第 1 章

Arduino 平台概述

Arduino 是什么？不少人都有这个疑问。有的人觉得 Arduino 就是一个开发板，然而这并不准确。Arduino 也是一个开源的开发平台，越来越多的人用它来开发电子产品，因为它简单易掌握，零基础的人也能快速上手，制作出各种好玩的东西。

因为 Arduino 的种种优势，越来越多的专业硬件开发者已经或正在使用 Arduino 来开发他们的项目和产品；越来越多的软件开发者使用 Arduino 进入硬件、物联网等开发领域；在高等教育领域，自动化、软件甚至艺术专业，也纷纷开设了 Arduino 相关课程。

1.1 什么是 Arduino

Arduino（读作阿尔杜伊诺）是一款便捷灵活、方便上手的开源电子原型平台，包括 Arduino 硬件平台和 Arduino 软件平台两部分。Arduino 是一套能够用来感应和控制现实物理世界的工具。它由一个基于单片机并且开放技术资料的硬件平台和一套为 Arduino 硬件板编写程序的集成开发环境（Integrated Development Environment，IDE）组成，如图 1-1 所示。

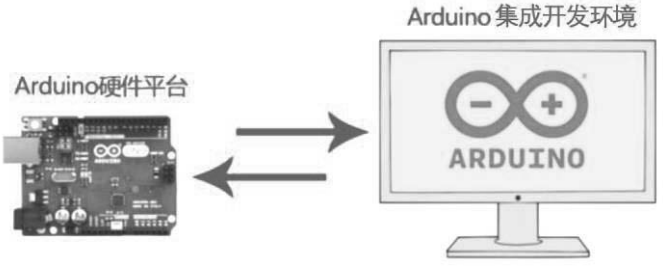

图 1-1

Arduino 平台最初由意大利的学生设计，因其具有开源、扩展性高、价格低廉等特点，一经推出就受到各界电子爱好者的青睐和推广。Arduino 平台的硬件方面，目前已经发布十多个不同类型的

电子控制板，其中使用最为广泛的是 Arduino UNO 开发板。本书中所用的 Arduino 套件均为 Arduino UNO 套件。Arduino UNO 开发板基本结构如图 1-2 所示。

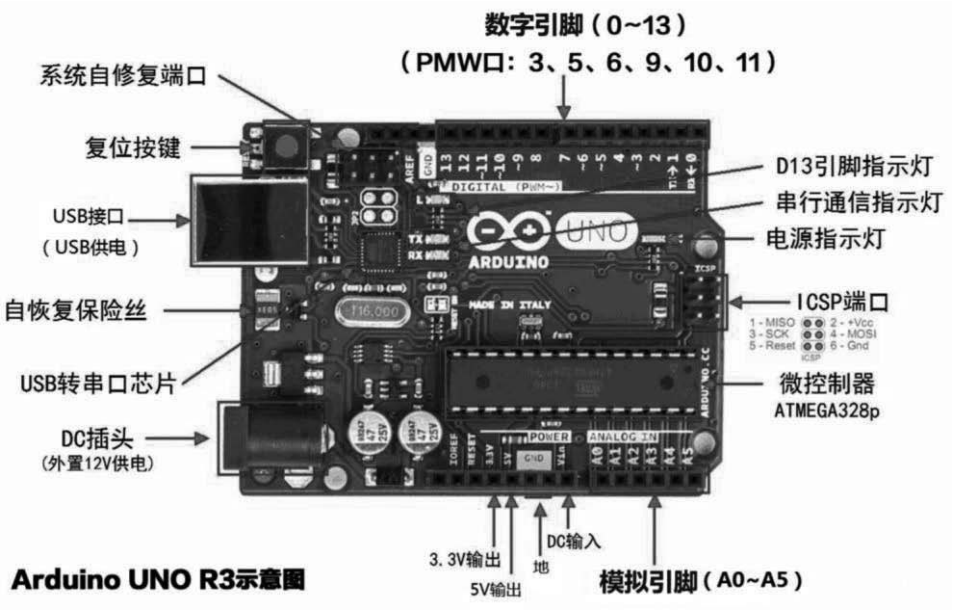

图 1-2（彩图参见配套下载资源）

Arduino UNO 开发板可以通过 USB 接口直接通电，也可以外接 6～12V 的 DC 电源通电。复位按键可以使程序运行回复到初始状态。从上图中可以看到，四个黄色的 LED 灯包括一个电源灯（ON）、一个接数字引脚 13 的 LED 灯和两个串行通信指示灯（TX/RX）。Arduino UNO 板中标出的数字引脚 0～13 和模拟引脚 A0～A5 为输入/输出引脚（也就是输入/输出接口）。Arduino UNO 开发板中有一个 3.3V 和一个 5V 的输出接口，可以根据实际情况选择使用。

Arduino 平台的软件方面是指 Arduino IDE。只有将硬件搭建与程序代码相结合，才能真正实现作品的功能。Arduino IDE 基本界面将在 2.3 节中介绍。

1.2 Arduino 的起源

意大利文艺复兴时期是一段持续了约 200 年的、令人难以置信的人类历史时期，其标志是艺术和科学技术都取得了显著进步。在这一时期，列奥纳多·达·芬奇、伽利略·伽利雷和桑德罗·波提切利等名字成为伟大思想洪流中的一小部分，他们为世界带来了难以计量的知识、艺术和发明。几个世纪后，电子技术的复兴在意大利的一个名叫伊夫雷亚的小镇出现。这一切始于一块手工焊接的电路板，这块电路板后来成为全球知名的 Arduino。

Massimo Banzi 是意大利伊夫雷亚的一家高科技设计学校的老师，他的学生经常抱怨找不到便宜好用的微控制器。2005 年冬天，Massimo Banzi 与 David Cuartielles 讨论了这个问题。David Cuartielles 是一位来自西班牙的晶片工程师，他当时在这所学校做访问学者。两人决定设计自己的电路板，并邀请 Massimo Banzi 的学生 David Mellis 为电路板设计编程语言。两天以后，David Mellis

就写出了程序代码。又过了三天，电路板就完工了。Massimo Banzi 喜欢去一家名叫 di Re Arduino 的酒吧，该酒吧是以 1000 年前的意大利国王 Arduin 的名字命名的。为了纪念这个地方，他将这块电路板命名为“Arduino”。

随后 Banzi、Cuartielles 和 Mellis 把设计图放到了网上。版权法可以监管开源软件，却很难用在硬件上，为了保持设计图的开放源码理念，他们决定采用 Creative Commons（CC，知识共享）的授权方式公开硬件设计图。在这样的授权下，任何人都可以生产电路板的复制品，甚至还能重新设计和销售原设计的复制品。人们不需要支付任何费用，甚至不用取得 Arduino 团队的许可。然而，如果重新发布了引用设计，就必须声明原始 Arduino 团队的贡献；如果修改了电路板，则最新设计必须使用相同或类似的 Creative Commons 的授权方式，以保证新版本的 Arduino 电路板也是自由和开放的。被保留的只有 Arduino 这个名字，它被注册成了商标，在没有官方授权的情况下不能使用它。

Arduino 经过十几年的发展，已经有了多种型号及众多衍生控制器。

1.3　Arduino 的主要特点

Arduino 如此流行，肯定有其独到之处，其主要特点如下：

1）跨平台

Arduino IDE 可以在 Windows、Macintosh OS X、Linux 三大主流操作系统上运行，而其他的大多数控制器只能在 Windows 上开发。

2）简单清晰易掌握

对于初学者来说，Arduino 极易掌握，同时有着足够的灵活性。初学者不需要太多的单片机基础、编程基础，简单学习后，就可以快速使用 Arduino 电路板进行开发。

3）开放性

Arduino 的硬件原理图、电路图、IDE 软件及核心库文件都是开源的，在开源协议范围内可以任意修改原始设计及相应代码。

4）发展迅速

Arduino 不仅是全球流行的开源硬件，也是一个优秀的硬件开发平台，更是硬件开发的趋势。Arduino 简单的开发方式使得开发者能够关注其创意与实现，更快地完成自己的项目开发，从而大大节约了学习成本，缩短了开发周期。

1.4　Arduino 的应用场景

Arduino 是一个极其灵活的开源电子原型平台，它允许开发者创建各种交互式产品。利用 Arduino 可以轻松读取大量的开关和传感器信号，控制多种电器，如灯光、电机及其他物理装置。

Arduino 的应用范围广泛，从电子时钟、电子显微镜、宠物喂养机、3D 打印机、无人机、智能

小车，到智能家居系统，它的能力几乎只受作品创造者想象力的限制。作为创客社区的热门选择，Arduino 平台支持连接多种传感器，可接收信号以监测外部环境，也能通过诸如电机和电源等输出设备来实际操作或改变这些环境。

在教育领域，Arduino 也非常受欢迎，成为许多青少年编程和电子学习项目的入门平台。许多非电子专业的大学课程也纳入了 Arduino，使得拥有好点子的非电子专业学生也能将自己的创意实现成具体的产品。另外，Arduino 以其开源和高度扩展性的特点，成为实施机器人教育的理想平台之一。与其他积木式搭建平台相比，Arduino 能够提供更多自定义的可能性，激发学生的创新和创造力。

第 2 章

搭建 Arduino 开发环境

既然是开发 Arduino 程序，那么首先需要一个 Arduino 集成开发环境。集成开发环境是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具，是集成代码编写、分析、编译、调试等功能于一体的开发软件服务套。所有具备这一特性的软件或者软件套（组）都可以叫作集成开发环境，如微软的 Visual Studio 系列，Borland 的 C++ Builder、Delphi 系列等。该程序可以独立运行，也可以和其他程序并用。IDE 多被用于开发 HTML 应用软件。例如，许多人在设计网站时使用 IDE（如 HomeSite、DreamWeaver 等），因为很多项任务会自动生成。

对于初学者而言，最常用的操作系统是 Windows，因此本书以在 Windows 下搭建 Arduino 开发环境为主。为了照顾一部分 Windows7 系统的使用者，笔者这里采用 Windows7 操作系统。通常而言，如果 Windows7 上的环境搭建起来了，那么 Windows10 也是可以的，反之则不一定。

