

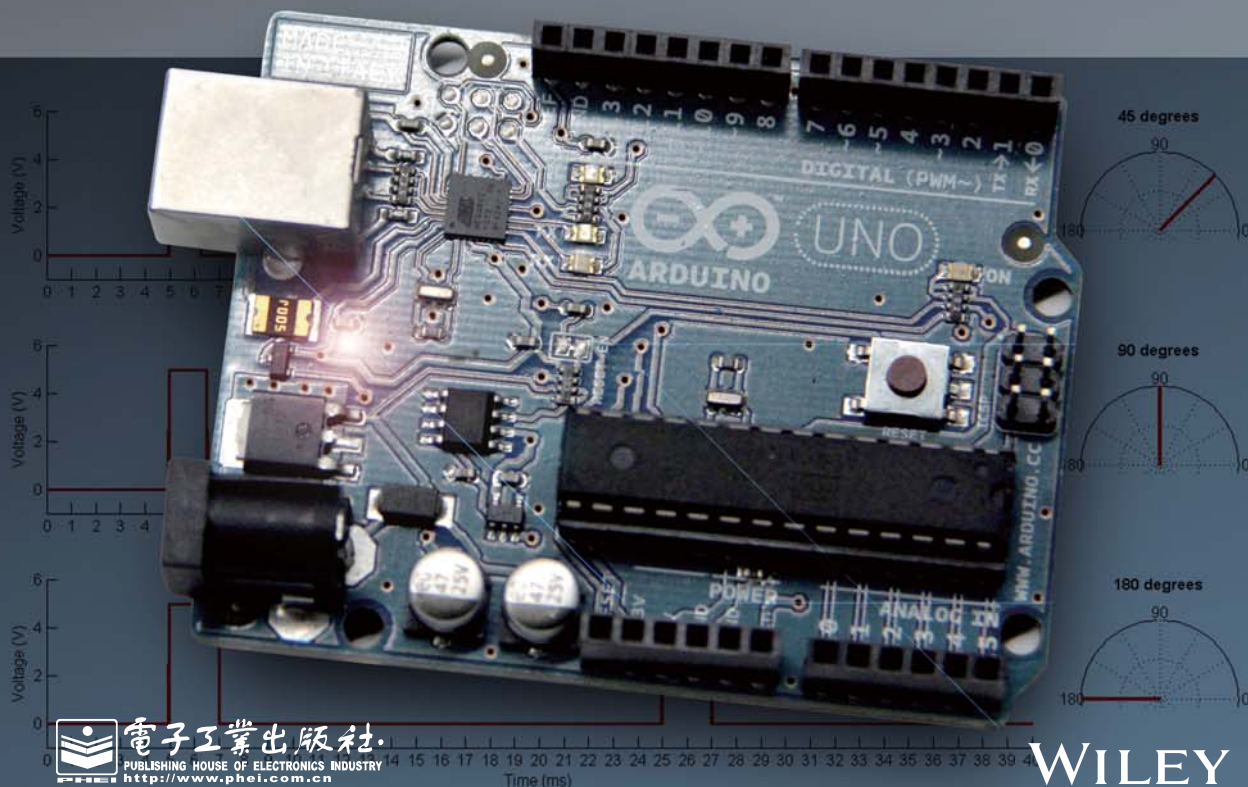
用世界上最流行的微控制器，
来制作酷炫、实用、兼具艺术性和教育意义的作品。

Exploring Arduino:
Tools and Techniques for Engineering Wizardry

Arduino魔法书

实现梦想的工具和技术

[美] Jeremy Blum 著
况琪 王俊升 译



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

WILEY

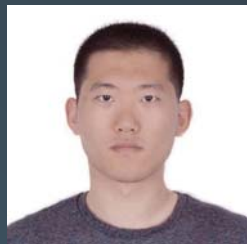
译者简介



况琪，北京师范大学计算机科学与技术专业理学学士，北京师范大学软件工程硕士。专注底层技术的研究与应用，熟悉C语言、嵌入式系统和电子工艺。也在物联网与智能家居、机器视觉、数字图像处理、互联网及信息系统方面有过科研或项目经验，与中国航天员中心合作的实验程序曾随“神舟九号”飞船进入太空。关注开源硬件发展和创客运动，曾出版译著《Arduino实战》。

个人网站: <http://kuangqi.me>

新浪微博: <http://weibo.com/kqwd>



王俊升，北京师范大学计算机科学与技术专业理学学士，德克萨斯大学达拉斯分校硕士。关注开源硬件和互联网创业，也在数据挖掘、可视分析、数值算法及软件工程等方面开展过相关研究。

作者简介

Jeremy Blum，在康奈尔大学获得电气和计算机工程专业的硕士学位，将激情付诸电气工程领域，设计了太阳能家庭能源监控系统、革命性的光导纤维LED照明系统及追踪太阳的智能太阳能面板。

他还设计了广受称赞的义肢控制方法、手势识别系统及楼宇自动化系统等许多项目，设计了MakerBot Replicator 3D打印机的电子系统，以及MakerBot Digitizer 3D扫描仪的原型电子系统和固件。作为知名组织Creative Machines Lab的研究员，他致力于制作能够自我组装的机器人、自学习四足机器人及重新定义个人制造的3D打印机，并将这项研究呈送给了同行审阅的期刊和远在印度的学术会议。

他制作的YouTube视频已经将数百万人带入了工程学的大门，是互联网上最流行的Arduino入门教程。他曾开发了入选探索频道的开源硬件项目和入门教程，并因此被国际上的开源社区和“创客”社区所熟知，赢得了一些奖项和黑客马拉松(hack-a-thons)大赛，并入选美国电气和电子工程师学会的2012工程学新面孔榜单。

Jeremy Blum的网站:
www.jeremyblum.com。

Exploring Arduino:
Tools and Techniques for Engineering Wizardry

Arduino魔法书

实现梦想的工具和技术

[美] Jeremy Blum 著
况琪 王俊升 译

电子工业出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

《Arduino魔法书：实现梦想的工具和技术》是一本关于使用Arduino实现梦想的作品。作者Jeremy Blum有一句名言：“我们生活在未来。”利用可用的工具和将在本书中将学到的知识，任何人都有机会、也有能力选择一款Arduino微控制器并在几分钟内利用它控制周围的世界——这一梦想直到最近才成为可能。微控制器是一个可编程的平台，它使我们能够使用相对简单的命令，来定义复杂的机械、电气和软件系统的操作。通过对本书的学习，我们将使用Arduino制作各种作品，从运动检测到无线控制系统再到互联网通信。

本书适用于任何经验层级的Arduino爱好者。读者不需要具备或者只需具备少量编程和电气工程方面的经验。为了照顾到各个层次的读者，本书设计了各种可选章节、小栏目或短节选，并在其中详细解释了具体的概念。

Exploring Arduino: Tools and Techniques for Engineering Wizardry, 978-1118549360, Jeremy Blum

Copyright © 2013 by Wiley Publishing, Inc., Indianapolis, Indiana

All rights reserved. This translation published under license.

No part of this book may be reproduced in any form without the written permission of John Wiley & Sons, Inc. Copies of this book sold without a Wiley sticker on the back cover are unauthorized and illegal.

本书简体中文版专有翻译出版版权由美国 John Wiley & Sons, Inc.公司授予电子工业出版社。未经许可，不得以任何手段和形式复制或抄袭本书内容。

本书封底贴有 John Wiley & Sons, Inc.防伪标签，无标签者不得销售。

版权贸易合同登记号 图字：01-2014-3370

图书在版编目（CIP）数据

Arduino 魔法书：实现梦想的工具和技术 /（美）布鲁姆（Blum, J.）著；况琪，王俊升译. —北京：电子工业出版社，2014.10

书名原文：Exploring arduino:tools and techniques for engineering wizardry

ISBN 978-7-121-24067-6

I. ①A… II. ①布… ②况… ③王… III. ①单片微型计算机 IV. ①TP368.1

中国版本图书馆 CIP 数据核字(2014)第 187091 号

策划编辑：林瑞和

责任编辑：徐津平

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×980 1/16 印张：20.5 字数：400 千字

版 次：2014 年 10 月第 1 版

印 次：2014 年 10 月第 1 次印刷


定 价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010) 88258888。

献给我的祖母——那个终生保持着好奇心并
善于激励他人的人，她启发我每天不断向前。



关于作者

Jeremy Blum 在康奈尔大学取得了电气和计算机工程专业的硕士学位，他还 在同一领域取得了学士学位。在康奈尔大学，他通过由他发起并领导的康奈尔大学 可持续设计组织（Cornell University Sustainable Design）监督了国内外几座生态 建筑的设计和施工，该组织是一个美国承认的可持续设计组织，曾受到美国 和世界绿色建筑委员会（U.S. and World Green Building Councils）CEO 的特别称赞。 同样，Jeremy 也将他的激情付诸电气工程领域，设计了太阳能家庭能源监控系统、 革命性的光导纤维 LED 照明系统及追踪太阳的智能太阳能面板。他还负责协助 启动了一个首创的创业者协同工作空间，每年致力于许多学生创业项目（也包括 一些他自己的创意）的合作开发。

Jeremy 还设计了广受称赞的义肢控制方法、手势识别系统及楼宇自动化系 统等许多项目。他设计了 MakerBot Replicator 3D 打印机（被世界各国的人们使 用，包括像 NASA 这样知名组织）的电子系统，以及 MakerBot Digitizer 3D 扫描 仪的原型电子系统和固件。作为知名组织 Creative Machines Lab 的研究员，他致 力于制作能够自我组装的机器人、自学习四足机器人及重新定义个人制造的 3D 打印机。他将这项研究呈送给了同行审阅的期刊和远在印度的学术会议。


Jeremy制作的YouTube视频已经将数百万人带入了工程学的大门，是互联网 上最流行的Arduino入门教程。他曾开发了入选探索频道（Discovery Channel）的 开源硬件项目和入门教程，并因此被国际上的开源社区和“创客”社区所熟知， 赢得了一些奖项和黑客马拉松（hack-a-thons）大赛。Jeremy入选了美国电气和电 子工程师学会（American Institute of Electrical and Electronics Engineers）的 2012 工程学新面孔（2012 New Face of Engineering）榜单。

他通过自己的公司 Blum Idea Labs LLC 提供工程学咨询服务，并向纽约的年

轻学生教授工程学和可持续发展课程。Jeremy 的激情正在通过创新的工程学解决方案，改变着人们的生活和我们的家园。你可以在 Jeremy 的网站上进一步了解他和他工作，网址是：www.jeremyblum.com。

关于技术编辑

Scott Fitzgerald 是一名艺术家和教育工作者。自 2006 年起，他就在教学中将 Arduino 平台当作教具，并且自 2005 年起，就在纽约大学的交互性电信项目（Interactive Telecommunications Program, ITP）中教授物理计算，向艺术家和设计师介绍微控制器。Scott 为 Arduino 团队工作，为新产品撰写文档，并创作入门教程来向人们介绍 Arduino 平台。他在 2011 年曾经是 *Making Things Talk* 第 2 版的技术编辑，并在 2012 年为 Arduino 官方入门套件（Arduino Starter Kit）撰写了附赠的图书。



致 谢

首先，必须感谢在 Wiley 出版社的朋友帮助我完成了这本书：感谢 Mary James 当初鼓励我撰写这本书；感谢 Jennifer Lynn 全程监督了我的写作。我还欠 Scott Fitzgerald 一个大大的感谢，感谢他在本书的技术编辑过程中的批判性眼光。

如果没有 element14 的鼎力支持，我可能永远无法制作出我的 Arduino 入门系列视频教程，它们是本书的序曲。尤其是 Sabrina Deitch 和 Sagar Jethani，他们是绝佳的合作伙伴，我很荣幸能与他们一起工作。

在撰写本书主要部分的同时，我还要完成硕士学位并经营两家公司，所以我要特别感谢我的教授和同事，感谢他们在我尝试平衡所有责任时理解我。

最后，我想感谢我的家庭，特别是我的父母，还有我的兄弟 David，是他们持续的激励，让我明白了做这些事情的意义。

前言

你拥有绝佳的时机。正如我经常喜欢说的，“我们生活在未来”。利用如今可用的工具和书中的知识，你就有机会、也有能力让梦想变成现实。让任何人选择一款微控制器并在几分钟内利用它控制周围的世界——这一梦想直到最近才成为可能。你可能已经猜到了，微控制器是一个可编程的平台，它使你能够使用相对简单的命令，来定义复杂的机械、电气和软件系统的操作。**Arduino** 微控制器平台将成为你的新宠，带你探索电子、编程、人机交互、艺术、控制系统及更加广阔的世界，有了它就有了无限的可能。通过对本书的学习，你将掌握使用 **Arduino** 制作各种作品的方法，从运动检测到无线控制系统再到互联网通信。

无论你是工程领域彻头彻尾的新手，还是打算入门嵌入式系统设计的经验丰富的老兵，**Arduino** 都是一个绝佳的起点。你在寻找一本 **Arduino** 开发的通用参考手册吗？本书也非常适合你，它会引导你完成一系列具体项目，而你以后也可以再次翻阅本书来查找代码片段、最佳范例和系统原理图等资料。电气工程、系统设计、编程实践……这些你在阅读时学到的知识在 **Arduino** 平台之外也有广泛的应用。通过对本书的阅读，你将具备承担各种工程项目的的能力，不论它们使用的是 **Arduino** 还是其他平台。

本书对象

本书适合任何经验层级的 **Arduino** 爱好者。章节是彼此相关的，要利用前面章节中的概念和项目组件来实现更加复杂的想法。但不要担心，当你面临全新的、复杂的想法时，会有一个交叉引用来提示你相关概念第一次出现在书中的何处，以便很快回想起来。

本书假设你不具备或者只具备很少的编程和电气工程方面的工作经验。为了照顾到各个层次的读者，本书设计了各种可选章节、小栏目或短节选，在其中详细解释了具体的概念。尽管这些小栏目对你更深入地理解 Arduino 的工作原理来说并不是必需的，但它们为更具好奇心的读者提供了进一步关注技术性话题的机会。

你将通过本书学到什么

本书不是食谱书。如果你想遵循手把手地指导自己如何搭建一个具体项目的书，而不需要解释为什么要这么做，那本书便不适合你。你可以把本书作为一本导论，其中介绍了电气工程、计算机科学、产品设计和高级思维；你也可以把 Arduino 作为媒介，通过动手实践的方式来体验这些概念。

当我们在本书中演示如何搭建 Arduino 项目的硬件组件时，你学到的将不仅是如何将它们连接在一起，还有如何阅读原理图，为什么将某个器件用于某项功能，如何通过数据手册为自己的项目选择合适的器件等。在编写软件时，我提供了完整的程序代码，但你会逐步走过几个迭代性的过程来创建最终的程序。这会帮助你巩固具体的程序函数、良好的代码格式化实践和对算法的理解。

本书会讲授物理概念、算法、数字化设计原理及具体的 Arduino 编程概念。我希望，通过对本书中案例的学习，你不仅能成为一名熟练的 Arduino 开发者，还能学会一些技巧，去开发更加复杂的电气系统，并在其他领域或其他平台上继续从事工程学方面的工作。

本书使用的标注

在本书中使用了下列标注和图标，以在一些最重要或最有用的信息上吸引你的注意。

警告 请你一定要留意这样的条目。如果一旦错误地执行某些步骤就可能导致电子器件损坏，你就会看到这种警告。

提示 这些条目包含了关于如何执行简单任务的快速提示，这可能对你手头的任务有所帮助。

注意 这些条目包含了对你来说可能很重要的额外信息,包括视频和在线资料的相关链接,它们会让具体项目的开发变得更加轻松。

样例标题

这样的条目对当前话题或相关话题做了更加深入的讲解。

获取器件

幸运的是,你可以方便地获得实施本书中项目所需的器件。本书译者已经根据本书内容提供了这些器件的获得地址: <http://kuangqi.me/>。

建议你先买基本套件。随着阅读的深入,你还可以购买扩展套件。不想购买套件吗?别担心。在每章的开头,有一个该章所需的详细器件列表。本书原著的专题网站 www.exploringarduino.com,也提供了一些链接,通过它们可以找到每章所需的器件。

注意 你是否已经从 Newark 获得了本书的捆绑套装?如果是,那么你就可以继续了。

你需要什么

除了用来搭建 Arduino 项目的具体器件以外,你的 Arduino 探险之旅还需要一些其他工具和材料。最重要的是,你需要一台兼容 Arduino 集成开发环境(IDE)的计算机(Mac OS X 10.4 以上版本,Windows XP 以上版本,或者一个 Linux 发行版)。在必要时我会为这些操作系统中的操作做出说明。

你可能还想要一些额外的工具,用来调试、组装硬件等。它们对完成本书中的项目来说不是必需的,但在你拓展电气工程技能时,这些工具在其他项目中迟早有用。我推荐如下工具。

- 一把烙铁和一些焊料(完成本书中的项目并不需要焊接,但你可能希望在洞洞板上组装自己的电路,或者你可能想购买需要焊接组装的扩展板。)
- 一台万用表(在调试过程中有用,但不是必需的。)
- 一组小型螺丝刀
- 一支热熔胶枪

源代码和数字化内容

本书的主要专题网站是 www.exploringarduino.com，它是由本书作者维护的。你可以在这个网站上找到每章的代码并下载（还有视频、链接及其他有用的材料）。Wiley 也维护了一个关于本书数字化内容的仓库，网址是 www.wiley.com。本书代码可在 www.wiley.com/go/exploringarduino 网页的“Download Code”标签下下载。

你也可以在 www.wiley.com 上根据 ISBN（本书原著的 ISBN 为 978-1-118-54936-0）搜索本书原著，从而找到代码。

在每章的开始处，你可以找到本章主要代码的下载地址。在每章中，你还能在代码清单的标题和文本中找到你需要的代码文件的文件名。

在网站 www.exploringarduino.com 和 www.wiley.com 上提供的代码是 ZIP 压缩包。下载代码之后，使用适当的工具解压缩即可。

注意 由于许多图书的书名大同小异，所以通过 ISBN 搜索可能会更方便一些。本书原著的 ISBN 是 978-1-118-54936-0。

勘误

我们尽力保证文本和代码的正确性。然而人无完人，错误在所难免。如果你在本书中发现了错误，比如拼写错误或者代码错误，我们会感谢你的反馈。通过提交勘误，你可能会消除其他读者几个小时的困惑，与此同时，你可以帮助我们提供更有用的信息。

