红鸟平台游戏开发技术文档

### 本技术文档是针对红鸟平台牌类游戏而言。游戏以静态链接的形式由平台调用，游戏内部实现由开发人员自己编写。游戏工程编辑生成的文件分为服务端和客户端，服务端生成文件格式为“游戏ID .dll”；客户端的生成文件格式为“游戏ID .ico”（此处需要特殊说明，客户端生成的文件就是一个 dll 文件，只是命名不一样而已）。

1. 工程目录

这里就涉及统一性，因为程序员的习惯都不一样，如果游戏很多那么管理就很成问题，所以红鸟设立了统一管理规范，此标准只针对红鸟平台棋牌而言。

文件目录标准：文件目录一般分为三层，最外一层以游戏 ID+\_游戏名称的第一个母命名，例如斗地主：10100003\_DDZ。第二层里有三个文件夹，所有游戏中三个文件夹名字都想同：Documents(游戏说明文档、策划案等)、GameCode(代码)、run(游戏资源)。第三层主要是对 GameCode 来讲的。GameCode 里面也有三个文件夹分别为：Client(客户端工程相关文件)、GameMessage(客户端和服务端公共文件)、Server(服务端工程文件)。除了三个文件还包括两个工程生成的解决方案文件和一个记录文件，解决方案文件由 游戏 ID+\_游戏名称的第一个母+.sln构成。例如斗地主：10100003\_HLDDZ\_2010.sln。

代码规范标准：根据红鸟代码规范标准为准。

1. 开发环境

开发的准备工作就是搭建开发环境，游戏开发要由包含很多平台的头文件，这些头文件都放在 sdk 文件夹下面，还有包含的一些库，这些都要在工程属性中设置好。

服务端：

依赖头文件目录：

../../../../SDK/include/common;

../../../../SDK/include/ServerCommon;

依赖库目录：

../../../../SDK/lib/common;

../../../../SDK/lib/Server;

依赖静态库：AFCComDllEx.lib;CfgEngine.lib

客户端：

依赖头文件目录：

../../../../SDK/include/GameLaunchSDK;

../../../../sdk/include/NewGameFrame/Client;

依赖库目录：

../../../../SDK/lib/Common;

../../../../SDK/lib/Game;

依赖静态库：DllForGame.lib;cfgEngine.lib

客户端除了这两个库可能还有其它库，例如音效支撑库:irrKlang.lib, 可能还有其它库根据游戏自己调用为准，但前面的两个库必须包含。

开发环境搭建好后就要有游戏运行环境用来调试程序，因为红鸟棋牌游戏不支持 debug 版本，所以只能编译 release 版本运行调试。游戏运行环境相信大家都很娴熟所以在此就不在迂腐了，不过还是提醒一下：注意编译生成文件的路径。

以上两步完成以后就可以开始写代码了，不过在写代码之前一定要对游戏框架熟悉，这样写出来的代码才可能健壮。

那么我们就来熟悉一下游戏的框架。

1. 游戏框架

游戏解决方案一般包括两个工程，相信大家都应该知道了，就是服务端工程和客户端工程。

**1、服务端工程框架：**

服务端项目命名为 Server。每个服务端都有一个公共的类CServerGameDesk 。该类继承自平台的GameDesk类，构成了一张虚拟游戏桌的所需要的逻辑和业务所需要的接口实现。

先介绍 CServerGameDesk，定义如下

class CServerGameDesk : public CGameDesk

{

public:

///构造函数CServerGameDesk();

///析构函数

virtual ~CServerGameDesk();

public:

///游戏开始

virtual bool GameBegin(BYTE bBeginFlag);

///游戏结束

virtual bool GameFinish(BYTE bDeskStation, BYTE bCloseFlag);

///判断是否正在游戏

virtual bool IsPlayGame(BYTE bDeskStation);

///游戏数据包处理函数

virtual bool HandleNotifyMessage(BYTEbDeskStation,NetMessageHead\*pNetHead, void \* pData, UINT uSize, UINT uSocketID, bool bWatchUser);