2.1 下载和安装 Arduino IDE

可以到官方网站 https://www.Arduino.cc/去下载 Arduino IDE。打开网站首页，在上部单击 SOFTWARE 标签进入下一个页面，或者直接打开 https://www.arduino.cc/en/software，然后在该页面右边找到“Windows MSI installer”，如图 2-1 所示。

单击“Windows MSI installer”开始下载。这里下载下来的文件是 arduino-ide_2.3.2_Windows_64bit.msi，这是个安装包。如果不想下载，也可以到随书源码根目录下的“somesofts”子目录下找到该安装包。

直接双击安装包开始安装。安装完毕后，在桌面上会生成一个名为“Arduino IDE”的图标，如图 2-2 所示。

图 2-1

图 2-2

双击该目标就可以启动 Arduino IDE 程序，刚启动时候，IDE 默认是黑色界面，如果不喜欢黑色界面，可以在菜单栏依次单击“File”→“Preferences...”，然后在“Preferences”对话框中设置“Theme”为“Light High Contrast”，也就是设置颜色主题为浅色高对比（当然也可以选择自己喜欢的颜色方案），再单击“OK”按钮，此时 IDE 界面就明亮了，如图 2-3 所示。图中默认打开了一个源代码编辑窗口，这个源代码文件 sketch_mar17a.ino 也是默认产生的，并且自动建立了两个函数 setup 和 loop。如果读者学过 C 语言编程，应该知道这两个函数是什么意思。

图 2-3

另外，第一次启动的时候，IDE 会下载一些软件包，并且会提示安装一些设备软件，如图 2-4 所示。直接单击“安装”按钮即可进入安装过程。

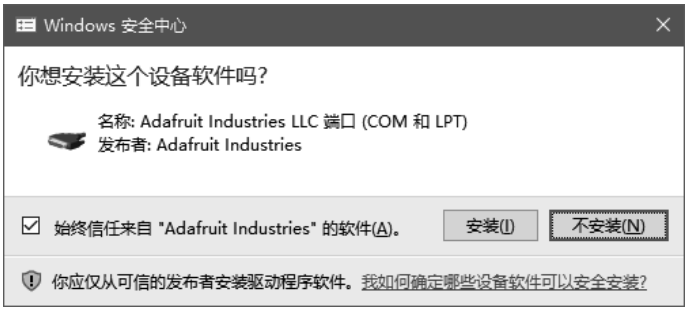

图 2-4

2.2 设置 Arduino IDE 中文界面

默认情况下，Arduino IDE 的窗口界面都是英文界面，比如菜单栏是英文，工具栏按钮提示也是英文。为了照顾英文不好的用户，Arduino IDE 提供了中文界面设置方法：在菜单栏依次单击“File”

→“Preferences...”，此时将显示“Preferences”对话框，如图 2-5 所示。

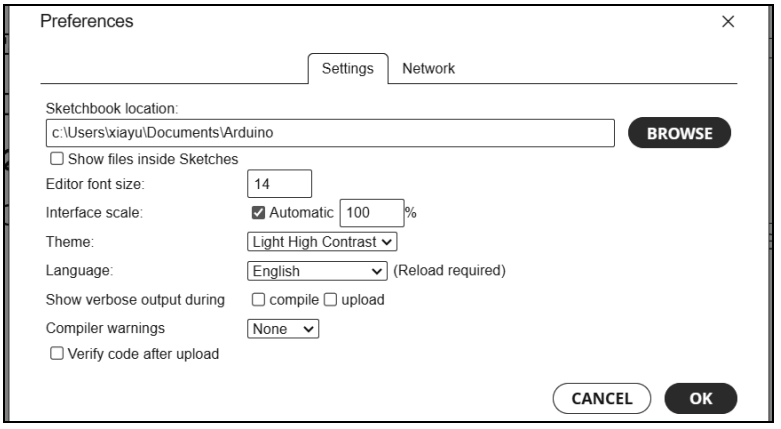

图 2-5

在该对话框上，展示“Language”下拉菜单，选择“中文（简体）”，如图 2-6 所示。

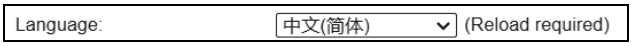

图 2-6

然后单击右下角的“OK”按钮，稍等片刻，IDE 界面就变为中文了，如图 2-7 所示。

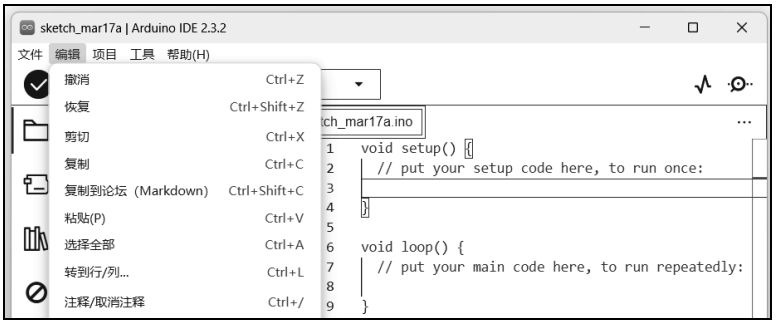

图 2-7

2.3　Arduino IDE 界面简介

我们设置了 Arduino IDE 为中文界面后，接下来就可以简单认识一下 IDE 上的各个界面元素。Arduino IDE 2 是一个多功能的编辑器，具有许多功能：可以直接安装库、与 Arduino Cloud 同步草图、调试草图等。在本节中将列出其核心功能。

和大多数 Windows 应用程序一样，Arduino IDE 窗口上也有标题栏、菜单栏和工具栏，只不过因为它是用来开发软件的，所以还多了代码编辑区和输出窗口，如图 2-8 所示。

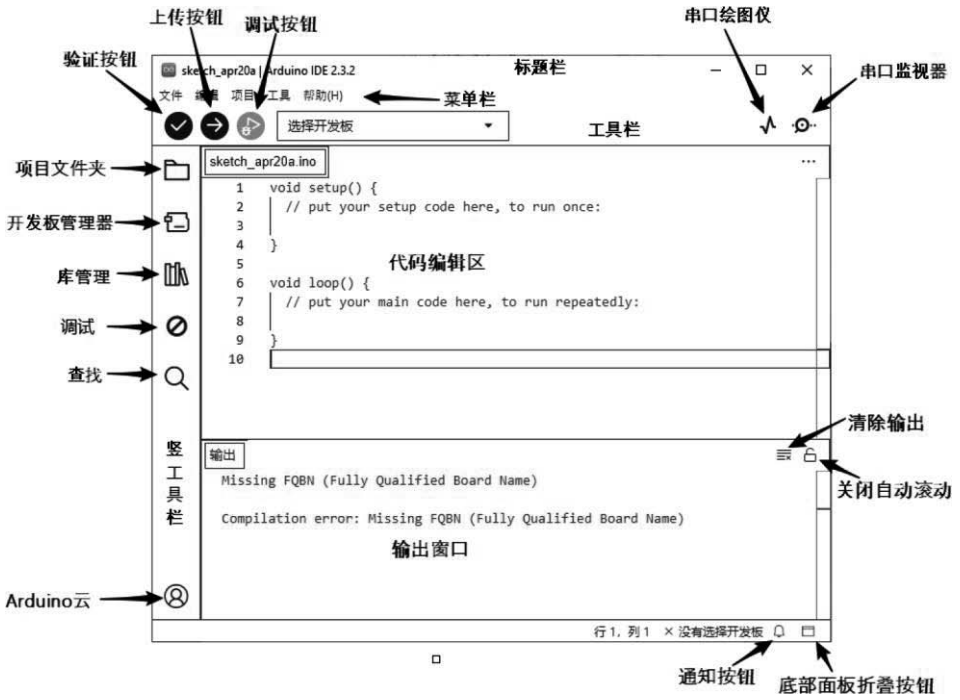

图 2-8

在图 2-8 中，输出窗口平时不显示，通常要在单击“验证”按钮或“上传”按钮后才会显示，其作用是显示编译的结果，比如编译错误信息。

2.3.1 标题栏

标题栏上通常会显示当前打开的源码文件的文件名，比如图 2-8 中的 sketch_apr20a，但没有包括文件后缀名。Arduino 源码文件的默认后缀名是.ino，因此在硬盘上存储的源码文件是 sketch_apr20a.ino。标题栏上除了显示源码文件名之外，还会显示 Arduino IDE 的版本号，比如这里的 Arduino IDE 2.3.2。

2.3.2 菜单栏

菜单栏上有 5 个菜单项，即“文件”“编辑”“项目”“工具”和“帮助”。

1. “文件”菜单

“文件”菜单下的子菜单通常用来打开、保存、关闭源码文件。编写源码之前，我们肯定需要打开源码文件，然后进行编辑，编辑完毕后再进行保存，这样下次打开的就是最新编辑过的源码文件了。另外，IDE 本身的一些特性设置在“文件”菜单的“首选项”子菜单中，这也是一个常用的功能，就像上一节设置中文界面，就要单击“首选项”子菜单。

2. “编辑”菜单

“编辑”菜单下的子菜单主要是和文本编辑有关的功能，代码编辑其实也就是文本编辑，和我

们平时在 Word 上打字区别不大，也会有复制、剪切、粘贴、删除、撤销、全选、查找等功能。

3. “项目”菜单

“项目”菜单下面的子菜单主要用于代码的编译、上传和调试。编译是将源码文件转换成二进制文件或库的过程，是一个将高级语言代码转换为计算机能够理解和执行的机器语言代码的过程。编译出来的二进制文件再上传到 Arduino 开发板中去运行。通常把编译和运行不在同一台机器上（编译在 x86 主机，运行在 Arduino 开发板中）的编译方式叫作交叉编译，也就是说，交叉编译是指在一台计算机上进行编译，生成可以在另一台不同架构的计算机或设备（比如 Arduino 开发板、嵌入式开发板）上运行的程序。

运行程序时经常会发生错误，结果和预期不一致，此时就需要进行调试（Debug）。调试是指在软件开发过程中，通过对程序进行检测、定位和修正来解决程序错误的过程。调试是软件开发的重要环节，它可以帮助开发人员找出程序中的错误，并对其进行修复，从而提高程序的质量和稳定性。在软件开发中，调试通常包括以下几个步骤：

