振动边缘场景方案设计

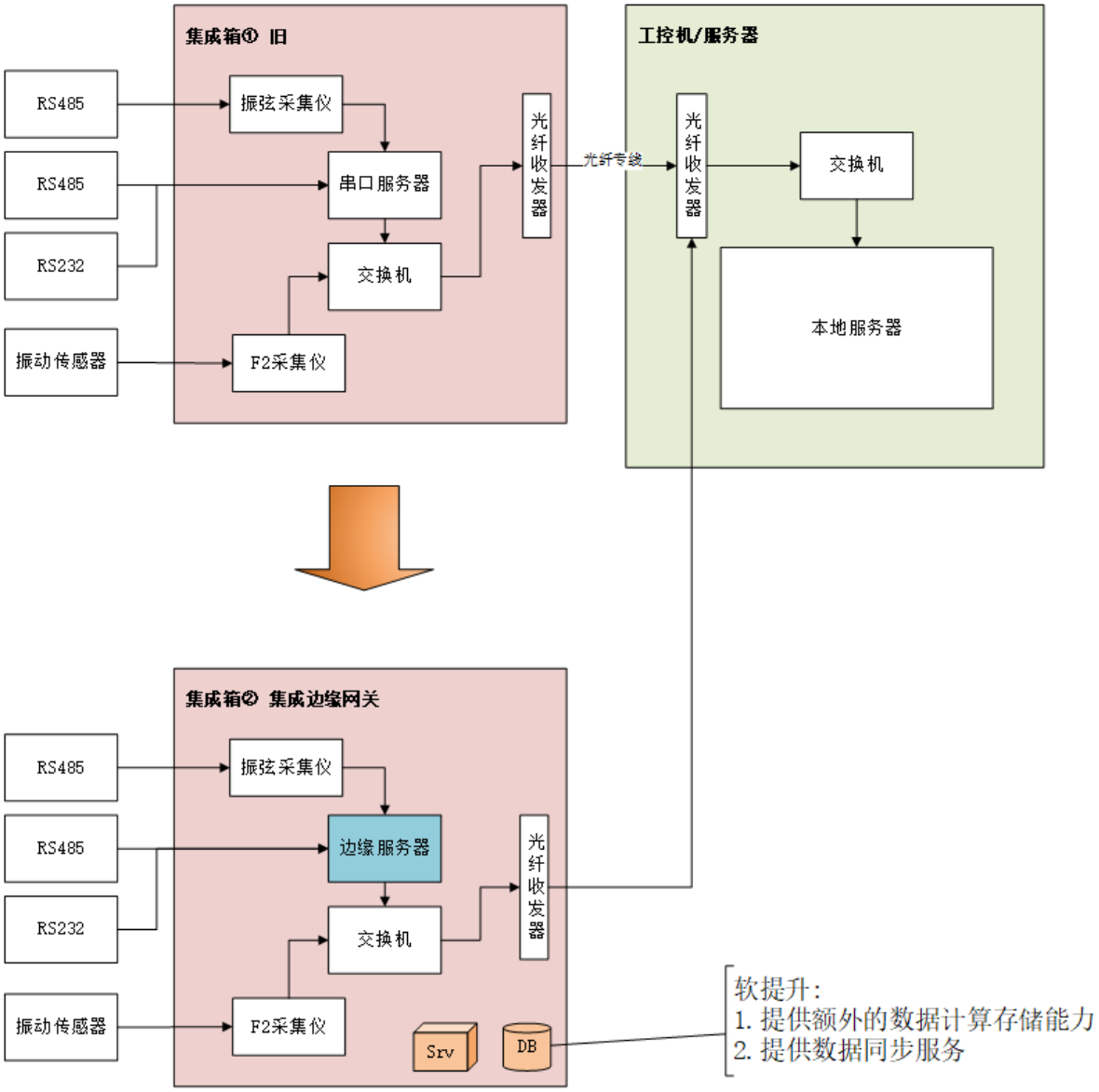
边缘场景

在结构物安全监测，包含物振动监测的场景中，将边缘网关将位于现场组网端（最接近“端”的位置）（一般会集成在采集箱内），通过边缘系统实现数据采集、计算和存储：

边缘场景

在结构物安全监测，包含**振动**监测的场景中，边缘网关将位于现场组网端（最接近“端”的位置）（一般会集成在采集箱内），通过边缘系统实现数据采集、计算和存储：

> 对现场组网的理解可能存在偏差，大概示意如图



可以看出，原来现场的配电箱内除了电源控制外，剩下的是采集控制（采集仪）和传输控制器（包含交换机、光纤收发器等），统一可以看作是对**信号量的调制过程**，即模拟信号->数字信号->光信号。

