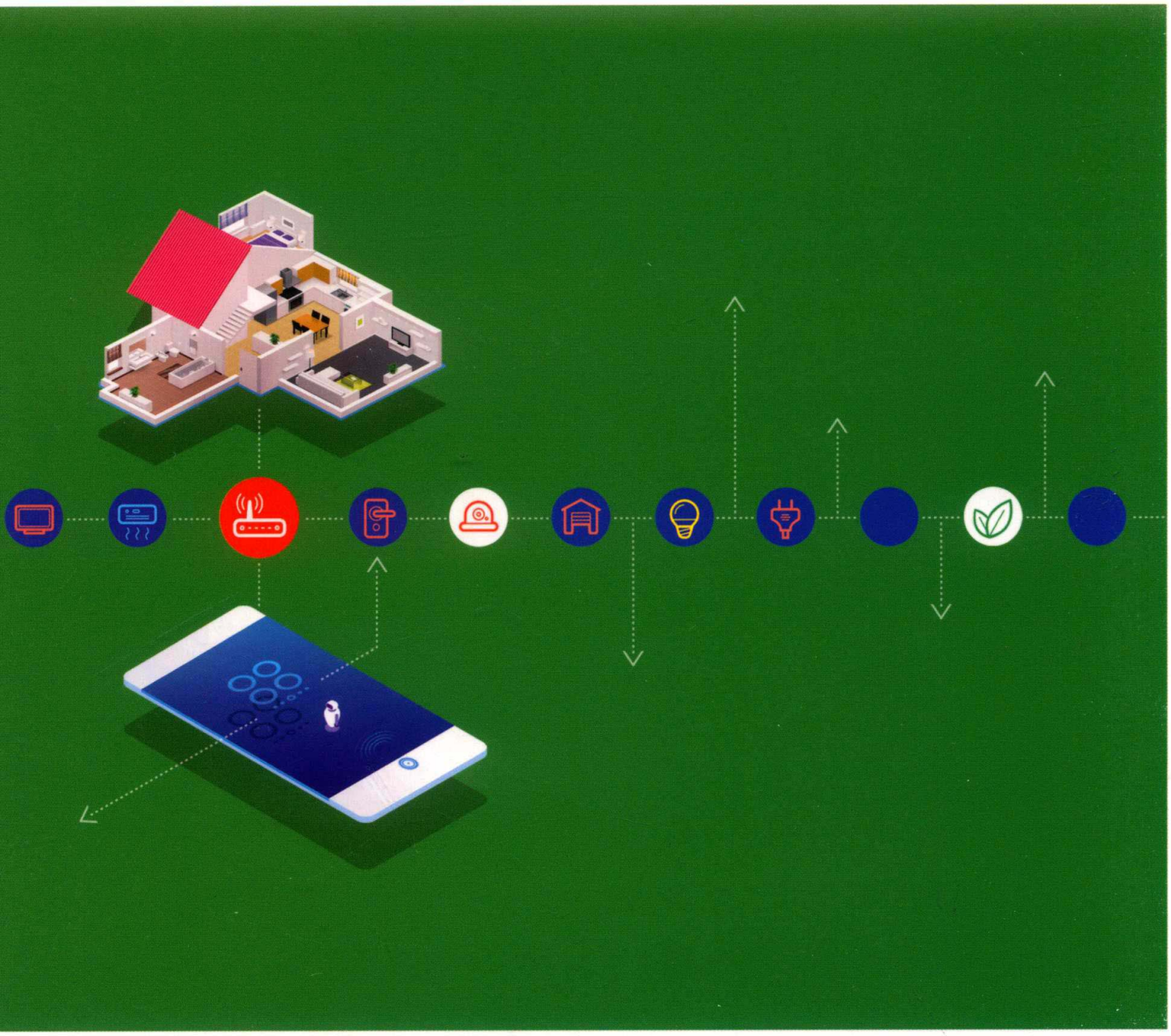
智能家居控制系统的设计与开发

——TI CC3200+物联网云平台+微信

王立华　高世皓　张　恒　周松江　编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn



ISBN 978-7-121-34676-7

责任编辑：富　军
封面设计：孙焱津

定价：69.80 元

智能家居控制系统的设计与开发

——TI CC3200 +物联网云平台+微信

王立华　高世皓　张　恒　周松江　编著

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

随着传感器技术、控制技术、大数据、移动互联网等基础应用的迅速发展，智能家居处于蓬勃发展阶段，近几年出现的云服务、微信等技术又加快了这种发展趋势。本书基于 TI CC3200 LaunchPad、物联网云平台和微信，详细地介绍了整个智能家居控制系统的开发过程，并给出了完整的应用开发实例，力图反映目前智能家居发展的一些新情况。

本书的特点是紧跟科技发展前沿，内容新颖翔实、图文并茂、条理清晰，具有较强的实用性和可操作性，既可作为高等院校物联网、计算机、电子、自动化、无线通信等相关专业的教材，也可作为从事嵌入式、物联网、智能家居等相关技术人员的参考用书。

图书在版编目（CIP）数据

智能家居控制系统的设计与开发：TI CC3200+物联网云平台+微信/王立华等编著．—北京：电子工业出版社，2018.8
ISBN 978-7-121-34676-7

Ⅰ．①智… Ⅱ．①王… Ⅲ．①住宅-智能控制-系统设计 ②住宅-智能控制-系统开发 Ⅳ．①TP273

中国版本图书馆 CIP 数据核字（2018）第 149353 号

责任编辑：富 军
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：787×1092 1/16 印张：15 字数：384 千字
版 次：2018 年 8 月第 1 版
印 次：2018 年 8 月第 1 次印刷
印 数：2 000 册 定价：69.80 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）88254888，88258888。

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

本书咨询联系方式：（010）88254456；E-mail：fujun@phei.com.cn。

前言

2016年，国务院印发的《“十三五”国家战略性新兴产业发展规划》明确提出，积极推进云计算和物联网的发展，加强行业云服务平台的建设，支持行业信息系统向云平台迁移，实施网络强国战略，加快建设“数字中国”，推动物联网、云计算和人工智能等技术向各行业全面融合渗透，构建万物互联、融合创新、智能协同、安全可控的新一代信息技术产业体系。

目前，物联网在智能家居、智能交通、智慧校园、智慧森林、智能物流、智能交通、智能电网、智能环境监测领域的应用正逐步广泛深入，同时随着一些新技术、新产品的研发和应用，未来的物联网将会朝着智能、便捷、高效、集成、标准等方面发展，对我们的日常生活、工业生产和社会发展将产生巨大的影响。据统计，我国物联网产业规模已从2009年的1 700亿元跃升至2016年的9 000亿元，已成为全球最大的市场；2017年，我国物联网产业规模已超过万亿元，同比增速连续多年超过20%。

智能家居作为物联网技术的重要应用，融合了无线传感器技术、信息通信技术、计算机网络技术等高新技术手段，能够实现智能家居用户对居家生活环境的个性化需求，可通过智能化控制和管理技术，真正实现“以人为本”的居家生活。随着物联网、大数据、云计算、人工智能等技术的迅速发展，海量智能家居的终端快速联网成为可能，也为智能家居的快速发展奠定了坚实的基础。在国内，从2014年开始，智能家居规模出现了明显的增长，至2018年，随着主要智能家居系统平台及大数据服务平台的搭建完毕，下游设备厂商完善，智能家居产品被消费级市场接受，市场规模将达到1 800亿元。这意味着，智能家居产品将会在两三年内迎来第一个爆发期，京东微联业务部总监邓正平用数据说明了这一点：“每小时有6万人在京东查看智能产品，有8万多件智能产品被用户肯定，智能家居一定是未来的趋势”。

本书针对目前正蓬勃发展的智能家居系统，从概念和技术上进行了比较全面的介绍，通过将移动互联网、无线局域网、云服务和微信等技术的结合，以及智能家电与嵌入式设备、移动设备等的结合，构建了基于WiFi、云服务、微信等为一体的智能家居控制系统。其中，硬件平台采用内嵌WiFi模块的TI CC3200 LaunchPad，软件平台采用阿里云+微信。

本书共 10 章。第 1 章介绍物联网的概念、发展及技术架构。第 2 章介绍智能家居发展历程及智能家居使用的联网标准、实现的功能、技术架构。第 3 章对 CC3200 LaunchPad 硬件平台进行介绍，重点讲述 CC3200 硬件结构的三个组成部分及 CC3200 LaunchPad 的板载资源。第 4 章介绍 CC3200 软件开发环境的搭建过程。第 5 章介绍 CC3200 的开发与应用，包括工程导入、工程创建、编译库重建、工程开发流程及常用传感器和驱动设备的应用程序，帮助读者了解和掌握 CC3200 的开发过程。第 6 章介绍微信公众平台及其在智能家居中的应用情况。第 7 章介绍云平台在物联网和智能家居中的应用、几种常用的云服务平台及云服务应用开发协助工具 git。第 8 章介绍 CC3200 微控制器连接到云服务器的程序开发过程及 CC3200 与云服务器之间的数据交换。第 9 章介绍微信服务器与云服务器之间的交互过程，包括微信公众平台如何接入云服务器、交互基本原理和消息格式、云服务器微信请求接口设计。第 10 章介绍智能家居的设计实例：将微信公众号和 CC3200 结合，利用云服务器存储数据，设计一款基于云服务的智能家居控制系统，用户使用微信公众号即可远程控制家居设备并获取家居环境的状态。

笔者希望通过本书使读者能够了解当前的 WiFi、云服务、微信等新兴技术在智能家居系统中的应用，从而对设计智能家居控制系统有所帮助。本书在编写过程中参阅了国内外大量的物联网和智能家居方面的论文和专著，内容中的智能家居实例来源于笔者的一些项目成果，还得到了山东省自然基金面上项目（项目编号：ZR2018MF005）的资助，同时还非常感谢德州仪器（上海）有限公司的友情支持。

智能家居是一项高新热点技术，涉及的技术非常前沿、广泛，处于快速发展中，限于笔者自身的学识和水平，书中难免会出现疏漏和不足之处，恳请读者批评指正。

编著者

目录

第 1 章　开启物联网的大门 …… 1

1.1　物联网概述 …… 1

1.1.1　物联网的定义 …… 1

1.1.2　物联网的发展状况 …… 2

1.2　物联网的技术架构 …… 3

1.2.1　感知层 …… 4

1.2.2　网络层 …… 7

1.2.3　应用层 …… 9

第 2 章　走进智能家居 …… 13

2.1　智能家居的发展状况 …… 15

2.1.1　国外发展现状 …… 15

2.1.2　国内发展现状 …… 16

2.2　智能家居组网技术基础 …… 17

2.2.1　组网方式分类 …… 17

2.2.2　主流技术分析 …… 18

2.2.3　HTTP 协议 …… 33

2.3　智能家居实现的功能 …… 36

2.4　智能家居技术架构 …… 37

第 3 章　CC3200 硬件平台 …… 41

3.1　CC3200 微控制器 …… 41

3.1.1　应用 MCU 子系统 …… 41

3.1.2　WiFi 网络处理器子系统（CC3100） …… 48

3.1.3　电源管理子系统 …… 49

3.2　CC3200 LaunchPad …… 51

3.2.1　硬件电路 …… 51

3.2.2 跳线设置 …… 53
3.2.3 按键和 LED 灯 …… 55
第 4 章 CC3200 软件开发环境的搭建 …… 57
4.1 CCS 集成开发环境 …… 57
4.1.1 获取 CCS V6 软件 …… 57
4.1.2 CCS V6 安装过程详解 …… 58
4.1.3 CCS V6 软件配置 …… 61
4.2 辅助软件工具 …… 62
4.2.1 CC3200 软件开发工具包 …… 62
4.2.2 引脚配置代码生成器 PinMux …… 63
4.2.3 Flash 烧写工具 UniFlash …… 66
4.2.4 CC3200 LaunchPad 驱动安装 …… 67
4.2.5 串口终端 Tera Term …… 70
第 5 章 CC3200 的开发与应用 …… 73
5.1 硬件运行测试 …… 73
5.1.1 导入工程 …… 73
5.1.2 编译与下载调试 …… 77
5.1.3 Uniflash 程序的烧写 …… 78
5.2 项目的开发过程 …… 81
5.2.1 CCS 编程库的重建 …… 81
5.2.2 新建工程 …… 83
5.2.3 硬件驱动程序的编写 …… 85
5.2.4 应用程序的编写 …… 88
5.3 基于 CC3200 的传感器应用 …… 91
5.3.1 板载温度传感器 …… 91
5.3.2 板载加速度传感器 …… 96
5.3.3 光强度传感器 …… 100
5.3.4 湿度传感器 …… 106
5.3.5 气体传感器 …… 110
5.3.6 测距传感器 …… 112
5.3.7 红外热释电传感器 …… 115
5.4 基于 CC3200 驱动设备的应用 …… 116
5.4.1 继电器的应用 …… 116
5.4.2 电动机驱动的应用 …… 117
第 6 章 智能家居与微信公众平台的结合 …… 121

6.1 微信公众平台 …… 121
6.1.1 注册微信公众账号 …… 121
6.1.2 开启公众平台测试账号 …… 122
6.1.3 自定义菜单介绍 …… 123
6.2 智能家居与微信公众平台结合 …… 125
6.2.1 微信与智能家居结合的原因 …… 125
6.2.2 微信在智能家居中的应用 …… 126
6.2.3 未来微信在智能家居中的发展 …… 127
第7章 云服务平台 …… 131
7.1 云服务的发展现状 …… 132
7.2 云服务在物联网中的应用 …… 133
7.2.1 云服务与物联网的结合 …… 133
7.2.2 云服务所提供的服务分类 …… 135
7.2.3 云服务在物联网应用中面临的问题 …… 136
7.3 基于云服务的智能家居 …… 137
7.3.1 基于云服务的智能家居的系统组成 …… 138
7.3.2 基于云服务的智能家居的特点 …… 140
7.4 常用的云服务平台 …… 141
7.5 云服务应用开发协助工具 git …… 145
7.5.1 分布式版本控制系统 git …… 146
7.5.2 推送方式一：代码托管平台作为中转站 …… 159
7.5.3 推送方式二：在云服务平台上搭建 git 服务器 …… 164
第8章 CC3200 微控制器连接到云服务器 …… 167
8.1 CC3200 微控制器的程序开发 …… 167
8.1.1 GPIO 配置函数 …… 168
8.1.2 CC3200 创建多任务 …… 171
8.1.3 传感器程序的移植 …… 172
8.2 CC3200 与云服务器之间的数据交换 …… 173
8.2.1 CC3200 连接到路由器 …… 173
8.2.2 CC3200 与云服务器之间的数据交换 …… 174
第9章 微信服务器与云服务器之间的交互 …… 177
9.1 微信公众平台接入云服务器 …… 177
9.1.1 开启开发者模式 …… 177
9.1.2 填写服务器配置 …… 178
9.1.3 验证服务器地址的有效性 …… 179

9.1.4 在云服务器上实现业务逻辑 …… 179
9.2 交互基本原理及消息格式 …… 179
9.2.1 交互基本原理 …… 179
9.2.2 微信客户端推送消息 …… 180
9.2.3 云服务器响应消息 …… 182
9.3 云服务器上的微信请求接口设计 …… 184
第 10 章 应用案例：基于 CC3200、微信及云服务的远程智能云家居系统 …… 187
10.1 系统设计方案 …… 187
10.2 系统硬件设计 …… 188
10.2.1 温/湿度传感器模块 …… 189
10.2.2 DS1302 实时时钟模块 …… 189
10.2.3 继电器模块 …… 190
10.2.4 电动机驱动模块 …… 191
10.2.5 烟雾传感器模块 …… 192
10.2.6 12864 液晶显示模块 …… 192
10.3 远程智能云家居系统软件设计 …… 193
10.3.1 CC3200 微控制器程序设计 …… 193
10.3.2 阿里云服务器程序设计 …… 211
10.3.3 微信公众账号程序设计 …… 218
10.4 系统测试 …… 223
10.4.1 测试前的准备 …… 223
10.4.2 CC3200 及其外围模块功能的测试 …… 223
10.4.3 阿里云服务器功能的测试 …… 224
10.4.4 微信公众号功能的测试 …… 226

第1章 开启物联网的大门

清晨，当滴滴的闹钟声将你唤醒，睁开双眼，卧室的窗帘徐徐拉开，柔和的阳光透过白色的细纱照到身上，卧室的音响开始传出悠扬轻盈的音乐，踏着美妙的音符走到洗手间门前，顶光灯自动打开，镜子旁的储物柜门向两侧伸展，里面洗漱用品一应俱全，伸手拿出牙刷放到旁边的牙膏机下面，牙膏被自动挤到牙刷上，消毒过的毛巾也从专用消毒盒里自动伸出等待使用。这时，在厨房里，咖啡机和面包机已经开始工作，一会儿你就可以美美地享受一份健康营养的早餐。洗漱完毕，吃完早饭后，出门上班，室内电器会自动进入休眠模式，环境监测和安全防护系统正常工作，当发现室内存在可燃气体或有不明人员进入时会自动向你的手机发出警报。晚上下班后，在回家的路上通过手机上的一条指令，家里的厨房会自动将饭菜做好，空调开始工作，调节室内温度，电冰箱会自动检测储备食物是否充足并向你发出提示。到家后，人脸识别门禁系统自动将门打开，开门后，室内照明灯自动打开，电视机自动调节到你喜欢的频道……

这不是在描述一部科幻电影，而是物联网技术将要给我们日常生活带来的巨大变革！何为物联网？它从何处而来？经历了怎样的发展？拥有哪些技术？下面将详细介绍物联网！

1.1 物联网概述

1.1.1 物联网的定义

物联网是新一代信息技术的重要组成部分，也是“信息化”时代的重要发展阶段。其英文名称为 Internet of Things（IoT）。物联网在不同时间点上的定义不同。

1999 年，麻省理工学院 Auto-ID 研究中心对物联网的定义为：物联网就是把所有的物品都通过射频识别和条码等信息传感设备与互联网连接起来，实现智能化识别和管理。

2005 年，国际电信联盟对物联网的定义为：物联网主要解决物品到物品、人到物品和

人到人之间的互联，与传统互联网不同的是，人到物品是指利用通用装置和物品之间的连接，人到人是指人与人之间不依赖于个人电脑而进行的互联。

2008 年 5 月 27 日，欧洲智能系统集成平台对物联网的定义为：物联网是由具有标识、虚拟个性的物体/对象所组成的网络，这些标识和个性等信息在智能空间使用智慧接口与用户、社会和环境等进行通信。

2009 年 9 月，欧盟第 7 框架 RFID 和物联网研究项目组在发布的研究报告中提出：物联网是未来互联网的一个组成部分，可以被定义为基于标准的和可操作的通信协议，是具有自配置能力的、动态的全球网络基础架构，物联网中的“物”都是具有标识、物理属性和实质上的个性，使用智能接口实现信息网络的无缝整合。

我国对物联网的定义首次出现在 2010 年的政府工作报告中，定义为：物联网是通过传感设备按照约定的协议，把各种网络连接起来，进行信息交换和通信，以实现智能化识别、定位、跟踪、监控和管理的一种网络。

其实，物联网目前仍没有一个精确且公认的定义，但通过对前面几个官方定义的分析可以得出：物联网的核心和基础仍然是互联网，是在互联网的基础上延伸和扩展的网络；用户端延伸和扩展到任何物品与物品之间进行信息交换和通信，也就是物物相联。

物联网通过智能感知、识别技术与普适计算等通信感知技术广泛应用于网络的融合中，也因此被称为继计算机、互联网之后世界信息产业发展的第三次浪潮。物联网是互联网的应用拓展，与其说物联网是网络，不如说物联网是业务和应用。物联网具有普通对象设备化、自治终端互联化和普适服务智能化三个重要特征。

1.1.2 物联网的发展状况

物联网的构想最早出现在比尔·盖茨 1995 年出版的《未来之路》一书中。在书中，尽管比尔·盖茨没有明确提出物联网的概念，但是却描绘了未来一种物联网式的生活构想。在当时的计算机水平和网络水平下，这种构想已经远远超越了那个年代，并引领社会朝着一个新的目标发展。

真正意义上的物联网概念是在 1999 年由美国麻省理工学院的 Auto-ID 研究中心首先提出的，当时的物联网主要建立在物品编码、RFID 技术和互联网的基础上，以 Auto-ID 中心研究的产品电子代码 EPC 为核心，利用射频识别、无线数据通信等技术，并基于计算机互联网构造的实物互联网。简单地说，物联网就是将各种信息传感设备与互联网结合形成一个巨大的网络，让相关的物品都与网络连接在一起，以实现物品的自动识别和信息的互联共享。由于 1999 年技术条件和社会条件的限制，因此提出的物联网概念并没有掀起热潮。直到 2005 年 11 月 17 日在突尼斯举行的信息社会世界峰会上，国际电信联盟发布了《ITU 因特网报告 2005：物联网》报告。该报告指出，无所不在的“物联网”通信时代即将来临，世界上的所有物体，从轮胎到牙刷、从屋顶到纸巾，都可以通过因特网主动进行信息交换，

射频识别技术、传感器技术、纳米技术、智能嵌入式技术将得到更加广泛的应用。

2009年，奥巴马就任美国总统后，于1月28日与美国工商业领袖举行了一次“圆桌会议”。IBM首席执行官彭明盛首次提出“智慧的地球”概念，建议新政府投资新一代的智慧型基础设施。随后，美国总统奥巴马确定物联网作为美国今后发展的国家战略方向之一。物联网一词立刻变得炙手可热起来，世界各国都把目光投向了物联网。

同样，在2009年，我国也开始了对物联网的研究与发展。2009年8月7日，国务院总理温家宝在无锡视察时发表重要讲话，首次提出了“感知中国”的战略构想。同年11月3日，温家宝总理向首都科技界发表了题为《让科技引领中国可持续发展》的讲话，再次强调科学选择新兴战略性产业的重要性，并指示要着力突破传感网、物联网的关键技术。在中央政策的引导下，中关村物联网产业联盟成立，无锡市与北京邮电大学就物联网技术研究和产业发展签署合作协议，全国高校首家物联网研究院在南京邮电大学正式成立，沪深股市甚至一夜之间打造出全新的“物联网板块”！我国物联网产业规模已从2009年的1700亿元跃升至2016年的9000多亿元，成为全球最大的市场，2017年甚至超过万亿元，同比增速连续多年超过20%。

目前，全球物联网的应用仍处于发展初期，在公共市场的应用也开始显现，M2M（机器与机器通信）、车联网、智能电网是近几年全球发展较快的重点应用领域。随着一些新技术、新产品的研发和应用，未来物联网将会朝着智能、便捷、高效、集成、标准等方面发展。

当然，我们需要清醒地认识到，物联网的发展不是一蹴而就的，物联网的应用存在着安全隐私、数据保护、资源控制、标准制定、信息共享、服务开发等方面的挑战，随着技术的提高和新技术的涌现，这些问题将会逐一解决。未来的物联网将会更加强大，应用更加广泛！

1.2 物联网的技术架构

从技术架构上来看，物联网可以分为三层：感知层、网络层和应用层，如图1.1所示。

感知层是由各种传感器和传感器网关组成的，包括温/湿度传感器、烟雾传感器、RFID标签、二维码标签、读写器、摄像头、GPS等。其作用是识别物体，采集信息，与人体结构中的皮肤和五官的作用相似。

网络层包括通信与互联网的融合网络、网络管理中心和信息处理中心、云服务平台等，相当于人的神经中枢和大脑，负责传递和处理感知层获取的信息。

应用层则是物联网与行业专业技术的深度融合，与行业需求结合，实现行业智能化，类似于人的社会分工，最终构成人类社会。

在各层之间，信息不是单向传递的，而是相互交互、控制的，所传递的信息是多种多样

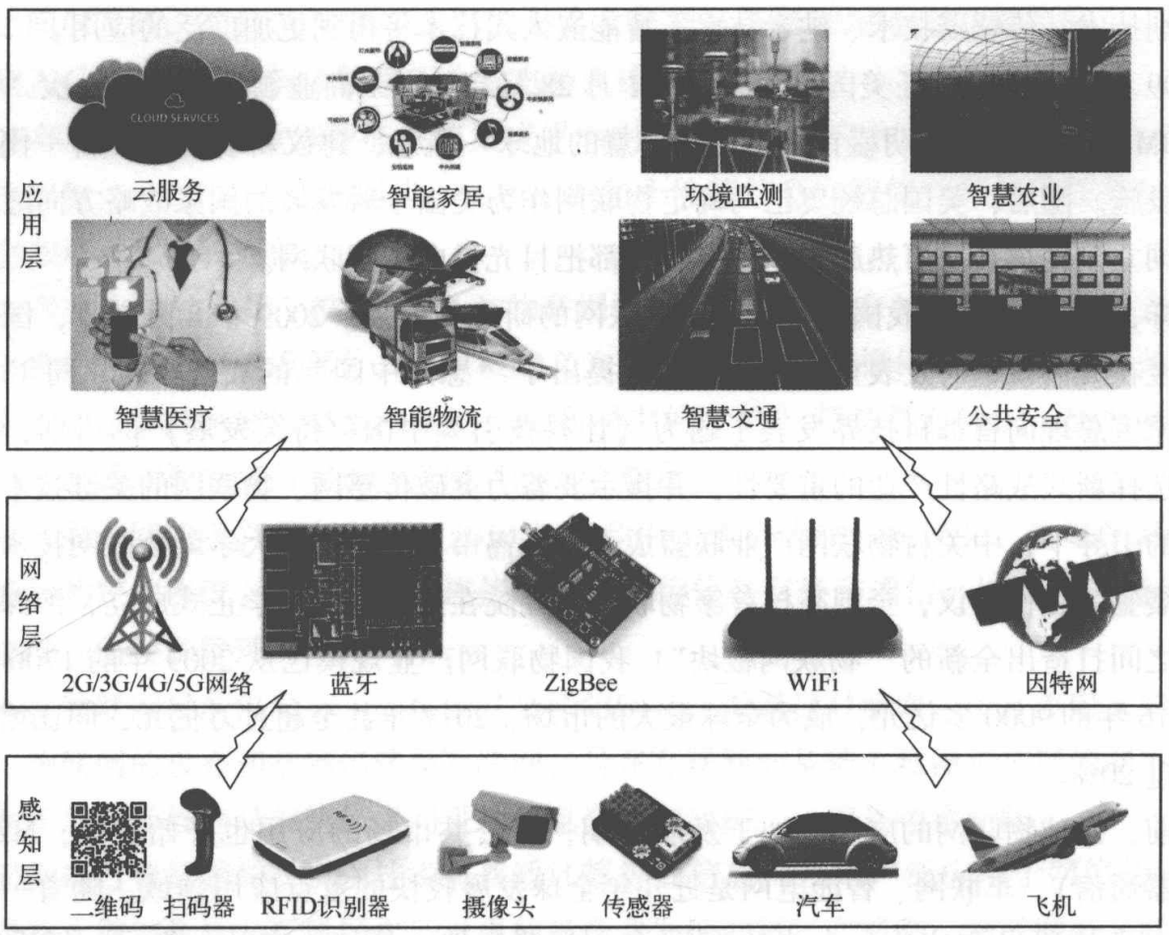


图 1.1　物联网的技术架构

的。其中最关键的是物品的信息，包括在特定应用系统范围内能唯一标识物品的识别码及物品的静态信息和动态信息。

1.2.1　感知层

1. 感知层的功能

感知层常被称为物联网的皮肤和五官，是物联网三层架构中的最底层。感知层是物联网发展和应用的基础，具有物联网全面感知的核心能力，其作用是不言而喻的。

在通常情况下，感知层的作用是采集信息并传送至上位机，即包括数据采集和数据传输两部分。首先通过温/湿度传感器、烟雾传感器、RFID 标签、二维码标签、读写器、摄像头、GPS 等采集外部数据，然后通过工业现场总线、蓝牙、ZigBee、WiFi 等短距离有线或无线传输技术将采集到的数据发送至网关设备，从而完成物联网数据的准备工作。也可以只有数据的短距离传输这一部分，特别是在仅传递物品识别码的情况下。

实际上，感知层的这两个部分有时是很难明确区分开的。

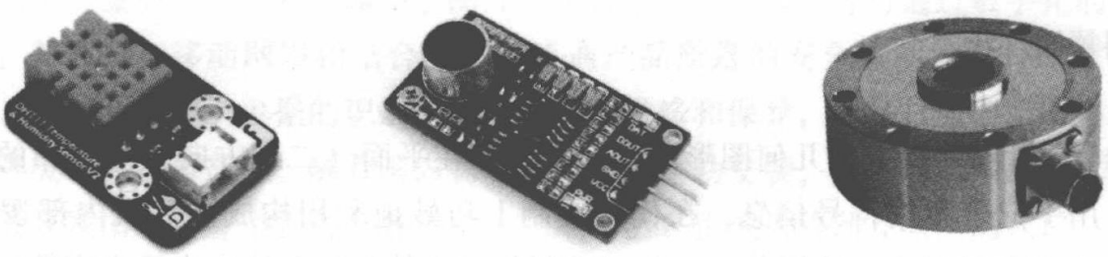
2. 感知层的关键技术

感知层处于物联网体系结构的最底层，其所涉及的关键技术主要包括传感器技术、

RFID 技术和二维码技术等。

（1）传感器技术

传感器是一种检测装置，能感受到被测量的信息，并能将感受的信息按一定的规律变换为电信号或其他所需形式的信息输出，可以满足信息的传输、处理、存储、显示、记录和控制等要求，具有微型化、数字化、智能化、多功能化、系统化、网络化等特点。图 1.2 为几种常见的传感器。



（a）温/湿度传感器　　（b）声音传感器　　（c）压力传感器

图 1.2　几种常用的传感器

传感器是实现自动检测和自动控制的首要环节，通常根据基本感知功能分为热敏元件、光敏元件、气敏元件、力敏元件、磁敏元件、湿敏元件、声敏元件、放射线敏感元件、色敏元件及味敏元件。

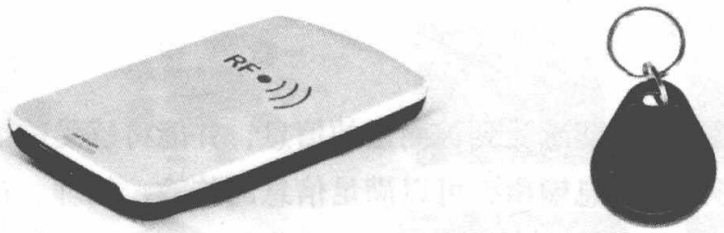
在物联网系统中，对各种参量进行信息采集和简单加工处理的设备被称为物联网传感器。传感器可以独立存在，也可以与其他设备以一体的方式呈现，但无论采用哪种方式，传感器都是物联网中的感知和输入部分。在未来的物联网中，传感器及其组成的传感器网络将在数据采集前端发挥重要的作用。

（2）RFID 技术

RFID 技术，即射频识别技术，是一种通信技术，可以通过无线电信号识别特定的目标并读写相关的数据，不需要依靠机械接触或光学接触。

RFID 系统主要由三部分组成：电子标签、读写器和天线。其中，电子标签芯片具有数据存储区，用于存储待识别物品的标识信息；读写器是将约定格式的待识别物品的标识信息写入电子标签的存储区中（写入功能），或在读写器的阅读范围内，以无接触的方式将电子标签内保存的信息读取出来（读出功能）；天线用于发射和接收射频信号，内置在电子标签和读写器中。RFID 技术的工作原理为：电子标签进入读写器产生的磁场后，读写器发出射频信号，凭借感应电流所获得的能量发送存储在芯片中的产品信息，或者主动发送某一频率的信号；读写器读取信息并解码后，送至中央信息系统进行有关数据的处理。图 1.3 为 RFID 产品。

在物联网系统中，RFID 标签中存储着规范而具有互用性的信息，通过有线或无线的方式把这些信息自动采集到中央信息系统，实现对不同物品（商品）的识别，进而通过开放式的计算机网络实现信息交换和共享，实现对物品的“透明”管理，具有无须接触、自动化程度高、耐用可靠、识别速度快、适应各种工作环境、可实现高速和多标签同时识别等优



（a）RFID读卡器　　（b）RFID门禁卡

图 1.3　RFID 产品

势，应用领域广泛。

（3）二维码技术

二维码是用某种特定的几何图形按一定的规律在平面（二维方向）上分布的黑白相间的图形，用于记录数据符号信息，在代码编制上巧妙地利用构成计算机内部逻辑基础的“0”“1”比特流的概念，使用若干个与二进制相对应的几何形体来表示文字数值信息，通过图像输入设备或光电扫描设备自动识读以实现信息的自动处理。

二维码具有条码技术的一些共性：每种码制有其特定的字符集；每个字符占有一定的宽度；具有一定的校验功能等；同时还具有对不同行的信息进行自动识别的功能；处理图形旋转变化点；等等。图 1.4 显示了一个二维码。

图 1.4　二维码

二维码的主要特点如下：

① 高密度编码，信息容量大：可容纳多达 1850 个大写字母或 2710 个数字或 1108 个字节或 500 多个汉字，比普通条码信息容量高几十倍。

② 编码范围广：可以将图片、声音、文字、签字、指纹等数字化的信息编码用条码表示出来，也可以表示多种语言文字及图像数据等。

③ 容错能力强，具有纠错功能：当二维码因穿孔、污损等引起局部损坏时，照样可以正确识读，损毁面积达 50%仍可恢复信息。

④ 译码可靠性高：比普通条码译码错误率要低得多，误码率不超过千万分之一。

⑤ 可引入加密措施：保密性、防伪性好。

⑥ 成本低，易制作，持久耐用。

⑦ 条码符号形状、尺寸大小的比例可变。

⑧ 可以使用激光或 CCD 阅读器识读。

物联网的应用离不开自动识别，条码、二维码及 RFID 的应用较普遍。二维码相对于一维码具有数据存储量大、保密性好等特点，能够更好地与智能手机等移动终端结合，可形成更好的互动性和用户体验。与 RFID 相比，二维码不仅成本优势凸显，其用户体验和互动性也具有更好的应用前景。

由于在移动互联业务模式下，人们的活动范围更加宽泛，因此更需要适时进行信息的交互和分享。随着 4G 智能手机的普及，二维码的应用不再受时空和硬件设备的局限。二维码技术可以用于对流通产品的基本属性、图片、声音、文字、指纹等可通过数字化的信息进行编码捆绑，并通过与移动网络相结合实现对流通产品质量的安全追溯、适时跟踪、物流仓储、促销、会议及身份、单据的识别，设备的远程维修和保养，产品打假防窜及终端用户激励，企业供应链流程再造等。随着国内物联网产业的蓬勃发展，更多的二维码技术应用解决方案不断被开发出来并应用到各行各业中。二维码已真正成为移动互联网的入口。

1.2.2 网络层

1. 网络层的功能

物联网的网络层如同人的大脑和神经中枢。物联网要求网络层能够把感知层感知到的数据进行无障碍、高可靠性、高安全性的传送，解决的是数据在一定范围内，尤其是远距离传输的问题。物联网的网络层建立在现有的移动通信网络、互联网和其他专用网络的基础上，对网络内各个物联网的终端、服务器、智能手机等接入设备进行互联，从而实现感知数据的传输。

2. 网络层的关键技术

物联网的网络层是建立在现有的移动通信网、互联网和其他专用网的基础上的，可实现物联网的“物物相连”。根据物联网的构成特点，网络层的主要应用技术为无线网络技术，包括蓝牙技术、ZigBee 技术、WiFi 技术、2G/3G/4G/5G 网络等。

（1）蓝牙技术

蓝牙（Bluetooth）是一种无线技术标准，可实现固定设备、移动设备和楼宇个人域网之间的短距离数据交换与通信，波段为 2400～2483.5MHz（包括防护频带）。蓝牙使用跳频技术将传输的数据分割成数据包，通过 79 个指定的蓝牙频道分别传输数据包。每个频道的频宽为 1 MHz。蓝牙 4.0 使用 2 MHz 间距，可容纳 40 个频道。第一个频道始于 2402 MHz，每 1 MHz 一个频道，直至 2480 MHz。蓝牙模块（CC2541）的实物图如图 1.5 所示。

（2）ZigBee 技术

ZigBee 是一种短距离、低功率无线网络技术。其技术方案介于无线标记技术和蓝牙技术之间，主要用于近距离无线连接。ZigBee 技术的基础是 IEEE 802.15.4 协议，也是 IEEE

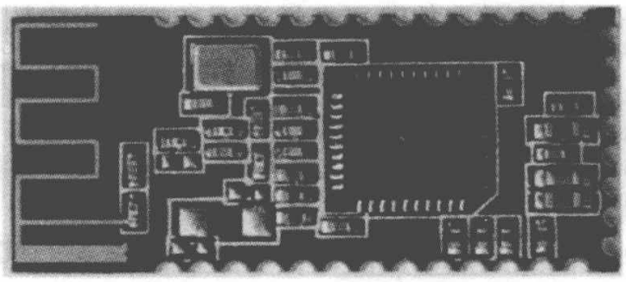


图 1.5　蓝牙模块（CC2541）的实物图

802.15.4 协议的代名词，所以该协议被称作 IEEE 802.15.4（ZigBee）协议。

相对于常见的无线通信标准，ZigBee 协议栈紧凑简单，具体实现要求很低，只要 8 位处理器再配上 4KB ROM 和 64KB RAM 等就可以满足最低需要，从而大大降低了芯片的成本。ZigBee 模块（CC2530）的实物图如图 1.6 所示。

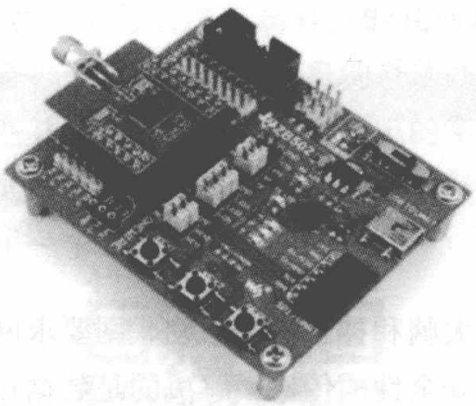


图 1.6　ZigBee 模块（CC2530）的实物图

（3）WiFi 技术

与 ZigBee 一样，WiFi（Wireless Fidelity，无线保真）技术也是属于短距离无线技术的一种，使用 IEEE 802.11 系列协议，能够将个人电脑、移动设备（平板、手机、手表）等终端以无线局域网的形式连接在一起，并实现数据的传输和下载。

目前在日常生活中，WiFi 技术已经得到了广泛的普及，给人们带来了极大的方便，相继出现了如无线网络电视、餐厅的无线网络点餐、无线网络支付等应用。图 1.7 为 WiFi 路由器的实物图。

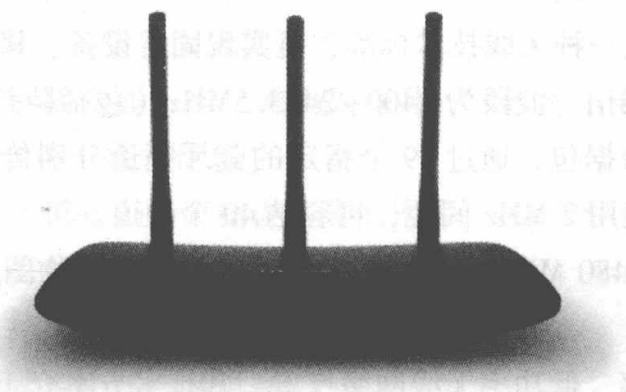


图 1.7　WiFi 路由器的实物图

（4）2G/3G/4G/5G 网络

一般来说，2G 网络是指第二代无线蜂窝电话通信协议，以无线通信数字化为代表，能够进行窄带数据通信。常见的 2G 无线通信协议有 GSM 频分多址（GPRS 和 EDGE）和 CDMA 1X 码分多址，传输速度很慢。

3G 网络是第三代无线蜂窝电话通信协议，主要是在 2G 的基础上发展起来的高带宽数据通信，提高了语音通话的安全性。3G 的数据传输速度都在 500kb/s 以上。目前，3G 常用的标准有 3 种：WCDMA、CDMA2000、TD-SCDMA，传输速度相对较快，可以很好地满足手机上网等需求，但是播放高清视频较为吃力。

4G 网络是指第四代无线蜂窝电话通信协议，集 3G 与 WLAN 于一体，能够传输高质量的视频图像，图像传输质量与高清晰度电视不相上下，能够达到 100Mb/s 的下载速度，比拨号上网快 2000 倍，上传速度也能达到 20Mb/s，可以满足几乎所有用户对于无线服务的要求。

5G 网络是指第五代移动通信网络。其峰值理论的传输速度可达每秒数十 Gb，比 4G 网络的传输速度快数百倍，整部超高画质的电影可在 1s 之内下载完成。随着 5G 网络技术的诞生，用智能终端分享 3D 电影、游戏及超高画质（UHD）节目的时代已向人们走来。

对于用户而言，2G、3G、4G、5G 网络的最大区别在于传输速度不同。

1.2.3 应用层

1. 应用层的功能

物联网的最终目的是把感知和传输过来的信息更好地利用。因此，应用层的主要功能是把感知和传输过来的信息进行存储、分析和处理，从而做出正确的控制和决策，以实现智能化的管理、应用和服务。

具体来说，应用层将通过感知层感知的并通过网络层传输过来的数据通过各类信息系统进行处理，并通过各种设备与人进行交互。应用层也可以按照形态直观地划分为两个子层：一个是应用程序层；另一个是终端设备层。其中，应用程序层进行数据处理，完成跨行业、跨应用、跨系统之间的信息协调、共享、互通的功能，包括电力、医疗、银行、交通、物流、工业、农业、环保、家居生活等，可用于企业、社会、政务、家庭和个人等，是物联网作为深度信息化网络的重要体现；终端设备层主要用于提供人机界面，虽然物联网是“物物相连的网”，但最终是要以人为本的，还是需要人的操作与控制，不过这里的人机界面已经远远超出了人与计算机交互的概念，而是泛指与应用程序相连的各种设备与人的反馈。

物联网的应用可以分为监控型（物流监控、交通监控、污染监控等）、查询型（远程抄表、智能检索等）、控制型（智能家居、智慧城市、智能交通等）、扫描型（手机钱包、扫码支付等）。目前，软件开发、智能控制技术迅速发展，应用层技术将会为用户提供更加丰

富多彩的物联网应用。同时，各行各业和家庭应用的开发也将会推动物联网的普及，给整个物联网产业链带来利润。

2. 应用层的关键技术

物联网的应用能够为用户提供丰富多彩的业务体验，如何合理高效地处理和利用从网络层传来的数据，并从中提取有效的信息，是物联网应用层要解决的一个关键问题。下面将对应用层中涉及的云服务、中间件等技术进行介绍。

（1）云服务

随着互联网时代信息与数据的快速增长，大规模、海量的数据需要处理。为了节省成本和系统的可扩展性，云服务（Cloud Computing）概念应运而生。云服务是一个美好的网络应用模型，由 Google 首先提出。云服务最基本的核心是通过网络将庞大的计算处理程序自动拆分为无数个较小的子程序，再交给由多个服务器所组成的庞大系统，经搜索、计算分析之后将结果回传给用户。

通过云服务技术，网络服务提供者可以在数秒内处理数以千万计的数据，达到与超级计算机具有同样强大效能的网络服务。这种服务可以与 IT 软件、互联网相关，也可以是任意的其他服务，具有超大规模、虚拟化、可靠安全等独特的功效。云服务的核心是提供服务。

物联网的发展需要“软件即服务”“平台即服务”及按需计算等云服务模式的支撑。可以说，云服务是物联网应用发展的基石。其原因有两个：一个是云服务具有超强的数据处理和存储能力；另一个是由于物联网无处不在的数据采集，需要大范围的支撑平台以满足规模需求。

云服务以下面几种方式支撑物联网的应用发展。

① 单中心、多终端应用模式。

在单中心、多终端应用模式中，分布范围较小的各物联网终端把云中心或部分云中心作为数据处理中心，终端所获得的信息和数据统一由云中心处理和存储，云中心提供统一界面给使用者操作或者查看。

② 多中心、多终端应用模式。

多中心、多终端应用模式主要用于区域跨度较大的企业和单位。例如，一个跨多地区或多个国家的企业，因其分公司或者分厂较多，需要对各分公司或者分厂的生产流程进行监控、对相关产品进行监控跟踪等。当有些数据或者信息需要及时甚至实时与各个终端用户共享时也可以采取这种模式。例如，假若某气象预测中心探测到某地 30 分钟后将发生重大气象灾害，则只需要通过以云服务为支撑的物联网途径，用几十秒的时间就能将预报信息发出。这种应用模式的前提是云服务中心必须包含公共云和私有云，并且它们之间的互联没有障碍。

③ 信息与应用层的处理、海量终端的应用模式。

这种应用模式主要是针对用户范围广、信息和数据种类多、安全性要求较高等特征实现

的物联网。根据应用模式和具体场景，对各种信息、数据进行分类、分层处理，然后选择相关途径提供给相应的终端。例如，对于需要大量传送、对安全性要求不高的数据，如视频数据、游戏数据等，可以采取本地云中心处理或存储的方式；对于计算要求高、数量不大的数据，则可以放在专门负责高端运算的中心；对于安全要求非常高的信息和数据，则可以由具有灾备中心的云中心处理。实现云服务的关键技术是虚拟化技术。通过虚拟化技术，单个服务器可以支持多个虚拟机运行多个操作系统和应用，从而提高服务器的利用率。

（2）中间件技术

中间件是为了实现每个小的应用环境或系统的标准化及它们之间的通信，在后台应用软件和读写器之间设置的一个通用平台和接口，在许多物联网体系架构中，经常把中间件单独划分为一层，位于感知层与网络层或者网络层与应用层之间。

在物联网中，中间件作为软件部分有着举重若轻的地位。物联网中间件是在物联网中采用中间件技术，实现多个系统或多种技术之间的资源共享，最终组成一个资源丰富、功能强大的服务系统，最大限度地发挥物联网系统的作用。具体来说，物联网中间件的主要作用在于将实体对象转换为信息环境下的虚拟对象。因此，数据处理是中间件最重要的功能。同时，中间件具有数据的搜集、过滤、整合和传递等特性，以便将正确的对象信息传递给后端的应用系统。

目前，IBM、Microsoft、BEA、Reva 等公司都提供物联网中间件产品，对建立物联网的应用体系进行了尝试，为物联网将来的大规模应用提供了支撑。

（3）M2M 技术

M2M 是 Machine-to-Machine（机器对机器）的缩写，根据不同的应用场景，往往也被解释为 Man-to-Machine（人对机器）、Machine-to-Man（机器对人）、Mobile-to-Machine（移动网络对机器）、Machine-to-Mobile（机器对移动网络）等。由于 Machine 一般特指人造的机器设备，而物联网（The Internet of Things）中的 Things 则是指更抽象的物体，范围也更广。例如，树木和动物属于 Things，可以被感知、被标记，属于物联网的范畴，它们不是 Machine，不是人为事物；电冰箱属于 Machine，同时也是一种 Things。所以，M2M 可以看作是物联网的子集或应用。

M2M 是现阶段物联网普遍的应用形式，是实现物联网的第一步。M2M 业务在现阶段通过结合通信技术、自动控制技术和软件智能处理技术，可以实现对机器设备信息的自动获取和自动控制。这个阶段的通信对象主要是机器设备，尚未扩展到任何物品，在通信过程中，以使用离散的终端节点为主，并且 M2M 的平台也不等于物联网的运营平台，只解决物与物的通信，解决不了物联网智能化的应用。随着应用软件的发展和中间件软件的发展，M2M 平台将可以逐渐过渡到物联网的应用平台上。

M2M 将多种不同类型的通信技术有机地结合在一起，将数据从一台终端传送到另一台终端，也就是机器与机器的对话。M2M 技术综合数据采集、GPS、远程监控、电信、工业控制等技术，可以在安全监测、自动抄表、机械服务、维修业务、自动售货机、公共交通系

统、车队管理、工业流程自动化、电动机械、城市信息化等环境中运行并提供广泛的应用和解决方案。

M2M 技术的目标就是使所有的机器设备都具有联网和通信能力。其核心理念就是网络一切（Network Everything）。随着科学技术的发展，越来越多的设备都具有通信和联网能力，网络一切逐步变为现实。

M2M 技术具有非常重要的意义，有着广阔的市场及应用，将会推动社会生产方式和生活方式的新一轮变革。

（4）人工智能

人工智能（Artificial Intelligence，AI）是探索使各种机器模拟人类的某些思维过程和智能行为（学习、推理、思考、规划等），使人类的智能得以物化与延伸的一门学科。人工智能领域的研究主要包括机器人、语言识别、图像识别、自然语言处理、音频事件检测等，目前的主要方法有神经网络、进化计算和粒度计算等。在物联网中，人工智能技术主要负责分析物品所承载的信息内容，从而实现计算机的自动处理。

人工智能技术的优点在于：大大改善操作者的作业环境，减轻工作强度；提高作业质量和工作效率；一些危险场合或者重点施工的应用得到解决；环保、节能；提高机器的自动化处理进程及智能水平；提高设备的可靠性，降低维护成本；故障诊断可实现智能化；等等。

（5）数据挖掘

数据挖掘（Data Mining，DM）是从大量的、不完全的、有噪声的、模糊的及随机的实际应用数据中，挖掘出隐含的、未知的、对决策有潜在价值的数据的过程。数据挖掘主要基于人工智能、机器学习、模式识别、统计学、数据库、可视化技术等，高度自动化地分析数据，做出归纳性的推理。数据挖掘一般分为描述型数据挖掘和预测型数据挖掘两种：描述型数据挖掘包括数据总结、聚类及关联分析等；预测型数据挖掘包括分类、归纳及时间序列分析等。通过对数据的统计、分析、综合、归纳和推理，揭示事件间的相互关系，预测未来的发展趋势，为决策者提供决策依据。

在物联网中，数据挖掘只是一个代表性的概念，是一些能够实现物联网“智能化”“智慧化”的分析技术和应用的统称。细分起来，数据挖掘技术包括数据挖掘和数据仓库、决策支持、商业智能、报表、在线数据分析、平衡计分卡等技术应用。

第 2 章
走进智能家居

智能家居的概念起源于 20 世纪 70 年代的美国，随后传播到欧洲、日本等，并且得到很好的发展。在我国，虽然智能家居概念的推广较晚，在 20 世纪 90 年代末才开始，但发展速度很快，当前国内已经出现了相当多的智慧家庭、智能小区等。

近几年，随着传感器技术、控制技术、云服务、大数据、移动互联等基础应用的迅速发展，智能家居的系统组成更加多样化，功能延伸更加广泛，对智能家居的定义也更加宽泛。简单来说，智能家居是在互联网影响之下物联化的体现，是以住宅为平台，利用综合布线技术、网络通信技术、安全防范技术、自动控制技术、音/视频技术将家居生活的有关设施集成，构建高效的住宅设施与家庭日程事务的管理系统，提升家居的安全性、便利性、舒适性、艺术性，从而实现环保节能的居住环境。智能家居不仅体现在家居单品的智能化，还体现在不同设备之间互联互通构成的完整生态系统的智慧化。未来，在统一的平台上，用户家庭里的各类家用电器、灯光、窗帘、安防、健康保健、家庭娱乐等全系列的家居设备均可实现跨品牌、跨产品的互联互通和融合。

智能家居主要体现在以下几个方面。

1. 家庭自动化（Home Automation）

家庭自动化是指利用微处理器技术集成或控制家庭中的电子电器产品或系统，如照明灯、咖啡炉、电脑设备、保安系统、暖气和冷气系统、视讯和音响系统等。家庭自动化系统主要是以一个中央微处理器接收来自相关电子电器产品的信息（外界环境因素的变化、控制指令等）后，再以既定的程序发送适当的信息给其他的电子电器产品。中央微处理器必须透过许多界面控制家庭中的电子电器产品。这些界面可以是键盘，也可以是触摸式荧幕、按钮、电脑、电话、遥控器等。用户可发送信号至中央微处理器或接收来自中央微处理器的信号。

家庭自动化是智能家居的一个重要系统。在智能家居刚出现时，家庭自动化甚至就等同于智能家居。目前，家庭自动化仍是智能家居的核心之一。随着网络技术在智能家居的普遍应用，智能家用电子电器产品的成熟，家庭自动化的许多产品功能将融入这些新的产品中，

使单纯的家庭自动化产品在系统设计中越来越少，其核心地位也将被智能家用电子电器产品系统所代替。家庭自动化将作为家庭网络中的控制网络部分在智能家居中继续发挥作用。

2. 家庭网络（Home Networking）

家庭网络区别于纯粹的“家庭局域网”，是在家庭范围内（可扩展至邻居，小区）将PC、家用电子电器产品、安全系统、照明系统和广域网相连接的一种新技术。目前，家庭网络所采用的连接技术可以分为“有线”和“无线”两大类。有线方案主要包括双绞线或同轴电缆连接、电话线连接、电力线连接等；无线方案主要包括红外线连接、无线电连接、基于RF技术的连接和基于PC的无线连接等。

家庭网络相比传统的办公网络来说，加入了很多家庭应用产品和系统，如家用电子电器产品、照明系统，因此相应的技术标准也错综复杂。

家庭网络的发展趋势是将智能家居中其他系统融合进去，最终一统天下。

3. 智能家用电子电器产品（Intelligent Home Appliances ）

智能家用电子电器产品并不是单指某一种家用电子电器产品，而应是一种技术系统。随着人类的应用需求和家用电子电器产品智能化的不断发展，其内容将会更加丰富。根据实际应用环境的不同，智能家用电子电器产品的功能也会有所差异，一般应具备以下基本功能：

① 通信功能，包括电话、网络、远程控制/报警等。

② 消费电子产品的智能控制：可以自动控制加热时间、加热温度的微波炉；可以自动调节温度、湿度的智能空调器；可以根据指令自动搜索电视节目并摄录的电视机/摄像机；等等。

③ 交互式智能控制，可以通过语音识别技术实现智能家用电子电器产品的声控功能；通过各种主动式传感器（温度、声音、动作等）实现智能家用电子电器产品的主动性动作响应；用户还可以自己定义不同场景、不同智能家用电子电器产品的不同响应。

④ 安防控制功能，包括门禁系统、火灾自动报警及煤气泄漏、漏电、漏水的报警等。

⑤ 健康与医疗功能，包括健康设备的监控、远程诊疗、老人/病人异常监护等。

智能家用电子电器产品就是将微处理器、传感器技术、网络通信技术引入家用电子电器产品设备后形成的家用电子电器产品，具有自动感知住宅空间的状态、家用电子电器产品自身的状态、家用电子电器产品的服务状态，能够自动控制及接收住宅用户在住宅内或远程的控制指令。同时，智能家用电子电器产品作为智能家居的组成部分，能够与住宅内其他家用电子电器产品和家居、设施互联组成系统，实现智能家居功能。

与传统的家用电子电器产品相比，智能家用电子电器产品具有如下特点：

① 网络化功能。各种智能家用电子电器产品可以通过家庭局域网连接在一起，通过家庭网关接口与制造商的服务站点连接，最终与互联网连接，实现信息的共享。

② 智能化。智能家用电子电器产品可以根据周围环境的不同自动做出响应，不需要人

为干预，如智能空调器可以根据不同的季节、气候及用户所在地域，自动调整工作状态，达到最佳的效果。

③ 开放性、兼容性。由于用户家庭的智能家用电子电器产品可能来自不同的厂商，智能家用电子电器产品的平台必须具有开发性和兼容性。

④ 节能化。智能家用电子电器产品可以根据周围的环境自动调整工作时间、工作状态，从而实现节能。

⑤ 易用性。由于复杂的控制操作流程已由内嵌在智能家用电子电器产品中的控制器解决，因此用户只需要了解非常简单的操作即可。

2.1 智能家居的发展状况

2.1.1 国外发展现状

1984 年，比尔·盖茨的家庭是国外第一个使用智能家居的。比尔·盖茨家庭的智能建筑是通过对旧建筑进行一定程度的改造后完成的，推出了情报资料、语音通信、电子邮件等方面的服务，通过计算机技术对整个楼宇的电器设备进行监控。这是第一次出现的真正意义上的智能建筑，从此开启了智能家居系统的新篇章。之后，新加坡、日本、德国、澳大利亚等一些经济发达的国家陆续开始了对智能家居系统的研究。

智能家居在不同时期采用不同的技术各自实现了不同的功能。从 20 世纪 80 年代开始，由于家用电子电器产品的大量使用，因此出现了住宅的电子化，然后将功能独立的家用电子电器产品、通信设备集成，又形成了住宅的自动化。20 世纪 90 年代初，随着总线技术的发展，实现了利用总线技术对家用电子电器产品、通信、安保等设备进行监视、管理和控制，形成了早期的智能家居系统。早期智能家居系统的主要功能包括集中控制、安全防护、环境控制、终端智能控制等。此时的智能家居系统采用综合布线的形式，建设成本高，难以改造。随着技术的发展，2000 年，智能家居系统从有线方式逐步发展为有线与无线相结合的形式。近几年，随着物联网、大数据、云计算、人工智能等技术的迅速发展，无线通信技术得到广泛的应用，智能家居系统通过 WiFi、ZigBee、蓝牙等无线技术实现控制，减少了布线的麻烦，使智能家居系统更加整体化，管理更加方便。

目前，在全球范围内，涉及家用电子电器产品、计算机及通信等领域的公司都把目光转移到智能家居和信息家电的发展上。同样，硅谷也不例外，将家居系统智能化技术作为投资和研发的热点。从目前来看，国外针对智能家居的开发技术不断完善，用户数量也越来越多，有一半以上用户的家庭都已具备智能家居系统的条件，伴随着技术的不断发展，新兴的智能家居系统实现起来会更容易，将会比当年比尔·盖茨花费高达 6000 万美元改造的高端

智能家居便宜得多。

2014 年初，Google 以 3.2 亿美元的报价收购了由托尼·法德尔（Tony Fadell）创办的智能家居公司 Nest Labs。托尼·法德尔是苹果公司 iPod 部门的主管，被誉为 iPod 之父。因此，Nest Labs 从创办之初就备受关注，开发了一个智能家居的平台，开放了 API 给开发者，让开发者可以利用 Nest Labs 的硬件和算法，将其他智能家用电子电器产品和 Nest Labs 的产品连接在一起，进而实现对空调器、电冰箱、台灯、电风扇等各式各样产品的智能化进行控制。目前，Nest Labs 可支持的设备包括 Nest 烟雾探测器、Dropcam Pro、Big Ass 风扇、Lifx 智能灯泡、罗技 Harmony 遥控器、Jawbone Up24 运动手环等。随着 Google 宣布收购 Nest Labs，Nest Labs 火了，智能家居也彻底被引爆了，从而拉开了“智能家居元年”的序幕。

目前，已有许多国际知名公司都进入到智能家居系统的领域，如美国的苹果公司、Honeywell、比利时的 TELETASK、新加坡的 NICO 及英国的 ARM 公司等。它们都在大力加强在智能家居系统领域的研发力度。不同的公司针对的市场不同，从而所从事的研发具有不同的特点。例如，TELETASK 专注于系统的控制，其家居自动化系统采用模块化的结构设计，每个模块的接口都由 AUTOBUS 总线进行连接，系统稳定可靠，具有很大的扩展空间；苹果公司在移动平台 iOS 系统上开放了 Home Kit 的智能家居管理应用，通过平台的形式进入智能家居市场；英国的 ARM 公司发布了面向智能家居领域的芯片设计方案，以期抓住物联网带来的芯片商机。此外，英特尔、联发科、三星、意法半导体也抓住这一机会，纷纷展开布局。

智能家居将作为下一个产业新的增长点，受到越来越多的重视。

2.1.2 国内发展现状

与国外相比，国内智能家居系统的起步相对较晚。20 世纪 90 年代后期，住宅智能化才刚刚开始，在 1997 年制定了《小康住宅电气设计（标准）导则》，开始努力探索如何践行互联网思维。2000 年，在《小康型城乡住宅科技产业工程项目实施方案》中把智能化小康示范小区列入国家重点发展方向。2010 年，国务院提出“三网融合”的概念并将一部分城市作为试点对象。

从 2014 年开始，随着移动互联网的高速发展，通信网络向 5G 演进，国家对云服务、三网融合等的推动作用成为发展智能家居等物联网应用产业的强大驱动力，给智能家居的发展带来机遇。云服务、物联网产业的发展进入高速成长期和全面整合期，给智能家居带来广阔的想象空间。智能家居涉及智能照明、智能开关、智能电器、智能传感、智能安保、智能健康等。

我国的智能家居从产品的角度可以分为三个发展阶段：第一，单品智能化阶段；第二，产品之间的联动阶段；第三，全面智能化阶段。

在第一个阶段，许多新生创业公司和小公司会从小型的家用设备着手，可以实现插座、

电灯、家用摄像头等电子设备的智能开发。例如，BroadLink 作为一家创业公司，在生产和销售智能插座的同时，将 WiFi 模块当成一个高优先级的业务，已有 80 多家家电企业采用了 BroadLink 的 WiFi 模块，包括万和、老板、志高、格兰仕、奥克斯、海信、万家乐、苏泊尔、奔腾、联想、长化、欧普等知名企业。于是，BroadLink 曲线进入了智能家电市场。大型家电企业会利用自身产业的优势进行大型家电设备的智能化开发，可以实现空调器、电冰箱、电视机等家电设备的智能化。在家居环境中，由于大型家用电子电器设备是必不可少的部分，智能家居的发展无法忽略这些设备，因此各个大型家电企业会占有一定的先天优势。例如，海尔集团从 2014 年开始将智慧家庭的概念作为主线，着手智能家电的研究，推出了相应的一些智能产品，如 Iseemini 智能投影、智能路由器、空气盒子、醒知道等，并且还通过与其他 40 多家企业的合作，推出了 GE 智慧照明、蓝信康血压计、土曼智能手表、Risco 安防套装、Power-tech 智能插座、Picooc 体脂仪等产品。

随着单品智能化阶段的不断发展，就会进一步进化到第二阶段，即不同产品的联动阶段。国内已有部分公司进行产品的数据互通，达成深度合作。例如，麦开和 bong 两家国内生产智能硬件的公司达成了合作关系，可以实现多种多样的智能产品，并且这些智能产品可以相互联动，使不同的智能产品之间的数据可以互通；幻腾智能公司也在致力于智能家居相关产品的研究，实现了自己公司的产品之间可以互联互动，推出的产品有 Sushi 安防门窗、Nova 智能灯泡及 UFO 红外传感器等，门窗的控制传感器可以与灯泡之间产生互联互动。

最后是第三个阶段，也是智能家居系统智能化的实现阶段，是一个对目前来说比较困难的阶段。许多的研究人员和公司都在致力于相应的研究工作。

在国家“十三五”规划期间，智能家居必然成为抢手的行业，互联网企业想借助自身的技术优势开拓新的市场，传统家电企业希望抓住这个机遇完成智能化升级，投资机构则看准了智能家居市场巨大的资本潜力。

智能家居改变生活，生活依赖智能家居，在国家政策的大力扶持之下，智能家居的发展将迎来新一轮的机遇和挑战。

2.2 智能家居组网技术基础

2.2.1 组网方式分类

智能家居领域由于具有多样性和个性化的特点，因此导致组网技术路线多样、没有统一通行技术标准体系的现状。从技术应用角度，组网方式主要分为三类。

1. 总线技术类

总线技术的主要特点是所有设备的通信和控制都集中在一条总线上，是一种全分布式智

能控制的网络技术。其产品模块具有双向通信能力及互操作性和互换性，控制部件都可以编程。典型的总线技术采用双绞线总线结构，各网络节点可以从总线上获得供电，也可以通过同一条总线实现节点间无极性、无拓扑逻辑限制的互联和通信。

总线技术类的产品比较适合楼宇智能化及小区智能化等大区域范围的控制，一般设置安装比较复杂，造价较高，工期较长，只适用于新装修的用户。

2. 无线通信技术类

无线通信技术众多，已经成功应用在智能家居领域的无线通信技术方案主要包括射频（RF）技术、WiFi 技术、蓝牙技术、VESP 协议、IrDA 红外线技术、HomeRF 协议、ZigBee 标准、Z-Wave 标准及 Z-world 标准等。

无线通信技术方案的主要优势在于不需要重新布线，安装方便灵活，根据需求可以随时扩展或改装，适用于新装修的用户和已装修过的用户。

3. 电力线载波通信技术

电力线载波通信技术充分利用现有的电网，两端加调制解调器，直接以 50Hz 交流电为载波，再以数百千赫兹的脉冲为调制信号，进行信号的传输与控制。

2.2.2 主流技术分析

虽然智能家居联网标准的争议由来已久，但由于无线通信技术不需要破墙布线，只需要根据用户的要求组合安装，系统就能自动组网，因此目前基本达成一致的观点是，无线通信技术比有线通信技术更具有潜力。无线通信技术有很多种，哪种最好尚无定论，WiFi、ZigBee、Z-wave 及蓝牙逐渐成为最佳选择。

四种技术各自的优、缺点是家庭网络发展需要考虑的因素。

1. WiFi 技术

前面提到，WiFi 是一种可以将个人电脑、手持设备等终端以无线方式互相连接的技术。简单来说就是 IEEE 802.11b 的别称，是一种短程无线传输技术，能够在几十米的范围内支持互联网接入的无线电信号。经过不断的开发和研制，随着 IEEE 802.11a、IEEE 802.11g 等许多标准的出现，目前 IEEE 802.11 其实是一个协议族，被称为 802.11x 系列标准，包含一系列的无线局域网协议标准，已被统称为 WiFi。WiFi 可以帮助用户访问电子邮件、Web 和流媒体，为用户提供了无线的宽带互联网访问，同时，也是在家里、办公室或在旅途中上网的快速、便捷的途径。WiFi 无线网络是由 AP（Access Point）和无线网卡组成的，在开放性区域，通信距离可达 305m；在封闭性区域，通信距离为 76~122m，方便与现有的有线以太网络整合，组网的成本更低。WiFi 以其自身的诸多优点，受到用户的推崇。

(1) IEEE 802.11 协议

IEEE802.11 是美国电气和电子工程师协会 IEEE 在 1997 年 6 月颁布的无线网络标准，是第一代无线局域网的标准之一。IEEE 802.11 规定，无线局域网需要在 2.4GHz 波段中进行操作，主要用于解决办公室局域网和园区网中用户与用户终端的无线接入，传输速率最高只能达到 2Mb/s。

IEEE 802.11 协议主要工作在 ISO 协议的最低两层上，并在物理层上进行了一些改动，加入了高速数字传输的特性和连接的稳定性。IEEE 802.11 的体系结构如图 2.1 所示。

图 2.1 IEEE 802.11 的体系结构

① IEEE 802.11 物理层的实现方式。

在 IEEE802.11 标准中的物理层定义了数据传输的信号特征和调制。

在物理层中定义了两个 RF 传输方法和一个红外线传输方法。RF 传输方法采用扩频调制技术来满足绝大多数国家的工作规范。在 IEEE802.11 标准中，RF 传输标准是跳频扩频（FHSS）和直接序列扩频（DSSS），工作在 2.4000～2.4835GHz 频段。直接序列扩频采用 BPSK 和 DQPSK 调制技术，支持 1Mb/s 和 2Mb/s 数据速率。跳频扩频采用 2～4 电平 GFSK 调制技术，支持 1Mb/s 数据速率，共有 22 组跳频图案，包括 79 个信道。红外线传输方法工作在 850～950nm 段，峰值功率为 2W，支持数据速率为 1Mb/s 和 2Mb/s。

② MAC 的结构及服务内容。

IEEE802.11 的 MAC 子层负责解决客户端的工作站和访问接入点之间的连接，当一个 IEEE802.11 的客户端进入一个或者多个接入点的覆盖范围时，会根据信号的强弱及包错误率自动选择 1 个接入点进行连接，一旦被 1 个接入点接受，客户端就会将发送接收信号的频率切换到接入点的频段。这种重新协商通常发生在无线工作站移出原连接的接入点服务范围的信号衰减之后，还可能发生在由于建筑物造成信号变化或者原接入点出现拥塞等情况。

无线局域网 MAC 提供的服务有安全服务、MAC 服务数据单元（MSDU）重新排序服务及数据服务。IEEE 802.11 中安全服务提供的服务范围局限于站与站之间的数据交换。其内容为加密、验证、与层管理实体相联系的访问控制。为提高成功发送的可能性，MAC 提供了重新排序服务，只有在节电方式工作下的站，且不处于激活状态时，才可先将 MSDU 缓存起来，等站被激活时再突发出去，对缓存数据进行重新排序。

MAC 数据服务可使对等 LLC 实体进行数据单元的交换。本地 MAC 利用下层的服务将 1 个 MSDU 传给 1 个对等的 MAC 实体，然后又传给对等的 LLC 实体，当信道特性限制长帧传输的可靠性时，可通过增加 MSDU 成功传输的可能性增加可靠性。

IEEE802.11 MAC 子层还提供了 CRC 校验和包分片功能，每一个在无线网络中传输的数

据报都被附加上校验位以保证在传送的时候没有出现错误。包分片功能可允许大的数据报在传送的时候被分成较小的部分分批传送。这在网络十分拥挤或者存在干扰的情况下是一个非常有用的特性，可以减少数据报被重传的概率。MAC 子层负责将收到的被分片的大数据报重新组装。对于上层协议，这个分片的过程是完全透明的。

③ CSMA/CA 协议。

IEEE802.11 的 MAC 和 IEEE802.3 协议的 MAC 非常相似，都是在一个共享介质上支持多个用户共享资源，发送方在发送数据前先进行网络的可用性检测。IEEE 802.3 协议采用 CSMA/CD 介质访问控制方法。在无线系统中，设备不能一边接收数据信号一边传送数据信号。

无线局域网采用 1 种与 CSMA/CD 相类似的载波监听多路访问/冲突防止协议（Carrier Sense Multiple Access with Collision Avoidance，CSMA/CA）实现介质资源共享。CSMA/CA 利用确认信号避免冲突的发生，也就是说，只有当客户端收到网络上返回的确认信号后，才可确认送出的数据已经正确到达目的地。CSMA/CA 通过这种方式来提供无线的共享访问，在处理无线问题时非常有效，以 CSMA/CA 的方式共享无线介质将时间域的划分与帧格式紧密联系起来，保证某一时刻只有 1 个站点发送，实现了网络系统的集中控制。

因传输介质不同，CSMA/CD 与 CSMA/CA 的检测方式也不同。CSMA/CD 通过电缆中的电压变化来检测，当数据发生碰撞时，电缆中的电压就会随着发生变化；CSMA/CA 采用能量检测（ED）、载波检测（CS）及能量载波混合检测 3 种检测信道空闲的方式进行检测。

（2）IEEE 802.11 标准分类

IEEE 于 1999 年 8 月，在 IEEE802.11 标准的基础上相继推出了 IEEE802.11b 和 IEEE802.11a 两个标准，规范了不同频点的产品及更高网络速率产品的开发和应用，在原 IEEE802.11 的内容之外，增加了基于简单网络管理协议（Simple Network Management Protocol，SNMP）的管理信息库（Management Information Base，MIB），以取代原 OSI 协议的管理信息库，另外还增加了高速网络的内容。

2001 年 11 月，IEEE 又推出第 3 个新的标准 IEEE802.11g。尽管目前 IEEE802.11a 和 IEEE802.11g 倍受业界关注，但从实际的应用上来说，IEEE802.11b 已成为无线局域网的主流标准，被多数厂商所采用，并且已经有成熟的无线产品推向市场。

下面就简单介绍一下 IEEE 802.11 的几种标准。

① IEEE 802.11a。

IEEE802.11a 在整个覆盖范围内提供了更高的速度，规定的频点为 5GHz。目前该频段用得不多，干扰和信号争用情况较少。IEEE802.11a 同样采用 CSMA/CA 协议。但在物理层中，IEEE802.11a 采用了正交频分复用（Orthogonal Frequency Division Multiplexing，OFDM）技术。

OFDM 技术的最大优势是多途径回声反射，特别适合室内及移动环境，传输速率为 6~54Mb/s，支持语音、数据、图像业务。这样的速率完全能满足室内、室外的各种应用场合。

OFDM 技术将 1 个无线信道分解成多个子载波同时传输数据，每个子载波的传输速率比总传输速率低许多，也就是每个传输符号的时长要长许多，有利于克服无线信道的衰落，改善信号的质量，提升整个网络的速度。通过对标准物理层的扩充，IEEE802.11a 支持的数据传输速率最高可达 54Mb/s。

② IEEE 802.11b。

IEEE802.11b 工作在开放的 2.4GHz 频点，不需要申请就可以使用，既可作为对有线网络的补充，也可独立组网，可使网络用户摆脱网线的束缚，实现真正意义上的移动应用。IEEE 802.11b 的关键技术之一是采用补偿码键控 CCK 调制技术，可以实现动态速率转换。当工作站之间的距离过长或干扰过大，信噪比低于某个限值时，其传输速率可从 11Mb/s 自动降至 5.5Mb/s，或者再降至直接序列扩频技术的 2Mb/s 和 1Mb/s 传输速率，IEEE 802.11b 标准传输速率的上限为 20Mb/s，可保持对 IEEE 802.11 的向后兼容。

IEEE802.11b 支持的范围是室外 300m，在办公环境中最远为 100m，当用户在楼房或公司部门之间移动时，允许在访问点之间进行无缝连接。IEEE802.11b 还具有良好的可伸缩性，最多 3 个访问点可以同时定位于有效的使用范围中，可支持上百个用户。目前，IEEE 802.11b 无线局域网技术已经在世界上得到了广泛的应用，已经进入了写字间、饭店、咖啡厅和候机室等场所。

③ IEEE 802.11g 和 IEEE 802.11e。

IEEE802.11a 和 IEEE802.11b 的产品因为频段与调制方式的不同而无法互通，使得已经拥有 IEEE 802.11b 产品的用户可能不会立即购买 IEEE802.11a 的产品，阻碍了 IEEE802.11a 的应用步伐。2001 年，IEEE 批准一种新技术 IEEE802.11g，其使命就是兼顾 IEEE802.11a 和 IEEE802.11b，为 IEEE802.11b 过渡到 IEEE802.11a 铺路修桥。

IEEE 802.11g 既适应传统的 IEEE802.11b 标准，在 2.4GHz 频率下提供 11Mb/s 的数据传输速率，也符合 IEEE802.11a 标准，在 5GHz 频率下提供 54Mb/s 的数据传输速率。IEEE802.11g 中规定的调制方式包括 IEEE802.11a 中采用的 OFDM 与 IEEE802.11b 中采用的 CCK，通过规定两种调制方式，既达到了用 2.4GHz 频率实现 IEEE 802.11a 54Mb/s 的数据传输速率，也确保了与 IEEE802.11b 产品的兼容。

IEEE 802.11e 被称为下一代 WLAN 标准，是 WLAN 标准方式 IEEE 802.11 的扩展标准。所谓 IEEE 802.11 的扩展标准，就是在现有的 IEEE802.11b 及 IEEE802.11a 的 MAC 层追加了服务质量（Quality of Service，QoS）功能及安全功能的标准。

④ IEEE 802.11n。

从最初的版本到 IEEE802.11g 都是关于传输速率的提升。IEEE802.11n 是最早尝试使用 5GHz 频率的，因为此频率会少许多干扰。IEEE 802.11n 是在 IEEE 802.11g 和 IEEE 802.11a 之上发展起来的一项技术，最大特点是速率得到了提升，理论速率最高可达 600Mb/s（目前业界主流为 300Mb/s）。IEEE 802.11n 采用智能天线技术，可以通过多组独立天线组成的天线阵列动态调整波束，保证能够让 WLAN 用户接收到稳定的信号，并可以减少其他信号的

干扰，覆盖范围可以扩大到几平方公里，使 WLAN 的移动性极大提高。

IEEE 802.11n 采用了一种软件无线电技术，是一个完全可编程的硬件平台，可使不同系统的基站和终端都可以通过这一平台的不同软件实现互通和兼容，使 WLAN 的兼容性得到极大改善，意味着 WLAN 将不但能实现 IEEE 802.11n 向前后兼容，还可以实现 WLAN 与无线广域网络的结合，如 3G。

⑤ IEEE 802.11ac。

IEEE 802.11ac 通过 5GHz 频带进行通信，大幅提高了数据传输的速度，在理论上能够提供最少 1Gb/s 的数据传输速度进行多站式无线局域网通信，或是最少 500Mb/s 数据传输速度的单一连接传输，是目前为止最快的 WiFi 标准。

(3) WiFi 技术的主要特点

① 无线电波的覆盖范围广 。WiFi 的覆盖半径可达 100m，适合办公室及单位楼层内部使用。蓝牙技术的覆盖半径只有 15m。

② 速度快，可靠性高 。如 IEEE802.11b 无线网络，最高带宽为 11Mb/s，在信号较弱或有干扰的情况下，带宽可调整为 5.5Mb/s、2Mb/s 和 1Mb/s，带宽的自动调整，可有效保障网络的稳定性和可靠性。

③ 不需要布线 。WiFi 可以不受布线条件的限制，非常适合移动办公用户的需要，具有广阔的市场前景。目前，WiFi 已经从传统的医疗保健、库存控制和管理服务等特殊行业向更多的行业拓展，甚至开始进入家庭及教育机构等领域。

④ 健康安全 。IEEE802.11 规定，WiFi 的发射功率不可超过 100mW，实际发射功率为 60~70mW，手机的发射功率为 0.2~1W，手持式对讲机的发射功率高达 5W。无线网络的使用方式并非像手机那样直接接触人体，是绝对安全的。

目前，基于 WiFi 技术的智能家居产品最为常见。其优势在于传输速率快，产品成本低，生活中也最为普及。对用户来说，基于 WiFi 的智能家居组合最为省事，购买设备后，直接组网即可。

凡事都存在两面性，WiFi 虽然传输快、普及广，但也存在自身的技术劣势。其最大的问题是安全性非常低，无线稳定性弱。一般的 WiFi 路由器组件廉价，编程简单，黑客盗取密码手到擒来，若有调皮的黑客入侵某个家庭网络中心，激活水龙头开关，简直分分钟能把房子淹了；功耗大也是其缺点之一，将导致在家居领域中的应用受限，如智能门锁、红外转发控制器、各种传感器等不适宜使用；此外，WiFi 的组网能力也相对较低，目前，WiFi 网络的实际规模一般不超过 16 个设备，而在实际的家居环境中，仅开关、照明、家电的数量就已远远多于 16 个，显然发展空间受到了一定的限制。

2. ZigBee 技术

ZigBee 是 ZigBee 联盟在 IEEE802.15.4 标准上制定的一种技术规范。ZigBee 实质上是一个通信标准，定义了短距离、低速率无线通信所需要的一类通信标准，可以工作在 2.4GHz、

868MHz、915MHz 频段，可以与 254 个节点联网，采用跳频技术。250kb/s 是 ZigBee 的基本传输速率，当传输速率达到 28kb/s 的时候，ZigBee 的传输范围能够增加到 134m，并且有更高的可靠性。因此，ZigBee 技术可以应用到众多领域，如自动控制领域、远程控制领域、嵌入式设备领域等。作为一种无线网络技术，ZigBee 具有近距离、低传输速率、低功耗等特点，主要用于近距离的无线连接。ZigBee 技术所使用的传感器能耗极低，以接力的方式将数据从一个网络节点传到其他网络节点，具有很高的传输效率。

ZigBee 网络具有三种拓扑结构：星状网、树状网和环状网。在星状网拓扑结构中，所有的终端设备只与协调器之间进行通信，协调器作为发起设备，建立一个自己的网络，路由设备和终端设备加入网络；树状网是由一个协调器、多个路由器及终端节点构建而成的，由协调器发起网络，其他节点加入网络；环状网的实现基于树状网，每个节点都可以与网络内的其他节点实现通信。

（1）ZigBee 协议规范

ZigBee 协议的体系结构总体可分为两部分。其中，一部分包括物理层（PHY）和媒体访问控制层（MAC），这两层是由 IEEE802.15.4 定义的；另一部分就是网络层（NWK）和应用层（APL），这两层是由 ZigBee 联盟在 IEEE802.15.4 标准的基础上定义的架构。

Z-Stack 协议栈即 ZigBee 协议栈，就是将所有相关的协议进行融合，以函数的形式呈现给用户，方便用户调用。Z-Stack 协议的框架结构如图 2.2 所示。

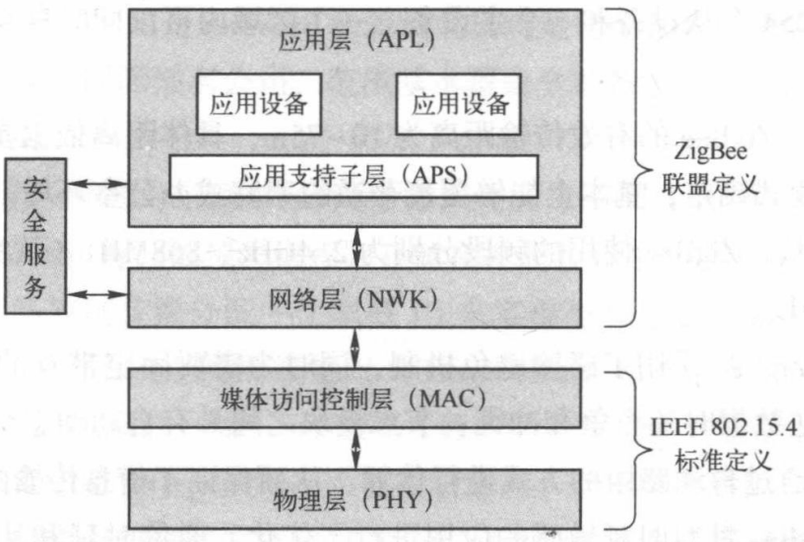


图 2.2　Z-Stack 协议的框架结构

Z-Stack 协议采用分层的思想，各层分工协作，相互支持，实现数据的传送。物理层主要负责 ZigBee 的激活、数据的发送和接收、信道的选择、开启和关闭无线射频收发器、接收机能量的检测等工作。媒体访问控制层为物理层提供相关的数据服务和管理服务。数据服务为媒体访问控制层的协议数据能够在物理层的正确传输提供了保证；管理服务负责媒体访问控制层的管理活动。网络层为媒体访问控制层和应用层提供一定实现性的功能，即数据服务实体和管理服务实体，分别为应用层提数据服务和管理服务。ZigBee 支持的三种网络拓扑结构——星状网、树状网和环状网的拓扑结构正是在网络层进行定义的。应用层主要是由开

发者按需进行定义实现的，可实现应用节点的搜索、维持节点的功能属性、根据需求实现多个节点之间进行的通信。另外，应用层中有两个关键的概念：规范（Profile）和簇（Cluster）。在 ZigBee 网络中进行的数据收发都是建立在规范的基础上的，每个规范都有对应的 ID。规范又可以分为公共规范和制造商规范。公共规范的 ID 范围为 0x0000～0x7FFF；制造商特定规范 ID 的范围为 0x0000～0x FFFF。在规范的基础上，又提出了群集（Cluster）的概念，群集又称为簇。簇是一个应用领域下的一个特定对象，如智能家居系统中的调光器，操作这个调光器需要一些命令，如开灯、关灯、变亮、变暗等，实现这些操作命令就需要簇来定义。

（2）ZigBee 技术的特点

① 数据传输速率低。ZigBee 每个网络模块射频前端的数据传输速率为 10～250kb/s，专注于低传输速率的应用。

② 低功耗。ZigBee 设备具有几种节约电量的工作模式，而且 ZigBee 网络中的通信循环次数非常少，工作周期很短，在待机的情况下，两节五号电池能够使用六个月到两年左右的时间。这也是 ZigBee 相比于其他无线通信技术的优势。

③ 成本低。ZigBee 的模块价格低廉，且 ZigBee 协议是免专利费的，大大降低了成本。

④ 网络容量大。ZigBee 可以采用星状网、簇—树状网、环状网的结构组网，可以通过任一节点连接组成更大的网络结构，从理论上讲，可连接的节点多达 65000 个。一个 ZigBee 网络可以最多容纳 254 个从设备和一个主设备，一个区域内可以同时存在最多 100 个 ZigBee 网络。

⑤ 有效范围小。ZigBee 的有效传输距离为 10～75m，具体距离依据实际发射功率的大小和各种不同的应用模式确定，基本上能够覆盖普通的家庭或办公室环境。

⑥ 工作频段灵活。ZigBee 使用的频段分别为 2.4GHz、868MHz（欧洲）及 915MHz（美国），均为免执照频段。

⑦ 可靠性高。ZigBee 采用了碰撞避免机制，同时为需要固定带宽的通信业务预留了专用时隙，避免了发送数据时的竞争和冲突；节点模块之间具有自动动态组网的功能，信息在整个 ZigBee 网络中通过自动路由的方式进行传输，从而保证了信息传输的可靠性。

