

第二章 EMQ基础功能

学习目标:

- 目标1：能够说出EMQ X Broker Dashboard各导航菜单的用途
- 目标2：能够说出EMQ X Broker 认证的方式都有哪些以及认证的流程
- 目标3：能够完成使用username认证的案例
- 目标4：能够完成使用ClientID认证的案例
- 目标5：能够说出HTTP认证的原理并且完成基于HTTP认证的案例
- 目标6：能够完成使用Eclipse Paho java进行消息收发的案例
- 目标7：能够完成使用MQTT.js进行消息收发的案例
- 目标8：能够查看EMQ X Broker的相关日志以及日志的追踪

1. Dashboard

EMQ X 提供了 Dashboard 以方便用户管理设备与监控相关指标。通过 Dashboard 可以查看服务器基本信息、负载情况和统计数据，可以查看某个客户端的连接状态等信息甚至断开其连接，也可以动态加载和卸载指定插件。除此之外，EMQ X Dashboard 还提供了规则引擎的可视化操作界面，同时集成了一个简易的 MQTT 客户端工具供用户测试使用。

EMQ X Dashboard 功能由 [emqx-dashboard](#) 插件实现，该插件默认处于启用状态，它将在 EMQ X 启动时自动加载。如果你希望禁用 Dashboard 功能，你可以将 `data/loaded_plugins` 中的 `{emqx_dashboard, true}` 修改为 `{emqx_dashboard, false}`。

1.1 查看和配置Dashboard

EMQ X Dashboard 是一个 Web 应用程序，你可以直接通过浏览器来访问它，无需安装任何其他软件。

当 EMQ X 成功运行在你的本地计算机上且 EMQ X Dashboard 被默认启用时，通过访问 <http://localhost:18083> 来查看Dashboard，默认用户名是 `admin`，密码是 `public`。

如果EMQ X是基于docker容器部署的，可以在容器中的 `etc/plugins/emqx_dashboard.conf` 中查看或修改 EMQ X Dashboard 的配置。EMQ X Dashboard 配置项可以分为**默认用户**与**监听器**两个部分：

默认用户

EMQ X Dashboard 可以配置多个用户，但在配置文件中仅支持配置默认用户。

需要注意的是，一旦您通过 Dashboard 修改了默认用户的密码，则默认用户的相关信息将以您在 Dashboard 上的最新改动为准，配置文件中的默认用户配置将被忽略。

监听器

EMQ X Dashboard 支持 HTTP 和 HTTPS 两种 Listener，但默认只启用了监听端口为 18083 的 HTTP Listener

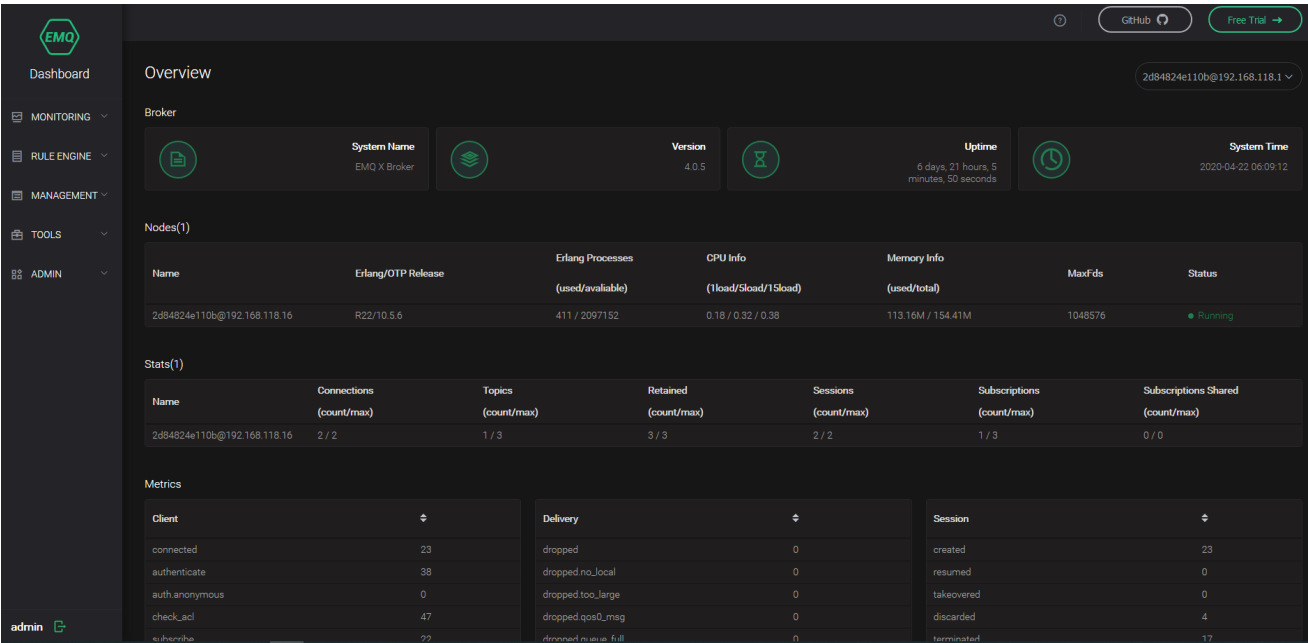
1.2 Dashboard界面

为了使用户在操作和浏览中可以快速地定位和切换当前位置，EMQ X Dashboard 采用了侧边导航的模式，默认情况下 Dashboard 包含以下一级导航项目：

最新版本EMQ X Broker的Dashboard界面布局略有不同，增加了些导航，但基本都差不多

导航项目	说明
MONITORING	提供了服务端与客户端监控信息的展示页面
RULE ENGINE	提供了规则引擎的可视化操作页面
MANAGEMENT	提供了扩展插件与应用的管理页面
TOOLS	提供了 WebSocket 客户端工具以及 HTTP API 速查页面
ADMIN	提供了 Dashboard 用户管理和显示设置等页面

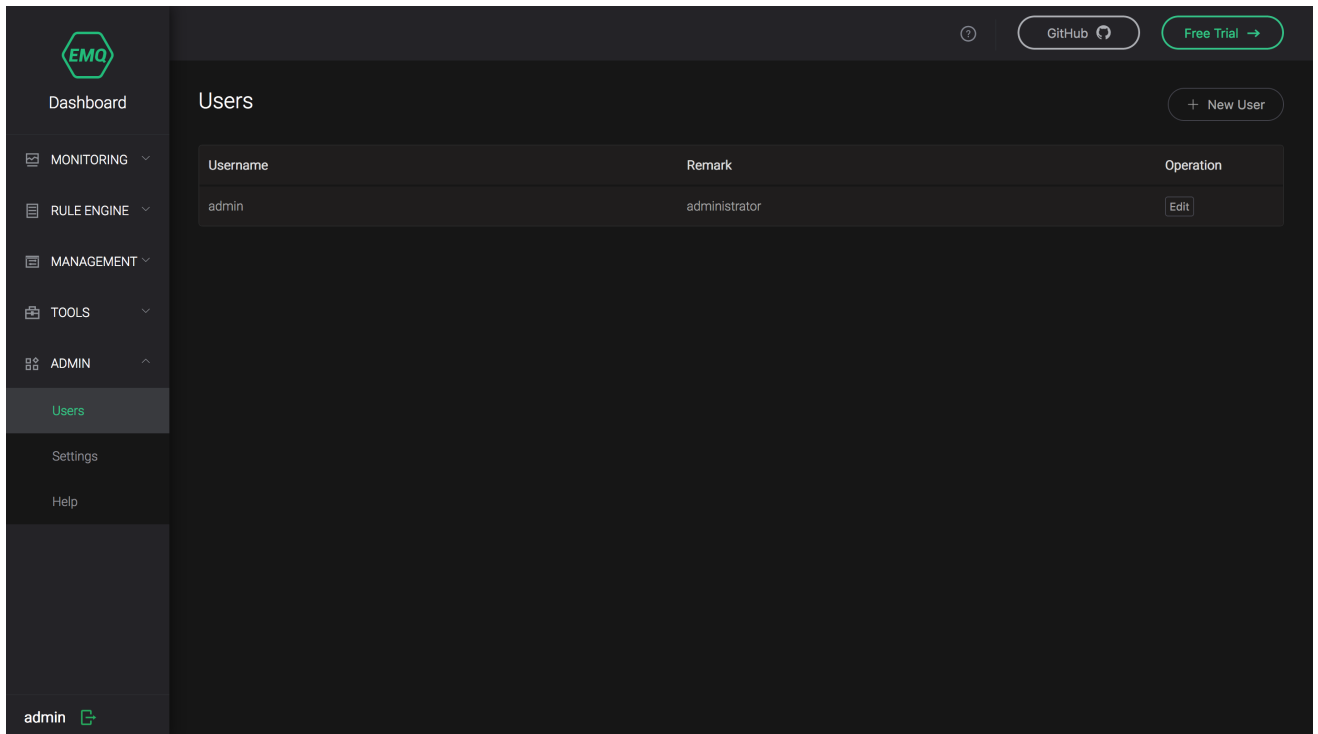
我们通过浏览器访问Dashboard之后会看到类似这样的界面：



1.3 ADMIN

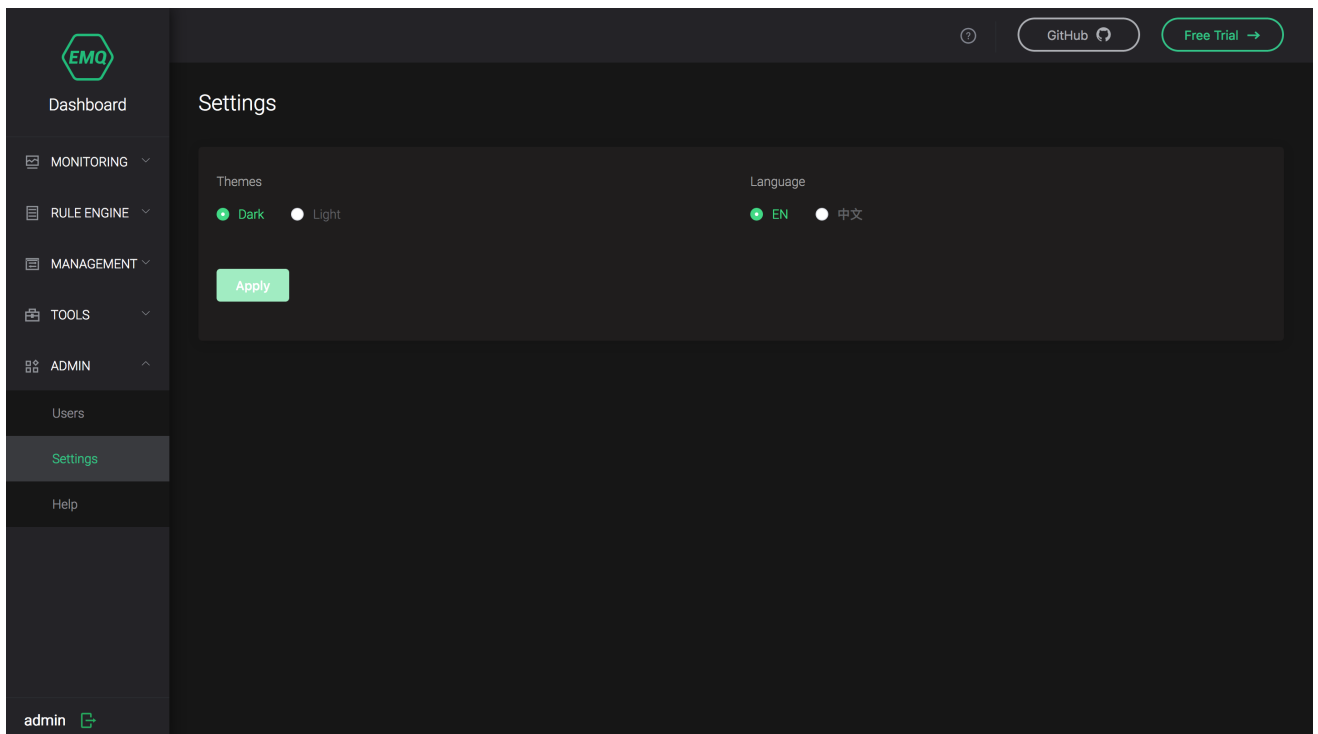
Users

您可以在 **Users** 页面查看和管理能够访问和操作 Dashboard 的用户：



Settings

目前 EMQ X Dashboard 仅支持修改主题和语言两种设置：




1.4 MONITORING

EMQ X Dashboard 提供了非常丰富的数据监控项目，完整地覆盖了服务端与客户端，这些信息都将在 **MONITORING** 下的页面中被合理地展示给用户。

