澳柯玛智能U+家电物联平台

技术方案

北京思佳维科技有限公司

开发部

2015年06月12日

**目 录**

[项目背景 2](#_Toc422140234)

[项目需求 3](#_Toc422140235)

[解决方案 4](#_Toc422140236)

[一、 系统概述 4](#_Toc422140237)

[1．系统构成 4](#_Toc422140238)

[2．通信模型 8](#_Toc422140239)

[二、云平台技术方案 14](#_Toc422140240)

[1．总体架构 14](#_Toc422140241)

[2．TCP通信服务 15](#_Toc422140242)

[3．消息推送服务 17](#_Toc422140243)

[4．数据库服务 20](#_Toc422140244)

[5．接口服务 22](#_Toc422140245)

[6．管理平台 25](#_Toc422140246)

[7．部署方案 32](#_Toc422140247)

[三、客户端技术方案 34](#_Toc422140248)

[1. 总体架构 34](#_Toc422140249)

[2. SDK概述 34](#_Toc422140250)

[四、设备端技术方案 38](#_Toc422140251)

[1、广域网通信 38](#_Toc422140252)

[2、局域网通信 39](#_Toc422140253)

[3、串口通信 40](#_Toc422140254)

# 项目背景

家电物联平台，是一个实现高性能，高可用，可拓展，多产品的物联网服务平台，集合设备通信，数据存储，消息推送，接口服务，后台管理等多种服务。该平台为智能手机实现对各种家电的数据显示，远程控制，用户服务，固件升级等功能实现接口服务，通过管理后台为管理者提供数据采集、显示、分析、设备控制，管理，用户管理，服务等功能。

# 项目需求

家电物联平台的需求如下：

一、为设备端提供接入服务：

1、设备认证：设备注册，登录，认证，上线，离线管理；

2、设备连接：与设备建立稳定的长连接，并做管理；

3、消息收发：设备数据上报，存储，控制指令下发设备；

4、状态存储：设备状态实时存储；

5、消息推送：设备状态推送到客户端；

二、为终端用户提供接口服务，实现如下功能：

1、用户注册，用户登录，用户设置；

2、设备绑定/解绑，设备设定，数据显示，设备控制，OTA升级；

3、故障报警，通知推送，留言反馈；

4、升级，分享，帮助，关于；

三、管理后台服务

1、基础功能：后台登录，用户管理，角色设置，权限分配，日志管理

2、产品管理：产品设定；

3、设备管理：设备状态信息的管理；

4、用户管理：用户状态信息的管理；

5、数据管理：用户操作行为分析，设备数据统计分析；

5、运营管理：消息推送，设备控制，OTA升级，用户反馈；

6、其他相关：第三方服务平台接口服务；

# 解决方案

## 系统概述

### 1．系统构成



**系统原理图**

整个系统由**“云平台，客户端，设备端”三**个物理部分以及 **通信协议** 共同组成。

#### 1.1 云平台

云平台是系统的核心，由五个子部分组成：

* **TCP通信服务**

主要负责：与设备建立连接，安全认证，保持心跳，传输数据与指令，模块升级等功能。由于TCP Server 负责与大量设备保持连接，通信，因此要求服务具备高并发，通信高可靠，是整个系统的基础部分，核心部分。

* **数据服务**

主要负责：关系数据存储，设备状态数据存储，系统运行日志存储，分析数据存储。DB系统是整个系统运转的基础，由于设备数据量大，读写频次高，因此要求数据系统高稳定，高可靠，高并发，是整个系统的基础部分，核心部分。

* **消息推送服务**

主要负责：向用户手持终端设备提供消息推送服务，由于设备量大，状态变化频繁，推送频繁，因此推送系统要求高稳定，高可靠，高并发，是整个系统的基础部分，核心部分。

* **接口服务**

主要负责：为用户手持终端提供Http服务，用户的所有操作均通过API service 实现。接口部分要求稳定，可靠，安全。是整个系统的基础部分。

* **管理平台**

主要负责：为管理人员，运营人员，售后人员，以及工厂的品检人员提供设备管理，数据分析，用户服务，设备检测等功能于一体的管理平台，为整个系统的检测，运营，管理，维护提供服务。要求易用，安全，可拓展。

#### 1.2 客户端

客户端是为用户提供的能够对设备进行远程控制的App软件。

通过手机客户端，用户能方便的查看到设备的运行信息，并对设备进行各种设定，控制，从而给用户提供了便利。同时通过客户端，厂家能够与客户建立零距离的交互，更好的替用户提供服务，将一次性的购买关系变成了长期的稳定的服务关系。

#### 1.3 设备端

设备可以通过多种方式实现联网功能，本方案采用Wifi方式，设备端通过嵌入**Wifi模块**，使设备具备联网功能。

设备通过Wifi模块与云平台保持连接，用户终端通过云平台获取设备状态，给设备下发指令。设备状态的变化通过云平台消息推送服务传递给用户终端。

Wifi模块通过串口与设备的Mcu保持通信，设备端的状态变化通过Mcu上报给Wifi模块，云平台下发的指令通过Wifi模块下发至Mcu并在设备端执行。

#### 1.4 通信协议

通信协议在整个系统中是对用户透明的，然而却是整个系统运行的规范和基础，也可以说系统必须实现通信协议。

通信协议分为四个部分，协议的设计目标如下：

* **Wifi模块与Mcu端协议**
  + - * + 指令下发：WiFi下发指令给MCU；
        + 状态上报：MCU状态变化后，及时上报WiFi模块；
        + 传输稳定：保证串口通信的稳定性；
        + 兼容性强：能够兼容字符协议以及二进制协议；
* **Wifi模块与云端协议**
  + - * + 远程监控：云端可实时监控设备的状态；
        + 远程控制：云端可随时控制在网的设备；
        + 响应及时：设备状态的上报和控制结果必须响应及时；
        + 链路可靠：设备与云端之间的连接链路必须可靠；
        + 节约流量：减少不必要的流量消耗；
* **Wifi模块与App端协议**
  + - * + 辅助大循环：作为大循环的辅助，主要为了提高大循环的用户体验，一般情况下不作为独立使用；
        + 快速高效：收状态发指令更加快捷，提高用户体验；
        + 降低云端负载：如果手机检测到设备在内网，则发指令优先走小循环而不经过云端转发，降低云端负载；
        + 双路降低丢包率：如果手机与设备同在内网，且内网可连通外网，则通过双路控制设备和监听状态，降低丢包率；
* **云端与App端协议**
  + - * + 一致性：

