LABCMS软件架构和详细设计说明书

[1. 相关的术语介绍 3](#_Toc22581393)

[2. LABCMS软件架构 4](#_Toc22581394)

[3. MQTT简介以及MQTT broker选型 6](#_Toc22581395)

[3.1 MQTT简介 6](#_Toc22581396)

[3.2 MQTT broker选型 10](#_Toc22581397)

[3.2.1 MQTT broker功能比较 10](#_Toc22581398)

[3.2.2 几种常见的MQTT broker功能比较 13](#_Toc22581399)

[4. LABCMS通信架构 14](#_Toc22581400)

[4.1 设备上报数据的过程 15](#_Toc22581401)

[4.2 Web客户端数据下发的过程 17](#_Toc22581402)

[4.3 Web客户端与服务器前台进程的交互过程 18](#_Toc22581403)

[5. MQTT消息主题设计 19](#_Toc22581404)

[6. MQTT数据包结构设计 21](#_Toc22581405)

[6.1 消息类型message\_type 21](#_Toc22581406)

[6.2 TLV格式的说明 21](#_Toc22581407)

[6.3 Buffer格式说明 21](#_Toc22581408)

[6.4 几个关键问题 21](#_Toc22581409)

[7. 通信模块软件设计 23](#_Toc22581410)

[7.1 通信模块软件的状态机 23](#_Toc22581411)

[7.2 通信模块软件的相关线程 23](#_Toc22581412)

[7.3 通信模块软件的具体流程 23](#_Toc22581413)

[7.4 通信模块软件的订阅和发布的消息主题 24](#_Toc22581414)

[8. 服务器后端进程软件设计 25](#_Toc22581415)

[8.1 服务后端进程软件的状态机 25](#_Toc22581416)

[8.2 MQTT线程处理通信模块消息流程 26](#_Toc22581417)

[8.3 MQTT线程处理web消息流程 26](#_Toc22581418)

[8.4 数据的封装和解析 27](#_Toc22581419)

[8.4.1 普通通信协议的数据封装和解析 27](#_Toc22581420)

[8.4.2程式的数据封装和解析 27](#_Toc22581421)

[8.5 数据库表的设计 29](#_Toc22581422)

[9. 服务器前端进程软件设计 31](#_Toc22581423)

[9.1 设备历史数据的获取 31](#_Toc22581424)

[9.2 设备历史数据文件的解析 32](#_Toc22581425)

[9.3 程式文件的管理和解析 32](#_Toc22581426)

[10. Web软件设计 33](#_Toc22581427)

[11. 开发环境说明 34](#_Toc22581428)

[11.1 操作系统 34](#_Toc22581429)

[Ubuntu 18.04 LTS 34](#_Toc22581430)

[11.2 通信模块的开发环境 34](#_Toc22581431)

[11.3 服务器的开发环境 34](#_Toc22581432)

[12. 相关的参考资料 35](#_Toc22581433)

1. 相关的术语介绍

|  |  |
| --- | --- |
| 术语名称 | 具体含义 |
| LABCMS | 实验室管理信息系统 |
| 设备 | 铁木真高低温设备 |
| 设备控制器 | 铁木真高低温设备控制模块 |
| 通信模块 | 负责将高低温设备数据外发的通信模块 |
| 通信协议 | 泰美科TH系列控制器采用半双工RS485，使用的通信格式 |
| web | 主要是web前端页面 |
| EMQ X | 杭州映云科技有限公司维护的MQTT开源软件 |
| JSON | [JSON](https://baike.baidu.com/item/JSON" \t "_blank)([JavaScript](https://baike.baidu.com/item/JavaScript" \t "_blank) Object Notation, JS 对象简谱) 是一种轻量级的数据交换格式 |

1. LABCMS软件架构

LABCMS的软件架构图如下所示，此图主要说明了各个不同的软件实体所在位置，以及使用的相关技术：



软件部分分为控制器软件，通信模块软件，服务器后端软件，前端软件，以及web上软件等几大部分。几大模块功能如下：

|  |  |
| --- | --- |
| **软件模块** | **功能** |
| **控制器软件** | 工作在现有的设备控制器板块上，大体沿用当前设备的功能，只需要在当前的设备上增加些新的功能即可； |
| **通信模块软件** | 工作在通信模块上，负责采集数据给数据中心，以及接受上层发送过来的数据，传递给设备 |
| **服务器后端软件** | 工作在服务器上，主要功能是通信协议的解析和封装，数据入库操作，以及消息的转发 |
| **服务器前端软件** | 工作在服务器上，接收web 客户端的请求，从数据库中获取相关数据，或者执行历史数据和程式的解析操作，解析出历史数据和程式内容，并返回给web |
| **Web前端软件** | 界面显示和执行用户操作；web可作为MQTT的客户端，通过MQTT与设备进行交互；此外，为了获取相关的数据，通过HTTP与前端服务进程交互，获取数据库中相关信息，或者请求历史数据或者程式文件内容 |

数据流分为

1. 数据发送流
2. 主动查询流
3. 设备控制流

上述三种数据流分别使用不同颜色的有向线条表示，见上述架构图。其中，数据发送流，设备控制流都是在MQTT协议下工作；主动查询流是通过HTTP协议来工作。

通信方式可选WIFI或者有线；服务器可以在本地局域网和云端。通用的通信模块都支持这些功能，只是通信方式不同，只要对通信接口进行适配。软件的核心代码逻辑不变，最大范围内支持客户的使用场景。

为了支持典型的物联网应用场景，采用MQTT通信协议，来支持通信模块，服务器端，和web端的数据交互。MQTT broker驻留在服务器上，负责消息的中转，具体的通信协议框架，见第3节。

