道哥的第 021 篇原创

一、前言

在上一篇中，我们聊了在一个嵌入式系统中，如何利用MQTT消息总线在各进程之间进行通信，文章链接：《我最喜欢的进程之间通信方式-消息总线》。

这样的通信模型，我之前已经在多个项目中应用过，对于非工控产品来说，通信速度完全足够。我以前做过测试，在x86平台和ARM平台，一条数据从本地到云端绕一下，然后再回到本地，可以控制在毫秒级别。

上篇文章只是简单的介绍了这样的一种设计思路，并没有详细的讨论其中的一些细节问题。这一次，我们就来具体的聊一聊物联网系统中的网关内部程序应该如何设计。

阅读这篇文章，你可以有如下收获：

> 1. 物联网系统中，设备之间是如何通信的；
> 2. 网关中的进程之间消息总线通信模型；
> 3. 网关内部消息总线上的数据如何与服务器进行通信；
> 4. 作为消遣，了解一下物联网系统中的一些基本知识；

二、网关的作用

物联网这个词语的范畴太广，似乎所有的硬件设备，只要能够接入网络，就可以称之为物联网产品，似乎物联网这个词可以把一切都纳入到其中。这么空洞的词语不利于我们的讲解，因此我们就用一个可以感知、想象的场景来代替，那就是智能家居系统，这是最能代表物联网时代的典型产品了。

