

www.shinelinker.com

闪麟网络科技

EZLINKER

物联网平台设计文档

2020

目录

1. 设计背景.....	3
2. 产品定位.....	3
3. 产品性质.....	3
4. 盈利模式.....	4
5. 技术栈.....	5
6. 核心功能拆分设计.....	8
7. 核心架构设计.....	15
8. 应用市场.....	20

1. 设计背景

2017年在大学的时候设计了 EasyLinker 的第一个版本,当初写那个版本的原因是自己准备做一个 yealink 替代品.第一个版本是一个非常基础的版本,仅仅包含了 MQTT 消息转发代理.

后来经历了一系列事情,2018年我大学毕业进入社会,再也没精力去维护了,勉强做了 EasyLinker 第二个版本,但是第二个版本太复杂了,于是在 2018年年底也放弃维护.

2019年,我找到了问题所在,主要是我们没有系统的设计过功能才导致流产,2019年我和其他开发者一起讨论研究了大半年需求,又重新设计了 EasyLinkerV3 版本.

但是问题还是存在的,v3 的设想实在是太大了,不适合个人小团队开发,最终还是流产.2019年夏天我和其他开发人员一起讨论了一次:要不就从 v3 设计理念提取出来一个版本,专门做一个领域的产品?这个提议得到了通过,于是我们设计出来了 EasyLinker 的子版本:EZlinker(因为 EasyLinker 已经是别人的品牌,我们换了同音字母 EZ).

EZlinker 和其他物联网平台的不同之处在于我们关注于厂家批量生产设备的场景,提供大量便捷的管理功能,还有物联网基础服务功能,方便厂家快速构建自己的流水线和产品.此处的关注点是批量生产设备情况下带来的集中管理问题.

目前关注本项目的人也比较多,因此我们决定花精力和时间认真做一个版本发布出去.

2. 产品定位

产品针对企业用户开发,私有化部署场景下运营,针对中小企业在物联网项目下的技术开发耗时耗力或者技术短板的痛点,推出一个一体化,简单化,轻量级的解决方案.

产品的用户群体为企业用户,或者个人用户,作用领域为物联网项目开发,管理,运营层面.

3. 产品性质

本产品以开源协议的形式发布,推崇商业友好的 ApacheV2 开源协议,所有贡献者共享代码版权.产品整体架构图如下所示:



整体架构设计包括设备接入层,数据汇总层,设备管理层,开发者接口层,整体设计构思图如图 1、2 所示。

EZLinker 版本分为两个,一个是开源版,一个是商业版,区别见下表 1。

区别\版本	开源版	商业版
HTTP 协议接口	√	√
COAP 协议接口	√	√
MQTT 协议接口	√	√
TCP 协议接口	√	√
UDP 协议接口	√	√
视频传输【RTMP】接口	√	√
数据持久层支持	√	√
数据可视化支持	√	√
文件存储支持	√	√
Docker 部署镜像	√	√
机器学习框架集成	×	√
P2P 形式 API 市场	×	√
开发者模式	×	√
分布式部署	×	√
用户计费系统	×	√
硬件市场	×	√
插件	×	√

4. 盈利模式

前期计划先以开源为依托,发展自由软件市场,然后通过孵化项目的形式,和企业客户建立合作关系,最终提供增值服务。

同时后期展望是不断地积累起来 EZLinker 的应用案例形成一个 APP 市场,该市场提供常

见的物联网解决方案,以通用的接口标准规范化接入,最终形成一个物联网云系统,用户可以自由组合自己的产品,直接一键发布产品.通过此模式收取增值服务费.

同时为了调动用户的积极性,用户也可以将自己的方案发布到市场供给市场用户使用,从中赚取一定的费用.

5. 技术栈

本项目技术栈比较繁杂,但是总体来说不难,算是中等项目规模.主要用到的技术栈如下罗列:

5.1 WEB 后台

WEB 后台我们用了 SpringBoot 及其周边框架做的,想必做 Java 的人非常熟悉.这个就不多赘述.

前端项目则是基于 Ant design 进行了少量自定义开发,也是很常见的技术,对于专业人员二次开发和使用应该问题不大.

所有业务组成技术如下表所示:

名称	功能
Springboot	后台管理系统主要框架
MybatisPlus	数据持久层框架
Netty	实现 WEB Terminal
PahoMqtt	实现 Mqtt 代理
Redis	系统数据缓存
MongoDb	设备数据,日志保存
AntDesign	前端框架

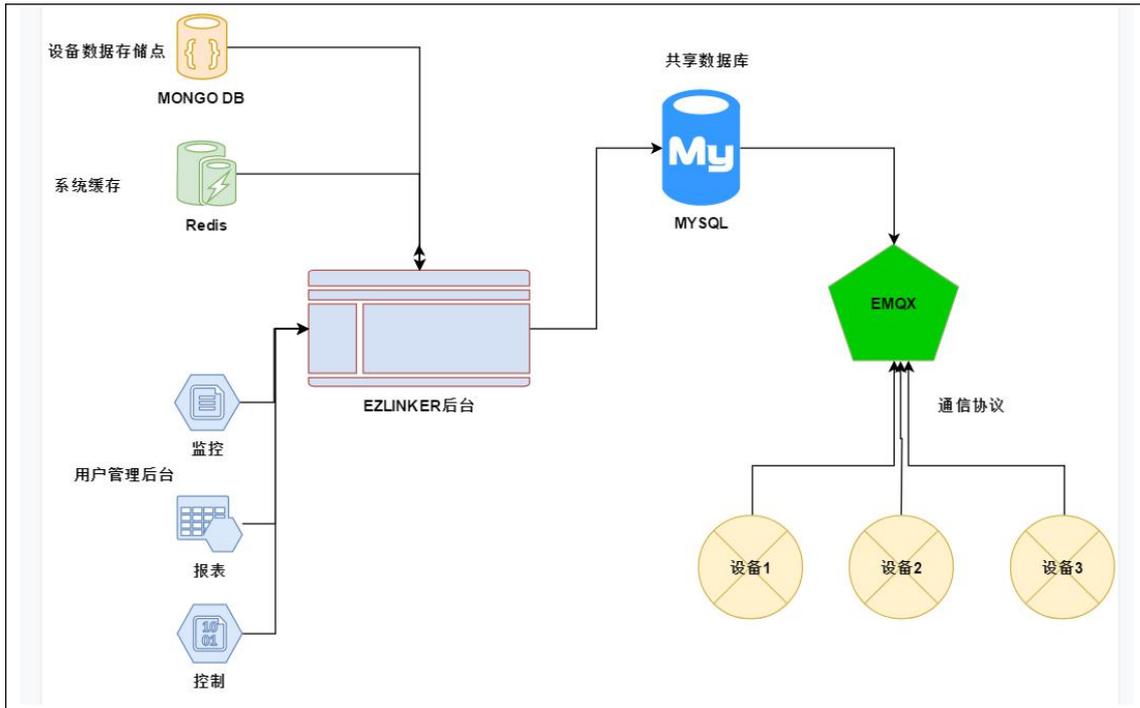
所有用到的三方服务下表所示:

名称	功能
阿里云短信 API	阿里云提供短信功能
又拍云文件存储	如果用户没有文件存储资源,可以使用又拍云的文件存储功能
阿里云邮件 API	阿里云提供邮箱功能

5.2 消息代理

消息代理我们使用了 EMQX4.X 版本,也就是本项目比较难操作的地方,好在 EMQX 的文档比较完善,大量资料可以阅读查询.

