XCode集成

前言

- **本文档介绍XCode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：1.6.2.0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：

- 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：

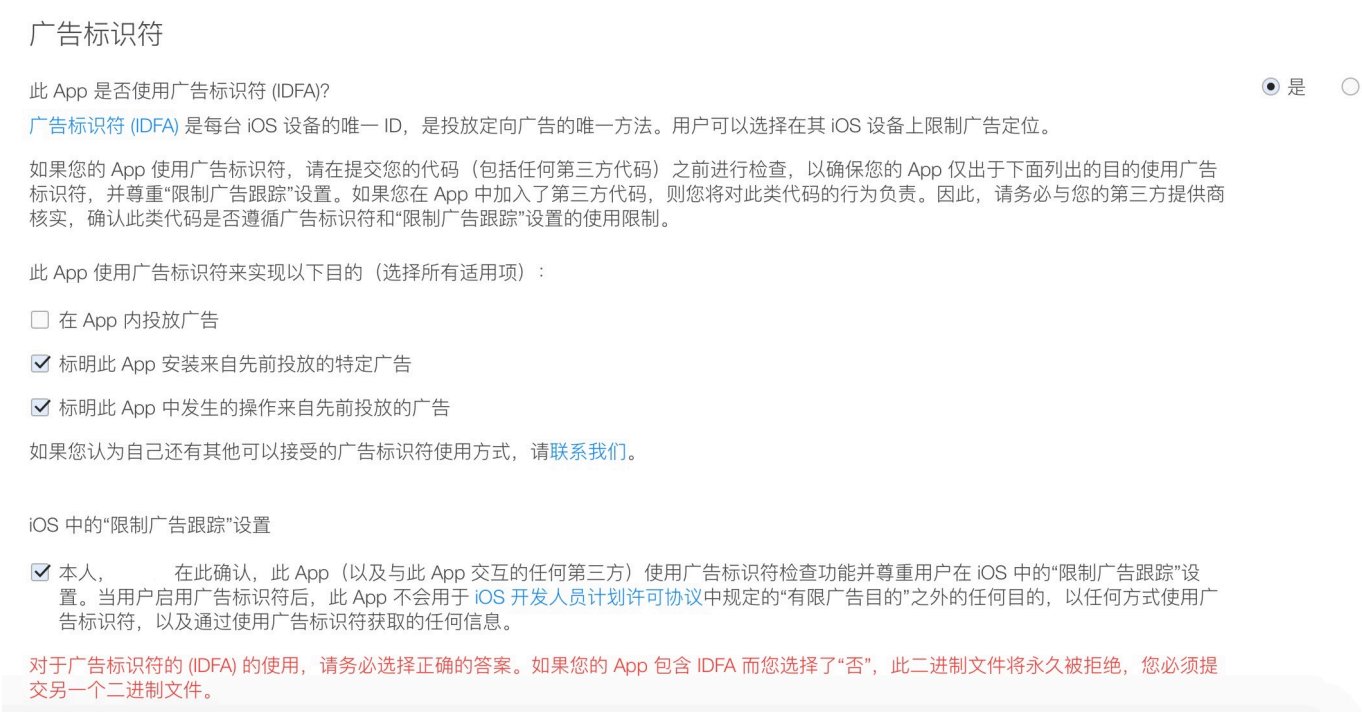
3. 添加个推SDK及相关配置

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.a文件和.h头文件)
  |     |    |- GeTuiSdk.h
  |     |    |- GeTuiExtSdk.h
  |     |    |- libGeTuiSdk-1.6.2.0.a
  |     |    |- libGtExtensionSdk-1.1.0.a
  |     |- GtSdkLib-noidfa/  (无IDFA版个推SDK.a文件和.h头文件)
  | - 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)
  |     |- 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。



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。

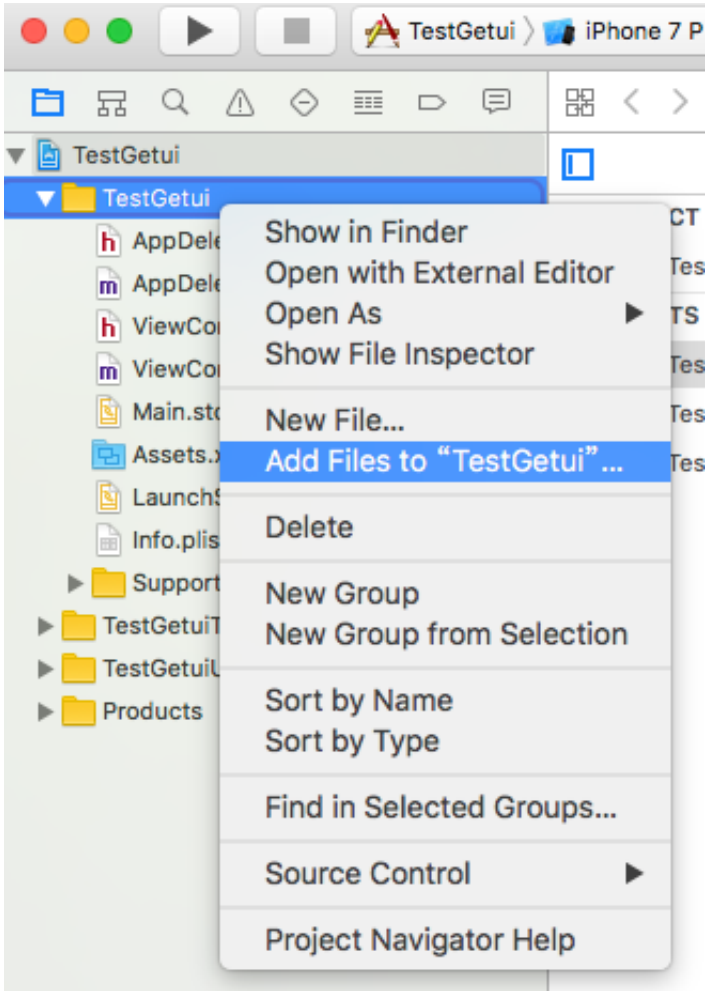
使用无 IDFA 版本SDK需删除 AdSupport.framework 库支持

