道哥的第 028 篇原创

一、前言

在上一篇文章中，我们主要聊了：在嵌入式系统的应用程序架构设计中，应该从哪些方面来进行需求整理和分析，文章链接：都说软件架构要分层、分模块，具体应该怎么做(一)。

这篇文章，我们继续聊一下在概要设计、详细设计阶段，我们应该做什么工作？用什么工具或手段来做？输出结果是什么？

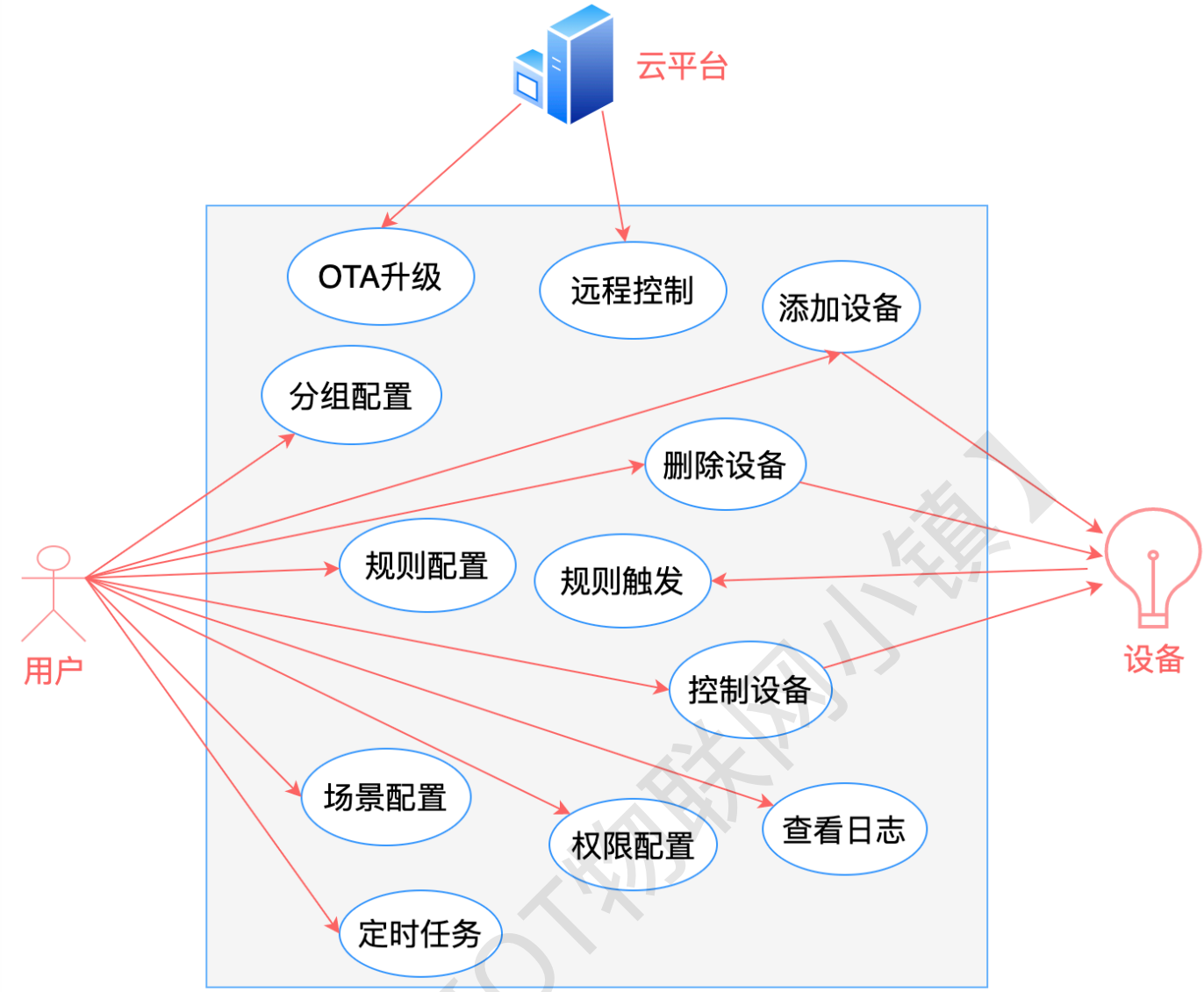
按照惯例，为了内容描述的方便，我会用一个物联网网关的设计过程，把所有的内容串接在一起。如果小伙伴对于网关不太了解，请滑到文章底部的推荐阅读列表，其中有几篇文章是关于网关功能介绍的。

