

# iOS SDK 集成指南

## 1. 概述

### 1.1. 集成压缩包内容：

我们提供的一个 SDK 开发工具包，包含了 iOS SDK 的全部所需资源。

### 1. 解压缩后的文件目录结构

- GtSdkDemo：SDK 演示 Demo，能更好的展示个推 SDK 功能点。
- GtSdkDemo-objc：objc 集成 Demo，方便 objc 开发者集成个推 SDK。
- GtSdkDemo-swift：Swift 集成 Demo，方便 Swift 开发者集成个推 SDK。
- GtSdkLib：

1、包含集成 SDK 所需的静态库和头文件。

2、使用情况：

(1)、在 App 内投放广告，获取 IDFA 可通过苹果审核。

(2)、App 内无广告，但由于先前投放的特定广告，可参考如下勾选，通过苹果审核。

勾选如图：

广告标识符

此 App 是否使用广告标识符 (IDFA)?

☒ 是 ☐ 否

广告标识符 (IDFA) 是每台 iOS 设备的唯一 ID，是投放定向广告的唯一方法。用户可以选择在其 iOS 设备上限制广告定位。

如果您的 App 使用广告标识符，请在提交您的代码（包括任何第三方代码）之前进行检查，以确保您的 App 仅出于下面列出的目的使用广告标识符，并尊重“限制广告跟踪”设置。如果您在 App 中加入了第三方代码，则您将对此类代码的行为负责。因此，请务必与您的第三方提供商核实，确认此类代码是否遵循广告标识符和“限制广告跟踪”设置的使用限制。

此 App 使用广告标识符来实现以下目的（选择所有适用项）：

- ☐ 在 App 内投放广告
- ☒ 标明此 App 安装来自先前投放的特定广告
- ☒ 标明此 App 中发生的操作来自先前投放的广告

如果您认为自己还有其他可以接受的广告标识符使用方式，请[联系我们](#)。

iOS 中的“限制广告跟踪”设置

- ☒ 本人，在此确认，此 App（以及与此 App 交互的任何第三方）使用广告标识符检查功能并尊重用户在 iOS 中的“限制广告跟踪”设置。当用户启用广告标识符后，此 App 不会用于 [iOS 开发人员计划许可协议](#) 中规定的“有限广告目的”之外的任何目的，以任何方式使用广告标识符，以及通过使用广告标识符获取的任何信息。

对于广告标识符 (IDFA) 的使用，请务必选择正确的答案。如果您的 App 包含 IDFA 而您选择了“否”，此二进制文件将永久被拒绝，您必须提交另一个二进制文件。

**注意: 获取 IDFA 需添加 AdSupport.framework 库支持**

3、libGtExtensionSdk 为 APP 扩展模块 SDK 包，支持 iOS10 APNs 通知展示数统计。

- GtSdkLib-noidfa:

1、包含集成 SDK 所需的静态库和头文件。

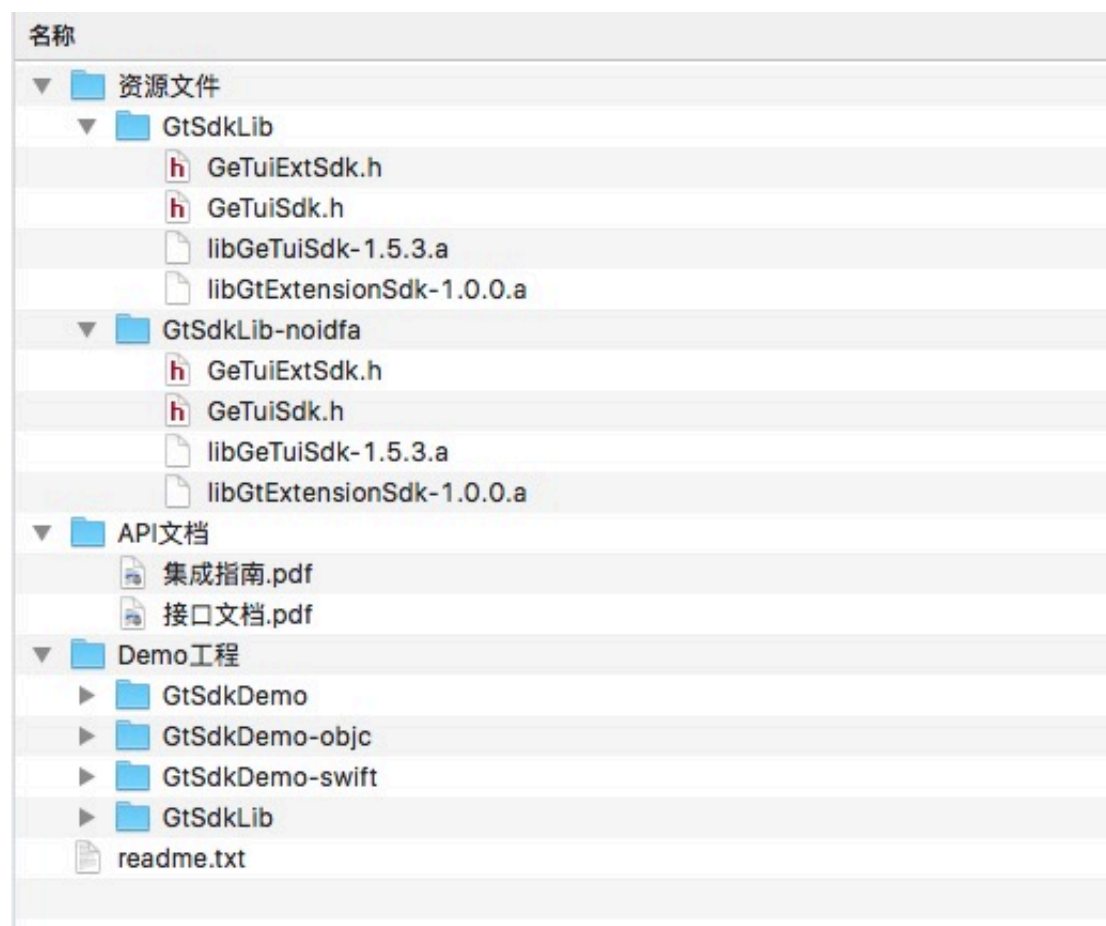
2、使用情况:

由于 IDFA(identifier for advertising) 能够较精准的识别用户，尤其对于广告主追踪广告转化率提供了很大帮助，在 App 内无广告情况下还是建议开发者使用获取 IDFA 版本，并参考(2)中所说的方式提交 AppStore 审核，当然开发者不想使用 IDFA 或者担忧采集 IDFA 而未集成任何广告服务遭到 Apple 拒绝，我们也准备了该无 IDFA 版本供开发者集成。

**注意: 不获取 IDFA 需删除 AdSupport.framework 库支持**

3、libGtExtensionSdk 为 APP 扩展模块 SDK 包，支持 iOS10 APNs 通知展示数统计。

## 2. 资源内容图示



**注意: libGeTuiSdk-{version}.a** (version为具体的sdk版本号) 使用 libo 工具将 支持i386、x86\_64、arm64、armv7的代码打包到了一起，所以这个库将同时支持 simulator 和 device，支持 iOS 版本为7.0及以上。

## 2. 项目设置

## 2.1 个推SDK头文件和.a库设置

将 GtSdkLib 目录拷贝到项目工程目录下，导入 GtSdkLib 目录所有的头文件、libGeTuiSdk-{version}.a 文件和几个系统库到 XCode 项目中。

添加头文件搜索目录













注：头文件搜索目录根据不同项目或目录结构不同会有不一样，请开发者根据自己项目需求自行修改。



## 2.2 添加依赖库（必须，如下图）

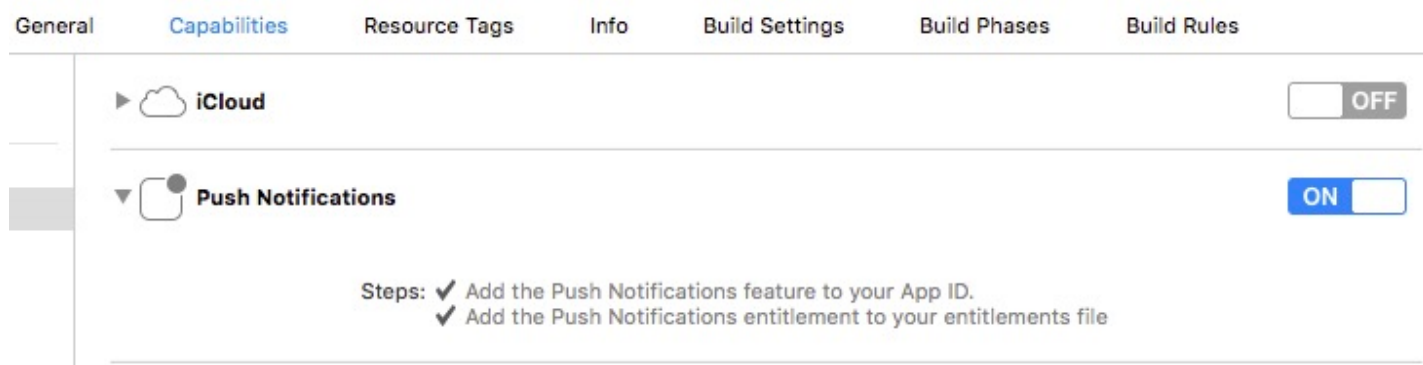
添加系统库支持：

- libc++.tbd
- libz.tbd
- libsqlite3.tbd
- Security.framework
- MobileCoreServices.framework
- SystemConfiguration.framework
- CoreTelephony.framework
- AVFoundation.framework
- JavaScriptCore.framework
- CoreLocation.framework
- UserNotifications.framework (iOS 10 及以上)
- AdSupport.framework （使用无IDFA版本接入需删除该 AdSupport 库）

 libc++.tbd	Required ⇅
 libz.tbd	Required ⇅
 libsqlite3.tbd	Required ⇅
 MobileCoreServices.framework	Required ⇅
 SystemConfiguration.framework	Required ⇅
 CoreTelephony.framework	Required ⇅
 AVFoundation.framework	Required ⇅
 JavaScriptCore.framework	Required ⇅
 CoreLocation.framework	Required ⇅
 Security.framework	Required ⇅
 UserNotifications.framework	Required ⇅
 AdSupport.framework	Required ⇅

