

**信鸽iOS SDK**

**开发者指南**

腾讯（深圳）科技有限公司

（版权所有）

目录

[1. 简介 3](#_Toc424627323)

[2. 运行环境 3](#_Toc424627324)

[3. 功能 3](#_Toc424627325)

[3.1. 获取AppID和AppKey 3](#_Toc424627326)

[3.2. 工程配置 3](#_Toc424627327)

[4. API接口 4](#_Toc424627328)

[4.1. Debug开关 4](#_Toc424627329)

[4.2. 注册苹果推送服务 4](#_Toc424627330)

[4.3. 注册设备信息 4](#_Toc424627331)

[4.4. 初始化Push信息 5](#_Toc424627332)

[4.5. 设置标签 5](#_Toc424627333)

[4.6. 删除标签 6](#_Toc424627334)

[4.7. 推送被打开效果统计 6](#_Toc424627335)

[4.8. 注销设备 7](#_Toc424627336)

[4.9. 注销设备后重新注册 7](#_Toc424627337)

[4.10. 设置别名(帐号) 7](#_Toc424627338)

[4.11. 设置部分预置标签 8](#_Toc424627339)

[4.12. 本地推送 8](#_Toc424627340)

[4.13. 本地推送，App在前端运行时响应 9](#_Toc424627341)

[4.14. 删除本地推送，方法一 9](#_Toc424627342)

[4.15. 删除本地推送，方法二 10](#_Toc424627343)

[4.16. 清空本地推送 10](#_Toc424627344)

[4.17. 回调函数的说明 10](#_Toc424627345)

# 简介

信鸽iOS SDK是一个能够提供Push服务的开发平台，提供给开发者简便、易用的API接口，方便快速接入。

# 运行环境

iOS5.0 及以上版本。在程序启动时注册通知，参考以下代码，

- (void) registerNofitication {  
[[UIApplication sharedApplication] registerForRemoteNotificationTypes:(UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound)];  
}  
  
- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions  
{  
    [self registerNofitication];   
    return YES;  
}

# 功能

本SDK主要提供以下功能：

参考目录

## 获取AppID和AppKey

前往前台<http://xg.qq.com>注册并获取AppKey。

## 工程配置

1. 下载信鸽SDK压缩包到本地并解压；
2. 创建或打开Xcode iOS工程；
3. 将XGSetting.h、XGPush.h和libXG-SDK.a添加到Xcode工程；

添加对以下libraries的引用。包括CFNetwork.framework , SystemConfiguration.framework , CoreTelephony.framework , libz.dylib , libXG-SDK.a，Security.framework

如果是信鸽2.0以后的版本，还需要增加对libsqlite.dylib的引用

**iOS9配置:**

由于iOS9中的安全性做了增强,需要修改工程的plist允许http连接.

打开工程的plist文件,在dict字段增加以下内容

<key>NSAppTransportSecurity</key>

<dict>

<key>NSAllowsArbitraryLoads</key>

<true/>

<key>NSExceptionDomains</key>

<dict/>

</dict>

# API接口

|  |  |
| --- | --- |
| **类名** | **说明** |
| XGPush | Push服务，推送效果 |

注意: 在Console(控制台)里会看到”[xgpush]Disconnected.”为正常现象，在调用信鸽接口时候，如果服务器响应超时或者已经成功收到服务器的返回数据，手机会断开和服务器的连接。

## Debug开关

Debug开关控制控制台是否打印debug输出.

