EMQ - 百万级开源MQTT消息服务器

发布 ***2.2-beta.1***

杭州映云科技有限公司 **<contact@emqtt.io>**

2018 年 **07** 月 **02** 日

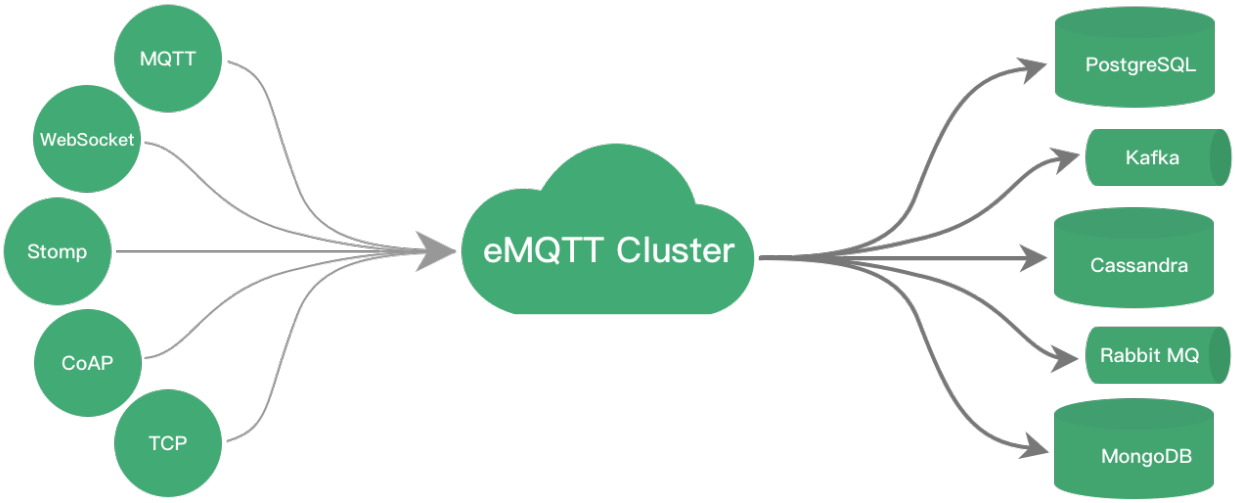
Contents

EMQ 2.0 (Erlang/Enterprise/Elastic MQTT Broker) 是基于 Erlang/OTP 语言平台开发，支持大规模连接和分布式集群，发布订阅模式的开源 MQTT 消息服务器。

注解： 2.0 版本开始 emqttd 消息服务器自正式简称为 EMQ

EMQ 2.0 完整支持 MQTT V3.1/V3.1.1 版本协议规范，并扩展支持 WebSocket、Stomp、CoAP、MQTT-SN 或私有 TCP 协议。*EMQ* 2.0 消息服务器支持单节点100万连接与多节点分布式集群:

TODO: 2.0-rc.1 图片更新.



EMQ 2.0 为大规模客户端连接 (C1000K+) 的移动推送、移动消息、物联网、车联网、智能硬件等应用，提供一个完全开放源码、安装部署简便、企业级稳定可靠、可弹性扩展、易于定制开发的 MQTT 消息服务器。

注解： MQTT-SN、CoAP 协议已在2.0-rc.1版本发布，LWM2M、LoRaWan 协议在2.3-beta.1版本发布。

EMQ 2.0 项目文档目录:

CHAPTER 1

开始使用 (Get Started)

1.1 *EMQ* 2.0 消息服务器简介

EMQ (Erlang/Enterprise/Elastic MQTT Broker) 是基于 Erlang/OTP 平台开发的开源物联网 MQTT 消息服务器。Erlang/OTP 是出色的软实时(Soft-Realtime)、低延时(Low-Latency)、分布式(Distributed) 的语言平台。MQTT 是轻量的(Lightweight)、发布订阅模式(PubSub) 的物联网消息协议。

EMQ 项目设计目标是承载移动终端或物联网终端海量 MQTT 连接，并实现在海量物联网设备间快速低延时消息路由:

1. 稳定承载大规模的 MQTT 客户端连接，单服务器节点支持50万到100万连接。
2. 分布式节点集群，快速低延时的消息路由，单集群支持1000万规模的路由。
3. 消息服务器内扩展，支持定制多种认证方式、高效存储消息到后端数据库。
4. 完整物联网协议支持，MQTT、MQTT-SN、CoAP、WebSocket 或私有协议支持。

1.2 MQTT 发布订阅模式简述

MQTT 是发布订阅(Publish/Subscribe) 模式的消息协议，与 HTTP 协议请求响应(Request/Response) 模式不同。

MQTT 发布者与订阅者之间通过”主题”(Topic) 进行消息路由，主题(Topic) 格式类似 Unix 文件路径，例如:

```
sensor/1/temperature

chat/room/subject

presence/user/feng

sensor/1/#
```

(continues on next page)

(续上页)

```
sensor/+/temperature

uber/drivers/joe/inbox
```

MQTT 主题(Topic) 支持'+', '#'的通配符，'+'通配一个层级，'#'通配多个层级(必须在末尾)。

MQTT 消息发布者(Publisher) 只能向特定'名称主题'(不支持通配符)发布消息，订阅者(Subscriber)通过订阅'过滤主题'(支持通配符)来匹配消息。

注解: 初接触MQTT协议的用户，通常会向通配符的'过滤主题'发布广播消息，MQTT 协议不支持这种模式，需从订阅侧设计广播主题(Topic)。例如 Android 推送，向所有广州用户，推送某类本地消息，客户端获得 GIS 位置后，可订阅 'news/city/guangzhou' 主题。

1.3 五分钟下载启动 EMQ

EMQ 2.0 消息服务器每个版本，会发布 Ubuntu、CentOS、FreeBSD、Mac OS X、Windows 平台程序包与 Docker 镜像。

下载地址: http://emqtt.com/downloads

程序包下载后，可直接解压启动运行，例如 Mac 平台:

```
unzip emqttd-macosx-v2.0.zip && cd emqttd

# 启动emqttd
./bin/emqttd start

# 检查运行状态
./bin/emqttd_ctl status

# 停止emqttd
./bin/emqttd stop
```

EMQ 消息服务默认允许匿名认证，启动后 MQTT 客户端可连接1883端口，启动运行日志输出在 log/ 目录。

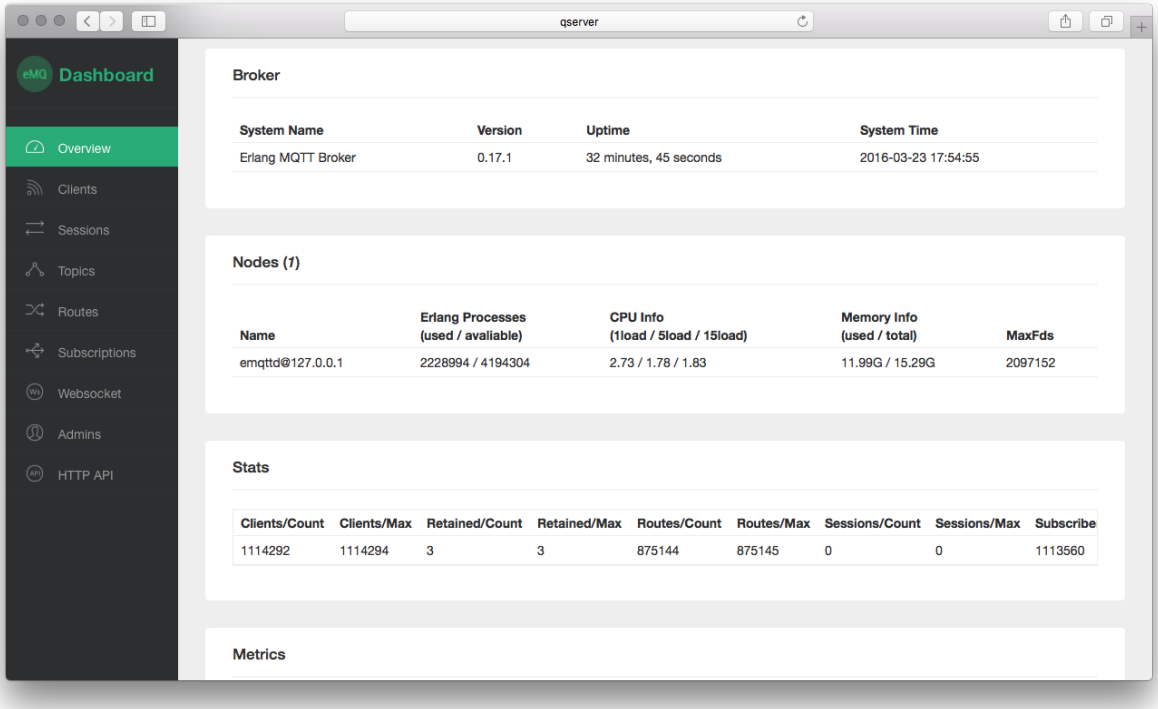
1.4 源码编译EMQ 2.0

```
git clone https://github.com/emqtt/emq-relx.git

cd emq-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

1.5 Web 管理控制台(Dashboard)

EMQ 消息服务器启动后，会默认加载 Dashboard 插件，启动 Web 管理控制台。用户可通过 Web 控制台，查看服务器运行状态、统计数据、客户端(Client)、会话(Session)、主题(Topic)、订阅(Subscription)、插件(Plugin)。

控制台地址: http://127.0.0.1:18083，默认用户: admin，密码：public



1.6 *EMQ* 2.0 消息服务器功能列表

- 完整的 MQTT V3.1/V3.1.1 协议规范支持
- QoS0, QoS1, QoS2 消息支持
- 持久会话与离线消息支持
- Retained 消息支持
- Last Will 消息支持
- TCP/SSL 连接支持
- MQTT/WebSocket/SSL 支持
- HTTP消息发布接口支持
- $SYS/# 系统主题支持
- 客户端在线状态查询与订阅支持
- 客户端 ID 或 IP 地址认证支持
- 用户名密码认证支持
- LDAP 认证
- Redis、MySQL、PostgreSQL、MongoDB、HTTP 认证集成
- 浏览器 Cookie 认证

- 基于客户端 ID、IP 地址、用户名的访问控制(ACL)
- 多服务器节点集群(Cluster)
- 多服务器节点桥接(Bridge)
- mosquitto 桥接支持
- Stomp 协议支持
- MQTT-SN 协议支持
- CoAP 协议支持
- Stomp/SockJS 支持
- 通过 Paho 兼容性测试
- 2.0新功能: 本地订阅($local/topic)
- 2.0新功能: 共享订阅($share/<group>/topic)
- 2.0新功能: sysctl 类似 k = v 格式配置文件

1.7 *EMQ* 2.0 扩展插件列表

EMQ 2.0 支持丰富的扩展插件，包括控制台、扩展模块、多种认证方式、多种接入协议等:

emq_plugin_template	插件模版与演示代码
emq_retainer	Retain 消息存储插件
emq_modules	Presence, Subscription 扩展模块插件
emq_dashboard	Web 管理控制台，默认加载
emq_auth_clientid	ClientId、密码认证插件
emq_auth_username	用户名、密码认证插件
emq_auth_ldap	LDAP 认证插件
emq_auth_http	HTTP 认证插件
emq_auth_mysql	MySQL 认证插件
emq_auth_pgsql	PostgreSQL 认证插件
emq_auth_redis	Redis 认证插件
emq_auth_mongo	MongoDB 认证插件
emq_sn	MQTT-SN 协议插件
emq_coap	CoAP 协议插件
emq_stomp	Stomp 协议插件
emq_recon	Recon 优化调测插件
emq_reloader	热升级插件(开发调试)
emq_sockjs	SockJS 插件(废弃)

扩展插件通过 'bin/emqttd_ctl' 管理命令行，或 Dashboard 控制台加载启用。例如启用 PostgreSQL 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

1.8 100万线连接测试说明

注解: *EMQ* 2.0 消息服务器默认设置，允许最大客户端连接是512，因为大部分操作系统 'ulimit -n' 限制为1024。

EMQ 消息服务器1.1.3版本，连接压力测试到130万线，8核心/32G内存的 CentOS 云服务器。

操作系统内核参数、TCP 协议栈参数、Erlang 虚拟机参数、EMQ 最大允许连接数设置简述如下:

1.8.1 Linux 操作系统参数

2M - 系统所有进程可打开的文件数量:

```
sysctl -w fs.file-max=2097152
sysctl -w fs.nr_open=2097152
```

1M - 系统允许当前进程打开的文件数量:

```
ulimit -n 1048576
```

1.8.2 TCP 协议栈参数

backlog - Socket 监听队列长度:

```
sysctl -w net.core.somaxconn=65536
```

1.8.3 Erlang 虚拟机参数

emqttd/etc/emq.conf:

```
## Erlang Process Limit
node.process_limit = 2097152

## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 1048576
```

1.8.4 EMQ 最大允许连接数

emqttd/etc/emq.conf 'listeners'段落:

```
## Size of acceptor pool
listener.tcp.external.acceptors = 64

## Maximum number of concurrent clients
listener.tcp.external.max_clients = 1000000
```

1.8.5 测试客户端设置

测试客户端在一个接口上，最多只能创建65000连接:

```
sysctl -w net.ipv4.ip_local_port_range="500 65535"

echo 1000000 > /proc/sys/fs/nr_open
```

1.8.6 按应用场景测试

MQTT 是一个设计得非常出色的传输层协议，在移动消息、物联网、车联网、智能硬件甚至能源勘探等领域有着广泛的应用。1个字节报头、2个字节心跳、消息 QoS 支持等设计，非常适合在低带宽、不可靠网络、嵌入式设备上应用。

不同的应用有不同的系统要求，用户使用emqttd消息服务器前，可以按自己的应用场景进行测试，而不是简单的连接压力测试:

1. Android 消息推送: 推送消息广播测试。
2. 移动即时消息应用: 消息收发确认测试。
3. 智能硬件应用: 消息的往返时延测试。
4. 物联网数据采集: 并发连接与吞吐测试。

1.9 开源 MQTT 客户端项目

GitHub: https://github.com/emqtt

emqttc	Erlang MQTT客户端库
emqtt_benchmark	MQTT连接测试工具
CocoaMQTT	Swift语言MQTT客户端库
QMQTT	QT框架MQTT客户端库

Eclipse Paho: https://www.eclipse.org/paho/

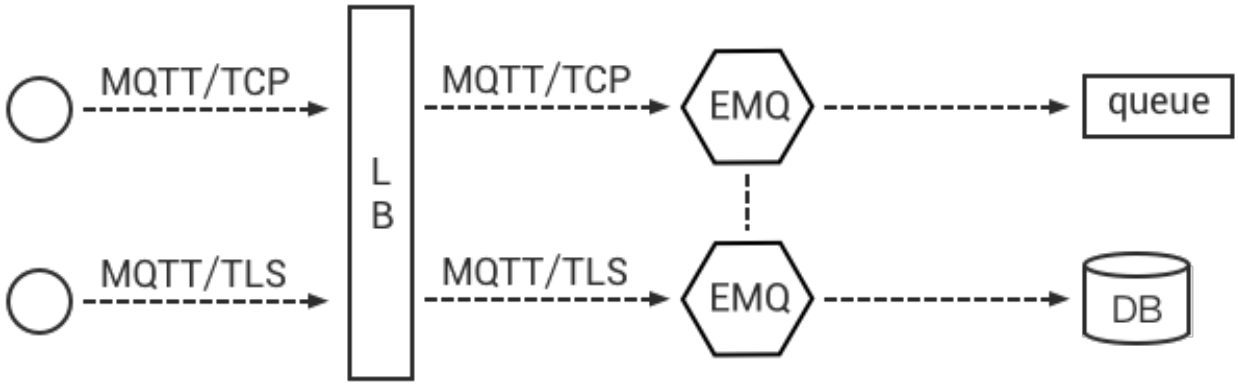
MQTT.org: https://github.com/mqtt/mqtt.github.io/wiki/libraries

CHAPTER 2

部署架构 (Deployment)

EMQ 消息服务器集群可作为物联网接入服务(IoT Hub)，部署在青云、AWS、阿里等公有云或企业私有云平台。

典型部署结构:



2.1 LB (负载均衡)

LB (负载均衡器) 负责分发设备的 MQTT 连接与消息到 EMQ 集群，LB 提高 EMQ 集群可用性、实现负载平衡以及动态扩容。

部署架构推荐在 LB 终结 SSL 连接。设备与 LB 之间 TLS 安全连接，LB 与 EMQ 之间普通 TCP 连接。这种部署模式下 EMQ 单集群可轻松支持100万设备。

公有云厂商 LB 产品:

云计算厂商	是否支持 TLS 终结	LB 产品介绍
青云	是	https://docs.qingcloud.com/guide/loadbalancer.html
AWS	是	https://aws.amazon.com/cn/elasticloadbalancing/
阿里云	否	https://www.aliyun.com/product/slb
UCloud	未知	https://ucloud.cn/site/product/ulb.html
QCloud	未知	https://www.qcloud.com/product/clb

私有部署 LB 服务器:

开源 LB	是否支持 TLS 终结	方案介绍
HAProxy	是	https://www.haproxy.com/solutions/load-balancing.html
NGINX	PLUS 产品支持	https://www.nginx.com/solutions/load-balancing/

国内公有云部署推荐青云(EMQ 合作伙伴)，国外部署推荐 AWS 。私有部署推荐使用 HAProxy 作为 LB。

2.2 EMQ 集群

EMQ 节点集群部署在 LB 之后，建议部署在 VPC 或私有网络内。公有云厂商青云、AWS、UCloud、QCloud 均支持 VPC 网络。

EMQ 默认开启的 MQTT 服务 TCP 端口:

1883	MQTT 协议端口
8883	MQTT/SSL 端口
8083	MQTT/WebSocket 端口
8084	MQTT/WebSocket/SSL 端口

防火墙根据使用的 MQTT 接入方式，开启上述端口的访问权限。

EMQ 节点集群使用的 TCP 端口:

4369	集群节点发现端口
6369	集群节点控制通道

集群节点间如有防护墙，需开启上述 TCP 端口互访权限。

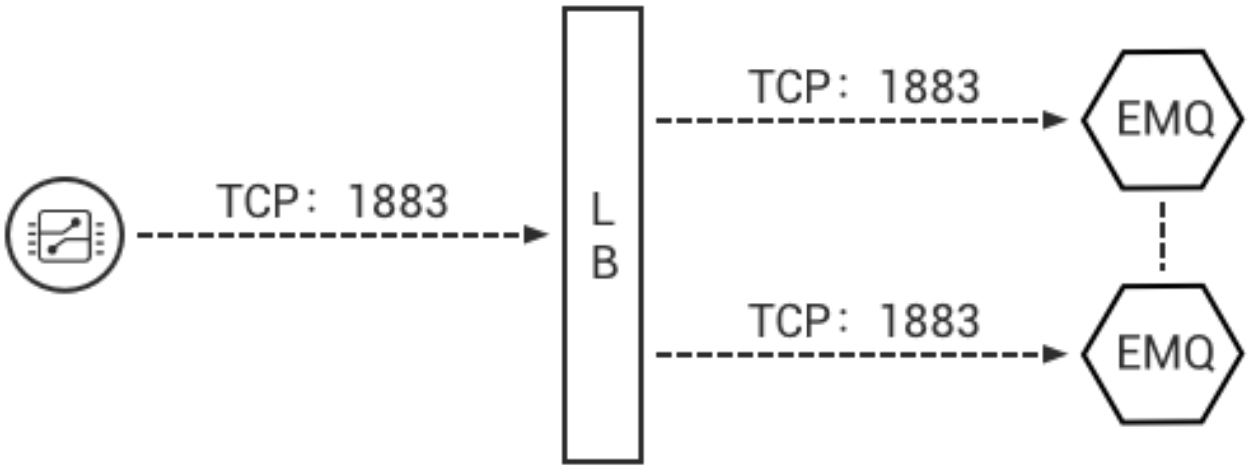
2.3 青云(QingCloud) 部署

1. 创建 VPC 网络。
2. VPC 网络内创建 EMQ 集群'私有网络'，例如: 192.168.0.0/24
3. 私有网络内创建两台 EMQ 主机，例如:

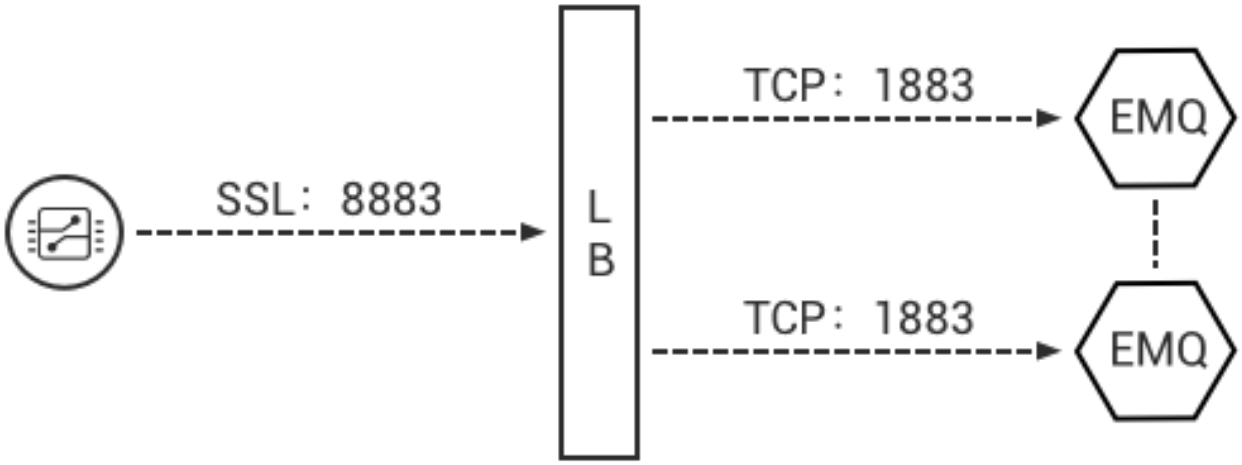
emq1	192.168.0.2
emq2	192.168.0.3

4. 安装并集群 EMQ 主机，具体配置请参考安装集群章节。

5. 创建 LB(负载均衡器) 并指定公网 IP 地址。
6. 在 LB 上创建 MQTT TCP 监听器:



或创建 SSL 监听器，并终结 SSL 在 LB :



7. MQTT 客户端连接 LB 公网地址测试。

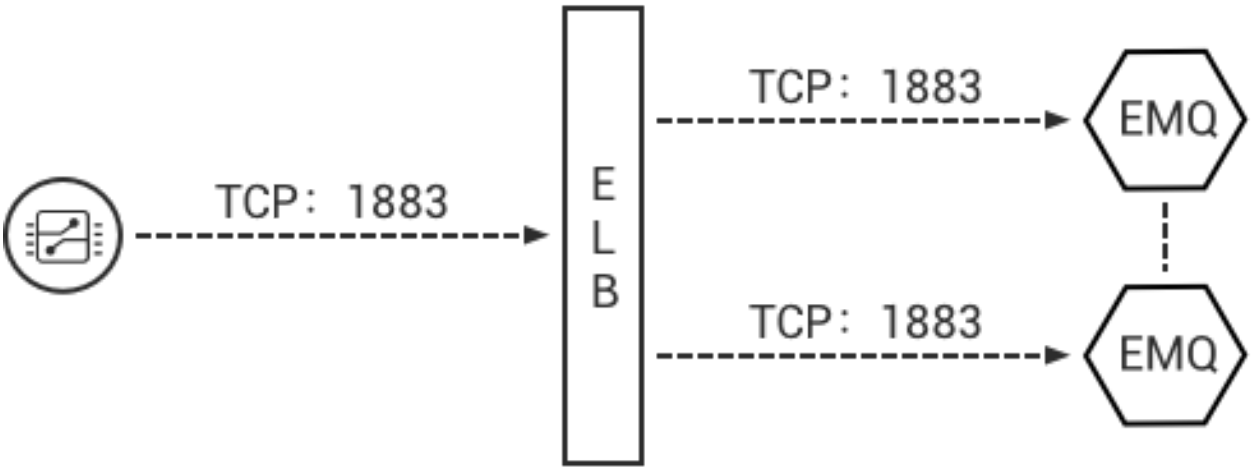
2.4 亚马逊(AWS)部署

1. 创建 VPC 网络。
2. VPC 网络内创建 EMQ 集群'私有网络'，例如: 192.168.0.0/24
3. 私有网络内创建两台 EMQ 主机，指定上面创建的 VPC 网络,例如:

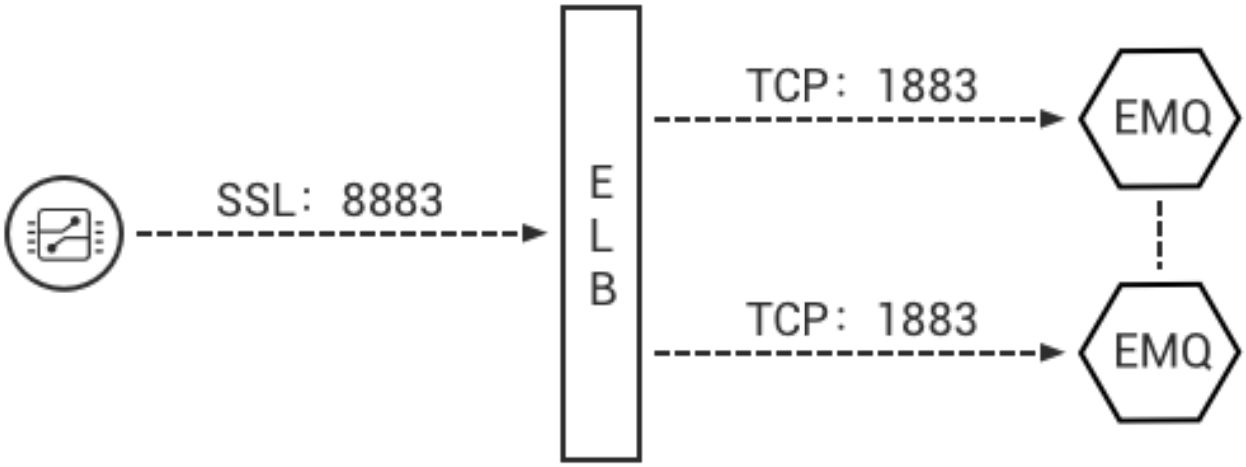
emq1	192.168.0.2
emq2	192.168.0.3

4. 在安全组中，开放 MQTT 服务的 TCP 端口，比如1883, 8883。
5. 安装并集群 EMQ 主机，具体配置请参考安装集群章节。
6. 创建 ELB (Classic负载均衡器)，指定 VPC 网络，并指定公网 IP 地址。

7. 在 ELB 上创建 MQTT TCP 监听器:



或创建 SSL 监听器，并终结 SSL 在 LB :



8. MQTT 客户端连接 LB 公网地址测试。

2.5 阿里云部署

2.6 私有网络部署

2.6.1 EMQ 集群直连

EMQ 集群直接挂 DNS 轮询，设备通过域名或者 IP 地址列表访问:

1. 部署 EMQ 集群，具体参考‘程序包安装‘与‘集群配置‘文档。
2. EMQ 节点防火墙开启外部 MQTT 访问端口，例如1883, 8883。
3. 设备通过 IP 地址列表或域名访问 EMQ 集群。

注解: 产品部署不推荐这种部署方式。

2.6.2 HAProxy -> EMQ 集群

HAProxy 作为 LB 部署 EMQ 集群，并终结 SSL 连接:

1. 创建 EMQ 集群节点，例如:

节点	IP 地址
emq1	192.168.0.2
emq2	192.168.0.3

2. 配置 /etc/haproxy/haproxy.cfg，示例:

```
listen mqtt-ssl
    bind *:8883 ssl crt /etc/ssl/emqttd/emq.pem no-sslv3
    mode tcp
    maxconn 50000
    timeout client 600s
    default_backend emq_cluster

backend emq_cluster
    mode tcp
    balance source
    timeout server 50s
    timeout check 5000
    server emq1 192.168.0.2:1883 check inter 10000 fall 2 rise 5 weight 1
    server emq2 192.168.0.3:1883 check inter 10000 fall 2 rise 5 weight 1
    source 0.0.0.0 usesrc clientip
```

2.6.3 NGINX Plus -> EMQ 集群

NGINX Plus 产品作为 EMQ 集群 LB，并终结 SSL 连接:

1. 注册 NGINX Plus 试用版，Ubuntu 下安装: https://cs.nginx.com/repo_setup
2. 创建 EMQ 节点集群，例如:

节点	IP 地址
emq1	192.168.0.2
emq2	192.168.0.3

3. 配置 /etc/nginx/nginx.conf，示例:

```
stream {
    # Example configuration for TCP load balancing

    upstream stream_backend {
        zone tcp_servers 64k;
        hash $remote_addr;
        server 192.168.0.2:1883 max_fails=2 fail_timeout=30s;
        server 192.168.0.3:1883 max_fails=2 fail_timeout=30s;
    }

    server {
        listen 8883 ssl;
        status_zone tcp_server;
        proxy_pass stream_backend;
```

(continues on next page)

(续上页)

```
        proxy_buffer_size 4k;
        ssl_handshake_timeout 15s;
        ssl_certificate     /etc/emqttd/certs/cert.pem;
        ssl_certificate_key /etc/emqttd/certs/key.pem;
    }
}
```

CHAPTER 3

程序安装 (Installation)

EMQ 2.0 消息服务器可跨平台运行在 Linux、FreeBSD、Mac OS X 或 Windows 服务器上。

注解: 产品部署建议 Linux、FreeBSD 服务器，不推荐 Windows 服务器。

3.1 *EMQ* 2.0 程序包下载

EMQ 2.0 消息服务器每个版本会发布 Ubuntu、CentOS、FreeBSD、Mac OS X、Windows 平台程序包与 Docker 镜像。

下载地址: http://emqtt.com/downloads

3.2 RPM 包安装

EMQ Linux RPM 程序包:

CentOS6.8	http://emqtt.com/downloads/latest/centos6-rpm
CentOS7	http://emqtt.com/downloads/latest/centos7-rpm

安装包命名由平台、版本、操纵系统位数组成，例如: emqttd-centos7-v2.0_x86_64.rpm

CentOS、RedHat 操作系统下，推荐 RPM 包安装。RPM 包安装后可通过操作系统，直接管理启停 EMQ 服务。

3.2.1 RPM 安装

```
rpm -ivh emqttd-centos7-v2.1.2-1.el7.centos.x86_64.rpm
```

注解: Erlang/OTP R19 依赖 lksctp-tools 库

```
yum install lksctp-tools
```

3.2.2 配置文件

EMQ 配置文件: /etc/emqttd/emq.conf，插件配置文件: /etc/emqttd/plugins/*.conf。

3.2.3 日志文件

日志文件目录: /var/log/emqttd

3.2.4 数据文件

数据文件目录：/var/lib/emqttd/

3.2.5 启动停止

```
systemctl start|stop|restart emqttd.service
```

3.3 DEB 包安装

EMQ Linux DEB 程序包:

Ubuntu12.04	http://emqtt.com/downloads/latest/ubuntu12_04-deb
Ubuntu14.04	http://emqtt.com/downloads/latest/ubuntu14_04-deb
Ubuntu16.04	http://emqtt.com/downloads/latest/ubuntu16_04-deb
Debian7	http://emqtt.com/downloads/latest/debian7-deb
Debian8	http://emqtt.com/downloads/latest/debian7-deb

安装包命名由平台、版本、操纵系统位数组成，例如: emqttd-debian7-v2.0_amd64.deb

Debian、Ubuntu 操作系统下，推荐 DEB 包安装。DEB 包安装后可通过操作系统，直接管理启停 EMQ 服务。

```
sudo dpkg -i emqttd-ubuntu16.04_v2.0_amd64.deb
```

注解: Erlang/OTP R19依赖lksctp-tools库

```
apt-get install lksctp-tools
```

3.3.1 配置文件

EMQ 配置文件: /etc/emqttd/emq.conf，插件配置文件: /etc/emqttd/plugins/*.conf。

3.3.2 日志文件

日志文件目录: /var/log/emqttd

3.3.3 数据文件

数据文件目录：/var/lib/emqttd/

3.3.4 启动停止

```
service emqttd start|stop|restart
```

3.4 Linux 通用包安装

EMQ Linux 通用程序包:

Ubuntu12.04	http://emqtt.com/downloads/latest/ubuntu12_04
Ubuntu14.04	http://emqtt.com/downloads/latest/ubuntu14_04
Ubuntu16.04	http://emqtt.com/downloads/latest/ubuntu16_04
CentOS6.8	http://emqtt.com/downloads/latest/centos6
CentOS7	http://emqtt.com/downloads/latest/centos7
Debian7	http://emqtt.com/downloads/latest/debian7
Debian8	http://emqtt.com/downloads/latest/debian7
FreeBSD	http://emqtt.com/downloads/latest/freebsd

安装包命名由平台、版本组成，例如: emqttd-macosx-v2.0.zip

CentOS 平台为例，下载安装过程:

```
unzip emqttd-centos7-v2.0.zip
```

控制台调试模式启动，检查 *EMQ* 是否可正常启动:

```
cd emqttd && ./bin/emqttd console
```

EMQ 消息服务器如启动正常，控制台输出:

```
starting emqttd on node 'emqttd@127.0.0.1'
emqttd ctl is starting...[ok]
emqttd hook is starting...[ok]
emqttd router is starting...[ok]
emqttd pubsub is starting...[ok]
emqttd stats is starting...[ok]
emqttd metrics is starting...[ok]
emqttd pooler is starting...[ok]
emqttd trace is starting...[ok]
emqttd client manager is starting...[ok]
emqttd session manager is starting...[ok]
emqttd session supervisor is starting...[ok]
emqttd wsclient supervisor is starting...[ok]
emqttd broker is starting...[ok]
emqttd alarm is starting...[ok]
emqttd mod supervisor is starting...[ok]
emqttd bridge supervisor is starting...[ok]
emqttd access control is starting...[ok]
emqttd system monitor is starting...[ok]
dashboard:http listen on 0.0.0.0:18083 with 2 acceptors.
mqtt:tcp listen on 0.0.0.0:1883 with 8 acceptors.
mqtt:ssl listen on 0.0.0.0:8883 with 4 acceptors.
mqtt:ws listen on 0.0.0.0:8083 with 4 acceptors.
Erlang MQTT Broker 2.0 is running now
```

CTRL+c 关闭控制台。守护进程模式启动:

```
./bin/emqttd start
```

启动错误日志将输出在 log/ 目录。

EMQ 消息服务器进程状态查询:

```
./bin/emqttd_ctl status
```

正常运行状态，查询命令返回:

```
$ ./bin/emqttd_ctl status
Node 'emqttd@127.0.0.1' is started
emqttd 2.0 is running
```

EMQ 消息服务器提供了状态监控 URL

```
http://localhost:8080/status
```

停止服务器:

```
./bin/emqttd stop
```

3.5 FreeBSD 服务器安装

EMQ FreeBSD 程序包下载: http://emqtt.com/downloads/latest/freebsd

FreeBSD 平台安装过程与Linux相同。

3.6 Mac OS X 系统安装

Mac 下开发调试 MQTT 应用，可直接下载安装: http://emqtt.com/downloads/latest/macosx
配置文件 'etc/emq.conf' log 段落打开 debug 日志，控制台可以查看收发 MQTT 报文详细:
EMQ 在 Mac 平台下安装启动过程与 Linux 相同。

3.7 Windows 服务器安装

Windows 平台程序包下载: http://emqtt.com/downloads/latest/windows10
程序包下载解压后，打开 Windows 命令行窗口，cd 到程序目录。
控制台模式启动:

```
bin\emqttd console
```

如启动成功，会弹出控制台窗口。
关闭控制台窗口，停止emqttd进程，准备注册 Windows 服务。

> **警告:** EMQ-2.0 暂不支持服务注册

EMQ 注册为 Windows 服务:

