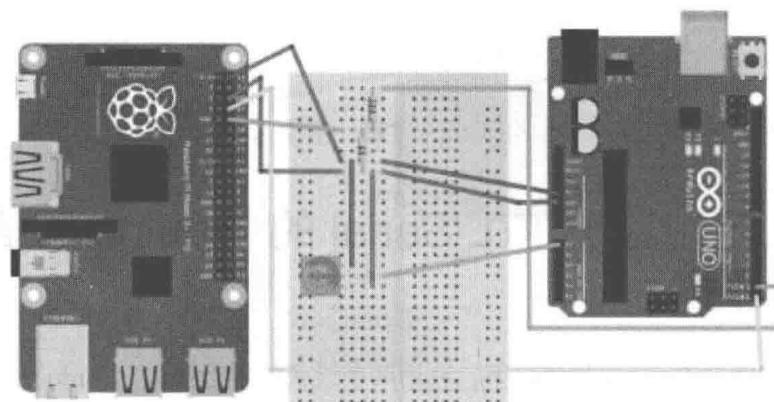


图 16-15 Arduino (使用电阻) 串行通信布线图

Arduino 的 Rx 输入可以直接使用来自树莓派的 TXD 引脚的 3.3V 输出，但是，来自 Arduino 的 Tx 引脚的 5V 电压必须降低为适用于树莓派的 3V 电压。

你需要按照 9.6 节介绍的方法来禁用串行端口登录，并安装 PySerial。

下面的 Python 脚本（`ardu_pi_serial.py`）会要求你输入命令 l 或 r。如果你输入 l 的话，那么 Arduino 内置的 LED 就会切换开关状态。

反之，如果你输入的是 r，那么 Arduino 就会读取其模拟输入 A0 上面的模拟值，并返回介于 0 到 1023 之间的一个数字作为其读数。

打开一个编辑器（nano 或者 IDLE）并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站（<http://www.raspberrypicookbook.com>），从代码下载区下载相应

的程序。本节的示例程序名为 ardu_pi_serial.py。

```
import serial

ser = serial.Serial('/dev/ttyAMA0', 9600)

while True:
    command = raw_input("Enter command: l - toggle LED, r - read A0 ")
    if command == 'l' :
        ser.write('l')
    elif command == 'r' :
        ser.write('r')
        print(ser.readline())
```

此外，你还需要上载下面的 sketch，即 ArduinoSerial，到你的 Arduino Uno。

你可以将其粘贴到一个新的 sketch 窗口中，此外，就像本书其他示例程序一样，你也可以从本书网站上面下载该程序。

```
#include "SoftwareSerial.h"

int ledPin = 13;
int analogPin = A0;

SoftwareSerial ser(8, 9); // RX, TX

boolean ledOn = false;

void setup()
{
    ser.begin(9600);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    if (ser.available())
    {
        char ch = ser.read();
        if (ch == 'l')
        {
            toggleLED();
        }
        if (ch == 'r')
        {
            sendAnalogReading();
        }
    }
}
void toggleLED()
{
    ledOn = ! ledOn;
    digitalWrite(ledPin, ledOn);
}
```

```
void sendAnalogReading()
{
    int reading = analogRead(analogPin);
    ser.println(reading);
}
```

现在，你可以运行上面的程序，并通过 l 命令来测试 Arduino 内置 LED 的开关切换情况。然后，可以试着设置调谐电位器到不同的位置，并通过发送 r 命令来取得模拟读数。

```
$ sudo python ardu_pi_serial.py
Enter command: l - toggle LED, r - read A0 l
Enter command: l - toggle LED, r - read A0 l
Enter command: l - toggle LED, r - read A0 r
0

Enter command: l - toggle LED, r - read A0 r
540

Enter command: l - toggle LED, r - read A0 r
1023
```

进一步探讨

Python 程序中的代码首先会通过 PySerial 库来打开串行端口的连接。之后，主程序的循环会不断请求命令并处理它们，就这里来说，就是你向串行端口写入的单字符命令。

就命令 r 而言，它还会从串行连接中读取一行内容，并将其输出。

需要注意的是不管看上去如何，实际上这里 serial.readline 的结果都是一个字符串。如果你想将其转换为数字的话，可以使用函数 int。举例来说：

```
line = ser.readline()
value = int(line)
```

Arduino sketch 相对来说比较复杂。这里的示例 sketch 只要稍做修改，就可以增加更多的输入或者输出来响应更多的命令。

这个 sketch 没有使用 Arduino 的硬件串口，因为它通常用于其 USB 连接。作为替代，通过一个名为 SoftwareSerial 的 Arduino 库，可以将 8 号和 9 号引脚分别用于 Rx 和 Tx。

就像所有 Arduino sketch 那样，必须要有一个 setup 函数供 Arduino 启动时调用，同时还要有一个 loop 函数供重复调用。

函数 setup 的作用是启动串行通信，并将用于内置 LED 的引脚，即 Arduino 的 13 号引脚，设置为输出。

函数 loop 首先会调用 ser.available 函数来检测是否有串行消息到达。如果有需要处理的消息，它就会读取字符，而后面的 if 子句会调用相应的辅助函数来处理命令。

