

# XCode集成

## 前言

- 开发环境下手动方式导入SDK资源进行集成的步骤，配置相对复杂，需要仔细阅读文档和Demo工程。
- 个推在国内首创APNs展示统计功能，可精确统计通过苹果APNs服务下发的通知数量，开发者可以获知更为精准的推送展示数和通知点击率，为运营提供了更强大的数据支持。
- 个推多媒体推送功能支持，可通过后台推送多媒体资源，支持图片，音频，视频的展示。
- 本文档适用SDK版本：2.3.0.0 及以后
- 请参考 `GtSdkDemo` Demo工程

## 1. 创建个推应用

- 请登录 <http://dev.getui.com>，选择 `登记应用` 并填写应用名称和包名信息，完成应用创建：



## 登记应用 ?

\* 应用图标：



请上传25K大小以内的图片

\* 应用名称：

\* 应用类型：

其他 ▼

\* 应用平台：



## 参数设置

Android

\* 应用标识：



取消

确定

- 点击 **应用配置**，获取到相应的 **AppID**、**AppKey**、**AppSecret** 信息：



## 2. 打开项目工程

- 启动XCode，打开您之前创建的iOS项目工程：



## 3. 添加个推SDK及相关配置

### 3.1 个推iOS SDK资料包结构

```
GETUI_IOS_SDK/
|- readme.txt (SDK资料包说明)
|- API文档/ (iOS SDK相关集成文档PDF版本)
|- 资源文件/
|   |- GtSdkLib/ (标准版个推SDK.framework文件)
|   |   |- GTEExtensionSDK.framework
|   |   |- GTSDK.framework
|   |- GtSdkLib-noidfa/ (无IDFA版个推SDK.framework文件)
|- Demo工程/
|   |- GtSdkDemo/ (XCode标准集成演示Demo工程)
|   |- GtSdkDemo-objc/ (Object-C标准集成框架代码工程, 示范CocoaPods集成)
|   |- GtSdkDemo-swift/ (Swift标准集成框架代码工程)
|   |- GtSdkLib/ (个推SDK库文件, 供上述Demo工程引用)
```

- 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: 标准版个推SDK, 包含集成所需的静态库和头文件。

(1)、如果在 App 内投放广告, 获取 IDFA 可通过苹果审核。

## (2)、如果 App 内无广告，可参考如下勾选，通过苹果审核。

### 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。

对于广告标识符的 (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。

## 使用标准版本SDK需添加 AdSupport.framework 库支持

- GtSdkLib-noidfa：无IDFA版个推SDK，包含集成所需的静态库和头文件。

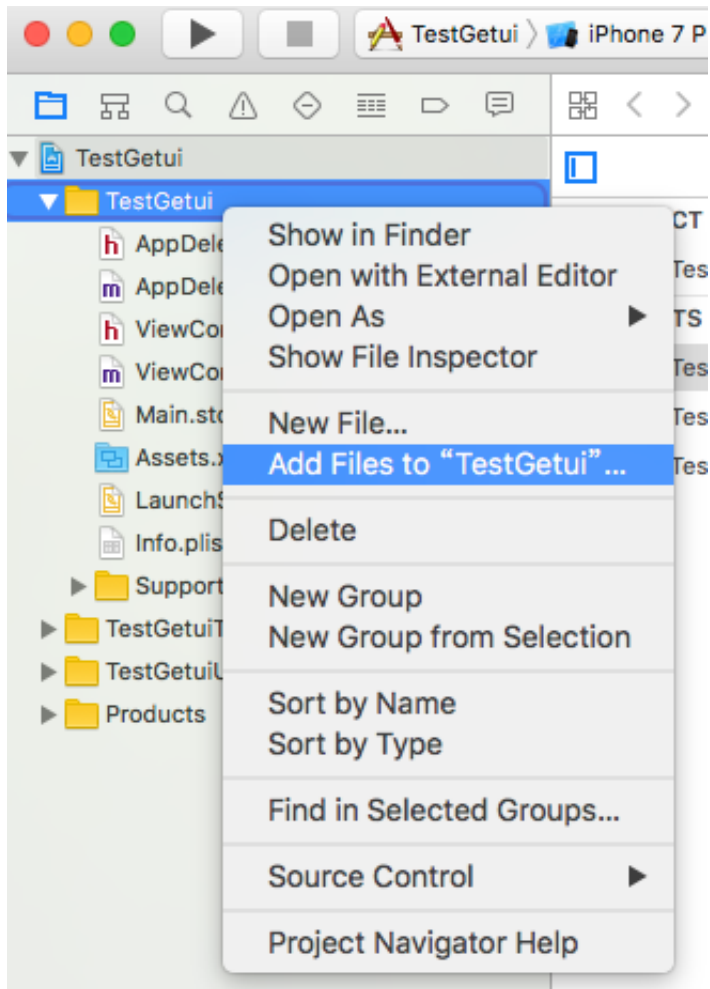
在 App 内无广告情况下还是建议开发者使用获取 IDFA 版本，并参考(2)中所说的方式提交 AppStore 审核。当然，如果开发者不想使用 IDFA 或者担忧采集 IDFA 而未集成任何广告服务遭到 Apple 拒绝，我们也准备了该无 IDFA 版本供开发者集成。

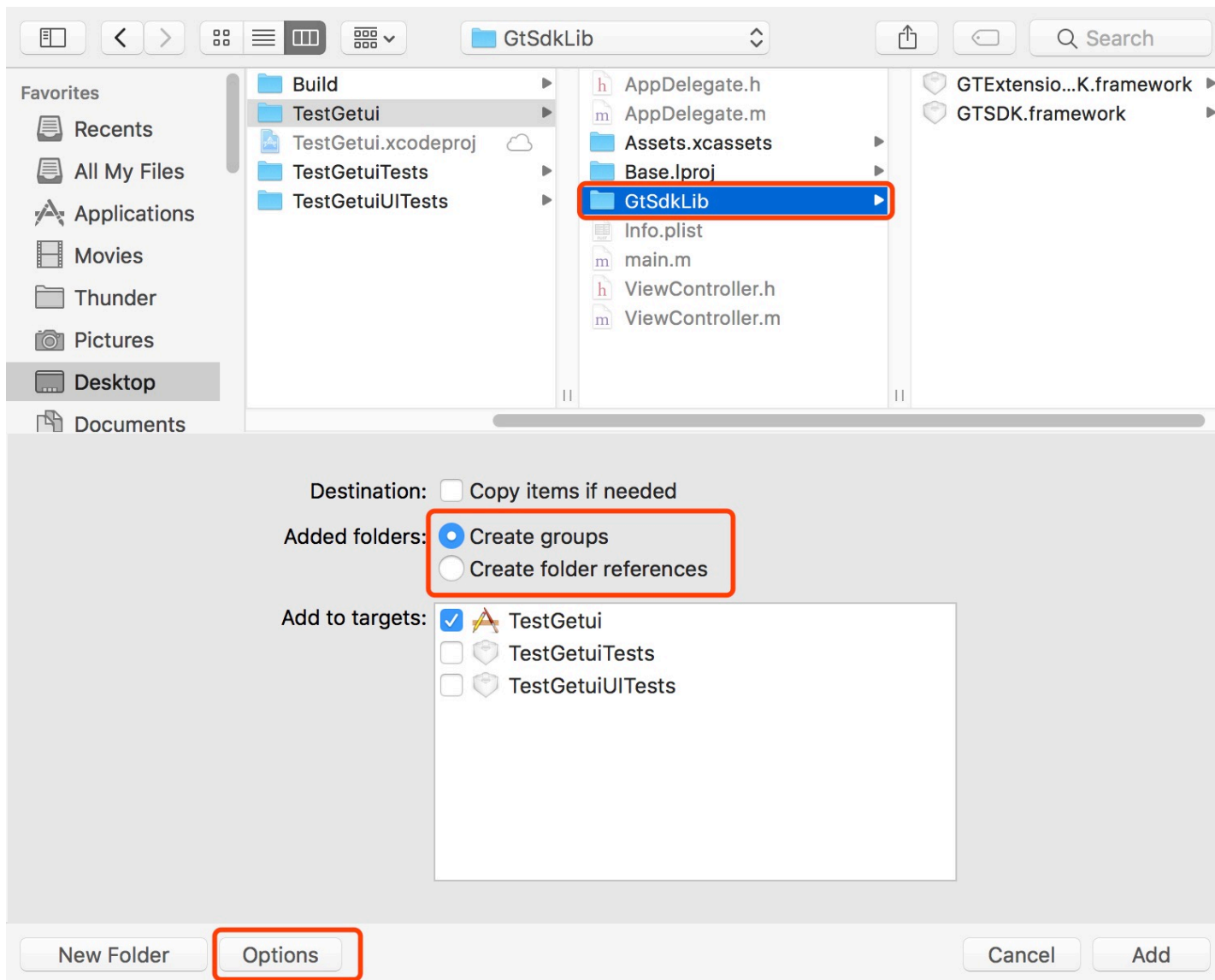
## 使用无 IDFA 版本SDK需删除 AdSupport.framework 库支持

- libGeTuiSdk-{version}：个推SDK核心库文件，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` 设备，支持 iOS 版本为7.0及以上。
- libGtExtensionSdk：为Notification Service Extension扩展库，支持 iOS 10 以上进行 APNs 通知展示数统计。

