基于STM32的嵌入式家庭智能终端设计

摘要

摘要内容。

**关键词：**STM32，Socket，Web服务器，微信小程序

Design of Smart Embedded Domestic Terminal Based on STM32

ABSTRACT

Abstract.

**Key words:** STM32, Socket, Web Server, Miniprogram

目录

摘要 I

ABSTRACT II

目录 III

1 绪论 1

1.1 课题意义 1

1.2 研究现状 1

1.3 主要工作 3

2 架构设计 5

2.1 系统架构 5

2.1.1 嵌入式设备 5

2.1.2 服务器 6

2.1.3 交互应用程序 7

2.2 通信接口 7

2.2.1 嵌入式设备-服务器 7

2.2.2 服务器-交互应用程序 8

3 硬件设计 11

3.1 硬件选型 11

3.1.1 微控制器 11

3.1.2 摄像头 13

3.1.3 温湿度传感器 14

3.2 硬件连接 14

4 软件设计 16

4.1 嵌入式设备开发 16

4.1.1 μC/OS-II环境配置 16

4.1.2 lwIP协议栈移植 18

4.1.3 HTTP与Socket网络通信 19

4.1.4 传感器数据捕获 20

4.2 Web服务器开发 22

4.2.1 设计模式 23

4.2.2 用户管理 23

4.2.3 设备校验 24

4.2.4 数据中转 24

4.3 Socket服务器开发 25

4.3.1 Socket创建 25

4.3.2 监听线程 25

4.3.3 数据缓存 26

4.4 交互应用程序开发 27

4.4.1 用户登录 29

4.4.2 设备绑定 29

4.4.3 请求数据 29

5 项目测试 31

5.1 嵌入式设备测试 31

5.1.1 操作系统测试 31

5.1.2 网络通信测试 31

5.1.3 传感器测试 32

5.2 服务器测试 32

5.2.1 基于pytest的测试环境配置 33

5.2.2 编写测试用例 33

5.2.3 测试结果 34

5.3 交互应用程序测试 34

5.3.1 编写模拟设备客户端 35

5.3.2 数据请求测试 35

5.4 系统综合测试 36

6 总结与展望 38

参考文献 39

致 谢 41

1 绪论

1.1 课题意义

一直以来，居家安全是日常生活中每个人都关注的重要话题。物联网普及之前，人们出门在外几乎无法实时掌握家中的情况，也无法及时对突发事件做出有效的反应，这在一定程度上为人们的日常生活带来了安全隐患。然而，随着信息科技的发展，各种新兴的技术正不断地涌入并填补这一领域的空白。物联网时代到来后，传感技术、网络技术、人工智能技术和自动控制技术的进步与融合使得设备之间的对话成为可能，从而将物联网的应用带入了家居、物流、农业、交通、医疗等各个领域。同时，嵌入式系统作为物联网设备端的核心部件，因其低功耗、低成本、高效率、软件硬件可裁剪等优点得以迅速发展并成功占有物联网智能化市场。随着技术限制的逐步突破，层出不穷的新产品为广大消费者提高居家生活安全性和便利性提供了越来越多样化的选择。

插图

图1.1 应用现状

在这一背景下，综合考虑领域研究现状、市场需求、技术安全性以及可实现性，从而设计一款能够满足广大消费者对于居家安全与生活便利需求的嵌入式家庭智能终端是十分可行且有必要的。

1.2 研究现状

在现阶段，主流的家庭智能终端可以从系统构成上分为主要工作在本地的嵌入式终端系统与需要协同服务器工作的综合性终端系统。其中，嵌入式终端系统倾向于将信息采集、处理与通信等功能集成于单一设备上，并采用LCD触摸屏或独立的平板电脑等进行人机交互，同时也可以通过蓝牙等方式连接手机以便进行无线控制[1]。这样的设计部署简单，但却只能在设备附近操作，使用灵活性较差。另一方面，综合性终端系统大致在一般上可以分为嵌入式设备、服务器、交互应用程序三个较为独立的部分。嵌入式设备端采集所需要的数据并上报服务器，由服务器处理后中转至相应的移动端设备，并由交互程序进行显示[2]。相较之下，这样的设计能够对设备的进行真正意义上的远程监控，同时由于数据采集与处理二者实现了分离，这种架构也减轻了嵌入式CPU的运算负担，为总体上进一步减少成本提供了可能。

对于嵌入式设备端，控制器的选择较为常见的包括两种。第一种方案是树莓派[3]，其作为一种功能强大而全面的嵌入式开发平台，尽管能够完全满足家庭智能终端的设计需求，但由于其造价相对高昂，同时也造成了包括蓝牙、USB、音视频输入输出在内的大量硬件功能与CPU算力的冗余，因此并不完全适用于此种环境。另一种方案是STM32，开发者可以充分利用其高度的可定制性与丰富的硬件接口[4]。具体而言，F1系列是小型的终端设备中经常被选择的型号，其在价格低廉的同时可以轻松胜任环境参数的检测、少量数据的网络传输与简单的图像处理任务[5]；而在较为大型的终端设备中更为常见的是F4系列，其具有更强的运算能力、更快的处理速度、更丰富的对于网络功能支持以及更完备的硬件设备接口（包括专门的以太网口与数字摄像头接口DCMI），同时还集成了部分DSP指令与FPU运算单元，因而能够进行更为高级的图像处理与网络通信任务，也可以轻松的加载诸如μC/OS[6]、FreeRTOS[7]、μCLinux[8]等嵌入式操作系统以支持更为复杂的任务调度，因而是一种非常贴合应用实际的选择。STM32F1、F4与树莓派系列的资源与性能比较如下表。

表1.1 STM32F1、F4与树莓派的比较

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | STM32F1 | STM32F4 | 树莓派 |
| CPU |  |  |  |
| 存储 |  |  |  |
| 外设 |  |  |  |
| 价格 |  |  |  |

在一些智能家居系统的设计中，对于终端设备会采用集群模式，例如在厨房、卧室、客厅等不同的房间中安放多个终端设备，并借助ZigBee技术实现同一组网下不同终端之间的信息交互[9]。此种大型系统虽然具有更加丰富完善的功能，但同时也有着结构复杂、部署困难、维护成本高等诸多缺点。此外，类似系统所包含的如电动窗帘、灯光调节与家电控制等部分功能的实现需要依赖于完全配套的被控设备[10]，在现实生活中实则难以运用。因此在一般情形下，由单体设备构成的家庭智能终端其实在很大程度上已经能够满足人们日常生活中的需求。

对于交互应用程序，这一部分的开发在现有的设计方案中并没有受到太多的重视。最常见的移动端交互方案是基于安卓系统的手机App，程序采用Java语言编写并通过服务器与嵌入式设备进行数据与指令的沟通[11]。但考虑到该程序无法运行在装有iOS或Windows系统的移动设备上，而是需要在移植时针对不同操作环境进行专门的设计，开发过程较为繁琐。相较之下，一个兼容性更好的选择是使用Web前端技术[12]，通过浏览器内核屏蔽移动终端在硬件与操作系统层面的差异，使得同样的程序能够以更小的代价运行在更多的设备上。

在传统的家庭智能终端系统中，服务器更多的只是作为数据中转的途径。对于一些轻量级的应用程序，可以直接使用市面上现成的物联网服务器如阿里云或OneNet等，通过已经集成好的Api进行数据交互，并使用诸如网页组件等技术实现数据的可视化与对设备的控制[13]。然而在通常情况下，运营商为了保证服务器面对简单应用需求时的使用友好性，一般会对相应的接口进行高度封装以保证简洁，随之而来的问题便是极大地削弱了服务器功能的灵活性与可定制性。例如OneNet平台不支持文件操作、不暴露数据库接口、也无法提供除网页组件和电子邮件之外任何方式的人机交互，更无法在服务器端进行个性化的运算或数据处理，因而也很难满足广大用户与开发者多样化的需要。

对于该问题的一种解决方案是由开发者自行搭建服务器，例如可以在设备端将采集到的传感器数据通过Socket传送至Apache服务器并暂存，接着由移动端程序使用HTTP请求获取数据并反馈给用户[14]。这样的设计好处在于能够根据项目需求有目标的进行服务器功能的开发，并结合实际情况选用合适的通信协议与接口实现方式。值得注意的是，在现有的基于类似架构的设计中，服务器与设备及用户程序之间的通信一般都是一对一的，即一台服务器在同一时刻只能为单一用户提供服务[15]，这显然是对服务器资源的极大浪费。在真正的项目部署与商用化过程中不可避免地需要应对多用户并发的情形，而这一功能的实现需要解决的主要问题在于用户的登录与鉴权，但该功能的设计与实现显然会造成开发成本的大幅度提高并且可能为系统带来更多的安全风险。

在采用多用户机制的系统设计中，用户与设备通过网络进行交互时的数据安全问题是必须加以考虑的，尤其是对于综合性的终端系统，如何保证私密数据在设备到服务器、服务器到交互程序的传输过程中不被窃取，以及如何保证位于服务器端的用户登录与鉴权机制没有明显漏洞是整个系统设计与开发过程中十分重要的一环[16]。由于任何数据在网络传输过程中都无法完全保证不被窃听，因而需要做的首先便是在嵌入式设备端发起传输之前对需要发送的数据进行加密[17]。其次，还应该在系统设计时预估可能存在的安全风险并采取相应规避措施，如恰当的设计系统不同部分之间的信息交互方式、恰当的选择更为安全的网络通信协议如HTTPS、以及对服务器进行SSL安全认证等。然而，考虑到真实应用场景下的网络信息安全是一项极为复杂的系统性工程，因此该部分的相关内容在本设计中仅做参考而不进行过多的研究。

1.3 主要工作

综合上述讨论，本课题旨在设计一款部署简单、使用便捷、功能丰富而不冗余、移动设备兼容性强、面向多用户且数据安全性较高的嵌入式家庭智能终端系统，能够在嵌入式设备端实现环境参数检测与视频监控等功能；并且在服务器端进行数据中转以及设备与用户的注册、登录与鉴权，以此保证使用时的数据安全；同时开发一款跨平台的移动端交互程序，使不同终端设备的用户均能够方便的通过该程序获取监测状态。

接下来，本文将在第二部分主要介绍该智能终端系统的架构设计，依照上述设计目标制定系统组成、模块功能以及网络接口规范；第三部分介绍系统硬件，阐述设备选型、器件通信方式与实物连接；第四部分介绍软件设计方案，给出嵌入式设备、服务器和交互应用程序三部分的功能设计实现以及各模块之间的交互协同策略；第五部分介绍项目测试方案，对本设计所包含的功能进行从模块级别到系统级别的测试覆盖，之后开展系统综合测试并给出最终评价指标；第六部分对项目设计与开发过程进行回顾总结，分析设计中尚存的问题并提出可能的改进方向。

根据前文提出的项目设计目标与相关现有研究成果，结合实际情况给出该设计的各项评价指标如下表所示。

表1.2 设计评价指标

|  |  |  |
| --- | --- | --- |
| 模块 | 预期效果 | 指标 |
| 嵌入式设备 | 使用STM32实现环境参数的检测、采集与上报，接受来自用户的控制指令 | 载入μC/OS-II操作系统  移植lwIP协议栈  实现HTTP与Socket通信  实现摄像头与温湿度传感器捕获 |
| 服务器 | 使用Flask框架并配置Socket，实现设备数据采集、用户管理与数据中转 | 搭建基于Flask的Web服务器  搭建Socket服务器  实现设备校验与用户管理  实现数据中转 |
| 交互程序 | 使用微信小程序连接服务器，实现用户鉴权与数据可视化 | 实现用户登录  实现设备绑定  实现数据获取与显示 |

2 架构设计

2.1 系统架构

本设计将该家庭智能终端系统分为嵌入式设备、交互应用程序以及服务器三个部分。其中，嵌入式设备负责采集数据并上传至服务器；服务器将设备传输的数据暂存并反馈给移动端；交互应用程序负责从服务器获取相应数据并进行显示，该程序同时携带用户身份信息，以便在服务器端进行登录。