（1）观察程序运行过程中的现象和错误表现，根据错误信息、日志等分析程序的问题所在。

（2）利用调试工具，在程序运行时暂停程序的执行，查看程序的状态，包括变量的值、函数的调用栈、线程的状态等，以确定程序的运行状态。

（3）对程序进行修改和调整，加入调试语句输出等，以便更好地观察程序的执行过程。

（4）重新编译和运行程序。重复以上步骤，直到问题被解决为止。

调试是软件开发过程中不可避免的部分。在程序出现错误时，调试是找到和解决问题的有效手段。开发人员应该掌握一些调试技巧和工具，如断点调试、单步调试、条件断点等，以便更加高效地解决程序错误。一旦软件项目规模大了，调试就是必需的，且最重要、最耗时的开发工作。有时候因为一个程序 bug，熬夜加班乃是家常便饭。

4. “工具”菜单

“工具”菜单是一个与 Arduino 开发板相关的工具和设置的集合，主要包括如下这些子菜单。

1）“自动格式化”子菜单

该子菜单可以整理代码的格式，包括缩进、括号，使程序更易读和更规范。

2）“项目存档”子菜单

该子菜单将程序文件夹中的所有文件整合到一个压缩文件（.zip）中，以便将文件备份或者分享。

3）“管理库”子菜单

“库”（Library）在编程中是个基础的概念。简单地说，在编程时，有些特定的功能模块、函数、常量等经常被重复利用，为了方便二次开发和利用，将这些功能模块抽象打包，并提供特定的接口（即使用方法），这样的功能组合就可以称为库。“管理库”子菜单用来对库进行管理，它与竖工具栏上的“库管理”按钮功能相同。通常，使用竖工具栏上的按钮更加方便。

4）“串口监视器”子菜单

这是 Arduino IDE 自带的一个小工具，可以查看串行端口（简称串口）传来的信息，也可以向连

接的 Arduino 开发板发送信息。该子菜单是一个非常实用而且常用的选项，类似即时聊天的通信工具，个人计算机（PC）与 Arduino 开发板连接的串口“交谈”的内容会在该串口监视器中显示出来。

5）串口绘图仪

Arduino IDE 具有丰富的串口绘图仪，用于跟踪从 Arduino 开发板接收的不同数据和变量。串口绘图仪是一个非常有用的视图工具，可以帮助我们更好地理解和比较数据。它还可以用于测试和校准传感器、比较数值等，如图 2-9 所示。

图 2-9

6）“烧录引导程序”子菜单

引导程序（BootLoader）是在系统上电或复位后运行的一段小程序。这段程序将系统的软硬件环境带到一个合适的状态，为最终调用应用程序准备好正确的环境，就像计算机的 BIOS 程序。BootLoader 是严重依赖于硬件而实现的，特别是在嵌入式系统中难以建立一个通用的 BootLoader。BootLoader 一上电就拿到单片机的控制权，实现了硬件的初始化。这个子菜单将烧录引导程序到 AVR 单片机中。

5. “帮助”菜单

“帮助”菜单主要提供了“入门”“环境”“故障排除”“参考”和“常见问题”等子菜单，单击它们，将跳转到官网的相关主题内容下，但都是英文介绍且讲得比较笼统，所以平时很少使用。

2.3.3 工具栏

相对于菜单栏，平时用得更多的是工具栏，因为工具栏上的按钮直接单击就可以了，不像菜单栏，需要一级菜单、二级菜单地逐级查找。Arduino IDE 的工具栏有两个，一个是在菜单栏下面的横行工具栏（一般简称工具栏），另外一个是在 IDE 窗口左边的竖形工具栏。

横行工具栏上包括的按钮说明如下：

（1）“验证”按钮：编译代码。
（2）“上传”按钮：上传编译后的二进制程序到 Arduino 开发板。
（3）“调试”按钮：实时测试和调试程序，必须已经连接 Arduino 开发板。

（4）“选择开发板”下拉框：检测到的 Arduino 开发板会自动显示在这里，并显示端口号。

（5）“串口绘图仪”和“串口监视器”按钮：它们和“工具”菜单下的子菜单“串口绘图仪”和“串口监视器”的功能一致，只不过是不同的入口而已。

竖工具栏（图 2-8 所示的最左侧）包括的按钮说明如下：

（1）“项目文件夹”按钮：通常我们把一个项目内的所有文件放在一个文件夹中，因此就有了项目文件夹。单击“项目文件夹”按钮将打开项目文件夹视图，我们可以在这个视图上管理（新建、删除等操作）项目中的文件，如图 2-10 所示。现在我们并没有新建项目，所以是一片空白。

（2）“开发板管理器”按钮：单击该按钮将打开“开发板管理器”视图，在这个视图上可以浏览和 Arduino 开发板有关的软件包，我们可以对这些软件进行安装或移除。

（3）“库管理”按钮：和“管理库”子菜单功能相同，单击该按钮会打开“库管理”视图，如图 2-11 所示。

图 2-10

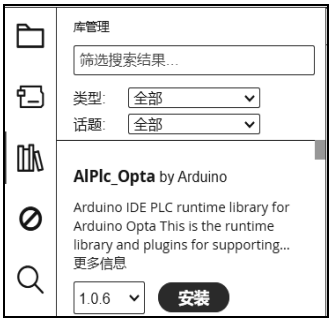

图 2-11

（4）“调试”按钮：单击该按钮将打开“调试”视图，如图 2-12 所示。通过这个视图，我们可以查看调试过程中的变量、调用堆栈的内容。

（5）“搜索”按钮：单击该按钮将打开“搜索”视图，让用户可以在源码中搜索内容，示例如图 2-13 所示。

图 2-12

图 2-13

（6）“Arduino 云”按钮：用于将代码同步到云端服务器上。对于在多台计算机上工作或想把代码安全地存储在云端的用户来说，Remote Sketchbook 的集成是一个非常有用的功能。我们在 Arduino Cloud 和 Arduino Web Editor 中编写所有代码都可以在 IDE 2 中进行编辑，这使得我们可以

轻松地从一台计算机切换到另一台计算机并继续工作。即使没有在所有的机器上都安装 Arduino IDE 2 也没关系，只要打开 Arduino Web Editor，就可以在在线 IDE 中通过浏览器进行代码编写，并可以访问所有代码和库。有了 Remote Sketchbook，再也不用担心丢失写好的代码了，只需要单击，它们就会被安全地推送到 Arduino 云端。

Remote Sketchbook 还支持先脱机工作，稍后同步，只需将代码从云端下载下来就可以进行离线编辑，等有网络时再推送，代码的修改部分就会上传，从而保证所有代码始终都是最新的，并随时可以使用。

2.3.4 代码编辑器

代码编辑器就是编写代码的地方，它是一个文本编辑器，但功能比普通文件编辑器要强得多。代码编辑器中比较常用的功能有以下 3 个。

（1）变量或函数导航功能：也就是当右击一个变量或函数时，将会提供导航快捷键，跳到它们被声明的行（和文件），如图 2-14 所示。

（2）代码自动补全功能：在输入文字时，编辑器可以根据我们输入的代码和包含的库给出一个提示框，里面会建议自动完成变量和函数，这样就减轻了我们的记忆负担。但该功能默认情况下不开启，我们可以自己开启。在菜单栏依次单击“文件”→“首选项”，然后在“首选项”对话框上勾选“编辑快速建议”复选框，如图 2-15 所示。

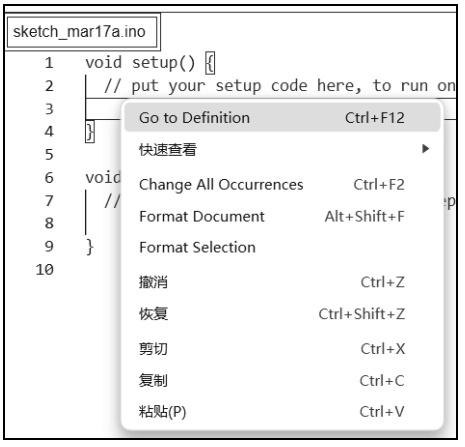

图 2-14

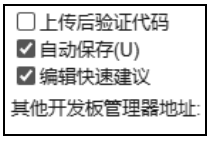

图 2-15

（3）黑夜模式：这也是现代化开发工具的一个标配了，比如大名鼎鼎的 Visual Studio Code 默认就是黑色的，或许是因为黑色不刺眼吧。如果我们的眼睛感觉累了，可以尝试切换到黑夜模式。一些读者可能在 Beta 版期间使用过这个功能，不过这次，Arduino 的设计团队重新设计了整个黑夜主题，使其更加一致、美丽和易于观看。

Arduino IDE 2.3.2 版本的 IDE 默认采用黑夜模式，如果不喜欢，可以在颜色主题旁选择自己喜欢的颜色。如果要设置白色，可以在菜单栏依次单击“文件”→“首选项”，然后在“首选项”对话框的“颜色主题”下拉菜单中选择“浅色高对比度”，单击“确定”按钮，IDE 就变为白色了，如图 2-16 所示。

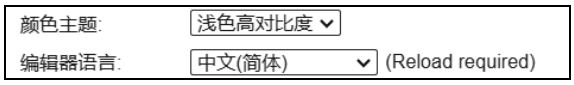

图 2-16

2.4　连接开发板并安装驱动

介绍完 Arduino IDE 软件后，接下来就要介绍 Arduino 开发板这个硬件了，本书使用的是 Arduino UNO R3 版本。虽然我们是在计算机上编写和编译 Arduino 程序，但运行和调试都需要在 Arduino 开发板中进行。因此，需要将 Arduino 开发板用线连接到计算机。首先确保计算机已经开机并进入 Windows 系统了，然后取出 Arduino 开发板和配套的 USB 数据线，将数据线的方形口插入 Arduino 开发板的方形口，将数据线的 USB 接口插入计算机的 USB 接口。连接示意图如图 2-17 所示。

图 2-17

此时在计算机屏幕右下角会出现安装驱动的提示，说明计算机探测到有设备插入了，但由于没有安装该设备的驱动程序，因此会在设备管理器中对该设备打上一个感叹号。我们打开计算机的设备管理器就可以看到，如图 2-18 所示。