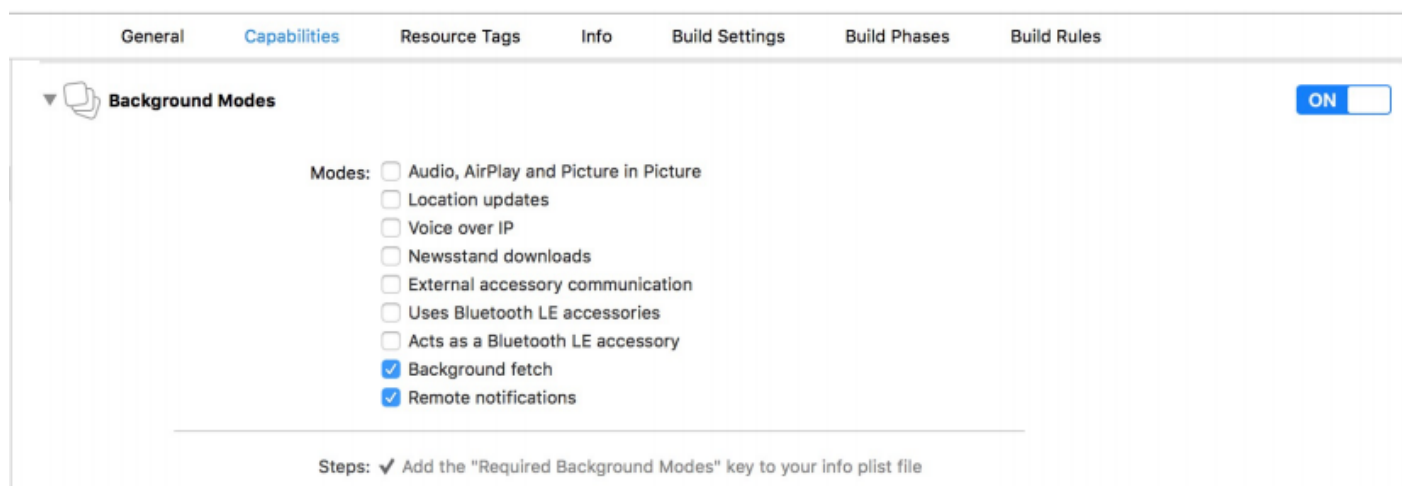
## 2.3 Xcode 中开启推送功能（Xcode8 上必开）

开启 Target -> Capabilities -> Push Notifications。如果没有开启，在 Xcode8 上编译中将获取不到 DeviceToken。



## 2.4 SDK后台运行权限设置

为了更好支持SDK 推送，APP定期抓取离线数据，需要配置后台运行权限：Background fetch：后台获取 Remote notifications：推送唤醒（静默推送，Silent Remote Notifications）



## 3. 基本集成

### 3.1 AppDelegate 中注册 GeTuiSdkDelegate

```

#import <UIKit/UIKit.h>
#import "GeTuiSdk.h"          // GetuiSdk头文件，需要使用地方需要添加此代码

// iOS10 及以上需导入 UserNotifications.framework
#if __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_10_0
#import <UserNotifications/UserNotifications.h>
#endif

/// 个推开发者网站中申请App时，注册的AppId、AppKey、AppSecret
#define kGtAppId              @"iMahVVxurw6BNr7XSn9EF2"
#define kGtAppKey              @"yIPfqwq6OMApp6dkqgLpG5"
#define kGtAppSecret           @"G0aBqAD6t79JfzTB6Z5lo5"

/// 需要使用个推回调时，需要添加"GeTuiSdkDelegate"
/// iOS 10 及以上，需要添加 UNUserNotificationCenterDelegate 协议，才能使用 UserNotifi
cations.framework 的回调
@interface AppDelegate : UIResponder <UIApplicationDelegate, GeTuiSdkDelegate, UN
UserNotificationCenterDelegate>

```

## 3.2 App运行时启动个推 SDK 并注册 APNs

在 AppDelegate didFinishLaunchingWithOptions 方法中:通过平台分配的 AppId、AppKey、AppSecret 启动个推 SDK，并完成注册 APNs 通知和处理启动时拿到的 APNs 透传数据。

```

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(
NSDictionary *)launchOptions {
    // 通过个推平台分配的appId、 appKey 、 appSecret 启动SDK，注：该方法需要在主线程中调用
    [GeTuiSdk startSdkWithAppId:kGtAppId appKey:kGtAppKey appSecret:kGtAppSecret
delegate:self];
    // 注册 APNs
    [self registerRemoteNotification];
    return YES;
}

