**目录**

1、 struct mosquito 1

概述 1

1.1、int sock 2

1.2、char \*address 2

1.3、char \*id 2

1.4、char \*username and char \*passwd 2

1.5、uint16\_t keepalive 2

1.6、time last\_msg\_in and time last\_msg\_out 2

1.7、struct mosquitto\_client\_msg \*msgs 2

2、 struct mosquitto\_db 5

概述 5

2.1、struct \_mosquitto\_subhier subs 5

2.2、struct mosquitto \*\*bridges 6

2.3、int bridge\_count 6

2.4、struct \_clientid\_index\_hash \*clientid\_index\_hash 6

2.5、struct mqtt3\_config \*config 6

3、 struct \_mosquitto\_subhier 7

概述 7

3.1、struct \_mosquitto\_subhier \*children 7

3.2、struct \_mosquitto\_subhier \*next 7

3.3、struct \_mosquitto\_subleaf \*subs 7

3.4、char \*topic 8

4、 struct \_mosquitto\_subleaf 8

概述 8

4.1、struct mosquitto \*context 8

4.2、struct \_mosquitto\_subleaf \*prev 8

4.3、struct \_mosquitto\_subleaf \*next 8

# struct mosquito

## 概述

结构体struct mosquito主要用于保存一个客户端连接的所有信息，例如用户名、密码、用户ID、向该客户端发送的消息等

## 1.1、int sock

Mosquitto服务器与该客户端连接通信所用的socket描述符

## 1.2、char \*address

该客户端的ip地址

## 1.3、char \*id

该客户端登陆mosquitto时所提供的id，该值与其他的客户端不能重复

## 1.4、char \*username and char \*passwd

记录客户端登陆时所提供的用户名和密码

## 1.5、uint16\_t keepalive

客户端在此时间内向服务器发送一条PING/PONG消息

## 1.6、time last\_msg\_in （收）and time last\_msg\_out

用于记录上次收发消息的时间

## 1.7、struct mosquitto\_client\_msg \*msgs

暂时缓存发往该bridge的消息

**struct mosquitto** {

#ifndef WIN32

int sock;

# ifndef WITH\_BROKER

int sockpairR, sockpairW;

# endif

#else

SOCKET sock;

# ifndef WITH\_BROKER

SOCKET sockpairR, sockpairW;

# endif

#endif

enum \_mosquitto\_protocol protocol;

char \*address;

char \*id;

char \*username;

char \*password;

uint16\_t keepalive;

uint16\_t last\_mid;

enum mosquitto\_client\_state state;

time\_t last\_msg\_in;

time\_t last\_msg\_out;

time\_t ping\_t;

struct \_mosquitto\_packet in\_packet;

struct \_mosquitto\_packet \*current\_out\_packet;

struct \_mosquitto\_packet \*out\_packet;

struct mosquitto\_message \*will;

#ifdef WITH\_TLS

SSL \*ssl;

SSL\_CTX \*ssl\_ctx;

char \*tls\_cafile;

char \*tls\_capath;

char \*tls\_certfile;

char \*tls\_keyfile;

int (\*tls\_pw\_callback)(char \*buf, int size, int rwflag, void \*userdata);

char \*tls\_version;

char \*tls\_ciphers;

char \*tls\_psk;

char \*tls\_psk\_identity;

int tls\_cert\_reqs;

bool tls\_insecure;

#endif

bool want\_write;

bool want\_connect;

#if defined(WITH\_THREADING) && !defined(WITH\_BROKER)

pthread\_mutex\_t callback\_mutex;

pthread\_mutex\_t log\_callback\_mutex;

pthread\_mutex\_t msgtime\_mutex;

pthread\_mutex\_t out\_packet\_mutex;

pthread\_mutex\_t current\_out\_packet\_mutex;

pthread\_mutex\_t state\_mutex;

pthread\_mutex\_t in\_message\_mutex;

pthread\_mutex\_t out\_message\_mutex;

pthread\_t thread\_id;

#endif

bool clean\_session;

#ifdef WITH\_BROKER

bool is\_dropping;

bool is\_bridge;

struct \_mqtt3\_bridge \*bridge;

struct mosquitto\_client\_msg \*msgs;

struct mosquitto\_client\_msg \*last\_msg;

int msg\_count;

int msg\_count12;

struct \_mosquitto\_acl\_user \*acl\_list;

struct \_mqtt3\_listener \*listener;

time\_t disconnect\_t;

struct \_mosquitto\_packet \*out\_packet\_last;

struct \_mosquitto\_subhier \*\*subs;

int sub\_count;

int pollfd\_index;

