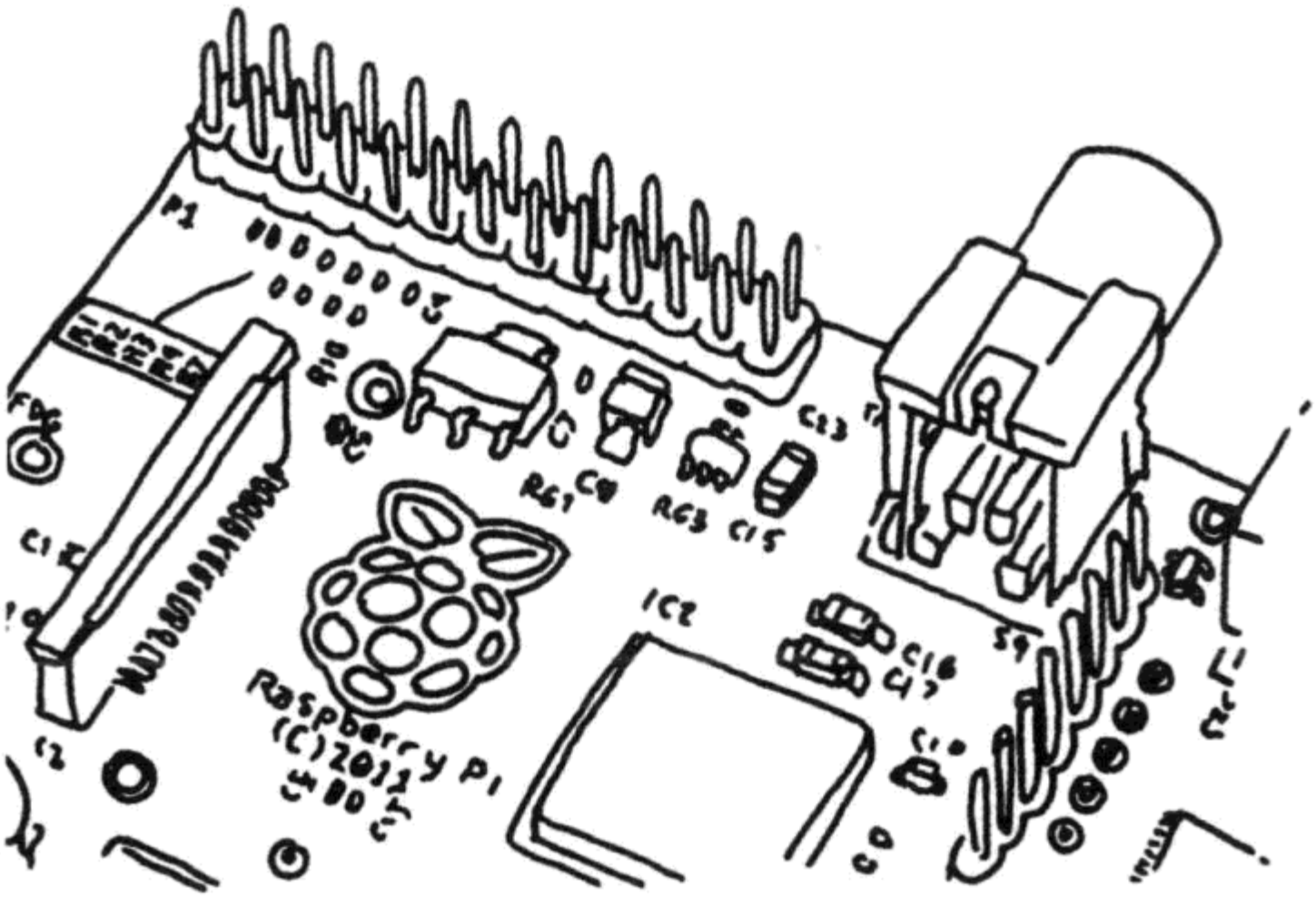
第 10 章

Python与Internet

Python and the Internet



Python 拥有一个非常活跃的开发社区，很多人乐意把他们的作品以开源库的形式贡献到社区中，这让使用 Python 完成各种任务变得更为容易。其中的一些库可以帮助我们轻松地在程序中实现连接 Internet 并从网上获取天气信息、发送电子邮件或文本信息、从 Twitter 获取热门话题或搭建一个 Web 服务器。

在本章中，我们会学习用 Raspberry Pi 接入网络并完成一些网络相关的应用开发。首先，我们来看看如何从 Internet 上获取信息，然后再尝试把我们的 Raspberry Pi 变成一个 Web 服务器。