## 3.2 导入个推SDK

- 将 `GtSdkLib` 目录拷贝到项目工程目录下，导入 `GtSdkLib` 文件夹：





### 3.3 库引用检查

- 在导入时，Xcode正常情况下会自动添加引用，但是偶尔也会不添加，注意检查下图中的引用路径，如果报错找不到库或者头文件，一般都是下面的引用没有添加。



### 3.4 添加系统依赖库













老版本升级到 1.6.2.0 及以上版本注意事项：

1. 不再需要 JavaScriptCore.framework 使用。
2. 不再需要 CoreBluetooth.framework 使用。
3. 添加 UserNotifications.framework 库并使用 Optional 方式引用，需要使用 XCode 8 及以上版本编译。

- 在项目设置中添加以下系统库支持：

```
libc++.tbd
libz.tbd
libsqlite3.tbd
Security.framework
MobileCoreServices.framework
SystemConfiguration.framework
CoreTelephony.framework
AVFoundation.framework
CoreLocation.framework
UserNotifications.framework (iOS 10 及以上需添加，使用 Optional 方式接入)
AdSupport.framework (如果使用无IDFA版本SDK，则需删除该 AdSupport 库)
```

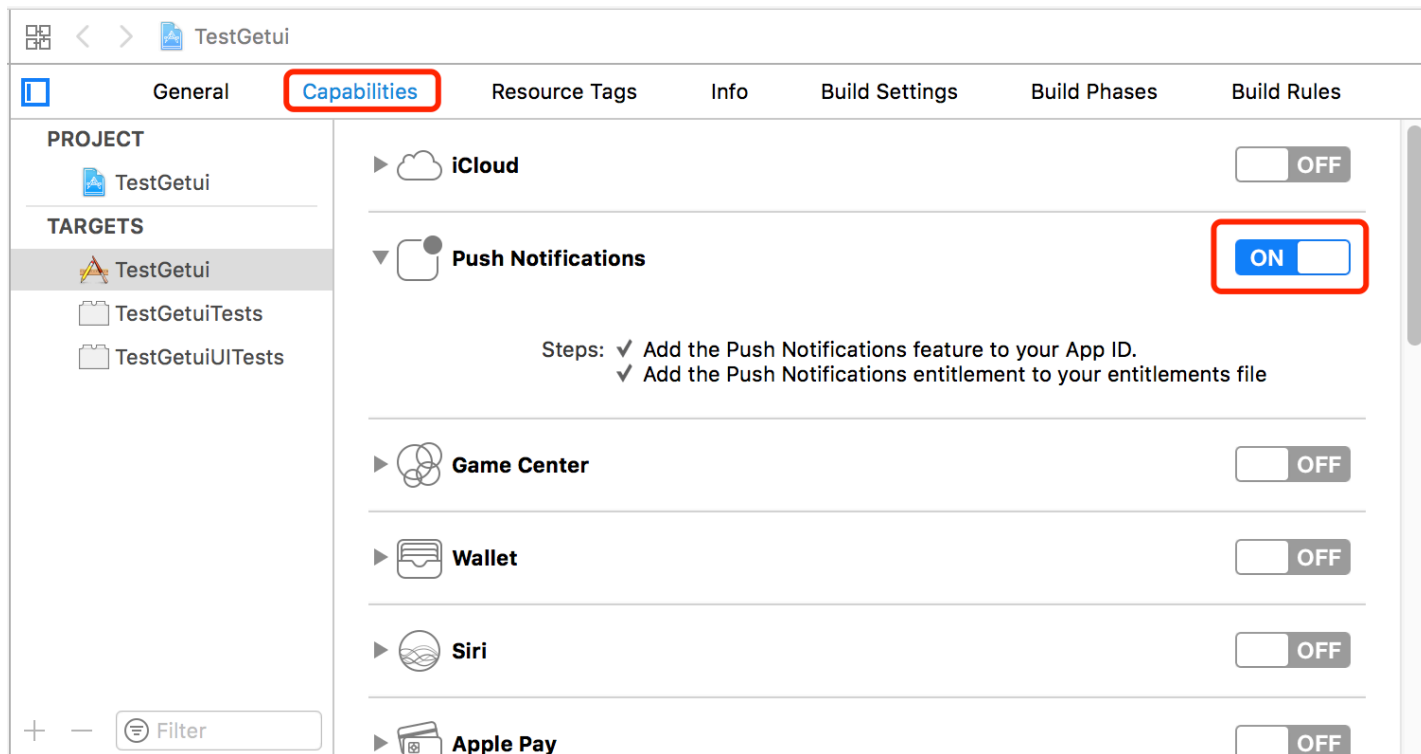
- 删除 JavaScriptCore.framework 引用

Name	Status
 GTSDK.framework	Required ⇅
 libz.tbd	Required ⇅
 libc++.tbd	Required ⇅
 libsqlite3.tbd	Required ⇅
 Security.framework	Required ⇅
 AdSupport.framework	Required ⇅
 UserNotifications.framework	Optional ⇅
 CoreLocation.framework	Required ⇅
 AVFoundation.framework	Required ⇅
 CoreTelephony.framework	Required ⇅
 SystemConfiguration.framework	Required ⇅
 MobileCoreServices.framework	Required ⇅
+ - Drag to reorder frameworks	

### 3.5 开启推送功能

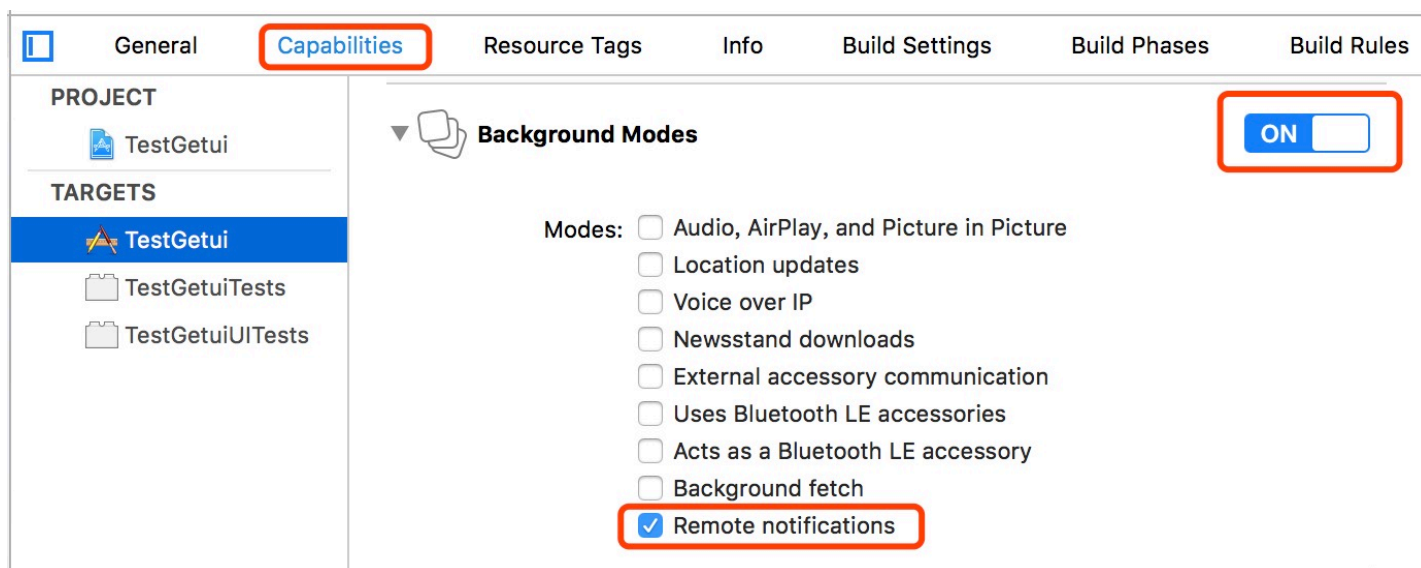
- 在 Xcode 8.x 以上，必须开启Push Notification能力。找到应用 Target 设置中的 Capabilities - > Push Notifications，确认开关已经设为 ON 状态。如果没有开启该开关，在 Xcode 8.x 上编译后的应用将获取不到 DeviceToken：





### 3.6 后台运行权限设置

- 为了更好支持消息推送，提供更多的推送样式，提高消息到达率，需要配置后台运行权限：



- Remote notifications : APNs静默推送权限

## 4. 编写集成代码

### 4.1 在 AppDelegate 中实现个推回调接口

- 为 AppDelegate 增加回调接口类。在iOS 10以前的设备，回调事件通过 GeTuiSdkDelegate 来

进行，在 iOS 10以后，可以使用UserNotifications框架来实现。示例代码如下：

```
// AppDelegate.h

#import <UIKit/UIKit.h>
#import <GTSDK/GeTuiSdk.h>      // GetuiSdk头文件应用

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

/// 使用个推回调时，需要添加"GeTuiSdkDelegate"
/// iOS 10 及以上环境，需要添加 UNUserNotificationCenterDelegate 协议，才能使用 UserNoti
fications.framework 的回调
@interface AppDelegate : UIResponder <UIApplicationDelegate, GeTuiSdkDelegate, UNU
serNotificationCenterDelegate>
```

## 4.2 初始化SDK并注册APNs

- 在 `[AppDelegate didFinishLaunchingWithOptions]` 方法中调用个推sdk初始化方法，传入个推平台分配的 `AppID`、`AppKey`、`AppSecret`。同时，调用APNs注册方法，尝试获取APNs DeviceToken。示例代码如下：

```
/// 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"

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NS
Dictionary *)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 d
elegate:self];
    // 注册 APNs
    [self registerRemoteNotification];
    return YES;
}
```

- 注册APNs获取DeviceToken的流程，根据项目设置的不同以及手机系统版本的不同，注册代码会有所区别，可以参考如下方式进行适配：

```
/** 注册 APNs */
- (void)registerRemoteNotification {
    /*
        警告：Xcode8 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];
    }
}
```