# ifdef WITH\_WEBSOCKETS

struct libwebsocket\_context \*ws\_context;

struct libwebsocket \*wsi;

# endif

#else

# ifdef WITH\_SOCKS

char \*socks5\_host;

int socks5\_port;

char \*socks5\_username;

char \*socks5\_password;

# endif

void \*userdata;

bool in\_callback;

unsigned int message\_retry;

time\_t last\_retry\_check;

struct mosquitto\_message\_all \*in\_messages;

struct mosquitto\_message\_all \*in\_messages\_last;

struct mosquitto\_message\_all \*out\_messages;

struct mosquitto\_message\_all \*out\_messages\_last;

void (\*on\_connect)(struct mosquitto \*, void \*userdata, int rc);

void (\*on\_disconnect)(struct mosquitto \*, void \*userdata, int rc);

void (\*on\_publish)(struct mosquitto \*, void \*userdata, int mid);

void (\*on\_message)(struct mosquitto \*, void \*userdata, const struct mosquitto\_message \*message);

void (\*on\_subscribe)(struct mosquitto \*, void \*userdata, int mid, int qos\_count, const int \*granted\_qos);

void (\*on\_unsubscribe)(struct mosquitto \*, void \*userdata, int mid);

void (\*on\_log)(struct mosquitto \*, void \*userdata, int level, const char \*str);

//void (\*on\_error)();

char \*host;

int port;

int in\_queue\_len;

int out\_queue\_len;

char \*bind\_address;

unsigned int reconnect\_delay;

unsigned int reconnect\_delay\_max;

bool reconnect\_exponential\_backoff;

bool threaded;

struct \_mosquitto\_packet \*out\_packet\_last;

int inflight\_messages;

int max\_inflight\_messages;

# ifdef WITH\_SRV

ares\_channel achan;

# endif

#endif

#ifdef WITH\_BROKER

UT\_hash\_handle hh\_id;

UT\_hash\_handle hh\_sock;

struct mosquitto \*for\_free\_next;

#endif

};

# struct mosquitto\_db

## 概述

结构体struct mosquitto\_db是mosquitto对所有内部数据的统一管理结构，可以认为是其内部的一个内存**[数据库](http://lib.csdn.net/base/14" \o "MySQL知识库" \t "http://blog.csdn.net/houjixin/article/details/_blank)**。它保存了所有的客户端，所有客户端的订阅关系等等

## 2.1、struct \_mosquitto\_subhier subs

保存订阅树的总根，mosquitto中对所有的topic都在该订阅树种维护，客户端的订阅关系也在该订阅树种维护

## 2.2、struct mosquitto \*\*bridges

用于存储所有客户端变量（类型为struct mosquitto）地址的数组，mosquitto程序中，所有客户端都在此数组中保存

## 2.3、int bridge\_count

保存数组bridge的大小，该值也是当前mosquitto程序中维护的所有客户端数目

## 2.4、struct \_clientid\_index\_hash \*clientid\_index\_hash

保存一个hash表，该hash表可通过客户端的ID快速找到该客户端在数组contexts中的索引

## 2.5、struct mqtt3\_config \*config

保存mosquitto的所有配置信息

**struct mosquitto\_db**{

dbid\_t last\_db\_id;

struct \_mosquitto\_subhier subs;

struct \_mosquitto\_unpwd \*unpwd;

struct \_mosquitto\_acl\_user \*acl\_list;

struct \_mosquitto\_acl \*acl\_patterns;

struct \_mosquitto\_unpwd \*psk\_id;

struct mosquitto \*contexts\_by\_id;

struct mosquitto \*contexts\_by\_sock;

struct mosquitto \*contexts\_for\_free;

#ifdef WITH\_BRIDGE

struct mosquitto \*\*bridges;

#endif

struct \_clientid\_index\_hash \*clientid\_index\_hash;

struct mosquitto\_msg\_store \*msg\_store;

struct mosquitto\_msg\_store\_load \*msg\_store\_load;

#ifdef WITH\_BRIDGE

int bridge\_count;

