|  |
| --- |
|  |
| Linux网络编程软件设计 |
| 制作人：封收 时间：2022/8/26 |

|  |
| --- |
|  |

# 目录

[目录 1](#_Toc112597396)

[1. 引言 2](#_Toc112597397)

[1.1 目的 2](#_Toc112597398)

[1.2 缩略语句 2](#_Toc112597399)

[2. 综合描述 2](#_Toc112597400)

[2.1 项目功能 2](#_Toc112597401)

[3. 软件系统架构设计 2](#_Toc112597402)

[3.1 软件框架设计理念 2](#_Toc112597403)

[3.2 软件框架层次说明 2](#_Toc112597404)

[4. app设计方案 4](#_Toc112597405)

[4.1 功能简介 4](#_Toc112597406)

[4.2 输入输出设计 4](#_Toc112597407)

[4.3 处理逻辑说明 5](#_Toc112597408)

[5. 模块方案设计 5](#_Toc112597409)

[5.1 Led\_app模块 5](#_Toc112597410)

[5.2 Uart\_app模块 8](#_Toc112597411)

[5.3 Inet\_app模块 11](#_Toc112597412)

[6. 工具方案设计 16](#_Toc112597413)

[6.1 hup工具 16](#_Toc112597414)

[6.2 ring\_fifo工具 18](#_Toc112597415)

[6.3 message\_queue工具 20](#_Toc112597416)

# 引言

## 目的

本文档用于设计和实现Linux网络编程的软件框架和模块，为Linux网络编程的软件开发提供文档支持，逐渐向统一的软件开发规范发展。

## 缩略语句

|  |  |  |
| --- | --- | --- |
| **缩略语** | **英文原文** | **中文含义** |
| r\_fifo | Uart\_read\_ring\_fifo | 接受串口数据的环形缓存区 |
| w\_fifo | Uart\_write\_ring\_fifo | 写入串口数据的环形缓存区 |
| cfifo | Client\_send\_ring\_fifo | 客户端数据发送服务端的环形缓存区 |
| sfifo | Server\_recive\_ring\_fifo | 服务端接受客户端数据的环形缓存区 |
|  |  |  |
|  |  |  |

# 综合描述

## 项目功能

该项目是音频组嵌入式的Linux阶段的任务，实现的功能有串口数据打包解包收发、TCP客户端和服务器本地通讯，HIP协议的封包解包，客户端断线重连，双方心跳检测及串口数据透传。

# 软件系统架构设计

## 软件框架设计理念

本次软件设计中，最首要的是能写出多个可供上层调用的串口、网络模块。在原先成功实现了串口通信模块的基础上进一步添加网络模块，继承之前实现串口模块的思想进行编写网络模块。

## 软件框架层次说明

* APP层：

对事件进行处理、判断、发送命令给下层模块。

* 模块层：

抽象功能的实现，通过整合驱动模块实现APP的命令执行以及上报事件。

* 工具层：

主要实现通用功能的工具，提供给APP层和模块层使用。

# app设计方案

APP层是整个系统控制中心，除了阻塞接收消息队列的数据，需要根据具体功能要求采用状态机进行管理，分有四种

* + Led
  + Uart
  + Inet
  + Shutdown

## 功能简介

* Led:

此状态下，APP模块负责分发从开发板串口经由hup解包后到达消息队列的命令，给到led\_app模块的函数，其进行下一步的控灯。

* Uart:

此状态下，APP模块负责分发从开发板串口经由hup解包后到达消息队列的命令，给到uart\_app模块的函数，其进行串口的写入操作。

* Inet:

此状态下，

服务端：APP模块分发命令给其，再转发客户端；只需接受其上报的消息；

客户端：APP模块负责分发从开发板串口经由hup解包后到达消息队列的命令；

给到inet模块的函数，其进行tcp服务端和客户端通信。

* Stop:

停止所有start的模块，退出程序。

## 输入输出设计

* 输入事件

消息队列结构体数据存在可消费的数据；

* 内部处理

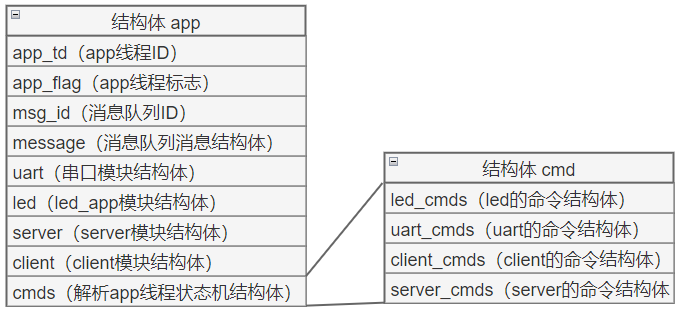
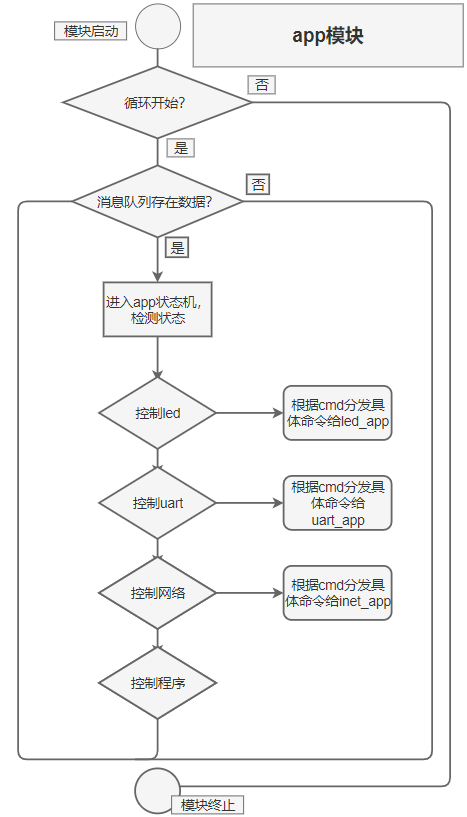
根据命令进行命令分流至四种状态机；

状态机内四种状态的不同处理；

* 输出事件

发送命令或者分发数据给下层；

## 处理逻辑说明



## 接口设计

* start

int app\_start(app\_p app);

|  |  |
| --- | --- |
| 功能描述 | 启动app线程，开始接受并处理消息队列消息， |
| 参数 | app\_p：app结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* stop

int app\_stop(app\_p app);

|  |  |
| --- | --- |
| 功能描述 | 终止app线程的运行 |
| 参数 | app\_p：app结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* process

void app\_process(app\_p app);

|  |  |
| --- | --- |
| 功能描述 | App模块处理信息中心，状态机 |
| 参数 | app\_p：app结构体指针 |
| 返回值 | void |

