# 传奇源码分析

2010年01月30日 星期六 下午 11:02

|  |
| --- |
| 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。 我分析的, 主要是VC SQL版本的, DELPHI翎风源码不做分析, 另外下载了乐都WIL编辑器和乐都MPA地图编辑器这些工具.  传奇源码分析-客户端(WindHorn简述和传奇文件格式分析)  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。  文件类型格式探讨： 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。 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。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。     第二张BMP图片(图片起始位置：0x436 10078) ： F0 01 69 01 , 07 00 D4 FF 刚好大小。第二张Wil起始位置：Ofs:481096 0x75748 知道了图片格式，我们可以写一个抓图片格式的程序了。  传奇源码分析-客户端(全局变量与总体执行流程)  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。  传奇源码分析-客户端(传奇2文件格式分析) 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0fs 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。  传奇源码分析-客户端(传奇2和3 文件格式分析比较)  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。  传奇源码分析-客户端(游戏逻辑处理源分析一)  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服务器发送的消息。     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）。      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）。 |

传奇源码分析（2）(C++)

2010年01月30日 星期六 下午 11:04

|  |
| --- |
| 传奇源码分析-服务器端(LoginGate服务器处理)  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。  传奇源码分析-服务器端(LoginSvr服务器分析)  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。c    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ThreadFuncForM  sg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。  传奇源码分析-客户端(游戏逻辑处理源分析二)  5．接受登录成功后，接收GameSrv服务器发送的消息: 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）。  传奇源码分析-客户端(游戏逻辑处理源分析三)  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\_DOOPENHEAL  TH，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。 |

传奇源码分析（3）(C++)

2010年01月30日 星期六 下午 11:05

|  |
| --- |
| 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。  传奇源码分析-客户端(游戏逻辑处理源分析四)  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\_PICKU  P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。  传奇源码分析-客户端(游戏逻辑处理源分析五 服务器端响应)  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。  传奇源码分析-服务器端(SelGate服务器分析)  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。  如果查看版权出现：引擎可出现异常,请确定是否加载了非法DLL文件,请检查程序目录是否有不正常DLL存在,如果有请删除！ |

传奇源码分析-服务器端(LoginGate服务器处理)

2010年01月30日 星期六 下午 11:08

|  |
| --- |
| **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。 |

传奇源码分析-服务器端(LoginSvr服务器分析)

2010年01月30日 星期六 下午 11:09

|  |  |
| --- | --- |
| |  | | --- | | **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。**c  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。 | |

传奇源码分析-客户端(游戏逻辑处理源分析五 服务器端响应)

2010年01月30日 星期六 下午 11:10

|  |
| --- |
| **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。 |

传奇源码分析-客户端(游戏逻辑处理源分析四)

2010年01月30日 星期六 下午 11:11

|  |
| --- |
| **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](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#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。 |

