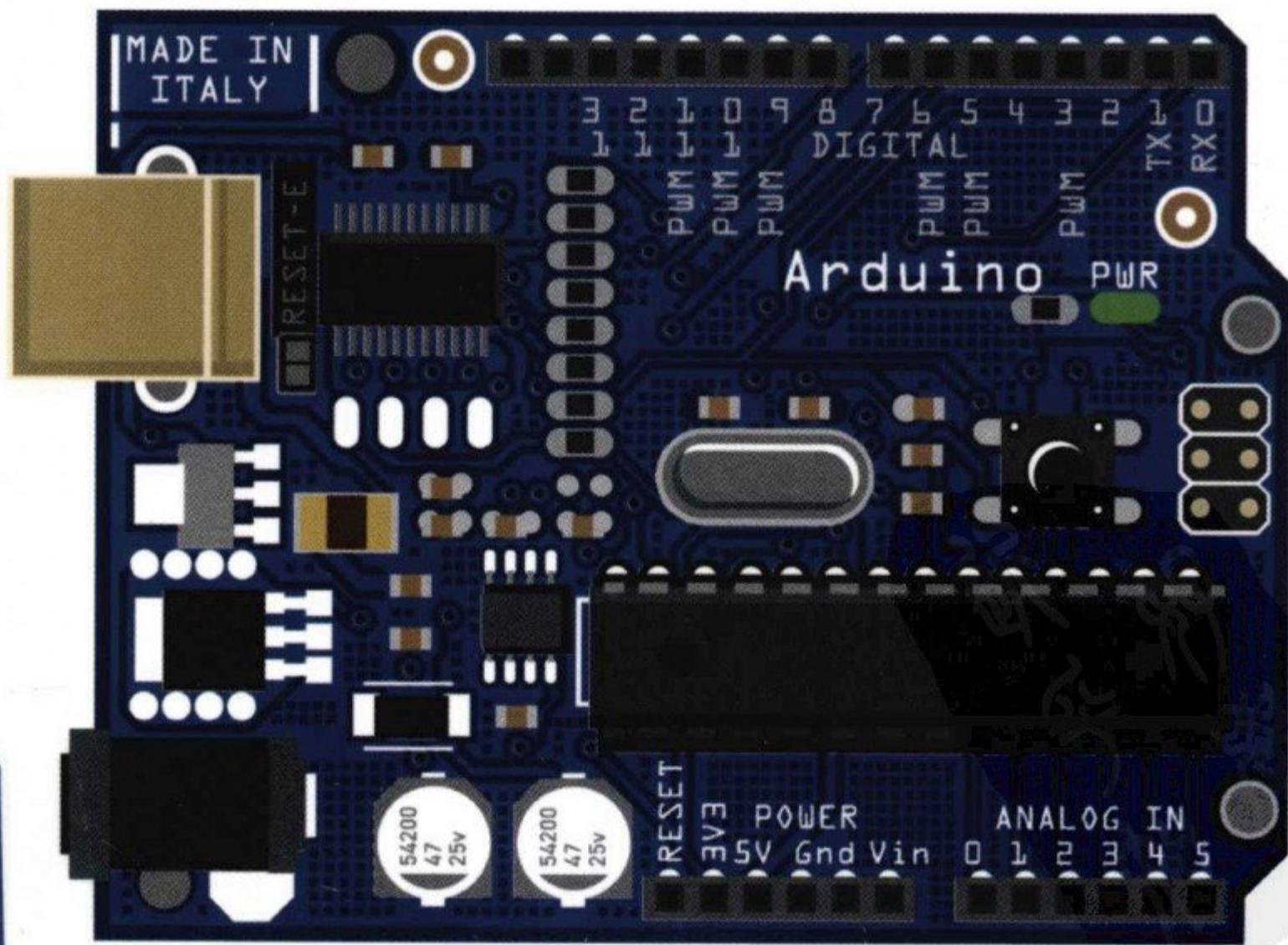


Arduino编程

(英) **Simon Monk** 著
刘惨楠 译

从零开始

Programming Arduino
Getting Started with Sketches



科学出版社

Arduino 编程就这么简单！

从易学易用的实例学Arduino编程：本书揭示了Arduino的软件特性，介绍了如何在Arduino上用C语言编写能稳定运行的Sketch。你不需要具备编程经验，书中的示例程序下载即可用，你也可以对其稍做修改后满足自己的需求。

- 理解Arduino硬件原理
 - 安装IDE并启动，上传你的第一个Sketch
 - 学习C语言基础
 - 在Sketch中写入函数
 - 使用数组和字符串
 - 调用模拟/数字IO
 - 探究数据存储的方法
 - 为LCD显示器编程
 - 使用网络扩展板将Arduino变成Web服务器
- 本书相关代码下载：[www.arduino\(book\).com](http://www.arduino(book).com)

McGraw-Hill
全球智慧中文化

<http://www.mheducation.com>

科学出版社 东方科龙公司
联系电话：010-82840399
E-mail: boktp@mail.sciencep.com
有关网址：<http://www.okbook.com.cn>

销售分类建议：工业技术/电子技术



www.sciencep.com

ISBN 978-7-03-036110-3

9 787030 361103 >

定 价：28.00 元

Arduino编程从零开始

〔英〕Simon Monk 著

刘惨楠 译

科学出版社
北京



内 容 简 介

本书从C语言基础开始，深入浅出地介绍了Arduino编程，对于零基础的初学者和有一定基础的Arduino玩家都有一定的帮助。

本书使用大量的程序范例一步步、手把手地教读者怎样为一块Arduino板进行编程，进而实现想要的功能，从让Arduino板载LED以各种不同的方式来闪烁，一直到通过附加扩展板实现更加复杂和实用的功能，甚至让Arduino板变成一个小型Web服务器。

本书适合广大“创客”和电子爱好者学习，也可供中学生科技竞赛、高等院校相关专业及电子设计（制作）比赛参考。

图书在版编目（CIP）数据

Arduino 编程从零开始/（英）Simon Monk著；刘穆楠译。
—北京：科学出版社，2013.2
ISBN 978-7-03-036110-3
I. A… II. ①S… ②刘… III. 单片微型计算机—程序设计 IV. TP368.1
中国版本图书馆CIP数据核字（2012）第286071号

责任编辑：喻永光 杨 凯 / 责任制作：董立颖 魏 谨

责任印制：赵德静 / 封面制作：段淮沱

北京东方科龙图文有限公司 制作

<http://www.okbook.com.cn>

科学出版社出版

北京东黄城根北街16号

邮政编码：100717

<http://www.sciencep.com>

北京中科印刷有限公司 印刷

科学出版社发行 各地新华书店经销

*

2013年2月第一版 开本：A5 (890×1240)

2013年2月第一次印刷 印张：6

印数：1—5 000 字数：128 000

定价：28.00元

（如有印装质量问题，我社负责调换）

Simon Monk

Programming Arduino™: Getting Started with Sketches

0-07-178422-5

Copyright © 2012 by McGraw-Hill Companies, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation edition is jointly published by McGraw-Hill Education(Asia) and Science Press Ltd. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2013 by the McGraw-Hill Education(Asia), a division of the Singapore Branch of the McGraw-Hill Companies, Inc. and Science Press.

版权所有。未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔（亚洲）教育出版公司和科学出版社有限责任公司合作出版。此版本经授权仅限在中华人民共和国境内（不包括香港特别行政区、澳门特别行政区和台湾）销售。

版权© 2013由麦格劳-希尔（亚洲）教育出版公司与科学出版社有限责任公司所有。

本书封面贴有McGraw-Hill公司防伪标签，无标签者不得销售。

北京市版权局著作权合同登记号：01-2012-7607

著者简介

Simon Monk是一个拥有计算机及控制学学士学位、计算机软件工程学硕士学位的工程师。早在读书的时候，他就已经是一位互动电子爱好者，并且常常为业余电子爱好者杂志撰稿。另外，他还是*30 Arduino Projects for the Evil Genius*（《基于Arduino的趣味电子制作》，科学出版社）及*15 Dangerously Mad Projects for the Evil Genius*的作者。

前 言

Arduino为创建基于单片机的工程项目提供了一种低价易用的技术。用一点点电子学的知识，就可以让你的Arduino做任何事——从控制一个娱乐设备的灯光到管理太阳能系统的能源。

有很多介绍怎样将你要实现的东西连接到Arduino的书，包括本书作者的《基于Arduino的趣味电子制作》，但本书的重点是对Arduino编程。

本书将介绍怎样简单并有趣地对Arduino编程，避免使用生涩难懂的代码而让制作变成痛苦的历程。你将从Arduino用到的基础C语言开始，一步步地完成对Arduino编程的过程。

什么是Arduino？

Arduino是一块带有用来连接电脑USB接口的小型单片机电路板，上面还有很多的插母，用来连接外部电子设备，如电机、继电器、光学传感器、激光二极管、扬声器、麦克风……它们可以通过电脑USB接口或9V电池，或者是独立电源来供电。它们可以直接通过电脑控制，或者是通过电脑编程之后断开连接，独立工作。

电路板的设计是开源的，这意味着任何人都可以制作Arduino兼容板，兼容板的制作可能使得造价变得更低。

基本板通过插在Arduino板上的一些扩展板来增加功能，在

本书中我们将用到2个扩展板——1个LCD显示扩展板和1个网络扩展板——这可以让我们的Arduino变成一个小型Web服务器。

用于Arduino编程的软件使用起来非常简单，而且在Windows、Mac和Linux平台下都是免费下载的。



你需要些什么？

本书是为初学者准备的，但是它对已经使用过Arduino一段时间的人也同样有用——如果他想学习更多关于Arduino编程的知识，或者想对基本原理有更深的理解的话。

你不需要有任何的编程经验或者是技术背景，而且实践本书内容也不需要使用锡焊，你需要的仅仅是学习和实践的欲望。

如果你想完成本书教授的大部分内容或者想自己动手做点试验的话，手头需要准备下面的东西：

- * 一些硬芯导线；
- * 一个便宜的数显万用表。

这两样东西花几十块钱在电子市场都可以轻松买到。当然，你还需要一块Arduino UNO板。

如果你想更深入一些，做一个含有网络扩展板和LCD 扩展板的试验，那么你就要在网店里买这些扩展板了，详情请见第9章和第10章。



怎样使用本书

本书带你从浅显的内容开始，让你逐渐用所学到的知识做一些东西。如果你有一定经验或背景，可以跳过一些章节来选择适

合自己的切入点。

本书由以下章节组成。

第1章 这就是Arduino

对Arduino硬件的介绍。本章描述了Arduino可以做些什么，以及现在流行的各种Arduino板。

第2章 从零开始

本章开始你的第一个实验：安装软件，接通电源，上传你的第一个Sketch。

第3章 C语言基础

本章介绍了C语言的基础知识。对于完全没有接触过C语言的新手，本章同样适合编程入门。

第4章 函数

本章介绍了在Arduino Sketch中写入函数的核心思想，这些Sketch通过可运行的代码范例加以演示。

第5章 数组和字符串

本章你将学会制作和使用比一般常数和变量更高级的数据结构，将通过一步步地开发一个莫尔斯码翻译器的范例来阐明这些思想。

第6章 输入和输出

本章你将学会怎样在你的Arduino程序中使用数字和模拟的输入/输出，万用表可以告诉你在Arduino的输入/输出连接中到底发生了什么。

第7章 标准Arduino库

本章介绍怎样使用标准Arduino库中的标准Arduino函数。

第8章 数据存储

本章你将学习在EEPROM中保存数据时是怎样写Sketch的，以及如何使用Arduino内存。

第9章 LCD显示器

在本章中，你将用到LCD 扩展板库，编程制作一个简单的USB信息板。

第10章 Arduino网络编程

如果你有一点HTML和HTTP的知识，你将学会怎样让Arduino像一个Web服务器一样工作。

第11章 C++和库

本章中，你将超越C语言，接触到面向对象语言，并且制作你自己的Arduino库。



学习资源

本书由合作网站www.arduinoobook.com提供支持，你可以在该站点找到本书中用到的所有开源代码及其他资源，如勘误表。



目 录

第1章 这就是Arduino

1.1 单片机	2
 开发板	3
1.2 Arduino板概览	4
 供 电	5
 电源接口	5
 模拟输入	5
 数字接口	6
 单片机	6
 其他元器件	7
1.3 Arduino的起源	8
1.4 Arduino大家庭	9
 UNO、Duemilanove和Diecimila	10
 Mega	11
 Nano	11
 Bluetooth	12
 Lilypad	12
 其他官方板	14

目 录

	Arduino兼容板	14
1.5	总 结	14

第2章 从零开始

2.1	开 机	16
2.2	安装软件	16
2.3	上传你的第一个Sketch	17
2.4	Arduino应用程序	22
2.5	总 结	25

第3章 C语言基础

3.1	编 程	28
3.2	什么是计算机语言	30
3.3	再闪烁一次	35
3.4	变 量	37
3.5	C语言实验	40
	数字变量和运算式	42
3.6	指 令	44
	if语句	45
	for循环	47
	while循环	51
	#define指令	52
3.7	总 结	53

第4章 函 数

4.1 什么 是 函 数?	56
4.2 参 数	57
4.3 全局、局部和静态变量	59
4.4 返回值	63
4.5 其他变量类型	65
浮点数	65
布 尔	66
其他数据类型	67
4.6 编程风格	68
首行缩进	69
大括号	70
留 白	71
注 释	71
4.7 总 结	73

第5章 数组和字符串

5.1 数 组	76
SOS莫尔斯码所使用的数组	80
5.2 字符串数组	81
字符串字面值	82
字符串变量	83
5.3 莫尔斯码翻译器	84

目 录

数 据	85
全局变量和setup	86
loop函数	87
函数flashSequence	90
函数flashDotOrDash	91
完整代码	92
5.4 总 结	95

第6章 输入和输出

6.1 数字输出	98
6.2 数字输入	102
上拉电阻	103
内部上拉电阻	106
消 抖	107
6.3 模拟输出	114
6.4 模拟输入	116
6.5 总 结	118

第7章 标准Arduino库

7.1 随机数	120
7.2 数学函数	123
7.3 位操作	124
7.4 高级输入/输出	126
声音生成	126

读取移位寄存器	127
7.5 中 断	128
7.6 总 结	130

第8章 数据存储

8.1 常 量	132
8.2 PROGMEM指令	133
8.3 EEPROM	134
在EEPROM中存储整数	136
在EEPROM中存储浮点数 (union)	137
在EEPROM中存储字符串	138
清空EEPROM的内容	139
8.4 压 缩	139
范围压缩	140
8.5 总 结	141

第9章 LCD显示器

9.1 USB 信息板	145
9.2 使用显示器	148
9.3 其他LCD库函数	148
9.4 总 结	149

第10章 Arduino 网络编程

10.1 网络扩展板	152
------------------	-----

目 录

10.2 和Web服务器通信	152
 HTTP	153
 HTML	153
10.3 将Arduino用作Web服务器	154
10.4 通过网络来设置Arduino的针脚	159
10.5 总 结	166

第11章 C++和库

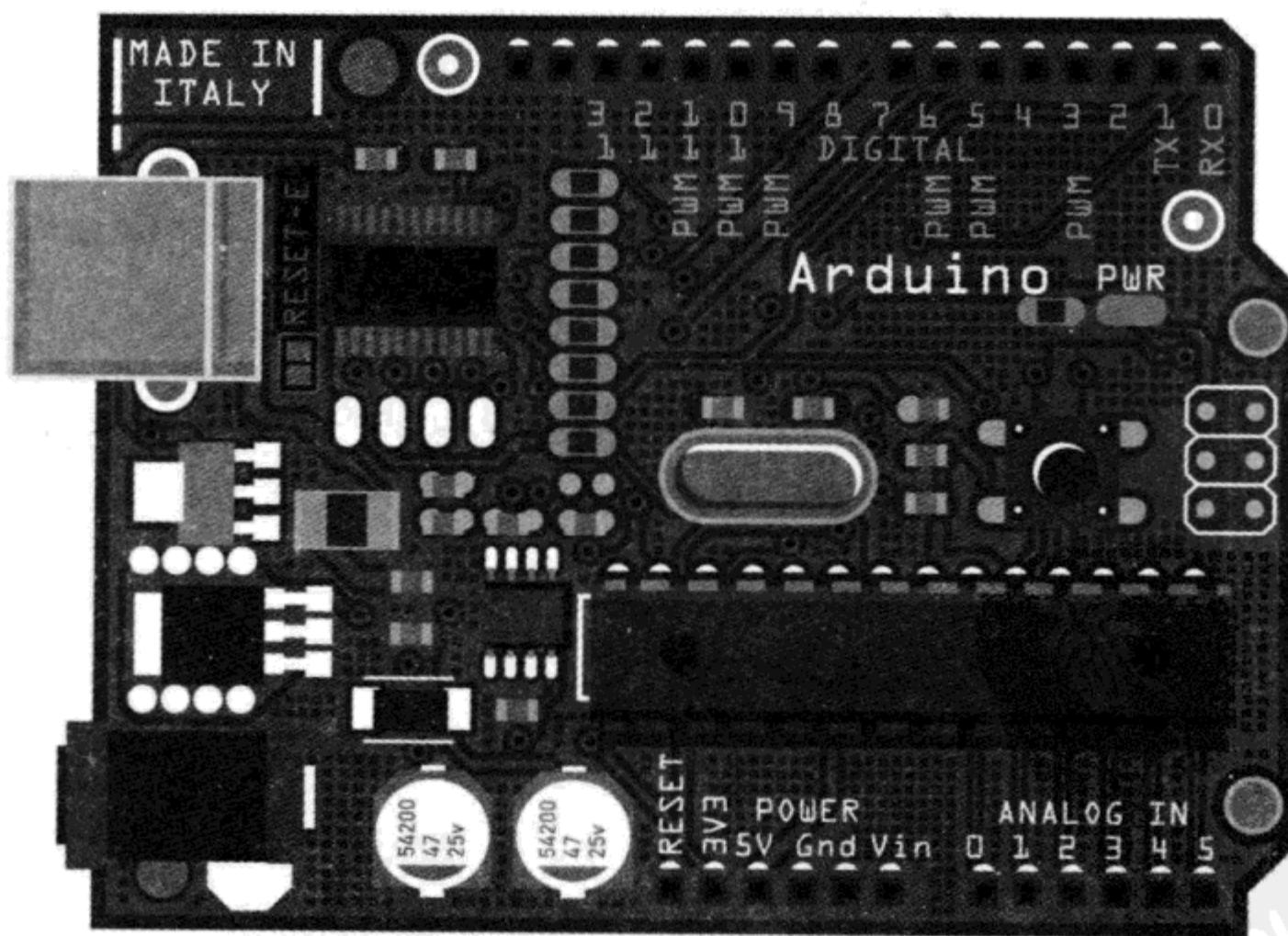
11.1 面向对象	168
 类和方法	168
11.2 内建库示例	169
11.3 写一个库	169
 头文件	170
 实现文件	172
 完成你的库	173
11.4 总 结	177

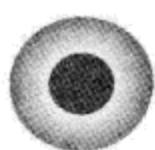


第1章

这就是Arduino

This Is Arduino





Arduino是一种备受电子发烧友关注的单片机平台，其易用和开源的特性使得想制作电子小产品的人有了一个很棒的选择。

最终，你可以通过在其插母上连接其他电子元器件以控制别的东西，比如：开关灯或电机，或者用来感知光、温度。Arduino有的时候被描述为交互装置（physical computing），这是因为Arduino可以通过USB连接到电脑，所以你可以将Arduino用作通过电脑控制电子元器件的接口。

本章是对Arduino的简介，包括了Arduino的历史和背景，也是对硬件的一个总览。

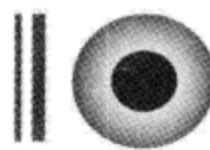
1.1 单片机

Arduino的核心是一个单片机，板子上的其他所有东西都是用于供电或者和电脑连接的。

单片机实际上是集成在一块芯片上的微型电脑，它具有第一台家用电脑所具有的任何东西，甚至更多。它有处理器，有1~2KB的RAM用于存储数据，还有几KB的EPROM或闪存用于存放你的程序，而且它还有输入/输出针脚，用于连接单片机和其他电子元器件。

输入可以是数字信号（开或关）或模拟信号（电压值），这样就可以连接很多不同种类的传感器：光、温度、声音……

输出也可以是模拟或数字信号，所以你可以通过将一个针脚设为开或关（0V或5V）直接控制一个LED的开关；或者用输



出信号去控制更高功率的设备，如电机。Arduino也可以设定输出模拟电压——将某个针脚的输出设定为一个特定的电压，这样比起简单的控制开关，你可以直接控制电机的转速或者是光源的亮度。

Arduino板上的单片机安装在板中央的28针插座中，这个单独的芯片包含了内存处理器和输入/输出针脚所需的所有电子元器件。它是由Atmel公司制造的，Atmel是最大的单片机制造商之一。事实上，每个单片机制造商都会生产不同系列的数十种不同的单片机。这些单片机可不全是为了我们这样的电子发烧友准备的，我们只是这个广大市场中的一小部分，其最终目的是为了植入消费品中，包括汽车、洗衣机、DVD播放器、儿童玩具，甚至是空气净化器。

Arduino最大的好处就是通过一个单片机的标准化把这些选择减少、简单化（后面我们会看到，这么说也不是特别的准确，但是已经很接近了）。这意味着，当你开始着手一个新的项目时，你不需要首先权衡众多单片机的利弊来做选择。

◆ 开发板

我们已经了解到单片机实际上只是一个芯片，如果没有其他的一些电子元器件为其提供稳定和精确的电源并和为其编程的电脑相连，它是无法独立工作的（单片机对这一点很挑剔）。

这就是为什么会有开发板的原因。Arduino实际上就是一款硬件开源设计的单片机开发板。也就是说，其印制电路板的设计文档和原理图都是对公众开放的，而且任何人都可以免费使用这



个设计来生产和销售他们自己的“Arduino”板。

包括生产Arduino板所用ATmega328芯片的Atmel公司在内的所有单片机制造商都提供他们自己的开发板和编程软件，虽然价格不贵，但它们都是为专业的电子工程师而不是发烧友所设计的。这意味着，这种板和软件更难以使用，并且在你真的用它们做出点有用的东西之前需要学习更多的专业知识。

1.2 Arduino板概览

图1.1是一块Arduino板，让我们先大致看一下板子上众多的元器件。

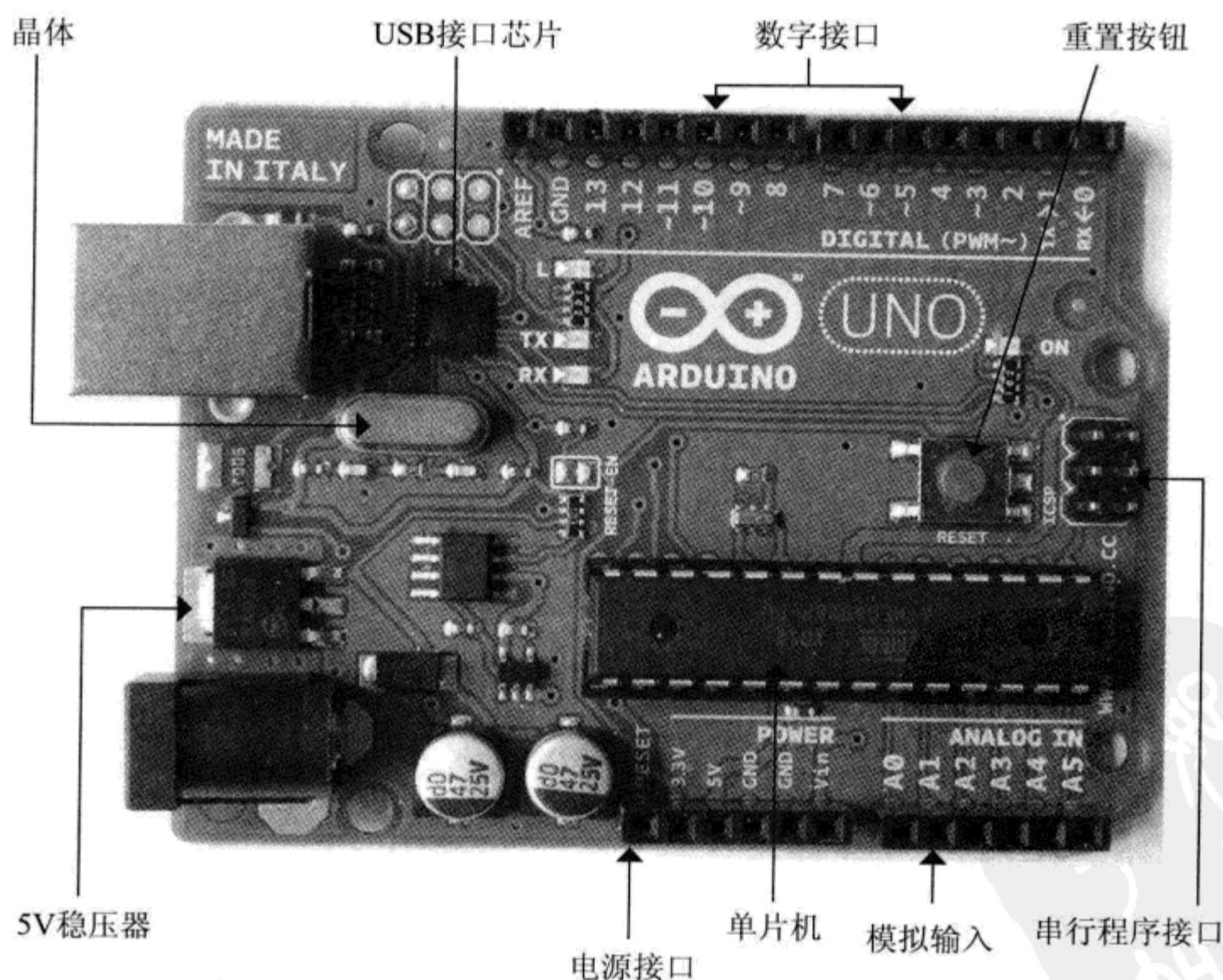
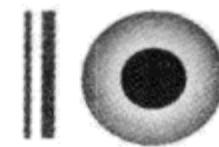


图1.1 一块Arduino UNO板



◆ 供 电

在图1.1中，USB接口下面是5V稳压器，它把由电源插孔供给的7~12V电压调节至稳定的5V。

其实，作为一个直插在板表面的元器件来说，5V稳压器占用的空间有点大，因为它需要散去将电压调节至需要值所产生的热量——这在驱动被动电子器件的时候很必要。

◆ 电源接口

接下来我们来看一看图1.1底部的接口，接口的旁边标有名称，第一位是重置（Reset）。它和Arduino板上重置按钮的作用是一样的，就像重启电脑一样。使用重置接口来重启单片机，将使它从头开始运行程序。若要用重置接口来重启单片机，暂时将此接口设为低电平（将其连接到0V接口）即可。

其他的接口用于提供其他不同的电压（3.5V、5V、GND和9V）。就像它们旁边的名称所标示的一样，GND又称地线，也就是0V，是板上其他电压值的参考值。

◆ 模拟输入

这个被标为A0~A5的模拟信号6针接口，可以用来测量连接到其上的电压，测量值可以在Sketch中使用。注意，它们测量的是电压而不是电流，只有很微小的电流从它们流向地线，因为它们有非常大的内部电阻，只能允许很小的电流流入针脚。

虽然这些输入被标做模拟，而且默认为模拟输入，但是这些接口也可以被用来当做数字信号的输入和输出。



数字接口

现在我们再来看看顶部标记有“DIGITAL(PWM~)”的接口0~13，它们既可以用作输入，也可以用作输出。当用作输出时，它们就像我们前面提到过的供电电压一样，但并不是全5V，而是通过你的Sketch打开或关闭。通过Sketch把它们打开时为5V，关闭时则为0V。正如供电接口一样，你要小心，别超出最大电流承受值。接口0和1也被标为RT和TX，用来接收和传送数据。

数字接口可以提供40mA（毫安）的5V电压。这足够点亮一个标准的LED了，但是还不足以驱动电机。

单片机

继续我们对Arduino板的概览，单片机芯片是那个黑色长方形的28针器件。它被安装在一个双列直插式插座中，这样方便更换。用在Arduino UNO板上的单片机型号为ATmega328，下面根据其内部框图（图1.2）来说明这个芯片的主要特征。

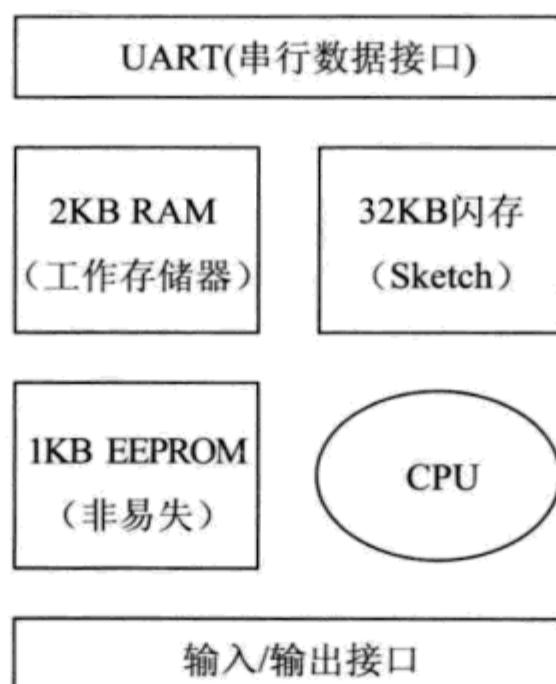


图1.2 ATmega328内部框图



这个器件的心脏——更恰当一些的说法是大脑——中央处理器（CPU），它控制器件中发生的所有事情，取出存放在闪存中的程序指令并执行它们，这可能涉及从工作存储器（RAM）中取出数据，改变它们，再把它们放回去；或者，将某一个数字输出从0V变为5V。

EEPROM存储器和闪存有一点类似，只不过它是非易失存储。也就是说，你可以关掉设备再打开，而EEPROM中的东西不会丢失。闪存用于存储程序指令（从Sketch中得到），EEPROM则用于存储那些你不想因为重置或者关闭设备而丢失的数据。

其他元器件

单片机左上角有一个小的银色长方形的元器件。这是一个石英晶体。它每秒振荡 16×10^6 次，每振荡1次，单片机可以完成1次运算——加法、减法或其他数学运算。

晶体右边是重置按钮。按这个按钮会发送一个逻辑脉冲到单片机的重置针脚，使单片机重新运行程序并清空内存。注意，任何在设备上的程序都不会丢失，因为它们是被存储在非易失闪存——即使设备不通电也可以记忆的存储器中的。

重置按钮的右边是串行程序接口，它提供了另一种不使用USB为Arduino编程的方法。但因为我们已经有了USB接口和对应的软件，而且使用上更方便，所以我们几乎不使用这一功能。

紧挨着USB接口的右边是USB接口芯片，此芯片将USB所用的信号电平转化为可以直接被Arduino板所使用的电平。



1.3 Arduino的起源

Arduino最初是为了教学而开发的，随后（2005年）被Massimo Banzi和David Cuartielles进行了商业性开发。从那时起，就逐渐因为其易用性和耐久性在制造商、学生和艺术家的群体中取得了极大的成功。

它获得成功的另外一个关键因素是，在知识共享许可（Creative Commons license）下Arduino所有的设计都是可以免费获得的。这样就出现了很多廉价的替代板。只有“Arduino”这个名字是受保护的，所以“山寨”产品一般名字里都有一个“duino”，如Boarduino、Seeeduino和Freeduino。但是，意大利的官方板仍然卖得很火。很多大的零售商只出售官方板，因为它们的包装和质量更好一些。

Arduino获得成功还有一个原因，它并不仅限于单片机板，有相当多的和Arduino兼容的扩展板可以直接插在Arduino板上使用。因为几乎在每个你能想到的领域都有对应的扩展板，所以平常可以避免使用锡焊，而是把扩展板一个叠一个地插在一起。下面是最流行的几种扩展板：

- 网络扩展板——使Arduino有Web服务器的功能；
- 电机扩展板——驱动一个电机；
- USB Host扩展板——可以控制USB设备；
- 继电器扩展板——通过Arduino开关继电器。

图1.3是一个插有网络扩展板的Arduino UNO板。

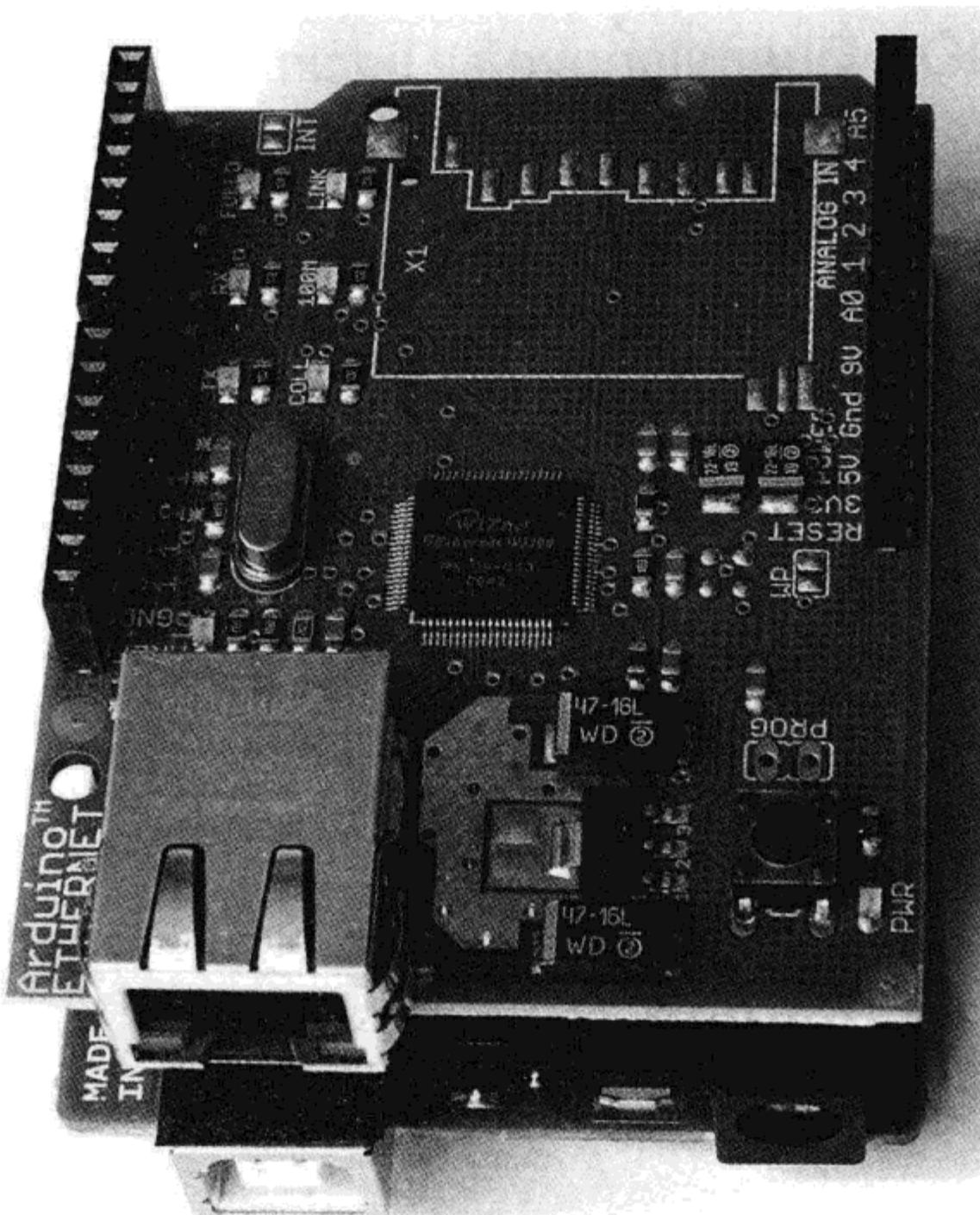
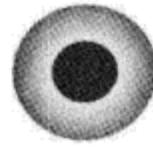


图1.3 插有网络扩展板的Arduino UNO板

1.4 Arduino大家庭

对各种各样的Arduino板都有一定的了解是有必要的。我们将使用Arduino UNO板作为我们的标准设备，当然这也是目前使用最为广泛的Arduino板。不过，所有的板子都用同样一种语言编程，而且大部分板子对外的接口也都是统一的，所以如果你用起别的板来也很简单。



UNO、Duemilanove和Diecimila

Arduino UNO是Arduino板系列最流行的新版本。这个系列包括Diecimila（意大利语“10 000”）和Duemilanove（意大利语“2011”）。图1.4为一块Arduino Duemilanove板。现在你可能已经猜到Arduino是一个意大利的发明了。

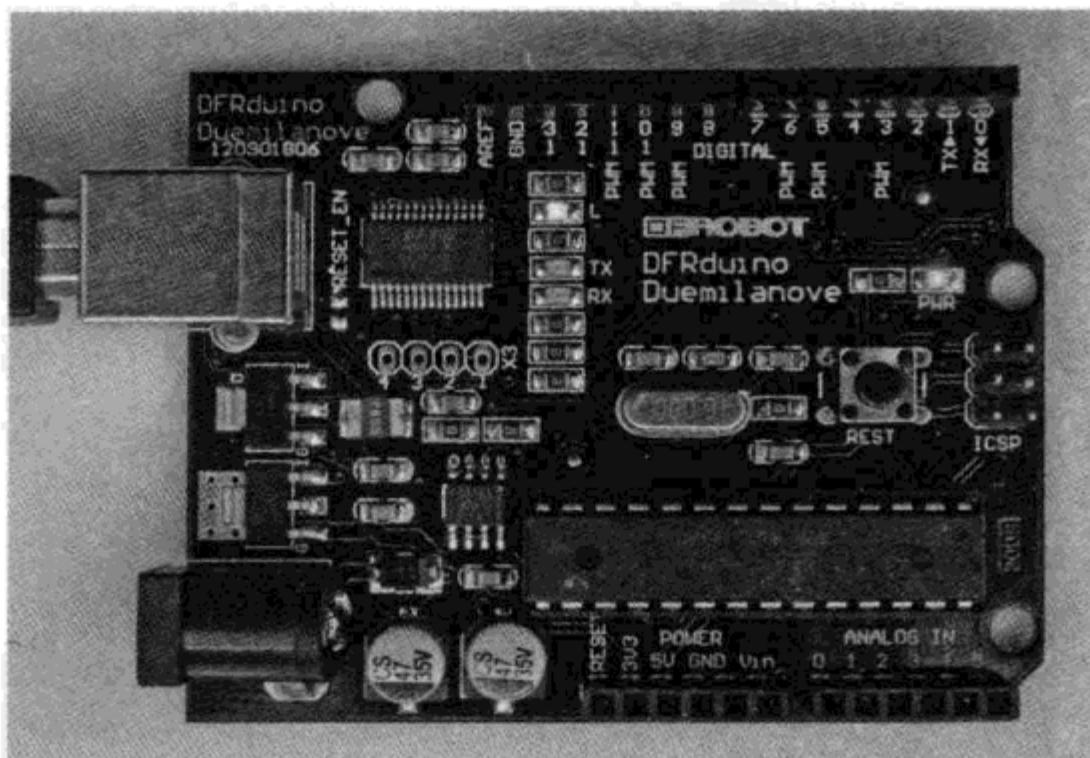


图1.4 Arduino Duemilanove

这些相对老一些的板子看上去和Arduino UNO很像。它们有着一样的接口，并且基本上都互相兼容。

UNO和早期板子最显著的区别是，UNO使用了不同的USB芯片，这对你怎样使用板子并没有影响，但它使得Arduino软件的安装更简单，并且和电脑之间数据交换的速度更快。

UNO还可以在其3.3V供电接口处提供更大的电流，并且UNO板都是配备ATmega328芯片的，早期的一些板子用的是ATmega328或者是ATmega168。ATmega328有更大的存储器，不过除非你要创建一个很大的Sketch，否则它们用起来没什么区别。



Mega

Arduino Mega是Arduino板中的高端产品，如图1.5所示。它上面有大量的输入/输出接口，却巧妙地将附加的接口安排在板子的一端，使它们对Arduino UNO和Arduino可用的所有扩展板可以针脚兼容。