Overview

Overview 作为 Dashboard 的默认展示页面，提供了 EMQ X 当前节点的详细信息和集群其他节点的关键信息，以帮助用户快速掌握每个节点的状态。



Dashboard

MONITORING

Overview

Clients

Topics

Subscriptions

RULE ENGINE

MANAGEMENT

TOOLS

ADMIN

admin


GitHub


Free Trial


emqx@127.0.0.1


Overview

Broker

System Name
EMQ X Broker

Version
develop

Uptime
2 minutes, 42 seconds

System Time
2020-02-26 09:52:20

Nodes(1)

Name	Erlang/OTP Release	Erlang Processes (used/available)	CPU Info (1load/5load/15load)	Memory Info (used/total)	MaxFds	Status
	R21/10.3.5	307 / 2097152	3.02 / 3.88 / 3.80	95.45M / 117.38M	10240	Running

Stats(1)

Name	Connections (count/max)	Topics (count/max)	Retained (count/max)	Sessions (count/max)	Subscriptions (count/max)	Subscriptions Shared (count/max)
emqx@127.0.0.1	0 / 0	0 / 0	3 / 3	0 / 0	0 / 0	0 / 0

Metrics

Client

connected	0
authenticate	0
auth.anonymous	0
check_acl	0
subscribe	0
unsubscribe	0
disconnected	0
connect	0
connack	0

Delivery

dropped	0
dropped.no_local	0
dropped.too_large	0
dropped.qos0_msg	0
dropped.queue_full	0
dropped.expired	0

Session

created	0
resumed	0
takeovered	0
discarded	0
terminated	0

The packets data

received	0
sent	0
disconnect.sent	0
disconnect.received	0
publish.received	0
publish.sent	0
puback.received	0
puback.sent	0
puback.missed	0
pubcomp.received	0
pubcomp.sent	0
pubcomp.missed	0
pubrec.received	0
pubrec.sent	0
pubrec.missed	0
pubrel.received	0
pubrel.sent	0
pubrel.missed	0
connect.received	0
puback.inuse	0
suback.sent	0
connack.sent	0
publish.error	0
auth.sent	0
pubcomp.inuse	0
connack.error	0
publish.inuse	0
pingresp.sent	0
unsubscribe.received	0
pubrec.inuse	0
unsuback.sent	0
disconnect.received	0

The messages data

received	0
sent	0
dropped	0
retained	3
qos0.received	0
qos0.sent	0
qos1.received	0
qos1.sent	0
qos2.received	0
qos2.sent	0
forward	0
publish	0
acked	0
dropped.expired	0
dropped.no_subscribers	0
delivered	0
delayed	0

The bytes data

received	0
sent	0

pingreq.received	0
subscribe.auth_error	0
publish.dropped	0
auth.received	0
publish.auth_error	0
subscribe.error	0
unsubscribe.error	0
subscribe.received	0
connack.auth_error	0

Clients

Clients 页面提供了连接到指定节点的客户端列表，同时支持通过 **Client ID** 直接搜索客户端。除了查看客户端的基本信息，您还可以点击每条记录右侧的 **Kick Out** 按钮踢掉该客户端，注意此操作将断开客户端连接并终结其会话。

Clients 页面使用快照的方式来展示客户端列表，因此当客户端状态发生变化时页面并不会自动刷新，需要您手动刷新浏览器来获取最新客户端数据。

EMQ

Dashboard

MONITORING

Overview

Clients

Topics

Subscriptions

RULE ENGINE

MANAGEMENT

TOOLS

ADMIN

admin

Clients

emqx@127.0.0.1

Client ID

Client ID	Username	IP Address	Keepalive(s)	Expiry Interval (s)	Subscriptions Count	Connect Status	Created At	Operation
efd9f875-82cf-4d9c-...	undefined	127.0.0.1:54942	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
38dd19ff-18c0-415e-...	undefined	127.0.0.1:54945	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
7a1dfceb-89c0-4f7e-...	test	127.0.0.1:54943	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
e6aca0a7-11e3-41b6-...	undefined	127.0.0.1:54938	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
b1d12a3b-33e4-44f1-...	undefined	127.0.0.1:54939	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
c46124fc-6a49-4923-...	undefined	127.0.0.1:54946	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
f4710fe1-2296-431b-...	undefined	127.0.0.1:54948	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
07144db7-bdfd-4355-...	undefined	127.0.0.1:54947	10	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
78082755-e8eb-4a8-...	test	127.0.0.1:54941	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out
b3dd8728-95a3-4c3-...	undefined	127.0.0.1:54940	60	7200	0	CONNECTED	2020-02-26 09:55:21	Kick Out

Total 1210/page12

如果你无法在客户端列表获取到你需要的信息，你可以单击 **Client ID** 来查看客户端的详细信息。

EMQ

Dashboard

MONITORING

Overview

Clients

Topics

Subscriptions

RULE ENGINE

MANAGEMENT

TOOLS

ADMIN

admin

Clients / View

efd9f875-82cf-4d9c-917a-64c748bc84101582682121179

Kick Out

Basic InfoSubscriptions

Connection

Node: emqx@127.0.0.1

Client ID: efd9f875-82cf-4d9c-917a-64c748bc84101582682121179

Username: undefined

Protocol: MQTT v3.1.1

IP Address: 127.0.0.1

Port: 63531

Keepalive(s): 60

Is Bridge: false

Connected At: 2020-02-26 13:08:51

Connect Status: CONNECTED

Zone: external

Collapse

Number of TCP Packets Received: 1

Number of PUBLISH Packets Received: 0

Number of Bytes Received: 63

Number of MQTT Packets Received: 1

Session

Clean Session: false

Expiry Interval(s): 7200

Created At: 2020-02-26 13:08:51

Subscriptions Count: 0 / Unlimited

Maximum Subscriptions Count: Unlimited

Inflight Queue Size: 0 / 32

Maximum Inflight Queue Size: 32

Message Queue Size: 0 / 1000

Maximum Message Queue Size: 1000

Unconfirmed PUBREC Packets Count: 0

Maximum Unconfirmed PUBREC Packets Count: 100

Number of TCP Packets Sent: 1

Number of PUBLISH Packets Sent: 0

Number of Bytes Sent: 4

Number of MQTT Packets Sent: 1

我们将客户端详情中的各个字段分为了 **连接**，**会话** 和 **指标** 三类，以下为各字段的说明：

连接

字段名	说明
Node	客户端连接的节点名称
Client ID	客户端标识符
Username	客户端连接时使用的用户名，出于安全性考虑，密码将不会被展示
Protocol	客户端使用的协议名称及其版本
IP Address	客户端的网络 IP 地址，地址可以是 IPv4 或 IPv6
Port	客户端源端口
Is Bridge	指示客户端是否通过桥接方式连接
Connected At	客户端连接时间
Disconnected At	客户端离线时间
Connection Status	客户端连接状态
Zone	指示客户端所使用的配置组
Keepalive	保持连接时间，单位：秒

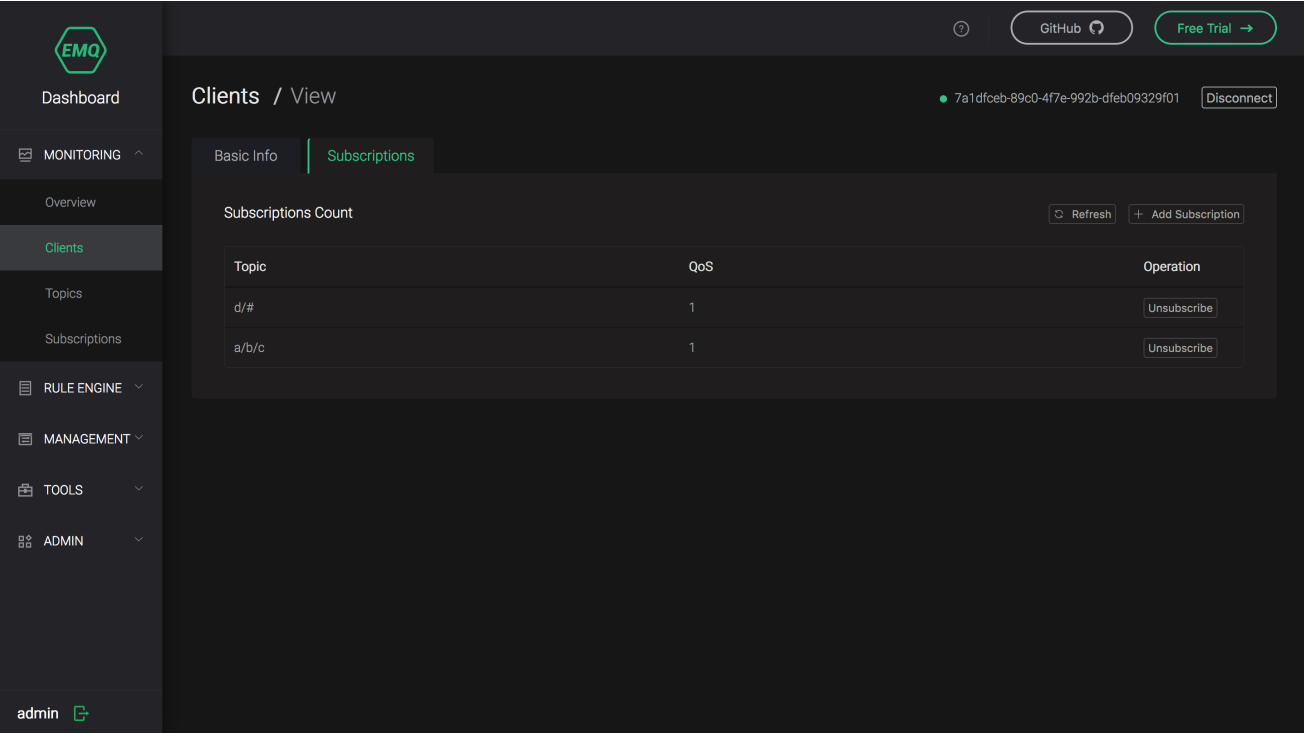
会话

字段名	说明
Clean Session	指示客户端是否使用了全新的会话
Expiry Interval	会话过期间隔，单位：秒
Created At	会话创建时间
Subscriptions Count	当前订阅数量
Maximum Subscriptions Count	允许建立的最大订阅数量
Inflight Window Size	当前飞行窗口大小
Maximum Inflight Window Size	飞行窗口最大大小
Message Queue Size	当前消息队列大小
Maximum Message Queue Size	消息队列最大大小
Unconfirmed PUBREC Packets	未确认的 PUBREC 报文数量
Maximum Unconfirmed PUBREC Packets	允许存在未确认的 PUBREC 报文的最大数量

