Confidential

(秘密)

传奇 3 游戏源码分析

Version 1.0

楠楠

传奇服务器技术

简述:

最近对高性能的服务器比较感兴趣，读过了 DELPHI 的 Socker 源码 WebService 及 RemObject 之后，高性能的服务器感兴趣。

你可能需要的以下知识才能更好的读懂一个商业源码：

1).SOCKET 的 I/O 模型熟悉掌握。

2).面向对象技术的熟悉掌握。

3).Socket 的 API 掌握。

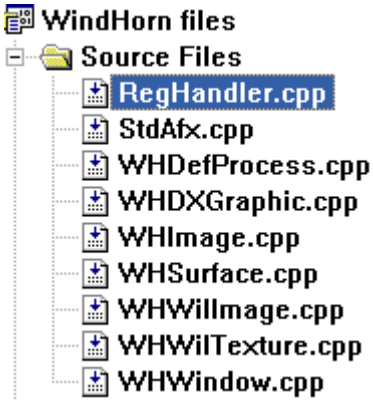
4).多线程技术等。

5).一门熟悉的开发工具掌握,和多种语言的源码阅读能力。

我下的源码 LegendOfMir2_Server：共包含 AdminCmd, DBSrv, GameGate, GameSvr,LoginGate, LoginSvr, SelGate 七个工程文件。传奇的客户端源代码有两个工程, WindHorn 和 Mir2Ex。

Mir2Ex 客户端

DirectX 类库分析(WindHorn)：



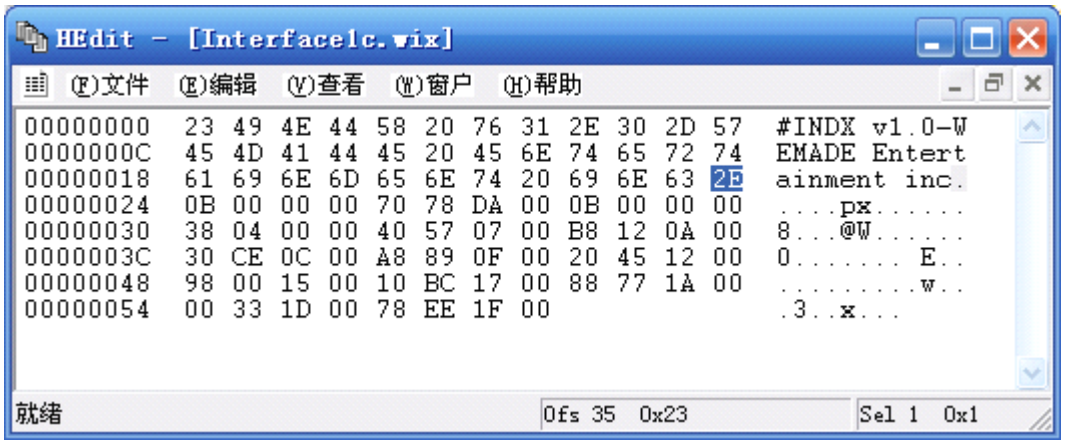
1. RegHandler.cpp 注册表访问(读写)。
2. CWHApp 派生 CWHWindow， CWHWindow 完成窗口的注册和创建。 CWHWindow 派生出 CWHDXGraphicWindow,CWHDXGraphicWindow 调用 CWHWindow 完成创建窗口功能，然后再调用 CreateDXG()来初始化 DirectX。
3. WHDefProcess.cpp 在构造函数中获得 CWHDXGraphicWindow 句柄。
   Clear 函数中调用在后台缓存上进行绘图操作，换页至屏幕。
   ShowStatus 函数,显示状态信息。
   DefMainWndProc 函数，调用 CWHDXGraphicWindow->MainWndProcDXG 消息处理。
4. WHImage.cpp 图象处理。加载位图，位图转换。优化处理。
5. WHSurface.cpp 主页面处理。
6. WHWilTexture.cpp 材质渲染。
   WILTextureContainer： WIL 容器类。m_pNext 指向下一个 WILTextureContainer，单链表。
7. WHWilImage.cpp 从 Data 目录中加载 Wix 文件（内存映射）。
8. WHDXGraphic.cpp 处理 DirectX 效果。

传奇文件类型格式探讨(一)：

Wix 文件：索引文件，根据索引查找到相应数据地址(数据文件)。

```
// WIX 文件头格式
typedef struct tagWIXFILEIMAGEINFO
{
    CHAR    szTmp[40];      // 库文件标题 'WEMADE Entertainment inc.' WIL 文件头
    INT     nIndexCount;    // 图片数量
    INT*    pnPosition;     // 位置
}WIXIMAGEINFO, *LPWIXIMAGEINFO;
```

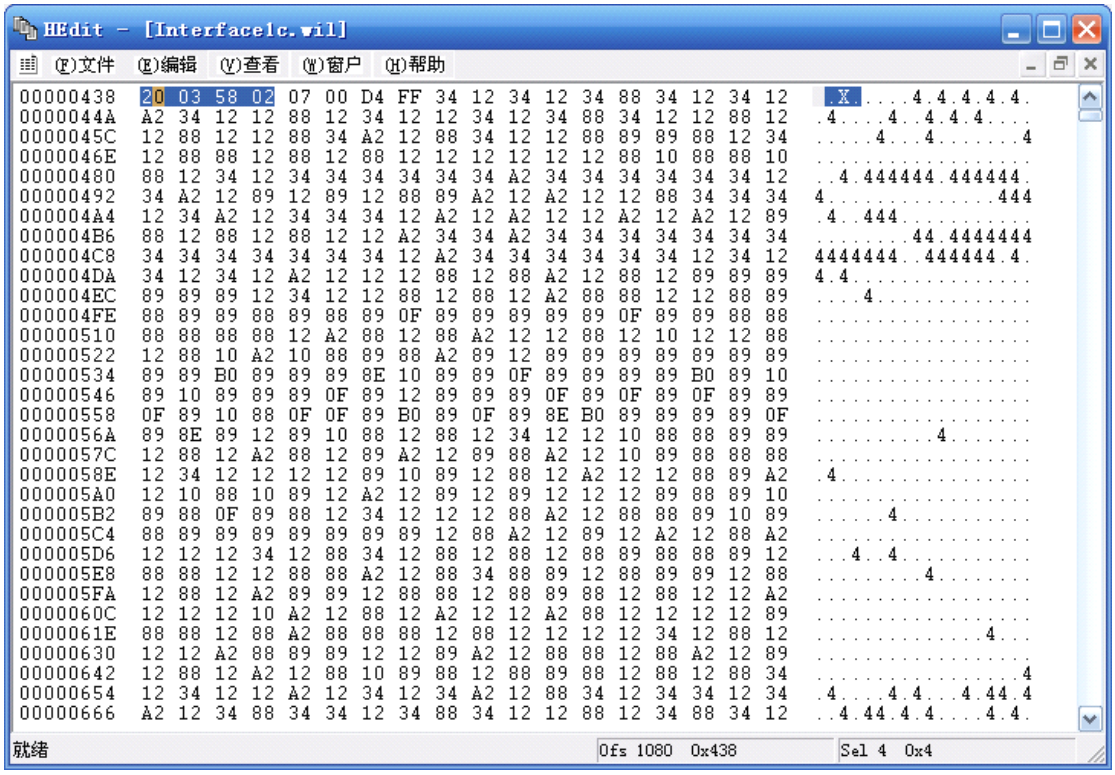
我们下载一个 Hedit 编辑器打开一个 Wil 文件，分析一下。我们发现 Wix 文件中，0x23 地址(含该地址)以前的内容是都相同的，即为：#INDX v1.0-WEMADE Entertainment inc.



Ofs44 0x2C 的地方：存放着 0B 00 00 00，高低位转换后为：0xB 转换十进制数为 11(图片数量)Ofs48 0x30 的地方：存放着 38 04 00 00，高低位转换后为：0x438 = 1080，这个就是图象数据的开始位置。

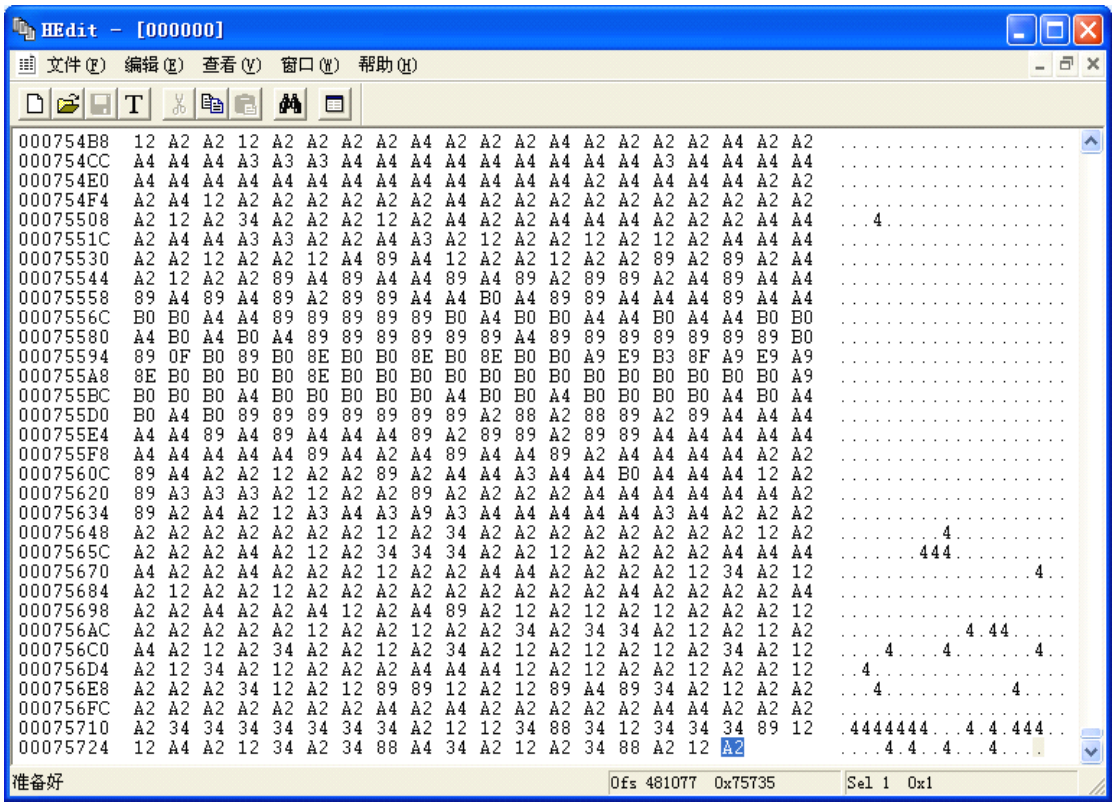
我们用 Wil 编辑打开对应的 Wil 文件，发现，果然有 11 张图片。另外我们发现，在 Ofs = 44 -47 之间的数据总是 38 04 00 00，终于明白，所有的图片起始位置是相同的。

Wil 文件：数据文件。
前面我们说了图象数据的开始位置为 0x438 = 1080，1080 中有文件开头的 44 字节都是相同的。所以，就是说有另外的 1036 字节是另有用途。1036 中有 1024 是一个 256 色的调色板。而 Wil 里面的图片格式都是 256 色的位图储存。



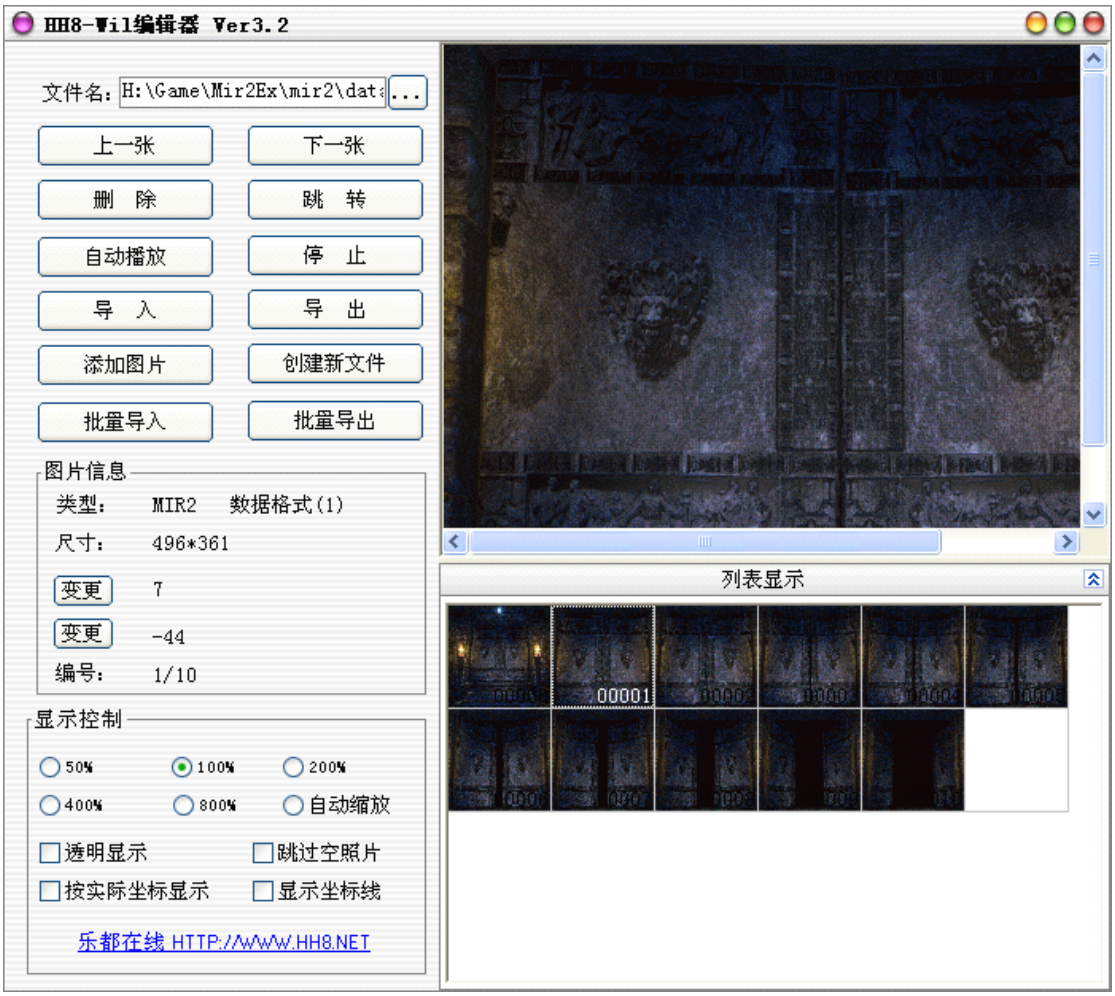
我们看到图片位置数据为： 20 03 58 02，转化为十六进制： 0x320，0x258 刚好就是 800*600 大小的图片。07 00 D4 FF 为固定值(标识)。图片起始位置为：
Ofs 1088：0x440 图片大小为 480000

起始位置：0x440 1088　　终止位置：0x7573F 481087 为了验证数据是否正确，我们通过 Wil 工具，把第一幅图片导出来，然后用 Hedit 编辑器打开，经过对比，我们发现，数据一致。大小一致。

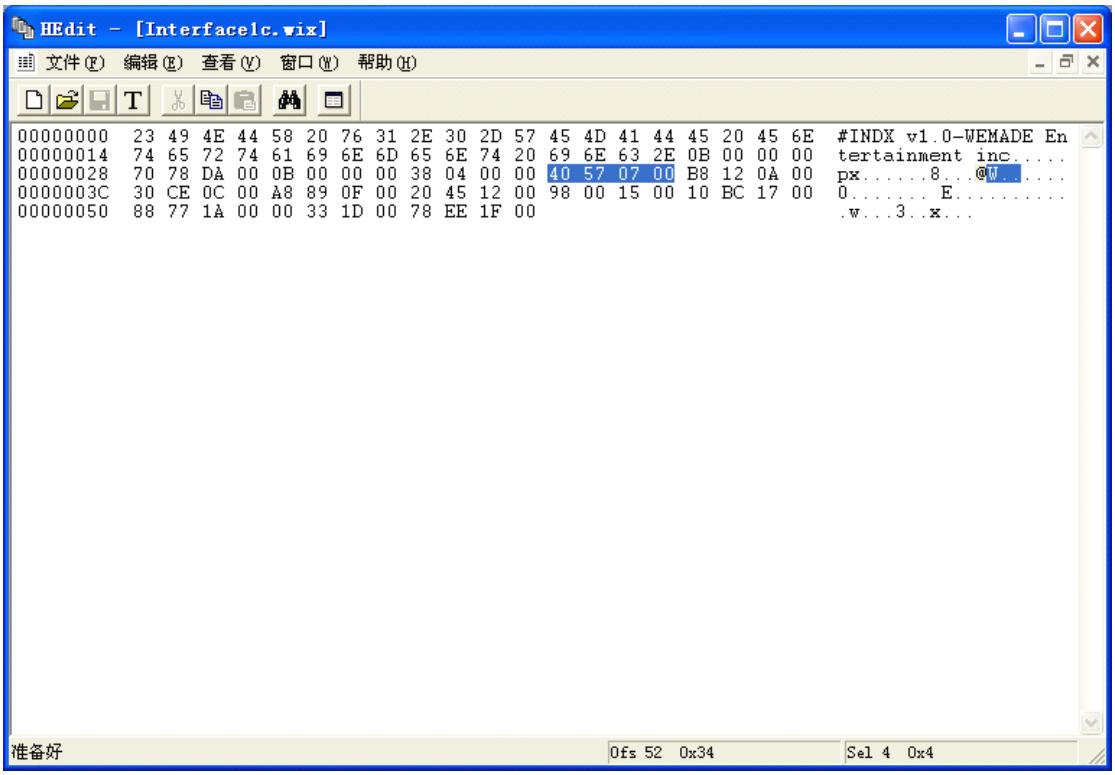


大家看到图片 1 的结束位置为 Ofs 481077,减去 1080+1 = 480000 刚好 800*600 大小。我们用 Wil 抓图工具打开看一下(确定是 800*600 大小)：

