摘　要

市场上现存的共享单车系统均存在数据传输实时性差，对于单车的保护及管理也比较混乱，形成了现在杂乱的局面，在针对用户的推送方面存在出现推送出错、推送未到达的问题；基于以上原因设计此系统实现附近单车的查询以及自动开锁、自动定位等功能，为用户提供优质的共享解决方案。

设计采用 STM32 平台进行硬件系统的搭建，通过 MH 红外传感器与 SW 震动传感器完成红外感应、共享单车状态实时监测功能，并且采用无线技术通过 ESP8266 模块 TCP 协议实时将数据发送至服务器；Linux 平台进行 Web 端软件以及数据服务器开发，完成用户管理、单车管理以及数据存储与处理，主要采用 Golang 语言开发，运行后完成整个系统的数据处理以及响应功能，并且通过 MySql 数据库以及 Redis 完成 Model 层的数据存储功能；在报警方面采用 SMTP 邮件报警方式进行，添加邮件接收人后，当单车发生异常情况后即可向负责人员发送报警邮件。

系统硬件电路响应实时迅速、传输层连接建立快捷、数据传输快速、软件简洁实用，并且可以很方便的对单车状况进行实时的检测以及控制，相比于市场现存的产品具有更突出的优势，能为用户提供更加优质、便捷的服务。

关键词：物联网，监测，共享单车

Abstract

The market has the problem of real-time data transmission for existing shared bicycle system. the current situation of bike's management and protect is muddled ,and there are problems of pushing errors and pushing unarrived in pushing for users. This system to realize the query of the nearby bicycle and automatic unlocking, automatic positioning and other functions base on this problem, to provide with high quality sharing solutions the users.

This design set up the hardware system uses the STM32. The real-time monitoring function of infrared induction and shared bicycle status is accomplished through MH infrared sensor and SW vibration sensor. And wireless technology is used to send data to the server in real time through the ESP8266 module by TCP protocol. Web software, user management and data server, bicycle management and data storage development by Linux, and the data process and response functions of the whole system development by Golang, and completes the data storage function of the Model layer through the MySql database and the Redis after it runned. In the alarm area, SMTP alarms are used. An alarm email can be sent to the addressee when an abnormal situation occurs in the bikes.

The system hardware circuit responds rapidly in real time, the connection layer of the transmission layer is established quickly, the software is simple and the condition of the bicycle can be conveniently detected and controlled in real time. Compared with the existing products in the market, it has more outstanding advantages and can provides users with more quality and convenient services.

Key words: Internet of Things, Monitoring, Shared Bikes

目 录

第 1 章 绪论 1
1.1 设计背景及意义 1
1.2 设计目标 2
第 2 章 系统总体设计 3
2.1 系统总体结构 3
2.2 设计方案 4
2.2.1 MCU 核心板选择 4
2.2.2 单车锁监测方式选择 5
2.2.3 传输方式选择 6
2.2.4 应用层开发方式选择 7
第 3 章 系统详细设计 8
3.1 系统感知层的详细设计 8
3.1.1 MCU 核心模块设计 8
3.1.2 红外采集模块设计 9
3.1.3 震动采集模块设计 11
3.1.4 单车锁控制模块设计 13
3.1.5 系统供电模块设计 15
3.2 系统传输层的详细设计 16
3.2.1 传输层通信协议 16
3.2.2 传输模块的电路设计 21
3.3 系统应用层详细设计 23
3.3.1 数据库设计 23
3.3.2 功能模块设计 27

第 4 章 系统测试 41
4.1 感知层测试 41
4.2 传输层测试 42
4.3 应用层测试 43
4.3.1 TCP 数据收发服务功能测试 43
4.3.2 Web 端软件功能测试 44
结 论 46
致 谢 47
参考文献 48
附录 设计系统部分源代码 50

Contents

Chapter I Introduction 1
1.1 Design background and significance 2
1.2 Design goals 2
Chapter II Overall System Design 3
2.1 Overall System Structure 3
2.2 Design Scheme 4
2.2.1 System Control Plane Selection 4
2.2.2 Sensing Layer Module Selection 5
2.2.3 Transmission Layer Module Selection 5
2.2.4 Application Layer Develops Module Selection 6
Chapter III Detailed System Design 8
3.1 Detailed Design of System-Aware Layer 8
3.1.1 MCU Core Module Design 10
3.1.2 Infrared Acquisition Module Design 11
3.1.3 Vibration Acquisition Module Design 13
3.1.4 Bicycle Lock Control Module Design 14
3.1.5 Design of system power supply module 17
3.2 Detailed Design of System Transport Layer 18
3.2.1 Transport Layer Communication Protocol 18
3.2.2 Circuit Design of Transmission Module 22
3.3 System Application Layer Detailed Design 24

3.3.1 Database Design 24
3.3.2 Functional Module Design 28
Chapter IV System Testing 41
4.1 Perception Layer Testing 41
4.2 Transport Layer Testing 42
4.3 Application Layer Testing 43
4.3.1 TCP Service Function Test 43
4.3.2 Web Client Software Function Test 44
Conclusion 46
Acknowledgements 47
Reference 48
Appendix Design System PartSource Code 50

第 1 章 绪论

本章通过对系统的开发背景、设计目标、最终目的以及意义进行相关叙述，主要分析论述国内外共享单车的实现技术以及现状，同时分析基于物联网技术的共享单车的技术前景，进一步的进行需求的分析及设计。

1.1 设计背景及意义

共享单车自荷兰发起后在全球迅速发展，在广大区域为人们的出行提供了极大的便利，缓解了很多大城市的堵车通病，并且随着经济能力的增强，绿色出行成为了很多人所追捧的。当今市场共享单车鱼龙混杂，很多的平台在当前的大环境下开始推广这一方式，但是能够长久的存留并且为用户提供便捷服务的却寥寥无几；通过分析不难发现，目前市场的共享单车存在很多问题，首先在数据传输方面并不实时、迅速，极大的降低了用户的体验，在自身安全性方面，也存在很多问题，比如部分单车的密码锁是固定的，只需要一次开锁便可以一直使用，这也是硬件锁导致的，同时在管理方面也存在很大的问题，乱停乱放、硬件破坏严重，已经从便捷变成了影响交通及城市管理。

根据市场目前共享单车所存在的诸多问题，亟需一款能够反应迅速并且使用便捷的共享单车，所以进行了本系统的设计。系统结构简单、管理方便，并且便于维护以及监控，有效避免单车遭受破坏，同时通过对单车的定位能够方便的进行管理，避免造成交通的拥堵，为市场提供了良好的解决方案。

本设计通过感知层、传输层、应用层三层的协调完成用户从寻找车辆、下单、开锁、关锁以及使用结束支付的过程，将“最后一公里”问题通过技

术驱动进行解决；感知层采用 Stm32 将用户开锁、关锁的过程进行智能化，使目前机械锁存在的问题加以解决，电磁锁方面通过继电器进行升压，解决了目前市场现存共享单车的开锁不稳定性以及电压不足的情况；传输层通过 WIFI 解决数据的不稳定传输以及丢包问题；应用层通过 golang 实现了数据的传输，同时采用 Web 端解决目前内存紧张的问题；总体来看该系统旨在使硬件电路响应实时迅速、传输层连接建立快捷、数据传输快速、软件简洁实用，并且可以很方便的对单车状况进行实时的检测以及控制，相比于市场现存的产品具有更突出的优势，能为用户提供更加优质、便捷的服务。

1.2 设计目标

设计通过Stm32单片机完成共享单车的底层控制以及感知层的数据采集，通过电磁技术完成单车开锁以及上锁功能，通过服务器对单片机发送指令完成电磁锁的控制，通过红外传感器以及振动传感器完成防盗报警功能，检测到异常数据时即可通过服务器对商家进行邮件报警;传输层采用 WIFI 技术完成，感知层采集相关数据后通过 WIFI 模块将数据传输到服务器进行处理、转换以及存储监测;系统的服务端通过 golang 线程对采集到的数据进行实时监测，采用 MySql 数据库将采集到的数据进行存储，通过主流 Vue 完成前端操作界面的编写以及通过 golang 完成服务器端程序的编写。

系统中 MH 红外传感器延迟为 0.5s,封锁时间为 0.2s,采集距离为 2-3m，感应范围锥角为 120°；SW-1801P 振动传感器采用模拟量数据输出，振压为 10Pa 以上，精度保持为 1.5u；ESP8266 模块在传输过程中数据延迟为 20ms，采用字节流传输，传输包大小为 16byte，传输速度为 800kb/s 左右，连接失败时候间隔 30s 重试;系统服务器应用程序 api 实时处理速度为 200ms 以下，对于 db 操作较多的 api 保持为 150-200ms 之间,api 超时响应率在 0.1%左右。

