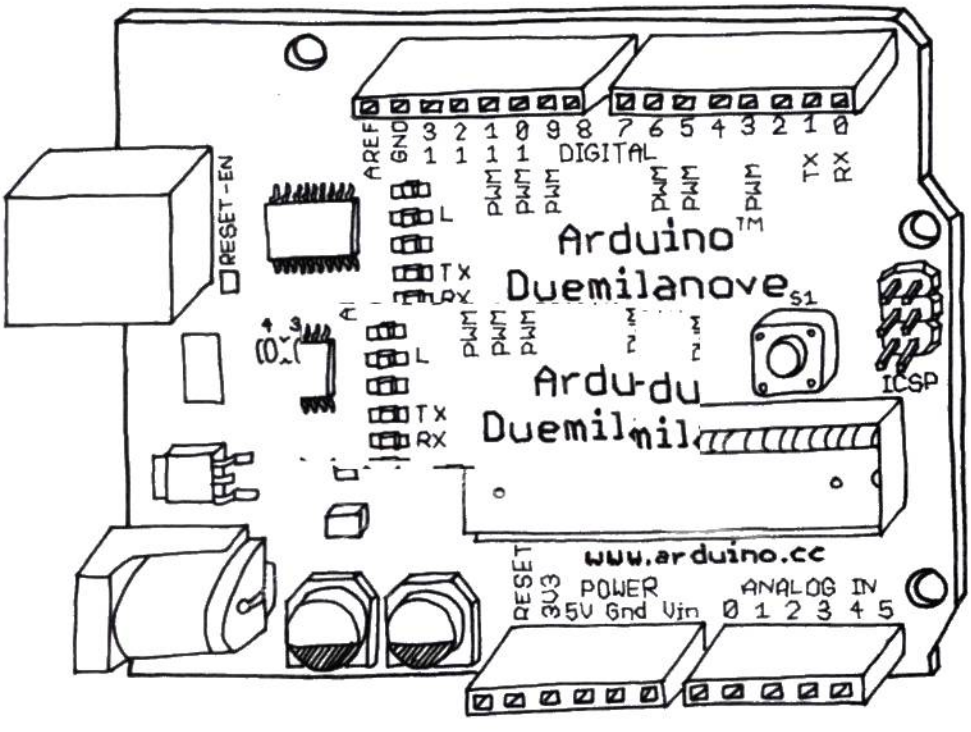
硬件开源
电子设计平台

爱上 Arduino

Getting Started with Arduino

[美] Massimo Banzi 著

于欣龙 郭浩赟 译



人民邮电出版社
POSTS & TELECOM PRESS

O'REILLY

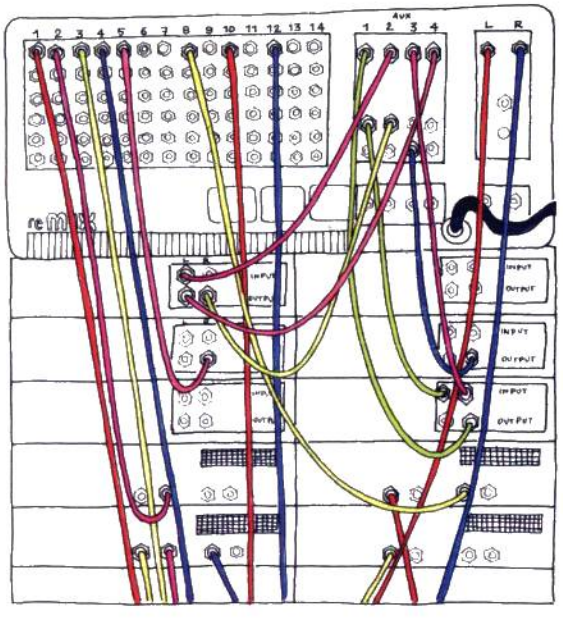
Make:
makezine.com

爱上Arduino

Arduino是开源电子原型制作平台，它引发了设计爱好者世界的一场风暴。本书中有透彻的Arduino介绍，它会给你带来许多项目点子，并帮助你顺利地实现它们。从开始策划直到完成安装，你所需的一切信息尽在本书当中。

从书中你将会学到：

- 交互设计和物理计算
- Arduino硬件和软件开发环境
- 电力和电子的基本知识
- 在焊接板上进行原型测试
- 绘制图表



更多内容：有了廉价的硬件和能免费下载的开源软件，爱上Arduino就非常容易了。要尝试本书的示例，只要一个USB Arduino，USB A-B线和一个LED就足够了。

加入数以万计的爱好者群，他们已经开发出了难以置信并能激发灵感的原型制作平台。本书由Arduino项目的合作创始人Massimo Banzi写成，《爱上Arduino》会把你带到快乐中去！

O'REILLY
oreilly.com.cn

Make:
makezine.com

封面设计：
Brian Scott（英文版）
张健（中文版）

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

分类建议：电子技术/计算机硬件

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-25350-7

ISBN 978-7-115-25350-7

定价：38.00 元

爱上Arduino

[美] Massimo Banzi 著
于欣龙 郭浩赟 译

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

爱上Arduino / (美) 班兹 (Banzi,M.) 著 ; 于欣龙, 郭浩赟译. -- 北京 : 人民邮电出版社, 2011.8
ISBN 978-7-115-25350-7

Ⅰ. ①爱… Ⅱ. ①班… ②于… ③郭… Ⅲ. ①单片微型计算机 Ⅳ. ①TP368.1

中国版本图书馆CIP数据核字(2011)第083998号

版权声明

Copyright ©2009 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2010. Authorized translation of the English edition, 2009 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

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

英文原版由 O'Reilly Media, Inc. 出版 2009。

简体中文版由人民邮电出版社出版 2010。英文原版的翻译得到 O'Reilly Media, Inc.的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 —— O'Reilly Media, Inc.的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

爱上 Arduino

◆ 著 [美]Massimo Banzi
译 于欣龙 郭浩赟
责任编辑 黄 彤
执行编辑 胡 洁

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：690×970 1/16
印张：7.25
字数：104 千字 2011 年 8 月第 1 版
印数：1 - 3 500 册 2011 年 8 月北京第 1 次印刷
著作权合同登记号 图字：01-2011-1846 号

ISBN 978-7-115-25350-7

定价：38.00 元

读者服务热线：(010)67132837 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

内容简介

Arduino是一个开源电子原型制作平台，包括一个简单易用的电路板以及一个软件开发环境。

Arduino既可以独立运行，又具备互动性。它可以与PC的外围装置相连接，还能与PC软件进行沟通。它在电子设计爱好者们中间引发了一场风暴。

本书透彻地介绍了Arduino的相关内容，它会给你带来许多制作项目的点子，并帮助你顺利地实现从开始策划直到完成安装的全过程。本书适合电子专业、互交设计专业、新媒体技术专业学生阅读，也是电子爱好者开展电子制作项目的参考手册。

译者序1

我于哈尔滨工程大学本科毕业。大学期间参加过各种科技类学术竞赛，曾作为主力队员参加2007年“中国机器人大赛暨RoboCup”中国公开赛，并荣获空中机器人组固定翼15公斤冠军、5公斤级亚军。由于我有较强的科技创新能力，学习成绩优异，因此获得免试保送本校硕士研究生的资格。现在攻读本校机械设计及理论专业，主要研究水下作业和特种机器人技术。近两年一直带领师弟师妹们参加中国机器人大赛，接触到了许多热爱机器人制作的朋友。在比赛中，我们的作品都面临整合各种系统的挑战，如果意志不坚定，很可能会失去信心和兴趣。所以选择一款合适的控制器做系统成为了开启机器人之旅大门的钥匙。

在我刚读本科的时候，参加的比赛多数是使用51单片机做控制核心。但对于非电子专业的学生来说，搞懂什么是寄存器、怎么使用中断函数，是一件难事。为了让学弟们快速掌握单片机硬件资源和软件编程，顺利投入机器人大赛作品制作中，当时下了不少功夫，最后结果却不尽人意。2008年的冬天，当人们陶醉在银装素裹的雪景中时，我和学弟们几乎每天泡在学校实验室里，偶然浏览一个国外机器人网站，发现他们做的机器人使用的控制器都是“Arduino”。Arduino是什么？当时大家并不知晓，通过几天搜索发现，在国外应用案例很多，也易于操作。让我喜出望外的是，相关作品设计资料全部开源，并且可以改进利用。我想到了Linux操作系统，这是自由软件和开放源代码发展中最著名的例子。希望以后Arduino能像Linux一样成功。

有很多人问我“Arduino”翻译过来是什么意思，其实它是11世纪北意大利国王的名字。后来被本书的原著者引用到他设计的这个控制器上。因为其程序语法简洁易懂，技术门槛低，吸引了不少各行各业的设计师加入其使用的行列。我作为第一批将其引入国内的爱好者，见证了它的成长历程，它有自己的文化，一种代表开源创意的文化，在其文化的背后，又有科学技术与艺术的结合。它能帮助你在现实与虚拟世界之间搭建桥梁，可促使你脑海中的创意模型快速呈现。

你可以参考Arduino的官方网站*http://www.arduino.cc*或者进入Arduino爱好者Rebecca的博客*http://blog.sina.com.cn/arduino*，通过链接你会找到许许多多创意作品的源码。我衷心希望本书能够给广大Arduino爱好者和初学者提供帮助，实现自己的每一个创意。在翻译本书的过程中，得到了《无线电》杂志黄彤老师的大力支持，本书的部分章节由郭浩赟编写，另外王晶也为本书的校对做了大量工作，在此，对他们表示衷心感谢。

由于译者水平有限，错误和疏漏之处在所难免，欢迎广大读者指正。我的联系邮箱是*robotbase@yahoo.cn*。

于欣龙

译者序2

2007年初，因为在学习互动装置的课程从而接触Arduino。网上也并没有太多的学习资料和开放资源。所以一路走来也磕磕绊绊颇有感慨。

Arduino的诞生是为了帮助那些零基础的人用电子元器件完成他们自己的创作和想法。所以本书的作者在写作时并没有用很多枯涩的专用名词和技术解析，取而代之的是一些编程思路和解决问题的技巧。

我也在一些艺术院校带的相关课程中也会发觉部分学生太沉溺与技术问题反而迷失了使用Arduino的初衷。技术研究的太深入反而会更觉迷茫。大家其实可以在大体了解Arduino功能的基础上，照着自己想玩的小东西去制作，有目标的小实验和小成就反而更有趣！推荐参考《爱上制作》里面的项目。我很期待大家在看完本书之后创造出那些富有个性的奇奇怪怪的小玩意儿。也希望中国会有越来越多的Tinker。

感谢《无线电》杂志的编辑在出版和翻译过程中做出的努力，以及在本书的翻译过程中给我意见的长江艺术与设计学院陈碧如老师和中国美术学院的施洪法老师等。

郭浩赟

推荐序

研究室外操场上的橄榄球队，一直在零散热身。我所在的项目组，经过漫长的预备期，在正式启动一年半时搭出了平台级别的原型。

原型，在此时映射出各方的寄望，给予投资人信心，帮助主导者做出策略调整，设计者也能从中得到理念的验证，进而在某种意义上降低长期项目的风险度。对原型制作的重视，并不只发生在大型公司的研发与实验机构，以创新设计为目标的机构与院校（IDEO, MIT, D - School, KMD），也在强调它参与设计流程的必要性并在实际施行，如，在 KMD 这个媒体设计研究院的工作室，堆满了PCB制版机、切割机、焊接器、激光切割机、3D 打印机等辅助原型制作的设备。创造力、智能化、工程与程序、模型工具勾勒出了当今生产与制造模式的轮廓。而原型与原型制作，在这些机床工具、模具工具、编程工具成为个人爱好者的今天，更给予了个人及小型创作团体扩大影响力的机会，如今借助轻量级别工具，一个小型团队就可以快速构造一个包含基础功能硬件，移动平台程序跟云端服务的原型，展示系统运作，同时缩减开支，减少创新项目风险。

Arduino、Processing、Open Frameworks、Laser Machine、3D Printer等，是以上提及的轻量生产力工具的典型代表，尤其是Arduino，这个为适应设计院校需要而生的简易电子创作入门工具，如今被应用到各类环境中：交互设计、新媒体艺术教育与创作、产品研发、科学实验、机器人研究等。社会化媒介（Social Media）的创新渐入佳境，普及运算（Ubiquitous Computing）声名鹊起，现在借助越来越多与此相关的轻量级创作工具，个人的、团体的创意与创造力将能够最快的递交到社会并施加影响。因此，在电子硬件与程序媒介已经成为艺术，设计教育中基础课题并与创作、创新生产力紧密关联的今天，了解、认识、学习并使用这些工具就显得尤其必要。

虽然新的工具还需要与之对应的新的设计方法论的配合才能够发挥最大作用，但在这新世纪的起点，新的生产与制造模式还在被探讨、被验证之时，从个人角度去尝试新的工具，不仅能够品尝到与以往不同的乐趣，也是应对未来的准备。

感谢本书的编辑和译者，以及无数为开源软硬件普及做出贡献的个人、组织、机构。他（她）们共同培养了一个新的群体，这个群体的成员背景多样：工程师、程序员、研究员、设计师、艺术家等，但使用共同的材料，通过网络与实体空间共享知识，热心帮助他人，尝试开放式的合作与开源式的创作。

相信这本系统介绍Arduino前世今生并引领读者开始基础使用它的书籍，能够吸引更多的人加入这个群体，开始个人创作、原型尝试。

高磊 imlab.cc

2011年5月于神奈川，日吉

目录

前言 1

1/介绍 5

目标读者 6

什么是Physical Computing？ 6

2/Arduino理念 8

原型 9

Tinkering 10

Patching 11

改装电路 13

改装键盘 15

我们爱垃圾！ 17

改装玩具 18

合作 19

3/Arduino工作平台 20

Arduino硬件 20

Arduino集成开发环境（IDE） 23

安装驱动程序：Macintosh操作系统下的方法 24

安装驱动程序：Windows操作系统下的方法 24
识别通信端口：Macintosh操作系统的情况 25
识别通信端口：Windows操作系统的情况 26

4/Arduino入门 28
解析互动装置 28
传感器与驱动器 29
LED闪烁 29
编写程序 31
给我个奶酪（Parmesan） 33
Arduino从不停止 34
真正的Tinker都写注释 34
代码，一步一步来 34
我们将会做什么？ 37
什么是电？ 38
使用按钮控制LED灯 40
它是如何工作的？ 42
一个电路，一千种用法 43

5/ 高级的输入输出控制方法 48
尝试其他开关类型传感器 48
使用PWM方式控制灯光亮度 51
使用光线传感器取代按钮 57
模拟输入 58
尝试其他模拟传感器 61
串行通信 62
驱动较大功率负载设备（直流电机、灯泡等） 63
复杂传感器 64

6/互动云 65
制订计划 67
编写程序源代码 68

组装电路 73
下面介绍如何安装 74

7/排疑解惑 76
测试板子 77
用面包板测试电路 78
将问题独立出来 79
开发环境（IDE）常见问题 79
利用网络资源解决问题 79

附录A/面包板 83

附录B/认识电阻和电容 85

附录C/Arduino语法参考 87

附录D/阅读电路简图 101

前言

几年前我接受了一项非常有趣的挑战任务：教设计师们学习一些初步的电子入门知识，然后他们自己动手做出一些他们之前设计的互动原型物件。

我下意识的根据我从学校学到的方法开始教他们电子电路知识。不久之后我就发现这并不简单，教学效果并没有我希望的那么好。教室里的情况糟糕极了，他们只是听我讲那些无聊至极的理论而没有实际动手去操作。

实际上，我在入学之前已经通过自己实验的方式学到了很多电子电路的知识：虽然只有非常少的理论，但是积累了很多动手的经验。

我开始思索我学到那些知识的过程：

- 我拆开那些我能拿在手上的小型电子设备。
- 我慢慢地学习认识那些电子元器件。
- 我开始调试它们，改变一些内部连接，看它们会发生什么变化：通常会发生一些爆炸或者是开始冒出烟雾。
- 我开始搭建一些电子杂志附带的小套件。
- 我组合那些我改装过的设备和那些不错的套件以及其他的电路。让它们变成具有新的用途的东西。

作为一个小孩，我对于探索事物如何运作非常着迷。因此，我常拆分它们。这样的热情使我逐渐把目标定在那些家中闲置的东西，把它们拆成很小的部件。

然后人们常常带来所有分类好的设备让我来拆卸。

最后人们都主动把自家东西拿来给我拆。当时我最感兴趣的是一台洗碗机和一家保险公司拿来的计算机，计算机附带一个巨型打印机，还有电子卡片、磁卡读卡器和一些其他零件，要彻底拆开也是个巨大的挑战。

拆过很多东西以后，我基本知道了电子元器件是什么和它们的大概作用。加之我爸爸可能从20世纪70年代初就开始购买电子杂志，堆在家里到处都是，我每天都花几个小时去阅读上面的电路图，尽管似懂非懂。

一遍又一遍地读文章，又常常拆各种东西，这两者渐渐形成了良性循环。

圣诞节是一个巨大的飞跃。这一天我爸爸送给了我一个工具箱，里面有帮助年轻人学习电子元器件的很多工具，每一个工具都装在小盒子里，小盒子上带有磁性小方块儿，能够与其他盒子相连接，顶端标着各自的电子符号。我当时不知道这是德国的标志性产品，是由Dieter Rams在20世纪60年代设计的。

有了这套新工具，我很快学会了怎样组装电路并试用，建立模型需要的时间也越来越短。

之后我自己制作了收音机、扩音器，还有的电路能够发出巨大的噪声也能放出美妙的音乐，我还设计了雨天感应器、小机器人。

很久以来我都想找到一个英语单词来形容这种工作方式：没有特别的目的，从一个模糊的想法开始，得到一个完全意外的结果。后来发现了“tinker”这个词，我注意到这个词在很多其他领域都被用来描述某种操作方式，也用来描述那些探索的人们。例如，法国创造了“Nouvelle Vague”的那一代导演就被人们称为tinkers。我找到的tinker的最佳定义来自旧金山的探索展馆的一次展览：

“Tinkering就是你开始做一件不怎么确定的事情的过程。只由灵感、创意和想象力和好奇心指引着，没有操作规则，也就没有失败，没有正确和错误。整个过程都是在观察事物的情况并不断修整它们。”

使用Arduino

新发明、小装置，各种风马牛不相及的东西和谐地工作，这就是tinkering。

从最根本上来说，tinkering就是探索和娱乐的结合。

我从早年的实验中明白了要想设计一个电路，让它能够控制着每一个元器件都按你的想法工作，需要积累多少经验。

1982年的夏天我迎来了另一个突破。在伦敦，我和父母一同参观了科学博物馆。那里新开了一个侧厅，展示计算机相关的展品。在有引导的实验中我大概了解了二进制数学和编程基础。

在那儿我明白了工程师已经不再用基础元器件设计电路，而是使用微处理器在很多产品中添加智能。软件为电子设计节约了大量时间，也使得tinker的过程周期越来越短。

回来以后我就开始攒钱，因为我想买台计算机学习编程。

我的第一个工程也是最重要的项目就是用我那台崭新的ZX81计算机控制一台焊接机。我知道这听上去没什么意思，但是因为当时有这个需要，而且因为我刚刚学习编程，这对我来说也着实是个挑战。这时我明白了写代码比不停地测试复合电路方便得多。

20多年过去了，我觉得这种经历可以教会那些连数学课也没上过的人把热情融入到tinker的过程中去，就像我少年时和那以后一直保持的一样。

——马西莫

致谢

此书要献给 Luisa和Alexandra。

首先我要感谢Arduino团队的伙伴们：David Cuartielles，David Mellis，Gianluca Martino和Tom Igoe。与你们合作的经验真的非常棒。

Barbara Ghella，她可能并不知道，但是如果没有她宝贵的建议，也许就不可能会有Arduino和这本书。

Bill Verplank，教了我更多Physical Computing以外的知识。

Gillian Crampton-Simth，给予我这个机会和从她身上所学到的一切。

Hernando Barragan，所投入在Wiring的精力。

Brian Jepson，称职的编辑，也是一直以来的热情支持者。

Nancy Kotary，Brian Scott，Terry Bronson和Patti Schiendelman把我所写的内容更流畅的呈现出来。

我还想感谢更多人，不过Brian 说我已经没有多余的空间了。以下是一小部分我特别想感谢的人：Adam Somlai-Fisher，Ailadi Cortelletti，Alberto Pezzotti，Alessandro Germinasi，Alessandro Masserdotti，Andrea Piccolo，Anna Capellini，Casey Reas，Chris Anderson，Claudio Moderini，Clementina Coppini，Concetta Capecchi，Csaba Waldhauser，Dario Buzzini，Dario Molinari，Dario Parravicini，Donata Piccolo，Edoardo Brambilla，Elisa Canducci，Fabio Violante，Fabio Zanola，Fabrizio Pignoloni，Flavio Mauri，Francesca Mocellin，FrancescoMonico，Giorgio Olivero，Giovanna Gardi，Giovanni Battistini，Heather Martin，Jennifer Bove，Laura Dellamotta，LorenzoParravicini，Luca Rocco，Marco Baioni，Marco Eynard，MariaTeresa Longoni，Massimiliano Bolondi，Matteo Rivolta，MatthiasRichter，Maurizio Pirola，Michael Thorpe，Natalia Jordan，Ombretta Banzi，Oreste Banzi，Oscar Zoggia，Pietro Dore，Prof Salvioni，Raffaella Ferrara，Renzo Giusti，Sandi Athanas，Sara Carpentieri，Sigrid Wiederhecker，Stefano Mirti，Ubi De Feo，Veronika Bucko。

1/介绍

Arduino是一个开源的、拥有简单输入/输出（I/O)的电路板，它沿用了Processing语言的开发环境。Arduino可以用来开发独立运作互动装置，或者可以连接到你计算机上的软件（例如：Flash、Processing、VVVV或Max/Msp）。你可以自己动手组装这个电路板，或者直接购买套件；开源IDE（集成开发环境）可以免费从**www.arduino.cc**下载。

Arduino和目前市面上其他平台相比，有以下特点：

- 支持多种操作系统：Windows、Macintosh、Linux。
- 为了方便设计师以及艺术家的使用，采用Processing的集成开发环境。
- 程序是通过USB而并非串行端口实现的。这一点非常实用，因为现今的多数计算机都没有串行端口。
- 这是开源的硬件和软件——如果你愿意，可以直接下载电路图，购买所需的电子元器件来自己制作，无须从制造商那里购买。
- 硬件很便宜。USB电路板只要20欧元（*译者注：差不多35美元，目前在中国所有的元器件成本仅需65元人民币左右*），如果只是替换电路板上烧坏的微处理器，那么只需5美元（*译者注：在中国是20元人民币左右*），所以你就算做错了，烧坏了部分元器件，也不用心疼。
- 有一个活跃Arduino玩家的论坛，所以你可以随时找人帮助你。

» Arduino Project是以教育为宗旨而开发的，因此就算是初学者也能很快地入门。

这本书的目的是帮助初学者理解如何使用Arduino平台并培养自我学习精神。

目标读者

这本书一开始就是针对设计师和艺术家所出版的。所以，书中解释和叙述的方式肯定会让一些工程师无法接受。事实上，这章节的一个草稿版已经被某位工程师形容为“看得发毛”。不过那也正是作者我的目的。工程师自己也常常无法向其他工程师解释自己的逻辑和做法。让我们现在就开始深究所谓的“发毛”吧！

注意

这样的方式，以及某些“有争议”的设计，也可延伸为利用科技来创造原型，尤其是电子科技的原型。

当Arduino开始盛行时，我意识到有很多实验者、兴趣爱好者和骇客已经利用它来进行各式各样的非常漂亮和疯狂的创作了。我相信你们都是艺术家和设计师，所以这本书适合任何人。

Arduino是为了传授互动设计而诞生的，把设计原型的能力定为使用者的学习目标。关于互动设计有很多不同的说法，但我偏爱下面的定义：

互动设计就是设计互动体验。

如今，互动设计意味着人与物体间具有深意的体验。它是一种探索科技美以及人对现代科技之体验的创作。互动设计鼓励一种不断制作原型并由此使创作逐步完善的设计过程。这样的方式也可引申阐释为：利用科技来创造原型。更具体地说，利用Arduino做互动设计称为物理计算（Physical Compution）或者物理互动设计（Physical Interaction Design）。

什么是Physical Computing?

