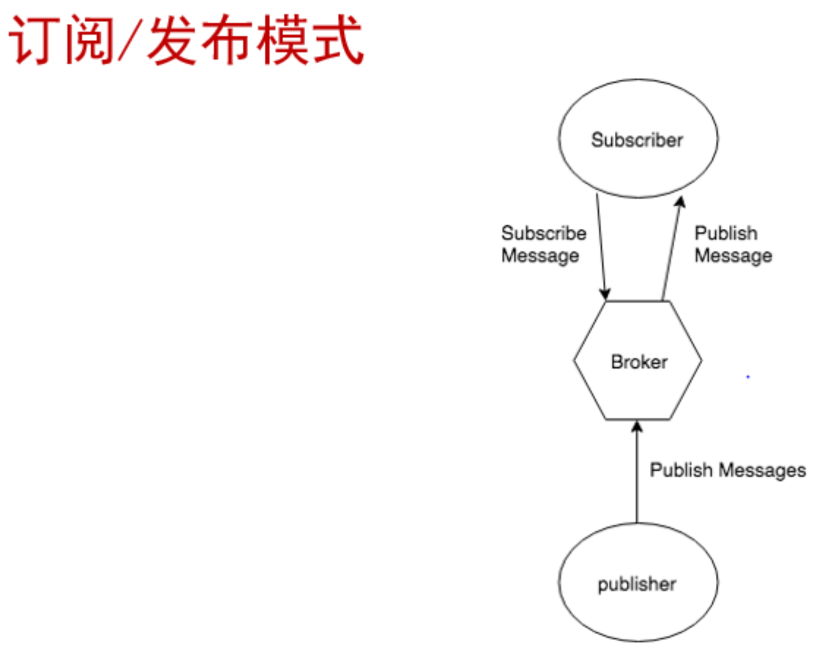
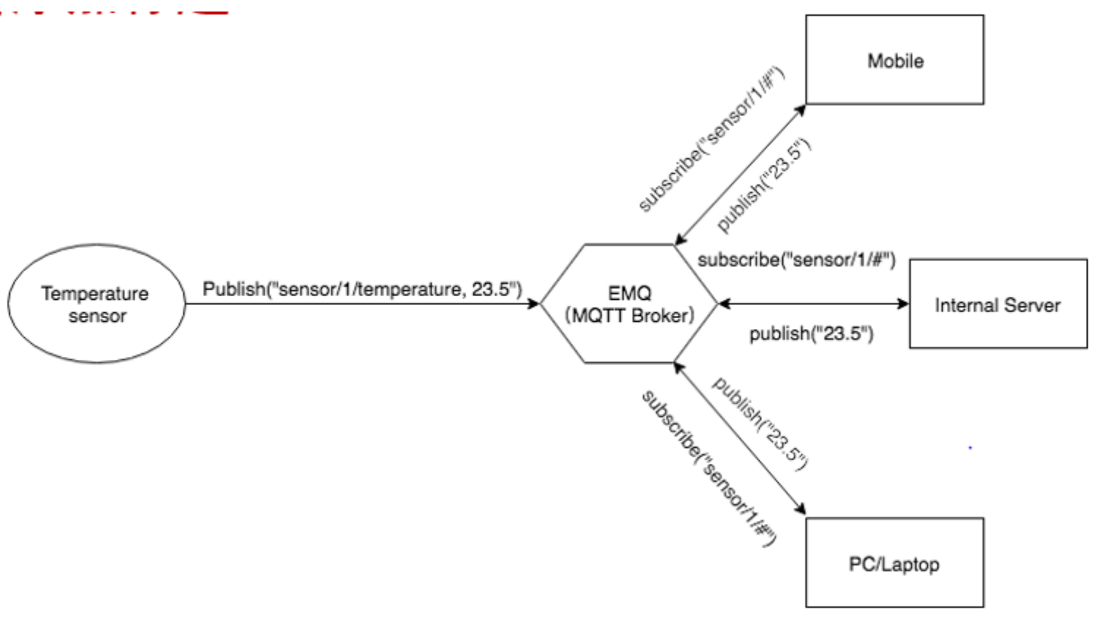
基于mosquitto的MQTT客户端实现C语言-站长资讯中心