RestFul风格：将系统提供的服务抽象为资源，使用Get，Post，Put，Delete等Http方法，保持接口简洁，美观，易于理解。

Json数据格式：Response返回数据采用通用的Json数据格式。

统一错误码：统一的错误码提示，便于App端做异常处理。

* + - * + 稳定性：

服务需要兼容多个不同的版本，同时为每个版本提供稳定的服务。系统设计需要根据业务的实际需要考虑做负载均衡。

* + - * + 安全性：

接口安全是一个非常重要的考虑因素，尤其对于物联网设备。对于接口访问做身份验证，数据一致性校验，异常监控等，以保障接口服务的安全性。

### 2．通信模型

整个系统物理上的三个组成部分：设备，客户端，云平台，在不同的场景，不同的阶段，通信模型有所不同。

##### 2.1 总体模型



##### 2.2 具体场景

###### 2.2.1 配置设备连接本地WiFi



###### 2.2.2 局域网本地发现&更新设备信息



###### 2.2.3 局域网本地控制设备



###### 2.2.4 外网远程控制



###### 2.2.5 外网更新设备状态信息



###### 2.2.6 外网推送设备报警



##### 2.3 协议设计

###### 2.3.1 设备端与云平台——TCP长连接

根据物联网系统的高实时、快速响应的需求，用户期望在操作手机APP后的最短时间内得到设备控制的真实反馈。为了满足这种用户体验，为了云平台的Server能够快速反向控制物联网设备Client，系统采用采用原生TCP协议，维护长连接以备用户随时的操控Request，此外采用原生TCP协议，便于定制应用层协议。

传统的HTTP协议的请求响应模式则无法承担此任务。如果要采用HTTP，则需要让APP频繁向服务器轮询（polling），浪费云平台网络带宽和用户手机流量，下面是一个对TCP和HTTP的简单对比：

|  |  |  |
| --- | --- | --- |
|  | TCP | HTTP |
| 协议头部长度 | 20字节 | 最少数百字节 |
| 传输效率 | 高 | 低 |
| APP流量耗费 | 低 | 较高（需要轮询） |
| 穿防火墙能力 | 不稳定 | 强 |
| 是否可以长连接 | 是 | 否（HTTP1.1有Keep-Alive选项） |
| 负载均衡 | 网络层/链路层负载均衡，方案较少 | HTTP应用层负载均衡，方案众多 |
| 运行环境 | 操作系统层，底层通信 | Web服务器，容器 |
| 开发难度 | 高 | 低 |

为了追求最佳的用户体验，我们权衡之后采用了TCP长连接的方案。

###### 2.3.2 客户端与云平台——Http协议

客户端与服务端进行通信，采用http协议请求——回复模型来保持手机与云平台的数据交互。（安全性要求高可采用https协议），其主要特点如下：

* 成熟通用：Http协议是互联网最为通用的协议，稳定性，可靠性好，规范成熟，各种应用层库多，稳定。80,8080时http协议默认端口，路由器对这两个端口比较友好。
* 简单快速：由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。
* 无状态：Http为短连接，通信完成之后连接立刻关闭，从而占用服务器资源非常小，高并发变得非常容易。因为http服务的无状态，也特别容易进行分布式部署。

客户端与服务器端的数据传输格式采用标准json格式，数据体积小，传输快，通用性强。

###### 2.3.3 消息推送服务——Mqtt 协议

设备状态变化，通过Socket连接推送到服务器，服务器如何反向推送状态变化到用户终端呢？本系统采用Mqtt协议，通过Mqtt Service 向手机客户端推送设备的状态变化。

###### 2.3.4 设备端与客户端——小循环

**基本概念：**

**大循环（外网）**：用户使用手机等端设备通过互联网和云平台远程操作智能家电，具体功能包括远程控制、状态提醒、故障告警、监控运行状态等。

**小循环（内网）**：用户使用手机等端设备通过家庭局域网本地操作智能家电，具体功能通常仅包括本地控制和家电状态显示（模拟家电本机遥控器功能），相当于多了一个遥控器。在小循环中，不需要使用互联网和云平台。

大小循环各有各的应用场景，互相不可替代。系统采用“小循环优先”的策略，使整体性能更加优化。

1、小循环控制是通过局域网TCP协议传输指令，相比于大循环响应实时性极高，延时小，用户体验最佳。

2、60%的控制指令在内网得以解决，对云平台的负载压力降低。

客户端会根据当前的网络状况（Wifi or 3G），网络质量（信号强弱），Wifi模块工作负荷决定是走小循环还是大循环，这一切都是自动实现的，对用户来说是透明的。

## 二、云平台技术方案

### 1．总体架构



云平台可采用公有云，私有云两种不同方式，服务器资源层面部署会有所不同，上层服务是一致的。本方案基于公有云。

云平台依据服务可以划分为5个子部分，随着业务的发展会衍生出更多的子部分。这些服务之间是松耦合的，可以根据实际的资源与业务负载合并部署或者分开部署，或做负载均衡，或作集群部署。服务可以云主机上作为一个服务存在，也可以直接部署于公有云，比如RDS，OSS等。

在目前提及的5个子部分中，底层的服务有三个：TCP通信服务，消息推送服务，数据库服务，应用层的服务有两个：Web管理平台，数据接口服务，下面分别对这些服务进行介绍。

