# 说明文档

## 工具介绍

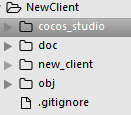
引擎采用cocos2dx-3.13，UI布局采用 cocos stdio2.3.2。

### 游戏流程图

查看当说明目录下的 “大厅流程图.vsd”

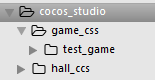
## 目录介绍

### 1级目录结构

，

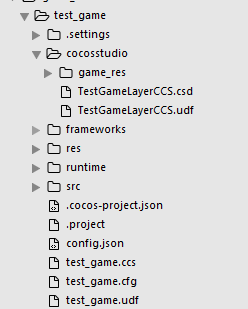
其中cocos\_studio用于存放 UI布局的cocos stdio工程。Doc为文档目录，new\_client为客户端目录

### cocos\_studio目录介绍



其中hall\_css为大厅，game\_css为存放每个子游戏目录，与test\_game并列，具体实现可以此为参考

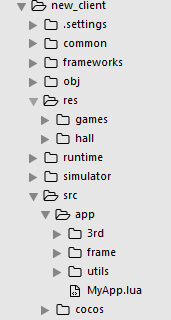
#### test\_game介绍

，

其中，所有ui美术资源放于 game\_res下，UI布局工程放于cocosstudio下，命名为单词小写以 下划线 分割

### new\_client目录介绍

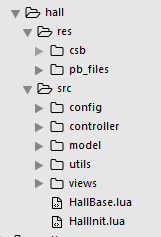
结构体为：



Common用于存放pb协议文件，方便查看各个协议定义。

#### res/hall介绍

为存放大厅相关资源文件夹（其中src为存放源码目录，res为存放美术资源目录），具体结构为

，其中

##### res目录：

res/ccs\_export：大厅cocosstudio到处资源目录。

res/pb\_files:大厅PB协议生成的二进制文件存放目录

##### src目录：

src/config:大厅常量文件存放目录。

src/controller:大厅管理器存放目录。

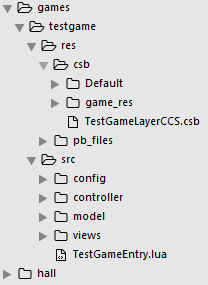
src/model:数据对象存放处

src/utils:大厅工具寸类型存放目录

src/views:存放前端界面目录

#### res/game为存放子游戏的根目录。每一个子游戏都存放在该目录下。

子游戏目录为



##### res目录：

res/ ccs\_export：cocosstduio工程到处的资源目录。

~~res/pb\_files：该子游戏所有PB协议二进制文件存放目录~~

##### src目录：

src/config：游戏配置文件存放目录

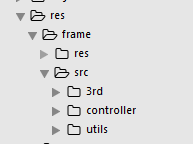
src/controller:游戏控制器存放目录

src/model:存放数据对象目录

src/views:存放前端界面目录

#### res/frame目录

用于底层框架目录 结构目录为：



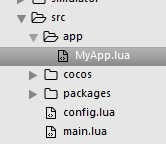
其中：

src//3d:用于存放第三方库

src/utils：框架自定义工具目录

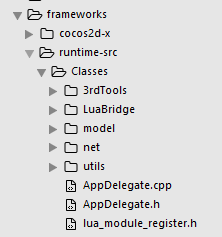
src/controller：存放控制器相关

#### src结构



项目工程基本配置在config.lua中设置

#### C++目录结构介绍

，其中

3rdTools:用于存放c++第三方库。

LuaBridge：存放导出lua的c++文件

net：存放网络层相关文件。

utils:存放自定义c++工具类文件

## 底层框架类介绍

### 主要类介绍

GameManager.lua:游戏总控制器，在MyApp.lua初始化时构造

属性成员：

self.networkManager:网络管理器实例对象。在GameManager.构造时启动。

self.msgPBParseManager:PB消息编码解码管理对象。

self.hallManager,在GameManager:callbackWhenHallVersionDownloadFinished()函数中实例化

NetworkManager.lua:lua网络层管理器。主要负责lua层调用c++相关网络接口

MsgPBParseManager.lua:PB消息解码编码管理器。

DownloaderManager.lua：lua层下载管理器，下载功能暂未实现。

CustomHelper.lua、框架工具类

## 大厅结构类介绍

### config目录

HallConfig.lua大厅游戏配置文件

HallClientDesc.lua 大厅提示文本配置文件

### controller目录

HallDataManager.lua:大厅数据管理对象

HallManager.lua：大厅管理器

HallMsgManager.lua:大厅消息管理器，主要负责游戏里面消息发送与接收处理

SceneController.lua：场景切换控制器

### model目录

PlayerInfo.lua：登录玩家数据对象类

### view目录

CustomBaseView.lua。所有ui界面父类

HallScene.lua：大厅场景

HallSceneUILayer.lua：大厅UILayer

IndictationLayer.lua：风火轮层

LaunchScene.lua：游戏启动场景，默认在hall目录下，方便热更

LoginScene.lua：大厅登录场景

SecondarySelectLayer.lua：大厅游戏列表二级界面

TipLayer.lua：游戏弹出提示框层

## 子游戏类介绍(TestGame为例)

### config目录

TestGameTipsDesc.lua：该游戏所有提示文字配置文件

### controller目录

TestGameDataManager.lua：用于管理该游戏数据对象

TestGameManager.lua：该子游戏总管理器，在大厅进入游戏房间成功创建

### views目录

TestGameScene.lua：子游戏主场景

##### 注意事项

TestGameEntry.lua 需要实现相关函数

用于返回进入场景需要加载资源数组

function TestGameEntry:getNeedPreloadResArray()

