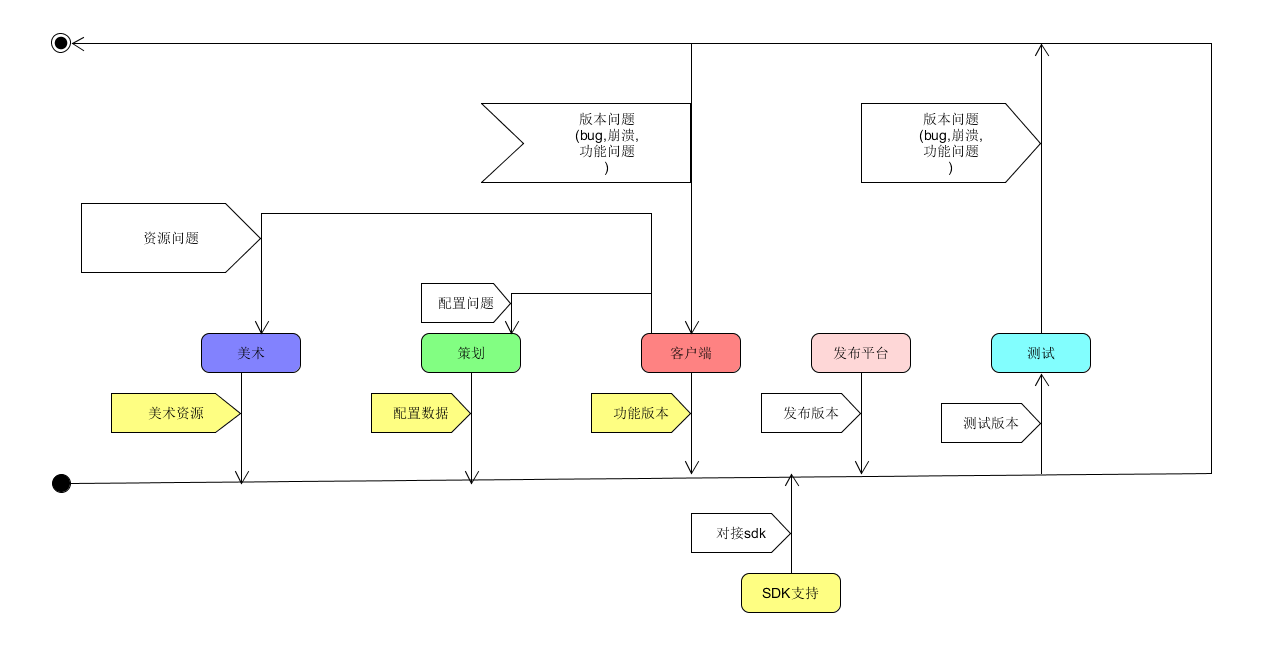
文档编写：胡鹏

邮箱：395841903@qq.com

项目工作流总览,



1. 客户端程序职能
2. 设计客户端架构
3. 约束美术资源制作规格和美术资源存放结构
4. 提供美术各资源规格检查工具
5. 提供策划各种制作工具和配置相关检测工具
6. 制作版本功能，
7. 优化客户端性能和解决程序bug
8. 美术职能
9. 按制作规格规范制作和资源存放结构，制作美术相关资源
10. 使用工具检查工具是否合乎标准
11. 协作程序优化影响程序性能的美术资源
12. 策划职能
13. 使用工具或配置表制作游戏内容
14. 使用自检工具检查游戏内容是否合乎规格
15. SDK支撑组职能
16. 按规格对接SDK，部署SDK
17. 解决游戏和外部SDK的一些问题
18. 发布平台

1.发布游戏版本

1. 测试职能
2. 测试游戏版本，反馈bug，反馈Crash,反馈游戏体验不佳问题
3. 将测试通过的版本对提交个运营准备对外发布

美术篇

1. 美术资源制作规格

1.1场景相关（项目运行测试中得出）

场景面数裸跑： 10W以下

DC裸跑：80以下

关于场景的相关优化：

参考：<http://blog.sina.com.cn/s/blog_471132920101gbfr.html>

全民仙逆的游戏的做法是(这样做会减少场景打包后包体的大小)

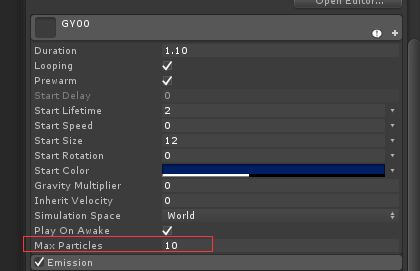
代码中使用UnityEngine.StaticBatchingUtility实现(可以在任何平台调用):

1. 将所有要合并的静态物体(不须勾Static)放入统一一个root
2. StaticBatchingUtility.Combine(root);

1.2 粒子特效（项目运行测试中得出）

DC：10以下

同时喷发的粒子数<=10



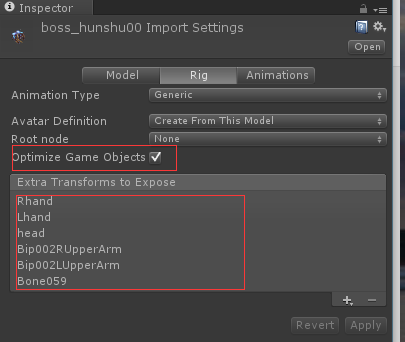
1.3 模型(FBX)