# 模块方案设计

## Led\_app模块

### 功能简介

负责控制led亮灭或者呼吸

### 输入输出设计

* 输入条件

APP层根据消息队列内的数据解析结果确定的具体控制参数，如哪几个灯亮，灯是什么颜色，每个灯是什么模式；

* 内部处理

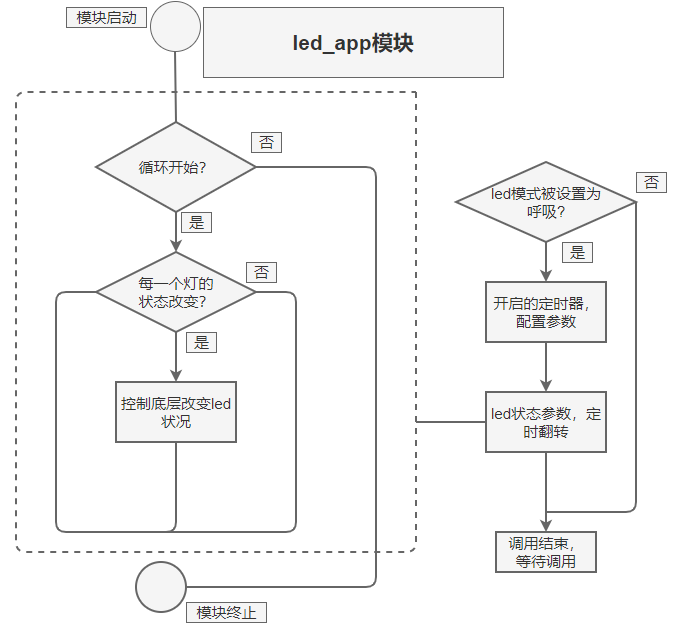
创建led\_thread\_entry处理线程，几乎不间歇地设定led数组中每个灯的亮或灭；

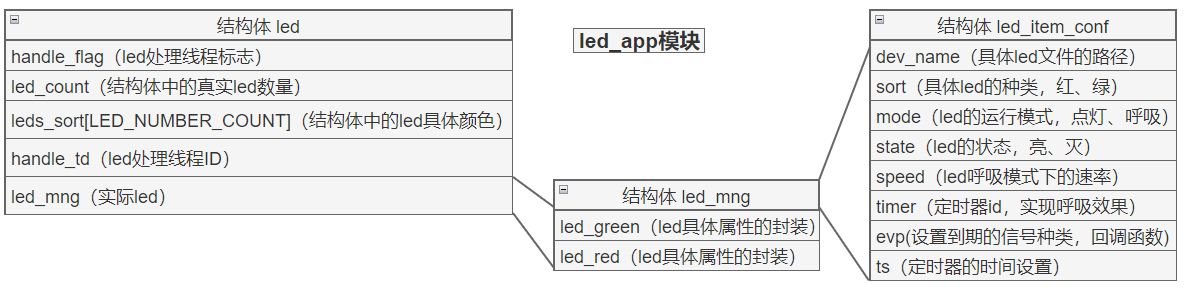
如有令led呼吸的需求，led\_conf()函数设置定时器，根据APP分发数据控制呼吸速度；

* 输出事件

处理完毕后，打印led操作成功字样；

### 处理逻辑说明





### 接口设计

* init

int app\_led\_init(led\_p led, int msg\_id)

|  |  |
| --- | --- |
| 功能描述 | 初始化LED驱动 |
| 参数 | led\_p：Led结构体指针 |
| 参数 | msg\_id：消息队列id |
| 返回值 | 0：成功  -1：失败 |

* deinit

int app\_led\_deinit(led\_p led);

|  |  |
| --- | --- |
| 功能描述 | 去初始化LED驱动 |
| 参数 | led\_p：Led结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* start

int app\_led\_start(led\_p led);

|  |  |
| --- | --- |
| 功能描述 | 启动LED驱动 |
| 参数 | led\_p：Led结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* stop

int app\_led\_stop(led\_p led);

|  |  |
| --- | --- |
| 功能描述 | 终止LED驱动 |
| 参数 | led\_p：Led结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* set

int app\_led\_set(led\_p led);

|  |  |
| --- | --- |
| 功能描述 | 设置所有led结构体中各led的状态（亮或者灭） |
| 参数 | led\_p：Led结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* conf

int app\_led\_conf(led\_p led, led\_num\_em led\_sort[], led\_cmd\_mode\_em led\_mode[], int conf\_val[], int ctrl\_num);

|  |  |
| --- | --- |
| 功能描述 | 具体配置led结构体中的各led状态 |
| 参数 | led\_p ：Led结构体指针 |
| 参数 | led\_num\_em[]：控制Led数组的具体个体选择 |
| 参数 | led\_cmd\_mode\_em[]：控制Led模式选择（点灯或呼吸） |
| 参数 | conf\_val[]：控制led的具体参数基数值（亮灭，快慢） |
| 参数 | ctrl\_num：本次调用函数的具体控灯数量 |
| 返回值 | 0：成功  -1：失败 |

* set\_cmd

int set\_led\_cmd(uint8\_t cmd, led\_cmd\_p led\_p)

|  |  |
| --- | --- |
| 功能描述 | 设置所有led结构体中各led的状态（亮或者灭） |
| 参数 | cmd：app分发的命令 |
| 参数 | led\_cmd\_p：Led关于命令的结构体指针 |
| 返回值 | 0：成功  -1：失败 |

