目录

[简介 1](#_Toc464067130)

[第一章 工具 2](#_Toc464067131)

[第一节 工具类 2](#_Toc464067132)

[第二章 入口 3](#_Toc464067133)

[第一节 入口main 3](#_Toc464067134)

[第二节 个人建议 4](#_Toc464067135)

[第三章 数据库系统 5](#_Toc464067136)

[第一节 什么是数据库 5](#_Toc464067137)

[第二节 Plist与二维表 5](#_Toc464067138)

[第四章 管理器 8](#_Toc464067139)

[第一节 数据库管理器 8](#_Toc464067140)

[第二节 语言管理器 8](#_Toc464067141)

[第三节 资源管理器 9](#_Toc464067142)

[一、 声音管理器 9](#_Toc464067143)

[二、 本地资源管理器 9](#_Toc464067144)

[第四节 UI管理器 9](#_Toc464067145)

[第五章 网络 10](#_Toc464067146)

[第六章 平台 10](#_Toc464067147)

[第一节 平台的差异化 10](#_Toc464067148)

[第二节 个人建议 11](#_Toc464067149)

[第七章 SDK 12](#_Toc464067150)

[第一节 SDK平台差异化 12](#_Toc464067151)

[第二节 个人建议 13](#_Toc464067152)

[第八章 UI 14](#_Toc464067153)

[第一节 界面视图 14](#_Toc464067154)

[第二节 界面对象 15](#_Toc464067155)

[第三节 界面管理器 15](#_Toc464067156)

[第四节 界面的工作流程 16](#_Toc464067157)

[一、 大致类图 16](#_Toc464067158)

[二、 大致流程图 17](#_Toc464067159)

[三、 示例demo 17](#_Toc464067160)

# 简介

在这些年的开发过程中，遇到些了不少的问题，发现刚开始如果没有做好前期的准备，写起代码来总会遇到，难扩展，难维护的问题，当然以现在我的个人之力，也不可能能设计出更好的方案去解决一些大家可能在开发中遇到的问题，

所以，希望大家共同探讨，相互学习，力争把项目做得更好，更容易维护与扩展。写此文档目的主要是为了提升自己的个人能力。把项目中存在的问题及可优化的部分记录下来。如果有不对，或者觉得不合理的地方。大家可以及时批出。大家共同探讨，共同进步。力争把我们的游戏做到一个一流的水平。

费话不多说，直接进入正题。

本文档主要是自己总结的一些东西。也是个人对游戏开发的一个理解。

其中会穿插的提及到些与我公司项目相关的一些优化点。

# 工具

## 工具类

做为游戏最常用的部分，不管做什么，首先得有一个可靠及高效的工具，这样做事才会显得得以应手。所以工具在游戏开发中是必不可少的。通常在开发前期会先把各种各样的工具类集中到一个包里，以便大家使用，如数值到字符串的转换，字符串切分，时间转换，游戏中单位转换等，共同的东西没有必要你来写一次我来写一次，当然尽量写全你的注译以方便使用者知道你的接口是用来做什么的，怎么使用，在使用时需要有哪些注意的事项。当然，如果你里面没有太多的坑，注意事项大可不必写出。

这里可能会给大家提出个小小的要求。尽量把觉得像多地方可能会使用到的公共函数放在此包中，不要随心所欲的到处放，这样后面开发的人员可能对项目的结构不熟悉的情况下，又会再一次的实现一个跟你功能一模一样的函数或者是类。无形中拉长了开发时间。

# 入口

## 入口main

程序的入口，个人理解并不是简简单单的main，一个程序的启动。在程序进入入口main之前，全局及静态对象已经被构建。然而我们完全不可预知这些构造函数的调用顺序。在main函数结束返回之后，会调用全局及静态对象的析构，这些过程我们也是无法预知的，目前，我们公司的项目个人觉得还不是很大，所以还不需要考虑到各个系统之的初始加载顺序。

但如果后面的维护与开发中，一些系统的启动必须依赖于另一个系统。到此，可能游戏就没办法正常的动作了。

所以在程序启动时，我们可能会更多的考虑各系统间的依赖关系。按照一定的顺序去启动并初始化她们。

如果你可能会在其它的游戏的启动初始化部分中看到如下代码。

bool CGameProcedure::init(){

//事件系统注册

s\_pEventSystem = new CEventSystem;

//输入管理器注册

s\_pInputSystem = new CInputSystem;

//计时器系统注册

s\_pTimeSystem = new CTimeSystem;

//数据库管理器注册

s\_pDatabaseSystem = new CDataSystem;

//ui管理器注册

s\_pUISystem = new CUISystem;

. . .

s\_pEventSystem-> init();

. . .