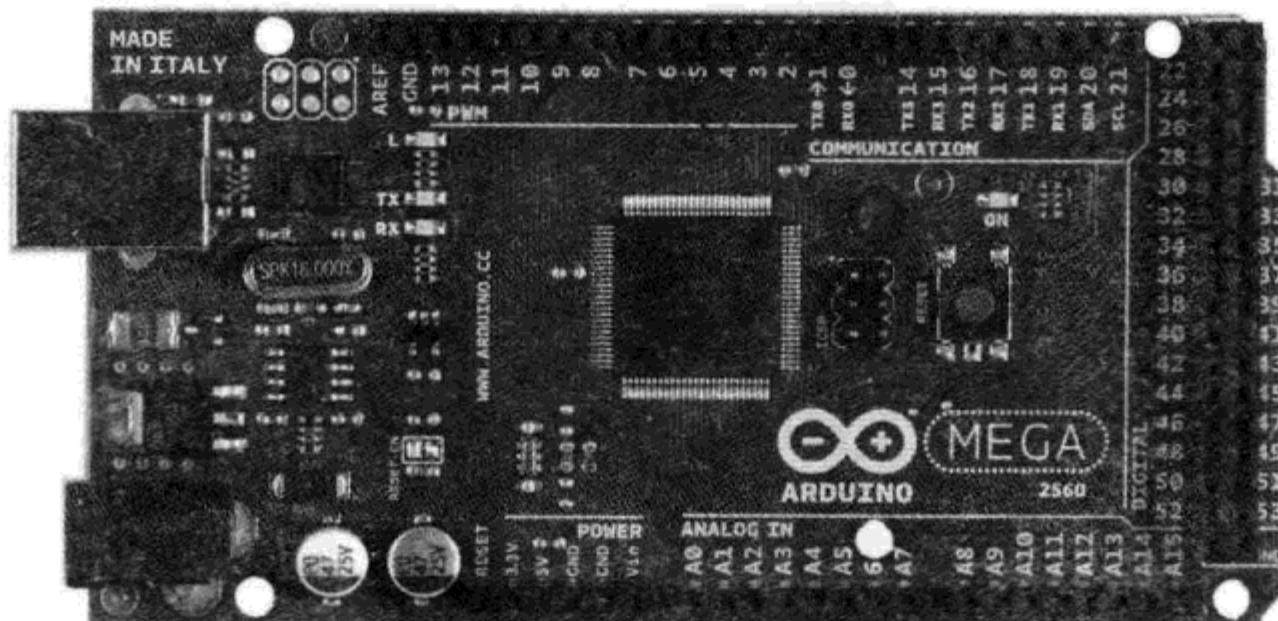


图1.5 Arduino Mega

它使用一个有更多输入/输出针脚的处理器ATmega1280，并且是固定安装在板子上的表面装贴芯片，不像UNO或者其他板子。如果你不小心弄坏了处理器就没得换了。

附加的连接器被安排在板的一端，下面是Mega拥有的一些额外功能：

- 54个输入/输出针脚；
- 128KB闪存，用来存储Sketch和固定数据；
- 8KB RAM，4KB EEPROM。

Nano

Arduino Nano如图1.6所示，和免焊面包板一起使用时特别方



便。如果给它装上插针，就可以像一块芯片一样插到面包板上。

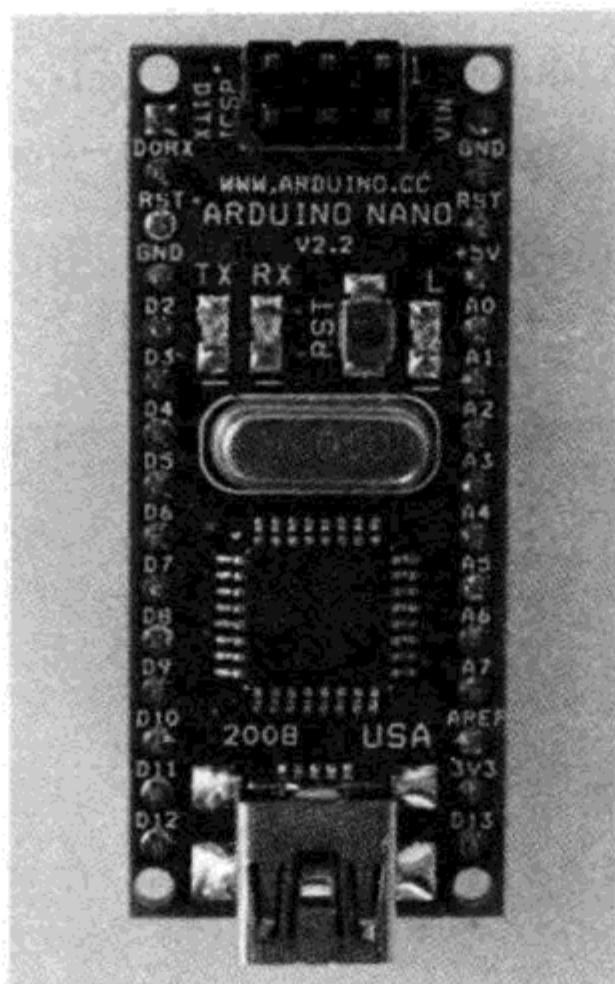


图1.6 Arduino Nano

Nano的不足之处是它太小了，所以不能插上UNO用的扩展板。



Bluetooth

Arduino Bluetooth（图1.7）是一个很有意思的设备，它使用蓝牙硬件取代了USB接口，这样你甚至可以无线为你的设备编程。

Arduino Bluetooth可不便宜，要是自己为Arduino UNO加上第三方的蓝牙组件就便宜得多了。



Lilypad

Lilypad（图1.8）是一块很小、很薄、可以缝到衣服上的Arduino板，是为那些“智能衣物”所设计的。

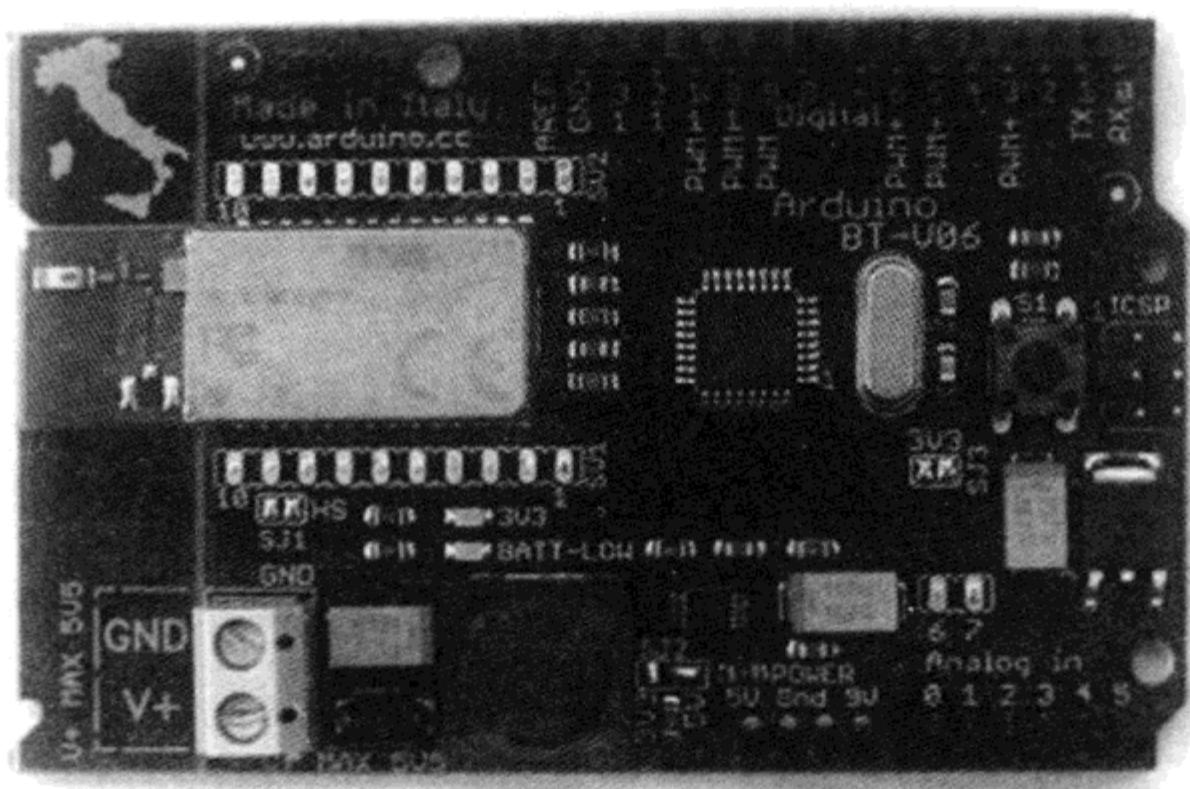
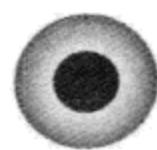


图1.7 Arduino Bluetooth

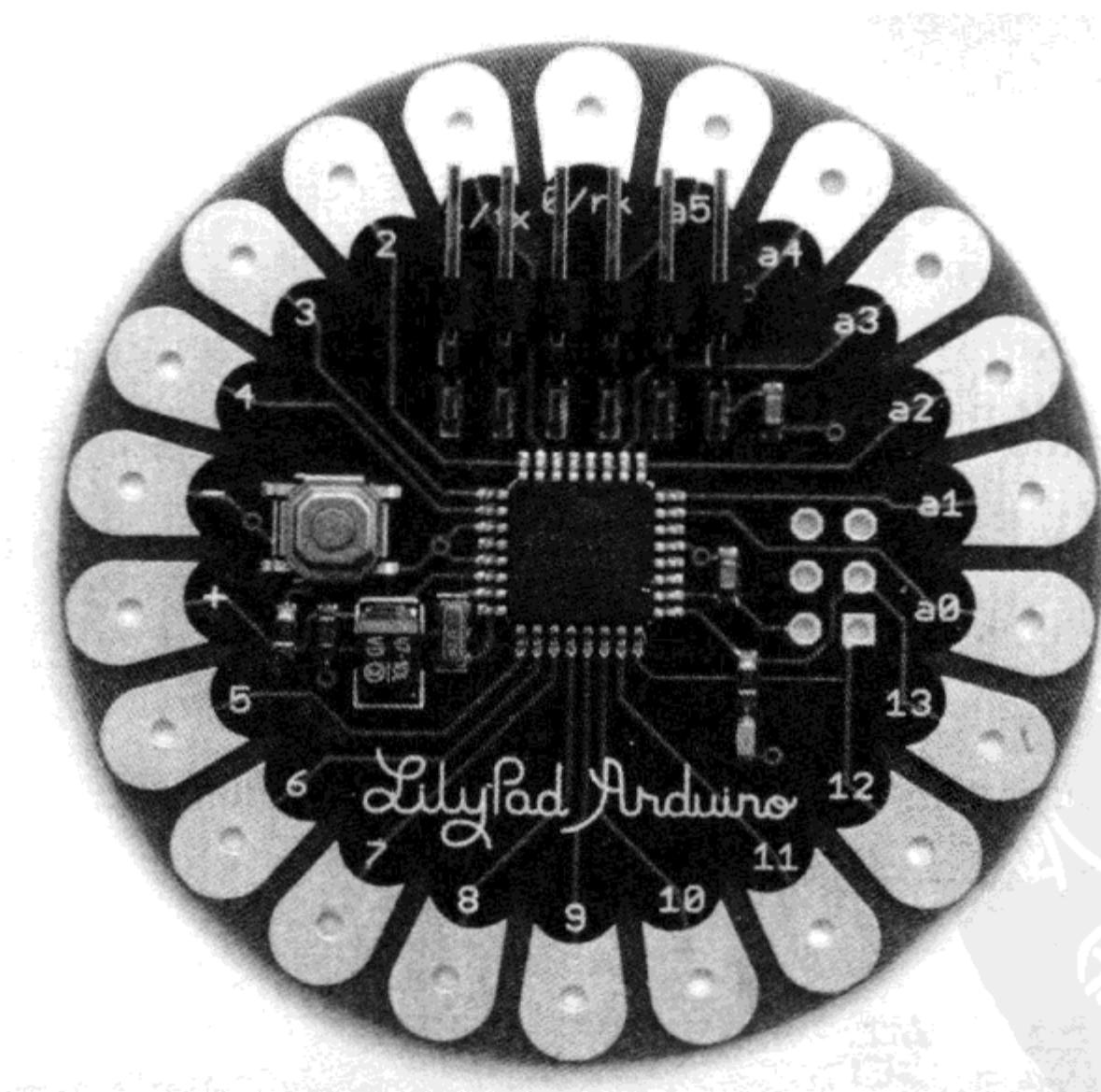


图1.8 Arduino Lilypad



Lilypad没有USB接口，你必须用单独的适配器为它编程。Lilypad的设计很漂亮。

其他官方板

前面提及的Arduino板都是应用最广泛并且最流行的，但是Arduino板的范围还在不断变化，所以要想了解完整的最新Arduino系列，请移步官方网站：www.Arduino.cc/en/mian/hardware。

Arduino兼容板

Arduino兼容板大致有两类，一类是使用标准的开源硬件设计做出的更廉价的产品，下面是你可以搜索到的几种板子：

- Roboduino；
- Freeduino；
- Seeeduino（没错，是有3个“e”）。

更有趣的是，一些Arduino的兼容设计旨在某些方面扩展或提升Arduino，而新的兼容板不停地出现，数量太多，这里不便一一提及。下面是一些更为有趣和流行的兼容板：

- Clipkit——基于PIC处理器的高速兼容板；
- Femtoduino——很小巧的一种Arduino兼容板；
- Ruggeduino——内建输入/输出保护的Arduino板；
- Teensy——低价的“nano”型设备。

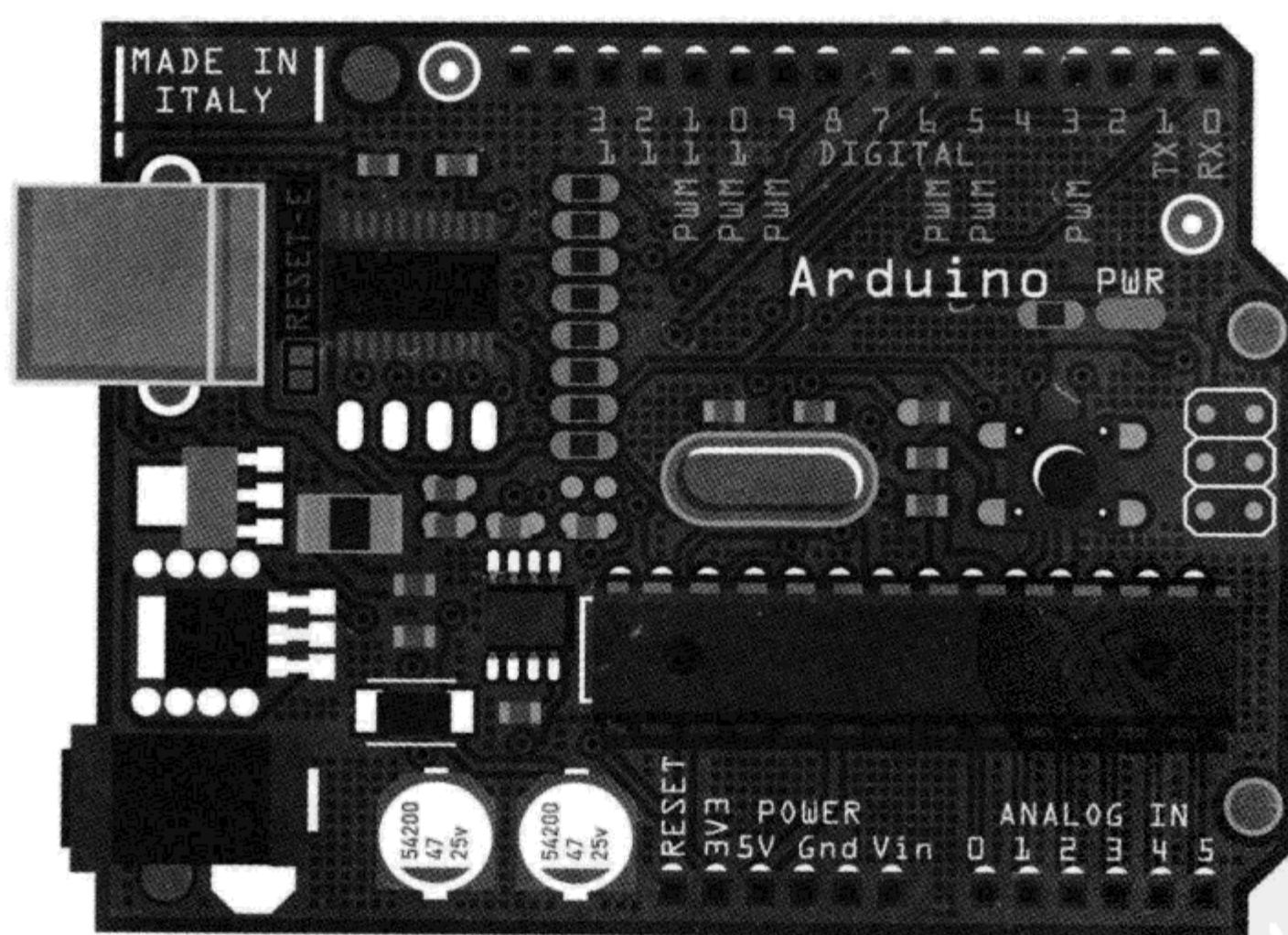
1.5 总结

现在你已经对Arduino硬件有了些许了解，是安装软件的时候了。

第2章

从零开始

Getting Started





介绍Arduino之后，还不了解我们到底是给什么编程，现在到了学习怎样安装我们电脑上所需软件和开始涉及一些代码的时候了。

2.1 开机

当你买到一块Arduino板的时候，它通常都预装了一个让LED闪烁的程序。

被标为“L”的板载LED连接到板上的一个数字接口，即数字针脚13。这是一个用于输出的针脚。当然，这个LED只需要很小的电流，你还可以在这个接口上插别的东西。

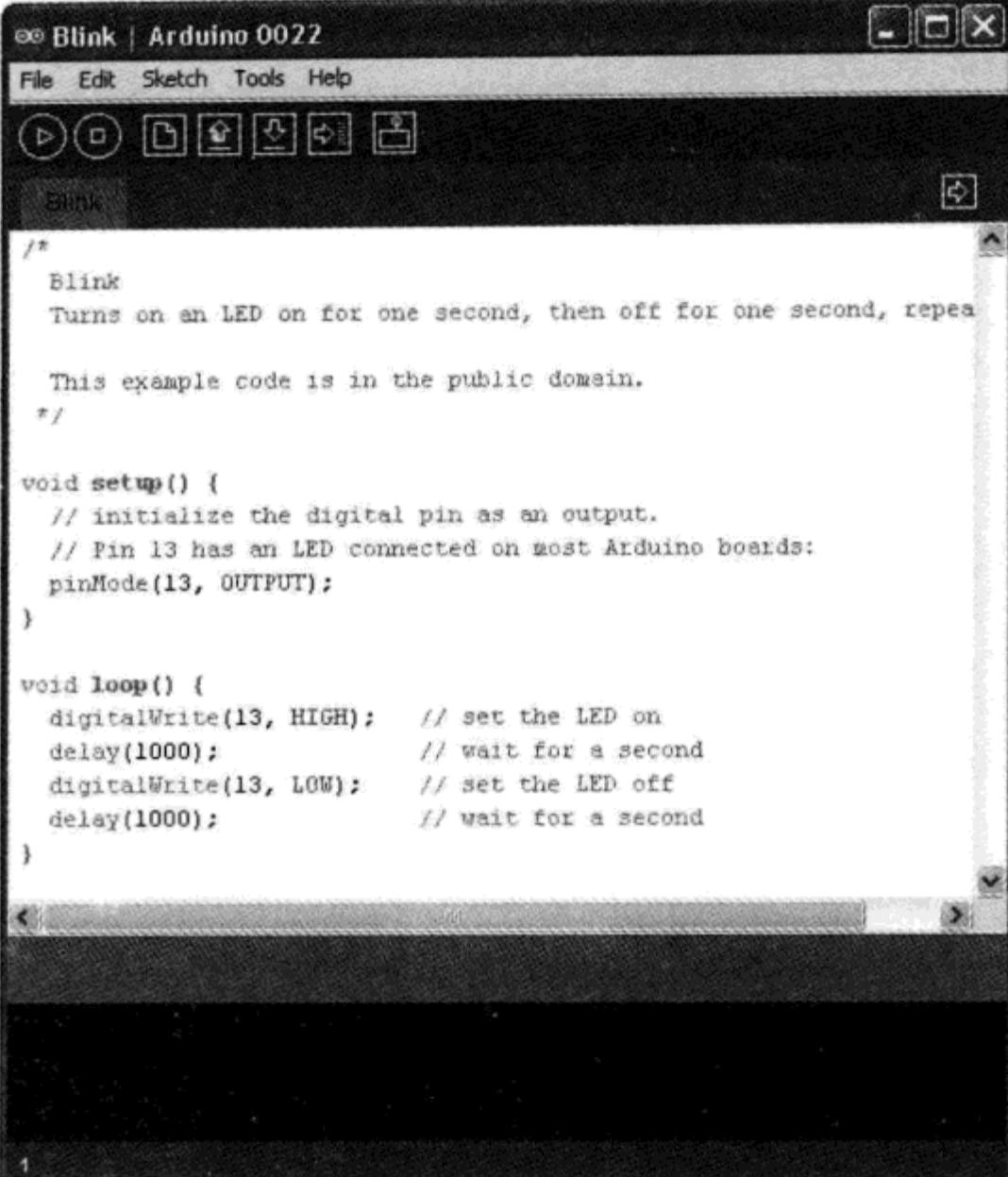
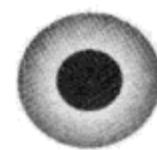
要让你的Arduino开机并运行，只需为它接上电。最简单的办法就是把它用USB连接线插在电脑的USB接口上。这需要一根A-B口的USB连接线，很多打印机用的就是这种线。

如果做得没错的话，LED已经开始闪烁了。新的Arduino板预装这个闪烁Sketch的目的就是让你检测板子是否可以正常工作。

2.2 安装软件

如果想把新的Sketch上传到Arduino，那么要做的就不仅仅是通过USB给其供电了，还需要安装Arduino软件（图2.1）。

Windows、Linux和Mac电脑的软件安装指南，可以在Arduino官方网站（www.Arduino.cc）上找到。根据你的操作系统成功安装USB驱动软件以后，就可以往Arduino板上传程序了。



The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 0022". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for upload, refresh, and other functions. The main code editor window contains the "Blink" sketch. The code is as follows:

```
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeating
 * This example code is in the public domain.
 */

void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);      // set the LED on
    delay(1000);                // wait for a second
    digitalWrite(13, LOW);       // set the LED off
    delay(1000);                // wait for a second
}
```

图2.1 Arduino应用程序

2.3 上传你的第一个Sketch

LED闪烁程序就相当于学习其他语言时候的“hello world”，是学习一个新语言时的第一个程序。让我们通过把它安装到Arduino并修改它来试验试验。

开始用电脑进行Arduino应用编程时，要从新建一个空的Sketch开始。幸运的是，软件中包含了很多有用的范例，所以我们从文件菜单中打开图2.2所示闪烁程序范例。

接下来，将Sketch上传到Arduino板。用USB线将Arduino板

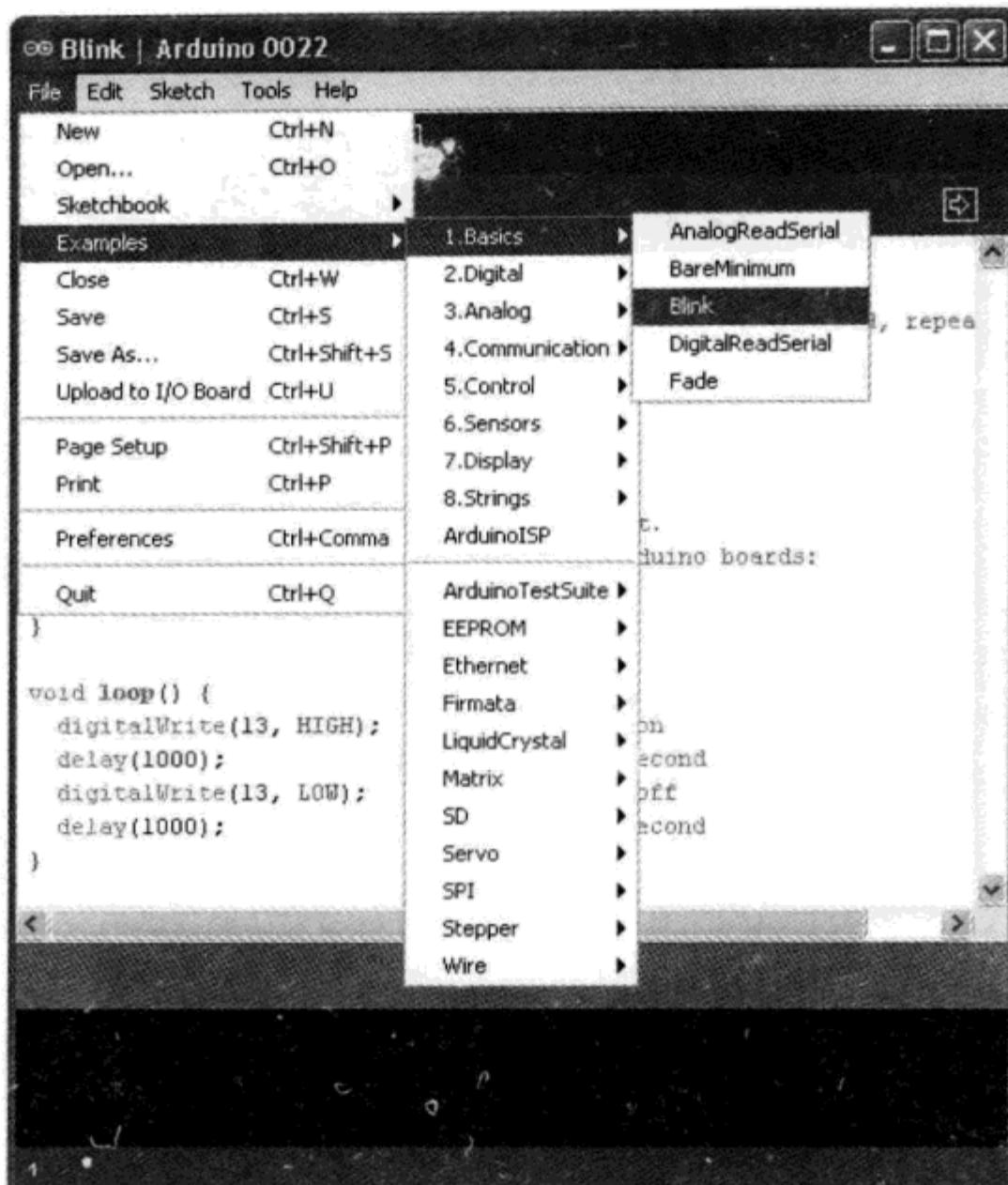


图2.2 闪烁程序Sketch

连接到电脑，你应该可以看到板上绿色的“ON”指示灯亮了起来，板载 LED 应该已经在闪烁了，因为大部分的板都预装了闪烁程序。没关系，让我们再安装一遍并修改一下这个程序。

如果你用的是 Mac 电脑，刚接上板子时你会看到一个提示“一个新的网络接口被检测到”，点“取消”就可以了。这是因为你的 Mac 把 UNO 板认作了一个 USB 调制解调器。上传 Sketch 之前，你必须告诉软件你现在用的是哪一类 Arduino 板，以及连接的是哪一个串口。图 2.3 和 2.4 告诉你怎样在“Tools”菜单中设置这些。

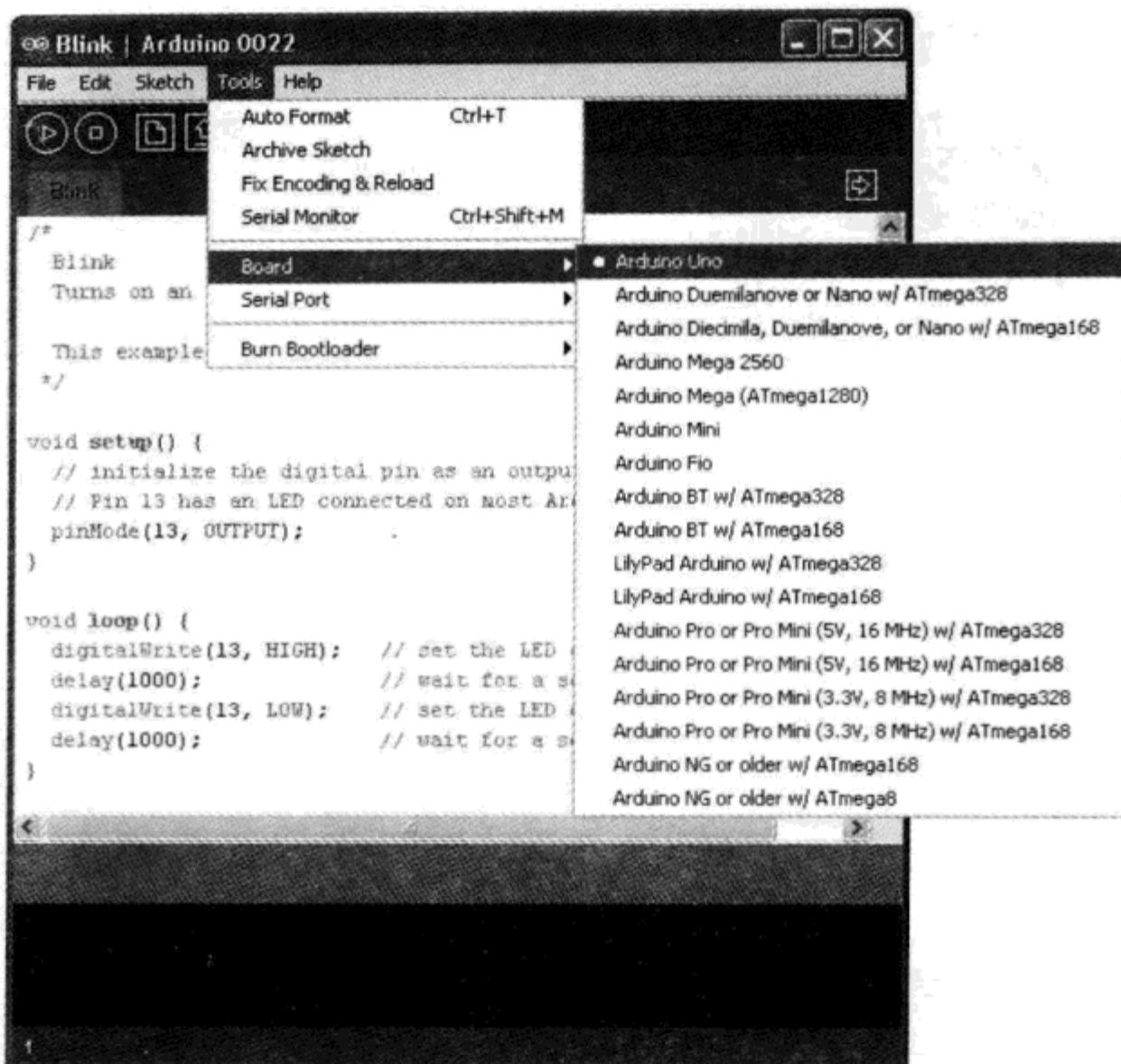


图2.3 选择板类型

在使用Windows操作系统的时候，串口总是COM3。在Mac和Linux系统下，你会看到一个很长的串口设备列表（图2.5）。要选的一般是列表里最顶端的那个，名字类似“/dev/tty.usbmodem621”。

现在，点击工具栏里的“上传”按钮。它在图2.6中被高亮显示。

在点击“上传”之后、开始上传之前，会因为编译Sketch而产生一个小的停顿。如果一切正常，Sketch上传时LED会一阵狂闪，完了之后你会在Arduino软件窗口的底部看到“Done uploading”和一个类似“Binary sketch size: 1018 bytes(of a 14336 byte maximum)”的消息。

一旦上传完成，板子就会自动开始运行Sketch，你应该可以

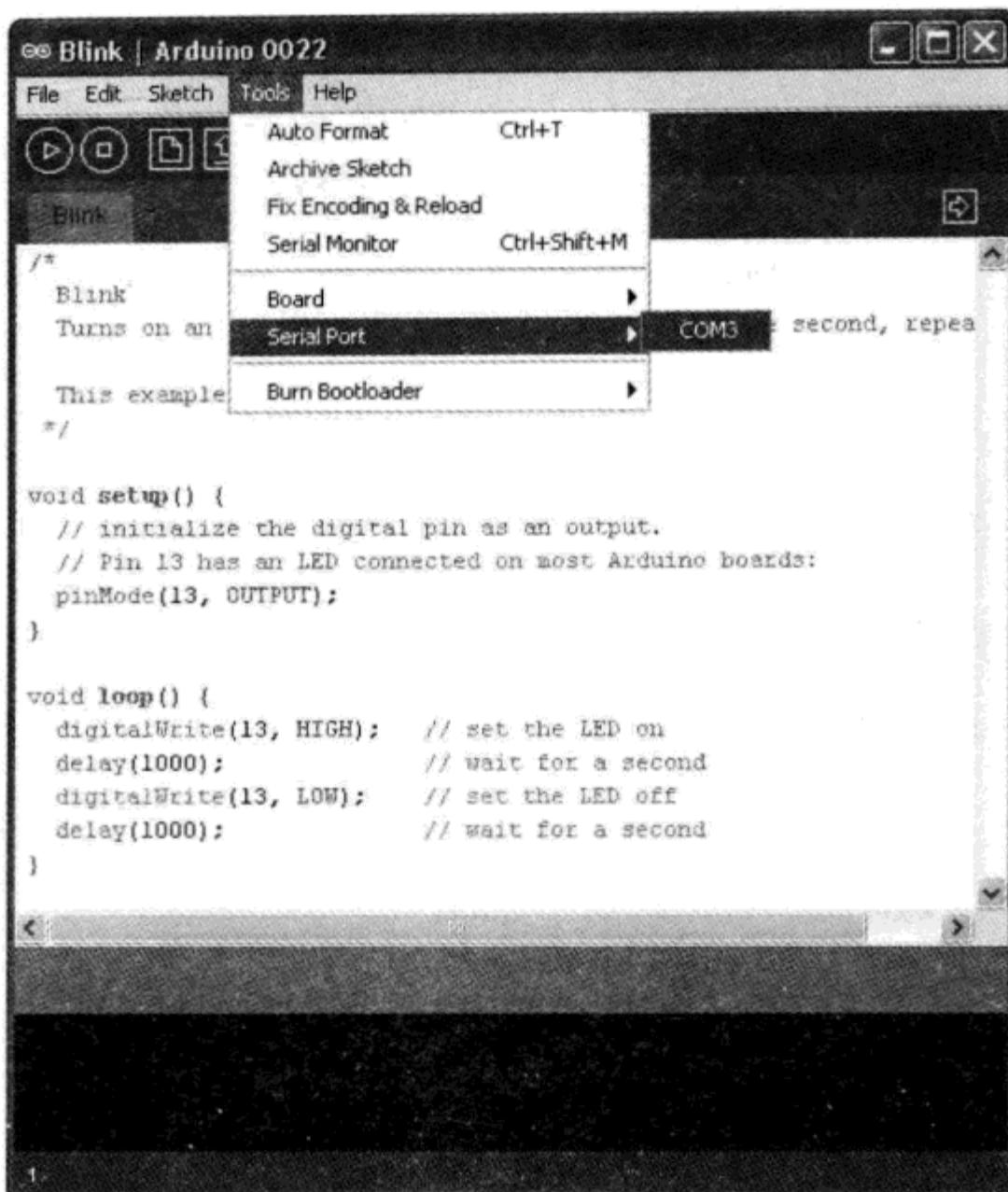


图2.4 选择串口（Windows下）

看到LED开始闪烁。

如果没有闪烁，请检查一下你的串口和板类型选择是否正确。

现在，让我们修改一下Sketch，让LED闪得更快一些。我们将Sketch中2个1000ms的延时值改为500ms。图2.7中用高亮标出了需要修改的地方。

再次点击“上传”按钮。上传完成后你会发现，LED的闪烁速度是原来的2倍。

恭喜你，现在你已经准备好开始为你的Arduino编程了。不过，我们还是先稍微了解一下Arduino软件。

Getting Started

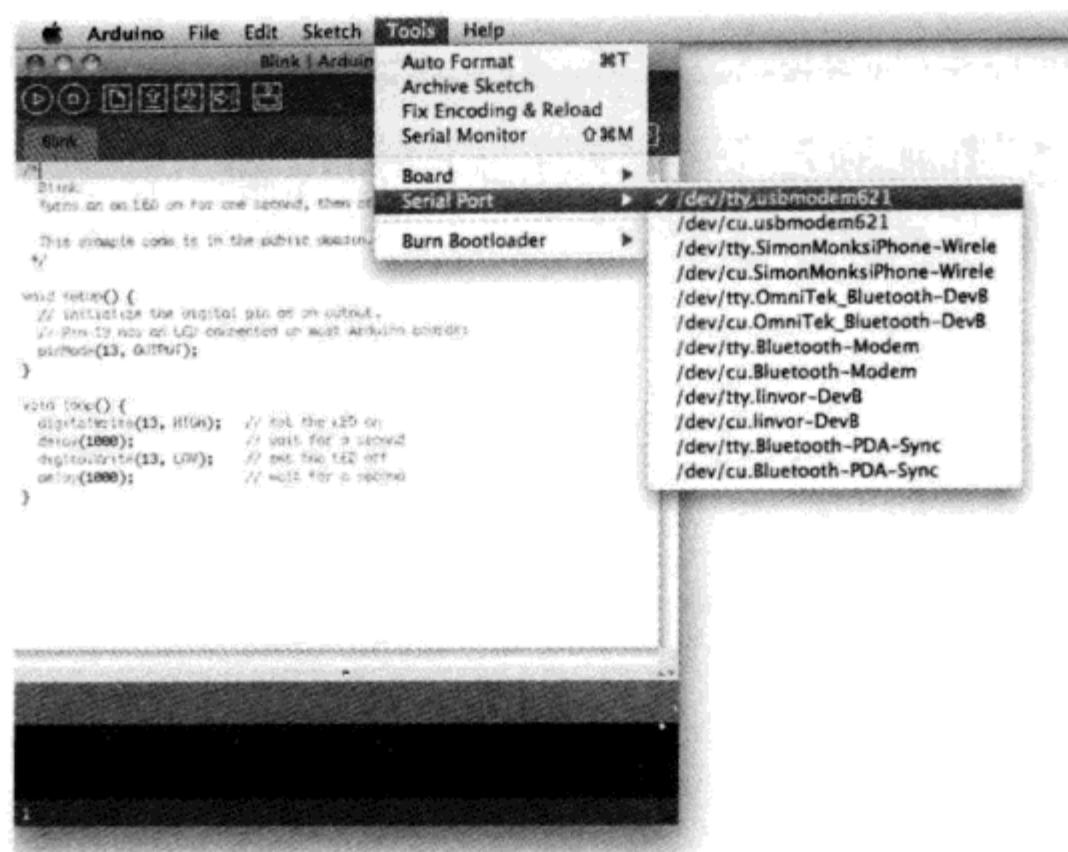


图2.5 串口选择（Mac系统）

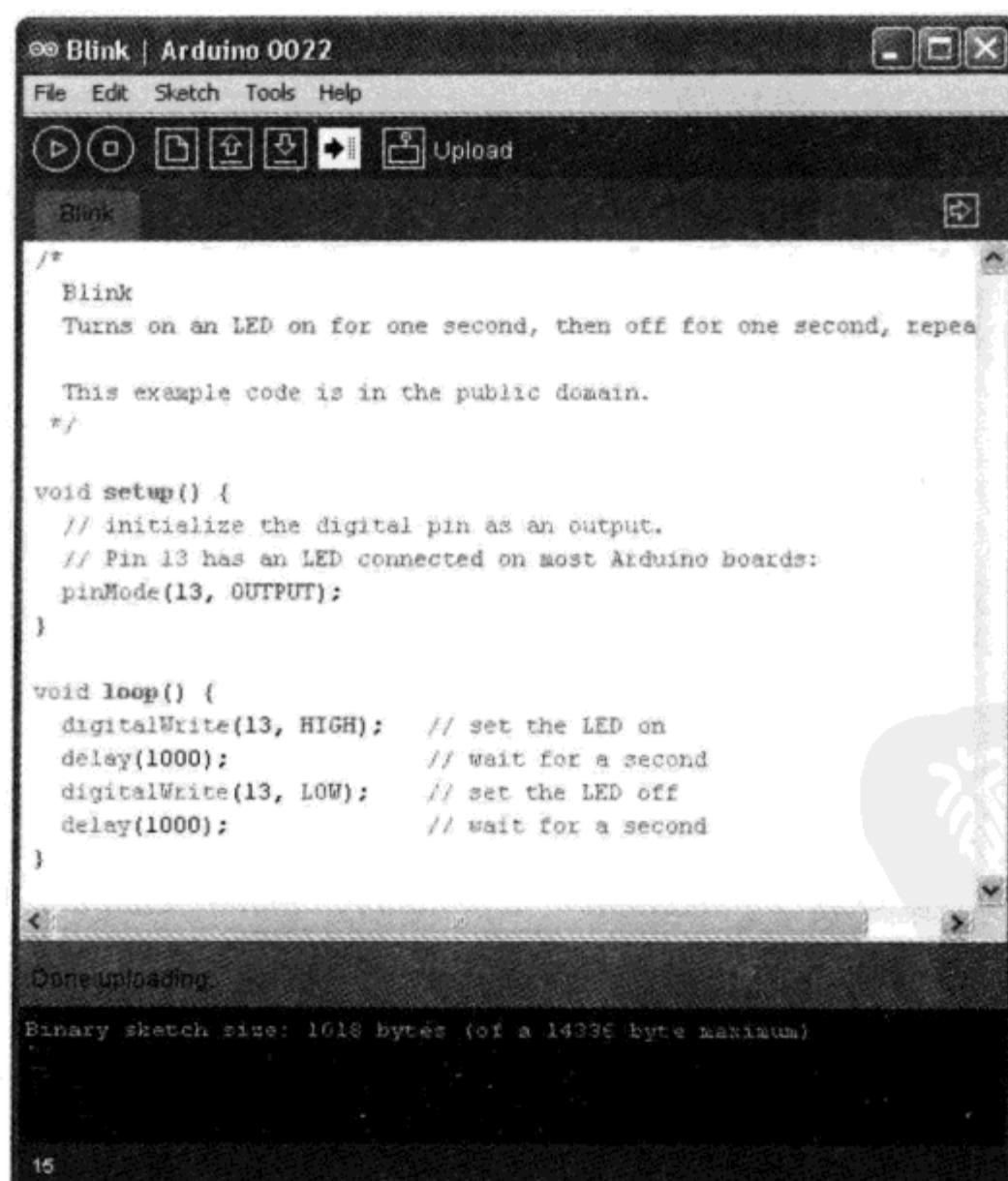


图2.6 上传Sketch



```
/*  
 * Blink  
 * Turns on an LED on for one second, then off for one second, repeating.  
 * This example code is in the public domain.  
 */  
  
void setup() {  
    // initialize the digital pin as an output.  
    // Pin 13 has an LED connected on most Arduino boards:  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH);    // set the LED on  
    delay(500);              // wait for half a second  
    digitalWrite(13, LOW);    // set the LED off  
    delay(500);              // wait for half a second  
}
```

Done uploading
Binary sketch size: 1010 bytes (of a 14336 byte maximum)

图2.7 修改闪烁Sketch

2.4 Arduino应用程序

Arduino中的Sketch就像是文字处理软件中的文档，你可以打开它们并拷贝其中一个的一部分到另一个中去。所以你可以在“File”菜单中看到“Open”、“Save”和“Save As”这些选项。人们通常不会直接使用“Open”，因为Arduino软件有一个“Sketchbook”功能。在“Sketchbook”中，所有的Sketch都被准确地分类保存在各个文件夹中。你可以从“File”菜单中获取



“Sketchbook”的路径。由于你才刚刚装好Arduino软件，所以你的“Sketchbook”是空的。

正如前面看到的，软件内带有一些Sketch范例，这非常有用。前面我们修改过闪烁LED的Sketch范例，如果你试图保存它，会有一个对话框显示“某些文件被标为只读，你需要将此Sketch保存至其他位置”。

如图2.8那样，试试使用原来的位置，但是请将文件名修改为“MyBlink”。

现在，如果你在“File”菜单中点击Sketch，就会看到列表中有“MyBlink”了。如果你想在电脑的文件系统中找到它，PC里请查找“Mydocuments\Arduino”，Mac和Linux电脑里它则被保存在“Documents/Arduino”中。

本书中用到的所有Sketch都可以从www.arduinoobook.com下载到zip文件（Programming_Arduino.zip），建议现在就把这个文件下载并解压缩到放置Sketch的“Arduino”文件夹中。这意味着，解压缩之后你的“Arduino”文件夹中就会有2个子文件夹了：一个是刚才保存的“MyBlink”文件夹，另一个则是名为“Programming Arduino”的文件夹（图2.9）。“Programming Arduino”文件夹中包含了所有的Sketch，并且根据本书章节编号，如“sketch_03_01”就是第3章的第一个Sketch。