我们导出第二张 BMP 图片

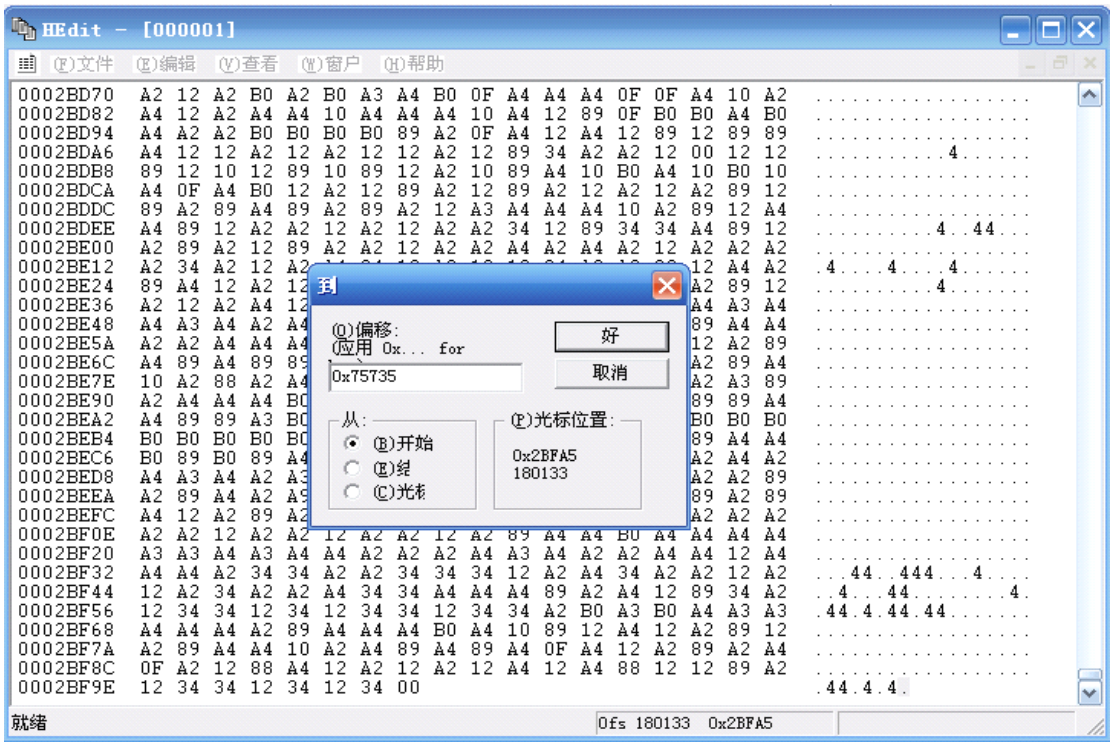


图片的大小为：496* 361，我们从 Wix 中读出第二张图片的索引位置：

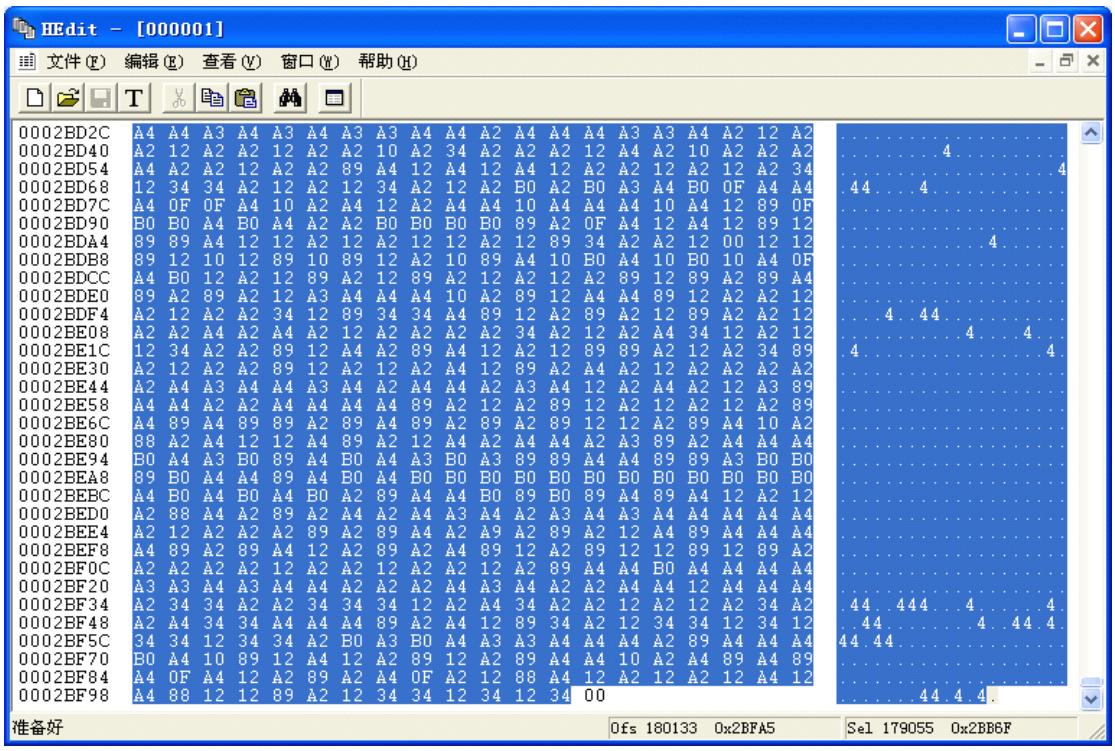


根据贴图，我们发现第二张图片的索引位置为： 40 57 07 00，转换为十六进制：0x75740,即为：481088，前面我们讲到第一张图片的结束位置是： 0fs 481077,从 Wix 中读出来的也刚好为第二张图片的起始位置：

（我们分析 Wil 中的第二张图片，起始位置：0x75740 481088）： F0 01 69 01 为图片长宽： 0x1F0, 0x169 为 496* 361 。 07 00 D4 FF 为固定值(标识)。



我们用工具打开第二张 BMP 图片，从起始位置，一直选取中至结束，发现刚好选 496* 361 字节大小。两边数据对比之后发现一致。知道了图片格式，我们可以写一个抓图片格式的程序了。

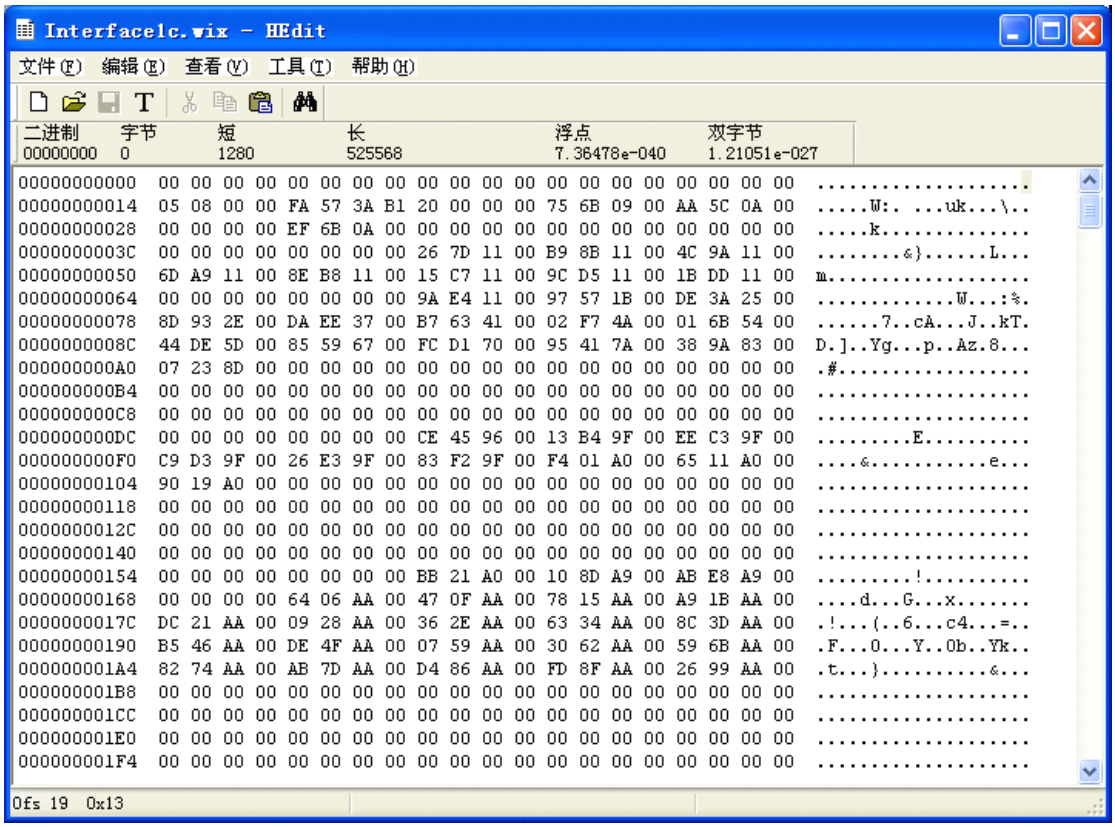


传奇文件类型格式探讨(二)：

```
// WIX 文件头格式 (NEW)
typedef struct tagNEWWIXFILEIMAGEINFO
{
    CHAR     szTitle[20];    // 库文件标题 '00 00 到 0x13 地址为止
    INT      nIndexCount;    // 图片数量
    INT*     pnPosition;     // 位置
}NEWWIXIMAGEINFO, *LPNEWWIXIMAGEINFO;
```

我们下载一个 Hedit 编辑器打开一个 Wil 文件，分析一下。我们发现 Wix 文件中，0x13 地址(含该地址)以前的内容是都相同的，即为： ‘ ’ 20 个空格。

图片数量： nIndexCount



Ofs 20, 0x14 的位置，存放的数据为 05 08 00 00，高低位转换后为：0x805 十制数为 2053 (图片数量)。Ofs28 0x1C 的地方：存放着 20 00 00 00，高低位转换后为：0x20 = 32, 这个就是图象数据的开始位置。

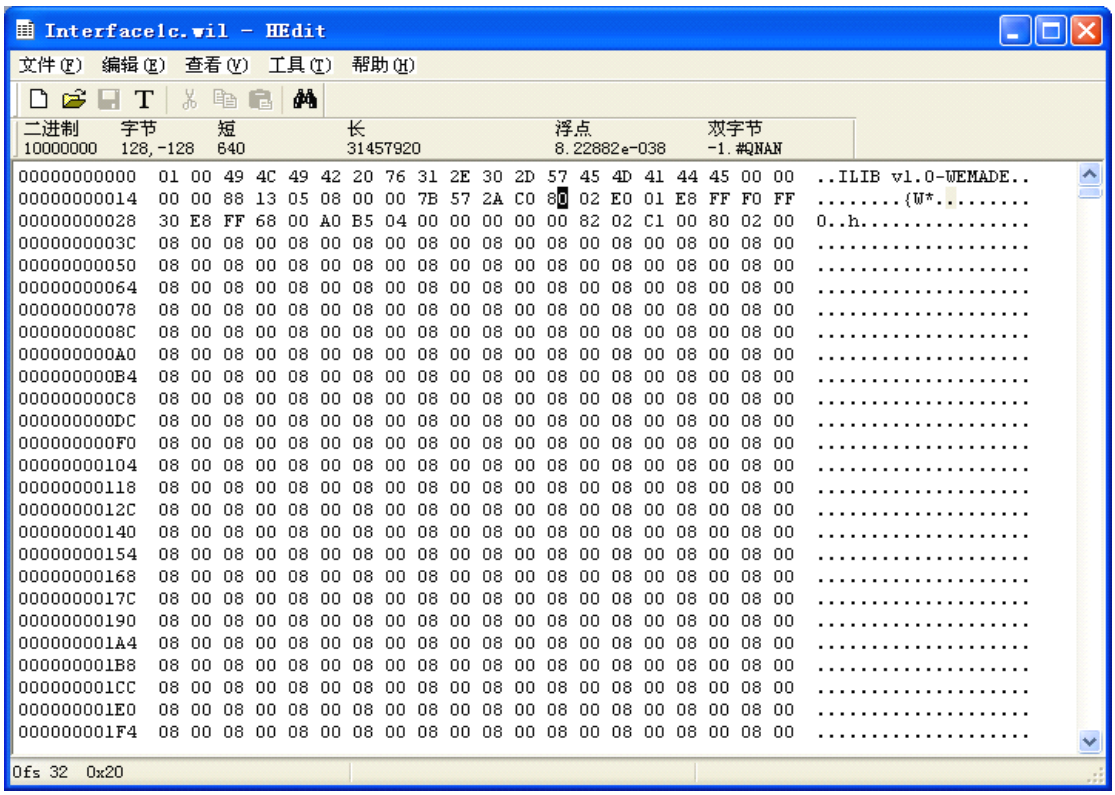
我们用 Wil 编辑打开图片文件夹显示结果如下：

0-2052 刚好为 2053 张图片。

```
//图片数*int 个空间，存放图片地址 （程序处理）
m_stNewWixImgaeInfo.pnPosition = new INT[m_stNewWixImgaeInfo.nIndexCount];
```

我们用 Wil 编辑打开对应的 Wil 文件，发现，果然有 2053 张图片（减 1）。另外我们发现，在 Ofs28 0x1C 的地方= 28 -31 之间的数据总是 20 00 00 00，终于明白，所有的图片起始位置是相同的。

Wil 文件：数据文件。
前面我们说了图象数据的开始位置为 0x20 32。



```
typedef struct tagNEWWILFILEIMAGEINFO
{
    SHORT    shWidth;
    SHORT    shHeight;
    SHORT    shPX;
    SHORT    shPY;
    CHAR     bShadow;
    SHORT    shShadowPX;
    SHORT    shShadowPY;
    DWORD    dwImageLength;
}NEWWILIMAGEINFO, *LPNEWWILIMAGEINFO;
```

WIL 结构只计 17 个字节，前两个字节为长和宽，根据图片起始位置:: 80 02 E0 01，转化为十六进制： 0x280, 0x1E0 刚好就是 600*480 大小的图片。和上面贴图中的 WIL 图片的分辨率是一样的。图片起始位置为:0fs 32: 0x20 图片大小为 288000 起始位置:0x20 32 终止位置：0x65532 为了验证数据是否正确，我们通过 Wil 工具，把第一幅图片导出来，然后用 Hedit 编辑器打开，经过对比，我们发现，数据一致。大小一致。

贴这个贴子，希望大家少走弯路。网上下载的那个版本应该是从传奇 2 改的，传奇 3 的格式。分析一下源码吧，g_xLoginProc.Load(); 之后就加载 m_Image.NewLoad(IMAGE_INTERFACE_1, TRUE, TRUE);

继续读 Wix 文件，

```
    ReadFile(hWixFile, &m_stNewWixImgaeInfo, sizeof(NEWWIXIMAGEINFO)-
sizeof(INT*), &dwReadLen, NULL);

    // WIX 文件头格式 (56Byte)(NEW)
typedef struct tagNEWWIXFILEIMAGEINFO
{
 CHAR szTitle[20];   // 库文件标题 'WEMADE Entertainment inc.' WIL 文件头
 INT  nIndexCount;   // 图片数量
 INT* pnPosition;    // 位置
}NEWWIXIMAGEINFO, *LPNEWWIXIMAGEINFO;
```

不看不知道，一看吓一跳，大家看到了吧，这个是新的 WIX 的定义，不是传奇 2 的，前面分析过传奇 2 的图片： 0x23 地址(含该地址)以前的内容是都相同的，即为：#INDX v1.0-WEMADE Entertainment inc. Ofs44 0x2C 的地方:存放着 0B 00 00 00,高低位转换后为：0xB 转换十进制数为 11(图片数量)Ofs48 0x30 的地方：存放着 38 04 00 00，高低位转换后为：0x438 = 1080, 这个就是图象数据的开始位置。这里才 20 个标题长度。 一看就不对。所以如果你下了网上的传奇 3 的格式，试着读传奇 2 的图片，是不正确的。具体大家可以调试一下，我调试过了，里面的图片数量根本不对。

汗，居然让人郁闷的是， // WIX 文件头格式 (56Byte)

```
typedef struct tagWIXFILEIMAGEINFO
{
 CHAR szTmp[40];      // 库文件标题 'WEMADE Entertainment inc.' WIL 文件头
 INT  nIndexCount;    // 图片数量
 INT* pnPosition;     // 位置
}WIXIMAGEINFO, *LPWIXIMAGEINFO;
```

我用了这种格式也不对。为什么不对，因为我前面分析过了，0xB 转换十进制数为 11(图片数量)Ofs48 0x30 的地方， 看到没有，图片数量的存放地方。 所以赶快改一下数据结构吧，不知道为什么，难道是我版本有问题，我下了几个资源文件，结果发现问题依然存在。看来不是图片的问题。

另外，下面的工程里的图片，如果要运行，不用改数据结构，请到传奇 3 客户端官方网站下载。我下载的是 1.5 版的资源文件。 是传奇 2 的资源文件。祝大家好运吧！

传奇文件类型格式探讨(二)：

```
// WIX 文件头格式 (NEW)
typedef struct tagNEWWIXFILEIMAGEINFO
{
    CHAR    szTitle[20];   // 库文件标题 'WEMADE Entertainment inc.' WIL 文件头
    INT     nIndexCount;   // 图片数量
    INT*    pnPosition;    // 位置
}NEWWIXIMAGEINFO, *LPNEWWIXIMAGEINFO;
```

我们下载一个 Hedit 编辑器打开一个 Wil 文件，分析一下。我们发现 Wix 文件中，0x13 地址(含该地址)以前的内容是都相同的，即为： ‘ ’ 20 个空格。
图片数量： nIndexCount 18

Ofs 20, 0x14 的位置，存放的数据为 12 00 00 00，高低位转换后为：0x12 十制数为 18(图片数量)。Ofs28 0x1C 的地方：存放着 20 00 00 00，高低位转换后为：0x20 = 32, 这个就是图象数据的开始位置。

我们用 Wil 编辑打开对应的 Wil 文件，发现，果然有 17 张图片（减 1）。另外我们发现，在 Ofs28 0x1C 的地方= 28 -31 之间的数据总是 20 00 00 00，终于明白，所有的图片起始位置是相同的。

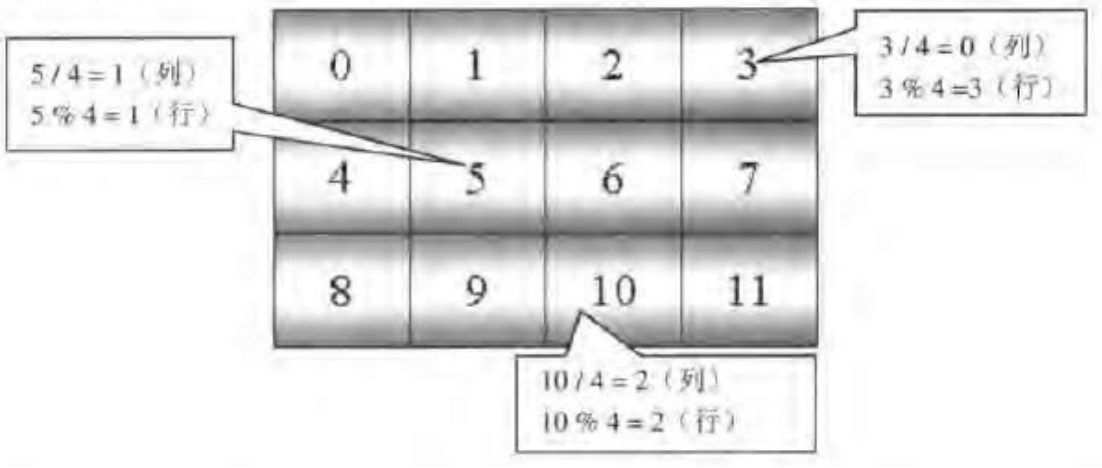
抓图分析，自己就再分析一下吧，和传奇 2 的结构差不多。

地图相关介绍：

```
class CMirMap
{
public:
    MAPFILEHEADER       m_stMapFH;    //地图头（版本号，长宽，颜色等）
    CMapCellInfo*       m_pMapCellInfo;  //地图单元格 （m_xpObjectList 玩家列表）
    char                m_szMapName[16];  //地图名
    char                m_szMapTextName[40];  //地图标识文字
```

GameSrv 启动时，InitializingServer-> LoadMap-> LoadMapData

1. ReadFile 中读出 MAPFILEHEADER 字节大小，
2. nMapSize = m_stMapFH.shWidth * m_stMapFH.shHeight;根据地图长宽，划分成长*宽个单元。
3. 加入 g_xMirMapList 地图列表中。



假设 shWidth=4，shHeight=3。有三幅图片(0,1,2) 每幅图片(20*20)。

```
Int mapBlock[9] = {0,1,2,1
                   2,0,1,2
                   1,2,0,0}
```

列编号 = 索引值/ 每一列的图块个数（行数）

行编号 = 索引值 % 每一列的图块个数（行数）

X 左上角坐标 = 行编号 * 图块宽度

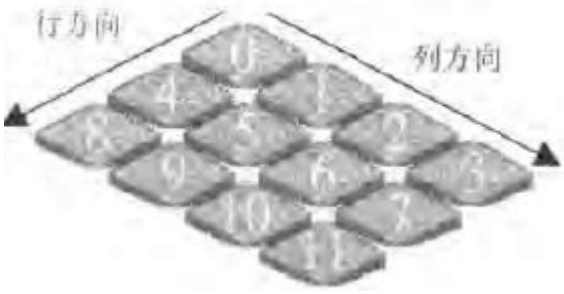
Y 左上角坐标 = 列编号 * 图块的高度

5 号地图：列： 5/4=1， 行：5 % 4 =1

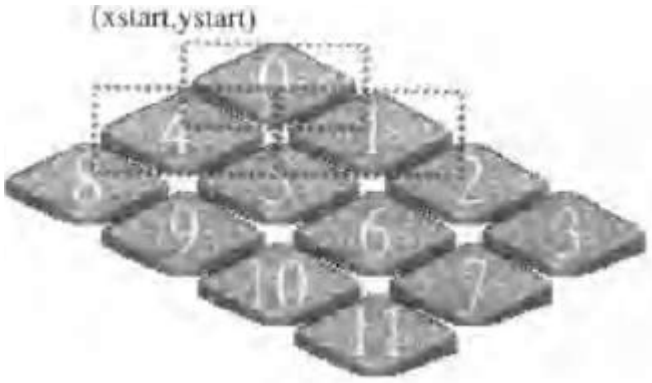
X 坐标：1*20 ， Y 坐标：1*20 坐标为(20, 20)