#endif

int msg\_store\_count;

struct mqtt3\_config \*config;

int persistence\_changes;

struct \_mosquitto\_auth\_plugin auth\_plugin;

#ifdef WITH\_SYS\_TREE

int subscription\_count;

int retained\_count;

#endif

struct mosquitto \*ll\_for\_free;

};

# struct \_mosquitto\_subhier

**struct \_mosquitto\_subhier** {

struct \_mosquitto\_subhier \*children;

struct \_mosquitto\_subhier \*next;

struct \_mosquitto\_subleaf \*subs;

char \*topic;

struct mosquitto\_msg\_store \*retained;

};

## 概述

该数据结构用于保存订阅树的节点（包括叶子节点和中间节点），mosquitto中对订阅树采用孩子-兄弟链表法的方式进行存储

## 3.1、struct \_mosquitto\_subhier \*children

该结构体指针指向同结构的第一个孩子节点

## 3.2、struct \_mosquitto\_subhier \*next

该成员指针指向该节点的下一个兄弟节点

## 3.3、struct \_mosquitto\_subleaf \*subs

该成员指向订阅列表

## 3.4、char \*topic

该节点指向对应的topic片段

# struct \_mosquitto\_subleaf

**struct \_mosquitto\_subleaf** {

struct \_mosquitto\_subleaf \*prev;

struct \_mosquitto\_subleaf \*next;

struct mosquitto \*context;

int qos;

};

## 概述

对某一topic的所有订阅者组织成一个订阅列表，该订阅链表是一个双向链表，链表的每一个节点都保存一个订阅者

## 4.1、struct mosquitto \*context

表示一个订阅客户端

## 4.2、struct \_mosquitto\_subleaf \*prev

表示前一个节点

## 4.3、struct \_mosquitto\_subleaf \*next

表示后一个节点

# struct mqtt3\_config

保存mosquitto的所有配置信息，mosquitto程序在启动时将初始化该结构体并从配置文件中读取配置信息保存于该结构体变量内

# 6、mosquitto源码分析

## 6.1、mosquitto概述

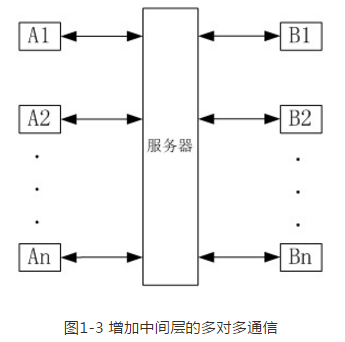
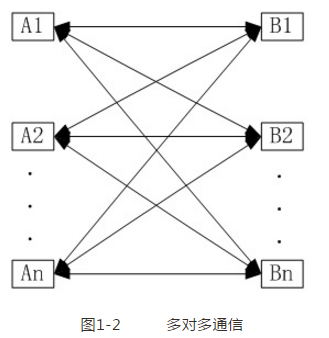
Mosquitto是一款实现了消息推送协议MQTTv3.1的开元代理软件，提供轻量级的，支持可发布/可订阅的消息推送模式，使得设备对设备之间的短消息变得简单，例如现在应用广泛的低功耗传感器，手机，嵌入式计算机，微型控制器等移动设备等移动设备。

Mosquitto采用发布/订阅的模式实现MQTT协议，这种设计模式将通信终端之间的关系统一到服务程序中管理，可大大减轻客户端的开发和维护。

**源码下载：**http://mosquitto.org/files/source/

## 6.2、发布/订阅模式简介

发布/订阅模式定义了如何向一个节点发布和订阅消息，这些节点被称作主题（topic），主题可以被认为是消息的传输中介，发布者（publisher）发布消息到主题，订阅者（subscriber）从主题订阅消息，这种模式使得消息订阅者和消息发布者保持相互独立，不需要解除即可保证消息的传送。



## 6.3、mosquitto核心功能分析

Mosquitto通过订阅树的方式来管理所有的topic以及客户端的订阅关系，它首先将所有的topic按照“/”分割并组织成一颗树结构，从根节点到树中的每个节点即组成该节点对应的一个topic，每个topic都保存一个订阅列表，该订阅列表中保存了所有订阅当前topic的客户端信息。

