haojia xie(谢浩佳)的主页

关于我 (haojia xie(谢浩佳))

你可以编辑本页面来介绍自己。

电子邮件: haojia.xie@narwal.com

最近的更新

工作交接表
3分钟以前 haojia xie(谢浩佳)更新 查看变动
事件机制.png
九月 19, 2023 haojia xie(谢浩佳)添加了附件
事件机制
九月 19, 2023 haojia xie(谢浩佳)添加了附件
统计非低功耗下mcu上报的0dom数据的转发延时及丢包情况
九月 19, 2023 haojia xie(谢浩佳)更新 查看变动
node_hal_丢包率.zip
九月 19, 2023 haojia xie(谢浩佳)添加了附件
node_hal.zip
九月 19, 2023 haojia xie(谢浩佳)添加了附件
监控程序添加Ros通讯模块
九月 15, 2023 haojia xie(谢浩佳)更新 查看变动
监控埋点上报APP.png
九月 15, 2023 haojia xie(谢浩佳)添加了附件
监控埋点上报APP
九月 15, 2023 haojia xie(谢浩佳)添加了附件
熟悉hal节点
九月 15, 2023 haojia xie(谢浩佳)更新 查看变动
E.png
九月 15, 2023 haojia xie(谢浩佳)添加了附件
E
九月 15, 2023 haojia xie(谢浩佳)添加了附件
C++ chrono 时钟库的使用
九月 15, 2023 haojia xie(谢浩佳)更新 查看变动
思维导图名称 - Thu Sep 14 2023 - 21:32:59
九月 15, 2023 haojia xie(谢浩佳)添加了附件
思维导图名称 - Thu Sep 14 2023 - 21:32:59.png
九月 15, 2023 haojia xie(谢浩佳)添加了附件

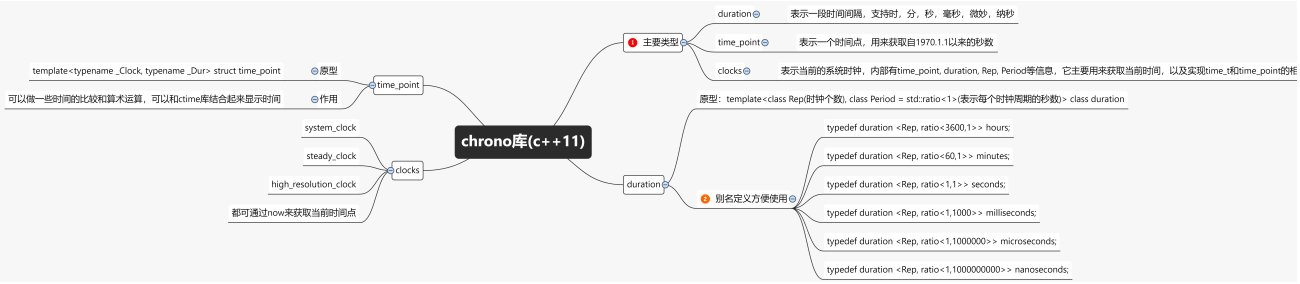
空间导航

00. 入职学习

00.01 基础知识

C++ chrono 时钟库的使用

> https://liucjy.blog.csdn.net/article/details/131198438?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EYuanLiJiHua%7EPosition-2-131198438-blog-28599655.235%5Ev38%5Epc_relevant_yljh&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EYuanLiJiHua%7EPosition-2-131198438-blog-28599655.235%5Ev38%5Epc_relevant_yljh&utm_relevant_index=5



ROS中的两种通信机制

- 实现ROS的异步话题通讯机制，进一步封装Pub/Sub接口。使用了模板类/模板成员函数。
- 熟悉在c++类中使用成员函数进行回调（需使用bind进行绑定）
- 实现ROS的同步服务通信和命令台相关控制命令的使用（主要有rqt_graph,rosnode,rostopic,rosmsg,rosserver等）
- 熟悉C++关于类型，变量，函数，宏等的命令规范。

一 以类的方式封装ROS中的两种通信机制（异步话题通讯，同步服务通讯）

1 异步话题通讯

适用场景：

- 数据需频繁更新，需要一对N的数据提供且无需反馈，实时性要求较低。如传感器数据的发布和订阅等

发布者实现代码：

```
//talker.cpp
#include <sstream>
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "communication/Person.h"
using namespace std;

template <typename T>
class SendHello
{
public:
        //queue_size:
        SendHello(string topicName, int queue_size)
        {
                 //
                chatter_pub_ = n_.advertise<T>(topicName, queue_size);
        }
        void Publish()
        {
                //
                ros::Rate loop_rate(10);
                // int count = 0;
                while (ros::ok())
                {
                        //std_msgs::String "hello world"
                        // // std_msgs::String
                        // std_msgs::String msg;
                        // std::stringstream ss;
                        // ss << "hello world" << count;
                        // msg.data = ss.str();
                        // //
                        // ROS_INFO("%s", msg.data.c_str());
                        // chatter_pub.publish(msg);
                        // //
                        // ros::spinOnce();
                        // //
                        // loop_rate.sleep();
                        // ++count;
                   //Person Person
                        communication::Person msg;
                        msg.name = "XieHaoJia";
                        msg.age = 26;
                        msg.sex = communication::Person::male;
                        chatter_pub_.publish(msg);
                        ROS_INFO("publish message is ok: name:%s age:%d sex: %d",
                                        msg.name.c_str(), msg.age, msg.sex);
                        loop_rate.sleep();
                }
        }

private:
        //
        ros::NodeHandle n_;
        // Publisherchattertopicstd_msgs::String
        ros::Publisher chatter_pub_;
};

int main(int argc, char **argv)
{
        ros::init(argc, argv, "talker");
        //communication::Person"Person"
        SendHello<communication::Person> sendh("Person", 1000);
        sendh.Publish();
        //
         ros::spinOnce();
        return 0;
}
```

订阅者代码：

```
#include "ros/ros.h"
#include "std_msgs/String.h"
// #include <functional>
#include "communication/Person.h"

using namespace std;
// std_msgs::String
// void chatterCallback(const std_msgs::String::ConstPtr &msg)
// {
//         //
//         ROS_INFO("I heard: [%s]", msg->data.c_str());
// }
// communication::Person
void PersonCallBack(const communication::Person::ConstPtr &msg)
{
        ROS_INFO("subscribe this messgae:[name:%s  age:%d  sex:%d] ",
                        msg->name.c_str(), msg->age, msg->sex);
}
//T1
template <typename T1>
class RecvHello
{
public:
        RecvHello() {}
    //T2
        template <typename T2>
        void Subscribe(string topicName, int queue_size, T2 callBack)
        {
                //bind
                // sub = n.subscribe<std_msgs::String>("chatter", 1000, std::bind(&RecvHello::
chatterCallback, this, _1));
                sub_ = n_.subscribe<T1>(topicName, queue_size, callBack);
        }

private:
        //
        ros::NodeHandle n_;
        ros::Subscriber sub_;
};

int main(int argc, char **argv)
{
        // ROS
        ros::init(argc, argv, "listener");
        //communication::Person
        RecvHello<communication::Person> recvh;
        // recvh.Subscribe("chatter", 1000, chatterCallback);
        //"Person"1000PersonCallBackT2
        recvh.Subscribe("Person", 1000, PersonCallBack);

        ros::spin();
        return 0;
}
```

2 同步服务通讯

服务器代码：

```
//server.cpp
#include <ros/ros.h>
#include "communication/AddTwoInts.h"
using namespace std;
//
bool add(communication::AddTwoInts::Request &req, communication::AddTwoInts::Response &res)
{
        //
        res.sum = req.a + req.b;
        ROS_INFO("request: x=%1d,y=%1d", (long int)req.a, (long int)req.b);
        ROS_INFO("sending back response:[%1d]", (long int)res.sum);
        return true;
}

class ServerAdd
{
public:
        //
        template <typename T>
        ServerAdd(string server_name, T add)
        {
                service_ = n_.advertiseService(server_name, add);
                //
                ROS_INFO("Ready to add two ints.");
        }

private:
        //
        ros::NodeHandle n_;
        // add_two_intsserver,add()
        ros::ServiceServer service_;
};

// servicereqres

int main(int argc, char **argv)
{
        // ROS
        ros::init(argc, argv, "add_two_ints_server");
        //"add_two_ints"add
        ServerAdd server_add("add_two_ints", add);
        ros::spin();
        return 0;
}
```

客户端代码：

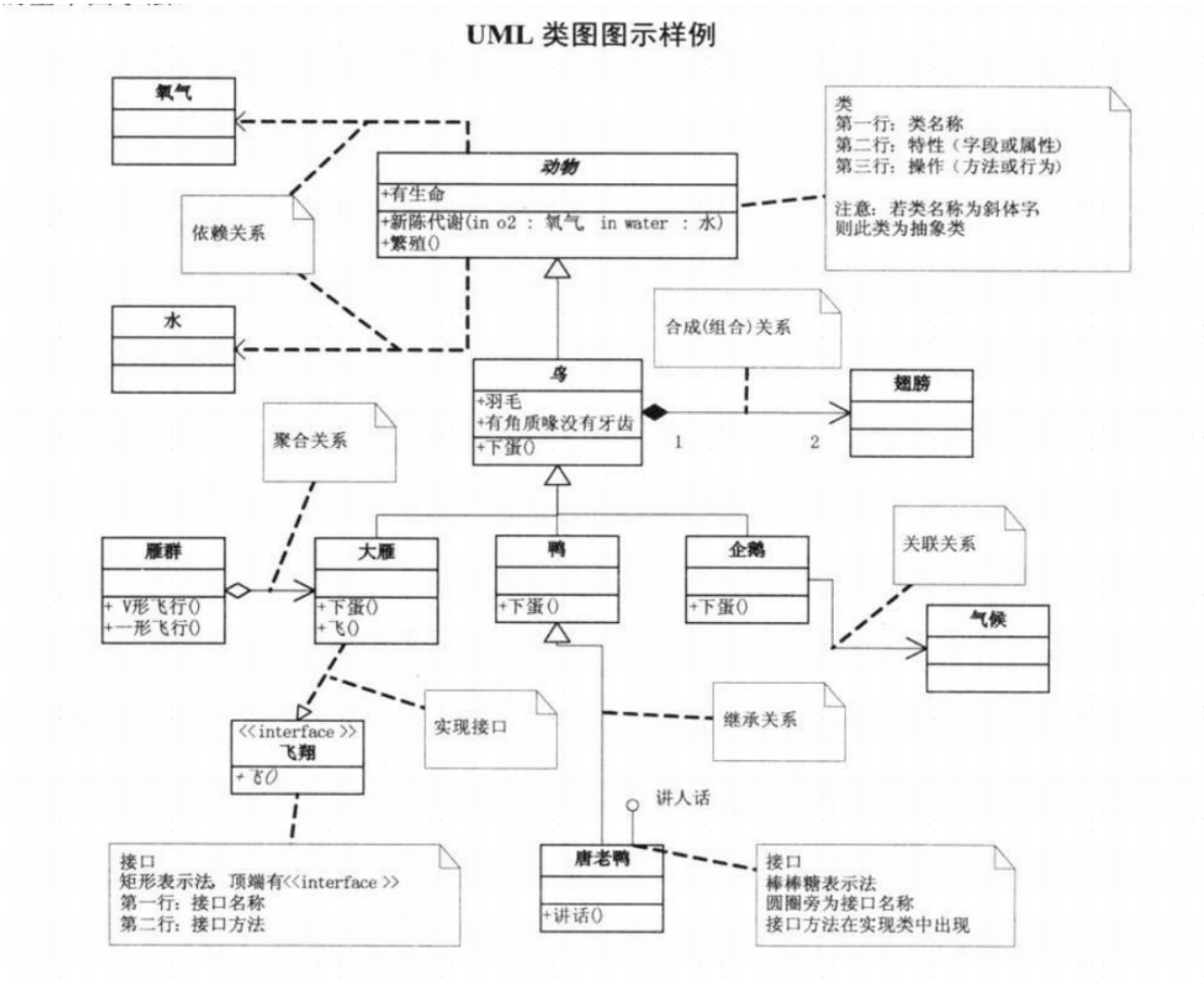
```
#include <cstdlib>
#include <ros/ros.h>
#include "communication/AddTwoInts.h"
using namespace std;
class Client
{
public:
        Client(string server_name)
        {
                // clientadd_two_ints_service
                // servicelearning_communication::AddTwoInts
                client_ = n_.serviceClient<communication::AddTwoInts>(server_name);
        }
        template <typename T>
        void Request(T srv)
        {

                // learning_communication::AddTwoIntsservice

                // service
                if (client_.call(srv))
                {
                        ROS_INFO("sum: %1d", (long int)srv.response.sum);
                }
                else
                {
                        ROS_INFO("Failed to call service add_two_ints");
                        return;
                }
        }

private:
        //
        ros::NodeHandle n_;
        ros::ServiceClient client_;
};

int main(int argc, char **argv)
{
        // ROS
        ros::init(argc, argv, "add_two_ints_client");
        // // clientadd_two_ints_service
        // servicelearning_communication::AddTwoInts
        if (argc != 3)
        {
                ROS_INFO("usage:add_two_ints_client X Y");
                return 1;
        }
    //
        communication::AddTwoInts srv;
        srv.request.a = atoll(argv[1]);
        srv.request.b = atoll(argv[2]);
        //
        Client client("add_two_ints");
        //
        client.Request(srv);
        return 0;
}
```

3 两者的区别

	话题	服务
同步性	异步	同步
通信模型	发布/订阅	服务器/客户端
底层协议	ROSTCP/ROSUDP	ROSTCP/ROSUDP
反馈机制	无	有
缓冲区	有	无
实时性	弱	强
节点关系	多对多	一对多（一个server）
适用场景	数据传输	逻辑处理

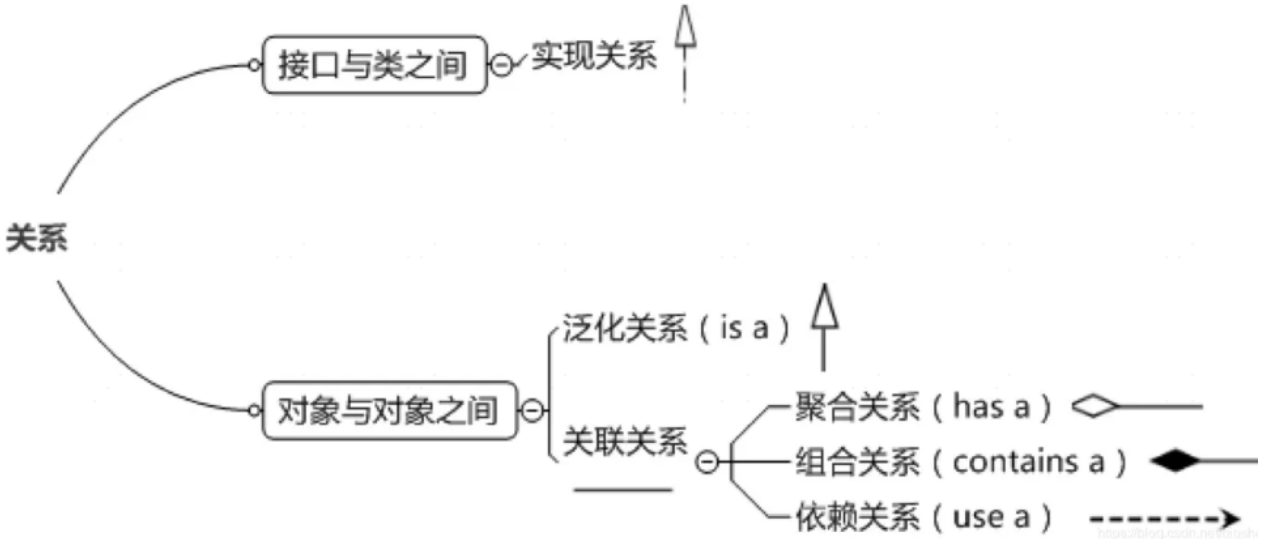
UML类图关系类别及表示方式

0 例图



1 关系类别

在UML类图中，常见的关系有泛化（Generalization），实现（Realization），关联（Association），聚合（Aggregation），组合(Composition)，依赖(Dependency)等



2 关系含义及表示方式（以C++代码为例）

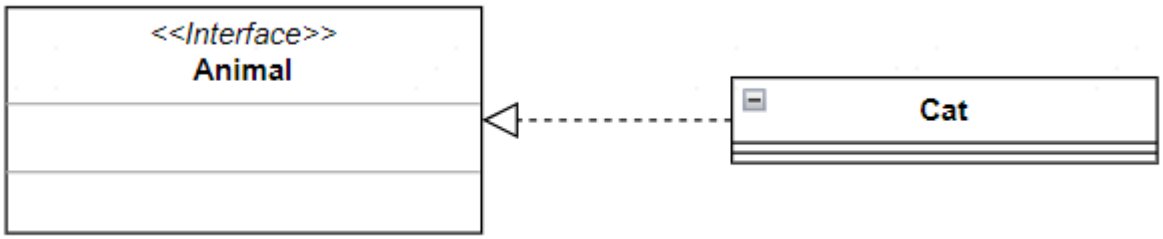
1）实现关系

实现关系是指接口及其实现类之间的关系。

代码形式：使用纯虚函数实现抽象类作为接口，派生类为实现类：

```
class Animal
{
public:
    virtual void do_something() = 0;
};

class Cat : public Animal
{
public:
    void do_something() override
    {
    }
};
```

UML类图表示：



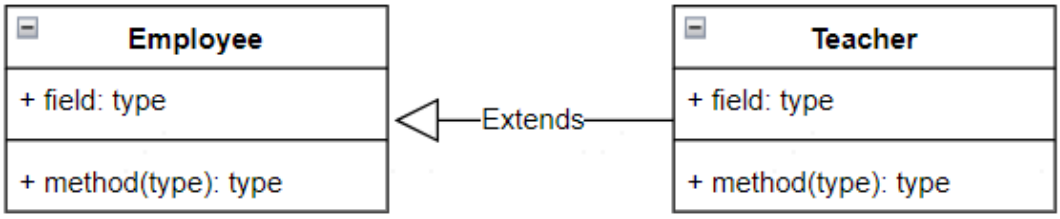
2）泛化关系（继承关系 “is a”）

泛化关系是指对象与对象之间的继承关系。如果两个对象之间拥有“is a” 的关系，那两者之间就存在继承关系。

代码形式：

```
class Employee
{

};

class Teacher : public Employee
{

};
```

UML类图表示：



3）关联关系（Assocation）

关联关系（Association）是指对象和对象之间的连接，它使一个对象知道另一个对象的属性和方法。如果一个对象的类代码中，包含有另一个类的实例对象，那么这两个对象之间就是关联关系。关联关系有单向关联和双向关联。如果两个对象都知道（即可以调用）对方的公共属性和操作，那么二者就是双向关联。如果只有一个对象知道（即可以调用）另一个对象的公共属性和操作，那么就是单向关联。大多数关联都是单向关联，单向关联关系更容易建立和维护，有助于寻找可重用的类。

代码形式：

```
class TimeCard
{
};

class Employee
{

private:
    TimeCard tc_;
};
```

UML类图表示（双向关联关系用带双箭头的实线或者无箭头的实线双线表示。单向关联用一个带箭头的实线表示，箭头指向被关联的对象）：

注：

- 数字：精确的数量
- *或者0..*：表示0到多个
- 0..1：表示0或者1个，在Java中经常用一个空引用来实现
- 1..*：表示1到多个

关联关系又分为**依赖关联，聚合关联和组合关联**三种类型。

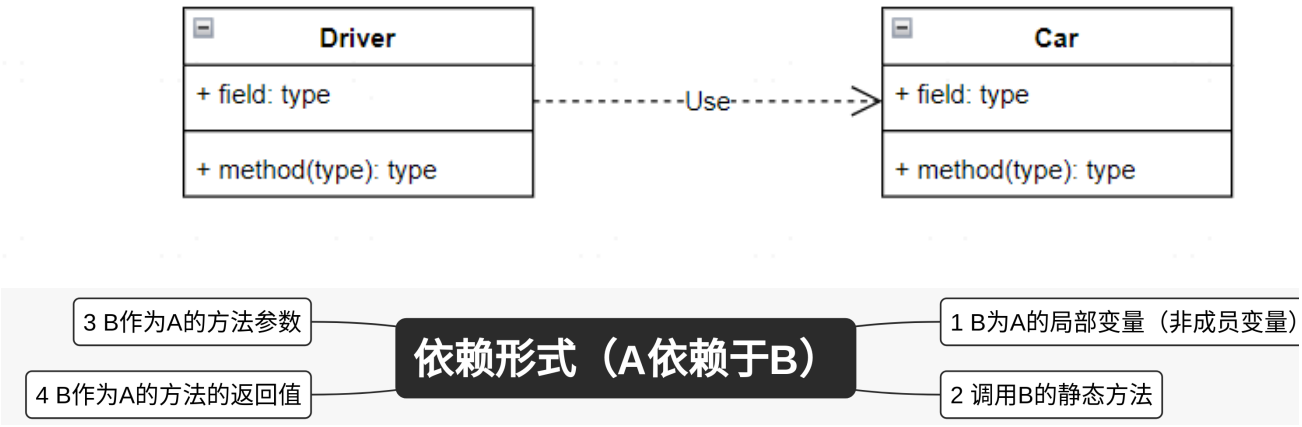
a）依赖关系（Dependency）

依赖（Dependency）关系是一种弱关联关系。如果对象A用到对象B，但是和B的关系不是太明显的时候，就可以把这种关系看作是依赖关系。如果对象A依赖于对象B，则 A “use a” B。比如驾驶员和汽车的关系，驾驶员使用汽车，二者之间就是依赖关系。

代码形式：

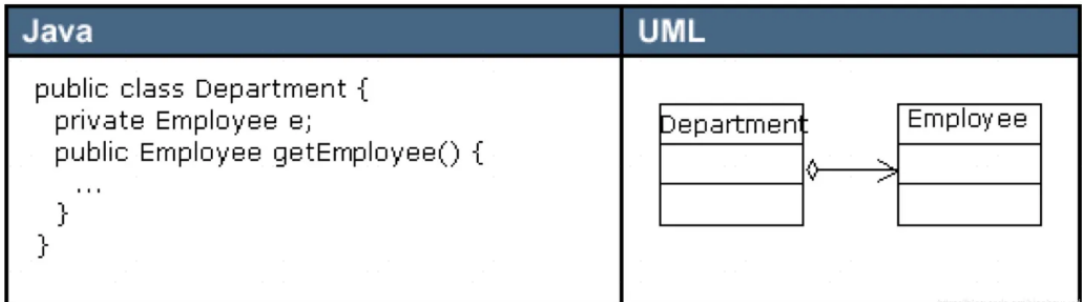
```
class Car
{
};

class Driver
{
private:
    void driverCar(Car c){

    }
};
```