3 号地图：X 坐标：3*20, Y 坐标：0*20 坐标为（60，0）

斜 45° 度地图：



列编号 = 索引值/ 每一列的图块个数（行数）
行编号 = 索引值 % 每一列的图块个数（行数）



左上点 X 坐标 = xStart + 行编号 * w/2 - 列编号 * (w/2);
左上点 Y 坐标 = yStart + 列编号 * h/2 +行编号 *(h/2);

例如：

5 号地图：列： 5/4=1， 行：5 % 4 =1
xStart 坐标：1*20 ， yStart 坐标：1*20 坐标为(20, 20)
左上点 X 坐标 = 20 + 1*20/2
左上点 Y 坐标 = 20 + 1*20/2 坐标为：(30,30)

AddNewObject 新加入一个用户时,会根据用户 X,Y 坐标得到相应的单元格(GetMapCellInfo)生成一个玩家对象_TOSOBJECT（玩家类型，玩家对象，游戏开始时间）。并且加入 pMapCellInfo->m_xpObjectList 中（地图单元格的一个对象列表中）。

客户端：

传奇的客户端源代码有两个工程，WindHorn 和 Mir2Ex。

先剖析一下 WindHorn 工程。

1. CWHApp、CWHWindow 和 CWHDXGraphicWindow。Window 程序窗口的创建。

CWHApp 派生 CWHWindow，CWHWindow 又派生 CWHDXGraphicWindow。CWHWindow 类中完成窗口的注册和创建。CWHDXGraphicWindow 调用 CWHWindow 完成创建窗口功能，然后再调用 CreateDXG()来初始化 DirectX。

2. CWHDefProcess 派生出 CloginProcess、CcharacterProcess、CgameProcess 三个类。这三个类是客户端处理的核心类。

3. 全局变量:

CWHDXGraphicWindow g_xMainWnd; 主窗口类。
CLoginProcess g_xLoginProc; 登录处理。
CCharacterProcess g_xChrSelProc; 角色选择处理。
CgameProcess g_xGameProc; 游戏逻辑处理。

4. 代码分析：

1.首先从 LoginGate.cpp WinMain 分析：

g_xMainWnd 定义为 CWHDXGraphicWindow 调用 CWHWindow 完成创建窗口功能，然后调用 DirectDrawEnumerateEx 枚举显示设备,(执行回调函数 DXGDriverEnumCallbackEx) 再调用 CreateDXG()来初始化 DirectX(创建 DirectDraw 对象，取得独占和全屏模式，设置显示模式等)。

g_xSound.InitMirSound 创建 CSound 对象。

g_xSpriteInfo.SetInfo();

初始化声音，加载 Socket 库之后，进行 CWHDefProcess*指针赋值(事件绑定)。g_bProcState 变量反应了当前游戏的状态(登录，角色选择，游戏逻辑处理)。调用 Load 初始化一些操作(登录，角色选择，游戏逻辑处理)。进行消息循环。

```
case _LOGIN_PROC:
    g_xLoginProc.RenderScene(dwDelay);
case _CHAR_SEL_PROC:
    g_xChrSelProc.RenderScene(dwDelay);
case _GAME_PROC:
    g_xGameProc.RenderScene(dwDelay);
```

根据 g_bProcState 变量标志，选择显示相应的画面。

2. 接收处理网络消息和接收处理窗口消息。

在不同的状态下(登录，角色选择，游戏逻辑处理)，接收到的消息(网络，窗口消息)会分派到不同的函数中处理的。这里是用虚函数处理(调用子类方法，由实际的父类完成相

应的处理)。

OnMessageReceive 主要处理网络消息。DefMainWndProc 则处理窗体消息(按键，重绘等)，创建窗体类为 CWHDXGraphicWindow，回调函数为：

```
MainWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
if ( m_pxDefProcess )
   m_pxDefProcess->DefMainWndProc(hWnd, uMsg, wParam, lParam);
else
   return MainWndProcDXG(hWnd, uMsg, wParam, lParam);
```

m_pxDefProcess->DefMainWndProc 调用父类的实际处理。

在 WM_PAINT 事件里： g_xClientSocket.ConnectToServer 连接登陆服务器。

登录处理事件：

0. WinMain 主函数调用 g_xLoginProc.Load();加载图片等初始化，设置 g_bProcState 的状态。

1. CLoginProcess::OnKeyDown-> m_xLogin.OnKeyDown->g_xClientSocket.OnLogin; WSAAsyncSelect 模型 ID_SOCKCLIENT_EVENT_MSG,因此，(登录，角色选择，游戏逻辑处理)都回调 g_xClientSocket.OnSocketMessage(wParam, lParam)进行处理。

OnSocketMessage 函数中：FD_READ 事件中：

2. g_bProcState 判断当前状态，_GAME_PROC 时，把 GameGate 的发送过来的消息压入 PacketQ 队列中，再进行处理。否则则调用 OnMessageReceive（虚方法，根据 g_bProcState 状态，调用 CloginProcess 或者是 CcharacterProcess 的 OnMessageReceive 方法）。

3. CloginProcess：调用 OnSocketMessageRecieve 处理返回情况。如果服务器验证失败(SM_ID_NOTFOUND, SM_PASSWD_FAIL)消息，否则收到 SM_PASSOK_SELECTSERVER 消息（SelGate 服务器列表消息）。m_Progress = PRG_SERVER_SELE;进行下一步选择 SelGate 服务器操作。

4. m_xSelectSrv.OnButtonDown->CselectSrv. OnButtonUp-> g_xClientSocket.OnSelectServer(CM_SELECTSERVER)，得到真正的 IP 地址。调用 OnSocketMessageRecieve 处理返回的 SM_SELECTSERVER_OK 消息。并且断开与 loginSrv 服务器连接。 g_xClientSocket.DisconnectToServer();设置状态为 PRG_TO_SELECT_CHR 状态。

角色选择处理：

1. WinMain 消息循环处理：g_xLoginProc.RenderScene(dwDelay)-> RenderScroll-> SetNextProc 调用

```
g_xClientSocket.m_pxDefProc = g_xMainWnd.m_pxDefProcess = &g_xChrSelProc;
g_xChrSelProc.Load();
g_bProcState = _CHAR_SEL_PROC;
```

2. g_xChrSelProc.Load();连接 SelGate 服务器（从 LoginGate 服务器得到 IP 地址）。g_xClientSocket.OnQueryChar();查询用户角色信息，发送消息：CM_QUERYCHR，设置状态为_CHAR_SEL_PROC, m_Progress = PRG_CHAR_SELE; 在 OnSocketMessageRecieve 函数中接收到 SelGate 服务器发送的玩家详细信息。

3. 点击 ChrStart 按钮：g_xChrSelProc.OnLButtonDown-> CSelectChr::OnButtonUp-> g_xClientSocket.OnSelChar->发送 CM_SELCHR 消息到 SelGate 服务器。

4. CClientSocket::OnSocketMessage->CCharacterProcess::OnMessageReceive (SM_STARTPLAY) 接受到 SelGate 服务器发送的 GameGate 服务器 IP 地址，并断开与 SelGate 服务器的连接。m_xSelectChr.m_nRenderState = 2;

5. WinMain 消息循环处理：g_xLoginProc.RenderScene -> m_xSelectChr.Render(nLoopTime);-> CSelectChr::Render(INT nLoopTime)-> m_nRenderState = m_nRenderState + 10; 为 12-> CCharacterProcess::RenderScene 执行

```
m_Progress = PRG_SEL_TO_GAME;
m_Progress  = PRG_PLAY_GAME;
SetNextProc();
```

6. SetNextProc();执行： g_xGameProc.Load(); g_bProcState = _GAME_PROC;进行游戏状态。

游戏逻辑处理：

1. 客户端处理：

CGameProcess::Load() 初始化游戏环境，加载地图等操作，调用 ConnectToServer（m_pxDefProc->OnConnectToServer）连接到 GameGate 游戏网关服务器（DBSrv 处理后经 SelGate 服务器返回的 GameGate 服务器 IP 地址）。

DBSrv -〉DBR_LOADHUMANRCD2

CClientSocket->ConnectToServer 调用 connect 时，由 GameGate 服务器发送 GM_OPEN 消息到 GameSrv 服务器。WSAAsyncSelect I/O 模型回调函数 g_xClientSocket.OnSocketMessage。然后由 m_pxDefProc->OnConnectToServer() 调用 CGameProcess::OnConnectToServer()函数，调用：g_xClientSocket.SendRunLogin。

2. GameGate 服务器 ServerWorkerThread 处理：

GameGate 服务器 ServerWorkerThread 收到消息，ThreadFuncForMsg 处理数据，生成 MsgHdr 结构，并设置

```
MsgHdr.nCode     = 0xAA55AA55; //数据标志
MsgHdr.wIdent    = GM_DATA;    //数据类型
```

3. GameSrv 服务器 ServerWorkerThread 线程处理

GameSrv 服务器 ServerWorkerThread 线程处理调用 DoClientCertification 设置用户信息，及 USERMODE_LOGIN 的状态。并且调用 LoadPlayer(CUserInfo* pUserInfo)函数-> LoadHumanFromDB-> SendRDBSocket 发送 DB_LOADHUMANRCD 请求，返回该玩家的所有数据信息。

4. 客户端登录验证(GameSrv 服务器的线程 ProcessLogin 处理)

用户的验证是由 GameSrv 服务器的线程 ProcessLogin 处理。g_xReadyUserInfoList2 列

表中搜索，判断用户是否已经登录，一旦登录就调用 LoadPlayer(这里两个参数)：

a. 设置玩家游戏状态。m_btCurrentMode 状态为 USERMODE_PLAYGAME

b. 加载物品，个人设置，魔法等。

c. pUserInfo->m_pxPlayerObject->Initialize();初始化用户信息，加载用户坐标，方向，地图。

Initialize 执行流程：

1) AddProcess(this, RM_LOGON, 0, 0, 0, 0, NULL);加入登录消息。
2) m_pMap->AddNewObject 地图中单元格（玩家列表）加入该游戏玩家。OS_MOVINGOBJECT 玩家状态。
3) AddRefMsg(RM_TURN 向周围玩家群发 RM_TURN 消息。以玩家自己为中心，以 24*24 的区域里，向这个区域所属的块里的所有玩家列表发送消息)广播 AddProcess。
4) RecalcAbilitys 设置玩家的能力属性（攻击力（手，衣服），武器力量等）。
5) 循环处理本游戏玩家的附属物品，把这些物品的力量加到（手，衣服等）的攻击力量里。
6) RM_CHARSTATUSCHANGED 消息，通知玩家状态改变消息。
7) AddProcess(this, RM_ABILITY, 0, 0, 0, 0, NULL); 等级
   AddProcess(this, RM_SUBABILITY, 0, 0, 0, 0, NULL);
   AddProcess(this, RM_DAYCHANGING, 0, 0, 0, 0, NULL); 校时
   AddProcess(this, RM_SENDUSEITEMS, 0, 0, 0, 0, NULL); 装备
   AddProcess(this, RM_SENDMYMAGIC, 0, 0, 0, 0, NULL); 魔法
   SysMsg(szMsg, 1) 攻击力

并把用户数据从 g_xReadyUserInfoList2 列表中删除。

说明：

一旦通过验证，就从验证列表中该玩家，改变玩家状态，LoadPlayer 加载用户资源（地图中加入用户信息，向用户 24*24 区域内的块内玩家发送上线消息 GameSrv 广播新玩家上线（坐标）的消息。向该新玩家发送玩家信息（等级，装备，魔法，攻击力等）。

5．接受登录成功后，接收 GameSrv 服务器发送的消息：

CGameProcess::OnSocketMessageRecieve

接收 GameGate 发送的消息：CClientSocket::OnSocketMessage 的 FD_READ 事件中，PacketQ.PushQ((BYTE*)pszPacket);把接收到的消息，压入 PacketQ 队列中。处理 PacketQ 队列数据是由 CGameProcess::Load()时调用 OnTimer 在 CGameProcess::OnTimer 中处理的，处理过程为：

```
OnMessageReceive;
ProcessPacket();
ProcessDefaultPacket();
```

OnMessageReceive 函数;

1. 判断是否收到心跳数据包，发送'*'，发送心跳数据包。
2. 调用 OnSocketMessageRecieve 函数。这个函数里面详细处理了客户端的游戏执行逻辑。如果是‘+’开头（数据包）则调用 OnProcPacketNotEncode 处理这种类型数据包。否则得到_TDEFAULTMESSAGE 数据包，进行游戏逻辑处理。

OnProcPacketNotEncode 说明：

收到 GameSrv 服务器的相应消息：

"GOOD"：可以执行动作。 m_bMotionLock 为假。

"FAIL"：不允许执行动作。人物被拉回移动前位置。

"LNG"：

"ULNG"：

"WID"：

"UWID"：

"FIR"：

"UFIR"：

"PWR"：

3. CGameProcess::OnSocketMessageRecieve(char *pszMsg)函数。处理游戏相关的消息。

SM_SENDNOTICE： 服务器提示信息：

SM_NEWMAP: 用户登录后，服务器发送的初始化地图消息。

SM_LOGON： 用户登录消息(服务器处理后返回结果)。用户登录成功后，在本地创建游戏对象，并发送消息，请求返回用户物品清单（魔法，等级，物品等）。

SM_MAPDESCRIPTION: 得到服务器发送的地图的描述信息。

SM_ABILITY：服务器发送的本玩家金钱，职业信息。

SM_WINEXP：

SM_SUBABILITY ： 服务器发送的玩家技能(魔法，杀伤力，速度，毒药，中毒恢复，生命恢复，符咒恢复)

SM_ SM_SENDMYMAGIC: 用户魔法列表信息。

SM_MAGIC_LVEXP: 魔法等级列表。

SM_BAGITEMS：用户物品清单 （玩家 CM_QUERYBAGITEMS 消息）

SM_SENDUSEITEMS：用户装备清单

SM_ADDITEM： 拣东西

SM_DELITEM： 丢弃物品。

等等。

4. 部分数据未处理，加入 m_xWaitPacketQueue 队列中由 ProcessPacket 处理。

新登录游戏玩家：在 OnSocketMessageRecieve 函数中依次收到的消息为：

1. GameSrv 服务器 ProcessLogin 线程返回 GameGate 服务器后返回的：

AddProcess(this, RM_LOGON, 0, 0, 0, 0, NULL);加入登录消息。

SM_NEWMAP, SM_LOGON, SM_USERNAME, SM_MAPDESCRIPTION 消息

AddProcess(this, RM_ABILITY, 0, 0, 0, 0, NULL); 等级

SM_ABILITY

AddProcess(this, RM_SUBABILITY, 0, 0, 0, 0, NULL);

SM_SUBABILITY

AddProcess(this, RM_DAYCHANGING, 0, 0, 0, 0, NULL); 校时

SM_DAYCHANGING

AddProcess(this, RM_SENDUSEITEMS, 0, 0, 0, 0, NULL); 装备

SM_SENDUSEITEMS
AddProcess(this, RM_SENDMYMAGIC, 0, 0, 0, 0, NULL); 魔法
SM_SENDMYMAGIC

客户端收到消息后相应的处理：

SM_NEWMAP 接受地图消息 OnSvrMsgNewMap
初始化玩家坐标，m_xMyHero.m_wPosX = ptdm->wParam;
m_xMyHero.m_wPosY = ptdm->wTag;
加载地图文件 m_xMap.LoadMapData(szMapName);
设置场景。 m_xLightFog.ChangeLightColor(dwFogColor);

SM_LOGON 返回登录消息 OnSvrMsgLogon
m_xMyHero.Create 初始化玩家信息（头发，武器，加载图片等），设置玩家地图 m_xMyHero.SetMapHandler(&m_xMap)，创建用户魔法。加入 m_xMagicList 列表，pxMagic->CreateMagic, m_xMagicList.AddNode(pxMagic);并向服务器发送 CM_QUERYBAGITEMS 消息（用户物品清单，血，气，衣服，兵器等）。

SM_USERNAME	获取玩家的游戏角色名字。
SM_MAPDESCRIPTION	地图对应的名字。
SM_BAGITEMS	用户物品清单 （玩家 CM_QUERYBAGITEMS 消息）
SM_CHARSTATUSCHANGED	通知玩家状态改变消息（攻击力，状态）。
SM_ABILITY	玩家金钱，职业
SM_SUBABILITY	玩家技能(魔法，杀伤力，速度，毒药，中毒恢复，生命恢复，符咒恢复)
SM_DAYCHANGING	返回游戏状态。（Day, Fog）让客户端随着服务器的时间，加载不同场景。
SM_SENDUSEITEMS	用户装备清单
SM_SENDMYMAGIC	用户魔法列表信息。

总结：

客户端连接到 GameGate 游戏网关服务器，并通过 GameSrv 服务器验证之后，就会收到 GameSrv 服务器发来的消息。主要是地图消息，登录消息，玩家的装备，技能，魔法，个人设置等等。GameSrv 把地图分成若干块，把该玩家加入其中一块，并加入这一块的用户对象列表中，设置其状态为 OS_MOVINGOBJECT。客户端加载地图，设置场景，设置自己的玩家状态（此时还没有怪物和其它玩家，所以玩家还需要接收其它游戏玩家和怪物的清单列表）。

6. **接收怪物，商人，其它玩家的消息：**

ProcessUserHuman:(其它玩家—服务器处理)

CPlayerObject->SearchViewRange();
CPlayerObject->Operate();

遍历 UserInfoList 列表，依次调用每个 UserInfo 的 Operate 来处理命令队列中的所有操作; pUserInfo->Operate()调用 m_pxPlayerObject->Operate()调用。根据分发消息（RM_TURN）向客户端发送 SM_TURN 消息。GameSrv 广播新玩家上线（坐标）的消息。向该新玩家发送玩家信息（等级，装备，魔法，攻击力等）。

玩家，移动对象：

1. 遍历 m_xVisibleObjectList 列表，所有(玩家，商人，怪物）发送调用 AddProcess (RM_TURN 向周围玩家发送消息)。

地图：

2. 遍历 m_xVisibleItemList，发送 AddProcess(this, RM_ITEMSHOW 消息更新地图。
3. 遍历 m_xVisibleEventList，发送 AddProcess(this, RM_SHOWEVENT

ProcessMonster 线程：(怪物—服务器处理)

GameSrv 服务器在 ProcessMonster 线程：创建不同的 CMonsterObject 对象，并且加入 xMonsterObjList 列表和 pMapCellInfo->m_xpObjectList 列表中，然后再调用 CMonsterObject::SearchViewRange()更新视线范围内目标，根据 g_SearchTable 计算出搜索坐标，转换为相应的地图单元格，遍历所有可移动生物，加入 m_xVisibleObjectList 列表，调用 Operate；Operate 遍历 m_DelayProcessQ 列表，过滤出 RM_DOOPENHEALTH, RM_STRUCK 和 RM_MAGSTRUCK 三个事件(恢复生命值，攻击，魔法攻击)，并处理。

ProcessMerchants 线程：(商人--服务器处理)

1). 遍历 g_pMerchantInfo 结构(根据 nNumOfMurchantInfo 数量)。得到商人类型相关的地图，创建商人对象，设置不同的编号，坐标，头像及所属地图。在该地图中加入该商人，且在 g_xMerchantObjList 商人清单中加入该商人。

