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()，下面详细讲述该方法中的执行内容：

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：

//建立窗口 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

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

//游戏广场 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

interface IRoomViewItem : public IViewItem

//游戏房间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，下面来分析主要的变量

//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，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，在主框架中可以看到

CTableFrame m\_TableFrame，这个对象是在主框架建立的时候创建的，那么在CTableFrame中，我们可以看到这个接口和具体的游戏桌子是怎么绑定的。

//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。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：

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（这里需要获取一个可用游戏服务器的列表），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，该消息由游戏大厅视图来响应ON\_COMMAND(IDM\_CONNECT\_SERVER,OnCommandConnect)；

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该消息的。

CTCPSocket::OnSocketNotify方法将被调用

在内核引擎的控制下

CRoomViewItem::OnEventTCPSocketLink被调用来响应登录事件，下面来看该方法的处理过程：

CRoomViewItem::SendLogonPacket发送登录事件到游戏服务器

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

再来看看三大神器的游戏服务器是怎么响应的，首先网络组件会根据请求调用到游戏服务器的CAttemperEngineSink::OnEventTCPNetworkRead方法，根据发送的命令，登陆处理请求将执行OnSocketMainLogon方法，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,

数据库校验请求，该部分会由内核引擎调用到内核引擎的CAttemperEngine::OnQueueServiceSink方法，在该方法中，根据参数类型进行数据库事件处理，由于是登录验证，所以会调用到登录服务器的内核调动引擎钩子实现

CAttemperEngineSink::OnEventDataBase，登录成功，OnDBLogonSuccess 方法会被调用

该方法中将发送如下信息

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

广场将响应该条信息，CPlazaViewItem::OnSocketMainLogon方法被调用

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

广场将响应该条信息，CPlazaViewItem::OnSocketMainServerList 方法被调用，这里貌似仅仅是更新了在线用户数

* //发送游戏列表

m\_ServerList.SendGameTypeList(dwContextID);

CPlazaViewItem::OnSocketMainServerList SUB\_GP\_LIST\_TYPE 分支将响应该信息

m\_ServerList.SendGameKindList(dwContextID);

CPlazaViewItem::OnSocketMainServerList SUB\_GP\_LIST\_KIND 分支将响应该信息

m\_ServerList.SendGameStationList(dwContextID);

CPlazaViewItem::OnSocketMainServerList SUB\_GP\_LIST\_STATION 分支将响应该信息

m\_ServerList.SendGameServerList(dwContextID);

CPlazaViewItem::OnSocketMainServerList SUB\_GP\_LIST\_SERVER 分支将响应该信息

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

CPlazaViewItem:: OnSocketMainLogon SUB\_GP\_LOGON\_FINISH 分支将响应该信息

//登录完成

##### 入座过程

然后查找是否已经创建房间，如果没有创建则进行创建，完成后连接游戏服务器

用户进入房间，CRoomViewItem::OnSocketSubUserCome（）会被调用，这个方法里面会有如下调用m\_ClientUserManager.ActiveUserItem(UserData)；这个m\_ClientUserManager是CRoomViewItem下的关系管理对象会记录所有进出房间的用户信息，因此这个对象持有一个在线用户列表。m\_ClientUserManager.ActiveUserItem会把进入的用户记录到在线用户列表中。

## 游戏端APP进程