EMQ 分布物联网消息队列

2019年03月

**修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订日期 | 修订人员 | 修订纪要 |
| V1.0 | 2019-03-25 | 徐强辉 | 初始版本 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**文档目录**

[EMQTT 分布物联网消息队列 1](#_Toc5809107)

[1. MQTT协议 5](#_Toc5809108)

[1.1. MQTT轻量发布订阅消息协议 5](#_Toc5809109)

[1.1.1. 概览 5](#_Toc5809110)

[1.1.2. 特点 5](#_Toc5809111)

[1.1.3. 应用 5](#_Toc5809112)

[1.2. MQTT基于主题(Topic)消息路由 5](#_Toc5809113)

[1.3. MQTT V3.1.1协议报文 6](#_Toc5809114)

[1.3.1. 报文结构 6](#_Toc5809115)

[1.3.2. 固定报头 6](#_Toc5809116)

[1.3.3. 报文类型 7](#_Toc5809117)

[1.3.4. PUBLISH发布消息 7](#_Toc5809118)

[1.3.5. PINGREQ/PINGRESP心跳 7](#_Toc5809119)

[1.4. MQTT消息QoS 7](#_Toc5809120)

[1.4.1. Qos0消息发布订阅 8](#_Toc5809121)

[1.4.2. Qos1消息发布订阅 9](#_Toc5809122)

[1.4.3. Qos2消息发布订阅 10](#_Toc5809123)

[1.5. MQTT会话(Clean Session) 10](#_Toc5809124)

[1.6. MQTT连接保活心跳 11](#_Toc5809125)

[1.7. MQTT遗愿消息(Last Will) 11](#_Toc5809126)

[1.8. MQTT保留消息(Retained Message) 11](#_Toc5809127)

[1.9. MQTT WebSocket连接 12](#_Toc5809128)

[1.10. MQTT协议客户端库 12](#_Toc5809129)

[1.10.1. emqtt客户端库 12](#_Toc5809130)

[1.10.2. Eclipse Paho客户端库 13](#_Toc5809131)

[1.10.3. mqtt.org官网客户端库 13](#_Toc5809132)

[1.11. MQTT与XMPP协议对比 13](#_Toc5809133)

[2. MQTT-SN 协议 14](#_Toc5809134)

[2.1. MQTT-SN 和 MQTT 的区别 14](#_Toc5809135)

[2.2. EMQ-SN 网关插件 14](#_Toc5809136)

[3. MQTT安装部署 15](#_Toc5809137)

[3.1. 安装启动 15](#_Toc5809138)

[3.1.1. 下载版本 v2.3.11 15](#_Toc5809139)

[3.1.2. 虚拟机准备 15](#_Toc5809140)

[3.1.3. 解压安装包 16](#_Toc5809141)

[3.1.4. 配置参数 16](#_Toc5809142)

[3.1.4.1. Erlang虚拟机参数 16](#_Toc5809143)

[3.1.4.2. EMQ消息服务器参数 16](#_Toc5809144)

[3.1.4.3. linux操作系统配置 16](#_Toc5809145)

[3.1.4.4. 其他配置 18](#_Toc5809146)

[3.1.5. 启动emq集群 18](#_Toc5809147)

[3.1.5.1. 启动单节点 18](#_Toc5809148)

[3.1.5.2. 加入emq集群 18](#_Toc5809149)

[3.1.5.3. 查看集群状态 18](#_Toc5809150)

[3.1.5.4. web控制台 19](#_Toc5809151)

[4. 安装Haproxy 19](#_Toc5809152)

[5. 压测工具 19](#_Toc5809153)

[5.1. 依赖安装 19](#_Toc5809154)

[5.1.1. erlang依赖 19](#_Toc5809155)

[5.1.2. 安装erlang 20](#_Toc5809156)

[5.1.2.1. 下载解压 20](#_Toc5809157)

[5.1.2.2. 配置 20](#_Toc5809158)

[5.1.2.3. 安装 20](#_Toc5809159)

[5.1.2.4. 配置profile 20](#_Toc5809160)

[5.2. 安装测试工具 20](#_Toc5809161)

[5.2.1. 从github上下载源码 20](#_Toc5809162)

[5.2.2. 发布测试指令 21](#_Toc5809163)

[5.2.3. 订阅测试指令 22](#_Toc5809164)

[5.2.4. 管理控制后台 23](#_Toc5809165)

# MQTT协议

## MQTT轻量发布订阅消息协议

### 概览

MQTT是一个轻量的发布订阅模式消息传输协议，专门针对低带宽和不稳定网络环境的物联网应用设计。

MQTT官网: http://mqtt.org

MQTT V3.1.1协议规范: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

### 特点

1. 开放消息协议，简单易实现
2. 发布订阅模式，一对多消息发布
3. 基于TCP/IP网络连接
4. 1字节固定报头，2字节心跳报文，报文结构紧凑
5. 消息QoS支持，可靠传输保证

### 应用

MQTT协议广泛应用于物联网、移动互联网、智能硬件、车联网、电力能源等领域。

1. 物联网M2M通信，物联网大数据采集
2. Android消息推送，WEB消息推送
3. 移动即时消息，例如Facebook Messenger
4. 智能硬件、智能家具、智能电器
5. 车联网通信，电动车站桩采集
6. 智慧城市、远程医疗、远程教育
7. 电力、石油与能源等行业市场

## MQTT基于主题(Topic)消息路由

**MQTT协议基于主题(Topic)进行消息路由，主题(Topic)类似URL路径，例如:**

chat/room/1

sensor/10/temperature

sensor/+/temperature

$SYS/broker/metrics/packets/received

$SYS/broker/metrics/#

**主题(Topic)通过’/’分割层级，支持’+’, ‘#’通配符:**

'+': 表示通配一个层级，例如a/+，匹配a/x, a/y

'#': 表示通配多个层级，例如a/#，匹配a/x, a/b/c/d

**订阅者与发布者之间通过主题路由消息进行通信，例如采用mosquitto命令行发布订阅消息:**

mosquitto\_sub -t a/b/+ -q 1

mosquitto\_pub -t a/b/c -m hello -q 1

**备注: 订阅者可以订阅含通配符主题，但发布者不允许向含通配符主题发布消息。**

## MQTT V3.1.1协议报文

### 报文结构

|  |
| --- |
| 固定报头(Fixed header) |
| 可变报头(Variable header) |
| 报文有效载荷(Payload) |

### 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| byte1 | Mqtt packet type | | | | Flags | | | |
| byte2 | Remaining Length | | | | | | | |

### 报文类型

|  |  |  |
| --- | --- | --- |
| 类型名称 | 类型值 | 报文说明 |
| CONNECT | 1 | 发起连接 |
| CONNACK | 2 | 连接回执 |
| PUBLISH | 3 | 发布消息 |
| PUBACK | 4 | 发布回执 |
| PUBREC | 5 | QoS2消息回执 |
| PUBREL | 6 | QoS2消息释放 |
| PUBCOMP | 7 | QoS2消息完成 |
| SUBSCRIBE | 8 | 订阅主题 |
| SUBACK | 9 | 订阅回执 |
| UNSUBSCRIBE | 10 | 取消订阅 |
| UNSUBACK | 11 | 取消订阅回执 |
| PINGREQ | 12 | PING请求 |
| PINGRESP | 13 | PING响应 |
| DISCONNECT | 14 | 断开连接 |

### PUBLISH发布消息

PUBLISH报文承载客户端与服务器间双向的发布消息。

PUBACK报文用于接收端确认QoS1报文，

PUBREC/PUBREL/PUBCOMP报文用于QoS2消息流程。

### PINGREQ/PINGRESP心跳

客户端在无报文发送时，按保活周期(KeepAlive)定时向服务端发送PINGREQ心跳报文，服务端响应PINGRESP报文。

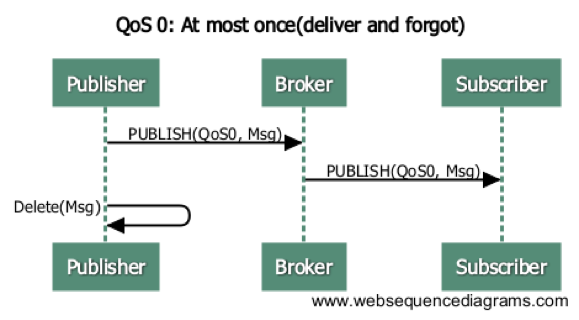
PINGREQ/PINGRESP报文均2个字节。

## MQTT消息QoS

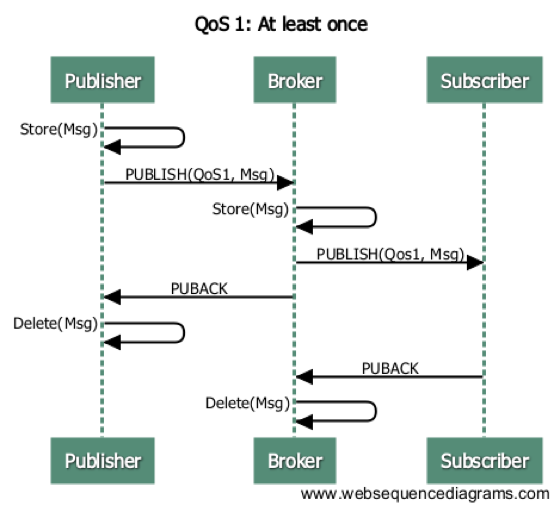
MQTT发布消息QoS保证不是端到端的，是客户端与服务器之间的。订阅者收到MQTT消息的QoS级别，最终取决于发布消息的QoS和主题订阅的QoS。

|  |  |  |
| --- | --- | --- |
| 发布消息的QoS | 主题订阅的QoS | 接收消息的QoS |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 2 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 2 | 0 | 0 |
| 2 | 1 | 1 |
| 2 | 2 | 2 |