Physical Computing利用电子零件为设计师和艺术家的新素材制作原型。

这包括了用传感器和驱动器设计出与人互动的装置，并且通过微处理器（单片机）上的软件来控制整个互动装置。

在过去，一说到要处理有关电子方面的事情，就总是会让人想到去把工程师找来，把多个小元器件拼凑成整个电路——这些问题使得做创意的人始终无法进入电子领域。大多数的工具都是为工程师设计的，并且想使用这些工具也需要很多相关知识。近几年来，微处理器变得更加便宜和更容易使用，也衍生出了很多实用的工具。

我们做的Arduino就是为了让初学者在经过两三天的学习就可以开始制作装置，而艺术家和设计师们，也可以通过Arduino很快地学习到电子电路以及传感器的基本知识，用很小的花费就可以制作出设计原型。

2/Arduino理念

Arduino的理念就是不要光说不练，要多动手做。我们不断追求更快更有利的方法来制作原型，利用双手探索更多的原型技术，并开发多方位的思维。

典型的程序思考模式偏向于单向思考，例如，如何从A到B；Arduino采用多方位思考模式，也许有时候会迷失方向，也有可能会发现新的目的地C。

把玩所有工具，找出意想不到的结果，这就是我们崇尚的Tinkering模式。在探索制作原型的过程中，我们可以选择一些套件，以便我们随时调用整块硬件和软件。

接下来的几个章节会介绍由Arduino理念引出的想法、事件和开拓者。

原型

原型是Arduino学习的核心：

我们制作东西，与其他物件、人和网络进行互动。我们找寻更快、更直接、更简单的方式来制作尽可能便宜的原型。

很多接触电子的初学者以为他们必须从头学起，但这根本就是浪费精力的做法：真正激发人前进的动力是快速看到成果。因此对初学者而言，他们的第一步应该是确认东西可以快速地制作出来而又能正常地运作，或是成果展现时能激发他人投资的欲望。

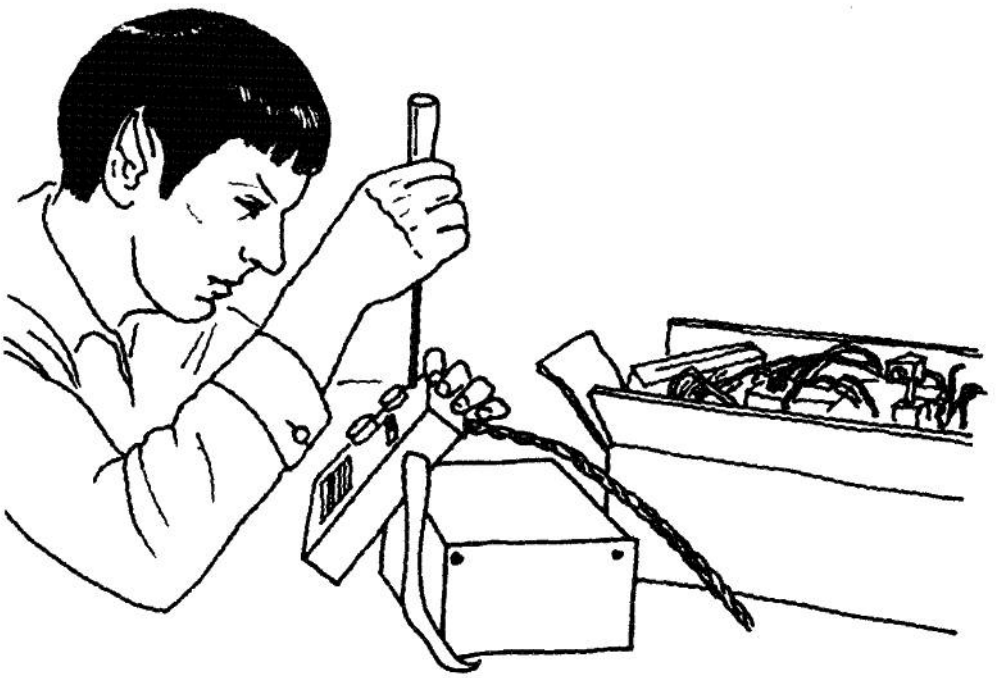
这就是为什么我们要发明“取巧原型制作法”（opportunistic prototyping）：当我们可以改装一个现成的由大公司和优秀工程师们开发的精密设备时，还有什么必要花费大量精力和时间从零开始呢？

譬如说，大师James Dyson试了5127 种吸尘器原型，才做出满意的成品(**www.international.dyson.com/jd/1947/asp**)，我们直接从他的成果中学习就好了。

Tinkering

我们相信探索任何硬件与软件的可能性是非常有必要的——有时并没有非常明确的目的。

利用现有的技术，尽量使用一些廉价或者废弃的旧设备，尝试改造它们并做出全新的东西，这就是最好的Tinkering。

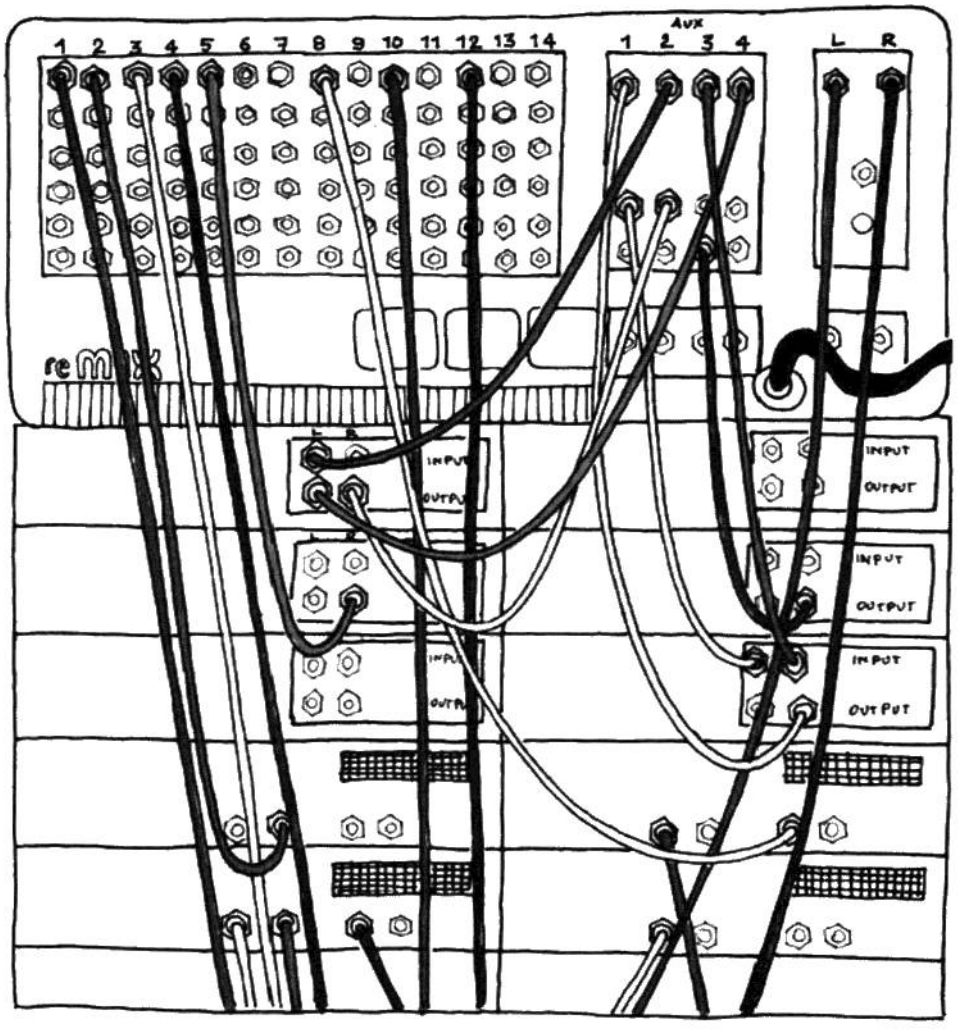


Patching

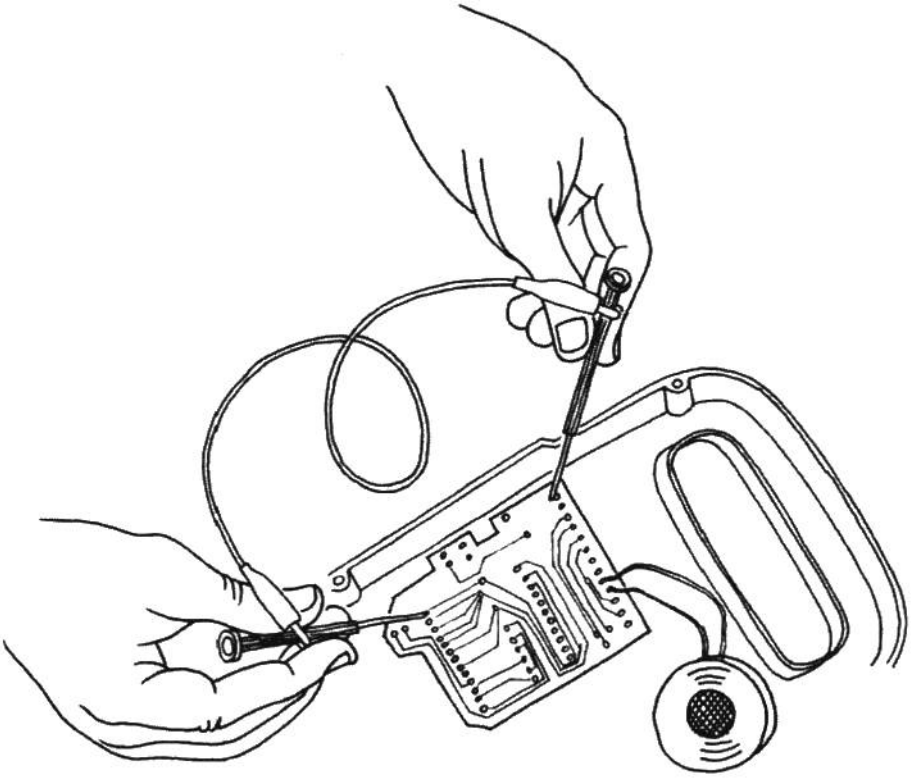
我对于使用简单的模块构建复杂的系统很感兴趣，最具有代表性的作品就是Robert Moog所制作的模拟电子音乐合成器。音乐家利用各种方式来连接不同的模块，从而编制出新奇的声音，这样的方式虽然让合成器看起来像是电话配线箱，但如果与一些旋钮结合，便能成为创新音乐平台。Moog形容这是一种“探索和见证”的过程。我相信大多数音乐家一开始都不懂这上百个旋钮的效果，但他们会不断地去尝试，找出适合自己风格的音乐。

减少创作过程的中断次数，对于整个创作流程是非常重要的。因为没有太多障碍和开发过程，更有利于激发创作的想法。

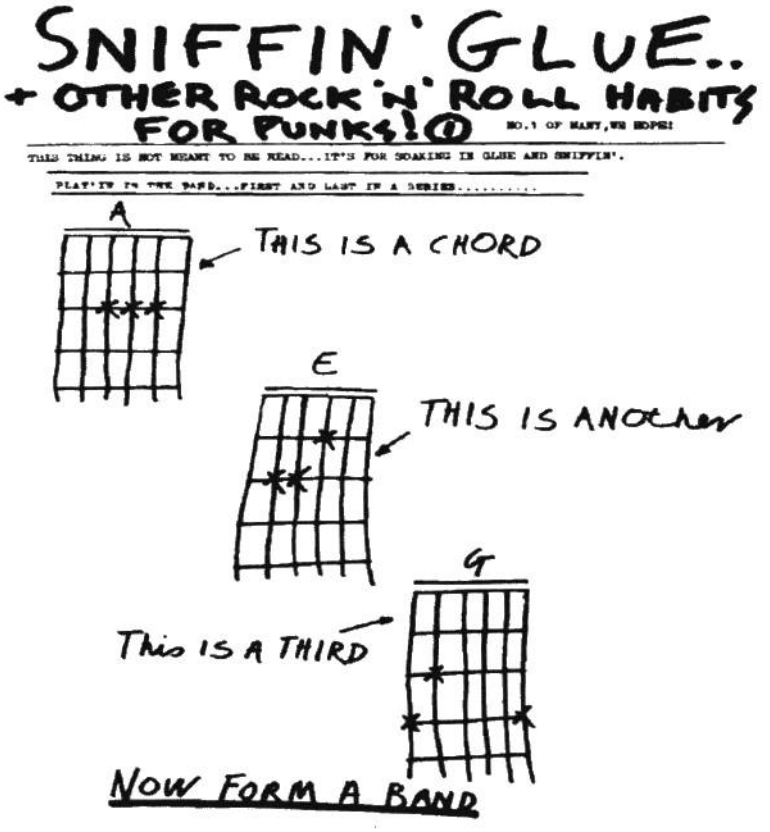
这样的编程概念在软件中被称为“视觉化编程”环境，例如MAX、Pure Data 或者VVVV。这些工具可以把不同的功能包装成“方块”，使用者可以按照自己不同的需求将方块连接起来。这些软件让使用者不须要被写代码所困扰，同时也能够体验到写代码的乐趣。如果采用传统编程方式，常常是一个令人受挫折的过程：编程，编译，该死！——出现错误，找错，重新编译，再执行。如果你的思考逻辑比较视觉化，建议你可以尝试上述这些工具试试看。


1 2 3 4 5 6 7 8 9 10 11 12 13 14
AUX
1 2 3 4
L R
INPUT
OUTPUT
INPUT
OUTPUT
INPUT
OUTPUT

改装电路

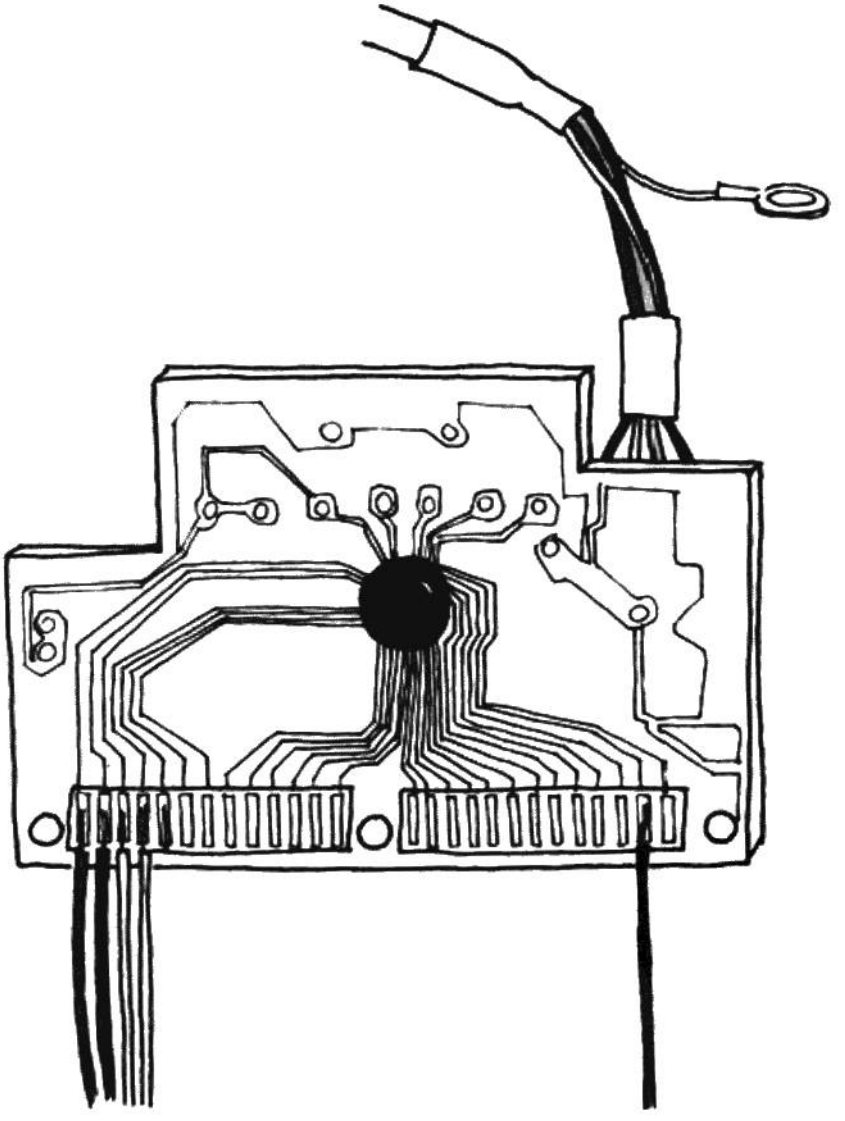


改装电路是Tinkering最有意思的方式之一：让一些低压电路、电池供电的电子音频设备（像吉他音效器、小孩们的玩具或者合成器）产生短路后制造出全新的乐器或者发声器。借此我们也可以深入了解“艺术的可能性”。电路改装源于1966年，Reed Ghazala无意中将一个玩具的扩音器和抽屉里的金属物件短路了，结果产生了一连串不同寻常的声音。我喜欢改装电路，因为它不需要任何理论上的解释或者专业知识就能创造出与众不同的装置。



就像是Sniffin Glue杂志爱好者所说的："在朋客时代，学会三个吉他和弦就可以组团了。"别让任何专业人士看扁了，别说你做不到像他们那样好。不用理会他们，你会让他们感到吃惊的!

改装键盘

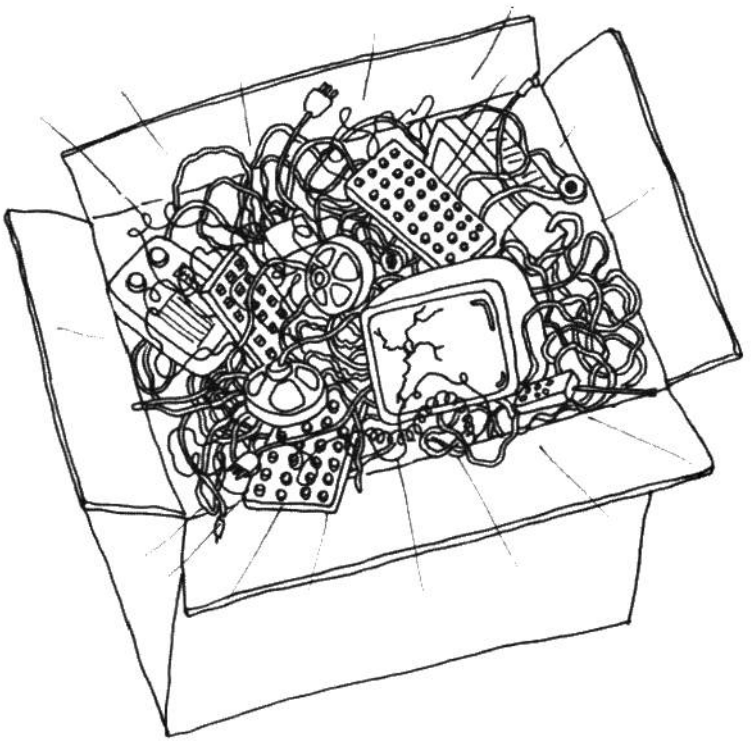


经历了超过60年的发展历史，计算机键盘始终是人与计算机互动的主要工具。MIT媒体实验室的学术带头人Alex Pentland曾说道：“原谅我的表达方式，但是，连坐便器都比计算机智能，真的。因为计算机与外界是完全隔离的。”

作为Tinker，我们可以用一些新的方法来和程序互动，替换相关部件从而使装置能够感应到周遭的变化。把计算机键盘拆分之后，我们便能够看到一个简单又廉价的电子部件。键盘的核心是个小型电路板，通常是绿色或者咖啡色的双层电路板，它负责接收键盘的输入信号。如果将连接键盘的线路移除，再用电线连接两个点，就会在屏幕上看到一个字母的输入。如果买一个能运动的传感器来连接键盘，每当有人在计算机前经过时，就会看见字母被输入。学习改装键盘，是制作互动装置原型与物理计算（Physical Computing）的关键基础。

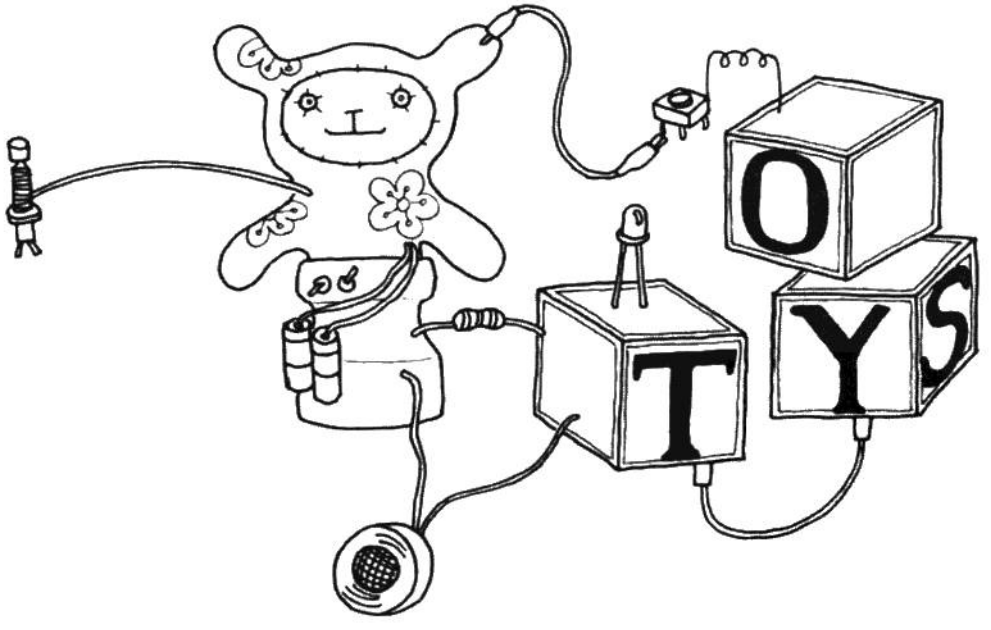
我们爱垃圾！

随着技术日新月异地变革，人们淘汰了很多使用旧技术的设备：旧的印表机、计算机、各式各样的办公设备、机械设备，甚至于军用器材都被丢弃。一直以来，这些过时的科技产物有很大的市场，特别是对于那些并不富裕的改装高手或者初学者。这个市场在意大利西北方向的一个城市Ivrea，也就是Arduino的原创地，这个城市原是Olivetti公司的总部。他们从20世纪60年代就开始生产计算机，一直到90年代中期，他们把所有的计算机零件、电子元器件和各式各样的装置都丢弃在附近的废物回收厂。我们在那边花了很长的时间，用很低的价格收购任何可以回收利用的东西。当你可以用很少的钱就买到数千台扬声器时，你一定会想出一些好点子来利用它们。与其从零开始，不如从累积的破铜烂铁中找出可以再利用的零件。



