MagicWindowSDK

使用文档（iOS）

——魔窗位

目录

一　集成准备

1.1 获取魔窗 AppKey

登录魔窗后台管理（http://mgnt.magicwindow.cn/），按照步骤提示注册应用，并新建多个魔窗位，可获得 AppKey 和魔窗位 Key。

1.2 导入 SDK

导入 SDK 有以下两种方法，第一种使用 Cocoapods 安装 SDK，第二种直接下载 SDK 并集成，选择其中一种即可。

注意: SDK 中包含了微信分享 SDK，如果您的工程里面也同样包含了微信分享 SDK，请将重复的删除，保留最新版的微信分享 SDK 即可，不会影响 SDK 的正常使用。

1.2.1 使用 Cocoapods 安装 SDK

使用 Cocoapods 可以方便的统一管理第三方库：https://cocoapods.org

安装 Cocoapods，并在项目根目录下创建 Podfile 文件，在 Podfile 文件中添加如下内容：

```
pod 'MagicWindowSDK'
//如果使用 bitcode，使用 pod 'MagicWindowSDKBitcode'
```

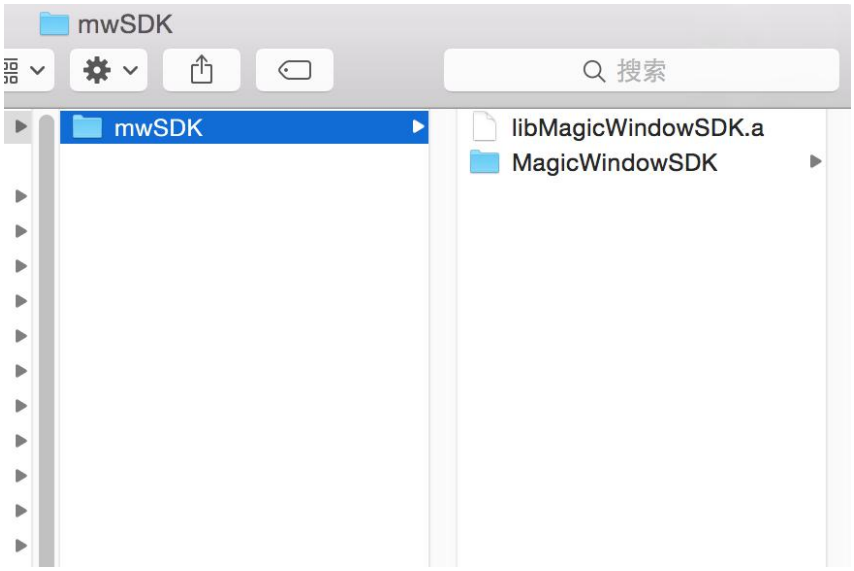
在 terminal 下运行命令如下：

```
pod install
```