function TestGameEntry:getStartScene()

# TCP消息的发送和接收

## 协议消息注册

### Pb二进制文件注册

调用HallMsgManager中的方法：HallMsgManager:registerProtoFileToPb(filePath)，具体调用为：GameManager:getInstance():getHallManager():getHallMsgManager():registerProtoFileToPb(filePath)

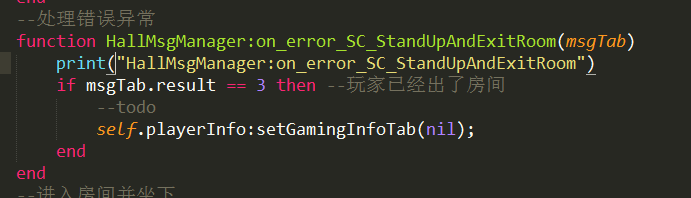
### 消息名注册

在注册PB文文件后，需要继续注册每个消息协议中对应协议，方便发送消息的编码，以及接收消息时候解码。调用方法：HallMsgManager:registerMsgNameToMsgPBMananager(msgName)，msgName:为具体的消息名称。

### 回调函数注册

在客户端接受到tcp消息后，

若要在HallMsgManager特殊处理该消息成功时返回的消息，只需要按规则实现实现函数即可，实现规则”on\_消息名”,如” HallMsgManager:on\_LC\_Login(msgID,msgTab)”为处理登录成功消息。

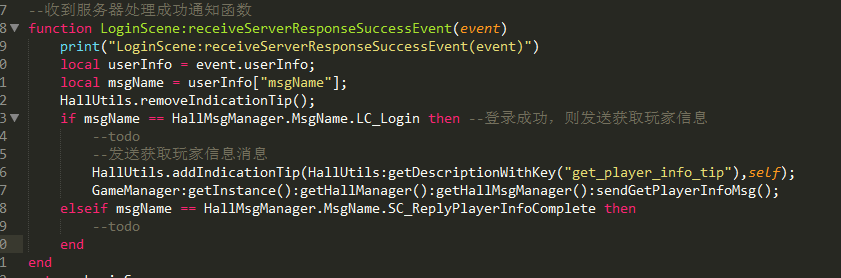
若要在底层处理相关错误消息，只需要按规则实现函数可，实现方式为 “on\_error\_消息名”，如：

## UI界面消息处理

底层每接收到一个消息，都会发出对应消息通知,具体情况如下：

### 成功

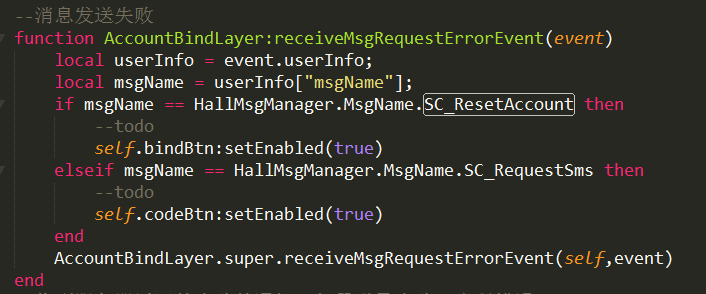
会发出HallMsgManager.kNotifyName\_MsgResponseSuccess通知，只需要在对应UI界面（继承CustomBaseView）中实现：receiveServerResponseSuccessEvent(event)即可，如登录界面中：



### 失败

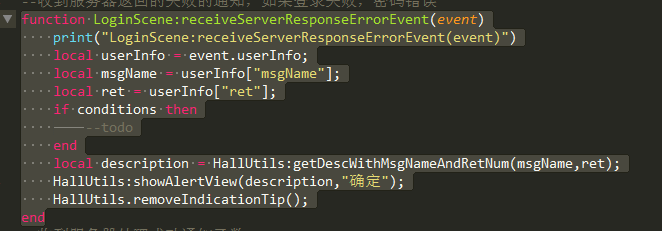
### 网络请求失败时

会发出HallMsgManager.kNotifyName\_ConnectionClose通知，只需要在对应UI界面（继承CustomBaseView）中实现：receiveMsgRequestErrorEvent(event)即可，如：