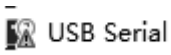

图 2-18

这个设备是一个 USB 转串口的芯片，该芯片已经集成在 Arduino 开发板上了，其作用就是实现 USB 转串口。现在的计算机，尤其笔记本电脑上已经没有传统的那种九针串口了，通常都只有 USB 接口。九针串口也就是 RS-232 接口，是个人计算机上的通信接口之一，是由电子工业协会（Electronic Industries Association，EIA）所制定的异步传输标准接口。它和 USB 接口的外观区别如图 2-19 所示。

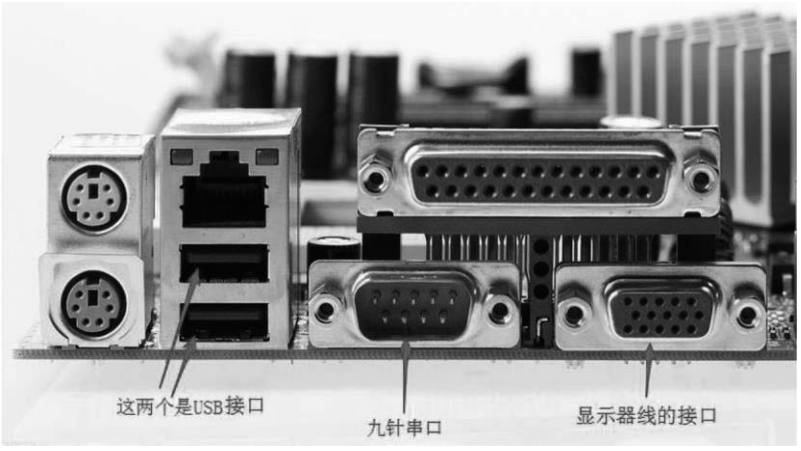

图 2-19

九针串口所插的串口线如图 2-20 所示。随着技术的进步，传统的九针串口已逐渐被淘汰。然而，在嵌入式开发领域，尤其是在 Arduino 等开发板与计算机之间的数据通信中，串口依然是最常用的通信接口。面对现代计算机普遍不再配备九针串口的现实，人们转而发明了 USB 转接芯片，以实现 USB 转串口。该芯片实物图如图 2-21 所示。

图 2-20

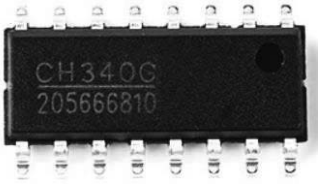

图 2-21

通过 CH340 转接芯片，可以将 USB 总线信号转为异步串口信号，或将并口（并行端口）打印机转为 USB 打印机，如图 2-22 所示。

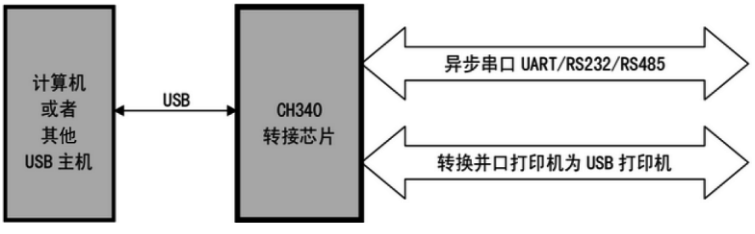

图 2-22

这样开发板上的单片机还以为和计算机相连的是传统的串口线呢。这些我们了解即可。这个转接芯片使得人们在软件的使用上和以前使用九针串口没区别，但我们需要在计算机上为这个芯片安装驱动，然后才能使用。

安装驱动的过程很简单，在随书源码的根目录下找到 somesofts 文件夹，然后在这个文件夹里找到子文件夹“usb 转串口驱动”，进入后再找到“CH341SER.exe”这个安装文件并右击，在弹出的快捷菜单上选择“以管理员身份运行”，如图 2-23 所示。

图 2-23

此时出现“DriverSetup(X64)”（驱动安装）对话框，如图 2-24 所示。

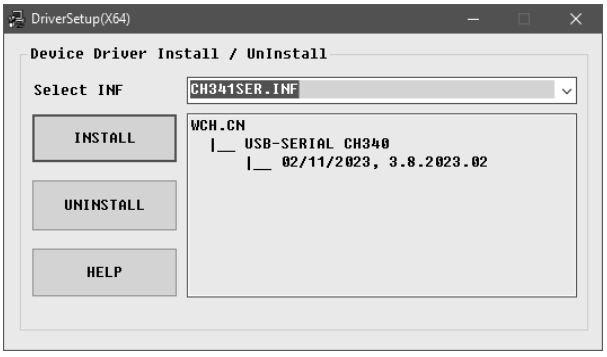

图 2-24

单击“安装”按钮，稍等片刻，出现安装成功的对话框，如图 2-25 所示。

单击“确定”按钮，然后关闭“驱动安装”对话框。我们再到设备管理器中去查看，可以发现感叹号没有了，而在“端口（COM 和 LPT）”下新增了一个“USB-SERIAL CH340（COM3）”，如图 2-26 所示。

图 2-25

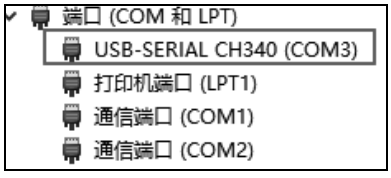

图 2-26

这就说明驱动安装成功了，并且可以知道这个 Arduino 开发板使用的 USB 转接芯片就是 CH340，占用的串口号是 COM3（不同的计算机环境，可能这个串口号不同）。串口号就是标记串口的一个名称，也就是说，计算机以为自己现在拥有了一个串口了，于是给这个串口一个名称，即 COM3。驱动程序除了做计算机和设备之间的数据通路之外，还经常“忽悠”计算机（确切地说应该是操作系统）。

至此，计算机和 Arduino 开发板之间的通信桥梁架设完毕，以后我们编译出来的程序就可以通过这个“桥梁”传到 Arduino 开发板的单片机中运行了。

2.5 验证开发环境

我们已经把 Arduino 开发环境建立好了，但好不好用、有没有问题，还必须通过实践才能得到答案。因此，趁热打铁，我们开始第一个 Arduino 程序。这个程序很简单，就是点亮开发板上的灯，然后熄灭它，再点亮它，再熄灭它，如此反复。

2.5.1 第一个 Arduino 程序

【例 2.1】第一个 Arduino 程序

（1）打开 Arduino IDE，此时会自动帮我们建好了一个.ino 源码文件，并且里面有 setup 和 loop 两个空函数。这个自动建立的文件在哪里呢？我们可以通过菜单栏依次单击“项目”→“显示项目文件夹”来查看。它存放在一个临时目录文件夹中：

```
    C:\Users\Administrator\AppData\Local\Temp\.arduinoIDE-unsaved2023730-12196-dfmycm.hn2ep\
```

如果不喜欢这么长的目录，可以另存到其他路径。首先在 D 盘下新建一个名为“myar”的文件夹，然后在菜单栏依次单击“文件”→“另存为”，此时出现“另存为”对话框，定位到 D:\myar，然后在“文件名”旁输入 test，如图 2-27 所示。

这个 test 就是项目文件夹的新名称，相应的源文件名称也变为“test.ino”。单击“保存”按钮，此时 IDE 中的源文件名变为“test.ino”了，如图 2-28 所示。

图 2-27

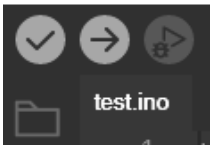

图 2-28

我们再到 D:\myar 下去查看，发现多了一个 test 文件夹，进入该文件夹，里面有一个 test.ino，这是源码文件，我们在 IDE 编辑框中输入的代码，就会保存到 D:\myar\test\test.ino 中。如果现在关

闭 IDE，然后到 D:\myar\test\下直接双击 test.ino，则 IDE 将启动并打开 test.ino 文件。以后的实例，如果不特别说明，都基于 D:\myar\test\test.ino 来编写程序。因此，以后实例会直接这样说：打开 test.ino、在 test.ino 中输入代码等。这个 test.ino 就位于 D:\myar\test\下。当然读者也可以保存在其他地方。

（2）在 test.ino 中添加一些代码。首先在 setup 函数前定义一个全局变量，并在 setup 函数中添加一行代码，如下所示：

```
int gPin=13;  //表示引脚 13，以后如果要改为其他引脚，只需要在这里修改即可
void setup() {
    //将 setup 代码放在这里，只运行一次
    pinMode(gPin,OUTPUT);  //这是我们添加的代码，作用是设置引脚 13 为输出模式
}
```

setup 函数在开发板上电后只执行一次，通常把一些只需要执行一次的初始化操作放在这个函数中。我们添加的函数是 pinMode，该函数用于设置特定引脚的工作模式，它接收两个参数：引脚号和工作模式，即 13 表示编号为 13 的引脚，OUTPUT 表示设置该引脚的工作模式为输出，简单地说就是设置引脚 13 为输出模式。

引脚又叫管脚，英文叫 Pin，它是从集成电路（芯片）内部电路引出与外围电路的接线，所有的引脚构成了这块芯片的接口。每个引脚都有特定的功能和用途，通过引脚，芯片可以与其他电子元件进行通信和交互。引脚的作用是将芯片内部的电子信号传输到外部电路，或者将外部电路的信号输入芯片内部。它们充当了芯片与外界之间的桥梁，起到了信号传输和数据交换的作用。相应地，引脚的工作模式至少有输入模式（INPUT）和输出模式（OUTPUT），现在我们使用的是 OUTPUT，也就是输出芯片内部的电子信号，通俗地讲就是输出电流，即当引脚设置为输出（OUTPUT）模式时，Arduino 可以向其他电路元件提供电流。也就是说，如果这个引脚连接了一个 LED 灯，那么引脚在输出（OUTPUT）模式下可以点亮该 LED 灯。

下面，我们再在 loop 函数中添加代码：

```
void loop() {
    //将主函数代码放在这里，以便重复运行
    digitalWrite(gPin,HIGH);     //向引脚 13 输出高电平，此时小灯被点亮
    delay(1000);                 //等待 1000ms，也就是等待 1s
    digitalWrite(gPin,LOW);      //向引脚 13 输出低电平，此时小灯被熄灭
    delay(1000);                 //再等待 1s
}
```