如果对应人体，就相当于一套**神经传输系统。**

边缘概念中，我们通常把边缘系统比做**章鱼。**章鱼有**40%**神经元在脑袋里,剩下的**60%**在它的8条腿上,所谓的**用“腿”思考。**

如集成箱②中描述，其中边缘服务器就类似这一区域内传感系统的“**大脑**”。

边缘计算的基本思想则是**功能缓存(function cache)**，是大脑功能的延生。 边缘计算是云计算的延生和补充。

边缘的大脑负责：

- 收集和转发数据 (**信息传导**)
- 数据存储（**记忆功能**）
- 分析和反馈（**思考功能**）

这样的优势显而易见：

👍 分布式和低延时计算

👍 效率更高、更加智能（AI）、更加节能

👍 缓解流量压力、云端存储压力

👍 隐私保护，更加安全

振动边缘设计

拟在`linux`嵌入式板上实现`golang`开发的边缘服务，实现对振动F2采集仪的数据**采集控制、计算、存储和传输**功能。

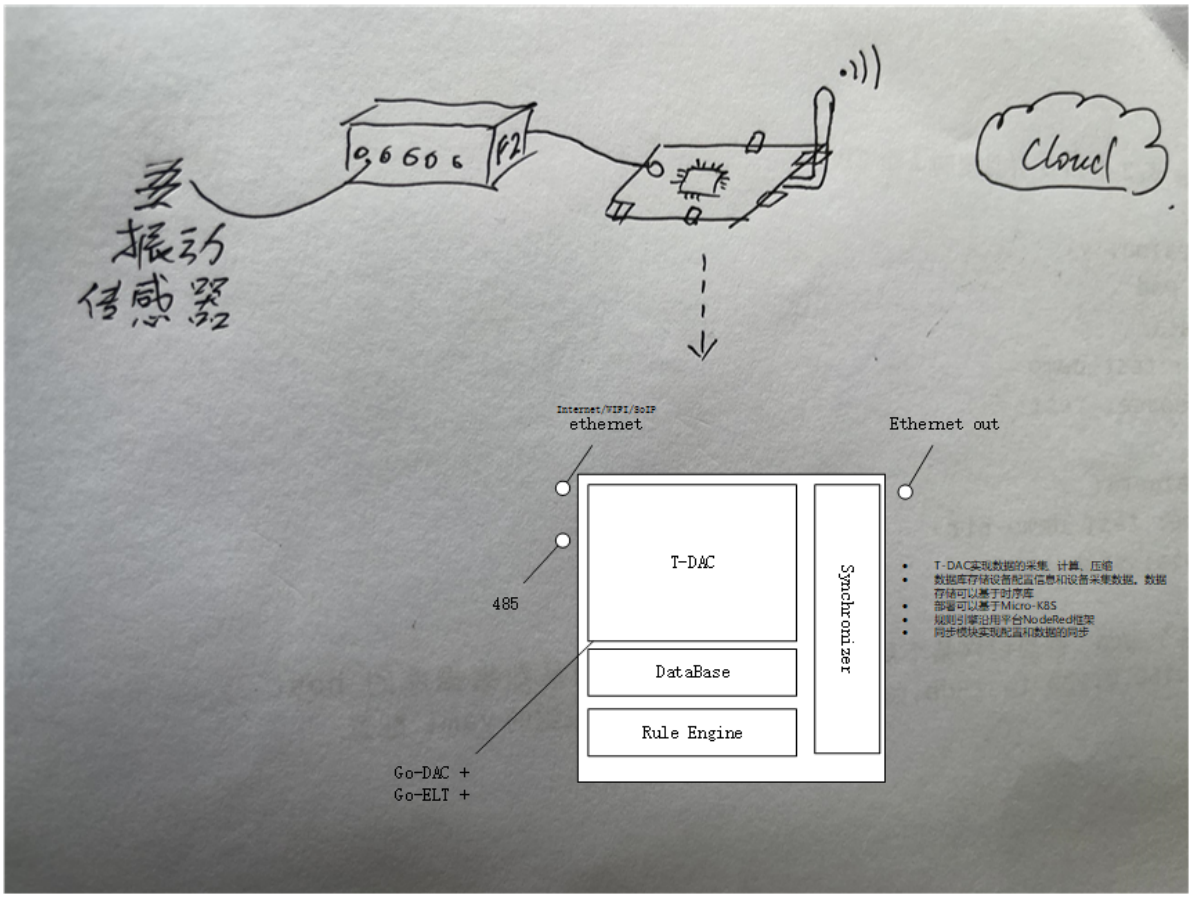
技术选型

硬件：跟硬件同事讨论后，选择了飞凌公司的OK1028A开发版，配置如下

软件：编程语言选择了目前以太DAC一致的Golang，开发Linux环境程序。

部署：考虑使用`MicroK8S`在板子上部署采集程序和其他服务（数据库/消息组件）

整体设计



如上，框架在《边缘网关的一些思考》中已经初步介绍，我们在开发板中，至少需要集成：

- 数据采集和处理程序(T-DAC)，这里主要是Go-DAAS负责采集振动数据
- 配置同步控制程序
- 数据库服务
- 规则引擎服务

需要安装的服务有:

数据库InfluxDB

消息中间件mosquitto

规则引起NodeRed

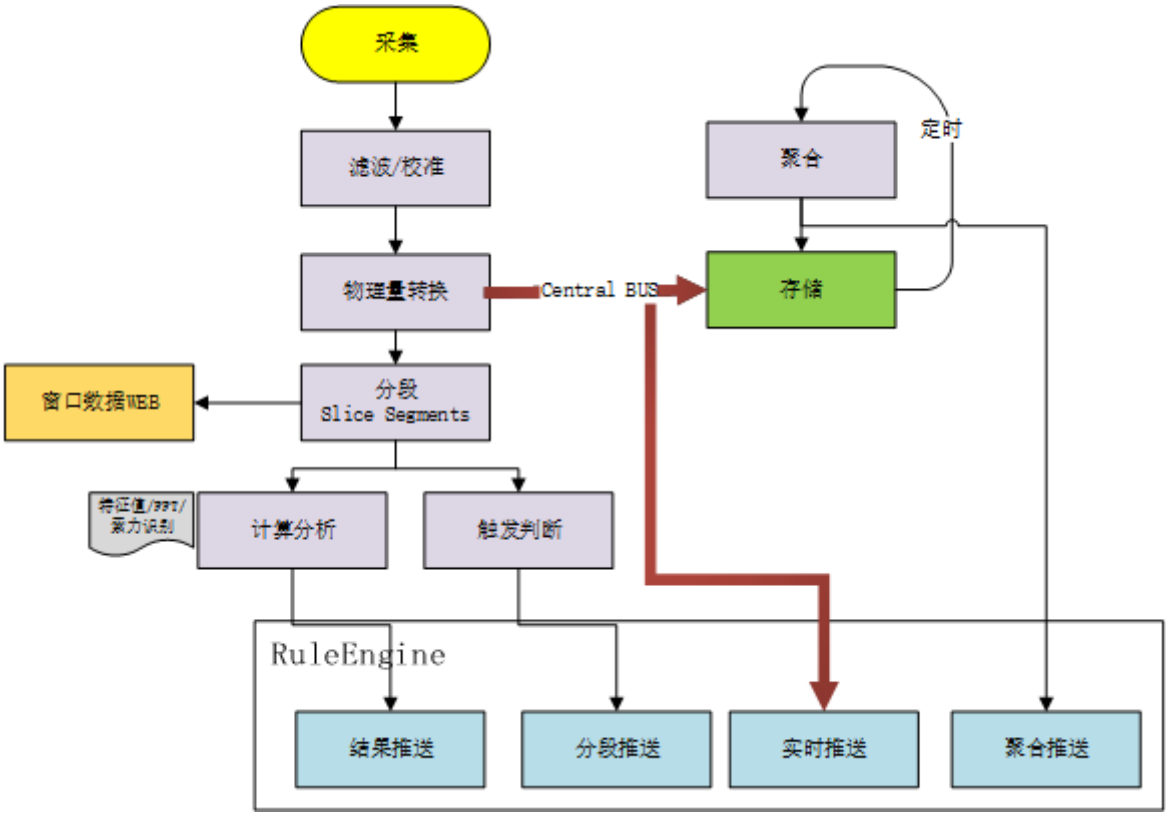
应用go-DAAS

应用go-DAC

数据流程

数据从采集开始，经过**ET**过程(图中滤波、校准、物理量计算)之后， 主要分三个主流向：

1. 存储到数据库
2. 计算分析后存储和上报平台
3. 自动聚合(压缩)后的数据上报平台



具体模块说明：

1. 滤波/校准

   校准去直流、滤波算法

2. 物理量计算

   输出电压值到监测物理量值转换

3. 分段

   设置窗口大小、刷新时间，将数据进行分段

4. 窗口数据WEB

   实现http服务提供实时窗口振动数据(类似DAAS实时采集展示)。并通过wobsocket实现实时更新

5. 计算分析

   通过设置的算法，实现特征值(TMRS)、FFT、索力识别对计算

6. 触发判断

   通过设置的触发条件（定时/信号量），对连续信号进行采样保存 （同DAAS采样生成.odb数据文件）

7. 存储

   数据存储到边缘数据库。

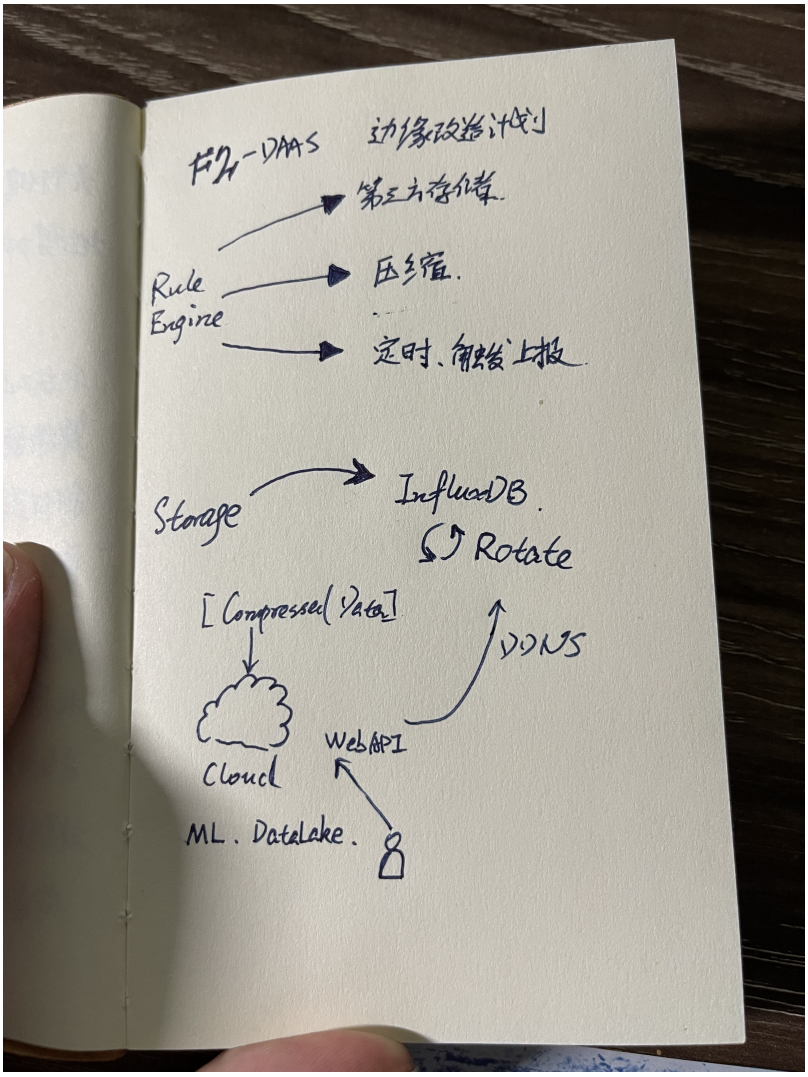
8. 聚合

   数据库中定时生成1s/10s等统计数据

9. 推送

   通过规则引擎实现

关于边缘网关**提供数据**的形式，初步思考：可以通过DDNS或NAT内网穿透，通过端的WEB-API服务向外提供。



采集模块

采集模块的设计稿如下，主要是设备连接控制的过程 。

Session：管理TCP连接会话

VbServer：设备控制主服务，包括管理连接、配置读取、配置设置、启动采集和数据回调

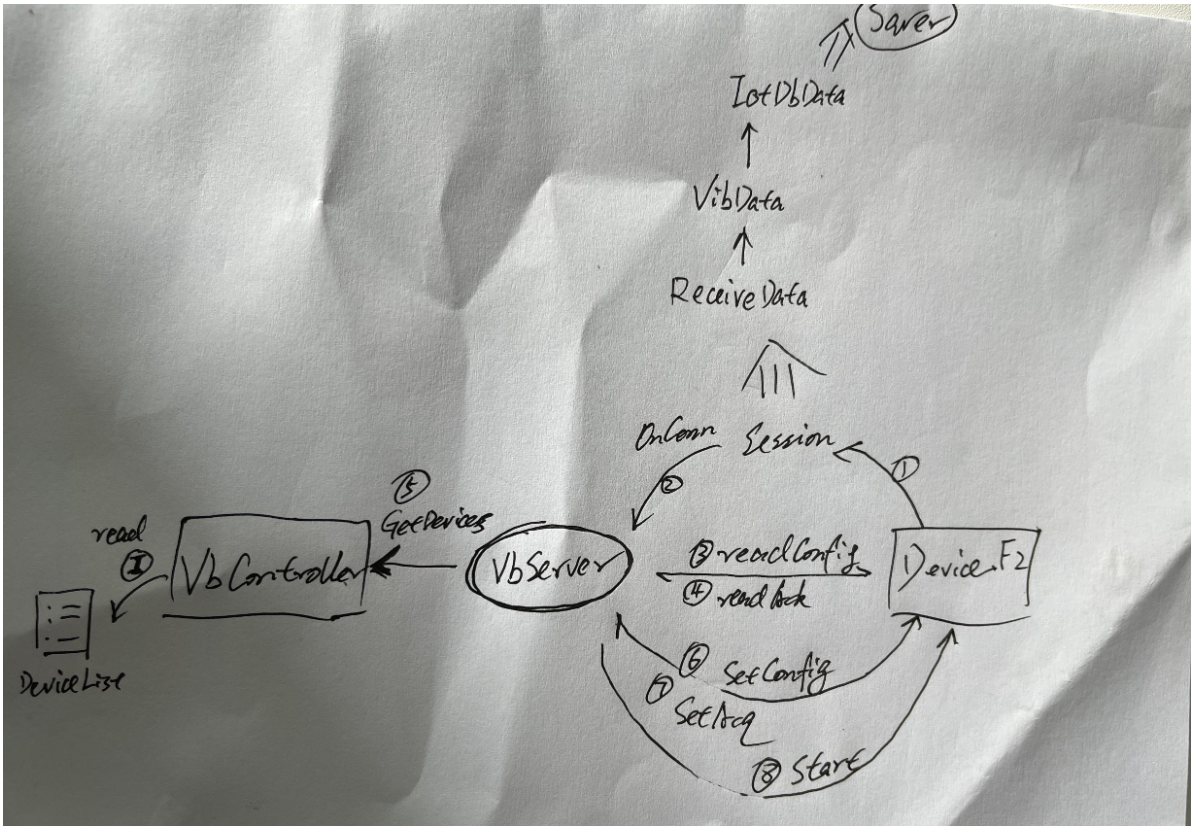
VbController: 负责设备配置和处理流程控制

数据模型：

ReceiveData: 参考C#DAAS，Session上的原始信号数据

VibData：转换后的振动数据

IotDbData: 待入库格式数据



存储设计

在 《边缘数据库选型》中对比了几种数据库，初步拟定使用`InfluxDB`作为边缘端存储引擎。

目前仅考虑振动数据的存储。存储到数据格式如下：

```