传奇服务器端源码文件结构

2010年01月30日 星期六 下午 11:12

|  |
| --- |
| MirServer(传奇服务端) │ ├Mir200(传奇游戏服务端，服务于RunGate) │ ├ConLog(登录日志) │ ├Envir(环境设置) │ │ ├Castle(城堡状态) │ │ │ ├AttackSabukWall.txt(攻城信息) │ │ │ ├Sabukw.txt(沙巴克状态信息) │ │ │ └沙巴克配置备份文件.txt(当Sabukw.txt损坏时备用) │ │ ├MapQuest\_def(任务记录) │ │ ├Market\_def(商人信息，与Merchant.txt对应) │ │ ├Market\_prices(商品物价信息) │ │ ├Market\_saved(商品存储信息) │ │ ├Market\_upg(武器升级信息) │ │ ├MonItems(怪物所爆物品及几率，与Monster.DB对应) │ │ ├Npc\_def(NPC人物信息，与NPCs.txt对应) │ │ ├AdminList.txt(GM列表文件) │ │ ├GuardList.txt(守卫坐标文件) │ │ ├MakeItem.txt(毒药配方文件) │ │ ├MapInfo.txt(地图信息文件) │ │ ├MapQuest.txt(任务地图信息) │ │ ├Merchant.txt(商人信息文件) │ │ ├MiniMap.txt(小地图信息文件) │ │ ├Mongen.txt(怪物刷新配置文件) │ │ ├Npcs.txt(NPC信息文件) │ │ ├StartPoint.txt(复活点信息文件) │ │ └UndinbList.txt(打捆物品拆包信息文件) │ ├GuildBase(行会信息) │ │ ├Guilds(行会信息) │ │ └GuildList.txt(行会列表文件) │ ├Log(游戏运行服务端日志) │ ├Map(地图文件) │ ├Notice(提示信息) │ │ ├LineNotice.txt(游戏运行中的提示，蓝字部分) │ │ └Notice.txt(选择服务器之后出现的版权信息) │ ├Share │ ├ShareL │ ├ShareV │ ├!Abuse.txt(脏话信息) │ ├!RunAddr.txt(服务器IP配置文件) │ ├!ServerTable.txt(服务器IP配置文件) │ ├!Setup.txt(游戏运行服务端配置文件) │ ├M2Server.exe(游戏运行主服务端程序) │ └Mir.dat(与客户端的Mir.exe对应) │ ├Mud2(服务端) │ ├DB(数据库) │ │ ├Magic.DB(技能数据库) │ │ ├Monster.DB(怪物数据库) │ │ └StdItems.DB(物品数据库) │ ├DBSrv200(角色选择服务端，服务于ChrSelGate) │ │ ├BackUp(玩家角色信息备份文件夹) │ │ ├Connection(连接日志记录文件夹) │ │ ├FDB(人物数据库，数据库格式为传奇自定义格式) │ │ ├Log(角色选择服务端日志) │ │ ├!AddrTable.txt(角色选择服务端IP配置文件) │ │ ├!IdList.txt(交费账号列表，!Setup.exe中ServiceMode=TRUE时起作用) │ │ ├!ServerInfo.txt(角色选择服务端IP配置文件) │ │ ├DBServer.exe(角色选择服务端) │ │ └DBSrc.ini(角色选择服务端配置文件) │ │ │ └LogSrv(ID登录服务端，服务于LoginGate) │    ├ChrLog(ID创建、修改日志) │    ├ConLog(ID登录日志) │    ├CountLog(ID登录数量统计日志) │    ├IDDB(ID数据库，数据库格式为传奇自定义格式) │    ├!AddrTable.txt(ID登录服务端IP配置文件) │    ├!ServerAddr.txt(ID登录服务端IP配置文件) │    ├!UserLimit.txt(ID登录服务端IP配置文件) │    ├LoginSrv.exe(ID登录帐号密码管理器) │    └LogSrv.ini(ID登录服务端配置文件) │ ├gamelog(物品日志) │   └LogDataSrv(物品日志记录文件夹) │       ├LogBase(包含物品日志文件) │       ├LogData.ini(物品日志配置文件) │       └LogDataServer.exe(日志记录程序) │ ├gateserver(登陆系统) │ ├LoginGate(ID登录接口) │ │ ├LoginGate.exe(账号登录管理器) │ │ └MirGate.ini(ID登录接口配置文件) │ │ │ ├SelChrGate(角色登录接口) │ │ ├Gate30.exe(选择角色管理器) │ │ └Mirgate.ini(角色登录接口配置文件) │ │ │ └RunGate(游戏运行接口) │    ├RunGate.exe(角色在线管理器) │    ├Mirgate.ini(游戏运行接口配置文件) │    └!Abuse.txt(脏话信息) │  ├Share │ └Feed(交费记录文件夹) │ 　 ├FeedIDList.txt(ID帐户冲值列表文件) │ 　 └FeedIPList.txt(IP帐户冲值列表文件) └人物数据存放位置 [Id.db] [id.db.idx] 路径:mirserver\mud2\logsrv\iddb [hum.db] [mir.db] [mir.db.idx] 路径:mirserver\mud2\dbsrv200\fdb  魔法、物品、怪物数据存放位置 [magic.db] [monster.db] [stditems.db] 路径:mirserver\mud2\db |