```

**注：注册APNs获取DeviceToken不同项目或版本会有所不同，可以参考如下方式注册APNs。**

```

/** 注册 APNs */
- (void)registerRemoteNotification {
    /*
    警告：Xcode8 的需要手动开启“TARGETS -> Capabilities -> Push Notifications”
    */

    /*
    警告：该方法需要开发者自定义，以下代码根据 APP 支持的 iOS 系统不同，代码可以对应修改。
    以下为演示代码，注意根据实际需要修改，注意测试支持的 iOS 系统都能获取到 DeviceToken
    */
}

```

```

    */
    if ([[UIDevice currentDevice].systemVersion floatValue] >= 10.0) {
#ifdef __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_10_0 // Xcode 8编译会调用
        UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
        center.delegate = self;
        [center requestAuthorizationWithOptions:(UNAuthorizationOptionBadge | UNAuthorizationOptionSound | UNAuthorizationOptionAlert | UNAuthorizationOptionCarPlay) completionHandler:^(BOOL granted, NSError *_Nullable error) {
            if (!error) {
                NSLog(@"request authorization succeeded!");
            }
        }];

        [[UIApplication sharedApplication] registerForRemoteNotifications];
#else // Xcode 7编译会调用
        UIUserNotificationType types = (UIUserNotificationTypeAlert | UIUserNotificationTypeSound | UIUserNotificationTypeBadge);
        UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:types categories:nil];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
        [[UIApplication sharedApplication] registerUserNotificationSettings:settings];
#endif
    } else if ([[UIDevice currentDevice].systemVersion floatValue] >= 8.0) {
        UIUserNotificationType types = (UIUserNotificationTypeAlert | UIUserNotificationTypeSound | UIUserNotificationTypeBadge);
        UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:types categories:nil];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
        [[UIApplication sharedApplication] registerUserNotificationSettings:settings];
    } else {
        UIRemoteNotificationType apn_type = (UIRemoteNotificationType)(UIRemoteNotificationTypeAlert |
                                                                           UIRemoteNotificationTypeSound |
                                                                           UIRemoteNotificationTypeBadge);
        [[UIApplication sharedApplication] registerForRemoteNotificationTypes:apn_type];
    }
}

```

### 3.3 向个推服务器注册DeviceToken

免除开发者管理 DeviceToken 的麻烦，需要向 GeTui Server 上报 DeviceToken。并可通过个推开发者平台

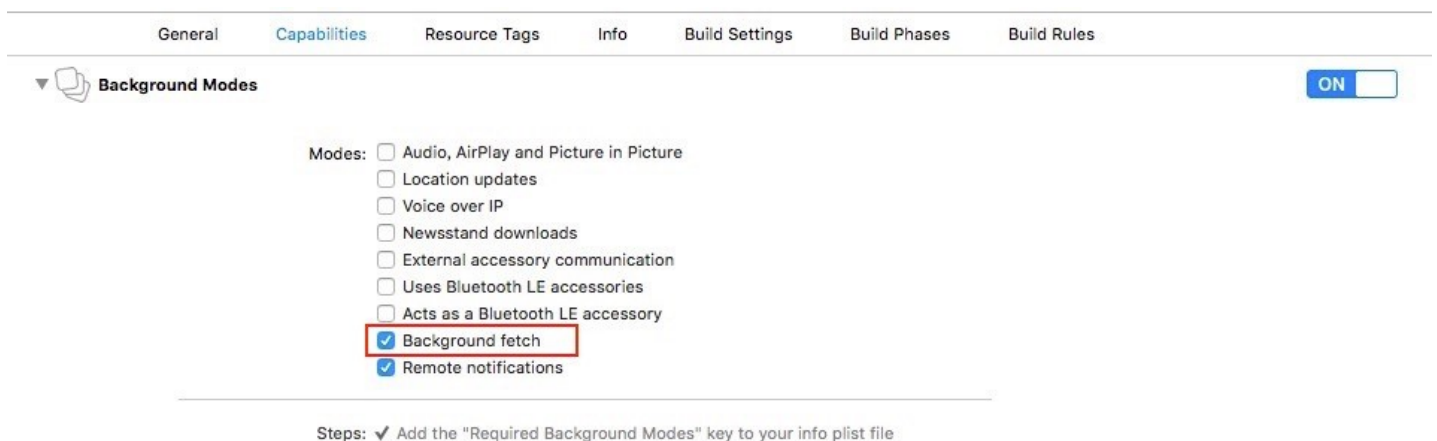
推送 APN 消息。

```
/** 远程通知注册成功委托 */
- (void)application:(UIApplication *)application didRegisterForRemoteNotifications
WithDeviceToken:(NSData *)deviceToken {
    NSString *token = [[deviceToken description] stringByTrimmingCharactersInSet:[
    NSStringCharacterSet characterSetWithCharactersInString:@"<>"]];
    token = [token stringByReplacingOccurrencesOfString:@" " withString:@""];
    NSLog(@"\n>>>[DeviceToken Success]:%@",token);

    //向个推服务器注册deviceToken
    [GeTuiSdk registerDeviceToken:token];
}
```