### 2．TCP通信服务

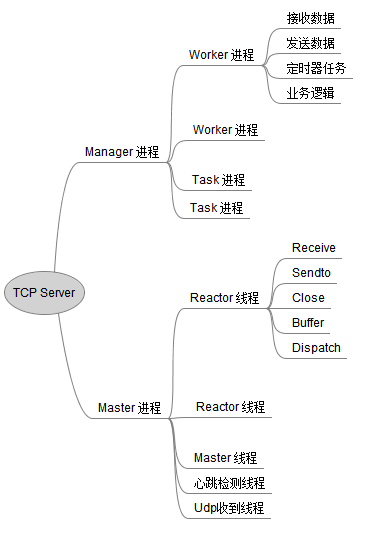
##### 2.1 概述

物联网的使用场景要求用户终端随时能够对设备进行反向控制，因此设备端需与云平台建立一条稳定可靠的通信链路。本方案采用较为常见的TCP长连接方式，无论设备是否开启，只要设备通电即与云平台保持TCP连接，设备端的状态发生变化立即上报数据。

一般而言，物联网通信有如下特点：

* 链路要求可靠，长期保持连接（不涉及反向控制则无此要求）；
* 设备端上报数据频次稳定，频繁；
* 通信数据包不大，一般在1k以内；
* 设备基数大（10k ~ 1000k）；

##### 2.2 方案



（TCP server 底层实现）

物联网应用对TCP通信服务提出了很高的要求，传统的TCP Socket编程采用 select，poll模型，随着连接数的增多，对系统资源占用很大，一般来说单设备支持的数量不超过10K，这样的模型显然无法支持大量设备的连接。epoll是Linux下多路复用IO接口select/poll的增强版本，它能显著提高程序在大量[并发连接](http://baike.baidu.com/view/922692.htm" \t "_blank)中只有少量活跃的情况下的系统CPU利用率。本方案采用了epoll模型，用异步非阻塞的编程方式，提高系统的连接数，通信高并发处理能力。

系统底层（见上图）对Socket建立连接，消息收发，心跳检测，连接关闭，数据缓存，定时器等做了封装，上层采用php写业务逻辑，非常便利。经过实际测试，系统运行稳定，效率非常高。单服务器可以支持的连接数20万（理论上内存够的话，100万亦可），并发通信（不虑业务逻辑）超过10万。考虑到数据库存储，消息推送等业务逻辑，并发的瓶颈已经不在通信层面了。

### 3．消息推送服务

##### 3.1 概述

在移动互联网时代以前的手机，如果有事情发生需要通知用户，则会有一个窗口弹出，将告诉用户正在发生什么事情。可能是未接电话的提示，日历的提醒，或是一封新的彩信。推送功能最早是被用于Email中，用来提示我们新的信息。由于时代的发展和移动互联网的热潮，推送功能更加地普及，已经不再仅仅用在推送邮件了，更多地用在我们的APP中了。

 当我们开发需要和服务器交互的应用程序时，基本上都需要获取服务器端的数据。一般来说实现有两种方法：第一种是客户端使用Pull（拉）的方式，就是隔一段时间就去服务器上获取一下信息，看是否有更新的信息出现。第二种就是 服务器使用Push（推送）的方式，当服务器端有新信息了，则把最新的信息Push到客户端上。这样，客户端就能自动的接收到消息。

 虽然Pull和Push两种方式都能实现获取服务器端更新信息的功能，但是明显来说Push方式比Pull方式更优越。因为Pull方式更费客户端的网络流量，电量，还需要我们的程序不停地去监测服务端的变化。

 物联网使用场景：

**1、设备状态推送：设备状态的变化需要及时的推送到终端设备。**

* + 基数庞大：设备数量一般来说都是10k以上；
  + 频次非常高：设备状态变化一般在10s/次变化；
  + 要求速度快：要求在2-3s以内，否则影响用户体验；
  + 可靠性要求高：要求无丢包现象，否则影响用户使用；
  + App开启才收到：一般是在APP控制界面才有推送需求；

**2、后台消息推送：管理人员通过管理平台向指定用户推送消息。**

* + 一次性推送数量大：单条信息可能会一次性推送很多用户；
  + 频次低：平均不会超过1条/天；
  + 速度要求低：不要求用户立刻收到；
  + 可靠性要求高：对于有些通知，要求不高，重要的通知要求必须到达每个用户；
  + APP不开也能收到：即使APP不开启的情况，也需要能收到推送；

对于前者，要求速度快，稳定性高，否则对于正在控制设备的用户来说，体验将非常差，一般针对少量用户，即设备的持有者。对于后者，对速度要求不高，到达即可，但可能会一次性推送至很多终端用户，比如给所有用户推送一则通知。针对两种场景的需求，可以对推送服务分别部署。

##### 3.2 方案



推送服务构架图

###### 3.2.1 设备状态推送

针对设备状态推送的特点和要求，对比众多推送方案，我们采用MQTT协议实现推送功能。

MQTT是一个轻量级的消息发布/订阅协议，它是实现基于手机客户端的消息推送服务器的理想解决方案，该协议的特点如下：

* 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。
* 对负载内容屏蔽的消息传输。
* 使用 TCP/IP 提供网络连接。
* 有三种消息发布服务质量可进行设定：
  + “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。
  + “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。这一级别可用于如下情况，在计费系统中，消息重复或丢失会导致不正确的结果。
* 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。
* 使用 Last Will 和 Testament 特性通知有关各方客户端异常中断的机制。
* 该协议在客户端有多种实现。

###### 3.2.2 后台消息推送

 后台消息推送使用第三方平台，服务器端调用第三方平台的SDK和接口，根据业务场景设定发送规则，将消息推送到第三方平台。客户端集成第三方平台的SDK，接收第三方平台转发的消息。

第三方平台有商用的也有免费的，可以根据实现情况使用。[极光推送](http://www.jpush.cn/index.jsp)，百度推送都是不错的选择。

### 4．数据库服务

##### 4.1 概述

数据库系统映射了各个实体之间的关系，是所有业务得以运行的基础。然而物联网云平台与普通的系统所不同的地方在于：

1、设备基数庞大；

2、设备数据更新频次高；

3、用户终端随时有可能需要获取最新设备状态；

4、需要保存复杂的关心型数据。

在这种情况下，单一的数据库系统将无法满足此种需求。

##### 4.2 方案



本系统采用 MYSQL + REDIS配合使用的方案，MYSQL用于存储关系型数据，REDIS用于存储设备高并发写入数据，更新设备状态。客户端读取数据时，结合获取MYSQL和REDIS数据。