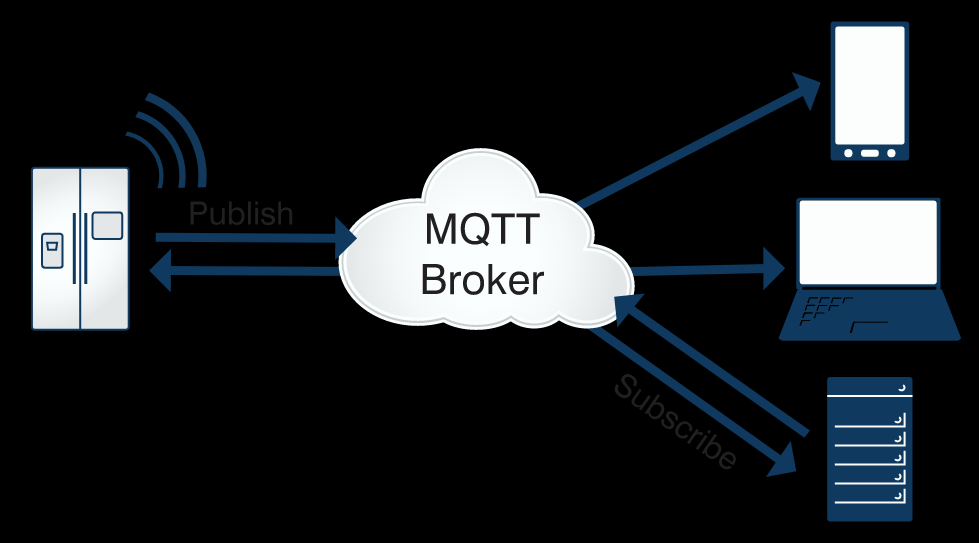
1. MQTT简介以及MQTT broker选型
   1. MQTT简介

<https://www.runoob.com/w3cnote/mqtt-intro.html>

一、简述

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议），是一种基于发布/订阅（publish/subscribe）模式的"轻量级"通讯协议，该协议构建于TCP/IP协议上，由IBM在1999年发布。MQTT最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。关于MQTT的具体介绍，请看中文版：<http://mqtt.p2hp.com/mqtt311>

MQTT是一个基于客户端-服务器的消息发布/订阅传输协议。MQTT协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。



二、设计规范

由于物联网的环境是非常特别的，所以MQTT遵循以下设计原则：

（1）精简，不添加可有可无的功能；

（2）发布/订阅（Pub/Sub）模式，方便消息在传感器之间传递；

（3）允许用户动态创建主题，零运维成本；

（4）把传输量降到最低以提高传输效率；

（5）把低带宽、高延迟、不稳定的网络等因素考虑在内；

（6）支持连续的会话控制；

（7）理解客户端计算能力可能很低；

（8）提供服务质量管理；

（9）假设数据不可知，不强求传输数据的类型与格式，保持灵活性。

三、主要特性

MQTT协议工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议，它具有以下主要的几项特性：

（1）使用发布/订阅消息模式，提供一对多的消息发布，解除应用程序耦合。

这一点很类似于XMPP，但是MQTT的信息冗余远小于XMPP，,因为XMPP使用XML格式文本来传递数据。

（2）对负载内容屏蔽的消息传输。

（3）使用TCP/IP提供网络连接。

主流的MQTT是基于TCP连接进行数据推送的，但是同样有基于UDP的版本，叫做MQTT-SN。这两种版本由于基于不同的连接方式，优缺点自然也就各有不同了。

（4）有三种消息发布服务质量：

"至多一次"，消息发布完全依赖底层TCP/IP网络。会发生消息丢失或重复。这一级别可用于如下情况，环境传感器数据，丢失一次读记录无所谓，因为不久后还会有第二次发送。这一种方式主要普通APP的推送，倘若你的智能设备在消息推送时未联网，推送过去没收到，再次联网也就收不到了。

"至少一次"，确保消息到达，但消息重复可能会发生。

"只有一次"，确保消息到达一次。在一些要求比较严格的计费系统中，可以使用此级别。在计费系统中，消息重复或丢失会导致不正确的结果。这种最高质量的消息发布服务还可以用于即时通讯类的APP的推送，确保用户收到且只会收到一次。

（5）小型传输，开销很小（固定长度的头部是2字节），协议交换最小化，以降低网络流量。

这就是为什么在介绍里说它非常适合"在物联网领域，传感器与服务器的通信，信息的收集"，要知道嵌入式设备的运算能力和带宽都相对薄弱，使用这种协议来传递消息再适合不过了。

（6）使用Last Will和Testament特性通知有关各方客户端异常中断的机制。

Last Will：即遗言机制，用于通知同一主题下的其他设备发送遗言的设备已经断开了连接。

Testament：遗嘱机制，功能类似于Last Will。

四、MQTT协议原理

4.1 MQTT协议实现方式

实现MQTT协议需要客户端和服务器端通讯完成，在通讯过程中，MQTT协议中有三种身份：发布者（Publish）、代理（Broker）（服务器）、订阅者（Subscribe）。其中，消息的发布者和订阅者都是客户端，消息代理是服务器，消息发布者可以同时是订阅者。

MQTT传输的消息分为：主题（Topic）和负载（payload）两部分：

（1）Topic，可以理解为消息的类型，订阅者订阅（Subscribe）后，就会收到该主题的消息内容（payload）；

（2）payload，可以理解为消息的内容，是指订阅者具体要使用的内容。

4.2 网络传输与应用消息

MQTT会构建底层网络传输：它将建立客户端到服务器的连接，提供两者之间的一个有序的、无损的、基于字节流的双向传输。

当应用数据通过MQTT网络发送时，MQTT会把与之相关的服务质量（QoS）和主题名（Topic）相关连。

4.3 MQTT客户端

一个使用MQTT协议的应用程序或者设备，它总是建立到服务器的网络连接。客户端可以：

（1）发布其他客户端可能会订阅的信息；

（2）订阅其它客户端发布的消息；

（3）退订或删除应用程序的消息；

（4）断开与服务器连接。

4.4 MQTT服务器

MQTT服务器以称为"消息代理"（Broker），可以是一个应用程序或一台设备。它是位于消息发布者和订阅者之间，它可以：

（1）接受来自客户的网络连接；

（2）接受客户发布的应用信息；

（3）处理来自客户端的订阅和退订请求；

（4）向订阅的客户转发应用程序消息。

4.5 MQTT协议中的订阅、主题、会话

一、订阅（Subscription）

订阅包含主题筛选器（Topic Filter）和最大服务质量（QoS）。订阅会与一个会话（Session）关联。一个会话可以包含多个订阅。每一个会话中的每个订阅都有一个不同的主题筛选器。