2). 遍历 g_xMerchantObjList, SearchViewRange，对每个商人更新视线范围内目标

a). 遍历 m_xVisibleObjectList，设置每个 pVisibleObject->nVisibleFlag = 0;设置状态（删除）。

b). 搜索 VisibleObjectList 列表，(服务器启动时 InitializingServer 加载 searchTable.tbl)，根据坐标，找到相应的地图单元格。然后遍历 pMapCellInfo->m_xpObjectList 列表，判断如果为 OS_MOVINGOBJECT 标志，调用 UpdateVisibleObject 函数，该函数遍历 m_xVisibleObjectList 列表，如果找到该商人对象，则 pVisibleObject->nVisibleFlag = 1;否则判断 pNewVisibleObject 对象，设置 nVisibleFlag 为 2，设置对象为该商人实体，然后加入 m_xVisibleObjectList 列表中。

总结：循环列表，找出地图单元格中的所有玩家，把所有玩家(OS_MOVINGOBJECT)加入到 m_xVisibleObjectList 列表中。

c). 遍历 m_xVisibleObjectList 列表，(pVisibleObject->nVisibleFlag == 0)则删除该 pVisibleObject 对象。

d). RunRace 调用 AddRefMsg 向周围玩家发送 SM_TURN 和 SM_HIT

客户端收到消息后相应的处理：

1. CGameProcess::OnSocketMessageRecieve 加入 m_xWaitPacketQueue 队列

遍历 m_xVisibleObjectList 队列中所有移动物体(角色)：

RM_DISAPPEAR 消失(SM_DISAPPEAR) ProcessDefaultPacket 函数

RM_DEATH 死亡(SM_NOWDEATH, SM_DEATH)

CHero::OnDeath 其它玩家。

CActor::OnDeath 怪物。

//g_xGameProc.m_xMagicList

RM_TURN 移动

SM_TURN 消息处理

遍历 m_xVisibleItemList 队列中所有移动物体(地图)：

RM_ITEMHIDE　从 m_stMapItemList 列表中删除该移动对象

RM_ITEMSHOW　遍历 m_stMapItemList，如果不存在，则创建一个 GROUNDITEM 结构，并加入 m_stMapItemList 列表中。

```
typedef struct tagGROUNDITEM
{
    INT             nRecog;
    SHORT           shTileX;
    SHORT           shTileY;
    WORD            wLooks;
    CHAR            szItemName[40];
}GROUNDITEM, *LPGROUNDITEM;
```

遍历 m_xVisibleEventList 队列中所有移动物体(事件)：

RM_HIDEEVENT

RM_SHOWEVENT

2. 部分数据未处理，加入 m_xWaitPacketQueue 队列中由 ProcessPacket 处理。CClientSocket::OnSocketMessage 的 FD_READ 事件中，PacketQ.PushQ 把接收到的消息，压入 PacketQ 队列中。处理 PacketQ 队列数据是由 CGameProcess::Load()时调用 OnTimer 在 CGameProcess::OnTimer 中处理的，处理过程为：

OnTimer -> ProcessPacket -> ProcessPacket 处理 m_xWaitPacketQueue 队列消息（OnSocketMessageRecieve 函数中未处理的消息）。

ProcessPacket 函数处理流程：

1. 处理本玩家（SM_NOWDEATH, SM_DEATH, SM_CHANGEMAP, SM_STRUCK）

a. 如果接收到消息是 SM_NOWDEATH 或 SM_DEATH 则加入 m_xPriorPacketQueue 队列。

b. 如果接收到消息是 SM_CHANGEMAP 则调用 LoadMapChanged,设置场景。

c. SM_STRUCK 处理受攻击（本玩家，或者其它的玩家，NPC 等）。

2. 其它消息：m_xMyHero.StruckMsgReassign();

m_xMyHero.m_xPacketQueue.PushQ((BYTE*)lpPacketMsg);

判断服务器发送来的消息 ID 是否相同。m_xMyHero.m_dwIdentity 在登录成功的时候由服务器发送的用户消息获取的。

if (lpPacketMsg->stDefMsg.nRecog == m_xMyHero.m_dwIdentity)

如果是服务器端游戏玩家自己发送的消息,则处理自己的消息。否则如果是其它玩家(怪物)发送的消息，遍历 m_xActorList 列表，判断该对象是否存在，如果该不存在，则根据 stFeature.bGender 的类型

_GENDER_MAN:　创建一个 CHero 对象，加入到 m_xActorList 列表中。

_GENDER_WOMAN:

_GENDER_NPC:　创建一个 CNPC 对象，加入到 m_xActorList 列表中。

_GENDER_MON:　创建一个 CActor 对象，加入到 m_xActorList 列表中。

然后 pxActor->m_xPacketQueue.PushQ 然后把消息压入该对象的 xPacketQueue 列表

中。

总结：ProcessPacket 处理 CClientSocket 类接受的消息(m_xWaitPacketQueue)，判断是否是服务器发送给自己的消息，处理一些发送给自己的重要消息，其它消息处理则加入 m_xMyHero.m_xPacketQueue 队列中，然后再遍历 m_xActorList 队列，判断如果服务器端发来的消息里的玩家(NPC，怪物)，在 m_xActorList 队列中找不到，就判断一个加入 m_xActorList 列表中，并且把该消息压入 pxActor->m_xPacketQueue 交给该 NPC 去处理该事件。

而 xPacketQueue 队列的消息分别由该对象的 UpdatePacketState 处理，如下：

BOOL CActor::UpdatePacketState() ,BOOL CNPC::UpdatePacketState()

BOOL CHero::UpdatePacketState()。

ProcessDefaultPacket 函数：

处理 CGameProcess::OnSocketMessageRecieve 中 SM_CLEAROBJECT 消息：

处理（SM_DISAPPEAR，SM_CLEAROBJECT）消息。

遍历 m_xWaitDefaultPacketQueue 消息列表

SM_DISAPPEAR 和 SM_CLEAROBJECT：

遍历 m_xActorList 列表，清除 pxActor->m_xPacketQueue 队列内所有消息。

m_xActorList.DeleteCurrentNodeEx();从对列中删除该对象。

CHero* pxHero = (CHero*)pxActor; delete((CHero*)pxHero);销毁该玩家。

游戏循环处理： CGameProcess::RenderScene(INT nLoopTime)函数：

主要流程如下：

wMoveTime += nLoopTime; 判断 wMoveTime>100 时，bIsMoveTime 置为真。

1. m_xMyHero.UpdateMotionState(nLoopTime, bIsMoveTime);处理本玩家消息。
   a. UpdatePacketState 函数：
      遍历 m_xPriorPacketQueue 队列，如果有 SM_NOWDEATH 或 SM_DEATH 消息，则优先处理。
      处理 m_xPacketQueue 队列中消息。
      SM_STRUCK:
      SM_RUSH
      SM_BACKSTEP
      SM_FEATURECHANGED:
      SM_OPENHEALTH:
      SM_CLOSEHEALTH:
      SM_CHANGELIGHT:
      SM_USERNAME:
      SM_CHANGENAMECOLOR:
      SM_CHARSTATUSCHANGE:
      SM_MAGICFIRE:
      SM_HEALTHSPELLCHANGED:

2. CheckMappedData 函数：遍历 m_xActorList 列表分别调用
    CActor::UpdateMotionState(INT nLoopTime, BOOL bIsMoveTime)
    CNPC::UpdateMotionState(INT nLoopTime, BOOL bIsMoveTime)
    CMyHero::UpdateMotionState(INT nLoopTime, BOOL bIsMoveTime)
  处理自己消息。

```
CHero::UpdatePacketState()
    case SM_SITDOWN:
    case SM_BUTCH:
    case SM_FEATURECHANGED:
    case SM_CHARSTATUSCHANGE:
    case SM_OPENHEALTH:
    case SM_CLOSEHEALTH:
    case SM_CHANGELIGHT:
    case SM_USERNAME:
    case SM_CHANGENAMECOLOR:
    case SM_HEALTHSPELLCHANGED:
    case SM_RUSH:
    case SM_BACKSTEP:
    case SM_NOWDEATH:
    case SM_DEATH:
    case SM_WALK:
    case SM_RUN:
    case SM_TURN:
    case SM_STRUCK:
    case SM_HIT:
    case SM_FIREHIT:
    case SM_LONGHIT:
    case SM_POWERHIT:
    case SM_WIDEHIT:
    case SM_MAGICFIRE:
    case SM_SPELL:

CNPC::UpdatePacketState()
    case SM_OPENHEALTH:
    case SM_CLOSEHEALTH:
    case SM_CHANGELIGHT:
    case SM_USERNAME:
    case SM_CHANGENAMECOLOR:
    case SM_HEALTHSPELLCHANGED:
    case SM_TURN:
    case SM_HIT:

CActor::UpdatePacketState()
```

```
case SM_DEATH:      SetMotionFrame(_MT_MON_DIE, bDir);
case SM_WALK:       SetMotionFrame(_MT_MON_WALK, bDir);
case SM_TURN:       SetMotionFrame(_MT_MON_STAND, bDir);
case SM_DIGUP:      SetMotionFrame(_MT_MON_APPEAR, bDir);
case SM_DIGDOWN:    SetMotionFrame(_MT_MON_APPEAR, bDir);
case SM_FEATURECHANGED:
case SM_OPENHEALTH:
case SM_CLOSEHEALTH:
case SM_CHANGELIGHT:
case SM_CHANGENAMECOLOR:
case SM_USERNAME:
case SM_HEALTHSPELLCHANGED:
case SM_BACKSTEP:       SetMotionFrame(_MT_MON_WALK, bDir);
case SM_STRUCK:             SetMotionFrame(_MT_MON_HITTED, m_bCurrDir);
case SM_HIT:            SetMotionFrame(_MT_MON_ATTACK_A, bDir);
case SM_FLYAXE:
case SM_LIGHTING:
case SM_SKELETON:
```

收到多个 NPC，玩家发送的 SM_TURN 消息：由下面对象调用处理：

CHero::OnTurn
CNPC::OnTurn
CActor::OnTurn

根据服务器发送的消息，(创建一个虚拟玩家 NPC，怪物，在客户端)，根据参数，初始化该对象设置(方向，坐标，名字，等级等)。在后面的处理中绘制该对象到 UI 界面中(移动对象的 UI 界面处理)。

SetMotionFrame(_MT_MON_STAND, bDir); m_bCurrMtn := _MT_MON_STAND
m_dwFstFrame , m_dwEndFrame , m_wDelay 第一帧，最后一帧，延迟时间。

3. AutoTargeting 自动搜索目标(NPC,怪物，玩家等)

4. RenderObject 补偿对象时间

5. RenderMapTileGrid
   m_xMagicList，处理玩家魔法后，UI 界面的处理。

6. m_xSnow, m_xRain, m_xFlyingTail, m_xSmoke, m_xLightFog 设置场景 UI 界面处理。

7. m_xMyHero.ShowMessage(nLoopTime); 显示用户(UI 处理)
   m_xMyHero.DrawHPBar(); 显示用户 HP 值。
   遍历 m_xActorList，处理所有 NPC 的 UI 界面重绘

```
pxHero->ShowMessage(nLoopTime);
pxHero->DrawHPBar();
```

8. DropItemShow 下拉显示。

9. 判断 m_pxMouseTargetActor(玩家查看其它玩家，NPC，怪物时)
   g_xClientSocket.SendQueryName 向服务器提交查询信息。
   m_pxMouseOldTargetActor = m_pxMouseTargetActor; 保存该对象
   m_pxMouseTargetActor->DrawName(); 重绘对象名字(UI 界面显示)

下面分析一下用户登录之后的流程：

从前面的分析中可以看到，该用户玩家登录成功之后，得到了服务器发送来的各种消息。处理也比较复杂，同时有一定的优先级处理。并且根据用户登录后的 XY 坐标，向用户发送来了服务器 XY 坐标为中心附近单元格中的所有玩家(NPC,怪物)的 SM_TURN 消息。

客户端根据数据包的标志，创建这些 NPC，设置属性，并且把它们加入 m_xActorList 对列中。最后在 UI 界面上绘制这些对象。

现在假设玩家开始操作游戏：

传奇的客户端源代码工程 WindHorn

一、CWHApp 派生 CWHWindow 和 CWHDXGraphicWindow。

二、CWHDefProcess 派生出 CloginProcess、CcharacterProcess、CgameProcess

客户端 WinMain 调用 CWHDXGraphicWindow g_xMainWnd;创建一个窗口。

客户端 CWHDXGraphicWindow 在自己的 Create 函数中调用了 CWHWindow 的 Create 来创建窗口，然后再调用自己的 CreateDXG()来初始化 DirectX。

消息循环：

因此，当客户端鼠标单击的时候，先调用 CWHWindow 窗口的回调函数 WndProc，即：g_pWHApp->MainWndProc g_pWHApp 定义为：static CWHApp* g_pWHApp = NULL; 在 CWHApp 构造函数中赋值为：g_pWHApp = this;

g_pWHApp->MainWndProc 便调用了 CWHApp::MainWndProc，这是一个虚函数，实际上则是调用它的派生类 CWHDXGraphicWindow::MainWndProc。

```
if ( m_pxDefProcess )
   return m_pxDefProcess->DefMainWndProc(hWnd, uMsg, wParam, lParam);
```

根据 g_xMainWnd.m_pxDefProcess 和全局变量 g_bProcState 标记当前的处理状态。调用

```
CLoginProcess->DefMainWndProc
CCharacterProcess->DefMainWndProc
CGameProcess->DefMainWndProc
```

当用户进行游戏之后，点击鼠标左键，来处理玩家走动的动作：

客户端执行流程：(玩家走动)

CGameProcess::OnLButtonDown(WPARAM wParam, LPARAM lParam)函数：该函数的处理流程:

1. g_xClientSocket.SendNoticeOK();如果点中 CnoticeBox 则 m_xNotice.OnButtonDown if m_xMsgBtn.OnLButtonDown 则调用 g_xClientSocket.SendNoticeOK()方法，发送还

CM_LOGINNOTICEOK 消息。

2. m_pxSavedTargetActor = NULL;设置为空。CInterface::OnLButtonDown 函数会判断鼠标点击的位置(CmirMsgBox, CscrlBar,CgameBtn，GetWindowInMousePos)

a. g_xClientSocket.SendItemIndex(CM_DROPITEM 丢弃物品)

游戏服务器执行流程 m_pxPlayerObject->Operate()调用

m_pUserInfo->UserDropGenItem

m_pUserInfo->UserDropItem	删除普通物品。
SM_DROPITEM_SUCCESS	返回删除成功命令
SM_DROPITEM_FAIL	返回删除失败命令

b. 遍历 m_stMapItemList 列表(存储玩家,怪物,NPC)， g_xClientSocket.SendPickUp 发送 CM_PICKUP 命令。

游戏服务器：m_pxPlayerObject->Operate()调用 PickUp(捡东西)消息处理：

m_pMap->GetItem(m_nCurrX, m_nCurrY) 返回地图里的物体(草药,物品,金子等)

1. **memcmp(pMapItem->szName, g_szGoldName 如果是黄金：**

m_pMap->RemoveObject 从地图中移走该的品。

if (m_pUserInfo->IncGold(pMapItem->nCount))增加用户的金钱(向周转玩家发送 RM_ITEMHIDE 消息，隐藏该物体，GoldChanged()，改变玩家的金钱。否则，把黄金返回地图中。

2. **m_pUserInfo->IsEnoughBag()**

如果玩家的还可以随身带装备(空间)。m_pMap->RemoveObject 从地图中移走该的品。UpdateItemToDB，更新用户信息到数据库。(向周转玩家发送 RM_ITEMHIDE 消息，隐藏该物体，SendAddItem(lptItemRcd)向本玩家发送捡到东西的消息。m_pUserInfo->m_lpTItemRcd.AddNewNode 并把该物品加入自己的列表中。

c. if m_pxMouseTargetActor g_xClientSocket.SendNPCClick 发送 CM_CLICKNPC 命令。

客户端 RenderScene 调用 m_pxMouseTargetActor = NULL; CheckMappedData(nLoopTime, bIsMoveTime)处理，如果鼠标在某个移动对象的区域内就会设置 m_pxMouseTargetActor 为该对象。

如果是 NPC：

if (m_pxMouseTargetActor->m_stFeature.bGender == _GENDER_NPC) g_xClientSocket.SendNPCClick(m_pxMouseTargetActor->m_dwIdentity);

CM_CLICKNPC 消息：

否则：

m_xMyHero.OnLButtonDown

d. 否则 m_xMyHero.OnLButtonDown

先判断 m_xPacketQueue 是否有数据，有则先处理。返回。

判断 m_pxMap->GetNextTileCanMove 根据坐标，判断地图上该点属性是否可以移动到该位置：

可移动时：

人：SetMotionState(_MT_WALK

骑马：SetMotionState(_MT_HORSEWALK

不可移动时：

人：SetMotionState(_MT_STAND, bDir);

骑马：SetMotionState(_MT_HORSESTAND, bDir);

SetMotionState 函数：

判断循环遍历目标点的周围八个坐标，如果发现是一扇门，则向服务器发送打开这扇门的命令。g_xClientSocket.SendOpenDoor,否则则发送 CM_WALK 命令到服务器。

m_bMotionLock = m_bInputLock = TRUE; 设置游戏状态

m_wOldPosX = m_wPosX; 保存玩家 X 点

m_wOldPosY = m_wPosY; 保存玩家 Y 点

m_bOldDir = m_bCurrDir; 保存玩家方向

然后调用 SetMotionFrame 设置 m_bCurrMtn = _MT_WALK，方向等游戏状态。设置 m_bMoveSpeed = _SPEED_WALK（移动速度 1）。m_pxMap->ScrollMap 设置地图的偏移位置（m_shViewOffsetX, m_shViewOffsetY）。然后滚动地图，重绘玩家由 CGameProcess::RenderScene CGameProcess::RenderObject->DrawActor 重绘。

游戏服务器执行流程：（玩家走动）

GameSrv 服务器 ProcessUserHuman 线程处理玩家消息：

遍历 UserInfoList 列表,依次调用每个 UserInfo 的 Operate 来处理命令队列中的所有操作;pUserInfo->Operate()调用 m_pxPlayerObject->Operate()调用。

判断玩家 if (!m_fIsDead)，如果已死，则发送_MSG_FAIL 消息。我们在前面看到过，该消息是被优先处理的。否则则调用 WalkTo，并发送_MSG_GOOD 消息给客户端。

WalkTo 函数的流程：

1） WalkNextPos 根据随机值产生，八个方向的坐标位置。

2） WalkXY 怪物走动到一个坐标值中。

CheckDoorEvent 根据 pMapCellInfo->m_sLightNEvent 返回四种状态。

a） 要移动的位置是一扇门 _DOOR_OPEN

b） 不是一扇门 _DOOR_NOT

c） 是一扇门不可以打开返回 _DOOR_MAPMOVE_BACK 或_DOOR_MAPMOVE_FRONT 玩家前/后移动

3） 如果_DOOR_OPEN 则发送 SM_DOOROPEN 消息给周围玩家。

4） m_pMap->CanMove 如果可以移动,则 MoveToMovingObject 从当前点移动到另一点。并发送 AddRefMsg(RM_WALK)给周围玩家。

AddRefMsg 函数，我们在后面的服务器代码里分析过：它会根据 X，Y 坐标，在以自己坐标为中心周围 26*26 区域里面，按地图单元格的划分，遍历所有单元格，再遍历所有单元格内的玩家列表，广播发送 RM_WALK 消息。

客户端执行流程：（反馈服务器端本玩家走动）

1. 服务器如果发送_MSG_FAIL 由客户端 CGameProcess::OnProcPacketNotEncode 处理。

m_xMyHero.SetOldPosition();

人： SetMotionFrame(_MT_STAND

AdjustMyPostion(); 重绘地图

m_bMotionLock = m_bInputLock = FALSE;

骑马：SetMotionFrame(_MT_HORSESTAND

AdjustMyPostion(); 重绘地图
m_bMotionLock = m_bInputLock = FALSE;