从 Web 服务器下载数据

当你在网络浏览器中输入网址并按下回车键后，你的浏览器就充当了一个客户端的角色，它向服务器发出连接请求，并建立一个连接，然后服务器会返回一个 Web 页面。当然，在网络环境中，客户端不仅可以是一个浏览器，它也可以是一个电子邮件程序，或者是你的电脑或手机上的一个天气预报组件，又或者是一个向网络上传你的最高分纪录的游戏程序。在本章的前半部分，我们会专注于把 Raspberry Pi 当作一个客户端来使用。你所编写的代码，会向服务器发起连接，并获取信息。在开始编写程序前，你需要先安装一个常用的 Python 库——Requests，Requests 库提供了通过 HTTP（超文本传输协议，Hypertext Transfer Protocol）向 Web 服务器发送请求的功能。在命令行上输入下面的命令安装这个库：

```
pi@raspberrypi ~ $ sudo apt-get install python-requests
```

确认是否正确安装了 Request 库：

```
pi@raspberrypi ~ $ python
Python 2.7.3rc2 (default, May 6 2012, 20:02:25)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import requests
>>>
```

如果没有出错提示，就意味着 Requests 库已经被正确地安装，并被导入到当前的 Python 会话，你就可以尝试使用它了：

```
>>> r = requests.get('http://www.google.com/')
>>>
```

也许你会有点失望，因为看上去什么也没有发生。不过事实上并非如此，这个请求获得的所有数据都已经保存在了 r 对象中。你可以用下面的方法来显示这个请求的返回码：

```
>>> r.status_code
200
```

HTTP 返回码 200 表示这个请求已经成功完成。表 10.1 中列出了其他常用的 HTTP 返回码。

表10.1 常见HTTP返回码

返回码	含 义	返回码	含 义
200	正常	401	没有权限
301	永久跳转	404	文件不存在
307	临时跳转	500	服务器错误

如果你想查看服务器返回的内容，用这个方法：

```
>>> r.text
```

如果一切正常，这个命令会显示出一串很长的文本。这些文本中有些是可读的文字，有些则是理解起来有点困难的符号。这是Google 首页的 HTML 代码，原本是应该由浏览器进行解析并渲染成最终网页展现给你的。

不过，也不是所有的 HTTP 请求都会被设计成由浏览器来渲染成网页。有时候会仅仅使用 HTTP 来传递数据本身，而不包含指定这些数据展现方式的信息。有很多网站向公众提供这样的数据协议，所以我们可以不通过浏览器，直接与这些网站交换数据。无论这个数据协议是公开的还是私有的，数据协议的规范常常被称为 API（Application Programming Interface，应用程序编程接口）。通过使用 API，可以在一个软件与另一个软件之间进行数据交换，在 Internet 的网站之间交换数据也是它的一个常见应用。

我们可以开发这样一个程序：如果天气预报有雨，则当你离开家门口时，程序会提醒你带上雨伞。当然，这不需要你建立一个气象台来预测天气是否会下雨，而是可以从很多现成的 API 获得当天的天气预报。

获取天气预报

为了可以获知今天会不会下雨，我们将使用 Weather Underground（*http://www.wunderground.com/*）的 API 获取天气信息。

不同 API 的使用方法不尽相同，你需要通过查看它们的文档来决定某个 API 是否可以满足你的需求。另外，大部分 API 都会有在一段时间内的调用次数限制，哪怕

是那些需要付费使用的。很多 API 提供者会提供免费试用，允许你每天免费少量调用这个 API，这些免费的 API 非常适合用于实验或个人日常使用。

1. 在浏览器中，打开 Weather Underground 的 API 主页（*http://www.wunderground.com/weather/api/*），输入你的信息注册一个账号。

2. 用你的账号登录后，打开 Key Settings 页面。

3. 这个页面提供了一个很长的字符串，这个字符串就是属于你的 API 密钥（图 10.1）。在你每次向 API 发起请求时，都需要带上这个 API 密钥。如果你滥用网站的服务，向他们的 API 发起了过多的请求，他们就可以把这个 API 密钥放入黑名单，拒绝你的后续请求，直到你付费购买他们的服务。