⑧ 低时延。ZigBee 针对时延敏感的应用进行了优化，通信时延和从休眠状态激活的时延都非常短，设备搜索时延的典型值为 30ms，休眠激活时延的典型值为 15ms，活动设备信道的接入时延为 15ms。

⑨ 高保密性。ZigBee 提供了基于循环冗余校验（CRC）数据包的完整性检查和鉴权功能，加密算法采用 AES-128，同时，各个应用可以灵活确定其安全属性。

⑩ 自愈功能强。当 ZigBee 网络中的节点发生改变，如减少或者新增加 ZigBee 节点、移动 ZigBee 节点的位置或节点出现问题等情况，能够针对节点的变化对网络拓扑结构进行自我调节，不需要人为干涉，具有很强的自愈能力。

⑪ 网络自组织性强。ZigBee 网络节点能够感知其他 ZigBee 节点的存在，并且确定连接

关系，组成结构化的网络，不需要人工干预。

3. Z-Wave 技术

Z-Wave 是一个低带宽半双工的传输协议，是为高可靠性低功耗环状网的无线通信设计的。协议的主要目的是在控制单元和一个或多个节点单元之间进行可靠的传输较短的控制信息。Z-Wave 的工作频带为 868.42~908.42MHz，采用 FSK（BFSK/GFSK）调制方式，数据传输速率为 9.6kb/s，信号的有效覆盖范围在室内为 30m，在室外可超过 100m，适合于窄宽带的应用场合。随着通信距离的增大，设备的复杂度、功耗及系统的成本都会增加。相对于现有的各种无线通信技术，Z-Wave 技术将是最低功耗和最低成本的技术，可有力推动低速率的无线个人局域网。

Z-Wave 技术主要用于住宅、照明商业控制及状态的读取，如抄表、照明及家电控制、HVAC、接入控制、防盗及火灾检测等。Z-Wave 技术可将任何独立的设备转换为智能网络设备，从而可以实现控制和无线监测。Z-Wave 技术在最初设计时，就定位在智能家居无线控制领域，采用小数据格式传输，40kb/s 的传输速率足以应对，早期甚至使用 9.6kb/s 的速率传输，与同类的其他无线技术相比，拥有相对较低的传输频率、相对较远的传输距离和一定的价格优势。

Z-Wave 是由丹麦公司 Zensys 主导的无线组网规格，Z-Wave 联盟（Z-Wave Alliance）虽然没有 ZigBee 联盟强大，但是 Z-Wave 联盟的成员均是已经在智能家居领域有现行产品的厂商，已经具有 160 多家国际知名公司，范围基本覆盖全球各个国家和地区。

（1）Z-Wave 协议

Z-Wave 协议采用一种名为家庭 ID 的网络标识符区分不同的 Z-Wave 网络。家庭 ID 是一种独一无二的 32 位标识符，被预先编程并存储在控制装置中，所有从设备的初始家庭 ID 都是零，都需要由网络控制装置分配一个家庭 ID 来实现与其他 Z-Wave 网络的通信。每一个 Z-Wave 网络都拥有自己独立的网络地址（Home ID）。网络内每个节点的地址（Node ID）都由控制节点（Controller）分配。每个网络最多可容纳 232 个节点（Slave），包括控制节点在内。控制节点可以有多个，只有一个主控制节点，即所有网络内节点的分配都由主控制节点负责，其他控制节点只是转发主控制节点的命令，已入网的普通节点（非控制节点）可以被有的控制节点控制。超出通信距离的节点可以通过控制器与受控节点之间的其他节点通过路由（Routing）的方式完成控制。

Z-Wave 协议由下至上分为 5 层：物理层、媒体访问控制层（MAC）、传输层、路由层和应用层。媒体访问控制层负责设备间无线数据链路的建立、维护和结束，同时控制信道的接入进行帧校验，并预留时隙管理。为了提高数据传输的可靠性，当有节点进行数据传送时，媒体访问控制层还采用载波侦听多址/冲突避免（CSMA/CA）机制防止其他节点传送信号。传输层用于提供节点之间可靠的数据传输，主要功能包括重新传输、帧校验、帧确认及实现流量控制等。路由层控制节点间数据帧的路由，确保数据帧在不同节点间能够多次重复

传输、扫描网络拓扑及维持路由表等。应用层负责 Z-Wave 网络中的译码和指令的执行，主要功能包括曼彻斯特译码、指令识别、分配 Home ID 和 Node ID、实现网络中控制器的复制及对于传送和接收帧的有效荷载进行控制等。

① 物理层。

Z-Wave 是一种低速率的无线技术，专注于低速率的应用，有 9.6kb/s 和 40kb/s 两种传输速率：前者用来传输控制命令；后者可以提供更为高级的网络安全机制。Z-Wave 的工作频段处于 900MHz（ISM 频带）、868.42MHz（欧洲）和 908.42MHz（美国）。相对于 ZigBee 或蓝牙所使用的日益拥挤的 2.4GHz 频带，这些频带上的设备相对较少，更能保证通信的可靠性。

Z-Wave 的功耗极低，使用频移键控（Frequency-Shift Keying，FSK）无线通信方式，适合智能家居网络，电池供电节点通常保持在睡眠状态，每隔一段时间唤醒一次，监听是否有需要接收的数据，两节普通 7 号电池可以使用长达 10 年的时间，免去了频繁充电和更换电池的麻烦，保证了应用的长久稳定。

Z-Wave 的系统复杂程度比 ZigBee 低。其设备比蓝牙设备要小得多，协议简单，所要求的存储空间很小。在标准的 Z-Wave 模块中设计了 32KB 的闪存用于存放协议，同等功能的 ZigBee 模块至少需要 128KB 的闪存才能适合使用，蓝牙模块则需要更大的闪存。所以，Z-Wave模块的成本要低于 ZigBee 或者蓝牙设备。

Z-Wave 的网络容量为单网络，最多 232 个节点，远低于 ZigBee 的 65535 个节点。Z-Wave节点的典型覆盖范围为室内 30m、室外 100m，最多支持 4 级路由。Z-Wave 在应用的普适性方面差于 ZigBee，不能使用单一技术建立大规模网络，对于智能家居应用来说足以覆盖全部范围。通过使用虚拟节点技术，Z-Wave 网络也可以与其他类型的网络进行通信。

② 媒体访问控制层（MAC）。

媒体访问控制层（MAC）的设计主要考虑尽可能做到低成本、易实现、数据传输可靠、短距离操作及低功耗。

媒体访问控制层（MAC）主要用于控制射频媒介，数据流采用曼彻斯特编码模式，除了在接收帧数据或二进制曼彻斯特编码/解码比特流外，基本不受射频媒介、频率和调制方式的影响，数据以 8 位数据块的结构进行传输。其中，从最重要的比特起始，主要数据以曼彻斯特编码的形式存在，以方便获得直流自由信号。

在传送时，只有数据帧传送给传输层，并且以低字节在前的格式或反字节的格式进行传送。另外，为了提高数据传输的可靠性，当有节点进行数据传送时，媒体访问控制层（MAC）采用载波侦听多址/冲突避免（CSMA/CA）机制防止其他节点开始传送。

③ 传输层。

传输层主要用于提供节点之间可靠的数据传输，主要功能包括重新传输、帧校验、帧确认和实现流量控制等。传输层帧共有三种类型。

a. 单播帧。单播帧向一个指定的节点发送，如果目标节点成功收到此帧，将会回复一个应答帧 ACK；如果单播帧或者应答帧丢失或损坏，则单播帧将被重发。为了避免与其他系统的碰撞，重发帧将会有一个随机延迟。随机延迟必须与传输最大帧长和接收应答帧所花费的时间一致。单播帧在不需要可靠传输的系统中可以选择关闭应答机制。应答帧是Z-Wave单播帧的一种类型，其数据域的长度为0。

b. 多播帧。多播帧将传输给网络中的节点1到节点232中的若干个。多播帧的目标地址指定了所有的目标节点，而不用向每个节点发送一个独立的帧。多播帧没有应答，不能用在需要可靠传输的系统中。如果多播帧一定要求可靠性，则需要在多播帧之后跟着发送单播帧。

c. 广播帧。广播帧将传输给网络中的所有节点，任何节点都不对该帧进行应答。与多播帧一样，广播帧也不能用于需要可靠传输的系统中。如果广播帧一定要求可靠性，则需要在广播帧之后跟着发送单播帧。

④ 路由层。

路由层的主要功能包括控制节点间数据帧的路由、确保数据帧在不同节点间能多次重复传输、扫描网络拓扑及维持路由表等。

控制器和节点都参与帧的路由。它们总是处在监听状态并且有一个固定的位置。路由层负责通过一个正确的转发表来发送帧，同时也保证帧在节点与节点之间的转发。路由层也要扫描网络拓扑结构并且维护控制器中的路由表。

路由层采用动态源路由（Dynamic Source Routing，DSR）协议。DSR协议是一种按需路由协议，允许节点动态发现到达目标节点的路由。每个数据帧的头部都附加到达目标节点之前所需经过的节点列表，即数据分组中包含到达目标节点的完整路由。与传统的路由方法不同，传统的路由方法，如按需距离矢量（Ad Hoc On-demand Distance Vector，AODV）路由协议在分组中只包含下一个跳转的节点和目的节点地址，而DSR不需要周期性的网络拓扑信息，可避免网络的大规模更新，能有效减少网络带宽的开销，节约能量的消耗。

在发现路由时，源节点发送一个含有源路由列表的路由请求帧。此时，路由列表只有源节点，收到请求帧的节点继续向前发送，并在路由列表中加入自己的节点地址，直至到达目标节点。每个节点都有一个用于保存最近收到路由请求帧的存储区，可以不重复转发已经收到的请求帧。部分节点（如果它们有额外的外部存储空间）会将已经获得的源路由表存储下来以减少路由的开销。当收到请求帧时，先查看存储的路由表中是否存在合适的路由，如果有，就不再转发，直接返回该路由至源节点；如果请求帧被转发到了目标节点，那么目标节点就将返回一个返回路由。

当源节点需要与目标节点通信时，源节点首先广播一个具有唯一ID的RREQ（RouteRequest，路由请求）消息，可被源节点无线覆盖范围内的一个或多个具有到目标节点路由信息的中间节点接收，返回该路由信息至源节点。每个节点的路由缓冲区都会记录该节点侦听到的路由信息。当一个节点收到RREQ消息时，如果在该节点最近的请求帧中包含

该请求帧，则会丢弃该请求帧；如果 RREQ 路由记录中包含当前节点的地址，则不进行处理，可防止形成环路；如果当前节点就是目标节点，则发送返回路由给源节点；在其他情况下，该节点在 RREQ 中添加自己的地址，并将该请求帧广播出去。

当路由列表上的一个节点移动或掉电时，网络拓扑会发生变化，路由不可用。当上游节点通过 MAC 协议发现连接不可用时，就会向上游的所有节点发送 RERR（RouteError，路由错误）。源节点收到 RERR 后，会从路由存储区中删除无效路由。如果需要，则源节点会重新发起路由发现过程来建立新路由。

DSR 协议不需要进行周期性的交换路由信息，可以减少网络开销，节点可以进入休眠模式，节省电池电量。数据帧中含有完整的路由信息，节点可以获取完整路由中所包含的部分有用信息，如在 A 节点到 B 节点到 C 节点的路由中包含了 B 节点到 C 节点的路由信息，B 节点不需要发起对 C 节点的路由发现，从而节省了路由发现所需的开销。同时，DSR 协议网络的规模受到了限制，因为数据包中有很多都带有路由信息，所以过长的路由表会大幅增加网络分组的开销。鉴于一个 Z-Wave 网络中最多 232 个节点的限定和最多支持 4 条路由，DSR 协议的额外开销不至于十分严重，增强型节点类型也有更大的外部存储空间，可以存储最近使用的路由信息，以硬件开销弥补网络的性能。

⑤ 应用层。

应用层负责 Z-Wave 网络中的译码和指令的执行，主要功能包括曼彻斯特译码、指令识别、分配 Home ID 和 Node ID、实现网络中控制器的复制及对于传送和接收帧的有效荷载进行控制等。Z-Wave 技术关注设备的互操作性和厂商开发的方便性，在应用层中引入相关机制实现这些特性。

为了实现智能家居控制系统中众多子系统的相互作用，加强各个领域厂商产品的相互操作性，Z-Wave 提供标准化的方法来实现设备和设备之间的相互作用。例如，允许从某一个厂商制造的遥控器中提供的照明子系统调光功能控制另一个厂商制造的灯光节点。这样，所有的厂商只需要集中精力开发所擅长的产品，所有的产品都可以很好地工作在一个 Z-Wave 网络中，不需要自己生产整套智能家居系统，给厂商的开发提供了便捷。

(2) Z-Wave 网络的基本配置

Z-Wave 网络的配置简单易行，各种符合标准的家庭设备都能方便地“安装”在家庭网络中，也能方便地从家庭网络中“卸载”。Z-Wave 设备在安装过程中的最主要问题是实现该设备的网络接入识别。Z-Wave 在家庭网络中定义 3 种类型的设备：控制器（controller）、路由从设备（routingslave）及从设备（slave）。当需要安装新的节点设备时，首先需要激活网络中的控制器和其他所有的节点，激活可以同步也可以不同步。控制器被第一次激活后，可以通过广播查询新节点的请求，如果收到新节点的回应，则控制器会给新节点分配一个 ID，通过这个 ID 规定新节点与控制器之间的主从关系。新节点需要向控制器报告它周围的邻居表，即在射频范围内的所有节点，使控制器有全面的网络拓扑信息，从而建立一个无缝的网状结构通信网络。

① 控制器。

控制器也是Z-Wave网络中的节点，主要负责产生并发出控制命令给其他节点。从设备主要接收并执行控制节点发出的指令。在控制节点无法与目标节点直接联系的情况下，从设备还可以负责中继传输控制节点发出的指令。

Z-Wave网络中的控制器存储了完整的路由表，可以与Z-Wave网络中的所有节点进行通信，只要控制器进入Z-Wave网络的覆盖范围便可以发挥控制功能。如果控制器产生一个新的Z-Wave网络，则会自动成为新网络的主控制器。一个Z-Wave网络中只能有一个主控制器。只有主控制器可以实现在网络中管理控制节点的功能，并通过这个功能实时改变网络的拓扑结构。控制器可分为动态控制器、静态控制器、配置控制器及桥控制器。

a. 动态控制器是Z-Wave网络中可以自由改变位置的控制器，可以采用一系列的方法进行实时定位，并依据定位信息判定最快的网络路由方式。动态控制器由于处于主动工作模式，因此耗电量比较大，需要定期更换电池。

b. 静态控制器的最大好处是可以实时接收路由节点的状态信息，根据这些信息判定各节点在网络中的位置。因此，静态控制器通常作为Z-Wave网络中的从控制器。静态控制器还可以作为因特网的网关，实现Z-Wave网络的远程实时监控。静态控制器主要包括SUC（Static Update Controller）和SIS（SUC ID Server）。SUC是Z-Wave网络可以选择性使用的静态控制器，用于实时更新网络的拓扑结构，能够不断地接收来自网络主控制器的网络变化信息，并将这些信息传输给其他的控制器。在Z-Wave网络中只能配置一个SUC。SIS也是Z-Wave网络可以选择性使用的具有节点ID服务功能的SUC，是网络中的主控制器，可以使其他的控制器根据自身的需要调整网络节点。当在SIS控制的Z-Wave网络中有新的控制器加入时，这些新的控制器将成为网络中的从控制器，需要根据SIS的指令调整网络节点。

c. 配置控制器是一种具备特殊功能的移动控制器，是用户安装Z-Wave网络的配置工具，具有比其他控制器更完善的网络管理功能，可以对网络的服务质量进行测试。

d. 桥控制器是Z-Wave网络选择性配置的静态控制器，具有网络拓展的功能，可以实现Z-Wave网络与其他网络的连接。桥控制器可以存储Z-Wave网络的节点信息，并可以控制多达128个虚拟从设备（虚拟从设备是指处于其他类型网络中的节点）。

② 从设备。

从设备基本不包含网络拓扑信息，也不具备Z-Wave网络节点的管理功能。从设备主要用来接收指令并完成指令赋予的任务。在没有获得指令的情况下，从设备不可以向其他节点发送路由信息。从设备在环状网中路由需要稳定的工作电源，保证时刻接收来自网络中其他节点的指令。

③ 路由从设备。

路由从设备具有从设备所有的功能。此外，它还可以主动向网络中的其他节点发送路由信息。当路由从设备主动向一部分节点发送信息时，便会存储这些静态的路由信息以供使用。路由从设备可以使用市电或电池作为电源，在获得稳定电源保障的情况下，可以在

Z-Wave网络中发挥路由器的功能。路由从设备在使用电池作为电源时，还可以处于一种特殊的工作模式——FLiRS（Frequently Listening Routing Slave），在睡眠状态下也可以接收来自其他节点的信息，并能被其他节点激活。

④ 增强型从设备。

增强型从设备的功能与路由从设备基本一致，只是增强型从设备配置了一个 EEPROM，用来存储应用数据。

（3）Z-Wave 技术的特点

Z-Wave 作为新兴的无线个域网技术，与其他无线个域网技术相比有非常鲜明的技术特征。

① 功耗低。

Z-Wave 与许多其他控制系统不同，采用轻量协议和压缩帧格式实现低功耗。此外，Zensys 采用 Z-Wave 单个模块方案及自适应发射功率模式有利于电池驱动设备（如调温器、传感器等）的节能，也有利于降低家居控制系统的功耗。

② 网络管理便捷。

在安装智能化网络时，Z-Wave 技术便于实现地址分配，同时还可以实现节点间的互联，只需几分钟便可安装组网，操作简单、直接。符合标准的各种家庭设施可以方便地“安装”在家庭网络中，也可以方便地从家庭网络中“卸载”。Z-Wave 用户对 Z-Wave 网络的管理将会非常轻松。

③ 抗干扰能力强。

Z-Wave 使用的是免授权通信频带，采用双向应答式的传送机制、压缩帧格式及随机式的逆演算法来减少干扰和失真。同时，每个 Z-Wave 网络都有自身独特的网络标识符，可以防止邻近网络的控制或干扰。

4. 蓝牙技术

蓝牙（Bluetooth）技术实际上是一种短距离无线通信技术，能够有效简化掌上电脑、笔记本电脑及手机等移动通信终端设备之间的通信，也能够成功简化这些设备与因特网的通信，使这些现代通信设备与因特网的数据传输变得更加迅速高效。其功耗及成本介于 WiFi 与 ZigBee 之间。限于传输距离，蓝牙技术仅适用于智能手环、耳机、蓝牙音箱等个人移动设备。若组建大型家庭网络，则需与其他技术配合使用。

蓝牙技术的载频为 2.4GHz，收发信机采用跳频扩谱技术，在 2.4GHz ISM 频带上以 1600 跳/s 的速率跳频。在发射带宽为 1MHz 时，蓝牙技术的有效数据传输速率为 721kb/s，采用低功率时分复用方式发射，适合 10m 距离范围内的通信。数据包在某个载频上的某个时隙内传递，不同类型的数据包（包括链路管理和控制消息）占用不同的信道，并通过查询和寻呼过程同步跳频频率和不同蓝牙设备的时钟。除采用跳频扩谱的低功率传输外，蓝牙技术还采用鉴权和加密等措施来提高通信的安全性。

蓝牙技术支持点到点和点到多点的连接，可采用无线方式将若干个蓝牙设备连接成一个微微网（Piconet），多个微微网又可互联为特殊分散网，形成灵活的多重微微网的拓扑结构，从而实现各类设备之间的快速通信。蓝牙技术能够在一个微微网内寻址8个设备（实际上，互联的设备数量是没有限制的，只不过在同一时刻只能激活8个。其中，1个为主，7个为从）。

（1）蓝牙协议

蓝牙协议的目标是允许遵循规范的应用能够进行相互间的操作。完整的蓝牙协议架构如图2.3所示。蓝牙协议可以分为4层，即核心协议层（BaseBand、LMP、L2CAP、SDP）、电缆替代协议层（RFCOMM）、电话控制协议层（TCS-Binary、AT命令集）及其他协议层（PPP、UDP/TCP/IP、OBEX、WAP、vCard、vCal、IrMC、WAE）。

除上述协议层外，蓝牙协议规范还定义了主机控制器接口（HCI），可以为基带控制器、连接管理器、硬件状态及控制寄存器提供命令接口。在图2.3中，HCI位于L2CAP的下层，也可以位于L2CAP的上层。

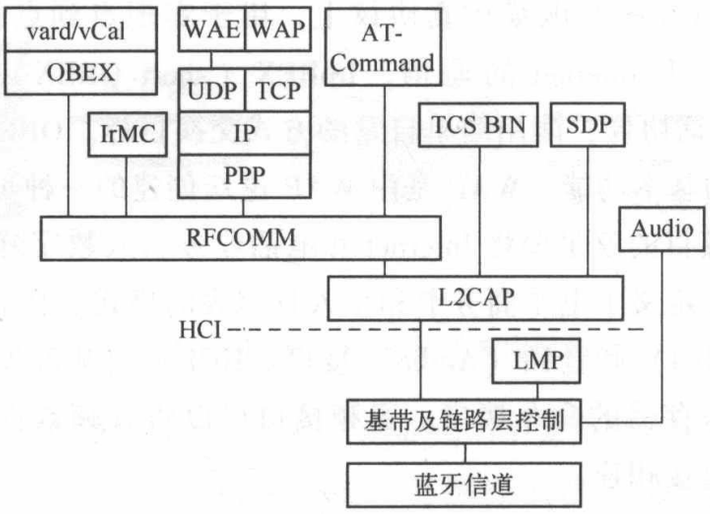


图2.3 完整的蓝牙协议架构

① 核心协议。

蓝牙协议的核心协议由基带、链路管理（LMP）、逻辑链路控制与适应协议（L2CAP）及业务搜寻协议（SDP）四部分组成。从应用的角度看，射频、基带和LMP可以归为蓝牙协议的低层协议，对应用而言是十分透明的。基带和LMP负责在蓝牙设备单元之间建立物理射频链路，构成微微网。此外，LMP还要完成身份验证和加密等安全方面的任务，包括生成和交换加密键、链路检查、基带数据包大小的控制、蓝牙无线设备的电源模式和时钟周期、微微网内蓝牙设备单元的连接状态等。逻辑链路控制与适应协议（L2CAP）完成基带与高层协议间的适配，并通过协议复用、分用及重组操作为高层提供数据业务和分类的提取，允许高层协议和应用接收或发送长达64K字节的L2CAP数据包。业务搜寻协议（SDP）是极其重要的部分，是所有使用模式的基础，通过SDP可以查询设备信息、业务及业务特征，并在查询之后，建立两个或多个蓝牙设备之间的连接。SDP支持3种查询方式，即按业

务类别搜寻、按业务属性搜寻及业务浏览（browsing）。

② 电缆替代协议（RFCOMM）。

RFCOMM 是基于 ETSI-07.10 规范的串行线仿真协议，在蓝牙基带协议上仿真 RS-232 控制和数据信号，为使用串行线传送机制的上层协议（如 OBEX）提供服务。

③ 电话控制协议。

电话控制协议包括电话控制规范二进制（TCSBIN）协议和一套电话控制命令（AT-Commands）。其中，TCSBIN 定义了在蓝牙设备之间建立话音和数据呼叫所需的呼叫控制信令；AT-Commands 是一套可在多使用模式下用于控制移动电话和调制解调器的命令，是由蓝牙技术特别兴趣小组在 ITU-TQ.931 的基础上开发而成的。

④ 采纳的其他协议。

电缆替代层、电话控制层及被采纳的其他协议层可归为应用专用（application-specific）协议。在蓝牙技术中，应用专用协议可以加在串行电缆仿真协议之上或直接加在 L2CAP 之上。被采纳的其他协议有 PPP、UDP/TCP/IP、OBEX、WAP、WAE、vCard、vCalendar 等。PPP 运行在串行电缆仿真协议上，用于实现点到点的连接。UDP/TCP/IP 由 IETF 定义，主要用于 Internet 的通信。IroBEX（short OBEX）是由红外数据协会（IrDA）开发的一个会话协议，能用简单自发的方式交换目标。OBEX 采用客户/服务器模式提供与 HTTP 相同的基本功能。WAP 是由 WAP 论坛创建的一种工作在各种广域无线网上的无线协议规范。其目的就是要将 Internet 和电话业务引入数字蜂窝电话和其他无线终端。vCard 和 vCalendar 定义了电子商务卡和个人日程表的格式。在蓝牙技术协议栈中还有一个主机控制接口（HCI）和音频（Audio）接口。HCI 是到基带控制器、链路管理器及访问硬件状态和控制寄存器的命令接口。音频接口可以将音频数据在一个或多个蓝牙设备之间传递，与基带直接相连。

（2）蓝牙技术的主要特点

相对于其他的短距离传输方式，蓝牙技术除具有适用于全球范围、功耗低、成本低、抗干扰能力强等特点外，还有许多自身的特点：

① 同时可传输话音和数据。蓝牙技术采用电路交换和分组交换技术，支持异步数据信道、三路话音信道及异步数据与同步话音同时传输的信道。

② 可以建立临时性的对等连接（Ad hoc Connection）。

③ 开放的接口标准。为了推广蓝牙技术的使用，蓝牙技术联盟将蓝牙技术的标准全部公开，在全世界范围内的任何单位和个人都可以进行蓝牙产品的开发，只要最终通过蓝牙技术联盟的蓝牙产品兼容性测试，就可以推向市场。

总之，业界赞成 WiFi 应用的呼声很高；ZigBee 良好的先天条件却让人欲罢不能；小小的蓝牙也不乏应用场景。从目前来看，对任一种技术盲目乐观或悲观均为时过早。例如，WiFi 在安全性方面有所欠缺，基础设备及网络覆盖面积普及最广，是 ZigBee 可望不可及的。随着物联网及智能家居产业的成熟，多样化的市场需求将逐渐显现，届时各种

技术都能在市场上找到自己的位置，尺有所短，寸有所长，无论以何种形式出现，最终融合才是大方向。

2.2.3 HTTP 协议

因特网使用的基本协议是 TCP/IP 协议，在 TCP/IP 协议模型最上层的是应用层。应用层包含所有的高层协议。高层协议有文件传输协议 FTP、电子邮件传输协议 SMTP、域名系统服务 DNS、网络新闻传输协议 NNTP 和 HTTP 协议等。

HTTP（Hypertext Transfer Protocol）协议，即超文本传输协议，是用于 WWW 服务器与本地浏览器进行超文本传输的协议，可以使浏览器与 Web 服务器通过互联网进行更加高效的传输网络信息，并使其标准化。HTTP 协议不仅可以保证计算机能够正确快速地传输超文本信息，还能够确定传输文档中的哪一部分及哪一部分内容首先显示（如文本先于图形）等。图 2.4 为 HTTP 协议的请求响应模型。

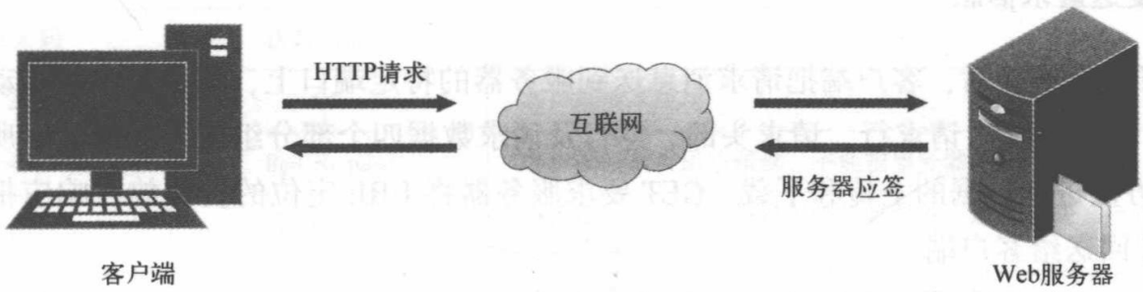


图 2.4 HTTP 协议的请求响应模型

由于 HTTP 协议是基于请求应答范式的，因此当一个客户机与服务器建立连接后，将发送一个请求给服务器，请求方式的格式为统一资源标识符（URL），协议版本号后面的 MIME 信息包括请求修饰符、客户机信息等内容。服务器在接收到请求后，将给予相应的响应信息。其格式为一个状态行（信息的协议版本号、一个成功或错误的代码），后边是 MIME 信息，包括服务器信息、实体信息及其他一些内容。

许多 HTTP 通信是由一个用户代理初始化的，并且包括一个申请在源服务器上的资源请求。最简单的情况可能是在用户代理和服务器之间通过一个单独的连接来完成。在因特网上，HTTP 通信经常发生在 TCP/IP 连接上。其默认端口是 TCP 协议的 80 端口，当然，其他端口也是可以使用的，但这并不预示着 HTTP 协议在因特网或其他网络的其他协议上才能完成，只预示一个可靠的传输。

HTTP 协议的内部操作过程大致为：在 WWW 中，“客户”与“服务器”是一个相对的概念，只存在于一个特定的连接期间，即某个计算机在某个连接中可能是客户机，而在另一个连接中可能被作为服务器。基于 HTTP 协议客户/服务器模式的信息交换过程可以分为四步：建立连接、发送请求信息、发送响应信息及关闭连接。图 2.5 为 HTTP 协议客户/服务器模式的信息交换过程。

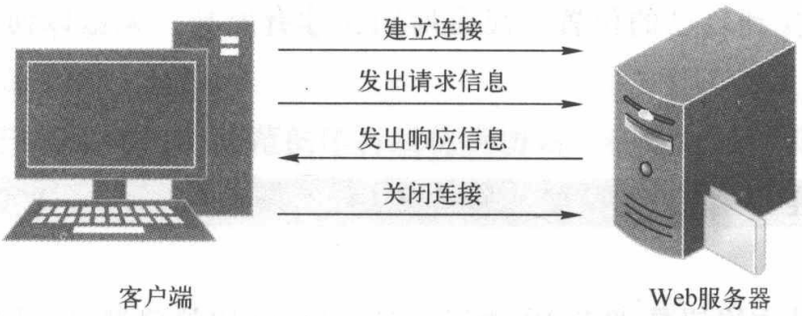


图 2.5　HTTP 协议客户/服务器模式的信息交换过程

1. 建立连接

连接的建立是通过申请套接字（Socket）实现的。客户打开一个套接字，并将套接字绑定在一个端口上，如果成功，就相当于建立一个虚拟文件，以后就可以在该虚拟文件上写数据并通过网络向外传送。

2. 发送请求信息

打开一个连接后，客户端把请求消息送到服务器的特定端口上，完成提出请求动作。一个 HTTP 请求报文由请求行、请求头部、空行及请求数据四个部分组成，如图 2.6 所示，采用 GET 方法进行数据的上传和下载。GET 要求服务器将 URL 定位的资源放在响应报文的数据部分并回送给客户端。

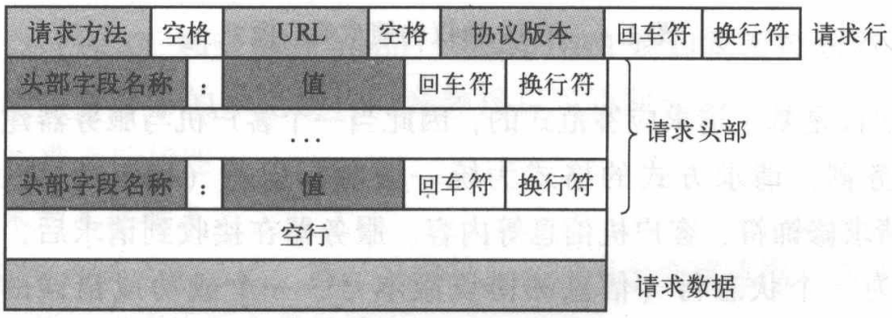


图 2.6　HTTP 请求报文格式

请求行由请求方法字段、URL 字段和 HTTP 协议版本字段组成。

请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端的请求信息，典型的请求头有 Host：请求的主机名，如本书中用到的 Host：139. 129. 9. 166。

最后一个请求头部后是空行，通知服务器下面不再有请求头部。另外，请求数据不在 GET 方法中使用，而是在 POST 方法中使用

3. 发送响应

在接收和解释请求消息后，服务器会返回一个 HTTP 响应消息。HTTP 响应由三部分构

成：状态行、响应头部及响应正文，如图 2.7 所示。

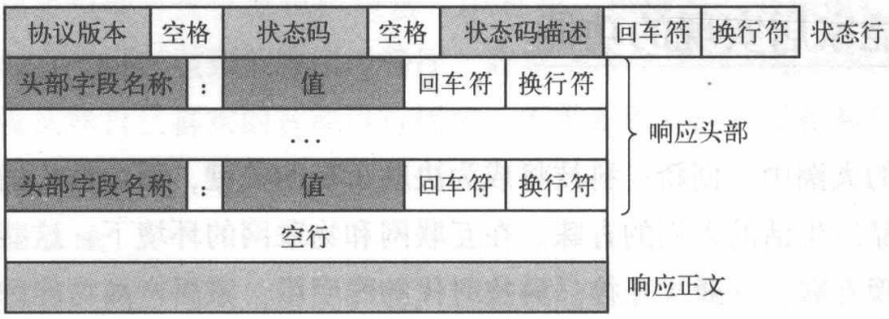


图 2.7 HTTP 响应报文格式

状态行：由协议版本、数字形式的状态码、相应的状态码描述组成，各元素之间以空格分隔，如 HTTP/1.1 200 OK\r\n。状态码描述给出了关于状态码的简短文本描述。状态码描述文本的取值说明见表 2.1。

表 2.1 状态码描述文本的取值说明

状态码	状态码描述	说明
200	OK	客户端请求成功
400	Bad Request	客户端请求有语法错误，不能被服务器理解
403	Forbidden	服务器收到请求，但是拒绝提供服务，通常会在响应正文中给出不提供服务的原因
404	Not Found	请求的资源不存在，如输入了错误的 URL

响应头部包括如下：

Server：Server 响应报头域包含服务器处理请求的软件信息，与 User-Agent 请求报头域是相对应的。前者发送服务器端软件的信息；后者发送客户端软件（浏览器）和操作系统的信息。例如，Server：Apache-Coyote/1.1。

Content-Length：Content-Length 实体报头域用于指明正文的长度，用字节方式存储的十进制数字来表示，也就是一个数字字符占一个字节，用其对应的 ASCII 码存储传输。这个长度仅仅是表示实体正文的长度，没有包括实体报头的长度。

Content-Type：Content-Type 实体报头域用于指明发送给接收者实体正文的媒体类型。例如，Content-Type：text/html;charset=GB2312。

空行：最后一个响应头部后是一个空行，发送回车符和换行符，通知服务器以下不再有响应头部。

响应包体：服务器返回给客户端的文本信息。

4. 关闭连接

客户端和服务器都可以通过关闭套接字来结束 TCP/IP 对话。

2.3 智能家居实现的功能

在智能家居的大潮中，创新、科技将成为决胜市场的关键，而个性化的定制类家居产品将越来越受追求品质生活的人们的青睐。在互联网和物联网的环境下，这些特色智能终端通过系统集成可实现互联、互通、互控。科技时代的聪明屋，家居产品功能的强大，只有让你想不到，没有做不到。除了已经进入千家万户的扫地机器人、智能马桶盖等终端产品，当下最流行的智能家居产品还有指纹锁、智能衣橱……

智能家居产品的一大特征是品类丰富，实现的功能主要有以下几类。

(1) 智能灯光控制

智能家居产品可以实现对全家居灯光的智能管理，通过遥控等多种智能控制方式实现对全家居灯光的遥控开/关、调光、全开/全关及“会客、影院”等多种一键式灯光场景效果的实现，并可用定时控制、本地及远程控制等实现功能，从而达到智能照明的节能、环保、舒适及方便。

(2) 智能电器控制

智能电器控制采用弱电控制强电的方式，既安全又智能，可以用遥控、定时等多种智能控制方式实现对饮水机、插座、空调器、地暖、投影机、新风系统等的智能控制：避免饮水机在夜晚反复加热影响水质；在外出时可断开插排通电，避免电器发热引发安全隐患；对空调器、地暖进行定时或者远程控制，让用户到家后马上就可以享受舒适的温度和新鲜的空气。

(3) 安防监控系统

随着人们居住环境的升级，人们越来越重视自己的个人安全和财产安全，对住宅小区的安全提出了更高的要求，人防保安方式难以适应要求，智能安防便成为当前安防监控的发展趋势。

虽然视频监控系统已经广泛应用于银行、商场、车站及交通路口等公共场所，但实际的监控任务仍需要较多的人工来完成，而且现有的视频监控系统通常只是录制视频图像，提供的信息是没有经过解释的视频图像，只能用作事后取证，不能充分发挥监控的实时性和主动性。为了能实时分析、跟踪、判别监控对象，并在异常事件发生时提示、上报，为政府部门、安全领域的及时决策、正确行动提供支持，视频监控的“智能化”就显得尤为重要。

(4) 智能背景音乐

智能背景音乐是在公共背景音乐基本原理的基础上，结合家庭生活的特点发展而来的新型背景音乐系统，简单地说，就是在家庭的任何一个房间，如花园、客厅、卧室、酒吧、厨房或卫生间，都可以将 FM、DVD、电脑等多种音源进行系统组合，让每个房间都能听到美妙的背景音乐，音乐系统既可以美化空间，又可以起到很好的装饰作用。

(5) 智能视频共享

智能视频共享系统是将数字电视机顶盒、DVD 机、摄像机、卫星接收机等视频设备集中安装在隐蔽的地方，可以做到让客厅、餐厅、卧室等多个房间的电视机共享家庭影音库，并可以通过遥控器选择自己喜欢的音源进行播放，不需要重复购买设备和布线，既节省了资金又节约了空间。

(6) 可视对讲系统

可视对讲系统的成熟案例随处可见，有大型联网的对讲系统，也有单独的对讲系统，可以实现的功能一般是呼叫、可视、对讲等。

(7) 网络远程控制

在办公室、外地出差，只要是有网络的地方，用户都可以通过因特网登录到家中，提供一个免费动态域名，在网络世界中就可以通过一个固定的智能家居控制界面控制家中的智能电器，主要用于远程网络控制和电器工作状态的信息查询。例如，用户出差在外地，利用外地的网络计算机，登录相关的 IP 地址，就可以控制远在千里之外的自家的灯光、电器，在返回上飞机之前，可以将家中的空调器或热水器打开……

(8) 系统手机控制

家中的灯光、电器都能使用手机进行远程控制，只要手机能够连接网络，就能使用随身的手机通过网络远程控制家里的灯光、电器及其他的用电设备。

2.4 智能家居技术架构

与传统的普通家居系统相比，智能家居系统能够为用户提供更多的功能，具有许多新的特性，主要体现在以下 3 个方面。

首先，普通家居系统是一个孤立的系统，家居内部的各子系统之间相互独立，彼此没有联系，不能形成一个有机的整体；家居与家居以外的社会缺乏足够的联系，是信息海洋中的“孤岛”。智能家居系统就是要打破家居内部及家居与社会之间的孤立，使家居系统成为一个充满联系的、有机的整体，成为信息社会中的重要组成部分。

其次，普通家居系统是一个静态的系统，要想实现家居的智能化，必须将静态的系统变为动态的系统。例如，用户出门时经常会忘记关窗户，每逢刮风下雨便担心不已，如果窗户能够动态感应到天气的变化，刮风下雨时可以自动关闭，那么用户就不会再有这样的担心了。智能家居系统中的智能窗户不但可以完成这样的功能，还具有防火、防盗、防烟雾等功能。当建筑物不慎发生火灾时，智能窗户会自动打开，方便室内人员紧急逃生，并启动报警装置，及时报警。当室内的煤气、天然气等可燃性气体或烟雾的浓度达到一定值时，智能窗户就会自动打开，并启动排风扇自动换气，让有毒的气体散发到窗户外，可以有效防止中毒和火灾事故的发生，确保室内的空气新鲜，用户身体不会受到伤害。当室外的噪声影响用户

的工作或休息时，智能窗户能够根据事先设定的室外噪声分贝值自动关闭。

最后，普通家居系统是一个被动的系统，智能家居系统是一个能动的系统。智能家居系统不只可以被动地接收控制指令，同时还会根据环境的变化和用户的习惯及喜好自动做出相应的调整。

智能家居系统的核心和特点是数字化、网络化、智能化。数字化是前提，网络化是基础，智能化是目标。智能家居系统由无线传感器模块、家庭网关、服务器、云平台及终端应用等组成。

在智能家居系统中，家居内部将会出现大量联网的设备、终端。这些设备虽然功能和形态各异，但要联入家居网络，数字化将是其共同的发展趋势。随着嵌入式技术的发展，每个联网设备都是一个拥有计算、存储及通信功能的智能节点。家居联网设备的结构如图 2.8 所示。其中，控制单元由处理器单元和存储单元组成；通信单元负责节点数据的收、发并联入家居网络；功能单元根据设备所完成的功能可以是传感器、采集单元或执行单元。根据节点所完成功能的复杂性，控制单元本身可能就是一个包含嵌入式处理器和嵌入式操作系统的嵌入式系统。

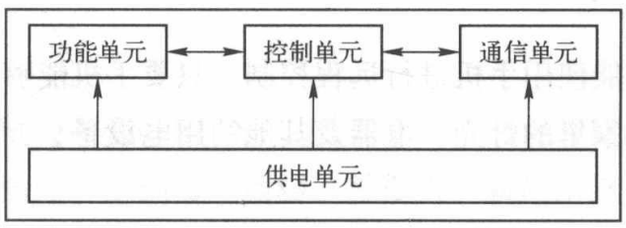


图 2.8　家居联网设备的结构

智能家居系统是一个复杂的系统，具体表现在如下几个方面。

(1) 智能家居系统组成的复杂性

智能家居系统涉及的子系统有家居安防、可视对讲、智能照明、家居影院、三表自动抄送、小区物业管理、信息家电、远程教育及远程医疗等。

(2) 家居内部组网方式多样

家居内部网络所采用的传输介质分为有线和无线两大类。有线传输介质有电话线、电力线、双绞线、同轴电缆等。其中，电话线和电力线可以利用家居已有的布线。无线传输介质有红外、微波等。

家居内部组网可用的网络协议很多，如 X-10 协议、以太网协议、Lonworks 控制网络技术所采用的 LongTalk 协议、RS485 协议、IEEE 802.11 协议、无线局域网协议、蓝牙协议等。

(3) 互联网设备差别大

接入家居局域网的设备主要包括家用电器、家庭仪表、桌面电脑、手提设备、信息娱乐设备、成像设备、智能传感器、控制器等。其中，有些设备具有丰富的软、硬件资源，功能强大，如家庭电脑；而有些设备的资源非常有限，功能单一，如各种传感器等。

(4) 家居宽带接入方式多样

目前，家居宽带接入技术有基于普通电话线的 DSL（Digital Subscriber Line，数字用户环线）技术、基于有线电视网的 HFC（Hybrid Fiber Coax，混合光纤同轴电缆）技术、基于纯光纤网的 FTTH（Fiber To The Home）技术、利用无线电超高频波段的 LMDS（Local Multipoint Distribute Service，本地多点分配业务）技术、基于宽带 IP 的以太网接入技术及采用电力线通信的 PLC（Power Line Communication）技术。

(5) 家居网络是动态变化的网络

因为有各种便携式设备和移动终端的存在，如笔记本电脑、PDA、无线遥控器等，所以对于它们的加入和退出，家居网络系统应该能够自动完成检测和动态调整配置。另外，家居中的设备摆放位置可能会发生变化，这对于综合布线系统来说将会遇到很大的困难，但对于无线系统来说则会使系统变得简单、灵活，同时也会给用户带来很大的方便。

(6) 智能家居系统的控制和管理模式多样

智能家居系统的控制和管理模式分为两类：无线网的独立控制和管理模式、基于家居网络的控制和管理模式。基于家居网络的控制和管理模式又有集中式和分布式两类。

对于这样的一个复杂系统，通常的处理方法是首先将其分解，分解为若干个容易处理的子系统，然后采用“分而治之”的办法逐个解决，最后进行整个系统的集成。分解可以是横向分解，也可以是纵向分解。横向分解就是按照产品和功能将整个系统划分为多个模块；纵向分解则是按照系统自身的层次结构进行划分，建立系统的层次模型。

智能家居系统的纵向分层体系结构如图 2.9 所示。网络互联层主要解决接听内部各种设备及各个子系统之间的互联互通问题，通过各种家居组网技术建立家居网络平台。家居网络是一种与家居环境相适应的异构网络，是在家居环境中利用各种介质将家用电器、数字设备和消费电子产品连接起来，通过各种网络技术和控制技术实现信息传输和资源共享的一种分布式异构网络。家居网络通过不同的传输介质和传输协议，为智能家居系统提供信息传输平台。

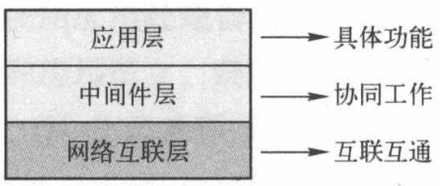


图 2.9 智能家居系统的纵向分层体系结构

中间件层主要用于解决系统的协同工作，包括即插即用和互操作性问题，通过家居网络中间件技术屏蔽底层的异构性建立一个统一的平台，构成系统整体集成的基础。家居网络中的中间件提供了一种简易的方法，可以实现家居分布式环境中的资源共享和协同工作，不但能降低系统的开发难度，提高开发效率，同时还对增强系统的稳定性和可扩展性具有很大的帮助。

应用层面向具体的应用，包括各种功能模块，可以提供人机接口并实现家居管理功能，为用户提供了方便、灵活及智能化的各种应用。

第3章 CC3200硬件平台

智能家居开发与实现的基本要求是能够使用通信接口控制外部设备和传感器实现数据的采集；能够对获取的数据进行整理、分析与处理；能够对相关的家庭设备进行控制；能够有较好的人机交互。随着嵌入式和微电子技术的飞速发展，嵌入式设备的功能越来越强大、性能越来越优良、成本越来越低廉，以嵌入式设备作为核心实现智能家居逐渐成为趋势。CC3200 LaunchPad是基于CC3200芯片的一款评估板。本书将以此评估板作为核心嵌入式设备实现智能家居的设计。

3.1 CC3200微控制器

CC3200是2014年由TI公司推出的一款单片无线微控制器，是业界第一个具有内置WiFi的MCU，是针对物联网应用、集成高性能ARM Cortex-M4的无线MCU。CC3200将WiFi的网络功能，包括协议栈、MAC地址层、数据链路层等单独使用网络处理器处理，将用户基于网络和外设的应用单独使用应用处理器处理，双处理器缓解了传统MCU对网络处理能力不足的问题，将网络处理和应用独立，避免了内存使用紧张的问题。用户能够使用单个CC3200开发整个应用。另外，TI公司提供了完整的软件开发工具包（SDK）来实现CC3200的快速开发，包括工具软件、示例应用程序、用户编程指南、参考设计及TI E2E支持社区。

CC3200采用易于布局的四方扁平无引线（QFN）封装，硬件组成如图3.1所示。

CC3200硬件组成分为三部分：应用MCU子系统、WiFi网络处理器（CC3100）子系统、电源管理子系统。

3.1.1 应用MCU子系统

应用MCU子系统包括一个运行频率为80MHz的工业标准ARM Cortex-M4内核、多种高

性能外设组件及相应的存储器部件。高性能外设组件包括快速并行摄像头接口、UART、SPI、I^2C、I^2S/PCM、SD/MMC、McASP 及一个四通道输入采集 ADC 等。存储部件包括可存储代码和数据的片内 RAM、存储外设驱动程序和引导加载程序的片内 ROM。

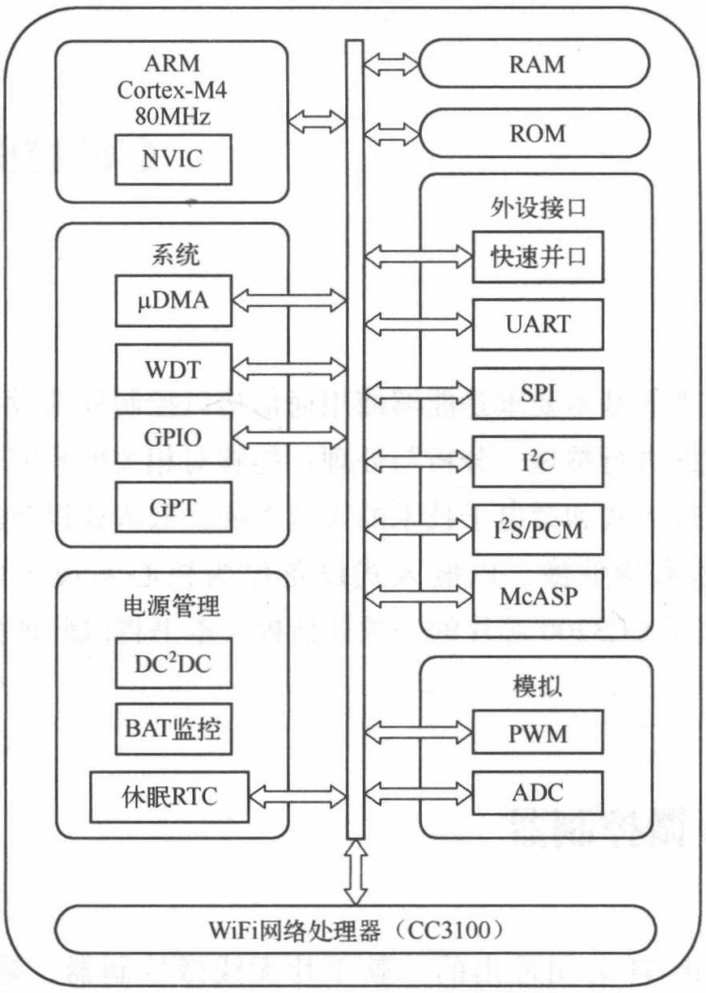


图 3.1　CC3200 硬件组成

1. ARM Cortex-M4

ARM Cortex-M4 是 ARM 公司设计的高性能低功耗处理器。在 ARM11 处理器之后，ARM 处理器的命名格式进行了调整，更改为 Cortex，并划分为 A、R、M 三个系列。M 系列是应用于低功耗嵌入式领域的处理器，是被广泛使用的主流处理器，拥有出色的计算能力和杰出的系统中断响应能力。高性能 ARM Cortex-M4 能够实现最小的内存需求，减少引脚数，降低功耗，并提供低成本的开发平台及卓越的计算性能和系统中断响应。Cortex-M4 处理器的结构如图 3.2 所示。

Cortex-M4 处理器的特点：

- 16/32 位的高优化度 Thumb II 指令集；
- 进出中断处理自动保存处理器状态；
- 支持处理模式和进程模式；
- 核内集成嵌套向量中断处理器（NVIC）；

- 中断支持可编程请求优先级和响应优先级；
- 支持休眠模式中断唤醒；
- 具有先进系统总线（AHB）和先进外围总线（APB）；
- 内核集成 JTAG 片上调试与仿真。

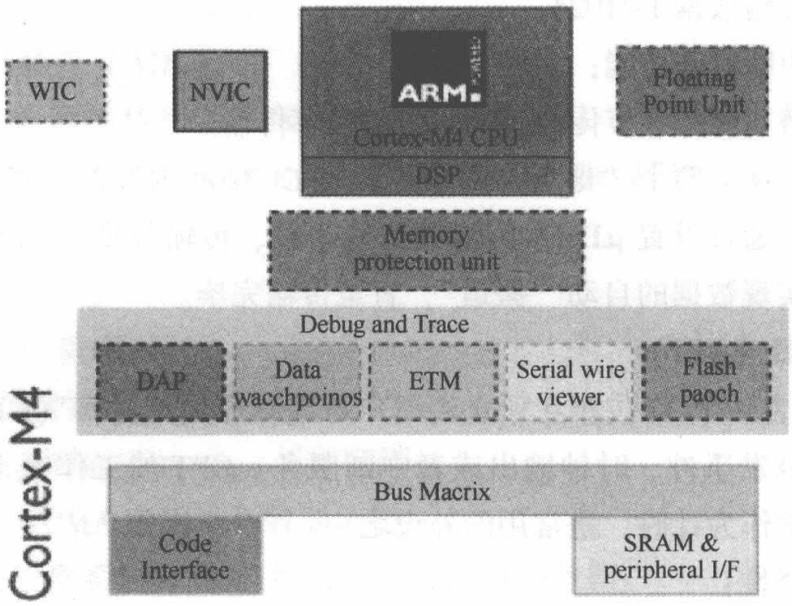


图 3.2　Cortex-M4 处理器的结构

在 ARM Cortex-M4 处理器中，可嵌套中断向量控制器（NVIC）和 SysTick 滴答定时器是两个重要部件，在应用开发时会经常用到。NVIC 可以处理所有的内核异常和中断，通过中断分组的方式，将一个字节分成任意两份作为主优先级组和次优先级组，从而提供多达 255 级的中断优先级控制，并且每个中断的优先级均可编程，使用非常灵活。中断之间可以通过优先级的高、低实现相互嵌套。SysTick 滴答定时器是一个简单的 24 位减计数器，可以为用户提供系统时钟，还会产生定时中断为内核服务。

2. 高性能外设组件

高性能外设组件包括 GPT、ADC、GPIO、WDT、μDMA 等实现基本的功能。例如，ADC 用来采集模拟量数据；GPT 用来为 CC3200 提供定时计数功能；快速并行摄像头接口、UART、SPI、I^2C、I^2S/PCM、McASP 等用来与外部其他器件通信。

（1）微型直接内存存取控制器（μDMA）

μDMA 是内部挂载至 AHB 总线上的多通道直接内存存取控制器，可以不经过内核操作完成数据的传输工作，还可以在存储器之间、存储器与外设之间进行正、反向的数据直接传输。

μDMA 也是一个可编程的控制器件，具有以下特性：

- 共有 32 个直接传输通道；
- 80MHz 的传输操作时钟；

- 支持存储器和外设之间的正、反向传输；
- 高度灵活且通道可编程配置；
- 包含两个优先级水平；
- 支持 8、16 和 32 位的数据宽度；
- 可编程设置传输数据 1~1024；
- 支持中断和中断屏蔽功能；
- 支持地址自增功能的字节传输、半字传输和字传输。

μDMA 一般用来搬运两个“地点”的数据，是 CC3200 内部的“搬运工”。“地点”可以是存储器和外设。通过设置 μDMA 传输数据的个数、传输数据的宽度、中断使能及传输方向等参数，即可实现数据的自动“搬运”，直至传输完毕。

（2）通用定时器（GPT）

CC3200 内部包含 4 个 32 位相互独立的用户可编程通用定时器（GPT）。GPT 可以通过计数或者定时产生触发事件、时钟输出或者中断服务。GPT 的工作频率稳定，计数间隔固定不变，计数也通常作为计时，是常用的外设之一。

GPT 具有如下特性：

- 可配置为 16/32 位的定时、计数模式；
- 可配置为周期运行或单次运行；
- 16 位 GPT 可使用 8 位的分频系数进行定时、计数分频；
- 16 位 GPT 具有输入边沿捕获、计数捕获和定时捕获功能；
- 16 位 GPT 可软件编程产生脉宽调制信号（PWM）；
- 能够编程为上升、下降计数；
- 可作为定时启动事件触发 μDMA 工作；
- 具有定时、计数中断；
- 可在定时器中断中产生突发请求。

（3）看门狗定时器（WDT）

当 CC3200 微控制器启动系统时，在出现错误但系统并没有做出预期响应而停顿的情况下，看门狗定时器就可以在系统运行超时的第一时间产生一个中断，在中断中发出一个复位信号，复位重启 CC3200。

WDT 可以防止系统无休止地进入停顿状态，在出现错误时及时复位，是一个 32 位的可复位、可编程定时长短的减计数定时器。其定时时间可以通过加载寄存器（Load register）来编程设定。WDT 可编程实现中断的屏蔽或开启。WDT 设置启动以后，锁寄存器（Lock Register）将会自动写保护来防止 WDT 的配置被无意中改变。

（4）通用输入/输出（GPIO）

CC3200 器件的所有数字引脚和部分模拟引脚都可以作为通用的输入或输出（GPIO）来使用，共有 32 个引脚可以作为 GPIO。32 个引脚被分为 4 组：PortA0、PortA1、PortA2 及

PortA3。每组中有 8 个引脚。

每个 GPIO 引脚根据寄存器的配置会有不同的功能。所有的 GPIO 引脚都能触发外部中断。外部中断可配置为 GPIO 引脚的电平高、低方式触发或上、下边缘方式触发，且外部中断也可编程屏蔽。GPIO 引脚可编程配置支持最小为 2mA、最大为 14mA 的驱动电流，同时还可编程配置为推挽模式和开漏模式。

(5) 模拟数字转换器 (ADC)

多数能够检测非电量的传感器一般都会将非电量转化为连续变化的模拟电压来表示。ADC 负责将连续变化的模拟电压转化为数字芯片可处理的离散数字量。

ADC 模块具有的特性如下：

- 12 位的采样分辨率；
- 10 位的有效采样精度；
- 共有 4 个模拟量的输入通道；
- 每个输入通道具有固定的 16μs 采样间隔；
- 采样数据可由 μDMA 通道进行数据传输；
- 每次采样都会根据系统时间自动记录 16 位的时间数据。

(6) 复通道音频接入接口 (McASP)

复通道音频接入接口是一种通用的音频接入接口，采用时分复用的数据流形式，使用非常灵活。CC3200 提供了一个可配置的 McASP，可将音频编解码与扬声器直接连接。McASP 包括两个能够独立发送和接收的串行器/解串器，且发送和接收能够工作在同步状态。

McASP 有两个立体声 I^2S 通道，可配置为一个通道用于接收，另一个通道用于发送；也可配置为两个通道均为发送。其时钟和帧同步的极性（上升沿或下降沿）、传输字长（位/字）及位时钟产生器的小数分频均可编程配置。

(7) 串行外设接口 (SPI)

串行外设接口 (SPI) 是一种高速、全双工、同步通信的接口，使用时需要四根线，分别为数据输入 (SDI)、数据输出 (SDO)、时钟 (SCLK)、片选 (CS)。SPI 较多工作在主从模式下，其传输线又可称为主输入从输出 (MISO) 和主输出从输入 (MOSI)。

SPI 接口具有以下特征：

- 可与多个 SPI 接口组成主从工作模式；
- 可工作在 3 线或 4 线模式；
- 支持全双工和半双工通信；
- 通信时钟可编程配置频率、极性及相位参数；
- 最高工作频率可达 20MHz；
- 片选使能可编程设置极性；
- 可在传输第一个字节数据后编程延时时间的长、短；
- 数据长度可编程为 8 位、16 位和 32 位；

- 支持 μDMA 通道传输数据；
- 可编程控制片选和外部时钟之间的时间长度。

（8）集成电路总线（I^2C）

集成电路总线（I^2C）使用串行数据线（SDA）和串行时钟线（SCL）提供双向的数据传输。I^2C 工作在同步半双工通信模式，通过仲裁设备地址的方式连接不同的 I^2C 设备，如传感器、串行存储器、图像传感器及音频编解码器等。CC3200 集成了一个 I^2C 接口，具有 8 位的地址，支持快速通信（通信速率为 400kb/s）和标准通信（通信速率为 100kb/s ）。

（9）通用异步收发器（UART）

通用异步收发器（UART）是一种基于 RS-232 串行通信标准的通信接口，可实现全双工异步串行通信。CC3200 共集成两个 UART 通信接口。UART 的应用较为广泛，具有以下特性：

- 通信波特率可编程修改，最高支持 3Mb/s；
- 收发器具有可编程设置参数的 FIFO 缓存器，并且具有中断；
- 接收和发送可独立触发中断，互不干扰；
- 通信格式完全可编程；
- 提供传输 UART 数据的 μDMA 通道。

本小节仅对应用 MCU 的外设进行了简单的介绍，具体的开发过程将在后面陆续补充。通信接口涉及通信协议内容，在开发中并不涉及具体的协议时序，有兴趣的读者可以查阅相关资料了解通信接口的通信协议时序，更多外设的详细介绍可参考 TI 公司网站提供的相关资料。

3. 存储器的部件

CC3200 应用 MCU 的存储器包括外部存储器和内部存储器。

（1）外部存储器

CC3200 在外部存储器（串行闪存 S-Flash）中保存特有的文件系统，包括服务包文件、系统文件、配置文件、证书文件、网页文件及用户文件等。用户可以使用格式化命令 API 分配文件系统的总容量。文件系统的起始地址固定为 S-Flash 的首地址。应用 MCU 不能直接访问 S-Flash，必须通过文件系统访问分配给文件系统的 S-Flash 区域。文件系统按照下载顺序管理存储文件 S-Flash 的内存块分配。这意味着，系统中特定文件的位置不固定。存储在 S-Flash 中的文件使用直观的文件名而不是文件标识。文件系统 API 使用纯文本，文件加密和解密用户不可见，加密文件只能通过文件系统进行访问。

文件系统中的所有文件类型可以支持多达 128 个文件，文件存储的最小单元为 4KB，带有安全保障和安全选项加密文件存储的最小单元为 8KB，最大文件为 16MB。

（2）内部存储器

内部存储器包括 RAM（静态存储器 SRAM）和 ROM。

① SRAM。

CC3200 采用容量为 256KB 的片上 SRAM 实现下载及运行应用程序。为了降低应用成本，应用程序的开发者必须共享 SRAM 的代码和数据。片上 SRAM 挂载至 ARM Cortex-M4 内核的 AHB 总线上，可使用 μDMA 通道实现 SRAM 与外部设备之间的数据直接传输。CC3200 的 SRAM 采用先进的 4 路交错结构，在 μDMA 和内核处理器同时访问 SRAM 时，几乎可以没有损失地实现数据交换。CC3200 的 ROM 拥有丰富的外设驱动程序，可以节省 SRAM 空间。

CC3200 提供多达 256KB 的零等待状态内部 SRAM，能够在低功耗深睡眠（LPDS）模式下有选择地保留。SRAM 在存储器映像中的偏移地址是 0x2000 0000。

② ROM。

CC3200 的 ROM 容量为 64KB，起始地址是 0x0000 0000，一般用来储存引导加载程序、芯片的初始化程序和外设及通信接口的驱动库函数。

当 CC3200 上电、复位或者被从休眠模式唤醒时，ROM 内的芯片初始化程序会立即得到执行，对芯片内的硬件进行初始化。初始化完毕后，引导加载程序开始执行，将应用程序代码加载至 SRAM 中运行，并将程序指针设置为应用程序入口地址，以此执行用户应用程序。

③ 存储器映像。

CC3200 存储器映像见表 3.1。其中包含各种外设在存储器中的映像地址。

表 3.1 存储器映像

起始地址	结束地址	说明	注释
0x0000 0000	0x0000 FFFF	片内 ROM（Bootloader+DriverLib）	64KB
0x2000 0000	0x2003 FFFF	位带片内 SRAM	256KB
0x4000 4000	0x4000 4FFF	通用输入/输出接口 GPIOA0	
0x4000 5000	0x4000 5FFF	通用输入/输出接口 GPIOA1	
0x4000 6000	0x4000 6FFF	通用输入/输出接口 GPIOA2	
0x4000 7000	0x4000 7FFF	通用输入/输出接口 GPIOA3	
0x4000 C000	0x4000 CFFF	串行异步收发器接口 UARTA0	
0x4000 D000	0x4000 DFFF	串行异步收发器接口 UARTA1	
0x4002 0000	0x4002 07FF	内部集成电路总线接 I^2CA0（主模式）	
0x4002 0800	0x4002 0FFF	内部集成电路总线接 I^2CA0（从模式）	
0x4003 1000	0x4003 0FFF	通用定时器 GPTA0	
0x4003 1000	0x4003 1FFF	通用定时器 GPTA1	
0x4003 2000	0x4003 2FFF	通用定时器 GPTA2	
0x4003 3000	0x4003 3FFF	通用定时器 GPTA3	
0x400F E000	0x400F EFFF	系统控制	
0x400F F000	0x400F FFFF	微型直接存储器存取（μDMA）	

续表

起始地址	结束地址	说　明	注　释
0x4401 C000	0x4401 EFFF	多通道音频串行接口（MrASP）	
0x4402 1000	0x4402 2FFF	串行闪存器行外设接口 SSPI	外部串行闪存使用
0x4402 E000	0x4402 EFFF	MCU 共享配置	
0x4402 F000	0x4402 FFFF	休眠配置	
0x4403 0000	0x4403 FFFF	加密范围（包括 4 个加密相关模块）	
0x4403 0000	0x4403 0FFF	DTHE 寄存器和 TCP 校验和	
0x4403 5000	0x4403 5FFF	安全散列算法 SHA/消息摘要算法 MD5	
0x4403 7000	0x4403 7FFF	高级加密标准 AES	
0x4403 9000	0x4403 9FFF	数据加密标准 DES	
0xE000 0000	0xE000 0FFF	测量跟踪宏单元（ITM）	
0xE000 1000	0xE000 1FFF	数据观察点和跟踪（DWT）	
0xE000 2000	0xE000 2FFF	闪存补丁和断点（FPB）	
0xE000 E000	0xE000 EFFF	嵌套向量中断控制器（NVIC）	
0xE004 0000	0xE004 0FFF	跟踪端口接口单元（TPIU）	

3.1.2 WiFi 网络处理器子系统（CC3100）

WiFi 网络处理器（CC3100）包含一个额外的专用 ARM MCU 负责 WiFi 功能，可完全免除应用 MCU 的处理负担，包含 802.11 b/g/n 射频（WiFi Radio）、基带（WiFi Baseband）、强大可快速的加密引擎 MAC（WiFi MAC），可以实现支持 256 位加密的快速安全的互联网连接。CC3200 的 WiFi 网络处理器内部结构如图 3.3 所示。

图 3.3　CC3200 的 WiFi 网络处理器内部结构

图 3.3 中，片内 ROM 中存储了所有 WiFi 和 Internet 的协议，包括完整的 WiFi 协议栈、TCP/IP 协议栈、TLS/SSL 协议栈及其他协议，专门用于处理网络的通信任务。WiFi 网络处

理器可设置为站点 STA、无线接入点 AP 及 WiFi 直接连接三种模式，支持个人 WPA2 安全加密、企业安全加密及 WPS2.0。

WiFi 网络处理器支持的主要特性见表 3.2。

表 3.2 WiFi 网络处理器支持的主要特性

特　性	类　别	域	说　明
IPv4	网络协议栈	TCP/IP	IPv4 协议栈
TCP/UDP	网络协议栈	TCP/IP	基本协议
TLS/SSL	安全	TCP/IP	TLS v1.2（客户端/服务器）/SSL v3.0
DHCP	协议	TCP/IP	客户端和服务器模式
mDNS	应用	TCP/IP	多插域名服务
HTTP	应用	TCP/IP	URL 静态和动态响应模版
Policies	连接	WLAN	允许管理连接和重连接策略
WPS2	配置	WLAN	初始化产品配置 PBC 和 PIN 方法
AP Config	配置	WLAN	初始化产品配置接入点方式（配置网页和信标信息元）
SmartConfig	配置	WLAN	初始化产品配置的替代方法
Station	角色	WLAN	具有传统 802.11 节电功能的 802.11b/g/n 站点
AP	角色	WLAN	具有传统 802.11 节电功能的 802.11b/g 站点
P2P	角色	WLAN	P2P 客户端

开发者在进行 WiFi 网络通信开发时，可根据 TI 公司提供的网络通信库手册，跳过协议栈的知识进行开发。对协议栈有兴趣的读者可以自行参考相关资料。

3.1.3 电源管理子系统

电源管理子系统集成了 DC/DC 直流变换器，支持宽电压输入，可以工作在低功耗模式（包括睡眠、深睡眠、低功耗深睡眠及休眠等），在只有实时时钟（RTC）工作的休眠模式下，所需求的电流可小于 4μA。

1. 供电方式

CC3200 电源管理子系统的供电方式可选择宽电源电压（VBAT）方式或预稳压 1.85V 模式。

宽电源电压模式的输入电压为 2.1~3.6V，主要通过直流—直流变换器（DC/DC）和低压差稳压器（LDO）为系统提供不同的电压和电流，即数字电压变换器（DIG DC/DC）、模

拟电压变换器（ANA DC/DC）及供电电压变换器（PADC/DC）。市面的大多数芯片均采用此方式供电。

预稳压 1.85V 模式需要旁路 ANA DC/DC 和 PA DC/DC，且供电引脚的连接方式不同于宽电源电压模式。另外，所提供的 1.85V 电压还需要满足负载电流能力大于等于 900mA，线路和负载在 500mA 阶跃电流调节下的纹波小于 2%，负载阶跃设置时间少于 4μs，与 CC3200 芯片的距离足够近，从而保证芯片电阻压降足够小。

2. 电源管理模式

CC3200 包含两个独立的系统：应用 MCU 和 WiFi 网络处理器系统。每个系统都有对应的电源管理模式，应用 MCU 系统的电源管理模式见表 3.3。用户可以编程控制其进入其中的一种模式类型。

表 3.3　应用 MCU 系统的电源管理模式类型

模式类型	描　述
运行模式	MCU 正常运行
睡眠模式	MCU 工作时钟被禁止，但保留设备的状态信息，在睡眠模式下可被定时器、GPIO 及外设唤醒
低功耗深度睡眠模式（LPDS）	MCU 禁止时钟且不保留设备状态信息，但用户能够编程设置保留的代码和 MCU 特殊配置信息，在该模式下仍可被定时器和 GPIO 唤醒
休眠模式	该模式将功耗降至最低，仅保留少部分供电，实时时钟（RTC）不受休眠模式影响，可被 RTC 定时器、外部事件及 GPIO 唤醒，但唤醒时间较长

WiFi 网络处理器会处于工作模式或者低功耗深度睡眠模式，需要注意两种模式的切换：没有网络活动时，WiFi 网络处理器会自动进入 LPDS 状态，仅在信标接收时才会自动被唤醒至工作状态。WiFi 网络处理器的电源管理模式类型见表 3.4。

表 3.4　WiFi 网络处理器的电源管理模式类型

模式类型	描　述
网络低三层工作模式	发送或接收 IP 协议数据包
网络低两层工作模式	发送或接收 MAC 数据帧，不需要对 IP 协议数据包进行处理
网络主动监听模式	优化降低主动接收信标帧（其他帧不主动接收）功耗的特殊模式
网络连接空闲模式	网络处理器在两个信标的接收之间会自动进入 LPDS 模式，并且会在接收一个信标后从 LPDS 模式醒来，在 LPDS 模式中自动检测接入点的情况，如果连接不通，则不会退出 LPDS 模式
低功耗深度睡眠模式（LPDS）	在两次信标接收之间，由网络处理器所保持的一种低功耗状态，该状态可被快速唤醒
禁用网络功能模式	无网络通信功能

3.2 CC3200 LaunchPad

CC3200 LaunchPad 是 TI 公司推出的专门为物联网应用提供解决方案的硬件平台，以基于 ARM Cortex-M4 低功耗内核的 CC3200 微控制器为核心，内置 WiFi 连接，具有无线网络可编程功能。CC3200 LaunchPad 提供了传感器、按键、发光二极管（LED）、调试器（Debugger）等方便开发者使用的外围部件。开发者只需安装相应的 CCS/IAR 集成开发环境，通过板载 USB 和 PC 连接即可实现应用调试开发。

CC3200 LaunchPad 具有如下特性：

- 板载业界首款集成无线 WiFi 的微控制器 CC3200；
- 板载温度计和加速度计传感器、3 个 LED 和 2 个用户按键；
- 使用 UART 接口通过 USB 与 PC 连接实现调试开发；
- 引出多达 40 个引脚拓展其他功能；
- 支持 JATG 和 SWD 调试接口；
- 外电源引脚提供多种供电方式；
- 具有在板天线和外接天线两种选择；
- 低功耗，普通 2 节 5 号电池即可供电；
- 使用串行 Flash 编程（S-Flash）。

3.2.1 硬件电路

CC3200 LaunchPad 的板载资源如图 3.4 所示。整个 CC3200 LaunchPad 板卡分为两部分：调试部分和无线控制通信部分。CC3200 LaunchPad 板卡还另外搭载了温度和加速度传感器，配合无线通信，可实现传感器数据采集传输。CC3200 LaunchPad 板卡搭载的 BoosterPack 扩展接口（2×20pin）可以兼容 TI 公司的 CC3200MODLAUNCHXL 和 CC3200AUDBOOST 扩展板，以实现快速的原型设计开发，如无线音乐盒等，使用时需要注意，防止插反、连接引脚错位。

CC3200 LaunchPad 硬件电路功能框图如图 3.5 所示。

由图 3.5 可知，CC3200 LaunchPad 的调试部分采用了 FTDI 的芯片 FT2232D，可以实现串口和调试功能，免去了在开发过程中对外部调试器的需求。由于 CC3200 LaunchPad 板卡采用单芯片的无线通信解决方案，所以外部所需的电子元器件较少，电路设计比较简单。

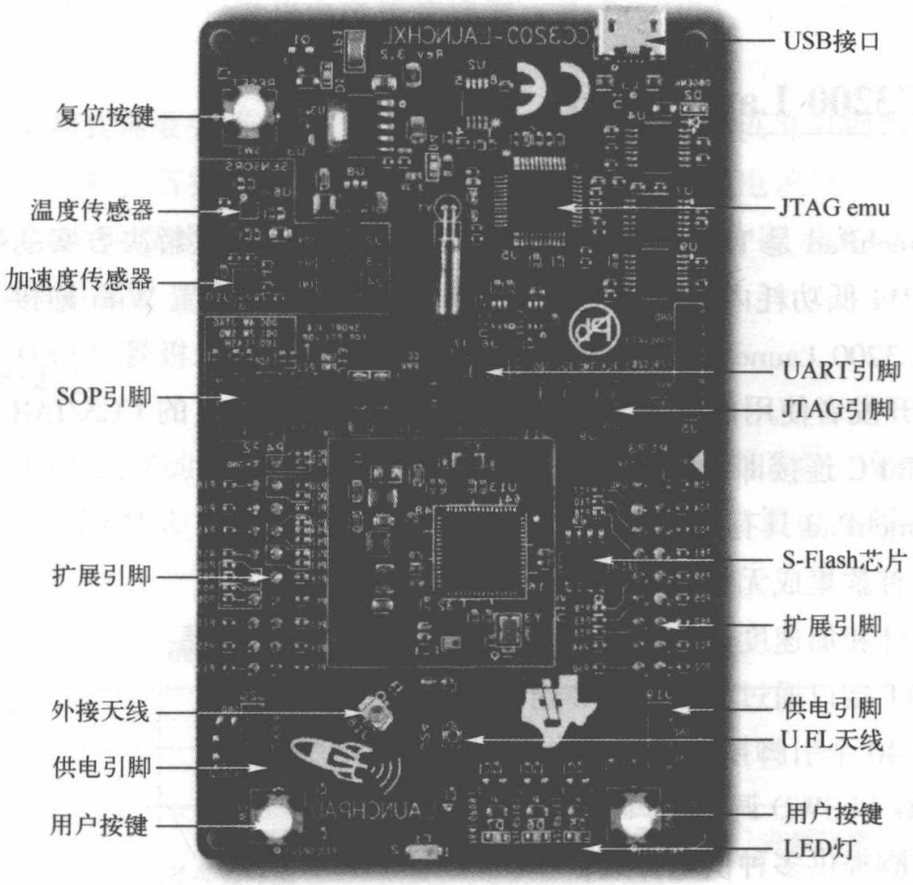


图 3.4　CC3200 LaunchPad 的板载资源

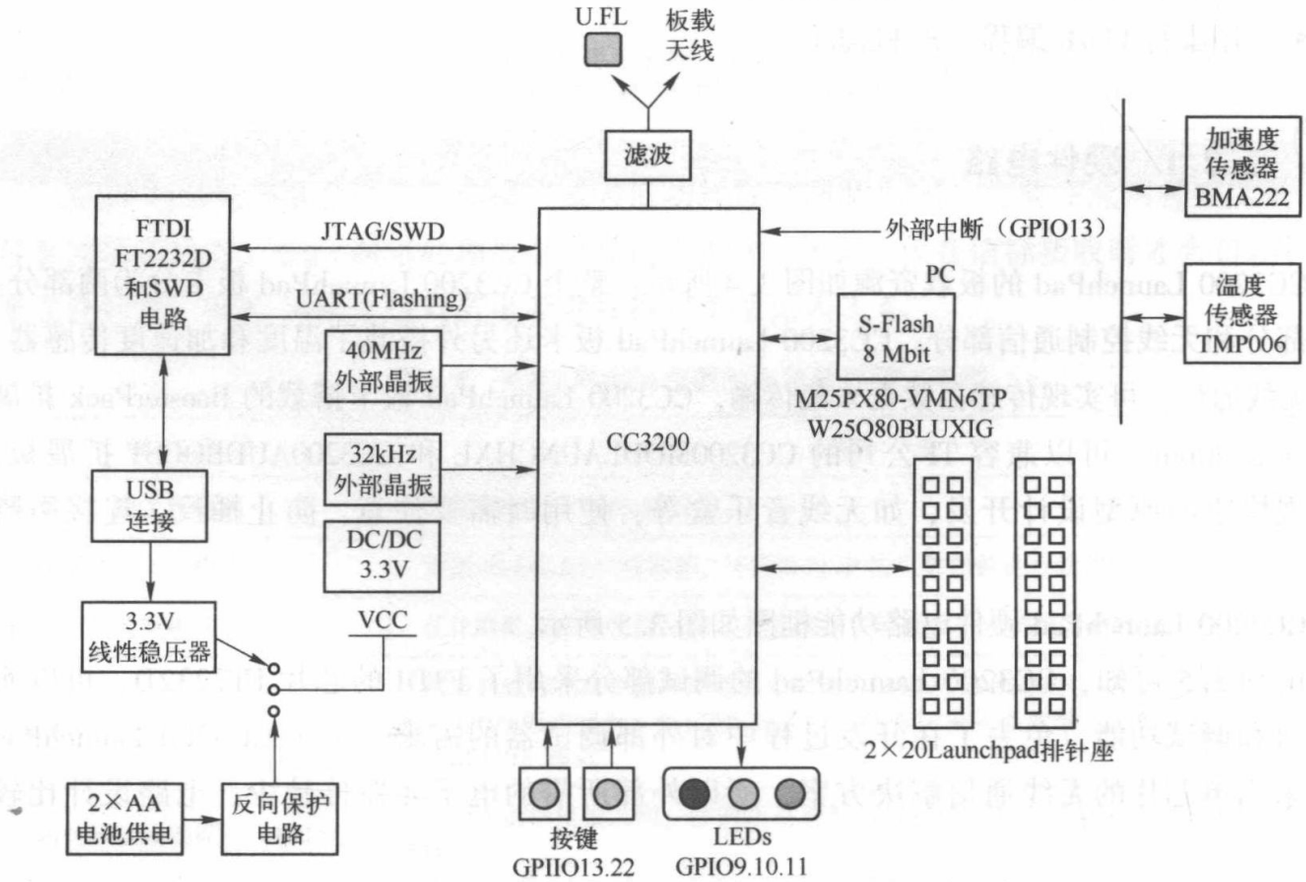


图 3.5　CC3200 LaunchPad 硬件电路功能框图

3.2.2 跳线设置

CC3200 LaunchPad 除了基本的板载资源，在应用过程中需要特别注意跳线及接口的使用。

1. JTAG 调试接口

JTAG 调试接口采用跳线的方式。JTAG 接口的跳线方式如图 3.6 所示。图中，上部分为板载 FTDI JTAG 仿真器，下部分为 CC3200 的 JTAG 引脚，使用 FTDI 仿真器时，直接通过短路帽连接。如果要使用外部的仿真器，则先移除跳线帽，然后直接连接下部分的 JTAG 引脚。需要注意的是，如果使用 SWD 模式，则只需要连接 TCK 和 TMS 引脚；如果使用电池供电，则为了减小功耗，需要拔掉 JTAG 引脚上的所有短路帽。

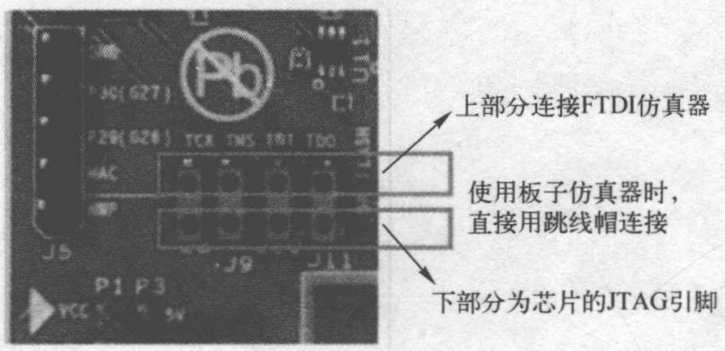


图 3.6 JTAG 接口的跳线方式

2. I^2C 接口

I^2C 接口的跳线方式如图 3.7 所示。J2 和 J3 用于 CC3200 芯片的 I^2C 总线与传感器模块单元的连接。移除 J2、J3 的短路帽，加速度计和温度计传感器将从 I^2C 总线上断开，同时也会移除 I^2C 总线的上拉电阻。另外，J4 用于加速度计传感器的中断输出，连接的是 CC3200 的 GPIO13 引脚。

3. 电源供电部分

电源供电部分的跳线如图 3.8 所示。CC3200 LaunchPad 可以通过 micro USB 接口供电。CC3200 LaunchPad 上的 LDO 可以给 CC3200 芯片和其他模块提供 3.3V 电压。J13 用于供电选择，在一般情况下需接上短路帽，由 CC3200 LaunchPad 上的 LDO 供电，否则从电池接口 J20 处取电。J20 是 3.3V 的电源输入接口，可以采用两节 AA 电池串联供电。J12 用于 CC3200 器件的电流测量，正常使用时，需要直接接上短路帽。J19 是 5V 输出接口，电源来自 USB 接口的 V_{BUS}（中间串接了一个二极管，电压降约为 0.4V）。

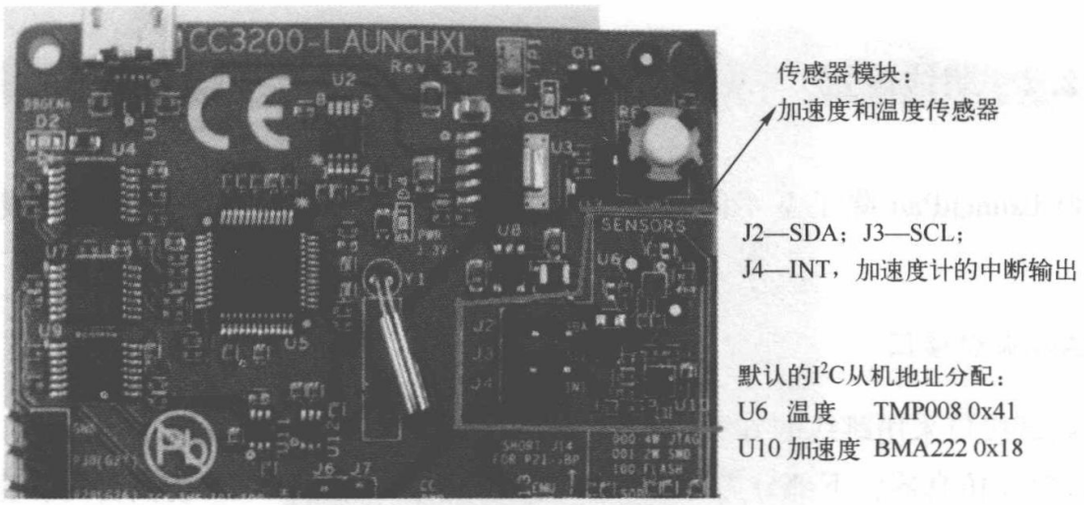


图 3.7 I^2C 接口的跳线方式

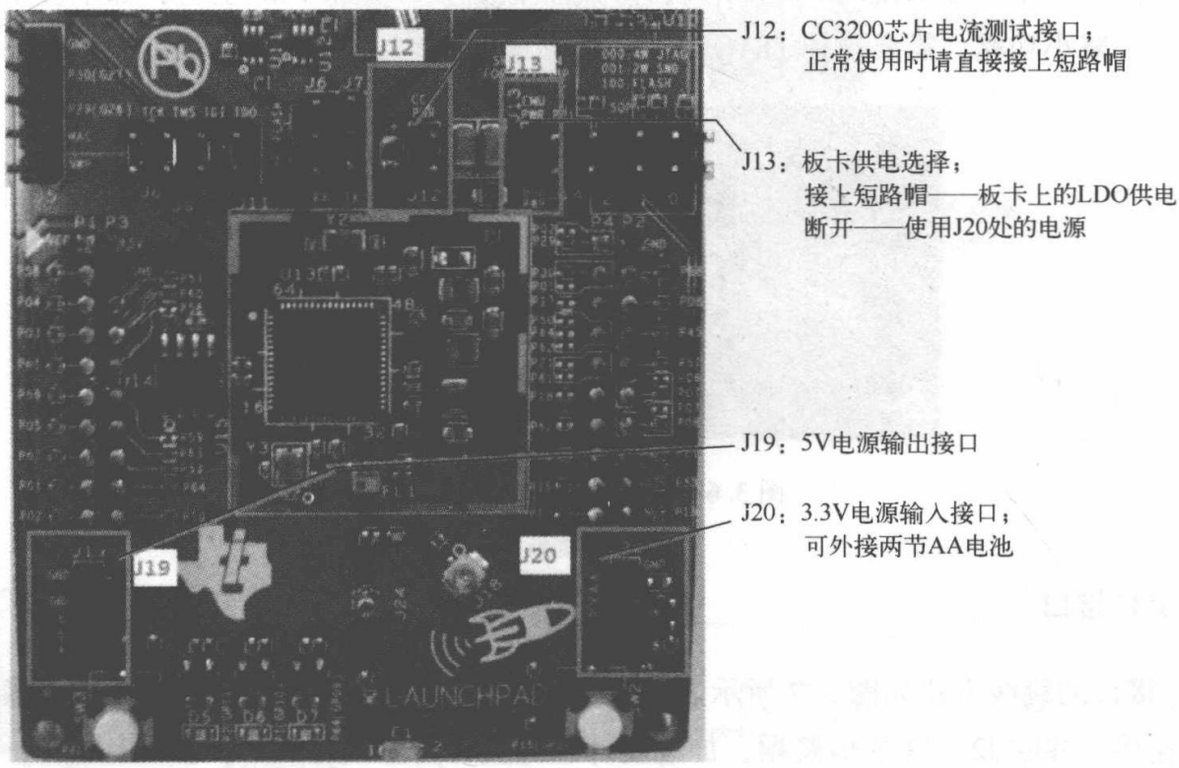


图 3.8 电源供电部分的跳线

4. UART 接口

CC3200 LaunchPad 提供了虚拟串口，使用了芯片 FT2232D。芯片 FT2232D 具有两路接口：一路用于仿真（JTAG/SWD）；另一路用于虚拟串口。同样，通过跳线方式，UART 接口可以连接到 20Pin 的 BoosterPack 上。UART 接口的跳线方式如图 3.9 所示。

