

一、Linux 系统中的进程之间通信(IPC)

二、基于 Socket 通信的优点

1. 跨主机，具有伸缩性
2. 操作系统会自动回收资源
3. 可记录、可重现
4. 跨语言

三、MQTT 消息总线

1. MQTT 是一个通信的机制
2. MQTT 的实现
3. 在 MQTT 之上，设计自己的通信协议

四、嵌入式系统中如何利用 MQTT 消息总线

1. 一个嵌入式系统的通信框架
2. 稍微复杂一点的通信模型

五、Mosquitto: 一个简单的测试代码

1. 直接通过 apt 来安装、测试
2. 通过源码来手动编译、测试

六、总结

七、资源下载

1. mosquitto-1.4.9.tgz
2. Mosquitto Demo 示例代码

一、Linux 系统中的进程之间通信(IPC)

作为一名嵌入式软件开发人员来说，处理进程之间的通信是很常见的事情。从通信目的的角度来看，我们可以把进程之间的通信分成 3 种：

1. 为了进程的调度：可以通过信号来实现；
2. 为了共享资源：可以通过互斥锁、信号量、读写锁、文件锁等来实现；
3. 为了传递数据：可以通过共享内存、命名管道、消息队列、Socket来实现。

关于上面提到的这些、操作系统为我们提供的通信原语，网络上的各种资料、文章满天飞，在这里就不啰嗦了。在这些方法中应该如何选择呢？根据我个人的经验，贵精不贵多，认真挑选三四样东西就能完全满足日常的工作需要。

我们今天想讨论的问题主要是第 3 个：传递数据，在上面这几种传递数据的方法中，我最喜欢、最常用的就是 Socket 通信。

有些小伙伴可能会说：Socket 通信就是 TCP/IP 的那一套东西，还需要自己管理连接、对数据进行组包、分包，也是挺麻烦的。

没错，Socket 通信本身的确需要手动来处理这些底层的東西，但是我们可以给 Socket 穿上一层“外衣”：利用 MQTT 消息总线，在系统的各进程之间进行数据交互，下面我们就一一道来。

二、基于 Socket 通信的优点

这里我就不自己发挥了，直接引用陈硕老师的那本书《Linux 多线程服务端编程》这本书中的观点(第 65 页，3.4 小节)：

1. 跨主机，具有伸缩性

反正都是多进程了，如果一台机器的处理能力不够，就能用多台主机来处理。把进程分散到同一台局域网的多台机器上，程序改改 Host:Port 配置就能继续用。相反，文章开头部分列出的那些进程之间通信方式都不能跨机器，这就限制了可扩展性。

2. 操作系统会自动回收资源

TCP port 由一个进程独占，当程序意外退出时，操作系统会自动回收资源，不会给系统留下垃圾，程序重启之后能比较容易地恢复。

3. 可记录、可重现

两个进程通过 TCP 通信，如果一个崩溃了，操作系统会关闭连接，另一个进程几乎立刻就能感受到，可以快速 failover。当然应用层的心跳是必不可少的。(补充：操作系统本身对于 TCP 连接有一个保活时间，默认是 2 个小时，而且是针对全局的。)

4. 跨语言

服务端和客户端不必使用同一种编程语言。

1. 陈硕老师描述的是通用的 Socket 通信，因此客户端和服务端一般位于不同的物理机器上。
2. 在嵌入式开发中，一般都是用同一种编程语言，因此，跨语言这个有点可以忽略不计了。

