SPP 基础库源码分析

一、原子操作基础库

在tbase文件夹下，包括以下源码文件，各文件的用途如下

**atomic.h**:通过\_\_GNUC\_\_和\_\_WORDSIZE宏判断当前编译环境引入的具体原子操作头文件

**atomic\_asm.h**:当编译环境为linux gcc版本小于4.0时被引入，

**atomic\_gcc.h**: 当编译环境为linux gcc版本大于4.0时被引入，

**atomic\_asm8.h**: 原用于32位系统时引入，现被注释了

**atomic\_gcc8.h**: 当编译环境系统为64位时被引入

**myatomic\_gcc8.h**: 当编译环境系统为64位时被引入

以上几个文件都是封装了glibc库中的cdefs.h的原子操作函数和相关的数据结构，以更简洁的名称提供给SPP各处代码使用。

注解：预定义\_\_GNUC\_\_宏

1 \_\_GNUC\_\_ 是linux平台，gcc编译器编译代码时预定义的一个宏。需要针对gcc编写代码时， 可以使用该宏进行条件编译。

2 \_\_GNUC\_\_ 的值表示gcc的版本。需要针对gcc特定版本编写代码时，也可以使用该宏进行条件编译。

3 \_\_GNUC\_\_ 的类型是“int”，该宏被扩展后， 得到的是整数字面值。可以通过仅预处理，查看宏扩展后的文本。

预定义\_\_WORDSIZE宏，用于判断系统为32位还是64位

二、链表基础库

**list.h：**定义和实现链表的数据结构及相关操作

三、日志基础库

在tbase文件夹下，包括以下源码文件，各文件的用途如下

**tlog.h**：定义了日志功能所需的宏、枚举结构及日记类CTLog

详细源码分析：

首先保障多进程和多线程时的文件句柄安全性问题

定义了日志文件打印相关的多个宏

在tbase的名字空间下，定义tlog名字空间。

以下定义都在tbase::tlog名字空间下

定义枚举类型LOG\_TYPE，LOG\_LEVEL

定义钩子函数原型

定义日志类CTLog，CTLog的主要公共接口如下：

*//初始化日志*

*int log\_open(int log\_level, int log\_type, const char\* log\_path, const char\* name\_prefix, int max\_file\_size, int max\_file\_no);*