5. 工作模式的选择

通过设置 Sense on Power（SOP），CC3200 配置了 3 种不同的工作模式，如图 3.10 所示。SOP 分别连接 CC3200 的 21 引脚、34 引脚及 35 引脚。

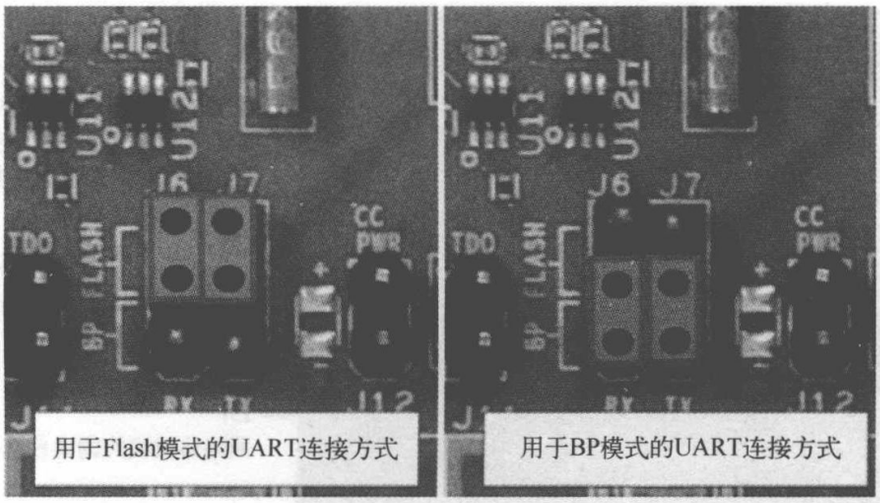


图 3.9　UART 接口的跳线方式

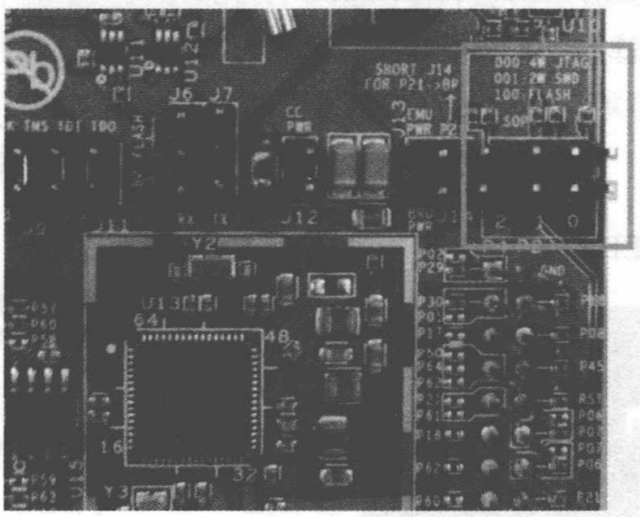


图 3.10　工作模式的选择

6. 其他接口

其他接口的跳线方式如图 3.11 所示。J5 用于网络协议栈的调试接口；J14 用于决定扩展引脚 P21 是否连接芯片的引脚 GPIO25，该引脚是复用的，另一端连接在 SOP2 上，使用时要特别注意。

3.2.3　按键和 LED 灯

1. 按键

CC3200 LaunchPad 上有 3 个按键。其中，1 个为复位按键 SW1，主要用于复位 CC3200，同时也接在 20pin 的扩展引脚上；另外两个为用户按键 SW2 和 SW3，分别与 GPIO 22 和 GPIO 13 连接，当用户按下按键时，会接通高电平。

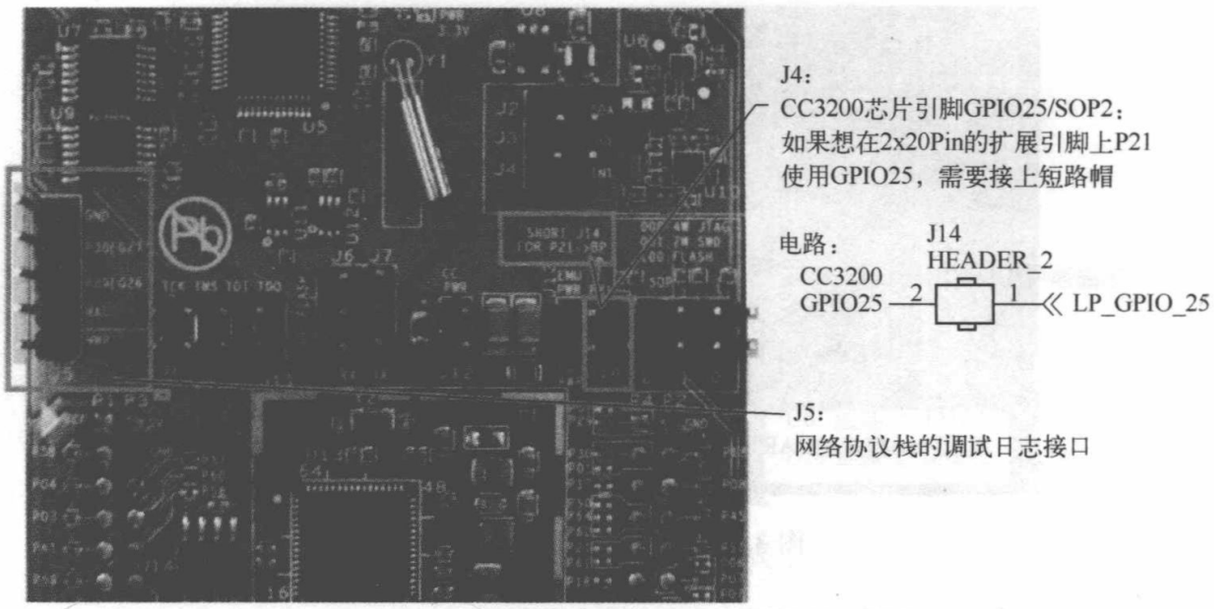


图 3.11　其他接口的跳线方式

2. LED 灯

CC3200 LaunchPad 上有 6 个 LED 灯，如图 3.12 所示。

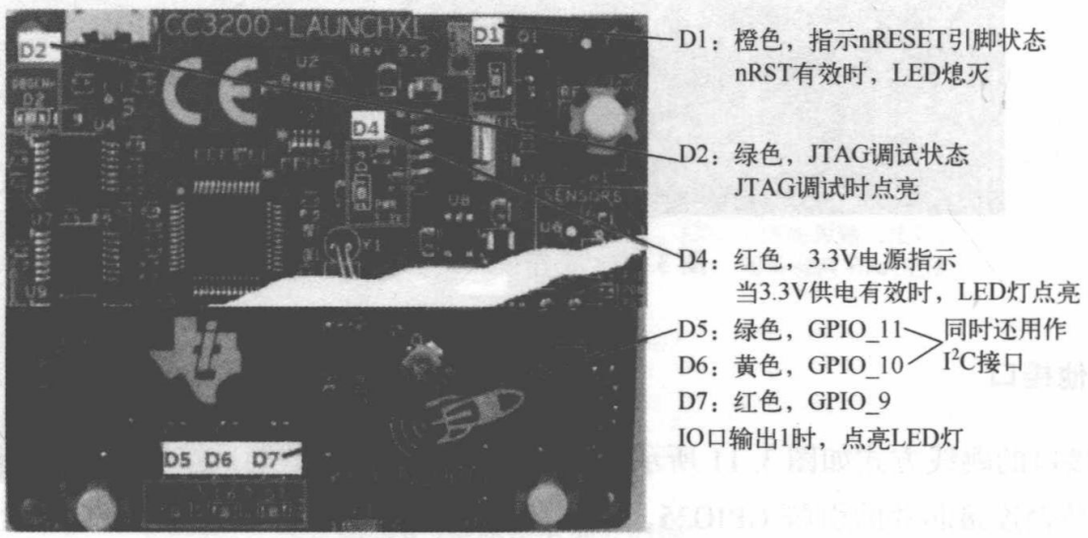


图 3.12　LED 灯

TI 公司为 CC3200 LaunchPad 提供了一整套开发方案，包括编程库、应用演示、开发工具、用户编程向导、参考设计和 TI E2E 技术论坛。其内部有 40 多个与 WiFi 协议、MCU 外设、网络连接等应用相关的实例，用户可方便地依靠这些资料快速上手。后续的章节将会介绍 TI 公司相应软件工具的安装和调试，为实际的应用开发搭建一个软件开发环境。

第 4 章
CC3200 软件开发环境的搭建

软件开发环境是支持系统软件和应用软件进行工程化开发和维护而使用的一组软件。一般来讲，软件开发环境可以被认为是对目标项目工程进行开发和设计的一系列工具。前面的章节已经对 CC3200 LaunchPad 硬件平台进行了介绍，本章将针对 CC3200 LaunchPad 硬件平台搭建一个完整的软件开发环境。

4.1 CCS 集成开发环境

CCS（Code Composer Studio）是一款支持所有 TI 公司微控制器和嵌入式处理器产品的集成开发环境，集成了 C/C ++编译器、代码编辑器、项目管理环境及调试器等一系列用于开发和调试嵌入式应用程序的工具。

4.1.1 获取 CCS V6 软件

登录 TI 公司的官方网站（www. ti. com），单击菜单“Tools&software”下的“Software& development tools”选项，弹出的选择界面如图 4.1 所示。

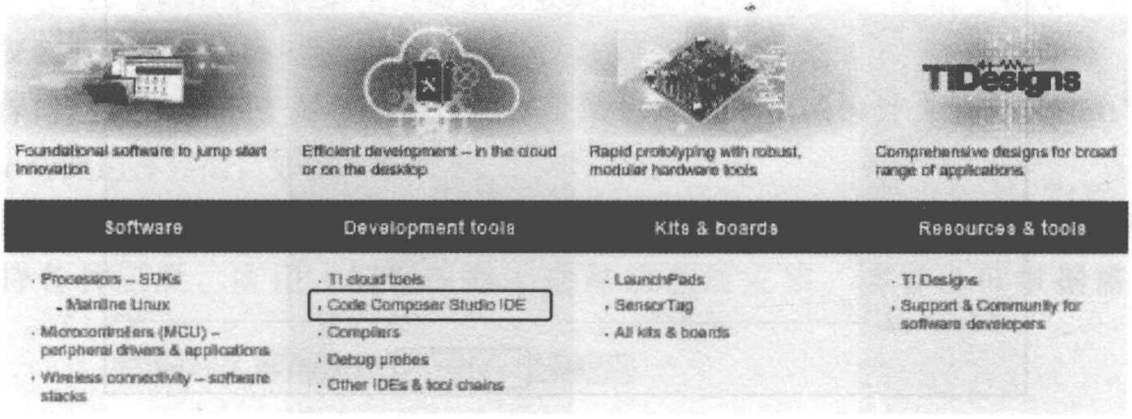


图 4.1　Software&development tools 选择界面

单击图 4.1 中“Development tools”下的“Code Composer Studio IDE”选项，进入 CCS 软件工具介绍界面，如图 4.2 所示。

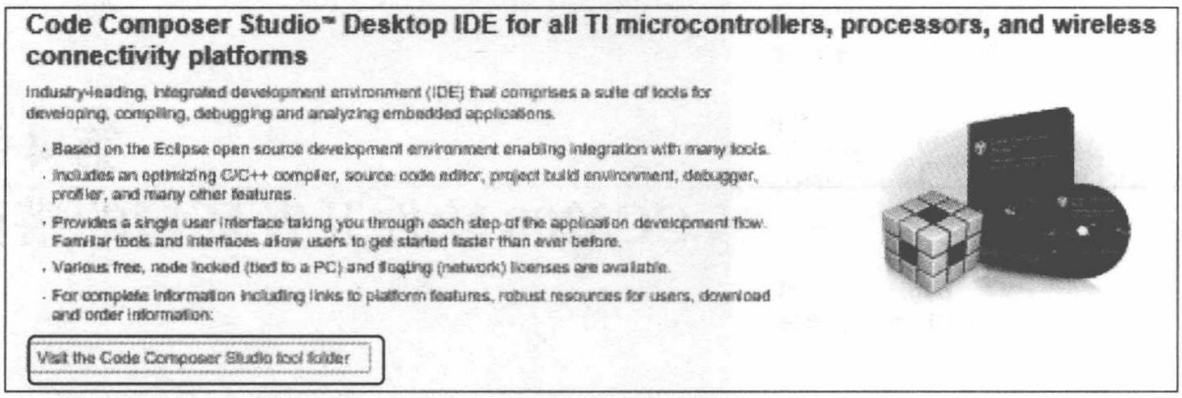


图 4.2　CCS 软件工具介绍界面

单击图 4.2 中的“Visit the Code Composer Studio tool folder”选项进入 CCS 工具介绍网站，在“Download”子项下根据电脑安装的操作系统类别选择“Windows”“Linux”和“Mac”，单击“Windows”即可开始下载。

4.1.2　CCS V6 安装过程详解

双击 CCS V6 安装程序，单击“yes”进入下一步，接受许可协议后，进入安装目录选择界面，如图 4.3 所示。

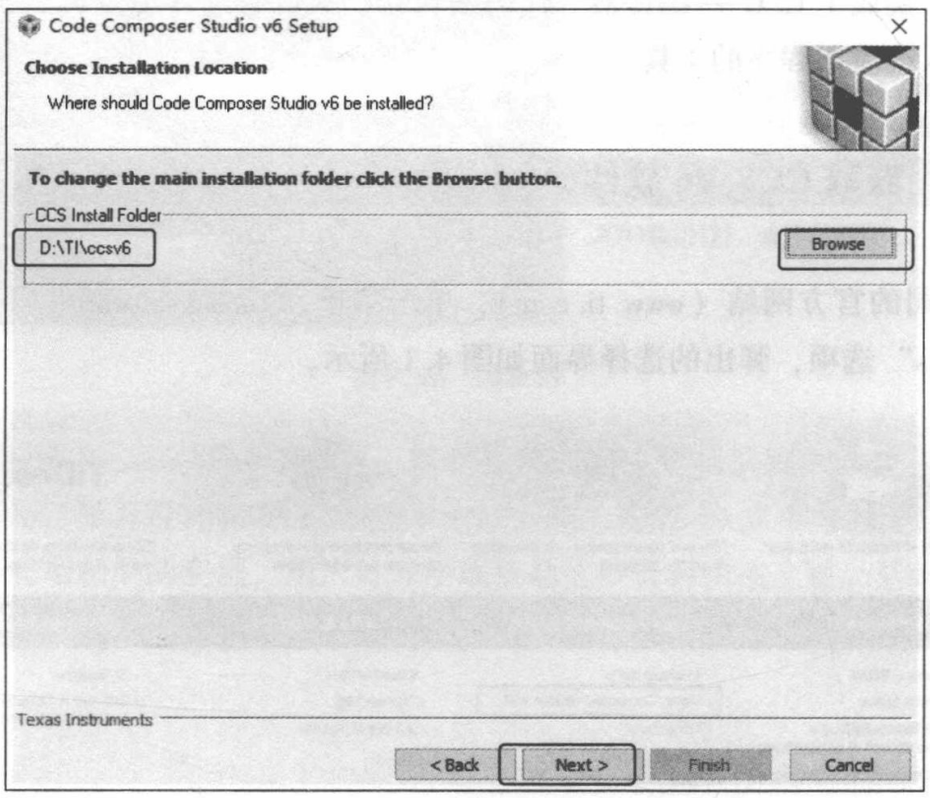


图 4.3　CCS V6 安装目录选择界面

这里不建议使用默认安装目录，通过单击后面的“Browse”选择合适的安装路径。注意，安装路径不可为含有中文、空格等特殊符号的路径，否则将会导致软件安装失败，必须使用英文、数字及下画线组合的路径名。

安装目录设置完毕后，单击“Next”，进入微控制器芯片选择界面，如图 4.4 所示。

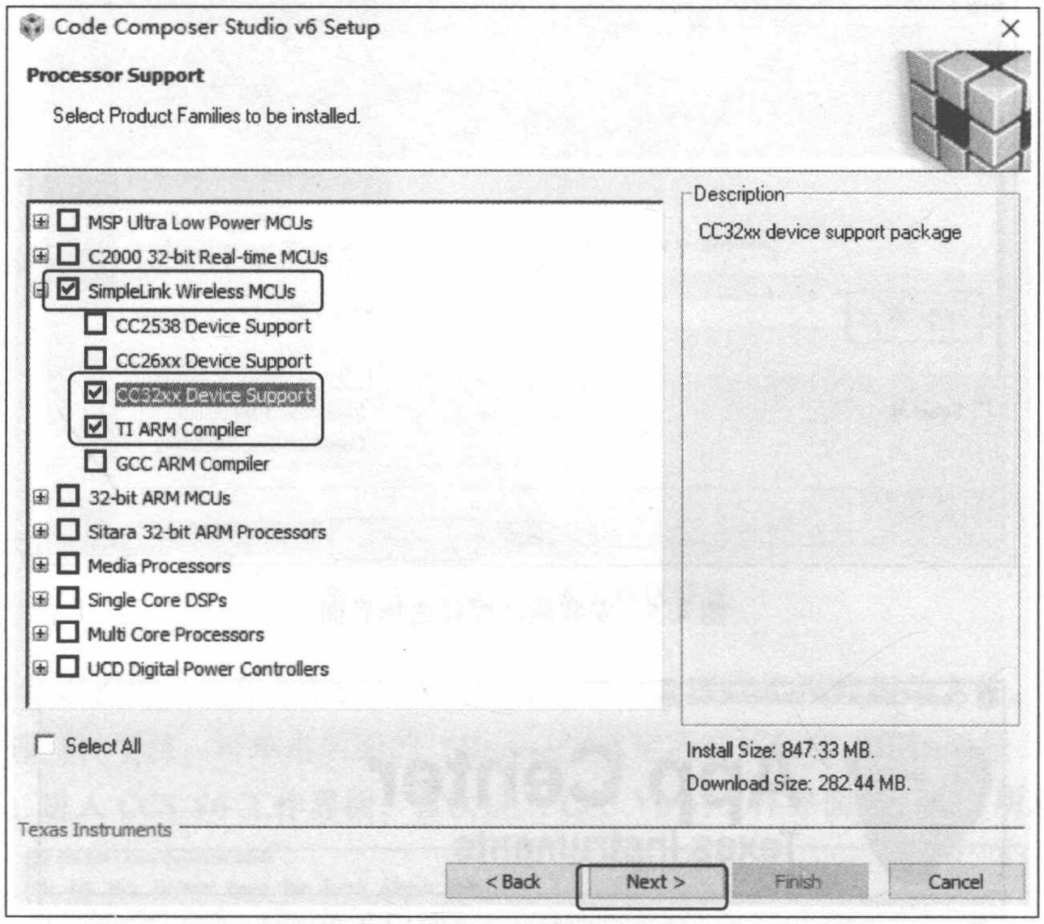


图 4.4　微控制器芯片选择界面

图 4.4 中有多个芯片类型以供选择，需要选中“SimpleLink Wireless MCUs”选项下的“CC32xx Device Support”和“TI ARM Compiler”复选框。由于 CCS 软件要求开发者根据需要来选择安装包，因此可以根据项目需求安装其他的器件安装包。单击“Next”进行下一步，进入安装调试探针选择界面，如图 4.5 所示。

选项保持默认选择不变即可，也可根据需要自行选择调试探针，单击“Next”，进入 App Center 功能选择界面，如图 4.6 所示。

保持默认选项不变，单击“Finish”完成配置，等待 CCS V6 安装结束。CCS 的 App Center 支持各种工具、应用、例程等附加功能的在线安装。读者也可根据需要自行勾选。

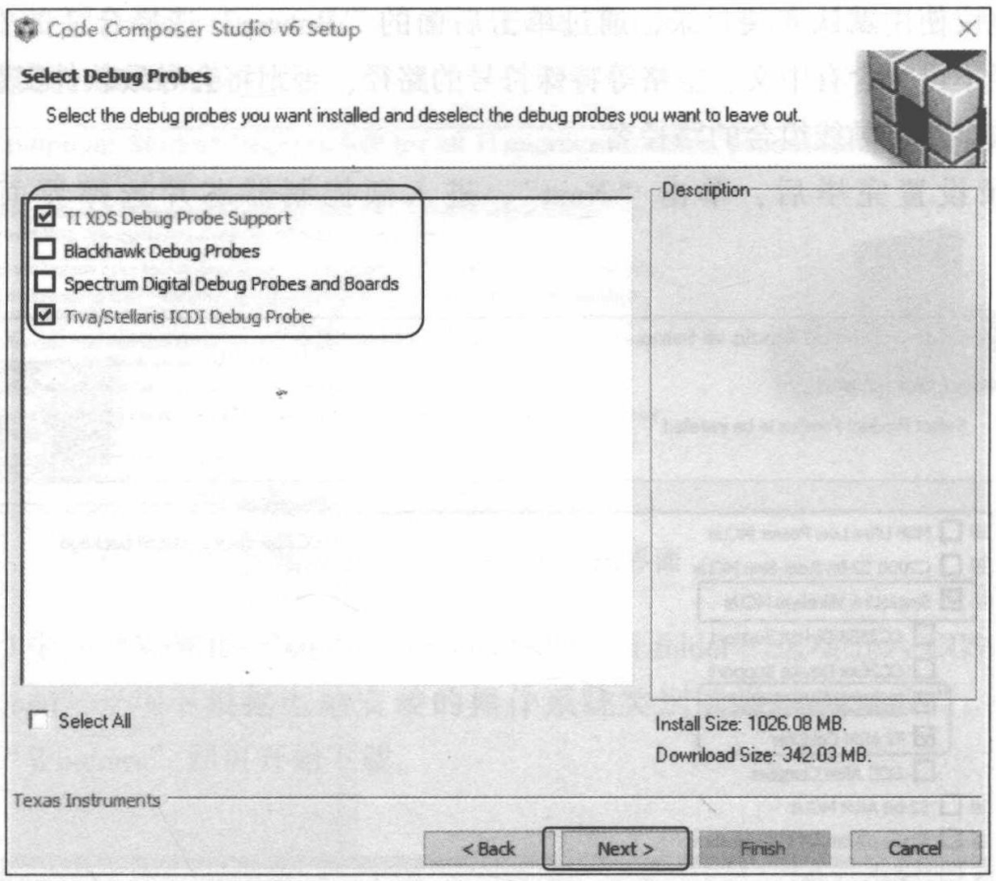


图 4.5　安装调试探针选择界面

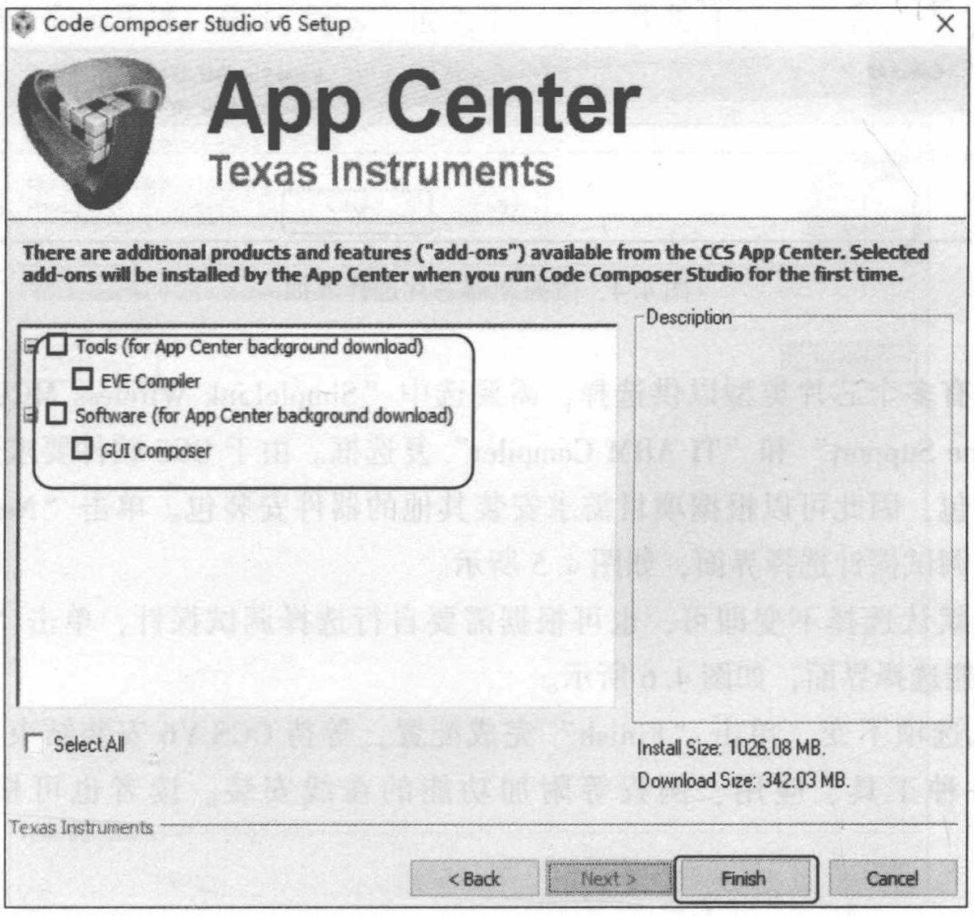


图 4.6　App Center 功能选择界面

4.1.3 CCS V6 软件配置

使用 CCS V6 之前，要进行相关的配置，安装必要的工具包。启动界面和工作区设置如图 4.7 所示。

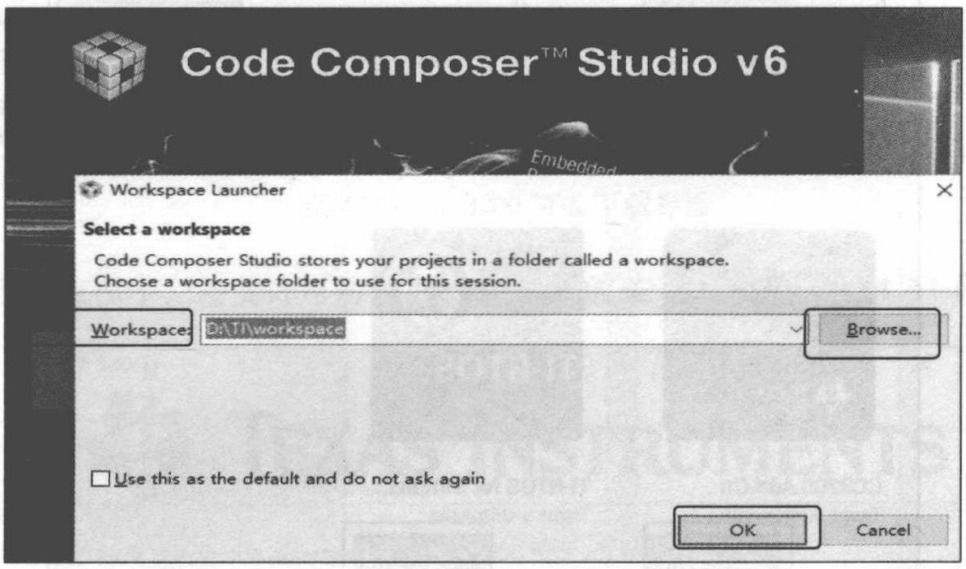


图 4.7　启动界面和工作区设置

CCS V6 需要预先设定开发工作区 Workspace 来存放项目工程的相关文件。这一功能有助于合理管理用户项目，可单击后面的“Browse”选定所需的任意工作区路径，单击“OK”完成设置后，进入 CCS V6 工作界面。首次进入 CCS V6 的工作界面如图 4.8 所示。

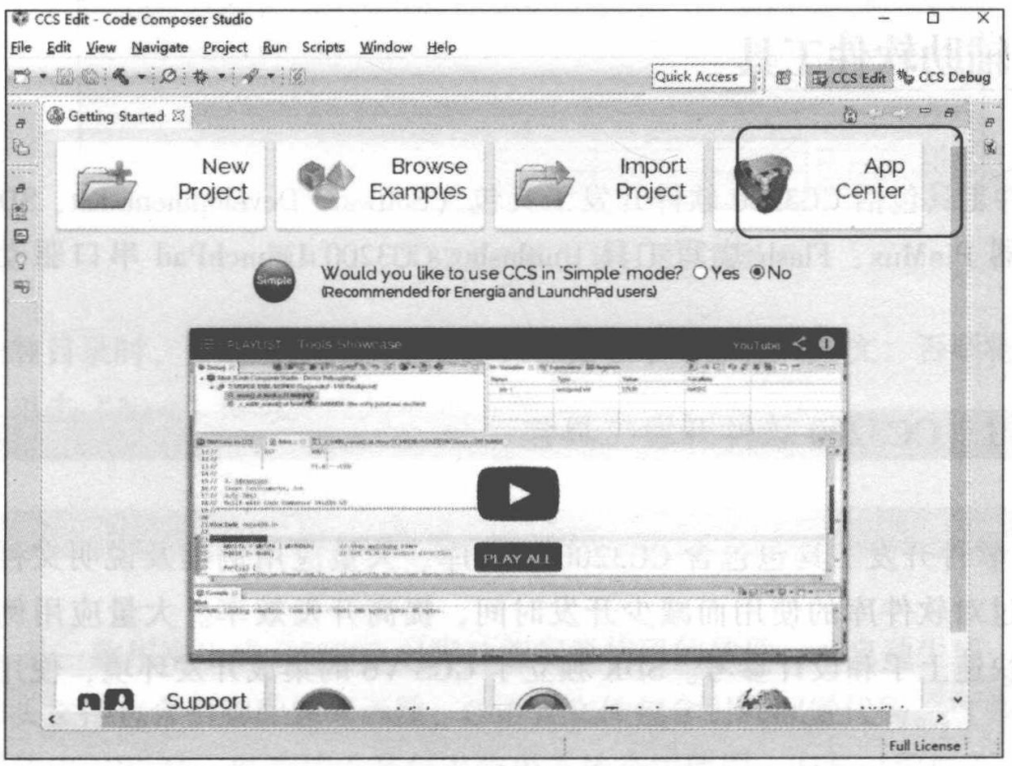


图 4.8　首次进入 CCS V6 的工作界面

单击图 4.8 中的“App Center”，在图中的界面搜索框内输入“cc3200”后，单击搜索框后面的下拉菜单，选择“All”选项。App Center 安装包搜索界面如图 4.9 所示。

图 4.9　App Center 安装包搜索界面

选中“CC3200 Add-On”的“Up to Date”和“TI-RTOS for SimpleLink”的“Selected”选项，单击“Install Software”完成安装。

4.2　辅助软件工具

辅助软件工具包括 CC3200 软件开发工具包（Software Development Kit，SDK）、引脚配置代码生成器 PinMux、Flash 烧写工具 Uniflash、CC3200 LaunchPad 串口驱动、串口终端 Tera Term 等。

4.2.1　CC3200 软件开发工具包

CC3200 软件开发工具包包含 CC3200 软件库、大量应用例程及说明文档。项目研发过程可以通过对软件库的使用而减少开发时间，提高开发效率。大量应用例程和说明文档也有助于快速上手和设计参考。SDK 独立于 CCS V6 的集成开发环境，使用时需要单独安装。

登录 www. ti. com/tool/cc3200sdk 网站，单击 SimpleLink Wi-Fi CC3200Software Development Kit 后面的“Download”完成下载，下载界面如图 4.10 所示。

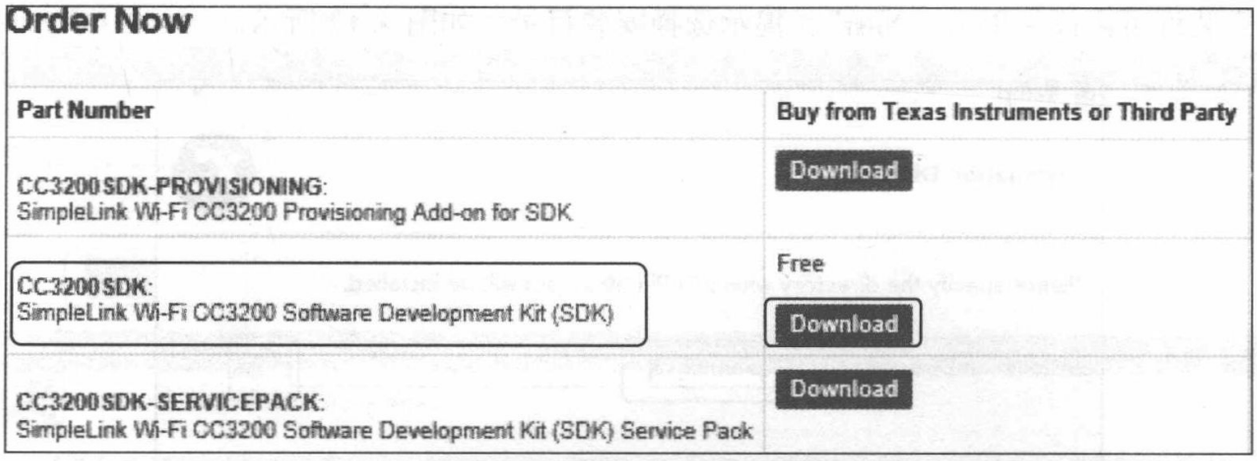


图 4.10　CC3200 SDK 下载界面

双击运行安装程序，接受许可协议后，选择 SDK 安装目录，如图 4.11 所示。

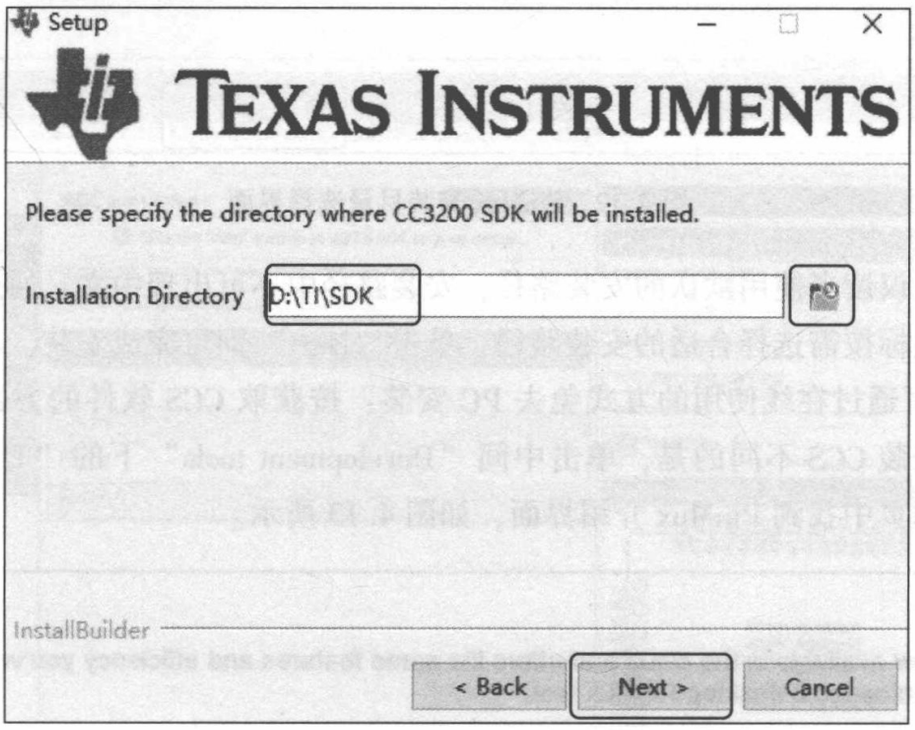


图 4.11　CC3200 SDK 安装目录选择界面

选择安装目录时，不建议使用默认的路径，路径中不要存在中文，否则将会出现不可预知的错误。单击“Next”即可完成 SDK 的安装。

4.2.2　引脚配置代码生成器 PinMux

PinMux 是一款用来生成 CC3200 引脚功能配置代码的软件，可自动生成一个头文件和 C 语言文件：头文件包含引脚配置库函数；C 语言文件包含引脚配置代码。这些文件可直接在项目工程中使用。PinMux 既支持安装包安装也支持在线使用，安装过程如下。

双击运行“pinmux-3.0.625.setup.exe”应用程序，单击“Next”进入安装许可界面，

接收安装许可协议。单击“Next”，提示选择安装目录，如图 4.12 所示。

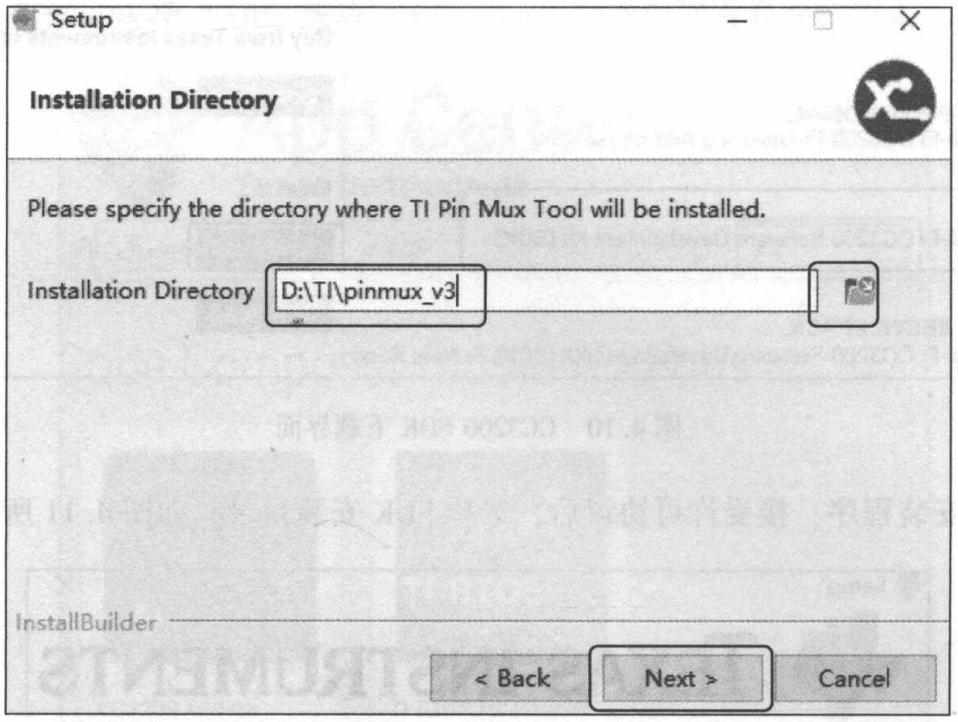


图 4.12　PinMux 安装目录选择界面

同样，不建议读者使用默认的安装路径，安装路径中不可出现中文。通过单击安装路径后面的文件夹图标按需选择合适的安装路径，单击“Next”即可完成安装。

PinMux 还可通过在线使用的方式免去 PC 安装：按获取 CCS 软件的方式进入图 4.1 所示的界面，与安装 CCS 不同的是，单击中间“Development tools”下的“TI cloud tools”选项，在弹出的网页中找到 PinMux 介绍界面，如图 4.13 所示。

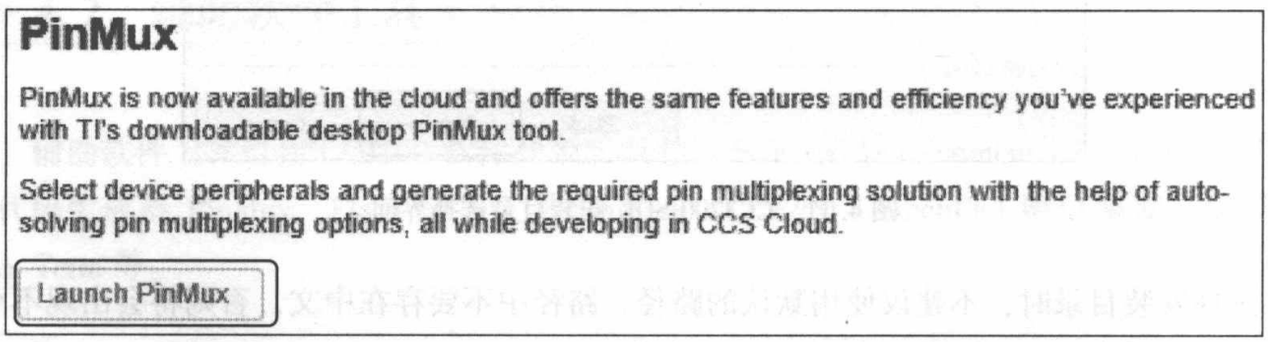


图 4.13　PinMux 介绍界面

单击图 4.13 中的“Launch PinMux”启动 PinMux。注意，在线使用 PinMux 需要 TI 账号，读者自行建立即可。PinMux 启动界面如图 4.14 所示。

设置图 4.14 中的“Device”为“CC3200”。注意，在“Device”选项下有 2 个“CC3200”选项，这里要选择第一个！单击“Start”即可启动 PinMux。如果有之前保存过的引脚配置文件，则可单击“Open”重新打开。在线 PinMux 的工作界面如图 4.15 所示。

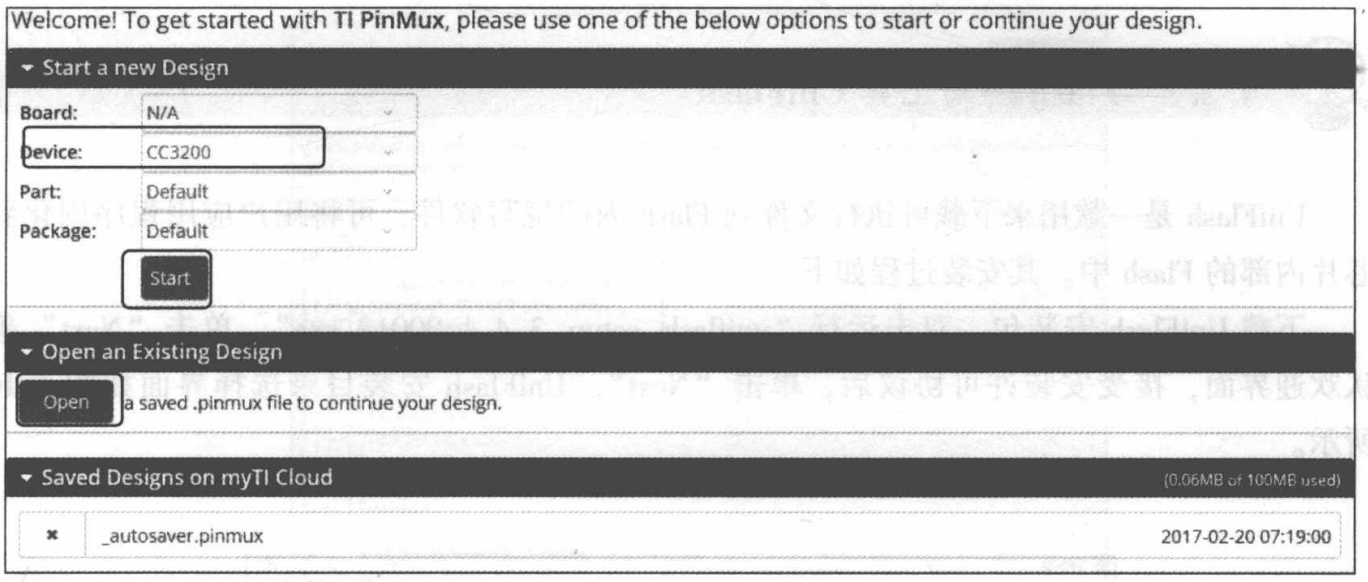


图 4.14　PinMux 启动界面

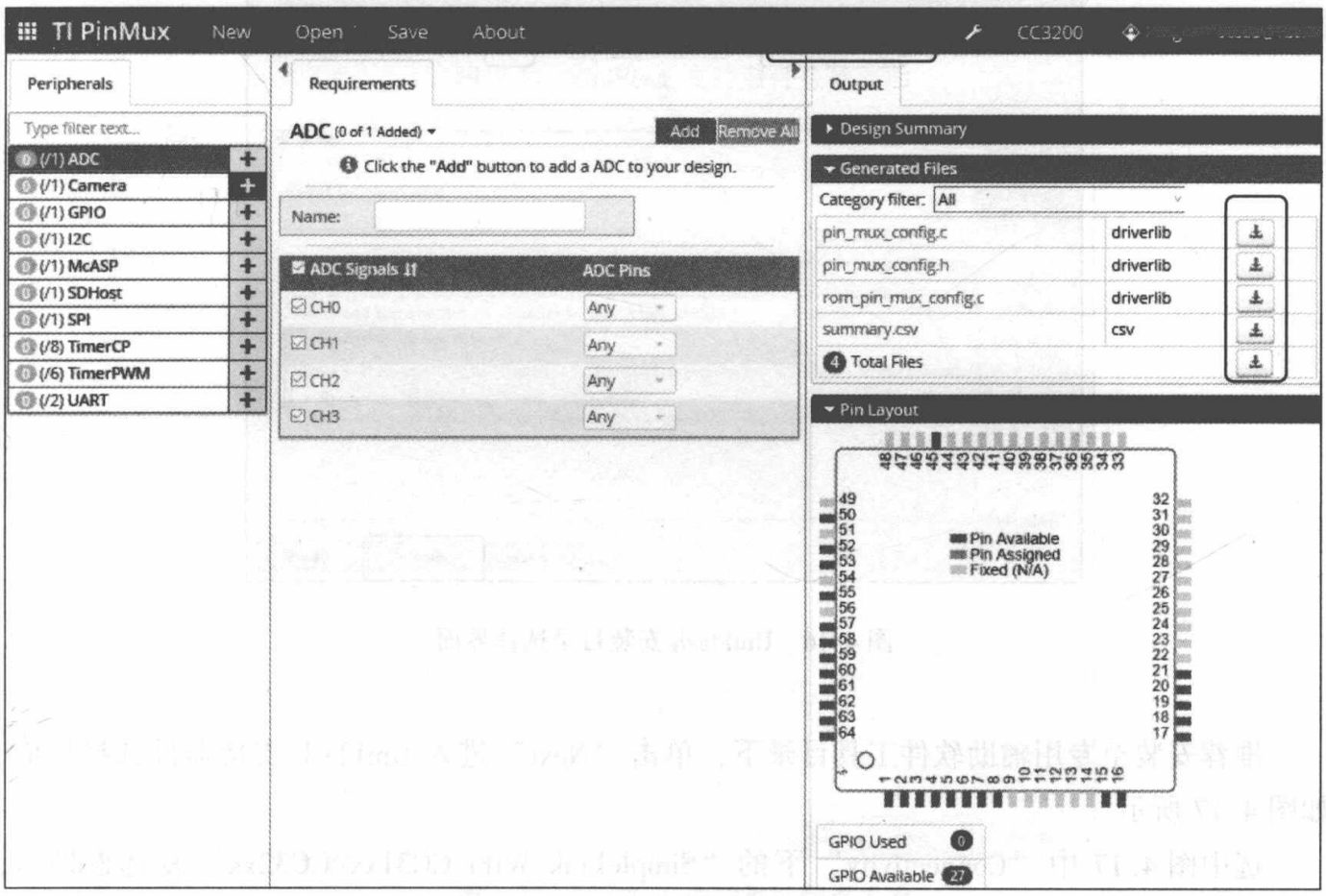


图 4.15　在线 PinMux 的工作界面

在线 PinMux 的工作界面分为三个部分：左侧 Peripherals、中间 Requirements、右侧 Output。Peripherals 为配置外设选择；Requirements 为引脚配置情况；Output 为引脚配置完成后的文件输出，可单击后面下载图标下载保存。

4.2.3 Flash 烧写工具 UniFlash

UniFlash 是一款用来下载可执行文件到 Flash 内的烧写软件，可将用户应用程序固化到芯片内部的 Flash 中。其安装过程如下。

下载 UniFlash 安装包，双击运行“uniflash_setup_3.4.1.00012.exe”，单击“Next”确认欢迎界面，接受安装许可协议后，单击“Next”。UniFlash 安装目录选择界面如图 4.16 所示。

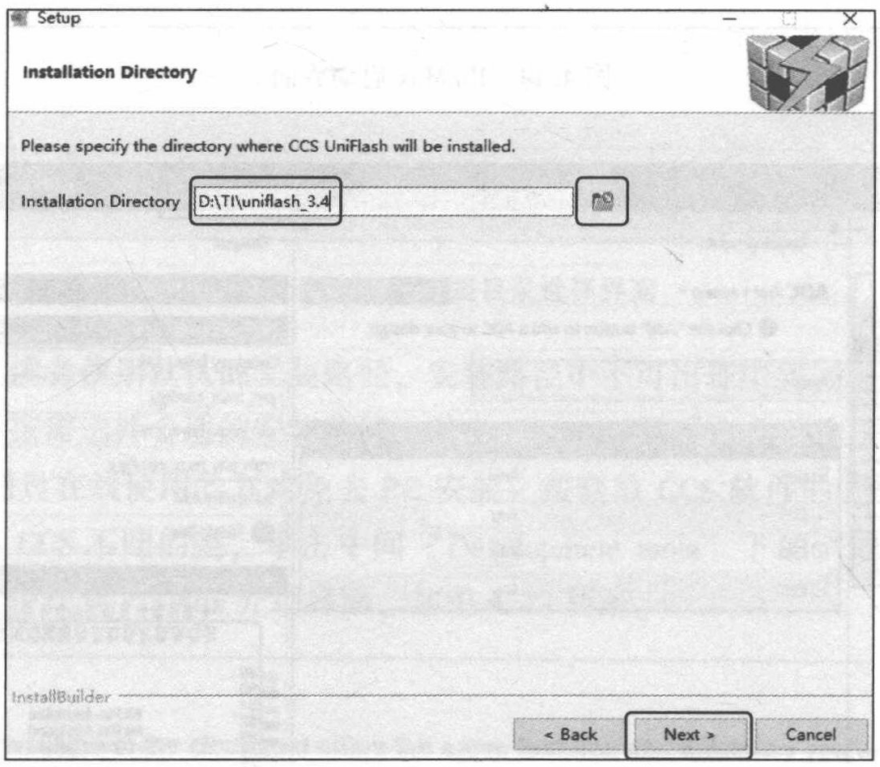


图 4.16 UniFlash 安装目录选择界面

推荐安装至专用辅助软件工具目录下，单击“Next”进入 UniFlash 支持器件选择界面，如图 4.17 所示。

选中图 4.17 中“Connectivity”下的“SimpleLink WiFi CC31xx/CC32xx”复选框即可，可根据需要安装其他的器件。单击“Next”进行下一步，选择调试探针，UniFlash 调试探针配置界面如图 4.18 所示。

默认不变即可，也可自行选择。单击“Next”确认安装信息即可完成安装。

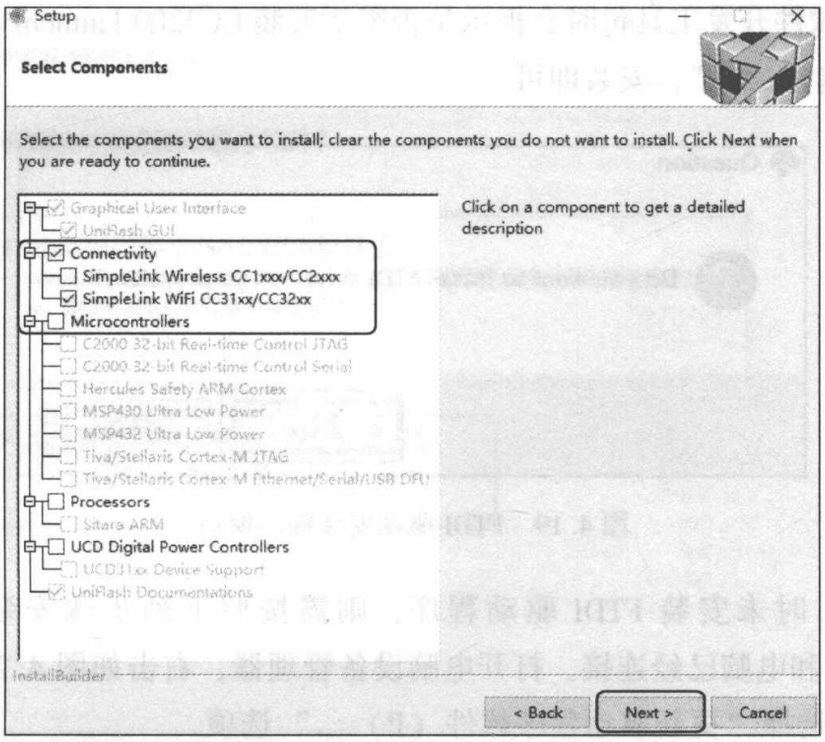


图 4.17　UniFlash 支持器件选择界面

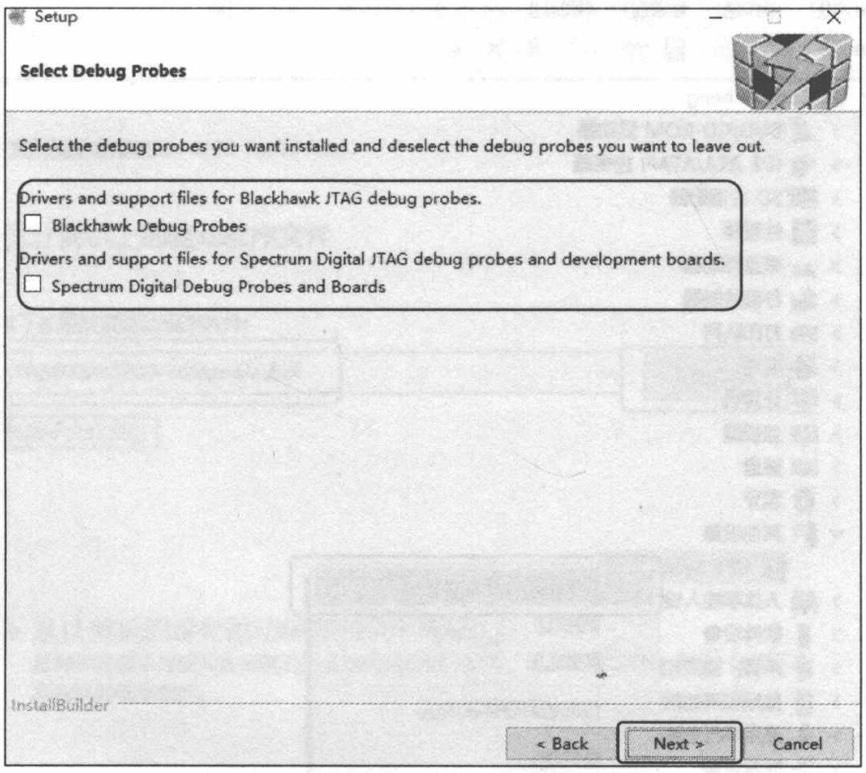


图 4.18　UniFlash 调试探针配置界面

4.2.4　CC3200 LaunchPad 驱动安装

CC3200 LaunchPad 需要通过 USB 和电脑连接识别后，才可通过 CCS V6 软件下载或调试用户应用程序。为了让电脑能够识别 CC3200 LaunchPad，需要安装对应的驱动程序。

安装 CC3200 软件开发工具包时会提示是否需要安装 CC3200 LaunchPad 驱动 FTDI 程序，如图 4.19 所示，选择“是”，安装即可。

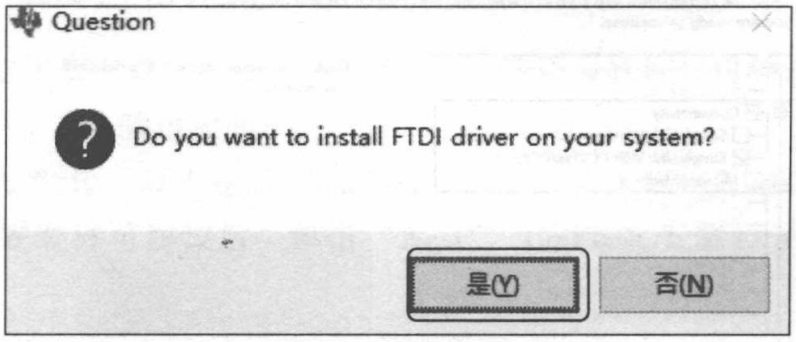


图 4.19　FTDI 驱动安装提示窗口

如果安装 SDK 时未安装 FTDI 驱动程序，则需按照下列步骤安装驱动程序：确定 CC3200 LaunchPad 和电脑已经连接，打开电脑设备管理器，右击如图 4.20 所示的驱动接口“USB Serial Port”选择“更新驱动程序软件（P）...”选项。

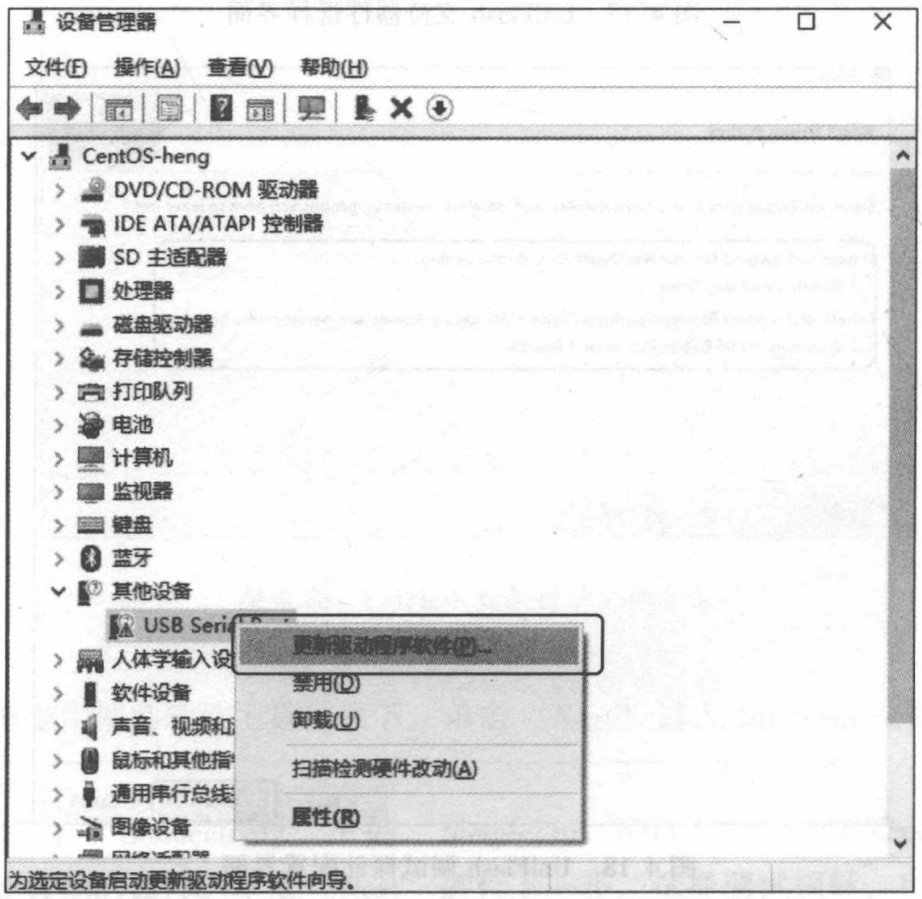


图 4.20　设备管理器界面

在弹出的窗口中选择“浏览计算机以查找驱动程序软件（R）”，如图 4.21 所示。

CC3200 LaunchPad 的接口驱动程序包含在 SDK 中，在之前的内容中已经完成了 SDK 的安装，可直接使用。单击路径后面的“浏览（R）...”按钮，定位路径为“D:\TI\SDK\cc3200-sdk\tools\ftdi”，并选中“包括子文件夹（I）”复选框，如图 4.22 所示。

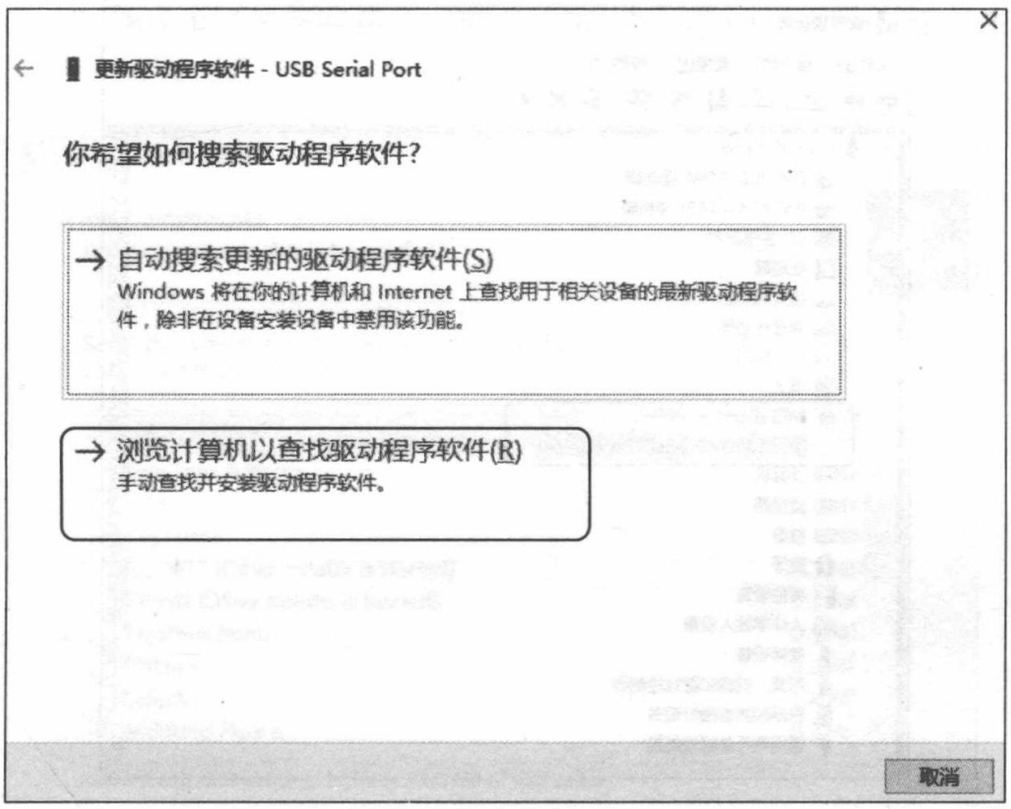


图 4.21　查找计算机驱动程序界面

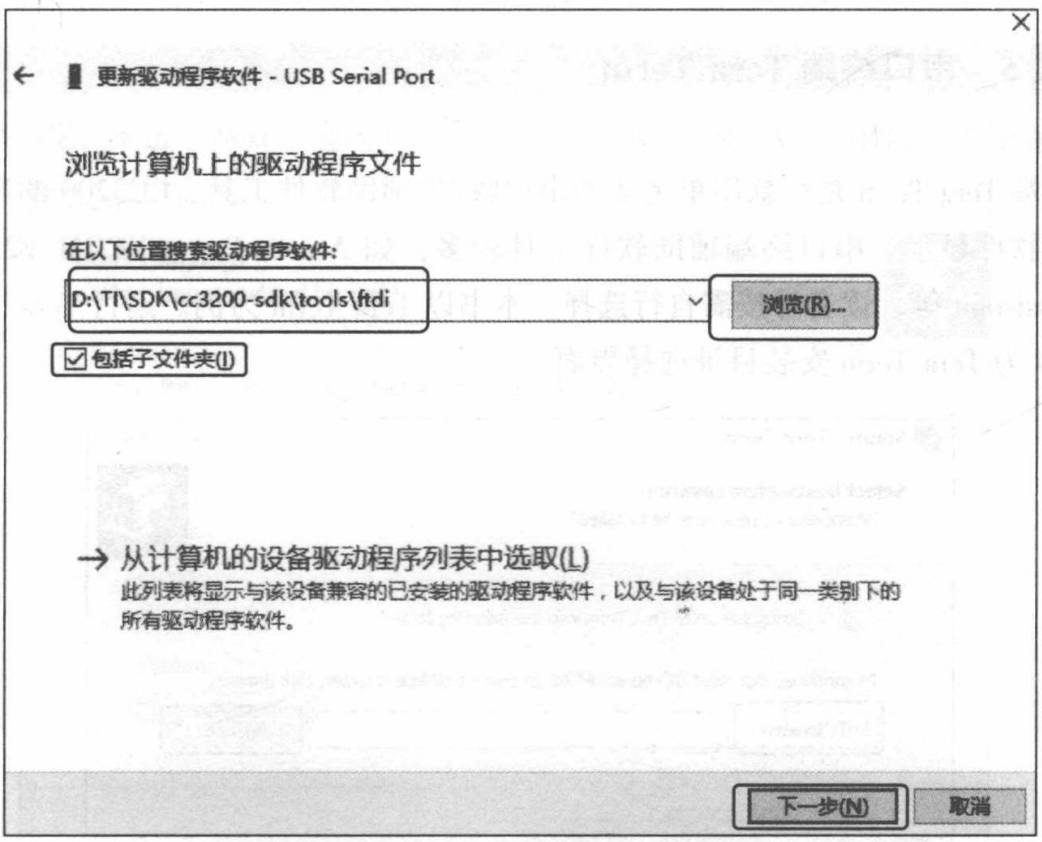


图 4.22　驱动程序路径设置

单击图 4.22 中的“下一步”后，即可完成 CC3200 LaunchPad 驱动程序的安装。驱动程序安装成功界面如图 4.23 所示。

图 4.23　驱动程序安装成功界面

4.2.5　串口终端 Tera Term

串口终端 Tera Term 是一款简单灵活的串口收/发辅助软件工具。CC3200 的串口收/发例程都通过该软件显示。串口终端辅助软件工具较多，如 Access Port、XCOM V2.0、超级终端、Com Assistant 等。读者可按需自行选择。本书以 Tera Term 为例。运行 Tera Term 安装程序，图 4.24 为 Tera Term 安装目录选择界面。

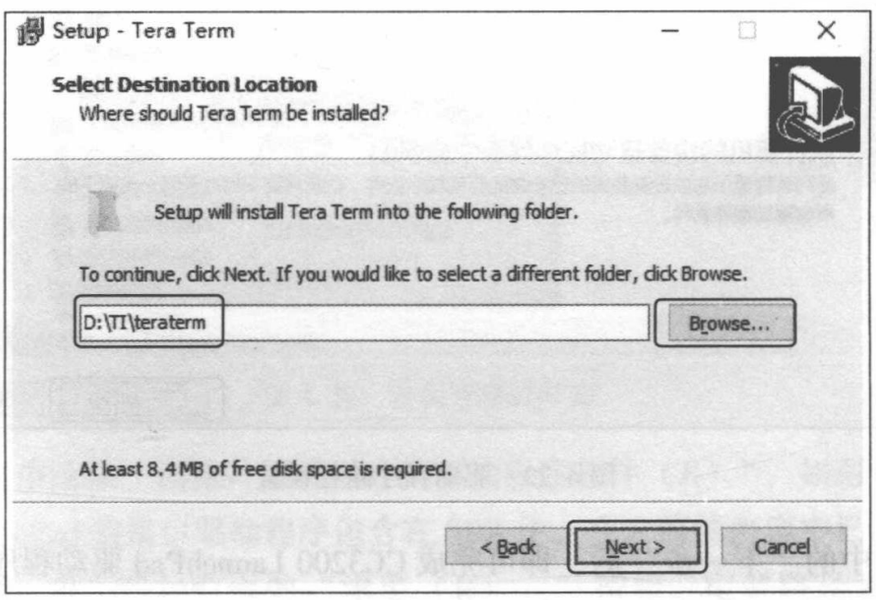


图 4.24　Tera Term 安装目录选择界面

通过“Browse”选择合适的安装路径，单击“Next”，进入 Tera Term 组件安装界面，如图 4.25 所示。

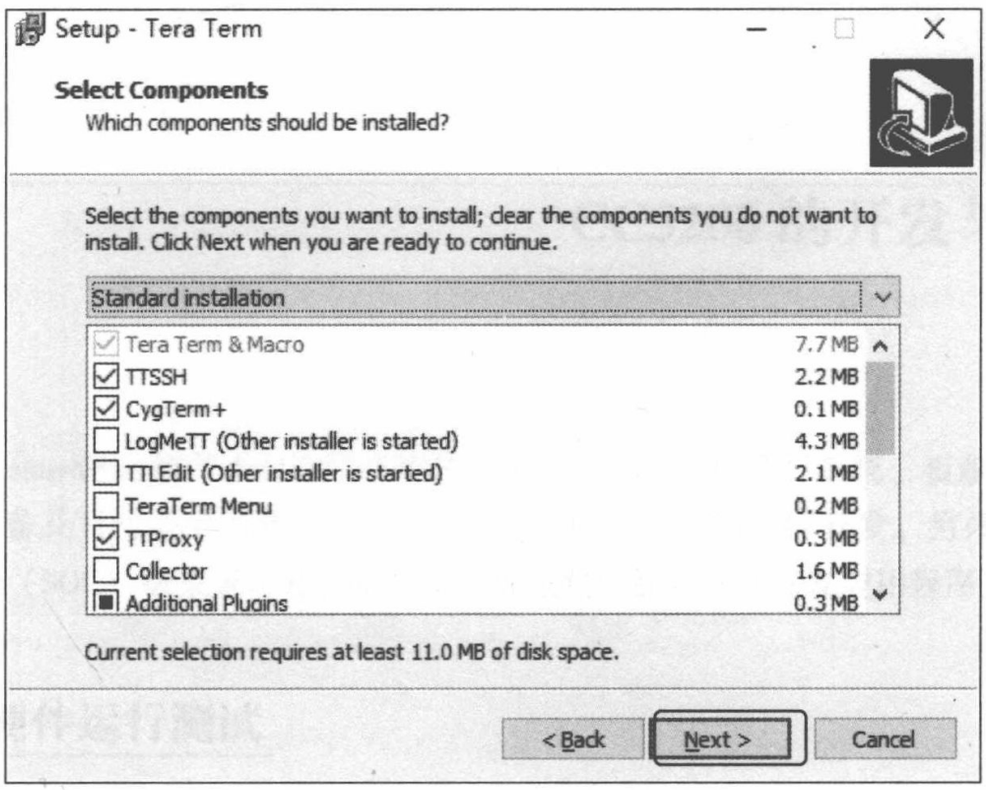


图 4.25　Tera Term 组件安装界面

默认选择不变，单击“Next”进入下一步，选择安装语言为中文（简体），如图 4.26 所示。

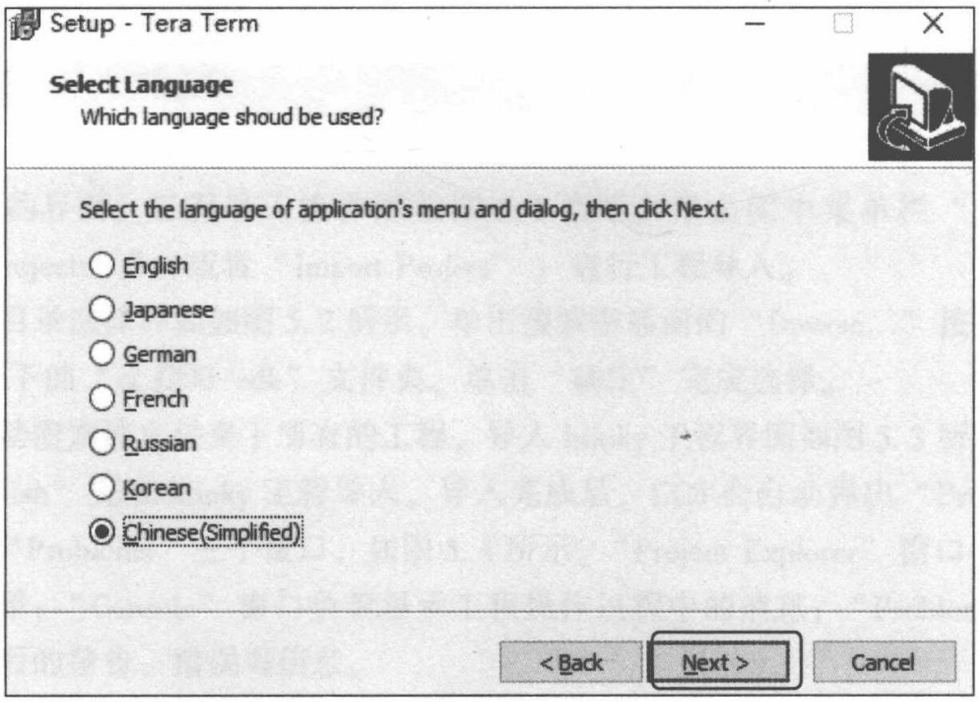


图 4.26　Tera Term 语言选择界面

单击“Next”后，会要求用户设置快捷方式和快速启动，自行按需设置即可。设置完成后，单击最后的“Install”即可完成安装。

第 5 章
CC3200 的开发与应用

CC3200 LaunchPad 硬件平台为用户提供了 CC3200 的最小运行系统、板载调试下载器、外围传感器设备及可拓展的引脚，极大地方便了用户对 CC3200 的开发。另外，CC3200 是一款片上系统（SOC）级集成芯片，不需要外围电路就能够独立开发应用程序。

5.1 硬件运行测试

为验证硬件平台无故障，与电脑连接无问题，在进行开发之前，首先导入 CC3200 软件开发工具包（SDK）中 blinky 流水灯实验测试 CC3200 LaunchPad 的硬件平台。

5.1.1 导入工程

打开 CCS 的界面，工程导入操作图如图 5.1 所示。单击图中菜单栏“Project”下的“Import CCS Projects...”（或者“Import Project”）进行工程导入。

工程导入目录选择界面如图 5.2 所示。单击搜索框后面的“Browse...”按钮，选择之前安装 SDK 目录下的“cc3200-sdk”文件夹，单击“确定”完成选择。

CCS 会自动搜索该文件夹下所有的工程，导入 blinky 工程界面如图 5.3 所示。

单击“Finish”完成 blinky 工程导入。导入完成后，CCS 会自动弹出“Project Explorer”“Console”和“Problems”三个窗口，如图 5.4 所示。“Project Explorer”窗口负责列出用户所有的工程清单；“Console”窗口负责显示工程操作过程中的消息；“Problems”窗口负责显示工程编译后的警告、错误等信息。

双击图 5.4 所示“Project Explorer”窗口中的“blinky”工程，在其后面出现“[Active-Release]”时，表示选中当前项目为有效工程。通过工程前的箭头可以展开工程中所包含的内容。右击工程“blinky”选择“Properties”选项，如图 5.5 所示。

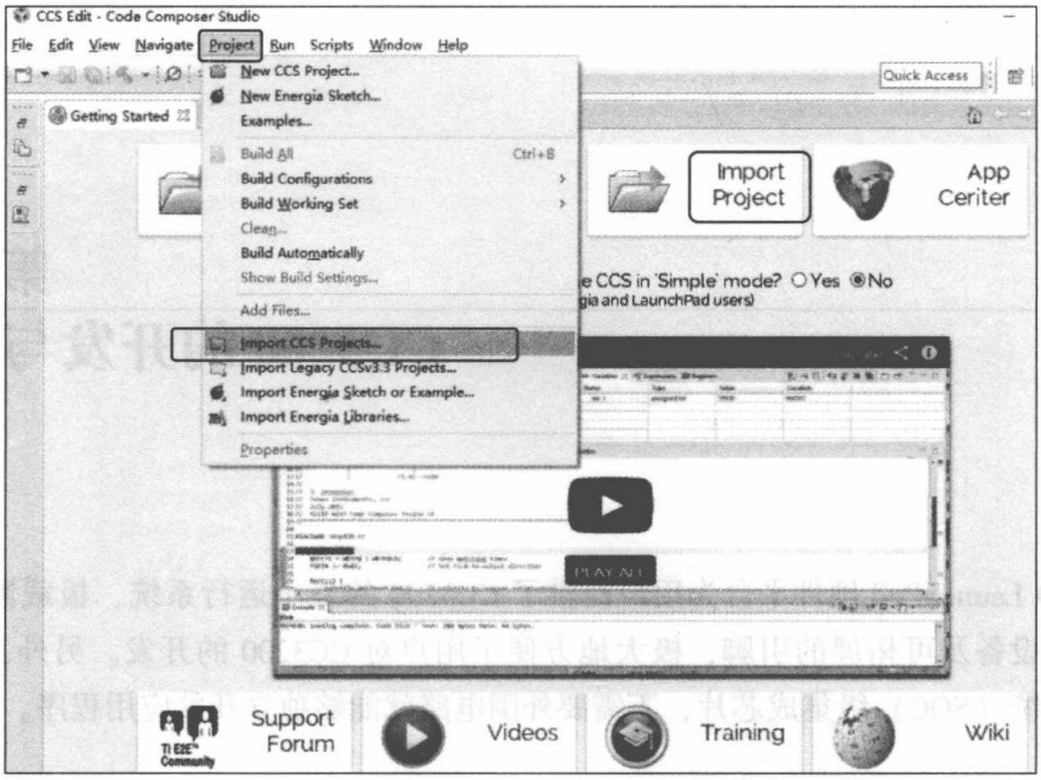


图 5.1　工程导入操作图

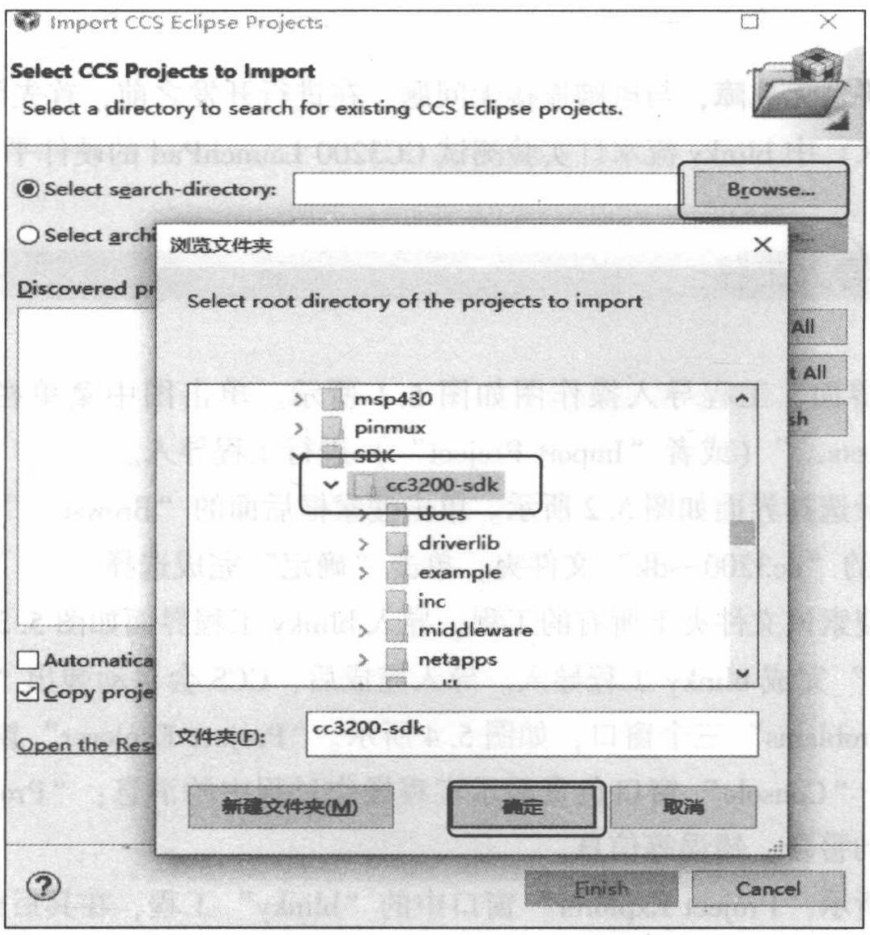


图 5.2　工程导入目录选择界面

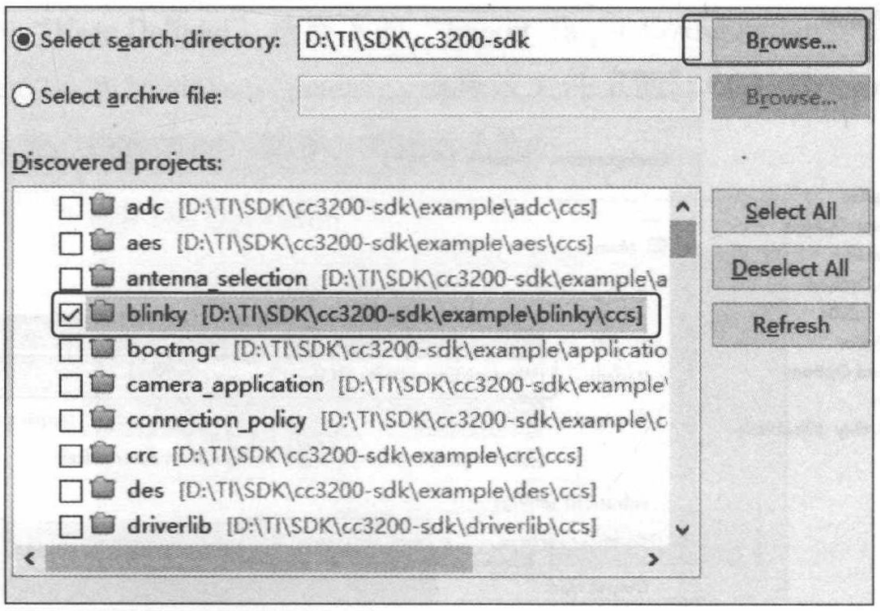


图 5.3　导入 blinky 工程界面

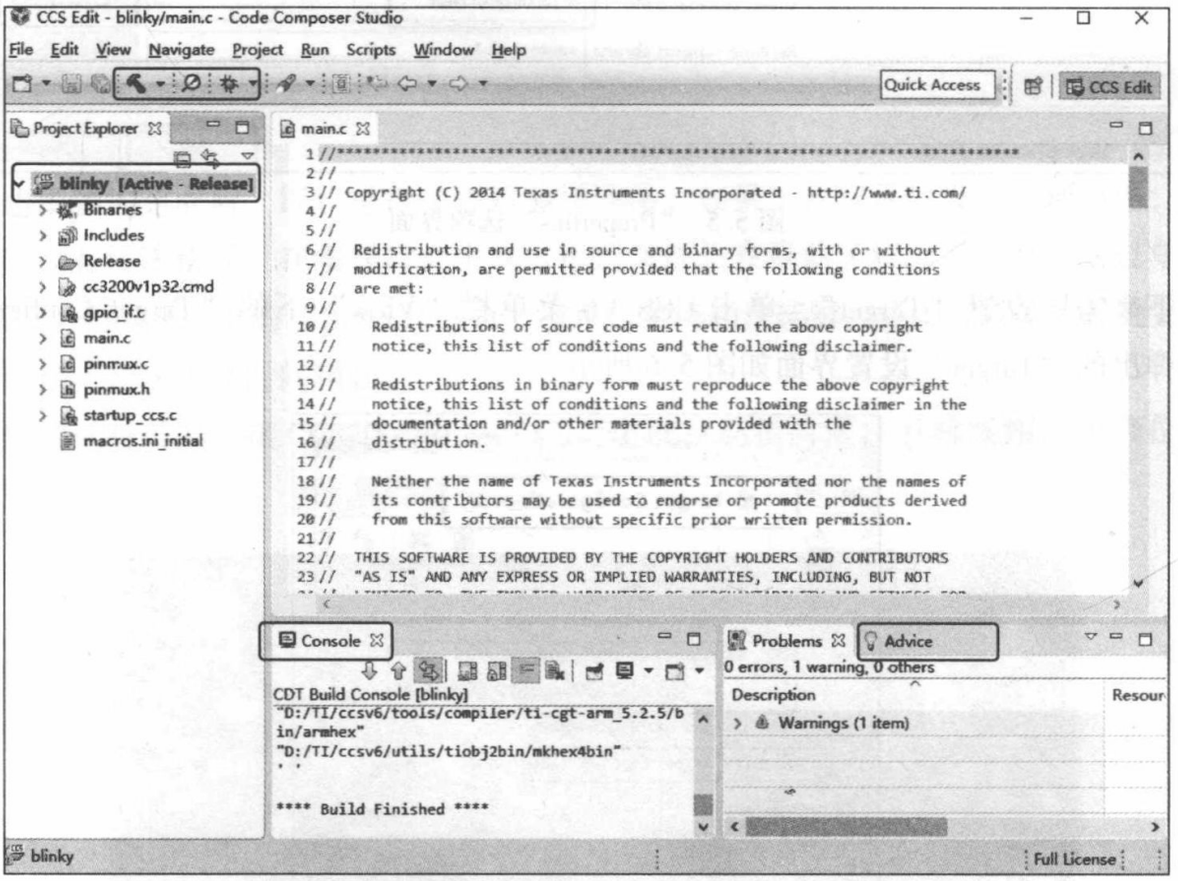


图 5.4　工程管理界面

在图 5.5 的“Variant”选项中选择“Wireless Connectivity MCU”，选中后面的“CC3200”器件，在“Compiler version”选项中选择已安装好的编译器版本，在“Linker command file”中选择“cc3200v1p32. cmd”，单击“OK”，完成设置。

