GplaySDK 游戏开发接入文档

Cocos2d-C++

简述

Gplay 手机页游解决方案, 可以让游戏

- 无需安装, 点开即玩
- 按需下载场景需要的资源,快速进入游戏
- 一次接入, 无需再次开发, 可直接上架所有 Gplay 合作渠道
- 像网页游戏一样灵活地部署和升级

本文档将介绍如何将你的游戏接入 **GplaySDK**, 主要内容包括

- 接入环境要求
- 接入步骤
- GplaySDK C++ API
- API使用说明与演示
- 接入FAQ

环境要求

- NDK: 建议使用NDK r10e (Window Mac)编译本地代码，具体缘由请查看FAQ。

更多Android开发相关工具（SDK、NDK、JDK等）可考虑从国内镜像网站Android Dev tools下载。

接入步骤

1. 创建游戏

游戏开发者需要登录 GPlay 开放平台 创建一个游戏, 填写游戏信息. 游戏创建成功以后, 会获得游戏的 `client_id`, `client_secret`, `private_key` 三个参数, 需要在初始化 GPlaySDK 的时候传入, 用于标识游戏和 SDK 信息的传输加密;

在接入阶段可以使用测试参数来完成接入，三个参数对应的测试值都是 `gplaydemo` 。测试完成之后务必将这三个参数改成从GPlay 开发平台申请到的值。

注: 游戏开发者需要保持这三个秘钥的保密性.

2. 集成 GplaySDK 库

- 获取 GplaySDKForCPP.zip, 解压得到 `gplay_for_cpp` 文件夹
- 添加对 GplaySDK 库的依赖:
  - 复制 `gplay_for_cpp` 文件夹到 游戏Android工程 `pro.android/jni` 目录下
  - 编辑 游戏Android工程 `pro.android/jni` 目录下的 **Android.mk** 文件, 添加对 GplaySDK 的引用

```
# 在项目原始 LOCAL_WHOLE_STATIC_LIBRARIES 之后添加
LOCAL_WHOLE_STATIC_LIBRARIES += gplay_static
# ...
# 在项目原始 import-module 之后添加
$(call import-module, ../proj.android/jni/gplay_for_cpp)
```

  - **引擎版本低于 3.4 的游戏**需要在 `cocos/platform/CCFileUtils.h` 中增加以下代码:

```
static void setDelegate(FileUtils *delegate) {
    if (s_sharedFileUtils != delegate)
        delete s_sharedFileUtils;
    s_sharedFileUtils = delegate;
}
```

3. 处理非引擎功能的JNI调用

处理非引擎功能JNI的调用有两种可选方法

- 使用扩展jar，保留大部分或者所有非引擎功能的JNI调用
- 完全去除非引擎功能的JNI调用

3.1 使用扩展jar，保留跟JNI交互的业务代码

将跟JNI交互的java代码生成一个dex格式的jar包，并通过接入工具添加进来，一般情况下，c++代码不用作任何修改。

这个jar包必须符合以下条件

- jar必须转成dex格式;
- 扩展jar不能访问Android SDK任何组件及不能对Android UI做任何控制;
- 如果Jar中有对Jni的调用，则必须确保游戏so中有此JNI方法;
- 如果游戏中有对Java方法的调用，jar必须正确包含此Java的类名引擎与方法定义;
- jar包不得与宿主App产生类冲突，否则jar需要自行解决

3.2 去除游戏代码（C、C++、脚本）中包含有对 标准cocos2d-x引擎Java层没有提供的功能 的依赖及相关

JNI调用

GPlay在Android平台Java层只提供了标准引擎原有功能和Gplay服务的支持，游戏代码如果包含有其他非标准引擎功能的JNI调用及相关依赖等, 可能导致游戏在运行时出现 **JNI相关调用异常**等。

在Gplay模式下需要屏蔽这类调用,可以使用预定义宏控制不参与编译，或者运行时判断是否为Gplay环境（`gplay::isInGplayEnv()`）。

这些调用通常是对第三方SDK的调用, 如统计SDK, 支付SDK, 推送SDK以及开发者在Java层扩展的其他功能等。

4. 通过 Gplay 接入工具运行游戏, 验证是否有适配问题

重新打包游戏APK，这一阶段建议编译 **Debug** 包, 接下来的调试阶段可能需要经常查看游戏和引擎的log输出。

打开 Gplay 接入工具 **GplayTool**, 导入生成的游戏 APK 来添加游戏. 连接 Android 设备, 点击右上角**运行游戏**. Android 设备上将会以 Gplay 形式启动游戏, 验证游戏功能是否正常