系统整体架构图如下所示:



其中 EMQX 和 EZlinker 是单独运行的,两者通过共享数据库实现数据统一,EZlinker 负责所有的业务内容,包含设备的管理,监控等,同时负责设备数据筛选清洗.EMQX 负责设备连接和通信协议转换,属于接入层.

5.4 流媒体服务

流媒体服务主要用在监控场景下.流媒体目前有两种解决方案:一种是使用三方服务,比如阿里云,七牛云等等.但是随之而来的问题是这些服务商的费用很高,或者有些敏感数据不愿意托管在别人的云上.另一种就是私有化部署方案.针对这种情况,设计了两套解决方案:

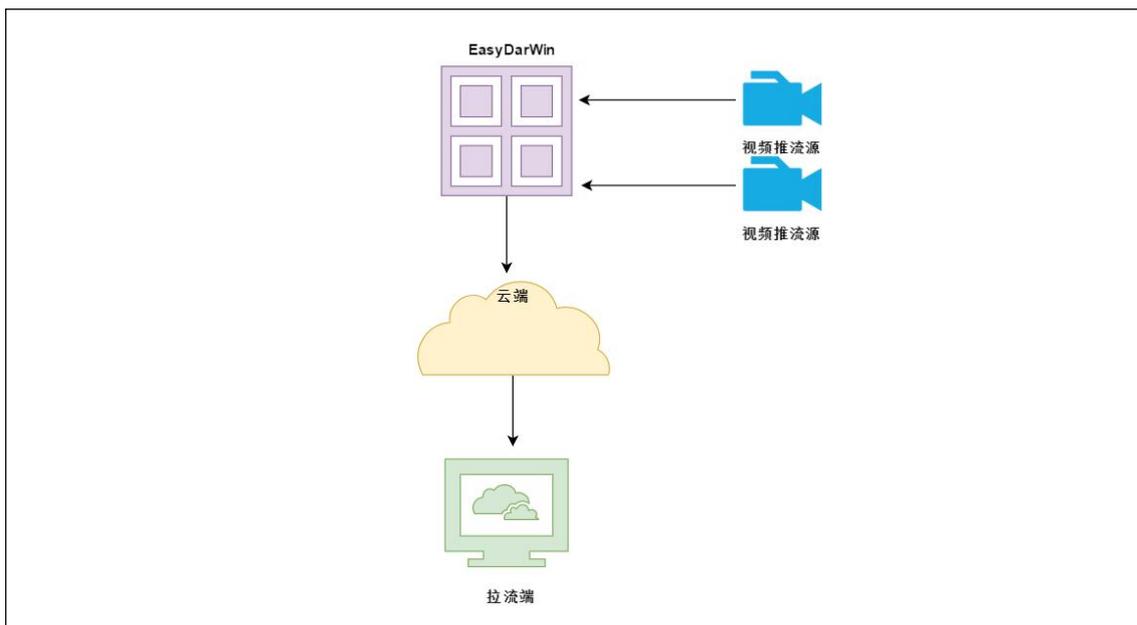
I.私有化部署

私有化部署面临的问题就是技术难点比较多,需要足够的技术实力,我们通过测试和对比发现目前比较稳定而且文档多的流媒体服务器产品:EasyDarWin.该产品是开源项目,代码免费共享,可以投入商业用途,项目地址为:<https://github.com/EasyDarwin/EasyDarwin.git>.

II.使用三方服务

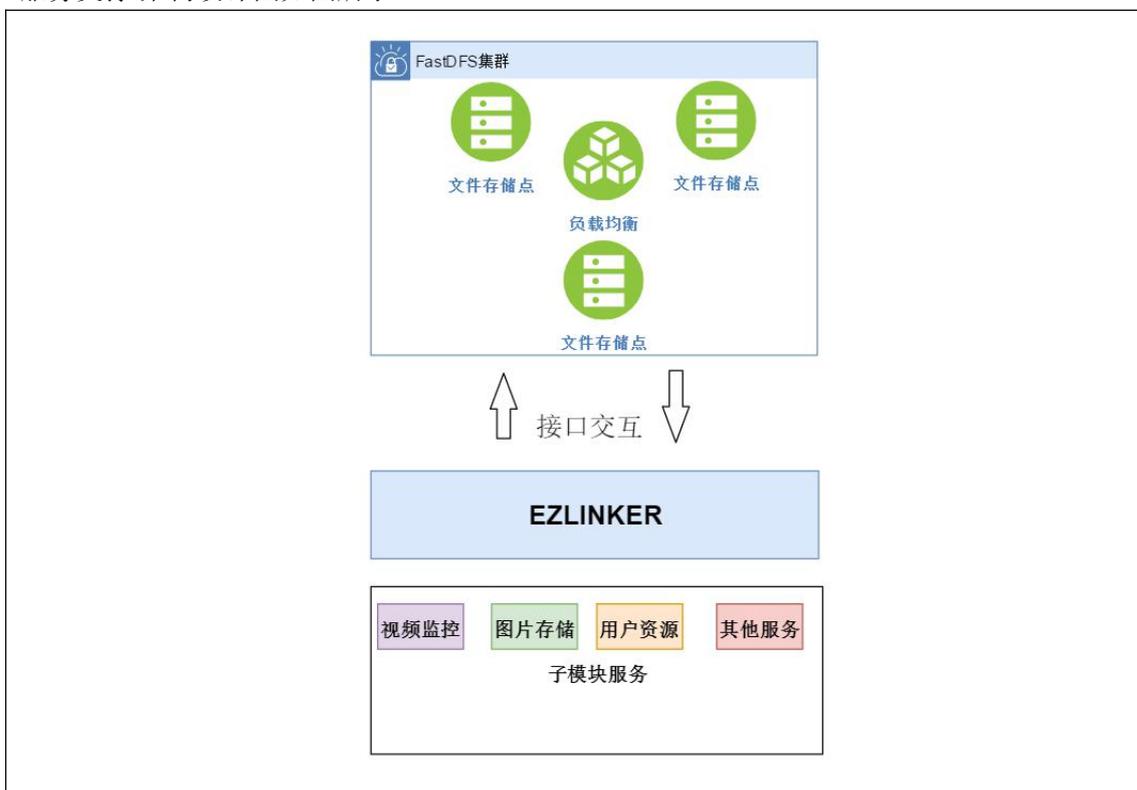
就是上面提到的阿里云或者七牛云的视频流服务,这种模式比较利于技术力量薄弱,精力有限的小团队和公司.