主角：3000面

贴图：512：1张(diffuse) 256:2张（mask）

1.4动画相关

1.FBX导入优化（项目中使用）

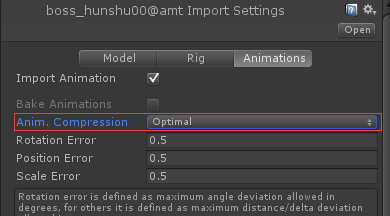


勾选Optimize Game Objects 减少蒙皮的计算量（Unity4.3版本推出的功能）

如果有特殊的挂点不想优化 在下方添加该节点

参考: https://blog.uwa4d.com/archives/QA\_Animator-1.html

2.动作文件导入优化（项目中使用）



Anim.compression 选择optimal

参考：https://blog.uwa4d.com/archives/Loading\_AnimationClip.html

3.降低动画片段精度（这个优化很明显，项目中未使用）

参考：<https://blog.uwa4d.com/archives/Optimization_Animation.html>

1.5:贴图格式

参考：<http://mp.weixin.qq.com/s/JKqq6TEFrZN1pB-avFnmeg>

给出一个讲解资源很全面的地址，直接看就好了。项目中使用也是以内存占用和不失表现效果为依据修改贴图格式的

1.6.UI资源

确定好UI公共图集，避免冗余的资源

规格 1024\*1024 一张图集，

超过256大小，做散图处理,

不允许开启**Mipmap**选项

各个平台给出不同的压缩参数，

UI 预设资源

1. prefab 不允许依赖超过3张图集

二、资源部署结构

XNRes

|--Arts(制作资源原始目录)

|--Ref(依赖相关的一些资源)

|-----Effect(特效资源制作目录) |

|--AssetBundle(AB打包资源)

|---Altas(图集目录,单图集为目录存储)

|---Font(字体目录,单字体为目录存储)

|---Mats(材质目录,特殊的一些效果材质,一般不需要单独创建材质)

|---Prefabs(预设目录,)

|---audioclips(音效目录,可以放置在Assetbundle这层)

|---Effect(特效目录,一般为UI特效)

|---UI(UI预设目录,以模块为单位存储)

|---SceneMap(打包场景目录)

|---SceneObj(原来准备动态构建场景使用，现在无用)

|---Textures(UI散图目录,超过256大小的图,)

|--Res(美术外包目录,原因是方便同步外包的美术资源)

|---Arts(制作资源原始目录)

|---Level(制作的场景资源,单场景为目录存储)

|-- Ref(依赖相关的一些资源)

|---Effects(特效资源制作目录,一般是主角，技能等相关特效)

|---FBX(模型资源制作目录,单模型为目录存储)

|---Scenes(场景资源制作目录)

|---AssetBundle(AB打包目录)

|---Effects(特效目录,Ref/Effects 制作完成后的产物)

|---Roles(角色相关目录,Ref/FBX 制作后的产物)

|---Weather(天气相关预设)

部署结构和打包策略是配套的，后续会在程序篇中介绍不同目录的打包策略

三、美术相关的工具

1.制作武器预设

功能：将武器和需要的特效按制作规则合并后制作预设输出到指定目录

2.模型编辑器

功能：

1.切割模型动作，按规制作动画状态机

2.配置不同使用场景的模型预设目前有（剧情，UI展示,场景实体等）

3.配置不同表现形式的外观.(时装,怪物)

4.将模型规则和特效按照规则合并后制作预设输出到指定目录

3.选择特效输出到指定目录

1.按规则将特效拷贝到输出目录

提供工具

1.是为了让美术产生可使用的资源。

规划结构

1.美术在多人协作制作只有容易有冲突

2.我们只需要关心打包的AB的目录

策划篇

项目权限限制不允许策划拥有代码相关的权限,项目在协作上是区分了程序工程和策划工程2个工程，策划工程中的程序相关部分是需要程序利用工具同步到策划那边,他们才拥有最新的版本内容

1.策划版本功能工程路径:在项目上级目录下XNGameSync

拥有不同功能的bat文件，分别是不同级别的处理，推荐使用syncAll.bat.

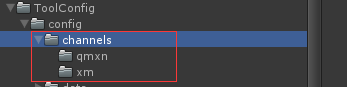
处理完成后提交svn,.通知策划更新

2.策划工具相关

1.将配置表格导出为lua 和服务器Sql,也可以导出成不同的类型，只需要添加不同语种的导出器就好了.

2.导出时使用渠道特有的表格（策划Excel在多版本合并时容易有冲突,提供了此功能，版本同步时只需同步Excel,然后将表格数据导出一遍就好）

部署的结构为，在配置表目录下的channels添加不同的渠道名文件夹，放入渠道不同其他渠道的配置表.顺便在工具代码里添加一个选项(是可以读配置的，但是我懒^\_^~)



3.场景编辑类

1.路点工具(目前游戏里不想贴边走,所以加入了路点工具,游戏目前的寻路方式为A\*+导航网格)

2.遮罩工具(给一些需要透视的物件添加标记)

3.刷怪点工具(这个有些low.可以考虑做的更好一些)

4.区域工具(游戏进出一些区域提示)

5.空气墙工具(游戏区域限制)

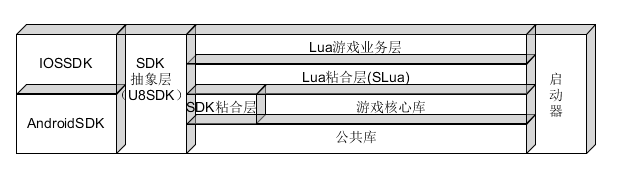
6.场景导出工具(将美术制作场景导出到指定目录)

7.UI图集检查工具(主要是检查一些图集引用的情况)

以上介绍的是目前常用的一些工具，之后尽可能的将一些约束制作为工具。减少人工引起的错误

客户端程序篇

客户端结构总览.