三、MQTT 消息总线

1. MQTT 是一个通信的机制

对物联网领域熟悉的小伙伴，对于 MQTT 消息总线一定非常熟悉，目前几大物联网云平台（亚马孙、阿里云、华为云）都提供了 MQTT 协议的接入方式。

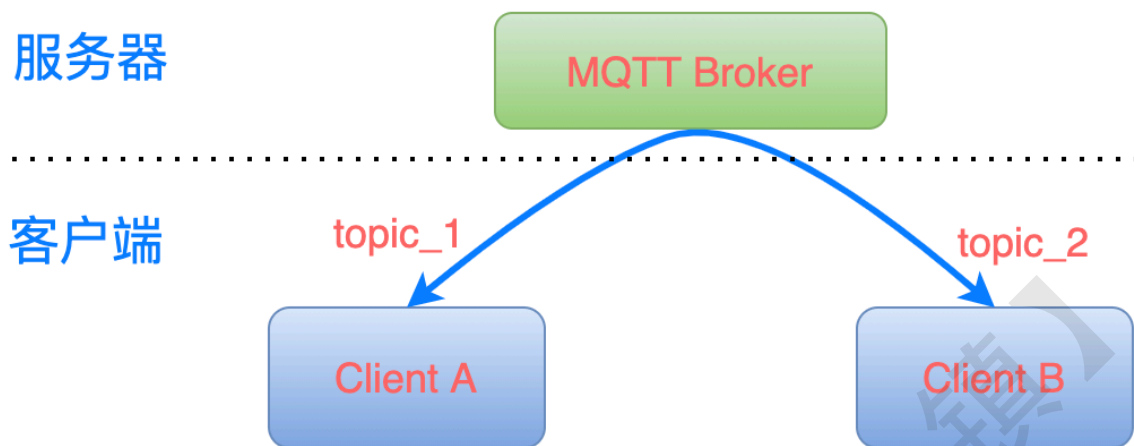
目前，学习 MQTT 最好的文档是 IBM 的在线手册：<https://developer.ibm.com/zh/technologies/messaging/articles/iot-mqtt-why-good-for-iot/>。

这里，我直接把一些重点信息列出来：

1. MQTT 协议轻量、简单、开放和易于实现；
2. MQTT 是基于发布 (Publish)/ 订阅 (Subscribe) 范式的消息协议；
3. MQTT 工作在 TCP/IP 协议族上；

4. 有三种消息发布服务质量;
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;

MQTT 消息传输需要一个中间件, 称为: Broker, 其实也就是一个 Server。通信模型如下:



1. MQTT Broker 需要首先启动;
2. ClientA 和 ClientB 需要连接到 Broker;
3. ClientA 订阅主题 topic_1, ClientB 订阅主题 topic_2;
4. ClientA 往 topic_2 这个主题发送消息, 就会被 ClientB 接收到;
5. ClientB 往 topic_1 这个主题发送消息, 就会被 ClientA 接收到;

基于 topic 主题的通信方式有一个很大的好处就是**解耦**, 一个客户端可以订阅多个 topic, 任何接入到总线的其他客户端都可以往这些 topic 中发送信息(一个客户端发送消息给自己也是可以的)。

2. MQTT 的实现

MQTT 只是一个协议而已, 在 IBM 的在线文档中可以看到, 有很多语言都实现了 MQTT 协议, 包括: C/C++、Java、Python、C#、JavaScript、Go、Objective-C 等等。那么对于嵌入式开发来说, 使用比较多的是这几个实现:

```
Mosquitto;  
Paho MQTT;  
wolfMQTT;  
MQTTRoute。
```