- 注:
  - 工具使用请参考 "**Gplay 接入工具使用说明文档**"
  - 未分场景的情况下运行游戏, 会先下载游戏的所有资源, 主要用于测试适配性, 分过场景以后, 只需要下载 Boot_Scene 里的资源就可以启动游戏
  - 点击**显示日志** 可以查看设备运行的 log
  - 若有修改native代码（C、C++）需要重新编译生成 Debug APK

完成条件: 通过 Gplay 接入工具运行游戏, 游戏正常运行, 没有 JNI 调用异常等错误

5. 游戏分场景, 调优分场景效果

- 使用 Gplay 接入工具为游戏资源分组:

对于游戏的不同场景如 **战斗**, **商店** 等模块建立场景分组, 在工具中配置此场景分组中需要用到的资源。以满足该场景正常运行的资源集合为准, 不同场景分组中的资源 **可以重复**。

- 修改游戏代码

进入场景前先调用 `gplay::preloadGroup(group, successCallback)` 加载该场景需要的资源分组, 在 successCallback 回调中执行进入场景的代码. (下一部分的 API 说明中有 Demo 演示)

- 测试场景划分是否遗漏

重新打包游戏APK，**关闭** 工具 **静默下载** 开关, 点击 **运行** 启动游戏。工具会按照当前的场景分组重新运行游戏, 但不会主动后台下载资源, 每次进入新的场景都会触发前台加载界面. 期间可能由于场景遗漏了必需的资源而导致游戏崩溃, 游戏开发者可以通过查看游戏 debug 日志可以获得提示, 找到当前场景需要加载到却没有配置的资源, 加入当前场景. 调整场景分组后继续点击 **运行** 再次启动游戏测试, 直到没有崩溃.

- 测试场景划分是否流畅

打开 工具 **静默下载** 开关, 点击 **运行** 启动游戏, 工具会按照当前的场景分组重新运行游戏, 并且自动静默下载后续资源, 观察游戏进入前台 loading 的情况, 如果次数过多, 则需要调整优化. 原则上将最先用到的几个场景分得细一点, 可以达到比较完美的游戏效果, 基本不存在 loading.

完成条件: 通过 Gplay 接入工具运行游戏, 没有崩溃, 没有无法接受的前台 loading 状况

- 注:
    - **这个过程需要多次反复测试以达到最佳效果**
    - **在 preloadGroup 回调之前, 请勿再次调用此接口**

6. Gplay 模式下屏蔽游戏自身的热更新

Gplay 模式有一套完善的游戏版本更新机制, 将游戏原本需要的大/小版本更新统一成了 **以场景分组为单位** 的 **增量更新**, 不需要特别制作, 工具会自动生成升级包随同新版本一起发布到线上. 升级更加方便和顺滑, 使用这种方式升级, **游戏开发者需要屏蔽游戏原有的热更新**

7. 调用 GplaySDK 接口

- 客户端接入
    - 包含 `gplay_cpp.h` 头文件, 调用渠道的**登陆**, **支付**, **分享**, **发送到桌面快捷方式**等功能
    - 修改原来结束游戏的接口, 改用 **gplay::quitGame()**
- 服务端接入
    - 使用 GplaySDK 的支付接口, 游戏开发者需要在 Gplay 开放平台配置当前游戏的 **支付回调接口**, 客户端支付成功后该回调接口会被调用, 返回支付信息 *(参考 **GplaySDK 服务端接入文档**)*

8. 发布, 测试, 上线

- 发布

修改 jni/Application.mk: `APP_ABI := armeabi armeabi-v7a x86` 生成包含各个架构的 **release** 版本 APK, 在工具中更换 APK 包, 点击 **发布**, 工具将根据最新的场景划分, 使用新 APK 中的资源和 release 代码, 生成一套 GplayPackage, 用于发布. 游戏开发者需要将此文件夹部署到 CDN 上, 并到 Gplay 开放平台上配置 **下载地址** 为游戏包根目录的 URL. 大功告成!

- 测试

安装工具包中附带的 GplayDemo.apk 到 Android 设备中, 运行并找到自己的游戏, 点开即玩.