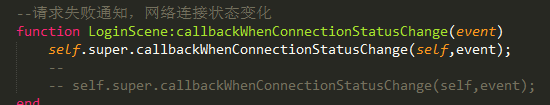
#### 2、服务器返回错误时

会发出HallMsgManager.kNotifyName\_MsgResponseError通知，只需要在对应UI界面（继承CustomBaseView）中实现：receiveServerResponseErrorEvent即可，如：



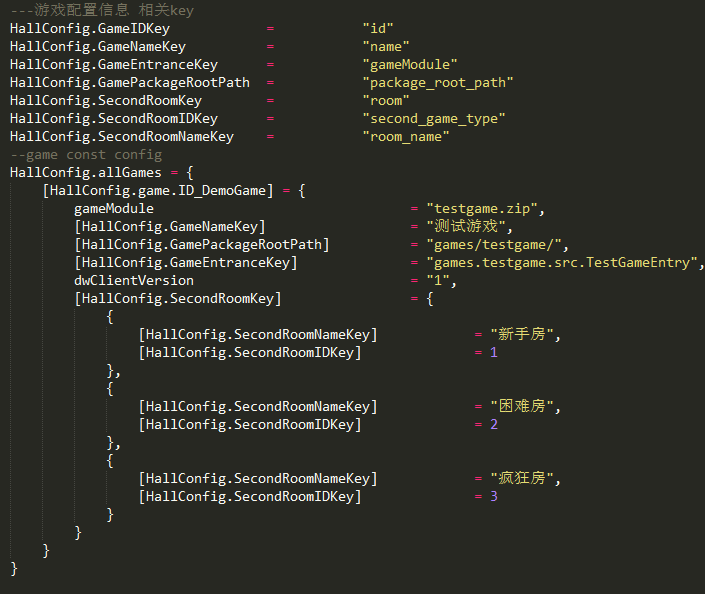
## UI网络状态变化消息

只需要实现callbackWhenConnectionStatusChange(event),一般这个不用实现，已在CustomBaseView处理。



# 子游戏实现

## 在大厅配置HallGameConfig中实现，该子游戏基本配置，如



## SubGameManger类(继承与基类：SubGameBaseManager)

在游戏退出时候，需要调用 SubGameManager: onExit()方法

## 获取玩家基本信息

local playerInfo = GameManager:getInstance():getHallManager():getPlayerInfo();

## 场景相关

子游戏场景的基类为：CustomBaseScene

需要实现一下方法，用来实现场景预加载-

--返回场景需要预加载的资源数组

function CustomBaseScene.getNeedPreloadResArray()

return {};

end

# 命名规范

## C++命名

文件夹：单词小写，多个单词以”-”分隔

类名:以首字母大写开头，多个单词之间直接相连，首字母大写。

成员变量：私有变量以 “\_” 开头

方法：首字母小写，其他单词首字母大写开头。

## Lua命名

文件名尽量与类名（table名）保持一致，方便查看

类名：首字母大写，多个单词首字母大写，直接相连。

函数名：首字母小写，其他单词首字母大写开头。（注：消息处理函数有固定格式，不在此范围之内，消息函数格式为 “on\_”+ 消息名）。

属性：

公有变量：小写字母开头，其余单词首字母大写开头，外部可直接访问

私有变量：下划线“\_”开头，第一个单词小写开头，其他单词首字母大写，外部不能直接用调用，通过方法访问

## Cocos Studio工程命名

1. 子游戏cocos studio命名：

工程文件夹：尽量以单词命名（小写），以”\_game”结尾

工程名：尽量与文件夹名保持一致

1. 美术资源文件夹：均以“game\_res”命令，方便底层统一处理。
2. 布局界面层文件：多个单词首字母大写开头。

# 重要说明

## 1、requireForGameLuaFile函数

所有创建的lua文件 require方法不能直接使用，应用以下全局方法

---重写require函数，用于在游戏热更新后，重新热加载

function requireForGameLuaFile(file)

## 2、子游戏维护开关提示框

1、消息类型的监听HallMsgManager.MsgName.SC\_GameMaintain，该消息已经在父类中，registerNotification中监听。

2、消息的处理

SubGameBaseScene::SubGameBaseScene:receiveServerResponseErrorEvent中已经处理了服务器返回的对应错误处理。

各个游戏需要的处理

在子游戏场景中receiveServerResponseSuccessEvent消息增加HallMsgManager.MsgName.SC\_GameMaintain的处理，分两种情况

A:如当前玩家未在游戏中，则直接弹出游戏维护中的提示框。调用方法：self:alertAlertViewWhenServerMaintain();

B:如玩家当前正在游戏局中，则在玩家该局游戏结束时，弹出提示框。

self: checkIsNeedAlertGameMaintainTipView (),用于检测是否需要弹出维护提示框

SubGameBaseScene:isContinueGameConditions()：用于判断当前是否能够进行下一局游戏，若游戏维护开关打开，该方法会弹出维护提示框