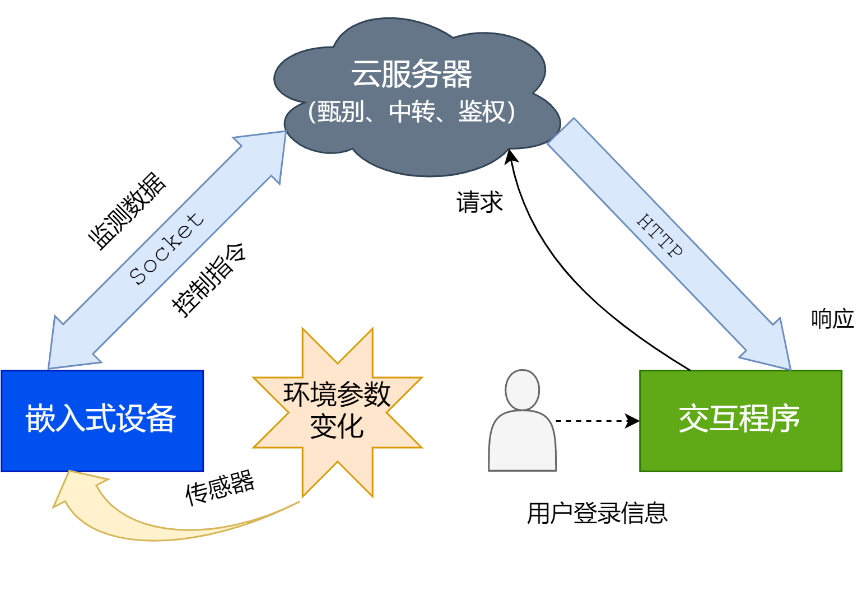


图2.1 系统架构图

2.1.1 嵌入式设备

对于嵌入式设备端，本设计选用STM32F4系列作为控制器，在价格相对低廉的前提下保证了运算速度，同时也可以利用现成的DCMI接口方便的连接摄像头。所有环境参数传感器均通过GPIO与控制器连接。

在网络通信方面，设备提供了专用的以太网口并挂接网卡芯片。但由于该芯片仅工作于链路层，因此应用层网络通信功能的实现需要移植TCP/IP协议栈。本设计选用由瑞典计算机科学学院开发的小型开源轻量级TCP/IP协议栈lwIP，其在保持TCP协议主要功能的前提下减少了对RAM的占用，并加入了操作系统接口以更好地支持多线程，极为适合小型嵌入式设备对于复杂网络通信的需求。

在通信方式的选择方面，由于设备不仅需要进行检测数据的实时上报，还需要时刻等待并响应通过服务器传来的用户指令。在此种情况下，采用Socket进行通信是一种较为合适的选择，它既实现了服务器与客户端的全双工通信机制，保证了信息交互的灵活性和时效性，也在最大程度上减少了不必要的流量开销。

此外，由于控制器需要同时执行温湿度检测、摄像拍照、用户指令响应以及网络传输等多项任务，尤其是网络传输部分的开销相对较大、流程相对复杂、传输发生的时间也相对较为灵活，仅依靠单线程式的简单程序逻辑实现存在一定困难。因此，本设计载入了嵌入式操作系统μC/OS-II以提供更为高级的任务调度机制，特别是能够对于较为复杂的网络通信需求实现更好的支持。

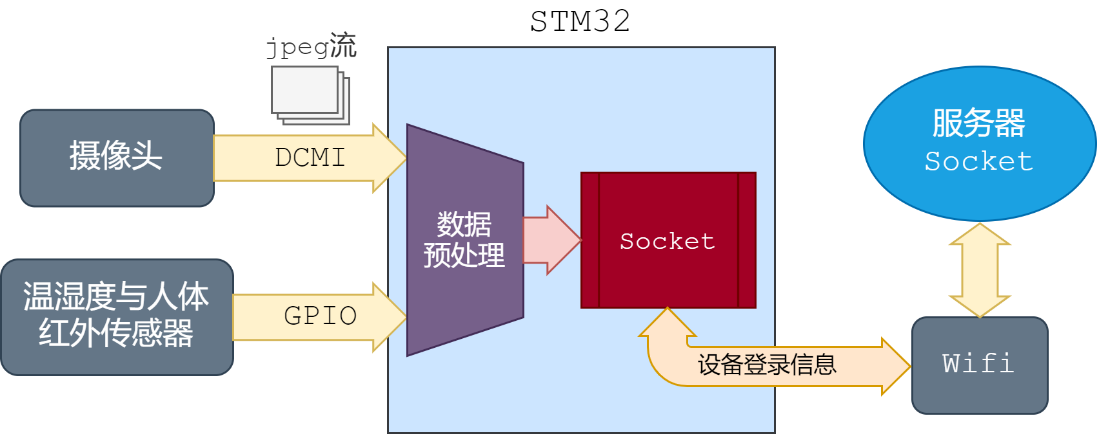


图2.2 嵌入式设备架构图

2.1.2 服务器

为配合嵌入式设备与交互应用程序的正常工作，本设计所采用的服务器需要实现对于Socket通信与HTTP请求的支持。设计拟选用Flask框架作为Web服务器，程序全部使用Python编写，并在此基础上自行配置Socket通信的相关功能，同时连接SQLite数据库以支持用户信息在服务器端的访问与存储。

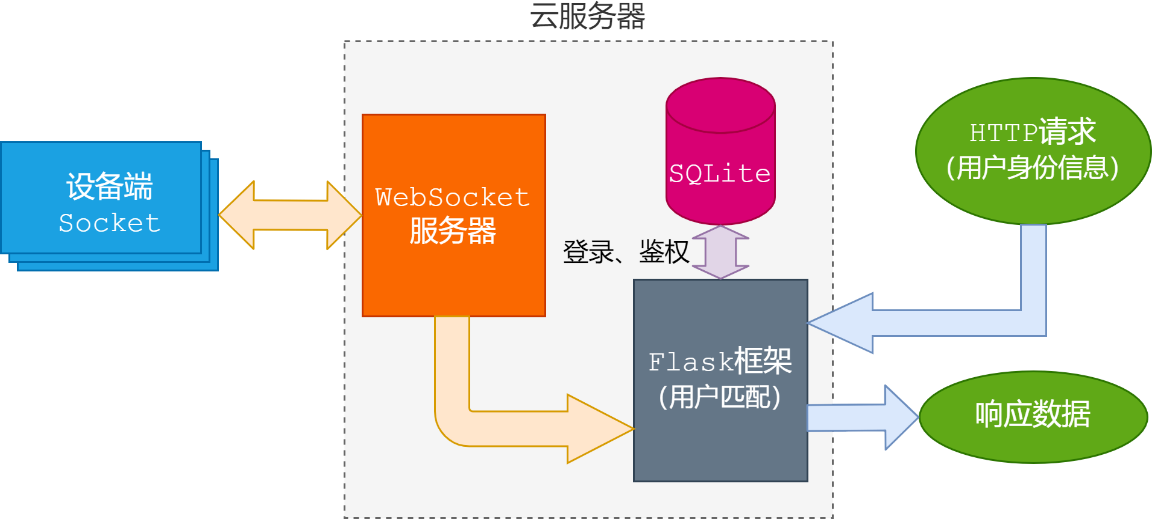


图2.3 云服务器架构图

交互应用程序在运行时通过向服务器发起HTTP请求进行登录、设备绑定、数据提取与控制指令上传等操作，在此过程中服务器将通过Socket接收到的对应设备的数据反馈给发起请求的用户。这样的设计能够屏蔽交互程序与硬件设备的直接通信，在加强了服务器对系统运行状态掌握的同时，也进一步保证了灵活性与用户数据的安全性。

2.1.3 交互应用程序

由于移动端原生App的开发必须依赖于相应的手机操作系统，而为了使交互程序具有更好的兼容性，本设计采用微信小程序作为交互手段，借助Web前端技术使程序能够轻易的运行在所有安装有微信的手机平台上。

插图

图2.4 小程序架构图

微信小程序的开发采用WXML+WXSS+JavaScript的前端技术路线，通过类似于网页开发的组件+事件响应机制与用户进行交互，并利用以HTTP为主的网络传输协议与服务器通信。同直接开发原生App或设计网页相比，采用微信小程序不仅能够通过现成组件的使用极大地减少软件开发成本，同时也能依托微信平台提供的开放能力辅助实现用户登录与用户身份鉴别等功能，从而能够为设计带来更好的兼容性、便捷性、安全性以及更加丰富的功能。

2.2 通信接口

由于本设计所包含的模块数量较多，且不同模块之间的数据交互行为较为繁复，因此有必要在进行具体功能的开发之前根据需求先行设计并制定接口规范。在本设计中，嵌入式设备到服务器以及服务器到交互应用程序之间全部采用网络进行数据传输，但所涉及的具体场景与通信需求又有所区别。因此，本设计分别对两者从软硬件环境、数据类型、可实现性与时效性等角度出发进行具体的分析。

2.2.1 嵌入式设备-服务器

本设计中，嵌入式设备与服务器的通信接口主要负责向服务器传输摄像头与温湿度传感器数据以及向设备下发控制指令。因此，该接口需要采用一种实时性强、允许数据量大的全双工通信机制。

在嵌入式设备端，设计选用的lwIP协议栈对于TCP/IP下的网络通信进行了BSD Socket Api封装，从功能角度已经能够较好地满足需求，但从信息安全角度却为服务器带来了一定的安全隐患。由于简单的Socket通信机制下对于连接请求的发起与接受没有任何限制，则一旦服务器的IP地址与端口号等信息暴露，外界设备将有可能不受控制的通过Socket连接至服务器，这或将造成服务器负担过重甚至崩溃。

因此，本设计在Socket创建过程中加入了设备校验机制。在设备投入使用之前，由设计者为每台设备分配公开的身份标识符（device\_id）与私有的密钥（device\_key），并在服务器启动时创建合法设备IP映射表。当一台设备试图与服务器建立Socket连接时，应首先向服务器提交身份信息进行校验。校验通过后，服务器即判定该请求来自可信设备，之后将该设备ID与IP地址的映射关系存入合法设备IP表，最后为该设备注册独立的Socket监听线程并等待设备发起连接。

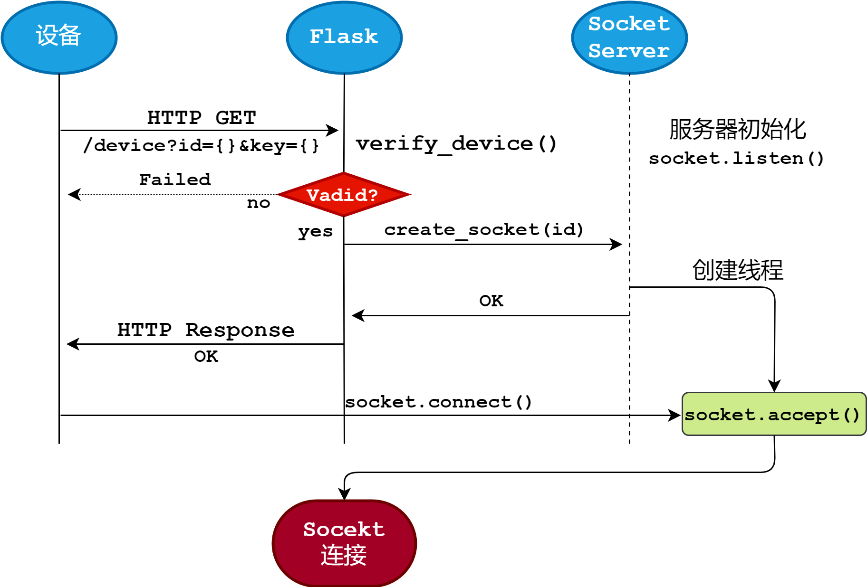


图2.5 设备校验与Socket创建

在监听线程接收到一个新的连接请求后，会首先查询该连接发起者的IP地址是否在服务器创建的合法设备IP表内，若查询成功则允许设备向服务器上传数据，反之则直接关闭连接，并重新启动监听直到接收到一个合法的连接为止。

由于设备身份信息在服务器端存储在SQLite数据库中，而本设计用于响应HTTP请求的Flask服务器已经提供了一套完整的面向数据库的连接规范，因此设备校验的相关工作直接交由Flask完成，具体方式为设备向服务器发起GET请求并携带相关身份信息，之后由处理该请求的视图函数进行数据库连接、数据查询与设备合法性判定，并在验证通过时调用Socket服务器提供的相关接口函数创建监听线程。有关该部分的具体内容见4.2.3章节。

当嵌入式设备与服务器的Socket连接被成功创建后，包括摄像头等传感器信息与用户指令在内的所有数据交互即可全部交由Socket完成。其中，从设备到服务器的摄像头数据采用JPEG格式，温湿度传感器数据则直接发送字符串，由服务器接收到后负责甄别.。

2.2.2 服务器-交互应用程序

本设计用于交互式应用的微信小程序基于Web前端技术，因而其所有与服务器之间的数据交互全部由HTTP协议实现。在面向多用户的应用场景下，任何功能的使用都必首先须进行用户注册与登录，之后在服务器端通过数据库管理用户身份信息，包括生成cookies和会话。借助微信平台的原生功能，设计中并不需要自定义用户身份，而是可以利用微信提供的用户全局唯一标识（openid）作为确定用户身份的最终依据。