2. 服务器如果发送_MSG_GOOD, 由客户端 CGameProcess::OnProcPacketNotEncode 处理。m_xMyHero.m_bMotionLock = FALSE;

其它客户端执行流程：(反馈服务器端其它玩家)

1. 其它玩家：
人： SetMotionFrame(_MT_WALK, bDir);
骑马：SetMotionFrame(_MT_HORSEWALK, bDir);
m_bMoveSpeed = _SPEED_WALK;
SetMoving(); 设置 m_shShiftPixelX， m_shShiftPixelY 坐标。
2. NPC，怪物：
SetMotionFrame(_MT_MON_WALK, bDir);
m_bMoveSpeed = _SPEED_WALK;
SetMoving(); 设置 m_shShiftPixelX， m_shShiftPixelY 坐标。
CGameProcess::RenderObject->DrawActor(m_shShiftPixelX， m_shShiftPixelY)
重绘发消息的玩家，NPC 怪物位置。

怪物切换地图执行流程：

CMonsterObject::Run()函数：
1. 判断是否超过边界，如果超过边界，则调用 SpaceMove 函数：
SpaceMove(m_nTargetX, m_nTargetY, m_pMasterObject->m_pMap);
m_pMap->RemoveObject，在当前地图上删除该对象，
m_pMap = pMirMap; m_pMap->AddNewObject 在新地图里加入该对象，
RM_CLEAROBJECTS **(ProcessDefaultPacket 函数处理)**，RM_CHANGEMAP(OnSvrMsgNewMap 函数：初始化玩家坐标，加载地图，设置场景等。)发送给本玩家 RM_SPACEMOVE_SHOW 发送给周围玩家。

玩家接受服务器操作命令顺序：

SM_NEWMAP	加载地图
SM_LOGON	创建英雄，魔法等。
SM_USERNAME	设置英雄的名字
SM_MAPDESCRIPTION	得到地图名称(根据序号)
SM_ABILITY	设置英雄的黄金数量和职业
SM_SUBABILITY	设置英雄的 m_stSubAbility 属性
SM_DAYCHANGING	设置本地与服务器的时间差 CGameProcess::OnTimer (定时处理数据)
SM_SENDUSEITEMS	设置物品
SM_SENDMYMAGIC	设置魔法
SM_SYSMESSAGE	OnSvrMsgHear

ProcessPacket
SM_NEWMAP

SM_BAGITEMS
ProcessLogin 线程处理完之后设置玩家状态 USERMODE_PLAYGAME,
由 ProcessUserHuman 线程完成玩家的 pUserInfo->Operate();及 pGateInfo->xSend();

LoginGate 服务器

服务器端：

1. 首先从 LoginGate.cpp WinMain 分析：
   1) CheckAvailableIOCP ： 检查是不是 NT，2000 的系统（IOCP）
   2) InitInstance: 初始化界面，加载 WSAStartup
   3) MainWndProc 窗口回调函数.
2. MainWndProc.CPP 中分析回调函数 MainWndProc

```
switch (nMsg)
 {
     case _IDM_CLIENTSOCK_MSG:
     case WM_COMMAND:
     case WM_CLOSE:
```

g_ssock Local 7000 游戏登陆端口
g_csock Remote 5000 发送到 logsrv 服务器上的套接字

1）_IDM_CLIENTSOCK_MSG 消息：处理与 logsrv 回调通讯事件。
调用：OnClientSockMsg,该函数是一个回调函数：
当启动服务之后，ConnectToServer 函数将(_IDM_CLIENTSOCK_MSG 消息 FD_CONNECT|FD_READ|FD_CLOSE)传入 WSAAsyncSelect 函数。在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知。函数 OnClientSockMsg,主要完成与 logsrv 服务器之间的通信（心跳，转发客户端数据包等）

```
switch (WSAGETSELECTEVENT(lParam))
   {
       case FD_CONNECT:
       case FD_CLOSE:
       case FD_READ:
```

FD_CONNECT：(重新连接情况)
A. CheckSocketError 返回正常时：
a). ConnectToServer 函数首先在服务启动的时候执行一次。回调 FD_CONNECT
b). 连接 logsrv 时,开启 ThreadFuncForMsg 线程，把从客户端发送的数据(g_xMsgQueue, FD_READ 事件读到的 logSrv 服务器发来的数据)投递 I/O，利用 IOCP 模型，发送到客户端。SleepEx 挂起线程。至到一个 I/O 完成回调函数被调用。 一个异步过程调用排队到此线

程。

ThreadFuncForMsg 线程检测(从 logSrv 收到的 g_xMsgQueue 数据包-心跳，处理包)。i/o 投递，利用 IOCP 发送给客户端。

```
if (nSocket = AnsiStrToVal(pszFirst + 1)) //得到 socket
WSASend((SOCKET)nSocket, &Buf, 1, &dwSendBytes, 0, NULL,
```

c). 终止定时器_ID_TIMER_CONNECTSERVER

```
KillTimer(g_hMainWnd, _ID_TIMER_CONNECTSERVER);
```

d). 设置_ID_TIMER_KEEPALIVE 定时器 （心跳数据包）

```
SetTimer(g_hMainWnd, _ID_TIMER_KEEPALIVE
```

调用定时器回调函数 OnTimerProc: 定时发关心跳数据包到 logsrv 服务器。SendExToServer(PACKET_KEEPALIVE);

B. 如果 socket 断开，设置_ID_TIMER_CONNECTSERVER 定时器 ConnectToServer 尝试重新连接服务器。

```
_ID_TIMER_CONNECTSERVER, (TIMERPROC)OnTimerProc);
```

FD_CLOSE:

断开与 logsrv 服务器 SOCKET 连接，OnCommand(IDM_STOPSERVICE, 0); 回调函数处理 IDM_STOPSERVICE。

FD_READ:

接收 logsrv 服务器发送的数据包（心跳,登陆验证，selCur 服务器地址），把数据加入缓冲区(g_xMsgQueue)中。

2）WM_COMMAND:

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

3）WM_CLOSE:

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

```
WSACleanup();
PostQuitMessage(0); //WM_DESTROY 消息
```

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

InitServerSocket：函数：

1) AcceptThread 线程：

Accept 之后生成一个 CSessionInfo 对象，pNewUserInfo->sock = Accept; 客户端 Socket 值赋值给结构体。记录客户相关信息。

新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收客户端消息)，因为是异步函数,WSASend/WSARecv 会马上返回,实际的发送或者接收数据的操作由 WINDOWS 系统去做。然后把 CSessionInfo 对象加入 g_xSessionList 中。向 logsrv 服务器发送用户 Session 信息。打包规则‘%0socket/ip$\0’

在客户 accept 之后，总投递一个 I/O(recv),然后把相应的数据发往 logsrv 服务器。

2) CreateIOCPWorkerThread 函数：

调用 CreateIoCompletionPort 并根据处理器数量，创建一个或多个 ServerWorkerThread 线程。

ServerWorkerThread 线程工作原理：

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus() 马上返回，并从完成端口取得刚完成的 WSASend/WSARecv 的结果。然后接着发出 WSASend/WSARecv，并继续下一次循环阻塞在 GetQueuedCompletionStatus()这里。

a). pSessionInfo 为空或者 dwBytesTransferred =0 ,在客户端 close socket，发相应数据包(异常)到 logsrv 服务器(X 命令-数据包)，关闭客户端套按字。

b). while (pSessionInfo->HasCompletionPacket()) 如果数据验证正确，就转发数据包(A 命令-数据包) logsrv 服务器。

c). if (pSessionInfo->Recv() 继续投递 I/O 操作。

总结：

我们不停地发出异步的 WSASend/WSARecv IO 操作，具体的 IO 处理过程由 WINDOWS 系统完成，WINDOWS 系统完成实际的 IO 处理后，把结果送到完成端口上（如果有多个 IO 都完成了，那么就在完成端口那里排成一个队列）。我们在另外一个线程里从完成端口不断地取出 IO 操作结果，然后根据需要再发出 WSASend/WSARecv IO 操作。

IDM_STOPSERVICE: **停止服务(IOCP 模型 Server 响应客户端请求)**

Close -> OnCommand(IDM_STOPSERVICE, 0L); ->g_fTerminated = TRUE; 线程退出。

```
if (g_hAcceptThread != INVALID_HANDLE_VALUE)
{
    TerminateThread(g_hAcceptThread, 0);
    WaitForSingleObject(g_hAcceptThread, INFINITE); //IOCP 的 Accept 线程
    CloseHandle(g_hAcceptThread);
    g_hAcceptThread = INVALID_HANDLE_VALUE;
}

if (g_hMsgThread != INVALID_HANDLE_VALUE)
{
    TerminateThread(g_hMsgThread, 0);  //窗口例程网络事件回调线程
    WaitForSingleObject(g_hMsgThread, INFINITE);
    CloseHandle(g_hMsgThread);
    g_hMsgThread = INVALID_HANDLE_VALUE;
}
ClearSocket(g_ssock);
ClearSocket(g_csock);
CloseHandle(g_hIOCP);
```

总结：

LoginGate（登录网关服务器），接受客户端连接，并且把用户 ID，密码直接发送到

LoginSvr 服务器中，由 LoginSrv 服务器验证之后，发送数据包返回给客户端。LoginGate 之间是通过定时器，定时发送“心跳”数据。验证服务器存活的。客户端与服务器端的数据在传输中，是进行过加密的。

向 loginSrv 发送‘%A’+Msg+‘$0’消息： 转发客户端消息。
‘%X’+Msg+‘$0’消息： 发送用户连接消息，增加到用户列表。
‘%O’+Msg+‘$0’消息： 发送用户上线消息。

主要流程：

服务启动后，LoginGate 启动了 AcceptThread, 和 ServerWorkerThread 线程，AcceptThread 线程接收客户端连接，并把 session 信息发送给 loginSrv 服务器，ServerWorkerThread 线程从完成端口取得刚完成的 WSASend/WSARecv 的结果后，把客户端数据转发给 loginSrv 服务器。 服务启动时，WSAAsyncSelect 模型连接到 loginSrv 服务器中。一旦连接成功，就启动 ThreadFuncForMsg 线程，该线程从 g_xMsgQueue(FD_READ 事件读到的 loginSrv 服务器发来的数据)中取出 loginSrv 服务器处理过的数据。投递 I/O，利用 IOCP 模型，发送到客户端。

ServerWorkerThread 转发客户端数据 -> WSAAsyncSelect 的 Read 读 loginSrv 处理后返回的数据-> ThreadFuncForMsg 线程，投递 WSASend 消息，由 Windows 处理（IOCP），发送数据给客户端。

LoginSvr 服务器

g_gcSock Local 5500 端口

1. 首先从 LoginSvr.cpp WinMain 分析：
   1) CheckAvailableIOCP ： 检查是不是 NT，2000 的系统（IOCP）
   2) InitInstance: 初始化界面，加载 WSAStartup
      GetDBManager()->Init(InsertLogMsg, "Mir2_Account", "sa", "prg");
      数据库管理类，做底层数据库操作。

   3) MainWndProc 窗口回调函数 OnCommand:
      IDM_STARTSERVICE:
      创建 LoadAccountRecords 线程
      a). UPDATE TBL_ACCOUNT 重置帐户验证状态。
      b). 读服务器列表（TBL_SERVERINFO, selGate 服务器），加入 g_xGameServerList
      遍历 xGameServerList 列表，把服务器信息加入到一个字符数组 g_szServerList 中。
      c). 启动 InitServerThreadForMsg 线程。
      d). 调用 InitServerSocket 函数创建两个线程：
      AcceptThread 线程：
      ServerWorkerThread 线程：

调用 InitServerSocket 函数创建两个线程：

1) AcceptThread 线程：

Accept 之后生成一个 CGateInfo 对象，CGateInfo->sock = Accept; 客户端 Socket 值赋值给结构体。记录客户相关信息。新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收客户端消息)，因为是异步函数，WSASend/WSARecv 会马上返回，实际的发送或者接收数据的操作由 WINDOWS 系统去做。然后把 CGateInfo 对象加入 g_xGateList 中。在客户 accept 之后，投递一个 I/O(recv)。

分析一下 g_xGateList 发现，每个 CGateInfo 里有 sock; xUserInfoList, g_SendToGateQ，该网关的相关信息依次（网关对应的 sock, 用户列列信息，消息队列），可以为多个 LoginGate 登录网关服务。

2) ServerWorkerThread 线程：

ServerWorkerThread 线程工作原理：

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus()马上返回,并从完成端口取得刚完成的 WSASend/WSARecv 的结果。然后接着发出 WSASend/WSARecv，并继续下一次循环阻塞在 GetQueuedCompletionStatus()这里。

a).if (g_fTerminated) 线程结束前：循环遍历 g_xGateList，取出 pGateInfo 关闭套接字，并删除节点。dwBytesTransferred =0 ,关闭**该服务器**套接字。

b).while (pGateInfo->HasCompletionPacket()) 验证消息格式。

case '-': 发送心跳数据包到每个 LoginGate 服务器。

case 'A': 处理每个 LoginGat 服务器转发的客户端的消息增加到各自网关(CGateInfo)g_SendToGateQ 队列中，然后 ThreadFuncForMsg 线程进行验证后再发送消息到各个 LoginGate 服务器。
pGateInfo->ReceiveSendUser(&szTmp[2]);

case 'O': 处理每个网关 Accept 客户端后增加 pUserInfo 用户信息到各自网关的 xUserInfoList 列表中。
pGateInfo->ReceiveOpenUser(&szTmp[2]);

case 'X': 处理每个网关收到客户端 Socket 关闭之后发送过来的消息。设置该网关 socket 相应状态。
pGateInfo->ReceiveCloseUser(&szTmp[2]);

case 'S': GameSvr 服务器发送的消息，更新 TBL_ACCOUNT，验证字段，说明用户已下线，下次登录必须先到 LoginSvr 服务器再次验证。
pGateInfo->ReceiveServerMsg(&szTmp[2]);

case 'M': GameSvr 服务器发送的消息,创建一个用户的消息，把用户 ID，密码,名字插入 TBL_ACCOUNT 表中插入成功返回 SM_NEWID_SUCCESS,

否则 SM_NEWID_FAIL，把在信息前加#，信息后加！ 不做 TBL_ACCOUNTADD 表的添加，只增加 TBL_ACCOUNT 表信息。

'A'：是 LoginGate 服务器转发客户端消息到 g_xMsgQueue 队列，由 ThreadFuncForMsg 线程处理后，转发到各个 loginGate 服务器

继续投递 I/O 操作。

启动 InitServerThreadForMsg 创建 ThreadFuncForMsg 线程。

收到 loginGate 服务器发送过来的消息之后，ServerWorkerThread 经过数据包分析之后（case 'A'），把客户端的消息，写入 g_SendToGateQ 队列中，然后在本线程中再进行处理。

遍历 g_SendToGateQ 队列中数据，验证数据包是否正确（#!字符）根据 DefaultMsg.wIdent 标志

case CM_IDPASSWORD: 处理登陆业务

遍历 xUserInfoList 用户列表信息，到数据库表 TBL_ACCOUNT 中找相应信息，如果失败发送(SM_ID_NOTFOUND, SM_PASSWD_FAIL)消息，否则发送 SM_PASSOK_SELECTSERVER+ g_szServerList（SelGate 服务器列表消息）

SelGate 服务器列表消息（对应 TBL_SERVERINFO 数据库表中数据），供用户选择登录的 SelGate 服务器。

CM_SELECTSERVER: 选择服务器(SelGate)

遍历 xUserInfoList 用户列表信息，根据 socket,找到用户密钥，消息解密后，遍历 g_xGameServerList 列表，把用户选择的 SelGate 服务器转化为 IP 地址，发送至 LoginGate 服务器，再转发至客户端。设置该用户 SelServer 的标志状态。从该网关的 xUserInfoList 用户列表中删除该用户。

CM_ADDNEWUSER: 新注册用户

判断用户名是否已存在，失败发送 SM_NEWID_FAIL 消息，成功，写插入表数据，并发送 SM_NEWID_SUCCESS 消息到 LoginGate 服务器，转发至客户端。

IDM_STOPSERVICE: **停止服务(IOCP 模型 Server 响应客户端请求)**

Close -> OnCommand(IDM_STOPSERVICE, 0L); ->g_fTerminated = TRUE; 三个线程退出。

主要流程：

服务启动后，LoginSvr 启动了 AcceptThread,和 ServerWorkerThread 线程，AcceptThread 线程接收 loginGate，GameSvr 服务器连接，加入 g_xGateList 网关列表中，ServerWorkerThread 线程从完成端口取得刚完成的 WSASend/WSARecv 的结果后，进行分析处理两个服务器发送来的消息。 服务启动同时，启动 ThreadFuncForMsg 线程，该线程从 g_xMsgQueue(iocp 读到的 loginGate 服务器发来的数据)中取出数据，处理数据。投递 I/O,利用 IOCP 模型，发送到 loginGate 服务器。

SelGate 服务器

注：客户端从 LoginSvr 服务器得到 SelGate 服务器 IP 之后，连接 SelGate 服务器，进行角色创建，删除，选择操作，然后发送数据到 DBSrv 服务器。

g_ssock Local 7100 客户端登陆端口

g_csock Remote 5100 发送到 DBSrv 服务器上的套接字

1. 首先从 SelGate.cpp WinMain 分析：
   1) CheckAvailableIOCP ： 检查是不是 NT，2000 的系统（IOCP）
   2) InitInstance: 初始化界面，加载 WSAStartup
   3) MainWndProc 窗口回调函数.
2. MainWndProc.CPP 中分析回调函数 MainWndProc

```
switch (nMsg)
 {
     case _IDM_CLIENTSOCK_MSG:
     case WM_COMMAND:
     case WM_CLOSE:
```

1）_IDM_CLIENTSOCK_MSG 消息：

处理与 SelGate 回调通讯事件。

调用：OnClientSockMsg,该函数是一个回调函数：

当启动服务之后，ConnectToServer 函数将(_IDM_CLIENTSOCK_MSG 消息 FD_CONNECT|FD_READ|FD_CLOSE)传入 WSAAsyncSelect 函数。在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知。函数 OnClientSockMsg,主要完成与 DBSrv 服务器之间的通信（心跳，转发客户端数据包等）

```
switch (WSAGETSELECTEVENT(lParam))
   {
       case FD_CONNECT:
       case FD_CLOSE:
       case FD_READ:
```

FD_CONNECT：(重新连接情况)

A. CheckSocketError 返回正常时：

a). ConnectToServer 函数首先在服务启动的时候执行一次。回调 FD_CONNECT

b). 连接 DBSrv 时,开启 ThreadFuncForMsg 线程，把从客户端发送的数据(g_xMsgQueue, FD_READ 事件读到的 DBSrv 服务器发来的数据)投递 I/O，利用 IOCP 模型，发送到客户端。SleepEx 挂起线程,至到一个 I/O 完成回调函数被调用。 一个异步过程调用排队到此线程。

ThreadFuncForMsg 线程检测(从 DBSrv 收到的 g_xMsgQueue 数据包-心跳，处理包)。i/o 投递，利用 IOCP 发送给客户端。

```
if (nSocket = AnsiStrToVal(pszFirst + 1)) //得到 socket
WSASend((SOCKET)nSocket, &Buf, 1, &dwSendBytes, 0, NULL, NULL);
```

c). 终止定时器_ID_TIMER_CONNECTSERVER
KillTimer(g_hMainWnd, _ID_TIMER_CONNECTSERVER);
d). 设置_ID_TIMER_KEEPALIVE 定时器 （心跳数据包）
SetTimer(g_hMainWnd, _ID_TIMER_KEEPALIVE
调用定时器回调函数 OnTimerProc: 定时发关心跳数据包到 DBSrv 服务器。SendExToServer(PACKET_KEEPALIVE);

B. 如果 socket 断开，设置_ID_TIMER_CONNECTSERVER 定时器
ConnectToServer 尝试重新连接服务器。
_ID_TIMER_CONNECTSERVER, (TIMERPROC)OnTimerProc);