改装玩具

就如前文所提到的电路改装，玩具是非常理想而又便宜的技术改装资源。现在市面上涌入很多低价位高科技玩具，让你能轻易试验不同玩具，并拼凑出创意，例如，会发出声音的猫或者是光剑。近年来，我不断灌输给学生们一个观念：科技既不可怕，也不是难以接近的领域。Usman Haqure 和 Adam Somlai-fischer 所著的《Low Tech Sensors and Actuators》（lowtech.propositions.org.uk）清楚地描述了各种改装技巧，我个人也常常翻阅参考他们所写的书。



合作

使用者互相交流合作是Arduino成功的关键原则，通过Arduino世界论坛 www.arduino.cc，来自世界各地的使用者都能互相学习如何运用Arduino平台。Arduino团队鼓励使用者在当地建立论坛，也帮助访问过的每一个城市建立用户群。同时我们也建立了一个Wiki叫“Playground”(www.arduino.cc/playground)，让使用者记载他们研究的成果。令人兴奋的是，我们看到很多人愿意在网络上大方地分享知识来帮助其他人，这种分享与互助的文化是Arduino最自豪的部分。

3/Arduino工作平台

Arduino包含两个主要的部分：硬件部分是可以用来做电路连接的Arduino电路板；另外一个则是Arduino IDE，你计算机中的程序开发环境。你只要在IDE中编写程序代码，将程序上传至Arduino电路板后，程序便会告诉Arduino电路板要做些什么了。

在Arduino诞生之前，设计电子电路还必须拼凑上百种诸如电阻、电容、电感、晶体管等各式各样的电子元器件。

每个电路都有独特的程序，要改装这些电路你必须切断线路，焊接连接等。

随着电子技术和微处理器的发展，这些线路大多被软件程序所代替。

软件比硬件更加容易修改。只需几行代码你就可以从本质上改变设备的逻辑，并尝试不同的两三个版本。这只花费了你焊接几个电阻的时间。

Arduino硬件

Arduino由一个小型微处理器和一个电路板所构成。这颗芯片如同一台微型计算机，它的运算能力虽然比苹果笔记本计算机少几千倍，但非常便宜、有用而且有趣。看看Arduino板上吧：你会看见一块有28个“脚”的黑色细长芯片——它就是整块板的心脏，名叫ATmega168。

我们（Arduino团队）将芯片所需的所有电子元器件都焊接完了。不同的电路板代号也各不相同。本书示范时所使用的是代号为Arduino Duemilanove的电路板，是由早期的Arduino NG与Arduino Diecimila改良而来的。

在图片中，你看到Arduino有很多引脚，可能会很困扰。下面就来解释每个部分的作用和功能：

14个数字引脚（Digital IO pins 0～13)

这些数字引脚可以在程序中设定每个引脚是用于输出还是输入。

6个模拟输入引脚 （Analogue in pins 0～5）

这些模拟引脚用于读取各种模拟输入信号（例如读取某个感应器的电压），并在程序中将其转换成0～1023之间的数值。

6个模拟输出引脚（Analogue out pins 3、5、6、9、10和11）

这实际上是6个数字引脚，但可以由程序指定变更为模拟的输出。

电路板可以由USB或是外部的直流9V变压器供电。如果同时接上两种电源，供电顺序会以外接电源优先。

注意：

如果是用Arduino NG或者Arduino Diecimila版本，则没有自动选择电源的功能，必须自行手动更改电路板上的跳线（jumper）。跳线位置在电路板上USB引脚与外部供电引脚的中间，并标示有PWR_SEL字样。

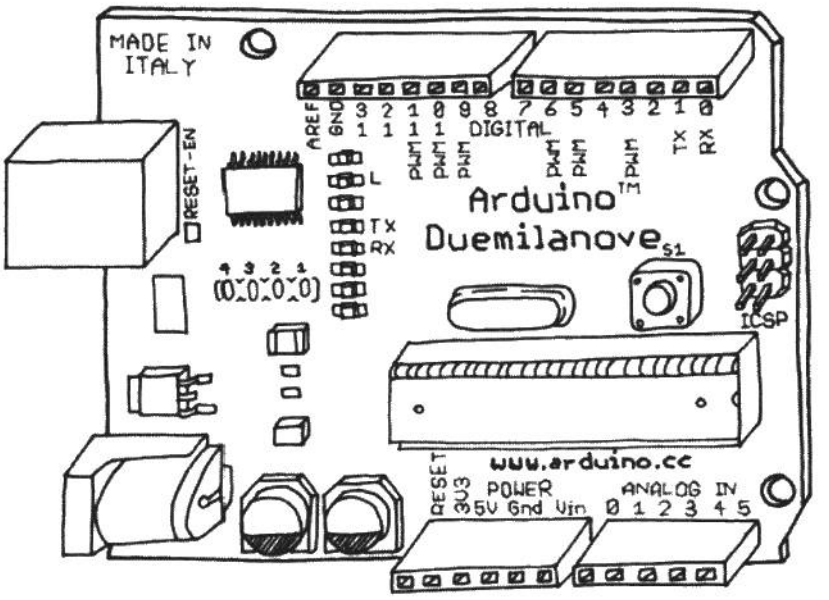


图3-1：Arduino Duemilanove

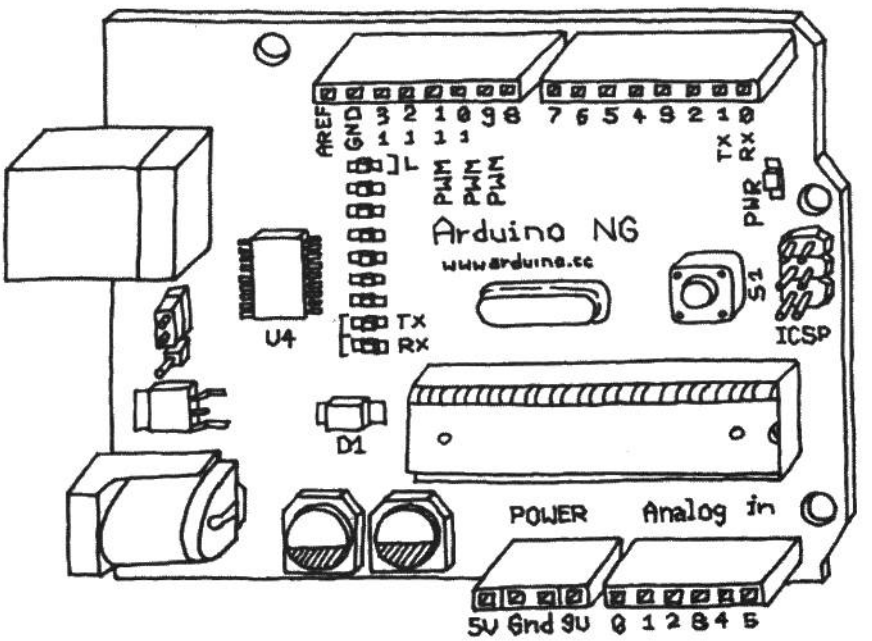


图3-2：Arduino NG

Arduino集成开发环境（IDE）

Arduino集成开发环境（IDE）是一个可以在你的计算机里运行的软件，你可以通过它来为你的Arduino上传不同的程序，而Arduino的编程语言也是由Processing（**www.processing.org**）语言改编而来的。

它的奇妙之处在于，当你把程序上传到Arduino时它会自动把你写的代码转换成C语言（C语言对于初学者来说非常晦涩），再传给avr-gcc编译器（一个重要的开源软件），然后把代码最终编译成微处理器能明白的指令。这些都是Arduino很重要的一部分，因为它隐藏了复杂的编译过程，让你用尽可能简单的方式去控制微处理器。

使用Arduino编写、执行程序的流程如下：

- 通过USB引脚把你的Arduino连上计算机。
- 开始编写代码。
- 通过USB上传你的代码，然后等待几秒，Arduino会自动更新。
- Arduino板会在数秒之后依照你所写的程序开始工作。

> **注意：**
>
> 在Linux下会有所不同，请参阅**www.arduino.cc/playground/Learning/Linux**。

在你的计算机上安装Arduino（IDE）

要开始使用Arduino前，必须先下载集成开发环境（IDE），地址是**www.arduino.cc/en/main/software**，按照你的操作系统来选择下载版本。

将下载档案解压缩之后，会看见一个名称为*arduino-[版本]*的文件夹，例如arduino-0012。你可以将文件夹放在任何地方，譬如桌面或者是你的*/Applications*文件夹（对于苹果操作系统而言），或者是*C:\Program Files*文件夹（对于Windows操作系统而言）。接着你可以打开文件夹，然后双击Arduino图标。当然，别忘记接上你的Arduino。

注意：

如果你有任何在Arduino集成开发环境运行时遇到的问题，请看第7章。

现在你必须装上能让你的计算机与键盘相沟通的USB接口的驱动。

安装驱动程序：Macintosh操作系统下的方法

在arduino-0012的文件夹中的Drivers子文件夹里，找到FTDIUSBSerialDriver_X_X_X.dmg。（*X_X_X*为版本号，例如*FTDIUSBSerialDriver_v2_2_9_Intel.dmg*），运行它，并把它加载到苹果系统中。

注意：

如果你用的是intel处理器，比如MacBook、MacBook Pro、MacBook Air、Mac Pro或者Mac Mini、iMac，请确认被安装的驱动程序的名字里带有intel的字样，如*FTDIUSBSerialDriver_v2_2_9_Intel.dmg*。如果不是intel处理器，则安装名字里不含intel字样的驱动程序。

下一步，点选FTDIUSBSerial驱动程序，并依照画面指示进行安装，如果出现密码询问窗口，请输入管理员密码。重启计算机后，接上USB传输线，电路板上的电源指示灯（PWR）就会亮起，而标有L的LED灯开始闪烁。如果灯没有亮，请参阅第7章。

安装驱动程序：Windows操作系统下的方法

请将Arduino连接上计算机；当显示“发现新的硬件”窗口时，Windows会先在Windows Update上面寻找驱动程序。

Windows XP会先问你是否查询Windows Update，如果你不想用Windows Update，选择“否”然后点击“下一步”。

接着选择“从指定位置安装”并按“下一步”。

选择“包含这个文件夹的位置”并按“下一步”按钮，按“浏览”，然后选取你解压缩的Arduino文件夹，选其下的Drivers\FTDI USB Driver文件夹，按“确定”，再点“下一步”。

Windows Vista则会先行在Windows Update上找寻驱动。如果失败，再由你指定驱动程序的安装文件目录。

安装驱动的过程需要进行两次：第一次是安装底层的驱动程序；第二次是安装将USB模拟成串口的驱动程序。

驱动程序安装成功后，就可以执行Arduino集成开发环境了。不过接着你要找到Arduino所用的序列串口编号，以便在IDE中能将程序烧录到板子上，这部分请参阅后面几个章节的说明。

识别通信端口：Macintosh操作系统的情况

打开Arduino IDE后，如图3-3所示，请在“Tools” →“Serial Port”下选择“/dev/cu.usbserial-序号”，这个是计算机识别出的Arduino电路板的名称。

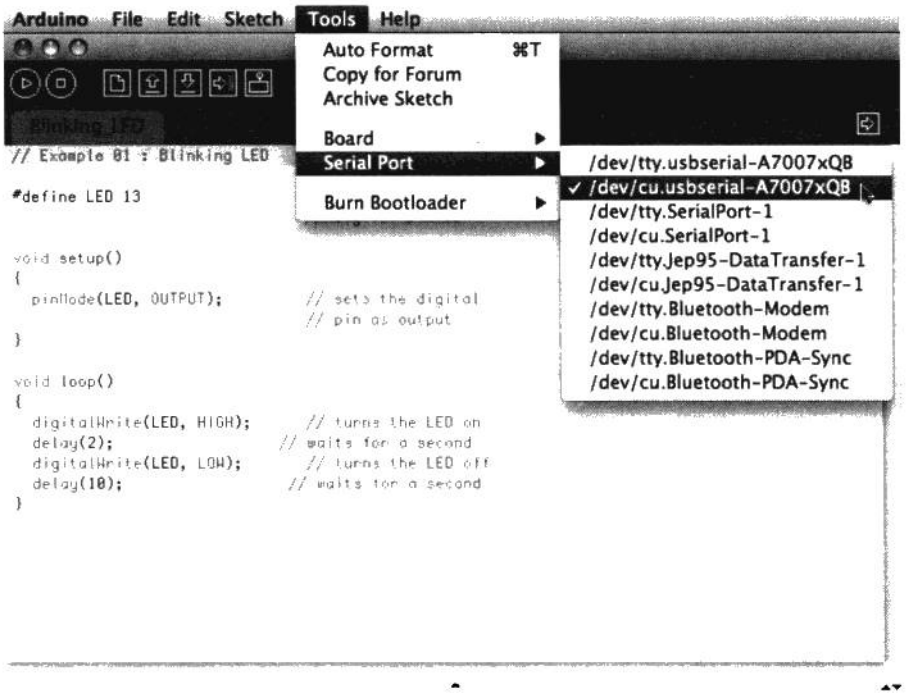


图3-3：串行端口的Arduino IDE列表

识别通信端口：Windows操作系统的情况

在Windows上，请按“开始”，在“我的电脑”上按鼠标右键，选择“属性”进入硬件，然后选择“设备管理器”。（Vista系统的可以直接在属性的左侧看到设备管理器。）

在“设备管理器”中的列表内找到“通用串行总线控制器”（COM & LPT），展开前方的加号，就会看到如图3-4所示的USB Serial Port，记下名称中连接的号码（COM3）。

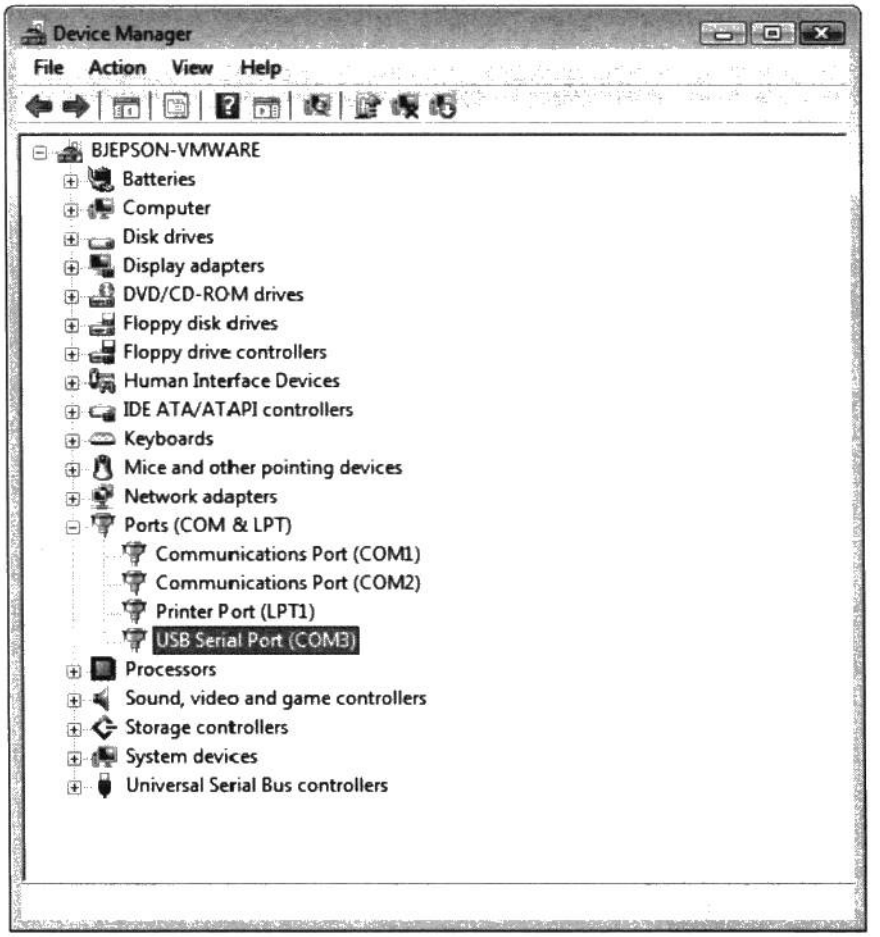


图3-4：Windows设备管理器显示的所有可用串行端口

注意：

在Windows操作系统中，有时候COM端口号会大于9，这对于Arduino IDE进行通信会有影响。遇见这种情况，请参阅第7章来解决这个问题。

知道了COM端口号之后，可以在Arduino IDE界面中选择“Tool”→“Serial Port”，并选择端口号。

现在Arduino IDE已经可以与电路板通信并烧录程序了。

4/Arduino入门

学习如何建立第一个互动装置

解析互动装置

使用Arduino制作出的物件统称为“互动装置”（Interactive Device）。互动装置是个能通过传感器（将现实生活中的衡量标准转换成电子信号的电子元器件）感知环境的电子电路。互动装置能通过程序来处理由传感器取得的信息，并且经由驱动传动装置和其他能把电子信号转成实体动作的电子元器件与外界互动。

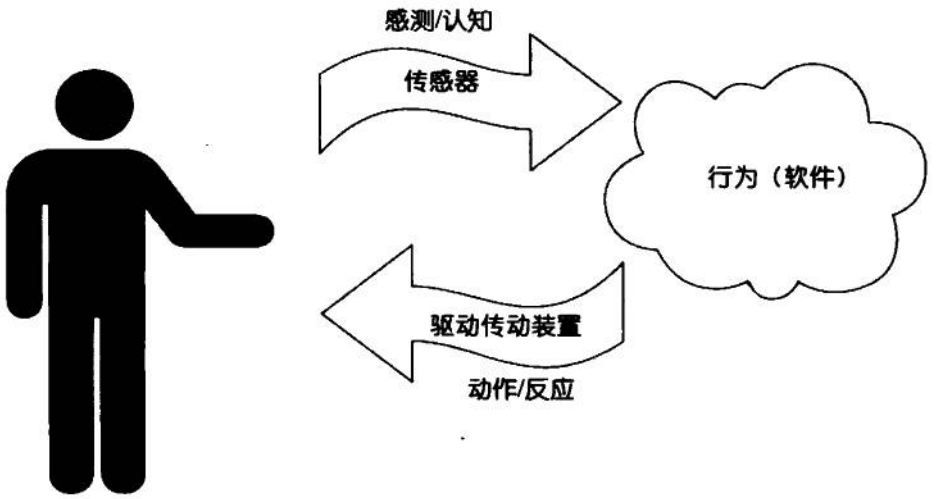


图4-1：互动装置

传感器与驱动器

传感器（sensor）与驱动器（actuator）是让电子装置能与外界沟通的部件。

微型处理芯片就好像是一台非常简单的计算机，它只能处理电子信号（就如同大脑中的神经元之间也是传递电子信号）。在现实世界中，光的明暗、温度的高低以及其他物理量，都可以通过传感器转换成电子信号。以我们人类的身体作比喻，就好像眼睛感受到光线后，转换成信号，再由神经传输到大脑。在电子电路的世界里，我们可以使用一种称为“光敏电阻”的电子元器件来感应到光的明暗程度，并转换成电子信号送给微处理器。

当微处理器经由传感器读取到环境变化之后，就会进行判断并做出回应。整个决策处理行为都由微处理器来运作，并利用驱动或者传动装置做出所需的回应。同样，这也好比肌肉接收到大脑传来的命令，就会转化成一个动作。在电子电路的世界当中，这样的功能可以用灯光或者电机来达成。

在接下来的几个章节里，你将会学到如何读取几种不同类型的传感器，以及如何控制几种不同的驱动传动装置。

LED闪烁

闪烁LED程序是Arduino初学者的一个程序，同时我们也能用它来测试Arduino电路板是否没有故障，通常这也是学习微型处理器程序设计的一个练习。LED灯是一个小型电子零件，如同灯泡般可以发光，但只需要很低的电压和很小的电量。

你的Arduino上已经预先安装了一个LED灯，方便刚开始接触的新手进行第一次测试。这个LED灯的位置会印有一个英文字母“L”。或者你也可以像图4-2那样，安插自己买的LED元器件到Arduino上。

自行安装LED灯要注意，图中较短的K脚为负极，较长的A脚为正极。

确定LED接好后，接着我们要用程序来告诉Arduino应该做什么事，也就是通过一连串的代码下达命令给微型处理器，让它完成我们想要做的事。

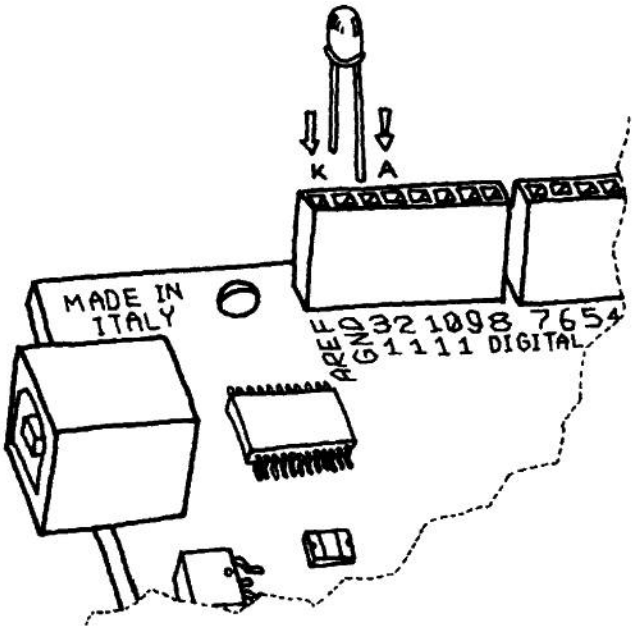


图4-2：LED与Arduino连接图示

编写程序

现在，在你的计算机上找到你存放Arduin IDE的文件夹，双击打开它。选择File/New来新建一个草稿文件（Sketch）并将其命名为“Blinking_LED”，再按“OK”按钮。在Arduino集成开发环境中的程序编辑界面会显示一个空白的编辑区，请参考以下的“例子01”。你也可以直接从**www.makezine.com/getstartedarduino**网站下载这段代码。程序见图4-3。

```
// Example 01 : Blinking LED

#define LED 13          // 定义LED引脚为13

void setup()
{
   pinMode(LED, OUTPUT);       // 定义13为输出引脚
}

void loop()
{
   digitalWrite(LED, HIGH);    // 设定LED灯开启
   delay(1000);                // 持续1秒
   digitalWrite(LED, LOW);     // 设定LED灯关闭
   delay(1000);                // 持续1秒
}
```