## 4.3 向个推服务器注册DeviceToken

- 为了免除开发者维护 `DeviceToken` 的麻烦，个推SDK可以帮开发者管理好这些繁琐的事务。应用开发者只需调用个推SDK的接口汇报最新的 `DeviceToken`，即可通过个推平台推送 APNs 消息。示例代码如下：

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

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

## 4.4 统计APNs通知的点击数

- 在iOS 10 以前，为处理 APNs 通知点击事件，统计有效用户点击数，需在 `AppDelegate.m` 里的 `didReceiveRemoteNotification` 回调方法中调用个推SDK统计接口：

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

- 对于iOS 10 及以后版本，为处理 APNs 通知点击，统计有效用户点击数，需先添加 `UNUserNotificationCenterDelegate`，然后在 `AppDelegate.m` 的 `didReceiveNotificationResponse` 回调方法中调用个推SDK统计接口：

```

#if __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

```

## 4.5 获取CID信息

- 个推SDK初始化完成后, 可以在 `[GeTuiSdkDelegate GeTuiSdkDidRegisterClient]` 回调方法中获取注册成功的ClientID (即CID) :

```

/** 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]:%\n\n", clientId);
}

```

## 5. 个推高级功能

### 5.1 接收个推通道下发的透传消息

- 当 SDK 在线时（即 App 在前台运行时）进行消息推送，该消息将直接通过个推通道发送给 App，通常这种方式比通过 APNs 发送来得更及时更稳定；当 SDK 离线时（即停止 SDK 或 App 后台运行 或 App 停止状态时）进行消息推送，个推平台会给苹果 APNs 推送消息，同时保存个推通道的离线消息，当 SDK 重新上线后，个推平台会重新推送所有离线的消息。
- APP 可以通过 `[GeTuiSdkDelegate GeTuiSdkDidReceivePayloadData]` 回调方法获取透传消息，其中 `payloadData` 参数为透传消息数据，`offLine` 参数则表明该条消息是否为离线消息。示例代码如下：

```
/** 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=%@,payloadMsg:%@%@",taskId,msgId, payloadMsg,offLine ? @"<离线消息>" : @""];
    NSLog(@"\n>>>[GexinSdk ReceivePayload]:%@\\n\\n", msg);
}
```

上述接口获取的透传消息为下图中 消息内容 输入框中的内容：

应用列表 > 创建推送 · 透传消息

创建推送

推送通知

透传消息

分组对比测试

数据统计

配置管理

指数报告

目标平台

Android

iOS生产环境

透传内容

\*描述：只用于后续查找识别,类似于备注 0/30

\*消息内容：请输入透传消息的命令代码 0/800

[Json参数辅助生成工具](#)

高级设置

## 5.2 苹果 APNs 静默推送

- 如果需要使用 静默推送 (Remote Notifications) 功能, 请在推送时指定 `content-available:1` 参数。
- APP 可以通过 `[AppDelegate didReceiveRemoteNotification]` 回调方法获取APNs 远程通知数据。
- 静默推送收到消息后也需要将APNs信息传给个推统计, 添加 `[GeTuiSdk handleRemoteNotification:userInfo];` 消息回执。
- 示例代码如下:

```
/** 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);
    ;

    //静默推送收到消息后也需要将APNs信息传给个推统计
    [GeTuiSdk handleRemoteNotification:userInfo];

    completionHandler(UIBackgroundFetchResultNewData);
}
```

请参照【3.5】节的内容正确添加 Remote notifications 权限

## 5.3 指定标签推送

- 应用可以为设备设置一组自定义标签, 后续可以针对某一特定标签用户组进行推送:

```
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];
}
```

## 5.4 设置别名

- 应用可以为设备设置不同的别名, 后续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" andIsSelf:YES];
```

- 解绑说明：解绑别名中，isSelf 字段为标识是否只对当前 cid 生效，如果是 true，只对当前 cid 做解绑；如果是 false，对所有绑定该别名的 cid 列表做解绑。
- 可以在 `GeTuiSdkDidAliasAction` 方法中处理 绑定/解绑 响应：

```
- (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);
        }
    }
}
```

## 5.5 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。个推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` 接口同步给个推服务器即可，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
```

## 5.6 重置角标

- 重置角标，重置服务器角标计数，使服务端计数变更为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
```

## 5.7 设置渠道标识

- 设置渠道信息，便于后续根据渠道各项统计信息，详情请咨询技术支持人员。

支持版本: v1.5.0及后续版本

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

## 6.多媒体及展示统计

- Apple 在 iOS 10 中新增了 `Notification Service Extension` 机制，可在消息送达时进行业务处理。为精确统计消息送达率，在集成个推SDK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：APNs 多媒体推送功能已经全平台支持，可以通过[个推开发者中心](#)进行下发或者 API 进行推送。

### 应用平台

☒ Android

应用标识： [修改](#)

☒ iOS

应用证书： [修改](#)

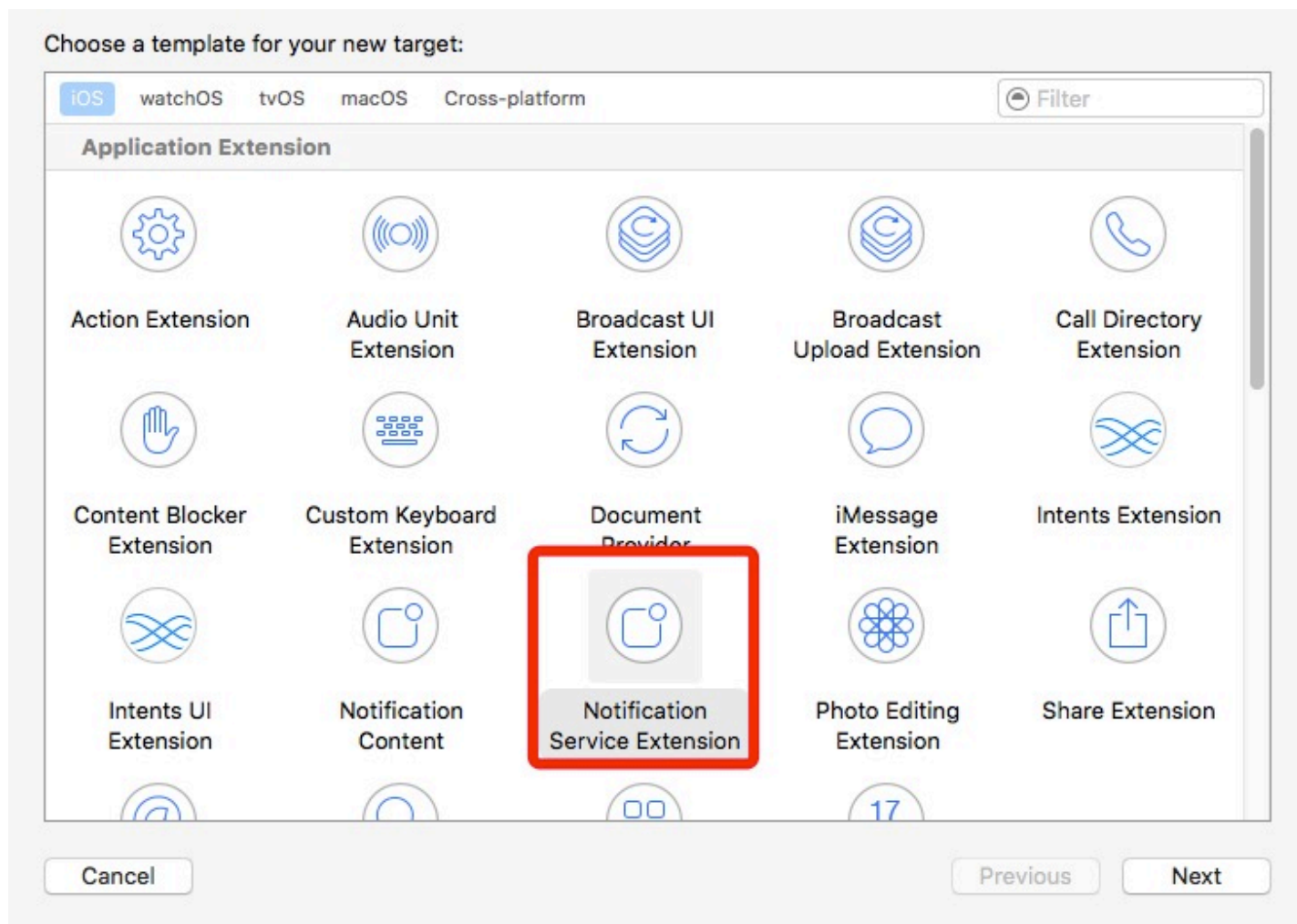
证书密码： [修改](#) [测试一下？](#)

证书环境：**生产环境**

证书日期：

### 6.1 在项目中添加 Notification Service Extension

- 打开 Xcode 8，菜单中选择 `File -> New -> Target -> Notification Service Extension`：



填写Target信息时需要注意以下两点：

- 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 BundeID 重复的错误。
- 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。

因此我们建议使用 `<Main Target Bundle ID>.NotificationService` 格式作为Extension的 BundleID。

- 添加 Notification Service Extension 后会生成相应的 Target。点 **Finish** 按钮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：



### Activate “NotificationServiceExtension” scheme?

This scheme has been created for the “NotificationServiceExtension” target. Choose Activate to use this scheme for building and debugging. Schemes can be chosen in the toolbar or Product menu.

☐ Do not show this message again

Cancel

Activate

- 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` 方法结束这次处理。但是如果处理时间过长，将会进入 `serviceExtensionTimeWillExpire` 方法进行最后的紧急处理。