“Properties”选项具有可配置器件类型、编译器版本、头文件目录、优化等级等功能，用户可根据自己的工程情况，合理地配置工程的有关信息。

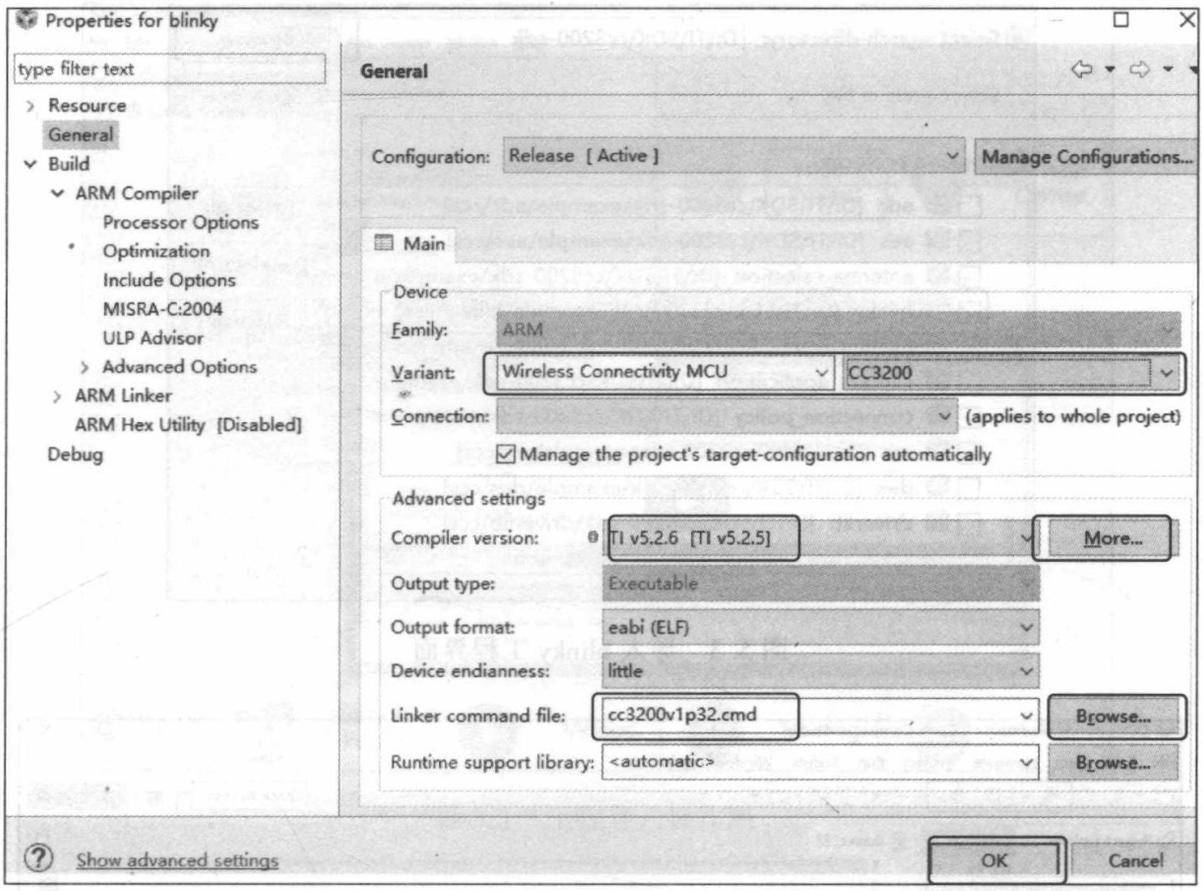


图 5.5 “Properties”选项界面

接下来需要设置“Target”，单击 CCS V6 菜单栏“View”下的“Target Configuration”选项，弹出的“Target”设置界面如图 5.6 所示。

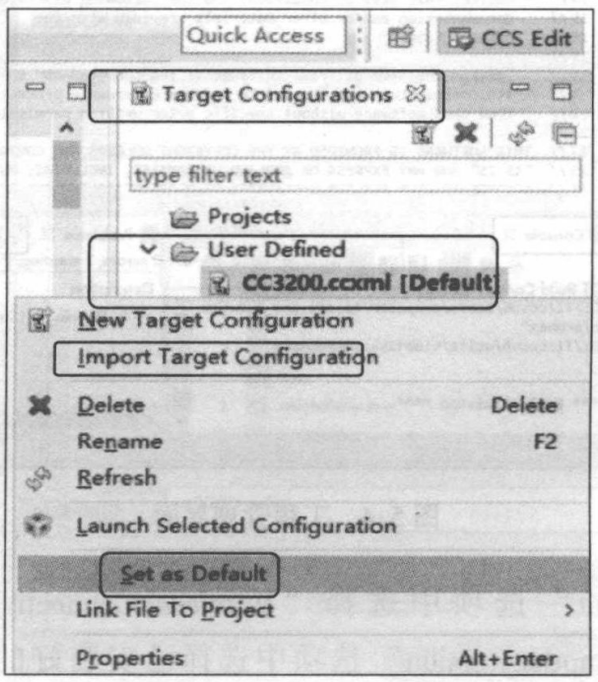


图 5.6 “Target”设置界面

右击图 5.6 所示中的“CC3200. ccxml”，在弹出的窗口中选择“Set as Default”选项，设置成功后，“CC3200. ccxml”会变成粗体，其尾后有“[Default]”字样。如果“CC3200. ccxml”文

件不存在，则右击“User Defined”选择下的“Import Target Configuration”选项，定位路径为 SDK 目录下“cc3200-sdk\tools\ccs_patch”，选择该文件下的“CC3200.ccxml”文件。注意，选择文件后，要单击“Copy files”选项，如图 5.7 所示。

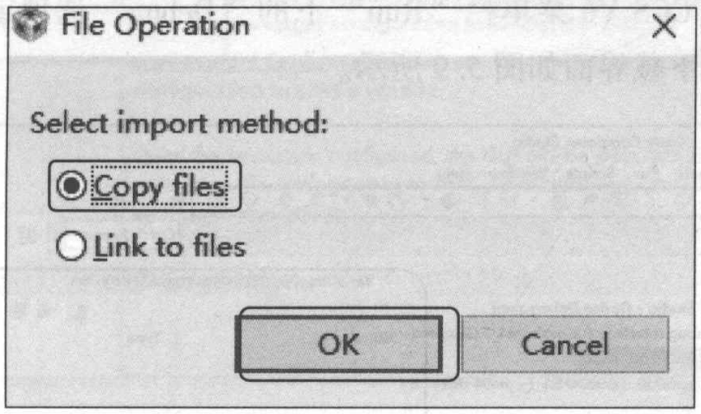


图 5.7 “CC3200.ccxml”导入设置界面

5.1.2 编译与下载调试

上述步骤均完成后，即可进行工程的编译查错。如果工程编译后没有出现错误，则说明应用程序没有语法错误，在编译后会生成可执行文件。在当前工程为有效时，通过单击 CCS V6 菜单栏“Project”下的“Build Project”或单击菜单栏快捷键：图标或使用快捷键 Ctrl+B 可实现编译，初次编译时间稍长。

下载调试时需要连接 CC3200 LaunchPad 至电脑，连接时需注意跳线帽的正确位置，如图 5.8 所示。

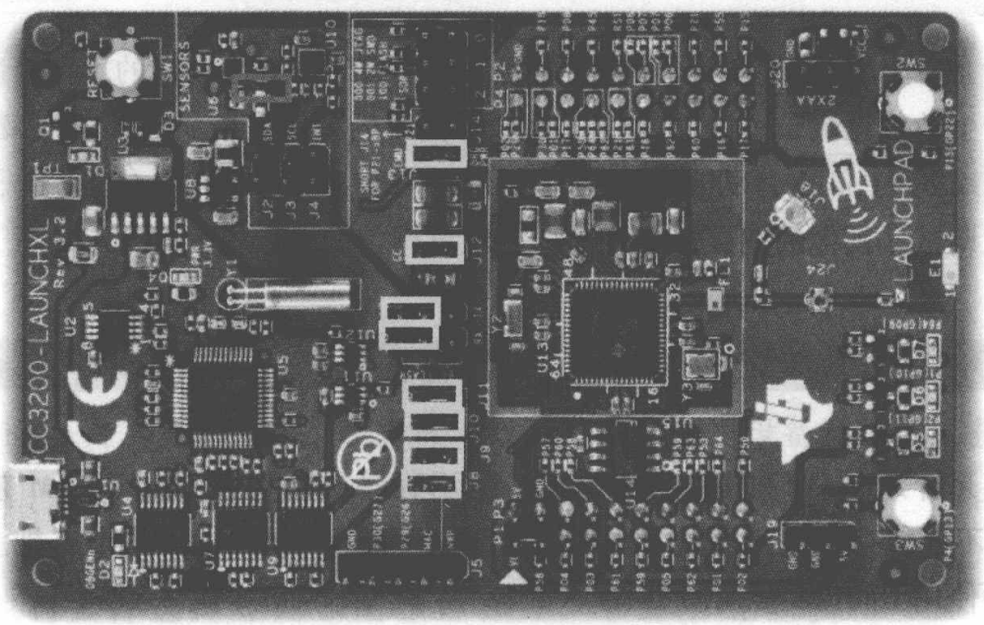


图 5.8 CC3200 LaunchPad 跳线帽的位置图

CC3200 LaunchPad 连接电脑时，红色和黄色的 LED 会亮起。红色 LED（D4）为电源指示灯。黄色 LED（D1）为复位信号。当复位时，黄色 LED 会熄灭，常亮表示为工作状态。

编译通过后，单击 CCS V6 菜单栏“Run”下的“Debug”选项或单击菜单栏快捷键图标进行下载调试。调试下载界面如图 5.9 所示。

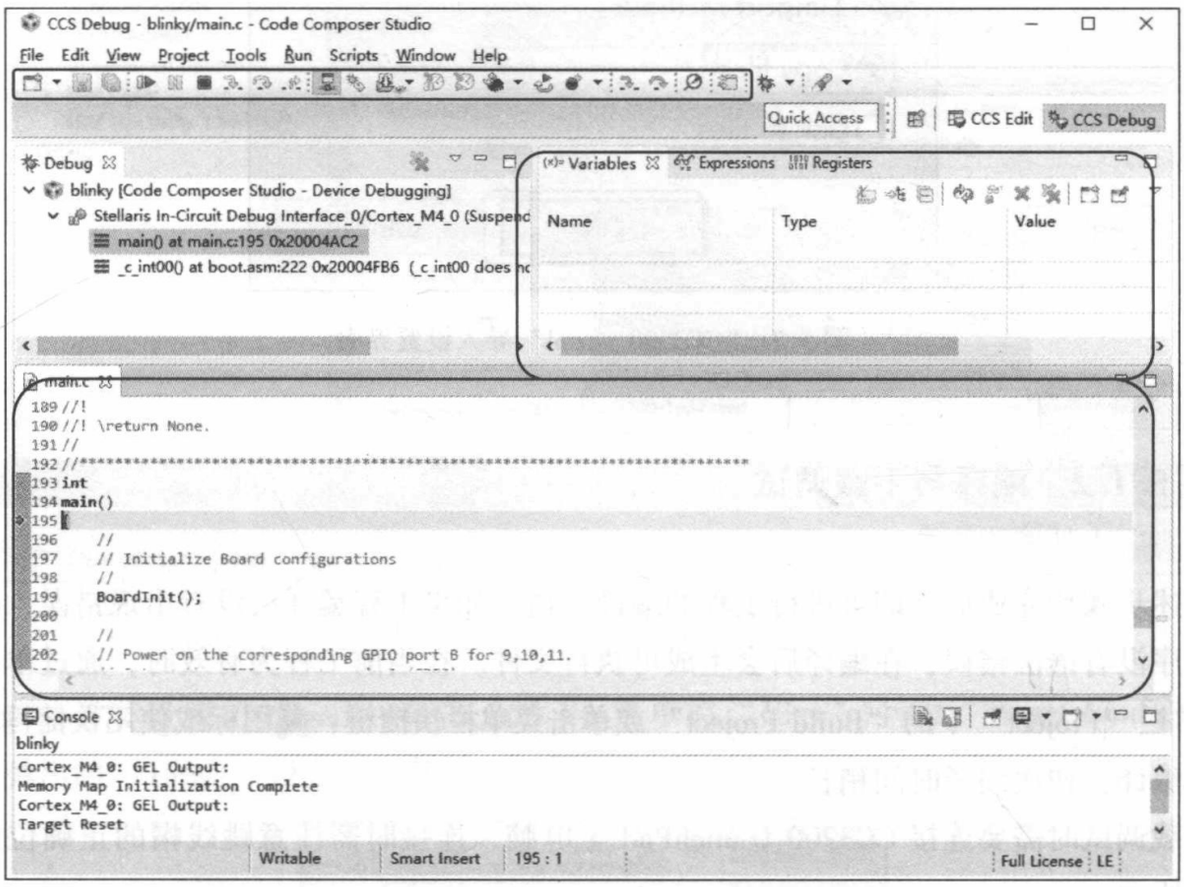


图 5.9　调试下载界面

图 5.9 中，右上方为寄存器和相应变量的值表；中间为程序调试的运行情况；最下方为调试过程中产生的信息记录。用户可以在此查看程序的调试情况。单击快捷键▶图标运行程序。此时，LaunchPad 平台的 3 个 LED 将会依次亮起，表示硬件平台正常。通过单击■图标可结束本次调试。其他的调试快捷键，读者可自行尝试。

5.1.3　Uniflash 程序的烧写

上一节只是介绍了程序的编译与调试，应用程序被暂时加载至 SRAM 中运行，因为 SRAM 掉电后不能保存数据，所以需要将应用程序下载至 Flash 中。本节将把上一节所生成的可执行文件使用 Uniflash 烧写至 flash。

确认 CC3200 LaunchPad 和电脑已经连接，运行 Uniflash，单击菜单栏“File”下的“New Configuration”选项，如图 5.10 所示。

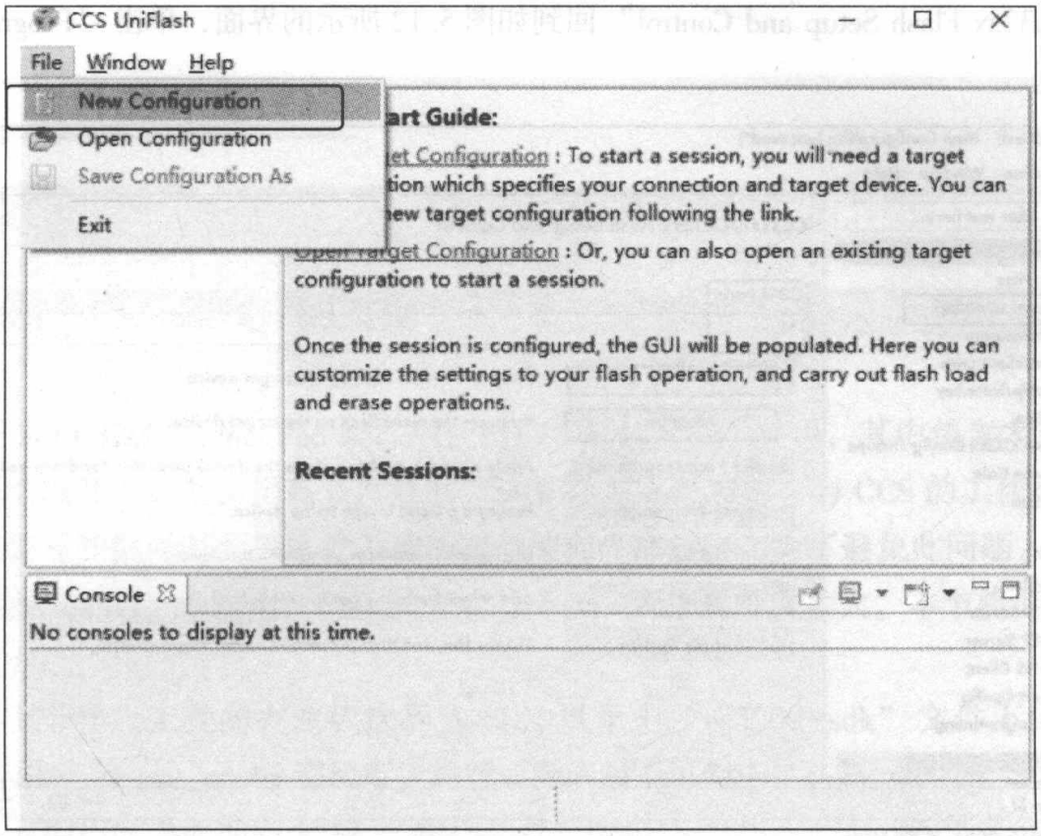


图 5.10　创建新的配置文件

弹出的配置文件器件选择界面如图 5.11 所示。图中，“Connection”选择“CC3x Serial (UART) Interface”，“Board or Device”选择“SimpleLink Wifi CC3100/CC3200”，单击“OK”完成设置。

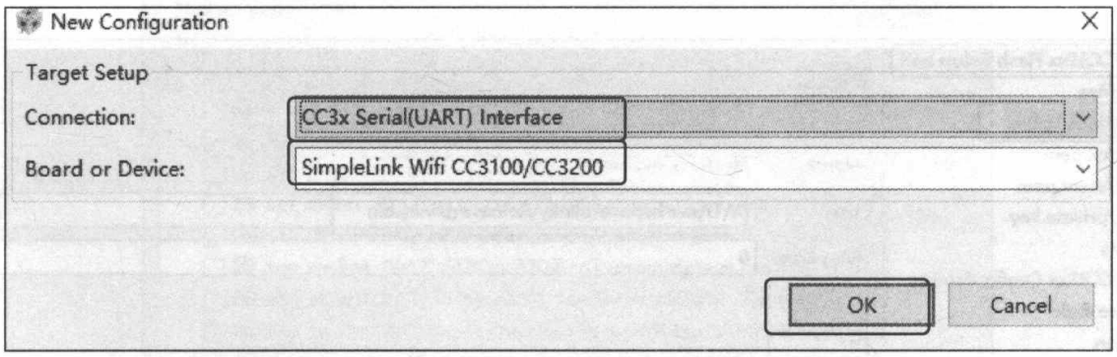


图 5.11　配置文件器件选择界面

弹出的下载参数配置界面如图 5.12 所示。由于 Uniflash 需要知晓 COM 端口号和二进制执行文件（.bin）才可进行下载，因此需要先通过电脑的“设备管理器”获取 COM 端口号，输入至图中的 COM Port 中，然后单击图中的“Format”按钮，将“Capacity”设置为“8MB”。

二进制执行文件选择界面如图 5.13 所示。单击左侧“System Flies”下的“/sys/mcuing.bin”。在“Url”选项中，单击“Browse”按钮，定位文件路径为导入工程“Release”目录下的“blinky.bin”，同时勾选下方的“Erase”“Update”“Verify”三个复选框，然后单击

“CC31xx/CC32xx Flash Setup and Control”回到如图 5.12 所示的界面，单击“Program”按钮完成程序下载。

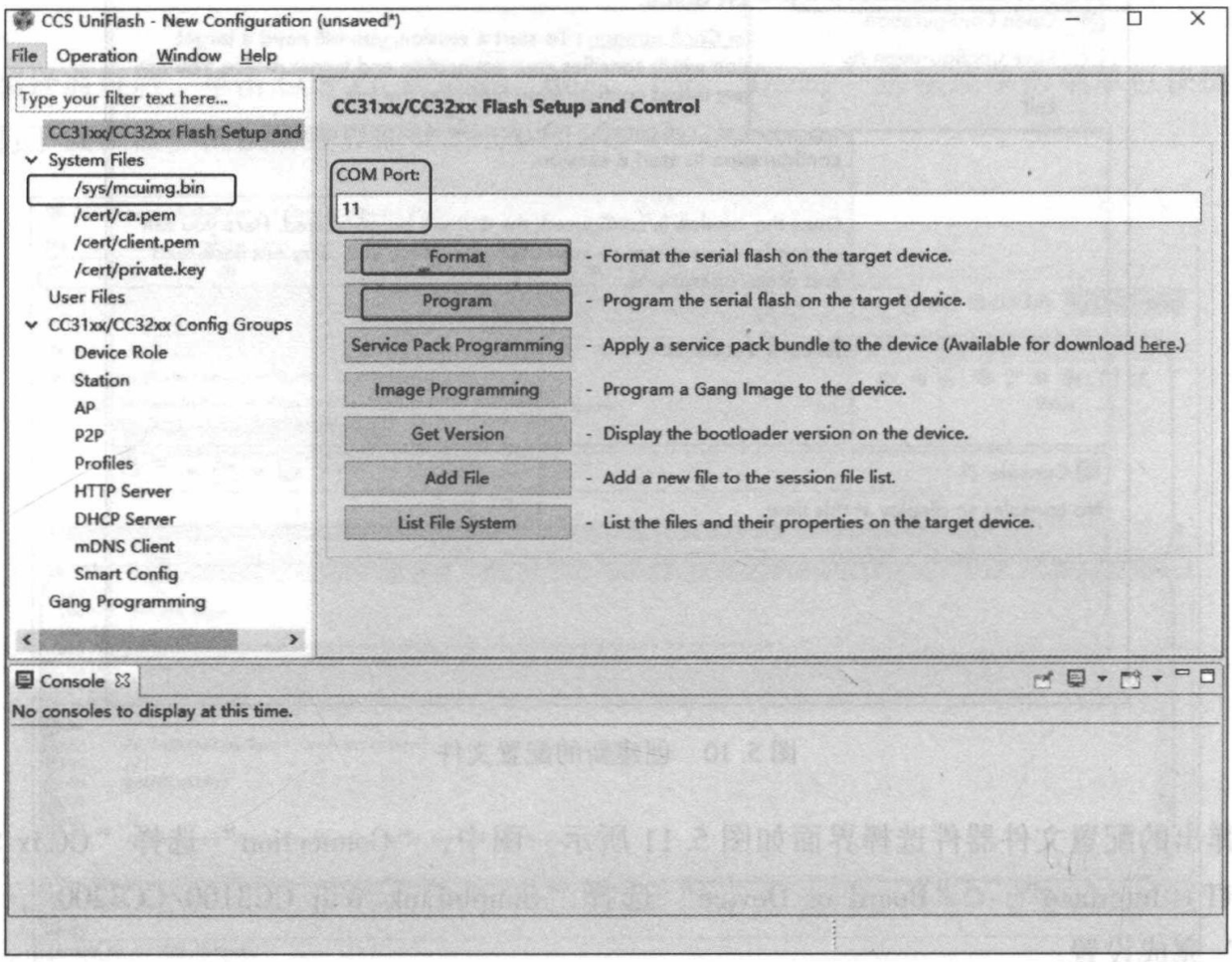


图 5.12　弹出的下载参数配置界面

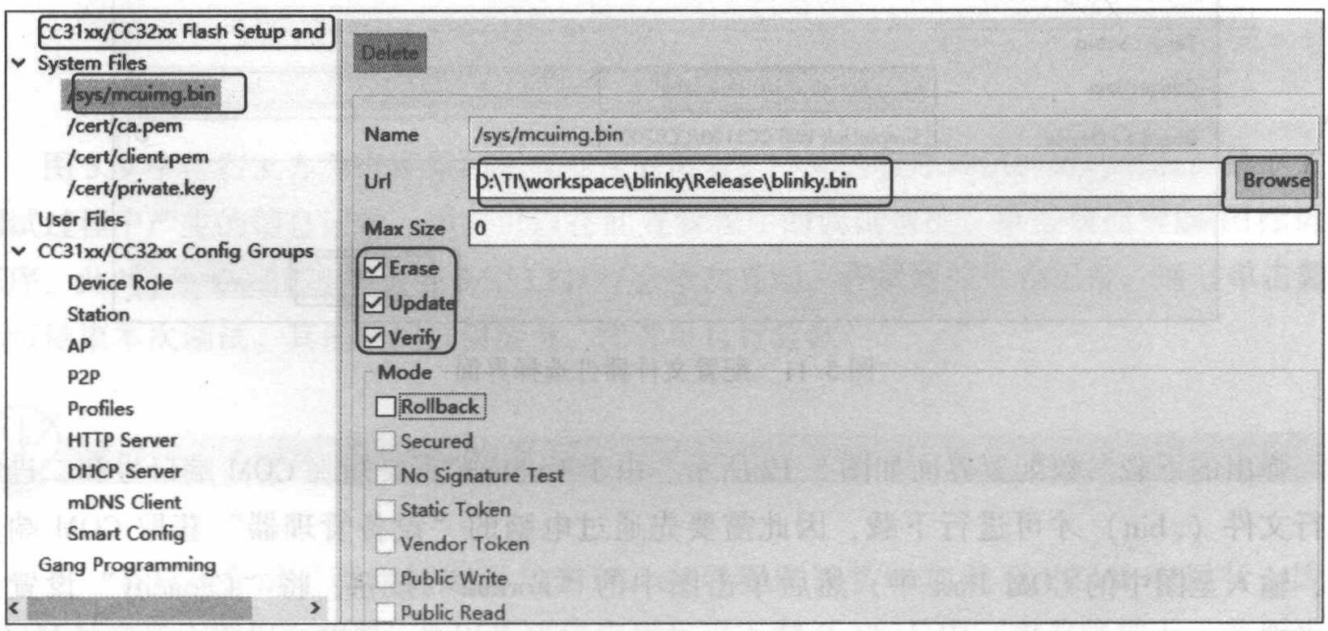


图 5.13　二进制执行文件选择界面

注意，在程序下载至 Flash 中后，需要拔掉图 5.8 中 SOP2 引脚的跳线帽，按下硬件平台复位按键才能够看到程序的运行结果。

5.2 项目的开发过程

5.2.1 CCS 编程库的重建

TI 公司为 CC3200 LaunchPad 硬件平台提供了软件开发工具包，其中包含大量的编程库，可帮助用户缩减开发周期，提高开发效率。SDK 是独立安装的，与 CCS 的工作目录不一致，项目工程在编写和编译时会提示找不到相应的编程库而报错，为了避免此问题，可通过一一添加“Include option”来解决，但这种方法对于初学者来说难度太大。本节将介绍另一种编程库的重建方法。

按照 5.1 节导入工程的方法依次导入 SDK 目录中“cc3200-sdk”文件下的 driverlib、oslib、simplelink、ti_rtos_config 编程库。注意，导入时不可勾中“Copy projects into workspace”复选框。导入编程库界面如图 5.14 所示。

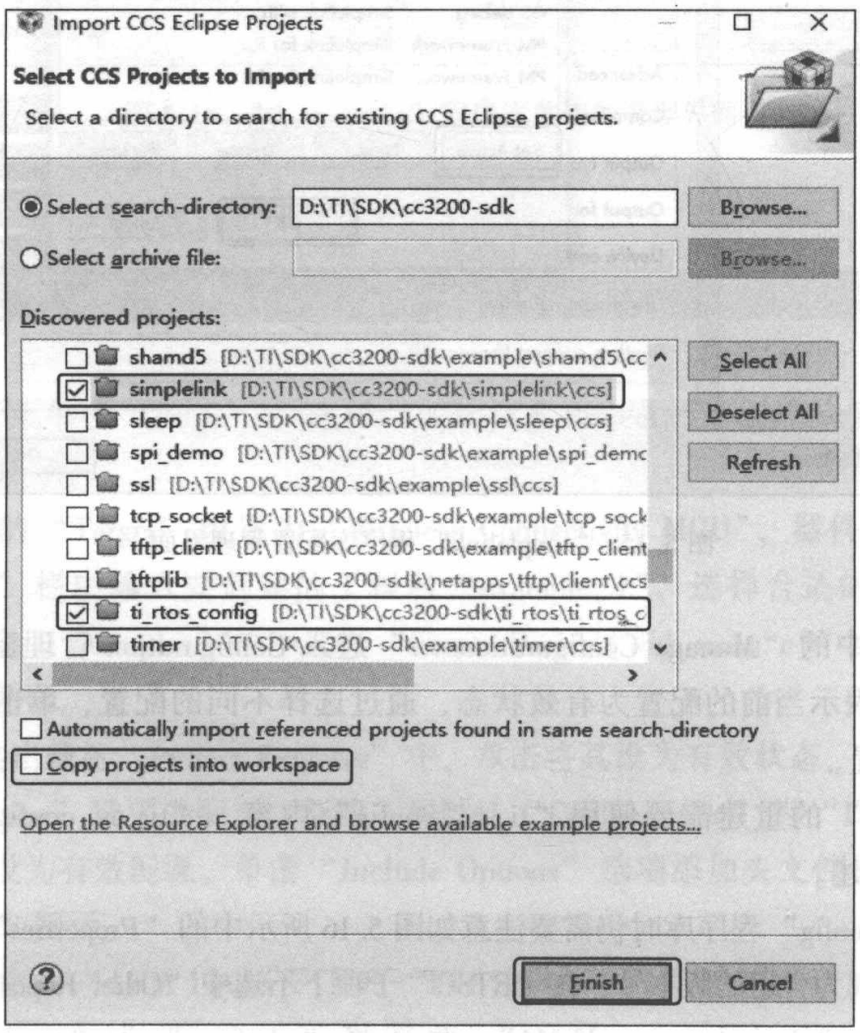


图 5.14 导入编程库界面

添加完成后，对编程库的重建需依次进行，单击编程库“driverlib”选择“Properties”选项，按照图 5.5 所示的方法进行配置，然后右击编程库“driverlib”选择“Build Project”选项完成该编程库的重建。

可按照顺序依次重建“simplelink”“ti_os_config”及“oslib”编程库。值得注意的是，“oslib”编程库有两种不同的配置：free_rtos 和 ti_rtos；“simplelink”编程库有四种不同的配置：NON_OS、OS、NON_OS_PM、PM_Framework，每种配置都需要按照上述步骤进行重建。“simplelink”程序库配置管理界面如图 5.15 所示。

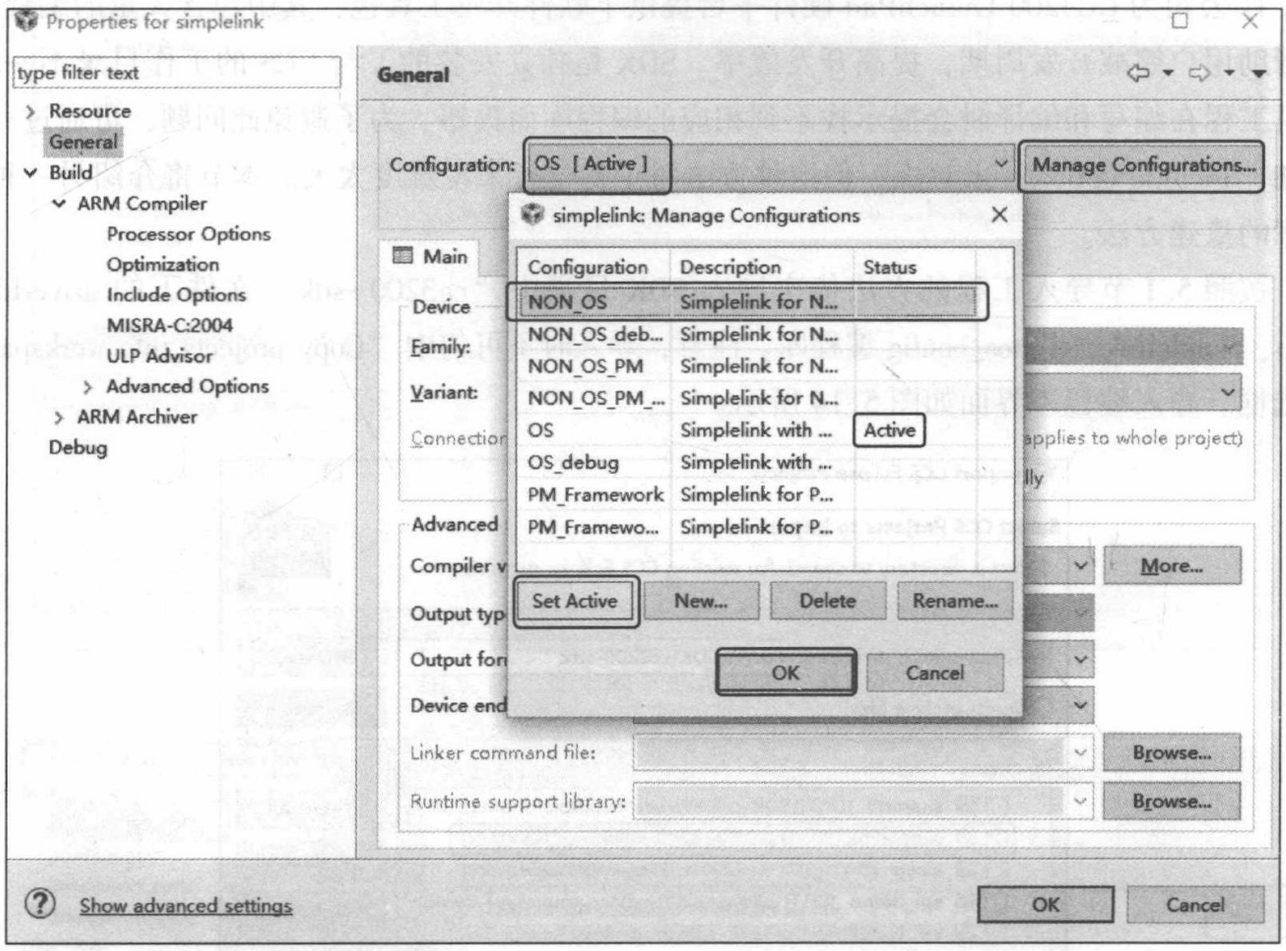


图 5.15 “simplelink”程序库配置管理界面

单击图 5.15 中的“Manage Configurations...”进入 Configuration 管理窗口，在“Status”栏中，“Active”表示当前的配置为有效状态，通过选择不同的配置，单击“Set Active”按钮可更新有效配置。

注意，“oslib”的重建需要使用“ti_os_config”内容，“ti_os_config”程序库要先于“oslib”程序库重建。

重建“ti_os_config”程序库时仍需要注意如图 5.16 所示中的“Properties”选项设置：选择“XDCtools verison”为合适的版本号，在“RTSC”子项下不选中“Other Repositories”下的内容，确认“Target”是“ti.targets.arm.elf.M4”，并选择“platform”为“ti.platforms.simplelink:CC3200”。单击“OK”完成设置后才可进行重建。

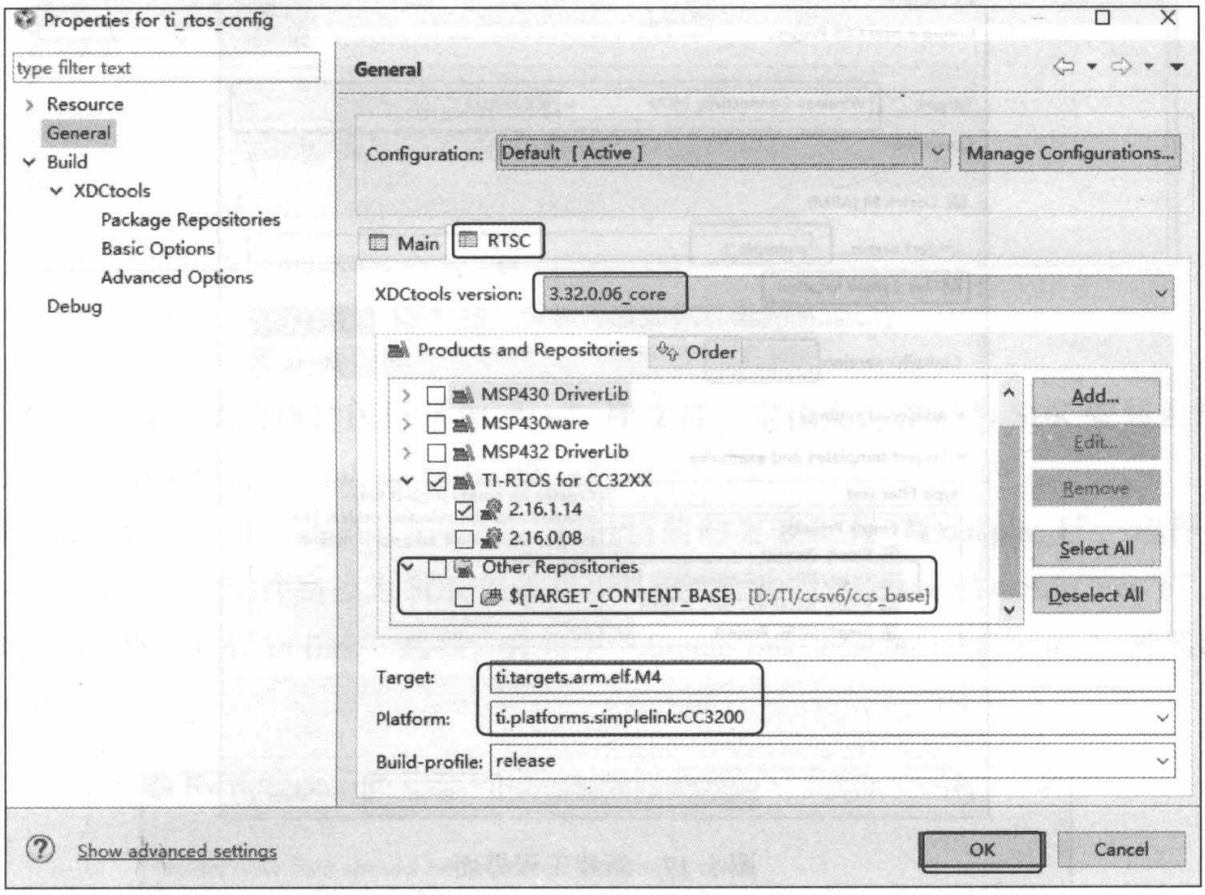


图 5.16 “ti_os_config”程序库重建的管理界面

5.2.2 新建工程

单击 CCS V6 菜单栏“Project”，选择“New CCS Project...”创建一个新工程。新建工程界面如图 5.17 所示。

将图 5.17 中的“Target”设置为“Wireless Connectivity MCU”，器件选择“CC3200”，在“Project name”栏中输入要创建的工程名“example_1”，选择合适的编译器版本，在“Project templates and examples”栏中选择“Empty Project(with main.c)”选项，单击“Finish”完成工程的创建。

新工程目录会出现在“Project Explorer”中，双击将其设为有效状态，右击“example_1”工程进行“Properties”选项的配置。新建工程默认“Configuration”为“Debug”，按前述方法将“Release”设为有效配置。单击“Include Options”选项添加头文件目录。添加头文件目录界面如图 5.18 所示。

单击图 5.18 中的“ARM Linker”下的“File Search Path”选项，添加链接文件目录界面如图 5.19 所示。

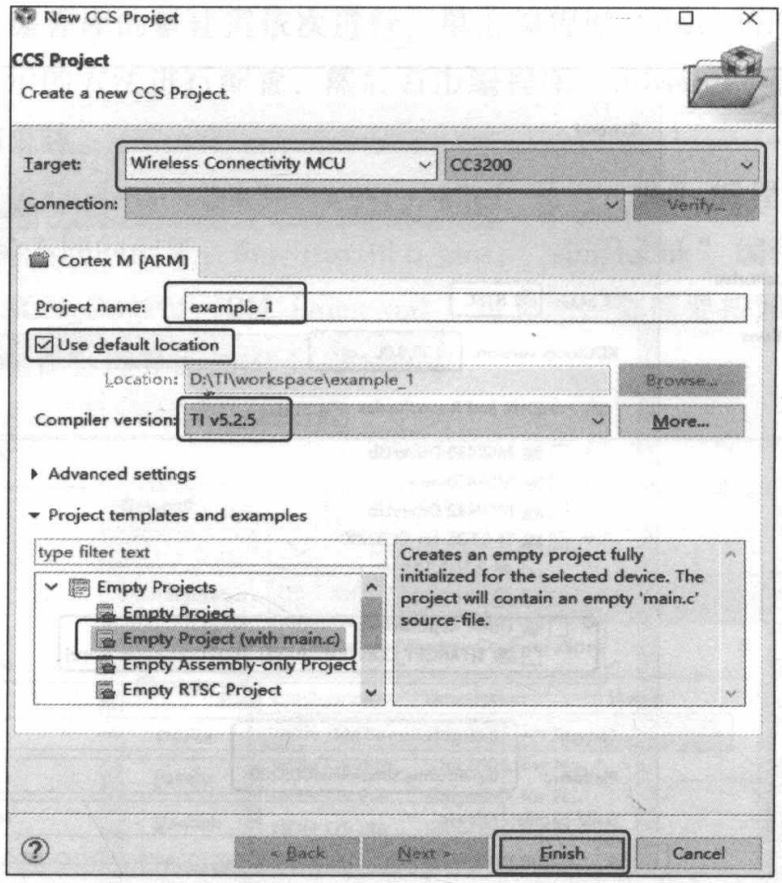


图 5.17　新建工程界面

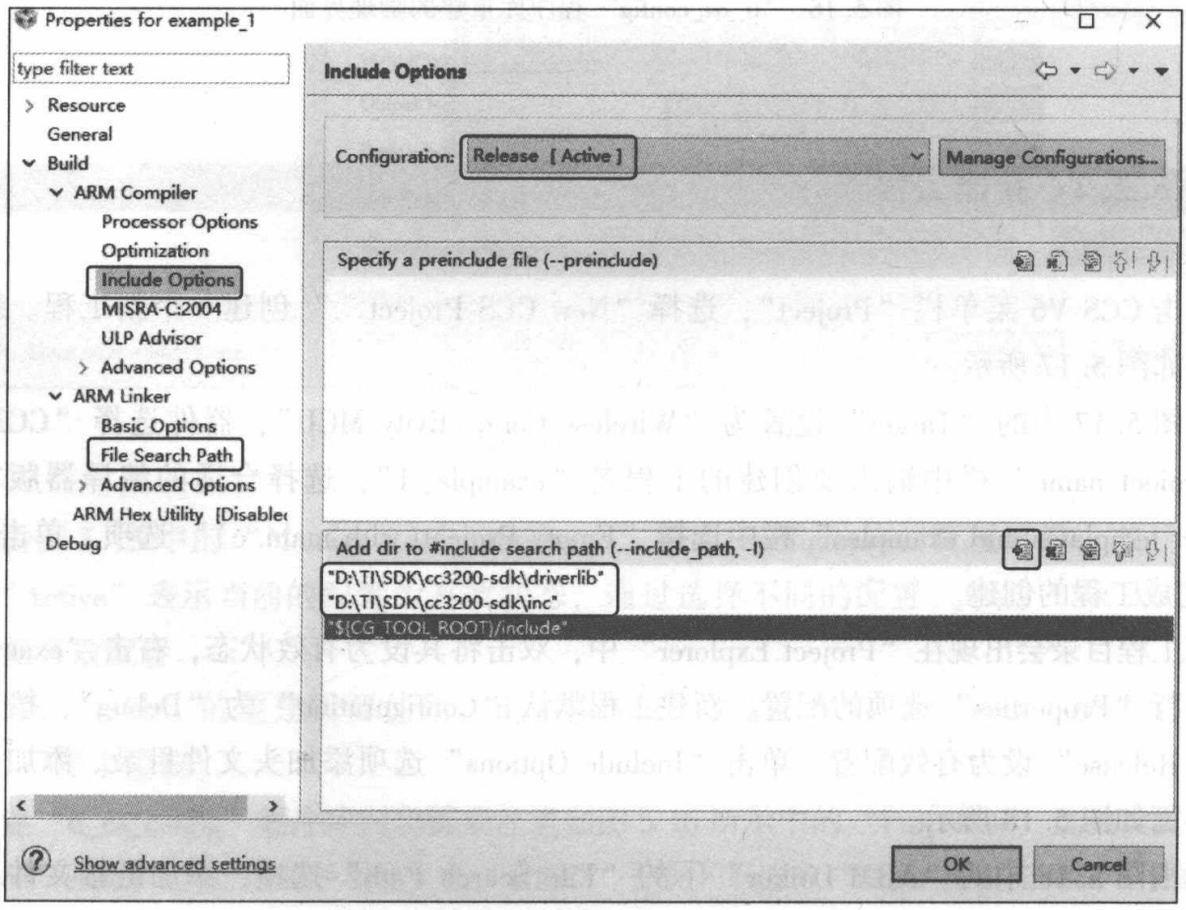


图 5.18　添加头文件目录界面

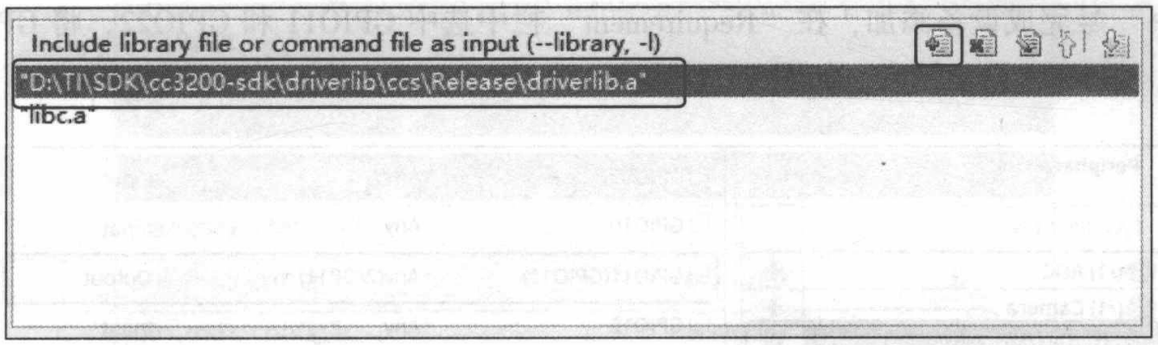


图 5.19　添加链接文件目录界面

单击图 5.19 中的快捷键图标添加“.a”库文件，定位文件路径为 SDK 安装目录下的“cc3200-sdk\driverlib\ccs\Release\driverlib.a”。

添加链接启动文件“startup_ccs.c”：右击当前的有效工程“example_1”，选择“Add Files...”选项，定位文件路径为 SDK 安装目录下的“example\common\startup_ccs.c”，在弹出的窗口中选择“Link to files”选项，并勾中“Create link location relative to:”复选框，如图 5.20 所示。

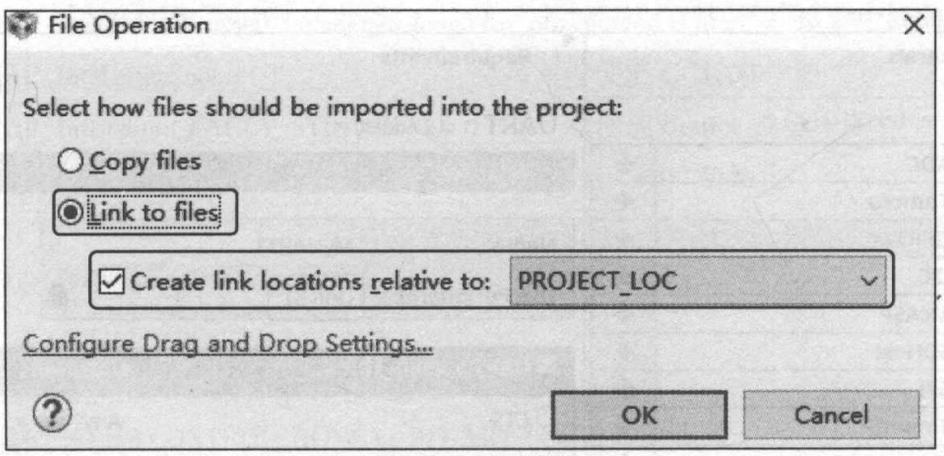


图 5.20　添加链接启动文件“startup_ccs.c”

5.2.3　硬件驱动程序的编写

以按键控制 LED 发光，并通过串口 UART 打印信息的工程为例，该工程涉及的硬件外设包括 GPIO、UART。值得注意的是，GPIO 和 UART 均有多个引脚可选择配置，为了确定所需的引脚，需要根据 CC3200 LaunchPad 实际的硬件连接来确定。用户可从 TI 公司的官网获取 CC3200 LaunchPad 的原理图。

从原理图可知，UART 端口为 GPIO01 和 GPIO02，SW2 按键连接 GPIO22 端口，发光二极管 D5 连接 GPIO11 端口，端口引脚的配置源码通过 PinMux 辅助软件工具产生。

按照 4.2.2 节中的启动方式启动 PinMux，在“Peripherals”栏中选择“GPIO”，单击后

面的“+”号完成设备添加，在“Requirement”栏中选中 GPIO11 和 GPIO22，将 GPIO11 设为输出，GPIO22 设为输入，如图 5.21 所示。

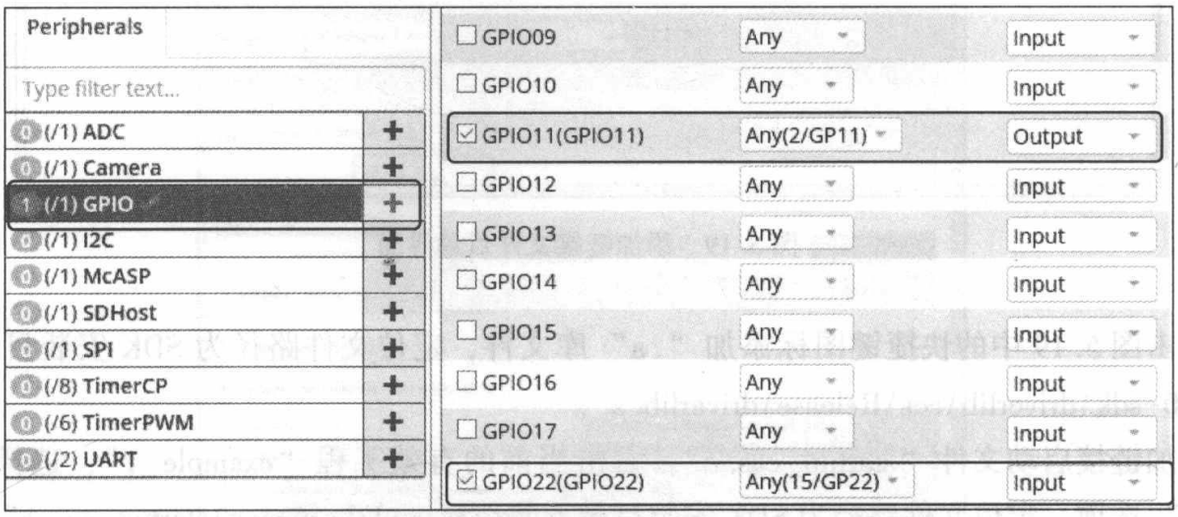


图 5.21　GPIO 的引脚配置

接着添加“UART”，引脚配置如图 5.22 所示。

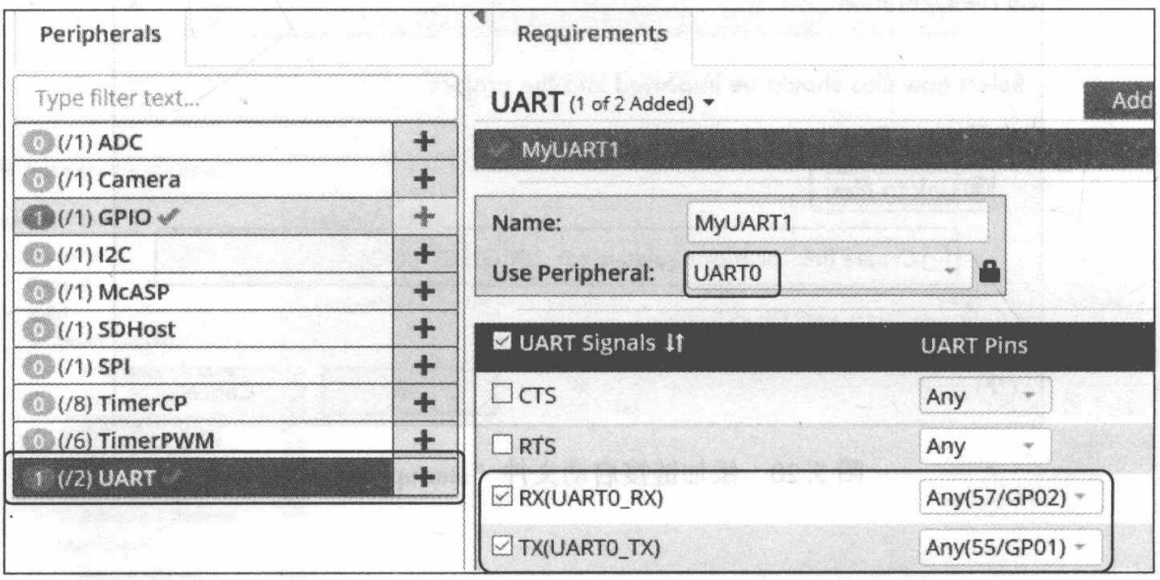


图 5.22　UART 的引脚配置

将生成的源码文件下载至创建的工程目录中。注意，生成的源码文件仅需要放入一个头文件和一个 C 文件。这里选择“rom_pin_mux_config. c”和“pin_mux_config. h”文件，如图 5.23 所示。

在 main. c 文件中编写硬件驱动函数。硬件驱动函数包括 Boardinit、Uart_Init、Uart_StringPuts、Key_Operation、Led_Operation。Boardinit 为 CC3200 LaunchPad 硬件平台的启动初始化函数，是必要的函数，代码格式固定；Uart_Init 为 UART 通信接口的初始化函数；Uart_StringPuts 为 UART 字符串的输出函数；Key_Operation、Led_Operation 分别是按键、LED 的操作函数。其代码如下：

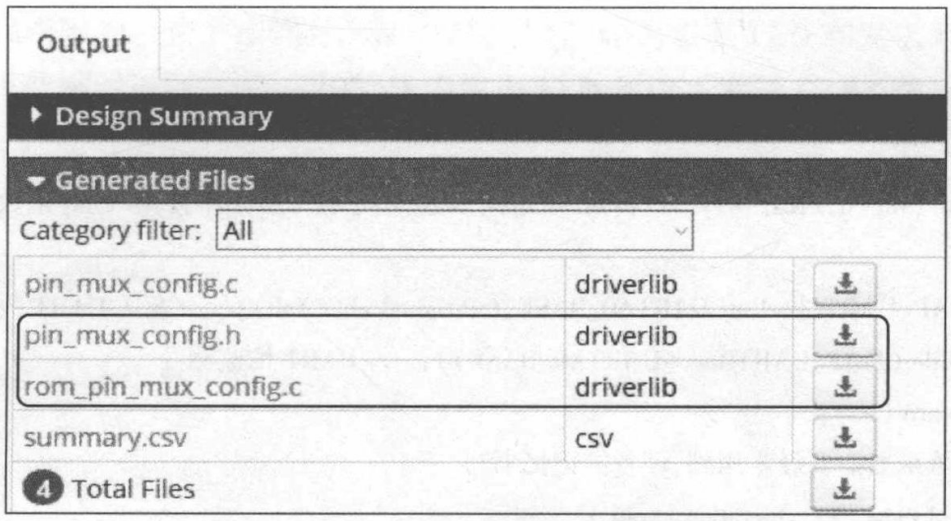


图 5.23　引脚配置源码的下载

```
#define ccs                                          //ccs 宏定义
extern void ( * const g_pfnVectors[ ])(void);        //中断向量表外部引用声明
void Boar dInit(void)
{
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);  //设置中断向量表
    MAP_IntMasterEnable();                           //使能 CC3200 中断
    MAP_IntEnable(FAULT_SYSTICK);                    //使能 Systick 故障中断
    PRCMCC3200MCUInit();                             //CC3200 初始化
}
void UART_Init(unsigned long band)
{
    MAP_UARTFlowControlSet(UARTA0_BASE,  \
    UART_FLOWCONTROL_NONE); //UART 接口流控制设置
    MAP_UARTConfigSetExpClk(UARTA0_BASE,  \
    PRCMPeripheralClockGet (UARTA0_BASE),band,\
    UART_CONFIG_WLEN_8|UART_CONFIG_STOP_ONE|  \
    UART_CONFIG_PAR_NONE);                           //UART 接口配置
}
void UART_StringPuts(const unsigned char * str)
{
    if(str!=NULL)
    {
        while ( * str!='\0')                         //循环输出字符串
        {
            MAP_UARTCharPut(UARTA0_BASE, * (str++)); //单个字符输出
            while (MAP_UARTBusy(UARTA0_BASE));       //UART 忙监测
```

```
        }
    }
}
int fputc (int ch,FILE *f)  //printf 重定向函数,用于在 UART 中使用 printf 函数
{
    MAP_UARTCharPut(UARTA0_BASE,(unsigned char )ch);  //发送 UART 数据
    while (MAP_UARTBusy(UARTA0_BASE));  //UART 忙监测
    return (ch);
}  //该函数需要经常用到,读者需记忆书写
unsigned char Key_Operation (void )
{
    static unsigned char KEY_PutDown=1;          //按键按下标志
    if (KEY_PutDown&&  \
    (MAP_GPIOPinRead(GPIOA2_BASE,GPIO_PIN_6)==GPIO_PIN_6))  //首判按键
    {
        MAP_UtilsDelay(160000);                  //延时去抖
        KEY_PutDown=0;
        if (MAP_GPIOPinRead(GPIOA2_BASE,GPIO_PIN_6)==GPIO_PIN_6)
            return 1;                            //按下返回 1
        else return 0;
    }
    else if (MAP_GPIOPinRead(GPIOA2_BASE,GPIO_PIN_6)!=GPIO_PIN_6))
        //确认是否松开
        KEY_PutDown=1;
    return 0;
}
void Led_Operation (unsigned char value)
{
    if (value)
        MAP_GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_3,GPIO_PIN_3);
    else
        MAP_GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_3,~GPIO_PIN_3);
}
```

5.2.4 应用程序的编写

系统运行需要将 CC3200 和外设进行初始化配置，以提供应用程序运行的环境。本工程

应用程序的实现逻辑为：按下按键时返回“1”，“1”作为点亮 LED 和发送 UART 数据的标志；当没有按下按键时返回“0”，“0”作为熄灭 LED 和停止发送 UART 数据的标志。其代码如下：

```
#include "hw_types.h"
#include "hw_memmap.h"
#include "hw_ints.h"
#include "rom_map.h"
#include "uart.h"
#include "pin_mux_config.h"
#include "interrupt.h"
#include "prcm.h"
#include "utils.h"
#include "gpio.h"
#include "stdio.h"
const unsigned char string[ ] = {"uart testing complete!  \r\n"};
int main(void){
    unsigned char KeyValue=0;
    Boardinit( );
    PinMuxConfig( );
    UART_Init(115200);
    while (1)
    {
        KeyValue=Key_Operation( );
        Led_Operation(KeyValue);
        UtilsDelay (800000);
        if(KeyValue)
            UART_StringPuts(string);
    }
}
```

注意，使用 include 添加头文件时，以“hw_”开头的头文件需要先声明，否则将会出现变量未定义的错误。应用程序编写完成后，可按前面的步骤进行程序编译、下载调试及烧写至 Flash 中运行。

为了验证应用程序运行的正确性，使用 Tera Term 软件打印按下按键后 UART 的传输信息。连接 CC3200 至电脑，打开 Tera Term 辅助软件工具，在如图 5.24 所示的窗口中选择“串口(E)”选项，选中后，“端口(R)”会自动识别已插入的 CC3200 设备。

单击“确定”，进入 Tera Term 工作界面，如图 5.25 所示。

单击图 5.25 中菜单栏“设置(S)”下的“串口 (E) ...”选项，进入 Tera Term 串口配

置界面，如图 5.26 所示。

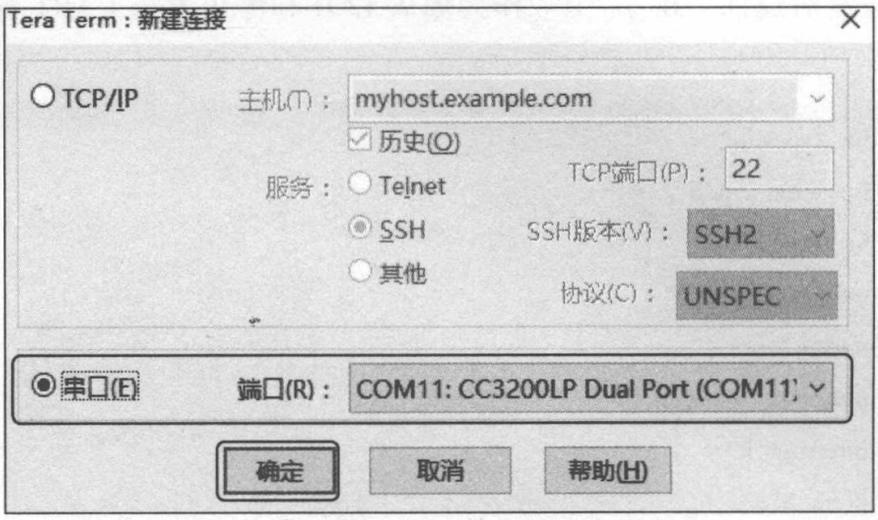


图 5.24　Tera Term 启动界面

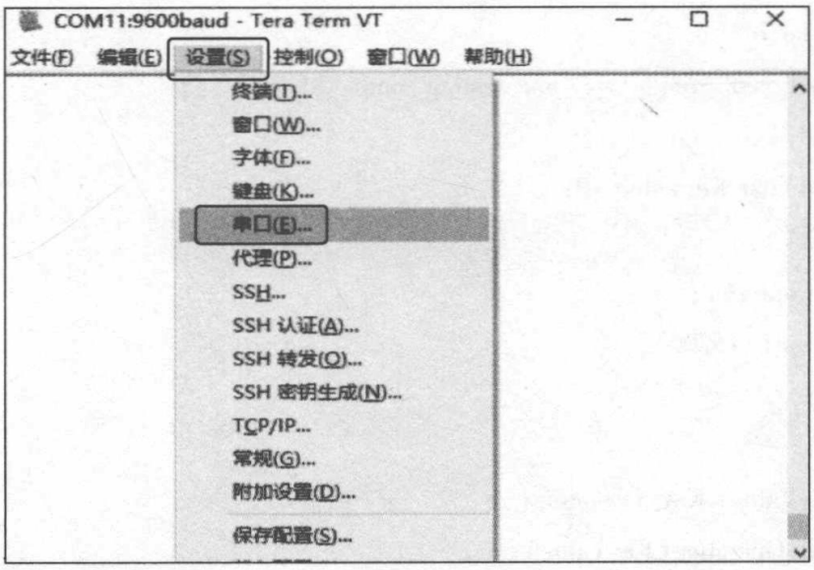


图 5.25　Tera Term 工作界面

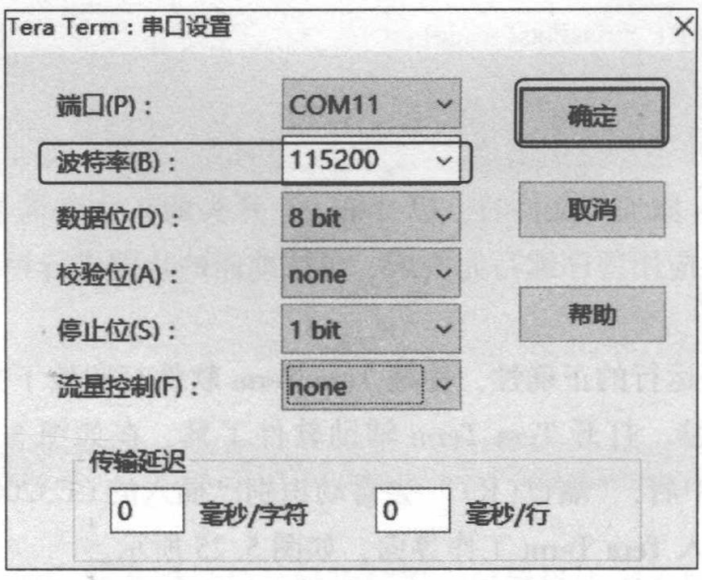


图 5.26　Tera Term 串口配置界面

图 5.26 中选择的波特率为“115200”，其他设置保持不变，单击“确定”，完成 Tera Term 串口的设置。按下 CC3200 LaunchPad 的 SW2 按键后，Tera Term 打印的信息如图 5.27 所示。打印的信息“uart test conplete!”与之前应用程序代码所编写的内容一致，验证了结果的正确性。

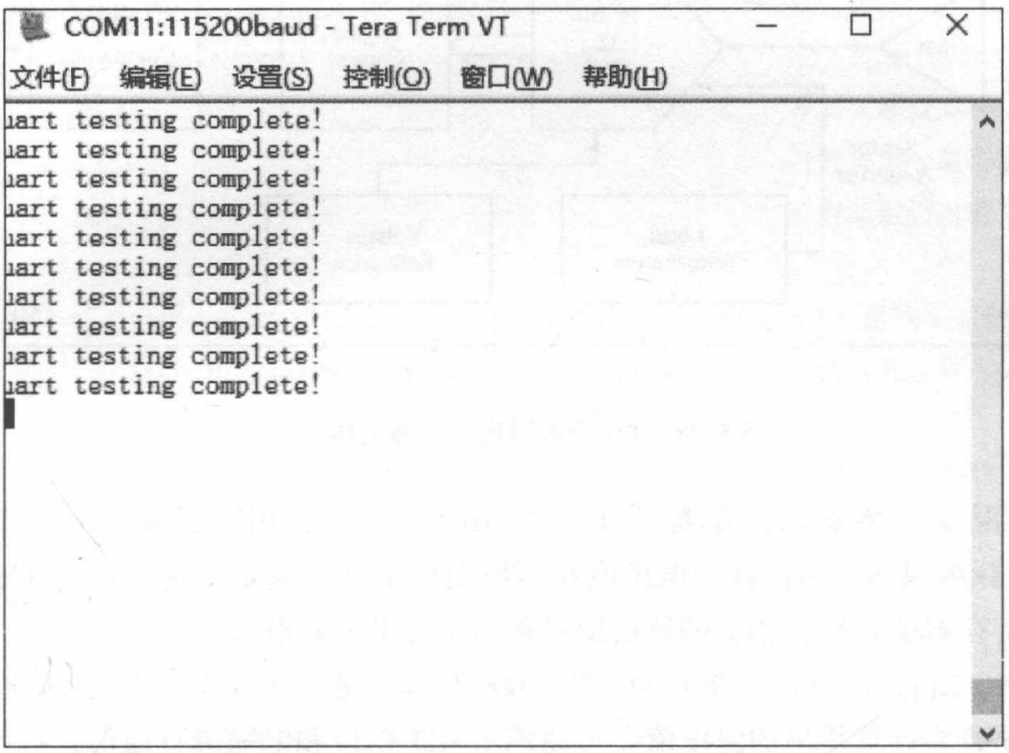


图 5.27 Tera Term 信息的打印

本节通过对项目工程开发流程的详细讲解，使读者可以掌握如何独立开发自己的项目工程。为了方便阅读，后续章节的应用开发均采用本节的工程内容。

5.3 基于 CC3200 的传感器应用

前面介绍了 CC3200 的一个工程开发过程。读者可通过学习 CCS 的使用和项目操作流程独立开发应用。智能家居中的传感器是获取信息的重要外部设备。CC3200 LaunchPad 可通过外部的通信接口实现传感器数据的采集。本节将使用 5.2 节的设计方法对智能家居中常用的传感器进行开发。常用的传感器包括温度传感器、加速度传感器、光强度传感器、湿度传感器、气体传感器、测距传感器及红外热释电传感器等。在实际使用中还包括各式各样的传感器，其开发方法均相同，在此不一一赘述。

5.3.1 板载温度传感器

CC3200 LaunchPad 硬件平台提供了一个板载的温度传感器 TMP006。TMP006 温度传感

器通过检测物体发出的红外能量确定物体的温度，温度测量范围为-40～125℃，工作电流为240μA，最小工作电压为2.2V。TMP006温度传感器的内部结构如图5.28所示。

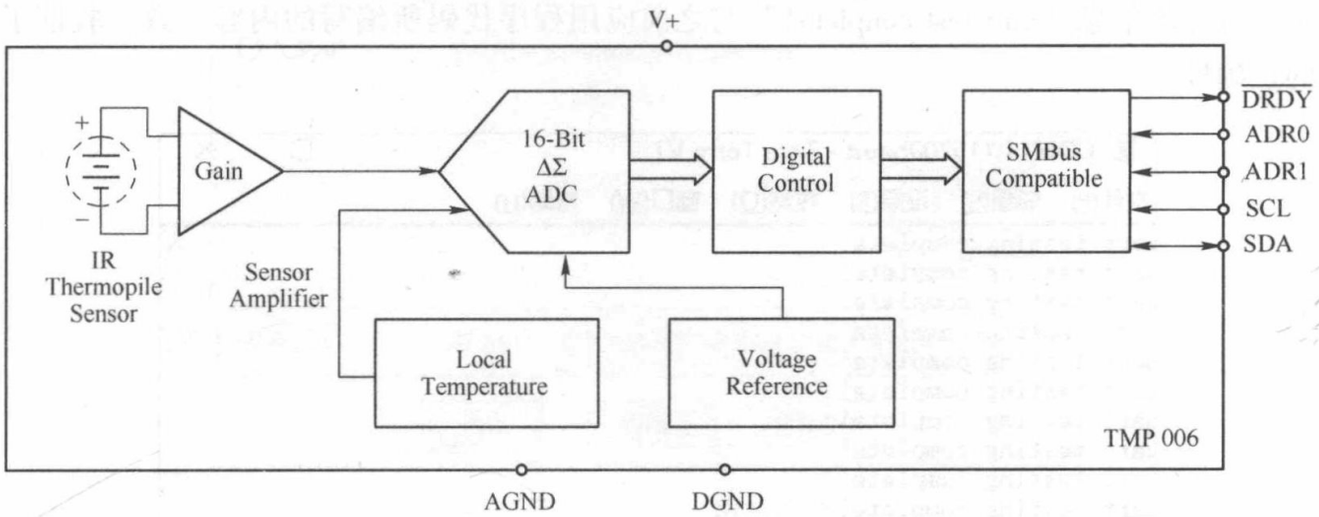


图5.28　TMP006温度传感器的内部结构

TMP006温度传感器的内部集成了一个16位的ADC用于转换电压和温度数据值。TMP006的实际测温结果由内部的电压值和本地温度值共同决定，通过读取TMP006内部的温度、电压寄存器值并结合相应的算法即可算出实际的测量温度。

在CC3200 LaunchPad硬件平台中，TMP006传感器通过I^2C总线和CC3200微控制器连接，为了方便查看计算得出的温度值，可使用UART串口和电脑进行通信，打印出测量的温度值。

使用TMP006传感器进行应用设计时，首先新建example_2项目，并设置好开发环境的配置。驱动TMP006需要配置I^2C、UART外设，按5.2.3节中PinMux的使用方法进行I^2C和UART的引脚配置，I^2C使用GPIO10、GPIO11引脚，UART使用GPIO01、GPIO02引脚。将PinMux生成的引脚配置代码文件“pin_mux_config.h”和“rom_pin_mux_config.c”复制到项目工程目录下。UART的硬件驱动程序可直接使用5.2.3节中的代码程序。I^2C的硬件驱动程序需要编写读/写函数。驱动函数包括IIC_Init、Sensor_WriteCmd、Sensor_ValueRead、UART_Init、UART_StringPuts、BoardInit、ComputeTemperature。其中，IIC_Init函数用来初始化I^2C设备；Sensor_WriteCmd、Sensor_ValueRead用来对传感器进行通信。其代码如下：

```
void IIC_Init (bool lowfast)
{
    MAP_IICMasterInitExpClk(IICA0_BASE,80000000,lowfast);   //IIC 主机配置
    MAP_IICMasterIntClear(IICA0_BASE);                      //关闭 IIC 主机中断
    MAP_IICMasterTimeoutSet(IICA0_BASE,0x7d);               //设置通信超时时间为 20ms
}
void Sensor_WriteCmd (unsigned char addr,unsigned char *cmd,  \
unsigned char datalen,unsigned char stopflag)
{
```

```
    if (datalen>=1&&cmd!=NULL)                          //判读参数输入是否正确
    {
        MAP_IICMasterSlaveAddrSet(IICA0_BASE,addr,false);  //设置从机地址
        MAP_IICMasterDataPut(IICA0_BASE,*cmd);          //放置写入数据
        MAP_IICMasterControl(IICA0_BASE,  \
        IIC_MASTER_CMD_BURST_SEND_START);               //开启 IIC 数据写入
        datalen--;                                      //写入数量减一
        cmd++;                                          //写入下一条命名
        while (datalen)                                 //判断数据是否全部写入
        {
            while (MAP_IICMasterBusy(IICA0_BASE));      //IIC 传输忙监测
            MAP_IICMasterDataPut(IICA0_BASE,*cmd);      //再次放置写入数据
            MAP_IICMasterControl(IICA0_BASE,  \
            IIC_MASTER_CMD_BURST_SEND_CONT);            //IIC 继续数据写入
            datalen--;
            cmd++;
            if (MAP_IICMasterErr(IICA0_BASE)!=IIC_MASTER_ERR_NONE)
                MAP_IICMasterControl(IICA0_BASE,  \
                IIC_MASTER_CMD_BURST_SEND_ERROR_STOP);//错误检测
        }
        if (stopflag==1)  //是否需要写入停止位(写/读连用时需要写入停止位)
            MAP_IICMasterControl(IICA0_BASE,  \
            IIC_MASTER_CMD_BURST_SEND_STOP);            //停止 IIC 写入操作
    }
}
void Sensor_ValueRead (unsigned char addr,unsigned char *readvalue,  \
unsigned char datalen)
{
    if (datalen>=1&&readvalue!=NULL)                    //判读参数输入是否正确
    {
        UtilsDelay (1000000);                           //延时
        MAP_IICMasterSlaveAddrSet(IICA0_BASE,addr,true);  //设置从机地址
        if (datalen==1)                                 //根据读取数据个数判断 IIC 读取模式
            MAP_IICMasterControl(IICA0_BASE,  \
            IIC_MASTER_CMD_SINGLE_RECEIVE);             //IIC 单次数据读取
        else
            MAP_IICMasterControl(IICA0_BASE,  \
            IIC_MASTER_CMD_BURST_RECEIVE_START);        //IIC 多数据读取
```

```
            while (datalen--)                                    //循环读取数据
            {
                while (MAP_IICMasterBusy(IICA0_BASE));           //IIC 传输忙监测
                *(readvalue++) = MAP_IICMasterDataGet(IICA0_BASE);  //数据读取
                if (MAP_IICMasterErr(IICA0_BASE) != IIC_MASTER_ERR_NONE)
                    MAP_IICMasterControl(IICA0_BASE,  \
                    IIC_MASTER_CMD_BURST_RECEIVE_ERROR_STOP);
                MAP_IICMasterControl(IICA0_BASE,  \
                IIC_MASTER_CMD_BURST_RECEIVE_CONT);    //IIC 继续接收
            }
            MAP_IICMasterControl(IICA0_BASE,  \
            IIC_MASTER_CMD_BURST_RECEIVE_FINISH);      //IIC 接收完成
        }
    }
    double ComputeTemperature (double dVobject, double dTAmbient)
    {   //本函数的数据均为规定的数值
        double Tdie2 = dTAmbient + 273.15;
        const double S0 = 6.4E-14;                             // 校准因子
        const double a1 = 1.75E-3;
        const double a2 = -1.678E-5;
        const double b0 = -2.94E-5;
        const double b1 = -5.7E-7;
        const double b2 = 4.63E-9;
        const double c2 = 13.4;
        const double Tref = 298.15;
        double S = S0*(1+a1*(Tdie2 - Tref)+a2*pow((Tdie2 - Tref),2));  //计算算法
        double Vos = b0 + b1*(Tdie2 - Tref) + b2*pow((Tdie2 - Tref),2);
        double fObj = (dVobject - Vos) + c2*pow((dVobject - Vos),2);
        double tObj = pow(pow(Tdie2,4) + (fObj/S),.25);
        tObj = (tObj - 273.15);
        return tObj;
    }
```

TMP006 的驱动程序编写完成后，就可以进行数据的读取和温度计算的应用程序开发。应用程序的主要内容是通过 I^2C 进行数据交换。I^2C 通信的第一帧数据是从机地址。由于从机可以挂载多个，因此为了区分，每一个从机都需要一个地址。从机地址一般可由电气连接来确定。TMP006 温度传感器的从机地址为 0x41。

TMP006 是一款 ASIC 芯片，内部集成的存储器可以存储采集的温度值和电压值。从 TMP006 中使用 I^2C 总线读取温度和电压寄存器的值是应用程序的核心。TMP006 温度传感

器的内部寄存器概况见表 5.1。

表 5.1 TMP006 温度传感器的内部寄存器概况

POINTER (HEX)	REGISTER	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
00H	V_{CBJECT}	V15	V14	V13	V12	V11	V10	V9	V8	V7	V6	V5	V4	V3	V2	V1	V0
01H	$T_{AMBIENT}$	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0	0	0
02H	Configuration	RST	MOD3	MOD2	MOD1	CR3	CR2	CR1	EN	$\overline{DRDY}$	0	0	0	0	0	0	0
FEH	Manufachurer ID	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
FFH	Device ID	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0

表 5.1 中，偏移量 00H 为电压值存储器；01H 为温度值存储器。存储格式为高位在前、低位在后。因为 I^2C 总线一次只能传输一个字节的内容，所以电压值和温度值都需要读取 2 次才能获取完整值。I^2C 总线传输的数据值也是高位在前，其他的存储器，如偏移量 02H 的配置存储器主要用于配置。因为 TMP006 温度传感器初始化默认的配置信息已经满足采集任务的需要，因此没有必要进行修改。偏移量 FEH、FFH 分别为传感器 ID 存储器和生产厂商 ID 存储器，只能读取，不能修改。

由于电压值和温度值的结合才可以算出最终的实际温度值，所以函数 Compute Temperature 包含了计算的算法，编写算法完成后，调用即可。根据以上信息编写的应用程序代码如下：

```
unsigned char Voltage[2] = {0} ,senddata[1] = {0} ,temperature[2] = {0} ;
short TempV = 0, VoltageV = 0;
double TempValue = 0, VoltageValue = 0, Tobj = 0;
BoardInit( ) ;
PinMuxConfig( ) ;
UART_Init(115200) ;
IIC_Init(1) ;
while (1)
{
    senddata[0] = 0x01;                          //设置偏移量
    Sensor_WriteCmd(0x41,&senddata[0],1,0); //发送 IIC 写操作
    Sensor_ValueRead(0x41,&temperature[0],2); //读取温度存储器值
    senddata[0] = 0x00;
    Sensor_WriteCmd(0x41,&senddata[0],1,0);
    Sensor_ValueRead(0x41,&Voltage[0],2);      //读取电压存储器值
    //读取数据处理
    TempV = temperature[0]&0x00ff;               //清除高 8 位数据
    TempV<<=8;                                   //左移 8 位用来合并数据
```

```
    TempV+=temperature[1];                                    //合并数据
    TempV&=0x0000ffff;                                        //清除无效位
    TempValue=(double)TempV/128;                              //温度数据处理
    VoltageV=Voltage[0]&0x00000000ff;
    VoltageV<<=8;
    VoltageV+=Voltage[1];
    VoltageV&=0x0000ffff;
    VoltageValue=(double)VoltageV*0.00000015625;              //电压数据处理
    //compute
    Tobj=ComputeTemperature(VoltageValue,TempValue);          //实际温度计算
    sprintf(string,"the temperature result: %.2lf\r\n",Tobj);
    UART_StringPuts(string);                                  //输出实际温度值
    printf("\r\n");
    UtilsDelay(80000000);                                     //延时 1s
}
```

上述代码只列出了 mian 的函数部分，其他部分并没有给出。这里需要注意，头文件书写位置的错乱也会造成编译错误。应用程序代码书写完成后，需要使用 CCS V6 “Debug”将执行程序下载至 CC3200 的 SRAM 中进行运行，并参照 5.1.2 节的内容完成下载、调试及 Tera Term 打印测试。

5.3.2 板载加速度传感器

板载加速度传感器可安置在防盗门或防盗窗上，用于在夜间进行非法入侵的检测。CC3200 LaunchPad 板载了一款加速度传感器 BMP222。BMP222 加速度传感器可对三个轴的加速度进行独立的配置和运算，内置的中断控制器可在没有 MCU 的情况下实现中断触发。BMP222 加速度传感器的供电电压为 1.2~3.6V，测量范围可配置为±2 g，±4 g，±8 g，±16 g，在低功耗睡眠模式下，最小供电电流仅为 0.7μA。

BMP222 加速度传感器使用前面介绍的方法配置操作模式。该操作模式可通过 BMP222 加速度传感器外部引脚 PS 的状态来决定。BMP222 加速度传感器的电气连接如图 5.29 所示。

BMP222 加速度传感器的工作模式包括通常模式和专用模式。专用模式应用在方向检测、斜坡斜率测量、敲击振动检测等场合。在此模式下，PS 引脚需要为浮空状态，且 BMP222 加速度传感器的内部配置需要固定设置。而在通常模式下，PS 引脚需要接至“VDDIO”或“GND”：接“GND”时，选择 SPI 接口；接“VDDIO”时，选择 I^2C 接口。通常模式下的内部配置可自由设置。图 5.28 中已为用户设置了固定的 I^2C 接口。