二、需求调研和需求分析

1. 用例图

上篇文章说到，在进行需求调研和需求分析的时候，用例图是非常非常好用的一个工具。通过用例图，我们可以把一个系统中需要完成的所有功能，从粗粒度上一目了然的呈现出来。

下面这张图，是网关的用例图(这里画的用例还不完全)：



2. 用例描述

用例图仅仅是描述了系统具有的功能，但是并没有描述每一个用例的行为，也就是执行过程。

在上一篇文章中说到，我们不需要对每一个用例进行分析，而是需要在这些用例中，找出那些关键用例，然后对这些关键用例写出用例描述，因为关键用例才是系统架构的决定因素。

那么又出现一个问题了：如果把所有的用例，按照重要程度进行优先级排序，那么从上到下应该选取多少个、或者说百分之多少的关键用例呢？这个就要看整个系统的复杂度了，30%不嫌少，50%不嫌多，根据你的时间自由把握。

以上图网关中的用例图来说，我认为：添加设备、删除设备、控制设备、规则配置、规则触发这几个用例比较关键，因此，我就针对这几个用例写用例描述。

（1）添加设备用例描述

1. 用例名称
   添加设备。

2. 简要说明
   把一个终端设备(插座、灯泡、报警器等等)，添加到系统中。

3. 事件流
   (3.1) 基本事件流
   a. 用户通过手机APP，选择设备类型，发起添加设备动作；
   b. 网关进入无线监听状态；
   c. 用户触发被添加的设备，此设备向网关发出添加请求；
   d. 网关接收设备添加请求，查询设备相关信息；
   e. 网关记录日志信息，并且把添加成功的设备信息通知用户；

   (3.2) 扩展事件流
   如果添加失败，提醒用户重新添加。

4. 非功需求
   网关监听设备请求信号有超时时间限制。

5. 前置条件
   必须是系统管理员才可以添加设备。

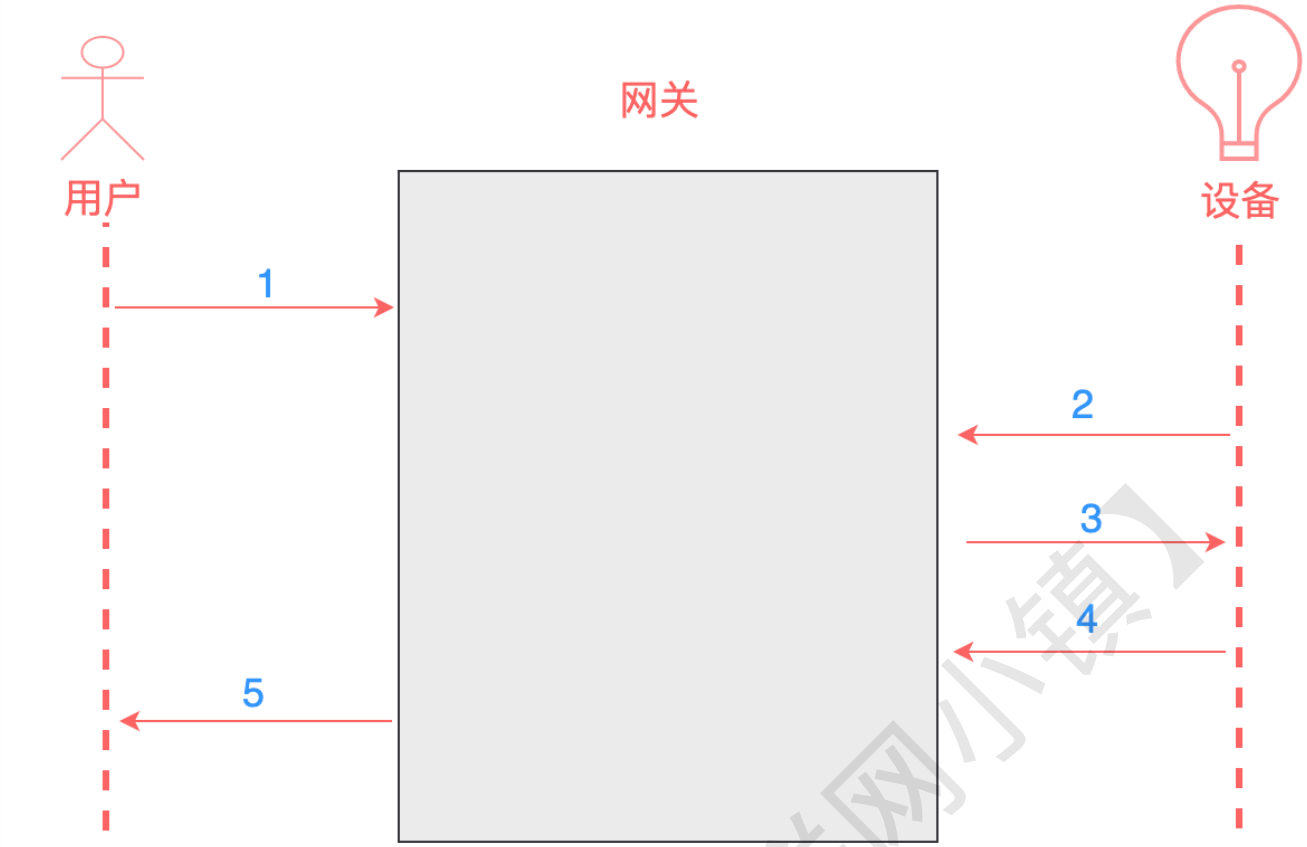
6. 后置条件
   无。

7. 优先级
   高。

其中有 2 点注意的地方：

> 1. 在事件流中，我们是把网关作为一个黑盒进行描述的，因为我们是在进行需求分析，而不是在进行设计，因此，不需要考虑网关内部的执行流程；
> 2. 红色部分都是一个执行主体，这个主体可以是一个人、一个界面、一个设备、一个系统等等；

