CProConsole ConnectAcceptor

这个类负责处理所有的Console(远程管理客户端)的accept对象

```
//Proactor创建关联客户端连接
CProConsoleHandle *make_handler();
//Proactor模式下连接有效性检查
int validate_connection();
```

CProServerManager

这个类负责框架初始化所有的Proactor（windows下默认模式）相关线程和线程池
App_ProServerManager(全局变量)

```
//初始化对象池和一些配置参数
bool Init();
//初始化启动监听连接，初始化工作线程
bool Start();
 //关闭所有内存池以及工作线程
bool Close();
```

CProConnectClient

负责处理连接服务器间连接的连接对象类，一个ServerID对应一个这个类

```
//用户建立一个链接
void open();
//接受用户数据
void handle_read_stream();
//发送用户数据
void handle_write_stream();
//关闭服务期间连接
void Close();
//服务器关闭远程服务器连接
void ClientClose();
//设置当前的ServerID
void SetServerID();
//设置消息接收处理类
void SetClientMessage();
//获得当前的ServerID
int GetServerID();
//发送数据
bool SendData();
//得到当前链接信息
_ClientConnectInfo
GetClientConnectInfo();
```

CProConnectManager

这个类负责处理所有监听客户端连接（可能是多个监听，通过配置文件获得）的accept，一个监听一个。

```
//初始化所有的accept
InitConnectAcceptor();
//关闭所有的accept
void Close();
 //得到当前accept的数量
int GetCount();
//得到指定的accept
ProConnectAcceptor* GetConnectAcceptor();
//得到错误描述
const char* GetError();
```

CProAsyncConnect

这个类负责处理所有服务器间连接的accept对象

```
//初始化对象池和一些配置参数
bool Init();
//初始化启动监听连接，初始化工作线程
bool Start();
 //关闭所有内存池以及工作线程
bool Close();
```

CProConsoleHandle

负责处理Console远程客户端管理连接

```
//接收远程管理客户端连接请求
void open();
//接收数据块
void handle_read_stream();
//发送数据块
void handle_write_stream();
//获得远程连接地址
void addresses();
//发送数据信息
bool SendMessage();
//连接关闭
bool Close();
//服务器关闭连接
bool ServerClose();
//得到连接错误信息
const char* GetError();
//得到当前连接状态
uint8 GetConnectState();
//得到缓冲中是否有待发送的数据
uint8 GetSendBuffState();
```

CProactorClientInfo

这个类负责处理服务器间连接的客户端相关方法

```
//初始化一个服务器间连接
bool Init();
//开始建立远程连接
bool Run();
 //发送指定的数据块到远端服务器
bool SendData();
//得到指定的连接错误
bool ConnectError();
//得到指定的远程服务器连接ID
int  GetServerID();
//关闭当前远程连接
bool Close();
//一旦连接成功，设置相关关联Handle指针
void SetProConnectClient();
//得到相关关联指针
CProConnectClient* GetProConnectClient();
//获得当前消息处理指针
IClientMessage* GetClientMessage();
//获得远端服务器的地址信息
ACE_INET_Addr GetServerAddr();
```

CClientProactorConnectManager

这个类负责管理所有服务器间连接信息
全局变量App_ClientProConnectManager

```
//初始化Manager，配置一个反应器
bool Init();
//设置一个远程的服务器连接信息(TCP)
bool Connect();
 //设置一个远程的服务器连接信息(UDP)
bool ConnectUDP();
//重练某一个指定的ServerID
bool ReConnect();
//由框架本身逻辑关闭指定的服务器间连接
bool CloseByClient();
//关闭指定的服务器间连接
bool Close();
//关闭指定的服务器间UDP连接
bool CloseUDP();
//远程连接服务器断开会调用此方法。
bool ConnectErrorClose();
//往指定的ServerID连接发送数据(TCP)
bool SendData();
//往指定的ServerID连接发送数据(UDP)
bool SendDataUDP();
//将指定的CProConnectClient*绑定给nServerID
bool SetHandler();
//获得ClientMessage对象
IClientMessage* GetClientMessage();
//获得ClientMessage对象
bool StartConnectTask();
//设置自动重连的检查定时器
bool StartConnectTask();
//关闭重连定时器
void CancelConnectTask();
//关闭所有连接
void Close();
//得到指定的ServerID当前连接状态
bool GetConnectState();
//返回当前存活链接的信息（TCP）
void GetConnectInfo();
//返回当前存活链接的信息（UDP）
void GetUDPConnectInfo();
//定时检测当前连接，已经断开的连接重新连接
int handle_timeout();
```