///框架数据包处理函数

bool HandleFrameMessage(BYTE bDeskStation, NetMessageHead \* pNetHead, void \* pData, UINT uSize, UINT uSocketID, bool bWatchUser);

///用户离开游戏桌

virtual BYTE UserLeftDesk(BYTE bDeskStation, CGameUserInfo \* pUserInfo)

///获取游戏状态信息

virtual bool OnGetGameStation(BYTE bDeskStation, UINT uSocketID, bool bWatchUser);

///重置游戏状态

virtual bool ReSetGameState(BYTE bLastStation);

///定时器消息

virtual bool OnTimer(UINT uTimerID);

///玩家重连处理

virtual bool UserReCome(BYTE bDeskStation,CGameUserInfo\* pNewUserInfo);

///初始化游戏逻辑

virtual bool InitDeskGameStation();

///玩家断线处理

virtual bool UserNetCut(BYTE bDeskStation, CGameUserInfo \* pLostUserInfo);

};

CServerGameDesk 继承平台的一个类 CGameDesk。上面的函数都是重写 CGameDesk里面的函数，除了 OnGetGameStation ，ReSetGameState 。OnGetGameStation ，ReSetGameState是抽象函数，所以必须在 CServerGameDesk里实现。

在红鸟平台上写游戏掌握以后几个函数就差不多了，但游戏的好坏和难易都要在游戏里面实现了，平台只是一个环境，游戏的本身还要自己去塑造。

## **2、客户端工程框架：**

客户端命名为 Client，CNewGameClient也继承自平台，用来在游戏不同的状态显示不同的界面元素，同时响应玩家请求并将请求发送至服务端处

理。CUpGradeGameLogic是与服务端公用的类。

class CNewGameClient : public IGameImpl

{

public:

CNewGameClient(IGameFrame \*pGameFrame, IUserInterface \*pUI);

virtual ~CNewGameClient();

private:

CNewGameClient(const CNewGameClient &other);

CNewGameClient &operator = (const CNewGameClient &other);

private:

///框架游戏指针

IGameFrame \*m\_pGameFrame;

///界面游戏指针

IUserInterface \*m\_pUI;

public:

/// 初始化

virtual int Initial();

/// 初始化UI

virtual int InitUI();

/// 游戏消息

virtual int HandleGameMessage(WORD nMainID,WORD nSubId,BYTE \* buffer,int nLen);

/// 界面消息

virtual int OnUIMessage(TUIMessage\* pMessage);

/// 重新设置状态

virtual int ResetGameFrame(void);

/// 玩家进入

virtual int GameUserCome(void);

/// 玩家离开

virtual int GameUserLeft(int nIndex);

/// 玩家断线

virtual int GameUserOffLine(void);

// 是否正在游戏，由游戏实现

// 旁观玩家返回false，游戏玩家正在游戏中返回true，其它状态返回false

virtual bool IsPlayingGame();

/// 定时器消息

virtual void OnTimer(int nTimerID);

/// 设置状态

virtual void SetStationParameter(BYTE bGameStation);

///设置游戏状态

virtual BYTE GetStationParameter();

///游戏结束

virtual bool OnGameQuiting();

};

CNewGameClient也是继承 GameLauncher的一个类 IGameImpl，这个类是一个抽象类，基类中除了构造函数，所有的函数都是抽象函数。所以必须在CNewGameClient 实现全部的抽象函数。同时还有两个对象成员，这两个成员的数据类型的定义也是在 GameLauncher中，m\_pGameFrame是游戏框架的数据，通过这个指针能获取自己和其它玩家的信息，还能控制游戏的走向，m\_pUI是游戏UI接口 ，通过这个指针能够调整和控制界面元素。

除此之外还有两个全局函数：

IGameImpl \*CreateNewGameImpl(IGameFrame\* pIBase, IUserInterface \*pIUI)

{

return dynamic\_cast<IGameImpl\*>(new CNewGameClient(pIBase, pIUI));

}

void DestroyNewGameImpl(IGameImpl \*\*ppImp)

{

if (\*ppImp!=NULL)