我们在 loop 函数中添加了 4 个函数，其基本功能是向引脚 13 输出高电平，等待 1s，再向引脚 13 输出低电平，再等待 1s。由于 loop 函数会反复执行，因此这 4 个函数会一直重复执行下去。也就是说，执行完最后一个 delay(1000);后，又从 digitalWrite(gPin,HIGH);开始执行，如此无限循环。那么，如此反复执行后会有什么效果呢？先不急着看实际效果，我们看代码中的注释，digitalWrite(gPin,HIGH);这个函数旁的注释是“向引脚 13 输出高电平，此时小灯被点亮”。前半句好理解，我们看函数就知道是向引脚 13 输出高电平，那么为何会点亮 LED 灯呢？首先要知道，Arduino UNO 开发板上的引脚 0～13 用作数字输入/输出（I/O）引脚，其中引脚 13 连接到板载的 LED 指示灯，这是开发板在设计时就规定好了的。我们可以仔细看开发板上有 L 的地方，如图 2-29 所示。

这个 L 就是 LED 的首字母，表示是一个 LED 灯，而且这个 LED 灯是接在引脚 13 上的。因此

可以推断出亮的灯就是字母“L”旁边的 LED 灯。那又有疑问，为什么输出高电平是点亮 LED 灯？其实这没有为什么，就是这样设计的，我们也可以设计成低电平点亮 LED 灯。

LED 灯有两种连线方法，第一种方法是当 LED 灯的阳极通过限流电阻与开发板上的数字输入/输出接口（也就是数字输入/输出引脚）相连时，引脚输出高电平，LED 灯导通，发光二极管发出亮光；数字引脚输出低电平时，LED 灯截止，发光二极管熄灭。另外一种方法就是当 LED 灯的阴极与开发板上的数字输入/输出接口（数字输入/输出引脚）相连时，引脚输出高电平，LED 灯截止，发光二极管熄灭；数字引脚输出低电平，LED 灯导通，发光二极管点亮。Arduino UNO 开发板的 LED 灯采用的是第一种方法，即高电平点亮，我们现在记住这 5 个字即可。

好了，现在我们把 digitalWrite(gPin,HIGH);搞清楚了，它就是让字母 L 旁边的 LED 灯亮起来。然后间隔 1s，再调用 digitalWrite(gPin,LOW);，此函数让引脚 13 输出低电平，那么 LED 灯就熄灭了。最终效果就是字母 L 旁的 LED 灯每间隔 1s 点亮和熄灭，如此循环反复。

（3）运行程序。首先确保 Arduino 开发板已经和计算机连接好，然后在 Arduino IDE 工具栏上的“选择开发板”下拉框中选择串口号。这个串口号是 Arduino 开发板连接到计算机上后，计算机自动分配的，可以在设备管理器“端口”下看到，并且每次分配的串口号几乎都不同，比如笔者现在的串口号是 COM6（不同的计算机，这个串口号可能不同），因此选择 COM6，如图 2-30 所示。

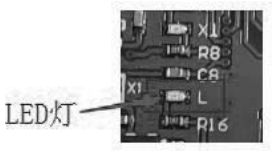

图 2-29

图 2-30

单击“COM6”后会出现“选择其他开发板和端口”对话框，在该对话框上搜索 uno，然后选中“Arduino Uno”，如图 2-31 所示。

图 2-31

单击“确定”按钮，这样开发板和串口就选择好了，IDE 工具栏的下拉框中的标题变为“Arduino Uno”了，如图 2-32 所示。

图 2-32

现在，单击工具栏上的上传按钮（就是横向箭头的那个按钮），此时 IDE 右下角会依次出现“正在编译”“正在上传”和“上传完成”等提示，并且输出窗口也会出现一些项目所用的存储空间信息，如图 2-33 所示。

图 2-33

这就说明我们编译上传成功了。这个时候，应该可以看到开发板上的 L 字母旁的 LED 灯每隔 1s 闪烁一次了，如图 2-34 所示。

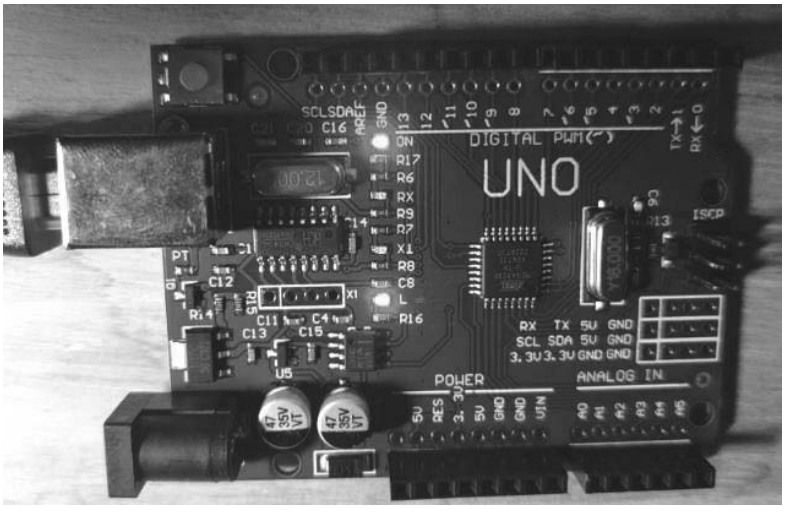

图 2-34

这个效果和我们预期的一样。下面我们来了解 Arduino 编程的基本方式。Arduino 编程本质上也是单片机编程，单片机编程的基本方式是先做一些初始化工作，比如初始化变量、设置针脚的输出/输入类型、配置串口、引入类库文件等，这些初始化工作在每次程序开始运行时做一次即可；然后就进入一个循环中反复执行某些操作，即使没有事情干，也要空循环。这就是单片机编程的特点。Arduino 编程也是如此，它提供了一个 setup 函数，在里面添加初始化的代码；另外提供了一个 loop 函数，在里面添加反复执行的工作代码，就像本例中的亮灯、灭灯。这种编程方式总结成一句话：初始化和循环。

再多说一句，有没有发现这个例子中的代码和 C 语言代码有点相似。其实，Arduino 使用的编

程语言主要是基于 C/C++语言的一种简化版本，称为 Arduino 语言或 Wiring 语言。Arduino 语言在 C/C++的基础上进行了一些简化和封装，使得用户可以更加轻松地进行硬件编程。库大部分是 C++语言。Arduino 的 C/C++语言编译环境是基于 gcc 的一个衍生版本 gcc-avr 修改而来的。

2.5.2 数字引脚和数字电平

数字信号使用二进制形式来表示信息，每个位（bit）可以是 0 或 1，这通常对应于两种不同的电压水平。Arduino 上的数字引脚根据用户需求设计为输入或输出的引脚。数字引脚可以打开或关闭。打开时，它们处于 5V 的高电平状态；关闭时，它们处于 0V 的低电平状态。也就是说，数字引脚上要么是 5V（高电平），要么是 0V（低电平），只有这两种状态。

前面例子中引脚 13 就是一个数字引脚。在 Arduino 上，当数字引脚配置为输出时，它们设置为 0V 或 5V。因此，函数 digitalWrite(13,HIGH);也可以这么注释：打开数字引脚 13。以后听到或看到打开数字引脚，就知道这个数字引脚处于高电平状态。而函数 digitalWrite(13,LOW);可以注释为关闭数字引脚 13，即该数字引脚处于低电平状态。在计算机中，用数字来表征一样东西，其含义通常就是要么是这种状态，要么是另一种状态。

Arduino UNO 的引脚 0～13 用作数字输入/输出引脚，其中引脚 13 连接到板载的 LED 指示灯，引脚 3、5、6、9、10、11 具有 PWM（脉冲宽度调制）功能。

当数字引脚配置为输入时，电压由外部设备提供。该电压可以在 0～5V 范围内变化，并被转换为数字形式（0 或 1）。为了保障这一点，设置了两个阈值：低于 0.8V 的电压被识别为 0；高于 2.0V 的电压被识别为 1。

将组件连接到数字引脚时，要确保逻辑电平匹配。如果电压在阈值之间，则返回值将不确定。

2.6 串口打印

上一节的实例成功了，说明我们搭建的开发环境没有问题，本节来验证一下 IDE 的另外一项功能——串口打印。串口打印也是一种调试手段，它能在运行过程中让我们看到某些变量的值或其他所需要的信息。本节的实例很简单，只需要在 2.5 节的实例的基础上添加两个函数即可。

串口用于 Arduino 开发板与计算机或其他设备之间的通信，所有 Arduino 开发板都至少有一个串口（也称为 UART 或 USART）。UART 的全称是 Universal Asynchronous Receiver/Transmitter，中文翻译为通用异步收发器，俗称串口，是一种用于芯片与 PC 之间或芯片与芯片之间的低速通信接口。

在 UNO、Nano、Mini 和 Mega 上，引脚 0 和 1 用于与计算机通信。将任何东西连接到这些引脚都可能干扰通信，甚至可能导致无法上传数据到开发板上。以常见的 Arduino UNO 为例，面板上只有两个串口，即引脚 0（RX）和 1（TX）。 计算机与 Arduino 的通信通过这两个串口进行，USB 接口通过一个转换芯片（通常为 ATmega16 u2）与这两个串口引脚连接，虽然表面上计算机没有直接用外置的电线与这两个引脚相连，但是二者之间的效果是一样的。当 Arduino 开发板使用 USB 线与计算机相连时，两者之间便建立了串口连接。通过此连接，Arduino 开发板可以与计算机相互传数据。通常一个串口只能连接一个设备进行通信。以上是硬件方面的一些基础知识，下面我们说说关

于串口的软件方面的知识。

串口打印输出是一个非常有用的调试技巧，我们可以在程序的某个地方加入串口打印语句，把需要查看的变量的值打印到串口输出窗口中，从而实时查看程序运行过程中变量值的变化。

Arduino 的类库提供 HardwareSerial 类来实现串口打印功能，该类继承 Stream 类，而 Stream 类又继承 Print 类。下面按照继承顺序，依次讲解 Print 类、Stream 类和 HardwareSerial 类。