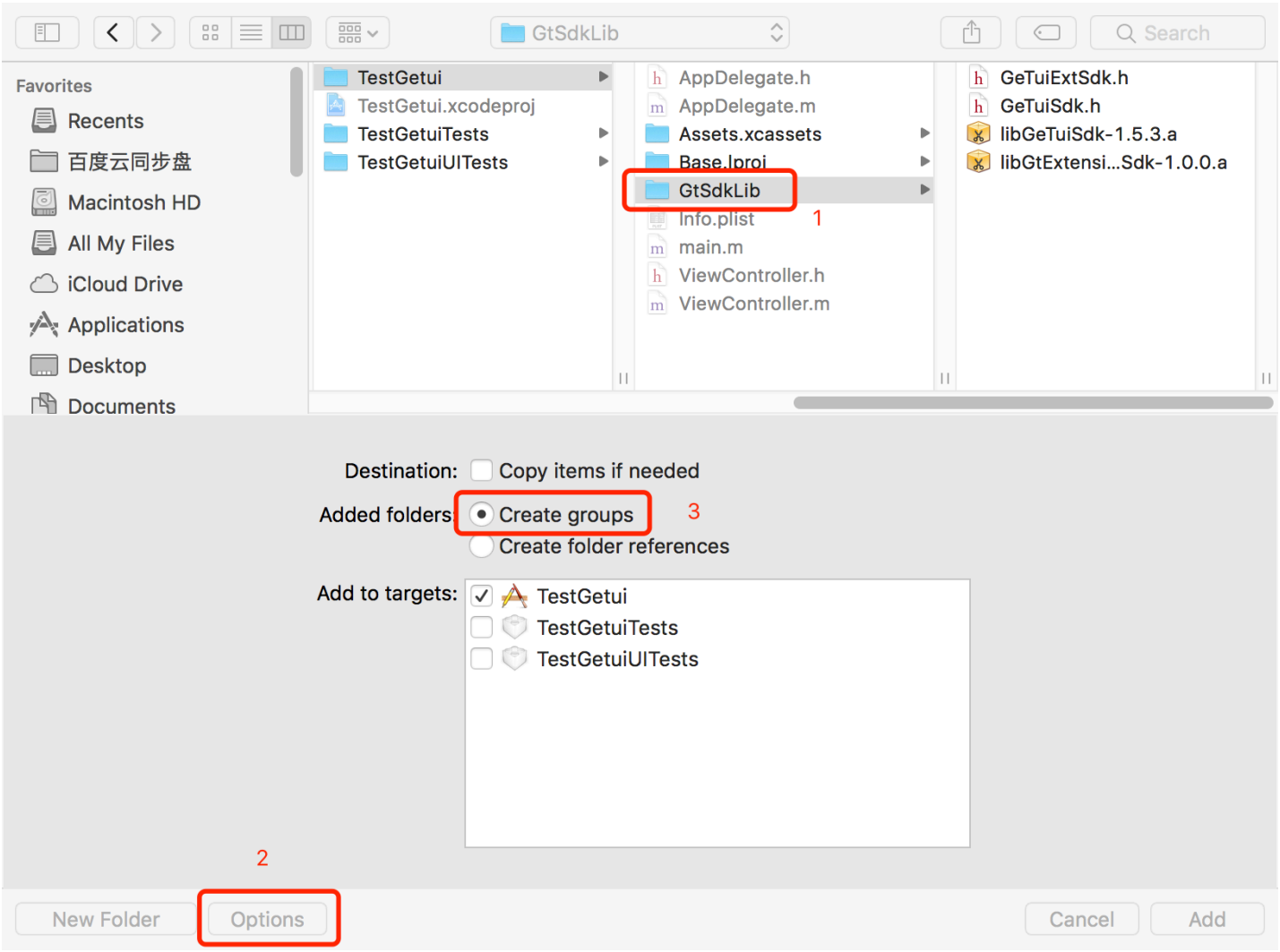
- libGeTuiSdk-{version}：个推SDK核心库文件，使用 `libo` 工具将 支持 `i386` 、 `x86_64` 、 `arm64` 、 `armv7` 的代码打包到了一起，所以这个库将同时支持 `simulator` 和 `device` 设备，支持 iOS 版本为7.0及以上。