Mosquitto程序在实现中根据topic消息的性质将订阅树分为两颗子树：业务子树和系统子树。两种子树的区别：

1. 生存周期不同，系统topic无论是否有用户订阅都会存在于订阅树中，而业务topic必须有用户订阅才能存在订阅树中（除非消息字段retain设置为1）
2. 创建方式不同，系统topic在消息发布时进行创建，业务topic即可在订阅时创建，也可在消息发布时创建（此时需要将消息retain字段设置为1）
3. 消息保存方式不同，凡是发布到系统topic的消息都会被保存下来，业务消息将直接挂到订阅列表的各个context的消息队列中，如果没有连接订阅或者未设置其retain字段，消息将不会被保存下来

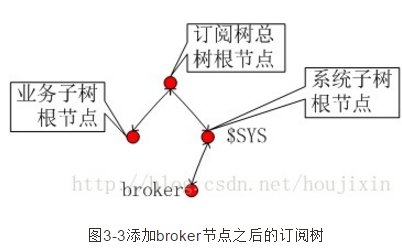
消息的retain字段是否被设置在函数mqtt3\_handle\_publish进行检查，在该函数中有如下代码：

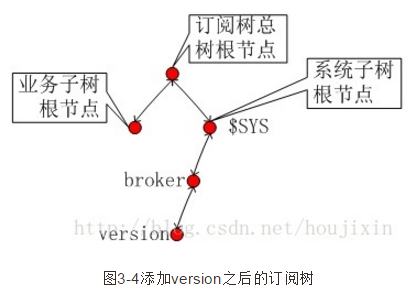
retain = (header & 0x01);

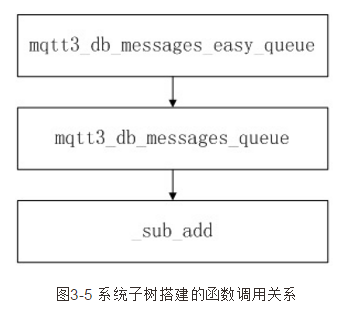
## 6.4、搭建订阅树

### 6.4.1、系统子树搭建过程

例如，mosquitto程序启动时，将首先向系统topic：$SYS/broker/version发送一条版本消息“mosquittoversion 1.2”，此时订阅树的系统子树只有一个根节点

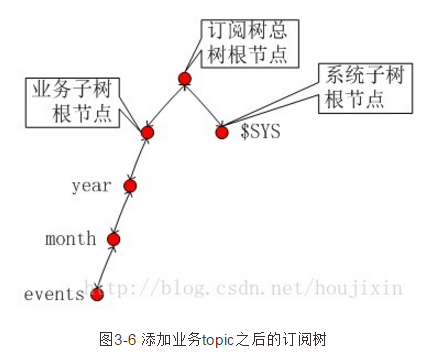


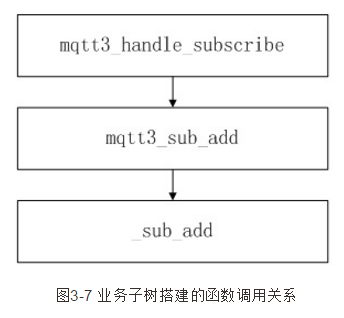




### 6.4.1、业务子树搭建过程

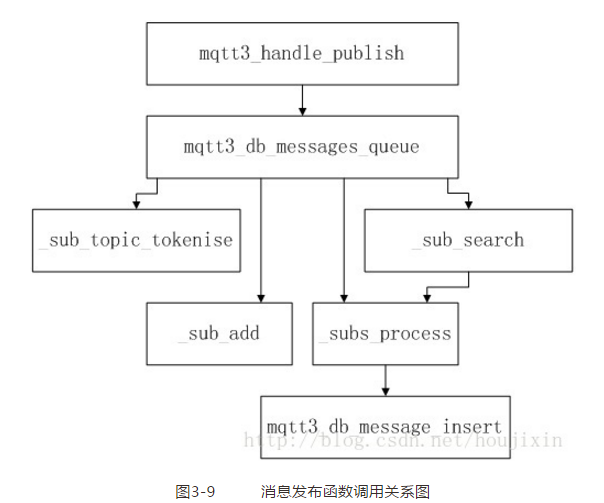
在mosquitto运行期间，收到一条客户端的订阅请求后将调用函数mqtt3\_sub\_add将该客户端挂到对应的业务子树节点的订阅列表中，此时，如果订阅树中不存在客户端所订阅的topic，则会自动为之添加相对应的节点，次过程即为订阅业务子树的搭建过程