二、会话（Session）

每个客户端与服务器建立连接后就是一个会话，客户端和服务器之间有状态交互。会话存在于一个网络之间，也可能在客户端和服务器之间跨越多个连续的网络连接。

三、主题名（Topic Name）

连接到一个应用程序消息的标签，该标签与服务器的订阅相匹配。服务器会将消息发送给订阅所匹配标签的每个客户端。

四、主题筛选器（Topic Filter）

一个对主题名通配符筛选器，在订阅表达式中使用，表示订阅所匹配到的多个主题。

五、负载（Payload）

消息订阅者所具体接收的内容。

4.6 MQTT协议中的方法

MQTT协议中定义了一些方法（也被称为动作），来于表示对确定资源所进行操作。这个资源可以代表预先存在的数据或动态生成数据，这取决于服务器的实现。通常来说，资源指服务器上的文件或输出。主要方法有：

（1）Connect。等待与服务器建立连接。

（2）Disconnect。等待MQTT客户端完成所做的工作，并与服务器断开TCP/IP会话。

（3）Subscribe。等待完成订阅。

（4）UnSubscribe。等待服务器取消客户端的一个或多个topics订阅。

（5）Publish。MQTT客户端发送消息请求，发送完成后返回应用程序线程。

五、MQTT协议数据包结构

在MQTT协议中，一个MQTT数据包由：固定头（Fixed header）、可变头（Variable header）、消息体（payload）三部分构成。MQTT数据包结构如下：

（1）固定头（Fixed header）。存在于所有MQTT数据包中，表示数据包类型及数据包的分组类标识。

（2）可变头（Variable header）。存在于部分MQTT数据包中，数据包类型决定了可变头是否存在及其具体内容。

（3）消息体（Payload）。存在于部分MQTT数据包中，表示客户端收到的具体内容。

5.1 MQTT固定头

固定头存在于所有MQTT数据包中，其结构如下：

5.1.1 MQTT数据包类型

位置：Byte 1中bits 7-4。

相于一个4位的无符号值，类型、取值及描述如下：

5.1.2 标识位

位置：Byte 1中bits 3-0。

在不使用标识位的消息类型中，标识位被作为保留位。如果收到无效的标志时，接收端必须关闭网络连接：

（1）DUP：发布消息的副本。用来在保证消息的可靠传输，如果设置为1，则在下面的变长中增加MessageId，并且需要回复确认，以保证消息传输完成，但不能用于检测消息重复发送。

（2）QoS：发布消息的服务质量，即：保证消息传递的次数

Ø00：最多一次，即：<=1

Ø01：至少一次，即：>=1

Ø10：一次，即：=1

Ø11：预留

（3）RETAIN： 发布保留标识，表示服务器要保留这次推送的信息，如果有新的订阅者出现，就把这消息推送给它，如果设有那么推送至当前订阅者后释放。 5.1.3 剩余长度（Remaining Length）

地址：Byte 2。

固定头的第二字节用来保存变长头部和消息体的总大小的，但不是直接保存的。这一字节是可以扩展，其保存机制，前7位用于保存长度，后一部用做标识。当最后一位为1时，表示长度不足，需要使用二个字节继续保存。例如：计算出后面的大小为0

5.2 MQTT可变头

MQTT数据包中包含一个可变头，它驻位于固定的头和负载之间。可变头的内容因数据包类型而不同，较常的应用是作为包的标识：

很多类型数据包中都包括一个2字节的数据包标识字段，这些类型的包有：PUBLISH (QoS > 0)、PUBACK、PUBREC、PUBREL、PUBCOMP、SUBSCRIBE、SUBACK、UNSUBSCRIBE、UNSUBACK。

5.3 Payload消息体

Payload消息体位MQTT数据包的第三部分，包含CONNECT、SUBSCRIBE、SUBACK、UNSUBSCRIBE四种类型的消息：

（1）CONNECT，消息体内容主要是：客户端的ClientID、订阅的Topic、Message以及用户名和密码。

（2）SUBSCRIBE，消息体内容是一系列的要订阅的主题以及QoS。

（3）SUBACK，消息体内容是服务器对于SUBSCRIBE所申请的主题及QoS进行确认和回复。

（4）UNSUBSCRIBE，消息体内容是要订阅的主题。

* 1. MQTT broker选型
     1. MQTT broker功能比较

下面是一个关于各种MQTT broker的功能比较：

<https://www.cnblogs.com/saryli/p/9739772.html>

this page attempts to document the features that various MQTT servers (brokers) support. This is specific to their MQTT support; many of these servers have much wider capabilities beyond just MQTT.

| **Server** | **QoS 0** | **QoS 1** | **QoS 2** | **auth** | **[bridge](https://github.com/mqtt/mqtt.github.io/wiki/bridge_protocol)** | **[$SYS](https://github.com/mqtt/mqtt.github.io/wiki/conventions" \l "%24sys)** | **SSL** | **[dynamic topics](https://github.com/mqtt/mqtt.github.io/wiki/are_topics_dynamic)** | **cluster** | **websockets** | **plugin system** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [2lemetry](http://2lemetry.com/platform/) | ✔ | ✔ | ✔ | ✔ | ✔ | § | ✔ | ✔ | ✔ | ✔ | ✘ |
| [Apache ActiveMQ](http://activemq.apache.org/) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [Apache ActiveMQ Artemis](http://activemq.apache.org/artemis) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [Bevywise IoT Platform](https://www.bevywise.com/iot-platform/) | ✔ | ✔ | ✔ | ✔ | rm | ✔ | ✔ | ✔ | ✔ | ✔ | rm |
| [emitter](https://github.com/emitter-io/emitter) | ✔ | § | ✘ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ |
| [emqttd](http://emqtt.io/) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [flespi](https://flespi.com/mqtt-broker) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ |
| [GnatMQ](https://github.com/ppatierno/gnatmq) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✘ | ✘ | ✘ |
| [HBMQTT](https://github.com/beerfactory/hbmqtt) | ✔ | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ |
| [HiveMQ](http://www.hivemq.com/) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [IBM MessageSight](http://www-03.ibm.com/software/products/en/messagesight/) | ✔ | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ | ✔ | § | ✔ | ✘ |
| [JoramMQ](http://mqtt.jorammq.com/) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [Mongoose](https://github.com/cesanta/mongoose) | ✔ | ✔ | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| [moquette](https://github.com/andsel/moquette) | ✔ | ✔ | ✔ | ✔ | ? | ? | ✔ | ? | rm | ✔ | ✘ |
| [mosca](https://github.com/mqtt/mqtt.github.io/wiki/mosca) | ✔ | ✔ | ✘ | ✔ | ? | ? | ? | ? | ✘ | ✔ | ✘ |
| [mosquitto](https://github.com/mqtt/mqtt.github.io/wiki/mosquitto_message_broker) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | § | ✔ | ✔ |
| [MQTT.js](https://github.com/mqttjs/MQTT.js) | ✔ | ✔ | ✔ | § | ✘ | ✘ | ✔ | ✔ | ✘ | ✔ | ✘ |
| [MqttWk](https://github.com/Wizzercn/MqttWk) | ✔ | ✔ | ✔ | ✔ | ✔ | ? | ✔ | ✔ | ✔ | ✔ | ✘ |
| [RabbitMQ](http://www.rabbitmq.com/blog/2012/09/12/mqtt-adapter/) | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ | ✔ | ✔ | ? | ? | ? |
| [RSMB](https://github.com/mqtt/mqtt.github.io/wiki/Really-Small-Message-Broker) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ | ? |
| [Software AG Universal Messaging](http://um.terracotta.org/" \l "page/%2Fum.terracotta.org%2Funiversal-messaging-webhelp%2Fto-mqttoverview.html%23) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | rm | ✘ |
| [Solace](http://dev.solacesystems.com/tech) | ✔ | ✔ | ✘ | ✔ | § | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ |
| [SwiftMQ](http://www.swiftmq.com/landing/router/index.html) | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ | ✔ | ✘ | ✔ |
| [Trafero Tstack](https://github.com/trafero/tstack) | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✘ | ✘ | ✘ |
| [VerneMQ](https://verne.mq/) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| [WebSphere MQ](http://www-03.ibm.com/software/products/en/wmq/) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ? | ? | ? |