2.6.1　Print 类

Print 类是一个非常有用的类，从名字就能看出来，这个类的作用是打印数据。不同的输出介质（如串口、LCD 液晶屏等）执行打印操作的过程基本相同，区别仅在于底层实现。底层实现的核心在于打印单个字符，但根据所使用的硬件，打印字符的具体方式会有所不同。例如，通过串口时，字符数据会按位发送；而在 LCD 液晶屏上打印字符时，则采用不同的方法。除此之外，其他部分都一样了。例如，打印一个字符串可以通过逐个字符的方式进行；同样，要打印数字 123，就需要依次打印“1”“2”和“3”这三个字符。这样的处理逻辑无论是应用于串口、LCD 屏幕还是其他任何硬件，都是相同的。因此，可以写一个类，给出共同部分的实现，而最底层实现写成虚函数，留给具体子类去实现。这个类就是 Print 类，它同时还包含了一些反映运行时状态（是否传送错误）的变量与方法。

接下来，让我们深入了解 Print 类中的一些关键成员函数。虽然 Print 类本身并不用于直接创建对象，但它的子类会频繁使用 Print 类提供的函数，因此了解这些函数的用法十分重要。其中，最核心的成员函数是 print，它能够将数据以可读的 ASCII 文本形式输出到串口。print 函数有多种重载形式，以适应不同数据类型的打印需求：

- 数字被转换成相应的 ASCII 字符后打印。
- 浮点数默认打印到小数点后两位。
- 字节数据作为单个字符发送。
- 字符和字符串则按原样输出。

print 函数声明如下：

```
    size_t print(const String &);                 //打印字符串
    size_t print(const char[]);                   //打印字符数组，也就是字符串
    size_t print(char);                           //打印单个字符
    size_t print(unsigned char, int = DEC);       //打印一个字节数据，默认是十进制
    size_t print(int, int = DEC);                 //打印一个整型数据，默认是十进制
    size_t print(unsigned int, int = DEC);        //打印一个无符号整型数据，默认是十进制
    size_t print(long, int = DEC);                //打印一个长整型数据，默认是十进制
    size_t print(unsigned long, int = DEC);       //打印一个无符号长整型数据，默认是十进制
    size_t print(long long, int = DEC);           //打印一个超长整型数据，默认是十进制
    size_t print(unsigned long long, int = DEC);//打印一个无符号超长整型数据，默认是十
进制
    size_t print(double, int = 2); //打印一个双精度浮点型的实数数据，默认保持 2 位小数
    size_t print(struct tm * timeinfo, const char * format = NULL);//打印时间
```

print 函数中，第一个参数是打印的内容，第二个参数用来指定打印的格式。如果第一个参数是整数，则第二个参数表示进制，取值是宏，定义如下：

```
#define DEC 10              //十进制
#define HEX 16              //十六进制
#define OCT 8               //八进制
#define BIN 2               //二进制
```

如果第一个参数是 double 型的实数，则第二个参数表示小数的位数个数。示例如下：

```
Serial.print(78, BIN)       //打印结果："1001110"，以二进制形式打印整数 78
Serial.print(78, OCT)       //打印结果："116"，以八进制形式打印整数 78
Serial.print(78, DEC)       //打印结果："78"，以十进制形式打印整数 78
Serial.print(78, HEX)       //打印结果："4E"，以十六进制形式打印整数 78
Serial.print(1.23456, 0)    //打印结果："1"，因为保留 0 位小数，所以打印结果为"1"
Serial.print(1.23456, 2)    //打印结果："1.23"
Serial.print(1.23456, 4)    //打印结果："1.2346"，保留 4 位小数
```

注意：真正打印时，没有双引号，这里加了双引号，是为了说明打印出来的内容都是字符。print 函数还会返回打印的字节数。

打印函数除了 print 之外，还有 println，它们的功能基本一样，只不过 println 输出完内容后，还会自动换行。

2.6.2 Stream 类

Stream 类继承 Print 类，它是二进制数据或者字符串数据流传输的基础类，不能直接被调用，但可以被继承。该类的重要成员函数介绍如下。

1）find 函数

该函数从串行缓冲器读取数据，直到找到目标为止。也就是说，该函数相当于一个搜索函数。该函数声明如下：

```
bool find(const char *target);   //从流中读取数据，直到找到目标字符串
bool find(uint8 t *target);      //从流中读取数据，直到找到字节数组
bool find(const char *target, size t length);  //从流中读取数据，直到找到给定长度的目标字符串
bool find(const uint8 t *target, size t length); //从流中读取数据，直到找到给定长度的目标字节数组
```

如果找到目标，函数将返回 true；如果超时，则返回 false。具体的超时时间可以通过成员函数 setTimeout 来设定。

2）findUntil 函数

该函数从串行缓冲器读取数据，直到找到给定长度的目标字符串或终止符字符串。也就是说，该函数有两种情况：一种是遇到目标字符串就停止搜索，另外一种是遇到终止符也停止搜索，这两种情况下函数都会返回 true。该函数声明如下：

```
bool findUntil(const char *target, const char *terminator);  //目标数据是字符串
bool findUntil(const uint8_t *target, const char *terminator);  //目标数据是字节数组
bool findUntil(const char *target, size_t targetLen, const char *terminate,
```

```
size t termLen); //为目标数据和终止数据指定了长度
    bool findUntil(const uint8 t *target, size t targetLen, const char *terminate,
size_t termLen);//为目标数据和终止数据指定了长度
```

3）setTimeout 函数

该函数用于设置等待串行数据的最大毫秒数，搜索函数就是基于这个时间来确定搜索的最长时间。默认值为 1000ms，即 1s。该函数声明如下：

```
    void setTimeout(unsigned long timeout);
```

参数 timeout 表示要设置的最大毫秒数。

4）getTimeout 函数

该函数用于获取等待串行数据的最大毫秒数。该函数声明如下：

```
    unsigned long getTimeout(void);
```

返回结果是等待串行数据的最大毫秒数。

5）parseInt 函数

该函数用于在传入序列中查找下一个有效整数。如果超时，该函数将终止。该函数声明如下：

```
    long parseInt();
```

该函数返回当前位置的第一个有效（长）整数值。如果发生超时（超时时间通过 Serial.setTimeout 来设置）时未读取有效数字，则返回 0。

6）parseFloat 函数

该函数返回串行缓冲区中的第一个有效浮点数，如果超时，该函数将终止。该函数声明如下：

```
    float parseFloat();
```

7）readBytesUntil 函数

该函数将串行缓冲区中的字符读取到数组中。如果已读取确定的长度、超时或检测到终止符字符串（在这种情况下，函数会将字符返回到提供的终止符之前的最后一个字符），则该函数会终止。缓冲区中不会返回终止符本身。该函数声明如下：

```
    size_t readBytesUntil(char terminator, char *buffer, size_t length);
    size_t readBytesUntil(char terminator, uint8_t *buffer, size_t length);
```

其中参数 terminator 表示终止符字符串；buffer 表示存放读取到的数据的缓冲区；length 表示要读取的数据的长度。函数返回读入缓冲区的数据的字节数，如果 length 小于或等于 0，或者出现超时，或者遇到终止符，则返回 0。注意：除非读取并复制到缓冲区的字符数等于长度，否则终止符字符串将从串行缓冲区中丢弃。示例代码如下：

```
    Serial.readBytesUntil(character, buffer, length);
```

8）readStringUntil 函数

该函数将串行缓冲区中的字符读取到字符串中。如果超时或遇到终止符，则函数将终止。该函数声明如下：

```
String readStringUntil(char terminator);
```

其中参数 terminator 是终止符。函数返回从串行缓冲区读取的字符串，直到遇到终止符，终止符将从串行缓冲区中丢弃。示例如下：

```
Serial.readStringUntil(terminator);
```

2.6.3 HardwareSerial 类

HardwareSerial 类通常被称为硬串口类，这是因为 Arduino 还有另一种串口类——SoftwareSerial，即软串口类。尽管 SoftwareSerial 也可用于串口通信，但它的使用相对较少。大多数 Arduino 开发板配备的是硬件实现的串口（硬串口），因此 HardwareSerial 的使用更为普遍。为了简便，HardwareSerial 类通常直接被称为“串口类”。HardwareSerial 类实现串口的读写，并在类的源文件中定义了一个对象 Serial：

```
HardwareSerial Serial(0);
```

其中 0 是传递给构造函数的参数。HardwareSerial 类的构造函数如下：

```
HardwareSerial(int uart_nr);
```

其中参数 uart_nr 表示要使用的串口序号，取 0、1、2 这 3 个值，0 表示开发板上的 0 号串口，也就是第一个串口；1 表示 1 号串口，也就是第二个串口；2 表示 2 号串口，也就是第三个串口。通常设备上只有一个串口，如果有多个，那么可以用对象 Serial1 来表示 1 号串口，Serial2 表示 2 号串口。这两个对象在 HardwareSerial 类源文件中也已经定义好了，代码如下：

```
#if SOC_UART_NUM > 1
HardwareSerial Serial1(1);
#endif
#if SOC_UART_NUM > 2
HardwareSerial Serial2(2);
#endif
```

我们直接使用即可。在 HardwareSerial 类的头文件中，也对这两个对象进行了声明：

```
extern HardwareSerial Serial;
#endif
#if SOC_UART_NUM > 1
extern HardwareSerial Serial1;
#endif
#if SOC_UART_NUM > 2
extern HardwareSerial Serial2;
#endif
```

因此，我们在应用程序中可以放心大胆地直接使用 Serial。

现在，我们来看一下 HardwareSerial 类的常用成员函数。

1）available 函数

该函数获取可从串口读取的字节数（字符数）。该函数声明如下：

```
int available(void);
```

函数返回可读取的字节数。

2）availableForWrite 函数

该函数获取在不阻止写入操作的情况下可在串行缓冲区中写入的字节数（字符数）。该函数声明如下：

```
int availableForWrite(void);
```

函数返回可写入的字节数。

3）begin 函数

该函数用于设置串行数据传输的速率，以“位/秒”即波特率（bps）为单位。要与串口监视器通信，请确保使用屏幕右下角菜单中列出的波特率中的一个。当然，我们也可以指定其他速率，例如通过引脚 0 和 1 与需要特定波特率的组件进行通信。该函数声明如下：

```
void begin(unsigned long baud, uint32_t config=SERIAL_8N1, int8_t rxPin=-1, int8_t txPin=-1, bool invert=false, unsigned long timeout_ms = 20000UL, uint8_t rxfifo_full_thrhd = 112);
```