在下面, 我们会重点介绍 Mosquitto 这个开源实现的编译和使用方式, 这也是我在项目中使用最多的。

3. 在 MQTT 之上, 设计自己的通信协议

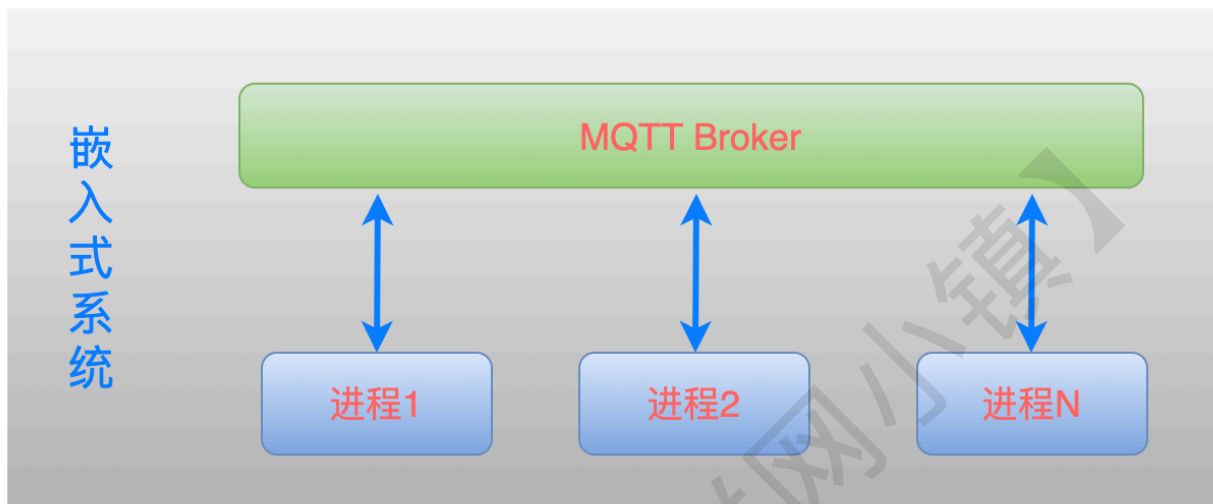
从上面的描述中可以看出, **MQTT 消息总线就是一个通信机制**, 为通信主体提供了一个传递数据的通道而已。

在这个通道之上, 我们可以根据实际项目的需要, 发送**任何格式、编码**的数据。在项目中, 我们最常用的就是 json 格式的纯文本, 这也是各家物联网云平台所推荐的方式。如果在文本数据中需要包含二进制数据, 那就转成 BASE64 编码之后再发送。

四、嵌入式系统中如何利用 MQTT 消息总线

从上面的描述中可以看到，只要在**服务端**运行着一个 MQTT Broker 服务，每个连接到总线的客户端都可以灵活地相互收发数据。

我们可以把这个机制**应用在嵌入式应用程序的设计中**：MQTT Broker 作为一个独立的服务**运行在嵌入式系统本地**，其他需要交互的进程，只要连接到本地的这个 Broker，就可以相互发送数据了。运行模型如下：

