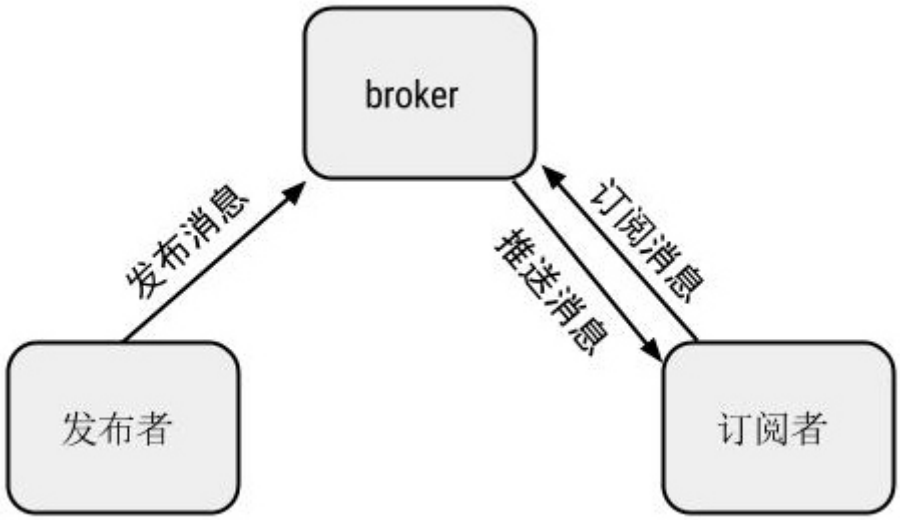
6 minutes

MQTT 协议和 mosquitto

1. MQTT 介绍

http://mqtt.org/

MQTT 是一个轻型协议，使用基于 TCP/IP 协议的发布/订阅消息转发模式，专门用于机器对机器 (M2M) 通信。 MQTT 协议的中心是 MQTT 服务器或代理 (broker) ，支持发布程序和订阅程序进行访问，如下图所示：



用户可以使用 MQTT 构建一个传感器网络，其中各种传感器都能够以其传感器独有的消息形式发布传感器值。 订阅程序能够订阅不同的消息，以据此采取措施。 MQTT 代理将处理从发布程序到订阅程序的转发消息。

如果已经有了一个 broker ，可以直接用 MQTT 客户端软件测试。这里有一个跨平台的 MQTT 客户端 MQTT.fx 。

2. Mosquitto

mosquitto 是一个开源的 MQTT broker ，目前支持 v3.1 和 v3.1.1 协议，同时提供了一个 C 语言动态链接库 libmosquitto ，用于实现 mqtt 客户端：

http://mosquitto.org/documentation/

下载 mosquitto-1.4.2.tar.gz 后，解压，然后执行 make ，make install 。即可得到几个二进制可执行文件：

- mosquitto：mqtt broker
- mosquitto_passwd：管理 mosquitto 密码文件的命令行工具
- mosquitto_sub：mqtt 订阅者程序
- mosquitto_pub ： mqtt 发布者程序

相关的配置文件安装在 /etc/mosquitto/ 目录下。在 Ubuntu 下可以直接安装 `sudo apt-get install mosquitto` 。

现在测试一下客户端和服务端程序。为了测试方便，将客户端和服务端程序都在本机，使用 localhost 连接。执行 `mosquitto -v` 启动 broker ，-v 参数表示打印出运行信息，可以看到默认使用的端口是1883：

```
cl@idp-Lenovo:~/idp/temp/mosquitto-1.3.4$ mosquitto -v
1439262361: mosquitto version 1.3.4 (build date 2015-07-03 18:12:36+0800) starting
1439262361: Using default config.
1439262361: Opening ipv4 listen socket on port 1883.
1439262361: Opening ipv6 listen socket on port 1883.
```

> 如果你的系统出现如下问题，就需要添加一个 mosquitto 用户：

```
root@clanton:~# mosquitto -v
1448587667: Error: Invalid user 'mosquitto'.
root@clanton:~# useradd  mosquitto
root@clanton:~#
```

可以使用 systemd 让 mosquitto 自动启动，添加如下配置文件：

```
ubuntu@VM-231-137-ubuntu:/etc/systemd/system$ cat
mosquitto.service
[Unit]
Description=Mosquitto MQTT Broker
ConditionPathExists=/etc/mosquitto/mosquitto.conf
After=network.target

[Service]
ExecStart=/usr/local/sbin/mosquitto -c
/etc/mosquitto/mosquitto.conf
ExecReload=/bin/kill -HUP $MAINPID
User=mosquitto
Restart=on-failure
RestartSec=10
```

```
[Install]
WantedBy=multi-user.target
```