指标

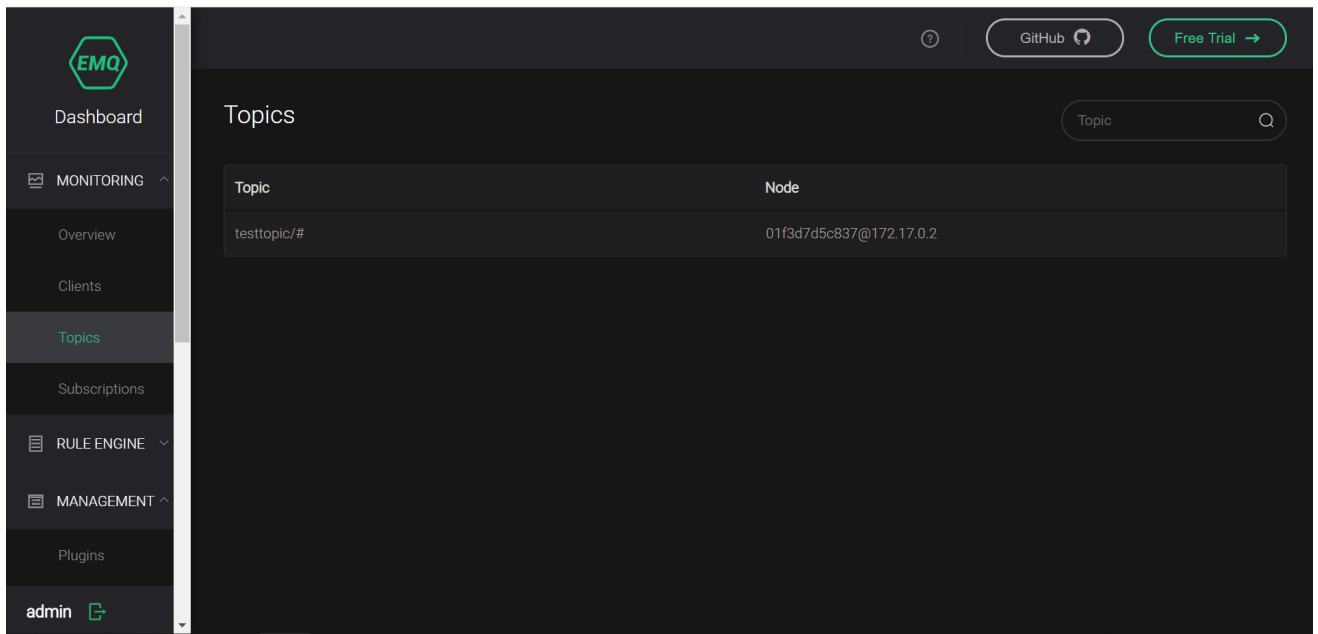
字段名	说明
Number of Bytes Received	EMQ X Broker（下同）接收的字节数量
Number of TCP Packets Received	接收的 TCP 报文数量
Number of MQTT Packets Received	接收的 MQTT 报文数量
Number of PUBLISH Packets Received	接收的 PUBLISH 报文数量
Number of Bytes Sent	发送的字节数量
Number of TCP Packets Sent	发送的 TCP 报文数量
Number of MQTT Packets Sent	发送的 MQTT 报文数量
Number of PUBLISH Packets Sent	发送的 PUBLISH 报文数量

在客户端详情的 `Subscriptions` 标签页中，您可以查看当前客户端的订阅信息，以及新建或取消订阅：



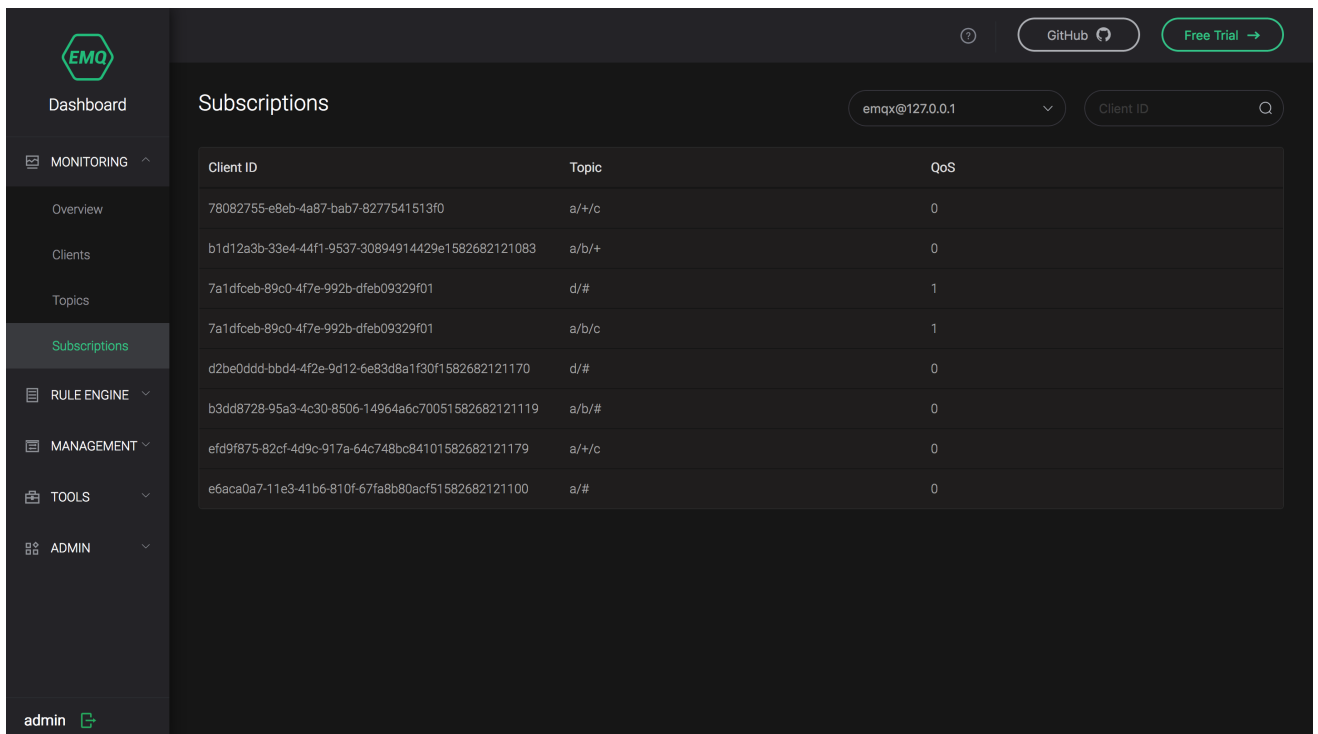
Topics

展示系统所有的Topic情况



Subscriptions

Subscriptions 页面提供了指定节点下的所有订阅信息，并且支持用户通过 **Client ID** 查询指定客户端的所有订阅。



1.5 RULE ENGINE

用 EMQ X 的规则引擎可以灵活地处理消息和事件，例如将消息转换成指定格式后存入数据库表或者重新发送到消息队列等等。为了方便用户更好地使用规则引擎，EMQ X Dashboard 提供了相应的可视化操作页面，您可以点击 **RULE ENGINE** 导航项目来访问这些页面。

鉴于规则引擎的相关概念比较复杂，涉及到的操作可能会占据相当大的篇幅，后面会单独开辟一个章节来介绍。

1.6 MANAGEMENT

目前 EMQ X Dashboard 的 **MANAGEMENT** 导航项目下主要包括扩展插件 的监控管理页面和用于 HTTP API 认证的 AppID 与 AppSect 的管理页面。

Plugins

Plugins 页面列举了 EMQ X 能够发现的所有插件，包括 EMQ X 官方插件与您遵循 EMQ X 官方标准自行开发的插件，您可以在此页面查看插件当前的运行状态以及随时启停插件。

Dashboard

MONITORING

RULE ENGINE

MANAGEMENT

Plugins

Listeners

Applications

TOOLS

ADMIN

admin

Plugins

emqx@127.0.0.1

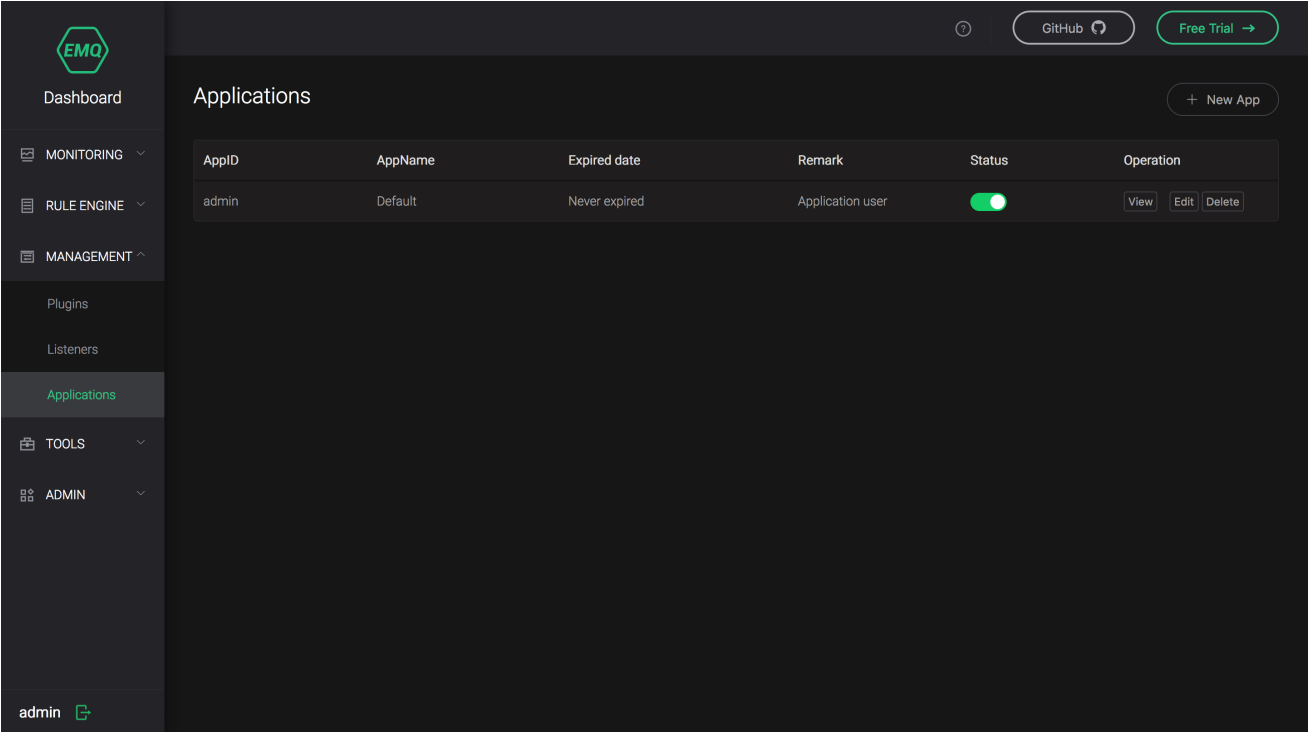
Search by plugin name

Name	Version	Description	Status	Operation
emqx_auth_clientid	develop	EMQ X Authentication with ClientId/Password	Stopped	Start
emqx_auth_http	develop	EMQ X Authentication/ACL with HTTP API	Stopped	Start
emqx_auth_jwt	develop	EMQ X Authentication with JWT	Stopped	Start
emqx_auth_ldap	develop	EMQ X Authentication/ACL with LDAP	Stopped	Start
emqx_auth_mongo	develop	EMQ X Authentication/ACL with MongoDB	Stopped	Start
emqx_auth_mysql	develop	EMQ X Authentication/ACL with MySQL	Stopped	Start
emqx_auth_pgsq	develop	EMQ X Authentication/ACL with PostgreSQL	Stopped	Start
emqx_auth_redis	develop	EMQ X Authentication/ACL with Redis	Stopped	Start
emqx_auth_username	develop	EMQ X Authentication with Username and Password	Stopped	Start
emqx_bridge_mqtt	develop	EMQ X Bridge to MQTT Broker	Stopped	Start
emqx_coap	develop	EMQ X CoAP Gateway	Stopped	Start
emqx_dashboard	develop	EMQ X Web Dashboard	Running	Stop
emqx_delayed_publish	develop	EMQ X Delayed Publish	Stopped	Start
emqx_lua_hook	develop	EMQ X Lua Hooks	Stopped	Start
emqx_lwm2m	develop	EMQ X Lwm2M Gateway	Stopped	Start
emqx_management	develop	EMQ X Management API and CLI	Running	Stop
emqx_plugin_template	develop	EMQ X Plugin Template	Stopped	Start
emqx_psk_file	develop	EMQX PSK Plugin from File	Stopped	Start
emqx_recon	develop	EMQ X Recon Plugin	Running	Stop
emqx_reloader	develop	EMQ X Reloader Plugin	Stopped	Start
emqx_retainer	develop	EMQ X Retainer	Running	Stop
emqx_rule_engine	develop	EMQ X Rule Engine	Running	Stop
emqx_sn	develop	EMQ X MQTT-SN Plugin	Stopped	Start
emqx_statsd	develop	Statsd for EMQ X	Stopped	Start
emqx_stomp	develop	EMQ X Stomp Protocol Plugin	Stopped	Start
emqx_web_hook	develop	EMQ X Webhook Plugin	Stopped	Start

可以看到，除了emqx-dashboard以外， EMQ X 还将默认启动 emqx-rule-engine等4个插件。

Applications

Applications 页面列举了当前已创建的应用，您可以在该页面进行诸如创建应用、临时禁用或启动某个应用的访问权限等操作。EMQ X 会创建一个 AppID 为 `admin`，AppSecret 为 `publish` 的默认应用方便用户首次访问：