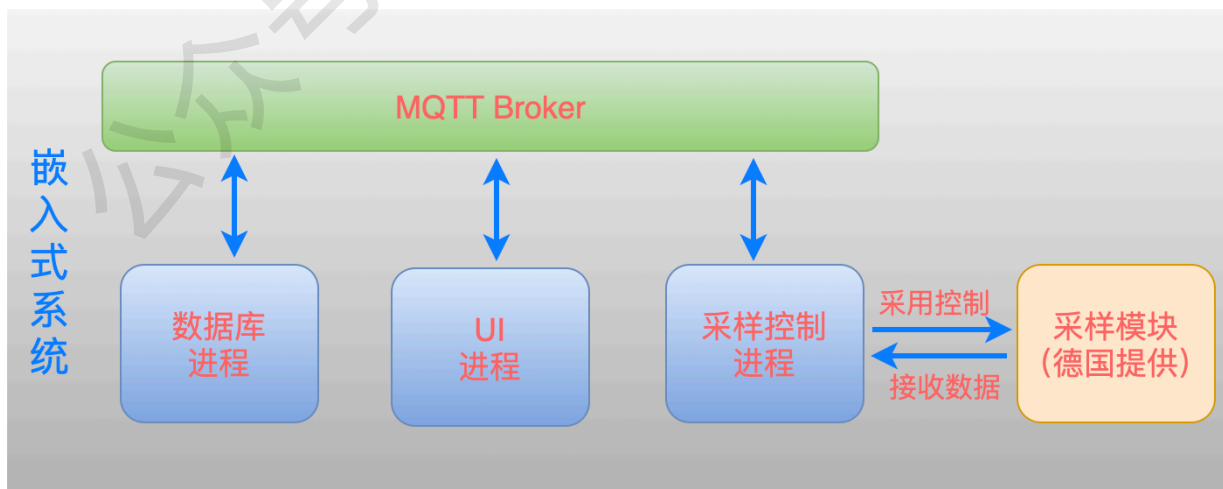


每一个进程只需要**订阅一个固定的 topic**（比如：自己的 client Id），那么其他进程如果想要发送数据给它，就直接发送到这个 topic 即可。

1. 一个嵌入式系统的通信框架

我之前开发过一个环境监测系统，采集大气中的 PM2.5、PM10等污染物参数，在 Contex A8 平台下开发，需要实现**数据记录(数据库)**、**UI 监控界面**等功能。

污染物的数据采样硬件模块是第三方公司提供的，我们只需要通过该模块提供的串口协议去控制采样设备、接收采样数据即可。最终设计的通信模型如下：



1. UI 进程通过消息总线，发送控制指令给采样控制进程，采样控制进程接收到后通过串口发送控制指令给采样模块；
2. 采样控制进程从串口接收采样模块发来的PM2.5等数据后，把所有的数据发送到消息总线上指定

的 topic 中；

3. UI 进程订阅该 topic，接收到数据后，显示在屏幕上；
4. 数据库进程也订阅该 topic，接收到数据后，把数据存储在 SQLite 数据库中；

在这个产品中，**核心进程是采样控制进程**，负责与采样模块的交互。通过把 UI 处理、数据库处理设计成独立的进程，**降低了系统的复杂性，即使这 2 个进程崩溃了，也不会影响到核心的采样控制进程。**

比如：如果 UI 进程出现错误崩溃了，会立刻重启，启动之后通过缓存信息知道此刻正在执行采样工作，于是 UI 进程立刻**连接到消息总线、进入采样数据显示界面，继续接收、显示采样控制进程发出的 PM2.5 等数据。**

