源码项目分析

# 三大神器

每一个神器都会有一个启动服务startService 和一个停止服务 stopService；

首先，每一个神器都会拥有各种引擎实例（引擎helper）和钩子实例,不同的神器会拥有不同的引擎实例和钩子实例。钩子实例中也会拥有各种引擎的指针

每个神器在startService中，首先得到引擎钩子实例，然后以神器中的引擎为参数设置钩子实例中的引擎参数。让神器和钩子实例拥有同样的引擎

首先，启动中心服务器，启动完成后，中心服务器进入等待状态

其次，启动登陆服务器，启动后，发起连接中心服务器的动作，连接成功后，由内核调度引擎调用登陆服务器的调度钩子，CAttemperEngineSink::OnEventTCPSocketLink，在该方法中

//获取列表

m\_pITCPSocketCorrespond->SendData(MDM\_CS\_SERVER\_LIST,SUB\_CS\_GET\_SERVER\_LIST,NULL,0);

//实现定时器启动，定时器启动后，会定期去更新服务器列表信息，也就是去执行上面的方法。

读取服务器列表服务的过程（此处略去网络传输部分）

CAttemperEngineSink::OnEventTCPNetworkRead 被调用，该方法中处理如下两种命令

case MDM\_CS\_SERVER\_LIST: //列表消息

case MDM\_CS\_SERVER\_MANAGER: //房间管理

中心服务器处理完毕后，发送结果到登陆服务器，登陆服务器以下方法会被调用

CAttemperEngineSink::OnEventTCPSocketRead（）

该方法处理case MDM\_CS\_SERVER\_LIST: 命令 //列表消息

return OnCenterMainServerList(Command.wSubCmdID,pData,wDataSize);

房间注册

过程： ON\_MESSAGE(WM\_SOCKET\_NOTIFY, OnSocketNotify)内核引擎的网络部分定义了响应函数OnSocketNotify,内核调度引擎的数据队列钩子接口被调用， CAttemperEngine::OnQueueServiceSink，根据调用的类型进行处理

登陆服务器调动引擎钩子CAttemperEngineSink::OnEventTCPSocketLink方法被调用

再来看该方法的内部结构，

//注册房间

CMD\_CS\_RegGameServer 设置游戏服务器参数，然后调用以下方法发送到客户端

m\_pITCPSocketCorrespond->SendData(MDM\_CS\_SERVER\_MANAGER,SUB\_CS\_REG\_GAME\_SERVER,&RegGameServer,sizeof(RegGameServer));

中心服务器调度钩子引擎实现中，CAttemperEngineSink::OnEventTCPNetworkRead

被调用，由于当前类型是MDM\_CS\_SERVER\_MANAGER，因此OnSocketMainServerManager方法被执行 ，该方法处理三种类型的消息

#define SUB\_CS\_REG\_GAME\_SERVER 1 //注册房间

#define SUB\_CS\_UNREG\_GAME\_SERVER 2 //注销房间

#define SUB\_CS\_SERVER\_ONLINE\_COUNT 3 //房间人数

由于是第一次登陆

## 中心服务器

### 中心服务类CCenterService结构

//组件变量

protected:

CDataBaseSink m\_DataBaseEngineSink; //数据库钩子

CAttemperEngineSink m\_AttemperEngineSink; //调度钩子

//内核组件

protected:

CAttemperEngineHelper m\_AttemperEngine; //调度引擎

CDataBaseEngineHelper m\_DataBaseEngine; //数据引擎

CTCPNetworkEngineHelper m\_TCPNetworkEngine; //网络引擎

* 三个引擎：**调度引擎、数据库引擎、网络引擎**；

调度引擎 CAttemperEngineHelper m\_AttemperEngine;

数据库引擎 CDataBaseEngineHelper m\_DataBaseEngine;

网络引擎 CTCPNetworkEngineHelper m\_TCPNetworkEngine;

数据库引擎钩子：DataBaseSink.h（中心服务器项目实现）

调度引擎钩子：AttemperEngineSink.h（中心服务器项目实现）

网络引擎 CTCPNetworkEngine（在内核引擎的网络引擎中实现）

数据库钩子：中心服务器中，未实现DataBaseSink.h描述的方法

* 两个钩子： **数据钩子、调度钩子**

数据库钩子 CDataBaseSink m\_DataBaseEngineSink;

调度钩子 CAttemperEngineSink m\_AttemperEngineSink;

### 启动流程

中心服务器启动从以下方法中开始

//开始服务bool CCenterService::StartService() ，定义在CenterService.h文件中

1. 检验服务运行状态
2. 加载参数
3. 设置钩子外挂
4. 配置网络端口以及最大连接数
5. 启动三大引擎，根据设置的钩子外挂去执行相应的功能，如启动、停止等，等待事件处理

### 中心服务器引擎钩子模块分析

#### 调度引擎钩子

调度引擎钩子模块：三大引擎接口、服务列表

//接口变量

protected:

ITimerEngine \* m\_pITimerEngine; //时间引擎

IDataBaseEngine \* m\_pIDataBaseEngine; //数据引擎

ITCPNetworkEngine \* m\_pITCPNetworkEngine; //网络引擎

//组件定义

protected:

CServerListCenterHelper m\_ServerListCenter;

CServerListCenterHelper 定义在“列表服务”项目下，主要定义了，关于游戏类型，房间、游戏种类，上线人数等操作。

从内核调度引擎执行的时候，会用到登录服务器结构中的引擎钩子

钩子引擎的设置是根据中心服务器的引擎helper对象的getInstance()方法来设置

调度引擎启动：

1. 创建服务列表组件

m\_ServerListCenter，是从“列表服务”项目项目读取COM接口并创建实例。

1. 加载服务列表项

tagDataBaseInfo DataBaseInfo; 定义数据库信息结构在 公共文件 GlobalService.h中定义。

if (m\_ServerListCenter->LoadServerList(DataBaseInfo)==false)

LoadServerList 第 2章节详细介绍。

#### 数据库引擎钩子

数据库引擎，在中心服务器调度引擎钩子实现中，通过startService方法设置到中心服务器中。

#### 调动引擎回调方法

以下两个方法由内核调度引擎发起，由中心服务器的调度钩子中被网络响应接口调用

//列表消息处理

bool OnSocketMainServerList(WORD wSubCmdID, VOID \* pData, WORD wDataSize, DWORD dwSocketID);

主要流程如下：

读取m\_ServerListCenter中存储的类型信息，参考第2章节描述；

m\_pITCPNetworkEngine->SendData，调用网络引擎发送数据，SendData参见第6章，内核网络引擎部分，SendData主要是根据子命令ID发送列表类型信息。

//房间消息处理

bool OnSocketMainServerManager(WORD wSubCmdID, VOID \* pData, WORD wDataSize, DWORD dwSocketID);

主要流程如下：

## 登录服务器

### 登录服务类CLogonService结构

//核心结构

protected:

CDataBaseSink m\_DataBaseEngineSink; //数据钩子

CAttemperEngineSink m\_AttemperEngineSink; //调度钩子

//内核组件

protected:

CTimerEngineHelper m\_TimerEngine; //时间引擎

CAttemperEngineHelper m\_AttemperEngine; //调度引擎

CDataBaseEngineHelper m\_DataBaseEngine; //数据引擎

CTCPNetworkEngineHelper m\_TCPNetworkEngine; //网络引擎

CTCPSocketServiceHelper m\_TCPSocketCorrespond; //协调服务

* 两个钩子： **数据库钩子、调度钩子**

CDataBaseSink m\_DataBaseEngineSink; //数据库钩子

CAttemperEngineSink m\_AttemperEngineSink; //调度钩子

* 四大引擎

CTimerEngineHelper m\_TimerEngine; //时间引擎

CAttemperEngineHelper m\_AttemperEngine; //调度引擎

CDataBaseEngineHelper m\_DataBaseEngine; //数据库引擎

CTCPNetworkEngineHelper m\_TCPNetworkEngine; //网络引擎

此四大引擎都是在内核中实现

* 一个服务

CTCPSocketServiceHelper m\_TCPSocketCorrespond; //协调服务

此服务在内核中实现，与网络引擎同目录下

### 启动流程

登录服务器启动从以下方法中开始

#### 启动服务

//开始服务bool CLogonService::StartService() ，定义在CenterService.h文件中

1. 检验服务运行状态
2. 加载参数
3. 设置绑定钩子外挂
4. 启动三大引擎

时间引擎、网络引擎、调度引擎、数据库引擎

调用内核引擎实现启动；

1. 启动一个服务

网络服务，调用内核引擎实现启动；

#### 响应连接

CLogonService::StartService启动完毕后，内核引擎会调用钩子引擎实现的OnAttemperEngineStart方法，在该方法中，连接中心服务器，中心服务器接收成功后，OnEventTCPSocketLink方法会被调用，在该方法中设置定时器从中心服务器上更新列表信息。

### 登录服务器引擎钩子模块分析

#### 调度引擎钩子

调度引擎钩子模块：三大引擎接口、服务器列表、一个服务

//辅助组件

protected:

CServerList m\_ServerList; //服务器列表

//接口变量

protected:

ITimerEngine \* m\_pITimerEngine; //时间引擎

IDataBaseEngine \* m\_pIDataBaseEngine; //数据引擎

ITCPNetworkEngine \* m\_pITCPNetworkEngine; //网络引擎

ITCPSocketService \* m\_pITCPSocketCorrespond; //协调服务

从内核调度引擎执行的时候，会用到登录服务器结构中的引擎钩子

钩子引擎的设置是根据登录服务器的引擎helper对象的getInstance()方法来设置

#### 数据库引擎钩子

数据库引擎，在登陆服务器调度引擎钩子实现中，通过startService方法设置到登陆服务器中。

## 服务装载器

服务装载器仅仅是一个外壳，核心部分是“游戏服务”工程项目，它才是服务装载器的心脏。

### 游戏服务类CGameService

class CGameService : public IGameService，核心结构如下

protected:

CDataBaseSink m\_DataBaseEngineSink; //数据库钩子

CAttemperEngineSink m\_AttemperEngineSink; //调度钩子

//内核组件

protected:

CTimerEngineHelper m\_TimerEngine; //时间引擎

CAttemperEngineHelper m\_AttemperEngine; //调度引擎

CDataBaseEngineHelper m\_DataBaseEngine; //数据引擎

CTCPNetworkEngineHelper m\_TCPNetworkEngine; //网络引擎

CTCPSocketServiceHelper m\_TCPSocketCorrespond; //协调服务

//配置变量

protected:

TCHAR m\_szModule[32]; //模块名字

tagDataBaseInfo m\_GameUserDBInfo; //数据配置

tagDataBaseInfo m\_GameScoreDBInfo; //数据配置

tagGameServiceAttrib m\_GameServiceAttrib; //服务属性

tagGameServiceOption m\_GameServiceOption; //服务配置

//组件接口

protected:

HINSTANCE m\_hDllInstance; //游戏实例

IGameServiceManager \* m\_pIGameServiceManager; //管理接口

结构分析：

* 两个钩子： **数据钩子、调度钩子**

数据库钩子 CDataBaseSink m\_DataBaseEngineSink;

调度钩子 CAttemperEngineSink m\_AttemperEngineSink;

* 四大引擎

CTimerEngineHelper m\_TimerEngine; //时间引擎

CAttemperEngineHelper m\_AttemperEngine; //调度引擎

CDataBaseEngineHelper m\_DataBaseEngine; //数据库引擎

CTCPNetworkEngineHelper m\_TCPNetworkEngine; //网络引擎

在helper 以Com组件方式使用内核引擎，通过

DataBaseEngine::SetDataBaseEngineSink 设置数据库钩子的具体实现

CAttemperEngine::SetAttemperEngineSink设置调度钩子的具体实现

CQueueService::SetQueueServiceSink 设置数据队列钩子的具体实现

* 一个服务

CTCPSocketServiceHelper m\_TCPSocketCorrespond; //协调服务

此服务在内核中实现，与网络引擎同目录下

* 配置变量

游戏模块名称、用户数据库、积分数据库、游戏服务器属性、游戏服务器配置

* 组件接口

游戏实例句柄、IGameServiceManager游戏服务管理接口

### 启动流程

#### 启动加载服务器外壳

从外壳中加载游戏组件，该外壳可以使得游戏的加载跟游戏组件分离，增强了扩展性

#### 启动游戏服务组件

装载服务器调用“游戏服务”组件，启动从以下方法中开始

##### 启动游戏服务

//开始服务bool CGameService::StartService() ，定义在CenterService.h文件中

1. 检验服务运行状态
2. 创建引擎实例
3. 加载游戏服务模块组件
4. 设置绑定钩子外挂
5. 启动四大引擎

时间引擎、网络引擎、调度引擎、数据库引擎

调用内核引擎实现启动；

1. 启动一个服务

网络服务，调用内核引擎实现启动；

##### 游戏服务消息响应

内核调度引擎会在启动调度钩子实现OnAttemperEngineStart方法，该方法实现初始化操作，具体参考第1.3.3节内容。

### 游戏服务引擎钩子模块

#### 调度引擎钩子

//变量定义

protected:

WORD m\_wMaxUserItem; //最大用户

CTableFrame \* m\_pTableFrame; //桌子指针

tagConnectItemInfo \* m\_pNormalParameter; //辅助信息

tagConnectItemInfo \* m\_pAndroidParameter; //辅助信息

//发送场所

protected:

BYTE m\_cbSendLocation; //发送场所

//消息变量

protected:

BYTE m\_cbRoomMsgIndex; //消息索引 BYTE m\_cbGameMsgIndex; //消息索引 //控制变量

protected:

bool m\_bShallClose; //关闭标志

bool m\_bAllowWisper; //私聊标志

bool m\_bAllowRoomChat; //聊天标志

bool m\_bAllowGameChat; //聊天标志

bool m\_bAllowEnterRoom; //进入标志

bool m\_bAllowEnterGame; //进入标志

//配置信息

protected:

DWORD m\_dwCenterServer; //中心地址

tagGameServiceAttrib \* m\_pGameServiceAttrib; //服务属性

tagGameServiceOption \* m\_pGameServiceOption; //服务配置

//游戏组件

protected:

HINSTANCE m\_hDllInstance; //游戏实例

IGameServiceManager \* m\_pIGameServiceManager; //管理接口

//列表解释

protected:

CMD\_GR\_ColumnInfo m\_ColumnInfo; //列表解释

WORD m\_wColumnInfoSize; //列表长度

//在线人数

protected:

WORD m\_wOnLineInfoSize; //信息长度

tagOnLineCountInfo m\_OnLineCountInfo[512]; //在线信息

//组件变量

protected:

CServerList m\_ServerList; //房间列表

CServerUserManager m\_ServerUserManager; //用户管理

CAndroidUserManager m\_AndroidUserManager; //用户管理

//接口变量

protected:

ITimerEngine \* m\_pITimerEngine; //时间引擎

IDataBaseEngine \* m\_pIDataBaseEngine; //数据引擎

IAttemperEngine \* m\_pIAttemperEngine; //调度引擎

ITCPNetworkEngine \* m\_pITCPNetworkEngine; //网络引擎

ITCPSocketService \* m\_pITCPSocketCorrespond; //协调服务

从内核调度引擎执行的时候，会用到登录服务器结构中的引擎钩子

钩子引擎的设置是根据登录服务器的引擎helper对象的getInstance()方法来设置

游戏服务调度钩子启动过程：CAttemperEngineSink::OnAttemperEngineStart

//设置网络连接参数

//加载游戏组件

m\_pIGameServiceManager->RectifyServiceOption(m\_pGameServiceOption)

//创建游戏桌子

m\_pTableFrame=new CTableFrame [m\_pGameServiceOption->wTableCount];

//初始化桌子

//设置机器人类

//构造列表解释，定义金币房间，比赛房间，积分房间等。

//连接中心服务器

m\_pITCPSocketCorrespond->Connect(ntohl(m\_dwCenterServer),PORT\_CENTER\_SERVER);

//系统消息定时器

m\_pITimerEngine->SetTimer(IDI\_SEND\_SYSTEM\_MESSAGE,300000L,(DWORD)(-1),0);

//启动机器人服务

m\_AndroidUserManager.StartService()

//加载用户，读取用户数据库信息

m\_pIDataBaseEngine->PostDataBaseRequest(DBR\_GR\_LOAD\_ANDROID,0,NULL,0);

//加载道具，读取道具数据库信息

m\_pIDataBaseEngine->PostDataBaseRequest(DBR\_GR\_LOAD\_PROPERTY, 0,NULL, 0);

//分配用户，机器人用户

m\_pITimerEngine->SetTimer(IDI\_DISTRIBUTE\_ANDROID,TIME\_DISTRIBUTE\_ANDROID\*1000L,(DWORD)(-1),NULL);

服务启动完成，等待用户连接

#### 数据库引擎钩子

数据库引擎，在游戏服务器调度引擎钩子实现中，通过startService方法设置到游戏服务器中。

# 列表服务

列表服务项目的定义：

ListService.h IServerListCenter 定义了游戏类型、游戏种类、游戏房间的“增删插改”功能接口

ServerListCenter.h //定义服务器列表类的具体实现

class CServerListCenter : public IServerListCenter

LoadServerList 方法实现：

//创建数据库

CDataBaseAide ServerInfoDBAide;

CDataBaseHelper ServerInfoDBModule;

创建了数据库引擎实例对象和数据库辅助实例

调用ServerInfoDBModule.CreateInstance() 获取内核数据库引擎，连接数据库

公共如下三个方法从数据库中加载游戏类型、游戏种类、游戏站点信息，结构存放在ServerInfoDBAide结构中。

LoadGameType(ServerInfoDBAide);

LoadGameKind(ServerInfoDBAide);

LoadGameStation(ServerInfoDBAide);

# 内核调度引擎

## 调度管理类结构

class CAttemperEngine : public IAttemperEngine, public IQueueServiceSink,

public IDataBaseEngineEvent, public ITCPNetworkEngineEvent

{

//主要核心变量

protected:

bool m\_bService; //运行标志

CQueueService m\_QueueService; //队列对象

CCriticalSection m\_CriticalSection; //同步锁定

//接口变量

protected:

ITCPSocketService \* m\_pITCPSocketService; //网络引擎

ITCPNetworkEngine \* m\_pTCPNetworkEngine; //网络引擎

IAttemperEngineSink \* m\_pIAttemperEngineSink; //挂接接口

//机器人链接维护

protected:

CArrayTemplate<DWORD> m\_AndroidUserSocketIDArray; //机器人连接

## 调度引擎启动过程

调度引擎启动，获取内核调动引擎接口，从内核调动引擎开始执行（服务器组件-> 内核引擎CAttemperEngine::StartService（））；

从CAttemperEngine::StartService() 开始：

1. 判断状态，是否启动；
2. 检查调度引擎外挂钩子设置是否正常，不正常返回；
3. 设置调度引擎与队列服务绑定，设置队列服务外挂钩子；
4. 启动调度引擎外挂钩子；
5. 启动内核引擎下的队列服务；
6. 清空机器人，启动完成，等待事件处理。

## 数据队列接口处理

void \_\_cdecl CAttemperEngine::OnQueueServiceSink(WORD wIdentifier, void \* pBuffer, WORD wDataSize)