第 2 章 系统总体设计

本章主要介绍系统的总体架构以及各模块的方案设计。主要从系统各个模块的功能以及设计方案选择方面切入，完成系统整体的技术方案设计以及各模块的方案选择。

2.1 系统总体结构

系统基于物联网的三层基本架构感知层、传输层以及应用层进行技术方案设计，系统组成架构如图 2-1 所示。

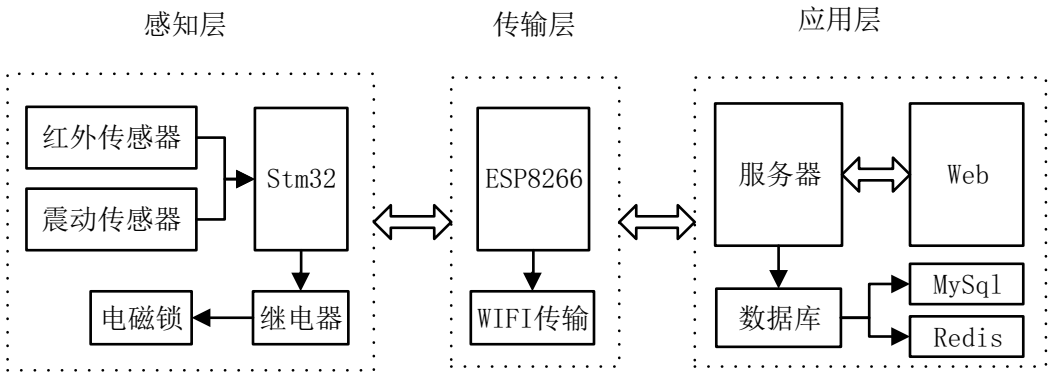


图 2-1 系统结构图

感知层由振动传感器、红外传感器组成，分别对单车的上锁状态以及单车是否遭受破坏进行检测，Stm32 主板主要控制传感器的中断以及时序操作，完成对传感器数据的接收以及处理后通过串口传递到传输层进行进一步的处理，同时在接收到传输层回环控制指令后通过继电器驱动电磁锁完成单车的

开关功能。

传输层采用ESP8266模块完成,实现数据发送以及局域网数据接收。Stm32主板接收到感知层回传的数据并对其进行数据处理后将数据以TTL格式通过串口发送到WIFI模块,WIFI模块接收到数据后需要与服务器建立TCP连接，传输层模块通过TCP将数据以指定协议格式传输到服务器，服务器接收到数据后做进一步的处理，同时ESP8266通过局域网接收服务器的指令推送，用户进行开锁操作时通过Socket使用TCP协议将指令发送到WIFI节点完成对单车锁的开关。

应用层包含了数据处理程序、Web端应用程序以及后台管理程序。数据处理程序采用协程监听的方式运行，监听到客户端的连接请求后开始建立Socket短连接，将接收到数据进行转换处理和校验，数据异常则通过邮件进行报警，同时将数据存储到MySql数据库中；Web端应用程序主要为用户提供服务,采用前后端分离的方式进行开发,后端采用Golang开发HTTP服务，前端采用Vue.js进行开发，通过vue-resource完成HTTP接口的请求；应用层的Redis则主要作为缓存载体，对数据操作进行缓存，更好的提高接口的响应速度以及减少对DB的硬操作。

2.2 设计方案

2.2.1 MCU核心板选择

1．STC89S51

STC89S51基于8位单片机处理芯片STC89S51RC搭建,主要应用于电路控制，在数据处理方面的能力较弱，对于功能复杂的系统很难满足需求。可完成一系列简单的控制应用，例如可应用于温度检测、简易循迹小车以及简易

无人机等。

2．Stm32F103ZET6

Stm32F103 系列单片机是一款 32ARM 微控制器，内核采用 Cortex-M3，具备 12k-512k 的内存容量[1]。具备 A、B、C、D、E、G 六路 I/O 口，串口为 Uart1、Uart2 以及复用 I/O 功能口 Uart3，采用 Usb 供电，供电电压为 2-3.6V，晶振为 4-16MHZ。此款单片机可实现一些复杂的应用，通过 Uart 口可以快速的实现串口的通信，同时进一步的驱动 WIFI 以及 GPRS 等无线通信模块；对于传感器的驱动可使用库函数方式编程快速实现，通过封装的库进行快速的开发，同时具备多个 VCC 以及 GND 口，能够同时连接以及驱动多个模块。

设计中需要通过串口进行感知层数据采集测试，同时在传输层进行数据传输时需要将主控板的串口与 ESP8266 进行连接，在设计中就要求主控板具备两路以上 Uart，同时具备 I/O 复用功能以及多个供电 I/O 口，以便驱动红外传感器、震动传感器以及 WIFi 模块；经过方案对比 Stm32F103ZET6 更具备设计所需要的功能，所以设计选择 Stm32F103ZET6 作为系统核心板。

2.2.2 单车锁监测方式选择

系统中需要对单车的上锁状态进行监测，数据检测后通过数据处理判断单车上锁状态以及完成自动上锁。

1．MH 红外对管

MH 红外传感器对环境光线适应能力强，具有一对红外线发射与接收管，其有效测试距离为 2-30cm，检测角度为 35°，工作电压为 3.3-5V[2]。同时该传感器采用 LM393 比较器，具有极强的稳定性，采用数字量输出，可直接通过 Stm32 单片机进行 I/O 输入驱动进行使用。

2．超声波模块

HC-SR04 超声波模块主要用于在实际应用中的距离检测，目前广泛应用于车载系统中。通过发射器发射信号后采用接收器通过数据解析得到目前的距离，其感应角度小于 15°，探测距离为 2cm-450cm，精度在 5mm 左右。

设计中单车锁的开关主要通过光敏信号进行检测，检测角度需大于 20°，同时系统要求单车锁检测需具备极强的稳定性，通过 I/O 直接进行数字量信号读取；经过方案对比，MH 红外对管较超声波模块更符合设计的需求。

2.2.3 传输方式选择

传输层主要完成设计数据的双向传输，即感知层到服务器的双向传输，需要将感知层采集到的数据传输到服务器进行处理，同时需要完成服务器对感知层的指令传输。

1．ESP8266 WIFI 传输

ATK-ESP8266 模块采用无线组网的方式进行通信，具备 AP、STA 以及 AP+STA 三种模式，支持 TCP 以及 UDP 协议，可将 TTL 数据转换为 RS232 数据进行传输[3]。芯片内置了 TCP/IP 协议栈，使用 A-MPDU、A-MSDU 的聚合以及 0.4us 的保护间隔，其待机功耗小于 1.0mW；在使用时可以作为服务端、客户端以及客户端加服务端的方式运行，通过组建局域网、开启透传模式进行数据的实时接收以及发送。

2．HC-05 蓝牙模块

HC-05 一体蓝牙模块输入电压为 3.6-6v，可以直接与单片机串口进行连接，不需要经过 MAX232 芯片的处理，在空旷地带传输距离为 10m 左右，但在数据丢包方面会有一些缺陷。

设计中传输层需具备极强的稳定性，需要保证数据的实时、迅速传输，能够实现 TTL 数据的格式转换，可以作为 Socket 服务端以及客户端使用，实现终端的数据传输；经过方案对比，蓝牙模块不具备设计所需求的功能，所以选择 ESP8266 作为传输层模块完成数据的上传以及指令的下发功能。

2.2.4 应用层开发方式选择

1．App

市场现存的共享单车应用层控制均为 Android App 方式开发，在界面方面使用 Layout 完成布局，通过 xml 文件进行界面文件的编写，通过请求 Java 后端进行数据的交互以及处理，在数据库方面使用 Sqlite 完成基本用户数据以及 App 临时数据缓存；在数据存储方面采用 Java 线程通过 Socket 监听完成 TCP 数据处理服务的搭建，在邮件报警方面通过 JavaMail 使用 SMTP 协议完成，JavaMail 具备封装好的 Apache Java 库，可以直接使用。

2．Web

使用 Web 的形式进行应用层开发，在数据接收处理方面采用 Golang 的协程通过 Socket 进行数据的传输，异常数据报警方面采用 goMail 通过 SMTP 协议进行邮件的发送；后端 api 采用 Golang 开发，使用 Gin 框架作为主要架构进行开发，使系统具备高可用性；前端应用程序需要能够与后端分离，单独构成一个应用程序。

设计中应用层需具备升级、维护简便、响应快速以及结构清晰的特点，同时需具备数据接收以及指令推送的功能；经过方案对比，Web 端具有更强的灵活性，在应用升级以及用户使用内存方面都具备很大的优势，使用前后端分离的方式也使系统架构更加清晰，在整体开发过程中也更加简便，同时满足设计的需求，所以设计采用 Web 端完成应用层的搭建。

第 3 章 系统详细设计

本章主要介绍感知层、传输层以及应用层的详细设计，包括硬件电路、工作流程以及软件实现逻辑等内容的描述。

3.1 系统感知层的详细设计

系统感知层包括主控板、红外传感器以及震动传感器模块，完成感知层的数据采集、数据处理以及硬件控制功能。

3.1.1 MCU 核心模块设计

设计采用 Stm32F103ZET6 作为主控芯片进行整体硬件系统核心的搭建，开发板具备 7 路 I/O 口，串口通信采用 Uart1、Uart2，同时支持 Uart3 端口复用，开发板内置 ADC 芯片，可直接驱动 I/O 口完成 A/D 转换；核心模块设计主要包括开关电路以及晶振电路模块的设计，通过模块之间的相互协调工作完成整体的系统主控制。

1．晶振电路

在系统中，晶振电路为系统提供基本的时钟信号以便用来完成程序运行的控制；晶振电路如图 3-1 所示，C1 与 C2 两个电容为晶振起振提供输入，作为晶振工作的一个必要条件，电路中 R1 主要使提高了系统中振荡器的稳定性，使逻辑反相器的器件工作在线性区，进一步的获得增益而使晶振得到振荡。

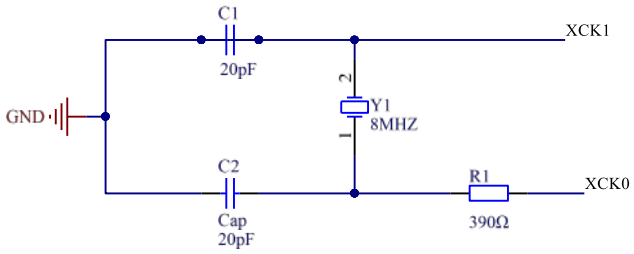


图 3-1 晶振电路图

2．复位电路

系统复位电路包括按键复位以及上电复位两种方式，主要作用为使 CPU 处于确定的初始状态，同使单片机从初始状态开始工作。复位电路如图 3-2 所示，S1 作为按键复位使用，按下开关后系统在 RST 加入高电平，系统开始初始化进入初始工作状态。上电复位时系统通过 C1 电容输入电平，电容对 RST 输入短暂的高电平后，高电平信号会随着 VCC 对电容的充电过程而逐渐回落；电路中 R1 电阻主要起到保护电路的作用，避免因 RST 端的下拉电阻出错时而导致电平过高而对系统造成损坏[4]。