驱动 BMP222 加速度传感器时，由于本节仅使用 BMP222 加速度传感器的基本功能，因

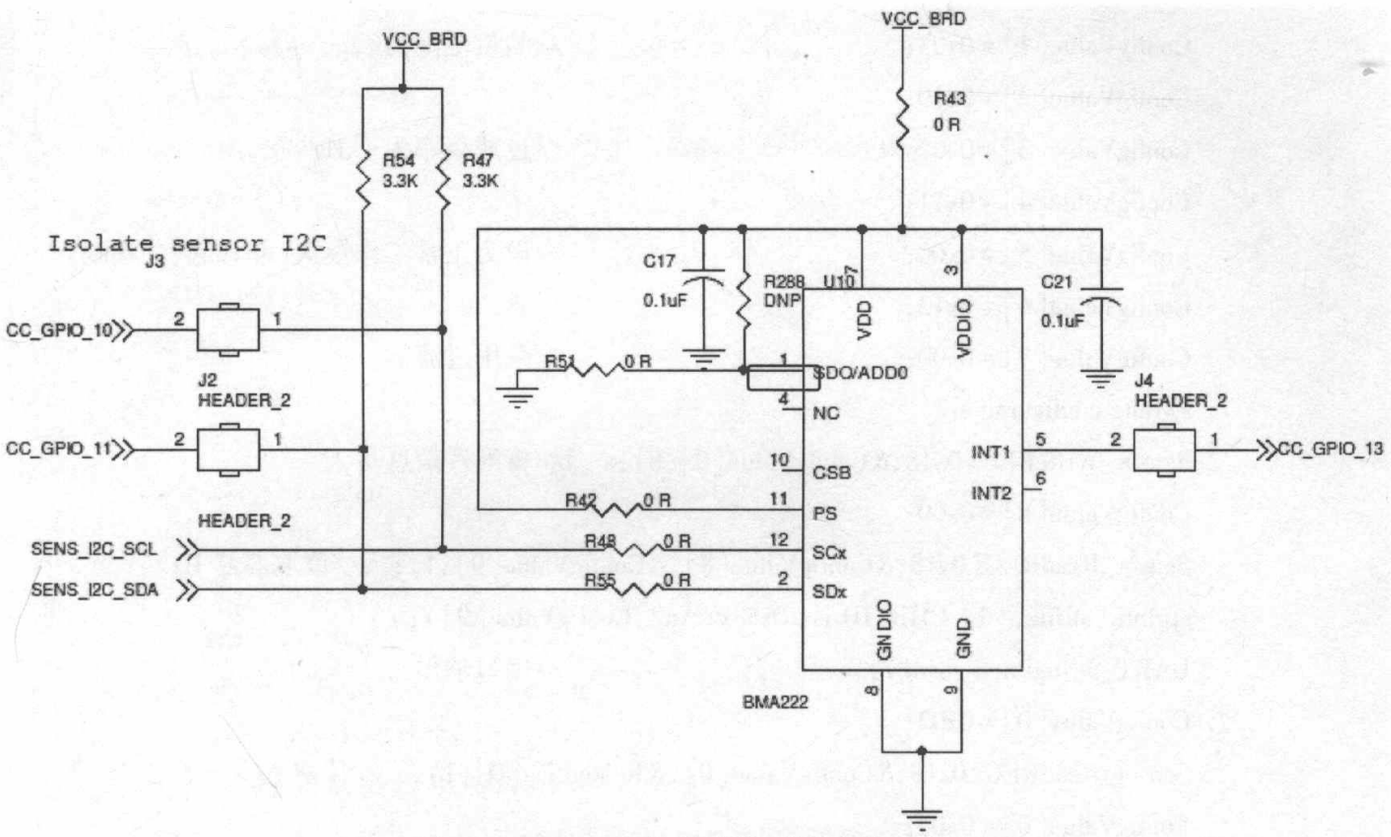


图 5.29　BMP222 加速度传感器的电气连接

此通过 I^2C 接口完成数据交换，使用 UART 打印三个坐标轴的加速度值，配置尽可能简单化。I^2C 和 UART 接口在前面的章节中已完成驱动，可直接应用。由于 BMP222 加速度传感器需要配置初始化程序，因此对 I^2C 接口驱动拓展了读/写寄存器函数：Sensor_WriteREG 和 Sensor_ReadREG，并编写初始化程序驱动 BMP222_Init 和 BMP222_Calibration 函数。其代码如下：

```
void Sensor_WriteREG (unsigned char addr,unsigned char * offsetREG,   \
        unsigned char Sdatalen)
{
    Sensor_WriteCmd(addr,offsetREG,Sdatalen,1);
}
void Sensor_ReadREG (unsigned char addr,unsigned char * offsetREG   \
        ,unsigned char * receivedata,unsigned char Rdatalen)
{
    Sensor_WriteCmd(addr,offsetREG,1,0);
    Sensor_ValueRead(addr,receivedata,Rdatalen);
}
void BMP222_Init (void )
{
    unsigned char ConfigValue[10]={0},ReadyFlag[3]={0};
    ConfigValue[0]=0x0f;
```

```
    ConfigValue[1]=0x03;                          //测量范围为±2g
    ConfigValue[2]=0x10;
    ConfigValue[3]=0x0f;                          //过滤频率为 1kHz
    ConfigValue[4]=0x11;
    ConfigValue[5]=0x00;                          //设为正常工作模式(非挂起、非睡眠)
    ConfigValue[6]=0x13;
    ConfigValue[7]=0x00;                          //采用过滤器
    //rang configuration
    Sensor_WriteREG(0x18,&ConfigValue[0],8);      //将配置信息写入
    ConfigValue[8]=0x00;
    Sensor_ReadREG(0x18,&ConfigValue[8],&ConfigValue[9],1);  //读取芯片 ID
    sprintf(string,"the CHIP ID is: 0x%x\r\n",ConfigValue[9]);
    UART_StringPuts(string);                      //串口输出
    ConfigValue[0]=0x02;
    Sensor_ReadREG(0x18,&ConfigValue[0],&ReadyFlag[0],1);  //读取 New_x 标志
    ConfigValue[0]=0x04;
    Sensor_ReadREG(0x18,&ConfigValue[0],&ReadyFlag[1],1);  //读取 New_y 标志
    ConfigValue[0]=0x06;
    Sensor_ReadREG(0x18,&ConfigValue[0],&ReadyFlag[2],1);  //读取 New_z 标志
    while(!(((ReadyFlag[0]&0x01)==0x01)&&((ReadyFlag[1]&0x01)==0x01)  \
    &&((ReadyFlag[2]&0x01)==0x01)));              //数值更新检测
}
void BMP222_Calibration(void)
{
    unsigned char ConfigValue[7]={0};
    ConfigValue[0]=0x02;
    Sensor_ReadREG(0x18,&ConfigValue[0],&ConfigValue[1],6); //读取加速度值
    ConfigValue[1]=0x38;
    Sensor_WriteREG(0x18,&ConfigValue[1],2);      //写入 x 轴补偿
    ConfigValue[3]=0x39;
    Sensor_WriteREG(0x18,&ConfigValue[3],2);      //写入 y 轴补偿
    ConfigValue[5]=0x3a;
    Sensor_WriteREG(0x18,&ConfigValue[5],2);      //写入 z 轴补偿
}
```

应用程序代码要求 BMP222 加速度传感器开机静止 1s，在 1s 后读取当前三轴的加速度值，为保证加速度值从 0 算起，采用手动补偿的方式进行补偿，对三轴的各个加速度值分别计算后，通过 UART 打印。应用代码如下（仅 main.c）：

```
unsigned char i=0,send[1]={0},receive[6]={0},   \
compensation[3]={0};
char datahandle[3]={0},sign[3]={0},compensign[3]={0};
BoardInit();
PinMuxConfig();
UART_Init(115200);
IIC_Init(1);
BMP222_Init();
UtilsDelay(80000000);
BMP222_Calibration();
while(1)
{
    send[0]=0x02;
    Sensor_ReadREG(0x18,&send[0],receive,6);
    send[0]=0x38;
    Sensor_ReadREG(0x18,&send[0],compensation,3);
    while(!((receive[0]==0x01)&&(receive[2]==0x01)   \
    &&(receive[4]==0x01)));
    for(i=0;i<3;i++){
        if((compensation[i]&0x80)==0x80){
            compensation[i]-=1;
            compensation[i]=~compensation[i];
            compensign[i]=1;
        }else
            compensign[i]=0;
    }
    if(compensign[0]==1)    receive[1]+=compensation[0];
    elsereceive[1]-=compensation[0];
    if(compensign[1]==1)    receive[3]+=compensation[1];
    else    receive[3]-=compensation[1];
    if(compensign[2]==1)    receive[5]+=compensation[2];
    else    receive[5]-=compensation[2];
    for(i=1;i<=5;i=i+2){
        if((receive[i]&0x80)==0x80){
            receive[i]-=1;
            receive[i]=~receive[i];
            datahandle[i/2]=receive[i];
            sign[i/2]=1;
        }
        else{
```

```
            datahandle[i/2]=receive[i];
            sign[i/2]=0;
        }
    }
    if(sign[0]!=1)
        print(string,"x axis acceleration is:%.2fg\r\n",  \
        (float)datahandle[0]*2/128);
    else
        print(string,"x axis acceleration is:-%.2fg\r\n",  \
        (float)datahandle[0]*2/128);
    UART_StringPuts(string);
    if(sign[1]!=1)
        print(string,"y axis acceleration is:%.2fg\r\n",  \
        (float)datahandle[1]*2/128);
    else
        print(string,"y axis acceleration is:-%.2fg\r\n",  \
        (float)datahandle[1]*2/128);
    UART_StringPuts(string);
    if(sign[2]!=1)
        print(string,"z axis acceleration is:%.2fg\r\n",  \
        (float)datahandle[2]*2/128);
    else
        print(string,"z axis acceleration is:-%.2fg\r\n",  \
        (float)datahandle[2]*2/128);
    UART_StringPuts(string);
    printf("\r\n");
    UtilsDelay(10000000);
```

5.3.3 光强度传感器

在智能家居系统中，自动窗帘通过光强度传感器采集的亮度信息来保持室内亮度的稳定。本节使用光强度传感器模块 GY-30 进行设计。GY-30 板载 BH1750FVI 环境光强度传感器的集成芯片，具有接近视觉灵敏度的光谱检测特性，采用标准的光强度单位勒克斯（lx），可对大多数光源进行检测。BH1750FVI 支持 I^2C 总线接口，从机地址由电气连接决定，共有两个地址可以选择：0x46 和 0xb8。BH1750FVI 集成芯片的数据交换使用 I^2C 接口来完成，读取测量结果按照严格的 I^2C 时序来完成。BH1750FVI 的测量过程如图 5.30 所示。

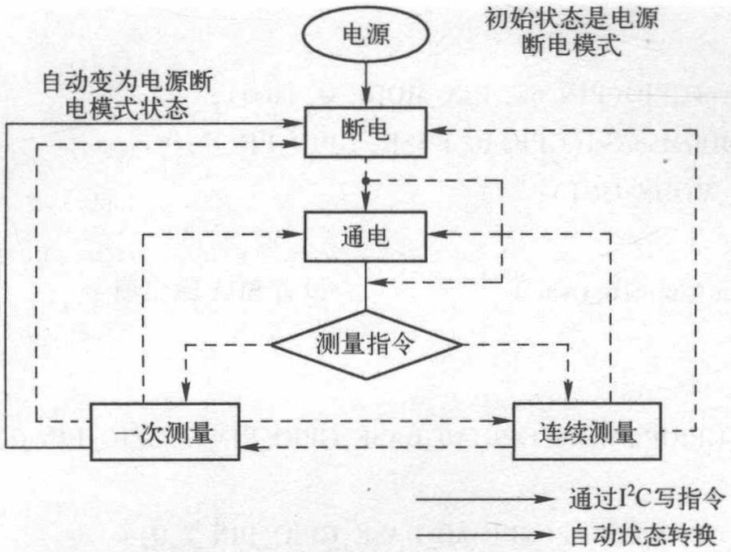


图 5.30　BH1750FVI 的测量过程

BH1750FVI 上电后处于断电模式，为开启光强度数据采集需要发送通电 I^2C 指令(0x01)，再次发送测量分辨率设置指令后，即可进行一次测量或者连续测量。一次测量和连续测量都可以通过循环发送通电 I^2C 指令、设置测量分辨率指令来实现不间断的数据获取。光强度数据采集完毕后就可进行数据的读取。

采集完成的数据通过 UART 和 Tera Term 辅助软件工具显示。本工程开发涉及 UART 外设和 I^2C 外设，为了简单化操作过程，使用模拟 I^2C 接口，仅发送 1lx 的测量分辨率指令 0x10 来完成应用。模拟 I^2C 接口需要编写相关的驱动函数，包括 IIC_Init、IIC_Start、IIC_Stop、IIC_SendAck、IIC_SendNAck、IICSendByte 及 IIC_ReceiveByte。驱动代码如下：

```
void IIC_Init ()  //模拟 IIC 初始化函数
{
    MAP_PRCMPeripheralClkEnable(PRCM_GPIOA0, PRCM_RUN_MODE_CLK);
    MAP_PinTypeGPIO(PIN_62, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, GPIO_PIN_7, \
    GPIO_DIR_MODE_OUT);                    //设置 PIN62 为 SDA
    MAP_PinTypeGPIO(PIN_61, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, GPIO_PIN_6, \
    GPIO_DIR_MODE_OUT);                    //设置 PIN61 为 SDA
}
void SDA_Input_Mode ()                     //配置 SDA 引脚输入模式
{
    MAP_PinTypeGPIO(PIN_62, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, GPIO_PIN_7, GPIO_DIR_MODE_IN);
}
void SDA_Output_Mode ()                    //配置 SDA 引脚出模式
```

```
{
    MAP_PinTypeGPIO(PIN_62, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, GPIO_PIN_7, \
    GPIO_DIR_MODE_OUT);
}
void SDA_Output ( uint16_t val )              //设置 SDA 输出值
{
    if ( val ) {
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,GPIO_PIN_7);
    }else {
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,0);
    }
}
void SCL_Output ( uint16_t val )              //设置 SCL 输出值
{
    if ( val ) {
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_6,GPIO_PIN_6);
    }else {
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_6,0);
    }
}
uint8_t SDA_Input ( )                         //获取 SDA 输出值
{
    if (MAP_GPIOPinRead(GPIOA0_BASE,GPIO_PIN_7)= =GPIO_PIN_7)
        return 1;
    else
        return 0;
}
void delay1(unsignedint n)                    //延时函数
{
    unsignedint i;
    for ( i=0;i<n;++i);
}
void IICStart (void )                         //IIC 启动信号
{
    SDA_Output(1);delay1(500);
    SCL_Output(1);delay1(500);
    SDA_Output(0);delay1(500);
    SCL_Output(0);delay1(500);
}
```

```
void IICStop ( void )                          //IIC 停止信号
{
    SCL_Output(0); delay1(500);
    SDA_Output(0); delay1(500);
    SCL_Output(1); delay1(500);
    SDA_Output(1); delay1(500);
}
unsigned char IICWaitAck ( void )              //等待从机应答
{
    unsigned short cErrTime = 5;
    SDA_Input_Mode();delay1(500);
    SCL_Output(1);delay1(500);
    while (SDA_Input())
    {
        cErrTime--;delay1(500);
        if (0 == cErrTime)
        {
            SDA_Output_Mode();
            IICStop();
            return FAILURE;
        }
    }
    SDA_Output_Mode();
    SCL_Output(0);delay1(500);
    return SUCCESS;
}
void IICSendAck ( void )                       //发送应答信号
{
    SDA_Output(0);delay1(500);
    delay1(500);
    SCL_Output(1);delay1(500);
    SCL_Output(0);delay1(500);
}
void IICSendNotAck ( void )                    //发送非应答信号
{
    SDA_Output(1);delay1(500);
    SCL_Output(1);delay1(500);
    SCL_Output(0);delay1(500);
}
```

```
void IICSendByte (unsigned char cSendByte)   //发送 IIC 数据
{
    unsigned char   i = 8;
    while (i--)
    {
        SCL_Output(0);delay1(500);
        SDA_Output(cSendByte & 0x80);delay1(500);
        cSendByte += cSendByte;delay1(500);
        SCL_Output(1);delay1(500);
    }
    SCL_Output(0);delay1(500);
}
unsigned char IICReceiveByte (void )          //接收 IIC 数据
{
    unsigned char i = 8;
    unsigned char cR_Byte = 0;
    SDA_Input_Mode();
    while (i--)
    {
        cR_Byte += cR_Byte;
        SCL_Output(0);delay1(500);
        delay1(500);
        SCL_Output(1);delay1(500);
        cR_Byte |=   SDA_Input();
    }
    SCL_Output(0);delay1(500);
    SDA_Output_Mode();
    return cR_Byte;
}
```

模拟 I^2C 接口代码仅列出 I^2C 通信所需要的必要操作步骤，并没有形成通信时序，还不能进行通信。CC_Sensor_Write 和 CC_Sensor_Read 函数为按照 BH175FVI 芯片数据通信时序而编写的通信函数。其源码如下：

```
unsigned char CC_Sensor_Write (unsigned char addr,unsigned char cmd)
{  //写 BH1750FVI 指令时序函数
    IICStart();                          //启动 IIC
    IICSendByte(addr);                   //输入从机地址写信号
    if (IICWaitAck()!=1) return 0;       //检测应答信号是否到达
```

```
    IICSendByte(cmd);  //输入指令 cmd
    if (IICWaitAck()!=1) return 0;
    IICStop();  //停止该传输帧
    return 1;
}
unsigned char CC_Sensor_Read (unsigned char addr,unsigned char * receive,\
unsigned char datalen)  //读 BH1750FVI 采集数据时序函数
{
    IICStart();  //启动 IIC
    IICSendByte(addr+1);  //输入从机地址读信号
    if (IICWaitAck()!=1) return 0;
    while (datalen--)
    {
        *(receive++)=IICReceiveByte();  //获取采集数据
        if (datalen==1)
            IICSendNotAck();  //发送应答信号
        else
            IICSendAck();  //发送非应答信号
    }
    IICStop();  //停止该传输帧
    return 1;
}
```

编写应用代码的主要任务是初始化 BH1750FVI，通过相应的 I^2C 时序函数来获取 BH1750FVI 光强度传感器的数据，并将采集到的数据处理后，通过 UART 串口打印显示。应用代码编写如下：

```
void BH1750FVI_Init (void){
    CC_Sensor_Write(0x46,0x01);              //写入 BH1750FVI 通电指令
}
unsigned short BH1750FVI_GetValue (void){
    unsigned short LightValue=0;
    unsigned char receive[2]={0};
    CC_Sensor_Write(0x46,0x01);              //再次写入 BH1750FVI 通电指令
    CC_Sensor_Write(0x46,0x10);              //写入 BH1750FVI 测量分辨率指令
    UtilsDelay (16000000);
    CC_Sensor_Read(0x46,&receive[0],2);      //读取 BH1750FVI 测量数值
    LightValue=(unsigned short )receive[0]<<8;
    LightValue+=receive[1];                  //读取数值为 16 位,合并处理
```

```
    return (LightValue);                          //返回读取到的值
}
int main (void) {
    unsigned short ReceiveData=0;
    float LightValue=0;
    BoardInit();
    PinMuxConfig();
    IIC_Init();
    UART_Init(115200);
    BH1750FVI_Init();
    while (1)
    {
        ReceiveData=BH1750FVI_GetValue();
        LightValue=(float)ReceiveData/1.2;      //将采集数据处理转化为勒克斯值
        sprintf((char *)string,"light intensity is:%.2flx\r\n",LightValue);  //串口打印
        UART_StringPuts(string);
        printf("\r\n");
        UtilsDelay(20000000);
}
```

上述代码仅列出必要代码，UART 的代码可参考前面的章节内容，在此不再赘述。读者可按已经讲述的方法进行仿真、下载及调试，使用 Uniflash 下载至 flash 中运行，通过打开 Tera Term 辅助软件工具显示采集到的光强度数据。

5.3.4 湿度传感器

湿度传感器 DHT11 能够测量室内的湿度值。在智能家居应用中，湿度值的多少将决定是否开启空调器除湿或加湿来保持室内湿度的均衡。DHT11 内含湿度校准，具有精确的相对湿度采集值，不需要额外的部件就能够实现长达 20m 的单总线串行数据传输，湿度值的测量范围为 20%~90%。DHT11 采用单总线串行口通信；需要接入阻值为 5kΩ 的上拉电阻；从单总线读取的数据分为整数部分和小数部分；小数部分只是预留的后续增强功能，并没有实际的数据值；一次完整的数据传输为 40 位；通信时间为 4ms 左右。图 5.31 为 DHT11 单总线通信时序图。

在图 5.31 中，单总线通信引脚为双向通信，DHT11 作为从机设备接入，主机（CC3200）在与 DHT11 通信时，首先需要发送一个开始信号并等待 DHT11 设备应答，应答完毕后，就自动传送采集的湿度值。DHT11 在收到主机设备的开始信号后才会进行一次湿度采集，其他时间处于低功耗状态。注意，开始信号需将通信引脚拉低至少 18ms 后再拉高

20~40μs。DHT11 通过将拉高的通信引脚再次拉低至少 80μs 来完成开始信号的应答。DHT11 开始信号时序图如图 5.32 所示。

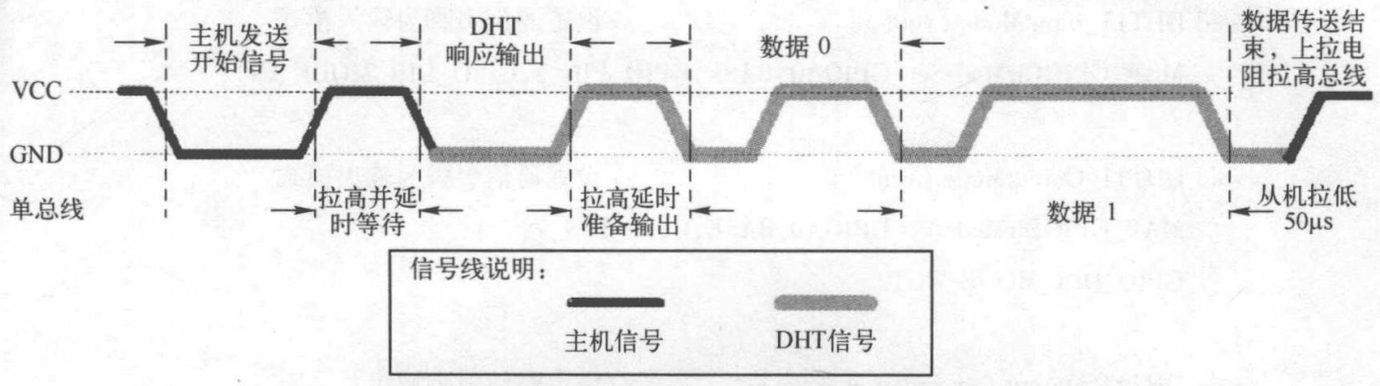


图 5.31　DHT11 单总线通信时序图

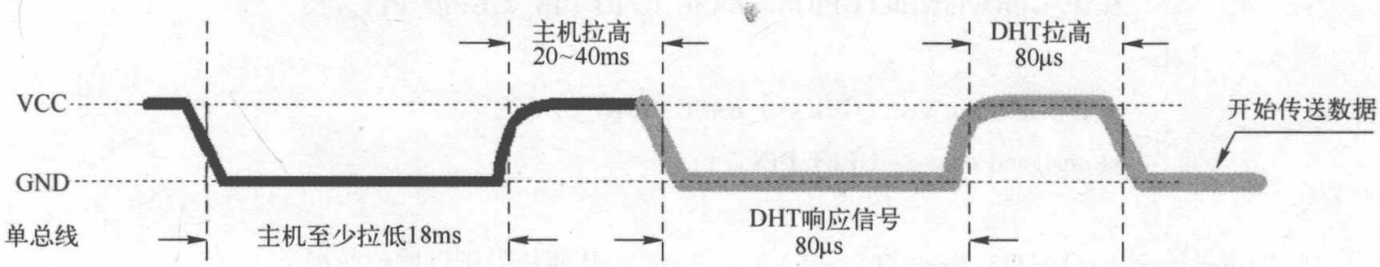
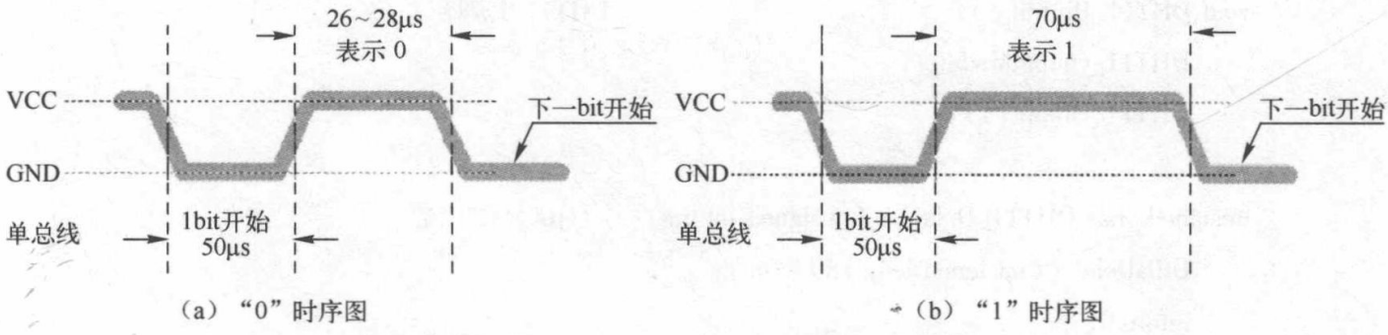


图 5.32　DHT11 开始信号时序图

响应信号发送完毕，拉高通信引脚 80μs 作为传输数据的开始。一旦应答信号发出，则数据便会连续自动传输，串行传输的数据通过特殊的时序来表示“0”和“1”。串行传输的数据时序图如图 5.33 所示。



（a）“0”时序图　　（b）“1”时序图

图 5.33　串行传输的数据时序图

在图 5.33 中，“1”时序逻辑中的高电平持续时间长达 70μs，是“0”时序逻辑高电平持续时长的两倍多，可通过判断 28μs 后的状态来确定该位是“0”或“1”。数据以此方式持续传输，主机若不停止，则 DHT11 会发送 40 位的数据：前 16 位为湿度采集值；后 16 位为温度采集值；最后 8 位为前 32 位的数据之和。由于 16 位湿度采集值的 8 位小数部分为 0，因此仅有前 8 位为实际湿度值。为简化传感器的应用设计，在接收到 8 位整数湿度值后，即可使用 CC3200 发送停止信号（停止信号只需要在传输完毕后拉高通信引脚即可）。

根据 DHT11 时序逻辑，其硬件驱动函数包括 DHT11_InputMode、DHT11_OutputMdoe、

DHT11_Output、DHT11_ReadBit、DHT11_PinInit、DHT11_Start、DHT11_ReadData 及 DHT11_DelayUS，相应的代码如下：

```
void DHT11_InputMode (void){                    //设置通信引脚为输入模式
    MAP_GPIODirModeSet(GPIOA0_BASE,GPIO_PIN_7,GPIO_DIR_MODE_IN);
}
void DHT11_OutputMode (void){                   //设置通信引脚为输出模式
    MAP_GPIODirModeSet(GPIOA0_BASE,GPIO_PIN_7,  \
    GPIO_DIR_MODE_OUT);
}
void DHT11_Output (unsigned char val){          //设定通信引脚输出
    if (val==1)
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,GPIO_PIN_7);
    else
        MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,
        \(unsigned char)~GPIO_PIN_7);
}
unsigned char DHT11_ReadBit (void){             //从通信引脚读取位数据
    if (MAP_GPIOPinRead(GPIOA0_BASE,GPIO_PIN_7)==GPIO_PIN_7)
        return 1;
    else
        return 0;
}
void DHT11_PinInit (){                          //DHT11 引脚初始化
    DHT11_OutputMode();
    DHT11_Output(1);
}
unsigned char DHT11_DelayUS (unsigned int tim){ //1μs 延时函数
    UtilsDelay ((unsigned long)80 * tim);
    return 0;
}
unsigned char DHT11_Start (void){               //通信开始信号
    unsigned char Cont=0;
    DHT11_OutputMode();                         //设为输出模式
    DHT11_Output(1);                            //拉高通信引脚
    UtilsDelay (2000);                          //延时 25μs
    DHT11_Output(0);                            //拉低通信引脚(开始信号开始)
    UtilsDelay (1600000);                       //20ms 拉低延时
```

```
    DHT11_Output(1);                          //再次拉高通信引脚
    UtilsDelay (2000);                        //20~40μs 延时等待
    DHT11_InputMode();                        //调整为输入模式,获取位数据
    while (! DHT11_ReadBit()&&Cont++<100) //读取应答信号
        DHT11_DelayUS(1);
    if (Cont>80)   return 0;                  //应答超时判断
    else   Cont=0;
    while (DHT11_ReadBit()&&Cont++<100)   //读取数据传输开始信号
        DHT11_DelayUS(1);
    if (Cont>80)   return 0;                  //传输开始信号超时判断
    else   Cont=0;
    return 1;                             //起始信号发送并接收到应答信号时返回 SUCCESS(1)
}
unsigned char DHT11_ReadData (void )          //读取数据函数
{
    unsigned char times=7,Cont=0;
    unsigned char SValue=0;
    while (times--)                           //仅读取 8 位整数湿度值数据
    {
        while (! DHT11_ReadBit()&&Cont++<100);  //DHT11 拉低通信引脚 50μs
            DHT11_DelayUS(1);
        if (Cont>51)   break ;                //通信超时判断
        else   Cont=0;
        while (DHT11_ReadBit()&&(DHT11_DelayUS(33)));  //判断位延时等待
        if (DHT11_ReadBit()==1)               //位"0"/"1"判读与处理
            SValue|=(1<<(times));             //获取湿度采集值
        else
            SValue|=0;
    }
    while (DHT11_ReadBit());                  //等待拉低信号到来
    DHT11_OutputMode();                       //设置为主机输出模式
    DHT11_Output(1);                          //停止通信
    return (SValue);                          //返回读取的湿度值数据
}
```

湿度经传感器采集后，通过 UART 打印。UART 硬件驱动程序可参考前面的章节进行编写。DHT11 内部的 ADC 为 8 位分辨率，采集的湿度值需要进行计算才能得出正确的结果。因此，应用程序需要读取湿度值数据并进行数值转换计算后，使用 Tera Term 辅助软件打印湿度值信息。应用程序代码如下：

```
int main (void ) {
    unsigned char flag=0;                          /开起始信号错误标志
    unsigned char ReturnValue=0;                   //8 位整型湿度值
    float Result=0;                                //转换计算最终结果
    BoardInit( );                                  //CC3200 初始化
    PinMuxConfig( );                               //引脚配置
    UART_Init(115200);                             //启动 UART 串口
    DHT11_PinInit( );                              //初始化 DHT11 引脚
    while (1){
        flag=DHT11_Start( );                       //发送开始信号
        if (flag!=1) printf ("error\r\n");         //错误打印
        ReturnValue=DHT11_ReadData( );             //获取 8 位数据值
        if (Result!=0)                             //判断是否接收超时
            Result=(float )ReturnValue/256 * 100;  //湿度值转换计算
        else
            printf ("error\r\n");                  //打印错误
        sprintf ((char * )string,"humidity is: %.2f%%\r\n",Result);
        UART_StringPuts(string);                   //UART 串口打印输出
        UtilsDelay (20000000);
    }
}
```

读者可自行通过“Debug”对项目工程进行编译、下载及调试，并在 Tera Term 中显示采集到的当前湿度值。

5.3.5 气体传感器

气体传感器 MQ-2 能够检测烟雾、液化气、丁烷、甲烷、酒精等气体，具有较好的检测灵敏度，是一款适合多种应用的低成本传感器，可在厨房中用来检测火灾的情况。MQ-2 传感器共有 6 个引脚：2 个引脚用于提供加热电流；4 个引脚用于信号的取出。MQ-2 气体传感器模块包含 MQ-2 的驱动电路，并引出可供测量和提供报警的功能引脚。MQ-2 气体传感器模块原理图如图 5.34 所示。

在图 5.34 中，“DOUT”是引出报警功能的引脚，是比较器的输出引脚。MQ-2 信号的输出电压值与可变电阻 RP 的电压值进行比较，一旦检测到气体浓度过高，MQ-2 信号的输出电压值大于可变电阻 RP 的电压值，则根据比较器原理，“DOUT”引脚将输出低电平，同时，LED 将会被点亮，表示报警触发。“AOUT”引脚与 MQ-2 信号的输出直接连接，用于提供模拟电压量供 ADC 采集。

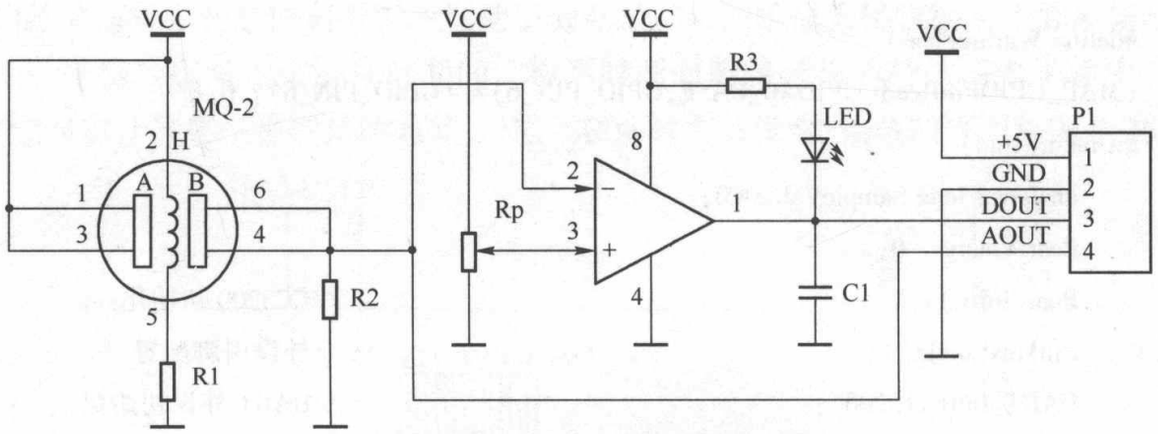


图 5.34 MQ-2 气体传感器模块原理图

应用 MQ-2 气体传感器模块需要驱动 CC3200 的 GPIO、ADC 及 UART 外设。GPIO 外设用来检测报警功能引脚。ADC 外设用来采集 MQ-2 气体传感器输出的模拟电压值。UART 外设主要用于 Tera Term 显示。对这些外设的引脚配置可以使用 PinMux 来完成：GPIO 需要配置 LED 引脚 PIN_64 和报警引脚 PIN_61；ADC 需要配置模拟输入引脚 PIN_59；UART 需要配置 PIN_55 和 PIN_57。GPIO 和 UART 外设硬件驱动程序代码可参考前面的章节内容。ADC 外设硬件的驱动程序代码如下：

```
void ADC_Init (void )  //ADC 初始化函数
{
    MAP_ADCTimerConfig(ADC_BASE,2^17);              //ADC 定时器时间配置
    MAP_ADCTimerEnable(ADC_BASE);                   //打开 ADC 定时器
    MAP_ADCEnable(ADC_BASE);                        //启动 ADC 外设
    MAP_ADCChannelEnable(ADC_BASE,ADC_CH_2);        //启动采集通道 2
}
unsigned long ADC_GetValue (void )                  //获取 ADC 值函数
{
    unsigned long ADValue=0;
    if(MAP_ADCFIFOLvlGet(ADC_BASE,ADC_CH_2)!=0)
        ADValue=MAP_ADCFIFORead(ADC_BASE,ADC_CH_2);
        //从 FIFO 中读取采集值
    ADValue=(ADValue>>2)&0xfff;                 //去除前两位无效位,并去除其他位信息
    //ADC 的采集值包含一个时间戳,存储在[31:14]位中,因此需要清除时间信息
    return(ADValue);                                //返回采集的电压值
}
```

在正常情况下，MQ-2 气体传感器模块启动后，需要等待一定的稳定时间，在稳定时间内，检测值并不是准确的。待其稳定后，ADC 外设开始进行信号的采集，芯片持续检测报警信号引脚 PIN_61 的高、低状态，一旦触发报警，则 LED 灯会通过亮/灭的方式进行表示。根据上述应用流程，应用程序代码如下：

```
#define WaringFlag \
(MAP_GPIOPinRead(GPIOA0_BASE,GPIO_PIN_6)= =GPIO_PIN_6)? 0:1
int main(void) {
    unsigned long SampleValue=0;
    float Volatge=0;
    BoardInit();                                        //CC3200 初始化
    PinMuxConfig();                                     //外设引脚配置
    UART_Init(115200);                                  //UART 外设初始化
    ADC_Init();                                         //ADC 外设初始化
    while(1)
    {
        SampleValue=ADC_GetValue();                     //获取 ADC 采集值
        Volatge=(float)SampleValue * 1.4/4096;          //采集值转换
        sprintf((char *)string,"MQ-2 sample value:%.2fV\r\n",Volatge);
        UART_StringPuts(string);                        //串口信息打印
        if(WaringFlag)                                  //检测报警用引脚
        {
            GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_1,GPIO_PIN_1);
            UtilsDelay(4000000);
            GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_1,~GPIO_PIN_1);
            UtilsDelay(4000000);
        } //持续亮/灭以表示报警被触发
        else
            UtilsDelay(40000000);
    }
}
```

5.3.6 测距传感器

测距传感器采用一个超声波测距探头发送超声波。超声波遇到障碍物将会折回，另一个测距探头负责接收返回的超声波，根据发送至接收的时间和超声波传输的速度推算与障碍物的距离。测距模块 HC-SR04 板载两个超声波测距探头：一个用于发送；另一个用于接收。HC-SR04 测距模块使用 STC11 系列单片机作为控制核心，可控制测距的开始、发送超声波、接收超声波等操作，引出 2 个功能引脚：一个引脚用于触发超声波的发送信号；另一个引脚用于检测发送至接收的时间。

HC-SR04 测距模块的工作电压为 4.5～5.5V；工作电流为 1～20mA；测量距离为 2～400cm；可提供一个 10μs 以上的脉冲触发信号触发传感器；测距探头会连续发送 8 个

40kHz 的脉冲信号；检测时间引脚一直处于低电平，从测距探头接收第一个脉冲信号开始至接收最后一个脉冲信号结束；在此期间，检测时间引脚将转换为持续高电平输出，并利用 CC3200 定时器计算高电平的持续时间。HC-SR04 超声波发送/接收时序图如图 5.35 所示。

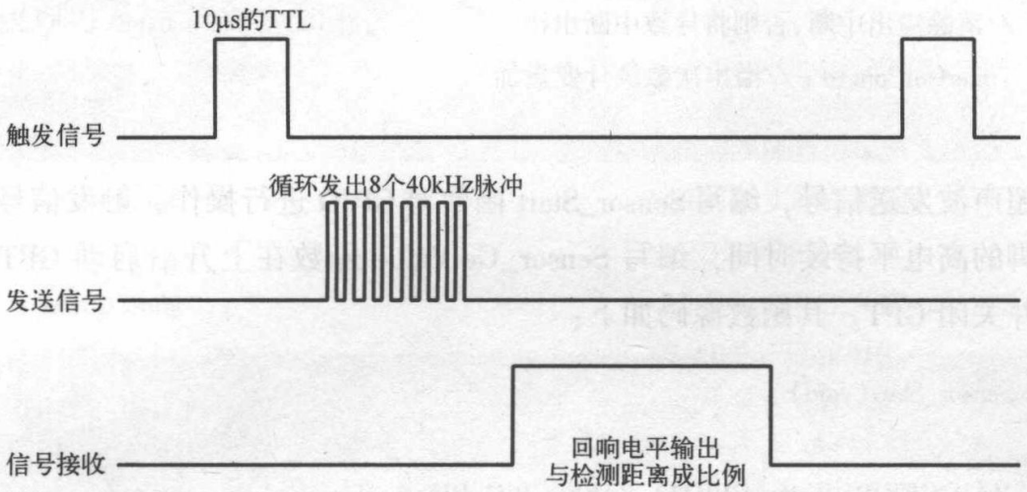


图 5.35　HC-SR04 超声波发送/接收时序图

计算距离时，超声波由 HC-SR04 测距模块自行发送，用户需要使能触发信号并在信号的接收端检测高电平的持续时间。触发信号可使用 CC3200 的 GPIO 外设发送，要求高电平的持续时间不短于 10μs；接收信号使用 GPT 外设，在上升沿启动定时器，在下降沿读取定时器的计数值并关闭定时器；为直观显示距离信息，使用 UART 串口进行打印，因此需要编写 GPIO、UART 及 GPT 外设的硬件驱动程序。编写 GPIO 和 UART 外设的硬件驱动程序可参考前面章节的内容。GPT 外设的驱动函数包括 TIMER_Init 和 TimerIntHandler。TIMER_Init 函数对 GPT 外设进行初始化，TimerIntHandler 为 GPT 外设中断服务程序函数，用于处理定时器计数溢出中断。其源码如下：

```
unsigned int TimerOutCont=0;
void TIMER_Init(void)
{
    MAP_TimerConfigure(TIMERA1_BASE,TIMER_CFG_A_PERIODIC);
    //设置模式为 16 位周期定时器 A
    MAP_TimerPrescaleSet(TIMERA1_BASE,TIMER_A,0);          //定时器预分频设置
    MAP_IntPrioritySet(INT_TIMERA1A, INT_PRIORITY_LVL_1);
    //设置中断优先级
    MAP_TimerIntRegister(TIMERA1_BASE, TIMER_A, TimerIntHandler);
    //定时器溢出中断服务函数注册
    MAP_TimerIntEnable(TIMERA1_BASE,TIMER_TIMA_TIMEOUT);
    //打开溢出中断
    MAP_TimerLoadSet(TIMERA1_BASE,TIMER_A,80000000);      //设置计数初值
    MAP_TimerDisable(TIMERA1_BASE,TIMER_A);               //关闭定时器
}
```

```
void TimerIntHandler(void)
{
    MAP_TimerIntClear(TIMERA1_BASE, TIMER_TIMA_TIMEOUT);
    //清除溢出中断,否则将导致中断出错
    TimerOutCont++; //溢出次数统计变量加一
}
```

为触发超声波发送信号，编写 Sensor_Start 函数对 GPIO 进行操作，触发信号作用后，为检测时间引脚的高电平持续时间，编写 Sensor_GetValue 函数在上升沿启动 GPT，在下降沿获取计数值并关闭 GPT。其函数源码如下：

```
void Sensor_Start(void)
{
    MAP_GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_1, \
    (unsigned char)~GPIO_PIN_1);                    //先拉低触发信号引脚保持低电平
    UtilsDelay(10000000);                           //延时一段时间
    MAP_GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_1,GPIO_PIN_1);    //发送高电平
    UtilsDelay(32000);                              //高电平持续延时,大于 10μs 即可
    MAP_GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_1, \
    (unsigned char)~GPIO_PIN_1);                    //再次拉低表示触发信号发送完成
}
unsigned int Sensor_GetValue(void)
{
    unsigned int FinallyValue=0;                    //用于保存定时器计数值
    while(MAP_GPIOPinRead(GPIOA0_BASE,GPIO_PIN_7)!=GPIO_PIN_7);
    //等待上升沿信号到来,未到来时挂起
    MAP_TimerLoadSet(TIMERA1_BASE,TIMER_A,80000000);
    //再次设置计数初值,定时器默认减计数,启动前需装入计数初值
    MAP_TimerEnable(TIMERA1_BASE,TIMER_A);    //启动定时器
    while(MAP_GPIOPinRead(GPIOA0_BASE,GPIO_PIN_7)==GPIO_PIN_7);
    //等待下降沿信号到来,未到来时挂起
    FinallyValue=80000000-MAP_TimerValueGet(TIMERA1_BASE,TIMER_A);
    //从定时器中读取计数值,由于定时器向下计数,所以要用初值减去计数值
    MAP_TimerDisable(TIMERA1_BASE,TIMER_A);    //读取值后关闭定时器
    FinallyValue+=(TimerOutCont*80000000);
    //加入定时器溢出值,防止高电平持续时间过长,导致定时器计数溢出
    TimerOutCont=0;                                 //溢出中断次数清除
    return(FinallyValue);                           //返回获取的定时器计数值
}
```

应用程序要求根据超声波发送至接收的定时器时间值和超声波的传播速度计算出到障碍物的距离。超声波的速度设为 340m/s，使用定时器时间和超声波速度的乘积表示发送和折回的总距离，至障碍物的实际距离需要除以 2，即距离=高电平时间 * 声速 / 2。相应的应用程序代码（仅列出 main 函数）如下：

```
int main(void) {
    unsigned int SensorValue=0;                          //存储定时器计数值
    float Distance=0;                                    //计算的实际距离值
    BoardInit();                                         //CC3200 初始化
    PinMuxConfig();                                      //外设引脚功能配置
    UART_Init(115200);                                   //UART 外设初始化
    TIMER_Init();                                        //GPT 外设初始化
    while(1)
    {
        Sensor_Start();                                  //发送触发信号
        SensorValue=Sensor_GetValue();                   //获取高电平持续时间计数值
        Distance=(float)SensorValue/80000 * 17;          //计算距离公式
        sprintf((char *)string,"Distance: %.2fcm\r\n",Distance);
        UART_StringPuts(string);                         //串口打印至障碍物距离信息
        UtilsDelay(40000000);                            //500ms 测量一次距离
}
```

5.3.7 红外热释电传感器

红外热释电传感器可检测人体红外，能实现自动冲水、自动开关水龙头、监控电子狗等功能，由热释电晶体、氧化膜、滤光镜片、结型场效应管及电阻等组成，能敏锐地探测处于检测范围内变化的人体红外。单独的红外热释电传感器在使用时需要信号放大电路、电压比较电路、延时电路及相应的电源电路等，设计较为复杂。因此，本节采用市面上已经封装好的红外传感模块 HC-SR501 进行介绍。HC-SR501 模块使用 BISS0001 混合数/模集成芯片处理红外信号，使用菲涅尔透镜增加红外探测范围，同时可避免复杂的电路设计；仅引出一个红外检测结果引脚，检测结果引脚一般处于低电平状态，一旦在监视区域内检测到红外变化，则该引脚将会输出高电平。

应用 HC-SR501 模块时，仅需要持续检测红外检测结果引脚的状态即可，用户不需要任何操作。CC3200 也仅需要配置 GPIO、UART 外设即可实现模块的应用。硬件的相应驱动程序可参考前面章节的内容。应用程序要求检测到人体红外时，LED 能够持续闪烁并且 UART 串口打印“warning!”信息。根据要求编写的应用程序代码如下：

```
int main(void) {
    BoardInit();                                    //CC3200 初始化
    PinMuxConfig();                                 //引脚配置
    UART_Init(115200);                              //UART 外设初始化
    while(1)
    {
        while(GPIOPinRead(GPIOA0_BASE,GPIO_PIN_6)==GPIO_PIN_6)
        {  //持续检测红外检测结果引脚输出状态
            GPIOPinWrite(GPIOA1_BASE, GPIO_PIN_1,GPIO_PIN_1);
            UtilsDelay(2000000);
            GPIOPinWrite(GPIOA1_BASE, GPIO_PIN_1,  \
            (unsignedchar)~GPIO_PIN_1);
            UtilsDelay(2000000);                    //LED 闪烁
            printf("warning! \r\n");                //检测到红外是发送串口消息
        }
        UtilsDelay(40000000);                       //延时等待后再次检测
    }
}
```

注意，红外传感器模块上电后，需要保持 1min 左右的时间预启动，预启动完毕后就可以正常检测了。读者可自行验证红外传感器的运行结果。

5.4 基于 CC3200 驱动设备的应用

CC3200 在使能内部或外部的设备时，由于引脚驱动能力有限，因此需要相应的驱动器来完成设备的控制与通信。驱动设备有多种类型，如继电器用于控制大电流、大电压设备的开启或关断；电动机驱动用来控制电动机的启动，并提供电动机所需的电流。

5.4.1 继电器的应用

继电器可以控制多种设备的开启或关断，仅需要很小的电流就可以实现驱动。继电器的类型较多。电磁继电器是一种常用的继电器，由带铁芯的线圈、衔铁及触点构成。其工作原理是利用铁芯和衔铁之间的吸合能力来实现开启或关断。电磁继电器共有 5 个引脚：2 个引脚用来给带铁芯的线圈供电；在剩余的 3 个引脚中，有一个是公共引脚，与其他两个引脚构成常开触点和常闭触点。每当线圈供电时，常开触点和常闭触点就会翻转。

使用 CC3200 对继电器进行控制时需要控制继电器线圈的供电，即控制常开触点和常闭触点的翻转，又称继电器动作。继电器动作可用开关三极管 9013（或 8050 等）来实现。开关三极管的基极和 CC3200 的 GPIO 外设引脚连接，给 GPIO 的外设引脚赋予高、低电平，即可控制三极管的打开与关闭，从而实现继电器线圈的通、断。电磁继电器的驱动电路如图 5.36 所示。

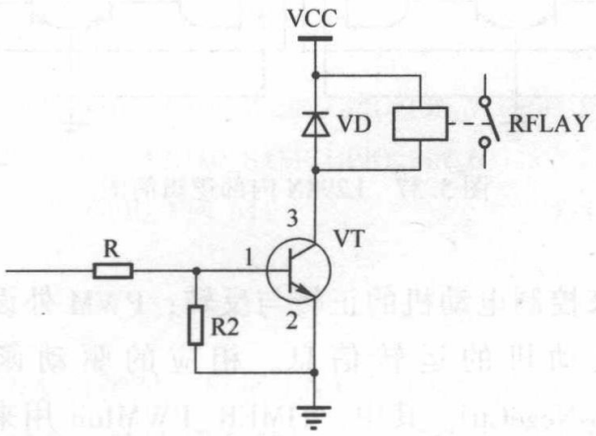


图 5.36　电磁继电器的驱动电路

硬件驱动程序的编写仅需要使用引脚复用配置工具 TI PinMux Tool 生成对应 GPIO 引脚配置的代码即可，应用时，对该引脚的赋值操作即可实现继电器的动作。

5.4.2　电动机驱动的应用

CC3200 引脚的驱动电流受限，不能直接驱动小电动机、步进电动机、舵机等的运转，在实际的开发过程中，电动机必不可少，数量可达 2~4 台，为了控制这些电动机，必须使用额外的驱动设备。本节以小电动机（直流电动机）为例，使用市面上的 L298N 电动机驱动模块进行驱动应用的开发。

L298N 是一款多通道全桥驱动器芯片，可驱动多种不同类型的设备运转，输入电压最大支持 46V，电流最大支持 4A。一个 L298N 芯片可以驱动两台直流电动机，并分别提供使能引脚控制任一电动机的通/断、提供 4 个输入引脚（每个电动机占用 2 个）控制任一电动机的正转与反转。图 5.37 为 L298N 内部逻辑简图。

在图 5.37 中，ENA 和 ENB 为使能引脚；IN1~IN4 为 4 个输入引脚；OUT1~OUT4 为电动机接口。当 ENA、ENB 无效时，与门逻辑会使所有的三极管截止，从而停止电动机的运转；电动机运转时，由 IN1、IN2 控制的三极管要保证一个导通、一个截止，IN1=1、IN2=0 和 IN1=0、IN2=1 是相互反向的（前者的 OUT1 流向 OUT2，后者的 OUT2 流向 OUT1）。

L298N 的应用需要额外附加电路。L298N 电动机驱动模块提供了完整的驱动电路，用户可以直接应用开发。该模块引出了所有的 L298N 引脚，使用方便。CC3200 需要编写 GPIO 外设、PWM 信号（GPT 外设生成）、UART 外设硬件驱动程序来开发 L298N 电动机驱动模

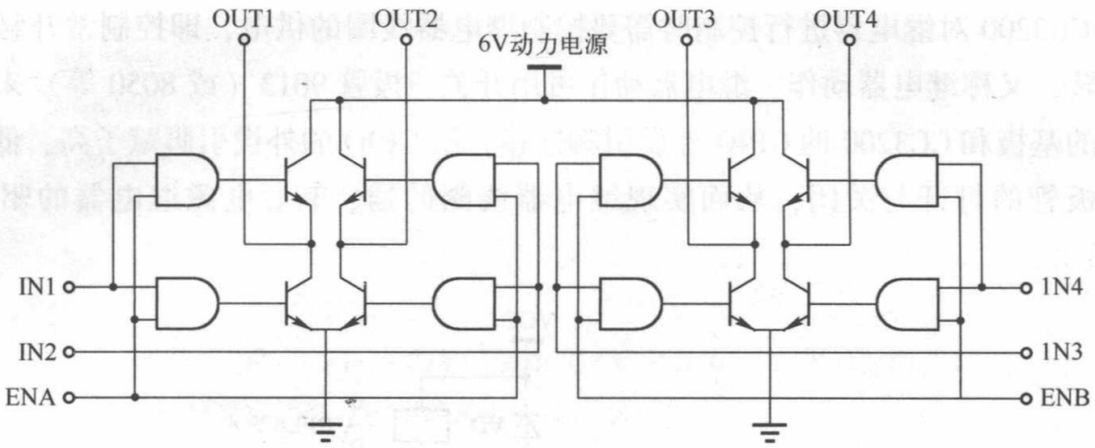


图 5.37　L298N 内部逻辑简图

块的应用。GPIO 外设用来控制电动机的正转与反转；PWM 外设用于控制电动机的转速；UART 外设负责打印电动机的运转信息。相应的驱动函数为 TIMER _ PWMInit、MortorSpeedCrtl 及 MortorPosNegeCtrl。其中，TIMER_PWMInit 用来初始化 PWM 信号；MortorSpeedCrtl 用来调节 PWM 信号占空比，进而控制电动机的转速；MortorPosNegeCtrl 用来控制电动机的正/反转。相应的程序代码如下：

```
#define PRELOARVALUE 40000000
void TIMER_PWMInit(void)
{
    MAP_TimerConfigure(TIMERA3_BASE,  \
    TIMER_CFG_SPLIT_PAIR|TIMER_CFG_A_PWM);          //配置定时器为 PWM 模式
    MAP_TimerControlLevel(TIMERA3_BASE,TIMER_A,1);  //设置 PWM 有效电平
    MAP_TimerPrescaleSet(TIMERA3_BASE,TIMER_A,0);   //预分频值设置
    MAP_TimerLoadSet(TIMERA3_BASE,TIMER_A,PRELOARVALUE);
    //装载定时器计数值,默认减计数
    MAP_TimerMatchSet(TIMERA3_BASE,TIMER_A,PRELOARVALUE);
    //装载 PWM 比较输出匹配值
    MAP_TimerEnable(TIMERA3_BASE,TIMER_A);          //启动定时器
}
void MortorSpeedCtrl(unsigned long dutyvalue)
{
    float MathValue=0;
    MathValue=(float)dutyvalue/100*PRELOARVALUE;    //计算占空比装载值
    TimerMatchSet(TIMERA3_BASE,TIMER_A,  \
    (unsigned long)MathValue);                      //改变 PWM 比较匹配值以改变占空比
}
void MortorPosNegeCtrl(tBoolean flag)
```

```
    {
        if(flag==false){                                    //OUT1 流向 OUT2
            MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_6,GPIO_PIN_6);     //IN1 为 1
            MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,  \
            (unsigned char)~GPIO_PIN_7);                    //IN2 为 0
        }
        else {                                              //OUT2 流向 OUT1
            MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_7,GPIO_PIN_7);     //IN1 为 0
            MAP_GPIOPinWrite(GPIOA0_BASE,GPIO_PIN_6,  \
            (unsigned char)~GPIO_PIN_6);                    //IN2 为 1
        }
    }
```

当 PWM 信号输入至 ENA 引脚时，根据 ENA 和 PWM 的信号特性，电动机处于间歇性的通/断工作状态，占空比越高，转速就会越快。编写的应用程序即为调节 PWM 占空比从而实现转速的控制。其程序代码如下：

```
int main(void) {
    unsigned char value=10;                 //初始占空比值
    BoardInit();                            //CC3200 初始化
    PinMuxConfig();                         //引脚配置
    UART_Init(115200);                      //UART 初始化
    MortorPosNegeCtrl(false);               //设置电动机正/反转
    TIMER_PWMInit();                        //PWM 信号初始化
    while(1)
    {
        for(value=10;value<=90;value+=10)
        {
            MortorSpeedCtrl(value);         //调节电动机占空比为 10%~90%
            sprintf((char *)string,"speed duty is:%d\r\n",value);
            UART_StringPuts(string);        //打印当前占空比信息
        }
        UtilsDelay(100000);                 //延时运行
    }
}
```

以上仅列出 main 的函数部分，读者可自行补充其他部分，编写应用程序完成后，也可自行编译、下载、调试及运行打印。

第 6 章
智能家居与微信公众平台的结合

6.1 微信公众平台

微信是腾讯公司在 2011 年推出的一款通过网络快速发送文字、语音、图片、视频，支持多人群聊的手机聊天软件，官方网址是 http://weixin.qq.com。微信公众平台是 2012 年 8 月 23 日在微信基础平台上新增的功能模块。微信公众平台是微信公众账号所有者进行品牌推广、提高影响力、与用户进行互动交流及提供服务的平台。微信公众账号通过消息、事件、菜单等交互方式为用户提供服务。用户可以关注自己感兴趣的微信公众账号。微信公众账号的所有者每推送一条消息，用户都可以及时看到，并且用户可以发送消息到微信公众账号，从而获取所需要的服务。

微信公众账号分为订阅号和服务号。

公众平台服务号是公众平台的一种账号类型，可以为企业和组织提供更强大的业务服务及用户管理能力，主要偏向服务类交互（功能类似 12315、114、95588，提供绑定信息、服务交互）。其功能为：1 个月（自然月）内仅可以发送 4 条群发消息；发给订阅用户的消息会显示在聊天列表中；服务号会在订阅用户的通信录中，通信录中有一个公众号的文件夹，点开可以查看所有的服务号；服务号可申请自定义菜单。

公众平台订阅号是公众平台的一种账号类型，可为媒体和个人提供一种新的信息传播方式（类似报纸、杂志，提供新闻信息或娱乐趣事）。其功能为：每天可以发送 1 条群发消息；发给订阅用户的消息将会显示在“订阅号”文件夹中，点击两次才可以打开；在订阅用户的通信录中，订阅号在订阅号文件夹中。

6.1.1 注册微信公众账号

要使用微信公众平台，需要打开微信公众平台官网 https://mp.weixin.qq.com/，注册一

个微信公众平台账号。依次填写基本信息→邮箱激活→选择类型为订阅号→信息登记→填写公众号信息即可完成注册。注册微信公众平台账号界面如图 6.1 所示。

图 6.1　注册微信公众平台账号界面

6.1.2　开启公众平台测试账号

由于智能家居系统需要使用微信语音接口，前面注册的普通订阅号并没有开放此高级接口，微信公众平台测试账号可以直接体验和测试公众平台的所有高级接口，因此需要在订阅号中开启公众平台测试账号：登录进入微信公众平台后，单击左下角的“开发者工具”，选择“公众平台测试账号”，或者直接打开网址 http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=sandbox/login，单击登录后，使用手机端微信扫描网页上显示的二维码，即可进入微信公众平台测试账号。

微信公众平台测试账号界面如图 6.2 所示。从图中可以看到该测试账号的信息：测试账号信息中有微信的 appID 和 appsecret，在获得 Token、修改创建自定义菜单或者其他需要验证权限的时候需要用到这两个密钥。注意，要对这两个参数的保密。在开发时，需要对接口配置信息中的 URL 和 Token 进行设置。URL 是微信公众平台服务器请求的开发服务器入口页面，注意不是网站的域名，而是要具体到网页。Token 相当于微信公众平台服务器与开发服务器交互的密钥，在正式部署的时候，需要将其设置得复杂一些，否则可能被黑客利用，伪装开发服务器向用户发送消息。图 6.2 的左下方为测试账号的二维码，可以用手机微信扫描进行关注，在用户列表中可以看到已经关注测试账号的用户。

图 6.2 微信公众平台测试账号界面

6.1.3 自定义菜单介绍

自定义菜单能够帮助公众账号丰富界面，让用户更好更快地理解公众账号的功能。自定义菜单有以下规则：

① 自定义菜单最多包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单；

② 一级菜单最多 4 个汉字，二级菜单最多 7 个汉字，多出来的部分将会以“...”代替；

③ 创建自定义菜单后，菜单的刷新策略是，在用户进入公众账号的会话页或公众账号的 profile 页时，如果发现上一次拉取菜单的请求在 5 分钟以前，就会拉取一下菜单；如果菜单有更新，就会刷新客户端的菜单。测试时，可以尝试取消关注公众账号后再次关注，可以看到创建后的效果。

自定义菜单接口可实现多种类型的按钮。比较常用的类型按钮如下。

(1) click：单击推事件

用户单击 click 类型按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给开发者，并且带上按钮中开发者填写的 key 值，开发者可以通过自定义的 key 值与用户进行交互。

(2) view：跳转 URL

用户单击 view 类型按钮后，微信客户端将会打开开发者在按钮中填写的网页 URL，可通过“网页授权获取用户基本信息”接口获取用户基本信息。

click 和 view 的请求实例如下。创建菜单时，需要将菜单内容组织为如下格式，然后以 POST 的方式向微信服务器提交，即

```
{
    "button":[
    {
        "type":"click",
        "name":"今日歌曲",
        "key":"V1001_TODAY_MUSIC"
    },
    {
        "name":"菜单",
        "sub_button":[
        {
            "type":"view",
            "name":"搜索",
            "url":"http://www.soso.com/"
        },
        {
            "type":"view",
            "name":"视频",
            "url":"http://v.qq.com/"
        },
        {
            "type":"click",
            "name":"赞一下我们",
            "key":"V1001_GOOD"
        }]
    }]
}
```

自定义菜单参数说明见表 6.1。

表 6.1 自定义菜单参数说明

参　数	是否必须	说　明
button	是	一级菜单数组，个数应为 1~3 个
sub_button	否	二级菜单数组，个数应为 1~5 个
type	是	菜单的响应动作类型
name	是	菜单标题，不超过 16 字节，子菜单不超过 40 字节
key	click 等单击类型必须	菜单 key 值，用于消息接口推送，不超过 128 字节
url	view 类型必须	网页链接，用户单击菜单可打开链接，不超过 1 024 字节

正确时，返回 JSON（JavaScript Object Notation，JS 对象标记）数据包为

```
{"errcode":0,"errmsg":"ok"}
```

错误时，返回 JSON 数据包（实例为无效菜单名长度）为

```
{"errcode":40018,"errmsg":"invalid button name size"}
```

6.2 智能家居与微信公众平台结合

6.2.1 微信与智能家居结合的原因

由第 2 章可知，智能家居是物联网的一种应用，是利用先进的计算机、嵌入式系统及网络通信技术将家庭中的各种电器设备通过网络连接在一起，使用户可以更方便快捷地管理家用电器设备，如通过语音、无线信号控制家用电器设备。另外，智能家居内的各种设备之间可以进行通信，根据用户设定的条件进行不同状态之间的联动运行，从而给用户带来便捷、安全和高效。

方便用户控制家用电器设备是实现智能家居的前提条件。目前，智能手机已经大规模普及，功能也越来越强大，移动网络也从原来的 2G 过渡到 3G、4G，并即将发展到 5G，上网速度有了显著提升。智能手机已经成为人们生活的必需品，使用智能手机控制智能家居是一个完美的选择。使用智能手机进行控制有两种方式：一种方式是开发智能家居的 APP，家用电器设备通过路由器上网，智能手机在 APP 中对家用电器设备进行控制；另一种方式是结合当下时兴的社交平台，如 Facebook、微信等。这些社交平台有着强大的社交功能，可以通过网络自由地发送消息。开发者需要做的便是将智能设备作为一个普通的个体加入到相应的平台即可。

针对上述的第一种方式，由于智能手机的操作系统各式各样，主流的有 Android 系统和

iOS 系统，开发者至少需要开发针对两种系统的 APP，有些用户不习惯智能手机中存放过多的 APP，因此此方式费时费力，不利于智能家居的发展。

针对上述的第二种方式，该方式有很多优点。以微信为例，首先，微信是如今大多数用户使用频率最高的社交平台，大多数用户的智能手机上都会有此 APP，如果开发出专门的智能家居公众号，那么不论用户使用的是什么系统，都可以使用扫一扫进行关注，进而绑定设备进行控制，方便快捷，用户体验性好；其次，此种方式开发周期短，开发时接口完善，只需一次开发即可满足所有用户的需要，大大减少了开发成本。因此，使用微信控制智能家居设备是如今信息产业革命的趋势，将两者紧密结合起来能够使得智能家居产业得到突飞猛进的发展。

微信具备物联网运营的三个基本条件：①成熟的平台技术；②庞大的基础用户群；③开放平台。微信不仅仅可以连接用户，还可以连接能上网的设备。每个设备都有一个二维码作为设备 ID，在微信里，可以通过与设备对话来控制设备。人与物可以对话，人可以直接向物发出指令，微信会把人和物链接起来。通过微信，链接全世界。

对于微信而言，最好的切入点是智能家居。智能家居是物联网九大试点领域最接近终端用户的行业。智能家居是最需要人与物互动的物联网行业之一。而微信作为国内最普及的一种终端应用，如果能成为智能家居设备的一个控制器，那么毫无疑问，微信将是智能家居控制终端中最有竞争力的。从智能家居行业向其他行业辐射，微信成为物联网运营平台有路可寻。事实上，海尔智能家居已经利用微信控制智能家电了。

微信公共账号可以实现“自定义界面”。微信成为众多轻型 APP、服务、硬件集合平台的潜力正在变为现实。未来，微信平台必将是物联网运营平台的一个有力竞争者。

6.2.2 微信在智能家居中的应用

在移动互联网发展迅猛的今天，微信的出现可以说是改变了人们原有的沟通方式，通过文字、图片、语音、视频等信息，让朋友之间的交流变得更加简单、高效。目前，微信已经不仅仅是一种社交工具，微信电视、微信空调的出现让人们看到了微信在智能家居应用方面的又一发展方向。

就目前情况来看，微信在智能家居中的应用还都比较简单，大多数都是以“控制”为主，兼顾信息查询和支付等功能。而仅仅是这些看似简单平常的功能，在实际使用上也已经让人们切实感受到了方便与快捷。

现在市面上可以见到的能用微信控制的家用电器有哪些呢？

1. 微信电视

不久前，腾讯携手创维和 iCNTV 推出了微信电视。只要将电视和“中国互联网电视”的微信服务号绑定，微信电视即可实现语音搜索、高级搜索、高清影院、一键收藏、微信照

片电视看、一键服务、微遥控器、远程点播、微信支付等应用。

例如，只要语音告诉微信想看什么台，就会有相关的节目推送过来；如果用户要在地铁里浏览微信推送过来的最新节目单，则只要收藏节目名称，回到家，在微信电视上打开收藏夹，就可以直接观看节目了。甚至当遇到需要付费的电影时，也只需要通过微信“扫一扫”，就可以直接付费了。

2. 微信空调

除了微信电视，微信空调也是微信在智能家居领域应用的一个成功案例。腾讯与海尔联合推出了微信空调，只要关注“海尔智能空调”的公众账号，并绑定家中的微信空调，就可以实现微信操控了。

通过微信可以将智能手机变成空调遥控器，方便操作。用户不仅能完成开/关指令，还可以通过语音、文字输入等方式调节空调。例如，“开机”，空调就会自动开启，输入“26度”，空调就会自动调节到26℃。与原来按键遥控的方式相比，用微信控制更像与家中的空调进行一次聊天。

6.2.3 未来微信在智能家居中的发展

1. 家用电器的控制更人性化

对于微信在目前家用电器中的应用来说，最直接、最普及的是各种控制功能。利用微信，用户可以控制微信电视、微信空调，甚至控制家里的照明系统。

目前，微信对家用电器的操控应用还处于初级阶段；未来，利用微信将可以实现对家用电器更多、更便捷的控制。

（1）语音、短信预约

微信的语音、短信功能是最常用的功能，从目前来看，利用语音或短信来控制微信电视、微信空调等家用电器是非常简单的。比如，如果用微信来控制微信空调，目前只能实现“打开空调”“调至23度”这样的单一指令操作，虽然已经比用遥控器显得便捷了，但仍然还是有可提升的空间。

在未来，用户可以将微信与电视、空调、洗衣机、电灯等所有的家用电器绑定在一起，建立一个“我的窝”的公共账号，当早晨离开家门去上班的时候，用语音或者短信来预约安排这些家用电器的开启、关闭时间。例如，可以一下发出一连串的指令：6点开启饮水机；7点关闭饮水机；18:30开启空调到20度；20:00打开热水器；等等。这些指令还可以与智能手机中的日历功能相关联，即便是节假日也不需要再次调整，可以最大限度地让用户的生活实现“全自动”。

(2) 协同控制

其实上面介绍的通过微信来对家用电器进行预约控制的操作，在多数情况下是要基于远程控制模式来实现的，就是说，只要微信与家用电器绑定，即便是不在同一个局域网下，也能实现对家用电器的管理。

这样，用户就觉得足够了吗？显然不是。

用户还可以通过微信与家用电器的绑定来实现协同控制。何谓协同控制？也就是说，当用户家中的所有家用电器与微信绑定后，就可以对所有的家用电器进行“编程”。例如，当用户在看电视时，如果有电话打来，则不管是打到家里的座机还是这部装有微信的智能手机，只要接通电话，电视就会自动降低音量；或者，用户也可以利用微信把家中的某几个家用电器组合起来，并设置成不同的情景模式，需要时，只要在微信上点一下，温馨的气氛就会出现。

2. 实际应用更个性化

微信对家用电器的控制不管多么复杂，也只是智能家居系统最基本的功能之一，通过微信能够实现的功能远不止于此。下面将一起分享几个在实际应用中个性化的使用实例，希望能够帮助读者开阔思路。

(1) 身份识别

在未来的智能家庭中，传统钥匙的功能越来越被弱化，各种智能电子门锁的概念逐渐将成为家庭安全屏障的首选。在微信时代，也许只要用智能手机中的微信对着门上一个事先拍摄好的位置“一扫”，当所扫描的图案与微信存储的图案匹配后，即可让大门自动开启。如果有人强行打开门锁时，则微信还可以进行实时提醒，起到防盗报警的作用。

实际上，这种形式的安全性还是比较可靠的。首先，需要智能大门与微信绑定到同一账号，即便是更换了智能手机也无妨，只要成功登录账号即可；其次，只有用户自己知道需要匹配的图案是什么，也就是说，只有用户自己知道事先所拍摄大门的位置在哪里。这样一来，即便是有人盗用了用户的微信账号，也不可能轻易打开智能大门，起到了双重防护的效果。

(2) 微信门铃

传统的视频门禁系统需要用户站在屋内显示器的位置才能看到门外的人，并进行交流，而有了微信门铃后，现在的这一切，用户在屋内的任何地方都能全部解决。

微信门铃的应用原理其实很简单，就是将装有微信的智能手机与门外的摄像头门铃绑定。一旦有人按动门铃，微信就会实时提醒，并开启视频模式进行通话。微信门铃的另外一个好处是，用户可以将门外的摄像头设置成常开状态，即便是用户不在家，也可以随时知道用户不在家的时候是谁去按动门铃了，而且用户还能跟来人实时通话。

(3) 定时提醒

也许用户在一些科技新闻中已经见过那些所谓的智能冰箱，大都在冰箱门上有一块触摸

屏，触摸屏上可以显示冰箱内部存放食物的各种信息。虽然这样的智能冰箱目前能够与平板电脑、智能手机互联，但操作过于复杂，甚至有些还需要记忆 IP 地址，登录网站才能看到冰箱的内部信息。

如果这样的智能冰箱与微信结合起来会怎么样呢？

只要将用户的微信与智能冰箱绑定，在微信的冰箱账号中输入“我的冰箱”，则冰箱中的各种食物信息就会马上显示到用户的智能手机上，包括还有多少鸡蛋、多少肉、多少啤酒、它们的保质期都是多少、需不需要补充、如何购买等信息，用户可一目了然，完全不用登录网站查看。除了查看信息，微信还有一个重要的功能，就是当用户智能冰箱中的某种食物即将过期或者所剩无几时，就会不定期地提示用户是否继续购买。可想而知，这个功能对于生活节奏快的都市白领来说是多么的重要。

当然，微信的定时提醒功能不仅只用在智能冰箱上，油烟机的定时清洗、水电煤气表的阈值设定等都可以通过这个功能来实现。

(4) 摇一摇，家用电器“故障”迎刃而解

家用电器出现故障确实是一件让人头疼的事情。如果是小问题，也许用户打打电话、研究研究说明书就能解决；如果是大问题，那么就要与厂商售后人员联系，请专业的维修人员来解决。但是，这些说起来简单，如果真正操作，的确是特别麻烦。

其实，当用户的家用电器出现故障后，在家用电器绑定的情况下，只要用微信摇一摇，就可以摇到与用户使用相同型号家用电器的朋友，如果是小故障或者操作的问题，就完全可以在微信中通过沟通来解决；如果周围没有朋友能解决，那么还可以通过微信视频联系厂家的专业售后人员，让专业的维修人员先通过视频来判断家用电器是否需要报修。这样一来，所有的关于家用电器的复杂售后问题都可以通过微信来解决，相当方便。

第 7 章
云服务平台

云服务平台就是通常讲的云计算平台。云计算是指以应用为目的，通过互联网将大量必要的硬件和软件按照一定的组织形式连接在一起，并随应用需求的变化不断调整组织所创建的一个内耗最小的、功效最大的虚拟资源服务集合。这意味着互联网的计算架构发生改变，即由“服务器客户端”向“云服务平台客户端”演变。这也侧面说明了云计算并不是技术，更不是计算，而是一种基于互联网的全新服务模式。与传统的以桌面为核心的 C/S 处理模式不同，云计算以网络服务为核心，传递的不仅仅是数据，更多的是一种服务，因此，将云计算称为云服务更贴切。

通过云服务可以将计算机设备和大型数据中心连接在一起，为广大用户提供可以共享的软、硬件资源，方便用户随时在平台上获取信息，满足用户的工作需要和生活需要。总体来说，云服务的推动和广泛应用，大大方便了人们的生产和生活。例如，人们经常使用的浏览器搜索引擎就是云服务，通过浏览器搜索关键字可以查到数万条信息，这些信息都不是存储在本地，而是存储在云端，这就是一种云服务；网页游戏、网页 QQ 等也都是一种云服务；当使用 Web QQ 聊天时，本地不需要安装客户端软件，就可以在 Web QQ 中查看自己的好友列表、个人资料等，这些信息都是存储在云端服务器中，在任何地方都可以访问。

云服务的主要优点有：

① 资源集中化管理并按需分配，利用率高；

② 采用分布式存储和计算技术，数据处理能力强；

③ 使用虚拟技术减少对设备的依赖性，适用性强，容错率高；

④ 数据的计算和处理过程放在云端，对客户端的要求低；

⑤ 用户可以定制服务并按需付费，成本较低。

虽然智能家居进入国内已经有 20 多年了，但直到近几年才得到快速发展，主要原因有：一是产品功能单一，智能化程度不高，用户享受不到高科技的魅力；二是价格较高，普通用户接受程度较低。云服务技术的兴起和应用，将有助于这两个问题的解决，从而促进智能家居的普及。

7.1 云服务的发展现状

在信息爆炸的今天，时时刻刻都在产生互联网数据，数据来自人们生活中的各个领域，衣、食、住、行都可以通过互联网进行交互。因此，来自各行各业的数据都在飞速增长，数据中心也在不断膨胀，各式各样的应用还继续不断地产生，利用相关的技术管理 IT 资源和提供资源服务是一个趋势。

云服务可以高效地管理服务器、存储、网络、中间件和软件等资源，为上层的软件开发和数据的分析提供坚实的基础。互联网的发展连同社会经济和商业模式的发展一同推动了云服务的形成和发展。

美国 IBM 公司早些年就推出了自己的云计划，当时 IBM 公司具备云计算相关业务的有利条件。而同为 IT 巨头的 Google 公司也不甘示弱，于 2007 年 10 月推出了相应的云计划，并且还与 IBM 公司进行合作，计划把全球多所大学纳入“云计算”中。随后，亚马逊公司也提供了“弹性计算机云”服务。该服务是面向开发者的。这个服务的推出使得其他没有能力维护大型数据计算中心的小型公司可以按照需求来购买亚马逊数据中心的相关服务。亚马逊、Google、IBM、微软及 Yahoo 等大公司是云服务的先行者，其他成功公司还包括 Salesforce、Facebook、Youtube、Myspace 等。

目前，虽然推出云服务的公司众多，但是 Google 公司一种处于领先位置。Google 公司一直从事搜索引擎的开发，先天优势明显，有分布在全球 200 多个地点的 100 多万台服务器，基于已有的这些基础设备，云服务技术的研究能够顺利进行。Google 对云服务技术的研究一直处于主导地位，研发出了文件的分布式管理系统 GFS，并且通过使用 Map Reduce（分布式计算技术）和 Big Table（一种大数据管理技术）来管理数据和文件，使云服务的相关研究工作在这些技术的支持下稳步进行。Google 还推出了 Google 应用程序引擎服务，是 Google 公司开发的 PaaS（Platform as a Service）应用的典型代表。PaaS 主要是面向开发者的。开发者可以直接在该系统上进行编程和调试，还可以将应用程序布置在系统上。PaaS 的目的是为互联网业务开发者提供开发平台和运行平台，是未来互联网创新的基础平台。

从 2009 年开始，国内才真正开始发展云服务的相关产业，部分城市的地方政府也纷纷建成了云计算中心。目前，国内云服务的发展处于成长期，市场规模逐步扩大。国内的互联网公司，如腾讯、新浪、阿里巴巴、世纪互联等，为了提高效率，在公司内部做出相应的 IT 设施改革，还向外部提供相关的云服务。这些措施有利于降低公司的成本，并拓展公司的业务范围。除了这些互联网企业，我国的传统电信运营商也相继开发了云计算的相关项目，为了不使自己在互联网的浪潮下失去优势，进行公司内部改革，整合内部的现有资源，逐步实现转型。在国内，云计算的 IaaS（Infrastructure as a Service）业务模式处于市场化的

初级阶段，已经出现了一些 IaaS 的商用产品，如世纪互联的 Cloud Ex、中国电信的“e 云”、阿里巴巴的“阿里云”等。而 PaaS 才刚刚进入市场化阶段。

7.2 云服务在物联网中的应用

1994 年，互联网开始进入中国，在 20 多年的时间里，互联网产业呈现出爆炸式的增长。众所周知，互联网产业的发展是与技术、服务、内容、客户端等相关配套设施的发展相辅相成的，人们通过计算机等终端互相连接实现数据信息的快速传递。在追求多方面应用的同时，人们要求互联网要能更好地适用于生活的细节，从而物联网应运而生，从概念的提出到如今各种应用实例的出现，表明了人们迫切需要物联网与现有的互联网整合起来，实现人类社会与物理系统的整合。物联网把新一代 IT 技术充分运用在各行各业中。计算机设备行业的迅猛发展为互联网的高速化、智能化的发展注入了强大的动力。“云服务”以具有的超大规模、虚拟化、高可靠性、通用性、高可扩展性、按需服务、廉价及方便等特点成为互联网发展的新主题。物联网与互联网的整合需要一个或多个强有力的计算中心，能够对整合网络内的人员、设备及基础设施进行实时的管理和控制。云服务的出现恰逢其时，物联网与云服务的结合势必是一种趋势。

7.2.1 云服务与物联网的结合

物联网将硬件和软件结合在一起，可融合多行业的先进技术，并带给开发者无尽的创新机会。从智能家居、智能校园到智慧城市、智能电网、智能医疗，物联网的应用层出不穷。在这些物联网的应用中，存储数据的云服务平台起着承上启下的作用。物联网与云服务都是由互联网的发展衍生出来的。互联网是二者的连接纽带，可以将硬件和软件通过云服务平台连接起来，使用户可以随时获取和使用存储在云服务器上的数据。云服务与物联网的有效结合使云服务技术从理论走向实际应用，并促进社会经济产业的辉煌发展。云服务是实现物联网的核心。运用云服务模式，可以实时动态管理物联网中的各类物品，使智能分析变得可能。物联网将射频识别技术、传感器技术、纳米技术等新技术充分运用在各行各业中，可将各种物体充分连接起来，并通过无线等网络将采集到的各种实时动态信息送达计算处理中心，进行汇总、分析和处理。

物联网把数据信息的载体扩展在实物上，对实物进行智能化管理，为了实现对海量数据的管理和计算，需要一个大规模的计算平台作为支撑。而云服务正好能够实现对海量数据的管理，并能够对数据库的数据信息进行实时动态管理和分析。云服务平台不仅能够负责接收感知层的数据，还可以向应用层提供统一的调用接口，从而可方便数据的调用，将云服务应用到物联网的数据传输和数据应用中，可在很大程度上提高物联网的运行速度。

云服务平台在物联网中发挥的优势主要体现在如下几个方面。

(1) 方便用户

云服务平台应用到物联网产业中可以使设备的控制不受地域限制，只要用户持有移动设备且接入互联网，就可以通过云服务平台控制设备，而且可以获取设备的状态信息，极大地方便了用户。

(2) 云服务能为物联网提供技术支持

物联网为了实现规模化和智能化的管理和应用，对数据信息的采集和智能处理提出了较高的要求。云服务具有规模较大、虚拟化、多用户、较高的安全性等优势，能够满足物联网的发展需求。云服务通过规模较大的计算集群和较高的传输能力，能够有效地促进物联网基层传感数据的互享。云服务技术的高可靠性和高扩展性为物联网提供了更为可靠的服务，也为物联网的建设与发展提供了更好的服务。

(3) 解决了物联网服务器节点不可靠的问题

随着物联网的发展，感知层和感知数据都在不断地增长，如果处理不当，服务器的各个部分很容易出现错误，在访问量不断增加的情况下，会造成物联网的服务器间歇性的崩塌，增加更多的服务器，资金成本较大，而且在数据信息较少的情况下，服务器会产生浪费的状态。基于这种情况，云服务的弹性计算技术很好地解决了这个问题。云服务能使物联网在更广泛的范围内进行数据信息互享。物联网的数据及信息直接存储到网络平台上，而网络平台的服务器分布在世界各地。网络平台的服务器可以不受地域限制，对信息的采集和传输能够很大限度地实现数据信息互享。云服务技术中的挖掘数据技术还能够有效地增强物联网的数据信息处理能力。同时，云服务还能够增强物联网总的数据信息处理能力，提高物联网智能化的处理程度。物联网应用用户的不断增加，可产生大量的数据信息，云服务通过计算机群，可为物联网提供强大的计算能力。

(4) 节约成本

物联网的硬件设备通过 WiFi 连接到云服务平台上；云服务平台记录了所有设备的状态数据；用户可以通过智能手机等对设备进行远程控制。另外，不同设备的数据之间必然有一定的联系，可以在云服务平台上对这些数据进行分析，从而实现节能减排的目的。如果没有云服务平台，则物联网公司不仅要进行设备软、硬件的开发及移动终端的开发，还要进行云服务平台的开发和维护，导致开发成本极其高。如果在现有技术成熟的云服务平台上进行物联网的应用开发，则不仅可以减少系统的开发时间，还可以节约成本，提高系统的稳定性。

(5) 降低创业难度

在云服务平台出现之前，创业团队想要进军物联网几乎是不可能的。一个物联网系统的开发需要各行各业的技术人员，假如没有云服务平台的支撑，则创业团队就会需要很多的启动资金，即使这样，创业团队要想实现所有的技术也是个不小的挑战。有了云服务平台后，它就可以为创业团队提供相应的服务，创业团队只需将剩余的工作做好做精即可。硬件团队

主攻硬件开发，在提高产品质量的同时尽量降低成本；移动终端开发团队主攻软件开发。这样就可以使每个创业团队找到自己相应的定位和目标，加速物联网产业的发展。

7.2.2 云服务所提供的服务分类

从物联网的结构来看，云服务已成为物联网的重要环节。物联网常见的层次结构包含：感知层，将物品信息进行识别、采集；传输层，通过现有的2G、3G、4G及未来的5G等通信网络将信息进行可靠的传输；信息处理层，通过后台的云服务系统进行智能分析和管理。物联网与云服务的结合必将通过对各种能力资源的共享（包括计算资源、网络资源、存储资源、平台资源等）、业务的快速部署、人物交互新业务的扩展、信息价值的深度挖掘等多方面的促进带动整个产业链和价值链的升级与跃进。物联网强调物与物相连，设备终端与设备终端相连。云服务能够为连接在云上的设备终端提供强大的运算处理能力，以降低终端本身的复杂性。物联网与云服务都是为满足人们日益增长的需求而出现的。

目前，物联网的服务器都部署在云端，通过云服务提供应用层的各项服务。用户根据自己的需要可以租用云中的服务、技术。目前被业界最广泛认同的云服务模式是SPI（SaaS、PaaS和IaaS）模式。SPI模式是由美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）定义的。

SPI模式提供了以下三个层次的服务。

（1）SaaS（Software as a Service）：软件即服务

SaaS是一种通过Internet提供软件的模式，提供给用户的服务是运营商运行在云服务基础设施上的应用程序。用户可以在各种设备上通过客户端界面访问，如浏览器，不需要管理或控制任何云服务的基础设施（网络、服务器、操作系统、存储等），也不需要购买软件，而是向运营商租用基于Web的软件来管理企业的经营活动，如向亚马逊租用。