## Uart\_app模块

### 功能简介

负责监听开发板串口数据读事件，读取和解析后上报消息队列；

成功接收串口数据后收集负载数据hup打包发往串口应答；

### 输入输出设计

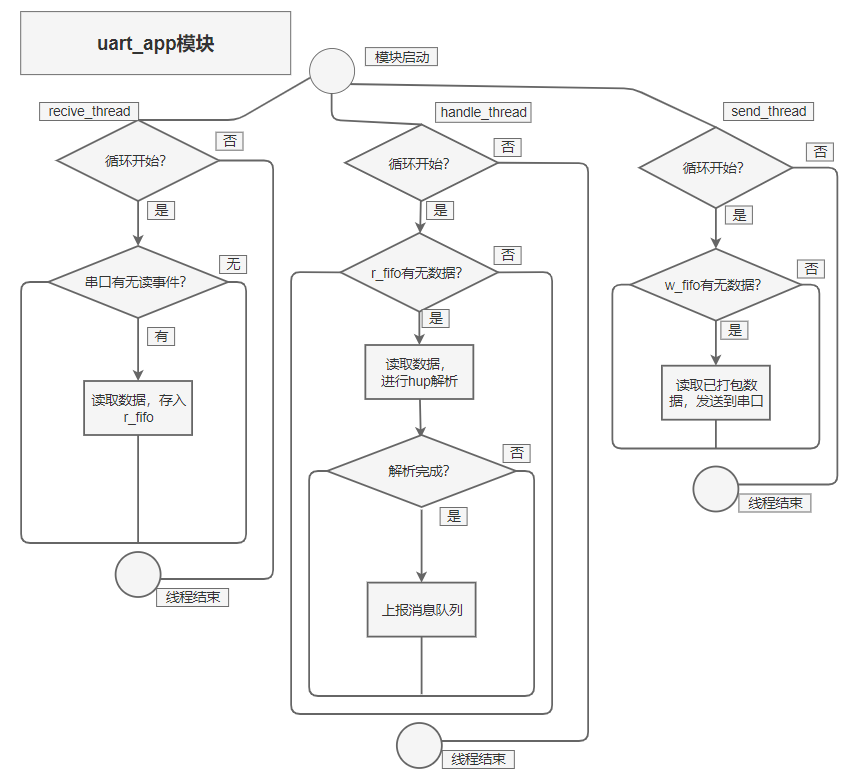
* 输入条件

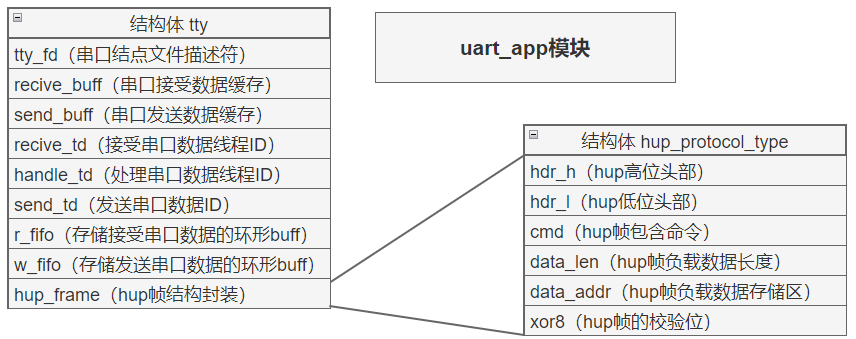
几乎不间歇地监听开发板串口的读事件，一旦可读，立马接收数据。

* 内部处理
* 创建三个线程，分别对事务进行分解承担；两个FIFO，用于存放读入和发出的数据；
* Recive\_thread，监听开发板上的串口读事件，读取到的数据存放入r\_fifo；
* Handle\_thread，读取r\_fifo，解析数据，上报消息队列；
* Send\_thread，读取w\_fifo，直发给串口；
* 输出事件

处理完毕后，消息队列获取到串口的负载数据和cmd;

### 处理逻辑说明





### 接口设计

* init

int tty\_init(tty\_p tty\_pointer, int msg\_id);

|  |  |
| --- | --- |
| 功能描述 | 初始化URAT驱动 |
| 参数 | tty\_p： uart结构体指针 |
| 参数 | msg\_id：消息队列id |
| 返回值 | 0：成功  -1：失败 |

* deinit

int tty\_deinit(tty\_p tty\_pointer);

|  |  |
| --- | --- |
| 功能描述 | 去初始化UART驱动 |
| 参数 | tty\_p： uart结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* start

int tty\_start(tty\_p uart);

|  |  |
| --- | --- |
| 功能描述 | 启动UART驱动 |
| 参数 | tty\_p： uart结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* stop

int tty\_stop(tty\_p uart);

|  |  |
| --- | --- |
| 功能描述 | 终止UART驱动 |
| 参数 | tty\_p： uart结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* send

int tty\_send(tty\_p uart, hup\_protocol\_type\_em sort, int cmd, uint8\_t\* addr, int addr\_len)

|  |  |
| --- | --- |
| 功能描述 | 写入数据到UART结点文件 |
| 参数 | tty\_p： 串口结构体指针 |
| 参数 | sort： 需求发送到串口设备的hup帧类型 |
| 参数 | cmd：需求发送到串口设备的hup帧命令 |
| 参数 | addr：需求发送到串口设备的hup帧数据 |
| 参数 | addr\_len：需求发送到串口设备的数据长度 |
| 返回值 | 0：成功  -1：失败 |

## Inet\_app模块

### 功能简介

分为两部分TCP的服务端和客户端

* 服务端：
* 接收客户端连接、数据（HIP封装）及断线重连；
* 心跳保持；
* 建立客户端信息表存储信息；
* 解析HIP后送入消息队列；
* 串口数据透传；
* 客户端：
* 建立与服务端的连接；
* 获取APP分发的数据（串口负载）HIP打包发送给服务端；
* 保持心跳；
* 接收服务端发来的数据（HIP负载）上报消息队列；
* 串口数据透传；

### 输入输出设计

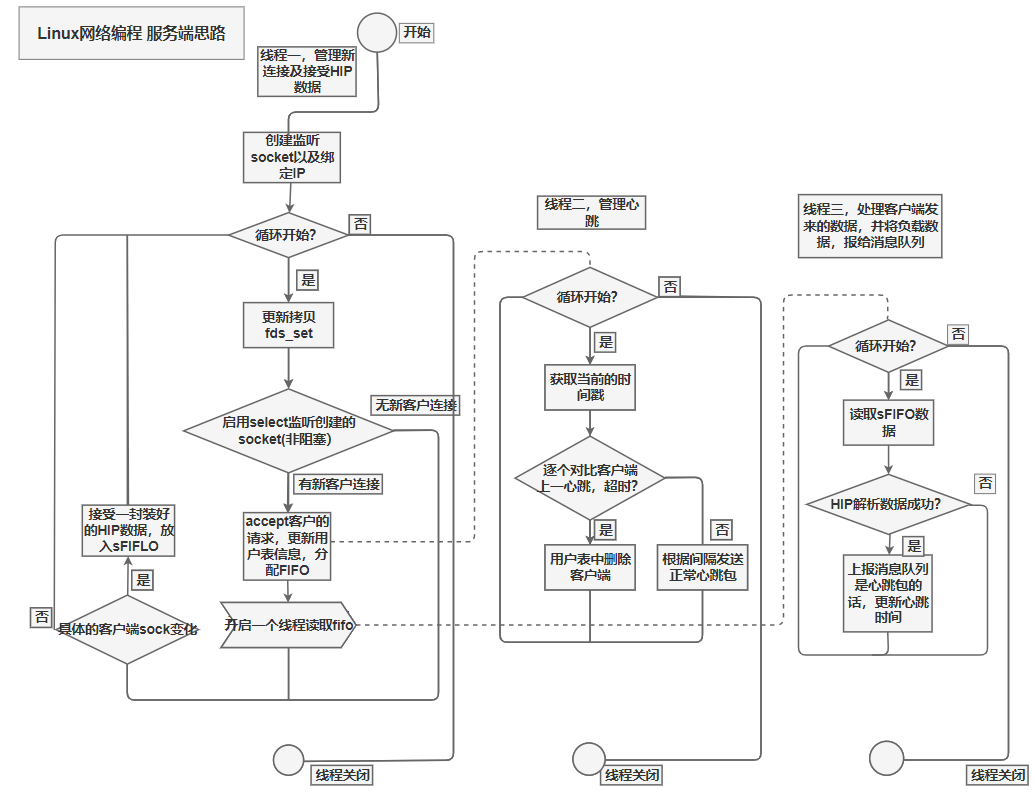
* 输入条件
* 服务端：接收客户端从app消息队列分发再经过HIP打包后的数据；
* 客户端：由app消息队列分发数据（串口负载）和服务端发送过来的数据（HIP负载）；
* 内部处理
* 服务端：
  + 创建三个线程，分别对事务进行分解承担；若干sfifo，用于存放不同客户端发来的数据；
* Mng\_thread，管理新连接和客户端发来的数据（HIP封装）；服务端循环监听客户端的请求连接，分辨其IP和端口号，如果之前连接过其IP与port，将对应IP、port与套接字加入结构体数组中，关闭已连接的收到的套接字；不然为新客户端申请资源，准备通信。循环监听所有在线的客户端的套接字，读取客户端发过来的数据，通过其套接字和数据长度，数据写入对应s\_fifo中。
* Keep\_thread，负责维护与各个客户端的心跳；心跳检测，循环检测，对每一个在线的客户端的最后一次接收其消息的时间，进行超时检测，超时就进行清理，已连接客户端数量减少。
* Commit\_thread，读取每个sfifo，HIP解析后上报消息队列；读取sfifo的数据，并进行HIP解析，解析出来的HIP负载数据上报消息队列。
* 客户端
* 创建三个线程，对事务进行分解；一个cfifo，用于存放服务端发来的数据（HIP负载）；
* Recive\_serve\_thread，负责连接服务端，监听服务端的数据（HIP负载），存入cfifo；主动发起连接，连接成功后，循环监听服务端的数据发送，接收数据存入cfifo后继续监听。
* Keep\_thread，负责维护与服务端的心跳，断线重连；检查当前时间戳与约定发送时间差值，大于既定值时（如3秒），发送心跳包，再若客户端接受服务端数据时间戳与当前时间戳相差过大（超过keep\_alive\_time），客户端断开服务器的连接。准备发起断线重连，之后发送登录消息，有回应就置为已连接状态。
* Commit\_thread，读取cfifo，数据（HIP负载）上报消息队列；几乎不断地读取cfifo中的数据，再将读取的数据上报消息队列。
* 输出事件