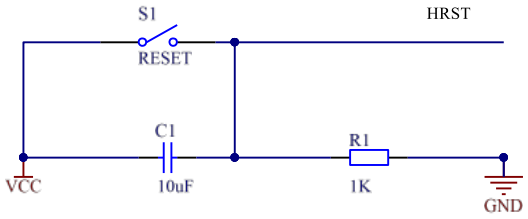


图 3-2 复位电路图

3.1.2 红外采集模块设计

1．MH 红外传感器电路设计

MH 红外传感器采用一对红外线发射管与接收管完成对障碍物的监测。红外传感器电路如图 3-3 所示。电平信号在经过 LM393 比较器处理以后向指示灯输出高电平，同时传感器 DO 口输出低电平数字信号，系统通过驱动主控板 PA8 为输入模式，连接传感器 DO 口接收数字信号，接收数字信号后通过电平信号传递给主控板进行使用。如图 3-3 所示。

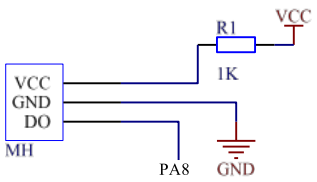


图 3-3 红外传感器电路图

MH 红外传感器具备三个 I/O 口，分别为 VCC、GND 以及数字量输出 OUT 口，在使用时 VCC 采用 3.3V 供电，GND 连接单片机的接地口，数字量输出口连接单片机 I/O 口 PA8，与单片机连接如表 3-1 所示。

表 3-1 红外传感器模块硬件线路连接表

MH 红外传感器引脚	STM32 主板连接引脚	说明
GND	GND	接地线
VCC	VCC 3.3V	3.3V 供电
DO	PA8	数字信号输出

2．MH 红外传感器软件设计

MH 红外传感器为数字量输出传感器，在使用时只需要驱动 I/O 口为输入

模式，即可在传感器检测到障碍物的时候接收到低电平信号，进一步的通过处理器处理后进行传输。其软件流程图如图 3-4 所示。传感器开始工作时调用 Init 函数完成 I/O 口的设置以及初始化，初始化过程中主要通过 GPIO_InitStructure 结构完成对 I/O 的相关初始化设置，在工作过程中通过 GPIO_ReadInputDataBit 函数完成对 I/O 口数字信号的读取。

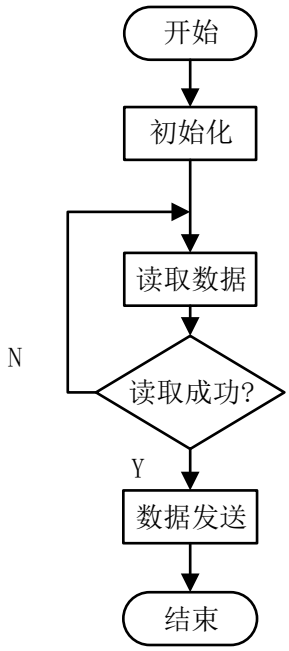


图 3-4 红外传感器软件流程图

3.1.3 震动采集模块设计

1．SW 震动传感器电路设计

SW 震动传感器为两路探头感应以及两路 OUT 口输出，工作电压为 3.3V-5V，

驱动能力超过 15mA[5]。震动传感器电路图如图 3-5 所示。传感器采用 SW-420 高灵敏震动开关，在感应头监测到震动时 SW 开关导通，同时在输出端输出低电平信号，感应头不震动时开关断开，输出端输出高电平信号，通过连接主控板 PD5、PD6 口完成数字信号的接收，通过 TTL 格式将数据发送到主控板处理使用。

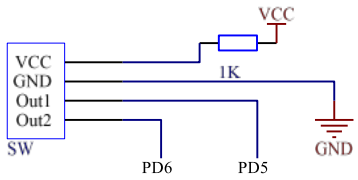


图 3-5 震动传感器电路图

震动传感器具备 VCC、GND 以及 OUT1、OUT2 四个 I/O 口，在使用时 VCC 与单片机的 3.3V 连接，GND 连接单片机的接地线，同时可以直接驱动单片机主板的 I/O 口来进行 OUT 口数字信号的输出。与单片机连接如表 3-2 所示。

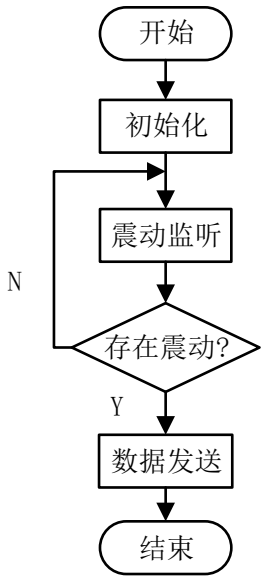
表 3-2 震动传感器模块硬件线路连接表

SW 震动传感器引脚	STM32 主板连接引脚	说明
GND	GND	接地线
VCC	VCC 3.3V	3.3V 供电
DO1	PD5	Out1 信号输出
DO2	PD6	Out2 信号输出

2．SW 震动传感器软件设计

震动传感器采用两路输出，输出信号为数字信号，在使用时需要将 I/O 口驱动为输入模式，实时接收震动传感器的状态，当输出为低电平时即表示

存在震动，数据异常[6]。其软件流程如图 3-6 所示，震动传感器在工作时通过 GPIO_Init 完成引脚初始化工作，初始化过程中需要将 I/O 通过 GPIO_MODE_INPUT 设置为浮空输入模式，同时设置 I/O 操作速度为 GPIO_SPEED_FREQ_LOW，在工作过程中通过 HAL_GPIO_ReadPin 完成模块输出信号的读取，经过 ScanDelay 延时消抖后输出。



3-6 震动传感器软件流程图

3.1.4 单车锁控制模块设计

1．电磁锁电路设计

单车锁采用电磁锁进行设计，电磁锁通过电生磁原理，具有通电即锁、断电即开的特性[7]。单车锁电路如图 3-7 所示，通过与主控板 PD9 引脚连接

完成对电磁锁的控制，引脚输出高电平信号时产生磁性，锁片被吸附后上锁完成，当接收到低电平信号后磁锁断电失去磁性，锁片开始也磁锁分离完成开锁功能;在电路中使用三极管对电压进行放大,使其足够对磁锁进行驱动。

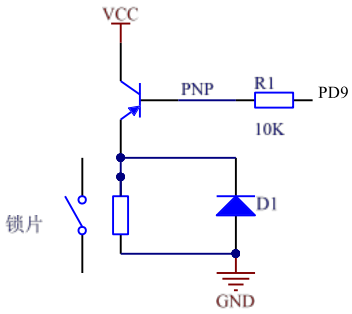


图 3-7 电磁锁电路设计

电磁锁采用 5V 供电，通过继电器与单片机连接[8]。单片机驱动 I/O 口为推挽输出模式，在需要开锁时发出低电平信号，I/O 口向继电器输出信号后电磁锁断电完成开锁功能。继电器与单片机连接如表 3-3 所示。

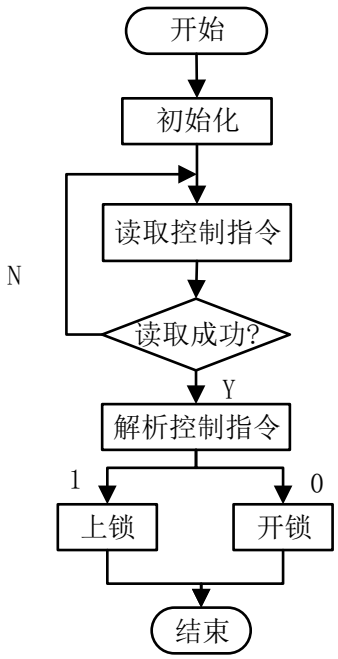
表 3-3 继电器模块硬件线路连接表

MH 红外传感器引脚	STM32 主板连接引脚	说明
GND	GND	接地线
VCC	VCC 3.3V	3.3V 供电
CH1	PD9	控制端口

2．电磁锁软件设计

电磁锁的开关通过继电器进行驱动，其软件流程图如图 3-8 所示，在使用时通过 GPIO_Mode_Out_PP 设置单片机 I/O 口为推挽输出模式，设置 I/O

口速度为 GPIO_Speed_50MHz，同时通过 RCC_APB2PeriphClockCmd 使能核心板的端口时钟；工作过程中向 I/O 口发送低电平时候电磁锁打开，发送高电平时电磁锁上锁，在设计中默认向继电器的 CH1 控制端口输出高电平，需要开锁时输出低电平。



3-8 电磁锁软件流程图

3.1.5 系统供电模块设计

系统采用 3.3V 的正向低压稳压器完成稳压供电电路的设计，输出电压为 3.267V-3.333V，电压差最大不超过 1.3V，随着负载电流的减小也会逐渐降低[9]。稳压供电电路图如图 3-9 所示，C1 与 C2 作为输入电容，将单向脉动