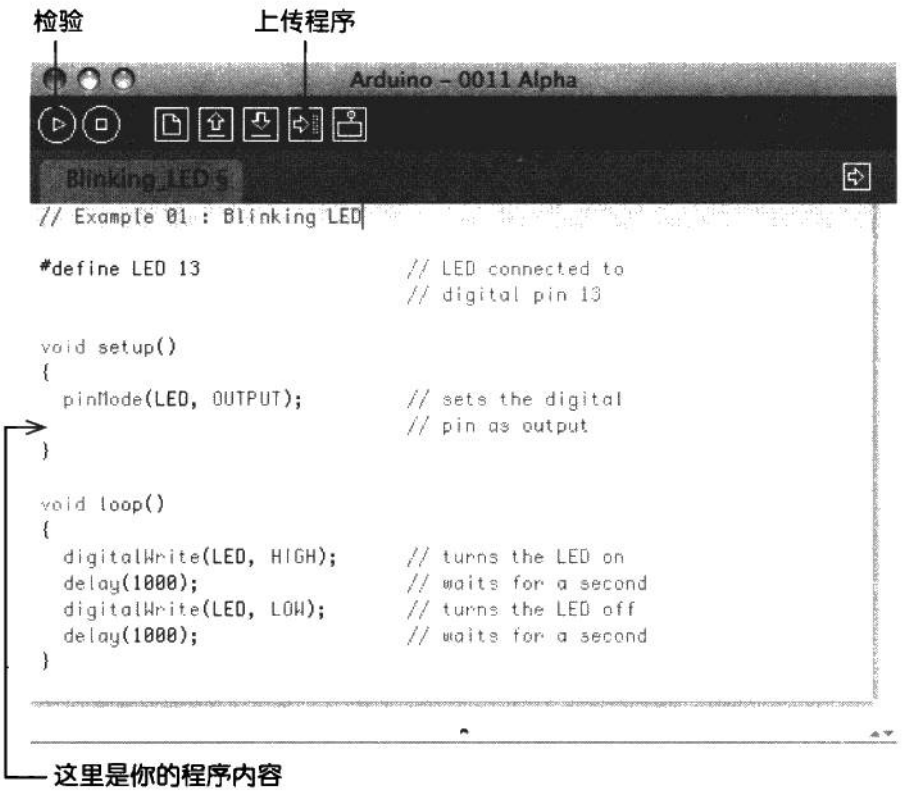


图4-3：在Arduin IDE上加载你的第一个草稿文件

输入完全部代码之后，点击检验Verify（位置参考4-3）；如果检验结果正确无误，你可以看到下方显示“Done Compiling”。该信息表示Arduino IDE已经将你的程序编译成可在芯片上执行的程序了，如同扩展名.exe 的执行程序可以在Windows中运行；或者是.app的执行程序可在MAC计算机中运行。

完成以上的步骤之后，你就可以把它从计算机中上传到Arduino芯片中。点击Upload上传程序之后程序会终止Arduino现有的处理程序，当芯片处于重置（Reset）状态之后，Arduino会通过USB来接受并且储存来自计算机的程序。

在Arduino IDE下方的黑色区域中，你会看到“Done uploading”，表示整个上传过程都已经完成。在Arduino上会有两颗LED灯显示传送（TX）和接收（RX）状态，每传送1字节（byte），该灯就会闪烁一次。在整个上传过程中，该LED会持续呈现闪烁状态。

如果传送、接收LED指示灯没有闪烁，或者在黑色区域中显示错误信息，有可能是计算机与板子之间的通信有问题。此时请确认你在Tool/Serial中是否选择了正确的串口编号（见第3章），同时也检查Tool/Board中有没有选择正确的Arduino型号。如果仍有连接上的问题，请参阅第7章。

当完成上传之后，芯片接收到“重置”（reset）命令或者电源被重新连接的时候，程序就会重新启动。所有程序代码都会被储存在芯片之中，就算切断电源也不会消失。

如果上传顺利完成，你应该会看见Arduino板子上印有“L”字样的LED灯重复亮起数秒后再熄灭。如果你按照图4-2自行安装LED灯，它也会呈现相同的效果。

你刚写的就是一个“计算机程序”，在Arduino IDE中称为sketch。前面说过，Arduino就像是一台小型计算机，我们可以利用程序命令它为我们做想做的事。程序指令在Arduino IDE中输入、编写，它会替我们将程序转换成Arduino芯片可以执行的机器代码。

之后将会告诉你如何理解整个程序代码。首先，整个程序执行顺序是从上到下，第一行是程序的开始，就如同quicktime、windows media player一样，电影时间轴是从最左边开始的，按照电影的播放进度向右移动。

给我个奶酪（Parmesan）

在计算机语言中我们必须逐行定义好每一个动作，这样计算机才可以完成执行动作。例如，在晚餐餐桌前，你对桌旁的一个人说：“请把Parmesan奶酪递给我”，接下来我们会很自然地做出一连串动作完成上述指令。但是计算机没有那么聪明，我们必须将所有需要执行的小动作都明白地告诉它。为了方便组织一整组指令，我们在写程序的时候使用左右大括号“{”和“}”，将一组动作（几行代码）包起来。

在我们的一个程序中，有两段是由“{”和“}”包起来的，而其前方会有一个命令用来表示这段程序的名称，例如：

```
void setup()
```

其中setup就是这组动作的名称，例如你想要让Arduino芯片执行拿Parmesan奶酪这个命令，你应该在程序的开头输入“*void passTheParmesan()*”。然后你就可以在程序的其他地方调用这个名为*passTheParmesan()*的代码。像这样的一块代码，我们称之为函数。它的开始执行点在左括号之后，一直执行到右括号结束。

Arduino从不停止

Arduino程序包含两个函数——*setup()*和*loop()*。

Setup()函数中所写的程序只会在你程序的最开始执行一次，而loop()函数中的代码会一遍又一遍地重复执行。Arduino程序不允许同时执行多个函数，也没有退出或关闭程序的功能。整个程序的开始和停止取决于Arduino芯片的电源开启和关闭。

真正的Tinker都写注释

带有“//”开头的代码都会被Arduino忽略。注释是用来在程序中加入解释和说明文字的，这个对于想要了解程序代码的人非常有帮助。

注释的文字穿插在代码之中，来帮助你了解代码的用途。

代码，一步一步来

起初，你可能认为这是不必要的解释，有点像我当时在学校要研究但丁的《神曲》时遇到的情况——对于每一行诗句，都跟着上百行的评论！然而，只有你开始写你自己的程序时，才会发现这种解释确实很有帮助。

```
// Example 01 : Blinking LED
```

当你有成百上千的程序时，开头的小注释对于我们分清程序的功能非常有作用。

```
#define LED 13    // LED 灯连接到电路板上
                  // 数字输出第13个引脚
```

在程序语言中使用#define功能，可以将一个数值定义成常数。在这里它的用处

是：此后的代码可以用LED这个名称来取代13这个整数，这样做可以提高程序的易读性。

```
void setup()
```

这个内容告诉Arduino有一个动作称为*setup*。在左大括号“{”之后，我们开始定义“初始化”功能（即setup动作），并且在右大括号“}”之前完成这个定义。

```
pinMode(LED, OUTPUT);      // 设置LED引脚模式为输出
```

终于，这个有趣的指令来了。*pinMode*(LED，OUTPUT)函数告诉Arduino如何运用指定的引脚（INPUT代表输入，OUTPUT代表输出）。在这个范例中，我们要通过引脚来控制LED灯，所以我们将实际上代表着13的LED放在这里。pinMode是一个函数，而它后面括号里的两项数值称为该函数的参数。INPUT和OUTPUT在Arduino语言中是一个预定义的常数（constants），意思是：程序中任何一个地方都不可以定义与它相同的名字，而且该常数所代表的数值永远不会改变。

```
}
```

这是函数的结束符号，代表着*setup()*整个函数的结束。

```
void loop()
{
```

*loop()*用来定义整个芯片主要重复动作的区域，芯片会一直重复执行该区域里的内容，直到断电为止。

```
digitalWrite(LED, HIGH);   // 开启LED
```

就如注释所解释的，*digitalWrite()*函数可以配置OUTPUT的引脚为“开启”状态。第一个参数（这里指LED）用来指定所要调整的引脚（在这个例子中就是Arduino板上数字为13的引脚），第2个参数则是指定要将输出状态改变成“开”（HIGH）或者“关”（LOW）。

你可以将板子上每一个引脚都想象成一个迷你的电源插座，如同你的公寓中的每一个电源插座一样，欧洲电压为220V，而Arduino的插座则是5V。奇妙之处在于，我们可以用软件轻易地控制电源的开启/关闭状态。程序代码*digitalWrite(LED,HIGH)*控制引脚输出电压为5V，如果该输出引脚连接LED，这个LED灯就会亮起。所以使用软件不再为着只能将结果呈现在计算机屏幕上，而是可以影响真实世界中硬件

的动作了。开启和关闭LED灯的显示，对人而言是一个更深切的感受，而LED就如同之前提到的驱动传动装置。

```
delay(1000); // 等待1秒
```

Arduino是一个非常基本的结构。因此，它可以有规律地重复某一个动作，而函数delay()就如同叫芯片静静地坐在原地什么也不做。函数中的参数是以毫秒为单位的，每1000毫秒等于1秒。所以这一行代码可以使LED灯保持在开启状态并且持续1秒钟。

```
digitalWrite(LED, LOW); // 关闭LED灯
```

同之前的一样，这个语句将把LED变为“关闭”状态。为什么Arduino要使用HIGHT和LOW两个词作为关键字？就是因为在电子电路领域里，HIGH表示引脚通电，对Arduino来说就是设定输出5V，而LOW代表0V。因此你也可以认为这两个词是ON和OFF。

```
delay(1000); // 等待1秒
```

这儿，我们再次等待1秒。这时LED灯的状态为持续关闭1秒钟。

```
}
```

来结束整个loop 函数。

总而言之，全部的程序做了下面这些事：

- 改变数字引脚13为输出（一开始就设定好了）
- 进入loop（循环程序）
- 将连接第13个引脚的LED灯设为开启
- 保持开启状态1秒
- 将连接第13个引脚的LED灯设为关闭
- 保持关闭状态1秒
- 再度回到loop的第一行

希望到这里为止，对你来说不算读得太痛苦。在之后的范例中，你会学到更多编写代码的技巧。

在进入下一章节之前，我希望你可以修改一下代码中的参数。例如，将等待的时间设置得更长一些，让LED灯闪烁的频率不一样。如果你将delay()函数的等待时间设置得非常短，你会见到一些特殊情况。这个特殊情况常被使用，我们称为PWM模式，这个模式会在后面的章节学到。

我们将会做什么？

我对于可以使用并改变控制灯光的技术和方式非常着迷。我有幸曾参与一些灯光与人互动的设计。这些互动装置的呈现，可以轻易用Arduino来完成。继续读下去，你将会学到更多关于如何设计并且完成一个“互动台灯”装置的知识。

下面，我将会尝试解释一些基本的电子电路原理，这些对工程师来说可能简单到了很无聊的地步，所以也不会吓倒初学者，这些原理对于设计师或者艺术工作者却是非常重要。

什么是电？

如果有自行更换水管的经验，那么理解电的原理就不会是太难的事情。要了解电的原理和电路板的工作方式，水车的工作模式是最适合的范例。图4-4是一个基本的由电池驱动的风扇的示意图。

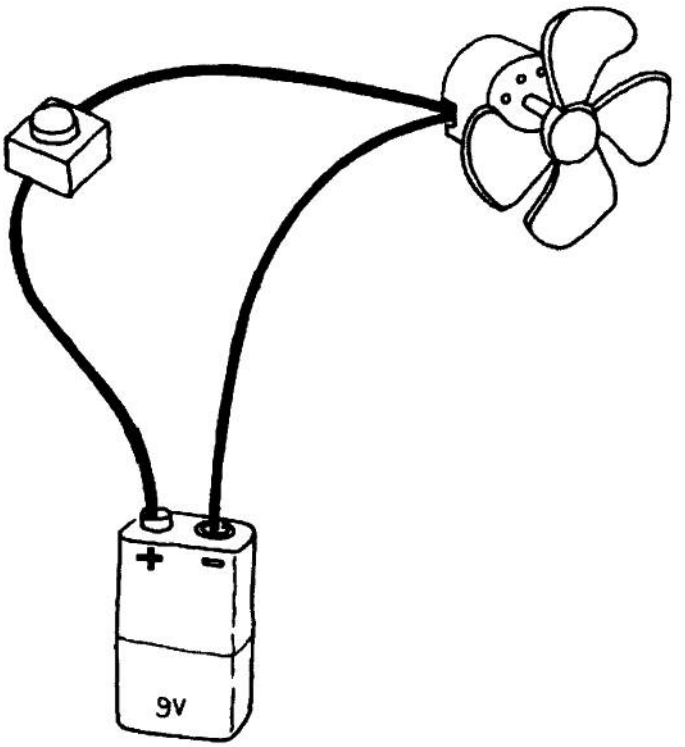


图4-4：一个便携风扇

你可以看到，其中的元器件组成包括一个电池、一组电线，还有直流电机。其中一条连接电机的电线，中间连接了一个按钮开关。如果电池电力充足，并且打开开关，那么直流电机将会开始旋转。它是怎么工作的？我们可以将整个装置想象成是一套水力运作装置，电池就像是水泵，按钮开关就是水龙头，直流电机就是水车。当你旋转打开水龙头时，就如同打开了按钮开关，水顺着水泵的压力被传输到水管之中，就会冲击水车，并使水车开始旋转。

如图4-5所示的是一个水力运作系统的装置示意图，其中两个重要的关键点是水压以及水量。你会发现，如果想要让水车的旋转速度加快，就要增加水管的尺寸和水泵的压力。增加水管的直径将有助于水的流通，也降低了水管对水的阻力。增加到一定程度之后，轮子的速度已经无法再增加。这时候就须要增加水泵的压力，直到水轮的轴承可以承受的上限为止。另外，轴承也会因为高速旋转而增加摩擦，从而使温度提高，产生热量。由此也可以理解一个重要的观念：在这样的系统中，我们提供的能量不会全部转化成动力，不可避免地会有一些能量的损耗，这些损耗将在系统中不同的部位以热的方式发散掉。

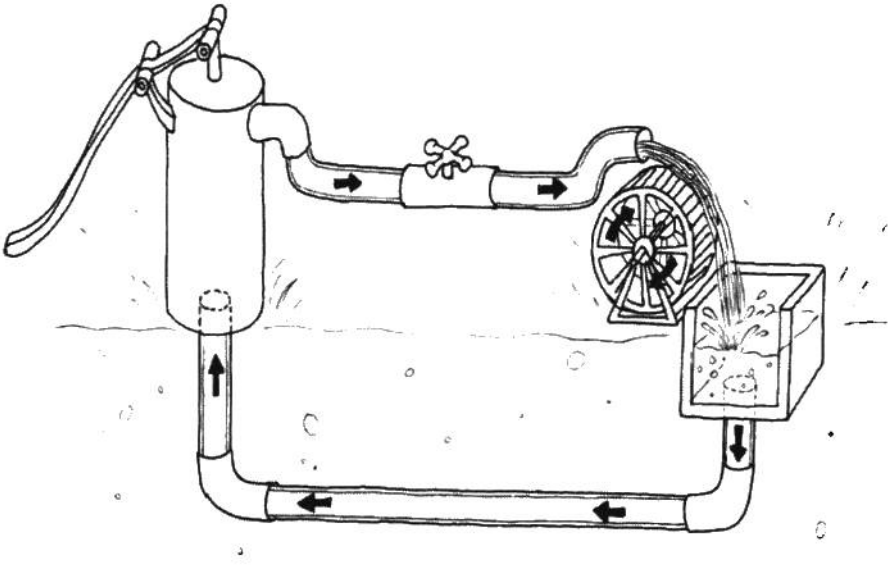


图4-5：液压系统

什么是运作水力系统时不可缺少的一部分？水泵产生的压力是其中之一，此外还有水管的阻力、水车转动所需的动力。电的现象就如同这套水力系统，水泵就如同电池或电源插座，水管就像电线一般，生活中很多电器的运作方式都是这样的。

所以当你看到电池电压为9V时，你可以想象电压如同水泵所提供的水压。电压单位是伏特，简写就是“V”，其名称来源于电池的发明者Alessandro Volta。

意识到水压如同电压之后，还有另外一个现象就是水的流速。水流的速度在电子

学中称为电流，单位为A（安培），单位命名是根据电磁学领域中的研究者André-Marie Ampère而来的。电压与电流的关系可以用水车的例子来理解，高电压（水压）可以让水车转动更快，高电流（水流）可以转动更大尺寸的水车。

最后，阻力是水车系统中需要考虑的因素之一。系统中的阻力就如同电力系统中的电阻，单位是Ω（欧姆，德国的物理学家Georg Ohm）。在电阻公式中可以看见电阻与电压、电流之间的关系。熟悉这个公式可以了解电力系统中所需的电流，从而可以提供正确的电压。

这个就非常直观了。如果在9V的电路中，当电路里的电阻R增加的时候，会造成电流下降。要说明此情况，我们再回到水力系统中，负载R的意义就如同我们在水管中安装一道阀门，可以用来控制水的流量（如同变阻器一般），使得水管中的水量减少，三者之间的关系可用如下数学公式来表示：

```
R (resistance) = V (voltage) / I (current)
V = R x I
I = V / R
```

这个是你唯一要熟记的公式，因为在以后的很多电路设计中，它都是你唯一要用到的电学公式。

使用按钮控制LED灯

在之前的范例中，我们学到了如何控制LED灯闪烁。整个互动装置是由控制器、驱动传动器和传感器构成的：Arduino是我们的控制器，LED灯是我们的驱动器，到目前为止我们还少一个传感器。当你在看书的时候，一定不希望台灯持续闪烁吧，所以接着我们要学习使用传感器来稳定控制灯光。

在这个范例中，我们使用最简单的传感器——按钮。

如果你将按钮元器件解体，会看到它是由两个金属节点和一个塑胶弹片构成的。它借塑胶弹片绝缘的特性，对金属进行阻隔与连接，达到开关的效果（类似水力系统中的止水阀门）；按下按钮时就完成开启的效果。

为了取得按钮的状态，必须使用一个新的Arduino程序指令：***digitalRead()***。

digitalRead()函数会检查括号中指定的引脚的电压状态，并传回HIGH或LOW。到目前为止，我们只在程序中命令Arduino去执行指令，还未用到会传回资料的指

令。但只使用这样的函数会使程序的功能受限，因为这样就只能照着固定剧本上演一出戏，无法使用输入元器件感测外在环境来为程序带入一些变化。使用*digitalRead()*函数，我们可以向Arduino“提问”并接受Arduino回应的答案。这个回应的答案，可以储存在记忆体中，以供立即使用或者稍后再使用。

请参考如图4-6所示的线路，你可能要去电子材料市场购买下列电子零件（这些零件在其他方案中也会经常用到）：

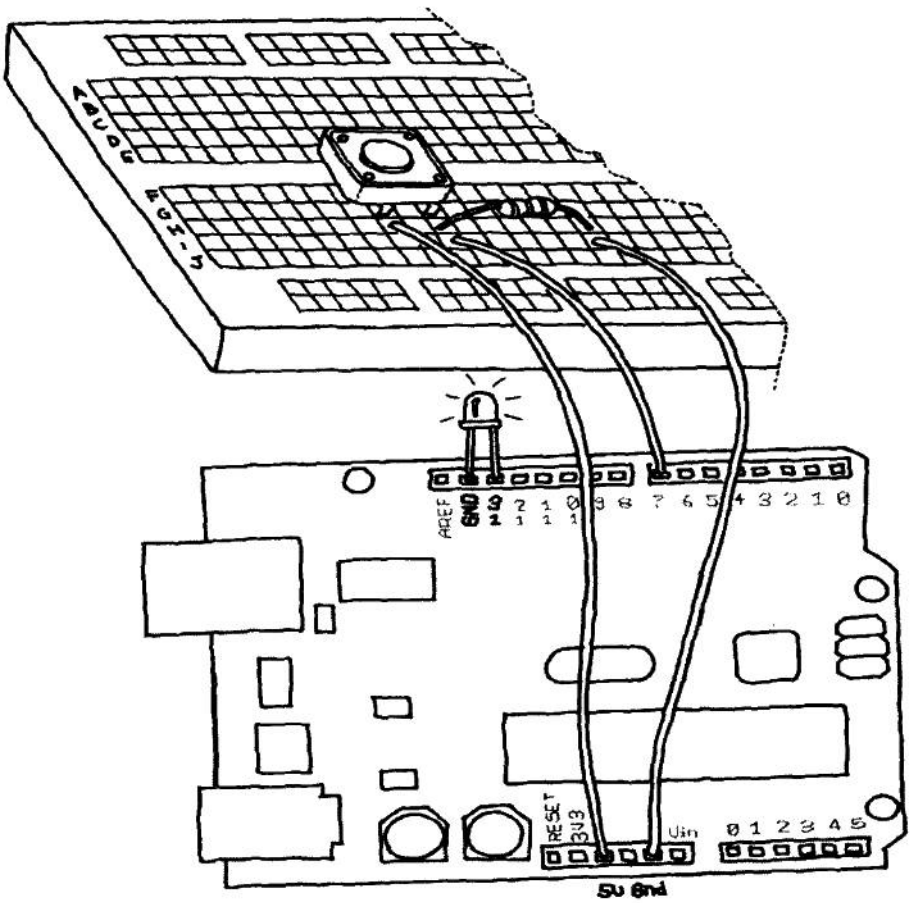


图4-6：连接上按钮控件

» 面包板：RadioShack（www.radioshack.com）编号276-002，Marker Shed（www.makershed.com）编号MKKN3。附录A是面包板的一些简介。

» 连接线：RadioShack编号276-173，Maker Shed 编号MKKN4。

» 10kΩ电阻：RadioShack编号271-1335（5个装），SparkFun（www.sparkfun.com）编号COM-08374。

» 按钮：SparkFun编号COM-00097。

注意：你也可以用剥线钳和切线器把买来的22号AWG实心线处理好，来代替那些预切跳线。

让我们看一下用按钮来控制LED的代码：

```
// Example 02: 用按钮来控制LED灯

#define LED 13 // 定义LED引脚为13
#define BUTTON 7 // 定义按钮开关引脚为7
int val = 0; // 变量val用来储存按钮状态

void setup() {
   pinMode(LED, OUTPUT); // 设定LED引脚为输出状态
   pinMode(BUTTON, INPUT); // 设定按钮引脚为输入状态
}

void loop(){
   val = digitalRead(BUTTON); // 读取按钮状态并储存

   // 当按钮按下时，设定val为HIGH
   if (val == HIGH) {
      digitalWrite(LED, HIGH); // 开启LED灯
   } else {
      digitalWrite(LED, LOW); // 关闭LED灯
   }
}
```

在Arduino IDE 中执行 File/New 命令（如果已经打开了其他程序，可以先储存后再打开新文档），输入程序名称PushButtonControl。照着程序4-2输入代码（或者可以从**www.makezine.com/getstartedarduino**下载Example02），如果一切都正确，LED灯会在你按下按钮后亮起。

它是如何工作的？

在这个范例中有必要介绍两个新的程序概念，分别是回传结果状态的函数和if声明控制。

if声明是程序设计中最重要的指令，因为它让计算机（请记得Arduino也是一台微型计算机）可以做决策判断。在if关键字之后接着在括号中写下“问题”，整个if函数会对问题做出是（TRUE）或者否（FALSE）的回应，如果答案是TURE，那么程序的一个大括号区块就会被执行，否则就执行else大括号区块中的代码。在程序中使用==（两个等号）用来判断符号两边的数值是否相同，并传回TRUE或者FALSE；而如果只使用一个等号，则意味着将右边变量的内容指定给左边的变量。请确认你使用了正确的运算符，因为这是一个很常见的错误，即使像我这样有着25年编程经验的人有时都难免会犯这种错误。

当我们需要灯光的时候，如果必须持续按着按钮，那将是一个很不切实际的动作。所以我们要找出一个方法，在程序中让它自己“持续按下”。

一个电路，一千种用法

以下将示范可编程芯片相对于传统电子零件的优势：只要通过修改程序，就能让相同的电子电路有不一样的“行为”。

之前我们提到，使用灯光时必须用手指持续按着按钮是很不切实际的做法。所以，当我们按下按钮再放开之后，程序必须“记住”我们刚才“按下”的行为，这会让程序能在我们放开按钮之后，仍然使LED灯保持亮起的状态。