要查找本书原著的勘误，请访问 www.wiley.com/go/exploringarduino 并单击“Errata”链接。

补充材料和技术支持

在你的 Arduino 探险之旅当中，不可避免地会遇到问题和麻烦。使用 Arduino 最大的一个好处就在于可以在网上找到绝佳的技术支持社区。这个极为活跃的 Arduino 用户大本营会十分乐意帮助你。下面列出了一些可能会帮到你的资源。

- 官方 Arduino 参考手册
www.arduino.cc/en/Reference/HomePage

- 我的 Arduino 入门教程系列视频
www.jeremyblum.com/category/arduino-tutorials
- adafruit 公司的 Arduino 入门教程系列视频
learn.adafruit.com/category/learn-arduino
- SparkFun 的电子学入门教程
learn.sparkfun.com/tutorials
- Arduino 官方论坛
www.arduino.cc/forum
- element14 Arduino 社区
www.element14.com/community/groups/arduino

如果你用尽了这些资源，却还是无法解决你的问题，则请在 Twitter (@sciguy14) 上联系我，我也许能帮上忙。你也可以直接通过我的网站上的联系页面 (www.jeremyblum.com/contact) 来与我取得联系，但我通常没法保证快速回复。

Arduino 是什么

Arduino 原型平台的最大特点在于你想让它是什么，它就是什么。Arduino 可以是一个植物灌溉自动控制系统，也可以是一个 Web 服务器，甚至可以是四轴飞行器的自动驾驶仪。

Arduino 是一个微控制器开发平台，它配备了直观的编程语言，你可以使用 Arduino 集成开发环境 (IDE) 来开发它。你可以为 Arduino 安装传感器、效应器、灯、扬声器、扩展模块 (shield, 本书译作“扩展板”) 及其他集成电路，来将 Arduino 变成一个可编程的“大脑”，应用于几乎任何控制系统。

本书不可能涵盖 Arduino 能做的所有事情，只要你的想象力是无穷的，可能性就是无穷的。因此，本书的目标是通过实施项目来让你熟悉 Arduino 的功能，学习开发自己的项目时所需的技巧。

你会在第 1 章中学到更多关于 Arduino 及这块板卡的变体。如果你渴望了解 Arduino 的所有内部工作原理，那么你很幸运：它是完全开源的，所有的电路图和文档都可以在 Arduino 网站上免费获得。附录提供了一些 Arduino 的技术指标。

一个开源平台

如果你是一个开源世界的新人，则你一定会满意的。本书不会详细介绍开源硬件运动，但你应该了解一点这种思想，是它让 Arduino 变得如此美妙。如果你想全面了解什么是开源硬件，则请查阅开源硬件协会网站（www.oshwa.org/definition）上的定义。

注意 你可以从我的 TEDx 演讲中学到所有关于开源运动的知识：
<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头处的 Wiley 网站上找到这个视频。

由于 Arduino 是开源硬件，所以任何人都可以免费获取所有的设计文件、原理图及源代码被。这不仅意味着你可以更方便地改造 Arduino，以用于非常特定的功能，还意味着你可以将 Arduino 平台整合到你的设计中，制作并销售 Arduino 的复制品，并在其他项目中使用 Arduino 软件库。尽管本书使用的几乎全部是官方 Arduino 硬件，但你也可以使用数以百计的 Arduino 衍生板卡（它们通常添加了特别的功能）来制作本书中的项目。

Arduino 的开源许可证还允许商业化地重用他人的设计（只要你不在自己的设计上使用 Arduino 商标）。所以，如果你在一个令人兴奋的项目原型中使用了 Arduino，而你还想将其转化为一个商业产品，那么你是可以这样做的。举例来说，你可能听说过像 MakerBot Replicator 3D 打印机这样的产品，它的电子系统是基于 Arduino Mega 平台的（www.thingiverse.com/thing:16058）。（大爆料：我设计了那块主板！）

在使用本书时，请确保遵守了本书中源代码和硬件的许可证。一些许可证要求你在发布基于他人已有的设计时提供原始作者的信息；有的则要求你必须在相同的许可下分享你的改进工作。这样的分享帮助社区成长，所有了不起的在线文档和技术支持也正是源自于此，在你的 Arduino 探险之旅中，你一定会经常参考它们。我为本书编写的所有的代码示例（除非特别声明）都是基于 GNU 通用公开许可（GPL）的，你可以随心所欲地使用它们。

本书之外

有人可能对我在 YouTube 上流行的系列视频“Arduino 和电子学入门教程”（www.youtube.com/sciguy14）很熟悉了。本书中我也多次提到它们，你可以通过这些视频更为深入地了解书中所涵盖的技术主题。对电子器件和微控制器利用计

计算机科学知识，将它们创造性地组合，来创造卓越的作品，如果你对此感到好奇，可以在我的作品集 (www.jeremyblum.com/portfolio) 中查看样例项目。同 Arduino 一样，我的绝大部分作品都通过开源许可发布，允许你针对自己的需求方便地复制它们。

我渴望知道你利用在本书中学到的技巧做了什么，并期待你将这些与我和其他朋友分享。祝你在 Arduino 探险之旅中好运！



第 1 部分	Arduino 工程基础	/ 1
第 1 章	让你的 Arduino 闪烁	/ 2
	探索 Arduino 生态系统	/ 3
	Arduino 功能	/ 3
	Arduino 板卡	/ 7
	创建你的第一个程序	/ 12
	下载并安装 Arduino IDE	/ 12
	运行 IDE 并连接 Arduino	/ 13
	分析你的第一个程序	/ 15
	本章小结	/ 17
第 2 章	数字输入、输出和脉冲宽度调制	/ 18
	数字输出	/ 19
	连接一个 LED 并使用面包板	/ 19
	数字输出端口编程	/ 23
	使用 for 循环	/ 24
	analogWrite()与脉冲宽度调制	/ 25
	读取数字输入	/ 28
	读取带有下拉电阻的数字输入	/ 28
	使用“有弹性的”按键	/ 30
	制作一个可控的 RGB LED 夜灯	/ 33

本章小结	/ 37
第 3 章 读取模拟传感器	/ 38
理解模拟和数字信号	/ 39
比较模拟和数字信号	/ 39
将模拟信号转换为数字信号	/ 40
利用 Arduino 读取模拟传感器: analogRead()	/ 41
读取电位器	/ 42
使用模拟传感器	/ 45
利用可变电阻制作自己的模拟传感器	/ 50
使用阻性分压器	/ 50
利用模拟输入控制模拟输出	/ 52
本章小结	/ 54
第 2 部分 控制环境	/ 55
第 4 章 三极管与电机驱动	/ 56
驱动直流电机	/ 57
操作大电流感性负载	/ 58
利用 PWM 控制电动机转速	/ 62
使用 H-桥控制直流电机的方向	/ 64
驱动伺服电机	/ 70
理解连续旋转伺服电机和标准伺服电机的区别	/ 71
理解伺服电机控制	/ 71
控制伺服电机	/ 75
制作扫描式距离传感器	/ 76
本章小结	/ 80
第 5 章 发出声音	/ 81
理解扬声器的工作原理	/ 82
声音的性质	/ 82
扬声器是如何发声的	/ 83
使用 tone()发出声音	/ 84
包含定义文件	/ 85
给扬声器接线	/ 86
产生声音序列	/ 88

理解 tone()函数的限制	/ 91
制作一架微型钢琴	/ 91
本章小结	/ 94
第 6 章 USB 和串口通信	/ 95
理解 Arduino 的串口通信功能	/ 96
使用内置或外置 USB-串口转换器的 Arduino 板卡	/ 97
使用次级含 USB 功能 ATmega MCU 来模拟串口转换器的 Arduino 板卡	/ 99
使用单个带 USB 功能 MCU 的 Arduino 板卡	/ 101
带有 USB-Host 功能的 Arduino 板卡	/ 101
监听 Arduino	/ 102
使用 print 语句	/ 102
使用特殊字符	/ 103
改变数据类型表示	/ 105
与 Arduino 通信	/ 105
从计算机或其他串口设备上读取信息	/ 106
与桌面应用程序通信	/ 112
与 Processing 通信	/ 113
学习 Arduino Leonardo（及其他基于 32U4 的 Arduino）的特殊技巧	/ 119
模拟键盘	/ 119
模拟鼠标	/ 124
本章小结	/ 127
第 7 章 移位寄存器	/ 129
理解移位寄存器	/ 130
发送并行和串行数据	/ 131
使用 74HC595 移位寄存器	/ 131
使用 Arduino 进行串行移位输出	/ 134
在二进制和十进制间转换	/ 136
用移位寄存器控制灯光动画	/ 137
搭建“光骑士”	/ 137
用 LED 条形图响应输入	/ 139
本章小结	/ 142

第 3 部分 Arduino 通信接口	/ 143
第 8 章 I²C 总线	/ 144
I ² C 总线的历史	/ 145
I ² C 硬件设计	/ 145
通信方案与 ID 值	/ 146
硬件要求和上拉电阻	/ 147
与 I ² C 测温探头通信	/ 148
设置硬件	/ 148
参考数据手册	/ 149
编写软件	/ 151
将移位寄存器、串口通信和 I ² C 通信结合	/ 153
为温度监控系统搭建硬件	/ 153
修改嵌入式程序	/ 154
编写 Processing 程序	/ 156
本章小结	/ 159
第 9 章 SPI 总线	/ 160
SPI 总线概述	/ 161
SPI 硬件和通信设计	/ 162
硬件配置	/ 162
通信方案	/ 163
比较 SPI 与 I ² C	/ 164
与 SPI 数字电位器通信	/ 164
从数据手册中获取信息	/ 164
设置硬件	/ 167
编写软件	/ 169
用 SPI 数字电位器制作视听显示系统	/ 171
设置硬件	/ 172
修改软件	/ 173
本章小结	/ 175
第 10 章 连接液晶显示器	/ 176
设置 LCD	/ 177
使用 LiquidCrystal 库写入 LCD	/ 180

向显示器输出文本	/ 180
创建特殊符号和动画	/ 182
制作一个私人自动恒温器	/ 185
设置硬件	/ 185
用按键调节设定值	/ 188
添加声音报警和电扇	/ 189
整合：完整的程序	/ 190
升级项目	/ 194
本章小结	/ 194

第 11 章 使用 XBee 收发器进行无线通信 / 195

理解 XBee 无线通信	/ 196
XBee 无线收发器	/ 197
XBee 无线收发器扩展板和串口连接	/ 198
配置 XBee	/ 201
用扩展板或 USB 适配器配置	/ 202
选择 Xbee 设置并将 XBee 连到主机上	/ 203
用 X-CTU 配置 XBee	/ 204
用串口终端配置 XBee	/ 207
与计算机无线通信	/ 209
为远程 Arduino 供电	/ 209
回顾串口示例：用电位器控制 Processing	/ 211
回顾串口示例：控制一个 RGB LED	/ 214
与另一个 Arduino 通信：制作无线门铃	/ 216
系统设计	/ 217
发射器硬件	/ 218
接收器硬件	/ 218
发射器软件	/ 219
接收器软件	/ 220
本章小结	/ 223

第 4 部分 高级的话题和项目 / 225

第 12 章 硬件和定时器中断 / 226

使用硬件中断	/ 227
--------	-------

理解轮询和中断间的折中	/ 228
理解 Arduino 的硬件中断能力	/ 229
搭建并测试硬件消抖动的按键中断电路	/ 230
使用定时器中断	/ 237
理解定时器中断	/ 237
获取软件库	/ 238
近乎于同时地执行两个任务	/ 238
搭建一个中断驱动的音响	/ 239
音响硬件	/ 239
音响软件	/ 240
本章小结	/ 242
第 13 章 用 SD 卡记录数据	/ 243
数据记录的准备工作	/ 244
用 CSV 文件格式化数据	/ 244
为数据记录准备 SD 卡	/ 245
将 SD 卡接入 Arduino	/ 249
SD 卡扩展板	/ 249
SD 卡 SPI 接口	/ 253
写入 SD 卡	/ 253
读取 SD 卡	/ 258
使用一个实时时钟	/ 261
理解实时时钟	/ 261
使用实时时钟	/ 263
做一个入口记录仪	/ 268
记录仪硬件	/ 268
记录仪软件	/ 270
数据分析	/ 273
本章小结	/ 274
第 14 章 将你的 Arduino 联网	/ 275
互联网、Arduino 和你	/ 276
网络术语	/ 276
客户端与服务器	/ 279
将 Arduino 联网	/ 279

在网上控制 Arduino	/ 279
准备 I/O 控制硬件	/ 280
设计一个简易网页	/ 280
编写 Arduino 服务器程序	/ 282
通过网络控制 Arduino	/ 286
把实时数据发送给绘图服务	/ 290
在 Xively 上创建实时数据源	/ 290
添加数据源组件	/ 295
本章小结	/ 299
附录 ATmega 数据手册和 Arduino 原理图揭秘	/ 300
阅读数据手册	/ 300
分解数据手册	/ 300
理解器件引脚定义	/ 302
理解 Arduino 原理图	/ 304

第1部分

Arduino 工程基础

本部分内容

第 1 章 让你的 Arduino 闪烁

第 2 章 数字输入、输出和脉冲宽度调制

第 3 章 读取模拟传感器

第 1 章

让你的 Arduino 闪烁

本章所需的器件：

Arduino Uno

USB 电缆

本章的代码和数字化内容

本章的代码、视频及其他数字化内容可以在以下网址中找到：
www.exploringarduino.com/content/ch1。

另外，所有的代码都可以在 www.wiley.com/go/exploringarduino 网页上的“Download Code”标签中找到。在第 1 章中提供下载的代码根据其在本章中的名称单独命名。

现在你已经对 Arduino 平台和它的功能有了一定的认识，是时候在 Arduino 的世界中探索你的选择了。在本章，你要考查可用的硬件，学习关于编程环境和语言的知识，然后编写你的第一个程序并运行它。一旦你掌握了 Arduino 所能提供的功能，你就要编写第一个程序并让你的 Arduino 闪烁起来！

注意 若要观看一个介绍 Arduino 平台的视频教程，则请访问：
<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头给出的 Wiley 网站上找到这个视频。

探索 Arduino 生态系统

在 Arduino 探险之旅中，你将在项目中依赖 3 个主要的器件：

- Arduino 板卡本身；
- 外部硬件（包括扩展板和手工制作的电路，你将在本书中探索它们）；
- Arduino 集成开发环境，或称 Arduino IDE。

所有的这些系统构件协同工作，使你几乎能利用 Arduino 做任何事情。

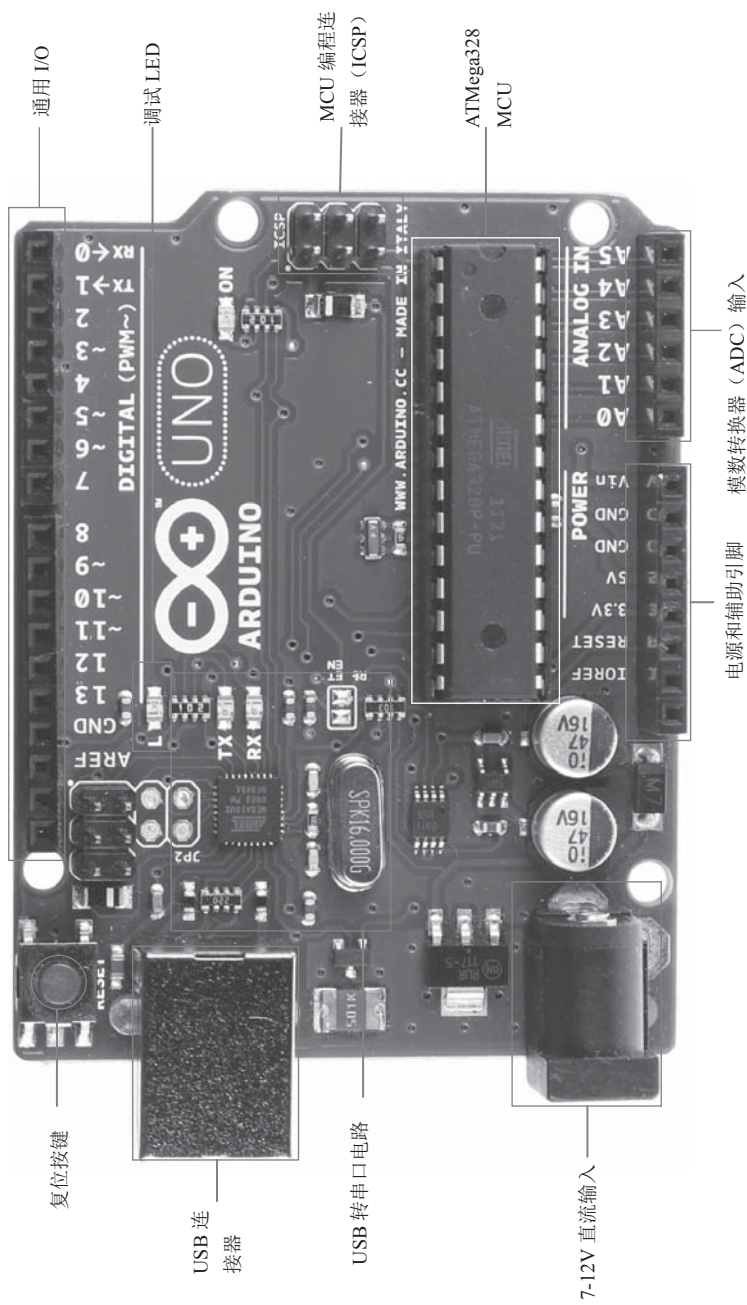
你有很多可供选择的 Arduino 开发板，但本书将主要使用官方的 Arduino 板卡。由于这些板卡都被设计成可以通过相同的 IDE 来编程，所以你通常可以使用任何现代的 Arduino 板卡，不经修改或稍加修改即可完成本书中的项目。然而在必要时，你会看到关于为一些项目使用不同板卡的提示。绝大部分项目使用的是 Arduino Uno。

你首先要探索被内建在每片 Arduino 板卡中的基本功能，然后考查每块现代板卡间的不同之处，这样你就能在为下一个项目选择板卡时做出明智的决定。

Arduino 功能

所有的 Arduino 板卡都具有几个关键的能力和功能。花点时间来考查 Arduino Uno（见图 1-1），这将是你的基本配置。下面是一些你需要关注的关键器件：