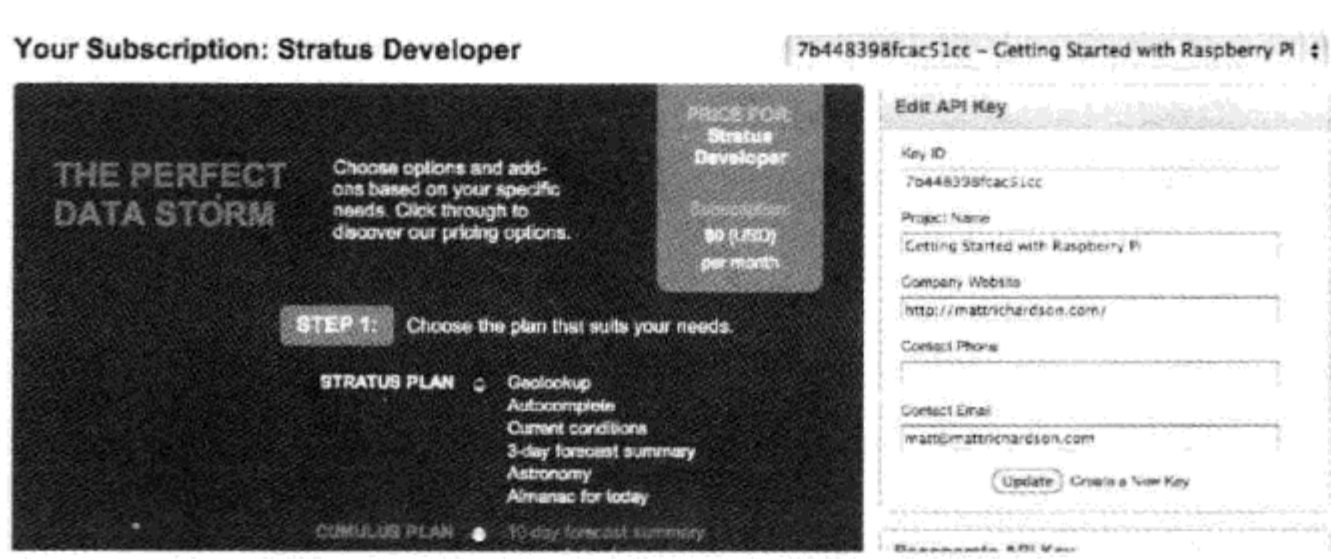


图10.1 本书作者Matt的Weather Underground API账号，右上角显示的是他的API密钥

4. 打开 Documentation 中的 Forecast 链接，你可以看到天气预报请求返回的数据内容。在这个页面底部，还给出一个用于获取旧金山（San Francisco）天气预报的请求 URL。注意，需要把你的 API 密钥填入这个 URL：

```
http://api.wunderground.com/api/YourAPIkey/forecast/q/CA/
San_Francisco.json
```

5. 你可以把这个 URL 输入到浏览器的地址栏来验证它是否可以正常工作，浏览器会以 JSON（JavaScript Object Notation，JavaScript 对象表示法）（代码 10.1）格式返回天气预报信息。请注意数据的层次结构。

尽管 JSON 的首字母 J 是 JavaScript 的缩写，JSON 其实在很多语言中都可以使用，多用于在应用程序之间通过 API 传递数据的场合。

6. 下面可以尝试一下用这个 API 来获取本地的天气预报了。修改上面的 URL，填入你所在的省和市，然后把代码写入一个新的 Python 脚本，命名为 *text-forcast.py*（代码 10.2）。

7. 正如你在代码中所看到的，如果要获取当天的天气预报文本，我们需要从返回的结果中找出正确的数据（代码 10.1）。文本格式的天气预报信息可以从 forecast → txt_forecast → forecastday → 0（表示第一个结果，也就是当天的结果）→ fcttext 节点获得。

8. 在命令行上运行脚本，输出结果应该就是你所在地当天的天气情况，如“晴。最高温度 47℉。东北风，风速 5 ~ 10 mph”。

代码 10.1　从 Weather Underground API 返回的部分 JSON 信息

```
"response": {
  "version": "0.1",
  "termsofService":
   "http://www.wunderground.com/weather/api/d/terms.html",
  "features": {
    "forecast": 1
  }
```

```
},
"forecast":{❶
  "txt_forecast": {❷
    "date":"10:00 AM EST",
    "forecastday": [❸
    {
      "period":0,
      "icon":"partlycloudy",

     "icon_url":"http://icons-ak.wxug.com/i/c/k/partlycloudy.
     gif",
      "title":"Tuesday",
      "fcttext":
        "Partly cloudy. High of 48F. Winds from the NNE at
         5 to 10 mph.",❹
      "fcttext_metric":
        "Partly cloudy. High of 9C. Winds from the NNE at
         10 to 15 km/h.",
      "pop":"0"
    },
```