队列服务CQueueService 中设置了队列服务钩子，由此钩子产生回调，在回调中根据wIdentifier标识不同的类型进行处理；wIdentifier有以下几种类型

//事件定义

时间事件 #define EVENT\_TIMER 0x0001

控制事件 #define EVENT\_CONTROL 0x0002 数据库事件 #define EVENT\_DATABASE 0x0003

读取事件 #define EVENT\_TCP\_SOCKET\_READ 0x0004

关闭事件 #define EVENT\_TCP\_SOCKET\_CLOSE 0x0005

连接事件 #define EVENT\_TCP\_SOCKET\_CONNECT 0x0006 应答事件 #define EVENT\_TCP\_NETWORK\_ACCEPT 0x0007

读取事件 #define EVENT\_TCP\_NETWORK\_READ 0x0008 关闭事件 #define EVENT\_TCP\_NETWORK\_CLOSE 0x0009

对不属于上面类型的其他定义，丢到调度引擎的钩子实现中去处理，如下

m\_pIAttemperEngineSink->OnEventAttemperData(wIdentifier, pBuffer, wDataSize);

实现钩子的有中心服务器、登录服务器、游戏服务三种，这三个钩子中，均没有实现OnEventAttemperData的处理，也就意味着不会处理不识别的类型。这里只留作将来扩展使用。

下面分别描述不同类型的处理过程

### EVENT\_TIMER 时间事件

直接调用钩子方法中实现的OnEventTimer方法

m\_pIAttemperEngineSink->OnEventTimer(pTimerEvent->dwTimerID, pTimerEvent->dwBindParameter);，有以下三种钩子实现

#### 中心服务器

未实现OnEventTimer的处理

#### 游戏服务

实现如下类型处理，具体的处理请见游戏服务章节的详细分析

case IDI\_CONNECT\_CENTER\_SERVER: //连接中心服务器

case IDI\_UPDATE\_SERVER\_LIST: //更新列表

case IDI\_UPDATE\_ONLINE\_COUNT: //更新人数

case IDI\_SEND\_SYSTEM\_MESSAGE: //系统消息

case IDI\_LOAD\_ANDROID\_INFO: //加载用户

case IDI\_DISTRIBUTE\_ANDROID: //分配用户

#### 登录服务器

实现了两种OnEventTimer事件处理，详细分析请见登录服务器章节

case IDI\_CONNECT\_CENTER\_SERVER: //连接中心服务器定时连接中心服务器，避免连接端口后停止工作。类似看门狗

case IDI\_UPDATE\_SERVER\_LIST: //更新服务器列表

### EVENT\_DATABASE数据库事件

实现了该事件处理钩子的有三种，中心服务器、游戏服务、登录服务器。

#### 中心服务器

未处理

#### 游戏服务

case DBR\_GR\_LOGON\_SUCCESS: //登录成功

case DBR\_GR\_LOGON\_ERROR: //登录失败

case DBR\_GR\_ANDROID\_USER: //机器信息

case DBR\_GR\_USER\_RIGHT\_RESULT: //用户权限

case DBR\_GR\_PROPERTY\_ATTRIBUTE: //加载道具

case DBR\_GR\_FLOWER\_ATTRIBUTE: //加载鲜花

case DBR\_GR\_USER\_PROPERTY: //玩家道具

//case DBR\_GR\_EXCHANGE\_RESULT: //兑换结果

case DBR\_GR\_LOAD\_PROP\_FINISHI: //完成加载

详细处理请看游戏服务章节；

#### 登录服务器。

case DBR\_GP\_LOGON\_ERROR: //登录失败

case DBR\_GP\_LOGON\_SUCCESS: //登录成功

case DBR\_GP\_DOWNLOADFACE\_SUCCESS: //下载成功

case DBR\_GP\_UPLOAD\_FACE\_RESULT: //上传结果

case DBR\_GP\_DELETE\_FACE\_RESULT: //删除结果

case DBR\_GP\_MODIFY\_RESULT: //修改结果

详细分析请见登录服务器章节；

### EVENT\_TCP\_NETWORK\_ACCEPT网络应答事件

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventTCPNetworkBind(pSocketAcceptEvent->dwClientIP, pSocketAcceptEvent->dwSocketID);

#### 中心服务器

绑定网络

#### 游戏服务

绑定网络

#### 登录服务器。

绑定网络

### EVENT\_TCP\_NETWORK\_READ: 网络读取事件

内核调度引擎实现如下调用，三种钩子实现具体方法

m\_pIAttemperEngineSink->OnEventTCPNetworkRead(pSocketReadEvent->Command, pSocketReadEvent + 1, pSocketReadEvent->wDataSize, pSocketReadEvent->dwSocketID);

#### 中心服务器

case MDM\_CS\_SERVER\_LIST: //列表消息

case MDM\_CS\_SERVER\_MANAGER: //房间管理

具体分析，请见中心服务器章节

#### 游戏服务

case MDM\_GR\_LOGON: //登录消息

case MDM\_GR\_USER: //用户消息

case MDM\_GR\_MANAGER://管理消息

case MDM\_GF\_FRAME: //框架消息

case MDM\_GF\_GAME: //游戏消息

case MDM\_GF\_VIDEO: //视频消息

case MDM\_GF\_PRESENT://礼物消息

case MDM\_GF\_BANK: //银行消息

具体分析，请见游戏服务器章节

#### 登录服务器。

case MDM\_GP\_LOGON: //登录消息

case MDM\_GP\_USER: //用户信息

具体分析，请见登录服务器章节

### EVENT\_TCP\_NETWORK\_CLOSE: 关闭读取事件

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventTCPNetworkShut(pSocketCloseEvent->dwClientIP, pSocketCloseEvent->dwActiveTime, pSocketCloseEvent->dwSocketID);

#### 中心服务器

见钩子实现文件

#### 游戏服务

见钩子实现文件

#### 登录服务器。

见钩子实现文件

### EVENT\_TCP\_SOCKET\_READ: 网络读取事件

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventTCPSocketRead(pSocketReadEvent->wServiceID, pSocketReadEvent->Command, pSocketReadEvent + 1, pSocketReadEvent->wDataSize);

#### 中心服务器

未处理该事件

#### 游戏服务

case MDM\_CS\_SERVER\_LIST: //列表消息

具体处理方法见钩子实现文件

#### 登录服务器。

case MDM\_CS\_SERVER\_LIST: //列表消息

具体处理方法见钩子实现文件

### EVENT\_TCP\_SOCKET\_CLOSE: 网络关闭事件

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventTCPSocketShut(pSocketCloseEvent->wServiceID, pSocketCloseEvent->cbShutReason);

#### 中心服务器

未处理该事件

#### 游戏服务

具体处理方法见钩子实现文件

#### 登录服务器。

具体处理方法见钩子实现文件

### EVENT\_TCP\_SOCKET\_CONNECT: 网络连接

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventTCPSocketLink(pSocketCloseEvent->wServiceID, pSocketCloseEvent->nErrorCode);

#### 中心服务器

未处理该事件

#### 游戏服务

具体处理方法见钩子实现文件

#### 登录服务器。

具体处理方法见钩子实现文件

### EVENT\_CONTROL 事件控制

调度引擎中，调用钩子的处理方法，见下面：

m\_pIAttemperEngineSink->OnEventControl(wIdentifier, pBuffer, wDataSize);

#### 中心服务器

未处理该事件

#### 游戏服务

未处理该事件

#### 登录服务器。

未处理该事件

# 内核数据库引擎

1. 检查外挂钩子有效性
2. 设置数据队列钩子

启动数据库外挂钩子

# 内核定时器引擎

由内核引擎的时间引擎接口开始执行 CTimerEngine::StartService（）

# 内核网络引擎

由内核引擎的网络引擎接口开始执行 CTCPNetworkEngine::StartService()

实现结构，定义在内核引擎项目中

class CTCPNetworkEngine : public ITCPNetworkEngine, public IServerSocketItemSink, public IQueueServiceSink

# 内核网络服务

由内核引擎的网络引擎接口开始执行 CTCPSocketService::StartService（）

# 客户端

## 大厅

GamePlaza.h 定义了大厅的APP对象，CGamePlazaApp::InitInstance()，在该方法中，

CGlobalOption::LoadOptionParameter（）加载参数，主要全局设置、房间设置、游戏设置、颜色设置、消息设置、代理设置、游戏配置、房间配置。创建游戏大厅主窗口：CGameFrame \* pMainFrame=new CGameFrame();该类时游戏大厅的窗口主框架。

大厅进程主要包含了两个视图，一个是广场视图，一个是房间视图，后面会详细分析这两个视图，下面分析广场主框架CGameFrame类：

//状态变量

public:

bool m\_bRectify; //调整标志

bool m\_bHideGameList; //隐藏标志

WORD m\_wShowListType; //列表标志

CRect m\_rcViewItem; //窗口位置

CPlazaViewItem m\_DlgGamePlaza; //游戏广场

IViewItem \* m\_pActiveViewItem; //当前视图

CRoomViewItem \* m\_pRoomViewItemCreate; //创建房间

CRoomViewItem \* m\_pRoomViewItem[MAX\_SERVER]; //房间接口

CDlgCustomFace m\_DlgCustomFace; //自定头像

//控件变量

public:

CWebBrowser m\_BrowerAD; //浏览窗口

CToolTipCtrl m\_ToolTipCtrl; //提示控件

CUserInfoView m\_UserInfoView; //游戏视图

CSkinSplitter m\_Splitter; //拆分窗口

CServerItemView m\_ServerItemView; //服务器视图

CUserCompanionList m\_UserCompanionList; //好友关系列表

//界面状态

protected:

bool m\_bMaxShow; //最大标志

CRect m\_rcNormalSize; //正常位置

//导航按钮

public:

CSkinButton m\_btButton1; //功能按钮

CSkinButton m\_btButton2; //功能按钮

CSkinButton m\_btButton3; //功能按钮

CSkinButton m\_btButton4; //功能按钮

CSkinButton m\_btButton5; //功能按钮

//大厅按钮

public:

CSkinButton m\_btMin; //最小按钮

CSkinButton m\_btMax; //最大按钮

CSkinButton m\_btClose; //关闭按钮

CSkinButton m\_btExChangeSkin; //换肤按钮

CSkinButton m\_btGamePlaza; //游戏广场

CSkinButton m\_btViewItem[MAX\_SERVER]; //房间按钮

//CSkinButton m\_btQuitGame; //退出游戏

CSkinButton m\_btSelfOption; //个人配置

CSkinButton m\_btSwitchUser; //切换帐号

CSkinButton m\_btGlobalOption; //系统配置

//列表按钮

public:

CSkinButton m\_btListControl1; //控制按钮

CSkinButton m\_btListControl2; //控制按钮

//环绕信息

protected:

tagEncircleInfo m\_ImageInfoFrame; //框架信息

tagEncircleInfo m\_ImageInfoServer; //列表信息

//资源变量

protected:

tagEncircleImage m\_EncircleFrame; //框架资源

tagEncircleImage m\_EncircleServer; //列表资源

PlazaViewItem.h 定义了游戏广场视图，CGameFrame框架不仅仅只是游戏广场视图，还有房间视图等其他（服务器列表视图、用户信息视图等），但主要是由两种视图，广场视图和房间视图，因此在CGameFrame的头文件中可以看到，还定义了

IViewItem \* m\_pActiveViewItem; //当前视图

我们来看看这种继承关系

* 游戏广场视图项接口interface IPlazaViewItem : public IViewItem

游戏广场实现 class CPlazaViewItem : public CDialog, public IPlazaViewItem,public

ITCPSocketSink

广场视图对象CPlazaViewItemm\_DlgGamePlaza 是在游戏广场主框架中创建的

* 房间视图项接口

定义在该部分RoomViewItem.h

游戏房间视图接口interface IRoomViewItem : public IViewItem

游戏房间视图实现class CRoomViewItem : public CDialog, public IRoomViewItem, public ITCPSocketSink, public IPurchaseInfo, public ITableFrameSink, public IUserManagerSink, public ISplitterSink, public IChannelMessageSink

广场框架调用树结构的CGameFrame::OnTreeLeftDBClick方法，在该方法中调用CreateRoomViewItem(pListServer)方法创建房间。

游戏房间视图对象 m\_pRoomViewItemCreate是在CGameFrame::CreateRoomViewItem调用过程中通过 pRoomViewItem=new CRoomViewItem; 创建的。

创建了房间对象后，调用pRoomViewItem->InitRoomViewItem(pListServer) 对房间进行初始化，初始化完成后调用 bool bSuccess=pRoomViewItem->ConnectGameServer();连接游戏服务器。

CRoomViewItem \* m\_pRoomViewItem[MAX\_SERVER]; //房间接口m\_pRoomViewItem 是多个房间视图指针的保存地址，在主框架中，可以打开多个房间，多个房间视图可以切换。

* 由此可以看出绿色的IViewItem是视图的公共接口

//激活广场 void ActivePlazaViewItem();

//激活房间 void ActiveRoomViewItem(CRoomViewItem \* pRoomViewItem);

通过调用上面两个函数，在主框架中设置当前视图是广场还是房间。

### 广场视图

先来分析一下广场视图对象的结构：

//游戏广场视图类

class CPlazaViewItem : public CDialog, public IPlazaViewItem,public ITCPSocketSink

其内部结构定义如下：

//状态变量

public:

bool m\_bInitDialog; //初始标志

bool m\_bLogonPlaza; //登录标志

//控件变量

public:

CDlgLogon m\_DlgLogon; //登录对话框

CTCPSocketHelper m\_ClientSocket; //网络连接

//头像变量

protected:

bool m\_bStartDownloadFace ; //下载标识

bool m\_bDownloadConnect; //下载标识

tagCustomFace m\_CustomFace; //自定义头像

CDownloadInfoArrary m\_DownloadInfoArrary ; //下载队列

//导航按钮

public:

CSkinButton m\_WEB\_BEFORE; //浏览器后退

CSkinButton m\_WEB\_BRUSH; //浏览器刷新

CSkinButton m\_WEB\_HOME; //浏览器主页

CSkinButton m\_WEB\_NEXT; //浏览器前进

CSkinButton m\_WEB\_STOP; //浏览器停止

//控件变量

public:

CHtmlBrower \* m\_pHtmlBrower; //浏览器窗口

CToolTipCtrl m\_ToolTipCtrl; //提示控件

//环绕信息

protected:

tagEncircleInfo m\_ImageInfoWeb; //网页浏览

//资源变量

protected:

tagEncircleImage m\_EncircleWeb; //网页浏览

由此可见，广场部分，主要是跟登陆相关，登陆后的网页显示，头像下载，内容不是太多。

### 房间视图

房间的内容比较多，因此该类的结构也比较复杂，RoomViewItem.h 定义了房间视图类，下面来分析其结构：

//应用变量

protected:

CListServer \*m\_pListServer; //房间信息

tagOptionBuffer m\_OptionBuffer; //配置信息

CMD\_GR\_ColumnInfo m\_ListColumnInfo; //列表信息

//房间配置

protected:

WORD m\_wGameGenre; //游戏类型

DWORD m\_dwVideoAddr; //视频地址

BYTE m\_cbHideUserInfo; //隐藏信息

//辅助变量

protected:

bool m\_bRectifyHor; //调整标志

bool m\_bRectifyVor; //调整标志

bool m\_bInitDialog; //初始标志

bool m\_bHideUserList; //隐藏标志

DWORD m\_dwLimitChatTimes; //限制聊时

WORD m\_wReqTableID; //请求桌子

WORD m\_wReqChairID; //请求位置

WORD m\_wFindTableID; //查找位置

DWORD m\_dwMenuUserID; //菜单用户

//配置变量

protected:

CGameOption \*m\_pGameOption; //游戏设置

CServerOption \*m\_pServerOption; //房间设置

//管理变量

protected:

IUserItem \*m\_pMeUserItem; //自己信息

enServiceStatus m\_ServiceStatus; //服务状态

CClientUserManager m\_ClientUserManager; //用户管理

//进程信息

protected:

HWND m\_hWndChannel; //信道窗口

HANDLE m\_hShareMemory; //共享内存

TCHAR m\_szShareName[64]; //共享名字

tagShareMemory \* m\_pShareMemory; //共享内存

PROCESS\_INFORMATION m\_GameProcessInfo; //进程信息

//聊天变量

protected:

DWORD m\_dwChatUserID; //聊天用户

TCHAR m\_szChatName[NAME\_LEN]; //聊天用户

//聊天控件

public:

CComboBox m\_ChatInput; //聊天输入

CComboBox m\_ChatObject; //聊天对象

//CSkinButton m\_btPhrase; //短语按钮

CSkinButton m\_btColorSet; //颜色设置

CSkinButton m\_btCleanScreen; //清屏按钮

CSkinButton m\_btSendChat; //发送按钮

CSkinButton m\_btExpression; //表情按钮

CSkinButton m\_btBugle; //喇叭按钮

CExpression m\_Expression; //表情窗口

CSkinRichEdit m\_ChatMessage; //聊天组件

//功能控件

public:

CSkinButton m\_btBank; //银行按钮

CSkinButton m\_btAutoSit; //加入按钮

CSkinButton m\_btFindUser; //查找按钮

CSkinButton m\_btQuitRoom; //退出按钮

//用户控件

public:

CWebBrowser m\_BrowerAD; //浏览窗口

CTableFrame m\_TableFrame; //桌子框架

CUserListView m\_UserListView; //用户列表

//管理控件

public:

CManagerServer m\_ManagerServer; //房间管理

CManagerMessage m\_ManagerMessage; //系统消息//OMA //系统消息管理

CManagerUserRight m\_ManagerUserRight; //权限管理

CManagerSendWarning m\_ManagerSendWarning; //警告管理

//环绕信息

protected:

tagEncircleInfo m\_ImageInfoTable; //桌子信息

tagEncircleInfo m\_ImageInfoList; //列表信息

tagEncircleInfo m\_ImageInfoChat; //聊框信息

//资源变量

protected:

tagEncircleImage m\_EncircleTable; //桌子资源

tagEncircleImage m\_EncircleList; //列表资源

tagEncircleImage m\_EncircleChat; //聊框资源

//数据缓冲

protected:

WORD m\_wPacketTableID; //缓冲位置

WORD m\_wPacketChairID; //缓冲位置

CDataQueue m\_PacketDataStorage; //数据缓冲

//其他控件

public:

CToolTipCtrl m\_ToolTipCtrl; //提示控件

CSkinSplitter m\_HorSplitter; //拆分窗口//OMA 水平

CSkinSplitter m\_VorSplitter; //拆分窗口//OMA 垂直

CIPCRecvCopyData m\_IPCRecvCopyData; //信道组件

CTCPSocketHelper m\_ClientSocket; //网络连接

CMessageProxyHelper m\_MessageProxyHelper; //消息代理

CPropertyBar m\_PropertyBar; //道具控件

//扩展变量

public:

CPropertyAttributeArrary m\_PropertyAttributeArrary; //道具信息

CFlowerAttributeArrary m\_FlowerAttributeArrary; //鲜花信息

#### 游戏桌子

游戏房间包含了游戏需要的桌子，虽然房间视图类拥有一个桌子对象CTableFrame m\_TableFrame，但是房间模块与桌子模块之间是没有绑定关系的，因此房间中只有一个桌子的接口（桌子框架接口），那么他们是怎么绑定到一起的呢？再来分析CTableFrame的结构：