从右至左来 分为,启动器层,游戏层,SDK层 三部分，各层间以接口交互,各层内互不可见，不互相侵蚀

游戏采用Lua

1.为了更新，目前android是（DLL+Lua） Ios(Lua),

2.提供游戏编译速度，

1. 启动器主要功能

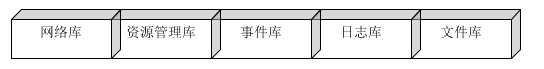
初始化游戏运行环境

1. 小包,为第一次启动做资源初始化工作
2. 小版本更新,下载更新包 更新游戏到最先版本
3. 大版本更新,android 下载apk 并按照，IOS 跳转到AppStore
4. 启动游戏(android 加载DLL启动游戏)
5. 游戏层

游戏业务和一些主玩法

游戏层次从下至上 有区分为 公共层，游戏核心层(游戏核心库和SDK粘合层) ,Lua粘合层，Lua游戏业务层

* 1. 公共层



* + 1. 公共层由 网络库,资源管理库,事件库,日志库,文件库,组成
       1. 网络库 Tcp，Http(BestHttp插件),

TCP相关只是实现了一层封装，目前支持IPV6和IPV4，使用IPV6记得是域名而不是IP，

* + - 1. 事件库,主要是很常见的,引入主要是为了减少模块间的耦合
      2. 日志库

输出日志信息到文件，采集异常信息,发送都异常统计后台,供修正游戏

目前没法收集Ios 原生层的崩溃信息，有一些第三方插件能做这些事情,但是之后肯定是要实现一个完整的，主要是为了定制,另一方面是为了采集游戏在生产环节下的一些游戏运行的效率参数,做持续调优（这个之后会衍生出一个新的库(监视库)）

* + - 1. 文件库

封装一些常用的文件读写,已经为Lua脚本定制了一个文件系统,目的是减少IO,加快游戏载入速度,(说白了就是把所有文件合成一个,利用seek来获取文件的字节流)

* + - 1. 资源管理库(AssetBundle管理)

共分为3层 ab缓存池,ab加载器,ab生命周期管理

1.ab缓存池

主要维护已经加载完成的资源

2.ab加载器

加载ab 这里的加载是有策略，优先加载外部的ab资源，没有才加载内部的随包资源

3.ab生命周期管理

主要是周期性检查不在使用的ab资源并卸载.其次是周期性回收资源(目前的依据是使用时间检查)

关于AB这里不做太多说明，给一个网址:

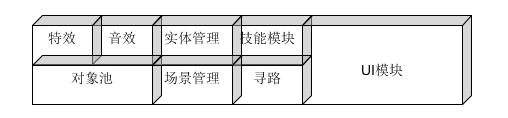
<https://unity3d.com/cn/learn/tutorials/s/best-practices> 这里面的一些文章都要好好看,对AB的解释很精辟,如果有时间多看看官方文档,有很多惊喜和彩蛋

对于AB 官方也有例子demo,网址

<https://bitbucket.org/Unity-Technologies/assetbundledemo>

多留意一下API的迭代，如果有时间多看看unity3d新版本的特性，有很多问题有时候其实升级个版本就能解决，但是不要在现有项目上做

* + - 1. 游戏核心层



游戏核心层,设计思路是为lua层提供一些服务，由lua驱动.这个结构包含对象池,UI模块,寻路，场景管理，实体管理，技能模块,特效模块，音效模块

1.对象池

建立在资源管理上的一层封装,隐藏资源加载细节，为上次提供一些更抽象的接口.

在管理上有 使用时间管理类型，场景管理类型，全局管理

按照类别使用，特效对象池,UI对象池，模型对象池，不同类别的缓存次在缓存策略上是有不同的要求的，其实主要的问题是减少instance 和destroy次数，以及active的CPU消耗

2.UI模块

一些自定义的UI控件,UGUI自有的控件封装，图集管理.

1. UI控件

目前定义的有 循环List, UI模型展示控件

1. UGUI控件的封装，

是为Lua层服务,也容易管理一些释放相关的东西.

1. 图集管理

主要是一些ICON类和散图 图片的加载,和释放提供一层封装，类似NGUI的 图集管理

3.场景管理

主要是场景切换相关的操作,和一些场景相关的出来，比如物件的透视效果，空气墙等

4.寻路模块

这一层是自己写的导航网格,没有采用Unity3d的.也加入了A\*

目前对于大范围的寻路是现采用A\*查找出路径(为了不贴边走路),局部小范围是采用导航网格寻路.这个是老王写的很溜,和服务器采用的是一套,建议有空多看看.

5.实体管理

有各种实体类型，这里不做列举，只是说下这里设计有些偏差，应该是提供一个实体的原型，一些实际的处理交给Lua去控制，游戏里的实体设计过重，导致在扩展的时候在IOS上更新起来不方便.