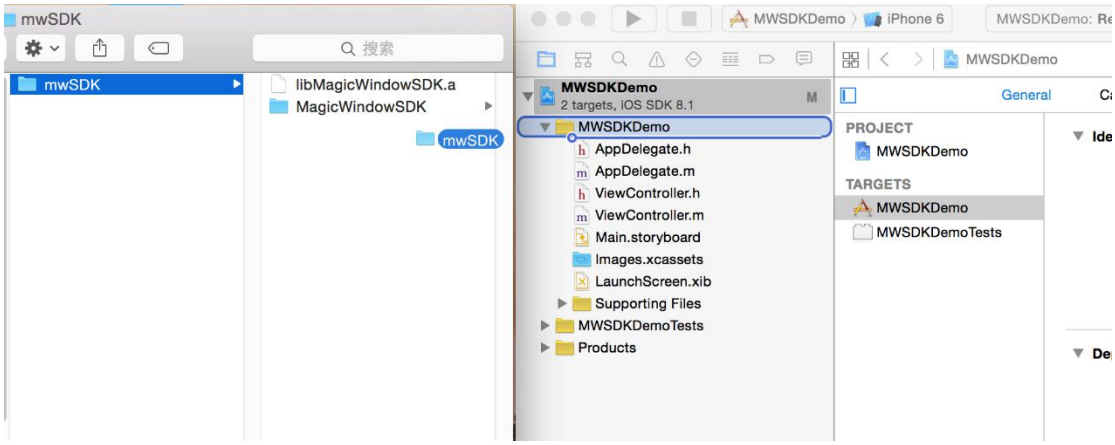
1.2.2 下载 SDK 并集成

（1） 下载 SDK 并集成

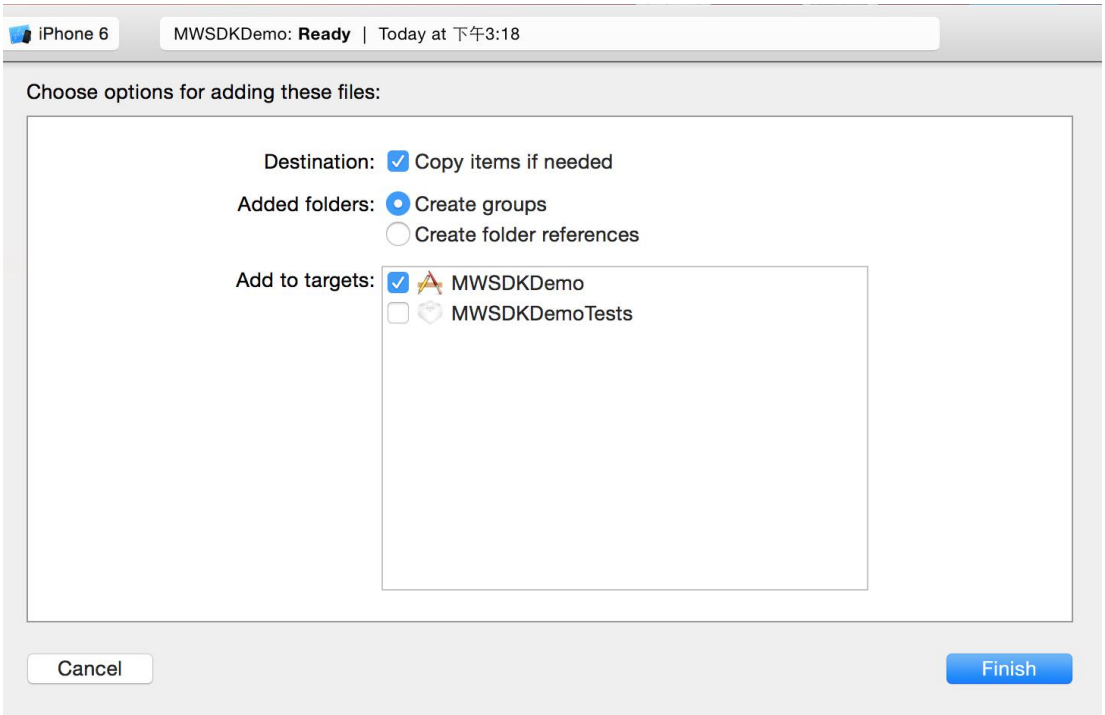
下载并解压最新版本 SDK 压缩包，解压后如下图：



将下载的 SDK 文件解压，拖动里面的 mwSDK 文件夹到工程中，如下图：



拖到工程中后，弹出以下对话框，勾选“**Copy items into destination group's folder(if needed)**”，并点击“**Finish**”按钮，如图：



注意：请务必在上述步骤中选择“**Create groups for any added folders**”单选按钮组。如果您选择“Create folder references for any added folders”，一个蓝色的文件夹引用将被添加到项目并且将无法找到它的资源。

（2） 添加依赖库

如果使用了 Cocoapods 集成的 SDK，可以忽略此步骤

AdSupport.framework

CoreTelephony.framework

CoreGraphics.framework

CoreFoundation.framework

SystemConfiguration.framework

CoreLocation.framework

CFNetwork.framework

Security.framework

WebKit.framework

ImageIO.framework

libz.tbd

libsqlite3.0.tbd

libc++.tbd

（3） 设置 Other Linker Flags

如果使用的是最新版本的微信分享 SDK，需要在你的工程文件中选择 Build Setting，在"Other Linker Flags"中加入"-Objc -all_load"，详情见微信官方文档：https://open.weixin.qq.com/cgi-bin/showdocument?action=dir_list&t=resource/res_list&verify=1&id=open1419319164&token=&lang=zh_CN

1.3 初始化 SDK

在 AppDelegate 中，增加头文件的引用

```
#import "MWApi.h"
```

在- (BOOL)application: didFinishLaunchingWithOptions:方法中调用 registerApp 方法来初始化 SDK，如下图所示：

```
#import "AppDelegate.h"
#import "MWApi.h"
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
//如果您创建应用时使用 storyBoard 可以省略此步骤
self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
self.window.rootViewController = viewController;
[self.window makeKeyAndVisible];

//初始化 SDK，必写
 [MWApi registerApp:AppKey];

return YES;
}
```