FD_CLOSE:
断开 SOCKET 连接， OnCommand(IDM_STOPSERVICE, 0); 回调函数处理 IDM_STOPSERVICE。

case FD_READ:
接收 DBSrv 服务器发送的数据包(心跳,登陆验证，selCur 服务器地址)，把数据加入缓冲区(g_xMsgQueue)中。

WM_COMMAND:
IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)
IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)
WM_CLOSE:
IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)
WSACleanup();
PostQuitMessage(0); //WM_DESTROY 消息

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

InitServerSocket：函数：

1) AcceptThread 线程：
Accept 之后生成一个 CSessionInfo 对象，pNewUserInfo->sock = Accept; 客户端 Socket 值赋值给结构体。记录客户相关信息。
新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收客户端消息)，因为是异步函数,WSASend/WSARecv 会马上返回,实际的发送或者接收数据的操作由 WINDOWS 系统去做。然后把 CSessionInfo 对象加入 g_xSessionList 中。向 DBsrv 服务器发送用户 Session 信息。打包规则‘%0socket/ip$\0’
在客户 accept 之后，总投递一个 I/O(recv),然后把相应的数据发往 DBSrv 服务器。

2) CreateIOCPWorkerThread 函数：

调用 CreateIoCompletionPort 并根据处理器数量，创建一个或多个 ServerWorkerThread 线程。

ServerWorkerThread 线程工作原理：

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus() 马上返回，并从完成端口取得刚完成的 WSASend/WSARecv 的结果。然后接着发出 WSASend/WSARecv，并继续下一次循环阻塞在 GetQueuedCompletionStatus()这里。

a). pSessionInfo 为空或者 dwBytesTransferred =0 ,在客户端 close socket，发相应数据包(异常）到 DBSrv 服务器(X 命令-数据包)，关闭客户端套按字。

b). while (pSessionInfo->HasCompletionPacket()) 如果数据验证正确，就转发数据包(A 命令-数据包) DBSrv 服务器。

c). if (pSessionInfo->Recv() 继续投递 I/O 操作。

总结：

我们不停地发出异步的 WSASend/WSARecv IO 操作，具体的 IO 处理过程由 WINDOWS 系统完成，WINDOWS 系统完成实际的 IO 处理后，把结果送到完成端口上（如果有多个 IO 都完成了，那么就在完成端口那里排成一个队列)。我们在另外一个线程里从完成端口不断地取出 IO 操作结果，然后根据需要再发出 WSASend/WSARecv IO 操作。

IDM_STOPSERVICE: **停止服务(IOCP 模型 Server 响应客户端请求)**

Close -> OnCommand(IDM_STOPSERVICE, 0L); ->g_fTerminated = TRUE; 线程退出。

```
ClearSocket(g_ssock);
ClearSocket(g_csock);
CloseHandle(g_hIOCP);
```

总结：SelGate（角色处理服务器)，接受客户端连接，并且把用户数据包(角色处理)发送到 DBSrv 服务器中，由 DBSrv 服务器处理之后，发送数据包返回给客户端。SelGate 之间是通过定时器，定时发送“心跳”数据。验证服务器存活的。客户端与服务器端的数据在传输中，是进行过加密的。

向 DBSrv 发送 ‘%A’+Msg+‘$0’消息： 转发客户端消息。

‘%X’+Msg+‘$0’消息： 发送用户连接消息，增加到用户列表。

‘%O’+Msg+‘$0’消息： 发送用户上线消息。

主要流程：

服务启动后,SelGate 启动了 AcceptThread,和 ServerWorkerThread 线程,AcceptThread 线程接收客户端连接，并把 session 信息发送给 DBSrv 服务器，ServerWorkerThread 线程从完成端口取得刚完成的 WSASend/WSARecv 的结果后，把客户端数据转发给 DBSrv 服务器。服务启动时，WSAAsyncSelect 模型连接到 DBSrv 服务器中。一旦连接成功，就启动 ThreadFuncForMsg 线程，该线程从 g_xMsgQueue(FD_READ 事件读到的 DBSrv 服务器发来的数据)中取出 DBSrv 服务器处理过的数据。投递 I/O，利用 IOCP 模型，发送到客户端。

ServerWorkerThread 转发客户端数据 -> WSAAsyncSelect 的 Read 读 DBSrv 处理后返回

的数据-> ThreadFuncForMsg 线程，投递 WSASend 消息，由 Windows 处理（IOCP），发送数据给客户端。

DBSrv 服务器

g_gcSock Local　　5500 端口

1. 首先从 DBSvr.cpp　WinMain 分析：

1) CheckAvailableIOCP ：检查是不是 NT，2000 的系统（IOCP）

2) InitInstance: 初始化界面，加载 WSAStartup
GetDBManager()->Init(InsertLogMsg, "Mir2_Account", "sa", "prg");
数据库管理类，做底层数据库操作。

3) MainWndProc 窗口回调函数 OnCommand:
case _IDM_GATECOMMSOCK_MSG:
WM_COMMAND:
WM_CLOSE:

处理与 DBSvr 回调通讯事件。OnGateCommSockMsg,该函数是一个回调函数：
当启动服务之后，　InitGateCommSocket 函数将(_IDM_CLIENTSOCK_MSG FD_CONNECT|FD_READ|FD_CLOSE)传入 WSAAsyncSelect 函数。在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知。函数 OnClientSockMsg,主要处理 SelGate 服务器发来的消息：

```
switch (WSAGETSELECTEVENT(lParam))
   {
       case FD_ACCEPT:
       case FD_CLOSE:
       case FD_READ:
```

FD_ACCEPT：(接受 SelGate 服务器连接情况)
Accept 之后生成一个 CGateInfo 对象，pGateInfo->sock = Accept; SelGate 服务器 Socket 值赋值给结构体。记录 SelGate 服务器相关信息。然后把 pGateInfo 对象加入 g_xGateInfoList 中。利用 WSAAsyncSelect 在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知(回调本函数处理新 socket 网络事件)

FD_CLOSE:
nNumOfCurrGateSession 减一。

case FD_READ:
接收 SelGate 服务器发送的数据，处理，并发送相应数据包。
'-': 发送心跳数据包到每个 SelGate 服务器。
'A': 处理每个 SelGate 服务器转发的客户端的消息增加到增加到各自网关(CGateInfo)m_GateQ 队列中，然后 ProcessGateMsg 线程进行验证后再发送消息到各个 SelGate 服务器。

pGateInfo->ReceiveSendUser(&szTmp[2]);

'O': 处理每个网关 Accept 客户端后增加 pUserInfo 用户信息到各自网关的 xUserInfoList 列表中。

pGateInfo->ReceiveOpenUser(&szTmp[2]);

case 'X': 处理每个网关收到客户端 Socket 关闭之后发送过来的消息。设置该网关 socket 相应状态。

pGateInfo->ReceiveCloseUser(&szTmp[2]);

WM_COMMAND:

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

WM_CLOSE:

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

WSACleanup();

PostQuitMessage(0); //WM_DESTROY 消息

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应 GameSvr 服务器请求)

InitServerSocket：函数：

1) LoadCharacterRecords

加载数据库表 TBL_GAMEGATEINFO 中 GameGate 服务器列表信息。

2) InitServerSocket（创建 IOCP 模型，处理 GameSvr 服务器发送的消息）

a) AcceptThread 线程：

Accept 之后生成一个 CServerInfo 对象，pServerInfo->m_sock= Accept; GameSvr 服务器 Socket 值赋值给结构体。记录 GameSvr 服务器相关信息。

新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收 GameSvr 服务器消息)，因为是异步函数，WSASend/WSARecv 会马上返回，实际的发送或者接收数据的操作由 WINDOWS 系统去做。然后把 pServerInfo 对象加入 g_xServerList 中。可以为多个 GameSvr 服务器服务。

Setsockopt 设置 SO_SNDBUF 缓冲区大小。WSARecv 投递一个 I/O 操作。

b) CreateIOCPWorkerThread 函数：

调用 CreateIoCompletionPort 并根据处理器数量，创建一个或多个 ServerWorkerThread 线程。

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus() 马上返回，并从完成端口取得刚完成的 WSASend/WSARecv 的结果。

dwBytesTransferred =0 ,关闭 GameSvr 服务器套按字,nNumOfCurrSession 计数减 1。拷贝 pServerInfo->RemainBuff, buffer 到 szBuff 字符数组中。验证数据包是否正确（#!字符）。

生成一个 _TSENDBUFF 结构， lpSendUserData->pServerInfo = pServerInfo;

根据 DefMsg.wIdent 标志处理。

DB_MAKEITEMRCD2：

为 lpSendUserData->lpbtAddDat 分配 sizeof(_TMAKEITEMRCD)个空间，从 pszDivide + DEFBLOCKSIZE 处取出_TMAKEITEMRCD 长度个数据，解密，并放入 lpbtAddData 缓冲。

DB_MAKEITEMRCD:

lpSendUserData->lpbtAddData 分配 sizeof(_TMAKEITEMRCD) 空间，从 pszDivide + DEFBLOCKSIZE+72 取出_TMAKEITEMRCD 数据，解密，并放入 lpbtAddData 缓冲。

DB_SAVEHUMANRCD:

lpSendUserData->lpbtAddData 分配 sizeof(_THUMANRCD) 空间，从 pszDivide + DEFBLOCKSIZE+72 取出_ THUMANRCD 数据，解密，并放入 lpbtAddData 缓冲。把剩下的拷贝到 lpbtAddData2 中。

```
if (lpSendUserData->DefMsg.wIdent>= 100 && <= 200)
   g_DBMsgQ.PushQ((BYTE *)lpSendUserData) //GameGate 服务器消息
else
   g_ServerMsgQ.PushQ((BYTE *)lpSendUserData); //SelGate 服务器消息
```

if (WSARecv(pServerInfo->m_sock, &(pServerInfo->DataBuf), 继续投递 I/O 操作。

3）InitGateCommSocket

创建 AsyncSelect 模型 socket，处理 SelGate 发来的以下消息。

4）创建 ProcessDBMsg 线程处理 GameSvr 服务器发送的消息，并把处理结果发送给 GameSrv 服务器。

case DB_LOADHUMANRCD

GetLoadHumanRcd 函数调用如下函数：
GetHumanRcd 得到一个玩家的详细信息（工作,性别,等级,金子,头发等）。
GetHumanItemRcd 得到玩家人物的物品。
GetHumanMagicRcd 得到玩家魔法（等级等）。
GetHumanGenItemRcd 得到玩家普通信息（前缀名等）
GetHorseRcd 得到玩家马匹信息。
把相应信息发送给 GameSrv 服务器。

case DB_SAVEHUMANRCD

SaveHumanRcd 保存一个玩家的详细信息
SaveHumanMagicRcd 保存玩家魔法
SaveGenItemRcd 保存玩家普通信息

case DB_MAKEITEMRCD

MakeNewItem 函数：

插入人物物品表数据，然后把_TDEFAULTMESSAGE 和_TUSERITEMRCD 数据打包，发送给 GameSrv 服务器。

case DB_MAKEITEMRCD2

同上。

5）创建 ProcessGateMsg 线程 处理 SelGate 服务器发送的消息。

case CM_QUERYCHR: //查询角色 (返回职业，性别，名字信息)

返回人物表中（职业，性别，名称）

case CM_NEWCHR: //创建新角色

查询当前用户角色，如果已经存在，返回，否则插入数据。

case CM_DELCHR: //删除角色

删除一个用户的信息(除帐号外)。

case CM_SELCHR: //选择角色开始游戏

从 g_xGameServerList 取一个 GameGateIP 发送 SelGate 服务器,转发至客户端。循环遍历 g_xGameServerList,找出 GameGate 在线人数最少的服务器。

IDM_STOPSERVICE: **停止服务(IOCP 模型 Server 响应客户端请求)**

Close -> OnCommand(IDM_STOPSERVICE, 0L); ->g_fTerminated = TRUE; 线程退出。

```
ClearSocket(g_ssock);
WSACleanup();
CoUninitialize();
PostQuitMessage(0);
```

总结：DBSvr（数据处理服务器），接受 GameGate 和 SelGate 服务器连接，并处理它们发送来的消息。

TBL_ACCOUNT ---帐号
TBL_ACCOUNTADD --帐号附加信息
TBL_CHARACTER --人物
TBL_CHARACTER_ITEM --人物物品
TBL_CHARACTER_MAGIC --人物学的魔法
TBL_GAMEGATEINFO --登陆文件信息
TBL_GUARD --卫士信息（比如大刀 弓箭手等）
TBL_HORSE --万众注目的骑马
TBL_MAGIC --魔法技能同
magic.db
TBL_MAPINFO --同 mapinfo TBL_MERCHANT --npc 刷新信息
TBL_MONGEN --怪物刷新信息 TBL_MONSTER --怪物数据
monster.db
TBL_MOVEMAPEVENT --按字面说，是地图移动事件

TBL_NPC --npc 同 merchant_def
TBL_STARTPOINT --复活点
TBL_STDITEM --物品数据

GameGate 服务器

1. 首先从 GameGate.cpp WinMain 分析：
   1) CheckAvailableIOCP : 检查是不是 NT，2000 的系统（IOCP）
   2) InitInstance: 初始化界面，加载 WSAStartup
   3) MainWndProc 窗口回调函数.
2. MainWndProc.CPP 中分析回调函数 MainWndProc

```
switch (nMsg)
 {
     case _IDM_CLIENTSOCK_MSG:
     case WM_COMMAND:
     case WM_CLOSE:
```

g_ssock Local 7200 游戏网关登陆端口
g_csock Remote 5000 发送到 GameSrv 服务器上的套接字

1）_IDM_CLIENTSOCK_MSG 消息：处理与 GameSrv 回调通讯事件。

调用：OnClientSockMsg,该函数是一个回调函数：

当启动服务之后，ConnectToServer 函数将(_IDM_CLIENTSOCK_MSG 消息 FD_CONNECT|FD_READ|FD_CLOSE)传入 WSAAsyncSelect 函数。在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知。函数 OnClientSockMsg,主要完成与 GameSrv 服务器之间的通信（心跳，转发客户端数据包等）

```
switch (WSAGETSELECTEVENT(lParam))
   {
       case FD_CONNECT:
       case FD_CLOSE:
       case FD_READ:
```

FD_CONNECT：(连接情况)

A. CheckSocketError 返回正常时：

a). ConnectToServer 函数首先在服务启动的时候执行一次。回调 OnClientSockMsg

b). InitServerThreadForMsg,启动 ThreadFuncForMsg 线程。

把从客户端发送的数据(g_SendToServerQ,Iocp 从客户端读到的数据)处理后， 投递 I/O，利用 IOCP 模型，发送到 GameSrv 服务器。

遍历 g_UserInfoArray (GameSrv 收到的数据包-心跳，处理包 pSessionInfo->SendBuffer)，（重叠 I/O 模型收到到 GameSrv 数据包），pSessionInfo->nSendBufferLen 根据长度判断， i/o 投递，利用 IOCP 发送给客户端。

SleepEx 挂起线程。至到一个 I/O 完成回调函数被调用。 一个

异步过程调用排队到此线程。

c). 终止定时器_ID_TIMER_CONNECTSERVER
KillTimer(g_hMainWnd, _ID_TIMER_CONNECTSERVER);
d). 设置_ID_TIMER_KEEPALIVE 定时器 （心跳数据包）
SetTimer(g_hMainWnd, _ID_TIMER_KEEPALIVE
调用定时器回调函数 OnTimerProc: 定时发关心跳数据包到 logsrv 服务器。SendExToServer(PACKET_KEEPALIVE);

e). 创建 WSACreateEvent，启动 ClientWorkerThread 线程（重叠 I/O 模型）。ClientWorkerThread 执行分析：
ClientOverlapped.Overlapped.hEvent = g_ClientIoEvent;// 设置事件，DataBuf. Len, DataBuf. Buf,投递 WSARecv 操作，WSAWaitForMultipleEvents->WSAWaitForMultipleEvents->WSAResetEvent->WSAGetOverlappedResult-> 一直到 ProcReceiveBuffer 函数。
1. 接收_TMSGHEADER 信息。
2. if (lpMsgHeader->nCode == 0xAA55AA55) 数据验证
3. switch (lpMsgHeader->wIdent)
GM_CHECKSERVER: 收到 GameSrv 心跳数据。
GM_SERVERUSERINDEX：
收到索引值(GameSrv 的 g_xUserInfoArr)的下标。
GM_RECEIVE_OK： 回复收到数据。
GM_DATA： 收到 GameSrv 处理后的数据包。
ProcMakeSocketStr 把 GameSrv 发回的数据包，加上包头等信息，放入 pSessionInfo->SendBuffer 中，设置其长度，以便 ThreadFuncForMsg 线程获取数据，发送到客户端。

B. 如果 socket 断开，设置_ID_TIMER_CONNECTSERVER 定时器
ConnectToServer 尝试重新连接服务器。
_ID_TIMER_CONNECTSERVER, (TIMERPROC)OnTimerProc);

FD_CLOSE:
断开与 logsrv 服务器 SOCKET 连接，OnCommand(IDM_STOPSERVICE, 0); 回调函数处理 IDM_STOPSERVICE。

FD_READ:
接收 logsrv 服务器发送的数据包（心跳,登陆验证，selCur 服务器地址），把数据加入缓冲区(g_xMsgQueue)中。

2）WM_COMMAND:
IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)
IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)
3）WM_CLOSE:
IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)
WSACleanup();

PostQuitMessage(0); //WM_DESTROY 消息

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

InitServerSocket：函数：

1) AcceptThread 线程：

Accept 之后，从 g_UserInfoArray(默认 5000)中得到一个空闲的 CSessionInfo 对象，pNewSessionInfo->sock = Accept; 客户端 Socket 值赋值给结构体。记录客户相关信息。

新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收客户端消息)，因为是异步函数，WSASend/WSARecv 会马上返回，实际的发送或者接收数据的操作由 WINDOWS 系统去做。向 GameSrv 服务器发送一个 GM_OPEN(开始)消息。消息格式_TMSGHEADER。

在客户 accept 之后，总投递一个 I/O(recv),然后把相应的数据发往 GameSrv 服务器。

2) CreateIOCPWorkerThread 函数：

调用 CreateIoCompletionPort 并根据处理器数量，创建一个或多个 ServerWorkerThread 线程。

ServerWorkerThread 线程工作原理：

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus() 马上返回，并从完成端口取得刚完成的 WSASend/WSARecv 的结果。然后接着发出 WSASend/WSARecv，并继续下一次循环阻塞在 GetQueuedCompletionStatus()这里。

a). dwBytesTransferred =0 ,发相应数据包(GM_CLOSE）到 GameSrv 服务器，消息格式 _TMSGHEADER 。调用 CloseSession 关闭客户端 socket，g_UserInfoArray 对应元素置为空。

b). while (pSessionInfo->HasCompletionPacket()) 如果数据验证正确，就生成_LPTSENDBUFF 对象，g_SendToServerQ.PushQ 压入堆栈中。

c). if （pSessionInfo->Recv() 继续投递 I/O 操作。

总结：

我们不停地发出异步的 WSASend/WSARecv IO 操作，具体的 IO 处理过程由 WINDOWS 系统完成，WINDOWS 系统完成实际的 IO 处理后，把结果送到完成端口上（如果有多个 IO 都完成了，那么就在完成端口那里排成一个队列）。我们在另外一个线程里从完成端口不断地取出 IO 操作结果，然后根据需要再发出 WSASend/WSARecv IO 操作。

IDM_STOPSERVICE: **停止服务(IOCP 模型 Server 响应客户端请求)**

Close -> OnCommand(IDM_STOPSERVICE, 0L); ->g_fTerminated = TRUE; 线程退出。

g_fTerminated = TRUE;

ClearSocket(g_ssock);

ClearSocket(g_csock);

总结：

GameGate（游戏网关服务器），接受客户端连接，并且把用户数据发送到 GameSvr 服务器中，由 GameSvr 服务器处理之后，返回发送数据包。GameSvr 之间是通过定时器，定时发送“心跳”数据。验证服务器存活的。客户端与 GameGate 数据在传输中，是进行过加密的。