❶ forecast 是我们需要解释的最顶层数据。

❷ 在 forecast 节点中，我们需要找到文本预报信息。

❸ 在文本预报信息节点中，我们需要当天的预报。

❹ fcttext 是当天天气预报的文本。

代码 10.2 *text-forecast.py* 的源代码

```
import requests

key = "YOUR KEY HERE"❶
ApiUrl = \
    "http://api.wunderground.com/api/" + key + "/forecast/q/
NY/New_York.json"

r = requests.get(ApiUrl)❷
forecast = r.json❸
print forecast["forecast"]["txt_forecast"]["forecastday"][0]
["fcttext"]❹
```

❶ 把这里的字符串替换成你的 API 密钥。

❷ 从 Weather Underground 获取纽约的天气预报（把这里的地址换为你自己所在的省和市）。

❸ 获取返回的 JSON 值并把它解析成一个 Python 的字典对象。

❹ 从这个字典对象中逐级获取到当天的天气预报文本。

现在，你已经完成了一个 Python 脚本，使用这个脚本可以随时从网上获取天气预报。但是，如何能让 Raspberry Pi 知道今天到底会不会下雨呢？虽然我们可以直接从天气预报的文本中寻找“rain”（下雨）、“drizzle”（小雨）、“thunderstorm”（暴风雨）、“showers”（阵雨）等关键词，但我们还有一个更好的办法：Weather Underground API 返回的结果中，有一个字段名为 pop，指的是降水概率。pop 字段的值为 0 ~ 100%，表示了下雨、下雪的可能性。

在我们的例子中，假设当降水概率超过 30% 时，就让 Raspberry Pi 提醒我们带上雨伞。

1. 把一个 LED 连接到 GPIO 25 接口（图 7.4）。

2. 创建一个名为 *umbrella-indicator.py* 的文件，输入代码 10.3 中的代码。别忘了把程序中的 Weather Underground API 的 URL 中的 API 密钥替换成你自己的。

3. 以 root 权限运行这个脚本：`sudo python umbrella-indicator.py`。

代码 10.3　*umbrella–indicator.py* 的源代码

```
import requests
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
```

```
key = "YOUR KEY HERE"❶
ApiUrl =\
    "http://api.wunderground.com/api/" + key + "/forecast/
q/NY/New_York.json"


while True:
    r = requests.get(ApiUrl)
    forecast = r.json
    popValue = forecast["forecast"]["txt_forecast"]
["forecastday"][0]["pop"]❷
    popValue = int(popValue)❸

    if popValue >= 30:❹
        GPIO.output(25, GPIO.HIGH)
    else:❺
        GPIO.output(25, GPIO.LOW)

    time.sleep(180) # 3 minutes❻
```

❶ 与以前一样，把这里的 API 密钥换成你自己的。

❷ 获取今天的降水概率并存入 popValue 变量中。

❸ 把 popValue 由字符串转换为数字，以便后续可以对它按数值来处理。

❹ 如果降水概率值大于 30，则点亮 LED。

❺ 否则，熄灭 LED。

❻ 等待 3min 后再次获取天气信息。所以，这个脚本一天对 API 的请求会小于 500 次的限制值。

实验完成后，按 Control-C 键结束程序运行。

除了 Weather Underground 以外，网上还有很多其他 API 可以使用。表 10.2 中列出了其他一些提供 API 的网站。

表10.2　常见网站的API

网　站	API 参考手册的地址
Facebook	*https://developers.facebook.com/*
Flickr	*http://www.flickr.com/services/api/*
Foursquare	*https://developer.foursquare.com/*
Reddit	*https://github.com/reddit/reddit/wiki/API*
Twilio	*http://www.twilio.com/*
Twitter	*https://dev.twitter.com/*
YouTube	*https://developers.google.com/youtube/*

用 Pi 提供服务（做 Web 服务器）

除了可以用 Raspberry Pi 从 Internet 服务器上获取信息以外，Pi 自身也可以作为服务器对外提供信息。在 Raspberry Pi 上有很多种 Web 服务器软件可供你选择。例如，传统的 Apache 或 lighttpd，它们都可以用来向外提供 Web 服务。这些服务器通常用于向外提供用于构成网页的 HTML 文件和图片，但它们也可以用于提供声音、视频、软件等其他类型的数据。