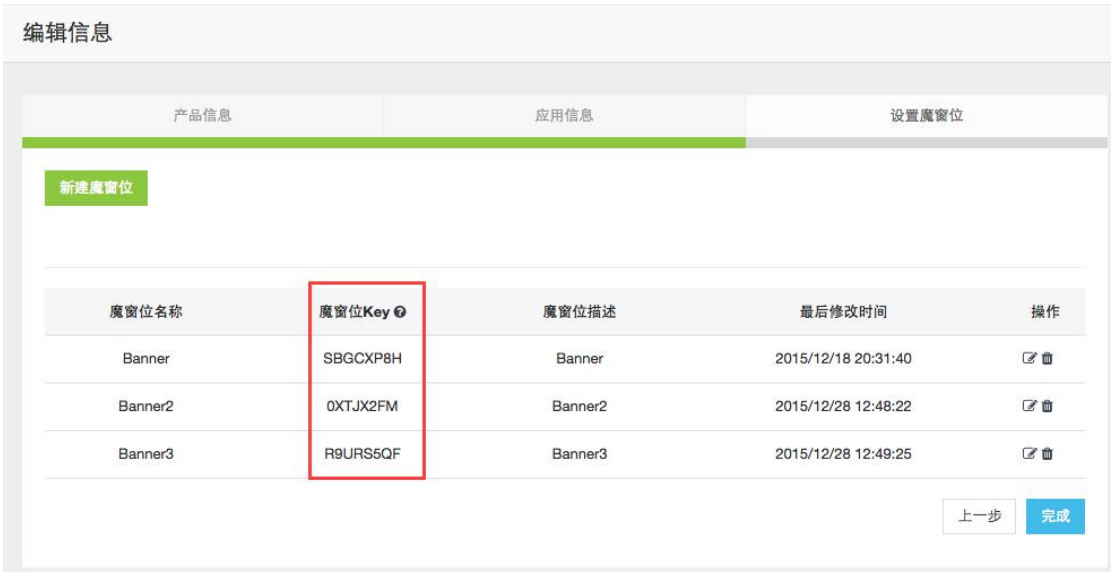
注意：必须设置 rootViewController，否则会导致无法弹出活动界面。如果您创建应用时使用 storyBoard 可以省略此步骤，系统会自动设置 rootViewController。

二　魔窗位的设置

2.1 添加魔窗位

魔窗位（动态位置管理）是魔窗的核心功能，所有的活动和 mLink 都需要基于魔窗位展开。魔窗位可以是 View，也可以是 ImageView，可以放在 App 的任意位置，也有用户把魔窗位叫做“任意门”。

魔窗位 Key 是魔窗位的唯一标识，在魔窗后台设置，如图：



注意：

a. 当返回 failure 参数 MWCampaignConfig 为空时，代表该活动已过期或者该活动不存在，请开发者自行处理相应的活动窗体（比如隐藏掉，或者显示别的内容）。

b. 开发者确保显示魔窗位视图后，调用 trackImpressionWithKey 这个 API 通知 SDK 魔窗位视图已成功展示

发送参数：

key：魔窗位 key

TargetView：展示魔窗位上投放活动的 view

TargetController：展示魔窗位上投放活动的 UIViewController

接收参数

MWCampaignConfig 对象，里面所有的数据和该活动对应，请在魔窗后台设置

success : callback 当成功获取到该魔窗位上活动的时候会调用这个回调

failure : callback 当获取到该魔窗位上活动失败的时候会调用这个回调

tap : callback 当点击该魔窗位上活动的时候会调用这个回调，return YES 允许跳转，NO 不允许跳转

mLinkHandler : callback 当活动类型为 mlink 的时候，点击的该活动的时候，会调用这个回调，return mlink 需要的相关参数

mLinkLandingPageHandler callback 当活动类型为 mlink landing page 的时候，点击的该活动的时候，会调用这个回调，return mlink landing page 需要的相关参数

```
+ (void)configAdViewWithKey:(nonnull NSString *)key withTargetView:(nonnull UIView *)view withTargetViewController:(nullable UIViewController *)controller success:(CallbackWithCampaignSuccess)success
failure:(CallbackWithCampaignFailure)failure
tap:(nullable CallbackWithTapCampaign)tap
mLinkHandler:(nullable CallbackWithMLinkCampaign)mLinkHandler
mLinkLandingPageHandler:(nullable CallbackWithMLinkLandingPage)landingPageHandler;
```

