# Tracer代码框架使用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修订人 | 更新日期 | 更新内容 | 审核人 |
| Ver1.0 | 操正 | 2023.07.10 | 初稿 | 吴青，滑国青 |
| Ver1.1 | 操正，谭绍军 | 2023.07.18 | 增加辅助调试功能说明 | 吴青，滑国青 |
|  |  |  |  |  |

目录

[Tracer代码框架使用 1](#_Toc140695729)

[一、框架概述 1](#_Toc140695730)

[1. 通信方式 1](#_Toc140695731)

[2. 日志存储 2](#_Toc140695732)

[3. 线程创建 2](#_Toc140695733)

[4. 消息队列创建 2](#_Toc140695734)

[5. 进程线程同步机制 2](#_Toc140695735)

[6. 单元测试 2](#_Toc140695736)

[7. 框架特性 2](#_Toc140695737)

[二、增加功能模块步骤 2](#_Toc140695738)

[1. DroneIdCmdproc（线程类） 2](#_Toc140695739)

[2. DroneIdCmdproc（命令处理类） 3](#_Toc140695740)

[3. DroneIdModule（模块类） 3](#_Toc140695741)

[4. 发送异步消息 3](#_Toc140695742)

[5. 发送同步消息 3](#_Toc140695743)

[6. 注意事项 3](#_Toc140695744)

[三、在线辅助调试功能 3](#_Toc140695745)

[1. 打开调试功能 3](#_Toc140695746)

[2. 添加调试信息 4](#_Toc140695747)

[3. 开始调试 4](#_Toc140695748)

[4. 注意事项 6](#_Toc140695749)

# 一、框架概述

## 1. 通信方式

1. 采用消息队列，用于进程、线程之间的消息传递。
2. 使用共享内存维护：线程与消息队列、进程与消息队列、响应进程与消息队列映射表。

## 2. 日志存储

日志按照不同级别分类输出到系统文件并可保存日志文件。

## 3. 线程创建

线程默认每个模块创建一个，创建线程可以指定绑定的核心。

## 4. 消息队列创建

消息队列每个线程创建一个，创建以后和本线程绑定。

## 5. 进程线程同步机制

提供互斥锁、信号量、条件变量、自动锁等线程进程同步机制。

## 6. 单元测试

使用gtest对于代码模块进行单元测试。

## 7. 框架特性

框架采用C++方式进行封装，将公共的属性和行为抽象成类，子类可以根据需要继承一个或多个父类的属性和行为，也可以通过类的组合形式实现功能的叠加，父类可以通过调用子类的重写的虚函数实现动态多态，封装继承多态的特性，使得程序的不同功能模块相互独立，代码变得更加简洁，极大程度降低代码冗余，提高程序扩展性，可轻松增加或拆卸功能模块或组件，完成一个新的系统或程序的组装。

功能模块的运行流程基本一致，以droneid为例

# 二、增加功能模块步骤

## 1. DroneIdCmdproc（线程类）

定义线程：DEFINE\_THREAD

定义定时器：DEFINE\_TIMER

创建线程关联的消息队列：\_InitQueue

注册定时器：\_RegisterTimer

线程入口：\_OnCommand，可以根据需要增加业务。

## 2. DroneIdCmdproc（命令处理类）

注册命令与处理函数映射表：Register

实现具体业务：根据业务需要增加函数实现

定时器入口：\_OnTimer，可以在里边增加定时任务。

## 3. DroneIdModule（模块类）

创建命令处理类对象：\_CreateCmdproc

创建线程实例：\_CreateThread

## 4. 发送异步消息

SendAsyncMsg

## 5. 发送同步消息

SendSyncMsg

## 6. 注意事项

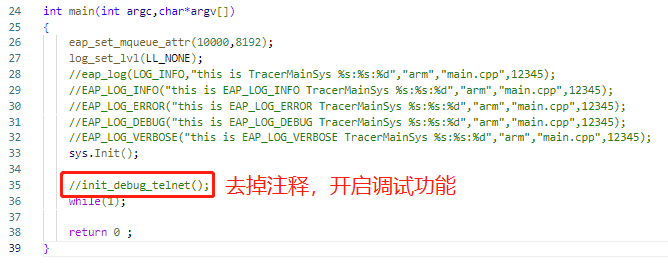
DroneId模块创建了命令处理类，所以底层会默认走命令处理流程，不需要实现线程入口函数\_OnCommand，只需要根据注册的命令，实现对应的处理函数即可，想实现定时任务通过定时器实现。