现在有一些新的工具可以扩展 Python、Ruby 或 JavaScript 等语言，在程序接收到 HTTP 请求时动态地生成 HTML 页面。这种方式非常适合用于从远端的 Web 浏览器中触发一个物理事件、保存数据或者读取传感器的值。你甚至可以为你的电子项目创建一个 JSON 格式的 API 服务。

Flask 入门

下面我们将使用 Flask（*http://flask.pocoo.org/*）这个 Python 的 Web 框架把 Raspberry Pi 变为一个动态 Web 服务器。Flask 自带了

很多强大的功能，并且还可以通过扩展来支持用户验证、生成表单和访问数据库等功能。当然，在使用 Flask 时，你也可以使用大量的标准 Python 库来帮助你完成工作。

要安装 Flask，需要先安装 pip。如果你还没有安装 pip，可以这样来安装它：

```
pi@raspberrypi ~ $ sudo apt-get install python-pip
```

安装 pip 后，你可以用它来安装 Flask 和相关的依赖包：

```
pi@raspberrypi ~ $ sudo pip install flask
```

要验证 Flask 是否正确安装，创建一个名为 *hello-flask.py* 的新文件并输入代码 10.4 中的代码。不要被这些代码吓倒，如果你现在暂时不能完全理解这些代码也没关系。这段代码中，最重要的部分就是包含 "Hello World!" 字符串的那一块。

代码 10.4 *hello-flask.py* 的源代码

```
from flask import Flask
app = Flask(__name__)❶

@app.route("/")❷
def hello():
    return "Hello World!"❸

if __name__ == "__main__":❹
    app.run(host="0.0.0.0", port=80, debug=True)❺
```

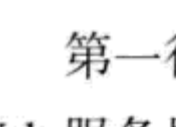
❶ 创建一个名为 app 的 Flask 对象。

❷ 当有人访问网页服务器的根目录时，执行下面的代码。

❸ 向客户端发送 "Hello World!" 字符串。

❹ 判断是否这个脚本是从命令行上直接运行。

❺ 让服务器在 80 端口上监听，并在出错时显示出错信息。



在运行这个程序之前，需要知道你的 Raspberry Pi 的 IP 地址（参考“网络”）。另一种方式是安装 `avahi-daemon`（在命令行上运行 `sudo apt-get install avahi-daemon`），这样可以让你通过 *http://raspberrypi.local* 这样的地址访问到局域网中 Raspberry Pi。如果你需要以这种方式从 Windows 上访问 Raspberry Pi，则需要在 Windows 中启用 Bonjour 服务（*http://support.apple.com/kb/DL999*）。

现在可以运行你的 Web 服务器了，这个程序也需要以 `root` 权限运行：

```
pi@raspberrypi ~ $ sudo python hello-flask.py
 * Running on http://0.0.0.0:80/
 * Restarting with reloader
```

在同一个局域网中的另一台电脑的浏览器地址栏中输入 Raspberry Pi 的 IP 地址，如果浏览器上显示出 "Hello World!"，就表示你已经把 Raspberry Pi 上的 Web 服务器正确地配置好了。与此同时，你也会发现，Raspberry Pi 的终端上会打印出几行信息：

```
10.0.1.100--[19/Nov/2012 00:31:31] "GET / HTTP/1.1" 200 -
10.0.1.100--[19/Nov/2012 00:31:31] "GET /favicon.ico HTTP/
1.1" 404 -
```

第一行信息表示浏览器请求了服务器上的根目录，然后我们的 Web 服务器返回了 HTTP 返回值 200，表示正常返回。第二行信息表示浏览器自动请求了收藏夹图标（通常用于在浏览器地址栏最左侧显示）。由于服务器上没有 *favicon.ico* 文件，所以返回了 404，

表示请求的文件不存在。

如果你想向浏览器返回一个 HTML 格式的网站，把所有的 HTML 代码都写在脚本中显示不是一个好主意。Flask 使用一个名为 Jinja2 的模板引擎（*http://jinja.pocoo.org/docs/templates/*），使你可以用一个包含占位符的页面模板来制作网页，然后再用程序把占位符替换成实际的动态数据。

如果 *hello-flask.py* 脚本还在运行，按 Control-C 键把它停止。

创建名为 *hello-template.py* 并输入代码 10.5 中的代码。在 *hello-template.py* 所在的目录下，创建一个名为 *templates* 的子目录。在 *templates* 子目录中，创建一个名为 *main.html* 的文件并输入代码 10.6 中的代码。这个 HTML 页面模板中所有括号的部分会被解析为变量名，在 Python 脚本调用 `render_template` 函数时，会被实际的数据所替换。