### 3.4 Background Fetch 接口回调

注: iOS7.0 以后支持 APP 后台刷新数据, 会回调 `performFetchWithCompletionHandler` 接口, 此处为保证个推数据刷新需调用`[GeTuiSdk resume]` 接口恢复个推 SDK 运行刷新数据。



```
- (void)application:(UIApplication *)application performFetchWithCompletionHandle
r:(void (^)(UIBackgroundFetchResult))completionHandler {
    /// Background Fetch 恢复SDK 运行
    [GeTuiSdk resume];
    completionHandler(UIBackgroundFetchResultNewData);
}
```

### 3.5 统计远程推送消息

iOS 10 以前, 处理 APNs 展示点击, 统计有效用户点击数, 在下面回调中调用处理方法:

```

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler {
    // 将收到的APNs信息传给个推统计
    [GeTuiSdk handleRemoteNotification:userInfo];
    completionHandler(UIBackgroundFetchResultNewData);
}

```

iOS 10 中，处理 APNs 展示点击，统计有效用户点击数，需先添加 `UNUserNotificationCenterDelegate` 后，在 `AppDelegate.m` 实现的 `didReceiveNotificationResponse` 方法中调用处理方法：

```

#ifdef __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_10_0

// iOS 10: App在前台获取到通知
- (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNotification:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotificationPresentationOptions))completionHandler {

    NSLog(@"willPresentNotification: %@", notification.request.content.userInfo);

    // 根据APP需要，判断是否要提示用户Badge、Sound、Alert
    completionHandler(UNNotificationPresentationOptionBadge | UNNotificationPresentationOptionSound | UNNotificationPresentationOptionAlert);
}

// iOS 10: 点击通知进入App时触发，在该方法内统计有效用户点击数
- (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNotificationResponse:(UNNotificationResponse *)response withCompletionHandler:(void (^)(UNNotificationPresentationOptions))completionHandler {

    NSLog(@"didReceiveNotification: %@", response.notification.request.content.userInfo);

    // [ GTSdk ]: 将收到的APNs信息传给个推统计
    [GeTuiSdk handleRemoteNotification:response.notification.request.content.userInfo];

    completionHandler();
}

#endif

```

### 3.6 GeTuiSdk注册回调，获取CID信息

在不确定是否启动个推 SDK 成功，可以通过回调查看注册结果



```
/** SDK启动成功返回cid */
- (void)GeTuiSdkDidRegisterClient:(NSString *)clientId {
    //个推SDK已注册，返回clientId
    NSLog(@"\n>>>[GeTuiSdk RegisterClient]:%@", clientId);
}

/** SDK遇到错误回调 */
- (void)GeTuiSdkDidOccurError:(NSError *)error {
    //个推错误报告，集成步骤发生的任何错误都在这里通知，如果集成后，无法正常收到消息，查看这里的
    通知。
    NSLog(@"\n>>>[GexinSdk error]:%@", [error localizedDescription]);
}
```

## 4. 高级功能

---

### 4.1 使用个推 SDK 透传消息, 由个推通道下发 (非 APNs)

SDK 在线状态时（App 在前台运行），个推服务器会直接给您的 App 发送透传消息，不发送苹果 APNs 消息，可以更快的把消息发送到手机端；SDK 离线状态时（停止 SDK 或 App 后台运行 或 App 停止），个推服务器会给 App 发送苹果 APNs 消息，同时保存个推的离线消息，当 SDK 在线后，SDK 会获取所有的个推透传消息，offLine 字段就是表明该条消息是否为离线消息。

```

/** SDK收到透传消息回调 */
- (void)GeTuiSdkDidReceivePayloadData:(NSData *)payloadData andTaskId:(NSString *)
taskId andMsgId:(NSString *)msgId andOffLine:(BOOL)offLine fromGtAppId:(NSString *)
)appId {
    //收到个推消息
    NSString *payloadMsg = nil;
    if (payloadData) {
        payloadMsg = [[NSString alloc] initWithBytes:payloadData.bytes
                                                    length:payloadData.length
                                                    encoding:NSUTF8StringEncoding];
    }

    NSString *msg = [NSString stringWithFormat:@"taskId=%@,messageId=%@,payloadMs
g:%@%@",taskId,msgId, payloadMsg,offLine ? @"<离线消息>" : @""];
    NSLog(@"\n>>>[GexinSdk ReceivePayload]:%@\\n\\n", msg);

    /**
    *汇报个推自定义事件
    *actionId: 用户自定义的actionid, int类型, 取值90001-90999。
    *taskId: 下发任务的任务ID。
    *msgId: 下发任务的消息ID。
    *返回值: BOOL, YES表示该命令已经提交, NO表示该命令未提交成功。注: 该结果不代表服务器收到该条
命令
    */
    [GeTuiSdk sendFeedbackMessage:90001 andTaskId:taskId andMsgId:msgId];
}