UML类图表示（双向关联关系用带双箭头的实线或者无箭头的实线双线表示。单向关联用一个带箭头的实线表示，箭头指向被关联的对象）：



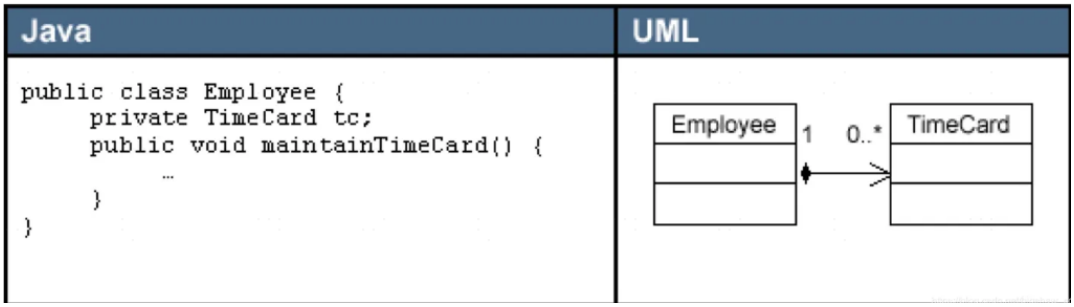
b）聚合关系（Aggregation）

聚合（Aggregation）是关联关系的一种特例，**它体现的是整体与部分的拥有关系，即 “has a” 的关系**。此时整体与部分之间是可分离的，它们可以具有各自的生命周期，**部分可以属于多个整体对象，也可以为多个整体对象共享，所以聚合关系也常称为共享关系**。例如，公司部门与员工的关系，一个员工可以属于多个部门，一个部门撤消了，员工可以转到其它部门。



c）组合关系（Composition）

组合（Composition）也是关联关系的一种特例，**它同样体现整体与部分间的包含关系，即 “contains a” 的关系**。但此时整体与部分是不可分的，部分也不能给其它整体共享，作为整体的对象负责部分的对象的生命周期。**这种关系比聚合更强，也称为强聚合**。如果A组合B，则A需要知道B的生存周期，即可能A负责生成或者释放B，或者A通过某种途径知道B的生成和释放。



机器人了解情况

- 公司部门的分工以及本部门(系统服务部)所负责的相关业务。
- MCU和SOC的关系？：MCU通常用于较简单的控制和调度任务，而SOC则用于较复杂的计算和通信任务。SOC可以包含一个或多个MCU。
- 每个产品都会有对应的项目名.（IP1221-1,IP1221-2，IP1401,IP1001等。）
- J1/J2,J3的构成结构。主要了解它们之间的大致差异，以及各种传感器的位置以及起到的作用。

1 了解ROS的主要功能和学习运行ros代码的步骤（①创建工作空间并初始化/catkin_make，②进入src创建ros包并添加依赖/catkin_create_pkg ROS roscpp rospy std_msgs,）

2 学习ROS进程间通信中的两种通讯机制（基于同步的RPC服务、基于异步的流媒体话题）。

3 学习ROS其它节点与master节点之间的关系，了解机器人上有哪几种传感器类型。

ROS运行代码的步骤：

```
### 1
mkdir-p firstSpace/src
cdfirstSpace
catkin_make

### 2. src  ros
cdsrc
catkin_create_pkg ROS roscpp rospy std_msgs

### 3. ros  src
cd/src
vim talker.cpp

### 4. ros  Cmakelist.txt
add_executable(3
  src/3.cpp(talker.cpp)
)
target_link_libraries(3
  ${catkin_LIBRARIES}
)
### 5

cdfirstSpace
catkin_make

### 6

roscore
cd
source./devel/setup.bash
rosrun ( communication ) C++/(talker)
```

ROS(提供了一套框架和工具库)

硬件抽象

底层设备控制

常用函数及库实现

进程间通信（主要有三种）

roscore , rosnode/rostopic list

机器人了解情况（下）

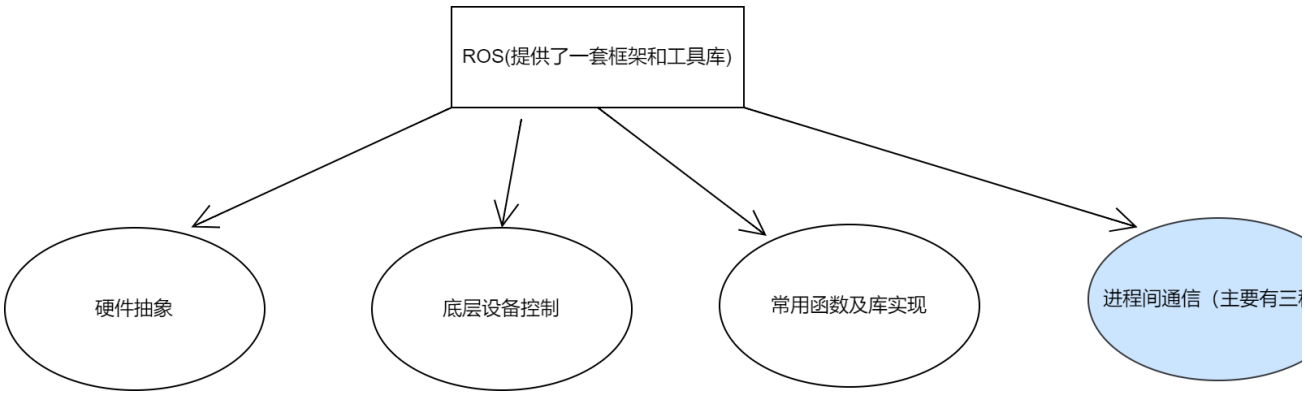
1 了解ROS的主要功能和学习运行ros代码的步骤（①创建工作空间并初始化/catkin_make，②进入src创建ros包并添加依赖/catkin_create_pkg ROS roscpp rospy std_msgs,）

2 学习ROS进程间通信中的两种通讯机制（基于同步的RPC服务、基于异步的流媒体话题）。

3 学习ROS其它节点与master节点之间的关系，了解机器人上有哪几种传感器类型。

ROS运行代码的步骤：

```
### 1
mkdir -p firstSpace/src
cd firstSpace
catkin_make

### 2. src  ros
cd src
catkin_create_pkg ROS roscpp rospy std_msgs

### 3. ros  src
cd /src
vim talker.cpp

### 4. ros  Cmakelist.txt
add_executable(3
  src/3.cpp(talker.cpp)
)
target_link_libraries(3
  ${catkin_LIBRARIES}
)
### 5

cd firstSpace
catkin_make

### 6

roscore
cd
source ./devel/setup.bash
rosrun ( communication ) C++/(talker)
```



roscore , rosnode/rostopic list

设计模式

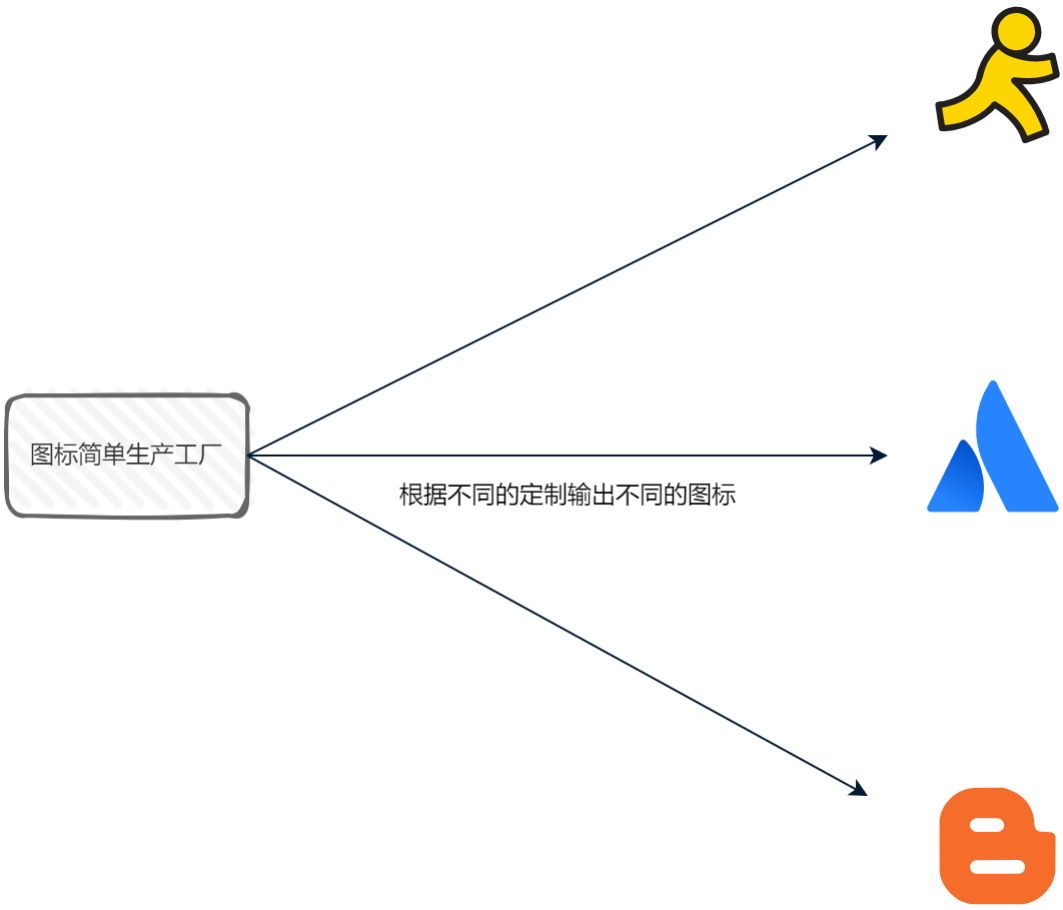
> *“面向对象的编程，并不是类越多越好，**类的划分是为了封装**，但**分类的基础是抽象**，具有相同属性和功能的对象的抽象集合才是类。”*

一 简单工厂模式

1 用途：解决对象创建的问题

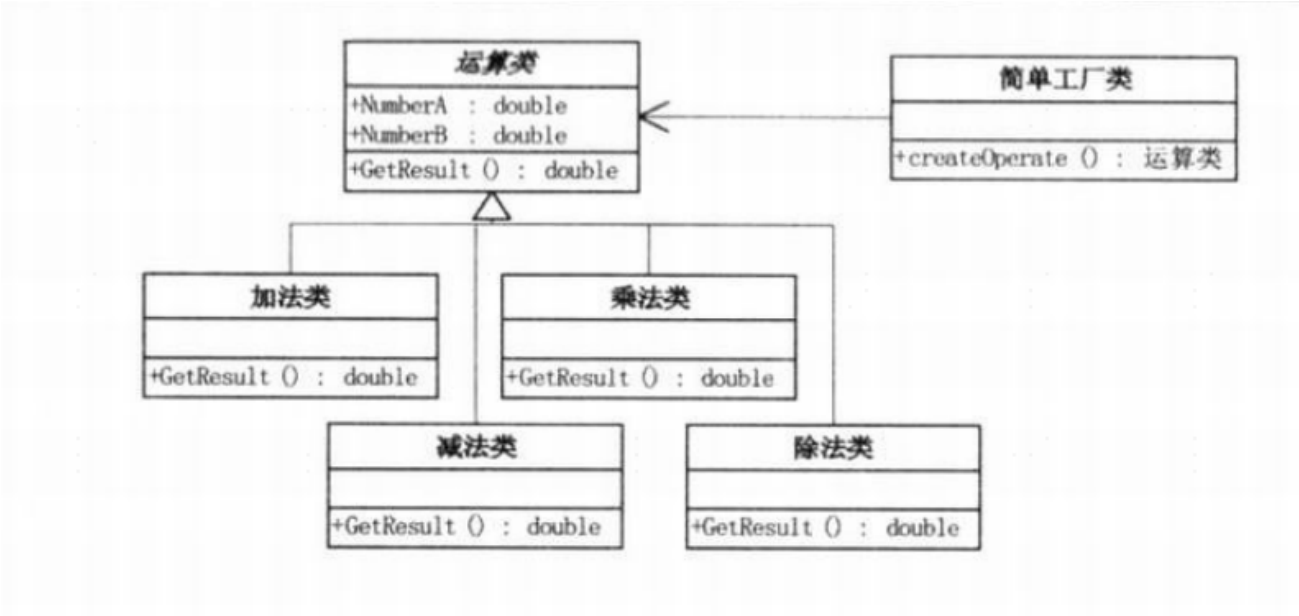
可以**集中**处理对象的创建过程，**将客户端代码与具体对象的创建过程解耦**，提供了一种灵活、可扩展的**对象创建方式**。

2 具体的例子：根据老化配置文件来定制具体的老化测试项目。

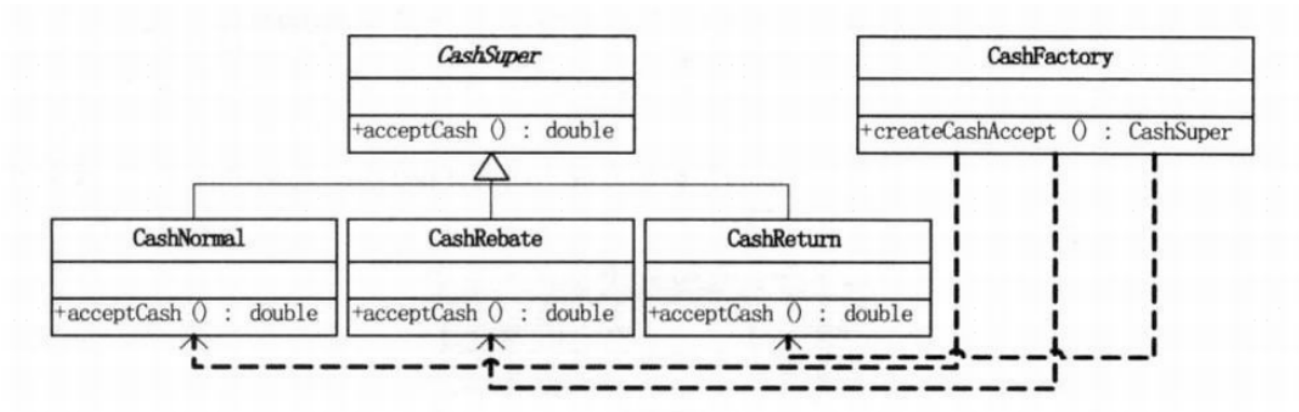


3 代码结构-UML图

①计算器



②商场促销收费软件

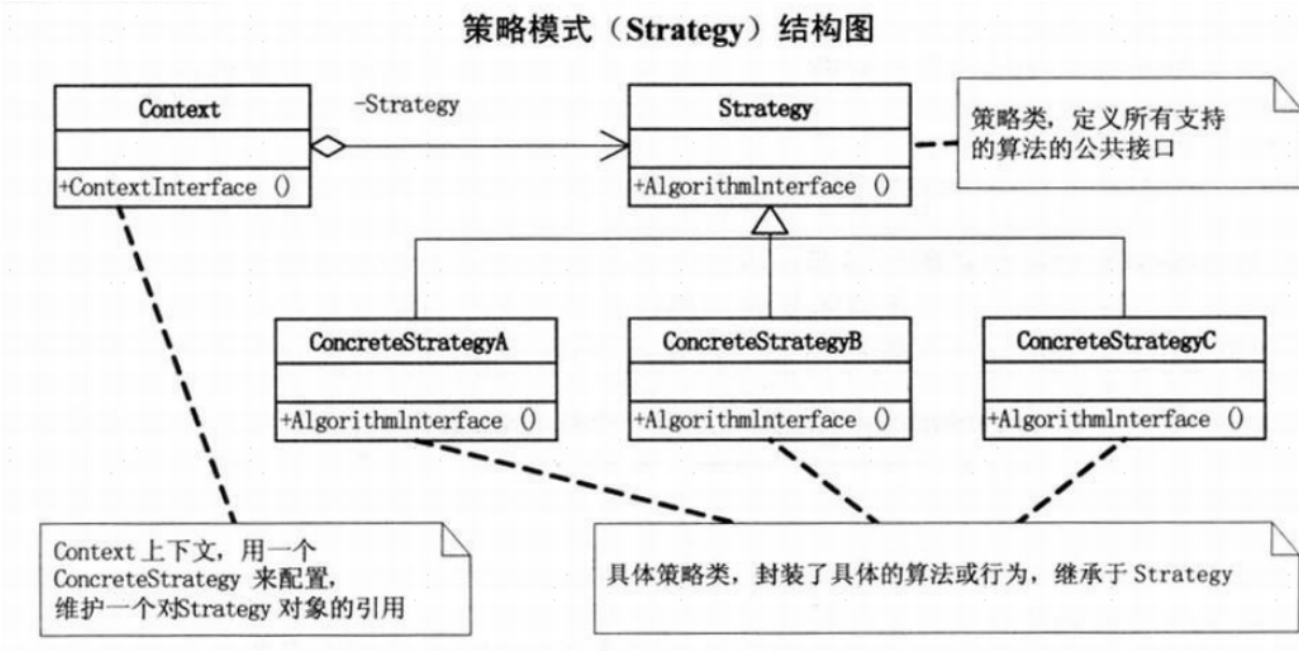


二 策略模式

1 用途：解决对象创建的问题

2 具体的例子：根据老化配置文件来定制具体的老化测试项目

3 代码结构-UML图


策略模式（Strategy）结构图
Context
+ContextInterface ()
-Strategy
Strategy
+AlgorithmInterface ()
策略类，定义所有支持的算法的公共接口
ConcreteStrategyA
+AlgorithmInterface ()
ConcreteStrategyB
+AlgorithmInterface ()
ConcreteStrategyC
+AlgorithmInterface ()
Context 上下文，用一个 ConcreteStrategy 来配置，维护一个对Strategy 对象的引用
具体策略类，封装了具体的算法或行为，继承于 Strategy

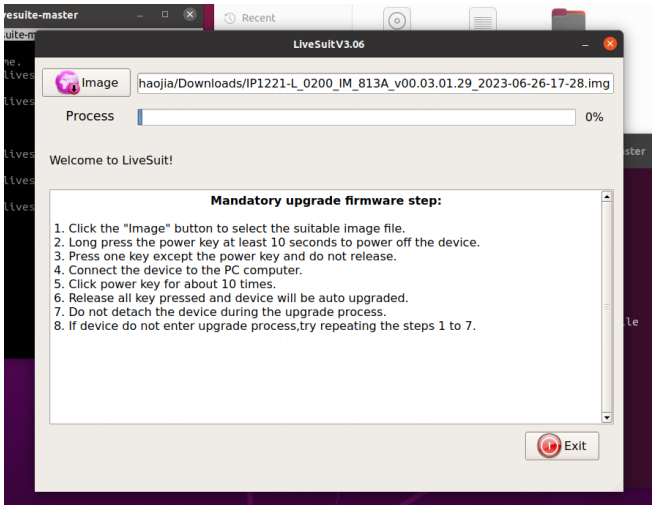
输出新版环境搭建指南

一 对J3Lite机器人的烧录过程 （Ubuntu20.04）

1 安装烧录软件sunxi-livesuite

```
### sunxi-livesuite make
cd ./sunxi-livesuite-master/awusb
sudo make clean
make
cp awusb.ko /lib/modules/`uname -r`/kernel/
sudo depmod -a
sudo insmod awusb.ko
sudo modprobe awusb

sudo apt install libpng12-dev
### ubuntu20.04
sudo add-apt-repository ppa:linuxuprising/libpng12
sudo apt update
sudo apt install libpng12-0

###
cd ../
sudo ./LiveSuit.sh
```

运行软件图，选择好对应版本的镜像然后开始进行烧录



二 对hal节点进行编译

三 rosbag的使用方法和熟悉foxglove平台。

1 作用：用于录制和回放Ros主题消息的一个工具集、实现了对数据的复用，方便调试、测试。

2 本质：rosbag本质也是ros的节点，当录制时，rosbag是一个订阅节点，可以订阅话题消息并将订阅到的数据写入磁盘文件；当重放时，rosbag是一个发布节点，可以读取磁盘文件，发布文件中的话题消息。

3 常用的命令：

rosbag对软件包来操作，一个包是ROS用于存储ROS消息数据的文件格式，rosbag命令可以记录、回放和操作包。指令列表如下：

命令	作用
cheak	确定一个包是否可以在当前系统中进行，或者是否可以迁移。
decompress	压缩一个或多个包文件。
filter	解压一个或多个包文件。
fix	在包文件中修复消息，以便在当前系统中播放。
help	获取相关命令指示帮助信息
info	总结一个或多个包文件的内容。
play	以一种时间同步的方式回放一个或多个包文件的内容。
record	用指定主题的内容记录一个包文件。
reindex	重新索引一个或多个包文件。

4 以乌龟demo来演示rosbag的使用

```
### demo
roscore
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key

###
cd ~/bagfiles
//-a
rosbag record -a -O my.bag

###    YAML rosbag info -y filename.bag
rosbag info xx.bag

###  topic
rosbag play <bagfile>
rosbag play -l  <bagfile>  # -l== --loop
rosbag play <bagfile> --topic /topic1

#

### topic
rosbag play file.bag /topic_name:=/reame_topic_name
#//topic_nametopicreame_topic_nametopic
```