Key: ✔ supported ✘ not supported ? unknown § see limitations rm roadmap (planned)

Limits

* MQTT.js will accept connections with username and password supplied, but do not actually authenticate the connection
* IBM MessageSight supports a High-Availability mode which provides the redundancy advantage of a cluster, but does not support any sort of load balancing for MQTT.
* 2lemetry uses domains, where the first topic segment is the domain name. The $SYS topic space is under the domain (i.e. com.example/$SYS/#)
* Solace does provide a proprietary bridge solution between brokers.
* mosquitto clustering is achieved on backend level (redis, amqp, etc).
* Software AG Universal Messaging provides Active/Active clustering (over a proprietary protocol) and bridging (over a proprietary protocol).

Potentially should add columns to track: LWT; additional protocols (WMQ, AMQP, MQTTs etc)

This is in need of expanding. Please add known information about known brokers to this table and include any known limitations below it.

* + 1. 几种常见的MQTT broker功能比较

通过上述MQTT broker的比较，有两款常用的broker功能比较强大，分别是emqttd和mosquitto。下面从开发者开发的难易程度，以及我们项目的适用性角度来比较上述两个broker。

|  |  |  |  |
| --- | --- | --- | --- |
| **MQTT broker** | **提供的功能** | **开发语言** | **其他说明** |
| **Emqttd** | 1. 除了3.1表格中所列的功能之外，还支持规则引擎，鉴权，流控等高级功能； 2. 支持dashboard功能； 3. <https://www.emqx.io/cn/> | 1. 服务器基于java-paho库的Java 2. 服务器基于paho-mgtt的python 3. 浏览器基于MQTT.js的Javascript | 1. 国内开发，公司维护的开源项目，会比传统的开源项目维护更好； 2. 资料较全，且基本都是中文的； 3. Dashboard非常强大，十分方便排查问题 4. 做了很多强大的功能，方便业务扩展，例如规则引擎 5. 此外提供了很多测试工具，方便使用 |
| **mosquitto** | 1. 是非常轻量级的MQTT broker，提供libmosquitto C语言库； 2. 其他提供了pub/sub命令行 3. <http://mosquitto.org/> | 1. 服务器端使用C语言， 2. 浏览器端需要使用基于MQTT.js的Javascript | 1. 传统的开源项目，维护性略差； 2. 资料全是英文的，有API相关的接口，查阅起来不太方便 3. 简单的工具，pub/sub |

通过上述比较，使用国产的Emqttd的优势明显：

1. 杭州映云科技有限公司维护，相比开源软件，更新会更及时；
2. 全中文文档，开发起来会比较方便；
3. Emqttd基本上是私有iot平台，与阿里云非常像，例如：规则引擎，dashboard，他们提供了非常强大的功能；
4. 也提供了非常丰富的测试工具，方便测试；
5. Emqttd分为企业版和非企业版，非企业版功能稍弱一点，但是能支撑10万级别的物联网项目，已经够绝大部分物联网应用了

基于上述理由，LABCMS选用Emqttd的非企业版EMQ X，是比较理性的选择。

1. LABCMS通信架构

LABCMS的通信架构如下图所示。设备（与通信模块连在一起，可看做是一体），服务器后端进程，web浏览器均作为MQTT协议的client端存在；服务器前端进程主要作用是与DB交互，以及相应web其他操作（解析历史数据文件，和程式文件），不作为MQTT的客户端。MQTT broker是MQTT的中枢，负责消息的转发，驻留在服务器上。MQTT所有的客户端都是通过pub/sub机制来工作，即发布（publish）和订阅（subscribe）。

例如：客户端1（Client 1）对一个主题（topic）感兴趣，会主动订阅这个主题，一旦有其他客户端发布了这个主题，它就能收到这个主题的消息；同时，客户端1也可以发布任何主题的消息，只要是订阅了这个消息的客户端，就会收到这个消息。



从上面这个图可以看出，设备，服务端后端进程，以及web，均作为MQTT的客户端，通过pub/sub实现M2M的通信。由于可能同时有多个web客户端在工作，因此要考虑到这种情况，保证多个客户端web1, web2,… webm能同时与设备进行数据交互。

例如，设备1发布了一个温度/湿度实时数据，服务器后端进程订阅了这个数据，就会收到这个数据，把这个数据解析出来之后，存入数据库，然后再把解析好的数据发布出去，web m客户端订阅了服务器后端进程发布的数据，就能收到这个解析好的数据，显示出来。与之相反，web m客户端下发配置或者动作给设备，则发布相关的配置和动作数据，服务器后端进程订阅了该数据，则会接收到，然后将此数据封装成串口通信协议，发布出去，设备端订阅了这个数据，则会收到，然后进行处理。不同的web客户端包含不同的client id，通过这个来区别不同的web客户端。