MySQL 是一种广泛使用的关系型数据库软件，使用标准化的SQL语言，体积小、速度快、成本低，开放源码，在 WEB 应用方面是使用最为广泛的数据库软件之一。

MYSQL支持主从配置，通过主数据库处理事务性查询，从数据库处理SELECT查询。数据库复制被用来把事务性查询导致的变更同步到集群中的从数据库。使用读写分离降低服务器压力，提高数据库服务效率，可靠性。

REDIS 是一个高性能的key-value数据库。它支持存储的value类型非常多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，REDIS支持各种不同方式的排序。为了保证效率，数据都是缓存在内存中。

REDIS数据库读写效率非常高，在数据包大小在100byte时，并发读写可以达到4万左右。而随着数据量的增大，请求数的增多，MYSQL查询效率会显著下降至1k以内，Redis可以对关系数据库起到很好的补充作用，除了设备端状态数据写入，必要的情况下还可以用来作为缓存服务。

REDIS支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得REDIS可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

### 5．接口服务

##### 5.1 概述

客户端软件用户的所有操作，均由服务端的接口提供服务，其中，大部分的接口根据协议的规范，基本上是固定的，然而根据不同设备的功能特点的不同，部分接口需要定制开发。



基础功能 结构图

##### 5.2 功能说明

* 用户系统：用户注册，手机验证，登陆，注销，修改密码，找回密码，修改个人信息等功能。
* 消息推送：
  + - 配件更换提醒：根据设备工作时长推送硬件更换提醒信息。
    - 系统升级提醒：根据软硬件版本号通知用户进行App升级和设备固件升级
    - 商业服务提醒：用户通过管理后台设置提醒，推送至手机客户端。
* 留言反馈：客户端发送留言到云平台，客服人员在管理后台对客户发来的留言进行回复。
* 设备管理：设备绑定，解绑，锁定（不允许他人管理此设备），获取设备状态。
* 设备控制：通过向云平台发送指令，远程控制设备，包括开机，关机，模式，定时等，当设备有报警信息时，对设备发送指令进行处理。
* 系统服务：软件升级，数据分享，公共服务接口（时间，天气，空气质量等），日志上报。

##### 5.3 方案

###### 5.3.1 接口设计

系统采用RESTFUL分层构架的原则，在服务器端，应用程序将状态和功能划分为各种资源。每个资源都使用 URI (Universal Resource Identifier) 得到一个惟一的地址。所有资源都共享统一的界面，以便在客户端和服务器之间传输状态。使用的是标准的 HTTP 方法，GET、PUT、POST 和 DELETE。这样，服务对外通过规范的API实现，简化了整个系统架构，改进了子系统之间交互的可见性。

服务端与客户端的数据传输格式采用标准json格式，数据体积小，传输快，通用性强。

服务器端针对可能出现的各种异常，设定标准的错误码，以便于测试以及给用户相应的说明与提示。

###### 5.3.2 服务设计

服务采用PHP开发。PHP作为Web开发的强大语言之一，具有开放源代码，跨平台性强，开发快捷，面向对象等诸多优点。

服务使用MVC的设计模式，将数据层，控制层进行分离，结构清晰，便于二次开发和维护。

服务运行于Linux操作系统运行，使用Ningx作为Web服务器，Mysql作为数据库存储，通过对系统配置的调优，可以保障系统最佳的运行效率，稳定性和安全性。



* index.php 作为前端控制器，初始化运行系统所需要的基本资源。
* Router 检查 HTTP 请求，以确定谁来处理请求。
* 如果Cache文件存在，它将绕过通常的系统执行顺序，被直接发送给浏览器。
* 应用程序控制器装载之前，HTTP 请求和任何用户提交的数据将被过滤。
* 控制器装载模型、核心库、辅助函数，以及任何处理特定请求所需的其它资源。
* 最终结果以Json格式返回终端用户。

###### 5.3.3 身份认证

RESTFUL 接口是无状态的，为了保证接口的安全性，需要对客户端的访问进行身份认证。用户登录时随机生成Token以及过期时间，后续的访问使用Token作为用户身份的标识。Token过期之后会要求用户重新登录以获得新的Token。

### 6．管理平台

#### 6.1 概述

物联网云平台是为了传统家电实现物联网功能提供的服务端平台，管理后台是云平台的一个重要组成部分，使用者为厂商或者系统管理员，通过管理后台可以查看用户信息，操作数据，设备采集的环境数据，并对这些数据进行分析，了解用户的使用习惯，用户的环境情况，设备的状态，更好地为顾客服务。同时，通过管理后台，还可以与用户进行交流，跟直接的做售后服务。

#### 6.2 功能结构

管理后台由系统管理，设备管理，用户管理，运营管理四个部分组成。



##### 6.2.1 系统管理

系统管理为平台的一些基础模块和功能。如系统设置，用户管理，权限设置，模块管理等。

* + **登录模块**
* **登录验证：**验证用户名密码是否正确，用户的密码经过做salt混淆，及md5双层加密。
* **权限验证：**用户登录完成，验证用户的身份，根据用户的权限显示用户的操作界面。不同角色的用户显示的界面和功能是不一样的。
* **SESSION验证：**当用户登录成功之后，在服务器保持SESSION信息，知道浏览器关闭，服务器注销SESSION。
  + **系统设置**
* **运行状态：**能够看到服务器当前的运行信息，比如cpu，内存，硬盘的使用量，操作系统信息，web服务软件的版本信息。
* **系统设置：**
  + 配置系统变量。
  + 填写短信发送配置信息，电子邮件配置信息。
  + 系统安全设置：IP白名单，黑名单设置；
* **修改密码：**修改自己的账号密码；
  + **权限管理**
* **用户管理：**增加，修改，删除，查找用户，为不同的用户设置不同的角色；
* **权限设置：**将不同的功能设置入权限体系，可以增加，修改，删除权限；
* **角色设置：**设置角色，不同的角色有不同的权限，可以增加，修改，删除角色；
  + **日志管理**