事件流可以用文字来描述(就像图中这样)，也可以画一个序列图来展现这个过程，就像下面这样(这里没有详细描述出更细的执行过程，主要以示意性为主):



(2) 删除设备用例描述

1. 用例名称
   添加设备。

2. 简要说明
   把一个终端设备(插座、灯泡、报警器等等)，添加到系统中。

3. 事件流
   (3.1) 基本事件流
   a. 用户通过手机APP，选择设备类型，发起添加设备动作；
   b. 网关进入无线监听状态；
   c. 用户触发被添加的设备，此设备向网关发出添加请求；
   d. 网关接收设备添加请求，查询设备相关信息；
   e. 网关记录日志信息，并且把添加成功的设备信息通知用户；

   (3.2) 扩展事件流
   如果添加失败，提醒用户重新添加。

4. 非功需求
   网关监听设备请求信号有超时时间限制。

5. 前置条件
   必须是系统管理员才可以添加设备。

6. 后置条件
   无。

7. 优先级
   高。

(3) 控制设备用例描述

(4) 规则配置用例描述

1. 用例名称
   添加规则。

2. 简要说明
   向系统中添加一条新的规则。

3. 事件流
   (3.1) 基本事件流
   a. 用户通过手机APP，发起添加规则动作(选择源设备--触发条件--目标设备)；
   b. 网关接收到指令，添加到规则数据库中，发送添加规则成功消息给用户；

   (3.2) 扩展事件流
   如果源设备或目标设备不存在，或者触发条件有问题，需要提醒用户添加规则失败。

4. 非功需求
   无。

5. 前置条件
   必须是系统管理员才允许添加规则。

6. 后置条件
   无。

7. 优先级
   高。

(5) 规则触发用例描述

三、概要设计

可以把概要设计理解成一个粗略、抽象的架构图，用来体现高层组件，以及它们之间的联系。那么应该怎么做，才能得到这样的一张架构图呢?