传奇BLUE转到JSM2的详细脚本分析

2010年01月31日 星期日 上午 00:39

|  |
| --- |
| 1：先替换所有引擎和网关程序。 2：修改QManage.txt去掉Gmexecute 接受 Self，去掉所有[@OnTimer]段 3：ReadRandomStr替换GetRandomName；ISONMAP替换CHECKMAPNAME Self 4：修改DB宝箱系统，宝箱Stdmode=48，钥匙Stdmode=49，参照JS说明书代码。    宝箱卡住问题，因为M2更新后加入了物品类型3表示永远不可得物品，添加Boxs物品类型(0,1,2,3)。 5：修改人形怪，Race值61改成60，不修改会不刷怪。 注：包括其他61代码。 6：先注释掉所有相关魔王岭和卧龙相关NPC连接。(主要这两个需要单独修改，下面有详细解释。) 7：修改魔王岭DB怪，Race值108改成136，109改成135，注：包括人关地关也有。 8：DIV除法计算，修改成DVI 如DIV M44 20 = DVI M44 <$STR(M44)> 20 (92TH赌博系统里用到了)。 9：卧龙山庄(重要)：修改卧龙笔记NPC代码70改68，卧龙古书和各种宝箱NPC代码参考JS说明。 10:修改地图参数，MISSION 如魔王岭地图不允许使用任何物品和技能。 11:卧龙山庄小地图，MiniMap.txt 如下 ：Hero1 100302。 12:怪物挖MonUseItems修改,挖物需要灵符。详细查看JS说明。 13:修改卧龙怪挖到物品全服红字提示。M2上面点查看-列表信息2-物品规则，填加物品选择挖取提示。 14:修改卧龙人型怪，提高挖取装备几率。小怪挖取几率设置3，中级怪设置5，大怪设置10。 15:解决M2报错问题：[Exception] TBaseObject.RecalcHitSpeed,检查MonUseItems下人型怪设置。 16:修改UserCmd.txt用户自定义命令文件路径在D:\MirServer\Mir200下面。 17:气血石回血问题：M2上面点选项-功能设置，看到HP设置50%间隔1秒，MP设置20%间隔1秒。 18:重制M2信息列表，M2上面点查看-列表信息-游戏日志过滤，删除全部，重新添加全部保存。    -物品名称自定义，删除所有。-管理员列表，不用说这里可以动态添加GM。 19:修改魔王岭怪速度正常，在DB里调，92TH的怪赶上飞机了。加入使用怒牌换取弓箭手。[详细脚本查看]    [@KillMissionMob] //在QFunction-0.txt下加入此段。    #if    large M98 0    #act    INC M30 1    SENDMSG 6 [提示]:您当前拦截怪物数量为:【<$STR(M30)>】个    BREAK 20:修改卧龙山庄地图PK掉装备，卧龙怪可以正常爆出火龙珠。适合FSD版本设计。[详细脚本查看] 21:修改天关人关地关，GetRandomName .\QuestDiary\ 替换成GetRandomName ..\QuestDiary\ 不然M2报错 22:修改天关获取经验列表，不能带几率，正确格式，每一行只需要写获得经验值。 23:修改天关获得经验红字提示：SENDMSG 0 [闯天关]：%S在天关寻宝中获得了经验值:[<$STR(S8)>] 24:修改地关，人关获得物品列表，不能带几率，正确格式，每一行只需要物品名称。[详细脚本查看]    [看到这里就知道JS的随机读取列表不支持几率怎么办？如：高级物品就写1行，垃圾物品多写几行即可。] 25:修改人关，采用非K004地图魔王岭系统，设置怪物目的地坐标。    [KILLSLAVE <$STR(S37)> 20 23 100 \* 10 0修改成KILLSLAVE]将当前人物的宝宝全部杀死。    [CHECKRANGEMONCOUNTEX SELF修改为CheckMapMobCount <$MAP>]检测指定坐标内自己宝宝数量。    [RECALLMOBEX 人关勇士 25 18 0 4 60 0 -1 0修改格式ReCallMobEx 人关勇士 2 25 18]手动修改。    [在QFunction-0.txt下[@KillMissionMob]实现杀怪统计]必须#if判断LARGE M59 0如下：    [@KillMissionMob]    #IF    LARGE M59 0    #ACT    INC M58 1    SENDMSG 5 目前已消灭<$STR(M58)>名人关匪徒。5u wl.net    BREAK 26:修改人关地图参数：添加MISSION，不允许使用任何物品和技能。 27:修改人关怪物DB：物类型 Race=136 任务不攻击怪。 28:人关提示：KillScTimer 5 删除掉，这个命令是终止个人定时系统，因为JS版本上没用到个人定时系统 |