然后在第二个终端启动订阅者程序: `mosquitto_sub -h localhost -t test -v` ，用 -h 参数指定服务器 IP ，用 -t 参数指定订阅的话题。

在第三个终端启动发布者程序: `mosquitto_pub -h localhost -t test -m "Hello world"` ，用 -m 参数指定要发布的信息内容，然后在订阅者的终端就可以看到由 broker 推送的信息：

```
cl@idp-Lenovo:~/idp/temp/mosquitto-1.3.4$ mosquitto_sub -h localhost -t test -v
test Hello world
```

在 broker 的终端也可以看到处理信息的过程：

```
cl@idp-Lenovo:~/idp/temp/mosquitto-1.3.4$ mosquitto -v
1439263764: mosquitto version 1.3.4 (build date 2015-07-03 18:12:36+0800) starting
1439263764: Using default config.
1439263764: Opening ipv4 listen socket on port 1883.
1439263764: Opening ipv6 listen socket on port 1883.
1439263767: New connection from 127.0.0.1 on port 1883.
1439263767: New client connected from 127.0.0.1 as mosqsub/13562-idp-Lenov (c1, k60).
1439263767: Sending CONNACK to mosqsub/13562-idp-Lenov (0)
1439263767: Received SUBSCRIBE from mosqsub/13562-idp-Lenov
1439263767:     test (QoS 0)
1439263767: mosqsub/13562-idp-Lenov 0 test
1439263767: Sending SUBACK to mosqsub/13562-idp-Lenov
1439263770: New connection from 127.0.0.1 on port 1883.
1439263770: New client connected from 127.0.0.1 as mosqpub/13563-idp-Lenov (c1, k60).
1439263770: Sending CONNACK to mosqpub/13563-idp-Lenov (0)
1439263770: Received PUBLISH from mosqpub/13563-idp-Lenov (d0, q0, r0, m0, 'test', ... (11 bytes)
1439263770: Sending PUBLISH to mosqsub/13562-idp-Lenov (d0, q0, r0, m0, 'test', ... (11 bytes))
1439263770: Received DISCONNECT from mosqpub/13563-idp-Lenov
```

mosquitto 语法是

```
mosquitto [-c config file] [ -d | --daemon ] [-p port
number] [-v]
```

- -c 是指定配置文件的路径，默认不需要配置文件。
- -d 表示作为守护进程运行在后台。
- -p 用来指定监听的端口，默认是 1883 ，使用 TCP 连接，如果要使用 UDP 连接，需要设为 1884。
- -v 表示生成详细的运行日志，等价于配置文件中将 log_type 设为 all 。

mosquitto 默认是不需要配置文件的，它会对所有的选项采用默认值，比如用户名和密码。默认不需要用户名和密码，如果需要，可以用 mosquitto_passwd 新建用户和密码，并管理，语法是：

```
mosquitto_passwd [ -c | -D ] passwordfile username
mosquitto_passwd -b passwordfile username password
mosquitto_passwd -U passwordfile
```

- -c 表示新建一个密码文件，如果文件已经存在，会被覆盖，用户名中不能包含冒号，因为密码文件中用户名和密码是用冒号隔开的。执行之后会要求设置密码，输入内容不可见，密码以加密 hash 值的方式存储在密码文件中。
- -D 表示删除用户名。
- -b 表示在命令行中，以明文方式设置密码。
- -U 用来将密码文件中的明文密码改成加密格式。如果文件中的密码已经是 hash 值，千万不要用这个选项，否则它会对 hash 值再做一次运算，然后修改密码文件。

设好密码后，在配置文件中设置 `allow_anonymous false` 再用 `password_file` 指定密码文件的路径就可以使用了。配置文件可以放在任何位置，只要 mosquitt 能找到它。配置文件中，每一行设置一个选项，选项名称和值用空格隔开，用井号可以注释。安装好的 mosquitto 在 /etc/mosquitto/ 目录下有配置文件和密码文件的例子，复制一份皆可使用：

```
$ ls /etc/mosquitto/
aclfile.example  mosquitto.conf.example
pskfile.example  pwfile.example
```