*//设置日志级别*

*int log\_level(int level);*

*//打印格式化日志*

*void log\_i(int flag, int log\_level, const char \*fmt, ...);*

*void log\_i\_va(int flag, int log\_level, const char\* fmt, va\_list va);*

*//打印bin日志*

*void log\_i\_bin(int log\_level, const char\* buf, int len);*

*//设置钩子函数*

*void log\_set\_hook(log\_hook hook);*

*//把二进制数据转换为可显示的hex字符串*

*static const char\* to\_hex(const char\* bin, int len, char\* buf, int size);*

定义用户使用应该调用跟函数对应的宏

**tlog.cpp**：具体实现CTLog类的各个函数

四、统计基础库

在tbase文件夹下，包括以下源码文件，各文件的用途如下

**tstat.h**：定义

1）各种统计操作宏，

2）各种统计相关数值宏，

3）各种错误码定义

4）统计对象结构体TStatObj

5）统计对象池TStatPool

6）统计池类CTStat

**tstat\_policy.h** 定义

1）原子数统计操作宏

2）外部可见的统计对象数据结构TStatObjWrapper

3）统计策略类CTStatPolicy（被CTStat类使用）

**tstat.cpp**：实现统计池类CTStat

五、共享内存基础库

在tbase文件夹下，包括以下源码文件，各文件的用途如下

**hex\_dump.h：**声明了两个文件倒成16进制数据的函数接口

**tshmcommu.h**：作了以下的一些定义：

锁的类型宏；

错误码宏：

共享内存管道统计结构体TMQStat，Q\_STATINFO

共享内存管道类CShmMQ：

共享内存生产者类（不带锁）CShmProducer：

共享内存生产者类（带锁）CShmProducerL：

共享内存消费者类（不带锁）CShmComsumer：

共享内存消费者类（带锁）CShmComsumerL：

共享内存通讯组件配置结构体TShmCommuConf

共享内存通讯组件结构体CTShmCommu

**tshmcommu.cpp**：实现tshmcommu.h中定义各个类

**notify.h**：定义了用于通知共享内存有数据可以接收的类CNotify

**notify.cpp：**实现了用于通知共享内存有数据可以接收的类CNotify

六、进程通信基础库

**tprocmon.h:** 定义了

各类消息相关宏，

进程事件相关宏，

进程操作相关宏，

进程状态相关宏，

进程组信息结构体TGroupInfo，

进程信息结构体TProcInfo，

事件通知结构体TProcEvent，

服务器端与客户端通讯消息包TProcMonMsg，

通讯基类CCommu，

消息队列通讯类CMQCommu，

内部使用的多种数据结构，进程组对象结构体TProcGroupObj，进程对象结构体TProcObj

进程通讯函数指针

进程查询对象结构体TProcQueryObj

进程监控服务端类CTProcMonSrv

进程监控客户端类CTProcMonCli

**tprocmon.cpp:**

实现tprocmon.h头文件中定义的各个类

**hide\_private.h：**定义消息进程信息结构体MG\_PROC\_INFO

**misc.h：**定义一个杂项操作封装类CMisc

**misc.cpp：**实现类CMisc

七、网络通信基础库

**1、在tbase文件夹下tcommu.h：定义了以下内容：**

**1）通用通信错误宏**

*#define COMMU\_ERR\_OVERLOAD\_PKG -100*

*#define COMMU\_ERR\_OVERLOAD\_CONN -200*

**2）SOCKET域参数类型宏**

*#define SOCK\_TYPE\_TCP 0x1*

*#define SOCK\_TYPE\_UDP 0x2*

*#define SOCK\_TYPE\_UNIX 0x4*

*#define SOCK\_TYPE\_NOTIFY 0x8*

*#define SOCK\_MAX\_BIND 300*

**3）回掉函数类型枚举cb\_type，  
4）控制命令枚举ctrl\_type：**

**5）回调函数类型:**

*//flow: 数据包唯一标示*

*//arg1: 通用参数指针1，一般指向数据blob*

*//arg2: 通用参数指针2，用户注册回调函数传入的自定义参数指针*

*typedef int (\*cb\_func)(unsigned flow, void\* arg1, void\* arg2);*

**6）数据blob结构体：blob\_type，**

*typedef struct {*

*int len; //数据长度*

*char\* data; //数据缓冲区*

*void\* owner; //组件指针*

*void\* extdata; //扩展数据, 具体组件有具体的含义*

*} blob\_type;*

**7）连接扩展信息结构体：TConnExtInfo**

*typedef struct {*

*int fd\_; //fd*

*int type\_; //fd type (SOCK\_STREAM\SOCK\_DGRAM\...)*

*unsigned localip\_; //local ip*

*unsigned short localport\_; //local port*

*unsigned remoteip\_; //remote ip*

*unsigned short remoteport\_; //remote port*

*time\_t recvtime\_; //recv tv\_sec*

*suseconds\_t tv\_usec; //recv tv\_usec*

*} TConnExtInfo;*

**8）通讯类抽象接口类：CTCommu**

类CTCommu用于定义通讯的抽象接口，

主要定义了以下通用接口

（1）初始化

*virtual int* ***init****(const void\* config) = 0;*

（2）轮询，收发数据

*virtual int* ***poll****(bool block = false) = 0*（3）发送数据提交

*virtual int* ***sendto****(unsigned flow, void\* arg1, void\* arg2) = 0;*

（4）控制接口

*virtual int* ***ctrl****(unsigned flow, ctrl\_type type, void\* arg1, void\* arg2) = 0;*