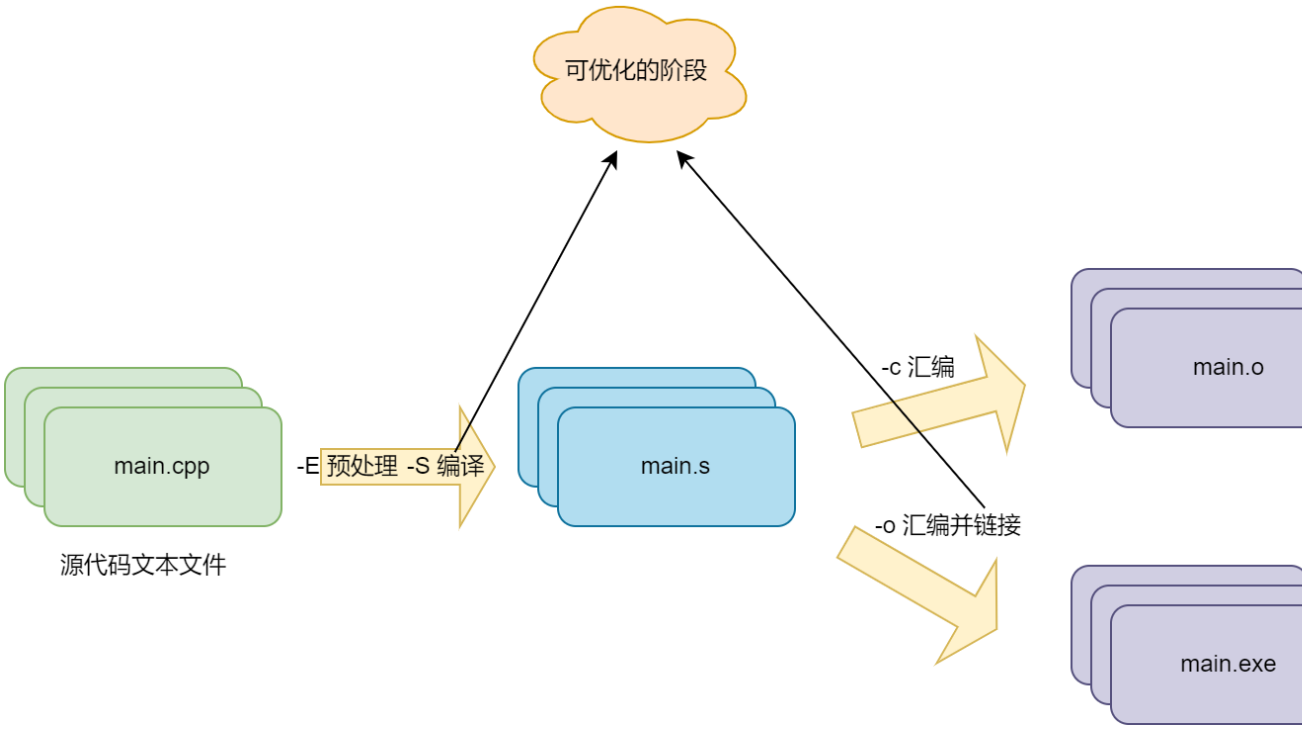
00.02 方法集

G++参数编译优化

注：不加任何跟编译优化相关参数的话，gcc/g++编译器默认是在确保能产生预期的调试信息下尽可能地降低编译成本。而打开优化标志会使编译器试图以牺牲编译时间，调试程序的能力以及更多的内存消耗为代价来提高性能/或代码大小。

参考文档：

1. GCC 参数详解
2. gcc6.4.0文档(编译选项优化可参考3.11节，特定平台下的参数可参考3.18节)



并非所有优化都直接由标志控制。 本节仅列出具有标志的优化。可使用g++ -Q --help=optimizers 列出该编译器所支持的优化参数。

一 与特定平台无关

1 是否生成警告信息：

-w(不生成任何警告信息)

-Wall(生成所有警告信息)

2 是否对代码进行编译优化：

-O/-O0： 不做任何优化，这是默认的编译选项 ；

-O1：使用能减少目标文件大小以及执行时间并且不会使编译时间明显增加的优化。 该模式在编译大型程序的时候会花费更多的时间和内存。在 -O1下：编译会尝试减少代码体积和代码运行时间，但是并不执行会花费大量时间的优化操作。

-O2：包含 -O1的优化并增加了不需要在目标文件大小和执行速度上进行折衷的优化。 GCC执行几乎所有支持的操作但不包括空间和速度之间权衡的优化，编译器不执行循环展开以及函数内联。这是推荐的优化等级，除非你有特殊的需求。 -O2会比 -O1启用多 一些标记。与 -O1比较该优化 -O2将会花费更多的编译时间当然也会生成性能更好的代码。

-O3：打开所有 -O2的优化选项并且增加 -finline-functions, -funswitch-loops,-fpredictive-commoning, -fgcse-after-reload and -ftree-vectorize优化选项。这是最高最危险的优化等级。用这个选项会延长编译代码的时间，并且在使用 gcc4.x的系统里不应全局 启用。自从 3.x版本以来 gcc的行为已经有了极大地改变。在 3.x，，-O3生成的代码也只 是比 -O2快一点点而已，而 gcc4.x中还未必更快。用 -O3来编译所有的 软件包将产生更大体积更耗内存的二进制文件，大大增加编译失败的机会或不可预知的程序行为（包括 错误）。这样做将得不偿失，记住过犹不及。在 gcc 4.x.中使用 -O3是不推荐的。

-Os：专门优化目标文件大小 ，执行所有的不增加目标文件大小的 -O2优化选项。同时 -Os还会执行更加优化程序空间的选项。这对于磁盘空间极其紧张或者 CPU缓存较小的 机器非常有用，**适合于嵌入设备上**。但也可能产生些许问题，因此软件树中的大部分 ebuild都过滤掉这个等级的优化。使用 -Os是不推荐的。

可对各个级别的优化进行微调，具体见文档page-96

3 比较细粒化的控制：

1. -finline-functions：这个参数启用函数内联优化，将函数调用替换为函数体的实际代码，可以减少函数调用的开销。
2. -floop-optimize：这个参数启用循环优化，对循环结构进行优化以提高执行速度。
3. -fomit-frame-pointer：这个参数告诉编译器不使用框架指针，从而节省寄存器并提高代码的执行速度。
4. -funroll-loops：这个参数启用循环展开优化，将循环展开为多个迭代的代码，以减少循环开销并提高性能。
5. -fprefetch-loop-arrays：这个参数启用循环数组预取优化，可以提前将数据加载到缓存中，以减少内存访问延迟。
6. -march=native：这个参数告诉编译器使用当前机器的本地指令集来生成优化的机器代码。
7. -finline-small-functions：这个参数类似于-finline-functions，但仅对较小的函数进行内联优化。
8. -fmerge-all-constants：这个参数合并所有常量，以减少可执行文件中的常量数据大小。
9. -flto：这个参数启用链接时优化（Link-Time Optimization），允许在链接阶段对整个程序进行全局的优化。
10. -fomit-frame-pointer：这个参数告诉编译器不使用框架指针，从而节省寄存器并提高代码的执行速度。
11. -ffunction-sections和-fdata-sections：这些参数将每个函数和数据放置在独立的节（section）中，以便在链接时进行更细粒度的优化和减少可执行文件的大小。
12. -Wl,--gc-sections：这个参数告诉链接器去除未使用的节，进一步减少可执行文件的大小。

4 削减可执行文件的空间大小：

"strip" 是一个工具，用于从可执行文件或共享库中去除符号信息（symbol information）。它通常用于减小可执行文件或共享库的大小，以及保护源代码的信息。

二 与特定平台相关（针对aarch64）

1. -march=armv8-a：指定目标处理器的架构为ARMv8-A，即aarch64架构。
2. -mtune=cortex-a53：指定目标处理器的调优为Cortex-A53，根据实际情况可以选择其他Cortex系列的处理器。
3. -mfpu=neon：启用NEON（AdvancedSIMD）浮点单元指令集，用于加速向量运算和并行计算。
4. -mfloat-abi=hard：使用硬件浮点运算，以提高浮点运算的性能。
5. -mcpu=native：根据当前编译机器的CPU类型自动选择最优的优化参数，包括架构、调优和浮点运算。

6. -ftree-vectorize：启用向量化优化，将循环中的计算转化为向量指令以提高执行效率。

7. -mve：启用 ARMv8.1-M Mainline Extension（MVE）的向量化优化。该扩展提供了更多的向量指令和寄存器，可以进一步提高向量化计算的性能。

使用Linux上自带的工具分析故障及性能问题

如何使用gdb查看程序的堆栈信息

参考文档：

https://zhuanlan.zhihu.com/p/297925056

https://blog.csdn.net/qq_39107832/article/details/119206954

一 gdb 常见命令

命令名称	命令缩写	命令说明
run	r	运行一个待调试的程序
continue	c	让暂停的程序继续运行
next	n	运行到下一行
step	s	单步执行，遇到函数会进入
until	u	运行到指定行停下来
finish	fi	结束当前调用函数，回到上一层调用函数处
return	return	结束当前调用函数并返回指定值，到上一层函数调用处
jump	j	将当前程序执行流跳转到指定行或地址
print	p	打印变量或寄存器值
backtrace	bt	查看当前线程的调用堆栈
frame	f	切换到当前调用线程的指定堆栈
thread	thread	切换到指定线程
break	b	添加断点
tbreak	tb	添加临时断点
delete	d	删除断点
enable	enable	启用某个断点
disable	disable	禁用某个断点
watch	watch	监视某一个变量或内存地址的值是否发生变化
list	l	显示源码
info	i	查看断点 / 线程等信息
ptype	ptype	查看变量类型
disassemble	dis	查看汇编代码
set args	set args	设置程序启动命令行参数
show args	show args	查看设置的命令行参数

如何阅读源码


如何阅读源码
好处
一 提升
1 提升自己的编码水平
2 开拓自己写代码的思路
二 修复Bug
三 增加新功能
方法
一 上帝视角
1 了解项目的主要用途
2 了解项目的架构
二 跑玩代码
1. 知道这个项目运行前有哪些必须的前置条件
2 了解该代码所最终呈现的样子
3 解决疑惑
方法变量不知其意
逻辑跳转绕来绕去
封装对象层次太深
三 细化目标
1 分而治之
1 把大问题分解成小问题
2 每一个小目标就是一个个业务流程
2 横纵兼顾
四 阅读源码
1 将业务跟代码映射起来
2 代码阅读技巧
技巧一：代码一定跳着看，最优先看正向流程代码
可以忽略的代码
异常处理代码
数据处理的代码
底层交互的代码
技巧二：调用关系需确定
几种实现非顺序调用的方式
通过中间件继续后续流程，比如 MQ
通过异步方式继续后续流程，比如 Future 模式、Promise
通过回调方式继续后续流程
通过代理委托方式继续后续流程，比如动态代理只
通过依赖注入方式继续后续流程，比如 Spring 的 autowi
技巧三：超难算法放最后
五 把A变B
1 尝试修改代码或验证自己的想法
2 不要为了读代码而读代码，代码不是文学作品

搭建ARM开发环境（针对本地编译）

搭建ARM开发环境（针对本地编译）

1 安装docker环境并将用户加入到docker用户组中

```
### docker
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun

###  docker
###  sudo vim /etc/docker/daemon.json ,  /etc/docker/daemon.json
{
    "insecure-registries": ["http://harbor.devops.narwal.com"]
}

 ### docker
sudo  systemctl daemon-reload  systemctl restart docker

###  docker group,  docker
$ sudo usermod -aG docker $USER
```

2 配置Git环境

- 先申请git代码仓库的访问权限并使用ssh-keygen生成密钥并添加到git中。
- 配置好git的个人用户信息：

```
$ git config --global user.name ""
$ git config --global user.email ""
```

3 下载ARM编译环境镜像

1. **先配置好harbor仓库互信再拉取镜像（harbor仓库配置互信）**
2. **拉取交叉编译镜像并拉取仓库代码：**

```
$ docker pull harbor.devops.narwal.com/arm/cross_release_env_t1:latest
# pita2_workspace
$ git clone gitlab@git.narwel.com:pita2/workspace.git pita2_workspace

$ cd pita2_workspace
##
$./scripts/init_workspace
```

调用脚本scripts/init_workspace拉取各个模块的代码 【注：init_workspace脚本中拉取的代码缺少了robokit_telemetry，但加上后又会报出现包重名的问题，因为robokit_telemetry中已经包含了log_upload和robokit_robata两个，只需要将其删掉即可】

3. **创建容器并执行编译**

```
###
$ docker run -it --name pita2_arm_new_20230407 -v ~/pita2_workspace:/home/narwal/workspace harbor.
devops.narwal.com/arm/cross_release_env_t1:latest bash
#
cd /home/narwal/workspace
# arm
$ ./scripts/arm_compile.sh
```

【注： 最后一步编译可能得多编译几次才能成功】

程序崩溃分析堆栈原因

一 使用gdb调试生成的core文件

参考：https://blog.csdn.net/qq_42570601/article/details/114842320

1 gdb [exec file] [core file]

2 bt 或 where：定位到段错误在源程序中具体文件的具体位置。

二 使用breakpad文件查看生成的dmp文件

1 查找容器中已有的工具：**dump_syms**(解析程序符号信息) 和 **minidump_stackwalk**(将符号信息映射到dmp文件中并进行解析)

```
ln -s /opt/breakpad/src/src/tools/linux/dump_syms/dump_syms /bin/dump_syms
ln -s /opt/breakpad/src/src/processor/minidump_stackwalk /bin/minidump_stackwalk
```

2 创建一个临时目录（breakpad），然后将要分析的**可执行程序或是库**移到该目录中，并dump_syms命令进行解析（**注意生成的xxx.sym文件名要和前面分析的文件名一样，比如sys_monitor_node->sys_monitor_node.sym**）：

```
dump_syms xxx/xxx.a/xxx.so(sys_monitor_node) > sys_monitor_node.sym
```

3 查看生成符号文件的索引并创建相对应的目录（**注意目录名要和索引和节点名一致**）：

```
head -n1 sys_monitor_node.sym
mkdir -p ./symbols/sys_monitor_node/0BF92B7CE840ABE9B198536A8CB11BF40
mv sys_monitor_node.sym ./symbols/sys_monitor_node/0BF92B7CE840ABE9B198536A8CB11BF40/
```

4 使用minidump_stackwalk 对dmp文件进行解析生成最终的分析表（**注意 生成dmp文件的程序版本要和symbols目录下程序的版本一致**）：

```
minidump_stackwalk  xxxxx.dmp ./symbols/ > sys_monitor.stack
```

00.03 项目了解情况

各个产品号所对应的项目号：

IP1100-J1

IP1101-J2

IP1220/T1/pita2-J3

IP1221/P2V-J3 SE

IP1401 - J4

sys_monitor使用文档

1 语法格式

```
sysmonitor [options] [interval]
```

2 参数解析

a. 可单独使用的参数

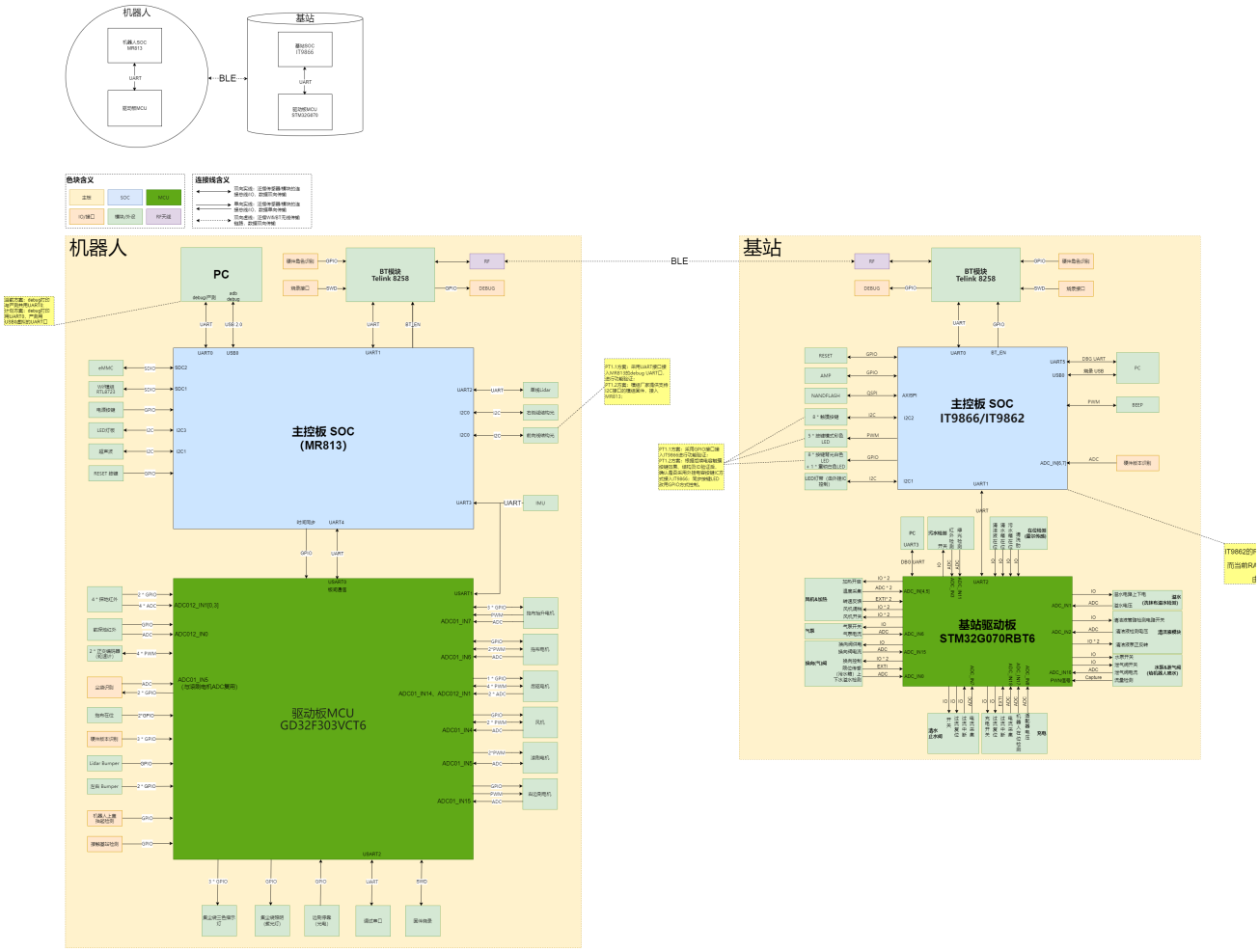
参数	功能
-h	打印使用手册
-v	打印版本信息
-H	转换为人类可读单位:K M G，默认单位为KB
-b	获取启动信息以及关机原因
-c/u	打印cpu信息
-m/r	打印内存信息
-d	打印磁盘信息
-N	打印所有节点信息，默认按CPU排序，可跟加参数-s按内存值进行排序
-n	暂不支持
-a	等于参数-HcmdNn
-A	等于参数-HcmdNnM
-P <pid>	监控进程
-p <name>	监控进程

b. 不可单独使用的参数

参数	功能
-t	打印每个进程的线程信息
-R	上报数据
-s	按内存值进行排序

熟悉hal节点

物理硬件连接图：

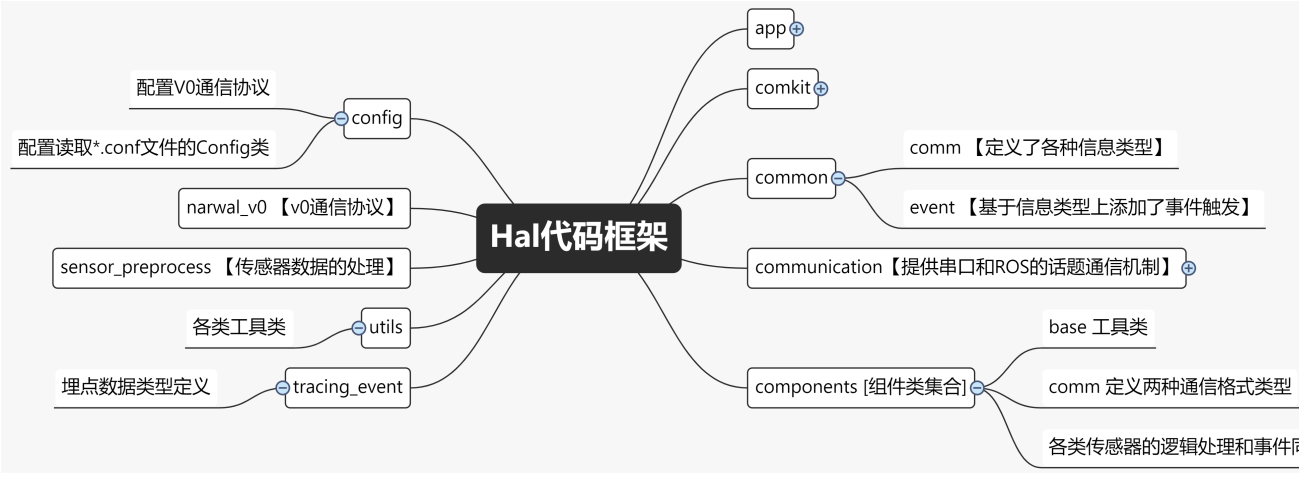


一 模块功能

提供对硬件设备(雷达、传感器、红外线等)的抽象，简化对硬件的控制，该层主要有以下作用：

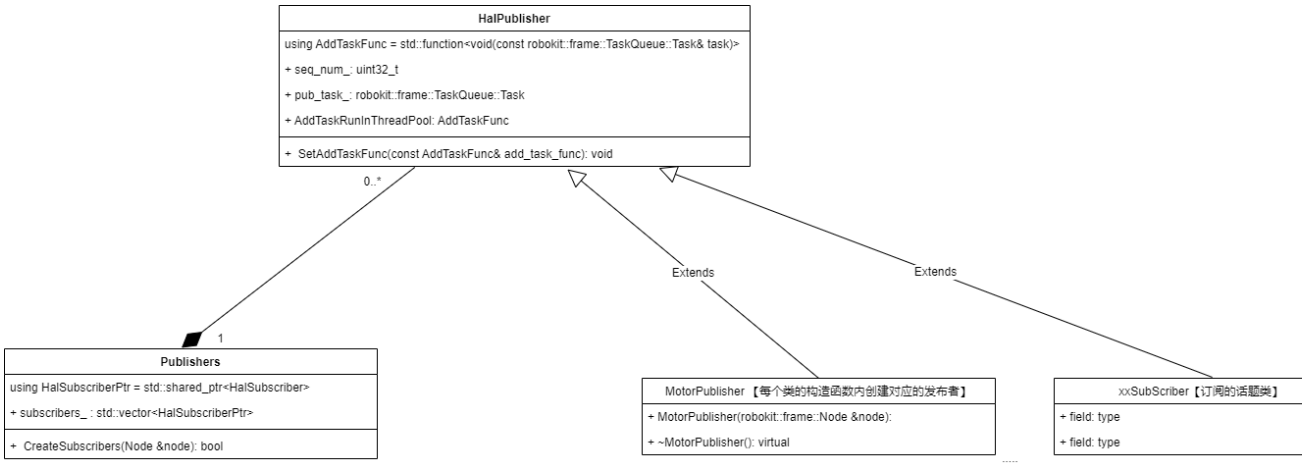
1. 与下位机mcu交互；
2. 获取各类传感器的数据并进行处理转发；
3. 对外设的控制（开关机按钮、LED控制等）