架构图如下图所示:



5.5 文件存储

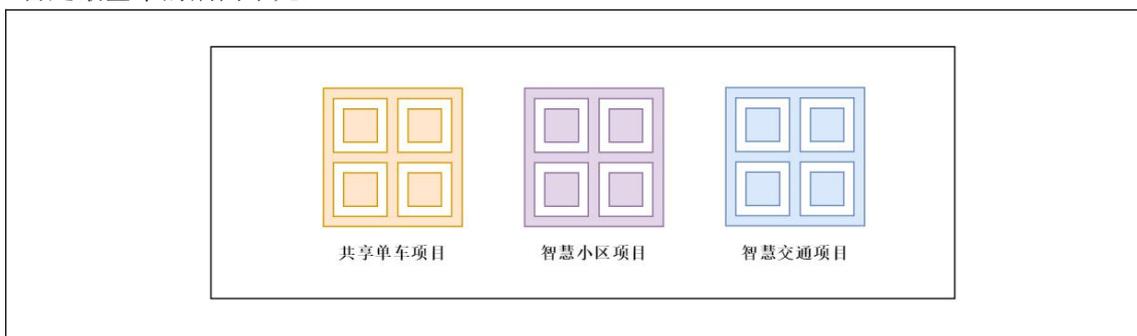
部分业务会产生文件,例如监控,可能会有大量的图片和视频文件产生,传统方式管理文件已经变得不可行,所以采取文件存储系统来解决,目前文件存储系统五花八门,有专门针对大数据的 HDFS,有小巧的 FastDFS.在本项目中为了简化,我们使用了 FastDFS 来作为文件存储服务支撑.架构设计图如图所示.



6. 核心功能拆分设计

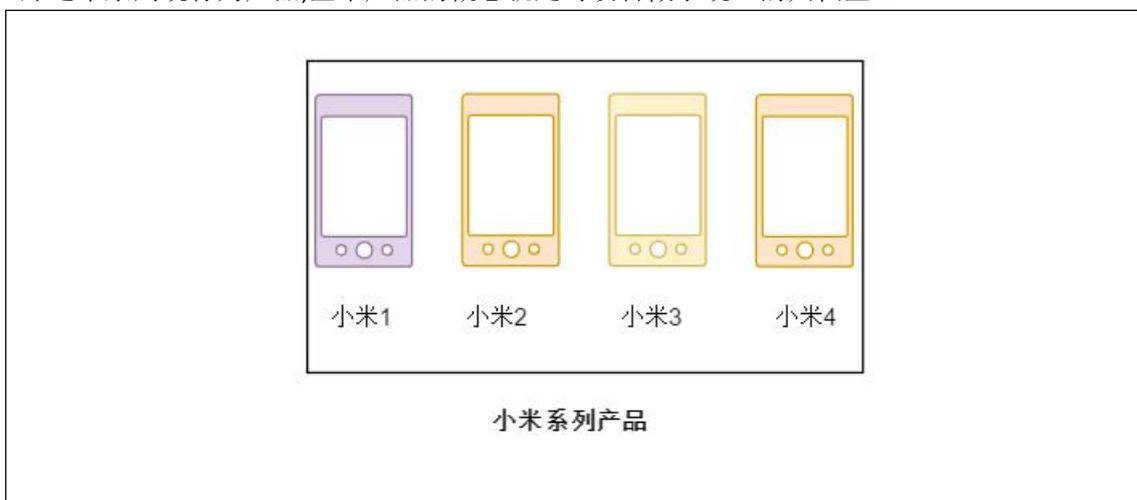
6.1 项目管理

项目是最上层的概念,表示执行的事件作用点和标记.简单来说就是区分不同的场景.比如我们新建一个共享单车项目,接下来又要有共享电动汽车,我们只需要再新建项目即可,项目是最基本的隔离单元.



6.2 产品管理

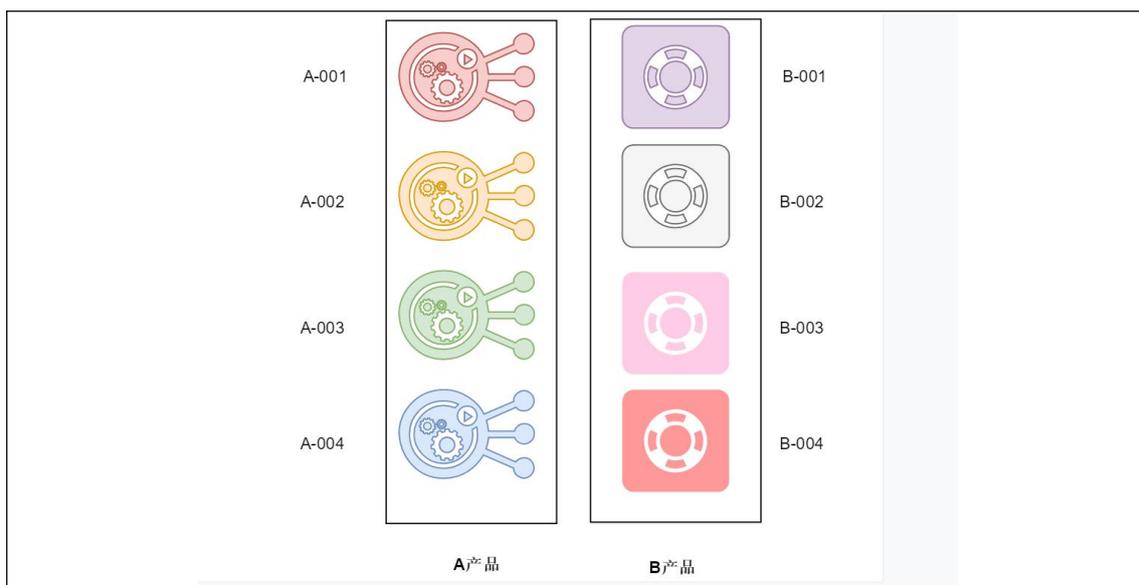
产品是对设备的一个抽象,继续上面的共享单车项目,我们要生产一个 A001 型号的系列,那这个系列统称为产品,整个产品的概念就是对设备做了统一的归档整理.



6.3 设备管理

设备是产品的一个实例,例如共享单车项目下有 A001 系列的产品,我们生产一辆车,就叫 A001-A,编号为 A,则这辆车就是一个设备.

产品-设备 之间的关系如图所示:



其中产品被制造出来以后成为具体的设备.