这些Sketch需要退出并重启软件才会出现在你的“Sketchbook”中，之后你的“Sketchbook”应该看起来和图2.10一样。

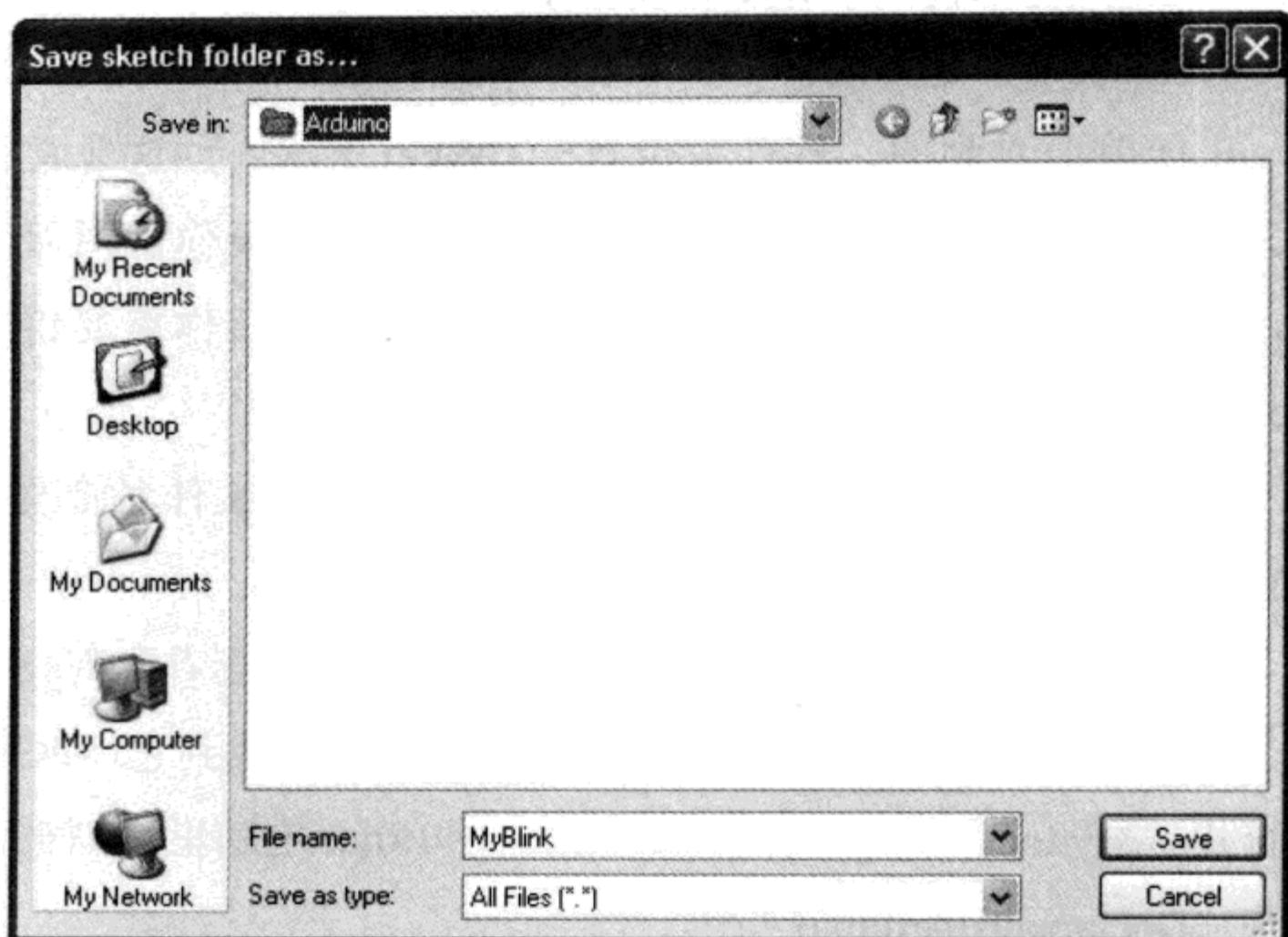


图2.8 保存一个“MyBlink”的副本

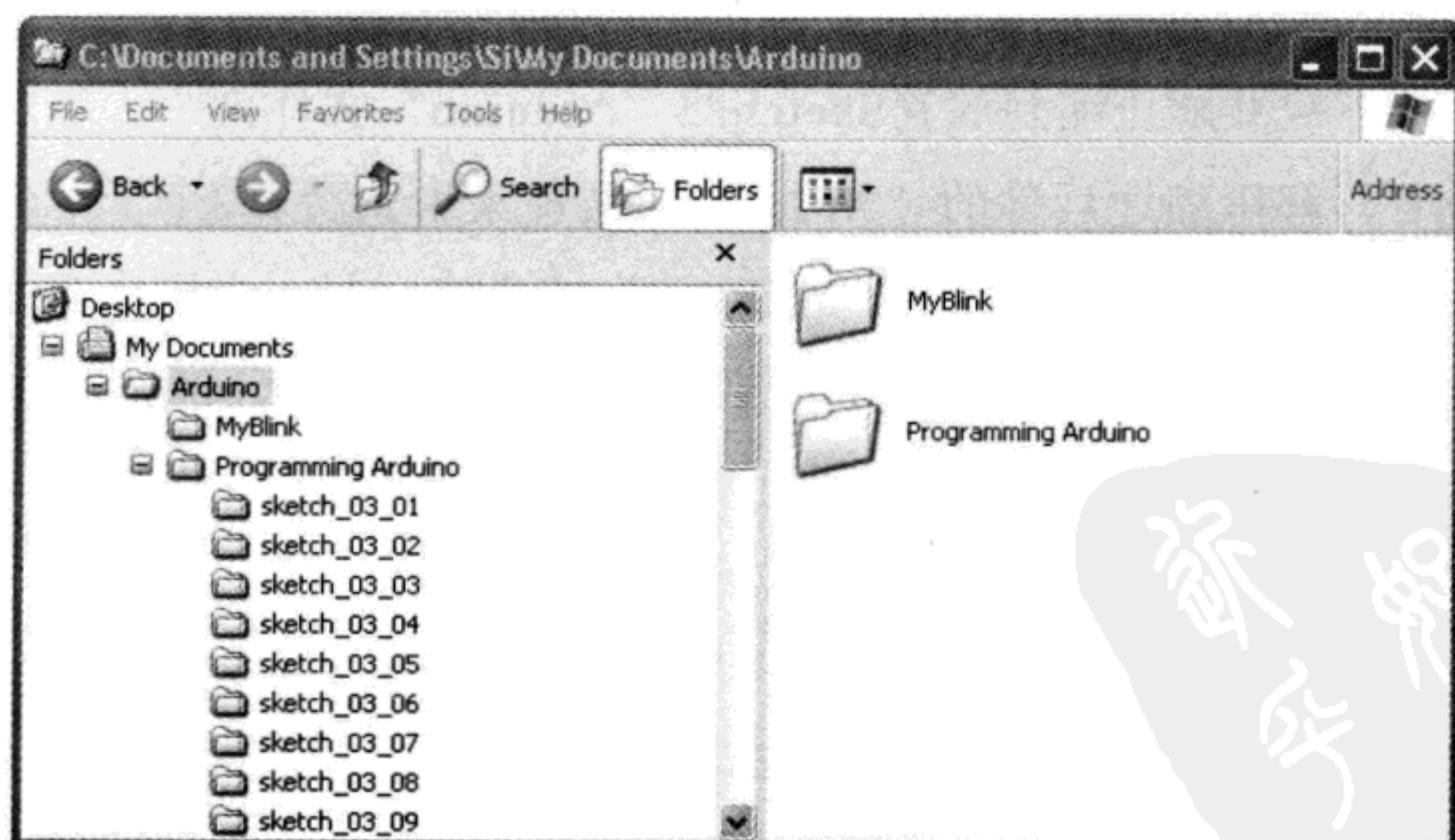


图2.9 添加书中的Sketch范例

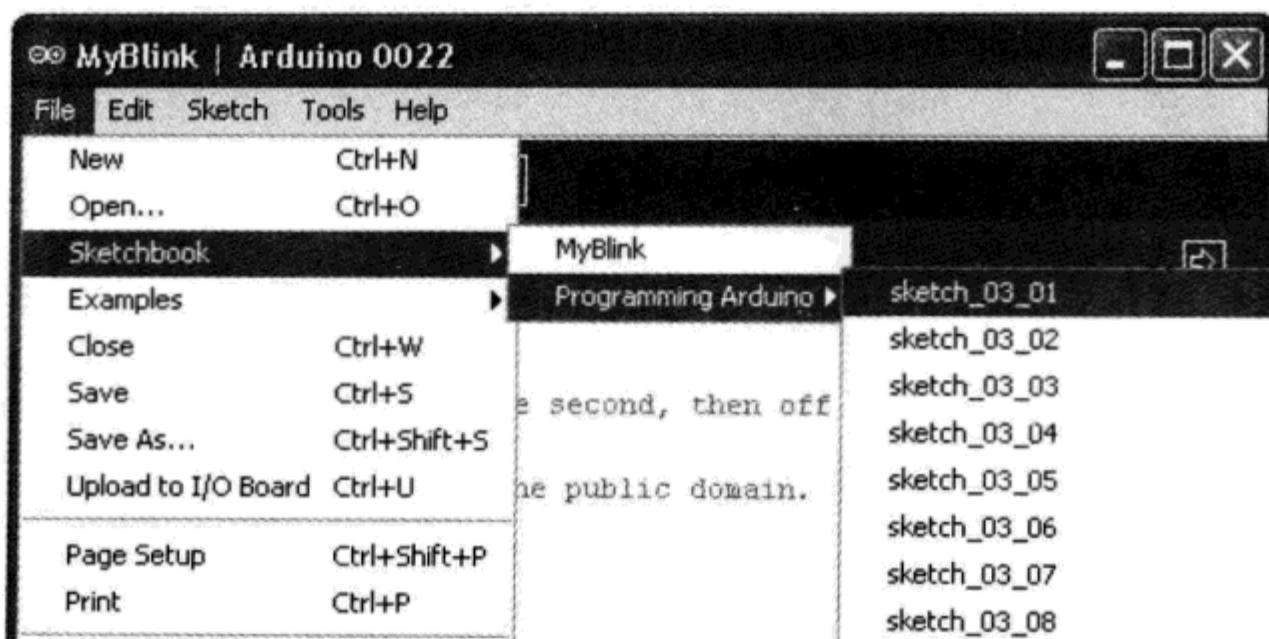


图2.10 添加过书中Sketch范例的“Sketchbook”

2.5 总 结

现在你的准备工作已经完成，可以开始了。

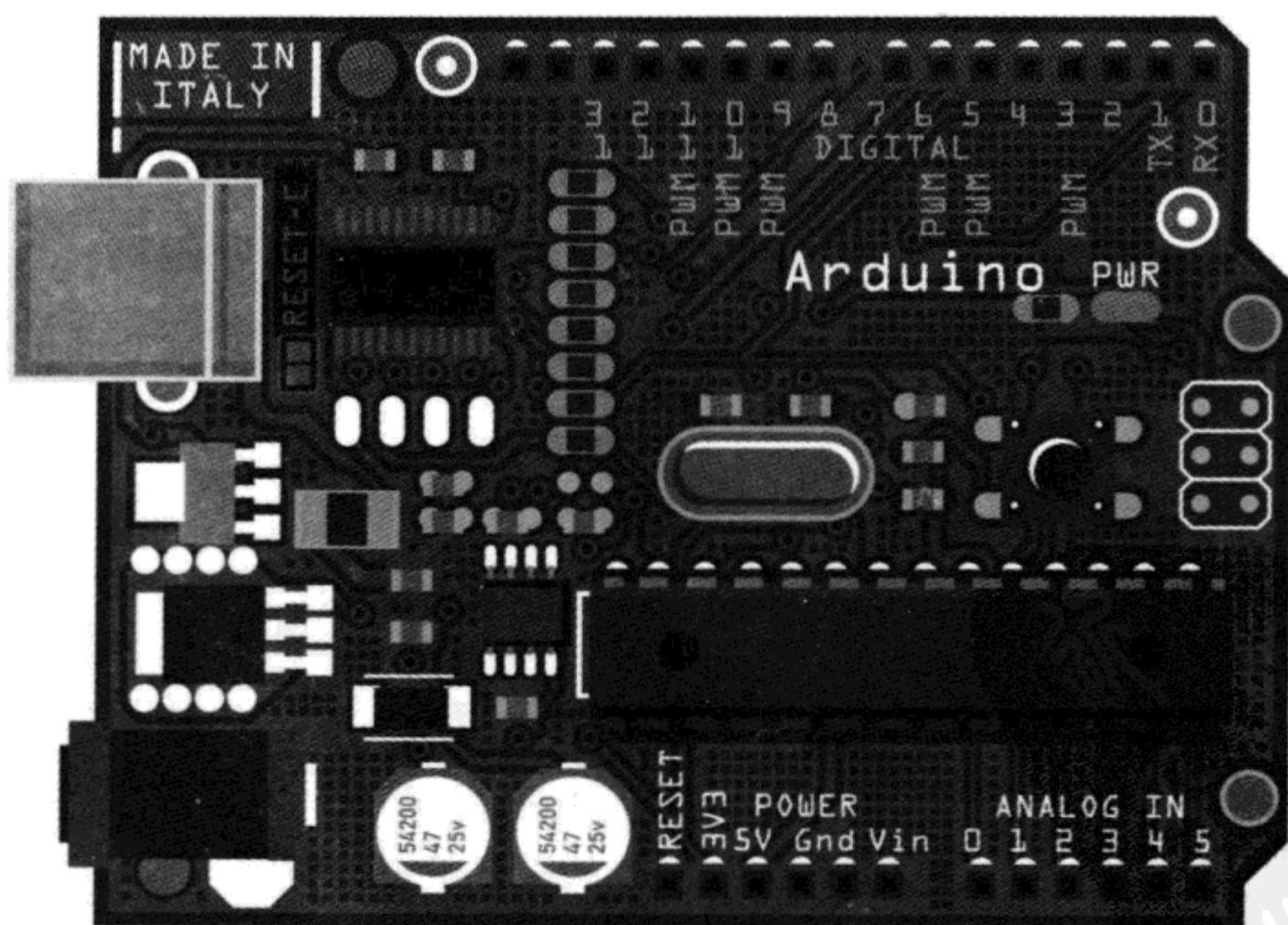
下一章中，我们将学一些Arduino中使用的C语言基本原则，并开始写一些代码了。



第3章

C语言基础

C Language Basics





用来给Arduino编程的是一种叫做C语言的计算机语言，在本章中，你将开始学习C语言的基础。你将会像一个Arduino程序员一样在你开发的每一个Sketch中用到本章所学，想玩转Arduino的话，最好打好这些基础。

3.1 编 程

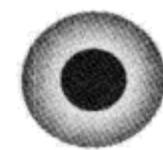
人们会说一种以上的语言，这很常见。实际上，你学过的语言越多，再学习一门新的语言的时候就会越简单，因为你会发现语法和词汇有很多共通之处。计算机语言也是如此。所以，如果你学习过其他计算机语言，学起C语言来也会很容易。

幸运的是，计算机语言的“词汇量”比真正的语言少多了。而且因为你只需要“写”，而并不需要“说”，所以可以随时查词典。还有，计算机语言的语法和句法规律性很强，当你一旦掌握了一些简单的概念后，就会学得很快。

最好将程序或者Sketch（Arduino中的程序就叫做Sketch）看做是按照书写顺序排好的指令列表。例如，你要写下面的代码：

```
digitalWrite(13, HIGH);  
delay(500);  
digitalWrite(13, LOW);
```

这3行代码每行分别有各自的用处。第1行将输出端子13设为高电平，它是一个连有Arduino板载LED的端子，所以此时LED会被点亮。第2行只是等待500ms的时间，什么也不做。然后，第3行关闭LED。所以，这3行代码可以做到让LED闪烁1次。



你已经看到了令人眼花缭乱的标点使用方式，并且单词之间没有空格。让新手们产生挫败感的是：“我知道我要做什么，但我不知道我应该怎么写。”别怕，一切都会给出解释的。

首先，我们先来看看标点和单词的组合方式。这两者都是计算机语言的语法规规定。大部分语言要求你在语法上必须精准。主要规则之一是给某个东西命名的时候只能用一个单词，也就是说单词之间不能有空格。例如，`digitalWrite`是某个东西（一个内建函数）的名字（后面你将会学到更多关于函数的知识），它可以完成设定Arduino板上一个输出端子的工作。你不仅要避免在名称中使用空格，还要注意名称是区分大小写的，所以你必须写成`digitalWrite`，而不是`DigitalWrite`或者`Digitalwrite`。

函数`digitalWrite`需要知道设置哪个针脚，并且要知道设为高电平还是低电平。这两个信息被称为参数，它们在需要时被递交给函数。函数的参数必须被放在括号中，并且在两个参数之间用逗号隔开。

通常是在函数名后面直接接括号，并在逗号和下一个参数之间加上一个空格。然而，如果你想在括号内多加一些空格也是没有问题的。如果函数只有一个参数，那当然没有必要用逗号。

注意每行结尾处的分号。如果那是句号的话，逻辑上似乎更说得通，因为分号表示一条指令的结束，有点像是一句话的结尾。

在下一节中，你将学习到当你在Arduino集成开发环境(IDE)中按下“上传”按钮的时候都发生了什么，之后你就就可以开始尝试一些范例了。



3.2 什么是计算机语言

进入第3章都还没告诉大家到底什么是计算机语言，可能有点不正常。我们现在已经认识了Arduino Sketch并且可能还大概知道它是干什么用的，我们还要更深入地了解程序代码是怎么用一些单词做到诸如控制LED开关一类的事情的。

图3.1概括了从在Arduino IDE中输入代码到在板上运行Sketch的全过程。

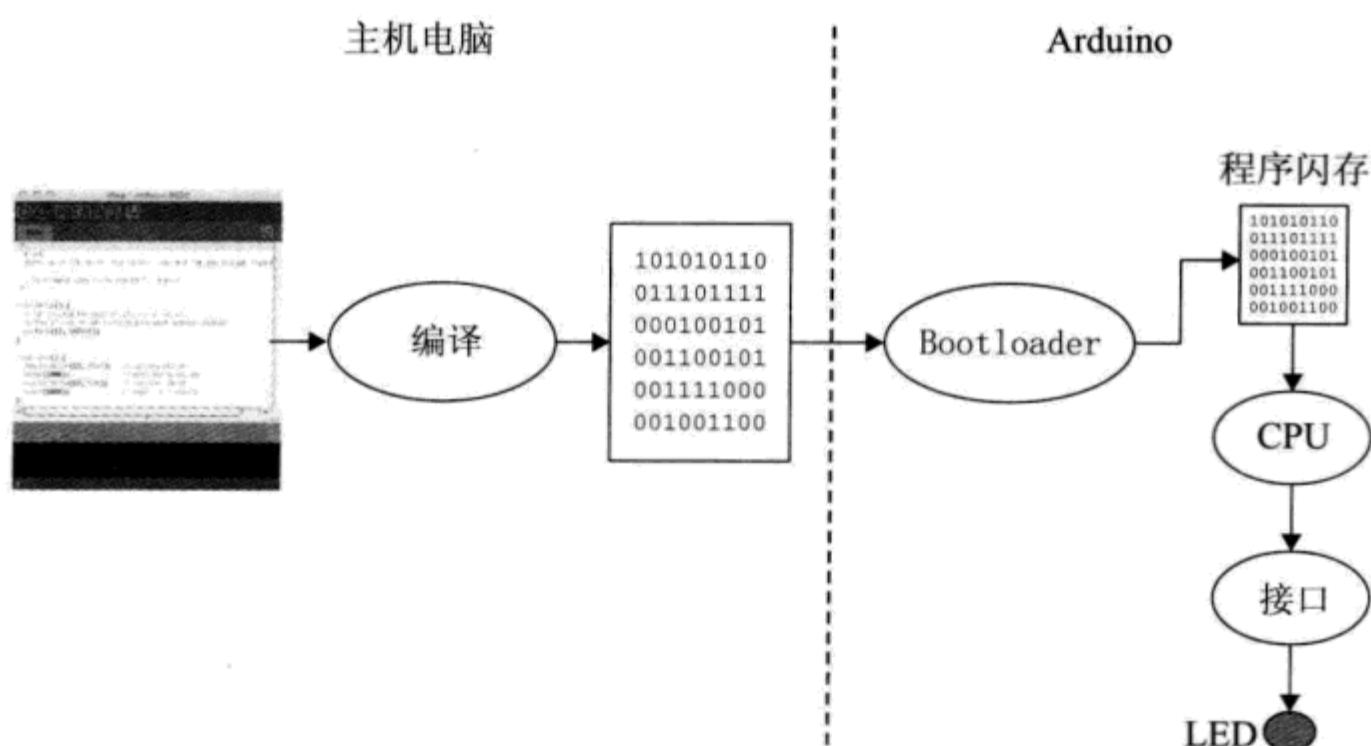
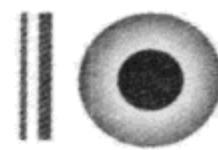


图3.1 从代码到板

当你按下Arduino IDE中的“上传”按钮时，将展开一连串的事件，致使你的Sketch被安装到Arduino上并运行。这可不像直接把你软件里输入的文字拿出来，然后放到Arduino里那么简单。

第一步是做一件叫做编译的事情。它把你写出的代码编译成机器码——Arduino可以理解的二进制语言。你点一下Arduino IDE中的三角形“验证”按钮，这实际上就是在尝试编译你写下的C语言，而并不把它上传到Arduino。编译代码的一个附加作用



是检查代码并确保它符合C语言的规则。

如果你在Arduino IDE中输入“Ciao Bella”并点击“运行”按钮，结果会如图3.2所示。

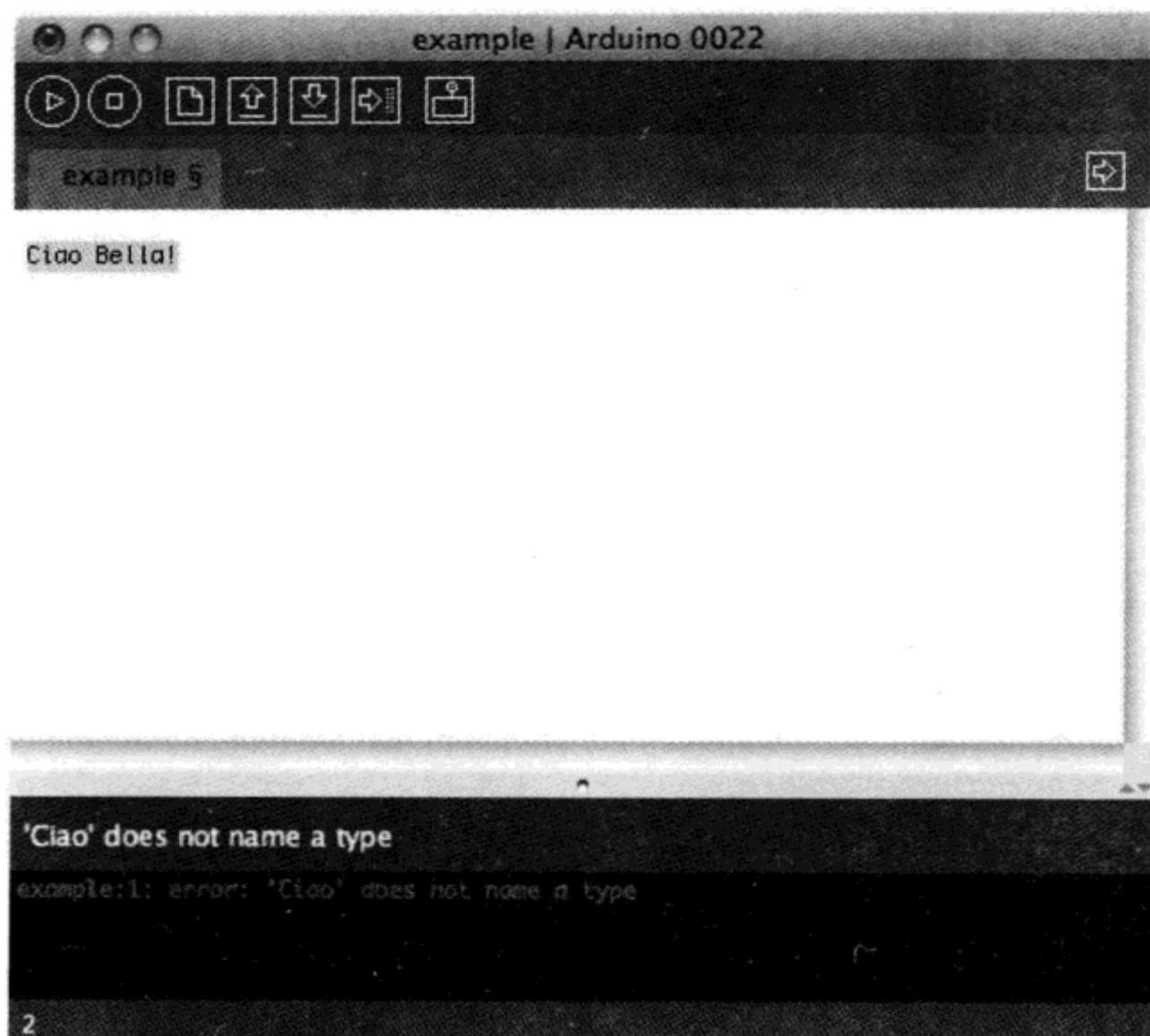


图3.2 Arduino不懂意大利语

Arduino尝试编译“Ciao Bella”，尽管这是一句意大利语（Arduino源自意大利），Arduino还是不明白你在说什么。因为这并不是C语言，所以结果是我们收到屏幕底部的提示：“error: Ciao does not name a type”——“Ciao”是一个未定义的数据类型，这意味着你写的东西出错了。

让我们试试另外一个例子，这次我们试试编译一个空的Sketch，如图3.3所示。

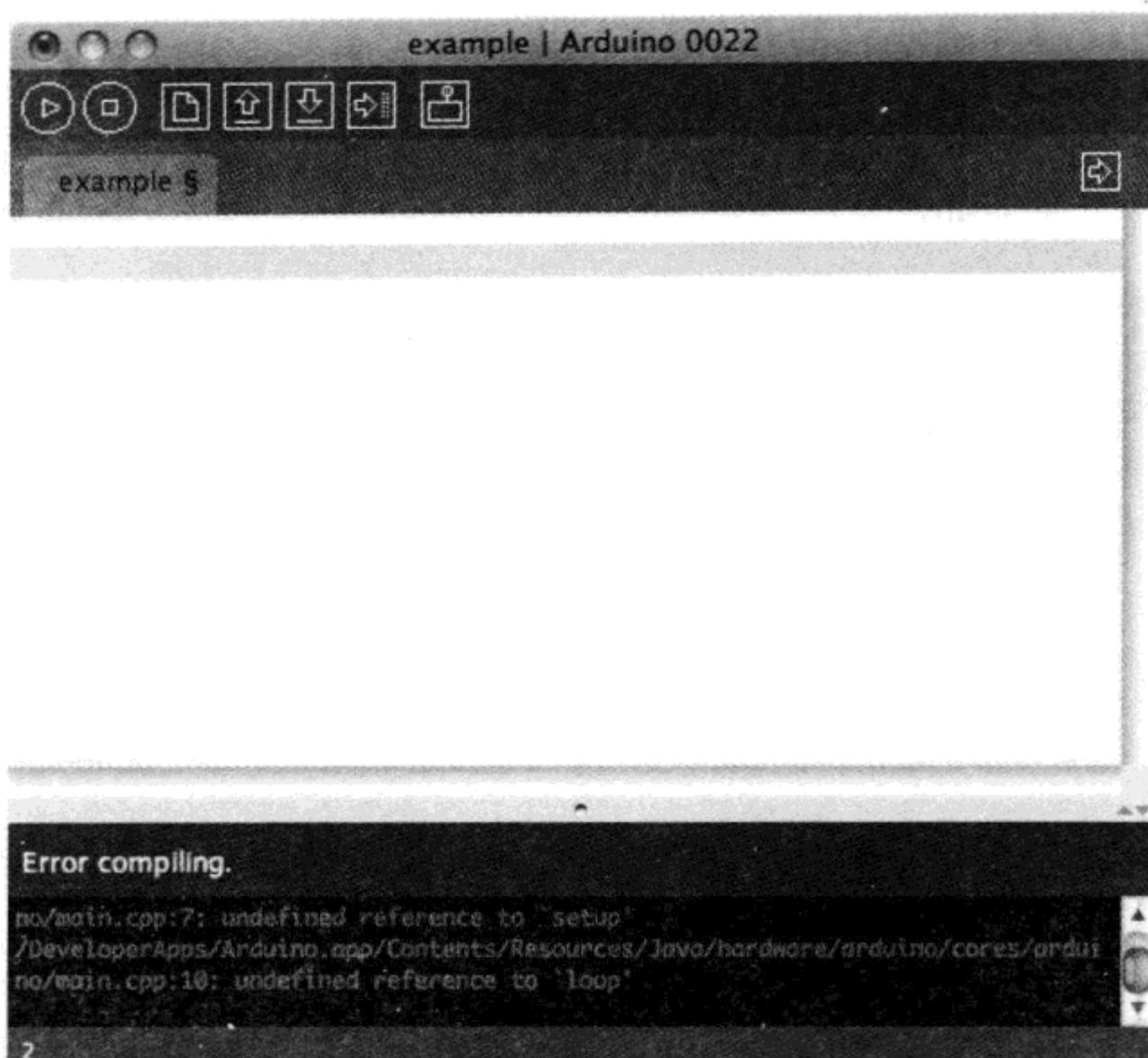


图3.3 找不到setup和loop

这一次，编译器告诉你：你的Sketch中缺少setup和loop函数。正如前面第2章中看到的LED闪烁Sketch范例一样，你必须有一些“固定语句”在代码中。为什么说是固定的呢？在你往Sketch中写代码之前，在Arduino程序中“固定语句”就是setup和loop两个函数必须作为一种格式一直在Sketch中出现。

关于函数在本书后面你将会学到更多的相关知识，但现在我们姑且用一下这些“固定语句”，好让你的Sketch可以通过编译（图3.4）。

Arduino IDE看到了你在写代码上作出的努力，并且接受了代码，因为它告诉你“编译完成”并且报告了Sketch的大小：450字

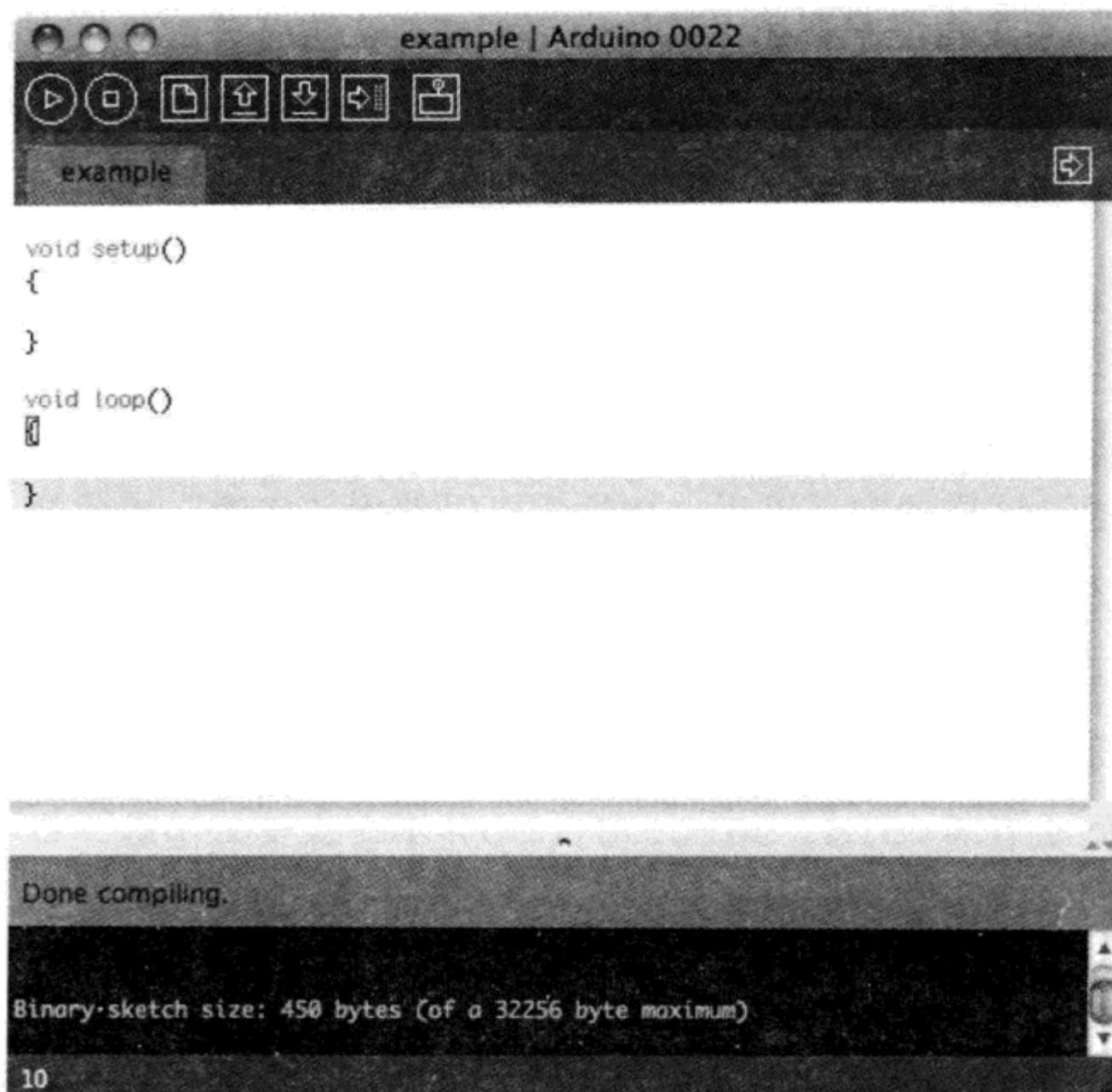
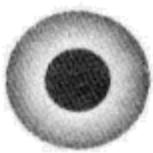


图3.4 可以编译的Sketch

节。IDE同时也告诉你：你的Sketch最大不能超过32256字节。所以，你还有很大的空间，可以将Sketch写得更大。

`void setup()`这一行的意思是，你正在定义一个名为 `setup` 的函数。在Arduino中，部分函数是已经定义好了的，就像是 `digitalWrite` 和 `delay`，其他的你可以并且必须由自己来定义。`setup` 和 `loop` 是你在所写的每一个Sketch中都必须自己定义的函数。

有一件很重要的事情需要理解，这里并不像调用 `digitalWrite` 一样来调用 `setup` 和 `loop`，你只是创建了这两个函数好让



Arduino系统自己来调用。这是一个挺难理解的概念，但你可以把它想象为法律文书中的“定义”部分。

大部分的法律文书中都有一个“定义”部分，若有的话可能为如下的内容：

定义

作者：对创作本书负责的个人或者群体

通过这样的定义以后，律师们可以使用“作者”来代替“对创作本书负责的个人或群体”来使他们的文书变得更短、更好读。函数就像这些定义一样。定义一个函数之后，你或系统都可以在你的Sketch的其他地方使用。

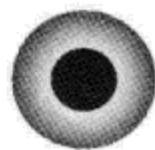
回头看看void。setup和loop并不会像某些函数一样有返回值，所以你必须使用关键词void来说明它们是“空”的。

假设你使用一个名为sin的函数，正如函数名一样，其作用是计算一个角度的sin值。这个函数会返回一个值——调用它时给出的参数的sin值。

就像在法律文书中我们使用一个词来定义某个条款一样，在C语言中我们用函数，而这个函数可以被C语言调用。

在关键词void之后跟着的是函数名和用来放置参数的括号。这个例子中没有参数，但是我们仍然要在这里写上括号，因为我们是在定义一个函数而不是在调用它。所以我们得讨论一下：如果函数真的被调用，会发生些什么。

当函数被调用时，所发生的事情必须被放在大括号中，大括号和被放在其中的代码被合称为代码块，这个我们在后面还会提到。



注意：虽然你必须定义setup和loop这两个函数，但并不一定要在程序中写些什么，只是没代码的Sketch就有些无聊了。

3.3 再闪烁一次

Arduino需要setup和loop这两个函数的原因是，当Arduino开始运行Sketch时从那些需要持续发生的事件中区分出只需要被运行一次的东西。

函数setup会在Sketch开始的时候只运行一次。让我们加入一些可以让板载LED闪烁的代码，让程序看起来像下面一样，然后将它们上传到板子上。

```
void setup()
{
    pinMode(13, OUTPUT);
    digitalWrite(13, HIGH);
}

void loop()
```

函数setup本身调用另外两个内建函数，pinMode和digitalWrite。你已经了解过digitalWrite，但是pinMode是一个新的函数。函数pinMode用来将一个指定针设置为输入或者输出。所以点亮LED的过程其实有两步。第一步，你必须先把针脚13设置为输出。第二步，将输出设为高电平。

当你运行这个Sketch的时候，你会看到板载LED被点亮，并



且一直亮着。这并不让人兴奋，所以让我们至少尝试下用loop函数不断地开关LED来让它闪烁。

你可以继续用setup函数来调用pinMode，因为它只需要被调用一次。当然，你也可以把它放到loop中，这样也不影响程序的运行，只是没有必要，而且只需要做一次的事情我们只做一次是一个很好的编程习惯。那么，如下修改你的Sketch：

```
void setup()
{
    pinMode(13, OUTPUT);
}

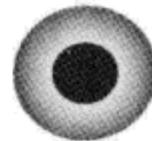
void loop()
{
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
}
```

运行它，看看发生了什么。可能结果并不是你想象的那样，LED基本上还是在常亮。为什么会这样呢？

让我们一步一步地再看一下：

- (1) 运行setup并将针脚13设为输出；
- (2) 运行loop并将针脚13的输出设为高电平；
- (3) 延迟0.5s；
- (4) 把针脚13设置为低电平；
- (5) 再次运行loop，回到第(2)步，设置针脚13为高电平。

问题出在第(4)步和第(5)步之间：LED在被关闭之后的一



瞬间就立刻被点亮了。这个过程发生太快，以至于LED看起来就是常亮的。

Arduino板上的单片机每秒可以处理 16×10^6 条指令，虽然并不是 16×10^6 条C语言命令，但也已经相当快了，我们的LED只会被关闭百万分之几秒而已。为了改正这个错误，你需要在关闭LED之后再加上一个延迟，代码看起来应该是这样：

```
// sketch 3-01
void setup()
{
    pinMode(13, OUTPUT);
}
void loop()
{
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
}
```

再试一次，LED现在已经应该以每秒1次的速度愉快地闪烁了。

你可能注意到了，代码顶部的注释是“sketch 3-01”。为了让你少敲点键盘，我们把所有的Sketch都放在了本书的网站上，并且顶部都有这样的注释，你可以从www.arduinoobook.com下载它们。

3.4 变量

在LED闪烁的例子中，代码中的3个地方都用到了针脚13（Pin13）。如果你决定使用别的针脚，那就需要在相应的3个地



方都修改代码。类似地，如果你想改变由delay控制的闪烁频率，你也需要在不止一个地方将数字500改为其他数值。

变量可以认为是给一个数值命名，实际上比这点要实用得多，但现阶段你只会因此而使用变量。

当在C语言中定义一个变量的时候，你必须指定变量的类型。我们想让我们的变量全都是数字，这在C语言中被称为整数(int)，那么若要定义名为ledPin的变量值为13，你应该这样写：

```
int ledPin 13;
```

注意：ledPin是一个名称，和函数命名的规则一样，中间是不能有空格的。约定写法是变量名的首字母小写，中间每个单词的首字母都大写。程序员一般把这种方式称为小鹿拼写法(pumpy case)或者骆驼拼写法(camel case)。让我们把变量加入到LED闪烁Sketch：

```
// sketch 3-02
int ledPin = 13;
int delayPeriod = 500;
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
```



```
digitalWrite(ledPin, LOW);  
delay(delayPeriod);  
}
```

我们在这里还使用了一个叫delayPeriod的变量。

Sketch里原本所有提到13的地方现在都替换为ledPin，原本提到500的地方现在也都被替换成了delayPeriod。

如果你想让LED闪烁得更快，只需要在一个地方修改变量delayPeriod的值就可以了。现在改成100，再运行一次试试。

你还可以对变量做一些其他有趣的事情。让我们来修改Sketch，使LED开始闪得很快，然后逐渐越闪越慢。想达到这个效果，你所要做的就是在每次闪烁的时候往变量delayPeriod上加上点东西。

像下面一样在loop函数结束的地方加上一行代码，然后在Arduino板上运行这个Sketch。按下重置按钮，看看是不是又重新开始快速闪烁了。

```
sketch 3-03  
int ledPin = 13;  
int delayPeriod = 100;  
  
void setup()  
{  
    pinMode(ledPin, OUTPUT);  
}  
  
void loop()  
{  
    digitalWrite(ledPin, HIGH);  
    delay(delayPeriod);  
    digitalWrite(ledPin, LOW);  
    delay(delayPeriod * 2);  
    delayPeriod++;  
}
```

PDG



```
delay(delayPeriod);
digitalWrite(ledPin, LOW);
delay(delayPeriod);
delayPeriod = delayPeriod + 100;
}
```

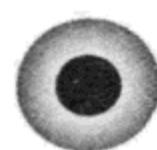
这是你的Arduino在做算术。每当loop被调用的时候，它都会正常闪烁一下LED，但结束时会在变量delayPeriod的值上加上100。我们稍后再来讨论算术，但首先你需要用比让LED闪烁更好的方法来看看Arduino到底能干些什么。

3.5 C语言实验

这是一个做C语言实验的方法——把你要测试的C语言语句放到setup函数中，在Arduino上运行它，然后让Arduino把输出的结果显示在串口监视器上（图3.5、图3.6）。

串口监视器是Arduino IDE的一个组件，你可以通过点击工具栏最右边的图标打开它。它的作用是让电脑和Arduino之间通信，你可以在文本输入区输入一条信息，按回车键或者点击发送，它将会把信息发送到Arduino。同时，如果Arduino有什么想说的，同样也会在串口监视器中显示出来。在这两个情况下，信息都是通过USB线传输的。

正如你所期待的，有一个内建函数可以在Sketch中用来往串口监视器返回消息。这就是serial.printIn。而且它只需要一个参数，此参数构成了你想要发送的消息。这个消息通常是一个变量。



The screenshot shows the Arduino IDE interface. The title bar says "sketch_03_fig5 | Arduino 0022". Below the title bar is a toolbar with various icons. The main area contains the C++ code for the sketch:

```
// sketch 03-fig 5

void setup()
{
    Serial.begin(9600);
    Serial.println(1234);
}

void loop()
{}
```

At the bottom of the code editor, there is a message: "Done Saving." Below the code editor, the status bar displays: "Binary sketch size: 1898 bytes (of a 32256 byte maximum)".