您可以单击 **Application** 页面右上角的 **New App** 按钮来创建一个新的应用，其中 AppID 与 AppSecret 是必选项。创建完成后您可以单击 **View** 按钮来查看应用详情，AppSecret 也会在详情中显示。以下是相关字段的说明：


字段名	说明
AppID	应用标识符，用于区分不同的应用，因此不可重复，在创建应用时 Dashboard 将自动为您生成一个随机的建议应用标识符
AppName	应用名称，可以重复，但是不建议使用重复的应用名称
AppSecret	由 EMQ X 分配的应用密钥，可以在应用详情中查看
Expired date	应用的过期时间，默认为永不过期
Remark	您对应用的描述，方便后期管理
Status	应用状态，只有 Allowed 与 Denied 两种，Denied 状态下 EMQ X 将拒绝使用该 AppID 与 App Secret 的 HTTP API 的访问请求

1.7 TOOLS

目前 EMQ X Dashboard 的 **TOOLS** 导航项目下主要包括 WebSocket 客户端工具页面以及 HTTP API 速查页面。

Websocket

Websocket 页面为您提供了一个简易但有效的 WebSocket 客户端工具，它包含了连接、订阅和发布功能，同时还能查看自己发送和接收的报文数据，我们期望它可以帮助您快速地完成某些场景或功能的测试验证：



Dashboard

MONITORING

RULE ENGINE

MANAGEMENT

TOOLS

Websocket

HTTP API

ADMIN

admin

Websocket

Connect

Host

127.0.0.1

Port

8083

Path

/mqtt

Client ID

mqttjs_cb699d6526

Username

Password

Keep Alive

60

☒ Clean Session

☐ SSL

ws://127.0.0.1:8083/mqtt

Connect

Disconnect

Current State: DISCONNECTED

Subscribe

Topic

testtopic/#

Subscribe

Topic	QoS	Time	Operation
No Data			

QoS

0

Subscribe

Messages

Topic

testtopic

Messages

{ "msg": "Hello, World!" }

QoS

0

☐ Retained

send

Messages already sent

Messages received

Messages	Topic	QoS	Time
No Data			

Messages	Topic	QoS	Time
No Data			

HTTP API

HTTP API 页面列举了 EMQ X 目前支持的所有 HTTP API 及其说明：



Dashboard

MONITORING

RULE ENGINE

MANAGEMENT

TOOLS

Websocket

HTTP API

ADMIN

admin



GitHub



Free Trial



HTTP API

Introduction

Apart from this help page, all URIs will serve only resources of type application/json, and will require HTTP basic authentication. The default user is admin / public. The emqx_dashboard plugin provides a web management console. The Dashboard helps monitor broker's running status, statistics and metrics of MQTT packets.

Reference

Method	Path	Description
GET	/api/v4/auth_clientid	List available clientid in the cluster
GET	/api/v4/auth_clientid/clientid	Lookup clientid in the cluster
POST	/api/v4/auth_clientid	Add clientid in the cluster
PUT	/api/v4/auth_clientid/clientid	Update clientid in the cluster
DELETE	/api/v4/auth_clientid/clientid	Delete clientid in the cluster
GET	/api/v4/auth_username	List available username in the cluster
GET	/api/v4/auth_username/username	Lookup username in the cluster
POST	/api/v4/auth_username	Add username in the cluster
PUT	/api/v4/auth_username/username	Update username in the cluster
DELETE	/api/v4/auth_username/username	Delete username in the cluster
POST	/api/v4/auth	Authenticate an user
POST	/api/v4/users/	Create an user
GET	/api/v4/users/	List users
PUT	/api/v4/users/name	Update an user
DELETE	/api/v4/users/name	Delete an user
PUT	/api/v4/change_pwd/username	Change password for an user
GET	/api/v4/alarms/present	List all alarms
GET	/api/v4/alarms/present/emqx@127.0.0.1	List alarms of a node
GET	/api/v4/alarms/history	List all alarm history
GET	/api/v4/alarms/history/emqx@127.0.0.1	List alarm history of a node
POST	/api/v4/apps/	Add Application
DELETE	/api/v4/apps/appid	Delete Application
GET	/api/v4/apps/	List Applications
GET	/api/v4/apps/appid	Lookup Application
PUT	/api/v4/apps/appid	Update Application
GET	/api/v4/banned/	List banned
POST	/api/v4/banned/	Create banned
DELETE	/api/v4/banned/who	Delete banned
GET	/api/v4/brokers/	A list of brokers in the cluster
GET	/api/v4/brokers/emqx@127.0.0.1	Get broker info of a node
GET	/api/v4/clients/	A list of clients on current node
GET	/api/v4/nodes/emqx@127.0.0.1/clients/	A list of clients on specified node
GET	/api/v4/clients/clientid	Lookup a client in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/clients/clientid	Lookup a client on the node
GET	/api/v4/clients/username/username	Lookup a client via username in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/clients/username/username	Lookup a client via username on the node
DELETE	/api/v4/clients/clientid	Kick out the client in the cluster
DELETE	/api/v4/clients/clientid/acl_cache	Clear the ACL cache of a specified client in the cluster
GET	/api/v4/clients/clientid/acl_cache	List the ACL cache of a specified client in the cluster

GET	/api/v4/listeners/	A list of listeners in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/listeners	A list of listeners on the node
GET	/api/v4/metrics/	A list of metrics of all nodes in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/metrics/	A list of metrics of a node
GET	/api/v4/nodes/	A list of nodes in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1	Lookup a node in the cluster
GET	/api/v4/plugins/	List all plugins in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/plugins/	List all plugins on a node
PUT	/api/v4/nodes/emqx@127.0.0.1/plugins/plugin/load	Load a plugin
PUT	/api/v4/nodes/emqx@127.0.0.1/plugins/plugin/unload	Unload a plugin
PUT	/api/v4/nodes/emqx@127.0.0.1/plugins/plugin/reload	Reload a plugin
PUT	/api/v4/plugins/plugin/unload	Unload a plugin in the cluster
PUT	/api/v4/plugins/plugin/reload	Reload a plugin in the cluster
POST	/api/v4/mqtt/subscribe	Subscribe a topic
POST	/api/v4/mqtt/publish	Publish a MQTT message
POST	/api/v4/mqtt/unsubscribe	Unsubscribe a topic
POST	/api/v4/mqtt/subscribe_batch	Batch subscribes topics
POST	/api/v4/mqtt/publish_batch	Batch publish MQTT messages
POST	/api/v4/mqtt/unsubscribe_batch	Batch unsubscribes topics
GET	/api/v4/routes/	List routes
GET	/api/v4/routes/topic	Lookup routes to a topic
GET	/api/v4/stats/	A list of stats of all nodes in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/stats/	A list of stats of a node
GET	/api/v4/subscriptions/	A list of subscriptions in the cluster
GET	/api/v4/nodes/emqx@127.0.0.1/subscriptions/	A list of subscriptions on a node
GET	/api/v4/subscriptions/clientid	A list of subscriptions of a client
GET	/api/v4/nodes/emqx@127.0.0.1/subscriptions/clientid	A list of subscriptions of a client on the node
POST	/api/v4/rules/	Create a rule
GET	/api/v4/rules/	A list of all rules
GET	/api/v4/rules/id	Show a rule
DELETE	/api/v4/rules/id	Delete a rule
GET	/api/v4/actions/	A list of all actions
GET	/api/v4/actions/name	Show an action
GET	/api/v4/resources/	A list of all resources
POST	/api/v4/resources/	Create a resource
GET	/api/v4/resources/id	Show a resource
GET	/api/v4/resource_status/id	Get status of a resource
POST	/api/v4/resources/id	Start a resource
DELETE	/api/v4/resources/id	Delete a resource
GET	/api/v4/resource_types/	List all resource types
GET	/api/v4/resource_types/name	Show a resource type
GET	/api/v4/resource_types/type/resources	List all resources of a resource type
GET	/api/v4/rule_events/	List all events with detailed info

2.认证

2.1 认证简介

身份认证是大多数应用的重要组成部分，MQTT 协议支持用户名密码认证，启用身份认证能有效阻止非法客户端的连接。

EMQ X 中的认证指的是当一个客户端连接到 EMQ X 的时候，通过服务器端的配置来控制客户端连接服务器的权限。

EMQ X 的认证支持包括两个层面：

- MQTT 协议本身在 CONNECT 报文中指定用户名和密码，EMQ X 以插件形式支持基于 Username、ClientID、HTTP、JWT、LDAP 及各类数据库如 MongoDB、MySQL、PostgreSQL、Redis 等多种形式的认证。
- 在传输层上，TLS 可以保证使用客户端证书的客户端到服务器的身份验证，并确保服务器向客户端验证服务器证书。也支持基于 PSK 的 TLS/DTLS 认证。

2.1.1 认证方式

EMQ X 支持使用内置数据源（文件、内置数据库）、JWT、外部主流数据库和自定义 HTTP API 作为身份认证数据源。

连接数据源、进行认证逻辑通过插件实现的，每个插件对应一种认证方式，使用前需要启用相应的插件。

客户端连接时插件通过检查其 username/clientid 和 password 是否与指定数据源的信息一致来实现对客户端的身份认证。

EMQ X 支持的认证方式：

内置数据源

- Username 认证
- Client ID 认证

使用配置文件与 EMQ X 内置数据库提供认证数据源，通过 HTTP API 进行管理，足够简单轻量。

外部数据库

- LDAP 认证
- MySQL 认证
- PostgreSQL 认证
- Redis 认证
- MongoDB 认证

外部数据库可以存储大量数据，同时方便与外部设备管理系统集成。

其他

- HTTP 认证
- JWT 认证

JWT 认证可以批量签发认证信息，HTTP 认证能够实现复杂的认证鉴权逻辑。

更改插件配置后需要重启插件才能生效，部分认证鉴权插件包含 ACL 功能

认证结果

任何一种认证方式最终都会返回一个结果：

- 认证成功：经过比对客户端认证成功
- 认证失败：经过比对客户端认证失败，数据源中密码与当前密码不一致

- 忽略认证 (ignore)：当前认证方式中未查找到认证数据，无法显式判断结果是成功还是失败，交由认证链下一认证方式或匿名认证来判断

匿名认证

EMQ X 默认配置中启用了匿名认证，任何客户端都能接入 EMQ X。没有启用认证插件或认证插件没有显式允许/拒绝 (ignore) 连接请求时，EMQ X 将根据匿名认证启用情况决定是否允许客户端连接。

配置匿名认证开关：

```
1 # etc/emqx.conf
2
3 ## Value: true | false
4 allow_anonymous = true
```

生产环境中请禁用匿名认证。

注意：我们需要进入到容器内部修改该配置，然后重启EMQ X服务

密码加盐规则与哈希方法

EMQ X 多数认证插件中可以启用哈希方法，数据源中仅保存密码密文，保证数据安全。

启用哈希方法时，用户可以为每个客户端都指定一个 salt（盐）并配置加盐规则，数据库中存储的密码是按照加盐规则与哈希方法处理后的密文。

以 MySQL 认证为例：

加盐规则与哈希方法配置：

```
1 # etc/plugins/emqx_auth_mysql.conf
2
3 ## 不加盐，仅做哈希处理
4 auth.mysql.password_hash = sha256
5
6 ## salt 前缀：使用 sha256 加密 salt + 密码 拼接的字符串
7 auth.mysql.password_hash = salt,sha256
8
9 ## salt 后缀：使用 sha256 加密 密码 + salt 拼接的字符串
10 auth.mysql.password_hash = sha256,salt
11
12 ## pbkdf2 with macfun iterations dklen
13 ## macfun: md4, md5, ripemd160, sha, sha224, sha256, sha384, sha512
14 ## auth.mysql.password_hash = pbkdf2,sha256,1000,20
```