- Atmel 微控制器；
- USB 编程/通信接口；
- 稳压器和电源连接；
- 引出的 I/O 引脚；
- 调试、供电和 RX/TX LED；
- 复位按键；
- 在线串行编程器（ICSP）连接器。



引用自：Arduino, www.arduino.cc

图 1-1 Arduino Uno 器件

【Atmel 微控制器】

每片 Arduino 的核心都是一个 Atmel 微控制器单元 (MCU)。绝大部分的 Arduino 板卡, 包括 Arduino Uno, 使用的是一片 AVR ATmega 微控制器。如图 1-1 所示的 Arduino Uno 使用了一片 ATmega 328p。Due 是一个例外, 它使用了一片 ARM Cortex 微控制器。这个微控制器负责保存所有已编译的代码并执行指定的命令。Arduino 编程语言使你能够访问微控制器的外设, 包括模数转换器 (ADC)、通用输入/输出 (I/O) 引脚、通信总线 (包括 I²C 和 SPI) 及串口。所有这些有用的功能都从微控制器上那些微小的引脚引出到了 Arduino 上更容易使用的母头连接器上, 你可以将导线或者扩展板插入其中。一个 16MHz 的陶瓷谐振器连接到了 ATmega 的时钟引脚, 作为所有程序指令执行的参考。你可以使用复位按键来重启你的程序。绝大部分 Arduino 板卡带有一个已经连接到 13 号引脚的调试 LED, 它使你能够不连接额外的电路而运行你的第一个程序 (让一个 LED 闪烁)。

【编程接口】

在通常情况下, ATmega 微控制器的程序是用 C 语言或者汇编语言编写的, 然后利用一个专用的编程器 (见图 1-2) 通过 ICSP 接口进行编程。也许 Arduino 最重要的特性就是它可以方便地通过 USB 接口编程, 而不需要使用一个单独的编程器。这个功能是由 Arduino 启动引导程序 (bootloader) 实现的。这个启动引导程序在工厂里 (使用 ICSP 接口) 预先装载到 ATmega 中, 它使你能够通过串行 USART (通用同步/异步收发器) 来将程序装载到 Arduino, 而不需要使用单独的编程器。(你可以在 “Arduino 启动引导程序和固件设置” 栏目中进一步了解启动引导程序的工作原理。)

在 Arduino Uno 和 Mega 2560 上, 还有第二个微控制器 (取决于不同的版本, 有可能是 ATmega 16U2 或 8U2) 作为 USB 电缆和主控制器的串行 USART 引脚间的接口。使用 ATmega 32U4 作为主控制器的 Arduino Leonardo 已经内建了 USB 接口, 因此就不需要第二个微控制器了。在更老的 Arduino 板卡上, 则使用了一片 FTDI 出品的 USB 转串口芯片, 作为 ATmega 的串行 USART 端口和 USB 口之间的接口。



引用自：©2013 Atmel Corporation. 保留所有权利

图 1-2 AVR ISP MKII 编程器

【通用 I/O 和模数转换器（ADC）】

在搭建项目期间，在 Arduino 上你最关注的部分就是通用 I/O 和模数转换器引脚。所有的这些引脚都可以通过你编写的程序来被独立地寻址。它们都可以用作数字输入、输出。ADC 引脚还可以作为模拟输入端，用来测定 0~5V 的电压（通常来自电阻式传感器）。在这些引脚中，有许多还可以被复用作其他功能，你将会在你之后的项目中去探索。这些特殊的功能包括各种通信接口、串口、脉冲宽度调制输出及外部中断。

【电源】

对于绝大部分项目，你只会用到由 USB 电缆提供的 5V 电源。然而，当你打算让你的项目脱离一台计算机的时候，还有其他供电方法可选。Arduino 可以通过直流（DC）圆孔插座或者 V_{in} 引脚接受 6~20V（推荐 7~12V）的供电。Arduino 内建了 5V 和 3.3V 的稳压器。

- 5V 用于板上的所有逻辑电路。换句话说，在切换一个数字 I/O 引脚时，你实际上是在 5V 和 0V 间切换。
- 3.3V 被引出到一个引脚上，用来适配 3.3V 的扩展板和外部电路。

Arduino 启动引导程序和固件设置

启动引导程序（bootloader）是一段代码，它保存在 Arduino 主 MCU 的程序存储器的保留空间中。通常情况下，AVR 微控制器需要通过一个 ICSP 来编程，它通过串行外设接口（SPI）来与微控制器通信。通过这种方式来编程十分简单，但用户必须要有一个硬件编程器，比如 STK500 或者 AVR ISP MKII

编程器（见图 1-2）。

当你首次启动 Arduino 板卡时，它会进入启动引导程序，然后运行几秒。如果在这段时间里，MCU 的 UART（串口）从 IDE 接收到了编程命令，它就会将你发送给它的程序烧写至 MCU 其余的程序存储器中。如果没有接收到编程命令，它就会开始执行你最近烧写至其余程序存储器中的程序。

当你从 Arduino IDE 发送一个“烧写”指令时，它会命令 USB 转串口芯片（在 Arduino Uno 上是一片 ATmega 16U2 或者 8U2）复位主 MCU，从而强制其进入启动引导程序。然后，你的计算机立即开始发送程序内容，而 MCU 也已经准备好从它的 UART 口（在 USB-串口转换器的协助下）接收程序。

启动引导程序是很棒的，因为它使我们能够不用添加额外的硬件而通过 USB 接口方便地编程。然而，它也有以下两个缺点。

- 首先，它占用了宝贵的程序存储空间。如果你编写了一个复杂的程序，则由启动引导程序所占用的约 2KB 的空间就会变得十分宝贵。
- 其次，使用启动引导程序意味着你的程序总是需要在启动时延迟几秒，来让启动引导程序检测编程请求。

如果你拥有一个编程器（或者另外一个可以被编程从而成为一个编程器的 Arduino），那么你可以从 ATmega 中移除启动引导程序，将编程器连接到 ICSP 插头，然后使用 IDE 中的 File-Upload Using Programmer 命令来为其编程。

Arduino 板卡

本书不可能涵盖所有可用的 Arduino 板卡，因为它们太多了，而且制造商们还在不断地发布具有不同特性的新板卡。下面介绍一些官方 Arduino 板卡的主要特性。

Uno（见图 1-3）是 Arduino 中的旗舰，在这本书中将大量使用这块板卡。它采用了一片 16U2 USB-串口转换器芯片，另有一片 ATmega 328p 作为主 MCU。它有双列直插（DIP）和表面贴装（SMD）两个版本（这决定了 MCU 是否可拆卸）。

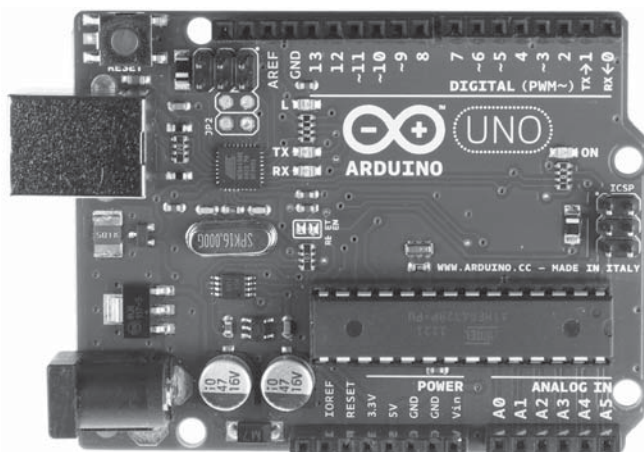
引用自: Arduino, www.arduino.cc

图 1-3 Arduino Uno

Leonardo (见图 1-4) 采用了内建 USB 接口的 32U4 作为主微控制器。因此，它不需要第二个 MCU 来作为串口-USB 转换器。这不但降低了成本，还使得你能够做一些独特的事情，比如模拟游戏摇杆或者键盘，而不仅仅是一个简单的串口设备。你将在第 6 章“USB 和串口通信”中学习如何使用这些特性。

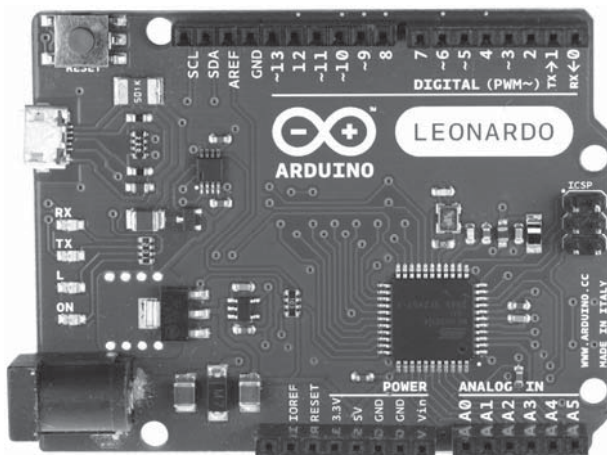
引用自: Arduino, www.arduino.cc

图 1-4 Arduino Leonardo

Mega2560 (见图 1-5) 采用了一片 ATmega 2560 作为主 MCU，它具有 54 个通用 I/O 口，使你能连接更多的设备。Mega 也拥有更多的 ADC 通道及 4 个硬件串口（而不像 Uno 那样只有一个串口）。

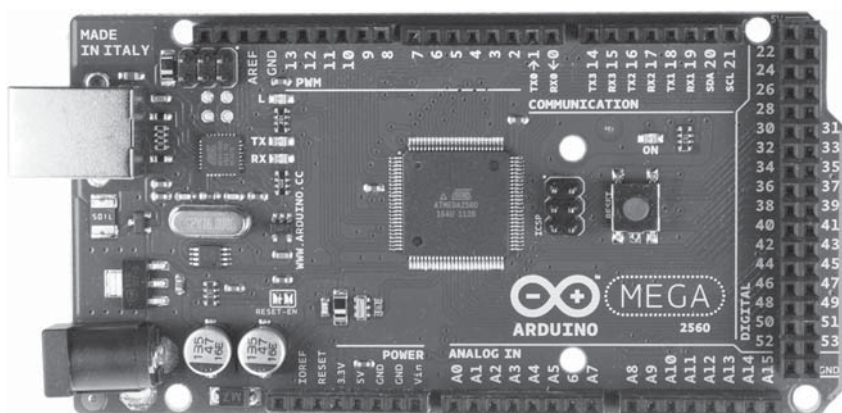
引用自: Arduino, www.arduino.cc

图 1-5 Arduino Mega 2560

与其他采用 8 位 AVR MCU 的 Arduino 变种不同的是, Due (见图 1-6) 使用了一片 32 位的 ARM Cortex-M3 SAM3X MCU。Due 还提供了更高精度的 ADC、可选分辨率的脉冲宽度调制 (PWM)、数模转换器 (DAC)、一个 USB Host 连接器及 84MHz 的时钟频率。

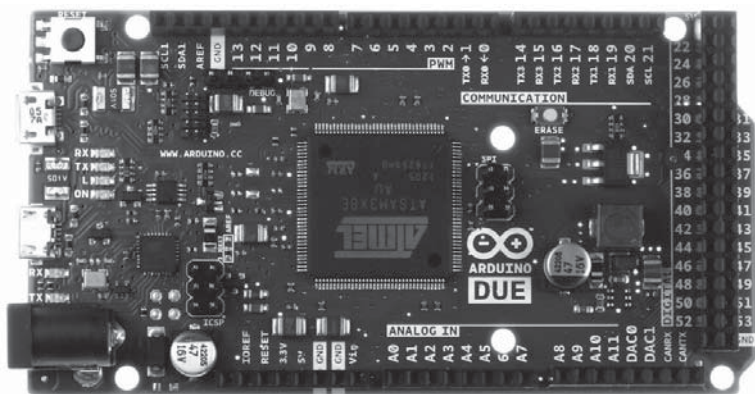
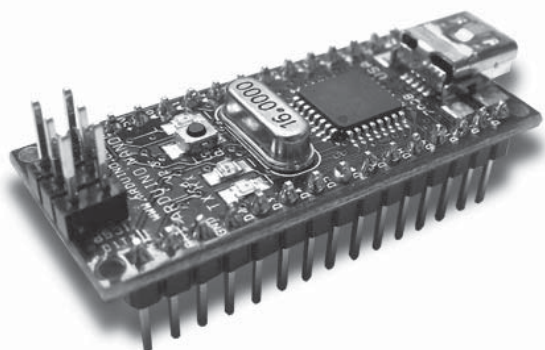
引用自: Arduino, www.arduino.cc

图 1-6 Arduino Due

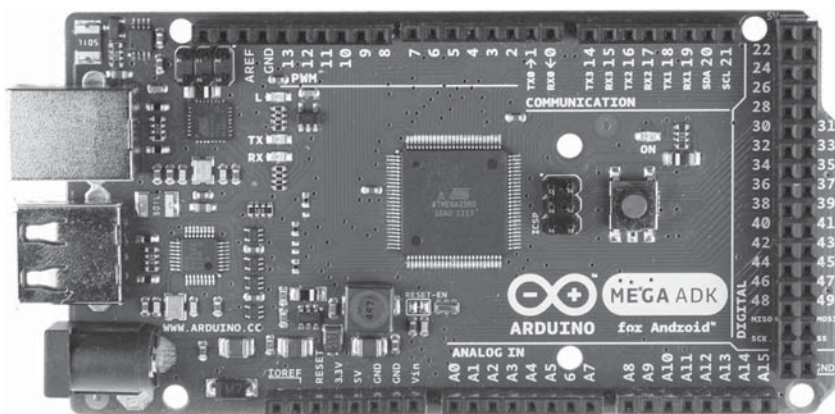
Nano (见图 1-7) 可以直接安装到一块面包板中, 小巧的尺寸使其非常适用于更加精巧的项目中。



引用自: Cooking Hacks,
www.cookinghacks.com

图 1-7 Arduino Nano

Mega ADK (见图 1-8) 与 Mega 2560 非常类似, 区别在于它的 USB Host 功能, 这使其能够连接到 Android 手机, 从而与你编写的手机应用通信。



引用自: Arduino, www.arduino.cc

图 1-8 Arduino Mega ADK

LilyPad (见图 1-9) 十分独特, 因为它可被缝进衣物之中。利用导电缝纫线, 你可以为其连接可缝制的传感器、LED 等。为了控制尺寸, 你需要使用 FTDI 电缆为它编程。

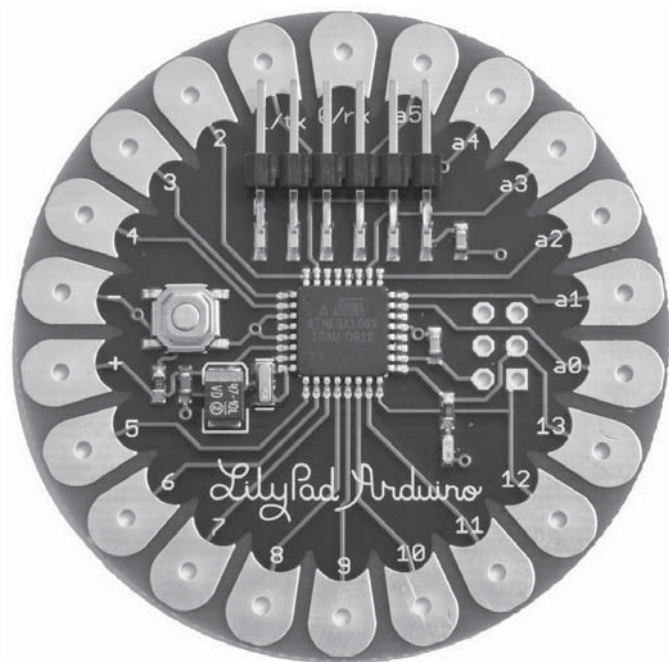
引用自: Arduino, www.arduino.cc

图 1-9 LilyPad Arduino

正如本书前言中所述, Arduino 是一个开源硬件。因此, 你可以找到各种各样的“Arduino 兼容型”设备, 它们都可以与 Arduino IDE 协同工作, 也都可以用来完成本书中的项目。一些比较流行的第三方板卡包括 Seeeduno、adafruit 32U4 接线板及 SparkFun Pro Mini Arduino 板卡等。许多第三方板卡通常是针对特定应用设计的, 并且已经板载了一些额外的功能。比如, ArduPilot 是一块自动飞控板, 用于自主控制的 DIY 四轴飞行器(见图 1-10)。你甚至可以在消费电子产品中找到 Arduino 兼容型电路的身影, 比如 MakerBot Replicator 和 Replicator 2 3D 打印机。

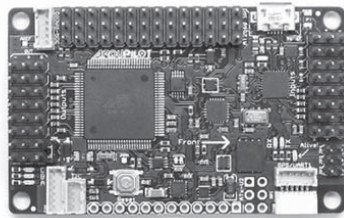
引用自: 3D Robotics, Inc., www.3drobotics.com

图 1-10 四轴飞行器和 ArduPilot Mega 控制器

创建你的第一个程序

现在你已经了解了要在本书中使用的硬件，可以安装软件并运行第一个程序了。首先要将 Arduino 软件下载并安装到你的计算机中。

下载并安装 Arduino IDE

访问 Arduino 网站 www.arduino.cc 并从“Download”页面下载最新版的 IDE（见图 1-11）。

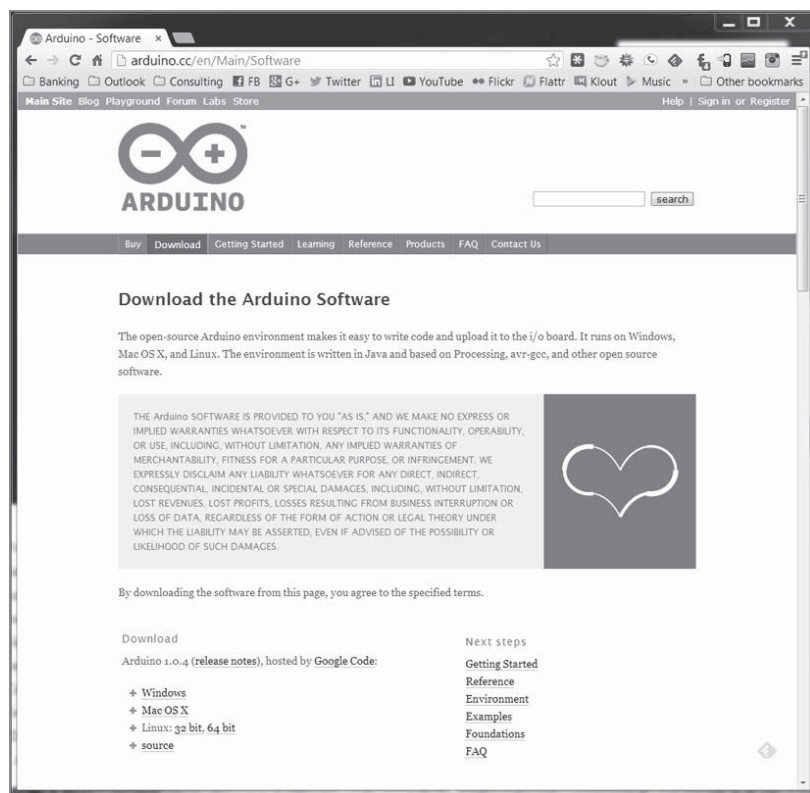


图 1-11 Arduino.cc 下载页面

完成下载之后将其解压，在其中你会找到 Arduino IDE。Windows 下新版的 IDE 还提供了安装包格式，你可以直接下载并运行，而不需要下载一个 ZIP 文件。