- 上线

Gplay 审核游戏过后, 会将游戏上线

9. 游戏版本更新

游戏在准备正式发布大小版本的时候, 需要同时部署更新对应版本的 GplayPackage

- 更新新版本资源

使用 Gplay 接入工具导入新版本APK, 工具会自动清理上版本资源配置文件中新版本中不再使用的资源, 其余资源配置保持不变. 游戏开发者根据新版本的资源变动, 回到 **步骤 5**, 将未分配的资源分配到各个场景中

- 生成

配置游戏的当前线上地址到工具游戏配置的 **发布地址** 栏目中, 点击 **发布** , 此时生成的 GplayPackage 将包含对当前线上版本资源的 **增量升级包**, 玩家在进入游戏场景时会自动下载场景 **增量升级包** 来升级游戏场景

- 部署

直接使用最新的 GplayPackage 的内容替换游戏发布 服务器 上的 GplayPackage

- 注:
  - Gplay 模式下, 不管是资源、脚本还是本地代码的变化, 都将统一使用 **增量更新**
  - 增量更新的用户可以快速开始新版本的游戏, 按需升级不同的场景分组.
  - **Gplay 模式的更新机制统一了游戏的大/小版本更新, 开发者需要屏蔽游戏自身的热更新逻辑**

API 列表

API 名称	API 说明
gplay::initSDK()	初始化SDK
gplay::getNetworkType()	获取网络类型
gplay::preloadGroup()	加载一个场景分组
gplay::preloadGroups()	加载多个场景分组
gplay::backFromGroup()	自动分场景资源模式下停止分配资源到指定场景
gplay::setPreloadResponseCallback()	设置加载场景分组的响应回调
gplay::isLogined()	检查登录状态
gplay::getUserID()	获取用户 ID
gplay::login()	登录
gplay::quitGame()	退出游戏
gplay::share()	分享游戏
gplay::pay()	支付
gplay::createShortcut()	创建桌面快捷图标
gplay::isFunctionSupported()	判断是否支持某个方法
gplay::callSyncFunc()	调用同步扩展接口
gplay::callAsyncFunc()	调用异步扩展接口

API 说明

gplay::initSDK(appKey, appSecret, privateKey) 获取参数信息

描述

初始化 GplaySDK, 需要在**使用登录支付等API调用前**调用

参数

initSDK 所使用到的三个参数, 是开发者在 Gplay 开放平台上配置游戏的时候获得的

参数	类型	说明
appKey	const string&	"游戏的 Gplay 唯一标识
appSecret	const string&	游戏的 Gplay 密钥
privateKey	const string&	游戏的 Gplay 签名盐值

结果

无, 将参数初始化到游戏运行时配置中, 用于敏感 API 的加密.

---

gplay::getNetworkType() 获取网络类型

描述

获取当前设备所处的网络类型

参数

无

结果

返回网络状态, 可以使用 SDK 提供的枚举判断, 有如下 3 种:

返回枚举值	说明
gplay::NetworkType::NO_NETWORK	设备当前**无网络连接**
gplay::NetworkType::MOBILE	设备当前处于**移动网络**环境下
gplay::NetworkType::WIFI	设备当前处于**WIFI网络**环境下

---

gplay::preloadGroup(group, callback) 加载场景资源

描述

游戏进入场景前调用, 指定当前需要用到的场景分组, gplay sdk 会检测判断是否需要下载缺失的资源:

- 指定的场景分组资源完整, 不需要下载：callback 会立即被调用通知游戏加载完成, 执行回调中的代码;
- 指定的场景分组资源本地不存在、不完整或者线上有更新, 需要下载：gplay sdk 开始下载指定的场景分组, 下载完成后通过 callback 通知游戏。
- 使用Gplay Tools自动分场景资源时，callback会立即被调用通用游戏加载完成，从这一时间点起访问的

相关资源会被分配到这个场景，直到加载新的场景。

- 在 preloadGroup 回调之前, 请勿再次调用此接口；
- 对应场景（Scene或者逻辑上的使用场景）退出时应该调用backFromGroup(sceneName)，以支持在GPlay Tools上使用自动分场景功能

参数

参数	类型	名称	参数说明
group	string	场景分组	指定需要确认加载的场景名称
callback	function	加载完成回调函数	加载场景完成的后续逻辑, 也就是当前场景的游戏逻辑