```

注：个推透传获取的消息内容为下图中“消息内容”

## 4.2 苹果官方静默推送

如果需要使用推送唤醒/APNs 透传/静默推送 (Remote Notifications) “content-available:1”,需要配置

- Modes:
- ☐ Audio, AirPlay and Picture in Picture
  - ☐ Location updates
  - ☐ Voice over IP
  - ☐ Newsstand downloads
  - ☐ External accessory communication
  - ☐ Uses Bluetooth LE accessories
  - ☐ Acts as a Bluetooth LE accessory
  - ☒ Background fetch
  - ☒ Remote notifications

Steps: ✓ Add the "Required Background Modes" key to your info plist file

```
/** APP已经接收到“远程”通知(推送) - 透传推送消息 */
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult result))completionHandler {
    // 处理APNs代码, 通过userInfo可以取到推送的信息(包括内容, 角标, 自定义参数等)。如果需要弹窗等其他操作, 则需要自行编码。
    NSLog(@"\n>>>[Receive RemoteNotification - Background Fetch]:%@",userInfo);
    completionHandler(UIBackgroundFetchResultNewData);
}
```

## 4.3 指定标签推送

用户设置标签, 标示一组标签用户, 可以针对该标签用户进行推送

```
NSString *tagName = @"个推,推送,iOS";
NSArray *tagNames = [tagName componentsSeparatedByString:@","];
if (![GeTuiSdk setTags:tagNames]) {
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Failed" message:@"设置失败" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alertView show];
}
```

## 4.4 设置别名, 别名推送

对用户设置别名, 可以针对具体别名进行推送

```
// 绑定别名
[GeTuiSdk bindAlias:@"个推" andSequenceNum:@"seq-1"];
// 取消绑定别名
[GeTuiSdk unbindAlias:@"个推" andSequenceNum:@"seq-2"];
```

处理 绑定/解绑 返回:

```

- (void)GeTuiSdkDidAliasAction:(NSString *)action result:(BOOL)isSuccess sequenceNum:(NSString *)aSn error:(NSError *)aError {
    if ([kGtResponseBindType isEqualToString:action]) {
        NSLog(@"绑定结果 : %@ !, sn : %@", isSuccess ? @"成功" : @"失败", aSn);
        if (!isSuccess) {
            NSLog(@"失败原因: %@", aError);
        }
    } else if ([kGtResponseUnBindType isEqualToString:action]) {
        NSLog(@"绑定结果 : %@ !, sn : %@", isSuccess ? @"成功" : @"失败", aSn);
        if (!isSuccess) {
            NSLog(@"失败原因: %@", aError);
        }
    }
}
}

```

## 4.5 设置角标

badge是iOS用来标记应用程序状态的一个数字，出现在程序图标右上角。sdk封装badge功能，允许应用上传badge值至个推服务器，由个推后台帮助开发者管理每个用户所对应的推送badge值，简化了设置推送badge的操作。实际应用中，开发者只需将变化后的Badge值通过setBadge接口同步个推服务器，无需自己维护用户与badge值之间的对应关系，方便运营维护。

支持版本: v1.4.1及后续版本

```

[GeTuiSdk setBadge:badge]; //同步本地角标值到服务器
[[UIApplication sharedApplication] setApplicationIconBadgeNumber:badge]; //APP 显示角标需开发者调用系统方法进行设置

```

## 4.6 重置角标

重置角标, 重置服务器角标计数，计数变更为0。

支持版本: v1.4.1及后续版本

```

[GeTuiSdk resetBadge]; //重置角标计数
[[UIApplication sharedApplication] setApplicationIconBadgeNumber:0]; // APP 清空角标

```

## 4.7 设置渠道

设置渠道信息，方便服务器根据渠道统计信息。

支持版本: v1.5.0及后续版本

```

[GeTuiSdk setChannelId:@"GT-Channel"];

```

## 5. iOS 10 Notification Service Extension 集成

---

Apple 在 iOS 10 中新增了 Notification Service Extension，可在消息送达时进行业务处理。为精确统计消息送达率，在集成 GetuiSdk 时，可以添加 Notification Service Extension，并在 Extensions 中添加 GtExtensionSdk 的统计接口，实现消息展示回执统计功能。