6.4 模块管理

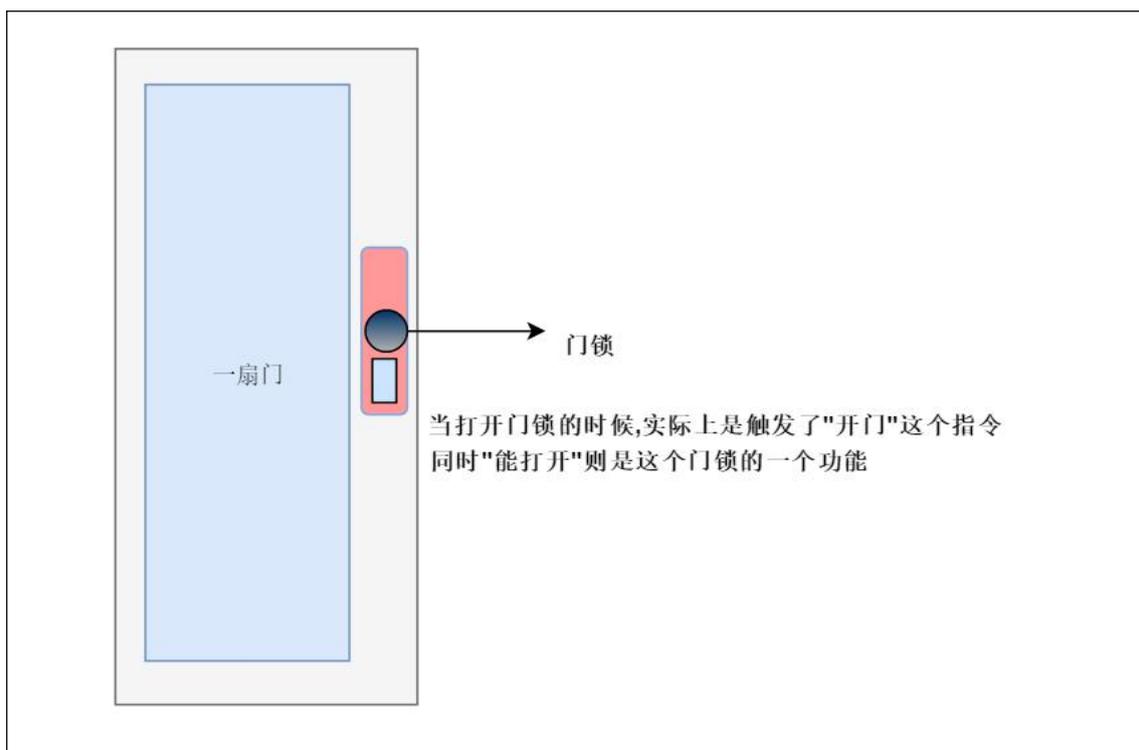
模块是挂载到设备上面的最小单独单元.想象一下,一个设备可能有最少一个以上的单元组成,比如共享单车,有车锁,太阳能充电板,车灯,这些基本的模块组成,模块是设备的最小组成成分.

一般在开发过程中,模块是一个传感器或者是开发板.如图所示:



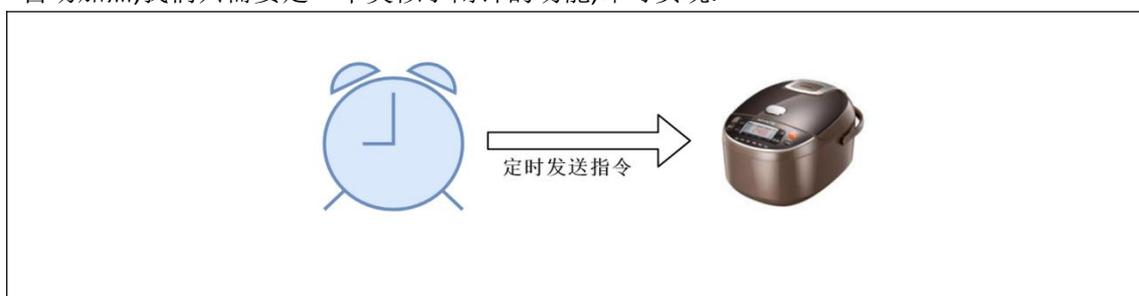
6.5 功能管理

功能是一个事件触发器,主要产生指令,发送给指定设备.比如“开锁”这个动作可以抽象出来,作为一个通用功能,发送的指令为“open”,然后我们每次想开锁的时候,只需要触发这个功能,系统就会发送这个指令到具体的设备.功能可以理解作为一种指令发送快捷方式.具体功能可参考下图模拟:



6.6 定时任务

定时任务主要用在设备上,目的是让用户设置一些定时触发的事件.例如电饭煲每天 6 点自动加热,我们只需要定一个类似于闹钟的功能,即可实现.

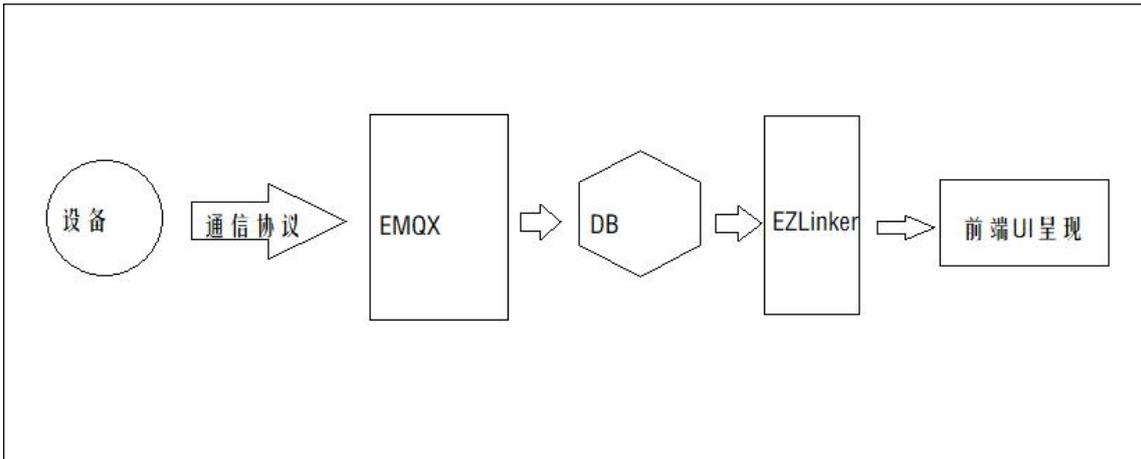


6.7 数据处理

数据处理分为 2 个方向:入站和出站.入站指的是设备数据采集,是从终端到数据库的过程.出站指的是从数据库过滤筛选展现的过程.

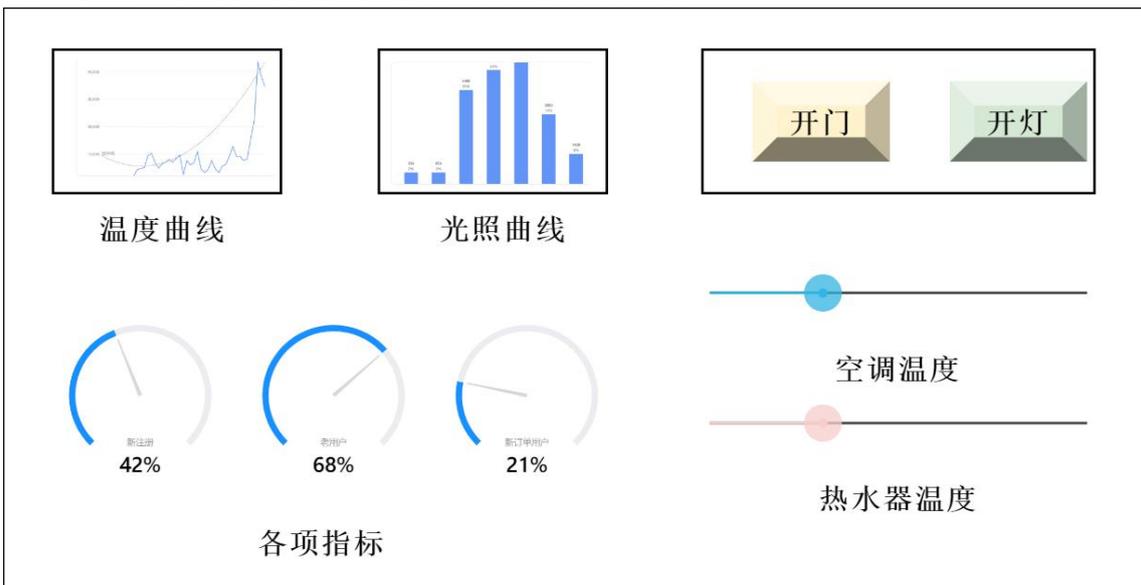
入站实现原理是直接通过 EMQ 把数据写入 MongoDB,不同的设备对应自己的一个数据库表(集合),数据全部入这个集合.