我们现在的掌握的材料就是：用例图和(关键用例的)用例描述，而且在用例描述的基本事件流中，把要设计的系统当做一个黑盒子进行描述。

现在我们需要做的事情，就是打开这个黑盒子，进入其中内部，从执行过程上来分析：需要哪些模块完成什么动作。

注意，这是我们的目的。要达成这个目的，使用鲁棒图这个工具。

也就是说，我们现在需要通过鲁棒图这个工具，去拆解用例描述中的事件流，把系统内部的、为了完成这个用例所需要的参与元素，全部都找出来，并标注它们之间的关系。

1. 针对关键用例的用例描述，画出鲁棒图

先说一下容易混淆的概念：鲁棒性，也称作健壮性，是指程序在运行过程中，即使出现了一些错误的状况，也已让能够顺利的执行下去。它描述的是程序的容错性。

鲁棒图是指：用图形建模的方式，来描述一个用例描述是否正确、是否完善。

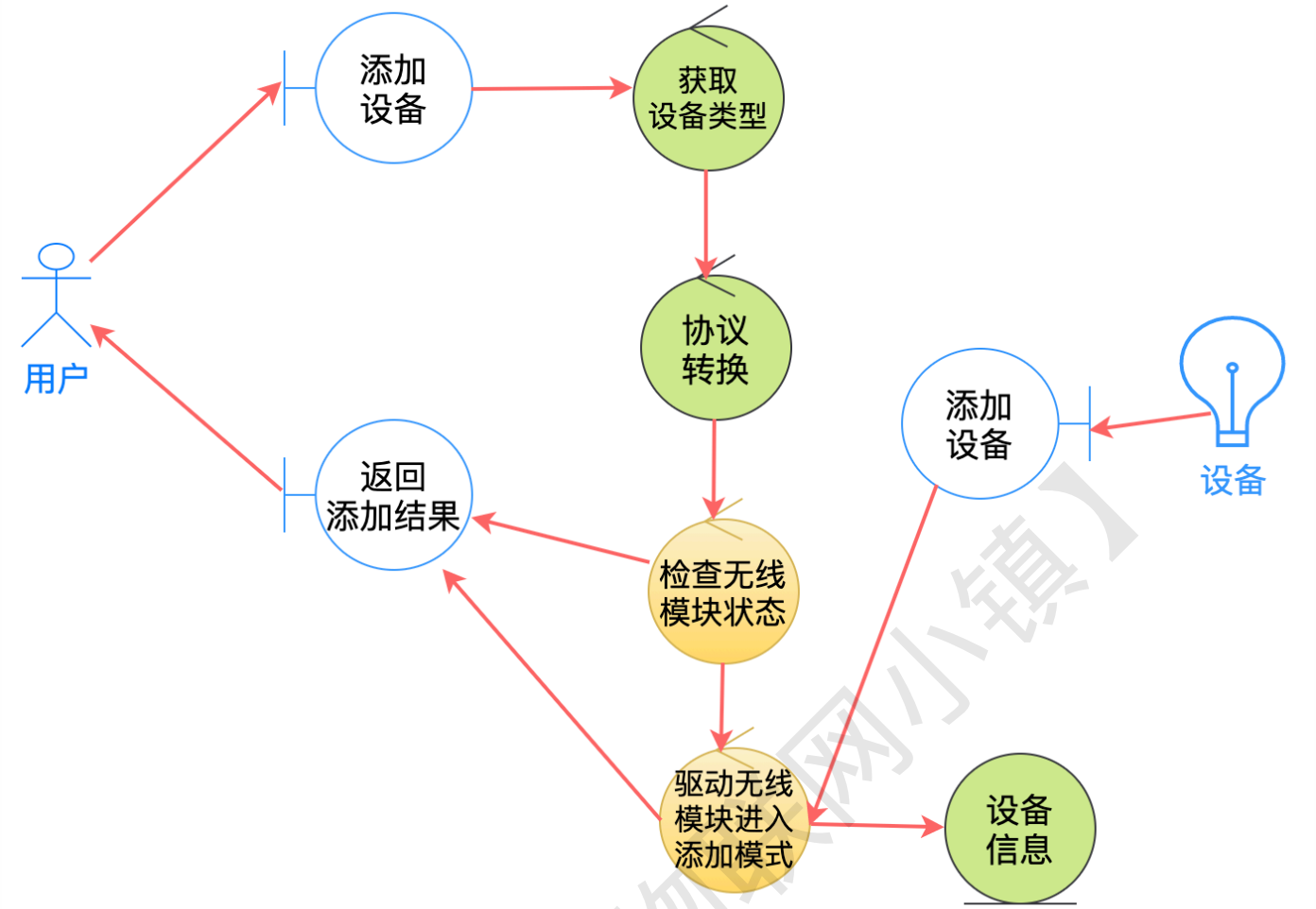
主要通过 3 种元素：边界对象，控制对象和实体对象，来画出一个用例描述中，待设计的系统内部各功能模块之间的交互关系。

> 1. 边界对象：在系统内部，需要与外界进行交互的元素。它负责接收外部的输入、向外部输出内部的处理结果；
> 2. 控制对象：描述动态的控制行为，强调从一个执行环节进入另一个执行环节；
> 3. 实体对象：对一个信息内容进行描述，比如：网关中的一个设备描述信息、一条规则配置信息等；

关于边界对象，在 Web 类项目中，可能比较好理解，就是与用户、外部系统所交互的界面。但是在嵌入式系统中，大部分情况下是没有界面的，但是我们只要抓住一个根本的东西：接收外部的输入、向外部输出数据。