Delphi通用源代码格式规则

2010年02月02日 星期二 下午 06:51

|  |
| --- |
| .1.通用源代码格式规则 1). 缩进 缩进就是每级间有两个空格。不要在源代码中放置制表符。这是因为，制表符的宽度随着不同的设置和代码管理实用程序(打印、文档及版本控制等)而不同。  通过使用Tools|Environment 菜单，在Environment Options 对话框的General页上，不要选中Use Tab Character 和Optional Fill 复选框，这样，制表符就不会被保存。  2). 边距 边距设置为80个字符。源代码一般不会因写一个单词而超过边距，但本规则比较灵活。只要可能，长度超过一行的语句应当用逗号或运算符换行。换行后，应缩进两个字符。  3). begin...end 语句 begin 语句必须单独占一行。例如，下面第一行是错误的，而第二行正确：  for i:=0 to 10 do begin // 错, begin 与f o r 在同一行  for i:=0 to 10 do // 对, begin 在另外一行中 begin  本规则的一个特殊情况是，当begin 为else 语句的一部分时，例如：  if some statement = then begin . . . end else begin Some Other Statement; end;  注意：end 语句总单独一行。当begin 不为else 语句的一部分时，相应的end 语句与begin 语句的缩进量相同。  4).注释 我们通常使用“{...}”类型的块注释，以前的“(\*...\*)”类型的块注释用于临时注释掉暂不使用的代码，从Delphi 2开始支持“//”行注释，如果决定不在支持Delphi 2.0以下的版本，可以使用“//”注释。  2.2.Object Pascal语句格式语句书写规范与用法 1). 括号 在左括号与下一字符之间没有空格。同样，右括号与前一字符也没有空格。下面的例子演示了正确与不正确的空格。  CallProc( Aparameter ); // 错! CallProc(Aparameter); // 正确!  不要在语句中包含多余的括号。在源代码中，括号只有在确实需要时才使用。下面的例子演示了正确与不正确用法：  if (I=42) then // 错，括号是多余的 if (I=42) or (J=42) then // 正确，必须使用括号  2). 保留字和关键字 Object Pascal 语言的保留字和关键字总是完全的小写。下面是Delphi 5保留字列表:  and array as asm  begin case class const  constructor destructor dispinterface div  do downto else end  except exports file finalization  finally for function goto  if implementation in inherited  initialization inline interface is  label library mod nil  not object of or  out packed procedure program  property raise record repeat  resourcestring set shl shr  string then threadvar to  try type unit until  uses var while with  xor private protected public  published automated  3). 过程和函数 (1). 格式 过程名应当以大写字母开始，且大小写交错以增加可读性。下面是一个不正确的写法：  procedure thisisapoorlyformattedroutinename;  改成这样写就对了：  procedure ThisIsMuchMoreReadableRoutineName;  (2). 形参 (1) 格式 只要可能，同一类型的形参应当归并在一起：  procedure Foo(Param1,Param2,Param3:Imteger;Param4:string);  (2) 参数顺序 形参的顺序主要要考虑寄存器调用规则。最常用的参数应当作为第一个参数，按使用频率依次从左到右排。输入参数位于输出参数之前。范围大的参数应当放在范围小的参数之前。例如：  SomeProc(aPlanet, aContinent, aCountry, aState, aCity).  有些则例外。例如，在事件处理过程中，TObject 类型的Sender 参数往往是第一个要传递的参数。  (3) 常量参数 要使记录、数组、短字符串或接口类型的参数不能被过程修改，就应当把形参标以Const 。这样，编译器将以最有效的方式生成代码，保证传递的参数不可变。  如果其他类型的参数希望不被过程所修改，也可以标上Const 。尽管这对效率没有影响，但这给过程的调用者带来了更多的信息。  4). 变量 (1). 局部变量 局部变量用于过程内部，果需要的话，应当在过程的入口处立即初始化变量。局部的AnsiString 类型的变量自动被初始化为空字符串，局部的接口和dispinterface类型的变量自动被初始化为nil，局部的Variant和OleVariant类型的变量自动被初始化为Unassigned。  (2). 全局变量 一般不鼓励使用全局变量。不过，有时候需要用到。即使如此，也应当把全局变量限制在需要的环境中。例如，一个全局变量可能只在单元的实现部分是全局的。  全局数据如果将由许多单元使用，就应移动到一个公用单元里被所有对象使用。全局数据可在声明时直接初始化为一个值。注意，所有全局变量自动进行零初始化，因此，不要将全局变量初始化为诸如0 、nil、或Unassigned等空值。零初始化的全局变量在.EXE文件中不占空间。零初始化的数据保存在虚拟的数据段中，而虚拟数据段只在应用程序启动时才分配内存。非零初始化的全局数据则在.EXE文件中占空间。  5). 类型 (1). 大小写规则 类型标识符是保留字，应当全部小写。