向 GameSrv 发送　GM_OPEN 消息：　转发客户端游戏开始消息。

SendSocketMsgS 转发客户端其它消息。

GM_CHECKSERVER：　接收 GameSrv 心跳数据。

GM_SERVERUSERINDEX：　收到索引值。

GM_RECEIVE_OK：　回复收到数据。

GM_DATA：　收到 GameSrv 处理后的数据包。

主要流程：

服务启动后，GameGate 启动了 AcceptThread, 和 ServerWorkerThread 线程，AcceptThread 线程接收客户端连接，并把 GM_OPEN 信息发送给 GameSrv 服务器，ServerWorkerThread 线程从完成端口取得刚完成的 WSASend/WSARecv 的结果后，把客户端数据压入 g_SendToServerQ。 服务启动时，WSAAsyncSelect 模型连接到 GameSrv 服务器中。一旦连接成功，就启动 ThreadFuncForMsg 线程，该线程将客户端发送的数据 g_SendToServerQ 发送到 GameSrv 服务器。同时启动 ClientWorkerThread 线程（重叠 I/O 模型），接收 GameSrv 服务器的数据，并把数据放入 pSessionInfo->SendBuffer 中，设置其长度，以便 ThreadFuncForMsg 线程获取数据，投递 I/O，利用 IOCP 模型，发送到客户端。

ThreadFuncForMsg 转发客户端数据，返回 GameSrv 服务器处理后返回的数据。 -> ClientWorkerThread 读 GameSrv 处理后返回的数据-> ThreadFuncForMsg 线程，投递 WSASend 消息，由 Windows 处理（IOCP），发送数据给客户端。

根据 AcceptThread 线程里的分析发现：g_UserInfoArray 最大容量（_MAX_USER_ARRAY）是 5000 人。

GameSrv 服务器

核心游戏服务器模块：连接到 DBSvr 服务器和 LoginSvr 服务器，并接受 GameGate 的连接。

g_gcSock Local　　5500 端口

1. 首先从 GameSvr.cpp　WinMain 分析：

1) CheckAvailableIOCP ： 检查是不是 NT，2000 的系统（IOCP）

2) InitInstance: 初始化界面，加载 WSAStartup

g_pConnCommon = g_MirDB.CreateConnection("Mir2_Common", "sa", "prg");

g_pConnGame　　= g_MirDB.CreateConnection(szDatabase, "sa", "prg");

数据库管理类，做底层数据库操作。

4) MainWndProc 窗口回调函数 OnCommand:

_IDM_CLIENTSOCK_MSG:

_IDM_LOGSVRSOCK_MSG:

WM_COMMAND: (IDM_STARTSERVICE 和 IDM_STOPSERVICE)

WM_CLOSE:

IDM_STARTSERVICE 消息：

1). CMapInfo* pMapInfo = InitDataInDatabase(); 初始化基本数据。

InitMagicInfo(); //加载魔法列表，总数，循环生成 CMagicInfo。

InitMonsterGenInfo(); //加载怪物列表,总数，循环生成 CMonsterGenInfo。

InitMonRaceInfo(); //怪物种族列表,总数，循环生成 CMonRaceInfo。

InitStdItemSpecial(); //玩家物品列表，总数，循环生成 CStdItemSpecial。

InitStdItemEtcInfo(); //其它列表, 总数，循环生成 CStdItem。

InitMerchantInfo(); //商人列表，总数，循环生成 CMerchantInfo.

InitMoveMapEventInfo();

//移动地图事件列表总数，循环生成 CMoveMapEventInfo.

InitMapInfo(nServerIndex); //地图列表, 总数，循环生成 CMapInfo。

nServerIndex 索引值返回地图名字。

注意：GameSrv 服务器可以由多台做成负载均衡，每个服务器加载不同的地图。做为场景服务器。

2）创建 InitializingServer 线程->ConnectToServer

_IDM_CLIENTSOCK_MSG

连接(DBSrv 数据库服务器) OnClientSockMsg 网络事件回调函数。

_IDM_LOGSVRSOCK_MSG:

连接 LoginSrv 服务器（登录服务器）OnLogSvrSockMsg 网络事件回调函数。

InitializingServer 线程执行流程分析：

a). 加载 searchTable.tbl 生成搜索 Table

b). 遍历所有 pMapInfo， 加载 LoadMap(&pMapInfo[i]);生成 CMirMap,并加入 g_xMirMapList 列表中。

c). InitAdminCommandList();

加载管理员命令列表。AdminCmd.DLL 该工程的 TableString。

加入 g_xAdminCommandList 列表中。

_IDM_LOGSVRSOCK_MSG

回调函数 OnLogSvrSockMsg 完成和 LoginSrv 之间的通信（用户验证等）。CloseAccount 时，向 loginSrv 发送‘S’命令，通知用户下线。下次连接时必须再次登录验证。

_IDM_CLIENTSOCK_MSG 消息:

调用：OnClientSockMsg,该函数是一个回调函数(连接 DBSrv 服务器)：

当启动服务之后， ConnectToServer 函数将(_IDM_CLIENTSOCK_MSG 消息 FD_CONNECT|FD_READ|FD_CLOSE)传入 WSAAsyncSelect 函数。在与 hWnd 窗口句柄对应的窗口例程中以 Windows 消息的形式接收网络事件通知。函数 OnClientSockMsg, 主要完成与 DBSrv 服务器之间的通信（心跳，数据库操作等）

```
switch (WSAGETSELECTEVENT(lParam))
  case FD_CONNECT:
```

case FD_CLOSE:
case FD_READ:

FD_CONNECT：

A. CheckSocketError 返回正常时：

终止定时器_ID_TIMER_CONNECTSERVER

KillTimer(g_hMainWnd, _ID_TIMER_CONNECTSERVER);

创建线程如下：

1. ProcessLogin 线程：（用户登录、退出处理）

遍历 g_xLoginOutUserInfo 列表，处理由 GameGate 发送的 GM_OPEN 消息和 GM_CLOSE 消息。（在接收到消息之后，就把 MsgHdr 信息和 pUserInfo 压入 g_xLoginOutUserInfo 列表中）。

pUserInfo->m_btCurrentMode 处理消息：

USERMODE_LOGOFF:

pUserInfo->CloseUserHuman();

SaveHumanToDB-> SaveHumanToDB-> SendRDBSocket 和 DBSrv 发送 DB_SAVEHUMANRCD 消息，保存用户数据。

并从 g_xLoginOutUserInfo 列表中删除该用户。从地图中删除人物。删除 m_lpTItemRcd，m_lpTGenItemRcd, m_lpTMagicRcd 中用户信息。

调用 CloseAccount 函数，发送‘S’命令到 LoginSrv 中。

USERMODE_LOGIN:

GM_OPEN 之后，客户端发送的登录消息处理->GM_DATA-> DoClientCertification-> LoadPlayer -> LoadHumanFromDB ->

SendRDBSocket 发送 DB_LOADHUMANRCD 请求,返回该玩家的所有数据信息。已经连接 DBSrv 服务器，回调函数 OnClientSockMsg 中事件 FD_READ 调用的 ProcReceiveBuffer 接收用户数据。生成一个 CReadyUserInfo2 数据，并压入 g_xReadyUserInfoList2 列表中。

从 g_xReadyUserInfoList2 中找到该用户，进行比较，验证。一旦找到，调用 LoadPlayer（重载函数），把 pReadyUserInfo 里的用户信息，赋值给 pUserInfo， 并把 m_btCurrentMode 模式设置 USERMODE_PLAYGAME 再从 g_xReadyUserInfoList2 删除该用户信息。

USERMODE_NOTICE：

Break;

总结：

GameSrv 的 ServerWorkerThread 线程处理-> pUserInfo->DoClientCertification 设置用户信息，及 USERMODE_LOGIN 的状态。并且调用 LoadPlayer(CUserInfo* pUserInfo)函数 -> LoadHumanFromDB-> SendRDBSocket 发送 DB_LOADHUMANRCD 请求，返回该玩家的所有数据信息。

用户的验证是由 GameSrv 的线程 ProcessLogin 处理。该线程从 g_xLoginOutUserInfo 列表中取出数据（OpenNewUser 事件数据 GM_OPEN）和 g_xReadyUserInfoList2 列表中的数

据对比，判断用户是否已经登录，一旦登录就调用 LoadPlayer，并把用户数据从 g_xReadyUserInfoList2 列表中删除。

LoadPlayer(CReadyUserInfo2* pReadyUserInfo, CUserInfo* pUserInfo)

设置 m_btCurrentMode 状态为 USERMODE_PLAYGAME; 游戏状态。
加载物品，个人设置，魔法等。
pUserInfo->m_pxPlayerObject->Initialize();
加载用户 X，Y 坐标，方向，地图。

1. AddProcess(this, RM_LOGON, 0, 0, 0, 0, NULL);加入登录消息。
2. m_pMap->AddNewObject 地图中单元格（玩家列表）加入该游戏玩家。OS_MOVINGOBJECT 玩家状态。
3. AddRefMsg(RM_TURN 向周围玩家群发 RM_TURN 消息。以玩家自己为中心，以 24*24 的区域里，向这个区域所属的块里的所有玩家列表发送消息)广播 AddProcess。
4. RecalcAbilitys 设置玩家的能力属性（攻击力（手，衣服），武器力量等）。
   a. 循环处理本游戏玩家的附属物品，把这些物品的力量加到（手，衣服等）的攻击力量里。
   b. RM_CHARSTATUSCHANGED 消息，通知玩家状态改变消息。

5. AddProcess(this, RM_ABILITY, 0, 0, 0, 0, NULL); 等级
   AddProcess(this, RM_SUBABILITY, 0, 0, 0, 0, NULL);
   AddProcess(this, RM_DAYCHANGING, 0, 0, 0, 0, NULL); 校时
   AddProcess(this, RM_SENDUSEITEMS, 0, 0, 0, 0, NULL); 装备
   AddProcess(this, RM_SENDMYMAGIC, 0, 0, 0, 0, NULL); 魔法
   SysMsg(szMsg, 1) 攻击力
   说明：4 处理本玩家的消息。

GameSrv 向 DBSrv 服务器查询用户的所有信息，并把客户端的数据放入 pUserInfo 中，验证由 ProcessLogin 来处理。一旦通过验证，就从验证列表中该玩家，改变玩家状态，LoadPlayer 加载用户资源（地图中加入用户信息，向用户 24*24 区域内的块内玩家发送上线消息 GameSrv 广播新玩家上线（坐标）的消息。向该新玩家发送玩家信息（等级，装备，魔法，攻击力等）。

未通过验证，则玩家游戏状态信息不改变。玩家只能继续进入验证流程。

2. ProcessUserHuman 线程

1) 遍历 UserInfoList 列表，依次调用每个 UserInfo 的 Operate 来处理命令队列中的所有操作; pUserInfo->Operate()调用 m_pxPlayerObject->Operate()调用。
2) 遍历 GateList 列表，依次调用每个 GateInfo 的 xSend 函数将发送缓冲区中的消息发往 GameGate;
3) 线程结束时，遍历 g_xUserInfoList，依次调用 CloseUserHuman 函数（向 DBSrv,loginSrv 发送消息，清除列表数据）。
4) 各个用户 m_pxPlayerObject->Operate()根据分发消息（RM_TURN）向客户端发送 SM_TURN 消息。GameSrv 广播新玩家上线（坐标）的消息。向该新玩家发送玩家信息（等级，装备，魔法，攻击力等）。

5） 遍历网关列表 g_xGateList，每个 pGateInfo 的 m_xSendBuffQ 消息，发送到 GameGate 服务器。m_xSendBuffQ 消息里存放的是 CPlayerObject::Operate()处理后的结果。

总结：

游戏玩家连接 GameGate 成功时，GameGate 向 GameSrv 发送 GM_OPEN 事件，游戏玩家登录操作，由GameGate 生成MsgHdr 数据发送到GameSrv 服务器端。GameSrv 向 DBSrv 服务器查询用户的所有信息，并把客户端的数据放入 pUserInfo 中，验证由 ProcessLogin 来处理。一旦通过验证，就从验证列表中该玩家，改变玩家状态，LoadPlayer 加载用户资源（地图中加入用户信息，向用户 24*24 区域内的块内玩家发送上线消息 GameSrv 广播新玩家上线（坐标）的消息。向该新玩家发送玩家信息（等级，装备，魔法，攻击力等）。未通过验证，则玩家游戏状态信息不改变。玩家只能继续进入验证流程。

3．ProcessMonster 线程：

调用 RegenMonster，创建不同的 CMonsterObject，并且加入到 xMonsterObjList 列表中，调用 SearchViewRange()更新视线范围内目标，根据 g_SearchTable 计算出搜索坐标，转换为相应的地图单元格，遍历所有可移动生物，加入 m_xVisibleObjectList 列表，调用 Operate；

Operate 遍历 m_DelayProcessQ 列表，过滤出 RM_DOOPENHEALTH，RM_STRUCK 和 RM_MAGSTRUCK 三个事件(恢复生命值，攻击，魔法攻击)，并处理。

4．ProcessNPC 线程：

1. CscripterObject.Operate 执行流程

遍历所有的脚本列表 g_xScripterList，执行脚本 CscripterObject.Operate（AI 系统）CscripterObject.Operate 执行流程（比如：走动）。

WalkTo（走动）CScripterObject 是从 CPlayerObject 派生的，所以实际调用：CCharObject::WalkTo

5） WalkNextPos 根据随机值产生，八个方向的坐标位置。

6） WalkXY 怪物走动到一个坐标值中。

CheckDoorEvent 根据 pMapCellInfo->m_sLightNEvent 返回四种状态。

a） 要移动的位置是一扇门 _DOOR_OPEN

b） 不是一扇门 _DOOR_NOT

c） 是一扇门不可以打开返回 _DOOR_MAPMOVE_BACK 或_DOOR_MAPMOVE_FRONT 玩家前/后移动

7） 如果_DOOR_OPEN 则发送 SM_DOOROPEN 消息给玩家

8） m_pMap->CanMove 如果可以移动，则 MoveToMovingObject 从当前点移动到另一点。并发送 AddRefMsg(RM_WALK)给周围玩家。

2. ProcessForUserSaid（和用户对话）执行流程：

ProcessForAdminSaid

遍历管理员命令列表 g_xAdminCommandList 执行每个动作。

IDS_COMMAND_MONGEN ：

GetMonRaceInfo 得到怪物种族。创建民 nMax 个怪物（根据种族）

加入 g_pMonGenInfo[g_nNumOfMonGenInfo].xMonsterObjList.AddNewNode
Map->AddNewObject 在地图中加上该怪物。AddRefMsg(RM_TURN 广播。

IDS_COMMAND_MOVE：怪物随机移动。
IDS_COMMAND_MONRECALL：
CmdCallMakeSlaveMonster 创建民 nMax 个怪物（根据种族）
m_xSlaveObjList.AddNewNode(pMonsterObject)； 在地图中加上该怪物

IDS_COMMAND_GENPOS
在指定的坐标内创建怪物。在地图中加上该怪物
IDS_COMMAND_RESERVED5
生成一个 CscripterObject 对象，pMap->AddNewObject 加入，g_xScripterList 里加入该对象。

3. ProcessMerchants：
1). 遍历 g_pMerchantInfo 结构(根据 nNumOfMurchantInfo 数量)。得到商人类型相关的地图，创建商人对象，设置不同的编号，坐标，头像及所属地图。在该地图中加入该商人，且在 g_xMerchantObjList 商人清单中加入该商人。
2). 遍历 g_xMerchantObjList, SearchViewRange，对每个商人更新视线范围内目标
a). 遍历 m_xVisibleObjectList，设置每个 pVisibleObject->nVisibleFlag = 0;设置状态（删除）。
b). 搜索 VisibleObjectList 列表，(服务器启动时 InitializingServer 加载 searchTable.tbl)，根据坐标，找到相应的地图单元格。然后遍历 pMapCellInfo->m_xpObjectList 列表，判断如果为 OS_MOVINGOBJECT 标志，调用 UpdateVisibleObject 函数，该函数遍历 m_xVisibleObjectList 列表，如果找到该商人对象，则 pVisibleObject->nVisibleFlag = 1;否则判断 pNewVisibleObject 对象，设置 nVisibleFlag 为 2，设置对象为该商人实体，然后加入 m_xVisibleObjectList 列表中。

总结：循环列表，找出地图单元格中的所有玩家，把所有玩家(OS_MOVINGOBJECT)加入到 m_xVisibleObjectList 列表中。
c). 遍历 m_xVisibleObjectList 列表,(pVisibleObject->nVisibleFlag == 0)则删除该 pVisibleObject 对象。
d). RunRace 调用 AddRefMsg 向周围玩家发送 SM_TURN 和 SM_HIT

4. ProcessEvents：
1) g_xEventList 列表处理
CPlayerObject::Operate()->CM_SPELL 消息->DoSpell-> _SKILL_EATTHFIRE
消息：MagMakeFireCross：
m_pMap->GetAllObject(m_nX, m_nY, &ObjectList)
对单元格内玩家，发送 RM_MAGSTRUCK_MINE 消息。

2) g_xHolySeizeList

CPlayerObject::Operate()->CM_SPELL 消息->DoSpell->_SKILL_HOLYSHIELD 消息：MagMakeHolyCurtain

5. AcceptThread 与 ServerWorkerThread（响应 GameGate 服务器）

AcceptThread 线程工作原理：

Accept 之后生成一个 CGateInfo 对象，CGateInfo->sock =Accept;客户端 Socket 值赋值给结构体。记录客户相关信息。新的套接字句柄用 CreateIoCompletionPort 关联到完成端口，然后发出一个异步的 WSASend 或者 WSARecv 调用(pNewUserInfo->Recv();接收客户端消息)，因为是异步函数，WSASend/WSARecv 会马上返回,实际的发送或者接收数据的操作由 WINDOWS 系统去做。然后把 CGateInfo 对象加入 g_xGateList 中。在客户 accept 之后，投递一个 I/O(recv)。

分析一下 g_xGateList 发现，每个 CGateInfo 里有 sock; m_xSendBuffQ, OverlappedEx[2];该网关的相关信息依次（网关对应的 sock, 发消缓冲区），可以为多个 GameGate 登录网关服务。

ServerWorkerThread 线程工作原理：

循环调用 GetQueuedCompletionStatus()函数来得到 IO 操作结果。阻塞函数。当 WINDOWS 系统完成 WSASend 或者 WSArecv 的操作，把结果发到完成端口。GetQueuedCompletionStatus() 马上返回，并从完成端口取得刚完成的 WSASend/WSARecv 的结果。然后接着发出 WSASend/WSARecv，并继续下一次循环阻塞在 GetQueuedCompletionStatus()这里。

if (dwBytesTransferred == 0) 一旦 GameGate 关闭 socket, 遍历 g_xUserInfoList 列表，if (pUserInfo->m_pGateInfo == pGateInfo)，如果时关闭的网关，m_pMap->RemoveObject，从地图上删除该用户信息。从 g_xUserInfoList 中删除用户本身信息。关闭套按字。并从网关 g_xGateList 列表中删除该网关信息。

while (pGateInfo->HasCompletionPacket())数据验证，

if (pMsgHeader->nCode != 0xAA55AA55)数据包头验证。

```
switch ( pMsgHeader->wIdent )
```

GM_OPEN: pGateInfo->OpenNewUser(completionPacket);

OpenNewUser 函数 g_xUserInfoArr 最大容量 10000 人。

g_xLoginOutUserInfo 加入外部用户登陆信息列表中,m_btCurrentMode 节点状态设置为 USERMODE_NOTICE;初始化用户信息，返回 GM_SERVERUSERINDEX 和 GameGate 的索引 ID 消息（_TMSGHEADER 格式）压入 m_xSendBuffQ.PushQ 中，发送给 GameGate，GameGate 设置 pSessionInfo->nServerUserIndex，