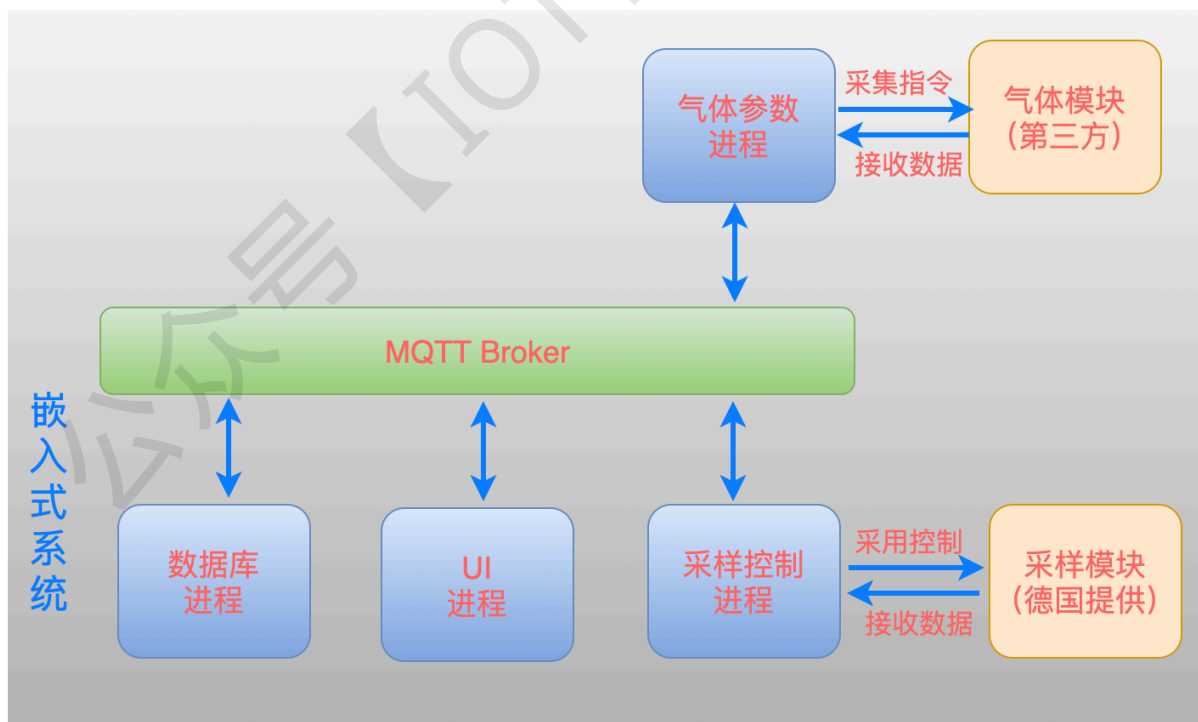
这个通信模型还有另外一个有点：**可扩展性。**

在项目开发的后期，甲方说需要集成一个第三方的**气体模块**，用来采集大气中 NO、SO2 等参数，通信方式是 RS485。

此时扩展这个功能模块就异常简单了，直接写一个**独立的气体参数进程**，**接入到消息总线上**。这个进程通过 RS485，从第三方气体模块接收到 NO、SO2 等气体参数时，直接往消息总线上的某个 topic 一丢，UI 进程、数据库进程订阅这个 topic，就可以立刻接收到气体相关的数据了。

此外，这个设计模型还有其他一些**优点**：

1. 并行开发：每个进程可以由不同的人员并行开发，只要相互之间定义好通信协议即可；
2. 调试方便：由于发送的数据都是 manual readable，在开发阶段，可以在 PC 机上专门写一个监控程序，接入到嵌入式系统中的 MQTT Broker 之后，这样就可以接收到所有进程发出的消息；
3. 通信安全：在产品 release 之后，为了防止其他人偷听数据(比如 2 中的调试进程)，可以为 MQTT Broker 指定一个配置文件，只能允许本地进程(127.0.0.1)连接到消息总线上。