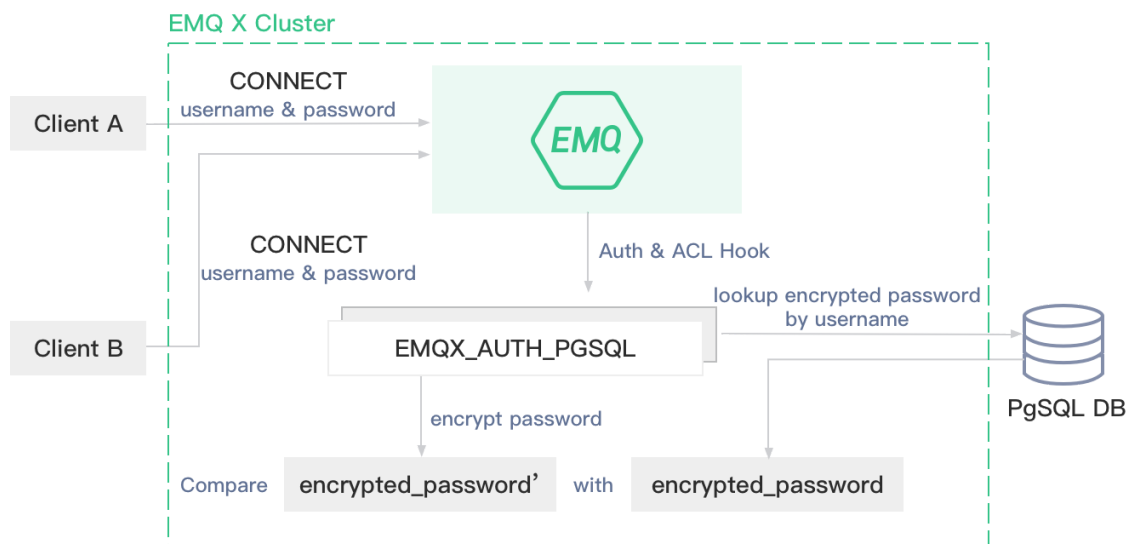
如何生成认证信息

1. 为每个客户端分用户名、Client ID、密码以及 salt（盐）等信息
2. 使用与 MySQL 认证相同加盐规则与哈希方法处理客户端信息得到密文
3. 将客户端信息写入数据库，客户端的密码应当为密文信息

2.1.2 EMQ X 身份认证流程

1. 根据配置的认证 SQL 结合客户端传入的信息，查询出密码（密文）和 salt（盐）等认证数据，没有查询结果时，认证将终止并返回 ignore 结果
2. 根据配置的加盐规则与哈希方法计算得到密文，没有启用哈希方法则跳过此步
3. 将数据库中存储的密文与当前客户端计算的到的密文进行比对，比对成功则认证通过，否则认证失败

PostgreSQL 认证功能逻辑图：



写入数据的加盐规则、哈希方法与对应插件的配置一致时认证才能正常进行。更改哈希方法会造成现有认证数据失效。

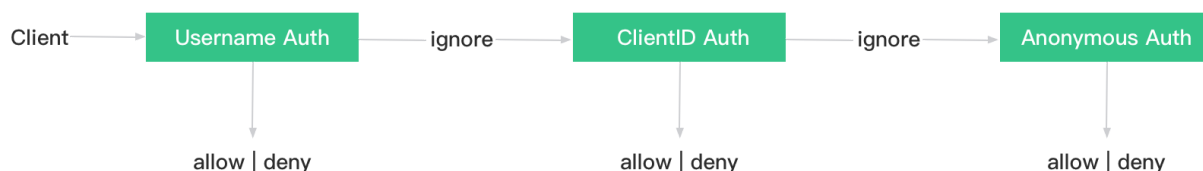
认证链

当同时启用多个认证方式时，EMQ X 将按照插件开启先后顺序进行链式认证：

- 一旦认证成功，终止认证链并允许客户端接入
- 一旦认证失败，终止认证链并禁止客户端接入
- 直到最后一个认证方式仍未通过，根据匿名认证

配置判定

- 匿名认证开启时，允许客户端接入
- 匿名认证关闭时，禁止客户端接入



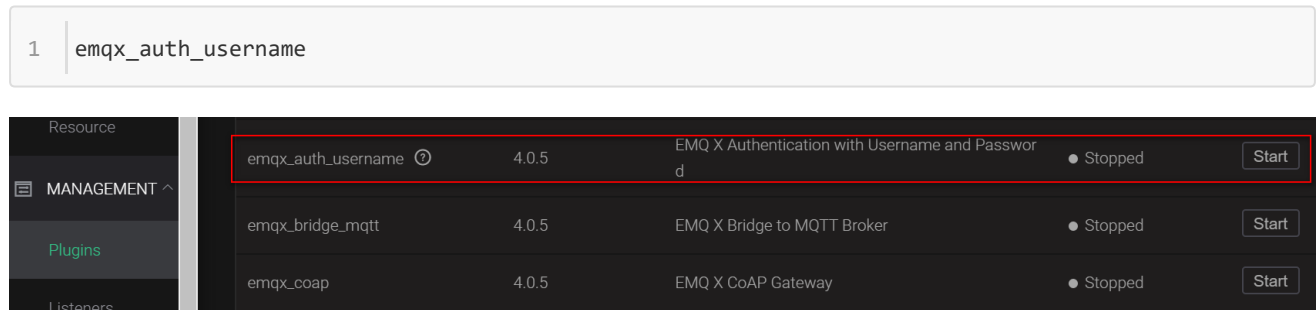
同时只启用一个认证插件可以提高客户端身份认证效率。

2.2 Username 认证

Username 认证使用配置文件预设客户端用户名与密码，支持通过 HTTP API 管理认证数据。

Username 认证不依赖外部数据源，使用上足够简单轻量。使用这种认证方式前需要开启插件，我们可以在 Dashboard 里找到这个插件并开启。

插件：



哈希方法

Username 认证默认使用 sha256 进行密码哈希加密，可在 `etc/plugins/emqx_auth_username.conf` 中更改：

```
1 # etc/plugins/emqx_auth_username.conf
2
3 ## Value: plain | md5 | sha | sha256
4 auth.user.password_hash = sha256
```

配置哈希方法后，新增的预设认证数据与通过 HTTP API 添加的认证数据将以哈希密文存储在 EMQ X 内置数据库中。

2.2.1 预设认证数据

可以通过配置文件预设认证数据，编辑配置文件：`etc/plugins/emqx_auth_username.conf`

```
1 # etc/plugins/emqx_auth_username.conf
2
3 ##-----
4 ## Username Authentication Plugin
5 ##-----
6
7 ## Examples:
8 ##auth.user.1.username = admin
9 ##auth.user.1.password = public
10 ##auth.user.2.username = feng@emqtt.io
11 ##auth.user.2.password = public
12 ##auth.user.3.username = name~!@#$$^&*()_+
13 ##auth.user.3.password = pwsswd~!@#$$^&*()_+
14
15 ## Password hash.
16 ##
17 ## Value: plain | md5 | sha | sha256
18 auth.user.password_hash = sha256
```

插件启动时将读取预设认证数据并加载到 EMQ X 内置数据库中，节点上的认证数据会在此阶段同步至集群中。

预设认证数据在配置文件中使用了明文密码，出于安全性与可维护性考虑应当避免使用该功能。

2.2.2 HTTP API 管理认证数据

EMQ X提供了对应的HTTP API用以维护内置数据源中的认证信息，我们可以添加/查看/取消/更改认证数据

我们通过VSCode来访问EMQ X的API `/auth_username` 完成认证数据的相关操作

1: 查看已有认证用户数据: `GET api/v4/auth_username`

```
1 @hostname = 192.168.200.129
2 @port=18083
3 @contentType=application/json
4 @userName=admin
5 @password=public
6
7 #####查看已有用户认证数据#####
8 GET http://{{hostname}}:{{port}}/api/v4/auth_username HTTP/1.1
9 Content-Type: {{contentType}}
10 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 20
4 content-type: application/json
5 date: Thu, 04 Jun 2020 08:02:39 GMT
6 server: Cowboy
7
8 {
9   "data": [],
10  "code": 0
11 }
```

2: 添加认证数据API 定义: `POST api/v4/auth_username{ "username": "emqx_u", "password": "emqx_p"}`

```
1 #####添加用户认证数据#####
2 POST http://{{hostname}}:{{port}}/api/v4/auth_username HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
5
6 {
7   "username": "user",
8   "password": "123456"
9 }
```

使用 POST 请求添加 username 为 `user` password 为 `123456` 的认证信息，返回信息中 `code = 0` 即为成功。

返回：

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 10
4 content-type: application/json
5 date: Thu, 04 Jun 2020 08:08:09 GMT
6 server: Cowboy
7
8 {
9   "code": 0
10 }
```

3：更改指定用户名的密码API 定义：`PUT api/v4/auth_username/${username}{ "password": "emqx_new_p"}`

指定用户名，传递新密码进行更改，再次连接时需要使用新密码进行连接：

```
1 #####更改指定用户名的密码#####
2 PUT http://{{hostname}}:{{port}}/api/v4/auth_username/user HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
5
6 {
7   "password": "user"
8 }
```

返回：

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 10
4 content-type: application/json
5 date: Thu, 04 Jun 2020 08:12:58 GMT
6 server: Cowboy
7
8 {
9   "code": 0
10 }
```

4：查看指定用户名信息API 定义：`GET api/v4/auth_username/${username}`

指定用户名，查看相关用户名、密码信息，注意此处返回的密码是使用配置文件指定哈希方式加密后的密码：

```
1 #####查看指定用户名信息#####
2 GET http://{{hostname}}:{{port}}/api/v4/auth_username/user HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 115
4 content-type: application/json
5 date: Thu, 04 Jun 2020 08:14:47 GMT
6 server: Cowboy
7
8 {
9   "data": {
10     "username": "user",
11     "password": "687f54edc0779f4e26314d9cc957eba27a078a39997069d3c52c9dda3db4aa07"
12   },
13   "code": 0
14 }
```

5: 删除认证数据API 定义: `DELETE api/v4/auth_username/${username}`

用以删除指定认证数据

```
1 #####删除指定的用户信息#####
2 DELETE http://{{hostname}}:{{port}}/api/v4/auth_username/user HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 10
4 content-type: application/json
5 date: Thu, 04 Jun 2020 08:17:09 GMT
6 server: Cowboy
7
8 {
9   "code": 0
10 }
```

2.2.3 MQTTX客户端验证

使用mqtt客户端工具验证使用username连接登录的功能。从 <https://github.com/emqx/MQTTX> 这个地址下载对应操作系统的mqtt客户端工具。

1: 新建连接, 参数配置如下

[< 返回](#)

新建

[连接](#)

基础

* 名称

测试username认证

* Client ID

mqttx_b471dd11

⌂

* 服务器地址

mqtt://

192.168.200.129

* 端口

1883

用户名

user

密码

.....

SSL/TLS

☐ true

☒ false

在对应的输入框内输入username和密码，clientId这里目前可以随便输入(因为基于clientId的认证功能还没有启用)，之后点连接，连接成功。用户名和密码如果输入错误的话是连接不成功的。

2: 再次创建一个客户端连接，可作为消息的订阅者，上一个连接作为发布者，如下

[< 返回](#)

新建

[连接](#)

基础

* 名称

测试username认证订阅者

* Client ID

mqttx_7237337d

⌂

* 服务器地址

mqtt://

192.168.200.129

* 端口

1883

用户名

user

密码

.....

SSL/TLS

☐ true

☒ false

3: 订阅者添加订阅

测试username认证订阅者

+ 添加订阅

添加订阅

* Topic

testtopic/#

* QoS

0

标记

#67357C

取消

确定

订阅完成后：

测试username认证订阅者

+ 添加订阅

testtopic/#

QoS 2

4：上一个客户端连接作为消息的发布者来进行消息的发布：

Payload:

JSON

QoS:

0

Retain: ☐

testtopic/123

{
 "msg": "hello"
}

5: 查看订阅者是否已经接收到消息:

● 测试username认证@192.1...

● 测试username认证订阅者...