二 模块目录结构

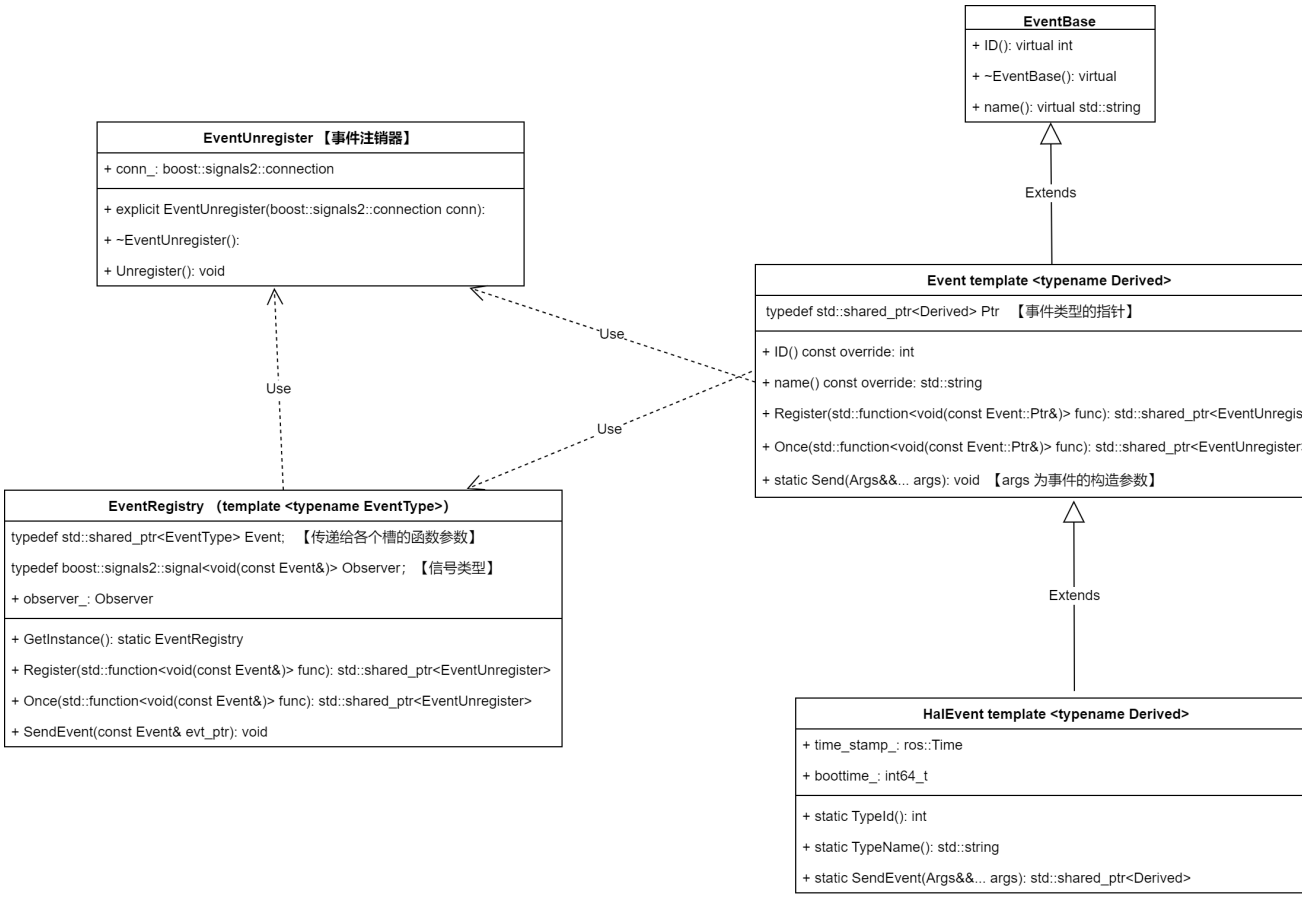


三模块代码结构

1 订阅/发布节点的框架

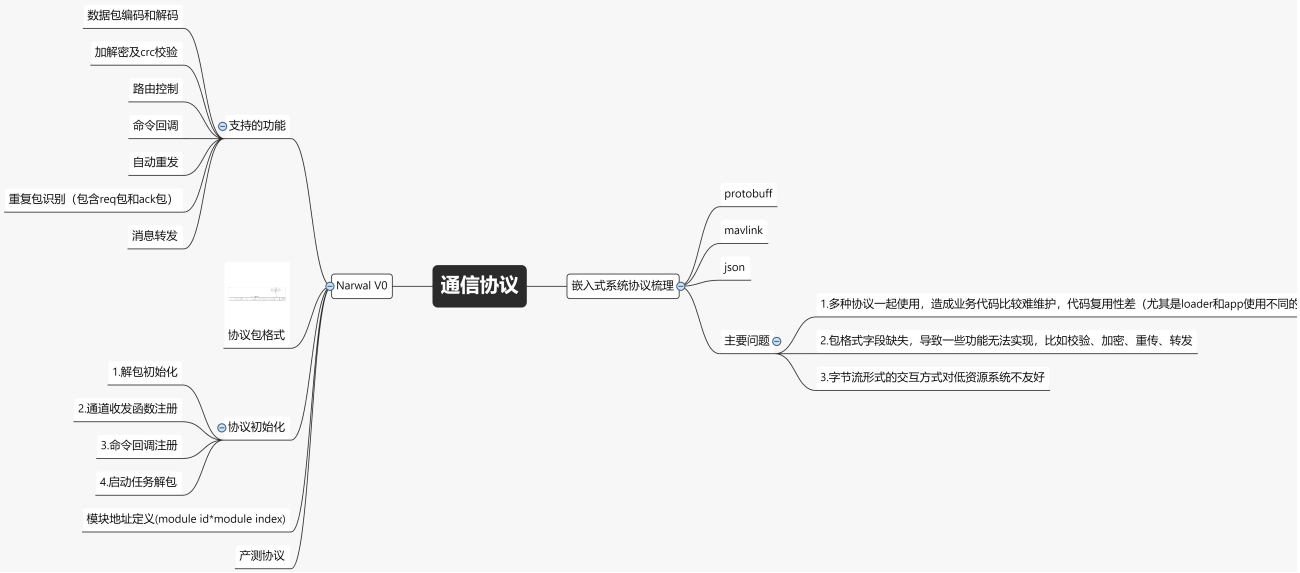


2 事件的实现机制



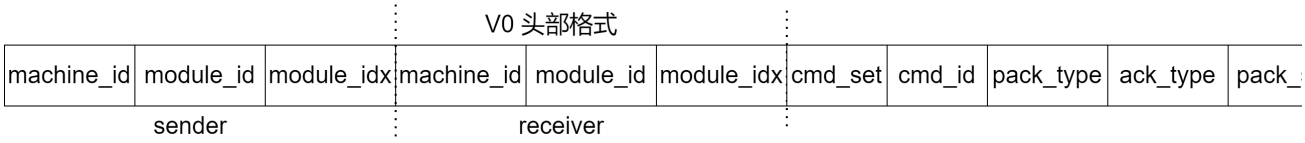
熟悉product_test节点

一 通信协议 V0



v0协议包（proto_pack_t）格式：

v0协议包（proto_pack_t）头部格式：

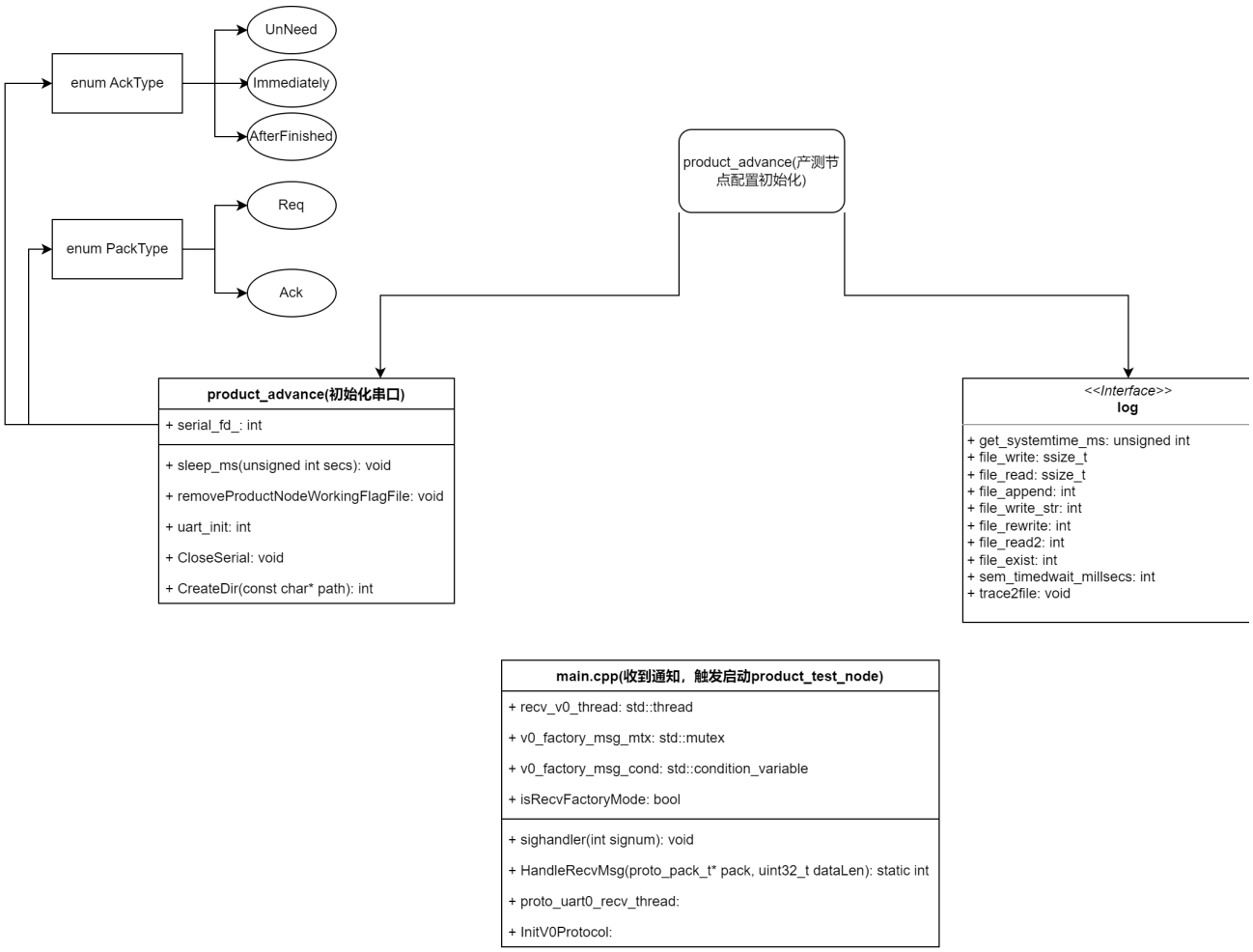


Kfifo
+ in_: uint64_t + out_: uint64_t + mask_: uint64_t 【使用与运算来确保索引的有效性】 + buffer_: ValType * + init_buffer_: std::atomic<bool> + in_mutex_: std::mutex + out_mutex_: std::mutex
+ Init(uint64_t element_size): bool + In(const ValType* data, uint64_t len): uint64_t + Out(ValType* data, uint64_t len): uint64_t + InWithLock(const ValType* data, uint64_t len): uint64_t + OutWithLock(ValType* data, uint64_t len): uint64_t + Unused(): uint64_t + Reset(): void **+ CopyIn(const ValType* data, uint64_t len): void** **+ CopyOut(ValType* data, uint64_t len): void** + IsFull(): bool + IsEmpty(): bool

二 产测节点代码结构（product_advance、product_test）

1 product_advance(主要作用是来接受V0协议数据包，如果当收到进入工厂模式的消息时则通知启动product_test node后退出)