```
bin\emqttd install
```

EMQ 服务启动:

```
bin\emqttd start
```

EMQ 服务停止:

```
bin\emqttd stop
```

EMQ 服务卸载:

```
bin\emqttd uninstall
```

3.8 Docker 镜像安装

EMQ 2.0 Docker 镜像下载: http://emqtt.com/downloads/latest/docker
解压 emqttd-docker 镜像包:

```
unzip emqttd-docker-v2.0.zip
```

加载镜像:

```
docker load < emqttd-docker-v2.0
```

启动容器:

```
docker run -tid --name emq20 -p 1883:1883 -p 8083:8083 -p 8883:8883 -p 8084:8084 -p␣
↪18083:18083 emqttd-docker-v2.0
```

停止容器:

```
docker stop emq20
```

开启容器:

```
docker start emq20
```

进入 Docker 控制台:

```
docker exec -it emq20 /bin/sh
```

3.9 源码编译安装

EMQ 消息服务器基于 Erlang/OTP 平台开发，项目托管的 GitHub 管理维护，源码编译依赖 Erlang 环境和 git 客户端。

注解: EMQ R2.3+ 依赖 Erlang R20+ 版本

Erlang 安装: http://www.erlang.org/

Git 客户端: http://www.git-scm.com/

Ubuntu 平台可通过 apt-get 命令安装，CentOS/RedHat 平台可通过 yum 命令安装，Mac 下可通过 brew 包管理命令安装，Windows 下... :(

编译环境准备好之后，clone 代码开始编译:

```
git clone https://github.com/emqtt/emq-relx.git

cd emq-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

编译成功后，可执行程序包在目录:

```
_rel/emqttd
```

控制台启动编译的 EMQ 程序包:

```
cd _rel/emqttd && ./bin/emqttd console
```

3.10 Windows 源码编译安装

Erlang 安装: http://www.erlang.org/

MSYS2 安装: http://www.msys2.org/

MSYS2 安装完成后，根据 MSYS2 中的 pacman 包管理工具安装 Git、 Make 工具软件:

```
pacman -S git make
```

编译环境准备之后，clone 代码开始编译:

```
git clone -b windows https://github.com/emqtt/emqttd-relx.git

cd emqttd-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

编译成功后，可执行程序包在目录:

```
_rel/emqttd
```

控制台启动编译的 EMQ 程序包:

```
cd _rel/emqttd && ./bin/emqttd console
```

3.11 TCP 服务端口占用

EMQ 2.0 消息服务器默认占用的 TCP 端口包括:

1883	MQTT 协议端口
8883	MQTT/SSL 端口
8083	MQTT/WebSocket 端口
8080	HTTP API 端口
18083	Dashboard 管理控制台端口

EMQ 2.0 占用的上述端口，可通过 etc/emq.conf 配置文件的 'listener' 段落设置:

```
## TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
listener.tcp.external = 0.0.0.0:1883

## SSL Listener: 8883, 127.0.0.1:8883, ::1:8883
listener.ssl.external = 8883

## External MQTT/WebSocket Listener
listener.ws.external = 8083

## HTTP Management API Listener
listener.api.mgmt = 127.0.0.1:8080
```

通过注释或删除相关段落，可禁用相关 TCP 服务启动。

18083端口是 Web 管理控制占用，该端口由 emq_dashboard 插件启用。

控制台 URL: http:://localhost:18083/ ，默认登录用户名: admin, 密码: public。

3.12 快速设置

EMQ 消息服务器主要配置文件:

etc/emq.conf	EMQ 消息服务器参数设置
etc/plugins/*.conf	EMQ 插件配置文件

etc/emq.conf 中两个重要的虚拟机启动参数:

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

注解: Erlang 的 Port 非 TCP 端口，可以理解为文件句柄。

node.process_limit = 参数值 > 最大允许连接数 * 2

node.max_ports = 参数值 > 最大允许连接数

警告: 实际连接数量超过 Erlang 虚拟机参数设置，会引起 EMQ 消息服务器宕机!

etc/emq.conf 配置文件的 *listener* 段落设置最大允许连接数:

```
listener.tcp.external = 0.0.0.0:1883

listener.tcp.external.acceptors = 8

listener.tcp.external.max_clients = 1024
```

EMQ 2.0 消息服务器详细设置，请参见文档: config

注解: ## erlexec: HOME must be set uncomment '# export HOME=/root' if "HOME must be set" error.

CHAPTER 4

青云映像 (Image)

EMQ 消息服务器1.1.2版本正式登陆 青云 映像市场，用户可直接创建映像启用 EMQ 消息服务器:

4.1 映像属性

属性	值
名称	EMQ Ubuntu映像-百万级分布式物联网MQTT消息服务器
费用	免费
类别	物联网
摘要	百万级连接和分布式集群，发布订阅模式的开源 MQTT 消息服务器。完整支持 MQTT、MQTT-SN、CoAP 等物联 网协议。
开发者	feng@emqtt.io
提交时间	2016-07-11
更新时间	2016-07-11

4.2 映像描述

1. 该映像基于EMQ 1.1.2版本创建，启动后 MQTT 服务器监听在 TCP 1883端口；

2. EMQ 服务器 WEB 控制台，端口18083。默认管理员: admin, 密码: public，映像启动后请登录控制台修改默认密码;
3. 映像默认没有开启 MQTT 认证，允许 MQTT 客户端匿名登录;
4. EMQ 程序部署在 /opt/emqttd/ 目录，用户可通过配置 etc/emqttd.config 或加载插件方式，开启 HTTP、PostgreSQL、MySQL、Redis、MongoDB 等多种认证方式;
5. EMQ 消息服务器详细配置和使用，请参考文档: http://emqtt.com/docs/

本映像由 emqtt.com 与青云共同提供技术支持，用户QQ群组: 196066320

4.3 映像当前版本

映像包含组件:

1. Erlang/OTP 18.3版本环境
2. EMQ 1.1.2版本

映像配置说明:

1. 防火墙开启1883(MQTT)、8083(WebSocket)、18083(Dashboard控制台)端口。
2. Dashboard 控制台, 端口18083。默认管理员: admin, 密码: public，映像启动后请立即修改密码!
3. 映像默认允许1万线 MQTT 连接，最大可配置到10万线。
4. 映像内存占用: 5万连接/1G内存。请用户根据 MQTT 消息吞吐量调整内存。

4.4 EMQ 手工启停

```
systemctl start emqttd
systemctl stop emqttd
```

CHAPTER 5

配置说明 (Configuration)

5.1 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

5.2 EMQ 配置变更历史

为方便用户与插件开发者使用，*EMQ* 配置文件经过三次调整。

1. EMQ 1.x 版本采用 Erlang 原生配置文件格式 etc/emqttd.config:

```
{emqttd, [
  %% Authentication and Authorization
  {access, [
    %% Authetication. Anonymous Default
    {auth, [
        %% Authentication with username, password
        %{username, []},

        %% Authentication with clientid
        %{clientid, [{password, no}, {file, "etc/clients.config"}]},
```

Erlang 的原生配置格式多层级嵌套，对非 Erlang 开发者的用户很不友好。

2. EMQ 2.0-beta.x 版本简化了原生 Erlang 配置文件，采用类似 rebar.config 或 relx.config 格式:

```
%% Max ClientId Length Allowed.
{mqtt_max_clientid_len, 512}.

%% Max Packet Size Allowed, 64K by default.
{mqtt_max_packet_size, 65536}.

%% Client Idle Timeout.
{mqtt_client_idle_timeout, 30}. % Second
```

简化后的 Erlang 原生配置格式方便用户配置，但插件开发者不得不依赖 gen_conf 库，而不是通过 application:get_env 读取配置参数。

3. EMQ 2.0-rc.2 正式版集成了 cuttlefish 库，采用了类似 sysctl 的 $k = v$ 通用格式，并在系统启动时翻译成 Erlang 原生配置格式:

```
## Node name
node.name = emqttd@127.0.0.1
...
## Max ClientId Length Allowed.
mqtt.max_clientid_len = 1024
...
```

EMQ 2.0 启动时配置文件处理流程:

```
----------------------                                                2.0/schema/*.schema
↪    --------------------
| etc/emq.conf       |                   ------------------              \|/
↪    | data/app.config |
|       +            | --> mergeconf --> | data/app.conf | -->  cuttlefish generate  -
↪-> |                  |
| etc/plugins/*.conf |                   ------------------
↪    | data/vm.args    |
----------------------
↪    --------------------
```

5.3 EMQ 2.2 环境变量

EMQ_NODE_NAME	Erlang 节点名称，例如: emq@127.0.0.1
EMQ_NODE_COOKIE	Erlang 分布式节点通信 Cookie
EMQ_MAX_PORTS	Erlang 虚拟机最大允许打开文件 Socket 数
EMQ_TCP_PORT	MQTT/TCP 监听端口，默认: 1883
EMQ_SSL_PORT	MQTT/SSL 监听端口，默认: 8883
EMQ_WS_PORT	MQTT/WebSocket 监听端口，默认: 8083
EMQ_WSS_PORT	MQTT/WebSocket/SSL 监听端口，默认: 8084

5.4 EMQ 集群设置

5.4.1 集群名称

```
## Cluster name
cluster.name = emqcl
```

5.4.2 自动发现策略

```
## Cluster discovery strategy: manual | static | mcast | dns | etcd | k8s
cluster.discovery = manual
```

5.4.3 启用集群自愈

```
## Cluster Autoheal: on | off
cluster.autoheal = on
```

5.4.4 节点自动清除

自动清除宕机节点:

```
## Clean down node of the cluster
cluster.autoclean = 5m
```

5.5 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

5.5.1 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
```

5.5.2 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
```

5.5.3 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
```

5.5.4 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
```

5.5.5 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
```

(continues on next page)

(续上页)

```
cluster.etcd.node_ttl = 1m
```

5.5.6 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
```

5.6 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。

5.7 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
```

(continues on next page)

(续上页)

```
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
```

5.8 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
```

Erlang 虚拟机主要参数说明:

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

5.9 日志参数配置

5.9.1 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
```

5.9.2 error 日志

```
## Error log file
log.error.file = log/error.log
```

5.9.3 crash 日志

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

log.crash.file = log/crash.log
```

5.9.4 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
```

5.10 MQTT 协议参数配置

5.10.1 ClientId 最大允许长度

```
## Max ClientId Length Allowed.
mqtt.max_clientid_len = 1024
```

5.10.2 MQTT 最大报文尺寸

```
## Max Packet Size Allowed, 64K by default.
mqtt.max_packet_size = 64KB
```

5.10.3 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
```

5.10.4 启用客户端连接统计

```
## Enable client Stats: on | off
mqtt.client.enable_stats = off
```

5.10.5 强制 GC 设置

```
## Force GC: integer. Value 0 disabled the Force GC.
mqtt.conn.force_gc_count = 100
```

5.11 匿名认证与 ACL 文件

5.11.1 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
```

5.11.2 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:

```
          ---------              ---------              ---------
Client -> | Rule1 | --nomatch--> | Rule2 | --nomatch--> | Rule3 | --> Default
          ---------              ---------              ---------
              |                      |                      |
            match                  match                  match
             \|/                    \|/                    \|/
       allow | deny           allow | deny           allow | deny
```

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。

5.12 MQTT 会话参数设置

```
## Upgrade QoS?
mqtt.session.upgrade_qos = off

## Max number of QoS 1 and 2 messages that can be "inflight" at one time.
## 0 means no limit
mqtt.session.max_inflight = 32

## Retry Interval for redelivering QoS1/2 messages.
mqtt.session.retry_interval = 20s

## Max Packets that Awaiting PUBREL, 0 means no limit
mqtt.session.max_awaiting_rel = 100
```

(continues on next page)

(续上页)

```
## Awaiting PUBREL Timeout
mqtt.session.await_rel_timeout = 20s

## Enable Statistics: on | off
mqtt.session.enable_stats = off

## Expired after 1 day:
## w - week
## d - day
## h - hour
## m - minute
## s - second
mqtt.session.expiry_interval = 2h
```

5.13 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:

```
## Type: simple | priority
mqtt.mqueue.type = simple

## Topic Priority: 0~255, Default is 0
## mqtt.mqueue.priority = topic/1=10,topic/2=8

## Max queue length. Enqueued messages when persistent client disconnected,
## or inflight window is full. 0 means no limit.
mqtt.mqueue.max_length = 0

## Low-water mark of queued messages
mqtt.mqueue.low_watermark = 20%

## High-water mark of queued messages
mqtt.mqueue.high_watermark = 60%

## Queue Qos0 messages?
mqtt.mqueue.store_qos0 = true
```

队列参数说明:

mqueue.type	队列类型。simple: 简单队列，priority: 优先级队列
mqueue.priority	主题(Topic)队列优先级设置
mqueue.max_length	队列长度, infinity 表示不限制
mqueue.low_watermark	解除告警水位线
mqueue.high_watermark	队列满告警水位线
mqueue.qos0	是否缓存 QoS0 消息

5.14 Broker 参数设置

broker_sys_interval 设置系统发布 $SYS 消息周期:

```
## System Interval of publishing broker $SYS Messages
mqtt.broker.sys_interval = 60s
```

5.15 发布订阅(PubSub)参数设置

```
## PubSub Pool Size. Default should be scheduler numbers.
mqtt.pubsub.pool_size = 8

mqtt.pubsub.by_clientid = true

## Subscribe Asynchronously
mqtt.pubsub.async = true
```

5.16 桥接(Bridge)参数设置

```
## Bridge Queue Size
mqtt.bridge.max_queue_len = 10000

## Ping Interval of bridge node. Unit: Second
mqtt.bridge.ping_down_interval = 1s
```

5.17 插件(Plugin) 配置目录设置

```
## Dir of plugins' config
mqtt.plugins.etc_dir = etc/plugins/

## File to store loaded plugin names.
mqtt.plugins.loaded_file = data/loaded_plugins
```

5.18 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”

5.19 MQTT/TCP 监听器 - 1883

EMQ 2.2 版本支持配置多个 MQTT 协议监听器，例如配置 external、internal 两个监听器:

```
##--------------------------------------------------------------------
## External TCP Listener

## External TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
listener.tcp.external = 0.0.0.0:1883

## Size of acceptor pool
listener.tcp.external.acceptors = 16

## Maximum number of concurrent clients
listener.tcp.external.max_clients = 102400

#listener.tcp.external.mountpoint = external/

## Rate Limit. Format is 'burst,rate', Unit is KB/Sec
#listener.tcp.external.rate_limit = 100,10

#listener.tcp.external.access.1 = allow 192.168.0.0/24

listener.tcp.external.access.2 = allow all

## Proxy Protocol V1/2
## listener.tcp.external.proxy_protocol = on
## listener.tcp.external.proxy_protocol_timeout = 3s

## TCP Socket Options
listener.tcp.external.backlog = 1024

#listener.tcp.external.recbuf = 4KB

#listener.tcp.external.sndbuf = 4KB

listener.tcp.external.buffer = 4KB

listener.tcp.external.nodelay = true

##--------------------------------------------------------------------
## Internal TCP Listener

## Internal TCP Listener: 11883, 127.0.0.1:11883, ::1:11883
listener.tcp.internal = 127.0.0.1:11883

## Size of acceptor pool
listener.tcp.internal.acceptors = 16
```

(continues on next page)

(续上页)

```
## Maximum number of concurrent clients
listener.tcp.internal.max_clients = 102400

#listener.tcp.external.mountpoint = internal/

## Rate Limit. Format is 'burst,rate', Unit is KB/Sec
## listener.tcp.internal.rate_limit = 1000,100

## TCP Socket Options
listener.tcp.internal.backlog = 512

listener.tcp.internal.tune_buffer = on

listener.tcp.internal.buffer = 1MB

listener.tcp.internal.recbuf = 4KB

listener.tcp.internal.sndbuf = 1MB

listener.tcp.internal.nodelay = true
```

5.20 MQTT/SSL 监听器 - 8883

```
##--------------------------------------------------------------------
## External SSL Listener
listener.ssl.external = 8883

## Size of acceptor pool
listener.ssl.external.acceptors = 16

## Maximum number of concurrent clients
listener.ssl.external.max_clients = 1024

## listener.ssl.external.mountpoint = inbound/

## Rate Limit. Format is 'burst,rate', Unit is KB/Sec
## listener.ssl.external.rate_limit = 100,10

## Proxy Protocol V1/2
## listener.ssl.external.proxy_protocol = on
## listener.ssl.external.proxy_protocol_timeout = 3s

listener.ssl.external.access.1 = allow all

## SSL Options
listener.ssl.external.handshake_timeout = 15
listener.ssl.external.keyfile = etc/certs/key.pem
listener.ssl.external.certfile = etc/certs/cert.pem
## 开启双向认证
## listener.ssl.external.cacertfile = etc/certs/cacert.pem
## listener.ssl.external.verify = verify_peer
## listener.ssl.external.fail_if_no_peer_cert = true
```

5.21 MQTT/WebSocket 监听器 - 8083

```
##--------------------------------------------------------------------
## External MQTT/WebSocket Listener

listener.ws.external = 8083

listener.ws.external.acceptors = 4

listener.ws.external.max_clients = 64

listener.ws.external.access.1 = allow all
```

5.22 MQTT/WebSocket/SSL 监听器 - 8084

```
##--------------------------------------------------------------------
## External MQTT/WebSocket/SSL Listener

listener.wss.external = 8084

listener.wss.external.acceptors = 4

listener.wss.external.max_clients = 64

listener.wss.external.access.1 = allow all

## SSL Options
listener.wss.external.handshake_timeout = 15s

listener.wss.external.keyfile = {{ platform_etc_dir }}/certs/key.pem

listener.wss.external.certfile = {{ platform_etc_dir }}/certs/cert.pem

## listener.wss.external.cacertfile = {{ platform_etc_dir }}/certs/cacert.pem

## listener.wss.external.verify = verify_peer

## listener.wss.external.fail_if_no_peer_cert = true
```

5.23 HTTP API 监听器 - 8080

```
##--------------------------------------------------------------------
## HTTP Management API Listener

listener.api.mgmt = 127.0.0.1:8080

listener.api.mgmt.acceptors = 4

listener.api.mgmt.max_clients = 64

listener.api.mgmt.access.1 = allow all
```

5.24 Erlang 虚拟机监控设置

```
## Long GC, don't monitor in production mode for:
sysmon.long_gc = false

## Long Schedule(ms)
sysmon.long_schedule = 240

## 8M words. 32MB on 32-bit VM, 64MB on 64-bit VM.
sysmon.large_heap = 8MB

## Busy Port
sysmon.busy_port = false

## Busy Dist Port
sysmon.busy_dist_port = true
```

5.25 扩展插件配置文件

EMQ 2.2 插件配置文件，全部在 etc/plugins/ 目录:

配置文件	说明
etc/plugins/emq_mod_presence	客户端上下线状态消息发布
etc/plugins/emq_mod_retainer	Retain 消息存储插件
etc/plugins/emq_mod_subscription	客户端上线自动主题订阅
etc/plugins/emq_auth_username.conf	用户名、密码认证插件
etc/plugins/emq_auth_clientid.conf	ClientId 认证插件
etc/plugins/emq_auth_http.conf	HTTP 认证插件配置
etc/plugins/emq_auth_mongo.conf	MongoDB 认证插件配置
etc/plugins/emq_auth_mysql.conf	MySQL 认证插件配置
etc/plugins/emq_auth_pgsql.conf	Postgre 认证插件配置
etc/plugins/emq_auth_redis.conf	Redis 认证插件配置
etc/plugins/emq_web_hook.conf	Web Hook 插件配置
etc/plugins/emq_lua_hook.conf	Lua Hook 插件配置
etc/plugins/emq_coap.conf	CoAP 协议服务器配置
etc/plugins/emq_dashboard.conf	Dashboard 控制台插件配置
etc/plugins/emq_plugin_template.conf	示例插件模版
etc/plugins/emq_recon.conf	Recon 调试插件配置
etc/plugins/emq_reloader.conf	热加载插件配置
etc/plugins/emq_sn.conf	MQTT-SN 协议插件配置
etc/plugins/emq_stomp.conf	Stomp 协议插件配置

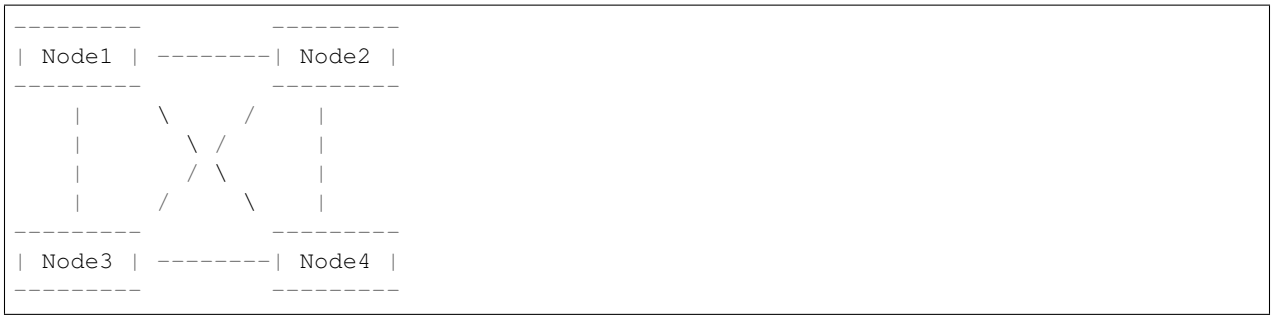
CHAPTER 6

分布集群 (Clustering)

6.1 Erlang/OTP 分布式编程

Erlang/OTP 最初是爱立信为开发电信设备系统设计的编程语言平台，电信设备(路由器、接入网关、...)典型设计是通过背板连接主控板卡与多块业务板卡的分布式系统。

Erlang/OTP 语言平台的分布式程序，由分布互联的 Erlang 运行系统组成，每个 Erlang 运行系统被称为节点(Node)，节点(Node) 间通过 TCP 互联，消息传递的方式通信:



6.1.1 节点(Node)

Erlang 节点由唯一的节点名称标识，节点间通过名称进行通信寻址。 例如在本机启动四个 Erlang 节点，节点名称分别为:

```
erl -name node1@127.0.0.1
erl -name node2@127.0.0.1
erl -name node3@127.0.0.1
erl -name node4@127.0.0.1
```

node1@127.0.0.1 控制台下建立与其他节点的连接:

```
(node1@127.0.0.1)1> net_kernel:connect_node('node2@127.0.0.1').
true
(node1@127.0.0.1)2> net_kernel:connect_node('node3@127.0.0.1').
true
(node1@127.0.0.1)3> net_kernel:connect_node('node4@127.0.0.1').
true
(node1@127.0.0.1)4> nodes().
['node2@127.0.0.1','node3@127.0.0.1','node4@127.0.0.1']
```

6.1.2 epmd

epmd(Erlang Port Mapper Daemon) - Erlang 端口映射服务程序，在 Erlang 节点运行主机上自启动，负责映射节点名称到通信 TCP 端口号:

```
(node1@127.0.0.1)6> net_adm:names().
{ok,[{"node1",62740},
     {"node2",62746},
     {"node3",62877},
     {"node4",62895}]}
```

6.1.3 安全

Erlang 节点间通过一个相同的 cookie 进行互连认证。

Erlang 节点 Cookie 设置:

```
1. $HOME/.erlang.cookie文件

2. erl -setcookie <Cookie>
```

本节内容来自: http://erlang.org/doc/reference_manual/distributed.html

6.1.4 连接

Erlang 集群节点可通过 TCPv4, TCPv6 或 TLS 方式连接，EMQ 2.3.5+ 版本支持在'etc/emq.conf' 中配置连接方式:

```
## 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
```

6.2 EMQ R2 分布集群设计

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:

```
topic1 -> node1, node2
topic2 -> node3
topic3 -> node2, node4
```

6.2.1 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):

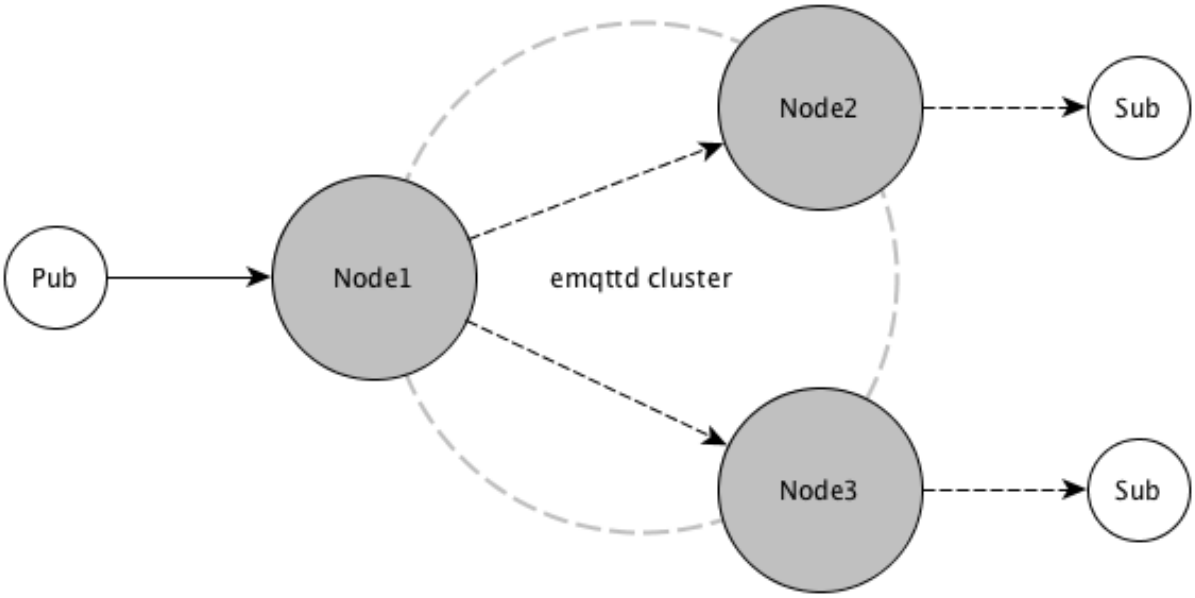
```
--------------------------
|              t           |
|             / \          |
|            +   #         |
|          /  \            |
|        x      y          |
--------------------------
| t/+/x -> node1, node3    |
| t/+/y -> node1           |
| t/#   -> node2           |
| t/a   -> node3           |
--------------------------
```

6.2.2 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]
```



假设部署两台服务器 s1.emqtt.io, s2.emqtt.io 上部署集群:

节点名	主机名(FQDN)	IP 地址
emq@s1.emqtt.io 或 emq@192.168.0.10	s1.emqtt.io	192.168.0.10
emq@s2.emqtt.io 或 emq@192.168.0.20	s2.emqtt.io	192.168.0.20

警告: 节点名格式: Name@Host, Host必须是IP地址或FQDN(主机名.域名)

6.2.3 emq@s1.emqtt.io 节点设置

emqttd/etc/emq.conf:

```
node.name = emq@s1.emqtt.io

或

node.name = emq@192.168.0.10
```

也可通过环境变量:

```
export EMQ_NODE_NAME=emq@s1.emqtt.io && ./bin/emqttd start
```

警告: 节点启动加入集群后，节点名称不能变更。

6.2.4 emq@s2.emqtt.io 节点设置

emqttd/etc/emq.conf:

```
node.name = emq@s2.emqtt.io

或

node.name = emq@192.168.0.20
```

6.2.5 节点加入集群

启动两台节点后，emq@s2.emqtt.io 上执行:

```
$ ./bin/emqttd_ctl cluster join emq@s1.emqtt.io

Join the cluster successfully.
Cluster status: [{running_nodes,['emq@s1.emqtt.io','emq@s2.emqtt.io']}]
```

或，emq@s1.emqtt.io 上执行:

```
$ ./bin/emqttd_ctl cluster join emq@s2.emqtt.io

Join the cluster successfully.
Cluster status: [{running_nodes,['emq@s1.emqtt.io','emq@s2.emqtt.io']}]
```

任意节点上查询集群状态:

```
$ ./bin/emqttd_ctl cluster status

Cluster status: [{running_nodes,['emq@s1.emqtt.io','emq@s2.emqtt.io']}]
```

6.2.6 节点退出集群

节点退出集群，两种方式:

1. leave: 本节点退出集群
2. remove: 从集群删除其他节点

emq@s2.emqtt.io 主动退出集群:

```
$ ./bin/emqttd_ctl cluster leave
```

或 emq@s1.emqtt.io 节点上，从集群删除 emq@s2.emqtt.io 节点:

```
$ ./bin/emqttd_ctl cluster remove emq@s2.emqtt.io
```

6.3 节点发现与自动集群

EMQ R2.3 版本支持基于 Ekka 库的集群自动发现(Autocluster)。Ekka 是为 Erlang/OTP 应用开发的集群管理库，支持 Erlang 节点自动发现(Discovery)、自动集群(Autocluster)、脑裂自动愈合(Network Partition Autoheal)、自动删除宕机节点(Autoclean)。

EMQ R2.3 支持多种策略自动发现节点创建集群:

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

6.3.1 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
```

6.3.2 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
```

6.3.3 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
```

6.3.4 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
```

6.3.5 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
```

6.3.6 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
```

6.4 集群脑裂与自动愈合

EMQ R2.3 版本正式支持集群脑裂自动恢复(Network Partition Autoheal):

```
cluster.autoheal = on
```

集群脑裂自动恢复流程:

1. 节点收到 Mnesia库 的 *inconsistent_database* 事件3秒后进行集群脑裂确认;

2. 节点确认集群脑裂发生后，向 Leader 节点(集群中最早启动节点)上报脑裂消息;
3. Leader 节点延迟一段时间后，在全部节点在线状态下创建脑裂视图(SplitView);
4. Leader 节点在多数派(majority)分区选择集群自愈的 Coordinator 节点;
5. Coordinator 节点重启少数派(minority)分区节点恢复集群。

6.5 集群节点自动清除

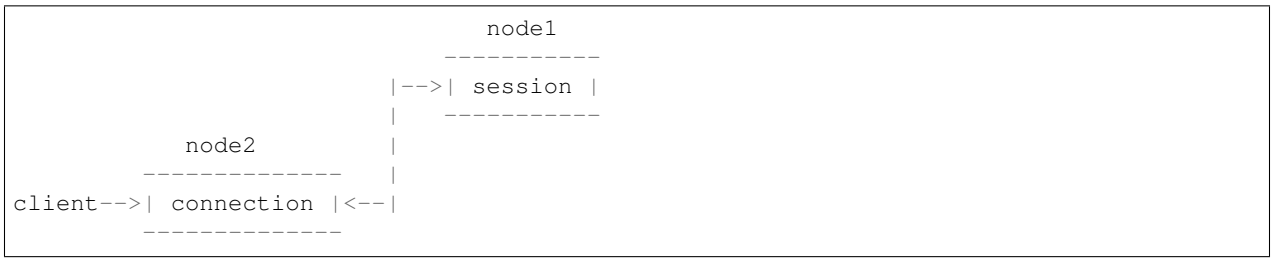
EMQ R2.3 版本支持从集群自动删除宕机节点(Autoclean):

```
cluster.autoclean = 5m
```

6.6 跨节点会话(Session)

EMQ 消息服务器集群模式下，MQTT 连接的持久会话(Session)跨节点。

例如负载均衡的两台集群节点: node1 与 node2，同一 MQTT 客户端先连接 node1，node1 节点会创建持久会话；客户端断线重连到 node2 时，MQTT 的连接在 node2 节点，持久会话仍在 node1 节点:



6.7 防火墙设置

如果集群节点间存在防火墙，防火墙需要开启 4369 端口和一个 TCP 端口段。4369 由 epmd 端口映射服务使用，TCP 端口段用于节点间建立连接与通信。

防火墙设置后，EMQ 需要配置相同的端口段，emqttd/etc/emq.conf 文件:

```
## Distributed node port range
node.dist_listen_min = 6369
node.dist_listen_max = 7369
```

6.8 一致性 Hash 与 DHT

NoSQL 数据库领域分布式设计，大多会采用一致性 Hash 或 DHT。EMQ 消息服务器集群架构可支持千万级的路由，更大级别的集群可采用一致性 Hash、DHT 或 Shard 方式切分路由表。

CHAPTER 7

节点桥接 (Bridge)

7.1 EMQ 节点间桥接

EMQ 消息服务器支持多节点桥接模式互联:

```
                ---------                          ---------                          ---------
Publisher --> | Node1 | --Bridge Forward--> | Node2 | --Bridge Forward--> | Node3 | --
↪> Subscriber
                ---------                          ---------                          ---------
```

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。

7.1.1 EMQ 节点桥接配置

假设在本机创建两个 EMQ 节点，并创建一条桥接转发全部传感器(sensor)主题消息:

目录	节点	MQTT 端口
emqttd1	emqttd1@127.0.0.1	1883
emqttd2	emqttd2@127.0.0.1	2883

启动 emqttd1, emqttd2 节点:

```
cd emqttd1/ && ./bin/emqttd start
cd emqttd2/ && ./bin/emqttd start
```

emqttd1 节点上创建到 emqttd2 桥接:

```
$ ./bin/emqttd_ctl bridges start emqttd2@127.0.0.1 sensor/#

bridge is started.
```

(continues on next page)

(续上页)

```
$ ./bin/emqttd_ctl bridges list

bridge: emqttd1@127.0.0.1--sensor/#-->emqttd2@127.0.0.1
```

测试 emqttd1–sensor/#–>emqttd2 的桥接:

```
#emqttd2节点上

mosquitto_sub -t sensor/# -p 2883 -d

#emqttd1节点上

mosquitto_pub -t sensor/1/temperature -m "37.5" -d
```

删除桥接:

```
./bin/emqttd_ctl bridges stop emqttd2@127.0.0.1 sensor/#
```

7.2 mosquitto 桥接

mosquitto 可以普通 MQTT 连接方式，桥接到 emqttd 消息服务器:

7.2.1 mosquitto.conf

本机 2883 端口启动 emqttd 消息服务器，1883 端口启动 mosquitto 并创建桥接。

mosquitto.conf 配置:

```
connection emqttd
address 127.0.0.1:2883
topic sensor/# out 2

# Set the version of the MQTT protocol to use with for this bridge. Can be one
# of mqttv31 or mqttv311. Defaults to mqttv31.
bridge_protocol_version mqttv311
```

7.3 rsmb 桥接

本机 2883 端口启动 emqttd 消息服务器，1883 端口启动 rsmb 并创建桥接。

broker.cfg 桥接配置:

```
connection emqttd
addresses 127.0.0.1:2883
topic sensor/#
```

CHAPTER 8

用户指南 (User Guide)

8.1 MQTT 认证设置

EMQ 消息服务器认证由一系列认证插件(Plugin)提供，系统支持按用户名密码、ClientID 或匿名认证。

系统默认开启匿名认证(anonymous)，通过加载认证插件可开启的多个认证模块组成认证链:

注解: EMQ 2.0 消息服务器还提供了 MySQL、PostgreSQL、Redis、MongoDB、HTTP、LDAP 认证插件。

8.2 开启匿名认证

etc/emq.conf 配置启用匿名认证:

```
## Allow Anonymous authentication
mqtt.allow_anonymous = true
```

EMQ 2.0 版本提供的认证插件包括:

插件	说明
emq_auth_clientid	ClientId 认证/鉴权插件
emq_auth_username	用户名密码认证/鉴权插件
emq_auth_ldap	LDAP 认证/鉴权插件
emq_auth_http	HTTP 认证/鉴权插件
emq_auth_mysql	MySQ L认证/鉴权插件
emq_auth_pgsql	Postgre 认证/鉴权插件
emq_auth_redis	Redis 认证/鉴权插件
emq_auth_mongo	MongoDB 认证/鉴权插件

8.3 用户名密码认证

基于 MQTT 登录用户名(username)、密码(password)认证。

etc/plugins/emq_auth_username.conf 中配置默认用户:

```
auth.user.$N.username = admin
auth.user.$N.password = public
```

启用 emq_auth_username 插件:

```
./bin/emqttd_ctl plugins load emq_auth_username
```

使用 *./bin/emqttd_ctl users* 命令添加用户:

```
$ ./bin/emqttd_ctl users add <Username> <Password>
```

8.4 ClientId 认证

基于 MQTT 客户端 ID 认证。

etc/plugins/emq_auth_clientid.conf:

```
auth.client.$N.clientid = clientid
auth.client.$N.password = passwd
```

启用 emq_auth_clientid 插件:

```
./bin/emqttd_ctl plugins load emq_auth_clientid
```

8.5 LDAP 插件认证

etc/plugins/emq_auth_ldap.conf 配置 LDAP 参数:

```
auth.ldap.servers = 127.0.0.1

auth.ldap.port = 389
```

(continues on next page)

(续上页)

```
auth.ldap.timeout = 30

auth.ldap.user_dn = uid=%u,ou=People,dc=example,dc=com

auth.ldap.ssl = false
```

启用 LDAP 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_ldap
```

8.6 HTTP 插件认证

etc/plugins/emq_auth_http.conf 配置 'super_req', 'auth_req':

```
## Variables: %u = username, %c = clientid, %a = ipaddress, %P = password, %t = topic

auth.http.auth_req = http://127.0.0.1:8080/mqtt/auth
auth.http.auth_req.method = post
auth.http.auth_req.params = clientid=%c,username=%u,password=%P

auth.http.super_req = http://127.0.0.1:8080/mqtt/superuser
auth.http.super_req.method = post
auth.http.super_req.params = clientid=%c,username=%u
```

启用 HTTP 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_http
```

8.7 MySQL 插件认证

通过 MySQL 数据库表认证，可创建如下的 'mqtt_user' 表:

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(20) DEFAULT NULL,
  `is_superuser` tinyint(1) DEFAULT 0,
  `created` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

etc/plugins/emq_auth_mysql.conf 配置 'super_query', 'auth_query', 'password_hash':

```
## Mysql Server
auth.mysql.server = 127.0.0.1:3306

## Mysql Pool Size
auth.mysql.pool = 8
```

(continues on next page)

(续上页)

```
## Mysql Username
## auth.mysql.username =

## Mysql Password
## auth.mysql.password =

## Mysql Database
auth.mysql.database = mqtt

## Variables: %u = username, %c = clientid

## Authentication Query: select password only
auth.mysql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.mysql.password_hash = sha256

## %% Superuser Query
auth.mysql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1
```

注解: 如果系统已有MQTT认证表，可通过配置'auth_query'查询语句集成。

启用 MySQL 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_mysql
```

8.8 Postgre 插件认证

通过 PostgreSQL 数据库表认证，可创建如下的 'mqtt_user' 表:

```
CREATE TABLE mqtt_user (
  id SERIAL primary key,
  is_superuser boolean,
  username character varying(100),
  password character varying(100),
  salt character varying(40)
);
```

etc/plugins/emq_auth_pgsql.conf 配置 'auth_query'、'password_hash':

```
## Postgre Server
auth.pgsql.server = 127.0.0.1:5432

auth.pgsql.pool = 8

auth.pgsql.username = root

#auth.pgsql.password =

auth.pgsql.database = mqtt
```

(continues on next page)

(续上页)

```
auth.pgsql.encoding = utf8

auth.pgsql.ssl = false

## Variables: %u = username, %c = clientid, %a = ipaddress

## Authentication Query: select password only
auth.pgsql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.pgsql.password_hash = sha256

## sha256 with salt prefix
## auth.pgsql.password_hash = salt sha256

## sha256 with salt suffix
## auth.pgsql.password_hash = sha256 salt

## Superuser Query
auth.pgsql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1
```

启用 Postgre 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

8.9 Redis 插件认证

Redis 认证。MQTT 用户记录存储在 Redis Hash, 键值: “mqtt_user:<Username>”

etc/plugins/emq_auth_redis.conf 设置 ‘super_cmd’、’auth_cmd’、’password_hash’:

```
## Redis Server
auth.redis.server = 127.0.0.1:6379

## Redis Pool Size
auth.redis.pool = 8

## Redis Database
auth.redis.database = 0

## Redis Password
## auth.redis.password =

## Variables: %u = username, %c = clientid

## Authentication Query Command
auth.redis.auth_cmd = HGET mqtt_user:%u password

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.redis.password_hash = sha256

## Superuser Query Command
auth.redis.super_cmd = HGET mqtt_user:%u is_superuser
```

启用 Redis 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_redis
```

8.10 MongoDB 插件认证

按 MongoDB 用户集合认证，例如创建 'mqtt_user' 集合:

```
{
    username: "user",
    password: "password hash",
    is_superuser: boolean (true, false),
    created: "datetime"
}
```

etc/plugins/emq_auth_mongo.conf 设置 'super_query'、'auth_query':

```
## Mongo Server
auth.mongo.server = 127.0.0.1:27017

## Mongo Pool Size
auth.mongo.pool = 8

## Mongo User
## auth.mongo.user =

## Mongo Password
## auth.mongo.password =

## Mongo Database
auth.mongo.database = mqtt

## auth_query
auth.mongo.auth_query.collection = mqtt_user

auth.mongo.auth_query.password_field = password

auth.mongo.auth_query.password_hash = sha256

auth.mongo.auth_query.selector = username=%u

## super_query
auth.mongo.super_query.collection = mqtt_user

auth.mongo.super_query.super_field = is_superuser

auth.mongo.super_query.selector = username=%u
```

启用 MongoDB 认证插件:

```
./bin/emqttd_ctl plugins load emq_auth_mongo
```

8.11 访问控制(ACL)

EMQ 消息服务器通过 ACL(Access Control List) 实现 MQTT 客户端访问控制。

ACL 访问控制规则定义:

```
允许(Allow)|拒绝(Deny) 谁(Who) 订阅(Subscribe)|发布(Publish) 主题列表(Topics)
```

MQTT 客户端发起订阅/发布请求时，EMQ 消息服务器的访问控制模块，会逐条匹配 ACL 规则，直到匹配成功为止:

```
          ---------               ---------               ---------
Client -> | Rule1 | --nomatch--> | Rule2 | --nomatch--> | Rule3 | --> Default
          ---------               ---------               ---------
              |                       |                       |
            match                   match                   match
             \|/                     \|/                     \|/
       allow | deny            allow | deny            allow | deny
```

8.12 默认访问控制设置

EMQ 消息服务器默认访问控制，在 etc/emq.conf 中设置:

```
## ACL nomatch
mqtt.acl_nomatch = allow

## Default ACL File
mqtt.acl_file = etc/acl.conf
```

ACL 规则定义在 etc/acl.conf，EMQ 启动时加载到内存:

```
%% Allow 'dashboard' to subscribe '$SYS/#'
{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

%% Allow clients from localhost to subscribe any topics
{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

%% Deny clients to subscribe '$SYS#' and '#'
{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

%% Allow all by default
{allow, all}.
```

8.13 HTTP 插件访问控制

HTTP API 实现访问控制: https://github.com/emqtt/emq_auth_http

配置 etc/plugins/emq_auth_http.conf, 启用 HTTP 认证插件后:

```
## 'access' parameter: sub = 1, pub = 2
auth.http.acl_req = http://127.0.0.1:8080/mqtt/acl
auth.http.acl_req.method = get
auth.http.acl_req.params = access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t
```

8.14 MySQL 插件访问控制

MySQL 插件访问控制，通过 mqtt_acl 表定义 ACL 规则:

```
CREATE TABLE `mqtt_acl` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `allow` int(1) DEFAULT NULL COMMENT '0: deny, 1: allow',
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

etc/plugins/emq_auth_mysql.conf 配置 'acl_query' 与 'acl_nomatch':

```
## ACL Query Command
auth.mysql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'
```

8.15 Postgre 插件访问控制

PostgreSQL 插件访问控制，通过 mqtt_acl 表定义 ACL 规则:

```
CREATE TABLE mqtt_acl (
  id SERIAL primary key,
  allow integer,
  ipaddr character varying(60),
  username character varying(100),
  clientid character varying(100),
  access  integer,
  topic character varying(100)
);

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

etc/plugins/emq_auth_pgsql.conf 设置 'acl_query' 与 'acl_nomatch':

```
## ACL Query. Comment this query, the acl will be disabled.
auth.pgsql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'
```

8.16 Redis 插件访问控制

Redis Hash 存储一个 MQTT 客户端的访问控制规则:

```
HSET mqtt_acl:<username> topic1 1
HSET mqtt_acl:<username> topic2 2
HSET mqtt_acl:<username> topic3 3
```

etc/plugins/emq_auth_redis.conf 配置 'acl_cmd' 与 'acl_nomatch':

```
## ACL Query Command
auth.redis.acl_cmd = HGETALL mqtt_acl:%u
```

8.17 MongoDB 插件访问控制

MongoDB 数据库创建 *mqtt_acl* 集合:

```
{
    username: "username",
    clientid: "clientid",
    publish: ["topic1", "topic2", ...],
    subscribe: ["subtop1", "subtop2", ...],
    pubsub: ["topic/#", "topic1", ...]
}
```

mqtt_acl 集合插入数据，例如:

```
db.mqtt_acl.insert({username: "test", publish: ["t/1", "t/2"], subscribe: ["user/%u",
↪"client/%c"]})
db.mqtt_acl.insert({username: "admin", pubsub: ["#"]})
```

etc/plugins/emq_auth_mongo.conf 配置 'acl_query' 与 'acl_nomatch':

```
## acl_query
auth.mongo.acl_query.collection = mqtt_user

auth.mongo.acl_query.selector = username=%u
```

8.18 MQTT 发布订阅

MQTT 是为移动互联网、物联网设计的轻量发布订阅模式的消息服务器:

EMQ 消息服务器安装启动后，任何设备或终端的 MQTT 客户端，可通过 MQTT 协议连接到服务器，发布订阅消息方式互通。

MQTT 协议客户端库: https://github.com/mqtt/mqtt.github.io/wiki/libraries

例如，mosquitto_sub/pub 命令行发布订阅消息:

```
mosquitto_sub -t topic -q 2
mosquitto_pub -t topic -q 1 -m "Hello, MQTT!"
```

MQTT V3.1.1 版本协议规范: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

EMQ 消息服务器的 MQTT 协议 TCP 监听器，可在 etc/emq.conf 文件中设置:

```
## TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
listener.tcp.external = 1883

## Size of acceptor pool
listener.tcp.external.acceptors = 8

## Maximum number of concurrent clients
listener.tcp.external.max_clients = 1024
```

MQTT/SSL 监听器，缺省端口8883:

```
## SSL Listener: 8883, 127.0.0.1:8883, ::1:8883
listener.ssl.external = 8883

## Size of acceptor pool
listener.ssl.external.acceptors = 4

## Maximum number of concurrent clients
listener.ssl.external.max_clients = 512
```

8.19 HTTP 发布接口

EMQ 消息服务器提供了一个 HTTP 发布接口，应用服务器或 Web 服务器可通过该接口发布 MQTT 消息:

```
HTTP POST http://host:8080/mqtt/publish
```

Web 服务器例如 PHP/Java/Python/NodeJS 或 Ruby on Rails，可通过 HTTP POST 请求发布 MQTT 消息:

```
curl -v --basic -u user:passwd -d "qos=1&retain=0&topic=/a/b/c&message=hello from␣
↪http..." -k http://localhost:8080/mqtt/publish
```

HTTP 接口参数:

参数	说明
client	MQTT 客户端 ID
qos	QoS: 0 | 1 | 2
retain	Retain: 0 | 1
topic	主题(Topic)
message	消息

注解: HTTP 发布接口采用 Basic 认证

注解: 该接口在 v2.3-beta.2 版本变更为: 'api/v2/mqtt/publish', 详见文档: 管理监控*API (REST API)*

8.20 MQTT WebSocket 连接

EMQ 消息服务器支持 MQTT WebSocket 连接，Web 浏览器可直接通过 MQTT 协议连接服务器:

WebSocket URI:	ws(s)://host:8083/mqtt
Sec-WebSocket-Protocol:	'mqttv3.1' or 'mqttv3.1.1'

Dashboard 插件提供了一个 MQTT WebSocket 连接的测试页面:

```
http://127.0.0.1:18083/websocket.html
```

EMQ 通过内嵌的 HTTP 服务器，实现 MQTT/WebSocket，etc/emq.conf 设置:

```
## MQTT/WebSocket Listener
listener.ws.external = 8083
listener.ws.external.acceptors = 4
listener.ws.external.max_clients = 64
```

8.21 $SYS-系统主题

EMQ 消息服务器周期性发布自身运行状态、MQTT 协议统计、客户端上下线状态到 *$SYS/* 开头系统主题。$SYS 主题路径以 "$SYS/brokers/{node}/" 开头，'${node}' 是 Erlang 节点名称:

```
$SYS/brokers/emqttd@127.0.0.1/version

$SYS/brokers/emqttd@host2/uptime
```

注解: 默认只允许 localhost 的 MQTT 客户端订阅 $SYS 主题，可通过 etc/acl.config 修改访问控制规则。

$SYS 系统消息发布周期，通过 etc/emq.conf 配置:

```
## System Interval of publishing broker $SYS Messages
mqtt.broker.sys_interval = 60
```

8.21.1 服务器版本、启动时间与描述消息

主题	说明
$SYS/brokers	集群节点列表
$SYS/brokers/${node}/version	EMQ 服务器版本
$SYS/brokers/${node}/uptime	EMQ 服务器启动时间
$SYS/brokers/${node}/datetime	EMQ 服务器时间
$SYS/brokers/${node}/sysdescr	EMQ 服务器描述

8.21.2 MQTT 客户端上下线状态消息

$SYS 主题前缀: $SYS/brokers/${node}/clients/

主题(Topic)	数据(JSON)	说明
${clientid}/connected	**{ipaddress: “127.0.0.1”, username: “test”,** session: false, version: 3, connack: 0, ts: 1432648482}	Publish when a client connected

| ts: 1432648482} | |

username: “test”, ts: 1432749431} | |

‘connected’ 消息 JSON 数据:

```
{
    ipaddress: "127.0.0.1",
    username:  "test",
    session:   false,
    protocol:  3,
    connack:   0,
    ts:        1432648482
}
```

‘disconnected’ 消息 JSON 数据:

```
{
    reason: normal,
    ts:     1432648486
}
```

8.21.3 Statistics - 系统统计消息

系统主题前缀: $SYS/brokers/${node}/stats/

Clients - 客户端统计

主题(Topic)	说明
clients/count	当前客户端总数
clients/max	最大客户端数量

Sessions - 会话统计

主题(Topic)	说明
sessions/count	当前会话总数
sessions/max	最大会话数量

Subscriptions - 订阅统计

主题(Topic)	说明
subscriptions/count	当前订阅总数
subscriptions/max	最大订阅数量

Topics - 主题统计

主题(Topic)	说明
topics/count	当前 Topic 总数(跨节点)
topics/max	Max number of topics

8.21.4 Metrics - 收发流量/报文/消息统计

系统主题(Topic)前缀: $SYS/brokers/${node}/metrics/

收发流量统计

主题(Topic)	说明
bytes/received	累计接收流量
bytes/sent	累计发送流量

MQTT报文收发统计

主题(Topic)	说明
packets/received	累计接收 MQTT 报文
packets/sent	累计发送 MQTT 报文
packets/connect	累计接收 MQTT CONNECT 报文
packets/connack	累计发送 MQTT CONNACK 报文
packets/publish/received	累计接收 MQTT PUBLISH 报文
packets/publish/sent	累计发送 MQTT PUBLISH 报文
packets/subscribe	累计接收 MQTT SUBSCRIBE 报文
packets/suback	累计发送 MQTT SUBACK 报文
packets/unsubscribe	累计接收 MQTT UNSUBSCRIBE 报文
packets/unsuback	累计发送 MQTT UNSUBACK 报文
packets/pingreq	累计接收 MQTT PINGREQ 报文
packets/pingresp	累计发送 MQTT PINGRESP 报文
packets/disconnect	累计接收 MQTT DISCONNECT 报文

MQTT 消息收发统计

主题(Topic)	说明
messages/received	累计接收消息
messages/sent	累计发送消息
messages/retained	Retained 消息总数
messages/dropped	丢弃消息总数

8.21.5 Alarms - 系统告警

系统主题(Topic)前缀: $SYS/brokers/${node}/alarms/

主题(Topic)	说明
${alarmId}/alert	新产生告警
${alarmId}/clear	清除告警

8.21.6 Sysmon - 系统监控

系统主题(Topic)前缀: $SYS/brokers/${node}/sysmon/

主题(Topic)	说明
long_gc	GC 时间过长警告
long_schedule	调度时间过长警告
large_heap	Heap 内存占用警告
busy_port	Port 忙警告
busy_dist_port	Dist Port 忙警告

8.22 追踪

EMQ 消息服务器支持追踪来自某个客户端(Client)的全部报文，或者发布到某个主题(Topic)的全部消息。

追踪客户端(Client):

```
./bin/emqttd_ctl trace client "clientid" "trace_clientid.log"
```

追踪主题(Topic):

```
./bin/emqttd_ctl trace topic "topic" "trace_topic.log"
```

查询追踪:

```
./bin/emqttd_ctl trace list
```

停止追踪:

```
./bin/emqttd_ctl trace client "clientid" off

./bin/emqttd_ctl trace topic "topic" off
```

CHAPTER 9

高级特性 (Advanced Features)

EMQ 2.0 版本新增了本地订阅与共享订阅功能。

9.1 本地订阅 (Local Subscription)

本地订阅(Local Subscription) 只在本节点创建订阅与路由表，不会在集群节点间广播全局路由，非常适合物联网数据采集应用:

```
mosquitto_sub -t '$local/topic'

mosquitto_pub -t 'topic'
```

使用方式: 订阅者在主题(Topic)前增加 '$local/' 前缀。

9.2 共享订阅 (Shared Subscription)

共享订阅(Shared Subscription)支持在多订阅者间采用分组负载平衡方式派发消息:

```
                            ---------
                            |       | --Msg1--> Subscriber1
Publisher--Msg1,Msg2,Msg3-->|  EMQ  | --Msg2--> Subscriber2
                            |       | --Msg3--> Subscriber3
                            ---------
```

共享订阅支持两种使用方式:

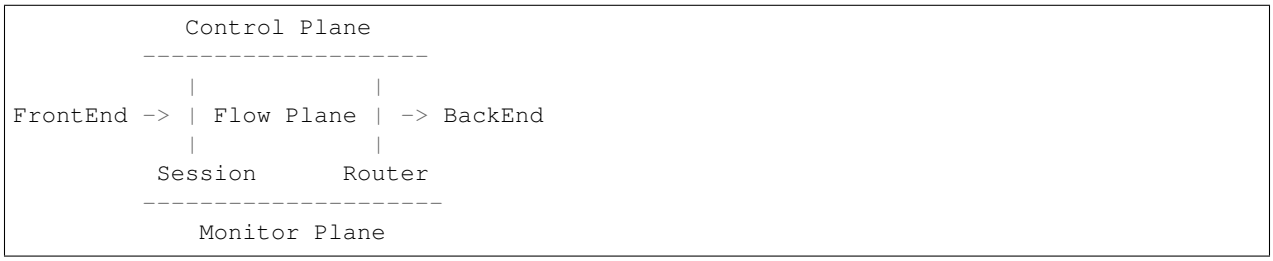
订阅前缀	使用示例
$queue/	mosquitto_sub -t '$queue/topic'
$share/<group>/	mosquitto_sub -t '$share/group/topic'

CHAPTER 10

架构设计 (Design)

10.1 前言

EMQ 2.0 消息服务器设计，在 1.x 版本的基础上，首先分离了前端协议(FrontEnd)与后端集成(Backend)，其次分离了消息路由平面(Flow Plane)与监控管理平面(Monitor/Control Plane)。EMQ 2.0 消息服务器将在 1.x 版本支持100万 MQTT 连接的基础上，向可管理可监控坚如磐石的稳定性方向迭代演进:



10.1.1 100万连接

多核服务器和现代操作系统内核层面，可以很轻松支持100万 TCP 连接，核心问题是应用层面如何处理业务瓶颈。

EMQ 消息服务器在业务和应用层面，解决了承载100万连接的各类瓶颈问题。连接测试的操作系统内核、TCP 协议栈、Erlang 虚拟机参数: http://docs.emqtt.cn/zh_CN/latest/tune.html

10.1.2 全异步架构

EMQ 消息服务器是基于 Erlang/OTP 平台的全异步的架构：异步 TCP 连接处理、异步主题(Topic)订阅、异步消息发布。只有在资源负载限制部分采用同步设计，比如 TCP 连接创建和 Mnesia 数据库事务执行。

一条 MQTT 消息从发布者(Publisher)到订阅者(Subscriber)，在 EMQ 消息服务器内部异步流过一系列 Erlang 进程 Mailbox:

```
                  ----------          -----------          ----------
Publisher --Msg-->| Client | --Msg--> | Session | --Msg--> | Client | --Msg-->␣
↪Subscriber
                  ----------          -----------          ----------
```

10.1.3 消息持久化

EMQ 1.0 版本不支持服务器内部消息持久化，这是一个架构设计选择。首先，EMQ 解决的核心问题是连接与路由；其次，我们认为内置持久化是个错误设计。

传统内置消息持久化的 MQ 服务器，比如广泛使用的 JMS 服务器 ActiveMQ，几乎每个大版本都在重新设计持久化部分。内置消息持久化在设计上有两个问题:

1. 如何平衡内存与磁盘使用？消息路由基于内存，消息存储是基于磁盘。
2. 多服务器分布集群架构下，如何放置 Queue 如何复制 Queue 的消息?

Kafka 在上述问题上，做出了正确的设计：一个完全基于磁盘分布式 Commit Log 的消息服务器。

EMQ 2.0 版本将发布 EMQ X 平台产品，支持消息持久化到 Redis、Kafka、Cassandra、PostgreSQL 等数据库。

设计上分离消息路由与消息存储职责后，数据复制容灾备份甚至应用集成，可以在数据层面灵活实现。

10.1.4 NetSplit问题

EMQ 1.0 消息服务器集群，基于 Mnesia 数据库设计。NetSplit 发生时，节点间状态是：Erlang 节点间可以连通，互相询问自己是否宕机，对方回答你已经宕机:(

NetSplit 故障发生时，EMQ 消息服务器的 log/emqttd_error.log 日志，会打印 critical 级别日志:

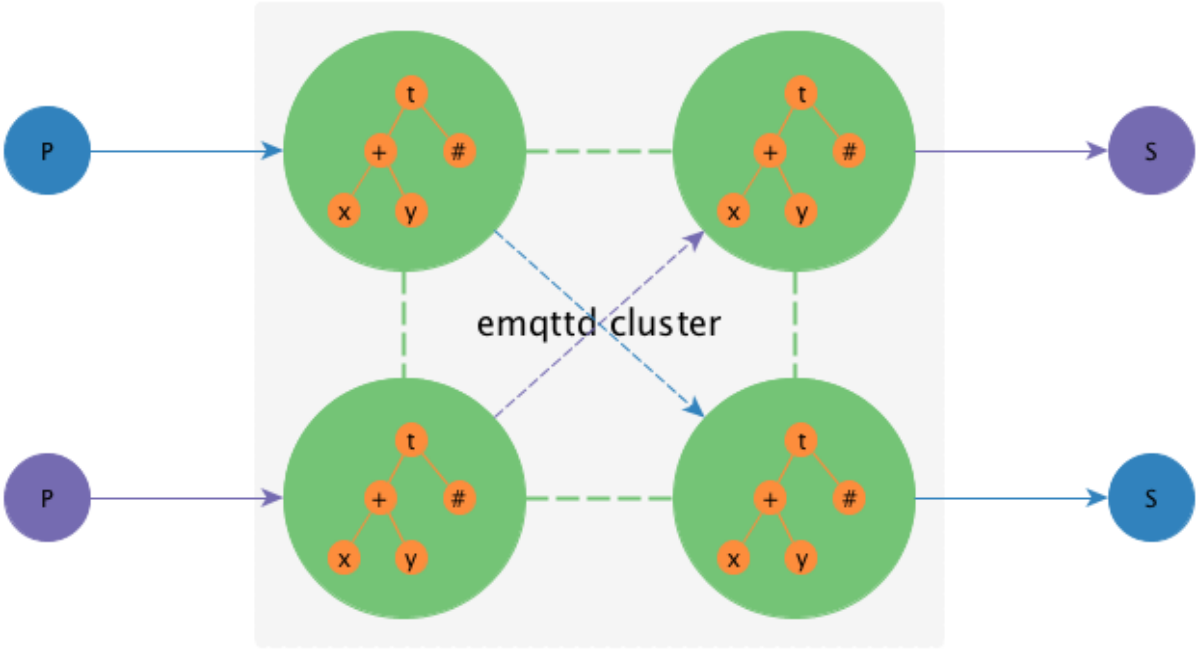
```
Mnesia inconsistent_database event: running_partitioned_network, emqttd@host
```

EMQ 集群部署在同一 IDC 网络下，NetSplit 发生的几率很低，一旦发生又很难自动处理。所以 EMQ 1.0 版本设计选择是，集群不自动化处理 NetSplit，需要人工重启部分节点。

10.2 系统架构

10.2.1 概念模型

EMQ 消息服务器概念上更像一台网络路由器(Router)或交换机(Switch)，而不是传统的企业级消息服务器(MQ)。相比网络路由器按 IP 地址或 MPLS 标签路由报文，EMQ 按主题树(Topic Trie)发布订阅模式在集群节点间路由 MQTT 消息:



10.2.2 设计原则

1. EMQ 消息服务器核心解决的问题：处理海量的并发 MQTT 连接与路由消息。
2. 充分利用 Erlang/OTP 平台软实时、低延时、高并发、分布容错的优势。
3. 连接(Connection)、会话(Session)、路由(Router)、集群(Cluster)分层。
4. 消息路由平面(Flow Plane)与控制管理平面(Control Plane)分离。
5. 支持后端数据库或 NoSQL 实现数据持久化、容灾备份与应用集成。

10.2.3 系统分层

1. 连接层(Connection Layer)：负责 TCP 连接处理、 MQTT 协议编解码。
2. 会话层(Session Layer)：处理 MQTT 协议发布订阅消息交互流程。
3. 路由层(Route Layer)：节点内路由派发 MQTT 消息。
4. 分布层(Distributed Layer)：分布节点间路由 MQTT 消息。
5. 认证与访问控制(ACL)：连接层支持可扩展的认证与访问控制模块。
6. 钩子(Hooks)与插件(Plugins)：系统每层提供可扩展的钩子，支持插件方式扩展服务器。

10.3 连接层设计

连接层处理服务端 Socket 连接与 MQTT 协议编解码：

1. 基于 eSockd 框架的异步 TCP 服务端

2. TCP Acceptor 池与异步 TCP Accept
3. TCP/SSL, WebSocket/SSL 连接支持
4. 最大并发连接数限制
5. 基于 IP 地址(CIDR)访问控制
6. 基于 Leaky Bucket 的流控
7. MQTT 协议编解码
8. MQTT 协议心跳检测
9. MQTT 协议报文处理

10.4 会话层设计

会话层处理 MQTT 协议发布订阅(Publish/Subscribe)业务交互流程:

1. 缓存 MQTT 客户端的全部订阅(Subscription)，并终结订阅 QoS
2. 处理 Qos0/1/2 消息接收与下发，消息超时重传与离线消息保存
3. 飞行窗口(Inflight Window)，下发消息吞吐控制与顺序保证
4. 保存服务器发送到客户端的，已发送未确认的 Qos1/2 消息
5. 缓存客户端发送到服务端，未接收到 PUBREL 的 QoS2 消息
6. 客户端离线时，保存持久会话的离线 Qos1/2 消息

10.4.1 消息队列与飞行窗口

会话层通过一个内存消息队列和飞行窗口处理下发消息:

```
      |<----------------- Max Len ----------------->|
      ------------------------------------------------
IN -> |      Messages Queue   |  Inflight Window    | -> Out
      ------------------------------------------------
                              |<---   Win Size  --->|
```

飞行窗口(Inflight Window)保存当前正在发送未确认的 Qos1/2 消息。窗口值越大，吞吐越高；窗口值越小，消息顺序越严格。

当客户端离线或者飞行窗口(Inflight Window)满时，消息缓存到队列。如果消息队列满，先丢弃 Qos0 消息或最早进入队列的消息。

10.4.2 报文 ID 与消息 ID

MQTT 协议定义了一个 16bits 的报文 ID(PacketId)，用于客户端到服务器的报文收发与确认。MQTT 发布报文(PUBLISH)进入消息服务器后，转换为一个消息对象并分配 128bits 消息 ID(MessageId)。

全局唯一时间序列消息 ID 结构:

1. 64bits 时间戳: erlang:system_time if Erlang >= R18, otherwise os:timestamp
2. Erlang 节点 ID: 编码为2字节
3. Erlang 进程 PID: 编码为4字节

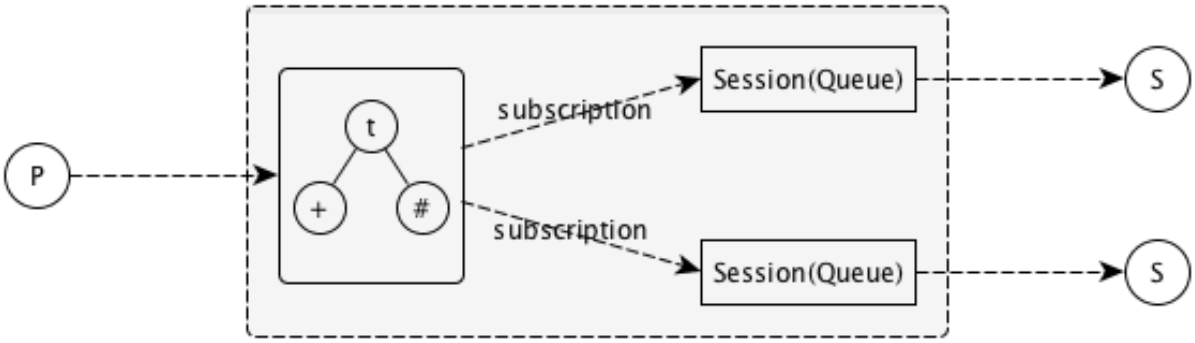
4. 进程内部序列号: 2字节的进程内部序列号

端到端消息发布订阅(Pub/Sub)过程中，发布报文 ID 与报文 QoS 终结在会话层，由唯一 ID 标识的 MQTT 消息对象在节点间路由:

```
PktId <-- Session --> MsgId <-- Router --> MsgId <-- Session --> PktId
```

10.5 路由层设计

路由层维护订阅者(subscriber)与订阅关系表(subscription)，并在本节点发布订阅模式派发(Dispatch)消息:



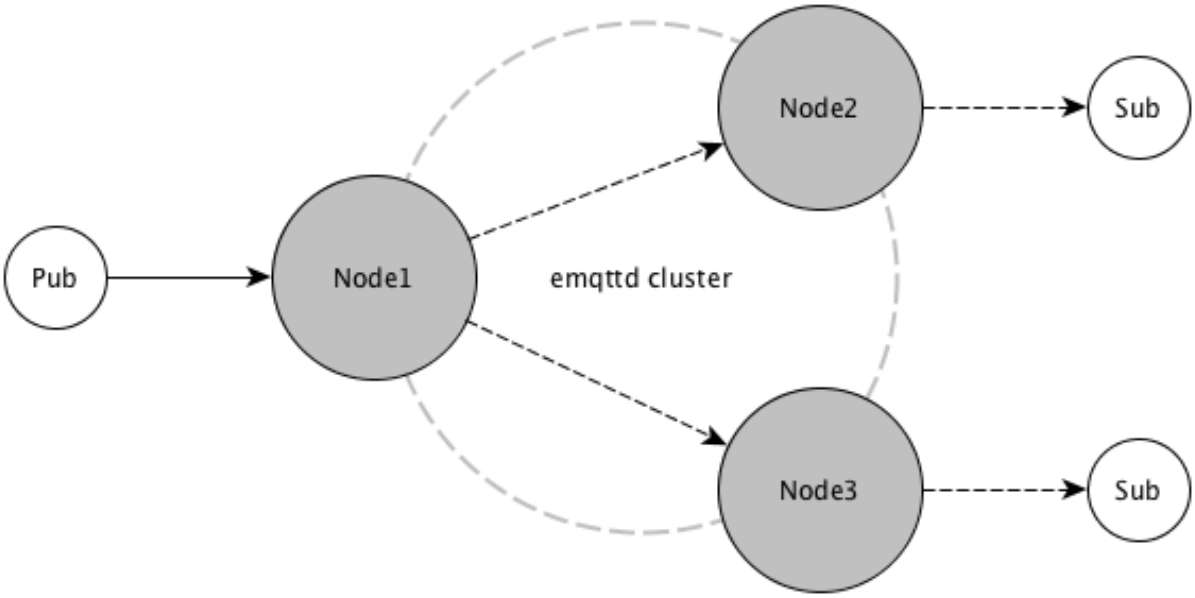
消息派发到会话(Session)后，由会话负责按不同 QoS 送达消息。

10.6 分布层设计

分布层维护全局主题树(Topic Trie)与路由表(Route Table)。主题树由通配主题构成，路由表映射主题到节点:

```
-------------------------
|             t         |
|            / \        |
|           +   #       |
|          /  \         |
|       x       y       |
-------------------------
| t/+/x -> node1, node3 |
| t/+/y -> node1        |
| t/#   -> node2        |
| t/a   -> node3        |
-------------------------
```

分布层通过匹配主题树(Topic Trie)和查找路由表(Route Table)，在集群的节点间转发路由 MQTT 消息:



10.7 认证与访问控制设计

EMQ 消息服务器支持可扩展的认证与访问控制，由 emqttd_access_control、emqttd_auth_mod 和emqttd_acl_mod 模块实现。

emqttd_access_control 模块提供了注册认证扩展接口:

```
register_mod(auth | acl, atom(), list()) -> ok | {error, any()}.

register_mod(auth | acl, atom(), list(), non_neg_integer()) -> ok | {error, any()}.
```

10.7.1 认证扩展模块

emqttd_auth_mod 定义认证扩展模块 Behavihour:

```
-module(emqttd_auth_mod).

-ifdef(use_specs).

-callback init(AuthOpts :: list()) -> {ok, State :: any()}.

-callback check(Client, Password, State) -> ok | ignore | {error, string()} when
    Client    :: mqtt_client(),
    Password  :: binary(),
    State     :: any().

-callback description() -> string().

-else.

-export([behaviour_info/1]).
```

(continues on next page)

(续上页)

```
behaviour_info(callbacks) ->
    [{init, 1}, {check, 3}, {description, 0}];
behaviour_info(_Other) ->
    undefined.

-endif.
```

EMQ 消息服务器自身实现的认证模块/插件包括:

模块/插件	认证方式
emq_auth_username	用户名、密码认证插件
emq_auth_clientid	ClientID、密码认证插件

10.7.2 访问控制(ACL)

emqttd_acl_mod 模块定义访问控制 Behavihour:

```
-module(emqttd_acl_mod).

-include("emqttd.hrl").

-ifdef(use_specs).

-callback init(AclOpts :: list()) -> {ok, State :: any()}.

-callback check_acl({Client, PubSub, Topic}, State :: any()) -> allow | deny | ignore␣
↪when
    Client   :: mqtt_client(),
    PubSub   :: pubsub(),
    Topic    :: binary().

-callback reload_acl(State :: any()) -> ok | {error, any()}.

-callback description() -> string().

-else.

-export([behaviour_info/1]).

behaviour_info(callbacks) ->
    [{init, 1}, {check_acl, 2}, {reload_acl, 1}, {description, 0}];
behaviour_info(_Other) ->
    undefined.