//网络的初始化

Net().init();

. . .

//激活游戏主逻辑

}

而目前我们项目的入口就直接的启动了游戏，并未做一些前期工作的准备。

## 个人建议

为了考虑后期系统的复杂度，个人建议先把必要的数据初始化完成，才真正的启动游戏

# 数据库系统

## 什么是数据库

这里说的数据库并不是我们通常所看到的如SQL , MySQL的这种类型的数据库，而是一种读取二维配置表的工具，只有读取的功能。这里也右理解为数据系统，在有些地方，如果现在的富文本，如果采用xml的形式配置可能配置不是很方便。

## Plist与二维表

以下是两种配置的对比。

以xml形式

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<dict>

<array>

<key>1</key>

<dict>

<key>R</key>

<integer>244</integer>

<key>G</key>

<integer>34</integer>

<key>B</key>

<integer>66</integer>

<key>font\_name</key>

<string>NULL</string>

<key>font\_name</key>

<string>NULL</string>

<key>font\_size</key>

<integer>14</integer>

<key>font\_type</key>

<integer>1</integer>

</dict>

<key>2</key>

<dict>

<key>R</key>

<integer>22</integer>

<key>G</key>

<integer>34</integer>

<key>B</key>

<integer>67</integer>

<key>font\_name</key>

<string>NULL</string>

<key>font\_name</key>

<string>NULL</string>

<key>font\_size</key>

<integer>30</integer>

<key>font\_type</key>

<integer>1</integer>

</dict>

</array>

</dict>

</plist>

以表格形式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| INT | INT | INT | INT | STRING | INT | INT |
| #id | R | G | B | font name | font size | font type |
| 1 | 244 | 34 | 66 | NULL | 14 | 1 |
| 2 | 22 | 34 | 67 | NULL | 30 | 1 |
| 3 | 3 | 34 | 68 | NULL | 16 | 1 |
| 4 | 120 | 34 | 69 | NULL | 17 | 1 |
| 5 | 222 | 34 | 70 | NULL | 18 | 1 |
| 6 | 33 | 34 | 71 | NULL | 19 | 1 |

以xml的形式，相信你也看到，上面我只配置了两条数据，可读性已经不是太好了，而且如果以后我想把其中的某个字段删除掉或统一修改里面的值，以xml的形式是不是感觉要花上很多时间，若以二维表方式，我们直接可以用excel进行编辑。方便、快捷.

# 管理器

## 数据库管理器

这里所说的数据库，主要针对本地二维表数据的读取，由于游戏到后期，可能会有大量的数据需要埋在本地，如有些rpg、arpg游戏，他们的道具系统可能会有成千上万种道具，这些道具的一些基本的属性，不可能都从服务器拉取，

如他们的物品的名称，描述，需求的等级等属性，这些属性如果全部从服务器拉取，无非增加了服务器的压力。

记得之前做过的一个项目，刚开始是为了同步方便，把所有的道具的属性表都从服务器摘取，不过到后面，由于道具多达3000多种，在有些地方，如，玩家出售一个未经过任何加工的道具，也就是这个道具只有最基本的属性，此时，如果你客户端已经有了这个道具的基本属性，只需要服务器给你这个道具的模板id，你就可以找到这个道具的所有基本属性承现给玩家，相反，如果客户端没有这些基本的数据，所以这个道具的属性都来源于服务器。不用我多说，你也知道他的数据量。

所以在客户端就应该提供这样的一个管理器，来针对客户端的一个数据配置表的管理。

## 语言管理器

游戏开发中，可能避免不了为了以后的扩展，会有多语言的支持，语言管理器就是为了实现读取相关的语言包。以不同的语言方式承现给用户。

这个地方，目前项目我感觉应该很不错了。相比之前做过的项目，之前做的项目由于只考虑到中文，所以可能写代码时为了方便，把文字直接写死在代码中，后面带来了一些修改维护上的困难。