我们这里就简单画一下添加设备、控制设备和规则触发，这 3 个用例描述对应的鲁棒图（先忽略这几张图中的颜色）：

添加设备：

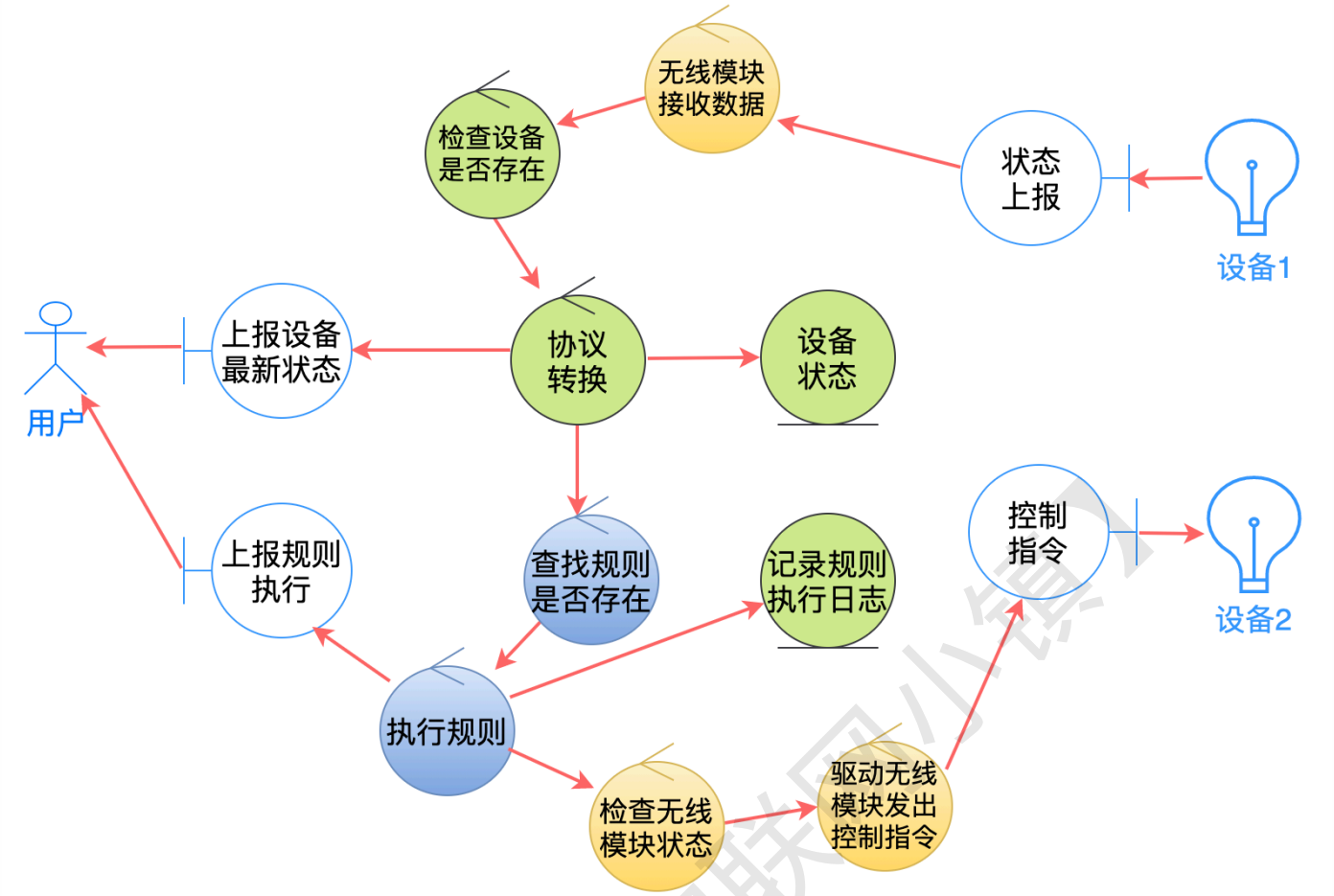


控制设备:

规则触发:

关于添加规则的执行过程中，大部分工作是在手机 APP 上完成的(选择源设备--触发条件--目标设备)，网关中只是把配置好的这条规则存储一下而已，没有其他过多的操作。

规则中更重要的部分是规则触发的处理，例如：当红外设备(源设备)检测到人体时，如果当前处于布防状态(触发条件)，就启动声光个报警器(目标设备)，因此下面这张图是描述执行一条规则的执行过程，这个过程的执行链条比较长，能把很多的模块串接起来。