电压转换为直流电压使用，同时 C3 与 C4 作为滤波电容，主要用来抑制自激振荡，避免稳压器输出为振荡波形，以达到系统稳压供电的作用。

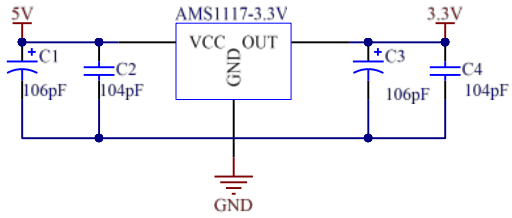


图 3-9 稳压供电电路图

3.2 系统传输层的详细设计

3.2.1 传输层通信协议

1．终端节点与网关节点之间通信协议设计

（1）终端节点的设计

终端节点与网关节点设计结构图如图 3-10 所示，设计中 ESP8266 采用 STA 工作模式，通过与单片机的连接作为一个终端节点，终端节点在底层需要完成与单片机的数据交互，数据交互采用 TTL 数据格式，单片机向 WIFI 模块传输 TTL 格式数据后，通过 WIFI 模块内置芯片将 TTL 数据转换为 WIFI 数据格式。WIFI 模块作为一个网关节点，接受多个终端节点的连接请求，处理来自多终端节点的数据请求，完成进一步与服务器的数据交互。

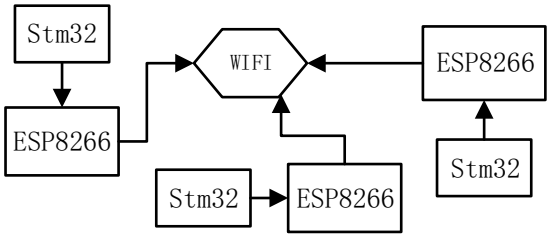


图 3-10 终端节点架构设计图

（2）终端节点与网关节点数据交互的设计

终端节点数据交互图如图 3-11 所示，设计中数据交互主要包括 Stm32 与 ESP8266 的 TTL 数据交互以及 WIFI 模块与局域网关的 WIFI 数据交互。Stm32 单片机接收到数据后通过串口向 ESP8266 WIFI 模块发送 TTL 数据，TTL 数据经过 WIFI 模块内置芯片转换为 WIFI 数据进行下一步的发送。

图 3-11 终端节点数据交互图

（3）协议帧格式的设计

终端节点 TTL 数据格式图如图 3-12 所示，设计在终端节点采用 TTL 格式数据向 WIFI 模块发送，经过 ESP8266 内置芯片进行数据转换为 WIFI 协议数据发送到网关节点。数据格式包括起始位、数据位以及校验位，接收到数据后从起始位进行数据读取，同时需要通过校验位对数据进行校验，校验成功读取数据位数据，读取到 8 字节后数据读取完成。

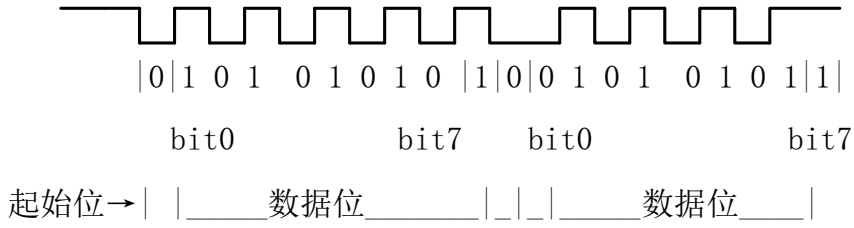


图 3-12 终端节点 TTL 数据格式

设计中需要通过网关节点向终端节点发送单车锁开关信息，实现单车的开关，设计中采用 Stm32 便于读取的协议格式进行信号发送，Stm32 单片机通过接收网关节点的数据指令对电磁锁进行相应的控制；数据主要包括校验位与指令位，格式如表 3-4 所示。

表 3-4 终端节点数据交互格式

数据头		校验	指令	数据尾	
2Byte		1Byte	1Byte	2Byte	
0XAA	0XAA	0X06	N	0X55	0X55

终端节点数据交互格式包括数据头、校验位、指令位以及数据尾。数据头用于进行数据标识,进行数据交互时通过检测数据头 0XAA 保留所需要的有用数据，对数据头进行分离；校验位在数据交互时主要用于检测接收数据的有效性，将校验位设置为 0X06 表示数据正常有用；指令位为数据包中需要处理使用的数据，模块接收到数据后检测指令位并且解析使用，本设计中指令为 0 与 1 两种，0 代表开锁，1 代表上锁；数据尾主要用于进行数据完整性的判断，本设计将数据位设为 0X55，数据在检测 0X55 后数据完整正常。

2．网关节点与服务器之间通信协议设计

（1）网关节点与服务器数据交互的设计

网关节点采用 WIFI 组网方式实现，服务器使用 TCP/IP 协议进行数据的接收以及发送。WIFI 网关节点在接收到数据后通过 Socket 采用 TCP 协议将数据发送到服务器，服务器通过对端口的监听实时接收来自网关节点的 Socket 数据。数据交互流程如图 3-13 所示。

图 3-13 网关节点与服务器交互流程图

（2）网关节点与服务器数据交互协议帧的设计

网关节点与服务器采用 TCP 协议格式进行数据交互，通过 Socket 进行数据的传输，数据帧主要包括端口号、序号、确认序号、数据头、校验和以及数据。在 TCP 数据交互过程中，包括 16 位端口号，用于服务端进行端口校验，目标端口号则用于发起连接请求时使用，连接时需要通过 32 位序号进行协议解析，同时需要通过 32 位确认需要进行连接确认，具体格式如表 3-5 所示。

表 3-5 网关节点与服务器数据帧格式

16 位源端口号							16 位目标端口号
32 位序号							
32 位确认序号							
4 位 Head	URG	ACK	RSH	RST	SYN	FIN	16 位窗口大小
16 位校验和							16 位紧急指针
选项							
数据							

网关节点与服务器通过 TCP 协议进行数据交互[10]。进行数据交互时源端口号为 WIFI 模块的开放通信端口，该端口在进行 ESP8266 模块配置时生成；目标端口号为服务器开放 Socket 接收端口,在进行使用时需要将端口进行开放或者使用默认的 80 开放端口；32 位序号为 WIFI 模块发送的 TCP 序号，表示发送数据报的长度；确认序号为发送数据报的 sq 值以及数据报长度，用于标识发送内容的字节数；窗口大小表示数据交互可发送的字节数，数据交互时设置窗口大小主要用于避免超出网络负载时而出现的数据堵塞；校验和包括 TCP 头、数据报内容以及伪头部，包含了源 IP、目的 IP 以及 TCP 所使用的协议号；紧急指针主要用于紧急数据的发送，在数据中放置紧急数据后，紧急数据的下一位为紧急指针，检测到数据中包含紧急指针后将 URG 置位，在紧急数据发送完成后 URG 完成复位；选项部分主要用于针对特定的环境进行头部设置，长度为 40byte，最大总长可设置 60byte；数据位即为发送的数据报内容本身，服务端在接收到数据报后通过解析数据位获取数据报内容。

3．数据传输的具体实现

设计中采用 golang 语言作为 TCP 服务开发语言，通过 Socket 监听端口接收节点发送来的数据信息。在终端节点通过 ESP8266 作为传输介质进行数据传输，通过 Stm32 对其进行控制[11]。其实现流程图如图 3-14 所示，模块开始工作时通过 ESP8266_Init 函数完成初始化工作，包括 ESP8266_GPIO_Config 引脚初始化以及 ESP8266_USART_Config 串口初始化，串口初始化过程中主要包括 GPIO_InitStructure 复用功能配置、USART_InitStructure 串口模式配置以及 USART_ITConfig 中断接收配置，初始化完成后通过 ESP8266_Net_Mode_Choose 函数完成模式的选择，ESP8266_Cmd 完成 ESP8266 的指令发送控制，配置完成后通过 ESP8266_SendString 完成数据的发送。

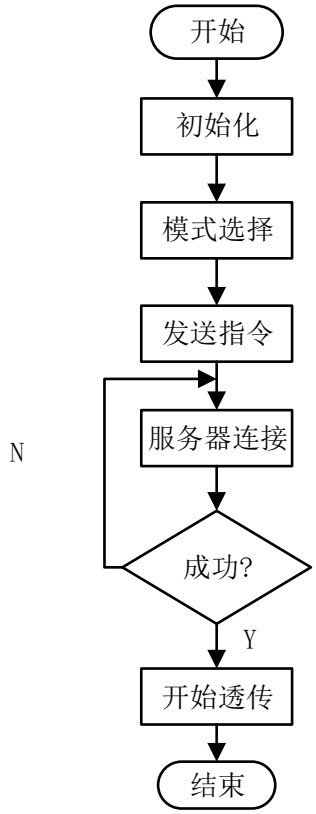


图 3-14 ESP8266 实现流程图

ESP8266 通过 STA 模式连接 AP，连接建立成功后 ESP8266 通过 TCP 开始发送数据，在建立连接的过程中模块具备重连机制连接失败后进行重试。

3.2.2 传输模块的电路设计

传输模块电路图如图 3-15 所示，传输模块采用 ATK-ESP8266 完成数据的无线传输。ESP8266 通过与单片机串口连接完成 TTL 数据到 RS232 数据的转换，在接收到单片机数据时通过内置芯片完成 TTL 电平数据的转换，数据转