2.1 指令转发

在一个智能家居系统中，假设有这么几个设备：

红外感应

门磁

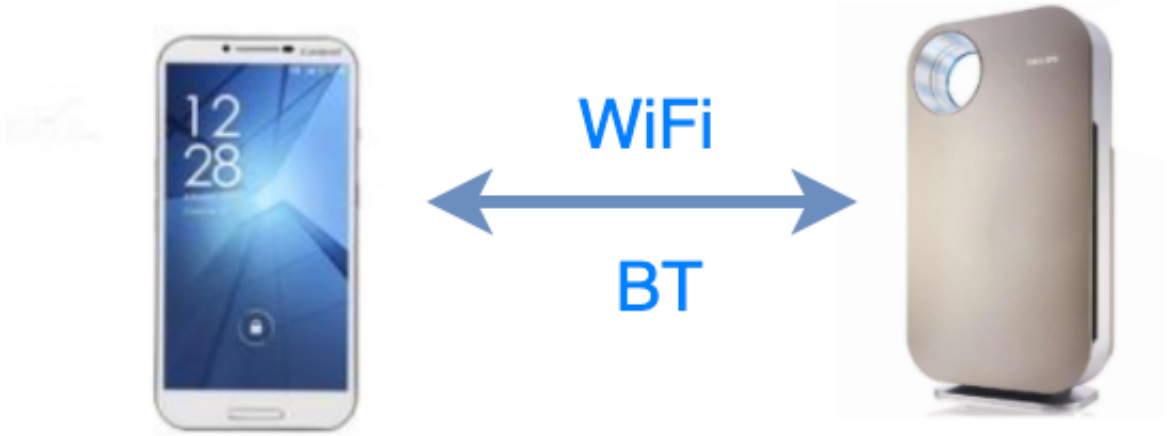
插座

排插

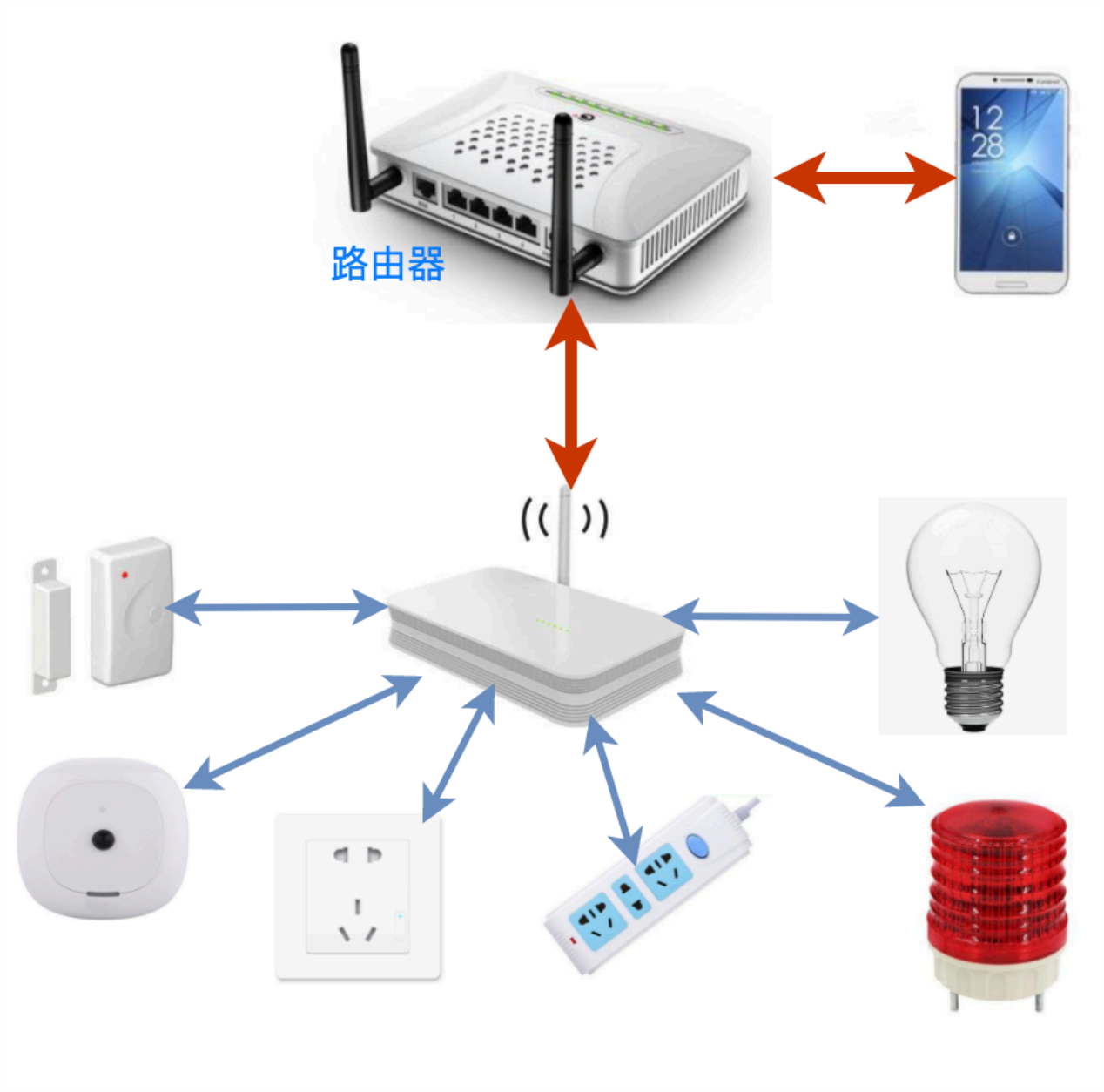
声光报警器

灯泡

这些设备的通信模块，如果是 WiFi 或者是蓝牙，那么一般都可以直接通过手机来控制(当然，需要厂家提供相应的手机 APP)，手机就相当于一个中心节点，控制着所有的设备。目前市面上的一些智能设备单品都是这样的通信方式，例如：空调、吸尘器、空气净化器、冰箱等等。只要在这些设备中加一个无线通信模块即可(例如：ESP8266模块)。

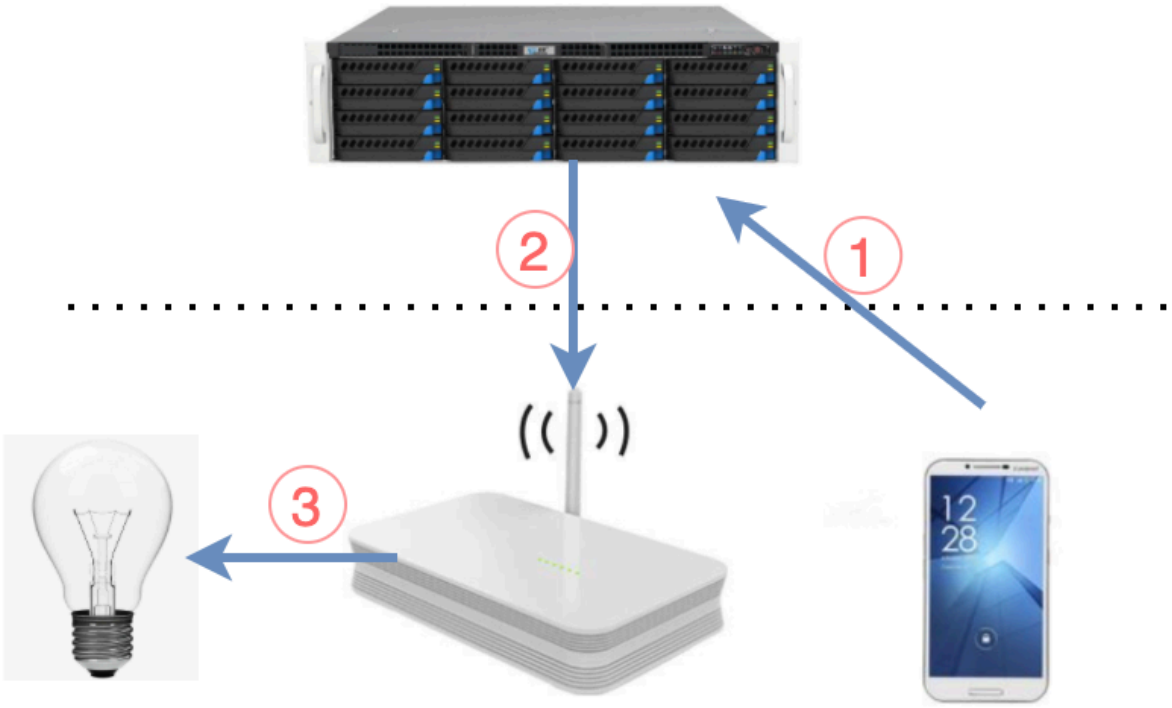


如果通信模块是其它的通信模块，例如：RF433、ZigBee、ZWave等，由于手机没有这些通信模块，因此就需要一个网关来“转发”指令。手机和网关都连接到家中的路由器，处于同一个局域网中，手机把控制指令发送给网关，网关再把指令转发给相应的设备。通信模型如下：