CProactorHandle

处理接收客户端数据和发送数据的类

```
//远程建立一个链接
void open();
//处理接受到用户数据包信息事件
void handle_read_stream();
//处理发送到用户数据完成的事件
void handle_write_stream();
//获得当前远程客户端的IP地址信息
void addresses();
//Connect Pool初始化调用的函数
void Init();
//检测当前链接是否超时的函数
bool CheckAlive();
//发送给客户端数据的函数
bool SendMessage();
//客户端连接关闭
bool Close();
//服务器主动断开连接
bool ServerClose();
//得到当前错误信息
GetError();
//设置当前链接的ID(自增量)
SetConnectID();
//得到当前的连接ID
GetConnectID();
//得到链接状态
GetConnectState();
//得到目前缓冲区有没有需要发送的数据
GetSendBuffState();
//得到当前连接的相关信息(远程监控功能)
GetClientInfo();
//得到客户端连接的IP相关信息
GetClientIPInfo();
```

CProConnectManager

管理当前指定连接组的所有客户端连接，管理CProactorHandle的集合

```
//Manager类启动相关Init部分。
int open();
//关闭所有连接
void CloseAll();
//添加一个客户端远程连接(CProactorhandle类)
bool AddConnect();
//往指定的客户端同步发送数据
bool SendMessage();
//往指定的客户端异步发送数据
bool PostMessage();
//往当前所有活跃的客户端连接发送数据
bool PostMessageAll();
//指定的客户端连接关闭
bool Close();
//服务器关闭指定的客户端连接
bool CloseConnect();
//返回当前存活链接的信息
void GetConnectInfo();
//记录指定链接数据处理时间
void SetRecvQueueTimeCost();
//得到指定链接信息
_ClientIPInfo GetClientIPInfo();
//得到当前客户端连接的个数
int GetCount()
```

CProConnectManagerGroup

管理CProConnectManager的集合，根据配置文件，按照Connect的最后一位取余，建立若干个CProConnectManager，用于同步发送，提高效率

```
//当连接添加的时候，在这里分配到指定的
CProConnectManager
bool AddConnect();
//向指定的和群组的ConnectID发送数据
bool PostMessage();
//向当前所有的客户端连接发送数据
bool PostMessageAll();
//关闭指定的客户端连接
bool CloseConnect();
//得到指定的客户端连接信息
_ClientIPInfo GetClientIPInfo();
//返回当前的存活连接
void GetConnectInfo();
//得到当前连接的全部个数
int GetCount();
//关闭所有已有连接
void CloseAll();
//客户端关闭指定连接
bool Close();
```

CProConnectHandlerPool

负责初始化所有的连接对象，并用池的方式管理所有连接。

```
//初始化指定数量的ProConnectHandler
void Init();
//关闭对象池
void Close();
//从对象池找出一个没有用的Connect对象
CProConnectHandle* Create();
//归还指定的连接对象到池中
bool Delete();
//得到当前已经使用的connect对象个数
int GetUsedCount();
//得到当前还没使用的connect对象个数
int GetFreeCount();
```

CConnectAcceptor

这个类负责处理所有的Console(远程管理客户端)的accept对象

```
//创建关联客户端连接
int make_svc_handler();
```

CServerManager

这个类负责框架初始化所有的Proactor（windows下默认模式）相关线程和线程池App_ProServerManager(全局变量)

```
//初始化对象池和一些配置参数
bool Init();
//初始化启动监听连接，初始化工作线程
bool Start();
 //关闭所有内存池以及工作线程
bool Close();
```

CConnectClient

负责处理连接服务器间连接的连接对象类，一个ServerID对应一个这个类

```
//用户建立一个链接
void open();
//接受用户数据
int handle_input();
//连接关闭事件
int handle_close();
//关闭服务期间连接
void Close();
//服务器关闭远程服务器连接
void ClientClose();
//设置当前的ServerID
void SetServerID();
//设置消息接收处理类
void SetClientMessage();
//获得当前的ServerID
int GetServerID();
//发送数据
bool SendData();
//得到当前链接信息
_ClientConnectInfo
GetClientConnectInfo();
```

CConsoleHandle

负责处理Console远程客户端管理连接

```
//接收远程管理客户端连接请求
void open();
//接收数据块
int handle_input();
//连接关闭
int handle_close();
//获得远程连接地址
void addresses();
//发送数据信息
bool SendMessage();
//连接关闭
bool Close();
//服务器关闭连接
bool ServerClose();
//得到连接错误信息
const char* GetError();
//得到当前连接状态
uint8 GetConnectState();
//得到缓冲中是否有待发送的数据
uint8 GetSendBuffState();
```