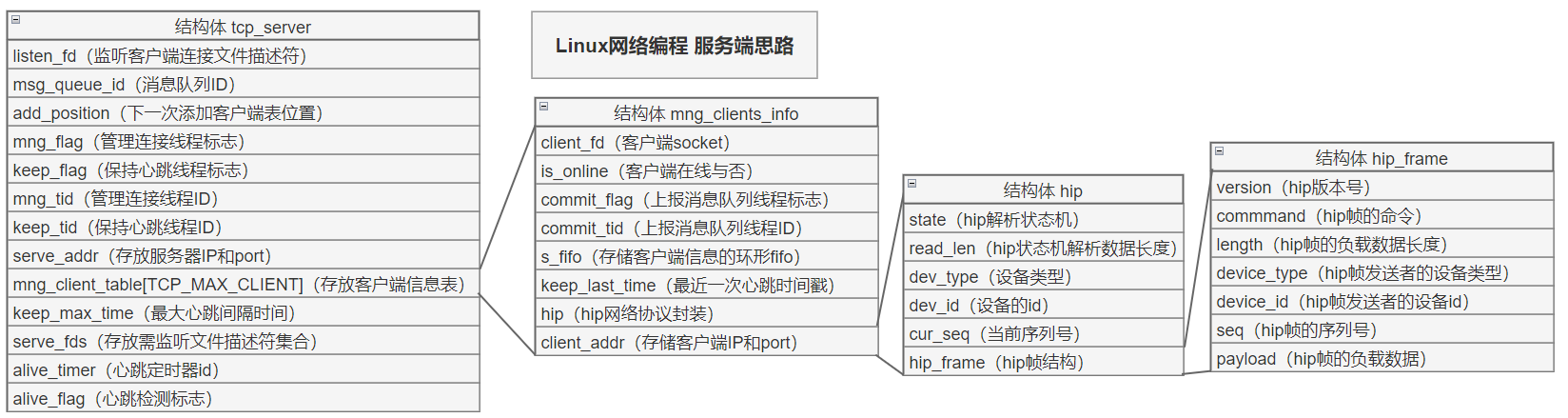
处理完毕后，

* 服务端的消息队列获取到客户端发来HIP封装的数据；
* 而客户端的消息队列获取到服务端发来HIP封装的数据;