图3.5 在setup中写C语言语句

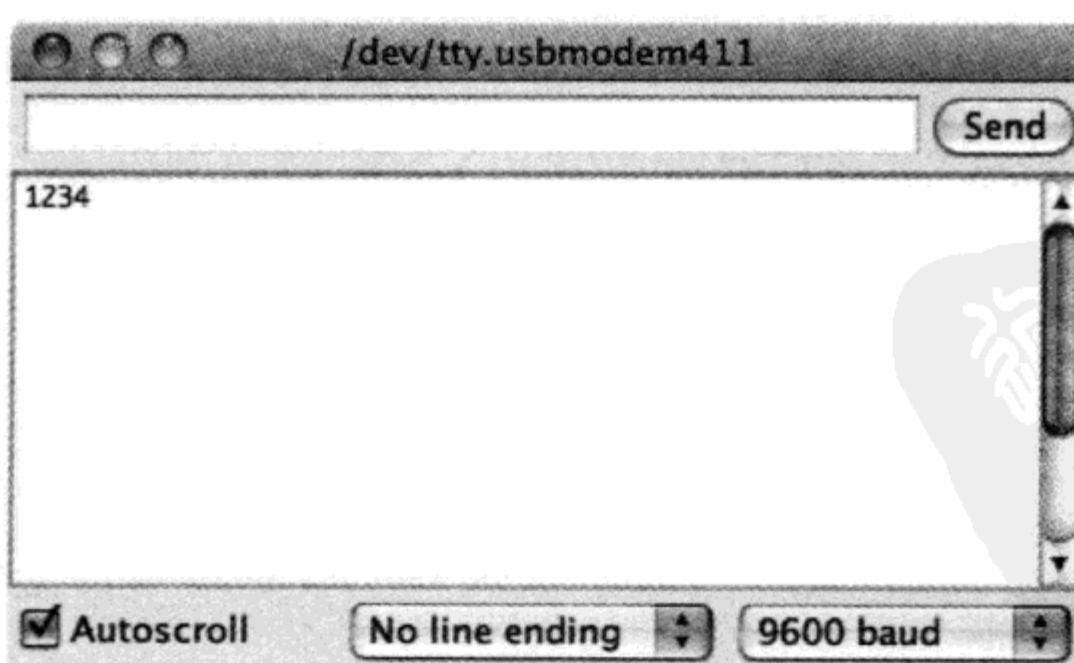


图3.6 串口监视器



你可以用这个方法来测试你能用C语言的变量和算法做到的一些事情，实际上这也是唯一可以看到C语言实验结果的方法。



数字变量和运算式

刚才你所做的事情是，将下面这行代码加到你的Sketch中，以不断延长闪烁的周期。

```
delayPeriod = delayPeriod + 100;
```

仔细看看这行代码，它是由变量名、一个等号和一个运算式(`delayPeriod + 100`)组成的。等号的功能是赋值。也就是说，它赋予变量一个新的值，这个值是由等号和后面的分号之间的東西决定的。在这个例子中，给变量`delayPeriod`的新值为`delayPeriod`原来的值加上100。

让我们用刚才说到的方法来测试下输入下面这段代码后Arduino能干什么。运行并打开串口监视器。

```
// sketch 3-04
void setup()
{
    Serial.begin(9600);
    int a = 2;
    int b = 2;
    int c = a + b;
    Serial.println(c);
}
void loop()
{}
```

图3.7是这段代码运行后在串口监视器里看到的内容。

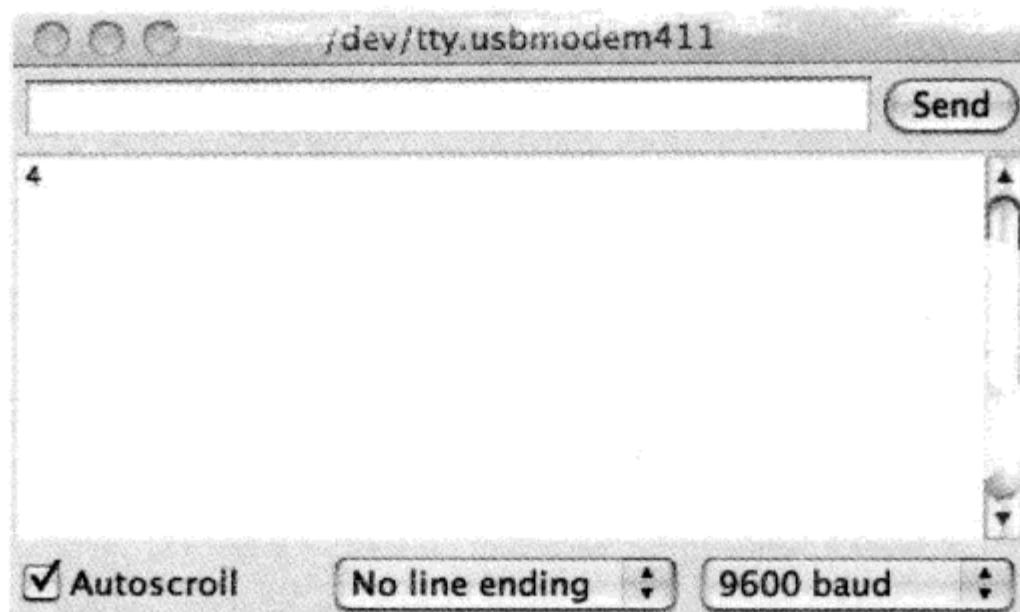
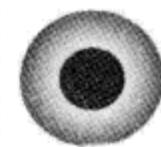


图3.7 简单运算

举一个稍微复杂点的例子，将摄氏温度换算成华氏温度——摄氏温度值乘以5，然后除以9，再加上32，所以你可以在Sketch中这样表达：

```
// sketch 3-05
void setup()
{
    Serial.begin(9600);
    int degC = 20;
    int degF;
    degF = degC * 9 / 5 + 32;
    Serial.println(degF);
}
void loop()
{}
```

这里有一些必须要注意的地方，首先是这一行：

```
int degC = 20;
```

写这一行代码的时候我们实际上做了两件事：声明了一个名为degC的整数型变量，而且已经把它的初始值定为了20。还有



另一种做法，你可以将这两件事分开写成两行：

```
int degC;  
degC = 20;
```

任何变量都只能声明一次，也就是告诉编译器变量的类型（如这个例子中的整数），但是给变量赋值多少次都可以：

```
int degC;  
degC = 20;  
degC = 30;
```

所以在摄氏温度和华氏温度换算的例子中，你定义了一个变量degC并给了它一个初始值20。但当你定义degF的时候，并没有给其初始值，而是根据换算公式算出并在发送到串口监视器之前才被赋值。

看一看运算式，你会发现我们用星号 (*) 作为乘号，反斜杠 (/) 作为除号。+ - * /这些运算符是有计算顺序的，先乘后除再加减。这和我们平时做算术的顺序是一样的，但是有的时候我们也会用括号来使运算式更清晰，例如：

```
degF = ((degC * 9) / 5) + 32;
```

运算式可以在需要的时候写得很长、很复杂，并且除了常用的几个运算符之外，还有一些不常用的运算符和大量的数学函数，后面将会学到它们。

3.6 指令

C语言中有一些内建指令。在这一部分，我们介绍其中的一



些并且来看看怎样在你的Sketch中使用它们。

if语句

在我们至今接触到的Sketch中，我们一直认为每行程序都被无一例外地逐行按顺序执行。但如果你不想这样呢？如果你想在某些条件成立的时候只执行Sketch的一部分呢？

让我们回到我们曾逐步展示的LED闪烁范例。在某一个时刻，它将会逐渐地越闪越慢，直到每一次闪烁都要以小时计。我们来看看怎样让它在慢到一定程度的时候重新回到开始时的闪烁速度。

要做到这一点，你必须用if语句。修改过的Sketch如下，试试吧。

```
// sketch 3-06
int ledPin = 13;
int delayPeriod = 100;

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
```



```
delayPeriod = delayPeriod + 100;  
if (delayPeriod > 3000)  
{  
    delayPeriod = 100;  
}  
}
```

`if`语句看起来有点像在定义一个函数，但这个相似只是表面上的。括号中的单词并不是一个参数，它被称为“条件”。此例中，条件为变量`delayPeriod`的值大于3000，若此条件成立，那么大括号中的指令就会被执行。在上面的例子中，代码将`delayPeriod`的值重新设为100。

如果条件不成立，Arduino将继续做下一件事。本例中，在`if`后面已经没东西了，所以Arduino会将`loop`函数再运行一次。

在脑海中演练一遍时间发生的顺序，会帮助你理解都发生了些什么。那么，下面就是所发生的事件：

- (1) Arduino运行`setup`并将LED针脚初始化为输出；
- (2) Arduino开始运行`loop`；
- (3) LED被点亮；
- (4) 延迟发生；
- (5) LED关闭；
- (6) 延迟发生；
- (7) 在`delayPeriod`上加100；
- (8) 如果`delayPeriod`最终大于3000，将其值设为100；
- (9) 回到第(2)步。

我们用符号`<`表示小于，这是一个比较运算符（表3.1）。



表3.1

运算符	含义	例子	结果
<	小于	9 < 10 10 < 10	真假
>	大于	10 > 10 10 > 9	假真
<=	小于等于	9 <= 10 10 <= 10	真真
>=	大于等于	10 >= 10 10 >= 9	真真
==	等于	9 == 9	真
!=	不等于	9 != 9	假

若要比较两个数，就要用到==运算符。这个双等号很容易和用于赋值的=混淆。

`if`有另一种形式可以做到：条件成立时做某件事，条件不成立时做另外一件事，我们将会在后面的例子中用到。

for循环

除了要在不同情况下执行不同的指令，你还经常需要在一个程序中多次运行一串指令。你已经知道了这么做的一种方法：使用`loop`函数。在`loop`函数中，所有的指令都运行完了之后，它会立刻自动重新来一遍，但是有时你需要更多的控制权。

举例来说，假设你想写一个每闪烁20次就暂停3s的Sketch。你可以在`loop`函数中一遍遍地重复相同的代码，如下：

```
// sketch 3-07
int ledPin = 13;
int delayPeriod = 100;
```



```
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);

    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);

    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);

    // repeat the above 4 lines another 17 times

    delay(3000);
}
```

但是这样做需要打太多的字。其实，有好几种更好的方法来实现，让我们先来看看一个使用for循环的方法，然后再看一下怎样用计数器和if语句来实现。

下面是用了for循环的Sketch。如你所见，它比前一个例子更短且更方便维护。



```
// sketch 3-08
int ledPin = 13;
int delayPeriod = 100;

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    for (int i = 0; i < 20; i++)
    {
        digitalWrite(ledPin, HIGH);
        delay(delayPeriod);
        digitalWrite(ledPin, LOW);
        delay(delayPeriod);
    }
    delay(3000);
}
```

for循环看上去有点像一个有3个参数的函数，只是这些参数使用分号间隔开，而不是用逗号。这是C语言的一个怪癖，如果你弄错了，编译器会立刻告诉你的。

for后面括号里的第一部分是变量声明，将一个变量指定为一个计数器变量，并赋予其初始值（本例中是0）。

第二部分是留在循环里的条件，只要*i*<20，都将运行循环中的指令；一旦*i*>20，程序将会停止loop中的指令。

最后一部分是每次在loop中所有的事都做完之后需要做的



事。在本例中，这件事是使`i`自加1，使`loop`运行20次后`i`不再小于20（原文是100，与前文不符——译者注）从而使程序从`loop`中退出。

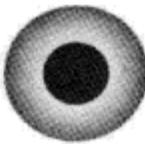
试着输入这段代码并运行它。熟悉语法和那些令人讨厌的标点符号的唯一办法就是输入它们并让编译器告诉你是不是有错误，一直这样做你最终会看到成效的。

这种方法有一个潜在的缺点就是`loop`函数会运行较长的时间，这对这个Sketch来说当然不是什么问题，因为它所做的不过是闪烁LED而已。但Sketch中的`loop`函数经常被用来检查有哪些按键被按下或接收了哪些串口的信号，如果处理器被一个`for`循环占用，它就不能做这些事情了。通常有一个好办法应对这个情况：若可以让`loop`函数尽可能快地运行，那么它就可以频繁运行更多次。

怎样实现呢？看看下面的Sketch：

```
// sketch 3-09
int ledPin = 13;
int delayPeriod = 100;
int count = 0;
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
```



```
delay(delayPeriod);
digitalWrite(ledPin, LOW);
delay(delayPeriod);
count++;
if (count == 20)
{
    count = 0;
    delay(3000);
}
}
```

你可能注意到下面的这一行：

```
count++;
```

这只是C语言中对

```
count = count + 1;
```

的缩写。

现在loop函数每运行一次，将只会用去比200ms稍长的时间，除非是第20次循环。因为每20次循环之间会有一个3s的延迟。实际上，对于某些应用，即使这样做也还是太慢了，而且纯粹主义者们会说：“你压根就不应该用delay”，最佳的解决方案要根据具体的应用来具体分析。

while循环

在C语言中，循环的另一个方法是在原本需要使用for循环的地方使用while循环。你可以完成像前面的例子可以完成一样的工作，如下面使用while循环：



```
int i = 0;  
while (i < 20)  
{  
    digitalWrite(ledPin, HIGH);  
    delay(delayPeriod);  
    digitalWrite(ledPin, LOW);  
    delay(delayPeriod);  
    i++;  
}
```

要想留在循环中，`while`后面括号中的表达式结果必须为真。否则，Sketch将继续运行大括号以后的指令。



#define指令

在Sketch运行中，像针脚的分配这样的常数值是不会改变的，除了使用变量外还有另一个方法——你可以使用`#define`指令把一个值和一个名称关联起来，Sketch中出现这个名称的地方都将在编译前用那个值来取代。

例如，你可以这样来定义，将LED分配给某个针脚：

```
#define ledPin 13
```

注意：`#define`指令并没有在名称和值之间用`=`，它甚至不需要在结尾处使用分号。这是因为它并不是C语言本身的一部分，而是预编译指令，在编译之前被运行。

笔者认为，这种方法比使用变量要麻烦。但它的好处是你不用耗费内存来存放它们，在内存紧张的时候不失为一个很好的办法。



3.7 总 结

本章我们开始学习C语言，你现在已经可以让LED以多种有趣的方式闪烁，并且可以使用serial.printIn函数将结果通过USB发回给你。你还学会了怎样用if和for语句来控制你的命令被执行的顺序，并学习了怎样用Arduino来做算术。

在下一章中，你将着重学习函数，还会学习除了整数(int)以外的其他几种变量类型。

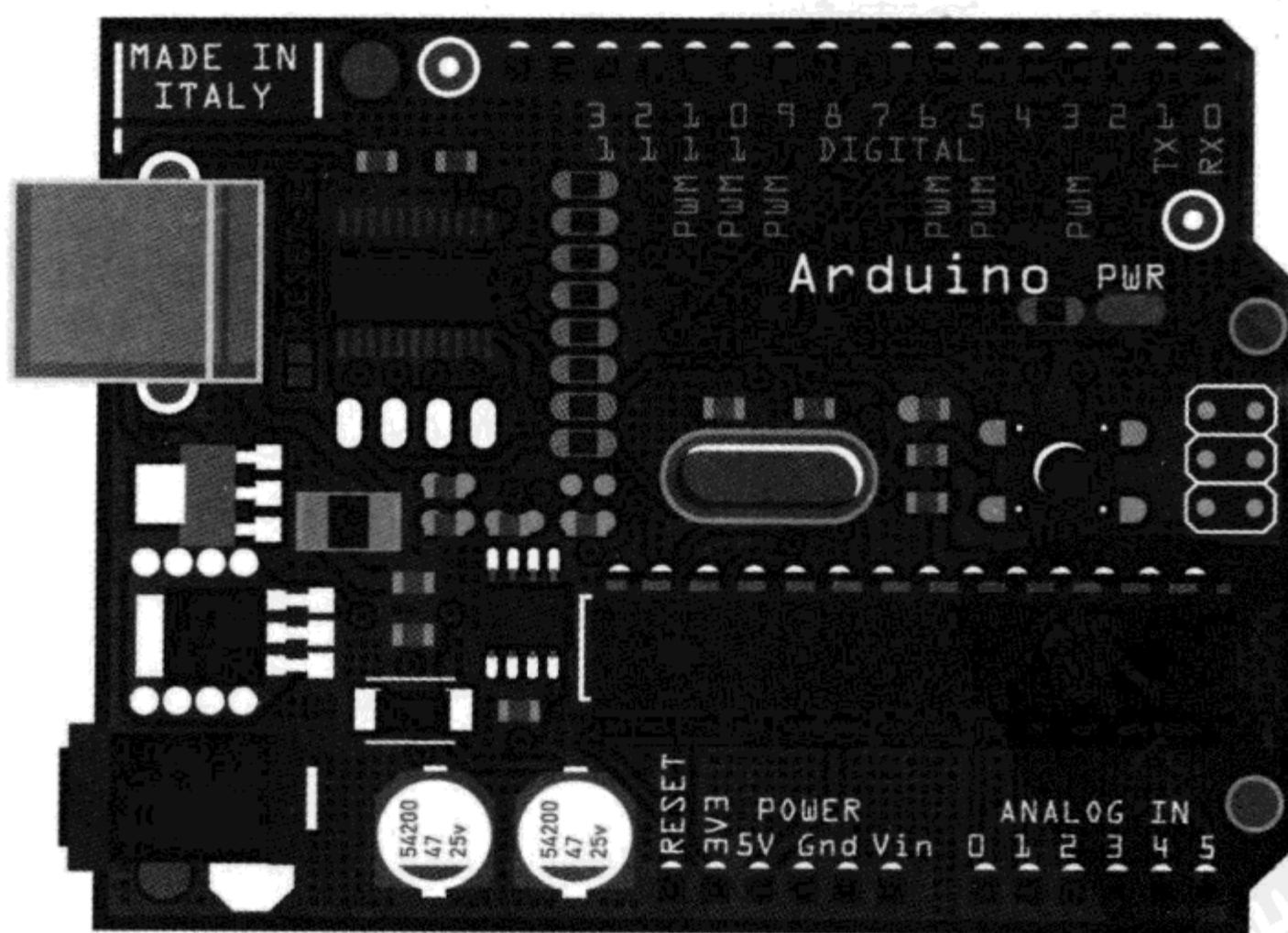




第4章

函 数

Functions





本章将着重介绍除了digitalWrite和delay这样已经为你定义好了的内建函数之外的那些你可以自己写的函数类型。

你需要写你自己的函数的原因是，随着Sketch变得复杂，你的setup和loop函数将会变得更长、更复杂，以至于最后很难看出它们是怎样工作的。

任何种类的软件开发，所面临的最大的问题是对于复杂度的控制。优秀的程序员写出来的程序可读性很强，而且只需要很少的注释。

想写出可读性好而且不会因为简单的改动就让整个程序变得一团糟的Sketch，函数是关键。

4.1 什么是函数？

函数有点像一个程序中的程序。你可以用它来包含一些小的任务。你定义的函数可以在Sketch的任何位置被调用，并包含它自己的变量和它自己的一串指令。当这些指令被运行完以后，程序回到调用函数后的位置开始运行。

用一个例子来说明，用来使LED闪烁的代码就是一个应该将部分代码放到一个函数中的例子，那么让我们来改一下“闪20次”的那个Sketch。这次我们将用到一个我们自己创建的函数flash：

```
// sketch 4-01
int ledPin = 13;
int delayPeriod = 250;

void setup()
```



```
{  
    pinMode(ledPin, OUTPUT);  
  
}  
  
void loop()  
{  
    for (int i = 0; i < 20; i ++)  
    {  
        flash();  
    }  
    delay(3000);  
}  
  
void flash()  
{  
    digitalWrite(ledPin, HIGH);  
    delay(delayPeriod);  
    digitalWrite(ledPin, LOW);  
    delay(delayPeriod);  
}
```

实际上，我们只是把那4行负责使LED闪烁的代码从for循环中移到一个为它们所创建的名为flash的函数中。现在，你可以写一个flash()来调用新函数，使LED在任何你需要的时候闪烁。本例中的新函数没有使用任何参数，其中用到的延迟值已经在前面用到的delayPeriod函数中被设定好了。

4.2 参 数

在把Sketch划分为一个个的函数时，通常值得考虑的是这些



函 数

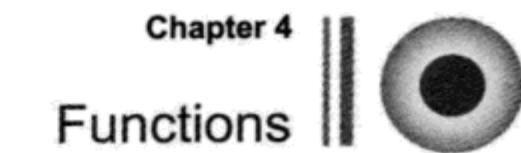
函数可以提供哪些功能，在函数flash的例子中，这点非常显而易见。现在，让我们给函数两个参数，告诉它需要闪烁多少次和每次闪烁的时间长短。看一看下面的代码，然后我们将更详细地解释参数是怎样工作的。

```
// sketch 4-02
int ledPin = 13;
int delayPeriod = 250;

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    flash(20, delayPeriod);
    delay(3000);
}

void flash(int numFlashes, int d)
{
    for (int i = 0; i < numFlashes; i++)
    {
        digitalWrite(ledPin, HIGH);
        delay(d);
        digitalWrite(ledPin, LOW);
        delay(d);
    }
}
```



如果我们看一下函数loop，会发现里面变得只有两行了，零散的工作交给函数flash了。注意我们是怎样调用函数flash的，括号里给了flash两个参数。

在Sketch的最下面，我们定义函数的地方，在括号里声明了变量的类型。在本例中，它们都是整数。这实际上是在定义新的变量，但是这些变量（numFlashes和d）都只能在函数flash内部使用。

这是一个很好用的函数，因为它包含了使LED闪烁所需的所有东西。它需要从外界获取的唯一信息就是LED在哪一个针脚上。如果你愿意，可以把这个也做成一个参数——当你的Arduino附有多个LED的时候，这样做是值得的。

4.3 全局、局部和静态变量

就像我们前面提到的，一个函数的参数只可在该函数内部使用，如果你写下面的代码，将会收到错误提示：

```
void indicate(int x)
{
    flash(x, 10);
}
x = 15;
```

或者，假设你这样写：

```
int x;
void indicate(int x)
{
    flash(x, 10);
```



```
}
```

```
x = 15;
```

这段代码在编译的时候不会出错，但是你需要小心。因为你现在实际上有2个名为x的变量，并且它们可以有不同的值。你在第1行中声明的那个被称为全局变量。之所以被称为全局，是因为你可以在程序的任何地方使用它，包括任何函数内。

但是，因为你在函数里也使用了相同变量名的x作为一个参数，就不能再简单的使用那个全局变量x了。因为当你在函数内提到变量x的话，那个局部版本的x的优先级较高，参数x就会覆盖同名的全局变量，将会在你为程序Debug的时候造成一些混淆。

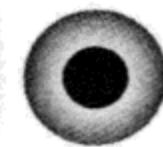
除了定义参数，你还可以定义不是参数的变量，但仍然只可以在函数内部使用。这些就是局部变量：

```
void indicate(int x)
{
    int timesToFlash = x * 2;
    flash(timesToFlash, 10);
}
```

局部变量timesToFlash只有当函数运行的时候才会出现。当该函数运行完它的最后1行代码后，就会立刻消失。这意味着，局部变量在程序中别的地方不能像在它被定义的函数中一样使用。

举个例子来说明，下面的代码将会导致出错：

```
void indicate(int x)
{
    int timesToFlash = x * 2;
```



```
    flash(timesToFlash, 10);  
}  
timesToFlash = 15;
```

经验丰富的程序员，在处理全局变量的时候往往持有怀疑的态度。因为它们不利于封装的主体。封装的概念是，你应该把有某一功能的代码包含在一个“包裹”内。因此，函数可以说是封装的一个很好的例子。全局变量的问题是，它们往往在Sketch的开头就被定义而且会在整个Sketch中被用到。有的时候这样做确实有很好的理由，但当使用传递参数更合适的时候，人们就懒得使用全局变量了。目前为止，在我们用过的例子中，ledPin是很适合作全局变量的，它可以非常方便地在Sketch开始的地方被找到，这使修改它变得很简单。实际上ledPin是一个常量，因为即使你修改了它并重新编译Sketch，你也不可能允许变量在Sketch运行的时候改变。在这种情况下，你可能更喜欢用我们在第3章中介绍过的#define指令。

局部变量的另一个特点是，它们的值在函数每次运行的时候都会被初始化，这在Arduino Sketch中的函数loop中是最为真实的（而且常常很不方便）。让我们试试在一个以前章节中的例子里用一个局部变量来代替全局变量。

```
// sketch 4-03  
int ledPin = 13;  
int delayPeriod = 250;  
void setup()  
{  
    pinMode(ledPin, OUTPUT);
```



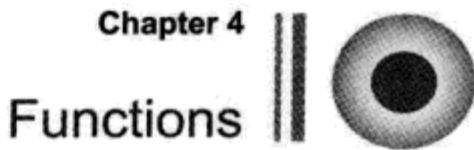
```
}

void loop()
{
    int count = 0;
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
    count++;
    if (count == 20)
    {
        count = 0;
        delay(3000);
    }
}
```

sketch 4-03是基于sketch 3-09修改的，用了一个局部变量代替全局变量来为闪烁计次。

这个Sketch是错误的，它不可能正常工作，因为每次函数loop运行的时候，变量count将会被重新赋值为0，所以count永远都不会达到20，而LED也会永远闪烁下去。我们把count作为全局变量的原因是，使它不会被重置。我们只会在函数loop中用到count，所以我们应该把它放在Sketch开始的地方。

幸运的是，C语言中有另一个机制来解决这一难题。这就是关键词static（静态），如果你在一个函数的变量声明前使用关键词static，就使变量只在函数第一次运行的时候被初始化。完美了！这正是这种情况所需要的。我们可以把变量放在它被用到



的函数中，而且它还不会在函数再次运行的时候被重置为0。

sketch 4-04是这种操作的实例：

```
// sketch 4-04
int ledPin = 13;
int delayPeriod = 250;
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    static int count = 0;
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
    count++;
    if (count == 20)
    {
        count = 0;
        delay(3000);
    }
}
```

4.4 返回值

作为一门数学学科，计算机科学的父母是数学和工程学，这使得编程中沿用了很多相关名称。函数本身是一个数学名词。在数学中，函数的输出完全由输入决定（参数），我们写过很多带



有输入的函数，但它们中没有一个返回给我们任何输出值——我们所有的函数都是void函数。如果函数可以返回一个输出值，那么你需要指定返回的类型。

让我们以写一个输入摄氏度，返回等效华氏温度值的函数：

```
int centToFaren(int c)
{
    int f = c * 9 / 5 + 32;
    return f;
}
```

现在定义函数不再以void开头，而是以int开头，表明此函数将给任何调用者返回一个整数。调用它的可能是像下面这样的一小段代码：

```
int pleasantTemp = centToFaren(20);
```

任何非空函数都必须包含一个return语句。如果你没写return语句，编译器会告诉你return缺失。你可以在一个函数中写不止一个return，这可能出现在使用if语句的时候。一些程序员对这个很头疼，但是如果你的函数很小（函数本应该很小），那么这做起来并不难。跟在return后面的可以是一个表达式，并不一定非要是一个变量名，所以你可以把前面的例子压缩为

```
int centToFaren(int c)
{
    return (f = c * 9 / 5 + 32);
}
```

如果返回的是一个表达式，而不是一个变量名，那么表达式需要被括在括号里，像刚才的例子中一样。



4.5 其他变量类型

我们至今为止只接触过整数型的变量，这也是我们现在最常用的变量类型，但是还有一些你需要知道的其他变量类型。

浮点数

这种类型和前面的温标转换范例有关，称为浮点。这种变量类型代表浮点数，也就是含有小数点的数，如 1.23 。当只用数字表示已经不够精确的时候，就需要用到这样的变量类型了。

注意下面的方程：

```
f = c * 9 / 5 + 32
```

如果 c 为 17 ，那么 f 将为 $17 * 9 / 5 + 32 = 62.6$ 。但如果 f 是一个整数（int），那么数值将被缩短为 62 。

如果我们没有注意计算的顺序，情况将会变得更糟。例如，假设我们先算了除法：

```
f = (c / 5) * 9 + 32
```

如果是一般的数学计算，结果仍为 62.6 。但如果所有的数都为整数，那么计算过程将如下：

- (1) 17 除以 5 等于 3.4 ，缩短为 3 ；
- (2) 3 乘以 9 ，然后加上 32 ，结果为 59 ——这和 62.6 差的有点远了。

这种情况下，我们可以使用浮点数。在下面的例子中，我们的温标转换方程用到了浮点数：



```
float centToFaren(float c)
{
    float f = c * 9.0 / 5.0 + 32.0;
    return f;
}
```

注意：我们在常数后面都加上了“.0”，这使得编译器把它们当做浮点数而不是整数来处理。

布 尔

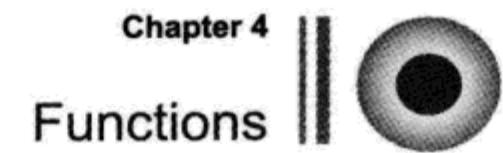
布尔值是一种逻辑值，不是真就是假。在C语言中，布尔用一个小写的字母b（boolean）表示；但在一般使用中Boolean要首字母大写，它是以发明它的数学家George Boole的名字命名的。布尔逻辑运算体系对计算机科学至关重要。

你可能没有注意到，但在我前面学习if语句的时候就已经见过布尔值了。if语句中的条件，`count == 20`实际上就是一个得到布尔运算结果的表达式。运算符`==`被称为比较运算符。就像`+`是一个可以将两数相加的运算符一样，`==`是比较两个数，然后返回真或假的一个运算符。

你可以像下面一样定义和使用布尔变量：

```
boolean tooBig = (x > 10);
if (tooBig)
{
    x = 5;
}
```

布尔值可以用布尔运算符来操作。所以类似对数字进行算术



运算一样，你也可以对布尔值进行运算，最常用的布尔运算符是`&&`（与运算）和`||`（或运算）

图4.1为与和或运算符的真值表。

		与		或	
		A		A	
		假	真	假	真
B	假	假	假	假	假
	真	假	真	真	真

图4.1 真值表

在图4.1所示真值表中，对于与运算，若A和B都是真，那么结果为真；否则，结果为假。对于或运算，只要A或B中有一个为真，则结果为真；只有当A和B都为假时，结果才为假。

除了与和或以外，还有一个非运算符，写做`!`。当然，真的非就是假，假的非也就是真了。

在`if`语句中，你可以把这些运算符结合起来用到布尔运算式中，如下：

```
if ((x > 10) && (x < 50))
```

其他数据类型

正如你所见，用整数和偶尔的用一下浮点数已经可以应付大部分情况。但是，在别的一些情况下也会用到一些其他数据类型。在Arduino Sketch中，整数可以占16位（二进制数字），即从-32767~32768之间的整数都可以使用整数型变量。



表4.1中总结了部分其他可用的数据类型。此表主要作为参考，本书后面的部分将会用到其中的一些数据类型。

表4.1 C语言中的数据类型

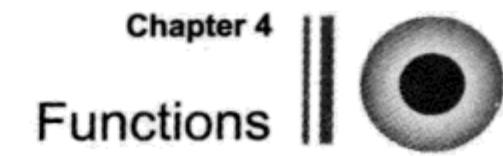
类 型	占 用 内 存 (字 节)	范 围	注 释
boolean	1	真或假 (0或1)	
char	1	-128~+128	用来存放 (ASCII) 字符，如A代表65。负数形式不常用
byte	1	0~255	作为一个单独的数据单元，常常被用来传输串行数据，见第9章
int	2	-32768~+32767	
unsigned int	2	0~65536	在不会出现负数的情况下，它有更大的精度，但是要小心使用，若和整数 (int) 之间进行数学运算，可能导致不想出现的结果。
long	4	-2147483648~2147483647	只用来存放超大的数
unsigned long	4	0~4294967295	见unsigened int
float	4	-3.4028235E+38~3.4028235E+38	
double	4	和float一样	它通常可以占8个字节，比float有更高的精度且范围更广，但在Arduino中，它和float是一样的

有一件需要考虑的事，如果数字超出了所使用的数据类型的范围，会有奇怪的事情发生。所以，如果有一个值为255的byte型变量，而你又在其上+1，就会得到0。更需要注意的是，如果你给值为32767的整数型变量+1，那么它的值将变为-32768。

在你完全熟悉这些数据类型之前，笔者还是建议使用整数型，因为它已经基本够用了。

4.6 编程风格

C语言编译器并不在意你怎样放置代码，你甚至可以把所



有的代码都写在1行里，只要语句之间有分号相隔。但是，整齐美观的代码比乱七八糟的代码更容易阅读和维护。这样说来，读代码和读书一样，格式很重要。

在某种程度上，格式是一个人的品位问题。没有人喜欢别人认为他的品位很差，所以关于代码应该看起来是什么样子的争论也变得很私人化。如果需要对别人的代码进行操作，开始的时候先把代码整理成自己喜欢的排列风格，这是程序员常做的事情。

C语言是有一个多年以来被公认的标准的，而本书也基本遵照这一标准。

首行缩进

在范例Sketch中你可以看到，我们常常在左侧缩进代码。如定义一个空函数的时候，关键词void在最左侧，和下面一行中的大括号对齐，接下来大括号里所有的内容都要缩进。缩几格并不重要，有人用2格，有人用4格，你也可以使用Tab键来缩进。本书中，我们都缩进2格。

如果你在函数定义中使用了if语句，那么if语句的大括号中的代码行都需要再多缩进2格，如下：

```
void loop()
{
    static int count = 0;
    count++;
    if (count == 20)
    {
```



```
count = 0;  
delay(3000);  
}  
}
```

你可能在第一个if中再套一个if，这样以前面的方法类推，相比最开始的大括号，缩进6格就好。

这些可能听起来很琐碎，但是如果你整理过别人排列的很差的代码以后，你就会发现，还是有规矩的比较好。

大括号

对于在定义函数的时候，或者if语句、for循环中在哪里放置第1个大括号有两派说法。一派就是像我们书中所有的例子一样，另起一行放置。另一派是放在语句或者函数定义的同一行中，如下：

```
void loop() {  
    static int count = 0;  
    count++;  
    if (count == 20) {  
        count = 0;  
        delay(3000);  
    }  
}
```

这种方法在java语言编程的时候很常用，java和C有很多相同的语法。笔者比较喜欢第1种方法，它是Arduino界内最常用的方法。



留 白

编译器会忽略空格和另起行，只是将它们用做将你的Sketch中的“令牌”和单词分开的工具，因此下面的代码可以顺利地通过编译：

```
void loop() {static int  
count=0;count++;if(  
count==20){count=0;  
delay(3000);}}
```

它确实没有什么太大的问题，但是要想看得明白，就只能祝你好运了。

在赋值的时候有人这样写：

```
int a = 10;
```

有的人则这样写：

```
int a=10;
```

你用哪种方法都可以，关键是保持前后一致。笔者习惯于使用第一种。

注 释

注释和代码都是程序的一部分，但并没有程序的功能。注释唯一的作用就是提醒你或其他人，写这些代码的意义何在。有的时候也用注释的格式来写标题。

编译器会完全忽略被标记为注释的文字。本书写到现在，我们也已经用了很多用注释格式写的标题，就在Sketch的开头位置。



下面是针对注释的两种语法格式：

- (1) 单行的注释用//开头，并写在此行的结尾处；
- (2) 多行的注释用/*开头，并用*/结束。

下面这个例子中用了两种注释方式：

```
/* A not very useful loop function.  
Written by: Simon Monk  
To illustrate the concept of comments  
*/  
  
void loop() {  
    static int count = 0;  
    count++; // a single line comment  
    if (count == 20) {  
        count = 0;  
        delay(3000);  
    }  
}
```

本书中用得较多的是单行的注释格式。

好的注释可以帮助说明Sketch中发生了什么，或Sketch的使用方法。这在别人使用你的Sketch的时候很有用。同时，当你隔了几个星期重新回到这个项目的时候，也会对快速地重新开始很有帮助。

可能有人在上编程课的时候学到：“注释越多越好”。但是大多数经验丰富的程序员会告诉你，写得好的程序只需要很少的注释，因为程序本身就是很好的注释了。

你应该只因为下面几个原因才使用注释：

- (1) 用来解释你做得较为技巧性的东西，或者那些不能立



刻见效的代码；

- (2) 用来描述需要用户操作的地方，而这些操作并不是程序的一部分，例如：//此针应被连接到晶体管来控制延迟；
- (3) 给你自己写备忘录，例如：//todo：整理这里，太乱了。

最后这一点是非常有用的技巧。程序员经常在代码中写备忘录来提醒自己稍后或者以后要做的事情，他们可以用IDE中的搜索功能来找到程序中所有的//todo。

你不应该由于以下的原因而写注释：

- (1) 解释再明白不过的事情，例如：a=a+1 //给a加上1；
- (2) 解释写得不好的代码，它们需要的不是解释，而是改正！

4.7 总 结

本章理论性较强，你应该已经吸收了一些关于怎样把你的Sketch组织为函数的集合，和养成自己的编程风格而从长远的角度节省更多时间的主要理念。

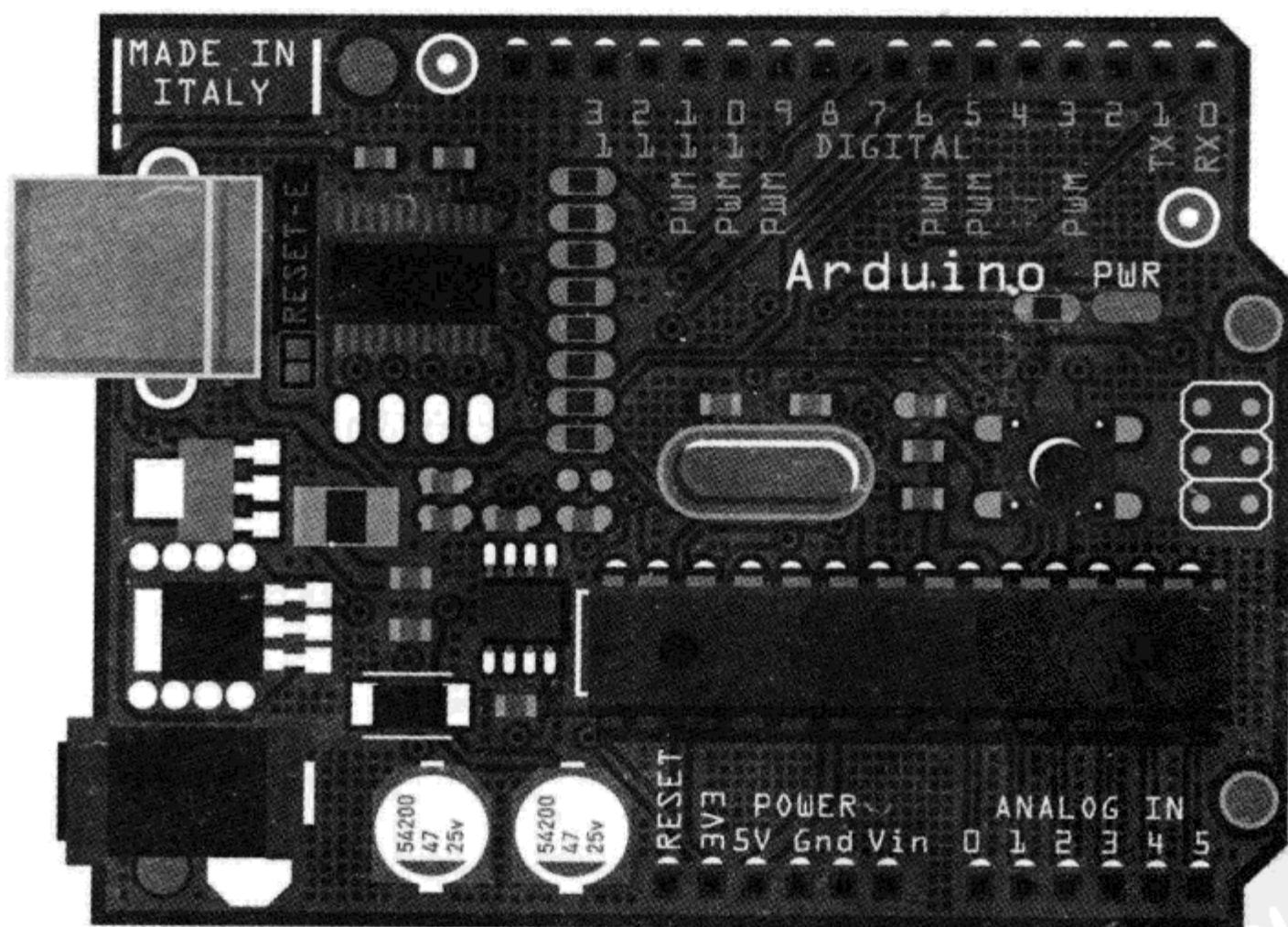
下一章，你将开始真正使用一些学过的知识，并了解一些构架数据的更好的方法，还将开始使用字符串。



第5章

数组和字符串

Arrays and Strings





在读完第4章以后，你对怎样合理地构架数据来让事情变得简单已经有了理性的认识。如果说有什么事是一个优秀的程序员最喜欢的，那就是让事情变得简单。现在，我们要把注意力转移到你的Sketch中所使用的数据上了。

Niklaus Wirth的*Algorithms+Data Structures=Programs*已经出版一段时间了，但它仍然是计算机科学尤其是编程的精华所在。笔者对所有被程序Bug所困扰的人强力推荐此书。他也提出了“你必须同时考虑算法（你所做的事情）和所使用的数据结构才能写出好的程序”的理念。

你已经学习了loop和if，还有那些在Arduino编程中被称为“算法”的部分。现在，你将要学习怎样构架你的数据了。

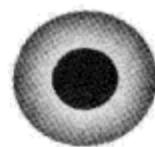
5.1 数组

数组是用来包含一组值用的。你现在所接触过的变量都只能包含一个单独的值，而且还基本上整数。比较起来，数组可以包含一组数值，而且你可以通过它们在数组中的位置来访问它们。

C语言和大多数语言一样，目录的起始是0而不是1。这意味着，数组中的第1个元素其实是元素0。

为了解释数组的用法，我们可以创建一个利用Arduino板载LED来不断闪烁出莫尔斯码“SOS”的范例。

莫尔斯码在19和20世纪曾是非常重要的通信手段。因为它将字母编码为一串长短不一的点或线，所以莫尔斯码可以通过电话线、无线电或信号灯来传输。“SOS”（Save Our Souls）这3个



字母现在仍然是国际通用的求救信号。

字母S用3个短闪（点）来表示，字母O则用3个长闪（线）来表示。你可以使用一个整数型的数组来决定你每次闪烁的长度，然后用for循环来逐个使用数组里的时长值，作出合适时长的闪烁。

我们先来看下怎样创建一个含有闪烁时长的整数型数组：

```
int durations[] = {200, 200, 200, 500, 500, 500, 200,  
200, 200};
```

定义一个其中包含数组的变量，要在变量名后面加上[]。

在本例中，在创建数组的时候就设置好每次闪烁的时长值。语法为，在大括号中写这些值，并用逗号分开。别忘了此行结束时的分号。

用方括号来访问数组中的每一个元素。例如，你要访问数组中的第1个元素，可以这样写：

```
durations[0]
```

为了说明，让我们创建一个数组，然后在串口监视器中显示它所有的值：

```
// sketch 5-01  
int ledPin = 13;  
  
int durations[] = {200, 200, 200, 500, 500, 500, 200,  
200, 200};  
void setup()  
{  
    Serial.begin(9600);
```



数组和字符串

```
for (int i = 0; i < 9; i++)  
{  
    Serial.println(durations[i]);  
}  
  
void loop() {}
```

将Sketch上传至板子，并打开串口监视器，如果没出错的话你会看到图5.1所示内容。

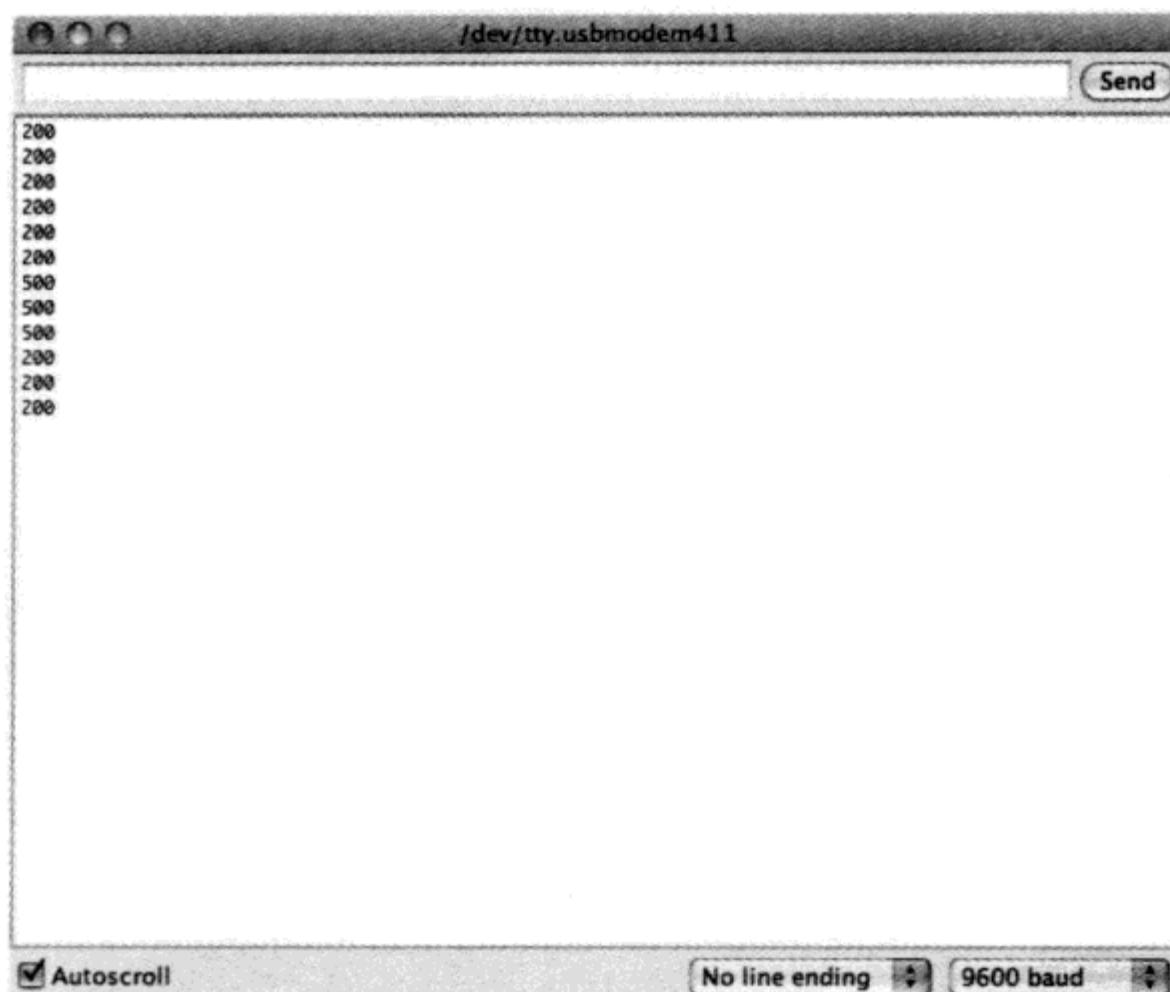


图5.1 串口监视器里显示的sketch 5-01的输出

这挺整洁的。如果你想要在数组中添加更多的时长值，只需要在大括号里加上新的值，然后把for循环中的9改成数组现在的大小就可以了。

使用数组的时候要谨慎一些，因为编译器不会阻止你访问数



组范围以外的元素。这是因为，数组中的元素实际上是指向对应的内存地址的，如图5.2所示。

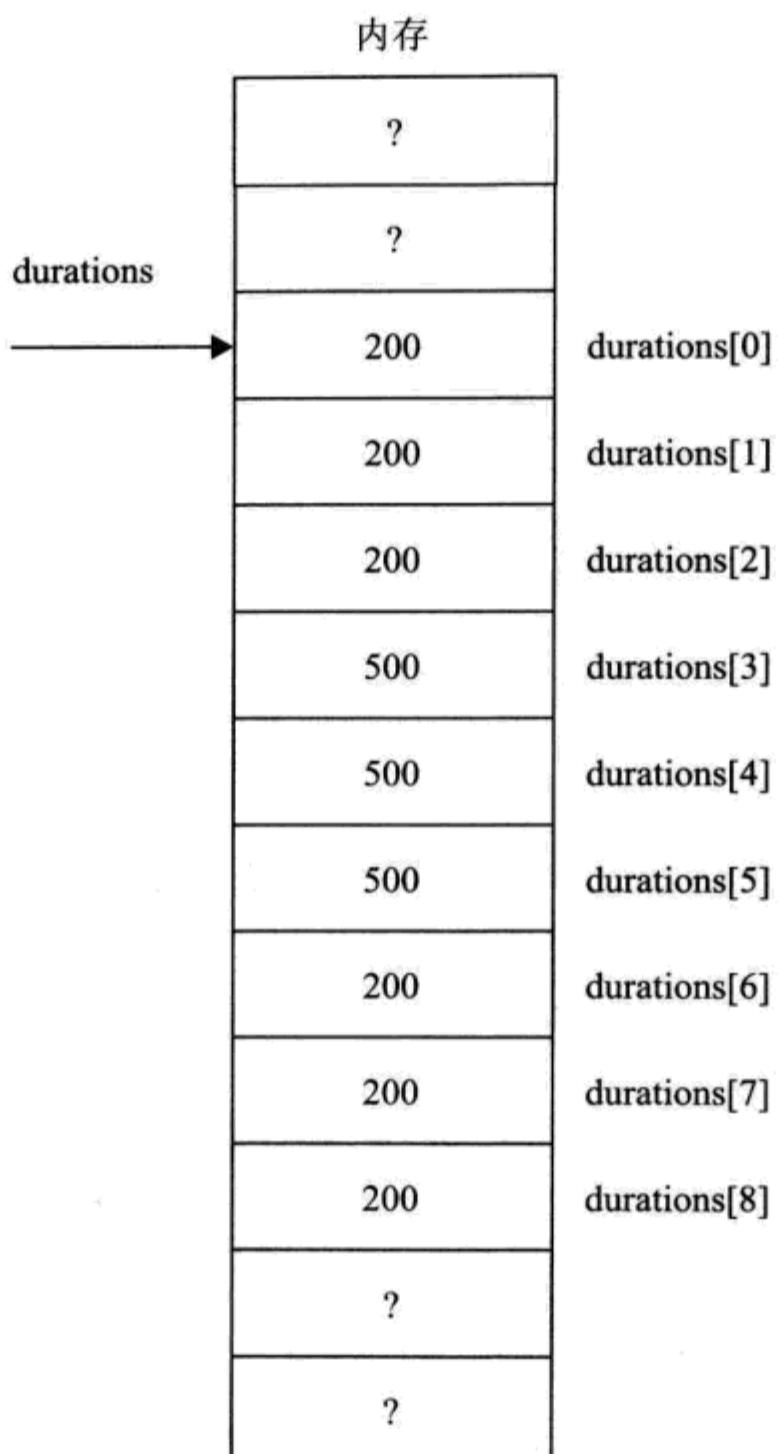


图5.2 数组和指针

程序把它的数据，包括一般的变量和数组都存储在内存中，计算机的内存比人类的大脑记忆要严谨的多。把Arduino的内存想象成为一些文件柜中的抽屉会比较容易理解，9个元素的数组相当于占用了9个抽屉，每个元素对应一个抽屉，变量指向抽屉或数组中的元素。对于访问超出数组范围的元素，可以理解