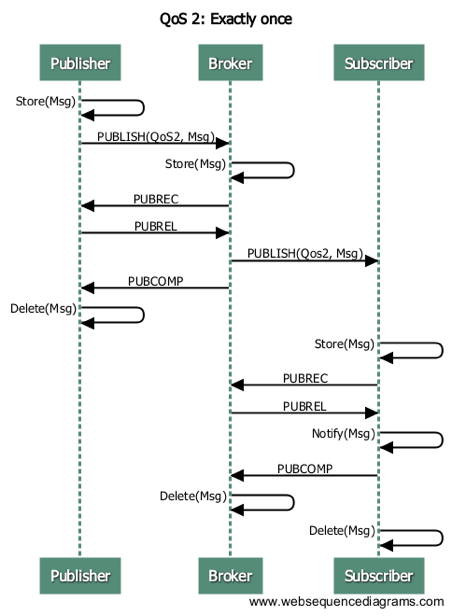
### Qos0消息发布订阅



### Qos1消息发布订阅



### Qos2消息发布订阅



## MQTT会话(Clean Session)

MQTT客户端向服务器发起CONNECT请求时，可以通过’Clean Session’标志设置会话。

‘Clean Session’设置为0，表示创建一个持久会话，在客户端断开连接时，会话仍然保持并保存离线消息，直到会话超时注销。

‘Clean Session’设置为1，表示创建一个新的临时会话，在客户端断开时，会话自动销毁。

## MQTT连接保活心跳

MQTT客户端向服务器发起CONNECT请求时，通过KeepAlive参数设置保活周期。

客户端在无报文发送时，按KeepAlive周期定时发送2字节的PINGREQ心跳报文，服务端收到PINGREQ报文后，回复2字节的PINGRESP报文。

服务端在1.5个心跳周期内，既没有收到客户端发布订阅报文，也没有收到PINGREQ心跳报文时，主动心跳超时断开客户端TCP连接。

注意: emqttd消息服务器默认按最长2.5心跳周期超时设计

## MQTT遗愿消息(Last Will)

MQTT客户端向服务器端CONNECT请求时，可以设置是否发送遗愿消息(Will Message)标志，和遗愿消息主题(Topic)与内容(Payload)。

MQTT客户端异常下线时(客户端断开前未向服务器发送DISCONNECT消息)，MQTT消息服务器会发布遗愿消息。

## MQTT保留消息(Retained Message)

MQTT客户端向服务器发布(PUBLISH)消息时，可以设置保留消息(Retained Message)标志。保留消息(Retained Message)会驻留在消息服务器，后来的订阅者订阅主题时仍可以接收该消息。

例如mosquitto命令行发布一条保留消息到主题’a/b/c’:

mosquitto\_pub -r -q 1 -t a/b/c -m 'hello'

之后连接上来的MQTT客户端订阅主题’a/b/c’时候，仍可收到该消息:

$ mosquitto\_sub -t a/b/c -q 1hello

保留消息(Retained Message)有两种清除方式:

1. 客户端向有保留消息的主题发布一个空消息:

mosquitto\_pub -r -q 1 -t a/b/c -m ''

2. 消息服务器设置保留消息的超期时间

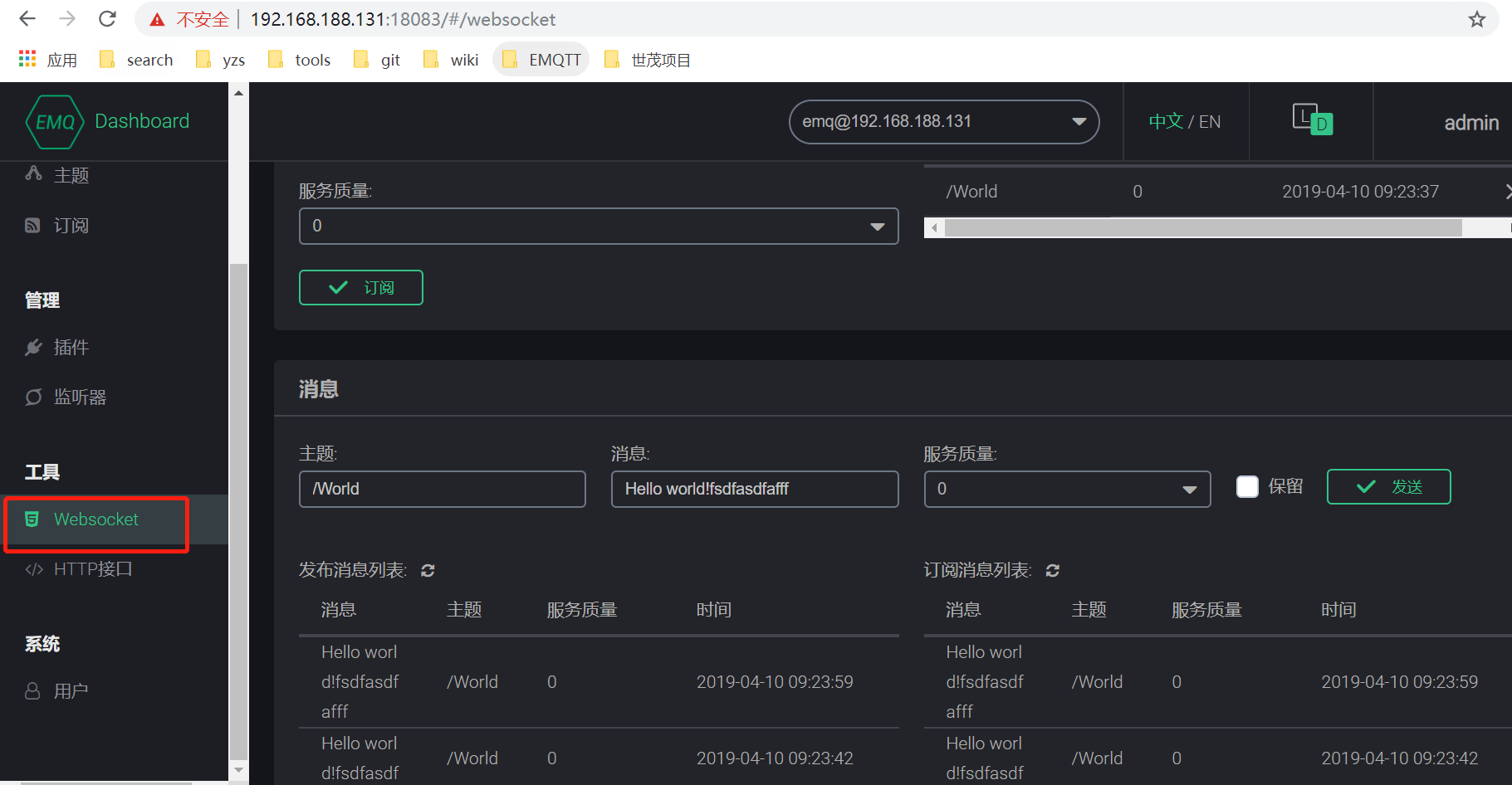
## MQTT WebSocket连接

MQTT协议除支持TCP传输层外，还支持WebSocket作为传输层。通过WebSocket浏览器可以直连MQTT消息服务器，发布订阅模式与其他MQTT客户端通信。

MQTT协议的WebSocket连接，必须采用binary模式，并携带子协议Header:

Sec-WebSocket-Protocol: mqttv3.1 或 mqttv3.1.1

可以用管理控制台websocket 工具测试.



## MQTT协议客户端库

### emqtt客户端库

emqtt项目组: <https://github.com/emqtt>

|  |  |  |
| --- | --- | --- |
| emqttc | https://github.com/emqtt/emqttc | Erlang MQTT客户端库 |
| CocoaMQTT | https://github.com/emqtt/CocoaMQTT | Swift语言MQTT客户端库 |
| QMQTT | https://github.com/emqtt/qmqtt | QT框架MQTT客户端库 |

### Eclipse Paho客户端库

Paho官网: <http://www.eclipse.org/paho/>

Spring-boot maven依赖

<!--mqtt-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-integration</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.integration</groupId>

<artifactId>spring-integration-stream</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.integration</groupId>

<artifactId>spring-integration-mqtt</artifactId>

</dependency>

### mqtt.org官网客户端库

mqtt.org: <https://github.com/mqtt/mqtt.github.io/wiki/libraries>

## MQTT与XMPP协议对比

**MQTT协议设计简单轻量、路由灵活，将在移动互联网物联网消息领域，全面取代PC时代的XMPP协议:**

1. MQTT协议一个字节固定报头，两个字节心跳报文，报文体积小编解码容易。XMPP协议基于繁重的XML，报文体积大且交互繁琐。
2. MQTT协议基于主题(Topic)发布订阅模式消息路由，相比XMPP基于JID的点对点消息路由更为灵活。
3. MQTT协议未定义报文内容格式，可以承载JSON、二进制等不同类型报文。XMPP协议采用XML承载报文，二进制必须Base64编码等处理。
4. MQTT协议支持消息收发确认和QoS保证，XMPP主协议并未定义类似机制。MQTT协议有更好的消息可靠性保证。

# MQTT-SN 协议

MQTT-SN 协议是 MQTT 的直系亲属，它使用 UDP 进行通信，标准的端口是1884。MQTT-SN 的主要目的是为了适应受限的设备和网络，比如一些传感器，只有很小的内存和 CPU，TCP 对于这些设备来说非常奢侈。还有一些网络，比如 ZIGBEE，报文的长度在300字节以下，无法承载太大的数据包。所以 MQTT-SN 的数据包更小巧。

MQTT-SN 的官方标准下载地址:

<http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>

## MQTT-SN 和 MQTT 的区别

MQTT-SN 的命令和 MQTT 大部分都相同，比如都有 Will, 都有 Connect/Subscribe/Publish 命令.

MQTT-SN 最大的不同是，Topic 使用 TopicId 来代替，而 TopicId 是一个16比特的数字。每一个数字对应一个 Topic, 设备和云端需要使用 REGISTER 命令映射 TopicId 和 Topic 的对应关系。

MQTT-SN 可以随时更改 Will 的内容, 甚至可以取消. 而 MQTT 只允许在 CONNECT 时设定 Will 的内容, 而且不允许更改.