用户登陆过程的核心内容在于通过服务器向微信后台请求openid。进行登录时首先需要通过小程序调用微信平台接口并获取临时登录凭证code，接着通过HTTP请求将code发送至服务器。之后，服务器使用code与小程序注册信息向微信后台换取openid，并以此为依据来维护用户在服务器的登录状态。出于安全性考虑，敏感信息如openid与session\_key一般只允许在服务器内部保存而不允许下发至用户程序。



图2.6 用户登录

在获取用户的openid之后，服务器向数据库查询该用户注册的设备信息，成功后则创建会话并向用户返回cookies，反之则返回响应提示用户进行注册。有关登录功能的具体实现见4.2.3章节。

在之后的所有请求中，用户将通过cookies向服务器表明登录身份并访问会话中保存的信息，包括与该用户所绑定的嵌入式设备身份标识（device\_id），服务器将以此为依据进行设备与用户的匹配，以便在多用户同时登陆的情形下向每个用户正确的返回属于其个人绑定设备的数据。本设计中，服务器对交互应用程序提供的所有Api如下表所示。

表2.1 服务器Api列表

|  |  |  |  |
| --- | --- | --- | --- |
| 请求类型 | URL | 参数 | 功能 |
| GET | /login?code=CODE | 微信临时登录凭证 | 用户登录 |
| GET | /login/logout | - | 用户登出 |
| POST | /login/register | 设备身份标识、设备密钥 | 设备绑定 |
| GET | /device/login?id=ID&key=KEY | 设备身份标识、设备密钥 | 设备校验 |
| GET | /stream | - | 请求数据 |
|  |  |  |  |

3 硬件设计

3.1 硬件选型

本章主要阐述系统在嵌入式设备端的开发过程中硬件部分的相关内容。嵌入式设备包含的主要硬件模块BOM清单如下。

表3.1 BOM清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 名称 | 规格 | 数量 | 单价/元 |
| 1 | 微控制器 | STM32F4 | 1 | 448 |
| 2 | 摄像头模块 | OV2640 | 1 | 98 |
| 3 | 温湿度传感器模块 | DHT11 | 1 | 10 |
| 4 | 调试器 | ST-Link | 1 | 78 |

3.1.1 微控制器



图3.1 STM32F4探索者开发板

由于本设计在嵌入式设备端需要载入操作系统并实现较为复杂的传感器检测与高性能网络传输任务，因而对处理器的运算能力与硬件功能支持提出了更高的要求。设计选用了正点原子公司STM32F4探索者开发板，该设备的MCU采用STM32F407VGT6，处理器为带FPU与DSP指令的ARM Cortex-M4内核，包含12个16位定时器，2个32位定时器，2个DMA控制器，以及SDIO、FSMC、10/100M以太网MAC控制器和数字摄像头接口等。正点原子公司在此基础上进行了外围电路设计与并添加了硬件接口，使得该开发板能够在很大程度上发掘处理器的潜力。在本设计中用到的部分硬件资源如下表所示。

表3.2 探索者开发板部分硬件资源

|  |  |
| --- | --- |
| 资源 | 型号说明 |
| CPU | STM32F407VGT6、1M Flash、192k SRAM |
| 存储设备 | 外扩1M SRAM |
| 硬件接口 | 100M以太网口、摄像头接口、DHT11接口 |
| 其他 | LED、按键、JTAG/SWD、USB转TTL串口、12V稳压电源 |

考虑到一些大型应用中MCU携带的192k SRAM可能面临内存不足，开发板通过FSMC挂载了XM8A51216芯片，能够提供1M的额外内存空间。在此基础上，正点原子公司在官方例程中提供了统一封装的动态内存分配接口，对MCU内部SRAM、外扩SRAM以及CCM三个内存池实现分块式管理，极大的方便了开发者的使用。

为了实现网络通信功能，MCU集成了MAC 802.3介质访问控制器，通过RMMI与外部PHY芯片通信并提供了独立的DMA控制器与收发缓冲区。开发板在此基础上挂接了LAN8720A以太网芯片，能够连接网线并支持链路层的网络通信。

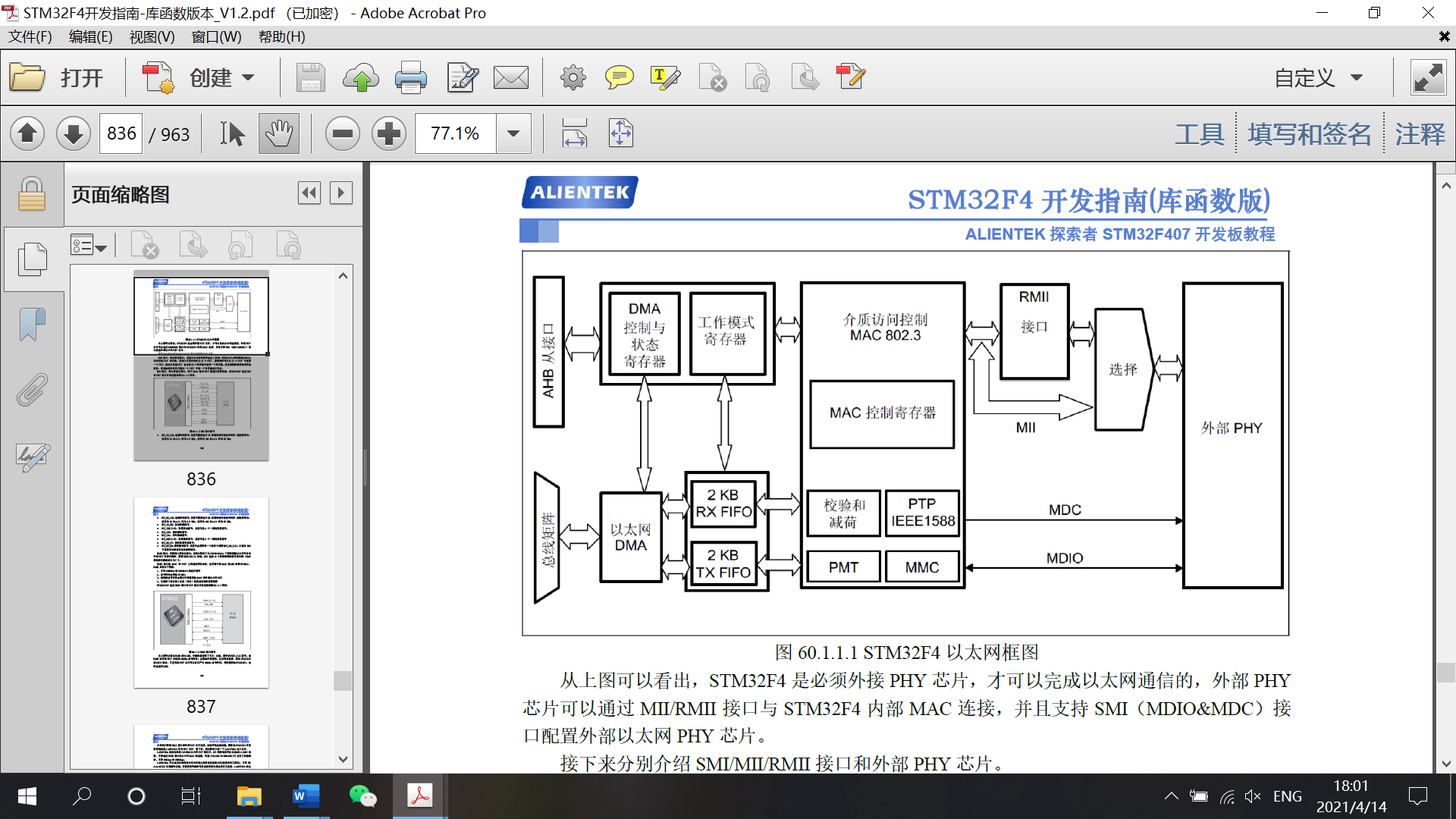


图3.2 STM32F4以太网口

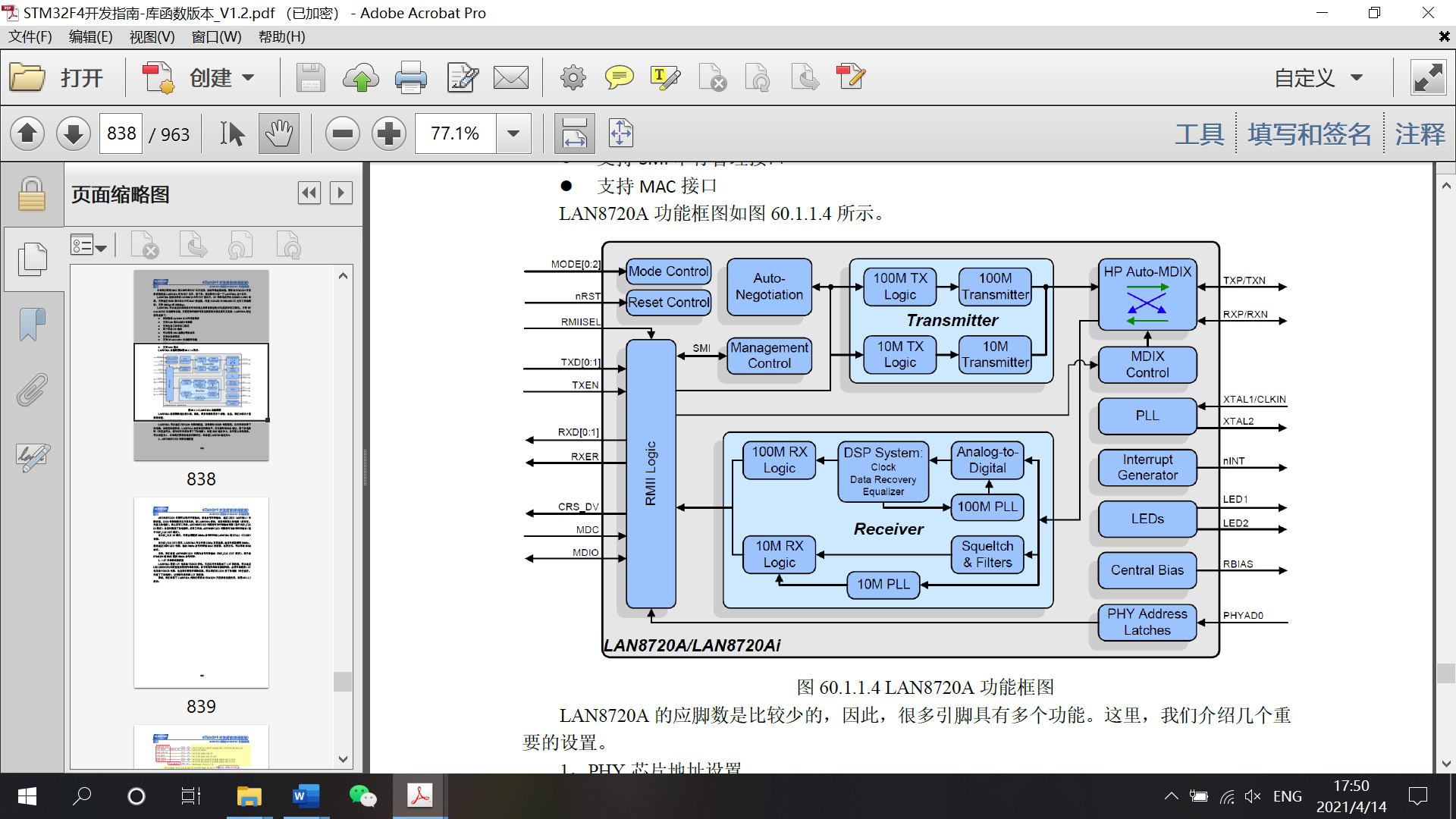


图3.3 LAN8720A芯片

3.1.2 摄像头

插图

图3.4 OV2640摄像头模块

OV2640摄像头模块是OV公司生产的CMOS UXGA图像传感器，通过串行摄像头控制总线（SCCB）连接，最高帧率可达15帧，支持自动曝光控制、自动增益与自动白平衡等功能，并可进行JPEG格式输出。