数组和字符串

为虽然抽屉中没有东西但是你仍然可以打开它。如果你要访问 `durations[10]`，那么你仍然会得到一个整数，但这个整数的值可以是任意的。这本身并没有什么害处，但是如果你不小心得到了一个数组以外的值，就可能会让你的Sketch得到一个错误的结果。

然而更糟的情况是，如果你改变一个数组范围以外的数据值，如你的程序中有像下面这么一行，结果一般会让你的Sketch崩溃。

```
durations[10] = 0;
```

标号为 `durations[10]` 的抽屉可能正在被其他完全不同的变量所占据。所以务必要记住，永远不要访问超出数组范围以外的值。如果你的Sketch运行有异，别忘了检查这方面。



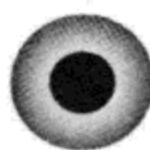
SOS莫尔斯码所使用的数组

sketch 5-02是这样用数组来实现SOS呼叫信号的：

```
// sketch 5-02
int ledPin = 13;

int durations[] = {200, 200, 200, 500, 500, 500, 200,
200, 200};

void setup()
{
    pinMode(ledPin, OUTPUT);
}
```



```
void loop()
{
    for (int i = 0; i < 9; i++)
    {
        flash(durations[i]);
    }
    delay(1000);
}

void flash(int delayPeriod)
{
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
}
```

这种方法的一个很明显的优势是，可以轻松通过改变 `durations` 数组来改变要表达的信息。在 sketch 5-05 中，将用数组进一步来制作含有更多意义的莫尔斯码。

5.2 字符串数组

在计算机语言中，`string` 已经不是原本“绳串”的意思了，而是一个字符的序列，是让 Arduino 可以处理字符的方法。例如，Sketch 将每秒向串口监视器发送一次“Hello”：

```
// sketch 5-03
void setup()
{
```



数组和字符串

```
Serial.begin(9600);  
}  
  
void loop()  
{  
    Serial.println("Hello");  
    delay(1000);  
}
```



字符串字面值

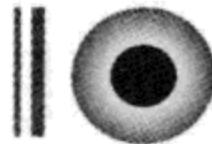
字符串的字面值被写在双引号中。字符串也是常量，就像是整数123。

如你所期待的一样，你可以把字符串赋给变量。而且还有一个高级字符串库，但暂时你还是用标准C语言字符串，就像sketch 5-03中的一样。

在C语言中，一个字符串的字面值其实就是一个类型为char的数据类型char和int类似，是数字，但是这个数字的范围是0~127，而且每个都代表1个字符。这个字符可能是字母表中的字母、数字、标点符号，或者是诸如Tab、换行之类的特殊字符。这些字符的数字代码使用遵照ASCII标准，表5.1是一些常用的ASCII码对照。

表5.1 常用ASCII码

字符	ASCII码（十进制）
a-z	97-122
A-Z	65-90
0-9	48-57
space	32



字符串“Hello”也是一个字符的数组，如图5.3所示。注意：字符串字面值结尾处有一个特殊的空字符，用来标明字符串的结束。

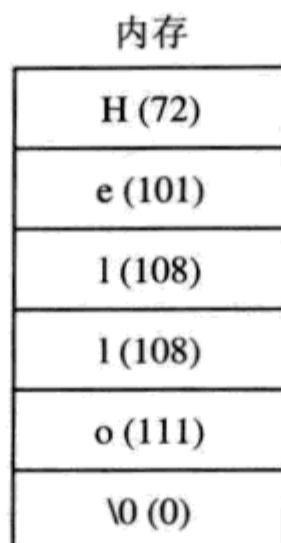


图5.3 “Hello”的字符串字面值

字符串变量

和你可能期待的一样，字符串变量和数组变量差不多，有较为快捷的方式来定义其初始值。

```
char name[] = "Hello";
```

它定义了一个字符的数组并将其初始化为单词“Hello”，也在最后加行一个空字符（ASCII 0）来标记字符串的结束。

虽然前面的例子和你所知道的数组的写法基本一致，但下面这样的写法更常见：

```
char *name = "Hello";
```

这样写和前面的例子效果是一样的，*表示一个指针，意思是name指向char数组的第一个char元素，也就是字母H的内存地址。



你也可以用变量和一个字符串常量来重写sketch 5-03：

```
// sketch 5-04
char message[] = "Hello";

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    Serial.println(message);
    delay(1000);
}
```

5.3 莫尔斯码翻译器

让我们把所学过的数组和字符串的知识放到一起，来写一个更为复杂的Sketch，让它可以接收一切来自串口监视器的信息，然后在板载LED上用莫尔斯码的方式闪烁出来。

字母对应的莫尔斯码见表5.2。

莫尔斯码的规则是，线时长为点时长的3倍，点或线之间的间隔时长为一个点的时长，每两个单词之间的间隔时长等于7个点时长之和。

这个项目没有考虑标点符号，如果你想把它们加进去的话将会是一个很有趣的练习。完全版本的莫尔斯码对照表，见en.wikipedia.org/wiki/Morse_code。

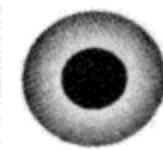


表5.2 莫尔斯码字母表

A	-.	N	-.	0	----
B	-...-	O	---	1	.----
C	-.-.	P	.--.	2	..---
D	--.	Q	--.~	3	...--
E	.	R	.~.	4-
F	...~.	S	...~	5
G	~.	T	-	6	-....
H	U	..~	7	--...
I	..	V	...~	8	---..
J	---.	W	.~	9	---.
K	~-.	X	~-~		
L	~-..	Y	~-~		
M	--	Z	--~		

数 据

一步步地完成这个项目，先从构架用来代表莫尔斯码的数据开始。

你要明白的是，解决这个问题并非只有一种方法，不同的程序员会用不同的方法。所以千万别告诉自己：“我可想不出什么别的方法”，你自己很有可能想出一些别的甚至是更好的方法。每个人思考的方式不同，而这个方案只是笔者突然想到的而已。

这些数据只是要想一个办法把上面的表格用C语言表达出来而已。实际上，你可以把数据分到两个表中：一个给字母，还有一个给数字。字母的数据构架如下：

```
char* letters[] = {
    ".-", "-...", "-.--.", "-..", ".",
    // A-I
    "...~.", "--.", "....", "...",
    ...
```



```
".---", "-.-", ".-.", "--", "-.", // J-R  
"---", ".--.", "--.-", "-.-",  
"...", "-", "...-", "...-", ".--", // S-Z  
"-..-", "-.--", "--.."  
};
```

这是一个字符串字面值数组。因为字符串字面值实际上是一个char类型的数组，实际上这里也就是一个由数组组成的数组——一种绝对符合规则而且很实用的东西。

这意味着，要想找到“A”的莫尔斯码，你只需要访问letters[0]，然后得到字符串“_”。这种方法的效率并不是特别高，因为你要为一个短线或点使用一整个字节的内存（8位）。其实只需要使用1位就可以了，但你可以很容易为这种方法找到解释：就算是这样，整个下来也只用了90字节，而我们有2000字节的内存可以使用，而最重要的是它让我们的代码更容易理解。

数字用同样的方法：

```
char* numbers[] = {  
    "-----", ".----", "...--", "...--", "....-",  
    ".....", "-....", "--... ", "---..", "----."};
```

全局变量和setup

你需要定义两个全局变量：一个是点的时长，一个是LED所用的针脚。

```
int dotDelay = 200;  
int ledPin = 13;
```



setup函数比较简单，你只需要把ledPin设为输出，然后设置好串口就可以了。

```
void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}
```

loop函数

你现在需要在loop函数中处理真正的工作了，函数的算法如下：

- (1) 假设从USB读到1个字符；
- (2) 如果是1个字母，用字母数组闪烁；
- (3) 如果是1个数字，用数字数组闪烁；
- (4) 如果是空格，闪烁4倍点时长。

这就是全部了，不要想得太过复杂。这个算法表达了你想做的或者你的意图，这种编程的风格就叫做“按意图编程”。

如果你用C语言来写这个算法，那么看起来应该是这样：

```
void loop()
{
    char ch;
    if (Serial.available() > 0)
    {
        ch = Serial.read();
        if (ch >= 'a' && ch <= 'z')
        {
```



```
    flashSequence(letters[ch - 'a']);

}

else if (ch >= 'A' && ch <= 'Z')

{

    flashSequence(letters[ch - 'A']);

}

else if (ch >= '0' && ch <= '9')

{

    flashSequence(numbers[ch - '0']);

}

else if (ch == ' ')

{

    delay(dotDelay * 4); // gap between words

}

}
```

这里有几个地方需要解释一下。第一，其中有一个serial.available()。想要理解它，首先要知道Arduino是怎样通过USB和你的电脑沟通的。图5.4概括了这一过程。

当电脑把数据从串口监视器发往Arduino板的时候，信号根据USB信号电平和协议转换成Arduino板上单片机可以使用的信号。这个对话发生在Arduino板上的一个特殊功能芯片中。然后，数据被单片机中的通用异步收发器（UART）接收。UART将接收到的数据放入一个缓冲区（buffer）。缓冲区是一段特殊的内存区域，其中的数据被读取完毕后就被从缓冲区里擦除了。

数据的通信和你的Sketch在做什么并没有关系，即使你可能正在开心地闪着LED，数据还是会到达缓冲区并呆在那里等着你

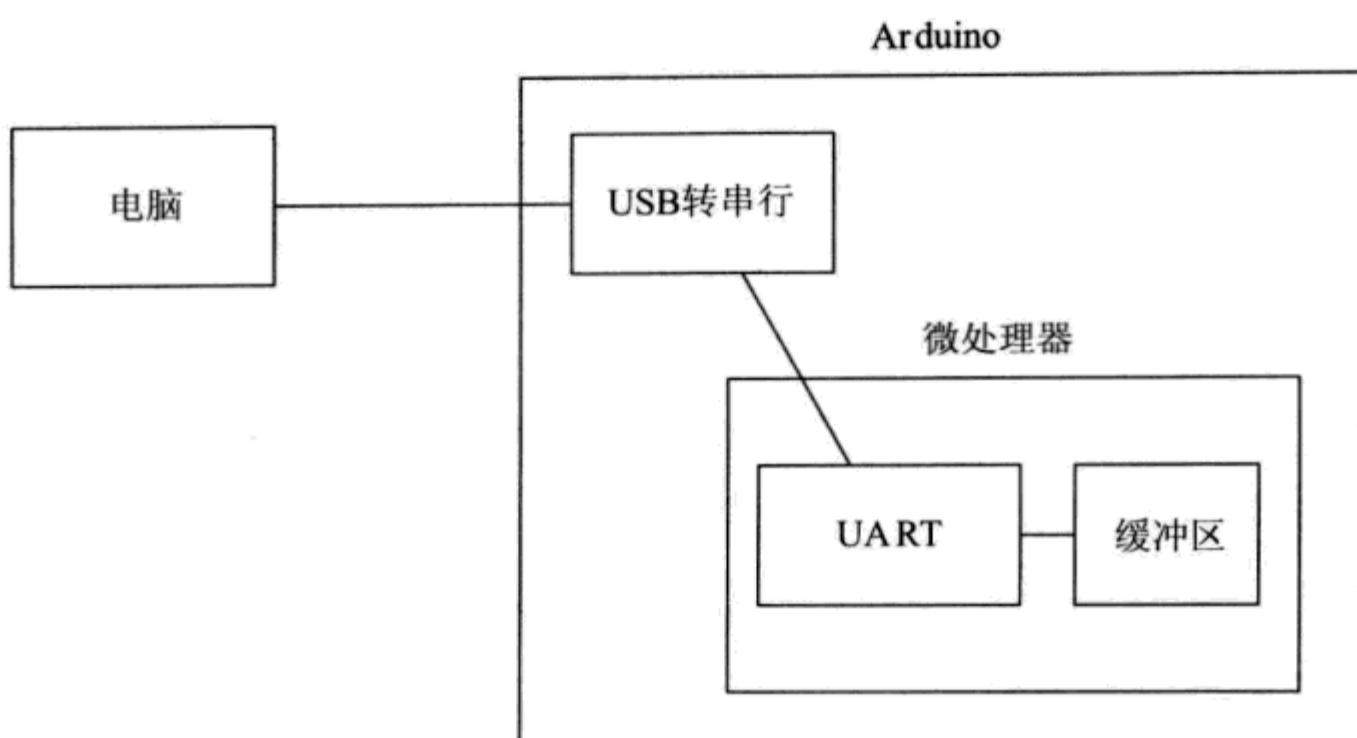


图5.4 和Arduino之间的串行通信

来读取。你可以把缓冲区看成有点像电子邮件的收件箱。

检视你是否有未读邮件的方法是，用函数`serial.available()`。这个函数返回的是缓冲区中等待读取的数据的字节数。如果没有待读的信息，函数将会返回0。这也就是为什么`if`语句的条件是，有多于0字节的数据可以读取。如果有的话，语句所做的第一件事情就是使用`serial.read()`函数读取下一个可用的`char`，这个函数被分配给局部变量`ch`。

接下来，另一个`if`用来决定你要闪出什么信息。

```

if (ch >= 'a' && ch <= 'z')
{
    flashSequence(letters[ch - 'a']);
}

```

这个一开始看起来可能有点奇怪，因为使用`>=`和`<=`来比较字符。实际上你是可以这么做的，因为每个字符其实都是由一个数字代表的（ASCII码）。所以，如果字符是在a-z之间的（97-



122），那么你就知道电脑发过来的字符是小写字母，接着就可以调用一个现在还没写到的函数flashSequence，给这个函数传递一个由点和线组成的字符串。例如，要闪出“a”，你需要传递“._”作为它的参数。

真正闪烁LED的责任被移交给了这个函数，这样会让我们的代码更易读。别试着在loop函数中做这件事。

下面的是决定你需要发送给函数flashSequence的是怎样的
一串点和线的C语句：

```
letters[ch - 'a']
```

这看起来又有点奇怪了。这个函数是在将两个字符相减。这
么做其实也非常的合理，因为函数确实是在用ASCII码来做减法。

回忆你将字母对应的莫尔斯码放入数组的步骤：数组的第一个元素是一个代表字母a的点和线的字符串，第2个是代表b的，以此类推。所以你要为刚从缓冲区里读到的字符在数组中找到对应的正确的位置，而这个位置也就是该字母的ASCII值减去字母a的ASCII值。那么， $a-a$ 就是 $97-97=0$ ， $c-a$ 也就是 $99-97=2$ 。所以在前面的代码中，如果ch为字母c，那么方括号中的结果就是2，你也将从数组中访问到2号元素“._”。

刚才这段描述的是怎样处理小写字母。你还需要处理大写字母和数字，方法类似。

函数flashSequence

我们前面已经提到了一个名为flashSequence的函数，并已经开始用到它了，现在要真的开始写这个函数了。我们计划用它



取用含有点和线的字符串来制造时长正确且需要的闪烁。

想一想算法，可以分为下面的步骤：

- (1) 字符串中的每个元素都是点和线（如.-.）；
- (2) 闪烁每个单独的点和线。

使用“按意图编程”的方法，让我们把函数写的像上面的步骤一样简洁。

每个字母的莫尔斯码的长度不尽相同，所以你必须不断循环字符串，直到遇到结束符\0。你还需要一个名为i的计数器变量，从0开始按照闪烁的点或线的进程递加：

```
void flashSequence(char* sequence)
{
    int i = 0;
    while (sequence[i] != '\0')
    {
        flashDotOrDash(sequence[i]);
        i++;
    }
    delay(dotDelay * 3);      // gap between letters
}
```

现在又将闪烁每个单独的点或者线的任务指派给了一个新的函数flashDotOrDash，它才真正地控制LED的开关。最终，当程序闪完这些点和线以后，它需要暂停3个点的时值。注意：这里的注释就很必要。

函数flashDotOrDash

这一长串的函数中，最后一个才是真正控制LED的开关的。



数组和字符串

函数用点(.)或线(-)构成它的参数。函数要做的只是在点的时候将LED点亮1个点的时长，在线的时候将LED点亮3个点的时长，然后关闭LED，最终再延迟一个点的时长作为闪烁之间的间隔。

```
void flashDotOrDash(char dotOrDash)

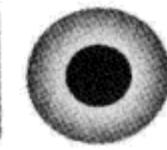
{
    digitalWrite(ledPin, HIGH);
    if (dotOrDash == '.')
    {
        delay(dotDelay);
    }
    else // must be a -
    {
        delay(dotDelay * 3);
    }
    digitalWrite(ledPin, LOW);
    delay(dotDelay); // gap between flashes
}
```



完整代码

把前面讲过的放在一起，完全版的代码见sketch 5-05，把它上传至Arduino板并运行。记住：如果要使用它，只要打开串口监视器，然后在顶部的框中输入一些单词并点击“发送”(send)，接着就会看到单词被闪烁成莫尔斯码。

```
// sketch 5-05
int dotDelay = 200;
```



```
int ledPin = 13;

char* letters[] = {
    ".-", "-...", "-.-.", "-..", ".-", "...-", "--.",
    "....", "..", // A-I
    ".---", "-.-", "-...-", "---", "-.", "---", ".---",
    "---.", ".-.", // J-R
    "...", "-.", "...-", "...-", ".--", "-..-", "-.--",
    "--.." // S-Z
};

char* numbers[] = {"-----", ".----", "...--", "....-",
    "....-", ".....", "-....", "--...-", "---..", "----."};

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    char ch;
    if (Serial.available() > 0)
    {
        ch = Serial.read();
        if (ch >= 'a' && ch <= 'z')
        {
            flashSequence(letters[ch - 'a']);
        }
        else if (ch >= 'A' && ch <= 'Z')
        {

```



数组和字符串

```
    flashSequence(letters[ch - 'A']);  
}  
  
else if (ch >= '0' && ch <= '9')  
{  
    flashSequence(numbers[ch - '0']);  
}  
  
else if (ch == ' ')  
{  
    delay(dotDelay * 4); // gap between words  
}  
}  
  
}  
  
void flashSequence(char* sequence)  
{  
    int i = 0;  
    while (sequence[i] != '\0')  
    {  
        flashDotOrDash(sequence[i]);  
        i++;  
    }  
    delay(dotDelay * 3); // gap between letters  
}  
  
void flashDotOrDash(char dotOrDash)  
{  
    digitalWrite(ledPin, HIGH);  
    if (dotOrDash == '.')  
    {  
        delay(dotDelay);  
    }
```



```
else // must be a -
{
    delay(dotDelay * 3);
}
digitalWrite(ledPin, LOW);
delay(dotDelay);           // gap between flashes
}
```

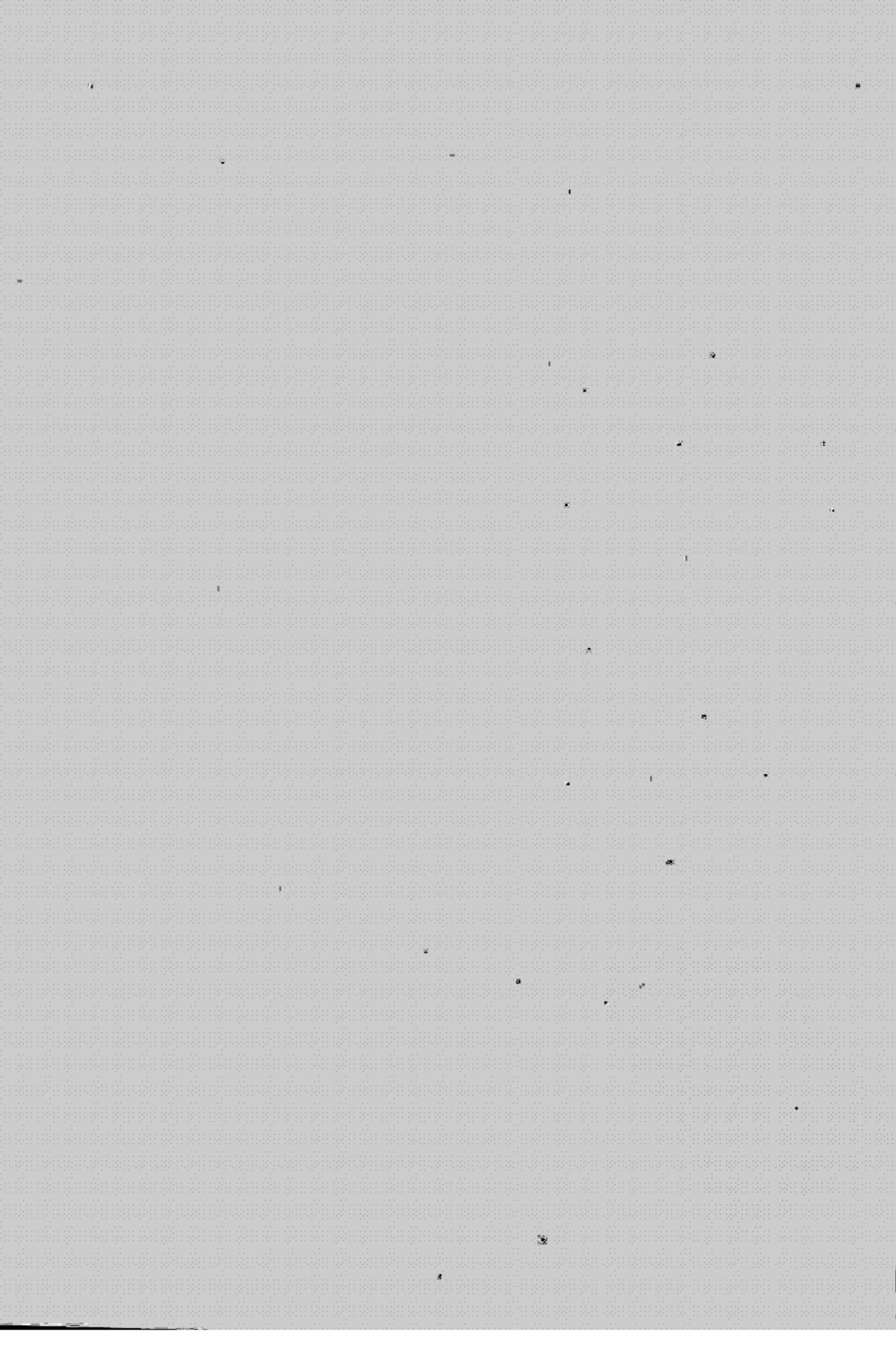
这个Sketch包含一个自动并且不断重复调用函数flashSequence的loop函数，而flashSequence又在不断调用另外一个函数flashDotOrDash，flashDotOrDash又在调用由Arduino提供的函数digitalWrite和delay。

这才应该是你的Sketch应该有的样子。把整个事件拆开成一个个的函数，使代码简洁清晰，即使一段时间没有碰它，也可以轻松回到工程上。

5.4 总结

本章除了学习了数组和字符串以外，还制作了一个更为复杂的莫尔斯码翻译器，希望通过它你可以更深刻地认识到通过函数来组成Sketch的重要性。

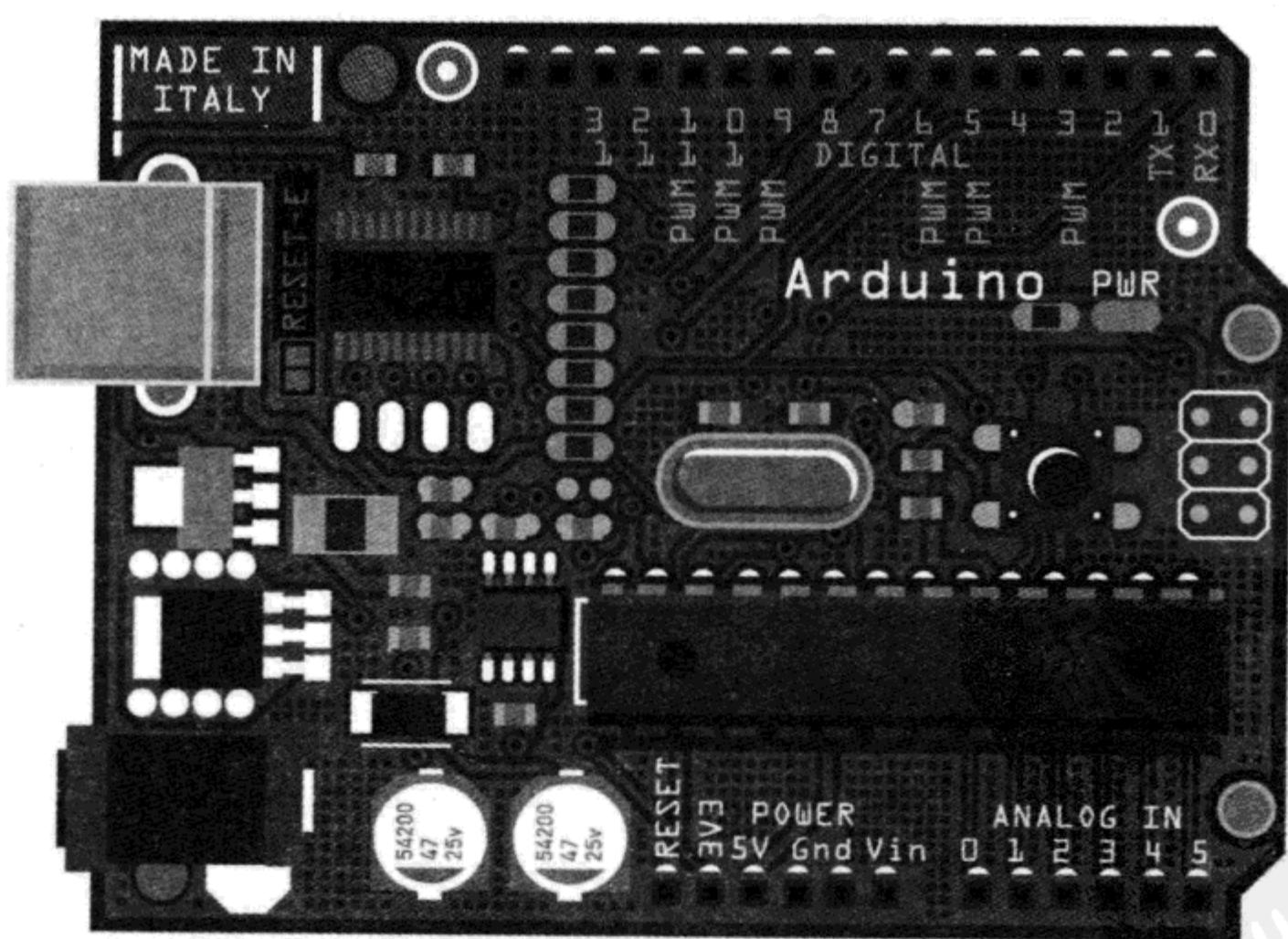
下一章，你将学习输入和输出——从Arduino输入或输出模拟和数字信号。



第6章

输入和输出

Input and Output





Arduino是一种交互装置，这意味着会在其上附加其他电子元器件，所以你需要理解怎样去使用它众多的连接针脚。

输出可以是数字信号，也就是只有0V或者5V两个值；输出也可以是模拟信号，即可以把电压调节到0~5V之间的任意值——虽然不像说起来这么简单，但以后会慢慢了解到。

同样，输入也可以是数字信号（如检测一个按钮是否被按下）或者模拟信号（如从一个光传感器得到的信号）。

本书主要介绍软件而不是硬件，故尽量不在电子学上赘述过多。但是，如果准备一个万用表和一些小段硬芯导线，将会帮助你更好地理解本章的内容。

6.1 数字输出

在前面的章节中，我们已经把附加在针脚13上的LED加以利用了。如在第5章中，我们将它用作一个莫尔斯码的信号灯。Arduino板上还有很多其他可以用的数字信号针脚。

让我们用板上其他针脚做个实验。这里使用针脚4，为了看清过程，需要将万用表连接到Arduino上。图6.1为连接方法。如果你的万用表有鳄鱼夹，只要把硬芯导线两端的绝缘层剥去，将鳄鱼夹夹在一端上，另一端则插在Arduino的插母里即可；如果没有鳄鱼夹，就需要把剥去绝缘层的导线缠绕在万用表的探针上。

万用表需要调节到直流0-20V量程，负极线（黑色）需要连接到板上的地线针脚（GND），正极连接到D4。硬芯导线的一

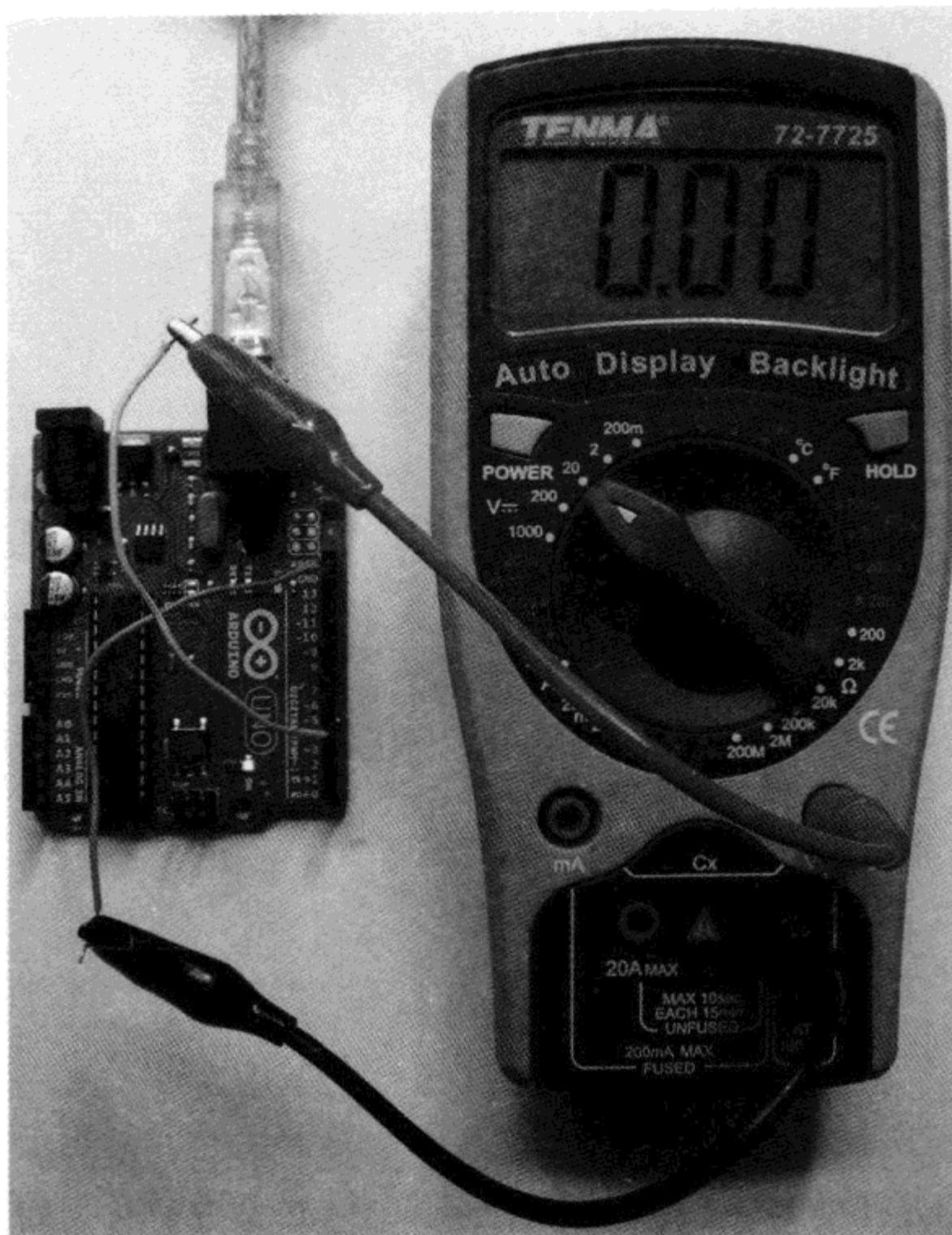
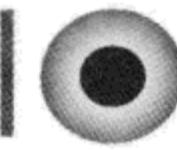


图6.1 用万用表测量输出

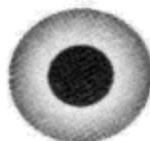
端连接探针，一端插到Arduino板的插母。

上传sketch 6-01：

```
//sketch 6-01

int outPin = 4;

void setup()
```

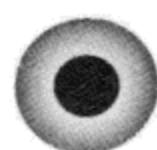


```
{  
    pinMode(outPin, OUTPUT);  
    Serial.begin(9600);  
    Serial.println("Enter 1 or 0");  
}  
  
void loop()  
{  
    if (Serial.available() > 0)  
    {  
        char ch = Serial.read();  
        if (ch == '1')  
        {  
            digitalWrite(outPin, HIGH);  
        }  
        else if (ch == '0')  
        {  
            digitalWrite(outPin, LOW);  
        }  
    }  
}
```

在Sketch的开头，你可以看到pinMode命令。你可以在项目中对用到的所用针使用这一命令，让Arduino可以将连接到针脚的电子器件配置为输入或输出，如下面的例子：

```
pinMode(outPin, OUTPUT);
```

可能和你猜想的一样，pinMode是一个内建函数。它的第一个参数是涉及的针脚号，一个整数；第2个参数是使用模式，其值必须为INPUT（输入）或者OUTPUT（输出）。注意：模式名必须全部大写。



loop函数等待从串口监视器发过来的1或0的指令。若为1，针脚4被打开；为0，则关闭。

将Sketch上传至Arduino，然后打开串口监视器，如图6.2所示。

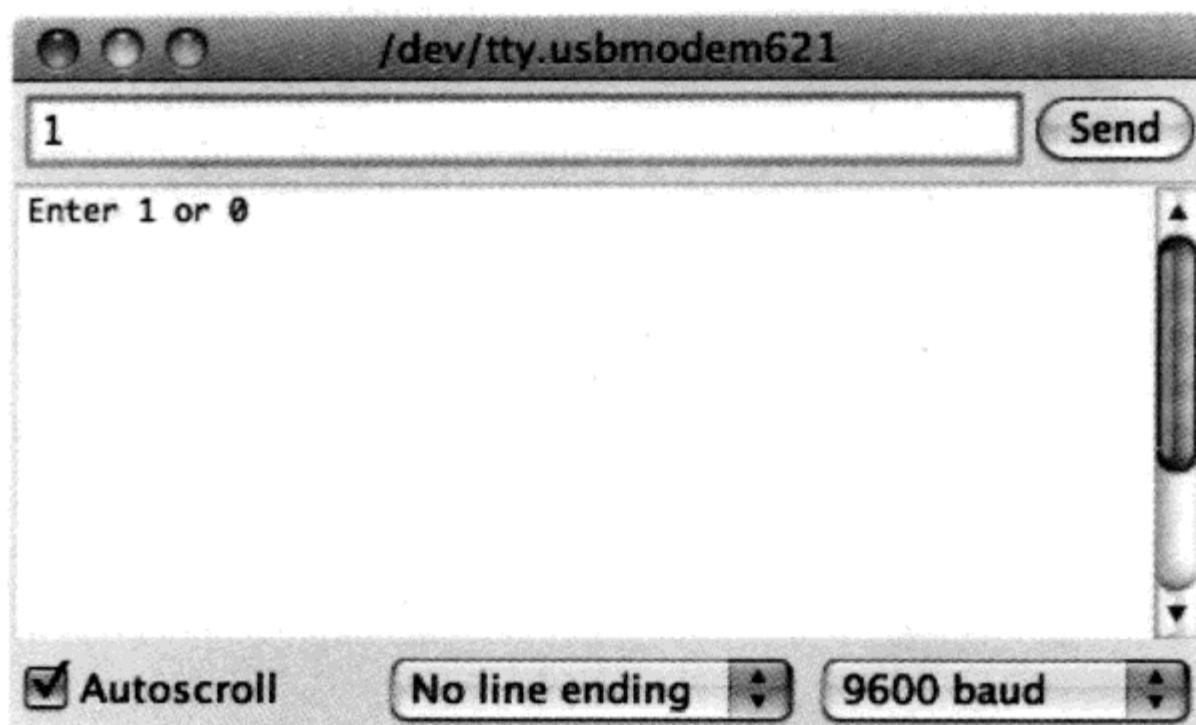


图6.2 串口监视器

因为已经把万用表打开并连接至Arduino，所以当你在串口监视器中输入0或1并回车后，会看到万用表上的读数在0V和5V之间变化。图6.3为从串口监视器上发送1后万用表的读数。

如果板上标有D的针脚不够用了，其实也可以使用标有A（模拟）的针脚。如果需要这样做的话，你只需在模拟针脚的编号上加上14。你可以通过修改sketch 6-01来使用14号针脚，并将万用表的正极探针连接到A0针脚来试验一下。

上面就是关于数字信号的输出了，接下来我们来看看数字信号输入。

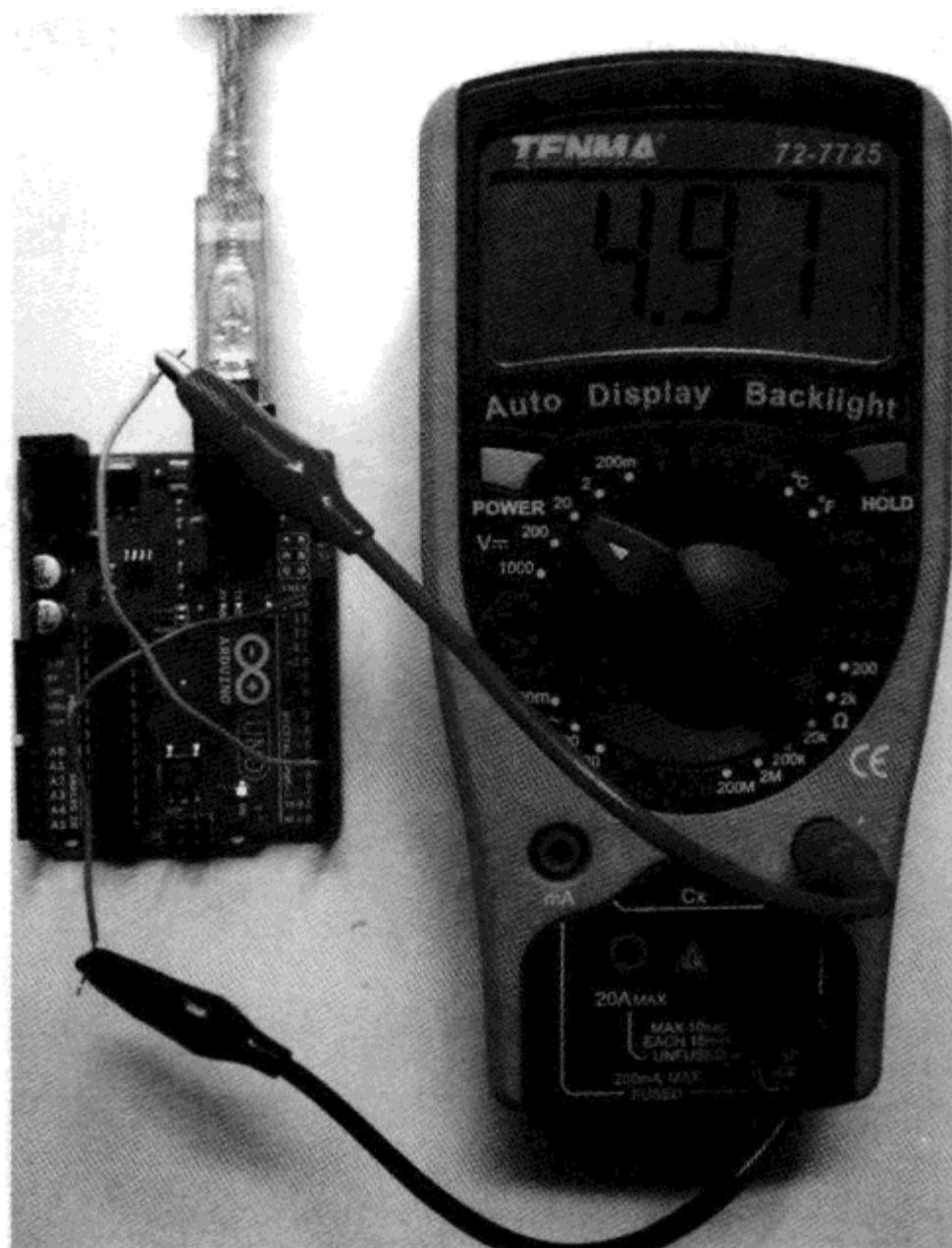


图6.3 将输出设为高电平

6.2 数字输入

数字输入最常见的用途是检测某个开关是否被关闭。数字输入值可以为开或关。如果输入的电压小于2.5V（5V的一半），则为0（关）；若大于2.5V，则为1（开）。

拔掉万用表，并将sketch 6-02上传至Arduino：

```
//sketch 6-02  
  
int inputPin = 5;
```



```
void setup()
{
    pinMode(inputPin, INPUT);
    Serial.begin(9600);
}

void loop()
{
    int reading = digitalRead(inputPin);
    Serial.println(reading);
    delay(1000);
}
```

用于使用的是一個輸出針腳，所以需要在**setup**函數中告訴Arduino把这个針作為輸入來使用。通過使用函數**digitalRead**來得到數字輸入的值，返回0或1。

上拉電阻

Sketch以每秒1次的頻率讀取輸入針腳並將讀取的值寫到串口監視器里。那麼上傳Sketch並將串口監視器打開，你會看到一個每秒顯示1次的值。將一截導線的一頭插入D5接口，並用手捏住另一頭。如圖6.4，保持幾秒鐘來看看串口監視器上顯示的內容。你會發現數值在1和0之間不斷變化。原因是Arduino板對輸入是相當敏感的，你這時候就相當於一根天線——收集電子干擾的天線。