## 资源管理器

这里所说的资源管理器，主要包括游戏资源在执行期的装载与卸载.。如声音、纹理、动画、音频片段、关卡布局、骷髅动画等。

一个游戏正常启动，依赖资源的加载完成，否则可能达不到预期的显示效果。

### 声音管理器

见名思意，主要是针对游戏的各音效与音乐的装载与卸载。播放或停止指定音效或音乐等，这里就不详细写出。

### 本地资源管理器

主要负责本地的游戏纹理，动画，骷髅动画的游戏资源的加载。针对资源的加载管理，以个人的经验提出以下几点：

* 按纹理从大到小加载
* 不要把全部的游戏资源一次性加载
* 活动或高级用户才能使用的资源尽量采用预加载资源方式
* 在某些平台收到内存警告时，一定要清楚未使用的缓存资源，如纹理，帧动画等

## UI管理器

这一节由于跟UI紧密的联系，请翻阅[第八章](#_UI)。

# 网络

略

# 平台

## 平台的差异化

目前我们处理的主要有三个平台，ios、android、windows三个平台，各平台在某些如输入框、复制文本、判断网络、打开应用前商城、打开本地相册等存在各平台的不同处理。

针对这些平台的差异化，上层应用开发人员尽量不要去涉及到平台的东西。我们更应该抽象出接口。去处理各平台之间的差异化。

尽量不要写出如下代码：

#if ( CC\_TARGET\_PLATFORM == CC\_PLATFORM\_IOS )

do ios handle

#elif ( CC\_TARGET\_PLATFORM == CC\_PLATFORM\_ANDROID )

do android handle

#elif ( CC\_TARGET\_PLATFORM == CC\_PLATFORM\_WIN32 )

do win32 handle

#else

do other handle

#endif

我们应该抽像出来、像这样

PlatformHelper::getInstance()->doHandle();

由各平台去各自实现。

## 个人建议

尽量做到上层开发人员不要考虑平台差异化的问题。如果存在平台差异化的地方，尽量封装，以接口方式提供给上层应用开发者。

# SDK

## SDK平台差异化

随着现在的游戏行业发，不接入第三方的sdk是否已经不太现实，而现在第三个的sdk如此的种类繁多。如百度、QQ、微信、91助手、皮皮助手、facebook等。当前我们公司只用了一种sdk facebook。

而这些sdk的登录、登出、是否登录等一系列的api可能存在有各平台的差异。国内多数sdk平台大多都已经集成了第三方的支付渠道。他们的支付接口也是千奇百怪。难道我们要在我们上层的界面代码中写出如下形式的代码：

#ifdef \_\_BaiDu\_\_

do baidu login

#endif

#ifdef \_\_Facebook\_\_

do facebook login

#endif

当然我们上层的开发人员可能会很不高兴写这样子的代码。

我们更倾向于写

SDKHelper::getInstance()->login();

来到这里，可能有会有人问，如果我游戏接口了多个SDK，你的SDKHelper又如何的动作。当然这个地方涉及到一个小技巧。SDKHelper只提供接口。由各自的SDK去实现。你的界面上应该有相应的按钮进行登录。如果用户点击了使用QQ登录的按钮，此时的调用变成了QQSDKHelper::getInstance()->login()。如果用户点击了使用Facebook登录按钮，此时的调用变成了FacebookSDKHelper::getInstance()->login()。这样不就区分出来了吗？

## 个人建议

虽然我们目前项目只有一个平台的接入，但还是建议把他们在封装起来，以满足后期如果要接入其它sdk时，不必要去改上层已经写好代码。只需要替换掉对应的 sdk实现即可。

# UI

单独把ui拿出来，个人感觉现在项目的ui部分还有可优化的点，这章我会着中的介绍一下自己对ui的见解。也希望能跟大家一起探讨、相互学习、共同进步。这章会给大家介绍一下我认为的ui大概的组成部分，他们之间如何的协同运作。

## 界面视图

界面视图是什么，个人理解为提供显示给用户的容器。在软件设计mvc框架中你可以理解为V视图。

当然这里不准备说mvc框架。而是感觉项目中的界面可能承载的功能过多。希望做些优化，数据与界面的分离。

那么，界面视图应该包含哪些功能呢？个人理解应该包含有如下功能：

* 显示与隐藏
* 是否显示与隐藏结束
* 是否接收键盘事件，如手机的返回按钮
* 接收用户事件（如点击）即处理用户事件,将用户事件移交给界面对象进行处理
* 接收游戏事件