换后通过 TCP 协议将数据发送到指定服务端；服务器发送指令时，ESP8266 将数据转换为 TTL 电平数据通过 Uart 串口传输到单片机进行硬件控制。

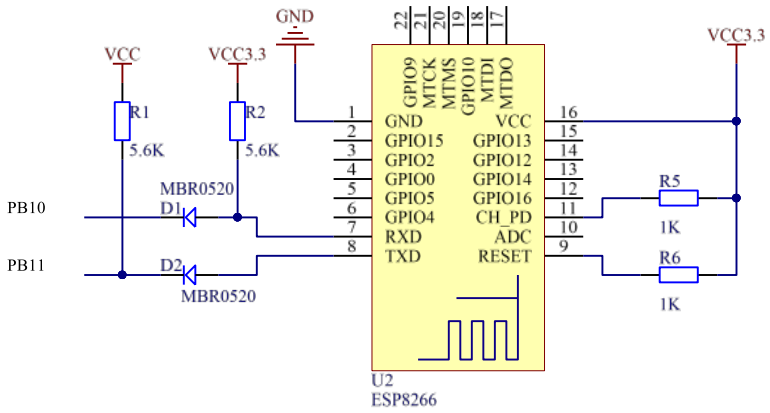


图 3-15 传输模块电路图

ESP8266 具备 6 个 I/O 口，分别为 GND、VCC、TXD、RXD、IO 以及 RST[11]。GND 与 VCC 主要用于模块的供电，IO 端口主要用于模块的设置，RXD 与 TXD 相互配合完成数据的接收，在设计中单片机通过 I/O 复用通过 Uart3 与 ESP8266 完成数据的交互。引脚连接表如表 3-6 所示。

表 3-6 WIFI 模块硬件线路连接表

ESP8266 引脚	STM32 主板连接引脚	说明
GND	GND	接地线
VCC	VCC 5.0V	5V 供电
RXD	PB10	串口数据接收
TXD	PB11	串口数据发送
IO_O	PB8	模块配置
Rst	PB9	模块复位

3.3 系统应用层详细设计

3.3.1 数据库设计

设计中主要数据采用 MySql 进行存储，使用 Innodb 引擎、UTF8mb4 字符编码，支持事务、存储过程等操作，具有良好的数据存储与查询能力。

1．数据库概念结构设计

E-R 图如图 3-16 所示，设计中主要数据存储为用户信息表、用户元信息表、后台管理用户信息表、单车信息表、单车使用信息记录表、用户反馈表、用户余额表以及充值记录表。设计中用户信息表作为主表使用，信息表的 ID 作为用户的 UID，作为用户数据表关联的主键。

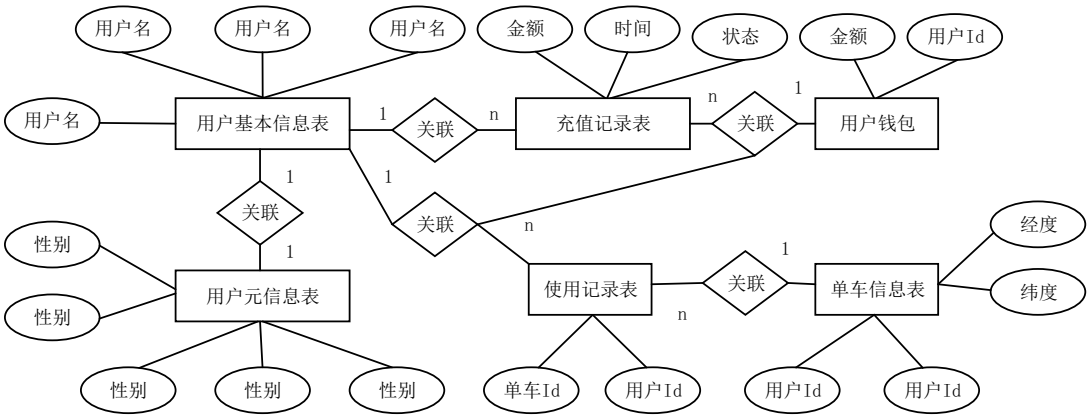


图 3-16 E-R 图

2．数据库逻辑结构设计

设计中共用到用户基本信息表、用户元信息表、单车信息表、用户反馈表、用户充值记录表、用户使用记录表以及后台管理用户表。设计中每一张表都需要具有创建时间、更新时间以及删除标识三个字段，具体设计如下。

（1）用户信息表

用户信息表用于保存对应的用户信息，如下表 3-7 所示。

表 3-7 用户信息表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
username	varchar	50	用户名
password	varchar	50	用户密码
phone	varchar	20	手机号
email	varchar	50	邮箱
card_name	varchar	20	认证姓名
delated	tinyint	4	删除标识

（2）用户元信息表

用户元信息表主要用于存储用户相关的元信息，如表 3-8 所示。

表 3-8 用户元信息表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
uid	int	11	用户 ID
sex	int	11	性别
bio	varchar	50	描述
card_name	varchar	50	真实姓名
card_id	varchar	50	身份证号

（3）用户余额表

用户余额表主要用于存储用户余额信息，如表 3-9 所示。

表 3-9 用户余额表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
uid	int	11	用户 ID
balance	int	11	余额

（4）用户充值记录表

用户充值记录表主要用于记录用户充值流水记录，如表 3-10 所示。

表 3-10 用户充值记录表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
uid	int	11	用户 ID
count	int	11	充值金额
status	int	11	订单状态

（5）用户反馈记录表

反馈记录表主要用于存储用户反馈信息，如表 3-11 所示。

表 3-11 反馈记录表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
uid	int	11	用户 ID
content	varchar	500	反馈内容

（6）单车信息表

单车信息表主要用于存储单车相关信息，如表 3-12 所示。

表 3-12 单车信息表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
by_id	int	11	单车编号
long_tude	varchar	48	经度
lati_tude	varchar	48	纬度
status	int	11	单车状态

（7）用户使用记录表

用户使用记录表主要记录用户使用单车的情况记录，如表 3-13 所示。

表 3-13 用户使用记录表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
by_id	int	11	单车编号
uid	int	11	用户 Id
start_ts	bigint	20	使用时间
start_ts	bigint	20	使用时间
amount	int	11	消费金额
deleted	tinyint	4	删除标识

（8）后台管理用户表

后台管理用户表主要用于存储后台管理用户信息表，如表 3-14 所示。

表 3-14 后台管理用户信息表

字段	数据类型	字段长度	字段说明
id	int	11	数据编号
username	varchar	50	用户名
name	varchar	50	真实姓名
password	varchar	50	密码
is_admin	int	11	管理员
login_ts	bingint	20	登录时间

3.3.2 功能模块设计

应用层主要包括服务端数据传输服务、共享单车 Web 端服务、后台管理服务。TCP 数据传输服务用于完成 WIFI 数据的接收存储以及单车锁指令控制的发送，共享单车 Web 服务主要完成用户使用操作界面，Web 后台管理服务主要完成单车数据以及用户数据的管理。后台服务采用 Golang 语言编写，通过 MakeFile 打包运行在 Linux 服务器上[12]；Web 前端界面采用 Node.js 以及 Vue2.0 完成，通过 Npm 运行以及完成依赖安装。

1．数据收发模块

（1）数据接收服务

数据接收服务流程图如图 3-17 所示。服务启动后监听指定端口，监听到有连接请求时开始建立连接，建立连接后即可开始接收数据，接收数据包后判断数据是否存在异常，数据异常时发送邮件报警，同时将数据存储在数据库中[13]。

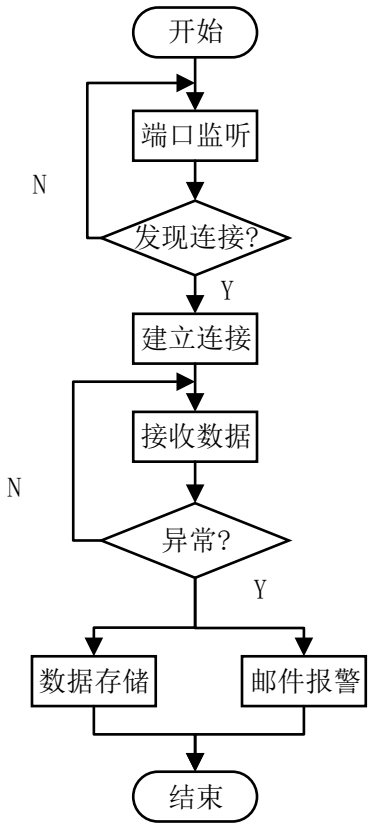


图 3-17 数据接收服务流程图

（2）指令发送服务

指令发送服务用于单车开、关锁的指令发送，指令发送流程图如图 3-18 所示。在进行单车开、关锁时服务作为 Socket 服务端通过 TCP 协议与单车建立连接，连接过程中需要指定单车 IP 以及端口号，Socket 通过 connect()函数建立连接及校验成功后，服务端获得新的套接字，开始通过 send()函数进行数据发送，数据发送成功后通过 closesocket()函数关闭 Socket 连接。

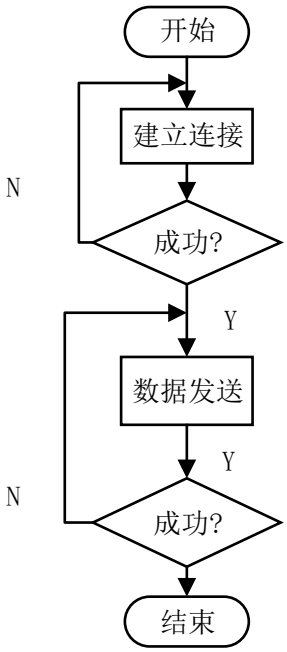


图 3-18 指令发送流程图

2．Web 端登录、注册模块

（1）UI 设计

登录与注册界面采用 Vue 编写完成，Cookie 设置方面采用 JavaScript 完成，Router 路由使用标准的 Vue-Router 进行管理[14]。界面中通过 div 完成主体的布局，通过 input 标签完成输入框的编写，button 标签实现点击按钮，按钮通过 v-on:click 绑定 login 与 register 进行事件触发完成登录与注册请求，span 标签完成登录界面与注册界面的切换，v-on:click 绑定 ToLogin 与 ToRegister 事件，点击时通过 v-show 标签实现登录界面与注册界面的 div 切换。登录界面如图 3-19 所示，注册界面如图 3-20 所示。