將原本捏在你手中的那一端插入+5V接口（圖6.5），串口監視器中的數值將變為持續的1。

|| 输入和输出

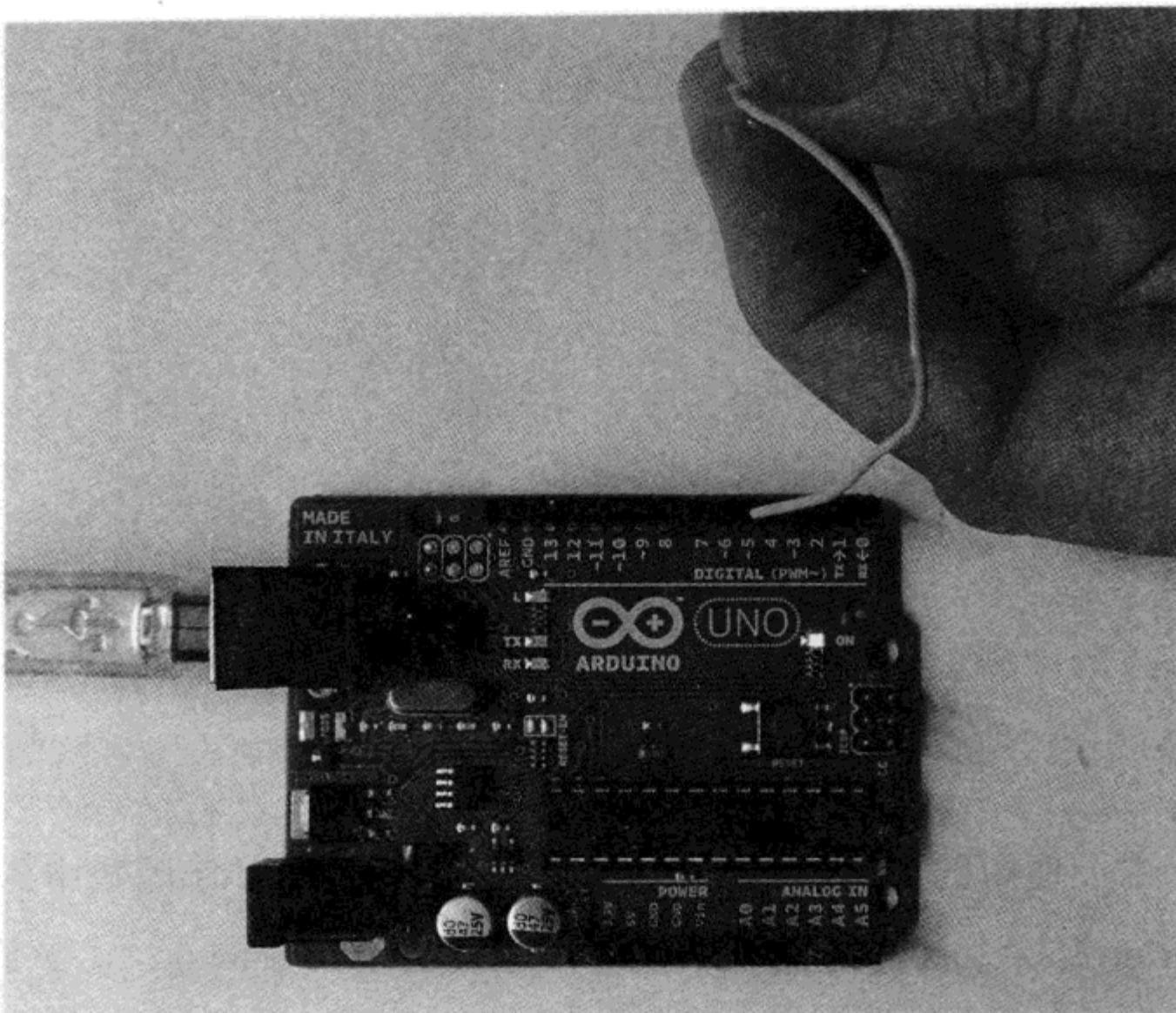


图6.4 用人体做天线的数字输入

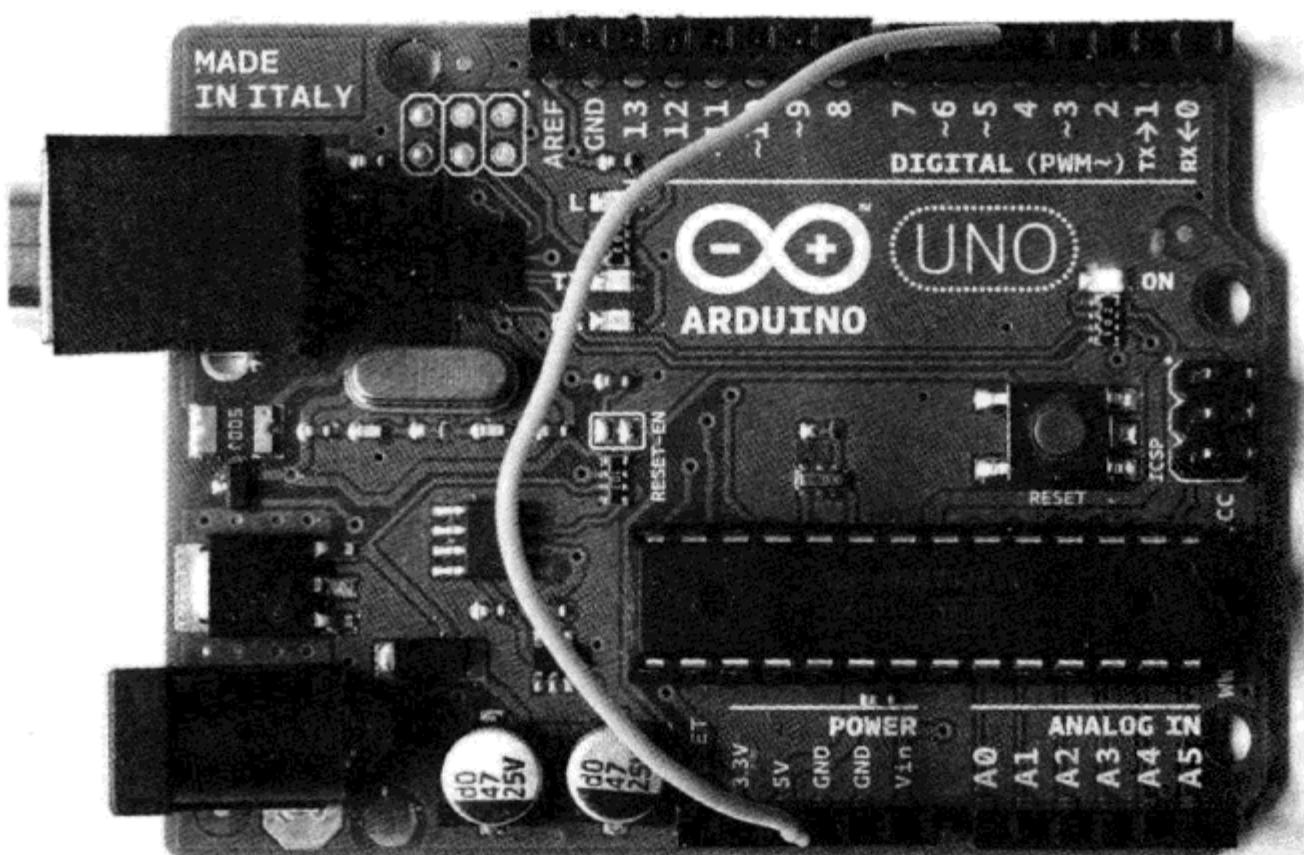
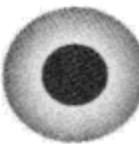


图6.5 将D5连接至+5V



再将插在+5V的线头插到GND中，如你想象的一样，串口监视器中现在显示的是0了。

输入针脚的一个典型的用法是将它连至一个开关，图6.6可能正是你所猜想的连接方式。

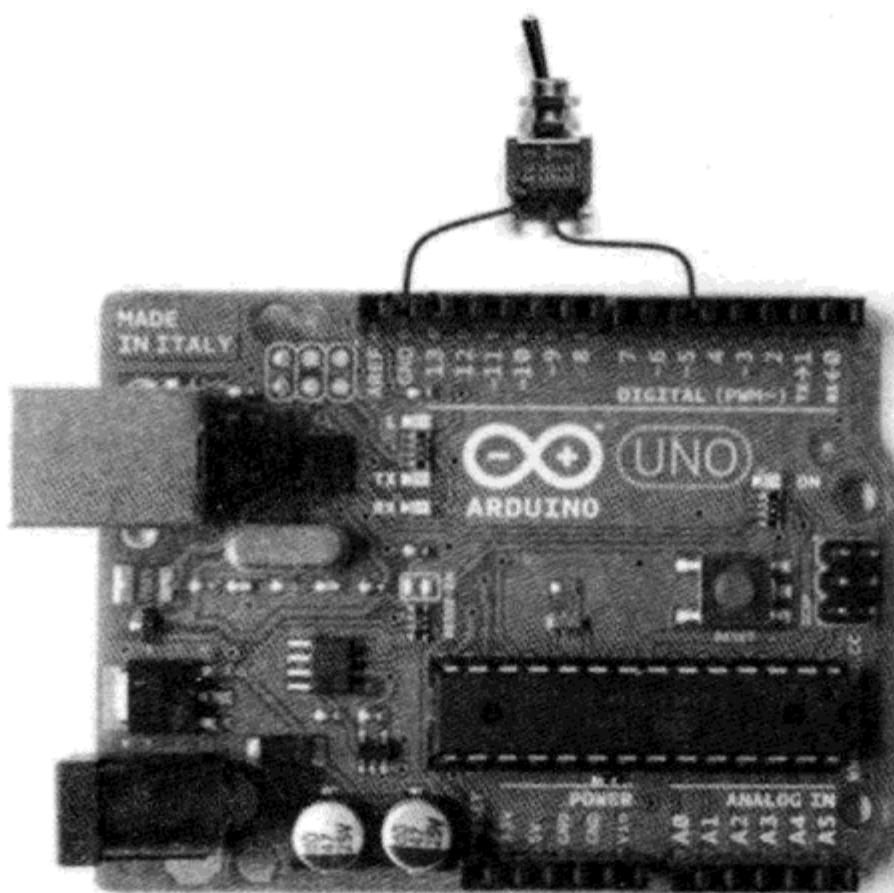


图6.6 将开关连至Arduino

这种做法的问题在于，若开关没有闭合的话，输入针脚就没有连接任何东西，这叫做“悬空”，很有可能会造成错误的读取。你的输入需要更加可靠，解决办法是一种叫做上拉电阻的东西。图6.7为上拉电阻的标准使用方法。它的效果是，在开关没有闭合的情况下，电阻将输入上拉至5V；当你按下开关使其闭合的时候，上拉电阻被开关短路，电阻不起作用，让输入变为0V。这个方法有一个副作用，当开关闭合时，5V将通过电阻产生电流。所以电阻的阻值要选得足够小，使其不会受任何电子干扰的影响；同时电阻也要选得足够大，以使开关闭合时电流不会过大。

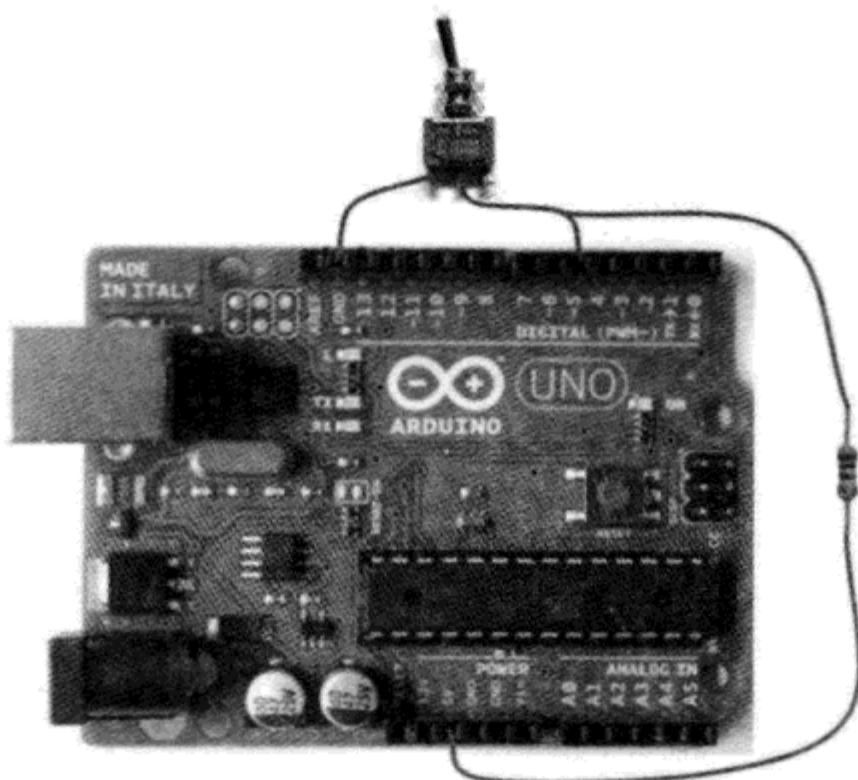


图6.7 开关和上拉电阻

内部上拉电阻

幸运的是，Arduino板载数字针脚中内置了可以用软件调节的上拉电阻。默认情况下，它们是被关闭的。如果你想让D5的上拉电阻生效，只需在sketch 6-02中加上如下一行：

```
digitalWrite(inputPin, HIGH);
```

这行放在setup函数中紧接着将针脚定义为输入的那行之后。可能对输入针脚使用digitalInput看起来有点怪，可它原本就是这么用的。

sketch 6-03是修改过的版本，将它上传至板并再做一次人体天线测试，你会发现这一次串口监视器上的数值保持在1了。

```
//sketch 6-03  
  
int inputPin = 5;
```



```
void setup()
{
    pinMode(inputPin, INPUT);
    digitalWrite(inputPin, HIGH);
    Serial.begin(9600);
}

void loop()
{
    int reading = digitalRead(inputPin);
    Serial.println(reading);
    delay(1000);
}
```



消抖

当你按下按钮的时候，可能期待的结果就是从1到0（从上拉电阻到按钮按下）的变化。图6.8为按下按钮时有可能发生的情况。按钮中的金属触点会发生抖动，所以简单地按下按钮其实是一连串的按下动作。

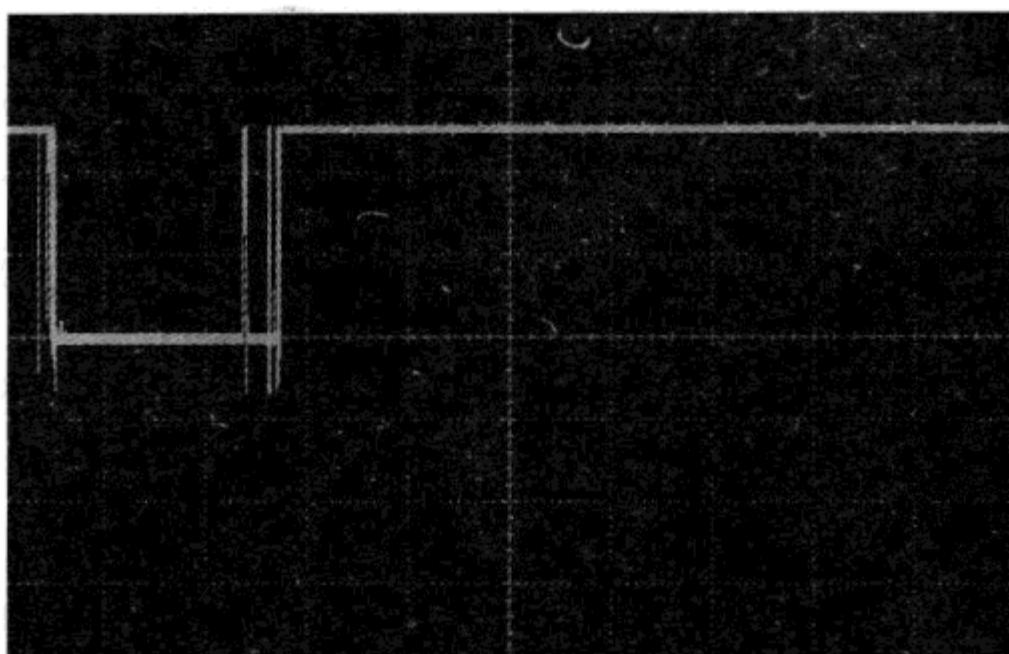
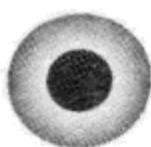


图6.8 按下按钮时示波器跟踪到的波形



这一切都发生得很快，在示波器上显示的时间跨度仅仅为200ms。这种情况通常发生在老式触点式开关上，新式触摸式或微动开关可能根本不会产生抖动。

有时，抖动根本不碍事。举例来说，sketch 6-04将会在按下开关的时候点亮LED，现实中根本不需要用Arduino来做这件事，这里我们只看看理论部分。

```
//sketch 6-04

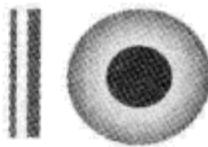
int inputPin = 5;
int ledPin = 13;

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(inputPin, INPUT);
    digitalWrite(inputPin, HIGH);
}

void loop()
{
    int switchOpen = digitalRead(inputPin);
    digitalWrite(ledPin, ! switchOpen);
}
```

看一下sketch 6-04中的loop函数，函数读取数字输入并将其值赋予变量switchOpen。如果按下按钮时值为0，则不按时为1（针脚在不按按钮的时候被上拉为1）。当你用digitalWrite编程控制LED的时候，需要用!或者说非运算符来翻转这个值。

将此Sketch上传并且按图6.9连接到D5和GND的话，会看到



LED点亮。这时是有可能发生抖动的，但它发生得太快，以至于你看不到，而且根本没有影响。

在一种情况下按钮抖动会产生影响，就是使用开关来切换LED的亮暗。也就是说，按下按钮时，LED被点亮并保持长亮；若再按一次按钮，则关闭LED。如果使用可能产生抖动的按钮，那么LED的开或关就取决于抖动的次数是奇数还是偶数了。

sketch 6-05仅仅为了切换LED的亮暗，并未做任何消抖的尝试，用导线代替开关试一下（图6.9）：

```
// sketch 6-05

int inputPin = 5;
int ledPin = 13;
int ledValue = LOW;

void setup()
{
    pinMode(inputPin, INPUT);
    digitalWrite(inputPin, HIGH);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    if (digitalRead(inputPin) == LOW)
    {
        ledValue = ! ledValue;
        digitalWrite(ledPin, ledValue);
    }
}
```

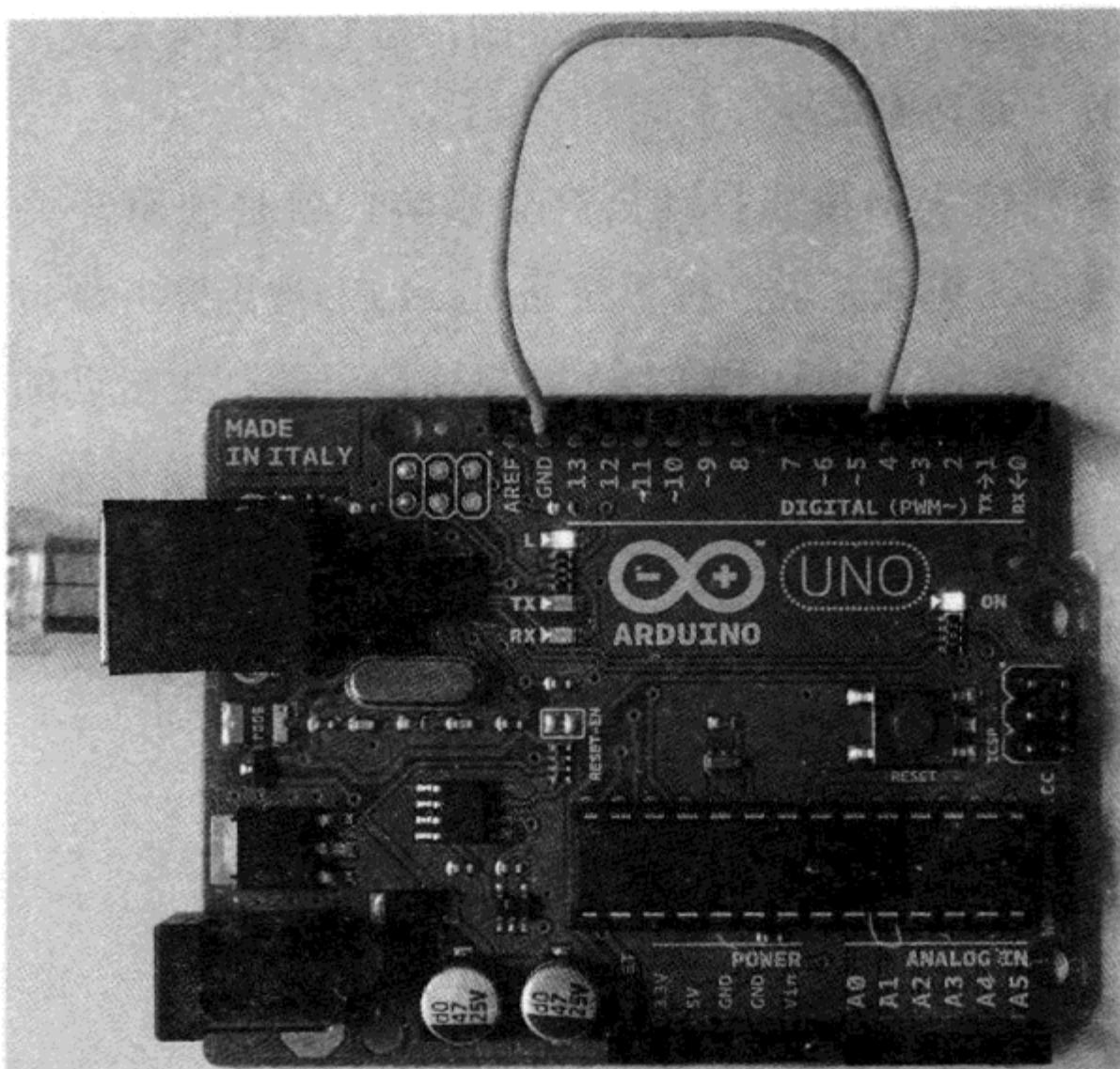


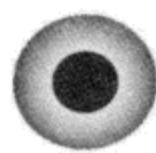
图6.9 用导线代替开关

你可能会发现，有的时候LED切换了状态，有的时候却没有。这就是抖动造成的。

解决这一问题的一个简单的办法是，在检测到第一次按钮按下之后加上一个延迟，如sketch 6-06：

```
// sketch 6-06
int inputPin = 5;
int ledPin = 13;
int ledValue = LOW;

void setup()
{
    pinMode(inputPin, INPUT);
```



```
digitalWrite(inputPin, HIGH);
pinMode(ledPin, OUTPUT);
}

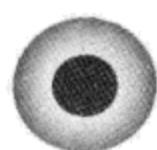
void loop()
{
    if (digitalRead(inputPin) == LOW)
    {
        ledValue = ! ledValue;
        digitalWrite(ledPin, ledValue);
        delay(500);
    }
}
```

加了一个延迟以后，在500ms之内不会发生任何事，这么长的时间无论抖动多少次都足够了。你会发现，这让切换变得可靠多了。一个有趣的副作用是，如果你把按钮按住不松的话，LED会不停地闪。

如果这就是Sketch的全部，这个延迟当然没什么问题。但是如果你在loop中做更多的事情，那么使用延迟就有问题了。例如，程序在接下来的500ms里将不会检测到其他任何按钮动作。

所以这个方法有的时候并不够好，你需要更全面的方法。你可以自己亲手写一些更高级的消抖代码，但这么做太复杂。幸运的是，一些好心人已经帮你做好了这一切了。

若要利用他们的成果，你需要在Arduino软件中添加一个库。先将库的zip文件从<http://www.arduino.cc/playground/Code/Bounce>下载下来。



下载完之后解压缩至文件夹“Bounce”，并放置在你所有Sketch的文件夹的“libraries”子文件夹中。在Windows下，这个文件夹是“Mydocuments\Arduino”，Mac和linux下则为“Documents/Arduino”。如果里面本来没有“libraries”子文件夹，则需要自己创建一个。图6.10笔者在Windows下添加完库之后的文件夹结构。

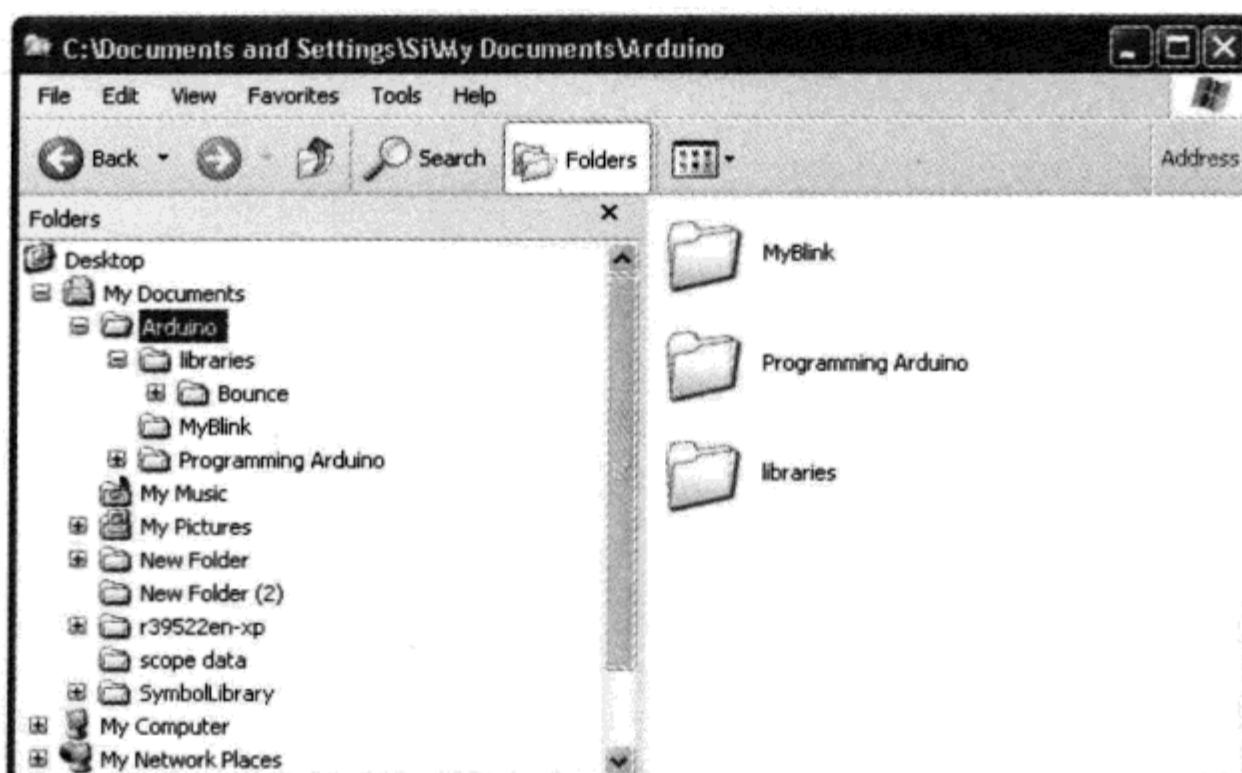


图6.10 在Windows下添加Bounce（抖动）库

库添加完以后，Arduino软件需要重启才能生效。然后，你就可以在写任何Sketch的时候使用Bounce库了。

sketch 6-07为Bounce库的使用方法。上传到板，来看看LED的切换是不是变得可靠了。

```
//sketch 6-07
#include <Bounce.h>
int inputPin = 5;
int ledPin = 13;
```

```
int ledValue = LOW;
Bounce bouncer = Bounce(inputPin, 5);

void setup()
{
    pinMode(inputPin, INPUT);
    digitalWrite(inputPin, HIGH);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    if (bouncer.update() && bouncer.read() == LOW)
    {
        ledValue = ! ledValue;
        digitalWrite(ledPin, ledValue);
    }
}
```

库的使用挺简单的。你可能最先注意到的是下面这行：

```
#include <Bounce.h>
```

这是用来告诉编译器使用Bounce库，然后是下面这行：

```
Bounce bouncer = Bounce(inputPin, 5);
```

现在先不要担心这一行的语法。这其实是C++的而不是C的语法，你将在第11章才能接触到C++。你现在只需要知道这是给指定的针脚创建了一个Bouncer对象，并有时长为5ms的消抖期就可以了。

现在开始，通过Bouncer对象找出程序在没有直接读取数字输入的这段时间里做了什么。它围绕针脚放置了一个消抖包装，



那么决定一个按钮是否被按下就被包装在下面这一行中了：

```
if (bouncer.update() && bouncer.read() == LOW)
```

函数update在Bouncer对象有变化的时候返回一个“真”值，条件的第二部分则是用来检视按钮是不是为低电平。

6.3 模拟输出

一部分数字针脚——数字针脚3、5、6、9和11可以提供0V和5V以外的可变输出。这些针脚在板上都被标有“~”或者是“PWM”。PWM代表脉冲宽度调制，旨在控制输出的功率，通过快速地开关输出来实现。

脉冲是一直以固定的频率送达的（大概500次/s），但脉冲的长度是变化的。如果脉冲很长，LED会一直亮；但如果脉冲的长度很短，LED也只会被点亮很短的时间。这一切都发生得太快了，以至于观察者根本不能看出LED在闪烁，只能看出LED的亮度变化。

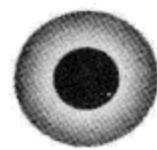
在你用LED试验前，你可以用万用表来测试。将万用表接在GND和D3之间（图6.11）。

现在，将sketch 6-08上传到板上，打开串口监视器（图6.12），输入数字3并回车，你会看到电压表读数为3V。你可以试试0~5之间的任何数。

```
//sketch 6-08

int outputPin = 3;

void setup()
```



```
{  
    pinMode(outputPin, OUTPUT);  
    Serial.begin(9600);  
    Serial.println("Enter Volts 0 to 5");  
}  
  
void loop()  
{  
    if (Serial.available() > 0)  
    {  
        char ch = Serial.read();  
        int volts = (ch - '0') * 51;  
        analogWrite(outputPin, volts);  
    }  
}
```

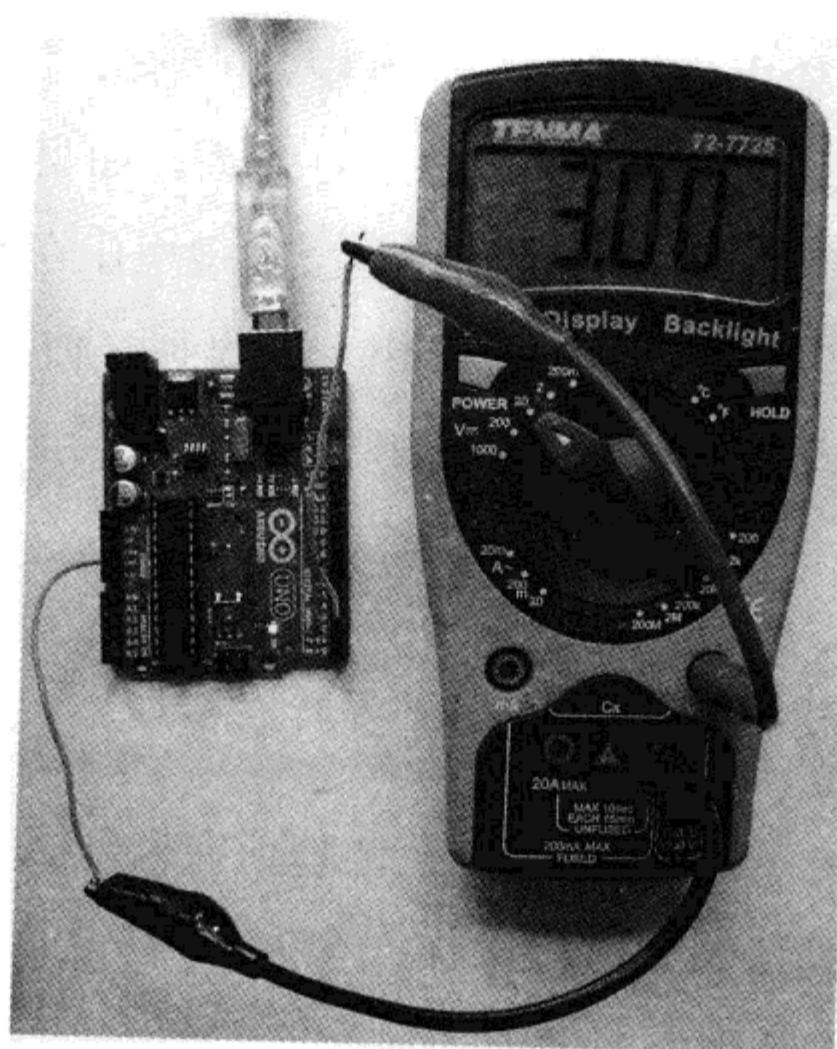


图6.11 测量模拟输出

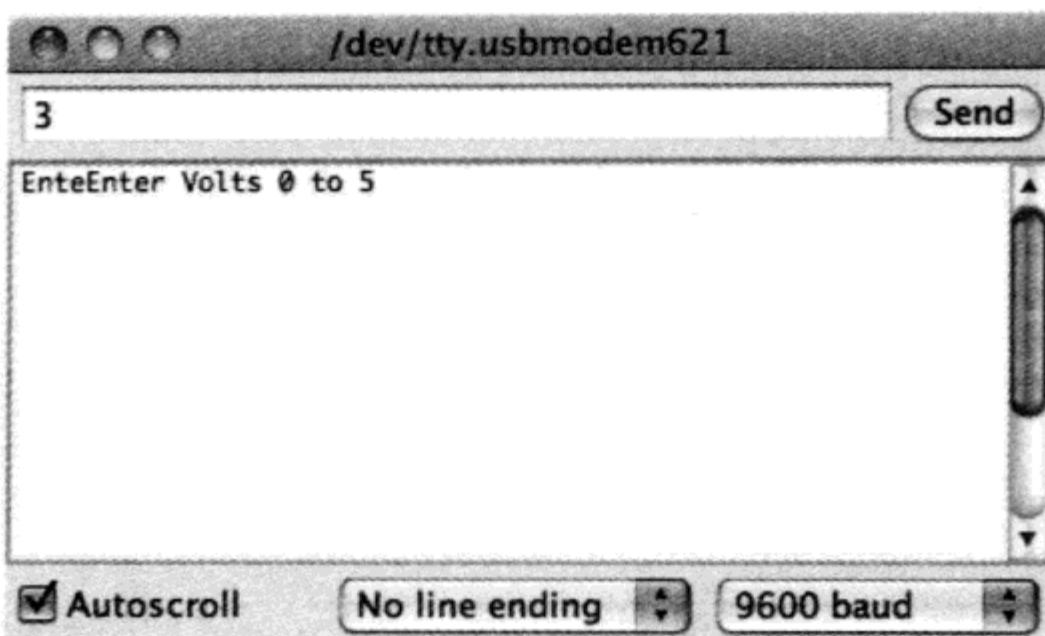


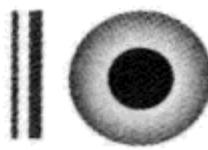
图6.12 设置一个模拟输出电压

程序通过将期待电压值（0~5V）乘以51来决定一个0~255之间的PWM输出值。（若想了解更多关于PWM的东西，请查阅维基百科）。

你可以使用函数analogWrite来设定输出值。函数需要一个0~255之间的输出值作为参数，其中0为关闭，255为全功率。这其实是控制LED亮度的一个很好的办法。如果你想通过改变LED上的电压来控制其亮度的话，你会发现电压在2V之前LED根本没反应，过了2V之后LED会很快变得很亮。使用PWM来控制亮度，则是改变LED被点亮的平均时长，你会得到对亮度更为线性的控制。

6.4 模拟输入

数字输入对于Arduino板上某针脚正在做什么只能给出开/关的答案。但是模拟输入可以给你一个0~1023之间的值，这个值取决于模拟输入针脚上的电压。



程序通过analogRead函数来读取模拟输入。sketch 6-09在串口监视器中显示，模拟针脚A0上每秒2次读取的也是实际的电压值。所以，打开串口监视器来看看读数。

```
//sketch 6-09

int analogPin = 0;

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int reading = analogRead(analogPin);
    float voltage = reading / 204.6;
    Serial.print("Reading=");
    Serial.print(reading);
    Serial.print("\t\tVolts=");
    Serial.println(voltage);
    delay(500);
}
```

运行这个Sketch，你会发现读数在不停地变化。就像数字输入一样，现在输入是浮动的。

将导线的一端插入GND，将A0连接至GND，现在你的读数应该稳定在0V了。将原本插在GND的那一端插入5V，你会得到1023的读数，也就是最大读数。那么，如果你将A0连接至3.3V，这个Arduino电压表会告诉你大致有3.3V的电压。



6.5 总 结

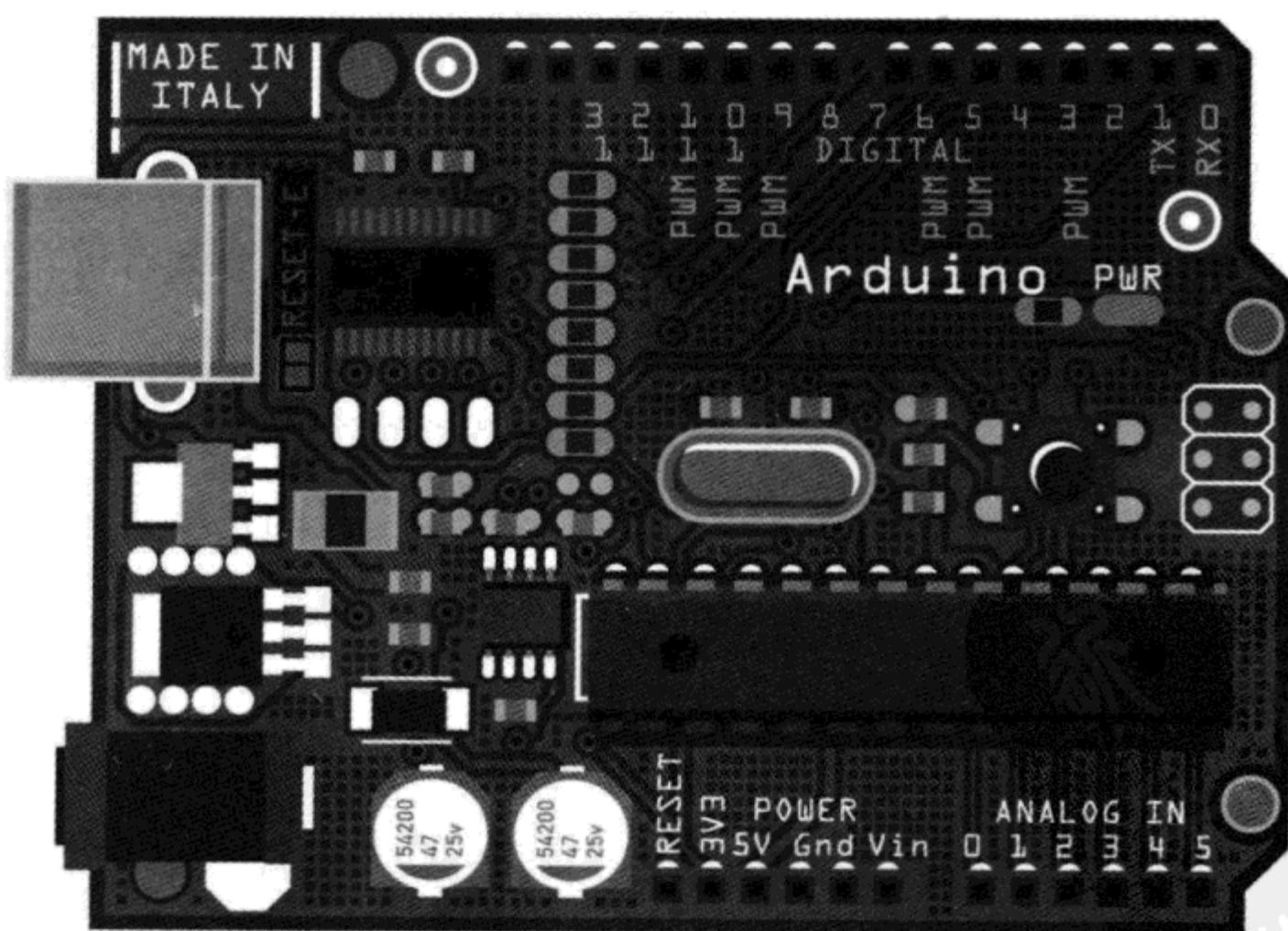
本章主要是将信号输入Arduino，或从Arduino输出。下一章，我们将看看标准Arduino库可以提供什么功能。



第7章

标准Arduino库

The Standard Arduino Library



电子
技术
PDG



这个库是所有好用的东西的集合。C语言可以只学到这里，你真正需要的是大量可以用在Sketch中的函数。

你已经学了一些函数，如pinMode、digitalWrite和analogWrite，但实际上远不止这些。还有其他函数可以用来做数学运算、生成随机数、位操作、检测一个输入针脚上的脉冲，还可以使用一种叫做“中断”的东西。

Arduino语言基于一个早期名为Wiring的库，并补充了另一个叫做Processing的库。Processing库和Wiring库很相似，但它是基于Java语言的，而不是C语言，在电脑上使用USB连接至Arduino。实际上，你在电脑上使用的Arduino软件就是基于Processing的。如果你想自己在电脑写一些华丽的接口来和Arduino对话，可以学习一下Processing（www.processing.org）。

7.1 随机数

尽管在每个使用者的经验中，计算机实际上是非常有可预见性的偶尔，我们也需要故意让你的Arduino变得不可预知。例如，你可能会想让一个机器人选择一条随机的路线来在房间中行走：在一个方向上行走随机的长度，然后转一个随机的角度再行走，最后关闭。或者，你考虑用Arduino做一个轮盘赌，可以从1~6之间随便抽一个数给你。

标准Arduino库为做这些提供了方法，那就是random函数。random返回一个整数，而且它可以有1个或2个参数。如果它只有1个参数，则会返回一个0~（参数-1）之间的随机数；如果有2个



参数，那么随机数将会从第1个参数（包含）~（第2个参数-1）之间抽取。如random(1, 10)，则可以产生一个1~9之间的随机数。

sketch 7-01在串口监视器中随机出现1~6之间的数：

```
//sketch 7-01

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int number = random(1, 7);
    Serial.println(number);
    delay(500);
}
```

如果你将这个Sketch上传到Arduino并打开串口监视器，你会看到类似图7.1的景象。

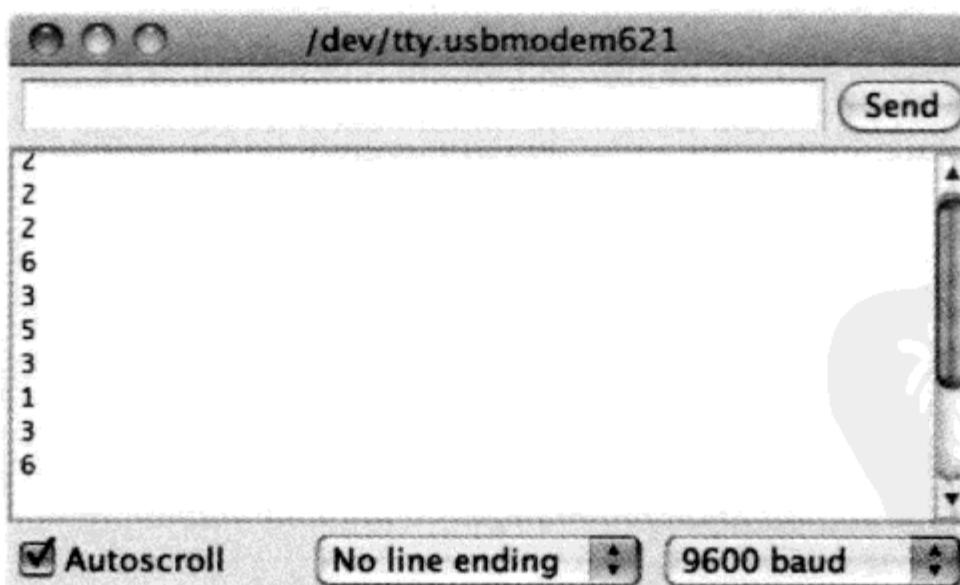


图7.1 随机数

如果你把这个Sketch运行几遍，你可能会惊讶地发现：每次



都会出现同一串“随机”数。

其实输出并不是真的随机的。这些数字被称为“伪随机”数，因为它们只是随机分布的。也就是说，如果你运行这个Sketch并得到100万个数字，得到的不外乎还是一些1、一些2和一些3等等。数字的随机并不旨在不可预见，实际上真正的随机是有悖于单片机工作原理的，它不可能在没有外界干预的情况下做到这一点。