（2）PaaS（Platform as a Service）：平台即服务

PaaS实际上是指将软件研发的平台作为一种服务，以SaaS模式提供给用户，用户将采用提供的开发语言和工具（如Java，python，.Net等）开发的应用程序部署在运营商的云服务基础设施上，不需要管理或控制底层的云基础设施（包括网络、服务器、操作系统、存储等）。用户可以控制部署的应用程序，也可以控制运行应用程序的托管环境配置。因此，PaaS可以看成SaaS模式的一种应用。PaaS的出现可以加快SaaS应用的开发速度，如软件的个性化定制开发。

（3）IaaS（Infrastructure as a Service）：基础设施即服务

用户可以通过Internet从完善的计算机设施上获得服务，如租用硬件服务器处理CPU、内存、存储、网络和其他基础的计算资源，能够部署和运行任意软件，包括操作系统和应用程序。用户不需要管理或控制任何云服务的基础设施，可以控制操作系统的选择、存储空间、部署的应用，也可以有限制地控制网络组件（如路由器、防火墙、负载均衡器等）。

PaaS 和 IaaS 源于 SaaS 理念。PaaS 和 IaaS 可以直接通过 SOA/Web Services 向平台用户提供服务，也可以作为 SaaS 模式的支撑平台间接向最终用户服务。

7.2.3 云服务在物联网应用中面临的问题

物联网主导着互联网和通信产业的发展，并将在以后的几年内形成一定的产业规模。云服务与物联网的结合是二者各自从概念走向应用的深刻体现，将是未来发展的趋势。

云服务与物联网的结合面临如下问题和挑战。

(1) 云平台标准化的通用性问题

随着云服务技术的发展，现在很多企业都建立了自己的云服务平台，为用户提供不同的云数据和云服务。但是，由于这些数据和服务的制定标准各不相同，因此导致了各种云服务平台之间难以实现共享和交流。众所周知，云服务更加注重的是云端数据的共享和交互，给用户带来连贯性的体验。例如，个人电脑从传统电脑向智能手机和平板电脑、互联网电视、车载资讯的娱乐终端等智能终端方向发展，操作系统也出现百花齐放的局面，云服务平台要支持这些系统，就要做好标准化的通用性。在标准化方面，云服务的目的之一是要将计算资源以最经济的方式提供给社会。由于计算资源规模巨大，不可能每一个企业和机构都发展自己的云服务平台，因此要由主要的互联网公司提供给全社会。云服务是不是在通用的标准上提供的，能否做到开放和合作，关系到云服务技术能否普及，以及能否带动整个 IT 产业上下游的发展。因此，需要建立一种统一的标准，促进云服务平台之间的共享和交流。

(2) 安全问题

云服务技术的应用正在逐步深入，应用领域不断扩展。伴随云服务技术的普及，云服务的安全日益成为影响其应用和发展的重要问题。

① 物联网系统的不安全性。

物联网系统的安全性是制约物联网是否被广泛采用的最大障碍之一。随着越来越多的设备变得“智能化”，越来越多的潜在安全性漏洞将会出现。用户的数据存储、处理、网络传输等都与云服务系统有关，包括如何有效存储数据以避免数据丢失或损坏、如何对多用户应用进行数据隔离、如何避免数据服务被阻塞等。这些都需要研究构建先进的软、硬件安全机制，同时还要将安全机制的成本和功耗保持在较低的水平上。

② 云共享存在不安全性。

虽然 IaaS 运营商的基础设施是自己建立的，具有良好的扩展能力，但这种架构面临的挑战是底层组件通常没有为多租户架构提供强大的隔离措施，非法用户很有可能窃取、篡改合法用户的数据，甚至干扰破坏合法用户应用程序的正常运行，还有可能出现合法用户和非法用户共同使用一台服务器的情况，给合法用户带来隐患。

③ 接口的不安全性。

对大多数企业而言，业务转移到云是不可避免的。遗憾的是，服务运营商一般只关心业

务转移到云，而不考虑 API 中的不安全因素。在一般情况下，管理、配置和监控都是通过这些接口执行的，安全性和可用性完全依赖于每个基础 API 内置的安全性，从身份认证到访问控制，再到加密和监控都通过 API 体现出来。目前，服务运营商通常不提供这样的设计，意味着攻击者发动恶意攻击的成功率会大幅度提高。

④ 特权用户访问和身份假冒问题。

在基于云服务的网络上，重要的数据会被远程办公的企业员工监控，企业以外的人就有可能获得访问重要数据的全部权限。如果攻击者获取到普通用户和特权用户的身份验证信息，假冒合法用户，那么攻击者就会进行一些非法活动，如窃取用户数据、篡改用户数据、恶意消费等。

⑤ 数据在云端的传输和存储存在不安全性。

在云服务模式下，大量的重要数据通过互联网传递到云服务运营商进行处理时面临以下几个问题：一是如果想访问自己的数据，那么如何在保证安全的前提下迅速得到访问信息；二是如果数据在传输过程中被非法用户截获，那么如何保证这些数据不被窃取和篡改；三是在重要数据传输到云服务运营商后，如何保证重要的数据不被篡改；四是由于用户、信息资源的高度集中，因此云服务平台很容易成为黑客攻击的目标，其造成的后果和影响将可能远远超过传统的网络；五是在云服务模式下，所有的重要数据都存储在云中，并不知道自己的重要数据被放在哪台服务器上，甚至不清楚数据被放在哪个国家或地区，也不清楚云服务运营商是否会按照企业的要求确保敏感数据不被泄露；六是重要的数据在云中通常是处在数据共享的环境中的，如果不能对重要的数据进行有效的隔离，那么一旦数据加密出现问题，就有可能造成重要数据无法正常使用；七是当发生云服务系统故障时，云服务运营商如何保证数据的迅速恢复；八是重要的数据和核心业务都处于云服务系统中，其业务流程将依赖于云服务运营商所提供的服务，这对云服务运营商的云服务平台服务的连续性、IT 流程、SLA、事务处理和安全策略等提出了挑战。

以上所说的只是物联网与云服务发展过程中所遇到的一部分主要问题，不可能面面俱到，在实践过程中仍可能遇到这样或那样的问题。有些问题可能无法预料，但可以肯定的是，只有把来自物联网及来自云服务两个方面的问题都解决之后，实现云服务在物联网系统中的完美利用才可能取得突破性的进展。

7.3 基于云服务的智能家居

众所周知，智能家居系统已经成为当前乃至今后家居行业发展的重要方向。云服务平台与智能家居系统的结合已经初见端倪，云数据可使智能家居行业受益更多。智能家居与云服务的结合，不仅可以满足用户对家用电器设备的控制需求，也会带来一些新的应用前景。云服务平台主要用于接入传感器数据，并为移动客户端提供远程服务，具有以下几个优势：

① 云服务平台会将自身的计算资源和存储资源打包成 API 接口以提供服务，用户只需简单的几步操作便能将传感器与云服务平台建立远程连接，以此实现智能家居的远程监控。

② 云服务平台采用高并发的传输和存储方案，可以满足海量传感器数据的并发处理，确保数据存储的完整性和高效性。

③ 云服务平台可提供数据的通信功能，实现对家用电器的远程控制。用户可以通过移动设备，如智能手机、平板电脑对各类家用电器进行实时远程控制，满足用户全天候监控家居生活的需求。

④ 云服务平台可提供数据实时分析功能，通过收集、存储的传感器数据和用户操作记录，并利用其提供的弹性服务对这些海量数据进行分析处理，为智能家居的智能化提供决策支持。

7.3.1 基于云服务的智能家居的系统组成

随着云服务技术的不断发展，云端将成为智能家居系统的中心，基于云服务的智能家居系统主要由三部分组成：云平台（数据中心）、控制端和家庭设备。

1. 云平台（数据中心）

云平台（数据中心）是一个提供云服务的服务集群，可提供以下功能：

① 通过 Internet 接收来自家庭网关的数据并存储，根据内置策略或来自控制端的指令将控制数据传输给家庭网关；

② 通过 Internet 与控制端连接，向控制端提供系统的实时数据或历史数据，接收来自控制端的指令；

③ 内置大量家用设备控制模型供家庭网关控制使用；

④ 对存储的大量数据进行数据挖掘，寻找可供进一步利用的知识。

云平台（数据中心）的实现既可以是自行搭建服务器集群的私有云，也可以直接租用大型云服务公司所提供的服务。例如，提供多平台文件同步的软件 DROPBOX 就直接采用了亚马逊公司的云服务，自身没有任何硬件。也就是说，在云平台（数据中心）的构建上可以不考虑硬件，而集中精力进行软件的开发，尽可能为用户提供可靠、易用的服务，所需的存储和计算资源采用随用随买的方式，使整个结构具有很高的灵活性和可扩展性。

云服务平台通过提供一个基础服务层为智能家居系统提供一些必要的功能，如用户认证、数据存储、与家庭网关和智能终端的联网和编程接口，可为更高级的应用层提供一些标准的基础服务。系统开发者是软件生态系统的维护者，大量的第三方厂商可开发应用该系统，如提供 Web 网站、手机应用程序、数据分析等。家用电器和传感器厂商可以利用这些程序接口向系统提供最新的设备驱动，供家庭网关调用。

2. 控制端

控制端是一个人机界面设备，是用户使用智能家居系统的媒介。控制端可以是一台普通家用计算机中的软件，也可以是智能手机或平板电脑中的应用，或者是智能电视机内置的功能，可使用户获得系统各方面的信息，对系统进行配置和使用。例如，Google 在几年前就已经发布了“Android@Home”的软件平台，该软件平台不仅能够作为智能手机和电脑的操作系统，也能够让家用电器智能化，包括电灯、电器、灌溉系统、可视对讲、报警、恒温器等。Android 的应用开发者可以开发一款基于该操作系统的应用，连接系统的云服务并绑定账户后，就可以查看自家系统的实时情况、历史数据并进行配置和操作。

用户是智能家居系统的控制者，可配置和使用系统，同时也是整个系统的一部分。通过简单的室内定位系统和随身的 RFID 标签，智能家居系统就能感知用户的存在。例如，当用户离开书房进入厨房时，厨房的灯光会自动开启，客厅的灯光自动关闭；当系统检测到房屋中已经没有人存在时，就会使整个系统进入最低功耗运行。这些控制由预先设在云平台（数据中心）的控制策略来执行，而用户是被服务者。

3. 家庭设备

家庭设备包括家庭网关、传感器和各种控制器。家庭网关是连接家庭内部网络和外部网络的连接设备。智能家居网关是家庭资源管理和配置中心，可完成家庭组网和节点控制等功能。智能家居网关通过无线组网技术连接家庭网络中各传感器的开关节点，通过标准的通信协议，对内实现智能家居内部网络的管理和控制，对外作为家庭网络和外部网络的信息交互接口。家庭网关可以在传感器和控制器接入家庭网络后调用基本功能。各种控制器主要由相应的厂商进行开发，根据系统的统一要求增加无线模块，并向云平台（数据中心）提交控制模型。当控制器被最终用户购买并接入智能家居系统时，家庭网关可以从云平台（数据中心）自动下载控制模型实现自动配置。另外，执行器是指所有执行控制策略的设备，包括各种家用电器。随着家用电器本身智能化的发展，有可能将每一件家用电器都接入家庭网络中。

相比较可以看出，传统的智能家居以家庭网关为核心，所有的设备均与家庭网关相连接，向家庭网关提供数据，并接受家庭网关的指令。如果采用云服务器来替代目前的家庭网关，在智能家居中引入云服务，就可以将家庭网关获取的各种传感器数据传送到云服务器，并接受来自云服务器的指令，从而对智能家居系统进行控制。这样的方案具备以下优势：

① 可缩减并明确家庭网关的任务，便于家庭网关的标准化和通用性；

② 云服务器可以接受家庭系统的实时数据，可在更大范围内进行统筹安排；

③ 云服务器可以存储大量的既往数据，便于未来在此基础上进行数据挖掘，从而为整个系统的优化和相关领域的发展提供支持。

7.3.2 基于云服务的智能家居的特点

云服务是智能家居最好的伙伴，通过云服务建设一个云端，既可精准快速地实现对家居设备的控制，同时成本也更加低廉。

基于云服务技术的智能家居系统具有以下特点。

1. 提供的是服务而非硬件

基于云服务的智能家居系统，其核心功能在于背后提供的服务。用户不需要了解云服务的具体机制就可以充分享用服务。可以说，“云”提供的服务是透明的，不像硬件那样具体有形。由云端连接的智能家居系统由系统的运营方来提供智能家居的服务，设备商和最终家庭用户都可以被视为该系统的用户。系统用户不需要了解系统的具体实现。设备商仅需按照系统的要求制造硬件设备。设备硬件将数据通过家庭网关向云平台（数据中心）输送，并执行由云平台（数据中心）下达的指令。家庭用户可通过各种智能终端获取云平台（数据中心）处理好的数据并提出自己的要求或控制策略。智能家居成为一种标准化的服务和基础设施，用户需要做的就是接入该系统，而不是像过去一样需要通过一个包揽所有功能的家庭网关及若干特殊的家用电器来建立一个智能家居系统。在基于云服务的智能家居系统模型中，家庭网关只是连接传感器、家用电器和云平台（数据中心）的中介，而非系统中枢。当大多数智能家居系统都利用分布式云端系统进行场景运算和学习时，智能家居才能真正实现用户的无感操作，摆脱对于智能手机 APP 等遥控手段的限制。因此，可以预见，未来智能家居行业的竞争，实质上将是云技术的竞争，比拼的也是云端和软件服务的能力。

2. 经济性

云服务提高了存储和计算的硬件使用效率。与独立的嵌入式控制器相比，云服务能够提供更廉价的单位存储和计算成本，设备商大批量地制造尽可能简单、接口统一的家庭网关和通信模块即可，大大降低了整个系统的成本。这种家庭网关只需要支持 ZigBee 网络和 TCP/IP 协议，并能缓存少量的数据便足够了。

3. 高可用性

通过集成海量存储和高性能的计算能力，云服务能够提供一定满意度的服务质量。云服务系统可以自动检测失效节点，并将失效节点排除，不影响系统的正常运行。云服务的引入使智能家居成为一种高可用性的基础服务。系统升级或维护时可以只暂停部分运算节点，而用户感受到的是无间断的系统运行。

4. 高层次的编程模型

云服务系统可提供高级别的编程模型。用户通过简单的学习就可以编写自己的云服务程

序，并在云服务系统上执行，满足自己的需求。云服务的运营方提供了统一的操作系统和编程环境。家庭硬件的制造商可在此基础上进行相应的开发，使硬件制造商可在更高的抽象层次上编程，不需要关心存储和计算的实现细节，只关注网络的传输和用户体验即可。

5. 可靠性

智能家居系统的稳定性、可靠性、安全性是建立在良好的硬件基础上的，没有容量足够大的存储设备，信息将会难以存储，甚至造成大量的数据丢失，就不要说对数据进行有针对性的查询分析和计算了。尤其是目前的智能家居，需要存储的信息包括音频和视频，这些都需要极大的存储容量。若是关键数据丢失，则很有可能就会造成很大的损失。因此，普通的存储设备很难跟得上数据存储所需要的增长速度。而云却是一种低成本的虚拟计算资源。云服务将这些资源集中起来，自动管理，用户可以随时随地申请部分资源，支持各种应用程序的运转，省去了大量的维护工作，降低了成本，提高了工作效率，获取了更好的服务。

云服务系统由大量的商用计算机组成机群，向用户提供数据处理服务，采用数据冗余和分布式存储来保证数据的可靠性。经过多年的发展，云平台（数据中心）的建设维护技术已经非常成熟，如 Google、亚马逊等运营商提供的云服务是非常稳定和可靠的。用户可以直接租用这些公司的云服务，甚至可以同时租用几家运营商的服务互为冗余，使整个系统的可靠性甚至可以高于这些大型运营商。

总之，基于云服务的智能家居系统将有力促进智能化的真正实现，同时其经济、易用的特点可让更多的普通用户也能体验到智慧生活。

7.4 常用的云服务平台

目前的云服务平台主要是以阿里云、百度云为主，还没有任何一家可以独占鳌头，仍处于多方竞争的局面。下面将介绍几种智能家居设计中常用的云服务平台。

1. 百度云—天工

百度云是百度提供的公有云平台，2015 年正式开放运营，可提供“云服务+大数据+人工智能”三位一体的云服务。百度云在语音和图像识别、智能推荐、深度学习、大数据挖掘及预测等新兴技术方面拥有一定的技术优势。天工是基于百度云构建的，是融合百度大数据和人工智能技术的“一站式、全托管”智能物联网平台，可提供物接入、物解析、物管理、规则引擎、时序数据库、机器学习、MapReduce 等一系列物联网核心产品和服务，帮助开发者快速实现从设备端到服务端的无缝连接，高效构建各种物联网的应用（如数据采集、设备监控、预测性维护和保养等）。

通过持续的技术创新和不断地积累行业经验，天工平台日益成为更懂行业的智能物联网

平台，为工业制造、能源、零售 O2O、车联网、物流等行业提供了完整的解决方案。同时，基于天工平台设备认证服务，可建立互信、共赢的生态合作机制，帮助用户快速实现万物互联。

2014 年，百度云推出了 Baidu Inside 创新智能硬件合作计划。利用百度的平台化和接口化优势，Baidu Inside 在技术上为智能硬件提供了智能语音、图像识别、云服务、LBS 定位等支持，为硬件创业公司提供技术、营销、数据等多方面支持。该计划已覆盖智能语音、智能影音、智能健康、智能办公等领域。2016 年，百度宣布推出百度智能家庭软件平台，借助百度的大数据、云服务及百度大脑等技术积累，不仅为硬件厂商提供了标准化的 SDK，还能在此基础上提供设备互通用户操作体验、智能化分析、服务变现等全套解决方案。百度希望通过开放平台策略联手第三方硬件厂商快速、低成本地切入智能家居市场。

2. 阿里云

阿里云是阿里巴巴集团旗下的公司，致力于打造公共、开放的云服务平台。阿里云平台与其他云平台相比具有以下优点。

稳定性：服务可用性高达 99.95%，数据可靠性高达 99.99% ，支持宕机迁移、数据快照备份和回滚、系统性能报警。

容灾备份：每份数据多份副本，单份损坏可在短时间内快速恢复。

安全性：支持配置安全组规则、云盾防 DDOS 系统、多用户隔离、防密码破解。

多线接入：基于边界网管协议（Border Gateway Protocol，BGP）的最优路由算法。BGP 多线机房，全国访问流畅均衡。骨干机房，出口带宽大，独享带宽。

弹性扩容：10 分钟内可启动或释放 100 台云服务器（Elastic Compute Service，ECS）实例；支持在线不停机升级带宽；5 分钟内停机升级 CPU 和内存。

成本低：不需要一次性大投入，可按需购买，弹性付费，灵活应对业务变化。

可控性：作为云服务器 ECS 的用户，拥有超级管理员的权限，能够完全控制云服务器 ECS 实例的操作系统，可以通过管理终端自助解决系统问题，并可以进行部署环境、安装软件等操作。

易用性：含有丰富的操作系统和应用软件，使用镜像可一键简单部署同一镜像；可在多台 ECS 实例中快速复制环境，轻松扩展；支持自定义镜像、磁盘快照批量创建云服务器 ECS 实例。

API 接口：可以使用 ECS API 调用管理，通过安全组功能对一台或多台服务器进行访问设置，开发使用更加方便。

目前，阿里云在中国的公有云市场占据最高的市场份额。阿里云之所以能够占据优势，最根本的原因还是来自阿里此前收购的万网。万网一直从事域名和服务器方面的服务，数年内积累了相当数量的企业客户和服务经验，为阿里云的顺利推出奠定了基础。

在智能家居平台上，阿里云采用 IT 基础架构模式，推出了云服务器 ECS、关系型数据

库服务 RDS、开放存储服务 OSS、开放数据处理服务 ODPS 等产品服务。开发者可以根据这些架构进行开发，增加自身的智能应用。阿里研发的基于云服务、以数据和服务为导向的万物互联网操作系统 YunOS，具备高兼容性和可扩展性，广泛适用于各种 IoT 设备，包括智能手机、互联网汽车、互联网电视、智能家居、智能穿戴等多种智能终端，以及芯片及传感器，拥有一站式的数据平台和多层级的安全框架。

阿里云智能家居始于2014年与美的的合作。双方共同构建基于阿里云的物联网开放平台，实现家用电器产品的连接对话和远程控制。美的的全系列产品接入平台，阿里云提供计算、存储和网络连接能力，并帮助美的实现大数据的商业化应用。阿里与美的联合在 IoT 领域推出了首款基于 YunOS 智能操作系统的智能冰箱后，2015 年，长帝与阿里云合作开发了云智能烤箱；2016 年，格力联手阿里推出了智能云技术空调。很多企业都通过阿里云平台实现了从软件到服务的战略转型。阿里在智能领域为企业所能提供的支持，也从早期的云存储和云服务升级到“一站式”的全过程服务。

3. 腾讯云

腾讯云相对来说起步较晚。腾讯云提供包括云服务器、云存储、云数据库及弹性 Web 引擎等基础云服务，还拥有腾讯云分析（MTA）、腾讯云推送（信鸽）等腾讯整体大数据能力，以及 QQ 互联、QQ 空间、微云、微社区等云端链接社交体系。腾讯以腾讯云为基础提供了大数据分享、优图人脸识别、VR 等多项核心产品，在金融、医疗、智能硬件、电商、O2O、旅游、政务等方面提供了解决方案。

2014 年，在智能家居领域，腾讯云推出了 QQ 物联，并着手布局物联平台，通过 QQ 物联将设备端、云端和用户端连接在一起，通过将 QQ 账号体系、QQ 消息通道能力等核心能力提供给可穿戴设备、智能家居等领域的合作伙伴，从而实现用户与设备、设备与设备之间的互联互通。2014 年，腾讯云联合因特网共同推出了软件、硬件一体化的智能家庭网关解决方案。2015 年，双方又共同打造了智能家居及物联网开放服务基础架构。腾讯云提供强大的端到云传输通道、智能云综合服务，并无缝集成在因特网智能硬件终端及智能网关的 QQ 物联 SDK，可方便各类智能设备通过腾讯 QQ 物联协议无缝接入。为瞄准场景化应用，腾讯云又推出了智能语音服务。腾讯云技术团队联合微信 AI 团队开放微信语音处理技术，在语音处理基础功能上结合云端能力。腾讯云智能语音服务支持云端+嵌入式，可以覆盖更多的应用场景。在 2016 年的腾讯全球合作伙伴大会上，腾讯正式发布物联云，不仅可支持消费物联网，还可支持工业物联网的新产品，构建物联云新生态。

4. 华为云

华为云成立于 2011 年，隶属华为公司。华为云立足互联网领域，依托华为公司雄厚的资本和强大的云服务研发实力，提供包括云主机、云托管、云存储等的基础云服务、超算、内容分发与加速、视频托管与发布、企业 IT、云电脑、云会议、游戏托管、应用托管等的

服务和解决方案。

与 BAT 相比，华为正在从硬件设备全面迈向云服务，依托以电子设备为核心的主营业务上所积累的用户资源、硬件能力及渠道优势，很快就在云服务领域占据了优势地位。HUAWEI-HiLink 是华为开发的智能家居开放互联平台，通过提供开放的 SDK，并建设开发者社区为开发者提供全方位的指导，帮助开发者从开发环境搭建到集成、测试提供一站式的开发服务。华为将通过 HUAWEI-HiLink 智能家居开放互联平台和所有的智能硬件厂家一起，形成开放、互通、共建的智能家居生态。2016 年，在华为全联接大会上，HUAWEI-HiLink 智慧家庭生态联盟成立。华为在智慧家庭领域的核心生态伙伴已经有包括美的、海尔、欧普照明、Broadlink、杜亚、鸿雁等在内的超过 20 家企业。企业将对接 HUAWEI-HiLink 家庭云平台，以 HUAWEI-HiLink 智能路由、SDK、生态伙伴产品硬件为基础，由 HUAWEI-HiLink 联盟认证，通过智能家居 APP 实现智能操控。

5. 京东云

在智能家居平台建设中，京东云走的是应用和业务驱动的云模式，由物流云、金融云、电商云、大数据分析云、智能硬件云等组成。京东云平台根本上还是为自身的业务服务，即如何简单便捷地用 APP 操作更多的家居设备。京东云于 2014 年正式推出，联合智能产业链上下游各类合作伙伴，具有大数据分析和开放能力，可提供强大的超级 APP。2016 年，京东与叮咚合作，京东云语音服务开放平台正式上线，将服务与智能家居入口连接。美的、小天鹅、智米等企业也加入了京东云服务开放平台。

6. 机智云

机智云是广州机智云物联网科技有限公司旗下的品牌，是一家智能硬件自助开发和物联网（IOT）云服务平台，是中国最大的物联网开发平台，中国第一个技术孵化平台，拥有中国最大的物联网开发者社区。机智云主要为开发者提供物联网设备的自助开发工具、后台技术支持服务、设备远程操控管理、数据存储分析、第三方数据整合、硬件社交化等技术服务，也为智能硬件厂家提供一站式的物联网开发和运维服务，将智能硬件产品的开发周期最快缩短到半天，可快速实现智能化。机智云服务的用户主要来自消费类智能硬件厂家（智能家居、可穿戴产品），以及工业、商业应用、智慧城市建设等，至 2016 年 4 月已服务全球超过 700 家用户，接入的激活智能硬件全球出货总量超过 600 万台，已聚集超过 5 万名物联网开发者。开发者在研智能硬件项目超过 2 万个，是物联网云服务提供商。

7. Yeelink 物联网云

Yeelink 物联网云是一个开放的通用物联网平台，旨在利用无线网络、开源硬件和软件、智能手机和 APP 共同打造一个家庭智能中心，主要提供传感器数据的接入、存储和展现服务，为所有的开源软/硬件爱好者、制造型企业提供一个物联网项目的平台，可使硬件和制

造业者能够在不关心服务器实现细节和运维的情况下，拥有交付物联网化电子产品的能力。

Yeelink 物联网云侧重于成为物联网的 Middleware 和 Enabler，是传统电子电器制造业者的朋友和伙伴。其平台已支持数值型、图像型、GPS 型及泛型等多种数据的接入，并提供完备的 API 文档和代码示例。通过 API 接口，用户只需要简单的几步操作就能将传感器接入 Yeelink 物联网云平台，实现传感器数据的远程监控。

Yeelink 物联网云独有设计的高并发接入服务器和云存储方案能够同时完成海量的传感器数据接入和存储任务，确保用户的数据能够安全地保存在互联网上，先进的鉴权系统和安全机制能够确保数据只在用户允许的范围内共享。

当用户的数据达到某个设定阈值的时候，Yeelink 物联网云平台会自动调用用户预先设定的规则发送短信、微博或者邮件。用户还可以充分利用平台的计算能力，定期将统计分析数据发送到邮箱内。这一切仅需要用户在网页上简单地单击几个按钮就可以实现。

Yeelink 物联网云平台的最大特点在于不仅能够提供数据的上行功能，还能够实现对家用电器的控制功能。

在 Yeelink 物联网云平台上，数据不再是孤单的节点，存储在 Yeelink 物联网云平台上的数据可以简单地被 API 取回，放置到用户的个人博客上，或者根据规则自动转发到指定的微博上，用户将会感受到数据和人之间的全面融合。

8. Xively 物联网云

Xively 物联网云是开放的通用物联网平台，可将应用、设备、虚拟电子物体、数据、用户链接在一起，创建方案，与物理世界的物体搜索、交互。Xively 物联网云提供了物体目录（物体根据权限搜索）、数据服务（时间序列数据的存储、调用）、业务服务（设备找寻、激活及管理）。为完成这些服务，Xively 物联网云建立了信息总线（实时信息的管理和路由），信息总线支持 Rest、Socket、MQTT、Xively API。

作为一个通用平台，Xively 物联网云提供基于网络的工具及丰富的开发资源，以支持硬件、应用、服务的研发，并将开发的硬件、应用、服务部署到平台上。

7.5 云服务应用开发协助工具 git

在云服务平台上开发应用时，开发者首先需要在本地使用集成开发环境（Integrated Development Environment，IDE）进行代码的开发，如 IntelliJ IDEA、Eclipse 等，然后利用分布式版本控制系统 git 将代码推送到云服务平台上。此处的推送方式有两种：一种方式是将代码托管平台作为中转站，即首先通过本地的 git 版本控制系统将代码推送到代码托管平台上，然后通过云服务平台从代码托管平台上拉取代码；另一种方式是在云服务平台上搭建 git 服务器，然后通过本地的 git 版本控制系统直接将代码推送到云服务平台上。下面分别介绍分

布式版本控制系统 git 和两种推送方式。

7.5.1 分布式版本控制系统 git

众所周知，1991 年，就读于赫尔辛基大学的 Linus 创建了开源的 Linux 操作系统。Linux 是一款免费的操作系统。开发者可以通过网络或其他途径免费获得，并可以任意修改其源代码。这是其他的操作系统做不到的。正是由于这一点，来自全世界的无数开发者都参与了 Linux 的修改和编写工作，开发者可以根据自己的兴趣和灵感对其进行改变，让 Linux 吸收了无数开发者的精华，从而不断发展壮大。截至目前，Linux 已经发展成为最大的服务器系统软件。

在 Linux 的发展期间，这么多的开发者在世界各地编写代码，Linux 代码是如何管理的呢？事实上，在 2002 年之前，世界各地的开发者把源代码文件通过 diff 的方式发送给 Linus，然后由 Linus 本人手工合成代码。此时，读者可能会问，为什么 Linus 不把 Linux 的代码放在 CVS、SVN 这些免费的版本控制系统里面呢？其原因是这些集中式的版本控制系统不但速度慢，而且必须联网才能使用，虽然当时有一些比 CVS、SVN 好用的商用版本控制系统，但是都是付费的。这与 Linux 的开源精神相悖。

2002 年，Linux 操作系统已经发展了 10 年，代码库已经十分庞大。此时，Linus 很难继续通过手工方式管理代码库，于是选择了一个商用的版本控制系统 BitKeeper。BitKeeper 的开发厂商 BitMover 授权 Linux 社区免费使用这个版本的控制系统。在之后的 3 年时间里，Linux 的开发步伐加快了两倍。

2005 年，Linux 社区中开发 Samba 的 Andrew 试图破解 BitKeeper 的协议，结果被 BitMover 公司发现，于是 BitMover 公司决定收回 Linux 社区的免费使用权。此后，Linus 大约花了两周的时间，使用 C 语言编写了 git 分布式版本控制系统，用来对 Linux 系统的源码进行管理。分布式版本控制系统的最大好处之一是在本地工作时完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而 SVN 在没有联网的时候是不会工作的！当有网络的时候，再把本地提交推送一下就完成了同步。在接下来的时间里，git 逐渐发展成为最流行的分布式版本控制系统，尤其是 2008 年，为开源项目免费提供 git 存储的 GitHub 网站上线了，更加促进了 git 的快速发展。

1. git 在多平台上的安装

在最开始的时候，由于 git 是在 Linux 系统上开发的，所以在相当长的一段时间内，git 只能在 Linux 和 Unix 系统上运行，之后，开发者将 git 移植到 Windows 和 Mac 系统上。现如今，git 可以在 Linux、Unix、Mac、Windows 系统上正常运行。

(1) 在 Linux 系统上安装 git

当在 Linux 系统上安装 git 时，首先应该做的是在终端输入“git”查看系统是否已经安

装 git，如果没有安装 git，则 Linux 系统会告诉用户没有安装 git，而且会给出安装 git 的命令语句。

如果使用的是 Debian 或者 Ubuntu Linux 系统，则只需要使用“sudo apt-get install git”语句即可完成 git 的安装；如果使用的是其他版本的 Linux，则可以直接通过在 git 官网上下载源码进行安装，将源码进行解压后，在终端依次输入“./config”“make”“sudo make install”这几个命令进行安装。

（2）在 Mac 系统上安装 git

当在 Mac 系统上进行开发时，因为 Xcode 中集成了 git，所以可以在 AppStore 中安装 Xcode，然后运行 Xcode 后，依次选择菜单“Xcode”→“Preferences”，在弹出的窗口中单击“Download”，选择最上面的“Command Line Tools”后，单击“install”即可完成 git 在 Mac 系统上的安装。

（3）在 Windows 系统上安装 git

在 Windows 系统上安装 git 时，只需要下载 msysgit 安装程序，下载地址是 https://git-for-windows.github.io 或者 https://gitforwindows.org，然后按照默认选项安装即可。安装完成后，在开始菜单中单击“git”→“git Bash”后弹出一个窗口，类似于命令行窗口，说明 git 在 Windows 系统上已经安装成功，如图 7.1 所示。

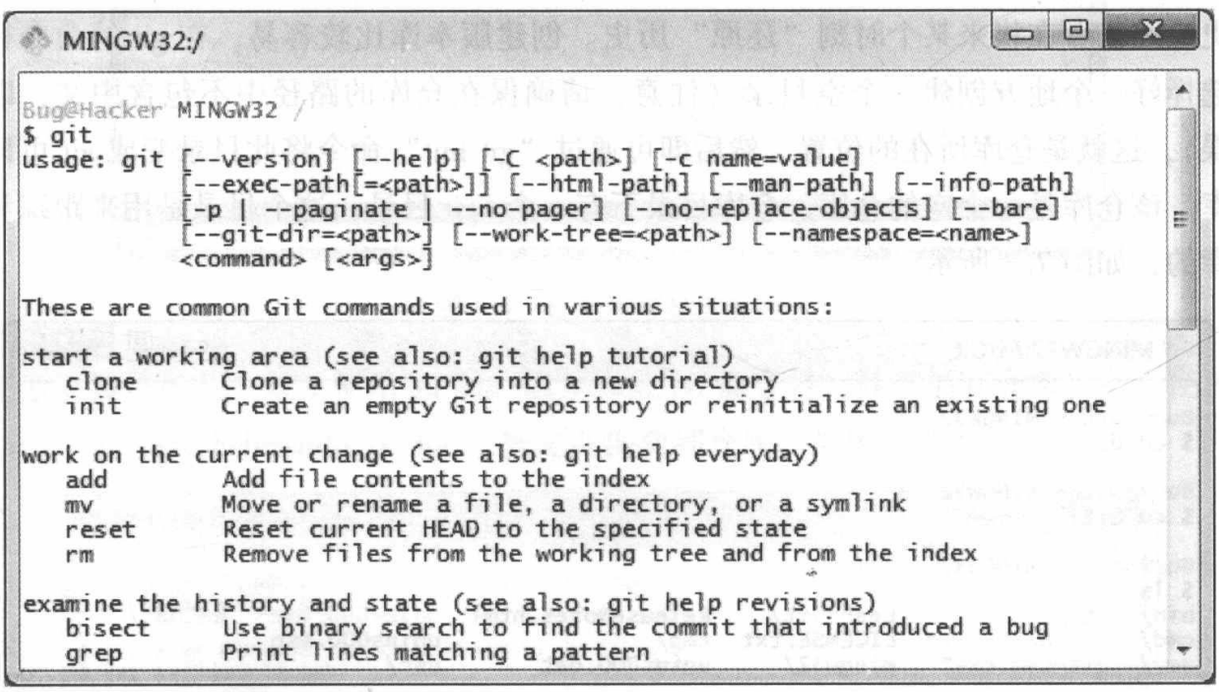


图 7.1　git 命令行窗口

由于 git 是分布式版本控制系统，因此每一个用户都需要登记自己的用户名和 E-mail 地址，在命令行中输入以下命令：

```
$git config --global user.name "Your Name"
git config --global user.email"you@example.com"
```

登记用户名和 E-mail 地址界面如图 7.2 所示。

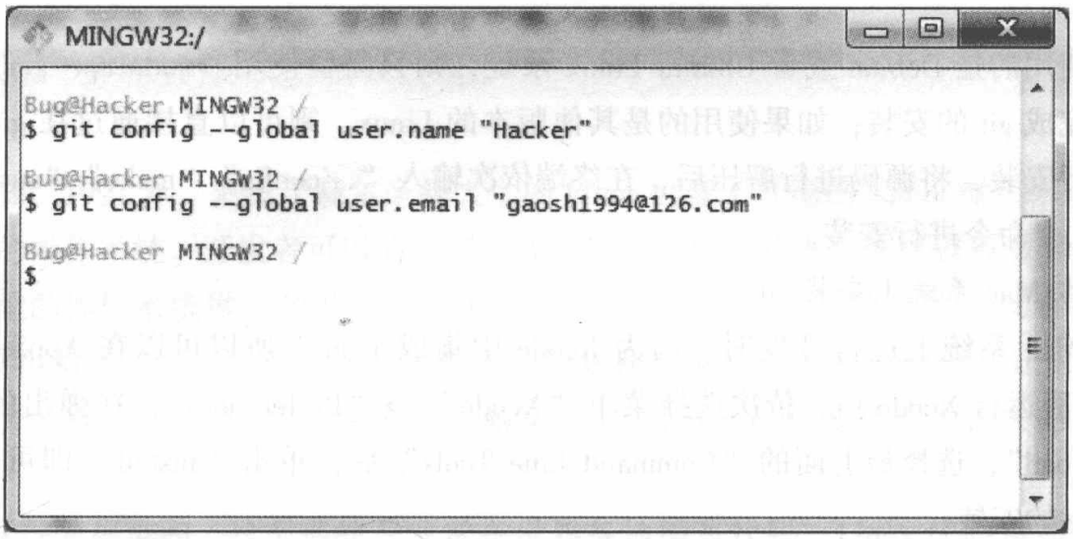


图 7.2　登记用户名和 E-mail 地址界面

2. 版本库的建立

版本库又被称为仓库（repository）。形象地理解，它就是一个目录，在此目录里面的所有文件都可以被 git 所管理。git 能够跟踪此目录中文件的修改或者删除，以便在任意时刻进行历史追踪或者在将来某个时刻“还原”历史。创建版本库比较容易，在 git 的命令行窗口中先选择好一个地方创建一个空目录（注意，请确保在仓库的路径中不包含中文，以免出现错误），这就是仓库所在的位置，然后即可通过“git init”命令将此目录变成 git 可以管理的仓库。该仓库是一个空的仓库。仓库目录下有一个 .git 目录。这个目录是用来跟踪和管理版本库的，如图 7.3 所示。

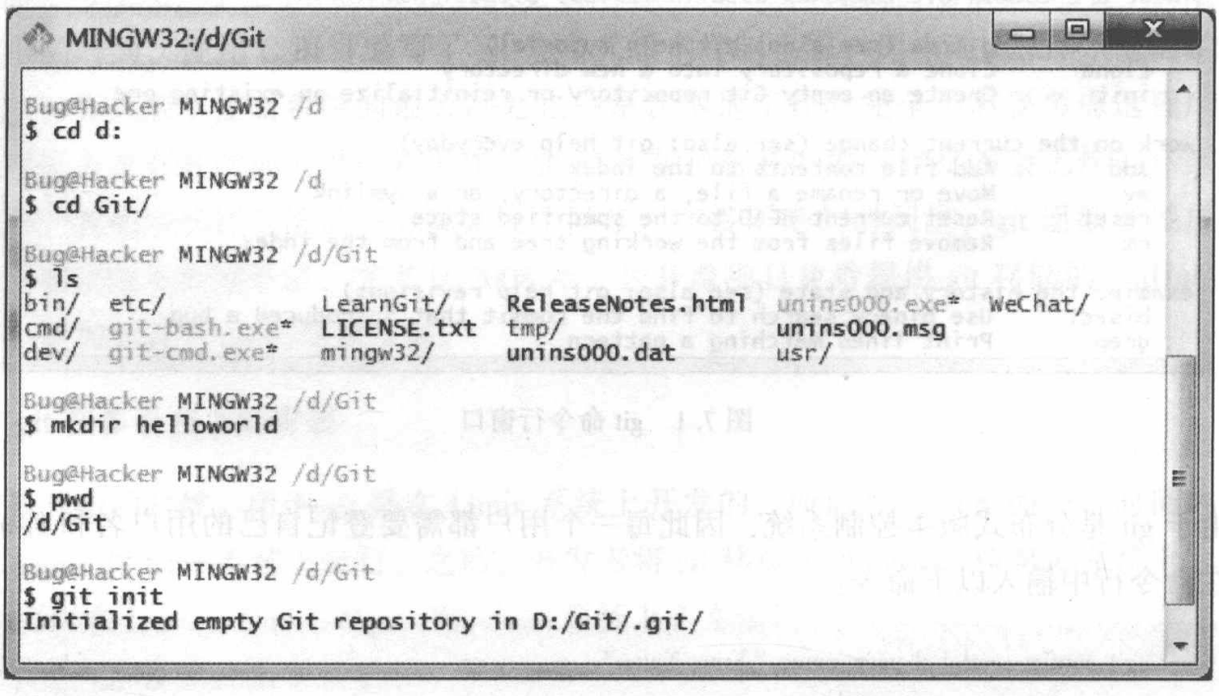


图 7.3　创建仓库

3. 把文件添加到版本库

git 等版本控制系统跟踪的是文本文件的改动，如 txt 文件、网页、程序代码等，可以告诉开发者每次进行的改动，如增添了某行代码或删除了某个单词。对于图片、视频、Word 文档等二进制文件，虽然可以放到版本控制系统中进行管理，但是没有办法跟踪文件的变化，如开发者改动了图片的内容，图片大小由 900 KB 变成了 1 000 KB，则版本控制系统只能把二进制文件的每次改动串接起来，只知道图片大小由 900 KB 变成了 1 000 KB，但是具体改动了什么内容，版本控制系统没有办法知道。

在 git 的命令行窗口中切换至上述 git 仓库目录下，使用 vi 命令新建 helloworld. txt 文件，按“i”键切换至插入模式，输入“welcome to git!”，按“Esc”键从插入模式切换到编辑模式，在编辑模式下键入“:”后，光标就会跳到屏幕的最后一行，并在那里显示冒号，此时已进入命令模式，命令模式又被称为“末行模式”，开发者输入的内容均显示在屏幕的最后一行，在命令模式下输入“wq”，按下回车键后，对文件进行保存并且退出，如图 7.4 所示。

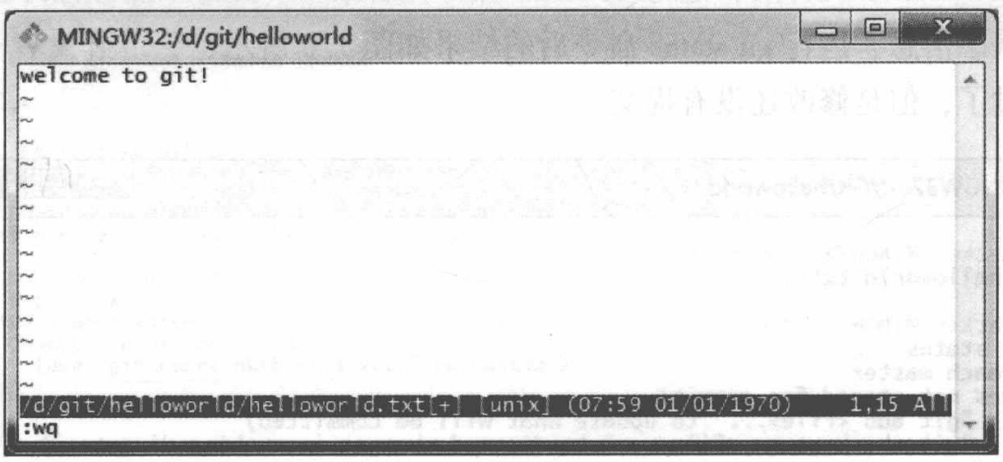


图 7.4 在版本库新建文件

之后要操作的首先就是使用 git add helloworld. txt 命令将文件添加到仓库，然后使用命令 git commit -m “add helloworld. txt file” 把文件提交到仓库，如图 7.5 所示。

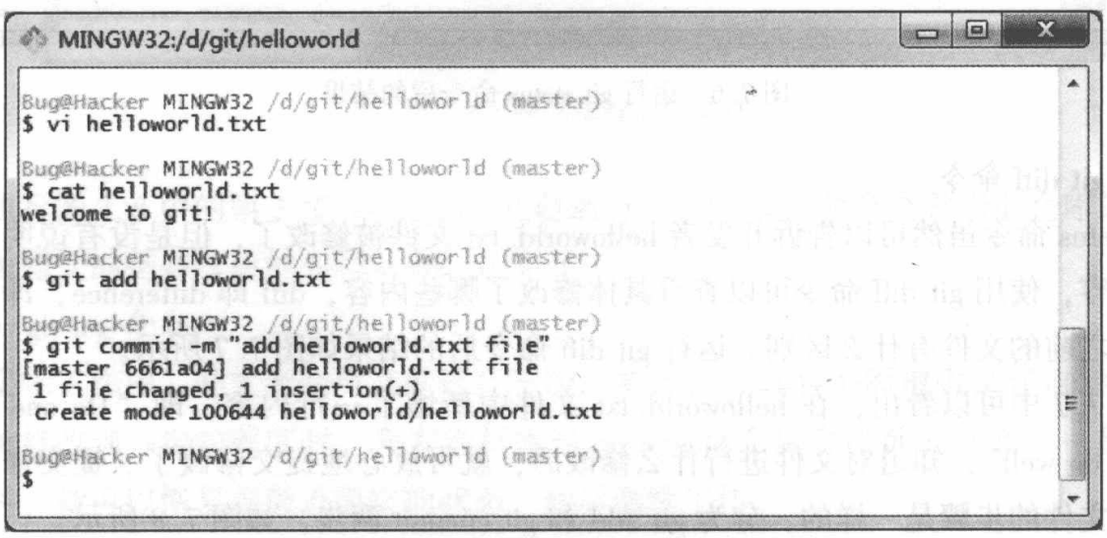


图 7.5 将文件添加并提交到仓库

git commit 命令“-m”后面输入的是本次提交的说明，一般输入有意义的内容，开发者就能够方便地从历史记录里找到改动的记录。git commit 命令执行成功后，显示“1 file changed”，即表示新添加了一个文件 helloworld. txt，“1 insertion(+)”，即表示 helloworld. txt 文件中新插入了 1 行内容。

由于 git 可以使用 commit 命令一次提交很多文件，因此开发者可以多次 add 不同的文件，最后再统一执行 commit 命令，如

```
$git add file1. txt
$git add file2. txt file3. txt
$git commit -m "add 3 files"
```

4. 版本库的管理

（1）git status 命令

修改 helloworld. txt 文件后，运行 git status 命令查看结果。此命令能够让开发者时刻了解仓库现在的状态信息，运行 git status 命令后的结果如图 7.6 所示，告诉开发者 helloworld. txt 文件被修改过了，但是修改还没有提交。

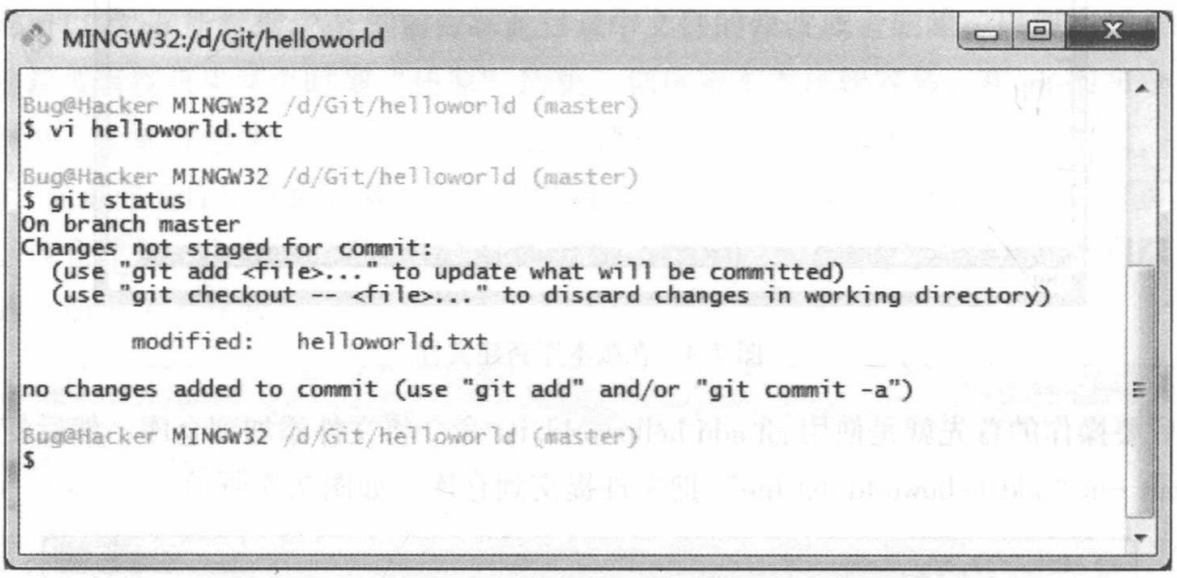


图 7.6　运行 git status 命令后的结果

（2）git diff 命令

git status 命令虽然可以告诉开发者 helloworld. txt 文件被修改了，但是没有说明具体修改了什么内容，使用 git diff 命令可以查看具体修改了哪些内容，diff 即 difference，即查看现在的文件与之前的文件有什么区别。运行 git diff 命令后的结果如图 7.7 所示。

从图 7.7 中可以看出，在 helloworld. txt 文件中新增了一行内容，即“Do one thing at a time, and do well”，知道对文件进行什么修改后，就可放心地提交修改了。提交修改的步骤和提交新文件的步骤是一样的，分为 git add 和 git commit 两步，如图 7.8 所示。

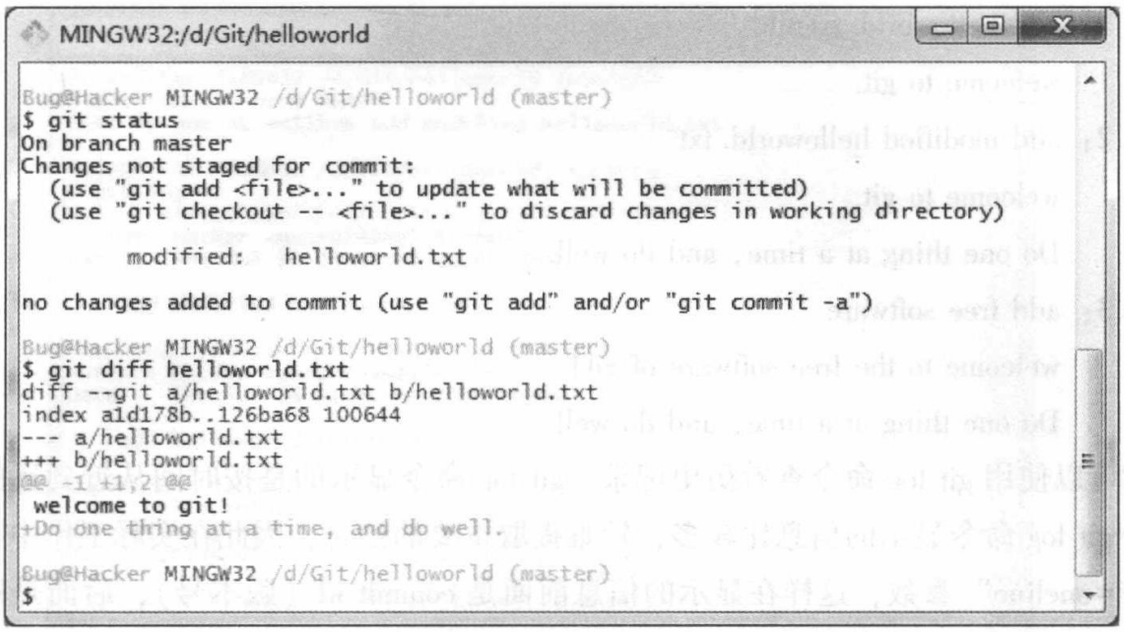


图 7.7　运行 git diff 命令后的结果

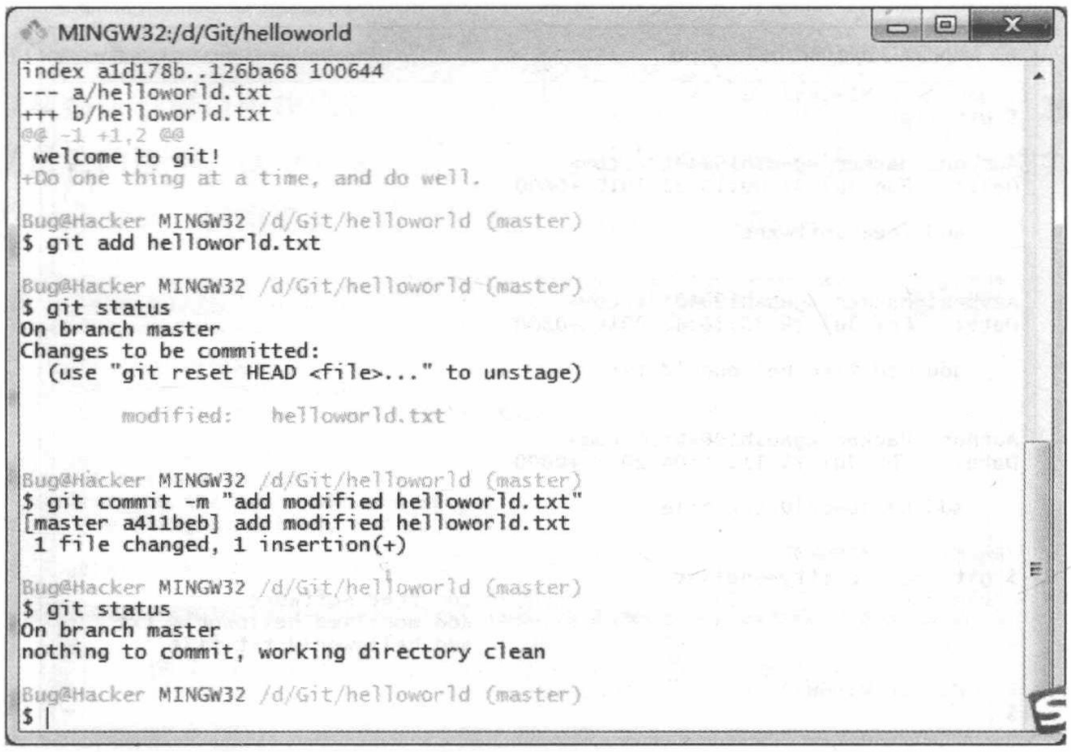


图 7.8　提交修改后的文件

在执行图 7.8 中的第 2 步 git commit 之前运行了 git status 命令查看当前仓库的状态，显示将要被提交的是修改后的 helloworld. txt 文件。

（3）git log 命令

在实际工作中，开发者会对文件进行不断的修改，并且会不断地提交修改到版本库中。当文件被修改到一定的程度时，开发者将通过 commit 命令保存当前的状态，一旦改错或误删了文件，就可以恢复到最近提交的状态，然后继续工作。

上述的 helloworld. txt 文件共有以下 3 种状态被提交到 git 仓库中。

状态 1：add helloworld. txt file

welcome to git.

状态 2：add modified helloworld. txt

welcome to git.

Do one thing at a time, and do well.

状态 3：add free software

welcome to the free software of git!

Do one thing at a time, and do well.

此时可以使用 git log 命令查看历史记录。git log 命令显示的是按时间从近到远的提交日志。由于 git log 命令显示的信息比较多，较难提取重要的信息，因此在实际工作中可以加上“--pretty=oneline”参数，这样在显示的信息前面是 commit id（版本号），后面是提交的说明。需要注意的是，加参数的 git log 命令显示的也是按时间从近到远的提交日志，如图 7.9 所示。

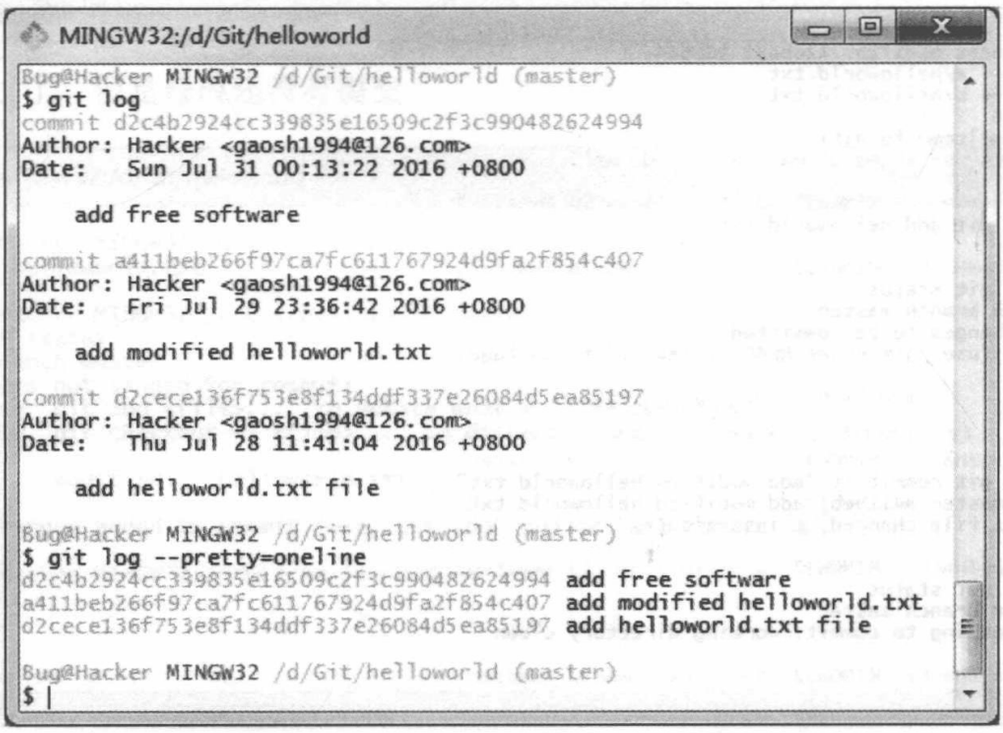


图 7.9　git log 命令查看历史记录

(4) 回退和恢复版本

如果开发者发现在最后提交的版本中有错误或者存在误删数据，想要恢复到上一个版本，那么“add modified helloworld. txt”版本可以使用 git reset 命令进行回退。由于 git 用 HEAD 表示最新提交的版本，因此倒数第二个版本是 HEAD^，倒数第三个版本是 HEAD^^，倒数第 n 个版本是 HEAD~n。回退到上一个版本的操作如图 7.10 所示。

在图 7.10 中运行了 git log 命令查看版本库的状态，发现此时最新的版本“add free software”已经消失了，下面分两种情况讨论如何恢复到最新的版本。

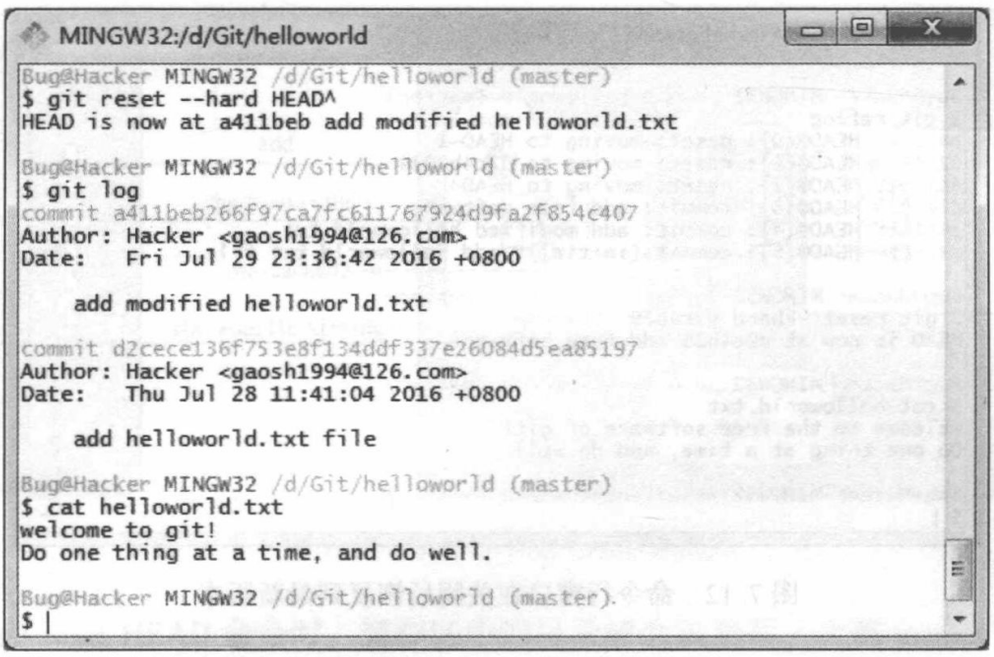


图 7.10 回退到上一个版本的操作

① 开发者回退到某个版本，但命令行窗口未关闭。

此时可以从命令行窗口中找到版本“add free software”的 commit id 为 d2c4b2924...，于是可以恢复到最新的版本。命令行窗口在未关闭的情况下恢复到最新版本如图 7.11 所示。此时的版本号不必写全，只需要给出前几位即可，git 可以自动查找。

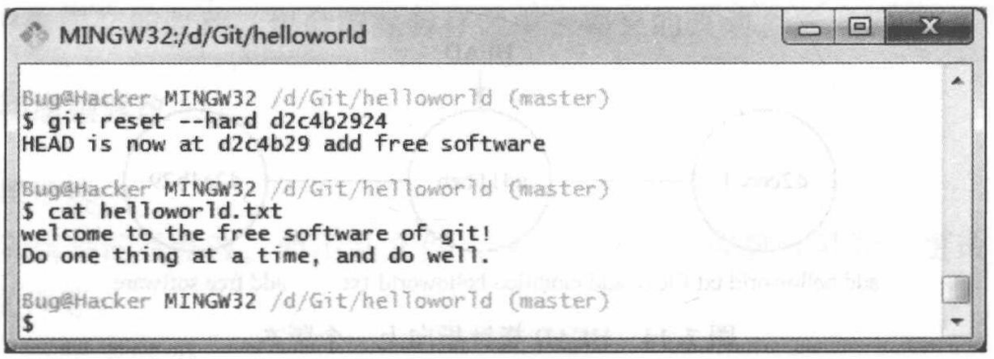


图 7.11 命令行窗口在未关闭的情况下恢复到最新版本

② 开发者回退到某个版本，且命令行窗口关闭。

此时可以通过 git reflog 命令寻找新版本的 commit id，找到版本号之后，即可通过第一种方式恢复到最新的版本。命令行窗口在关闭后恢复到最新版本如图 7.12 所示。

需要说明的是，git 的版本回退速度很快，因为在 git 内部有一个 HEAD 指针指向当前版本，所以当进行回退版本或者恢复新版本时，git 只需要移动 HEAD 指针的指向即可，然后更新工作区的文件。HEAD 指针指向最新版本如图 7.13 所示。

当运行 git reset --hard HEAD^命令后，HEAD 指针指向上一个版本，如图 7.14 所示。

5. 工作区和版本库

工作区和版本库的原理图如图 7.15 所示。图中，左侧为工作区，就是在电脑中能够看到

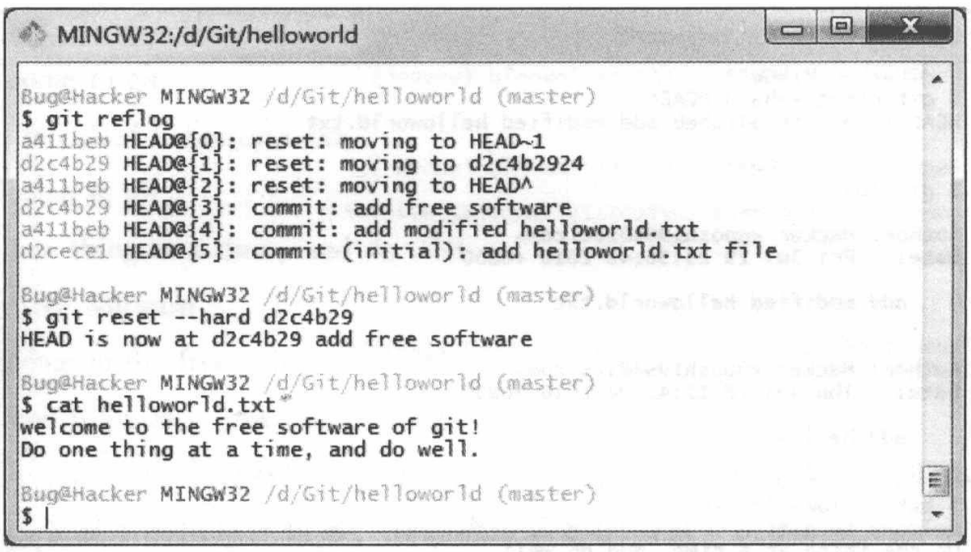


图 7.12　命令行窗口在关闭后恢复到最新版本

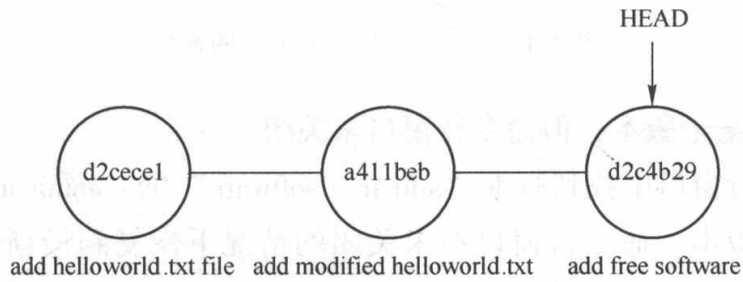


图 7.13　HEAD 指针指向最新版本

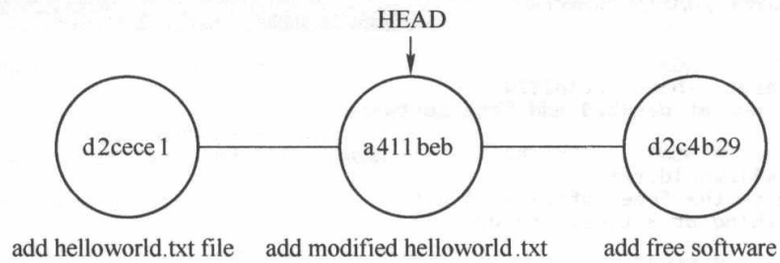


图 7.14　HEAD 指针指向上一个版本

的目录（如上面提到的 helloworld 文件夹就是一个工作区）；右侧为版本库（工作区中的隐藏目录 .git 就是 git 的版本库）；在版本库中标记为 index 的区域为暂存区，git 与其他版本控制系统（如 SVN）的一个不同之处就是有暂存区的概念；标记为 master 的是 master 分支所代表的目录树。可以看出，此时的 HEAD 实际上是指向 master 分支的一个“游标”，命令中出现 HEAD 的地方可以用 master 来替换。图中的 objects 区域为 git 的对象库，位于 .git/objects 目录下。

当对工作区修改或新增的文件执行 git add 命令时，实际上就是把文件的修改添加到暂存区中，暂存区中的目录树会被更新，同时工作区中修改或新增的文件内容会被写入对象库中的一个新的对象中，而该对象的 id 被记录在暂存区中的文件索引中。

当执行提交操作 git commit 时，实际上就是把暂存区中的所有内容提交到当前分支，暂存区中的目录树会被写到对象库中，master 分支会进行相应的更新，即 master 最新指向的目录树就是提交时原暂存区的目录树。

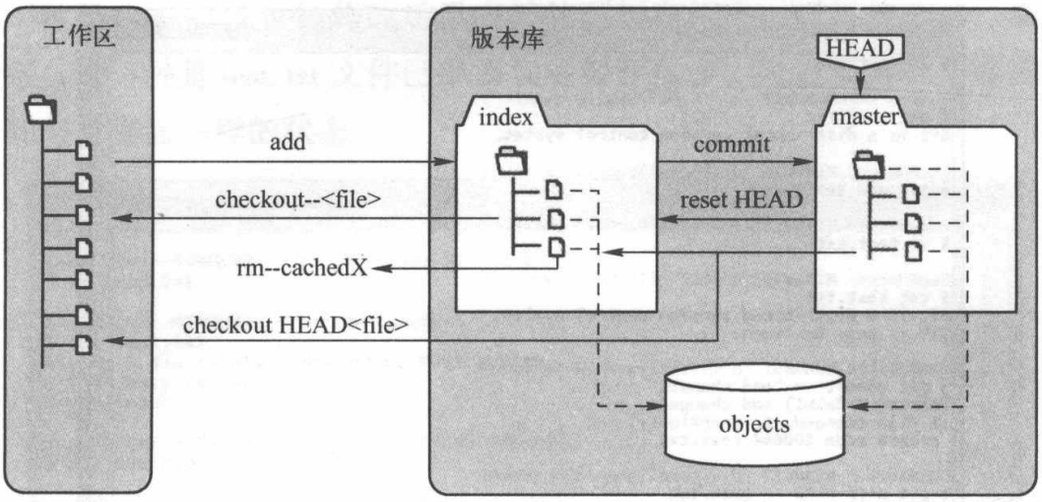


图 7.15　工作区和版本库的原理图

当执行 git reset HEAD 命令时，暂存区中的目录树会被重写，会被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 git rm --cached 命令时，会直接从暂存区中删除文件，工作区则不改变。

当执行 git checkout . 或 git checkout --命令时，会用暂存区中全部的文件或指定的文件替换工作区中的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 git checkout HEAD . 或 git checkout HEAD 命令时，会用 HEAD 指向的 master 分支中的全部或部分文件替换暂存区和工作区中的文件，此命令也是极度危险的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

6. 管理和撤销修改

(1) 管理修改

git 跟踪和管理的是修改，而不是文件。修改就是新增或删除某行、更改某些字符或者创建一个新文件等。

首先用一个例子说明。

新建 test. txt 文件，在文件中新增一行，运行 git add 命令，然后新增一行内容，之后运行 git commit 提交修改，即第一次修改→git add→第二次修改→git commit，最终运行 git diff HEAD -- test. txt 命令查看工作区中和版本库里最新版本的区别后，发现第二次的修改并没有被提交，如图 7. 16 所示。

出现上述结果的原因是，git 管理的是修改，当运行 git add 命令时，在工作区中的第一次修改被放入暂存区中准备提交，但是第二次的修改并没有放入暂存区中，当运行 git commit 命令时只负责提交暂存区中的修改，也就是说，只把第一次的修改提交了，而第二次的修改没有被提交。

如何将第二次的修改顺利提交呢？可以继续运行 git add 和 git commit 命令，或者在第二次的修改之后，运行 git add 命令将第二次的修改放入暂存区中，然后运行 git commit 命令提交两次的修改，即第一次的修改→git add→第二次的修改→git add→git commit，如图 7. 17 所示。

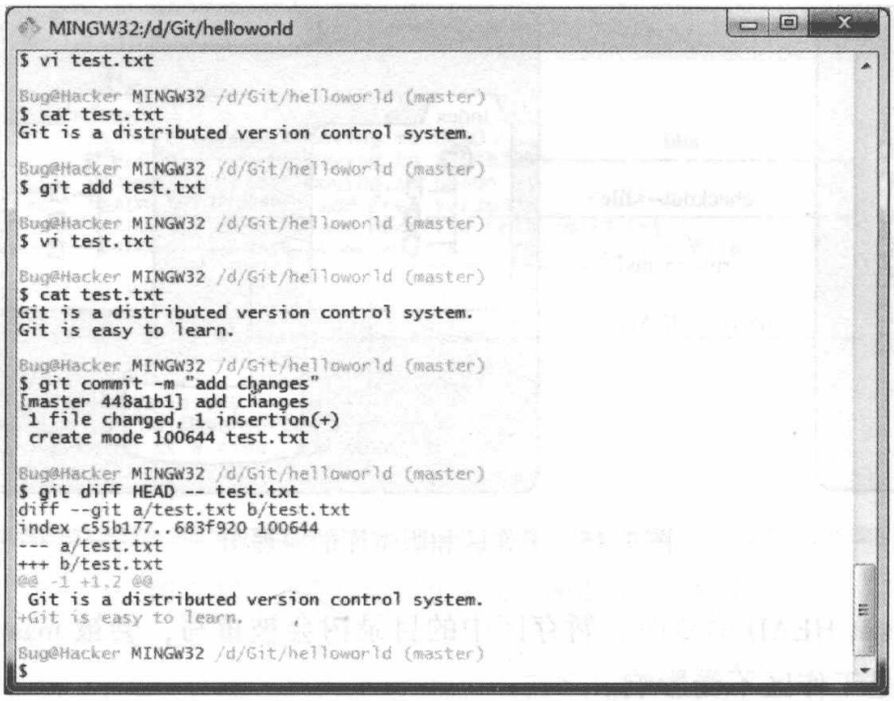


图 7.16　git 管理修改

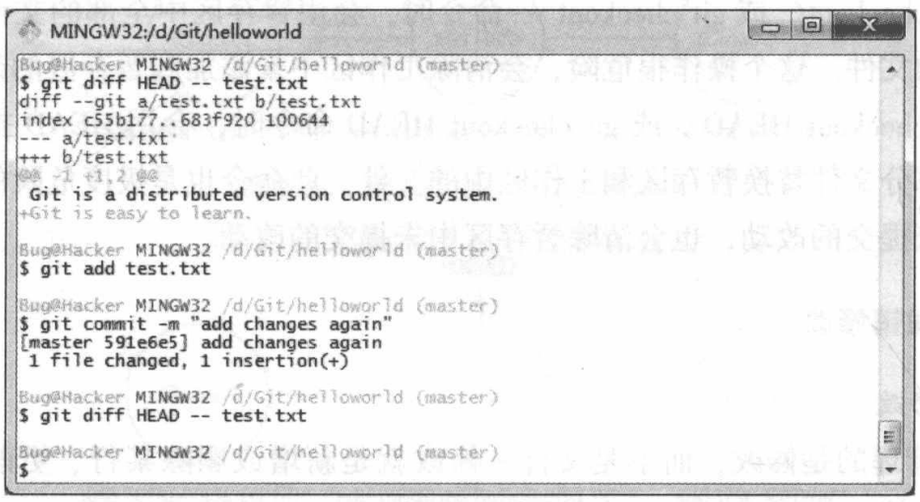


图 7.17　提交第二次的修改

(2) 撤销修改

现假设四种情况：

第 1 种情况：改乱了工作区中某个文件的内容，想直接丢弃工作区中的修改怎么办？

第 2 种情况：不但改乱了工作区中某个文件的内容，还添加到了暂存区中，想要丢弃修改怎么办？

第 3 种情况：已提交了不合适的修改到版本库，并且没有推送到远程库，想要撤销本次提交怎么办？

第 4 种情况：删除了本地的某个文件，想要恢复删除的文件怎么办？

对于第 1 种情况，当开发者在 test. txt 文件中增添了一行错误信息，并且想删除此行错误信息，则可以删掉最后一行，手动把文件恢复到上一个版本的状态，也可以使用“git checkout -- file”命令丢弃工作区中的修改，如图 7. 18 所示。这里丢弃工作区中的修改分为

两种情况：一种是 test. txt 文件被修改后没有被放到暂存区中，撤销修改后，就回到与版本库一样的状态；另一种是 test. txt 文件已经添加到暂存区中，之后又进行了修改，则撤销修改后，就回到与暂存区一样的状态。

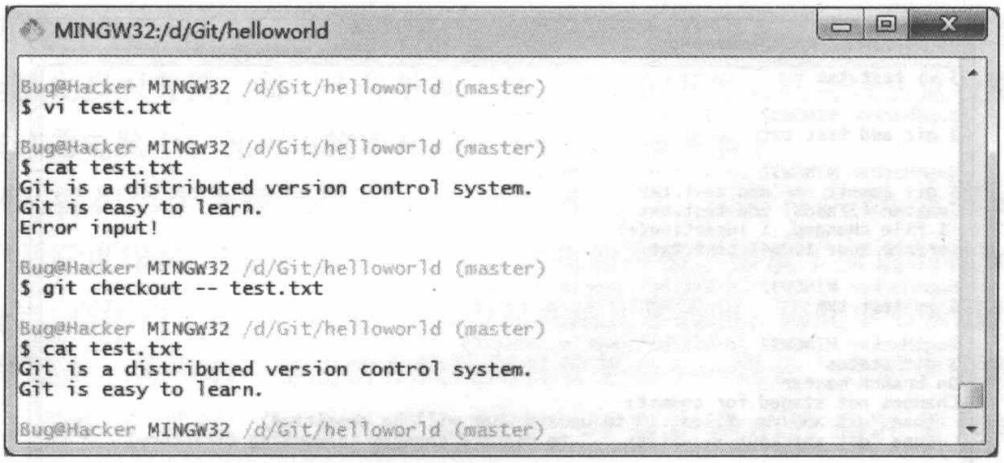


图 7.18　使用“git checkout -- file”命令丢弃工作区中的修改

对于第 2 种情况，开发者在 test. txt 文件中增添了一行错误信息，并且提交到了暂存区中，此时可以使用“git reset HEAD file”命令撤销暂存区中的修改，将其重新放回到工作区中，如图 7. 19 所示。

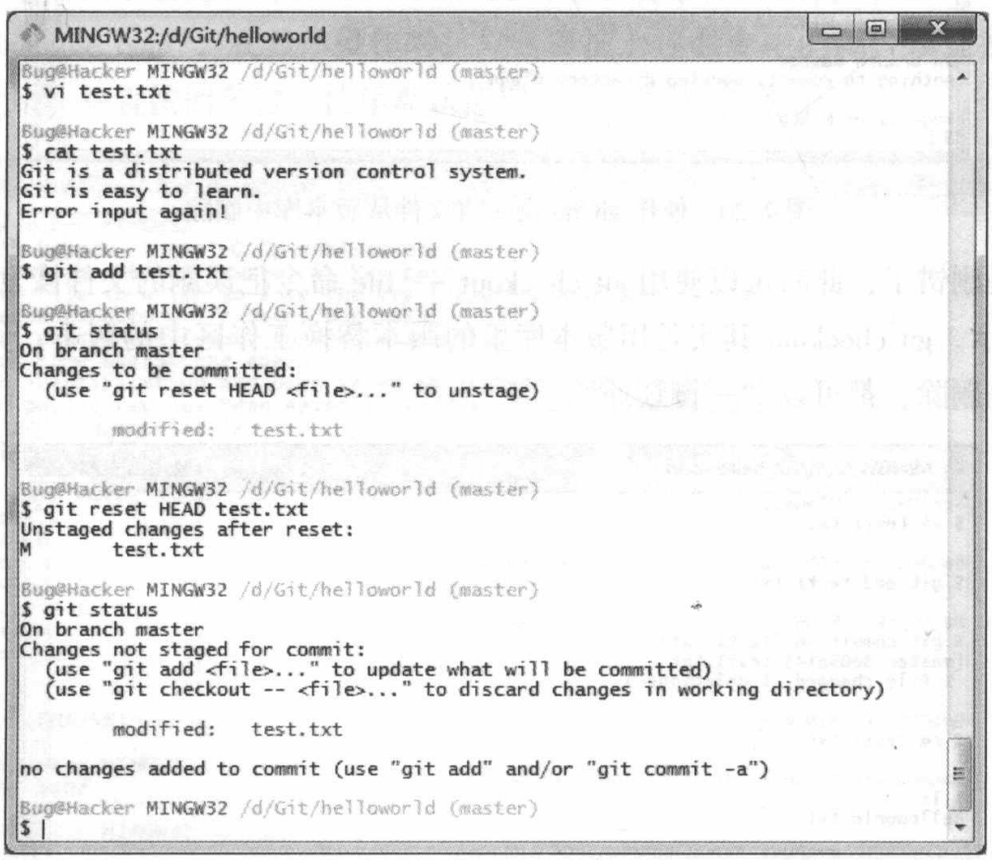


图 7.19　使用“git reset HEAD file”命令撤销暂存区中的修改

对于第 3 种情况，开发者不但改错了内容，还从暂存区中提交到了版本库，但没有推送到远程库，此时可以通过版本回退的方法回退到上一个版本。

对于第 4 种情况，当开发者将 test. txt 文件删除，运行 git status 命令后，git 便会告诉开

发者哪些文件被删除了，有两种情况：

一种是确定要在版本库中删除该文件，此时可以使用 git rm 命令，并且 git commit 即可将其在版本库中删除，如图 7.20 所示。

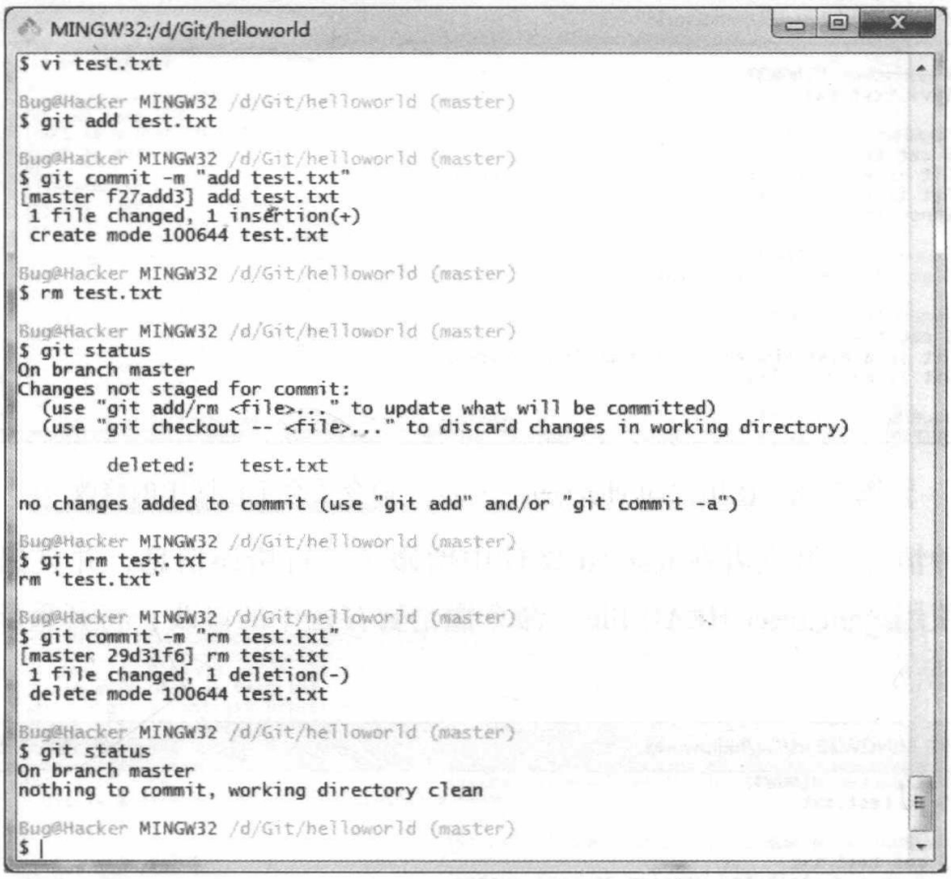


图 7.20　使用 git rm 命令将文件从版本库中删除

另一种是删错了，此时可以使用 git checkout -- file 命令把误删的文件恢复到最新版本，如图 7.21 所示。git checkout 其实是用版本库里的版本替换工作区中的版本，无论在工作区中是修改还是删除，都可以“一键还原”。

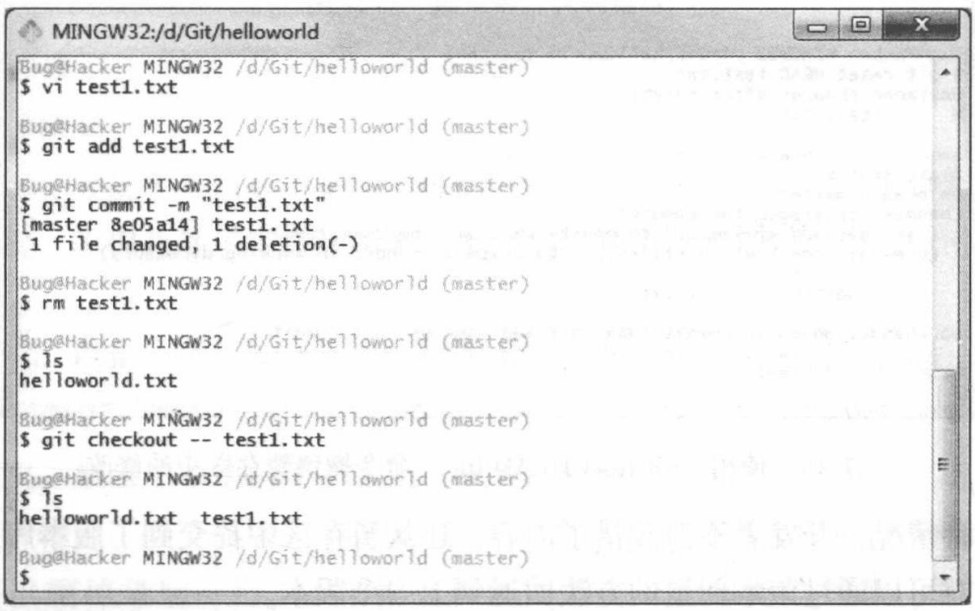


图 7.21　使用 git checkout -- file 命令恢复误删的文件

7.5.2 推送方式一：代码托管平台作为中转站

git 是分布式版本控制系统。同一个 git 仓库可以分布在不同的电脑上。在实际情况中，可以找一台电脑充当服务器，每天不停歇地工作，所有的开发者既可以从这台服务器上将版本库克隆到自己的电脑上，又可以把本地的仓库推送到服务器上。