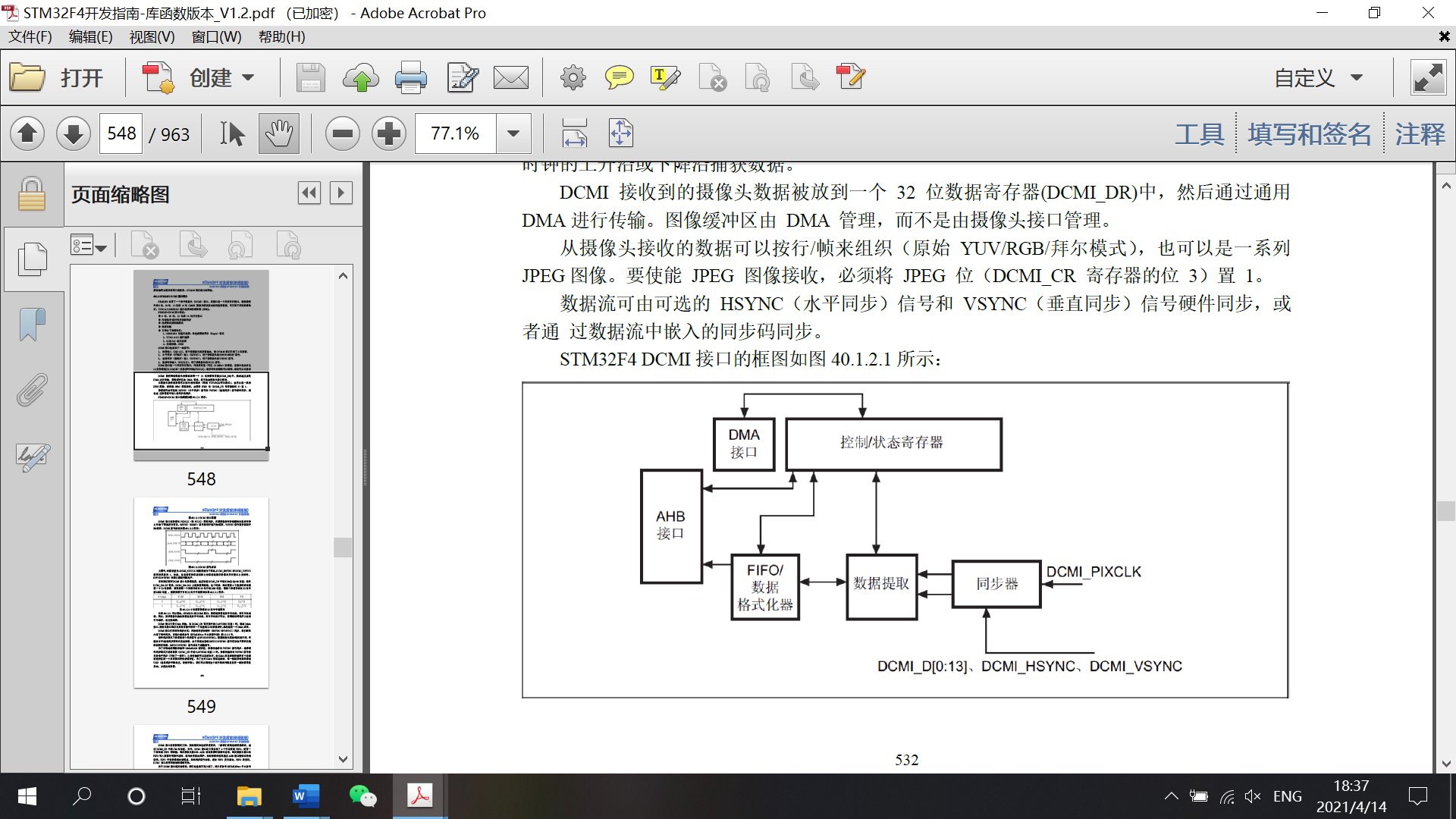


图3.5 STM32F4 DCMI

在连接方面，STM32F4芯片自带数字摄像头接口（DCMI），支持8-14位的CMOS数字摄像头控制与数据接收，并由DMA负责FIFO管理与数据搬运。正点原子公司在此基础上进行了开发板硬件设计，提供了一个符合SCCB规范的高性能8位并行OV2640插口并提供了硬件驱动程序。开发板摄像头接口各引脚如下表所示。

表3.3 开发板摄像头接口引脚

|  |  |  |  |
| --- | --- | --- | --- |
| 信号 | 作用 | 信号 | 作用 |
| VCC | 电源 | OV\_PCLK | 像素时钟输出 |
| GND | 接地 | OV\_PWDN | 掉电使能信号 |
| OV\_SCL | SCCB时钟信号 | OV\_VSYNC | 帧同步信号 |
| OV\_SDA | SCCB数据信号 | OV\_HREF | 行同步信号 |
| OV\_D[0:7] | 8位数据输出 | OV\_RESET | 复位信号 |

3.1.3 温湿度传感器

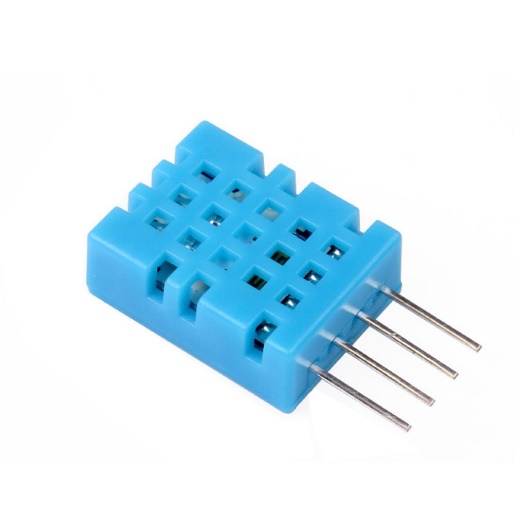


图3.6 DHT11温湿度传感器模块

DHT11是一款温湿度一体化数字传感器，包括一个电阻式测湿元件和一个 NTC测温元件，并与一个高性能 8 位单片机相连接，与开发板之间采用简单的单总线协议进行通信。该协议下传感器的单个完整数据包由40 bit组成，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校验和。该模块的通信协议与开发过程相对较为简单。

3.2 硬件连接

插图

图3.7 嵌入式设备硬件连接图

给出MCU各IO引脚使用情况如下表所示。

表3.4 STM32F407 IO引脚使用情况

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 引脚 | 功能 | 引脚 | 功能 | 引脚 | 功能 |
| PA0 | KEY\_WKUP | PA13 | SWDIO | PE2-4 | KEY2-0 |
| PA1 | RMII\_REF\_CLK | PA14 | SWDCLK | PE5-6 | DCMI6-7 |
| PA2 | ETH\_MDIO | PB6 | DCMI\_D5 | PF9-10 | LED0-1 |
| PA4 | DCMI\_HREF | PB7 | DCMI\_VSYNC | PG9 | DCMI\_PWDN |
| PA6 | DCMI\_PCLK | PC1 | ETH\_MDC |  | 1WIRE\_DQ |
| PA7 | RMII\_CRS\_DV | PC4 | RMII\_RXD0 | PG13-14 | RMII\_TXD0-1 |
| PA8 | DCMI\_XCLK | PC5 | RMII\_RSD1 | PG15 | DCMI\_RESET |
| PA9 | USART1\_TX | PC6-PC9 | DCMI\_D0-3 |  |  |
| PA10 | USART1\_Rx | PC12 | DCMI\_D4 |  |  |

4 软件设计

4.1 嵌入式设备开发

在本设计中，嵌入式设备端负责采集传感器数据并通过Socket上传至服务器。开发过程中的主要任务包括μC/OS-II操作系统与lwIP协议栈移植、网络通信功能的实现以及传感器驱动与数据捕获等。

嵌入式设备软件部分的功能使用Keil μVision5进行开发。这是美国Keil Software公司出品的单片机C语言软件开发系统，提供了包括C编译器、宏汇编、链接器、库管理和一个功能强大的仿真调试器在内的完整开发方案，通过一个集成开发环境（μVision）将这些部分组合在一起，是一款经典的嵌入式C语言开发工具。



图4.1 Keil μVision5

4.1.1 μC/OS-II环境配置

本设计选用的嵌入式操作系统μC/OS-II是一个可以基于ROM运行的、可裁剪的、抢占式的实时多任务内核，除一个与CPU硬件直接相关的汇编文件之外其余代码全部采用ANSI C编写，具有高度的可移植性。

作为系统硬件层与应用程序软件的接口，μC/OS-II源码主要由以下三个部分组成。

首先是CPU硬件相关的部分。该部分由一个汇编文件（os\_cpu\_a.asm，负责任务切换的具体实现）、一个源文件（os\_cpu.c，负责用户Hook函数的定义）以及一个头文件（os\_cpu.h，负责数据类型的定义）组成，也是操作系统移植时唯一需要修改的部分。同时，内核依靠硬件层提供的Systick定时器提供时钟节拍

其次是硬件无关的部分。该部分定义了操作系统的核心功能，包括系统内核、标志位、消息邮箱、内存分配、互斥锁、消息队列、信号量、任务管理、系统时钟与软件定时器等。该部分代码全部采用C语言编写，在移植时不需要进行任何修改。

最后是配置文件部分。该部分由两个头文件组成，其中os\_config.h中通过宏定义结合条件编译机制实现了系统功能裁剪，保证只有在配置文件中被用户选择的特定功能才会被最终编译并烧录，有助于减小系统内核体积；而includes.h负责向工程中的其他文件提供操作系统相关的变量与接口函数声明。

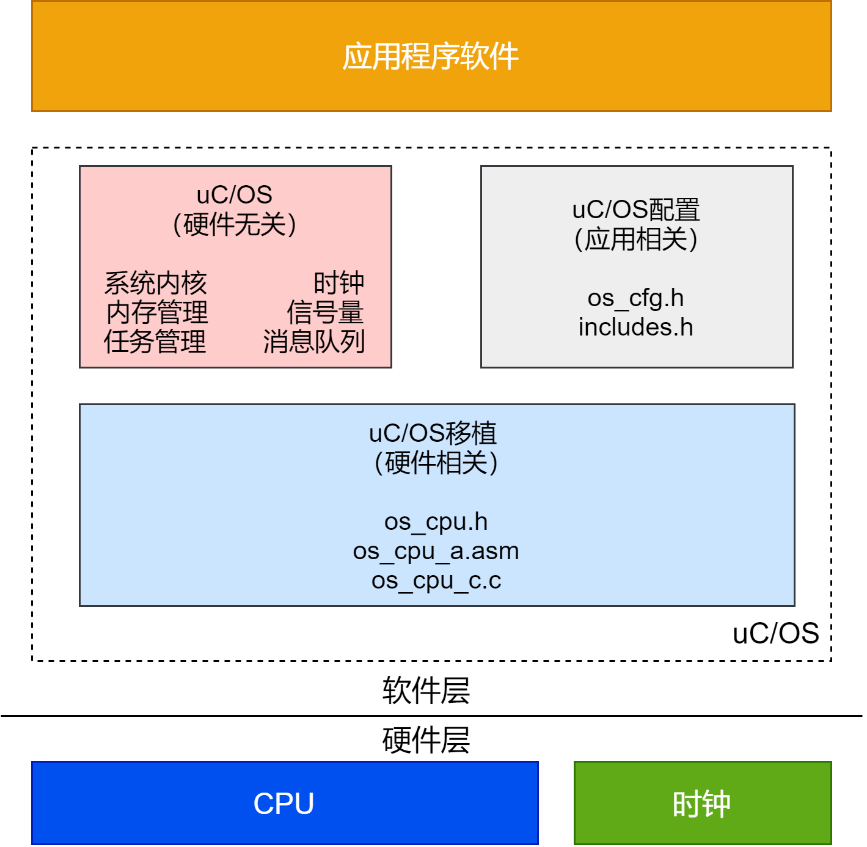


图4.2 μC/OS-II内核架构

由于μC/OS-II最多支持64个任务优先级且不允许优先级重复，同时系统内优先级最高的4个任务与优先级最低的4个任务均由内核占用（如软件定时器、统计任务与空闲任务等），因而系统最多可以支持56个用户自定义任务，对于小型的嵌入式系统是完全足够的。本设计共包含3个系统任务与10个用户自定义任务，如下表所示。

表4.1 嵌入式设备系统线程表

|  |  |  |
| --- | --- | --- |
| 优先级 | 名称 | 功能 |
| 3 | TimerTask() | μC/OS软件定时器任务 |
| 5 | tcpip\_thread() | lwIP协议栈内核线程 |
| 7 | dhcp\_thread() | DCHP线程 |
| 10 | send\_task() | Socket发送线程 |
| 11 | recv\_task() | Socket接收线程 |
| 12 | http\_get\_task() | HTTP请求处理线程 |
| 13 | app\_main\_task() | 网络功能主线程 |
| 14 | camera\_task() | 摄像头线程 |
| 18 | key\_task() | 按键响应线程 |
| 19 | led\_task() | LED线程 |
| 20 | start\_task() | 系统起始任务 |
| 62 | StatTask() | μC/OS统计任务 |
| 63 | IdleTask() | μC/OS空闲任务 |

