# 透过HTTP\_DEMO学习MCP架构实现

## 程序源文件分析

#### 1.1 首先要知道这个文件由哪些源文件编译而来：

* + - 查看mcp\http\_demo\so\makefile第8行：include $(MCP)smcp/src/makefile.inc

再看makefile.inc中的内容： (IdxObjMng.o在这个文件中出现了两次，这个可能与编译有关，但也有可能是多写了，这里先不管)

OBJ += \

$(SMCP)auxhandle.o \

$(SMCP)mainctrl.o \

$(SMCP)mainsvrd.o \

$(PUBLIC)Base.o \

$(PUBLIC)thread.o \

$(PUBLIC)Statistic.o \

$(PUBLIC)tlib\_cfg.o \

$(PUBLIC)tlib\_log.o \

$(PUBLIC)IdxObjMng.o \

$(PUBLIC)CodeQueue.o \

$(PUBLIC)CodeQueueMutil.o \

$(PUBLIC)Sem.o \

$(PUBLIC)HashFunc.o \

$(PUBLIC)AnsyTimerQueue.o \

$(PUBLIC)IdxObjMng.o

* + - Makefile第20行，10~12行，可知OBJ又多了：

MainSoWork.o

AuxSoWork.o

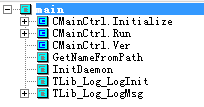
* + - 找到所有的.o以后，再找对应的源文件就不难了，不要漏掉.h和.hpp

## 程序运行分析

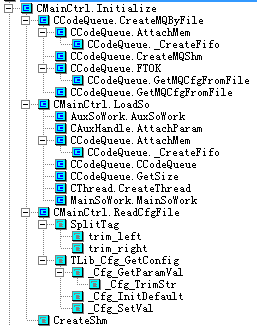
#### 2.1 调用层次关系

* + - 先找MAIN函数：

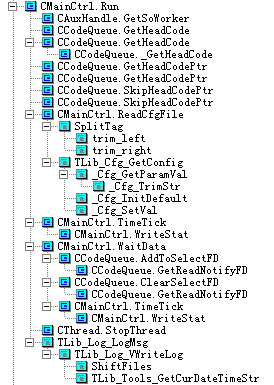
在smcp\src\mainsvrd.cpp中，其调用与执行关系如下：



图表 1 HTTP\_DEMO MAIN函数调用层次



图表 2 HTTP\_DEMO MAIN函数调用层次



图表 3 HTTP\_DEMO MAIN函数调用层次

* + - MAIN函数的调用关系并不一定是程序的执行顺序，还有多线程，多进程的问题需要考虑分析

查找fork，发现除了创建后台服务的进程外，没有新的进程

查找pthread\_create，发现在Thread.cpp中创建了一个接收处理线程，具体处理在CAuxHandle类中的Run方法中。（CAuxHandle继承了CThread，并实现了Run方法），具体调用在CMainCtrl.Initialize🡪CMainCtrl.LoadSo🡪CThread.CreateThread中。

这里只是可能被调用，具体有没有被调用，还要看具体配置

#### 2.2 分析程序中的关系逻辑

配置文件内容(可以结合mcp\http\_demo\etc下的http\_demo\_mcp.cfg的配置文件查看)：

//基本

char m\_szSvrName[256];

char m\_szCfgFileName[256];

//IN通信管道

char m\_szXToMeMQ[256];

char m\_szMeToXMQ[256];

//SCC通信管道

char m\_szSCCToMeMQ[256];

char m\_szMeToSCCMQ[256];

int m\_aiAuxBindCpu[1024];

int m\_iAuxHandleMode;

int m\_iAuxHandleNum;

int m\_iDispatchAuxMQSize;

int m\_iCPUAffinity;

int m\_iStatTimeGap;

int m\_iMaxLogSize;

int m\_iMaxLogNum;

//消息包在队列中呆的最长时间us

int m\_iMaxQueueWaitus;

//日志开关

TShmLog\* m\_pShmLog;

类的关系：

CMainCtrl中有：CCS\_CQ，CCodeQueue，SoWorker，CAuxHandle成员

CCS\_CQ 等价于CCodeQueue

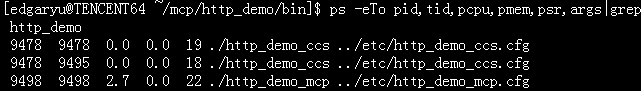
MainSoWork，AuxSoWork继承SoWorker

SoWorker中有：CCodeQueue，CCS\_CQ成员

MCAuxHandle继承CThread

CMainCtrl \*g\_pMainCtrl 是一个全局变量，在main函数中被实例化，并开始程序调用。

http\_demo的实际运行效果如下：



图表 4 http\_demo 运行效果

从图中可以看出，运行后，ccs运行后有两个线程，但mcp运行后只有一个线程。（本文档中不讨论ccs，实际上这里运行着的ccs是指向mcp/ccs/bin 目录下的ccs的软连接）

#### 2.3 分析程序中的运行逻辑步骤

实例化全局变量g\_pMainCtrl🡪

初始化

读取配置文件🡪

创建IN共享内存管道🡪

创建SCC共享内在管道🡪

创建共享内存日志空间🡪

实例化m\_pMainSoWorker🡪

根据是否连接scc的配置进行初始化🡪

根据辅助线程的配置（数量和模式）进行相关操作🡪

CPU亲和性设置🡪

SERVER主流程（循环开始）🡪

程序退出处理（不期望－unlikely）🡪

重加载配置(不期望－unlikely)🡪

统计信息写入🡪

emCompeteINMQ模式下主线程短暂等待并回到循环开始处。

处理CCS发送过来的消息🡪

处理SCC发送过来的消息🡪

没有数据，则继续等消息（SELECT重置，清除通知，记入统计）🡪循环开始

## 思考与问题

* + - 本http\_demo的配置说明：

未配置连接SCC

未配置辅助线程

* + - 值得思考的问题：

1. 立即数问题（代码中出现大量的立即数，不利于统一管理和分析）
2. Mainctrl.cpp第150行，223行对SoWorker直接进行实例化成两个子类，我觉得这里可以使用工厂方法