为了达到这个目的，我们必须使用变量（之前的范例中已经用过一次，但是还未解释）。变量是Arduino记忆体中的一个空间，可以用来储存资料，就像现实生活中我们用便笺来记下电话号码这类事情，并贴在计算机屏幕前或者是冰箱上提醒自己一样。在Arduino程序语言里，上述的动作也是一样地简单，给变量取一个名字，然后给予数值，就可以在此后的程序代码中存取使用了。例如：

```
int val=0;
```

关键字int表示变量名称val将储存一个整数（integer）形态的数值，并把数值0赋予val变量。

变量的数值可以在其后的程序代码中被改变，例如你可以写：

```
val=122;
```

这将会赋予变量val一个新的数值122。

注意：

你是否注意到在Arduino程序语言中，除了#define定义关键字之外，每行都会有一个结束符号“;”在行尾？这是为了让Arduino编译器知道这行代码指令结束了。除了#define之外，请记得在每一行程序结尾都使用结束符来完成输入。#define指令定义的常数，会在Arduino编译的时候将程序中所有的常数转换成定义的数值。

接下来的程序是用val变量来储存函数digitalRead();所读取到的数值，变量被指定数值之后，会在记忆体中保持数值不变，直到另一行代码将它改变为止。变量所储存的记忆体称之为RAM，读取和写入的动作非常快速，只要电源不中断，资料就会持续保存着。若切断设备电源，所有储存在RAM中的资料都会被清除。但是程序本身此时仍然储存在快速记忆体中，即使没有电也还是会被保存着——就如同手机中储存的电话号码，在关机或者更换电池后，电话资料仍旧保存着。

现在让我们用另一个变量来记录手指离开按钮时，LED灯的开启或关闭的状态。

Example03A是我们的一个尝试：

```
// Example 03A: 按下按钮放开之后保持LED灯开启


#define LED 13        // 定义LED引脚为13
#define BUTTON 7      // 定义Button引脚为7
                      // 连接上按钮
int val = 0;          // val储存按钮的状态
                      //
int state = 0; // 0表示LED关闭; 1表示打开

void setup() {
   pinMode(LED, OUTPUT); // 告诉Arduino LED是输出引脚
   pinMode(BUTTON, INPUT); // BUTTON是输入引脚
}


void loop() {
   val = digitalRead(BUTTON); // 读取输入数值并储存

   // 检查按钮状态（按钮按下）
   // 并且改变LED灯状态
   if (val == HIGH) {
      state = 1 - state;
   }


   if (state == 1) {
      digitalWrite(LED, HIGH); // 打开LED灯
```

```
    } else {
      digitalWrite(LED, LOW);
    }
}
```

你会发现灯光变化得太快以至于你都不能靠一个按钮来控制它的开关。

现在测试代码，你会发现，程序不太符合我们所需要的要求。回头检查一下程序代码之中的一段，变量state储存0或1来记忆灯的状态。当按钮被放开的时候，我们将它设置为0（LED灯关闭）。

之后，当程序要读取目前按钮状态时，如果按钮是按下的状态（val==HIGH），我们将状态由0改成1。我用一个简单的算式，让数值的变化只可能有0和1两种状态：1－0=1与1－1=0。

```
state=1-state;
```

符号=的意思是将右边的运算数值赋予左边的变量，整个运算的顺序是：先完成等号右边的所有运算，再把新算出的数值赋予左边的变量（1减去变量state数值后再指定给state变量）。

在这之后的程序，再用state变量来判断LED灯的状态，结果这也造成了LED没有达到预期状态。

得到不稳定的输出结果，是因为Arduino芯片以每秒16 000 000次的速度执行程序指令。简而言之，当你按下按钮并且放开的同时，芯片已经反复读取按钮的状态数千次，造成最后的结果达不到预期；当你想要LED灯亮起时，其反应却是熄灭，反之也是。就像一个坏掉不走的闹钟，一天也会出现两次正确的时间，但实用意义不大。我们这个有问题的程序，也偶尔会出现正常反应，但是并不稳定。

该如何来修改这个不稳定的输出结果呢？答案是：在程序中应该过滤出适合更新state变量的时机，并且在读取当下按钮状态之前，将上一个时间单位按钮状态储存到另外一个变量去。将按钮的上一个状态保存下来，有助于我们过滤出由开转至关的情况。

Example 03B：

```
// Example 03B: 按下按钮放开之后保持LED灯开启
// 现在是新的升级版本!


#define LED 13        //定义LED引脚为13
#define BUTTON 7      //定义Button引脚为7
                      // 连接上按钮
int val = 0;          // val 储存按钮的状态
int old_val = 0;      // 暂存val变量的上一个时间单位

int state = 0;        // 0 表示LED关闭; 1表示打开

void setup() {
   pinMode(LED, OUTPUT);       // 告诉Arduino LED是输出引脚
   pinMode(BUTTON, INPUT);     // BUTTON是输入引脚
}
void loop(){
   val = digitalRead(BUTTON); // 读取输入数值并且储存


   //检查按钮的变化情况
   if ((val == HIGH) && (old_val == LOW)){
      state = 1 - state;
   }

   old_val = val;              // val 现在已经是旧的了，让我们暂存一下

   if (state == 1) {
      digitalWrite(LED, HIGH); // 打开LED灯
   } else {
      digitalWrite(LED, LOW);
   }
}
```

测试之后发现已经接近正确的预期，但是仍然不够完美！这种瑕疵是按钮的机械构造所造成的。按钮由两片金属的接触实现电路的导通，这听起来很简单，但在现实中，金属之间的连接并不密合，尤其是当按钮尚未按到底的瞬间和按钮弹回的时候，会产生杂波（bouncing）。

当杂波产生的时候，Arduino会接受到一连串开关交错的信号。要克服这个杂波，有几种不同的技巧。对我们这个简单的范例而言，只要在程序中加入10～50毫秒的延迟指令，就可以改善这个情况。

Exaple 03C 是最终版本的代码：

```
// Example 03C: 按下按钮放开之后保持LED灯开启
// 简单的抗杂波
// 现在是另外一个新的升级版本!


#define LED 13          //定义LED引脚为13
#define BUTTON 7        //定义Button引脚为7
                        // 连接上按钮
int val = 0;            // val 储存按钮的状态
int old_val = 0;        // 暂存val变量的上一个时间单位

int state = 0;          // 0 表示LED关闭; 1表示打开

void setup() {
   pinMode(LED, OUTPUT); // 告诉Arduino LED是输出引脚
   pinMode(BUTTON, INPUT); // BUTTON是输入引脚
}

void loop(){
   val = digitalRead(BUTTON);     // 读取输入数值并且储存
                                  //检查按钮的变化情况

   if ((val == HIGH) && (old_val == LOW)){
      state = 1 - state;
      delay(10);
   }

   old_val = val; // val 现在已经是旧的了，让我们暂存一下

   if (state == 1) {
      digitalWrite(LED, HIGH); // 打开LED灯
   } else {
      digitalWrite(LED, LOW);
   }
}
```

5/ 高级的输入输出控制方法

你在第4章刚刚学到的内容是Arduino的最基础的操作方式：控制数字输出与数字读取输入。如果把Arduino比作一串由七个字母组成的人类语言，那么你已经学会前两个字母啦。还剩下五个字母需要思考呢，在完全掌握Arduino之前，你可以看到我们还有很多工作要做。

尝试其他开关类型传感器

现在你已经学会了如何使用一个按钮，那么你还应该知道，按照同样原理工作的还有许多非常基础的传感器，如：

开关

很像一个按钮，但是当其被释放时不会自动改变状态。

温度传感器

当温度达到预设值时，开关打开。

磁力开关 （最常用的称“干簧管”）

它具有两个受到磁铁吸引就能闭合的金属片，通常用于能检测窗户什么时候被打开的防盗报警器中。另外，常见的继电器，工作原理也与之类似。

压力开关

你可以在家中的地毯或者门口擦鞋垫下面放置一个小的弹簧金属片，可用于侦测是否有人（或是一只肥猫）存在。

倾角开关

它是一个简单的电子元器件，其内部包含两个触点和一个金属小球（或者一滴导电的水银，但是我不推荐使用这种，因为有毒而且不环保）。倾角开关也称作倾角传感器，图5-1展示了倾角传感器的内部构造，当传感器到达垂直位置时，金属小球接触到两个桥接点使之导通，就好像你按下了按钮。当你将传感器倾斜时，金属小球移动离开桥接点后，桥接点就会断开，这一过程就相当于你松开按钮。使用这个简单的元器件，你可以轻松侦测到物体是否被晃动、移动或倾斜。

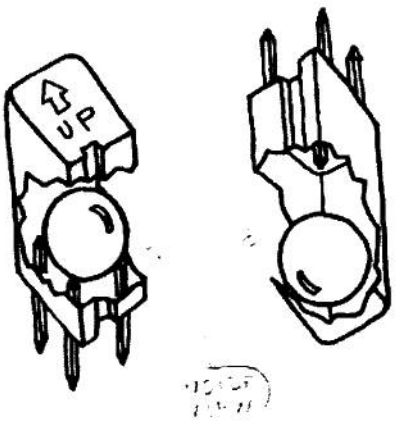


图5-1：倾角传感器的内部构造示意图

你会很想尝试的另一款传感器是可以在防盗报警器中发现的红外传感器（也称为被动红外传感器，见图5-2）。当一个人（或者其他生物）在其附近移动时，这个小装置就会即刻触发，输出反馈信号。这是一个简单的侦测运动物体的好方法。因为它所发出的红外线是人眼无法看到的，所以常被用于防盗报警器中。

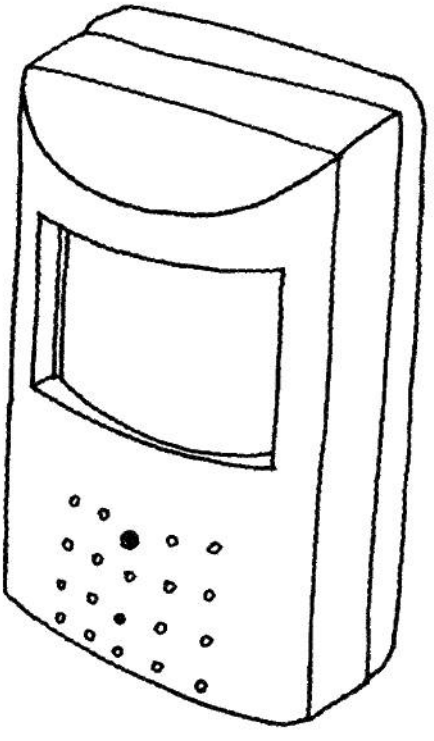


图5-2：典型的红外传感器

你应该尽可能地尝试观察身边的所有设备，如调节房间温度的空调恒温器（当然要玩旧的，而且是没有连接使用的）。例如，通过使用第4章的最后一个实例并结合红外传感器，你就可以设定当人在场时电灯自动点亮。或者你还可以使用倾角开关，设定当它倾斜到某一角度时，电灯或装置就自动关闭。

使用PWM方式控制灯光亮度

到目前为止，你已经学会了很多知识，你可以尝试自己制作一个互动灯，而它不仅仅是用单调的开关控制灯的亮灭，虽然也许在某种程度上，那样看上去效果更优雅一点。我们现在已经完成的灯光控制仅局限于亮或灭，而一个奇特的互动灯是要能调节亮度的。为了解决这个问题，我们可以使用一个在电视或电影中经常用到的小技巧：视觉暂留。

在第4章第一个例子中，我们使用延时函数控制灯光闪烁间隔时间。现在如果你改变延时函数中的数值，直到你不能看出LED点亮与熄灭动作的时间间隔为止，你就会发现LED的亮度似乎是正常情况的50%。现在继续改变数值，使点亮的维持时间设置为原本的1/4，你将会看到LED亮度大约只有正常情况的25%。这项技术被称为脉冲宽度调制（PWM），简单地说，LED闪烁时，如果你让它闪得足够快，你就不会察觉到它在闪烁，因此你可以通过改变亮灭时间比让LED看上去产生不同亮度。图5-3所示是其工作原理。

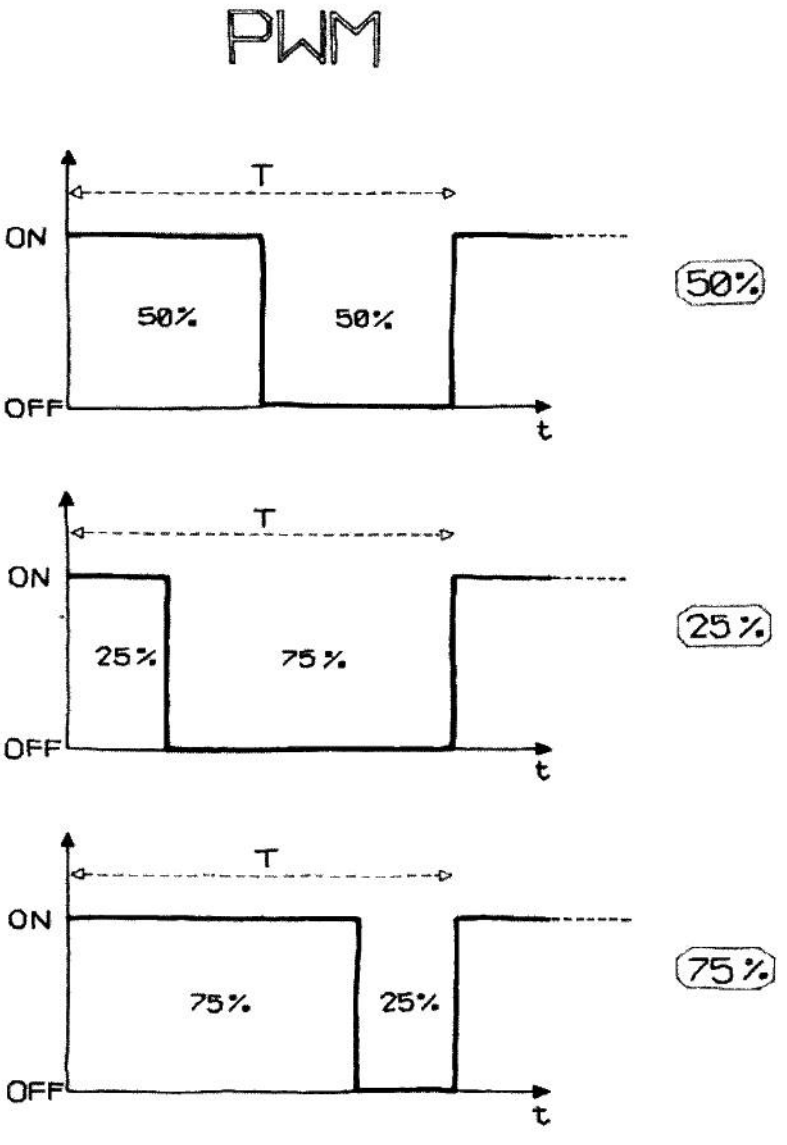


图5-3：PWM示意图

这项技术不仅用于LED，还可以用于其他装置。例如，你可以用同样的方式改变电机的转速。

通过实验，你将会发现在你的程序中使用延时函数点亮LED并使之闪烁有点不方便，因为只要你想读取一个传感器数据或者通过串口发送数据，LED就将会闪烁，同时它将等待着你完成传感器数据读取。幸亏Arduino电路板上使用的处理器可以

在处理别的事情的同时非常有效地点亮3个LED使其闪烁，这部分PWM输出接口是9、10与11，这三个PWM输出接口可以使用analogwrite()函数指令控制。

例如：**analogwrite(9，128)**指令的意思是使连接到第9接口的LED灯以50%的亮度点亮。为什么第二个参数是128呢？原来，**analogWrite()**指令中PWM参数的可设范围是从0～255：255表示最亮，0表示最暗（相当于OFF状态）。所以参数设定128刚好就代表一半的亮度输出。

注意：

拥有这三个PWM的输出接口非常好，因为如果你买来红、绿和蓝色LED，就可以组合它们的灯光，混合调试出你喜欢的各种各样的颜色！

让我们来试试吧。按照图5-4所示在面包板上搭建电路，注意LED是有极性的：长脚（正极）在右侧，短脚（负极）在左侧。连接电阻（470Ω～1kΩ）接地（GND）。此外要仔细观察LED发光灯的塑料部分，通常负极一端会削成直边。

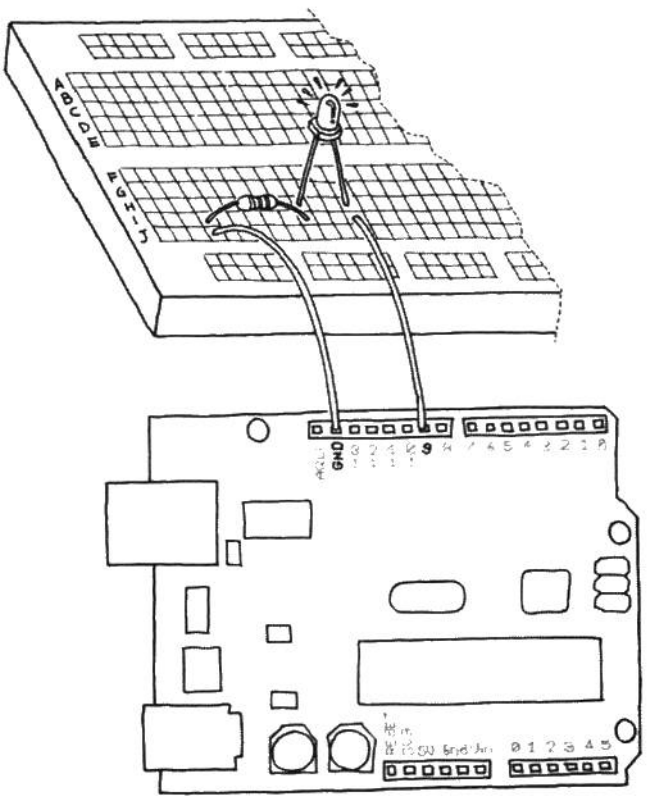


图5-4：LED连接到PWM输出接口

然后，创建一个新的程序编辑文件，参阅例子04的程序（你也可以从www.makezine.com/getstartedarduino下载程序源代码）：

```
// 例子 04: 逐渐改变亮度的LED灯
// 复制并粘贴这个例子程序到新创建的Arduino编辑文件中

#define LED   9       // 定义LED灯引脚
int i = 0;            // 存储最新数值变数，我们将使用它来从上到下计数

void setup() {
  pinMode(LED, OUTPUT); //设定Arduino LED引脚为输出
}

void loop(){

  for (i = 0; i < 255; i++) { // 变数从0 到254之间变化（渐强）
    analogWrite(LED, i);      // 设定LED亮度

    delay(10);          // 要延时10ms（毫秒），因为analogWrite执行瞬间完成
                        // 速度太快眼睛不易察觉
                        //
  }

  for (i = 255; i > 0; i--) { // 变数从255 到1之间变化（渐弱）

    analogWrite(LED, i);      // 设定LED亮度
    delay(10);                // 延时10ms
  }

}
```

现在你已经通过笔记本计算机制作了一个具有奇特功能的灯（光渐亮后渐暗，用Arduino做这种简单的应用，有些浪费资源），让我们运用这项技术使我们的灯光变得更加完美吧！

在新设计增加一个电路之前，让我们回看一个在面包板上的按钮电路的范例（回看第4章），看看你能不能不看下一页就可以完成这部分的设计。之所以这样做，是因为我想让你开始思考我在这里展示的每个基本电路的用途与扩展性，本书中的小型电路就好像一块块“积木”，更好地利用这些“积木”，你就可以完成更大的项目。如果你想要提前偷看，也没有关系，最重要的事情是你需要花一些时间思考它是如何实现的。

用按钮控制LED灯亮度

要创建这个电路用按钮来控制LED灯的亮度，你须要将前一章图4-6所示的按钮电路与刚才建立的图5-4所示的LED灯电路结合起来。如果你喜欢，你可以在面包板

上不同的位置分别建立这两部分电路，当然前提是面包板上要有足够空间。每次在面包板上建立的电路可以暂时保留，供以后参考，不用将电子元器件全部取下。面包板有一个优点（见附录A）就是有一对贯穿底部与顶部电路插孔的导轨，一根用红色（正极）标记，另一根用蓝色或者黑色（负极）标记。

这些导轨通常是分隔我们需要的电源正极与负极的，在你要设计搭接这个例子的电路时，你有两个元器件（其中两个是电阻）须要连接到Arduino的GND引脚上，因为Arduino有两个GND引脚，所以你可以很简单地直接将它们分别接到这两个引脚中；另一种方法则是将电阻都接到面包板上的地线共通插孔，再用一条导线连接地线共通插孔和Arduino的GND引脚。

如果你还没有准备好尝试改变电路路线，也不要担心：把按照图4-6和图5-4所示搭建的电路都连接到Arduino电路板即可。稍后你会在后面第6章看到使用面包板共通插接的例子。

再回到下面我们要做的例子，如果我们只有一个按钮，如何控制灯的亮度呢？我们要学习另一个交互设计的技术——检测按钮被按下的时间。要做到这一点，我须要在第4章例子03的程序基础上做改进，增加调光功能。这个功能要建立在同一个“接口”上，即按下或松开的动作，控制灯的开启或关闭；而按下持续不放的动作，能控制调节灯光亮度。

邮
电

让我们来看下面的例子：

```
// 例子05：按下松开按钮开关LED灯，按住不放调节LED灯光亮度
// 复制并粘贴这个例子程序到新创建的Arduino编辑文件中

#define LED 9          //定义LED灯引脚
#define BUTTON 7       //定义检测按钮引脚

int val = 0;           /存储输入引脚状态

int old_val = 0;       //存储val上一状态数值
int state = 0;         //当state=0时关闭LED灯，state=1时开启LED灯

int brightness = 128;          //存储亮度数值
unsigned long startTime = 0;   //按下按钮的开始时间

void setup() {
  pinMode(LED, OUTPUT);        //设定LED引脚为输出接口
  pinMode(BUTTON, INPUT);      //设定BUTTON引脚为输入接口
}

void loop() {
```

```
  val = digitalRead(BUTTON);  // 读取按钮状态并存储至val

  // 检测按钮状态变化过程（设定LED灯状态）
  if ((val == HIGH) && (old_val == LOW)) {

    state = 1 - state;          //如果按钮从松开转变为按下状态
                                //那么改变state数值

    startTime = millis(); //暂存最后一次按钮按下的开始时间（毫秒）

    delay(10);
  }

  //按住按钮并保持不放（调节亮度）
  if ((val == HIGH) && (old_val == HIGH)) {

    //如果按住按钮持续时间超过0.5秒（500毫秒）
    if (state == 1 && (millis() - startTime) > 500) {

      brightness++;  //亮度变量逐渐+1
      delay(10);     //延时10毫秒避免增加速度过快


      if (brightness > 255) { //255是亮度变量最大值（PWM输出最大值）

        brightness = 0;         //如果亮度变量增加后结果大于255
                                //重新设定成0，灯光亮度由暗到亮
      }
    }
  }

  old_val = val;      // 存储当前val变量状态，用来判断按钮变化状态

  if (state == 1) {
    analogWrite(LED, brightness);  //点亮LED灯并设定亮度


  } else {
    analogWrite(LED, 0);        //关闭LED灯
  }
}
```

现在赶快试试吧，你会看到我们的互动模型正在逐步完成。如果你按下按钮并立即松开它，你可以切换LED灯的打开或关闭状态。如果你按住按钮不放，此时灯光亮度会一直变化，当灯光达到你想要的亮度时再放手松开按钮即可。