MQTT-SN 的网络中有网关这种设备，它负责把 MQTT-SN 转换成 MQTT，和云端的 MQTT Broker 通信. MQTT-SN 的协议支持自动发现网关的功能。

MQTT-SN 还支持设备的睡眠功能，如果设备进入睡眠状态，无法接收 UDP 数据，网关将把下行的 PUBLISH 消息缓存起来，直到设备苏醒后再传送。

## EMQ-SN 网关插件

EMQ-SN 是 EMQ 的一个网关插件，实现了 MQTT-SN 的大部分功能，它相当于一个在云端的 MQTT-SN 网关，直接和 EMQ Broker 相连。

配置参数

File: etc/plugins/emq\_sn.conf:

mqtt.sn.port = 1884

mqtt.sn.advertise\_duration = 900

mqtt.sn.gateway\_id = 1

mqtt.sn.username = mqtt\_sn\_user

mqtt.sn.password = abc

启动emq-sn

MQTT-SN 客户端库

<https://github.com/eclipse/paho.mqtt-sn.embedded-c/>

<https://github.com/ty4tw/MQTT-SN>

<https://github.com/njh/mqtt-sn-tools>

<https://github.com/arobenko/mqtt-sn>

# MQTT配置

## EMQ 2.0配置文件

**EMQ 2.0 消息服务器通过 etc/ 目录下配置文件进行设置，主要配置文件包括:**

|  |  |
| --- | --- |
| 配置文件 | 说明 |
| etc/emq.conf | EMQ 2.0 消息服务器配置文件 |
| etc/acl.conf | EMQ 2.0 默认ACL规则配置文件 |
| etc/plugins/\*.conf | EMQ 2.0 各类插件配置文件 |

## EMQ 2.0 环境变量

|  |  |
| --- | --- |
| 参数名 | 描述 |
| node.name | Erlang 节点名称，例如: emq@127.0.0.1 |
| node.cookie | Erlang 分布式节点通信 Cookie |
| node.max\_ports | Erlang 虚拟机最大允许打开文件 Socket 数 |
| listener.tcp.external | MQTT/TCP 监听端口，默认: 1883 |
| listener.ssl.external | MQTT/SSL 监听端口，默认: 8883 |
| listener.ws.external | MQTT/WebSocket 监听端口，默认: 8083 |
| listener.wss.external | MQTT/WebSocket/SSL 监听端口，默认: 8084 |

## EMQ 集群设置

### 集群名称

## Cluster name

cluster.name = emqcl

### 自动发现策略

## Cluster discovery strategy: manual | static | mcast | dns | etcd | k8s

cluster.discovery = manual

### 启用集群自愈

## Cluster Autoheal: on | off

cluster.autoheal = on

### 节点自动清除

自动清除宕机节点:

## Clean down node of the cluster

cluster.autoclean = 5m

## EMQ集群自动发现

EMQ R2.3 版本支持多种策略的节点自动发现与集群:

|  |  |
| --- | --- |
| 策略 | 说明 |
| manual | 手工命令创建集群 |
| static | 静态节点列表自动集群 |
| mcast | UDP 组播方式自动集群 |
| dns | DNS A 记录自动集群 |
| etcd | 通过 etcd 自动集群 |
| k8s | Kubernetes 服务自动集群 |

### manual 手动创建集群

默认配置为手动创建集群，节点通过 ./bin/emqttd\_ctl join <Node> 命令加入:

cluster.discovery = manual

### 基于 static 节点列表自动集群

配置固定的节点列表，自动发现并创建集群:

cluster.discovery = static

##--------------------------------------------------------------------

## Cluster with static node list

cluster.static.seeds = [emq1@127.0.0.1,ekka2@127.0.0.1](mailto:emq1@127.0.0.1,ekka2@127.0.0.1)

### 基于 mcast 组播自动集群

基于 UDP 组播自动发现并创建集群:

cluster.discovery = mcast

##--------------------------------------------------------------------

## Cluster with multicast

cluster.mcast.addr = 239.192.0.1

cluster.mcast.ports = 4369,4370

cluster.mcast.iface = 0.0.0.0

cluster.mcast.ttl = 255

cluster.mcast.loop = on

### 基于 DNS A 记录自动集群

基于 DNS A 记录自动发现并创建集群:

cluster.discovery = dns

##--------------------------------------------------------------------

## Cluster with DNS

cluster.dns.name = localhost

cluster.dns.app = ekka

### 基于 etcd 自动集群

基于 `etcd`\_ 自动发现并创建集群:

cluster.discovery = etcd

##--------------------------------------------------------------------

## Cluster with Etcd

cluster.etcd.server = http://127.0.0.1:2379

cluster.etcd.prefix = emqcl

cluster.etcd.node\_ttl = 1m

### 基于 Kubernetes 自动集群

`Kubernetes`\_ 下自动发现并创建集群:

cluster.discovery = k8s

##--------------------------------------------------------------------

## Cluster with k8s

cluster.k8s.apiserver = http://10.110.111.204:8080

cluster.k8s.service\_name = ekka

## Address Type: ip | dns

cluster.k8s.address\_type = ip

## The Erlang application name

cluster.k8s.app\_name = ekka

## EMQ 节点与 Cookie

**Erlang 节点名称、分布式节点间通信 Cookie:**

## Node name

node.name = emqttd@127.0.0.1

## Cookie for distributed node

node.cookie = emq\_dist\_cookie

**注解:**

Erlang/OTP 平台应用多由分布的 Erlang 节点(进程)组成，每个 Erlang 节点(进程)需指配一个节点名，用于节点间通信互访。 所有互相通信的 Erlang 节点(进程)间通过一个共用的 Cookie 进行安全认证。

## EMQ 节点连接方式

**EMQ 节点基于 Erlang/OTP 平台的 TCPv4, TCPv6 或 TLS 协议连接:**

## Specify the erlang distributed protocol.

##

## Value: Enum

## - inet\_tcp: the default; handles TCP streams with IPv4 addressing.

## - inet6\_tcp: handles TCP with IPv6 addressing.

## - inet\_tls: using TLS for Erlang Distribution.

##

## vm.args: -proto\_dist inet\_tcp

node.proto\_dist = inet\_tcp

## Specify SSL Options in the file if using SSL for Erlang Distribution.

##

## Value: File

##

## vm.args: -ssl\_dist\_optfile <File>

## node.ssl\_dist\_optfile = {{ platform\_etc\_dir }}/ssl\_dist.conf

## Erlang 虚拟机参数

## SMP support: enable, auto, disable

node.smp = auto

## Enable kernel poll

node.kernel\_poll = on

## async thread pool

node.async\_threads = 32

## Erlang Process Limit

node.process\_limit = 256000

## Sets the maximum number of simultaneously existing ports for this system

node.max\_ports = 65536

## Set the distribution buffer busy limit (dist\_buf\_busy\_limit)

node.dist\_buffer\_size = 32MB

## Max ETS Tables.

## Note that mnesia and SSL will create temporary ets tables.

node.max\_ets\_tables = 256000

## Tweak GC to run more often

node.fullsweep\_after = 1000

## Crash dump

node.crash\_dump = log/crash.dump

## Distributed node ticktime

node.dist\_net\_ticktime = 60

## Distributed node port range

## node.dist\_listen\_min = 6000

## node.dist\_listen\_max = 6999

### 主要参数说明

|  |  |
| --- | --- |
| node.process\_limit | Erlang 虚拟机允许的最大进程数，一个 MQTT 连接会消耗2个 Erlang 进程，所以参数值 > 最大连接数 \* 2 |
| node.max\_ports | Erlang 虚拟机允许的最大 Port 数量，一个 MQTT 连接消耗1个 Port，所以参数值 > 最大连接数 |
| node.dist\_listen\_min | Erlang 分布节点间通信使用 TCP 连接端口范围。注: 节点间如有防火墙，需要配置该端口段 |
| node.dist\_listen\_max | Erlang 分布节点间通信使用 TCP 连接端口范围。注: 节点间如有防火墙，需要配置该端口段 |

## 日志参数配置

### console 日志

## Console log. Enum: off, file, console, both

log.console = console

## Console log level. Enum: debug, info, notice, warning, error, critical, alert, emergency

log.console.level = error

## Console log file

## log.console.file = log/console.log

### error 日志

## Error log file

log.error.file = log/error.log

### crash 日志

## Enable the crash log. Enum: on, off

log.crash = on

log.crash.file = log/crash.log

### syslog 日志

## Syslog. Enum: on, off

log.syslog = on

## syslog level. Enum: debug, info, notice, warning, error, critical, alert, emergency

log.syslog.level = error

## MQTT 协议参数配置

### ClientId 最大允许长度

## Max ClientId Length Allowed.

mqtt.max\_clientid\_len = 1024

### MQTT 最大报文尺寸

## Max Packet Size Allowed, 64K by default.

mqtt.max\_packet\_size = 64KB

### 客户端连接闲置时间

**设置 MQTT 客户端最大允许闲置时间(Socket 连接建立，但未收到 CONNECT 报文):**

## Client Idle Timeout (Second)

mqtt.client.idle\_timeout = 30

### 启用客户端连接统计

## Enable client Stats: on | off

mqtt.client.enable\_stats = off

### 强制 GC 设置

## Force GC: integer. Value 0 disabled the Force GC.

mqtt.conn.force\_gc\_count = 100

## 匿名认证与 ACL 文件

### 是否开启匿名认证

**默认开启，允许任意客户端登录:**

## Allow Anonymous authentication

mqtt.allow\_anonymous = true

### 默认访问控制(ACL)文件

**EMQ 支持基于 etc/acl.conf 文件或 MySQL、 PostgreSQL 等插件的访问控制规则。**

## ACL nomatch

mqtt.acl\_nomatch = allow

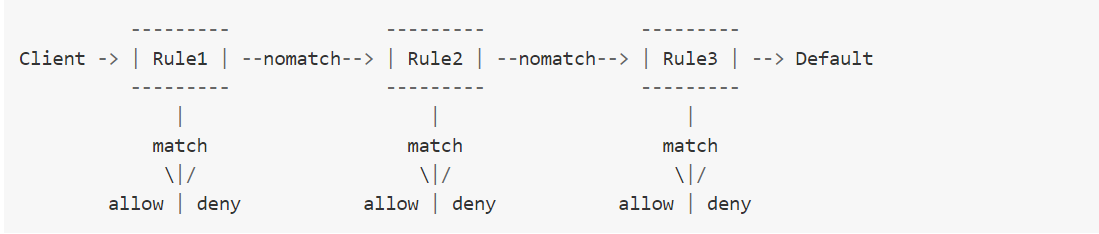
## Default ACL File

mqtt.acl\_file = etc/acl.conf

**etc/acl.conf 访问控制规则定义:**

允许|拒绝 用户|IP地址|ClientID 发布|订阅 主题列表

**访问控制规则采用 Erlang 元组格式，访问控制模块逐条匹配规则:**



**etc/acl.conf 默认访问规则设置:**

%% 允许'dashboard'用户订阅 '$SYS/#'

{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

%% 允许本机用户发布订阅全部主题

{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

%% 拒绝用户订阅'$SYS#'与'#'主题

{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

%% 上述规则无匹配，允许

{allow, all}.