代码 10.5 *hello-template.py* 的源代码

```
from flask import Flask, render_template
import datetime
app = Flask(__name__)

@app.route("/")
def hello():
    now = datetime.datetime.now()❶
    timeString = now.strftime("%Y-%m-%d%H:%M")❷
    templateData = {
        'title' : "HELLO!",
        'time': timeString
    }❸
    return render_template("main.html", **templateData)❹

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80, debug=True)
```

❶ 获取当前日期时间值，并保存在 now 变量中。

❷ 把 now 中的时间值按指定的格式转换成字符串。

❸ 创建一个字典变量（一组键值对，如 title 是键，它所对应的值是 HELLO!）用于渲染模板。

❹ 用 tempateData 字典中的键值对渲染 *main.html* 模板并返回给浏览器。

代码 10.6　*templates/main.html* 的源代码

```
<!DOCTYPE html>
  <head>
    <title>{{ title }}</title>❶
  </head>
  <body>
   <h1>Hello, World!</h1>
   <h2>The date and time on the server is: {{ time }}</h2>❷
  </body>
</html>
```

❶ 用 title 变量的值作为 HTML 页面的标题。

❷ 用 time 变量的值作为页面内容的一部分。

当你运行 *hello-template.py*（与以前一样，需要用 sudo 来运行它）并在浏览器中访问 Raspberry Pi 的 IP 地址时，你应该可以看到一个标题为 “HELLO!” 的页面，上面显示着 Raspberry Pi 上的当前日期时间。

取决于你的网络的情况，你在 Raspberry Pi 上在局域网内搭建的网站很可能无法通过 Internet 直接访问。如果你想让你的网站在局域网外也可以访问，你需要配置网络路由器上的“端口映射”功能。请参考你的路由器说明书了解如何进行相关的配置。

把 Web 与现实世界相连

可以在使用 Flask 的同时也使用其他 Python 库，以便给你的网站增加更多的功能。例如，通过使用 RPi.GPIO 模块（第 8 章），你可以创建一个与现实世界相连的网站。如果想尝试一下的话，请像简易发音板项目中那样按图 8.2 的方法在 GPIO 接口 23、24、25 上连接 3 个按钮。

下面的代码对 *hello-template.py* 进行了扩展，把 *hello-template.py* 复制一份命名为 *hello-gpio.py* 并进行如下修改：添加 RPi.GPIO 模块，并针对读取按钮状态添加一个新的 `route`。新添加的 `route` 会从 URL 中获取一个传入变量的值，用于决定读取哪一个 GPIO 接口的状态。

你还需要创建一个新的网页模板，命名为 *pin.html*。它与 *main.html* 也很相似，所以你可以基于 *main.html* 参考进行修改，参考代码 10.7。

修改过的 *hello-gpio.py*：

```
from flask import Flask, render_template
import datetime
import RPi.GPIO as GPIO
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

@app.route("/")
def hello():
    now = datetime.datetime.now()
    timeString = now.strftime("%Y-%m-%d%H:%M")
    templateData = {
        "title" : "HELLO!",
```

```
        "time": timeString
    }
    return render_template("main.html", **templateData)

@app.route("/readPin/<pin>")❶
def readPin(pin):
    try:❷
        GPIO.setup(int(pin), GPIO.IN)❸
        if GPIO.input(int(pin)) == True:❹
            response = "Pin number " + pin + " is high!"
        else:❺
            response = "Pin number " + pin + " is low!"
    except:❻
        response = "There was an error reading pin " + pin
        + "."

    templateData = {
        "title" : "Status of Pin" + pin,
        "response" : response
    }

    return render_template("pin.html", **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

❶ 添加一个新的 route，并以 GPIO 接口编号作为参数。

❷ 如果缩进的代码中发生了异常，就执行 except 块中的代码。

❸ 从 URL 中获取 GPIO 接口编号，转换为数字，并把对应接口状态设置为输入设置。

❹ 如果读到的接口状态是高电平，则返回表示高电平的文本信息。

❺ 否则返回表示低电平的文本信息。

❻ 如果读取接口状态时发生问题，显示相应的出错信息。

代码 10.7 *templates/pin.html* 的源代码