现在让我们学习使用一些更有趣的传感器。

使用光线传感器取代按钮

现在我们来尝试一个更有趣的实验，取一个光线传感器（光敏电阻），如图5-5所示。

图5-5：光敏电阻（LDR）

传感器在市面上有很多种，例如温度传感器、湿度传感器、倾角传感器、颜色传感器、声音传感器等，其中可以检测灯光明暗变化的传感器称为光线传感器（光敏电阻）。

在黑暗中，光敏电阻具有很高的电阻值。当你用一点光照射它时，电阻值会迅速下降，变成一个相当不错的导体，利用这种特性可以将其当做光敏开关使用。

按照例子02搭建电路测试光敏电阻（参看第4章使用按钮控制LED灯例子，这里是用光敏电阻替换按钮），将例子02的程序代码编译下载到Arduino。

如果你用手遮住光敏电阻，你将会发现LED灯熄灭，反之，当光敏电阻受到光的照射时，LED灯会亮起。这是你到目前为止所做的第一个利用传感器控制LED的互动装置，这点非常重要，因为在这本书中，我们第一次使用了电子元器件，而不是靠简单的机械开关来做实验。它可算是一个非常有价值的传感器呀。

模拟输入

正如我们在前几个章节所学过的，可以利用digitalRead()函数指令读取Arduino某一接口是否有高低电平的变化，但是digitalRead()函数只能使Arduino检测高电平或者低电平两种状态。然而，光敏电阻不只会告诉我们现在所处的环境是亮还是暗，它还会测出环境光的明暗程度。这就是开/关式数字传感器（它只告诉我们是否有东西）和模拟传感器的区别。为了读取和测量模拟传感器的值，我们需要用到另外一个引脚（模拟输入引脚）。

在Arduino电路板的右下方，你会看到有6个标有“Analog In”的引脚，这些特别的引脚就是模拟引脚。这些引脚不仅可以测出是否有输入电压，还可以测出输入的电压值是多少。我们可以利用analogRead()函数指令来读取这些引脚的电压值，这个函数的返回值范围是0到1023，对应于0V到5V的电压值。例如，若施加一个2.5V的电压到模拟引脚0，并在程序中写入analogRead(0)函数，则读取到的返回值是512。

如果你现在开始按照图5-6搭建电路，注意其中要在光敏电阻一端串联一个10kΩ的电阻，完成后下载例程06A到Arduino电路板上，你将会看到Arduino上的LED（你也可以根据第4章“闪烁的LED”在13引脚和GND之间插入自己的LED）是根据环境光的明暗程度，以不同的频率在闪烁。

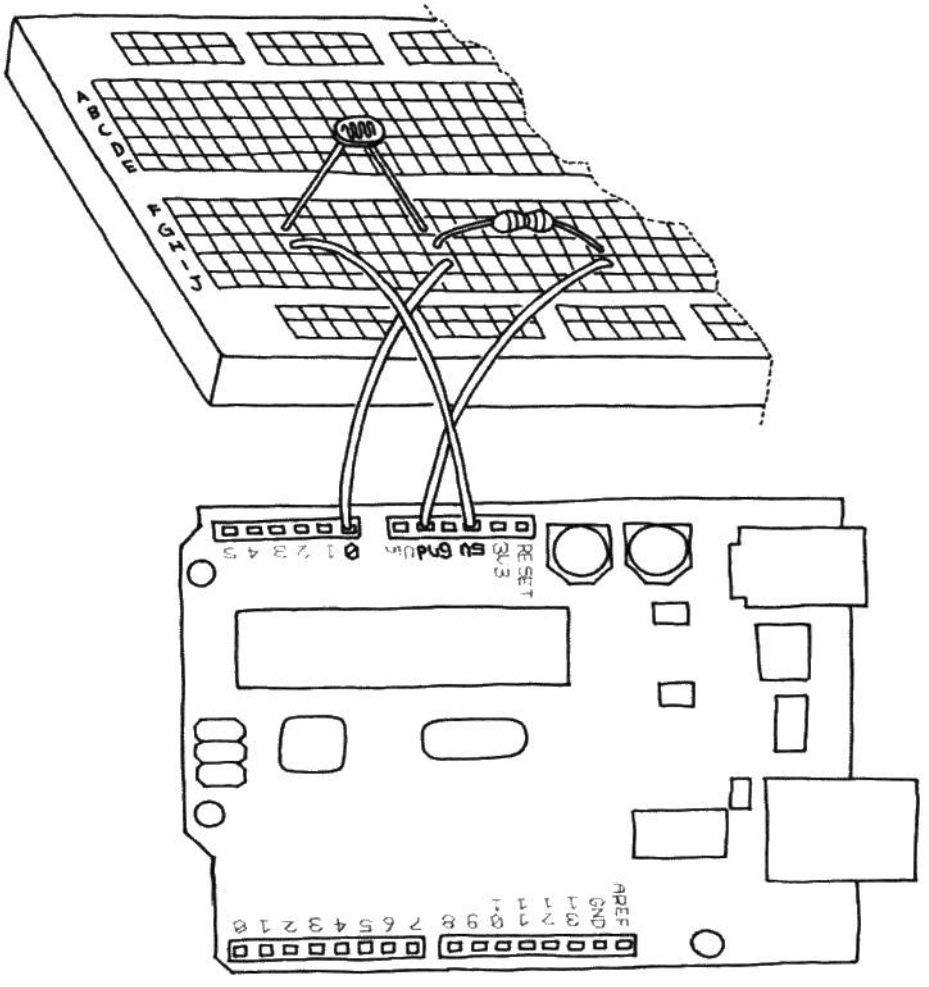


图5-6：模拟传感器电路图

```
// 例程06A:根据模拟输入值大小来设定LED闪烁频率

#define LED 13                  // 定义LED引脚

int val = 0;                    // 暂存来自传感器的变量数值
void setup() {
   pinMode(LED, OUTPUT);        // 定义LED引脚为输出接口

   // 注意：模拟接口自动设置为输入
}

void loop() {

   val = analogRead(0);         // 读取传感器的模拟值并赋值给val

   digitalWrite(13, HIGH);      // 点亮LED

   delay(val);                  // 延迟一段时间（延时时间根据传感器的模拟值决定）

   digitalWrite(13, LOW);       // 熄灭LED

   delay(val);                  // 延迟一段时间（延时时间根据传感器的模拟值决定）
}
```

下面我们尝试一下例程06B的程序，但在你实验之前，需要修改一下你的电路。请按照图5-4所示将一个LED接到数字接口9，因为面包板上已经接插了一些元器件和导线，所以在接线时要小心不要让它们遮挡住光敏电阻，以免影响效果。连接完电路以后，将例程06B的程序下载到Arduino电路板上，我们将会发现当环境光明暗程度变化时，LED的亮度也会随之变化。

```
// 例程 06B: 根据模拟输入值大小来设定LED的亮度

#define LED 9          // 定义LED的引脚

int val = 0;           //暂存来自传感器的变量数值

void setup() {

   pinMode(LED, OUTPUT);      // 定义LED引脚为输出接口

   //注意：模拟接口自动设置为输入
}

void loop() {

   val = analogRead(0);       // 读取传感器的模拟值并赋值给val
   analogWrite(LED, val/4);   // 打开LED并设置亮度（PWM输出最大值255）

   delay(10); // 延时10毫秒
}
```

注意：

这里我们将传感器返回值除以4，原因是模拟输入analogRead()函数的返回值范围是0～1023，而模拟输出analogWrite()函数的输出值范围是0～255。

尝试其他模拟传感器

我们还可以用上一节中看到的电路尝试更多其他的传感器，很多模拟传感器工作原理大同小异，基本上都是电阻式的，通过改变阻值而改变输出的电压值。例如，你可以连接一个热敏电阻（这是个简单的元器件），其电阻值随温度的变化而改变。下面我将向你展示如何运用Arduino检测传感器输出变化的电压值。

当测量热敏电阻的时候，要知道你读取到的数值并不代表实际温度值，如果你想要准确地测量，你要读取现在传感器返回的数值和真正温度计的测量值，把这些数字记录到纸上进行比对，制定一种方法来校准结果，也就是寻找二者之间的一个换算关系，以求模拟出真实世界的温度。

到目前为止，我们只学会了用LED作为输出设备，如何通过Arduino读取传感器的数值获取有实际意义的数据呢？我们总不能用LED的闪烁作为莫尔斯电码的值吧！（我们其实也可以这样做，但我们显然还是需要一个更为简单明确的数值读取方式。）为此，我们在下一节将要讨论如何通过串口让Arduino与计算机通信。

串行通信

在本书前面的部分，你所学到的只是通过Arduino与计算机之间的USB连接线将IDE（集成开发环境）中的程序下载到Arduino上，其实我们还可以通过这条USB连接线，让计算机与Arduino之间互相通信，让Arduino把数据发送回计算机，并且也能接收来自计算机的命令。要实现这个功能，我们要在程序中使用serial()函数指令（我们可以将函数理解为一系列功能的集合，让我们在编写程序时，能够方便地调用某一系列功能）。

这个函数内包含了我们发送或者接收数据所需要的所有程序代码。接下来，我们利用前面刚刚搭建的光敏电阻电路，输入下面的程序，并将其下载到Arduino中，将读取的光敏电阻的值发送回计算机上显示（你可以到**www.makezine.com/getstartedarduino**下载该程序）。

```
// 例程07：读取模拟口0的值并回传到计算机
// 确保在你下载完程序时打开串口监视窗

#define SENSOR 0      //定义传感器输入引脚

int val = 0;          // 定义变量储存传感器的返回值

void setup() {

Serial.begin(9600);   // 打开串口并设置通信波特率为9600
}

void loop() {

   val = analogRead(SENSOR);  // 读取传感器接口反馈值
   Serial.println(val);       // 串口监视窗显示传感器反馈值
   delay(100);                // 每100毫秒发送一次数据
}
```

你将程序下载到Arduino以后，如果按下Arduino IDE软件中的“Serial Monitor”按钮（上方工具栏中最右边的按钮），将会在Arduino IDE下方的子视窗看到一串数字不停地滚动出现，这些就是程序中Val的数值。现在凡是可以从串口读取数据的软件都可以与Arduino实现串行通信。Processing（**www.processing.org**）是一个非常

不错的选择，可以与Arduino很好地结合使用，因为编程语言与集成开发环境两者非常相似。

驱动较大功率负载设备（直流电机、灯泡等）

Arduino电路板每一个引脚只能为我们的设备提供20mA的电流：这个电流其实是非常小的，仅能满足驱动一个LED灯。如果你想尝试驱动类似直流电机这种较大功率的设备，该引脚将立即停止工作，并有可能烧坏整个处理器（AVR单片机芯片）。为了驱动控制像直流减速电机或者白炽电灯这样的大功率设备，我们还需要一些外加的电子元器件，这些元器件可以通过Arduino的引脚来控制启动与停止。我们可使用的电子元器件之一就是MOSFET晶体管，忽略这个奇怪的名字吧，它就是一个电子开关，有三个引脚，我们将其中的一个G引脚（栅极）连接到Arduino引脚上，并给予电压，使此元器件导通。这个元器件类似于家中的电灯开关，我们用手指开关电灯的动作现在就由Arduino来替代完成。

注意：

MOSFET 的意思是“金属氧化物半导体场效应晶体管”。这是一个以场效应为基本原理的特殊类型的晶体管，它意味着施加电压到G（栅极），电流就将经过一块半导体材料（漏极与源极之间的通道）。也就是说，通过Arduino给予G（栅极）引脚额定电压，D（漏极）与S（源极）之间就会导通，这种元器件是控制大功率负载设备的理想之选。

在图5-7中，你可以看到用IFR520晶体管控制一个小型直流电机的例子。在图5-7中可以看到电机，你能够看到用IFR520晶体管控制一个安装上风扇的小型直流电机开启与关闭的例子。你还会注意到电机是从Arduino NG（Arduino早期版本）的9V输出接口取电的。这就是使用晶体管当做开关的另一个好处：我们的大功率设备可以不与Arduino使用同一个电源，这样我们就可以用小电压控制大功率设备了。由于晶体管接到了Arduino的9号接口，我们也可以使用像analogWrite()这样的函数指令，利用PWM输出去控制电机的转速。

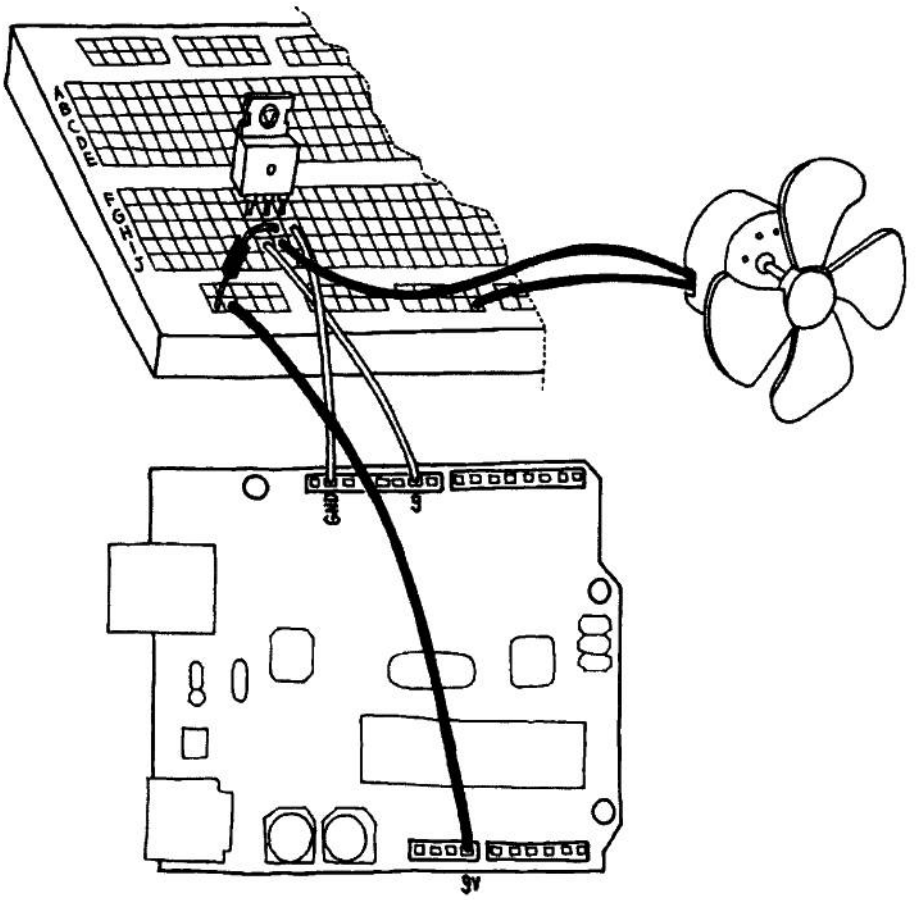


图5-7：Arduino驱动控制直流电机电路图

复杂传感器

一些传感器有较为复杂的内部结构，通常会嵌入一个小的电路或者微控制器在内部对信息进行预处理，这时就需要除了digitalRead()和analogRead()以外的其他函数。

经常会用到的复杂传感器包括超声波传感器、红外测距传感器、加速度计传感器等。你可以到Arduino官方网站“Tutorials”中找到使用它们的方法和范例（**www.arduino.cc/en/Tutorial/HomePage**）。

另外Tom Igoe的《Making Things Talk》（O'Reilly出版）一书中有更多有趣、新奇的传感器介绍。

6/互动云

在前面的章节中，我们学习到了Arduino的基础知识和基本组成模块。接下来让我们复习一下Arduino所拥有的功能，继续我们的Arduino互动之旅。

数字输出

我们可以使用数字输出功能，控制一个LED灯，确保此电路正确，还可以控制电机，制造出声音甚至操控更多设备。

模拟输出

我们可以使用模拟输出功能控制LED灯的亮度，不仅是使LED发光或者熄灭，还可让它停留在两者之间的某种亮度上。此外，我们甚至可以控制电机的转速。

数字输入

数字输入功能可以让我们获知简单传感器的状态，例如：按钮、倾角开关。

模拟输入

我们可以使用模拟输入功能读取模拟传感器的连续信号，而不只是开/关状态的信号，例如：光线传感器、电位计模块。

串行通信

这使我们能够与计算机进行沟通和数据交换，让我们能够监视正在运行的Arduino接收或者发送的数据。

在这一章中，我们将让你看到如何整合在前面章节中所学到的程序。本章要告诉你如何将每一个简单的例子整合到一起，搭建出一个复杂的工程项目。

这个作品就是我作为设计师想出来的，我们将要完成的是21世纪经典的互动灯。这个作品的灵感来源是我最喜欢的意大利设计师Joe Colombo在1964年设计的一盏叫“Aton”的灯。

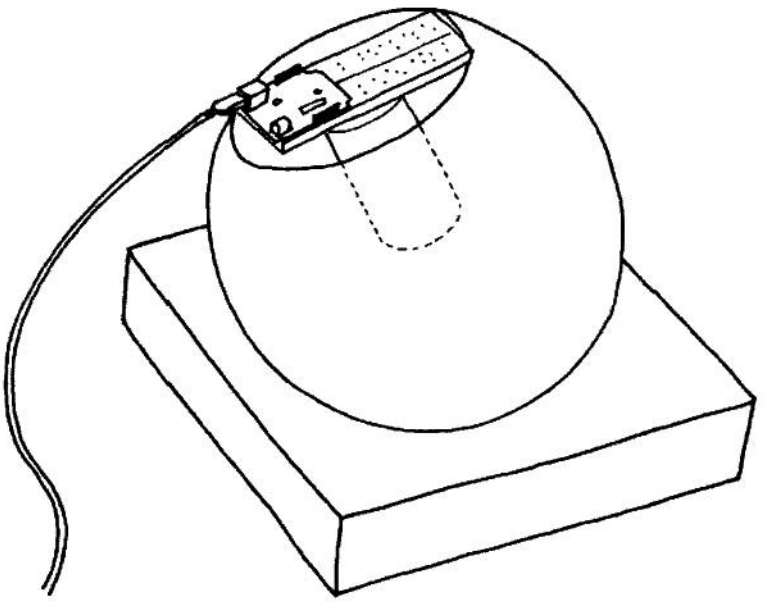


图6-1：互动灯成品图

如图6-1所示，这盏灯是将一个球形的灯具安装在带有凹槽的灯座上，凹槽可以防止球形灯具滚动，这种设计的另一个优点是我们可以依照个人喜好自由转动其方向。

在功能方面，我们要制作一个设备将其连接到互联网，彩灯将会以Make博客（**blog.makezine.com**）里“peace”、“love”、“Arduino”三个词的出现次数作为灯光色彩变化的参数。有了这些数据，我们要生成的颜色就会在互动灯上显示出来。互动灯除了自身具有一个我们可以控制的开关外，还有一个光线传感器，可以通过光感使其自动开启。

制订计划

首先，让我们想一下要实现的功能是什么，然后了解整个系统中的每一处细节。我们要将Arduino接入互联网。由于Arduino电路板上仅有一个USB接口，所以我们不能直接插入网线连接到互联网。为此，我们需要一个将两者连接到一起的装置。通常，人们会在计算机上运行一个应用程序连接到互联网，由它采集处理网络数据，再通过USB接口把指令传送到Arduino。

Arduino相当于一台内存容量比较小的计算机，无法处理大量的数据资料，当我们从网络上获取一个RSS文件时，会得到一个容量不小的XML文件，处理时将需要更多的RAM。所以我们使用计算机简化XML处理，作为Arduino与网络之间的过渡桥梁。

> **关于Processing**
>
> Arduino是由Processing软件开发出的硬件应用。我们都很喜欢Processing语言，并且利用它来教初学者编程，写出漂亮的程序代码。Processing和Arduino是完美的软硬件的组合。Processing的另一个优点是它也是开源的，在各种系统（Mac、Linux、Windows）上都可以运行。Processing也可以生成在这些平台上独立运行的应用程序。更好的是，Processing语言是全世界各地爱好者的共有资源，所以可以在互联网上找到成千上万的应用程序源代码。

我们利用Processing程序语言编写代理程序（proxy），它将处理从**makezine.com**下载的RSS文件并提取生成XML文件中的所有关键词，然后它将计算“peace”、“love”、“Arduino”三个词的数量，并且根据这三个数据产生一个代表色彩的值，再通过USB接口传给Arduino。Arduino也会回传光线传感器的反馈值并将其显示到屏幕上。

在硬件方面，我们可以结合之前学习到的按钮的例子、光线传感器的例子、通过PWM调节LED亮度（此章利用PWM控制3个LED）的例子以及串口通信的例子。

由于Arduino计算能力有限，我们需要一个简单的方式用数值来表示指定的颜色。这里直接选用颜色在HTML中的标准表示方法：用#号后的六位十六进制数字组成（六个字代表RGB三个颜色的混合程度，两个字为一组十六进制数）。十六进制数是表示数字的一种很方便的方式，因为每八位数字只需要两个字符即可表示，而如果用十进制则需要三个字符——例如：11111111（二进制）=255（十进制）=

FF（16进制）。由此可见，十六进制可以使程序代码变得更加简单。程序中读到一个“#”时，我们就知道接下来的六个字符是三原色的值，我们将这六个字符保存到缓冲区（将这个值赋给变量暂时保存），最后用每两个字符（一个字节）代表每个LED（共三个LED）的亮度。

编写程序源代码

这个系统需要两个程序代码：一个是Processing的程序代码，另一个是Arduino的程序代码。下面是6-1中的Processing程序代码，你可以从**www.makezine.com/getstartedarduino**下载。