示例：

```
    [MWApi configAdViewWithKey:key withTargetView:imgView withTargetViewController:self WithCallBackMLinkKey: @"mLink key" success:^(NSString * _Nonnull key, UIView * _Nonnull view, MWCampaignConfig * _Nonnull campaignConfig) {
    //获取到相应活动资源的时候，请自行渲染相应的 view
    if (campaignConfig != nil)
    {
        [imgView sd_setImageWithURL:[NSURL URLWithString:campaignConfig.imageUrl] placeholderImage:nil];
    }
} failure:^(NSString * _Nonnull key, UIView * _Nonnull view, NSString * _Nullable errorMessage) {
        //获取相应活动失败的时候，请将 view 隐藏，否则会出现空白页面
        imgView.hidden = YES;
```

```
        NSLog(@"failed key = %@,view = %@,error = %@",key,view,errorMessage);
} tap:^BOOL(NSString * _Nonnull key, UIView * _Nonnull view) {
        //当用户点击了魔窗位，在跳转到活动页之前，app 可以做些处理，比如登录等等事件
        //return YES:跳转到活动页 ，NO：停止跳转
          if (!_isLogin)
          {
                [self loginBtn:nil];
          }
         return _isLogin;
} mLinkHandler:^NSDictionary * _Nullable(NSString * _Nonnull key, UIView * _Nonnull) {
          //return mlink 所需的相关参数
          return @{@"cityId":@"123",@"categoryId":@"abc"};
} mLinkLandingPageHandler:^NSDictionary * _Nullable(NSString * _Nonnull key, UIView * _Nonnull view) {
           //return mlink landing page 所需的相关参数
           return @{@"cityId":@"123",@"categoryId":@"abc"};
    }
];
```

2.2 魔窗位视图的曝光统计

为魔窗位上广告和活动提供单独的曝光接口用于统计

```
/**
 *  发送展现日志
 *  @param key 魔窗位 key
 */
+ (void)trackImpressionWithKey:(nonnull NSString *)key;
```

注意：确定视图显示在 window 上之后再调用 trackImpressionWithKey，不要太早调用，在 tableview 或 scrollview 中使用时尤其要注意。

2.3 信息流广告

（1） 导入头文件，设置代理

```
#import "MWAdNative.h"
#import "MWAdNativeView.h"
#import "MWAdNativeAdObject.h"
@interface ViewController : UIViewController<MWAdNativeDelegate>
@property (nonatomic, strong) MWAdNative *native;
```

（2） 请求广告，设置魔窗位 key，用 MWAdNative 请求广告

```
- (void)loadAd
{
    if (self.native == nil)
    {
        self.native = [[MWAdNative alloc] initWithMWKey:@"魔窗位 key"
delegate:self];
    }
    //请求原生广告
    [self.native loadNativeAds];
}
```

（3） nativeAdsSuccessLoad 回调函数表明请求广告成功

(NSArray *)nativeAds 参数表示成功返回的广告数组。

MWAdNativeAdObject.h 中可获取的属性见下表，可以通过以下属性返回广告字段。

```
//魔窗位 key
@property (copy, nonatomic) NSString *mwKey;
//广告顺序
@property (assign, nonatomic) NSInteger adIndex;
```

```
//标题 text
@property (copy, nonatomic)   NSString *title;
//描述 text
@property (copy, nonatomic)   NSString *text;
//小图 url
@property (copy, nonatomic) NSString *iconImageURLString;
//大图 url
@property (copy, nonatomic) NSString *mainImageURLString;
//多图信息流的 image url array
@property (strong, nonatomic) NSArray *morepics;
//品牌名称，若广告返回中无品牌名称则为空
@property (copy, nonatomic)   NSString *brandName;
//对返回的广告单元，需先判断 MWADMaterialType 再决定使用何种渲染组件
@property MWADMaterialType materialType;
```

（4） 创建信息流广告视图

魔窗 SDK 中提供了信息流广告视图样式，可以直接使用，也可以自定义信息流广告视图样式。

a. 使用魔窗的信息流广告视图样式

```
MWAdNativeView *adview = [[MWAdNativeView alloc]
initAdViewWithFrame:CGRectMake(0, 0, CGRectGetWidth(self.view.frame), 0)
Object:adObject];
```