图 3-19 用户登录界面

图 3-20 用户注册界面

（2）算法流程

用户登录流程如图 3-21 所示，用户进入主页后首先进行授权判断，通过

geCookie 函数检查是否存在用户的登录信息 access_token,如果信息存在用户可以直接进入主页,如果 Cookie 不存在则跳转到登录界面提示用户进行登录，用户没有账号可以选择进入注册界面进行账号注册；用户注册以及登录成功后通过 setCookie 函数将 access_token 以及 username 存储到 cookie 中，设置超时时间为 2 天，同时通过 this.$router.push()进行成功页面跳转,页面跳转过程中通过 bind()设置延迟为 1 秒。

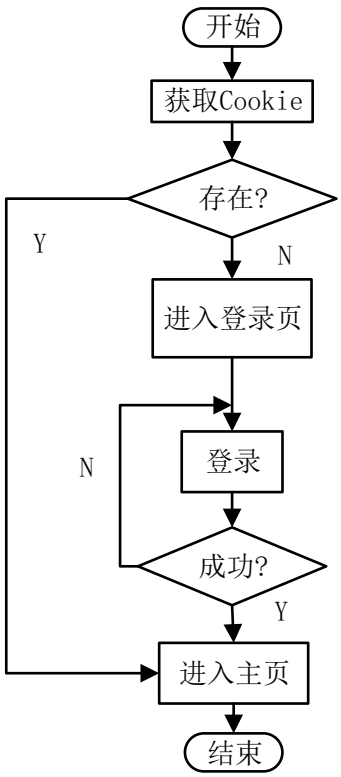


图 3-21 用户登录流程图

（3）接口服务

登录、注册服务主要包括用户登录以及注册接口。登录接口主要完成用

户的登录操作，用户登录时将密码进行 md5 加密，加密完成后检查密码与用户名是否一致，一致时则登录成功，不合法则提示用户登录错误，接口请求如表 3-16 所示。

表 3-16 用户登录接口

名称	login
说明	本接口验证用户登录合法性
输入	username：用户名 password：密码
输出	success：false 登录失败 access_token：登录成功，返回用户唯一合法标识

用户进行注册时首先需要检查用户输入信息的合法性，用户输入合法时开始进行注册操作，主要操作用户表以及用户元信息表，注册成功后跳转到登录页面提示用户进行操作。接口请求如表 3-17 所示。

表 3-17 用户注册接口

名称	register
说明	本接口用于用户注册
输入	username：用户名 password：密码
输出	access_token：注册成功，返回用户唯一合法标识

3．单车使用模块

（1）UI 设计

用户进行单车使用时需要完成定位功能，用户通过百度 Api 完成定位，

通过引入百度开开放 Js 包采用 Vue 完成地图容器及页面的渲染，在进行定位时需要获取用户经纬度信息，获取成功后存储到页面 Cookie 中，用户定位页面通过 div 完成地图容器的创建，在 div 中引入地图 id 进行地图页面的渲染；用户开始使用时需要输入单车编号进行开锁，页面使用 Vue 完成编写，界面通过 input 实现单车编号输入框，<p>标签采用 v-show 方式实现开锁成功以及失败提示，同时通过 button 采用 v-on:click 绑定 open 事件完成开锁接口的请求触发。地图定位页面如图 3-22 所示。

图 3-22 地图定位界面

（2）算法流程

单车使用流程如图 3-23 所示。用户进入定位界面使用 Marker 以及 panTo 函数完成定位后通过 setCookie 函数将当前的经纬度信息存储在 Cookie 中[15]。开始使用单车时通过 getCookie 函数从 Cookie 中取出经纬度信息以及登录信息，点击开始使用按钮后将取出的 Cookie 信息以及用户标识

access_token 放在 Body 中进行 Api 请求。

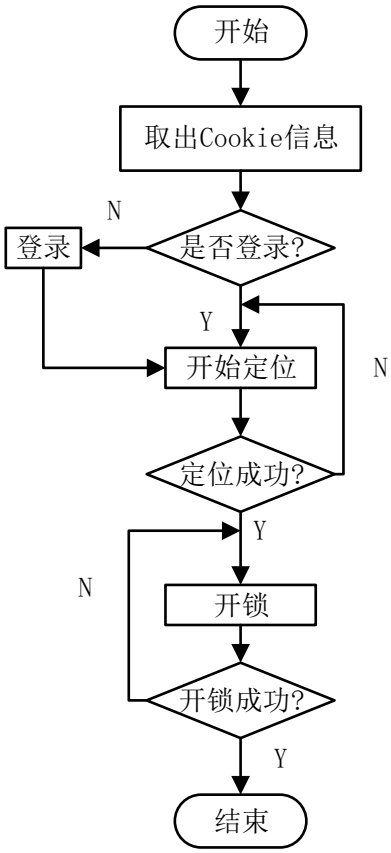


图 3-23 单车使用流程

（3）接口服务

用户使用单车时需要输入单车编号，在进行使用单车请求时需要将经纬度、用户 Id 以及经纬度等信息通过 POST 请求传递到后端服务。服务接收到请求后开始校验用户、单车合法性，校验完成后开始调用 TCP 服务完成单车开锁功能。接口请求如表 3-18 所示。

表 3-18 用户使用单车接口

名称	use_bike
说明	本接口用于用户使用单车开锁
输入	by_id：单车编号
输出	success：false 开锁失败 success：true 开锁成功

4. 充值模块

（1）UI 设计

用户在使用前可进行充值，充值界面采用 Vue 编写完成，通过 JavaScript 完成动作以及 Cookie 处理，界面采用 Input 标签完成输入框，通过<h>标签完成提示信息，button 通过 v-on:click 绑定 topup 事件完成充值 api 的请求。充值界面如图 3-24 所示。

图 3-24 充值界面

（2）算法流程

用户充值流程如图 3-25 所示。用户输入进入界面后需要检测登录状态，通过 getCookie 函数取出 access_token 用户标识信息，如果 cookie 不存在则通过 this.$router.push()跳转到登录界面；用户登录后输入充值金额以及手机号进行充值。

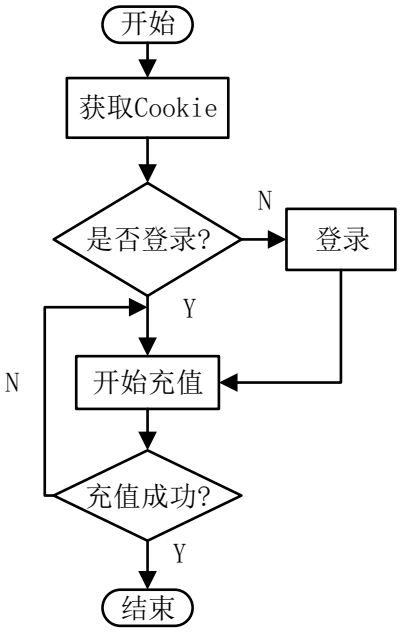


图 3-25 用户充值流程

（3）接口服务

进行充值时用户输入充值金额及手机号后通过 POST 方式开始请求接口，接口收到请求后通过 getUserByName()方法校验用户合法性，校验主要包括用户实名认证校验、手机号校验以及充值合法性校验，校验成功后通过 Insert()添加记录，同时设置订单为待处理。用户充值接口如表 3-19 所示。

表 3-19 用户充值接口

名称	topup
说明	本接口用于用户充值
输入	amount：金额 phone：手机号

续表 3-19

输出	success：false 充值失败 success：true 充值成功

5. 后台管理员添加模块

（1）UI 设计

在管理后台用户可以进行添加后台管理用户操作，页面通过 Input 标签完成添加用户信息的录入，通过 Button 配合 onClick 完成后台接口请求的事件绑定，通过 select 标签的 v-model 属性完成用户类型选择框，通过 Js 完成输入校验以及动作的处理。界面如图 3-26 所示。

图 3-6 管理员添加界面

（2）算法流程

管理员添加算法流程图如图 3-27 所示。用户进入界面后首先通过 geCookie 函数获取 isAdmin 的值，获取成功后判断是否是超级管理员用户，如果是超级管理员用户则可以继续进行操作，普通用户则不可以进行添加用

户的操作。

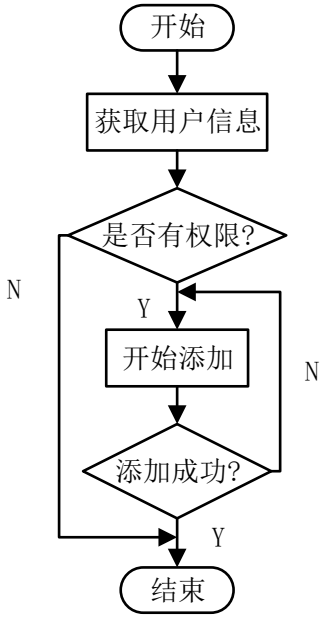


图 3-27 管理员添加算法流程图

（3）接口服务

后台接口在接收到前端添加用户请求后首先判断该用户是否具备添加权限，具备则继续进行输入合法性校验，输入校验主要为输入用户相关信息的校验，包括用户重复校验、密码合法性校验以及管理权限校验，校验成功后开始执行 Insert()完成添加管理员操作。如表 3-20 所示。

表 3-20 添加管理用户接口

名称	add_user
说明	本接口用于用户使用单车
输入	username：用户名

续表 3-20

输入	name：真实姓名 password：用户密码 is_admin：1 管理员
输出	success：false 添加失败 success：true 添加成功