* **示例**
* //打开debug开关
* XGSetting \*setting = [XGSetting getInstance];
* [setting enableDebug:YES];
* //查看debug开关是否打开
* BOOL debugEnabled = [setting isEnableDebug];

## 注册苹果推送服务

在Demo中可以看到具体实现，对iOS8和之前的版本做了区分

* **示例**
* //注册Push服务，注册后才能收到推送
* //iOS8注册push方法
* #if \_\_IPHONE\_OS\_VERSION\_MAX\_ALLOWED >= \_IPHONE80\_
* [self registerPushForIOS8]; //具体实现参考Demo
* #else
* //iOS8之前注册push方法
* [[UIApplication sharedApplication] registerForRemoteNotificationTypes:(UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound)];
* #endif

## 注册设备信息

在application: didRegisterForRemoteNotificationsWithDeviceToken中调用。

**注意**:设备的deviceToken由苹果下发，可能会产生变化。同一设备在开发环境和生产环境是不相同的。

发起注册条件 1:未注册 2:account变更 3:距离上次注册成功超过15小时。达到以上任意条件，才能向后台进行网络请求，以及触发回调函数。

* **函数原型**
* +(NSString\*)registerDevice:(NSData \*)deviceToken;
* +(NSString \*)registerDeviceStr:(NSString \*)deviceToken; //字符串版本
* **参数**

deviceToken：

函数didRegisterForRemoteNotificationsWithDeviceToken的参数。

* **返回值**

NSString：获取到的deviceToken字符串

* **示例**
* - (void)application:(UIApplication \*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData \*)deviceToken {

//注册设备

NSString \* deviceTokenStr = [XGPush registerDevice: deviceToken];

//打印获取的deviceToken的字符串

NSLog(@"deviceTokenStr is %@",deviceTokenStr);

}

## 初始化Push信息

在application: didFinishLaunchingWithOptions中调用startWithAppkey接口，初始化。

* **函数原型**
* +(void)startApp:(uint32\_t)appId appKey:(NSString \*)appKey;
* **参数**

appId:

通过xg.qq.com注册app时得到的appID。

appkey：

通过xg.qq.com注册app时得到的key。

* **返回值**

无

* **示例**
* [XGPush startApp:101 appKey:@"akey"];

## 设置标签

开发者可以针对**不同的用户**设置标签。

* **函数原型**
* +(void)setTag:(NSString\*)tag;
* **参数**

tag：

字符串

* **返回值**

无

* **示例**
* NSString\* tag= @"age:10";

[XGPush setTag: tag];

## 删除标签

开发者可以删除设置过的标签。

* **函数原型**
* +(void)delTag:(NSString \*)tag;
* **参数**

tag：

字符串

* **返回值**

无

* **示例**
* NSDictionary\* tag= @"age:10";

[XGPush delTag: tag];

## 推送被打开效果统计

如果需要统计由信鸽推送的点击或者打开，需要开发者在didReceiveRemoteNotification(如果没有则按照下面的示例手动添加)中调用handleReceiveNotification , 然后在didFinishLaunchingWithOptions中调用handleLaunching。

* **函数原型**

+(void)handleReceiveNotification:(NSDictionary \*)userInfo;//app在前台运行时

+(void)handleLaunching:(NSDictionary \*)launchOptions;//app不在前台运行时，点击推送激活时

* **参数**

事件本身默认的参数名

* **返回值**

无

* **示例**

- (void)application:(UIApplication\*)application didReceiveRemoteNotification:(NSDictionary\*)userInfo  
{  
  [XGPush handleReceiveNotification:userInfo];  
}

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

[XGPush handleLaunching: launchOptions];

}

## 注销设备

注销设备,此台设备不接收推送。

* **函数原型**
* +(void)unRegisterDevice;
* **参数**

无

* **返回值**

无

* **示例**

 [XGPush unRegisterDevice];

## 注销设备后重新注册

如果注销设备后，需要重新注册设备，需要先调用以下函数。

+(void)initForReregister:(void (^)(void)) successCallback;

示例:游戏退出是，注销了设备。然后在下次启动时再次注册设备。

[XGPush startApp:2200022728 appKey:@"IMJ34Y25JN4I"];

void (^successCallback)(void) = ^(void){

//中间部分参考Demo中的代码

};

[XGPush initForReregister:successCallback];