CConnectAcceptorManager

这个类负责处理所有的Console(远程管理客户端)的accept对象

```
//创建指定数量的监听器
bool InitConnectAcceptor(int nCount);
//关闭所有accept
void Close();
//得到当前accept的个数
int GetCount();
//得到指定的accept对象
ConnectAcceptor* GetConnectAcceptor
();
//得到当前accept对象是错误信息
const char* GetError();
```

CReactorClientInfo

这个类负责处理服务器间连接的客户端相关方法

```
//初始化一个服务器间连接
bool Init();
//开始建立远程连接
bool Run();
 //发送指定的数据块到远端服务器
bool SendData();
//得到指定的连接错误
bool ConnectError();
//得到指定的远程服务器连接ID
int  GetServerID();
//关闭当前远程连接
bool Close();
//一旦连接成功，设置相关关联Handle指针
void SetProConnectClient();
//得到相关关联指针
CProConnectClient* GetProConnectClient();
//获得当前消息处理指针
IClientMessage* GetClientMessage();
//获得远端服务器的地址信息
ACE_INET_Addr GetServerAddr();
```

CClientReConnectManager

这个类负责管理所有服务器间连接信息
全局变量App_ClientProConnectManager

```
//初始化Manager，配置一个反应器
bool Init();
//设置一个远程的服务器连接信息(TCP)
bool Connect();
 //设置一个远程的服务器连接信息(UDP)
bool ConnectUDP();
//重练某一个指定的ServerID
bool ReConnect();
//由框架本身逻辑关闭指定的服务器间连接
bool CloseByClient();
//关闭指定的服务器间连接
bool Close();
//关闭指定的服务器间UDP连接
bool CloseUDP();
//远程连接服务器断开会调用此方法。
bool ConnectErrorClose();
//往指定的ServerID连接发送数据(TCP)
bool SendData();
//往指定的ServerID连接发送数据(UDP)
bool SendDataUDP();
//将指定的CProConnectClient*绑定给nServerID
bool SetHandler();
//获得ClientMessage对象
IClientMessage* GetClientMessage();
//获得ClientMessage对象
bool StartConnectTask();
//设置自动重连的检查定时器
bool StartConnectTask();
//关闭重连定时器
void CancelConnectTask();
//关闭所有连接
void Close();
//得到指定的ServerID当前连接状态
bool GetConnectState();
//返回当前存活链接的信息（TCP）
void GetConnectInfo();
//返回当前存活链接的信息（UDP）
void GetUDPConnectInfo();
//定时检测当前连接，已经断开的连接重新连接
int handle_timeout();
```

CConnectHandler

处理接收客户端数据和发送数据的类

```
//远程建立一个链接
void open();
//处理接受到用户数据包信息事件
void handle_input();
//处理连接断开事件
void handle_close();
//获得当前远程客户端的IP地址信息
void addresses();
//Connect Pool初始化调用的函数
void Init();
//检测当前链接是否超时的函数
bool CheckAlive();
//发送给客户端数据的函数
bool SendMessage();
//客户端连接关闭
bool Close();
//服务器主动断开连接
bool ServerClose();
//得到当前错误信息
GetError();
//设置当前链接的ID(自增量)
SetConnectID();
//得到当前的连接ID
GetConnectID();
//得到链接状态
GetConnectState();
//得到目前缓冲区有没有需要发送的数据
GetSendBuffState();
//得到当前连接的相关信息(远程监控功能)
GetClientInfo();
//得到客户端连接的IP相关信息
GetClientIPInfo();
```

CConnectManager

管理当前指定连接组的所有客户端连接，管理CProactorHandle的集合

```
//Manager类启动相关Init部分。
int open();
//关闭所有连接
void CloseAll();
//添加一个客户端远程连接(CProactorhandle类)
bool AddConnect();
//往指定的客户端同步发送数据
bool SendMessage();
//往指定的客户端异步发送数据
bool PostMessage();
//往当前所有活跃的客户端连接发送数据
bool PostMessageAll();
//指定的客户端连接关闭
bool Close();
//服务器关闭指定的客户端连接
bool CloseConnect();
//返回当前存活链接的信息
void GetConnectInfo();
//记录指定链接数据处理时间
void SetRecvQueueTimeCost();
//得到指定链接信息
_ClientIPInfo GetClientIPInfo();
//得到当前客户端连接的个数
int GetCount()
```