上述是MQTT的各客户端之间的通信。 web客户端和服务器前端进程之间的数据交互，则是通过HTTP来进行的。Web客户端通过HTTP请求，发送给服务器前端进程，服务器前端进程进行数据库查询，然后将数据发送给web客户端；另外，web客户端可能会对文件操作，此时，需要将文件上传给服务器前端程序，通过服务器前端程序进行相关的解析操作，然后把数据发送给web客户端。

在使用MQTT发送数据的过程中，数据包的格式也比较重要。设备与服务器之间的数据基本就是TLV和通信协议数据，而服务器与web客户端采用JSON数据格式，这个比较通用，也方便web客户端。在第6章，专门来设计数据的包格式。

此外，对于MQTT的消息主题（topic），需要专门设计，要保证设备，服务器，和web 都能准确，方便订阅到自己关心的数据。这个在第7章专门来设计消息主题。

接下来具体描述各个模块之间的通信过程。

* 1. 设备上报数据的过程



设备n上报一个数据的过程如下：设备n为任意一台设备，web m为任意一个客户端。Web m需要显示设备的实时数据，其实有多种方法，一个是轮询，可以每隔一段时间，去发送数据请求给服务器前端进程，去轮询是否有数据更新；一种，就是设备主动上报数据给web，一旦有数据，就发送给web，web就显示出来。显然，第二种方式会比较合理，减少了交互次数。由于基于websocket的MQTT支持第二种方式，web客户端只要作为MQTT的一个client，只要有数据更新发布，就能接收，就节省了自己主动轮询的过程。具体过程如下：

1. 设备n发布了一个温度/湿度实时数据
2. 服务器后端进程订阅了这设备端个数据，就会收到这个数据，把这个数据解析出来之后，存入数据库，然后再把解析好的数据组织成JSON格式，并发布出去
3. web客户端订阅了服务器后端进程发布的数据，就能收到这个解析好的数据，显示出来。

此处，可能会觉得，为何不能设备端直接发送数据给web端，省了服务端后端进程“倒了一次手”。看似是麻烦了，实际上这里面从设计层面，就隔离了web客户端和设备端，进行了解耦。原因如下：

1. 如果web客户端自己拿到原始数据，那么它自己要解析，在web客户端做这么重的事情，会导致效率低下，放在服务器后端进程上来完成，充分利用服务器强大的计算能力；
2. 如果设备发布的时候，通信模块就直接把通信数据解析出来，那么web客户端就能拿到解析好的数据，看似解决了web端负载重的问题，但是通信模块的压力就大了，通信模块需要做通信协议的解析和封装。通信模块的资源有限，如果它来做，同样也是压力比较大；
3. 那放在服务器上做，设备一旦很多，数据报文交互较多，服务器是否能承受起这个压力？

如果有1000台设备同时连入服务器，采样频率1s，那么也就是1000个数据包的出来。

有一个测试结果：

*硬件环境：  
内存4G  
CPU4核*

*测试方法  
并发测试：192.168.6.156 上用 emqttd\_benchmark 测试 192.168.6.157 上的各MQTT SERVER 并发量  
消息发送测试：本地电脑 用php程序 使用一个客户端连接情况下 发送10万消息 到 192.168.6.157上的各MQTT SERVER。*

*测试时间：1个工作日。*

*默认 retain=0 非持久化消息。*

*QoS0: 最多一次 服务器与 客户端 交互1次 。  
QoS1 :至少一次 服务器与 客户端 交互2次 。  
QoS2:洽好一次 服务器与 客户端 交互4次 。*

*测试结果：  
mosquitto:  
发送消息：QoS0: 18.57秒 cpu:10% ， QoS1: 86.9秒 cpu 10% ， QoS2: 157秒 cpu 10% ， retain=1的各值和retain=0几乎一样 。*

*emqtt ：  
发送消息： QoS0 66秒 cpu: 80%, QoS1 204秒 cpu: 55%  
并发连接 27600 第二次 19000 第3次28200 第4次28200 cpu 70%  
稳定性高*

<https://blog.csdn.net/educast/article/details/78352641>

因此，放在服务器上做通信协议的解包和封装，是比较合适的。

* 1. Web客户端数据下发的过程

Web客户端数据下发，这个过程比上个过程要复杂一点。由于web客户端除了需要进行数据下发，还需要接收设备端返回的结果，过程如下：

1. web客户端发布相关的配置和数据
2. 由于服务器后端进程订阅了web客户端发布的消息，便接收到web客户端发布的数据，该数据是以JSON格式封装，其中，对于一些配置数据，服务器会存进DB；对于，其他设备关心的数据，会转发给设备。
3. 服务器后端进程将设备关心的数据，封装成设备通信协议数据，发布出去
4. 设备端订阅了服务器发布的消息，服务器后端进程一旦发布这个消息，设备端就会收到，进行相关操作，并返回操作结果
5. 服务器后端进程接收到设备端返回的结果，解析出来之后，组装成JSON格式，发布给web端
6. Web端收到结果，显示出来



* 1. Web客户端与服务器前台进程的交互过程

Web客户端与服务器前台进程之间的交互就比较简单了，与传统的web客户端与前端服务进程的交互相同，都是通过http来进行。http基本上都是请求和应答构成。不再赘述。



