模块通信和日志、故障诊断使用说明

ROS 系统有一个小乌龟程序来演示系统运行的方式，我们编写了类似程序来演示模块通信的运行方式。

本文档会使用 Tool 下的数个程序，包括 sample1，sample2，ivdiagnosis，view_ivlog 四个模块。

一 编写 sample1 和 sample2 之间通信的消息

在 src/include/proto 下增加了一个 proto 文件，名称为 samplemsg.proto，定义传递的消息。如下：

```
syntax = "proto2";

package iv;

message samplemsg
{
  required int32   mvalue = 1;
  optional int64   msendtime = 2;
};
```

执行 src/include/proto 下的 protomake.sh 脚本生成头文件和源文件。

二 编写 sample1 模块

1）增加库引用

在.pro 文件内加入如下代码

```
LIBS += -lprotobuf
INCLUDEPATH += $$PWD/../../include/msgtype
```

```
INCLUDEPATH += $$PWD/../../../include/
LIBS += -L$$PWD/../../../bin/ -lxmlparam -lmodulecomm -livlog -livfault
```

增加了对 xmlparam 库、modulecomm 库、ivlog 库、ivfault 库的引用。

2）修改 main.cpp 文件

包含头文件。

```
#include "xmlparam.h"
#include "ivlog.h"
#include "ivfault.h"
```

定义全局变量

```
iv::Ivlog   * givlog;
iv::Ivfault * givfault;
std::string gstrmemname;
```

从 xml 配置文件导入模块名称定义和共享内存消息名称定义。

```
QString strpath = QCoreApplication::applicationDirPath();
if(argc < 2)
    strpath = strpath + "/sample1.xml";
else
    strpath = argv[1];
std::cout<<strpath.toStdString()<<std::endl;
iv::xmlparam::Xmlparam xp(strpath.toStdString());
gstrmodulename = xp.GetParam("modulename","sample1");
gstrmemname = xp.GetParam("msgname","sample1");
```

初始化日志变量和故障诊断变量。

```
givlog = new iv::Ivlog(gstrmodulename.data());
givfault = new iv::Ivfault(gstrmodulename.data());
```

3）修改 mainwindow.h 文件

增加变量定义

```
extern iv::Ivlog   * givlog;
extern iv::Ivfault * givfault;
extern std::string gstrmemname;
```

定义共享内存的句柄

```
void * mpa;
```

4）修改 mainwindow.cpp 文件

在构造函数内初始化共享内存句柄。

```
mpa = iv::modulecomm::RegisterSend(gstrmemname.data(),1000,1);
```

将当前 Slider 的状态写入到共享内存中。

```
    iv::samplemsg xsam;
    xsam.set_mvalue(value);
    xsam.set_msendtime(QDateTime::currentMSecsSinceEpoch());

    int ndatasize = xsam.ByteSize();
    char * str = new char[ndatasize];
    std::shared_ptr<char> pstr;pstr.reset(str);
    if(!xsam.SerializeToArray(str,ndatasize))
    {
        std::cout<<"MainWindow::on_horizontalSlider_valueChanged serialize
error."<<std::endl;
        return;
    }
    iv::modulecomm::ModuleSendMsg(mpa,str,ndatasize);
```

写入日志和更新当前故障状态。

```
    givlog->debug("send slider value %d",xsam.mvalue());
    givfault->SetFaultState(0,0,"Run OK");
```

三 编写 sample2 模块

1）增加库引用

在.pro 文件内加入如下代码

```
LIBS += -lprotobuf
INCLUDEPATH += $$PWD/../../include/msgtype

INCLUDEPATH += $$PWD/../../../include/
LIBS += -L$$PWD/../../../bin/ -lxmlparam -lmodulecomm -livlog -livfault
```

增加了对 xmlparam 库、modulecomm 库、ivlog 库、ivfault 库的引用。

2）修改 main.cpp 文件

包含头文件。

```
#include "xmlparam.h"
#include "ivlog.h"
#include "ivfault.h"
```

定义全局变量

```
iv::Ivlog   * givlog;
iv::Ivfault * givfault;
std::string gstrmemname;
```

从 xml 配置文件导入模块名称定义和共享内存消息名称定义。

```
    QString strpath = QCoreApplication::applicationDirPath();
    if(argc < 2)
        strpath = strpath + "/sample2.xml";
    else
        strpath = argv[1];
    std::cout<<strpath.toStdString()<<std::endl;
    iv::xmlparam::Xmlparam xp(strpath.toStdString());
    gstrmodulename = xp.GetParam("modulename","sample2");
    gstrmemname = xp.GetParam("msgname","sample1");
```

初始化日志变量和故障诊断变量。

```
givlog = new iv::Ivlog(gstrmodulename.data());
givfault = new iv::Ivfault(gstrmodulename.data());
```

3）修改 mainwindow.h 文件

增加变量定义

```
extern iv::Ivlog   * givlog;
extern iv::Ivfault * givfault;
extern std::string gstrmemname;
```