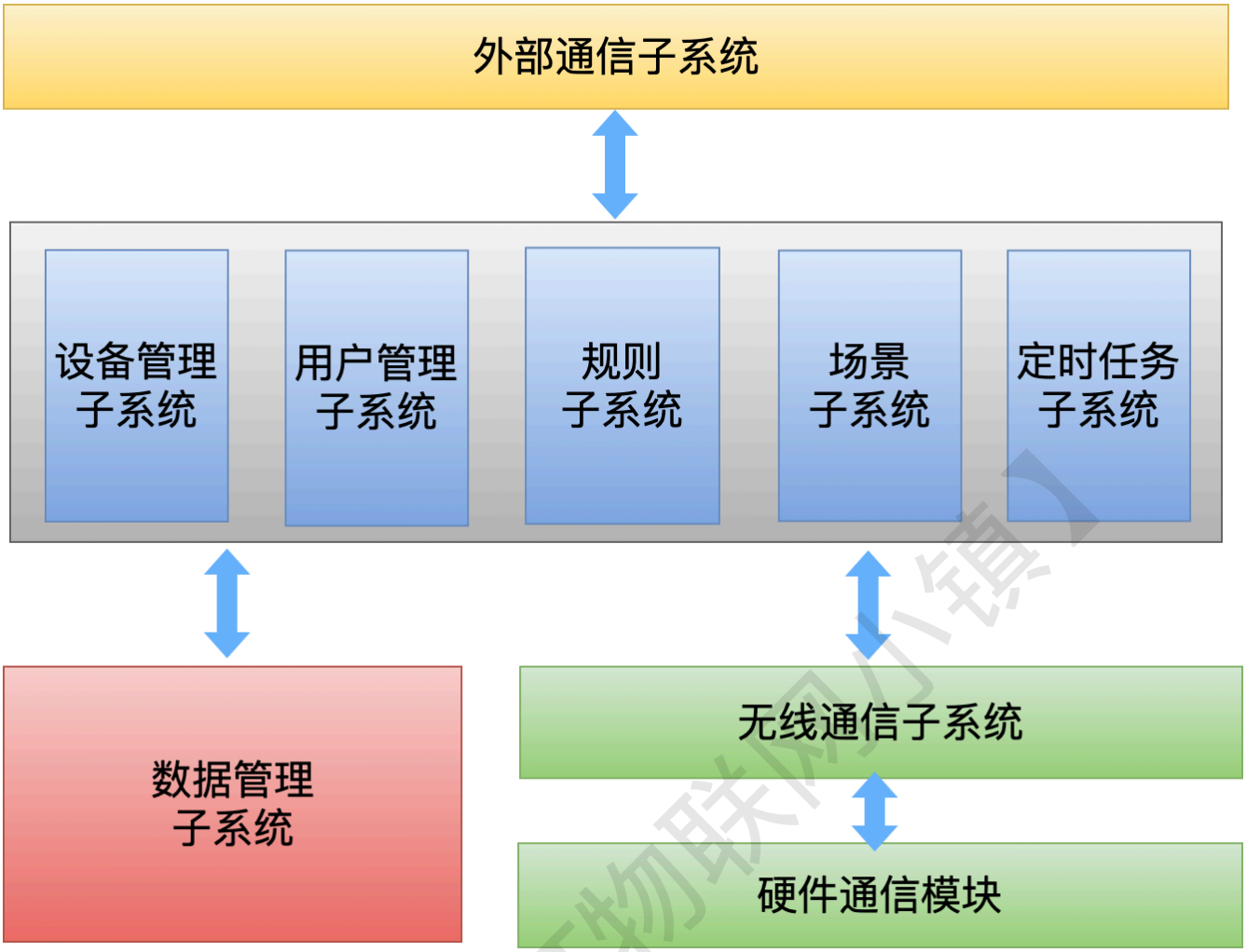


2. 对鲁棒图中的模块进行归类，归纳出子系统

假设我们现在把所有关键用例的鲁棒图都画出来了，下一步的动作就是对这些模块进行分类。上面几张图中，有些模块被标记了不同的颜色，相同的颜色表示它们是属于一类的。

1. 黄色部分的模块都是与无线通讯相关的，那么这些模块就可以归类为无线通信管理子系统;
2. 绿色部分的模块都是与设备相关的，那么它们就归类为设备管理子系统;
3. 蓝色部分的模块都是与规则相关的，那么它们就归类为规则管理子系统;
4. 继续找出其他的子系统。。。

最终，我们把这些子系统（或者称之为功能组）画到一张图中如下:



这张图就从上层组件的视角，把整个系统划分为几个子系统，每一个子系统都是一个独立的、可以交付的实体模块。

这张图的作用还是挺大的，可以用于向领导进行汇报(领导才没有时间看详细的设计)，也可以用于产品说明书中的技术架构描述部分，还可以用于团队成员分工，因为每一部分都是一个独立的单位，与其他子系统之间的耦合性，从静态和动态两方面都隔离开来了（待会在后面的开发架构设计中进行说明）。

这些子系统之间是需要通信的，因此，在画出这个设计图之后，我们还需要做出下面的几个决策：

1. 使用的技术栈：开发语言 C，进程之间的通信方式：消息总线；
2. 并发：每个子系统以进程为执行单位运行在系统中，通过 MQTT 消息总线的C语言实现 mosquitto 库，来接入到总线系统上；
3. 系统不支持二次开发;

四、详细设计

在上面的概要设计图中，已经把所有的功能模块划分到不同的子系统中，也可以称之为功能组。下一步的工作，就是把每一个功能组中的内部对象、需要完成的功能、交互流程找出来，具体来说，就是要分析出系统的逻辑架构、运行架构和开发架构。

