数据包的解析和封装

1. *MQTT数据包结构设计*

*每个消息除了消息主题外，还有消息内容。由于设备上使用的是串口通信协议，而浏览器上通常使用的是JSON格式，因此，*不同的通信实体间的数据包格式使用并不一样。为了适应我们的设备，从设备到后端服务进程之间的数据包格式，设计如下：

typedef struct message\_between\_device\_server {

int\_32 message\_version; //版本信息，方便后面做消息兼容，4字节

int\_32 device\_id; //设备id，4字节，用于记录设备

int\_32 device\_type; //设备类型，不同设备所用的通信协议可能不一样

int\_32 client\_id; //浏览器id，4字节，用于记录浏览器

int\_32 message\_type; //消息类型，用于区分通信协议，或者普通的消息

int\_32 req\_or\_ack; //1 请求， 2应答

int\_32 req\_len; //请求的协议长度

int\_32 ack\_len; //应答的协议长度，如果请求，则长度为0

char buff[0]; //变长的数据内容，里面包含请求和应答两部分数据，请求的数据长度为req\_len, 而应答的数据长度为ack\_len

} message\_between\_device\_server;



* 1. 消息类型message\_type

|  |  |
| --- | --- |
| 消息类型 | 含义 |
| 1 | 配置类消息，例如采样频率，采用TLV格式，type-len-value |
| 2 | 普通的通信协议类消息，除程式之外的通信协议数据 |
| 3 | 程式数据，用于程式的数据交互 |

* 1. TLV格式的说明

TLV实际就是type-len-value的说明，用来说明类型，长度，和数值。里面可以多个数据合并在一起。

|  |  |  |
| --- | --- | --- |
| 字段 | 长度 | 说明 |
| Type | 1 Byte | 类型，用来表明具体的配置类型；例如1 采样频率 2 … |
| Length | 4 Bytes | 用来说明数据的长度，数据的长度是可以变化的 |
| Value | 可变的 | 数据的值 |

* 1. Buffer格式说明

请求的协议数据：buff[0]-buff[req\_len-1]，取出这个数据，按照具体的消息类型来解析。

应答的协议数据：buff[req\_len]-buff[req\_len+ack\_len-1]，取出这个数据，按照协议格式单独解析。具体的格式解析，见8.4.

* 1. 几个关键问题

1. 之所以要保存device\_id和client\_id，是方便消息准确回复。例如，web m发送一个消息到设备n, 设备n返回一个应答时，后台服务进程需要知道发送给准确的web，就要通过web的client id了，并同时告知web m是设备 n的应答。
2. 由于消息的大小是变长的，所以设计的buff是变长的
3. Buff中包含了通信协议的数据，既包含了请求，也包含了应答，之所以要包含请求，是方便后台服务进程处理，告诉后台服务进程，发给设备是什么请求，它的应答是什么，这样后台服务进程就能很方便处理；否则，后台服务进程，要处理很多消息，根本不知道发送的什么请求，返回的这个应答
4. 请求消息和应答消息的长度，就分别是req\_len和ack\_len，后台服务进程取到这个消息之后，就按照req\_len长度截取出来，然后按照协议解析，发送的请求是什么；同时，按照ack\_len的长度解析应答。

8.3

* 1. MQTT线程处理web消息流程

Web发过来的数据都是JSON格式的，比较好处理。

[JSON](https://baike.baidu.com/item/JSON" \t "_blank)([JavaScript](https://baike.baidu.com/item/JavaScript" \t "_blank) Object Notation, JS 对象简谱) 是一种轻量级的数据交换格式。它基于[ECMAScript](https://baike.baidu.com/item/ECMAScript" \t "_blank)(欧洲计算机协会制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

这个格式比较通用。用一个简单的例子来说明下，例如一组网站的信息可以组织成：

var sites = [ { "name":"runoob" , "url":"www.runoob.com" },

{ "name":"google" , "url":"www.google.com" },

{ "name":"微博" , "url":"www.weibo.com" }

];

三个网站，每个网站有两个关键属性，一是name，二是url；每个属性由一组key: value对组成，中间以”：”隔离。“{}”之内是一个元素，”[]”之内是一组元素，通过”[]”和“{}”组合，就能表示一个复杂的结构体。

MQTT线程把数据组织成JSON格式，发送给web，web就能快速方便解析出数据，反过来也是一样。

* 1. 数据的封装和解析

8.4.1 普通通信协议的数据封装和解析

一些关键的数据结构定义如下：

通信协议的数据结构：

#define MAX\_DATA\_NUM 100

Typedef struct protocol\_message{

int\_32 device\_id; // 设备id

int\_32 device\_type; // 设备类型

int\_32 req\_or\_ack; // 请求或者应答，向设备请求数据，或从设备获取数据

char command[8]; //命令：AMI/RSD/RRD/WSD/WRD/STD/CLD, 字符串表示

int\_32 result; //应答时返回的结果，请求是忽略此字段

int\_32 data\_num; //数据个数，请求或者返回是的数据个数

int\_32 data[MAX\_DATA\_NUM]; //十六进制数据

}PRO\_MSG;

对于通信协议的封装：

1. 入参：从结构体PRO\_MSG中获取入参
2. 根据入参，按照通信协议的格式，封装出通信协议数据

对于通信协议的解析：

1. 入参：通信数据报文
2. 根据通信协议的格式，对通信数据进行解析
3. 解析出的结果，存放至结构体PRO\_MSG中

8.4.2程式的数据封装和解析

程式的数据结构：

Typedef struct pgm\_data{

Unsigned char name[PEG\_NAME\_SIZE+1];

Struct SgmData{

Float temp;

Float humi;

Unsigned int hour;

Unsigned char min;

Unsigned char sec;

Unsigned char p[P\_NUM];

Unsigned char wt;

}sgm[PEG\_SGM\_SUM];

Float high\_limt;

Float low\_limit;

Short all\_cycle;

Unsigned char link\_pgm;

Struct partcycle{

Unsigned char start;

Unsigned char end;

Short num;

}part\_of\_cycle[4];

}PGM\_DATA\_F;

对程式数据的封装：

1. 入参：从结构体PGM\_DATA\_F中获取入参
2. 根据入参，按照通信协议的格式，封装出通信协议的数据

对程式数据的解析：

1. 入参：通信数据报文
2. 根据通信协议的格式，对通信数据进行解析
3. 解析出的结果，存放至结构体PGM\_DATA\_F中
4. 总体实现以上内容
5. 需要定义详细设计书中定义的message\_between\_device\_server
6. 将通信协议的数据包解析成message\_between\_device\_server
7. 将message\_between\_device\_server定义的数据结构封装成通信协议
8. 使用语言python 3.7，开发环境Ubuntu 16.04