1. MQTT消息主题设计

MQTT的消息通信，通过消息的订阅和发布来实现，因此，消息主题十分重要，一个好的消息主题，能够简化编程模型。在通信过程中，有几个要素比较重要：

1. 消息的方向，例如，是发给后台服务进程（server），还是设备（device）,或者web前端
2. 由于设备可能有多个，设备id比较重要，用来区分是哪个设备关心的
3. 由于可能有多个前端web同时连接上来，需要用web的client id来区分
4. 数据的类型需要区分，主要有以下几种：data/config/action/status

data表示设备实时数据

config表示配置数据

action表示对设备的控制动作

status表示查询相关信息

订阅主题时，#表示多级主题，+表示单级主题，详见：

<http://mqtt.p2hp.com/mqtt311>

那么组合就有一下几种：

1. 发送给后台服务进程的消息主题：

/to\_server/data

/to\_server/config

/to\_server/action

/to\_server/status

后台服务进程应该订阅 /to\_server/#

1. 发送给设备的消息主题：

/to\_device/device\_id/config

/to\_device/device\_id/action

/to\_device/device\_id/status

其中，device\_id是一个数字，例如1,2,3，……。

如果device\_id是全f，例如0xffffffff，则是发送给所有的device

如果Device的id是1，自己在订阅时，就可以订阅：

/to\_device/1/# 和 /to\_device/0xffffffff/#

这样就可以收到单独发给自己的消息，以及发送该给所有设备的信息。全f主要是方便批量操作。

1. 发送给前端浏览器的消息主题：

/to\_browser/client\_id/data

/to\_browser/client\_id/config

/to\_browser/client\_id /action

/to\_browser/client\_id /status

其中，client\_id为浏览器的id, 也是一个数字，例如1000,1001, ……，全f表示发给所有的浏览器，如果浏览器的id是1000，那么浏览器可以订阅一下消息主题：

/to\_browser/1000/#和/to\_browser/0xffffffff/#

1. *MQTT数据包结构设计*

*每个消息除了消息主题外，还有消息内容。由于设备上使用的是串口通信协议，而浏览器上通常使用的是JSON格式，因此，*不同的通信实体间的数据包格式使用并不一样。为了适应我们的设备，从设备到后端服务进程之间的数据包格式，设计如下：

typedef struct message\_between\_device\_server {

int\_32 message\_version; //版本信息，方便后面做消息兼容，4字节

int\_32 device\_id; //设备id，4字节，用于记录设备

int\_32 device\_type; //设备类型，不同设备所用的通信协议可能不一样

int\_32 client\_id; //浏览器id，4字节，用于记录浏览器

int\_32 message\_type; //消息类型，用于区分通信协议，或者普通的消息

int\_32 req\_or\_ack; //1 请求， 2应答

int\_32 req\_len; //请求的协议长度

int\_32 ack\_len; //应答的协议长度，如果请求，则长度为0

char buff[0]; //变长的数据内容，里面包含请求和应答两部分数据，请求的数据长度为req\_len, 而应答的数据长度为ack\_len

} message\_between\_device\_server;



* 1. 消息类型message\_type

|  |  |
| --- | --- |
| 消息类型 | 含义 |
| 1 | 配置类消息，例如采样频率，采用TLV格式，type-len-value |
| 2 | 普通的通信协议类消息，除程式之外的通信协议数据 |
| 3 | 程式数据，用于程式的数据交互 |

* 1. TLV格式的说明

TLV实际就是type-len-value的说明，用来说明类型，长度，和数值。里面可以多个数据合并在一起。

|  |  |  |
| --- | --- | --- |
| 字段 | 长度 | 说明 |
| Type | 1 Byte | 类型，用来表明具体的配置类型；例如1 采样频率 2 … |
| Length | 4 Bytes | 用来说明数据的长度，数据的长度是可以变化的 |
| Value | 可变的 | 数据的值 |

* 1. Buffer格式说明

请求的协议数据：buff[0]-buff[req\_len-1]，取出这个数据，按照具体的消息类型来解析。

应答的协议数据：buff[req\_len]-buff[req\_len+ack\_len-1]，取出这个数据，按照协议格式单独解析。具体的格式解析，见8.4.

* 1. 几个关键问题

1. 之所以要保存device\_id和client\_id，是方便消息准确回复。例如，web m发送一个消息到设备n, 设备n返回一个应答时，后台服务进程需要知道发送给准确的web，就要通过web的client id了，并同时告知web m是设备 n的应答。
2. 由于消息的大小是变长的，所以设计的buff是变长的
3. Buff中包含了通信协议的数据，既包含了请求，也包含了应答，之所以要包含请求，是方便后台服务进程处理，告诉后台服务进程，发给设备是什么请求，它的应答是什么，这样后台服务进程就能很方便处理；否则，后台服务进程，要处理很多消息，根本不知道发送的什么请求，返回的这个应答
4. 请求消息和应答消息的长度，就分别是req\_len和ack\_len，后台服务进程取到这个消息之后，就按照req\_len长度截取出来，然后按照协议解析，发送的请求是什么；同时，按照ack\_len的长度解析应答。
5. 通信模块软件设计



* 1. 通信模块软件的状态机

通信模块软件的状态机比较简单，就是分为初始化和工作态两种。

1. 初始化，主要就是对通信模块进行各种初始化，使得通信模块能正常工作。初始化主要创建线程，获取设备id，获取MQTT服务器ip（这些信息都通过串口在通信模块上配置），获取必须的配置信息。
2. 工作态，就是软件进入正常流程，一直是处于循环状态，一是定时采样实时数据，二是响应上层发送过来的动作和状态查询消息。
   1. 通信模块软件的相关线程

通信模块主要有以下几个线程：

1. 主线程

主线程主要是做初始化，以及设定状态机，状态机跳转

1. 串口处理线程

串口处理线程，主要是通过485接口与设备打交道，所有与设备打交道都是走串口处理线程，由于485是半双工，串口处理线程需要等待设备返回信息之后，才能往下走

1. MQTT线程

MQTT线程，作为MQTT的客户端，负责MQTT的通信接口，所有MQTT消息交互都有MQTT线程负责

1. 定时器

定时器负责定时采样，定时获取设备的数据信息

* 1. 通信模块软件的具体流程

1. 主线程一上电，就创建串口线程，并设定状态机为初始化
2. 主线程发送初始化事件给串口线程，串口线程获取设备id与MQTT服务器ip地址
3. 串口线程返回上述信息给主线程，主线程根据信息创建MQTT线程
4. MQTT线程获取必须的配置信息（例如采样周期），返回给主线程
5. 主线程创建采样定时器，定时器定时发送采样消息给串口线程
6. 串口线程查询设备数据，并返回给MQTT线程，发布出去
7. MQTT线程监听消息，如果有对设备的动作或者状态查询，则装发给串口线程
8. 串口线程把信息传递给设备，并通过MQTT线程返回结果
   1. 通信模块软件的订阅和发布的消息主题
9. 订阅的消息主题