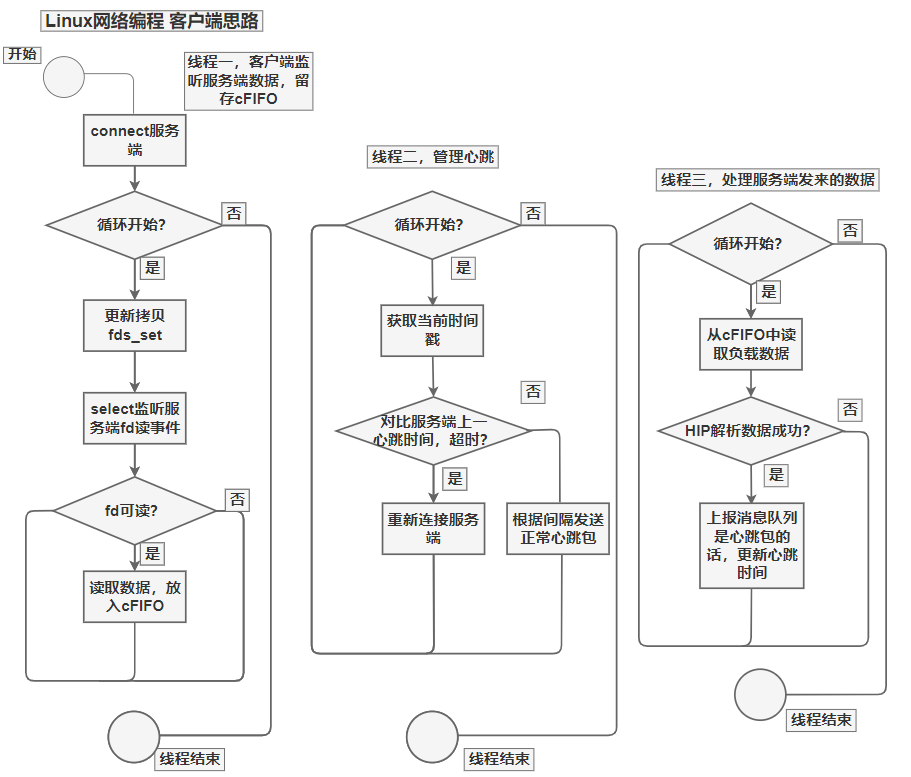
### 处理逻辑说明

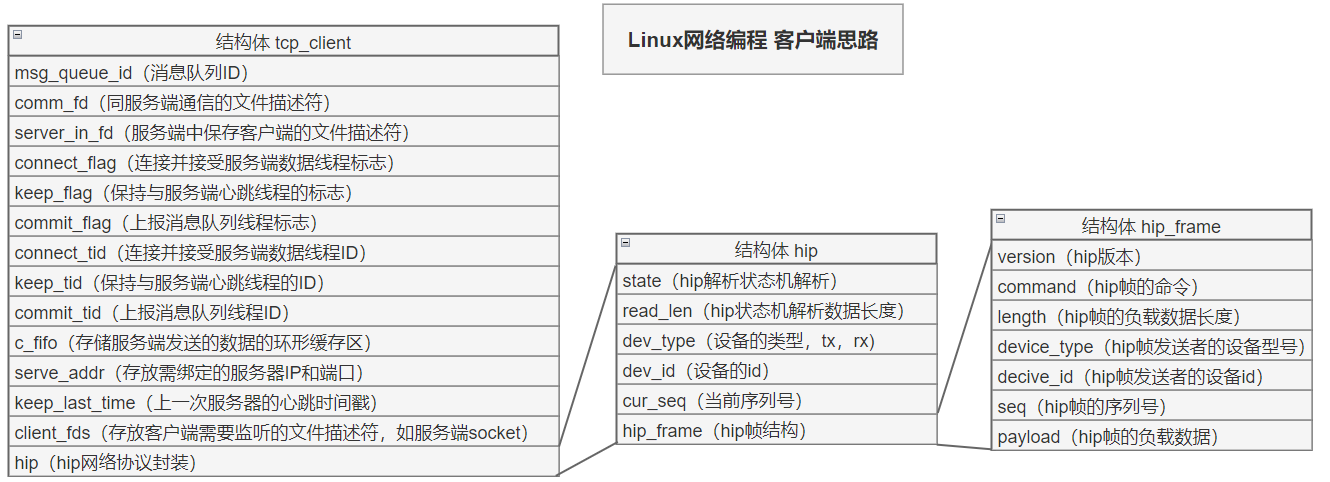
* 服务端





* 客户端





### 接口设计

* 服务端：
* init

int tcp\_serve\_init(tcp\_serve\_p server, int msg\_id)

|  |  |
| --- | --- |
| 功能描述 | 初始化SERVER |
| 参数 | tcp\_server\_p：SERVER结构体指针 |
| 参数 | msg\_id：消息队列id |
| 返回值 | 0：成功  -1：失败 |

* deinit

int tcp\_serve\_deinit(tcp\_server\_p server)

|  |  |
| --- | --- |
| 功能描述 | 去初始化SERVER |
| 参数 | tcp\_server\_p：SERVER结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* start

int tcp\_serve\_start(tcp\_server\_p server)

|  |  |
| --- | --- |
| 功能描述 | 启动SERVER |
| 参数 | tcp\_server\_p：SERVER结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* stop

int tcp\_serve\_stop(tcp\_server\_p server)

|  |  |
| --- | --- |
| 功能描述 | 终止SERVER |
| 参数 | tcp\_server\_p：SERVER结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* send

int tcp\_server\_send(tcp\_server\_p server, int sock\_fd, hip\_protocol\_cmd\_em cmd, uint8\_t\* buf, uint16\_t buf\_len)

|  |  |
| --- | --- |
| 功能描述 | SERVER向指定客户端发送数据 |
| 参数 | tcp\_server\_p：SERVER结构体指针 |
| 参数 | sock\_fd：客户端的socket |
| 参数 | cmd：发送客户端命令 |
| 参数 | buf：需求发送的数据指针 |
| 参数 | buf\_len：需求发送数据的长度 |
| 返回值 | 0：成功  -1：失败 |

❖ set\_cmd

int set\_tcp\_server\_cmd(uint8\_t cmd, uint8\_t \*data, tcp\_server\_cmd\_p inet\_p)

|  |  |
| --- | --- |
| 功能描述 | SERVER接收到app分发的数据后，做出具体应答 |
| 参数 | cmd：app分发数据中的指令 |
| 参数 | data：app分发的负载数据 |
| 参数 | Inet\_p：SERVER关于命令的结构体 |
| 返回值 | 0：成功  -1：失败 |

* 客户端：
* init

int tcp\_client\_init(tcp\_client\_p client, int msg\_id)

|  |  |
| --- | --- |
| 功能描述 | 初始化CLIENT |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 参数 | msg\_id：消息队列id |
| 返回值 | 0：成功  -1：失败 |

* deinit

int tcp\_client\_deinit(tcp\_client\_p client)

|  |  |
| --- | --- |
| 功能描述 | 去初始化CLIENT |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* start

int tcp\_client\_start(tcp\_client\_p client)

|  |  |
| --- | --- |
| 功能描述 | 启动CLIENT |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* stop

int tcp\_client\_stop(tcp\_client\_p client)

|  |  |
| --- | --- |
| 功能描述 | 终止CLIENT |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 返回值 | 0：成功  -1：失败 |

❖ send

int tcp\_client\_send(tcp\_client\_p client, int sock\_fd, hip\_protocol\_cmd\_em cmd, uint8\_t\* buf, uint16\_t buf\_len)

|  |  |
| --- | --- |
| 功能描述 | CLIENT向指定客户端发送数据 |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 参数 | sock\_fd：服务端的socket |
| 参数 | cmd：发送服务端的命令 |
| 参数 | buf：需求发送的数据的指针 |
| 参数 | buf\_len：需求发送数据的长度 |
| 返回值 | 0：成功  -1：失败 |

❖ set\_cmd

int set\_tcp\_client\_cmd(uint8\_t cmd, uint8\_t \*data, tcp\_client\_cmd\_p inet\_p)

|  |  |
| --- | --- |
| 功能描述 | CLIENT接收到app分发的数据，做出具体应答 |
| 参数 | tcp\_client\_p：CLIENT结构体指针 |
| 参数 | cmd：app分发数据中的指令 |
| 参数 | data：app分发的负载数据 |
| 参数 | Inet\_p：CLIENT关于命令的结构体 |
| 返回值 | 0：成功  -1：失败 |

# 工具方案设计

## hup工具

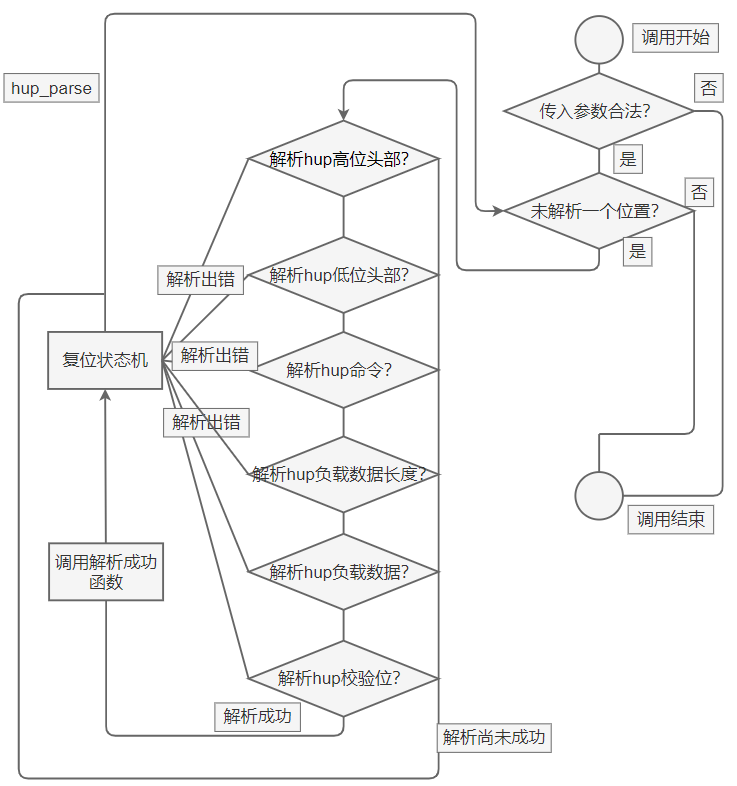
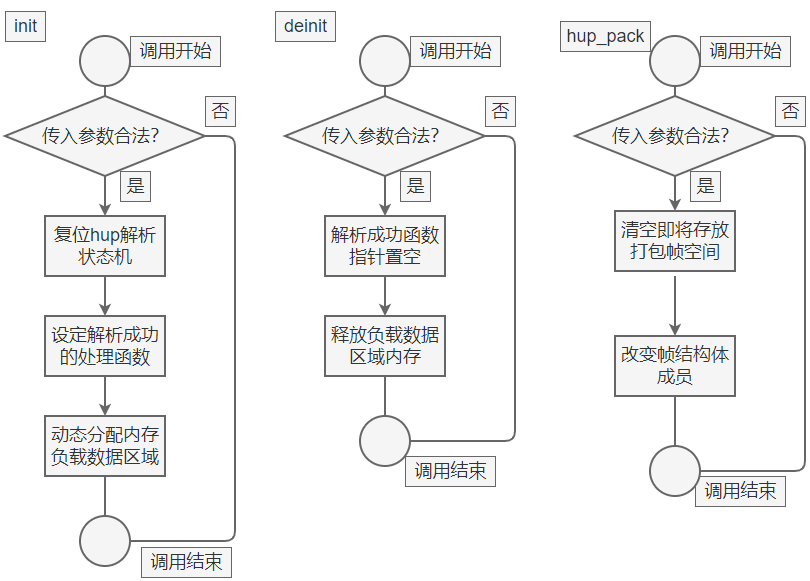
### 功能简介