运行 IDE 并连接 Arduino

IDE 已经下载完成并可以运行了，你可以将 Arduino 通过 USB 线连接到计算机，如图 1-12 所示。Mac 和安装了 Linux 操作系统的计算机（几乎）可以自动安装驱动程序。

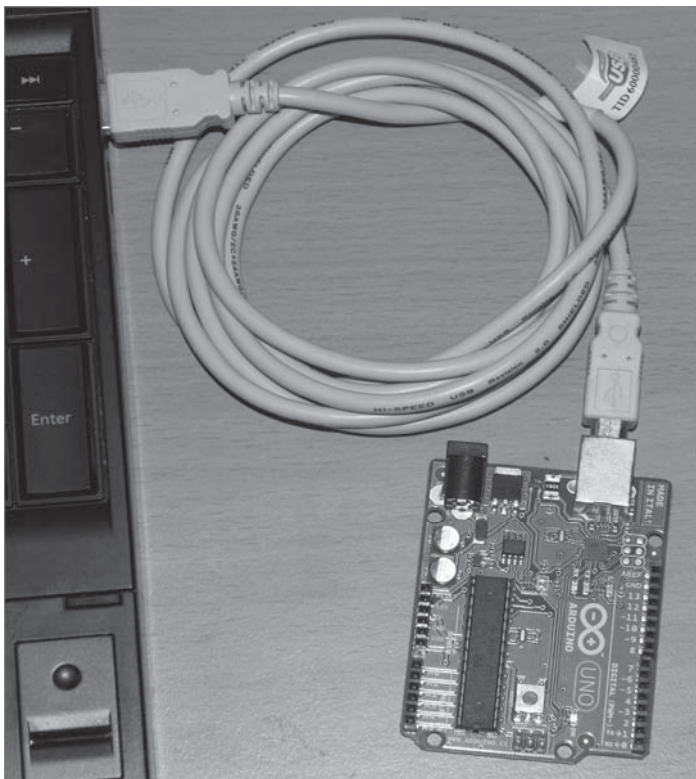


图 1-12 Arduino Uno 通过 USB 连接到了一台计算机

如果你使用的是 OS X，则在第一次插入 Uno 或者 Mega 2560 时，会弹出一个已添加新网络设备的提示。单击“Network Preferences”（网络偏好设置）按钮。在新窗口中，单击“Apply”按钮。尽管这块板卡在网络设备列表中显示为“Not Configured”（未配置），但它已经可以使用了。现在就可以退出“System Preferences”（系统偏好设置）了。

如果在安装了 Windows 操作系统的计算机上使用新版本的 Arduino，则你可能需要安装驱动程序。如果你使用的不是安装了 Windows 操作系统的计算机，则可以跳过下列步骤。如果你使用 Windows 安装包安装了 IDE，则下列步骤已经自动完成了。如果你在安装了 Windows 操作系统的计算机上下载了 ZIP 压缩

包，则需要遵循下列步骤。

在安装了 Windows 操作系统的计算机上，请遵循以下步骤来安装驱动程序（以下步骤改编自 Arduino.cc 网站）。

1. 等待自动安装进程失败。
2. 打开“开始”菜单，用鼠标右键单击“我的电脑”并选择“属性”。
3. 选择“设备管理器”。
4. 在“端口（COM 和 LPT）”项下找到你连接的 Arduino。
5. 用鼠标右键单击它，选择“更新驱动程序软件”。
6. 选择“浏览计算机以查找驱动程序软件”。
7. 从你刚下载的 Arduino IDE 目录中的 driver 目录下选择恰当的驱动程序（不是 FTDI 驱动程序目录）。
8. Windows 会自动完成驱动程序的安装。

现在，运行 Arduino IDE，你就可以将第一个程序烧写至 Arduino 中了。为了确保一切都工作正常，你可以烧写 Blink 示例程序，它会让板载 LED 闪烁。绝大多数 Arduino 都有一个连接到 13 号引脚的 LED。定位到 File-Example-Basic 菜单并单击 Blink 程序，会打开一个新的 IDE 窗口，其中是已经写好的 Blink 程序。首先，你要用这个示例程序为 Arduino 编程，然后分析这个程序，理解其中的重要部分，这样你就可以在第 2 章中开始编写自己的程序了。

在烧写程序之前，你需要告诉 IDE 你将哪种 Arduino 连接到了哪个端口。在 Tools-Board 菜单下选择正确的板卡。在这个示例中使用的是 Uno，但如果你使用的是不同的板卡，则请选择它（假设它也有一个连接到 13 号引脚的 LED）。

在编程之前的最后一步是告诉 IDE 你的板卡连接到了哪个端口。定位到 Tools-Serial Port 菜单并选择恰当的端口。在 Windows 操作系统下，端口会显示为 COM*，其中的“*”是一个表示串口编号的数字。

提示 如果你在计算机上连接了多个串口设备，则可以尝试拔掉你的板卡，看哪个 COM 端口从菜单中消失了，然后将它插回去并选择那个 COM 端口。

在安装了 Linux 操作系统的计算机和 Mac 上，串口会显示为类似于 /dev/tty.usbmodem* 或 /dev/tty.usbserial* 的形式，其中“*”是字母或数字组成的字符串。

现在你终于可以烧入第一个程序了。单击 IDE 左上方的 Upload 按钮 ()。IDE 下方的状态栏会在编译和烧写程序时显示一个进度条。烧写完成后, Arduino 上的黄色 LED 就会每秒闪烁一次。祝贺你! 你刚刚烧写了第一个 Arduino 程序。

分析你的第一个程序

花点时间分析这个 Blink 程序, 就可以让你了解一个 Arduino 程序的基本结构。参见图 1-13, 其中的编号对应下列清单。

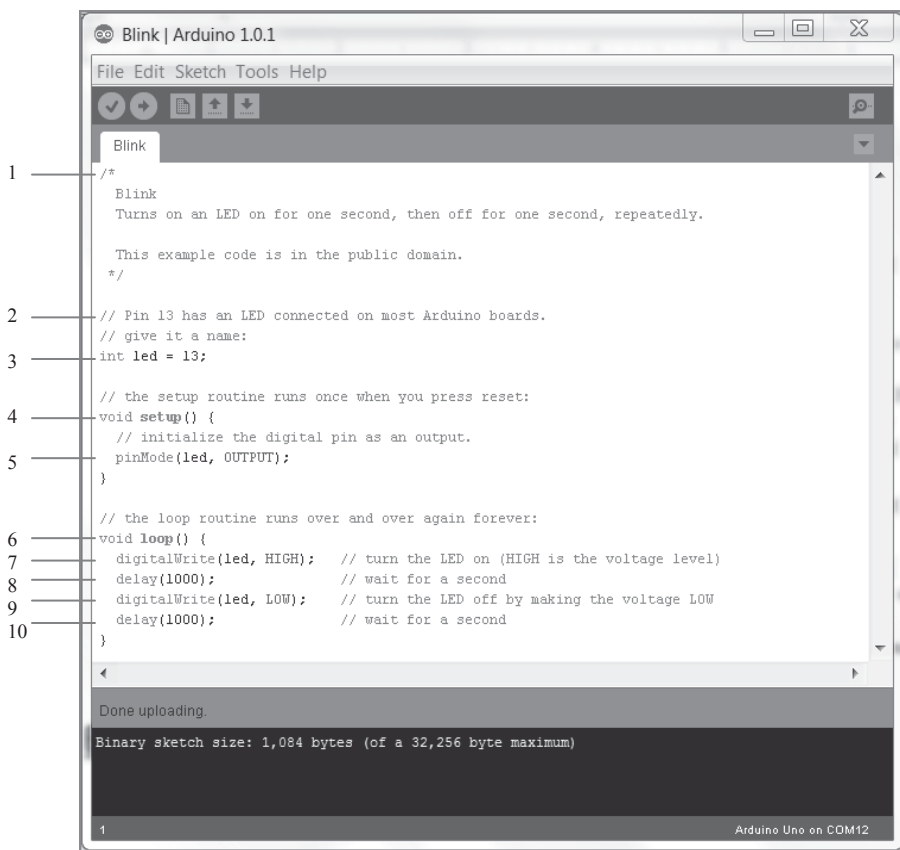


图 1-13 Blink 程序的各个部分

下面逐行讲解了这段代码是如何工作的。

1. 这是一个多行注释。注释对文档化你的代码是非常重要的。所有写在这些符号之间的内容都不会被编译, 甚至根本不会被 Arduino 看到。多行注释以 “/*” 开头并以 “*/” 结尾。多行注释通常在你需要说很多话时使用 (比如这个

程序的描述)。

2. 这是一个单行注释。当你在任何行中插入“//”时，编译器会忽略从这个符号起直到行尾的所有内容。这对标注特定的一行代码是非常好的，在你感觉某些代码可能导致问题时，也可以使用这个符号将其“注释掉”。

3. 这是一个变量声明。变量是在 Arduino 的内存中的一块用于保存信息的区域。变量有不同的类型。在这里，它的类型是 `int`，意味着它可以保存一个整数 (integer)。在这里，一个被称为 `led` 的变量的值被设置为 13，代表与 LED 连接的 Arduino Uno 引脚。在之后的程序中，需要控制 13 号引脚时就可以简单地使用 `led` 来表示。设立变量是很有用的，如果你之后将 LED 连接到了不同的 I/O 引脚，则只需修改一行代码，而其余的代码仍能正常工作。

4. `void setup()` 是两个必须包含在每个 Arduino 程序中的函数之一。函数是一段执行特定任务的代码。包含在 `setup()` 函数花括号间的代码在每次程序启动时都会执行一次。这对一些一次性的设置是很有用的，比如设置引脚方向、初始化通信接口，等等。

5. Arduino 的数字引脚可以工作在输入或输出模式。要配置它们的方向，可使用 `pinMode()` 函数。这个命令接收两个参数。参数用于给出如何操作的命令信息。参数通常位于命令之后的圆括号之中。`pinMode` 的第一个参数用于指定要设置方向的引脚是哪一个。你在之前的程序中定义了 `led` 变量，即通知了这条命令要设置 13 号引脚的方向。第二个参数设置了引脚的方向：`INPUT`（输入）或 `OUTPUT`（输出）。引脚在默认情况下是输入模式，如果你想让其作为输出功能，则需要显式地将其设置为输出模式。因为你想点亮一个 LED，所以必须将 `led` 引脚设置为输出（电流从 I/O 引脚中流出）模式。注意你只需要这样做一次。它在整个的程序中都会保持输出状态，直到你手动将其改为输入。

6. Arduino 程序中第二个必要的函数是 `void loop()`。`loop()` 函数的内容在 Arduino 上电之后就会不断地重复执行。即使你只需要把一些东西在 Arduino 启动时执行一次，也必须包含 `loop()` 函数，但你可以将其留空。

7. `digitalWrite()` 用于设置一个输出引脚的状态。它可以将引脚设置为 5V 或 0V。当将一个 LED 和电阻连接到引脚时，将引脚设置为 5V 就会点亮 LED。（你会在第 2 章中进一步学习这些。）`digitalWrite()` 的第一个参数是你想要操作的引脚。第二个参数是你想要设置的值，可以是 `HIGH(5V)` 或者 `LOW(0V)`。引脚的状态会保持原状，直到它在代码中被改变。

8. `delay()` 函数接收一个参数：延时时间的毫秒数。当调用 `delay()` 时，Arduino 会在指定的时间内停止一切工作。在这个例子中，你将程序延时 1000ms，

即 1s。这就使得 LED 在执行下一条指令前保持原状态 1s。

9. 这里, `digitalWrite()` 通过将引脚状态设置为 LOW 来使 LED 熄灭。

10. 我们再次延时 1s, 将 LED 保持在熄灭状态, 然后重新开始循环来再次点亮它。

这就是整个过程。如果你还没有完全理解代码, 则也不要担心。随着你在之后的章节中接触到更多的代码, 你就会越来越熟练地掌握程序流程并编写自己的代码。

本章小结

在本章你学到了如下内容:

- 组成 Arduino 板卡的所有器件;
- Arduino 的启动引导程序如何使你能够通过 USB 连接烧写 Arduino 固件;
- 各种不同 Arduino 板卡间的区别;
- 如何将 Arduino 连接到你的计算机并安装软件;
- 如何烧写并运行你的第一个程序。

第 2 章

数字输入、输出和脉冲宽度调制

本章所需的器件：

Arduino Uno

小号面包板

跳线

10k Ω 电阻

220 Ω 电阻 (×3)

USB 电缆

按键

5mm 单色 LED

5mm 共阴极 RGB LED

本章的代码和数字化内容

本章的代码、视频及其他数字化内容可以在以下网址找到：
www.exploringarduino.com/content/ch2。

另外，所有的代码都可以在 www.wiley.com/go/exploringarduino 网页的“Download Code”标签中找到。在第 2 章中提供下载的代码根据其在本章中的名称单独命名。

正如你在第 1 章学到的，让一个 LED 闪烁确实很棒，而让 Arduino 微控制器平台变得如此有用的原因则是这个系统可以同时具备输入、输出的能力。通过二者的组合，你的机会几乎是无穷的。举例来说，你可利用一个磁簧开关在门打

开时播放音乐、制作一个电子锁箱或者制作一个乐器。

在本章，你要就要开始学习搭建这种项目所需的技巧。你会探索 Arduino 的数字输入、输出功能，学习上拉和下拉电阻及如何控制数字输出。绝大多数 Arduino 没有模拟输出，但在很多情境下都可以通过脉冲宽度调制技术来模拟它。你会在本章学习如何生成脉冲宽度调制信号。你还将学到如何为数字按键消除抖动，这是一个读取用户输入时的关键技巧。完成本章后，你将能够制作一个可控的 RGB（红、绿、蓝）LED 夜灯并为其编程。

注意 你可以跟随一个视频来学习数字输入、输出、抖动消除及脉冲宽度调制（PWM）：<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头处的 Wiley 网站上找到这个视频。

如果你想学习更多本章相关的电气工程的基础知识，则请观看这个视频：<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头处的 Wiley 网站上找到这个视频。

数字输出

在第 1 章“让你的 Arduino 闪烁”中，你学习了如何让一个 LED 闪烁。在本章，你将进一步探索 Arduino 的数字输出功能，包括以下话题：

- 将引脚设为输出模式；
- 连接外部元件；
- 新的程序设计概念，包括 for 循环和常量；
- 数字和模拟输出对比以及脉冲宽度调制技术。

连接一个 LED 并使用面包板

在第 1 章中，你学习了如何让板载的 LED 闪烁，但这有什么意思呢？现在我们拿出面包板并将一个外部 LED 连接到 Arduino 的 9 号引脚。添加这个外部 LED 将会成为一块垫脚石，帮助你理解如何在之后的章节中连接更加复杂的外部电路。更重要的是，9 号引脚带有 PWM 功能，本章将能用它完成模拟输出的示例程序。

【使用面包板】

理解面包板的工作原理是十分重要的，这样你才能在本书的项目中有效地使

用它。面包板是一个简单的原型工具，用它能够连接简单的电路，而不需要将元器件焊接在定制的印刷电路板上。首先，考虑板上横向的蓝线和红线。这些彩线附近的引脚用于电源和接地总线。所有红线旁的引脚都是电气相连的，通常用于供电。在本书的项目乃至绝大部分 Arduino 项目中，这条线通常都是 5V。所有蓝线旁的引脚也都是电气相连的，通常用于接地总线。所有纵向对齐的引脚都是按列电气相连的，在板子的中间还有一条分割槽，以便在面包板上安装集成电路。图 2-1 突出显示了这些引脚是如何电气相连的，所有的粗线表示这些孔是相连的。