6.技能模块

这边的设计是对技能的规则做了一些支撑。详细的发起是由Lua层组织的，所以在做一些新技能的时候更新方便

7.特效模块

基于对象池上对特效的使用做了一层维护和封装,也有不同的特效类型,这层的代码相对简单，也只是提供业务支持，由其他模块调用

8.音效模块(这个模块如果稳定后可以放到公共库中去)

对于所有的音效的管理，维护音效的播放，和销毁。

* 1. Lua粘合层

这里使用的是SLua 插件,主要是代码结构清晰，缺点是，很多Lua的插件需要自己去集成有些复杂，

关于其他用的比较多的是Xlua(腾讯爸爸的产品,也还可以,如果主要用C#开发,又要考虑更新.推荐用这个,) tolua(全家桶,很多公司都在使用没啥大问题)

* 1. Lua游戏业务层

游戏主要的业务都在这一层中完成，也是大部分程序开发接触最多的一层.

Lua业务层层次结构如下.



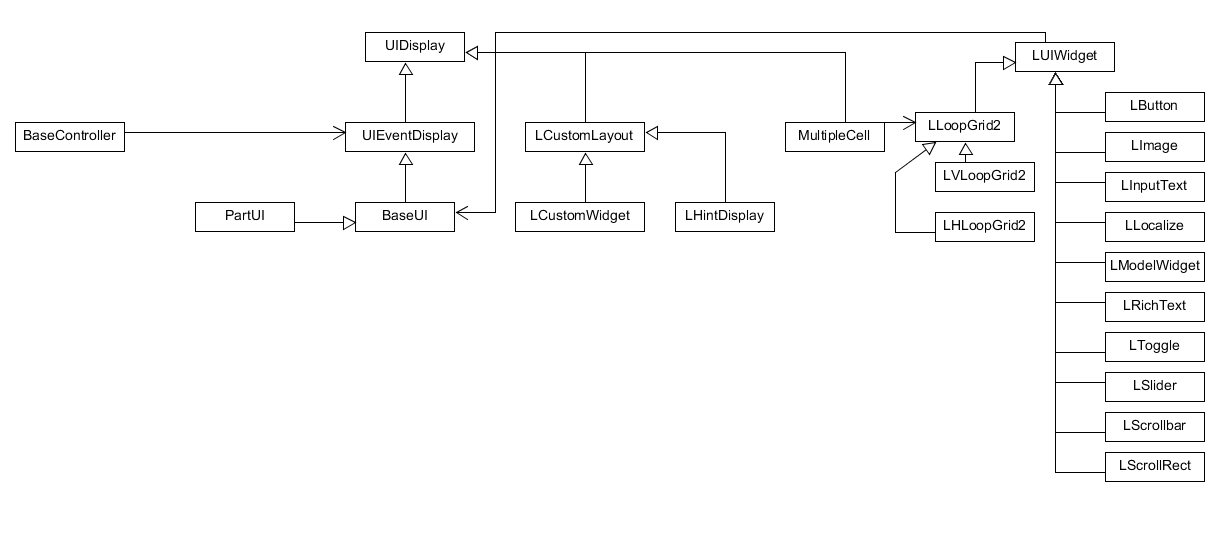
这里大致分3部分，UI 层,LuaWarp 层，lua业务

1. LuaWrap 层

主要的用途是封装所有由C#层次导出的接口,最大可能不让其渗透到lua业务中,让业务人员使用一些封装的业务和原来的lua调用接口相同,没有二义性

1. UI层

UI的结构类图,和模块设计图



1.UIDisplay

Build(uiObj, class, parent) 构建UI的主入口,

1. 关联UI根节点
2. 绑定布局
3. 初始化布局
4. 显示UI

\_bindControl(self) 控件绑定,

绑定规则 self.控件名 =”控件枚举名”, 控件名为预设上的GameObject名称,控件枚举名为UIWidgetEnum 或UIStencilEnum中定义的值,

所有被绑定的控件都会被存入到\_usePool中标记,在销毁是自动释放

\_bindEvent(self, class) 事件绑定

绑定规则 控件名\_控件事件\_event 控件名为预设上的GameObject名称，控件事件现在又,click,dowm,enter,exit,up,Repeat,等,拖曳相关的事件被单独抽出去实现,原因是和点击相关的事件会有冲突.所有拥有事件的控件也会被存入到\_usePool中标记

destroy() UI销毁入口

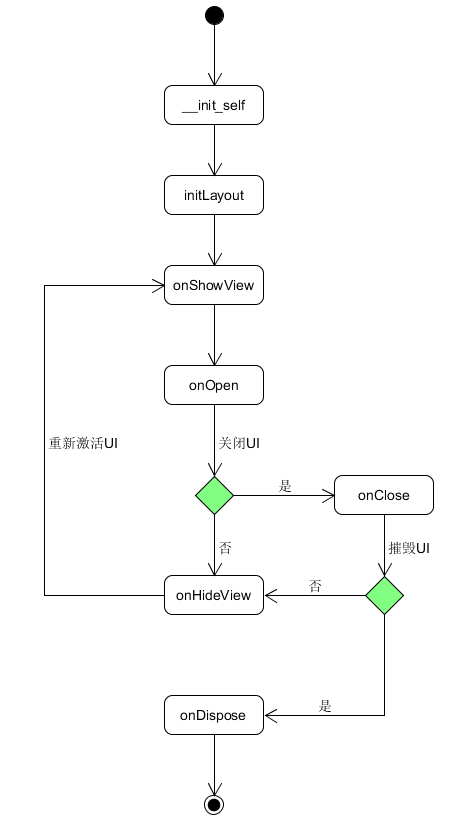
1. 隐藏UI
2. 销毁子UI相关
3. UI自动布局控件解引用
4. 销毁UI