结果

加载情况	效果
加载成功	执行传入的回调, 继续游戏
空间不足	弹框提示“空间不足”错误信息, 提示玩家选择**重试**或者**退出**
校验失败	弹框提示“校验失败”错误信息, 提示玩家选择**重试**或者**退出**
网络异常	弹框提示“网络异常”错误信息, 提示玩家选择**重试**或者**退出**

说明

接入过程中, 只需要把进入场景时的原有逻辑, 或逻辑入口放在 callback 中即可, 如将

```
auto scene = SceneGame::createScene();
Director::getInstance()->replaceScene(scene);
```

改为

```
static void* preloadResultCallback(int result, const char* jsonResult) {
    auto scene = SceneGame::createScene();
    Director::getInstance()->replaceScene(scene);
}
gplay::preloadGroups("scene1", preloadResultCallback);
```

gplay::preloadGroups(groups, callback) 加载多个场景分组资源

描述

使用此方法, 可以在游戏进入场景前, 指定下载当前游戏场景需要的多个分组资源。 使用GPlay Tools自动分场景功能，不支持一次加载多个分组（工具无法明确资源应该划分给哪一个分组）

参数

参数	类型	名称	参数说明
groups	vector	场景分组列表	指定需要确认加载的场景名称列表
callback	function	加载完成回调函数	加载场景完成的后续逻辑, 也就是当前场景的游戏逻辑

示例

```
std::vector<std::string> scenes;
scenes.push_back("scene1");
scenes.push_back("scene2");
//...
gplay::preloadGroup(scenes, [](int result, const char* jsonResult) {
    auto scene = SceneGame::createScene();
    Director::getInstance()->replaceScene(scene);
});
```

---

gplay::backFromGroup(group)

描述

GPlay Tools自动分场景资源模式下停止分配资源到指定场景，建议在场景资源相关联的Scene等节点的cleanup调用。

示例

```
void GameScene::cleanup()
{
    Scene::cleanup();
    gplay::backFromGroup("gameScene");
}
```

---

gplay::setPreloadResponseCallback(GPreloadResponseCallback callback) 设置加载场景分组的响应回调

描述

设置加载场景分组的响应回调，用于CP定制加载场景分组的交互界面。如果CP没准备定制加载交互界面则不需要调用这个接口。

示例

```
gplay::setPreloadResponseCallback([&](int resultCode, int errorCode, const std::st
ring& groupName, float percent, float speed) {

});
```

gplay::isLogined() 检查登录状态

描述

检查游戏在渠道用户系统上的登录状态

参数

无

结果

返回布尔值	说明
true	已登录
false	未登录或已断线

gplay::getUserID() 获取用户 ID

描述

获取游戏玩家在渠道用户系统上分配的 ID

参数

无

结果

string 类型, 用户 ID

gplay::login(callback) 登录

描述

调用 GplaySDK 的登录服务, GPlay通过回调函数通知游戏登录结果。

参数

参数	类型	名称	参数说明
callback	function	登录结果回调函数	登录模块处理完毕后调用, 通知游戏登录结果

结果

调用传入的 callback 方法, 会传入两个参数

参数	类型	名称	参数说明
code	int	登录结果代码	用于标示此次登录的结果
msg	const char*	登录结果消息	相对于登录结果的一条消息

使用说明

游戏传入的回调函数, 需要对此回调结果做出相应的处理, 将登录逻辑移至回调当中, 例:

```
gplay::login([](int result, const char* jsonResult) {
    switch (result)
    {
        case USER_LOGIN_RESULT_SUCCESS:
        {//玩家登录成功
            break;
        }
        case USER_LOGIN_RESULT_FAIL:
        {//玩家登录失败
            break;
        }
        case USER_LOGIN_RESULT_CANCEL:
        {//玩家取消登录
            break;
        }
        case USER_RESULT_NETWROK_ERROR:
        {//发生网络错误
            break;
        }
        case USER_RESULT_USEREXTENSION:
        {//其他扩展的结果类型
            break;
        }
        default:
        {//未知错误
            break;
        }
    }
});
```

gplay:quitGame() 退出游戏

描述

游戏开发者需要将结束游戏的 `Director::getInstance()->end();` 等结束游戏的 API 替换为 `gplay::quitGame()`.