bucket : data
measurement: vib
tags: id (设备id)
fields: phy (物理量值)
```

通过配置持续聚集（CQ）实现数据的自动聚合

```
create database data_10sec_agg;

-- 数据库 ${db}

-- 10s持续聚集 每10s执行 (FOR 1min)允许数据晚到1min内
CREATE CONTINUOUS QUERY "cq_ten_sec" ON "vib"
RESAMPLE EVERY 10s FOR 1m
BEGIN
  SELECT mean(*),max(*),min(*),spread(*),stddev(*) INTO
"data_10sec_agg"."autogen".:MEASUREMENT FROM /.*/ GROUP BY time(10s),*
END;
```

同步系统

TODO

推送系统

TODO

系统验证

MicroK8S安装出错

```
开发版型号：FET3399-C核心板
系统：ForlinxDesktop 18.04
内核：Linux node37 4.15.0-136-generic #140-Ubuntu SMP Thu Jan 28 05:20:47 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux

按照官方安装MicroK8S步骤：
1. apt update
2. apt install snapd
3. snap install microk8s --classic --channel=1.21/edge

报错：
error: system does not fully support snapd: cannot mount squashfs image using
       "squashfs": mount: /tmp/sanity-mountpoint-058597126: wrong fs type, bad
       option, bad superblock on /dev/loop0, missing codepage or helper
       program, or other error.
```