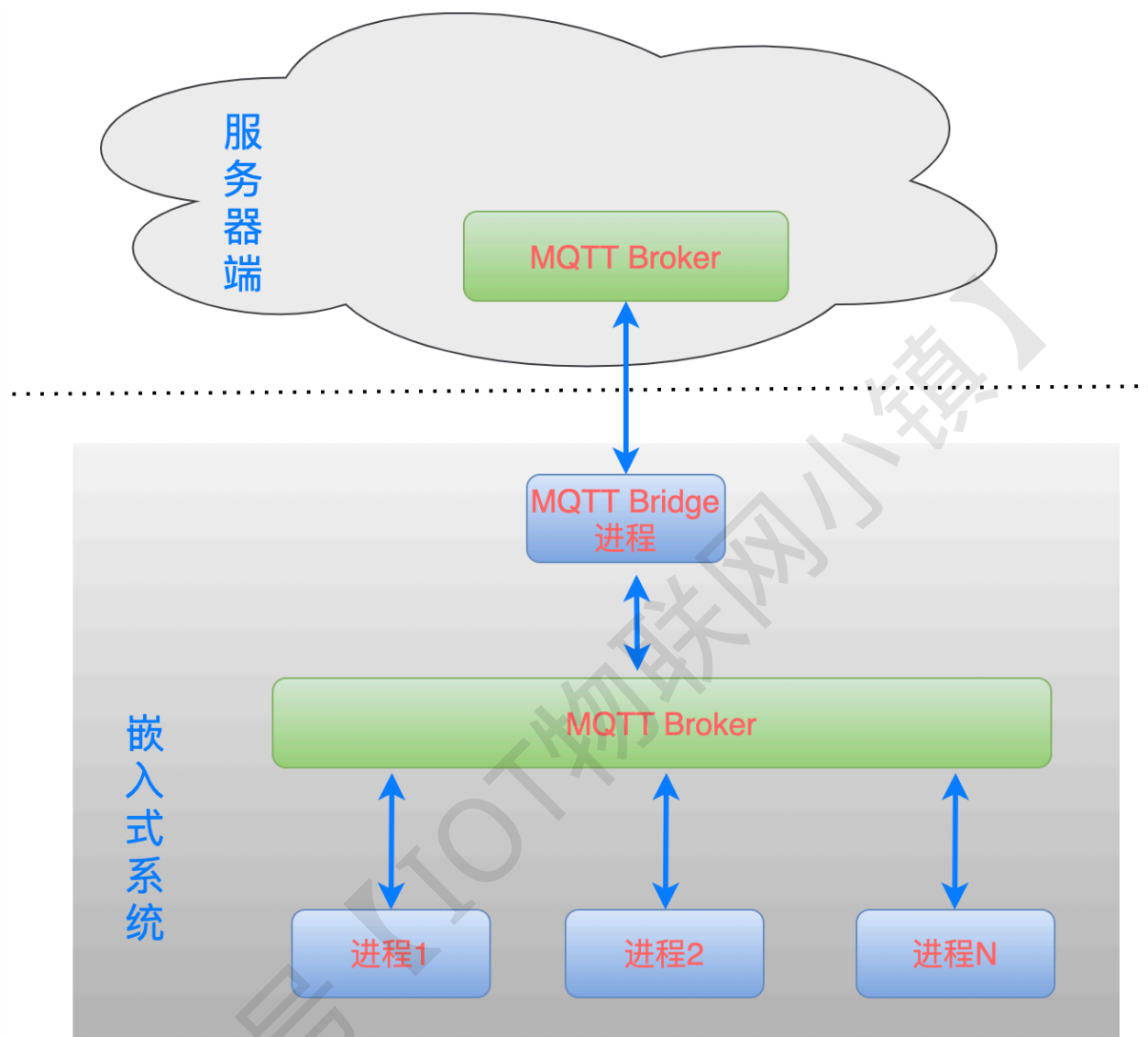


2. 稍微复杂一点的通信模型

公众号【IOT物联网小镇】

在刚才描述的嵌入式系框架设计中，每一个进程都是运行在本地的，所有的消息也都是在系统内进行收发。那么，如果需要把数据传输到云端、或者需要从云端接收一些控制指令，又该如何设计呢？

加入一个 MQTT Bridge 桥接模块即可！也就是再增加一个进程，这个进程同时连接到云端的 MQTT Broker 和本地的 MQTT Broker，通信模型如下：



1. MQTT Bridge 接收到云端发来的指令时，转发到本地的消息总线上；
2. MQTT Bridge 接收到本地的消息时，转发到云端的消息总线上。

五、Mosquitto: 一个简单的测试代码

上面的内容主要讨论的是设计思想，具体到代码层面，我一般使用的是 Mosquitto 这个开源的实现。

在 Linux 系统中安装、测试都非常方便，下面就简单说明一下。

1. 直接通过 apt 来安装、测试

可以参考这个文档(<https://www.vultr.com/docs/how-to-install-mosquitto-mqtt-broker-server-on-ubuntu-16-04>)来安装测试。

(1) 安装

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
sudo apt-get update
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
```

(2) 测试

mosquitto broker 在安装之后会 **自动启动**，可以用 netstat 查看 **1883 端口** 来确认一下。

接收端：连接到 broker 之后，订阅 "test" 这个 topic。

```
mosquitto_sub -t "test"
```

发送端：连接到 broker 之后，往 "test" 这个 topic 发送字符串 "hello"。

