

一、 版本信息

2.4.6-2020.08.20

二、 概述

QuickSDK 是针对不同平台渠道的统一中间件。

QuickSDK 封装了百家渠道 SDK 的各种接口，屏蔽各个渠道 SDK 的差异性，并解决渠道 SDK 频繁更新的问题。

使得 CP 只需接入一次 QuickSDK，并配合使用 PC 端自动化打包工具，就顺利能打出百家渠道包。

二、 开发包说明

A) SMPCQuickSDK.framework 是基础库，libSMPCQuickChannel.a 是各个渠道的接入实现库。

三、 接入步骤

1.QuickSDK 一般流程

- A) 游戏接入 SDK 随带的测试母包库进行接口检查。
- B) 在 QuickSDK 后台添加渠道，配置从渠道获取的参数。
- C) 母包接口测试完毕之后使用打包工具生成渠道包

2.开发环境配置

A) 将 SMPCQuickSDK.framework 和母包的 libSMPCQuickChannel.a 添加到工程中，并选择相应的 target

B) 在 Build Settings 中将 Architectures 设置为 armv7+arm64，项目本身需要支持 arm64。

Other Linker Flags 添加-ObjC

Link With Standard Libraries	Yes ↕
Other Linker Flags	-ObjC
Quote Linker Arguments	Yes ↕

C) Device Orientation 中选中游戏兼容的方向。常见配置

Device Orientation

☐

Portrait

☐

Upside Down

☒

Landscape Left

☒

Landscape Right

Device Orientation

☒

Portrait

☐

Upside Down

☐

Landscape Left

☐

Landscape Right

D) info.plist 中添加配置，以允许 http 访问

▼ App Transport Security Settings

Allow Arbitrary Loads

Dictionary (1 Item)

Boolean

YES

E) 游戏版本设置

游戏的 Version(CFBundleShortVersionString)和 Build(CFBundleVersion)需要保持一致，且使用 x.y.z 这样的点分式，比如：1.2.1。

数字前面不要加 0。因为每个渠道更新读取的版本号不一样，设置 Version 和 Build 一致才

能保证各个渠道更新都正常,如果不愿意这样设置，可以在审核不通过后再修改，通过生成

的调试工程手动出有这样要求的渠道包，要求 Version 和 Build 一致的渠道不多。

Version

1.2.1

Build

1.2.1

四、接口说明

首先配置项目信息，通过调用项目配置信息的相关接口。

1.初始化（必接）

1.1.获取实例

类：

SMPCQuickSDK

函数：

+ (SMPCQuickSDK *)defaultInstance;

功能：

获取 QuickSDK 实例

1.2.初始化 QuickSDK

类：

SMPCQuickSDK

函数：

- (int)initWithConfig:(SMPCQuickSDKInitConfigure *)configure
application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions;

功能：

使用 productKey , productCode , 初始化 SDK, 由 QuickSDK 平台分配。在 iOS 打包工

具上可以看到，打包时需要和 iOS 打包工具上的选择的产品参数一致。

参数：

configure 封装了 productKey 和 productCode

application: 应用的 application, 调用函数的参数原封不动的传入即可

launchOptions:应用的 launchOptions, 调用函数的参数原封不动的传入即可。

说明：

该接口需要在应用加载完成回调中调用，即

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

案例：

```
//初始化
//监听初始化完成事件
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(smpcQpInitResult:)
name:kSmpcQuickSDKNotiInitDidFinished
```

```

        object:nil];
- (void)smqcQpInitResult:(NSNotification
        *)notify { NSLog(@"%@",notify);

//初始化成功，进行下一步流程
}

SMPCQuickSDKInitConfigure *cfg = [[SMPCQuickSDKInitConfigure alloc]
init]; cfg.productKey = @"09633196";
cfg.productCode = @"50840817638746911281319234382938";
int error = [[SMPCQuickSDK sharedInstance] initWithConfig:cfg application:
application didFinishLaunchingWithOptions:launchOptions];
if (error != 0) {
    NSLog(@"不能启动初始化: %d",error);
}

```

//大部分渠道会处理自己内部的初始化失败，如果收到渠道初始化失败，游戏需要去处理，通常的做法是让界面处于等待，等 0.5s 再初始化一次，如果失败就提示检查网络退出游戏

2.用户接口

2.1.登录接口（必接）

类名：

SMPCQuickSDK

函数：

- (int)login;

功能：

打开登录界面，进入用户登陆流程。

返回值：

返回 0 说明调用接口成功，其他值产考说明。

收到登录成功通知 kSmpcQuickSDKNotiLogin 后，取得返回值 userInfo,取得用户 uid、token、用户昵称 /用户名和 user_token，也可直接使用接口获取 uid，昵称，参考 demo，（可选内容）**可以使用该 user_token 从服务器获取用户信息**。随带 Demo 中有相关实现。该 uid 是渠道 sdk 提供的用户唯一标识，不同渠道间的 uid 可能发生重叠。