裸运行程序

问题1：无法通过网络远程

```
通过串口连接后查看ip配置：
ifconfig:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.8.30.195  netmask 255.255.255.0  broadcast 10.8.30.255
        inet6 fe80::dd83:a430:5e3a:d947  prefixlen 64  scopeid 0x20<link>
        ether b6:a8:21:1d:72:0b  txqueuelen 1000  (Ethernet)
        RX packets 46157  bytes 19908170 (19.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31199  bytes 22566677 (22.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 24

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 42657  bytes 145116329 (145.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 42657  bytes 145116329 (145.1 MB)
```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

需要ping一下外面的机器（10.8.30.117），然后117上才能访问它。

安装influxdb2. https://portal.influxdata.com/downloads/

```
wget https://dl.influxdata.com/influxdb/releases/influxdb2-2.0.9-arm64.deb
sudo dpkg -i influxdb2-2.0.9-arm64.deb

# start
influxd
```

安装`mosquitto`

```
apt install mosquitto
```

安装Node-Red

```

```

编译在arm上运行的edge程序：

```
export PATH=$PATH:/usr/local/go/bin
export GOPROXY=https://goproxy.io
export GOPATH=${GOPATH}:"`pwd`"

CGO_ENABLED=0 GOOS=linux GOARCH=arm64 go build -ldflags "-extldflags -static " -
ldflags "-X main.VERSION=1.0.0 -X 'main.SVN_REVISION=$SVN_REVISION_1' -X
'main.BUILD_NUMBER=$BUILD_NUMBER' -X 'main.BUILD_TIME=$BUILD_TIMESTAMP' -X
'main.GO_VERSION=`go version`'" -tags netgo -a -v -o ../../$BUILD_NUMBER/edge
```

在本机构建：

启动WSL安装的ubuntu

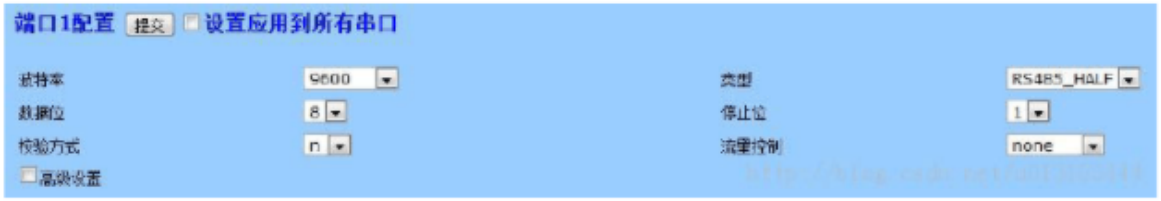
```
wget https://studygolang.com/dl/golang/go1.16.4.linux-amd64.tar.gz
sudo rm -rf /usr/local/go
sudo tar -C /usr/local -xzf go1.16.4.linux-amd64.tar.gz
sudo sh -c "echo 'export PATH=\$PATH:/usr/local/go/bin'>> /etc/profile"
source /etc/profile

go version

go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.io,direct
#安装 gcc arm的交叉编译工具
sudo apt-get install -y gcc-aarch64-linux-gnu
aarch64-linux-gnu-gcc -v

yww@DESKTOP-5R6FOH1:/mnt/e/Iota/trunk/code/gowork/src/edge$ CGO_ENABLED=1 \
CC=aarch64-linux-gnu-gcc \
GOOS=linux \
GOARCH=arm64 \
```

```
go build -ldflags '-s -w --extldflags "-static -fpic"' -o ./edge
```