参考资料

有关 SoftwareSerial 库的详细说明, 请参考 <http://arduino.cc/en/Reference/SoftwareSerial>。

除了自己编写通信代码之外, 你还可以使用 PyFirmata (见 16.3 节)。

除了通过串口通信之外, 你还可以使用 I2C 与 Arduino 进行通信 (见 16.11 节)。

16.11 通过 I2C 自定义与 Arduino 之间的通信

面临问题

你想通过两个设备的 I2C 接口实现与 Arduino 互换数据。

解决方案

I2C 总线具有主设备和从设备的概念。主设备用来控制总线, 一个主设备可以连接多个从设备, 并且每个从设备都有自己的地址。

你需要在作为 I2C 从设备的 Arduino 上面安装一个 sketch, 并通过名为 SMBus 的 Python 库来给树莓派编写一个 I2C 程序。两个设备的 SDA、SCL 和接地需要连接到一起, 同时, 除了使用单独电源给 Arduino 供电外, 还可以使用树莓派的 5V 输出进行供电。

为了进行本节中的实验, 你需要:

- Arduino Uno;
- Female-to-Female (母头转母头) 跳线。

请连接面包板, 并按照图 16-16 所示将元件连接到 Arduino 上面。

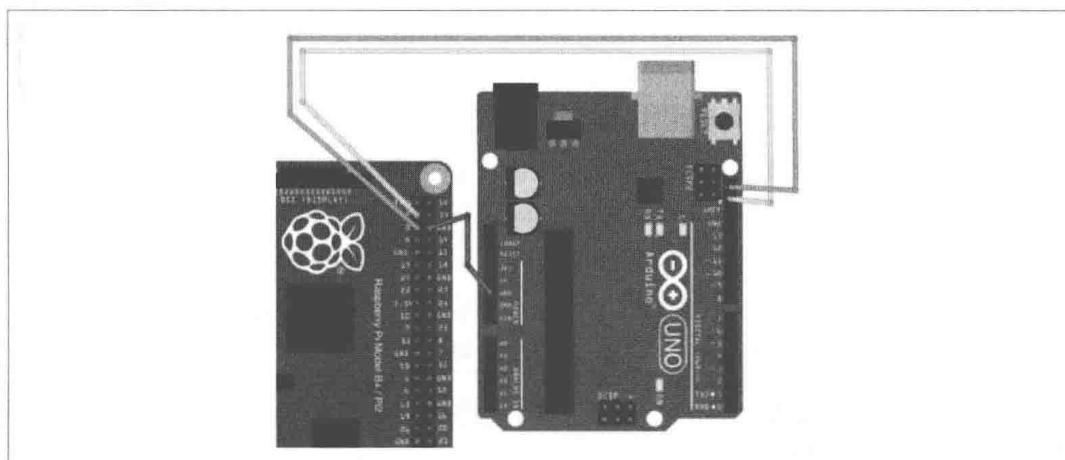


图 16-16 Arduino I2C 通信布线图

这里所用的 Arduino 是最新版的 Arduino Uno R3。如果你所用的电路板型号较旧的话，它们通常都没有提供两个专用的 SCL 和 SDA 引脚，这时候可以将树莓派上面的 SDA 和 SCL 引脚连接到 Arduino 的 A4 和 A5 引脚上面。

请将下列 sketch 安装到 Arduino 上面。你可以从本书网站 (<http://www.raspberrypicookbook.com>) 的程序下载区下载该程序，具体名称为 ArduinoI2C。

```
#include <Wire.h>

int SLAVE_ADDRESS = 0x04;
int ledPin = 13;
int analogPin = A0;

boolean ledOn = false;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Wire.begin(SLAVE_ADDRESS);
    Wire.onReceive(processMessage);
    Wire.onRequest(sendAnalogReading);
}

void loop()
{
}

void processMessage(int n)
{
    char ch = Wire.read();
    if (ch == '1')
    {
        toggleLED();
    }
}

void toggleLED()
{
    ledOn = ! ledOn;
    digitalWrite(ledPin, ledOn);
}

void sendAnalogReading()
{
    int reading = analogRead(analogPin);
    Wire.write(reading >> 2);
}
```

你需要按照 9.3 节介绍的方法来设置树莓派的 I2C 通信。

打开一个编辑器 (nano 或者 IDLE) 并粘贴下面的代码。与本书中的其他示例程序一样，你也可以访问本书网站 (<http://www.raspberrypicookbook.com>)，从代码下载区下载相应的程序。本节的示例程序名为 ardu_pi_i2c.py。

```

import smbus
import time

# for RPI revision 1, use "bus = smbus.SMBus(0)"
bus = smbus.SMBus(1)

# This must match what's in the Arduino Sketch
SLAVE_ADDRESS = 0x04

def request_reading():
    reading = int(bus.read_byte(SLAVE_ADDRESS))
    print(reading)

while True:
    command = raw_input("Enter command: l - toggle LED, r - read A0 ")
    if command == 'l':
        bus.write_byte(SLAVE_ADDRESS, ord('l'))
    elif command == 'r':
        request_reading()