案例：

```

(void)smqcQpLoginResult:(NSNotification *)notify {
NSDictionary *userInfo = [notify userInfo];
int error = [userInfo objectForKey:@" error" ] intValue];
if (error == 0) {
    NSString *uid = [[SMPCQuickSDK sharedInstance] userId]; NSString *userNick
    = [[SMPCQuickSDK sharedInstance] userNick
];

```

```

        NSString *user_token = [[SMPCQuickSDK sharedInstance] userToken];
    }
}

```

2.2.注销接口

类名：

SMPCQuickSDK

函数：

- (int)logout;

功能：

主动注销当前用户。

返回值：

返回 0 说明调用接口成功，其他值产考说明。

注意：

调用该接口将引起 QuickSDK 发出一个注销通知 kSmpcQuickSDKNotiLogout。游戏开发

者收到注销通知后走下一步流程，比如回到游戏登录界面，再调用登录。

2.3.用户 uid

类名：

SMPCQuickSDK

函数：

- (NSString *)userId;

功能：

获取最后登录用户 uid，通常在收到登录通知回调中调用，该 uid 是渠道 sdk 提供的用户唯一

标识，不同渠道间的 uid 可能发生重叠。

返回值：最后登录用户的 uid，可能为空，在没有收到登录通知时可能已经有值，不能作为

判断当前是否有用户登录游戏的依据。

2.4.用户昵称

类名：

SMPCQuickSDK

函数：

- (NSString *)userNick;

2.5 用户 userToken

类名：SMPCQuickSDK

函数：- (NSString *)userToken

功能：获取最后一次登录用户的 userToken，用于到服务器验证用户。

返回值：最后一次登录用户的 userToken

2.6.更新角色信息（必接）

类名：

SMPCQuickSDK

函数：

- (void)updateRoleInfoWith:(SMPCQuickSDKGameRoleInfo *)info

isCreate:(BOOL)isCreate;

功能：

更新角色信息。在进入游戏，角色升级时调用

参数：

info 角色信息对象，参考 header 中 SMPCQuickSDKGameRoleInfo.h，isCreate 表示是否为刚刚创建的角色。

SMPCQuickSDKGameRoleInfo 属性如下所示：

```
@property (nonatomic, copy) NSString *serverId;           // 服务器 Id，必填
@property (nonatomic, copy) NSString *serverName;         // 服务器名称，必填
@property (nonatomic, copy) NSString *gameRoleName;       // 角色名，必填
@property (nonatomic, copy) NSString *gameRoleID;        // 角色 ID，必填
@property (nonatomic, copy) NSString *gameUserBalance;   // 玩家虚拟货币余额
必填，可随意
@property (nonatomic, copy) NSString *vipLevel;          // 玩家 vip 等级，必填
必填，可随意
@property (nonatomic, copy) NSString *gameUserLevel;     // 玩家等级，必填，
可随意
@property (nonatomic, copy) NSString *partyName;         // 公会名称，必填，
可随意
@property (nonatomic, copy) NSString *createTime;       // 角色创建时间(10
位时间戳)，必填，没有传 0
@property (nonatomic, copy) NSString *fightPower;       // 战力，可不传
@property (nonatomic, copy) NSString *other;            // 扩展参数
```

返回值:

3.充值接口

3.1.充值（必接）

类名：

SMPCQuickSDK

函数：

- (int)payOrderInfo:(SMPCQuickSDKPayOrderInfo *)orderInfo

roleInfo:(SMPCQuickSDKGameRoleInfo *)roleInfo;

功能:

订单支付

参数：

orderInfo 订单信息

roleInfo 玩家角色信息

说明：

orderInfo 中属性

goodId //商品 ID IAP 时为苹果开发者后台配置的商品 id，非 IAP 时随意，必填

productName //应该为一个通用名称，不包含商品个数，如“勾玉”，“元宝”，必填

cpOrderID //游戏订单号，必填

count //商品数量，如@"60"，必填

amount //总价 如@"6"，必填

callbackUrl 如果后台没有配置充值回调地址，callbackUrl 将作为充值回调地址。选填 extrasParams 透传字段，服务器回调时原样传递，选填。

异步充值，用户完成充值操作后会发送充值结果通知。kSmpcQuickSDKNotiRecharge，sdk 客户端通知的结果仅供参考，很多渠道没有客户端充值结果回调，以服务器端同步为准。