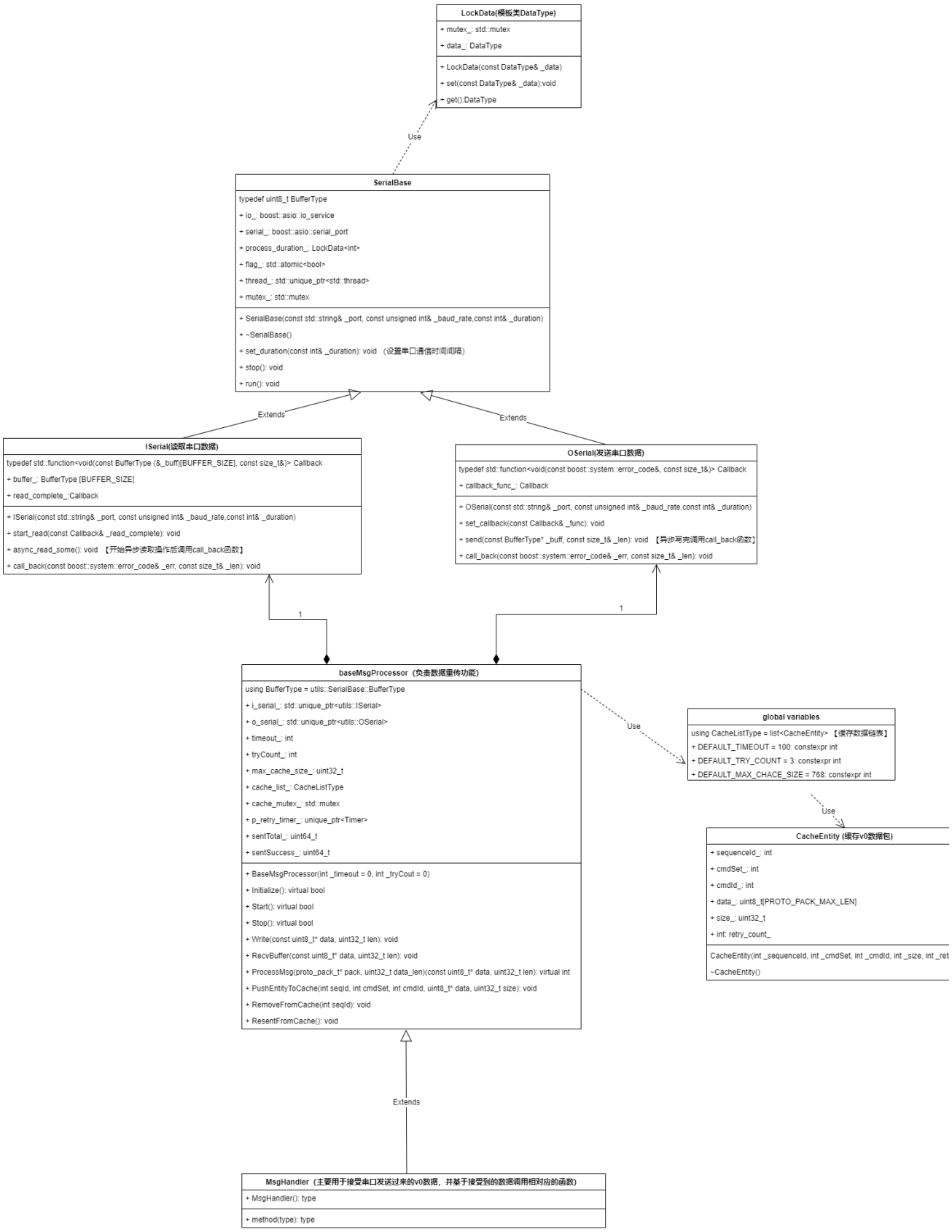
a 结构图



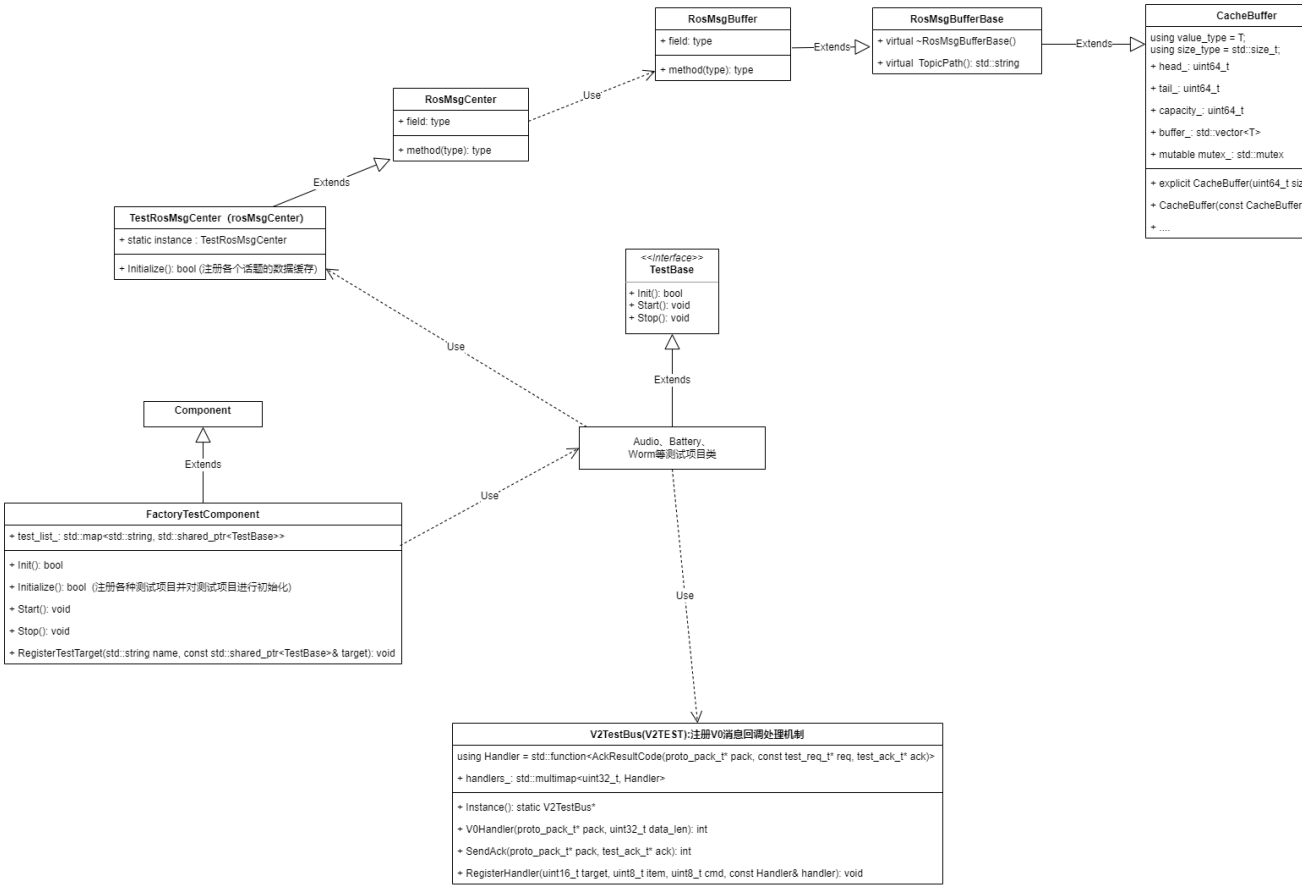
b UART协议的数据包格式：

2 product_test

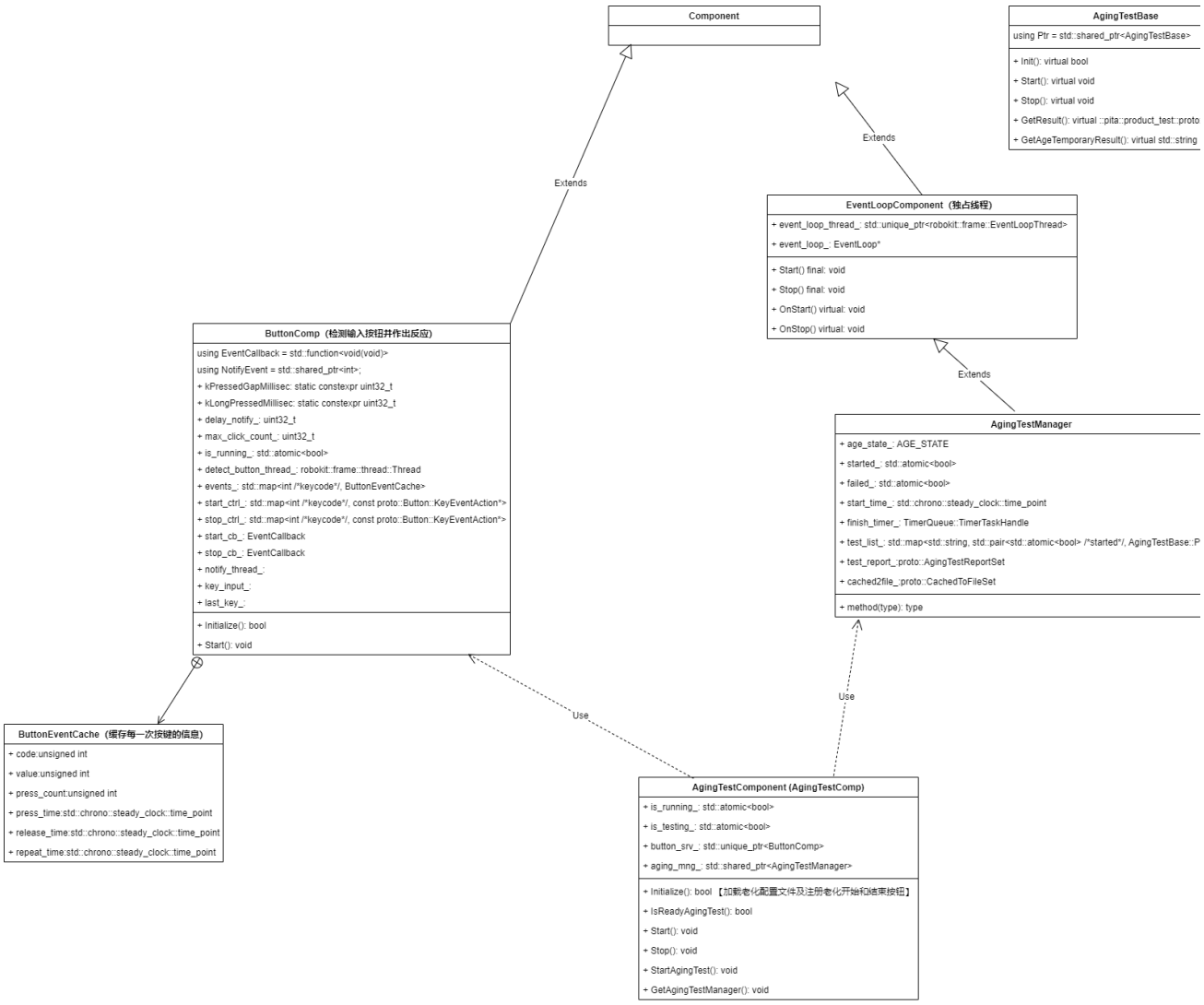
0 Serial串口通信模块



a FactoryTestComponent



b AgingTestComponent



c DataTransfer 【负责串口通讯并处理收到v0数据】

熟悉sysmonitor节点

一 sysmonitor使用手册

1 语法格式

```
sysmonitor [options] [interval]
```

2 参数解析

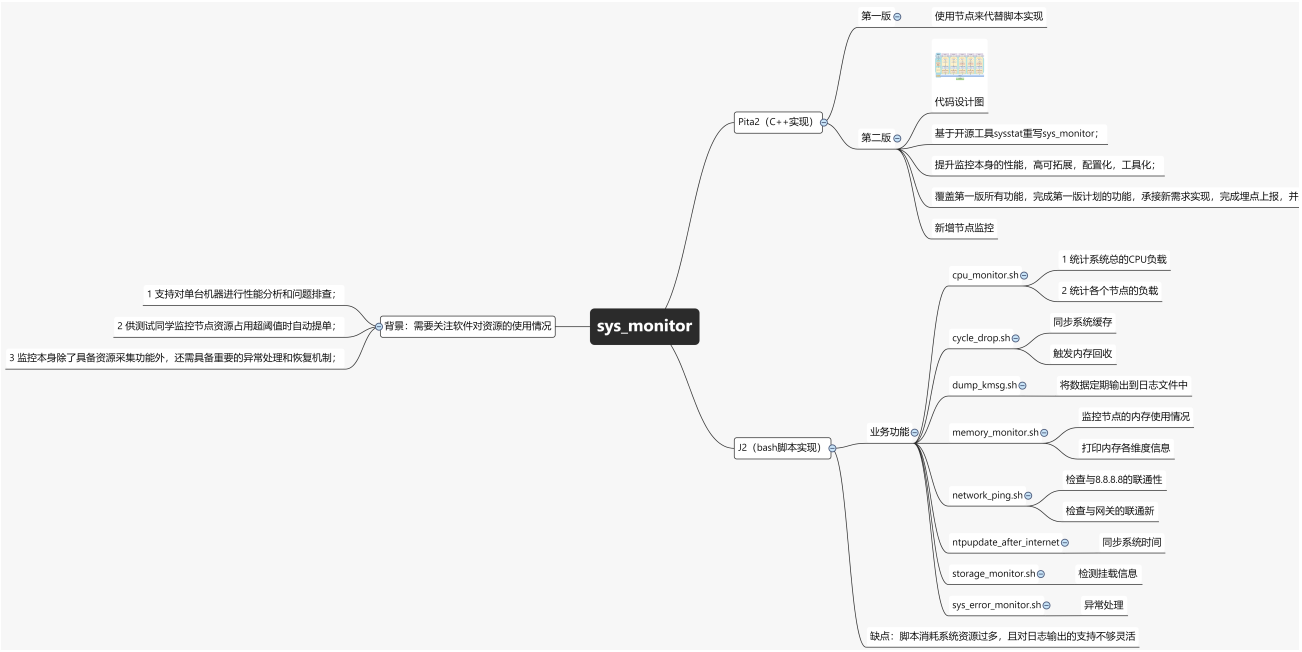
a. 可单独使用的参数

参数	功能
-h	打印使用手册
-v	打印版本信息
-H	转换为人类可读单位:K M G，默认单位为KB
-b	获取启动信息以及关机原因
-c/u	打印cpu信息
-m/r	打印内存信息
-d	打印磁盘信息
-N	打印所有节点信息，默认按CPU排序，可跟加参数-s按内存值进行排序
-n	暂不支持
-a	等于参数-HcmdNn
-A	等于参数-HcmdNnM
-P <pid>	监控进程
-p <name>	监控进程

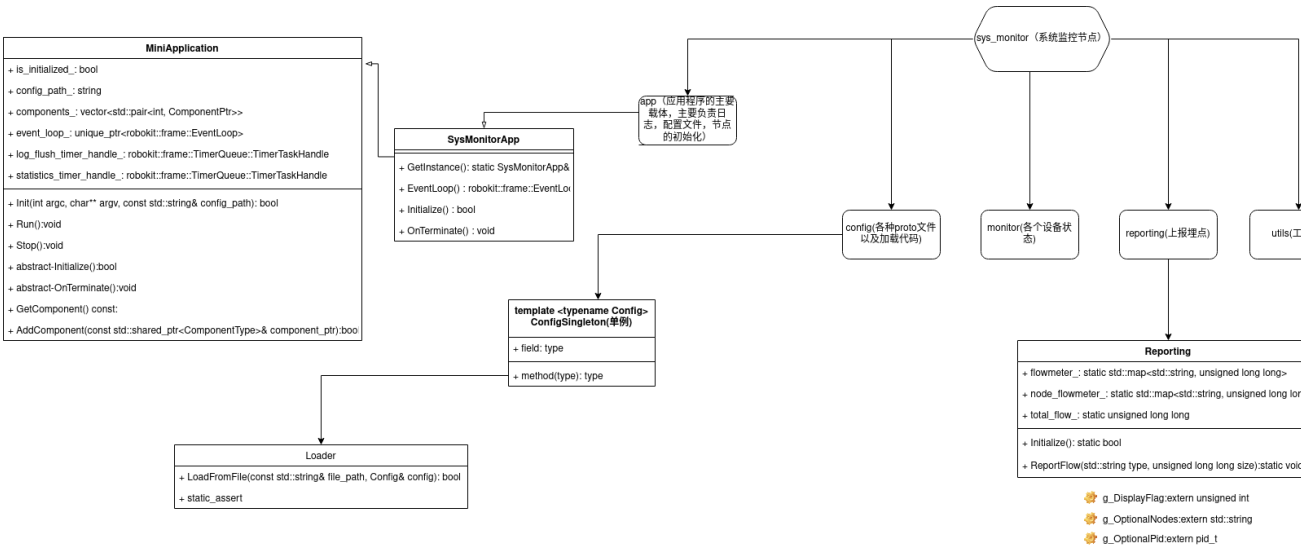
b. 不可单独使用的参数

参数	功能
-t	打印每个进程的线程信息
-R	上报数据
-s	按内存值进行排序

二 熟悉sysmonitor


sys_monitor
背景：需要关注软件对资源的使用情况
1 支持对单台机器进行性能分析和问题排查；
2 供测试同学监控节点资源占用超阈值时自动提单；
3 监控本身除了具备资源采集功能外，还需具备重要的异常处理和恢复机制；
Pita2（C++实现）
第一版
使用节点来代替脚本实现
第二版
代码设计图
基于开源工具sysstat重写sys_monitor；
提升监控本身的性能，高可拓展，配置化，工具化；
覆盖第一版所有功能，完成第一版计划的功能，承接新需求实现，完成埋点上报，并
新增节点监控
J2（bash脚本实现）
业务功能
cpu_monitor.sh
1 统计系统总的CPU负载
2 统计各个节点的负载
cycle_drop.sh
同步系统缓存
触发内存回收
dump_kmsg.sh
将数据定期输出到日志文件中
memory_monitor.sh
监控节点的内存使用情况
打印内存各维度信息
network_ping.sh
检查与8.8.8.8的联通性
检查与网关的联通新
ntpupdate_after_internet
同步系统时间
storage_monitor.sh
检测挂载信息
sys_error_monitor.sh
异常处理
缺点：脚本消耗系统资源过多，且对日志输出的支持不够灵活

三 业务架构图


sys_monitor（系统监控节点）
app（应用程序的主要载体，主要负责日志，配置文件，节点的初始化）
MiniApplication
+ is_initialized_: bool
+ config_path_: string
+ components_: vector<std::pair<int, ComponentPtr>>
+ event_loop_: unique_ptr<robokit::frame::EventLoop>
+ log_flush_timer_handle_: robokit::frame::TimerQueue::TimerTaskHandle
+ statistics_timer_handle_: robokit::frame::TimerQueue::TimerTaskHandle
+ Init(int argc, char** argv, const std::string& config_path): bool
+ Run():void
+ Stop():void
+ abstract-Initialize():bool
+ abstract-OnTerminate():void
+ GetComponent() const:
+ AddComponent(const std::shared_ptr<ComponentType>& component_ptr):boo
SysMonitorApp
+ GetInstance(): static SysMonitorApp&
+ EventLoop() : robokit::frame::EventLo
+ Initialize() : bool
+ OnTerminate() : void
config(各种proto文件以及加载代码)
monitor(各个设备状态)
reporting(上报埋点)
utils(工
template <typename Config>
ConfigSingleton(单例)
+ field: type
+ method(type): type
Loader
+ LoadFromFile(const std::string& file_path, Config& config): bool
+ static_assert
Reporting
+ flowmeter_: static std::map<std::string, unsigned long long>
+ node_flowmeter_: static std::map<std::string, unsigned long lor
+ total_flow_: static unsigned long long
+ Initialize(): static bool
+ ReportFlow(std::string type, unsigned long long size):static voic
g_DisplayFlag:extern unsigned int
g_OptionalNodes:extern std::string
g_OptionalPid:extern pid_t

常用指令

01. 入职实践

sys_monitor节点的功能优化

一 对sysmonitor节点增加可按cpu/mem/rss使用率进行排序显示的功能；

1 参数语法： -s [cpu/mem/rss] （默认不排序）

```
   -s <cpu/mem/rss>  Sort the monitored data by cpu/mem/rss fields
```

2 用例：

① 统计各个节点

a. 统计各个节点的数据，按cpu从高到底排序：

```
root@pita2_mr813_tina35:/userdata/logs/sys_monitor# sysmonitor -Ns cpu
sampling_interval is 2 s
# Time    UID      TGID    STATE    %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:59:22    0      2207        S    8.54   11.27   19.81   1  2779176   105852   10.56       2.22      0.41      72  154  planning_node
16:59:22    0      2215        S    6.36    5.33   11.68   1  2883044   106832   10.66      12.20      0.45      84  107  application_
16:59:22    0      2238        S    6.27    3.20    9.47   0  2803352    33608    3.35       2.06      0.18      90  112  toolkit_recor
16:59:22    0      2200        S    4.75    4.24    9.00   2  2511576    74536    7.44       2.14      0.20      44   71  mapping_node
16:59:22    0      2179        S    3.75    4.29    8.03   2  2518660    26700    2.66       2.10      0.11      51   98  hal
16:59:22    0      2193        S    3.88    2.36    6.24   2  2521484    28540    2.85       2.04      0.12      54   67  perception_no
16:59:22    0      2186        S    4.98    1.08    6.05   3  2299192    35672    3.56       1.85      0.19      37   65  map_server
16:59:22    0      2222        S    1.47    1.58    3.05   2  2536424    20860    2.08       2.03      0.18      59   54  wireless_comm
16:59:22    0      2230        S    0.93    0.64    1.57   0  2217152    22968    2.29       2.05      0.06      35   37  upgrader_node
16:59:22    0      2245        S    0.80    0.44    1.24   0  2434188    20060    2.00       0.38      0.03      46   46  robokit_teler
16:59:22    0      2172        S    0.44    0.45    0.89   3  1991644    17496    1.75       2.03      0.05      33   53  power_health
16:59:22    0     12223        S    0.04    0.14    0.17   2   182568    11880    1.19       0.22      0.10       5   13  sys_monitor
16:59:22    0      2253        S    0.00    0.00    0.00   1    15868     9328    0.93       0.02      0.01       1    3  update_runnin
16:59:22    0     17514        R    0.00    0.00    0.00   2   181920    11204    1.12       0.00      0.00       5   15  sys_monitor
16:59:22    0      2231        S    0.00    0.00    0.00   0    25580     8880    0.89       0.00      0.01       2    5  product_advar
```

b. 统计各个节点的数据，按mem从高到底排序：

```
root@pita2_mr813_tina35:/userdata/logs/sys_monitor# sysmonitor -Ns mem
sampling_interval is 2 s
# Time    UID      TGID    STATE    %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:55:02    0      2215        S    6.36    5.33   11.69   3  2883044   106832   10.66      12.20      0.48      84  107  application_
16:55:02    0      2207        S    8.53   11.28   19.81   3  2779176   105852   10.56       2.23      0.43      72  154  planning_nod
16:55:02    0      2200        S    4.76    4.24    9.00   2  2511576    74536    7.44       2.15      0.21      44   71  mapping_node
16:55:02    0      2186        S    5.05    1.06    6.12   3  2299192    35672    3.56       1.85      0.20      37   65  map_server
16:55:02    0      2238        S    6.27    3.21    9.48   0  2803352    33608    3.35       2.06      0.19      90  112  toolkit_rec
16:55:02    0      2193        S    3.89    2.36    6.25   3  2521484    28540    2.85       2.04      0.12      54   67  perception_n
16:55:02    0      2179        S    3.78    4.31    8.09   0  2518660    26700    2.66       2.10      0.12      51   98  hal
16:55:02    0      2230        S    0.93    0.64    1.57   1  2217152    22968    2.29       2.05      0.07      35   37  upgrader_no
16:55:02    0      2222        S    1.48    1.58    3.05   0  2536424    20860    2.08       2.03      0.19      59   54  wireless_co
16:55:02    0      2245        S    0.79    0.44    1.22   0  2434188    20060    2.00       0.38      0.03      46   46  robokit_tel
16:55:02    0      2172        S    0.44    0.45    0.89   3  1991644    17496    1.75       2.04      0.05      33   53  power_healt
16:55:02    0     12223        S    0.02    0.08    0.10   0   182568    11880    1.19       0.13      0.06       5   13  sys_monitor
16:55:02    0     15156        R    0.00    0.00    0.00   2   181920    11108    1.11       0.00      0.00       5   15  sys_monitor
16:55:02    0      2253        S    0.00    0.00    0.00   1    15868     9328    0.93       0.02      0.01       1    3  update_runn
16:55:02    0      2231        S    0.00    0.00    0.00   0    25580     8880    0.89       0.00      0.01       2    5  product_adv
```

c. 统计各个节点的数据，按rss从高到底排序：

```
root@pita2_mr813_tina35:/userdata/logs/sys_monitor# sysmonitor -Ns rss
sampling_interval is 2 s
# Time    UID      TGID    STATE    %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
17:00:34    0      2215        S    6.36    5.33   11.68   0  2883044   106832   10.66      12.20      0.45      84  107  application_
17:00:34    0      2207        S    8.53   11.27   19.80   3  2779176   105852   10.56       2.22      0.41      72  154  planning_node
17:00:34    0      2200        S    4.75    4.24    8.99   3  2511576    74536    7.44       2.14      0.20      44   71  mapping_node
17:00:34    0      2186        S    4.96    1.08    6.04   2  2299192    35672    3.56       1.85      0.19      37   65  map_server
17:00:34    0      2238        S    6.27    3.20    9.47   1  2803352    33608    3.35       2.06      0.18      90  112  toolkit_recor
17:00:34    0      2193        S    3.88    2.36    6.24   3  2521484    28540    2.85       2.04      0.12      54   67  perception_no
17:00:34    0      2179        S    3.74    4.28    8.02   1  2518660    26700    2.66       2.10      0.11      51   98  hal
17:00:34    0      2230        S    0.93    0.64    1.57   1  2217152    22968    2.29       2.05      0.06      35   37  upgrader_node
17:00:34    0      2222        S    1.47    1.58    3.05   1  2536424    20860    2.08       2.03      0.18      59   54  wireless_comm
17:00:34    0      2245        S    0.80    0.44    1.24   1  2434188    20060    2.00       0.38      0.03      46   46  robokit_teler
17:00:34    0      2172        S    0.44    0.45    0.89   1  1991644    17496    1.75       2.03      0.05      33   53  power_health
17:00:34    0     12223        S    0.04    0.15    0.19   3   182568    11880    1.19       0.25      0.10       5   13  sys_monitor
17:00:34    0     18193        R    0.00    0.00    0.00   0   181920    11104    1.11       0.00      0.00       5   15  sys_monitor
17:00:34    0      2253        S    0.00    0.00    0.00   1    15868     9328    0.93       0.02      0.01       1    3  update_runnin
17:00:34    0      2231        S    0.00    0.00    0.00   0    25580     8880    0.89       0.00      0.01       2    5  product_advar
```

② 统计各个节点各个线程

a 按cpu值排序

```
root@pita2_mr813_tina35:/userdata/logs/sys_monitor# sysmonitor -tP hal -s cpu
sampling_interval is 2 s
[ .595][I][node][NodeStat.cpp:255][Init][20164] monitor nodes: hal
# Time   UID      TGID       TID    %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
17:04:08   0      2179         S    3.72   4.26   7.97   3 2518660   26700   2.66      2.09      0.11      51   98  hal
17:04:08   0         S      2344    0.60   0.33   0.93   0 2518660   26700   2.66     19.33      0.64      51   98  |__hal
17:04:08   0         S      2270    0.20   0.68   0.87   1 2518660   26700   2.66     15.94      0.74      51   98  |__hal
17:04:08   0         S      2740    0.20   0.36   0.55   3 2518660   26700   2.66    105.69      0.44      51   98  |__T:HalPublish
17:04:08   0         S      3036    0.29   0.18   0.46   0 2518660   26700   2.66     50.85      0.23      51   98  |__hal
17:04:08   0         S      2735    0.22   0.24   0.46   3 2518660   26700   2.66     63.25      0.35      51   98  |__0_CommComp
17:04:08   0         S      2739    0.22   0.24   0.46   0 2518660   26700   2.66     63.19      0.38      51   98  |__4_CommComp
17:04:08   0         S      2738    0.22   0.24   0.46   0 2518660   26700   2.66     62.53      0.38      51   98  |__3_CommComp
17:04:08   0         S      2736    0.21   0.25   0.46   3 2518660   26700   2.66     63.12      0.33      51   98  |__1_CommComp
17:04:08   0         S      2737    0.22   0.24   0.45   1 2518660   26700   2.66     63.29      0.34      51   98  |__2_CommComp
17:04:08   0         S      2531    0.24   0.15   0.39   1 2518660   26700   2.66     98.91      2.58      51   98  |__UnpackProtoc
17:04:08   0         S      2530    0.06   0.32   0.38   1 2518660   26700   2.66     98.49      0.23      51   98  |__SerialPortWr
17:04:08   0         S      2529    0.01   0.08   0.10   1 2518660   26700   2.66     31.47      1.10      51   98  |__SerialPortRe
17:04:08   0         S      2440    0.07   0.02   0.09   3 2518660   26700   2.66      5.01      0.04      51   98  |__HalPm
17:04:08   0         S      2754    0.06   0.03   0.08   0 2518660   26700   2.66     11.54      0.05      51   98  |__hal
17:04:08   0         S      2753    0.06   0.02   0.08   0 2518660   26700   2.66     11.54      0.06      51   98  |__hal
17:04:08   0         S      2751    0.07   0.02   0.08   3 2518660   26700   2.66     11.54      0.06      51   98  |__hal
17:04:08   0         S      2752    0.06   0.02   0.08   3 2518660   26700   2.66     11.55      0.05      51   98  |__hal
17:04:08   0         S      2755    0.07   0.02   0.08   1 2518660   26700   2.66     11.54      0.06      51   98  |__hal
17:04:08   0         S      2734    0.08   0.00   0.08   0 2518660   26700   2.66      9.96      0.08      51   98  |__T:sta_contac
17:04:08   0         S      2505    0.05   0.02   0.07   3 2518660   26700   2.66      7.96      0.05      51   98  |__EL:heartbeat
17:04:08   0         S      2741    0.03   0.02   0.05   0 2518660   26700   2.66     14.93      0.07      51   98  |__0_SensorHub
17:04:08   0         S      2745    0.03   0.02   0.05   0 2518660   26700   2.66     14.91      0.08      51   98  |__4_SensorHub
17:04:08   0         S      2744    0.03   0.02   0.05   1 2518660   26700   2.66     14.91      0.08      51   98  |__3_SensorHub
17:04:08   0         S      2743    0.03   0.02   0.05   3 2518660   26700   2.66     14.89      0.07      51   98  |__2_SensorHub
17:04:08   0         S      2742    0.03   0.02   0.05   0 2518660   26700   2.66     14.91      0.08      51   98  |__1_SensorHub
17:04:08   0         S      2503    0.02   0.02   0.04   1 2518660   26700   2.66      5.39      0.03      51   98  |__hal
17:04:08   0         S      2179    0.02   0.02   0.04   3 2518660   26700   2.66      2.09      0.11      51   98  |__hal
17:04:08   0         S      2271    0.02   0.02   0.04   0 2518660   26700   2.66      9.96      0.02      51   98  |__hal
17:04:08   0         S      2747    0.02   0.02   0.04   3 2518660   26700   2.66      9.97      0.01      51   98  |__hal
17:04:08   0         S      2748    0.01   0.02   0.03   0 2518660   26700   2.66     11.00      0.01      51   98  |__hal
17:04:08   0         S      2750    0.01   0.02   0.03   1 2518660   26700   2.66     10.93      0.01      51   98  |__hal
17:04:08   0         S      2746    0.02   0.02   0.03   2 2518660   26700   2.66     10.54      0.01      51   98  |__hal
17:04:08   0         S      2749    0.02   0.02   0.03   3 2518660   26700   2.66     10.24      0.02      51   98  |__hal
17:04:08   0         S      2346    0.01   0.02   0.03   0 2518660   26700   2.66      9.95      0.01      51   98  |__hal
17:04:08   0         S      2504    0.01   0.01   0.02   1 2518660   26700   2.66      1.99      0.00      51   98  |__EL:heartbeat
17:04:08   0         S      2430    0.01   0.01   0.02   3 2518660   26700   2.66      9.95      0.01      51   98  |__ShortPressCh
17:04:08   0         S      2534    0.00   0.01   0.01   1 2518660   26700   2.66      1.99      0.00      51   98  |__T:AppHeart
```