```
- (void)serviceExtensionTimeWillExpire {
    // Called just before the extension will be terminated by the system.
    // Use this as an opportunity to deliver your "best attempt" at modified conte
nt, otherwise the original push payload will be used.
    self.contentHandler(self.bestAttemptContent);
}
```

- 如果 `didReceiveNotificationRequest` 方法在限定时间内没有调用 `contentHandler` 方法结束处理，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。





## 6.2 添加 GtExtensionSdk 依赖库

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

```
libz.tbd
libsqlite3.tbd
GtExtensionSDK.framework
UserNotifications.framework
```

## ▼ Link Binary With Libraries (4 items)



Name	Status
 GTExtensionSDK.framework	Required ⇅
 libz.tbd	Required ⇅
 libsqlite3.tbd	Required ⇅
 UserNotifications.framework	Required ⇅
+ — Drag to reorder frameworks	

需设置 NotificationService 最低运行版本为 10.0:

### ▼ Deployment Info

Deployment Target	10.0
Devices	Universal
Main Interface	

## 6.3 使用 GtExtensionSdk 进行 APNs 展示统计

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

```
#import <GtExtensionSDK/GeTuiExtSdk.h>
```

- NotificationService.m 中有两个回调方法 didReceiveNotificationRequest 和 serviceExtensionTimeWillExpire。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。示例代码如下:

```

- (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:^(

        //注意：是否修改下发后的title内容以项目实际需求而定
        //self.bestAttemptContent.title = [NSString stringWithFormat:@"%s@ [需求而定]",
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);
    }]];
}

```

- 我们对这个统计处理过程限定必须在5s内完成，因此开发者无需担心因为这个统计处理影响到消息的正常展示。

## 6.4 使用 GtExtensionSdk 进行多媒体推送及展示统计（建议使用该方法）

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

```
#import <GtExtensionSDK/GeTuiExtSdk.h>
```

- 2.查看：点击 `GeTuiExtSdk` 头文件，查看有两个方法，如下：

```

/**
 * 统计APNs到达情况
 */
+ (void)handelNotificationServiceRequest:(UNNotificationRequest *) request withCom
plete:(void (^)(void))completeBlock;

/**
 * 统计APNs到达情况和多媒体推送支持,建议使用该方法接入
 */
+ (void)handelNotificationServiceRequest:(UNNotificationRequest *)request withAtta
chmentsComplete:(void (^)(NSArray* attachments, NSArray* errors))completeBlock;

/**
 * sdk销毁，资源释放
 */
+ (void)destory;

```

接口使用说明：

1).如果需要多媒体推送，请使用第二种方法进行调用。接收到 APNs 通知后，SDK 判断是否有多媒体资源，如果多媒体资源存在则SDK下载资源，下载完成后以 Block 方式回调返回 attachments 资源数组对象和 errors 错误数组信息。

2).多媒体大小限制：

资源	类型	大小	超时
图片	1	<=10M	120s
音频	2	<=5M	60s
视频	3	<=50M	180s

3).可设置是否仅在 WIFI 下下载资源，参考服务端API。如果为True，数据网络环境下将不下载资源，展示通知不带附件。默认是:False,支持 WIFI 和数据网络下下载。

4). 资源URL格式注意：不支持中文及转译地址，请使用英文合法地址。

- 3.接入： `NotificationService.m` 中有两个回调方法 `didReceiveNotificationRequest` 和 `serviceExtensionTimeWillExpire` 。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。示例代码如下：

```

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

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

    NSLog(@"----将APNs信息交由个推处理----");

    [GeTuiExtSdk handelNotificationServiceRequest:request withAttachmentsComplete:
^(NSArray *attachments, NSArray* errors) {

        //注意：是否修改下发后的title内容以项目实际需求而定
        //self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [需求而定]
", self.bestAttemptContent.title];

        self.bestAttemptContent.attachments = attachments; //设置通知中的多媒体附件

        NSLog(@"个推处理APNs消息遇到错误: %@",errors); //如果APNs处理有错误，可以在这里查看
相关错误详情

        self.contentHandler(self.bestAttemptContent); //展示推送的回调处理需要放到个推回
执完成的回调中
    }];
}

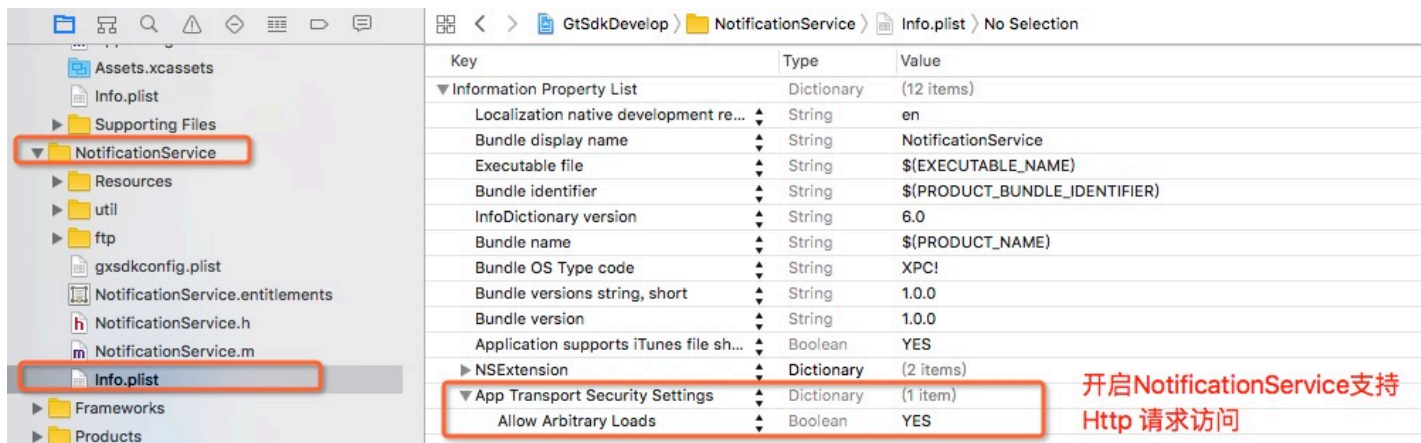
```

- 4.错误分析：多附件下载时每个附件遇到错误都将以 NSError\* 对象返回错误信息。当多媒体无法展示时，请注意检查 NSError 中错误信息。

ErrorCode	描述	解决
10001	多媒体地址错误, 无法下载	请使用正常地址下发
10002	Http URL 不支持	支持 ATS 使用 Https 地址或者将 NotificationService 配置为支持 Http 请求，如下图所示
10003	多媒体资源超过大小限制	请检查多媒体资源大小是否符合，更正后重新下发

开启多媒体地址 Http 访问支持：





## 6.4 超时资源释放

- 在 `NotificationService` 处理超时时，需调用 `GtExtensionSdk` 中 `destory` 方法，进行资源释放，防止因为资源未正常释放导致 crash。

```
//NotificationService处理超时系统调用
- (void)serviceExtensionTimeWillExpire {

    //销毁SDK，释放资源
    [GeTuiExtSdk destory];
    //结束 NotificationService 调用
    self.contentHandler(self.bestAttemptContent);
}
```

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

- 使用了 Notification Service Extension 的项目在制作待上传至 AppStore 的 IPA 包时，编译设备需要选择 `Generic iOS Device`，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。如下图所示：



## 7. VOIP 推送

- Apple 在 iOS 8 后引入了一个基于 PushKit 框架的 VoIP 推送，这可以使用户收到一个 VoIP 推送时唤醒 APP。有了这种 VoIP 的推送，开发者们不需要让 APP 持续保持后台运行。当 APP 在后台被系统挂起后如果需要 APP 后台运行，可以发送这个 VoIP 消息将会在后台唤醒 APP，并执行制定的代码 (在弹出通知消息给用户之前)。



- VoIP 服务需系统 iOS8 及以上系统支持。
- VoIP 证书不区分开发和生产环境，VoIP 证书只有一个，生产和开发都可用同一个证书。

## 7.1 添加 PushKit.Framework 库

1、导入 `PushKit.Framework` 库以 `Optional` 方式引入。

2、导入头文件 `#import <PushKit/PushKit.h>` 。

3、AppDelegate 实现 `PKPushRegistryDelegate` 协议，并实现 Delegate 中的接口。

```
<!--实现 PKPushRegistryDelegate 协议-->
@interface AppDelegate : UIResponder <UIApplicationDelegate, PKPushRegistryDelegat
e, GetTuiSdkDelegate> {
}

- (void)pushRegistry:(PKPushRegistry *)registry didUpdatePushCredentials:(PKPushCr
edentials *)credentials forType:(NSString *)type {
<!--具体实现参考 7.3 -->
}

- (void)pushRegistry:(PKPushRegistry *)registry didReceiveIncomingPushWithPayload:
(PKPushPayload *)payload forType:(NSString *)type {
<!--具体实现参考 7.4 -->
}
```

## 7.2 添加 VOIP 权限

在 info.plist 中添加 voip 权限，如图：

```
<key>UIBackgroundModes</key>
<array>
    <string>audio</string>
    <string>fetch</string>
    <string>remote-notification</string>
    <string>voip</string>
</array>
```

## 7.3 注册 VOIP Token

1、注册 VoIP

- 实现 VoIP 注册

```

//注册 VoIP
- (void)voipRegistration {
    dispatch_queue_t mainQueue = dispatch_get_main_queue();
    PKPushRegistry *voipRegistry = [[PKPushRegistry alloc] initWithQueue:mainQueue
];
    voipRegistry.delegate = self;
    voipRegistry.desiredPushTypes = [NSSet setWithObject:PKPushTypeVoIP];
}

```

- AppDelegate 的 `didFinishLaunchingWithOptions` 中调用 VoIP 注册