- 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：


TestGetui
iPhone 7 P
TestGetui
TestGetui
AppDele
AppDele
ViewCo
ViewCo
Main.st
Assets.x
Launch
Info.plis
Support
TestGetuiT
TestGetuiU
Products
Show in Finder
Open with External Editor
Open As
Show File Inspector
New File...
Add Files to “TestGetui”...
Delete
New Group
New Group from Selection
Sort by Name
Sort by Type
Find in Selected Groups...
Source Control
Project Navigator Help



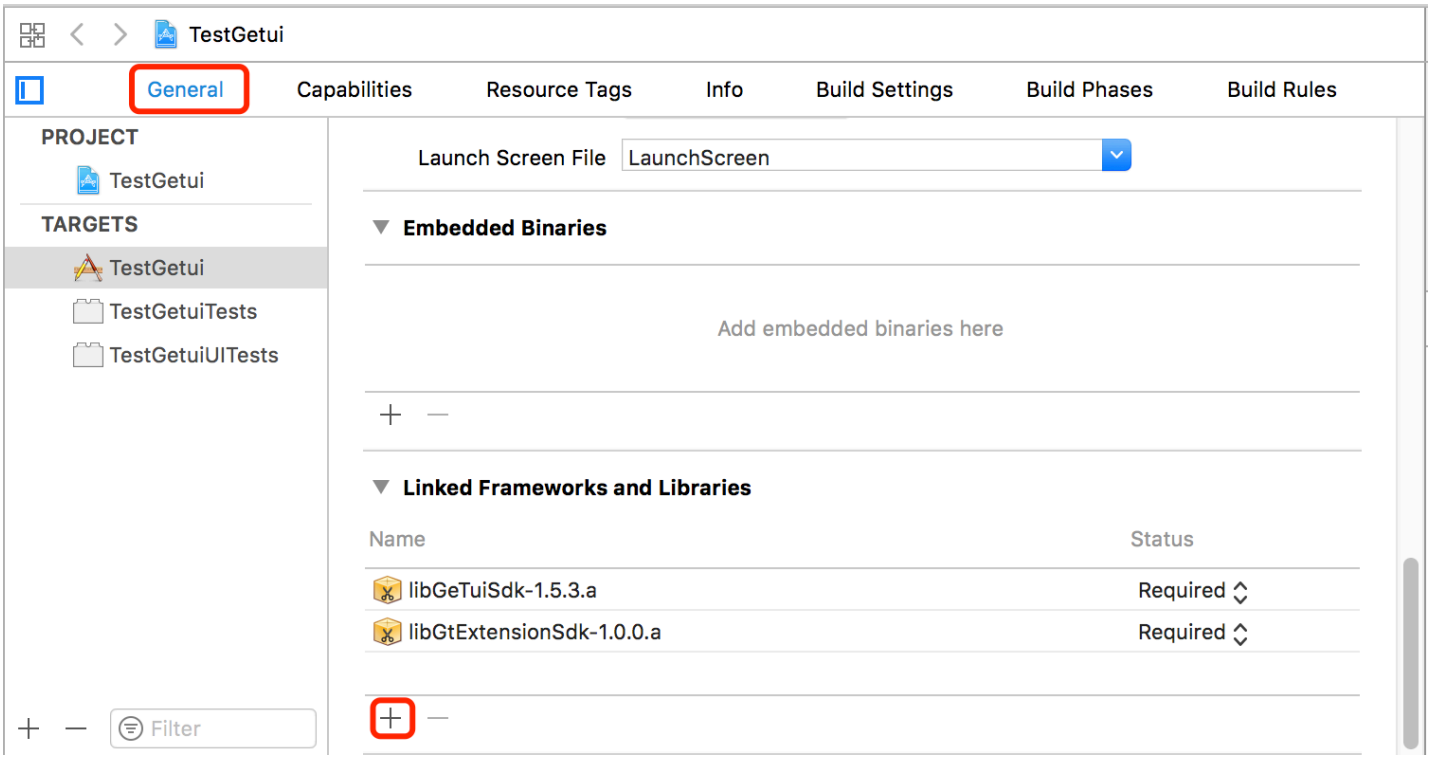
3.3 添加系统依赖库

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

- 在项目设置中添加以下系统库支持：

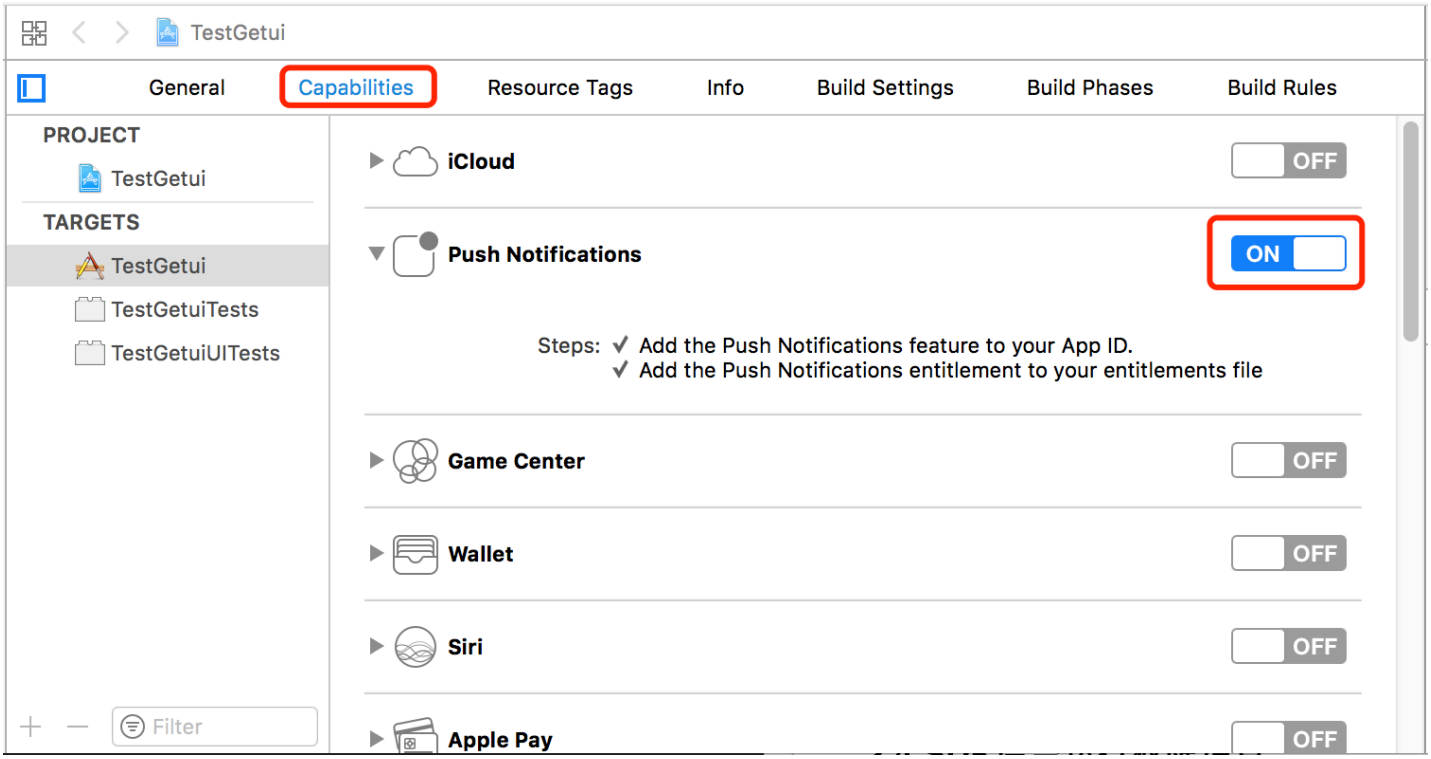
```
libc++.tbd
libz.tbd
libsqlite3.tbd
Security.framework
MobileCoreServices.framework
SystemConfiguration.framework
CoreTelephony.framework
AVFoundation.framework
CoreLocation.framework
UserNotifications.framework (iOS 10 及以上需添加)
AdSupport.framework    (如果使用无IDFA版本SDK，则需删除该 AdSupport 库)
```