releaseRef() UI自动布局控件解引用

通过自动绑定的控件在此进行lua 脚本 和UI的引用解除

控件的自动绑定和控件的自动销毁是UIDisplay 中关注的重点,

UI的生命周期,



\_\_init 这个方法每个UI都需要定义,但是可以不做任何实现

\_\_init\_self 定义所需要控制的控件.整个生命周期只执行一次

initLayout 初始化布局上的一些操作,只和UI有关,整个生命周期只执行一次

onOpen UI打开的时候需要执行的一些操作,每一次触发Open.都会执行此方法

onShowView UI显示的时候需要执行的操作,每次Active(True) 都会执行一次

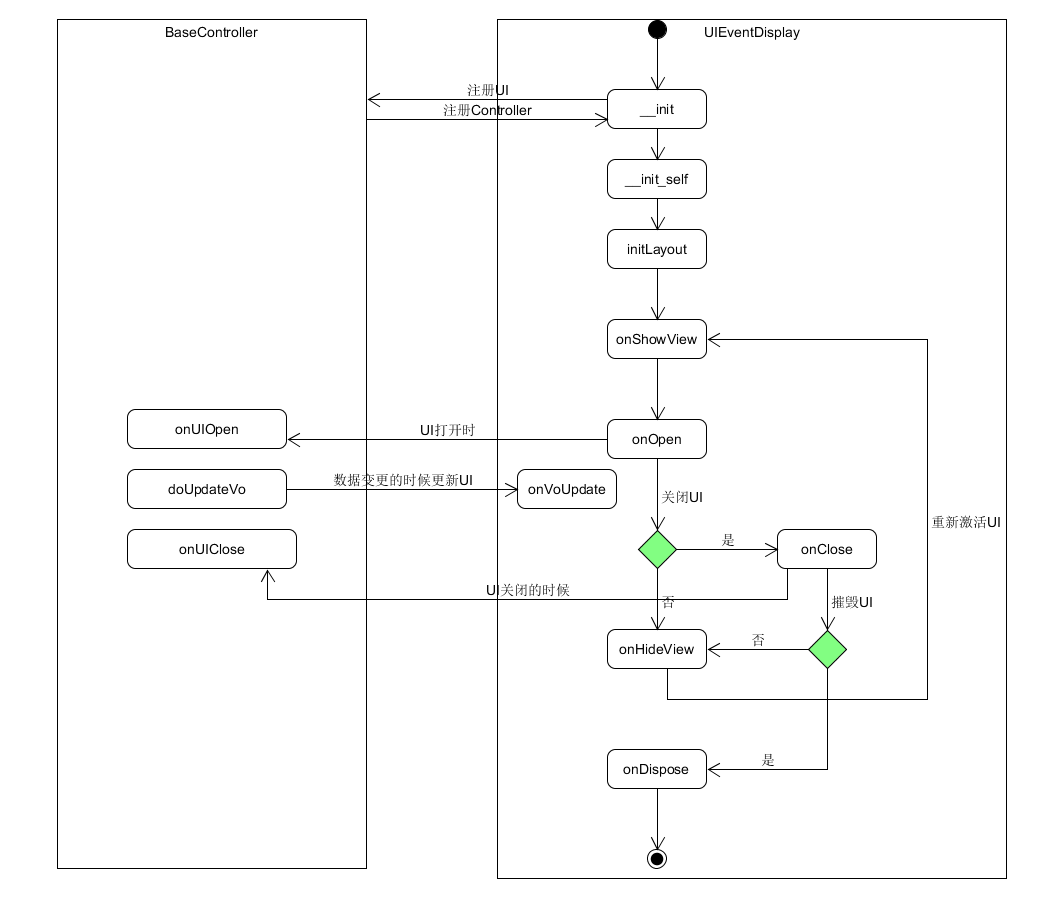
onHideView UI隐藏的时候需要执行的操作,每次Active(False)都会执行的操作

onClose UI关闭的时候执行的操作,每次UI关闭都会执行此方法

onDispose UI销毁的时候调用此方法，如果有申请一些资源，需要在此释放掉.

2.UIEventDisplay 和 BaseController

这2个类定义了UI 和 业务模块交互的操作，大部分的模块都是这样实现的，核心思想是由逻辑业务影响UI的显示,在创建UI的时候这会将这2个类相互绑定.,那么由上图之后的流程是如下图所示



在UI 的\_\_init 的时候会触发双向绑定，

在UI的onOpen 的时候会触发controller 中的onUIOpen,可以在这里处理一些UI打开相关的逻辑业务,.

在UI的onClose 的时候会触发controller 中的 onUIClose 方法，在这里可以做一些UI关闭之后的逻辑业务.在这里也会触发UI和Controller的解引用.

在业务数据发生数据变化的时候调用doUpdateVo 方法会触发UI的onVoUpdate方法，这onVoUpdate中需要根据业务数据去刷新UI。

此是大部分功能模块的通用套路，总的来说，数据和显示分离，数据去影响UI展示(这里说的就是业务逻辑)

1. LUIWidget

所有游戏里控件的基类，这里是一层控件封装，通说说也是一层适配，方便以后再不同的UI架子上兼容.(UGUI,NGUI)，提供统一的解引用的释放接口,可以去查阅源码.