GitHub 网站就相当于上述服务器，于 2008 年上线，用来提供 git 仓库的托管服务。GitHub 的用户活跃度很高，在开源世界里享有很高的声望，形成了所谓的社交化编程文化(Social Coding)。在国内，由于 GitHub 网站有时无法正常访问，因此本书采用开源中国推出的基于 git 的快速的、免费的、稳定的在线代码托管平台——码云。开源中国的一个账号最多可以创建 1 000 个项目，包含公有和私有。开源中国的代码托管地址为 http://git.oschina.net，只要注册一个开源中国账号，就可以免费获得 git 远程仓库。

1. SSH 密钥

由于开发者的本地 git 仓库和远程仓库之间的传输是通过 SSH 加密的，因此在使用远程仓库之前需要进行以下设置：打开 Git Bash，使用命令创建 SSH Key：$ssh-keygen -t rsa -C "youremail@example.com"，在具体操作时，只需要把上述命令中的邮件地址换成开发者自己的邮件地址后，一直按回车键，使用默认值即可，如图 7.22 所示。

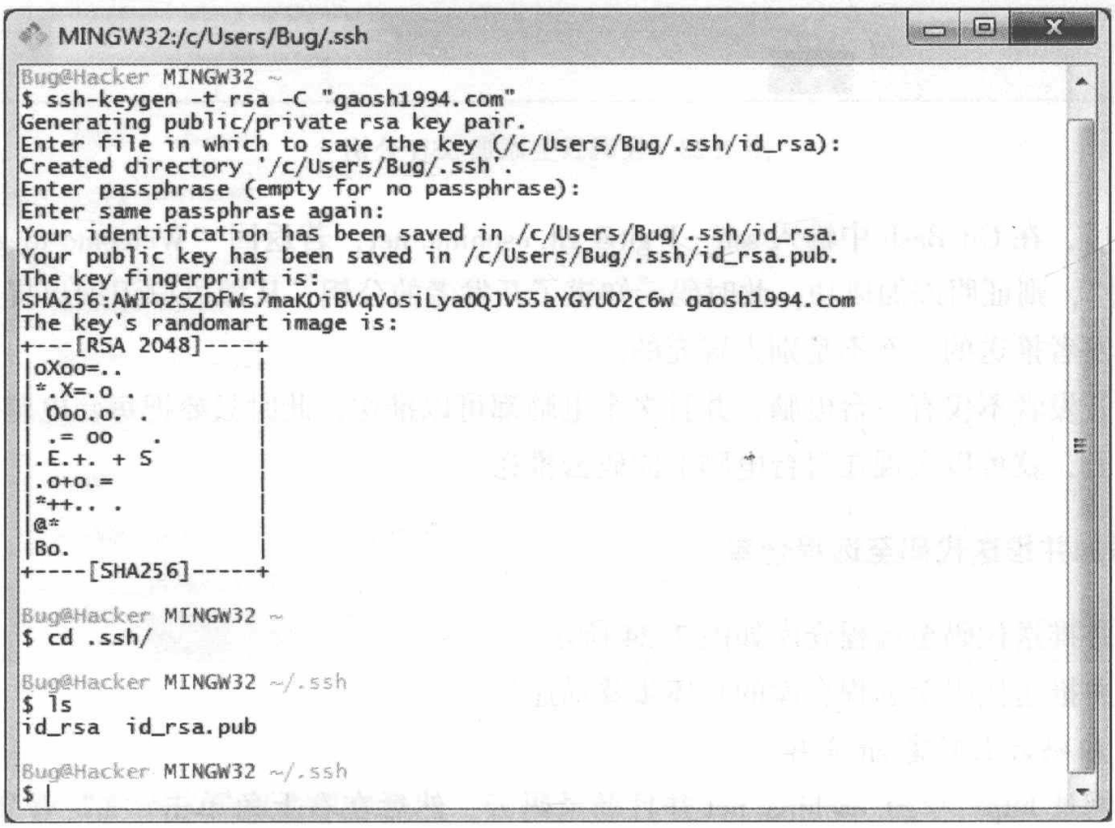


图 7.22 创建 SSH Key

此时可以看到开发者的目录里出现了 .ssh 目录，里面有 id_rsa 和 id_rsa.pub 文件。这两个文件就是 SSH Key 的密钥对：id_rsa 是私钥，不能泄露出去；id_rsa.pub 是公钥。

在码云上添加 SSH 公钥：登录码云，打开“个人资料”中的“SSH 公钥”页面后，填上有含义的标题，在 Key 文本框里粘贴 id_rsa.pub 文件的内容，如图 7.23 所示。

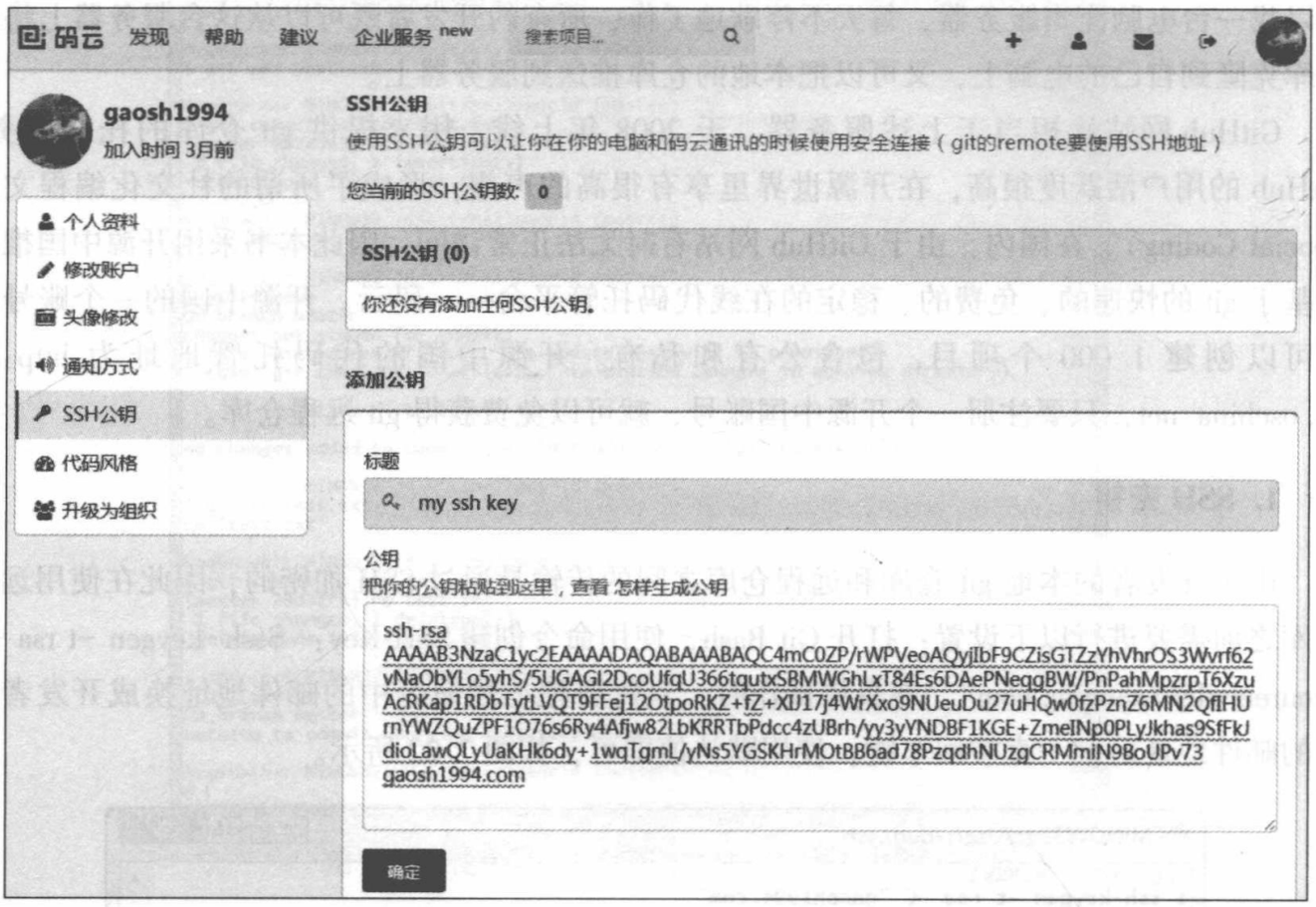


图 7.23　在码云上添加 SSH 公钥

添加后，在 Git Bash 中输入 ssh -T git@git.oschina.net，若返回“Welcome to git@OSC, yourname!”，则证明添加成功。此时码云知道了开发者的公钥，从而可以识别出提交的内容确实是开发者推送的，而不是别人冒充的。

假设开发者不仅有一台电脑，并且多个电脑都可以推送，此时只要把每台电脑的 Key 都添加到码云，就可以实现在每台电脑上向码云推送。

2. 添加并推送代码至远程仓库

添加并推送代码至远程仓库如图 7.24 所示。

添加并推送代码至远程仓库的具体步骤描述如下。

(1) 在码云上创建 git 仓库

打开网址 https://git.oschina.net 并且登录码云，然后在右上角单击“+”号新建项目，在“项目名”中填入 helloworld，其他保持默认选项，最后单击“创建”按钮就可成功地创建了一个新的 git 仓库，如图 7.25 所示。

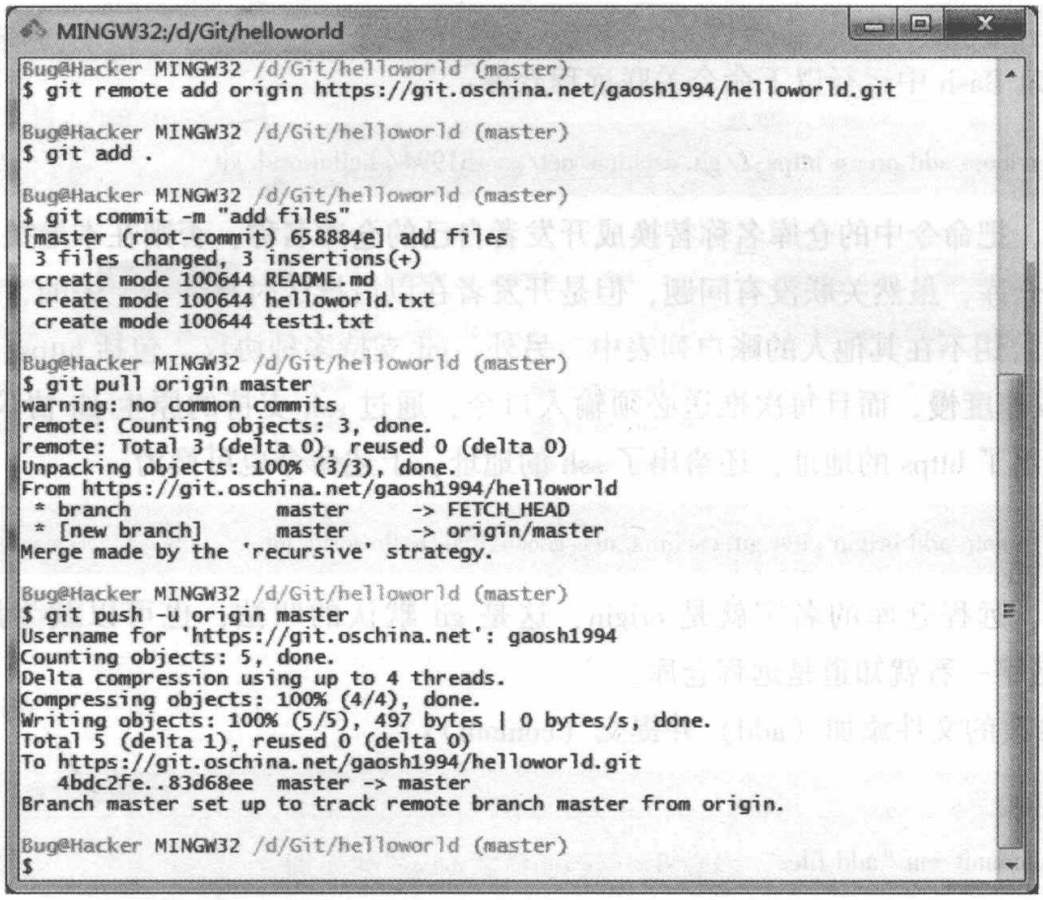


图 7.24　添加并推送代码至远程仓库

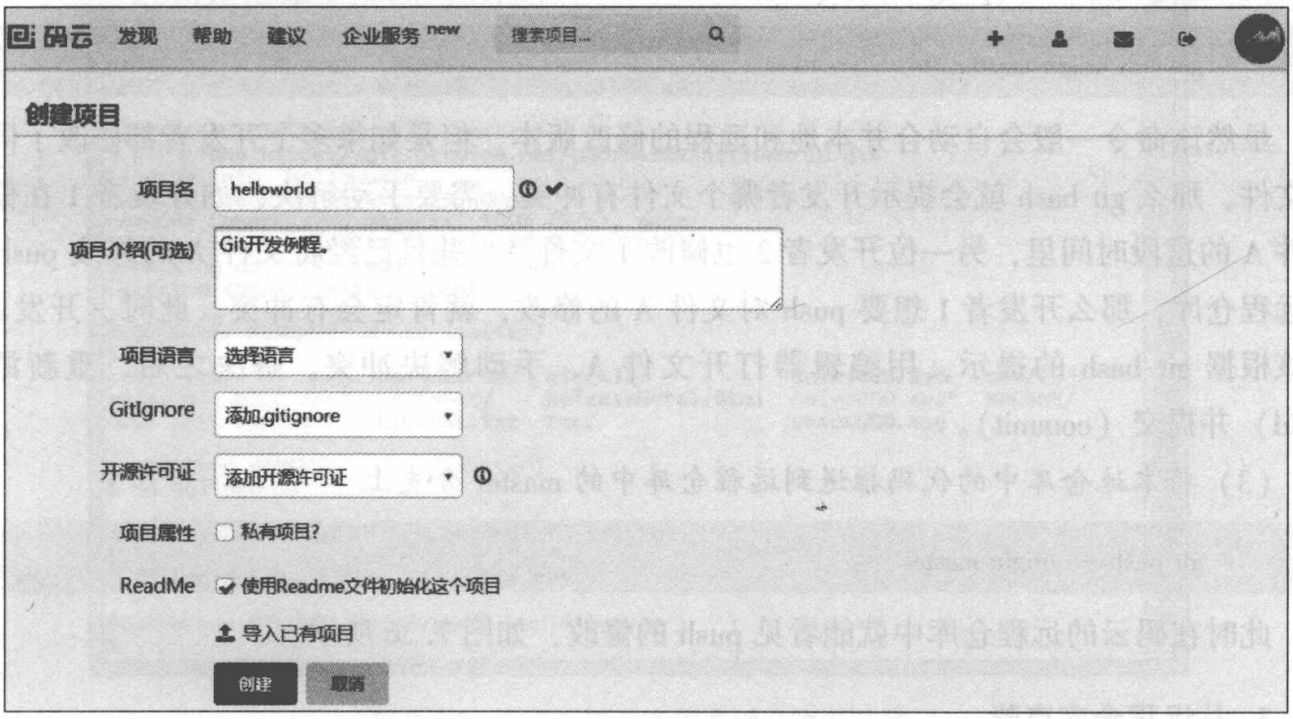


图 7.25　在码云上创建 git 仓库

(2) 推送本地代码到远程仓库

因为在本地已经创建了一个 git 仓库，所以想让这两个仓库进行远程同步，从而实现码云上的仓库既可以作为备份，又可以让其他人通过该仓库来协作。同步本地仓库和远程仓库

的步骤如下。

① 在 Git Bash 中运行以下命令关联远程仓库：

```
git remote add origin https://git. oschina. net/gaosh1994/ helloworld. git
```

请注意，把命令中的仓库名称替换成开发者自己的仓库名称，否则在本地关联的就是其他人的远程仓库，虽然关联没有问题，但是开发者在以后推送时是推不上去的，因为开发者的 SSH Key 公钥不在其他人的账户列表中。另外，git 支持多种协议，包括 https、ssh 等，使用 https 不仅速度慢，而且每次推送必须输入口令，通过 ssh 支持的原生 git 协议速度最快。码云不仅给出了 https 的地址，还给出了 ssh 的地址，上述命令也可换为

```
git remote add origin git@ git. oschina. net:gaosh1994/helloworld. git
```

添加后，远程仓库的名字就是 origin，这是 git 默认的叫法，也可以改为别的，但是 origin 这个名字一看就知道是远程仓库。

② 将修改的文件添加（add）并提交（commit）：

```
git add.
git commit -m "add files"
```

③ 因为在提交代码前需要先与仓库中已有的代码保持一致，所以需要先把远程仓库中代码的最新版本拉取（pull）下来：

```
git pull origin master
```

虽然该命令一般会自动合并本地和远程的修改版本，但是如果多个开发者都修改了同一个文件，那么 git bash 就会提示开发者哪个文件有冲突，需要手动解决，如开发者 1 在修改文件 A 的这段时间里，另一位开发者 2 也修改了文件 A，并且已经将文件 A 的修改 push 到了远程仓库，那么开发者 1 想要 push 对文件 A 的修改，就肯定会有冲突。此时，开发者 1 应该根据 git bash 的提示，用编辑器打开文件 A，手动解决冲突。解决之后，重新添加（add）并提交（commit）。

（3）将本地仓库中的代码推送到远程仓库中的 master 分支上

```
git push -u origin master
```

此时在码云的远程仓库中就能看见 push 的修改，如图 7.26 所示。

3. 从远程仓库克隆

前面讲到的是先有本地仓库，然后有远程仓库，最后介绍了本地仓库如何关联远程仓库。现在假设从零开发，先创建远程仓库，再从远程仓库克隆。克隆远程仓库到本地仓库的步骤如图 7.27 所示。首先选择合适的目录，然后运行 git clone 命令即可克隆相应的远程仓库到本地仓库中。

图 7.26　push 之后的码云远程仓库

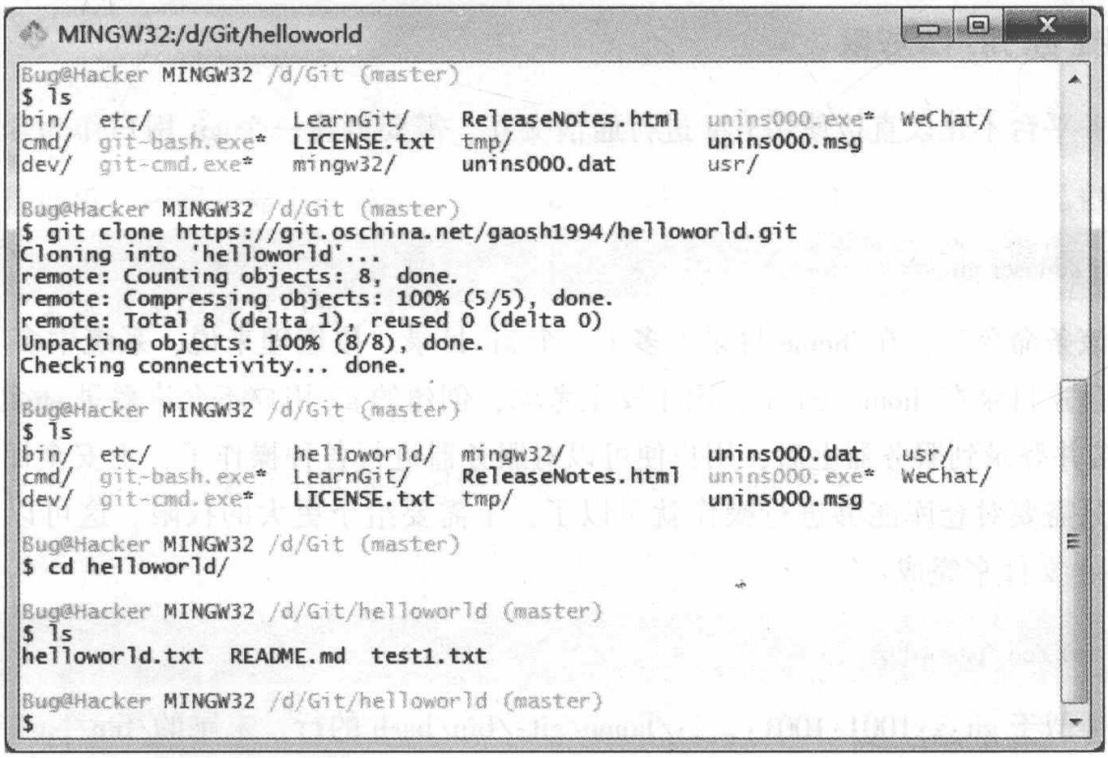


图 7.27　克隆远程仓库到本地仓库的步骤

另外，在本地仓库中使用 Xshell 软件连接云服务平台后，运行 git clone 命令即可将远程仓库克隆到云服务平台上，从而实现应用代码在云服务平台上的部署。

7.5.3 推送方式二：在云服务平台上搭建 git 服务器

1. 在云服务平台上安装 git

（说明：下述操作都是以 root 用户登录服务器的，目的在于演示方便。）

在本地仓库中使用 Xshell 软件连接云服务平台，首先安装 git，一般而言，现在的服务器已经内置了 git 安装包，只需要执行简单的安装命令即可安装，如

```
$ yum install git # centos
$ apt-get install git # ubuntu
```

git 与 mysql 不一样，在安装 mysql 时需要安装 mysql-server，即 mysql 服务器。git 是分布式的，每一个安装了 git 的电脑，既是客户端，也是服务器，git 与 git 之间可以相互通信。所谓的 git 服务器，实际上与自己的电脑没有本质上的差别，只是为了更有效地管理项目，都采取中心化的管理方式，因此创建一个 git 服务器作为其他所有人提交代码的最终终端。

2. 创建 git 用户及权限

云服务平台不建议直接使用 root 进行通信交互，需要创建一个 git 用户作为今后提交代码的用户：

```
$ adduser git
```

执行该条命令后，在/home 目录下多了一个 git 目录。按道理来说，系统中多了这个 git 用户，并且家目录在/home/git 中，出于安全考虑，创建的 git 用户不允许登录 shell。这是因为 ssh 连接并登录到服务器上后，用户便可以对服务器进行各种操作了，不安全而且也不合适。用户只需要对仓库能够进行操作就可以了，不需要给予更大的权限。这可以通过编辑/etc/passwd 文件来完成：

```
$ vi /etc/passwd
```

找到类似于 git:x:1001:1001:,,,:/home/git:/bin/bash 的行，末尾的/bin/bash 就是允许登录 shell 的权限，把它改为/user/bin/git-shell，结果为

```
git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell
```

这样，git 用户只可以通过 ssh 使用 git，无法登录 shell，因为 git 用户指定的 git-shell 在每次登录后就会自动退出。

另外还需要给 git 分配一个密码，执行

```
$ passwd git 123456(读者自己设定的密码)
```

这个密码在后面提交代码的时候使用。

3. 创建证书登录

这是 git 比较特殊的一步操作，通信的时候，客户端与服务器需要一个证书进行验证，操作方法很简单，首先在自己的电脑上生成自己的一个公钥：

```
$ cd ~
$ ssh-keygen -t rsa
```

这时，电脑就会在 .ssh 目录下生成一个公钥，. 开头的文件夹都是隐藏的，但是可以 cd 进去。

```
$ cd . ssh
$ vi id_rsa. pub
```

这样就能看到公钥了，把所有的内容都复制下来后，再回到服务器上面操作：

```
$ cd /home/git/
$ mkdir . ssh
$ cd . ssh
$ vi authorized_keys
```

如果是裸机，则服务器上的/home/git 目录下应该没有 .ssh 目录，需要自己创建，打开（自动创建）authorized_keys 后，把刚才复制下来的公钥粘贴进去，保存后退出。使用证书主要是为了不需要密码就可以提交代码。

4. 初始化一个 git 仓库

在/var 下面（此目录可根据读者习惯选择）创建一个 git 目录：

```
$ cd /var
$ mkdir git
$ chown -R git:git git
$chmod 777 git
$ cd git
```

使用 git 命令初始化一个仓库：

```
$ git init --bare yourproject. git
```

git 就会创建一个裸仓库。裸仓库没有工作区，因为服务器上的 git 仓库纯粹是为了共享，所以不让用户直接登录到服务器中去修改工作区。这里有一个细节，就是 .git 目录必须

要有可读/写权限，因为当在 push 的时候是使用 git 用户推送到服务器上的，所以会有一个写入的过程，如果不赋予可写权限，则 push 就会失败。

5. 本地电脑克隆

回到本地电脑上，通过克隆来验证仓库是否可以使用：

```
$ git clone git@10.0.0.11:/var/git/yourproject.git(此处的 10.0.0.11 可以更换为读者服务器的地址)
```

电脑会提示输入 git 的密码，输入密码后，提示克隆了一个空白的版本库。这说明在云服务平台上已经成功搭建了 git 服务器，此时可以将本地开发的代码直接 git push 到云服务平台上。

第 8 章
CC3200 微控制器连接到云服务器

8.1 CC3200 微控制器的程序开发

CC3200 微控制器的程序开发主要涉及将设备联网后，再进行数据的上传和下载、传感器程序的移植及关键数据的显示。CC3200 芯片嵌入式软件概况如图 8.1 所示。

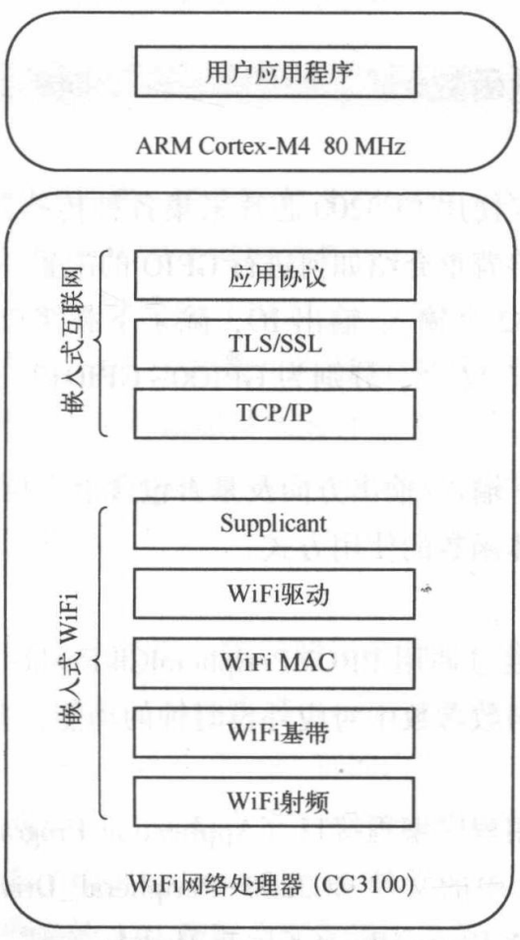


图 8.1 CC3200 芯片嵌入式软件概况

设计 CC3200 微控制器的软件可以参考 TI 公司推出的软件开发工具包 SDK。其包含用于 CC3200 可编程 MCU 的驱动程序、40 个以上的实例应用及使用该解决方案所需要的文档。

CC3200 微控制器程序流程如图 8.2 所示。首先进行开发板的一系列初始化，配置系统所需要传感器的相关引脚后，创建至少两个任务：数据交互任务和显示任务，然后启动任务调度。

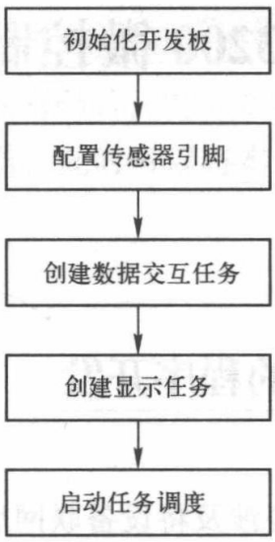


图 8.2　CC3200 微控制器程序流程

8.1.1　GPIO 配置函数

在实际应用中，因为需要使用 CC3200 芯片采集各种传感器的数据，所以 GPIO 的正确配置显得尤为重要。本小节将着重介绍如何进行 GPIO 的配置。CC3200 共有 4 组 IO（Port0、Port1、Port2、Port3），共计 32 个输入/输出 IO，除了下载接口、SOP 模式选择及天线这些固定的引脚，可用的也就只有 27 个，分别为 GPIO0～GPIO17、GPIO22～GPIO25、GPIO28～GPIO32。

设置引脚包括时钟配置、输入/输出方向及是否包含上下拉的问题，所有的 IO 都需要涉及这几个问题。下面来看具体函数的使用方式。

（1）使能外设时钟

此过程是非常重要的，通过调用 PRCMPeripheralClkEnable 函数来完成使能外设时钟的功能。该函数能够实现使能函数参数中对应外设时钟的功能，在默认情况下，外设处于未使能状态。

这里需要调用相应的应用程序编程接口（Application Programming Interface，API），打开之前安装的 cc3200-sdk\docs 中的文件 CC3200-Peripheral_Driver_Library_User's_Guide.cm，在“Module”下单击“PRCM_Power_Reset_Clock_Module_api”后，可以看到 PRCM（Power ResetClockModule）电源管理模块的 API 函数。PRCM 电源管理模块的 API 函数界面如图 8.3 所示。

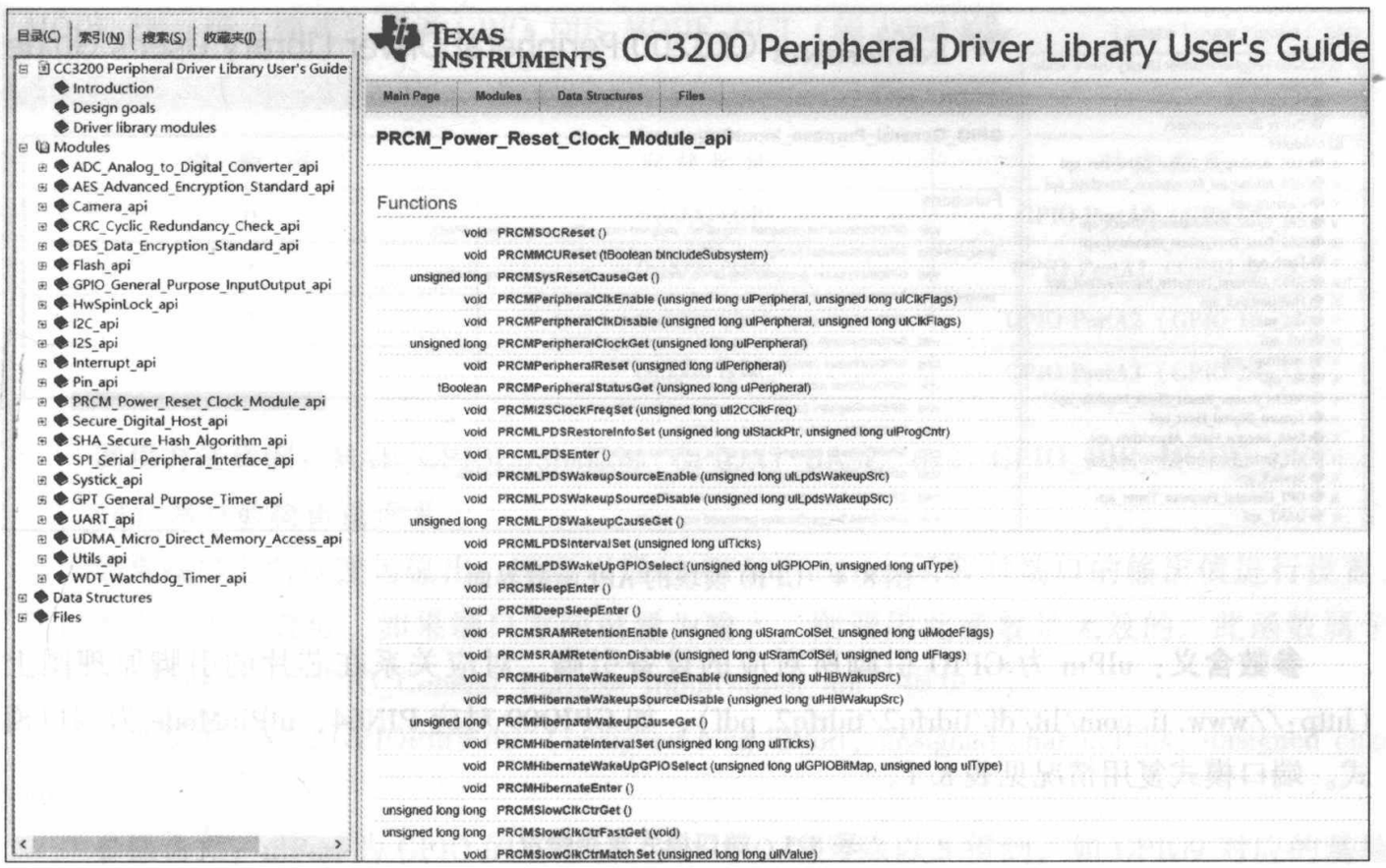


图 8.3　PRCM 电源管理模块的 API 函数界面

函数原型： void PRCMPeripheralClkEnable（unsigned long ulPeripheral，unsigned long ulClkFlags）。

参数含义： ulPeripheral 为外设名称，如 PRCM_GPIOA0（通用输入/输出外设）、PRCM_UARTA0（串口外设）、PRCM_TIMERA0（定时器外设）等。ulClkFlags 为时钟标志位，可以为 PRCM_RUN_MODE_CLK（运行模式）、PRCM_SLP_MODE_CLK（睡眠模式）、PRCM_DSLP_MODE_CLK（深睡眠模式）。这 3 个参数可以进行逻辑或操作后作为函数的第 2 个参数，（深）睡眠模式在睡眠情况下仍然可以保持该时钟是工作的。

调用方法举例： MAP_PRCMPeripheralClkEnable(PRCM_GPIOA0，PRCM_RUN_MODE_CLK)，使能 GPIOA0 时钟为运行模式。

（2）端口的模式设置

通过调用 PinTypeGPIO 函数可设置某一端口的工作模式，配置为通用输入/输出模式、I^2C 时钟模式等。不同的端口能够配置的工作模式也不同，具体需要查阅芯片手册（http://www.ti.com/lit/ug/swru367c/swru367c.pdf）中 483 页的内容。

打开之前安装的 cc3200-sdk \ docs 中的文件 CC3200-Peripheral_Driver_Library_User's_Guide.cm，在“Module”下单击“GPIO_General_Purpose_InputOutput_api”后，就可以看到 GPIO 模块的 API 函数。GPIO 模块的 API 函数界面如图 8.4 所示。

函数原型： void PinTypeGPIO（unsigned long ulPin，unsigned long ulPinMode，tBoolean bOpenDrain）。

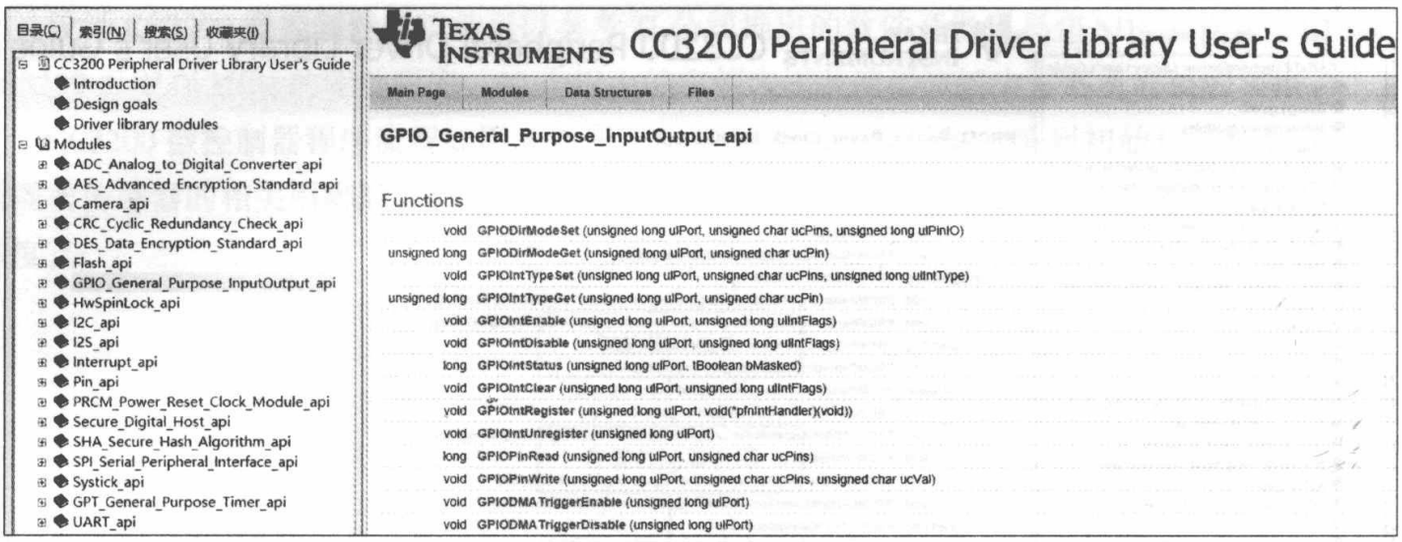


图 8.4　GPIO 模块的 API 函数界面

参数含义：ulPin 为 GPIO 引脚所对应的设备引脚，对应关系在芯片的引脚原理图上（http://www.ti.com/lit/df/tidrfq2/tidrfq2.pdf），如 GPIO09 对应 PIN64，ulPinMode 为端口模式。端口模式复用情况见表 8.1。

表 8.1　端口模式复用情况

编号	引脚	用途	是否作为唤醒源	是否配置为模拟复用	是否与 JTAG 复用	数字引脚复用配置寄存器	数字引脚复用配置模式值	引脚名称	含义
1	GPIO10	I/O	否	否	否	GPIO_PAD_CONFIG_10（0x4402 E0C8）	0	GPIO10	通用输入/输出
							1	I2C_SCL	I^2C 时钟
							3	GT_PWM06	脉冲宽度调制
							7	UART1_TX	串口发送数据
							6	SDCARD_CLK	SD 卡时钟
							12	GT_CCP01	时钟捕获端口

调用方法举例：MAP_PinTypeGPIO(PIN_64, PIN_MODE_0, false)。

(3) 端口的方向设置

通过调用 GPIODirModeSet 函数设置某一引脚为输入/输出方向，此函数属于图 8.4 中所示的“GPIO_General_Purpose_InputOutput_api”模块。

函数原型：void GPIODirModeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulPinIO)。

参数含义：ulPort 为 GPIO 的基地址，由 GPIO 编号除以 8 得到，如 GPIO9 对应的基地址为 9/8=1，即为 GPIOA1_BASE。GPIO 基地址的对应关系见表 8.2。ucPins 为 PAx 口对应的位值，如 GPIO08 对应 PA1 口的第 0 BIT 位（000b），GPIO9 对应 PA1 口的第 1 BIT 位（010b），GPIO10 对应 PA1 口的第 2 BIT 位（100b），依此类推。这个地方完全可以使用已在 gpio.h 中定义的 GPIO_PIN_0、GPIO_PIN_1 等代替。ulPinIO 参数可以为 GPIO_DIR_

MODE_IN（输入模式）或者 GPIO_DIR_MODE_OUT（输出模式）。

表 8.2 GPIO 基地址的对应关系

中 断 号	向 量 地 址	对 应 关 系
0	0x0000 0040	GPIO PortA0（GPIO 0-7）
1	0x0000 0044	GPIO PortA1（GPIO 8-15）
2	0x0000 0048	GPIO PortA2（GPIO 16-23）
3	0x0000 004C	GPIO PortA3（GPIO 24-31）

调用方法举例：MAP_GPIODirModeSet(GPIOA1_BASE, 0x2, GPIO_DIR_MODE_OUT)。

(4) 端口的输出值设置

如果端口方向设置为输出，则可以通过 GPIOPinWrite 函数对端口的输出值进行设置，可以置高也可以置低。如果端口方向设置为输入，则调用此函数是无效的。此函数属于图 8.4 中所示的“GPIO_General_Purpose_InputOutput_api”模块。

函数原型：void GPIOPinWrite（unsigned long ulPort, unsigned char ucPins, unsigned char ucVal)。

参数含义：ulPort 为 GPIO 的基地址，由 GPIO 编号除以 8 得到，如 GPIO9 对应的基地址为 9/8=1，即为 GPIOA1_BASE。GPIO 基地址的对应关系见表 8.2。ucPins 为 PAx 口对应的位值，如 GPIO08 对应 PA1 口的第 0 BIT 位（000b），GPIO9 对应 PA1 口的第 1 BIT 位（010b），GPIO10 对应 PA1 口的第 2 BIT 位（100b），依此类推。这个地方完全可以使用已在 gpio.h 中定义的 GPIO_PIN_0、GPIO_PIN_1 等代替。ucVal 为设置的引脚输出值，置高为 1，置低为 0。

调用方法举例：置高 GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_2,1)；

置低 GPIOPinWrite(GPIOA1_BASE,GPIO_PIN_2,0)。

以上函数都来源于 TI 公司的库函数。函数的具体实现可以在\cc3200-sdk\driverlib 目录下查看。

8.1.2 CC3200 创建多任务

1. 创建任务函数

创建任务函数的原型为

```
OsiReturnVal_e osi_TaskCreate(P_OSI_TASK_ENTRY pEntry,
                              const signed char * const pcName,
                              unsigned short usStackDepth,
                              void * pvParameters,
```

```
unsigned long uxPriority,
OsiTaskHandle * pTaskHandle);
```

创建任务函数参数的中、英文含义见表 8.3。

表 8.3 创建任务函数参数的中、英文含义

参数名称	英文含义	中文含义
pEntry	pointer to the Task Function	任务函数名
pcName	Task Name String	任务名称
usStackDepth	Stack Size Stack Size in 32-bit long words	任务栈深度
pvParameters	pointer to structure to be passed to the Task Function	传给该函数的指针
uxPriority	Task Priority	任务优先级

2. 数据交互任务和显示任务

数据的上传和下载是通过数据交互任务完成的。该任务的目的是将由 CC3200 LaunchPad 硬件平台采集的传感器状态信息上传至云服务器，并且获取云服务器上最新的控制指令，然后根据控制指令改变家中的电器状态。创建此任务的代码为

```
lRetVal = osi_TaskCreate(TaskName, (const signed char *)"Update Data", OSI_STACK_SIZE,
NULL, 1, NULL);
```

LCD 显示任务的目的是在液晶显示屏上显示传感器的状态、家用电器的状态及系统时间。创建此任务的代码为

```
lRetVal = osi_TaskCreate(TaskName, (const signed char *)"Display",OSI_STACK_SIZE, NULL, 2,
NULL);
```

8.1.3 传感器程序的移植

传感器程序的移植过程有 3 个关键点，下面以 DS1302 芯片程序的移植为例进行介绍。

(1) 传感器的功能引脚需要接到 CC3200 芯片的引脚上，然后配置为对应模式：选用 CC3200 的 PIN_50(GPIO00)作为 SCLK 引脚，PIN_59(GPIO04)作为 I/O 引脚，PIN_60(GPIO05)作为 RST 引脚，允许相应的 GPIO 时钟，配置这三个引脚为 GPIO 模式并且统一定义为输出模式。

(2) 传感器需要精准延时来确保时序正确：51 单片机驱动程序延时函数_nop_()的执行时间为 1μs，因为 CC3200 活动模式的主频为 80MHz，而 UtilsDelay(1)的运行时间为三个时钟周期，所以假设延时的数值为 x，则计算公式为

$$x\times 3\times\frac{1}{80\times10^{6}}=10^{-6} \tag{8-1}$$

计算可得 $x\approx27$。也就是说，UtilsDelay(27)的执行时间接近 1μs，可以使用 UtilsDelay(27)替换延时函数_nop_()；

(3) 根据需要配置引脚的输入/输出模式：当需要获取时间时，需要对 DS1302 芯片进行读操作，首先配置引脚为输入模式，若引脚为高电平，则通过函数 MAP_GPIOPinRead(GPIOA0_BASE,0x10)读取的值为引脚的位权，即 PIN_59(GPIO04)引脚读取的值为 16(0x10)。此函数的原型为 long MAP_GPIOPinRead(unsigned long ulPort, unsigned char ucPins)。其中，ulPort 为 GPIO 的基地址；ucPins 为 PAx 口对应的位值，如 GPIO04 对应 PA0 口的第 4 BIT 位（10000b/0x10)。此时需要对读取的数据进行移位处理，以便之后进行处理，从而显示正确的时间。详情请参考链接：https://biog csdn. net/gsh _hello _world/articie/details/51345614。

8.2 CC3200 与云服务器之间的数据交换

8.2.1 CC3200 连接到路由器

在前面的章节中提到，CC3200 可设置为站点 STA、无线接入点 AP 及 WiFi 直接连接三种模式。下面介绍三种模式的具体含义及 SSID、BSSID 和 RSSI 的含义。

AP 即为 Access Point。无线接入点是一个无线网络的接入点，俗称“热点”，是使用无线设备（智能手机等移动设备及笔记本电脑等无线设备）的用户进入有线网络的接入点，主要用于宽带家庭、大楼内部、工厂等地点，典型距离覆盖几十米至上百米，也可以用于远距离传送，将 CC3200 作为一个无线路由器。该路由器的特点是不能插入网线，没有接入 Internet，只能等待其他设备的链接，并且只能接入一个设备。

STA 即 Station，任何一个接入无线 AP 的设备都可以被称为一个站点，也就是平时接入路由器的设备类似于无线终端。STA 本身并不接受无线接入，可以连接到 AP，一般无线网卡工作在该模式下，通过设置 CC3200 在 STA 模式下，能够使其连接至无线接入点 AP（如无线路由器），从而能够使设备上网。

WiFi 直接连接即 WiFi Direct，是针对两个 WiFi 设备之间的通信，CC3200 如果使用 WiFi Direct 功能，就不能再连接其他的路由器。

服务集标识（Service Set Identifier, SSID)：每一个无线 AP 都应该有一个标识用于用户的识别，SSID 就是这个用于用户识别的名字，也就是经常说到的 WiFi 名。

BSSID：每一个网络设备都有用于识别的物理地址，被称为 MAC 地址，在一般情况下，

出厂时都会有一个默认值，可以更改，有固定的命名格式，是识别设备的标识符。BSSID 是针对设备来说的，对于 STA 的设备来说，拿到 AP 接入点的 MAC 地址就是这个 BSSID。

接收的信号强度指示（Received Signal Strength Indication，RSSI）：这个理解起来更简单，就是通过 STA 扫描到 AP 站点的信号强度。

CC3200 通过路由器连接到云服务器的示意图如图 8.5 所示。CC3200 接入一个能连入网络的无线路由器中，进而能够连接到云服务器上进行数据交换。

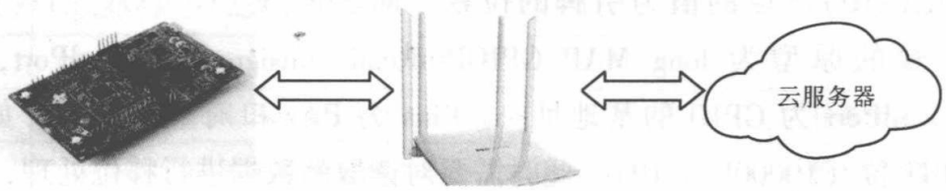


图 8.5　CC3200 通过路由器连接到云服务器的示意图

为了使 CC3200 接入路由器，首先需要配置 CC3200 为 Station 模式，需要在头文件中进行的参数配置见表 8.4；然后参考例程 getting_started_with_wlan_station 即可将 CC3200 连接到路由器上。

表 8.4　在 Station 模式下进行的参数配置

参 数 名	参 数 含 义
SSID_NAME	要连接无线 AP 的名称
SECURITY_TYPE	要连接无线 AP 的安全类型（WPA/WPA2 和 Open）
SECURITY_KEY	要连接无线 AP 的密码
SSID_LEN_MAX	无线 AP 名称的最大长度
BSSID_LEN_MAX	MAC 地址的最大长度

其中，如果使用的安全类型为开放，则将宏 SECURITY_TYPE 定义为 SL_SEC_TYPE_OPEN；如果使用的类型为 WPA 或 WPA2，则将宏 SECURITY_TYPE 定义为 SL_SEC_TYPE_WPA。

8.2.2　CC3200 与云服务器之间的数据交换

CC3200 连接路由器后，可以通过 HTTP GET 方式进行传感器数据的上传，云服务器通过 HTTP 响应将控制指令下发到 CC3200。首先，CC3200 通过 HTTP GET 请求中的 URL 上传传感器数据，HTTP GET 请求到达阿里云服务器后分发到相应的函数中进行处理，从请求的 URL 中取出上传的传感器数据值后存入云端数据库，从数据库中获取最新的电器控制指令，通过 HTTP 协议响应给 CC3200 微控制器，从而开发板可以更新被控电器的状态。CC3200 与云服务器之间的数据交换流程如图 8.6 所示。

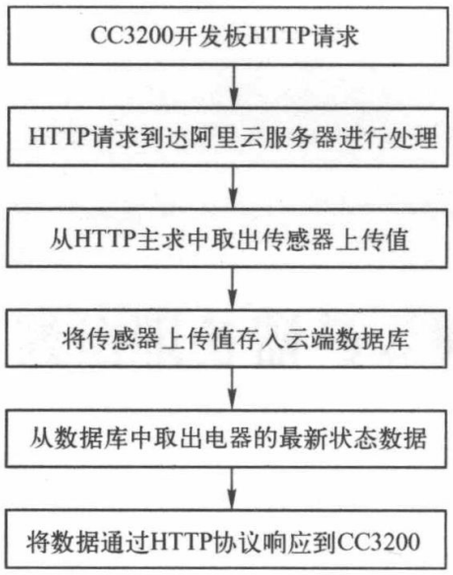


图 8.6　CC3200 与云服务器之间的数据交换流程

第9章 微信服务器与云服务器之间的交互

9.1 微信公众平台接入云服务器

开发者将云服务器接入微信公众平台进行开发时需要进行以下配置：

① 开启微信公众平台开发者模式；

② 在微信公众平台上填写服务器配置；

③ 验证服务器地址的有效性；

④ 在云服务器上实现业务逻辑。

9.1.1 开启开发者模式

登录微信公众平台后，单击“开发”→“基本配置”，如果微信公众平台显示“你还没有成为开发者”，就单击“成为开发者”，如图 9.1 所示。

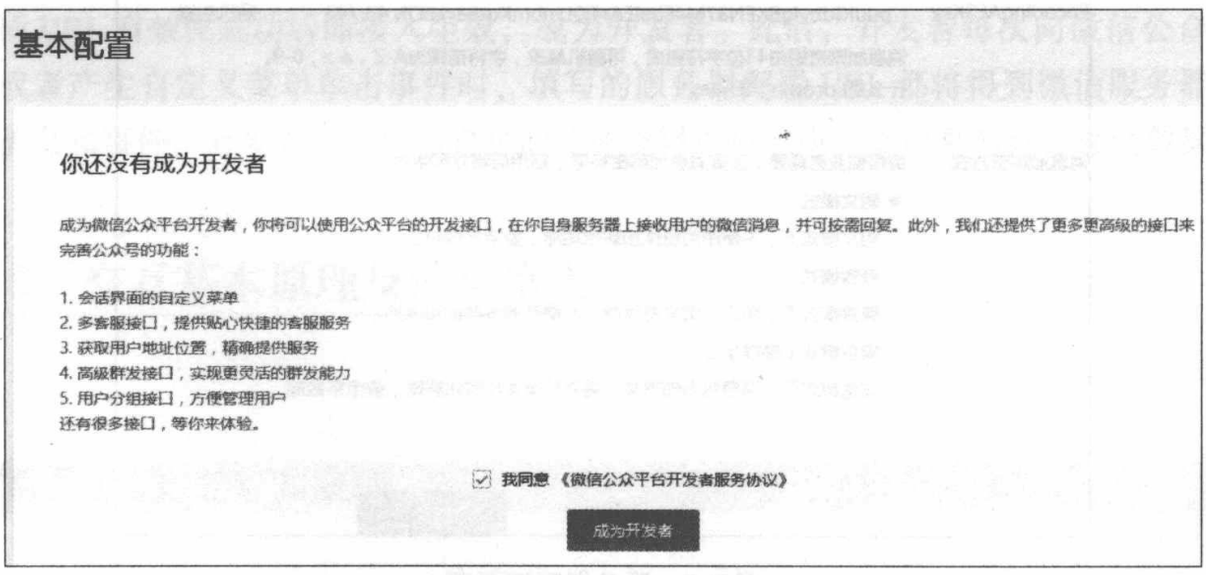


图 9.1 开启开发者模式

9.1.2 填写服务器配置

成为开发者之后，在“基本配置”中的“服务器配置”中单击“修改配置”按钮，填写服务器地址 URL、令牌 Token 及密钥 EncodingAESKey。其中，URL 是开发者用来接收微信消息和事件的接口地址，若开发者的服务器域名是 139.129.9.166，准备用/WeChat/wechat/index.do 来接收消息，就填写 http://139.129.9.166/WeChat/wechat/index.do；Token 是微信服务器和开发者服务器进行通信时验证身份用的，可以随便填写，但要注意保密，用作生成签名。该 Token 会与接口 URL 中包含的 Token 进行比对，从而验证安全性；EncodingAESKey 由开发者手动填写或随机生成，将用作消息体的加解密密钥。

同时，开发者可选择消息加解密方式：明文模式、兼容模式和安全模式。模式的选择与服务器的配置在提交后都会立即生效，开发者需谨慎填写及选择。本书采用的消息加解密方式为明文模式，服务器配置界面如图 9.2 所示。

图 9.2 服务器配置界面

9.1.3 验证服务器地址的有效性

开发者提交信息后，微信服务器将发送 GET 请求到填写的服务器地址 URL，GET 请求携带四个参数，见表 9.1。

表 9.1 GET 请求携带的四个参数

参　数	含　义
signature	微信加密签名，signature 结合开发者填写 Token 参数和请求中的 timestamp 参数、nonce 参数
timestamp	时间戳
nonce	随机数
echostr	随机字符串

开发者服务器通过检验 signature 对请求进行校验。校验方式如下：

① 将 Token、timestamp、nonce 三个参数按英文字母顺序排序；

② 将三个参数的字符串拼接成一个字符串进行 sha1 加密；

③ 开发者获得加密后的字符串可与 signature 对比，判断该请求是否来源于微信。

如果计算出来的加密字符串和由微信传过来的 signature 相等，则说明这个请求确实是微信后台发过来的；如果是别人伪造的请求，则因为不知道 Token，所以无法计算出正确的 signature；若确认此次的 GET 请求来自微信服务器，就把微信后台传过来的 echostr 原封不动地传回去，这样就可以通过验证，云服务器接入微信公众平台生效，成为开发者成功，否则接入失败。

9.1.4 在云服务器上实现业务逻辑

验证 URL 有效性成功后即接入生效，成为开发者。此后，开发者每次向微信公众号发送消息或者产生自定义菜单单击事件时，填写的服务器配置 URL 都将得到微信服务器推送过来的消息和事件，开发者可以依据自身的业务逻辑进行响应，如回复消息、存储数据等。

9.2 交互基本原理及消息格式

9.2.1 交互基本原理

微信服务器相当于一个转发服务器。微信客户端发起请求到微信服务器后，微信服务器

负责把请求转发到云服务器。开发者在云服务器上自定义服务具体实现对转发的 HTTP 请求做出响应。需要注意的是，请求和响应都是以 XML 形式进行的。云服务器处理完毕后，响应给微信服务器，微信服务器转发响应给微信客户端。微信服务器转发消息框架如图 9.3 所示。

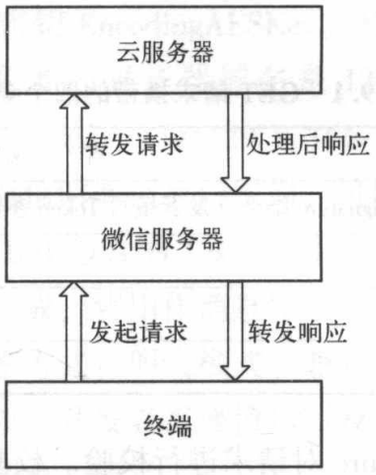


图 9.3　微信服务器转发消息框架

9.2.2　微信客户端推送消息

当微信客户端向微信公众账号发送消息时，微信服务器向云服务器发送的消息是一个 POST 请求，与普通的 POST 请求不同的是，URL 会带上 signature、timestamp、nonce 这 3 个参数，如

```
POST http://139.129.9.166/WeChat/wechat/index.do? signature=xxx&timestamp=123456&nonce=123
```

POST 请求的 BODY 是一个不规范的 XML。下面将详细介绍微信客户端推送消息的 XML 数据包格式。

(1) 请求的文本消息 XML 数据包为

```
<xml>
 <ToUserName><![CDATA[toUser]]></ToUserName>
 <FromUserName><![CDATA[fromUser]]></FromUserName>
 <CreateTime>1348831860</CreateTime>
 <MsgType><![CDATA[text]]></MsgType>
 <Content><![CDATA[this is a test]]></Content>
 <MsgId>1234567890123456</MsgId>
 </xml>
```

请求的文本消息 XML 数据包参数及含义见表 9.2。

表 9.2 请求的文本消息 XML 数据包参数及含义

参　数	含　义
ToUserName	开发者的微信号
FromUserName	发送方账号（一个 OpenID）
CreateTime	消息创建时间戳（整型）
MsgType	text
Content	文本消息内容
MsgId	消息 id，64 位整型

（2）请求的图片消息 XML 数据包为

```
<xml>
 <ToUserName><![CDATA[toUser]]></ToUserName>
 <FromUserName><![CDATA[fromUser]]></FromUserName>
 <CreateTime>1348831860</CreateTime>
 <MsgType><![CDATA[image]]></MsgType>
 <PicUrl><![CDATA[this is a url]]></PicUrl>
 <MediaId><![CDATA[media_id]]></MediaId>
 <MsgId>1234567890123456</MsgId>
 </xml>
```

请求的图片消息 XML 数据包参数及含义见表 9.3。

表 9.3 请求的图片消息 XML 数据包参数及含义

参　数	含　义
ToUserName	开发者的微信号
FromUserName	发送方账号（一个 OpenID）
CreateTime	消息创建时间戳（整型）
MsgType	image
PicUrl	图片链接（由系统生成）
MediaId	图片消息媒体 id，可以调用多媒体文件下载接口拉取数据
MsgId	消息 id，64 位整型

（3）请求的语音消息 XML 数据包为

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>1357290913</CreateTime>
<MsgType><![CDATA[voice]]></MsgType>
```

```
<MediaId><![CDATA[media_id]]></MediaId>
<Format><![CDATA[Format]]></Format>
<MsgId>1234567890123456</MsgId>
</xml>
```

请求的语音消息 XML 数据包参数及含义见表 9.4。

表 9.4　请求的语音消息 XML 数据包参数及含义

参　数	含　义
ToUserName	开发者的微信号
FromUserName	发送方账号（一个 OpenID）
CreateTime	消息创建时间戳（整型）
MsgType	voice
MediaId	语音消息媒体 id，可以调用多媒体文件下载接口拉取数据
Format	语音格式，如 amr、speex 等
MsgID	消息 id，64 位整型

注意，开通语音识别后，开发者每次发送语音给微信公众号时，微信会在推送的语音消息 XML 数据包中增加一个 Recongnition 字段（注：由于客户端缓存，因此开发者开启或者关闭语音识别功能，对新关注者立刻生效，对已关注者需要 24 小时生效。开发者可以重新关注此账号进行测试）。

开启语音识别后的语音 XML 数据包为

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>1357290913</CreateTime>
<MsgType><![CDATA[voice]]></MsgType>
<MediaId><![CDATA[media_id]]></MediaId>
<Format><![CDATA[Format]]></Format>
<Recognition><![CDATA[腾讯微信团队]]></Recognition>
<MsgId>1234567890123456</MsgId>
</xml>
```

在多出的字段中，Format 为语音格式，一般为 amr；Recognition 为语音识别结果，使用 UTF8 编码。

9.2.3　云服务器响应消息

对于每一个 POST 请求，开发者都需要进行响应。响应是以 XML 数据包的形式进行的，

可对文本、图片、图文、语音、视频、音乐等消息进行响应。微信服务器在将用户的消息发送给开发者服务器后，微信后台要求必须在5秒内回复。如果微信服务器在5秒内收不到响应，就会断掉连接，重新发起请求，总共重试三次。假如微信服务器无法保证在5秒内处理并回复，则必须做出下述回复，这样微信服务器才不会对此进行任何处理，并且不会发起重试，否则，将出现严重的错误提示。回复方式有两种：

① 推荐方式为直接回复success。

② 直接回复空串。空串是指字节长度为0的空字符串，而不是XML结构体中content字段的内容为空。

一旦遇到以下情况，微信服务器就会在微信公众号中向用户下发系统提示，“该公众号暂时无法提供服务，请稍后再试”：

① 开发者在5秒内未回复任何内容；

② 开发者回复了异常数据，如JSON数据等。

下面将详细介绍云服务器响应消息的XML数据包格式。

（1）响应的文本消息XML数据包格式为

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>12345678</CreateTime>
<MsgType><![CDATA[text]]></MsgType>
<Content><![CDATA[你好]]></Content>
</xml>
```

响应的语音消息XML数据包参数及含义见表9.5。

表9.5 响应的语音消息XML数据包参数及含义

参　数	是否必须	含　义
ToUserName	是	接收方账号（收到的OpenID）
FromUserName	是	开发者的微信号
CreateTime	是	消息创建时间（整型）
MsgType	是	text
Content	是	回复的消息内容（换行：在content中能够换行，微信客户端就支持换行显示）

（2）响应的图片消息XML数据包格式为

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>12345678</CreateTime>
```

```
<MsgType><![CDATA[image]]></MsgType>
<Image>
<MediaId><![CDATA[media_id]]></MediaId>
</Image>
</xml>
```

响应的图片消息 XML 数据包参数及含义见表 9.6。

表 9.6　响应的图片消息 XML 数据包参数及含义

参　数	是否必须	含　义
ToUserName	是	接收方账号（收到的 OpenID）
FromUserName	是	开发者的微信号
CreateTime	是	消息创建时间戳（整型）
MsgType	是	image
MediaId	是	通过素材管理接口上传多媒体文件得到的 id

（3）响应的语音消息 XML 数据包格式为

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>12345678</CreateTime>
<MsgType><![CDATA[voice]]></MsgType>
<Voice>
<MediaId><![CDATA[media_id]]></MediaId>
</Voice>
</xml>
```

响应的语音消息 XML 数据包参数及含义见表 9.7。

表 9.7　响应的语音消息 XML 数据包参数及含义

参　数	是否必须	含　义
ToUserName	是	接收方账号（收到的 OpenID）
FromUserName	是	开发者的微信号
CreateTime	是	消息创建时间戳（整型）
MsgType	是	voice
MediaId	是	通过素材管理接口上传多媒体文件得到的 id

9.3　云服务器上的微信请求接口设计

由于微信后台的限制，微信所有的消息只能转发到同一个接口。如果将所有的业务逻辑

都集中在一个接口方法中，会导致该方法急速膨胀并最终不可维护，因此云服务器可以根据微信发送过来的 MsgType 及消息内容进行反射调用，将不同的消息类型 MsgType 及不同的消息内容通过转换为关键字 Key 的方式路由到不同的方法中。这样，在以后添加新的功能（如操作窗帘等）或者响应新的消息类型（如图片消息等）时，只需要添加对应的方法及 Key 即可，不用修改原有的逻辑。云服务器微信请求接口设计流程如图 9.4 所示。

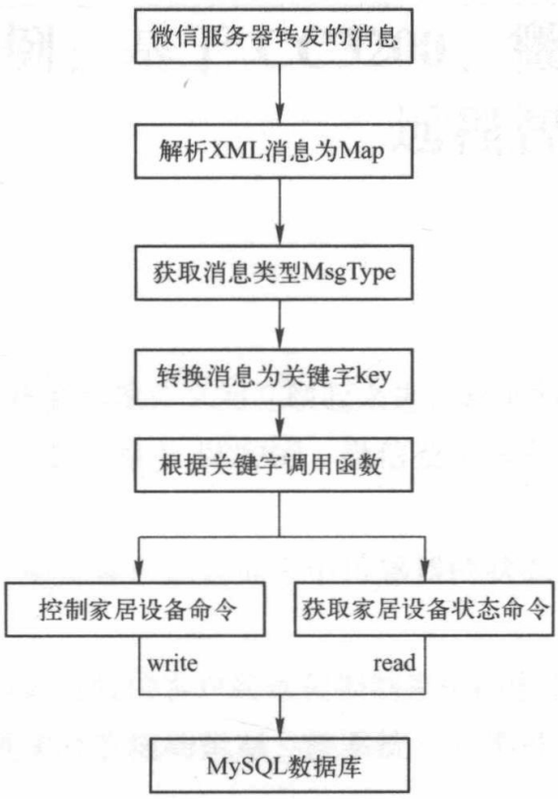


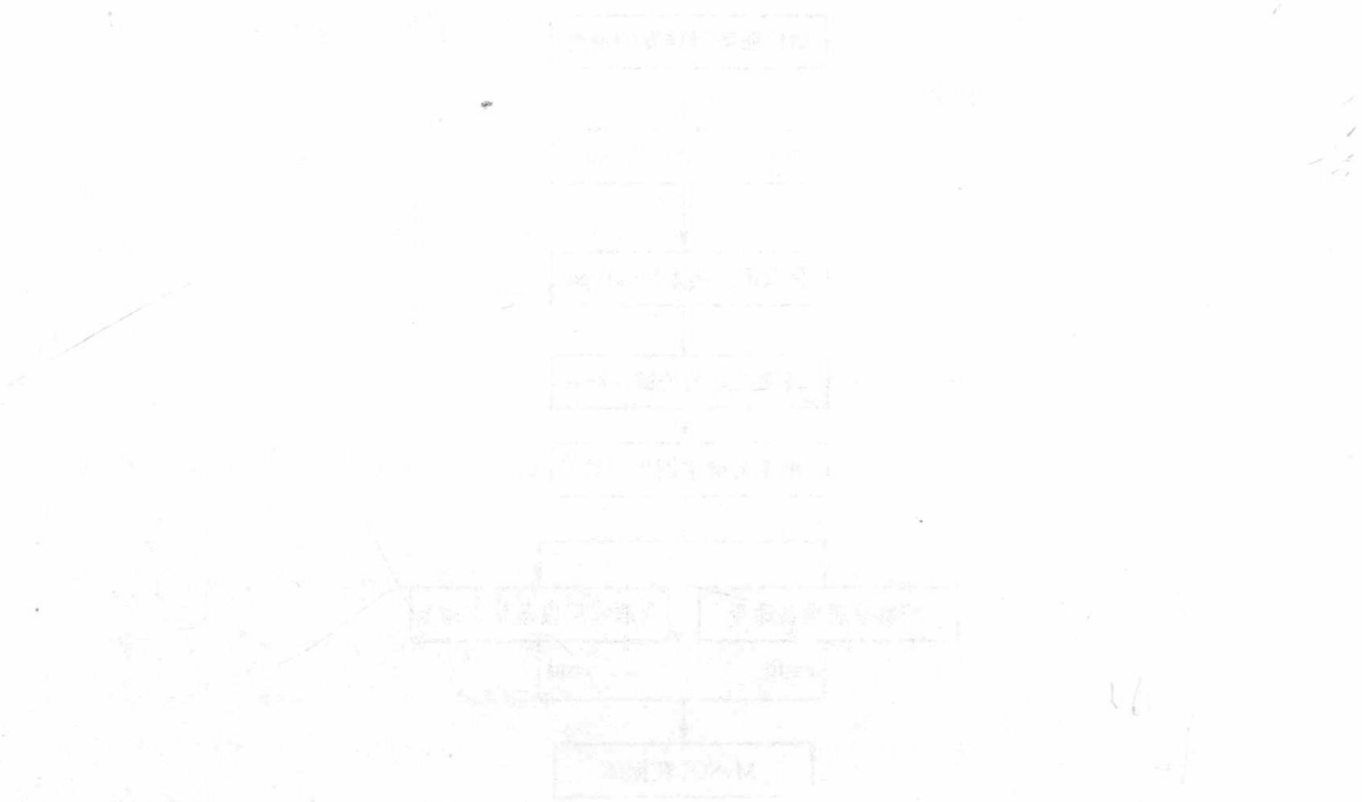
图 9.4　云服务器微信请求接口设计流程

云服务器微信请求接口设计如下：

① 请求到达阿里云服务器，经处理函数将 XML 消息解析为 Map，然后获取消息类型 MsgType。

② 根据获取到的消息类型 MsgType 分别路由到相关的函数进行处理，将消息转换为自定义的关键字 key。

③ 根据关键字调用相应的操作函数：如果操作为控制家居设备的命令时，则操作函数首先把命令保存到数据库，然后将响应的文本等信息打包成 XML 格式回应给微信服务器；如果操作为获取家居设备状态的命令时，则操作函数首先从数据库中读取状态信息，然后打包回应给微信服务器。



第 10 章

应用案例：基于 CC3200、微信及云服务的远程智能云家居系统

智能家居系统不仅能够将家居设备之间互联起来进行数据的交换和通信，而且还可以接入互联网进行数据的上传和下载。基于 CC3200、微信及云服务的远程智能云家居系统具有以下功能：

① 查询：用户可以通过微信客户端查询家中传感器的状态，如家中卧室的温度、烟雾传感器的状态等；

② 远程控制：用户可以通过微信客户端远程控制家中的电器，如空调、热水器等；

③ 预警监测：能够提供实时监测的机制，将系统异常事件及时反馈给用户，如出现异常气体后通知用户。

10.1 系统设计方案

远程智能云家居系统框图如图 10.1 所示。用户首先需要将智能手机接入网络并且关注智能云家居公众号，然后通过密钥绑定智能云家居系统，就可以在智能云家居公众号中发送

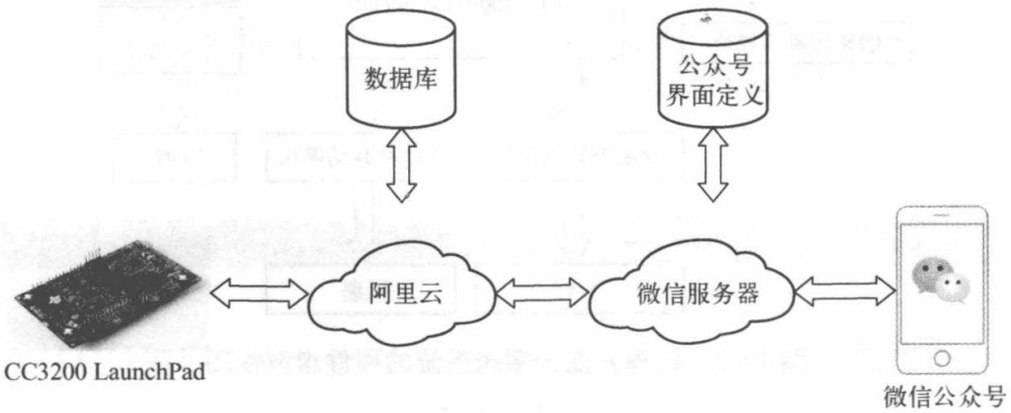


图 10.1 远程智能云家居系统框图

文本、语音消息，或者单击智能云家居公众号中的菜单实现以下功能：远程控制家居设备，如台灯、风扇、空调等；查询设备状态，如获取台灯的状态、风扇的转速、空调的温度等；获取传感器数据，如室内温/湿度等。

远程智能云家居系统分为客户端和服务器端。客户端分为微信公众号和 CC3200 LaunchPad 开发板。这两个客户端都可以向服务器提交数据并下载数据。服务器端分为微信服务器端和阿里云服务器端。微信服务器端主要实现智能云家居公众号菜单的定义，并且把微信客户端发送的消息以可扩展标记语言（eXtensible Markup Language，XML）形式的数据发送到阿里云服务器端。阿里云服务器端主要实现接收微信服务器端转发的操作指令与 CC3200 LaunchPad 开发板上传的温/湿度等数据，经过一定的组织和处理后存入数据库：一方面，当微信客户端请求数据时，可将响应消息打包成 XML 数据通过微信服务器端发送到微信客户端；另一方面，当 CC3200 LaunchPad 开发板客户端有连接请求时，可接收开发板上传的传感器数据，并将数据存入数据库，同时将请求的数据通过 HTTP 协议发送给客户端。

10.2 系统硬件设计

远程智能云家居系统的硬件由 CC3200 LaunchPad 开发板和外围模块组成，如图 10.2 所示。外围模块由温/湿度传感器模块、烟雾传感器模块、继电器模块、电动机驱动模块、DS1302 实时时钟模块及 12864 液晶显示模块等组成。CC3200 LaunchPad 开发板通过 WiFi 连接到路由器，进而接入互联网，然后通过 HTTP GET 的方式获取台灯或空调等家用电器的最新状态，同时将温/湿度等传感器的数值通过 WiFi 上传到阿里云服务器。

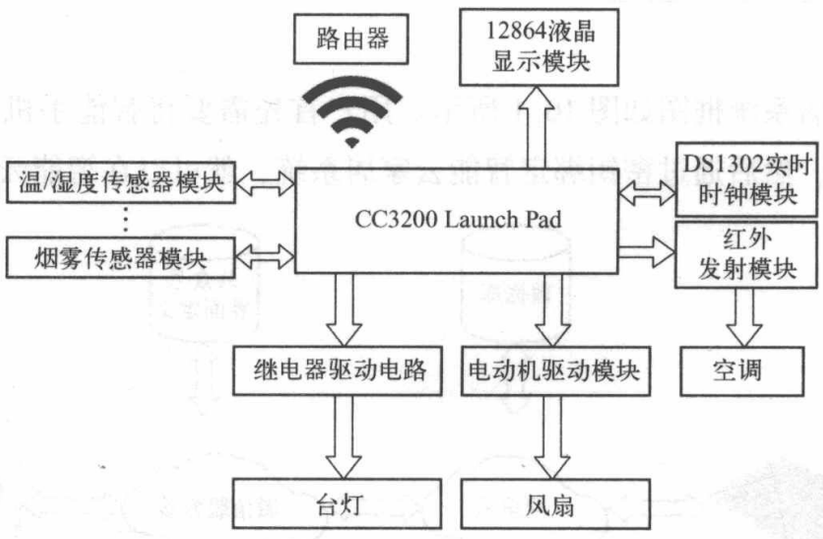


图 10.2 远程智能云家居系统的硬件组成框图

10.2.1 温/湿度传感器模块

温/湿度传感器模块采用 DHT11 型数字温/湿度传感器。传感器包括一个电阻式感湿元件和一个 NTC 测温元件。DATA 引脚需要上拉后与微处理器的 I/O 端口连接，连接线的长度短于 20m 时，用 5.1kΩ 的上拉电阻，供电电压范围为 3.3~5.5V；使用 3.3V 电压供电时，连接线的长度不能太长（要小于 100cm），否则线路压降会导致传感器的供电不足，造成测量偏差。另外，每一次读出的温/湿度数值是上一次的测量结果，欲获取实时数据，需要连续读取两次，每一次读取传感器的时间间隔超过 5s 即可获得准确的数据。

DHT11 型数字温/湿度传感器采用简化的单总线方式串行传输数据，一次传送 40 位数据，高位先出，数据格式为 8bit 湿度整数数据+8bit 湿度小数数据+8bit 温度整数数据+8bit 温度小数数据+8bit 校验位。其中，温/湿度小数数据为 0。单总线即只有一根数据线，系统中的数据交换、控制均由单总线完成。设备（主机或从机）通过一个漏极开路或三态端口连接单总线，以允许设备在不发送数据时能够释放单总线，而让其他设备使用单总线。由于采用主从结构，只有主机呼叫从机时，从机才能应答，因此主机访问器件都必须严格遵循单总线序列。如果出现序列混乱，则器件将不响应主机。DHT11 型温/湿度传感器模块的电路结构如图 10.3 所示。

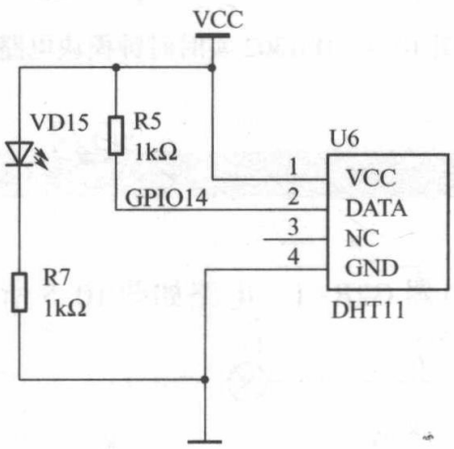


图 10.3 DHT11 型温/湿度传感器模块的电路结构

10.2.2 DS1302 实时时钟模块

DS1302 实时时钟模块是美国 DALLAS 公司推出的一种高性能、低功耗、具有涓细电流充电能力、带 RAM 的实时时钟电路，引脚功能见表 10.1。

表 10.1　DS1302 的引脚功能

引脚名称	引脚功能	引脚名称	引脚功能
X1、X2	30.768kHz 晶振引脚	SCLK	串行时钟
VCC1、VCC2	电源供电引脚	RST	复位引脚
I/O	数据输入/输出引脚	GND	地

DS1302 实时时钟模块可以对年、月、日、周、时、分、秒计时；年计数可达 2100 年；在闰年时还可以补偿天数；可选择 12 小时制和 24 小时制；可以设置 AM、PM；能够正常工作的电压范围为 2.5~5.5V；与 CPU 进行通信时可以采用同步模式，该模式采用的接口为三线形式，当需要传送多个字节的数据时，可采用突发方式进行一次传送。DS1302 实时时钟模块包括时钟/日历寄存器和 31 字节（8 位）的数据暂存寄存器，仅通过一条串行输入/输出口进行数据通信。DS1302 实时时钟模块在只有一个主电源引脚的基础上又增加了一个后背电源引脚，在正常工作的同时能够对后背电源进行充电。在系统断电情况下，DS1302 实时时钟模块依旧可以正常工作，实时更新系统时间。DS1302 实时时钟模块电路如图 10.4 所示。

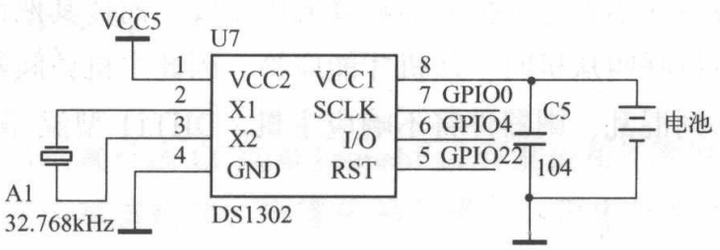


图 10.4　DS1302 实时时钟模块电路

10.2.3　继电器模块

继电器模块采用欧姆龙继电器 G2R-1，电路如图 10.5 所示。继电器的输入连接 CC3200

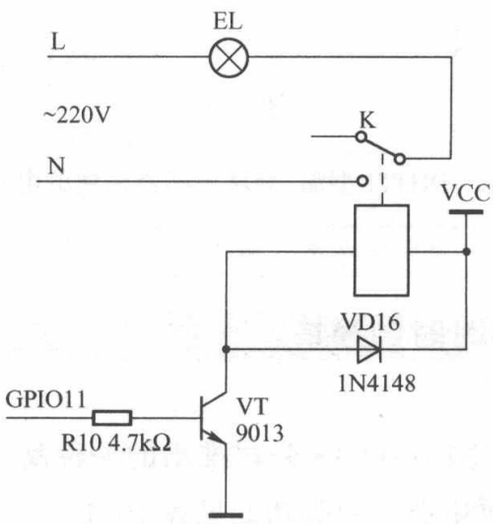


图 10.5　继电器模块电路

芯片的 PIN_02 引脚，控制对象为台灯，台灯的工作电压为交流 220V，当台灯的状态由开切换到关时，线圈会突然断电，于是在线圈的两端会产生极大的反向电动势，极易造成系统的损坏。为了解决这个问题，继电器模块在继电器的线圈两端增加了一个二极管，可以消除线圈突然断电时产生的反向电动势。

10.2.4 电动机驱动模块

电动机驱动模块电路如图 10.6 所示，采用 L298N 双 H 桥直流电动机驱动芯片，驱动部分端子的供电电压为+5~+35V，如果需要板内取电，则供电电压为+7~+35V，驱动部分的峰值电流为 2A；逻辑部分端子的供电电压为+5~+7V，工作电流范围为 0~36mA。

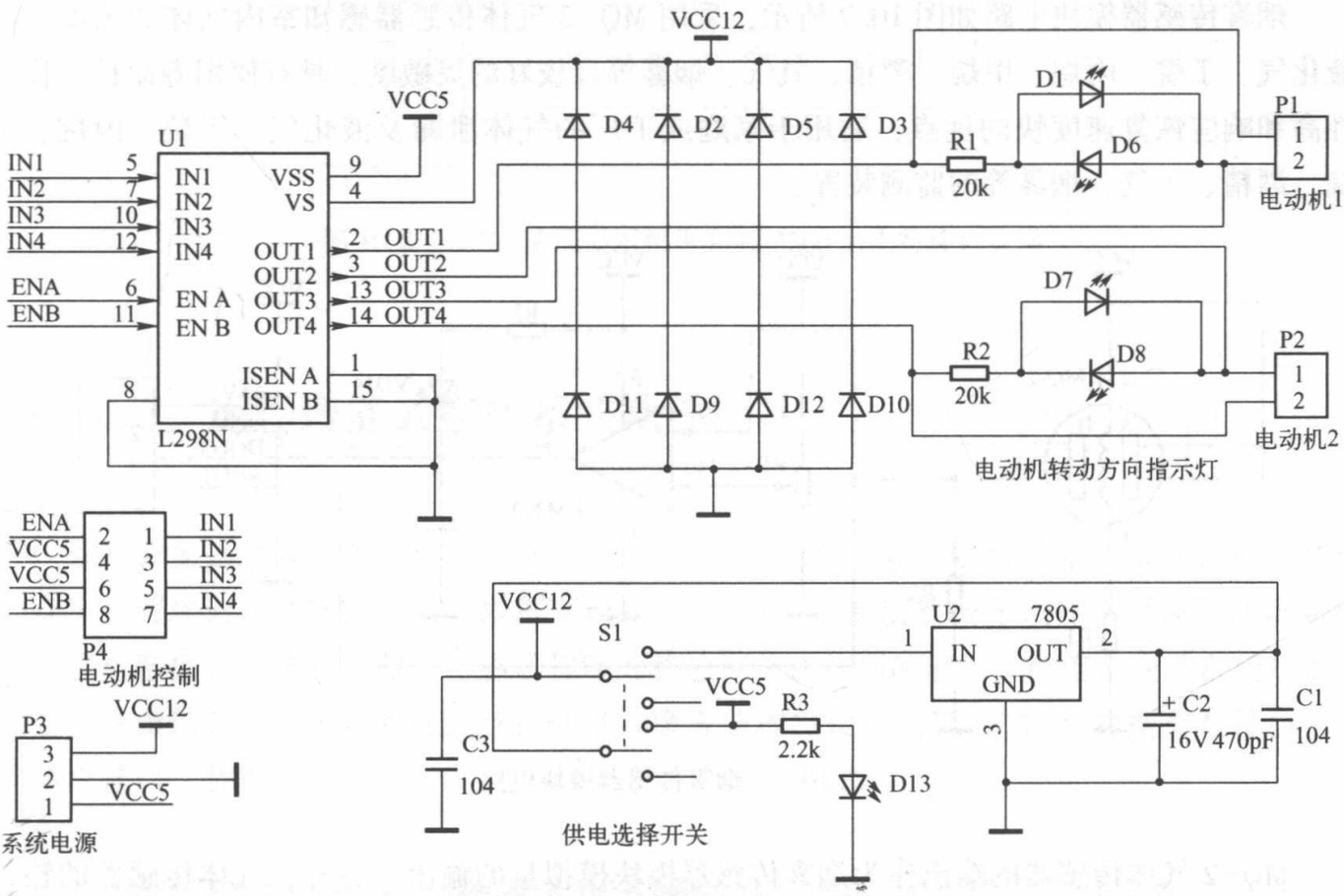


图 10.6 电动机驱动模块电路

电动机驱动模块可以驱动两路直流电动机，使能端 ENA 和 ENB 为高电平时有效，电平的控制方式和直流电动机的状态见表 10.2。如果想要通过 PWM 对直流电动机调速，则需要设置两个输入端 IN1 和 IN2 确定电动机的转动方向，然后输出 PWM 脉冲到使能端即可实现电动机转速的调整。当使能信号输入为 0 时，电动机处于自由停止状态；当使能信号输入为 1 时，并且 IN1 和 IN2 为 00 或者 11 时，电动机处于制动状态，此时可阻止电动机转动。

表 10.2　电平的控制方式和直流电动机的状态

ENA	IN1	IN2	直流电动机的状态
0	×	×	停止
1	0	0	制动
1	0	1	正转
1	1	0	反转
1	1	1	制定

10.2.5　烟雾传感器模块

烟雾传感器模块电路如图 10.7 所示，采用 MQ-2 气体传感器感知室内气体的浓度，对液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等有较好的灵敏度，具有使用寿命长、稳定性高和响应恢复速度快的优点，适用于家庭或工厂的气体泄漏及液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等的监测装置。

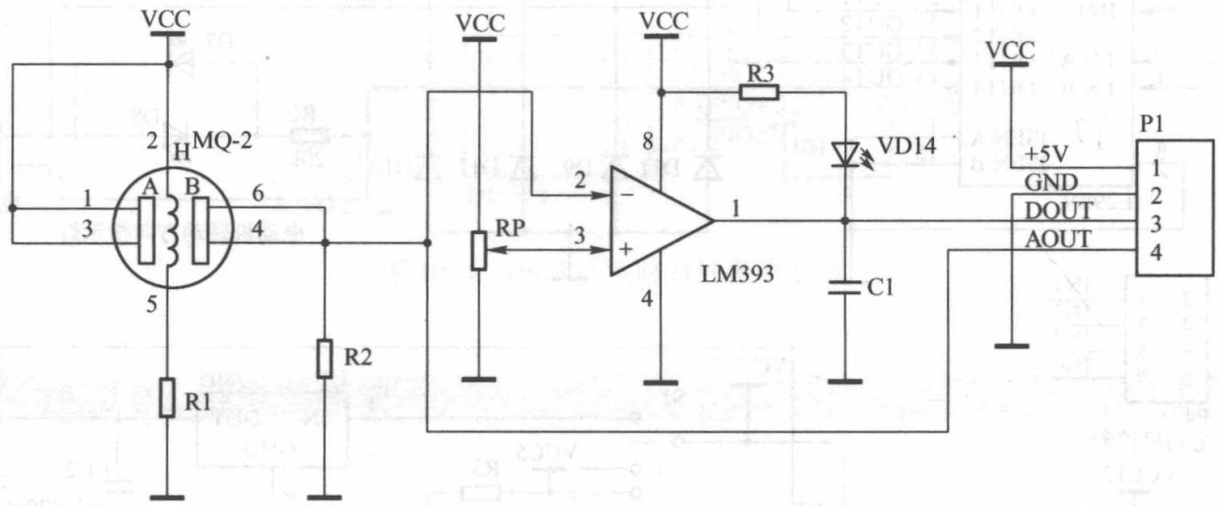


图 10.7　烟雾传感器模块电路

MQ-2 气体传感器的输出作为烟雾传感器模块模拟量的输出。另外，气体传感器的输出接到电压比较器 LM393 芯片的 2 号引脚，最后经 LM393 芯片的 1 号引脚输出 TTL 电平。所以，该模块既能实现模拟量输出 0~5V 电压，浓度越高，电压越高；又能实现 TTL 电平输出，有毒气体浓度高时，输出低电平。

10.2.6　12864 液晶显示模块

12864 液晶显示模块是深圳市晶联讯电子公司生产的，采用 3.3V 和 5V 两种供电方案。因为远程智能云家居系统中的 CC3200 芯片采用 5V 供电，所以 12864 液晶模块也采用 5V 供

电。12864 液晶显示模块可以显示 128×64 点阵单色图片，或显示 8 个/行×4 行 16×16 点阵的汉字，或显示 16 个/行×8 行 8×8 点阵的英文、数字、符号。12864 液晶显示模块和 CC3200 的串行接口电路如图 10.8 所示。

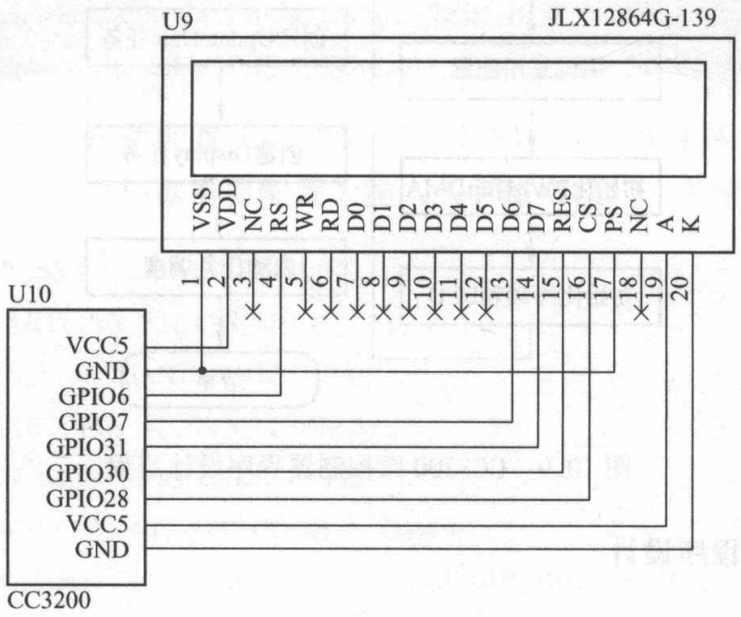


图 10.8　12864 液晶显示模块和 CC3200 的串行接口电路

10.3　远程智能云家居系统软件设计

远程智能云家居系统的软件设计主要包括三个部分：CC3200 微控制器程序设计、微信公众平台程序设计及阿里云服务器程序设计。CC3200 微控制器程序设计主要涉及将设备联网并进行数据的上传和下载、传感器程序的移植和关键数据的显示；微信公众平台程序设计主要包含微信公众号菜单界面设计；阿里云服务器程序设计包括 CC3200 数据交互接口、微信请求接口、数据库的设计。

10.3.1　CC3200 微控制器程序设计

CC3200 微控制器程序设计流程如图 10.9 所示。首先进行开发板的初始化，进行引脚复用配置，并且对 PWM 和 μDMA 进行初始化；然后初始化终端、LCD、I^2C 和显示标题；最后创建 UpdateData 任务和 Display 任务，并启动任务调度。其中，实时操作系统使用的是 TI-RTOS。该系统是 TI 公司在 2012 年推出的面向 MCU 平台、基于抢占式多线程内核的完整实时操作系统，可以显著加速软件的开发。开发者不需要编写和维护调度工具、协议栈及低级驱动器等复杂的系统软件程序。

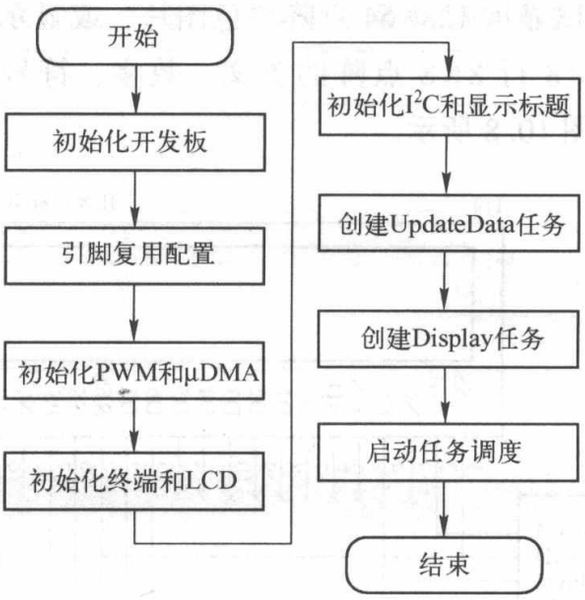


图 10.9　CC3200 微控制器程序设计流程

1. 初始化开发板程序设计

初始化开发板程序设计流程如图 10.10 所示。其中包括设置中断向量表函数 MAP_IntVTableBaseSet( )、允许处理器中断函数 MAP_IntMasterEnable( )、允许中断函数 MAP_IntEnable( )及初始化函数 PRCMCC3200MCUInit( )。

图 10.10　初始化开发板程序设计流程

2. 引脚复用配置程序设计

首先使用函数 MAP_PRCMPeripheralClkEnable( )允许 GPIO 时钟，包含 GPIOA0、GPIOA1、UARTA0、IICA0、TIMERA3 时钟。关键代码为