{

delete \*ppImp;

\*ppImp = NULL;

}

}

### 看函数名字就应该能理解函数的功能，分别是由 GameLauncher 调用创建和销毁

游戏。

## **3、逻辑实现：**

公司根据常见的扑克、麻将、跑胡子等类型游戏特点制定了多套业务框架。

**扑克类：**

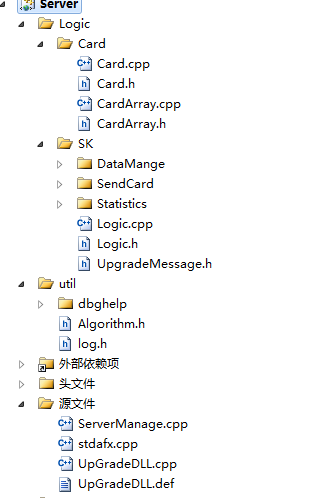
a、以斗地主为例的传统框架：

CUpGradeGameLogic

所有的牌型逻辑全部写在这一个类里面、游戏的主干业务实现的在ServerManage里面

b、以双扣为例的新框架：

代码的路径设计如下:



ServerManage:只负责封装与平台相关的接口并不负责具体的业务实现。

Card :封装的具体的牌型逻辑。

DataManage:封装游戏中的数据

Logic:负责游戏的基本流程

在本例中甚至针对具体的流程作了进一步细分

SendCard:发牌流程

Statistcs:结算

Util：封装游戏中可能会用到的功能函数的

**麻将类：**

麻将是业务相对复杂多变的一类游戏，因此针对不同的游戏和开发人群逐渐演进了2套成熟的开发框架

a、以长沙麻将为例的经典框架：

主要有几个主要的模块

ServerManage:负责封装与平台相关的接口

ThingHandle及其扩展ThingHandleEx:主要负责与客户端的消息处理，与整个游戏的流程

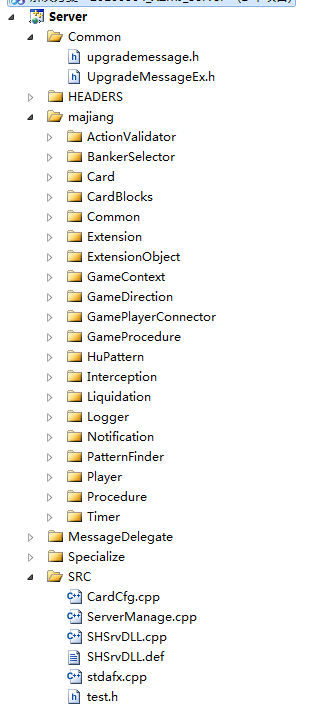
CheckHuPai及其扩展CheckHuPaiEx:主要负责胡牌检测与牌型逻辑算法

CheckCPGAction及其扩展CheckCPGActionEx:主要负责拦牌行为的逻辑判断

GameData及其扩展GameDataEx:封装游戏中的数据

b、以血战麻将为例的新框架：该框架采用扩展对象的设计模式

代码的基本目录结构如图所示：



看似比经典框架代码量增加很多，但是由于其大部分为框架设计，在此基础上新增加的代码量并不多，而且每个新增加的框架很方便按模块进行单元测试，下面简单介绍一下几个主要的模块：

Extension：扩展对象，将每个模块封装成一个对象，并根据GameContext统一管理，比如胡牌检测、定庄、行为监测、玩家管理、日志，配置、结算、消息委托、流程控制器等等。每个扩展类之间只需要作如下处理：

