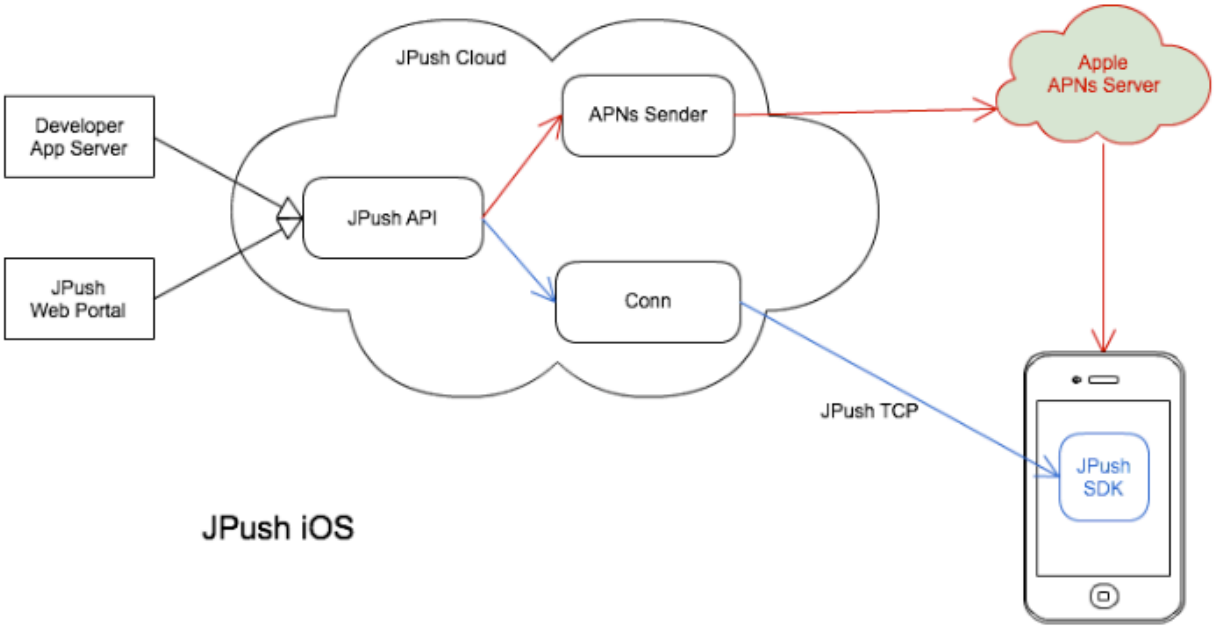
极光推送 v3.0.0

1. 简述：



JPush 包括 2 个部分，APNs 推送(代理)，与 JPush 应用内消息。红色部分是 APNs 推送，JPush 代理开发者的应用(需要基于开发者提供的应用证书)，向苹果 APNs 服务器推送。由 APNs Server 推送到 iOS 设备上。蓝色部分是 JPush 应用内推送部分，即 App 启动时，内嵌的 JPush SDK 会开启长连接到 JPush Server，从而 JPush Server 可以推送消息到 App 里

1) APNs 通知

APNs 通知是指通过向 Apple APNs 服务器发送通知，到达 iOS 设备，由 iOS 系统提供展现的推送。用户可以通过 IOS 系统的 “设置” >> “通知” 进行设置，开启或者关闭某一个 App 的推送能力。

JPush iOS SDK 不负责 APNs 通知的展现，只是向 JPush 服务器端上传 Device Token 信息，JPush 服务器端代理开发者向 Apple APNs 推送通知。

2) 应用内消息

应用内消息：JPush iOS SDK 提供的应用内消息功能，在 App 在前台时能够收到推送下来的消息。App 可使用此功能来做消息下发动作。

此消息不经过 APNs 服务器，完全由 JPush 提供功能支持。

3) APNs通知和应用内消息对比

JPush API v3 支持同时一次调用同时推送 APNs 通知与 JPush 应用内消息。这在某些应用场景里是有意义的。

	APNS	应用内消息
推送原则	由JPush服务器发送至APNS服务器，再下发到手机。	由JPush直接下发，每次推送都会尝试发送，如果用户在线则立即收到。否则保存为离线。
离线消息	离线消息由APNS服务器缓存按照Apple的逻辑处理。	用户不在线JPush server 会保存离线消息,时长默认保留一天。离线消息保留5条。
推送与证书环境	应用证书和推送指定的iOS环境匹配才可以收到。	自定义消息与APNS证书环境无关。
接收方式	应用退出，后台以及打开状态都能收到APNS	需要应用打开，与JPush 建立连接才能收到。
展示效果	如果应用后台或退出，会有系统的APNS提醒。 如果应用处于打开状态，则不展示。	非APNS，默认不展示。可通过获取接口自行编码处理。
处理函数	Apple提供的接口：didReceiveRemoteNotification	JPush提供的接口：networkDidReceiveMessage