```

这个测试程序与 16.10 节中用来演示串行通信的程序非常接近。它设计用来展示双向通信。当你输入命令 l 的时候，会切换 Arduino 内置 LED 的开关状态；当输入命令 r 的时候，会报告来自 Arduino 的 A0 模拟输入的读数。你可以使用一个 male-to-male 跳线将 Arduino 的 A0 连接到 3.3V、5V 或 GND，以便获得不同的读数：

```

$ sudo python ardu_pi_i2c.py
Enter command: l - toggle LED, r - read A0 l
Enter command: l - toggle LED, r - read A0 l
Enter command: l - toggle LED, r - read A0 r
184
Enter command: l - toggle LED, r - read A0

```

进一步探讨

在这种安排下，Arduino 充当了从设备的角色，所以当有多个从设备的时候，必须为其指定一个地址，以便运行在树莓派上面的主设备能够识别它。就本例而言，Arduino 的地址被设置为 0x04，并被保存到变量 SLAVE_ADDRESS 中。

然后，Arduino 上面的 setup 函数配置了两个将来用到的回调函数。当 onReceive 事件发生时，就会调用 processMessage 函数。每当树莓派发送命令时，就会触发该事件。另一个回调函数是 sendAnalogReading，它与 onRequest 事件有关。

当树莓派请求数据时就会触发该事件，该函数会读取模拟值，然后将它除以 4，以便转换成一个单字节，最后将其返回给树莓派。

与这个 sketch 对应的 Python 程序首先新建一个名为 bus 的 SMBus 实例。

为此，仅需要单个参数，即 1。这个参数表示使用树莓派上的哪个 I2C 端口。这个参数肯定是 1，除非你使用的是最早的树莓派（第一版）。这一版本的电路板的音频接口是

黑色的，而第二版上面的音频接口是浅蓝色的。如果你使用的是旧型号的树莓派的话，这个参数就取 0。在 GPIO 接口上的 I2C 端口在这两个版本之间发生了变化。

该程序会提示用户输入一个命令 (l 或 r)。如果该命令为 l，它就会把字符 l 写到 Arduino 上，从而引发 onReceive 处理程序 (处理消息) 运行。这个函数随后将调用 toggleLED。反之，如果用户输入命令 r，那么 request_reading 函数就会被调用。这将调用 SMBus 库的 read_byte，从而导致 onRequest 事件，继而引发调用 Arduino 的 sketch。

你也许奇怪，为什么在串行通信 (见 16.5 节) 时必需电平转换模块，而现在却可以将 5V 的 Arduino 直接连到树莓派的 I2C 连接上面呢？原因是 I2C 总线标准可以在连接了 SDA 和 SCL 的上拉电阻上面的任意电压下工作。在本实例中，Arduino Uno 没有将任何上拉电阻连接到 I2C 线上面。

但是，树莓派提供了上拉电阻，并将电压上拉到了 3.3V。



尽管 Arduino 在 I2C 线上面没有上拉电阻，但是这不能代表所有 I2C 设备都是如此。所以，如果 I2C 设备的工作电压是 5V，一定确保它没有上拉电阻 (不过，即使它上面有上拉电阻，只要把它们去掉即可)。

参考资料

关于串口的等效设置，请参考 16.10 节。

16.12 在树莓派上使用小型 Arduino

面临问题

你希望在树莓派上面使用一个更加紧凑的 Arduino 电路板。

解决方案

你可以使用小型的面包板友好型的 Arduino 电路板。

图 16-17 展示的是 Arduino Pro Mini 电路板。这类电路板的一个巨大优点是可以与项

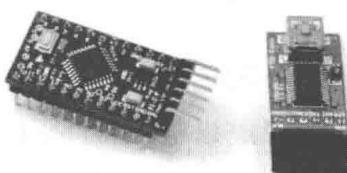


图 16-17 Arduino Pro Mini 与编程接口

目中的其他元件一起插到面包板上面。Pro Mini 电路板还有一个 3.3V 的版本，在与树莓派一起使用的时候，无需使用任何电平转换器。

进一步探讨

在这些电路板中，有一些（例如图 16-17 所示的 Pro Mini）需要使用一个 USB 编程接口。如果遇到类似电路板，你可以通过树莓派或其他电脑对其进行编程。

除了官方的 Arduino 电路板，你还可以找到许多廉价的仿制品，它们通常能够更好地兼容树莓派。

参考资料

其他可以考虑的电路板包括：

- Teensy (<http://www.pjrc.com/teensy/>).
- Arduino Micro (<http://arduino.cc/en/Main/ArduinoBoardMicro>).
- Arduino Nano (<http://arduino.cc/en/Main/ArduinoBoardNano>).

16.13 aLaMode 板与树莓派入门

面临问题

你想使用 aLaMode 板来连接树莓派和外部设备。

解决方案

图 16-18 中展示的 aLaMode 板（见附录 A 中的表 A-3）是一种强大的接口板，从本质上来说，它实际上就是 Arduino Uno，只不过在电路板的一个角上提供了树莓派 GPIO 接口而已。

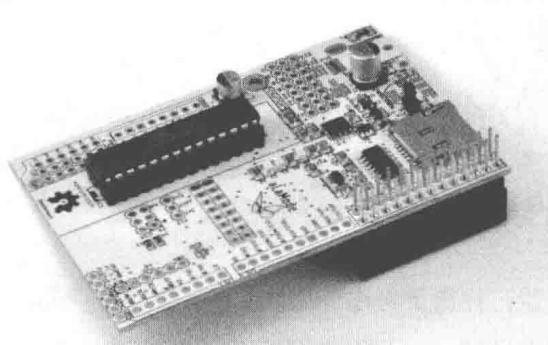


图 16-18 aLaMode 接口板

aLaMode 与树莓派几乎可以完美相配，有了它，你无需额外导线就能够使用 ArduinoShield。虽然这个电路板是设计用于原始版本的树莓派的，但是它仍然可以用于树莓派 2。