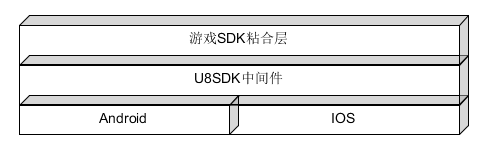
1. LCustomLayout

这个是为了做一些组合型控件来的，方便减少业务代码,如格子控件，tips控件等,有兴趣去查阅下代码.*程序员写代码只是为了更少的写代码，以后希望多做一些自定义的控件,*

总体来说这个UI架子设计目的是基本达成了，但是还有些地方没实现的很完美，可以慢慢的优化.

3.SDK层

所有SDK相关的业务在这层完成,不侵蚀到游戏业务中，之后扩展也不要在游戏层次做一些SDK业务,游戏层



1. 游戏SDK粘合层.

这一层主要是适配不同的SDK(有可能是一些聚合SDK)提供的接口，提供给游戏统一的调用接口,通讯的格式定义为Json字符串,适配器做不同的分发.(之后可以精简，原来使用过anysdk,但是现在只需要给U8提供接口就可以了)

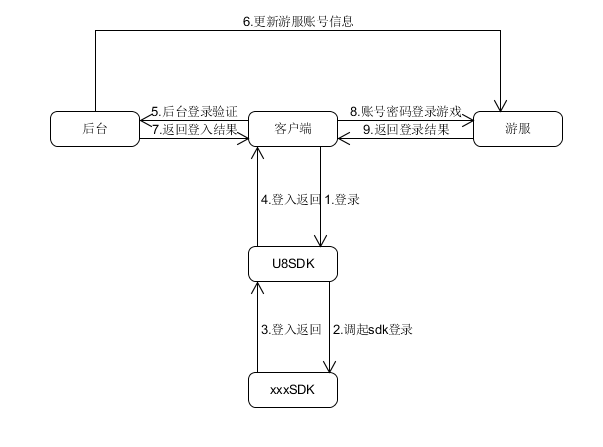
1. U8SDK 中间件.

U8SDK 包括2部分，

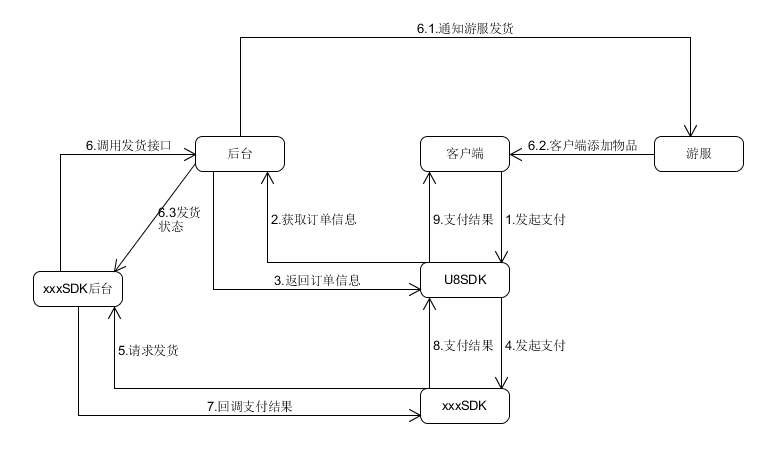
1. 承担实际SDK 对接和游戏对接的中间件

我们游戏和U8SDK交互的流程图,我们只是修改登录和支付相关的接口，更符合我们游戏，没有修改流程

我们目前的登陆流程



我们目前的支付流程



支付相关的回调是异步回调来的.

其他如游戏打点，推送，等相关和此流程类似，这里就不做一一列举了

1. 打包工具(这个需要重点去看，关于Android和IOS如何做渠道打包)

整个U8SDK 我个人感觉最为精华的是工具这块的代码.

1. 对接实际平台的SDK

我们只做过些定制，但是还是可以查看U8SDK对接地址.

<http://wiki.u8sdk.com/#!index.md>

这里可以查阅到所有U8相关的资料.

对于整个SDK这块主要是2个粘合层,不要超过粘合层去处理问题,.

案例一：如渠道方的也采用U8SDK,我们如何处理，

1. android，实现一层针对渠道U8SDK的中间件,保证对游戏提供的接口不变，保证我们的登录和支付流程不变,切除一个启动器,去适配新做的粘合层.
2. IOS ,也可以同android一样处理，只是需要些OC，目前我们采用的策略是使用抛弃掉我们U8SDK粘合层,使用渠道的U8SDK,游戏粘合层去适配这个中间件层，只需要保证对游戏提供的接口不变,保证登录流程和支付流程不变，我们就能做到在不修改游戏主体业务的情况下，尽肯能多的去适配渠道的需求.

当然这增加了我们游戏的维护成本(虽然他们的补丁包可以共用)，但是我们还是要为这样的渠道每新增一个，我们要为他专门做一条发布线.所以这样的渠道需求。还是能少就少一些.

4.打包工具

核心的原理就是 依赖资源要优于被依赖资源被打包

1.分析依赖,构建依赖树

2.打依赖资源

3.打被依赖资源.