- 删除 JavaScriptCore.framework 引用



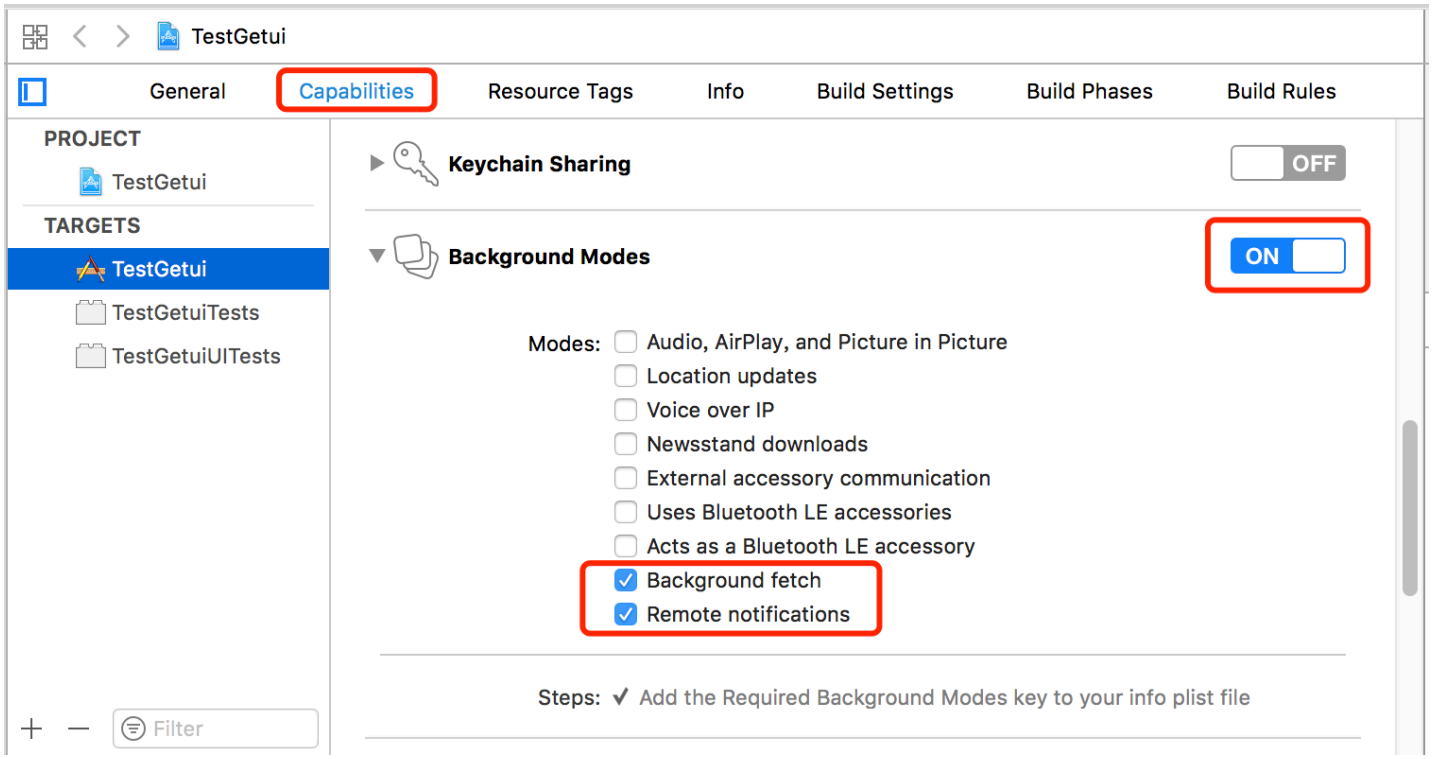
3.4 开启推送功能

- 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`：



3.5 后台运行权限设置

- 为了更好支持消息推送，SDK可定期抓取离线消息，提高消息到达率，需要配置后台运行权限：



- `Background fetch`：后台定期获取权限
- `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 "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) {
#if __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_10_0 // Xcode 8编译会调用
        UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotifi
cationCenter];
        center.delegate = self;
        [center requestAuthorizationWithOptions:(UNAuthorizationOptionBadge | UNAu
thorizationOptionSound | 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 | UIUserNotifi
cationTypeSound | UIUserNotificationTypeBadge);
        UIUserNotificationSettings *settings = [UIUserNotificationSettings setting
sForTypes:types categories:nil];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
        [[UIApplication sharedApplication] registerUserNotificationSettings:settin
gs];
#endif
    } else if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 8.0) {
        UIUserNotificationType types = (UIUserNotificationTypeAlert | UIUserNotifi
cationTypeSound | UIUserNotificationTypeBadge);
        UIUserNotificationSettings *settings = [UIUserNotificationSettings setting
sForTypes:types categories:nil];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
        [[UIApplication sharedApplication] registerUserNotificationSettings:settin
gs];
    } else {
        UIRemoteNotificationType apn_type = (UIRemoteNotificationType)(UIRemoteNot
ificationTypeAlert |
                                                                       UIRemoteNot
ificationTypeSound |
                                                                       UIRemoteNot
ificationTypeBadge);
        [[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 Background Fetch 接口回调处理

- iOS7.0 以后支持 APP 后台刷新数据，会回调 `performFetchWithCompletionHandler` 接口。为保证个推SDK的数据刷新，需在该回调接口中调用 `[GeTuiSdk resume]` 方法帮助个推 SDK 刷新数据。示例代码如下：

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

请参照【3.5】节的内容正确添加 `Background fetch` 权限

4.5 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 willPresentNotif
ication:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotificat
ionPresentationOptions))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 | UNNotificationPresen
tationOptionSound | 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 didReceiveNotifi
cationResponse:(UNNotificationResponse *)response withCompletionHandler:(void (^)(
))completionHandler {

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

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

    completionHandler();
}

#endif
```

4.6 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);
}

/** 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]:%@\n\n", [error localizedDescription]);
 }
```

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：

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:(NSD
ictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult resu
lt))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);
 }
```

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"];
```

- 可以在 `GeTuiSdkDidAliasAction` 方法中处理 绑定/解绑 响应：

```
- (void)GeTuiSdkDidAliasAction:(NSString *)action result:(BOOL)isSuccess sequenceN
um:(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"];
```