4.1.2 lwIP协议栈移植

为基于开发板以太网口实现对于应用层网络通信功能的支持，本设计移植了lwIP-1.4.1，并加入对操作系统的支持以提供更好的多线程性能。

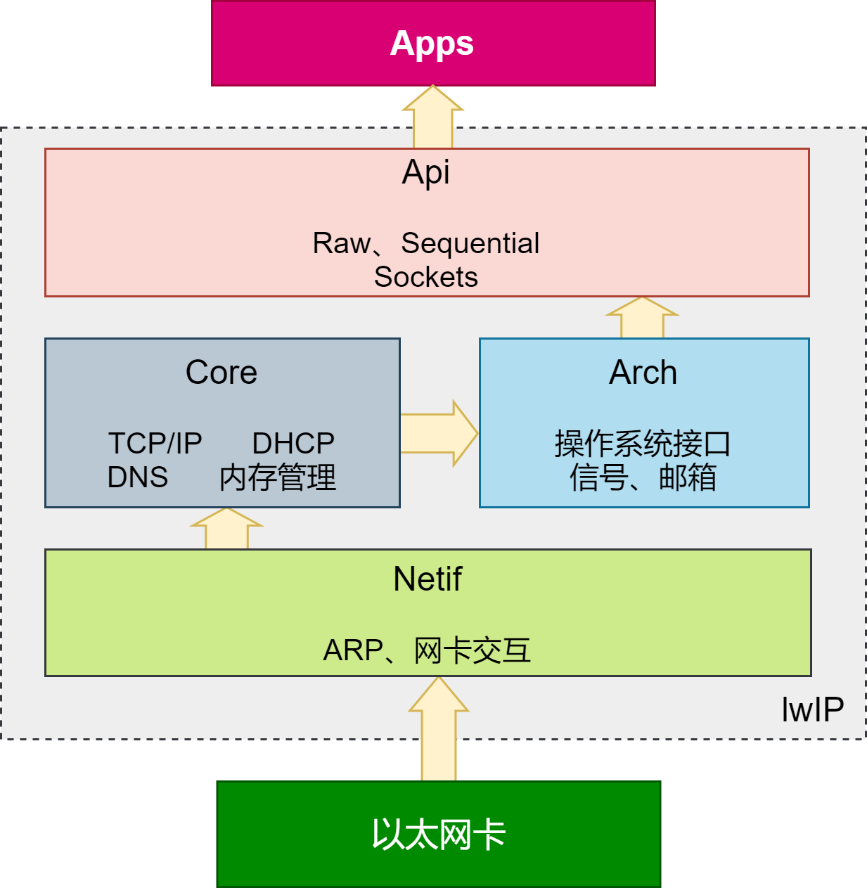


图4.3 lwIP协议栈架构

该协议栈主要工作于网络层与传输层，其源码部分由网络接口层、内核层、系统架构层与Api层组成，并向上提供了Raw、Sequential与Socket三种Api。

在以太网口接收到数据之后，网络接口层负责进行协议栈与以太网卡间的数据交互并实现ARP协议，之后将数据包上交内核层进行处理。内核层集成了TCP/IP协议族，同时也负责实现DHCP、DNS以及处理网络数据时的内存管理。在支持操作系统的lwIP版本中，考虑到应用层的网络通信任务可能是多线程的，因此为了实现更好的并发支持，协议栈不允许应用软件直接通过接口函数与内核层通信，而是在内核层与应用层之间设计了系统架构层，并在其中对操作系统提供的信号量与消息邮箱进行了封装，从而借助内核任务调度与通信机制来支持多线程。

插图

图4.4 lwIP协议栈内部工作流程

协议栈在初始化时会向操作系统注册一个优先级很高的tcpip\_thread作为网络功能的内核线程，该线程负责实现包括TCP/IP在内的所有协议栈内核层功能。当应用层软件发起网络通信时，将通过协议栈提供的Api调用系统架构层接口，使用信号量与消息邮箱将数据包下发给内核线程。之后，内核线程将依次处理每个通信任务，并同样通过信号量与消息邮箱向应用层线程返回数据。这样的架构设计保证了对于并发的支持并解决了协议栈的线程安全问题，同时也避免了应用软件层出现意外阻塞而影响系统正常的任务调度。

4.1.3 HTTP与Socket网络通信

由于协议栈没有实现对于HTTP协议的直接支持，本设计基于Sequential Api通过TCP协议实现了HTTP请求的发起与响应数据的解析，并注了册独立的HTTP GET请求处理任务http\_get\_task()，利用信号量机制向其他线程提供接口。

网络应用在初始化时会向操作系统注册HTTP线程并与服务器建立TCP连接。在设备校验阶段，网络功能主线程负责在全局发送缓冲区内构建HTTP报文并触发请求信号，之后挂起等待响应信号。HTTP线程在收到请求信号后通过TCP连接将报文发送至服务器并等待服务器响应或进行超时重传。收到响应数据后，HTTP线程将数据存入全局接收缓冲区并触发响应信号。主线程将监听到该信号，并在恢复运行后解析响应数据。在本设计中，若服务器返回的响应报文数据段为“OK”则代表设备校验成功，而经过校验的合法设备将被允许与服务器建立Socket连接。

而对于Socket通信，网络应用在初始化时基于协议栈提供的BSD Socket Api向操作系统分别注册了独立的Socket读写线程以负责相关网络数据的处理。其中，发送任务send\_task()负责将摄像头缓冲区内的数据以及温湿度传感器数据上报服务器；接收任务recv\_task()负责监听来自服务器的用户控制指令。

在各个功能线程的支持下，本设计编写了网络任务主线程负责设备不同运行阶段的控制，包括引用环境初始化、缓冲区内存管理、向服务器发起设备校验、创建Socket连接并响应用户指令等。

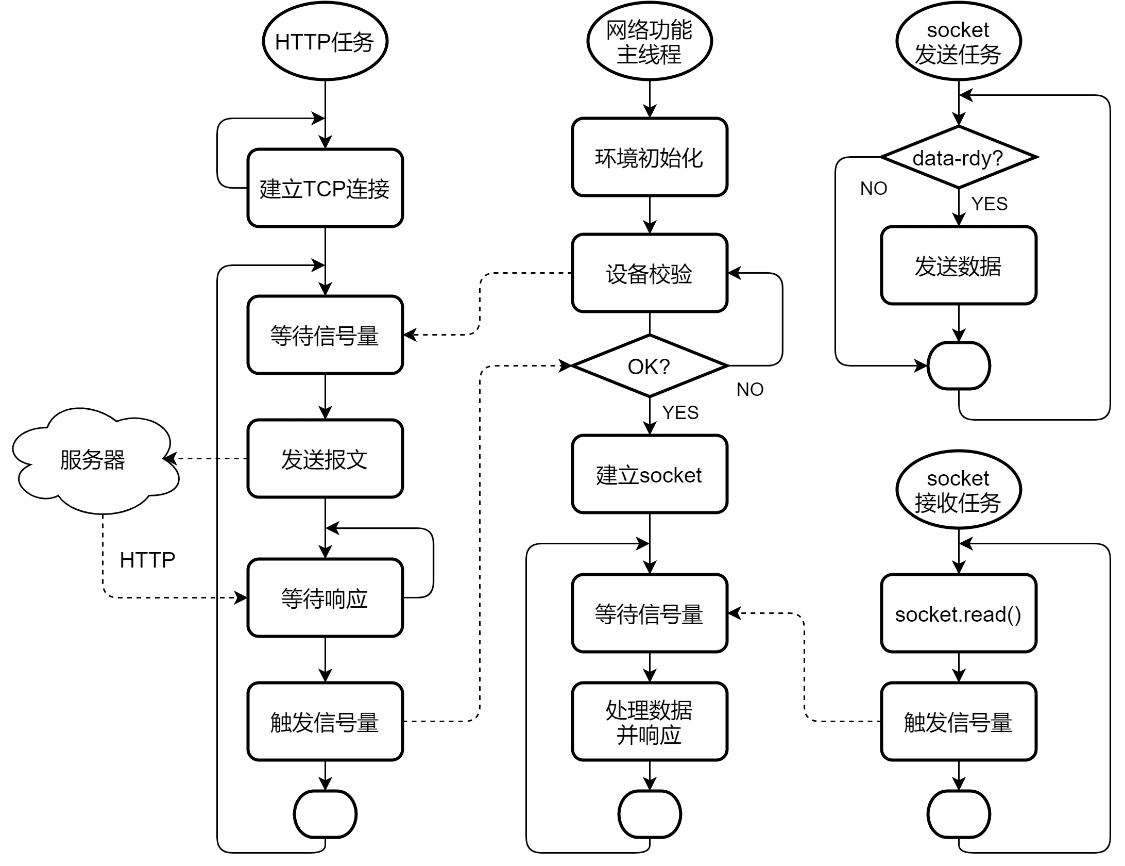


图4.5 嵌入式设备网络通信流程

4.1.4 传感器数据捕获

本设计中，摄像头模块通过DCMI与处理器连接，并由DMA控制器负责进行数据搬运与缓冲区管理。

为配合网络传输的工作并减轻服务器端的数据处理负担，系统在初始化时将摄像头模块配置为JPEG格式输出并采用帧中断。当DCMI接收到一帧完整的图像后，DMA控制器会将数据块搬运至内存并触发DCMI中断。系统在进入该中断后会首先关闭DMA，等待数据传输完成并将数据块标记为待处理状态。直到本次数据被处理完成之前，内存中的图像缓冲区将不再会进行刷新。

在摄像头任务camera\_task()中，当程序检测到数据处于待处理状态时，会将数据标记发送并通知Socket进行网络传输，完成后将标志位归零。在下一次DCMI中断时，系统检测到上一次的数据已经处理完成后会清除缓存并重新使能DMA以进行下一次的数据传输。

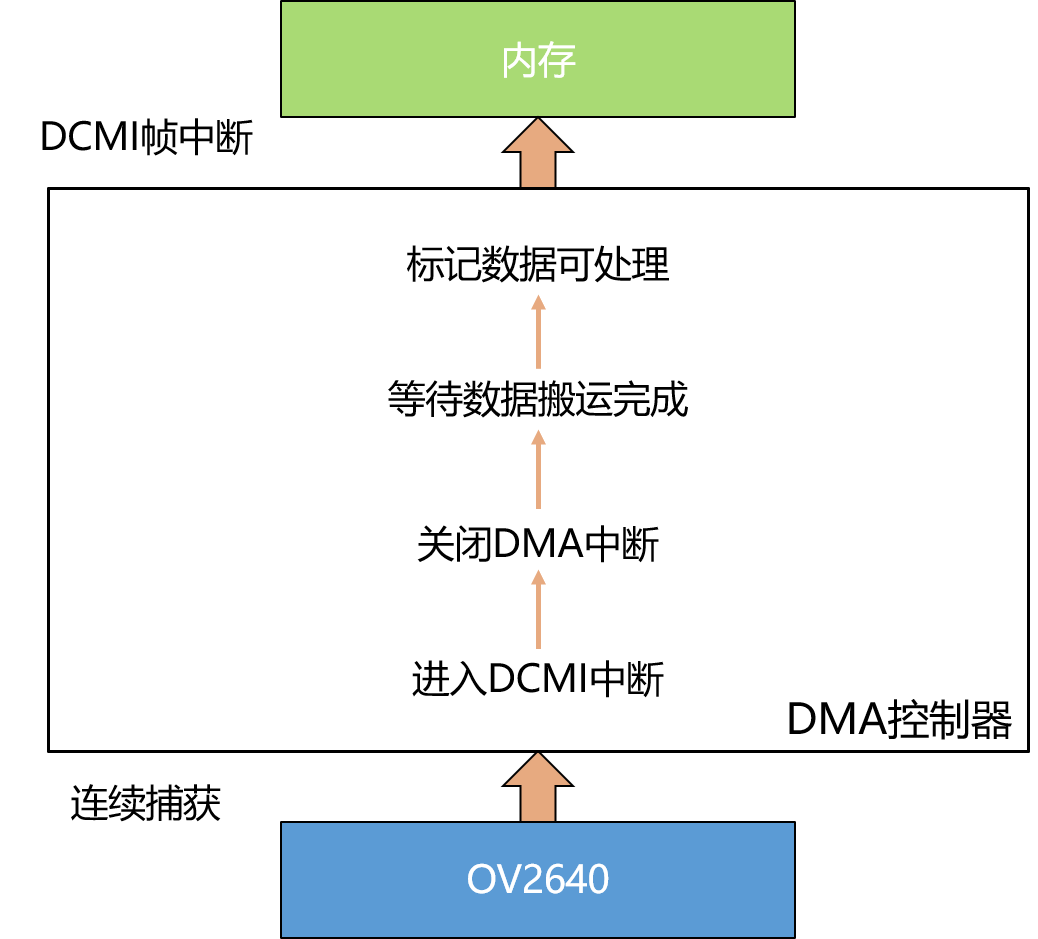


图4.6 DCMI工作流程

对于温湿度传感器数据，由于本设计使用的开发板所提供的单总线接口数据线引脚1WIRE\_DQ与摄像头接口的掉电使能引脚PV\_PWDN重合（均使用了PG9），设备运行过程中的对于温湿度传感器的读取操作将导致摄像头模块掉电关闭，因而只能对两者进行分时复用。由于摄像头在数据传输未完成时掉电会使设备与摄像头失去同步，并导致DCMI与DMA控制器进入未知状态，因而对与温湿度传感器的读取必须等待本次摄像头数据全部处理完成且下一次传输还未开始时进行，且在读取完成后需要对摄像头模块与DCMI全部进行重新初始化。