```

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

    [GeTuiSdk runBackgroundEnable:YES]; // [是否允许APP后台运行]
    [GeTuiSdk lbsLocationEnable:YES andUserVerify:YES]; // [是否运行电子围栏Lbs功能和是否SDK主动请求用户定位]
    [GeTuiSdk setChannelId:@"GT-Channel"]; // 渠道
    // [1]:使用APPID/APPKEY/APPSECRET创建个推实例
    [GeTuiSdk startSdkWithAppId:[AppConfig GtAppId] appKey:[AppConfig GtAppKey] appSecret:[AppConfig GtAppSecret] delegate:self];

    // [2]:注册APNS
    [self registerRemoteNotification];

    // [3]:注册 VoIP

    [self voipRegistration];

    return YES;
}

```

## 2、绑定 VoIP Token

```

- (void)pushRegistry:(PKPushRegistry *)registry didUpdatePushCredentials:(PKPushCredentials *)credentials forType:(NSString *)type {
    NSString *voiptoken = [credentials.token.description stringByTrimmingCharactersInSet:[NSCharacterSet characterSetWithCharactersInString:@"<>"]];
    voiptoken = [voiptoken stringByReplacingOccurrencesOfString:@" " withString:@""];

    // 向个推服务器注册 VoipToken
    [GeTuiSdk registerVoipToken:voiptoken];

    NSLog(@"[VoipToken is:]:%@", voiptoken);
}

```

7.4 接收 VoIP 推送处理具体业务逻辑，并调用个推 VoIP 回执统计，统计 VoIP 到达情况。

```
- (void)pushRegistry:(PKPushRegistry *)registry didReceiveIncomingPushWithPayload:
(PKPushPayload *)payload forType:(NSString *)type {
    // TODO:接收 VoIP 推送中的 Payload 信息进行业务处理。
    <!--TODO: 具体 VoIP 业务处理-->
    NSLog(@"[VoIP Payload]:%@",payload, payload.dictionaryPayload);

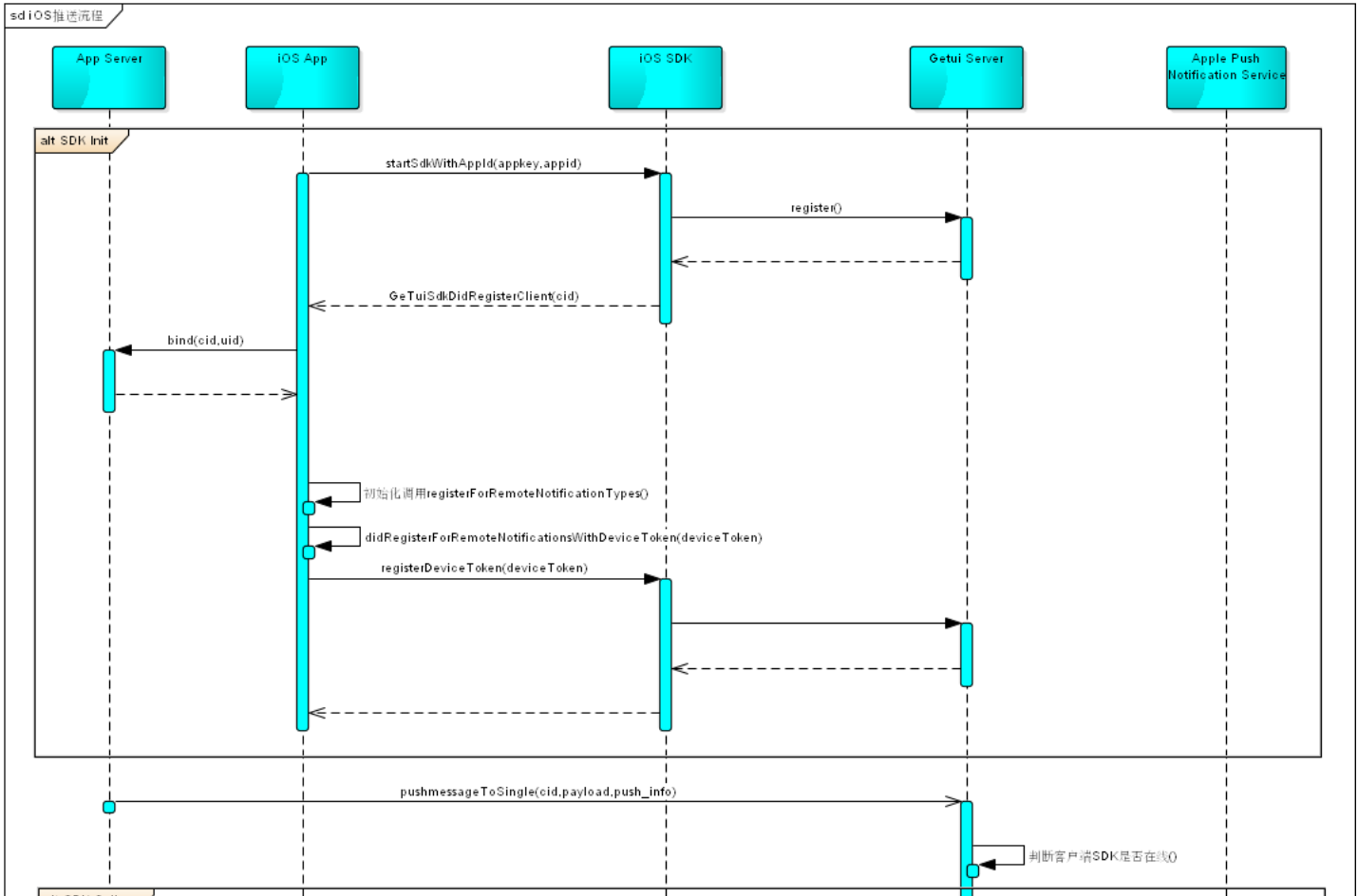
    // 调用个推 VoIP 回执统计接口
    [GeTuiSdk handleVoipNotification:payload.dictionaryPayload];
}
```

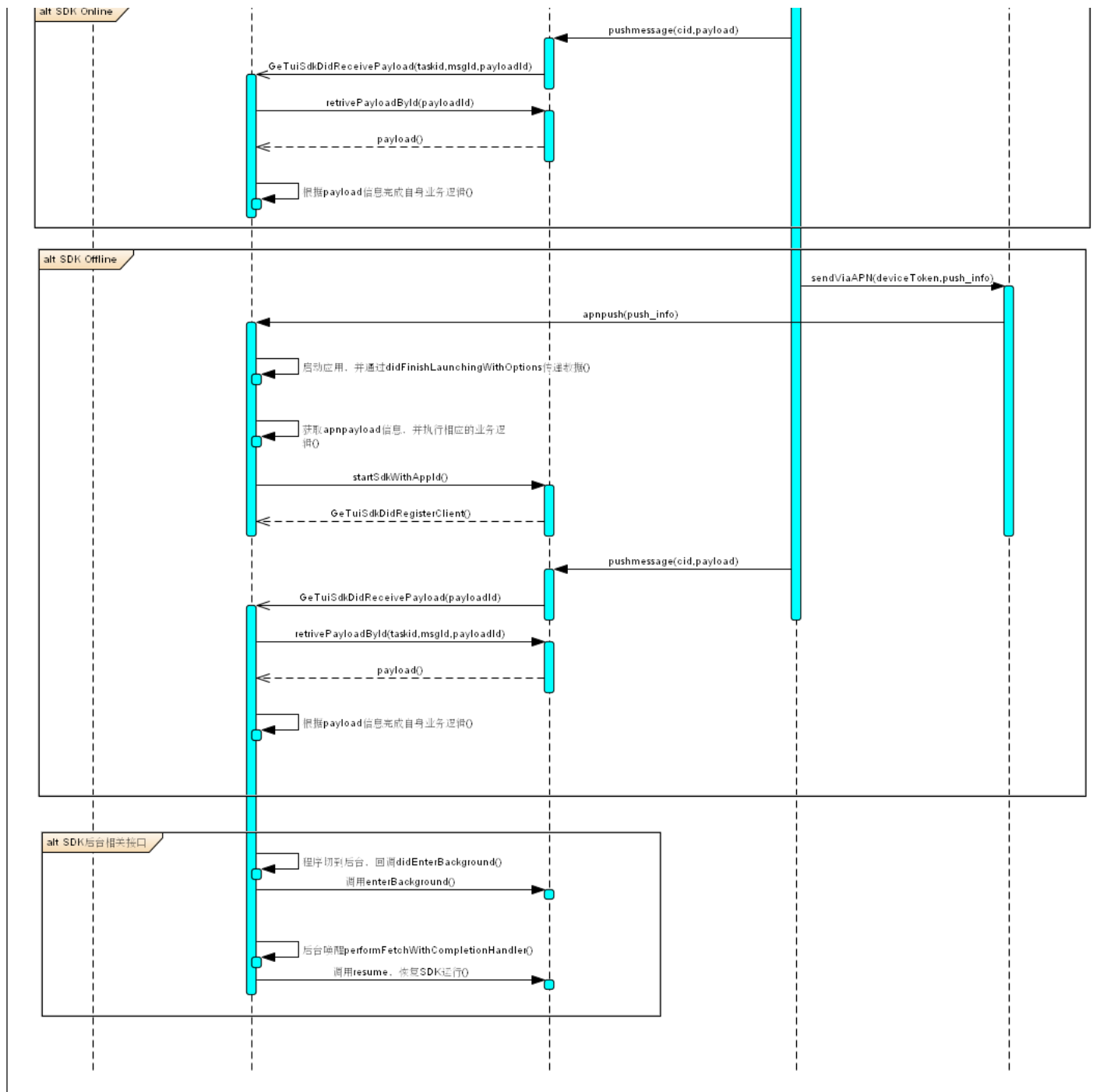
8. AppLink 支持

[AppLink接入说明](#)

9. 推送流程

- iOS应用、应用服务器、个推SDK、个推服务器、Apple Push Notification Server之间的交互流程图，如下图所示：





- 尤其需要注意在SDK在线和离线状态下分别不同的处理流程，APP 需要很好理解不同场景下的推送消息处理流程。# **XCode集成**

## ## 前言

\* 开发环境下手动方式导入SDK资源进行集成的步骤，配置相对复杂，需要仔细阅读文档和Demo工程。

\* 个推在国内首创APNs展示统计功能，可精确统计通过苹果APNs服务下发的通知数量，开发者可以获知更为精准的推送展示数和通知点击率，为运营提供了更强大的数据支持。

\* 个推多媒体推送功能支持，可通过后台推送多媒体资源，支持图片，音频，视频的展示。

- \* 本文档适用SDK版本: 2.3.0.0 及以后
- \* 请参考`GtSdkDemo`Demo工程

## ## 1. 创建个推应用

\* 请登录 [<http://dev.getui.com>](<http://dev.getui.com>) , 选择`登记应用`并填写应用名称和包名信息, 完成应用创建:

reateapp1.png) ! [img\_xcode\_create](img/xcode\_c

reateapp2.png) ! [img\_xcode\_create](img/xcode\_c

\* 点击`应用配置`, 获取到相应的`AppID`  
`AppKey`、`AppSecret`信息:

reateapp3.png) ! [img\_xcode\_create](img/xcode\_c

## ## 2. 打开项目工程

\* 启动xCode, 打开您之前创建的ios项目  
工程:

n.png) ! [img\_xcode\_open](img/xcode\_ope

## ## 3. 添加个推SDK及相关配置

### ### 3.1 个推iOS SDK资料包结构