在CTableFrame中，//游戏桌子框架class CTableFrame : public CWnd, public ITableFrameView，可以看到这里有一个桌子视图接口ITableFrameView。

再看看桌子框架中主要的对象：

CTableResource m\_TableResource; //桌子资源

CTableView \* m\_pTableViewArray; //桌子数组

ITableFrameSink \* m\_pITableFrameSink; //回调接口

桌子资源、桌子视图、桌子框架回调借口，到这里，关于桌子视图接口应该已经清晰了。

#### 用户列表

用户列表，在房间类结构中，我们可以看到有这样一个变量CUserListView m\_UserListView，用户列表视图，再来看用户列表展示的流程。

如果是用户自己首次进入，从广场左侧的游戏房间列表点击CGameFrame::OnTreeLeftDBClick （）将被调用，该方法将调用CreateRoomViewItem（）创建房间，创建房间方法首先检查是否登陆。

##### 登陆过程：

###### 大厅登录

首先会有一个登录对话框，CDlgLogon，该类的实现中，存储了登录服务器的地址，该地址的设置要么是从输入框中获取，要么是从以前输入过的历史记录中获取，有了这个地址，点击登陆的过程才会去做登陆操作，我们来看点击登陆后的调用过程：

CDlgLogon::OnOK（） 该方法中检查输入有效性，然后获取游戏大厅主框架视图的接口，然后调用CPlazaViewItem::SendConnectMessage()方法发起连接，再看该方法的调用过程，PostMessage(WM\_COMMAND,IDM\_CONNECT\_SERVER,0); 发送了一个连接到服务器的消息 ON\_COMMAND(IDM\_CONNECT\_SERVER,OnCommandConnect)；该消息定义在PlazaViewItem.cpp。

OnCommandConnect方法中，m\_ClientSocket->Connect(pszServerIP,PORT\_LOGON\_SERVER)

发起连接后，由内核引擎调度器进行网络传输处理，iErrorCode=WSAAsyncSelect(m\_hSocket,m\_hWnd,WM\_SOCKET\_NOTIFY,FD\_READ|FD\_CONNECT|FD\_CLOSE); WSAAsyncSelect方法定义了网络连接的响应处理句柄hWnd，现在去看内核引擎是怎么处理WM\_SOCKET\_NOTIFY消息的。CTCPSocket::OnSocketNotify方法将响应WM\_SOCKET\_NOTIFY消息。

CTCPSocket::OnSocketNotifyConnect方法被调用，因为是从大厅视图进入的，因此大厅视图方法CPlazaViewItem::OnEventTCPSocketLink被调用，

m\_DlgLogon.SendLogonPacket(m\_ClientSocket.GetInterface()); 登陆视图组件发送登陆包数据，再来看看内部结构

CDlgLogon::SendLogonPacket(ITCPSocket \* pIClientSocke)中发送如下命令

pIClientSocke->SendData(MDM\_GP\_LOGON,SUB\_GP\_LOGON\_ACCOUNTS

pIClientSocke 指向的是CTCPSocket类指针，命令发送后，登陆服务器CAttemperEngineSink::OnSocketMainLogon方法响应。处理SUB\_GP\_LOGON\_ACCOUNTS命令，然后发生数据库命令 m\_pIDataBaseEngine->PostDataBaseRequest(DBR\_GP\_LOGON\_BY\_ACCOUNTS,

登陆服务器的数据库钩子实现方法CDataBaseSink::OnRequestLogon被调用，该方法中

SPLogonByAccounts 执行存储过程，验证用户数据，然后数据库存储过程的执行情况判断，如果成功构造一个DBR\_GP\_LogonSuccess结构的数据，再调用数据库结构处理钩子m\_pIDataBaseEngineEvent->OnEventDataBaseResult(DBR\_GP\_LOGON\_SUCCESS,dwContextID,&LogonSuccess

登陆服务器调度钩子CAttemperEngineSink::OnEventDataBase处理

case DBR\_GP\_LOGON\_SUCCESS: //登录成功

return OnDBLogonSuccess(dwContextID,pData,wDataSize);

再来看看登陆服务器调度钩子CAttemperEngineSink::OnDBLogonSuccess 方法内部

* 发送登陆结果

m\_pITCPNetworkEngine->SendData(dwContextID,MDM\_GP\_LOGON,SUB\_GP\_LOGON\_SUCCESS

响应：大厅视图方法CPlazaViewItem::OnSocketMainLogon响应SUB\_GP\_LOGON\_SUCCESS消息，保存CMD\_GP\_LogonSuccess结构的数据，提示用户，正在下载列表信息，下面看看列表的发送过程

* 发送列表配置

m\_pITCPNetworkEngine->SendData(dwContextID,MDM\_GP\_SERVER\_LIST,SUB\_GP\_LIST\_CONFIG,

响应：大厅视图方法CPlazaViewItem::OnEventTCPSocketRead

case MDM\_GP\_SERVER\_LIST: //列表消息

return OnSocketMainServerList(Command,pData,wDataSize);

再来看大厅视图方法OnSocketMainServerList的内部结构：

处理case SUB\_GP\_LIST\_CONFIG: //列表配置命令

显示在线人数

* 发送游戏列表

m\_ServerList.SendGameTypeList(dwContextID);

m\_ServerList.SendGameKindList(dwContextID);

m\_ServerList.SendGameStationList(dwContextID);

m\_ServerList.SendGameServerList(dwContextID);

m\_ServerList 是登陆服务器实现方法，还有一个是游戏服务实现方法，注意这里还没有到房间登陆，所有还是登陆服务器处理。

SendGameTypeList 发送游戏类型信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GP\_SERVER\_LIST,SUB\_GP\_LIST\_TYPE

响应：广场视图方法CPlazaViewItem::OnEventTCPSocketRead MDM\_GP\_SERVER\_LIST: //列表消息命令

return OnSocketMainServerList(Command,pData,wDataSize);其中处理SUB\_GP\_LIST\_TYPE命令并调用g\_GlobalUnits.m\_ServerListManager.InsertTypeItem 保存列表信息

SendGameKindList 发送游戏种类

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GP\_SERVER\_LIST,SUB\_GP\_LIST\_KIND

响应：广场视图方法CPlazaViewItem::OnEventTCPSocketRead MDM\_GP\_SERVER\_LIST

处理命令 SUB\_GP\_LIST\_KIND

g\_GlobalUnits.m\_ServerListManager.InsertKindItem保存到客户端全局列表

SendGameStationList 发送站点信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GP\_SERVER\_LIST,SUB\_GP\_LIST\_STATION

响应：广场视图方法CPlazaViewItem::OnEventTCPSocketRead MDM\_GP\_SERVER\_LIST

处理命令 SUB\_GP\_LIST\_STATION

g\_GlobalUnits.m\_ServerListManager.InsertStationItem保存到客户端全局列表

SendGameServerList 发送房间列表信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GP\_SERVER\_LIST,SUB\_GP\_LIST\_SERVER

响应：广场视图方法CPlazaViewItem::OnEventTCPSocketRead MDM\_GP\_SERVER\_LIST

处理命令 SUB\_GP\_LIST\_STATION

g\_GlobalUnits.m\_ServerListManager.InsertServerItem保存到客户端全局列表

* 发送登录完成

m\_pITCPNetworkEngine->SendData(dwContextID,MDM\_GP\_LOGON,SUB\_GP\_LOGON\_FINISH);

响应：大厅视图方法CPlazaViewItem::OnSocketMainLogon

关闭提升正在下载列表信息，并展开列表，调用m\_DlgLogon.OnLogonSuccess();记录登陆信息

到处大厅登陆完成了，可以看到左侧的游戏房间列表等数据，但还没有进入房间，接下来分析房间的登陆过程。

###### 房间登陆

游戏大厅主框架下，CGameFrame::OnTreeLeftDBClick响应房间双击事件，分析一下双击事件方法的内部结构

首先检查是否安装了游戏客户端，如果没有则让用户选择下载，如果已经安装了，看下面分析

CreateRoomViewItem(pListServer);在大厅中，创建一个房间视图，看内部是怎么实现的：

CGameFrame::CreateRoomViewItem 可以看到，这个还是一个主框架中的方法，首先检查用户是否登陆，如果没有登陆大厅广场，先让用户登陆大厅广场，然后查找房间，看房间是否已经创建过了，如果创建过，设置已经创建的房间视图到主框架的当前视图，然后判断当前打开的房间数目，如果超过了预定的数目，提升用户关闭一些房间再开始。然后检查大厅版本，下面看看房间视图的创建

pRoomViewItem=new CRoomViewItem;

pRoomViewItem->InitRoomViewItem(pListServer)

创建完成后，调用CRoomViewItem::ConnectGameServer() 连接游戏服务器，连接过程中用到“网络组件”的CTCPSocket::Connect 方法，CTCPSocket::OnSocketNotifyConnect

中，调用m\_pITCPSocketSink->OnEventTCPSocketLink(m\_wSocketID,iErrorCode);

由于当前是从房间视图中产生的连接，因此OnEventTCPSocketLink调用的是房间视图的方法CRoomViewItem::OnEventTCPSocketLink，该方法仅仅简单调用//发送登录信息

SendLogonPacket();