2.2 外网通信

在上面的通信模型中，手机和网关由于处于同一个局域网中，因此可以直接通信。如果手机不在局域网中呢？那么就要通过云端的服务器来转发了，通信模型如下：



1. 手机把指令发到服务器；
2. 服务器把指令转发给网关；
3. 网关把指令发给指定的设备；

以上描述的是控制指令的流程，如果是设备发出的报警信息呢，数据的流向就是倒过来进行的。

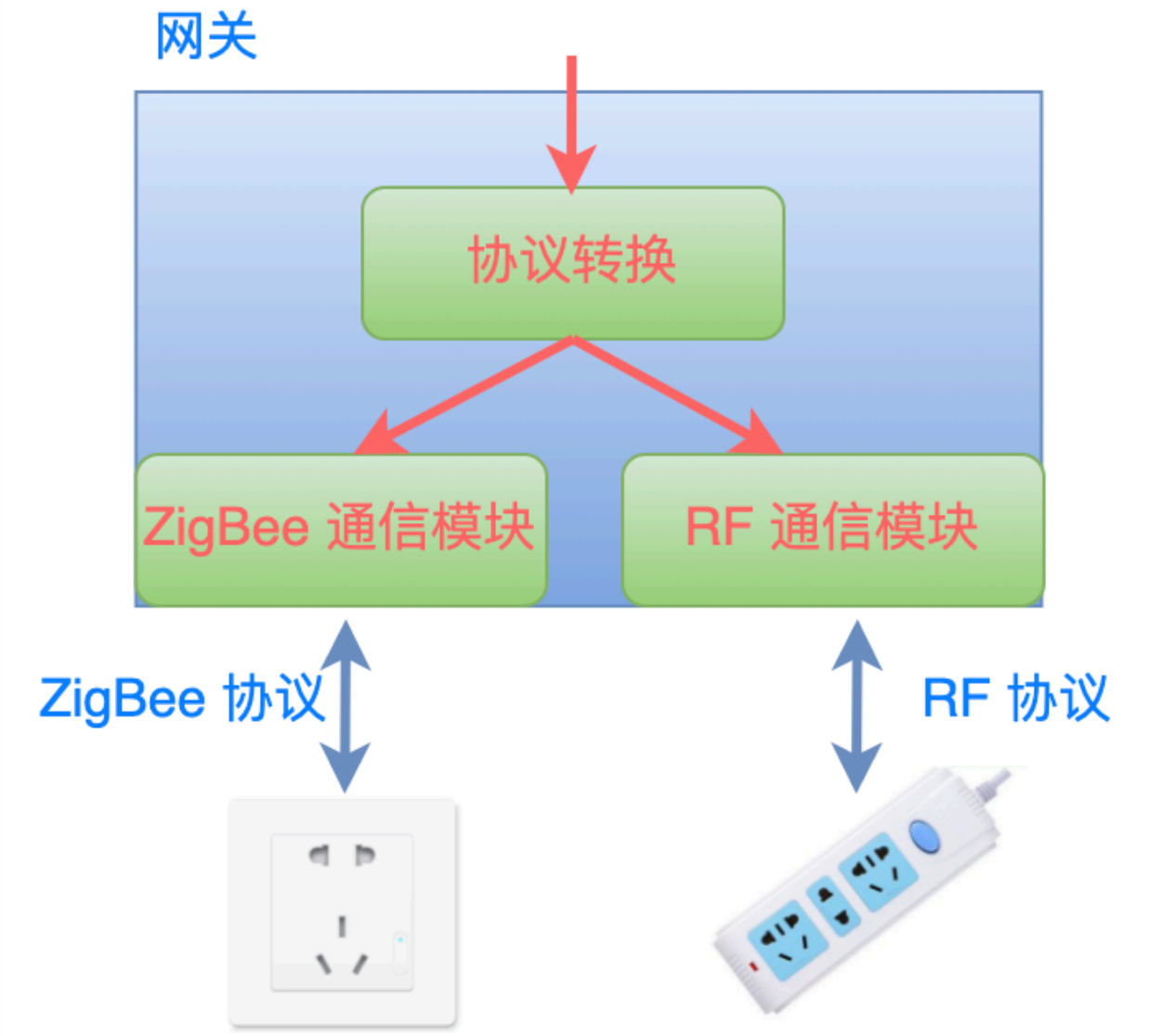
可以看出，网关是所有设备之间通信的中心节点，也是内网与外网之间通信的中转节点，也就是把各种智能设备连接到互联网的中转器。

2.3 协议转换

上面已经提到，硬件设备上的通信模块都是确定的(RF，ZigBee，ZWave等等)，一般来说，可以把这些通信模块称呼为无线通信协议。在一套智能家居系统中，所有设备的无线通信协议大部分都是相同的。

那么，不同类型的无线通信协议设备是否可以共存在同一个系统中呢？

答案是：可以。只要在网关中，集成了相应的无线通信协议模块就可以达到这个目的！如下图所示：



从手机APP上看，所有的设备都是相同的，不会关心设备的无线通信协议是什么，因此，发出的控制指令都是协议无关的。

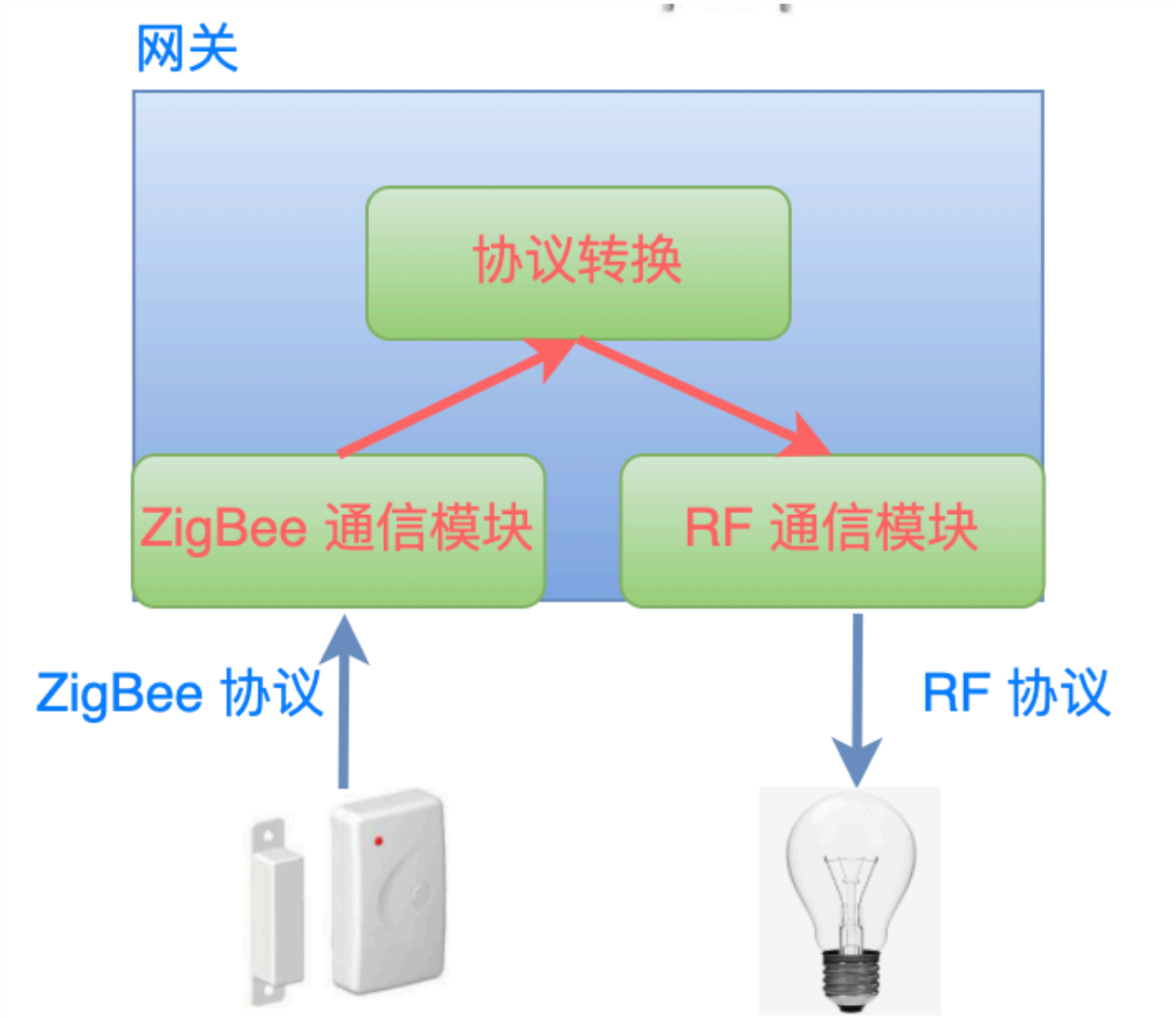
当网关接收到控制指令时，首先根据指令内容查找出目标设备，然后确定目标设备的无线通信协议，最后把指令发送给对应的硬件通信模块，由该通信模块通过无线电信号把控制指令发送到设备。