6.告警信息查看

（1）UI 设计

用户进入告警界面首先需要通过 vue-resource 发起 GET 请求进行列表查询，获取数据后界面通过 V-For 标签完成列表的循环读取及渲染，采用 table 完成数据列表的绘制，通过 onClick 结合 Button 完成数据的请求交互，在列表的美化方面使用 SaCss 完成。界面如图 3-28 所示。

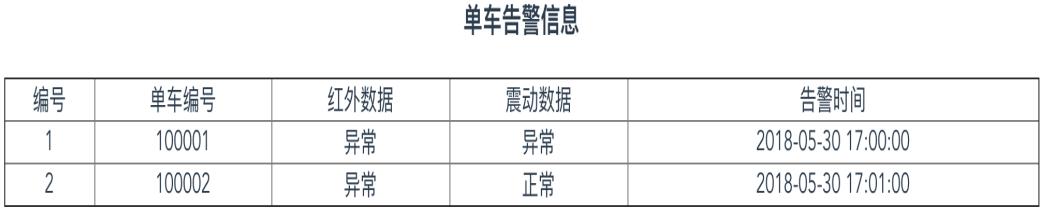


图 3-28 告警信息界面

（2）算法流程

告警信息页面算法流程图如图 3-29 所示。用户进入界面后开始发起请求获取数据列表，获取到数据列表后通过 V-For 标签进行页面数据列表的渲染，数据主要包括单车编号、告警信息、告警时间。

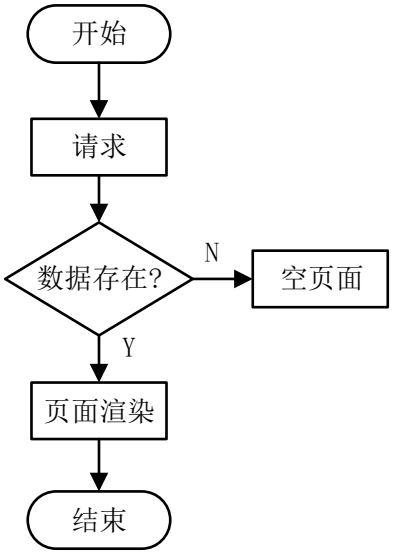


图 3-29 告警信息页面算法流程

（3）接口服务

前端界面开始请求告警信息接口时，首先对用户合法性进行校验，校验成功后开始获取最新的告警信息，获取后通过列表的形式返回给前端界面使用。如表 3-21 所示。

表 3-21 告警信息列表接口

名称	alarm_list
说明	本接口用户获取告警信息列表数据
输入	access_token：用户标识
输出	success：false 获取失败 list: [{1,1,0,0,1},{1,1,1,1,1}] 列表数据

第 4 章 系统测试

本章主要讲解设计的测试以及调试，分为感知层、传输层以及应用层三层的测试。感知层进行传感器数据采集的测试，传输层进行数据传输准确性的测试，应用层包括 TCP 服务数据收发以及 Web 端服务功能模块的测试。

4.1 感知层测试

感知层包括 MH 红外传感器与 SW 震动传感器，测试过程中主要测试传感器数据采集的准确性。测试通过串口接收工具接收传感器采集到的数据，如图 4-1 所示，串口助手显示接收到的红外传感器以及震动传感器数据，数据正常，表明感知层功能模块工作正常。

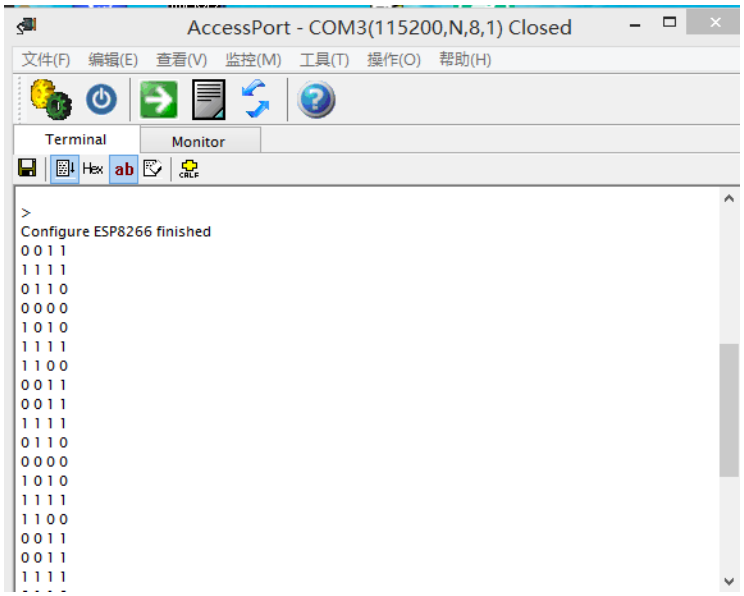


图 4-1 感知层数据接收测试图

4.2 传输层测试

设计通过 WIFI 模块完成整体数据的收发工作，传输层主要完成将感知层采集到的数据通过 WIFI 传输到服务器进行处理；在测试过程中，通过网络调试助手进行数据接收的测试工作，如图 4-2 所示，网络调试助手接收到来自传输层模块的传感器数据，并且数据正常，表明传输层数据传输正常。

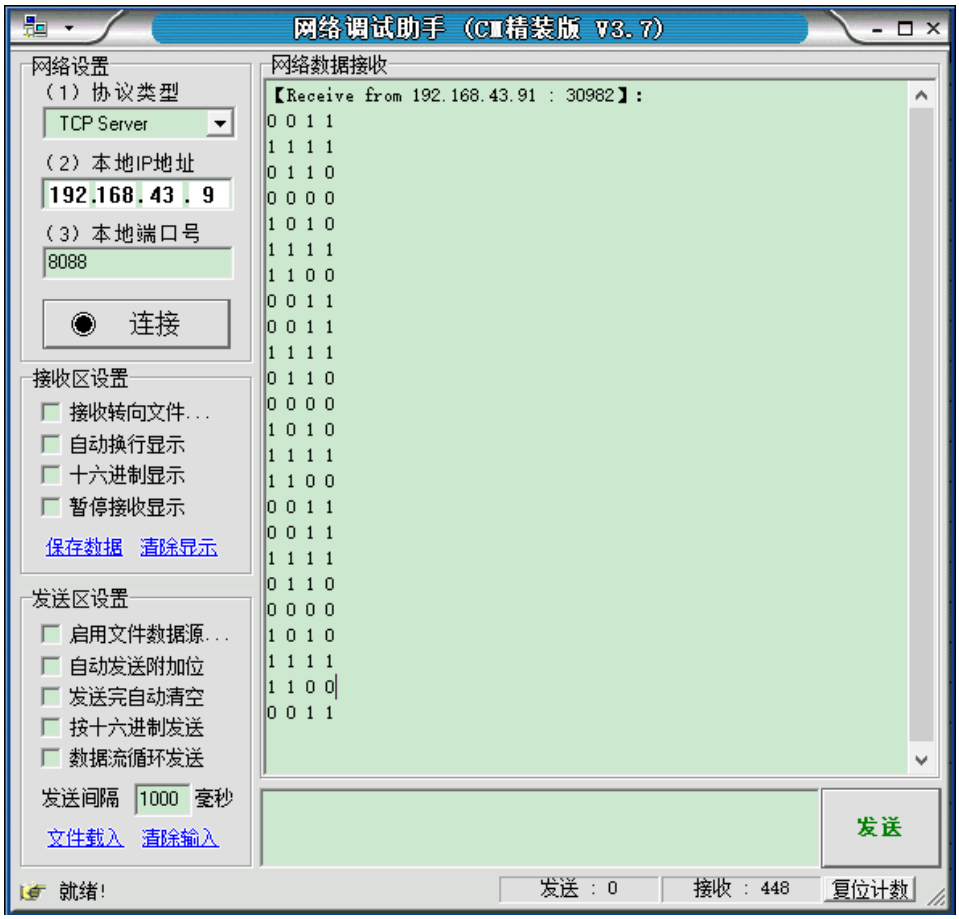


图 4-2 传输层数据传输测试图

4.3 应用层测试

4.3.1 TCP 数据收发服务功能测试

TCP 服务器端主要完成数据接收、数据存储、数据监控的功能，服务器接收来自传输层的数据，在接收过程中将异常数据存储到 MySql 数据库，同时进行数据异常性的检测，数据异常时通过发送邮件的方式完成报警；测试过程中主要测试 TCP 数据接收以及异常数据报警功能。

TCP 服务启动后自动监听指定端口，端口有新连接发现时，开始建立链接并且接收数据；如图 4-3 所示，服务发现连接后开始进行数据接收，表明 TCP 服务工作正常。