二 当节点CPU占用率超过阈值(默认40%)时，统计该节点CPU占用率前三的线程。

```
[2023-07-11 14:07:34.176][I][node][NodeStat.cpp:557][ReportStat][20847]{"name":"robokit_telemet","pid":2184,"state":"S","threads":46,"usr":0.35,"sys":0.14166
":0.49166","vsz":2434000,"rss":20156,"mem":2.01172,"cswch":0,"nvcswch":0,"fds":45,"topThreads":[{"name":"2_Upload thread","tid":2336,"state":"S","usr":0.058
"sys":0.00833333,"cpu":0.0666667,"cswch":2,"nvcswch":0},{"name":"1_Upload thread","tid":2335,"state":"S","usr":0.05,"sys":0.0166667,"cpu":0.0666667,"cswch":2
swch":0},{"name":"0_Attr thread p","tid":2333,"state":"S","usr":0.0583333,"sys":0.00833333,"cpu":0.0666667,"cswch":2,"nvcswch":0}],"eventId":2055,"time":1689
4176}
[2023-07-11 14:07:34.176][I][node][NodeStat.cpp:557][ReportStat][20847]{"name":"update_running_","pid":2191,"state":"S","threads":1,"usr":0,"sys":0,"cpu":0,"
15868,"rss":9552,"mem":0.953362,"cswch":0,"nvcswch":0,"fds":3,"topThreads":[],"eventId":2055,"time":1689055654176}
[2023-07-11 14:07:34.176][I][node][NodeStat.cpp:557][ReportStat][20847]{"name":"sys_monitor","pid":20847,"state":"R","threads":5,"usr":0.075,"sys":0.308333,"
0.383333,"vsz":182604,"rss":11792,"mem":1.17693,"cswch":2,"nvcswch":1,"fds":15,"topThreads":[],"eventId":2055,"time":1689055654176}
```

三 当超过阈值时会根据MAX_REPORT_TIMES来调整上报次数，而不是一直上报。

四 增加功能后性能对比

1 增加排序功能

a 没排序前：

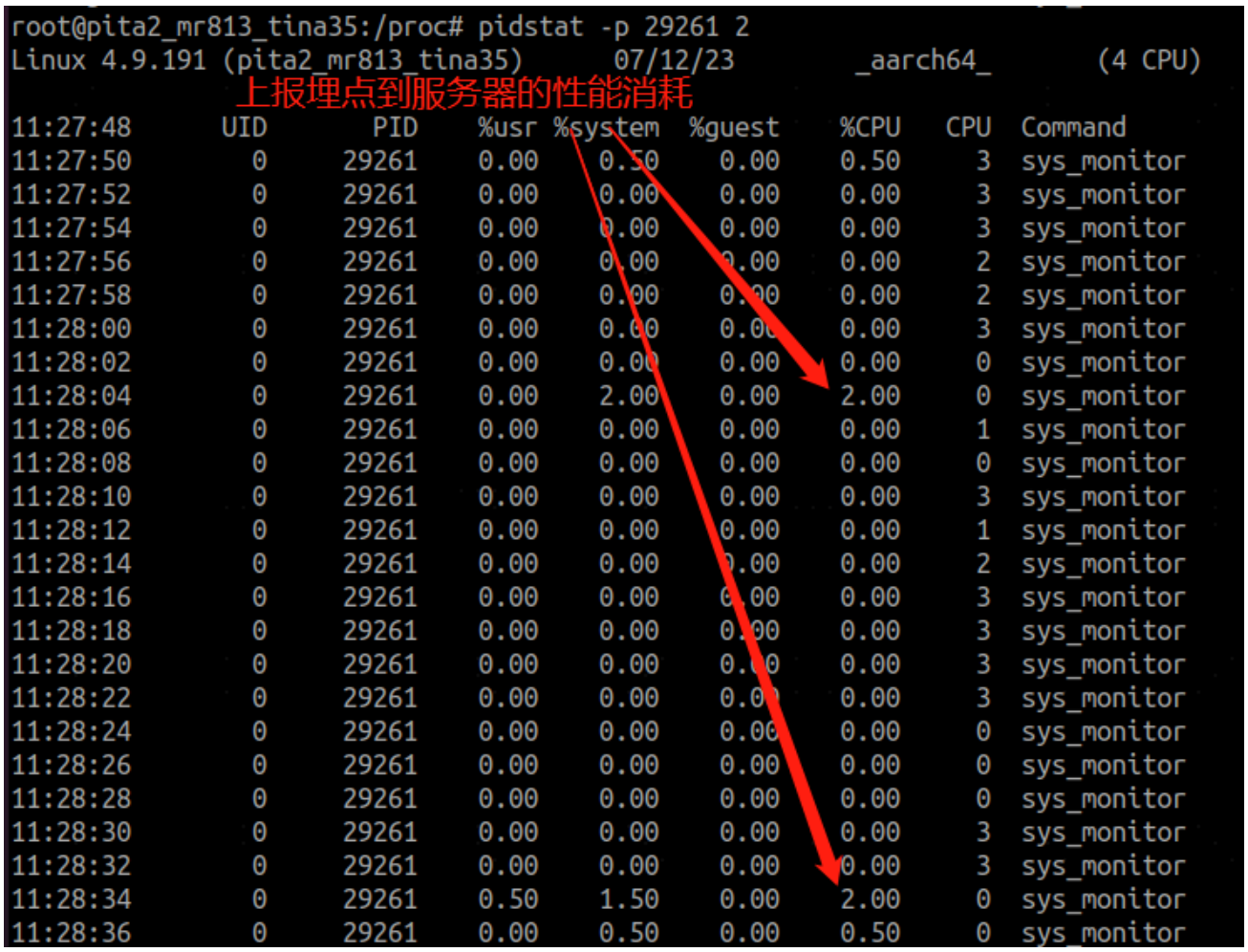
```
root@pita2_mr813_tina35:/proc# pidstat -p 2545 2
Linux 4.9.191 (pita2_mr813_tina35)      07/12/23        _aarch64_       (4 CPU)

10:38:30      UID       PID    %usr %system  %guest    %CPU   CPU  Command
10:38:32        0      2545    0.50    1.50    0.00    2.00     1  sys_monitor
10:38:34        0      2545    0.00    2.00    0.00    2.00     1  sys_monitor
10:38:36        0      2545    0.50    2.00    0.00    2.50     2  sys_monitor
10:38:38        0      2545    0.50    1.50    0.00    2.00     3  sys_monitor
10:38:40        0      2545    0.00    2.00    0.00    2.00     2  sys_monitor
```

b 排序后：

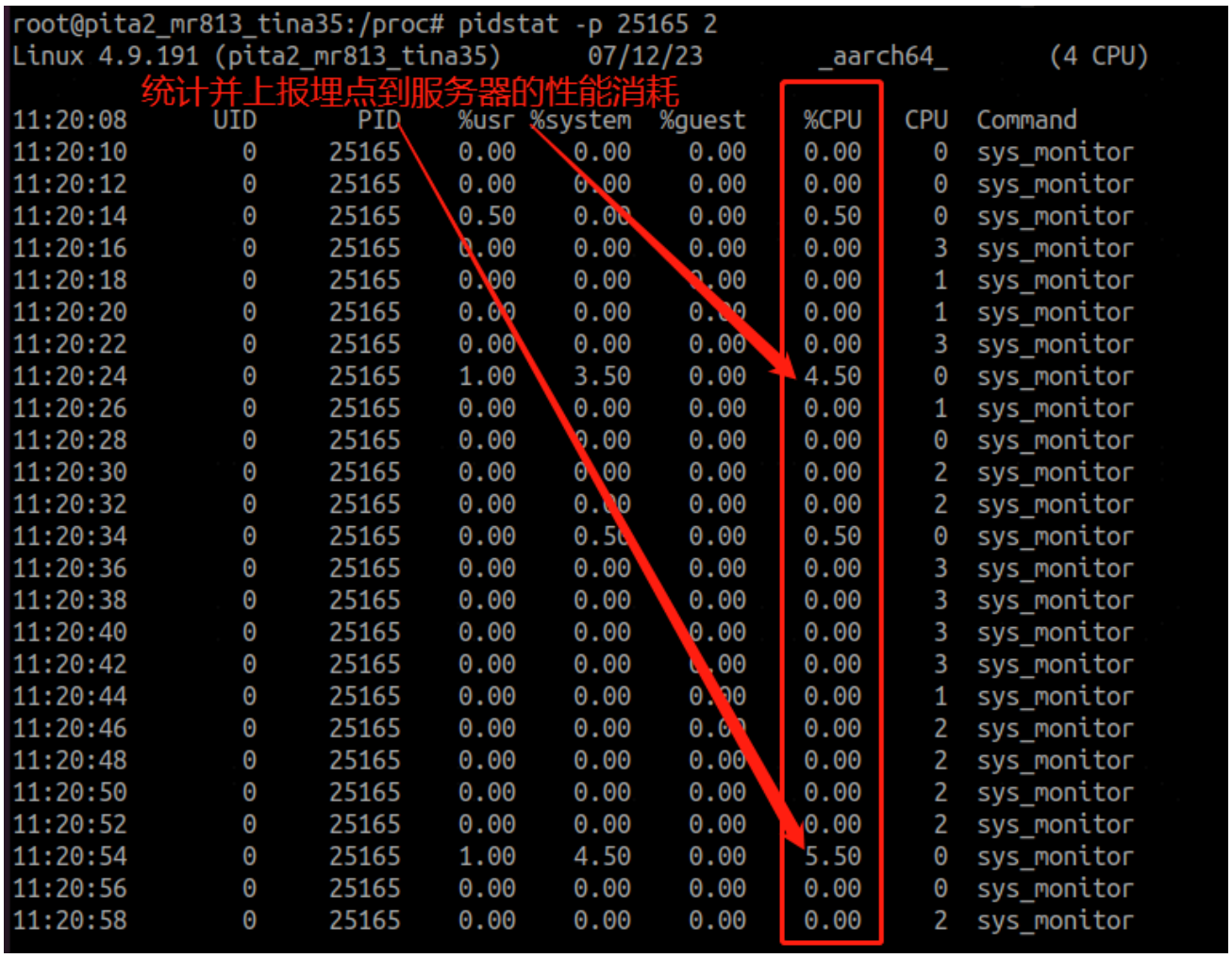
```
root@pita2_mr813_tina35:/proc# pidstat -p 3523 2
Linux 4.9.191 (pita2_mr813_tina35)      07/12/23        _aarch64_       (4 CPU)

10:39:38      UID       PID    %usr %system  %guest    %CPU   CPU  Command
10:39:40        0      3523    0.00    1.99    0.00    1.99     1  sys_monitor
10:39:42        0      3523    1.00    2.00    0.00    3.00     1  sys_monitor
10:39:44        0      3523    0.00    1.50    0.00    1.50     1  sys_monitor
10:39:46        0      3523    0.50    1.50    0.00    2.00     1  sys_monitor
10:39:48        0      3523    0.50    2.00    0.00    2.50     1  sys_monitor
10:39:50        0      3523    0.50    1.00    0.00    1.50     3  sys_monitor
10:39:52        0      3523    0.50    2.00    0.00    2.50     0  sys_monitor
10:39:54        0      3523    0.00    2.00    0.00    2.00     0  sys_monitor
10:39:56        0      3523    0.00    2.00    0.00    2.00     0  sys_monitor
10:39:58        0      3523    0.50    2.00    0.00    2.50     1  sys_monitor
```

2 增加统计节点CPU占用率前三的线程。

a 没超过阈值时



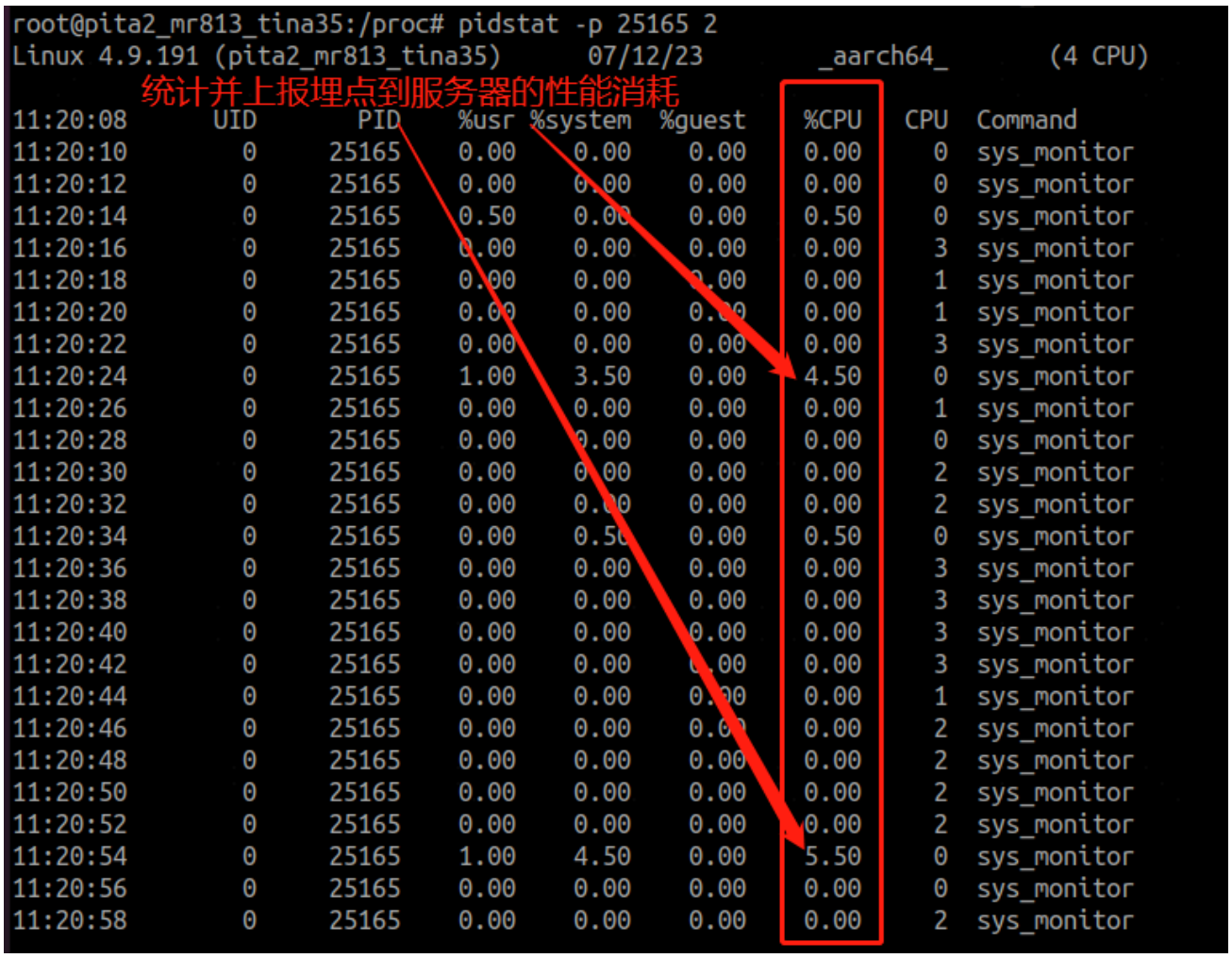
b 超过阈值时(性能消耗多少取决于超过阈值的节点数)



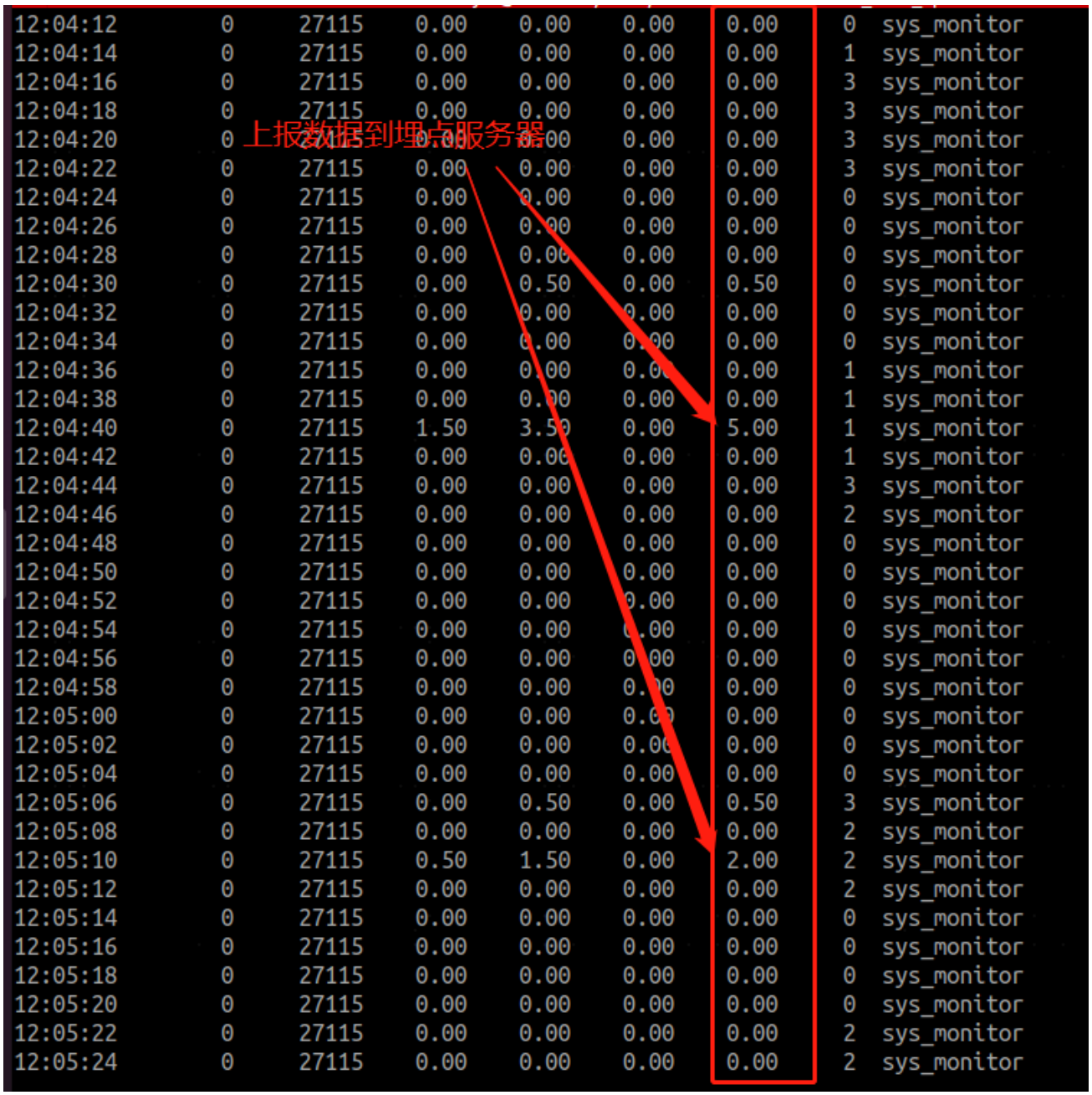
3当超过阈值时会根据MAX_REPORT_TIMES来调整上报次数，而不是一直上报。