MWAdNativeView 的高度会自适应，可以通过 adview.frame.size.height 来获取实际的 MWAdNativeView 的视图高度

b. 自定义信息流广告视图

```
UILabel *brandLabel = [[UILabel alloc] initWithFrame:CGRectMake(85, 15, 212,
20)];
brandLabel.font = [UIFont fontWithName:brandLabel.font.familyName
size:15];
```

```
    UILabel *titleLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 10, 200,
10)];
    titleLabel.textColor = [UIColor blueColor];
    titleLabel.font = [UIFont fontWithName:titleLabel.font.familyName size:12];
    UIImageView *iconImageView = [[UIImageView alloc]
initWithFrame:CGRectMake(20, 5, 60, 60)];
    UIImageView *mainImageView = [[UIImageView alloc]
initWithFrame:CGRectMake(50, 80, 250, 250)];
    UILabel *adLogoLabel = [[UILabel alloc] initWithFrame:CGRectMake(286, 318,
24, 12)];
    UIImageView *imgview1 = [[UIImageView alloc] initWithFrame:CGRectMake(20,
80, 100, 100)];
    UIImageView *imgview2 = [[UIImageView alloc]
initWithFrame:CGRectMake(140, 80, 100, 100)];
    UIImageView *imgview3 = [[UIImageView alloc]
initWithFrame:CGRectMake(260, 80, 100, 100)];
    MWAdNativeView *nativeAdView = [[MWAdNativeView alloc]
initWithFrame:CGRectMake(0, 0, CGRectGetWidth(self.view.frame), 200)
brandName:brandLabel title:titleLabel icon:iconImageView
mainImage:mainImageView moreImages:@[imgview1,imgview2,imgview3]];
    nativeAdView.adLogoLabel = adLogoLabel;
    [nativeAdView addSubview:adLogoLabel];
    nativeAdView.backgroundColor = [UIColor whiteColor];
```

（5） loadAndDisplayNativeAdWithObject: completion:加载和显示广告视图。

开发者确保显示视图后 ,调用 trackImpression 通知 SDK 广告视图已成功展示

```
- (void)nativeAdsSuccessLoad:(NSArray *)nativeAds
{
     [nativeAds enumerateObjectsUsingBlock:^(id  _Nonnull obj, NSUInteger idx,
BOOL * _Nonnull stop) {
    //取得每一条广告 object
        MWAdNativeAdObject *object = obj;
```

```
    //创建每一条的广告 view
        MWAdNativeView *adview = [self newAdViewWithFrame:CGRectMake(0, 0,
CGRectGetWidth(self.view.frame), 300) Object:object];
        [self.adviewArr addObject:adview];
        [adview loadAndDisplayNativeAdWithObject:object
completion:^(NSString *error) {
            if (error == nil)
            {
                [self.tableView reloadData];
            //确定视图显示在 window 上之后再调用 trackImpression，不要太早调用
            //如果是在 tableView 上，可以在 tableViewCell 展现时调用 trackImpression
                [adview trackImpression];
            }
        }];
    }];
}
```

（6） nativeAdsFailLoad:表明请求广告失败

```
- (void)nativeAdsFailLoad:(MWADFailReason) reason
{
    NSLog(@"nativeAdsFailLoad,reason = %d",reason);
}
```

（7） nativeAdClicked:是广告点击函数

```
//广告被点击，打开后续详情页面
- (void)nativeAdClicked:(MWAdNativeView*)nativeAdView
{
    NSLog(@"nativeAdView clicked");
}
```

（8） didDismissLandingPage:是广告详情页被关闭

```
-(void)didDismissLandingPage:(MWAdNativeView *)nativeAdView
{
    NSLog(@"didDismissLandingPage");
}
```