Win32 API 类型常常全部大写，并且遵循诸如Windows.pas或其他API单元中关于特定类型名的规则。对于其他变量名，第一个字母应大写，其他字母则大小写交错。下面是一些例子：  var MyString: string; // 保留字 WindowsHandle: HWND; // Win32 API 类型 I: Integer; //在System单元中引入的类型标识  (2). 浮点型 不鼓励使用Real类型，因为它只是为了与老的Pascal代码兼容而保留的。通常情况下，对于浮点数应当使用Double。Double可被处理器优化，是IEEE定义的标准的数据格式。当需要比Double提供的范围更大时，可以使用Extend。Extend是intel专用的类型，Java不支持。当浮点变量的物理字节数很重要时(可能使用其他语言编写DLL)，则应当使用Single。  (3).Variant和OleVariant 一般不建议使用Variant和OleVariant。但是，当数据类型只有在运行期才知道时(常常是在COM和数据库应用的程序中)，这两个类型对编程就有必要。当进行诸如自动化ActiveX控件的COM编程时，应当使用OleVariant；而对于非COM编程，则应当使用Variant。这是因为，Variant能够有效地保存Delphi的原生字符串，而OleVariant则将所有字符串转换为OLE字符串(即WideChar字符串)，且没有引用计数功能。  6). 语句 (1). If 语句 在if/then/else语句中，最有可能执行的情况应放在then子句中，不太可能的情况放在else子句中。为了避免出现许多if语句，可以使用case语句代替。如果多于5级，不要使用if语句。请改用更清楚的方法。不要在if语句中使用多余的括号。  如果在if语句中有多个条件要测试，应按照计算的复杂程度从右向左排。这样，可以使代码充分利用编译器的短路估算逻辑。例如，如果Condition1比Condition2快，Condition2比Condition3快，则if语句一般应这样构造：  if Condition1 and Condition2 and Condition3 then  如果Condition3为False的机会很大，利用短路估算逻辑，我们也可以将Condition3放在最前面：  if Condition3 and Condition1 and Condition2 then  (2). case 语句 (1) 概述 case语句中每种情况的常量应当按数字或字母的顺序排列。每种情况的动作语句应当简短且通常不超过4 - 5 行代码。如果动作太复杂，应将代码单独放在一个过程或函数中。Case语句的else子句只用于默认情况或错误检测。  (2) 格式 case语句遵循一般的缩进和命名规则。  (3). while 语句 建议不要使用Exit过程来退出while循环。如果需要的话，应当使用循环条件退出循环。所有对while循环进行初始化的代码应当位于while入口前，且不要被无关的语句隔开。任何业务的辅助工作都应在循环后立即进行。  (4). for 语句 如果循环次数是确定的，应当用for语句代替while语句。  (5). repeat 语句 repeat语句类似于while循环，且遵循同样的规则。  (6). with 语句 (1) 概述 with语句应小心使用。要避免过度使用with语句，尤其是在with语句中使用多个对象或记录。例如：  with Record1,Record2 do  这些情况很容易迷惑编程人员，且导致调试困难。  (2) 格式 with语句也遵循本章关于命名和缩进的规则。  7). 结构化异常处理 (1). 概述 异常处理主要用于纠正错误和保护资源。这意味着，凡是分配资源的地方，都必须使用try...finally来保证资源得到释放。不过，如果是在单元的初始/结束部分或者对象的构造器/析构器中来分配/释放资源则例外。  (2). try...finally的用法 在可能的情况下，每个资源分配应当与try...finally结构匹配，例如，下面代码可能导致错误：  SomeClass1 := TSomeClass.Create; SomeClass2 := TSomeClass.Create; try { do some code } finally SomeClass1.Free; SomeClass2.Free; end;  上述资源分配的一个安全方案是：  SomeClass1 := TSomeClass.Create; try SomeClass2 := TSomeClass.Create; try     { do some code } finally     SomeClass2.Free; end; finally SomeClass1.Free; end;  (3). try...except的用法 如果你希望在发生异常时执行一些任务，可以使用try...except。通常，没有必要为了简单地显示一个错误信息而使用try...except，因为Application对象能够自动根据上下文做到这一点。如果要在子句中激活默认的异常处理，可以再次触发异常。  (4). try...except...else的用法 不鼓励使用带else子句的try...except，因为这将阻塞所有的异常，包括你没有准备处理的异常。  3.命名规范 3.1.过程(Procedure)与函数(Function) 1).命名 过程与函数名应当有意义。进行一个动作的过程最好在名称前加上表示动作的动词为前缀。例如：  procedure FormatHardDrive;  设置输入参数值的过程名应当以Set 为其前缀，例如：  procedure SetUserName;  获取数值的过程名应当以Get 为其前缀，例如：  function GetUserName:string;  2).形参 所有形参的名称都应当表达出它的用途。如果合适的话，形参的名称最好以字母a 为前缀，例如：  procedure SomeProc(aUserName:string; aUserAge:integer);  当参数名与类的特性或字段同名时，前缀a 就有必要了。  3).