需要注意的是图 16-18 中所示的电路板并没有提供连接 ArduinoShield 所需的插头。

在 16.1 节中介绍的方法同时也为你的 Arduino 环境配置了 aLaMode。在 Arduino IDE 的 Tools 菜单下面，选择 Board，然后选择 aLaMode（见图 16-19）。



图 16-19 Arduino IDE 中的 aLaMode 板类型

在对 aLaMode 进行编程之前，你还需要将串口设置为/devttyS0，因为编程是通过串口进行连接的。

进一步探讨

本章所有与 Firmata 有关的示例都可以与 aLaMode 一起工作，其中包括 16.10 节中自定义的串行通信示例。唯一需要修改的就是 Python 程序与 aLaMode 通信的时候，必须使用串口，而不是 USB 接口。ALaMode 是通过内置电平转换硬连接到 Arduino 的 Tx 和 Rx 引脚上面的。

在使用 Firmata 的时候，你需要将端口从/dev/ttyACM0 改为/dev/ttyAMA0。

```
board = pyfirmata.Arduino('/dev/ttyAMA0')
```

类似地，如果你打算编写自己定义的代码，那么你需要将打开串行连接的代码修改成

下面的样子。

```
ser = serial.Serial('/dev/ttyAMA0', 9600)
```

为了试用 aLaMode，只需将 Standard Firmata 这个 sketch 上载至 aLaMode，然后运行 Python 程序 ardu_lash_ser.py（见 16.5 节）即可。

aLaMode 有许多有趣的特性，其中包括以下几个。

- 它既可以通过树莓派 GPIO 接口上的 5V 引脚供电，也可以使用一个单独的 5V 电压通过 micro-USB 接口来供电。
- RTC 直接连接到树莓派，而非该电路板的 Arduino 端。
- 可以附加 ArduinoShield，并且提供高度的兼容性（见 16.14 节）。
- 它可以用作 I2C 从设备。除了进行了串行连接外，aLaMode 的 I2C 连接也连接到了树莓派（见 16.11 节）。
- 它提供了一个迷你 SD 读卡器。
- 它提供了直接连接伺服电机的接头（见 16.9 节）。



aLaMode 在设计上面存在一个小缺陷——Arduino shield 接头是焊接上的，模拟输入 A0 到 A5 非常容易触到树莓派上的 RJ45 以太网接口的裸露金属。为了避免这些问题，在附加 aLaMode 之前，可以在树莓派的以太网接口上面缠上几层绝缘胶带（见图 16-20）。

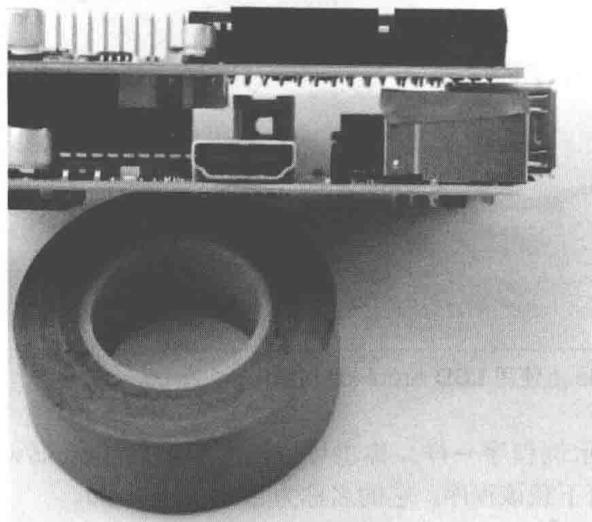


图 16-20 为树莓派 RJ45 接口进行绝缘处理

参考资料

aLaMode 的官方使用指南, 请访问 <http://bit.ly/1d2YMxh>。

16.14 在树莓派和 aLaMode 板上使用 Arduino Shield

面临问题

你想在树莓派上面使用 Arduino shield。

解决方案

你可以使用 aLaMode 接口板。你可能会发现, 该接口板需要一个 Arduino 样式的接头, 才能使 Arduino shield 焊接到适当的位置上面。

作为一个示例, 下面将会把一个 Arduino LCD shield 连接到 aLaMode 上面(见图 16-21)。

为了进行本节中的试验, 你需要:

- aLaMode 板;
- Freetronic (或其他) Arduino LCD shield。

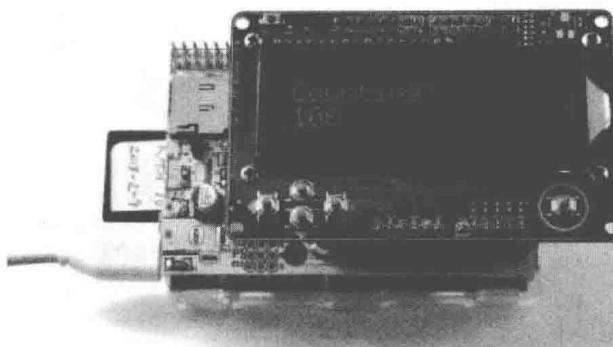


图 16-21 在 aLaMode 上使用 LCD Arduino shield

就像本书中的所有示例程序一样, 你也可以从本书网站 (<http://www.raspberrypicookbook.com>) 的代码区下载该程序, 它的名称为 AlaModeShield。

```
#include <LiquidCrystal.h>  
  
// pins for Freetronics LCD Shield
```