2. SDK说明

- 支持的iOS版本为6.0及以上版本。
- 支持iOS版本为10.0以上的版本时需知。
  - Notification Service Extension证书配置时需要注意BundleID不能与Main Target一致，证书需要单独额外配置。
  - 请将Notification Service Extension中的Deployment Target设置为10.0。
  - 在XCode7或者更低的版本中删除Notification Service Extension所对应的Target。
  - 在XCode7或者更低的版本中请将引入的'UserNotifications.framework'删除。

3. SDK集成

(1) 创建应用获取AppKey用以标示该应用，上传待集成、发布的应用所使用的App ID的APNs 证书的p12文件，注意上传证书的Apple ID中的Bundle ID需和xcode一致

(2) 配置工程

1) 添加sdk

2) 添加系统库

CFNetwork、CoreFoundation、CoreTelephony、SystemConfiguration、CoreGraphics、Foundation、UIKit、Security、libz、AdSupport(获取IDFA需要，如果不使用IDFA，请不要添加)、UserNotifications(Xcode8及以上)、libresolv

(3) 注意事项

1) 如果你的工程需要支持小于7.0的iOS系统，请到Build Settings 关闭bitCode 选项，否则将无法正常编译通过。

2) 如使用Xcode8及以上环境开发，请开启Application Target的Capabilities->Push Notifications选项

3) 关于ios9的http支持问题

允许Xcode7支持Http传输方法

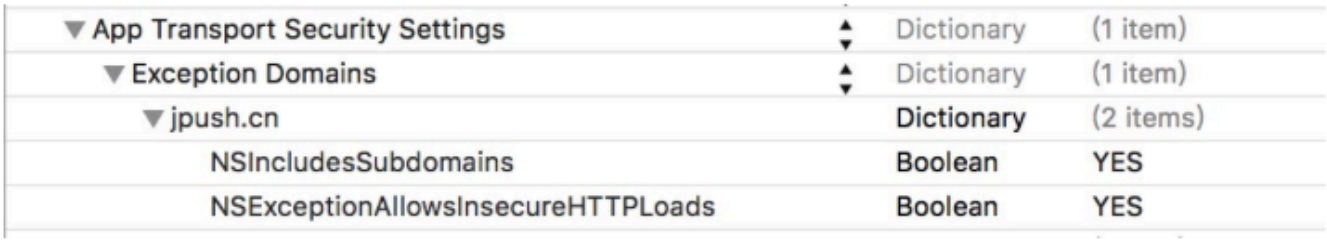
如果您使用的是2.1.9及以上的版本则不需要配置此步骤

如果用的是Xcode7或更新版本，需要在App项目的plist手动配置下key和值以支持http传输:

选择1：根据域名配置

- 在项目的info.plist中添加一个Key：NSAppTransportSecurity，类型为字典类型。
- 然后给它添加一个NSExceptionDomains，类型为字典类型；
- 把需要的支持的域添加給NSExceptionDomains。其中jpush.cn作为Key，类型为字典类型。
- 每个域下面需要设置2个属性：NSIncludesSubdomains、NSExceptionAllowsInsecureHTTPLoads。 两个属性均为Boolean类型，值分别为YES、YES。

如图:



选择2：全局配置

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSAllowsArbitraryLoads</key>
  <true/>
</dict>
```

(4) 初始化sdk

请将以下代码添加到 AppDelegate.m 引用头文件的位置。

```
// 引入JPush功能所需头文件
#import "JPUSHService.h"
// iOS10注册APNs所需头文件
#ifdef NSFoundationVersionNumber_iOS_9_x_Max
#import <UserNotifications/UserNotifications.h>
#endif
// 如果需要使用idfa功能所需要引入的头文件（可选）
#import <AdSupport/AdSupport.h>
```

为AppDelegate添加Delegate。

参考代码:

```
@interface AppDelegate ()<JPUSHRegisterDelegate>

@end
```

添加初始化APNs代码

请将以下代码添加到

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

```
  //Required
  //notice: 3.0.0及以后版本注册可以这样写，也可以继续用之前的注册方式
  JPUSHRegisterEntity * entity = [[JPUSHRegisterEntity alloc] init];
  entity.types = JPAuthorizationOptionAlert|JPAuthorizationOptionBadge|JPAuthorizationOptionSound;
  if ([[UIDevice currentDevice].systemVersion floatValue] >= 8.0) {
    // 可以添加自定义categories
    // NSSet<UNNotificationCategory *> *categories for iOS10 or later
    // NSSet<UIUserNotificationCategory *> *categories for iOS8 and iOS9
  }
  [JPUSHService registerForRemoteNotificationConfig:entity delegate:self];
```

添加初始化JPush代码

请将以下代码添加到

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

```
  // Optional
  // 获取IDFA
  // 如需使用IDFA功能请添加此代码并在初始化方法的advertisingIdentifier参数中填写对应值
  NSString *advertisingId = [[[ASIdentifierManager sharedManager] advertisingIdentifier] UUIDString];

  // Required
  // init Push
  // notice: 2.1.5版本的SDK新增的注册方法，改成可上报IDFA，如果没有使用IDFA直接传nil
  // 如需继续使用pushConfig.plist文件声明appKey等配置内容，请依旧使用[JPUSHService setupWithOption:launchOptions]方式初始化。
  [JPUSHService setupWithOption:launchOptions appKey:appKey
                        channel:channel
               apsForProduction:isProduction
          advertisingIdentifier:advertisingId];
```

- appKey
  - 填写管理Portal上创建应用后自动生成的AppKey值。请确保应用内配置的 AppKey 与 Portal 上创建应用后生成的 AppKey 一致。
- channel
  - 指明应用程序包的下载渠道，为方便分渠道统计，具体值由你自行定义，如：App Store。
- apsForProduction
  - 1.3.1版本新增，用于标识当前应用所使用的APNs证书环境。
  - 0 (默认值)表示采用的是开发证书，1 表示采用生产证书发布应用。
  - 注：此字段的值要与Build Settings的Code Signing配置的证书环境一致。

注册APNs成功并上报DeviceToken

请在AppDelegate.m实现该回调方法并添加回调方法中的代码

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

  /// Required - 注册 DeviceToken
  [JPUSHService registerDeviceToken:deviceToken];
}
```

实现注册APNs失败接口（可选）

```
- (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWi
thError:(NSError *)error {
  //Optional
  NSLog(@"did Fail To Register For Remote Notifications With Error: %@", error);
}
```

```
#pragma mark- JPUSHRegisterDelegate

// iOS 10 Support
- (void)jpushNotificationCenter:(UNUserNotificationCenter *)center willPresentNotification
:(UNNotification *)notification withCompletionHandler:(void (^)(NSInteger))completionHandl
er {
  // Required
  NSDictionary * userInfo = notification.request.content.userInfo;
  if([notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class]]) {
    [JPUSHService handleRemoteNotification:userInfo];
  }
  completionHandler(UNNotificationPresentationOptionAlert); // 需要执行这个方法，选择是否提醒用户
，有Badge、Sound、Alert三种类型可以选择设置
}

// iOS 10 Support
- (void)jpushNotificationCenter:(UNUserNotificationCenter *)center didReceiveNotificationR
esponse:(UNNotificationResponse *)response withCompletionHandler:(void (^)())completionHan
dler {
  // Required
  NSDictionary * userInfo = response.notification.request.content.userInfo;
  if([response.notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class
]]) {
    [JPUSHService handleRemoteNotification:userInfo];
  }
  completionHandler();  // 系统要求执行这个方法
}
```

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionar
y *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler {

  // Required, iOS 7 Support
  [JPUSHService handleRemoteNotification:userInfo];
  completionHandler(UIBackgroundFetchResultNewData);
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionar
y *)userInfo {