```

```
GETUI_IOS_SDK/  
  |- readme.txt (SDK资料包说明)  
  |- API文档/ (iOS SDK相关集成文档  
  
  |- 资源文件/  
    |- GtSdkLib/ (标准版个推S  
  
    |- GTExtensionSDK.fr  
  
    |- GTSDK.framework  
    |- GtSdkLib-noidfa/ (无I  
  
  - Demo工程/  
    |- GtSdkDemo/ (xCode标准
```

PDF版本)

DK.framework文件)

amework

DFA版个推SDK.framework文件)

集成演示Demo工程)

ct-C标准集成框架代码工程，示范CocoaPods集成)  
ft标准集成框架代码工程)  
件，供上述Demo工程引用)

的展示个推 SDK 功能点。

，方便 objc 开发者集成个推 SDK。

emo，方便 Swift 开发者集成个推 SDK。

所需的静态库和头文件。

FA 可通过苹果审核。</br>

选，通过苹果审核。

amework 库支持

K，包含集成所需的静态库和头文件。

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，如果开发者不想使用 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。

t.framework 库支持

核心库文件，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`设备，支持 iOS 版本为7

```
|      |- GtSdkDemo-objc/ (Objc
|      |- GtSdkDemo-swift/ (Swi
|      |- GtSdkLib/ (个推SDK库文
~~~
```

```
* GtSdkDemo: SDK 演示 Demo，能更好
* GtSdkDemo-objc: objc 集成 Demo
* GtSdkDemo-swift: Swift 集成 D
* GtSdkLib: 标准版个推SDK，包含集成
```

```
<mark class="note">
```

```
(1)、如果在 App 内投放广告，获取 ID
```

```
(2)、如果 App 内无广告，可参考如下勾
```

```
</mark>
```

```

```

```
<mark class="notice">
```

```
使用标准版本SDK需添加 AdSupport.fr
```

```
</mark>
```

```
* GtSdkLib-noidfa: 无IDFA版个推SD
```

```
<mark class="notice">
```

```
使用无 IDFA 版本SDK需删除 AdSuppor
```

```
</mark>
```

```
* libGeTuiSdk-{version}: 个推SDK
```

```
支持`i386`、`x86_64`、`arm64`、`a
```

.0及以上。

\* libGtExtensionSdk: 为Notification Service Extension扩展库, 支持 iOS 10 以上进行 APNs 通知展示数统计。

### ### 3.2 导入个推SDK

\* 将`GtSdkLib`目录拷贝到项目工程目录下, 导入`GtSdkLib`文件夹:





### ### 3.3 库引用检查

\* 在导入时, Xcode正常情况下会自动添加引用, 但是偶尔也会不添加, 注意检查下图中的引用路径, 如果报错找不到库或者头文件, 一般都是下面的引用没有添加。



### ### 3.4 添加系统依赖库

<mark class="notice">

老版本升级到 1.6.2.0 及以上版本注意

事项: </br>

ork 使用.</br>

rk 使用.</br>

1. 不再需要 JavaScriptCore.framework

2. 不再需要 CoreBluetooth.framework

3. 添加 UserNotifications.framework

ork 库并使用 Optional 方式引用, 需要使用 Xcode 8 及以上版本编译.</br>

</mark>

\* 在项目设置中添加以下系统库支持:

~~~

libc++.tbd

libz.tbd

libsqlite3.tbd

Security.framework

MobileCoreServices.framework

SystemConfiguration.framework

CoreTelephony.framework

AVFoundation.framework

CoreLocation.framework

UserNotifications.framework (iOS

10 及以上需添加, 使用 Optional 方式接入)

AdSupport.framework (如果使用

无IDFA版本SDK, 则需删除该 AdSupport 库)

引用

Notification能力。找到应用`Target`设置中的`Capabilities` -> `Push Notifications`, 确认开关已经设为`ON`状态。如果没有开启该开关, 在 Xcode 8.x 上编译后的应用将获取不到`Device Token`:

送样式, 提高消息到达率, 需要配置后台运行权限:

默推送权限

回调接口

iOS 10以前的设备, 回调事件通过`GeTuiSdkDelegate`来进行, 在 iOS 10以后, 可以使用UserNotifications框架来实现。示例代码如下:

```
```objective-c
// AppDelegate.h

#import <UIKit/UIKit.h>
#import <GTSDK/GeTuiSdk.h>

// GetuiSdk头文件应用

ations.framework

OWED >= __IPHONE_10_0

Notifications.h>
```

```
```
* 删除 JavaScriptCore.framework
```

```

```

### 3.5 开启推送功能

\* 在 Xcode 8.x 以上, 必须开启Push Notification能力。找到应用`Target`设置中的`Capabilities` -> `Push Notifications`, 确认开关已经设为`ON`状态。如果没有开启该开关, 在 Xcode 8.x 上编译后的应用将获取不到`Device Token`:

```

```

### 3.6 后台运行权限设置

\* 为了更好支持消息推送, 提供更多的推

```

```

\* `Remote notifications`: APNs静

## 4. 编写集成代码

### 4.1 在 AppDelegate 中实现个推

\* 为`AppDelegate`增加回调接口类。在 iOS 10以前的设备, 回调事件通过`GeTuiSdkDelegate`来进行, 在 iOS 10以后, 可以使用UserNotifications框架来实现。示例代码如下:

```
```objective-c
// AppDelegate.h

#import <UIKit/UIKit.h>
#import <GTSDK/GeTuiSdk.h>

// iOS10 及以上需导入 UserNotific
ations.framework

#ifdef __IPHONE_10_0
#import <UserNotifications/User
Notifications.h>
#endif
```
```



```

#endif

/// 使用个推回调时，需要添加"GeTuiS
dkDelegate"

/// iOS 10 及以上环境，需要添加 UNU
serNotificationCenterDelegate 协议，才能使用 UserNotifications.framework 的回调
@interface AppDelegate : UIResp
onder <UIApplicationDelegate, GeTuiSdkDelegate, UNUserNotificationCenterDelega
te>
...

### 4.2 初始化SDK并注册APNs

* 在`[AppDelegate didFinishLaunc
hingWithOptions]`方法中调用个推sdk初始化方法，传入个推平台分配的 `AppID`、`AppKey`、`A
ppSecret`。同时，调用APNs注册方法，尝试获取APNs DeviceToken。示例代码如下：

```objective-c
/// 个推开发者网站中申请App时，注册
的AppId、AppKey、AppSecret
ahVVxurw6BNr7XSn9EF2"
Pfqwq6OMAPp6dkqgLpG5"
aBqAD6t79JfzTB6Z5lo5"

#define kGtAppId @"iM
#define kGtAppKey @"yI
#define kGtAppSecret @"GO

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 通过个推平台分配的appId、 ap
pKey 、 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 registerRemoteNotific
ation];

    return YES;
}
...