除了第一个参数之外，其他参数都有默认值，因此我们一般只需要关注第一个参数，该参数就是要设置的波特率。第二个参数 config 用于设置数据、奇偶校验和停止位，可取的有效值如表 2-1 所示。

表 2-1　参数 config 可取的有效值

SERIAL_5N1	SERIAL_6N1	SERIAL_7N1	SERIAL_8N1	SERIAL_5N2	SERIAL_6N2
SERIAL_7N2	SERIAL_8N2	SERIAL_5E1	SERIAL_6E1	SERIAL_7E1	SERIAL_8E1
SERIAL_5E2	SERIAL_6E2	SERIAL_7E2	SERIAL_8E2	SERIAL_5O1	SERIAL_6O1
SERIAL_7O1	SERIAL_8O1	SERIAL_5O2	SERIAL_6O2	SERIAL_7O2	SERIAL_8O2

其中，SERIAL_8N1 是默认值；倒数第二个字符是 E 的都是偶校验，比如 SERIAL_5E1、SERIAL_8E1 等；倒数第二个字符是 O 的都是奇校验，比如 SERIAL_5O1、SERIAL_8O2 等。

通常该函数只需在开发板初始化时候调用一次即可，因此一般将它放在 setup 函数中，比如：

```
void setup() {
    Serial.begin(9600); //打开串口，将数据传输的速率设置为 9600 bps
}
```

4）end 函数

该函数用于禁用串行通信，从而允许 RX 和 TX 引脚用于一般输入和输出。要重新启用串行通信，请调用 serial.begin 函数。end 函数声明如下：

```
void end(bool fullyTerminate = true);
```

参数 fullyTerminate 表示是否完全停止，默认值是 true，即完全停止。示例如下：

```
Serial.end();
```

5）flush 函数

该函数等待传出串行数据的传输完成。该函数有两种形式，声明如下：

```
void flush(void);
void flush( bool txOnly);
```

一般我们只需用不带参数的形式即可，比如：

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

6）peek 函数

该函数用于读取接收缓存区中的第一个字节数据，并且不将其从接收缓存区中删除。也就是说，对 peek 的连续调用将返回相同的字符。该函数声明如下：

```
int peek(void);
```

函数返回传入串行数据的第一个可用字节，如果没有可用数据，则返回-1。

7）read 函数

该函数也用于读取接收缓存区中的第一个字节数据，但读取过的数据将从接收缓存中清除。该函数声明如下：

```
int read(void);
```

函数返回传入串行数据的第一个可用字节数据，如果没有可用数据，则返回为-1。示例如下：

```
int incomingByte = 0; //对于传入的串行数据
void loop() {
    //仅在接收到数据时发送数据
    if (Serial.available() > 0) {
        //读取传入的字节：
        incomingByte = Serial.read();
        //输出你收到了什么
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
    }
}
```

8）readBytes 函数

该函数将字符从串口读取到缓冲区中。如果读取了确定的长度或者超时，则函数终止。该函数声明如下：

```
size_t readBytes(uint8_t *buffer, size_t length);
size_t readBytes(char *buffer, size_t length);
```

其中参数 buffer 是用于存储字节的缓冲区；length 表示要读取的字节数。函数返回读到并放置在缓冲区中的数据的字节数。示例如下：

```
Serial.readBytes(buffer, length);
```

9）write 函数

该函数将二进制数据写入串口，此数据以字节或一系列字节的形式发送。write 函数声明如下：

```
size_t write(uint8_t n);
inline size_t write(const char * s);
```

```
size_t write(const uint8_t *buffer, size_t size);
```

其中参数 n 是要写入的单个字节数据；参数 s 表示要写入的字符串；参数 buffer 指向要写入数据的缓冲区；参数 size 表示要写入数据的字节长度。函数返回实际写入的字节数。

下面看一个实例，在串口监视器上输出字符串。我们把程序设计得足够简单，主要目的是测试开发板上的串口能否工作。

【例 2.2】在串口监视器上输出字符串

（1）连接好开发板，打开 Arduino IDE，此时会自动建好一个.ino 源码文件，并且里面有 setup 和 loop 两个空函数。选择开发板和串口号，在 setup 函数中添加如下代码：

```
void setup() {
    //将 setup 代码放在这里，只运行一次
    Serial.begin(9600);  //设置串口波特率为 9600
}
```

我们调用 begin 函数设置串口波特率为 9600，这个数值要和串口监视器右上角显示的波特率一致。打开串口监视器的方法是单击 IDE 右上角的工具栏中的“串口监视器”按钮，如图 2-35 所示。此时会在 IDE 下方出现“串口监视器”窗口，并且可以在该窗口的右上方看到波特率是 9600，如图 2-36 所示。

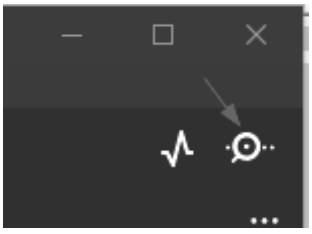

图 2-35

图 2-36

因此，我们通过 begin 函数设置 9600 波特率是没问题的。

（2）设置好波特率后，我们就可以在 loop 函数中调用 print 函数进行打印输出了，代码如下：

```
void loop() {
    //将主代码放在这里，以便重复运行
    Serial.print("hi,world.  ");     //向串口发送字符串
    delay(1000);                    //延时 1s
}
```

我们调用 print 函数向串口发送字符串"hi,world. "，在串口监视器中将可以看到这个字符串，并且由于 loop 函数是反复执行的，因此串口监视器将每隔 1s 就输出"hi,world. "。保存源码文件为 test.ino。

（3）单击工具栏上的上传按钮进行编译上传，然后就可以看到串口监视器上输出字符串了，如图 2-37 所示。

这里我们使用的是 print 函数，因此输出内容没有换行，如果用的是 println 函数，则会每输出一个字符串就换行，读者可以自行尝试。至此，我们成功验证了串口的工作。

图 2-37

上面的例子比较简单，现在尝试一个稍微复杂一些的例子——打印输出 ASCII 表中的可视字符。相信学过 C 语言的都知道 ASCII 表，ASCII 表就是美国信息互换标准代码表，是一套基于拉丁字母的字符编码，共收录了 128 个字符，用一个字节就可以存储。我们针对 ASCII 表中的字符，打印出所有可能格式的字节值，包括原始二进制值，ASCII 编码的十进制值、十六进制值、八进制值和二进制值。

【例 2.3】打印输出 ASCII 表中的可视字符

（1）连接好开发板，打开 Arduino IDE，此时会自动建好了一个.ino 源码文件，并且里面有 setup 和 loop 两个空函数。选择开发板和串口号，在 setup 函数中添加如下代码：

```
void setup() {
    //初始化串口并等待端口打开
    Serial.begin(9600);   //设置波特率为 9600
    while (!Serial) {
        ;  //等待串口连接。仅本机 USB 接口需要
    }
    //打印标题并换行
    Serial.println("ASCII Table ~ Character Map");  //输出一行字符串
}
```

然后定义一个全局变量，代码如下：

```
int thisByte = 33;  //第一个可见的 ASCII 字符“！”的数值是 33
//也可以用单引号编写 ASCII 字符。例如，“！”与 33 相同，因此也可以这样定义：
//int thisByte = '!';
```

接着，在 loop 函数中添加代码：

```
void loop() {
    //打印未更改的值，即字节的原始二进制版本
    //串口监视器将所有字节解释为 ASCII，所以第一个数字 33 将显示为“！”
    Serial.write(thisByte);

    Serial.print(", dec: ");
    //将值打印为 ASCII 编码的十进制（以 10 为基数）的字符串
    //十进制是 Serial.print 和 Serial.println 的默认格式，因此不需要修改
    Serial.print(thisByte);
    //如果你愿意，也可以为 decimal 声明修饰符。如果取消对其的注释，这也适用
    //Serial.print(thisByte, DEC);

    Serial.print(", hex: ");
    //将值打印为十六进制字符串（以 16 为基数）
    Serial.print(thisByte, HEX);
```

```
        Serial.print(", oct: ");
        //将值打印为八进制字符串（基数为 8）
        Serial.print(thisByte, OCT);

        Serial.print(", bin: ");
        //将值打印为二进制字符串（基数为 2），而且还打印结束换行符，因为用了 println
        Serial.println(thisByte, BIN);

        //如果打印的是最后一个可见字符“~”或 126，则停止
        if (thisByte == 126)  //也可以用：if (thisByte == '~')
        {
            while (true) {    //这个循环反复执行
                continue;
            }
        }
        //转到下一个字符
        thisByte++;
    }
```

我们在 loop 中打印了每个可见字符的十进制值、八进制值和二进制值。注意，loop 函数会反复被调用，因此 thisByte++;执行后，再执行 loop 时，将打印下一个字符。

保存文件为 test.ino，然后打开串口监视器。

（2）单击工具栏上的上传按钮进行编译上传，然后就可以看到串口监视器输出字符串了，如下所示：

```
ASCII Table ~ Character Map
!, dec: 33, hex: 21, oct: 41, bin: 100001
", dec: 34, hex: 22, oct: 42, bin: 100010
#, dec: 35, hex: 23, oct: 43, bin: 100011
$, dec: 36, hex: 24, oct: 44, bin: 100100
%, dec: 37, hex: 25, oct: 45, bin: 100101
&, dec: 38, hex: 26, oct: 46, bin: 100110
', dec: 39, hex: 27, oct: 47, bin: 100111
(, dec: 40, hex: 28, oct: 50, bin: 101000
), dec: 41, hex: 29, oct: 51, bin: 101001
*, dec: 42, hex: 2A, oct: 52, bin: 101010
+, dec: 43, hex: 2B, oct: 53, bin: 101011
…
n, dec: 110, hex: 6E, oct: 156, bin: 1101110
o, dec: 111, hex: 6F, oct: 157, bin: 1101111
p, dec: 112, hex: 70, oct: 160, bin: 1110000
q, dec: 113, hex: 71, oct: 161, bin: 1110001
r, dec: 114, hex: 72, oct: 162, bin: 1110010
s, dec: 115, hex: 73, oct: 163, bin: 1110011
t, dec: 116, hex: 74, oct: 164, bin: 1110100
u, dec: 117, hex: 75, oct: 165, bin: 1110101
v, dec: 118, hex: 76, oct: 166, bin: 1110110
w, dec: 119, hex: 77, oct: 167, bin: 1110111
```

```
x, dec: 120, hex: 78, oct: 170, bin: 1111000
y, dec: 121, hex: 79, oct: 171, bin: 1111001
z, dec: 122, hex: 7A, oct: 172, bin: 1111010
{, dec: 123, hex: 7B, oct: 173, bin: 1111011
|, dec: 124, hex: 7C, oct: 174, bin: 1111100
}, dec: 125, hex: 7D, oct: 175, bin: 1111101
~, dec: 126, hex: 7E, oct: 176, bin: 1111110
```