  // Required,For systems with less than or equal to iOS6
  [JPUSHService handleRemoteNotification:userInfo];
}
```

JPush SDK 相关事件监听

建议开发者加上API里面提供的以下类型的通知：

extern NSString *const kJPFNetworkIsConnectingNotification; // 正在连接中

extern NSString * const kJPFNetworkDidSetupNotification; // 建立连接

extern NSString * const kJPFNetworkDidCloseNotification; // 关闭连接

extern NSString * const kJPFNetworkDidRegisterNotification; // 注册成功

extern NSString *const kJPFNetworkFailedRegisterNotification; //注册失败

extern NSString * const kJPFNetworkDidLoginNotification; // 登录成功

> 温馨提示：
> Registration id 需要添加注册kJPFNetworkDidLoginNotification通知的方法里获取，也可以调用[registrationIDCompletionHandler:]方法，通过completionHandler获取

extern NSString * const kJPFNetworkDidReceiveMessageNotification; // 收到自定义消息(非APNs)

其中，kJPFNetworkDidReceiveMessageNotification传递的数据可以通过NSNotification中的userInfo方法获取，包括标题、内容、extras信息等

在方法- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *) launchOptions 加入下面的代码：

```
    NSNotificationCenter *defaultCenter = [NSNotificationCenter defaultCenter];
    [defaultCenter addObserver:self selector:@selector(networkDidReceiveMessage:) name:kJP
FNetworkDidReceiveMessageNotification object:nil];
```

实现回调方法 networkDidReceiveMessage

```
    - (void)networkDidReceiveMessage:(NSNotification *)notification {
        NSDictionary * userInfo = [notification userInfo];
        NSString *content = [userInfo valueForKey:@"content"];
        NSDictionary *extras = [userInfo valueForKey:@"extras"];
        NSString *customizeField1 = [extras valueForKey:@"customizeField1"]; //服务端传递的Ex
tras附加字段，key是自己定义的

    }
```

(5) 相关接口

API setBadge

设置JPush服务器中存储的badge值

接口定义

```
+ (BOOL)setBadge:(int)value
```

参数说明

- value 取值范围：[0,99999]

设置badge值，本地仍须调用UIApplication:setApplicationIconBadgeNumber函数

- 返回值
  - 在value的取值区间内返回 TRUE，否则返回FALSE

API resetBadge

清空JPush服务器中存储的badge值，即 [setBadge:0]

接口定义

```
+ (void)resetBadge
```

iOS 设备收到一条本地通知，用户点击通知打开应用时，应用程序根据状态不同进行处理需在 AppDelegate 中的以下两个方法中添加代码以获取本地通知内容

- 如果 App 状态为未运行，此函数将被调用，如果launchOptions包含UIApplicationLaunchOptionsLocalNotificationKey表示用户点击本地通知导致app被启动运行；如果不含有对应键值则表示 App 不是因点击本地通知而被启动，可能为直接点击icon被启动或其他。

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions;
// 本地通知内容获取：NSDictionary *localNotification = [launchOptions objectForKey: UIApplicationLaunchOptionsLocalNotificationKey]
```

- 如果 App状态为正在前台或者后台运行，那么此函数将被调用，并且可通过AppDelegate的applicationState是否为UIApplicationStateActive判断程序是否在前台运行。此种情况在此函数中处理：

```
// NS_DEPRECATED_IOS(4_0, 10_0, "Use UserNotifications Framework's -[UNUserNotificationCenterDelegate willPresentNotification:withCompletionHandler:] or -[UNUserNotificationCenterDelegate didReceiveNotificationResponse:withCompletionHandler:]")
- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification;
// 本地通知为notification
```

- 在iOS 10以上上述方法将被系统废弃，由新增UserNotifications Framework中的-[UNUserNotificationCenterDelegate willPresentNotification:withCompletionHandler:] 或者 -[UNUserNotificationCenterDelegate didReceiveNotificationResponse:withCompletionHandler:]方法替代。为此，SDK封装了JPUSHRegisterDelegate协议，只需实现相应的协议方法即可适配iOS10新增的delegate方法，与上述远程推送新回调方法一致，也即是如下方法：

```
- (void)jpushNotificationCenter:(UNUserNotificationCenter *)center willPresentNotification:(UNNotification *)notification withCompletionHandler:(void (^) (NSInteger))completionHandler;
   // if (![notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class]])
{
   // 本地通知为notification
   // }

- (void)jpushNotificationCenter:(UNUserNotificationCenter *)center didReceiveNotificationResponse:(UNNotificationResponse *)response withCompletionHandler: (void (^)())completionHandler;
  // if (![response.notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class]]) {
  // 本地通知为response.notification
  // }
```