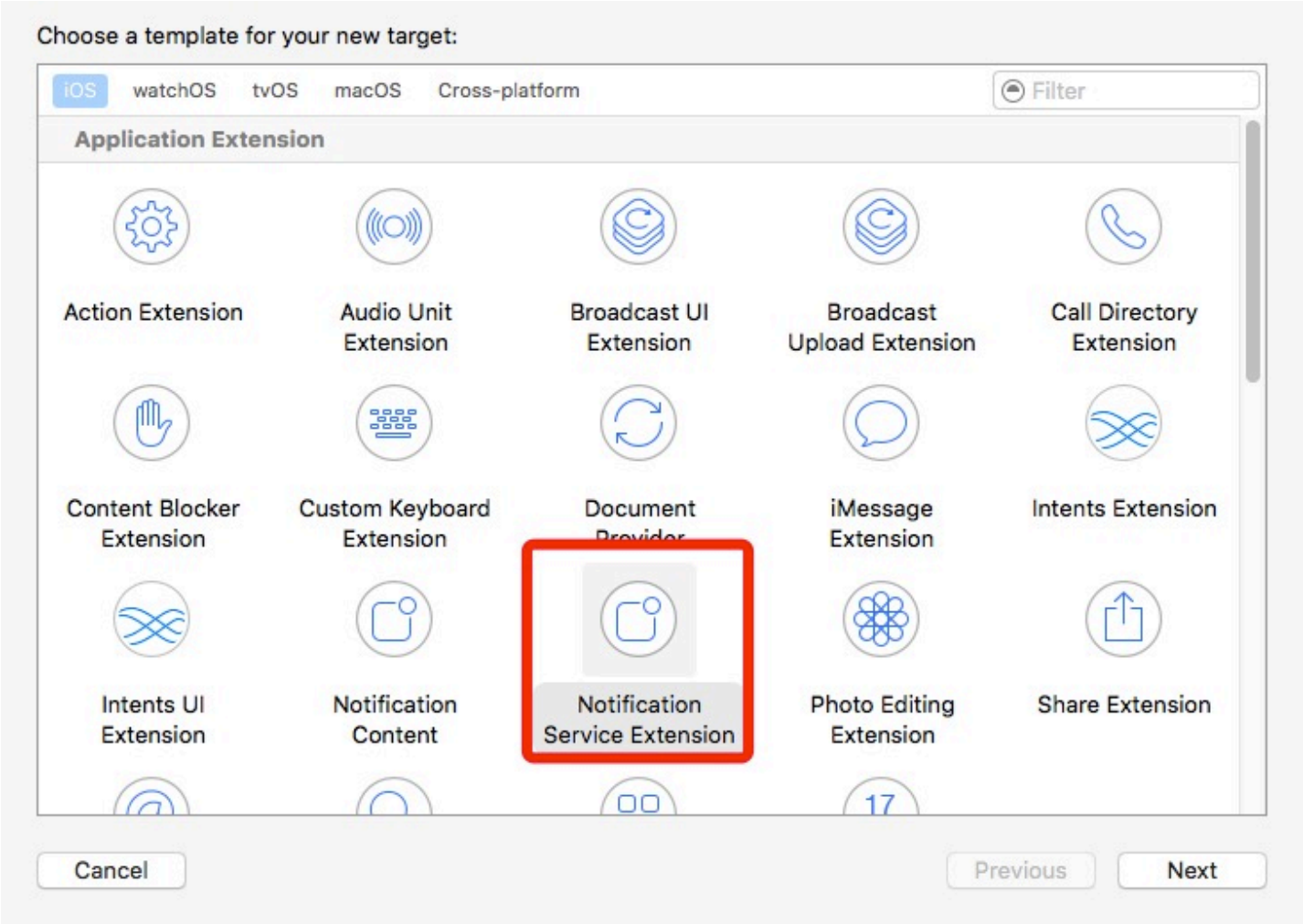
6.APNs 多媒体推送

- 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：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。

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`：

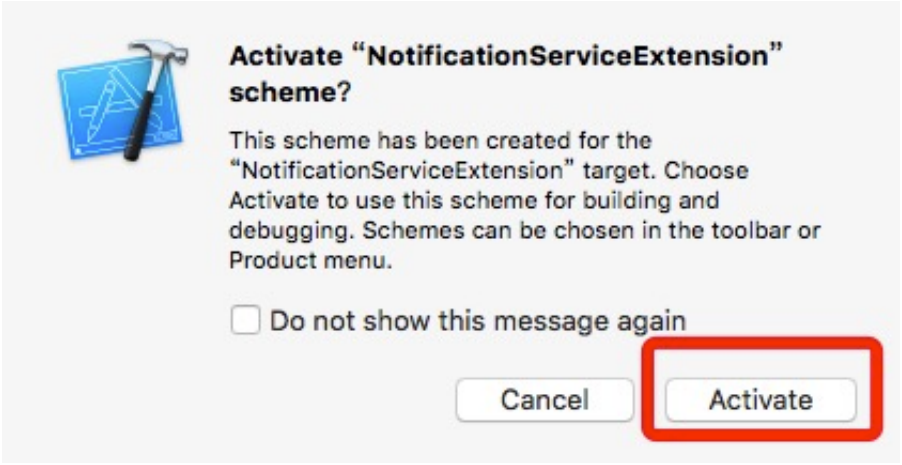


填写Target信息时需要注意以下两点：

- 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：



- 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 content, 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.tbz
libsqlite3.tbd
libGtExtensionSdk.a
```

需设置 NotificationService 最低运行版本为 10.0：

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 "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:^{

     self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [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);
    }];
}
```