定义共享内存的句柄

```
void * mpa;
```

增加回调函数。

```
void UpdateSlider(const char * strdata,const unsigned int nSize,const unsigned int index,const QDateTime * dt,const char * strmemname);
```

4）修改 mainwindow.cpp 文件

在构造函数内初始化共享内存句柄，并关联回调函数。

```
    ModuleFun funupdate =std::bind(&MainWindow::UpdateSlider,this,std::placeholders::_1,std::placeholders::_2,std::placeholders::_3,std::placeholders::_4,std::placeholders::_5);
    mpa = iv::modulecomm::RegisterRecvPlus(gstrmemname.data(),funupdate);
```

在回调函数内解析接收到的数据，并控制 Slider 的位置。

```
    iv::samplemsg xsam;
    if(!xsam.ParseFromArray(strdata,nSize))
    {
        std::cout<<" MainWindow::UpdateSlider parese error."<<std::endl;
        return;
    }
```

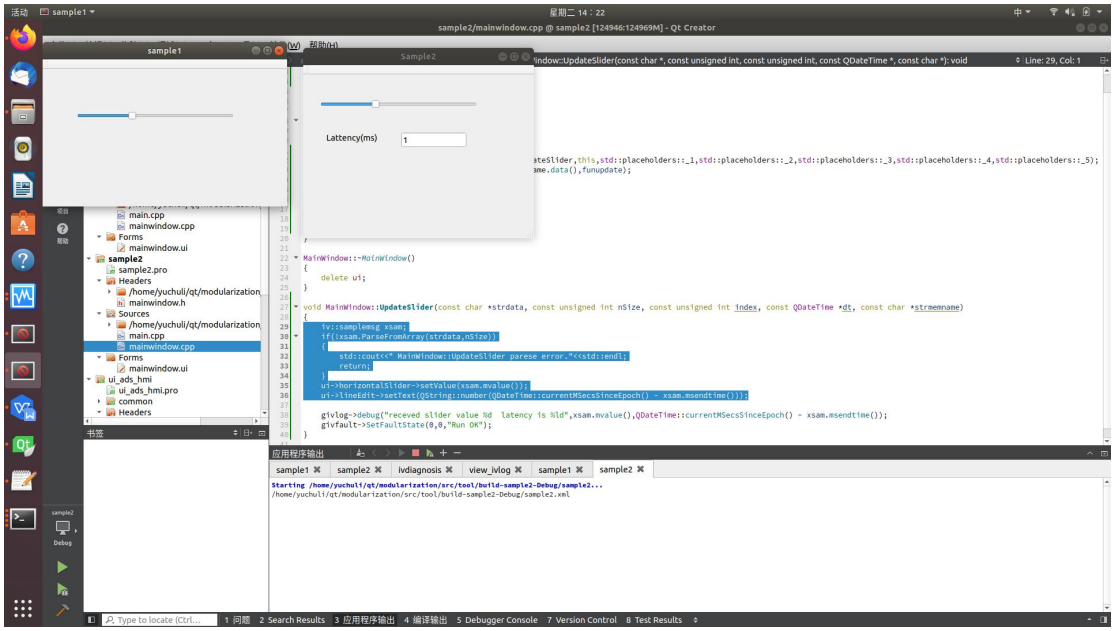
```
    ui->horizontalSlider->setValue(xsam.mvalue());
    ui->lineEdit->setText(QString::number(QDateTime::currentMSecsSinceEpoch()
- xsam.msendtime()));
```

写入日志和更新当前故障状态。

```
    givlog->debug("send slider value %d",xsam.mvalue());
    givfault->SetFaultState(0,0,"Run OK");
```