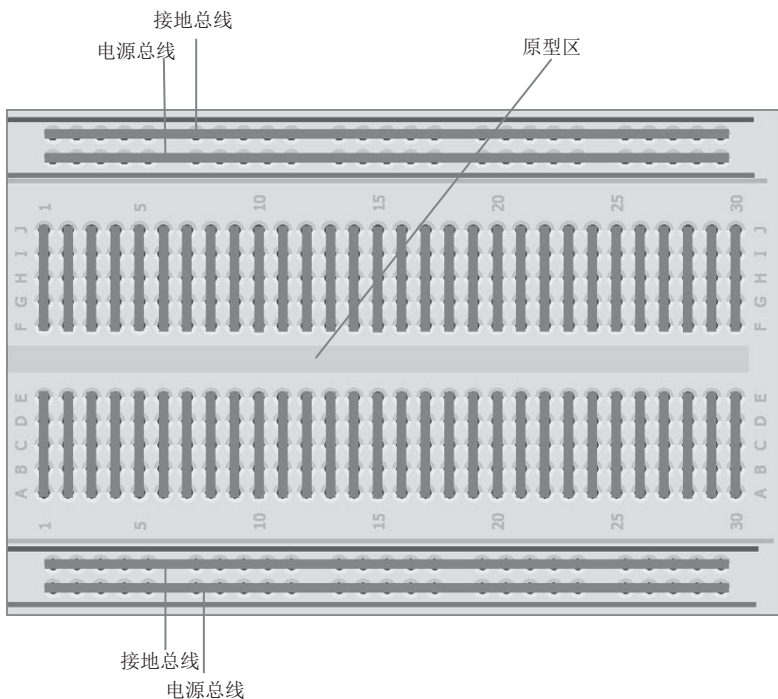


图 2-1 面包板的电气连接

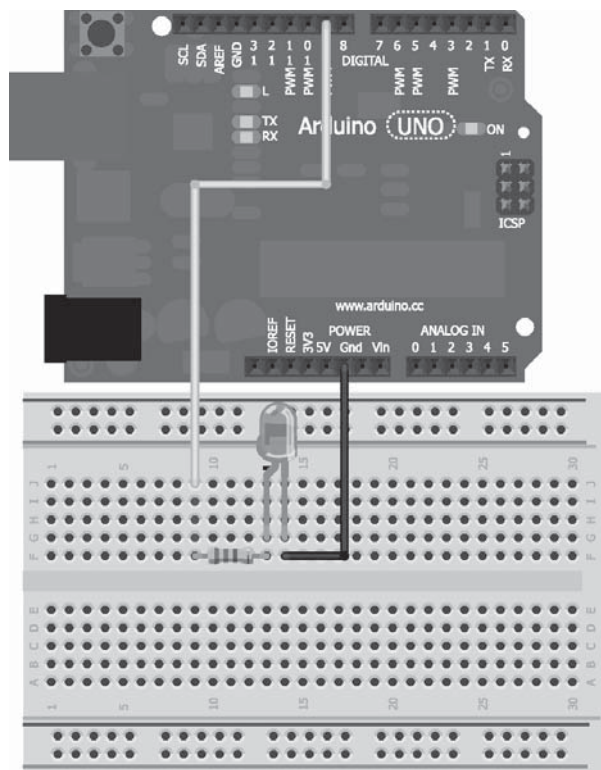
【连接 LED】

在本书的项目中，LED 几乎是最常用的部件之一了。LED 是有极性的，换句话说，它的连接方式很重要。正极引脚被称为阳极，而负极引脚被称为阴极。如果你观察 LED 透明的上部，通常会在外壳边缘处有一个扁平端，那就是阴极。另一种分辨阴极和阳极的方式是看引脚，较短的引脚是阴极。

你可能已经知道，LED 是 Light-Emitting Diode（发光二极管）的缩写。跟

所有的二极管一样，LED 只允许电流单向通过——从阳极流向阴极。因为电流从正极流向负极，所以 LED 的阳极应该连接到电流源（在这里是一个 5V 的数字信号），而阴极应该接地。电阻可以串联在 LED 的任意一端。电阻是没有极性的，所以你不需要担心它的方向。

你要将 LED 通过一个串联的电阻连接到 9 号引脚。LED 必须串联一个电阻进行限流。电阻值越大，对电流的阻碍作用就越大，LED 的光就越暗。在这里，你使用的是一个 220Ω 的电阻。按照如图 2-2 所示的方式对其进行连线。



该图片是使用Fritzing制作的

图 2-2 Arduino Uno 连接了一个 LED

欧姆定律和功率方程

对一位电子工程师来说，其需要了解的最重要的公式就是欧姆定律。欧姆定律揭示了电路中的电压（单位是“伏特”）和电流（单位是“安培”）间的关系，也就是电阻（单位是“欧姆”或“ Ω ”）。电路是一个闭合的回路，由电源（比如 9V 电池）和负载（消耗能量的东西，比如 LED）构成。在深入了解这个定律之前，首先要理解每个术语的含义，至少要基本理解如下内容。

- 电压表示两点间的电势差。
- 电流从高电势点流向低电势点。你可以将电流想象成水流，将电压想象成地势。水（电流）总是从地势高的地方（高电势点）流向地势低的地方（大地，低电势点）。电流，就像河流中的水，总是随着阻碍小的路径流动。
- 电阻在这个类比中，表示了电流流动的难易程度。当水（电流）流过一个狭窄的管道时，在相同时间内流过的水量就比大管道的少。狭窄的管道就相当于高电阻值，水流过它时更加困难。宽阔的管道则相当于低电阻值（就像导线），电流可以自如地流过它。

欧姆定律定义如下：

$$U = IR$$

其中 U 是电压差的伏特值， I 是电流的安培值，而 R 就是电阻的欧姆值。

在电路中，所有的电压都会被分配，每个器件的电阻都会降低电压。根据这个规律，就可以利用上述公式方便地计算出与一个 LED 匹配的电阻值。LED 具有一个预定义的压降并工作于一个特定的电流。在一定范围内，通过 LED 的电流越大，LED 的发光强度也就越大。大多数常用的 LED，通过电流最大为 20 毫安（1 毫安是 1/1000 安培，通常缩写为 mA）。LED 的压降在其数据手册中定义，常见的数值是 2V 左右。请参考如图 2-3 所示的 LED 电路。



该图片是使用Eagle制作的

图 2-3 简单的 LED 电路

你可以利用欧姆定律来计算这个电路中的电阻值。假设这是一个标准的 LED，其正向电流为 20mA，压降为 2V。由于电源电压为 5V 并最终接地，所以整个电路的压降应为 5V。由于 LED 有 2V 的压降，则电阻必须具有 3V 的压降。我们已经知道大约有 20mA 的电流流过这些元件，你就可以通过求解 R 值来得到电阻值：

$$R = U/I$$

其中 $U = 3V$, $I = 20mA$ 。

求解 R , $R = 3V / 0.02A = 150\Omega$ 。因此, 在电阻值为 150Ω 时, $20mA$ 电流同时流过电阻和 LED。当增加电阻值时, 通过的电流更小。 220Ω 要略高于 150Ω , 但仍然可以使 LED 的光足够明亮, 而且这是一个十分常用的电阻阻值。

另一个需要牢记的公式是功率方程。功率方程可以告诉你一个给定的阻性器件消耗的功率是多少(单位是“瓦特”)。增加功率意味着增加散热, 所以元器件通常都有一个最大的功率值。你要确保不超过这个最大的功率值, 否则可能导致过热。常见的电阻功率值是 $1/8$ 瓦特(瓦特写为 W , 毫瓦写为 mW)。功率方程如下:

$$P = UI$$

其中 P 是功率(单位是“瓦特”), 而 I 和 U 仍然定义为电流和电压。

对于先前规定为 $3V$ 压降、 $20mA$ 的电阻来说, $P = 3V \times 0.02A = 60mW$, 远低于电阻的额定功率 $1/8W$ ($125mW$)。因此, 你不需要担心电阻过热, 它正处于其极限工作条件以内。

数字输出端口编程

在默认情况下, 所有的 Arduino 引脚都被设置为输入。如果想将一个引脚设置为输出, 则需要先告诉 Arduino 这个引脚该如何配置。在 Arduino 编程语言中, 程序需要两个部分: `setup()` 函数和 `loop()` 函数。

正如你在第 1 章中学到的, `setup()` 函数在程序开始时执行一次, 而 `loop()` 函数则会反复执行。由于需要指定每个引脚工作在输入还是输出模式, 所以在 `setup()` 函数中定义所有引脚是输入还是输出是很常见的做法。下面你要编写一个简单的程序, 在程序启动时将 9 号引脚设置为输出并将其开启。

要编写这个程序, 需要使用 `pinMode()` 命令来设置 9 号引脚的方向, 然后使用 `digitalWrite()` 使其输出高电平 ($5V$), 见代码清单 2-1。

代码清单 2-1: 点亮一个 LED——led.ino

```
const int LED=9;           //将 LED 定义在 9 号引脚
void setup()
{
    pinMode (LED, OUTPUT);   //将 LED 引脚设置为输出
    digitalWrite(LED, HIGH); //将 LED 引脚设置为高电平
}

void loop()
{
    //在 loop 中我们什么也不做
}
```

将这个程序烧写至你的 Arduino，接线图如图 2-2 所示。注意，我在定义引脚整数变量之前使用了 `const` 关键字。在通常情况下，你可以使用变量来保存程序执行期间可能发生改变数值。将 `const` 放在变量声明前面，即告诉编译器这个变量是“只读的”，而且不会在程序执行期间改变。在程序中所有 LED 的实例都会在其被调用时被 9 所“替换”。在定义不会改变的值时，推荐使用 `const` 修饰符。在本章稍后的示例中，还将定义会在程序执行期间改变的非恒定变量。

你必须为声明的变量指定类型。在上述示例中，它是一个整数（引脚编号总是整数），所以你应该将其设置成整数。你可以轻而易举地修改这个程序，像在第 1 章中那样将 `digitalWrite()` 移到 `loop` 中并添加一些延迟。试着修改延时值来产生不同的闪烁频率。

使用 for 循环

在修改变量值和调节程序参数时会频繁地用到循环。在刚刚编写的那个程序中，你可以实现一个 `for` 循环，看看不同的闪烁频率如何影响系统的运转。利用一个 `for` 循环来尝试不同的数值，你就可以可视化不同的闪烁频率。代码清单 2-2 中的代码就完成了这项工作。

代码清单 2-2：闪烁频率可变的 LED——`blink.ino`

```
const int LED=9;           //将 LED 定义在 9 号引脚
void setup()
{
    pinMode (LED, OUTPUT); //将 LED 引脚设置为输出
}

void loop()
{
    for (int i=100; i<=1000; i=i+100)
    {
        digitalWrite(LED, HIGH);
        delay(i);
        digitalWrite(LED, LOW);
        delay(i);
    }
}
```

编译上述代码并将其烧入你的 Arduino。发生了什么？我们花点时间讲解一下 `for` 循环，你就能理解它的工作原理。`for` 循环的声明总是包含三部分，由分号分隔。

- 第一部分设置用于循环的索引变量。在这个示例中，索引变量是 `i`，它的初始值设置为 100。

- 第二部分指定了循环何时停止。当这个条件为真时，循环内的程序就会反复执行。`<=`表示小于或等于。因此，对于这个循环，在变量 `i` 仍小于或等于 1000 时，程序就会一直执行。
- 第三部分指定每次循环执行后，应该对索引变量执行什么操作。在这个示例中，设置为 `i` 的当前值加 100。

为了更好地理解这些概念，请考虑这个 `for` 循环前两次执行时的具体操作。

1. `i` 等于 100。
2. LED 被设置为高电平，然后保持高电平 100ms，也就是 `i` 的当前值不变。
3. LED 被设置为低电平，然后保持低电平 100ms，也就是 `i` 的当前值不变。
4. 在循环的最后，`i` 增加 100，所以它现在是 200。
5. 200 小于或等于 1000，所以循环再次执行。
6. LED 被设置为高电平，然后保持高电平 200ms，也就是 `i` 的当前值不变。
7. LED 被设置为低电平，然后保持低电平 200ms，也就是 `i` 的当前值不变。
8. 在循环的最后，`i` 增加 100，所以它现在是 300。
9. 这个过程会反复执行，直到 `i` 超过 1000，然后外部的 `loop()` 函数会重复执行，将 `i` 的数值重新设置为 100 并重新开始整个过程。

现在你已经从 Arduino 产生了数字输出，下面将学习使用 PWM 从 Arduino 的 I/O 引脚上产生模拟输出。

analogWrite()与脉冲宽度调制

你已经掌握了引脚的数字输出，可以让 LED 闪烁、控制继电器，以及让电动机以恒定速度转动。但如果你想输出除了 0V 和 5V 之外的电压又该怎么办呢？好吧，没办法——除非使用 Due 上的数模转换器（DAC）引脚或者使用一片外部 DAC 芯片。

不过，你可以使用脉冲宽度调制（PWM）技术来产生一个十分近似于模拟信号的输出。在每片 Arduino 上都有一些引脚可以使用 `analogWrite()` 命令来产生 PWM 信号，当与特定的外设一起使用时，这个信号就可以模拟一个纯模拟信号。这些引脚在板上都有一个“~”标记。在 Arduino Uno 上，3、5、6、9、10 和 11 号引脚是 PWM 引脚。如果你使用的是 Uno，则可以继续使用图 2-1 中的电路来测试 `analogWrite()` 命令对 LED 的操作。不难想象，如果降低电阻

上的压降，LED 的光就会变暗，因为流过它的电流变得更小。这也正是通过 `analogWrite()` 命令进行的 PWM 操作所完成的事情。`analogWrite()` 命令接收两个参数：要控制的引脚和要写入的数值。

PWM 的输出是一个 8 位的值。换句话说，你可以写入的数值范围为 $0 \sim 2^8 - 1$ ，或者说 $0 \sim 255$ 。下面用一个跟之前类似的 `for` 循环结构来循环改变 LED 的亮度值（见代码清单 2-3）。

代码清单 2-3：LED 呼吸灯程序——`fade.ino`

```
const int LED=9;           //将 LED 定义在 9 号引脚
void setup()
{
    pinMode (LED, OUTPUT);  //将 LED 引脚设置为输出
}

void loop()
{
    for (int i=0; i<256; i++)
    {
        analogWrite(LED, i);
        delay(10);
    }
    for (int i=255; i>=0; i--)
    {
        analogWrite(LED, i);
        delay(10);
    }
}
```

运行这段代码时 LED 会怎么样呢？你应该观察到 LED 从熄灭状态渐变到发光状态，然后从发光状态慢慢熄灭。当然，由于这就是主循环做的事情中，这个模式会一直循环。请注意这个 `for` 循环的几处不同。在第一个循环中，`i++` 是一个简化写法，表示 `i=i+1`。类似地，`i--` 的功能等价于 `i=i-1`。第一个 `for` 循环让 LED 逐渐点亮，而第二个循环让它逐渐熄灭。

PWM 控制可以在很多情况下模拟纯模拟量控制，但它在确实需要模拟信号时不见得总能用。举例来说，PWM 在驱动直流（DC）电动机并调节其速度时是十分有效的（你会在之后的章节中体验它），但它在驱动扬声器时却不好用，除非你再增加一些额外的电路。我们花点时间考查一下 PWM 究竟是如何工作的。考虑如图 2-4 所示的曲线图。

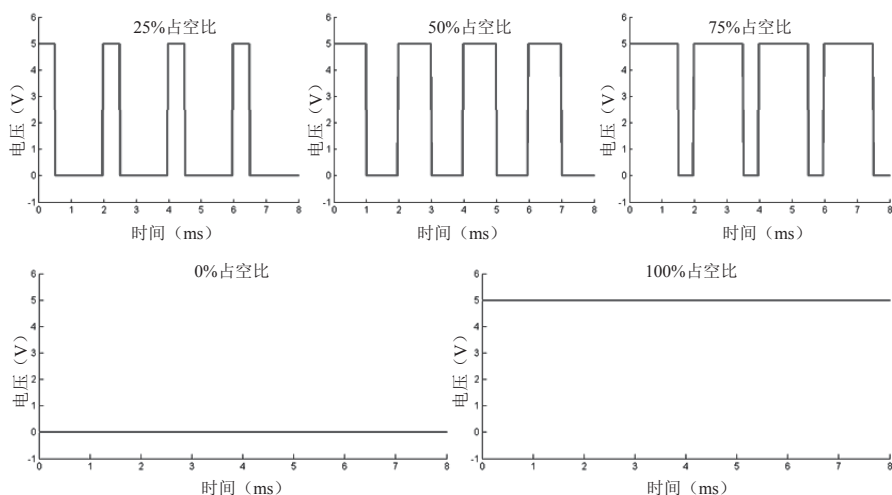


图 2-4 具有不同占空比的 PWM 信号

PWM 的工作原理是调制一串方波（只具有开和关两种状态的信号）的占空比。占空比指的是在一串方波中高电平的时间所占的百分比。你最熟悉的也许是占空比为 50% 的方波——它们有一半的时间是高电平，另一半的时间为低电平。

`analogWrite()` 命令根据传递给它的值来设置一串方波的占空比：

- 为 `analogWrite()` 写入数值 0，表示输出占空比为 0 的一串方波（总是低电平）；
- 写入 255，表示输出占空比为 100% 的一串方波（总是高电平）；
- 写入 127，表示输出占空比为 50% 的一串方波（一半时间为高电平，一半时间为低电平）。

图 2-4 还展示了占空比为 25% 的信号，它有 25% 的时间是高电平，有 75% 的时间为低电平。这串方波的频率在 Arduino 上是 490Hz。换句话说，这个信号每秒在高电平（5V）和低电平（0V）间切换约 490 次。

所以，你并没有改变输送到 LED 的电压，为何又能在降低占空比时让 LED 变暗呢？实际上是你的眼睛捉弄了你！如果 LED 每 1ms 就开关一次（也就是占空比为 50% 的情况），它看起来就是近乎一半的亮度，这是因为它闪烁的速度超过了人眼能察觉的速度。因此，大脑实际上是平均了这个信号，并欺骗你相信这个 LED 只有一半的亮度。

读取数字输入

现在让我们了解另一项技术。你已经成功地产生了数字和模拟输出。下一步是读取数字输入，比如开关和按键，这样就能与你的项目进行实时交互了。在这一节中，你要学习读取输入，实现上拉和下拉电阻及用软件消除按键抖动。

读取带有下拉电阻的数字输入

你应该从修改根据图 2-1 制作的第一个电路开始。根据图 2-5，添加一个按键，以及一个连接到数字输入引脚的下拉电阻。

提示 确保已将面包板的供电和接地总线连接到了 Arduino。这样你就可以在面包板上使用多个设备了，这迟早会有用的。

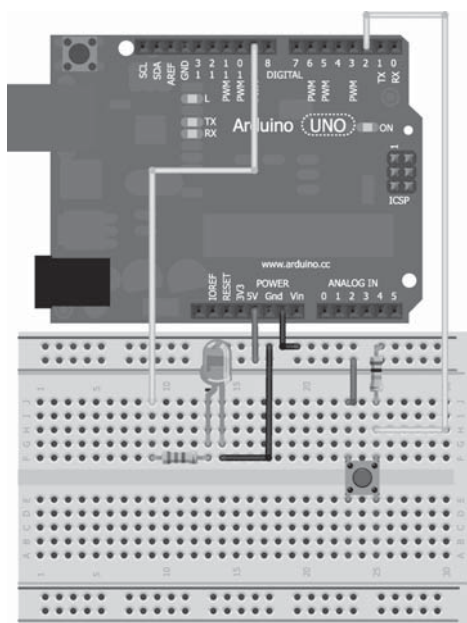
在你编写代码读取按键之前，理解下拉电阻在这个电路中的重要性是十分必要的。几乎所有的数字输入都要使用上拉或下拉电阻来设置输入引脚的“默认状态”。想象一下在图 2-5 所示的电路没有 $10\text{k}\Omega$ 电阻的情境，按键按下时引脚显然会读取到高电平值。