1. 逻辑架构

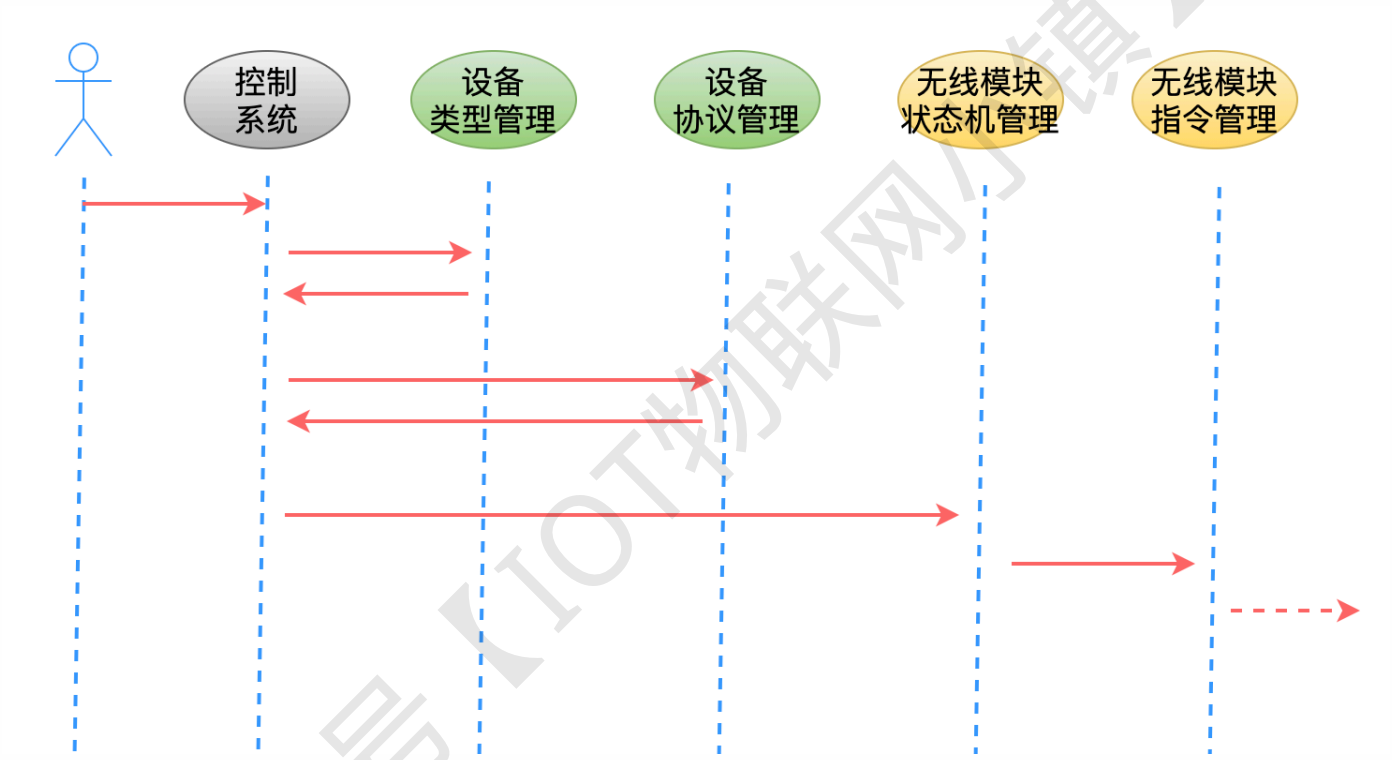
逻辑架构就是把每一个子系统再分为粒度更细的功能块，如果想粒度更细的话，也可以拆解到类这个级别。此外，还需要定义好各模块之间的交互接口。

根据上面的描述，我们已经决定把各子系统设计为一个独立的进程，各进程之间通过消息总线进行数据交互，而这个消息总线，是基于 topic 主题来进行消息路由的，因此，下面就要设计好每一个进程需要处理哪些数据交互：

1. 入口：对其他哪些模块的请求进行响应；
2. 出口：为了完成自己的工作，需要依赖其他哪些模块提供服务；

一句话总结：就是找出每一个模块，为了完成自己的工作，需要与其他哪些单元模块之间进行交互？交互的接口(函数、方法或者协议)是什么？

那么怎么来找到这些对象和接口呢？用序列图或者类图来完成。下面是控制设备的一个简单序列图：



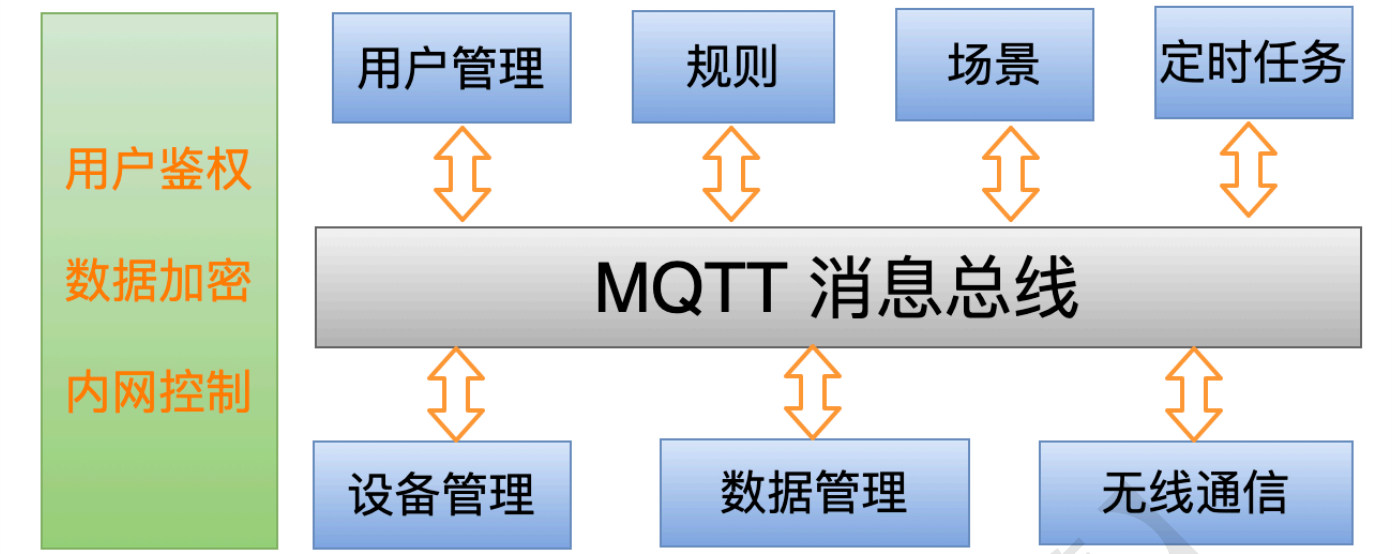
图中的每一个箭头，都代表一个接口，对于这个网关来说，就代表处理的一个 topic 主题。