基于mosquitto的MQTT客户端实现C语言

在对MQTT的学习过程中 一下的内容对我提供了帮助
https://www.runoob.com/w3cnote/mqtt-intro.html 对MQTT的入门级介绍 很基础讲解了什么是MQTT
https://mosquitto.org/api/files/mosquitto-h.html 这个网站记载了几乎所有的mosquitto的接口 你想知道的函数接口他都有 函数简介可能看不出来什么 点进去看一下详细解释还是很清晰的

以下的图片能够更加直观地说明MQTT协议的通信方式

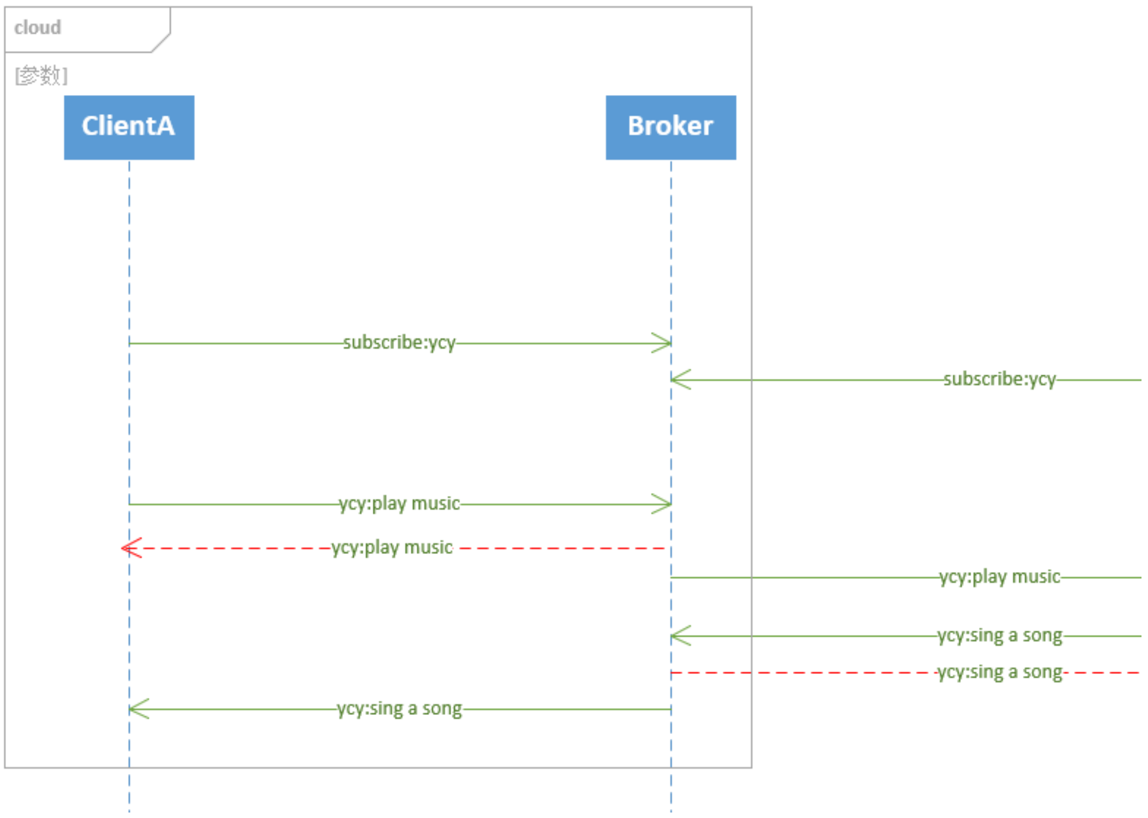




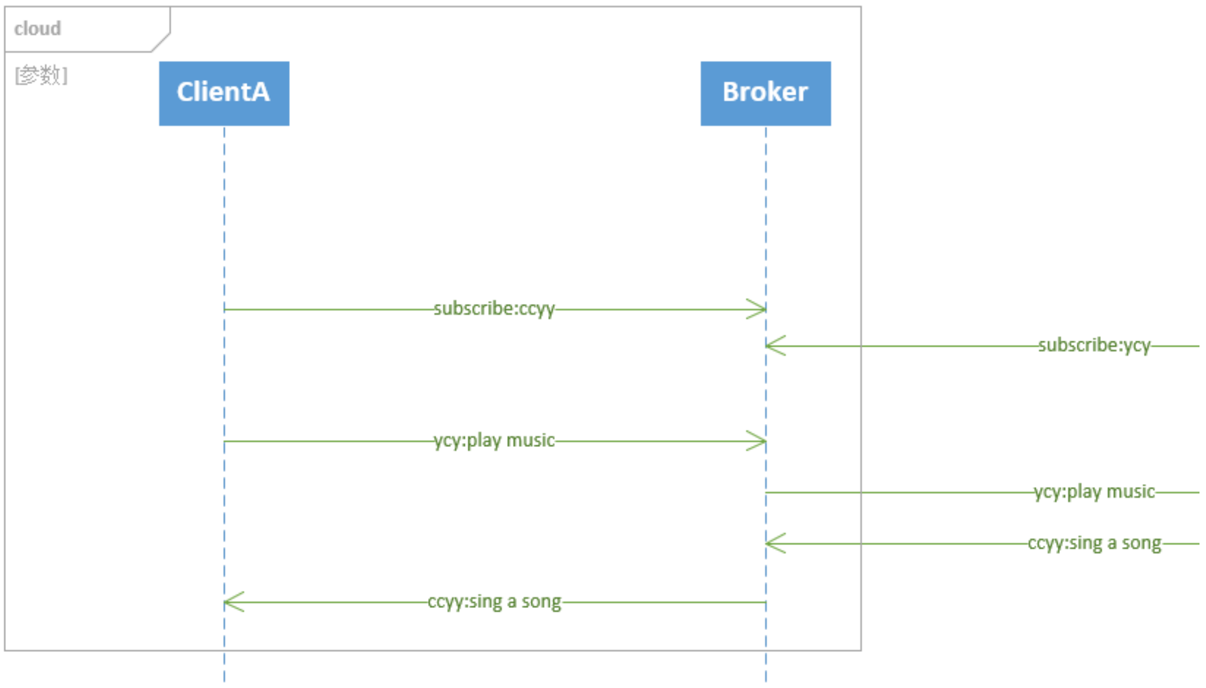
根据这两张图片谈一下我对MQTT的了解：
在MQTT协议中 运行了broker的才算是真正的服务器 它掌控着所有消息的publish和subscribe 而他的客户端程序可以有一下三重情况
1.运行了publisher的程序 做着publish的工作
2.运行了subscriber的程序 做着subscribe的工作
3.运行了subscriber和publisher的程序 既可以publish也可以subscribe 这并不意味着这样的程序就是broker
这三种情况你可以根据自己实际的需要来确定自己的客户端可以是哪一种
例如你的客户端仅仅是需要收到来自broker的topic为"ycy"的消息然后解析处理 并不需要做任何的publish工作 那么你就可以选择第2种客户端

我的程序的功能是从云端得到固定topic的消息之后就开始处理这个消息 处理完成后publish出去 由于我在Linux机上 我必须做一个仿broker来实现从云端得到消息这一步