# 三、在线辅助调试功能

Tracer的Linux版本，将业务封装成libeap.so，使用调试工具可输入动态库中的全局函数进行在线执行，或输入动态库中的全局变量查看实时数据。

## 1. 打开调试功能

修改linux\_app/src/main.cpp打开init\_debug\_telnet()。



## 2. 添加调试信息

以linux\_app/src/eap/base/thread/ThreadMgr.cpp为例，

添加函数以显示本模块想要获取的调试信息：

（由于C++函数名称在编译后会改名，改为C函数更方便调试打印）

extern "C"

{

void ShowAllThreadsMsgStats()

{

EapGetSys().GetThreadMgr().ShowAllThreadsMsgStats();

}

void ShowOneThreadMsgStats(uint16\_t threadId)

{

EapGetSys().GetThreadMgr().ShowOneThreadMsgStats(threadId);

}

}

## 3. 开始调试

在serial或者ssh连上后，输入“telnet 127.0.0.1 10000”进入调试命令行；

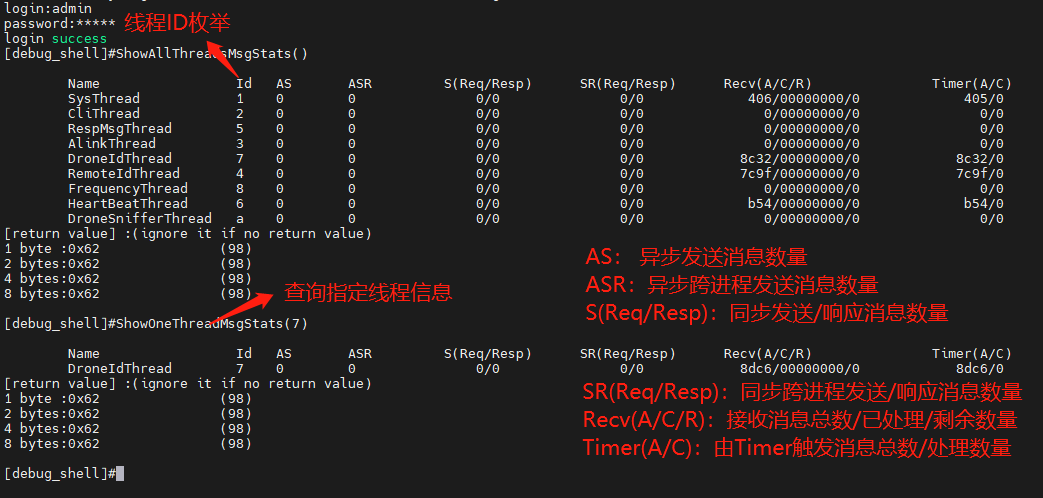
或者直接新建telnet登录到tracer，注意IP地址能连通，端口是10000，