注意：该统计APNs展示回执统计功能目前仅支持生产环境证书且通过[个推管理控制台](#)下发的透传消息。API推送任务的APNs展示回执统计功能将在后续提供。



应用证书： ent.com.getui.demo.dis.p12

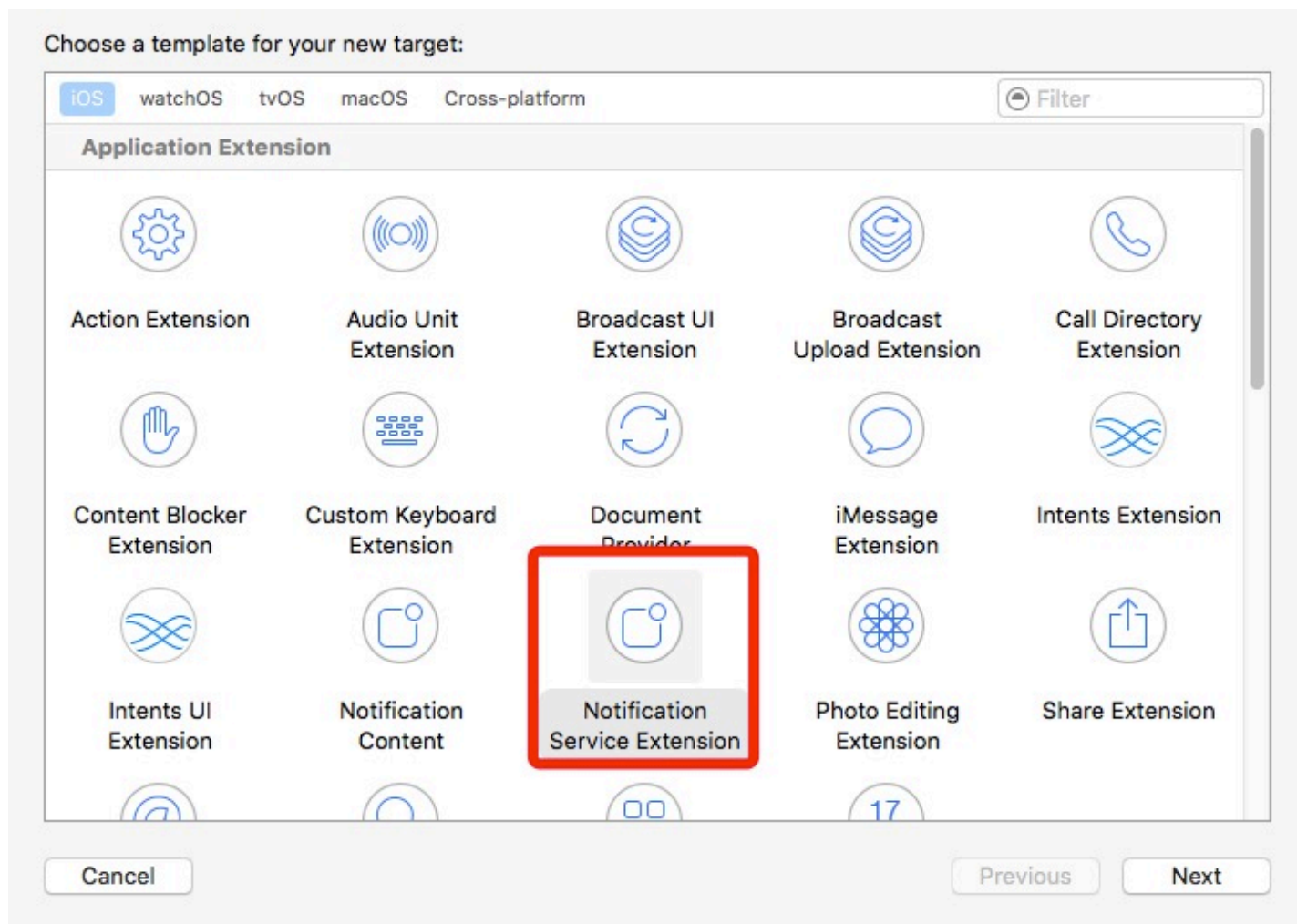
证书密码： \*\*\*\*\*

证书环境： 生产环境

证书日期： 2016年2月16日 10:49:18 - 2017年3月17日 10:49:18

### 5.1 添加 Notification Service Extension

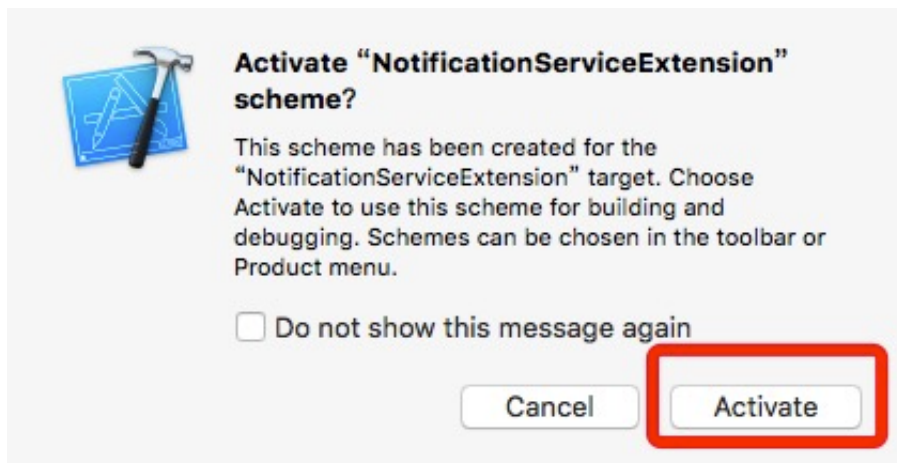
打开 Xcode 8，选择 File -> New -> Target -> Notification Service Extension