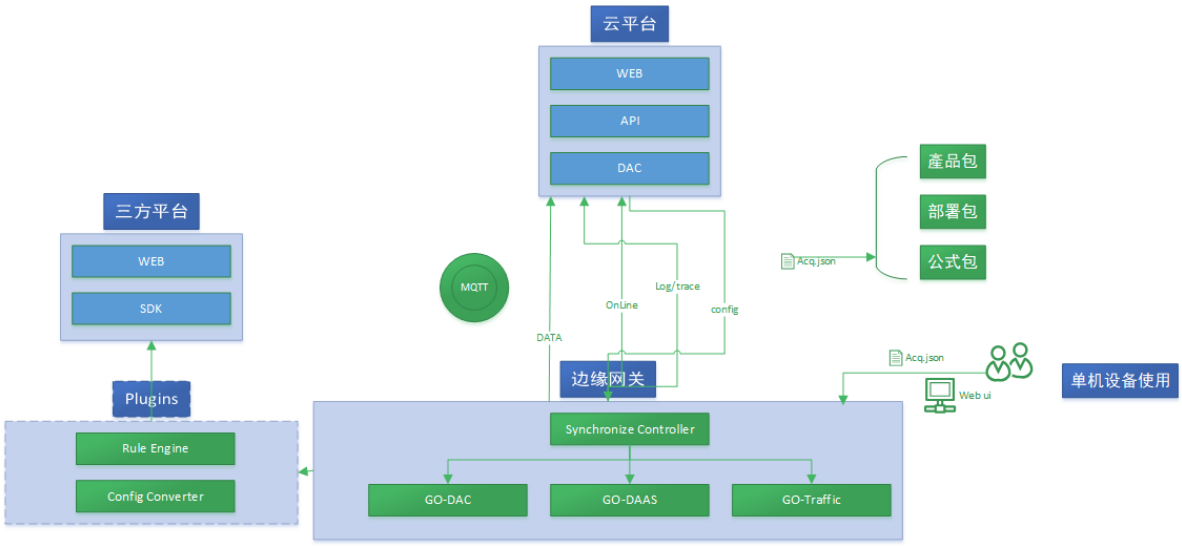
串口服务器

开发linux上的串口服务器 虚拟串口VSPM软件。

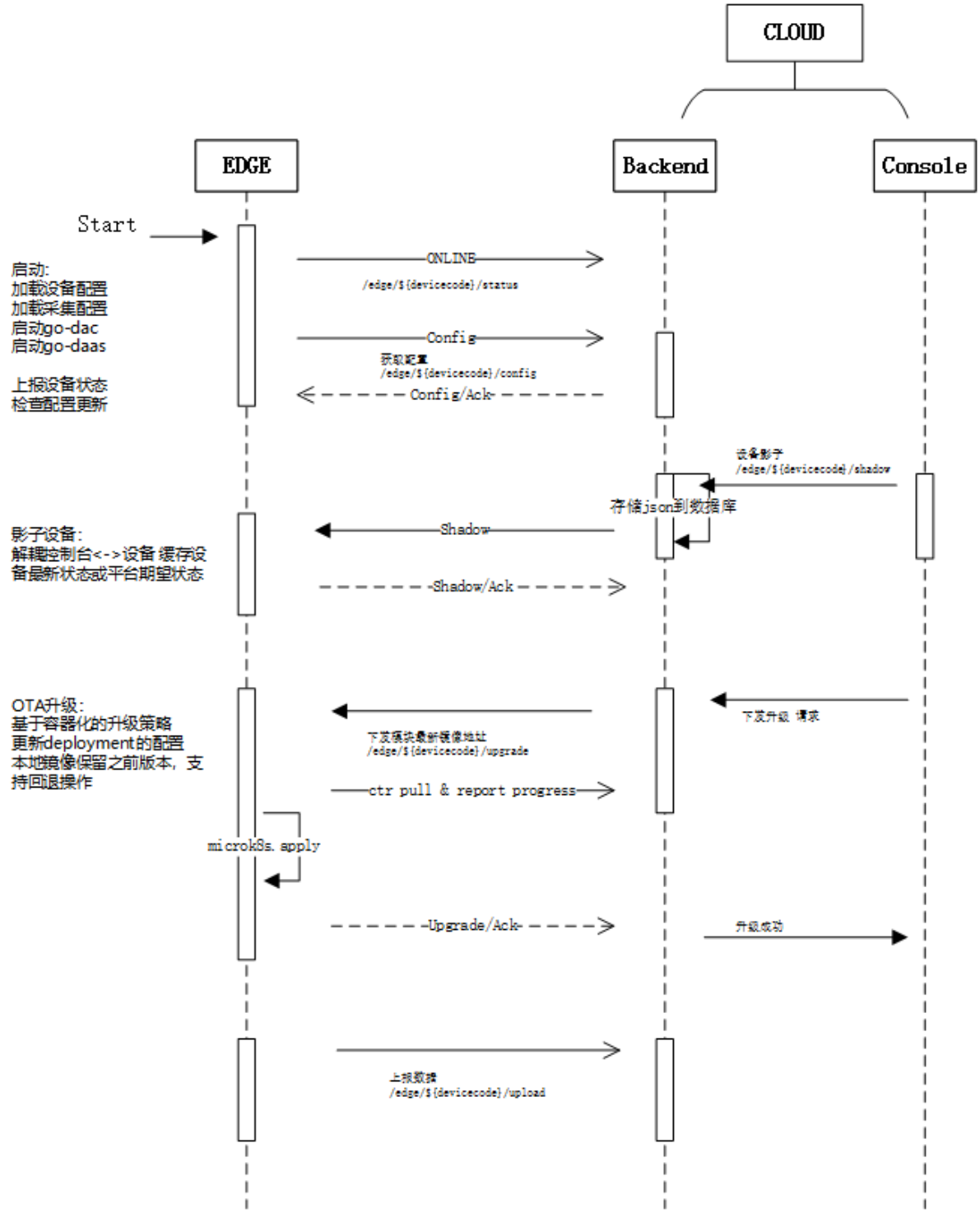
串口服务器将串口转为Tcp连接，这里设定串口服务器作为TCP客户端。本工具是在PC机（边缘板）上启动tcp服务，将连接转到serial-port。



云边协同设计



工作序列



Ali： https://help.aliyun.com/document_detail/73731.html?spm=5176.11485173.help.7.380b59afFTM5xR#section-qej-6sd-o53

原則

1. 配置至上而下
   平臺執行下發指令。 無外網情況：平臺到處json
   項目復用，MEC生產過程：
   平臺生產連續json配置，下發or導出。
2. 數據至下而上
   如字面意思.
3. 兼容第三方平臺？？？
   第三方平臺配置協議 very difficute!!

數據通過規則引擎轉發、任意格式

下發配置包
包含：

> 產品包
> 協議、設備型號、能力、接口

> 部署包
> 鏈接關係、采集參數設置

> 公式包
> 公式選取和參數設置

邊緣特許場景：
1、問題排查
問題日志上傳。
工作日志上傳（跟蹤長期狀態）
Can switch to off of cause
2、遠程升級
看其他平臺如何實現的?