从这个指令的传输过程来看，网关就充当着协议转换的角色。

另外还有一种通信场景：当系统中的一个“输入”设备与一个“输出”设备进行绑定/关联时，例如：

1. 红外感应器与声光报警器绑定：当红外感应器监测到人体时，发出信号，然后控制声光报警器发出报警；
2. 门磁与灯绑定：当开门时，门磁发出信号，自动打开灯光；

如果“输入”设备与“输出”设备是不同类型的无线通信协议，也需要网关来进行协议转换。



2.4 设备管理

在一个智能家居系统中，设备可多可少，对这些设备进行管理也是很重要的事情。网关作为系统的中心节点，对设备进行管理的重任理所当然就由网关来承担。

设备管理功能包括：

> 设备的添加和删除；
> 设备状态的管理(电量、设备断网、失联等等)；
> 设备树的管理；

2.5 边沿计算(自动化控制)

在正常的情况下，网关是可以通过路由器，与服务器保持着长连接的。如果服务器的处理能力比较强大，智能家居系统中所有需要处理的事情都可以丢给服务器来计算、处理，服务器在计算之后把处理结果再发送给网关。看起来想法很完美！

但是，考虑下面这 2 种情况：

> 1. 路由器出现问题了，网关无法连接到服务器，因此就无法把本地数据及时上报；
> 2. 系统中出现了异常情况，需要紧急处理，如果把信息上报到服务器，由服务器计算之后再回传给网关，耗费的时间可能超过了可容忍时间，该如何处理？(可以用车联网系统来脑补一下这个场

> 景：自动驾驶中的汽车遇到紧急情况，如果把所有信息上传给服务器，然后等待服务器的下一步指令？）

对于上面的这些场景，把一些计算、处理操作放在网关这一端来处理也许更合适！这也是近几年比较流行的边沿计算。

> 1．边缘计算，是指在靠近物或数据源头的一侧，采用网络、计算、存储、应用核心能力为一体的开放平台，就近提供最近端服务。
>
> 2．其应用程序在边缘侧发起，产生更快的网络服务响应，满足行业在实时业务、应用智能、安全与隐私保护等方面的基本需求。
>
> 3．边缘计算处于物理实体和工业连接之间，或处于物理实体的顶端。而云端计算，仍然可以访问边缘计算的历史数据

三、网关内部进程之间的通信

在设计一个应用程序的架构时，可以通过多线程来实现，也可以通过多进程来实现，每个人的习惯都不一样，各有各的好处。我们这里不去讨论孰优孰劣，因为我对多进程这样的设计思想比较偏爱，所以就直接按照多进程的程序架构来讨论。

3.1 网关中需要哪些进程

网关中需要执行的所有进程，是根据网关的功能来决定的，假设包括如下的功能：

（1）连接外网的进程 Proc_Bridge

网关需要连接到云端的服务器，需要一个进程与服务器之间保持长连接，这样就可以及时接收到服务器发来的控制指令，以及把系统内部数据及时上报给服务器。

这个进程需要把从服务器接收到的指令转发到网关系统内部，把从系统内部接收到的信息转发给服务器，类似于桥接的功能，因此命名为 Proc_Bridge。

（2）设备管理进程 Proc_DevMgr

这个进程用来执行设备管理功能，设备的添加(入网)、删除(退网)，都由此进程来管理。

（3）协议转换进程 Proc_Protocol

下行：把应用层的统一通信协议，转换成不同类型无线通信协议，发送给相应的无线模块。

上行：把设备上报的、不同类型的无线通信协议，转换成应用层的统一通信协议。

（4）边沿计算进程(自动化控制) Proc_Auto

很明显，这需要一个独立的进程来处理各种计算，这个进程就相当于系统的大脑。

（5）无线通信协议相关的进程 Proc_ZigBee, Proc_RF, Proc_ZWave

在硬件上，每一种无线通信模块通过串口或其他硬件连接方式与到网关的 CPU 进行通信，因此，每一种无线通信模块都需要一个相应的进程来处理。

（6）其他“软设备”进程 Proc_Xxx

在之前的项目中，还遇到一些硬件设备，它们与门磁、插座等设备在逻辑上处于同一个层次，但是与网关之间是通过 TCP 来连接。对于这样的设备，也可以使用一个独立的进程来进行管理。