-endif.
```

emqttd_acl_internal 模块实现缺省的基于 etc/acl.conf 文件的访问控制:

```
%%%-----------------------------------------------------------------------------
%%%
%%% -type who() :: all | binary() |
%%%                {ipaddr, esockd_access:cidr()} |
%%%                {client, binary()} |
```

(continues on next page)

(续上页)

```
%%%                 {user, binary()}.
%%%
%%% -type access() :: subscribe | publish | pubsub.
%%%
%%% -type topic() :: binary().
%%%
%%% -type rule() :: {allow, all} |
%%%                 {allow, who(), access(), list(topic())} |
%%%                 {deny, all} |
%%%                 {deny, who(), access(), list(topic())}.
%%%
%%%----------------------------------------------------------------------------

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

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

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

{allow, all}.
```

10.8 钩子(Hook)设计

10.8.1 钩子(Hook)定义

EMQ 消息服务器在客户端上下线、主题订阅、消息收发位置设计了扩展钩子(Hook):

钩子	说明
client.connected	客户端上线
client.subscribe	客户端订阅主题前
client.unsubscribe	客户端取消订阅主题
session.subscribed	客户端订阅主题后
session.unsubscribed	客户端取消订阅主题后
message.publish	MQTT 消息发布
message.delivered	MQTT 消息送达
message.acked	MQTT 消息回执
client.disconnected	客户端连接断开

钩子(Hook) 采用职责链设计模式(Chain-of-responsibility_pattern)，扩展模块或插件向钩子注册回调函数，系统在客户端上下线、主题订阅或消息发布确认时，触发钩子顺序执行回调函数:

```
                 --------  ok | {ok, NewAcc}   --------  ok | {ok, NewAcc}   --------
(Args, Acc) --> | Fun1 | ------------------> | Fun2 | ------------------> | Fun3 | -
↪-> {ok, Acc} | {stop, Acc}
                 --------                     --------                     --------
                    |                            |                            |
                stop | {stop, NewAcc}        stop | {stop, NewAcc}        stop |
↪{stop, NewAcc}
```

不同钩子的回调函数输入参数不同，用户可参考插件模版的 **'emqttd_plugin_template'_** 模块，每个回调函数应该返回:

返回	说明
ok	继续执行
{ok, NewAcc}	返回累积参数继续执行
stop	停止执行
{stop, NewAcc}	返回累积参数停止执行

10.8.2 钩子(Hook)实现

emqttd 模块封装了 Hook 接口:

```
-module(emqttd).

%% Hooks API
-export([hook/4, hook/3, unhook/2, run_hooks/3]).
hook(Hook :: atom(), Callback :: function(), InitArgs :: list(any())) -> ok | {error,␣
↪any()}.

hook(Hook :: atom(), Callback :: function(), InitArgs :: list(any()), Priority ::␣
↪integer()) -> ok | {error, any()}.

unhook(Hook :: atom(), Callback :: function()) -> ok | {error, any()}.

run_hooks(Hook :: atom(), Args :: list(any()), Acc :: any()) -> {ok | stop, any()}.
```

emqttd_hook 模块实现 Hook 机制:

```
-module(emqttd_hook).

%% Hooks API
-export([add/3, add/4, delete/2, run/3, lookup/1]).

add(HookPoint :: atom(), Callback :: function(), InitArgs :: list(any())) -> ok.

add(HookPoint :: atom(), Callback :: function(), InitArgs :: list(any()), Priority ::␣
↪integer()) -> ok.

delete(HookPoint :: atom(), Callback :: function()) -> ok.

run(HookPoint :: atom(), Args :: list(any()), Acc :: any()) -> any().

lookup(HookPoint :: atom()) -> [#callback{}].
```

10.8.3 钩子(Hook)使用

emq_plugin_template 提供了全部钩子的使用示例，例如端到端的消息处理回调:

```
-module(emq_plugin_template).

-export([load/1, unload/0]).

-export([on_message_publish/2, on_message_delivered/4, on_message_acked/4]).

load(Env) ->
```

(continues on next page)

(续上页)

```
    emqttd:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqttd:hook('message.delivered', fun ?MODULE:on_message_delivered/4, [Env]),
    emqttd:hook('message.acked', fun ?MODULE:on_message_acked/4, [Env]).

on_message_publish(Message, _Env) ->
    io:format("publish ~s~n", [emqttd_message:format(Message)]),
    {ok, Message}.

on_message_delivered(ClientId, _Username, Message, _Env) ->
    io:format("delivered to client ~s: ~s~n", [ClientId, emqttd_
↪message:format(Message)]),
    {ok, Message}.

on_message_acked(ClientId, _Username, Message, _Env) ->
    io:format("client ~s acked: ~s~n", [ClientId, emqttd_message:format(Message)]),
    {ok, Message}.

unload() ->
    emqttd:unhook('message.publish', fun ?MODULE:on_message_publish/2),
    emqttd:unhook('message.acked', fun ?MODULE:on_message_acked/4),
    emqttd:unhook('message.delivered', fun ?MODULE:on_message_delivered/4).
```

10.9 插件(Plugin)设计

插件是一个可以被动态加载的普通 Erlang 应用(Application)。插件主要通过钩子(Hook)机制扩展服务器功能，或通过注册扩展模块方式集成认证访问控制。

emqttd_plugins 模块实现插件机制，提供加载卸载插件 API

```
-module(emqttd_plugins).

-export([load/1, unload/1]).

%% @doc Load a Plugin
load(PluginName :: atom()) -> ok | {error, any()}.

%% @doc UnLoad a Plugin
unload(PluginName :: atom()) -> ok | {error, any()}.
```

用户可通过 *./bin/emqttd_ctl* 命令行加载卸载插件:

```
./bin/emqttd_ctl plugins load emq_auth_redis

./bin/emqttd_ctl plugins unload emq_auth_redis
```

开发者请参考模版插件: http://github.com/emqtt/emqttd_plugin_template

10.10 Mnesia/ETS 表设计

表	类型	描述
mqtt_trie	mnesia	Trie Table
mqtt_trie_node	mnesia	Trie Node Table
mqtt_route	mnesia	Global Route Table
mqtt_local_route	mnesia	Local Route Table
mqtt_pubsub	ets	PubSub Tab
mqtt_subscriber	ets	Subscriber Tab
mqtt_subscription	ets	Subscription Tab
mqtt_session	mnesia	Global Session Table
mqtt_local_session	ets	Local Session Table
mqtt_client	ets	Client Table
mqtt_retained	mnesia	Retained Message Table

10.11 Erlang 设计相关

1. 使用 Pool, Pool, Pool... 推荐 GProc 库: https://github.com/uwiger/gproc
2. 异步，异步，异步消息...连接层到路由层异步消息，同步请求用于负载保护
3. 避免进程 Mailbox 累积消息，负载高的进程可以使用 gen_server2
4. 消息流经的 Socket 连接、会话进程必须 Hibernate，主动回收 binary 句柄
5. 多使用 Binary 数据，避免进程间内存复制
6. 使用 ETS, ETS, ETS... Message Passing vs ETS
7. 避免 ETS 表非键值字段 select, match
8. 避免大量数据 ETS 读写, 每次 ETS 读写会复制内存，可使用 lookup_element, update_counter
9. 适当开启 ETS 表 {write_concurrency, true}
10. 保护 Mnesia 数据库事务，尽量减少事务数量，避免事务过载(overload)
11. 避免 Mnesia 数据表索引，和非键值字段 match, select

CHAPTER 11

管理命令 (Commands)

EMQ 消息服务器提供了 './bin/emqttd_ctl' 的管理命令行。

11.1 status 命令

查询 EMQ 消息服务器运行状态:

```
$ ./bin/emqttd_ctl status

Node 'emqttd@127.0.0.1' is started
emqttd 2.0 is running
```

11.2 broker 命令

broker 命令查询服务器基本信息，启动时间，统计数据与性能数据。

broker	查询 EMQ 消息服务器描述、版本、启动时间
broker pubsub	查询核心的 Erlang PubSub 进程状态(调试)
broker stats	查询连接(Client)、会话(Session)、主题(Topic)、 订阅(Subscription)、路由(Route)统计信息
broker metrics	查询 MQTT 报文(Packet)、消息(Message)收发统计

查询 EMQ 消息服务器基本信息包括版本、启动时间等:

```
$ ./bin/emqttd_ctl broker

sysdescr  : Erlang MQTT Broker
version   : 2.0
```

(continues on next page)

(续上页)

```
uptime    : 25 seconds
datetime  : 2016-10-18 10:42:10
```

11.2.1 broker stats

查询服务器客户端连接(Client)、会话(Session)、主题(Topic)、订阅(Subscription)、路由(Route)统计:

```
$ ./bin/emqttd_ctl broker stats

clients/count       : 1
clients/max         : 1
queues/count        : 0
queues/max          : 0
retained/count      : 2
retained/max        : 2
routes/count        : 2
routes/reverse      : 2
sessions/count      : 0
sessions/max        : 0
subscriptions/count : 1
subscriptions/max   : 1
topics/count        : 54
topics/max          : 54
```

11.2.2 broker metrics

查询服务器流量(Bytes)、MQTT报文(Packets)、消息(Messages)收发统计:

```
$ ./bin/emqttd_ctl broker metrics

bytes/received          : 297
bytes/sent              : 40
messages/dropped        : 348
messages/qos0/received  : 0
messages/qos0/sent      : 0
messages/qos1/received  : 0
messages/qos1/sent      : 0
messages/qos2/received  : 0
messages/qos2/sent      : 0
messages/received       : 0
messages/retained       : 2
messages/sent           : 0
packets/connack         : 5
packets/connect         : 5
packets/disconnect      : 0
packets/pingreq         : 0
packets/pingresp        : 0
packets/puback/received : 0
packets/puback/sent     : 0
packets/pubcomp/received: 0
packets/pubcomp/sent    : 0
packets/publish/received: 0
packets/publish/sent    : 0
```

(continues on next page)

(续上页)

```
packets/pubrec/received : 0
packets/pubrec/sent     : 0
packets/pubrel/received : 0
packets/pubrel/sent     : 0
packets/received        : 9
packets/sent            : 9
packets/suback          : 4
packets/subscribe       : 4
packets/unsuback        : 0
packets/unsubscribe     : 0
```

11.3 cluster 命令

cluster 命令集群多个 EMQ 消息服务器节点(进程):

cluster join <Node>	加入集群
cluster leave	离开集群
cluster remove <Node>	从集群删除节点
cluster status	查询集群状态

cluster 命令集群本机两个 EMQ 节点示例:

目录	节点名	MQTT 端口
emqttd1	emqttd1@127.0.0.1	1883
emqttd2	emqttd2@127.0.0.1	2883

启动 emqttd1

```
cd emqttd1 && ./bin/emqttd start
```

启动 emqttd2

```
cd emqttd2 && ./bin/emqttd start
```

emqttd2 节点与 emqttd1 集群，emqttd2 目录下:

```
$ ./bin/emqttd_ctl cluster join emqttd1@127.0.0.1

Join the cluster successfully.
Cluster status: [{running_nodes,['emqttd1@127.0.0.1','emqttd2@127.0.0.1']}]
```

任意节点目录下查询集群状态:

```
$ ./bin/emqttd_ctl cluster status

Cluster status: [{running_nodes,['emqttd2@127.0.0.1','emqttd1@127.0.0.1']}]
```

集群消息路由测试:

```
# emqttd1节点上订阅x
mosquitto_sub -t x -q 1 -p 1883

# emqttd2节点上向x发布消息
mosquitto_pub -t x -q 1 -p 2883 -m hello
```

emqttd2 节点离开集群:

```
cd emqttd2 && ./bin/emqttd_ctl cluster leave
```

emqttd1 节点下删除 emqttd2:

```
cd emqttd1 && ./bin/emqttd_ctl cluster remove emqttd2@127.0.0.1
```

11.4 clients 命令

clients 命令查询连接的 MQTT 客户端。

clients list	查询全部客户端连接
clients show <ClientId>	根据 ClientId 查询客户端
clients kick <ClientId>	根据 ClientId 踢出客户端

11.4.1 clients list

查询全部客户端连接:

```
$ ./bin/emqttd_ctl clients list

Client(mosqsub/43832-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64896, connected_at=1452929113)
Client(mosqsub/44011-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64961, connected_at=1452929275)
...
```

返回 Client 对象的属性:

clean_sess	清除会话标记
username	用户名
peername	对端 TCP 地址
connected_at	客户端连接时间

11.4.2 clients show <ClientId>

根据 ClientId 查询客户端:

```
./bin/emqttd_ctl clients show "mosqsub/43832-airlee.lo"

Client(mosqsub/43832-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64896, connected_at=1452929113)
```

11.4.3 clients kick <ClientId>

根据 ClientId 踢出客户端:

```
./bin/emqttd_ctl clients kick "clientid"
```

11.5 sessions 命令

sessions 命令查询 MQTT 连接会话。EMQ 消息服务器会为每个连接创建会话，clean_session 标记 true，创建临时(transient)会话；clean_session 标记为 false，创建持久会话(persistent)。

sessions list	查询全部会话
sessions list persistent	查询全部持久会话
sessions list transient	查询全部临时会话
sessions show <ClientId>	根据 ClientID 查询会话

11.5.1 sessions list

查询全部会话:

```
$ ./bin/emqttd_ctl sessions list

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
Session(mosqsub/44101-airlee.lo, clean_sess=true, max_inflight=100, inflight_queue=0,␣
↪message_queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,
↪ created_at=1452935401)
```

返回 Session 对象属性:

clean_sess	false: 持久会话，true: 临时会话
max_inflight	飞行窗口(最大允许同时下发消息数)
inflight_queue	当前正在下发的消息数
message_queue	当前缓存消息数
message_dropped	会话丢掉的消息数
awaiting_rel	等待客户端发送 PUBREL 的 QoS2 消息数
awaiting_ack	等待客户端响应 PUBACK 的 QoS1/2 消息数
awaiting_comp	等待客户端响应 PUBCOMP 的 QoS2 消息数
created_at	会话创建时间戳

11.5.2 sessions list persistent

查询全部持久会话:

```
$ ./bin/emqttd_ctl sessions list persistent

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
```

(continues on next page)

(续上页)

11.5.3 sessions list transient

查询全部临时会话:

```
$ ./bin/emqttd_ctl sessions list transient

Session(mosqsub/44101-airlee.lo, clean_sess=true, max_inflight=100, inflight_queue=0,␣
↪message_queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,
↪ created_at=1452935401)
```

11.5.4 sessions show <ClientId>

根据 ClientId 查询会话:

```
$ ./bin/emqttd_ctl sessions show clientid

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
```

11.6 routes 命令

routes 命令查询路由表。

11.6.1 routes list

查询全部路由:

```
$ ./bin/emqttd_ctl routes list

t2/# -> emqttd2@127.0.0.1
t/+/x -> emqttd2@127.0.0.1,emqttd@127.0.0.1
```

11.6.2 routes show <Topic>

根据 Topic 查询一条路由:

```
$ ./bin/emqttd_ctl routes show t/+/x

t/+/x -> emqttd2@127.0.0.1,emqttd@127.0.0.1
```

11.7 topics 命令

topics 命令查询当前的主题(Topic)表。

11.7.1 topics list

查询全部主题(Topic):

```
$ ./bin/emqttd_ctl topics list

$SYS/brokers/emqttd@127.0.0.1/metrics/packets/subscribe: static
$SYS/brokers/emqttd@127.0.0.1/stats/subscriptions/max: static
$SYS/brokers/emqttd2@127.0.0.1/stats/subscriptions/count: static
...
```

11.7.2 topics show <Topic>

查询某个主题(Topic):

```
$ ./bin/emqttd_ctl topics show '$SYS/brokers'

$SYS/brokers: static
```

11.8 subscriptions 命令

subscriptions 命令查询消息服务器的订阅(Subscription)表。

subscriptions list	查询全部订阅
subscriptions show <ClientId>	查询某个 ClientId 的订阅

11.8.1 subscriptions list

查询全部订阅:

```
$ ./bin/emqttd_ctl subscriptions list

mosqsub/91042-airlee.lo -> t/y:1
mosqsub/90475-airlee.lo -> t/+/x:2
```

11.8.2 subscriptions show <ClientId>

查询某个 Client 的订阅:

```
$ ./bin/emqttd_ctl subscriptions show 'mosqsub/90475-airlee.lo'

mosqsub/90475-airlee.lo -> t/+/x:2
```

11.9 plugins 命令

plugins 命令用于加载、卸载、查询插件应用。EMQ 消息服务器通过插件扩展认证、定制功能，插件置于 plugins/ 目录下。

plugins list	列出全部插件(Plugin)
plugins load <Plugin>	加载插件(Plugin)
plugins unload <Plugin>	卸载插件(Plugin)

11.9.1 plugins list

列出全部插件:

```
$ ./bin/emqttd_ctl plugins list

Plugin(emqttd_dashboard, version=0.16.0, description=emqttd web dashboard,␣
↪active=true)
Plugin(emqttd_plugin_mysql, version=0.16.0, description=emqttd Authentication/ACL␣
↪with MySQL, active=false)
Plugin(emqttd_plugin_pgsql, version=0.16.0, description=emqttd PostgreSQL Plugin,␣
↪active=false)
Plugin(emqttd_plugin_redis, version=0.16.0, description=emqttd Redis Plugin,␣
↪active=false)
Plugin(emqttd_plugin_template, version=0.16.0, description=emqttd plugin template,␣
↪active=false)
Plugin(emqttd_recon, version=0.16.0, description=emqttd recon plugin, active=false)
Plugin(emqttd_stomp, version=0.16.0, description=Stomp Protocol Plugin for emqttd␣
↪broker, active=false)
```

插件属性:

version	插件版本
description	插件描述
active	是否已加载

11.9.2 load <Plugin>

加载插件:

```
$ ./bin/emqttd_ctl plugins load emq_recon

Start apps: [recon,emq_recon]
Plugin emqttd_recon loaded successfully.
```

11.9.3 unload <Plugin>

卸载插件:

```
$ ./bin/emqttd_ctl plugins unload emq_recon

Plugin emq_recon unloaded successfully.
```

11.10 bridges 命令

bridges 命令用于在多台 EMQ 服务器节点间创建桥接:

```
                ---------                          ---------
Publisher --> | node1 | --Bridge Forward--> | node2 | --> Subscriber
                ---------                          ---------
```

bridges list	查询全部桥接
bridges options	查询创建桥接选项
bridges start <Node> <Topic>	创建桥接
bridges start <Node> <Topic> <Options>	创建桥接并带选项设置
bridges stop <Node> <Topic>	删除桥接

创建一条 emqttd1 -> emqttd2 节点的桥接，转发传感器主题(Topic)消息到 emqttd2:

```
$ ./bin/emqttd_ctl bridges start emqttd2@127.0.0.1 sensor/#

bridge is started.

$ ./bin/emqttd_ctl bridges list

bridge: emqttd1@127.0.0.1--sensor/#-->emqttd2@127.0.0.1
```

测试 emqttd1–sensor/#–>emqttd2 的桥接:

```
#emqttd2节点上

mosquitto_sub -t sensor/# -p 2883 -d

#emqttd1节点上

mosquitto_pub -t sensor/1/temperature -m "37.5" -d
```

11.10.1 bridge options

查询 bridge 创建选项设置:

```
$ ./bin/emqttd_ctl bridges options

Options:
  qos     = 0 | 1 | 2
  prefix  = string
  suffix  = string
  queue   = integer
Example:
  qos=2,prefix=abc/,suffix=/yxz,queue=1000
```

11.10.2 bridges stop <Node> <Topic>

删除 emqttd1–sensor/#–>emqttd2 的桥接:

```
$ ./bin/emqttd_ctl bridges stop emqttd2@127.0.0.1 sensor/#

bridge is stopped.
```

11.11 vm 命令

vm 命令用于查询 Erlang 虚拟机负载、内存、进程、IO 信息。

vm all	查询 VM 全部信息
vm load	查询 VM 负载
vm memory	查询 VM 内存
vm process	查询 VM Erlang 进程数量
vm io	查询 VM io 最大文件句柄

11.11.1 vm load

查询 VM 负载:

```
$ ./bin/emqttd_ctl vm load

cpu/load1               : 2.21
cpu/load5               : 2.60
cpu/load15              : 2.36
```

11.11.2 vm memory

查询 VM 内存:

```
$ ./bin/emqttd_ctl vm memory

memory/total            : 23967736
memory/processes        : 3594216
memory/processes_used   : 3593112
memory/system           : 20373520
memory/atom             : 512601
memory/atom_used        : 491955
memory/binary           : 51432
memory/code             : 13401565
memory/ets              : 1082848
```

11.11.3 vm process

查询 Erlang 进程数量:

```
$ ./bin/emqttd_ctl vm process

process/limit           : 8192
process/count           : 221
```

11.11.4 vm io

查询 IO 最大句柄数:

```
$ ./bin/emqttd_ctl vm io

io/max_fds               : 2560
io/active_fds            : 1
```

11.12 trace 命令

trace 命令用于追踪某个客户端或 Topic，打印日志信息到文件。

trace list	查询全部开启的追踪
trace client <ClientId> <LogFile>	开启 Client 追踪，日志到文件
trace client <ClientId> off	关闭 Client 追踪
trace topic <Topic> <LogFile>	开启 Topic 追踪，日志到文件
trace topic <Topic> off	关闭 Topic 追踪

11.12.1 trace client <ClientId> <LogFile>

开启 Client 追踪:

```
$ ./bin/emqttd_ctl trace client clientid log/clientid_trace.log

trace client clientid successfully.
```

11.12.2 trace client <ClientId> off

关闭 Client 追踪:

```
$ ./bin/emqttd_ctl trace client clientid off

stop to trace client clientid successfully.
```

11.12.3 trace topic <Topic> <LogFile>

开启 Topic 追踪:

```
$ ./bin/emqttd_ctl trace topic topic log/topic_trace.log

trace topic topic successfully.
```

11.12.4 trace topic <Topic> off

关闭 Topic 追踪:

```
$ ./bin/emqttd_ctl trace topic topic off

stop to trace topic topic successfully.
```

11.12.5 trace list

查询全部开启的追踪:

```
$ ./bin/emqttd_ctl trace list

trace client clientid -> log/clientid_trace.log
trace topic topic -> log/topic_trace.log
```

11.13 listeners

listeners 命令用于查询开启的 TCP 服务监听器:

```
$ ./bin/emqttd_ctl listeners

listener on mqtt:api:127.0.0.1:8080
  acceptors       : 4
  max_clients     : 64
  current_clients : 0
  shutdown_count  : []
listener on mqtt:wss:8084
  acceptors       : 4
  max_clients     : 64
  current_clients : 0
  shutdown_count  : []
listener on mqtt:ssl:8883
  acceptors       : 16
  max_clients     : 1024
  current_clients : 0
  shutdown_count  : []
listener on mqtt:ws:8083
  acceptors       : 4
  max_clients     : 64
  current_clients : 0
  shutdown_count  : []
listener on mqtt:tcp:0.0.0.0:1883
  acceptors       : 16
  max_clients     : 102400
  current_clients : 0
  shutdown_count  : []
listener on mqtt:tcp:127.0.0.1:11883
  acceptors       : 16
  max_clients     : 102400
  current_clients : 0
  shutdown_count  : []
listener on dashboard:http:18083
  acceptors       : 2
  max_clients     : 512
```

(continues on next page)

(续上页)

```
  current_clients : 0
  shutdown_count  : []
```

listener 参数说明:

acceptors	TCP Acceptor 池
max_clients	最大允许连接数
current_clients	当前连接数
shutdown_count	Socket 关闭原因统计

重启监听端口:

```
$ ./bin/emqttd_ctl listeners restart mqtt:tcp 0.0.0.0:1883
Restart mqtt:tcp listener on 0.0.0.0:1883 successfully.
```

停止监听端口:

```
$ ./bin/emqttd_ctl listeners stop mqtt:tcp 0.0.0.0:1883
Stop mqtt:tcp listener on 0.0.0.0:1883 successfully.
```

11.14 mnesia 命令

查询 mnesia 数据库系统状态。

11.15 admins 命令

Dashboard 插件会自动注册 admins 命令，用于创建、删除管理员账号，重置管理员密码。

admins add <Username> <Password>	创建 admin 账号
admins passwd <Username> <Password>	重置 admin 密码
admins del <Username>	删除 admin 账号

11.15.1 admins add

创建 admin 账户:

```
$ ./bin/emqttd_ctl admins add root public
ok
```

11.15.2 admins passwd

重置 admin 账户密码:

```
$ ./bin/emqttd_ctl admins passwd root private
ok
```

11.15.3 admins del

删除 admin 账户:

```
$ ./bin/emqttd_ctl admins del root
ok
```

CHAPTER 12

扩展插件 (Plugins)

EMQ 消息服务器通过模块注册和钩子(Hooks)机制，支持用户开发扩展插件定制服务器认证鉴权与业务功能。

EMQ 2.0 版本官方提供的插件包括:

插件	说明
emq_dashboard	Web 控制台插件(默认加载)
emq_auth_clientid	ClientId 认证插件
emq_auth_username	用户名、密码认证插件
emq_auth_ldap	LDAP 认证/访问控制
emq_auth_http	HTTP 认证/访问控制
emq_auth_mysql	MySQL 认证/访问控制
emq_auth_pgsql	PostgreSQ L认证/访问控制
emq_auth_redis	Redis 认证/访问控制
emq_web_hook	Web Hook 插件
emq_lua_hook	Lua Hook 插件
emq_auth_mongo	MongoDB 认证/访问控制
emq_modules	扩展模块插件
emq_retainer	Retain 消息存储模块
emq_coap	CoAP 协议支持
emq_sn	MQTT-SN 协议支持
emq_stomp	Stomp 协议支持
emq_sockjs	Stomp/SockJS 协议支持
emq_recon	Recon 性能调试
emq_reloader	Reloader 代码热加载插件
emq_plugin_template	插件开发模版

12.1 emq_retainer Retainer 模块插件

2.1-beta 版本将 emq_mod_retainer 模块更名为 emq_retainer 模块，Retainer 模块负责持久化 MQTT Retained 消息。

12.1.1 配置 Retainer 模块

etc/plugins/emq_retainer.conf:

```
## disc: disc_copies, ram: ram_copies
## Notice: retainer's storage_type on each node in a cluster must be the same!
retainer.storage_type = disc

## Max number of retained messages
retainer.max_message_num = 1000000

## Max Payload Size of retained message
retainer.max_payload_size = 64KB

## Expiry interval. Never expired if 0
## h - hour
## m - minute
## s - second
retainer.expiry_interval = 0
```

12.1.2 加载 Retainer 模块

Retainer 模块默认加载。

12.2 emq_auth_clientid - ClientID 认证插件

EMQ 2.0-rc.2 版本将 ClientId 认证模块改为独立插件: https://github.com/emqtt/emq_auth_clientid

12.2.1 ClientID 认证配置

etc/plugins/emq_auth_clientid.conf:

```
##auth.client.$N.clientid = clientid
##auth.client.$N.password = passwd

## Examples
##auth.client.1.clientid = id
##auth.client.1.password = passwd
##auth.client.2.clientid = dev:devid
##auth.client.2.password = passwd2
##auth.client.3.clientid = app:appid
##auth.client.3.password = passwd3
```

12.2.2 加载 ClientId 认证插件

```
./bin/emqttd_ctl plugins load emq_auth_clientid
```

12.3 emq_auth_username - 用户名密码认证插件

EMQ 2.0-rc.2 版本将用户名认证模块改为独立插件: https://github.com/emqtt/emq_auth_username

12.3.1 用户名认证配置

etc/plugins/emq_auth_username.conf:

```
##auth.user.$N.username = admin
##auth.user.$N.password = public

## Examples:
##auth.user.1.username = admin
##auth.user.1.password = public
##auth.user.2.username = feng@emqtt.io
##auth.user.2.password = public
```

两种方式添加用户:

1. 直接在 etc/plugins/emq_auth_username.conf 中明文配置默认用户例如:

   ```
   auth.username.test = public
   ```

2. 通过 './bin/emqttd_ctl' 管理命令行添加用户:

   ```
   $ ./bin/emqttd_ctl users add <Username> <Password>
   ```

12.3.2 加载用户名认证插件

```
./bin/emqttd_ctl plugins load emq_auth_username
```

12.4 emq_plugin_template: 插件开发模版

EMQ 插件实际是一个普通的 Erlang 应用，插件配置文件: 'etc/${PluginName}.conflconfig"。

emq_plugin_template 是模版插件，编译发布在 lib/emq_plugin_template-2.0 目录，配置文件: etc/plugins/emq_plugin_templat.config

12.4.1 加载、卸载插件

管理命令行 './bin/emqttd_ctl' 加载卸载插件。

加载插件:

```
./bin/emqttd_ctl plugins load <PluginName>
```

卸载插件:

```
./bin/emqttd_ctl plugins unload <PluginName>
```

查询插件:

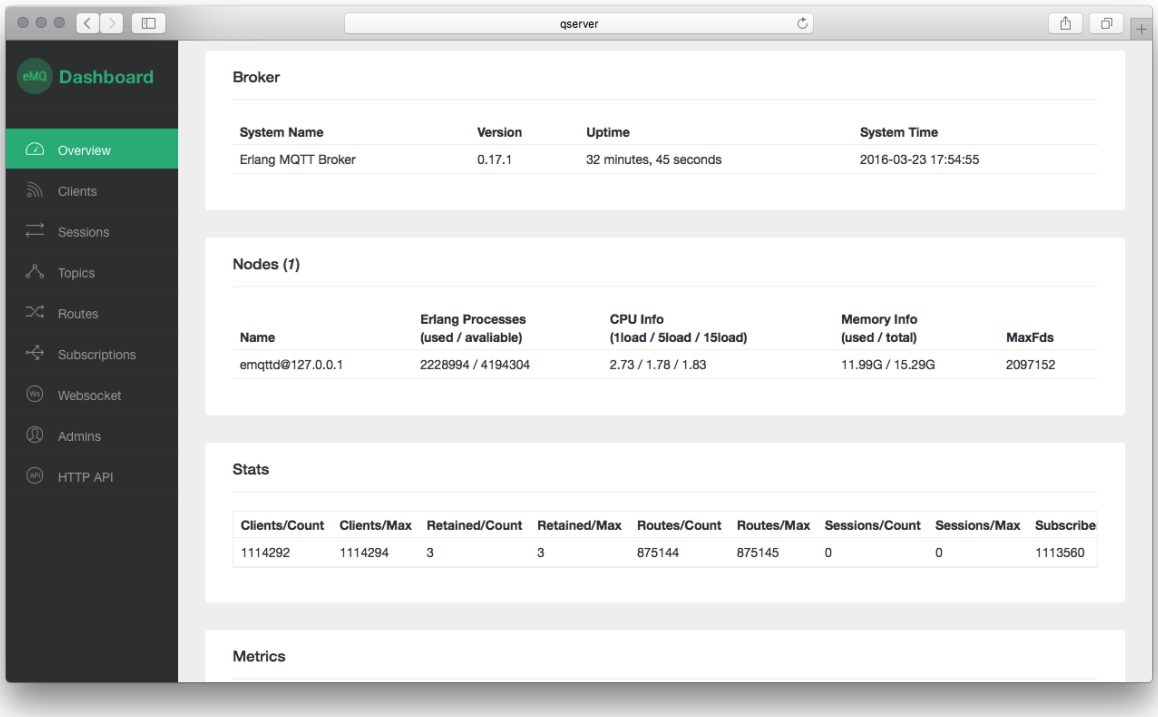
```
./bin/emqttd_ctl plugins list
```

12.5 emq_dashboard: Dashboard 插件

EMQ 消息服务器的 Web 管理控制台。插件项目地址: https://github.com/emqtt/emqttd_dashboard

EMQ 消息服务器默认加载 Dashboard 插件。URL 地址: http://localhost:18083 ，缺省用户名/密码: admin/public。

Dashboard 插件可查询 EMQ 消息服务器基本信息、统计数据、度量数据，查询系统客户端(Client)、会话(Session)、主题(Topic)、订阅(Subscription)。



12.5.1 Dashboard 插件设置

etc/plugins/emq_dashboard.conf:

```
## HTTP Listener
dashboard.listener.http = 18083
```

(continues on next page)

(续上页)

```
dashboard.listener.http.acceptors = 2
dashboard.listener.http.max_clients = 512

## HTTPS Listener
## dashboard.listener.https = 18084
## dashboard.listener.https.acceptors = 2
## dashboard.listener.https.max_clients = 512
## dashboard.listener.https.handshake_timeout = 15s
## dashboard.listener.https.certfile = etc/certs/cert.pem
## dashboard.listener.https.keyfile = etc/certs/key.pem
## dashboard.listener.https.cacertfile = etc/certs/cacert.pem
## dashboard.listener.https.verify = verify_peer
## dashboard.listener.https.fail_if_no_peer_cert = true
```

12.6 emq_auth_ldap: LDAP 认证插件

LDAP 认证插件: https://github.com/emqtt/emq_auth_ldap

注解: 2.0-beta1 版本支持

12.6.1 LDAP 认证插件配置

etc/plugins/emq_auth_ldap.conf:

```
auth.ldap.servers = 127.0.0.1

auth.ldap.port = 389

auth.ldap.timeout = 30

auth.ldap.user_dn = uid=%u,ou=People,dc=example,dc=com

auth.ldap.ssl = false
```

12.6.2 LDAP 认证插件加载

./bin/emqttd_ctl plugins load emq_auth_ldap

12.7 emq_auth_http: HTTP 认证/访问控制插件

HTTP 认证/访问控制插件: https://github.com/emqtt/emq_auth_http

注解: 1.1版本支持

12.7.1 HTTP 认证插件配置

etc/plugins/emq_auth_http.conf:

```
## Variables: %u = username, %c = clientid, %a = ipaddress, %P = password, %t = topic

auth.http.auth_req = http://127.0.0.1:8080/mqtt/auth
auth.http.auth_req.method = post
auth.http.auth_req.params = clientid=%c,username=%u,password=%P

auth.http.super_req = http://127.0.0.1:8080/mqtt/superuser
auth.http.super_req.method = post
auth.http.super_req.params = clientid=%c,username=%u

## 'access' parameter: sub = 1, pub = 2
auth.http.acl_req = http://127.0.0.1:8080/mqtt/acl
auth.http.acl_req.method = get
auth.http.acl_req.params = access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t
```

12.7.2 HTTP 认证/鉴权 API

认证/ACL 成功，API 返回200

认证/ACL 失败，API 返回4xx

12.7.3 加载 HTTP 认证插件

./bin/emqttd_ctl plugins load emq_auth_http

12.8 emq_auth_mysql: MySQL 认证/访问控制插件

MySQL 认证/访问控制插件，基于 MySQL 库表认证鉴权: https://github.com/emqtt/emq-auth-mysql

12.8.1 MQTT 用户表

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(35) DEFAULT NULL,
  `is_superuser` tinyint(1) DEFAULT 0,
  `created` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

注解: MySQL 插件可使用系统自有的用户表，通过 'authquery' 配置查询语句。

12.8.2 MQTT 访问控制表

```
CREATE TABLE `mqtt_acl` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `allow` int(1) DEFAULT NULL COMMENT '0: deny, 1: allow',
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `mqtt_acl` (`id`, `allow`, `ipaddr`, `username`, `clientid`, `access`,␣
↪`topic`)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

12.8.3 配置 MySQL 认证鉴权插件

etc/plugins/emq_auth_mysql.conf:

```
## Mysql Server
auth.mysql.server = 127.0.0.1:3306

## Mysql Pool Size
auth.mysql.pool = 8

## Mysql Username
## auth.mysql.username =

## Mysql Password
## auth.mysql.password =

## Mysql Database
auth.mysql.database = mqtt

## Variables: %u = username, %c = clientid

## Authentication Query: select password only
auth.mysql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.mysql.password_hash = sha256

## %% Superuser Query
auth.mysql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1

## ACL Query Command
auth.mysql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid =␣
↪%c'
```

(continues on next page)

(续上页)

12.8.4 加载 MySQL 认证鉴权插件

./bin/emqttd_ctl plugins load emq_auth_mysql

12.9 emq_auth_pgsql: Postgre 认证/访问控制插件

Postgre 认证/访问控制插件，基于 PostgreSQL 库表认证鉴权: https://github.com/emqtt/emqttd_plugin_pgsql