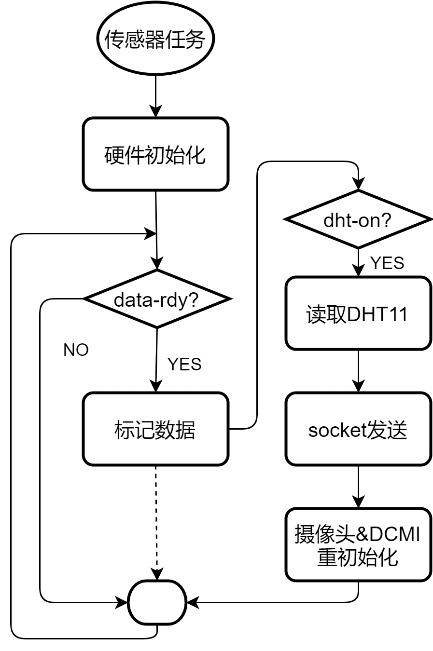


图4.7 摄像头与温湿度传感器工作流程

尽管对于温湿度传感器的读取将打断摄像头数据的正常传输，但由于实际应用情形中用户对于温湿度数据的请求频率远远低于对摄像头数据的请求频率，因此这种暂时的打断在一般情况下是可以接受的。

4.2 Web服务器开发

本设计的服务器部分由一个负责支持HTTP请求的Web服务器和一个Socket服务器两部分组成。其中，本节介绍的Web服务器主要面向微信小程序提供登录注册以及获取设备数据的接口，同时也面向嵌入式设备提供校验服务。

服务器端的主要功能全部通过Python编写并使用VSCode作为开发环境。该软件是微软出的一款功能强大的轻量级开源代码编辑器，支持几乎所有主流程序语言的语法高亮、智能代码补全、自定义热键、括号匹配、代码片段、代码对比并支持插件扩展，同时针对网页开发和云端应用开发也做了相应的优化。



图4.8 VS Code

本设计中Web服务器的相关功能基于Flask框架实现。这是一个使用Python编写的轻量级WSGI应用框架，底层由Werkzeug套件与Jinja2引擎支持，在简单易用的同时也具有高度的可定制性与可扩展性。Flask框架处理 HTTP请求的基本方式为将一个视图函数分配给一个URL，在用户访问URL时Flask将调用该函数处理请求，并将其返回值经渲染后作为响应返回给用户。

插图

图4.9 Flask框架工作流程

Flask框架内部集成了包括对于Cookie与会话管理等常见Web应用功能的支持，在本设计中用于进行用户身份验证与跨请求的数据管理。为确保灵活性，框架一般不指定数据库与模板引擎对象，相关功能可由用户根据需求自行配置。本设计在数据库方面选用SQLite3，有关数据库接口的具体内容见4.2.2章节。另外，由于客户端模板文件（WXML）的定义与渲染已经包含在微信小程序框架之内，服务器面对请求时只需要返回业务数据而不需要生成整个模板文档，这也在一定程度上减轻了服务器端的开发负担。

4.2.1 设计模式

为方便项目测试以及发布运行时的托管，本设计中的Flask采用工厂模式，并通过Python包对应用进行管理。

插图

图4.10 服务器项目文件夹目录结构

在项目文件夹下应用目录内的\_\_init\_\_.py文件中编写了工厂函数create\_app()，负责创建一个Flask应用实例并对其进行配置。该函数首先设置应用密钥、项目位置、数据库路径以及客户端微信小程序ID与密钥等全局信息，之后通过添加蓝图（blueprint）的方式注册项目中所有用户编写的视图函数并生成Socket服务器实例（具体内容见4.3章节），最后返回一个Flask应用实例以供其它托管程序调用。

在部署服务器并运行时，Flask提供的开发服务器或其他WSGI托管服务器将导入该Python包，并通过工厂函数获取一个应用实例加以运行。

4.2.2 用户管理

本设计中用户管理的相关功能主要包含用户登录与设备绑定两部分。

设计采用SQLite3数据库进行用户与设备信息的存储。这是一个遵循ACID规范的轻型关系数据库系统，其基于文件系统而不需要专门的服务器支持。需要访问数据库的视图函数通过调用db.py文件提供的接口get\_db()来进行数据库连接，并获取一个可操作的Row对象，之后通过该对象进行具体的查询操作。

数据库中所存储的信息主要包括两个表。其一是设备表Devices，包括由开发者登记并信任的设备ID、密钥、型号与生产日期信息；其二是运行时创建的用户表Users，包括已注册用户的openid、注册日期与绑定设备信息。

插图

图4.11 SQLite3 E-R图

由于服务器将openid作为用户身份的唯一识别依据，当客户端微信小程序投入运行时会首先向服务器发起登录请求并上报临时登录凭证code，负责管理登录功能的视图函数会使用该凭证配合小程序ID及密钥向微信接口服务器换取该用户的openid，之后向数据库Users表查询该用户绑定的设备情况并将结果存入session，最后向用户返回携带有cookie信息的“OK”以表示登陆成功。

对于首次登陆的用户来说，其openid此时并不存在于数据库之中，则此时登录视图函数将返回“REG”提示客户端进行注册与设备绑定。具体方式为客户端通过POST请求向服务器提交待绑定设备的ID与密钥，之后由负责管理注册功能的视图函数查询Devices表并进行校验，若正确无误则将当前用户的openid与设备信息作为一条新数据插入Users表，之后向客户端返回“OK”表示设备绑定成功。

插图

图4.12 服务器用户管理流程

4.2.3 设备校验

本设计中，该Api的主要功能为面向嵌入式终端提供设备合法性校验接口并在服务器上创建Socket监听线程。该视图函数从来自设备的HTTP请求中取得设备ID与密钥，并向数据库Devices表发起查询并进行校验。若校验通过，服务器将记录当前请求的IP作为该设备的合法IP地址，并将设备IP到ID的映射存入Socket服务器合法设备表，之后调用Socket服务器提供的接口函数创建监听线程，并通知Socket服务器允许接受来自该IP地址的Socket连接请求。

插图

图4.13 服务器设备校验流程

有关Socket服务器接口的具体实现见4.3.1章节。

4.2.4 数据中转

为了将来自嵌入式设备的数据正确返回给绑定了该设备的客户端用户，该部分功能的实现借助了用户身份管理系统以及Socket服务器数据缓存，其具体方式如下。

对于已经在服务器成功登录的用户，Flask会自动解析来自该用户的请求内所携带的cookie并在上下文中还原客户端登录时保存在session里的用户信息。在此基础上，用户管理系统向Flask注册并挂接了load\_logged\_in\_user()函数，该函数将在所有视图处理具体的客户端请求前被Flask调用，其功能为获取session中保存的用户openid，并向数据库Users表查询该用户绑定的设备信息，之后将该信息存入Flask提供的全局上下文对象g。

当客户端向服务器请求获得一份设备数据时，Flask会首先检查全局上下文对象g中是否包含与该用户对应的有效设备信息，若包含则继续调用视图函数处理请求，否则返回登陆提示。在视图函数中，服务器会根据上下文对象中当前用户绑定的设备信息，从Socket服务器的数据缓存取得对应设备的数据并将其返回客户端。有关Socket服务器数据缓存的内容见4.3.3章节。

插图

图4.14 服务器数据中转流程

4.3 Socket服务器开发

Socket服务器主要负责接收嵌入式设备通过Socket上传的监测数据，在服务器内部缓存并将其提供给Flask。本设计通过一个SocketServer类管理服务器端Socket通信的所有相关工作，并向Flask中的视图函数提供接口。

插图

图4.15 SocketServer UML图

出于简洁性考虑， Socket服务器本身并不会注册独立的线程，其工作方式为在服务器初始化阶段生成一个全局唯一的SocketServer实例，当Flask需要进行Socket创建或访问某一设备的数据时，视图函数将通过接口调用该实例提供的相关方法来获取对应的服务。

4.3.1 Socket创建

由于本设计中只有通过校验的合法设备才能够被允许连接至Socket服务器，而Socket本身在接受连接时是不加限制的，因此需要采用一种主动性的设备校验机制。

SocketServer在实例化阶段进行Socket配置与参数定义，并创建合法设备表devices以及监听线程列表threads。当负责设备校验的视图函数接收到一个来自设备的校验请求并校验通过后，视图函数将调用create\_socket()接口。该函数接受合法设备的ID与IP地址作为参数，将映射关系以字典形式保存在devices属性中，之后调用SocketServer提供的方法注册一个监听线程，在该线程中接受Socket连接并检查合法性。

由于服务器无法实时获知设备端Socket的连接状态，且服务器上已经启动的监听线程在非异常状态下不会自动退出，当设备意外重启或出于其他原因需要重新与服务器建立Socket连接时，服务器内部可能仍然维持着上一次的连接状态而拒绝设备新发起的连接请求。为提高系统鲁棒性，设计在create\_socket()方法中加入了如下机制。

插图

图4.16 服务器Socket创建流程

当设备首次与服务器建立连接时，服务器生成该设备的监听线程并将线程对象存入threads属性。在设备意外重启并重新向服务器发起设备校验与Socket连接请求时，create\_socket()函数发现与当前设备IP地址匹配的Socket监听线程已经处于运行状态，此时服务器将通知并等待该线程关闭，之后重新注册线程并等待新的连接。有关该机制的具体实现见4.3.2章节。

4.3.2 监听线程

在本设计中，服务器与设备创建Socket连接的本质即在服务器端注册一个新的Socket监听线程并使其投入运行。为了实现设备数据在服务器端的缓存以及外部函数对线程运行状态的控制，本设计自行编写了SocketThread类。该类继承自Python标准库中threading模块提供的Thread对象，除了具有普通线程对象应具备的基本属性与方法之外，SocketThread类增添了设备数据缓冲区属性buffers与运行状态标志位属性running，同时提供了访问缓冲区数据的接口方法。

插图

图4.17 SocketThread UML图

与普通的线程对象类似，该类在实例化时接受一个target参数作为线程函数句柄。该函数在运行时会首先接受一个Socket连接并校验该连接的合法性，其具体方法为检查该连接的远端IP地址是否存在于SocketServer的devices表中。若校验成功从该表中则取得与IP地址对应的设备ID，并使用其从threads列表中查询得到自身所属的线程对象以便访问缓冲区，若失败则直接关闭连接。

当一个合法的连接被成功创建后，线程函数将循环读取Socket接收到的设备数据并将其存入该线程对象的buffer以供Flask读取。在此过程中函数不断检查线程对象的running标志位，若为False则表示服务器已经通知该线程终止。此时函数将跳出循环，关闭Socket并释放资源，之后线程便可安全退出。

插图

图4.18 服务器Socket监听流程

4.3.3 数据缓存

为方便Flask视图函数访问Socket服务器接收到的设备数据，本设计为SocketThread对象添加了buffer属性，该属性使用具名元组（namedtuple）存储一个包含温湿度数据的JSON对象以及一个图像数据缓冲队列。其中图像数据缓冲队列使用Python标准库queue中的Queue模块实现，这是一个基于Python标准库双端队列deque的线程安全的FIFO对象，提供了互斥锁以及多种阻塞与非阻塞式的访问方法。

当Socket服务器监听线程试图将一帧JPEG数据存入缓冲区时，监听线程函数使用线程对象提供的push\_img()方法，调用Queue模块的非阻塞式入队接口函数put\_nowait()。若此时缓冲队列已满，该方法会抛出Full异常，push\_img()函数将捕获该异常并直接将该帧图像丢弃，之后等待下一次插入操作。