GM_CLOSE：

调用 CloseOpenedUser 函数。g_xLoginOutUserInfo 加入外部用户登陆信息列表中，m_btCurrentMode 节点状态设置为 USERMODE_LOGOFF 生成 _TSENDBUFF 消息，消息为 GM_CLOSE。把消息压入 m_xSendBuffQ.PushQ 中。

case GM_CHECKCLIENT:

调用 SendGateCheck 函数。生成_TSENDBUFF 消息,消息为 GM_CHECKSERVER。

把消息压入 m_xSendBuffQ.PushQ 中。

case GM_DATA

从 g_xUserInfoArr 中找到用户信息。判断玩家状态为 USERMODE_PLAYGAME 则调用 pUserInfo->ProcessUserMessage 函数。游戏逻辑处理该用户消息。否则先调用 pUserInfo->DoClientCertification 调用 LoadPlayer 函数 -> LoadHumanFromDB-> SendRDBSocket 发送 DB_LOADHUMANRCD 请求，返回该玩家的所有数据信息。CUserInfo::ProcessUserMessage 是游戏的核心处理函数。

ProcessUserMessage 函数功能：

switch (lpDefMsg->wIdent)

m_pxPlayerObject->AddProcess，把相应操作压入 m_ProcessQ.PushQ 中。

if (pGateInfo->Recv()，继续投递 I/O 操作(IOCP)。

B. 如果 socket 断开，设置_ID_TIMER_CONNECTSERVER 定时器

ConnectToServer 尝试重新连接服务器。

_ID_TIMER_CONNECTSERVER, (TIMERPROC)OnTimerProc);

FD_CLOSE:

断开与 logsrv 服务器 SOCKET 连接，OnCommand(IDM_STOPSERVICE, 0)；回调函数处理 IDM_STOPSERVICE。

FD_READ:

接收 logsrv 服务器发送的数据包（心跳,登陆验证，selCur 服务器地址），把数据加入缓冲区(g_xMsgQueue)中。

2）WM_COMMAND:

IDM_STARTSERVICE: 启动服务(IOCP 模型 Server 响应客户端请求)

启动 AcceptThread 与 ServerWorkerThread 线程

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

3）WM_CLOSE:

IDM_STOPSERVICE: 停止服务(IOCP 模型 Server)

WSACleanup();

PostQuitMessage(0); //WM_DESTROY 消息

GameSrv 服务器启动流程分析：

GameSrv 启动后创建 CConnection 对象连接，点击启动按钮之后，InitDataInDatabase 函数初始化。创建 InitializingServer 线程。加载 searchTable.tbl 文件，生成搜索表，遍历 CMapInfo，加载地图(生成地图单元格，并加入 g_xMirMapList 地图列表)。再调用 InitAdminCommandList，加载管理员 OP 命令列表。连接 DBSrv 和 LoginSrv 服务器。接受 LoginSrv 服务器的用户登录信息。连接到 DBSrv 服务器时，创建 ProcessLogin，ProcessUserHuman，ProcessMonster，ProcessNPC 四个业务处理线程。

然后，再创建 IOCP 模型，接受 GameGate 服务器连接，这里创建两个线程。AcceptThread 和 ServerWorkerThread 线程。

AcceptThread：接受 GameGate 服务器连接。

ServerWorkerThread：接受 GameGate 服务器转发的玩家消息（登录等）。

ProcessLogin：处理用户登录线程。该线程工作时，首先 ServerWorkerThread 线程会向 DBSrv 发送 DB_LOADHUMANRCD 请求，返回该玩家的所有数据，并把数据加入到验证列表，然后 ProcessLogin 处理验证数据。在地图单元格中的玩家列表中加入该玩家，并发送广播消息给附近的玩家。然后返回玩家的所有游戏信息。

ProcessUserHuman：处理由 GameGate 服务器接受的玩家消息，调用 Operate(处理玩的数据加入发送缓冲区)，把缓冲区数据发送给 GameGate 服务器，调用 CloseUserHuman 向 DBSrv 和 LoginSrv 发送玩家下线消息。

ProcessMonster：遍历怪物种族，循环对该种族 xMonsterObjList 所有的怪物，调用 SearchViewRange，

ProcessNPC：遍历脚本 g_xScripterList，执行脚本操作，ProcessMerchants 生成商人对象 CMerchantObject，加入商人列表 g_xMerchantObjList 和地图中。遍历每个商人，执行更新视线范围内目标，调用 RunRace。

游戏服务器数据结构

控件：

CChatEdit 类：

CGameBtn：

CMirButton：

CMirMsgBox

CMirMsgBox3D

CMsgBox

CMsgBoxBtn

CNoticeBox： 消息框

CScrlBar：

CTextButton：

声音：

CAirWave 类：播放 Wave 格式文件的一个类。

CAvi 类： 播放 AVI 格式文件的一个类。

CBMMp3 类： 播放 MP3 格式文件的一个类。

CElec 类： 闪电

CLightFog： 光，雾

CMirSound： 声音 CBMMp3* m_pMp3; CSBuffer*, CSound 变量。

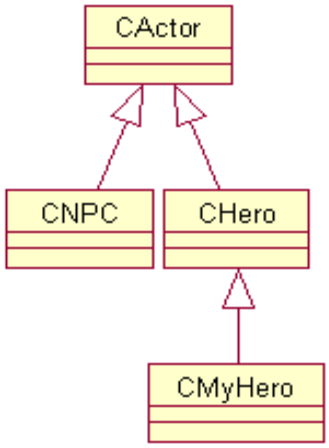
CMist：薄雾

CSound

CElec：闪电。

人物：

Cactor 派生出 Chero 又派生出 CMYHero



CActor 类：

属性说明:

m_dwIdentity： 身份标识，玩家的 m_xActorList 列表内，存放所有玩家，NPC。该字段用来标识每个不同的可移动物体，接收到 GameSrv 服务器消息之后，根据该字段，调用不同 Cactor 的方法。

FEATURE： 结构体定义了 性别，武器，衣服，头发。

m_wPosX， m_wOldPosX： 鼠标坐标，上一次鼠标坐标。

CWHQueue m_xPacketQueue： 消息队列（一部分消息加入该队列中）。

方法说明：

ChangeFeature 改变玩家图象，头发等。

Create： 创建一个 Cactor 对象，初始化地图，收到服务器玩家列表的时候。

Draw 开头的几个方法： 绘制玩家到地图中,绘制玩家名称，HP 值等。

UpdateMotionState 处理动作，状态的改变，并调用 UpdatePacketState 方法。

UpdateMove 玩家移动(进行判断，否则拉回)

UpdatePacketState 遍历 m_xPacketQueue 队列，响应服务器端消息。

其它的一些方法：m_xPacketQueue 消息队列遍历并调用的具体方法。

CHero 类：

属性说明:

FEATUREEX：结构体玩家，骑马，头发衣服颜色。

m_bIsMon： 是否怪物。

方法说明：

CMyHero 类：

属性说明:

CMapHandler* m_pxMap; 所在地图。

CWHQueue m_xPriorPacketQueue; 优先处理的消息队列。

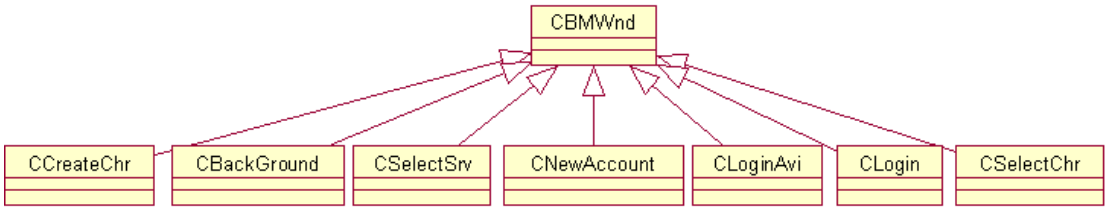
方法说明：

加入鼠标和键盘事件响应。

AdjustMyPostion： 调整位置。

CNPC 类：

CBMWnd 基类及派生类：



CBMWnd 类： Wnd 基类，判断是否获得焦点，失去焦点状态。鼠标位置是否在 Wnd 内部。另外定义了鼠标，键盘事件的虚方法。鼠标点击的时候，设置 point。

CBackGround：派生 CBMWnd 类。加入了 CAvi，m_nRenderState 状态机，背景图片等成员变量。键盘和鼠标事件都是虚函数。重要的方法 Render 和 SetRenderState。根据不同的游戏状态，切换不同的背景，声音。

CCreateChr： 派生 CBMWnd 类。创建角色。

CLogin： 派出 CBMWnd 类,登录处理。调用 g_xClientSocket.OnLogin 方法。

CloginAvi 类：

CNewAccount：创建一个帐号。

CSelectChr：

CSelectChr::OnButtonUp->

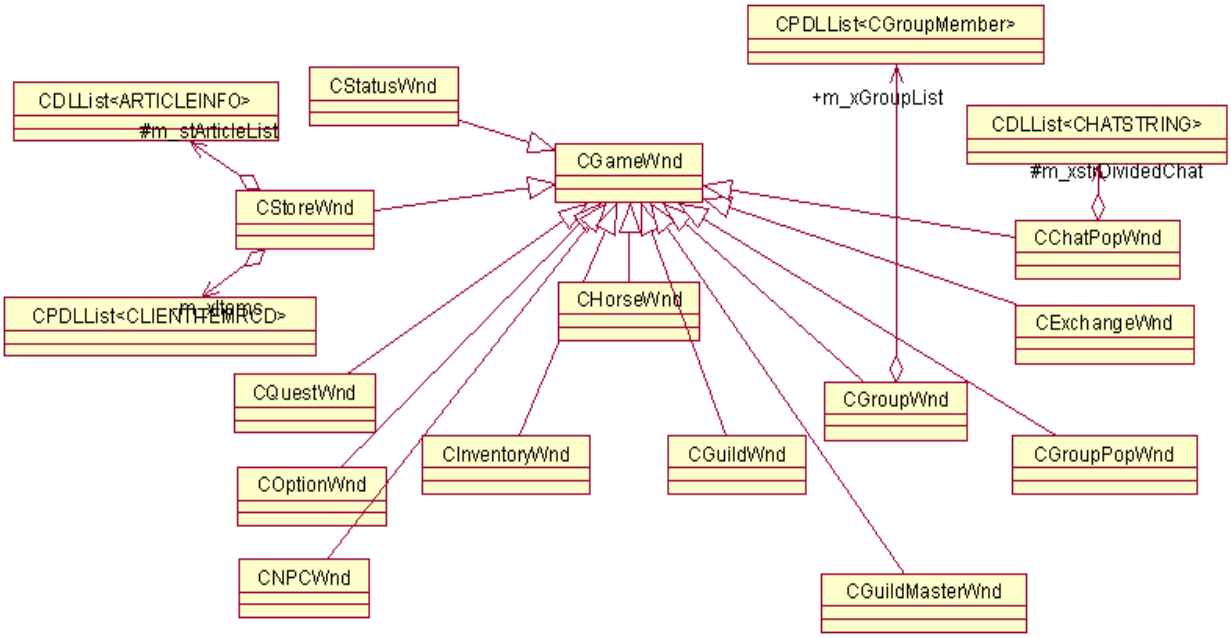
g_xClientSocket.OnSelChar->发送 CM_SELCHR 消息到 SelGate 服务器。

CSelectSrv：选择服务器。g_xClientSocket.OnSelectServer

m_xSelectSrv.OnButtonDown->CselectSrv. OnButtonUp->

g_xClientSocket.OnSelectServer(CM_SELECTSERVER)，得到真正的 IP 地址。调用 OnSocketMessageRecieve 处理返回的 SM_SELECTSERVER_OK 消息。并且断开与 loginSrv 服务器连接。

CGameWnd 基类及派生类：



CChatPopWnd：派生 CGameWnd 类,聊天。
CExchangeWnd：
CGroupPopWnd 类：
CGroupWnd 类：
CGuildMasterWnd 类：
CGuildWnd 类：
CHorseWnd 类：
CInventoryWnd 类：
CNPCWnd 类：
COptionWnd 类：
CQuestWnd 类：
CStatusWnd 类：
CStoreWnd 类：

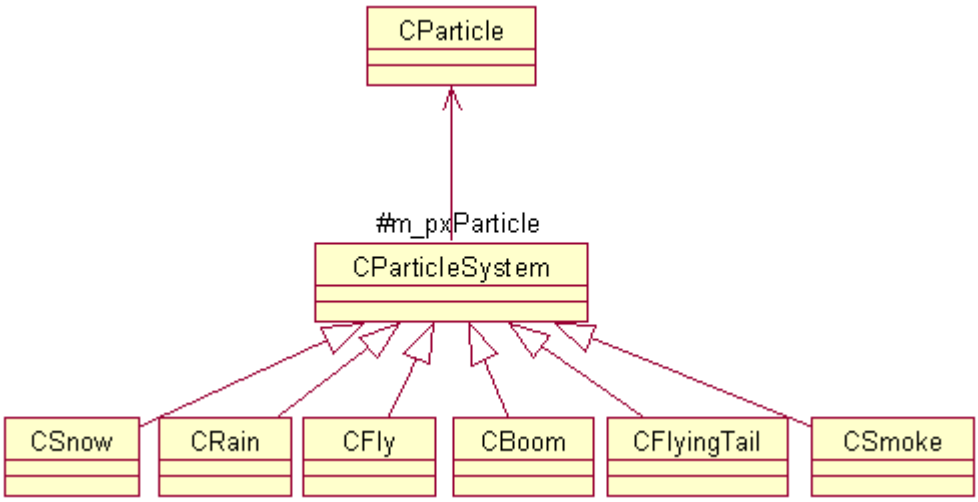
Cinterface 类：
成员变量：

```
CInventoryWnd       m_xInventoryWnd;
CStatusWnd          m_xStatusWnd;
CStoreWnd           m_xStoreWnd;
CExchangeWnd        m_xExchangeWnd;
CGuildMasterWnd     m_xGuildMasterWnd;
CGuildWnd           m_xGuildWnd;
CGroupWnd           m_xGroupWnd;
CGroupPopWnd        m_xGroupPopWnd;
CChatPopWnd         m_xChatPopWnd;
CNPCWnd             m_xNPCWnd;
CQuestWnd           m_xQuestWnd;
COptionWnd          m_xOptionWnd;
```

CHorseWnd m_xHorseWnd;

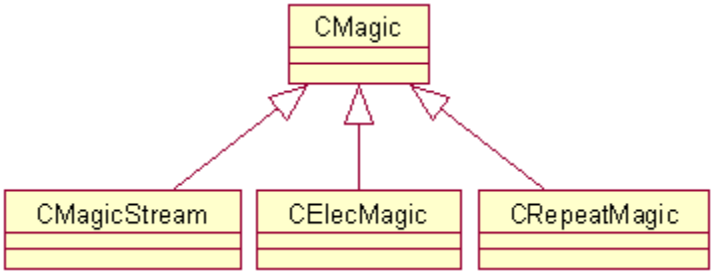
m_xChat: 聊天记录列表
m_xWndOrderList：命令列表，根据不同窗体的 ID，做不同的消息处理。
m_xInterBtn[_MAX_INTER_BTN]：按钮列表

CParticleSystem 基类及派生类（粒子场景）： CParticle 粒子



CParticleSystem 类：场景基类，处理移动，碰撞等。
CBOOM 派生 CParticleSystem 类: 加载 shine0-9.bmp 图片.
CFly 类派生。
CFlyingTail:
CRain 类:
CSmoke 类:
CSnow 类:

CMagic 魔法：



CElecMagic 派生 CMagic 魔法
CmagicStream 类:
CRepeatMagic 类:

主要消息处理：

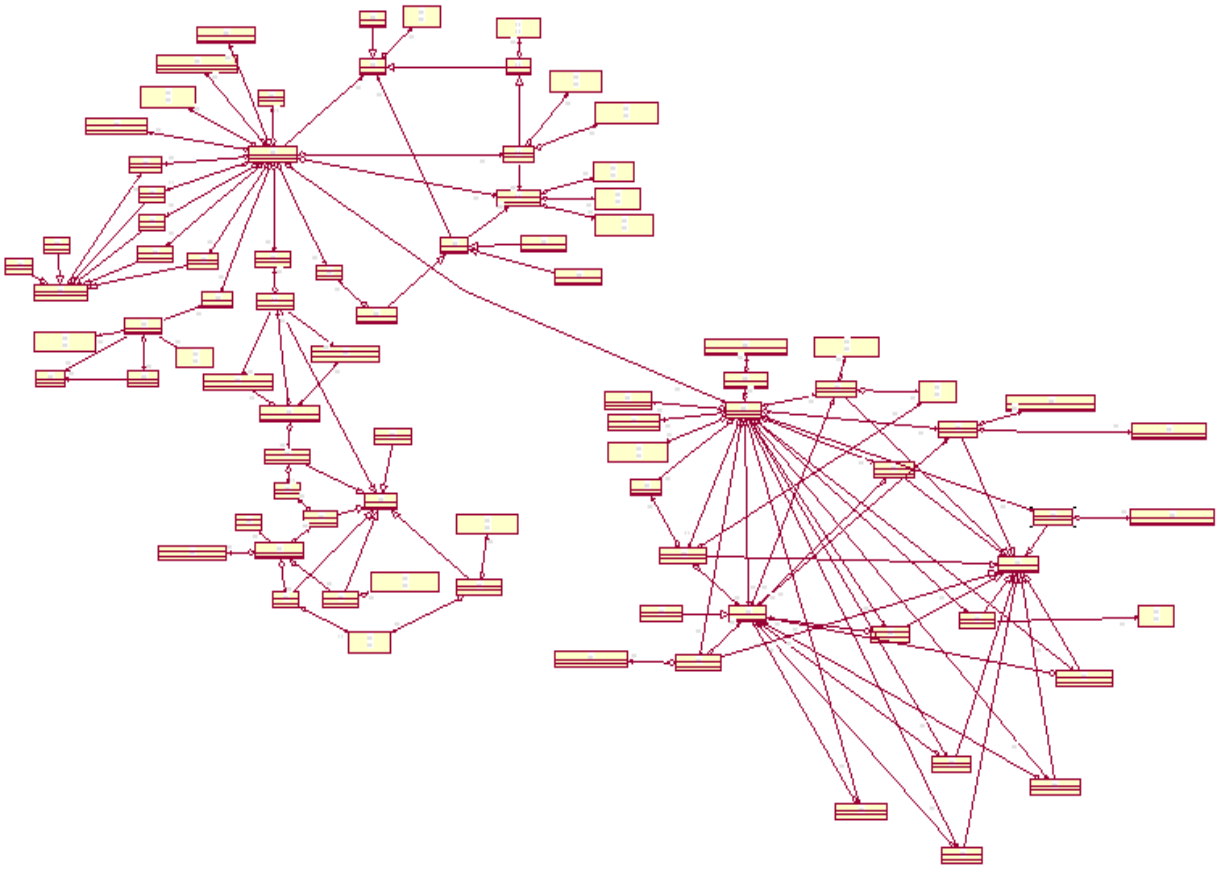
CClientSocket：通信类，发送消息给服务器，并接收服务器端消息。
CCharacterProcess 用户选择角色。
CGameProcess 游戏处理。
CLoginProcess 登录处理：

其它类：
CCreateChar 类： 创建角色。
CgroupMember 类：组
ClientSysMsg
CItem

CImageHandler
CInterface
CmapHandler

CNPCTxtAnalysis
CPatch：FTP 下载在线更新。
CSBuffer： sound
CScriptLine
Csprite 类：
CWHQueue 类：

Rose 逆向工程分析：



游戏服务器负载均衡

1. 多个 LoginGate 服务器。登录时选择。
2. 多个 SelGate 服务器。　用户选择角色服务器。
3. 由 DBSrv 处理用户 CM_SELCHR，选择 GameGate 游戏网关地址转发至客户端。循环遍历 g_xGameServerList,找出 GameGate 在线人数最少的服务器。
4. 多个 GameGate 游戏服务器网关处理用户连接请求,每个 GameGate 服务器连接一个 GameSrv 服务器。
5. GameSrv 根据索引值，加载不同的地图。