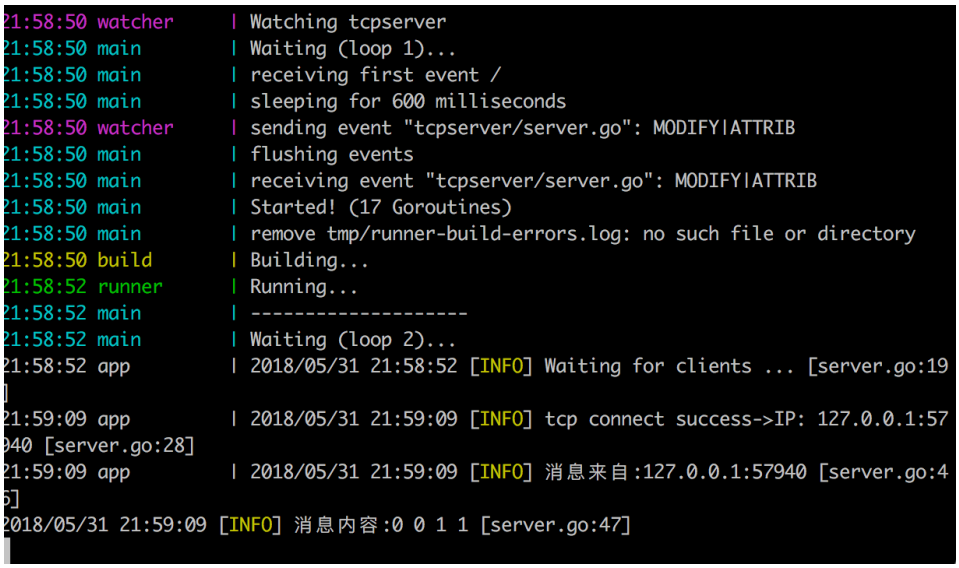


图 4-3 TCP 数据接收服务测试

TCP 服务接收到数据后对数据是否异常做校验，数据异常时通过邮件发送报警信息，同时将数据存储到数据库中；如图 4-4 所示，数据异常后发送了报警邮件，同时将异常信息存储到了数据库中，表明数据监控以及报警功能模块工作正常。


```
mysql> select *from bike_alarm;
+----+--------+----------+----------+------------+---------------------+---------------------+---------+
| id | by_id  | mh_count | sw_count | alarm_ts   | created_at          | updated_at          | deleted |
+----+--------+----------+----------+------------+---------------------+---------------------+---------+
|  1 | 100001 | 1        | 1        | 1527814901 | 2018-06-01 09:00:00 | 2018-06-01 09:00:00 |       0 |
+----+--------+----------+----------+------------+---------------------+---------------------+---------+
1 row in set (0.00 sec)
```

图 4-4 数据监控及报警测试图

4.3.2 Web 端软件功能测试

Web 端软件功能测试主要为用户使用单车的测试，用户开始使用单车时向后台请求 Api 进行开锁，Api 接收到请求后通过 TCP 服务向传输层模块发送指令完成单车开锁功能；如图 4-5 所示，后台服务接收到开锁请求后开始工作，并且数据处理正常，表明开锁功能正常。

```
2018/06/01 08:49:59 [INFO] 开始接入TCP服务... [controller.go:49]
2018/06/01 08:49:59 [INFO] 开始发送指令(1) [controller.go:50]
2018/06/01 08:49:59 [INFO] 指令发送成功 [controller.go:51]
2018/06/01 08:49:59 [INFO] 开锁成功 [controller.go:52]
2018/06/01 08:49:59 [TRACE] end access token handler [handler.go:42]
2018/06/01 08:49:59 [TRACE] response: {"result":null,"success":true} [logger.go:86]
2018/06/01 08:49:59 [INFO] "200" "955.311µs" "127.0.0.1" "POST" "127.0.0.1:8890" "/bike
/use/v1/start_use" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36" [logger.go:101]
```

图 4-5 用户使用单车功能测试图

结　论

系统采用物联网标准架构完成整体的设计。在感知层通过 Stm32 对 MH 红外传感器、震动传感器的控制实现单车状态以及单车安全状态的检测；传输层采用ESP8266通过WIFI技术完成感知层采集数据的实时传输以及单片机指令的收发，主要完成了单车开锁以及安全状态信息上传的功能；应用层通过 Golang 编写 TCP 收发服务，完成对传输层数据的接收以及异常判断，通过完成与 DB 的交互进行数据存储，在数据存储方面使用主流的 MySql，同时为了提高查询速度，系统使用 Redis 作为 DB 数据缓存，在进行数据查询以及更新时能够极大的提高效率。通过设计开发，设计目前具备如下功能：

（1）感知层通过红外传感器可检测到单车的上锁状态，同时通过震动传感器完成单车安全状态的检查；在单车没有使用时，单车产生震动后即开始将数据发送到服务器，服务器接收到数据后再次进行异常判断，数据判断异常后可自动向工作人员发送邮件进行报警，同时将数据存储到数据库中，可以实时查看历史告警信息。

（2）设计通过 Golang 语言完成 TCP 收发服务的编写，服务主要用于自动发现连接，服务发现后自动建立连接开始进行数据的传输，数据传输主要包括传输层数据的接收以及开关锁指令的发送。

（3）设计完成共享单车 Web 端服务，通过该服务用户可完成使用共享单车的一整个流程。用户完成登录进入主页后系统将获取用户的位置信息，包括经度以及纬度信息,定位成功后用户可以进入使用界面开始使用共享单车，使用时通过输入单车编号进行开锁，开锁完成后即可开始进行使用。

设计完成了共享单车从感知层数据监测到传输层数据发送以及应用层用户使用的整个流程。但设计还是存在一些缺陷，未实现扫码开锁的功能，操作还不是很友好,支付方面由于权限限制没有接入流行的微信、支付宝支付。

致　谢

。

参考文献

1 蒙博宇. STM32 自学笔记[M]. 北京航空航天大学出版社, 2013:45-55.
2 袁欢. 基于计算机视觉技术的零件自动分检系统[D]. 浙江理工大学, 2015:09-20.
3 赵铖. 超密集 WIFI 组网的干扰协调研究[D]. 中国科学院大学, 2016:12-15.
4 王鹏雁. 斜盘式轴向柱塞泵的单片机控制研究[D]. 兰州理工大学, 2013:15-20.
5 张逢雪, 王香婷, 王通生,等. 基于 STM32 单片机的无线智能家居控制系统[J]. 自动化技术与应用, 2014, 30(8):98-101.
6 杨卫东, 邓冠群, 张国平,等. 基于 STM32 单片机的库房安全远程控制系统[J]. 电子测量技术, 2015(8):94-98.
7 李艳红, 李自成, 孙仕琪. 基于 STM32 单片机的金属物体探测定位器系统的设计与实现[J]. 仪表技术与传感器, 2016(4):63-66.
8 刘波. 基于 TCP/IP 协议栈的智能网关模块的研究与开发[D]. 新疆大学, 2013:3-15.
9 孟迪. 基于表达谱分析的肿瘤 miRNA 调控机制研究[D]. 吉林大学, 2014:09-15.
10 宋艳霞. 基于 ZigBee 的智能照明设备控制系统设计[D]. 中北大学, 2013:10-18.
11 许式伟, 吕桂华. Go 语言编程[M]. 人民邮电出版社, 2013:198-220.
12 黄瑞阳. 地图符号共享关键技术研究[D]. 解放军信息工程大学, 2013:01-12.
13 周亮, 姜胜明, 熊晨霖. Research on Semi-TCP for Ship Ad-hoc Network[J]. 计算机工程, 2018, 44(2):119-123.
14 刘亚楠. 共享单车发展研究分析[J]. 时代金融, 2017(8).
15 伍耀常, 赵利, 王阳明. 基于 WiFi 组网的通用智能车载终端设计[J]. 桂林

电子科技大学学报, 2016, 36(5):401-405.
16 Xing Y, Li-Wei L I, Zhang H W. Design of CAN Bus Analyzer Based on STM32F103 MCU[J]. Journal of Qingdao University, 2013.
17 Excoffier L, Lischer H. Arlequin suite ver 3.5: a new series of programs to perform population genetics analyses under Linux and Windows[J]. Molecular Ecology Resources, 2013, 10(3):564.
18 Margolang A K. AnalisisPerbandinganProtokol Better Approach To Mobile Ad Hoc Network (BATMAN) DenganProtokol Babel UntukLayanan Voice Over Internet Protocol (VOIP) Pada Mobile Ad Hoc Network (MANET)[J]. Air Line Pilot, 2014.
19 Ameur C B, Mory E, Cousin B, et al. Performance evaluation of TcpHas: TCP for HTTP adaptive streaming[J]. Multimedia Systems, 2018(2):1-18.
20 Hull S. DRBD and MySQL - An HA Match Made In Heaven - O'Reilly Media Free, Live Events[J]. 2018:12-23.

附录 设计系统部分源代码

TCP 服务 Sever 部分程序代码如下：

```
func DoTcpServer() {
    netListen, err := net.Listen("tcp", "localhost:8891")
    if err != nil {
        log.Errorf("listent port err->%v", err)
    }
    defer netListen.Close()
    for {
        conn, err := netListen.Accept()
        if err != nil {continue}
        log.Infof("tcp connect success->IP: %s", conn.RemoteAddr().String())
        go handleConnection(conn)
    }
}
```

Js 设置 Cookie 部分代码：

```
export function setCookie(c_name, value, expire) {
    var date = new Date()
    date.setSeconds(date.getSeconds() + expire)
    document.cookie = c_name + "=" + escape(value) + "; expires=" +
date.toGMTString()
 }
export function getCookie(c_name) {
    if (document.cookie.length > 0) {
```

```
            let c_start = document.cookie.indexOf(c_name + "=")
            if (c_start != -1) {
                c_start = c_start + c_name.length + 1
                let c_end = document.cookie.indexOf(";", c_start)
                if (c_end == -1) c_end = document.cookie.length
                return unescape(document.cookie.substring(c_start, c_end))
            }
        }
        return ""
}
export function delCookie(c_name) {
        setCookie(c_name, "", -1)
}
```

ESP8266 端口配置部分代码：

```
static void ESP8266_GPIO_Config ( void )
{
    GPIO_InitTypeDef GPIO_InitStructure;
    macESP8266_CH_PD_APBxClock_FUN    (    macESP8266_CH_PD_CLK,
ENABLE );
    GPIO_InitStructure.GPIO_Pin = macESP8266_CH_PD_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init ( macESP8266_CH_PD_PORT, & GPIO_InitStructure );
    macESP8266_RST_APBxClock_FUN        (        macESP8266_RST_CLK,
ENABLE );
    GPIO_InitStructure.GPIO_Pin = macESP8266_RST_PIN;
    GPIO_Init ( macESP8266_RST_PORT, & GPIO_InitStructure );
}
```