Method AddNotification

支持版本

v2.1.9及后续版本

功能说明

API 用于注册或更新推送（支持iOS10，并兼容iOS10以下版本）

接口定义

```
+ (void)addNotification:(JPushNotificationRequest *)request;
```

参数说明

- request [JPushNotificationRequest]实体类型，可传入推送的属性

调用说明

request中传入已有推送的request.requestIdentifier即更新已有的推送，否则为注册新推送。

代码示例

```
- (void)testAddNotification {
  JPushNotificationContent *content = [[JPushNotificationContent alloc] init];
  content.title = @"Test Notifications";
  content.subtitle = @"2016";
  content.body = @"This is a test code";
  content.badge = @1;
  content.categoryIdentifier = @"Custom Category Name";

  // 5s后提醒 iOS 10 以上支持
  JPushNotificationTrigger *trigger1 = [[JPushNotificationTrigger alloc] init];
  trigger1.timeInterval = 5;
  //每小时重复 1 次 iOS 10 以上支持
  JPushNotificationTrigger *trigger2 = [[JPushNotificationTrigger alloc] init];
  trigger2.timeInterval = 3600;
  trigger2.repeat = YES;

  //每周一早上8：00提醒，iOS10以上支持
  NSDateComponents *components = [[NSDateComponents alloc] init];
  components.weekday = 2;
  components.hour = 8;
  JPushNotificationTrigger *trigger3 = [[JPushNotificationTrigger alloc] init];
  trigger3.dateComponents = components;
  trigger3.repeat = YES;

  //#import <CoreLocation/CoreLocation.h>
  //一到某地点提醒，iOS8以上支持
  CLRegion *region = [[CLRegion alloc] initCircularRegionWithCenter:CLLocationCoordinate2D
Make(0, 0) radius:0 identifier:@"test"];
  JPushNotificationTrigger *trigger4 = [[JPushNotificationTrigger alloc] init];
  trigger4.region = region;
```

```
  //5s后提醒，iOS10以下支持
  JPushNotificationTrigger *trigger5 = [[JPushNotificationTrigger alloc] init];
  trigger5.fireDate = [NSDate dateWithTimeIntervalSinceNow:5];

  JPushNotificationRequest *request = [[JPushNotificationRequest alloc] init];
  request.requestIdentifier = @"sampleRequest";
  request.content = content;
  request.trigger = trigger1;//trigger2;//trigger3;//trigger4;//trigger5;
  request.completionHandler = ^(id result) {
    NSLog(@"结果返回：%@", result);
  };
  [JPUSHService addNotification:request];
}
```

Method RemoveNotification

支持版本

v2.1.9及后续版本

功能说明

API 用于移除待推送或已在通知中心显示的推送（支持iOS10，并兼容iOS10以下版本）

接口定义

```
+ (void)removeNotification:(JPushNotificationIdentifier *)identifier;
```

参数说明

- identifier [JPushNotificationIdentifier]实体类型

调用说明

- iOS10以上identifier设置为nil，则移除所有在通知中心显示推送和待推送请求，也可以通过设置identifier.delivered和identifier.identifiers来移除相应在通知中心显示推送或待推送请求，identifier.identifiers如果设置为nil或空数组则移除相应标志下所有在通知中心显示推送或待推送请求；iOS10以下identifier设置为nil，则移除所有推送，identifier.delivered属性无效，另外可以通过identifier.notificationObj传入特定推送对象来移除此推送。

代码示例

```
- (void)testRemoveNotification {
  JPushNotificationIdentifier *identifier = [[JPushNotificationIdentifier alloc] init];
  identifier.identifiers = @[@"sampleRequest"];
  identifier.delivered = YES;  //iOS10以上有效，等于YES则在通知中心显示的里面移除，等于NO则为在待推送的里面移除；iOS10以下无效
  [JPUSHService removeNotification:identifier];
}
```

```
- (void)testRemoveAllNotification {
  [JPUSHService removeNotification:nil];  // iOS10以下移除所有推送；iOS10以上移除所有在通知中心显示推送和待推送请求

//  //iOS10以上支持
//  JPushNotificationIdentifier *identifier = [[JPushNotificationIdentifier alloc] init];
//  identifier.identifiers = nil;
//  identifier.delivered = YES;  //等于YES则移除所有在通知中心显示的，等于NO则为移除所有待推送的
//  [JPUSHService removeNotification:identifier];
}
```

