## SDK管理方案

### 方案说明

* 将主程序 和 渠道 代码 彻底分离，原则上主程序中不对渠道SDK做特殊逻辑。
* 打包的时候，再拷贝渠道文件夹，再修改工程配置（可采用脚本方式） 。

### 整个工程文件夹布局

Assets

Channel(渠道和主体的差异以及新增的部分)

渠道1(例如)

Assets(所有改变的资源以及脚本)

Animation

Aduio

Image

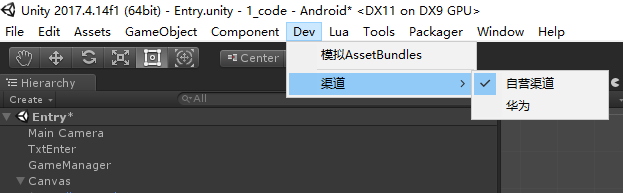
Lua

Materials

Prefab

(以上文件夹视情况添加，因为有可能这个渠道没有改变图片，那就不用去建Image了，建了也没事)

测试和打包会根据工程里的Dev-渠道-渠道类型去使用



Editor

目前只放了打包选项脚本

…

Game

Activity(所有活动)

Activity1(例如)

(同游戏目录)

…

Channel

每个渠道的SDK文件夹

## SDK 调用方式

参数格式：用 json 打包

结果返回：异步回调

## 接口定义

/// <summary>

/// 主程序 => SDK 接口： 在 SDK 中实现

/// </summary>

public abstract class JYSDKInterface{

    // 初始化

    public abstract void Init(string \_json\_data);

    // 登录

    public abstract void Login(string \_json\_data);

    // 支付

    public abstract void Pay(string \_json\_data);

    // 登出

    public abstract void LoginOut(string \_json\_data);

    // 分享

    public abstract void Share(string \_json\_data);

    // 个人中心

    public abstract void ShowAccountCenter(string \_json\_data);

}

/// <summary>

/// 结果返回：SDK 接口 => 主程序， 在主程序中实现

/// </summary>

public abstract class JYSDKResult{

    // 初始化

    public abstract void InitResult(string \_json\_data);

    // 登录

    public abstract void LoginResult(string \_json\_data);

    // 支付

    public abstract void PayResult(string \_json\_data);

    // 登出

public abstract void LoginOutResult(string \_json\_data);

// 分享

    public abstract void Share(string \_json\_data);

    // 个人中心

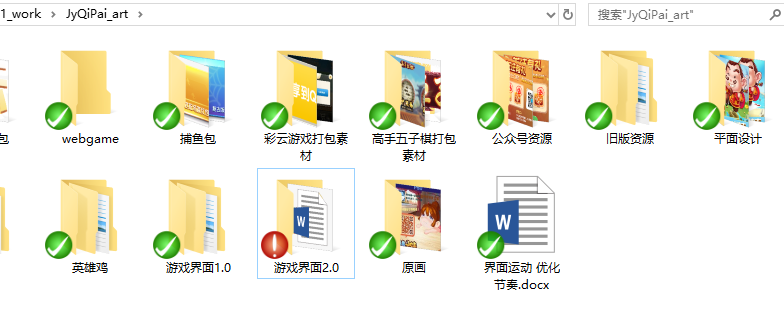
    public abstract void ShowAccountCenterResult(string \_json\_data);

}

## 实施清单

1. 逻辑分离：将SDK和主游戏逻辑分离。
2. SDK框架实现。
3. 打包工具：自动拷贝、修改工程设置、打包。

## 美术活动资源目录结构



在这个同级目录下新建一个活动文件夹(比如叫活动1.0)，里面再按照活动划分文件夹，这个文件夹存放这个活动的所有资源。

活动可以使用游戏中的基础资源，如关闭、按钮等。

对于活动之间资源复用的问题，尽量把背景底板做成可复用，被复用的资源放在活动1.0的活动公用资源文件夹中，其他差异部分切出来，放到对应活动目录下。活动公用资源目录下要保存一份文档，记录每个资源被那些活动使用，方便以后进行修改和删除。