出站则是 EZLinker 进行筛选,通过后端处理,把数据进行和法国率和加工,最终呈现出来,前端的表现形式则是图表或者原始数据表格.处理过程如下图所示:



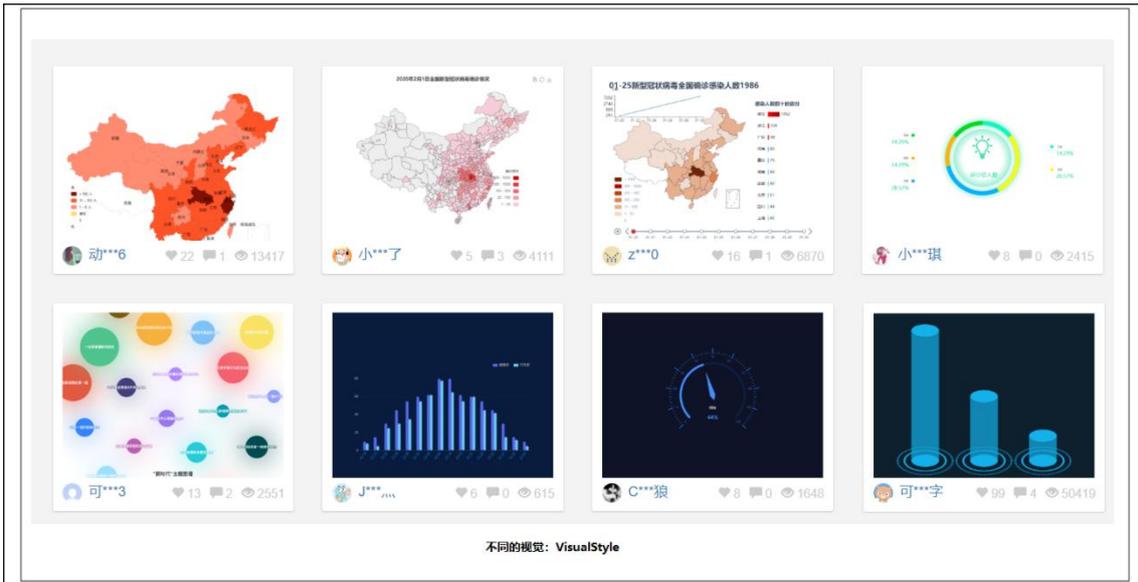
6.8 可视化面板

通常情况下数据展示是个大问题,因为要选择合适的 UI 元素去呈现,比图图表,或者进度条,仪表盘等等.我们需要考虑的问题比较多.为了制定一个自由的可视化布局界面,我们设计了一套用户可自由组合元素的设备控制面板.如下图所示:



其中可视化面板的组件分为 2 类,第一种是有动作的控件 (Component), 另一种是展示类视图 (View)。

其中视图 (View) 是所有可以可视化展示数据的模块的统称,而数据的展现形式称之为:视觉 (VisualStyle)。例如我们的温湿度传感器可以上传温度,湿度两种数据,但是温度和湿度有不同的展现形式,我们可以用柱状图,也可以用折线图,这些展现形式就叫:视觉 (VisualStyle)。视觉是作用于数据行上的一种效果。模块的视觉包含了模块、视觉、作用字段。



下表是目前支持的模块的类型和视觉类型:

模块类型

名称	标识	备注
按钮	BUTTON	
按钮组	BUTTON_GROUP	
开关	SWITCH	
开关组	SWITCH_GROUP	
进度条	PROGRESS	
数据体	DATA_ENTITY	
流媒体	STREAM	

视觉类型

名称	样式	备注
按钮	按钮	
按钮组	按钮组	
开关	开关	
开关组	开关组	
进度条	进度条	
数据体	表格: Table; 多样式	
流媒体	播放器	

当进入设计面板的时候,首先加载的是该设备的所有模块的视觉样式。有个对应关系:

设备	模块	样式
养鸡场监控中心	温度模块: DHT11 湿度模块: DHT11	数据体: 多曲线。温湿度传感器: DHT11, 具有多个指标, 此处可以选择数据曲线数据点。

	光照模块: LX-A	数据体: 单曲线
	视频监控: 海康威视-S0	播放器
	大门开关: SW2	开关
	灯光控制: Progress1	进度条

6.9 开放 API

开放 API 其实就是高级开发者模式.开发者模式的主要功能是允许通过申请的开发者调用系统接口.通常创建设备都是在系统里面操作界面进行创建,如果批量操作这样略嫌麻烦,开发者可以通过 API 批量创建,或者是批量获取数据等等,使用 API 构建自己的系统,相当于是开发者把系统的部分功能集成进自己的系统里面了,有了开发者功能,用户甚至可以基于 EZLinker 的 API 构建自己的扩展系统,而不研究 EZLinker 的代码.

开发者中心的目标是什么?

I. 简化开发

相当于是封装了基础业务平台,针对性的提取了物联网业务场景中的基础功能,使得公共部分抽象出来变成了云平台,所以第一个目标就是:**简化开发,减少重复劳动。**

II. 降低开发难度

可以假设这么一个场景: A 公司是一个做电商的公司,主要业务就是做商城系统,销售分发,公司的技术人员都是 PHP 或者 Java,主要 WEB 技术栈为主。但是某天领导为了顺应市场准备开发智能购物柜,这是一个物联网项目。要设计这套项目,公司必须需要架构师一

名, PCB 工程师一名, 电子工程师一名, 嵌入式工程师一名。这样的话会加重公司的人力成本和转型成本, 甚至会动摇公司的技术层的部门结构, 这样的代价是非常大的。而 EZLinker 可以解决这个问题, 因为 EZLinker 封装了物联网场景下的常见业务和协议接口标准, 所以此时就是开箱即用, 想想如果仅仅需要一个 PHP 程序员就能短时间内通过调用 REST 接口来实现一个物联网系统, 多少企业会选择使用这套方案。