**注解:**

**默认规则只允许本机用户订阅’$SYS/#’与’#’**

EMQ 消息服务器接收到 MQTT 客户端发布(PUBLISH)或订阅(SUBSCRIBE)请求时，会逐条匹配 ACL 访问控制规则，直到匹配成功返回 allow 或 deny。

## MQTT 会话参数设置

##--------------------------------------------------------------------

## MQTT Session

##--------------------------------------------------------------------

## Maximum number of subscriptions allowed, 0 means no limit.

## 最大订阅数限制, 0表示不限制

## Value: Number

mqtt.session.max\_subscriptions = 0

## Force to upgrade QoS according to subscription.

## 根据订阅方 强制更新Qos

## Value: on | off

mqtt.session.upgrade\_qos = off

## Maximum size of the Inflight Window storing QoS1/2 messages delivered but unacked.

## 存储Qos 1/2 已送达,但是未确认的消息数量.

## Value: Number

mqtt.session.max\_inflight = 32

## Retry interval for QoS1/2 message delivering.

##

## Value: Duration

mqtt.session.retry\_interval = 20s

## Maximum QoS2 packets (Client -> Broker) awaiting PUBREL, 0 means no limit.

##

## Value: Number

mqtt.session.max\_awaiting\_rel = 1000

## The QoS2 messages (Client -> Broker) will be dropped if awaiting PUBREL timeout.

##

## Value: Duration

mqtt.session.await\_rel\_timeout = 30s

## Enable per session statistics.

##

## Value: on | off

mqtt.session.enable\_stats = on

## Session expiration time.

##

## Value: Duration

## -d: day

## -h: hour

## -m: minute

## -s: second

##

## Default: 2h, 2 hours

mqtt.session.expiry\_interval = 2h

## Whether to ignore loop delivery of messages.

##

## Value: true | false

##

## Default: false

mqtt.session.ignore\_loop\_deliver = false

## MQTT 消息队列参数设置

**EMQ 消息服务器会话通过队列缓存 Qos1/Qos2 消息:**

1. 持久会话(Session)的离线消息
2. 飞行窗口满而延迟下发的消息

**队列参数设置:**

##--------------------------------------------------------------------

## MQTT Message Queue

##--------------------------------------------------------------------

## Message queue type.

##

## Value: simple | priority

mqtt.mqueue.type = simple

## Topic priority. Default is 0.

##

## Value: Number [0-255]

##

## mqtt.mqueue.priority = topic/1=10,topic/2=8

## Maximum queue length. Enqueued messages when persistent client disconnected,

## or inflight window is full. 0 means no limit.

##

## Value: Number >= 0

mqtt.mqueue.max\_length = 1000

## Low-water mark of queued messages.

##

## Value: Percent

mqtt.mqueue.low\_watermark = 20%

## High-water mark of queued messages.

##

## Value: Percent

mqtt.mqueue.high\_watermark = 60%

## Whether to enqueue Qos0 messages.

##

## Value: false | true

mqtt.mqueue.store\_qos0 = true

**队列参数说明:**

|  |  |
| --- | --- |
| mqtt.mqueue.type | 队列类型。simple: 简单队列，priority: 优先级队列 |
| mqtt.mqueue.priority | 主题(Topic)队列优先级设置 |
| mqtt.mqueue.max\_length | 队列长度, 0 表示不限制 |
| mqtt.mqueue.low\_watermark | 解除告警水位线 |
| mqtt.mqueue.high\_watermark | 队列满告警水位线 |
| mqtt.mqueue.store\_qos0 | 是否缓存 QoS0 消息 |

## Broker 参数设置

**broker\_sys\_interval 设置系统发布 $SYS 消息周期:**

## System interval of publishing $SYS messages.

##

## Value: Duration

##

## Default: 1m, 1 minute

mqtt.broker.sys\_interval = 1m

## 发布订阅(PubSub)参数设置

## The PubSub pool size. Default value should be same as scheduler numbers.

##

## Value: Number > 1

mqtt.pubsub.pool\_size = 8

## TODO: Subscribe asynchronously.

##

## Value: true | false

mqtt.pubsub.async = true

## 桥接(Bridge)参数设置

##--------------------------------------------------------------------

## MQTT Bridge

##--------------------------------------------------------------------

## The pending message queue size of bridge.

##

## Value: Number

mqtt.bridge.max\_queue\_len = 10000

## Ping interval of bridge node.

##

## Value: Duration

##

## Default: 1s, 1 second

mqtt.bridge.ping\_down\_interval = 1s

## 插件(Plugin) 配置目录设置

##-------------------------------------------------------------------

## MQTT Plugins

##-------------------------------------------------------------------

## The etc dir for plugins' config.

##

## Value: Folder

mqtt.plugins.etc\_dir =etc/plugins/

## The file to store loaded plugin names.

##

## Value: File

mqtt.plugins.loaded\_file = data/loaded\_plugins

## MQTT Listeners 参数说明

**EMQ 消息服务器支持 MQTT、MQTT/SSL、MQTT/WS 协议服务端，可通过 listener.tcp|ssl|ws|wss|.\* 设置端口、最大允许连接数等参数。**

**EMQ 2.2 消息服务器默认开启的 TCP 服务端口包括:**

|  |  |
| --- | --- |
| 1883 | MQTT 协议端口 |
| 8883 | MQTT/SSL 端口 |
| 8083 | MQTT/WebSocket 端口 |
| 8080 | HTTP 管理 API 端口 |
| 8084 | MQTT/WebSocket/SSL 端口 |

**Listener 参数说明:**

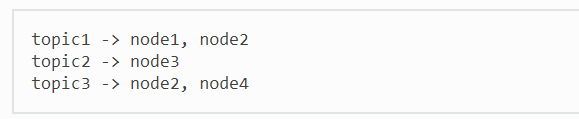
|  |  |
| --- | --- |
| listener.tcp.${name}.acceptors | TCP Acceptor 池 |
| listener.tcp.${name}.max\_clients | 最大允许 TCP 连接数 |
| listener.tcp.${name}.rate\_limit | 连接限速配置，例如限速10KB/秒: “100,10” |

# EMQ分布式集群

**EMQ消息服务器集群基于Erlang/OTP分布式设计，集群原理可简述为下述两条规则:**

1. MQTT客户端订阅主题时，所在节点订阅成功后广播通知其他节点：某个主题(Topic)被本节点订阅。
2. MQTT客户端发布消息时，所在节点会根据消息主题(Topic)，检索订阅并路由消息到相关节点。

**EMQ消息服务器同一集群的所有节点，都会复制一份主题(Topic) -> 节点(Node)映射的路由表，例如:**



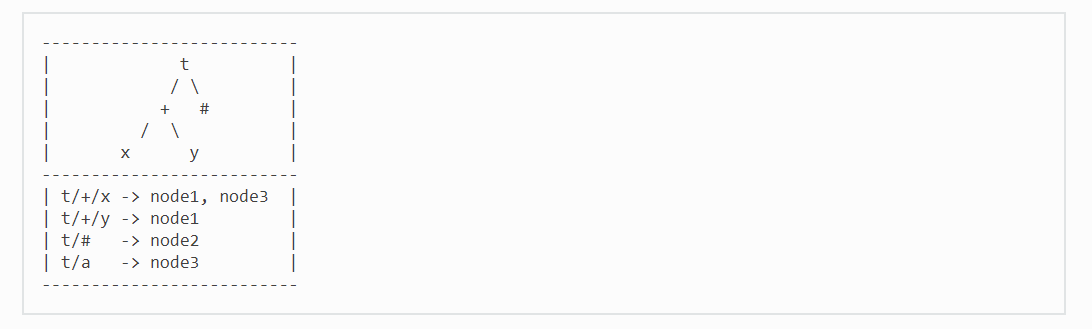
## 主题树(Topic Trie)与路由表(Route Table)

EMQ消息服务器每个集群节点，都保存一份主题树(Topic Trie)和路由表。

例如下述主题订阅关系:

|  |  |  |
| --- | --- | --- |
| 客户端 | 节点 | 订阅主题 |
| client1 | node1 | t/+/x, t/+/y |
| client2 | node2 | t/# |
| client3 | node3 | t/+/x, t/a |

**最终会生成如下主题树(Topic Trie)和路由表(Route Table):**



## 订阅(Subscription)与消息派发

**客户端的主题订阅(Subscription)关系，只保存在客户端所在节点，用于本节点内派发消息到客户端。**

**例如client1向主题’t/a’发布消息，消息在节点间的路由与派发流程:**

title: Message Route and Deliver

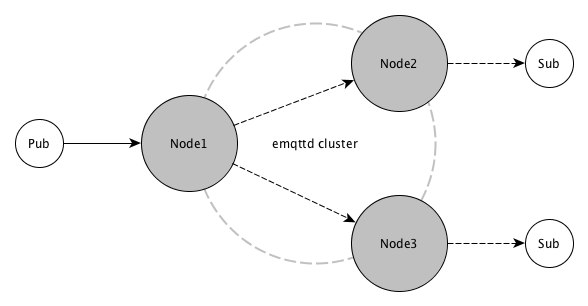
client1->node1: Publish[t/a]

node1-->node2: Route[t/#]

node1-->node3: Route[t/a]

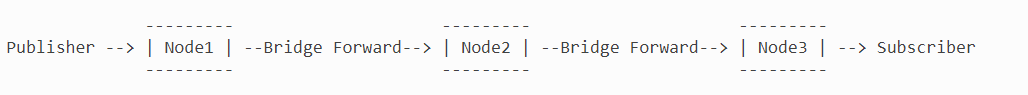
node2-->client2: Deliver[t/#]

node3-->client3: Deliver[t/a]



# EMQ节点桥接

## 桥接模式



节点间桥接与集群不同，不复制主题树与路由表，只按桥接规则转发MQTT消息。

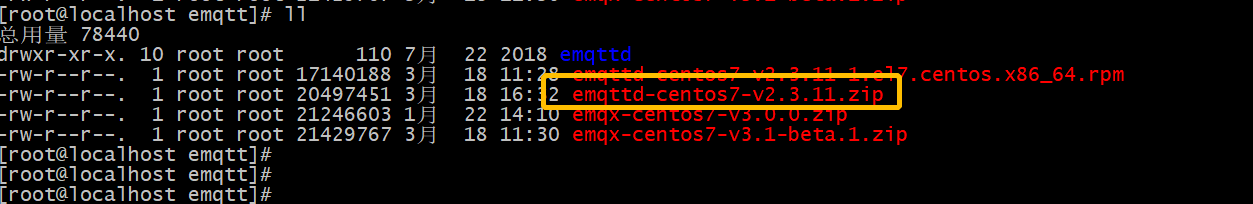
# EMQ安装部署

我这里是选择 安装包安装, 还可以 docker, rpm 等安装方式.