Method FindNotification

支持版本

v2.1.9及后续版本

功能说明

API 用于查找推送（支持iOS10，并兼容iOS10以下版本）

接口定义

```
+ (void)findNotification:(JPushNotificationIdentifier *)identifier;
```

参数说明

- identifier [JPushNotificationIdentifier]实体类型

调用说明

- iOS10以上可以通过设置identifier.delivered和identifier.identifiers来查找相应在通知中心显示推送或待推送请求，identifier.identifiers如果设置为nil或空数组则返回相应标志下所有在通知中心显示推送或待推送请求；iOS10以下identifier.delivered属性无效，identifier.identifiers如果设置nil或空数组则返回所有推送。
- 须要设置identifier.findCompletionHandler回调才能得到查找结果，通过(NSArray *results)返回相应对象数组。

代码示例

```
- (void)testFindNotification {
  JPushNotificationIdentifier *identifier = [[JPushNotificationIdentifier alloc] init];
  identifier.identifiers = @[@"sampleRequest"];
  identifier.delivered = YES;  //iOS10以上有效，等于YES则在通知中心显示的里面查找，等于NO则在待推送的里面查找；iOS10以下无效
  identifier.findCompletionHandler = ^(NSArray *results) {
  NSLog(@"返回结果为：%@", results); // iOS10以下返回UILocalNotification对象数组，iOS10以上根据delivered传入值返回UNNotification或UNNotificationRequest对象数组
};
  [JPUSHService findNotification:identifier];
}
```

```
- (void)testFindAllNotification {
  JPushNotificationIdentifier *identifier = [[JPushNotificationIdentifier alloc] init];
  identifier.identifiers = nil;
  identifier.delivered = YES;  //iOS10以上有效，等于YES则查找所有在通知中心显示的，等于NO则为查找所有待推送的；iOS10以下无效
  identifier.findCompletionHandler = ^(NSArray *results) {
  NSLog(@"返回结果为：%@", results); // iOS10以下返回UILocalNotification对象数组，iOS10以上根据delivered传入值返回UNNotification或UNNotificationRequest对象数组
};
  [JPUSHService findNotification:identifier];
}
```

(6) 标签与别名

提供几个相关 API 用来设置别名（alias）与标签（tags）。
这几个 API 可以在 App 里任何地方调用。

别名 alias

为安装了应用程序的用户，取个别名来标识。以后给该用户 Push 消息时，就可以用此别名来指定。
每个用户只能指定一个别名。
同一个应用程序内，对不同的用户，建议取不同的别名。这样，尽可能根据别名来唯一确定用户。
系统不限定一个别名只能指定一个用户。如果一个别名被指定到了多个用户，当给指定这个别名发消息时，服务器端API会同时给这多个用户发送消息。
举例：在一个用户要登录的游戏中，可能设置别名为 userid。游戏运营时，发现该用户 3 天没有玩游戏了，则根据 userid 调用服务器端API发通知到客户端提醒用户。

标签 tag

为安装了应用程序的用户，打上标签。其目的主要是方便开发者根据标签，来批量下发 Push 消息。
可为每个用户打多个标签。
举例： game, old_page, women

Method - setTagsWithAlias (with Callback)

调用此 API 来同时设置别名与标签，支持回调函数。
需要理解的是，这个接口是覆盖逻辑，而不是增量逻辑。即新的调用会覆盖之前的设置。
在之前调用过后，如果需要再次改变别名与标签，只需要重新调用此 API 即可。

支持的版本

开始支持的版本：1.4.0

接口定义

```
+ (void)setTags:(NSSet *)tags alias:(NSString *)alias callbackSelector:(SEL)cbSelector object:(id)theTarget;
```