客户端可以通过订阅 $SYS 层的主题来获取 broker 的信息，这些主题每 sys_interval 秒更新一次，如果 sys_interval 设为 0 则不会发送，标记为 static 的主题只会为每个订阅者发送一次。如果是在命令行中使用，要用反斜杠把 $SYS 作为普通字符串传递给客户端，否则 $SYS 会被当做环境变量来处理。

- $SYS/broker/bytes/received ，broker 从启动开始收到的总字节数。
- $SYS/broker/bytes/sent ，broker 从启动开始发送的总字节数。
- $SYS/broker/clients/connected, $SYS/broker/clients/active (不建议使用)，当前连接的客户端数目。
- $SYS/broker/clients/expired ，由于 persistent_client_expiration 选项过期而断开的客户端数量。
- $SYS/broker/clients/disconnected, $SYS/broker/clients/inactive (不建议使用)，所有断开的已注册客户端（包括清除进程）的数量。

- $SYS/broker/clients/maximum ，过去所有时间所连接的最大客户端数量（从服务器开机开始）。
- $SYS/broker/clients/total ，所有连接过的客户端数量(包括活跃的客户端和已经断开的客户端数量)。
- $SYS/broker/connection/# ，当代理服务器被配置为桥接模式的时候, 通常做法是提供一种状态话题来只是连接的状态，这个话题默认为 $SYS/broker/connection/， 如果数值为1，证明连接是活跃的,为0证明连接不活跃. 查看桥接一节来获取更多信息。
- $SYS/broker/heap/current size ，mosquitto 当前使用的最大内存，请注意，由于编译时候的选择这个话题可能不可用。
- $SYS/broker/heap/maximum size ，mosquitto 曾经使用的最大内存，请注意，由于编译时候的选择这个话题可能不可用。
- $SYS/broker/load/connections/+ ，不同的时间间隔内代理服务器收到连接数据包的平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/bytes/received/+ ，不同的时间间隔内代理服务器收到的平均比特数， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/bytes/sent/+ ，不同的时间间隔内代理服务器发送的平均比特数， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/messages/received/+ ，不同的时间间隔内代理服务器收到各种类型数据包的平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/messages/sent/+ ，不同的时间间隔内代理服务器发送各种类型数据包的平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/publish/dropped/+ ，不同的时间间隔内代理服务器发布数据包丢失的数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/publish/received/+ ，不同的时间间隔内代理服务器发布数据包被收到的平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/publish/sent/+ ，不同的时间间隔内代理服务器发布的数据包被发送的平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/load/sockets/+ ，不同的时间间隔内代理服务器打开socket连接平均数量， 最后的+可以为 5分钟、10分钟、15分钟。
- $SYS/broker/messages/inflight ，具有QoS>0正在等待的确认消息的数量。
- $SYS/broker/messages/received ，从代理服务器开机开始收到的消息总数。
- $SYS/broker/messages/sent ，从服务器开机开始所发送的各种类型消息的总数。
- $SYS/broker/messages/stored ，在消息存储机制中保留的消息总数，包括客户端保留消息和持久客户端的队列消息。

- $SYS/broker/publish/messages/dropped ， 由于队列机制或者传输限制所丢弃数据包的数量. 参照mosquitto.conf 的 max_inflight_messages 和 max_queued_messages 选项获取更多解释。
- $SYS/broker/publish/messages/received ，从代理服务器开机开始发布的信息被收到总数。
- $SYS/broker/publish/messages/sent ，从代理服务器开机开始发布的信息总数。
- $SYS/broker/retained messages/count ，代理服务器中活跃的保留消息的总数。
- $SYS/broker/subscriptions/count ，代理服务器中活跃的订阅的总数。
- $SYS/broker/timestamp ，代理服务器编译的时间戳. Static.
- $SYS/broker/uptime ，服务器合计在线时间（以秒计）。
- $SYS/broker/version ，代理服务器版本. Static.