2.4 魔窗位分享

SDK 为投放到魔窗位上的活动提供了分享功能，与 App 的分享功能可以并存。

2.4.1 微信分享

（1） 在微信开放平台申请并取得应用的 AppID

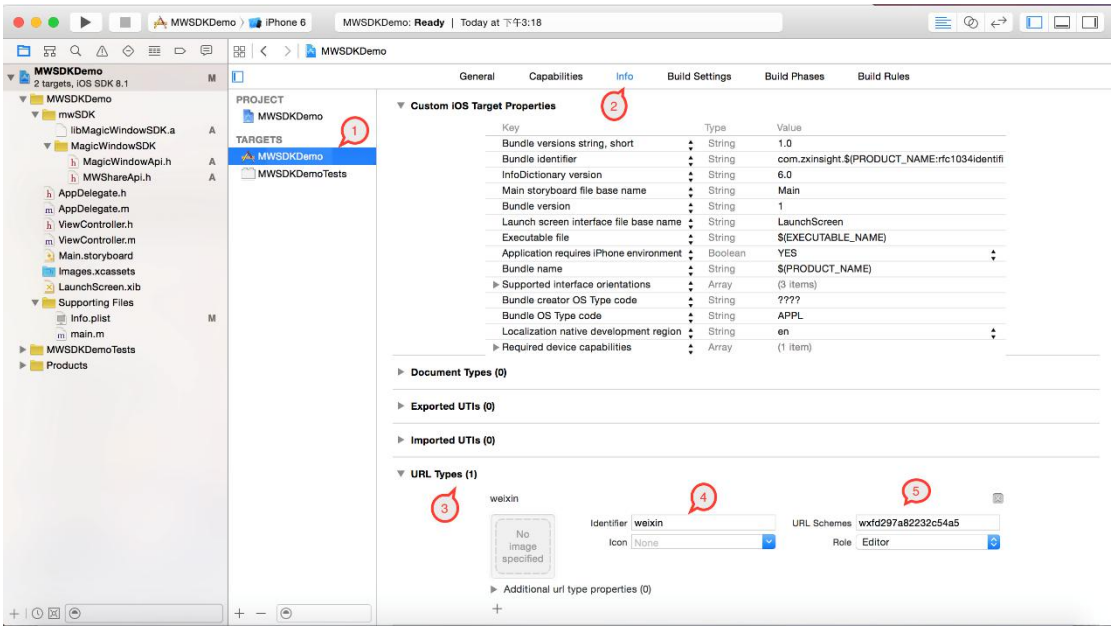
登录微信开放平台（http://open.weixin.qq.com）注册应用并取得应用的 AppID。

（2） 在魔窗管理后台配置应用的微信分享 ID

如图：

（3） 在 App 中配置微信分享 AppID

在 URL Types 中添加微信 AppID。操作步骤如图所示：



在 iOS9 中，需要在 info.plist 中将微信的 URL scheme 列为白名单，否则影响分享功能的使用，配置如下所示：

```
<key>LSApplicationQueriesSchemes</key>
<array>
    <string>weixin</string>
</array>
```

2.5 判读魔窗位上是否有内容

（1） 判断单个魔窗位上是否有内容

发送参数：魔窗位 key

返回参数：bool yes:该魔窗位上有内容；no：该魔窗位上没有内容

```
+(BOOL)isActiveOfmwKey:(NSString  *)mwkey;
```

（2） 批量判断魔窗位上是否有内容

发送参数：魔窗位 keys

返回参数：有内容的魔窗位 keys

```
+(NSArray  *)mwkeysWithActiveCampign:(NSArray  *)mwKeys;
```

三　高级设置

3.1 魔窗位相关通知

3.1.1 魔窗位上内容更新通知

当内容有更新的时候（包括新增内容，关闭内容，更新内容信息等），SDK 会发送 MWUpdateCampaignNotification 通知，当监听到这个通知的时候，开发者需要更新魔窗位上投放的活动展示信息。

目前 SDK 不会实时的发送 MWUpdateCampaignNotification 通知 ,发送通知机制如下：

（1） 启动 App

（2） App 退到后台超过 5 分钟，再次在前台打开 App

NotificationName：MWUpdateCampaignNotification

```
注册通知
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateCampaign) name:MWUpdateCampaignNotification object:nil];
移除通知
[[NSNotificationCenter defaultCenter] removeObserver:self name:MWUpdateCampaignNotification object:nil];
```

3.1.2 内容页面打开关闭通知

当用户点击魔窗位的时候，会打开内容详情页面，即 webview。

判断是否发送 webview 的相关通知

```
+ (void)setWebViewNotificationEnable:(BOOL)enable;
```

发送参数：enable，YES：打开，NO:关闭，默认为 NO

返回参数：无

当状态为 YES 的时候，将收到以下两个通知

（1） 内容详情页面即将打开通知

NotificationName：MWWebViewWillAppearNotification

```
注册通知
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(mwW
ebViewWillAppear:) name:MWWebViewWillAppearNotification object:nil];
- (void)mwWebViewWillAppear:(NSNotification*)notification
{
        //获取相应的魔窗位 key，根据魔窗位 key 进行相应的操作
    NSDictionary *dic = [notification userInfo];
    NSString *key = [dic objectForKey:@"mwkey"];
}
移除通知
[[NSNotificationCenter defaultCenter] removeObserver:self name: MWWebView
WillAppearNotification object:nil];
```