但是，按键没有按下时会发生什么呢？在那种情形下，你要读取的输入引脚其实没有连接到任何东西——这个输入引脚被称为“悬空”。而由于这个引脚没有实际地连接到 0V 或者 5V ，读取它时会导致意料之外的结果，因为附近引脚的电气噪声会导致其值在高电平和低电平间来回波动。要解决这个问题，就需要像图 2-5 中那样安装一个下拉电阻。

现在，考虑一下在存在下拉电阻的情况下键未按下会发生什么：输入引脚会通过一个 $10\text{k}\Omega$ 的电阻接地。尽管电阻会限制电流，但仍有足够的电流确保输入引脚会读到逻辑低电平。 $10\text{k}\Omega$ 是一个相当常用的下拉电阻值。更大的阻值被称为弱下拉，因为其下拉作用很容易被克服，而更小的阻值被称为强下拉，因为它们更容易让更多的电流通过它们接地。当按键被按下时，输入引脚直接通过按键连接到 5V 。

现在，电流有以下两个选择：

- 通过一个几乎零阻抗的路径到 5V ；
- 通过一个高阻抗的路径接地。



该图片是使用Fritzing制作的

图 2-5 将按键和 LED 连接到 Arduino

回顾一下之前讲解的欧姆定律和功率方程，它们指出电流总是沿着电路中阻抗最小的路径流动的。在这种情境下，绝大部分电流会流过按键，从而在输入引脚产生一个高电平，因为这条路径电阻最小。

注意 这个示例使用了下拉电阻，但你也可以使用上拉电阻，也就是将电阻连接到 5V 而非接地，然后将按键的另一端接地。在这种设定下，按键未按下时输入引脚读到的是逻辑高电平，而按键按下时读到的是逻辑低电平。

上拉和下拉电阻是十分重要的，因为它们确保了按键按下时不会造成 5V 和地间短路，同时还确保了输入引脚永远不会处于悬空状态。

现在就可以为这个电路编写程序了！在第一个示例中，你只需让 LED 在按键按住时保持点亮状态，而在按键松开时保持熄灭状态即可（见代码清单 2-4）。

代码清单 2-4：利用按键进行简单的 LED 控制——led_button.ino

```
const int LED=9;           //LED 连接在 9 号引脚
const int BUTTON=2;        //按键连接在 2 号引脚

void setup()
{
```

```
pinMode (LED, OUTPUT);      //将 LED 引脚设置为输出
pinMode (BUTTON, INPUT);    //将按键引脚设置为输入（不必需）
}

void loop()
{
  if (digitalRead(BUTTON) == LOW)
  {
    digitalWrite(LED, LOW);
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
}
```

注意，这段代码出现了一些新的概念，包括 `digitalRead` 和 `if/else` 语句。按键引脚添加了新的 `const int` 语句。而且，这段代码还在 `setup()` 函数中将按键引脚定义为输入。尽管这不是必需的，因为引脚在默认情况下就是输入状态，这是为了代码的完整性而添加的。`digitalRead()` 读取一个输入引脚的值，在这里它会读取 `BUTTON` 引脚的值。如果按键被按下，则 `digitalRead()` 返回 `HIGH` 或 `1`；如果没有被按下，则返回 `LOW` 或 `0`。将其放入 `if()` 语句中，你就可以检查引脚的状态，以判断其是否满足已声明的条件。在这个 `if()` 语句中，检查 `digitalRead()` 的返回值是不是 `LOW`。其中 `==` 是一个比较运算符，用于测试第一项（`digitalRead()`）是否等于第二项（`LOW`）。如果为真（也就是说，按键没有被按下），则花括号中的代码就会被执行，将 `LED` 设置为 `LOW`。如果不为真（按键被按下），则 `else` 语句就会被执行，从而将 `LED` 设置为 `HIGH`。

就是这样！用这段代码为你的电路编程，然后确认其工作状态是否符合预期。

使用“有弹性的”按键

还记得上次通过一直按住按键来让灯保持点亮状态是什么时候吗？也许从没有过。按一下按键来开灯，再按一下来关灯，这样似乎要更合理一些。这样，你就不用一直按住按键来让灯保持点亮状态。不幸的是，这可能不像你最初想得那么简单。你不能只是检测开关的值是否从低电平变成高电平，你还需要担心一种被称为“开关抖动”的现象。

按键是基于弹簧-阻尼系统的机械部件。换句话说，当你按下一个按键时，你读取到的信号并非只是从低到高，它还会在这两个状态间来回跳动，持续几毫秒才会最终稳定。如图 2-6 所示是期望的按键的反应和实际使用示波器可能探测到的反应（尽管这张图是使用 `MATLAB` 脚本生成的）。

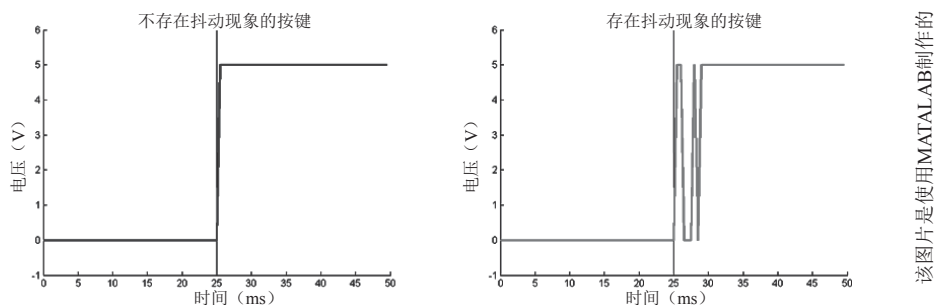


图 2-6 按键的抖动现象

按键在第 25ms 被按下，你也许期望按键的状态会像如图 2-6 左图所示的那样立即被读到逻辑高电平。然而，按键实际上会反复抖动多次才最终稳定，如图 2-6 右图所示。

如果你知道了开关会存在这种现象，那么用软件处理它就相对容易了。接下来，你要写一个开关消抖动软件，它会发现按键状态的变化，等待抖动结束并再次读取开关的状态。这个程序的逻辑可以表示如下。

1. 存储上一个按键状态和当前的按键状态（初始化为 LOW）。
2. 读取当前按键状态。
3. 如果当前按键状态与上一个按键状态不同，则等待 5ms，因为按键的状态一定发生了改变。
4. 5ms 之后，再次读取按键状态，并将其设置为当前按键状态。
5. 如果上一个按键状态是低电平，而当前按键状态是高电平，则翻转 LED 的状态。
6. 将当前按键状态值赋值给上一个按键状态。
7. 返回第 2 步。

这是一个很好的首次探索使用函数的机会。函数是一段代码，它能接收输入参数，基于这些参数执行代码，然后视需要返回一个结果。不知不觉中，你已经在程序中使用了一些预定义的函数。举例来说，`digitalWrite()` 就是一个函数，它接收引脚和状态，然后将那个状态写入到给定的引脚。为了简化程序，你可以定义自己的函数，来封装一些会反复用到的操作。

在程序流程中（前面所列的步骤）需要有一系列步骤来改变变量的值。因为你需要反复地为按键值消抖动，所以将消抖动的步骤定义成一个可以直接调用的函数是十分有用的。这个函数接收先前的按键状态作为输入，并输出当前消抖动

的按键状态。下列程序实现了上述步骤，并在每次按键按下时切换 LED 的状态。你可以使用跟上一个示例相同的电路。尝试将其烧写至 Arduino 并观察它是如何工作的（见代码清单 2-5）。

代码清单 2-5：消抖动的按键切换——debounce.ino

```
const int LED=9;           //LED 连接在 9 号引脚
const int BUTTON=2;        //按键连接在 2 号引脚
boolean lastButton = LOW;  //保存前一个按键状态的变量
boolean currentButton = LOW; //保存当前按键状态的变量
boolean ledOn = false;     //LED 的当前状态（开/关）

void setup()
{
    pinMode (LED, OUTPUT);    //将 LED 引脚设置为输出
    pinMode (BUTTON, INPUT);  //将按键引脚设置为输入
}

/*
 * 消抖动函数
 * 传入前一个按键状态，返回当前消抖动的按键状态
 */
boolean debounce(boolean last)
{
    boolean current = digitalRead(BUTTON);    //读取按键状态
    if (last != current)                      //如果它不同于之前的状态
    {
        delay(5);                            //等待 5ms
        current = digitalRead(BUTTON);        //再次读取
    }

    return current;                          //返回当前值
}

void loop()
{
    currentButton = debounce(lastButton);      //读取消抖动状态
    if (lastButton == LOW && currentButton == HIGH) //如果按键被按下
    {
        ledOn = !ledOn;                      //翻转 LED 值
    }
    lastButton = currentButton;                //重置按键值

    digitalWrite(LED, ledOn);                 //改变 LED 状态
}
```

下面详细讲解代码清单 2-5 中的代码。首先为连接按键和 LED 的引脚定义常量，然后声明了 3 个布尔型变量。如果在变量声明前面不加 const 修饰符，

那么就指示了这个变量的值在程序中可以改变。在程序的最开始定义这些值，就将它们定义为了全局变量，全局变量可以在程序中的任何一个函数中使用和修改。这 3 个声明在程序顶部的布尔变量还进行了初始化，意味着它们被设置了一个初始值（分别是 LOW、LOW 和 false）。在之后的程序中，这些变量的值可以通过赋值运算符（单个等号“=”）来修改。

考虑定义在上述代码中的函数：`boolean debounce(boolean last)`。这个函数接收一个被称为 `last` 的布尔型（一种只有两个状态的数据类型：真/假、高/低、开/关、1/0）输入变量，并返回一个布尔值，表示当前消抖动的引脚值。这个函数将当前的按键状态与作为参数传入的前一个（`last`）按键状态做比较。其中“`!=`”表示“不等于”，在 `if` 语句中用它来比较当前的和先前的按键状态值。如果它们不同，则说明按键已被按下，而 `if` 语句将执行其内容。`if` 语句等待 5ms 后再次检查按键状态。这 5ms 给了按键足够的时间来停止抖动。然后按键会再次被检查来确定其稳定值。正如你之前学到的，函数可以视需要返回一个值。在这个函数中，`return current` 语句在函数被调用时返回 `current` 布尔变量的值。`current` 是一个局部变量——它在 `debounce()` 函数中声明，也只能在这个函数中使用。当 `debounce` 函数被主循环调用时，其返回值被写入到程序开头定义的全局变量 `currentButton` 中。因为这个函数被定义为 `debounce`，所以您可以通过语句 `currentButton=debounce(lastButton)` 来从 `setup()` 或 `loop()` 函数中调用这个函数。`currentButton` 的值将被设置为 `debounce` 函数所返回的值。

调用这个函数并填充了 `currentButton` 变量之后，你可以通过 `if` 语句来方便地将其与之前的按键状态值做比较。其中“`&&`”是一个逻辑运算符，意思是“AND”（与）。在 `if` 语句中将两个或更多的等式用“`&&`”连接起来，表示 `if` 语句块的内容只在同时满足这些等式时执行。如果按键之前的状态是 LOW，而现在是 HIGH，则你可以假设按键已经被按下，然后翻转变量 `ledOn` 的值。在变量 `ledOn` 前加一个“`!`”，就可以将变量设置为相反的值，而不论其当前值是什么。在循环的最后更新了按键状态变量并写入更新后的 LED 状态。

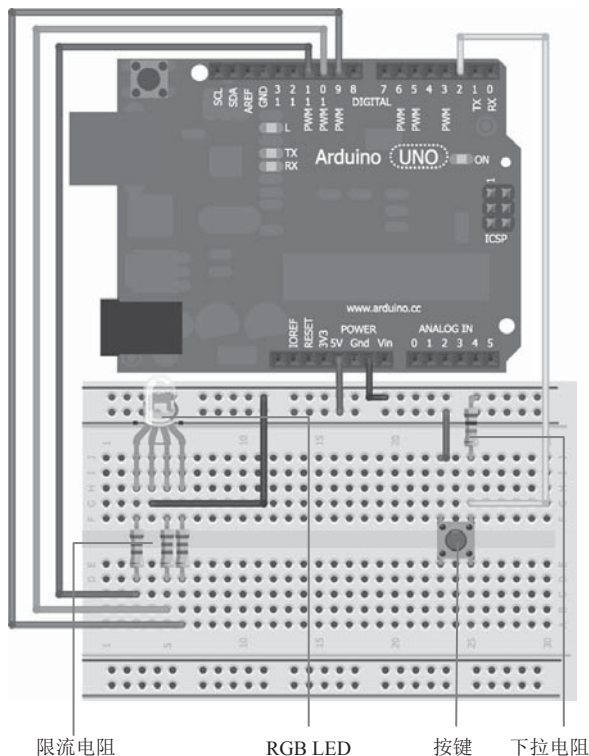
这段代码应该可以在每次按下按键时改变 LED 的状态。如果不对按键进行抖动消除，则你会发现其结果是不可预测的，LED 有时候能按预期工作，有时候则不能。

制作一个可控的 RGB LED 夜灯

在本章，你学习了如何控制数字输出、如何读取存在抖动的按键及如何使用

PWM 来改变 LED 的亮度。你现在就可以利用这些技巧，将 RGB LED 和一个用于切换颜色的按键连接在一起，制作一个可控的 RGB LED 夜灯。我们可以通过改变 RGB LED 中每种颜色的亮度来混合出不同的颜色。

在这种情境下，你可以使用一个共阴极 LED，这意味着 LED 有 4 个引脚。其中一个 3 个二极管共享的阴极，而其他三个连接到每种颜色二极管的阳极。将这个 LED 通过限流电阻连接到 Arduino 的 3 个 PWM 引脚上，如图 2-7 所示。



该图片是使用Fritzing制作的

图 2-7 夜灯的接线图

你可以配置一个按键，在每次按下它时切换 LED 的颜色。要实现这个功能，最好先添加一个额外的函数来将 RGB LED 的颜色切换成颜色循环中的下一个状态。在下列程序中（见代码清单 2-6），总共定义了 7 种颜色状态，外加一个 LED 关闭状态。利用 `analogWrite()` 函数，你可以选择自己的混色组合。与上一个示例相比，对 `loop()` 的改动只是将按下按键时的操作从翻转单个 LED 变成了递增 LED 状态计数器，并在循环完所有的选项后将其重置为零。将这个程序烧写至 Arduino，然后就可以使用你的夜灯了。通过改变 `analogWrite()` 的数值来改变颜色状态，选择自己想要的颜色。

代码清单 2-6: 切换 LED 夜灯——rgb_nightlight.ino

```

const int BLED=9;      //蓝色 LED 在 9 号引脚
const int GLED=10;     //绿色 LED 在 10 号引脚
const int RLED=11;     //红色 LED 在 11 号引脚
const int BUTTON=2;    //按钮连接到 2 号引脚

boolean lastButton = LOW;    //最后一次按钮状态
boolean currentButton = LOW; //当前按钮状态
int ledMode = 0;            //在 LED 状态间循环

void setup()
{
    pinMode (BLED, OUTPUT); //将蓝色 LED 设置为输出
    pinMode (GLED, OUTPUT); //将绿色 LED 设置为输出
    pinMode (RLED, OUTPUT); //将红色 LED 设置为输出
    pinMode (BUTTON, INPUT); //将按钮设置为输入 (非必需)
}

/*
 * 消抖动函数
 * 传入前一个按钮状态, 返回当前消抖动的按钮状态
 */
boolean debounce(boolean last)
{
    boolean current = digitalRead(BUTTON);    //读取按钮状态
    if (last != current)                      //如果它不同于之前的状态
    {
        delay(5);                            //等待 5ms
        current = digitalRead(BUTTON);        //再次读取
    }
    return current;                          //返回当前值
}

/*
 * LED 模式选择
 * 传入一个表示 LED 状态的数值, 从而设置相应的状态
 */
void setMode(int mode)
{
    //红色
    if (mode == 1)
    {
        digitalWrite(RLED, HIGH);
        digitalWrite(GLED, LOW);
        digitalWrite(BLED, LOW);
    }
    //绿色
    else if (mode == 2)
    {

```

```
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
}
//蓝色
else if (mode == 3)
{
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
}
//紫色(红色+蓝色)
else if (mode == 4)
{
    analogWrite(RLED, 127);
    analogWrite(GLED, 0);
    analogWrite(BLED, 127);
}
//青色(蓝色+绿色)
else if (mode == 5)
{
    analogWrite(RLED, 0);
    analogWrite(GLED, 127);
    analogWrite(BLED, 127);
}
//橙色(绿色+红色)
else if (mode == 6)
{
    analogWrite(RLED, 127);
    analogWrite(GLED, 127);
    analogWrite(BLED, 0);
}
//白色(绿色+红色+蓝色)
else if (mode == 7)
{
    analogWrite(RLED, 85);
    analogWrite(GLED, 85);
    analogWrite(BLED, 85);
}
//关闭 (mode = 0)
else
{
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
}
}

void loop()
{
    currentButton = debounce(lastButton);           //读取消抖动状态
    if (lastButton == LOW && currentButton == HIGH) //如果按键被按下
    {
```

```
    ledMode++;                                //递增 LED 状态值
}
lastButton = currentButton;                  //重置按键值
//如果你已经遍历了所有选项，则将计数器重置到 0
if (ledMode == 8) ledMode = 0;
setMode(ledMode);                            //改变 LED 状态
}
```

看起来有好多代码，但这只是你在本章中已经写过的代码块的组合而已。

对这些代码还能做哪些修改呢？你可以添加额外的按键来独立地控制三种颜色中的某一种。你还可以添加闪烁模式，利用第 1 章中的代码来让 LED 闪烁。总之，有无限的可能性。

本章小结

在本章中你学到了如下内容：

- 面包板是如何工作的；
- 如何选择一个电阻为 LED 限流；
- 如何将外部 LED 连接到 Arduino；
- 如何使用 PWM 向 LED 写入“模拟”值；
- 如何读取一个按键；
- 如何为按键消抖动；
- 如何使用 for 循环；
- 如何利用上拉和下拉电阻。

第 3 章

读取模拟传感器

本章所需的器件：

Arduino Uno

小号面包板

跳线

10k Ω 电位器

10k Ω 电阻 ($\times 2$)

220 Ω 电阻 ($\times 3$)

USB 电缆

光敏电阻

TMP36 温度传感器 (或任意其他 5V 模拟传感器)

5mm 共阴极 RGB LED

本章的代码和数字化内容

本章的代码、视频及其他数字化内容可以在以下网站找到：
www.exploringarduino.com/content/ch3。本章视频也可以在以下网址找到：
<http://kuangqi.me/arduino/>，进入该网站即可找到。