3. 安全性

MQTT 协议没有对安全性设置强制标准，只是在第五章提出了建议，提供合适的安全功能是实现者的责任。默认情况下，mosquitto 不需要任何验证，用户可以匿名连接。如果设置了 allow_anonymous false ，客户端必须提供正确的用户名和密码进行验证，连接时应该将用户名和密码加密传输，否则有被拦截的危险。此外，mosquitto 还提供基于 SSL/TLS 证书的安全验证，使用 OpenSSL 作为 SSL/TLS 的实现。

3.1. SSL/TLS

我们可以通过这个脚本 https://github.com/owntracks/tools/raw/master/TLS/generate-CA.sh 自建 CA 并颁发证书：

```
$ wget
https://github.com/owntracks/tools/raw/master/TLS/gener
ate-CA.sh .
$ ./generate-CA.sh
```

生成的文件：

- ca.crt ，CA 根证书
- localhost.crt ，mosquitto 服务器上的证书
- localhost.key ，mosquitto 服务器上的密钥

将这三个文件复制到 /etc/mosquitto/certificates/ 目录下。然后修改配置文件，开启 SSL/TLS ：

```
port 8883
cafile /etc/mosquitto/certificates/ca.crt
certfile /etc/mosquitto/certificates/localhost.crt
keyfile /etc/mosquitto/certificates/localhost.key
require_certificate true
```

启动 mosquitto：

```
$ mosquitto -c /etc/mosquitto/mosquitto.conf -v
1495335112: mosquitto version 1.4.11 (build date 2017-
05-20 17:44:03+0800) starting
1495335112: Config loaded from
/etc/mosquitto/mosquitto.conf.
1495335112: Opening ipv4 listen socket on port 8883.
1495335112: Opening ipv6 listen socket on port 8883.
```

再用根证书为客户端生成密钥和证书：

```
$ openssl genrsa -out client.key 2048
$ openssl req -new -out client.csr  -key ./client.key
$ openssl x509 -req -in client.csr -CA ca.crt -CAkey
ca.key -CAserial ./ca.srl -out client.crt -days 3650 -
addtrust clientAuth
```

将根证书 ca.crt 、客户端密钥 client.key 、证书 client.crt 发送给客户端，本地测试的话，直接连接 localhost：

```
$ mosquitto_sub -h localhost -p 8883 -t
\$SYS/broker/bytes/\# -v --cafile ca.crt  --cert
client.crt --key client.key
$SYS/broker/bytes/received 0
$SYS/broker/bytes/sent 0
```

如果设置了 `require_certificate false` ，就是 SSL 单向认证，客户端只需提供 cafile，也无需设置 –cert 和 –key 。如果设置了 `allow_anonymous false` ，还要提供用户名和密码，否则会客户端会报错：

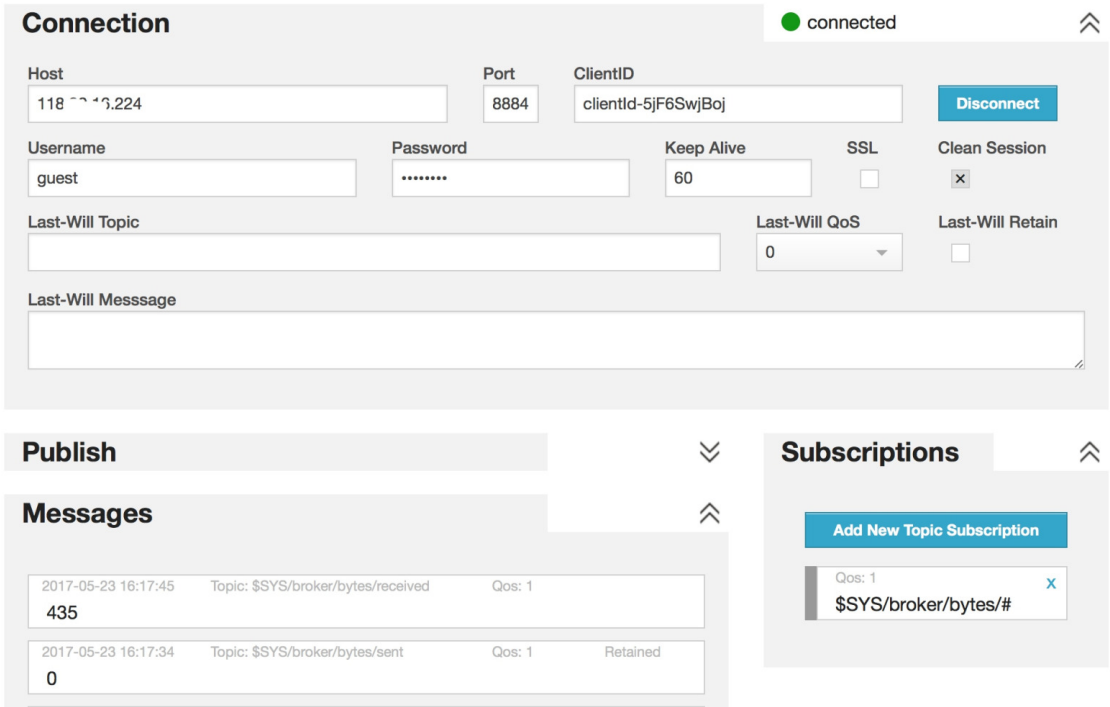
```
$ mosquitto_sub -h localhost -p 8883 -t
\$SYS/broker/bytes/\# -v --cafile ca.crt  --cert
client.crt --key client.key
Connection Refused: not authorised.
```