参数说明

- alias
  - nil 此次调用不设置此值。
  - 空字符串 （@""）表示取消之前的设置。
  - 每次调用设置有效的别名，覆盖之前的设置。
  - 有效的别名组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：alias 命名长度限制为 40 字节。（判断长度需采用UTF-8编码）
- tags
  - nil 此次调用不设置此值。
  - 空集合（[NSSet set]）表示取消之前的设置。
  - 集合成员类型要求为NSString类型
  - 每次调用至少设置一个 tag，覆盖之前的设置，不是新增。
  - 有效的标签组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：每个 tag 命名长度限制为 40 字节，最多支持设置 1000 个 tag，但总长度不得超过7K 字节。（判断长度需采用UTF-8编码）
  - 单个设备最多支持设置 1000 个 tag。App 全局 tag 数量无限制。
- callbackSelector
  - nil 此次调用不需要 Callback。
  - 用于回调返回对应的参数 alias, tags。并返回对应的状态码：0为成功，其他返回码请参考错误码定义。
  - 回调函数请参考SDK 实现。

- theTarget
  - 参数值为实现了callbackSelector的实例对象。
  - nil 此次调用不需要 Callback。

```
-(void)tagsAliasCallback:(int)iResCode
                    tags:(NSSet*)tags
                   alias:(NSString*)alias
{
    NSLog(@"rescode: %d, \ntags: %@, \nalias: %@\n", iResCode, tags , alias);
}
```

Method - setTagsWithAlias (background)

调用此 API 在后台同时设置别名与标签，不需要处理设置结果，SDK会自动进行失败重试
需要理解的是，这个接口是覆盖逻辑，而不是增量逻辑。即新的调用会覆盖之前的设置。
在之前调用过后，如果需要再次改变别名与标签，只需要重新调用此 API 即可。
需要注意，该background模式的设置和非background的设置是两种不同的设置，互相不影响，意味着，非background的设置不会终止当前进行的background设置，除非另一个background设置发生。

接口定义

```
+ (void)setTags:(NSSet *)tags aliasInbackground:(NSString *)alias;
```

参数说明

- alias
  - nil 此次调用不设置此值。
  - 空字符串 （@""）表示取消之前的设置。
  - 每次调用设置有效的别名，覆盖之前的设置。
  - 有效的别名组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：alias 命名长度限制为 40 字节。（判断长度需采用UTF-8编码）
- tags
  - nil 此次调用不设置此值。
  - 空集合（[NSSet set]）表示取消之前的设置。
  - 集合成员类型要求为NSString类型
  - 每次调用至少设置一个 tag，覆盖之前的设置，不是新增。
  - 有效的标签组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：每个 tag 命名长度限制为 40 字节，最多支持设置 1000 个 tag，但总长度不得超过7K字节。（判断长度需采用UTF-8编码）
  - 单个设备最多支持设置 1000 个 tag。App 全局 tag 数量无限制。

```
[JPUSHService setTags:tags aliasInbackground:alias];
```

Method - setTagsWithAlias (with block)

调用此 API 来同时设置别名与标签，通过block来返回设置别名与标签的结果。
需要理解的是，这个接口是覆盖逻辑，而不是增量逻辑。即新的调用会覆盖之前的设置。
在之前调用过后，如果需要再次改变别名与标签，只需要重新调用此 API 即可。

支持的版本

开始支持的版本：2.1.0

接口定义

```
+ (void)setTags:(NSSet *)tags alias:(NSString *)alias fetchCompletionHandle:(void (^)(int iResCode, NSSet *iTags, NSString *iAlias))completionHandler
```

参数说明

- alias
  - nil 此次调用不设置此值。
  - 空字符串 （@""）表示取消之前的设置。
  - 每次调用设置有效的别名，覆盖之前的设置。

    - 有效的别名组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
    - 限制：alias 命名长度限制为 40 字节。（判断长度需采用UTF-8编码）
- tags
    - nil 此次调用不设置此值。
    - 空集合（[NSSet set]）表示取消之前的设置。
    - 集合成员类型要求为NSString类型
    - 每次调用至少设置一个 tag，覆盖之前的设置，不是新增。
    - 有效的标签组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
    - 限制：每个 tag 命名长度限制为 40 字节，最多支持设置 1000 个 tag，但总长度不得超过7K字节。（判断长度需采用UTF-8编码）
    - 单个设备最多支持设置 1000 个 tag。App 全局 tag 数量无限制。
- (void (^)(int iResCode, NSSet *iTags*, *NSString* iAlias))completionHandler
    - completionHandler用于处理设置返回结果
    - iResCode返回的结果状态码
    - iTags和iAlias返回设置的tag和alias

```
[JPUSHService setTags:tags alias:alias fetchCompletionHandle:^(int iResCode, NSSet *iTags, NSString *iAlias){
 NSLog(@"rescode: %d, \ntags: %@, \nalias: %@\n", iResCode, iTags, iAlias);
}];
```