Processing程序代码：

```
// 例子08A: 基于Arduino的网络互动灯
// 部分源代码的灵感源于todbot.com网站Tod E. Kurt的博客
// 复制并粘贴例子程序源码到Processing新建编辑文件中

import processing.serial.*;

String feed = "http://blog.makezine.com/index.xml";

int interval = 10; // 每60秒重新读取RSS文件
int lastTime; // 记录最后读取RSS文件的计数器时间

int love = 0;
int peace = 0;
int arduino = 0;

int light = 0; // 灯光亮度测量值

Serial port;
color c;
String cs;

String buffer = ""; // 暂存从Arduino发送过来的数值

PFont font;

void setup() {
    size(640,480);
    frameRate(10); // 调整更新速度，此处我们并不需要快速更新

    font = loadFont("HelveticaNeue-Bold-32.vlw");
    fill(255);
    textFont(font, 32);

    // 注意事项:
    //程序Serial.list()检索到的第一个串口，通常就是Arduino使用中的，如果不是，
```

```
    // 请删除println(Serial.list());这条语句前的“//”注释符，
    // 重新下载运行程序，再次
    // 查看串口端口列表，然后修改 Serial.list()[0]这条语句[ ]中的数字，
    // 从0开始直到你的Arduino的串口端口号。
    // println(Serial.list());
    String arduinoPort = Serial.list()[0];
    port = new Serial(this, arduinoPort, 9600); // 连接到Arduino

    lastTime = 0;
    fetchData();
}

void draw() {
    background( c );
    int n = (interval - ((millis()-lastTime)/1000));

    // 建立颜色的三个数值
    c = color(peace, love, arduino);
    cs = "#" + hex(c,6); // 准备字符串发送给Arduino

    text("Arduino Networked Lamp", 10,40);
    text("Reading feed:", 10, 100);
    text(feed, 10, 140);

    text("Next update in "+ n + " seconds",10,450);
    text("peace" ,10,200);
    text(" " + peace, 130, 200);
    rect(200,172, peace, 28);

    text("love ",10,240);
    text(" " + love, 130, 240);
    rect(200,212, love, 28);

    text("arduino ",10,280);
    text(" " + arduino, 130, 280);
    rect(200,252, arduino, 28);

    // 将表示颜色的数值显示到屏幕上
    text("sending", 10, 340);
    text(cs, 200,340);
    text("light level", 10, 380);
    rect(200, 352,light/10.23,28); // 将0～1023之间的数值变成0～100之间的数值，
                                   所以需要除以10.23

    if (n <= 0) {
      fetchData();
      lastTime = millis();
}

port.write(cs); // 发送数据给Arduino

if (port.available() > 0) { // 检测是否有等待的数据
    int inByte = port.read(); // 读取一个字节
    if (inByte != 10) { // 如果不是新一行的数据
```

```
      buffer = buffer + char(inByte); // 将它保存到缓冲区
    }
    else {

      // 新的一行数据已经准备好，我们来处理新的数据
      if (buffer.length() > 1) { // 确认有足够的数据

        // 除去最后一个字符（文本最后的字符是回车符）
        buffer = buffer.substring(0,buffer.length() -1);

        // 将字符串转换为数字
        light = int(buffer);

        // 清除缓存区数据以备下次使用
        buffer = "";

        // 我们读取Arduino的数据可能会滞后
        // 因此我们要保持读取传感器的值在最新状态
        port.clear();
      }
    }
  }

}

void fetchData() {
    // 读取和分析RSS文件
    String data;
    String chunk;

    //计数器归零
    love = 0;
    peace = 0;
    arduino = 0;
    try {
      URL url = new URL(feed); // 用于开启网址
      // 准备网络连接
      URLConnection conn = url.openConnection();
      conn.connect();          // 现在连接到网络

    // 每次读取一行数据后将其存入缓存区
    BufferedReader in = new
      BufferedReader(new InputStreamReader(conn.getInputStream()));

    // 读取每一行RSS内容
    while ((data = in.readLine()) != null) {

      StringTokenizer st =
        new StringTokenizer(data,"\"<>,.()[] ");// break it down
      while (st.hasMoreTokens()) {
        // 转换所有字符成小写
        chunk= st.nextToken().toLowerCase() ;

        if (chunk.indexOf("love") >= 0 ) // 判断是否是“love”？
```

```
        love++; // 将love变量加一
      if (chunk.indexOf("peace") >= 0)        // 判断是否是 "peace"?
        peace++; // 将peace变量加一
      if (chunk.indexOf("arduino") >= 0)      // 判断是否是 "arduino"?
        arduino++; // 将arduino变量加一
    }
  }

  // 每个关键字出现的最大值为64
  if (peace > 64) peace = 64;
  if (love > 64) love = 64;
  if (arduino > 64) arduino = 64;
    peace = peace * 4;        // 将变量乘以4，刚好是每个颜色LED需要的数值
    love = love * 4;
    arduino = arduino * 4;
  }
  catch (Exception ex) {      // 如果发现错误立即停止程序
    ex.printStackTrace();
    System.out.println("ERROR: "+ex.getMessage());
  }
}
```

在运行Processing之前，为了确保程序的正确性，我们需要确认两件事：首先要确认Processing程序所需的字体（第一次执行此程序，如果出现“Could not load font HelveticcaNeue-Bold.vlw”的错误信息，就是这部分没有选择正确），要做到这一点，我们就要创建并保存这个程序，选择Tool，找到Create Font选项，找到HelveticcaNeue-Bold字体并选择之，至于字体大小，选择32号，然后点击确定。

其次，你要确认程序中的串口端口号与Arduino的串口端口号一致。这一点在我们下载程序运行时才可以确认。大多数情况下，程序都会顺利运行。然而，如果Arduino没有任何反应，或者在计算机屏幕上没有看到任何Arduino返回光线传感器的数值。那么就请找到“重要提示”中的注释信息，并按照指示操作，核查更正程序代码。

下面是运行在Arduino的程序（同样可从**www.makezine.com/getstartedarduino**免费下载）：

```
// 例子08B: 基于Arduino的网络互动灯
// 复制并粘贴例子程序源码到Arduino IDE新建编辑文件中

#define SENSOR 0
#define R_LED 9
#define G_LED 10
#define B_LED 11
#define BUTTON 12
```

```
int val = 0;          // 存储传感器返回值的变量

int btn = LOW;
int old_btn = LOW;
int state = 0;
char buffer[7] ;
int pointer = 0;
byte inByte = 0;

byte r = 0;
byte g = 0;
byte b = 0;

void setup() {
   Serial.begin(9600);          // 打开串口，设置波特率为9600
   pinMode(BUTTON, INPUT);
}

void loop() {
val = analogRead(SENSOR);       // 读取传感器的返回值
Serial.println(val);            // 显示传感器的返回值

if (Serial.available() >0) {

   // 读取串口数据
   inByte = Serial.read();

   // 如果读取到#符号，取之后的六个代表颜色的数字
   if (inByte == '#') {

       while (pointer < 6) { // 连续读取六个数字
         buffer[pointer] = Serial.read(); // 存储在缓存区
         pointer++; //将指针向下移动一位
       }

       // 完成读取三个十六进制色彩编码的数值
       // 将十六进制数换算成十进制数值
       r = hex2dec(buffer[1]) + hex2dec(buffer[0]) * 16;
       g = hex2dec(buffer[3]) + hex2dec(buffer[2]) * 16;
       b = hex2dec(buffer[5]) + hex2dec(buffer[4]) * 16;

       pointer = 0; // 重置指针位置，以便我们可以重复使用缓存区
   }
}

btn = digitalRead(BUTTON); // 读取按钮状态并存储到变量btn中

// 检测按钮状态是否改变
if ((btn == HIGH) && (old_btn == LOW)){
   state = 1 - state;
}

old_btn = btn; // 储存当前状态
```

```
    if (state == 1) { // 如果该指示灯亮

        analogWrite(R_LED, r); //设定从计算机读取到的rgb数值
        analogWrite(G_LED, g); //将数值赋给三个LED的灯
        analogWrite(B_LED, b);
    } else {

        analogWrite(R_LED, 0); //否则关掉LED
        analogWrite(G_LED, 0);
        analogWrite(B_LED, 0);
    }

    delay(100); // 等待0.1秒
}

int hex2dec(byte c) { // 转换成十六进制字符
    if (c >= '0' && c <= '9') {
        return c - '0';
    } else if (c >= 'A' && c <= 'F') {
        return c - 'A' + 10;
    }
}
```

组装电路

图6-2是安装完成的互动装置硬件电路。按照图示，你需要三个10kΩ电阻与LED串联。

还记得在前面第5章中讲到的PWM的例子吗？LED是有极性的，在这个电路中长引脚（正极）在右边，短引脚（负极）在左边（大多数LED的扁平一端为负极）。

按图6-2所示搭建电路，电路中三个LED的颜色分别是红色、绿色、蓝色。连接电路完成后，我们导入Processing和Arduino的程序分别下载并且运行，你就可以看到灯光的变化了。如果你遇到了任何问题，请查看第7章“排疑解惑”。

现在让我们将电路放到球形的玻璃罩内。最简单，最便宜的方法就是去宜家买一个现成的“FADO”台灯，它现在的售价约合15美元。

图中使用的是独立单色LED，你也可以使用三色合一的单个RGB LED，在SparkFun（www.sparkfun.com；元器件编号COM-00105）上售价大约几美元。这种三色合一的LED共有四个引脚，其中最长的是GND引脚，它的三种颜色分别对应其中一个引脚，可接到Arduino的数字接口9、10、11（Arduino引脚与LED之间用电阻连接，就如同单独LED一样），而接地端引脚合并成一个，通常最长的引脚称“共阴极”。

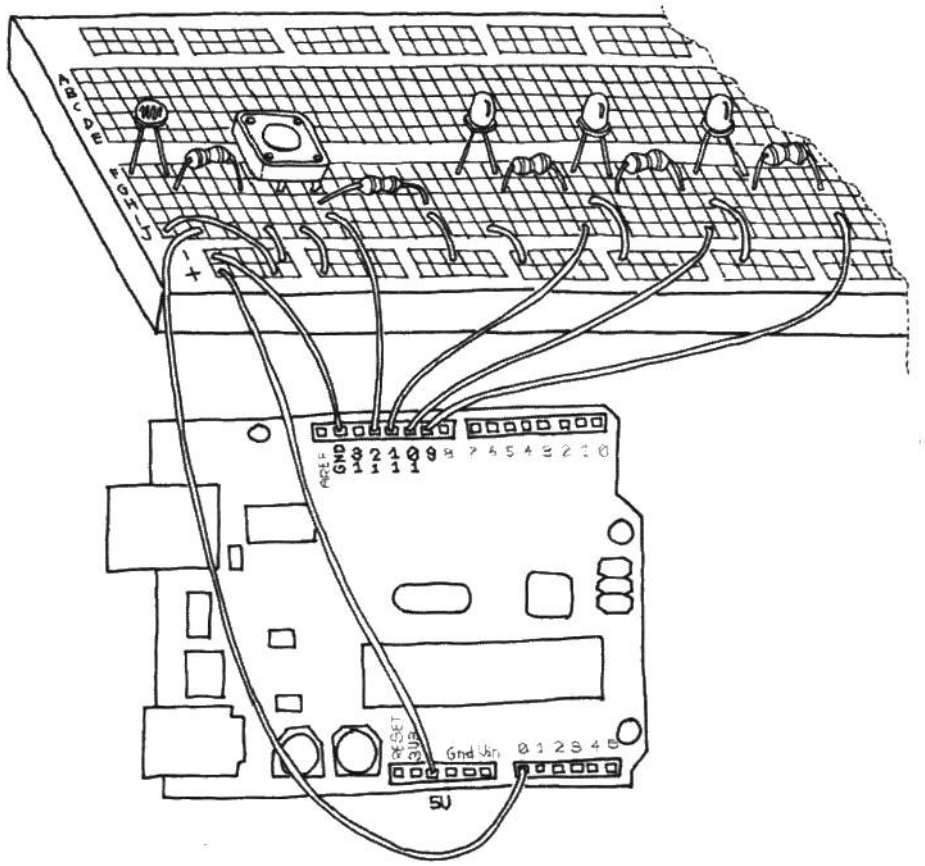


图6-2：Arduino网络灯电路

下面介绍如何安装

首先要将我们的灯具拆解，如果购买的是成品灯，要拆开底座分离电线，因为以后将不会再把它插到墙壁的插座上了。用我们自制的Arduino与RGB LED灯电路板替换原有的灯具电路和底座。

将我们的LED固定在原成品灯泡的位置，LED的电源线要在卸下前连接好。如果用的是三合一LED，请注意GND引脚要接地。

我们要找到一个让球体固定的底座，底座要根据球体大小来制作，凹槽所需的边长尺寸大约是5cm，最后可以用热熔胶将球形灯罩与底座固定。

将USB线连接到计算机。执行Processing程序，按下电源开关，你就会看到你的互动灯开始闪亮了。

作为练习，我们还可以尝试编写程序让互动灯在天色变暗时自动亮起。此外还有其他的增强功能如下：

- 增加一个倾角传感器，当角度变化时开启互动灯来代替开关；
- 增加红外热释电传感器，当检测到有人出现的时候灯光才亮起，当检测到没人的时候，装置进入休眠待机省电模式；
- 试着去建立不同的模式，通过手动模式去改变不同的颜色。

尽情发挥你的想象力，试着去实现它们，去体验互动的无限乐趣！

7/排疑解惑

在我们做实验的时候，总会遇到各种各样的问题，这时我们须要找到解决的办法。一些简单的问题通常有规律可循，但是我们仍旧要花费很多精力去解决。

当我们与电子世界和Arduino接触得越多，我们积累的知识和经验也会变得越多。永远不要被问题所困住，慢慢地你就会发现无论什么问题都会变得越来越简单。

每个通过Arduino完成的作品都是由硬件和软件两部分组成的，一旦发生问题，我们就要从多方面去研究，寻找问题的所在。通常可以遵循以下3个原则：

了解和认识

尽可能地去了解我们的系统中每一部分的性能、功能以及用途，这种方式可以使你完成对每一部分的检测。

简化以及各个击破

古罗马有这样的谚语：divide et impera，意思是分而治之。我们要试着将我们的系统分解成各个部分，并利用我们对各个部分以及元器件的了解和认识分析各个部分的工作情况。

排除干扰确认问题所在

单独地去测试每一部分，来确认这部分完好，以减少思维干扰。随着时间和经验的积累，我们就能很容易地判断出哪一部分是问题的所在，而哪一部分是完好的。

调试（Debugging，抓虫）是软件开发中的一个术语，由Grace Hopper在1940年提

出，当时的计算机还是电动机械式的，传说中有一只昆虫卡在计算机里导致计算机出现了故障。这个词就一直沿用至今。

现在我们所说的“虫”已不再是指真正的昆虫了，而是指无形的、程序上的逻辑错误，因此我们要花掉一些时间去剔除程序上的错误。

测试板子

想象一下当我们在做本书的第一个例程“LED灯闪烁”的时候，如果LED没有按照预期的情况来闪烁，我们该怎么办？岂不是有些郁闷？让我们想想我们该怎么办。

就像飞行员在起飞之前，会拿出检查清单逐一核查确保机械一切运行正常一样，我们在苛责我们的作品之前，也要确认检查以下几点，当然首先要确保Arduino已经插到计算机的USB接口上：

- 确认计算机是开启的（没错！这也许听起来有点傻，但这确实发生过）。在Arduino上标有PWR的位置上的绿灯如果是亮着的，则表示此时计算机在给Arduino供电。如果绿灯亮起但却很微弱，就表示供电部分有问题，我们需要换一条USB线试试。换过USB线后再次测试，如果问题仍然存在，可以换一个计算机USB插口试试，要是还不能解决问题，那就换一台计算机试试吧。
- 如果你的Arduino是全新的，那么标有L的黄色LED会快速闪烁，这是原厂设定的一个测试程序，用来检测Arduino是否正常工作。
- 如果您使用的是旧版（Extreme、NG、Diecimila）的Arduino并且需要外部供电，首先我们要确认外部供电适配器已经插到电源上。其次是注意Arduino上标有SV1的跳线位置，务必将跳线帽跳到靠近圆形电源插座的一边；反之，如果我们用的是USB供电，就要将跳线帽跳到靠近USB插座的一边。

注意：

当我们在Arduino上写入其他程序出现问题时，如果我们想确认Arduino是否在正常工作，可以打开Arduino IDE里面提供的范例程序，将Blink an LED下载到Arduino里。要是Arduino上的LED能有规律地闪烁，那么我们的Arduino就是正常的。

如果上述步骤都顺利完成了，我们就可以来确认Arduino是否运行正常。

用面包板测试电路

用面包板连接跳线插到Arduino的5V输出引脚，跳线另一边接到面包板上作为正极，同样将GND引出作为负极。如果标有PWR的电源指示LED熄灭，此时就要立即将电源接线拔掉（拔掉USB连接线或者外部供电电源），因为出现这个现象就表示电路板或者面包板上电路的某个部分发生了短路现象。产生这种情况时，Arduino检测出有过大电流或者功率出现，就自动切断电源以保护计算机。

注意：

发生短路时，也许会损坏计算机，但是目前计算机电源保护都做得非常好。此外，Arduino有一个保险丝，这是一个电流保护装置，当故障排除时，Arduino会自动复位。所以当出现短路问题时，我们只须将USB接线拔出即可，这将不会对你的计算机造成任何损害。

解决短路问题，我们采用的就是开头说过的“各个击破”的方法，每次只将一部分接入电路，逐一测试来找出问题所在。

通常我们都从电源部分开始（5V和GND之间的电路连接）。在确认供电部分无误时，我们再来检测每一部分的电路是否都供电正常。

在排除干扰的时候，“一次只改变一个地方”是首要的原则，这是从我的导师Maurizio Pirola教授那里学到的经验。每当我调试我的系统，解决一个一个的问题，弄得一头雾水的时候（相信我，这样的情况经常发生），导师的声音就会出现在我的脑海：“一次修改一个地方……，一次修改一个地方。”这个方法真的很好用，这可以让我们知道问题的所在，以及解决问题的方法（修改电路真的很容易让我们弄不清到底是修改了哪一部分才解决了问题，所以可见一次只修改一处是多么重要）。

每次调试错误的经历，都会充实你的知识库，慢慢地你就会变成一个电路调试专家。这会让你变得很酷，因为当一个新手说“我做的东西有问题”时，你只要帮他看一眼就会找到问题的所在，并能很快解决它。

将问题独立出来

分析问题的另一个准则是要找到一个可靠的方法来重现问题。如果你的电路中不定期地出现异常，那么就要试着找出问题发生的确定时间和原因。这样的过程能够让你认真地思考问题发生的可能原因，同时也有助于将问题叙述给其他人。

尽可能准确地描述问题也是有助于解决问题的好办法。试着找一个人来解释这个问题存在的多种情况，当你试着阐明你的问题让其他人了解时，问题的解决方法也许就会突然冒出来。在Brian W. Kernighan和Rob Pike所著的《The Practice of Programming》（Addison-Wesley，1999）一书中，提到了一个大学生的故事，他们在服务台上放了一个泰迪熊，当学生遇到了问题时必须先要跟小熊解释清楚，如果这个方法没有激发学生找出解决方案，才能寻求服务人员的帮助。

开发环境（IDE）常见问题

在某些情况下，你有可能在Windows 环境下对Arduino软件的使用出现问题。

如果你遇到了这样的问题——当你双击Arduino图标但是没有反应，我们可以试着用另一种启动Arduino的方法：双击run.bat。

Windows 用户可能会遇到另一个问题，当系统给USB的串口分配的端口号大于等于COM10时，Arduino将无法判断，这时你必须手动修改端口号，修改成小于10的COM口。方法如下：首先在桌面找到“我的电脑”，右键选择“管理”，再找到并点击“任务管理器”，找到“端口”（COM和LPT），再找到我们所使用的串口端口号。双击选择“端口设置”，找到“高级”，左键单击COM端口号的下拉框，就可以选择空闲着的且号码小于9的端口了。如果还没有解决你的问题，可以到**www.arduino.cc/en/**指南/故障排除页查找。

利用网络资源解决问题

当你陷入一个问题得不到解答时，不要自己闷着——来寻求帮助吧！Arduino的优势之一就在于它的社区，如果你能很好地描述你的问题，通常都会得到其他人的帮助。

遇到问题时，要养成善于利用搜索引擎的习惯。首先要看看是否有人在谈论同样的问题。例如当Arduino软件出现一个奇怪的错误信息时，你就可以将之直接复制粘贴到Google去搜搜看，同样的方式也可以来查询特定的函数的功能或者范例程

序。在网络上看一下，你会发现几乎所有的问题都已出现过，而解决方法也可以在网络上找到。

想要进一步学习，你可以到Arduino官方网站**www.arduino.cc**找到常见的问题与解答（**www.arduino.cc/en/Main/FAQ**），甚至可以到Arduino wiki平台Arduino playground（**www.arduino.cc/playground**）上去自由地编辑和修改文件，贡献你所了解的知识。这就是开源硬件的最大好处之一，Arduino playground给我们提供了一些简单的程序源代码和原理图，让初学者更容易上手。

如果这样还是找不到问题的解决方法，那就去（**www.arduino.cc/cgi-bin/yabb2/YaBB.pl**）论坛寻找吧。如果没有找到你的问题的答案，那就将自己的问题发布到论坛上吧。找到你的问题所在的讨论区，软件或者硬件；另外还要选择正确的语言区，因为这里有5个不同的语言讨论区。还有就是要尽可能详细地说明你的信息：

- 你使用的是哪个版本的Arduino?
- Arduino软件是在什么系统上运行的?
- 提供一份对你的系统的描述，告诉大家你想要实现的功能、所用的器件规格以及网址。

你能提供的信息越详尽，你得到的答案将会越满意。

如果你能避免以下问题发生，那么你找到答案的概率也会增加（这些规则不只是适用于Arduino论坛，网上所有的论坛都可以）：

- 全部信息都是用大写字母写出，这样会降低信息被搜索到的可能性，这样的行为就像是在我们的额头上写下“菜鸟”二字（网络论坛中，语言全部用大写字母写出，就像你说话是用吼叫的方式一样）。
- 在论坛的不同讨论区贴相同的帖子。
- 在你自己的帖子下回复一些不客气的语句，例如：“嘿，怎么没人回答？”甚至更坏的就是回复一些粗俗的语句。如果真的没有人回复你的帖子，那么就需要认真的检查你的帖子，看看表述的是不是不正确？是不是够详细？其他人是否能明白你的意思？还有，就是询问的态度是否诚恳？
- 如果你的帖子像这样：“我想用Arduino做一个航天飞机，我要怎么做？”

这很明显是想要别人帮助你完成你的工作，那么如果没有人对你的帖子感兴趣，这很正常。比较好的方式是应该先解释你想要实现的目标，然后咨询这个项目的一个部分的具体的一个问题。

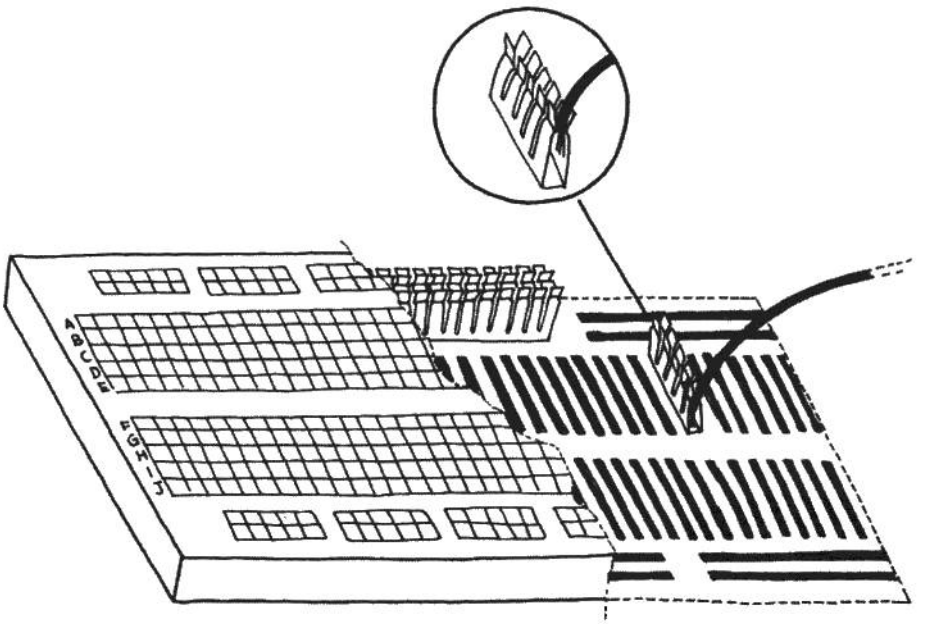
» 类似前一种情况，如果你在做有报酬的工作，你问了特定部分的具体问题，会有人很乐意帮你解答，但是如果请求别人帮助你完成全部的工作且拿不到报酬，没有人愿意回复你也是理所当然的。

» 你发布的信息要是看上去像是你的作业或者你的工作，让其他人帮你做功课，这是不当的行为。许多教授，包括我，就常常逛论坛来抓这些偷懒耍滑的学生。

附录A/面包板

在一个电路正常运行之前，我们必须做大量的修改与调整。类似于一个电子电路的草图，这是一个快速、反复的过程。在尝试不同的电路组合时，这个设计在你的手中便有了进展。最好的结果就是，应用一个能让你以最快、最实用、最小破坏性的方式修改连接的实验系统。以上要求很明显地排除了焊接这种方式，因为每次加热或冷却都会使元件处于受压状态，而且焊接也是一个很耗时的方式。