通过记录用户的登录，退出信息，操作行为，实现对用户操作的管理，也增强了系统的安全性。

##### 6.2.2 产品管理

产品是是设备，APP的基础信息，在新的产品即将上市之前，需要提前做好各种设定。

* 产品设置：增加，设定，修改，删除产品，将产品与app相关联。
* 设备授权：批量导入设备SN号码，为即将出厂的设备进行授权。
* APP管理：为产品所对应的物联APP进行设定，提供版本管理，下载链接。
* 固件管理：为产品所对应的物联固件机型设定，提供版本管理，下载链接。

##### 6.2.3设备管理

设备管理一方面为厂家的管理人员提供设备检索，查看的功能，另一方面提供了基于设备状态历史信息的各种数据分析报表。

###### 6.2.3.1设备分析

* 设备概况：当前所有设备的统计信息
* 新增设备：以折线图的方式显示本月，上月以及历史各个时期的设备新增情况。
* 设备分布：以饼状图的方式显示设备的地理分布情况。
* 设备在线：以折线图的方式显示今天，昨天以及近期设备每个时段在线的情况，从而了解服务器的服务数量，负载情况。
* 运行设备：以折线图的方式显示今天，昨天，以及近期设备每个时段处于运行的设备的数量，从而了解用户使用设备的总体情况。
* 异常设备：显示当前处于异常的设备，便于客服人员联系客户，帮助客户开展服务。

###### 6.2.3.2 设备管理

* 提供设备检索，查看设备，查看设备绑定用户，管理设备分类等功能。

##### 6.2.4 用户管理

用户管理一方面为厂家的管理人员提供用户检索，查看的功能，另一方面提供了基于用户行为的各种数据分析报表。

###### 6.2.4.1 用户分析

* 用户概况：当前用户的统计信息
* 注册分析：以折线图的方式显示本月，上月以及历史各个时期的设备新增情况。
* 客户端分析：对用户的客户端进行统计分析，以了解用户使用客户端的类型，软件版本，分辨率等情况。
* 用户行为分析：统计用户在使用客户端时的主要使用功能，偏好设置，使用时间短，时长等。

###### 6.2.4.2 用户管理

* 提供用户检索，查看用户信息，给用户推送消息，查看用户绑定设备等功能。

##### 6.2.5 运营管理

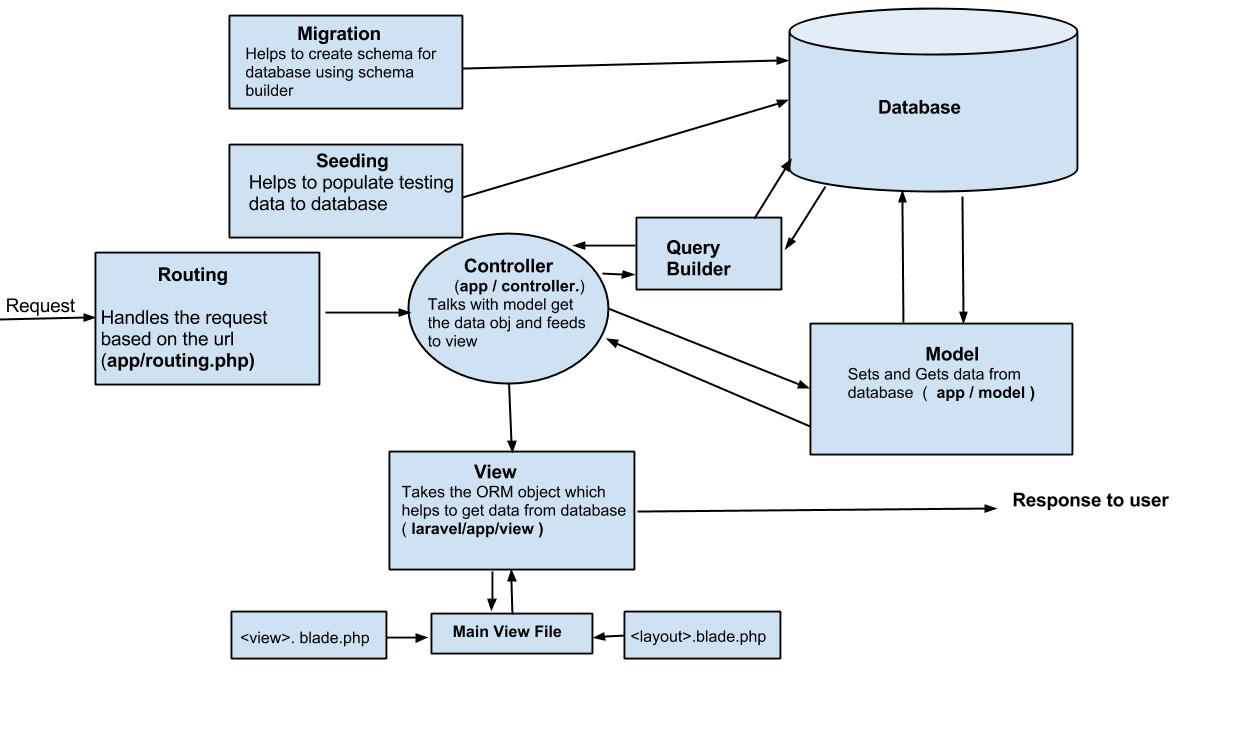
###### 6.2.5.1 留言反馈

* 反馈列表：用户反馈列表，提供按照类别，状态，用户进行检索。
* 回复反馈：管理员可以对用户的反馈进行回复，并显示在用户的手机客户端上。
* 分类设置：可以设置反馈的类型，以及用户在前台的不同的反馈的表单。

###### 6.2.5.2 消息推送

* 可以根据用户的不同的属性，针对性的对不同用户群组实施推送。

#### 6.3 技术方案



框架结构图

系统采用成熟开源的框架Laravel进行开发，该框架是一套简洁、优雅的PHP Web开发框架，该框架使用群体广，社区活跃，文档丰富。

Laravel代码规范，支持PSR-0，PSR-4，支持Composer 安装，有丰富的插件，通过Composer可以很容易的安装扩展包。

Laravel集成了高级的PHP Active Record实现Eloquent ORM。它能方便的将“约束（constraints）”应用到关系的双方。同时，Eloquent原生支持Fluent中查询构造器（query-builder）的所有方法。