## 安装启动

### 下载版本 v2.3.11

下载地址: <http://www.emqtt.com/downloads>



### 虚拟机准备

|  |  |
| --- | --- |
| emq | 192.168.188.131, 192.168.188.132 |
| haproxy | 192.168.188.133 |

### 解压安装包

131, 132两台虚拟机分别解压安装包到对应安装目录.

unzip emqttd-centos7-v2.3.11.zip

### 配置参数

#### Erlang虚拟机参数

##vim emqttd/etc/emq.conf

#节点名称

node.name = emq@192.168.188.131

#Erlang 虚拟机允许的最大进程数，EMQ 一个连接会消耗2个Erlang进程,

#参数值 > 最大允许连接数 \* 2

node.process\_limit = 2097152

#Erlang 虚拟机允许的最大 Port 数量，EMQ 一个连接消耗1个 Port.

#Erlang 的 Port 非 TCP 端口，可以理解为文件句柄。

#参数值 > 最大允许连接数

node.max\_ports = 1048576

#### EMQ消息服务器参数

##vim emqttd/etc/emq.conf

listener.tcp.external = 0.0.0.0:1883

#设置 TCP 监听器的 Acceptor 池大小

listener.tcp.external.acceptors = 64

#emq最大允许连接数

listener.tcp.external.max\_clients = 1000000

#### linux操作系统配置

1. vim /etc/sysctl.conf

#系统全局允许分配的最大文件句柄数

fs.file-max=2097152

fs.nr\_open=2097152

#并发连接 backlog 设置

net.core.somaxconn=32768

net.ipv4.tcp\_max\_syn\_backlog=16384

net.core.netdev\_max\_backlog=16384

#可用知名端口范围

net.ipv4.ip\_local\_port\_range='1000 65535'

#TCP Socket 读写 Buffer 设置

net.core.rmem\_default=262144

net.core.wmem\_default=262144

net.core.rmem\_max=16777216

net.core.wmem\_max=16777216

net.core.optmem\_max=16777216

net.ipv4.tcp\_rmem='1024 4096 16777216'

net.ipv4.tcp\_wmem='1024 4096 16777216'

#TCP 连接追踪设置

net.nf\_conntrack\_max=1000000

net.netfilter.nf\_conntrack\_max=1000000

net.netfilter.nf\_conntrack\_tcp\_timeout\_time\_wait=30

#TIME-WAIT Socket 最大数量、回收与重用设置

net.ipv4.tcp\_max\_tw\_buckets=1048576

# 注意: 不建议开启該设置，NAT模式下可能引起连接RST

# net.ipv4.tcp\_tw\_recycle = 1

# net.ipv4.tcp\_tw\_reuse = 1

#FIN-WAIT-2 Socket 超时设置

net.ipv4.tcp\_fin\_timeout = 15

2. vim /etc/systemd/system.conf

DefaultLimitNOFILE=1048576

3. vim /etc/security/limits.conf

#持久化设置允许用户/进程打开文件句柄数

\* soft nofile 1048576

\* hard nofile 1048576

#### 其他配置

echo 2097152 > /proc/sys/fs/nr\_open

#允许当前会话/进程打开文件句柄数

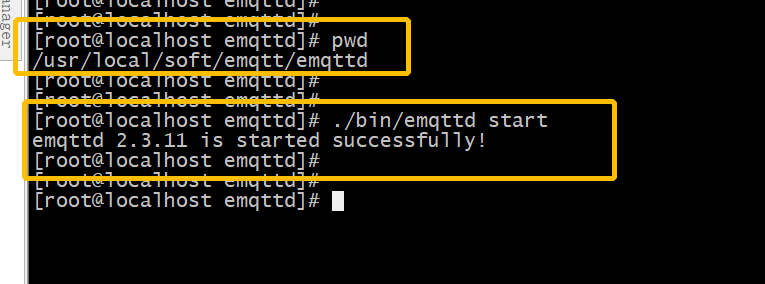
ulimit -n 1048576

### 启动emq集群

131,132两台虚拟机修改好配置

#### 启动单节点

./bin/emqttd start



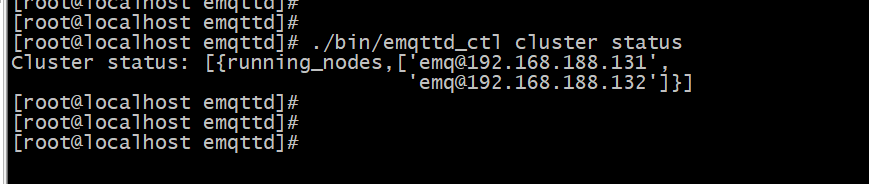
#### 加入emq集群

在132节点 执行. (也可以在131节点执行)

./bin/emqttd\_ctl cluster join [emq@192.168.188.131](mailto:emq@192.168.188.131)

#### 查看集群状态

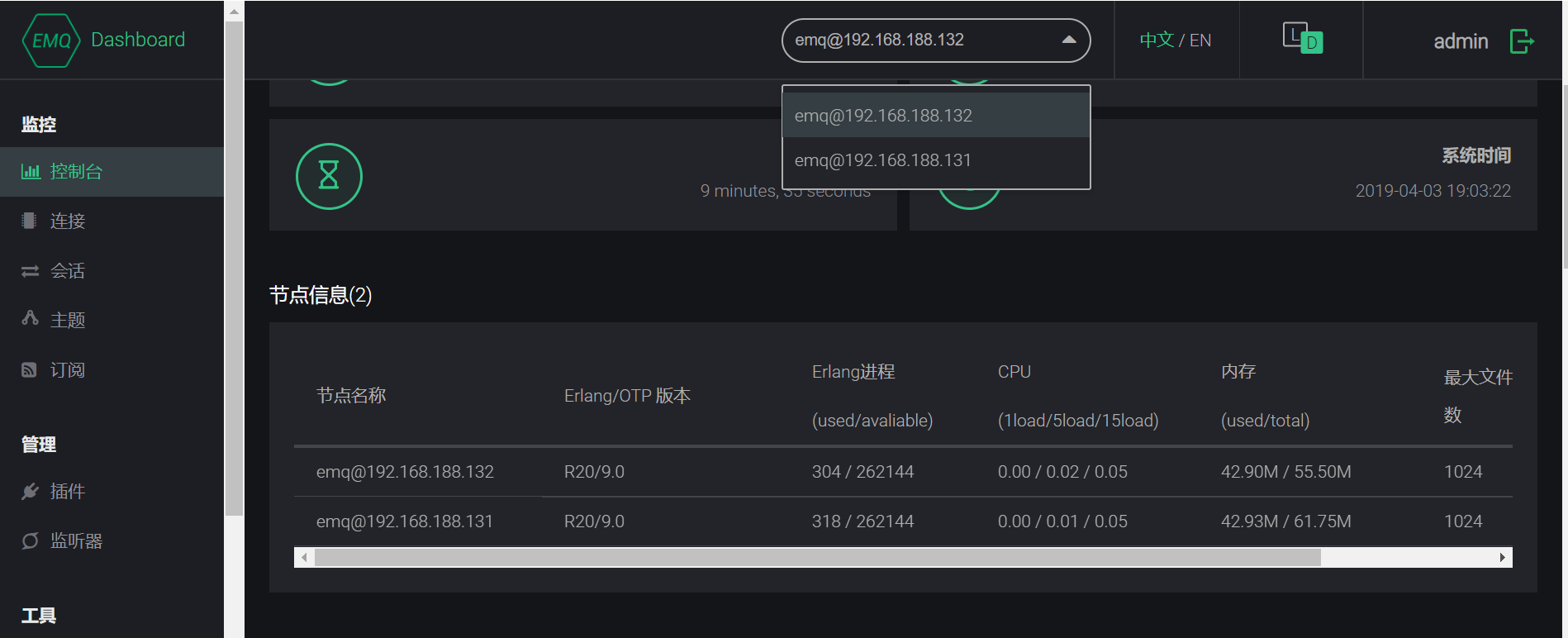
./bin/emqttd\_ctl cluster status



#### web控制台

默认web控制台端口为18083 账号 admin /public

<http://192.168.188.131:18083/#/>



# 安装Haproxy

根据官网私有云部署架构推荐, 使用haproxy 做 负载均衡.

在192.168.188.133安装好 haproxy 并做好对应配置.

监听tcp端口为 1883

# 压测工具

## 依赖安装

### erlang依赖

yum -y install make gcc gcc-c++ kernel-devel m4 ncurses-devel openssl-devel unixODBC-devel

### 安装erlang

#### 下载解压

wget [http://erlang.org/download/](http://erlang.org/download/otp_src_21.0.tar.gz)otp\_src\_20.0.tar.gz

tar -xvzf otp\_src\_21.0.tar.gz

#### 配置

进入 解压缩目录

./configure --prefix=/usr/local/erlang --with-ssl -enable-threads -enable-smmp-support -enable-kernel-poll --enable-hipe --without-javac