（2） 内容详情页面关闭通知

NotificationName：MWWebViewDidDisappearNotification

```
//注册通知
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(mwW
ebViewDidDisappear:) name: MWWebViewDidDisappearNotification object:nil];
- (void)mwWebViewDidDisappear:(NSNotification*)notification
{
        //获取相应的魔窗位 key，根据魔窗位 key 进行相应的操作
    NSDictionary *dic = [notification userInfo];
    NSString *key = [dic objectForKey:@"mwkey"];
}
移除通知
[[NSNotificationCenter defaultCenter] removeObserver:self name: MWWebView
DidDisappearNotification object:nil];
```

3.2 自定义魔窗位详情页导航条

是否自定义导航条，当为 NO 的时候，将使用魔窗自带的样式

```
+ (void)setWebViewBarEditEnable:(BOOL)enable;
```

发送参数：enable，YES：打开，NO:关闭，默认为 NO

返回参数：无

当状态为 YES 的时候，可以自定义导航条，详情见 MWApiObject.h 文件

3.2.1 按钮样式自定义

```
CGRect mwframe;                    //set frame
NSString *title;                   //set button title
UIFont *titleFont;                 //set button title font
UIColor *titleColorNormal;         //set title color for UIControlStateNormal
UIColor *titleColorHighlighted;//set title color for UIControlStateHighlighted
UIImage *imageNormal;              //set image for UIControlStateNormal
UIImage *imageHighlighted;         //set image for UIControlStateHighlighted
UIImage *backgroundImageNormal;//set background image for UIControlState
Normal
UIImage *backgroundImageHighlighted;//set background image for UIControlSt
ateHighlighted
UIColor *mwTintColor;              //set tint color
UIEdgeInsets mwTitleEdgeInsets;//set title edgeInsets
UIEdgeInsets mwImageEdgeInsets;//set image edgeInsets
```

示例：

```
    左边返回按钮设置：
    [[MWBarLeftButton appearance] setMwframe:CGRectMake(0, 0, 60, 30)];
    [[MWBarLeftButton appearance] setImageNormal:[UIImage imageNamed:@"
back.png"]];
    [[MWBarLeftButton appearance] setMwImageEdgeInsets:UIEdgeInsetsMake(0,
 0, 0, 50)];
    右边分享按钮设置：
    [[MWBarRightButton appearance] setTitle:@"分享"];
    [[MWBarRightButton appearance] setTitleFont:[UIFont systemFontOfSize:1
3]];
    [[MWBarRightButton appearance] setTitleColorNormal:[UIColor orangeColo
r]];
```

```
[[MWBarRightButton appearance] setMwframe:CGRectMake(0, 0, 60, 30)];
```

3.2.2 标题样式自定义

```
UIColor *mwTitleColor;                  //set title color
UIFont *mwTitleFont;                    //set title font
NSTextAlignment mwTextAlignment;        //set title alignment
```

四　基础指标统计

4.1 页面统计

统计某个页面的访问情况：

标记一次页面访问的开始

+ (void)pageviewStartWithName:(NSString *)name;

参数：name 页面名

```
- (void)viewWillAppear:(BOOL)animated
{
   [super viewWillAppear:animated];
   [MWApi pageviewStartWithName:@"homePage"];
}
```

标记一次页面访问的结束

+ (void)pageviewEndWithName:(NSString *)name;

参数：name 页面名

```
- (void)viewWillDisappear:(BOOL)animated
{
  [super viewWillDisappear:animated];
  [MWApi pageviewEndWithName:@"homePage"];
}
```

（注意：pageviewStartWithName 和 pageviewEndWithName 要成对匹配使用才能正常统计页面情况）

4.2 计数统计

统计指定行为被触发的次数

```
[MWApi setCustomEvent:@"eventId" attributes:@{@"type":@"pan",@"color":@"white",@"price":@"3.0"}];
```

（注意：eventId 需要先在魔窗后台管理上注册，才能参与正常的数据统计）

4.3 设置用户信息

配置用户的基本信息

登录的时候，设置用户信息

+ (void)setUserPhone:(NSString *)userPhone;

+ (void)setUserProfile:(MWUserProfile *)user;

退出登录的时候，取消当前的用户信息

+ (void)cancelUserProfile;