上面的这些进程，在网关中的运行模型如下：

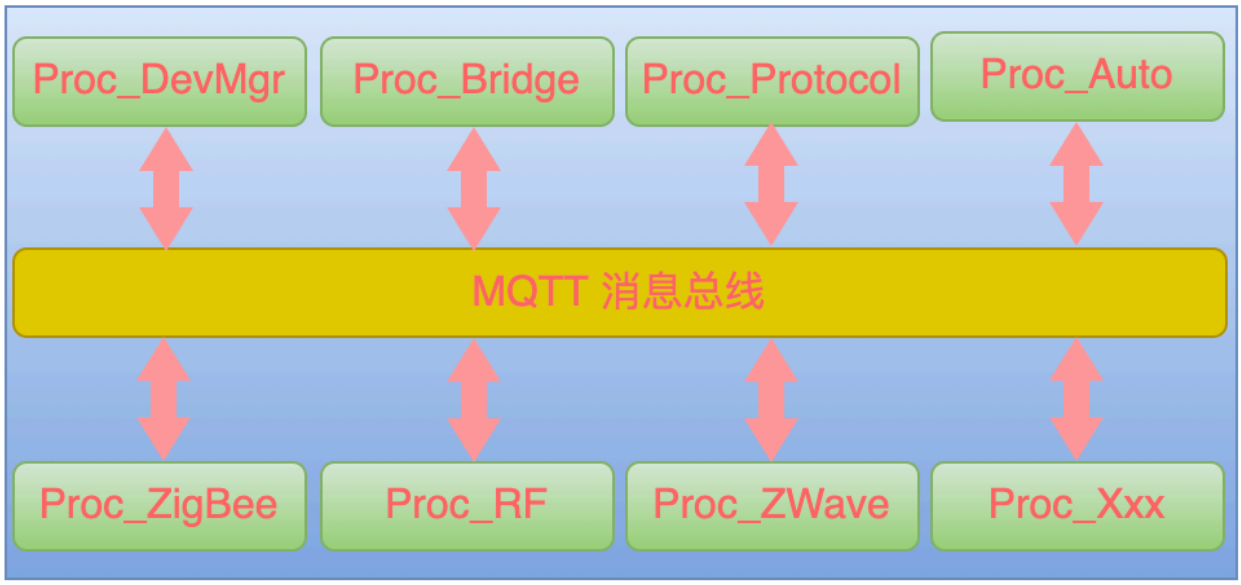
3.2 MQTT消息总线

以上这些进程之间需要相互通信，不是简单的点对点通信，而是一个网状的通信模型。比如：

1. 设备管理进程 Proc_DevMgr：当任何一种设备被添加到系统中时，都需要处进行处理，因此它需要与 Proc_ZigBee, Proc_RF, Proc_ZWave 这些进程进行通信;
2. 当某个设备上报数据时(例如：Proc_ZigBee)，Proc_Protocol 进程需要把数据进行协议转换，然后 Proc_Bridge 进程把转换后的数据上报给服务器，同时 Proc_Auto 进程需要检查这个设备上报的数据是否触发了其他相关联的设备;

也就是说，这些进程中间的通信是相互交叉的，如果通过传统的 IPC 方式(共享内存、命名管道、消息队列、Socket)等，处理起来比较复杂。

引入了 MQTT 消息总线之后，每个进程只需要挂载到总线上。每个进程只需要监听自己感兴趣的 topic，就可以接收到相应的数据。



既然这些进程之间的通信关系比较复杂，那么一个良好的 topic 设计规范就显得很重要了！

3.3 Topic 的设计

MQTT 的通信模型是基于订阅/发布的模式，一个客户端(进程)接入到消息总线之后，需要注册自己感兴趣的 主题 topic，其他客户端(进程)往这个 topic 发送消息，即可被订阅者接收到。

主题 topic 是一个以反斜线(/)分割的字符串，用来表示多层的分级结构，例如下面的这 2 个 topic，是亚马逊 AWS 平台中在线升级(OTA)相关的 topic:

> 1. $aws/things/MyThing/jobs/get/accepted
> 2. $aws/things/MyThing/jobs/get/rejected

在我们的示例场景中，可以按照下面这样来设计主题 topic:

(1) Proc_DevMgr

订阅主题:

> $iot/v1/ZigBee/Register
> $iot/v1/ZigBee/UnRegister
> $iot/v1/RF/Register
> $iot/v1/RF/UnRegister
> $iot/v1/ZWave/Register
> $iot/v1/ZWave/UnRegister

(2) Proc_Bridge

订阅主题:

> $iot/v1/Device/Report

发出数据的主题:

> $iot/v1/Device/Control
> $iot/v1/Device/Remove
> $iot/v1/Auto/AddRule
> $iot/v1/Auto/RemoveRule

(3) Proc_Protocol

订阅主题:

> $iot/v1/Device/Control
> $iot/v1/Device/Remove
> $iot/v1/ZigBee/Report
> $iot/v1/RF/Report
> $iot/v1/ZWave/Report

发送数据主题:

> $iot/v1/Device/Report
> $iot/v1/ZigBee/Control
> $iot/v1/ZigBee/Remove
> $iot/v1/RF/Control
> $iot/v1/RF/Remove
> $iot/v1/ZWave/Control
> $iot/v1/ZWave/Remove

(4) Proc_Auto

订阅主题:

> $iot/v1/Auto/AddRule
> $iot/v1/Auto/RemoveRule
> $iot/v1/Device/Report

发送数据主题:

> $iot/v1/Device/Control

(5) Proc_ZigBee

订阅主题:

> $iot/v1/ZigBee/Control
> $iot/v1/ZigBee/Remove

发送数据主题:

> $iot/v1/ZigBee/Register
> $iot/v1/ZigBee/UnRegister
> $iot/v1/ZigBee/Report

(6) Proc_RF

订阅主题:

> $iot/v1/RF/Control
> $iot/v1/RF/Remove

发送数据主题：

> $iot/v1/RF/Register
> $iot/v1/RF/UnRegister
> $iot/v1/RF/Report

(7) Proc_ZWave

订阅主题：

> $iot/v1/ZWave/Control
> $iot/v1/ZWave/Remove

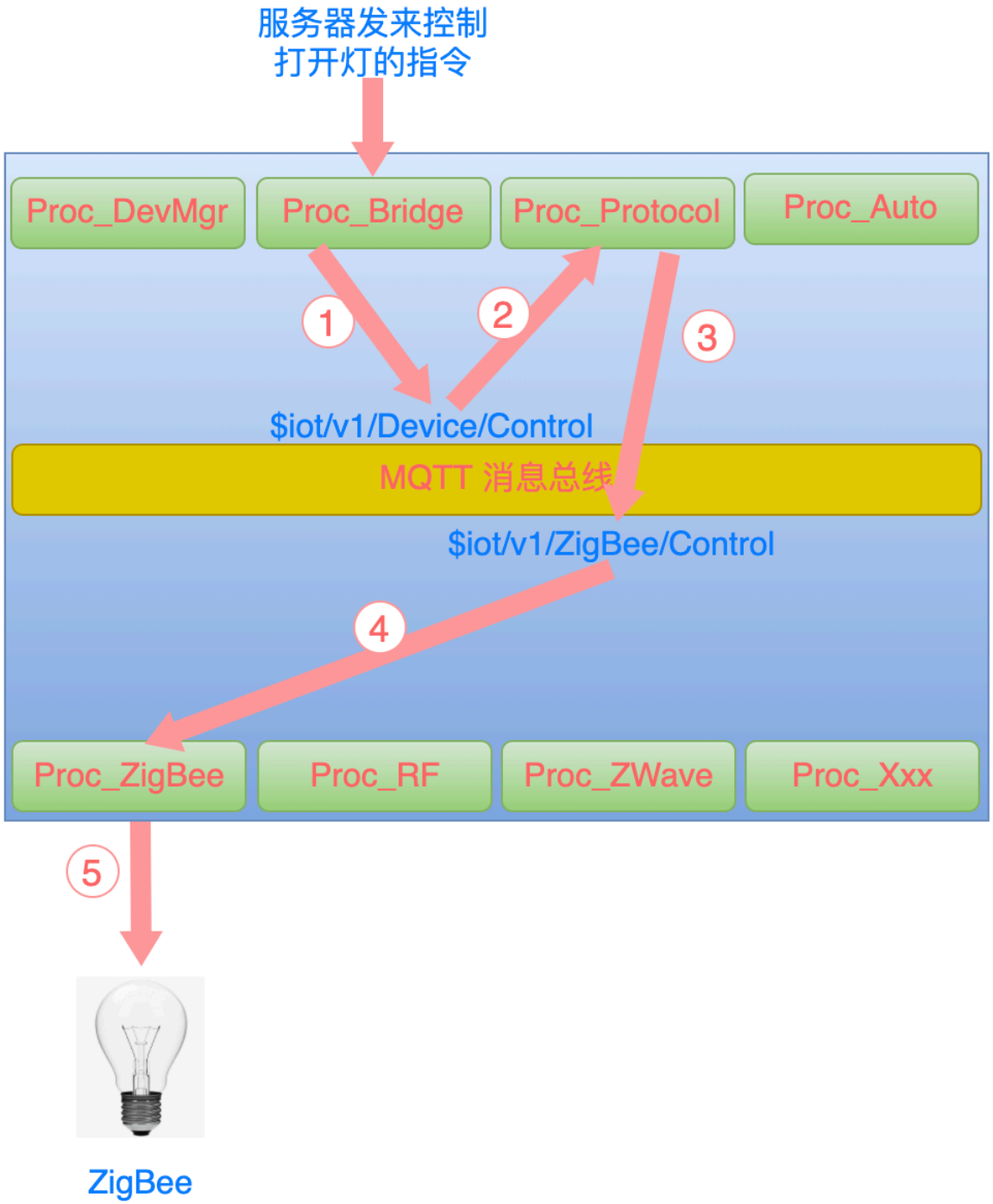
发送数据主题：

> $iot/v1/ZWave/Register
> $iot/v1/ZWave/UnRegister
> $iot/v1/ZWave/Report

以上这些主题 topic 的设计，还是有些粗略的。如果借助通配符(#, +, $)，可以设计出更灵活的层次结构。

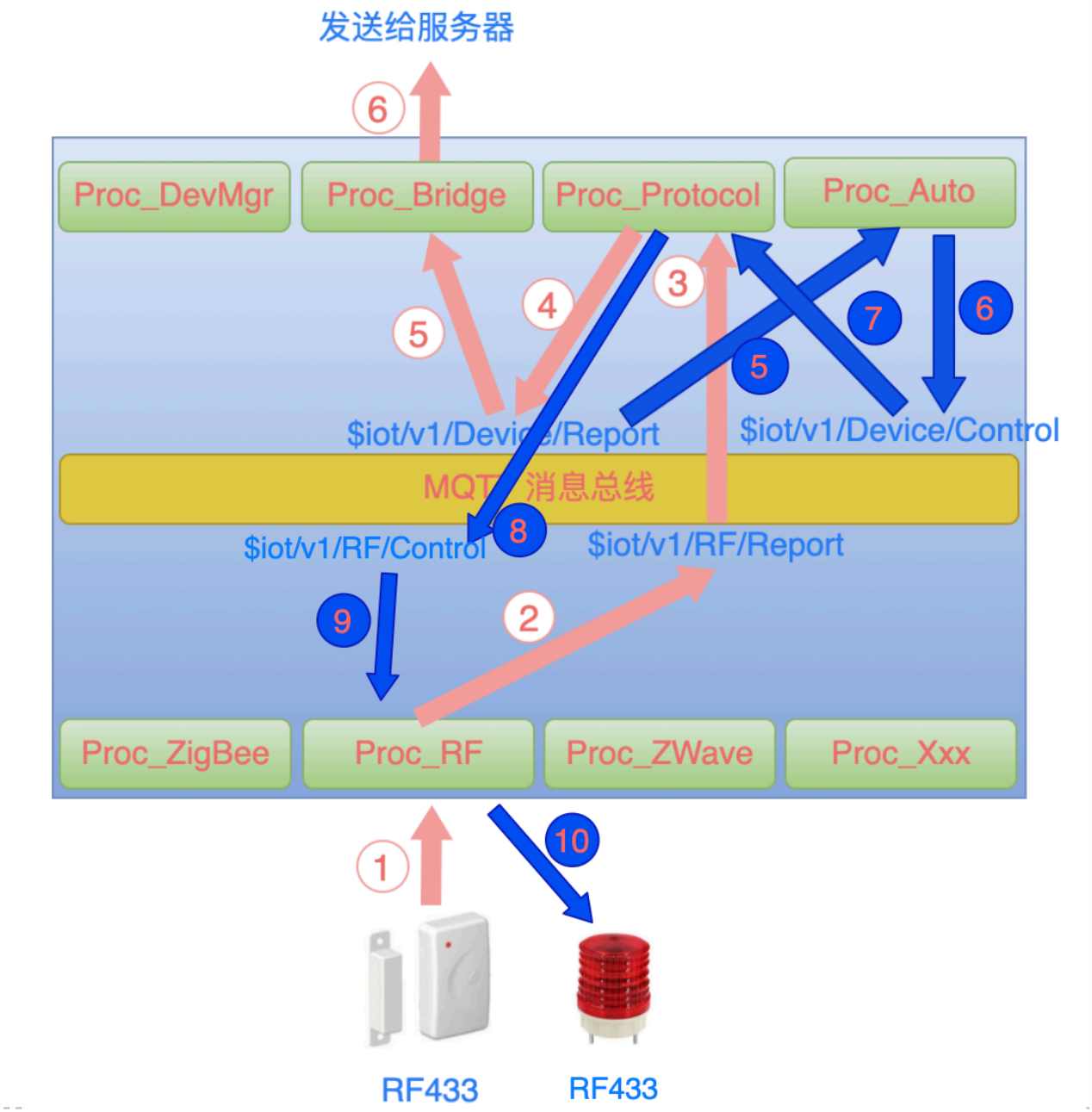
> 1. 多层通配符: “#”是用于匹配主题中任意层级的通配符，多层通配符表示它的父级和任意数量的子层级。
> 2. 单层通配符：“+”加号是只能用于单个主题层级匹配的通配符，在主题过滤器的任意层级都可以使用单层通配符，包括第一个和最后一个层级。
> 3. 通配符：“$”表示匹配一个字符，只要不是放在主题的最开头，其它情况下都表示匹配一个字符。

我们以一个控制指令为例，来梳理一下数据是如何通过 topic 进行流动：



1. Proc_Bridge 进程从服务器接收到控制指令后，发送到消息总线上的 topic: $iot/v1/Device/Control。
2. 由于 Proc_Protocol 进程订阅了这个 topic，所以立刻接收到指令。
3. Proc_Protocol 分析指令内容，发现是一个 ZigBee 设备，于是进行协议转换，发送一个 ZigBee 控制指令到消息总线上的 topic: $iot/v1/ZigBee/Control。