HUP全称为Hollyland Uart Protocol（串口通信协议），作为公司内部使用的串口数据传输协议。对需要通过串口进行收发的数据进行相应的解包和打包操作，每一包收发的数据都做相应的校验，很大程度上提高了串口传输数据的可靠性。

### 功能设计

* 输入
* 上层需要通过串口设备结点文件发送出去的数据；
* 从串口设备结点文件中读取到的数据；
* 内部处理
* 对数据进行协议打包，添加协议头部、控制命令、校验位等；
* 对数据帧进行协议解包，检验数据帧的正确性，并从完整数据帧中取出负载数据和命令；
* 输出
* 通过hup协议封包好的数据帧；
* 经过协议解包后的负载数据和命令；

### 统一建模语言（UML）



### 接口设计

* init

int hup\_init(hup\_protocol\_type\_p hup\_p, int buf\_size, void (\*hup\_success\_handle\_func)(hup\_protocol\_type\_p));

|  |  |
| --- | --- |
| 功能描述 | 初始化HUP |
| 参数 | hup\_protocol\_type\_p：UART结构体指针 |
| 参数 | buf\_size：暂存大小 |
| 参数 | (\*hup\_success\_handle\_func)(hup\_protocol\_type\_p)：解析成功函数指针 |
| 返回值 | 0：成功  -1：失败 |

* deinit

int hup\_deinit(hup\_protocol\_type\_p hup\_p);

|  |  |
| --- | --- |
| 功能描述 | 去初始化UART |
| 参数 | hup\_protocol\_type\_p：UART结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* hup\_pack

hup\_protocol\_type\_p hup\_pack(hup\_protocol\_type\_em sort, int cmd, int len, char\* addr);

|  |  |
| --- | --- |
| 功能描述 | 根据具体的帧类型和数据，打包帧 |
| 参数 | sort：需求打包帧的类型，响应帧或者请求帧 |
| 参数 | cmd：需求打包帧的指令 |
| 参数 | len：需求打包帧的负载数据长度 |
| 参数 | addr：需求打包帧的负载数据 |
| 返回值 | 打包好的帧指针：成功  NULL：失败 |

* hup\_parse

int hup\_parse(hup\_protocol\_type\_p hup\_frame, int data);

|  |  |
| --- | --- |
| 功能描述 | 解析hup帧协议后，调用init（）时注册的回调函数 |
| 参数 | hup\_protocol\_type\_p：UART结构体指针 |
| 参数 | data：帧结构的数据之一，FIFO读取的字节数据 |
| 返回值 | 0：成功  -1：失败 |

## hip工具

### 功能简介

HIP全称为Hollyland Internet Protocol ，HIP报文结构由固定字段和扩展字段组成，固定字段包含命令类型、报文长度、设备类型、通道号、设备ID 、ntp时间和对应的rtp时间功能设计。对需要通过网络进行收发的数据进行相应的解包和打包操作，每一包收发的数据都做相应的校验，很大程度上提高了网络传输数据的可靠性。

### 功能设计

* 输入

­ 上层需要通过网络模块发送需要打包出去的数据；

­ 从其他地方输入需要解析中的数据包；

* 内部处理

­ 对数据进行协议打包，添加协议头部、控制命令、设备类型、设备ID等；

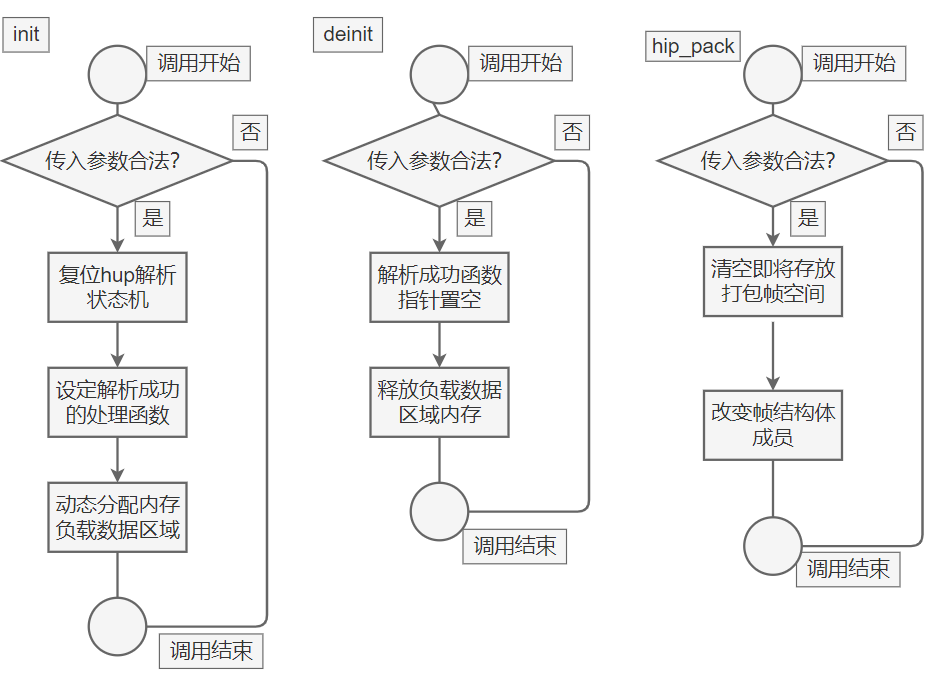
­ 对数据帧进行协议解包，检验数据帧的正确性，并从完整数据帧中取出负载数据和命令；