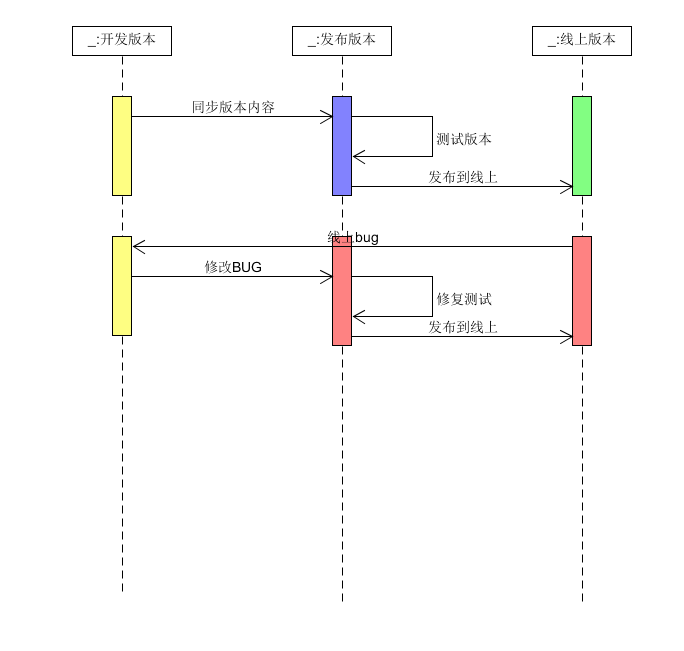
为了减少资源冗余.我们给资源做了区分.也就是上面提到的目录结构.

1. 该目录下的的每个资源都会被单独打包 其依赖的资源也会被打包出去,按照最小的打包力度 适用于 预设，或者是散图资源目录下的单个资源文件为单位打包(U)
2. 目录下的每一个子文件夹打包为一个AB, 适合用于所有依赖都在一个文件夹内 如图集
3. 子目录中的文件打包到一个ab, 目录下的文件单独打包 如字体

打包的源代码在Script\XNGame\Tools\Packer,详细了解可以看看的代码.

发布篇

开发和发布的流程



这个部署平台包含2个部分,打包工程部署，Jenkins部署

1.打包工程的部署结构

Android:

Android启动器

程序代码

程序DLL资源

美术资源

U8SDKAndroid工具

IOS:

IOS启动器

程序代码

美术资源

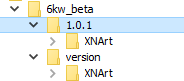
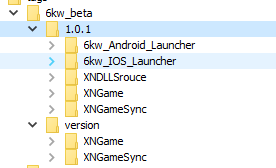
U8SDKIOS工具

所有的打包工程 都是有启动器去隐射其他的工程资源,这样能保证不同类型渠道只有启动器不一样，但是所需要的资源和代码是一样的，版本维护补丁包共享..

这里给出一个SVN的部署结构.以6kw发布地址 为例

<https://3dproject_src:444/svn/XN/tags/6kw_beta/1.0.1>

客户端部署结构为



左为程序结构，右为策划工程结构

XNDLLSrouce ,DLL编译资源,

提供android工程dll资源

XNGame 代码资源，

1.提供给XNDLLSrouce 源码隐射资源

2.提供给6kw\_IOS\_Launcher 程序代码资源

3.提供给6kw\_Android\_Laucher 启动器代码,和ab打包工具

XNGameSync. 代码同步

1. 主要是提供给策划工程的编译源代码 和各类工具

6kw\_IOS\_Launcher IOS启动器工程，也是Ios打包使用的工程

6kw\_Android\_Laucher Android启动工程，也是Android 打包使用的工程。

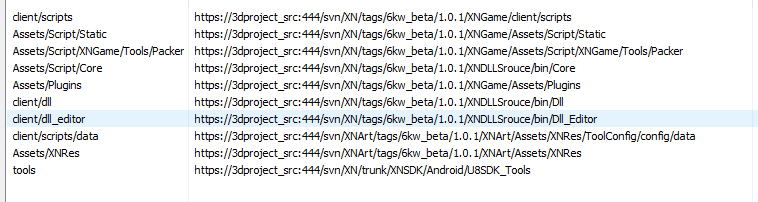
Version 目录下的工程为每次版本发布后镜像工程，这样可以在做联运是切一个稳定版本今夕修改

XNArt 资源工程

1. 提供6kw\_IOS\_Launcher 和 6kw\_Anddroid\_Launcher 工程的 美术资源 以及 策划配置表资源.

Version 目录下的。为每次版本发布够的镜像资源工程，同程序镜像一样

这里给出6kw\_Android\_Launcher的SVN隐射配置.



6kw\_IOS\_Launcher svn 隐射配置



1. Jenkins部署

android的发布机器Jenkins为<http://192.168.1.13:8080>

IOS发布机器Jenkins

<http://192.168.1.104:8082> 全民仙逆渠道发布

<http://192.168.1.137:8080> 6kw和新马发布渠道

1. android部署构建

1.参数相关

|  |  |  |
| --- | --- | --- |
| 参数名 | 参数类型 | 参数描述 |
| BuildFrom | Choices | 发布的工程来源，追查bug来源那个工程 |
| AppParam | Choises | App参数, |
| BuildChannel | Choises | 渠道包出去的母渠道,和连接那个后台有关 |
| SDKMode | Choices | Sdk接入模式,（u8sdk,debug(调试模式)，hd(会打聚合SDK的母包)） |
| channels | Choices | 打该渠道的子包,只有在U8SDK 这个选项时生效 |
| BuildMode | Choices | 默认选择BuildAll |
| ResVer | Choices | 资源版本号 |
| ClientVer | Choices | 客户端版本号 |
| BuildPatch | Boolean | 是否打补丁包,（开启此选项，会针对上一个版本到这个版本的补丁包） |
| Develop | Boolean | 是否开启调试模式(会打开游戏日志输出，查看错误) |
| ForceBuild | Boolean | 锁定当前发布版本的SVN版本号，每次打包发布新版本的时候会默认锁定一次,之后如果在打此版本不勾选是默认不会锁定的, |
| versionName | String | 应用程序显示版本,每次打包向上提以为,默认格式为,1.0.0.1 |
| versionCode | String | 应用程序版本号，每次打包比上一次高就可以了 |

2.bat 模板.