```
MAP_PRCMPeripheralClkEnable(PRCM_GPIOA0, PRCM_RUN_MODE_CLK);
MAP_PRCMPeripheralClkEnable(PRCM_GPIOA1, PRCM_RUN_MODE_CLK);
MAP_PRCMPeripheralClkEnable(PRCM_UARTA0, PRCM_RUN_MODE_CLK);
MAP_PRCMPeripheralClkEnable(PRCM_IICA0, PRCM_RUN_MODE_CLK);
MAP_PRCMPeripheralClkEnable(PRCM_TIMERA3, PRCM_RUN_MODE_CLK);
```

然后配置 GPIO 引脚，包括配置 TX、RX、GT_PWM、I^2C 及连接传感器的 GPIO 引脚；最后配置 GPIO 的方向，可以根据需要配置为输入或者输出模式。关键代码为

```
// Configure PIN_55 for UART0 UART0_TX
MAP_PinTypeUART(PIN_55, PIN_MODE_3);
// Configure PIN_57 for UART0 UART0_RX
MAP_PinTypeUART(PIN_57, PIN_MODE_3);
// Configure PIN_64 for GPIOOutput——LED
MAP_PinTypeGPIO(PIN_64, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x2, GPIO_DIR_MODE_OUT);
// Configure PIN_01 for TIMERPWM6 GT_PWM06
MAP_PinTypeTimer(PIN_01, PIN_MODE_3);
// Configure PIN_02 for GPIOOutput——Light
MAP_PinTypeGPIO(PIN_02, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x8, GPIO_DIR_MODE_OUT);
// Configure PIN_05 for GPIOOutput——DHT11
MAP_PinTypeGPIO(PIN_05, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x40, GPIO_DIR_MODE_OUT);
// Configure PIN_06 for GPIOIn——Smoke
MAP_PinTypeGPIO(PIN_06, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x80, GPIO_DIR_MODE_IN);
// Configure PIN_50 for SCLK——DS1302
MAP_PinTypeGPIO(PIN_50, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA0_BASE, 0x1, GPIO_DIR_MODE_OUT);
// Configure PIN_59 for I/O——DS1302
MAP_PinTypeGPIO(PIN_59, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA0_BASE, 0x10, GPIO_DIR_MODE_OUT);
// Configure PIN_15 RST——DS1302
MAP_PinTypeGPIO(PIN_15, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA2_BASE, 0x40, GPIO_DIR_MODE_OUT);
// Configure PIN_07 for beep
MAP_PinTypeGPIO(PIN_07, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA2_BASE, 0x01, GPIO_DIR_MODE_OUT);
```

```
// Configure PIN_03 for IIC0 IIC_SCL
MAP_PinTypeIIC(PIN_03, PIN_MODE_5);
// Configure PIN_04 for IIC0 IIC_SDA
MAP_PinTypeIIC(PIN_04, PIN_MODE_5);
```

引脚复用配置程序流程设计如图 10.11 所示。

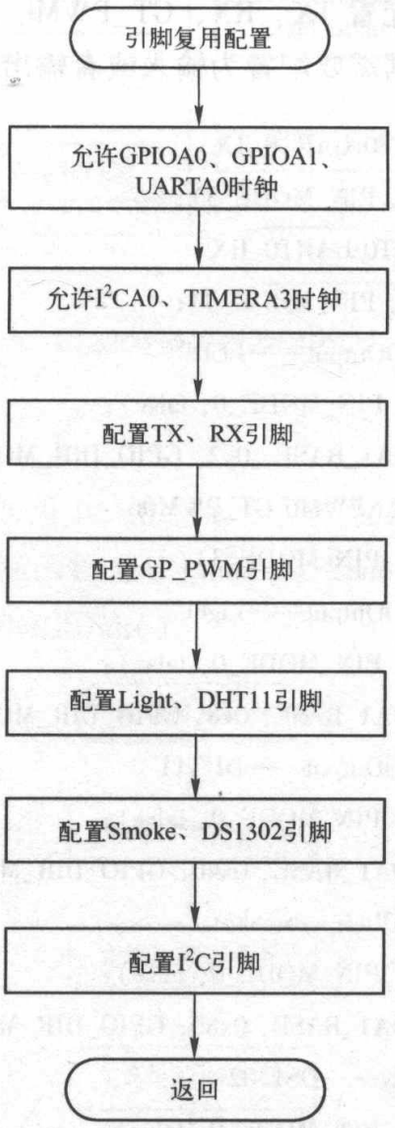


图 10.11　引脚复用配置程序设计流程

3. 初始化 PWM 程序设计

初始化 PWM 程序设计流程如图 10.12 所示。首先需要允许定时器时钟 MAP_PRCMPeripheralClkEnable()；然后设置 PWM 模式 SetupTimerPWM Mode()；最后使能定时器 MAP_TimerEnable()。关键代码为

```
void InitPWMModules()
{
    // Initialization of timers to generate PWM output
```

```
    MAP_PRCMPeripheralClkEnable(PRCM_TIMERA3, PRCM_RUN_MODE_CLK);
    // TIMERA3 (TIMER B) as YELLOW of RGB light.  GPIO 10 --> PWM_6
    SetupTimerPWMMode(TIMERA3_BASE, TIMER_A,
    (TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PWM | TIMER_CFG_B_PWM), 1);
    MAP_TimerEnable(TIMERA3_BASE,TIMER_A);
}
```

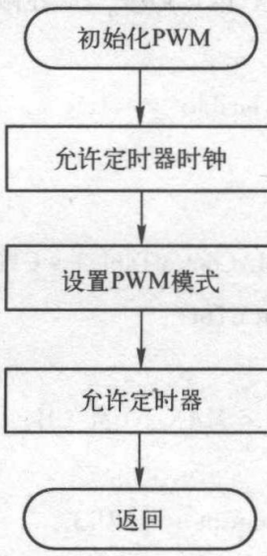


图 10.12　初始化 PWM 的程序设计流程

4. 初始化 μDMA 程序设计

首先通过函数 MAP_ PRCMPeripheralClkEnable( ) 允许 μDMA 时钟；然后利用函数 MAP_ PRCMP eripheralReset( ) 复位 μDMA，使用函数 MAP_uDMAIntRegister( ) 注册 μDMA 中断；最后调用函数 MAP_μDMAEnable( ) 允许 μDMA，函数 MAP_μDMAControlBaseSet( ) 设置通道控制表。关键代码为

```
void MDMAInit( )
{
    unsigned int uiLoopCnt;
    // Enable McASP at the PRCM module
    MAP_PRCMPeripheralClkEnable(PRCM_MDMA,PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralReset(PRCM_MDMA);
    // Register interrupt handlers
#if defined (USE_TIRTOS) || defined (USE_FREERTOS) || defined (SL_PLATFORM_MULTI_
THREADED)
    // USE_TIRTOS: if app uses TI-RTOS (either networking/non-networking)
    // USE_FREERTOS: if app uses Free-RTOS (either networking/non-networking)
    // SL_PLATFORM_MULTI_THREADED: if app uses any OS + networking(simplelink)
```

```
        osi_InterruptRegister(INT_MDMA, DmaSwIntHandler, INT_PRIORITY_LVL_1);
        osi_InterruptRegister(INT_MDMAERR, DmaErrorIntHandler, INT_PRIORITY_LVL_1);
    #else
        MAP_IntPrioritySet(INT_MDMA, INT_PRIORITY_LVL_1);
        MAP_μDMAIntRegister(MDMA_INT_SW, DmaSwIntHandler);
        MAP_IntPrioritySet(INT_MDMAERR, INT_PRIORITY_LVL_1);
        MAP_μDMAIntRegister(MDMA_INT_ERR, DmaErrorIntHandler);
    #endif
        // Enable μDMA using master enable
        MAP_μDMAEnable();
        // Set Control Table
        memset(gpCtlTbl,0,sizeof(tDMAControlTable) * CTL_TBL_SIZE);
        MAP_μDMAControlBaseSet(gpCtlTbl);
        // Reset App Callbacks
        for(uiLoopCnt = 0; uiLoopCnt < MAX_NUM_CH; uiLoopCnt++)
        {
            gfpAppCallbackHndl[uiLoopCnt] = NULL;
        }
    }
```

初始化 μDMA 程序设计流程如图 10.13 所示。

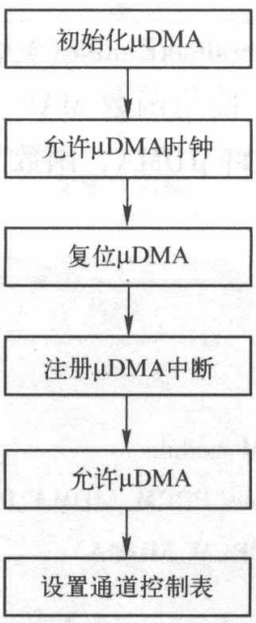


图 10.13　初始化 μDMA 程序设计流程

5. 初始化 12864 液晶显示模块程序设计

首先对 12864 液晶显示模块（LCD）进行初始化，关键代码为

```
void initial_lcd( )
{
        RS_H;SCLK_H;SDA_H;CS1_H;RESET_H;
        RESET_L;                    /*低电平复位*/
        MAP_UtilsDelay(1000);
        RESET_H;                    /*复位完毕*/
        MAP_UtilsDelay(1000);
        transfer_command(0xe2);     /*软复位*/
        MAP_UtilsDelay(500);
        transfer_command(0x2c);     /*升压步聚 1*/
        MAP_UtilsDelay(500);
        transfer_command(0x2e);     /*升压步聚 2*/
        MAP_UtilsDelay(500);
        transfer_command(0x2f);     /*升压步聚 3*/
        MAP_UtilsDelay(500);
        transfer_command(0x24);     /*粗调对比度,可设置范围 0x20~0x27*/
        transfer_command(0x81);     /*微调对比度*/
        transfer_command(0x1A);     /*微调对比度的值,可设置范围 0x00~0x3f*/
        transfer_command(0xa2);     /*1/9 偏压比(bias)*/
        transfer_command(0xc8);     /*行扫描顺序:从上到下*/
        transfer_command(0xa0);     /*列扫描顺序:从左到右*/
        transfer_command(0x42);
        transfer_command(0xaf);     /*开显示*/
}
```

其中，宏定义在 LCD.c 文件中的宏定义如下：

```
#define      RS_H          GPIO_SET(RS_PORT)= 1 << (RS_PORT % 8)
#define      RS_L          GPIO_SET(RS_PORT)= 0 << (RS_PORT % 8)
#define      SCLK_H        GPIO_SET(SCLK_PORT)= 1 << (SCLK_PORT % 8)
#define      SCLK_L        GPIO_SET(SCLK_PORT)= 0 << (SCLK_PORT % 8)
#define      SDA_H         GPIO_SET(SDA_PORT)= 1 << (SDA_PORT % 8)
#define    SDA_L           GPIO_SET(SDA_PORT)= 0 << (SDA_PORT % 8)
#define      CS1_H         GPIO_SET(CS1_PORT)= 1 << (CS1_PORT % 8)
#define    CS1_L           GPIO_SET(CS1_PORT)= 0 << (CS1_PORT % 8)
#define      RESET_H       GPIO_SET(RESET_PORT)= 1 << (RESET_PORT % 8)
#define      RESET_L     GPIO_SET(RESET_PORT)= 0 << (RESET_PORT % 8)
```

初始化 12864 液晶显示模块后，使用函数 MAP_PRCMPeripheralClkEnable( ) 允许 GPIOA3 时钟；然后利用函数 MAP_PinTypeGPIO( ) 配置 GPIO 引脚，包括配置 RS、RES、CS、D6、D7 引脚；最后需要配置 GPIO 的方向，可以根据需要配置为输入或者输出模式。关键代码为

```
void LCD_PinMuxConfig(void)
```

```
{
    /* PIN_61:RS
     * PIN_62:D6
     * PIN_53:RES
     * PIN_18:CS
     * PIN_45:D7
     */
    // Enable Peripheral Clocks
    MAP_PRCMPeripheralClkEnable(PRCM_GPIOA0, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_GPIOA3, PRCM_RUN_MODE_CLK);
    // Configure PIN_61 for GPIO Output ——RS
    MAP_PinTypeGPIO(PIN_61, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, 0x40, GPIO_DIR_MODE_OUT);

    // Configure PIN_62 for GPIO Output ——D6
    MAP_PinTypeGPIO(PIN_62, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA0_BASE, 0x80, GPIO_DIR_MODE_OUT);
    // Configure PIN_53 for GPIO Input ——RES
    MAP_PinTypeGPIO(PIN_53, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA3_BASE, 0x40, GPIO_DIR_MODE_OUT);
    // Configure PIN_18 for GPIO Output ——CS
    MAP_PinTypeGPIO(PIN_18, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA3_BASE, 0x10, GPIO_DIR_MODE_OUT);
    // Configure PIN_45 for GPIO Output ——D7
    MAP_PinTypeGPIO(PIN_45, PIN_MODE_0, false);
    MAP_GPIODirModeSet(GPIOA3_BASE, 0x80, GPIO_DIR_MODE_OUT);
}
```

初始化 LCD 的程序设计流程如图 10.14 所示。

图 10.14　初始化 LCD 的程序设计流程

6. 初始化 DS1302 程序设计

初始化 DS1302 实时时钟芯片时，首先关闭写保护功能，然后向芯片指定地址 WRITE_RTC_ADDR[7] = {0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c}写入初始时间 TIME[7] = {0, 0x46, 0x10, 0x20, 0x05, 0x05, 0x16}，存储顺序是秒、分、时、日、月、周、年，使用 BCD 码存储，即 2016 年 5 月 20 日（周五）10:46。关键代码为

```
/*************************************************************************
 * 函数名          : Ds1302Init
 * 函数功能        : 初始化 DS1302.
 * 输入            : 无
 * 输出            : 无
*************************************************************************/
void Ds1302Init()
{
    uchar n;
    Ds1302Write(0x8E,0X00);          //禁止写保护,就是关闭写保护功能
    for (n=0; n<7; n++)              //写入 7 个字节的时钟信号:分秒时日月周年
    {
        Ds1302Write(WRITE_RTC_ADDR[n],TIME[n]);
    }
    Ds1302Write(0x8E,0x80);          //打开写保护功能
}
```

其中，向 DS1302 写数据函数的关键代码为

```
/*************************************************************************
 * 函数名          : Ds1302Write
 * 函数功能        : 向 DS1302 命令(地址+数据)
 * 输入            : addr,dat
 * 输出            : 无
*************************************************************************/
void Ds1302Write(uchar addr, uchar dat)
{
    uchar n;
//  RST = 0;
    RST_CLR;
//  _nop_();
```

```
    Delay_1us;

//  SCLK = 0;                   //先将 SCLK 置低电平。
    SCLK_CLR;
//  _nop_();
    Delay_1us;
//  RST = 1;                    //然后将 RST(CE)置高电平。
    RST_SET;
//  _nop_();
    Delay_1us;

    for (n=0; n<8; n++)         //开始传送八位地址命令
    {
//      DSIO = addr & 0x01;     //数据从低位开始传送
        if(addr & 0x01)
            IO_SET;
        else
            IO_CLR;
        addr >>= 1;
//      SCLK = 1;               //数据在上升沿时,DS1302 读取数据
        SCLK_SET;
//      _nop_();
        Delay_1us;
//      SCLK = 0;
        SCLK_CLR;
//      _nop_();
        Delay_1us;
    }
    for (n=0; n<8; n++)         //写入 8 位数据
    {
//      DSIO = dat & 0x01;
        if(dat & 0x01)
            IO_SET;
        else
            IO_CLR;
```

```
        dat >>= 1;
//      SCLK = 1;//数据在上升沿时,DS1302 读取数据
        SCLK_SET;
//      _nop_();
        Delay_1us;
//      SCLK = 0;
        SCLK_CLR;
//      _nop_();
        Delay_1us;
    }

//  RST = 0;//传送数据结束
    RST_CLR;
//  _nop_();
    Delay_1us;
}
```

读取 DS1302 数据函数的关键代码为

```
/*****************************************************************************
 * 函数名              : Ds1302Read
 * 函数功能            : 读取一个地址的数据
 * 输入                : addr
 * 输出                : dat
*****************************************************************************/

uchar Ds1302Read(uchar addr)
{
    uchar n,dat,dat1;
//  RST = 0;
    RST_CLR;
//  _nop_();
    Delay_1us;

//  SCLK = 0;                   //先将 SCLK 置低电平。
    SCLK_CLR;
//  _nop_();
    Delay_1us;
```

```
//  RST = 1;                    //然后将 RST(CE)置高电平。
    RST_SET;
//  _nop_();
    Delay_1us;

    for(n=0; n<8; n++)          //开始传送八位地址命令
    {
//      DSIO = addr & 0x01;     //数据从低位开始传送
        if(addr & 0x01)
            IO_SET;
        else
            IO_CLR;
        addr >>= 1;
//      SCLK = 1;               //数据在上升沿时,DS1302 读取数据
        SCLK_SET;
//      _nop_();
        Delay_1us;
//      SCLK = 0;               //DS1302 下降沿时,放置数据
        SCLK_CLR;
//      _nop_();
        Delay_1us;
    }
//  _nop_();
    Delay_1us;
    IO_IN;
    for(n=0; n<8; n++)          //读取 8 位数据
    {
//      dat1 = DSIO;            //从最低位开始接收
        dat1=IO_Read    ;
        if(dat1)
        {
//          UART_PRINT("dat1=%x\n\r",dat1);
            dat1=0x01;
        }
        else
        {
            dat1=0x0;
```

```
        }
        dat = (dat>>1) | (dat1<<7);
//      SCLK = 1;
        SCLK_SET;
//      _nop_();
        Delay_1us;
//      SCLK = 0;                    //DS1302 下降沿时,放置数据
        SCLK_CLR;
//      _nop_();
        Delay_1us;
    }

//  RST = 0;
    RST_CLR;
//  _nop_();                         //以下为 DS1302 复位的稳定时间
    Delay_1us;
//  SCLK = 1;
    SCLK_SET;
//  _nop_();
    Delay_1us;
    IO_OUT;
//  DSIO = 0;
    IO_CLR;
//  _nop_();
    Delay_1us;
//  DSIO = 1;
    IO_SET;
//  _nop_();
    Delay_1us;
//  UART_PRINT("dat=%x\n\r",dat);
    return dat;
}
```

7. UpdateData 任务程序设计

UpdateData 任务程序设计流程如图 10.15 所示。首先将开发板连接到指定的 SSID 路由器上，进而连接到互联网；然后通过 DNS 服务获取域名的 IP 地址，创建 HTTP 客户端，连接到 HTTP 服务器上获取最新的控制指令并且上传家中的状态；最后睡眠 3.5 s 挂起该任务，

以便执行其他任务。

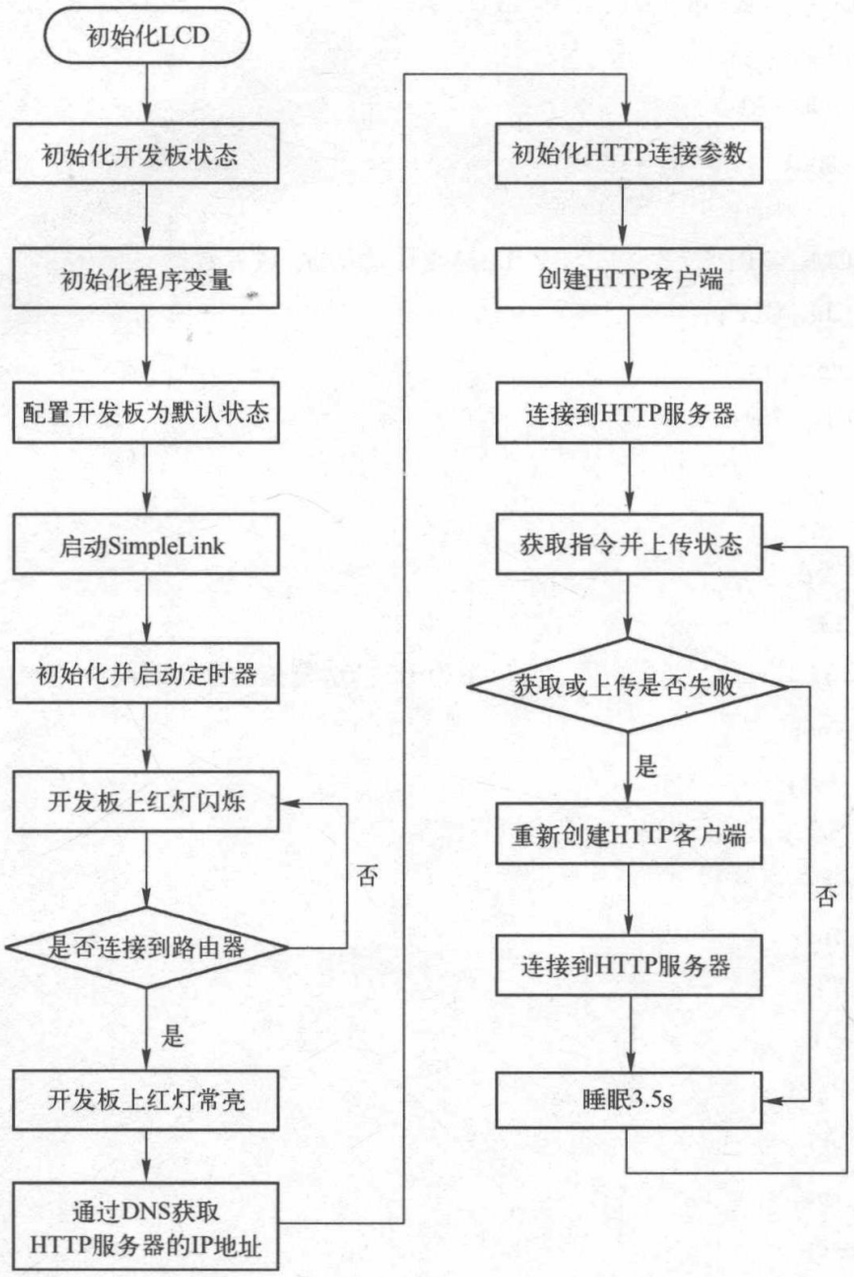


图 10.15　UpdateData 任务程序设计流程

其中，获取最新的控制指令并上传状态的流程如图 10.16 所示。

首先需要设置 HTTP GET 的请求头部，本设计中需要设置请求的主机名为 Host: 139.129.9.166；然后通过 sprintf 函数设置请求的统一资源标识符（Uniform Resource Identifier，URI）：sprintf(acSendBuff,"/WeChat/cc3200/ upload_and_get.do? sequence=number1&temperature=%.2f&humidity=%d",TempValue,humidity)。其中，temperature 和 humidity 的数据通过此 URI 上传。

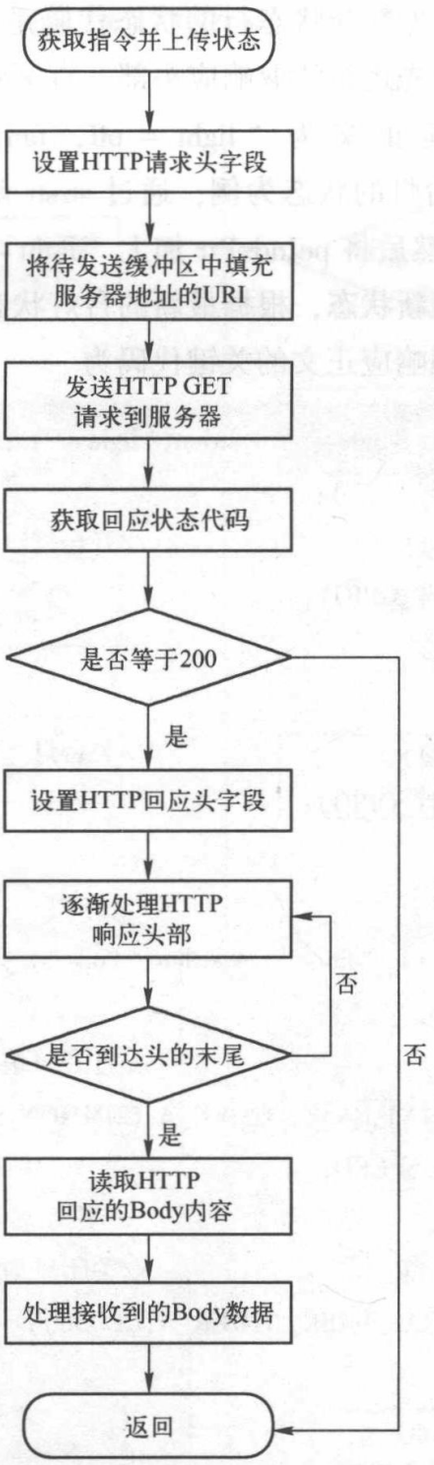


图 10.16　获取最新的控制指令并上传状态的流程

最后，开发板发送 HTTP GET 请求到阿里云服务器，阿里云服务器接收到 HTTP 请求后，解析请求并定位请求资源。阿里云服务器 HTTP 响应报文见表 10.3。

表 10.3　阿里云服务器 HTTP 响应报文

HTTP 响应报文	说　明
HTTP/1.1 200 OK	状态行
Server: Apache-Coyote/1.1 Content-Length: 47	响应头部
Date: Sun, 22 May 2016 09:56:20 GMT	响应正文

开发板收到 HTTP 响应后，先判断状态行的状态代码是否等于 200。如果等于 200，就代表客户端请求成功。此时需要先逐行读取响应头部，直至到达响应头部的末尾；然后开始读取响应正文，返回的响应正文为“light = off, fan = on, airconditionState = cold, airconditionTemp=26”。以获取台灯的状态为例，通过 strstr 函数寻找“light=”字符串在响应正文中的位置（pcIndxPtr），然后将 pcIndxPtr 加上“light=”的长度得到台灯状态字符串的起始地址，进而获取台灯的最新状态，根据最新的台灯状态更新台灯的状态标志，以供显示任务中改变台灯的状态。处理响应正文的关键代码为

```
pcIndxPtr=strstr(acRecvbuff, "light=") + strlen("light=") ;
pcEndPtr = strstr(pcIndxPtr, " ,");
if( strstr(pcIndxPtr,"on") ){                    //开台灯
    GPIO_IF_LedOn(LIGHT_GPIO);
    LightStatus = LightOn;
}
if( strstr(pcIndxPtr,"off") ){                   //关台灯
    GPIO_IF_LedOff(LIGHT_GPIO);
    LightStatus = LightOff;
}
pcIndxPtr = strstr(pcEndPtr+1, "fan=") + strlen("fan=") ;
pcEndPtr = strstr(pcIndxPtr, " ,");
if( strstr(pcIndxPtr,"on") ){                    //打开风扇
MAP_TimerMatchSet(TIMERA3_BASE, TIMER_A,COMMON_SPEED);
    FanStatus = COMMON_SPEED;
}
if( strstr(pcIndxPtr,"off") ){                   //关闭风扇
MAP_TimerMatchSet(TIMERA3_BASE, TIMER_A,CLOSE);
FanStatus = CLOSE;
}
pcIndxPtr = strstr(pcEndPtr+1, "airconditionState=") + strlen("airconditionState=") ;
pcEndPtr = strstr(pcIndxPtr, " ,");
if( strstr(pcIndxPtr,"hot") )                    //空调制热
    airconditionState = HOT;
if( strstr(pcIndxPtr,"cold") )                   //空调制冷
    airconditionState = COLD;
if( strstr(pcIndxPtr,"close") )                  //空调关闭
    airconditionState = CLOSE;
pcIndxPtr = strstr(pcEndPtr+1, "airconditionTemp=") + strlen("airconditionTemp=") ;
IRTransmit(airconditionState, pcIndxPtr);        //控制空调
```

处理响应正文程序设计流程如图 10.17 所示。

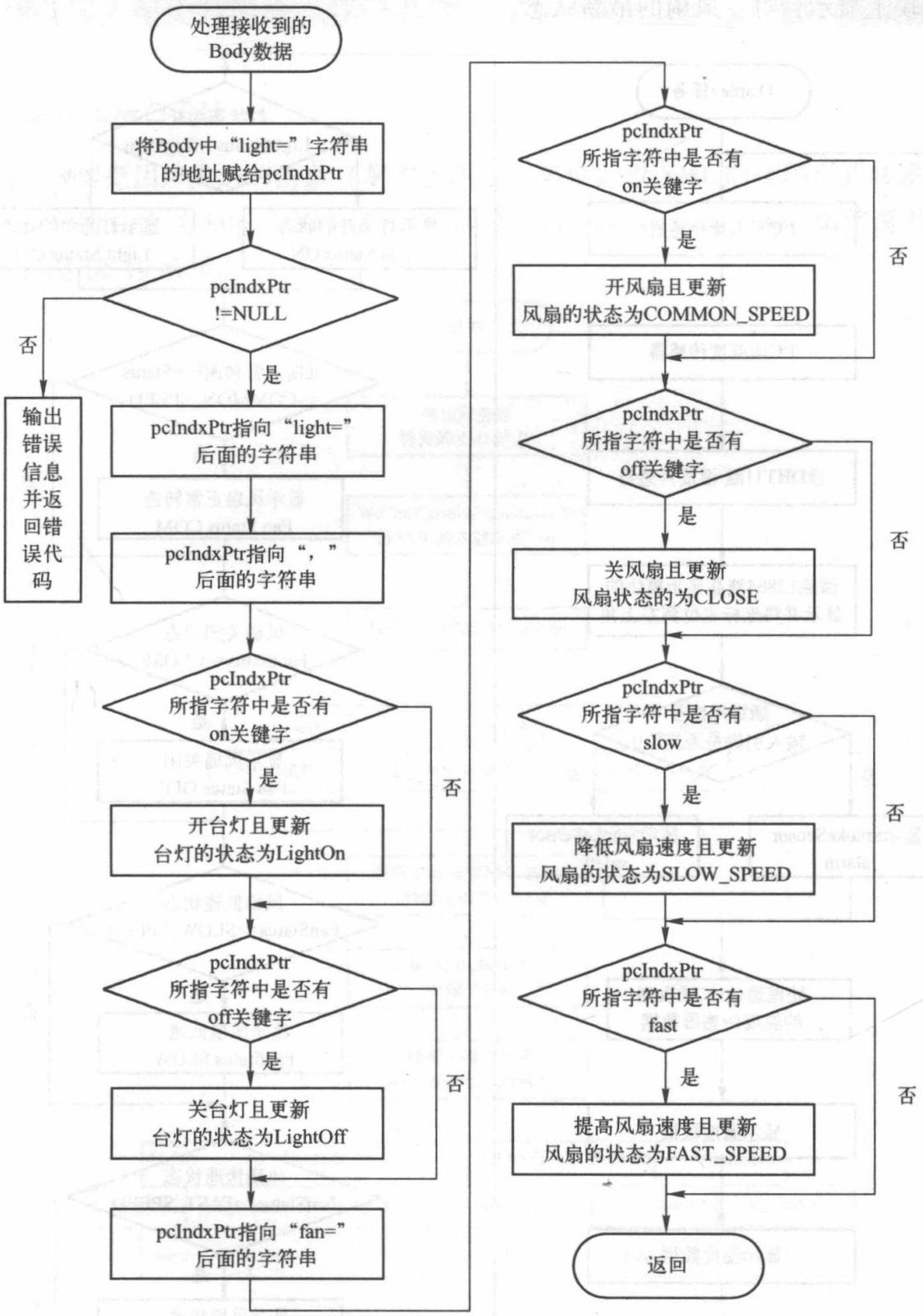


图 10.17　处理响应正文程序设计流程

8. Display 任务程序设计

Display 任务程序设计流程如图 10.18 所示。因为 CC3200 LaunchPad 自带温度传感器的精度较高，所以任务首先读取开发板自带的温度传感器值，再读取 DHT11 温/湿度传感器的

湿度值后，判断室内是否有有害气体，然后分别判断台灯、风扇的关键字，从而在 12864 液晶显示模块上显示台灯、风扇的最新状态。

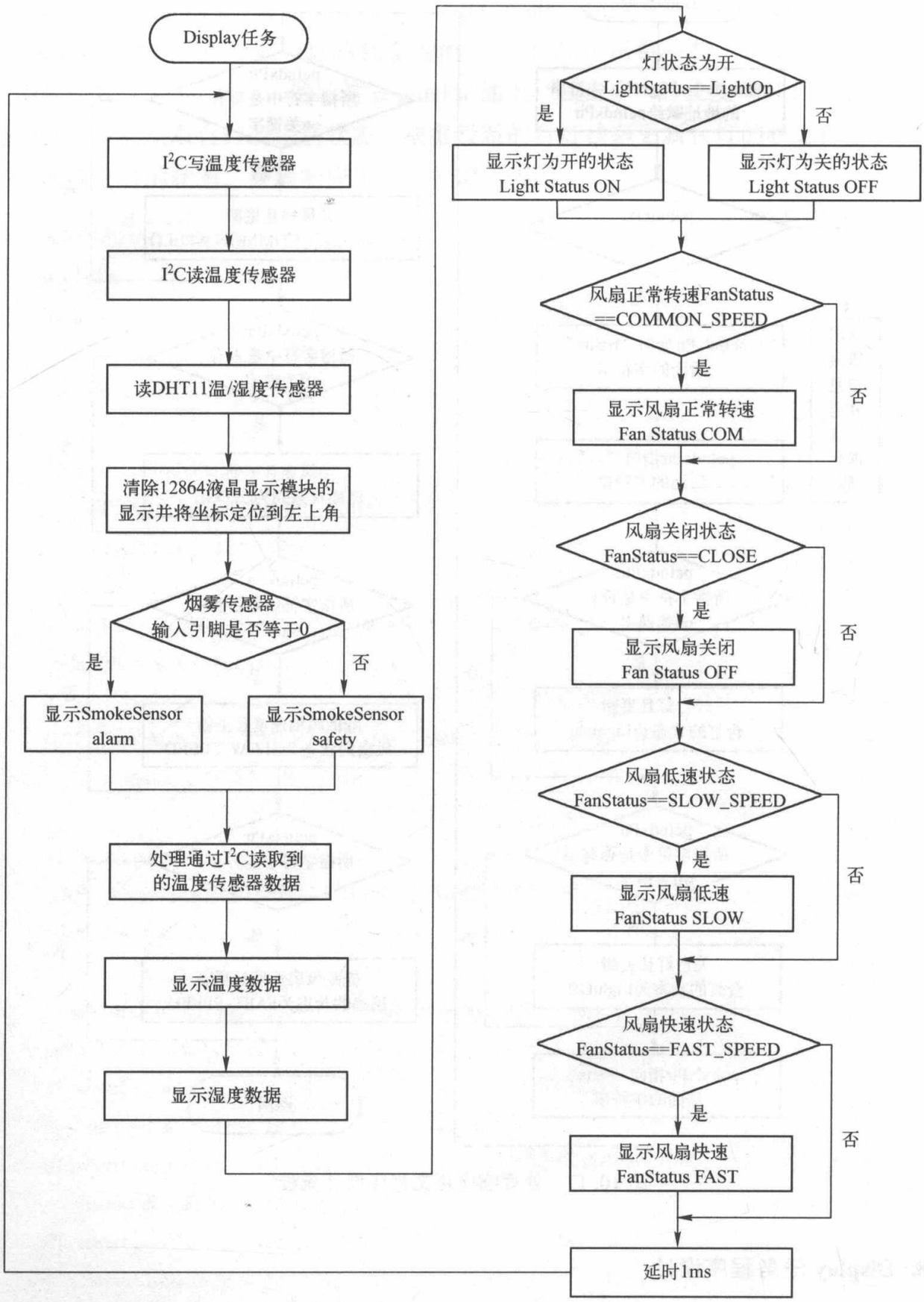


图 10.18　Display 任务程序设计流程

10.3.2 阿里云服务器程序设计

1. 微信请求接口设计

微信服务器把用户发送的消息（文本、语音、菜单）以 XML 的代码形式转发到阿里云服务器，转发的 URL 为 http://139.129.9.166/WeChat/wechat/index.do。微信请求接口程序设计流程如图 10.19 所示。

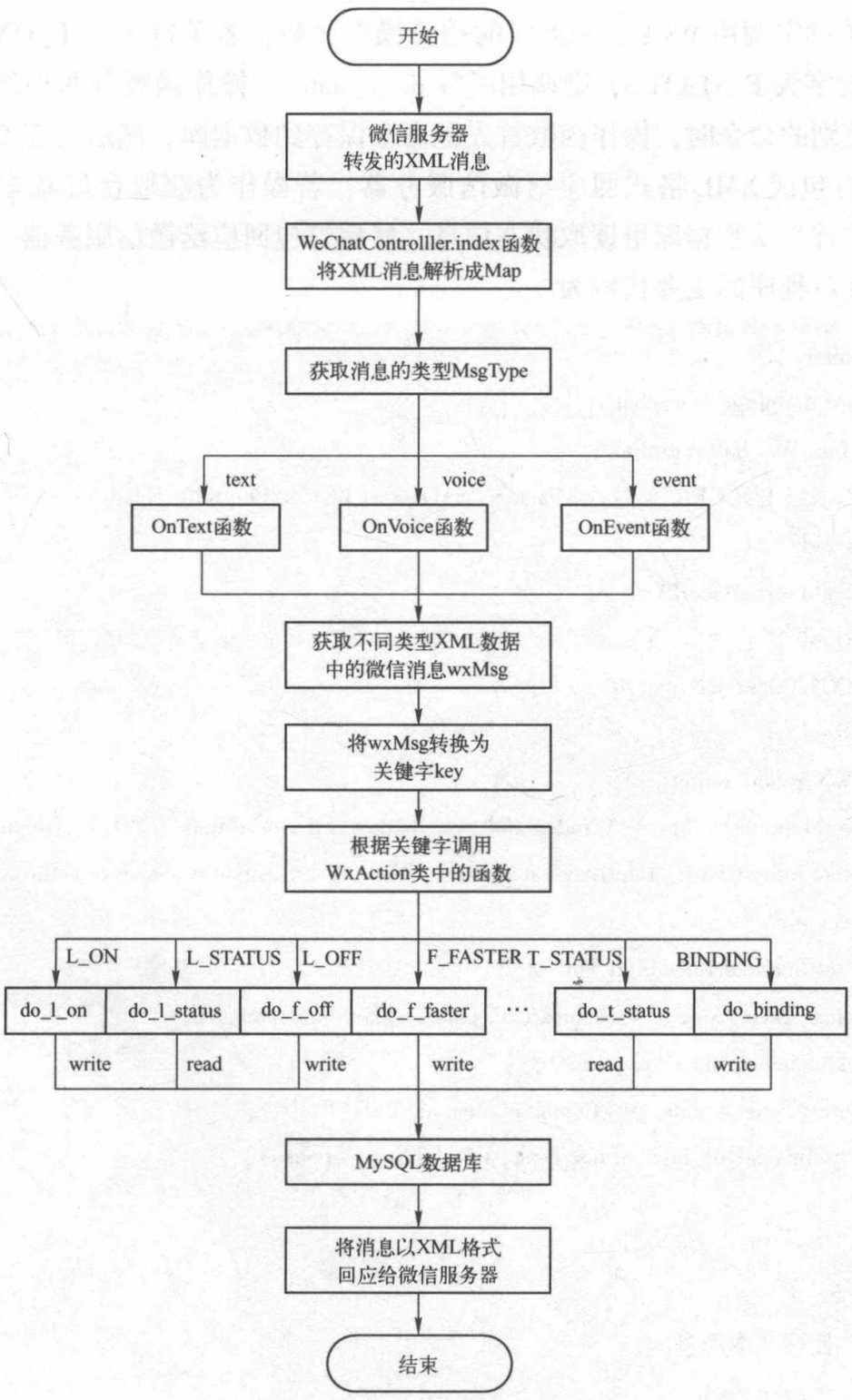


图 10.19　微信请求接口程序设计流程

图 10.19 的具体流程如下：

（1）请求经过阿里云服务器的处理后，分发到 WeChatController.java 文件中的 index 函数进行分析。此函数先将 XML 消息解析成 Map，然后获取消息的类型 MsgType：文本消息对应 text；语音消息对应 voice；菜单消息对应 event。

（2）根据消息的类型分别路由到不同的函数进行处理：文本、语音和菜单消息分别对应函数 onText、onVoice、onEvent，在这些函数中获取不同类型数据中的微信消息 wxMsg，然后判断 wxMsg 中是否存在命令信息，如开台灯、风扇状态等命令。若存在，则将其转换为关键字 key，如 L_ON、F_STATUS 等。

（3）根据关键字调用 WxAction 类中的相应操作函数：若关键字为 L_ON，则调用函数 do_l_on；若关键字为 F_STATUS，则调用函数 do_f_status。操作函数分两种处理方式：若操作为开台灯等类别的命令时，操作函数首先把命令保存到数据库，然后把需要回复的文本及对应语音 URL 打包成 XML 格式回应给微信服务器；若操作为获取台灯状态等类型的命令时，则操作函数首先从数据库里读取状态信息，然后打包回应给微信服务器。

微信请求接口程序的主要代码为

```
@Controller
@RequestMapping("/wechat")
public class WeChatController {
private Logger LOGGER = LoggerFactory.getLogger(WeChatController.class);
@Autowired
private FollowerService fService;
@Autowired
private CC3200Service cService;
@Autowired
private WXAction action;
@RequestMapping(value = "/index.do", method = {RequestMethod.POST, RequestMethod.GET})
public void index(HttpServletRequest request, HttpServletResponse response) throws UnsupportedEn-
codingException {
request.setCharacterEncoding("utf-8");
Map<String, String> data = RequestUtil.parseDataToMap(request);
if (checkSequence(data, response)) {
  String msgType = data.get(CommonConstant.MSG_TYPE);
  MsgTypeInvokeUtil.invoke(msgType, this, data, response);
 }
}
    /**
     * 处理文本消息
     * @param data
```

```
     * @param response
     * @throws IOException
     */
    private void onText(Map<String, String> data, HttpServletResponse response) throws
IOException {
        String wxMsg = data.get(CommonConstant.CONTENT);
        String key = RequestUtil.parseMsgToKey(wxMsg);
        MsgInvokeUtil.invoke(key, action, data, response);
    }
    /**
     * 处理语音消息
     * @param data
     * @param response
     * @throws IOException
     */
    private void onVoice(Map<String, String> data, HttpServletResponse response) throws
IOException {
    String wxMsg = data.get(CommonConstant.RECOGNITION);
    String key = RequestUtil.parseMsgToKey(wxMsg);
    MsgInvokeUtil.invoke(key, action, data, response);
    }
    /**
     * 处理事件消息
     * @param data
     * @param response
     * @throws IOException
     */
    private void onEvent(Map<String, String> data, HttpServletResponse response) throws
IOException {
        String event = data.get(CommonConstant.U_EVENT);
        // 微信菜单消息
        if (event.equals(CommonConstant.CLICK)) {
    String wxMsg = data.get(CommonConstant.EVENT_KEY);
    String key = RequestUtil.parseMsgToKey(wxMsg);
    MsgInvokeUtil.invoke(key, action, data, response);
    }
    // 关注
    else if (event.equals(CommonConstant.SUBSCRIBE)) {
            String follower = data.get(CommonConstant.FROM_
```

```
USER_NAME);
                fService.addFollower(follower);
                    String content = "欢迎关注智能云家居公众号,请单击右下方查看用户手册!";
action.sendResponse(data, content, ResponseContainer.MUSIC.get(MsgKey.E_MANUAL), re-
sponse);
        }
    // 取消关注
    else if (event.equals(CommonConstant.UN_SUBSCRIBE)) {
    String follower = data.get(CommonConstant.FROM_USER_NAME);
    String sequence = fService.getFollowerSequence(follower);
    if (! Strings.isNullOrEmpty(sequence)) {
        cService.removeSensor(sequence);
    }
        fService.removeFollower(follower);
    }
    }
    /**
     * 处理其他消息
     * @param data
     * @param response
     * @throws IOException
     */
    private void onDefault(Map<String, String> data,HttpServletRes ponse response) throws IOExcep-
tion {
        action.do_default(data, response);
    }
    private boolean checkSequence(Map<String, String> data,HttpSer vletResponse response) {
        // 如果是关注/取消关注公众号,则直接放过
        String eventKey = data.get(CommonConstant.U_EVENT);
                if (! Strings.isNullOrEmpty(eventKey) && (eventKey.equals
(CommonConstant.SUBSCRIBE) || eventKey.equals(CommonConstant.UN_SUBSCRIBE)))
        {

            return true;
        }
        String content = data.get(CommonConstant.CONTENT);
        if (! Strings.isNullOrEmpty(content) && content.contains("绑定"))      {
            return true;
```

```
        }
String follower = data.get(CommonConstant.FROM_USER_NAME);
String sequence = fService.getFollowerSequence(follower);
if (Strings.isNullOrEmpty(sequence)) {
action.sendResponse(data, "尚未绑定任何设备，请输入”绑定(设备序列号)”进行绑定", null,
response);
return false;
} else {
    data.put(CommonConstant.SEQUENCE, sequence);
    return true;
    }
  }
}
```

2. CC3200 数据交互接口程序设计

CC3200 通过 HTTP GET 方式进行数据的上传和下载，请求的 URL 为/WeChat/cc3200/upload_and_get.do? sequence = number1&temperature = %.2f&humidity = %d。其中，temperature 和 humidity 的数据通过此 URL 进行上传，主要代码为

```
@Controller
@RequestMapping("/cc3200")
public class CC3200Controller {
  private Logger LOGGER = LoggerFactory.getLogger(CC3200 Controller.class);
    @Autowired
    private CC3200Service cService;
    private static final String formatter = "light=%s,fan=%s";
    @RequestMapping(value = "/upload_and_get.do", method = {RequestMethod.POST, Request-
Method.GET})
    public void uploadAndGet(SensorVO input, HttpServletRequest request, HttpServletResponse re-
sponse) throws IOException {
        SensorDO sensor = new SensorDO();
        String light = input.getLight();
        if (! Strings.isNullOrEmpty(light)) {
            sensor.setLight(light);
        }
        sensor.setTemperature(input.getTemperature());
        sensor.setHumidity(input.getHumidity());
        // 保存温度/湿度信息
```

```
            cService. saveSensor( sensor, input. getSequence( ) );
            // 获得光照/风扇状态
            sensor = cService. getSensor( input. getSequence( ) );
            if ( sensor == null) {
                sensor = new SensorDO( );
                sensor. setLight( CommonConstant. OFF);
                sensor. setFan( CommonConstant. OFF);
            }
            String result = String. format( formatter, sensor. getLight( ), sensor. getFan( ) );
            request. setAttribute( "memcached_key", result);
            response. getWriter( ). print( result + "<hr>" + " from WeChat Backend Server" );
        }
    }
```

CC3200 数据交互接口程序设计流程如图 10. 20 所示。其具体过程为：CC3200 的 HTTP-GET 请求经阿里云服务器处理后，分发到 CC3200Controller. uploadAndGet 函数进行处理，从请求的 URL 中取出 CC3200 上传的温/湿度和天然气浓度值存入 MySQL 数据库后，从数据库中读取台灯等设备的状态，通过 HTTP 协议响应给 CC3200，从而可以更新设备的状态。

图 10. 20　CC3200 数据交互接口程序设计流程

3. 阿里云数据库设计

阿里云数据库设计需要在阿里云 MySQL 数据库中建立两个表：CC3200 表和 follower 表。数据库表代码见表 10.4。

表 10.4　数据库表代码

数据库表	代　码
CC3200 表	CREATE TABLE `cc3200` (`id` INT(11) NOT NULL AUTO_INCREMENT, `temperature` VARCHAR(64) DEFAULT NULL, `fan` VARCHAR(64) DEFAULT NULL, `light` VARCHAR(64) DEFAULT NULL, `humidity` VARCHAR(64) DEFAULT NULL, `type` INT(11) DEFAULT NULL, `extension` VARCHAR(64) DEFAULT NULL, PRIMARY KEY (`id`)) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8.
follower 表	CREATE TABLE `follower` (`id` INT(11) NOT NULL AUTO_INCREMENT, `user_name` VARCHAR(126) DEFAULT NULL, `type` INT(11) DEFAULT NULL, `extension` VARCHAR(64) DEFAULT NULL, PRIMARY KEY (`id`)) ENGINE=INNODB AUTO_INCREMENT=30 DEFAULT CHARSET=utf8.

CC3200 表包含温度值 temperature、风扇状态 fan、台灯状态 light、湿度值 humidity；follower 表包含加密后的微信用户名，如图 10.21 所示。

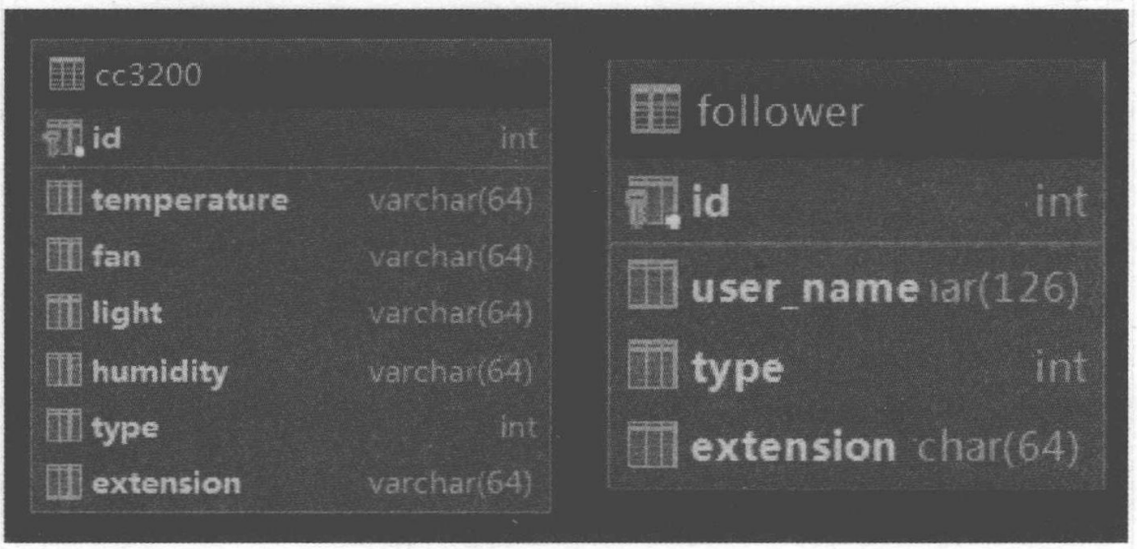


图 10.21　阿里云 CC3200 表和 follower 表

10.3.3 微信公众账号程序设计

微信公众账号程序设计就是要在微信公众平台上创建微信公众号的菜单。微信公众平台上自定义的菜单有助于丰富公众号的界面，让用户更好更快地理解公众号的功能。自定义的菜单接口可实现多种类型的按钮，如 click 类型的按钮、view 类型的按钮等。

其中，用户单击 click 类型的按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给阿里云服务器，并且带上按钮中所填写的 key 值，如台灯的状态等；用户单击 view 类型的按钮后，微信客户端将会打开在按钮中填写的网页 URL，如在本设计中会打开用户手册 URL：http://r. xiumi. us/stage/v3/2sjga/12658678，用户可以通过单击公众号右下角的“用户手册”查看该公众号如何使用。

微信公众号菜单的创建步骤如下。

1. 获取 access_token 接口

获取 access_token 接口的界面如图 10.22 所示。接口类型选择“基础支持”，参数列表中的 grant_type 填写 client_credential，appid 和 secret 填写第 6 章中介绍的微信测试账号界面中的数值。

图 10.22　获取 access_token 接口的界面

单击“检查问题”便可以获取 access_token 接口，access_token 接口的获取结果如图 10.23所示。

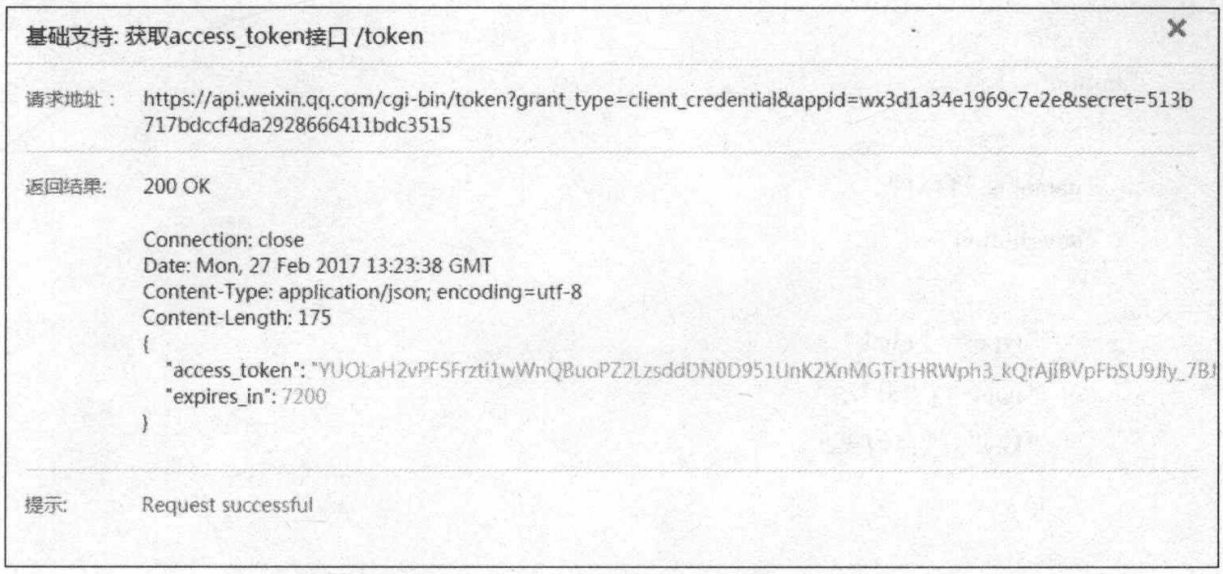


图 10.23　access_token 接口的获取结果

2. 填写自定义菜单 JSON 代码

填写自定义菜单 JSON 代码的界面如图 10.24 所示。

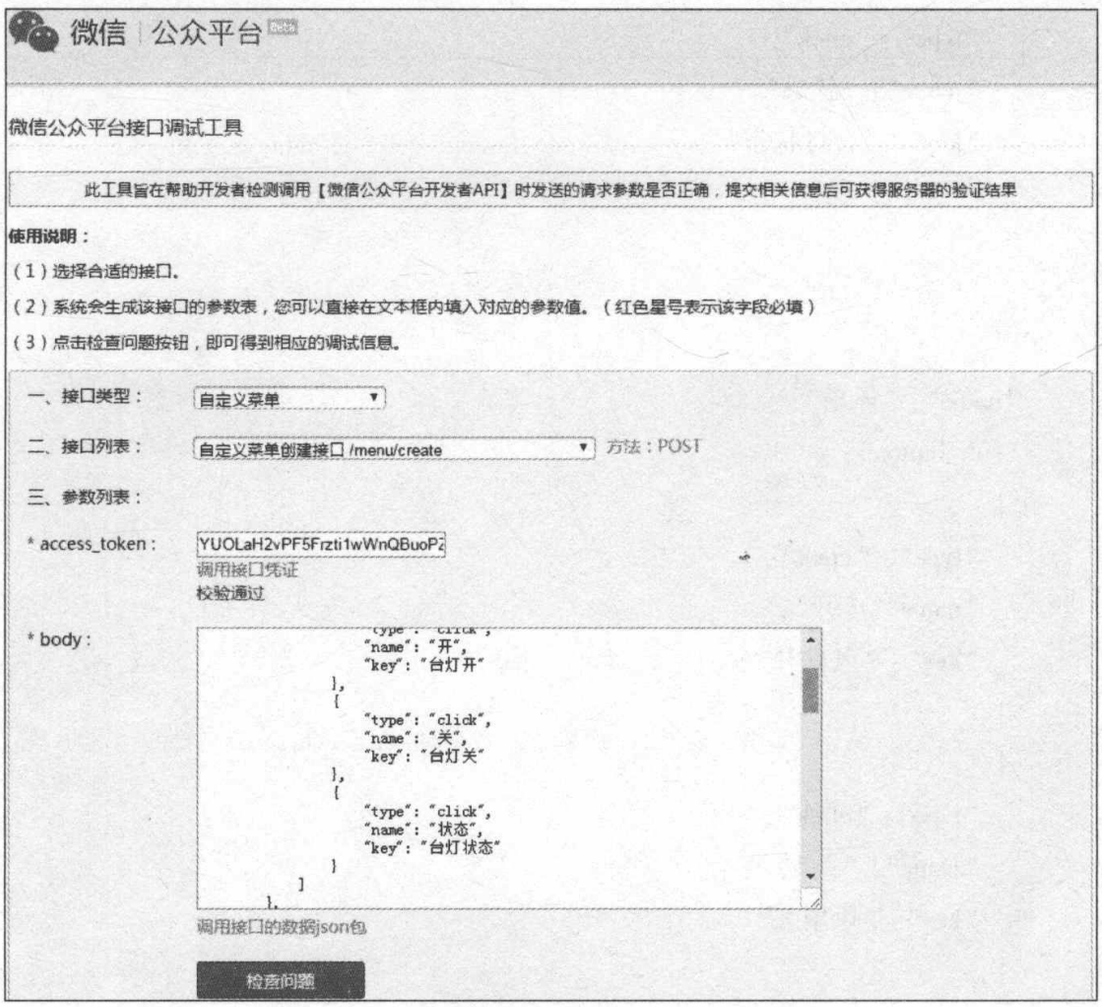


图 10.24　填写自定义菜单 JSON 代码的界面

接口类型选择为“自定义菜单”，access_token 填入步骤 1 中获取的接口，然后在“body”中填入以下自定义菜单 JSON 代码，即

```
{
  "button":[
    {
     "name":"台灯",
     "sub_button":[
      {
        "type":"click",
        "name":"开",
        "key":"台灯开"
      },
      {
        "type":"click",
        "name":"关",
        "key":"台灯关"
      },
      {
        "type":"click",
        "name":"状态",
        "key":"台灯状态"
      }
     ]
    },
    {
     "name":"风扇",
     "sub_button":[
      {
        "type":"click",
        "name":"开",
        "key":"风扇开"
      },
      {
        "type":"click",
        "name":"关",
        "key":"风扇关"
      },
```

```
            {
              "type": "click",
              "name": "加速",
              "key": "风扇快"
            },
            {
              "type": "click",
              "name": "减速",
              "key": "风扇慢"
            },
            {
              "type": "click",
              "name": "状态",
              "key": "风扇状态"
            }
          ]
        },
        {
          "name": "我的",
          "sub_button": [
            {
              "type": "click",
              "name": "室内温度",
              "key": "温度"
            },
            {
              "type": "click",
              "name": "室内湿度",
              "key": "湿度"
            },
            {
              "type": "view",
              "name": "用户手册",
              "url": "http://r.xiumi.us/stage/v3/2sjga/12658678"
            }
          ]
        }
      ]
    }
```

单击“检查问题”，自定义菜单返回结果界面如图 10.25 所示。

自定义菜单: 自定义菜单创建接口 /menu/create

请求地址： https://api.weixin.qq.com/cgi-bin/menu/create?access_token=YUOLaH2vPF5Frzti1wWnQBuoPZ2LzsddDN0D951UnK2XnMGTr1HRWph3_kQrAjIBVpFbSU9Jly_7BJ1YrLezepHGrLNPTK4hOqBw3KgeCsy_WMLpqcnH3VmsWhG2dEpxXZEaAEANPU

返回结果: 200 OK

```
Connection: keep-alive
Date: Mon, 27 Feb 2017 13:29:03 GMT
Content-Type: application/json; encoding=utf-8
Content-Length: 27
{
   "errcode": 0,
   "errmsg": "ok"
}
```

提示: Request successful

基础支持: 获取access_token接口 /token

图 10.25　自定义菜单返回结果界面

此时使用手机登录微信，打开智能云家居公众号就可以看到下边已经创建好的菜单，如图 10.26 所示。

图 10.26　微信公众号自定义菜单界面

10.4 系统测试

10.4.1 测试前的准备

① 开启路由器并接入互联网，给 CC3200 LaunchPad 开发板供电。

② 查看开发板上灯的状态，确保开发板已经连接到路由器。

③ 打开微信，关注智能云家居微信公众号，输入设备 ID 及密钥绑定的设备后便可进行控制。

10.4.2 CC3200 及其外围模块功能的测试

测试时，首先给 CC3200 LaunchPad 供电，此时电源灯正常点亮，开发板上的红色小灯不断闪烁，表明 CC3200 正在通过路由器接入互联网，发送 HTTP GET 请求到阿里云服务器，当红色小灯由闪烁变为常亮时，表示成功接入互联网并且发送请求成功。

此时，12864 液晶显示模块显示的内容如图 10.27 所示。图中，烟雾传感器的状态为“safety”；室内温度为 28.78℃；室内湿度为 35%RH；台灯的状态为 ON；风扇的状态为 OFF；最下方显示的是系统时间，方便用户查看当前的时间。

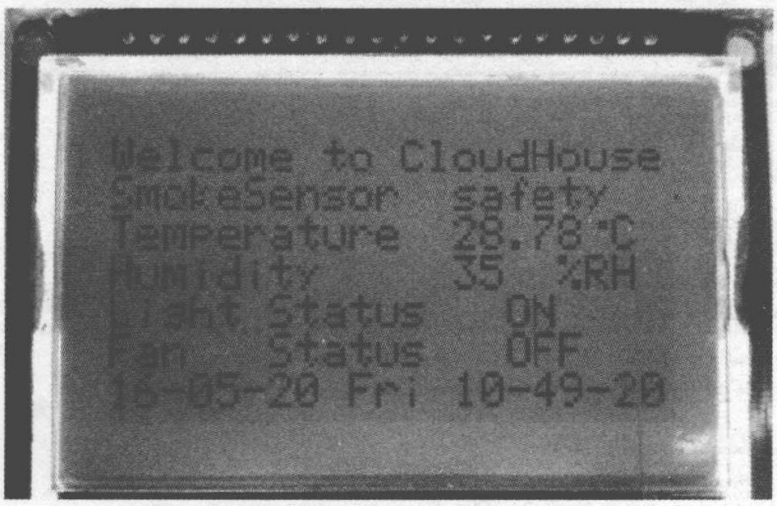


图 10.27 12864 液晶显示模块显示的内容

串口调试工具 Tera Term 显示的内容如图 10.28 所示。从图中可以看到，访问阿里云服务器 IP 的地址为 139.129.9.166，请求的 URI 为/WeChat/cc3200/upload_and_get.do?sequence=number1&temperature=29.46&humidity=35。其中，温/湿度数据通过此 URI 上传。另外还可以看到服务器返回的 HTTP 状态代码为 200，代表客户端请求成功，请求的响应正

文为 light = on , fan = off。

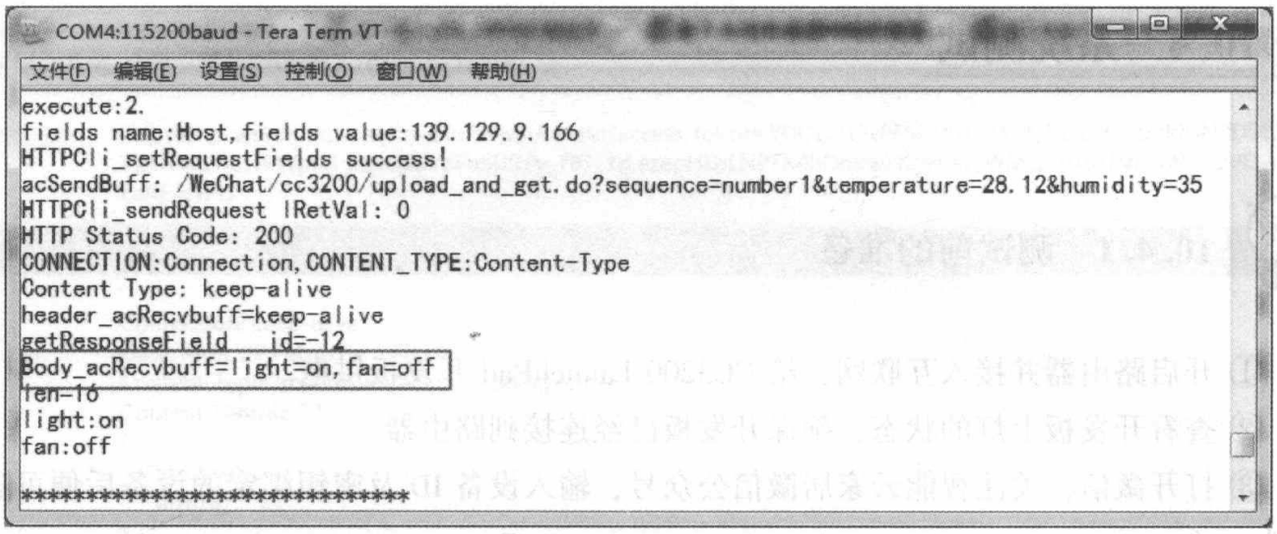


图 10.28　串口调试工具 Tera Term 显示的内容

10.4.3　阿里云服务器功能的测试

（1）使用 Xshell 5 软件通过 SSH 协议远程登录阿里云服务器，主机名为 139.129.9.166，端口号为 22，登录后切换至工作目录进行开发。

（2）在工作目录下使用命令“git clone”获取码云上的最新代码，成功后，运行脚本文件 publish.sh 对代码进行编译。获取码云代码并编译的界面如图 10.29 所示。

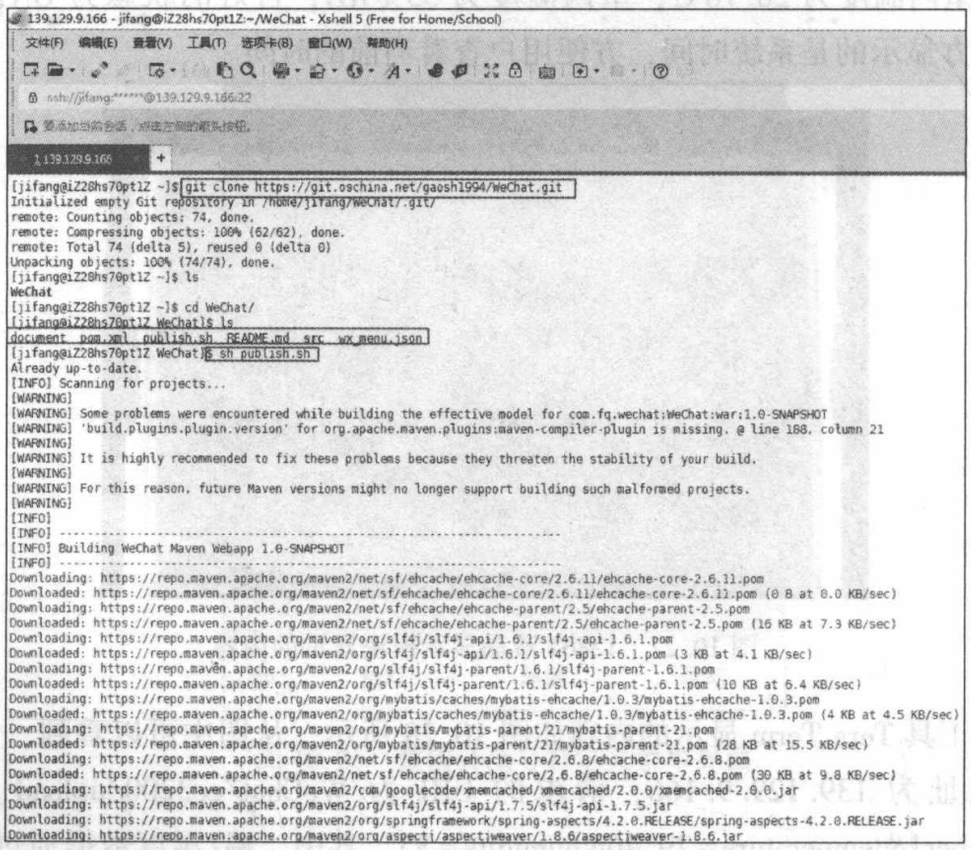


图 10.29　获取码云代码并编译的界面

图中，publish. sh 脚本文件代码为

```
#! /bin/bash
## 判断是否执行成功
check_result() {
if [ "$1" != "0" ] ; then
exit -1
fi
}

## 拉取最新代码
git pull
check_result $?

## 重新打包
rm -rf target
mvn package -Dmaven. test. skip=true

check_result $?

## 拷贝 war 包到目标目录
rm -rf /usr/local/tomcat-8082/webapps/WeChat *
mv ./target/WeChat. war /usr/local/tomcat-8082/webapps/

check_result $?
start_time=`date +%s`

## 重启 tomcat
cd /usr/local/tomcat-8082/bin/
echo "--> shutdown. sh"
./shutdown. sh
echo "--> startup. sh"
./startup. sh

end_time=`date +%s`
sum=$(( $end_time-$start_time ))
echo "total consuming $sum ms"

jps
```

(3) 使用 SQLyog Ultimate 32 软件连接阿里云服务器的数据库，SQL 主机的地址为 139. 129. 9. 166，用户名为 admin，如图 10. 30 所示。

图 10. 30　建立阿里云服务器的数据库连接

(4) 连接成功后，可以查看 CC3200 表和 follower 表。CC3200 表中的内容如图 10. 31 所示。在接下来微信公众号的测试中可以通过微信号改变风扇或台灯的状态，然后查看 CC3200 表中的数据是否改变。

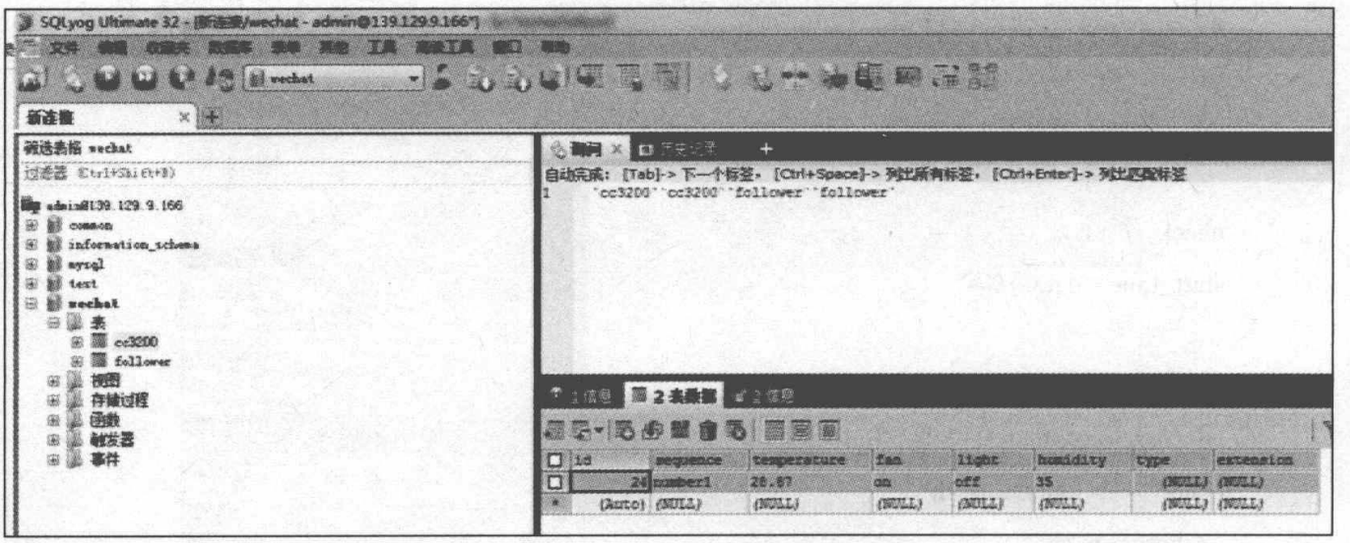


图 10. 31　CC3200 表中的内容

10. 4. 4　微信公众号功能的测试

1. 菜单功能的测试

扫描微信公众号的二维码，首次关注该公众号及打开用户手册的界面如图 10. 32

所示。

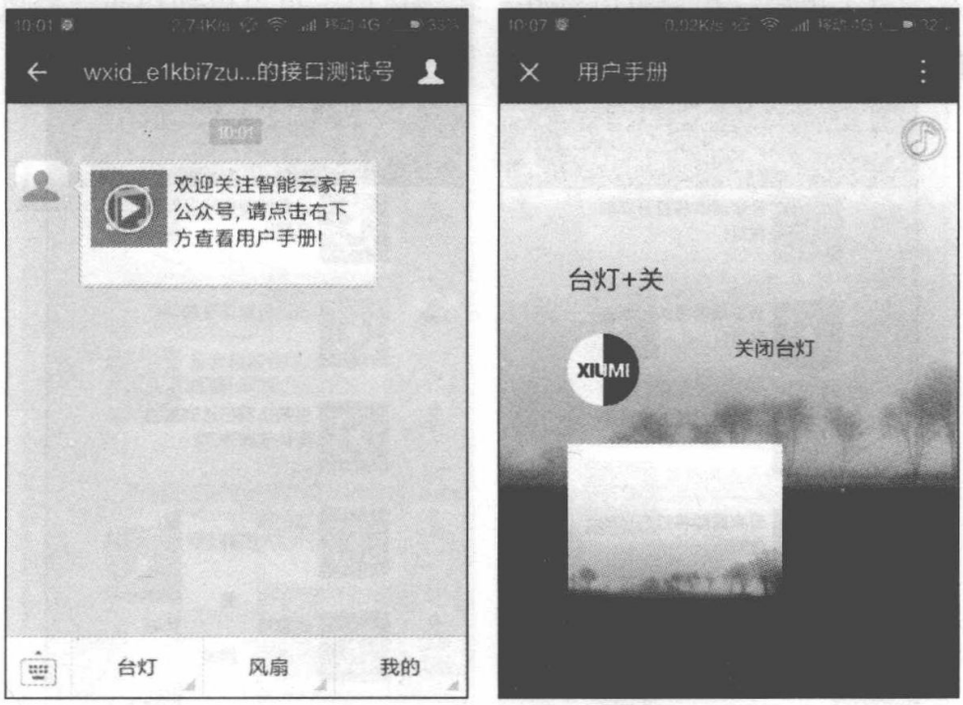


图 10.32　首次关注该公众号及打开用户手册的界面

首次关注之后，需要用户绑定设备，根据提示在测试号中输入“绑定（number1）”即可实现测试账号与家居系统的绑定。绑定之后，可以通过菜单获取室内的温/湿度，如图 10.33所示。

图 10.33　用户绑定设备和获取温/湿度

控制台灯和风扇的界面如图 10.34 所示。对于台灯，用户可以打开、关闭台灯，也可以查看台灯的状态；对于风扇，用户可以打开、关闭风扇，也可以加快或减慢风扇的速度，还可以查看风扇的状态。

图 10.34　控制台灯和风扇的界面

2. 文本和语音功能测试

（1）通过发送文本和语音消息控制台灯的界面如图 10.35 所示。

图 10.35　通过发送文本和语音消息控制台灯的界面

（2）通过发送文本和语音消息控制风扇的界面如图 10.36 所示。

图 10.36　通过发送文本和语音消息控制风扇的界面

（3）通过发送文本和语音消息获取温/湿度的界面如图 10.37 所示。

图 10.37　通过发送文本和语音消息获取温/湿度的界面

{
  "filename": "MTQ1Mjk0ODUuemlw",
  "filename_decoded": "14529485.zip",
  "filesize": 116536271,
  "md5": "b0eecf46f2c831753141545726b5edc1",
  "header_md5": "27992202337d5a8b4eeba07af973ad72",
  "sha1": "a00203ed8d682ae4decb151627af3c9647fd1f9a",
  "sha256": "5c916aa6d4a6782282397eae3d8e44fb48f999de19cfd187c339af13bd49d6df",
  "crc32": 1524813127,
  "zip_password": "",
  "uncompressed_size": 128814185,
  "pdg_dir_name": "\u2553\u255f\u2500\u2584\u255d\u2565\u255b\u2559\u2510\u256a\u2553\u255e\u2567\u2561\u2550\u2502\u2561\u2500\u2554\u03a6\u255d\u255e\u2559\u03b4\u2510\u00ac\u2556\u253c_TI
CC3200+\u256c\u2229\u2534\u00ac\u2550\u00b0\u2558\u255e\u255e\u255c\u2560\u00bf+\u256c\u00f3\u2568\u253c_14529485",
  "pdg_main_pages_found": 229,
  "pdg_main_pages_max": 229,
  "total_pages": 239,
  "total_pixels": 1383728424,
  "pdf_generation_missing_pages": false
}