命名冲突 当两个单元中含有相同名称的过程时，如果调用该过程，实际被调用的是Uses 子句中较后出现的那个单元中的过程。为避免这种情况，可在方法名前加想要的单元名，例如：  SysUtils.FindClose(SR);  或Windows.FindClose(Handle);  3.2.变量(Variable) 变量的名称应当能够表达出它的用途。循环控制变量常常为单个字母，诸如I 、J 或K 。也可以使用更有意义的名称，例如UserIndex。布尔变量名必须能清楚表示出True 和False 值的意义。  1). 局部变量 局部变量遵循其他变量的命名规则。  2). 全局变量 全局变量以大写字母“G”打头，并遵循其他变量的命名规则。  3.3.类型(Type) 3.3.1.一般类型 1). 枚举型 枚举类型名必须代表枚举的用途。名称前要加T字符作为前缀，表示这是个数据类型。枚举类型的标识符列表的前缀应包含2 - 3 个小写字符，来彼此关联。例如：  TSongType=(stRock, stClassical, stCountry, stAlternative, stHeavyMetal, stRB);  枚举类型的变量实例的名称与类型相同，但没有前缀T ，也可以给变量一个更加特殊名称，诸如：FavoriteSongTypel、FavoriteSongType2等等。  3.3.2.构造类型 1). 数组类型 数组类型名应表达出该数组的用途。类型名必须加字母“T”为前缀。如果要声明一个指向数组类型的指针，则必须加字母P 为前缀，且声明在类型声明之前。例如：  type PCycleArray = ^TCycleArray; TCycleArray=array[1..100] of integer;  实际上，数组类型的变量实例与类型名称相同，但没有“T”前缀。  2). 记录类型 记录类型名应表达出记录的用途。类型名必须加字母T为前缀。如果要声明一个指向记录类型的指计，则必须加字母P为前缀，且其声明在类型声明之前。例如：  type PEmployee = ^TEmployee; TEmployee = record     EmployeeName: string;     EmployeeRate: Double; end;  3.3.3.类类型(Class) 1). 命名与格式 类的名称应当表达出类的用途。一般的类名前要加字母“T”，如果是接口类那么类名前要加“I”，错误异常类的类名前要加“E”，而类引用类型（Class-reference type）则要在类名后加“Class”。例如：  type TCustomer = class(TObject); ICustomer = interface; TCustomerClass = class of TCustomer ECustomerException = class(Exception);  类的实例名称通常与类名相同，只不过没有前缀“T”。  var Customer: TCustomer;  注意:关于元件的命名，请参阅“元件类型”。  2).字段 (1) 命名与格式 字段的命名遵循与变量相同的规则，只不过要加前缀F ，表示这是字段。  (2) 可见性 所有字段必须为私有。如果要在类的作用域之外访问字段，可借助于类的属性来实现。  3).方法 (1) 命名与格式 方法的命名遵循与过程和函数相同的规则。  (2) 静态方法 当你不希望一个方法被派生类覆盖时，应当使用静态方法。  (3) 虚拟方法(virtual)与动态方法(dynamic) 当你希望一个方法能被派生类覆盖，应当使用虚拟方法(virtual)。如果类的方法要被多个派生类直接或间接地使用，则应当用动态方法(dynamic)。例如，某一个类含有一个被频繁覆盖的方法，并有100个派生类，则应将方法定义为动态的，这样可以减少内存的开销。  (4) 抽象方法(abstract) 如果一个类要创建实例，则不要使用抽象方法。抽象方法只能在那些从不创建实例的基类中使用。  (5) 属性访问方法 所有属性访问方法应当定义在类的私有或保护部分。属性访问方法遵循与过程和函数相同的规则。用于读的方法应当加“Get”前缀，用于写的方法应当加“Set”前缀，并且有一个叫Value的参数，其类型与属性的类型相同。例如：  TSomeClass = class(TObject) private FSomeField: Integer; protected function GetSomeField: Integer; procedure SetSomeField(Value: Integer); public property SomeField: Integer read GetSomeField write SetSomeField; end;  尽管不是必须，但还是建议你使用写访问方法来访问代表私有字段属性。  4).属性 属性作为私有字段的访问器，遵循与字段相同的命名规则，只不过没有F前缀。属性名应为名词，而不是动词。属性是数据，而方法是动作。数组属性名应当是复数，而一般的属性应当是单数。  3.3.4.元件类型 1). 元件类型的命名标准 元件的命名与类的命名类似，只不过当它与其它元件名称冲突时，你可以加上3个字符的前缀，用以标识公司、个人或其他实体。例如，一个时钟元件可以这样声明：  TddgClock = class(TComponent)  注意，作为前缀的3 个字符要小写。  2). 元件实例的命名规则 元件实例的名称应当能够描述其实际意义，这里命名规则使用了一个变更的匈牙利前缀命名规范。使用前缀而不使用后缀的原因是在搜寻时，在对象检查器和代码探索器中搜寻构件的名字比搜寻构件的类型更容易。在这个标准中，元件实例名包括两个部分：前缀和性质标识名。  (1). 元件的前缀 元件的前缀多是表现元件类型的字母缩写。