（5）控制接口

*virtual int* ***reg\_cb****(cb\_type type, cb\_func func, void\* args = NULL)*

（6）清空所有共享内存队列，仅供proxy启动时使用

*virtual int* ***clear****() = 0;*

**在tbase/tsockcommu文件夹下，包括以下源码文件，各文件的用途如下**

**1、tsockcommu.h: 定义了以下内容：**

1）结构体TIpport；

2）结构体TSockBind；

3）套接字配置结构体TSockCommuConf：

**4）继承于CTCommu类的Socket方式通讯的公共类CTSockCommu**

类CTSockCommu除了实现接口CTCommu类中的虚函数外，另外新增以下函数：

（1）增加worker进程到proxy进程的通知

*int addnotify(int fd);*

（2）获取连接池中被使用的连接流

*int getconn();*

类CTSockCommu中重要的成员变量如下：

（1）std::map<int, TSockBind> sockbind\_; //套接字绑定信息

（2）blob\_type buff\_blob\_接收的数据

（3）TConnExtInfo extinfo\_连接相关信息

（4）struct TInternal\* ix\_; 操纵内存池，连接池和epoll的句柄

**2、tsockcommu.cpp:**

实现类CTSockCommu

1. 首先定义了CTSockCommu内部结构体CTSockCommu::TInternal，其内部包含一个指向共享内存池的指针mempool\_，一个指向连接池的指针connset\_，一个操作epoll对象的指针epollflow\_。

同时在CTSockCommu内有一个TInternal类型的指针ix\_，用于处理连接，内存和epoll操作

1. **实现构造、析构函数**
2. **函数check\_expire：**

实现超时处理函数，控制空闲连接超时，调用回调函数。

1. **函数create\_sock：**

实现创建listen 的Socket

（1）调用CSocket的静态成员函数create建立监听套接字  
（2）设置该套接字被重用

（3）设置套接字为非阻塞

（4）设置套接字的接收buff，绑定ip端口或路径

（5）调用CSocket的静态成员函数listen开启套接字监听

1. **函数handle\_accept**

实现接入客户端连接的

1. 每次poll最多建立50个连接，调用CSocket的accept函数
2. 设置跟客户端连接的套接字为非阻塞的
3. 将对应的连接加入连接池
4. 连接建立回调
5. 连接加入epoll池
6. **函数handle\_accept\_udp**

实现接入客户端udp请求

1. 将连接加入连接池
2. 收数据，udp数据一次就可以收完整
3. **函数init**

实现解析配置，初始化

1. **函数addnotify**

加入worker到proxy之间的通知

1. **函数poll**

收发数据

1. 通过CEPollFlowResult接收epoll wait通知，轮询事件，根据事件类型不同做如下各种处理
2. 如果是建立新连接（新数据udp）请求，则根据套接字类型，分别调用handle\_accept、CNotify::notify\_recv、handle\_accept\_udp进行处理
3. 如果是非正常事件，则关闭连接
4. 如果是发送数据（写事件EPOLLOUT），处理发送请求
5. 如果是收取数据（读事件EPOLLIN），处理接收请求
6. **函数sendto**

使用连接池发送数据

1. 函数ctrl，控制接口暂不使用
2. 函数getconn，获取对应的数据流

**3、tsocket.h：**定义了

socket地址类CSocketAddr，该类主要用于对结构体sockaddr\_in的封装，增添了一些常用的操作

类定义中直接实现了以下成员函数：

1. 函数addr返回结构体sockaddr\_in类型成员变量\_addr的地址，转换为sockaddr类型指针
2. 函数addr\_in返回结构体sockaddr\_in类型成员变量的地址，转换为sockaddr\_in类型指针
3. 函数length返回结构体sockaddr\_in类型成员变量\_addr的长度
4. 函数get\_numeric\_ipv4返回ipv4格式的IP地址
5. 函数set\_numeric\_ipv4设置ipv4格式的IP地址
6. 函数get\_family返回\_addr的协议族
7. 函数set\_family设置\_addr的协议族