看看其中的结构，CRoomViewItem::SendLogonPacket，发送登陆命令

m\_ClientSocket->SendData(MDM\_GR\_LOGON,SUB\_GR\_LOGON\_USERID,

响应：

由于连接的是游戏服务器，因此是游戏服务调度钩子的方法来响应SUB\_GR\_LOGON\_USERID命令

CAttemperEngineSink::OnSocketMainLogon

下面看该方法的内部结构：

判断当前登陆用户是在线用户还是断线用户，服务器端维持了一个在线用户和断线用户列表，如果不是当前登录用户在另外一个地方登录，也不是断线用户，那么需要到数据库中进行对比验证用户的有效性

m\_pIDataBaseEngine->PostDataBaseRequest(DBR\_GR\_LOGON\_BY\_USERID,

游戏服务器数据库引擎钩子的CDataBaseSink::OnDataBaseEngineRequest 方法会响应数据库用户登录消息，CDataBaseSink::OnRequestLogon 会被调用，SPLogonByUserID方法执行数据库存储过程的执行，之后会根据数据库读取的数据构造一个DBR\_GR\_LogonSuccess结构，投递DBR\_GR\_LOGON\_SUCCESS消息

1. m\_pIDataBaseEngineEvent->OnEventDataBaseResult(DBR\_GR\_LOGON\_SUCCESS,dwContextID,&LogonSuccess,

响应：游戏服务器调度引擎钩子实现文件中，CAttemperEngineSink::OnEventDataBase来响应数据库事件，根据消息类型DBR\_GR\_LOGON\_SUCCESS，游戏服务器调度引擎钩子调用CAttemperEngineSink::OnDBLogonSuccess 方法

该方法首先检查是否登陆用户，是否可登陆房间、判断房间是否已经满、构造用户信息tagServerUserData ServerUserData;

1. //激活用户pConnectItemInfo->pIServerUserItem=m\_ServerUserManager.ActiveUserItem(
2. //发送房间信息SendGameServerInfo(pConnectItemInfo->pIServerUserItem,dwContextID);

CAttemperEngineSink::SendGameServerInfo（）内部发生的消息

* 发送登陆成功消息m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_LOGON,SUB\_GR\_LOGON\_SUCCESS

响应：CRoomViewItem::OnSocketMainLogon 方法中仅仅设置登录房间成功状态

* 发送CMD\_GR\_ServerInfo房间信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_INFO,SUB\_GR\_SERVER\_INFO

响应：CRoomViewItem::OnSocketMainInfo 方法读取CMD\_GR\_ServerInfo \* pServerInfo结构的信息并记录。

创建游戏桌子并初始化

m\_TableFrame.CreateTableFrame(this,100);

m\_TableFrame.InitTableFrame

* 发送列表解释

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_INFO,SUB\_GR\_COLUMN\_INFO

响应：CRoomViewItem::OnSocketMainInfo 读取CMD\_GR\_ColumnInfo结构类型的数据并设置用户列表数据m\_UserListView.SetColumnDescribe

* 发送配置完成消息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_INFO,SUB\_GR\_CONFIG\_FINISH)

响应：CRoomViewItem::OnSocketMainInfo 处理SUB\_GR\_CONFIG\_FINISH 命令，设置游戏进程路径和游戏等级，显示房间视图

* //发送自己信息、在线用户、断线用户

SendUserItem(pIServerUserItem,dwSocketID);

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_USER,SUB\_GR\_USER\_COME,发送SUB\_GR\_USER\_COME命令

响应：房间视图类中

CRoomViewItem::OnSocketMainUser 根据类型SUB\_GR\_USER\_COME 调用CRoomViewItem::OnSocketSubUserCome方法

根据传递过来的数据，读取tagUserInfoHead信息构造tagUserData用户信息

读取扩展信息（游戏用户名、签名、用户社团）

判断当前的用户列表中是否已经存在，如果没有那么调用下面语句

m\_ClientUserManager.ActiveUserItem(UserData) 在用户列表中增加一个用户，否则调用

OnUserItemUpdate(pIUserItem) 更新用户信息。

设置提示信息，XX进来了、更新人数等

* 发送房间所有桌子信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_STATUS,SUB\_GR\_TABLE\_INFO

响应：CRoomViewItem::OnSocketMainStatus 处理SUB\_GR\_TABLE\_INFO命令

读取CMD\_GR\_TableInfo结构数据，设置数据状态

* 发送道具鲜花信息

SendPresentAttribute(pIServerUserItem);

响应：CRoomViewItem::OnSocketMainGameFrame SUB\_GF\_PROP\_ATTRIBUTE

* 发送登陆完成信息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_LOGON,SUB\_GR\_LOGON\_FINISH)

响应：CRoomViewItem::OnSocketMainLogon 响应SUB\_GR\_LOGON\_FINISH命令

发送游戏规则m\_ClientSocket->SendData(MDM\_GR\_USER,SUB\_GR\_USER\_RULE

游戏服务器响应：读取规则数据，构造tagUserRule规则数据，并设置记录到服务器，主要限制逃跑、最高分数、最低分数等由用户设置的信息。

启动游戏客户端进程CRoomViewItem::StartGameClient()

1. //广播用户信息SendUserItem(pConnectItemInfo->pIServerUserItem,INVALID\_WORD);

m\_pITCPNetworkEngine->SendDataBatch(MDM\_GR\_USER,SUB\_GR\_USER\_COME

响应：房间视图类

bool CRoomViewItem::OnSocketMainUser

case SUB\_GR\_USER\_COME: //用户进入

return OnSocketSubUserCome(Command,pData,wDataSize);

tagUserData 和tagServerUserData 这两个结构是一样的，之所以这样定义，是因为我们需要区分客户端和服务器端。

1. //发送消息SendRoomMessage(dwContextID,TEXT("亲爱的用户，欢迎你多提宝贵建议！"),SMT\_INFO);

判断房间类型（积分、比赛、金币等），发送房间消息

m\_pITCPNetworkEngine->SendData(dwSocketID,MDM\_GR\_SYSTEM,SUB\_GR\_MESSAGE

响应：房间视图CRoomViewItem::OnSocketMainSystem响应消息

case SUB\_GR\_MESSAGE: //系统消息

1. //投递道具调度通知，构造了一个道具结构DBR\_GR\_UserProperty

* 加载道具

SPLoadUserProperty执行道具数据库存储过程，投递道具调度通知 m\_pIDataBaseEngineEvent->OnEventDataBaseResult(DBR\_GR\_USER\_PROPERTY,dwContextID, &UserProperty

响应：游戏服务调度引擎CAttemperEngineSink::OnEventDataBase

CAttemperEngineSink::OnDBUserProperty

构造DBR\_GR\_UserProperty道具数据结构，挂接用户数据与道具数据

游戏服务CAttemperEngineSink::OnEventDataBase的

case DBR\_GR\_USER\_PROPERTY: //玩家道具命令

return OnDBUserProperty(dwContextID,pData,wDataSize);//执行方法

* //投递道具加载完成调度通知 m\_pIDataBaseEngineEvent->OnEventDataBaseResult(DBR\_GR\_LOAD\_PROP\_FINISHI

游戏服务CAttemperEngineSink::OnEventDataBase的

case DBR\_GR\_LOAD\_PROP\_FINISHI: //完成加载命令

return OnDBLadPropFinish(dwContextID,pData,wDataSize);

游戏服务CAttemperEngineSink::OnDBLadPropFinish内容

//发送道具

SendResidualProperty(pConnectItemInfo->pIServerUserItem);其内部结构

SendData(pIServerUserItem, MDM\_GF\_PRESENT, SUB\_GF\_RESIDUAL\_PROPERTY, &ResidualProperty, sizeof(ResidualProperty));

//发送道具的响应：

//房间视图类bool CRoomViewItem::OnSocketMainGameFrame (CMD\_Command Command根据命令类型进行处理。

读取CMD\_GF\_ResidualProperty结构数据，获取玩家，设置玩家道具。

//登录完成

##### 入座过程

CRoomViewItem::OnLButtonHitTable 该方法是由用户选择桌子座位触法的，CRoomViewItem::SendSitDownPacket被调用，下面分析其中的结构

构造一个CMD\_GR\_UserSitReq用户请求结构，发送SUB\_GR\_USER\_SIT\_REQ 命令m\_ClientSocket->SendData(MDM\_GR\_USER,SUB\_GR\_USER\_SIT\_REQ

响应：游戏服务CAttemperEngineSink::OnSocketMainUser 被调用，其内容如下

SUB\_GR\_USER\_SIT\_REQ 和SUB\_GR\_USER\_SIT\_REQ执行相同的方法。

解析CMD\_GR\_UserSitReq 结构的数据，然后冲服务器用户列表读取tagServerUserData用户数据，执行pTableFrame->PerformSitDownAction 动作，结构如下

游戏服务下CTableFrame::PerformSitDownAction，从游戏服务列表获取用户数据，读取分数、规则、用户数据

游戏规则检验、游戏状态判断已经开始不能加入、比赛类型校验、地址校验（不允许相同IP玩家）、对比赛局数的判断、权限的判断等

发送桌子以及用户状态

m\_pIGameServiceFrame->SendTableStatus(m\_wTableID);

m\_pIGameServiceFrame->SendUserStatus(pIServerUserItem);

桌子状态：

构造CMD\_GR\_TableStatus类型的桌子状态变量，发送m\_pITCPNetworkEngine->SendDataBatch(MDM\_GR\_STATUS,SUB\_GR\_TABLE\_STATUS

响应：CRoomViewItem::OnSocketMainStatus

解析CMD\_GR\_TableStatus结构的状态数据，OnUserItemUpdate更新所有用户状态

通知游戏进程

SendProcessData(IPC\_MAIN\_USER,wSubCmdID,NULL,0);子命令IPC\_SUB\_GAME\_START 或者 IPC\_SUB\_GAME\_FINISH