Laravel具备非常自由灵活的路由系统，只需使用路由名称，框架就会自动帮你创建正确的URI。这样你就可以随时改变你的路由，框架会自动更新所有相关的链接。

Laravel使用了自动加载类，极大的简化了类的加载工作，使用时只需要通过命名空间的方式正确的引入类名即可。当你想加载任何库或模型时，立即使用就行了，框架会自动帮你加载需要的文件。

Laravel是易于测试的，单元测试是Laravel中很重要的组成部分。框架自身就包含数以百计的测试用例，以保障任何一处的修改不会影响其它部分的功能，通过Artisan命令行工具就可以运行所有的测试用例。

总而言之，使用Laravel开发管理后台这样中型的项目，非常合适，不但效率高，而且稳定性好，易于控制代码规范，易于测试，易于扩展，易于维护，是非常理想的开发方案。

#### 6.4 安全措施

安全问题不容忽视，虽然管理后台不涉及金融，但安全方面的疏忽有时会给企业带来一些不必要的损失，影响整个系统对用户提供的服务。

本系统采用全访问的安全措施，尽最大的可能降低系统的安全风险。

* **合理设置权限**
  + 1. 通过设置合理的权限。从最坏情况的可能性，对不同权限带来的不同程度的危害对权限进行评级，采用最小权限原则，将可能造成严重危害的权限进行严格控制，做到万无一失。
    2. 涉及关键权限的功能需要通过手机短信验证才能够进行操作，从而避免因用户信息泄露带来的危害。
* **系统级的安全防护**

1. 本方案采用公有云方案，公有云一般都配置了高安全，高性能的防火墙技术，可以提供DDoS防护，主机入侵防护，以及漏洞检测、木马检测等一整套安全服务，最大程度的抵御了一般的攻击行为。
2. 操作系统配置IPTABLES防火墙，关闭不必要的端口，提高系统的安全级别；
3. 合理配置文件权限，防止黑客渗透攻击；
4. 数据库与应用分离，配置权限，定期备份数据，保障数据安全。
5. 撰写Shell脚本，自动分析服务器日志，及时发现及跟踪黑客嗅探的可疑行为。

* **平台安全机制**

1. 用户密码增加混淆，并进行双层MD5加密，防止密码泄露被黑客破解。
2. 平台上线前经第三方安全测试，完善系统，提高系统的安全级别。
3. 设防火墙，过滤非法探测，防止SQL注入，记录操作日志，提高平台安全。
4. 设置IP白名单，黑名单以及必要的规则，有效防止非法人员入侵，增强平台的安全性。
5. 过滤用户输入，防止跨站脚本攻击及SQL注入。
6. 平台合理的权限设置，详细的登陆及操作日志记录，能够对非法的登陆及操作行为进行防范和记录。

### 7．部署方案

物联网应用初期，设备数量不多，可以简单的将业务与数据分开部署，定期的做数据备份，采用自动化监控策略，保障系统的稳定运行。随着业务的增加。单台服务器难以应对大量设备，高并发的数据请求，需要采取分布式部署策略。由于整个系统在设计初期已经考虑了分布式部署的问题，系统的各个服务之间只需要做简单的配置即可进行横向拆分。

系统采用基于阿里云的公有云平台构架，不必再关心硬件层面的问题，比如硬盘损坏，电源故障等问题。阿里云提供了丰富的构架组建，可以根据不同的业务规模进行各种伸缩性的构架与调整。

可用的服务有：

1、ECS 云主机：小规模应用时可用

2、RDS 关系型数据库：小数据量时使用

3、OTS 开放结构化数据服务：大数据量时使用

4、SLB 负载均衡服务器：并发量〉10万时采用

5、云盾 防火墙：基础服务，提供基础的安全防护

6、云监控 服务监控系统：基础服务，服务器运行自动化的监控及报警

7、MQS 消息队列服务：当对服务应用采用分布式构架时，需要使用此服务。

分布式架构原理图如上所示，总体的思路就是分而治之，具体的技术点可分为：

#### 7.1 应用服务与数据服务分离

随着业务的发展与访问量的增大，一台集数据与应用于一体的Server不能满足性能上的需求。这就应该将应用和数据分离到三个服务器集群：应用服务器集群、文件服务器集群和数据库服务器集群。

这三类服务器集群对硬件的要求各不相同，应用服务器需要处理大量的业务逻辑，因此需要更快更强大的CPU；数据库服务器需要快速磁盘检索和数据缓存，因此需要更快的硬盘和更大的内存；文件服务器需要存储大量用户上传的文件和系统生成文件，因此需要更大的硬盘。

应用与数据分离后，不同特性的服务器承担不同的服务角色，云平台的并发处理能力和数据存储空间得到了很大改善，支持业务不断扩展。

具体到应用服务器，又可以分为：TCP服务器，推送服务器，API服务器，管理后台服务器。

具体到数据服务器，又可以分为：缓存服务器，Mysql服务器。

#### 7.2 使用应用服务器集群和负载均衡LVS

使用集群是web系统解决高并发、海量数据问题的常用手段。当服务器处理资源不足时，横向扩展增加新服务器比提高单台Server性能要更具有性价比。

对于云平台架构而言，只要能通过增加一台Server的方式改善负载压力，就可以以同样的方式持续增加服务器不断改善性能，从而实现系统的可伸缩性。

通过负载均衡调度服务器，可将来自Client的访问请求分发到应用服务器集群中的任何一台服务器上，如果有更多的用户，就在集群中加入更多的应用Server，使应用服务器的负载压力不再成为整个网站的瓶颈。

#### 7.3 数据库读写分离、主从热备

由于物联网云平台对实时性有较高的要求，故数据读取都是不经过缓存而直接读取数据库，这对数据库的压力非常大，故我们采取Mysql Cluster集群，进行读写分离、主从热备。