注：当某个节点超过阈值时，会累计上报次数Count，当达到 MAX_REPORT_TIMES时不再上报；如果该节点不再超过阈值，则对Count减1；

a.没调整上报次数时：


root@pita2_mr813_tina35:/proc# pidstat -p 25165 2
Linux 4.9.191 (pita2_mr813_tina35) 07/12/23 _aarch64_ (4 CPU)
统计并上报埋点到服务器的性能消耗
11:20:08 UID PID %usr %system %guest %CPU CPU Command
11:20:10 0 25165 0.00 0.00 0.00 0.00 0 sys_monitor
11:20:12 0 25165 0.00 0.00 0.00 0.00 0 sys_monitor
11:20:14 0 25165 0.50 0.00 0.00 0.50 0 sys_monitor
11:20:16 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:18 0 25165 0.00 0.00 0.00 0.00 1 sys_monitor
11:20:20 0 25165 0.00 0.00 0.00 0.00 1 sys_monitor
11:20:22 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:24 0 25165 1.00 3.50 0.00 4.50 0 sys_monitor
11:20:26 0 25165 0.00 0.00 0.00 0.00 1 sys_monitor
11:20:28 0 25165 0.00 0.00 0.00 0.00 0 sys_monitor
11:20:30 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:32 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:34 0 25165 0.00 0.50 0.00 0.50 0 sys_monitor
11:20:36 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:38 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:40 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:42 0 25165 0.00 0.00 0.00 0.00 3 sys_monitor
11:20:44 0 25165 0.00 0.00 0.00 0.00 1 sys_monitor
11:20:46 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:48 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:50 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:52 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor
11:20:54 0 25165 1.00 4.50 0.00 5.50 0 sys_monitor
11:20:56 0 25165 0.00 0.00 0.00 0.00 0 sys_monitor
11:20:58 0 25165 0.00 0.00 0.00 0.00 2 sys_monitor

b 调整上报次数后



1. 统计每个进程打开的文件和目录
2. 统计所有的网络连接以及支持显示指定端口上的网络连接

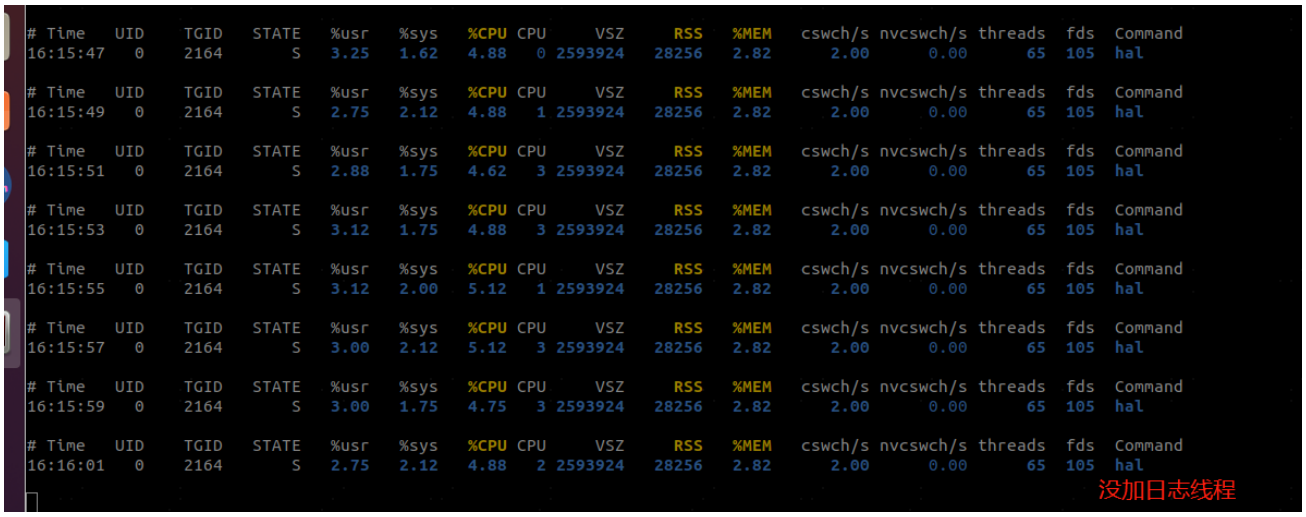
分离Hal节点中对mcu的log写操作

> 问题：对mcu发给hal节点的传感器，设备和日志等信息不加区分，从而当高IO和高Cpu loading下，因对mcu日志文件的写操作阻塞导致了其它节点无法及时得到数据。（
>
> IP1221-8269 - 【组件测试】【OT2+V01.00.01.00】【冒烟测试】先扫后拖，机器人拖地弓形时突然停顿一下 关闭
>
> ）

key：解包线程写日志操作改为写缓存，时间效率提升了100～1000倍左右，总体进程资源消耗相差不大。

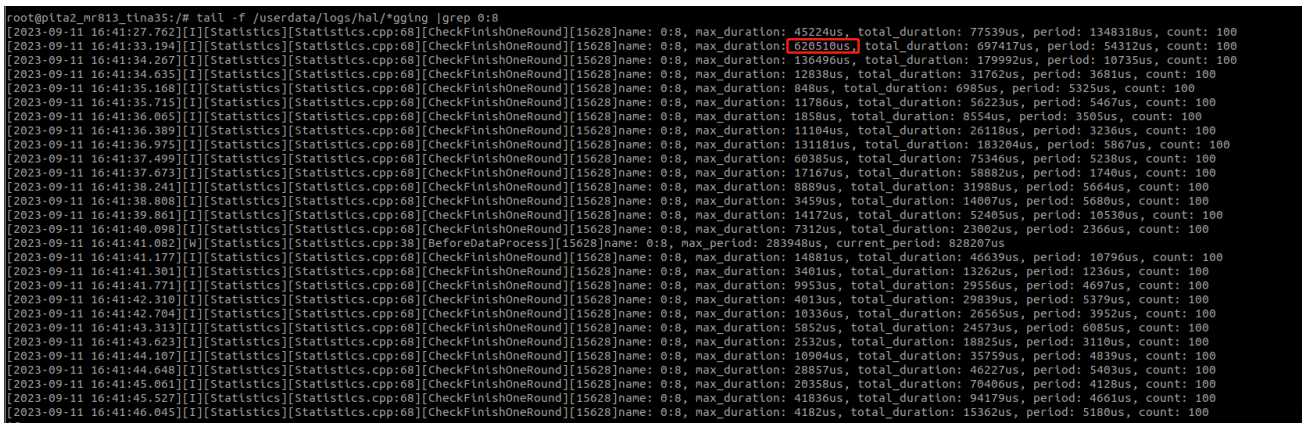
一 分离前：

1 节点资源消耗：



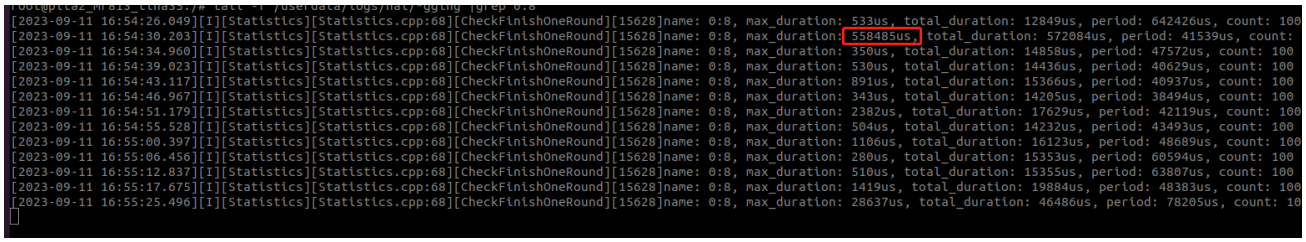

```
# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:47    0     2164        S   3.25    1.62    4.88   0 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:49    0     2164        S   2.75    2.12    4.88   1 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:51    0     2164        S   2.88    1.75    4.62   3 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:53    0     2164        S   3.12    1.75    4.88   3 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:55    0     2164        S   3.12    2.00    5.12   1 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:57    0     2164        S   3.00    2.12    5.12   3 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:15:59    0     2164        S   3.00    1.75    4.75   3 2593924    28256    2.82       2.00      0.00      65  105  hal

# Time    UID     TGID    STATE   %usr    %sys    %CPU CPU      VSZ      RSS    %MEM    cswch/s nvcswch/s threads  fds  Command
16:16:01    0     2164        S   2.75    2.12    4.88   2 2593924    28256    2.82       2.00      0.00      65  105  hal
```

2 模拟高IO和高CPU loading 的情况下：（写日志操作时间最高达到600ms左右，JIAR单中最高能达到1.5s）



```
root@pita2_mr813_tina35:/# tail -f /userdata/logs/hal/*gging |grep 0:8
[2023-09-11 16:41:27.762][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 45224us, total_duration: 77539us, period: 1348318us, count: 100
[2023-09-11 16:41:33.194][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 620510us, total_duration: 697417us, period: 54312us, count: 100
[2023-09-11 16:41:34.267][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 136496us, total_duration: 179992us, period: 10735us, count: 100
[2023-09-11 16:41:34.635][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 12838us, total_duration: 31762us, period: 3681us, count: 100
[2023-09-11 16:41:35.168][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 848us, total_duration: 6985us, period: 5325us, count: 100
[2023-09-11 16:41:35.715][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 11786us, total_duration: 56223us, period: 5467us, count: 100
[2023-09-11 16:41:36.065][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 1858us, total_duration: 8554us, period: 3505us, count: 100
[2023-09-11 16:41:36.389][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 11104us, total_duration: 26118us, period: 3236us, count: 100
[2023-09-11 16:41:36.975][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 131181us, total_duration: 183204us, period: 5867us, count: 100
[2023-09-11 16:41:37.499][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 60385us, total_duration: 75346us, period: 5238us, count: 100
[2023-09-11 16:41:37.673][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 17167us, total_duration: 58882us, period: 1740us, count: 100
[2023-09-11 16:41:38.241][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 8889us, total_duration: 31988us, period: 5664us, count: 100
[2023-09-11 16:41:38.808][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 3459us, total_duration: 14007us, period: 5680us, count: 100
[2023-09-11 16:41:39.861][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 14172us, total_duration: 52405us, period: 10530us, count: 100
[2023-09-11 16:41:40.098][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 7312us, total_duration: 23002us, period: 2366us, count: 100
[2023-09-11 16:41:41.082][W][Statistics][Statistics.cpp:38][BeforeDataProcess][15628]name: 0:8, max_period: 283948us, current_period: 828207us
[2023-09-11 16:41:41.177][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 14881us, total_duration: 46639us, period: 10796us, count: 100
[2023-09-11 16:41:41.301][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 3401us, total_duration: 13262us, period: 1236us, count: 100
[2023-09-11 16:41:41.771][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 9953us, total_duration: 29556us, period: 4697us, count: 100
[2023-09-11 16:41:42.310][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 4013us, total_duration: 29839us, period: 5379us, count: 100
[2023-09-11 16:41:42.704][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 10336us, total_duration: 26565us, period: 3952us, count: 100
[2023-09-11 16:41:43.313][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 5852us, total_duration: 24573us, period: 6085us, count: 100
[2023-09-11 16:41:43.623][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 2532us, total_duration: 18825us, period: 3110us, count: 100
[2023-09-11 16:41:44.107][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 10904us, total_duration: 35759us, period: 4839us, count: 100
[2023-09-11 16:41:44.648][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 28857us, total_duration: 46227us, period: 5403us, count: 100
[2023-09-11 16:41:45.061][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 20358us, total_duration: 70406us, period: 4128us, count: 100
[2023-09-11 16:41:45.527][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 41836us, total_duration: 94179us, period: 4661us, count: 100
[2023-09-11 16:41:46.045][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 4182us, total_duration: 15362us, period: 5180us, count: 100
```

3 模拟高IO情况下：（写日志操作时间最高达到500ms左右，JIAR单中最高能达到1.5s）



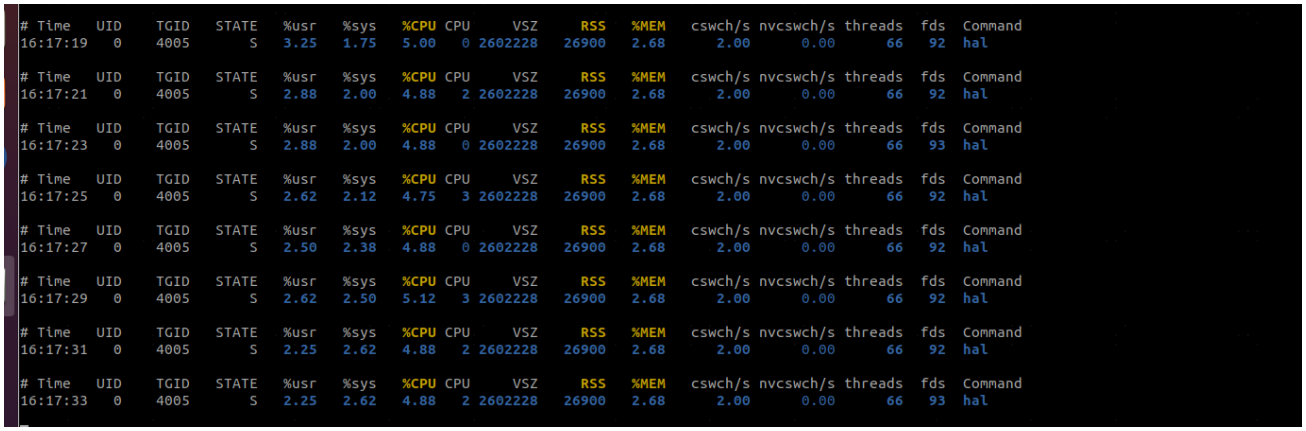
```
[2023-09-11 16:54:26.049][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 533us, total_duration: 12849us, period: 642426us, count: 100
[2023-09-11 16:54:30.203][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 558485us, total_duration: 572084us, period: 41539us, count:
[2023-09-11 16:54:34.960][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 350us, total_duration: 14858us, period: 47572us, count: 100
[2023-09-11 16:54:39.023][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 530us, total_duration: 14436us, period: 40629us, count: 100
[2023-09-11 16:54:43.117][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 891us, total_duration: 15366us, period: 40937us, count: 100
[2023-09-11 16:54:46.967][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 343us, total_duration: 14205us, period: 38494us, count: 100
[2023-09-11 16:54:51.179][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 2382us, total_duration: 17629us, period: 42119us, count: 100
[2023-09-11 16:54:55.528][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 504us, total_duration: 14232us, period: 43493us, count: 100
[2023-09-11 16:55:00.397][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 1106us, total_duration: 16123us, period: 48689us, count: 100
[2023-09-11 16:55:06.456][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 280us, total_duration: 15353us, period: 60594us, count: 100
[2023-09-11 16:55:12.837][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 510us, total_duration: 15355us, period: 63807us, count: 100
[2023-09-11 16:55:17.675][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 1419us, total_duration: 19884us, period: 48383us, count: 100
[2023-09-11 16:55:25.496][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][15628]name: 0:8, max_duration: 28637us, total_duration: 46486us, period: 78205us, count: 10
```

二 分离后：

A:使用Kfifo 无锁队列实现，极端情况下有阻塞延时的风险；且无锁的实现需要确保内存的执行顺序（考虑内存屏障）

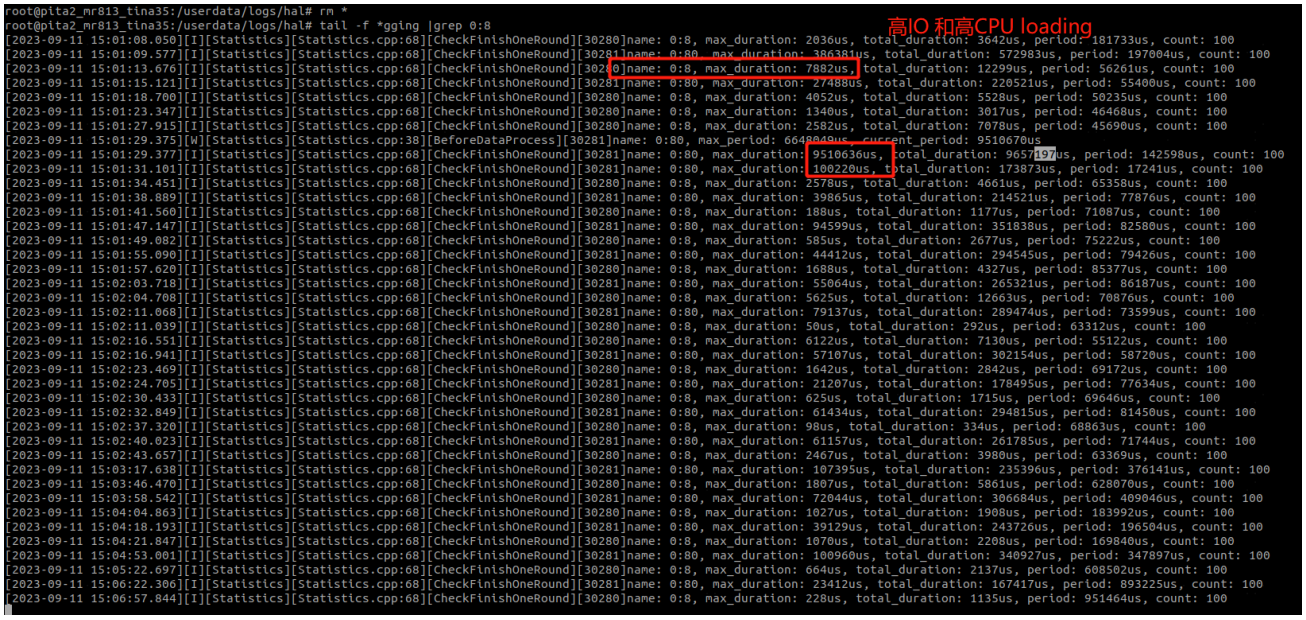
> *使用缓存大小：64KB*

1 节点资源消耗：



```
# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:19   0    4005       S   3.25   1.75   5.00   0 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:21   0    4005       S   2.88   2.00   4.88   2 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:23   0    4005       S   2.88   2.00   4.88   0 2602228   26900   2.68      2.00      0.00      66   93  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:25   0    4005       S   2.62   2.12   4.75   3 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:27   0    4005       S   2.50   2.38   4.88   0 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:29   0    4005       S   2.62   2.50   5.12   3 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:31   0    4005       S   2.25   2.62   4.88   2 2602228   26900   2.68      2.00      0.00      66   92  hal

# Time   UID    TGID   STATE   %usr   %sys   %CPU CPU     VSZ     RSS   %MEM   cswch/s nvcswch/s threads  fds  Command
16:17:33   0    4005       S   2.25   2.62   4.88   2 2602228   26900   2.68      2.00      0.00      66   93  hal
```

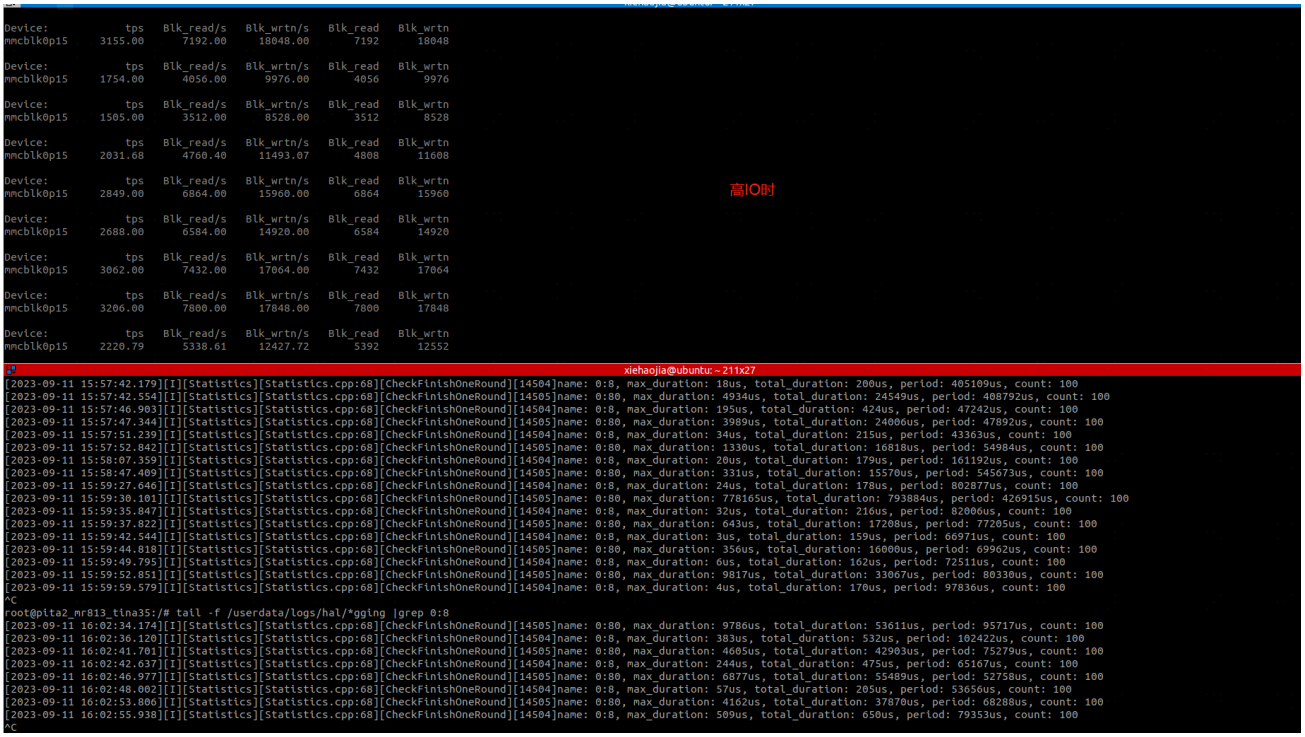
2 模拟高IO和高CPU loading 的情况下：（写日志操作时间最高达到9s左右，但解包线程写缓存时间，7882us）



```
root@pita2_mr813_tina35:/userdata/logs/hal# rm *
root@pita2_mr813_tina35:/userdata/logs/hal# tail -f *gging |grep 0:8
[2023-09-11 15:01:08.050][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 2036us, total_duration: 3642us, period: 181733us, count: 100
[2023-09-11 15:01:09.577][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 386381us, total_duration: 572983us, period: 197004us, count: 100
[2023-09-11 15:01:13.676][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][3028 0]name: 0:8, max_duration: 7882us, total_duration: 12299us, period: 56261us, count: 100
[2023-09-11 15:01:15.121][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 27488us, total_duration: 220521us, period: 55400us, count: 100
[2023-09-11 15:01:18.700][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 4052us, total_duration: 5528us, period: 50235us, count: 100
[2023-09-11 15:01:23.347][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1340us, total_duration: 3017us, period: 46468us, count: 100
[2023-09-11 15:01:27.915][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 2582us, total_duration: 7078us, period: 45690us, count: 100
[2023-09-11 15:01:29.375][W][Statistics][Statistics.cpp:38][BeforeDataProcess][30281]name: 0:80, max_period: 6648049us, current_period: 9510670us
[2023-09-11 15:01:29.377][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 9510636us, total_duration: 9657197us, period: 142598us, count: 100
[2023-09-11 15:01:31.101][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 100220us, total_duration: 173873us, period: 17241us, count: 100
[2023-09-11 15:01:34.451][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 2578us, total_duration: 4661us, period: 65358us, count: 100
[2023-09-11 15:01:38.889][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 39865us, total_duration: 214521us, period: 77876us, count: 100
[2023-09-11 15:01:41.560][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 188us, total_duration: 1177us, period: 71087us, count: 100
[2023-09-11 15:01:47.147][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 94599us, total_duration: 351838us, period: 82580us, count: 100
[2023-09-11 15:01:49.082][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 585us, total_duration: 2677us, period: 75222us, count: 100
[2023-09-11 15:01:55.090][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 44412us, total_duration: 294545us, period: 79426us, count: 100
[2023-09-11 15:01:57.620][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1688us, total_duration: 4327us, period: 85377us, count: 100
[2023-09-11 15:02:03.718][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 55064us, total_duration: 265321us, period: 86187us, count: 100
[2023-09-11 15:02:04.708][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 5625us, total_duration: 12663us, period: 70876us, count: 100
[2023-09-11 15:02:11.068][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 79137us, total_duration: 289474us, period: 73599us, count: 100
[2023-09-11 15:02:11.039][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 50us, total_duration: 292us, period: 63312us, count: 100
[2023-09-11 15:02:16.551][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 6122us, total_duration: 7130us, period: 55122us, count: 100
[2023-09-11 15:02:16.941][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 57107us, total_duration: 302154us, period: 58720us, count: 100
[2023-09-11 15:02:23.469][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1642us, total_duration: 2842us, period: 69172us, count: 100
[2023-09-11 15:02:24.705][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 21207us, total_duration: 178495us, period: 77634us, count: 100
[2023-09-11 15:02:30.433][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 625us, total_duration: 1715us, period: 69646us, count: 100
[2023-09-11 15:02:32.849][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 61434us, total_duration: 294815us, period: 81450us, count: 100
[2023-09-11 15:02:37.320][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 98us, total_duration: 334us, period: 68863us, count: 100
[2023-09-11 15:02:40.023][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 61157us, total_duration: 261785us, period: 71744us, count: 100
[2023-09-11 15:02:43.657][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 2467us, total_duration: 3980us, period: 63369us, count: 100
[2023-09-11 15:03:17.638][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 107395us, total_duration: 235396us, period: 376141us, count: 100
[2023-09-11 15:03:46.470][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1807us, total_duration: 5861us, period: 628070us, count: 100
[2023-09-11 15:03:58.542][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 72044us, total_duration: 306684us, period: 409046us, count: 100
[2023-09-11 15:04:04.863][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1027us, total_duration: 1908us, period: 183992us, count: 100
[2023-09-11 15:04:18.193][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 39129us, total_duration: 243726us, period: 196504us, count: 100
[2023-09-11 15:04:21.847][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 1070us, total_duration: 2208us, period: 169840us, count: 100
[2023-09-11 15:04:53.001][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 100960us, total_duration: 340927us, period: 347897us, count: 100
[2023-09-11 15:05:22.697][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 664us, total_duration: 2137us, period: 608502us, count: 100
[2023-09-11 15:06:22.306][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30281]name: 0:80, max_duration: 23412us, total_duration: 167417us, period: 893225us, count: 100
[2023-09-11 15:06:57.844][I][Statistics][Statistics.cpp:68][CheckFinishOneRound][30280]name: 0:8, max_duration: 228us, total_duration: 1135us, period: 951464us, count: 100
```