类似地，当Flask试图访问并获得缓冲区中的数据时，视图函数使用线程对象提供的get\_img()方法，调用Queue模块的阻塞式出队接口函数get()。该方法将在队列为空时将线程阻塞，之后挂起并等待新数据入队。

插图

图4.19 服务器数据缓存流程

值得一提的是，由于Flask多作为一种基于WSGI的应用，其处理请求的方式为单线程还是多线程并不取决于Flask本身，而是取决于托管Flask应用的WSGI服务器具体工作于何种模式之下。因此，在采用阻塞式get()方法进行数据获取时，应首先保证Flask应用服务于多线程模式以避免调用该方法的视图函数阻塞来自其他设备的请求处理。

4.4 交互应用程序开发

本设计使用微信小程序作为客户端的人机交互方案，主要负责提供基于微信平台身份系统的用户登录操作并对来自设备的监测数据进行显示。

该部分的相关功能使用微信开发者工具进行开发。这是微信官方提供的专门针对微信小程序的开发环境，集成了代码编辑、开发调试及程序发布等功能，能够帮助开发者简单和高效地开发微信小程序或小游戏。



图4.20 微信开发者工具

微信小程序的代码部分主要由四种文件构成，分别为JSON配置（负责配置页面路径，窗口参数，工程环境等）、WXML模板（负责定义页面结构，类似HTML）、WXSS样式（负责定义页面样式，类似CSS）以及JS逻辑交互（负责响应用户操作）构成。

插图

图4.21 小程序工程目录结构

相比于普通的网页前端应用，微信小程序工作在微信客户端提供的宿主环境内，分为渲染层与逻辑层两部分。当小程序启动时，微信客户端将整个小程序代码包下载到本地，根据全局配置文件中的页面信息注册一个页面实例Page，并根据其中的JSON配置生成界面，之后装载WXML模板与WXSS样式，最后加载JS进行页面渲染。

在小程序中，渲染层和逻辑层分别由两个独立的线程管理。渲染层的界面使用了WebView 进行渲染；逻辑层采用JsCore线程运行JS脚本。一个小程序存在多个界面，所以渲染层存在多个WebView线程，WebView与JsCore线程的通信会经由微信客户端做中转，逻辑层发送网络请求也经由微信应用进行转发。

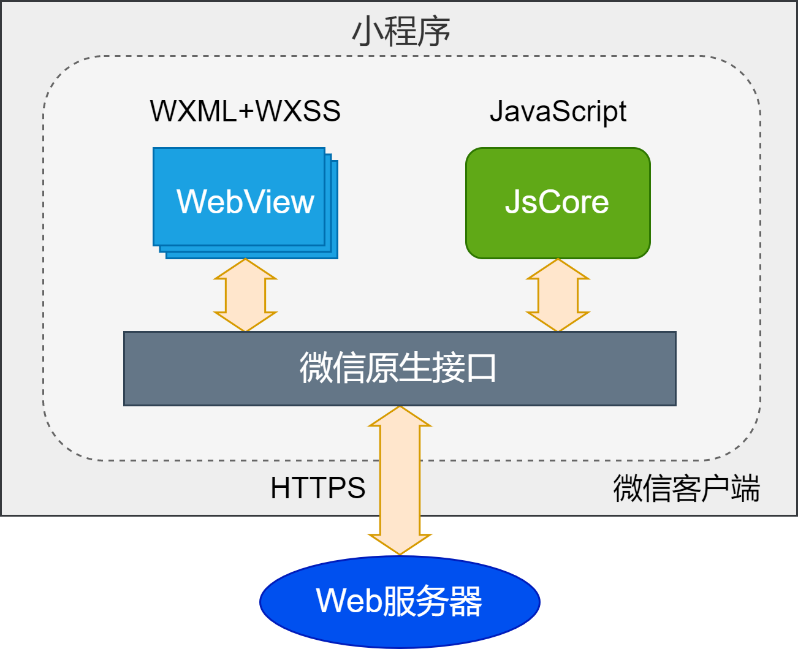


图4.21 小程序通信模型

在微信平台的依托下，开发者可以通过JS方便的调用Api进行事件监听、页面切换、发起网络请求与播放媒体等，或使用微信客户端与操作系统提供的登录、支付、转发、地图定位以及生物验证接口。在微信小程序环境中，除以“Sync”字段结尾的个别Api为同步接口之外，其它大部分接口功能采用异步方式实现。本设计中用到的部分Api如下表所示。

表4.2 小程序部分Api列表

|  |  |
| --- | --- |
| 名称 | 功能 |
| setTimeout/setInterval | 设置一个定时器并在到期后/按周期执行回调函数 |
| clearTimeout/clearInterval | 清除一个定时器 |
| wx.redirectTo | 关闭当前页面并跳转到其他页面 |
| wx.navigateTo | 保留当前页面并跳转到其他页面 |
| wx.request | 发起HTTPS网络请求 |
| wx.downloadFile | 下载文件资源到本地并暂存 |
| wx.getStorageSync | 将数据键值对以同步方式存储在本地缓存中 |
| wx.login | 调用接口获取临时登录凭证code |
| wx.getUserInfo | 获取用户信息 |
| wx.authorize | 向用户就某项功能发起授权请求 |

4.4.1 用户登录

由于小程序所有功能的实现均依赖于用户登录信息，因此用户登录的相关功能注册在app.js文件里全局应用实例App的onLaunch()方法中。该方法在小程序运行环境初始化完成时触发且全局只触发一次。

小程序会首先调用wx.login()方法获取一个临时登录凭证code，之后使用wx.request()将该凭证通过HTTP请求发送至服务器。对于一个已经完成设备绑定的用户，服务器在成功向微信后台换取用户openid并完成登录操作后，会向小程序返回带有cookie的“OK”以表示登录成功；而对于一个新用户，服务器会返回“REG”以提示小程序进行设备绑定。

插图

图4.22 小程序登录流程

由于小程序并不是一个真正意义上的HTTP客户端，其所有网络请求的发起均需要通过微信后台中转，因而小程序没有内建的cookie机制。则若想在服务器端保持session需要开发者在每个请求的headers字段手动设置cookie值。本设计中，一个小程序的cookie在登录完成后会被从响应头中的cookies[]数组里解析出来，并保存在App实例下globalData对象的cookie字段以方便所有页面的请求进行调用。

4.4.2 设备绑定

当登录请求返回的响应数据段为“REG”时，表示服务器未在数据库Users表中查询到该用户的设备信息，需要用户进行设备绑定。此时，小程序在完成加载后将自动跳转至设备绑定页面，用户可以在当前页面中输入欲绑定设备的ID和密钥，之后由小程序通过POST请求将表单提交至服务器进行验证。完成绑定的用户将收到来自服务器的“OK”响应，之后重新调用App实例的login()方法进行登录，完成后将页面重定向至应用首页。

插图

图4.23 小程序设备绑定流程

4.4.3 请求数据

在用户登录和设备绑定功能的基础上，微信小程序可以向服务器请求由已绑定设备上传并暂存在服务器的监测数据。

对于来自摄像头的JPEG流数据来说，由于图像以每秒15帧的频率刷新，而一个获取图像的HTTP请求可能需要更长的时间来完成响应并渲染。因此，若在程序中采用每次刷新一张图片的方式则远远不能达到设计所要求的帧率。

相应地，本设计在小程序中也同样加入了图像缓冲队列ImgQueue。该类采用循环队列实现，利用JS的单线程机制保证访问安全，对外提供入队push()和出队pop()方法。在小程序数据显示页面加载完成的回调方法onLoad()中注册有一个定时器对象imgInter，该对象将按照固定的时间间隔通过imgDownload()方法调用wx.downloadFile()接口从服务器请求一帧图片。该Api将从服务器图像缓冲区内下载一帧图片到本地并返回一个临时资源位置，之后在下载成功的回调函数中将该图片的临时资源位置存入缓冲队列末尾。接着，小程序从缓冲队列首端取出一帧图片的临时资源位置，并调用Page对象的setData()方法将视图层中负责图片显示的相应组件的src字段更新，随后由小程序底层JsCore线程通知WebView线程重新渲染页面上相应位置以完成显示。

由于wx.downloadFile()为异步接口，imgDownload()函数并不会在请求一张图片并等待响应的过程中发生阻塞，因此小程序可以在允许范围内向服务器发起并等待多个请求，同时不受干扰地将之前已经入队的图片资源取出后渲染至视图层。

插图

图4.24 小程序请求数据流程

本设计中温湿度数据的请求与显示原理与此类似，但由于其数据量很小且请求频率远低于图像数据，所以并不需要加入缓冲机制，因而其实现也相对简单。

5 项目测试

5.1 嵌入式设备测试

本节主要进行嵌入式设备端的单元测试。

作为整个系统的主要数据来源，嵌入式设备所承担的传感器捕获与网络传输任务是实现本设计功能的基础，同时也是应首先进行开发并测试的部分。相关测试环境如下表所示。

表5.1 嵌入式设备测试环境

|  |  |  |
| --- | --- | --- |
| 环境 | 型号/版本 | 说明 |
| 微控制器 | STM32F4 | 正点原子探索者开发板 |
| 电源 | GQ12-120100-CC | 12V直流稳压电源，用于设备供电 |
| 调试器 | ST-Link/V2 | 用于代码烧写与程序调试 |
| 路由器 |  | 通过以太网线连接开发板，提供网络通信 |
| 串口调试助手 | XCOM/V2.3 | 监听串口调试信息 |
| 网络调试助手 | XNET/V1.1 | 作为TCP服务器接收网络数据 |

5.1.1 操作系统测试

首先测试操作系统是否移植成功并能够正常进行任务调度。

根据CPU型号修改工程UCOSII/PORT目录下的相关文件，之后定义两个任务分别控制LED0与LED1。将程序编译无误后烧录至开发板，启动并观察运行效果。

插图

图5.1 操作系统测试结果

通过串口调试信息，可以看到操作系统成功初始化。此时观察到两个LED分别以不同频率闪烁。

测试表明操作系统移植成功且运行正常。

5.1.2 网络通信测试

接下来测试lwIP协议栈的移植情况以及网络通信功能。

根据操作系统环境改写协议栈架构层接口，按照4.1.3章节实现各功能线程并使用按键调用相关接口，配置服务器IP地址为PC端WLAN网卡地址并设置端口号为8088。将程序编译无误后烧录至开发板，将开发板通过以太网线连接至路由器，同时在PC端运行网络调试助手，启动并观察运行效果。

插图

图5.2 网络通信测试

通过串口信息以及网络调试助手窗口，可以看到协议栈成功初始化，能够通过DHCP获取设备IP地址与端口号。按下KEY0键，开发板通过TCP连接至网络调试助手并成功发送HTTP报文；按下KEY1键，开发板通过socket连接至网络调试助手并能够进行数据接收；按下KEY2键，开发板能够向网络调试助手进行数据发送。

测试表明系统HTTP与Socket网络通信功能运行正常。

5.1.3 传感器测试

最后测试摄像头与温湿度传感器是否能正常捕获数据并实现传输。

按照4.1.4章节完成代码编写并使用按键控制传输，将摄像头模块与温湿度传感器模块分别插入开发板的相应接口，将程序编译无误后烧录至开发板，启动并观察运行效果。

插图

图5.3 传感器测试

通过串口调试信息，可以看到各硬件传感器模块初始化成功，且由DCMI控制的LED1不断闪烁，表示摄像头数据正常捕获。按下KEY0键，开发板将一帧JPEG数据通过Socket发送至网络调试助手，在16进制下查看发现有效数据部分以“0xFF 0xD8”起始并终止于“0xFF 0xD9”，为合法JPEG数据格式；按下KEY1键，开发板停止摄像头捕获，初始化并读取温湿度传感器数据，通过Socket发送至网络调试助手，之后重新初始化并继续摄像头捕获。

在PC端读取并解析JPEG数据，可以得到一帧来自摄像头的图片。

插图

图5.4 摄像头拍摄效果

测试表明传感器捕获与数据传输功能运行正常。

5.2 服务器测试

本节主要进行服务器端的测试覆盖。

为了方便开发与调试，Flask框架提供了测试客户端test\_client()，可以在不借助托管应用真正启动服务器的条件下模拟向应用发送请求并返回响应数据。本设计基于pytest编写单元测试用例，该模块将自动识别并运行测试目录下全部以test\_开头的文件中所有以test\_开头的函数，并通过断言（assert）机制判定测试结果是否符合设计预期。相关测试环境如下表所示。