通信模块只关心本设备的消息，以及组播的消息，因此其订阅的消息格式：

/to\_device/device\_id/# 和 /to\_device/0xffffffff/#

device\_id是从设备那里获取到的设备id；0xffffffff是一个广播地址，是所有的设备都应该获能收到。‘#’是一个通配符，表示后面接任意的主题，一级或者多级都可以。设备关心config/action/status三种类型的消息，因此‘#’可以是这三种的任意一种。

1. 发布的消息主题

由于设备只与服务器后端进程通信，因此设备发布的消息都是/to\_server/。可以是data/config/action/status的任意一种。

1. 服务器后端进程软件设计



* 1. 服务后端进程软件的状态机

服务器后端进程的状态机分为初始化和工作态两种；服务器后端进程主要分为主线程和MQTT线程，其中主线程负责状态机的跳转。两种状态机的说明：

1. 初始化状态机，主要负责MQTT线程的创建，和一些初始化。这里面区分服务器第一次运行，和重启的情况。第一次运行，需要创建相关的数据表，读取默认配置，并将配置入库，默认配置都是xml格式的，方便解析，以及可读性高；重启的时候，配置数据可以直接从数据库中恢复，配置数据是存盘的
2. 工作态状态机，主要是接收通信模块和web发过来的消息，对这些消息进行解析，处理，和封装，这一块工作主要MQTT线程完成。关于这些消息的详细处理流程，在8.2，8.3，和8.4三节详细说明。
   1. MQTT线程处理通信模块消息流程

通信模块发送过来的消息，主要三大类：

1. 一是配置请求，来请求设备的基本配置，例如采样频率等；数据格式是TLV格式
2. 二是设备数据上报，数据格式是设备的通信协议或者程式，需要解析出来
3. 三是查询/操作结果上报，数据格式也是设备的通信协议，需要解析出来

针对这三类消息，需要根据消息的不同类型，分别解析。针对TLV格式，见6.2节；Web消息，见8.3节；通信协议，见8.4节。

* 1. MQTT线程处理web消息流程

Web发过来的数据都是JSON格式的，比较好处理。

[JSON](https://baike.baidu.com/item/JSON" \t "_blank)([JavaScript](https://baike.baidu.com/item/JavaScript" \t "_blank) Object Notation, JS 对象简谱) 是一种轻量级的数据交换格式。它基于[ECMAScript](https://baike.baidu.com/item/ECMAScript" \t "_blank)(欧洲计算机协会制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

这个格式比较通用。用一个简单的例子来说明下，例如一组网站的信息可以组织成：

var sites = [ { "name":"runoob" , "url":"www.runoob.com" },

{ "name":"google" , "url":"www.google.com" },

{ "name":"微博" , "url":"www.weibo.com" }

];

三个网站，每个网站有两个关键属性，一是name，二是url；每个属性由一组key: value对组成，中间以”：”隔离。“{}”之内是一个元素，”[]”之内是一组元素，通过”[]”和“{}”组合，就能表示一个复杂的结构体。

MQTT线程把数据组织成JSON格式，发送给web，web就能快速方便解析出数据，反过来也是一样。

* 1. 数据的封装和解析

8.4.1 普通通信协议的数据封装和解析

一些关键的数据结构定义如下：

通信协议的数据结构：

#define MAX\_DATA\_NUM 100

Typedef struct protocol\_message{

int\_32 device\_id; // 设备id

int\_32 device\_type; // 设备类型

int\_32 req\_or\_ack; // 请求或者应答，向设备请求数据，或从设备获取数据

char command[8]; //命令：AMI/RSD/RRD/WSD/WRD/STD/CLD, 字符串表示

int\_32 result; //应答时返回的结果，请求是忽略此字段

int\_32 data\_num; //数据个数，请求或者返回是的数据个数

int\_32 data[MAX\_DATA\_NUM]; //十六进制数据

}PRO\_MSG;

对于通信协议的封装：

1. 入参：从结构体PRO\_MSG中获取入参
2. 根据入参，按照通信协议的格式，封装出通信协议数据

对于通信协议的解析：

1. 入参：通信数据报文
2. 根据通信协议的格式，对通信数据进行解析
3. 解析出的结果，存放至结构体PRO\_MSG中

8.4.2程式的数据封装和解析

程式的数据结构：

Typedef struct pgm\_data{

Unsigned char name[PEG\_NAME\_SIZE+1];

Struct SgmData{

Float temp;

Float humi;

Unsigned int hour;

Unsigned char min;

Unsigned char sec;

Unsigned char p[P\_NUM];

Unsigned char wt;

}sgm[PEG\_SGM\_SUM];

Float high\_limt;

Float low\_limit;

Short all\_cycle;

Unsigned char link\_pgm;

Struct partcycle{

Unsigned char start;

Unsigned char end;

Short num;

}part\_of\_cycle[4];

}PGM\_DATA\_F;

对程式数据的封装：

1. 入参：从结构体PGM\_DATA\_F中获取入参
2. 根据入参，按照通信协议的格式，封装出通信协议的数据

对程式数据的解析：

1. 入参：通信数据报文
2. 根据通信协议的格式，对通信数据进行解析
3. 解析出的结果，存放至结构体PGM\_DATA\_F中
   1. 数据库表的设计
4. 设备配置表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 设备id | 逻辑id | 采样频率 | 运行模式 | TSV | HSV | 设定程式 | 程式监控时间 | 所属实验室 | 所属用户 | 实验人 | 实验名称 |

没有设置的内容项，显示为空

1. 设备状态表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 设备id | 设备连接状态 | 设备运行状态 | TPV | HPV | 运行模式 | 开始  运行时间 | 已运行时间 | 运行程式名称 | 程式运行总段次 | 程式运行当前段数 | 程式总循环次数 | 当前循环次数 | 当前循环运行时间 |

设备连接状态：指设备通过通信模块连接到服务器的状态，可能是未连接，连接，掉线等状态；

1. 设备历史数据表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 时间 | 设备id | 设备连接状态 | 设备运行状态 | TPV | HPV | 运行模式 | 开始  运行时间 | 已运行时间 | 运行程式名称 | 程式运行总段次 | 程式当前运段次 | 程式总循环次数 | 当前循环次数 | 当前循环运行时间 | 所属实验室 | 所属用户 | 实验人 | 实验名称 |

1. 用户表

|  |  |  |  |
| --- | --- | --- | --- |
| 用户名称 | 用户等级 | 用户权限 | 实验室 |

实验室为用户管理的实验室

1. 当前告警记录表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 告警id | 告警时间 | 告警名称 | 告警具体内容 | 告警恢复步骤 |