+ 添加订阅

testtopic/# QoS 2

Topic: testtopic/123 QoS: 2

{
 "msg": "hello"
}

2.3 Client ID 认证

Client ID 认证使用配置文件预设客户端Client ID 与密码，支持通过 HTTP API 管理认证数据。

Client ID 认证不依赖外部数据源，使用上足够简单轻量，使用该种认证方式时需要开启 `emqx_auth_clientid` 插件，直接在Dashboard中开启即可，

哈希方法

Client ID 认证默认使用 sha256 进行密码哈希加密，可在 `etc/plugins/emqx_auth_clientid.conf` 中更改：

```
1 # etc/plugins/emqx_auth_clientid.conf
2
3 ## Value: plain | md5 | sha | sha256
4 auth.client.password_hash = sha256
```

配置哈希方法后，新增的预设认证数据与通过 HTTP API 添加的认证数据将以哈希密文存储在 EMQ X 内置数据库中。

2.3.1 预设认证数据

可以通过配置文件预设认证数据，编辑配置文件： `etc/plugins/emqx_auth_clientid.conf`

```
1 # etc/plugins/emqx_auth_clientid.conf
2
3 ##-----
4 ## ClientId Authentication Plugin
5 ##-----
6
```

```

7  ## Examples
8  ##auth.client.1.clientid = id
9  ##auth.client.1.password = passwd
10 ##auth.client.2.clientid = dev:devid
11 ##auth.client.2.password = passwd2
12 ##auth.client.3.clientid = app:appid
13 ##auth.client.3.password = passwd3
14 ##auth.client.4.clientid = client~!@#%&^&*(*)_+
15 ##auth.client.4.password = passwd~!@#%&^&*(*)_+
16
17 ## Password hash.
18 ##
19 ## Value: plain | md5 | sha | sha256
20 auth.client.password_hash = sha256

```

插件启动时将读取预设认证数据并加载到 EMQ X 内置数据库中，节点上的认证数据会在此阶段同步至集群中。

预设认证数据在配置文件中使用了明文密码，出于安全性与可维护性考虑应当避免使用该功能。

2.3.2 HTTP API 管理认证数据

我们使用VSCode来通过EMQ X的API来添加和查看Client ID的认证数据。

1: 添加认证数据API 定义: `POST api/v4/auth_clientid{ "clientid": "emqx_c", "password": "emqx_p"}`

```

1  #####添加clientId和密码#####
2  POST http://{{hostname}}:{{port}}/api/v4/auth_clientid HTTP/1.1
3  Content-Type: {{contentType}}
4  Authorization: Basic {{userName}}:{{password}}
5
6  {
7      "clientid": "emq-client1",
8      "password": "123456"
9  }

```

使用 POST 请求添加 clientid 为 `emq-client1` password 为 `123456` 的认证信息，返回信息中 `code = 0` 即为成功。

返回:

```

1  HTTP/1.1 200 OK
2  connection: close
3  content-length: 10
4  content-type: application/json
5  date: Thu, 04 Jun 2020 09:34:16 GMT
6  server: Cowboy
7
8  {
9      "code": 0
10 }

```

2: 查看已经添加的认证数据API 定义: GET api/v4/auth_clientid

```
1 #####获取所有详细信息#####
2 GET http://{{hostname}}:{{port}}/api/v4/auth_clientid HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 33
4 content-type: application/json
5 date: Thu, 04 Jun 2020 09:42:24 GMT
6 server: Cowboy
7
8 {
9   "data": [
10     "emq-client1"
11   ],
12   "code": 0
13 }
```

3: 更改指定 Client ID 的密码API 定义: PUT api/v4/auth_clientid/\${clientId}{ "password":
"emqx_new_p"}

指定 Client ID, 传递新密码进行更改, 再次连接时需要使用新密码进行连接:

```
1 #####更改指定 Client ID 的密码#####
2 PUT http://{{hostname}}:{{port}}/api/v4/auth_clientid/emq-client1 HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
5
6 {
7   "password": "654321"
8 }
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 10
4 content-type: application/json
5 date: Thu, 04 Jun 2020 09:41:29 GMT
6 server: Cowboy
7
8 {
9   "code": 0
10 }
```

4: 查看指定 Client ID 信息API 定义: `GET api/v4/auth_clientid/${clientid}`

指定 Client ID, 查看相关 Client ID、密码信息, 注意此处返回的密码是使用配置文件指定哈希方式加密后的密码:

```
1 #####获取指定ClientId详细信息#####
2 GET http://{hostname}://{port}/api/v4/auth_clientid/emq-client1 HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 122
4 content-type: application/json
5 date: Thu, 04 Jun 2020 09:48:36 GMT
6 server: Cowboy
7
8 {
9   "data": {
10     "password": "cab4af60ae344f27da3de35b8f031286468f7983553f5cd2ac1e6f960b6d76b6",
11     "clientid": "emq-client1"
12   },
13   "code": 0
14 }
```

5: 删除认证数据API 定义: `DELETE api/v4/auth_clientid/${clientid}`

删除指定 Client ID:

```
1 #####删除指定的client信息#####
2 DELETE http://{hostname}://{port}/api/v4/auth_clientid/emq-client1 HTTP/1.1
3 Content-Type: {{contentType}}
4 Authorization: Basic {{userName}}:{{password}}
```

返回:

```
1 HTTP/1.1 200 OK
2 connection: close
3 content-length: 10
4 content-type: application/json
5 date: Thu, 04 Jun 2020 09:50:23 GMT
6 server: Cowboy
7
8 {
9   "code": 0
10 }
```

2.3.3 MQTTX客户端验证

使用mqtt客户端工具验证使用Client ID连接登录的功能

[< 返回](#)[新建](#)[连接](#)

基础

* 名称

测试clientID登陆认证

* Client ID

emq-client1

C

* 服务器地址

mqtt://

192.168.200.129

* 端口

1883

用户名

x

密码

.....

SSL/TLS

☐ true

☒ false

此时用户名字段需要输入一个，但是可以随便填写！

2.4 HTTP认证

HTTP 认证使用外部自建 HTTP 应用认证数据源，根据 HTTP API 返回的数据判定认证结果，能够实现复杂的认证鉴权逻辑。启用该功能需要将 `emqx_auth_http` 插件启用，并且修改该插件的配置文件，在里面指定HTTP认证接口的url。`emqx_auth_http` 插件同时还包含了ACL的功能，我们暂时还用不上，通过注释将其禁用。

1: 在Dashboard中开启 `emqx_auth_http` 插件，同时为了避免误判我们可以停止通过username，clientId进行认证的插件 `emqx_auth_clientid`，`emqx_auth_username`

2.4.1 认证原理

EMQ X 在设备连接事件中使用当前客户端相关信息作为参数，向用户自定义的认证服务发起请求查询权限，通过返回的 HTTP **响应状态码** (HTTP statusCode) 来处理认证请求。

- 认证失败：API 返回 4xx 状态码
- 认证成功：API 返回 200 状态码
- 忽略认证：API 返回 200 状态码且消息体 ignore

2.4.2 HTTP 请求信息

HTTP API 基础请求信息，配置证书、请求头与重试规则。

```
1 # etc/plugins/emqx_auth_http.conf
2
3 ## 启用 HTTPS 所需证书信息
4 ## auth.http.ssl.cacertfile = etc/certs/ca.pem
5
6 ## auth.http.ssl.certfile = etc/certs/client-cert.pem
7
```

```

8  ## auth.http.ssl.keyfile = etc/certs/client-key.pem
9
10 ## 请求头设置
11 ## auth.http.header.Accept = */*
12
13 ## 重试设置
14 auth.http.request.retry_times = 3
15 auth.http.request.retry_interval = 1s
16 auth.http.request.retry_backoff = 2.0

```

加盐规则与哈希方法

HTTP 在请求中传递明文密码，加盐规则与哈希方法取决于 HTTP 应用。

2.4.3 认证请求

进行身份认证时，EMQ X 将使用当前客户端信息填充并发起用户配置的认证查询请求，查询出该客户端在 HTTP 服务器端的认证数据。

打开etc/plugins/emqx_auth_http.conf配置文件，通过修改如下内容：修改完成后需要重启EMQX服务

```

1  # etc/plugins/emqx_auth_http.conf
2
3  ## 请求地址
4  auth.http.auth_req = http://192.168.200.10:8991/mqtt/auth
5
6  ## HTTP 请求方法
7  ## Value: post | get | put
8  auth.http.auth_req.method = post
9
10 ## 请求参数
11 auth.http.auth_req.params = clientid=%c,username=%u,password=%P

```

HTTP 请求方法为 GET 时，请求参数将以 URL 查询字符串的形式传递；POST、PUT 请求则将请求参数以普通表单形式提交（content-type 为 x-www-form-urlencoded）。

你可以在认证请求中使用以下占位符，请求时 EMQ X 将自动填充为客户端信息：

- %u：用户名
- %c：Client ID
- %a：客户端 IP 地址
- %r：客户端接入协议
- %P：明文密码
- %p：客户端端口
- %C：TLS 证书公用名（证书的域名或子域名），仅当 TLS 连接时有效
- %d：TLS 证书 subject，仅当 TLS 连接时有效

推荐使用 POST 与 PUT 方法，使用 GET 方法时明文密码可能会随 URL 被记录到传输过程中的服务器日志中。

2.4.4 认证服务开发

创建基于springboot的应用程序： `emq-demo`

1: 相关坐标如下:

```
1 <parent>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-parent</artifactId>
4   <version>2.3.0.RELEASE</version>
5   <relativePath/> <!-- lookup parent from repository -->
6 </parent>
7 <groupId>com.itheima</groupId>
8 <artifactId>emq-demo</artifactId>
9 <version>0.0.1-SNAPSHOT</version>
10 <name>emq-demo</name>
11 <description>emq demo演示</description>
12
13 <properties>
14   <java.version>1.8</java.version>
15 </properties>
16
17 <dependencies>
18   <dependency>
19     <groupId>org.springframework.boot</groupId>
20     <artifactId>spring-boot-starter-web</artifactId>
21   </dependency>
22
23   <dependency>
24     <groupId>org.springframework.boot</groupId>
25     <artifactId>spring-boot-devtools</artifactId>
26     <scope>runtime</scope>
27     <optional>true</optional>
28   </dependency>
29   <dependency>
30     <groupId>org.springframework.boot</groupId>
31     <artifactId>spring-boot-configuration-processor</artifactId>
32     <optional>true</optional>
33   </dependency>
34   <dependency>
35     <groupId>org.springframework.boot</groupId>
36     <artifactId>spring-boot-starter-test</artifactId>
37     <scope>test</scope>
38     <exclusions>
39       <exclusion>
40         <groupId>org.junit.vintage</groupId>
41         <artifactId>junit-vintage-engine</artifactId>
42       </exclusion>
43     </exclusions>
44   </dependency>
45 </dependencies>
46
47 <build>
48   <plugins>
49     <plugin>
50       <groupId>org.springframework.boot</groupId>
51       <artifactId>spring-boot-maven-plugin</artifactId>
```

```
52     </plugin>
53 </plugins>
54 </build>
```

2: 创建application.yml配置文件并配置

```
1 server:
2   port: 8991
3 spring:
4   application:
5     name: emq-demo
```

3: 创建Controller: com.itheima.controller.mqtt.AuthController; 编写如下

```
1  /**
2   * Created by 传智播客*黑马程序员.
3   */
4  @RestController
5  @RequestMapping("/mqtt")
6  public class AuthController {
7
8      private Logger log = LoggerFactory.getLogger(AuthController.class);
9
10     private Map<String,String> users;
11
12     @PostConstruct
13     public void init(){
14         users = new HashMap<>();
15         users.put("user", "123456"); //实际的密码应该是密文, mqtt的http认证组件传输过来的密码是明文, 我们需要自己进行加密验证
16         users.put("emq-client2", "123456");
17         users.put("emq-client3", "123456");
18     }
19
20     @PostMapping("/auth")
21     public ResponseEntity<?> auth(@RequestParam("clientid") String clientid,
22                                   @RequestParam("username") String username,
23                                   @RequestParam("password") String password){
24         log.info("emqx认证组件调用自定义的认证服务开始认证, clientid={}, username={}, password={}", clientid, username, password);
25         //在此处可以进行复杂的认证逻辑, 但是我们为了演示方便做一个固定操作
26         String value = users.get(username);
27         if(StringUtils.isEmpty(value)){
28             return new ResponseEntity<Object>(HttpStatus.UNAUTHORIZED);
29         }
30         if(!value.equals(password)){
31             return new ResponseEntity<Object>(HttpStatus.UNAUTHORIZED);
32         }
33         return new ResponseEntity<Object>(HttpStatus.OK);
34     }
35 }
```


2.4.5 MQTTX客户端验证

使用MQTTX客户端工具连接EMQX服务器，如下

[< 返回](#)[新建](#)[连接](#)

基础

* 名称

测试http登陆认证

* Client ID

mqttx_8c5ac831

C

* 服务器地址

mqtt://

192.168.200.129

* 端口

1883

用户名

emq-client2

密码

.....