```
<!DOCTYPE html>
  <head>
   <title>{{ title }}</title>❶
  </head>

  <body>
    <h1>Pin Status</h1>
    <h2>{{ response }}</h2>❷
  </body>
</html>
```

❶ 根据 *hello-gpio.py* 中的 `title` 值渲染页面的标题。

❷ 根据 *hello-gpio.py* 中返回的 `response` 值渲染 HTML 页面。

当上面的脚本运行起来后，如果你在浏览器中访问 Raspberry Pi 的 IP 地址，你仍然会看到之前我们所创建的 “Hello World!” 页面。如果在访问的 URL 的最后加上 */readPin/24*，得到一个类似于 *http://10.0.1.103/readPin/24* 这样的 URL，则会返回一个页面提示这个 GPIO 接口上读取到的状态为低电平。如果你按下 GPIO 24 上所接的按钮不放，并刷新这个页面，则页面上显示状态应该为高电平。

可以尝试按下其他按钮并更换对应的 URL。这段代码最大的优点是，我们只需要写一份读取 GPIO 接口状态并在页面上显示状态的代码，就可以操作所有的 GPIO 接口，就像我们为每一个 GPIO 接口都写了一个状态页面一样。

项目：Web 台灯

在第 7 章的“项目：定时台灯”一节中，我们演示了如何用 Raspberry Pi 制作一个定时开关来控制台灯。现在你学会了 Python 中的 Flask 库，就可以实现一个通过 Web 界面来控制的台灯了。这

个简单的项目演示了如何用 Raspberry Pi 完成一个可以通过 Internet 来远程控制的设备。

在前一个 Flask 例子中，我们实现了用一个程序来控制多个 GPIO 接口。一旦你完成了这个例子，以后你需要通过网络来控制更多设备时，也会变得非常容易。

1. 这个项目所需的硬件与实现“项目：定时台灯”中所用到硬件完全一致。

2. 与定时台灯项目中一样，把 PowerSwitch Tail II 连接到 GPIO 25 接口上。

3. 如果你还有一个 PowerSwitch Tail II，则可以把它接到 GPIO 24 接口上，用于控制其他的电器设备。否则，在 GPIO 24 上连接一个 LED，我们用它来演示如何用一个程序去控制多个设备。

4. 在你的主目录下创建一个新的目录并命名为 *WebLamp*。

5. 在 *WebLamp* 目录中，创建 *weblamp.py* 文件并输入代码 10.8。

6. 在 *WebLamp* 目录中创建一个新的目录并命名为 *templates*。

7. 在 *templates* 目录中，创建 *main.html*，输入代码 10.9 中的代码。

在命令行下，切换到 *WebLamp* 目录中并启动服务器（如果之前已经启动过 Flask 服务器，按 Control-C 键中止）：

```
pi@raspberrypi ~/WebLamp $ sudo python weblamp.py
```

代码 10.8　*weblamp.py* 的源代码

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

pins = {
```

```
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},
    25 : {'name' : 'lamp', 'state' : GPIO.LOW}
}❶

for pin in pins:❷
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)❸
    templateData = {
        'pins' : pins❹
    }
    return render_template('main.html', **templateData)❺

@app.route("/<changePin>/<action>")❻
def action(changePin, action):
    changePin = int(changePin)❼
    deviceName = pins[changePin]['name']❽
    if action == "on":❾
        GPIO.output(changePin, GPIO.HIGH)❿
        message = "Turned " + deviceName + " on."⓫
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."
    if action == "toggle":
        GPIO.output(changePin, not GPIO.input(changePin))⓬
        message = "Toggled " + deviceName + "."

    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)⓭

    templateData = {
        'message' : message,
        'pins' : pins
    }⓮
    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

❶ 创建一个名为 pins 的字典对象，存放 GPIO 接口编号、名字和状态信息。

❷ 把每个 GPIO 接口都设为输出模式，并置为低电平。

❸ 读取每个 GPIO 接口的状态，并把状态值存入 pins 字典中的对应项。

❹ 把 pins 字典对象放入模板数据字典中。

❺ 用模板数据字典渲染 *main.html* 页面并把结果返回给用户。

❻ 当用户访问带有接口编号及操作的URL时，执行下面的函数。

❼ 把 URL 中的接口编号转换为数值。

❽ 获取发生变化的接口所对应的设备名称。

❾ 如果 URL 中指定的操作是“on”，执行下面缩进的代码块。

❿ 把 GPIO 接口状态设置为高电平。

⓫ 保存表示状态的字符串并把它传给页面模板。

⓬ 读取接口状态，把接口设置为与当前状态相反的状态（切换接口状态）。

⓭ 读取每一个接口的状态，并把它们的状态存入 pins 字典对象中。

⓮ 把pins字典对象与表示状态的字符串放入模板数据字典中。

代码 10.9 *templates/main.html* 的源代码