表5.2 服务器测试环境

|  |  |  |
| --- | --- | --- |
| 环境 | 型号/版本 | 说明 |
| python | 3.7.1 | 开发环境 |
| flask | 1.1.2 | Web服务器框架 |
| click | 7.1.2 | 命令行工具 |
| pillow | 7.2.0 | 图像处理库 |
| pytest | 6.2.2 | 测试框架 |
| pip | 21.0.1 | 包管理工具 |

5.2.1 基于pytest的测试环境配置

由于单元测试中每个测试用例之间都是相互独立的，pytest会通过固件（fixture）为测试用例配置运行环境并提供调用相关方法的接口，使用时只需要将测试所需要的固件作为函数参数传入即可。配置文件conftest.py定义了测试中可能用到的固件。

插图

图5.5 pytest测试固件UML图

文件首先注册了应用固件app。固件函数创建了用于测试的临时文件夹，之后调用create\_app()工厂生成一个应用实例并初始化数据库，最后将该实例作为一个固件返回给pytest。随后，文件又使用app实例提供的test\_client()以及test\_cli\_runner()方法分别向pytest注册了测试客户端client以及测试命令行工具runner以供测试用例调用。

由于本设计中所有功能的实现均依赖于用户管理系统，因此不妨定义一个用户对象AuthActions来模拟用户行为，将client固件添加为该类的一个属性，并在其方法中执行与登录和注册相关的操作，之后生成一个该类的实例auth并将其注册为一个pytest固件。

类似地，还可以定义一个DeviceActions类用以模拟设备行为。该类同样包含一个client固件作为属性负责进行设备校验，另外和还包含一个socket属性负责连接Socket服务器并进行数据交互。之后文件同样生一个该类的实例device并将其注册为一个pytest固件。

5.2.2 编写测试用例

基于配置文件中注册的各个固件，可以编写单元测试用例对应用工厂、数据库、用户管理系统、设备校验系统以及数据中转系统进行测试。其中部分用例如下表所示。

表5.3 服务器单元测试部分用例

|  |  |  |
| --- | --- | --- |
| 模块 | 用例 | 内容 |
| 应用工厂测试 | test\_factory | 测试应用工厂 |
| 数据库测试 | test\_get\_close\_db  test\_init\_db\_command | 测试数据库连接与关闭  测试数据库命令行指令 |
| 用户测试 | test\_login  test\_register | 测试用户登录  测试设备绑定 |
| 设备测试 | test\_verify  test\_socket  test\_uplaod | 测试设备校验  测试Socket连接  测试图像上传 |
| 数据测试 | test\_match  test\_buffer  … | 测试设备匹配  测试缓冲区  … |

5.2.3 测试结果

将测试配置到添加至项目根目录下的setup.cfg文件中，使用pytest运行测试并使用coverage生成测试覆盖率报告，得到结果如下表所示。

表5.4 服务器单元测试覆盖率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Stmt | Miss | Branch | BrPart | Cover |
| \_\_init\_\_.py |  |  |  |  |  |
| db.py |  |  |  |  |  |
| device.py |  |  |  |  |  |
| login.py |  |  |  |  |  |
| socket\_server.py |  |  |  |  |  |
| stream.py |  |  |  |  |  |
| TOTAL |  |  |  |  |  |

测试结果表明服务器工作各项功能运行正常。

5.3 交互应用程序测试

本节主要进行微信小程序的测试。

基于已经编写好并测试通过的服务器，可以对微信小程序中的相关功能展开测试。由于微信开发者工具已经集成了面向多种移动端设备的模拟器，因此在最初的调试阶段程序可以直接使用PC编译运行而不需要借助手机，可以等待开发完成后再使用手机设备查看具体效果。相关测试环境如下表所示。

表5.5 小程序测试环境

|  |  |  |
| --- | --- | --- |
| 环境 | 型号/版本 | 说明 |
| 微信开发者工具 | 1.05.2103200 | 小程序集成开发环境 |
| iOS版微信 | 8.0.4 | 真机调试环境 |
|  |  |  |

5.3.1 编写模拟设备客户端

为配合微信小程序对于数据请求部分功能的测试需要，设计使用Python编写了用于模拟嵌入式设备工作状态的HTTP与Socket客户端测试脚本。

插图

图5.5 模拟设备脚本工作流程

该脚本首先仿照设备启动时的工作方式，使用HTTP向服务器发起设备校验请求并等待响应。之后使用Socket连接至服务器并上传用于测试的JPEG图片与温湿度传感器数据等信息，用以测试时供小程序进行获取。

5.3.2 小程序功能测试

重启服务器，编译并启动小程序，授权小程序获取用户信息并进入首页。

插图

图5.6 小程序首页

等待调试器中的输出信息显示用户需要进行设备绑定，程序自动跳转至设备绑定页面。

插图

图5.7 小程序设备绑定页面

输入设备测试ID（“test”）以及测试密钥（“123456”），点击Register按钮，等待设备绑定成功，程序自动跳转至数据显示页面。

插图

图5.8 小程序数据显示页面

在PC端运行模拟设备客户端脚本，通过该页面可以看到脚本上传的测试图片与测试温湿度信息。

测试表明微信小程序各项功能运行正常。

5.4 系统综合测试

本节对整个系统进行综合测试。

为了使服务器能够真正运行在一个WSGI托管服务器上并接受来自嵌入式设备与手机的网络信息，需要首先使应用项目可安装化。在项目根目录下添加setup.py文件，在其中指定包名称、版本、目录以及依赖项等信息，并在MANIFEST.in文件中指定其他需要包含的文件夹，之后使用pip工具将项目打包为一个可安装的Python包。

表5.6 项目安装情况

|  |  |  |
| --- | --- | --- |
| Pakage | Version | Location |
| click |  |  |
| Flask |  |  |
| flaskr |  |  |
| itsdangerous |  |  |
| Jinja2 |  |  |
| pillow |  |  |
| pip |  |  |
| setuptools |  |  |
| Werkzeug |  |  |
| wheel |  |  |

打包完成后进行项目部署工作。使用wheel工具为应用构建一个.whl格式的发行版，之后复制该文件至部署位置并使用pip命令安装即可。

投入正式运行前，在项目实例文件夹创建一个setup.py文件用以装载配置信息如随机密钥、小程序AppID与SecretKey等。应用工厂在被调用时能够自动识别该文件并导入相关的配置。

在将应用投入运行时，使用产品级别的WSGI服务器替代flask的内建开发服务器。本设计选用waitress作为应用托管对象，这是一个由全部由Python实现、可以运行于Windows及Unix等多种环境下的产品级WSGI服务器，支持HTTP/1.0与HTTP/1.1。

在服务器开始运行后，启动嵌入式设备，等待其初始化完成并连接至服务器。之后启动微信小程序，按提示注册设备信息并完成登录。待应用跳转至数据显示页面后，即可通过屏幕看到来自嵌入式设备的摄像头数据与温湿度信息。

插图

图5.9 系统综合测试

测试表明整个系统各部分功能运行正常且相互之间能够正确协同。项目最终评价指标如下表所示。

表5.7 项目评价指标

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

6 总结与展望

总结与展望。

参考文献

[1] 邓亮才,肖卫初,黄楚喻,等. 基于STM32的书童机器人控制系统设计与实现[J]. 湖南城市学院学报(自然科学版), 2021, 30(1): 70-75.

[2] 李淑敬,孙楠,柴文君,等. 基于WebSocket服务型机器人远程交互的开发[J]. 西安文理学院学报(自然科学版), 2021, 24(1): 52-58.

[3] Lu li,Changkun wang,Fuqiang zhao. A Smart Home Control System Integrated the Improved Streaming Media Technology and Raspberry Pi[C]//Proceedings of 2015 Ssr International Conference on Social Sciences and Information(ssr-ssi 2015 V11), Singapore Management and Sports Science Institute, 2015: 656-659.

[4] Zhang ya'nan,Xiao guijin,Xu jiansheng. The Wireless Image Transmission System of Capsule Endoscope Based on Stm32f103[C]//Proceedings of 2016 2nd International Conference on Mechanical,electronic and Information Technology Engineering(icmite 2016), Destech Publications, 2016: 309-315.

[5] Deshun fan,Jichun zhao. The Design of Image Transmission System Based on Stm32f429zi[C]//Proceedings of 2017 4th International Conference on Machinery,materials and Computer(macmc 2017), Atlantis Press, 2017: 256-260.

[6] 姜艳茹,孟令军,尚桠朝,等. 基于STM32的图像采集与网络传输系统设计[J]. 自动化与仪表, 2020, 35(12): 45-49, 53.

[7] Zheng liu,Gang du,Wenhui zhuang, et al. Design of Camera Type Handling Trolley Based on Stm32[C]//Proceedings of 2019 3rd International Conference on Computer Engineering,information Science and Internet Technology(cii 2019), Clausius Scientific Press,canada, 2019: 145-150.

[8] 文馨,朱思曼,廖香芝. 基于微信小程序的学习平台研究[J]. 家庭科技, 2021, (1): 14-15.

[9] 朱宝明. 基于ZigBee和STM32的智能家居系统的研究与设计[D]. 华北理工大学, 2018.

[10] 刘后文,唐成章. 基于STM32的物联网智能家居控制系统分析[J]. 集成电路应用, 2021, 38(1): 16-17.

[11] 夏祥祥. 基于UCOSⅢ的嵌入式无线照片采集系统设计[J]. 科学技术创新, 2021, (2): 171-172.

[12] 褚治广,陈昊,张兴. 基于微信小程序的智能家居系统[J]. 辽宁工业大学学报(自然科学版), 2020, 40(6): 362-366.

[13] 高天,朱晨旭,刘博通,等. 基于OneNet云平台的多功能防盗监测系统设计[J]. 电子制作, 2020, 393(7): 25-28, 51.

[14] 黄松茂. 基于STM32的家庭环境监测系统的设计与实现[D]. 西北师范大学, 2018.

[15] 杨高扬. 远程遥控与家庭安防聊天机终端设计[D]. 西南交通大学, 2018.

[16] 鲍敏. 家用智能摄像头的网络安全问题及应对策略分析[J]. 现代信息科技, 2019, 3(13): 172-174.

[17] 柯亚文,蔡挺,夏晓峰,等. 面向嵌入式系统的加密算法性能检测方法[J]. 重庆大学学报, 2020, 43(11): 1-10.

[18] Ghoraani behnaz,Galvin james e.,Jimenez-shahed joohi. Point of View: Wearable Systems for At-home Monitoring of Motor Complications in Parkinson's Disease Should Deliver Clinically Actionable Information[J]. Parkinsonism and Related Disorders, 2021, 84: .

[19] Wang ju,Spicher nicolai,Warnecke joana m., et al. Unobtrusive Health Monitoring in Private Spaces: the Smart Home[J]. Sensors, 2021, 21(3): .

[20] Evaluation of the Effectiveness, Implementation and Cost-effectiveness of the Stay One Step Ahead Home Safety Promotion Intervention for Pre-school Children: a Study Protocol.[J]. Injury Prevention : Journal of the International Society for Child and Adolescent Injury Prevention, 2020, 26(6): .

[21] Evan r. polzer ma,Kathryn nearing phd M,Christopher e. knoepke msw P, et al. “safety in Dementia”: Development of an Online Caregiver Tool for Firearm, Driving, and Home Safety[J]. Journal of the American Geriatrics Society, 2020, 68(9): .

[22] 王凯彬,李梦思,陈佳雪,等. 基于STM32的居所智能防盗系统设计[J]. 山西电子技术, 2020, 213(6): 26-28.

[23] 王丹,刘国栋,张海涛,等. 基于ARM的嵌入式系统开发[J]. 微处理机, 2021, 42(1): 62-64.

[24] 辛慧娟,肖军. 基于NB-IoT技术的多传感器数据采集系统设计[J]. 微型电脑应用, 2020, 36(12): 35-37.

[25] Ghorayeb abir,Comber rob,Gooberman hill rachael. Older Adults' Perspectives of Smart Home Technology: Are We Developing the Technology That Older People Want?[J]. International Journal of Human-computer Studies, 2021, 147: .

[26] 张佳君. Design of Wireless Target Detection System Based on Stm32[D]. 华中师范大学, 2020.

[27] 盛琦鑫. 基于WebSocket和MSE的视频监控Web页播放的实现方法[J]. 广播电视网络, 2020, 27(12): 62-66.

[28] 刘宇健. 流媒体网络视频监控系统的研究与开发[J]. 电子技术与软件工程, 2020, 188(18): 15-17.

[29] 谈学钊,肖磊. 基于物联网云平台的热水实时监控设计[J]. 计算机与网络, 2020, 46(20): 67-70.

致 谢

致谢。