4. 由于 Proc_ZigBee 进程订阅了这个 topic，因此它接收到这个控制指令。
5. Proc_ZigBee 把控制指令转换成 ZigBee 无线通信模块要求的格式，通过硬件发送给设备灯泡。

我们再分析一下设备数据上报的场景：



先关注图中红色箭头，忽略蓝色箭头：

1. 门磁打开后，通过无线通信把信息上报给进程 Proc_CF。
2. Proc_RF 进程接收到 RF433 通信模块上报的数据，把“门磁打开”这个信息发送到消息总线上的 topic：$iot/v1/RF/Report。
3. 由于 Proc_Protocol 进程订阅了这个 topic，因此接收到上报的门磁数据。
4. Proc_Protocol 分析数据，把 RF433 协议的数据转成统一的应用层协议的数据，发送到消息总线上的 topic：$iot/v1/Device/Report。
5. 由于 Proc_Bridge 进程订阅了这个 topic，因此就接收到了这次上报的数据。
6. Proc_Bridge 进程把数据上报给服务器。

再来看一下蓝色箭头流程：

在上面的第 4 步：Proc_Protocol 进程把 RF433 协议数据转成应用层统一协议之后，把数据发送到消息总线上的 topic：$iot/v1/Device/Report 之后，Proc_Auto 进程同时进行如下操作：

5. 由于 Proc_Auto 也订阅了这个 topic，因此它也接收到了门磁上报的这个应用层协议的数据。

6. Proc_Auto 查找自己的配置信息(假设用户已经提前配置好了一条规则：当门磁打开的时候，就触发声光报警器)，发现匹配到了“门磁->报警器”这条规则，于是发出一条控制报警器的指令，发送到消息总线上的 topic: $iot/v1/Device/Control。

后面的 7，8，9，10 这四个步骤就与上面的控制指令流程完全一样了。

3.4 与 DBUS 总线的对比

从上面描述的 3 个数据流向的场景中，是不是感觉到使用 topic 为“数据管道”的这种通信方式，与 Linux 系统中的 DBUS 总线特别的相似?