III.促进转型

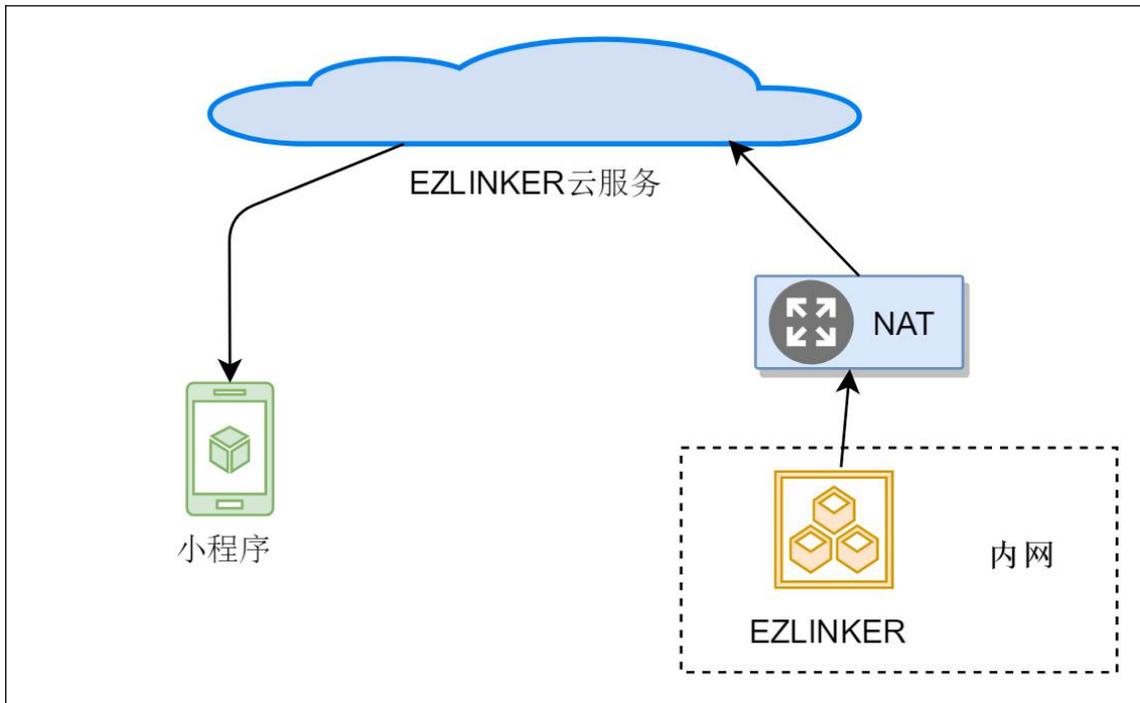
因为 EZLinker 设计完善, 所以一些初创企业和传统行业, 可以在不用改变现有的企业技术架构的情况下, 快速开发系统, 能在短时间内促进企业转型。

6.10 映射公网服务

因为 EZLinker 是针对内网用户打造的, 但是不排除用户有公网的需求. 针对这个特殊需求, 我们给 EZLinker 加了一套特殊功能接口: 通过回调接口, 把 EZLinker 注册到 EZLinker 团队部署在公网上的小程序服务器上, 用户就可以通过小程序来访问自己的内网服务。

我们通过一种特殊的 NAT 手法, 把内网映射出来, 然后登录 EZLinker 团队开发的微信小程序, 注册进服务以后, 就能互相通信. 此服务相当于是 EZLinker 团队提供的一种增值服务。

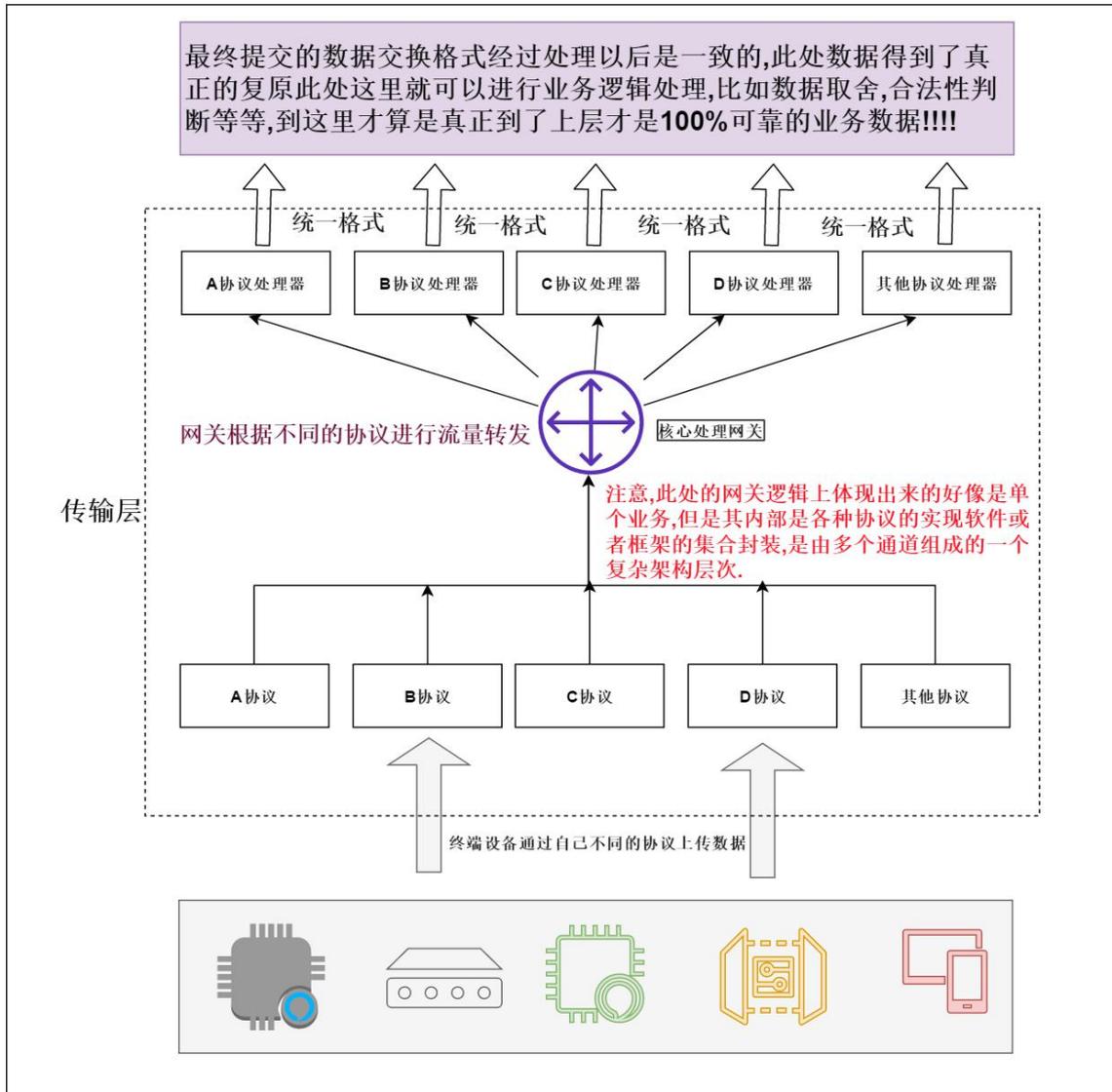
注意: 需要内网映射技术. 技术架构图如图所示:



7. 核心架构设计

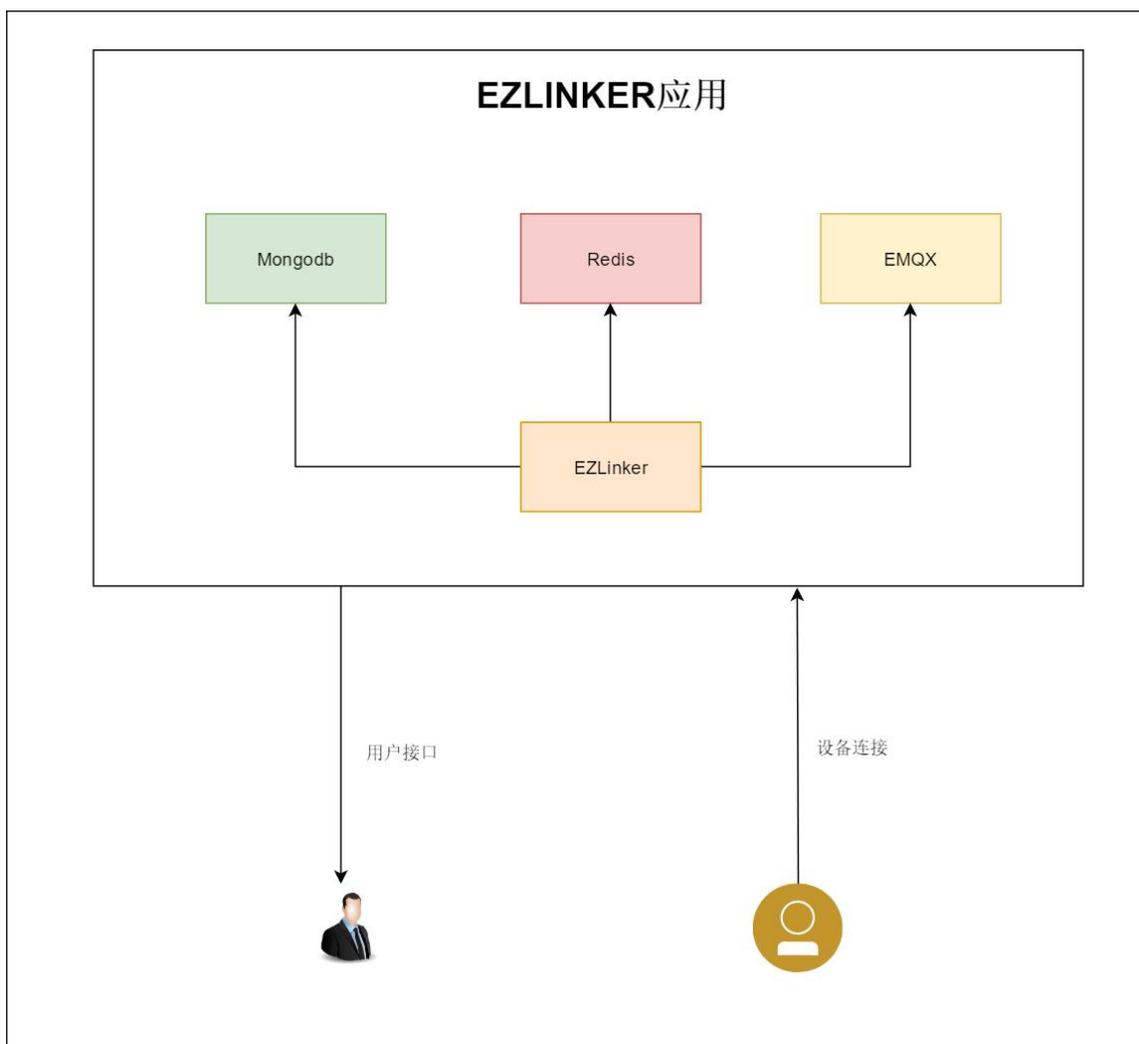
7.1 统一网关

网关是本项目的核心难点也是技术亮点.我们通过一个公用的协议处理机制,把不同的协议数据分析,解包成统一的数据格式,最终到达上层应用层,投入业务逻辑中,具体架构可以参考以下设计图.



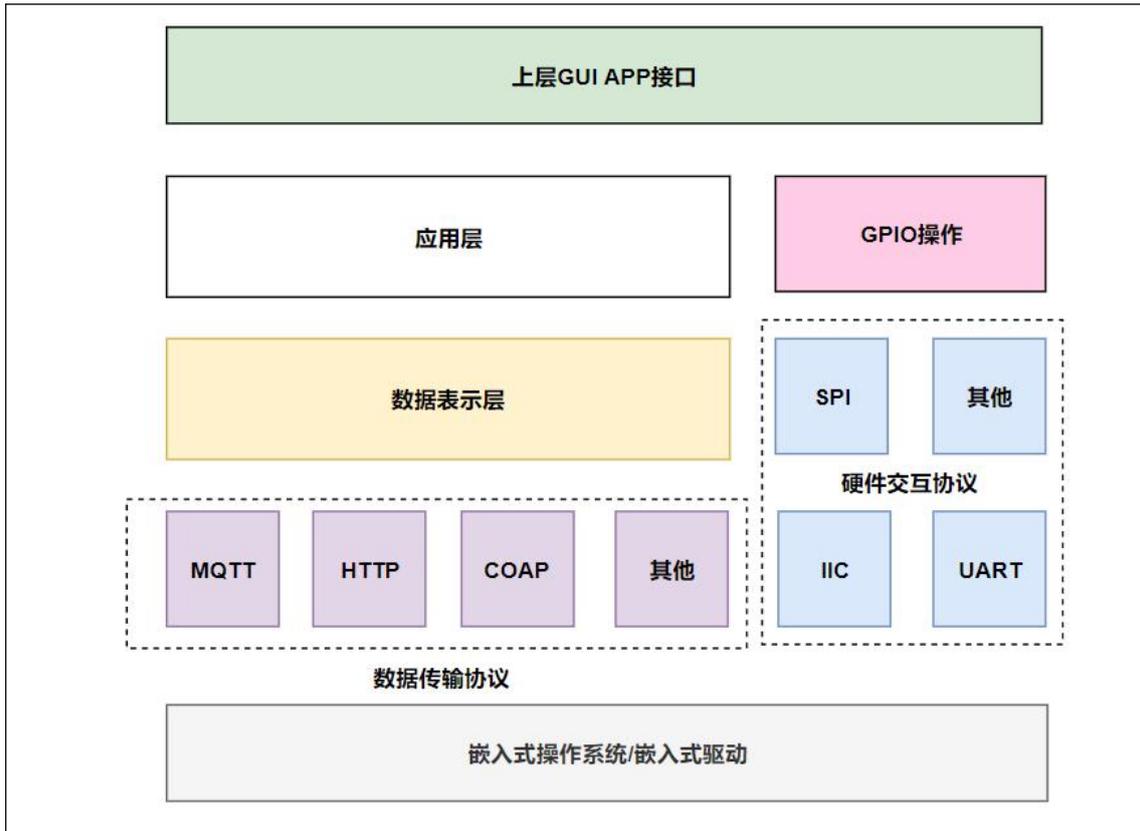
7.2 部署环境

下图是单机版部署,单机版预测设备数量能到 1000-10000 之间稳定运行,超过 10000 就要考虑使用多节点集群的形式.单机节点包含了 MongoDB, Redis, Mysql, EMQX, EZLINKER 这几个单体应用,在初期或者试验环境下可以架设,但是生产环境下建议最少保持 2 节点。