tcp/udp/unixsocket通讯类CSocket主要封装了一系列对套接字进行操作的静态成员函数

**4、tsocket.cpp：**实现类CSocketAddr，类CSocket

CPP文件中，实现类CSocketAddr以下两个静态成员函数

1. 函数in\_n2s

转换网址地址为字符串

1. 函数in\_s2n

转换字符串为网络地址

类CSocket实现以下

1. 函数create

创建套接字

1. 函数close

关闭套接字

1. 函数bind

绑定套接字的IP和端口，绑定unix本地套接字

1. 函数bind\_any

绑定套接字的任意IP和指定端口

1. 函数listen

监听socket

1. 函数accept

接收连接

1. 函数connect

发起连接（远程网络地址或本地套接字）

1. 函数receive

接收数据（TCP或UDP）

1. 函数send

发送数据（TCP或UDP）

1. 函数shutdown

关闭连接

1. 函数get\_peer\_name

获取对端地址和端口

1. 函数get\_sock\_name

获取本地地址和端口

1. 函数set\_reuseaddr

设置地址重用

1. 函数set\_nonblock

设置socket非阻塞

1. 函数set\_timeout

设置接收和发送的超时时间

1. 函数set\_recvbuf

设置socket对应的接收数据的缓冲区大小

**5、tmempool.h: 定义类CMemPool**

**类CMemPool用于分配和回收内存，此处在类体中定义了对应的操作函数。**

**6、tmempool.cpp:** 实现类CMemPool

**实现以下两个函数：**

1. **函数allocate**

调用malloc分配内存

1. **函数recycle**

调用free释放内存

**7、myepoll.h：**定义为了兼容老系统的epoll三大主要函数，因为兼容老系统, glibc没有直接的epoll接口

**8、tepollflow.h:** 定义类CEPollFlow，类CEPollFlowResult

类CEPollFlow定义了socket对应的epoll操作，它封装了3个成员变量：套接字\_fd，注册事件结构体指针\_events，和最大监听事件数\_maxFD，它封装的对epoll的操作函数有create、ctl、add、modify、del和wait。

类CEPollFlowResult定义了对epoll事件通知结果的封装，在类的内部还定义了一个iterator类，专门用于对event事件数组进行访问的方法

9、**tepollflow.cpp: 实现**类CEPollFlow，类CEPollFlowResult

在本文件中，类CEPollFlow实现了以下成员函数：

1. 函数create：调用epoll\_create函数创建epoll，并new出对应参数个事件的event数组
2. 函数wait：调用epoll\_wait函数，将返回的event数组和事件个数生成对应的CEPollFlowResult类对象
3. 函数ctl：调用epoll\_ctl函数，完成对对应fd的事件修改（MOD、ADD、DEL）

在本文件中，类CEPollFlowResult中的类iterator实现了以下成员函数：

1. 函数flow：返回事件通知数组中某个元素对应的连接ID
2. 函数fd：返回事件通知数组中某个元素对应的套接字 fd

**10、tconnset.h:** 定义连接池集合类CConnSet

定义了以下函数

1. 函数dumpinfo：打印flow所连接的信息
2. 函数addconn：添加连接
3. 函数fd：查询fd
4. 函数recv：接收数据
5. 函数send：发送数据
6. 函数sendfromcache：从缓冲区接收数据
7. 函数closeconn：关闭连接
8. 函数check\_expire：检查连接是否过期
9. 函数getusedflow：获取当前使用的flow个数
10. 函数getcc：查找连接对应的cache

封装的成员函数如下：

1. ConnCache指针数组类型成员变量：ccs\_
2. 建立的最大连接数：maxconn\_
3. 当前使用的flow：usedflow\_

**11、tconnset.cpp:** 实现连接池集合类CConnSet

**12、tcache.h: 定义**纯cache对象类CRawCache，连接对象cache类

**13、tcache.cpp: 实现**纯cache对象类CRawCache，连接对象cache类