## 界面对象

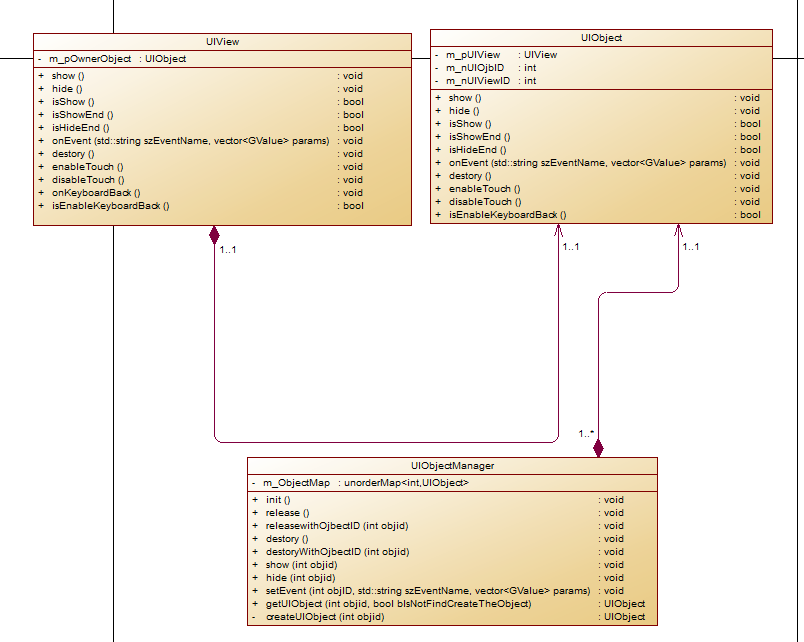
什么是界面对象？界面对象是指拥有能控制界面视图的显示与隐藏、界面视图的销毁与创建。向界面视图派发数据模型改变的事件的容器

## 界面管理器

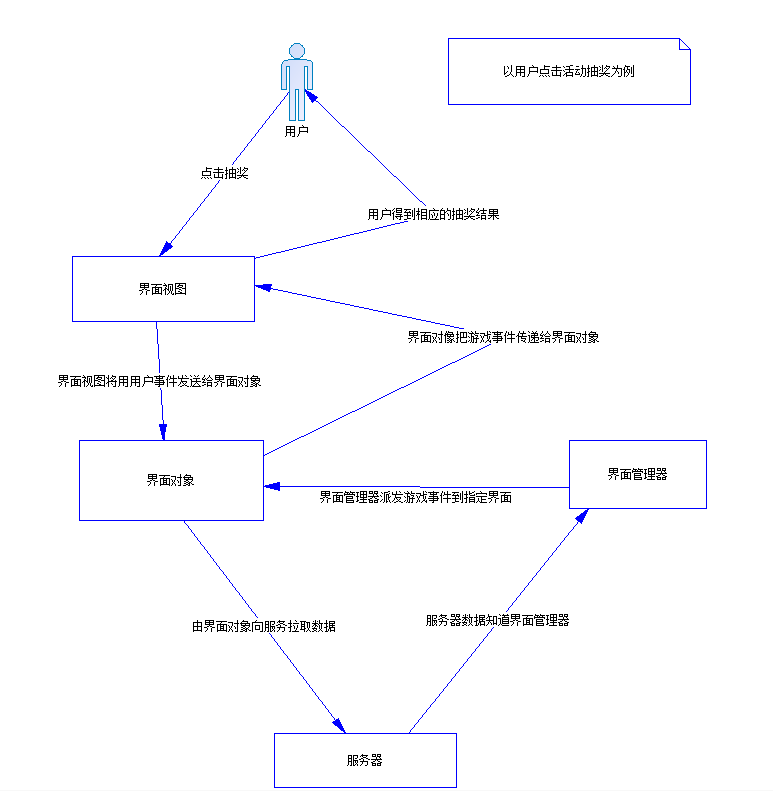
什么是界面管理器？界面管理器是指拥有所有界面对象所有权。像界面对象派发游戏事件。管理所有界面对象的显示与隐藏、销毁与创建的容器。

## 界面的工作流程

### 大致类图



### 大致流程图



### 示例demo

请观看与文档同级示例工程。

这里说明下，可能这个模式不是严格的mvc，界面对象充当着控件器跟数据模型的两个角色。即用户的数据只保存在界面对象中，界面视图只是充当一个纯粹的显示功能。