```
<!DOCTYPE html>
<head>
  <title>Current Status</title>
</head>

<body>
  <h1>Device Listing and Status</h1>

  {% for pin in pins %}❶
  <p>The {{ pins[pin].name }}❷
```

```
    {% if pins[pin].state == true %}❸
      is currently on (<a href="/{{pin}}/off">turn off</a>)
    {% else %}❹
      is currently off (<a href="/{{pin}}/on">turn on</a>)
    {% endif %}
    </p>
    {% endfor %}

    {% if message %}❺
    <h2>{{ message }}</h2>
    {% endif %}

</body>
</html>
```

❶ 遍历 pins 字典对象中的所有接口信息。

❷ 显示接口的名称。

❸ 如果接口状态为高电平，显示对应的设备正处在打开状态，并显示一个链接，可以用于关闭设备。

❹ 否则，显示对应的设备处在关闭状态，并显示一个链接，可以用于打开设备。

❺ 如果向模板中传入了状态字符串，则把它显示出来，如图 10.2 所示。

这样设计这段代码的优点在于，增加要控制的设备时非常容易，只需把相应设备的信息加入 pins 字典对象中即可。当你重启服务器后，新添加的设备就会在状态列表中显示出来，并且相应的控制链接也可以正常使用。

这段代码还包含了一个很有用的功能：如果你需要切换某个设备的工作状态，只需在手机上点击一次就可以了。可以把 *http://ipaddress/pin/toggle* 这个 URL 加入浏览器的书签，访问这个 URL 会自动检查相应接口的状态并切换到相反的状态上。

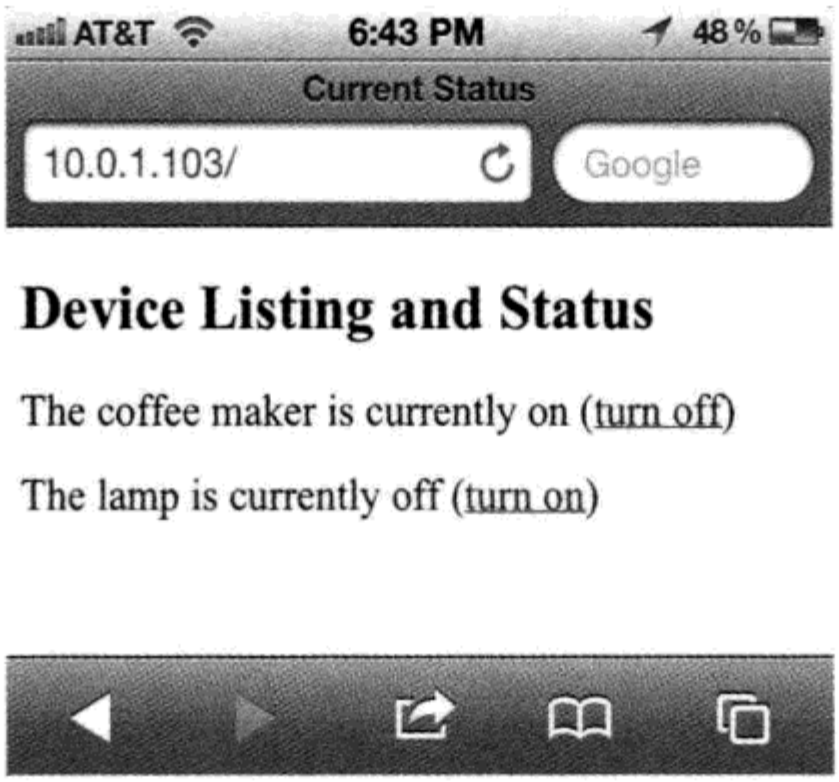


图10.2　从手机浏览器上看到设备控制界面

进一步学习

Requests 库

（*http://docs.python-requests.org/en/latest/*）

Requests 库的主页上提供了非常详尽的文档和容易理解的实例。

Flask

（*http://flask.pocoo.org/*）

本章内容中还有很多 Flask 的特性没有提到，它的官方网站上提供了它完整的功能列表。

Flask 扩展

（*http://flask.pocoo.org/extensions/*）

通过使用 Flask 扩展，可以更方便地向你的网站上添加功能。