Method - setTags

调用此 API 来设置标签，支持回调函数。

该方法是 setTagsWithAlias (with Callback) 的简化版本，用于只变更标签的情况。

> 使用建议
> 如果待设置的 alias / tags 是动态的，有可能在调用 setTagsWithAlias 时因为 alias / tags 无效而整调用失败。
> 调用此方法只设置 tags，可以排除可能的无效的 alias 对本次调用的影响。

支持的版本

开始支持的版本：1.4.0

接口定义

```
+ (void)setTags:(NSSet *)tags callbackSelector:(SEL)cbSelector object:(id)theTarget;
```

参数说明

- tags
  - nil 此次调用不设置此值。
  - 空集合（[NSSet set]）表示取消之前的设置。
  - 每次调用至少设置一个 tag，覆盖之前的设置，不是新增。
  - 有效的标签组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：每个 tag 命名长度限制为 40 字节，最多支持设置 1000 个tag，但总长度不得超过7K字节。（判断长度需采用UTF-8编码）
  - 单个设备最多支持设置 1000 个 tag。App 全局 tag 数量无限制。
- callbackSelector
  - nil 此次调用不需要 Callback。
  - 用于回调返回对应的参数 alias, tags。并返回对应的状态码：0为成功，其他返回码请参考错误码定义。
  - 回调函数请参考SDK 实现。
- theTarget
- 参数值为实现了callbackSelector的实例对象。
- nil 此次调用不需要 Callback。

```
- (void)tagsAliasCallback:(int)iResCode tags:(NSSet*)tags alias:(NSString*)alias {
            NSLog(@"rescode: %d, \ntags: %@, \nalias: %@\n", iResCode, tags , alias);}
```

Method - setAlias

调用此 API 来设置别名，支持回调函数。

该方法是 setTagsWithAlias (with Callback) 的简化版本，用于只变更别名的情况。

支持的版本

开始支持的版本：1.4.0

接口定义

```
+ (void)setAlias:(NSString *)alias callbackSelector:(SEL)cbSelector object:(id)theTarg
et;
```

参数说明

- alias
  - 空字符串 （@""）表示取消之前的设置。
  - 每次调用设置有效的别名，覆盖之前的设置。
  - 有效的别名组成：字母（区分大小写）、数字、下划线、汉字，特殊字符(v2.1.9支持)@!#$&*+=.|。
  - 限制：alias 命名长度限制为 40 字节。（判断长度需采用UTF-8编码）

- callbackSelector
  - nil 此次调用不需要 Callback。
  - 用于回调返回对应的参数 alias, tags。并返回对应的状态码：0为成功，其他返回码请参考错误码定义。
  - 回调函数请参考SDK 实现。
- theTarget
  - 参数值为实现了callbackSelector的实例对象。
  - nil 此次调用不需要 Callback。

```
    - (void)tagsAliasCallback:(int)iResCode tags:(NSSet*)tags alias:(NSString*)alias {
            NSLog(@"rescode: %d, \ntags: %@, \nalias: %@\n", iResCode, tags , alias)
}
```

Method - filterValidTags

用于过滤出正确可用的 tags。

如果总数量超出最大限制则返回最大数量的靠前的可用tags。

> 使用建议
> 设置 tags 时，如果其中一个 tag 无效，则整个设置过程失败。
> 如果 App 的 tags 会在运行过程中动态设置，并且存在对 JPush SDK tag 规定的无效字符，则有可能一个 tag 无效导致这次调用里所有的 tags 更新失败。
> 这时你可以调用本方法 filterValidTags 来过滤掉无效的 tags，得到有效的 tags，再调用 JPush SDK 的 set tags / alias 方法。

支持的版本

开始支持的版本：1.4.0

接口定义

```
+ (NSSet*)filterValidTags:(NSSet*)tags;
```

参数说明

- tags
  - 原 tag 集合。

接口返回

有效的 tag 集合。