1.打包开始的清理模板

[android\清理模板.txt](android/清理模板.txt)

2.构建资源模板

[android\构建资源模板.txt](android/构建资源模板.txt)

注意修改其中svn 的工程地址 和 unity3d安装目录

3.构建渠道App模板

[android\构建渠道App模板.txt](android/构建渠道App模板.txt)

注意修改其中的python 安装地址，默认的python为 python2.7.x版本.

4.备份到发布地址模板

[android\备份到发布地址模板.txt](android/备份到发布地址模板.txt)

默认android 备份的制止 为 192.168.1.11 如果要修改。记得修改里面的备份地址

1. IOS部署构建
2. 参数模板

|  |  |  |
| --- | --- | --- |
| 参数名 | 参数类型 | 参数描述 |
| BuildFrom | Choices | 发布的工程来源，追查bug来源那个工程 |
| AppParam | Choises | App参数, |
| BuildChannel | Choises | 渠道包出去的母渠道,和连接那个后台有关 |
| SDKMode | Choices | Sdk接入模式,（u8sdk,debug(调试模式)，hd(会打聚合SDK的母包)） |
| channels | Choices | 打该渠道的子包,只有在U8SDK 这个选项时生效 |
| BuildMode | Choices | 默认选择BuildAll |
| ResVer | Choices | 资源版本号 |
| ClientVer | Choices | 客户端版本号 |
| BuildPatch | Boolean | 是否打补丁包,（开启此选项，会针对上一个版本到这个版本的补丁包） |
| Develop | Boolean | 开发模式 这里的开发指能在测试设备上安装, |
| ForceBuild | Boolean | 锁定当前发布版本的SVN版本号，每次打包发布新版本的时候会默认锁定一次,之后如果在打此版本不勾选是默认不会锁定的, |
| Version | String | 应用程序显示版本,每次打包向上提以为,默认格式为,1.0.0.1 |
| Build | String | 应用程序版本号，每次打包比上一次高就可以了 |

1. 构建 shell 模板

1.打包开始的清理模板

[Ios\清理模板.txt](Ios/清理模板.txt)

2.构建资源模板

[Ios\构建资源模板.txt](Ios/构建资源模板.txt)

注意修改其中svn 的工程地址 和 unity3d安装目录

3.构建渠道App模板

[Ios\构建渠道App模板.txt](Ios/构建渠道App模板.txt)

注意修改其中的python 安装地址，默认的python为 python2.7.x版本.

4.备份到发布地址模板

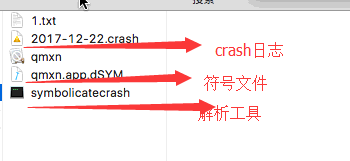
[Ios\备份到发布地址模板.txt](Ios/备份到发布地址模板.txt)

默认android 备份的制止 为 本机，当然也可以统一去管理,这里是我比较懒不想怎么研究 如果要修改。记得修改里面的备份地址

3.关于IOS Crash查看,

1.备份符号文件.现在所有的发布版本都会把发布包的符号文件和app文件同步备份.

2.将crash日志和对应匹配的符号 按照如下目录放置



在终端依次执行如下指令

export DEVELOPER\_DIR=/Applications/Xcode.app/Contents/Developer

/Users/chuangyou/test/Unity-iPhone/symbolicatecrash /Users/chuangyou/test/Unity-iPhone/2017-12-22.crash /Users/chuangyou/test/Unity-iPhone/qmxn.app.dSYM >1.txt

执行完成后在1.txt 中就是一家被符号化的crash日志，找到对应的错误堆栈解决就好了.

网上资料参考: https://www.jianshu.com/p/12a2402b29c2

其他相关

1. ios审核包

ios 包过审主要有2步

1. 机审 看是否有相同的字节码在他的游戏库中

我们解决这个步骤是在oc层次加入大量无用代码参与编译。目前容易过机审

1. 人审，主要看的是在appstore 上是否有同类的产品.是一个很看脸的审核步骤

我们目前的解决办法有2种

1. 制作不同体验内容的角色。 角色停留在不同的游戏场景，角色穿戴不同的游戏时装，来达到差异性.
2. 抽取主界面的UI图集元素，交由美术重新制作，然后打包.上传到CDN.并提高该渠道的审核资源版本.这样在游戏启动的时候回下载这个资源包.更改游戏主界面，审核完成后修改资源版本，抹除资源版本号.不会影响后续通过这个app进入玩家主界面相关的显示.<此方案目前没有投入使用>

工具在 <渠道工具/导出资源列表>

工具源码Script\XNGame\Tools\Channel\Editor

工具配置 XNRes\Channel\ exportRes.txt 每一行为UI预设路径

工具SVN地址：<https://3dproject_src:444/svn/XN/trunk/XNChannelPublish>

工具打包工程：<http://192.168.1.137:8080/job/ChannelResPublish>