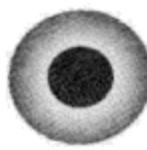
你可以主动提供这一干预，通过给随机数生成器“播种”来让随机数变得不那么可预见。这也就是说，为生成的序列提供一个起始点。但如果你动动脑子就会想到，你不能简单地用“随机”来作为生成随机数的起始点。一个常用技巧是利用模拟输入的浮动（前一章提到过），所以你可以用读取到的模拟输入的值来为随机数生成器“播种”。

实现这一功能的函数名为randomSeed。sketch 7-02告诉你怎样为随机数生成器再增加一些随机性。

```
//sketch 7-02

void setup()
{
    Serial.begin(9600);
    randomSeed(analogRead(0));
}

void loop()
{
    int number = random(1, 7);
    Serial.println(number);
```



```
delay(500);
}
```

试试多按几次重置按钮，你现在可以看到随机数序列每次都不同了。

这种类型的随机数生成不可能用于任何形式的彩票，若要有更好的随机数生成，你需要使用硬件随机生成。它们有一些是基于随机事件的，例如：宇宙射线。

7.2 数学函数

在极少数的情况下，你可能需要在Arduino上做大量的数学运算，而不仅仅是那么一点点算术。但当你需要的时候，有一个巨大的数学函数库来供你使用。最常用的函数在表7.1中被列出：

表7.1

函 数	描 述	例 子
abs	返回参数的绝对值	abs(12) returns 12 abs(-12) returns 12
constrain	压缩数字以防止溢出范围，第1个参数为被压缩的数，后2个参数为范围	constrain(8, 1, 10) returns 8 constrain(11, 1, 10) returns 10 constrain(0, 1, 10) returns 1
map	将一个数所在的范围映射到另一个更大的范围，第1个参数是需要映像的数字，第2个、第3个参数是原本的范围（源范围），最后2个参数是被映射到的范围（目标范围）。这个函数用于重新映射模拟输入的值	map(x, 0, 1023, 0, 5000)
max	返回2个参数中较大的那个	max(10, 11) returns 11
min	返回2个参数中较小的那个	min(10, 11) returns 10
pow	返回第1个参数的第2个参数幂方	pow(2, 5) returns 32
sqrt	返回参数的平方根	sqrt(16) returns 4
sin,cos,tan	三角函数，不常用	
log	例如：通过对数热敏电阻计算温度	



7.3 位操作

位是2进制信息的最小数位，也就是0或1。单词bit是2进制位（Binary digit）的缩写。大部分时候你所使用的整数变量实际上包含16位，如果你只需存储一个简单的真或假（1或0）的值，就浪费了很多位。事实上，除非你的内存不够用了，与位的浪费相比较写一个难以理解的程序来说不算什么，但是有些时候给你的数据瘦身也是相当有必要的。

整数中的每一位都可以看做是有一个对应的10进制值，而且你可以将所有为1的位对应的10进制值相加，得到整个整数的10进制值。所以在图7.2中，整数的10进制为38。实际上，处理负数的时候会变得更加复杂，但这只发生在当最左边的一位为1的时候。

16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

$$32 + 4 + 2 = 38$$

图7.2 一个整数

当你考虑单独的某些位的时候，10进制数值就变得不那么好用了。你很难看出10进制中的位对应的是2进制中的哪些位，如123。因此，程序员经常使用一种叫做16进制（Hex）的方法。16进制是基于16的数字。也就是说，相比十进制的0~9来说，你还有6个额外的位A~F。这意味着每个16进制数位都对应4个2进制位。表7.2为0~15的十进制、十六进制和二进制对照表。



表7.2

10进制	16进制	2进制（4位）
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

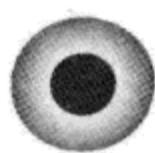
16进制中，任何整数都可以用一个4位的2进制数来表示，如2进制数10001100在16进制中为8C。C语言在使用16进制数的时候有一个特殊的语法，你可以这样把一个16进制赋值给一个整数变量：

```
int x = 0x8C;
```

标准Arduino库提供了一些函数，让你可以单独操作一个整数中16位中的某一位。函数bitRead返回的是整数中16位的某一个指定的位的值。下面的例子将把值“0”赋给名叫bit的变量。

```
int x = 0x8C; // 10001100
int bit = bitRead(x, 0);
```

它的第2个参数是位的编号，从0开始到15。从最小的一位开始，所以最右边的一位是0号位，左边相邻的则为1号位，以



此类推。

如你所期待的，对应bitRead的还有一个bitWrite。它有3个参数，第1个为要操作的数，第2个是位编号，第3个为位的值。下面的例子中，将整数的值从2变为3（10进制或者16进制）。

```
int x = 2; // 0010  
bitWrite(x, 0, 1);
```

7.4 高级输入/输出

当处理较多的输入/输出任务的时候，有一些小函数可以让事情变得简单。

声音生成

tone函数可以在一个输出针脚产生一个方波信号，如图7.3所示。这样做的最常见目的是用扬声器或者蜂鸣器来发出一个特定声调的声音。

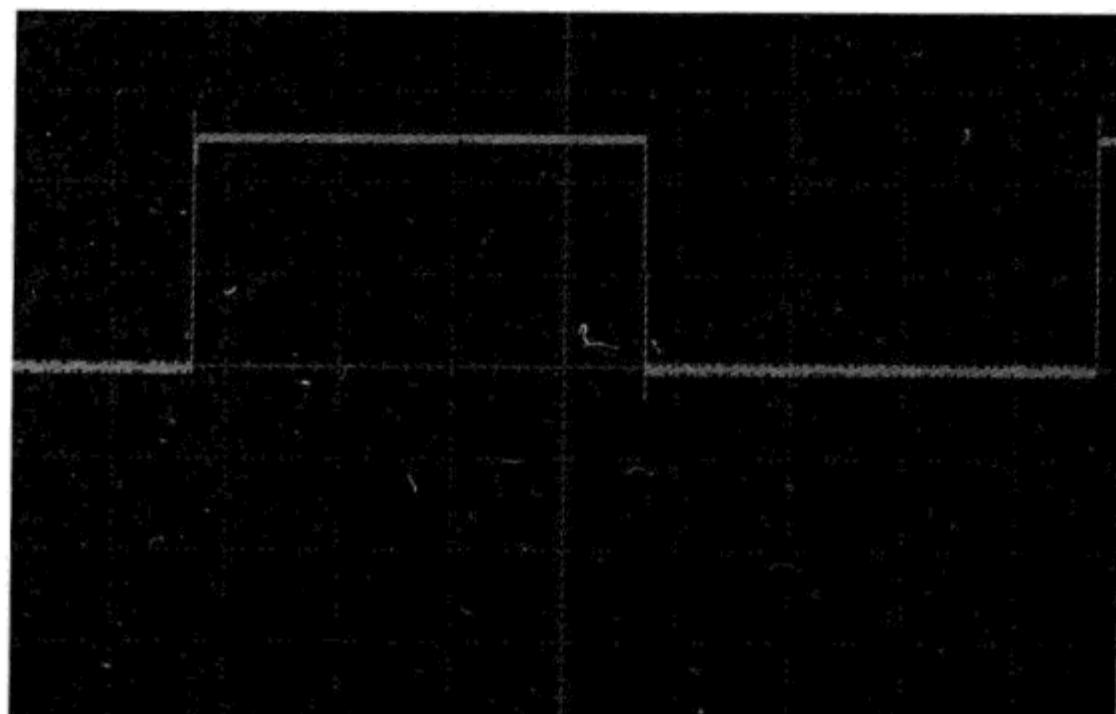
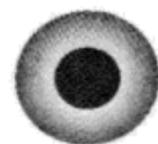


图7.3 方波信号



这个函数有2个或3个参数。第1个参数当然是用来发出声音的针脚；第2个为声音的频率，单位为赫兹（Hz）；第3个可选的参数为声音的长度。如果不指定长度的话，声音会一直响下去，如sketch 7-03的情况。这也是为什么你要用setup来调用函数tone，而不是用loop来调用的原因。

```
//sketch 7-03

void setup()
{
    tone(4, 500);
}

void loop() {}
```

若要让声音停止，你需要使用函数noTone。这个函数只有一个参数，即声音所在的针脚。

读取移位寄存器

有时候Arduino UNO的针脚不够用，如需要驱动很多LED的时候。有一个简单的技术是使用移位寄存器芯片，这个芯片每次读取一位数据，当读取足够的时候，它将这些位全部锁存到输出针脚上去（每次一位）。

为了帮助你使用这个技术，有一个工具函数名为shiftOut，这个函数有4个参数：

- (1) 位需要发送到的针脚号；
- (2) 被用作时钟的针脚号，每发送一位后被触发一次；
- (3) 一个用于决定位是从最小的一位开始发送还是从最大



的一位开始发送的标签，必须是常量MSBFIRST或LSBFIRST中的一个；

(4) 发送数据的字节数。

7.5 中 断

一件可能让那些写“大程序”的程序员灰心的事情是，Arduino一次只能做一件事情，如果想在你的程序中同时运行多个线程，是不行的。虽然有些人开发出了可以让Arduino在某些方面使用多线程的项目，但一般来说这种功能对于Arduino一般的用途来说没有必要。Arduino所使用的近似方法是中断。

Arduino上的2个针脚（D2和D3）可以附加中断。如果这两个针脚作为输入使用，通过某个特殊的方式收到信号的时候，Arduino处理器会停止正在做的任何事情并执行中断函数。

sketch 7-04在闪烁LED，但是当收到中断的时候会改变闪烁的时长。你可以把D2和GND用导线连接起来模拟1次中断，并且使用内置上拉电阻让中断在多数时候保持高电平。

```
//sketch 7-04

int interruptPin = 2;
int ledPin = 13;
int period = 500;

void setup()
{
    pinMode(ledPin, OUTPUT);
```

```
pinMode(interruptPin, INPUT);
digitalWrite(interruptPin, HIGH); // pull-up
attachInterrupt(0, goFast, FALLING);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(period);
    digitalWrite(ledPin, LOW);
    delay(period);
}

void goFast()
{
    period = 100;
}
```

下面是这个Sketch中setup的关键行：

```
attachInterrupt(0, goFast, FALLING);
```

第1个参数指定了你想要使用哪个中断。有点绕，这里0意味着你要使用D2，而1意味着你要使用D3。

下一个参数是中断时被调用的函数名，最后一个参数是常量CHANGE、RISING和FALLING中的一个，图7.4为这些选项的示意图。

如果中断类型是CHANGE，那么无论是从0→1的RISING还是从1→0的FALLING都会触发一次中断。

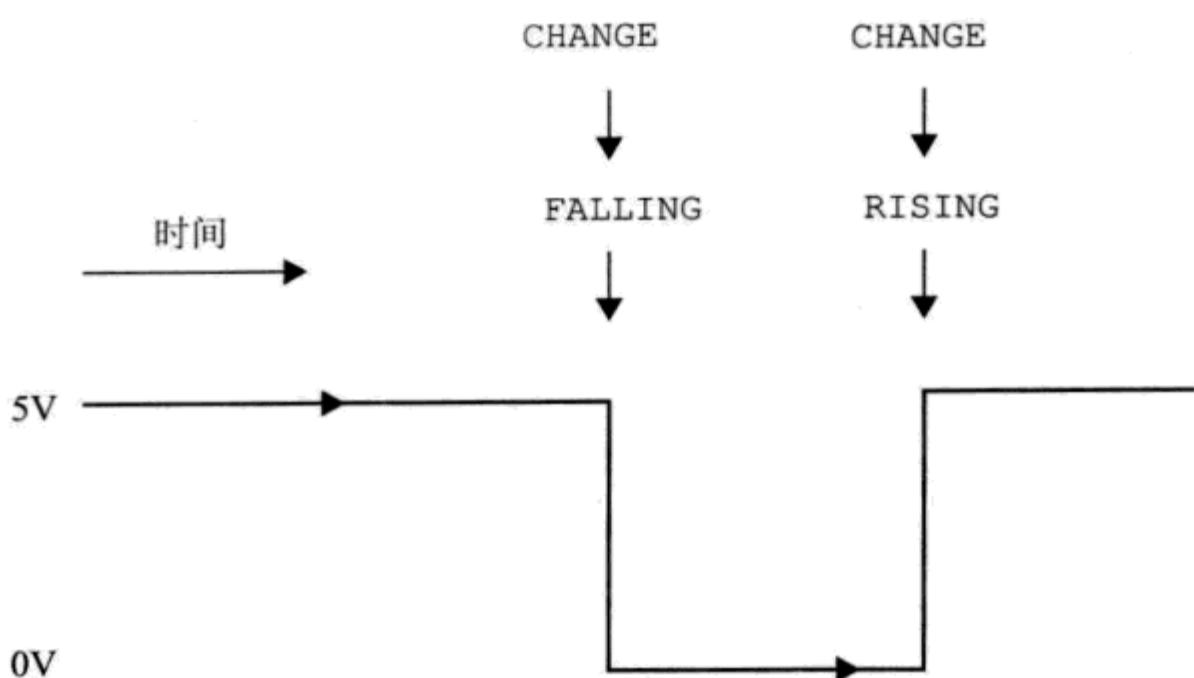


图7.4 中断信号的类型

你可以用函数`noInterrupts`来禁用中断。这将在所有的中断通道停止所有的中断。你可以通过再次调用函数`interrupts`来恢复中断。

7.6 总 结

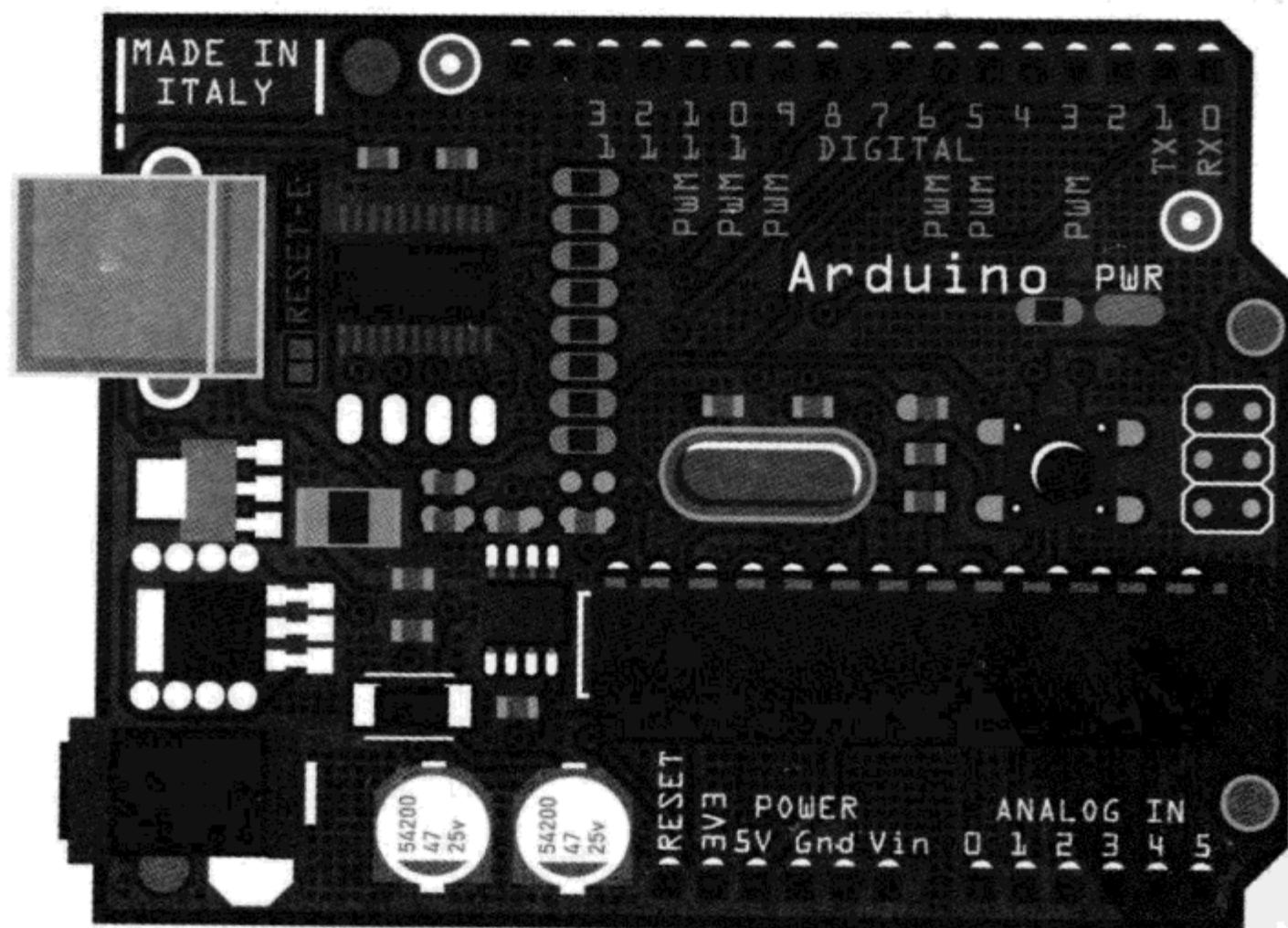
本章学习了一些标准Arduino库所提供的一些方便的功能，这些功能可以节省你在编程上做所花的时间。程序员最喜欢的就是使用别人已经完成的高质量的工作。

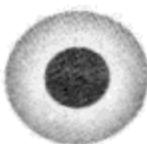
下一章中，我们将扩展在第5章学到的数据结构，并看一看怎样在Arduino关闭的时候将数据保存在其中。

第8章

数据存储

Data Storage





当你给变量赋值的时候，Arduino只会在电源接通的时候记得这些值。当你关闭电源或者按下重置按钮的时候，所有的数据都会丢失。

在这一章里，我们看一看一些保存数据的方法。

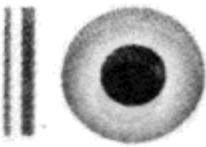
8.1 常量

如果想要你存储的数据不会改变，那么你只能在Arduino每次启动的时候重新设置数据。这种方法的一个实例是第5章莫尔斯码翻译器中的字母数组（sketch 5-05）。

试着使用下面的代码来定义一个正确大小的变量，并且用你需要的数据来填充它。

```
char* letters[] = {  
    ".-", "-...", "-.-.", "-..", ".,",  
    "....", "--.", "....", "...",      // A-I  
    ".---", "-.-.", "-...-", "--", "-.",  
    "----", ".---.", "--.-", ".-",     // J-R  
    "...", "-.", "-..-", "...-", ".--",  
    "-...-", "-.--", "--..",           // S-Z  
};
```

你可能还记得我们做过的计算，我们说“还有2KB的存储可以用来浪费”。但是如果内存有点紧张的话，那么将这些数据存储在用来存储程序的32KB的闪存中将会比2KB的RAM中要好得多。有一个这样做的方法——PROGMEM指令，它存在于一个库中，但是使用起来有一点尴尬。



8.2 PROGMEM指令

如果需要你的数据存储在闪存中，就必须像下面一样包含PROGMEM库：

```
#include <avr/pgmspace.h>
```

这一行命令的目的在于告诉编译器在编译这个Sketch的时候使用pgmspace库。在这个例子中，库是别人写好的一组函数，你可以直接把它们用在自己的Sketch中，而不需要理解这些函数是怎样工作的细节问题。

因为你使用了这个库，关键词PROGMEM和函数pgm_read_word就可以使用了。这两个都会在下面的Sketch中使用到。

这个库是被包含在Arduino软件中的，而且是Arduino库的一个官方支持。像这样的官方库有很多，网上还有一些非官方的库，由像你一样的人写出来供他人使用，也可以下载。先这样，非官方的库必须被安装到Arduino环境中，你在第11章中会学到更多关于这些库的东西，包括怎样写你自己的库。

使用PROGMEM的时候，必须确保你在使用特殊的PROGMEM可以支持的数据类型。不幸的是，其中并不包括char的字符串。实际中必须使用PROGMEM字符串类型来为每个字符串定义变量，然后把它们放到一个PROGMEM类型的数组中，像这样：

```
PROGMEM prog_char sA[] = ".-";  
PROGMEM prog_char sB[] = "-...";  
// and so on for all letters
```



```
PROGMEM const char* letters[] =  
{  
    sA, sB, sC, sD, sE, sF, sG, sH, sI, sJ, sK, sL, sM,  
    sN, sO, sP, sQ, sR, sS, sT, sU, sV, sW, sX, sY, sZ  
};
```

笔者并没有把sketch 8-01写在这里，而且它太长了，但你可能想要把它上传并且验证它是不是和放在RAM里的版本工作起来是一样的。

除了用特殊的方式来创建数据以外，你也必须用特殊的方式来读取它们。原本用来读取单词的莫尔斯码字符串的代码也需要被修改成这样：

```
strcpy_P(buffer, (char*)pgm_read_word(&(letters[ch -  
'a'])));
```

用PROGMEM方法重写的字符串中使用了buffer变量，这样它就可以像普通char数组一样使用了。这需要像下面一样被定义为一个全局变量：

```
char buffer[6];
```

这种方法只有在数据为常量的时候才能使用，即不会在Sketch运行的时候改变它。下一节中，你将学习使用EEPROM存储持续改变的数据。

8.3 EEPROM

Arduino UNO核心部位的ATMEGA328有1KB的EEPROM。EEPROM被设计用来数以年计地存储它其中的数据。尽管它名



字中含有“只读”，其实并不是“只读”的，你也可以往里头写东西。

用来读写EEPROM的Arduino指令，用起来和操作PROGMEN的一样尴尬。无论是读取还是写入，每次只能读写1个字节。

范例sketch 8-02中允许你从串口监视器中输入一个单独的字母码。Sketch记住这个字母，并且不断地串口监视器中重复写出这个字母。

```
// sketch 8-02
#include <EEPROM.h>

int addr = 0;
char ch;

void setup()
{
    Serial.begin(9600);
    ch = EEPROM.read(addr);
}

void loop()
{
    if (Serial.available() > 0)
    {
        ch = Serial.read();
        EEPROM.write(0, ch);
        Serial.println(ch);
    }
    Serial.println(ch);
    delay(1000);
}
```



打开串口监视器来试试这个Sketch，输入一个新的字符。把Arduino从电脑上拔掉，然后重新插上。当你重新打开串口监视器的时候，你会看到那个字母被记住了。

函数EEPROM.write有2个参数。第1个是地址，也就是EEPROM中的内存地址，数值必须在0~1023之间。第2个参数是需要在这个地址写入的数据，只能有1字节。1个字符占用8位，所以这没问题，但你不能直接存一个16位的整数。

在EEPROM中存储整数

想要在EEPROM的地址0和1中存储一个2字节的整数，你必须这样做：

```
int x = 1234;  
EEPROM.write(0, highByte(x));  
EEPROM.write(1, lowByte(x));
```

函数highByte和lowByte用来把一个整数拆成两个部分。

图8.1显示了这个整数是怎样存储在EEPROM中的。

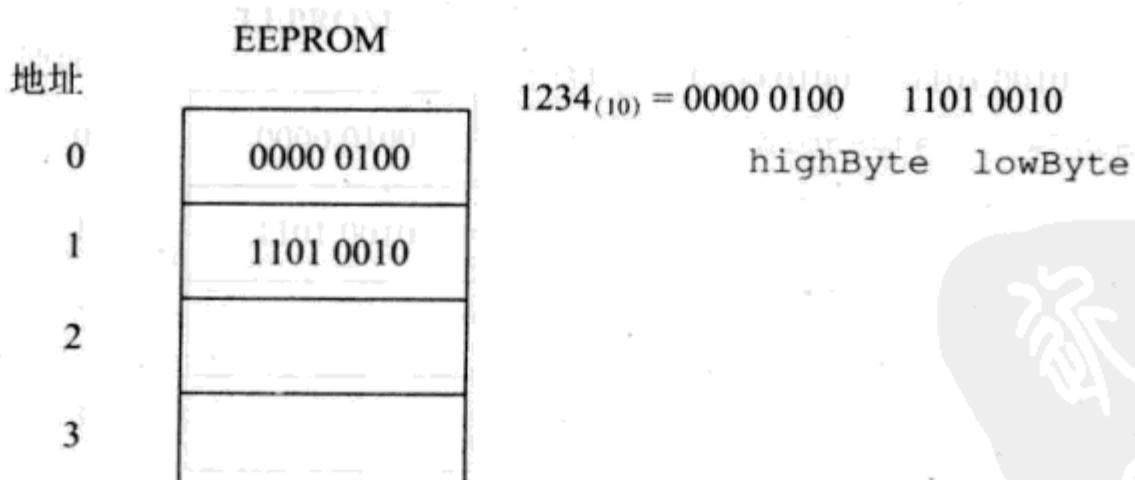


图8.1 在EEPROM中存储一个16位整数

想要把这个整数从EEPROM中读取回来，你需要把EEPROM中的2字节读取出来，并重新构建这个整数，如下：

```
byte high = EEPROM.read(0);
byte low = EEPROM.read(1);
int x = (high << 8) + low;
```

运算符`<<`是一个位移运算符，用来把8位的`highByte`挪到整数的顶部，并且和`lowByte`相加。

在EEPROM中存储浮点数（union）

在EEPROM中存储浮点数技巧性更强一些，你可以用C语言的`union`（并集）方法。这种数据结构可以看做是让几个不同变量访问同一片内存区域的一种办法，只要它们的字节数相同。

下面的`union`定义允许一个浮点数和一个整数指向同一个2字节的内存：

```
union data
{
    float f;
    int i;
} convert;
```

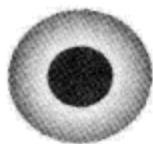
接下来，你可以如下在`union`中放置一个浮点数：

```
float f = 1.23;
convert.f = f;
```

然后，你可以像下面一样把一个整数拆成2字节，并存储到EEPROM中：

```
EEPROM.write(0, highByte(convert.i));
EEPROM.write(1, lowByte(convert.i));
```

想要把浮点数重新读取出来，则需要你做一个反转。首先需



要把2字节组装为一个单独的整数，然后将整数放入union并作为浮点数读取出来。

```
byte high = EEPROM.read(0);
byte low = EEPROM.read(1);
convert.i = (high << 8) + low;
float f = convert.f;
```



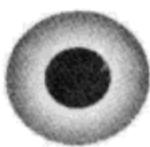
在EEPROM中存储字符串

在EEPROM中写入或者读取字符串就比较简单了。你只需要每次写入一个字符，如下所示：

```
char *test = "Hello";
int i = 0;
while (test[i] != '\0')
{
    EEPROM.write(i, test[i]);
    i++;
}
```

要想把字符串读取回字符数组中，你可以这样做：

```
char test[10];
int i = 0;
char ch;
ch = EEPROM.read(i);
while (ch != '\0' && i < 10)
{
    test[i] = ch;
    ch = EEPROM.read(i);
    i++;
}
```



清空EEPROM的内容

往EEPROM中写入的时候你要记住，即使你上传新的Sketch也不会清空EEPROM，所以可能会有在其中遗留一些以前的项目的数据。sketch 8-03可以将EEPROM中所有的数据重置为0：

```
// sketch 8-03
#include <EEPROM.h>

void setup()
{
    Serial.begin(9600);
    Serial.println("Clearing EEPROM")
    for (int i = 0; i < 1023; i++)
    {
        EEPROM.write(i, 0);
    }
    Serial.println("EEPROM Cleared");
}

void loop()
```

还需要注意的是，EEPROM的同一个位置只能被读写10万次，之后就变得不再可靠了，所以最好只在其中写一些真正需要写入的东西。而且EEPROM还很慢，需要3ms才能写入1字节。

8.4 压 缩

在EEPROM中存储数据或者使用PROGMEM的时候，有时可



能会发现你的存储空间不太够用了。这时候，就需要使用最有效率的方法来表示数据。

范围压缩

可能有某个值表面上看起来需要使用16位的整数或者是浮点数。例如，表达一个摄氏温度，你可能使用一个如20.25的浮点数值。当你把它存储到EEPROM的时候，如果能将其放入1字节的空间，事情就变得简单得多；而且如果你用浮点数的话，能存储以前存储量2倍的东西。

这样做的一种方法是在存储前改变数据。还记得你可以在1字节中存储一个0~255之间的整数吗？如果你只需要摄氏度的近似值，就可以简单地把浮点数转化成整数，抛弃小数点以后的部分。

```
int tempInt = (int)tempFloat;
```

变量tempFloat包含的是浮点值。`(int)`被称为类型转换（type cast），用来把变量从一个类型转化为另一个类型。在本例中，类型转换把值为20.25的浮点数转换为整数，这会使数值截取为20。

如果你只考虑0~60之间的摄氏温度值，那么你可以把所有的温度值在转换成1字节并存储之前乘以4。然后，当你从EEPROM中读回它的时候再除以4，以得到仍然精确到0.25的温度值。

下面的代码范例（sketch 8-04）将一个温度值存储到EEPROM，然后读取回来，并在串口监视器中显示。

```
//sketch 8-04

#include <EEPROM.h>

void setup()
{
    float tempFloat = 20.75;
    byte tempByte = (int)(tempFloat * 4);
    EEPROM.write(0, tempByte);

    byte tempByte2 = EEPROM.read(0);
    float temp2 = (float)(tempByte2) / 4;
    Serial.begin(9600);
    Serial.println("\n\n\n");
    Serial.println(temp2);
}

void loop() {}
```

还有一些其他的压缩数据的方法。例如，如果你需要操作变化很慢的度数——同样的，温度的变化也是一个很好的例子——那么你可以以全分辨率存储第1个温度值，然后每次只记录温度相对于前一个值的变化值，这种变化值一般来说会很小，而且占据更少的字节数。

8.5 总 结

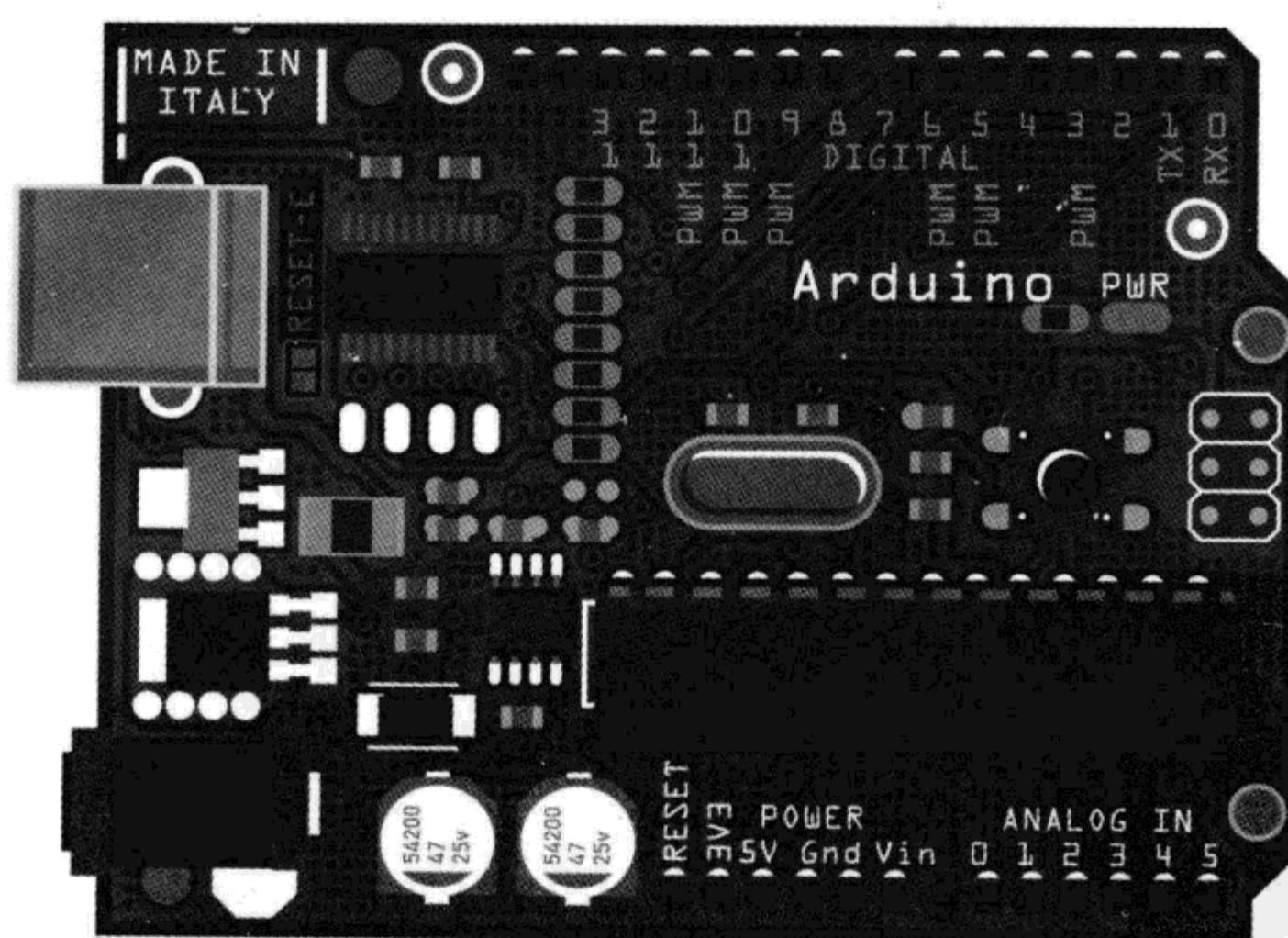
你现在已经知道了怎样在关闭电源以后仍然保留你的数据。
下一章，你将会接触到LCD显示器。



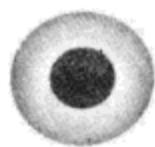
第9章

LCD显示器

LCD Displays



君
浩
PDG



本章中，你将学习怎样通过写软件来控制LCD显示器。图9.1为我们所使用的LCD显示器。

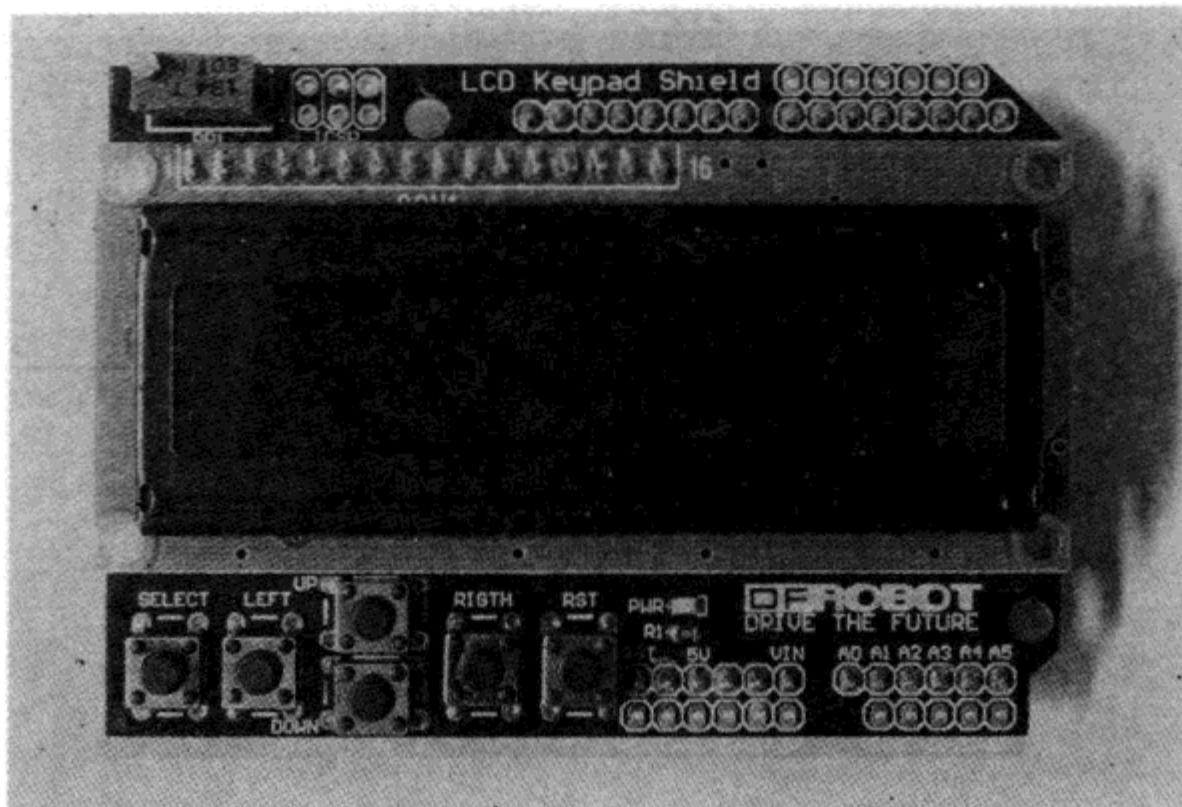


图9.1 一个字符式LCD扩展板

这本书是讲软件的，而不是硬件，但是在这一章中，我们不得不解释一点关于这些显示器中的电子元器件的工作原理，以便你可以理解怎样来驱动它们。

我们使用的LCD组件是一个预制Arduino扩展板，可以刚好插在Arduino板上。除了显示器以外，上面还有一些按钮。有很多种不同的扩展板，但是它们大部分用的都是同一种LCD控制芯片(HD44780)，所以找一个用这种控制芯片的扩展板就可以了。

我用的是DFROBOT LCD 键盘扩展板（图9.2）。这个组件由DFROBOT (www.robotshop.com) 制售，提供一个 16×2 的LCD显示器，并有6个按键。

这个LCD 扩展板使用Arduino上的7个针脚来控制LCD，1个模拟针脚用于按钮，所以这些针脚我们就不能用于其他目的了。

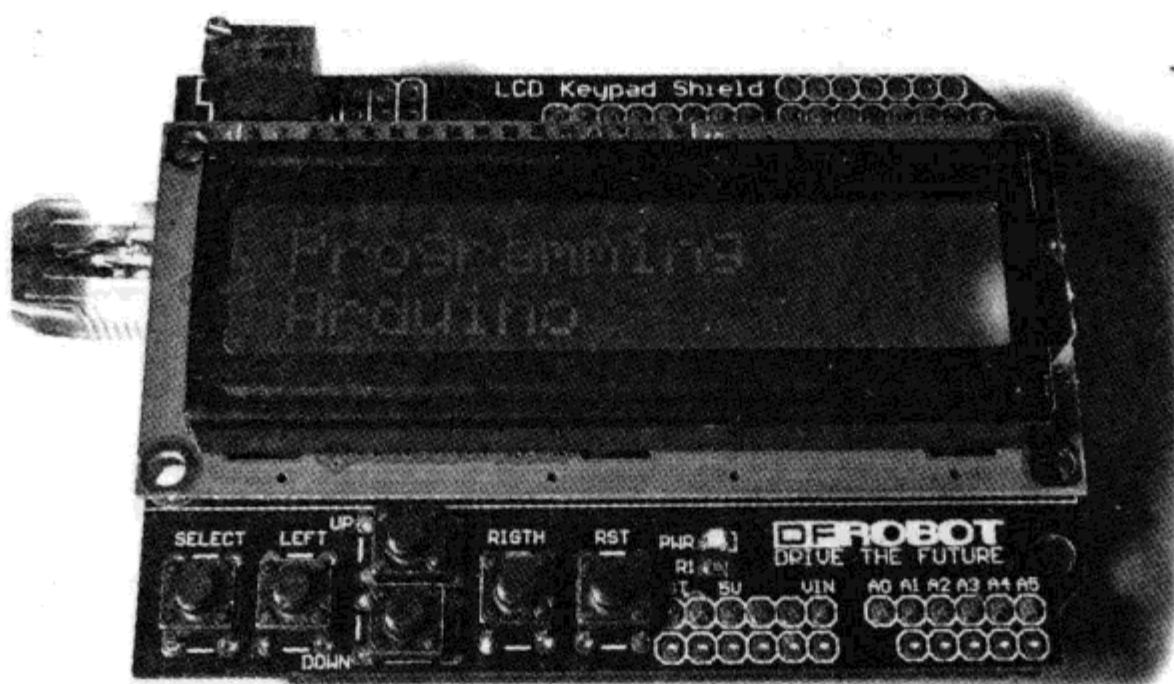


图9.2 将LCD扩展板插到Arduino板上

9.1 USB 信息板

为了简单说明显示器的一般用法，我们将制作一个USB信息显示板。它可以显示从串口监视器发送过去的消息。

Arduino IDE中内置了一个LCD库，这让使用LCD显示器变得更加简单。这个库给了一些实用的函数供你调用：

- `clear`: 清空显示屏;
- `setCursor`: 设置下一个将要显示的信息，在一行中的什么位置或者哪一行中出现;
- `print`: 在该位置写下一串字符。

sketch 9-01为范例：

```
// sketch 9-01 USB Message Board

#include <LiquidCrystal.h>

// lcd(RS, E, D4, D5, D6, D7)
```

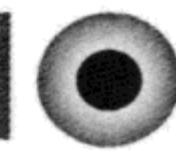


```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
int numRows = 2;
int numCols = 16;

void setup()
{
    Serial.begin(9600);
    lcd.begin(numRows, numCols);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Arduino");
    lcd.setCursor(0,1);
    lcd.print("Rules");
}

void loop()
{
    if (Serial.available() > 0)
    {
        char ch = Serial.read();
        if (ch == '#')
        {
            lcd.clear();
        }
        else if (ch == '/')
        {
            // new line
            lcd.setCursor(0, 1);
        }
        else
        {

```



```

    lcd.write(ch);
}
}
}

```

和所有的Arduino库一样，你必须在Sketch开头的地方包含这个库，才能让编译器使用它。

接下来的一行定义了那些针脚以何种目的被用于扩展板。如果使用的是别的扩展板，针脚的位置也不一定相同，所以你需要查阅扩展板的说明书。

在这个例子中，用来控制LCD的6个针脚为D4、D5、D6、D7、D8和D9，每个针脚的用途在表9.1中列出。

表9.1 LCD扩展板的针脚分配

LCD() 的参数	Arduino针脚号	用 途
RS	8	寄存器选择：将被设为1或0，这取决于Arduino是在发送字符还是一个指令，如使光标闪烁的指令
E	9	使能：通过切换状态来告诉LCD控制芯片下面4针脚的数据已经准备好被读取
数据4	4	这4个针脚用来传输数据，扩展板所用的LCD控制芯片可以使用4位或者8位数据。现在使用的扩展板使用4位数据，即使用的是4~7位，而不是0~7位
数据5	5	
数据6	6	
数据7	7	

setup函数比较简单。开始串口通信后，串口监视器发送指令，根据所使用的显示器的尺寸初始化LCD库，并且在显示器上分两行显示“Arduino Rules”：先将光标移到第1行的最左边显示“Arduino”，然后将光标移到第2行的最左边显示“Rules”。

大部分的动作发生在loop函数中，它检查是否有从串口监视器发过来的字符。Sketch每次处理1个字符。



除了Sketch可以直接显示的传统字符以外，还有两个特殊字符。如果字符为“#”，Sketch将清空显示器；如果是“/”，Sketch将把光标移动到第2行。否则，Sketch只会在光标所在位置使用函数write直接显示字符。函数write和print类似，但是它只能显示1个单独的字符而不是一串字符。