12.9.1 Postgre MQTT 用户表

```
CREATE TABLE mqtt_user (
  id SERIAL primary key,
  is_superuser boolean,
  username character varying(100),
  password character varying(100),
  salt character varying(40)
);
```

12.9.2 Postgre MQTT 访问控制表

```
CREATE TABLE mqtt_acl (
  id SERIAL primary key,
  allow integer,
  ipaddr character varying(60),
  username character varying(100),
  clientid character varying(100),
  access  integer,
  topic character varying(100)
);

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

12.9.3 配置 Postgre 认证鉴权插件

etc/plugins/emq_auth_pgsql.conf:

```
## Postgre Server
auth.pgsql.server = 127.0.0.1:5432

auth.pgsql.pool = 8

auth.pgsql.username = root

#auth.pgsql.password =

auth.pgsql.database = mqtt

auth.pgsql.encoding = utf8

auth.pgsql.ssl = false

## Variables: %u = username, %c = clientid, %a = ipaddress

## Authentication Query: select password only
auth.pgsql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.pgsql.password_hash = sha256

## sha256 with salt prefix
## auth.pgsql.password_hash = salt sha256

## sha256 with salt suffix
## auth.pgsql.password_hash = sha256 salt

## Superuser Query
auth.pgsql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1

## ACL Query. Comment this query, the acl will be disabled.
auth.pgsql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'
```

12.9.4 加载 Postgre 认证鉴权插件

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

12.10 emq_auth_redis: Redis 认证/访问控制插件

基于 Redis 认证/访问控制: https://github.com/emqtt/emqttd_plugin_redis

12.10.1 配置 Redis 认证鉴权插件

etc/plugins/emq_auth_redis.conf:

```
## Redis Server
auth.redis.server = 127.0.0.1:6379

## Redis Pool Size
auth.redis.pool = 8

## Redis Database
auth.redis.database = 0

## Redis Password
## auth.redis.password =

## Variables: %u = username, %c = clientid

## Authentication Query Command
auth.redis.auth_cmd = HGET mqtt_user:%u password

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.redis.password_hash = sha256

## Superuser Query Command
auth.redis.super_cmd = HGET mqtt_user:%u is_superuser

## ACL Query Command
auth.redis.acl_cmd = HGETALL mqtt_acl:%u
```

12.10.2 Redis 用户 Hash

默认基于用户 Hash 认证:

```
HSET mqtt_user:<username> is_superuser 1
HSET mqtt_user:<username> password "passwd"
```

12.10.3 Redis ACL 规则 Hash

默认采用 Hash 存储 ACL 规则:

```
HSET mqtt_acl:<username> topic1 1
HSET mqtt_acl:<username> topic2 2
HSET mqtt_acl:<username> topic3 3
```

注解: 1: subscribe, 2: publish, 3: pubsub

12.10.4 Redis 订阅 Hash

插件还支持 Redis 中创建 MQTT 订阅。当 MQTT 客户端连接成功，会自动从 Redis 加载订阅:

```
HSET mqtt_sub:<username> topic1 0
HSET mqtt_sub:<username> topic2 1
HSET mqtt_sub:<username> topic3 2
```

警告： 2.0-rc.2 版本已将订阅加载迁移至 EMQX 产品的emqx_backend_redis插件。

12.10.5 加载 Redis 认证鉴权插件

```
./bin/emqttd_ctl plugins load emq_auth_redis
```

12.11 emq_auth_mongo: MongoDB 认证/访问控制插件

基于 MongoDB 认证/访问控制: https://github.com/emqtt/emqttd_plugin_mongo

12.11.1 配置 MongoDB 认证鉴权插件

etc/plugins/emq_auth_mongo.conf:

```
## Mongo Server
auth.mongo.server = 127.0.0.1:27017

## Mongo Pool Size
auth.mongo.pool = 8

## Mongo User
## auth.mongo.user =

## Mongo Password
## auth.mongo.password =

## Mongo Database
auth.mongo.database = mqtt

## auth_query
auth.mongo.auth_query.collection = mqtt_user

auth.mongo.auth_query.password_field = password

auth.mongo.auth_query.password_hash = sha256

auth.mongo.auth_query.selector = username=%u

## super_query
auth.mongo.super_query.collection = mqtt_user

auth.mongo.super_query.super_field = is_superuser

auth.mongo.super_query.selector = username=%u

## acl_query
auth.mongo.acl_query.collection = mqtt_user

auth.mongo.acl_query.selector = username=%u
```

12.11.2 MongoDB 数据库

```
use mqtt
db.createCollection("mqtt_user")
db.createCollection("mqtt_acl")
db.mqtt_user.ensureIndex({"username":1})
```

注解: 数据库、集合名称可自定义

12.11.3 MongoDB 用户集合(User Collection)

```
{
    username: "user",
    password: "password hash",
    is_superuser: boolean (true, false),
    created: "datetime"
}
```

示例:

```
db.mqtt_user.insert({username: "test", password: "password hash", is_superuser: false}
↪)
db.mqtt_user:insert({username: "root", is_superuser: true})
```

12.11.4 MongoDB ACL 集合(ACL Collection)

```
{
    username: "username",
    clientid: "clientid",
    publish: ["topic1", "topic2", ...],
    subscribe: ["subtop1", "subtop2", ...],
    pubsub: ["topic/#", "topic1", ...]
}
```

示例:

```
db.mqtt_acl.insert({username: "test", publish: ["t/1", "t/2"], subscribe: ["user/%u",
↪"client/%c"]})
db.mqtt_acl.insert({username: "admin", pubsub: ["#"]})
```

12.11.5 加载 Mognodb 认证插件

```
./bin/emqttd_ctl plugins load emq_auth_mongo
```

12.12 emq_modules 扩展模块插件

2.1 版本将全部扩展模块项目(emq_mod_presence, emq_mod_subscription, emq_mod_rewrite)合并为一个 emq_modules 项目。

12.12.1 配置 Modules 插件

```
##--------------------------------------------------------------------
## Presence Module
##--------------------------------------------------------------------

## Enable Presence, Values: on | off
module.presence = on

module.presence.qos = 1

##--------------------------------------------------------------------
## Subscription Module
##--------------------------------------------------------------------

## Enable Subscription, Values: on | off
module.subscription = on

## Subscribe the Topics automatically when client connected
module.subscription.1.topic = $client/%c
## Qos of the subscription: 0 | 1 | 2
module.subscription.1.qos = 1

## module.subscription.2.topic = $user/%u
## module.subscription.2.qos = 1

##--------------------------------------------------------------------
## Rewrite Module
##--------------------------------------------------------------------

## Enable Rewrite, Values: on | off
module.rewrite = off

## {rewrite, Topic, Re, Dest}
## module.rewrite.rule.1 = x/# ^x/y/(.+)$ z/y/$1
## module.rewrite.rule.2 = y/+/z/# ^y/(.+)/z/(.+)$ y/z/$2
```

12.12.2 加载 Modules 插件

Modules 插件默认加载。

12.13 emq_mod_presence Presence 模块插件

2.0-rc.3 版本将 Presence 模块改为独立插件，Presence 模块会向 $SYS 主题(Topic)发布客户端上下线消息。

警告： 2.1 版本该插件已并入 emq_modules 项目

12.13.1 配置 Presence 模块

etc/plugins/emq_mod_presence.conf:

```
## Enable presence module
## Values: on | off
module.presence = on

module.presence.qos = 0
```

12.13.2 加载 Presence 模块

Presence 模块默认加载。

12.14 emq_mod_subscription 自动订阅模块插件

2.0-rc.3 版本将 Subscription 模块改为独立插件，Subscription 扩展模块支持客户端上线时，自动订阅或恢复订阅某些主题(Topic)。

警告： 2.1 版本该插件已并入 emq_modules 项目

12.14.1 配置 Subscription 模块

etc/plugins/emq_mod_subscription.conf:

警告： 2.1 版本该插件已并入 emq_modules 项目

```
## Subscribe the Topics automatically when client connected
module.subscription.1.topic = $client/%c
## Qos of the subscription: 0 | 1 | 2
module.subscription.1.qos = 1

##module.subscription.2.topic = $user/%u
##module.subscription.2.qos = 1

## Load static subscriptions from backend storage
## Values: on | off
module.subscription.backend = on
```

12.14.2 加载 Subscription 模块

Subscription 模块默认加载。

12.15 emq_mod_rewrite 主题重写插件

2.0-rc.2 版本将 rewrite 模块改为独立插件，rewrite 插件支持重写发布订阅的主题(Topic)。

警告: 2.1版本该插件已并入 emq_modules 项目

12.15.1 配置 Rewrite 插件

etc/plugins/emq_mod_rewrite.conf:

```
[
  {emq_mod_rewrite, [
    {rules, [
      %% {rewrite, Topic, Re, Dest}

      %% Example: x/y/ -> z/y/
      %% {rewrite, "x/#", "^x/y/(.+)$", "z/y/$1"},

      %% {rewrite, "y/+/z/#", "^y/(.+)/z/(.+)$", "y/z/$2"}
    ]}
  ]}
].
```

12.15.2 加载 Rewrite 插件

```
./bin/emqttd_ctl plugins load emq_mod_rewrite
```

12.16 emq_coap: CoAP 协议插件

CoAP 协议插件，支持 RFC 7252 规范。

12.16.1 配置 CoAP 协议插件

```
coap.server = 5683

coap.prefix.mqtt = mqtt

coap.handler.mqtt = emq_coap_gateway
```

12.16.2 加载 CoAP 协议插件

```
./bin/emqttd_ctl plugins load emq_coap
```

12.16.3 libcoap 客户端

```
yum install libcoap

% coap client publish message
coap-client -m post -e "qos=0&retain=0&message=payload&topic=hello" coap://localhost/
↪mqtt
```

(continues on next page)

(续上页)

12.17 emq_sn: MQTT-SN 协议插件

MQTT-SN 协议插件，支持 MQTT-SN 网关模式。

12.17.1 配置 MQTT-SN 协议插件

注解: 默认 MQTT-SN 协议 UDP 端口: 1884

etc/plugins/emq_sn.conf:

```
mqtt.sn.port = 1884
```

12.17.2 加载 MQTT-SN 协议插件

```
./bin/emqttd_ctl plugins load emq_sn
```

12.18 emq_stomp: Stomp 协议插件

Stomp 协议插件。支持 STOMP 1.0/1.1/1.2 协议客户端连接 EMQ，发布订阅 MQTT 消息。

12.18.1 配置插件

注解: Stomp 协议端口: 61613

etc/plugins/emq_stomp.conf:

```
stomp.default_user.login = guest

stomp.default_user.passcode = guest

stomp.allow_anonymous = true

stomp.frame.max_headers = 10

stomp.frame.max_header_length = 1024

stomp.frame.max_body_length = 8192

stomp.listener = 61613

stomp.listener.acceptors = 4
```

(continues on next page)

(续上页)

```
stomp.listener.max_clients = 512
```

12.18.2 加载 Stomp 插件

```
./bin/emqttd_ctl plugins load emq_stomp
```

12.19 emq_sockjs: Stomp/Sockjs 插件

警告: 2.0 版本不再维护 SockJS 插件

12.19.1 配置 SockJS 插件

etc/plugins/emq_sockjs.config:

注解: 缺省端口: 61616

```
[
  {emq_sockjs, [

    {sockjs, []},

    {cowboy_listener, {stomp_sockjs, 61616, 4}},

    %% TODO: unused...
    {stomp, [
      {frame, [
        {max_headers,       10},
        {max_header_length, 1024},
        {max_body_length,   8192}
      ]}
    ]}
  ]}
].
```

12.19.2 加载 SockJS 插件

```
./bin/emqttd_ctl plugins load emq_sockjs
```

12.19.3 插件演示页面

http://localhost:61616/index.html

12.20 emq_recon: Recon 性能调试插件

emq_recon 插件集成 recon 性能调测库，'./bin/emqttd_ctl' 命令行注册 recon 命令。

12.20.1 配置 Recon 插件

etc/plugins/emq_recon.conf:

```
%% Garbage Collection: 10 minutes
recon.gc_interval = 600
```

12.20.2 加载 Recon 插件

```
./bin/emqttd_ctl plugins load emq_recon
```

12.20.3 recon 插件命令

```
./bin/emqttd_ctl recon

recon memory                #recon_alloc:memory/2
recon allocated             #recon_alloc:memory(allocated_types, current|max)
recon bin_leak              #recon:bin_leak(100)
recon node_stats            #recon:node_stats(10, 1000)
recon remote_load Mod       #recon:remote_load(Mod)
```

12.21 emq_reloader: 代码热加载插件

用于开发调试的代码热升级插件。加载该插件后，EMQ 会自动热升级更新代码。

注解: 产品部署环境不建议使用该插件

12.21.1 配置 Reloader 插件

etc/plugins/emq_reloader.conf:

```
reloader.interval = 60

reloader.logfile = log/reloader.log
```

12.21.2 加载 Reloader 插件

```
./bin/emqttd_ctl plugins load emq_reloader
```

12.21.3 Reloader 插件命令

```
./bin/emqttd_ctl reload

reload <Module>              # Reload a Module
```

12.22 EMQ 2.0 插件开发

12.22.1 创建插件项目

参考 emq_plugin_template 插件模版创建新的插件项目。

12.22.2 注册认证/访问控制模块

认证演示模块 - emq_auth_demo.erl

```
-module(emq_auth_demo).

-behaviour(emqttd_auth_mod).

-include_lib("emqttd/include/emqttd.hrl").

-export([init/1, check/3, description/0]).

init(Opts) -> {ok, Opts}.

check(#mqtt_client{client_id = ClientId, username = Username}, Password, _Opts) ->
    io:format("Auth Demo: clientId=~p, username=~p, password=~p~n",
              [ClientId, Username, Password]),
    ok.

description() -> "Demo Auth Module".
```

访问控制演示模块 - emqttd_acl_demo.erl

```
-module(emq_acl_demo).

-include_lib("emqttd/include/emqttd.hrl").

%% ACL callbacks
-export([init/1, check_acl/2, reload_acl/1, description/0]).

init(Opts) ->
    {ok, Opts}.

check_acl({Client, PubSub, Topic}, Opts) ->
    io:format("ACL Demo: ~p ~p ~p~n", [Client, PubSub, Topic]),
    allow.

reload_acl(_Opts) ->
    ok.

description() -> "ACL Module Demo".
```

注册认证、访问控制模块 - emq_plugin_template_app.erl

```
ok = emqttd_access_control:register_mod(auth, emq_auth_demo, []),
ok = emqttd_access_control:register_mod(acl, emq_acl_demo, []),
```

12.22.3 注册扩展钩子(Hooks)

通过钩子(Hook)处理客户端上下线、主题订阅、消息收发。

emq_plugin_template.erl:

```
%% Called when the plugin application start
load(Env) ->
    emqttd:hook('client.connected', fun ?MODULE:on_client_connected/3, [Env]),
    emqttd:hook('client.disconnected', fun ?MODULE:on_client_disconnected/3, [Env]),
    emqttd:hook('client.subscribe', fun ?MODULE:on_client_subscribe/4, [Env]),
    emqttd:hook('client.unsubscribe', fun ?MODULE:on_client_unsubscribe/4, [Env]),
    emqttd:hook('session.subscribed', fun ?MODULE:on_session_subscribed/4, [Env]),
    emqttd:hook('session.unsubscribed', fun ?MODULE:on_session_unsubscribe/4, [Env]),
    emqttd:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqttd:hook('message.delivered', fun ?MODULE:on_message_delivered/4, [Env]),
    emqttd:hook('message.acked', fun ?MODULE:on_message_acked/4, [Env]).
```

扩展钩子(Hook):

钩子	说明
client.connected	客户端上线
client.subscribe	客户端订阅主题前
session.subscribed	客户端订阅主题后
client.unsubscribe	客户端取消订阅主题
session.unsubscribed	客户端取消订阅主题后
message.publish	MQTT 消息发布
message.delivered	MQTT 消息送达
message.acked	MQTT 消息回执
client.disconnected	客户端连接断开

12.22.4 注册扩展命令行

扩展命令行演示模块 - emq_cli_demo.erl

```
-module(emq_cli_demo).

-include_lib("emqttd/include/emqttd_cli.hrl").

-export([cmd/1]).

cmd(["arg1", "arg2"]) ->
    ?PRINT_MSG("ok");

cmd(_) ->
    ?USAGE([{"cmd arg1 arg2",  "cmd demo"}]).
```

注册命令行模块 - emq_plugin_template_app.erl

```
emqttd_ctl:register_cmd(cmd, {emq_cli_demo, cmd}, []).
```

插件加载后，'./bin/emqttd_ctl'新增命令行:

```
./bin/emqttd_ctl cmd arg1 arg2
```

12.22.5 插件配置文件

插件自带配置文件放置在 etc/${plugin_name}.conflconfig，EMQ 支持两种插件配置格式:

1. ${plugin_name}.config，Erlang 原生配置文件格式:

```
[
  {plugin_name, [
    {key, value}
  ]}
].
```

2. ${plugin_name}.conf, sysctl 的 *k* = *v* 通用格式:

```
plugin_name.key = value
```

注解: *k* = *v* 格式配置需要插件开发者创建 priv/plugin_name.schema 映射文件。

12.22.6 编译发布插件

1. clone emq-relx 项目:

```
git clone https://github.com/emqtt/emq-relx.git
```

2. Makefile 增加 *DEPS*:

```
DEPS += plugin_name
dep_plugin_name = git url_of_plugin
```

3. relx.config 中 release 段落添加:

```
{plugin_name, load},
```

CHAPTER 13

管理监控API (REST API)

用户可以通过 REST API 查询 MQTT 客户端连接(Clients)、会话(Sessions)、订阅(Subscriptions)和路由(Routes)信息，还可以检索和监控服务器的性能指标和统计数据。

13.1 URL 地址

REST API 访问 URL 地址:

```
http(s)://host:8080/api/v2/
```

13.2 Basic 认证

REST API 采用 HTTP Basic 认证(Authentication):

```
curl -v --basic -u <user>:<passwd> -k http://localhost:8080/api/v2/nodes/emq@127.0.0.
↪1/clients
```

13.3 集群与节点

13.3.1 获取全部节点的基本信息

API 定义:

```
GET api/v2/management/nodes
```

请求示例:

```
GET api/v2/management/nodes
```

返回数据:

```
{
    "code": 0,
    "result": [
            {
                    "name": "emq@127.0.0.1",
                    "version": "2.3.10",
                    "sysdescr": "Erlang MQTT Broker",
                    "uptime": "3 minutes, 32 seconds",
                    "datetime": "2018-06-29 09:03:52",
                    "otp_release": "R20/9.3.3",
                    "node_status": "Running"
            }
    ]
}
```

13.3.2 获取指定节点的基本信息

API 定义:

```
GET api/v2/management/nodes/{node_name}
```

请求示例:

```
GET api/v2/management/nodes/emq@127.0.0.1
```

返回数据:

```
{
    "code": 0,
    "result": {
            "version": "2.3.10",
            "sysdescr": "Erlang MQTT Broker",
            "uptime": "5 minutes, 12 seconds",
            "datetime": "2018-06-29 09:05:32",
            "otp_release": "R20/9.3.3",
            "node_status": "Running"
    }
}
```

13.3.3 获取全部节点的监控数据

API 定义:

```
GET api/v2/monitoring/nodes
```

请求示例:

```
GET api/v2/monitoring/nodes
```

返回数据:

```
{
    "code": 0,
    "result": [
            {
                    "name": "emq@127.0.0.1",
                    "otp_release": "R20/9.3.3",
                    "memory_total": "72.94M",
                    "memory_used": "50.55M",
                    "process_available": 262144,
                    "process_used": 324,
                    "max_fds": 7168,
                    "clients": 0,
                    "node_status": "Running",
                    "load1": "1.65",
                    "load5": "1.93",
                    "load15": "2.01"
            }
    ]
}
```


13.3.4 获取指定节点的监控数据

API 定义:

```
GET api/v2/monitoring/nodes/{node_name}
```

请求示例:

```
GET api/v2/monitoring/nodes/emq@127.0.0.1
```

返回数据:


```
{
    "code": 0,
    "result": {
            "name": "emq@127.0.0.1",
            "otp_release": "R20/9.3.3",
            "memory_total": "73.69M",
            "memory_used": "50.12M",
            "process_available": 262144,
            "process_used": 324,
            "max_fds": 7168,
            "clients": 0,
            "node_status": "Running",
            "load1": "1.88",
            "load5": "1.99",
            "load15": "2.02"
    }
}
```

13.4 客户端连接(Clients)

13.4.1 获取指定节点的客户端连接列表

API 定义:

```
GET api/v2/nodes/{node_name}/clients
```

请求参数:

```
curr_page={page_no}&page_size={page_size}
```

请求示例:

```
api/v2/nodes/emq@127.0.0.1/clients?curr_page=1&page_size=20
```

返回数据:

```
{
    "code": 0,
    "result": {
            "current_page": 1,
            "page_size": 20,
            "total_num": 1,
            "total_page": 1,
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "username": "undefined",
                            "ipaddress": "127.0.0.1",
                            "port": 58459,
                            "clean_sess": true,
                            "proto_ver": 4,
                            "keepalive": 60,
                            "connected_at": "2018-06-29 09:15:25"
                    }
            ]
    }
}
```

13.4.2 获取节点指定客户端连接的信息

API 定义:

```
GET api/v2/nodes/{node_name}/clients/{clientid}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/clients/mqttjs_722b4d845f
```

返回数据:

```
{
    "code": 0,
```

(continues on next page)

(续上页)


```
    "result": {
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "username": "undefined",
                            "ipaddress": "127.0.0.1",
                            "port": 58459,
                            "clean_sess": true,
                            "proto_ver": 4,
                            "keepalive": 60,
                            "connected_at": "2018-06-29 09:15:25"
                    }
            ]
    }
}
```


13.4.3 获取集群内指定客户端的信息

API 定义:

```
GET api/v2/clients/{clientid}
```

请求示例:

```
GET api/v2/clients/mqttjs_722b4d845f
```

返回数据:


```
{
    "code": 0,
    "result": {
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "username": "undefined",
                            "ipaddress": "127.0.0.1",
                            "port": 58459,
                            "clean_sess": true,
                            "proto_ver": 4,
                            "keepalive": 60,
                            "connected_at": "2018-06-29 09:15:25"
                    }
            ]
    }
}
```


13.4.4 断开集群内指定客户端连接

API定义:

```
DELETE api/v2/clients/{clientid}
```

请求示例:

```
DELETE api/v2/clients/mqttjs_722b4d845f
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.4.5 清除集群内指定客户端的ACL缓存

API定义:

```
PUT api/v2/clients/{clientid}/clean_acl_cache
```

请求参数:

```
{
    "topic": "test"
}
```

请求示例:

```
PUT api/v2/clients/mqttjs_722b4d845f/clean_acl_cache

请求的 json 参数:
{
    "topic": "test"
}
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.5 会话(Sessions)

13.5.1 获取指定节点的会话列表

API 定义:

```
GET api/v2/nodes/{node_name}/sessions
```

请求参数:

```
curr_page={page_no}&page_size={page_size}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/sessions?curr_page=1&page_size=20
```

返回数据:


```
{
    "code": 0,
    "result": {
            "current_page": 1,
            "page_size": 20,
            "total_num": 1,
            "total_page": 1,
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "clean_sess": true,
                            "subscriptions": 0,
                            "max_inflight": 32,
                            "inflight_len": 0,
                            "mqueue_len": 0,
                            "mqueue_dropped": 0,
                            "awaiting_rel_len": 0,
                            "deliver_msg": 0,
                            "enqueue_msg": 0,
                            "created_at": "2018-06-29 10:05:13"
                    }
            ]
    }
}
```


13.5.2 获取节点上指定客户端的会话信息

API 定义:

```
GET api/v2/nodes/{node_name}/sessions/{clientid}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/sessions/mqttjs_722b4d845f
```

返回数据:


```
{
    "code": 0,
    "result": {
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "clean_sess": true,
                            "subscriptions": 0,
                            "max_inflight": 32,
                            "inflight_len": 0,
                            "mqueue_len": 0,
                            "mqueue_dropped": 0,
                            "awaiting_rel_len": 0,
                            "deliver_msg": 0,
```


(continues on next page)

(续上页)

```
                            "enqueue_msg": 0,
                            "created_at": "2018-06-29 10:05:13"
                    }
            ]
    }
}
```

13.5.3 获取集群内指定客户端的会话信息

API 定义:

```
GET api/v2/sessions/{clientid}
```

请求示例:

```
GET api/v2/sessions/mqttjs_722b4d845f
```

返回数据:

13.6 订阅(Subscriptions)

13.6.1 获取某个节点上的订阅列表

API 定义:

```
GET api/v2/nodes/{node_name}/subscriptions
```

请求参数:

```
curr_page={page_no}&page_size={page_size}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/subscriptions?curr_page=1&page_size=20
```

返回数据:

```
{
    "code": 0,
    "result": {
            "current_page": 1,
            "page_size": 20,
            "total_num": 1,
            "total_page": 1,
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "topic": "/World",
                            "qos": 0
                    }
            ]
```

(continues on next page)

(续上页)

```
    }
}
```

13.6.2 获取节点上指定客户端的订阅信息

API 定义:

```
GET api/v2/nodes/{node_name}/subscriptions/{clientid}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/subscriptions/mqttjs_722b4d845f
```

返回数据:

```
{
    "code": 0,
    "result": {
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "topic": "/World",
                            "qos": 0
                    }
            ]
    }
}
```

13.6.3 获取集群内指定客户端的订阅信息

API 定义:

```
GET api/v2/subscriptions/{clientid}
```

请求示例:

```
GET api/v2/subscriptions/mqttjs_722b4d845f
```

返回数据:

```
{
    "code": 0,
    "result": {
            "objects": [
                    {
                            "client_id": "mqttjs_722b4d845f",
                            "topic": "/World",
                            "qos": 0
                    }
            ]
    }
}
```

13.7 路由(Routes)

13.7.1 获取集群路由表

API 定义:

```
GET api/v2/routes
```

请求参数:

```
curr_page={page_no}&page_size={page_size}
```

请求示例:

```
GET api/v2/routes?curr_page=1&page_size=20
```

返回数据:

```
{
    "code": 0,
    "result": {
            "current_page": 1,
            "page_size": 20,
            "total_num": 1,
            "total_page": 1,
            "objects": [
                    {
                            "topic": "/World",
                            "node": "emq@127.0.0.1"
                    }
            ]
    }
}
```

13.7.2 获取集群内指定主题的路由信息

API 定义:

```
GET api/v2/routes/{topic}
```

请求示例:

```
GET api/v2/routes//World
```

返回数据:

```
{
    "code": 0,
    "result": {
            "objects": [
                    {
                            "topic": "/World",
                            "node": "emq@127.0.0.1"
                    }
```

(continues on next page)

(续上页)

```
            ]
    }
}
```

13.8 发布/订阅

13.8.1 发布消息

API 定义:

```
POST api/v2/mqtt/publish
```

请求参数:

```
{
    "topic" : "/World",
    "payload": "hello",
    "qos": 0,
    "retain" : false,
    "client_id": "mqttjs_722b4d845f"
}
```

注解: topic 参数必填，其他参数可选。payload 默认值空字符串，qos 默认为 0，retain 默认为 false，client_id 默认为 'http'。

请求示例:

```
POST api/v2/mqtt/publish

请求参数 json:
{
          "topic" : "/World",
    "payload": "hello",
          "qos": 0,
          "retain" : false,
      "client_id": "mqttjs_722b4d845f"
}
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.8.2 创建订阅

API 定义:

```
POST api/v2/mqtt/subscribe
```

请求参数:

```
{
    "topic"     : "/World",
    "qos"       : 0,
    "client_id": "mqttjs_722b4d845f"
}
```

请求示例:

```
POST api/v2/mqtt/subscribe
请求参数 json:
{
          "topic" : "/World",
          "qos": 0,
      "client_id": "mqttjs_722b4d845f"
}
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.8.3 取消订阅

API 定义:

```
POST api/v2/mqtt/unsubscribe
```

请求参数:

```
{
          "topic" : "/World",
      "client_id": "mqttjs_722b4d845f"
}
```

请求示例:

```
POST api/v2/mqtt/unsubscribe
请求参数 json:
{
          "topic" : "/World",
      "client_id": "mqttjs_722b4d845f"
}
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.9 插件(Plugins)

13.9.1 获取节点的插件列表

API 定义:

```
GET api/v2/nodes/{node_name}/plugins
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/plugins
```

返回数据:

```
{
    "code": 0,
    "result": [
            {
                    "name": "emq_auth_clientid",
                    "version": "2.3.10",
                    "description": "Authentication with ClientId/Password",
                    "active": false
            },
            {
                    "name": "emq_auth_http",
                    "version": "2.3.10",
                    "description": "Authentication/ACL with HTTP API",
                    "active": false
            },
            {
                    "name": "emq_auth_jwt",
                    "version": "2.3.10",
                    "description": "Authentication with JWT",
                    "active": false
            },
            {
                    "name": "emq_auth_ldap",
                    "version": "2.3.10",
                    "description": "Authentication/ACL with LDAP",
                    "active": false
            },
            {
                    "name": "emq_auth_mongo",
                    "version": "2.3.10",
                    "description": "Authentication/ACL with MongoDB",
                    "active": false
            },
            {
                    "name": "emq_auth_mysql",
                    "version": "2.3.10",
                    "description": "Authentication/ACL with MySQL",
                    "active": false
            },
            {
                    "name": "emq_auth_pgsql",
                    "version": "2.3.10",
```

(continues on next page)

(续上页)


```
                "description": "Authentication/ACL with PostgreSQL",
                "active": false
        },
        {
                "name": "emq_auth_redis",
                "version": "2.3.10",
                "description": "Authentication/ACL with Redis",
                "active": false
        },
        {
                "name": "emq_auth_username",
                "version": "2.3.10",
                "description": "Authentication with Username/Password",
                "active": false
        },
        {
                "name": "emq_coap",
                "version": "2.3.10",
                "description": "CoAP Gateway",
                "active": false
        },
        {
                "name": "emq_dashboard",
                "version": "2.3.10",
                "description": "EMQ Web Dashboard",
                "active": true
        },
        {
                "name": "emq_lua_hook",
                "version": "2.3.10",
                "description": "EMQ Hooks in lua",
                "active": false
        },
        {
                "name": "emq_modules",
                "version": "2.3.10",
                "description": "EMQ Modules",
                "active": true
        },
        {
                "name": "emq_plugin_template",
                "version": "2.3.10",
                "description": "EMQ Plugin Template",
                "active": false
        },
        {
                "name": "emq_recon",
                "version": "2.3.10",
                "description": "Recon Plugin",
                "active": true
        },
        {
                "name": "emq_reloader",
                "version": "2.3.10",
                "description": "Reloader Plugin",
                "active": false
```


(continues on next page)

(续上页)

```
            },
            {
                    "name": "emq_retainer",
                    "version": "2.3.10",
                    "description": "EMQ Retainer",
                    "active": true
            },
            {
                    "name": "emq_sn",
                    "version": "2.3.10",
                    "description": "MQTT-SN Gateway",
                    "active": false
            },
            {
                    "name": "emq_stomp",
                    "version": "2.3.10",
                    "description": "Stomp Protocol Plugin",
                    "active": false
            },
            {
                    "name": "emq_web_hook",
                    "version": "2.3.10",
                    "description": "EMQ Webhook Plugin",
                    "active": false
            }
    ]
}
```

13.9.2 开启/关闭节点的指定插件

API 定义:

```
PUT /api/v2/nodes/{node_name}/plugins/{name}
```

请求参数:

```
{"active": true | false}
```

请求示例:

```
PUT api/v2/nodes/emq@127.0.0.1/plugins/emq_recon
json请求参数:
{
    "active": true
}
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.10 监听器(Listeners)

13.10.1 获取集群节点的监听器列表

API 定义:

```
GET api/v2/monitoring/listeners
```

返回数据:


```
{
    "code": 0,
    "result": {
        "emq@127.0.0.1": [
            {
                "protocol": "dashboard:http",
                "listen": "18083",
                "acceptors": 2,
                "max_clients": 512,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:tcp",
                "listen": "127.0.0.1:11883",
                "acceptors": 16,
                "max_clients": 102400,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:tcp",
                "listen": "0.0.0.0:1883",
                "acceptors": 16,
                "max_clients": 102400,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:ws",
                "listen": "8083",
                "acceptors": 4,
                "max_clients": 64,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:ssl",
                "listen": "8883",
                "acceptors": 16,
                "max_clients": 1024,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:wss",
```


(continues on next page)

(续上页)

```
                "listen": "8084",
                "acceptors": 4,
                "max_clients": 64,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:api",
                "listen": "127.0.0.1:8080",
                "acceptors": 4,
                "max_clients": 64,
                "current_clients": 1,
                "shutdown_count": []
            }
        ]
    }
}
```

13.10.2 获取指定节点的监听器列表

API 定义:

```
GET api/v2/monitoring/listeners/{node_name}
```

请求示例:

```
GET api/v2/monitoring/listeners/emq@127.0.0.1
```

返回数据:

```
{
    "code": 0,
    "result": [
        {
            "protocol": "mqtt:api",
            "listen": "127.0.0.1:8080",
            "acceptors": 4,
            "max_clients": 64,
            "current_clients": 1,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:wss",
            "listen": "8084",
            "acceptors": 4,
            "max_clients": 64,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:ssl",
            "listen": "8883",
            "acceptors": 16,
            "max_clients": 1024,
            "current_clients": 0,
```

(continues on next page)

(续上页)

```
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:ws",
            "listen": "8083",
            "acceptors": 4,
            "max_clients": 64,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:tcp",
            "listen": "0.0.0.0:1883",
            "acceptors": 16,
            "max_clients": 102400,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:tcp",
            "listen": "127.0.0.1:11883",
            "acceptors": 16,
            "max_clients": 102400,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "dashboard:http",
            "listen": "18083",
            "acceptors": 2,
            "max_clients": 512,
            "current_clients": 0,
            "shutdown_count": []
        }
    ]
}
```

13.11 收发报文统计

13.11.1 获取全部节点的收发报文统计

API 定义:

```
GET api/v2/monitoring/metrics/
```

返回数据:

```
{
    "code": 0,
    "result": {
        "packets/disconnect":0,
        "messages/dropped":0,
        "messages/qos2/received":0,
```

(continues on next page)

(续上页)

```
        "packets/suback":0,
        "packets/pubcomp/received":0,
        "packets/unsuback":0,
        "packets/pingresp":0,
        "packets/puback/missed":0,
        "packets/pingreq":0,
        "messages/retained":3,
        "packets/sent":0,
        "messages/qos2/dropped":0,
        "packets/unsubscribe":0,
        "packets/pubrec/missed":0,
        "packets/connack":0,
        "packets/pubrec/sent":0,
        "packets/publish/received":0,
        "packets/pubcomp/sent":0,
        "bytes/received":0,
        "packets/connect":0,
        "packets/puback/received":0,
        "messages/sent":0,
        "packets/publish/sent":0,
        "bytes/sent":0,
        "packets/pubrel/missed":0,
        "packets/puback/sent":0,
        "messages/qos0/received":0,
        "packets/subscribe":0,
        "packets/pubrel/sent":0,
        "messages/qos2/sent":0,
        "packets/received":0,
        "packets/pubrel/received":0,
        "messages/qos1/received":0,
        "messages/qos1/sent":0,
        "packets/pubrec/received":0,
        "packets/pubcomp/missed":0,
        "messages/qos0/sent":0
    }
}
```

13.11.2 获取指定节点的收发报文统计

API 定义:

```
GET api/v2/monitoring/metrics/{node_name}
```

请求示例:

```
GET api/v2/monitoring/metrics/emq@127.0.0.1
```

返回数据:

```
{
    "code": 0,
    "result": {
        "packets/disconnect":0,
        "messages/dropped":0,
        "messages/qos2/received":0,
```

(continues on next page)

(续上页)


```
        "packets/suback":0,
        "packets/pubcomp/received":0,
        "packets/unsuback":0,
        "packets/pingresp":0,
        "packets/puback/missed":0,
        "packets/pingreq":0,
        "messages/retained":3,
        "packets/sent":0,
        "messages/qos2/dropped":0,
        "packets/unsubscribe":0,
        "packets/pubrec/missed":0,
        "packets/connack":0,
        "messages/received":0,
        "packets/pubrec/sent":0,
        "packets/publish/received":0,
        "packets/pubcomp/sent":0,
        "bytes/received":0,
        "packets/connect":0,
        "packets/puback/received":0,
        "messages/sent":0,
        "packets/publish/sent":0,
        "bytes/sent":0,
        "packets/pubrel/missed":0,
        "packets/puback/sent":0,
        "messages/qos0/received":0,
        "packets/subscribe":0,
        "packets/pubrel/sent":0,
        "messages/qos2/sent":0,
        "packets/received":0,
        "packets/pubrel/received":0,
        "messages/qos1/received":0,
        "messages/qos1/sent":0,
        "packets/pubrec/received":0,
        "packets/pubcomp/missed":0,
        "messages/qos0/sent":0
    }
}
```


13.12 连接会话统计

13.12.1 获取全部节点的连接会话统计

API 定义:

```
GET api/v2/monitoring/stats
```

请求示例:

```
GET api/v2/monitoring/stats
```

返回数据:

```
{
    "code": 0,
    "result": [
            {
                    "emq@127.0.0.1": {
                            "clients/count": 0,
                            "clients/max": 0,
                            "retained/count": 3,
                            "retained/max": 3,
                            "routes/count": 0,
                            "routes/max": 0,
                            "sessions/count": 0,
                            "sessions/max": 0,
                            "subscribers/count": 0,
                            "subscribers/max": 0,
                            "subscriptions/count": 0,
                            "subscriptions/max": 0,
                            "topics/count": 0,
                            "topics/max": 0
                    }
            }
    ]
}
```

13.12.2 获取指定节点的连接会话统计

API 定义:

```
GET api/v2/monitoring/stats/{node_name}
```

请求示例:

```
GET api/v2/monitoring/stats/emq@127.0.0.1
```

返回数据:

```
{
      "code": 0,
      "result": {
    "clients/count": 0,
    "clients/max": 0,
    "retained/count": 3,
    "retained/max": 3,
    "routes/count": 0,
    "routes/max": 0,
    "sessions/count": 0,
    "sessions/max": 0,
    "subscribers/count": 0,
    "subscribers/max": 0,
    "subscriptions/count": 0,
    "subscriptions/max": 0,
    "topics/count": 0,
    "topics/max": 0
      }
}
```

13.13 热配置

13.13.1 获取全部节点的可修改配置项

API定义:

```
GET api/v2/configs
```

请求示例:

```
GET api/v2/configs
```

返回数据:


```
{
    "code": 0,
    "result": {
        "emq@127.0.0.1": [
            {
                "key": "log.console.level",
                "value": "error",
                "datatpye": "enum",
                "app": "emqttd"
            },
            {
                "key": "mqtt.acl_file",
                "value": "etc/acl.conf",
                "datatpye": "string",
                "app": "emqttd"
            },
            {
                "key": "mqtt.acl_nomatch",
                "value": "allow",
                "datatpye": "enum",
                "app": "emqttd"
            },
            {
                "key": "mqtt.allow_anonymous",
                "value": "true",
                "datatpye": "enum",
                "app": "emqttd"
            },
            {
                "key": "mqtt.broker.sys_interval",
                "value": "60",
                "datatpye": "integer",
                "app": "emqttd"
            },
            {
                "key": "mqtt.cache_acl",
                "value": "true",
                "datatpye": "enum",
                "app": "emqttd"
            }
        ]
    }
}
```

13.13.2 获取指定节点的可修改配置项

API定义:

```
GET api/v2/nodes/{node_name}/configs
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/configs
```

返回数据:


```
{
    "code": 0,
    "result": [
        {
            "key": "log.console.level",
            "value": "error",
            "datatpye": "enum",
            "app": "emqttd"
        },
        {
            "key": "mqtt.acl_file",
            "value": "etc/acl.conf",
            "datatpye": "string",
            "app": "emqttd"
        },
        {
            "key": "mqtt.acl_nomatch",
            "value": "allow",
            "datatpye": "enum",
            "app": "emqttd"
        },
        {
            "key": "mqtt.allow_anonymous",
            "value": "true",
            "datatpye": "enum",
            "app": "emqttd"
        },
        {
            "key": "mqtt.broker.sys_interval",
            "value": "60",
            "datatpye": "integer",
            "app": "emqttd"
        },
        {
            "key": "mqtt.cache_acl",
            "value": "true",
            "datatpye": "enum",
            "app": "emqttd"
        }
    ]
}
```


13.13.3 修改全部节点的配置项

API定义:

```
PUT /api/v2/configs/{app_name}
```

请求参数:

```
{
    "key"   : "mqtt.allow_anonymous",
    "value" : "false"
}
```

请求示例:

```
PUT /api/v2/configs/emqttd
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.13.4 修改指定节点的配置项

API定义:

```
PUT /api/v2/nodes/{node_name}/configs/{app_name}
```

请求参数:

```
{
    "key"   : "mqtt.allow_anonymous",
    "value" : "false"
 }
```

请求示例:

```
PUT /api/v2/nodes/emq@127.0.0.1/configs/emqttd
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.13.5 获取指定节点的指定插件的配置项

API定义:

```
GET api/v2/nodes/{node_name}/plugin_configs/{plugin_name}
```

请求示例:

```
GET api/v2/nodes/emq@127.0.0.1/plugin_configs/emq_auth_http
```

返回数据:


```
{
    "code": 0,
    "result": [
            {
                    "key": "auth.http.auth_req",
                    "value": "http://127.0.0.1:8080/mqtt/auth",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.auth_req.method",
                    "value": "post",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.auth_req.params",
                    "value": "clientid=%c,username=%u,password=%P",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.super_req",
                    "value": "http://127.0.0.1:8080/mqtt/superuser",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.super_req.method",
                    "value": "post",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.super_req.params",
                    "value": "clientid=%c,username=%u",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.acl_req",
                    "value": "http://127.0.0.1:8080/mqtt/acl",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.acl_req.method",
                    "value": "get",
                    "desc": "",
                    "required": true
            },
            {
                    "key": "auth.http.acl_req.params",
                    "value": "access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t",
                    "desc": "",
                    "required": true
```


(continues on next page)

(续上页)

```
            }
    ]
}
```

13.13.6 修改指定节点的指定插件的配置项

API定义:

```
PUT api/v2/nodes/{node_name}/plugin_configs/{plugin_name}
```

请求参数:

```
{
    "auth.http.auth_req.method": "get",
    "auth.http.auth_req": "http://127.0.0.1:8080/mqtt/auth",
    "auth.http.auth_req.params": "clientid=%c,username=%u,password=%P",
    "auth.http.acl_req.method": "get",
    "auth.http.acl_req": "http://127.0.0.1:8080/mqtt/acl",
    "auth.http.acl_req.params": "access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t
↪",
    "auth.http.super_req.method": "post",
    "auth.http.super_req.params": "clientid=%c,username=%u",
    "auth.http.super_req": "http://127.0.0.1:8080/mqtt/superuser"
}
```

请求示例:

```
PUT api/v2/nodes/emq@127.0.0.1/plugin_configs/emq_auth_http
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.14 用户管理

13.14.1 获取管理用户列表

API定义:

```
GET api/v2/users
```

请求示例:

```
GET api/v2/users
```

返回数据:

```
{
    "code": 0,
    "result": [
        {
            "username": "admin",
            "tags": "administrator"
        }
    ]
}
```

13.14.2 添加管理用户

API定义:

```
POST api/v2/users
```

请求参数:

```
{
    "username": "test_user",
    "password": "password",
    "tags": "user"
}
```

请求示例:

```
POST api/v2/users
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.14.3 修改管理用户信息

API定义:

```
PUT api/v2/users/{username}
```

请求参数:

```
{
    "tags": "admin"
}
```

请求示例:

```
PUT api/v2/users/test_user
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.14.4 删除管理用户

API定义:

```
DELETE api/v2/users/{username}
```

请求参数:

请求示例:

```
DELETE api/v2/users/test_user
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.14.5 认证管理用户

API定义:

```
POST api/v2/auth
```

请求参数:

```
{
    "username": "test_user",
    "password": "password"
}
```

请求示例:

```
POST api/v2/auth
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.14.6 修改管理用户密码

API定义:

```
PUT api/v2/change_pwd/{username}
```

请求参数:

```
{
    "new_pwd": "newpassword",
    "old_pwd": "password"
}
```

请求示例:

```
PUT api/v2/change_pwd/test_user
```

返回数据:

```
{
    "code": 0,
    "result": []
}
```

13.15 返回错误码

错误码	备注
0	成功
101	badrpc
102	未知错误
103	用户名密码错误
104	用户名密码不能为空
105	删除的用户不存在
106	admin用户不能删除
107	请求参数缺失
108	请求参数类型错误
109	请求参数不是json类型
110	插件已经加载，不能重复加载
111	插件已经卸载，不能重复卸载
112	用户不在线
113	用户已经存在
114	旧密码错误

CHAPTER 14

测试调优 (Tuning Guide)

EMQ 消息服务器1.x版本 MQTT 连接压力测试到130万，在一台8核心、32G内存的 CentOS 服务器上。

100万连接测试所需的 Linux 内核参数，网络协议栈参数，Erlang 虚拟机参数，EMQ 消息服务器参数设置如下:

14.1 Linux 操作系统参数

系统全局允许分配的最大文件句柄数:

```
# 2 millions system-wide
sysctl -w fs.file-max=2097152
sysctl -w fs.nr_open=2097152
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
```

14.1.1 /etc/sysctl.conf

持久化 'fs.file-max' 设置到 /etc/sysctl.conf 文件:

```
fs.file-max = 1048576
```

/etc/systemd/system.conf 设置服务最大文件句柄数:

```
DefaultLimitNOFILE=1048576
```

14.1.2 /etc/security/limits.conf

/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
```

14.2 TCP 协议栈网络参数

并发连接 backlog 设置:

```
sysctl -w net.core.somaxconn=32768
sysctl -w net.ipv4.tcp_max_syn_backlog=16384
sysctl -w net.core.netdev_max_backlog=16384
```

可用知名端口范围:

```
sysctl -w net.ipv4.ip_local_port_range='1000 65535'
```

TCP Socket 读写 Buffer 设置:

```
sysctl -w net.core.rmem_default=262144
sysctl -w net.core.wmem_default=262144
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.core.optmem_max=16777216

#sysctl -w net.ipv4.tcp_mem='16777216 16777216 16777216'
sysctl -w net.ipv4.tcp_rmem='1024 4096 16777216'
sysctl -w net.ipv4.tcp_wmem='1024 4096 16777216'
```

TCP 连接追踪设置:

```
sysctl -w net.nf_conntrack_max=1000000
sysctl -w net.netfilter.nf_conntrack_max=1000000
sysctl -w 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
```

14.3 Erlang 虚拟机参数

优化设置 Erlang 虚拟机启动参数，配置文件 emqttd/etc/emq.conf:

```
## Erlang Process Limit
node.process_limit = 2097152

## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 1048576
```

14.4 EMQ 消息服务器参数

设置 TCP 监听器的 Acceptor 池大小，最大允许连接数。配置文件 emqttd/etc/emq.conf:

```
## TCP Listener
listener.tcp.external = 0.0.0.0:1883
listener.tcp.external.acceptors = 64
listener.tcp.external.max_clients = 1000000
```

14.5 测试客户端设置

测试客户端服务器在一个接口上，最多只能创建65000连接:

```
sysctl -w net.ipv4.ip_local_port_range="500 65535"
echo 1000000 > /proc/sys/fs/nr_open
ulimit -n 100000
```

14.5.1 emqtt_benchmark

并发连接测试工具: http://github.com/emqtt/emqtt_benchmark

CHAPTER 15

版本发布 (Changes)

15.1 2.3.10 版本

发布日期*: 2018-06-27*

15.1.1 Bugfix and Enhancements

Upgrade the esockd library to v5.2.2

15.1.2 emq-auth-http

Ignore auth on ignore in body, allows for chaining methods

15.2 2.3.9 版本

发布日期*: 2018-05-20*

15.2.1 Bugfix and Enhancements

Bugfix: check params for REST publish API (#1599)

Upgrade the mongodb library to v3.0.5

15.2.2 esockd

Bugfix: proxy protocol - set socket to binary mode (#78)

15.3 2.3.8 版本

发布日期: *2018-05-11*

15.3.1 Bugfix and Enhancements

Bugfix: unregister users CLI when unload emq_auth_username (#1588)

Bugfix: Should be an info level when change CleanSession (#1590)

Bugfix: emqttd_ctl crashed when emq_auth_usename doesn't exist (#1588)

15.3.2 emq-auth-mongo

Improve: Support authentication database (authSource) (#116)

15.4 2.3.7 版本

发布日期: *2018-04-22*

15.4.1 Bugfix and Enhancements

Bugfix: fixed spec of function setstats/3 (#1575)

Bugfix: clean dead persistent session on connect (#1575)

Bugfix: dup flag not set when re-deliver (#1575)

Bugfix: Upgrade the lager_console_backend config (#1575)

Improve: Support set k8s namespace (#1575)

Upgrade the ekka library to v0.2.3 (#1575)

Improve: move PIPE_DIR dir from /tmp/${WHOAMI}_erl_pipes/$NAME/ to /$RUNNER_DATA_DIR/${WHOAMI}_erl_pipes/$NAME/ (emq-relx#188)

15.4.2 emq-auth-http

Improve: Retry 3 times when httpc:request occurred socket_closed_remotely error (emq-auth-http#70)

15.5 2.3.6 版本

发布日期: *2018-03-25*

15.5.1 Bugfix and Enhancements

Security: LWT message checking the ACL (#1524)

Bugfix: Retain msgs should not be sent to existing subscriptions (#1529)

15.5.2 emq-auth-jwt

Validate JWT token using a expired field (#29)

15.6 2.3.5 版本

发布日期: *2018-03-03*

15.6.1 Bugfix and Enhancements

Feature: Add etc/ssl_dist.conf file for erlang SSL distribution (emq-relx#178)

Feature: Add node.ssl_dist_optfile option and etc/ssl_dist.conf file (#1512)

Feature: Support Erlang Distribution over TLS (#1512)

Improve: Tune off the 'tune_buffer' option for external MQTT connections (#1512)

15.6.2 emq-sn

Clean registered topics if mqtt-sn client send a 2nd CONNECT in connected state (#76)

Upgrade the esockd library to v5.2.1 (#76)

15.6.3 emq-auth-http

Remove 'password' param from ACL and superuser requests (#66)

15.7 2.3.4 版本

发布日期: *2018-01-29*

15.7.1 Bugfix and Enhancements

Feature: Forward real client IP using a reverse proxy for websocket (#1335)

Feature: EMQ node.name with link local ipv6 address not responding to ping (#1460)

Feature: Add PROTO_DIST_ARG flag to support clustering via IPv6 address. (#1460)

Bugfix: retain bit is not set when publishing to clients (when it should be set). (#1461)

Bugfix: Can't search topic on web dashboard (#1473)

15.7.2 emq-sn

Bugfix: CONNACK is not always sent to the client (emq-sn#67)

Bugfix: Setting the port to ::1:2000 causes error (emq-sn#66)

15.8 2.3.3 版本

发布日期: *2018-01-08*

15.8.1 Bugfix and Enhancements

Add a full documentation for *emq.conf* and plugins.

Repair a dead link in README - missing emq-lwm2m. (#1430)

Subscriber with wildcard topic does not receive retained messages with sub topic has $ sign (#1398)

Web Interface with NGINX Reverse Proxy not working. (#953)

15.8.2 emq-dashboard

Add *dashboard.default_user.login*, *dashboard.default_user.password* options to support configuring default admin.

15.8.3 emq-modules

The emq-modules rewrite config is not right. (#35)

15.8.4 emq-docker

Upgrade alpine to 3.7 (#31)

15.8.5 emq-packages

Support ARM Platform (#12)

15.9 2.3.2 版本

发布日期: *2017-12-26*

15.9.1 Bugfix and Enhancements

Support X.509 certificate based authentication (#1388)

Add proxy_protocol, proxy_protocol_timeout options for ws/wss listener.

Cluster discovery etcd nodes key must be created manually. (#1402)

Will read an incorrect password at the last line of emq_auth_username.conf (#1372)

How can I use SSL/TLS certificate based client authentication? (#794)

Upgrade the esockd library to v5.2.

15.9.2 esockd

Improve the parser of proxy protocol v2.

Add 'send_timeout', 'send_timeout_close' options.

Rename esockd_transport:port_command/2 function to async_send/2.

Add test case for esockd_transport:async_send/2 function.

Add esockd_transport:peer_cert_subject/1, peer_cert_common_name/1 functions.

15.9.3 emq-auth-mysql

Update depends on emqtt/mysql-otp.

Fixed the issue that Cannot connect to MySQL 5.7 (#67).

15.9.4 emq-relx

Fix mergeconf/3 appending line break error. (#152)

15.9.5 emq-sn

Fix crash in emq_sn_gateway:transform() function which handles SUBACK. (#57)

Define macro SN_RC_MQTT_FAILURE. (#59)

15.9.6 emq-web-hook

Filter auth_failure client for disconnected hook. (#30)

15.10 2.3.1 版本

发布日期*: 2017-12-03*

15.10.1 Bugfix and Enhancements

Remove the unnecessary transactions to optimize session management.

Should not exit arbitrarily when clientid conflicts in mnesia.

Change the default value of 'mqtt.session.enable_stats' to 'on'.

The DUP flag should be set to 0 for all QoS0 messages. (emqttd#1319)

Fix the 'no function clause' exception. (emqttd#1293)

The retained flags should be propagated for bridge. (emqttd#1293)

The management API should listen on 0.0.0.0:8080. (emqttd#1353)

Fast close the invalid websocket in init/1 function.

erlang:demonitor/1 the reference when erasing a monitor. (emqttd#1340)

15.10.2 emq-retainer

Don't clean the retain flag after the retained message is stored.

Add three CLIs for the retainer plugin. (emq-retainer#38)

15.10.3 emq-dashboard

Refactor(priv/www): improve the *routing* page. (emq-dashboard#185)

15.10.4 emq-modules

Turn off the *subscription* module by default. (emq-modules#26)

15.10.5 emq-sn

Add an integration test case for sleeping device.

Do not send will topic if client is kicked out.

Prevent crash information in log when emq_sn_gateway getting timeout, since it is a possible procedure.

15.10.6 emq-relx

Support node cookie value with = characters. (emq-relx#146)

15.10.7 mochiweb

Improve Req:get(peername) funciton to support *x-forwarded-for* and *x-remote-port*. (emqtt/mochiweb#9)

15.11 2.3.0 版本 "Passenger's Log"

发布日期*: 2017-11-20*

EMQ 2.3.0 版本正式发布，改进了 PubSub 设计与消息路由性能，更新 EMQ 自带的自签名 SSL 证书，改进 Dashboard 界面与 API 设计。

15.11.1 Bugfix and Enhancements

Fix the issue that Retained message is not sent for Subscribe to existing topic. (emqttd#1314)

Fix the issue that The DUP flag MUST be set to 0 for all QoS0 messages.(emqttd#1319)

Improve the pubsub design and fix the race-condition issue. (emqttd#PR1342)

Crash on macOS High Sierra (emqttd#1297)

15.11.2 emq-dashboard Plugin (emq-dashboard#PR174)

Upgraded the 'subscriptions' RESTful API.

Improvement of the auth failure log. (emq-dashboard#59)

15.11.3 emq-coap Plugin (emq-coap#PR61)

Replaced coap_client with er_coap_client.

Fix: correct the output format of coap_discover() to enable ".well-known/core".

Refactor the coap_discover method.

15.11.4 emq-relx

Upgraded the *bin/nodetool* script to fix the *rpcterms* command.

15.11.5 emq-web-hook Plugin

Fix the emq_web_hook plugin getting username from client.connected hook. (emq-web-hook#19)

15.11.6 emq-auth-jwt Plugin(emq-auth-jwt#PR15)

Added test cases for emq_auth_jwt.

Fix jwt:decode/2 functions's return type.

15.11.7 emq-auth-mongo Plugin(emq-auth-mongo#PR92)

Updated the default MongoDB server configuration.

15.12 2.3-rc.2 版本

发布日期*: 2017-10-22*

Change the default logging level of *trace* CLI. (emqttd#1306)

15.12.1 emq-dashboard Plugin (emq-dashboard#164)

Fix the 'Status' filters of plugins's management.

Fix the URL Redirection when deleting an user.

Compatible with IE,Safari,360 Browsers.

15.13 2.3-rc.1 版本

发布日期: *2017-10-12*

Fix the issue that invalid clients can publish will message. (emqttd#1230)

Fix Dashboard showing no stats data (emqttd#1263)

Fix a rare occurred building failure (emqttd#1284)

Support Persistence Logs for longer time (emqttd#1275)

Fix for users APIs (emqttd#1289)

Changed passwd_hash/2 function's return type (emqttd#1289)

15.13.1 emq-dashboard Plugin (emq-dashboard#154)

Improved the Dashboard Interface of Monitoring/Management/Tools.

Allow switching dashboard themes.

Supoort both EN and CN languages.

15.14 2.3-beta.4 版本

发布日期: *2017-09-13*

15.14.1 Highlights

Released a new sexy dashboard.

Add more RESTful APIs for manangement and monitoring.

Configuring the broker through CLI or API without having to restart.

15.14.2 Bugfix

Job for emqttd.service failed because the control process exited with error code. (emqttd#1238)

Travis-CI Build Failing (emqttd#1221)

Https listener of Dashboard plugin won't work (emqttd#1220)

Service not starting on Debian 8 Jessie (emqttd#1228)

15.14.3 emq-dashboard

1. Support switching to other clustered node.
2. Configure and reboot the plugins on the dashboard.
3. A login page to replace the basic authentication popup window.

15.14.4 emq-coap

1.Try to clarify the relationship between coap and mqtt in EMQ. (emq-coap#54).

2.Fix crashes in coap concurrent test(gen-coap#3).

15.15 2.3-beta.3 版本

发布日期*: 2017-08-21*

15.16 2.3-beta.3 版本

发布日期*: 2017-08-21*

15.16.1 Enhancements

Add HTTP API for hot configuration.

15.16.2 Bugfix

1. Parse 'auth.mysql.password_hash' error when hot configuration reload (emq-auth-mysql#68)
2. Set 'auth.pgsql.server' error when hot configuration reload (emq-auth-pgsql#67)
3. Set 'auth.redis.server' and 'auth.redis.password_hash' error when hot configuration reload (emq-auth-redis#47)
4. Fix the issue that when deleting retained message subscribed clients are not notified (emqttd#1207)
5. Support more parameters for hot configuration reload:

- mqtt.websocket_protocol_header = on
- mqtt.mqueue.low_watermark = 20%
- mqtt.mqueue.high_watermark = 60%
- mqtt.client.idle_timeout = 30s
- mqtt.client.enable_stats = off

15.17 2.3-beta.2 版本

发布日期*: 2017-08-12*

EMQ R2.3-beta.2 版本发布！该版本新增 HTTP 管理 API，支持配置 Keepalive 检测周期，支持配置参数热更新。

目前支持配置热更新的插件有:

- emq-stomp
- emq-coap
- emq-sn

- emq-lwm2m
- emq-retainer
- emq-recon
- emq-web-hook
- emq-auth-jwt
- emq-auth-http
- emq-auth-mongo
- emq-auth-mysql
- emq-auth-pgsql
- emq-auth-redis

注解: 为支持命令行更新配置参数，部分认证插件参数值采用','替代了空格分隔符。

15.17.1 Enhancements

1. Introduce new HTTP management API.
2. Add ClientId parameter for HTTP Publish API.
3. Allow configuring keepalive backoff.
4. Remove the fullsweep_after option to lower CPU usage.
5. Authorize HTTP Publish API with clientId.

15.17.2 emq-sn Plugin (emq-sn#49)

1. Support CONNECT message in connected/wait_for_will_topic/wait_for_will_msg states.
2. Clean registered topic for a restarted client.
3. Bug fix of not clearing buffered PUBLISH messages received during asleep state as those messages are sent to client when client wakes up.

15.17.3 emq-auth-ldap Plugin (emq-auth-ldap#21)

Improve the design LDAP authentication.

15.17.4 emq-coap Plugin (emq-coap#51)

Support CoAP PubSub Specification (https://www.ietf.org/id/draft-ietf-core-coap-pubsub-02.txt)

15.18 2.3-beta.1 版本

发布日期: *2017-07-24*

EMQ R2.3-beta.1版本发布！该版本正式支持集群节点自动发现与集群脑裂自动愈合，支持基于IP Multicast、Etcd、Kubernetes等多种策略自动构建集群。

15.18.1 节点发现与自动集群

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
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

15.18.2 集群脑裂与自动愈合

EMQ R2.3版本正式支持集群脑裂自动愈合(Network Partition Autoheal):

```
cluster.autoheal = on
```

集群脑裂自动恢复流程:

1. 节点收到Mnesia库的'inconsistent_database'事件3秒后进行集群脑裂确认;
2. 节点确认集群脑裂发生后，向Leader节点(集群中最早启动节点)上报脑裂消息;
3. Leader节点延迟一段时间后，在全部节点在线状态下创建脑裂视图(SplitView);
4. Leader节点在多数派(Majority)分区选择集群自愈的Coordinator节点;
5. Coordinator节点重启少数派(minority)分区节点恢复集群。

15.18.3 节点宕机与自动清除

EMQ R2.3版本支持从集群自动删除宕机节点(Autoclean):

```
cluster.autoclean = 5m
```

15.18.4 LWM2M协议支持

EMQ R2.3 版本正式支持LWM2M协议网关，实现了LWM2M协议的大部分功能。MQTT客户端可以通过EMQ-LWM2M访问支持LWM2M的设备。设备也可以往EMQ-LWM2M上报notification，为EMQ后端的服务采集数据。

LWM2M是由Open Mobile Alliance(OMA)定义的一套适用于物联网的协议，它提供了设备管理和通讯的功能。LWM2M使用CoAP作为底层的传输协议，承载在UDP或者SMS上

15.18.5 JSON Web Token认证支持

EMQ R2.3 版本支持基于JWT(JSON Web Token)的MQTT客户端认证。

15.18.6 Retainer插件

Retainer插件支持'disc_only'模式存储retained消息。

15.18.7 Debian 9 安装包

EMQ R2.3 支持Debian 9系统安装包。

15.18.8 Erlang/OTP R20

EMQ R2.3 版本兼容Erlang/OTP R20，全部程序包基于Erlang/OTP R20构建。

15.19 2.2 正式版 "Nostalgia"

发布日期*: 2017-07-08*

版本别名*: Nostalgia*

EMQ-2.2.0版本正式发布！EMQ R2.2版本完整支持CoAP(RFC 7252)、MQTT-SN协议，支持Web Hook、Lua Hook、Proxy Protocol V2，支持Elixir语言插件开发。

Feature: Add 'listeners restart/stop' CLI command (emqttd#1135)

Bugfix: Exit Code from emqttd_ctl (emqttd#1133)

Bugfix: Fix spec errors found by dialyzer (emqttd#1136)

Bugfix: Catch exceptions thrown from rpc:call/4 (emq-dashboard#128)

Bugfix: Topic has been decoded by gen-coap, no conversion needed (emq-coap#43)

15.20 2.2-rc.2 版本

发布日期*: 2017-07-03*

> **警告:** 2.2-rc.2版本源码编译需要Erlang/OTP R19.3+

15.20.1 问题与改进

Compatible with Erlang/OTP R20 (emq-relx#77)

CoAP gateway plugin supports coap-style publish & subscribe pattern. (emq_coap#33)

MQTT-SN gateway plugin supports sleeping device (emq_sn#32)

Upgrade esockd and mochiweb libraries to support restarting a listener

15.21 2.2-rc.1 版本

发布日期: *2017-06-14*

15.21.1 问题与改进

Add a new listener for HTTP REST API (emqttd#1094)

Fix the race condition issue caused by unregister_session/1 (emqttd#1096)

Fix the issue that we cannot remove a down node from the cluster (emqttd#1100)

Passed org.eclipse.paho.mqtt_sn.testing/interoperability tests (emq_sn#29)

Fix the issue that send http request and return non-200 status code, but AUTH/ACL result is denied (emq-auth-http#33)

Fix the issue that fail to stop listener (emq_stomp#24)

Support using systemctl to manage emqttd service on CentOS

15.22 2.2-beta.3 版本

发布日期: *2017-05-27*

15.22.1 问题与改进

Call emit_stats when force GC (emqttd#1071)

Update the default value of 'mqtt.mqueue.max_length' to 1000 (emqttd#1074)

Update emq-auth-mongo READEME (emq-auth-mongo#66)

Update default password field (emq-auth-mongo#67)

Upgrade the mongodb library to v3.0.3

Remove 'check password===undefined && userName!== undefined' (emq-dashboard#120)

15.22.2 emq_auth_redis插件

认证支持HGET查询

15.22.3 emq_auth_mongo插件

支持mongodb集群、Replica Set

15.22.4 文档更新

更新Windows源码编译安装

15.23 2.2-beta.2 版本

发布日期: *2017-05-20*

15.23.1 问题与改进

Add a 'websocket_protocol_header' option to handle WebSocket connection from WeChat (emqttd#1060)

Assign username and password to MQTT-SN's CONNECT message (emqttd#1041)

Allow for Content-Type:application/json in HTTP Publish API (emqttd#1045)

emqttd_http.erl:data conversion (emqttd#1059)

Seperate emq_sn from emqttd (emq-sn#24)

Check St0's type, making it easier to debug crash problems (emq-lua-hook#6)

Fix error: load xxx.lua (emq-lua-hook#8)

Leave luerl alone as a rebar project (emq-lue-hook#9)

Display websocket data in reverse order (emq-dashboard#118)

priv/www/assets/js/dashboard.js:Fixed a typo (emq-dashboard#118)

15.23.2 Update README

Update README of emq-auth-pgsql: add the 'ssl_opts' configuration (emq-auth-pgsql#56)

Update README of emq-auth-mysql: fix the 'passwd_hash' typo (emq-auth-mysql#54)

Update README of emq-auth-mongo: change 'aclquery' to 'acl_query' (emq-auth-mongo#63)

15.23.3 Elixir Plugin

Add a new plugin emq-elixir-plugin to support Elixir language.

15.24 2.2-beta.1 版本

发布日期: *2017-05-05*

EMQ 2.2-beta.1版本正式发布！EMQ2.2 版本发布主要新功能包括:

1. 支持MQTT协议多监听器配置，支持HAProxy的Proxy Protocol V1/V2
2. 新增Web Hook插件(emq-web-hook)、Lua Hook插件(emq-lua-hook)

15.24.1 MQTT协议监听器配置

一个EMQ节点可配置多个MQTT协议监听端口，例如下述配置external, internal监听器，分别用于设备连接与内部通信:

```
                          -------
-- Ex, 支持Web Hook、Lua Hook、ernal TCP 1883 --> |     |
                          | EMQ | -- Internal TCP 2883 --> Service
-- External SSL 8883-->   |     |
                          -------
```

EMQ 2.2 版本etc/emq.conf监听器配置方式:

```
listener.tcp.${name}= 127.0.0.1:2883

listener.tcp.${name}.acceptors = 16

listener.tcp.${name}.max_clients = 102400
```

15.24.2 Proxy Protocol V1/2支持

EMQ 集群通常部署在负载均衡器(LB)后面，典型架构:

```
               -----
               |   |
               | L | --TCP 1883--> EMQ
--SSL 8883--> |   |                |
               | B | --TCP 1883--> EMQ
               |   |
               -----
```

HAProxy、NGINX等常用的负载均衡器(LB)，一般通过Proxy Protocol协议传递TCP连接源地址、源端口给EMQ。

EMQ 2.2 版本的监听器开启Proxy Protocol支持:

```
## Proxy Protocol V1/2
## listener.tcp.${name}.proxy_protocol = on
## listener.tcp.${name}.proxy_protocol_timeout = 3s
```

15.24.3 Web Hook插件

新增WebHook插件: emq-web-hook ， 支持在MQTT客户端上下线、消息发布订阅时触发WebHook回调。

15.24.4 Lua Hook插件

新增Lua Hook插件: emq-lua-hook ， 支持Lua脚本注册EMQ扩展钩子来开发插件。

15.24.5 改进认证链设计

EMQ 2.2 版本改进认证链设计，当前认证模块返回ignore(例如用户名不存在等情况下)，认证请求将继续转发后面认证模块:

```
            -------------             ------------             -------------
Client --> | Redis认证 | -ignore-> | HTTP认证 | -ignore-> | MySQL认证 |
            -------------             ------------             -------------
                  |                         |                        |
                 \|/                       \|/                      \|/
            allow | deny              allow | deny              allow | deny
```

15.24.6 支持bcrypt密码Hash

EMQ 2.2 版本支持bcrypt密码Hash方式，例如Redis认证插件配置:

```
auth.redis.password_hash = bcrypt
```

15.24.7 etc/emq.conf配置变更

'mqtt.queue.*' 配置变更为 'mqtt.mqueue.*'

15.24.8 emq-dashboard

WebSocket页面支持Unsubscribe

15.25 2.1.2 版本

发布日期*: 2017-04-21*

Fix *emqttd_ctl sessions list* CLI

Newline character in emq.conf causing error;(emqttd#1000)

Fix crash caused by duplicated PUBREC packet (emqttd#1004)

Unload the 'session.created' and 'session.teminated' hooks (emq-plugin-template)

15.26 2.1.1 版本

发布日期*: 2017-04-14*

Localhost:8083/status returns 404 when AWS LB check the health of EMQ (emqttd#984)

Https listener not working in 2.1.0 as in 2.0.7 (emq-dashboard#105)

Fix mqtt-sn Gateway not working (emq-sn#12)

Upgrade emq-sn Plugin (emq-sn#11)

Upgrade emq-coap Plugin (emq-coap#21)

15.27 2.1.0 版本

发布日期: *2017-04-07*

The stable release of 2.1 version.

Trouble with auth.mysql.acl_query (emq-auth-mysql#38)

Filter the empty fields in ACL table (emq-auth-mysql#39)

15.28 2.1.0-rc.2 版本

发布日期: *2017-03-31*

Support pbkdf2 hash (emq-auth-mongo#46)

Kickout the conflict WebSocket connection (emqttd#963)

Correct licence in app.src (emqttd#958)

SSL options to connect to pgsql (emq-auth-pgsql#41)

15.29 2.1.0-rc.1 版本

发布日期: *2017-03-24*

EMQ fails to start if run under a different linux user than that which first ran it (emqttd#842)

Depend on emqtt/pbkdf2 to fix the building errors of Travis CI (emqttd#957)

Depend on goldrush and emqtt/pbkdf2 to resolve the building errors (emqttd#956)

Fix 'rebar command not found' (emq-relx#33)

Compile error in v2.1.0-beta.2 (emq-relx#32)

Support salt with passwords (emq-auth-mongo#11)

Change the default storage_type to 'ram' (emq-retainer#13)

15.30 2.1.0-beta.2 版本

发布日期: *2017-03-13*

Cannot find AwaitingAck (emqttd#597)

EMQ V2.1 crash when public with QoS = 2 (emqttd#919)

Support pbkdf2 hash (emqttd#940)

Add src/emqttd.app.src to be compatible with rebar3 (emqttd#920)

Add more test cases (emqttd#944)

CRASH REPORT Process <0.1498.0> with 0 neighbours crashed with reason: {ssl_error,{tls_alert,"certificate unknown"}} in esockd_connection:upgrade (emqttd#915)

'auth.redis.password_hash = plain' by default (emq-auth-redis#20)

15.31 2.1.0-beta.1 版本

发布日期: *2017-02-24*

EMQ 2.1.0-beta.1版本发布。

> **警告:** 2.1.x版本源码编译需要Erlang/OTP R19+

EMQ正式采用 **'Semantic Versioning 2.0.0<http://semver.org>'_** 规范创建发布版本号，按'Tick-Tock'方式按月发布迭代版本。奇数版本问题修复与性能改进，偶数版本架构改进和新功能布。

15.31.1 GC优化

1. WebSocket、Client、Session进程空置一段时间后自动Hibernate与GC。
2. 新增'mqtt.conn.force_gc_count'配置，Client、Session进程处理一定数量消息后强制GC。
3. 大幅降低WebSocket、Client、Session进程fullsweep_after设置，强制进程深度GC。

15.31.2 API改进

Hooks API支持注册带Tag的回调函数，解决相同模块函数多次Hook注册问题。

15.31.3 问题修复

emqttd#916: Add 'mqtt_msg_from()' type

emq-auth-http#15: ACL endpoint isnt called

15.32 2.1-beta 版本

发布日期: *2017-02-18*

EMQ v2.1-beta版本正式发布，改进Session/Inflight窗口设计，一个定时器负责全部Inflight QoS1/2消息重传，大幅降低高消息吞吐情况下的CPU占用。

15.32.1 Client, Session统计信息

支持对单个Client、Session进程进行统计，etc/emq.conf配置文件中设置'enable_stats'开启:

```
mqtt.client.enable_stats = 60s

mqtt.session.enable_stats = 60s
```

15.32.2 新增missed统计指标

EMQ收到客户端PUBACK、PUBREC、PUBREL、PUBCOMP报文，但在Inflight窗口无法找到对应消息时，计入missed统计指标:

```
packets/puback/missed

packets/pubrec/missed

packets/pubrel/missed

packets/pubcomp/missed
```

15.32.3 Syslog日志集成

支持输出EMQ日志到Syslog，etc/emq.config配置项:

```
## Syslog. Enum: on, off
log.syslog = on

##  syslog level. Enum: debug, info, notice, warning, error, critical, alert,␣
↪emergency
log.syslog.level = error
```

15.32.4 Tune QoS支持

支持订阅端升级QoS，etc/emq.conf配置项:

```
mqtt.session.upgrade_qos = on
```

15.32.5 'acl reload'管理命令

Reload acl.conf without restarting emqttd service (#885)

15.32.6 配置项变更

1. 变更 mqtt.client_idle_timeout 为 mqtt.client.idle_timeout
2. 新增 mqtt.client.enable_stats 配置项
3. 新增 mqtt.session.upgrade_qos 配置项
4. 删除 mqtt.session.collect_interval 配置项
5. 新增 mqtt.session.enable_stats 配置项
6. 变更 mqtt.session.expired_after 为 mqtt.session.expiry_interval

15.32.7 合并扩展模块到emq_modules项目

合并emq_mod_presence, emq_mod_subscription, emq_mod_rewrite到emq_modules项目

变更emq_mod_retainer为emq_retainer项目

15.32.8 Dashboard插件

Overview页面增加missed相关统计指标。 Client页面增加SendMsg、RecvMsg统计指标。 Session页面增加DeliverMsg、EnqueueMsg指标。

15.32.9 recon插件

变更recon.gc_interval配置项类型为duration

15.32.10 reloader插件

变更reloader.interval配置项类型为duration

15.33 2.0.7 版本

发布日期*: 2017-01-20*

The Last Maintenance Release for EMQ 2.0, and support to build RPM/DEB Packages.

emq-auth-http#9: Update the priv/emq_auth_http.schema, *cuttlefish:unset()* if no super_req/acl_req config exists

emq-auth-mongo#31: *cuttlefish:unset()* if no ACL/super config exists

emq-dashboard#91: Fix the exception caused by binary payload

emq-relx#21: Improve the *binemqttd.cmd* batch script for windows

emqttd#873: Documentation: installing-from-source

emqttd#870: Documentation: The word in Documents is wrong

emqttd#864: Hook 'client.unsubscribe' need to handle 'stop'

emqttd#856: Support variables in etc/emq.conf: {{ runner_etc_dir }}, {{ runner_etc_dir }}, {{ runner_data_dir }}

15.34 2.0.6 版本

发布日期*: 2017-01-08*

Upgrade the esockd library to v4.1.1

esockd#41: Fast close the TCP socket if ssl:ssl_accept failed

emq-relx#15: The EMQ 2.0 broker cannot run on Windows.

emq-auth-mongo#31: Mongodb ACL Cannot work?

15.35 2.0.5 版本

发布日期*: 2016-12-24*

emq-auth-http#9: Disable ACL support

emq-auth-mongo#29: Disable ACL support

emq-auth-mongo#30: {datatype, flag}

15.36 2.0.4 版本

发布日期: *2016-12-16*

emqttd#822: Test cases for SSL connections

emqttd#818: trap_exit to link WebSocket process

emqttd#799: Can't publish via HTTPS

15.37 2.0.3 版本

发布日期: *2016-12-12*

emqttd#796: Unable to forbidden tcp lisener

emqttd#814: Cannot remove a 'DOWN' node from the cluster

emqttd#813: Change parameters order

emqttd#795: Fix metrics of websocket connections

emq-dashboard#88: Rename the default topic from "/World" to "world"

emq-dashboard#86: Lookup all online clients

emq-dashboard#85: Comment the default listener port

emq-mod-retainer#3: Retained messages get lost after EMQTT broker restart.

15.38 2.0.2 版本

发布日期: *2016-12-05*

emqttd#787: Stop plugins before the broker stopped, clean routes when a node down

emqttd#790: Unable to start emqttd service if username/password contains special characters

emq-auth-clientid#4: Improve the configuration of emq_auth_clientid.conf to resolve emqttd#790

emq-auth-username#4: Improve the configuration of emq_auth_username.conf to resolve emqttd#790

15.39 2.0.1 版本

发布日期: *2016-11-30*

emqttd#781: 更新项目README到2.0版本

emq_dashboard#84: 显示节点集群状态

emq_dashboard#79: 集群节点采用disc_copies存储mqtt_admin表

emq_auth_clientid: 集群节点采用disc_copies存储mqtt_auth_clientid表

emq_auth_username: 集群节点采用disc_copies存储mqtt_auth_username表

emq_mod_subscription#3: 删除emq_mod_subscription表与module.subscription.backend配置

emq_plugin_template#5: 插件停止时注销认证/ACL模块

15.40 2.0 正式版 “西湖以西”

发布日期: *2016-11-24*

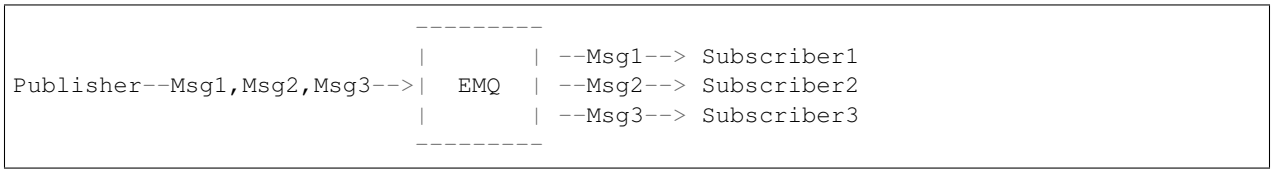
版本别名: 西湖以西*(West of West Lake)*

EMQ-2.0版本正式发布！EMQ-1.0版本产品环境下已支持900K并发连接，EMQ-2.0版本重构了整个项目架构并正式支持共享订阅功能:

1. 支持共享订阅(Shared Subscription)与本地订阅(Local Subscription)，解决MQTT协议负载平衡消费问题;
2. 支持CoAP(RFC 7252)、MQTT-SN协议和网关，支持CoAP、MQTT-SN客户端与MQTT客户端互通;
3. 重构配置文件格式与加载方式，支持用户友好的’K = V’文件格式，支持操作系统环境变量;
4. 增加了扩展钩子和大量的认证插件，支持与大部分数据库或NoSQL的认证集成;
5. 支持全平台编译部署，Linux/Unix/Windows以及ARM平台网关，支持Docker镜像制作。

15.40.1 共享订阅(Shared Subscription)

共享订阅(Shared Subscription)支持在多订阅者间采用分组负载平衡方式派发消息:



使用方式: 订阅者在主题(Topic)前增加’$queue’或’$share/<group>/’前缀。

15.40.2 本地订阅(Local Subscription)

本地订阅(Local Subscription)只在本节点创建订阅与路由表，不会在集群节点间广播全局路由，非常适合物联网数据采集应用。

使用方式: 订阅者在主题(Topic)前增加’$local/’前缀。

15.40.3 erlang.mk与relx

2.0版本分离 emqttd 主项目和发布项目 emq-relx, 采用 erlang.mk 和 relx 编译发布工具替换1.x版本使用的rebar，项目可以跨平台在Linux/Unix/Windows系统下编译。

15.40.4 CoAP协议支持

2.0版本支持CoAP协议(RFC7252)，支持CoAP网关与MQTT客户端互通。

CoAP插件: https://github.com/emqtt/emq_coap

15.40.5 MQTT-SN协议支持

2.0版本支持MQTT-SN协议，支持MQTT-SN网关与MQTT客户端互通。

MQTT-SN插件: https://github.com/emqtt/emq_sn

15.40.6 'K = V'格式配置文件

2.0版本支持用户友好的'K = V'格式配置文件etc/emq.conf:

```
node.name = emqttd@127.0.0.1

...

mqtt.listener.tcp = 1883

...
```

15.40.7 操作系统环境变量

2.0版本支持操作系统环境变量。启动时通过环境变量设置EMQ节点名称、安全Cookie以及TCP端口号:

```
EMQ_NODE_NAME=emqttd@127.0.0.1
EMQ_NODE_COOKIE=emq_dist_cookie
EMQ_MAX_PORTS=65536
EMQ_TCP_PORT=1883
EMQ_SSL_PORT=8883
EMQ_HTTP_PORT=8083
EMQ_HTTPS_PORT=8084
```

15.40.8 Docker镜像支持

EMQ-2.0版本支持Docker镜像制作，Dockerfile开源在: https://github.com/emqtt/emq_docker

15.40.9 Windows平台支持

2.0版本完整支持Windows平台的编译、发布与运行，支持Windows平台下的'emqttd_ctl'控制命令，支持在Windows节点间的集群。

15.40.10 问题与改进

#764: add mqtt.cache_acl option

#667: Configuring emqttd from environment variables

#722: *mqtt/superuser* calls two times *emqtt_auth_http*

#754: "-heart" option for EMQ 2.0

#741: emq_auth_redis cannot use hostname as server address

15.40.11 扩展插件

2.0版本发布的认证与扩展插件列表:

插件	说明
emq_dashboard	Web控制台插件(默认加载)
emq_auth_clientid	ClientId认证插件
emq_auth_username	用户名、密码认证插件
emq_auth_ldap	LDAP认证/访问控制
emq_auth_http	HTTP认证/访问控制
emq_auth_mysql	MySQL认证/访问控制
emq_auth_pgsql	PostgreSQL认证/访问控制
emq_auth_redis	Redis认证/访问控制
emq_auth_mongo	MongoDB认证/访问控制
emq_mod_rewrite	重写主题(Topic)插件
emq_mod_retainer	Retain消息存储模块
emq_mod_presence	客户端上下线状态消息发布
emq_mod_subscription	客户端上线自动主题订阅
emq_coap	CoAP协议支持
emq_sn	MQTT-SN协议支持
emq_stomp	Stomp协议支持
emq_sockjs	Stomp over SockJS协议支持
emq_recon	Recon性能调试
emq_reloader	Reloader代码热加载插件
emq_plugin_template	插件开发模版

15.41 2.0-rc.3 版本

15.42 2.0-rc.3 版本

发布日期*: 2016-11-01*

1. 将Presence、Retainer、Subscription三个扩展模块改为独立插件:

emq_mod_retainer	Retain消息存储模块
emq_mod_presence	客户端上下线状态消息发布
emq_mod_subscription	客户端上线自动主题订阅

2. 更新EMQ自带的自签名SSL证书，修复SSL双向认证配置文件错误
3. Bugfix: Fixed a typo (#716)
4. Bugfix: emqttd_http can not use emq_auth_http? #739
5. Bugfix: emq_auth_redis cannot use hostname as server address (#741)
6. 升级Redis, MySQL, Postgre, MongoDB插件，支持主机名或域名配置

15.43 2.0-rc.2 版本

发布日期*: 2016-10-19*

1. 集成cuttlefish库，支持'K = V'通用配置文件格式，重构EMQ与全部插件配置文件:

```
node.name = emqttd@127.0.0.1

...

mqtt.listener.tcp = 1883

...
```

2. 支持操作系统环境变量。启动时通过环境变量设置EMQ节点名称、Cookie以及TCP端口号:

```
EMQ_NODE_NAME
EMQ_NODE_COOKIE
EMQ_MAX_PORTS
EMQ_TCP_PORT
EMQ_SSL_PORT
EMQ_HTTP_PORT
EMQ_HTTPS_PORT
```

3. 重构认证模块、ACL模块与扩展模块，更新全部插件项目名称以及配置文件。

TODO: issues closed.

15.44 2.0-rc.1 版本

发布日期*: 2016-10-03*

1. 超级用户认证成功后，发布订阅时不进行ACL鉴权 (#696)
2. MQTT客户端认证失败后，EMQ服务器主动关闭TCP连接 (#707)
3. 改进插件管理设计，新增插件无需修改rel/sys.config配置
4. 改进全部插件Makefile的emqttd依赖:

```
BUILD_DEPS = emqttd
dep_emqttd = git https://github.com/emqtt/emqttd emq20
```

5. 重新设计Redis插件的ACL鉴权模块

15.45 2.0-beta.3 版本

发布日期*: 2016-09-18*

15.45.1 共享订阅(Shared Subscription)

Shared Suscriptions (#639, #416):

```
mosquitto_sub -t '$queue/topic'
mosquitto_sub -t '$share/group/topic'
```

15.45.2 本地订阅(Local Subscription)

Local Subscriptions that will not create global routes:

```
mosquitto_sub -t '$local/topic'
```

15.45.3 问题修复

Error on Loading *emqttd_auth_http* (#691)

Remove 'emqttd' application from dependencies (emqttd_coap PR#3)

15.46 2.0-beta.2 版本

发布日期*: 2016-09-10*

15.46.1 CoAP协议支持

CoAP协议支持插件(Beta): https://github.com/emqtt/emqttd_coap

15.46.2 API Breaking Changes

'$u', '$c' variables in emqttd.conf and modules/acl.conf changed to '%u', '%c'

Improve the design of mqtt retained message, replace emqttd_retainer with emqttd_mod_retainer.

Add 'session.subscribed', 'session.unsubscribed' hooks, remove 'client.subscribe.after' hook

Tab 'retained_message' -> 'mqtt_retained'

15.46.3 Bugfix

[2.0 beta1] FORMAT ERROR: "~s PUBLISH to ~s: ~p" (PR #671)

Fixing issues in cluster mode. (PR #681)

Fixing issues with unsubscribe hook (PR #673)

15.47 2.0-beta.1 版本

发布日期*: 2016-08-30*

版本别名*:* 西湖以西*(West of West Lake)*

EMQ 2.0-beta1预览版本(Preview Release)发布。EMQ 2.0版本改进了项目结构、发布方式、Git分支结构以及配置文件格式，以奠定EMQ消息服务器项目长期演进基础。

注解: 1.x版本产品部署用户请勿升级到该版本，2.0正式版本发布前会有API变更。

15.47.1 项目简称 - EMQ

项目简称变更为EMQ(Erlang/Enterprise/Elastic MQTT Broker)，E含义Erlang/OTP平台、企业(Enterprise)、弹性(Elastic)。

15.47.2 项目发布方式

2.0 版本后采用预览版(Preview Release) + 候选版本(Release Candidate)版本方式迭代发布，2.0版本将陆续发布beta1, beta2, beta3, rc1, rc2等迭代，直到2.0正式版本发布。

15.47.3 应用与发布

2.0 版本后 emqttd 项目只包括消息服务器应用源码，分离发布(rel)为独立项目: emqttd_relx ，以解决1.0版本的插件(plugins)与emqttd应用编译依赖问题。

源码编译请clone emqttd_relx:

```
git clone https://github.com/emqtt/emqttd-relx.git

cd emqttd-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

15.47.4 erlang.mk与relx

2.0 版本发布项目 emqttd_relx 采用 erlang.mk 和 relx 编译发布工具替换1.x版本使用的rebar。原因: https://erlang.mk/guide/why.html

15.47.5 Git分支结构

stable	1.x 稳定版本分支
master	2.x 主版本分支
emq10	1.x 版本开发分支
emq20	2.x 版本开发分支
emq30	3.x 版本开发分支
issue#{id}	Issue修复分支

etc/emqttd.conf配置文件 ———=————-

2.0 版本改进项目配置文件格式，采用rebar.config、relx.config类似格式，提高配置文件的可读性和可编辑性。

etc/emqttd.conf配置示例:

```
%% Max ClientId Length Allowed.
{mqtt_max_clientid_len, 512}.

%% Max Packet Size Allowed, 64K by default.
{mqtt_max_packet_size, 65536}.
```

(continues on next page)

(续上页)

```
%% Client Idle Timeout.
{mqtt_client_idle_timeout, 30}. % Second
```

15.47.6 MQTT-SN协议支持

2.0-beta1版本正式发布 emqttd_sn 项目支持MQTT-SN协议，插件加载方式启用emqttd_sn项目，MQTT-SN默认UDP端口: 1884:

```
./bin/emqttd_ctl plugins load emqttd_sn
```

15.47.7 改进插件架构

2.0 版本从emqttd项目删除plugins/目录，插件作为一个普通的Erlang应用，直接依赖(deps)方式在编译到lib目录，插件配置文件统一放置在etc/plugins/目录中:

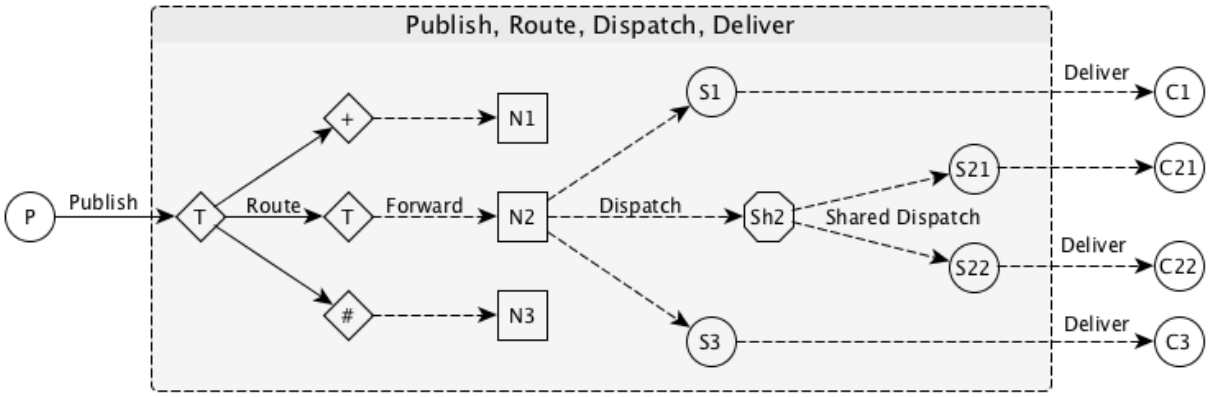
```
emqttd-relx/
  etc/
    modules/
    plugins/
       emqtt_coap.conf
       emqttd.conf
       emqttd_auth_http.conf
       emqttd_auth_mongo.conf
       emqttd_auth_mysql.conf
       emqttd_auth_pgsql.conf
       emqttd_auth_redis.conf
       emqttd_coap.conf
       emqttd_dashboard.conf
       emqttd_plugin_template.conf
       emqttd_recon.conf
       emqttd_reloader.conf
       emqttd_sn.conf
       emqttd_stomp.conf
```

15.47.8 2.0 版本项目文档

2.0 版本中文文档: http://emqtt.com/docs/v2/index.html 或 http://docs.emqtt.cn/zh_CN/emq20

2.0 版本英文文档: http://emqtt.io/docs/v2/index.html 或 http://docs.emqtt.com/

15.47.9 发布订阅流程



15.48 1.1.3 版本

发布日期*: 2016-08-19*

Support './bin/emqttd_ctl users list' CLI (#621)

Cannot publish payloads with a size of the order 64K using WebSockets (#643)

Optimize the procedures that retrieve the Broker version and Borker description in the tick timer (PR#627)

Fix SSL certfile, keyfile config (#651)

15.49 1.1.2 版本

发布日期*: 2016-06-30*

Upgrade mysql-otp driver to 1.2.0 (#564, #523, #586, #596)

Fix WebSocket Client Leak (PR #612)

java.io.EOFException using paho java client (#551)

Send message from paho java client to javascript client (#552)

Compatible with the Qos0 PUBREL packet (#575)

Empty clientId with non-clean session accepted (#599)

Update docs to fix typos (#601, #607)

15.50 1.1.1 版本

发布日期*: 2016-06-04*

Compatible with the Qos0 PUBREL packet (#575)

phpMqtt Client Compatibility (#572)

java.io.EOFException using paho java client (#551)

15.51 1.1 版本

发布日期*: 2016-06-01*

1.1版本升级eSockd库到4.0，支持IPv6与监听特定IP地址。新增MongoDB认证插件、HTTP认证插件与Reloader插件。升级MySQL、PostgreSQL、Redis认证插件，采用参数化查询避免SQL注入，并支持超级用户(superuser)认证。

15.51.1 问题与改进

Allow human-friendly IP addresses (PR#395)

File operation error: emfile (#445)

emqttd_plugin_mongo not found in emqttd (#489)

emqttd_plugin_mongo Error While Loading in emqttd (#505)

Feature request: HTTP Authentication (#541)

Compatible with the Qos0 PUBREL packet (#575)

Bugfix: function_clause exception occurs when registering a duplicated authentication module (#542)

Bugfix: ./emqttd_top msg_q result: {"init terminating in do_boot",{undef,[{etop,start,[],[]},{init,start_it,1,[]},{init,start_em,1,[]}]}} (#557)

15.51.2 Dashboard插件

WebSocket连接页面支持Clean Session, Qos, Retained参数设置 (emqttd_dashboard#52)

升级eSockd库到4.0版本，Overview页面显示OTP版本 (emqttd_dashboard#61)

Changing dashboard credentials for username authentication (emqttd_dashboard#56)

新增'./bin/emqttd_ctl admins'管理命令，支持通过命令行重新设置admin密码

15.51.3 HTTP认证插件

支持通过HTTP API认证/鉴权MQTT客户端: https://github.com/emqtt/emqttd_auth_http

15.51.4 MongoDB认证插件

升级Erlang Mongodb驱动到v1.0.0 (emqttd_plugin_mongo#1)

支持超级用户认证

支持基于MongoDB的ACL (emqttd_plugin_mongo#3)

15.51.5 MySQL认证插件

支持超级用户认证

采用参数化查询避免SQL注入

15.51.6 Postgre认证插件

支持超级用户认证

采用参数化查询避免SQL注入

15.51.7 Redis认证插件

支持超级用户认证

支持ClientId认证/ACL (emqttd_plugin_redis#4)

15.51.8 Reloader插件

开发调试代码热升级插件: https://github.com/emqtt/emqttd_reloader

15.52 1.0.2 版本

发布日期*: 2016-05-04*

Issue#534 - './bin/emqttd_ctl vm' - add 'port/count', 'port/limit' statistics

Issue#535 - emqttd_client should be terminated properly even if exception happened when sending data

PR#519 - The erlang '-name' requires the fully qualified host name

emqttd_reloader plugin - help reload modified modules during development.

15.53 1.0.1 版本

发布日期*: 2016-04-16*

PR#515 - Fix '$queue' pubsub, add 'pubsub_queue' test and update docs

15.54 1.0 (七英里) 版本

发布日期*: 2016-04-13*

版本别名*:* 七英里*(The Seven Mile Journey)*

经过两年开发，五十个版本迭代，我们正式发布1.0(七英里)版本，和完整的中英文项目文档。

1.0版本基本实现了设计目标: 稳定承载来自移动互联网或物联网终端的大量并发MQTT连接，并实现在大数量的终端间快速低延时的MQTT消息路由。

1. 完整支持MQTT V3.1.1协议，扩展支持WebSocket、Stomp或私有TCP等多协议。
2. 稳定承载大规模的并发MQTT客户端连接，单服务器节点支持50万到100万连接。
3. 分布式节点集群或桥接，快速低延时的消息路由，单集群支持1000万规模的路由。
4. 支持消息服务器内扩展，支持定制多种认证方式，插件方式存储消息到后端数据库。

15.54.1 问题与改进

1.0版本主要发布完整项目文档，相比0.17.1版本很少代码变更:

Possible race condition using emqttd_cm (#486)

Improve the design of retained message expiration (#503)

Should not expire the retained messages from $SYS/# topics (#500)

15.54.2 项目文档

1.0 版本中文文档: http://emqtt.com/docs/ 或 http://docs.emqtt.cn

1.0 版本英文文档: http://emqtt.io/docs 或 http://docs.emqtt.com/

15.54.3 官方站点

中文站点: http://emqtt.com

英文站点: http://emqtt.io/

15.54.4 致谢

爱立信与Erlang/OTP语言平台团队(http://www.erlang.org/)!

贡献者(GitHub帐户): @callbay @lsxredrain @hejin1026 @desoulter @turtleDeng @Hades32 @huangdan @phanimahesh @dvliman @Prots @joaohf

公司: 开源中国，鲁能电力，太极计算机，电信天翼云直播，研色科技，杭州华思

乐队: 七英里(The Seven Mile Journey)，腰乐队，万能青年旅店

15.55 0.17.1-beta 版本

发布日期*: 2016-03-22*

15.55.1 Enhancements

Time unit of session 'expired_after' changed to minute. (#479)

15.55.2 Dashboard

Code Review and improve the design of Dashboard.

15.56 0.17.0-beta 版本

发布日期*: 2016-03-15*

15.56.1 Highlights

Installation and Configuration Guide released on http://docs.emqtt.com

Improve and Consolidate the design of Hook, Server, PubSub and Router

Upgrade the [Web Dashboard](https://github.com/emqtt/emqttd_dashboard) to support pagination

Bridge emqttd broker to another emqttd broker & emqttd to mosquitto bridge (#438)

15.56.2 Enhancements

emqttd_ctl: better error message (#450)

./bin/emqttd_ctl: add 'routes' command

```
` routes list # List all routes routes show <Topic> # Show a route `
```

Add 'backend_subscription' table and support static subscriptions (emqttd_backend)

Add 'retained_message' table and refactor emqttd_retainer module (emqttd_backend)

A New Hook and Callback Design (emqttd_hook)

Add PubSub, Hooks APIs to emqttd module (emqttd)

Move start_listeners/0, stop_listeners/0 APIs to emqttd_app module (emqttd_app)

15.56.3 Tests

Add 100+ common test cases.

15.56.4 Plugins

Upgrade Dashboard, Redis, Stomp and Template Plugins

15.57 0.16.0-beta 版本

发布日期*: 2016-02-16*

15.57.1 Highlights

Licensed under the Apache License, Version 2.0 Now.

Improve the design of cluster, support to join or leave the cluster (#449):

```
` $ ./bin/emqttd_ctl cluster cluster join <Node> #Join the cluster cluster
leave #Leave the cluster cluster remove <Node> #Remove the node from cluster
cluster status #Cluster status `
```

Improve the design of Trie and Route, only the wildcard topics stored in Trie.

Common Test to replace EUnit.

15.57.2 Enhancements

mqtt_message record: add 'sender' field (#440)

refactor the emqttd, emqttd_time, emqttd_opts, emqttd_node modules.

15.57.3 Bugfix

noproc error when call to gen_server2:call(false, {add_route,Topic,<0.685.0>}, infinity) (#446)

Plugins

Changed the license of all plugins.

15.58 0.15.0-beta 版本

发布日期*: 2016-01-31*

15.58.1 Highlights

Optimize for Push Application, 500K+ Subscribers to a Topic.

Optimization for Route ETS insertion (#427)

Priority Message Queue for Persistent Session (#432)

Add Redis, MongoDB Plugins (#417)

15.58.2 Enhancements

Username/Password Authentication: Support to configure default users (#428)

Improve CLI Commands: pubsub, bridges, trace (#429)

emqttd_mod_subscription: fix client_connected/3

emqttd_auth_mod: add passwd_hash/2 function

priority_queue: add plen/2, out/2 functions

15.58.3 Bugfix

Fix dequeue/1 of emqttd_bridge...

Add emqttd:seed_now/0 function

15.58.4 Plugins

emqttd_plubin_mysql: Changed mysql driver to mysql-otp

emqttd_plugin_pgsql: Integrate with ecpool

emqttd_plugin_redis: First release

emqttd_plugin_mongo: First release

15.59 0.14.1-beta 版本

发布日期: *2015-12-28*

Bugfix: emqttd_ws_client.erl: Unexpected Info: {'EXIT',<0.27792.18>,{shutdown,destroy}} (#413)

Improve: fix spec errors found by dialyzer

15.60 0.14.0-beta 版本

发布日期: *2015-12-18*

15.60.1 Highlights

Scaling to 1.3 Million Concurrent MQTT Connections on a 12 Core, 32G CentOS server.

New PubSub, Router Design (#402). Prepare for scaling to 10 millions on one cluster.

15.60.2 Enhancements

Improve the gproc_pool usage with a general emqttd_pool_sup

Improve the design of emqttd_pubsub, add a new emqttd_router module

Improve the design of the whole supervisor tree

Route aging mechanism to remove the topics that have no subscriptions

Improve the dashboard, mysql, pgsql, stomp, sockjs plugins

Add 'topics', 'subscriptions' admin commands

Avoid using mnesia table index and mnesia:index_read API to lower CPU usage

Subscribe timeout exception (#366)

Long Delay on Multiple Topic Subscription (#365)

Subscriptions persistence (#344)

emqttd_ctl: 'subscriptions' command to force clients to subscribe some topics (#361)

15.60.3 Bugfix

emqttd_sm: spec of lookup_session/1 is not right BUG (#411)

Observer application should be removed from reltool.config for 'wx' app is not available (#410)

15.60.4 Benchmark

1.3 million concurrent MQTT connections on a 12 Core, 32G CentOS Server, consume about 15G Memory and 200% CPU.

15.61 0.13.1-beta 版本

发布日期: *2015-11-28*

Bugfix: Plugin pathes error under windows (#387)

Improve: Too many error logs "[error] Session ..... Unexpected EXIT: client_pid=<0.14137.35>, exit_pid=<0.30829.22>, reason=nop..." (#383)

Improve: Define QOS0/1/2, Pooler Error (PR#382)

Improve: High CPU load when 400K unstable mobile connections (#377)

BugFix: emqttd_plugin_pgsql - error using same query with latest update plugin (pgsql#5)

15.62 0.13.0-beta 版本

发布日期: *2015-11-08*

15.62.1 Highlights

Rate Limiting based on [Token Bucket](https://en.wikipedia.org/wiki/Token_bucket) and [Leaky Bucket](https://en.wikipedia.org/wiki/Leaky_bucket#The_Leaky_Bucket_Algorithm_as_a_Meter) Algorithm

Upgrade eSockd and MochiWeb libraries to support Parameterized Connection Module

Improve emqttd_client to support fully asynchronous socket networking

15.62.2 Enhancements

Protocol Compliant - Session Present Flag (#163)

Compilation fails if repo is cloned with a different name (#348)

emqttd_client: replace gen_tcp:send with port_command (#358)

TCP sndbuf, recbuf, buffer tuning (#359)

emqttd_client.erl to handle 'inet_async', 'inet_reply' properly (#360)

Refator the [client/session management design](https://github.com/emqtt/emqttd/blob/master/doc/design/ClientSession.md)

15.62.3 Bugfix

Cannot kick transient client out when clientId collision (#357)

Fix the order of emqttd_app:start_server/1 (#367)

emqttd_session:subscribe/2 will crash (#374)

15.62.4 Benchmark

[benchmark for 0.13.0 release](https://github.com/emqtt/emqttd/wiki/benchmark-for-0.13.0-release)

3.1G memory and 50+ CPU/core:

```
Connections: 250K
Subscribers: 250K
Topics:      50K
Qos1 Messages/Sec In:  4K
Qos1 Messages/Sec Out: 20K
Traffic In(bps):  12M+
Traffic Out(bps): 56M+
```

15.63 0.12.3-beta 版本

发布日期*: 2015-10-22*

Bugfix: emqttd_sysmon crasher for 'undefined' process_info (#350)

Bugfix: emqttd_client: catch parser exception (#353)

15.64 0.12.2-beta 版本

发布日期*: 2015-10-16*

Bugfix: Retained messages should not be expired if 'broker.retained.expired_after = 0' (#346)

15.65 0.12.1-beta 版本

发布日期*: 2015-10-15*

Highlight: Release for Bugfix and Code Refactor.

Feature: Retained message expiration (#182)

Improve: '$SYS/#' publish will not match '#' or '+/#' (#68)

Improve: Add more metrics and ignore '$SYS/#' publish (#266)

Improve: emqttd_sm should be optimized for clustered nodes may be crashed (#282)

Improve: Refactor emqttd_sysmon and suppress 'monitor' messages (#328)

Task: benchmark for 0.12.0 release (#225)

Benchmark: About 900K concurrent connections established on a 20Core, 32G CentOS server.

15.66 0.12.0-beta 版本

发布日期*: 2015-10-08*

15.66.1 Highlights

Enhance the **emqttd_ctl** module to allow plugins to register new commands (#256)

Add [emqttd_recon plugin](https://github.com/emqtt/emqttd_recon) to debug/optimize the broker (#235)

Add **'./bin/emqttd_ctl broker pubsub'** command to check the status of core pubsub processes

Add **'./bin/emqttd_top'** command(like etop) to show the top 'msg_q', 'reductions', 'memory' or 'runtime' processes

'rel/files/emqttd.config.production' for production deployment(default)

'rel/files/emqttd.config.development' for development deployment

15.66.2 Enhancements

Qos1/2 messages will not be dropped under unstable mobile network (#264)

emqttd_session:subscribe/2, emqttd_session:unsubscribe/2 APIs should be asynchronous (#292)

etc/emqttd.config: 'idle_timeout' option to close the idle client(socket connected but no 'CONNECT' frame received)

etc/emqttd.config: 'unack_retry_interval' option for redelivering Qos1/2 messages

How to monitor large 'message_queue_len' (#283)

15.66.3 Bugfix

Behaviour emqttd_auth_mod is missing init callback (#318)

15.66.4 Benchmark

Write a new [benchmark tool](https://github.com/emqtt/emqtt_benchmark) to benchmark this release

Hw requirements - 5K users, 25-50 msgs/sec, QoS=1 (#209)

Supported Number of Connections Greatly Reduced When Clients are Subscribing (#324)

15.67 0.11.0-beta 版本

发布日期*: 2015-09-25*

Highlight: Rebar to manage plugin dependencies.

Highlight: [Stomp](https://github.com/emqtt/emqttd_stomp) and [SockJS](https://github.com/emqtt/emqttd_sockjs) Plugins!

Improve: add rel/files/emqttd.config.development|production.

Improve: rel/reltool.config.script to release deps of plugin.

Improve: persist mnesia schema on slave nodes.

Improve: use timer:seconds/1 api.

Improve: The binary release will be compiled with R18.1 now.