首先我的预想是这样的



但是这样就会导致我每次publish出去数据 我自己又会收到一份 这样的话我又会再次扫描这个数据 以确定是不是需要解析处理 万一的情况恰好是约定的需要解析的字符(这就误会大了)
所以 在我这里没有使用同一个topic 其实还有解决方法二(下文再说)



解决方法二：
就是在my_message_callback这个函数中过滤一下 将自己publish的topic过滤一下(在我下面的代码中并没有实现 只是提供这样的思路 strcmp函数可以帮到很多 这个方法我另外试了一下 一定要注意多了一个'\n'字符 这是因为你输入之后会按下回车发送 所以发送的字符中会带有这个字符)

mqtt_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mosquitto.h>
#include <string.h>

#define HOST "localhost"
#define PORT  1883
#define KEEP_ALIVE 60
#define MSG_MAX_SIZE  512
#define TOPIC_NUM 3

bool session = true;

const static char* topic[TOPIC_NUM] =
{
    "Gai爷:",
    "ycy ",
    "CCYY "
};

void my_message_callback(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message)
{

    if(message->payloadlen){
        printf("%s %s", message->topic, (char *)message->payload);
    }else{
        printf("%s (null)\n", message->topic);
    }
    fflush(stdout);
}

void my_connect_callback(struct mosquitto *mosq, void *userdata, int result)
{
    int i;
    if(!result){
        /* Subscribe to broker information topics on successful connect. */
        mosquitto_subscribe(mosq, NULL, "CCYY ", 2);
    }else{
        fprintf(stderr, "Connect failed\n");
    }
}

void my_subscribe_callback(struct mosquitto *mosq, void *userdata, int mid, int qos_count, const int *granted_qos)
{
    int i;
    printf("Subscribed (mid: %d): %d", mid, granted_qos[0]);
    for(i=1; i<qos_count; i++){
        printf(", %d", granted_qos[i]);
    }
    printf("\n");
}

void my_log_callback(struct mosquitto *mosq, void *userdata, int level, const char *str)
{
    /* Pring all log messages regardless of level. */
    printf("%s\n", str);
}

int main()
{
    struct mosquitto *mosq = NULL;
    char buff[MSG_MAX_SIZE];

    //libmosquitto 库初始化
    mosquitto_lib_init();
    //创建mosquitto客户端
    mosq = mosquitto_new(NULL,session,NULL);
    if(!mosq){
        printf("create client failed..\n");
        mosquitto_lib_cleanup();
        return 1;
    }
    //设置回调函数，需要时可使用
    mosquitto_log_callback_set(mosq, my_log_callback);
    mosquitto_connect_callback_set(mosq, my_connect_callback);
    mosquitto_message_callback_set(mosq, my_message_callback);
    mosquitto_subscribe_callback_set(mosq, my_subscribe_callback);
    //连接服务器
    if(mosquitto_connect(mosq, HOST, PORT, KEEP_ALIVE)){
        fprintf(stderr, "Unable to connect.\n");
        return 1;
    }
    //开启一个线程，在线程里不停的调用 mosquitto_loop() 来处理网络信息
    int loop = mosquitto_loop_start(mosq);
    if(loop != MOSQ_ERR_SUCCESS)
    {
        printf("mosquitto loop error\n");
        return 1;
    }


    while(fgets(buff, MSG_MAX_SIZE, stdin) != NULL)
    {
        /*发布消息*/
        mosquitto_publish(mosq,NULL,"ycy ",strlen(buff)+1,buff,0,0);
        memset(buff,0,sizeof(buff));
    }

    mosquitto_destroy(mosq);
    mosquitto_lib_cleanup();