参数

无

结果

关闭游戏, 游戏中的生命周期函数在关闭游戏之前都会被调用到.

gplay::share(shareParams, callback) 分享游戏

描述

调用 GplaySDK 的分享游戏方法, 呼唤渠道的分享游戏功能, GPlay通过回调函数通知游戏分享结果.

参数

参数	类型	名称	参数说明
shareParams	GplayShareParams	分享参数	对象实例保存着参数键值对
callback	function	分享结果回调	分享功能处理完毕后通知游戏分享结果

结果

调用传入的 callback 方法, 会传入两个参数

参数	类型	名称	参数说明
code	int	分享结果代码	用于标示此次分享的结果
msg	const char*	分享结果消息	相对于分享结果的一条消息

分享接口的结果有如下几种:

枚举值	说明
gplay::SHARE_RESULT_SUCCESS	分享成功
gplay::SHARE_RESULT_FAIL	分享失败
gplay::SHARE_RESULT_CANCEL	取消分享
gplay::SHARE_RESULT_NETWORK_ERROR	网络错误
gplay::SHARE_RESULT_SHAREREXTENSION	扩展结果

使用说明

GplayShareParams 是分享接口需要传入的分享参数类型, 需要提前设置好必填参数, 否则会报错.

```
GplayShareParams params;

params.pageUrl = "http://192.168.1.3";                                    -- 分享页面
params.title = "CppSDKDemo";                                              -- 分享标题
params.content = "CocosCppSDKDemo 是 GplaySDK 的线下测试版本";             -- 分享内容
params.imgUrl = "http://192.168.1.3:8888/moon/icon/large_icon.png";       -- 分享图片
params.imgTitle = "large_icon";                                           -- 分享图片标
题

gplay::share(params, shareResultCallback)
```

```
static void shareResultCallback(int result, const char* jsonResult) {
    switch (result)
    {
        case SHARE_RESULT_SUCCESS:
        {//分享成功
            break;
        }
        default:
            break;
    }
}
```

gplay::pay(payParams, callback) 支付

描述

调用 GplaySDK 的支付方法, 呼唤第三方支付系统的支付功能, 成功以后调用回调方法, 通知游戏支付结果.

参数

参数	类型	名称	参数说明
payParams	GplayPayParams	支付参数	对象实例保存着参数键值对
callback	function	支付结果回调	支付功能处理完毕后调用通知游戏支付结果

结果

调用传入的 callback 方法, 会传入两个参数

参数	类型	名称	参数说明
code	int	支付结果代码	用于标示此次支付的结果
msg	const char*	支付结果消息	相对于支付结果的一条消息

支付接口的结果有如下几种:

枚举值	说明
gplay::PAY_RESULT_SUCCESS	支付成功
gplay::PAY_RESULT_FAIL	支付失败
gplay::PAY_RESULT_CANCEL	支付取消
gplay::PAY_RESULT_NETWORK_ERROR	网络错误
gplay::PAY_RESULT_INVALID	支付参数不完整
gplay::PAY_RESULT_NOW_PAYING	正在支付
gplay::PAY_RESULT_PAYEXTENSION	扩展结果

使用说明

GplayPayParams 是支付接口需要传入的支付参数类型, 需要提前设置好必填参数, 否则会报错.

```
static void payResultCallback(int result, const char* jsonResult) {
    switch (result)
    {
        case PAY_RESULT_SUCCESS:
        {//支付成功
            break;
        }
        case PAY_RESULT_FAIL:
        {//支付失败
            break;
        }
        default:
            break;
    }
}
//...
GplayPayParams params;
params.productId = "gplayid001";
params.productName = "monthly card";
params.productPrice = 100;
params.productCount = 100;
params.productDescription = "monthly";
params.gameUserId = "userid001";
params.gameUserName = "zhangjunfei";
params.serverId = "001";
params.serverName = "server01";
params.extraData = "{}";
pay(params, payResultCallback);
```

gplay:createShortcut(callback) 创建桌面快捷图标

描述

调用 GplaySDK 的创建桌面快捷图标功能, 成功以后调用回调方法, 通知游戏是否创建成功.