## 设置别名(帐号)

上报用户的别名(帐号)，以便支持按别名(帐号)推送。

在初始化信鸽后，注册设备之前调用。调用完之后需要调用RegisterDevice函数。

注销帐号:传入\*作为参数，见下文示例。

* **函数原型**
* +(void)setAccount:(NSString \*)account;
* **参数**

account:

用户的别名(帐号)

* **返回值**

无

* **示例**

 [XGPush setAccount:@"123456"];

//[XGPush setAccount:@"\*"]; //注销帐号

//然后调用registerDevice函数注册设备

## 设置部分预置标签

部分预置标签是通过用户设置的，例如渠道和游戏大区等。在XGSetting.h中可以看到可以设置的属性。

* **示例**

[[XGSetting getInstance] setChannel:@"appstore"];

[[XGSetting getInstance] setGameServer:@"巨神峰"];

## 本地推送

在指定时间进行本地推送。

* **函数原型**
* +(void)localNotification:(NSDate \*)fireDate alertBody:(NSString \*)alertBody badge:(int)badge alertAction:(NSString \*)alertAction userInfo:(NSDictionary \*)userInfo;
* **参数**

fireDate:

推送触发的时间

alertBody:

推送的内容

badge:

改变App的角标，如果为-1，则不改变。

alertAction:

推送的右侧按钮的文字，默认为“启动”。

userInfo:

自定义的一组key/value信息

* **返回值**

无

* **示例**

 [XGPush localNotification:fireDate alertBody:@"测试本地推送" badge:2 alertAction:@"确定" userInfo:userInfo];

## 本地推送，App在前端运行时响应

App在前台运行时，苹果的推送默认不弹窗。通过此接口可进行弹窗。

* **函数原型**
* +(void)localNotificationAtFrontEnd:(UILocalNotification \*)notification userInfoKey:(NSString \*)userInfoKey userInfoValue:(NSString \*)userInfoValue;
* **参数**

userInfoKey:

标识本地推送的Key

userInfoValue:

标识本地推送的Value

* **返回值**

无

* **示例**

 [XGPush localNotificationAtFrontEnd:notification userInfoKey:@"clockID" userInfoValue:@"myid"];

## 删除本地推送，方法一

iOS最多允许一个App存储64个本地推送。以下函数可以进行删除。

* **函数原型**
* +(void)delLocalNotification:(NSString \*)userInfoKey userInfoValue:(NSString \*)userInfoValue;
* **参数**

userInfoKey:

标识本地推送的Key

userInfoValue:

标识本地推送的Value

* **返回值**

无

* **示例**

 [XGPush delLocalNotification:@"clockID" userInfoValue:@"myid"];

## 删除本地推送，方法二

iOS最多允许一个App存储64个本地推送。以下函数可以进行删除。

* **函数原型**
* +(void)delLocalNotification:(UILocalNotification \*)myUILocalNotification;
* **参数**

myUILocalNotification:

本地推送对象

* **返回值**

无

* **示例**

 [XGPush delLocalNotification:notification];

## 清空本地推送

iOS最多允许一个App存储64个本地推送。以下函数可以进行清空。

* **函数原型**
* [XGPush clearLocalNotifications];
* **参数**

无

* **返回值**

无

* **示例**

 [XGPush clearLocalNotifications];

## 回调函数的说明

从2.1.0(含)版本之后，带有网络请求的接口都增加了回调函数，包括成功时的回调succussCallback和失败时的回调errorCallback。2.1.0之前的版本可以通过Console里的rspCode判断结果，为0代表成功。

* **函数原型**
* succussCallback:(void (^)(void)) succussCallBack
* errorCallback:(void (^)(void)) errorCallback
* **返回值**

无

* **示例**

  void (^succussBlock)(void) = ^(void){

//成功之后的处理

NSLog(@"[xgpush]register succussBlock ,deviceToken: %@",deviceTokenStr);

};

void (^errorBlock)(void) = ^(void){

//失败之后的处理

NSLog(@"[xgpush]register errorBlock");

};

//注册设备

[XGPush registerDevice:deviceToken succussCallback:succussBlock errorCallback:errorBlock];