4.扩展接口

4.1.AppDelegate 事件调用 (必接)

说明：对能够使用 OC 接口的开发者这些回调应该比较容易添加，为了减轻大部分开发者负担，Unity3D 和 Cocos2d-x 的游戏不用接入下面事件调用的代码，打包工具将会自动添加。打包工在 Unity3d 和 cocos2d-x 默认系统生命周期 delegate 文件中添加如下代码，进行并不严格的排重，如果开发者修改了默认的 delegate 文件将导致添加失败。如果打开调试工程发现打包工具没能添加事件回调，需要开发者手动添加。


```

- (void)applicationWillResignActive:(UIApplication
    *)application { [[SMPCQuickSDK sharedInstance]
applicationWillResignActive:application];
}
- (void) applicationDidEnterBackground:(UIApplication *)application
    { [[SMPCQuickSDK sharedInstance]
applicationDidEnterBackground:application]; }

- (void) applicationWillEnterForeground:(UIApplication *)application
    { [[SMPCQuickSDK sharedInstance]
applicationWillEnterForeground:application];
}
- (void) applicationDidBecomeActive:(UIApplication
    *)application { [[SMPCQuickSDK sharedInstance]
applicationDidBecomeActive:application];
}
- (void) applicationWillTerminate:(UIApplication
    *)application { [[SMPCQuickSDK sharedInstance]
applicationWillTerminate:application];
}

- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken
{
    [[SMPCQuickSDK sharedInstance] application:application
didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

- (void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{
    [[SMPCQuickSDK sharedInstance] application:application
didFailToRegisterForRemoteNotificationsWithError:error];
}

- (UIInterfaceOrientationMask)application:(UIApplication
*)application supportedInterfaceOrientationsForWindow:(nullable
UIWindow *)window
{
    [[SMPCQuickSDK sharedInstance] application:application
supportedInterfaceOrientationsForWindow:window];
    return UIInterfaceOrientationMaskAll;
}

- (BOOL)application:(UIApplication
*)application handleOpenURL:(NSURL *)url{
    [[SMPCQuickSDK sharedInstance]
openURL:url application:application];
}

```

```

        return YES;
    }

    - (BOOL)application:(UIApplication *)application
    openURL:(NSURL *)url sourceApplication:(NSString
    *)sourceApplication annotation:(id)annotation{
        [[SMPCQuickSDK sharedInstance] openURL:url
        sourceApplication:sourceApplication
        application:application annotation:annotation];
        return YES;
    }

    - (BOOL)application:(UIApplication *)app openURL:(NSURL
    *)url options:(NSDictionary<NSString*, id> *)options{
        [[SMPCQuickSDK sharedInstance]
        openURL:url application:app options:options];
        return YES;
    }

    - (BOOL)application:(UIApplication
    *)application continueUserActivity:(NSUserActivity
    *)userActivity restorationHandler:(void(^)(NSArray
    * restorableObjects))restorationHandler{
        [[SMPCQuickSDK sharedInstance] application:application
        continueUserActivity:userActivity
        restorationHandler:restorationHandler];
        return YES;
    }
}

```

4.2.显示浮动菜单

类名：

SMPCQuickSDK

函数：

- (void)showToolBar:(SMPC_QUICK_SDK_TOOLBAR_PLACE)place;

功能：

显示浮动工具栏。若渠道无此对应接口，调用无效。

案例：

当用户登录成功后调用

```

[[SMPCQuickSDK sharedInstance]
showToolBar:SMPC_QUICK_SDK_TOOLBAR_TOP_LEFT];

```

4.3.隐藏浮动菜单

类名：

SMPCQuickSDK

函数：

- (void)hideToolBar;

功能：

隐藏浮动菜单。若渠道无此对应接口，调用无效。

案列：

必要时隐藏。

4.4.进入用户中心

类名：

SMPCQuickSDK

函数：

- (int)enterUserCenter;

功能：

进入用户中心。若渠道无此对应接口，调用无效。

4.5.进入客服中心

类名：

SMPCQuickSDK

函数：

- (int)enterCustomerCenter;

功能：

进入客服中心。

若渠道无此对应接口，直接调用会返回接口不支持的错误码。

4.6.进入 BBS

类名：

SMPCQuickSDK

函数：

- (int)enterBBS;

功能：

进入渠道 BBS。

若渠道无此对应接口，直接调用会返回接口不支持的错误码。

4.7.是否支持指定方法

类名：

SMPCQuickSDK

函数：

-(BOOL)isFunctionTypeSupported:(SMPC_QUICK_SDK_FUNC_TYPE)type;