响应：游戏框架CClientKernel::OnIPCUser

游戏框架项目，由于有不同的游戏，但是游戏界面大致结构是一样的，因此对整个游戏提供一个游戏框架，不同的游戏只需要设置不同的游戏视图即可，详细看游戏框架章节

CClientKernelSink::OnEventUserStatus

## 游戏框架

游戏框架项目主要是为游戏客户端提供一个框架，该项目本身仅仅提供虚方法，具体的游戏客户端应用程序实现框架的虚方法。

APP框架本身只代表一个应用程序，APP框架应该持有对话框等视图作为显示之用，CGameFrameDlg 就代表了APP框架的对话框，游戏客户端需要继承实现框架的对话框类

看看游戏客户端中，类的继承关系

class CGameClientApp : public CGameFrameApp

class CGameClientDlg : public CGameFrameDlg

游戏客户端必须实现游戏框架中的两个类CGameFrameApp和CGameFrameDlg

具体游戏中的实现，这里暂时不分析，但是需要分析APP框架的两个重要类

### CGameFrameApp

CGameFrameApp : public CWinApp 定义在GameFrameApp.h

CGameFrameDlg \* m\_pGameFrame; //框架窗口

virtual CGameFrameDlg \* GetGameFrameDlg()=NULL; //创建主窗口

拥有一个框架对话框指针，和一个获取对话框指针的方法，该类时框架类，因此具体的执行方法是继承了该类的接口实现。

来看看运行过程：

APP对象创建，首先会运行CGameFrameApp::InitInstance()方法，在该方法中初始化

加载配置 g\_GlobalOption.LoadOptionParameter();

创建主窗口 m\_pGameFrame=GetGameFrameDlg();

设置主窗口 m\_pMainWnd=m\_pGameFrame; 设置主窗口时把游戏客户端的对话框窗口设置为框架APP窗口指针。由此可见，整个控制权都转移到了CGameFrameDlg上，下面分析CGameFrameDlg类

### CGameFrameDlg

CGameFrameDlg主要实现对话框消息控制，界面控制、通信等功能，下面看具体的实现和主要结构变量。

class CGameFrameDlg : public CSkinDialog, public ISplitterSink

//控件变量

protected:

CSkinButton m\_btControl; //控制按钮

CSkinSplitter m\_VorSplitter; //拆分控件

CGameFrameView \* m\_pGameFrameView; //视图指针

CGameFrameControl \* m\_pGameFrameControl; //控制视图

//内核变量

private:

CClientKernelSink \* m\_pKernelSink; //钩子接口

CUserFaceResHelper m\_UserFaceResHelper; //头像组件

CClientKernelHelper m\_ClientKernelHelper; //游戏逻辑

CMessageProxyHelper m\_MessageProxyHelper; //聊天代理

这里有两类主要的变量，一个是控件变量，一个是内核变量下面具体来分析一下这两类变量

#### 控件变量

控件变量里面有主要是需要分析CGameFrameView 和CGameFrameControl两类

##### 视图指针CGameFrameView

CGameFrameView是一个游戏视图框架，这里依然是框架，不同的游戏客户端有不同的UI实现，下面分析该类的构成。

主要变量包括：位置变量、用户变量、组件指针、画图变量等，其主要作用是控制绘制游戏界面，注意这里仅仅只是一个框架，需要在游戏客户端中去实现该类。

##### 控制视图CGameFrameControl

框架控制视图，主要控制玩家的操作过程，需要在游戏客户端去实现该类，下面看具体的变量定义

tagUserData \* m\_pUserData; //用户指针

//组件变量

protected:

CGameFrameDlg \* m\_pGameFrameDlg; //对话框

IClientKernel \* m\_pIClientKernel; //内核接口

//环绕信息

protected:

tagEncircleInfo m\_ImageInfoList; //列表信息

tagEncircleInfo m\_ImageInfoChat; //聊框信息

//资源变量

protected:

tagEncircleImage m\_EncircleList; //列表资源

tagEncircleImage m\_EncircleChat; //聊框资源

//聊天控件

public:

CComboBox m\_ChatInput; //聊天输入

CComboBox m\_ChatObject; //聊天对象

CWebBrowser m\_BrowerAD; //浏览窗口

//CSkinButton m\_btPhrase; //短语按钮

CSkinButton m\_btColorSet; //颜色设置

CSkinButton m\_btCleanScreen; //清屏按钮

CSkinButton m\_btBugle; //啦八按钮

CSkinButton m\_btSendChat; //发送按钮

CSkinButton m\_btExpression; //表情按钮

CExpression m\_Expression; //表情窗口

CSkinRichEdit m\_ChatMessage; //聊天组件

//其他变量

public:

CSkinButton m\_btSound; //游戏声音

CSkinButton m\_btLookOn; //旁观设置

CSkinButton m\_btGameRule; //游戏规则

CSkinButton m\_btGameOption; //游戏设置

CSkinButton m\_btGameQuit; //游戏设置

//控件变量

public:

CToolTipCtrl m\_ToolTipCtrl; //提示控件

CUserListView m\_UserListView; //用户列表

CUserInfoView m\_UserInfoView; //玩家信息

在所有的内部变量中，最重要的是

CGameFrameDlg \* m\_pGameFrameDlg; //对话框

IClientKernel \* m\_pIClientKernel; //内核接口

对话框对象引用，控制的动作需要传递到对话框上去， IClientKernel内核是客户端内核引用，具体看下面章节分析。

#### 内核变量

主要由以下四个构成，请看具体分析

CClientKernelSink \* m\_pKernelSink; //钩子接口

CUserFaceResHelper m\_UserFaceResHelper; //头像组件

CClientKernelHelper m\_ClientKernelHelper; //游戏逻辑

CMessageProxyHelper m\_MessageProxyHelper; //聊天代理

##### CClientKernelSink内核钩子

客户端内核钩子类实现了内核钩子接口，持有一个对话框CGameFrameDlg对象引用，处理时间消息、网络消息、进程消息，处理游戏事件游戏状态、旁观更新、聊天信息、处理用户事件用户进入、用户离开、用户积分、用户状态、会员等级，礼物事件初始道具、初始鲜花、鲜花消息。

class CClientKernelSink : public IClientKernelSink

//变量定义

protected:

CGameFrameDlg & m\_GameFrameDlg; //游戏框架

##### 头像组件CUserFaceResHelper

主要是用户头像资源的使用接口

##### 客户端内核CClientKernelHelper

主要的实现在CClientKernel类中

//客户端内核

class CClientKernel : public CWnd, public IClientKernel, public IChannelMessageSink

//房间信息

protected:

WORD m\_wTableID; //桌子号码

WORD m\_wChairID; //椅子号码

//固有信息

protected:

DWORD m\_dwUserID; //用户I D

tagServerAttribute m\_ServerAttribute; //房间属性

//游戏信息

protected:

bool m\_bInquire; //关闭询问

bool m\_bLookonMode; //旁观模式

BYTE m\_bGameStatus; //游戏状态

//配置信息

protected:

bool m\_bAllowLookon; //允许旁观

bool m\_bAllowUserLookon; //允许旁观

//定时器信息

private:

UINT m\_nTimerID; //定时器ID

UINT m\_nElapseCount; //定时器计数

WORD m\_wTimerChairID; //定期器位置

HWND m\_hWndGameFrame; //框架句柄

//用户信息

protected:

tagUserData \* m\_pMeUserItem; //自己信息

tagUserData \* m\_pUserItem[MAX\_CHAIR]; //玩家信息

CUserItemPtrArray m\_UserItemLookon; //旁观数组

CUserItemFactory m\_UserItemFactory; //用户工厂

//通讯变量

protected:

HANDLE m\_hShareMemory; //共享句柄

tagShareMemory \* m\_pShareMemory; //共享内存

CChannelServiceHelper m\_ChannelServiceHelper; //信道服务

//接口指针

protected:

IMessageProxy \* m\_pIMessageProxy; //消息代理

IClientKernelSink \* m\_pIClientKernelSink; //钩子接口

##### 聊天代理CMessageProxyHelper

信息代理组件的实现

class CMessageProxy : public IMessageProxy

//变量定义

protected:

CWnd \*m\_pParentWnd; //父窗口

CSkinRichEdit \* m\_pSkinRichEdit; //消息组件

tagMessageOption m\_MessageOption; //配置信息

TCHAR m\_szPath[MAX\_PATH]; //当前路径

信道服务模块见8.3节

## 信道模块

主要用于大厅进程和游戏进程的进程间通信

//信道模块类

class CChannelService : public CWnd, public IChannelService, public IChannelMessageSink

//变量定义

protected:

HWND m\_hWndServer; //接收窗口

CDataQueue m\_DataStorage; //数据对象

CIPCRecvCopyData m\_IPCRecvCopyData; //IPC 接收

//接口指针

protected:

IChannelEventSink \* m\_pIChannelEventSink; //回调接口

IChannelMessageSink \* m\_pIChannelMessageSink; //回调接口

信道事件钩子接口，主要处理连接消息、应答消息、断开消息

信道消息钩子接口，主要处理信道数据内容

# 游戏流程

主要以麻将为例，描述游戏运行过程中的流程以及数据传输的方向

如果游戏客户端已经打开的情况下，用户执行“开始”按钮命令，那么将发生消息到服务器，服务器也会将该消息进行记录，如果有其他用户已经在桌子上，需要发送用户开始的消息，具体的过程，接下来进行详细分析。

## 用户ready

游戏客户端进程

大厅客户端进程

游戏服务器端

IPC

网络通信

用户ready的过程，首先是游戏客户端发起请求，然后经过游戏大厅进程接受消息，然后通过大厅进程发送到服务器端，服务器的响应也是反馈到大厅进程，再由大厅进程发送到游戏客户端进程。

### 客户端进程

#### 游戏客户端进程

GameClientView.h 定义了消息响应ON\_BN\_CLICKED(IDC\_START, OnStart)，CGameClientView::OnStart() 方法将发送消息SendMessage(IDM\_START,0,0);，IDM\_START消息会被GameClientDlg.cpp 中定义的消息ON\_MESSAGE(IDM\_START,OnStart)响应，游戏对话框类的CGameClientDlg::OnStart(WPARAM wParam, LPARAM lParam)方法将会执行游戏开始的具体事务。