```
mosquitto_pub -m "hello" -t "test"
```

当发送端执行 mosquitto_pub 时，在接收端的终端窗口中，就可以接收到 "hello" 这个字符串。

2. 通过源码来手动编译、测试

通过 apt 来安装主要是用来简单的学习和测试，如果要在项目开发中使用 Mosquitto，肯定需要 **手动编译**，得到头文件和库文件，然后复制到应用程序中使用。

(1) 手动编译、安装 Mosquitto

我的开发环境是：

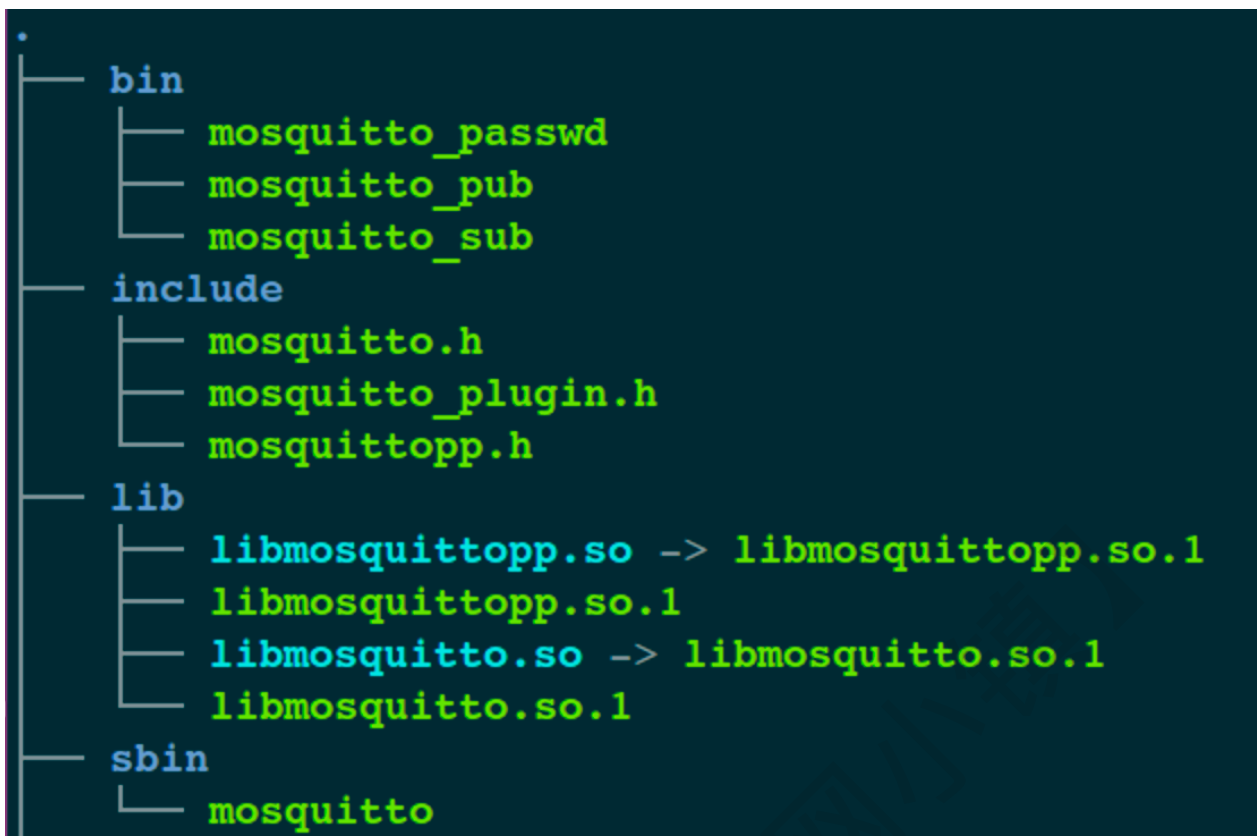
1. 编译器：gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
2. Mosquitto 版本：mosquitto-1.4.9

mosquitto-1.4.9 可以到官方网站下载，也可以从文末的网盘中下载，你也可以尝试更高的版本。

编译、安装指令：