功能：

判断渠道 SDK 是否支持某个接口，用户中心，客服中心，BBS，暂停等接口不是每个渠道 SDK 都实现了，所以可以在代码里判断来执行。若渠道无此对应接口，直接调用会返回接口不支持的错误码。

4.8 获取渠道实名信息（选接）

类名：

SMPCQuickSDK

函数：

```
- (NSDictionary *)realNameInfo;
```

功能：

获取渠道实名认证信息

返回值

类型:字典

kSmpcQuickSDKKeyRealName:是否实名认证 YES:已认证 NO:未认证

kSmpcQuickSDKKeyRealAge: 年龄如果渠道没有返回默认为-1

kSmpcQuickSDKKeyRealResumeGame: 渠道实名认证失败之后是否可以继续游戏
YES:可以 NO:不可以 渠道没返回默认 YES

kSmpcQuickSDKKeyRealOther:预留字段默认为空字符串

4.9.调起渠道实名认证（选接）

类名：

SMPCQuickSDK

函数：

```
- (int)realAuth;
```

功能：

调起渠道实名认证界面

若渠道无此对应接口，直接调用会返回接口不支持的错误码。

5.其他接口

- (int)channelType; //渠道唯一标识

- (NSString *)getConfigValue:(NSString *)key; //获取 quick 后台配置的自定义参数

五、SDK 通知说明

1.初始化通知

名称：

kSmpcQuickSDKNotiInitDidFinished

说明：

成功调用初始化接口后，异步。收到该通知后通过错误码判断渠道 SDK 是否初始化成功。

SMPC_QUICK_SDK_ERROR_NONE、SMPC_QUICK_SDK_ERROR_INIT_FAILED。

```
NSDictionary *userInfo = notify.userInfo;
```

```
int errorCode = [userInfo[kSmpcQuickSDKKeyError] intValue];
switch (errorCode) {
    case SMPC_QUICK_SDK_ERROR_NONE:
    {
        //初始化成功，进行下一步流程
    }
    break;
    case SMPC_QUICK_SDK_ERROR_INIT_FAILED:
    default:
    {
        //初始化失败
    }
    break;
}
```

2.登录成功通知

名称：

kSmpcQuickSDKNotiLogin

说明：

渠道 SDK 有用户登录成功时发出。如果游戏已经存在正在游戏的用户，收到该通知时认为

是切换账号，需要开发者注销游戏内用户，使用新用户信息进入游戏。

3.注销通知

名称：

kSmpcQuickSDKNotiLogout

说明：

这有 2 种情况

A) 用户从渠道 SDK 内的用户管理界面注销成功时会发出该通知。

B) 游戏调用 QuickSDK 的用户注销接口后会收到该通知，根据错误码判断注销成功或失败，

一般都认为是注销成功，不用考虑失败。

4.充值结果通知

名称：

kSmpcQuickSDKNotiRecharge

说明：

成功调用 QuickSDK 充值接口后会收到该通知。

通知结果的 3 种情况根据错误码来判断 (SMPC_QUICK_SDK_ERROR_NONE、SMPC_QUICK_SDK_ERROR_RECHARGE_FAILED、SMPC_QUICK_SDK_ERROR_RECHARGE_CANCELLED) 。

因为大部分渠道回调都以服务器为准，该结果仅供客户端参考，实际结果以服务器端同步

为准。

通知 userInfo 中信息说明：

//错误码

```
int error = [[userInfo objectForKey:kSmpcQuickSDKKeyError] intValue];
```

//QuickSDK 订单号、cp 下单时传入的订单号

```
NSString *orderId = userInfo[kSmpcQuickSDKKeyOrderId]; NSString
```

```
*cpOrderId = userInfo[kSmpcQuickSDKKeyCpOrderId];
```

5.暂停结束通知

名称：

kSmpcQuickSDKNotiPauseOver

说明：

不用监听，某些渠道 SDK 暂停结束时发出通知。

5.实名认证结果通知

名称：

kSmpcQuickSDKNotiRealAuth

说明：

调起渠道实名认证界面，实名认证成功后若渠道有通知则会收到该通知，反之则收不到该通知
通知 [userInfo](#) 中信息说明

kSmpcQuickSDKKeyRealName:是否实名认证 YES:已认证 NO:未认证

kSmpcQuickSDKKeyRealAge: 年龄如果渠道没有返回默认为-1

kSmpcQuickSDKKeyRealResumeGame: 渠道实名认证失败之后是否可以继续游戏
YES:可以 NO:不可以 渠道没返回默认 YES

kSmpcQuickSDKKeyRealOther:预留字段默认为空字符串

六、其他说明（必读）

1.接口说明

QuickSDK 初始化接口，用户接口，充值统一都支持。所有提供的接口都可以直接调用，有些可能会因为渠道不支持而调用无效。