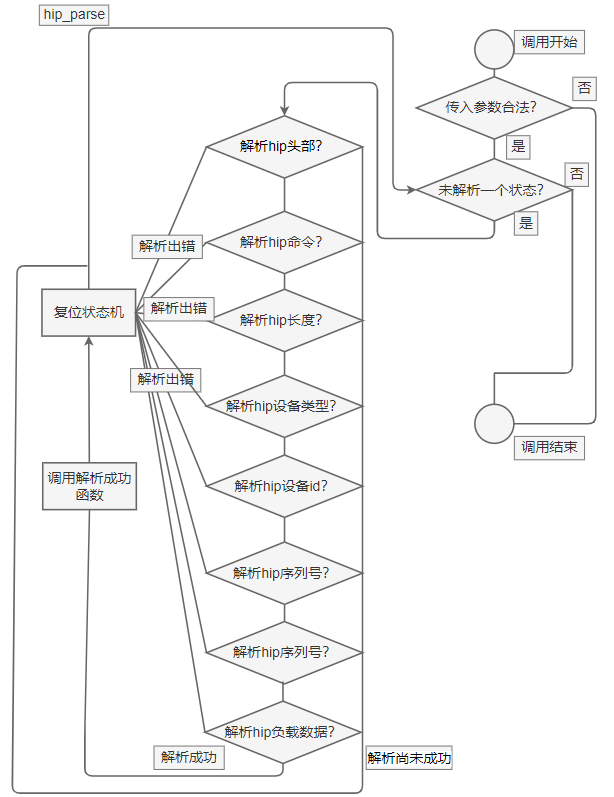
* 输出

­ 通过hip打包好的数据帧；

­ 经过协议解包后的负载数据和命令；

### 统一建模语言（UML）





### 接口设计

* init

int hip\_init(hip\_protocol\_handle\_p hip, void (\*hip\_success\_handle\_func)(hip\_protocol\_type\_p))

|  |  |
| --- | --- |
| 功能描述 | 初始化HIP |
| 参数 | hip\_protocol\_handle\_p：HIP结构体指针 |
| 参数 | (\*hip\_success\_handle\_func)(hip\_protocol\_type\_p)：解析成功函数指针 |
| 返回值 | 0：成功  -1：失败 |

* deinit

int hip\_deinit(hip\_protocol\_handle\_p hip)

|  |  |
| --- | --- |
| 功能描述 | 去初始化HIP |
| 参数 | hip\_protocol\_handle\_p：HIP结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* hip\_pack

int hip\_pack(hip\_protocol\_handle\_p hip, hip\_protocol\_cmd\_em cmd, uint8\_t\* frame\_buf,

uint8\_t frame\_len, uint8\_t\* data\_addr, uint8\_t data\_len)

|  |  |
| --- | --- |
| 功能描述 | 根据具体的帧类型和数据，打包帧 |
| 参数 | hip\_protocol\_handle\_p：HIP结构体指针 |
| 参数 | cmd：需求打包帧的指令 |
| 参数 | frame\_buff：需求打包帧的负载数据容器（传出参数） |
| 参数 | frame\_len：上一参数的最大可用容量 |
| 参数 | data\_addr：需求打包帧的数据地址 |
| 参数 | date\_addr：需求打包帧的数据长度 |
| 返回值 | 整个帧的数据长度：成功  -1：失败 |

* hup\_parse

int hip\_parse(hip\_protocol\_handle\_p hip, uint8\_t data\_byte)

|  |  |
| --- | --- |
| 功能描述 | 解析hip帧协议成功后，调用init（）时注册的回调函数 |
| 参数 | hip\_protocol\_handle\_p：HIP结构体指针 |
| 参数 | data：帧结构的数据之一，FIFO读取的字节数据 |
| 返回值 | 1：成功  -1：失败 |

## ring\_fifo工具

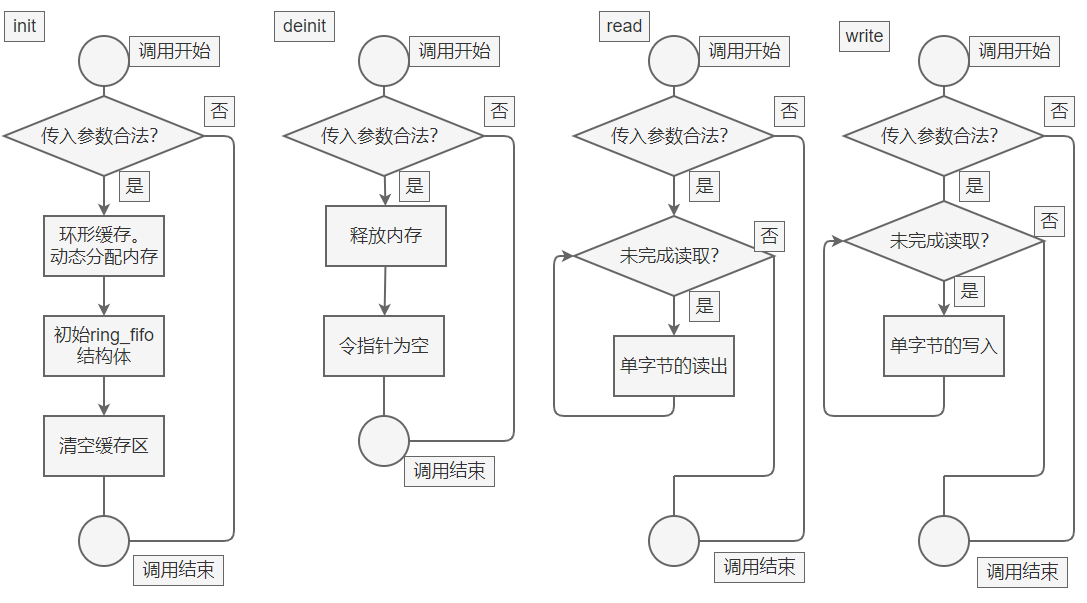
### 功能简介

ring\_fifo是一个具有先进先出特性环形的缓存空间，程序当涉及到数据交互如串口通信、tcp网络通信时，为了尽可能不影响数据读取的实时性，或者避免帧的丢失，读取操作往往需要和处理隔开，在多线程环境中，读取和处理或处于不同的线程进行，而两者的速度也是不一致的，使得两者能够互不影响的正是FIFO的缓冲作用。

### 功能设计

* 输入
* 任意一段数据写入FIFO；
* 内部处理
* 将用户数据以先进先出形式存入FIFO；
* 将FIFO数据以先进先出形式读出到用户空间；
* 输出
* 输出FIFO已缓存的数据；

### 统一建模语言（UML）



### 接口设计

* init

int buffer\_fifo\_init(buffer\_fifo\_p fifo, uint16\_t size);

|  |  |
| --- | --- |
| 功能描述 | 初始化ring\_fifo |
| 参数 | buffer\_fifo\_p fifo：ring\_fifo结构体指针 |
| 参数 | size：环形缓存大小 |
| 返回值 | 0：成功  -1：失败 |