如果用类图来分析，对于面向对象的开发语言来说，可能会更容易理解，比如：可以明确的定义出每一个对象的属性，私有函数，共有函数，并且能够清晰的构建出对象之间的关系。

2. 运行架构

运行架构描述的是每一个执行单元的动态状态、执行时的控制流程，需要考虑的重点是：系统是否安全？性能是否满足质量要求？可扩展性如何？

具体到网关来说，每一个子系统是以进程为执行单位的，每个进程通过一个第三方的附件(也就是动态库)，挂接到消息总线上，如下图所示：



系统的并发性，是通过多进程来实现；系统的安全性，主要通过消息总线的安全机制来管理。

比如在开发阶段，消息总线允许系统外的其他客户端接入，这样就可以在 PC 机上写一个调试程序，接入到总线中，可以监听所有的数据，此时数据可以不加密，全部是 human readable 的；但是在项目 release 阶段，那么就关闭这个权限，PC 机上的客户端就不能接入总线，并且总线中所有数据的需要加密、压缩，进一步提高系统的安全性。

3. 开发架构

作为以撸代码为主力的我们来说，开发架构就容易理解了，无非就是定义好项目结构、编译流程、测试步骤等等。

具体来说，我们可以从下面几方面来做出规定：

1. 并行开发：每个子系统是一个独立的进程，因此可以划分为一个独立的项目，提高开发效率；
2. 第三方库：作为基础的公共模块来使用(SSL加密、消息总线接入、通信协议解析)；
3. 代码安全：每位开发人员只能有权限拿到自己负责的代码，只有管理员有权限获取所有代码；
4. 代码管控：使用 git、svn 等工具进行代码版本的管理；
5. 集成编译：使用 Jenkins + git module 功能，自动拉取所有的子系统代码，自动编译。如果需要自动部署的话，也可以使用脚本来实现。

五、架构验证

终于来到最后一个环节了，其实项目经历多了，以上设计出来的架构，是否能满足需求中提出的功能和质量要求，我们在心中已经大概知道答案了。

为了保险起见，我们还是需要对其中的某些关键部分进行验证。这个验证过程是有价值的，或者说可以把这个验证过程所得到的成果，作为正式的代码进行提交。

验证的大方向有 2 点：系统的框架是否合理、稳定；一些技术瓶颈是否可以搞定。如果这两部分都没问题，那后面就可以大胆的往前走了。

六、总结

经过 2 篇文章的介绍，我基本上把自己在平常工作中，对应用程序架构设计的这个思考过程描述了一遍。

佛经里说了：渡人就像帮助一个人过河，过了河上了岸，就应该把乘坐的木筏丢掉，心中不要再想着木筏。

这篇文章介绍的设计流程，也是一个套路而已。这个套路在面对一个新领域、新项目时，就像一个脚手架一样，告诉我们这一步该做什么，下一步该做什么，应该使用什么样的工具。

在僵化的运用这个套路之后，你可以继续改造、优化，然后丢掉这个套路，从而形成适合你自己的套路，从此走向思考致富的道路！

祝你好运！

(如果您觉得这是一篇干货，对其他的小伙伴有价值，请您转发、分享！非常感谢！也非常欢迎在留言区一起讨论、吹牛！)

推荐阅读

公众号【IOT物联网小镇】

【C 语言】

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
原来gdb的底层调试原理这么简单
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
利用C语言中的setjmp和longjmp，来实现异常捕获和协程

【应用程序设计】

物联网网关开发：基于MQTT消息总线的设计过程(上)
物联网网关开发：基于MQTT消息总线的设计过程(下)
我最喜欢的进程之间通信方式-消息总线

【物联网】

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

【胡说八道】

以我失败的职业经历：给初入职场的技术人员几个小建议