3.2. WebSockets with SSL/TLS

mosquitto 编译时默认是不支持 WebSockets 的，需要在 config.mk 中将 `WITH_LIBWEBSOCDETS:=no` 改为 yes 。在配置文件中追加 WebSockets 的选项，并加上用户名和密码：

```
listener 8884
protocol websockets
password_file /etc/mosquitto/mosquitto.password
```

然后重启 mosquitto 。可以在 http://www.hivemq.com/demos/websocket-client/ 页面测试，这是一个 Websockets Client 。输入 mosquitto 服务器的 IP 、端口、用户名和密码，即可连接，然后添加订阅话题：



4. libmosquitto 库

关于客户端的编程可以参考 mosquitto_sub 和 mosquitto_pub 的源码。libmosquitto API 文档：http://mosquitto.org/api/files/mosquitto-h.html

4.1. 获取库版本

libmosquitto 是 C 语言共享库，可以创建 MQTT 客户端程序。所有的 API 函数都有 `mosquitto_` 前缀。

```
int mosquitto_lib_version(int *major,int *minor,int
*revision);
```

获取库的版本信息，返回到三个参数中。

4.2. 初始化和清除

```
int mosquitto_lib_init();
int mosquitto_lib_cleanup();
```

使用 libmosquitto 库函数前，要先调用 `mosquitto_lib_init()` 初始化；使用 libmosquitto 库后，要调用 `mosquitto_lib_cleanup()` 完成清除工作。

4.3. 构建和释放客户端

```
struct mosquitto *mosquitto_new(const char *id, bool
clean_session, void *userdata);
```

新建一个 mosquitto 客户端实例。调用成功时，返回一个 struct mosquitto 指针，失败时返回 NULL 指针，并产生错误代码，可以用 `mosquitto_strerror()` 函数获取错误代码的含义。第一个参数需要传递一个字符串作为 client ID ，如果设为 NULL ，会自动生成一个随机 ID 。第二个参数是布尔型，如果设为 false ，当 client 断开连接后，broker 会保留该 client 的订阅和消息，直到再次连接成功；如果设为 true ，client 断开连接后，broker 会将所有的订阅和消息删除。第三个参数是传递给回调函数的用户数据。

释放一个 mosquitto 客户端对象：

```
void mosquitto_destroy(struct mosquitto *mosq);
```

还用一个函数 `mosquitto_reinitialise()` 可以重新初始化一个已经存在的客户端实例。

4.4. 验证和编码

如果 broker 要求提供用户名和密码，可以通过函数设置提供，用户名和密码都是通过字符串传递的：

```
int mosquitto_username_pw_set(  struct mosquitto *
mosq,
                                const char *username,
                                const char *password
                              );
```

4.5. 发布

```
int mosquitto_publish(  struct mosquitto *mosq,
                        int *mid,
                        const char *topic,
                        int payloadlen,
                        const void *payload,
                        int qos,
                        bool retain
                     );
```

发布一个消息。第一个参数是 client 实例。第二个参数要指向一个整数，不能为 NULL ，它会被当做这个消息的 ID 。payloadlen 表示消息的长度，*playload 表示消息的内容。 qos 表示服务质量，retain 表示是否保留信息，详细含义可以查看 MQTT 协议的 3.1 节，对 PUBLISH 控制报文的详细描述。

4.6. 订阅

```
int mosquitto_subscribe( struct mosquitto *mosq,
                            int *mid,
                            const char *sub,
                            int qos
                        );
```

向 broker 发送 SUBSCRIBE 报文请求订阅一个话题。第一个参数是 client 实例。第二个参数如果不为 NULL ，函数会把它作为该话题消息的 ID ，它可以用于订阅回调函数。第三个参数是话题名称。第四个参数是向 broker 请求的服务质量： * 0 表示最多分发一次，消息的分发依赖于底层