* deinit

int buffer\_fifo\_deinit(buffer\_fifo\_p fifo);

|  |  |
| --- | --- |
| 功能描述 | 去初始化ring\_fifo |
| 参数 | buffer\_fifo\_p fifo：ring\_fifo结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* read

int buffer\_fifo\_read(buffer\_fifo\_p fifo, uint\_8\* read\_buffer\_addr, int read\_buffer\_length);

|  |  |
| --- | --- |
| 功能描述 | 定长读取ring\_fifo中的数据 |
| 参数 | buffer\_fifo\_p fifo：ring\_fifo结构体指针 |
| 参数 | read\_buffer\_addr：读取ring\_fifo后数据存放的区域 |
| 参数 | read\_buffer\_length：准备读取ring\_fifo的长度 |
| 返回值 | 0：成功  -1：失败 |

* write

int buffer\_fifo\_write(buffer\_fifo\_p fifo, uint\_8\* write\_buffer\_addr, int write\_buffer\_length);

|  |  |
| --- | --- |
| 功能描述 | 写入定长数据到ring\_fifo中 |
| 参数 | buffer\_fifo\_p fifo：ring\_fifo结构体指针 |
| 参数 | write\_buffer\_addr：写入ring\_fifo后数据存放的区域 |
| 参数 | write\_buffer\_length：准备写入ring\_fifo的长度 |
| 返回值 | 0：成功  -1：失败 |

## message\_queue工具

### 功能简介

[消息队列](https://so.csdn.net/so/search?q=%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "_blank)常见的使用场景有很多：系统解耦、异步处理以及流量削峰；系统解耦：此处使用解开了次app层之间的耦合；异步处理：多应用对消息队列中同一消息进行处理，应用间并发处理消息，相比串行处理，减少处理时间；流量削峰：广泛应用于秒杀或抢购活动中，避免流量过大导致应用系统挂掉的情况；消息队列的存在随进程或者内核持续存在。

### 功能设计

* 输入

­ 根据已经确定好的消息队列结构体，发送结构体数据到消息队列的数据就是输入；

* 内部处理

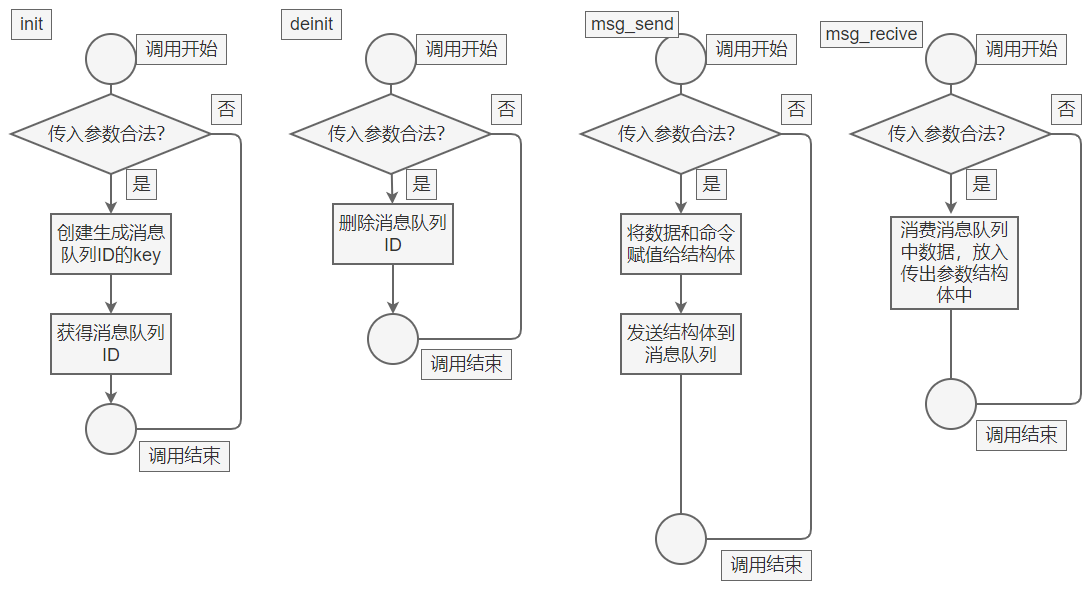
­ 将用户上报的消息以先进先出形式存入；

­ 用户接受消息后，消息会被消费；

* 输出

­ 输出消息队列已经缓存的数据；

### 统一建模语言（UML）



### 接口设计

* init

int msg\_init(int \*msg\_id);

|  |  |
| --- | --- |
| 功能描述 | 初始化message\_queue |
| 参数 | msg\_id：消息队列ID |
| 返回值 | 0：成功  -1：失败 |

* deinit

int msg\_deinit(int msg\_id);

|  |  |
| --- | --- |
| 功能描述 | 去初始化message\_queue |
| 参数 | msg\_id：消息队列ID |
| 返回值 | 0：成功  -1：失败 |

* recive
* int msg\_recive(int msg\_id, msg\_queue\_p recv\_msg);

|  |  |
| --- | --- |
| 功能描述 | 阻塞地接受消息队列中的数据 |
| 参数 | msg\_id：消息队列ID |
| 参数 | msg\_queue\_p：自定义消息结构体指针 |
| 返回值 | 0：成功  -1：失败 |

* send

int msg\_send(int msg\_id, msg\_queue\_p recv\_msg);

|  |  |
| --- | --- |
| 功能描述 | 解析hup帧协议后，调用init（）时注册的回调函数 |
| 参数 | msg\_id：消息队列ID |
| 参数 | msg\_queue\_p：自定义消息结构体指针 |
| 返回值 | 0：成功  -1：失败 |

# 使用协议

## HUP

HUP全称为Hollyland Uart Protocol（串口通信协议），作为公司内部使用的串口数据传输协议。对需要通过串口进行收发的数据进行相应的解包和打包操作，每一包收发的数据都做相应的校验，很大程度上提高了串口传输数据的可靠性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tag | cmd | length | data | checksum |
| 2字节 | 1字节 | 2字节 | 0-65534 | 1字节 |

## HIP

HIP全称为Hollyland Internet Protocol ，HIP报文结构由固定字段和扩展字段组成，固定字段包含命令类型、报文长度、设备类型、通道号、设备ID 、ntp时间和对应的rtp时间；扩展字段根据commond不同而不同，方便功能扩展，详细说明见下表：

|  |  |  |  |
| --- | --- | --- | --- |
| **元素** | **长度（位）** | **描述** | **说明** |
| Version | 8 | 版本号 | 标识协议版本 |
| commond | 8 | 命令类型 | 详见commond对照表 |
| Length | 16 | 报文长度 | 整个HIP报文的长度,单位是字节 |
| Device type | 8 | 设备型号 | 详见device type对照表 |
| Device id | 64 | 设备id，唯一标识符 | 设备唯一的硬件ID,使用网卡物理地址 |
| Seq | 32 | 命令序列号 | 命令唯一的序列号，用于处理丢包、乱序等场景 |
| reserved | 不定 | 扩展字段 | 用于每个commond差异化的数据传输，可使用length-8得到扩展字段的长度 |