    return 0;
}
```

mqtt_client.c

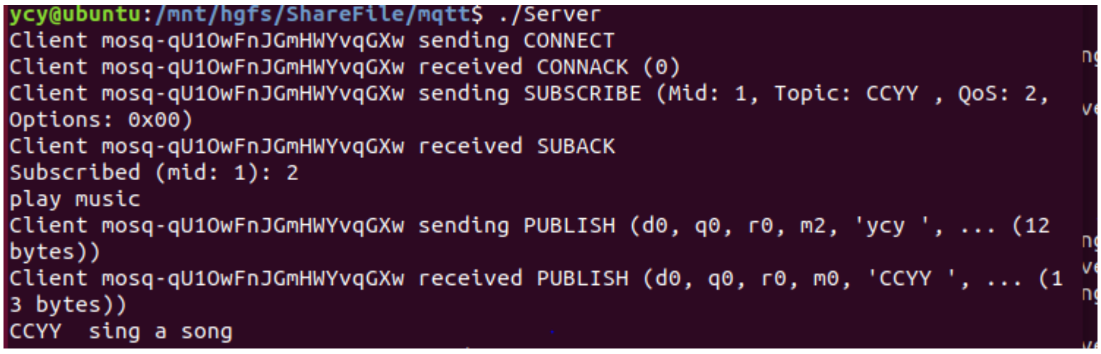
```
#include <stdio.h>
#include <stdlib.h>
#include <mosquitto.h>
#include <string.h>

#define HOST "localhost"
#define PORT  1883
#define KEEP_ALIVE 60
#define MSG_MAX_SIZE  512
#define TOPIC_NUM 3

bool session = true;

const static char* topic[TOPIC_NUM] =
{
    "Gai爷:",
    "ycy ",
    "CCYY "
};

void my_message_callback(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message)
{

    if(message->payloadlen){
        printf("%s %s", message->topic, (char *)message->payload);
    }else{
        printf("%s (null)\n", message->topic);
    }
    fflush(stdout);
}

void my_connect_callback(struct mosquitto *mosq, void *userdata, int result)
{
    int i;
    if(!result){
        /* Subscribe to broker information topics on successful connect. */
        mosquitto_subscribe(mosq, NULL, "ycy ", 2);
    }else{
        fprintf(stderr, "Connect failed\n");
    }
}

void my_subscribe_callback(struct mosquitto *mosq, void *userdata, int mid, int qos_count, const int *granted_qos)
{
    int i;
    printf("Subscribed (mid: %d): %d", mid, granted_qos[0]);
    for(i=1; i<qos_count; i++){
        printf(", %d", granted_qos[i]);
    }
    printf("\n");
}

void my_log_callback(struct mosquitto *mosq, void *userdata, int level, const char *str)
{
    /* Pring all log messages regardless of level. */
    printf("%s\n", str);
}

int main()
{
    struct mosquitto *mosq = NULL;
    char buff[MSG_MAX_SIZE];

    //libmosquitto 库初始化
    mosquitto_lib_init();
    //创建mosquitto客户端
    mosq = mosquitto_new(NULL,session,NULL);
    if(!mosq){
        printf("create client failed..\n");
        mosquitto_lib_cleanup();
        return 1;
    }
    //设置回调函数，需要时可使用
    mosquitto_log_callback_set(mosq, my_log_callback);
    mosquitto_connect_callback_set(mosq, my_connect_callback);
    mosquitto_message_callback_set(mosq, my_message_callback);
    mosquitto_subscribe_callback_set(mosq, my_subscribe_callback);


    //连接服务器
    if(mosquitto_connect(mosq, HOST, PORT, KEEP_ALIVE)){
        fprintf(stderr, "Unable to connect.\n");
        return 1;
    }
    //开启一个线程，在线程里不停的调用 mosquitto_loop() 来处理网络信息
    int loop = mosquitto_loop_start(mosq);
    if(loop != MOSQ_ERR_SUCCESS)
    {
        printf("mosquitto loop error\n");
        return 1;
    }


    while(fgets(buff, MSG_MAX_SIZE, stdin) != NULL)
    {
        /*发布消息*/
        mosquitto_publish(mosq,NULL,"CCYY ",strlen(buff)+1,buff,0,0);
        memset(buff,0,sizeof(buff));
    }

    mosquitto_destroy(mosq);
    mosquitto_lib_cleanup();

    return 0;
}
```

makefile

```
#不是系统默认库 要记得添加连接选项

all:Client
    @echo ""
    @echo "Start compiling......"
    @echo ""
Client:Server
    gcc -o Client mqtt_client.c -lmosquitto -lpthread