#### 安装

make && make install

#### 配置profile

vim /etc/profile

export ERLANG\_HOME=/usr/local/erlang

export PATH=$PATH:$JAVA\_HOME/bin:$ERLANG\_HOME/bin

使配置生效

source /etc/profile

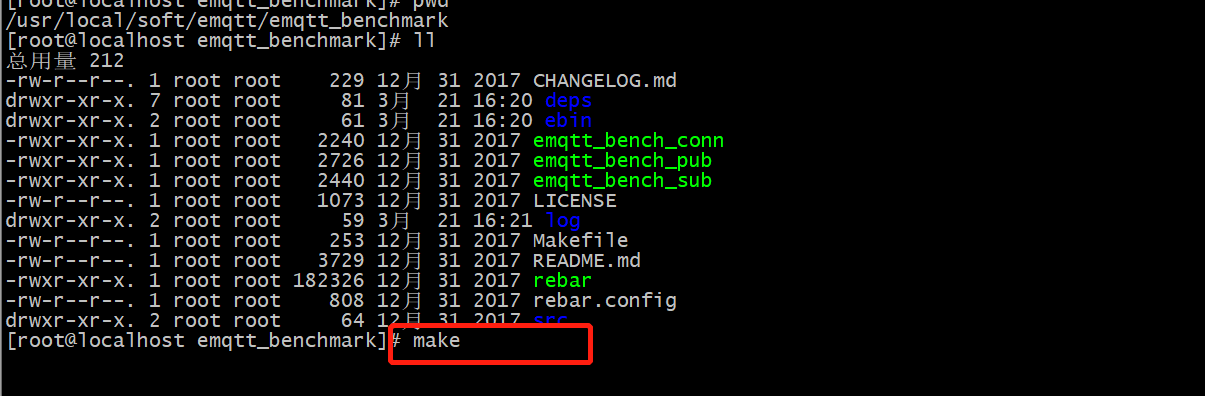
## 安装测试工具

### 从github上下载源码

<https://github.com/emqtt/emqtt_benchmark/tree/benchmark>

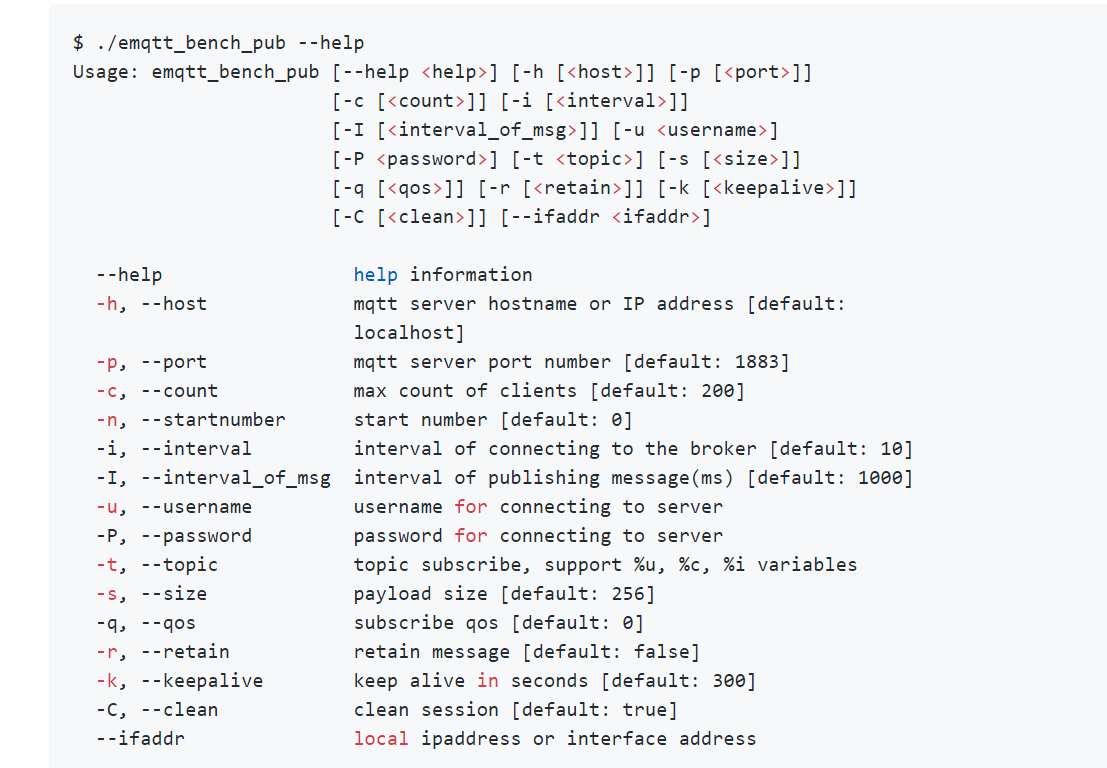
解压到对应目录 进入对应目录

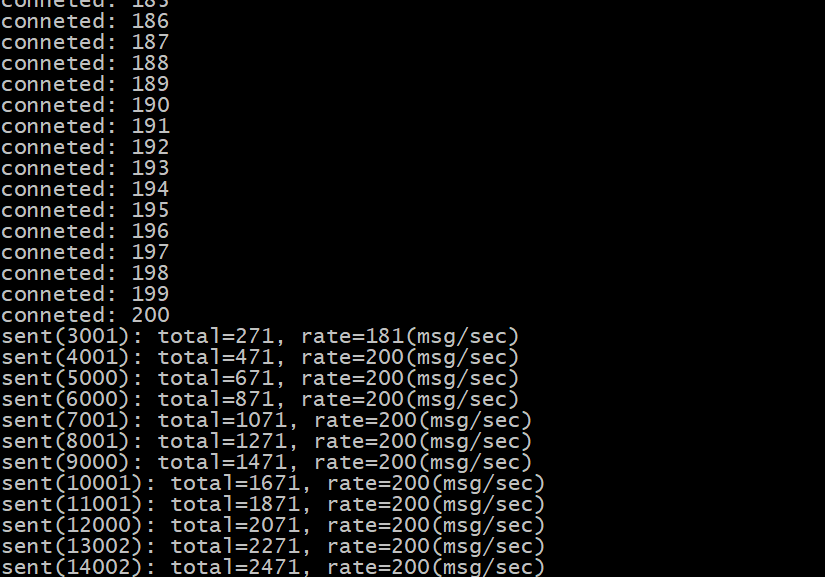
执行 make (这里对erlang版本有要求, 20.0版本的才能用)



make执行完之后

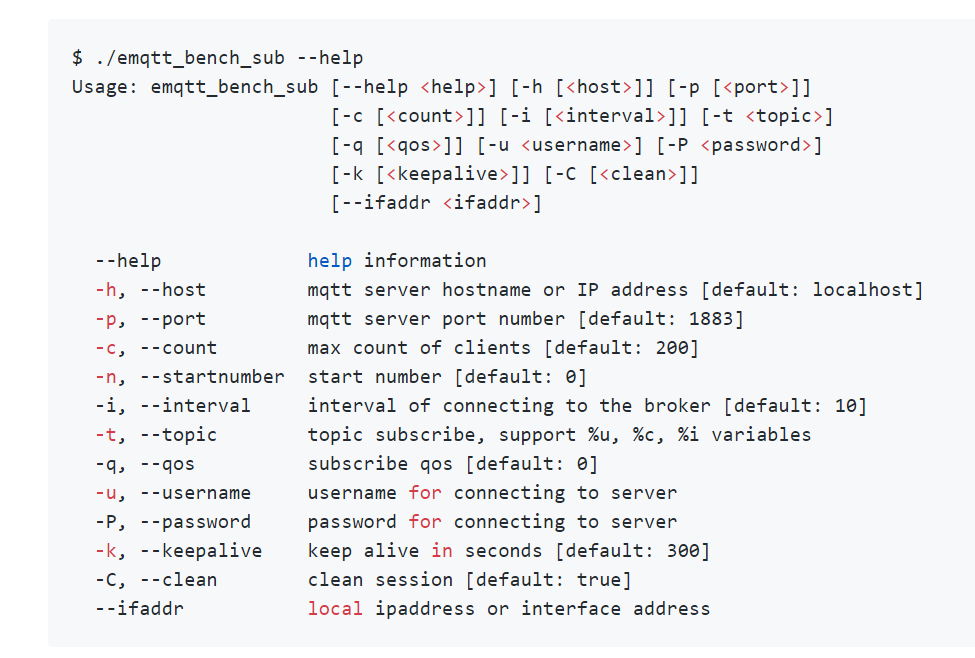
### 发布测试指令

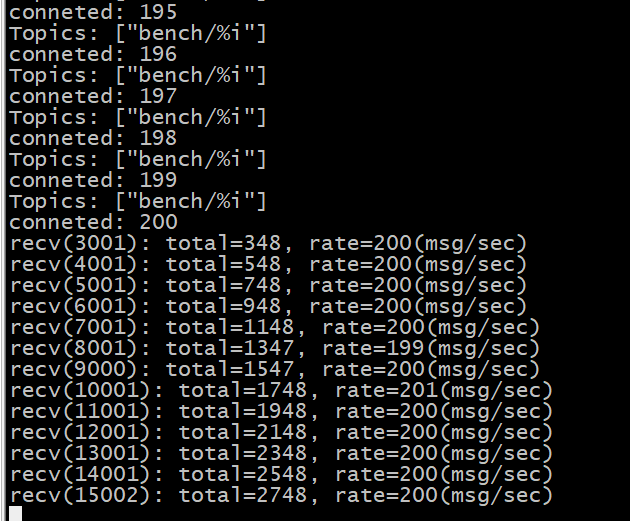
./emqtt\_bench\_pub -h 192.168.188.133 -c 200 -i 10 -t bench/%i