目前大部分主流数据库都提供主从热备功能，通过配置两台数据库的主从关系，可以将一台数据库服务器的数据更新同步到另一台服务器上。正是利用数据库这一功能，实现数据库读写分离，从而改善数据库负载压力。

应用服务器在写数据的时候，访问主数据库，主数据库通过主从复制将数据更新同步到从数据库，遮阳挡应用服务器读数据的时候，就可以通过从数据库获得数据。为了便于应用服务器访问读写分离后的数据库，我们在应用服务器端使用专门的数据访问模块，使数据库读写分离对应用透明。

## 三、客户端技术方案

### 1. APP实现方案

##### 1.1 总体架构

App开发工作分为三大部分：UI/UE界面交互层的设计与实现，设备与业务逻辑的设计与实现，以及底层的协议与通信逻辑的设计与实现。

其中UI/UE界面交互层与设备业务逻辑层是与具体的设备相关，要根据具体的需求来设计。而协议与通信逻辑是通用的，在SkySDK中实现。



##### 1.2 SDK封装

###### 1.2.1 设计原则

* **1、模块化解耦：**

将APP与Server/Device间的通信细节屏蔽，将大小循环的路由选择屏蔽，让APP专注于功能和UI/UE设计，提高生产效率。通信逻辑与UI逻辑的分离使得双方的修改都不会影响到对方，只要保证接口的稳定。这样的设计提高了APP端程序的灵活性与可维护性。

* **2、本地模式：**

降低云端负载，迅速感知设备状态改变

* **3、大数据采集与上报：**

实时采集用户APP使用行为，设备控制行为信息，定期上报云平台供大数据分析

###### 1.2.2 主要功能模块



* **标准服务：**

包括大数据采集服务、LBS定位服务、第三方登录与分享服务以及短信验证服务。

其中大数据采集服务负责采集用户行为与设备数据。

* **WiFi配网：**

App端支持三种配网方式，分别是 SoftAP、WebConfig、Smartlink

其中SoftAP是AP模式的配网方式，App实现自动切网，类似一键配置，成功率接近100%

WebConfig是网页配网方式，类似SoftAP，通过HTTP协议配网，成功率高。

SmartLink是一键配网方式，用户无需切换网络，快捷高效。

* **消息层：**

消息层分为推送与HTTP，其中推送层采用物联网轻量级传输协议MQTT协议与第三方推送相结合的方式，同时保障推送的高速与可靠。接口层采用HTTP/HTTPS的方式通信，保证可靠性与安全性。

* **传输层：**

传输层分为大循环（远程）与小循环（本地）两种通信方式。其中大循环是完全通过外网对设备进行控制与监控。小循环是通过内网对设备进行控制与监控，更加快捷高效且节省流量，降低云端负载。

这两种模式目前可以根据不同的网络环境进行无缝切换，相互配合。

###### 1.2.3 本地模式（小循环）概述：

* **发现：**

1) WiFi 模块上电后，30 秒内，每一秒发送一个上电宣告包

2) 上电宣告包采取广播，广播地址：255.255.255.255，端口：2415；

3) 收到 SDK 发送的搜索包后，立即回复搜索相应包；

4) 搜索的响应，使用 UDP 单播；

5) 宣告包和搜索相应包按照协议规定严格执行；

* **获取 pkey：**

1) 模块初始化后，自身随机产生 pkey，用于云端注册和 APP 绑定的安全鉴权；

2) pkey为 10-32 个字节的随机字符串；

3) 模组收到进入配置的命令后 （进入SmartLink和SoftAP） ，按照MCU在握手过程中发送模组的bindable time 开启 bind 功能；

4) 若模组当前未处于 bindable 状态，获取 pkey的时候返回空；

* **控制与状态：**

1) 模组采用透传的策略将控制数据进行转发；

2) 收到 SDK 的控制命令时，将 0x03 包头数据发送给 MCU；

3) 收到云端的控制命令时，将 0x02包头数据发送给 MCU；

4) 收到 MCU 的主动状态上报时，先回复 ACK，再发送给本地 SDK，再同时发送给云端；

5) 当收到 MCU 对某一个查询命令的返回时，模块仅将数据单独回复给查询命令发起者；

6) 当收到 MCU 对某一个控制命令的返回时，模块仅将 MCU 的 ACK 单独回复给控制命令发起者；然后将 MCU 上报的状态进行广播，包括服务器；

* **模组与心跳**

1) 在连续 4 秒未收到模组数据时，SDK 主动向模块发送心跳数据，模块收到后返回心跳确认；

2) SDK 连续 12 秒收不到来自于模块的数据（包含心跳和其他数据） ，认为与模块连接失效，SDK 主动断开与模块的连接；

3) 模块连续 12 秒收不到来自于 SDK 的数据（包含心跳和其他数据） ，认为与 SDK 连接失效，模块主动断开与模块的连接；

* **Reset清空机制**

需要清空：

1) DNS 解析保存的 IP；

2) pkey；

3) 路由器 SSID、路由器 PASS；

### 2.微信实现方案

##### 2.1 概述

基于wifi的微信物联网方案，目前有两种方式，一种是基于自己开发的html页面，一种是基于微信模板，前一种基本上不用微信的设备接口（除了设备配网，扫描二维码），后一种则大量依赖微信的设备接口。前一种方案灵活度高，后一种方案用户体验好。

##### 2.2 方案

（设备控制模型）

##### 2.3 功能

###### 2.3.1 菜单生成

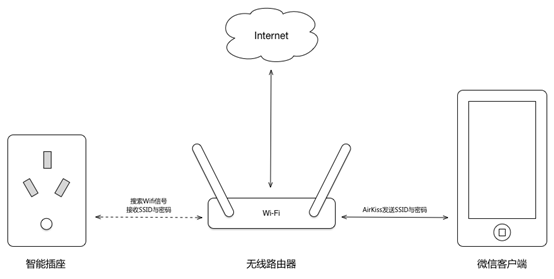
指定每个菜单所对应的微信服务平台的URL。

###### 2.3.2 用户授权

授权后可以获取用户的OpenID及用户信息，用户无感知，程序完成。

###### 2.3.3 设备配网

通过调用JSAPI触发AirKiss，实现Wifi设备配网。



用户使用Air Kiss的交互流程如下：

1、用户按下智能插座上的配置按键，AirKiss指示灯闪烁，智能插座进入信息接收状态。

2、用户打开微信手机客户端，进入设备的联网配置界面（设备厂商开发的HTML5页面），唤起AirKiss的SSID与密码发送界面，当前无线网络环境下无线路由器的SSID已经默认选中，用户只需要填写密码，然后点击发送即可。