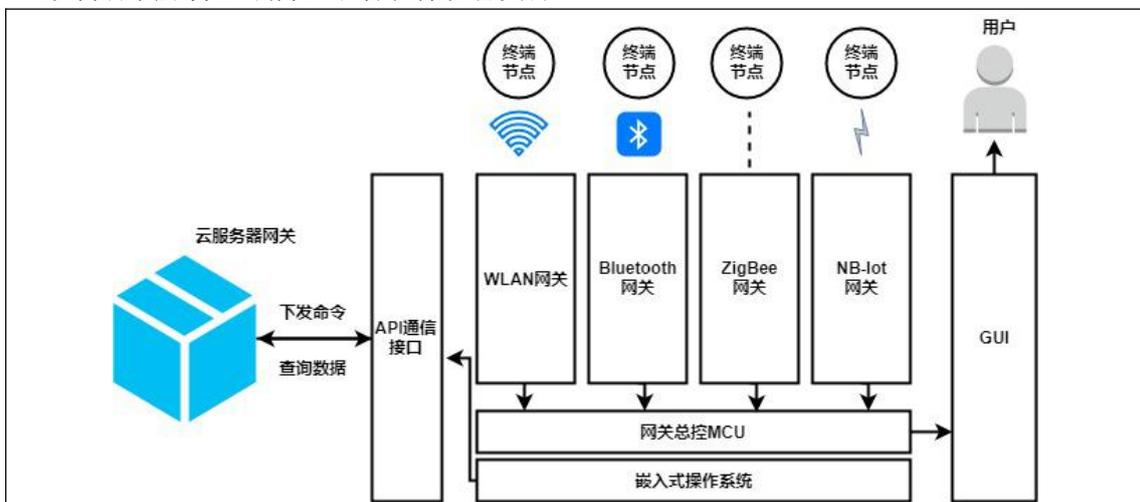


7.3 终端交互设计

未来通过嵌入式系统的开发和移植，实现多设备统一标准开发如图所示，以嵌入式系统为容器开发硬件接口实现外设驱动，内部通过移植 `mqtt` 等协议栈实现通信，对用户层暴露一定的硬件接口完成用户逻辑，尽量抽象化，使用户通过实现接口完成自己需要的业务代码。如有必要可以增加外设。



通过嵌入式操作系统结合多种通信模块实现网关对终端节点的管理,通过与云服务器通信实现网关节点命令下发、数据采集、状态查询等任务。在应对密集设备环境下通过此架构可以降低服务器载荷,节省硬件设备资源。



7.4 SDK 设计

SDK(Standard Develop Kit)是提供给外部程序对接的重要依赖库,尽可能的支持更多的编程语言和平台的支持.目前阶段准备先支持常见的 C\C++,Java,Python,Lua,Arduino 这几个常见的编程语言和开发框架。

下面来讲讲 SDK 的一些设计。

目前 SDK 主要是运行在终端设备上,工作模式为先配置,后工作,因此包含基本的公用属

性,函数,方法等等,下面是公用的部分细化设计(本 SDK 案例设计采用标准 C++作为模板进行讲解).

I.初始化

初始化是第一部要做的事情,一般包含了 SDK 的密钥配置和基本配置解读,所以最上层是一个大类:EZLinkerSDK.其中包含了初始化的一个类方法:init(String token,String secret),表示全局注册,这一步 C++代码如下:

```
# include("ezlinker.h")
int main()
{
EZLinkerSDK * sdk = EZLinkerSDK::init("token","密钥");
return 0;
}
```

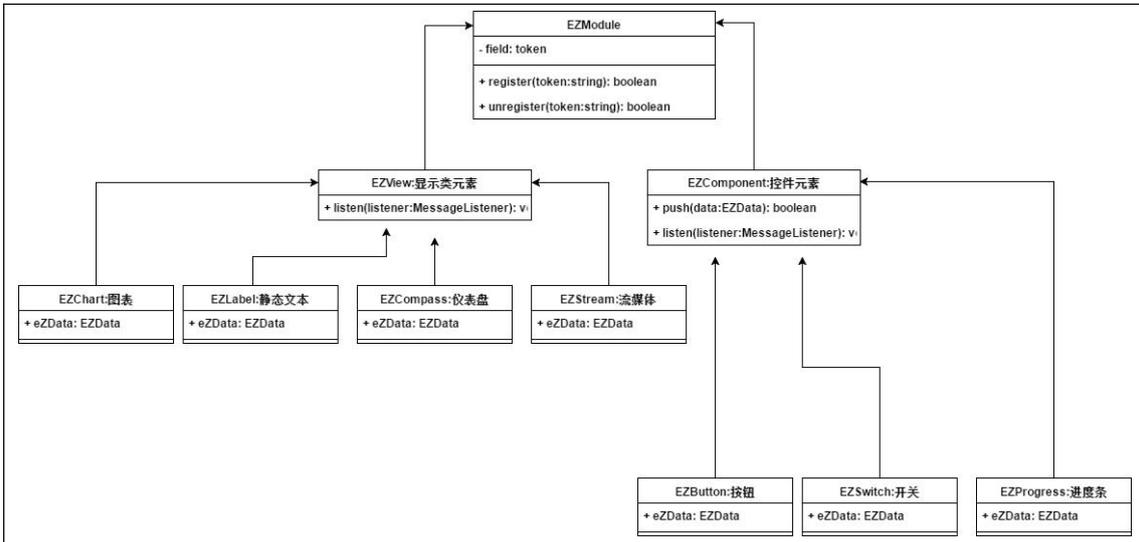
以上则构建一个 SDK 实例,采用了单例模式来实现,所以只能实例化一次.

II.注册组件

组件是模块对应的 UI 界面元素,比如 Button,ProgressBar 等等常见界面组件.同样的,我们在 SDK 端也做了和服务端一样的统一.

组件设计起来比较复杂,但是使用起来简单,接下来我们首先讲一下实现原则.

整体上采用了 OOP 的设计思想,所有的组件来源于同一个基类:EZModule,派生出各种类型的组件,类继承结构图如下所示.



组件包含注册,事件响应,数据推送的功能,理解清楚了类继承图以后,就可以很容易实现一套从设备到 UI 的业务.

UI 具体表现出来同步的控制和动作:



III.上传数据

本操作,我们以一个按钮响应事件作为案例来演示:

```
/**
 * 需求:XXX 单片机,需要相应来自 UI 上的一个按钮的事件,来实现开关门
 **/
# include("ezlinker.h")
int main(){
EZLinkerSDK * sdk = EZLinkerSDK::init("token","密钥");
EZButton & btn = sdk->registerButton("Token");
btn->addListener(new ButtonListener(){
virtual EZState onData(EZData& data){
// Data 为服务器端发来的数据
// 处理数据业务在这里进行
// 处理完了以后必须返回状态
// 如果返回空,或者返回的状态不正确
// UI 显示为上一次最后的正常状态
return new EZState("1","2");
}
});
return 0;
}
```

假如说是一个显示组件呢?我们只需要上传数据即可,不用监听数据.下面我们看看如何上传温湿度数据,然后 UI 显示为曲线

```
/**
 * 需求:显示温湿度曲线
 **/
# include("ezlinker.h")
int main(){
EZLinkerSDK * sdk = EZLinkerSDK::init("token","密钥");
EZChart * chart= sdk->registerChart("Token");
chart->push(new EZDate(100,35));//构造方法传入数据
return 0;
}
```

对于前端而言,只需要轮询或者 WS 监控数据变化即可快速响应变化.

8. 应用市场

APP 市场是一个大后期的设想:EZLINKER 有了大部分业务场景下的实践案例,然后把这些案例进行统一整合,投入一个 APP 市场进行管理,用户可以上传自己的项目和部署市场里面的项目.设计效果如下图所示.



市场的基本功能：官方可以发布一些镜像上去，用户直接部署使用，这种是自营市场。另外一种是用用户上传，基于 EZLinker 的一些物联网项目做成 Docker 容器的形式，提交给 EZLinker 云，审核以后可以作为市场的 APP 运营。