当前告警记录表，只存储当前没有消除的告警，不存盘

1. 告警历史记录表

|  |  |  |  |
| --- | --- | --- | --- |
| 告警时间 | 设备id | 告警名称 | 告警动作 |

告警动作分为产生和清除，二者都要存盘，方便历史查询

1. 程式管理表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 程式名称 | 程式序号 | 程式总循环次数 | 高温保护 | 低温保护 | 链接程序号 | 程式段次1的运行时间 | 程式段次1运行设定温度 | 程式段次1运行设定湿度 | 程式段次1时序1 | 程式段次1时序2 | 程式段次1时序3 | 程式段次1计时方式 | …… | 程式段次n的运行时间 | 程式段次n运行设定温度 | 程式段次n运行设定湿度 | 程式段次n时序1 | 程式段次n时序2 | 程式段次n时序3 | 程式段次n计时方式 | 程式部分循环次数 | 程式部分循环1的开始段次 | 程式部分循环1的结束段次 | 程式部分循环1的循环次数 | …… | 程式部分循环m的开始段次 | 程式部分循环m的结束段次 | 程式部分循环m的循环次数 |

1. 用户操作记录表

|  |  |  |
| --- | --- | --- |
| 时间 | 用户 | 动作 |

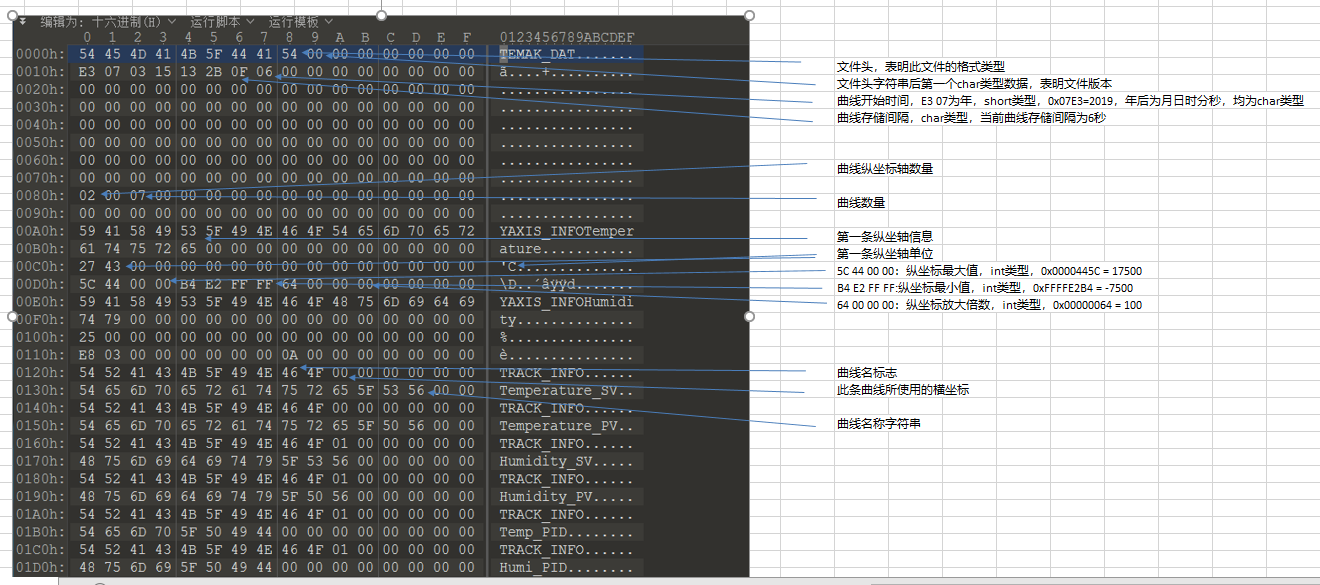
注：数据库的表象中，没有设置的内容项，显示为空

1. 服务器前端进程软件设计
   1. 设备历史数据的获取



服务器前端进程主要是与web交互，执行web的请求。分为主线程和服务线程，服务线程执行实际的操作。服务线程读取数据库，解析历史数据文件，或者解析程式文件，然后将这些数据组织成JSON格式返回给web。

* 1. 设备历史数据文件的解析



上述是一个历史文件的格式分析。Web将文件上传到服务器，服务器前端进程对该文件按上述的格式定义对这个文件进行解析。解析之后，将格式组成JSON格式，发送给web，web对历史数据进行呈现。

* 1. 程式文件的管理和解析

程式文件是程式结构体对应的二进制文件，可以直接倒成结构体，比较简单，不再赘述。

1. Web软件设计

Web主要是页面内容显示，不是编程阶段关注的重点，从编程角度，web中使用MQTT的MQTT.js进行MQTT的通信是一个热点，这个需要关注下。前期demo已经验证了这个功能。

在前期的详细设计阶段，对于web的页面设计暂不做过多的关注。Web需要按照需求说明书，实现相关页面设计。

1. 开发环境说明
   1. 操作系统

Ubuntu 18.04 LTS

* 1. 通信模块的开发环境

通信模块是基于乐鑫（<https://www.espressif.com/zh-hans/products/hardware/esp-wroom-32/overview>）的ESP模块开发的，整个一套环境是基于ESP的开发环境，环境搭建过程：<https://docs.espressif.com/projects/esp-idf/zh_CN/latest/get-started/index.html>

* 1. 服务器的开发环境

|  |
| --- |
| Ubuntu 18.04 LTS |
| Mysql 5.7 |
| Docker 18.03 |
| python 3.1 |
| EMQ X 3.1 |

1. 相关的参考资料

Eclipse Paho JavaScript Client

<https://www.eclipse.org/paho/clients/js/>

Using the Paho Node.js MQTT Client-Starting Guide

<http://www.steves-internet-guide.com/using-node-mqtt-client/>

mosquito

<http://mosquitto.org/>

libwebsocket

<https://libwebsockets.org/git/libwebsockets/>

<https://libwebsockets.org/>

用例

<https://blog.csdn.net/zhushuanghe/article/details/71453556>

<https://www.cnblogs.com/saryli/p/9739772.html>

<http://www.steves-internet-guide.com/mqtt-websockets/>