高IO 和高CPU loading

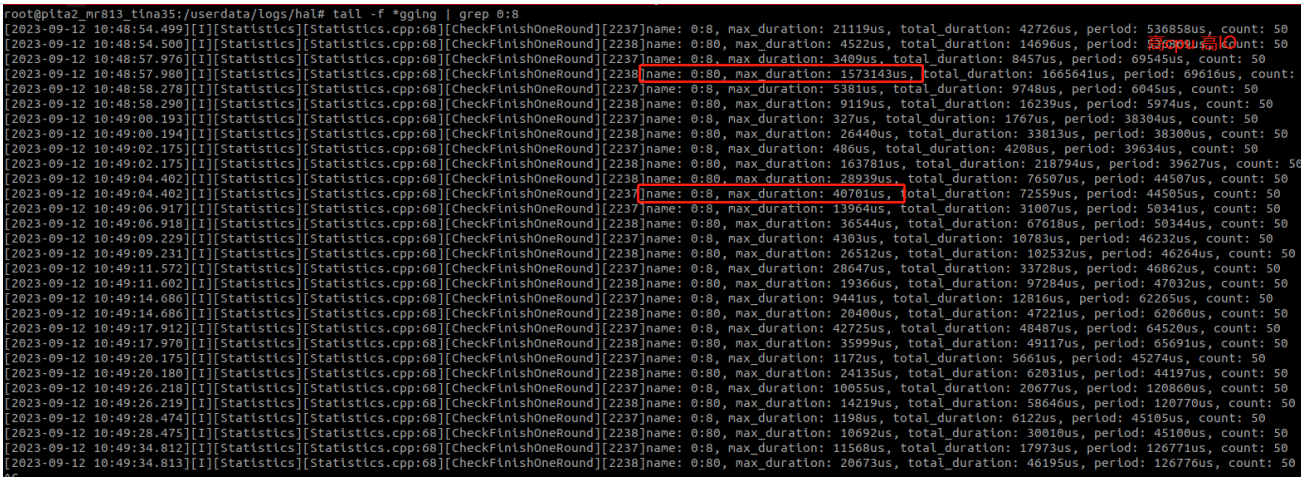
3 模拟高IO情况下：（写日志操作时间最高达到700ms左右，但解包线程写缓存时间，509us）



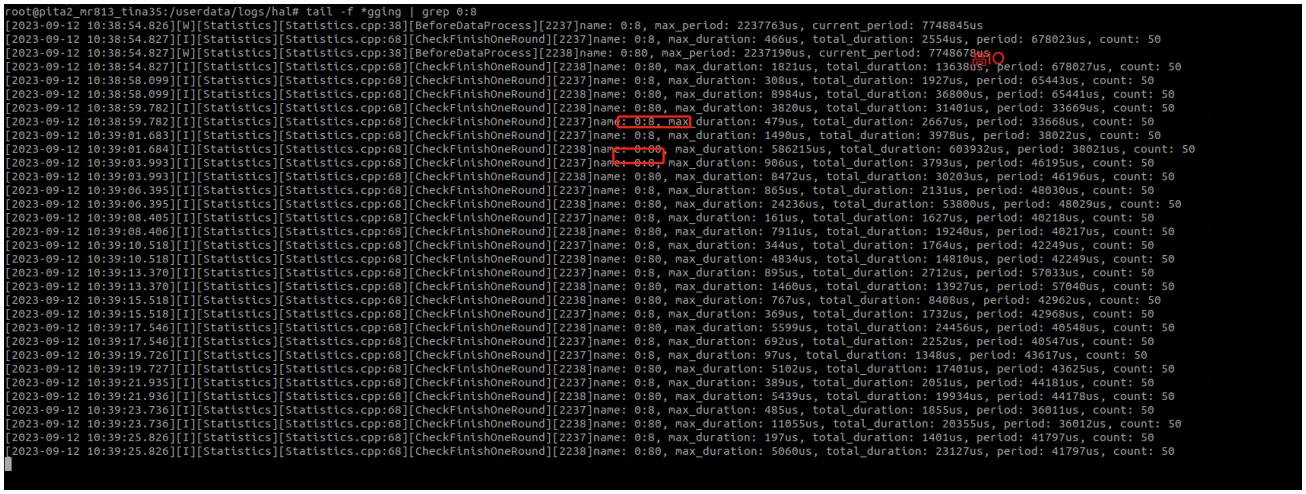
B 使用robokit::frame::Channel：使用链表实现，基本无阻塞的风险，但极端情况下可能造成内存膨胀。

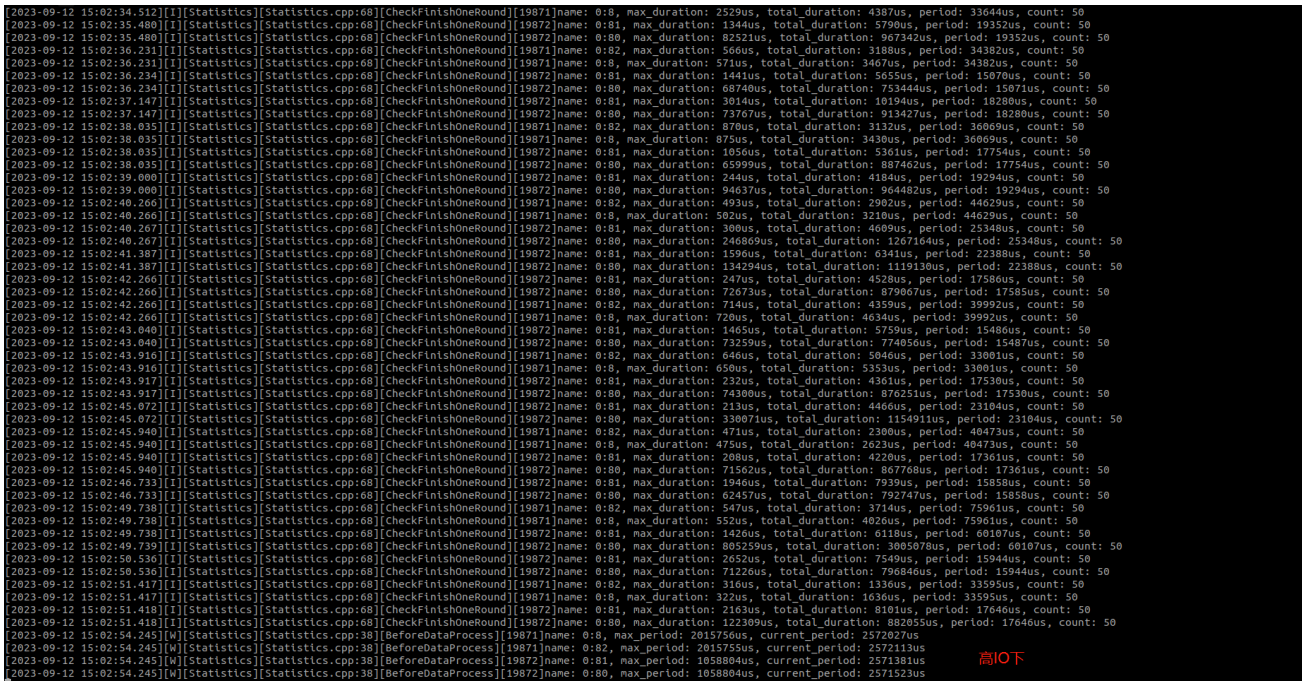
1 节点 cpu占用率和内存比以上高一丢丢。

2 模拟高IO和高CPU loading 的情况下：（写日志操作时间最高达到1.5s左右，但解包线程写缓存时间，40ms）



3 模拟高IO情况下：（写日志操作时间最高达到586ms左右，但解包线程写缓存时间，479us）





C 最后根据具体测试情况和两种结构对高IO和高loding 的敏感度不同，决定采用Kfifo

选择的数据结构	描述	优点	缺点	选择的方案
robokit::frame::Channel	使用**锁和条件变量**来同步，同个时间点不支持同时写入缓存和取出缓存，**底层使用链表数据结构**。	线程安全的，支持多线程读写日志文件，且当数据包过大时不会有缓冲区满的情况。	**对cpu loading高过于敏感。极端情况下可能造成内存膨胀**	
Kfifo	使用**无锁队列**，支持只有一个生产者和一个消费者的情况，但需**内存屏障来保证执行顺序，底层使用连续数组。**	没使用到锁和条件变量，**对cpu loading过高不敏感**。	**存在极端情况下缓存区满导致延时过大。**	√

同步MR813监控程序system_monitor到AR9341的开发计划

When

Starttime	2023-7-18
Endtime	

Statement & Background

因AR9341板上有对资源消耗和性能分析的需求，需要同步现有MR813板上的system_monitor节点程序到AR9341上，并根据需要增加相应的功能。
需要增删的功能需求如下：
1.
2.
3.

Stakeholders Analysis

相关方	产生的影响	备注
haojia xie(谢浩佳)	需求实现方	终端软件中心-嵌入式系统部-系统服务部
rui dai(戴锐)	需求实现方	终端软件中心-嵌入式系统部-系统服务部
samuel	需求提出方	终端软件中心-AI感知部-优化部署部

Target

同步AR9341监控程序system_monitor开发计划

Who is the Owner

haojia xie(谢浩佳)

Reviewed by

- [ ] 吴彦龙
- [ ] rui dai(戴锐)
- [ ] 魏法明
- [x] haojia xie(谢浩佳)
- [ ] samuel

Plan & Check Point 计划与校验点

Stage	Time Duration	Owner	Task Description	Dependence
同步AR9341板上的监控程序(第一版)	2023-8-1	haojia xie(谢浩佳)	1. 移植相关的第三方库 2. 监控程序原有功能的验证和解决 3. 具备本地日志文件输出，不支持日志和埋点上报功能。 4. AR9341板上节点信息收集和配置相关节点的信息 5. 读取9341板上CPU内核数以状态	2023-7-25 实现对操作系统cpu核心数的状态读取［已完成］ 2023-7-27 优化-t 参数性能消耗过大的问题［已完成］ 2023-7-31 ar9341移植第一版（git链接），目前还没合并develop分支[已完成] 2023-8-4 合并develop分支后没问题需 配置自启动相关配置文件，并同步到jenkins上。

ACT 复盘总结

Done to Standard	Not resolved to next loop

工作交接表

一 系统监控节点-sys_monitor

交接人：魏法明

交接日期：2023-9-28

工作内容	MR链接	相关文档
1 增加可按cpu/mem/rss排序功能； 2 统计每个节点CPU占用率前三的线程； 3 实现可自定义节点的cpu警告阈值； 4 实现超过设定阈值时上报前三线程；	http://git.narwal.com/pita2/sys_monitor/-/merge_requests/178	熟悉sysmonitor节点 sys_monitor节点的功能优化
5 增加打印每个CPU核心的资源使用情况； 6 增加超时功能,如果节点超过阈值的时间超过设置的值(默认一小时)则重新上报,解决连续超过阈值无法更新上报前三线程的问题。 7 一套监控程序代码同时支持两个平台(MR813 / AR9341) 8 优化-t参数性能消耗过大 9 参数功能优化，去掉参数-A(显示单核使用率),-t和-N不能同时使用	http://git.narwal.com/pita2/sys_monitor/-/merge_requests/185	同步MR813监控程序system_monitor到AR9341的开发计划
10 cpu/mem/disk/node增加uptime字段记录系统启动时间，和platform字段区分813/9341平台 11 node增加app_uptime记录节点启动以来的秒数	http://git.narwal.com/pita2/sys_monitor/-/merge_requests/247	
12 增加通过ros发布埋点数据给应用节点的功能 13 区分不同板子发送给app节点的ros消息,并抽象发布器和订阅器 14 增加接收app发来的开关信号以及心跳机制	http://git.narwal.com/pita2/sys_monitor/-/merge_requests/242	IP1401 APP开发者模式下显示系统监控需求 监控程序添加Ros通讯模块

二 产测节点 product_test

交接人：任琦

交接日期：2023-9-28

工作内容	链接	相关文档
1 增加Y1的产测项目 2 蠕动泵的控制以及状态查询 3 无水电压传感器数据的读取 4 MCU保护状态解除。	http://git.narwal.com/pita2/product_test/-/merge_requests/625	熟悉product_test节点 IP1001产测指令需求
5 增加Y1上蠕动泵的老化测试代码. 6 增加老化时可屏蔽拖布在位检测.(配置文件total_invalid设置为0就行)	http://git.narwal.com/pita2/product_test/-/merge_requests/644	06_IP1001_清水蠕动泵老化测试方案说明文档V1.0（已评审）
7 增加读取wifi ko驱动文件的md5sum值.	http://git.narwal.com/pita2/product_test/-/merge_requests/662	
~~加热器使能开，使能关~~ ~~读取加热器温度~~ ~~水箱在位检测 MopHALL_R_IN~~ ~~读取蓝牙模组识别Recognize~~	已完成但被取消的；	

三 硬件抽象层节点 hal

交接人：rui dai(戴锐)

交接日期：2023-9-28

工作内容	链接	相关文档

1 优化hal中写mcu日志的逻辑	http://git.narwal.com/pita2/hal/-/merge_requests/1081	熟悉hal节点 分离Hal节点中对mcu的log写操作
2 统计非低功耗下mcu上报的Odom数据的转发延时及丢包情况		统计非低功耗下mcu上报的Odom数据的转发延时及丢包情况
3 提取hal中日志打印到事件线程中 4 数据置信度的压测与优化	http://git.narwal.com/pita2/hal/-/merge_requests/1130	

监控程序添加Ros通讯模块

Ref:

IP1401 APP开发者模式下显示系统监控需求

一 需求：

sys_monitor通过ros把系统监控数据发布给application，同时接收App的开关命令处理。

二 问题：

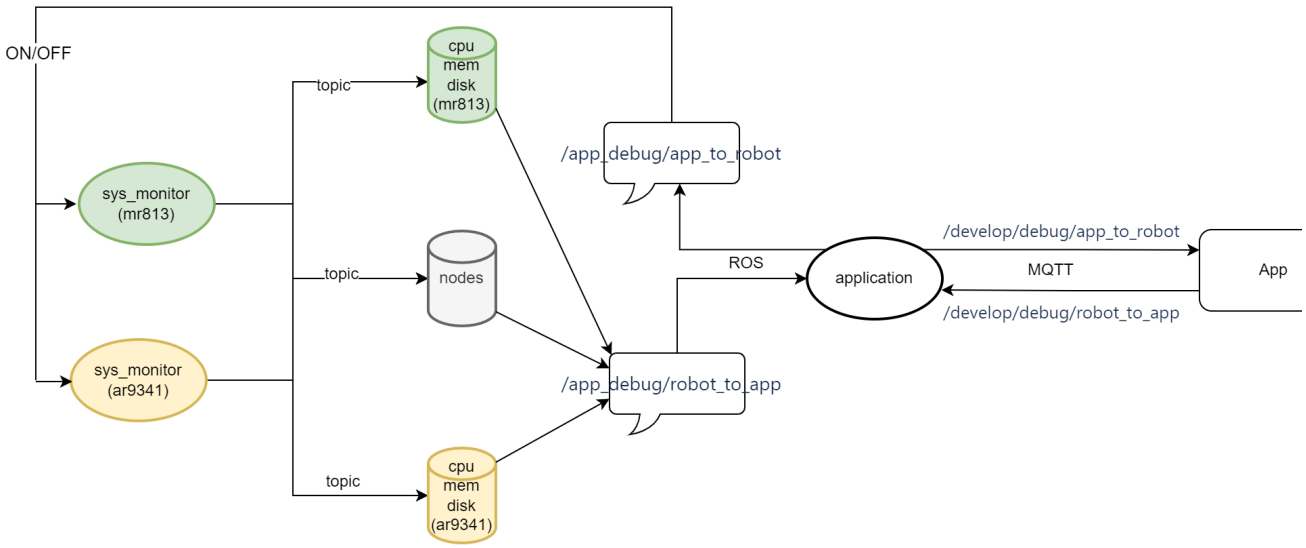
1 ar9341如何上传？

基于ros的话题可以跨板端通讯，cpu,mem和disk需要定义不同的话题名。

2 通讯接口如何定义？

要能跟通讯方式解耦，不能依赖于指定的通讯方式，而是应该定义通用的接口。

3 通讯流程是如何的？



4 使用单线程还是多线程？

①直接单线程创建订阅者和发布者对象，拿到数据就发送，并使用标志位和if语句实现开关。【】

②还是另外创建线程使用一个缓存队列进行发送。（生产者-消费者）

5 通讯话题格式定义：

字段名	类型	用途
url	string	用于标识消息类型，各节点可根据url处理感兴趣的内容
correction_data	string	用于标识消息包，如果出现需要与app请求应答的场景时，可使用。 建议尽量不使用该交互类型，调试需求尽可能使用简单的单向推送
payload	uint8_t[]/bytes	交互内容，具体的需求方与app约定格式、字段、解析方法，建议统一在八爪鱼内创建一个调试消息专用目录，并用protobuf解析

6 通讯协议如何定义：

收到设置开关指令都返回对应的ack包，如果请求url 有问题，返回一个错误包。

心跳机制：设置定时器，当发现没收到app段发来的心跳开关命令，自主动关闭话题发布。

统计非低功耗下mcu上报的0dom数据的转发延时及丢包情况

> *由于没办法获取mcu发送数据的时间，所以只能**假设mcu和mr813的通讯时间很短可以忽略不计**，即mcu发送数据的时间 = mr813上hal收到数据的时间。*
>
> *所以下面的延时主要来于hal节点的转发延时。*

设备id: 5bd1a844146d4c49af7984c5f0b6ac59　时间： 2023-09-19

以下测试以收到**100个包为单位**，每种情况总测试时间为10分钟左右。

注：

max_delay：最大延时

min_delay：最小延時

avg_delay：平均延时

duration：整个组100个包的总转发时间

转发延时统计

一 正常情况下 16

最大延时：max_delay：14056 us ，min_delay : 23us ， avg_delay:182.67us , duration：1978838us

最小延時：max_delay：68 us **，min_delay : 13us ，** avg_delay:29.23us, duration：1975803us

整个时间段的平均延时：42.85us

最大平均延时：182.67us

最小平均时间：29.23us

二 模拟高Io下（四核 idle 为34% ~40%）

最大延时：max_delay：13293us ，min_delay : 24us ， avg_delay:194.38 us, duration：1975469us

最小延時：max_delay：4252 us **，min_delay : 3us ，** avg_delay:258.87us , duration：1979164us

整个时间段的平均延时：819.25us

最大平均延时：3876.72us

最小平均时间：183.46us

三 模拟高cpu下（四核 idle 为1%~5%）

最大延时：max_delay：248965us ，min_delay : 4us ， avg_delay:2702.22 us, duration：1902517us

最小延時：max_delay：3655 us **，min_delay : 9us ，** avg_delay:158.42us , duration：1987138us

整个时间段的平均延时：61.85us

最大平均延时：194.38us

最小平均时间：35.62us

四 模拟高I0和高cpu下（四核 idle 为1%~5%）

最大延时：max_delay：386075us ，min_delay : 3us ， avg_delay: 6771.76 us, duration：2054261us

最小延時：max_delay：17383 us **，min_delay : 3us ，** avg_delay:614.68us , duration：1981683us

整个时间段的平均延时：957.89us

最大平均延时：67771.76us

最小平均时间：40.12us

丢包率统计

一 正常情况下

丢包率为0

二 模拟高Io下

丢包率为0

三 模拟高cpu下（四核 idle 为2%~5%）

丢包率为0

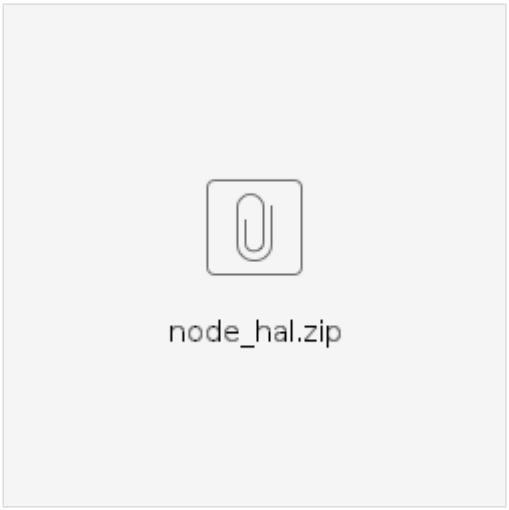
四 模拟高I0和高cpu下

丢包率为0

数据来源（日志文件）

转发延时

可以在下面的日志文件搜**max_delay，min_delay等**相关字段找到对应的记录：



丢包率

可以在下面的日志文件搜loss packet**等**相关字段找到对应的记录：