```
make
make install prefix=$PWD/install
```

成功安装之后，可以在 **当前目录的 install 文件夹** 下看到输出文件：



1. bin: mqtt 客户端程序;
2. include: 应用程序需要 include 的头文件;
3. lib: 应用程序需要链接的库文件;
4. sbin: mqtt broker 服务程序。

在编译过程中，如果遇到一些诸如：ares.h、uuid.h 等依赖文件找不到的错误，只需要通过 apt 指令安装响应的开发包即可。

(2) 最简单的 mosquitto 客户端代码

在 mosquitto 源码中，提供了丰富的 Sample 示例。如果你不乐意去探索，可以直接下载文末的这个网盘中的 Demo 示例程序，这个程序连接到消息总线上之后，订阅“topic_01”这个主题。当然，你也可以修改代码去发送消息(调用：mosquitto_publish 这个函数)。

进入 c_mqtt 示例代码目录之后，可以看到已经包含了 bin、include 和 lib 目录，它们就是从上面（1）中安装目录 install 中复制过来的。

执行 make 指令之后，即可编译成功，得到可执行文件：mqtt_client。

测试过程如下：

Step1: 启动 MQTT Broker

在第 1 个终端窗口中，启动 sbin/mosquitto 这个 Broker 程序。如果你在上面测试中已经启动了一个 broker，需要先 kill 掉之前的那个 broker，因为它们默认都使用 1883 这个端口，无法共存。

Step2: 启动接收端程序 mqtt_client

在第 2 个终端窗口中，启动 mqtt_client 也就是我们的示例代码编译得到的可执行程序，它订阅的 topic 是“topic_01”。


```
./mqtt_client 127.0.0.1 1883
```

参数 1: Broker 服务的 IP 地址，因为都是在本地系统中，所以是 127.0.0.1；

参数 2: 端口号，一般默认是1883。

Step3: 启动发送端程序 bin/mosquitto_pub

在第 3 个终端窗口中，启动 bin/mosquitto_pub，命令如下：

```
./mosquitto_pub -h 127.0.0.1 -p 1883 -m "hello123" -t "topic_01"
```

参数 -h: Broker 服务的 IP 地址，因为都是在本地系统中，所以是 127.0.0.1；

参数 -p: 端口号 1883；

参数 -m: 发送的消息内容；

参数 -t: 发送的主题 topic。

此时，可以在第 2 个终端窗口（mqtt_client）中打印出接收到的消息。

六、总结

这篇文章主要介绍了嵌入式系统中的一个设计模式：通过消息总线来实现进程之间的通信，并介绍了 Mosquitto 这个开源实现。

在实际的项目中，还需要更加严格的权限控制，比如：在接入消息总线时提供用户名、密码、设备证书，客户端的名称必须满足指定的格式，订阅的 topic 必须符合一定的格式等等。

在下一篇文章中，我们继续讨论这个话题，给出一个更具体、更实用的 Demo 例程。

七、资源下载

1. mosquitto-1.4.9.tgz

链接:<https://pan.baidu.com/s/1izQ3dAlGbHiHwDvKnOSfyg>

密码:dozt

2. Mosquitto Demo 示例代码

链接:<https://pan.baidu.com/s/1M-dU3xapNbKyk2w07MtDyw>

密码:aup3

公众号【IOT物联网小镇】

不吹嘘，不炒作，不浮夸，认真写好每一篇文章！

欢迎[转发、分享](#)给身边的技术朋友，道哥在此表示衷心的感谢！

转发的[推荐语](#)已经帮您想好了：

道哥总结的这篇总结文章，写得很用心，对我的技术提升很有帮助。好东西，要分享！

【原创声明】

作者：道哥(公众号: [IOT物联网小镇](#))

知乎：道哥

B站：道哥分享

掘金：道哥分享

CSDN：道哥分享

转载：欢迎转载，但未经作者同意，必须保留此段声明，必须在文章中给出原文连接。

关注+星标公众号，不错过最新文章



微信搜一搜

🔍 IOT物联网小镇

推荐阅读

C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻

一步步分析-如何用C实现面向对象编程

提高代码逼格的利器：宏定义-从入门到放弃

原来gdb的底层调试原理这么简单

公众号【IOT物联网小镇】

利用C语言中的setjmp和longjmp，来实现异常捕获和协程

关于加密、证书的那些事

深入LUA脚本语言，让你彻底明白调试原理

公众号【IOT物联网小镇】