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup()
{
    lcd.begin(16, 2);
    lcd.print("Counting!");
}

void loop()
{
    lcd.setCursor(0, 1);
    lcd.print(millis() / s1000);
}
```

你只要将该 sketch 上载至 aLaMode，你就能看到消息“Counting”出现在显示器的第一行中，而最后一行则是统计的秒数。

进一步探讨

这个示例使用的是 Freetronics LCD shield。此外，还有许多其他的 LCD 显示器 shield，并且某些还具有不同的引脚布局。如果你使用的是跟这里不同的模块，则需要修改下列一行代码。

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

这些参数表示的是 Arduino 的引脚，它们依次用于下列 LCD 连接。

Rs、enable、d4、d5、d6 和 d7。

参考资料

关于 Arduino LiquidCrystal 库的参考材料，请访问 <http://arduino.cc/en/Reference/LiquidCrystal>。

附录 A

配件与供应商

配件

你可以通过下面的表格寻找本书所用到的配件。如果有可能的话，我会尽量列出供应商的产品代码。

目前，有许多电子元件供应商为硬件制造达人和业余电子爱好者提供相应服务。表 A-1 列出了其中最流行的一些供应商。

表 A-1 配件供应商

供应商	网站	备注
Adafruit	http://www.adafruit.com	提供优质模块
Digikey	http://www.digikey.com/	提供各种元件
MakerShed	http://www.makershed.com/	提供优质模块、套件和工具
MCM Electronics	http://www.mcmelectronics.com/	提供各种元件
Mouser	http://www.mouser.com	提供各种元件
SeeedStudio	http://www.seeedstudio.com/	提供各种有趣的廉价模块
SparkFun	http://www.sparkfun.com	提供优质模块
MonkMakes	http://www.monkmakes.com	提供树莓派的各种套件等
Pimoroni	https://shop.pimoroni.com	各种有趣 HAT 的英国制造商和零售商
Polulu	https://www.pololu.com/	提供优质电机控制器和机器人
CPC	http://cpc.farnell.com/	英国供应商；提供各种元件
Ciseco	http://shop.ciseco.co.uk/	PiLite、Humble Pi 等产品的供应商

续表

供应商	网站	备注
Farnell	http://www.farnell.com/	国际供应商；提供各种元件
Maplin	http://www.maplin.co.uk/	英国供应商；提供砖和砂浆部件，以及树莓派商店
Proto-pic	http://proto-pic.co.uk/	英国供应商；提供 SparkFun 和 Adafruit 模块

除此之外，eBay 也是一个非常棒的元件来源。

寻找元件是一件既费时又费劲的事情，为此，你可以借助于元件搜索引擎 Octopart (<http://www.octopart.com>) 来寻找各种配件。此外，MonkMakes、Adafruit 和 SparkFun 也都为初学者提供了各种元件包。

成型设备

本书中的许多硬件项目都使用了各种不同的跳线。其中，male-to-female 跳线（将树莓派的 GPIO 接口连接到面包板）和 male-to-Male 跳线（在面包板上面进行连接）最为常用。

在将模块直接连接到 GPIO 引脚的时候，偶尔会用到 Female-to-Female（母头转母头）跳线。你很少用到长度超过 3 英寸（75mm）的跳线。表 A-2 列出了一些跳线和面包板的规格及其相应的供应商。

实际上，要想入手面包板、跳线和某些基本元件的话，最简便的方法就是直接从 MonkMakes.com (<http://monkmakes.com>) 上面购买类似于 the Electronics Starter Kit for Raspberry Pi 这样的入门套件。

表 A-2 成型设备

说明	供应商
M-M 跳线	SparkFun: PRT-08431; Adafruit: 759
M-F 跳线	SparkFun: PRT-09140; Adafruit: 825
F-F 跳线	SparkFun: PRT-08430; Adafruit: 794