协议选型

需要如下功能：

1. 配置下发
2. 心跳
3. 数据
4. 诊断
5. 固件OTA升级

选择iDAU MOP协议，通讯方式仅考虑MQTT(暂不考虑SoIP)，进行改造。

参考《整体方案-软件.docx》

```
// 对象管理协议
// thing 下发
// /edge/thing/uuid
{
    "M":"thing",
    "O":"set",
    "P":{
        ...
    }
}

// 心跳
    ANY
// A
{

}
// B
{

}
```

```
// C
{

}
// D
{

}
// E
{

}
```

DAC中後處理接入位置:
`src\iota\scheme\driverBase.go`

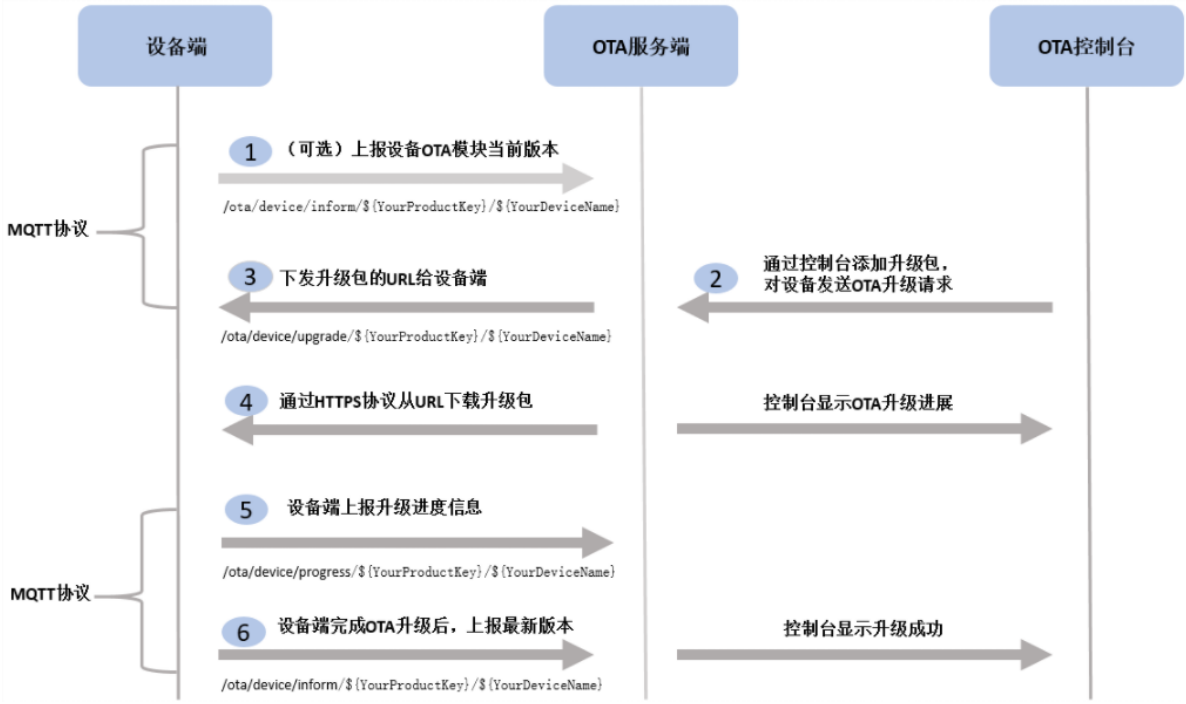
```
SELECT id, name, "desc", version, impl, resource, "resType",
"enableTime","updatedAt" FROM "ProtocolMeta"
         WHERE ("enableTime"<=$1 or "enableTime" is null)


 OnProtocolChanged --> updateProtocol  支持雲上協議動態應用。



 “DeviceMeta”
 // cacheDeviceMeta 缓冲设备元型
 SELECT id, name, model, "desc", "belongTo", category from "DeviceMeta"

 SELECT id, name, "desc", "deviceMetaId", "protocolMetaId" from "CapabilityMeta"
 where "deviceMetaId" in (%s)

 SELECT
cp.name,cp."showName",cp.category,cp.enum,cp."defaultValue",cp."min",cp."max",cp
."unit",cp."capabilityMetaId",cp."propertyTypeId"
        FROM "CapabilityProperty" cp %s WHERE cp."capabilityMetaId" in
             (select id from "CapabilityMeta"  where "deviceMetaId" in (%s))


獲取THING
GetThingsByIds》
SELECT id,name,"desc","belongTo",enable, release FROM "Thing" WHERE TRUE AND
enable=true

getThinkLinksByID》

SELECT a.*, dstdi."deviceId", dc.id, dc."protocolMetaId" dcpmid,
cm."protocolMetaId" cmpmid from (
```

```
        SELECT t."belongTo", layout."thingId", lnk.id, lnk."parentLinkId",
                lnk."fromDeviceInterfaceId", it.key, im.name, di.properties,
                lnk."toDeviceInterfaceId", di."deviceId", dmi."deviceMetaId"
            FROM "LayoutLink" lnk, "LayoutNode" node ,"Layout" layout,
                "DeviceInterface" di, "DeviceMetaInterface" dmi, "InterfaceMeta"
im , "InterfaceType" it, "Thing" t
            WHERE
                layout."thingId" = t.id
                AND lnk."fromDeviceInterfaceId" = di.id AND
di."deviceMetaInterfaceId" = dmi.id
                AND dmi."interfaceMetaId" = im.id AND im."interfaceTypeId"=it.id
                AND lnk."fromLayoutNodeId" = node.id AND node."layoutId" =
layout.id %s
        ) a
    LEFT JOIN "DeviceInterface" dstdi ON dstdi."id" = a."toDeviceInterfaceId"
    LEFT JOIN "DeviceCapability" dc on dc."deviceInterfaceId" =
a."toDeviceInterfaceId"
    LEFT JOIN "CapabilityMeta" cm on cm.id = dc."capabilityMetaId

    AND t.enable=true

    AND layout."thingId" = '%s'

getDimsOfThings》
SELECT id,name,"desc", "thingId" FROM "Dimension" where "thingId" in (%s) order
by id

getSchemes>>
SELECT s.id,d."id" did, s.name, s.unit,s.mode,
s.interval,s.repeats,s."notifyMode",s."capabilityNotifyMode",
s."beginTime",s."endTime"
FROM "Scheme" s, "Dimension" d
WHERE s."dimensionId" = d.id and d."thingId" in (%s)

getDimCaps>>
fields := `
        dc.id, dc."dimensionId", dc.repeats,dc.interval,dc.qos, dc.timeout,
dc."errTimesToDrop",` + //DimensionCapability
        `cm.id, cm.name, ` + // CapabilityMeta=> Protocol/Interface/Device
        `c."id", c."protocolMetaId", c."capabilityMetaId", c."properties", ` +
// DeviceCapability
        `d.id, d.name, d.properties, dm.id, dm.name,dm.category,` +
//Device+DeviceMeta
        `di.id, di.properties, im.id, im.name, im."interfaceTypeId", ` + //
DeviceInterface +InterfaceMeta
        `cm."protocolMetaId"`  //DeviceProtocol  +ProtocolMeta
    relation := `
        "Dimension" dim, ` +
        `"DimensionCapability" dc, ` + // 主表
        `"DeviceCapability" c, ` + // 使用的能力
        `"Device" d, ` + // 设备
        `"DeviceMeta" dm, ` + // 设备元型
        `"DeviceInterface" di,` + // 设备接口
        `"DeviceMetaInterface" dmi,` + // 设备元型接口
        `"InterfaceMeta" im, ` + // 接口对应的接口元型
        `"CapabilityMeta" cm ` // 能力元型
    // ""ProtocolMeta" pm" // 协议元型 (协议信息)
    where := `
```

```
        dc."deviceCapabilityId" = c.id ` + // 主引用: 能力元型
        `AND d.id = c."deviceId" ` + // > 设备
        `AND di.id = c."deviceInterfaceId" ` + // > 能力使用的接口
        `AND cm.id = c."capabilityMetaId" ` + // > 能力使用的能力元型
        `AND dm.id = d."deviceMetaId" ` + // Device => DeviceMeta : 设备-> 设备元型.
        `AND dmi.id = di."deviceMetaInterfaceId" ` + // DeviceInterface => DeviceMetaInterface
        `AND im.id = dmi."interfaceMetaId" ` + // DeviceMetaInterface => InterfaceMeta : 接口-> 接口元型.
    // "AND pm.id = cm."protocolMetaId" AND ` + // DeviceProtocol => ProtocolMeta     : 协议-> 协议元型.
        fmt.Sprintf(` AND dc."dimensionId"=dim.id AND dim."thingId" in (%s) `, joinStr(thingIds)) // 限定 Thing

scanRowAsDimCap>>
    ds.getSubDevices(dc.Device)  (获得网关设备的子设备，并在子设备里添加链接实例)
    ds.GetFormula(dc.Capability.ID)
            SELECT id, "properties", "formulaId" FROM "CapabilityFormula" WHERE "capabilityId"=$1
    ds.GetProtocol(dc, dc.Capability.PMID) // from cache
```