参见下面表中的元件前缀：  元件类名 元件前缀  TActionList, TAction表示动作的列表项 act  TButton, TSpeedButton, TBitBtn等所有的按钮类 btn  TCheckBox, TDBCheckBox等所有的检查框 chk  TRadioButton单选按钮类 rdo  TToolBar工具条 tb  TMainMenu所有的主菜单类 mm  TMainMenuItem所有的菜单项类 mi  TPopupMenu所有的弹出式菜单类 pm  TPopupMenuItem所有的弹出式菜单项类 pmi  TLabel, TStaticText等所有用来显示的标签类 lbl  TPanel等所有的面板类 pnl  TPageControl等所有的页式控件类 pgc  TEdit, TMaskEdit等所有的单行编辑框类 edt  TMemo, TRichEdit等所有的多行编辑框类 mmo  TDrawGrid, TStringGrid等所有的网格类 grd  TAnimate等所有的动画类 ani  TImageList等所有的图片列表类 il  TImage等图片类 img  TChart图表类 cht  TComboBox, TDBComboBox等所有的下拉式列表框类 cbo  TListBox, TDBList等所有的列表框类 lst  TTreeView tv  TListView lv  THotKey hk  TSplitter等所有的分隔符类 spt  TOpenDialog等所有的对话框元件类 dlg  TTable等所有的数据表类 tbl  TQuery等所有的SQL查询类元件 qry  TClientDataSet所有的客户数据集元件 cds  TDataSource ds  TDatabase db  TSockConnection,TDCOMConnection等连接元件类 con  TQuickRep, TFastReport等所有的报表元件类 rpt  TDDEClientConv,TDDEClientItem等所有的DDE元件类 dde  TMonthCalendar等所有的日历类 cal  TGroupBox等控件类 grp  如上所示，元件类型前缀是从分析描述元件的类型性质而来的。通常情况下，下面的规则描述如何定义一个元件类型前缀：  从元件类型名中移去T前缀。例如TButton变成Button。  除了第一个元音，删去所有元音字母。例如，Button变成bttn，Edit变成edt。  压缩双字母。例如，bttn变成btn。  如发生冲突，则在某一元件前缀中加入一个元音。例如在TBatton元件的前缀中加入元音变为batn，以区别TButton的前缀。  不过，上述规则首先得保证前缀名称必须符合习惯，做到见名知意，如:TDDEClientConv控件的前缀就是一个例外。  注意：元件的前缀是为了表示出元件的类型，是按钮，还是标签等等，因此没有必要为每一个特别元件类建立一个元件前缀，如: TMyButton的元件前缀仍为btn。  (2). 元件性质标识名 元件性质标识名是元件意图的描述。例如，一个用于关闭窗体的TButton元件实例可命名为btnClose。一个编辑姓名的元件实例可命名为edName。  3.3.5.窗体与对话框类型 1). 窗体类型的命名标准 窗体或对话框类型的名称应当表达出窗体的用途，如果是窗体要加“Tfrm”前缀，如果是对话框要加“Tdlg”，后跟描述性名。例如，About窗体类型名称为：  TfrmAbout = class(TForm)  主窗体的类型名称为:  TfrmMain = class(TForm)  客户登录窗体的类型名称为:  TfrmCustomerEntry = class(TForm)  登陆对话框的类型名称为：  TdlgLogin = class(TForm)  2). 窗体实例的命名标准 窗体实例的名称与相应的类型名称相同，但没有前缀T 。例如，前面提到的窗体类型与实例的名称为：  类型名 实例名  TfrmAbout frmAbout  TfrmMain frmMain  TfrmCustomerEntry frmCustomerEntry  TdlgLogin dlgLogin  3). 自动创建的窗体 除非特别原因,只有主窗体才自动生成。其他所有窗体必须从Project Options对话框的自动生成列表中删除。更进一步信息,请参阅后面几节。  4). 模式窗体实例化函数 所有窗体单元都应当含有实例化函数，用于创建、设置、模式显示和释放窗体。这个函数将返回由窗体返回的模式结果。传递给这个函数的参数遵循参数传递的规则。之所以要这样封装，是为了便于代码的重用和维护。  窗体的变量应当从单元中移走，改在窗体实例化函数中作为局部变量定义(注意，要求从Project Options对话框的自动生成列表中移走该窗体。请看前面的内容。  例如，下面的单元文件演示了GetUserData的实例化函数。  Unit UserDataFrm;  Interface  Uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;  Type  TfrmUserData = class(TForm)     edtUserName: TEdit;     edtUserID: TEdit; private { Private declarations } public { Public declarations } end;  function GetUserData(var aUserName: String;var aUserID: Integer): Word;  implementation  {$R \*.DFM}  function GetUserData(var aUserName: String;var aUserID: Integer): Word; var frmUserData: TfrmUserData; begin frmUserData := TfrmUserData.Create(Application); frmUserData.Caption:='Getting User Data' ; Result : = frmUserData.ShowModal; if Result=mrOK then begin     aUserName := frmUserData.edtUserName.Text;     aUserID := StrToInt(frmUserData.edtUserID.Text); end; finally     frmUserData.Free; end; end;  End.  5).