* 注册APNs获取DeviceToken的流程，
根据项目设置的不同以及手机系统版本的不同，注册代码会有所区别，可以参考如下方式进行适配：

```objective-c
/** 注册 APNs */
- (void)registerRemoteNotificat
ion {

```

GETS -> Capabilities -> Push Notifications"

下代码根据 APP 支持的 iOS 系统不同，代码可以对应修改。

修改，注意测试支持的 iOS 系统都能获取到 DeviceToken

```
e].systemVersion floatValue] >= 10.0) {
```

```
OWED >= __IPHONE_10_0 // Xcode 8编译会调用
```

```
r *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
```

```
/*
警告：Xcode8 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
为演示代码，注意根据实际需要
```

```
*/
```

```
if ([[UIDevice currentDevice].systemVersion floatValue] >= 10.0) {
```

```
#if __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 *settings = [UIUserNotificationSettings settingsForTypes:types categories:
    nil];

    [[UIApplication sharedApplication] registerForRemoteNotifications];

    [[UIApplication sharedApplication] registerUserNotificationSettings:settings];
    } else {
        UIRemoteNotificationType
        e apn_type = (UIRemoteNotificationType)(UIRemoteNotificationTypeAlert |

        UIRemoteNotificationTypeSound |

        UIRemoteNotificationTypeBadge);
        [[UIApplication sharedApplication] registerForRemoteNotificationTypes:apn_type];
    }
}
```

```

### ### 4.3 向个推服务器注册DeviceToken

n

\* 为了免除开发者维护`DeviceToken`的麻烦，个推SDK可以帮开发者管理好这些繁琐的事务。应用开发者只需调用个推SDK的接口汇报最新的`DeviceToken`，即可通过个推平台推送 APNs 消息。示例代码如下：

```

```objective-c
/** 远程通知注册成功委托 */
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
    NSString *token = [[deviceToken description] stringByTrimmingCharactersInSet:[NSCharacterSet characterSetWithCharactersInString:@"<>"]];

    token = [token stringByReplacingOccurrencesOfString:@" " withString:@""];

    NSLog(@"\n>>>[DeviceToken Success]:%@", token);

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

```

### ### 4.4 统计APNs通知的点击数

\* 在iOS 10 以前，为处理 APNs 通知点

击事件，统计有效用户点击数，需在`AppDelegate.m`里的`didReceiveRemoteNotification`回调方法中调用个推SDK统计接口：

```
````objectivec
- (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);
}
````
```

\* 对于iOS 10 及以后版本，为处理 APNs 通知点击，统计有效用户点击数，需先添加 `UNUserNotificationCenterDelegate`，然后在`AppDelegate.m`的 `didReceiveNotificationResponse`回调方法中调用个推SDK统计接口：

```
````objectivec
#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 (^)(void))completionHandler {

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

传给个推统计

```

// [ GTSdk ]: 将收到的APNs信息
[GeTuiSdk handleRemoteNotification:response.notification.request.content.userInfo];

completionHandler();
}

#endif
~~~

```

### ### 4.5 获取CID信息

\* 个推SDK初始化完成后，可以在`[GeTuiSdkDelegate GeTuiSdkDidRegisterClient]`回调方法中获取注册成功的ClientID（即CID）：

```

~~~objectivec
/** SDK启动成功返回cid */
- (void)GeTuiSdkDidRegisterClient:(NSString *)clientId {

    //个推SDK已注册，返回clientId
    NSLog(@">>>[GeTuiSdk RegisterClient]:%@\\n\\n", clientId);

}
~~~

```

## ## 5. 个推高级功能

### ### 5.1 接收个推通道下发的透传消息

\* 当 SDK 在线时（即 App 在前台运行时）进行消息推送，该消息将直接通过个推通道发送给 App，通常这种方式比通过APNs发送来得更及时更稳定；当 SDK 离线时（即停止 SDK 或 App 后台运行 或 App 停止状态时）进行消息推送，个推平台会给苹果 APNs 推送消息，同时保存个推通道的离线消息，当 SDK 重新上线后，个推平台会重新推送所有离线的消息。

\* APP 可以通过`[GeTuiSdkDelegate GeTuiSdkDidReceivePayloadData]`回调方法获取透传消息，其中`payloadData`参数为透传消息数据，`offLine`参数则表明该条消息是否为离线消息。示例代码如下：

```

~~~objectivec
/** 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:%@,payloadMsg:%@%@",taskId,msgId, payloadDataMsg,offLine ? @"<离线消息>" : @""];

    NSLog(@"\n>>>[GexinSdk ReceivePayload]:%@\n\n", msg);

    }
    ...

```

上述接口获取的透传消息为下图中`消息内容`输入框中的内容:

```

<mark class="note">
</mark>

```

```

![@xcode_6](./img/xcode_6.png)

```

### ### 5.2 苹果 APNs 静默推送

\* 如果需要使用 静默推送 (Remote Notifications) 功能, 请在推送时指定`content-available:1`参数。  
 \* APP 可以通过`[AppDelegate didReceiveRemoteNotification]`回调方法获取APNs 远程通知数据。  
 \* \*\*静默推送收到消息后也需要将APNs信息传给个推统计, 添加`[GeTuiSdk handleRemoteNotification:userInfo];`消息回执。\*\*  
 \* 示例代码如下:

```

```objective-c
/** 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]:%@\n\n",userInfo);

    //静默推送收到消息后也需要将APNs信息传给个推统计

    [GeTuiSdk handleRemoteNotification:userInfo];

    completionHandler(UIBackgroundFetchResultNewData);
}

```

notifications`权限

后续可以针对某一特定标签用户组进行推送：

OS";

mponentsSeparatedByString:@","];

l) {

UIAlertView alloc] initWithTitle:@"Failed" message:@"设置失败" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];

可以针对具体别名进行推送：

equenceNum:@"seq-1"];

dSequenceNum:@"seq-2" andIsSelf:YES];

为标识是否只对当前 cid 生效，如果是 true，只对当前 cid 做解绑；如果是 false，对所有绑定该别名的 cid 列表做解绑。

方法中处理 绑定/解绑 响应：

```

<mark class="note">

请参照【3.5】节的内容正确添加`Remote

</mark>

### ### 5.3 指定标签推送

\* 应用可以为设备设置一组自定义标签，

```objective-c

NSString \*tagName = @"个推,推送,i

NSArray \*tagNames = [tagName co

if (![GeTuiSdk setTags:tagNames

UIAlertView \*alertView = [[

UIAlertView alloc] initWithTitle:@"Failed" message:@"设置失败" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];

[alertView show];

}

```

### ### 5.4 设置别名

\* 应用可以为设备设置不同的别名，后续

```objective-c

// 绑定别名

[GeTuiSdk bindAlias:@"个推" andS

// 取消绑定别名

[GeTuiSdk unbindAlias:@"个推" an

```

\* 解绑说明：解绑别名中，isSelf 字段为标识是否只对当前 cid 生效，如果是 true，只对当前 cid 做解绑；如果是 false，对所有绑定该别名的 cid 列表做解绑。

\* 可以在`GeTuiSdkDidAliasAction`

```objective-c

```

- (void)GetTuiSdkDidAliasAction:(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);
        }
    }
}

```

### ### 5.5 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。个推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`接口同步给个推服务器即可，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

```

        ````objective-c
        [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
    ````

```

### ### 5.6 重置角标

\* 重置角标，重置服务器角标计数，使服务端计数变更为0。

支持版本： v1.4.1及后续版本



```

    ````objective-c
    [GeTuiSdk resetBadge]; //重置角标
计数
    [[UIApplication sharedApplication] setApplicationIconBadgeNumber:0]; // APP 清空角标
    ````

```

### ### 5.7 设置渠道标识

\* 设置渠道信息，便于后续根据渠道各项统计信息，详情请咨询技术支持人员。

支持版本：v1.5.0及后续版本

```

    ````objective-c
    [GeTuiSdk setChannelId:@"GT-Cha
nnel"];
    ````

```

### ## 6.多媒体及展示统计

\* Apple 在 iOS 10 中新增了`Notification Service Extension`机制，可在消息送达时进行业务处理。为精确统计消息送达率，在集成个推SDK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。

**<mark class="notice"> 注意：APNs 多媒体推送功能已经全平台支持，可以通过[个推开发者平台](<https://dev.getui.com/>)进行下发或者 API 进行推送。</mark>**



### ### 6.1 在项目中添加 Notification Service Extension

\* 打开 Xcode 8，菜单中选择`File` -> `New` -> `Target` -> `Notification Service Extension`：



填写Target信息时需要注意以下两点：

\* 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。

\* 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。

`<mark class="notice">` 因此我们建议使用`<Main Target Bundle ID>.NotificationService`格式作为Extension的BundleID。`</mark>`

\* 添加 Notification Service Extension 后会生成相应的 Target。点`Finish`按钮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：

```
```\objectivecc
didReceiveNotificationRequest:(
UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationCon
tent *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`方法结束这次处理。但是如果处理时间过长，将会进入`serviceExtensionTimeWillExpire`方法进行最后的紧急处理。

```
```\objectivecc
- (void)serviceExtensionTimeWillExpire {

    // Called just before the e
xtension will be terminated by the system.

    // Use this as an opportuni
ty to deliver your "best attempt" at modified content, otherwise the original
push payload will be used.

    self.contentHandler(self.be
stAttemptContent);

}
```
```

\* 如果`didReceiveNotificationRequest`方法在限定时间内没有调用`contentHandler`方法结束处理，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。

### 6.2 添加 GtExtensionSdk 依赖库

\* 选择 Notification Service Exte

nsion 所对应的 Target, 添加如下依赖库支持:

```
~~~  
libz.tbd  
libsqlite3.tbd  
GTEExtensionSDK.framework  
UserNotifications.framework  
~~~
```

```

```

ficationService 最低运行版本为 10.0: </mark>

```

```

### 6.3 使用 GtExtensionSdk 进行

APNs 展示统计

\* 在 `NotificationService.m` 文

件中, 导入 GtExtensionSdk 的头文件:

```
~~~  
#import <GTEExtensionSDK/GeTuiExt  
Sdk.h>  
~~~
```

\* `NotificationService.m`中有两个回调方法 `didReceiveNotificationRequest` 和 `serviceExtensionTimeWillExpire`。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。示例代码如下:

```
~~~  
- (void)didReceiveNotificationR  
equest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotific  
ationContent * _Nonnull))contentHandler {  
  
    self.contentHandler = conte  
ntHandler;  
  
    self.bestAttemptContent = [  
request.content mutableCopy];  
  
    //使用 handelNotificationServi  
ceRequest 上报推送送达数据。  
  
    [GeTuiExtSdk handelNotifica  
tionServiceRequest:request withComplete:^(  
  
        //注意: 是否修改下发后的title
```