半尺寸面包板	SparkFun: PRT-09567; Adafruit: 64
树莓派排线	Adafruit: 1105
Raspberry Leaf (26 pin)	Adafruit: 1772
Raspberry Leaf (40 pin)	Adafruit: 2196

续表

说明	供应商
Electronics Starter Kit for Raspberry Pi	Amazon; monkmakes.com
Adafruit PermaProto for Pi (半个面包板)	Adafruit: 1148
Adafruit PermaProto for Pi (整个面包板)	Adafruit: 1135
Adafruit PermaProto HAT	Adafruit: 2314
DC 插孔到螺丝端子的适配器 (母头)	Adafruit: 368

电阻与电容

表 A-3 展示了本书用到的一些电阻和电容以及这些元件的部分供应商。

表 A-3 电阻与电容

配件	供应商
270Ω 0.25W 电阻器	Mouser: 293-270-RC
470Ω 0.25W 电阻器	Mouser: 293-470-RC
1kΩ 0.25W 电阻器	Mouser: 293-1k-RC
3.3kΩ 0.25W 电阻器	Mouser: 293-3.3k-RC
4.7kΩ 0.25W 电阻器	Mouser: 293-4.7k-RC
10 kΩ 调谐电位器	Adafruit: 356; SparkFun: COM-09806; Mouser: 652-3362F-1-103LF
光敏电阻器	Adafruit: 161; SparkFun: SEN-09088
330nF 电容器	Mouser: 80-C330C334K5R
Thermistor T0 of 1k Beta 3800 NTC	Mouser: 871-B57164K102J (注意: Beta 为 3730)

晶体管与二极管

表 A-4 列出了本书涉及的一些晶体管和二极管，以及这些元件的部分供应商。

表 A-4 晶体管与二极管

配件	供应商
FQP30N06L N-通道逻辑级 MOSFET 晶体管	Mouser: 512-FQP30N06L; Sparkfun: COM-10213
2N3904 NPN 双极晶体管	SparkFun: COM-00521; Adafruit: 756

续表

配件	供应商
1N4001 二极管	Mouser: 512-1N4001; SparkFun: COM-08589; Adafruit: 755
TIP120 达林顿晶体管	Adafruit: 976; CPC: SC10999
2N7000 MOSFET 晶体管	Mouser: 512-2N7000; CPC: SC06951

集成电路

表 A-5 列出了本书中用到的集成电路及其部分供应商。

表 A-5 集成电路

配件	供应商
7805 稳压器	SparkFun: COM-00107; Adafruit: 2164; Mouser: 511-L7805CV, CPC: SC10586
L293D 电机驱动	SparkFun: COM-00315; Adafruit: 807; Mouser: 511-L293D; CPC: SC10241
ULN2803 Darlington 驱动 IC	SparkFun: COM-00312; Adafruit: 970; Mouser: 511-ULN2803A; CPC: SC08607
DS18B20 温度传感器	SparkFun: SEN-00245; Adafruit: 374; Mouser: 700-DS18B20; CPC: SC10426
MCP3008 八通道 ADC IC	Adafruit: 856; Mouser: 579-MCP3008-I/P; CPC: SC12789
TMP36 温度传感器	SparkFun: SEN-10988; Adafruit: 165; Mouser: 584-TMP36GT9Z; CPC: SC10437

光电器件

表 A-6 列出了本书所用的各种光电器件及其部分供应商。

表 A-6 光电器件

配件	供应商
5mm 红色 LED	SparkFun: COM-09590; Adafruit: 299
RGB 共阴极 LED	SparkFun: COM-11120; eBay
TSOP38238 IR 传感器	SparkFun: SEN-10266; Adafruit: 157

模块

表 A-7 列出了本书用到的各种模块及其部分供应商。

表 A-7

模块

配件	供应商
树莓派摄像头模块	Adafruit: 1367; MCM: 28-17733; CPC: SC13023
Arduino Uno	SparkFun: DEV-11021; Adafruit: 50; CPC: A000066