SSL/TLS

☐ true

☒ false

这个地方的Client-ID随便输入，因为在验证的代码里没有对该字段做校验，之后点连接，发现会连接成功，然后可以去自定义的认证服务中查看控制台输出，证明基于外部的http验证接口生效了。在实际项目开发过程中，HTTP接口校验的代码不会这么简单，账号和密码之类的数据肯定会存在后端数据库中，代码会通过传入的数据和数据库中的数据做校验，如果成功才会校验成功，否则校验失败。

当然EMQ X除了支持我们之前讲过的几种认证方式外，还支持其他的认证方式，比如：MySQL认证、PostgreSQL认证、Redis认证、MongoDB认证，对于其他这些认证方式只需要开启对应的EMQ X插件并且配置对应的配置文件，将对应的数据保存到相应的数据源即可。

3. 客户端SDK

在实际项目中我们要针对MQTT消息代理服务端，从而向其发布消息、订阅消息等来完成我们自己的业务逻辑的开发。EMQ X针对不同的客户端语言都提供了不同的SDK工具包，可以在官网上查看并下载：

<https://www.emqx.io/cn/products/broker>

3.1 Eclipse Paho Java

3.1.1 Paho介绍

Paho Java客户端是用Java编写的MQTT客户端库，用于开发在JVM或其他Java兼容平台（例如Android）上运行的应用程序。

Paho不仅可以对接EMQ X Broker，还可以对接满足符合MQTT协议规范的消息代理服务端，目前Paho可以支持到MQTT5.0以下版本。MQTT3.3.1协议版本基本能满足百分之九十多的接入场景。

Paho Java客户端提供了两个API：

1: MqttAsyncClient提供了一个完全异步的API，其中活动的完成是通过注册的回调通知的。

2: MqttClient是MqttAsyncClient周围的同步包装器，在这里，功能似乎与应用程序同步。

3.1.2 Paho实现消息收发

(1) 找到项目：emq-demo，添加坐标依赖

```
1 <dependency>
2   <groupId>org.eclipse.paho</groupId>
3   <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
4   <version>1.2.2</version>
5 </dependency>
```

(2) 编写客户端封装类的代码：com.itheima.mqtt.client.EmqClient

```
1 /**
2  * Created by 传智播客*黑马程序员.
3  */
4 @Component
5 public class EmqClient {
6
7     private Logger log = LoggerFactory.getLogger(EmqClient.class);
8
9     private IMqttClient mqttClient;
10
11     @Autowired
12     private MqttProperties mqttProperties;
13
14     @Autowired
15     private MqttCallback mqttCallback;
16
17     @PostConstruct
18     private void init(){
19         //MqttClientPersistence是接口 实现类有: MqttDefaultFilePersistence; MemoryPersistence
20         MqttClientPersistence memoryPersistence = new MemoryPersistence();
21         try {
22             mqttClient = new
23 MqttClient(mqttProperties.getBrokerUrl(),mqttProperties.getClientId(),memoryPersistence);
24         } catch (MqttException e) {
25             log.error("MqttClient初始化失败,brokerurl={},clientId=
26 {} ",mqttProperties.getBrokerUrl(),mqttProperties.getClientId());
27         }
28     }
29
30     /**
31      * 连接broker
32      * @param username
33      * @param password
34      */
35     public void connect(String username,String password){
36         //创建MQTT连接选项对象--可配置mqtt连接相关选项
37         MqttConnectOptions connectOptions = new MqttConnectOptions();
38
39         //自动重连
```

```

37     connectOptions.setAutomaticReconnect(true);
38     /**
39      * 设置为true后意味着：客户端断开连接后emq不保留会话保留会话，否则会产生订阅共享队列的存活
客户端收不到消息的情况
40      * 因为断开的连接还被保留的话，emq会将队列中的消息负载到断开但还保留的客户端，导致存活的客户
端收不到消息
41      * 解决该问题有两种方案：1.连接断开后不要保持；2.保证每个客户端有固定的clientId
42      */
43     connectOptions.setCleanSession(true);
44     connectOptions.setUsername(username);
45     connectOptions.setPassword(password.toCharArray());
46     //设置mqtt消息回调
47     mqttClient.setCallback(mqttCallback);
48     //连接broker
49     try {
50         mqttClient.connect(connectOptions);
51     } catch (MqttException e) {
52         log.error("连接mqtt broker失败,失败原因:{",e.getMessage());
53     }
54
55 }
56
57 /**
58  * 发布
59  * @param topic
60  * @param msg
61  */
62 public void publish(String topic, String msg, QosEnum qos, boolean retain){
63     MqttMessage mqttMessage = new MqttMessage();
64     mqttMessage.setQos(qos.value());
65     mqttMessage.setRetained(retain);
66     mqttMessage.setPayload(msg.getBytes());
67     if(mqttClient.isConnected()){
68         try {
69             mqttClient.publish(topic,mqttMessage);
70         } catch (MqttException e) {
71             log.error("mqtt消息发布失败,topic={},msg={},qos={},retain={},errorMsg=
{}",topic,msg,qos,retain,e.getMessage());
72         }
73     }
74 }
75
76 /**
77  * 订阅
78  * @param topicFilter
79  * @return
80  */
81 public void subscribe(String topicFilter,QosEnum qos){
82     try {
83         mqttClient.subscribe(topicFilter,qos.value());
84     } catch (MqttException e) {
85
86         log.error("订阅失败,topicfilter={},qos={},errorMsg=

```

```

    {"",topicFilter,qos,e.getMessage());
86     }
87 }
88
89 /**
90  * 断开连接
91  */
92 @PreDestroy
93 public void disconnect(){
94     try {
95         mqttClient.disconnect();
96     } catch (MqttException e) {
97         log.error("断开连接出现异常,errormsg={}",e.getMessage());
98     }
99 }
100
101 }

```

需要在application.yml中添加自定义的配置:

```

1 mqtt:
2   broker-url: tcp://192.168.200.129:1883
3   client-id: demo-client
4   username: user
5   password: 123456

```

同时需要创建属性配置类来加载该配置数据, 创建: com.itheima.mqtt.properties.MqttProperties

```

1 /**
2  * Created by 传智播客*黑马程序员.
3  */
4 @Configuration
5 @ConfigurationProperties(prefix = "mqtt")
6 public class MqttProperties {
7
8     private String brokerUrl;
9
10    private String clientId;
11
12    private String username;
13
14    private String password;
15
16
17    public String getBrokerUrl() {
18        return brokerUrl;
19    }
20
21    public void setBrokerUrl(String brokerUrl) {
22        this.brokerUrl = brokerUrl;
23    }

```

```

24
25     public String getClientId() {
26         return clientId;
27     }
28
29     public void setClientId(String clientId) {
30         this.clientId = clientId;
31     }
32
33     public String getUsername() {
34         return username;
35     }
36
37     public void setUsername(String username) {
38         this.username = username;
39     }
40
41     public String getPassword() {
42         return password;
43     }
44
45     public void setPassword(String password) {
46         this.password = password;
47     }
48
49     @Override
50     public String toString() {
51         return "MqttProperties{" +
52             "brokerUrl='" + brokerUrl + '\'' +
53             ", clientId='" + clientId + '\'' +
54             ", username='" + username + '\'' +
55             ", password='" + password + '\'' +
56             '}';
57     }
58 }

```

还需创建QoS服务之类枚举：com.itheima.mqtt.enums.QosEnum

```

1  /**
2   * Created by 传智播客*黑马程序员.
3   */
4  public enum QosEnum {
5
6      QoS0(0), QoS1(1), QoS2(2);
7
8      QosEnum(int qos) {
9          this.value = qos;
10     }
11
12     private final int value;
13
14     public int value(){

```

```

15         return this.value;
16     }
17 }

```

(3) 在连接接收到消息之后，我们需要将消息传入消息回调：com.itheima.mqtt.client.MessageCallback

```

1  /**
2   * Created by 传智播客*黑马程序员.
3   */
4  @Component
5  public class MessageCallback implements MqttCallback {
6
7      private Logger log = LoggerFactory.getLogger(MessageCallback.class);
8
9      @Override
10     public void connectionLost(Throwable cause) {
11         //丢失对服务端的连接后触发该方法回调，此处可以做一些特殊处理，比如重连
12         log.info("丢失了对broker的连接");
13     }
14
15     /**
16      * 订阅到消息后的回调
17      * 该方法由mqtt客户端同步调用,在此方法未正确返回之前，不会发送ack确认消息到broker
18      * 一旦该方法向外抛出了异常客户端将异常关闭，当再次连接时；所有QoS1,QoS2且客户端未进行ack确认的
消息都将由
19      * broker服务器再次发送到客户端
20      * @param topic
21      * @param message
22      * @throws Exception
23      */
24     @Override
25     public void messageArrived(String topic, MqttMessage message) throws Exception {
26         log.info("订阅到了消息;topic={},messageid={},qos={},msg={}",
27             topic,
28             message.getId(),
29             message.getQos(),
30             new String(message.getPayload()));
31     }
32
33     /**
34      * 消息发布完成且收到ack确认后的回调
35      * QoS0：消息被网络发出后触发一次
36      * QoS1：当收到broker的PUBACK消息后触发
37      * QoS2：当收到broker的PUBCOMP消息后触发
38      * @param token
39      */
40     @Override
41     public void deliveryComplete(IMqttDeliveryToken token) {
42         int messageId = token.getMessageId();
43         String[] topics = token.getTopics();
44         log.info("消息发送完成,messageId={},topics={}",messageId,topics);
45     }

```

```
46 }
```

(4) 编写消息发布和订阅的测试，在启动类中添加如下代码

```
1  @Autowired
2  private EmqClient emqClient;
3
4  @Autowired
5  private MqttProperties mqttProperties;
6
7
8  @PostConstruct
9  public void init(){
10     emqClient.connect(mqttProperties.getUsername(),mqttProperties.getPassword());
11     //订阅某一主题
12     emqClient.subscribe("testtopic/#", QosEnum.QoS2);
13     //开启一个新的线程向该主题发送消息
14     new Thread()->{
15         while (true){
16             emqClient.publish("testtopic/123", "mqtt msg:"+
17                 LocalDateTime.now().format(DateTimeFormatter.ISO_DATE_TIME),QosEnum.QoS2,false);
18             try {
19                 TimeUnit.SECONDS.sleep(5);
20             } catch (InterruptedException e) {
21                 e.printStackTrace();
22             }
23         }
24     }.start();
25 }
```

(5) 测试：在Dashboard中开启使用username进行认证的组件，其他组件停止即可，然后启动项目，查看控制台输出即可

3.2 MQTT.js

MQTT.js是MQTT协议的客户端JS库，是用JavaScript为node.js和浏览器编写的。

GitHub项目地址：<https://github.com/mqttjs/MQTT.js>

3.2.1 API列表

- mqtt.**connect()**
- mqtt.**Client()**
- mqtt.Client.**publish()**
- mqtt.Client.**subscribe()**
- mqtt.Client.**unsubscribe()**
- mqtt.Client.**end()**
- mqtt.Client.**removeOutgoingMessage()**
- mqtt.Client.**reconnect()**
- mqtt.Client.**handleMessage()**
- mqtt.Client.**connected**

- mqtt.Client***reconnecting**
- mqtt.Client***getLastMessageId()**
- mqtt.**Store()**
- mqtt.Store***put()**
- mqtt.Store***del()**
- mqtt.Store***createStream()**
- mqtt.Store***close()**

直接从官方项目中查看

3.2.2 MQTT.js实现消息收发

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>mqtt.js测试</title>
6      <style type="text/css">
7          div{
8              width: 300px;
9              height: 300px;
10             float: left;
11             border: red solid 1px;
12         }
13     </style>
14     <script src="https://cdn.staticfile.org/jquery/1.10.2/jquery.min.js" ></script>
15
16     <script src="https://unpkg.com/mqtt/dist/mqtt.min.js" ></script>
17     <script type="text/javascript">
18         $(function () {
19             //页面加载完成后
20             // 连接选项
21             const options = {
22                 clean: true, // 保留回话
23                 connectTimeout: 4000, // 超时时间
24                 // 认证信息
25                 clientId: 'emqx_client_h5',
26                 username: 'user',
27                 password: '123456',
28             }
29
30             // 连接字符串, 通过协议指定使用的连接方式
31             // ws 未加密 WebSocket 连接 8083端口
32             // wss 加密 WebSocket 连接 8084端口
33             // mqtt 未加密 TCP 连接
34             // mqtts 加密 TCP 连接
35             // wxs 微信小程序连接
36             // alis 支付宝小程序连接
37             const connectUrl = 'ws://192.168.200.129:8083/mqtt'
38             const client = mqtt.connect(connectUrl, options)
39
40             /**

```