Bugfix: issue#306 - emqttd_cm should unregister the duplicated client

Bugfix: issue#310 - usage of emqttd_ctl error: 'session list' should be 'sessions list'

Bugfix: issue#311 - './bin/emqttd_ctl sessions list' error

Bugfix: issue#312 - unsubcribe will lead to crash if emqttd_plugin_template plugin loaded

15.68 0.10.4-beta 版本

发布日期: *2015-09-18*

Optimize session management and upgrade eSockd library to 2.7.1

[Benchmark for 0.10.4 release](https://github.com/emqtt/emqttd/wiki/benchmark-for-0.10.4-release)

Improve: issue#294 - [error] failed to start connection on 0.0.0.0:1883 - enotconn

Improve: issue#297 - How do I allow user with some pattern to access topic with some pattern?

Bugfix: issue#291 - "./bin/emqttd attach ..." cannot work

Bugfix: issue#284 - Should not use erlang:list_to_atom/1 in emqttd_vm.erl

15.69 0.10.3-beta 版本

发布日期: *2015-08-30*

Bugfix: issue#271 - add emqttd_ws_client:subscribe/2 function

Bugfix: issue#269 - bin/emqttd Syntax error on ubuntu

Improve: issue#265 - client under unstable mobile network generate a lot of logs

15.70 0.10.2-beta 版本

发布日期: *2015-08-26*

Improve: issue#257 - After the node name changed, the broker cannot restart for mnesia schema error.

15.71 0.10.1-beta 版本

发布日期: *2015-08-25*

Bugfix: issue#259 - when clustered the emqttd_dashboard port is close, and the 'emqttd' application cannot stop normally.

Feature: issue#262 - Add 'http://host:8083/mqtt/status' Page for health check

15.72 0.10.0-beta 版本

发布日期: *2015-08-20*

[Web Dashboard](https://github.com/emqtt/emqttd_dashboard) and [MySQL](https://github.com/emqtt/emqttd_plugin_mysql), [PostgreSQL](https://github.com/emqtt/emqttd_plugin_pgsql) Authentication/ACL Plugins!

Highlight: Web Dashboard to monitor Statistics, Metrics, Clients, Sessions and Topics of the broker.

Highlight: JSON/HTTP API to query all clients connected to broker.

Highlight: A new [Plugin Design](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) and a [Template project](https://github.com/emqtt/emqttd_plugin_template) for plugin development.

Highlight: Authentication/ACL with MySQL, PostreSQl databases (#194, #172)

Feature: Session Statistics including inflight_queue, message_queue, message_dropped, awaiting_rel, awaiting_ack, awaiting_comp (#213)

Feature: Cookie based authentication for MQTT over websocket connections (#231)

Feature: Get all clients connected to the broker (#228, #230, #148, #129)

Feature: "./bin/emqttd_ctl clients show ClientId" to query client status (#226)

Feature: "./bin/emqttd_ctl clients kick ClientId" to kick out a client

Feature: "./bin/emqttd_ctl sessions list" to show all sessions

Feature: "./bin/emqttd_ctl sessions show ClientId" to show a session

Feature: Erlang VM metrics monitor with Web Dashboard (#59)

Improve: Too many "inflight queue is full!" log when session is overloaded (#247)

Improve: There are two many "MQueue(~s) drop ~s" logs if the message queue of session is small (#244)

Improve: gen_server2(from RabbitMQ) to improve emqttd_session, emqttd_pubsub

Improve: Makefile to build plugins

Bugfix: emqttd_broker:unhook/2 cannot work (#238)

Bugfix: emqttd plugin cannot include_lib("emqttd/include/emqttd.hrl") (#233)

Bugfix: Too many 'Session ~s cannot find PUBACK' logs (#212)

Bugfix: emqttd_pooler cannot work

15.73 0.9.3-alpha 版本

发布日期*: 2015-07-25*

Wiki: [Bridge](https://github.com/emqtt/emqttd/wiki/Bridge)

Improve: emqttd_protocol.hrl to define 'QOS_I'

Improve: emqttd_pubsub to add subscribe/2 API

Improve: ./bin/emqttd_ctl to support new bridges command

Bugfix: issue #206 - Cannot bridge two nodes

15.74 0.9.2-alpha 版本

发布日期: *2015-07-18*

Improve: issue #196 - Add New Hook 'client.subscribe.after'

15.75 0.9.1-alpha 版本

发布日期: *2015-07-10*

Bugfix: issue #189 - MQTT over WebSocket(SSL) cannot work?

Bugfix: issue #193 - 'client.ack' hook should be renamed to 'message.acked', and called by emqttd_broker:foreach_hooks

15.76 0.9.0-alpha 版本

发布日期: *2015-07-09*

[Session, Queue, Inflight Window, Hooks, Global MessageId and More Protocol Compliant](https://github.com/emqtt/emqttd/releases/tag/0.9.0-alpha) Now!

Feature: Session/Queue/Inflight Window Design (#145).

Feature: Support to resume a persistent session on other clustered node.

Feature: Support alarm management.

Feature: emqttd_guid to generate global unique message id.

Feature: Hooks for message pub/ack.

Feature: Protocol compliant - message ordering, timeout and retry.

Improve: Every client will start_link a session process, whether or not the client is persistent.

Improve: etc/emqttd.config to support more session, queue configuration.

Improve: issue #179 - Max offline message queue {max_queue, 100} meaning.

Improve: issue #180 - Should change project structure for other projects maybe depend on 'emqttd'. Merge emqtt, emqttd apps.

Improve: issue #185 - PacketId and MessageId: the broker should generate global unique message id.

Improve: issue #187 - etc/emqttd.config to support https listener

Improve: issue #186 - emqttd_cm to store client details

Improve: issue #174 - add 'from' field to mqtt_message record.

Improve: issue #170 - $SYS Topics should support alarms.

Improve: issue #169 - Add More [Hooks](https://github.com/emqtt/emqttd/wiki/Hooks-Design)

Improve: issue #167 - Inflight window to assure message ordering.

Improve: issue #166 - Message delivery timeout and retry.

Improve: issue #143 - Qos1, Qos2 PubSub message timeout.

Improve: issue #122 - Labeling message with unique id. emqttd_guid module to generate global unique msgid.

Improve: emqttd_bridge to support pending message queue, and fix the wrong Qos design.

Improve: mqtt_message record to add 'msgid', 'from' and 'sys' fields.

Change: Add emqttd_mqueue, emqttd_guid, emqttd_alarm modules.

Bugfix: issue #184 - emqttd_stats:setstats is not right.

Bugfix: Closed issues #181, #119.

Tests: fix the parser, acl test cases.

15.77 0.8.6-beta 版本

发布日期*: 2015-06-17*

Bugfix: issue #175 - publish Will message when websocket is closed without 'DISCONNECT' packet

15.78 0.8.5-beta 版本

发布日期*: 2015-06-10*

Bugfix: issue #53 - client will receive duplicate messages when overlapping subscription

15.79 0.8.4-beta 版本

发布日期*: 2015-06-08*

Bugfix: issue #165 - duplicated message when publish 'retained' message to persistent client

15.80 0.8.3-beta 版本

发布日期*: 2015-06-05*

Bugfix: issue #158 - should queue:in new message after old one dropped

Bugfix: issue #155 - emqtt_parser.erl: parse_topics/3 should reverse topics

Bugfix: issue #149 - Forget to merge plugins/emqttd_auth_mysql from 'dev' branch to 'master' in 0.8.x release

15.81 0.8.2-alpha 版本

发布日期*: 2015-06-01*

Bugfix: issue #147 - WebSocket client cannot subscribe queue '$Q/queue/${clientId}'

Bugfix: issue #146 - emqttd_auth_ldap: fill(Username, UserDn) is not right

15.82 0.8.1-alpha 版本

发布日期: *2015-05-28*

Client [Presence](https://github.com/emqtt/emqttd/wiki/Presence) Support and [$SYS Topics](https://github.com/emqtt/emqttd/wiki/\protect\T1\textdollarSYS-Topics) Redesigned!

Bugfix: issue #138 - when client disconnected normally, broker will not publish disconnected $SYS message

Bugfix: fix websocket url in emqttd/priv/www/websocket.html

Improve: etc/emqttd.config to allow websocket connections from any hosts

Improve: rel/reltool.config to exclude unnecessary apps.

15.83 0.8.0-alpha 版本

发布日期: *2015-05-25*

[Hooks](https://github.com/emqtt/emqttd/wiki/Hooks%20Design), Modules and [Plugins](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) to extend the broker Now!

Plugin: emqttd_auth_mysql - MySQL authentication plugin (issues #116, #120)

Plugin: emqttd_auth_ldap - LDAP authentication plugin

Feature: emqttd_broker to support Hooks API

Feature: issue #111 - Support 'Forced Subscriptions' by emqttd_mod_autosub module

Feature: issue #126 - Support 'Rewrite rules' by emqttd_mod_rewrite module

Improve: Support hooks, modules to extend the broker

Improve: issue #76 - dialyzer check

Improve: 'Get Started', 'User Guide', 'Developer Guide' Wiki

Improve: emqtt_topic to add join/1, feed_var/3, is_queue/1

Improve: emqttd_pooler to execute common tasks

Improve: add emqttd_sm_sup module, and use 'hash' gproc_pool to manage sessions

Tests: add more test cases for 'emqttd' app

15.84 0.7.1-alpha 版本

发布日期: *2015-05-04*

Add doc/design/* and merge doc/* to github Wiki

Bugfix: issue #121 - emqttd cluster issuse

Bugfix: issue #123 - emqttd:unload_all_plugins/0 cannot unload any plugin

Bugfix: fix errors found by dialyzer

15.85 0.7.0-alpha 版本

发布日期: *2015-05-02*

[MQTT over WebSocket(SSL)](https://github.com/emqtt/emqttd/wiki/MQTT-Over-WebSocket) Now!

[Plugin Achitecture](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) based on OTP application

[Trace MQTT Packets or Messages](https://github.com/emqtt/emqttd/wiki/Trace%20Design) to log files

Feature: issue #40, #115 - WebSocket/SSL Support

Feature: issue #49, #105 - Plugin Architecture Support

Feature: issue #93 - Trace API Design

Improve: issue #109 - emqttd_broker should add subscribe, notify API

Improve: update README.md to add 'Goals', 'Contributors' chapters

Change: rename etc/app.config to etc/emqttd.config

Change: etc/emqttd.config changed

Bugfix: critical issue #54 - error when resume session!

Bugfix: issue #118 - error report when UNSUBSCRIBE with no topics

Bugfix: issue #117 - sys_interval = 0 config cannot work

Bugfix: issue #112 - Makefile to support build plugins

Bugfix: issue #96 - "make clean" cannot work

15.86 0.6.2-alpha 版本

发布日期: *2015-04-24*

Bugfix: critical issue #54, #104, #106 - error when resume session

Improve: add emqttd_cm_sup module, and use 'hash' gproc_pool to register/unregister client ids

Improve: kick old client out when session is duplicated.

Improve: move mnesia dir config from etc/app.config to etc/vm.args

15.87 0.6.1-alpha 版本

发布日期: *2015-04-20*

Integrate with [gproc library](https://github.com/uwiger/gproc) to support pool

Feature: issues#91 - should use worker_pool to handle some async work?

Feature: issues#95 - Topic filters in ACL rule should support 'eq' tag

Improve: issues#84 - emqttd_pubsub is redesigned again to protect mnesia transaction

Improve: issues#74 - ACL Support and update [ACL Design Wiki](https://github.com/emqtt/emqttd/wiki/ACL-Design)

15.88 0.6.0-alpha 版本

发布日期: *2015-04-17*

ACL Support Now: [ACL-Design Wiki](https://github.com/emqtt/emqttd/wiki/ACL-Design)

Authentication with username, clientid Now: [Authentication Wiki](https://github.com/emqtt/emqttd/wiki/Authentication)

Seperate common MQTT library to 'emqtt' application

Redesign message pubsub, route and retain modules

Redesign mnesia database cluster

Feature: issues#47 - authentication, authorization support

Feature: issues#92 - merge emqttd_acl and emqttd_auth to emqttd_access_control

Feature: emqttd_acl_mod, emqttd_auth_mod behaviour to extend ACL, authentication

Feature: issues#85 - lager:info to log subscribe, unsubscribe actions

Feature: issues#77 - authentication with clientid, ipaddress

Improve: issues#90 - fix lager_file_backend log format, and rotate 10 log files

Improve: issues#88 - use '-mneisa_create', '-mnesia_replicate' attributes to init mneisa

Improve: issues#87 - record mqtt_user and mqtt_client is duplicated

Improve: issues#81 - redesign nodes cluster to support disc_copies mnesia tables

Improve: issues#80 - redesign emqttd_cm to handle more concurrent connections

Improve: issues#70 - how to handle connection flood? Now could support 2K+ CONNECT/sec

Change: redesign mnesia tables: message, topic, subscriber, trie, trie_node

Bugfix: issues#83 - emqttd_broker stats cannot work

Bugfix: issues#75 - careless about function name when emqttd_pubsub handle getstats message

15.89 0.5.5-beta 版本

发布日期: *2015-04-09*

Bugfix: issue #75 - careless about function name when emqttd_pubsub handle getstats message.

Bugfix: issue #79 - cannot find topic_subscriber table after cluster with other nodes.

15.90 0.5.4-alpha 版本

发布日期: *2015-03-22*

Benchmark this release on a ubuntu/14.04 server with 8 cores, 32G memory from QingCloud.com:

‘‘‘ 200K Connections, 30K Messages/Sec, 20Mbps In/Out Traffic, 200K Topics, 200K Subscribers,

Consumed 7G memory, 40% CPU/core ‘‘‘

Benchmark code: https://github.com/emqtt/emqttd_benchmark

Change: rewrite emqttd_pubsub to handle more concurrent subscribe requests.

Change: ./bin/emqttd_ctl add 'stats', 'metrics' commands.

Bugfix: issue #71, #72

15.91 0.5.3-alpha 版本

发布日期*: 2015-03-19*

Bugfix: issues#72 - emqttd_cm, emqtt_sm ets:match_delete/2 with wrong pattern

15.92 0.5.2-alpha 版本

发布日期*: 2015-03-18*

Change: upgrade esockd to 2.1.0-alpha, do not tune socket buffer for mqtt connection.

15.93 0.5.1-alpha 版本

发布日期*: 2015-03-13*

Change: upgrade esockd to v1.2.0-beta, rename 'acceptor_pool' to 'acceptors'

15.94 0.5.0-alpha 版本

发布日期*: 2015-03-12*

RENAME 'emqtt' to 'emqttd'!

Support [Broker Bridge](https://github.com/emqtt/emqttd/wiki/Bridge-Design) Now!

Change: rename project from 'emqtt' to 'emqttd'

Change: lager:debug to dump RECV/SENT packets

Feature: emqttd_bridge, emqttd_bridge_sup to support broker bridge

Feature: emqtt_event to publish client connected/disconnected message to $SYS topics

Feature: ./bin/emqttd_ctl add more commands: listeners, broker, bridges, start_bridge, stop_bridge...

Feature: issue#57 - support to configure max packet size

Feature: issue#68 - if sys_interval = 0, emqttd_broker will not publish messages to $SYS/brokers/#

Bugfix: issue#67 - subscribe '#' to receive all messages

Bugfix: issue#64 - emqtt_app start/2: should wait_for_databases

Test: emqttd_topic_tests add more '_match_test'

15.95 0.4.0-alpha 版本

发布日期: *2015-03-10*

Support [$SYS Topics of Broker](https://github.com/emqtt/emqttd/wiki/\protect\T1\textdollarSYS-Topics-of-Broker) Now!

Feature: emqtt_broker to publish version, uptime, datetime to $SYS/brokers/# topics

Feature: emqtt_broker to publish count of clients, sessions, suscribers to $SYS/brokers/# topics

Feature: emqtt_metrics to publish bytes, packets, messages metrics to $SYS/brokers/# topics

Feature: add include/emqtt_systop.hrl

Change: emqtt_cm to count current clients

Change: emqtt_sm to count current sessions

Change: emqtt_pubsub to count current topics and suscribers

Change: emqtt_pubsub to add create/1 API

Change: emqtt_pubsub dispatch/2 to return number of subscribers

Change: emqtt_pubsub to count 'dropped' messages

Change: emqtt_opts to add merge/2 function

Test: add emqtt_serialiser_tests.erl

15.96 0.3.4-beta 版本

发布日期: *2015-03-08*

Bugfix: emqtt_serialiser.erl cannot serialise UNSUBACK packets

15.97 0.3.3-beta 版本

发布日期: *2015-03-07*

Bugfix: emqtt_serialiser.erl cannot serialise PINGRESP issue#60

15.98 0.3.2-beta 版本

发布日期: *2015-03-05*

Improve: merge emqttc serialiser, parser, packet

Add: emqtt_opts to merge socket options

15.99 0.3.1-beta 版本

发布日期: *2015-03-02*

Feature: SSL Socket Support

Feature: issue#44 HTTP API should add Qos parameter

Bugfix: issue#52 emqtt_session crash

Bugfix: issue#53 sslsocket keepalive error

Upgrade: esockd to v0.2.0

Upgrade: mochiweb to v3.0.0

15.100 0.3.0-beta 版本

发布日期: *2015-01-19*

Feature: HTTP POST API to support 'qos', 'retain' parameters

Feature: $SYS system topics support

Change: Rewrite emqtt_topic.erl, use '', '#', '+' to replace <<"">>, <<"#">>, <<"+">>

Change: fix emqtt_pubsub.erl to match '#', '+'

Tests: emqtt_topic_tests.erl add more test cases

15.101 0.3.0-alpha 版本

发布日期: *2015-01-08*

NOTICE: Full MQTT 3.1.1 support now!

Feature: Passed org.eclipse.paho.mqtt.testing/interoperability tests

Feature: Qos0, Qos1 and Qos2 publish and suscribe

Feature: session(clean_sess=false) management and offline messages

Feature: redeliver awaiting puback/pubrec messages(doc: Chapter 4.4)

Feature: retain messages, add emqtt_server module

Feature: MQTT 3.1.1 null client_id support

Bugfix: keepalive timeout to send will message

Improve: overlapping subscription support

Improve: add emqtt_packet:dump to dump packets

Test: passed org.eclipse.paho.mqtt.testing/interoperability

Test: simple cluster test

Closed Issues: #22, #24, #27, #28, #29, #30, #31, #32, #33, #34, #36, #37, #38, #39, #41, #42, #43

15.102 0.2.1-beta 版本

发布日期: *2015-01-08*

pull request 26: Use binaries for topic paths and fix wildcard topics

emqtt_pubsub.erl: fix wildcard topic match bug caused by binary topic in 0.2.0

Makefile: deps -> get-deps

rebar.config: fix mochiweb git url

tag emqtt release accoding to [Semantic Versioning](http://semver.org/)

max clientId length is 1024 now.

15.103 0.2.0 版本

发布日期: *2014-12-07*

rewrite the project, integrate with esockd, mochiweb

support MQTT 3.1.1

support HTTP to publish message

15.104 0.1.5 版本

发布日期: *2013-01-05*

Bugfix: remove QOS_1 match when handle PUBREL request

Bugfix: reverse word in emqtt_topic:words/1 function

15.105 0.1.4 版本

发布日期: *2013-01-04*

Bugfix: fix “mosquitto_sub -q 2 . . . . . ” bug

Bugfix: fix keep alive bug

15.106 0.1.3 版本

发布日期: *2013-01-04*

Feature: support QOS2 PUBREC, PUBREL,PUBCOMP messages

Bugfix: fix emqtt_frame to encode/decoe PUBREC/PUBREL messages

15.107 0.1.2 版本

发布日期: *2012-12-27*

Feature: release support like riak

Bugfix: use ?INFO/?ERROR to print log in tcp_listener.erl

15.108 0.1.1 版本

发布日期: *2012-09-24*

Feature: use rebar to generate release

Feature: support retained messages

Bugfix: send will msg when network error

15.109 0.1.0 版本

发布日期: *2012-09-21*

The first public release.

CHAPTER 16

版本升级 (Upgrade)

16.1 2.0升级到2.0.3版本

升级流程:

1. 下载解压2.0.3版本到新安装目录，例如 /opt/emqttd-2.0.3/;
2. 旧版本的 'etc/' 配置文件、'data/' 数据文件覆盖到新版目录;
3. 停止旧版本，启动新版。

16.2 升级到2.0版本

注解: 2.0版本全新设计了项目架构、配置方式与插件管理方式。1.x版本升级需要重新配置部署。

升级流程:

1. 下载解压2.0版本到新安装目录，例如 /opt/emqttd-2.0/
2. 参考旧版本 etc/vm.args、etc/emqttd.config，配置2.0版本的 etc/emq.conf
3. 重新配置插件 etc/plugins/${your-plugin}.conf
4. 编辑插件加载文件 data/loaded_plugins
5. 停止旧版本，启动新版。

16.3 升级到1.1.2版本

16.4 升级到1.1.2版本

注解: 1.0以后版本可平滑升级到1.1.2

升级流程:

1. 下载解压1.1.2版本到新安装目录，例如 /opt/emqttd_112;
2. 旧版本的 'etc/' 配置文件、'data/' 数据文件覆盖到新版目录;
3. 如果有加载插件，将旧版插件配置文件覆盖到新版;
4. 停止旧版本，启动新版。

CHAPTER 17

MQTT协议

17.1 MQTT轻量发布订阅消息协议

17.1.1 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

17.1.2 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

17.1.3 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

17.2 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
```

注解: 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。

17.3 MQTT V3.1.1协议报文

17.3.1 报文结构

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

17.3.2 固定报头

Bit	7	6	5	4	3	2	1	0
byte1	MQTT Packet type				Flags			
byte2...	Remaining Length							

17.3.3 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

17.3.4 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。

17.3.5 PINGREQ/PINGRESP心跳

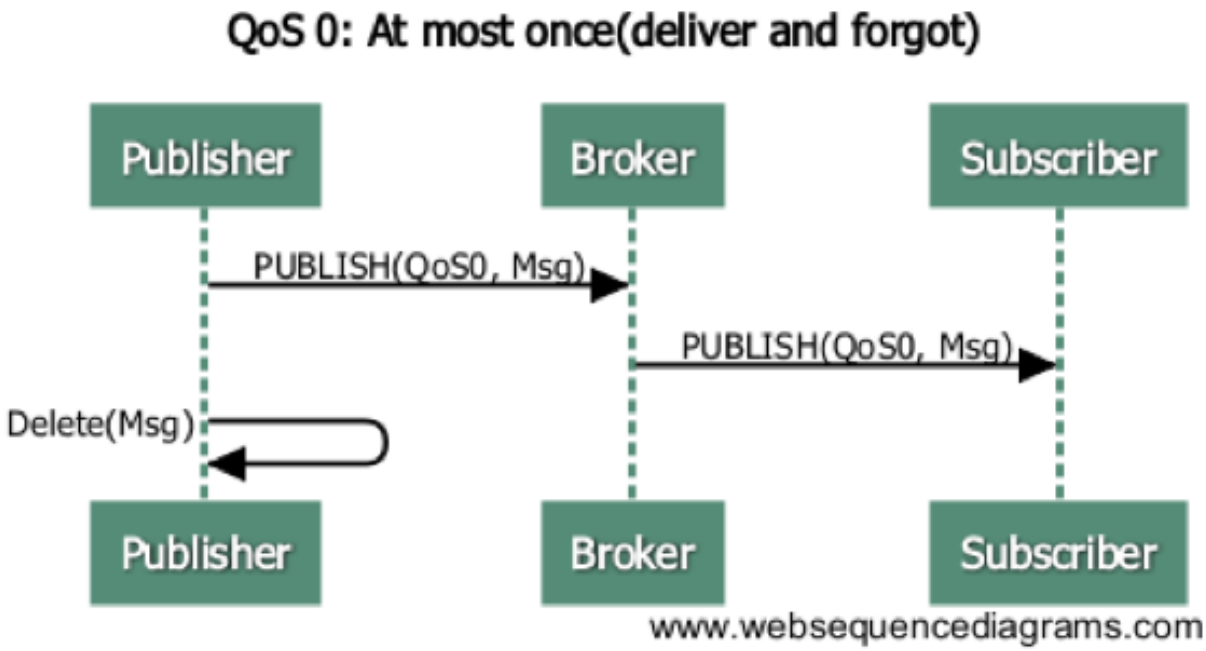
客户端在无报文发送时，按保活周期(KeepAlive)定时向服务端发送PINGREQ心跳报文，服务端响应PINGRESP报文。PINGREQ/PINGRESP报文均2个字节。

17.4 MQTT消息QoS

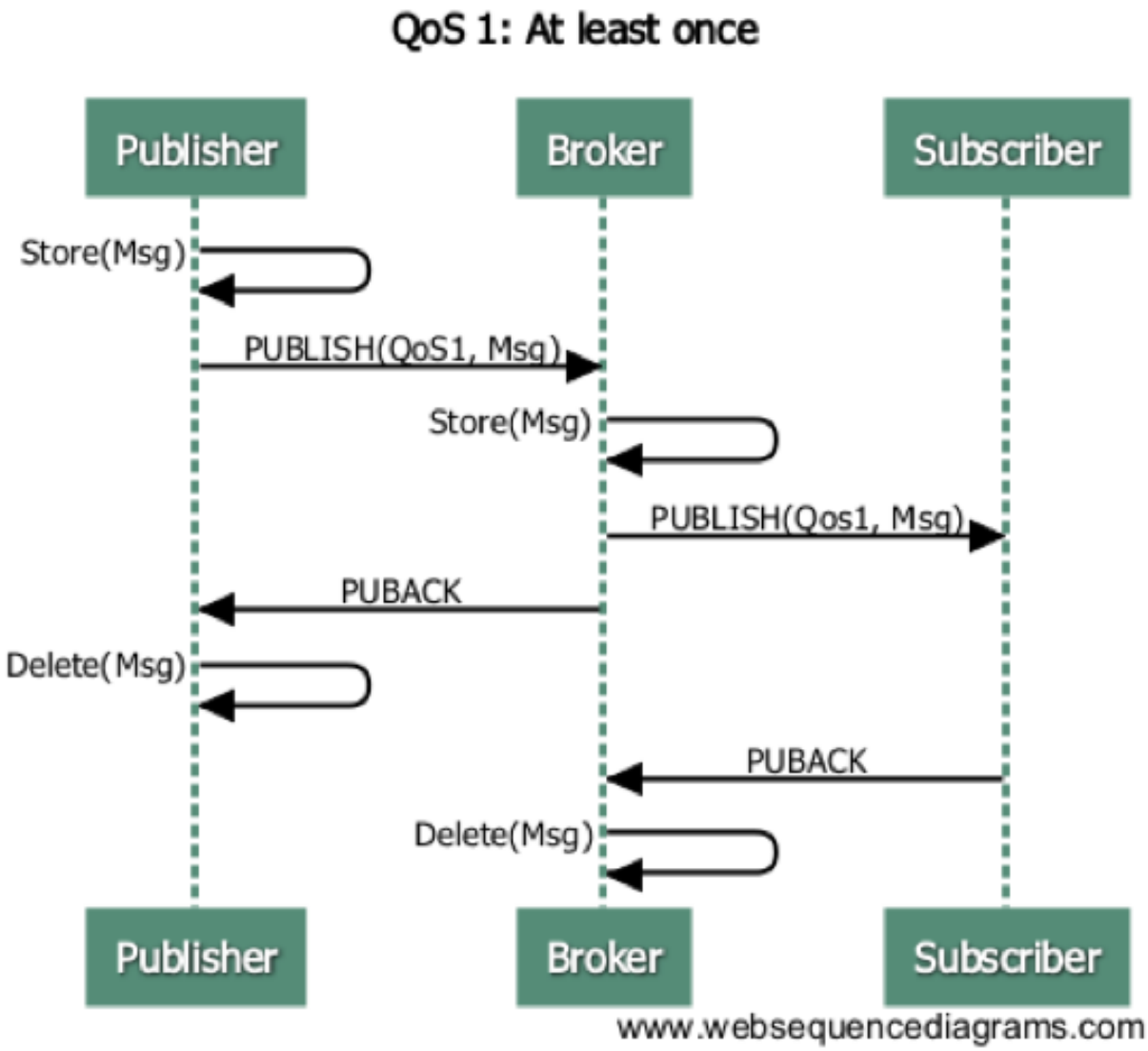
MQTT发布消息QoS保证不是端到端的，是客户端与服务器之间的。订阅者收到MQTT消息的QoS级别，最终取决于发布消息的QoS和主题订阅的QoS。

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

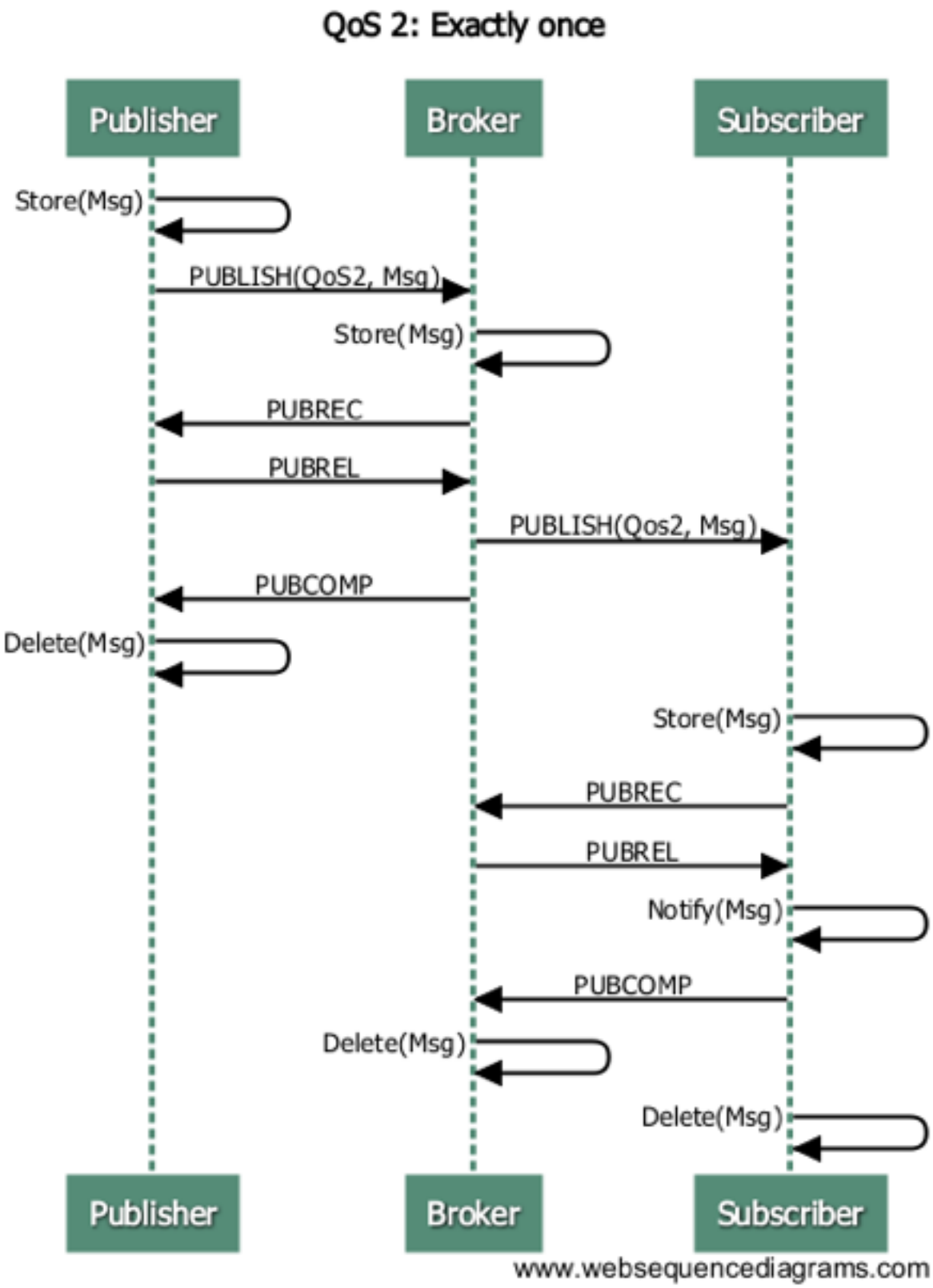
17.4.1 Qos0消息发布订阅



17.4.2 Qos1消息发布订阅



17.4.3 Qos2消息发布订阅



17.5 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。

17.6 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。

注解: emqttd消息服务器默认按最长2.5心跳周期超时设计。

17.7 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。

17.8 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 1
hello
```

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。

17.9 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
```

17.10 MQTT协议客户端库

17.10.1 emqtt客户端库

emqtt项目组: https://github.com/emqtt

emqttc	Erlang MQTT客户端库
CocoaMQTT	Swift语言MQTT客户端库
QMQTT	QT框架MQTT客户端库

17.10.2 Eclipse Paho客户端库

Paho官网: http://www.eclipse.org/paho/

17.10.3 mqtt.org官网客户端库

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

17.11 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。

CHAPTER 18

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

18.1 MQTT-SN 和 MQTT 的区别

MQTT-SN 的信令和 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。

18.2 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。

18.2.1 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
```

18.2.2 启动emq-sn

18.3 MQTT-SN 客户端库

1. https://github.com/eclipse/paho.mqtt-sn.embedded-c/
2. https://github.com/ty4tw/MQTT-SN
3. https://github.com/njh/mqtt-sn-tools
4. https://github.com/arobenko/mqtt-sn

CHAPTER 19

LWM2M 协议

LWM2M 是由 Open Mobile Alliance(OMA) 定义的一套适用于物联网的协议，它提供了设备管理和通讯的功能。协议可以在 这里 下载。

LWM2M 使用 CoAP 作为底层的传输协议，承载在 UDP 或者 SMS 上。

LWM2M 定义了两种服务器

- 一种是 LWM2M BOOTSTRAP SERVER，emq-lwm2m 插件并未实现该服务器的功能。
- 一种是 LWM2M SERVER，emq-lwm2m 实现该服务器在 UDP 上的功能，SMS 并没有实现。

LWM2M 把设备上的服务抽象为 Object 和 Resource, 在 XML 文件中定义各种 Object 的属性和功能。可以在 这里 找到 XML 的各种定义。

19.1 EMQ-LWM2M 插件

EMQ-LWM2M 是 EMQ 服务器的一个网关插件，实现了 LWM2M 的大部分功能。MQTT 客户端可以通过 EMQ-LWM2M 访问支持 LWM2M 的设备。设备也可以往 EMQ-LWM2M 上报 notification，为 EMQ 后端的服务采集数据。

19.2 MQTT 和 LWM2M的转换

从 MQTT 客户端可以发送 Command 给 LWM2M 设备。MQTT 到 LWM2M 的命令使用如下的 topic

其中 MQTT Payload 是一个 json 格式的字符串，指定要发送的命令，更多的细节请参见 emq-lwm2m 的文档。

LWM2M 设备的回复用如下 topic 传送

MQTT Payload 也是一个 json 格式的字符串，更多的细节请参见 emq-lwm2m 的文档。

19.2.1 配置参数

File: etc/emq_lwm2m.conf:

```
lwm2m.port = 5783

lwm2m.certfile = etc/certs/cert.pem

lwm2m.keyfile = etc/certs/key.pem

lwm2m.xml_dir =  etc/lwm2m_xml
```

lwm2m.port	指定 lwm2m 监听的端口号，为了避免和 emq-coap 冲突，使用了非标准的5783端口
lwm2m.certfile	DTLS 使用的证书
lwm2m.keyfile	DTLS 使用的秘钥
lwm2m.xml_dir	存放 XML 文件的目录，这些 XML 用来定义 LWM2M Object

19.2.2 启动 emq-lwm2m

19.3 LWM2M 的客户端库

- https://github.com/eclipse/wakaama
- https://github.com/OpenMobileAlliance/OMA-LWM2M-DevKit
- https://github.com/AVSystem/Anjay
- http://www.eclipse.org/leshan/

EMQ 项目支持与联系:

官网:	http://emqtt.com
项目:	https://github.com/emqtt
微信:	emqttd
微博:	http://weibo.com/emqtt
Twitter:	@emqtt
作者:	李枫 <feng@emqtt.io>


EMQ