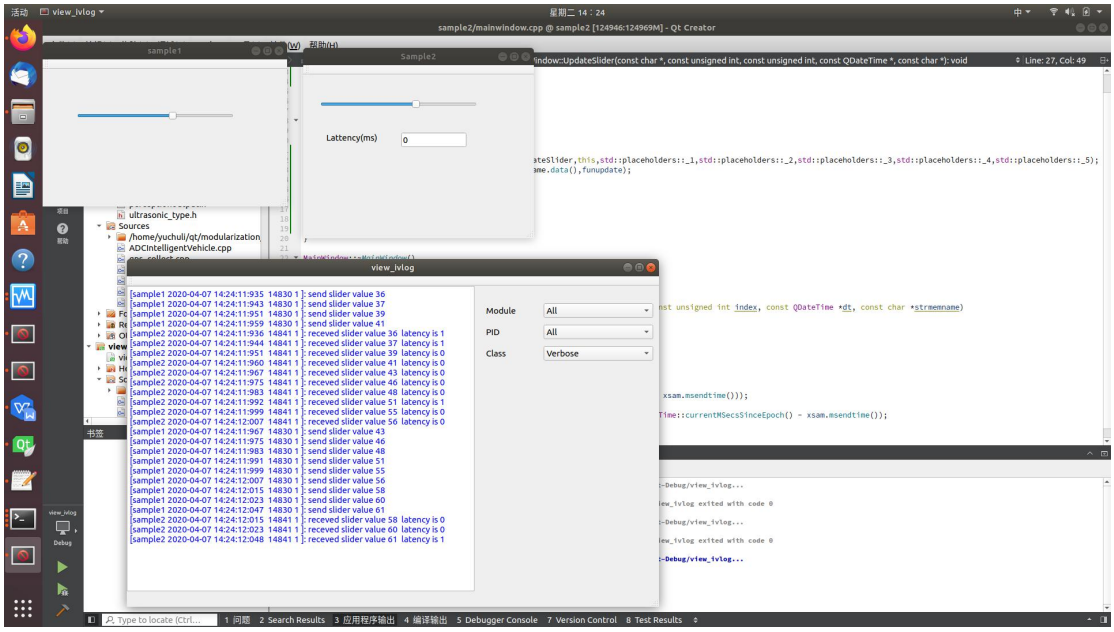
四 效果

在 sample1 内拖动 Slider，sample2 内的 Slider 会同步运动。



五 日志系统

打开 view_ivlog，将 Class 调到 Verbose 来显示所有日志。



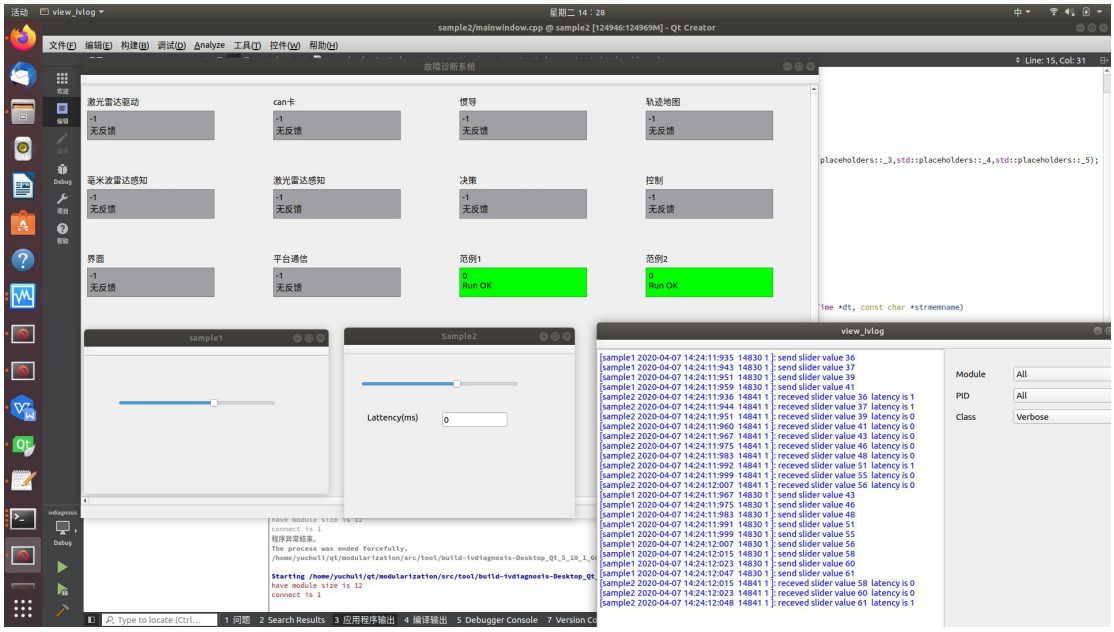
六 故障诊断系统

在故障诊断系统的 yaml 标定文件内增加 sample1 和 sample2。在 module 下增加模块，然后增加模块的定义，moudlename（模块名称）和 titile（显示名称）。

```
module:
   - driver_lidar
   - driver_can
   - driver_gps
   - driver_map
   - detect_radar
   - detect_lidar
   - decition
   - contoller
   - ui
   - platform
   - sample1
   - sample2
driver_lidar:
   modulename: driver_lidar_rs16
   title: 激光雷达驱动
driver_can:
   modulename: driver_can_kvaser
```

```
  title: can 卡
driver_gps:
  modulename: driver_gps_hcp2
  title: 惯导
driver_map:
  modulename: driver_map_trace
  title: 轨迹地图
detect_radar:
  modulename: detection_radar_delphi_esr
  title: 毫米波雷达感知
detect_lidar:
  modulename: detect_lidar_grid
  title: 激光雷达感知
decition:
  modulename: decition_ge3
  title: 决策
contoller:
  modulename: contoller_ge3
  title: 控制
ui:
  modulename: ui_ads_hmi
  title: 界面
platform:
  modulename: platform
  title: 平台通信
sample1:
  modulename: sample1
  title: 范例 1
sample2:
  modulename: sample2
  title: 范例 2
```

将 ivdiagnosis.yaml 放到 ivdiagnosis 的运行路径。运行效果如图所示。



在故障诊断系统界面上能看到每个模块的状态。颜色对应的状态分别为：

灰色　　不在线

黄色　　警告

红色　　故障

绿色　　正常