### 订阅测试指令

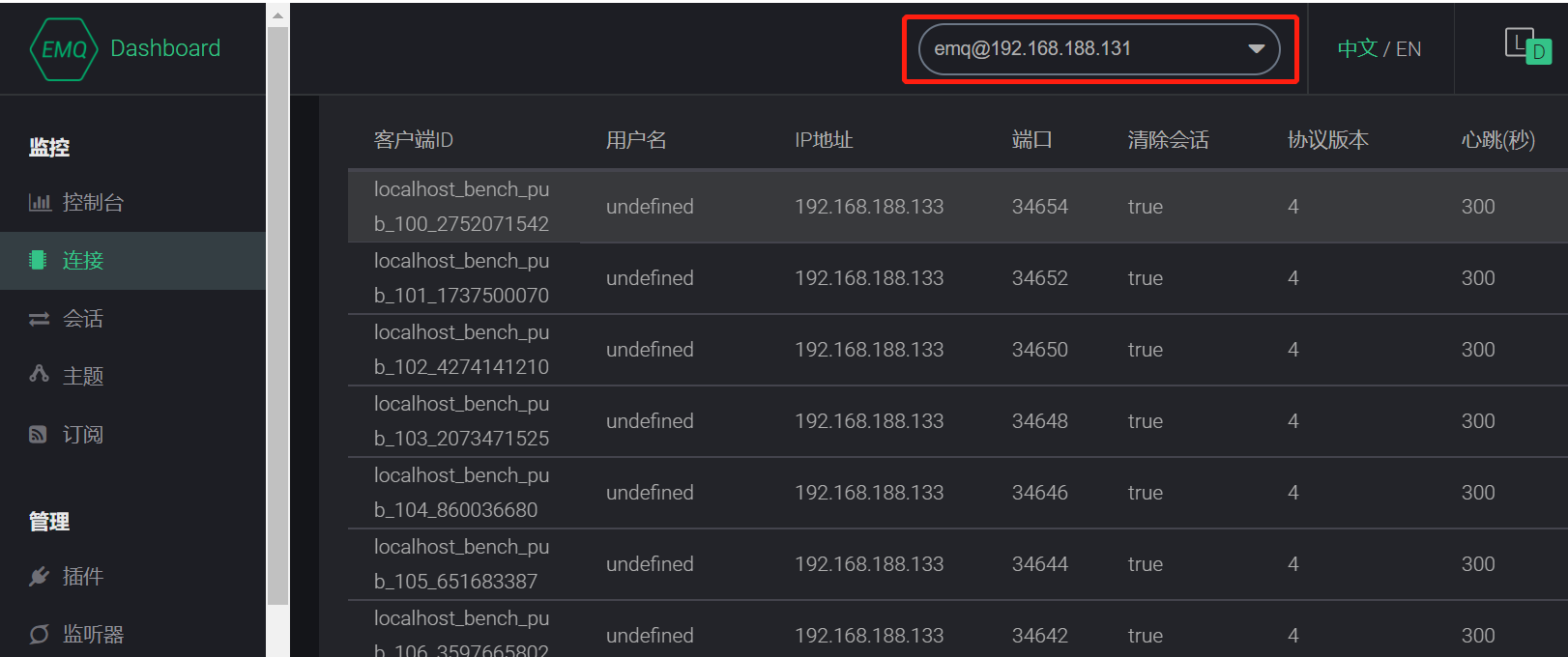
./emqtt\_bench\_sub –h 192.168.188.133 –c 200 –i 10 –t bench/%i





### 管理控制后台

Haproxy balance 配置为source 根据请求源的ip 所有请求都路由到 131节点.





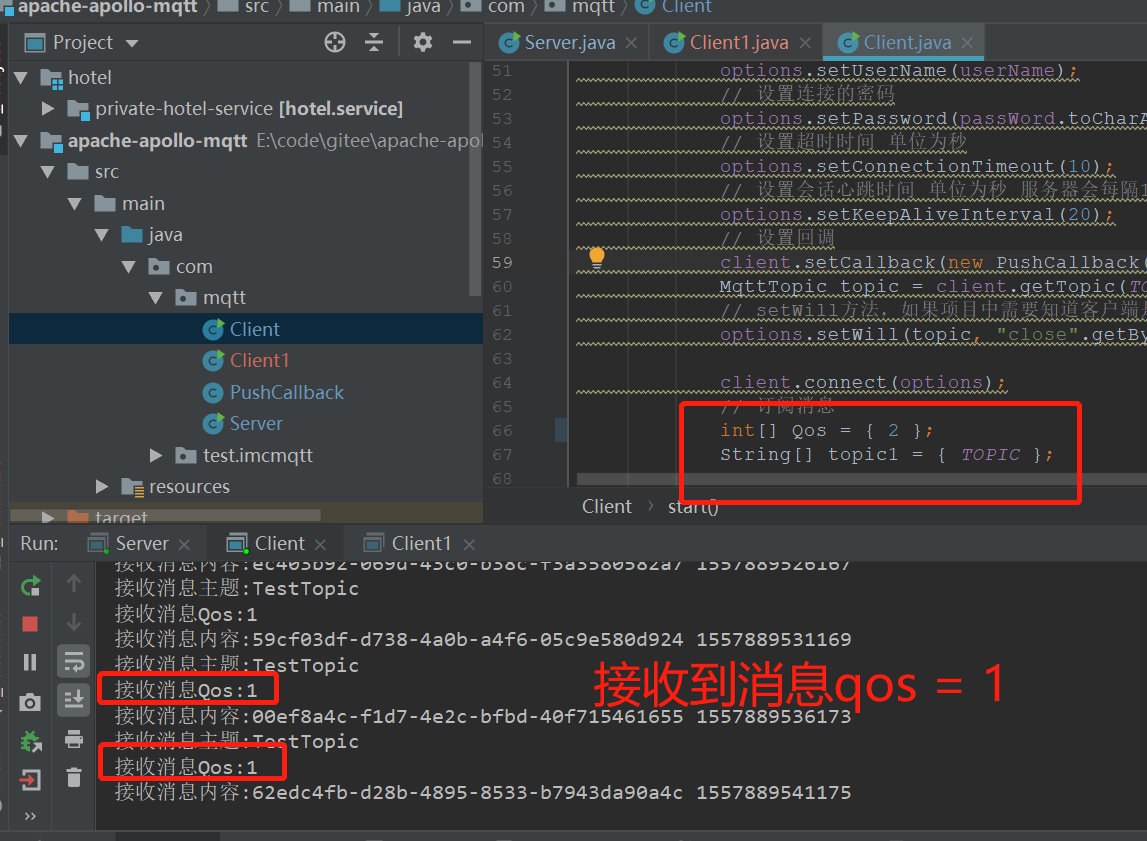
# QA

Q: Qos 012 以哪个为主

1. 以消息发布方 的qos 为准. 发布方设置Q0S为 1, 订阅者设置qos为2. 最终订阅方收到的消息 qos级别还是 1. (已验证)



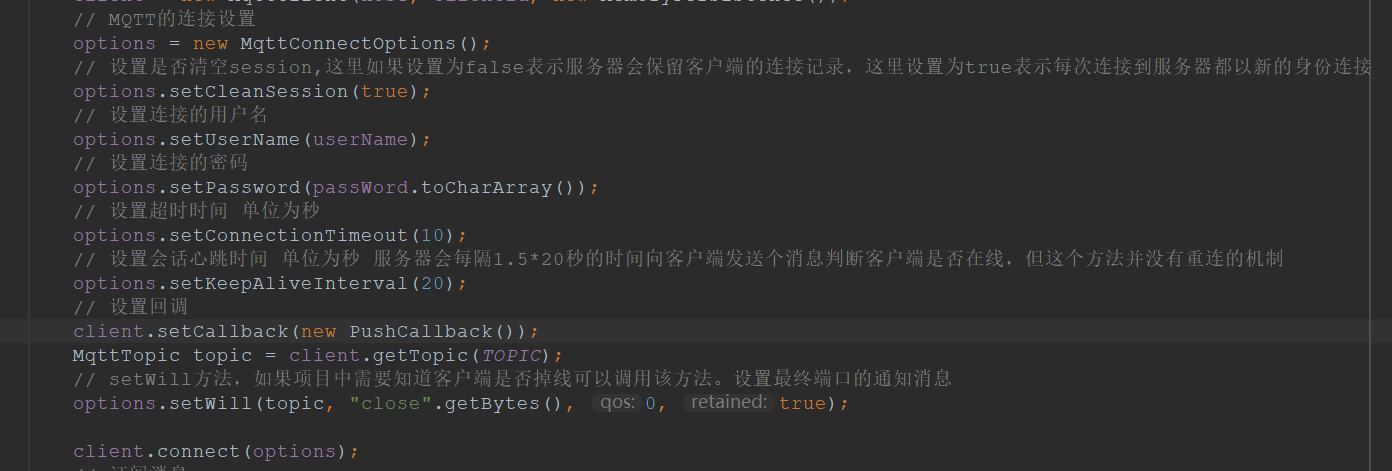




2. 具体使用 哪一个qos级别 看具体应用场景.

Q: 会话， 客户端怎么设置

客户端连接会话参数







….

Q: 遗愿消息

见 QA 9.2

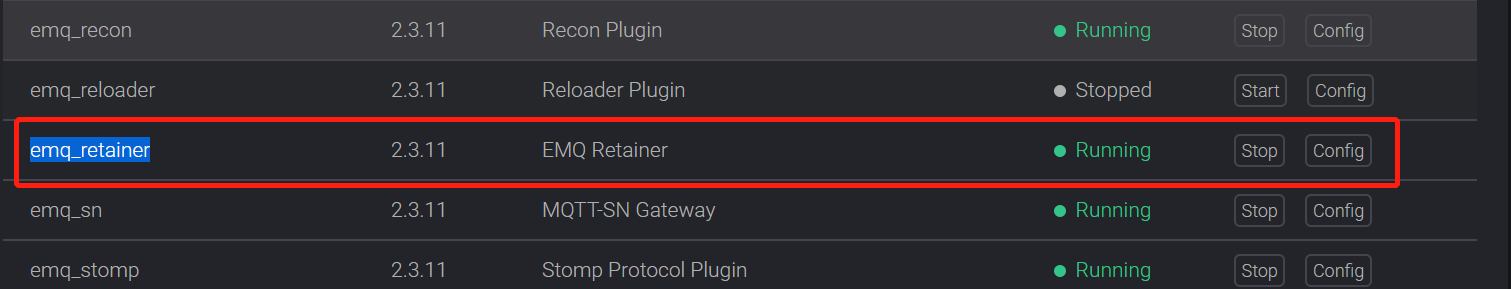
Q: 遗愿消息,保留消息 多长时间被消费掉

1. 遗愿消息 是在 与EMQ服务端建立连接时 设置的, 我认为在 服务端 通过心跳机制 检测到该 连接已经中断, 然后 由服务端 将遗愿消息发布到对应的topic.

options.setWill(topic, "遗愿消息".getBytes(), 0, true);

1. Retained消息保留是 由 emq\_retainer Retainer模块插件控制

该插件默认加载.



对应 配置文件 在 etc/plugins/emq\_retainer.conf

## disc: disc\_copies, ram: ram\_copies 存储在 硬盘 还是 内存

## Notice: retainer's storage\_type on each node in a cluster must be the same!

集群环境 所有节点 该参数要配置一致.

retainer.storage\_type = disc

## Max number of retained messages 最大条数

retainer.max\_message\_num = 1000000

## Max Payload Size of retained message 大小限制.

retainer.max\_payload\_size = 64KB

## Expiry interval. Never expired if 0 到期时间设置, 默认0 永久存储.

## h - hour

## m - minute

## s - second

retainer.expiry\_interval = 0

Q: 集群最小几台，集群如果掉一台，数据安全性

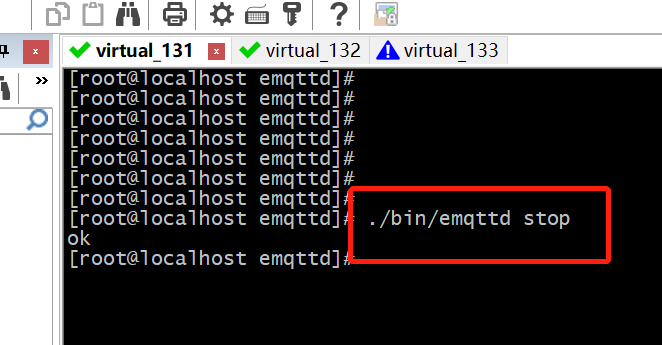
应该没有最小几台限制.