- 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 "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;
```

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 | <=50 | 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。

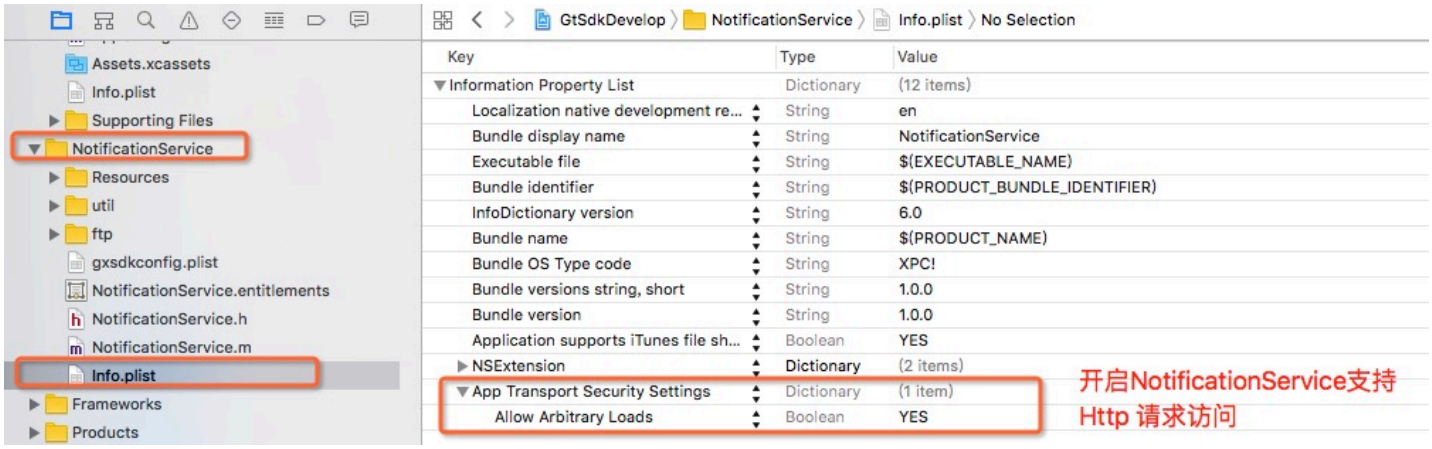
- 3.接入：`NotificationService.m` 中有两个回调方法 `didReceiveNotificationRequest` 和 `serviceExtensionTimeWillExpire` 。需要在 `didReceiveNotificationRequest` 中添加 GtExtensionSdk 的推送回执统计处理，由于回执操作是异步请求，因此待展示推送的回调处理需要放到回执完成的回调中。示例代码如下：

```
- (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) {

        //TODO:用户可以在这里处理通知样式的修改，eg:修改标题
        self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [modified]
", self.bestAttemptContent.title]; //修改展示title信息