用户名：admin

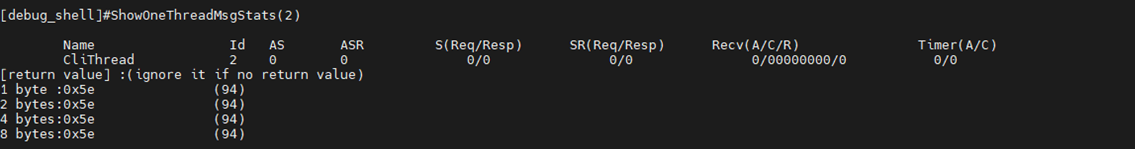
密码：admin

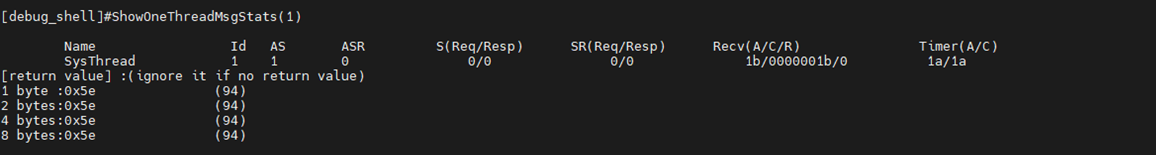
调用函数打印调试信息：

ShowAllThreadsMsgStats()，可以打印线程状态信息，不传参数就打印所有线程



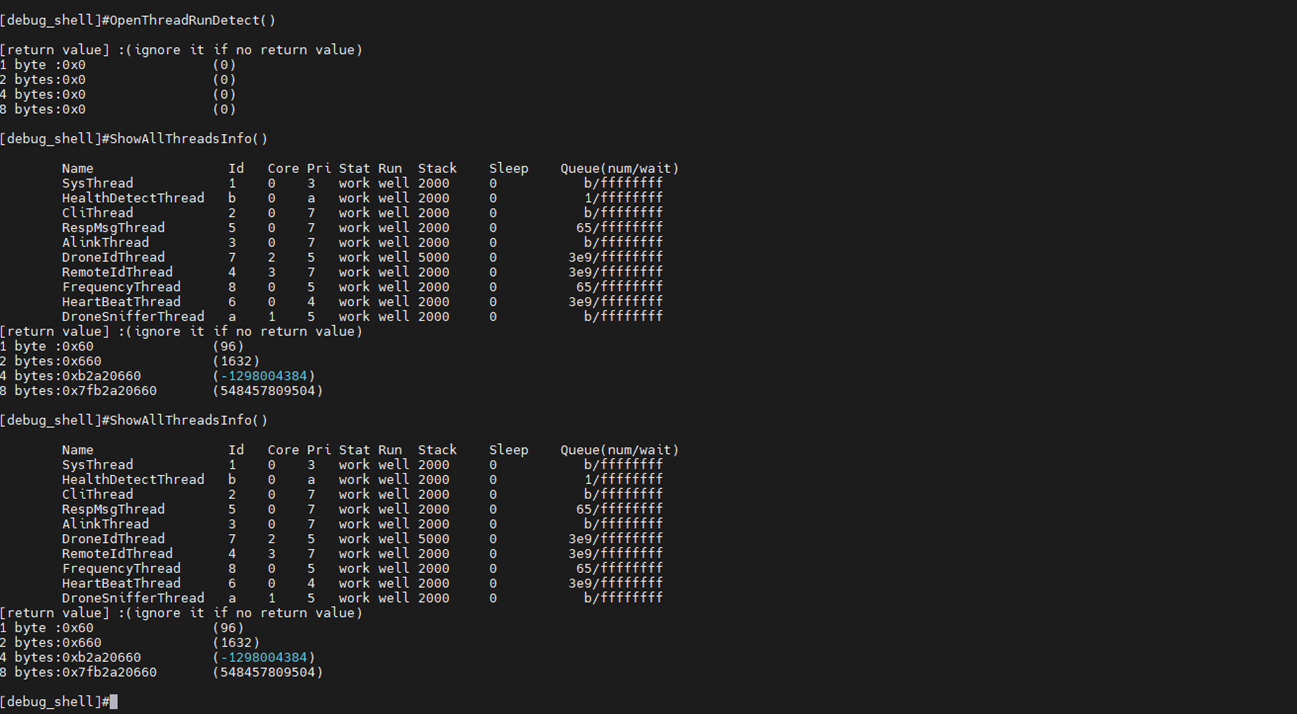
传递线程id，可以打印指定线程信息





OpenThreadRunDetect()，开启线程运行状态监测

开启后再调用ShowAllThreadsInfo( )



还可以添加C函数用来查询变量的值

## 4. 注意事项

也可通过“变量=值”的方式修改全局变量的值；

在该调试环境，要注意全局函数不能有数据并发访问的情况，在正式版本中这个环境也是要关闭的。