整个AirKiss过程将在15秒内完成。

###### 2.3.4 设备绑定

调用我方的API接口，实现添加绑定设备的流程，与APP过程一致，绑定完成后跳转至设备列表页，可以看到新添加的设备。

绑定设备，设备SN作为标识，调用JSAPI，可实现扫描设备SN的功能，也可以手动输入。

###### 2.3.5 设备解绑

调用我方的API接口，解除微信OpenId与设备的绑定关系，从我的设备列表中移除该设备。

###### 2.3.6 设备列表

调用我方的API接口，实现设备列表展示。

###### 2.3.7 设备锁定

更新设备状态为锁定，禁止其他人绑定。

###### 2.3.8 设备控制

* 控制页面：制作设备控制h5页面及交互效果，通过websocket协议实现与服务器保持长连接通信。
* 订阅设备：进入控制页面，通过websocket发送订阅设备的请求，实现设备订阅，实现心跳保持。
* 设备控制：通过http请求，通过厂方API接口，向设备发送控制指令，实现设备控制。
* 数据推送：设备端状态变化，服务端通过websocket服务实现数据推送。
* 协议解析：设备端推送过来的数据，通过js进行解析，在html页面展示设备状态变化的动态效果。

## 四、设备端技术方案

设备端通过嵌入一款实现云平台协议的Wifi模块，实现设备端联网的功能。Wifi模块涉及的协议要点大致如下：

### 1. 广域网通信

* **TCP长连接**：

为了保证Server反向控制WiFi的实时性和高效率，WiFi与Server之间的通信采用TCP长连接方式，而长连接的维持需要心跳机制来保证。

WiFi端作为TCP Client，主动向Server发起三次握手。断开时Dev端主动断开。（正常情况下一直保持连接，除非遇到异常等特殊情况才主断开）

* **心跳维持：**

设备主动向Server发送的用来维持TCP长连接的短包。一般维持在30s-60s左右，防止传输过程中转发路由的NAT端口映射老化的问题。Server则根据此心跳包来判断WiFi的在线情况。

* **异步上报：**

Server -> Dev 控制命令的发送是通过TCP长连接推送过去，而Dev -> Server的状态回馈是通过TCP长连接主动上报。Dev接到命令后立即给server反馈 ret:200，告知已收到并执行。然后Dev等待MCU执行完毕后将操作后的状态异步反馈给server。

* **固件升级：**

采用内网和远程两种升级方式。两者从指令层面相同。WiFi模块需要在升级前进行同步回复来确认收到，升级后进行异步回复来告知升级的结果。

* **流量控制：**

物联系统的并发与流量控制是云端性能优化的关键。在很多场景下，并不需要对设备的实时监控，故需要由云端对WiFi模块的状态上报行为进行约束与控制，来保证整个系统的稳定，减少不必要的流量和带宽损耗。

### 2. 局域网通信

* **UDP广播发现设备：**

与设备通信的前提是知道设备在内网的IP，由于设备在内网的IP由路由动态分配，所以无法预先知道。只能采用在局域网内UDP广播的方式进行设备发现。只要WiFi模块端收到指定查询串就会立即回复模块相应信息。

* **内网TCP长连接：**

内网采用TCP长连接的通信方式，设备控制、上传状态、协议格式与外网相同，此处WiFi模块作为TCP Server，手机App作为TCP Client。

* **长连接的维持：**

若App希望通过内网实时监控设备状态，则建立TCP连接后，只需按照指定频率发送心跳包，来维持连接不被WiFi模块收回，即可实时收到设备的上传状态协议包。App退出实时监控状态时主动断开TCP连接。

* **长连接的回收：**

由于WiFi模块的TCP连接数有限，需要定期清理长连接。清理的规则是判断15s之内是否有收到App的有效数据（包括查询、命令、状态回复以及心跳），如果15s内没有收到任何数据则回收TCP长连接（WiFi模块主动发起TCP连接关闭）。

* **长连接数的监听：**

App在建立内网TCP长连接的时候，需要知道WiFi模块当前有效的TCP连接数，作为是否维持长连接的依据。如果连接数接近WiFi模块的最大可承受连接数，则为了WiFi模块的稳定运行，则放弃长连接采用短连接或走大循环。

* **双信道上报：**

设备MCU的所有状态上报数据都将同时发送到服务器（T口）和内网（L口），手机App可以使用根据sn过滤掉后到的重复数据包(两条通道的sn号相同，便于判断不同通道的相同数据包，避免误决策)

* **AP模式与AP配网：**

AP模式是WiFi模块建立一个独立的AP热点，可供App直接连接而无需经由第三方路由器连接的直连模式。在AP模式下，除了WiFi本身绑定到一个固定IP上之外，其他的与正常STA模式完全相同，包括对所有协议指令的支持。

### 3. 串口通信

* **协议组成：**

协议分为协议头，协议长度，协议体（包含键值对与分隔符，或者二进制串），结束符；

* **协议体与编码：**

采用键值对的形式表示状态或者命令，默认采用 ASCII 编码。

但如果考虑传输效率，需要采用二进制方式传输协议体的内容。具体内容格式由MCU与云端协商，WiFi模块无需解析，但需要将外网和内网传输的 data 字段进行**BASE64**解码后转发给MCU，将MCU上报的二进制body进行BASE64加密，封装到data 字段中上报。

MCU如果有二进制传输的需求，需要在 基础信息中增加 isbin 的字段，WiFi模块在login登录之后查到此字段，便对MCU上报与向MCU下发的数据body进行BASE64编解码。