这里需要注意以下两点：

- Service Extension 的 Bundle Identifier 不能和 Main Target（也就是你自己的 App Target）的 Bundle Identifier 相同，否则会报 BundleID 重复的错误。
- Service Extension 的 Bundle Identifier 需要在 Main Target 的命名空间下，比如说 Main Target 的 BundleID 为 ent.getui.xxx，那么 Service Extension 的 BundleID 应该类似与 ent.getui.xxx.yyy 这样的格式。如果不这么做，会引起命名错误。

添加 Notification Service Extension 后会生成相应的 Target。添加后会弹出是否激活该 Target 对应 scheme 的选项框，选择 Activate，如果没有弹出该选项框，需要自行添加相应的 scheme。如下图：



Notification Service Extension 添加成功后会在项目中自动生成 `NotificationService.h` 和 `NotificationService.m` 两个类，包含以下两个方法：

```
didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:
(void (^)(UNNotificationContent *contentToDeliver))contentHandler
```

我们可以在这个方法中处理我们的 APNs 通知，并个性化展示给用户。APNs 推送的消息送达时会调用这个方法，此时你可以对推送的内容进行处理，然后使用 `contentHandler` 完成这次处理。但是如果处理时间过长，通知将会进入第二方法进行最后的紧急处理。

```
serviceExtensionTimeWillExpire
```

如果第一个方法在限定时间内没有调用 `contentHandler` 返回，则会在过期之前进行回调本方法，此时你可以对你的 APNs 消息进行紧急处理后展示，如果没有处理，则显示原始 APNs 推送。

## 5.2 添加支持 GtExtensionSdk 的依赖库

选择 Notification Service Extension 所对应的 Target，添加如下依赖库支持：

Name	Status
 libz.tbd	Required ⌵
 libsqlite3.tbd	Required ⌵
 libGtExtensionSdk-1.0.0.a	Required ⌵
+ -	

- libz.tbz
- libsqlite3.tbd
- libGtExtensionSdk.a

**注意：**需设置 NotificationService 最低运行版本 10.0 。

Deployment Target 10.0

Devices Universal

Main Interface

## 5.3 使用 GtExtensionSdk 推送统计回执功能

在 `NotificationService.m` 文件中，导入 `GtExtensionSdk` 的头文件：

```
#import "GeTuiExtSdk.h"
```

`NotificationService.m` 中有两个回调方法 `didReceiveNotificationRequest`（接收到通知但未展示前回调，可在通知展示前修改通知）和 `serviceExtensionTimeWillExpire`（处理超时回调）。需要在 `didReceiveNotificationRequest` 中添加 `GtExtensionSdk` 的推送回执统计处理，由于回执操作是异步请求，因此待展示推送的回调处理需要放到回执完成的回调中：

```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContent
Handler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {

    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];

    //使用 handelNotificationServiceRequest 上报推送送达数据。
    [GeTuiExtSdk handelNotificationServiceRequest:request withComplete:^(

        self.bestAttemptContent.title = [NSString stringWithFormat:@"%s@ [modified]",
self.bestAttemptContent.title];
        // 待展示推送的回调处理需要放到回执完成的回调中
        self.contentHandler(self.bestAttemptContent);
    )];
}
```

此外，由于网络延迟等原因，该统计方法的回调有最多只有 5s 的延迟，因此不需要担心消息无法送达。

## 5.4 带 Notification Service Extension 项目上传 AppStore 须知

**注意：**使用了 Notification Service Extension 的项目中，在制作待上传至 AppStore 的 iPA 包时，编译设备需要选择 `Generic iOS Device`（如下图所示），然后再进行 `Product -> Archive` 操作。只有选择 `Generic iOS Device` 才能打包编译出带有 Notification Service Extension 且适配全机型设备的 iPA 包。





## 6. 使用 CocoaPods 集成

### 6.1 安装 Cocopods

安装方式异常简单，Mac 下都自带 ruby，使用 ruby 的 gem 命令即可下载安装：

```
$ sudo gem install cocoapods
$ pod setup
```

### 6.2 配置 Cocopods Podfile 文件，导入 GTSDK

使用时需要新建一个名为 Podfile 的文件，以如下格式，将依赖的库名字依次列在文件中即可：

```
platform :ios
pod 'GTSDK'
```

如果需要使用 不获取 IDFA 版本的库，请如下配置：

```
platform :ios
pod 'GTSDK', '1.5.3-noidfa'
```

### 6.3 执行 Install，完成GTSDK 导入

然后将编辑好的 Podfile 文件放到你的项目根目录中，执行如下命令即可：

```
$ cd "your project home"
$ pod install
```

注：CocoaPods 详细使用参考[“CocoaPods安装和使用”](#)

### 6.4 SDK 代码接入

注：请查看第二点 基本使用 和 第三点 高级使用 完成 SDK 接入。

## 7. 推送流程

iOS应用、Server、getui SDK、getui Server、Apple Push Notification Server的交互过程，如下图