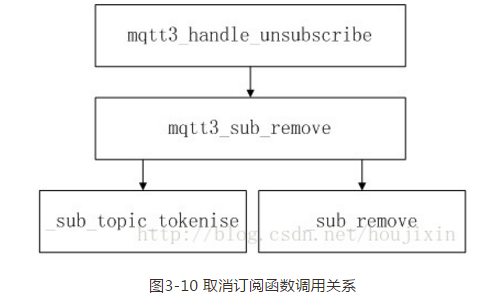


### 6.4.3、使用订阅树发布消息

1. mqtt3\_db\_messages\_queue函数：topic分片
2. \_sub\_search函数递归遍历各个topic片段
3. mqtt3\_db\_message\_insert：将消息插入订阅者的消息队列中



### 6.4.4、取消订阅树



## 6.5、mosquitto的消息收发机制

### 6.5.1、poll机制

struct pollfd{

int fd; //主要是socket文件描述符

short event; //告诉内核关系该描述符的什么

short revent; //描述该文件描述符发生了什么

};

注意：每次调用poll之前需要重新设置一下各文件描述符的状态

### 6.5.2、mosquitto的消息机制

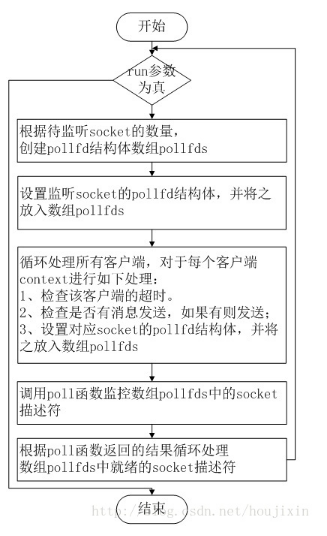
Mosquitto在工作过程中用poll机制监听所有socket以判断其是否有数据发送或者接收，mosquitto所监听的socket按照功能分为监听socket和业务socket，监听socket主要负责监听各客户端的连接，业务socket主要负责mosquitto服务器与客户端之间的数据收发。

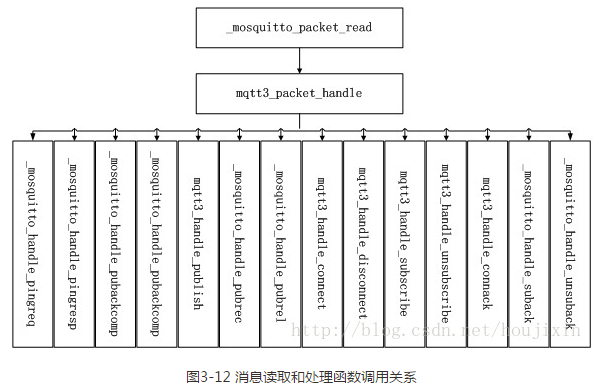
Mosquitto中对poll操作主要在/ mosquitto-1.2/src loop.c文件的mosquitto\_main\_loop函数中，在该函数中将按照以下步骤循环处理所有的socket：

1. 创建pollfd结构体数组pollfds
2. 将要监听的socket放入pollfds中
3. 查询pollfds数组中各描述符的状态
4. Poll函数返回之后，遍历pollfds数组对其中就绪的socket描述符进行处理

使用poll函数监听socket的处理过程在mosquitto\_main\_loop函数中完成，它主要完成的工作是处理每个监听的socket，如果有客户端连接进入时，为之创建一个业务socket和一个对应的context结构体，在mosquitto中一个context就代表一个**客户端连接**。

业务socket的处理主要通过loop\_handle\_reads\_writes函数来完成，在该函数中循环检测所有的context，如果该context对应的socket有数据写入则调用\_mosquitto\_packet\_write进行写操作，如果对应的socket有数据要读取，则调用函数\_mosquitto\_packet\_read完成socket的读取和处理，其中读取数据的处理在mqtt3\_packet\_handle中完成，在该函数中根据不同的消息类型，调用不同的处理函数，业务socket的消息读取及处理的函数调用关系如下图





Mosquitto的消息收发（相对于mosquitto服务器而言）：

1. 消息的接收过程是mosquitto服务器接收到客户端向某个主题发布一条消息
2. 消息的发送过程是mosquitto服务器将消息推送给订阅客户端

在mosquitto的程序实现中，上述两个过程是分开实现和操作的

