**框架使用说明**

1. **配置部分（yaml文件）**
2. **Sensor 相关类介绍**
3. **Alg 相关类介绍**
4. **如何编译，如何运行，依赖的第三方库**
5. **配置部分**

**依代码执行流程，依次将解析globle\_cfg.yaml sensors\_cfg.yaml alg\_cfg.yaml alg1.yaml datastore.yaml ….下面将分别介绍相应文件，格式，以及示例给出的简单配置信息，以及相应的解析类ConfigParse，及其提供的接口**

1. **globle\_cfg.yaml**

**存储全局配置信息，如某个传感器是否存在，格式类似如下：**

**HaveGNSS: 1**

**HaveMIMU: 1**

**Havecompass: 1**

**HavePolCamera: 1**

**HaveCamera: 1**

**HaveBarometric: 1**

**通过调用**

ConfigParse::parse\_cfg\_mapstring(std::string cfg\_file, std::map<std::string, std::string>& cfg\_data);

**解析为key-value格式。**

**保存在std::map<std::string,std::string>类型的全局变量**

std::map<std::string,std::string> g\_globa\_cfgData;

**可通过接口**

ConfigParse::get\_glocfg\_byname(std::string& glocfgName);

**于任一地方获取相应的全局配置。**

1. **sensors\_cfg.yaml**

**存储相应sensor对应的配置文件，格式类似如下：**

**myimu: /home/hannuo/RoboosterFramework/config/myimu.yaml**

**mycamera: /home/hannuo/RoboosterFramework/config/mycamera.yaml**

**mypolcamera: /home/hannuo/RoboosterFramework/config/mypolcamera.yaml**

**继而解析对应sensor的配置文件，如myimu.yaml内容如下：**

**type: IMU**

**interval: 100**

**para1: 111**

**para2: 222**

**通过上述ConfigParse提供的接口，解析出相应sensor的信息。**

**创建type为IMU，interval为100的IMU Sensor对象，同时，相应Sensor初始化时，会创建命名为IMU的发布者，供算法对象订阅，同时通过Sensor Factory，将管理所有Sensor对象，开启sensor，发布数据。**

**3. alg\_cfg.yaml**

**存储相应alg对应的配置文件，格式类似如下：**

**datastore: /home/hannuo/RoboosterFramework/config/datastore.yaml**

**alg1: /home/hannuo/RoboosterFramework/config/alg1.yaml**

**alg2: /home/hannuo/RoboosterFramework/config/alg2.yaml**

**继而解析对应alg1的配置文件，如alg1.yaml内容如下：**

**name: alg1**

**data: imu**

**que\_size: 1000**

**inteval: 200**

**如datastore.yaml内容如下**

**name: datastore**

**pubtosub:**

**IMU: imu\_datastore**

**COMPASS: compass\_datastore**

**BAROMETRIC: barometric\_datastore**

**POLCAMERA: polcamera\_datastore**

**que\_size: 1000**

**inteval: 200**

**这里以datastore为例，简单介绍如下：**

**解析alg\_cfg.yaml，匹配相应name，创建对应name的算法对象，解析对应的配置文件，初始化算法对象。这里将依据相应pubtosub分支点信息，分别创建绑定于IMU发布者的命名为imu\_datastore的订阅者对象，绑定于COMPASS发布者的命名为compass\_datastore的订阅者对象……**

**若相应发布者为空，则不创建相应订阅者。**

**通过Alg Factory，将管理所有alg对象，star 算法对象，这里实现为开启线程，通过alg对象的订阅者成员，获取数据，做相应计算处理，存储到local file中。**

1. **Sensor 相关类简介**
2. **CSensorFactory 接口介绍**

**1.1**

create\_sensors(std::string cfg\_file);

传入sensor配置文件路径，依配置信息，创建对应sensor对象，

以及对应的发布者对象

**1.2**

start\_sensors();

为每个sensor对象创建线程，通过其发布者成员，周期发布数据

**1.3**

stop\_sensors();

关闭停止发布数据

1. **具体Sensor类，如CImu**

基本数据成员，接口信息简介：

publisher<IMU\_DATA\_STRU>\* m\_publiser; //发布者成员

uint32\_t m\_interval；//时间间隔成员

publish();//发布数据接口函数

1. **Alg 相关类介绍**
2. **CAlgFactory 接口简介**

**1.1**

CAlgFactory::create\_algs(std::string alg\_name)