另外，所有的代码都可以在 www.wiley.com/go/exploringarduino 网页上的“Download Code”标签中找到。在第 3 章中提供下载的代码根据其在本章中的名称单独命名。

我们周围的世界是模拟的。尽管你可能听说这个世界正在“走向数字化”，

但自然绝大部分能观察到的特性在本质上都是模拟的。这个世界可以承载无限多的可能状态，无论是阳光的颜色、海洋的温度还是空气中污染物的浓度。本章将着力于开发一些技术，将这些无穷的可能性离散化为我们喜爱的数字值，这样它们就可以被 Arduino 这种微控制器系统所分析。

在本章，你将学到模拟与数字信号的不同及如何在这二者间转换，你还会学到一些可以与 Arduino 交互的模拟传感器。利用在之前章节中掌握的技巧，你会添加光线传感器来自动调节夜灯的亮度，会学到如何将模拟数据利用 USB 转串口连接从 Arduino 发送到你的计算机，这为开发更复杂的系统提供了巨大的潜力，使你能将环境数据传送到你的计算机。

注意 你可以跟随一个视频来学习模拟输入：<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头处的 Wiley 网站上找到这个视频。

如果你想学习更多关于模拟和数字信号的不同点，则在这个视频中会有深入的讲解：<http://kuangqi.me/arduino/>，进入该网站即可找到。你也可以在本章开头处的 Wiley 网站上找到这个视频。

理解模拟信号和数字信号

如果你想让设备与周围的世界交互，就不可避免要使用模拟数据。考虑在第2章中完成的项目，使用一个开关来控制 LED。开关是数字输入——它只有两种可能的状态：开或关、高或低、1 或 0，等等。数字化信息（计算机或者 Arduino 能够处理的信息）是一系列二进制（或数字化）数据。每一位都只能是两个可能值中的一个。

然而，我们身边的世界却很少只使用两种状态来表示信息。看一眼窗外，你看到了什么？如果现在是白天，那么你也许会看到阳光，看到树木在微风中摇摆，抑或是看到川流不息的车辆和人群。你看到的一切都不能轻易地用二进制数据表示。阳光不能以“开”或“关”表示，它的亮度在一天中不断发生变化。类似地，风也不是只有两种状态，其风力无时无刻不在改变。

比较模拟信号和数字信号

如图 3-1 所示的曲线对比了模拟信号和数字信号。左边的图是一串方波，它只在两个状态间变化：0V 和 5V。就像你在第 2 章中使用的按键一样，这个信号也只有“逻辑高电平”和“逻辑低电平”两个值。右边的图是余弦波的一部分，

尽管其上下界仍是 0V 和 5V，但这个信号可以在这两个电压间取无穷多个值。

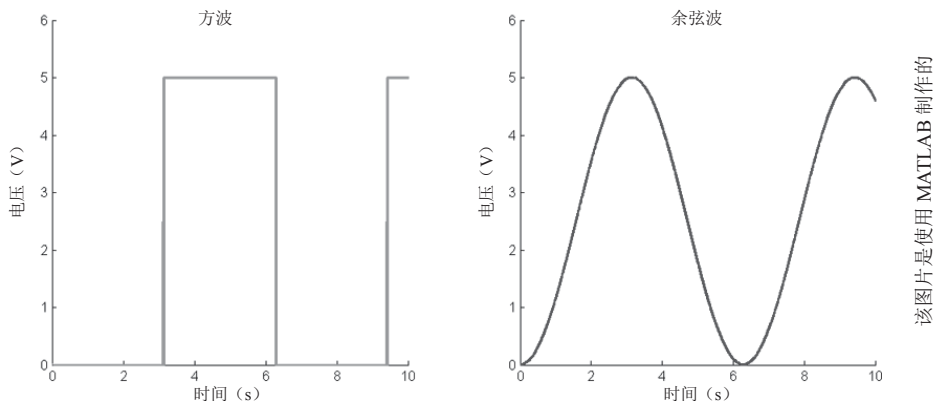


图 3-1 数字信号和模拟信号

模拟信号不能被离散地分类，它们在一个范围内变化，理论上可以在那个范围内取到无穷多个值。设想一下，将阳光作为一个想要测量的模拟输入示例。当然，你要测量的阳光也有一个合理的范围。单位通常为勒克斯（lux），即单位面积的光通量，你可以合理地期望其测量值在 0（漆黑一片）~130 000 lux（阳光直射）。如果你的测量设备是无穷精确的，那么你就可以在这个范围内测到无穷多个不同的数值。一个室内的场景也许是 400lux，如果稍微亮一点，则也许是 401lux，或者 401.1lux，或者 401.11lux 等。计算机系统根本无法测量一个具有无穷多小数位的模拟量，这是因为计算机的存储和计算能力都是有限的。那么在这种情况下，怎样才能将 Arduino 与“真实世界”连接在一起呢？答案就是模数转换器（analog-to-digital converter, ADC），它可以将模拟量以有限的精度和速度表示为数字量。

将模拟信号转换为数字信号

假设你想测定房间的亮度。大概来说，一个好的亮度传感器可以产生随房间亮度变化而变化的输出电压。全黑时，设备会输出 0V，而在日光下完全饱和时则输出 5V，在这之间的电压值的值则表示了各种不同的光照强度。看起来还不错，可是如何才能在 Arduino 上读取这些数值并计算出房间的亮度呢？你可以使用 Arduino 的模数转换器（ADC）引脚来将模拟电压值转换为数字表示，这样你就可以处理它们了。

一个 ADC 的精度是由其分辨率决定的。在 Arduino Uno 中，有一个 10 位的 ADC 来完成模拟转换工作。“10 位”意味着这个 ADC 可以把一个模拟信号细分（或称“量化”）为 2^{10} 个不同的值。如果你简单计算一下，那么就会发现 $2^{10}=1024$ ，

因此，Arduino 可以为给定的任意模拟量分配一个 0~1023 的值。尽管可以改变参考电压，但在本书中的所有模数转换示例中，我们将使用默认的 5V 参考电压。参考电压确定了你能测量的最高电压，也就是会被映射到 1023 的那个电压值。因此，在使用 5V 参考电压时，在 ADC 引脚上输入 0V 会返回 0，输入 2.5V 会返回 512（1023 的一半），而输入 5V 会返回 1023。为了更好地理解这个过程，让我们来设想一个 3 位 ADC 的工作过程，如图 3-2 所示。

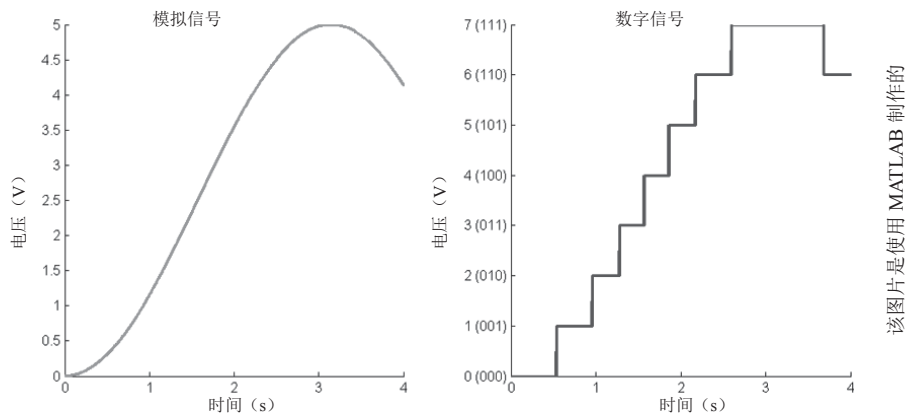


图 3-2 3 位模拟量化

注意 如果你想进一步学习如何使用自己的参考电压或者使用不同的内部参考电压,则请访问 Arduino 网站的 `analogReference()` 页面: www.arduino.cc/en/Reference/AnalogReference。

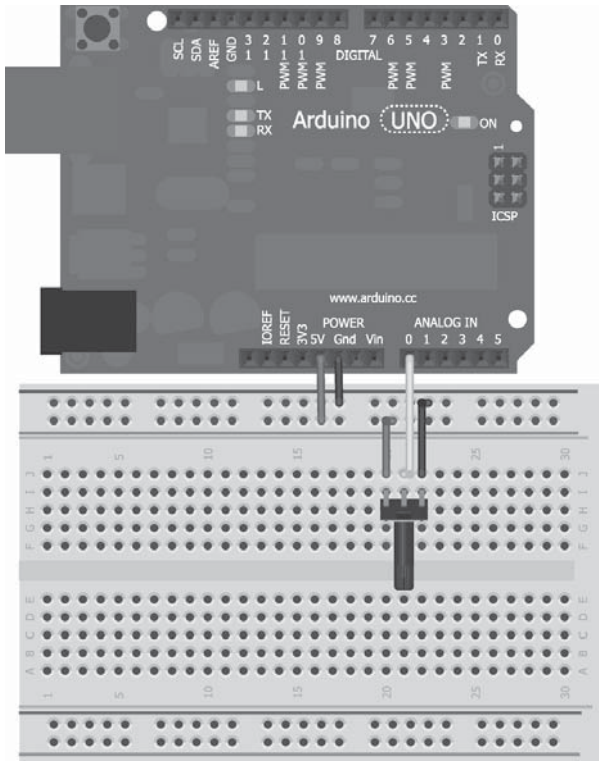
3 位 ADC 具有 3 位的分辨率。因为 $2^3=8$, 所以共有 8 个逻辑电平, 即 0~7。因此, 任何传递给 3 位 ADC 的模拟量都会被转换为 0~7 的一个值。请看图 3-2, 你会发现这个电压电平被转换为离散的数字值, 这样它才能被微控制器使用。分辨率越高, 用来表示每个值的“台阶”数就越多。在 Arduino Uno 中, 一共有 1024 个“台阶”, 而不是如图 3-2 所示的 8 个。

利用 Arduino 读取模拟传感器: `analogRead()`

现在你已经理解了如何将模拟信号转换为数字信号, 下面就可以将这些知识整合到程序和电路中。不同的 Arduino 的模拟输入引脚的数目也不同, 但读取它们的方式却是相同的, 那就是使用 `analogRead()` 命令。首先, 用一个电位器和一个封装的模拟传感器进行实验。然后, 学习分压器的工作原理, 并利用一些能根据某种输入而改变电阻的设备, 来制作自己的模拟传感器。

读取电位器

电位器（或简称 pot）是可读取的最简单的模拟传感器。可能在你家中就有大量的电位器：在你的立体音响、扬声器、恒温控制器、汽车里，以及其他地方。电位器是一个可变的分压器（本章稍后详解），它看起来像个旋钮。电位器有各种不同的尺寸和形状，但大都具有 3 个引脚。外侧的一个引脚连接到地，另外一个连接到 5V。电位器是对称的，因此将哪一边连接到地或 5V 是无所谓的。将中间的引脚连接到 Arduino 的 0 号模拟输入端。图 3-3 展示了如何正确地将电位器连接到 Arduino。



该图片是使用Fritzing制作的

图 3-3 电位器电路

在转动电位器时，输入到 0 号模拟输入端的电压就会在 0~5V 间改变。如果你想确认这个过程，则可以将一个万用表置于电压档，并像图 3-4 中那样将其连接到电路中，然后在转动旋钮时观察显示器上的读数。红色的（正极）表笔应该连接到中间的引脚，而黑色的（负极）表笔应该连接到接地的那一边。需要注意的是，你的电位器和万用表可能与如图 3-4 所示的有些不同。



图 3-4 万用表测量

在使用电位器控制其他硬件之前，我们要利用 Arduino 的串口通信功能，在电位器的 ADC 值改变时将其打印到你的计算机上。使用 `analogRead()` 函数来读取 Arduino 模拟输入端的值并使用 `Serial.println()` 函数将其打印到 Arduino IDE 的串口监视器上。首先，请为你的 Arduino 编写并烧写代码清单 3-1 中的程序。

代码清单 3-1：电位器读数程序——pot.ino

```
//电位器读数程序
const int POT=0;           //电位器连接在 0 号模拟引脚
int val = 0;               //保存电位器模拟读数的变量

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

void loop()
{
  val = analogRead(POT);
  Serial.println(val);
  delay(500);
}
```

关于串口的功能，在之后的章节中将进一步研究。现在，只需要知道连接到计算机的串口必须在 `setup()` 函数中初始化。`Serial.begin()` 接收一个参

数，该参数指定了通信速率（或称“波特率”）。波特率指定了每秒传输的位数。波特率越高传输的数据越多，所花时间的越短，但在某些通信系统中也会引入一些传输错误。波特率 9600 是一个常用值，我们在整本书中都会使用这个波特率。

在循环的每次迭代中，变量 `val` 都被设置为 ADC 从 0 号模拟引脚读到的当前值。`analogRead()` 命令需要传入 ADC 引脚的编号。在这里其值为 0，也就是你连接电位器的引脚。你也可以传入 `A0`，由于 `analogRead()` 函数知道你一定会传给它的一定是一个模拟引脚的编号，所以也可以简写为 0。当这个数值被读取之后（一个 0~1023 的数），`Serial.println()` 会通过串口将那个值打印到计算机的串口终端，并以一个换行符结尾，使光标移到下一行。接下来，这个循环会停顿 0.5s（这样数字就不会滚动得过快以至于没法阅读），然后重复上述过程。

将这个程序烧写至 Arduino 后，你会注意到 Arduino 上的 TX LED 每隔 500ms 闪烁一次（至少它应该这样）。这个 LED 表明 Arduino 正在将数据通过 USB 接口发送到计算机上的串口终端。你可以使用各种终端程序监视 Arduino 发送的数据，而在 Arduino IDE 中就内置了一个这样的程序！单击如图 3-5 所示圆形的按钮就可以运行这个串口监视器。

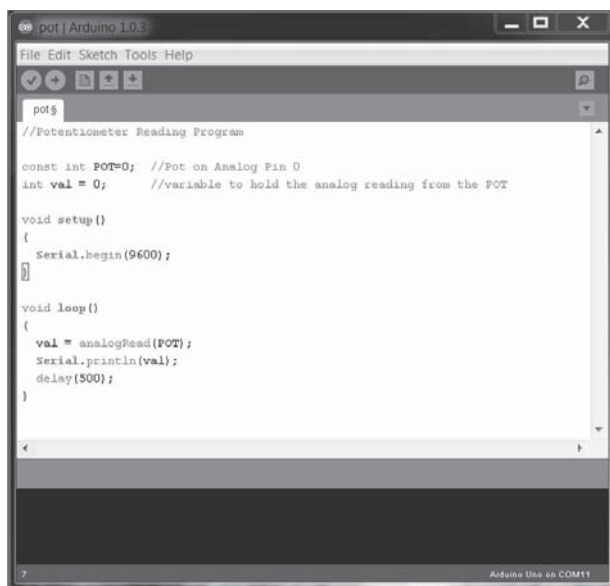


图 3-5 串口监视器按钮

在运行这个串口监视器之后，你应该看到在一个窗口中有数字流过。转动电位器，你就会发现这些数字随着电位器的位置而上下波动。如果将沿一个方向转

到底，则这个数字应该接近 0，而沿另外一个方向转到底，则这个数字应该接近 1023。它看起来与图 3-6 中的示例类似。

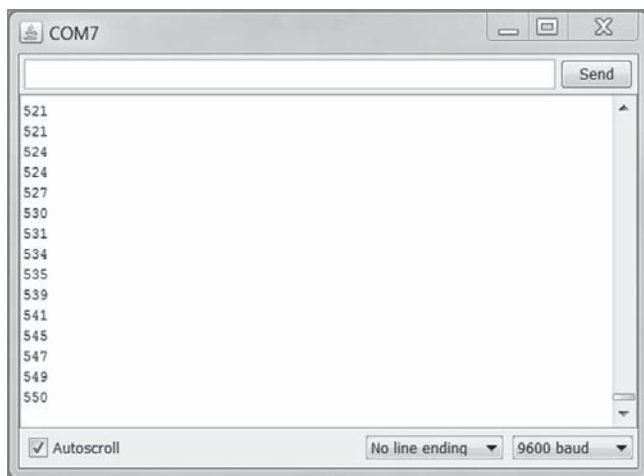


图 3-6 传入的串口数据

注意 如果你得到的是乱码，则请确认已经选择了正确的波特率。由于在代码中将其设置为 9600，因此在这个窗口中也要将波特率设置为这个值。

你已经能够通过旋转旋钮来改变一些数值了，很有意思吧？没意思？好吧，这只是第一步。接下来，你要学习其他类型的模拟传感器及如何利用来自传感器的数据去控制其他硬件。现在只是使用熟悉的 LED，但在之后的章节中，你会使用电动机和其他输出设备来可视化模拟输入。

使用模拟传感器

尽管电位器可以在一个引脚上产生模拟电压，但它们在传统意义上并非真正的传感器。它可以“感知”你对电位器的转动，但这很快就变得无聊了。好在所有的传感器都可以根据“真实世界”的活动产生模拟输出。这样的例子包括：

- 检测倾角的加速度计（许多智能手机和平板电脑里都有它的身影）；
- 检测磁场的磁力计（用于制作电子罗盘）；
- 测距使用的红外传感器；
- 可以感知操作环境的温度传感器。

大多数此类传感器的工作方式与你刚刚体验过的电位器是类似的：为它们提供供电（VCC）和接地（GND）连接，它们就会从连接到 Arduino ADC 的第三个引脚上输出值位于 VCC 和 GND 之间的模拟电压。

对于下一个实验，你可以自己选择想要使用的模拟传感器。它们都会在连接到 Arduino 时输出一个 0~5V 的电压，所以用哪个都可以。下面是一些你可以使用的传感器的示例。

■ 夏普红外接近传感器

www.exploringarduino.com/parts/IR-Distance-Sensor

连接器：www.exploringarduino.com/parts/JST-Wire

夏普红外距离传感器是一款非常流行的传感器，它可以测定你与其他物体间的距离。当你远离所指向的物体时，输出电压就会下降。数据手册（上面链接给出的器件网页）中的图 5 给出了测定的距离和电压间的关系。

■ TMP36 温度传感器

www.exploringarduino.com/parts/TMP36

我们可以很容易地将 TMP36 温度传感器的输出电压电平与摄氏温标读数关联起来。每 10mV 对应 1℃，你可以很容易地创建一个线性函数来将测到的电压值转换为任意环境下的绝对温度： $^{\circ}\text{C}=[(\text{输出电压毫伏数}) - 500]/10$ 。其中偏移量 -500 是为了处理低于 0℃ 的温度。如图 3-7（提取自数据手册）所示的曲线显示了这种转换。