比如要是用ExtensionPlayer



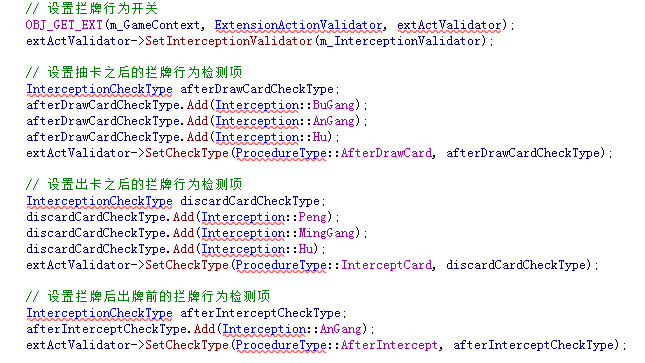


即可获得该扩展对象的指针访问其数据

GameProcedure:游戏过程，比如麻将基本过程有发牌、抓牌、补花、出牌、拦牌、胡牌、结算等流程，每一个流程有单独的一个类负责该过程的逻辑实现，并可根据不同条件跳转到不同过程

ActionValidator：拦牌行为监测，吃、碰、杠、听、胡都有默认实现，如需定制开发，可重写具体行为的拦牌检测类，并且可自行设置每种监测发生在什么游戏过程

例如：



**跑胡子类：**

a、以告胡子为例经典跑胡子框架：

ServerManage:负责所有的业务逻辑

b、以常德跑胡子为例新框架：

ServerManage:负责所有的与平台的接口

PHZGameProc:负责流程控制

PHZChoiceOperate:负责拦牌行为处理

PHZHuCardChecker:胡牌检测

PHZCardComboHuxi:胡息

## **4、服务端和客户端的公共类头文件：**upgrademessage.h

upgrademessage.h包函数了一些基本信息，游戏 ID(NAME\_ID)、游戏名称(GAMENAME)、游戏人数(PLAY\_COUNT)、玩家手牌数、使用总牌数、扑克副数等。还有一些消息数据包、消

息协议。

1. 游戏流程：

服务端和客户端的框架就是如此，关于游戏的基本流程、服务端和客户端的通信都是由平台控制，但一些比较特殊的游戏也可以由游戏自身来控制。游戏规则这一块主要体现在逻辑类中。

了解服务端和客户端的框架后就可以了解游戏的基本流程，并通过流程来介绍相关函数的实现。

启动 G 平台加载游戏 Dll后创建游戏对象，这时就调用 CServerGameDesk 的构造函数，构造函数中初始化数据成员然后进入 InitDeskGameStation针对桌子初始化。开启大厅进放游戏，客户端调用构造函数、Initial 加载编辑器生成的dat文件、InitUI初始化游戏界面，客户端发送消息玩家进入游戏，服务端收到消息后进入 OnGetGameStation 初始化游戏状态，将游戏状态相关数据发送给客户端、客户端通过 HandleGameMessage 接受消息，通过 ASS\_GM\_GAME\_STATION消息协议来确定是状态消息然后调用客户端的 SetGameStation设置客户端的状态，状态设置好后用户点击开始按钮关发送消息至服务端，服务端收到消息后( HandleNotifyMessage)设置玩家的状态为同意游戏，然后每一个玩家点击开始游戏服务端就会统计开始游戏人数，当人数达到游戏基本人数后，服务端会调用IsPlayGame 只有当 IsPlayGame 返回 FALSE 才会开始游戏，所有当 IsPlayGame返回 FALSE 时服务端调用 GameBegin,这时游戏已经开始 IsPlayGame 返回为TRUE，然后就进入游戏自己的流程，当游戏结束后服务端会调用 GameFinish 结束游戏，游戏开发人员可以在此时计算游戏分数并发送数据至客户端，通知客户端显示数据和结束游戏。

五、重要函数：

服务端 OnGetGameStation 和 客户端 SetGameStation 用来获取游戏状态和设置游戏状态，所以又用作断线重联。

服务端 SendGameData 和 客户端 HandleGameMessage 实现服务端和客户端之间的通信。

客户端 m\_pGameFrame->SendGameData 和服务端 HandleNotifyMessage 实现客户端和服务端之间的通信。

客户端 OnUIMessage，这个函数是 界面消息 也就是相当于封装 MFC消息机制。各个控制的单击、双击、鼠标移动等事件都可以在这个函数中捕获。

一个普通的游戏流程就是如此，具体游戏当然要具体分析。如果明白了这些，那么其它流程也一定很容易了解。