传入alg配置文件路径，依配置信息，创建相应alg对象，

以及对应的订阅者对象

**1.2**

start\_algs();

遍历管理的每个算法对象，调用算法的start接口，开启线程，通过alg对象的订阅者成员，获取数据，做相应计算处理，存储到local file中。

**1.3**

stop\_algs();

关闭算法，停止获取数据

**2.具体Alg类，如CAlgDataStore**

基本数据成员，接口信息简介：

subscriber<IMU\_DATA\_STRU>\* m\_algIMUSub = NULL;

subscriber<COMPASS\_DATA\_STRU>\* m\_algCPSub = NULL;

……//算法对象类内的订阅者成员

std::map<std::string, std::string> m\_algCfgPubToSub;

//存储解析配置文件获取的配置信息

star(); //开启算法，订阅数据，处理数据，存储数据到本地文件

stop(); //关闭算法

1. **如何编译，如何运行，依赖的第三方库**
2. **如何编译**

**进入script目录，运行build.sh脚本文件**

**cd script**

**./build.sh**

**（即相应编译过程，已写入脚本build.sh，会自动创建目录，cmake，编译，链接）**

**2.如何运行**

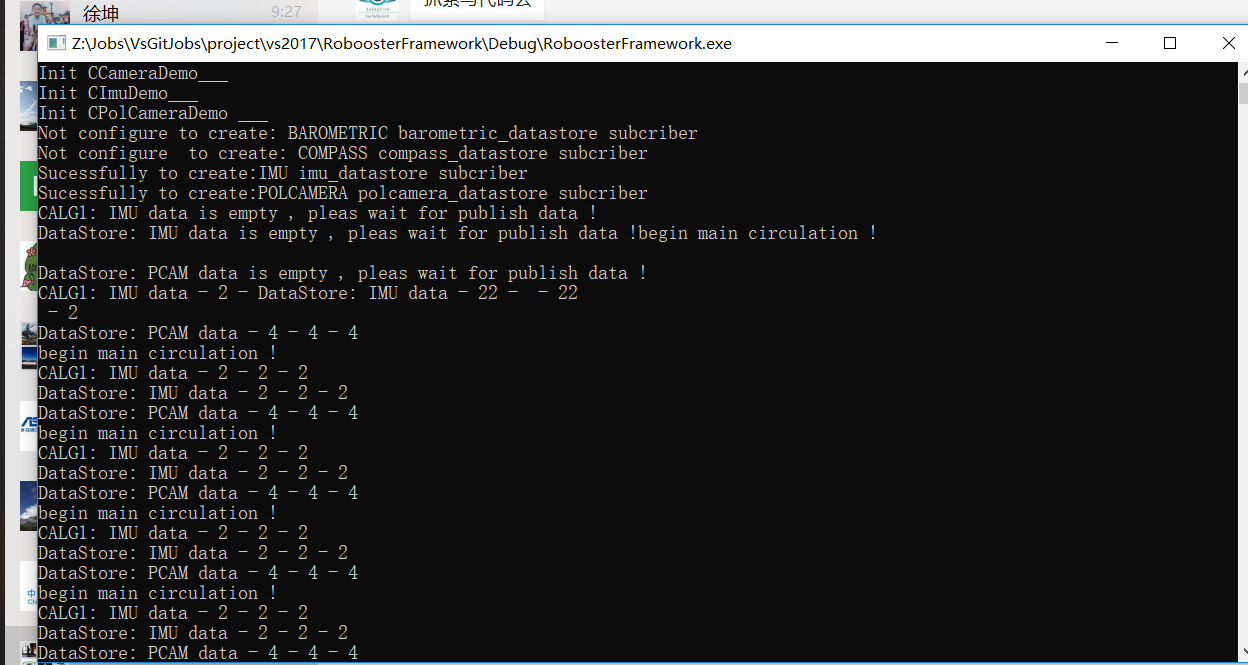
**在主目录下的release目录，将生成的可执行文件Robooster\_framwork**

**执行指令**

**cd release**

**./Robboster\_frmawork /home/hannuo/RoboosterFramework/config**

**即可查看当前示例debug版本**

****

**3.依赖的库**

**请确保yaml-cpp/lib 目录下，有对应的第三方库文件**

**libyaml-cpp.so libyaml-cpp.so.0.6 libyaml-cpp.so.0.6.0**

**（目前使用的为yaml 0.6.0版本，需注意arm，X86下需分别生成对应cpu架构库）**