窗体框架与复合窗体 如果一个窗体结构过于复杂，就必须将其分化成为一个主窗体框架以及嵌入到主窗体框架的若干子窗体框架。如：  TfrmMainFrame: TfrmInfoFrame,TfrmEditorFrame  使用窗体框架，主要是为了解决界面和代码复用问题，以及提高单元代码的内聚力（划分后，每一个窗体框架为一个独立单元），从而提高软件工程质量。你必须提炼出界面关联代码（可复用的）和应用关联代码（不能复用的）。  3.3.6.数据模块类型 1). 数据模块的命名标准 数据模块类型名称应表达出它的用途，且要加前缀“Tdm”，后跟描述性名称。例如，Customer数据模块的类型名称为：  TdmCustomer = class(TDataModule)  Orders 数据模块的类型名称为：  TdmOrder = class(TDataModule)  2). 数据模块实例的命名标准 数据模块实例的名称应当与相应的类型名称相同，但没有前缀T 。例如，前面的数据模块类型、实例名称如下：  类型名称 实例名  TdmCustomer dmCustomer  TdmOrder dmOrder  3.4.文件 建议在所有源文件、项目文件和单元文件使用结构化的文件头信息。一个文件头至少应包含以下信息：  { Copyright @ Year by Authors }  3.4.1.项目文件 项目文件的名称应当具有描述意义。例如，“The Delphi 5 Developer’s Guide Bug Manager ”的项目名称为DDGBugs.dpr，一个系统信息程序的名称为SysInfo.dpr。  3.4.2.窗体文件 窗体文件的名称应当表达出窗体的用途，且具有Frm后缀。例如，About窗体的文件名叫AboutFrm.dfm，主窗体的文件名叫MainFrm.dfm。  3.4.3.数据模块文件 数据模块文件的名称应当表达出数据模块的作用，且具有DM后缀。例如，Customers数据模块的文件名叫CustomersDM.dfm。  3.4.4.远程数据模块文件 远程数据模块文件的名称应当表达出远程数据模块的用途。名称后要加RDM后缀。例如，Customers远程数据模块的文件叫CustomersRDM.dfm。  3.4.5.单元文件 1). 普通单元 (1) 单元名 单元的名称应当有描述性。例如，应用程序的主窗体单元叫MaimFrm.pas。  (2) Uses 子句 Interface部分的Uses子句应当只包含该部分需要的单元。不要包含可能由Delphi自动添加的单元名。Implementation部分的Uses子句应当只包含该部分需要的单元，不要有多余的单元。  (3) Interface 部分 Interface部分应当只包含需要被外部单元访问的类型、变量、过程与函数的声明。而且，这些声明应当在Implementation部分之前。  (4) Implementation 部分 Implementation部分包括本单元私有的类型、变量、过程与函数的实现。  (5) Initialization 部分 不要在Initialization部分放置花费时间很多的代码。否则，将导致应用程序启动时显得很慢。  (6) Finalization 部分 确保释放所有在Initialization部分中分配的资源。  2).窗体单元 窗体单元文件的名称与相应的窗体名称相同，只是要将前缀变成后缀。例如，About窗体的单元名称叫AboutFrm.pas。主窗体的单元文件名称叫MainFrm.pas。  3).数据模块单元 数据模块单元文件的名称与相应的数据模块名称相同。例如，数据模块单元的名称叫CustomersDM.pas。  4).通用单元 通用单元的名称应当表达出它的用途，名称前要加“u”前缀。例如，一个实用调试工具单元的名称叫uDebugUtilities.pas，包含全局变量的单元名称叫uCustomerGlobals.pas。  注意，一个项目中单元名称必须是唯一的。通用单元名不能重名。  5).元件单元 (1). 命名 元件单元应放在单独的路径中，以表明它们是定义元件的单元。它们一般与项目不放在同一路径下。单元文件名称应表达出其内容。  注意，有关元件命名标准的更多信息，请参阅“元件类型的命名标准”。  元件单元只能含有一个主要元件，这是指出现在元件选项板上的元件。其他辅助性的元件或对象也可以包含在同一单元中。  (2). 注册单元 元件的注册过程应当从元件单元中移走，放在一个单独的单元中。这个注册单元用于注册所有元件、属性编辑器、元件编辑器、向导等。  元件注册应当在设计期包中进行。因此，注册单元应当包含在设计期包而不是运行期包中。建议注册单元这样命名：  xxxReg.pas  其中，xxx字符前缀，以标识元件包名称或公司、个人、其他实体。例如，注册单元命名为xxxReg.pas。  6).包文件（.dpk）命名规则 (1). 运行期包与设计期包 运行期包中应当只包含所需要的单元。那些属性编辑器和元件编辑器的单元应当放在设计期包中。注册单元也应当放在设计期包中。  (2). 文件命名标准 包的命名遵循下列模式：  dcliiiDescvvCn.pkg —设计期包 iiiDescvvCn.pkg    —运行期包  其中，iii代表一个2-3字符的前缀，用于标识公司、个人或其他需要标识的事情，也可不要；Desc表示该控件包的简短描述；vv代表包的版本号，你可以根据需要取舍；前缀“dcl”表示设计期包，没有该前缀表示运行期包；字母“Cn”表示编译器类型与编译器版本号，如：Delphi5=D5, Delphi4=D4, CBuilder3=C3...。  注意包名称中的lib或std分别表示这是设计期包还是运行期包。例如：  dclrbStdCompsD5.pkg —Delphi 5的设计期包 rbStdCompsD5.pkg       —Delphi 5的运行期包  4.Delphi代码自动格式化工具 尽管大多数的代码自动格式化工具能够帮你重排源程序格式，以及更新保留字和标示符的大小写，但是这最好在使用版本控制前进行，如果你已经使用了版本控制，建议你不要轻易使用代码自动格式化工具，哪怕多一个空格，版本控制工具也会认为该行已被修改，从而给程序管理带来不变。 |