参数

参数	类型	名称	参数说明
callback	function	创建图标结果回调	创建图标功能处理完毕后调用通知游戏创建结果

结果

调用传入的 callback 方法, 会传入两个参数

参数	类型	名称	参数说明
code	int	创建结果代码	用于标示此次创建的结果
msg	const char*	创建结果消息	相对于创建结果的一条消息

使用方法请参考 login 接口的例子

创建桌面快捷图标接口的结果有如下几种:

枚举值	说明
gplay::SHORTCUT_RESULT_SUCCESS	创建成功
gplay::SHORTCUT_RESULT_FAILED	创建失败

---

isFunctionSupported(funcName) 判断是否支持某个接口

描述

除了 login 等通用接口外，通过此函数可以来判断是否还支持其他接口.

参数

参数	类型	名称	参数说明
funcName	const string&	方法名称	希望调用到的方法名

结果

返回一个 bool 类型结果, true 表示支持, false 表示不支持

---

gplay:callSyncFunc(funcName, params) 渠道同步扩展接口

描述

除 Gplay 标准 API 之外, 渠道单独提供特殊接口需要实现时, 使用此通用的扩展接口来调用同步方法.

参数

参数	类型	名称	参数说明
funcName	const string&	方法名称	希望调用到的方法名
params	const string&	扩展接口参数	jsonString, 保存着参数信息

结果

返回一个 JSON 格式的 string 字符串, 游戏开发者根据文档从中解析出方法执行的结果

---

gplay:callAsyncFunc(funcName, params, callbackFunc) 渠道异步扩展接口

描述

除 Gplay 标准 API 之外, 渠道单独提供特殊接口需要实现时, 使用此通用的扩展接口来调用异步方法.

参数

参数	类型	名称	参数说明
funcName	const string&	方法名称	希望调用到的方法名
params	const string&	扩展接口参数	jsonString, 保存着参数信息
callbackFunc	function	扩展接口回调	异步接口调用, 通知游戏执行结果

结果

调用传入的 callbackFunc 方法, 会传入两个参数

参数	类型	名称	参数说明
code	int	执行代码	用于标示此次执行的结果
msg	const char*	消息	相对于执行状态的一条消息

游戏开发者根据约定的 code 和 msg 获得方法执行的状态, 执行相应的回调代码.

使用方法请参考 login 接口的例子

FAQ

- **Q: 为什么要使用特定版本的 NDK**

> Gplay 使用的引擎标准 so 是基于 NDK r10e 制作的. 开发者使用相同版本的 NDK 编译, 可以使游戏在 Gplay 模式下更快地启动

- **Q: 我该如何分场景最好?**

> 一般基于以下几个原则

  - 初始几个场景要尽量小, 可以使游戏运行更加流畅
  - Boot_Scene 控制在 5M 以下, 可以更快速地进入游戏
  - 后续场景按实际会使用到资源来划分

- **Q: 我该如何减小 Boot_Scene 的体积?**

> 首场景通常是登录界面, 可以从以下几点下手做到小包优化:

  - 在画质可以接受范围内压缩登录界面图片资源的大小
  - 中文字体库通常较大, 更换更精简的字体库文件可以大大减小包的体积
  - 登陆界面可能用到了 1 分多钟的背景音乐, 考虑改为十几秒音乐循环播放

- **Q: 我的场景分组之间有很多重复资源, 会被重复下载吗?**

> 工具会对场景分组中重复的资源进行优化划分, 确保资源不会在实际运行时被多次下载

- **Q: 游戏每次 preloadGroups 都会去下载指定的资源吗?**

> SDK 会检查当前的场景资源是否完整, 如果当前场景完整则直接调用成功的回调

- **Q: 为什么需要屏蔽游戏对非引擎 Java 层代码的调用（JNI调用）?**

> Gplay 模式分离出纯粹的游戏脚本和使用的引擎, 不包含游戏开发者自行添加的 Android 层代码, 如对第三方 SDK, 或其他 Java 开源库. 如果在 Gplay 模式下运行的时候仍然对这些 Android 代码发起调用, 会因为找不到这些方法而崩溃.

- **Q: 游戏在 Gplay 模式下正常运行, 打包出原生游戏却崩溃要怎么办?**

> Gplay 在运行环境中添加了自己的 Android 层, 响应脚本 gplay::login() 等接口的调用. 游戏在原生 APK 中运行的时候是找不到这些 Android 方法的, 仍然调用这些接口就会出现问题. 建议通过预定义宏控制编译Runtime（GPlay）模式和非Runtime模式（原生）游戏。

- **Q: x86 架构编译失败, gcc no such file or directory**

> 检查 gcc 是否被 360 安全卫士等安全保护软件当作破坏性程序拦截了