网络的能力。接收者不会发送响应，发送者也不会重试。消息可能送达一次也可能根本没送达。 * 1 表示确保消息至少送达一次，需要发布者和订阅者双方在报文中确认，可能重复。 * 2 表示仅分发一次，这是最高等级的服务质量，消息丢失和重复都是不可接受的。使用这个服务质量等级会有额外的开销。

调用成功会返回 MOSQ_ERR_SUCCESS ，但这不代表订阅成功。关于消息质量的详情可以查看 MQTT 协议的 4.3 节。

取消订阅的函数是：

```
int mosquitto_unsubscribe( struct mosquitto *mosq,
                           int *mid,
                           const char *sub,
                         );
```

4.7. 遗嘱

简单的说，当 broker 检测到网络故障、客户端异常等问题，需要关闭某个客户端的连接时，可以向该客户端发布一条消息，叫做遗嘱消息。默认是没有遗嘱消息的，需要用函数设置遗嘱消息的话题、内容、服务质量等，在连接时由客户端告诉 broker ：

```
int mosquitto_will_set(struct   mosquitto *mosq,
                       const char *topic,
                       int payloadlen,
                       const void *payload,
                       int qos,
                       bool retain
                       );
```

该函数必须在 `mosquitto_connect()` 之前调用。对应的清除遗嘱的函数是：

```
int mosquitto_will_clear(struct mosquitto *mosg);
```

4.8. 连接和断开

```
int mosquitto_connect(  struct mosquitto *mosq,
                           const char *host,
```

```
                    int port,
                    int keepalive,
                    );
```

连接一个 broker 。第一个参数是 client 实例。第二个参数是 broker 的 IP 或者 hostname 。第三参数是连接的端口，通常是 1883 。第四个参数是保持连接的时间间隔，单位是秒，关于这个参数的详细含义可以查看 MQTT 协议的 CONNECT 报文格式。连接成功会返回 MOSQ_ERR_SUCCESS 。

该函数还有一些扩展，比如可以绑定本地网络接口的 `mosquitto_connect_bind` 。如果调用了这两个函数连接 broker ，就必须使用 `mosquitto_loop()` 或者 `mosquitto_loop_forever()` 启动网络循环。还有一些非阻塞的连接函数，例如 `mosquitto_connect_async()` ，调用它们的时候，必须使用 `mosquitto_loop_start()` 启动网络循环。断开连接可以调用 `int mosquitto_disconnect(struct mosquitto *mosq)` ，此外还有重连函数 `mosquitto_reconnect()` 。

4.9. 网络循环

```
int mosquitto_loop(struct mosquitto *mosq,
                   int timeout,
                   int max_packets
                   );
```

客户端主网络循环，必须调用该函数来保持 client 和 broker 之间的通讯。收到或者发送消息时，它会调用相应的回调函数处理。当 QoS>0 时，它还会尝试重发消息。第一个参数是 client 实例。timeout 是阻塞等待时间，单位是微妙，设为 0 表示立即返回，设为负数表示使用默认值 1000ms 。max_packets 目前没有使用，设为 1 即可。

通常不会直接调用 `mosquitto_loop()` 。如果程序里只会运行一个 MQTT client 的循环，可以调用 `mosquitto_loop_forever()` ，参数完全相同:

```
int mosquitto_loop_forever(struct mosquitto *mosq,
                   int timeout,
                   int max_packets
                   );
```

它会在一个阻塞的无限循环里调用 `mosquitto_loop()` ，如果从服务器掉线了，它会自动重连。直到在某个回调函数中调用了 `mosquito_disconnect()` ，该函数才会返回。

另一种方法是使用 `mosquitto_loop_start()` ，调用一次就会新建一个线程，在线程里不停的调用 `mosquitto_loop()` 来处理网络信息。

如果要将 mosquitto 客户端操作与您自己的 `select()` 调用集成，请使用 `mosquitto_socket()` ， `mosquitto_loop_read()` ， `mosquitto_loop_write()` 和 `mosquitto_loop_misc()` 函数。

4.10. 回调函数

设置不同的回调函数， `mosquitt_loop()` 会根据不同的情况，调用相应的回掉函数。

####确认连接回调函数

```
void mosquitto_connect_callback_set(struct mosquitto
*mosq,
    void (*on_connect)(struct mosquitto *mosq, void
*obj, int rc)
    );
```