节点宕机 测试.

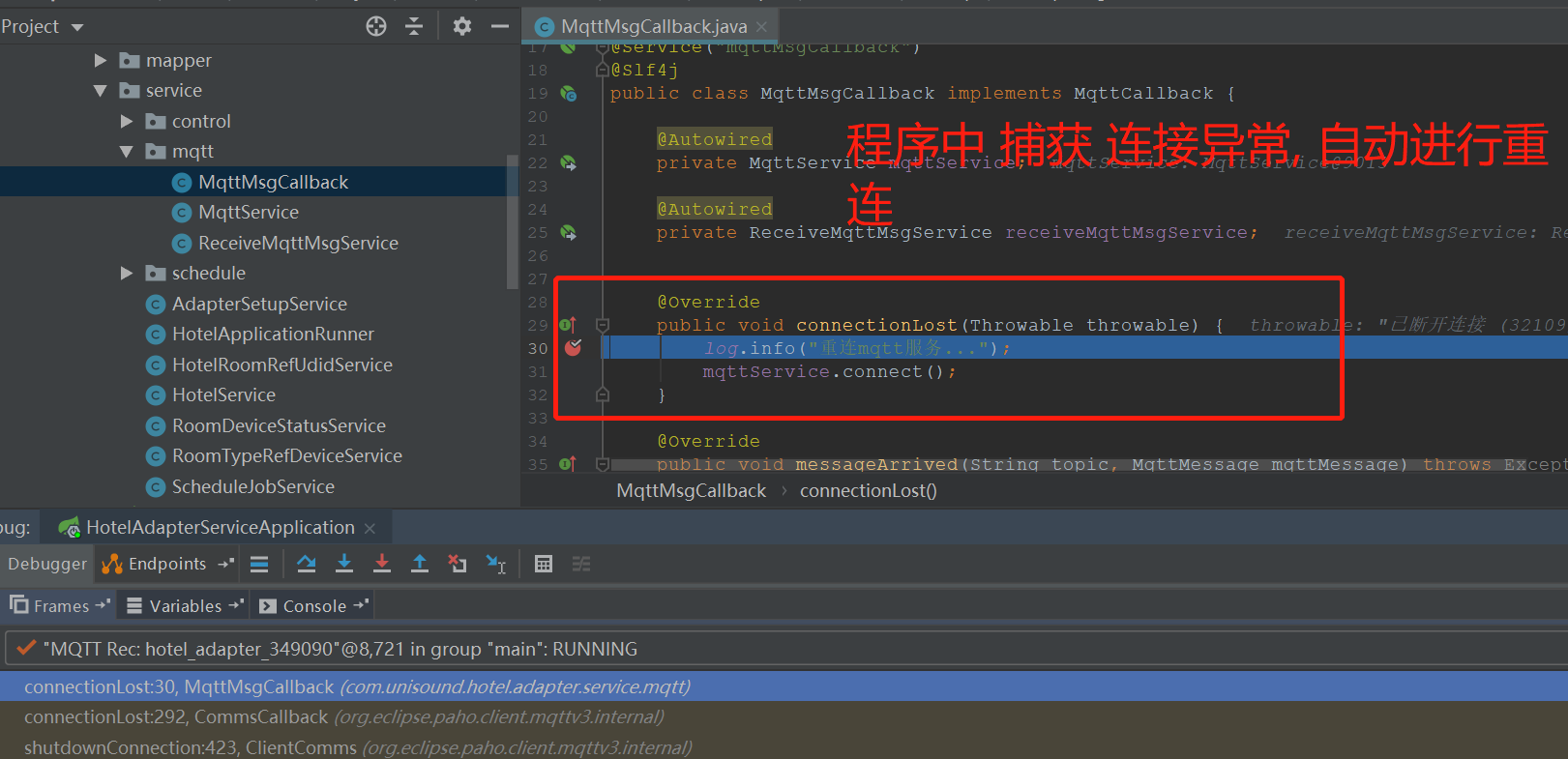
1. 启动两个节点 131, 132
2. 
3. 客户端启动通过haproxy 路由 连接到 节点



1. 关闭节点 131



1. 应用程序捕获到 连接中断 执行重连程序.



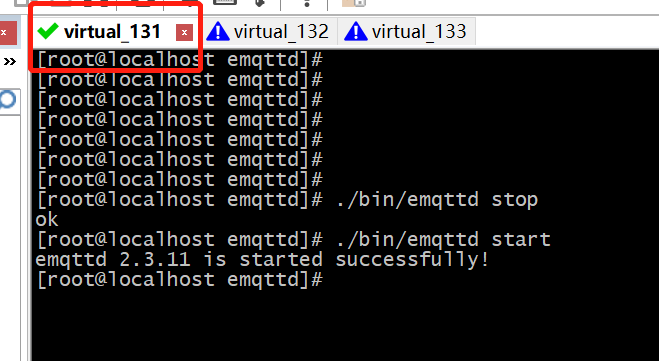
1. 重连emq服务 到132节点

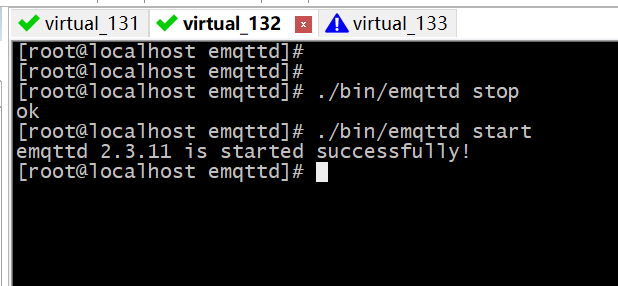


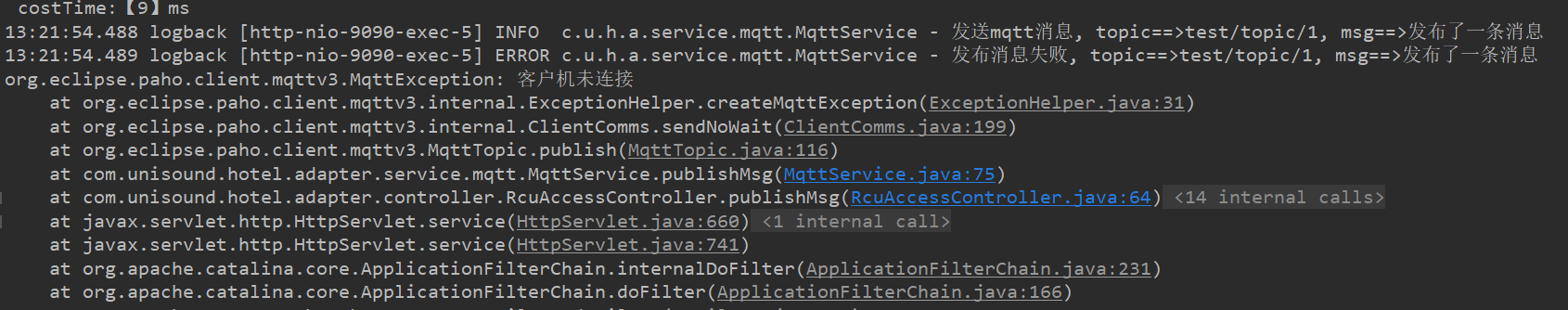
**虽然此时 客户端重新连上 emq 132节点, 但是 之前的 连接 相关的 订阅相关的 信息 会失效. 所以 在重连的逻辑那里 要加上 重新订阅相关主题的逻辑.**

1. **特殊情况 如果 当前节点 131, 132同时都挂了, 等五分钟 之后. 再重启两个节点. 此时客户端 不会自动连接emq服务. 只能重新启动客户端服务.**

**所以 客户端 最好有定时轮询逻辑, 定时检测 连接是否正常.**







Q: 集群自愈 解释，跟集群发现有什么区别

1. 集群自愈: 当前 有131, 132两个运行的节点. 131突然停止, 重启后 会自动加入集群.
2. 集群发现: 集群发现 是 动态添加emq节点, 自动加入集群机制. 对应策略 见 3.4

## 节点cookie配置 - 苏亮

Q: 1.1. Qos 2 精准模式时，订阅端保存消息是怎么保存的，需要终端来实现保存还是 emqtt以某种机制存储

订阅端 有相应的mqtt sdk 都是基于mqtt协议开发的. 见 1.10.1

Q: 存储机制，删除机制

Emq服务实现. 配置见QA 9.4

Q: 4.1 节点挂掉时，消息怎么路由，主题树是怎样的;

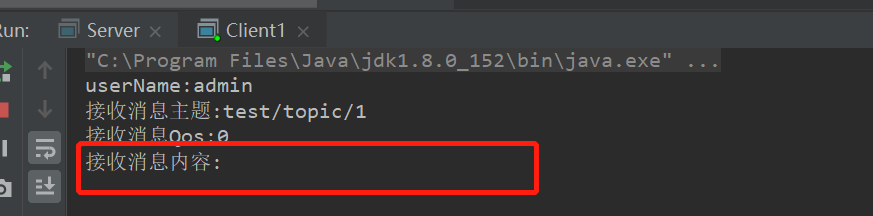
类似集群自愈, 与集群发现机制. 各节点通过广播方式同步 主题树, 路由表数据

## 

Q: 1.8 正常情况，发一条空消息，是新增一条还是替换之前的消息

经过验证, 发送空消息, 订阅方收到的也是空消息.





Q: 如果客户端重启，会不能再收到消息. (session false 看 timeout？) 要看客户端重启后 连接emq 使用的clientId 是否和启动前一致.

还要看最开始连接服务端 设置的 cleanSession 是否为 false. 如果在服务端

超时注销 连接之前 重连到emq服务. 那么 会继续收到消息, 因为 订阅主题相关的数据 是保存在emq服务端.

Q: 保留消息： 超时时间 是针对于全局的 还是别的什么

见 QA 5

Q: 14. 数据一致性策略， 故障转移策略

见 QA 5