免焊接的面包板可以帮助我们解决这个问题。如图A-1所示，它是一块小塑料板，上面布满了插孔，每个插孔都有一个弹簧接头。当你把一个元器件引脚插入插孔时，它就会与同一垂直纵列的其他插孔建立电路连接。插孔间距为2.54mm。



图A-1：无焊接实验板

大多数元器件的引脚间距都是标准距离，多引脚的芯片很适合用这个面包板。面包板上的插孔并非都一样，它们也是有区别的。面包板的顶部和底部（分别标有红色“+”和蓝色“－”）的所有插孔都是水平导通的，用来给电路板供电，当你需要正电压或者接GND时，通过一条跳线就可以提供电源。关于面包板你需要知道的最后一点就是在它中间有一个宽度如一块小芯片的空白区域。每个插孔的垂直线都被从中间隔开。所以当你插入芯片时，不会使两边的芯片引脚短路。这是很人性化的设计。

附录B/认识电阻和电容

在使用这些电子元器件之前，你必须认识它们。对于初学者可能是一项艰巨的任务。

你能在商店里找到的电阻大多是有两个突出引脚和许多颜色标记的圆柱体。当第一个商业电阻问世时，由于尺寸限制，没办法将足够小的代表阻值的数字印到电阻上，所以聪明的工程师决定用色环表示它们的电阻值。

今天我们初学者必须要了解这些色环的意思。其实很简单：通常有四个色环，每种颜色代表一个数字。其中有一个环是金色，它代表电阻精密度。为了按顺序读出色环，先将电阻的金色（某些时候是银色）环朝向右侧。然后将颜色对照到相应的数字。在下面的表格中，你可以找到不同颜色的环所代表的数字。

颜色	数值
黑	0
棕	1
红	2
橙	3
黄	4
绿	5
蓝	6
紫	7
灰	8
白	9
银	10%
金	5%

例如，“棕色、黑色、橙色和金色”就表示了1 0 3 ±5%。很容易，对吧？其实不然，因为第三环实际上代表的是0的个数。因此1 0 3 是10乘以10的三次方，所以结果是10 000Ω ±5%。电子迷倾向于用千欧（kΩ）和百万欧（MΩ）表示来缩短数值，所以10 000Ω的电阻通常缩短为10kΩ，而10 000 000Ω缩短为10MΩ。请注意，由于电子工程师喜欢充分利用每样东西，你可能会在一些图表上看到4k7这样的数值，其实它的意思是4.7kΩ，或说4 700Ω。

电容理解起来会更容易：圆柱形的电容（电解质电容器）的容值一般是直接印在柱体上的。电容容值的标称单位是法拉（F），但大多数容值的单位是微法拉（μF）。所以如果你看到一个电容标称为100μF，那它就是一个100微法拉的电容。

许多盘状的电容（陶瓷电容器）通常没有列出单位，而是使用3位数字码表示“百万分之一个微法”（pF）电容值。1 000 000 pF等于1μF。这部分类似于电阻，用第三个数字表示在前两个数字之后的0的个数，但有一点不同的是：如果你看到0～5，那么它表示0的个数。6和7不用，而8和9使用方式有所不同。8表示将前两个数字乘以0.01，9表示前两个数字乘以0.1。

因此，一个电容标有104，表示100 000pF或说0.1μF；如果标有229，则表示2.2pF。

附录C/Arduino语法参考

这里给你提供一个Arduino语法函数的操作指令讲解。

更详细的内容请参考：**arduino.cc/en/Reference/HomePage**

程序结构

一个Arduino程序分为两部分：

```
void setup()
```

在这个函数里放置初始化Arduino的程序，使主循环程序在开始之前设置好相关参数。

```
void loop()
```

这是Arduino的主函数。这套程序会一直重复执行，直到电源被断开。

特殊标点及符号

Arduino语言运用了一些符号来描绘程序代码，例如，注释和程序部分。

；（分号）

Arduino每条指令（代码行）都以分号结束。这样的语法可以让你自由安排程序代码，甚至可以将两条指令放在同一行，只要你用一个分号把它们隔开就行了（然而，这样做会让你的程序代码更难以阅读）。

例如：

```
delay(100);
```

{}（大括号）

用来标注程序代码的分区。例如，当你为loop()函数编写代码时，必须在编码的前后用大括号括起。

例如：

```
void loop() {
   serial.println("ciao");
}
```

注释

这是Arduino程序编译中忽略的一部分，但是它有助于提醒你自己或别人这小段程序代码的功能。

在Arduino里有两种形式的注释。

```
// 单行注释： 这行会被忽略掉
/* 多行注释：
   你可以
   在这里
   写出
   整整一首
   诗
*/
```

常量

Arduino语言有一些有特殊意义的关键字。例如**HIGH**和**LOW**用来表示你打开或关闭（高电平或者低电平）Arduino的一个引脚（pin），**INPUT**和**OUTPUT**用来设定某个特定引脚是输入接口还是输出接口。

True和**False**，像它们的字面意思一样：表示一个条件或一个运算是真还是假。

变量

变量保存在Arduino内存中，它可以用来保存数据，你可以应用这个变量的数据来计算或者将这个数据应用到程序中。正如字面意思，变量在程序中是可以随意更改的。

因为Arduino是一个非常简易的微处理器，所以当你声明一个变量时，还必须指定它的类型。这意味着告诉处理器为你所要储存的数据留出多大的空间。

下面是一些常用的数据类型：

Boolean(布尔)

只能是真或者假两种值。

char（字符）

保存一个字符，如“A”。和任何计算机一样，Arduino将字符储存为一个数字，虽然你看到的是文字。当字符用数字来储存时，数值范围是－128到127。

> **注意：**
>
> 在计算机系统里主要有两组有效字符：ASCII和UNICODE。ASCII有127个可用字符，主要用于串行终端之间文本的传输，相应的计算机系统的例子如大型机、小型机之间传送文本。UNICODE在现代计算机操作系统中有大量的实用的字符，可代表多种语言。在传输短字节语言方面，ASCII仍很实用，如只用拉丁文（意大利语、英语）、阿拉伯数字、常见的打印机符号、标点符号等情况。

byte（字节）

储存0～255的数字。像chars一样，byte只能用一个字节（8位）的存储空间。

int（整型）

用2个字节表示一个内存空间，从－32 768到32 767之间的整数，这是用于Arduino的最普遍的数据类型。

unsigned int（无符号整型）

像int一样，也用2个字节的空间，但是无符号的前缀意味着它不能储存负数，它的范围是0到65 535。

long（长整型）

它是int的两倍大小。能够储存－2 147 483 648到2 147 483 647之间的数字。

unsigned long（无符号长整型）

无符号长整数的范围是0到4 294 967 295。

float（浮点型）

它的空间很大，能够储存浮点值，你能用它储存带小数点的数字。每个浮点型会用掉4个字节，所以要谨慎使用。

double（双精度浮点型）

双精度浮点数最大值为1.797 693 134 862 315 7乘以10的308次方。哇，非常大！

string（字符串）

用一组ASCII字符来存储文本信息（你可以用字符串通过串口发送一条信息，或者在LCD显示屏上展示）。字符串的每一个字符会占用一个字节的存储空间，加上一个零字符，表示字符串的结束。

两种表达方式：

```
char string1[] = "Arduino"; // 7 个字符 + 1个零字符
char string2[8] = "Arduino"; // 同上
```

array（数组）

数组就是通过索引存取的变量列表，它们用来建立数值的表格。例如，如果你想存储不同亮度的LED值，你固然可以创立6个变量，分别为light01、light02等，但更好的方法是用一个像int light[6] = {0, 20, 50, 75, 100, 150};这样的数组。

array这个词实际上不用在变量声明：数组用符号[]和{}来表示即可。

控制指令

Arduino利用了一些关键字控制了程序的执行流程。

If...else

If后面的括号里必须要有一个表示判断的表达式。如果表达式为真，则继续执行下面的语句；如果是假，则下面的代码将被跳过，执行else下的指令代码。你也可以只用if而不搭配else。

例如：

```
if (val == 1) {
  digitalWrite(LED,HIGH);
}
```

For

用来指明一段代码重复的次数。

例如：

```
for (int i = 0; i < 10; i++) {
   Serial.print("ciao");
}
```

switch case

如果说if就像程序的岔路口，那么*switch case*就像一个多选择环形路口。switch case根据变量的数值让程序有了更多的选择，比起一长串的if函数，switch case可以让程序看上去更为简洁。

例如：

```
switch (sensorValue) {
    case 23:
      digitalWrite(13,HIGH);
      break;
    case 46:
      digitalWrite(12,HIGH);
      break;
    default: // 以上条件都不满足时执行预设指令
      digitalWrite(12,LOW);
      digitalWrite(13,LOW);
}
```

While

当while后的条件成立时，执行大括号内的程序代码。

例如：

```
// 当sensor值小于512时，让LED闪烁
sensorValue = analogRead(1);
while (sensorValue < 512) {
    digitalWrite(13,HIGH);
    delay(100);
    digitalWrite(13,HIGH);
    delay(100);
    sensorValue = analogRead(1);
}
```

do . . . While

与while相似，但不同之处在于：while函数是先判断while后的表达式，而do...while是先执行do后的程序段，再对while后的表达式进行判断。因此do...while的主程序段至少会被执行一次。

例如：

```
do {
    digitalWrite(13,HIGH);
    delay(100);
    digitalWrite(13,HIGH);
    delay(100);
    sensorValue = analogRead(1);
} while (sensorValue < 512);
```

break

这个语句可以让程序跳出一个循环，然后继续执行循环之后的程序代码段。也用于分隔 switch case 语句的不同部分。

例如：

```
// 当sensor值小于512时，让LED闪烁
do {
    // 当按键按下时跳出循环
    if (digitalRead(7) == HIGH)
      break;
    digitalWrite(13,HIGH);
    delay(100);
    digitalWrite(13,LOW);
    delay(100);
    sensorValue = analogRead(1);
} while (sensorValue < 512);
```

Continue

在循环程序中使用，能让你跳过程序里的某些部分，然后再次判断表达式。

例如：

```
for (light = 0; light < 255; light++)
{
    // 忽略介于140～200之间的数值
    if ((x > 140) && (x < 200))
      continue;
    analogWrite(PWMpin, light);
    delay(10);
}
```

return

用return在一个函数的结尾返回一个数值。例如，现在有一个函数叫做compute-Temperature()，你想要传回储存在目前变量中的温度值，你可以这样写程序：

```
int computeTemperature() {
    int temperature = 0;
```

```
    temperature = (analogRead(0) + 45) / 100;
    return temperature;
}
```

运算符

你可以通过特殊语法让Arduino去做一些复杂的运算。+和－就是我们在学校学习的加减法，乘法用*表示，除法用/表示。

例如：

```
a = 2 + 2;
light = ((12 * sensorValue) －5 ) / 2;
remainder = 3 % 2; //返回1，因为3除以3余1（译者注：原版书籍有错误，把“返回1”写成“返回2”）
```

比较运算符

当你在运用if、while、for函数时，可以用到下列比较运算符：

== 等于
!= 不等于
< 小于
> 大于
<= 小于等于
>= 大于等于

布尔运算符

当你要结合多个判断条件时，可以用到布尔运算符。

例如，你要检测一个传感器的返回值是否是5～10之间的数，你可以这样写程序：

```
if ((sensor => 5) && (sensor <=10))
```

布尔运算符有三种：&& 逻辑与 || 逻辑或 ！逻辑非

复合运算符

这些运算符常用在类似于“递增”这些常见的函数中，让程序代码看上去更整洁。

例如让变量自身加1可以写成：

```
value = value +1;
```

但也可以使用复合运算符把上面的式子简化成：

```
value++;
```

递增和递减（++和--）

当对一个数值进行递增或递减时，要注意一点，那就是*i*++和++*i*之间的不同。*i*++是将i的值加1使*i*的值等于*i*+*1*；而当使用++*i*时，i在第一次执行程序时是*i*，直到第二次执行程序时才会被加1。这个原理同样适用于-。

+=，-=，*=及/=

这些运算符可以使程序运算式更加精简，下面是两个等价式：

```
a = a + 5;
a += 5;
```

输入输出函数

Arduino有一些处理输入输出功能的函数，在书中的一些例程里我们已经用到了一些。

pinMode(pin, mode)

将一个引脚配置成输入或者输出。

例如：

```
pinMode(7,INPUT); // 将引脚7定义为输入接口
```

digitalWrite(pin, value)

打开一个数值引脚并将其赋值高电平或者低电平，此引脚必须是前面定义过的输入或者输出模式，否则digitalWrite不生效。

例如：

```
digitalWrite(8,HIGH); // 给8号引脚高电平
```

int digitalRead(pin)

读取一个输入状态的引脚值。当引脚处于高电平状态时返回HIGH，否则返回LOW。

例如：

```
val = digitalRead(7); // 读取7号引脚的值并赋值给val
```

int analogRead(pin)

读取模拟输入引脚的值，并将其表示为0～1023之间的数值，对应0～5V的电压。

例如：

```
val = analogRead(0); // 读取模拟接口0的值，并赋值给val
```

analogWrite(pin, value)

改变该引脚的PWM输出数值，引脚（pin）可以是3、5、6、9、10、11。PWM值的改变范围是0～255，对应的电压输出值是0～5V。

例如：

```
analogWrite(9,128); // 将9号引脚的LED点亮至50%亮度
```

shiftOut(dataPin, clockPin, bitOrder, value)

发送数据到移位寄存器，用来扩大数字输出的范围，此函数使用时有一个引脚作为数据输出，另一个引脚用来表示时钟，bitOrder用来表示字节的格式（LSBFIRST是最低有效位，MSBFIRST是最高有效位），value则要输出字节的值。

例如：

```
shiftOut(dataPin, clockPin, LSBFIRST, 255);
```

unsigned long pulseIn(pin, value)

读取一个引脚的脉冲持续时间，例如使用红外传感器或者加速度计时。它们都是利用单位时间不同的脉冲来获得状态值的传感器。

例如：

```
time = pulsein(7,HIGH); // 读取7号引脚持续为高电平的时间
```

时间函数

计算与控制芯片执行程序的时间。

unsigned long millis()

检测程序执行开始到当前的时间。

例如：

```
duration = millis()-lastTime; // 表示从lastTime到当前的时间
```

delay(ms)

延迟一定毫秒的时间。

例如：

```
delay(500); // 延时500ms
```

delayMicroseconds(us)

延迟一定微秒的时间。

例如：

```
delayMicroseconds(1000); //延时1000μs
```

数学函数

Arduino也有很多自带的数学函数，包括三角函数等。

min(x, y)

x和y中返回较小的值。

例如：

```
val = min(10,20); // val的值是10
```

max(x, y)

x和y中返回较大的值。

例如：

```
val = max(10,20); // val的值是20
```

abs(x)

传回x的绝对值，正数的绝对值是其本身，负数的绝对值是其相反的数。例如：

```
val = abs(-5); // val的值是5
```

constrain(x, a, b)

判断x与a和b的关系，若x小于a，则传回a；若x介于a与b之间，则传回x本身；若x大于b，则传回b。

例如：

```
val = constrain(analogRead(0), 0, 255); // 忽略大于255的数
```

map(value, fromLow, fromHigh, toLow, toHigh)

将value的值按照fromLow与fromHigh的范围，对等转换至toLow与toHigh的范围。通常应用于读取相似信号，并将其转换到程序中所需的范围。

例如：

```
val = map(analogRead(0),0,1023,100,200); /*将模拟接口0读取的0～1023的值转换成0～100的值*/
```

double pow(base, exponent)

传回一个数（base）的指数（exponent）值。

例如：

```
double x = pow(y, 32); // 使x是y的32次方
```

double sqrt(x)

取x的平方根。

例如：

```
double a = sqrt(1138); // 1138的平方根是33.73425674438
```

double sin(rad)

传回某个角度（用弧度单位）的正弦值。

例如：

```
double sine = sin(2); // 2弧度的正弦值近似为 0.90929737091
```

double cos(rad)

传回某个角度（用弧度单位）的余弦值。

例如：

```
double cosine = cos(2); // 2弧度的余弦值近似为 -0.41614685058
```

double tan(rad)

传回某个角度（用弧度单位）的正切值。

例如：

```
double tangent = tan(2); // 2弧度的正切值近似为 -2.18503975868
```

随机数函数

如果你需要随机数，你可以利用Arduino的随机数函数来产生随机数。

randomSeed(seed)

复位Arduino的随机数发生器，会产生一系列的随机数。虽然这些数字貌似是随机产生，但是它们的顺序其实还是可以被预测的。所以如果我们需要真正的一组随机数，我们就要重新设定随机数种子。若我们没有连接一个模拟值引脚，它可以从周围环境（无线电波、宇宙射线、手机或者荧光灯的电磁干扰等）获得随机噪声。

例如：

```
randomSeed(analogRead(5)); // 利用5号引脚的噪声
```

long random(max)

long random(min, max)

传回指定区间的长整型随机数。如果没有规定最小值，则默认最小值为0。

例如：

```
long randnum = random(0, 100); // 传回一个介于0～100之间的数
long randnum = random(11); //传回一个介于0～10之间的数
```

串行通信

前面第5章我们介绍过，可以通过USB与Arduino进行串口通信。下面我们介绍一些有关串口通信的函数。

Serial.begin(speed)

为与Arduino串口通信做准备，我们可以通过Arduino的上位机软件检测返回值，这里设置通信的波特率，我们通常使用9 600，我们也可以使用其他的通信波特率，但最大值是115 200。

例如：

```
Serial.begin(9600);
```

Serial.print(data)

Serial.print(data, encoding)

将数据通过串口传回，encoding指明数据传回类型，默认为纯文本格式。

例如：

```
Serial.print(75); // 显示75
Serial.print(75, DEC); // 同上
Serial.print(75, HEX); // "4B" (75的十六进制表达)
Serial.print(75, OCT); // "113" (75的八进制表达)
Serial.print(75, BIN); // "1001011" (75的二进制表达)
Serial.print(75, BYTE); // "K" (K的ASCII码值是75)
```

Serial.println(data)

Serial.println(data, encoding)

与Serial.print(data)相同，只是在传回数据的末尾多加一个换行符（\r\n）。换行符的意义就等于你在输入一些文字后敲入的回车键。

例如：

```
Serial.println(75); // 显示 "75\r\n"
Serial.println(75, DEC); // 同上
Serial.println(75, HEX); // "4B\r\n"
Serial.println(75, OCT); // "113\r\n"
Serial.println(75, BIN); // "1001011\r\n"
Serial.println(75, BYTE); // "K\r\n"
```

int Serial.available()

传回一个数值，告诉上位机有多少字节没有被read()函数读取，若Serial.available()返回值是0，则代表串行数据都已被read()读取。

例如：

```
int count = Serial.available();
```

int Serial.read()

读取一个字节的串行数据。

例如：

```
int data = Serial.read();
```

Serial.flush()

因为数据传输的速度要大于Arduino程序处理的速度，所以Arduino会将数据先存放在缓存区中。如果有需要，我们可以利用Serial.flush()函数来清空缓存区，以确保缓存区的数据都是最新的。

例如：

```
Serial.flush();
```

附录D/阅读电路简图

到目前为止，我们已经详细地描述了如何组装电路，相信你的大脑里已经有了一些基本的概念，但是只用具体的文字来表达确实不是一件很简单的事。

类似的问题出现在各个领域。就像在音乐方面，作者必须用乐谱将他们大脑中的旋律表达出来。

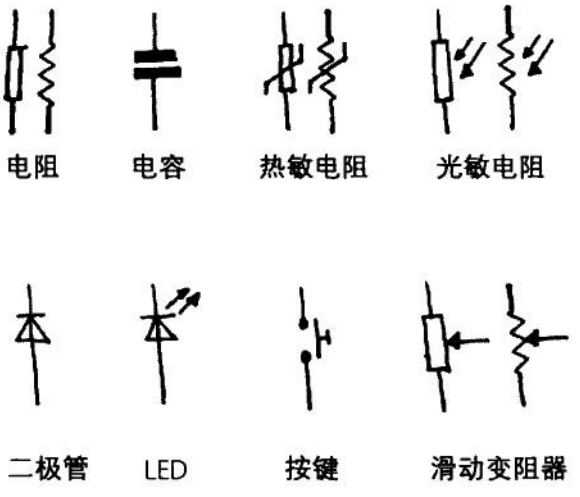
务实的工程师们已经开发出一种快速的方式来描绘电路的本质，用来将电路特性清晰化，也更方便工程师们沟通。电路图是一种电路绘制标准，让其他人能更快地了解到设计者的意图。

电路原理图是让人能够理解电路的一种方式。每个元器件都有代表它的一种特定的符号，这样一类具体的元器件就可以用一个抽象的符号来表示了。例如，电容是把两块金属板用空气或者塑料分离开而做成的，因此电容的标志是：

另一个例子是电感，电感是由铜线缠绕成的圆柱形的线圈，因此它的符号如左图。

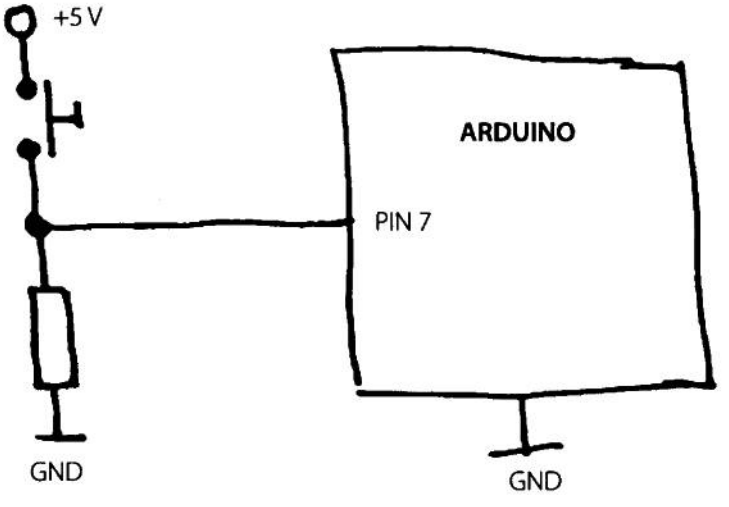
两个元器件的连接通常用直线表示，当两条线有交汇且有连接时，我们要在交叉的位置上点一个圆点，表示两条线之间确实是连通的，如左图。

下面是我们要了解的常用元器件的符号及其含义的列表：



在实际的电路图中，你也许会遇到很多符号的变化（例如图上所示的两种“电阻”符号的变化）。在**en.wikipedia.org/wiki/Electronic_symbol**你可以查到更多的电阻元器件符号的表示方法。按照通常习惯，电路图应是自左到右绘制的，例如，一台收音机的电路绘制应从左侧天线开始，随着无线电信号移动的路径，画到右侧的扬声器（这是正确的画法）。

下图描述了本书中按钮实验的电路图：



O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——**Wired**

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——**Business 2.0**

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——**CRN**

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——**Irish Times**

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即瞬的机会，尽管大路也不错。”

——**Linux Journal**

[General Information]
书名=爱上Arduino
作者=（美）班兹著
页数=103
出版社=北京市：人民邮电出版社
出版日期=2011.08
SS号=12856795
DX号=000008150890
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=00000815089O&d=9E903BEDD8B72C8B587CE6A0FEC173AA