CGameClientDlg::OnStart房间进行游戏画面设置和定时器设置，然后发送消息

CGameFrameDlg::SendUserReady(void \* pBuffer, WORD wDataSize)房间通过客户端内核助手对象调用发送消息，return m\_ClientKernelHelper->SendUserReady(pBuffer,wDataSize);

CClientKernel::SendUserReady(void \* pBuffer, WORD wDataSize)方法将发送消息

return SendSocketData(MDM\_GF\_FRAME,SUB\_GF\_USER\_READY,pBuffer,wDataSize);

CClientKernel::SendSocketData(）中通过调用信道服务辅助对象发生数据 m\_ChannelServiceHelper->SendData(IPC\_MAIN\_SOCKET,IPC\_SUB\_SOCKET\_SEND

到这里，发送逻辑将离开游戏客户端进程，游戏大厅进程将接受客户端进程发送过来的消息，中间的进程间通信这里暂时不再讲述，下面看看游戏大厅房间视图对象怎么处理消息的。

#### 客户端大厅进程

CRoomViewItem::OnIPCSocket 处理IPC\_SUB\_SOCKET\_SEND 命令，通过调用如下 m\_ClientSocket->SendData(pCommand->wMainCmdID,pCommand->wSubCmdID,pSocketPackage->cbBuffer,wSendSize); SUB\_GF\_USER\_READY被构造数据包发送消息到游戏房间服务器。中间的网络通信模块暂时不进行分析，下面看房间服务器对SUB\_GF\_USER\_READY的响应。

游戏服务的CTableFrame::OnEventSocketFrame方法将响应，首先取得tagServerUserData结构类型的信息，然后判断用户是否旁观用户做出处理，然后设置用户状态，设置用户同意后需要执行的动作，在麻将中貌似没有需要做的动作，因此仅仅是返回一个值。

m\_pITableUserAction->OnActionUserReady(pUserData->wChairID,m\_pIUserItem[pUserData->wChairID],(VOID \*)pDataBuffer,wDataSize);

发送状态 m\_pIGameServiceFrame->SendUserStatus(pIServerUserItem); 游戏服务的调度引擎钩子CAttemperEngineSink::SendUserStatus(IServerUserItem \* pIServerUserItem)被调用，下面看看是怎么发送用户状态的。

构造CMD\_GR\_UserStatus类型的数据包，发送数据调用如下

m\_pITCPNetworkEngine->SendDataBatch(MDM\_GR\_USER,SUB\_GR\_USER\_STATUS,&UserStatus,sizeof(UserStatus)); SendDataBatch的具体实现暂时不进行分析，批量信息发送了，再来看看谁来响应。

接受数据的依然是游戏大厅进程，游戏房间视图RoomViewItem::OnSocketMainUser方法，SUB\_GR\_USER\_STATUS:调用 OnSocketSubStatus(Command,pData,wDataSize);处理接受到的数据，下面看看是怎么处理的。

CRoomViewItem::OnSocketSubStatus 解析CMD\_GR\_UserStatus结构的数据，得到tagUserData类型的用户数据和自己的用户数据，更像大厅进程维护的用户数据信息，设置桌子状态调用进程间通信方法通知游戏客户端进程SendProcessData(IPC\_MAIN\_USER,IPC\_SUB\_USER\_STATUS,&UserStatus,sizeof(UserStatus)

到此，消息处理切换到游戏客户端进程

CClientKernel::OnIPCUser 处理IPC\_SUB\_USER\_STATUS 信息，CClientKernel::UpdateUserItem(DWORD dwUserID, 调用CClientKernelSink::OnEventUserStatus(tagUserData \* pUserData,更像用户信息，设置游戏客户端窗口视图m\_GameFrameDlg.m\_pGameFrameControl->UpdateUserItem(pUserData);

## 游戏开始

当服务收到最后一个用户ready 后，就会调用CTableFrame::StartGame()方法开始游戏，但是游戏客户端进程到目前为止还没有受到开始游戏的影响，下面看看具体过程。

标记游戏开始、记录玩家状态、发送桌子状态信息、发送用户状态信息，调用游戏开始事件

### 第一回合

#### 发送桌子状态

游戏服务器中CTableFrame::StartGame()下面m\_pIGameServiceFrame->SendTableStatus(m\_wTableID);

CAttemperEngineSink::SendTableStatus 发送状态信息，构造CMD\_GR\_TableStatus类型m\_pITCPNetworkEngine->SendDataBatch(MDM\_GR\_STATUS,SUB\_GR\_TABLE\_STATUS,&TableStatus,sizeof(TableStatus));命令发送后，由大厅进程响应消息。

大厅进程中CRoomViewItem::OnSocketMainStatus，解析CMD\_GR\_TableStatus 结构信息，设置状态信息，设置用户状态发送命令道游戏客户端进程

IPC\_SUB\_GAME\_START:IPC\_SUB\_GAME\_FINISH; SendProcessData(IPC\_MAIN\_USER,wSubCmdID,NULL,0);

CClientKernelSink::OnEventUserStatus CClientKernelSink::OnEventUserStatus 被调用，该方法更像游戏客户端进程中的对话框，视频区域等

//发送用户状态

m\_pIGameServiceFrame->SendUserStatus(m\_pIUserItem[i]);

游戏服务调度引擎钩子实现调用CAttemperEngineSink::SendUserStatus(IServerUserItem \* pIServerUserItem)，发送 m\_pITCPNetworkEngine->SendDataBatch(MDM\_GR\_USER,SUB\_GR\_USER\_STATUS,&UserStatus,sizeof(UserStatus));

响应：大厅进程的CRoomViewItem::OnSocketMainUser 被调用，SUB\_GR\_USER\_STATUS: CRoomViewItem::OnSocketSubStatus 进行处理SendProcessData(IPC\_MAIN\_USER,IPC\_SUB\_USER\_STATUS,&UserStatus,sizeof(UserStatus));发送游戏客户端进程消息，

//通知启动事件 m\_pITableFrameSink->OnEventGameStart();

CTableFrameSink::OnEventGameStart() 是服务器端的处理函数，这就正式的步入了游戏启动状态，看下节的分析

### 第二回合

CTableFrameSink::OnEventGameStart()是游戏正式开始的命令，以麻将为例，开始记录桌子状态，各种复位操作，洗牌，构造游戏开始数据结构 CMD\_S\_GameStart GameStart;

发送玩家数据和旁观数据

发送开始数据

m\_pITableFrame->SendTableData(i,SUB\_S\_GAME\_START,&GameStart,sizeof(GameStart)); m\_pITableFrame->SendLookonData(i,SUB\_S\_GAME\_START,&GameStart,sizeof(GameStart));

响应：

游戏客户端进程CGameClientDlg::OnGameMessage响应消息，此处忽略了进程间通信内容，游戏客户端CGameClientDlg::OnSubGameStart 将被调用，解析CMD\_S\_GameStart结构类型数据，设置界面，设置摇塞子动画等。

## 开始出牌

在游戏开始后，还有一个摇筛子的动画，在动画结束后，CGameClientDlg::OnSiceFinish方法处理庄家出牌。设置显示界面后，设置出牌定时器来限定出牌事件

SetGameTimer(m\_wCurrentUser,IDI\_OPERATE\_CARD,TIME\_OPERATE\_CARD);

CGameClientDlg::OnTimerMessage来响应定时器消息，对IDI\_OPERATE\_CARD类型，选择自动出牌或者托管等操作。

开始出牌

用户选择视图上的牌，CGameClientView::OnLButtonDown方法被执行，投递PostMessage(IDM\_OUT\_CARD,cbHoverCard,cbHoverCard); CGameClientDlg::OnOutCard(WPARAM wParam, LPARAM lParam)将响应该消息，该方法进行界面设置，发送出牌数据，SendData(SUB\_C\_OUT\_CARD,&OutCard,sizeof(OutCard));播放声音。游戏服务器端CTableFrameSink::OnGameMessage 响应消息，CTableFrameSink::OnUserOutCard(WORD wChairID, BYTE cbCardData)处理用户出牌操作，后续的操作将根据游戏的规则来完成，此处不再继续介绍。

## 游戏结束

### 游戏结束调用处理

当游戏根据规则需要结束的时候会调用

CTableFrameSink::OnEventGameEnd将被调用

//发送结束信息 m\_pITableFrame->SendTableData( INVALID\_CHAIR,SUB\_S\_GAME\_END,&GameEnd,sizeof(GameEnd) );

消息的响应此处不再描述

CTableFrame::ConcludeGame()，这里面会对一些游戏规则进行检验，对不符合条件的用户进行限制。

### 游戏分数结算

#### 分数计算调用过程

游戏结束后，分数写入WriteUserScore(WORD wChairID, LONG lScore, LONG lRevenue, enScoreKind ScoreKind, LONG lPlayTimeCount)被调用，CTableFrame::WriteUserScore(IServerUserItem \* pIServerUserItem, LONG lScore, LONG lRevenue, enScoreKind ScoreKind, LONG lPlayTimeCount)被调用

IServerUserItem 类型表示服务器上的用户数据项目 pIServerUserItem->WriteScore(ScoreInfo,dwPlayTimeCount);

看看写入的过程是怎么样的，在ServerUserManager.h定义的如下方法被调用

CServerUserItem::WriteScore(tagScoreInfo & ScoreInfo, DWORD dwPlayTimeCount)

可以看到，这里只是在服务器上修改了积分，但是并没有写入到数据库中，那么神秘时刻才会写入到数据库中去呢？

#### 分数数据库写入