        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：

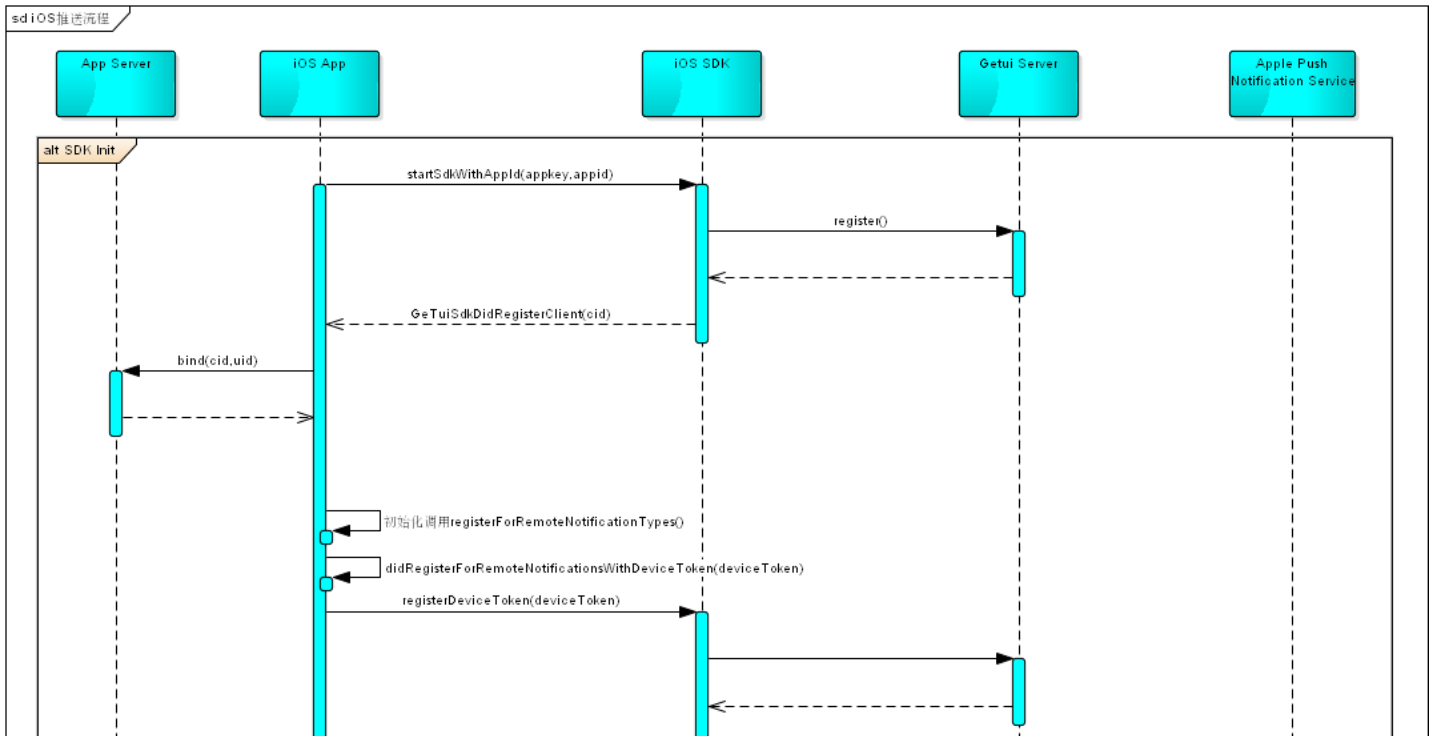


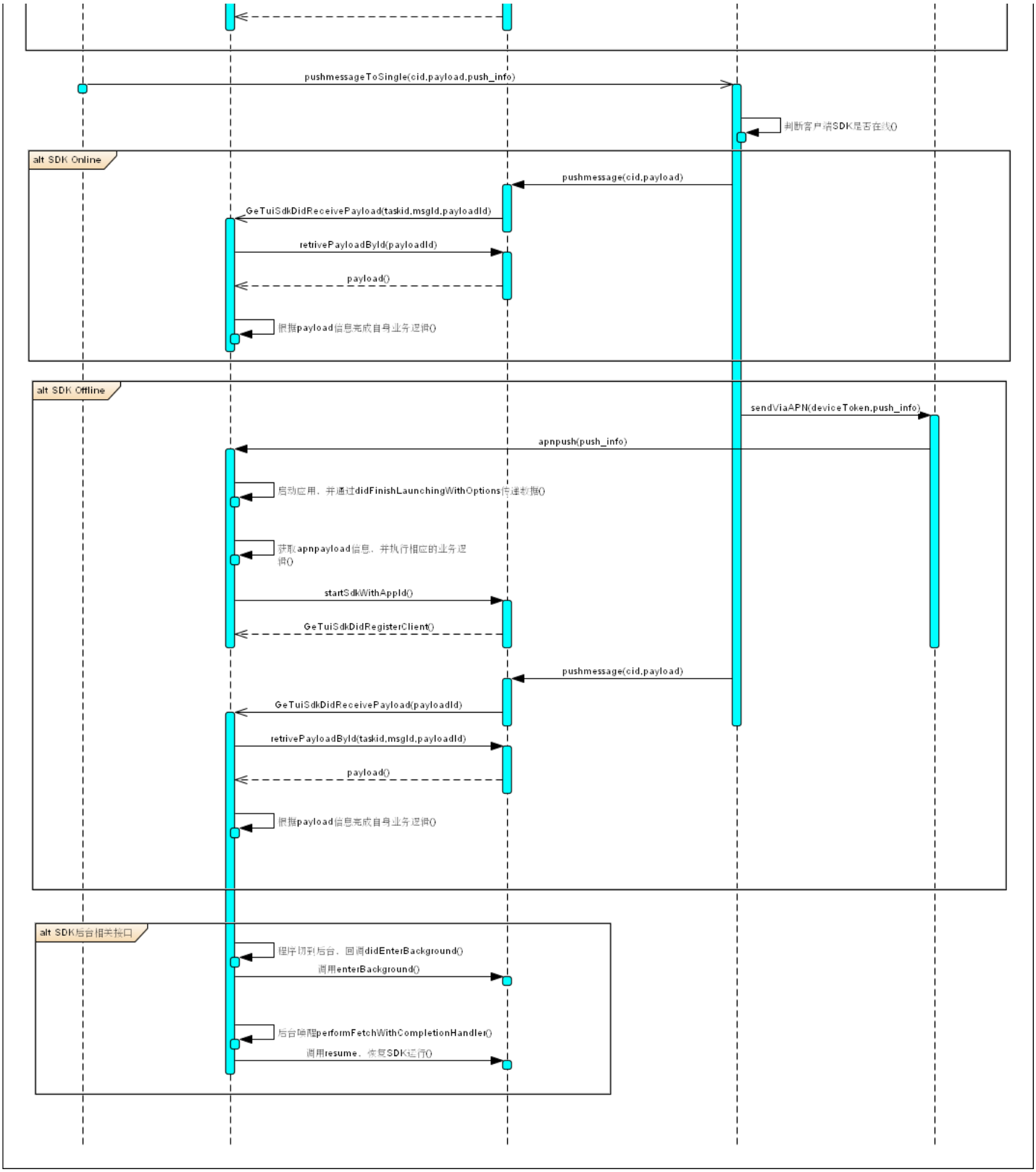
6.5 带 Notification Service Extension 项目上传 AppStore 须知

- 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. 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。