可见，我们把可视字符都打印在串口监视器上了。这两个实例输出的内容都在程序中固定了，并不灵活。一线开发过程中，经常要和用户进行交互，需要根据用户的要求输出相应的内容。下面我们设计串口输出程序，根据用户的输入来决定输出内容。比如，用户输入“h”，则串口监视器输出字符串“Hello World!”；用户输入“b”，则输出“boy”；用户输入“g”，则输出“girl”。

【例 2.4】根据用户的输入来决定输出内容

（1）打开 Arduino IDE，在 IDE 中打开 test.ino，并在 test.ino 中输入如下代码：

```
void setup()  //初始化函数，每次上电或复位后只执行一次
{
    Serial.begin(9600);  //设置串口波特率为 9600，这里要跟 IDE 软件设置保持一致
}
void loop() //循环反复执行
{
    int val=Serial.read(); //读取 PC 发送给 Arduino 开发板的字符，并将读到的字符赋给 val
    if(val=='h')//判断接收到的是不是字符 h
        Serial.println("Hello World!");//如果是字符 h，则在串口监视器上显示字符串
“Hello World! ”
    else if(val=='b') Serial.println("boy"); //如果是字符 b，则在串口监视器上显示字
符串“boy”
    else if(val=='g') Serial.println("girl"); //如果是字符 g，则在串口监视器上显示
字符串“girl”
}
```

（2）确保开发板和计算机已经通过数据线相连，再在 IDE 中选择要连接的串口（不同计算机的串口号可能不同，这里是 COM3）。然后就可以编译并上传程序了。在工具栏上单击验证按钮和上传按钮，如果没报错，就可以在 IDE 的右上角处单击串口监视器按钮来打开串口监视器。

在串口监视器的消息编辑框中输入“h”后按回车键，则会显示“Hello World!”，再输入“b”，则显示“boy”，再输入“g”，则显示“girl”，如图 2-38 所示。这就是和用户的一个交互过程，根据用户的指令来输出相应的内容。

图 2-38

2.7　常见的第三方软件

Arduino IDE 虽然功能强大，但一个好汉三个帮，有了更多的第三方软件的辅助，使得 Arduino 开发如虎添翼。本节将介绍几款常见的第三方软件。

2.7.1　Arduino 的模拟仿真利器 Virtual Breadboard

Virtual Breadboard 是一款 Arduino 开发版仿真软件，其功能非常强大，可以说是专门为了 Arduino 而定制的，当然它也支持很多其他芯片，还支持用户自定义的元件。它的界面如图 2-39 所示。

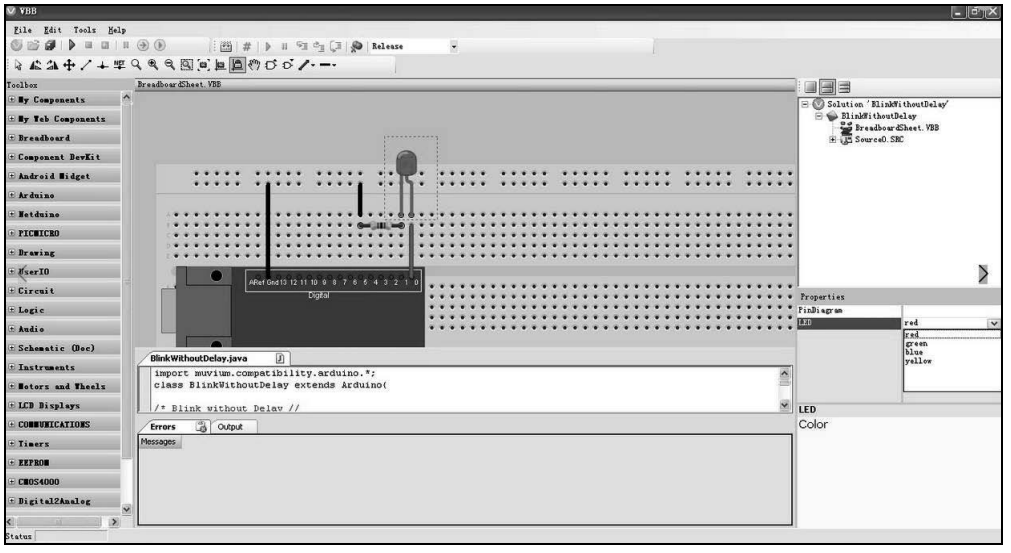

图 2-39

Virtual Breadboard（以下简称 VBB）中文名可直译为“虚拟面包板”，它通过单片机实现嵌入式软件的模拟器和开发环境。VBB 非常简单易用，我们可以轻松地用它取代日常使用的面包板。更加令人兴奋的是，它不但可以像著名的 Fritzing 那样包括所有 Arduino 的样例电路，可以实现面包板电路的设计和布置，还包括所有样例程序，并且可以实现对程序的仿真调试。当然，VBB 的强大不仅如此，它还支持 PIC 系列芯片、Netduino，以及 Java、VB、C++等主流编程环境。

VBB 可以模拟 Arduino 和各种各样的电子模块，例如液晶屏、舵机、逻辑数字电路以及其他的输入/输出设备。这些部件不仅可以直接使用，还可以通过组合的方式设计出更复杂的电路和模块。也就是说，即使零件库里没有我们想要的零件，也可以轻松地从网上的分享区下载或者自己设计制作一个全新的零件。VBB 具有如下特点：

- 先做原型模拟，然后快速实现。
- 界面友好，具有可视化的模拟和交互效果，可以实时看到 LED 的闪烁和电机的转动。
- 100%安全的电子实验，不必担心触电或者冒烟。
- 可分享自己的作品，或下载他人分享的模块。
- 通过样例来快速学习。

VBB 目前更多专注于教育领域。这个曾经免费的软件现在已经收费了，目前单用户要 49 美元，可以无限制使用并且免费升级 1 年。如果不想花钱，可以在官方网站（https://www.virtualbreadboard.com）下载免费版本的 VBB Express。

2.7.2 电路分析与实物仿真软件 Proteus

由于 Arduino 的使用者一般是那些对电路知识、电子技术及单片机技术等了解不深入的初学者，因此，如何在 Arduino 开发过程中快速有效地提高他们的片机系统开发能力和电子电路设计能力，这是一个迫切需要解决的问题。

Proteus 的引入较好地解决了这个问题。Proteus 是一款电路分析与实物仿真软件，它除了能进行基本电子电路仿真外，还能直接在单片机虚拟系统上对微控制器单元（Microcontroller Unit，MCU）编程。该软件的主界面如图 2-40 所示。

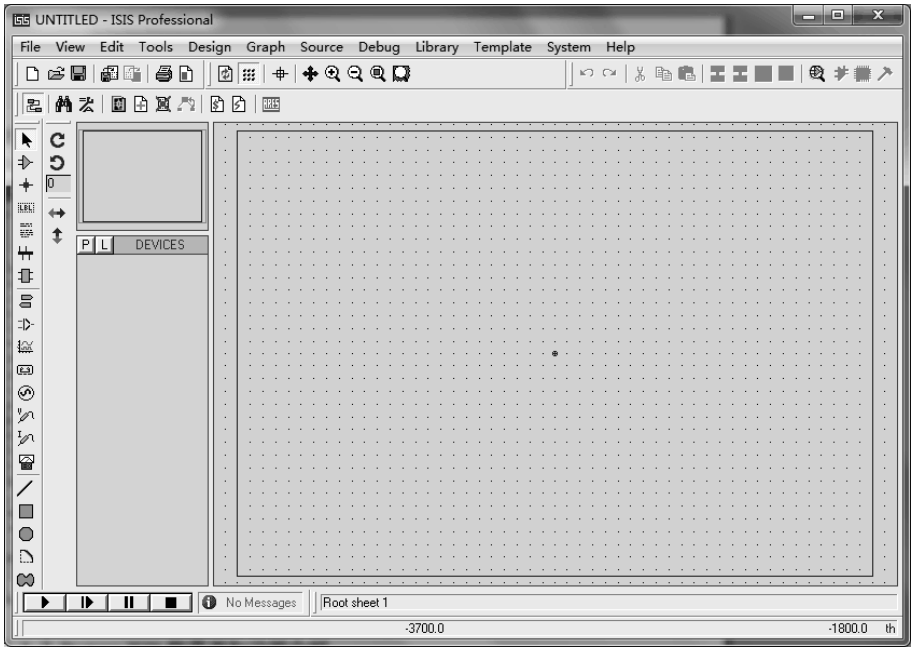

图 2-40

Proteus 虚拟开发技术的应用形成了一种全新的 Arduino 系统开发理念，其系统开发流程如下：

（1）电路设计与仿真。
（2）电路修改与完善。
（3）绘制印刷电路板（Printed Circuit Board，PCB）与生成印刷电路板三维效果图。
（4）硬件组装与调试。

这个流程颠覆了传统的系统设计方法，使得 Arduino 的使用者能够在设计的早期阶段就发现潜在的系统设计缺陷，避免了在设计过程中不断对焊接电路进行修改所带来的问题。此外，印刷电路板的三维效果图将元件的符号与其实际的封装形式直观地结合起来，为初学者提供了强烈的视觉和感知体验，从而深化了他们对于单片机系统设计的理解和感知。该软件的官网为 https://www.labcenter.com/。