OTA升级

参考阿里云设计：https://help.aliyun.com/document_detail/85700.html



- OTA deployment operator security. 操作权限足够安全
- Incremental roll-out of OTA updates. 作增量升级
- Securely downloading the update. 建立安全的下载通道

Nginx ON Windows

windows上启动iota进行调试：

download http://nginx.org/en/docs/windows.html

```
cd /d E:\WorkSpace\nginx-1.13.10
.\nginx.exe
```

`nginx.conf`

```
server {
    listen 80;
    server_name 10.8.30.38;
        ssl off;
        client_max_body_size 10M;

    location /v1/api {
        proxy_pass  http://10.8.30.38:19090/v1/api;
        proxy_cookie_domain localhost localhost;
        proxy_cookie_path / /;
        proxy_connect_timeout 300;
        proxy_send_timeout 1200;
        proxy_read_timeout 3000;
        proxy_set_header X-Real-Ip $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location / {
        proxy_pass    http://10.8.30.38:9080;
        proxy_cookie_domain localhost localhost;
        proxy_cookie_path / /;
        proxy_connect_timeout 100;
        proxy_send_timeout 6000;
        proxy_read_timeout 600;
    }
}
```

nginx -s stop	**fast shutdown**
nginx -s quit	graceful shutdown
nginx -s reload	changing configuration, starting new worker processes with a new configuration, graceful shutdown of old worker processes
nginx -s reopen	re-opening log files

通过docker启动

```
docker run -d --name nginx-server -p 8080:80 -p 443:443 -v
E:\WorkSpace\nginx:/etc/nginx/conf.d:rw nginx
```

win10上安装micro-k8s

https://ubuntu.com/tutorials/install-microk8s-on-windows#2-installation