```

41      * mqtt.Client相关事件
42      */
43      //当重新连接启动触发回调
44      client.on('reconnect', () => {
45          $("#div1").text("正在重连.....");
46      });
47      //连接断开后触发的回调
48      client.on("close",function () {
49          $("#div1").text("客户端已断开连接.....");
50      });
51      //从broker接收到断开连接的数据包后发出。MQTT 5.0特性
52      client.on("disconnect",function (packet) {
53          $("#div1").text("从broker接收到断开连接的数据包....."+packet);
54      });
55      //客户端脱机下线触发回调
56      client.on("offline",function () {
57          $("#div1").text("客户端脱机下线.....");
58      });
59      //当客户端无法连接或出现错误时触发回调
60      client.on("error", (error) =>{
61          $("#div1").text("客户端出现错误....."+error);
62      });
63
64      //以下两个事件监听粒度细
65
66      //当客户端发送任何数据包时发出。这包括published()包以及MQTT用于管理订阅和连接的包
67      client.on("packetsend", (packet)=>{
68          $("#div1").text("客户端已发出数据包....."+packet);
69      });
70      //当客户端接收到任何数据包时发出。这包括来自订阅主题的信息包以及MQTT用于管理订阅和连接
71      的信息包
72      client.on("packetreceive", (packet)=>{
73          $("#div1").text("客户端已收到数据包....."+packet);
74      });
75
76      //成功连接后触发的回调
77      client.on("connect",function (connack) {
78          $("#div1").text("成功连接上服务器"+new Date());
79
80          //订阅某主题
81          /**
82           * client.subscribe(topic/topic array/topic object, [options], [callback])
83           * topic:一个string类型的topic或者一个topic数组,也可以是一个对象
84           * options
85           */
86          client.subscribe("testtopic/#",{qos:2});
87
88          //每隔2秒发布一次数据
89          setInterval(publish,2000)
90      });
91
92      function publish() {

```

```

93         //发布数据
94         /**
95          * client.publish(topic,message,[options], [callback])
96          *
97          * message: Buffer or String
98          * options:{
99          *     qos:0, //默认0
100         *     retain:false, //默认false
101         *     dup:false, //默认false
102         *     properties:{}
103         * }
104         * callback:function (err){}
105         */
106         const message = "h5 message "+Math.random()+new Date();
107         client.publish("testtopic/123",message,{qos:2});
108         $("#div2").text("客户端发布了数据:"+message);
109     }
110
111
112     //当客户端接收到发布消息时触发回调
113     /**
114     * topic:收到的数据包的topic
115     * message:收到的数据包的负载payload
116     * packet:收到的数据包
117     */
118     client.on('message', (topic, message,packet) => {
119         $("#div3").text("客户端收到订阅消息,topic="+topic+";消息数据:"+message+";数据
120         包:"+packet);
121     });
122
123     //页面离开自动断开连接
124     $(window).bind("beforeunload",()=>{
125         $("#div1").text("客户端窗口关闭,断开连接");
126         client.disconnect();
127     })
128 })
129 </script>
130 </head>
131 <body>
132     <div id="div1"></div>
133     <div id="div2"></div>
134     <div id="div3"></div>
135 </body>
136 </html>

```

测试：启动项目前将启动类 `EmqDemoApplication` 中init方法上的注解注释掉，启动后访问如下地址查看网页端的输出

<http://localhost:8991/index.html>

4. 日志与追踪

4.1 控制日志输出

EMQ X 支持将日志输出到控制台或者日志文件，或者同时使用两者。可在 `emqx.conf` 中配置：

```
1 log.to = both
```

`log.to` 默认值是 `both`，可选的值为：

- **off**: 完全关闭日志功能
- **file**: 仅将日志输出到文件
- **console**: 仅将日志输出到标准输出(emqx 控制台)
- **both**: 同时将日志输出到文件和标准输出(emqx 控制台)

4.2 日志级别

EMQ X 的日志分 8 个等级，由低到高分别为：

```
1 debug < info < notice < warning < error < critical < alert < emergency
```

EMQ X 的默认日志级别为 `warning`，可在 `emqx.conf` 中修改：

```
1 log.level = warning
```

此配置将所有 log handler 的配置设置为 `warning`。

4.3 日志文件和日志滚动

EMQ X 的默认日志文件目录在 `./log` (zip包解压安装) 或者 `/var/log/emqx` (二进制包安装)。可在 `emqx.conf` 中配置：

```
1 log.dir = log
```

在文件日志启用的情况下 (`log.to = file` 或 `both`)，日志目录下会有如下几种文件：

- **emqx.log.N**: 以 `emqx.log` 为前缀的文件为日志文件，包含了 EMQ X 的所有日志消息。比如 `emqx.log.1`, `emqx.log.2` ...
- **emqx.log.siz** 和 **emqx.log.idx**: 用于记录日志滚动信息的系统文件。
- **run_erl.log**: 以 `emqx start` 方式后台启动 EMQ X 时，用于记录启动信息的系统文件。
- **erlang.log.N**: 以 `erlang.log` 为前缀的文件为日志文件，是以 `emqx start` 方式后台启动 EMQ X 时，控制台日志的副本文件。比如 `erlang.log.1`, `erlang.log.2` ...

可在 `emqx.conf` 中修改日志文件的前缀，默认为 `emqx.log`：

```
1 log.file = emqx.log
```

EMQ X 默认在单日志文件超过 10MB 的情况下，滚动日志文件，最多可有 5 个日志文件：第 1 个日志文件为 `emqx.log.1`，第 2 个为 `emqx.log.2`，并以此类推。当最后一个日志文件也写满 10MB 的时候，将从序号最小的日志的文件开始覆盖。文件大小限制和最大日志文件个数可在 `emqx.conf` 中修改：

```
1 log.rotation.size = 10MB
2 log.rotation.count = 5
```

4.4 针对日志级别输出日志文件

如果想把大于或等于某个级别的日志写入到单独的文件，可以在 `emqx.conf` 中配置 `log..file`：将 `info` 及 `info` 以上的日志单独输出到 `info.log.N` 文件中：

```
1 log.info.file = info.log
```

将 `error` 及 `error` 以上的日志单独输出到 `error.log.N` 文件中

```
1 log.error.file = error.log
```

4.5 日志格式

可在 `emqx.conf` 中修改单个日志消息的最大字符长度，如长度超过限制则截断日志消息并用 `...` 填充。默认不限制长度：

将单个日志消息的最大字符长度设置为 8192：

```
1 log.chars_limit = 8192
```

日志消息的格式为(各个字段之间用空格分隔)

date time level client_info module_info msg

- **date:** 当地时间的日期。格式为：YYYY-MM-DD
- **time:** 当地时间，精确到毫秒。格式为：hh:mm:ss.ms
- **level:** 日志级别，使用中括号包裹。格式为：[Level]
- **client_info:** 可选字段，仅当此日志消息与某个客户端相关时存在。其格式为：ClientId@Peername 或 ClientId 或 Peername
- **module_info:** 可选字段，仅当此日志消息与某个模块相关时存在。其格式为：[Module Info]
- **msg:** 日志消息内容。格式任意，可包含空格。

日志消息举例 1：

```
1 2020-05-18 16:10:03.872 [debug] <<"mqttjs_9e49354bb3">>@127.0.0.1:57105 [MQTT/WS] SEND
  CONNACK(Q0, R0, D0, AckFlags=0, ReasonCode=0)
```

此日志消息里各个字段分别为：

- **date:** 2020-02-18
- **time:** 16:10:03.872

- **level:** [debug]
- **client_info:** <<"mqttjs_9e49354bb3">>@127.0.0.1:57105
- **module_info:** [MQTT/WS]
- **msg:** SEND CONNACK(Q0, R0, D0, AckFlags=0, ReasonCode=0)

日志消息举例 2:

```
1 2020-05-18 16:10:08.474 [warning] [Alarm Handler] New Alarm: system_memory_high_watermark,
Alarm Info: []
```

此日志消息里各个字段分别为:

- **date:** 2020-02-18
- **time:** 16:10:08.474
- **level:** [warning]
- **module_info:** [Alarm Handler]
- **msg:** New Alarm: system_memory_high_watermark, Alarm Info: []

注意此日志消息中, client_info 字段不存在。

4.6 日志级别和log handlers

EMQ X 使用了分层的日志系统, 在日志级别上, 包括全局日志级别 (primary log level)、以及各 log handler 的日志级别。

```
1 [Primary Level]          -- global log level and filters
2 / \
3 [Handler 1] [Handler 2]  -- log levels and filters at each handler
```

log handler 是负责日志处理和输出的工作进程, 它由 log handler id 唯一标识, 并负有如下任务:

- 接收什么级别的日志
- 如何过滤日志消息
- 将日志输出到什么地方

我们来看一下 emqx 默认安装的 log handlers:

```
1 /opt/emqx $ emqx_ctl log handlers list
2 LogHandler(id=ssl_handler, level=debug, destination=console)
3 LogHandler(id=file, level=warning, destination=log/emqx.log)
4 LogHandler(id=default, level=warning, destination=console)
```

- file: 负责输出到日志文件的 log handler。它没有设置特殊过滤条件, 即所有日志消息只要级别满足要求就输出。输出目的地为日志文件。
- default: 负责输出到控制台的 log handler。它没有设置特殊过滤条件, 即所有日志消息只要级别满足要求就输出。输出目的地为控制台。
- ssl_handler: ssl 的 log handler。它的过滤条件设置为当日志是来自 ssl 模块时输出。输出目的地为控制台。

日志消息输出前，首先检查消息是否高于 primary log level，日志消息通过检查后流入各 log handler，再检查各 handler 的日志级别，如果日志消息也高于 handler level，则由对应的 handler 执行相应的过滤条件，过滤条件通过则输出。

设想一个场景，假设 primary log level 设置为 info，log handler `default` (负责输出到控制台) 的级别设置为 debug，log handler `file` (负责输出到文件) 的级别设置为 warning：

- 虽然 console 日志是 debug 级别，但此时 console 日志只能输出 info 以及 info 以上的消息，因为经过 primary level 过滤之后，流到 default 和 file 的日志只剩下 info 及以上的级别；
- emqx.log.N 文件里面，包含了 warning 以及 warning 以上的日志消息。

在日志级别小节中提到的 `log.level` 是修改了全局的日志级别。这包括 primary log level 和各个 handlers 的日志级别，都设置为了同一个值。

Primary Log Level 相当于一个自来水管道系统的总开关，一旦关闭则各个分支管道都不再有水流通过。这个机制保证了日志系统的高性能运作。

4.7 运行时修改日志级别

可以使用 EMQ X 的命令行工具 `emqx_ctl` 在运行时修改 emqx 的日志级别：

修改全局日志级别：

例如，将 primary log level 以及所有 log handlers 的级别设置为 debug：

```
1 $ emqx_ctl log set-level debug
```

修改主日志级别：

例如，将 primary log level 设置为 debug：

```
1 $ emqx_ctl log primary-level debug
```

修改某个log handler的日志级别：

例如，将 log handler `file` 设置为 debug：

```
1 $ emqx_ctl log handlers set-level file debug
```

3.8 日志追踪

EMQ X 支持针对 ClientID 或 Topic 过滤日志并输出到文件。在使用日志追踪功能之前，必须将 primary log level 设置为 debug：

```
1 $ emqx_ctl log primary-level debug
```

开启 ClientID 日志追踪，将所有 ClientID 为 emqx-demo 的日志都输出到 log/my_client.log：

```
1 $ emqx_ctl log primary-level debug
2 debug
3
4 $ emqx_ctl trace start client emq-demo log/emq-demo.log
5 trace clientid emq-demo successfully
```

开启 Topic 日志追踪，将主题能匹配到 'testtopic/#' 的消息发布日志输出到 log/topic_testtopic.log:

```
1 $ emqx_ctl log primary-level debug
2 debug
3
4 $ emqx_ctl trace start topic 'testtopic/#' log/topic_testtopic.log
5 trace topic testtopic/# successfully
```

提示：即使 `emqx.conf` 中，`log.level` 设置为 `error`，使用消息追踪功能仍然能够打印出某 client 或 topic 的 `debug` 级别的信息。这在生产环境中非常有用。