电平转换器, 四通道	SparkFun: BOB-11978; Adafruit: 757
电平转换器, 八通道	Adafruit: 395
锂电池升压转换器/充电器	SparkFun: PRT-11231
PowerSwitch tail	Adafruit: 268
16 通道伺服控制器	Adafruit: 815
电机驱动 1A dual	SparkFun: ROB-09457
RasPiRobot 板 V3	Adafruit: 1940; Amazon
MonkMakes RasPiRobot Rover Kit	Amazon
Pi Plate	Adafruit: 801
PIR 运动侦测器	Adafruit: 189
Venus GPS 模块	SparkFun: GPS-11058
甲烷传感器	SparkFun: SEN-09404
气体传感器分线板	SparkFun: BOB-08891
ADXL335 三轴加速度计	Adafruit: 163
带 I2C backpack 的 4x7 段	LEDAdafruit: 878
带 I2C backpack 的双色 LED 方形像素矩阵	Adafruit: 902
aLaMode 接口板	Makershed: MKWY1; Seeedstudio: ARD10251P
Freetronics Arduino LCD shield	www.freetronics.com
RTC 模块	Adafruit: 264
16x2 HD44780 兼容 LCD 模块	SparkFun: LCD-00255; Adafruit: 181
配件	供应商
Sense HAT	Adafruit: 2738
Adafruit Capacitave Touch HAT	Adafruit: 2340

续表

配件	供应商
Stepper Motor HAT	Adafruit: 2348
16 Channel PWM HAT	Adafruit: 2327
Pimoroni Explorer HAT Pro	pimoroni.com; Adafruit: 2427
Squid Button	monkmakes.com; Amazon
Raspberry Squid RGB LED	monkmakes.com; Amazon
I2C OLED display 128x64 pixels	eBay

杂项

表 A-8 列出了本书用到的一些杂项工具和元件及其部分供应商。

表 A-8 杂项

配件	供应商
1200mAh 锂电池	Adafruit: 258
5V 继电器	SparkFun: COM-00100
5V 电压表	SparkFun: TOL-10285
标准伺服电机	SparkFun: ROB-09065; Adafruit: 1449
9g 迷你伺服电机	Adafruit: 169
5V 1A 电源	Adafruit: 276
低功率 6V DC 电机	Adafruit: 711
0.1 英寸插头引脚	SparkFun: PRT-00116; Adafruit: 392
5V 5 引脚单机步进电机	Adafruit: 858
12V 4 引脚双极步进电机	Adafruit: 324
Magician chassis 与减速机	SparkFun: ROB-10825
6 × AA 电池仓	Adafruit: 248
轻触按键开关	SparkFun: COM-00097; Adafruit: 504
Miniature 滑动开关	SparkFun: COM-09609; Adafruit: 805
旋转编码器	Adafruit: 377
4 × 3 数字键盘	SparkFun: COM-08653
Piezo 蜂鸣器	SparkFun: COM-07950; Adafruit: 160
簧片开关	Adafruit: 375

附录 B

树莓派引脚

树莓派 3/2 B、B+、A+、Zero

图 B-1 展示了当前 40 引脚树莓派的 GPIO 接口的各个引脚。

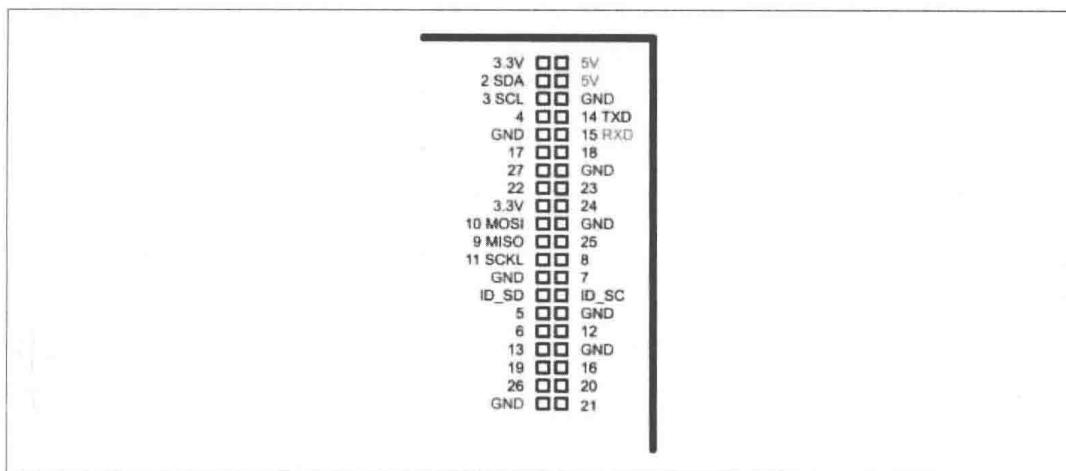


图 B-1 40 引脚树莓派的引脚

B 型树莓派第二版与 A 型树莓派

如果你已经拥有了一个树莓派的话，那么它很可能是 B 型第二版的，具体如图 B-2 所示。

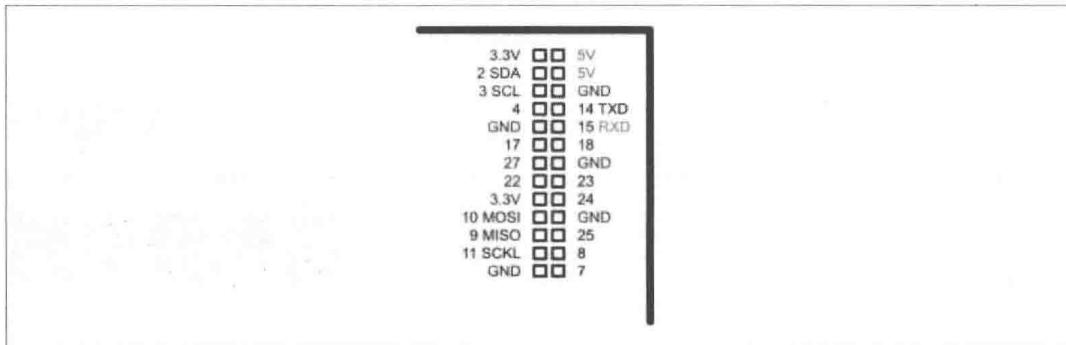


图 B-2 B 型树莓派第二版与 A 型树莓派的 GPIO 引脚

B 型树莓派第一版

B 型树莓派的最初版本（第一版）与随后的第二版相比，提供的引脚要少一些。它是唯一与后续版本的引脚不兼容的版本，在图 B-3 中，我们用粗体显示了这些不兼容的引脚。

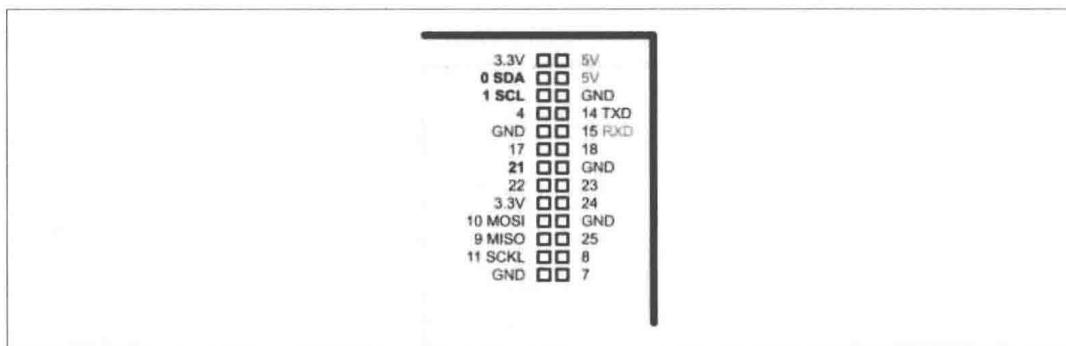


图 B-3 B 型树莓派第一版的 GPIO 引脚

作者简介