Server:
    gcc -o Server mqtt_server.c -lmosquitto -lpthread

clean:
    -rm Server Client
```

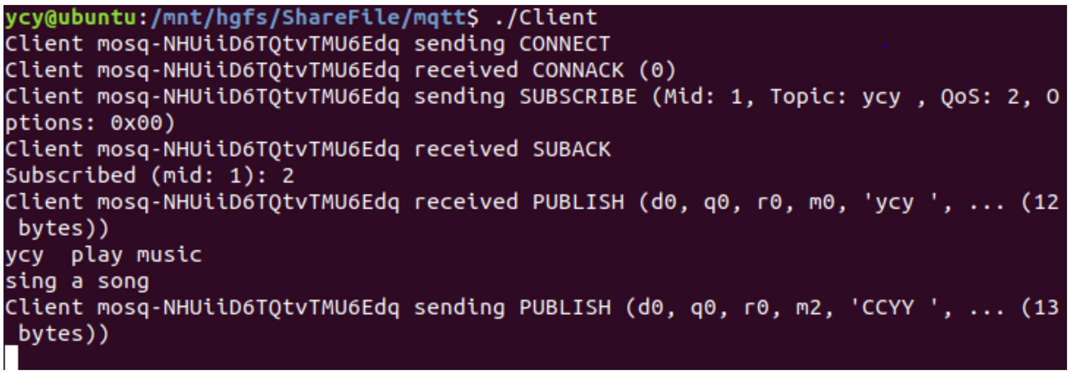
程序运行截图如下：

Server



```
ycy@ubuntu:/mnt/hgfs/ShareFile/mqtt$ ./Server
Client mosq-qU1OwFnJGmHWYvqGXw sending CONNECT
Client mosq-qU1OwFnJGmHWYvqGXw received CONNACK (0)
Client mosq-qU1OwFnJGmHWYvqGXw sending SUBSCRIBE (Mid: 1, Topic: CCYY , QoS: 2,
Options: 0x00)
Client mosq-qU1OwFnJGmHWYvqGXw received SUBACK
Subscribed (mid: 1): 2
play music
Client mosq-qU1OwFnJGmHWYvqGXw sending PUBLISH (d0, q0, r0, m2, 'ycy ', ... (12
bytes))
Client mosq-qU1OwFnJGmHWYvqGXw received PUBLISH (d0, q0, r0, m0, 'CCYY ', ... (1
3 bytes))
CCYY  sing a song
```

Client



```
ycy@ubuntu:/mnt/hgfs/ShareFile/mqtt$ ./Client
Client mosq-NHUiiD6TQtvTMU6Edq sending CONNECT
Client mosq-NHUiiD6TQtvTMU6Edq received CONNACK (0)
Client mosq-NHUiiD6TQtvTMU6Edq sending SUBSCRIBE (Mid: 1, Topic: ycy , QoS: 2, O
ptions: 0x00)
Client mosq-NHUiiD6TQtvTMU6Edq received SUBACK
Subscribed (mid: 1): 2
Client mosq-NHUiiD6TQtvTMU6Edq received PUBLISH (d0, q0, r0, m0, 'ycy ', ... (12
 bytes))
ycy  play music
sing a song
Client mosq-NHUiiD6TQtvTMU6Edq sending PUBLISH (d0, q0, r0, m2, 'CCYY ', ... (13
 bytes))
```

以上

原文链接:https://www.cnblogs.com/y-c-y/p/11686916.html
如有疑问请与原作者联系

标签：QPS数据使用HTML方法