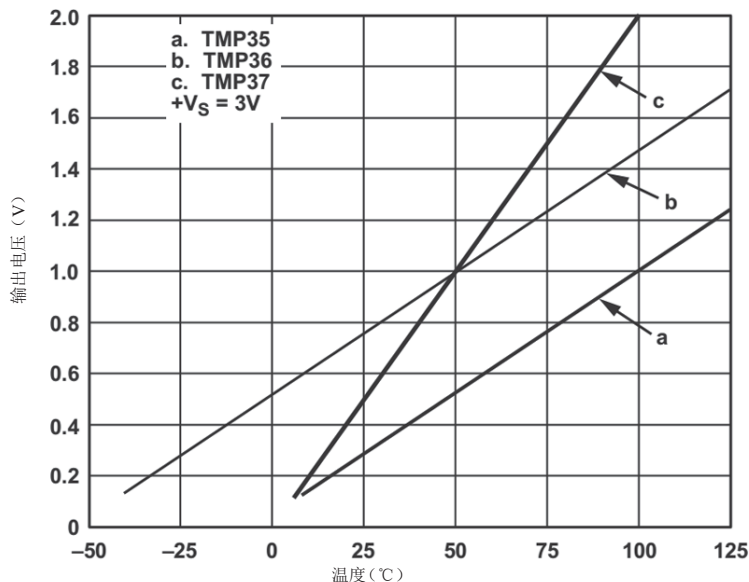


图 3-7 电压与温度的对应关系

引用自：Analog Devices, Inc., www.analog.com

00337-007

■ 三轴模拟加速度计

www.exploringarduino.com/parts/TriAxis-Analog-Accelerometer

三轴加速度计是检测方向的好方法。模拟加速度计输出的是与各轴方向运动相关的模拟值：X、Y 和 Z（每个值都在不同的引脚上）。使用一些有趣的数学公式（三角函数和重力相关的知识），你就能利用这些电压值去确定你的项目在三维空间中的位置！重要的是，许多这样的传感器是 3.3V 的，所以你需要使用 `analogReference()` 函数配合 AREF 引脚，将 3.3V 设置为参考电压，以获得传感器的完整的分辨率。

■ 二轴模拟陀螺仪

www.exploringarduino.com/parts/DualAxis-Analog-Gyroscope

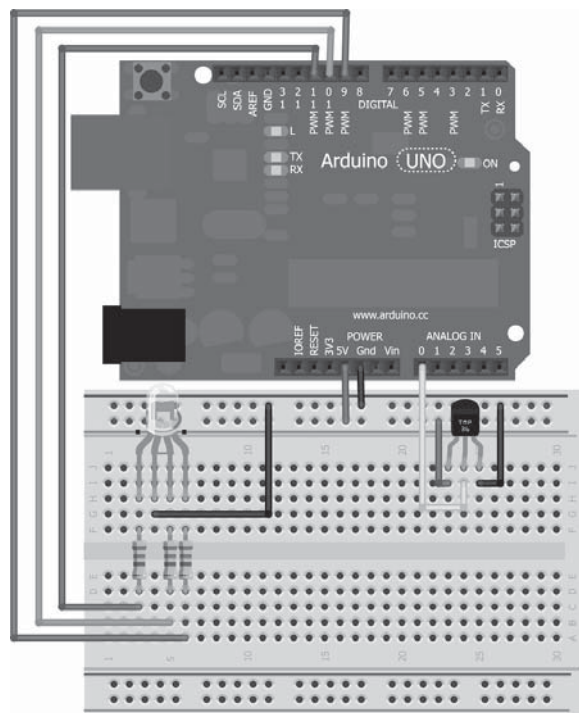
陀螺仪与加速度计不同，它不受重力的影响。它的模拟输出电压取决于绕着一个轴的角加速度。该特性对于检测扭转运动特别有用。有一个陀螺仪与 Arduino 联动的示例，请查看我的 SudoGlove: (www.sudoglove.com)，它是我设计的一只用来采集手势的手套，从而控制诸如音乐合成器和遥控车等硬件。跟加速计一样，要注意许多陀螺仪都是 3.3V 的器件。

现在你已经选择了一个传感器，让我们把这个传感器用起来吧。

【利用模拟传感器来感知温度】

这是一个简单的示例，使用了前一节提到的 TMP36 温度传感器。然而，你可以使用买到的任何模拟传感器。你既可以使用之前列表中列出的任意一款，也可以自己找一个（如果你使用的是 Arduino Uno，则传感器必须兼容 5V。）下列步骤对于任何你想要使用的模拟传感器基本相同。

首先，像在第 2 章中所做的那样连接 RGB LED，然后将温度传感器连接到 0 号模拟输入端，如图 3-8 所示。



该图片是使用Fritzing制作的

图 3-8 温度传感器电路

使用这个电路，你可以制作一个简单的温度报警系统。当温度在可接受的范围内时，LED 发绿光，而在太热时发红光，在太冷时发蓝光。

首先，需要确定想要使用的温度阈值。使用与代码清单 3-1 完全相同的程序，并使用串口监视器来找到你关注的温度阈值所对应的模拟数值。我的房间大约 20℃，其对应的模拟读数大概是 143。对你来说，这些值可能是不同的，所以请运行前面的程序，打开串口终端，看看你得到的读数。你也可以根据图 3-7，通过数学方法确定读数。对我来说，143/1023 的值对应一个约 700mV 的电压输入。下面的公式取自数据手册，可以用于在温度（℃）和电压（mV）间进行转换：

$$\text{温度 (}^{\circ}\text{C)} \times 10 = \text{电压 (mV)} - 500$$

代入 700mV 的值，你就会发现得到的温度值为 20℃。利用同样的方法（或者简单地观察串口窗口并选取一个值），你可以求得 22℃对应的数字值是 147，而 18℃对应的数字值是 139。这些值会被作为阈值，来改变 LED 的颜色，表明房间太热还是太冷。利用你目前学到的 if 语句、digitalWrite() 函数和 analogRead() 函数，你可以方便地读取温度，求出温度所在的范围并相应地设置 LED。

注意 在你复制代码清单 3-2 中的代码之前，尝试着自己写一下程序，看看你自己的程序能否工作。尝试过后，与这里的代码对比。你感觉如何？

代码清单 3-2：温度报警器程序——tempalert.ino

```
const int BLED=9;           //蓝色 LED 在 9 号引脚
const int GLED=10;          //绿色 LED 在 10 号引脚
const int RLED=11;          //红色 LED 在 11 号引脚
const int TEMP=0;           //温度传感器在 A0 引脚

const int LOWER_BOUND=139;   //阈值下限
const int UPPER_BOUND=147;   //阈值上限

int val = 0;                //保存模拟读数的变量

void setup()
{
    pinMode (BLED, OUTPUT); //将蓝色 LED 设置为输出
    pinMode (GLED, OUTPUT); //将绿色 LED 设置为输出
    pinMode (RLED, OUTPUT); //将红色 LED 设置为输出
}

void loop()
{
    val = analogRead(TEMP);

    if (val < LOWER_BOUND)
    {
        digitalWrite(RLED, LOW);
        digitalWrite(GLED, LOW);
        digitalWrite(BLED, HIGH);
    }
    else if (val > UPPER_BOUND)
    {
        digitalWrite(RLED, HIGH);
        digitalWrite(GLED, LOW);
        digitalWrite(BLED, LOW);
    }
    else
    {
        digitalWrite(RLED, LOW);
        digitalWrite(GLED, HIGH);
        digitalWrite(BLED, LOW);
    }
}
```

这个代码清单没有引入任何新概念，只是将你已经学过的东西进行了组合，让这个系统同时利用输入、输出与环境进行交互。要测试这个程序，可以用手指捏住温度传感器或者向它哈气来使其升温，然后将其吹凉。

利用可变电阻制作自己的模拟传感器

多亏有物理学，我们知道许多设备都可以借由物理活动改变自身的电阻。举例来说，一些导电墨的阻抗可以在受到挤压或弯曲时改变（压力传感器和弯曲传感器），一些半导体的阻抗能在受到光线照射时改变（光敏电阻），一些聚合物的阻抗能在受热或遇冷时改变（热敏电阻）。这只是几个例子，你可以利用这些特性来制作自己的模拟传感器。由于这些传感器都改变电阻而不是电压，因此需要制作一个分压器电路，这样你才能测定它们的电阻的变化。

使用阻性分压器

阻性分压器使用两个电阻来对输入电压分压。其输出电压是一个取决于这两个电阻阻值的函数。所以，如果其中的一个电阻是可变的，那么就可以通过监测分压器的电压变化，获知电阻的变化。另外一个电阻的阻值可以用来设置这个电路的灵敏度，也可以使用电位器来调节灵敏度。

首先，让我们来看一个固定的分压器及它的方程，如图 3-9 所示。图 3-9 中的 A0 表示的是 Arduino 上的 0 号模拟引脚。

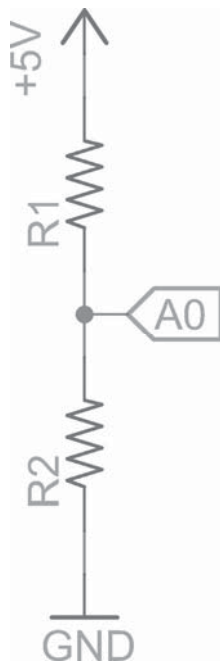


图 3-9 简单的分压器电路

分压器的方程如下：

$$V_{out} = V_{in}(R_2 / (R_1 + R_2))$$

在本例中，输入电压是 5V，而输出电压则要导入到一个 Arduino 的模拟引脚。这里 R1 和 R2 是相等的（例如均为 10kΩ），则 5V 除以 2，在模拟输入端得到 2.5V 的电压。可以将上述数值带入方程来验证这个结果：

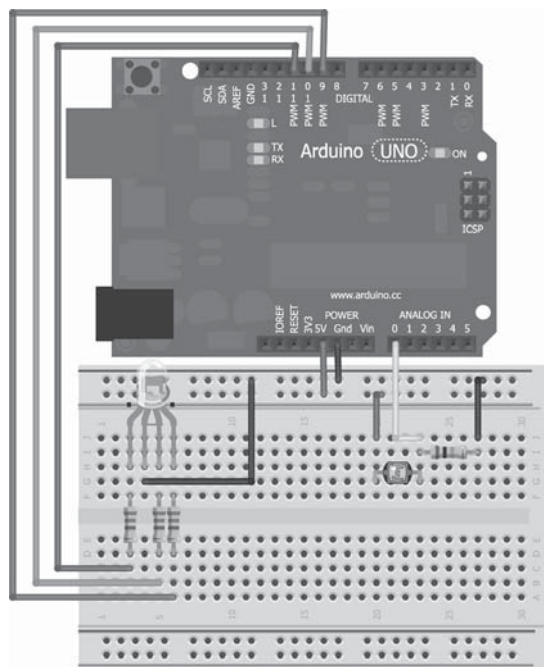
$$V_{out} = 5V(10k / (10k + 10k)) = 5V \times 0.5 = 2.5V$$

现在，假设将其中的一个电阻用可变电阻（比如光敏电阻）代替。光敏电阻（见图 3-10）可以根据照射它的光的强度来改变其电阻。在本例中，我们要使用一个 200kΩ 的光敏电阻。在全黑的环境中，它的电阻约为 200kΩ，而在饱和的日光下，其电阻下降到近乎为零。对定值电阻 R1 或 R2 阻值的选择可能会影响读数的范围和精度。尝试用不同的配置做实验，并通过串口监视器来查看这些数值是如何改变的。举例来说，将 R1 替换为光敏电阻，而将 R2 的阻值选为 10kΩ（见图 3-11）。你可以将 RGB LED 保留在原位，尽管在这个练习中只用到了—种颜色。



引用自：element14, www.element14.com

图 3-10 光敏电阻



该图片是使用Fritzing制作的

图 3-11 光敏电阻电路

再次烧写可靠的串口打印程序（见代码清单 3-1），并尝试着改变光敏电阻上方的光照条件。将其朝向光源，再用手盖住它。奇怪的是，它并不会在 0~1023 间全量程变化，这是因为可变电阻的阻值不可能真正为零。有趣的是，你也许还可以算出能够接收的最大值和最小值。你可以利用光敏电阻的这些数据来制作一个更智能的夜灯。在屋子变暗时，这个夜灯会变得更亮，反之亦然。利用串口监视器程序，你可以选取表示屋子全亮或全暗状态下的数值。对于我来说，黑暗屋子这个值在 200 左右，而明亮屋子值在 900 左右。这些值可能会有所改变，它取决于光照条件、选用的固定电阻值及光敏电阻值。

利用模拟输入控制模拟输出

回顾一下，你可以使用 `analogWrite()` 命令来设置 LED 的亮度。然而，这是一个 8 位的值，也就是说，取值范围在 0~255，而 ADC 返回的数值最高可达 1023。幸运的是，Arduino 编程语言有两个函数能在两组数值之间进行映射，它们就是 `map()` 和 `constrain()` 函数。其中 `map()` 函数为：

```
output = map(value, fromLow, fromHigh, toLow, toHigh)
```

`value` 是你处理的数值。在这个例子中，也就是模拟输入端的读数。`fromLow`

和 `fromHigh` 是输入的上下限。这两个值应该与你房间中的最小和最大亮度值一致。在我这里，它们分别是 200 和 900。`toLow` 和 `toHigh` 是你想要映射到的数值范围的上下限。由于 `analogWrite()` 期望的数值是 0~255，所以就在这里填写这两个数。然而，我们想要将较暗的房间映射到较亮的 LED。因此，当输入给 ADC 一个较小的数值时，你要输出给 LED 较大的数值，反之亦然。

`map()` 函数可以方便地自动处理这种情况，只需要简单地交换最大和最小值就可以了，也就是将最小值设置为 255 而将最大值设置为 0。`map()` 函数可以生成一个线性映射。举例来说，如果 `fromLow` 和 `fromHigh` 的值分别为 200 和 900，而 `toLow` 和 `toHigh` 的值分别为 255 和 0，则 550 就会映射为 127，这是因为 550 是 200~900 的中值，而 127 是 255~0 的中值。重要的是，`map()` 函数不会约束这些数值。所以，如果光敏电阻的测定值低于 200，则会映射到一个高于 255 的值（由于进行的是反向映射）。显然，你并不希望这样，因为不能将高于 255 的值传递给 `analogWrite()` 函数。你可以使用 `constrain()` 函数来处理这种情况。`constrain()` 函数为：

```
output = constrain(value, min, max)
```

如果你将 `map()` 函数的输出传递给 `constrain()` 函数，则可以将 `min` 设置为 0，将 `max` 设置为 255，这就能确保任何高于或低于限定值的数值都会被限制为 0 或 255。最后，你就可以使用这些数值来命令 LED 了！现在，让我们来看看最终的程序（见代码清单 3-3）。

代码清单 3-3：自动夜灯程序——nightlight.ino

```
const int RLED=9;           //红色 LED 在 9 号引脚 (PWM)
const int LIGHT=0;          //光敏传感器在 0 号模拟引脚
const int MIN_LIGHT=200;    //预期的最小亮度值
const int MAX_LIGHT=900;    //预期的最大亮度值
int val = 0;                //保存模拟读数的变量

void setup()
{
    pinMode(RLED, OUTPUT); //将 LED 引脚设置为输出
}

void loop()
{
    val = analogRead(LIGHT);           //读取光敏传感器
    val = map(val, MIN_LIGHT, MAX_LIGHT, 255, 0); //映射亮度读数
    val = constrain(val, 0, 255);      //约束亮度值
    analogWrite(RLED, val);            //控制 LED
}
```

注意这段代码重用了变量 `val`。你也可以为每个函数使用一个不同的变量。在诸如 `map()` 这样的函数中，其中 `val` 既是输入也是输出，`val` 先前的数值被用作输入，而当这个函数执行完成之后，它的数值又被设置为更新之后的值。

试着用一下你的夜灯，它是否像预期的那样工作呢？记住，你可以通过改变映射函数的上下界及定值电阻的阻值来调节灵敏度。利用串口监视器来观察不同设定值之前的区别，直到找到工作状态最好时的设定值。你可以将这个程序与在第 2 章中设计的变色夜灯结合起来吗？尝试添加一个按键来切换颜色，并使用光敏电阻来调节每种颜色的亮度。

本章小结

在本章你学到了如下内容：

- 模拟信号和数字信号的区别；
- 如何将模拟信号转换为数字信号；
- 如何从电位器读取一个模拟信号；
- 如何使用串口监视器显示数据；
- 如何与集成模拟传感器接驳；
- 如何制作你自己的模拟传感器；
- 如何映射和约束模拟读数从而驱动模拟输出。

探索Arduino的大千世界 本书就是你的翅膀

Jeremy Blum因其在Youtube上发布的系列教学视频而广为人知，这些视频向世界范围内的人们讲授了工程学的概念。他制作了大量的基于微控制器的系统，包括太阳跟踪器、手臂义肢、桁架遍历机器人、泰勒明电子琴、基于计算机视觉、手套的手势控制器等。

《Arduino魔法书：实现梦想的工具和技术》是著名创客Jeremy Blum汇聚多年开发经验并结合当下热门技术撰写的一本实用性极强的工具书，也是继电子工业出版社推出《Arduino从基础到实践》、《Arduino机器人权威指南》图书后的又一旷世之作。全书内容丰富，采用“理论入、实践出”的教学方式，促使读者加快对基础知识的认知和理解。实验案例图文并茂、注释清晰，由浅入深，通俗易懂。

于欣龙，奥松机器人创始人、资深创客

Jeremy在电气工程领域使用Arduino制作电子产品，犹如能工巧匠用锤子建造房子。

Bre Pettis, MakerBot Industries创始人

Jeremy以其简明、易懂的风格对Arduino进行了深度解析，这不仅能让数码新手找到他们的立足点，也能让硬件老兵发现电子世界对业余爱好者友好的一面，并开始连接他们的器件。

Chris Gammell, Amp Hour播客 (Podcast) 联合主办人

要获取书中项目的电路图、入门视频、源代码等资源，请访问本书专题页面：

<http://kuangqi.me/arduino>

WILEY



策划编辑：林瑞和
责任编辑：徐津平
封面设计：李玲

上架建议：Arduino

ISBN 978-7-121-24067-6



定价：59.00元