Simon Monk 博士（英国普雷斯顿）拥有控制论和计算机科学本科学位以及软件工程博士学位。Simon 在回归工业界之前，曾经从事多年的学术研究工作，并与人合作创立了移动软件公司 Momote Ltd。

目前，Simon 是一位职业作家，撰写包括树莓派和 Arduino 在内的各种开源硬件方面的作品，以及多部通用电子方面的图书。

若想获得 Simon 著作方面的更多信息，请访问 <http://www.simonmonk.org> 网站，或者在 Twitter 上面关注 @simonmonk2。

封面简介

本书封面上面的动物为欧亚雀鹰（学名：Accipiter nisus），又称为北部雀鹰，或简称雀鹰。这种小型猛禽遍布于整个东半球。成年雄鸟上体羽毛呈暗灰色，下体有红褐色横斑。雌鸟及幼鸟通体呈棕色，并且下体有棕色条纹。雌鸟个体要比雄鸟大 25%。

雀鹰专门捕食林地鸟，但是人们发现，有时它也会追捕城镇中的庭院鸟类。雄性雀鹰喜欢捕猎小型鸟类，例如山雀、雀和麻雀；雌性雀鹰则喜欢捕食鸠鸽类和鹑鸡类，并且能够猎杀体重达到或超过 18 盎司（500 克）以上的鸟类。

欧亚雀鹰的繁育鸟巢长达 60 厘米（约 2 英尺），通常由树枝筑成。筑巢之后，雌鸟会产下 4 枚或 5 枚淡蓝色且带有棕色条纹的蛋。孵化成功与否取决于雌鸟是否能够维持一个较大的体重，雄鸟会为它的伴侣提供食物。孵化 33 天之后，雏鸟就会破壳而出，再过 24 到 28 天，它们就会长出羽毛。

幼鸟在第一年的成活率只有 34%。之后，它们的成活率会提高两倍，成年雀鹰的年度成活率为 69%。这些鸟类通常的寿命为 4 年。在死亡率方面，雄性幼鸟的死亡率一般高于雌性幼鸟的死亡率。尽管在第二次世界大战之后，雀鹰的数量急剧减少，但是它们仍然是欧洲大陆上最常见的猛禽。在播种前对种子施用有机氯杀虫剂能够导致雀鹰致残，甚至死亡。这些化学药物残留还会导致雀鹰产下易碎的鸟蛋，从而导致它们在孵化过程中被压碎。在禁用这些化学制品之后，雀鹰的数量已经有所恢复，因此被 BirdLife International 评价为无生存危机的物种。

封面图片来自 Cassell 的 Natural History。

树莓派开发实战 (第2版)

随着数以百万计的新用户和多种新型号的到来，树莓派生态系统正在继续扩张，随之而来的是与树莓派性能有关的大量新问题。作为一本畅销的实战类图书的第2版，我们提供了240多个动手实验，与你一起在这种便宜的微型计算机上运行Linux，通过Python对树莓派进行编程，为树莓派连接传感器、电机和其他硬件（包括Arduino和物联网）。

作为一名富于创造力的黑客和图书作者，Simon Monk还会向你传授相应的基本原理，以便在树莓派的生态系统发展出各种新技术的时候，你能够更好地运用它们。本书是程序开发人员和业余爱好者通过各种资源（包括*Getting Started with Raspberry Pi*, O'Reilly出版）来精通树莓派的理想之选。本书相关的Python及其他语言的示例代码均可以通过GitHub下载。

- 配置树莓派并连接网络。
- 使用基于Linux的操作系统。
- 使用Python进行树莓派编程。
- 使用机器视觉让树莓派“睁眼看世界”。
- 通过GPIO接口控制硬件。
- 通过树莓派控制不同类型的电机。
- 使用开关、键盘及其他数字输入。
- 通过传感器测量温度、亮度和距离。
- 利用各种方式连接物联网。
- 借助Arduino创建动态项目。

HARDWARE / ELECTRONICS / RASPBERRY PI

封面设计：Karen Montgomery 张健

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

分类建议：计算机/开源硬件

人民邮电出版社网址：www.ptpress.com.cn

Simon Monk博士拥有控制论和计算机科学的学位以及软件工程博士学位，同时还是移动软件公司Momote公司的联合创始人。目前，Simon是一位职业作家，已经撰写了多部关于电子学和开源硬件的图书，其中包括*Making Android Accessories with IOIO* (O'Reilly出版) 和 *30 Arduino Projects for the Evil Genius* (McGraw-Hill出版)。



人民邮电出版社

www.epubit.com.cn



异步社区 www.epubit.com.cn

新浪微博 @人邮异步社区

投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-44720-3



ISBN 978-7-115-44720-3

定价：89.00 元

[General Information]

书名=树莓派开发实战 第2版

作者=(英) Simon

页数=417

SS号=14187277

DX号=

出版日期=2017.03

出版社=北京人民邮电出版社