内容以项目实际需求而定

```
        //self.bestAttemptContent.  
title = [NSString stringWithFormat:@"%@" [需求而定]", self.bestAttemptContent.tit  
le];  
  
        // 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.b  
estAttemptContent);  
  
    }];  
}  
~~~
```

\* 我们对这个统计处理过程限定必须在5s内完成，因此开发者无需担心因为这个统计处理影响到消息的正常展示。

### 6.4 使用 GtExtensionSdk 进行多媒体推送及展示统计（建议使用该方法）

\* 1.引用：在 `NotificationService.m` 文件中，导入 GtExtensionSdk 的头文件：

```
~~~  
#import <GtExtensionSDK/GeTuiExtSdk.h>  
~~~
```

\* 2.查看：点击 `GeTuiExtSdk` 头文件，查看有两个方法，如下：

```
~~~  
/**  
 * 统计APNs到达情况  
 */  
+ (void)handelNotificationServiceRequest:(UNNotificationRequest *) request withComplete:(void (^)(void))completeBlock;  
~~~
```

/\*\*  
 \* 统计APNs到达情况和多媒体推送支持  
 \*/  
+ (void)handelNotificationServiceRequest:(UNNotificationRequest \*)request withAttachmentsComplete:(void (^)(NSArray\* attachments, NSArray\* errors))completeBlock;

,建议使用该方法接入

```
/**  
 * sdk销毁，资源释放  
 */  
+ (void)destory;
```

...

<mark class="note"> 接口使用说明:

</mark>

>1).如果需要多媒体推送,请使用第二种方法进行调用。接收到 APNs 通知后,SDK 判断是否有多媒体资源,如果多媒体资源存在则SDK下载资源,下载完成后以 Block 方式回调返回 attachments 资源数组对象和 errors 错误数组信息。

>2).多媒体大小限制:

| 时       | 资源    | 类型    | 大小    | 超       |
|---------|-------|-------|-------|---------|
|         | ----- | ----- | :     | :-----: |
| :-----: |       |       |       |         |
| s       | 图片    | 1     | <=10M | 120     |
|         | 音频    | 2     | <=5M  | 60s     |
|         | 视频    | 3     | <=50M | 18      |
| 0s      |       |       |       |         |

>3).可设置是否仅在 WIFI 下下载资源,参考服务端API。如果为True,数据网络环境下将不下载资源,展示通知不带附件。默认是:False,支持WIFI 和数据网络下下载。

>4). 资源URL格式注意:不支持中文及转译地址,请使用英文合法地址。

\* 3.接入: `NotificationService.m` 中有两个回调方法 `didReceiveNotificationRequest` 和 `serviceExtensionTimeWillExpire`。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。示例代码如下:

...

```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {  
  
    self.contentHandler = contentHandler;  
  
    self.bestAttemptContent = [request.content mutableCopy];  
  
    NSLog(@"-----将APNs信息交由个推处理-----");  
  
    [GeTuiExtSdk handelNotifica
```

```
tionServiceRequest:request withAttachmentsComplete:^(NSArray *attachments, NSArray* errors) {
```

```
le内容以项目实际需求而定 //注意：是否修改下发后的title
```

```
nt.title = [NSString stringWithFormat:@"%@" [需求而定]", self.bestAttemptContent.title]; //self.bestAttemptContent
```

```
.attachments = attachments; //设置通知中的多媒体附件 self.bestAttemptContent
```

```
到错误：%@",errors); //如果APNs处理有错误，可以在这里查看相关错误详情 NSLog(@"个推处理APNs消息遇
```

```
f.bestAttemptContent); //展示推送的回调处理需要放到个推回执完成的回调中 self.contentHandler(self.bestAttemptContent);
```

\* 4. 错误分析：多附件下载时每个附件遇到错误都将以 NSError\* 对象返回错误信息。当多媒体无法展示时，请注意检查 NSError 中错误信息。

|  | ErrorCode | 描述           | 解决  |
|--|-----------|--------------|---|
|  | 10001     | 多媒体地址错误,无法下载 | 请使用正常地址下发   |
|  | 10002     | Http URL 不支持 | 支持 ATS 使用 Https 地址或者将 NotificationService 配置为支持 Http 请求，如下图所示 |
|  | 10003     | 多媒体资源超过大小限制  | 请检查多媒体资源大小是否符合，更正后重新下发  |

址 Http 访问支持：</mark>

```
 ### 6.4 超时资源释放
```

\* 在 `NotificationService` 处理超时时，需调用 GtExtensionSdk 中 `destory` 方法，进行资源释放，防止因为资源未正常释放导致 crash。

统调用

lExpire {

调用

stAttemptContent);

~~~

//NotificationService处理超时时系

- (void)serviceExtensionTimeWil

//销毁SDK, 释放资源

[GeTuiExtSdk destory];

//结束 NotificationService

self.contentHandler(self.be

}

~~~

### 6.5 带 Notification Service

Extension 项目上传 AppStore 须知

\* 使用了 Notification Service Extension 的项目在制作待上传至 AppStore 的 IPA 包时, 编译设备需要选择 `Generic iOS Device`, 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。如下图所示:



## 7. VOIP 推送

\* Apple 在 iOS 8 后引入了一个基于 PushKit 框架的 VoIP 推送, 这可以使用户收到一个 VoIP 推送时唤醒 APP。有了这种 VoIP 的推送, 开发者们不需要让 APP 持续保持后台运行。当 APP 在后台被系统挂起后如果需要 APP 后台运行, 可以发送这个 VoIP 消息将会在后台唤醒 APP, 并执行制定的代码 (在弹出通知消息给用户之前)。

\* VoIP 服务需系统 iOS8 及以上系统支持。

\* VoIP 证书不区分开发和生产环境, VoIP 证书只有一个, 生产和开发都可用同一个证书。

### 7.1 添加 PushKit.Framework

库

`Optional` 方式引入。

1、导入 `PushKit.Framework` 库以

ushKit.h> `。

2、导入头文件 `#import <PushKit/P

3、AppDelegate 实现 `PKPushRegis

tryDelegate` 协议, 并实现 Delegate 中的接口。

协议-->

```

    @interface AppDelegate : UIResponder
    <UIApplicationDelegate, PKPushRegistryDelegate, GetTuiSdkDelegate> {
    }

    - (void)pushRegistry:(PKPushRegistry *)registry didUpdatePushCredentials:(PKPushCredentials *)credentials for
    Type:(NSString *)type {
        <!--具体实现参考 7.3 -->
    }

    - (void)pushRegistry:(PKPushRegistry *)registry didReceiveIncomingPushWithPayload:(PKPushPayload *)payload fo
    rType:(NSString *)type {
        <!--具体实现参考 7.4 -->
    }

```

图:

```

    ### 7.2 添加 VOIP 权限
    在 info.plist 中添加 voip 权限, 如

    ### 7.3 注册 VOIP Token

    1、注册 VoIP

    * 实现 VoIP 注册

    ...

    //注册 VoIP
    - (void)voipRegistration {
        dispatch_queue_t mainQueue
        = dispatch_get_main_queue();

        PKPushRegistry *voipRegistr
        y = [[PKPushRegistry alloc] initWithQueue:mainQueue];
        voipRegistry.delegate = sel
        f;

        voipRegistry.desiredPushType
        s = [NSSet setWithObject:PKPushTypeVoIP];
    }
    ...

```



```

* AppDelegate 的 `didFinishLaunc
hingWithOptions` 中调用 VoIP 注册

    ...
    - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
    {
        [GeTuiSdk runBackgroundEnable:YES]; // [是否允许APP后台运行]
        [GeTuiSdk lbsLocationEnable:YES andUserVerify:YES]; // [是否运行电子围栏Lbs功能和是否SDK主动请求用户定位]
        [GeTuiSdk setChannelId:@"GT-Channel"]; // 渠道
        // [1]:使用APPID/APPKEY/APPS
        ECRENT创建个推实例
        [GeTuiSdk startSdkWithAppId:[AppConfig GtAppId] appKey:[AppConfig GtAppKey] appSecret:[AppConfig GtAppSecret] delegate:self];
        // [2]:注册APNS
        [self registerRemoteNotification];
        // [3]:注册 VoIP
        [self voipRegistration];
        return YES;
    }
    ...

2、绑定 VoIP Token

    ...
    - (void)pushRegistry:(PKPushRegistry *)registry didUpdatePushCredentials:(PKPushCredentials *)credentials forType:(NSString *)type {
        NSString *voiptoken = [credentials.token.description stringByTrimmingCharactersInSet:[NSCharacterSet characterSetWithCharactersInString:@"<>"]];
        voiptoken = [voiptoken stringByReplacingOccurrencesOfString:@" " withString:@""];
        // 向个推服务器注册 VoipToken
        [GeTuiSdk registerVoipToken

```

```
, voiptoken);
```

}

### ### 7.4 接收 VOIP 推送处理具体业务

— — —

```
// TODO:接收 VoIP 推送中的 Pa
```

<!--TODO: 具体 VoIP 业务处理-

```
NSLog(@"[VoIP Payload]:%@", %
```

```
// 调用个推 VoIP 回执统计接口
[GeTuiSdk handleVoipNotific
```

}

## ## 8. AppLink 支持

[AppLink接入说明](../ios/appli

\* iOS应用、应用服务器、个推SDK、个推流程图，如下图所示：

```

```

\* 尤其需要注意在SDK在线和离线状态下分别不同的处理流程，APP 需要很好理解不同场景下的推送消息处理流程。