当 client 请求连接后，broker 会回应一条 CONNECK 消息（确认连接请求），client 收到后会调用回调函数。我们可以在这个回调函数里判断连接是否成功，成功后再调用 `mosquitto_subscribe()` 订阅话题，可以依次订阅多个话题。第一个参数是用 `mosquitto_new()` 函数新建的 client 实例。第二个参数是回调函数。回调函数的三个参数：

- mosq ，client 实例
- obj ， `mosquitto_new()` 函数提供的用户数据
- rc ，broker 回应的代码，0 表示连接成功，其他值表示拒绝连接，可以用 `mosquitto_connack_string()` 函数获取代码的含义。

####断开连接回调函数

```
void mosquitto_disconnect_callback_set(struct mosquitto
*mosq,
        void (*on_disconnect)(struct mosquitto *mosq,
void *obj, int rc)
        );
```

当 client 与 broker 断开连接后，会调用这里设置的函数。回调函数中的 rc 表示断开连接的原因，0 表示 client 调用了 `mosquitt_disconnect()` ，其他值都表示未知原因。我们可以在这个函数里判断客户端是否意外掉线。

####消息回调函数

```
void mosquitto_message_callback_set( struct mosquitto
*mosq,
    void (*on_message)(struct mosquitto *mosq, void
*obj, const struct mosquitto_message *message)
);
```

当 client 收到 broker 发来的消息时会调用它。第一个参数是 client 实例。第二个参数是回调函数，它的第二个参数是 `mosquitto_new()` 函数提供的用户数据。第三个参数保存了收到的消息，回掉函数退出后这个指针指向的内存就会被释放，它的定义是：

```
struct mosquitto_message{
    int mid;
    char *topic;  //消息话题
    void *payload;  //消息内容，MQTT 中叫做有效载荷
    int payloadlen;  //消息的长度，单位是字节
    int qos;  //服务质量
    bool retain;  //是否保留消息
};
```

qos 和 retain 的详细含义可以查看 MQTT 协议的 3.1 节，对 PUBLISH 控制报文的详细描述。

####订阅回调函数

```
void mosquitto_subscribe_callback_set(struct mosquitto
*mosq,
        void (*on_subscribe)(struct mosquitto *mosq,
void *obj, int mid, int qos_count, const int
*granted_qos)
        );
```

broker 收到订阅请求后，会向 client 发送一个 SUBACK 报文作为回应，client 收到后就会调用这里设置的函数，可以在回调函数里判断订阅是否

成功。回调函数中， mid 表示消息 ID ，granted_qos 指向一个数组，里面存放的是每一个已经批准的订阅的 QoS ，qos_count 表示 granted_qos 数组的大小。

####取消订阅回调函数

```
void mosquitto_unsubscribe_callback_set(struct
mosquitto *mosq,
        void (*on_unsubscribe)(struct mosquitto *mosq,
void *obj, int mid)
        );
```

broker 收到取消订阅的请求后，会想 client 发送一个 UNSUBACK 报文作为回应，client 收到后就会调用这里设置的函数。

####日志回调函数

```
void mosquitto_log_callback_set(struct mosquitto *mosq,
        void (*on_log)(struct mosquitto *mosq, void
*obj, int level, const char *str)
        );
```

当 libmosquitto 产生日志信息时会调用这里设置的函数，我们可以在回调函数中打印这些日志。level 表示日志信息的等级，包括 MOSQ_LOG_INFO、MOSQ_LOG_NOTICE、MOSQ_LOG_WARNING、MOSQ_LOG_ERR 和 MOSQ_LOG_DEBUG 。str 是日志信息的内容。

5. 后记

其实，目前可选的 MQTT Borker 很多，除了 mosquitto ，还有 Apache Apollo，支持多种协议，还有国人开发 emqtt，号称是百万级分布式开源物联网MQTT消息服务器。还有一个跟好用的 MQTT client —— paho ，也是开源产品，支持多种语言，包括 C 、C++、Python、Java、JavaScript 等。

6. 参考

- MQTT 协议中文版
- MQTT 入门篇
- MQTT 进阶篇
- MQTT 安全篇
- MQTT 实战篇

- mosquitto 协议之 mosquitto(8) – the broker
- Mosquitto SSL Configuration

1181 Words

2015-08-11 08:00 +0800

← 获得 IP 所在地的网… Python 的多线程 →