CConnectManagerGroup

管理CProConnectManager的集合，根据配置文件，按照Connect的最后一位取余，建立若干个CProConnectManager，用于同步发送，提高效率

```
//当连接添加的时候，在这里分配到指定的
CProConnectManager
bool AddConnect();
//向指定的和群组的ConnectID发送数据
bool PostMessage();
//向当前所有的客户端连接发送数据
bool PostMessageAll();
//关闭指定的客户端连接
bool CloseConnect();
//得到指定的客户端连接信息
_ClientIPInfo GetClientIPInfo();
//返回当前的存活连接
void GetConnectInfo();
//得到当前连接的全部个数
int GetCount();
//关闭所有已有连接
void CloseAll();
//客户端关闭指定连接
bool Close();
```

CLoadModule

这个类负责动态加载模块中的方法和事件,全局变量App_ModuleLoader

```
//删除所有注册插件
void Close();
//加载指定的插件
bool LoadModule();
//删除指定的插件
bool UnLoadModule();
//把一个插件的消息转发给另一个插件
int SendModuleMessage();
//得到当前插件个数
int GetCurrModuleCount();
//得到指定的插件相关加载信息
_ModuleInfo* GetModuleIndex();
//根据插件的名称得到某一插件的相关加载信息
_ModuleInfo* GetModuleInfo();
```

CMessage

这个类负责运载消息给插件注册的相关消息处理类

```
//清理消息中的数据
void Close();
//删除消息中的相关消息内容（与消息池匹配）
void Clear();
//设置消息的来源信息，比如IP，端口等
void SetMessageBase();
//把一个消息转换成BuffPacket对象
bool SetRecvPacket();
//得到消息头数据块
ACE_Message_Block* GetMessageHead();
//得到一个消息体数据块
ACE_Message_Block* GetMessageBody();
//得到这个消息的来源对象
_MessageBase* GetMessageBase();
//得到消息头并转换成_PacketInfo对象
bool GetPacketHead();
//得到一个消息体并转换成PacketInfo对象
bool GetPacketBody();
//将一个数据块给消息头
bool SetPacketHead();
//将一个数据块给消息体
bool SetPacketBody();
```

CMessageManager

这个类负责处理所有插件的注册消息，属于消息的发布者，将消息传递给模块的订阅者。全局变量App_MessageManager

```
//调用插件内的消息处理模块
bool DoMessage();
//关闭所有注册消息
void Close();
//建立一个消息和插件的关联关系
bool AddClientCommand();
//删除一个指定的消息和插件的关联关系
bool DelClientCommand();
//消除一个消息和插件的关系
bool UnloadModuleCommand();
//得到注册消息模块的个数
int  GetCommandCount();
//得到当前命令的执行列表
CClientCommandList* GetClientCommandList
();
//返回所有模块绑定注册命令信息
mapModuleClient* GetModuleClient();
```

CMessageService

这个类是消息处理的线程类，用于处理消息的工作线程

```
//工作线程初始化
int open();
//工作线程初始化信息
bool Init();
//将工作线程的状态计入日志
bool SaveThreadInfo();
//工作线程启动
bool Start();
//将消息放入工作线程队列
bool PutMessage();
//得到当前的工作线程信息
CThreadInfo* GetThreadInfo();
```

CMessageServiceGroup

管理所有的工作线程类,工作线程的个数由配置文件决定，全局变量App_MessageServiceGroup

```
//定时检测所有的工作线程状态
int handle_timeout();
//工作线程初始化信息
bool Init();
//工作线程启动
bool Start();
//将消息放入工作线程队列
bool PutMessage();
//得到指定的工作线程信息
CThreadInfo* GetThreadInfo();
```

CModuleMessageManager

管理所有的模块间命令调用信息，全局变量App_ModuleMessageManager

```
//将指定的信息发给某一个信息。
int SendModuleMessage();
```

CLogFile

负责记录日志的类，按照日志类型，记录相关的信息。

```
//记录日志，每天会自动生成一个新文件
int doLog();
//得到当前日志名称
ACE_TString& GetLoggerName();
//设置日志类型
void SetLoggerClass();
//得到日志类型
int GetLoggerType();
//设置日志名称
void SetServerName();
//日志对象初始化
bool Run();
```

CFileLogger

负责管理所有的日志对象

```
//记录日志
int DoLog();
//得到当前日志个数
int GetLogTypeCount();
//得到指定日志对象的类型
int GetLogType();
//初始化管理类
bool Init();
//删除所有的日志对象
bool Close();
```

CLogManager

日志线程类，为了支持多线程日志写入，这里采用了消息队列的方式。

```
//日志线程初始化
int open ();
//日志线程运行
int svc ();
//关闭日志线程
int Close();
//初始化日志线程，读取配置文件
void Init();
//将日志信息放入消息队列
int PutLog();
//注册日志对象
int RegisterLog();
//注销日志对象
int UnRegisterLog();
//记录日志信息(外部调用)
int WriteLog();
```