9.2 使用显示器

将sketch 9-01上传到板，然后插上扩展板。注意：必须在把Arduino连接到电脑之前并确保它在关闭的情况下插扩展板。

打开串口监视器，试试输入图9.3所示的文本。

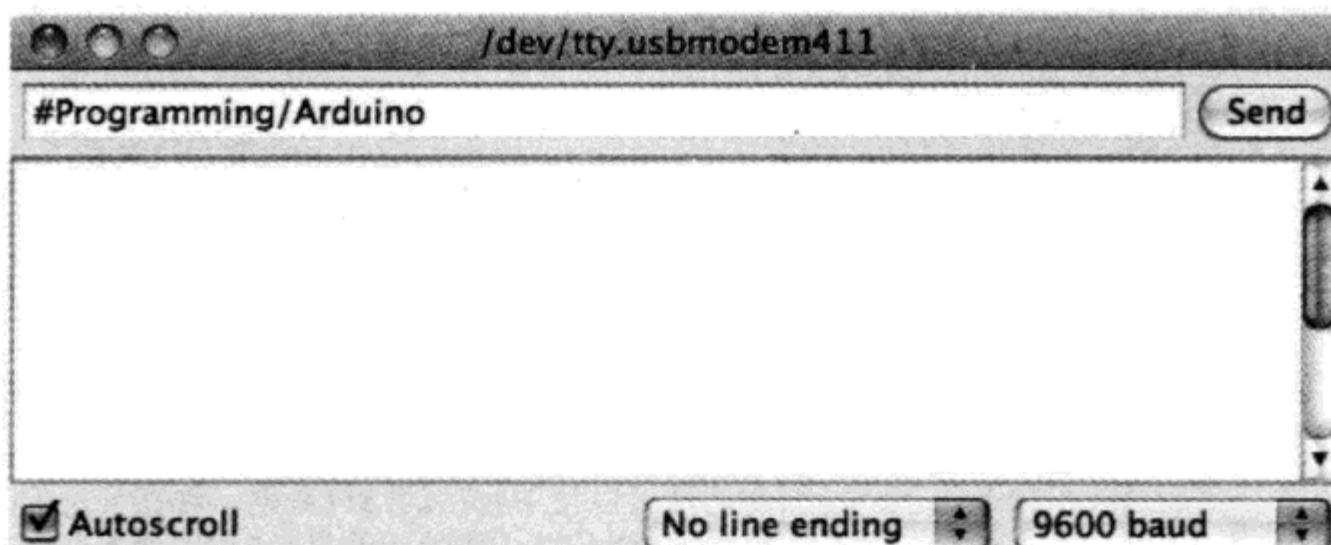


图9.3 向显示器发送指令

9.3 其他LCD库函数

除了在范例中使用过的函数以外，还有很多其他可用的函数：

- **home**: 和setCursor(0, 0)是一样的，都是将光标移至屏幕最左上角；
- **cursor**: 显示光标；



- noCursor: 不要显示光标;
- blink: 让光标闪烁;
- noBlink: 让光标停止闪烁;
- noDisplay: 直接关闭显示器, 不需要把它拔下来;
- display: 在noDisplay后重新开启显示器;
- scrollDisplayRight: 把显示器上所有的文本往右移动1格;
- autoscroll: 激活一个状态, 被激活时, 每当在光标处加入一个新的字符, 现存所有的文本将向左或者向右移动一格, 方向由函数leftToRight或者RightToLeft决定;
- noAutoscroll: 把autoscroll激活的状态关闭。

9.4 总 结

你会发现, 给扩展板编程并不难, 特别是还有一个可以做大部分工作的库。

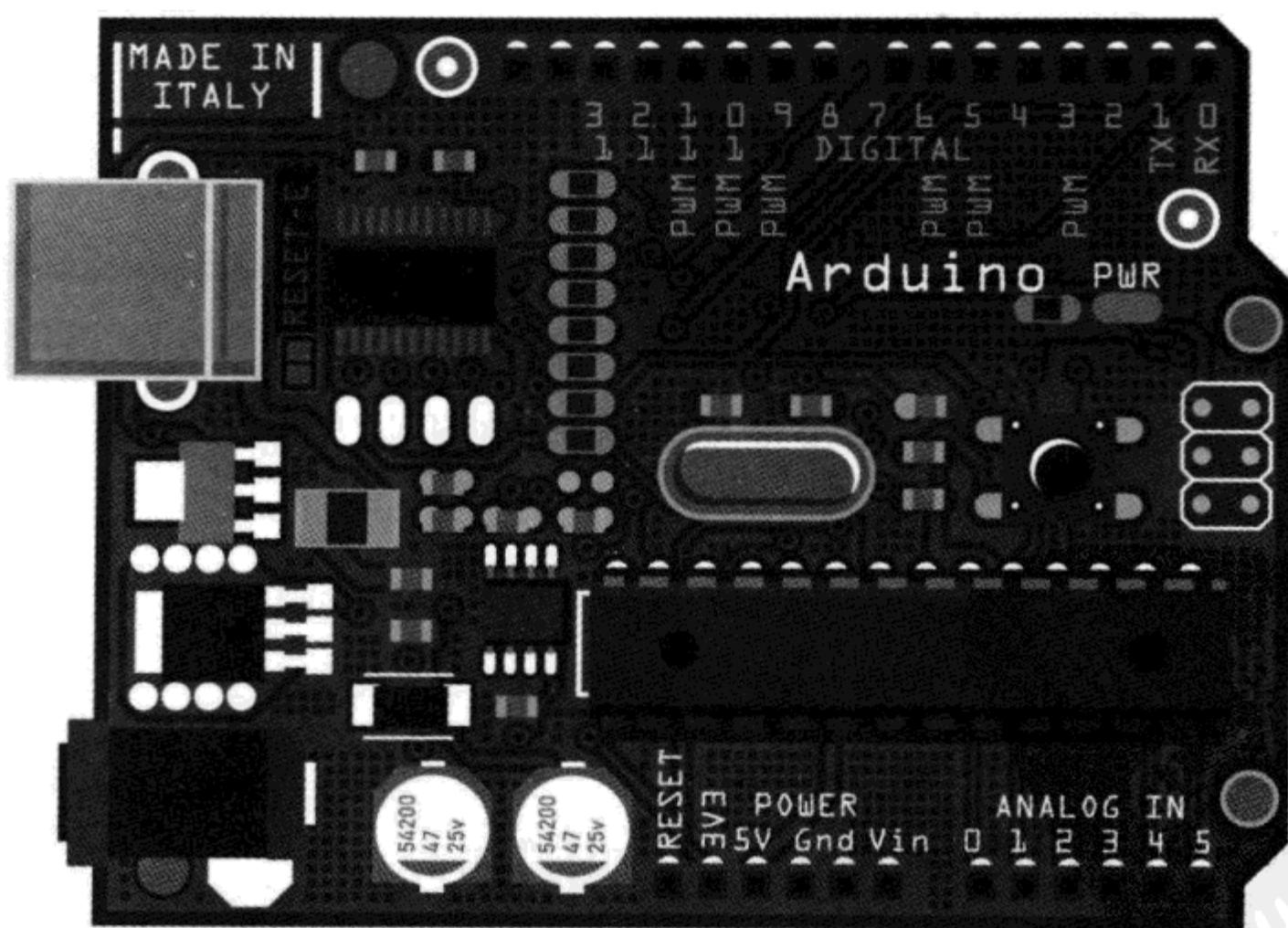
下一章, 你将会使用网络扩展板来把你的Arduino连接到网络。

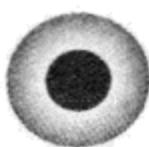


第10章

Arduino 网络编程

Arduino Ethernet Programming





本章将使用网络扩展板，让你的Arduino可以通过家里的网络工作，如图10.1所示。

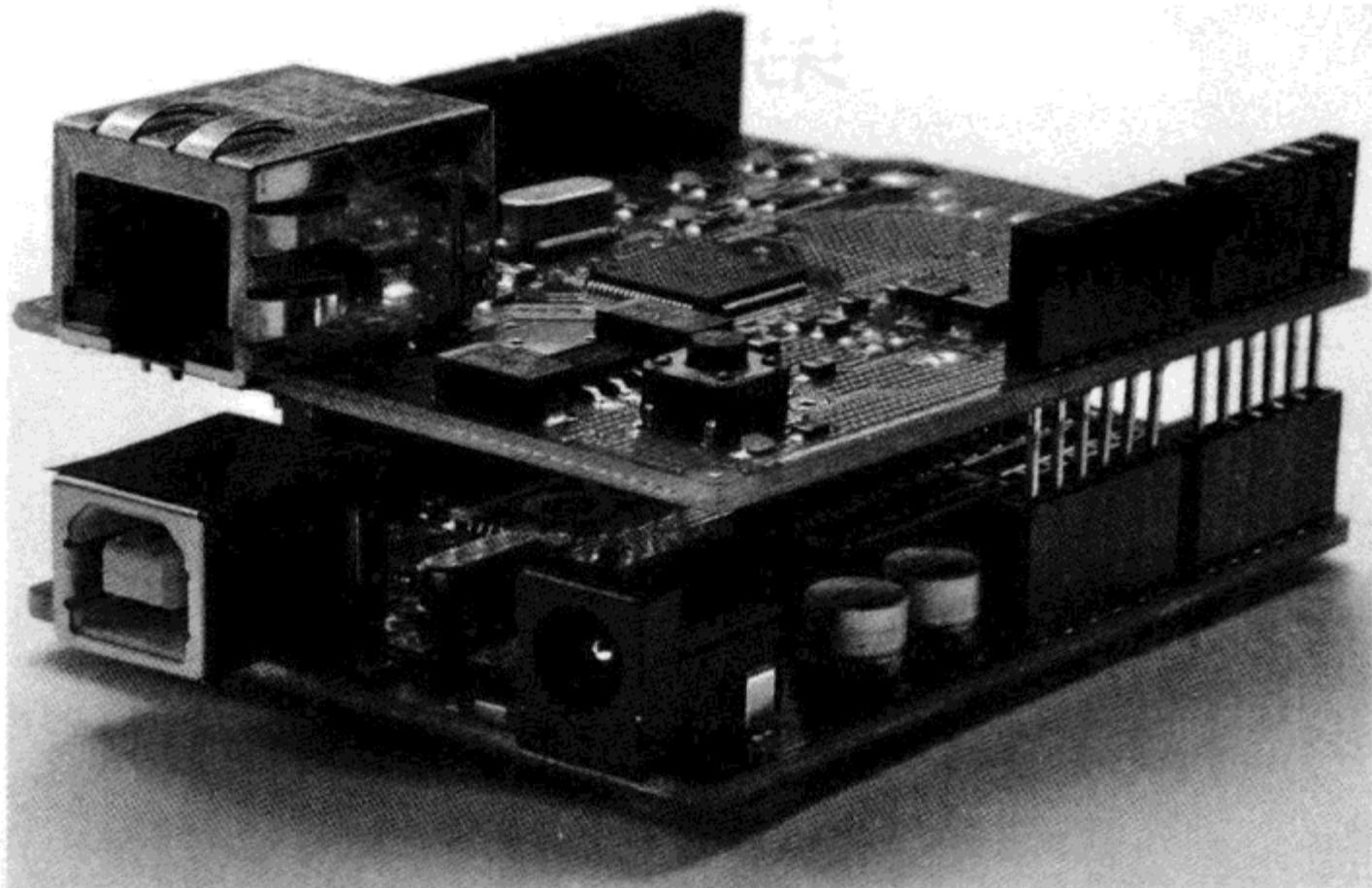


图10.1 Arduino 和网络扩展板

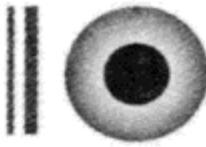
10.1 网络扩展板

购买网络扩展板的时候需要注意，要买基于Wiznet芯片组的“官方”扩展板，而不是基于ENC28J60以太网控制芯片的难用的“非官方”扩展板。

网络扩展板挺费电的，所以你需要一个9V或12V、电流大于1A的电源，这个电源插在Arduino电源接口上使用。

10.2 和Web服务器通信

在了解Arduino是怎样与它所使用的浏览器和Web服务器通信



之前，你需要了解一些超文本传输协议（HTTP）和一些超文本标记语言（HTML）。

HTTP

HTTP（超文本传输协议）是用于Web浏览器和Web服务器之间的通讯的。

当你使用浏览器访问一个页面的时候，浏览器向页面所在服务器发送一个请求，告诉服务器它的需求。浏览器所请求的一般可能只是页面中用HTML所写的内容。Web服务器一直在等候这些请求，一旦收到就会进行处理。在本书的情况中，对请求的处理只是发回你在Arduino Sketch中所指定的HTML语言。

HTML

HTML（超文本标记语言）是给一般文本加上格式，而让它们在浏览器中显示的时候看起来更加好看用的。例如，下面的HTML代码在浏览器页面中显示的时候会如图10.2所示：

```
<html>
<body>
<h1>Programming Arduino</h1>
<p>A book about programming Arduino</p>
</body>
</html>
```

HTML含有标签。标签有开头和结尾，而且其中一般还包含其他标签。一个标签的开头是<标签名>，如<html>。标签的结尾很类似，除了<后面会有一个/。在前面的例子中，最外层的

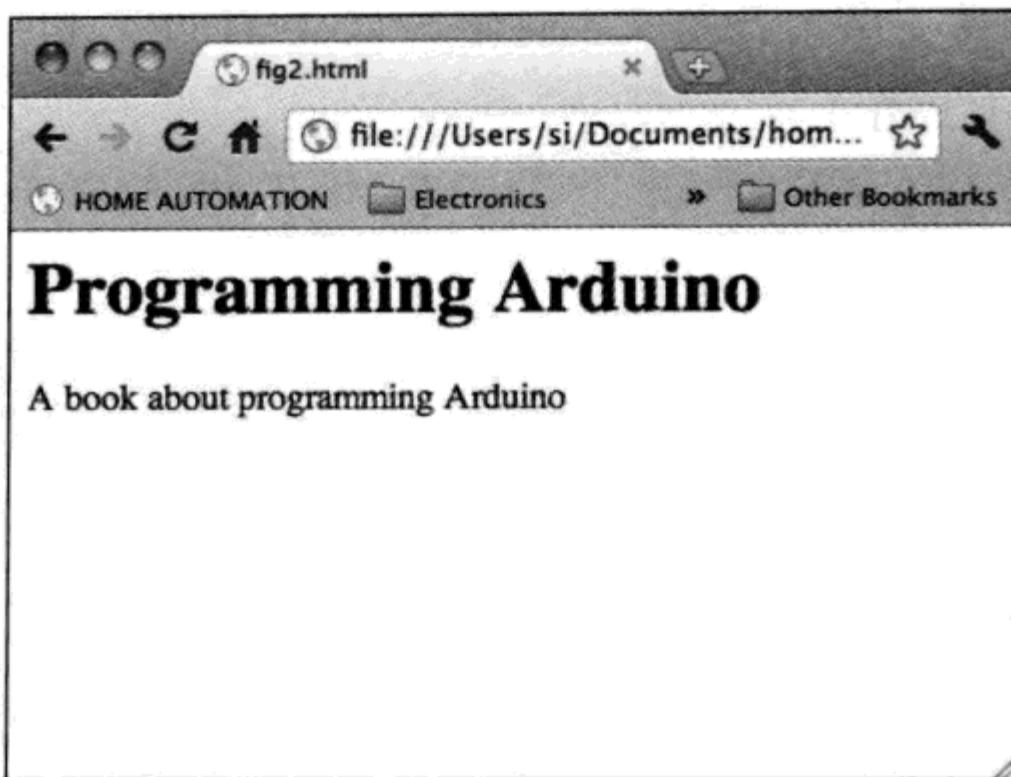


图10.2 一个HTML例子

标签是<html>，它包含一个叫做<body>的标签。所有的Web页面都要以这样的标签作为开始，并且在文件结束处对应着这些标签的结尾。注意：必须按照正确的顺序来放置结尾标签，如标签body必须紧挨着放在标签html的前面。

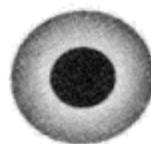
现在我们来看看较为有趣的中间部分，标签h1和标签p。这些是范例中真正显示出来的部分。

标签h1的意思是一级标题，它的效果是将其中的内容用大的粗体字显示。标签p是段落标签，其中的所有内容将会显示为一个段落。

这些其实只是非常浅的HTML语言。有很多的书和网络资源可以用来学习HTML语言。

10.3 将Arduino用作Web服务器

第一个简单的Sketch范例是将Arduino和网络扩展板制作成了



一个小型的Web服务器。当然不能像Google那么厉害，但是它可以让你发送一个Web请求到Arduino，并在电脑上的浏览器中查看结果。

在上传sketch 10-1之前，还需做一些修改。如果你看一下Sketch的开头，就会看到下面这一行：

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 30 };
```

第1行为mac地址，在你所有连接到网络的设备中必须是唯一的。第2行为IP地址，而你的网络中大部分的设备都有一个由动态主机配置协议（DHCP）自动分配给它们的IP地址，但这对网络扩展板并不适用，你必须手动为它配置一个IP地址。这个IP地址可不是随便的4个数字，它必须可以用作内网IP，而且必须在路由器的地址池中。典型的IP地址一般为10.0.1.x或者192.168.1.x，这里x必须是一个0~255之间的数字。这些IP地址中有一些可能已经被你网络中的其他设备占用了，如果要找到一个可用的并且未被占用的IP地址，你可以登陆到你的路由器，找到DHCP选项，就会看到一个使用中的设备及IP地址列表，类似于图10.3。在本例中，192.168.1.30看起来是个不错的选择，而且它也确实可以使用。

将Arduino连接到电脑，并上传Sketch。现在，你可以断开USB连接，插上电源和网线。

用电脑上的浏览器连接刚才你为网络扩展板指定的IP，应该会看到图10.4所示效果。



User Name	IP Address	MAC Address
Android_358225044128768	192.168.1.2	DE-B3-77-3B-6C-38
Simons-Mac	192.168.1.3	FB-1E-DF-D9-6F-A4
	192.168.1.4	3C-4A-92-AB-2E-93
android_f2fb9d9761e892ef	192.168.1.5	00-23-76-B6-A4-63
clare-Aspire-one	192.168.1.6	00-24-2C-3D-19-FF
	192.168.1.7	D8-30-62-84-E1-2F
Clares-iPod	192.168.1.8	64-B9-E8-FC-FD-3B
	192.168.1.9	28-EF-01-64-7E-94
BLACKBERRY-7BB6	192.168.1.10	3C-74-37-84-55-22
nny	192.168.1.11	00-17-F2-FC-EF-01
android_ff52e0ea4d3fa336	192.168.1.12	00-23-76-98-4C-FB
android_f023908324ec978b	192.168.1.13	F8-DB-7F-75-E9-20
Michaela-PC	192.168.1.14	00-17-C4-F2-A5-98
Steve'sPC	192.168.1.15	00-1F-3C-28-EC-20
matthew-AMILO-Li-1718	192.168.1.16	00-C0-A8-E6-B1-1D
The-Titanic	192.168.1.17	E8-06-88-58-57-6A
Nicole's-iPhone	192.168.1.18	00-21-E9-24-33-E7

图10.3 找到未被使用的IP



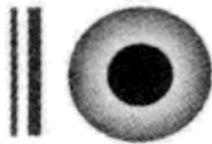
图10.4 一个简单的Arduino服务器范例

sketch 10-1的代码如下：

```
// sketch 10-01 Simple Server Example

#include <SPI.h>
#include <Ethernet.h>

// MAC address just has to be unique. This should
work:
```



```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// The IP address will be dependent on your local
network:
byte ip[] = { 192, 168, 1, 30 };

Server server(80);

void setup()
{
    Ethernet.begin(mac, ip);
    server.begin();
    Serial.begin(9600);
}

void loop()
{
    // listen for incoming clients
    Client client = server.available();
    if (client)
    {
        while (client.connected())
        {
            // send a standard http response header
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println();
            // send the body
            client.println("<html><body>");
            client.println("<h1>Arduino Server</h1>");
            client.print("<p>A0=");
            client.print(analogRead(0));
        }
    }
}
```



```
    client.println("</p>");  
    client.print("<p>millis=");  
    client.print(millis());  
    client.println("</p>");  
    client.println("</body></html>");  
    client.stop();  
}  
delay(1);  
}  
}
```

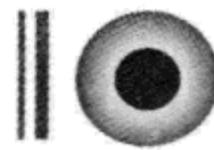
如同我们在第9章中讨论过的LCD库一样，也有一个专门为网络扩展板提供的标准Arduino库。

setup函数，试用刚才你设置过的mac和IP地址来初始化网络扩展库。

loop函数，负责处理所有从浏览器发往Web服务器的请求。如果一个请求正在等待回应，那么对函数server.available的调用将会返回一个客户端。客户端是一个对象，你将在第11章中学到更多这方面的知识。但是现在，你需要知道的只是如果客户端存在的话（通过第一个if语句来检测），那么就可以通过决定是否调用函数client.connected来决定它是否连接到Web服务器。

接下来的3行代码可以显示出一个返回的标题，告诉浏览器要显示什么类型的内容。在这个例子中，浏览器需要显示HTML的内容。

当标题被写下之后，要写的就是要返回到浏览器中剩下的HTML了。这些必须包含常用的<html>和<body>标签，还需要包含1个<h1>标题标签和2个<p>标签，它们用来显示A0的模拟输入值和函数millis的返回值——这个值是从Arduino上一次被



重置到现在的时间的毫秒数。

最后，函数`client.stop`告诉浏览器，消息已经完整了，这时浏览器显示出整个页面。

10.4 通过网络来设置Arduino的针脚

使用网络扩展板的另一个例子，是让你可以通过网络来开关Arduino的D3 和D7针脚。

和简单的服务器范例不太一样，你需要找到一个将针脚设置传递给Arduino的方法。

这样做的方法被称为数据投递（posting data），它是HTTP协议的一部分。想使用这种方法，就必须在HTML中建立一个投递途径，让Arduino返回给浏览器的HTML中可以提供一个表格。这个表格（图10.5）为每个针脚都提供了一个开关按钮和一个“Update”按钮，用来把针脚设置发送给Arduino。

Output Pins

Pin 3	On	Off
Pin 4	On	Off
Pin 5	Off	On
Pin 6	Off	On
Pin 7	Off	On

Update

图10.5 信息发送表格



按下“Update”之后，会往Arduino发送第2次请求。它和第1次请求很类似，这次请求除了会包含请求参数，参数中还包含着针脚的值。

请求参数在概念上和函数的参数类似。函数的参数允许你将信息传达给函数，如需要闪烁多少次，而请求参数允许你将数据传递给用来处理Web请求的Arduino。当Arduino收到Web请求之后，就会从参数中提取针脚设置，然后改变针脚的值。

第2个范例的代码如下：

```
// sketch 10-02 Internet Pins

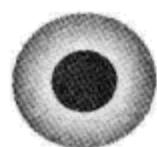
#include <SPI.h>
#include <Ethernet.h>

// MAC address just has to be unique. This should
// work:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// The IP address will be dependent on your local
// network:
byte ip[] = { 192, 168, 1, 30 };
Server server(80);

int numPins = 5;
int pins[] = {3, 4, 5, 6, 7};
int pinState[] = {0, 0, 0, 0, 0};
char line1[100];

void setup()
{
    for (int i = 0; i < numPins; i++)
```

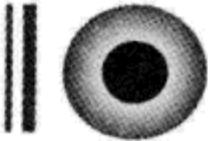
```
{  
    pinMode(pins[i], OUTPUT);  
}  
Serial.begin(9600);  
Ethernet.begin(mac, ip);  
server.begin();  
  
}  
  
void loop()  
{  
    Client client = server.available();  
    if (client)  
    {  
        while (client.connected())  
        {  
            readHeader(client);  
            if (! pageNameIs("/"))  
            {  
                client.stop();  
                return;  
            }  
            client.println("HTTP/1.1 200 OK");  
            client.println("Content-Type: text/html");  
            client.println();  
  
            // send the body  
            client.println("<html><body>");  
            client.println("<h1>Output Pins</h1>");  
            client.println("<form method='GET'>");  
            setValuesFromParams();  
            setPinStates();  
        }  
    }  
}
```



```
for (int i = 0; i < numPins; i++)
{
    writeHTMLforPin(client, i);
}
client.println("<input type='submit' value='Update' />");
client.println("</form>");
client.println("</body></html>");
client.stop();
}

}

void writeHTMLforPin(Client client, int i)
{
    client.print("<p>Pin ");
    client.print(pins[i]);
    client.print("<select name='");
    client.print(i);
    client.println("'");
    client.print("<option value='0'");
    if (pinState[i] == 0)
    {
        client.print(" selected");
    }
    client.println(">Off</option>");
    client.print("<option value='1'");
    if (pinState[i] == 1)
    {
        client.print(" selected");
    }
}
```



```
client.println(">On</option>");  
client.println("</select></p>");  
}  
  
void setPinStates()  
{  
    for (int i = 0; i < numPins; i++)  
    {  
        digitalWrite(pins[i], pinState[i]);  
    }  
}  
  
void setValuesFromParams()  
{  
    for (int i = 0; i < numPins; i++)  
    {  
        pinState[i] = valueOfParam(i + '0');  
    }  
}  
  
void readHeader(Client client)  
{  
    // read first line of header  
    char ch;  
    int i = 0;  
    while (ch != '\n')  
    {  
        if (client.available())  
        {  
            ch = client.read();  
            line1[i] = ch;
```





```
i++;
}
line1[i] = '\0';
Serial.println(line1);
}

boolean pageNameIs(char* name)
{
    // page name starts at char pos 4
    // ends with space
    int i = 4;
    char ch = line1[i];
    while (ch != ' ' && ch != '\n' && ch != '?')
    {
        if (name[i-4] != line1[i])
        {
            return false;
        }
        i++;
        ch = line1[i];
    }
    return true;
}

int valueOfParam(char param)
{
    for (int i = 0; i < strlen(line1); i++)
    {
        if (line1[i] == param && line1[i+1] == '=')
        {
            return (line1[i+2] - '0');
        }
    }
}
```

```
    }  
}  
return 0;  
}
```

Sketch中使用了2个数组来控制针脚。pins用来指定需要用到哪些针脚，pinState数组保存每个针脚的状态——0或者1。

若要从浏览器获得关于哪些针脚被打开、哪些被关闭的信息，必须读取从浏览器发过来的标题。实际上，你只需要在标题的第1行中包含这些信息就可以了。可以使用字符数组line1来包含标题的第1行。

当用户点击“update”按钮并从浏览器提交表格的时候，当前页面的链接看起来应该和下面的这一个差不多：

```
http://192.168.1.30/?0=1&1=1&2=0&3=0&4=0
```

请求参数被写在?后面，并且用&分割开来。看一下第一个参数(0=1)，这表示数组中的第1个针脚(pins[0])的值应该为1。如果你注意了标题的第1行，就会看到一些类似的请求参数：

```
GET /?0=1&1=1&2=0&3=0&4=0 HTTP/1.1
```

在参数前面有一个GET/，它指定了请求是由浏览器发送的。在本例中，/代表根页面。

在Sketch的loop部分，通过调用函数readHeader来读取标题的第1行。接下来，使用函数pageNameIs来检查根页面/的页面请求。

Sketch接下来生成标题和需要显示的HTML表格开头部分。在写关于每个针脚的HTML语言之前，Sketch调用函数



`setValuesFromPaRAMs`来读取每一个请求参数，并且在`pinStates`数组中放置适当的值。这些值在接下来为每一个针脚调用函数`writeHTMLforPin`之前，用来设置针脚的输出值。函数`writeHTMLforPin`还用来为每一个针脚生成选项列表。这个列表必须一部分接一部分地建立。`if`语句用来保证选择适当的选项。

函数`readHeader`、`pageNameIs`和`valueOfPaRAM`是非常实用的通用函数，你可以在自己的Sketch中使用它们。

你可以像在第6章里一样，使用万用表来测试针脚是不是真的被打开或者关闭了。如果你更有探险精神的话，可以在针脚上附加上LED或者继电器来控制更多的东西。

10.5 总 结

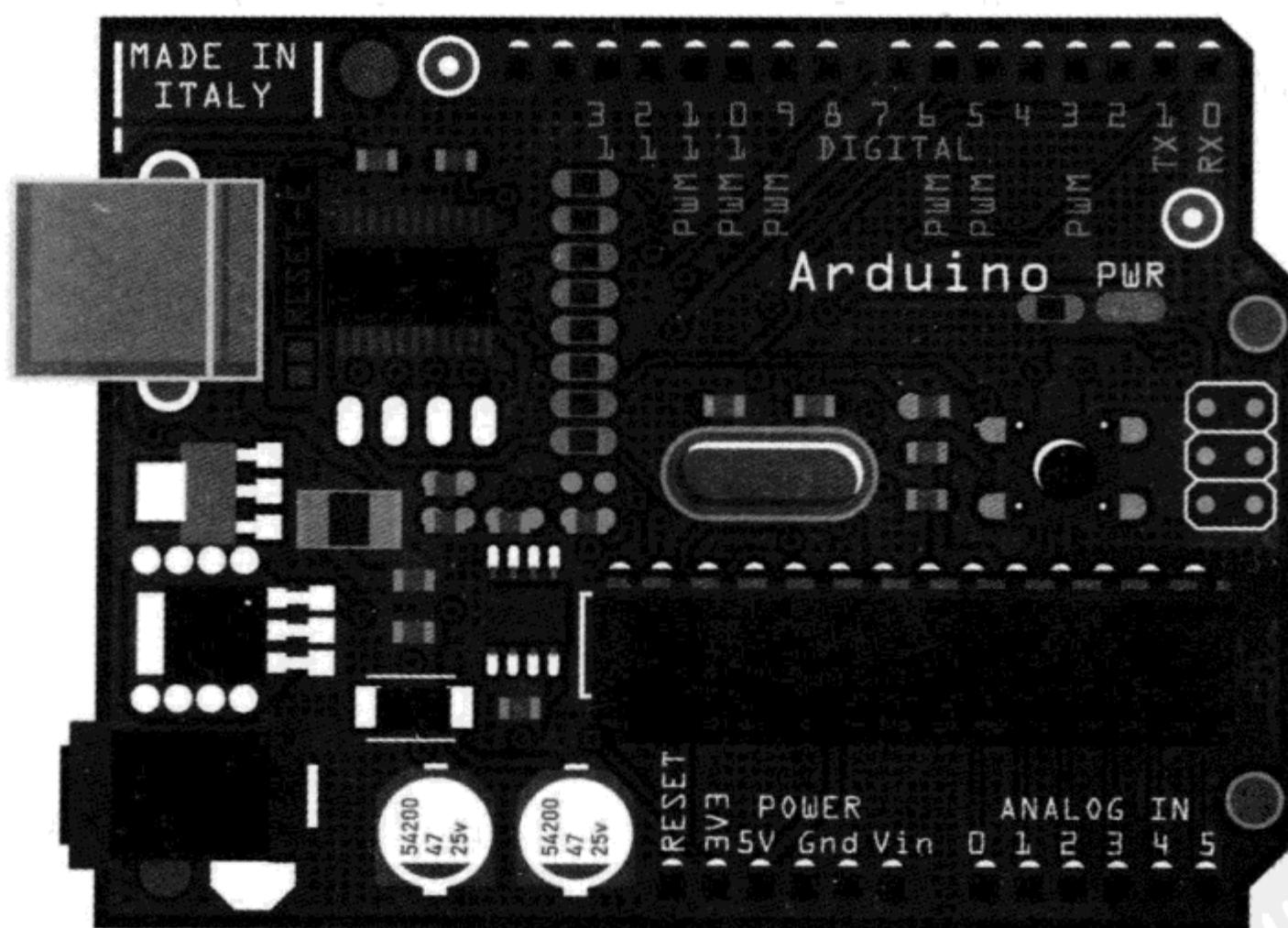
在前2章中我们使用了扩展板和相关的库，现在是时间看看库是怎样被写出来的，并且学习写你自己的库了。



第11章

C++和库

C++ and Libraries





Arduino可以看做是简单的单片机，大部分时候Arduino Sketch是很小的，所以仅仅使用C语言基本够用。但是，Arduino的编程语言是C++而不是C。C++是C语言的扩展，它加入了称为“面向对象”的东西。

11.1 面向对象

本书只是很短的一本书，对C++的深入介绍就超出它的范围了。但是我们可以对C++和面向对象有一个基础的了解，主要的目的是为了提高你的程序的封装度。封装就是将相关的事情放到一起，这让C++变得非常适合写类似于前面你在网络扩展和LCD扩展的Sketch中用过的库。

有很多很好的关于C++和面向对象编程的书，你可以找来看一看。

类和方法

面向对象语言使用一个被称为类的概念来帮助封装。一般来说，类相当于程序的一部分，其中包括变量——称为成员变量——和方法。方法类似于函数，但它是应用于类的。这些函数可以是公有的，也就是说方法和函数可以被其他类使用；或者是私有的，即该方法只可以被所在类中的其他方法调用。

鉴于Arduino Sketch是包含在一个单独的文件中的，当你用C++编程的时候，需要使用不止一个文件。实际上，一个类通常需要有2个文件：一个头文件，扩展名为.h；一个实现文件，扩



展名为 .cpp。

11.2 内建库示例

LCD库在前面曾经有2章使用过，现在让我们更深入地看看它到底在细节方面做了些什么。

参见sketch 9-01（并将其在Arduino IDE中打开），你会发现include指令包含了文件LiquidCrystal.h：

```
#include <LiquidCrystal.h>
```

这个文件是类LiquidCrystal的头文件，告诉Arduino Sketch使用这个库需要知道些什么。如果你在Arduino安装文件夹中检索，就可以找到“/LiquidCrystal”这个文件库，这个文件需要用文本编辑器才能打开。如果你使用的是Mac电脑，那么在Arduino程序上右键选择显示内容，然后定位到“Contents/Resources/Java/libraries/LiquidCrystal”。

LiquidCrystal.h文件包含了很多代码，因为它是较大的一个类库。类本身的代码，也就是在LCD中显示消息的具体工作的细节，是在包含在文件LiquidCrystal.cpp里的。

下一节将会创建一个简单的库的范例，使库的概念更加具体化。

11.3 写一个库

创建一个Arduino库，可能看起来是一个经验丰富的Arduino老手才可以做的事情，但是实际上制作一个库是比较简单的。例



如，你可以把第4章中的使LED闪烁指定次数的flash函数转化成一个库。

要创建一个C++文件，就要用到文本编辑器，如Windows中的记事本或者是Mac电脑里的TextMate。

头文件

从创建一个文件夹来包含所有的库所需的文件开始。你需要在你的“Arduino documents”文件夹中的“libraries”文件夹中来创建这个文件夹。在Windows下，你的“libraries”文件夹在“My Documents\Arduino”下。在Mac系统下，你可以在“Documents/Arduino/”下找到他。Linux系统下，它在根目录下的“Sketchbook”文件夹中。如果你的Arduino文件夹中没有“libraries”文件夹，那就新建一个。

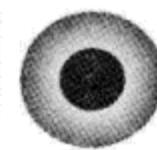
“libraries”文件夹是放置所有你自己写的或者安装的非官方库的地方。

将文件夹命名为“Flasher”。打开文本编辑器，输入以下内容：

```
// LED Flashing library

#include "WProgram.h"

class Flasher
{
public:
    Flasher(int pin, int duration);
    void flash(int times);
```



```
private:  
    int _pin;  
    int _d;  
};
```

在“Flasher”文件夹下保存这个文件为“Flasher.h”，这就是类库的头文件了。这个文件指定了类的不同部分。如你所见，它被分为私有和公有两部分。

公有部分看起来像是两个函数的开头。这些被叫做方法，它们和函数只有一点不同，就是它们是和类相关的，只能被作为类的一部分使用。和函数不同，它们不可以单独使用。

第1个方法是Flasher，使用大写字母打头，你在命名函数的时候不会这样做。它的名字和类的名字相同。这个方法被称为构造函数，你可以通过它来创建一个新的Flasher对象在Sketch中使用。

例如，你可以将下面一行放在Sketch中：

```
Flasher slowFlasher(13, 500);
```

这一行将会创建一个新的被称为slowFlasher的新Flasher，它可以以500ms为周期在D13针脚上闪烁。

类中的第2个方法是flash。这个方法只有1个参数，就是闪烁的次数。因为它和类是相关的，所以调用它的时候必须提到你刚才创建的对象，如下：

```
slowFlasher.flash(10);
```

这行代码会使LED按照你在对象Flasher的构架函数中指定的周期闪烁10次。



类的私有部分包含了2个变量的定义：一个是针脚，一个是周期（以d为缩写）。每当你创建一个Flasher类的对象的时候，必须包含这两个变量。这使得在新的Flasher对象被创建的时候可以记住针脚和周期。

这些变量被称为成员变量，因为它们是类中的成员。它们的名字很特别，都是以下划线作为开头的，但这只是习惯用法而已，并不是程序所必需的。另外一个常用的命名方法是，用小写字母“m”（member）作为变量名的首字母。

实现文件

头文件仅仅定义了类看起来是什么样的，你现在需要一个独立的文件来做真正的工作。这个文件被称为实现文件，以.cpp作为扩展名。

那么，在“Flasher”文件夹中建立一个新文件，包含以下内容并且命名为“Flasher.cpp”：

```
#include "WProgram.h"
#include "Flasher.h"

Flasher::Flasher(int pin, int duration)
{
    pinMode(pin, OUTPUT);
    _pin = pin;
    _d = duration / 2;
}

void Flasher::flash(int times)
```



```
{  
    for (int i = 0; i < times; i++)  
    {  
        digitalWrite(_pin, HIGH);  
        delay(_d);  
        digitalWrite(_pin, LOW);  
        delay(_d);  
    }  
}
```

这个文件中有一些陌生的语法。两个方法的名称都用 `Flasher::` 作为前缀，这表明方法都属于 `Flasher` 类。

构造函数方法 (`Flasher`) 将它的每个参数分配给了适当的私有成员变量。周期参数在被分配给成员变量 `_d` 之前被除以2，这是因为 `delay` 被调用了2次，而且将周期看做是闪烁时长和两次闪烁之间的间隔时长之和会更有逻辑性一些。

函数 `flash` 实际上是负责闪烁LED的。它循环适当的次数，以合适的周期不断地开关LED。

完成你的库

你现在已经看过了完成库所需的所有要点，已经可以完成这个库，而且可以让它工作得很好。但是，完成这个库，你还有两件事情需要做。一是定义在库中使用的关键词，使Arduino IDE可以在用户编辑代码的时候显示出适当的颜色；二是包含一些关于怎样使用这个库的例子。

- 关键词

如果要定义关键词，则必须在“Flasher”文件夹中创建一个



名为“keywords.txt”的文件，这个文件只有下面的两行：

```
Flasher KEYWORD1  
flash KEYWORD2
```

这实际上是一个在文本文件中的两列的表格。左边一列为关键词，右边一列是关键词类型。类名称应该为KEYWORD1，方法应该是KEYWORD2。两列中间用了多少空格并不要紧，但是每一个关键词都必须另起一行。

● 范例

想成为一个优秀的Arduino玩家，你还需要在库所在文件夹里包含一个范例文件夹，作为库的一部分。在本例中，这个库太简单，以至于用一个例子就足够了。

范例必须放在“Flasher”文件夹下的“examples”文件夹下。这个范例实际上就是一个Arduino Sketch，所以你可以用Arduino IDE来创建这个范例。但是首先，你必须重启Arduino IDE来让新的库生效。

重启Arduino IDE以后，在IDE的“文件”菜单中选择“新建”来打开一个新的Sketch窗口，然后在Sketch菜单中选择“Import Library”选项，如图11.1所示。

子菜单中横线上方的是官方库，下方的则是第三方库。如果一切顺利的话，你可以在列表中看到“Flasher”。

如果列表中没有“Flasher”，八成是因为“Flasher”文件夹不在你的“Sketches”文件夹下的“libraries”文件夹中，检查一下是否如此。

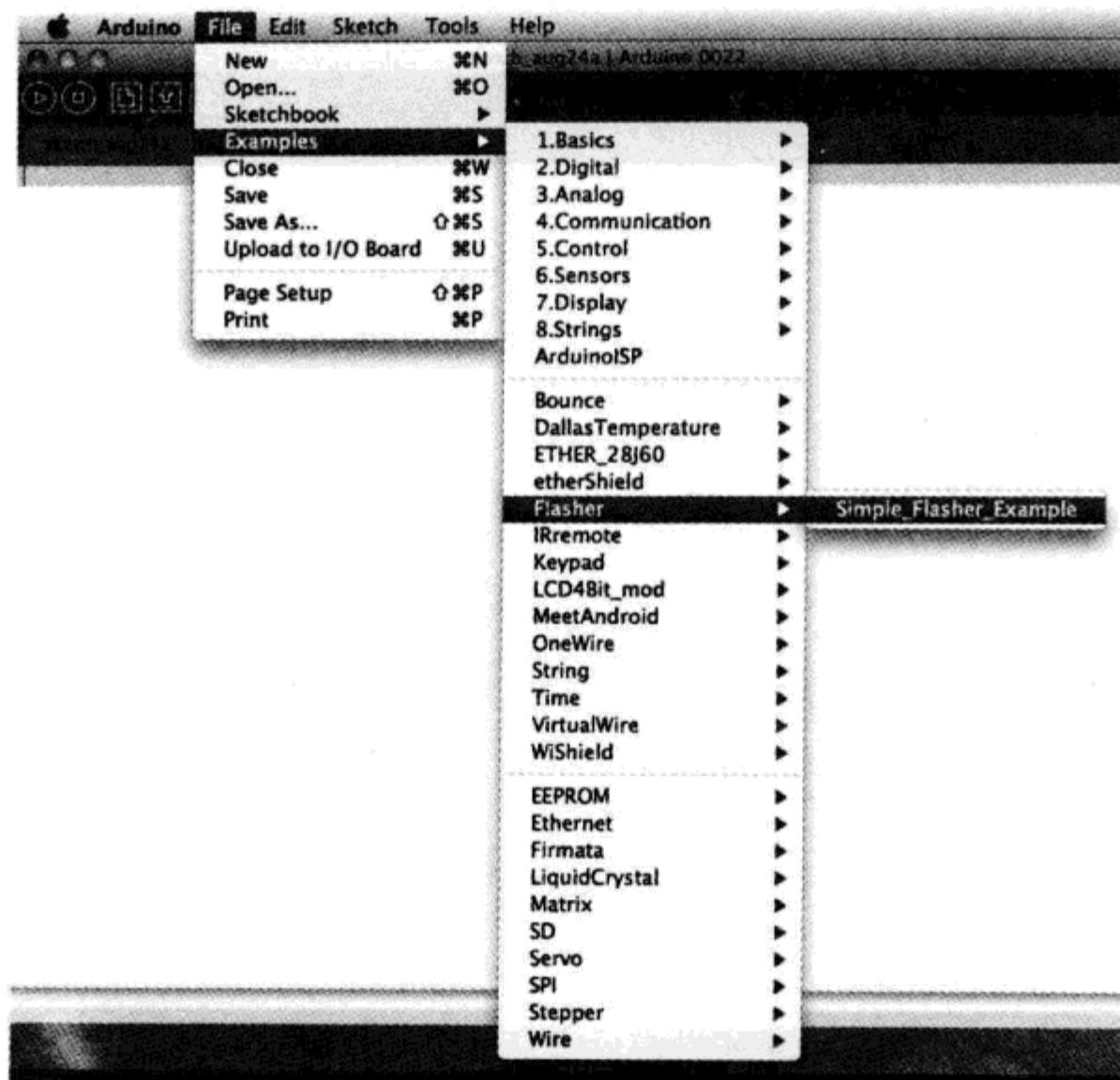


图11.1 导入Flasher库

```
#include <Flasher.h>

int ledPin = 13;
int slowDuration = 300;
int fastDuration = 100;

Flasher slowFlasher(ledPin, slowDuration);
Flasher fastFlasher(ledPin, fastDuration);

void setup() {}

void loop()
{
```



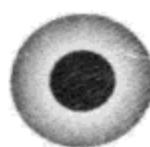
```
slowFlasher.flash(5);  
delay(1000);  
fastFlasher.flash(10);  
delay(2000);  
}
```

Arduino IDE不会允许你直接将写好的Sketch保存在“libraries”文件夹下，所以在别的地方新建一个名为“Simple Flasher Example”的文件夹将其保存到里面，然后将整个“Simple Flasher Example”文件夹移动至你的库文件夹下的“examples”文件夹下。

重启Arduino IDE，你就可以在图11.2所示位置找到你的范例Sketch了。



图11.2 打开范例Sketch



11.4 总 结

关于C++和库的编写有很多很多的内容，这一章只是将你带入其中，而且这对于Arduino的使用也足够了。Arduino是很小的设备，它经常可以让本需要较强工程学的项目变得简单、方便。

以上就是本书的主体部分了，如果想更进一步地学习Arduino的知识，Arduino的官方网站www.arduino.cc会是一个不错的切入点。另外，也可以看看本书网站www.arduino.cc，在那里你可以找到勘误表和其他的一些有用的资源。

如果你需要帮助和建议的话，www.arduino.cc/forum上的Arduino社区相当有用，笔者在那里的用户名为Si。