DBUS 总线也是用于进程之间的通信，按照我个人的理解，DBUS中其实是把进程之间的两种通信组织在一起了：

1. 基于信号的数据传输；
2. 基于方法的 RPC 远程调用；

DBUS 总线包含的概念更复杂一些，包括：路径、对象、接口、方法等等，这些概念组织在一起共同定位到一个具体的服务提供者了。

相比较而言，我感觉 MQTT 这样的方式更简洁一些。

所谓的 RPC 远程调用，就是调用位于远程机器上的一个函数，主要解决两个问题：

1. 网络连接；
2. 数据的序列化和反序列化；

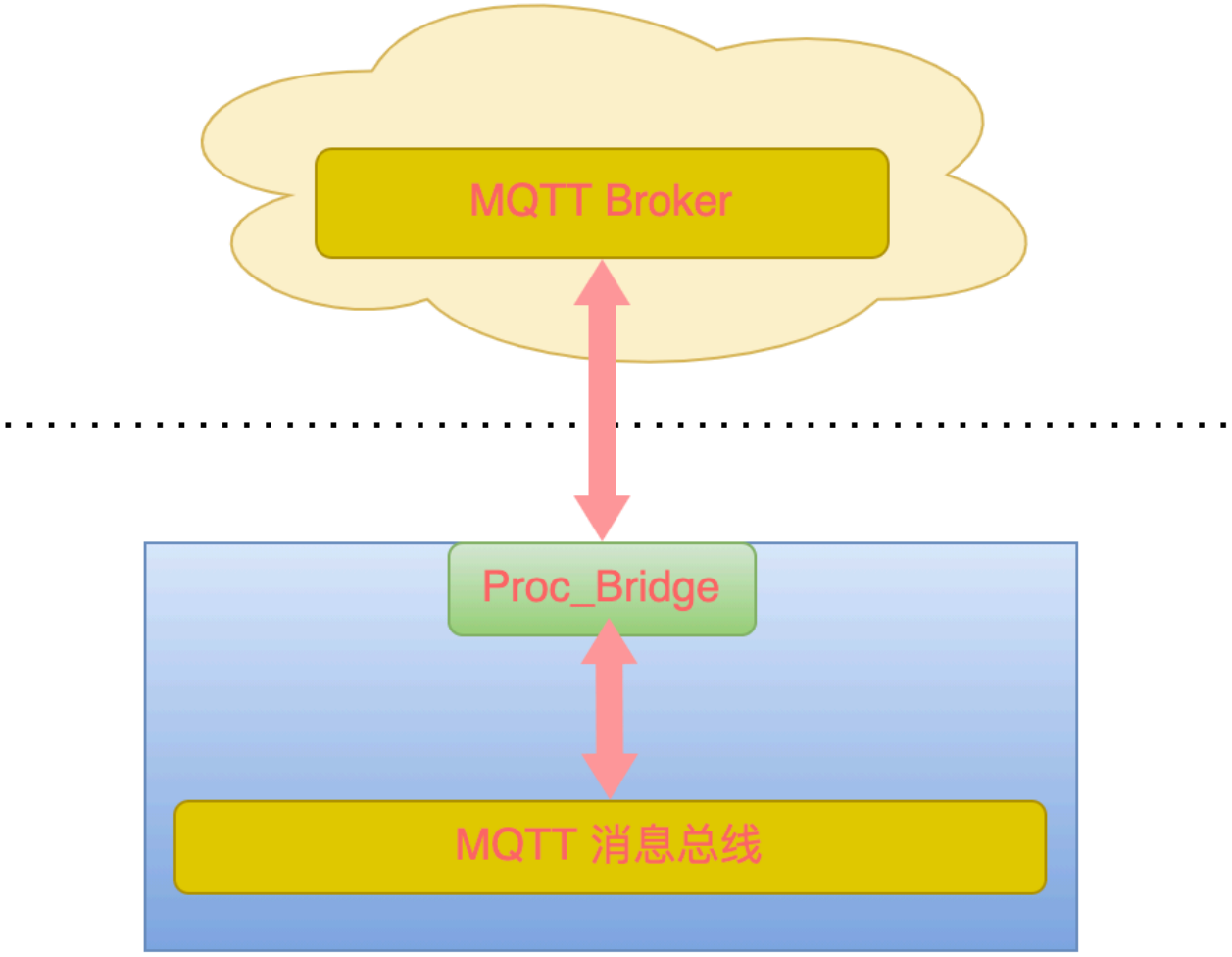
后面我会专门写一篇文章，利用 protobuf 框架来实现 RPC 调用。

四、网关与云平台之间的通信

上面讲解的设计过程，是网关内部的各功能模块之间通信方式，这也是我们作为嵌入式开发者能充分发挥的部分。

网关与云平台之间的通信方式一般都是客户指定的，就那么几种(阿里云、华为云、腾讯云、亚马逊AWS平台)。一般都要求网关与云平台之间处于长连接的状态，这样云端的各种指令就可以随时发送到网关。

当然了，这些云平台都会提供相应的 SDK 开发包，一般使用 MQTT 协议来连接云平台的更多一些。在一些文档中，会把位于云端的 MQTT 服务器称作 Broker，其实就是一个服务器。



进程 Proc_Bridge 的功能主要有 2 点：

1. 与云平台的数据传输通道；
2. 协议转换：把云平台相关的协议转换成网关内部的协议，以及相反的转换。

也就是说：Proc_Bridge 进程需要同时连接到云平台的 MQTT Broker 和网关内部的 MQTT 消息总线。在下一篇文章中，我们来专门讲解这部分的内容，并提供一个实现桥接功能的代码模板。

五、总结

作为一名嵌入式软件开发人员，仅仅根据别人设计好的框架来填充代码，时间久了就会有些倦怠，不知道技术提升的方向在哪里。仔细想想，其实方向挺多的：Linux 内核、文件系统、算法、应用程序设计等等。

这篇文章讨论的内容还谈不上架构设计，仅仅是一个简单的物联网网关内部各功能模块的通信模型。如果你有机会设计类似的产品，不妨尝试一下这样的通信模型，当然你一定会设计的更好！

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！

推荐阅读

我最喜欢的进程之间通信方式-消息总线
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
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