（注意：设置 MWUserProfile 的时候，用户唯一标识 mwUserId 不能为空）

五　使用多渠道分析

如果您要对 APP 不同的发布渠道进行统计，不需要在魔窗后台创建多个 APP，只需要设置不同的渠道即可。如果您只有 App Store 一个分发渠道，则不再需要做设定，我们会默认标记为 App Store。

例如您在 91 助手发布，需要统计 91 渠道：

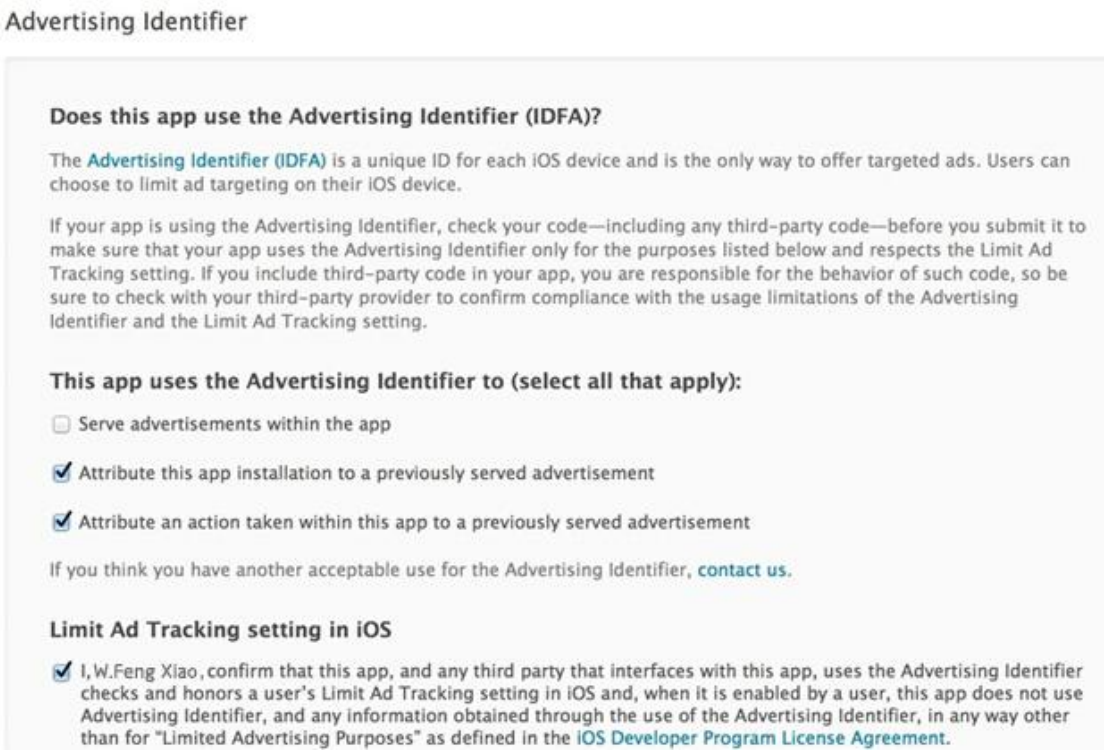
```
[MWApi setChannelId:@"channelId"];
```

（注意：channelId 需要先在魔窗后台管理上注册，才能参与正常的数据统计）

六　注意事项

6.1 如何防止 app 因获取 IDFA 被 App Store 拒绝

如果您的 app 使用魔窗 SDK 而未集成任何广告服务 ,但需要跟踪广告带来的激活行为，请按照下图填写 App Store 中的 IDFA 选项（勾选 2，3，4）:



（1） Serve advertisements within the app

服务 app 中的广告。如果你的 app 中集成了广告，你需要勾选这一项。

（2） Attribute this app installation to a previously served advertisement

跟踪广告带来的安装。

（3） Attribute an action taken within this app to a previously served advertisement

跟踪广告带来的用户的后续行为。

（4） Limit Ad Tracking setting in iOS

这一项下的内容其实就是对你的 app 使用 idfa 的目的做下确认，只要你选择了采集 idfa，那么这一项都是需要勾选的。

6.2 支持 ATS

苹果：2017 年 1 月 1 日后所有 iOS 应用必须启用 ATS。
SDK 已启用 ATS。

6.3 如何验证 SDK 已经对接成功

如何验证 SDK 已经对接成功，请参看
http://documentation.magicwindow.cn/?mlink-integration-validation

七 FAQ

FAQ：https://github.com/magicwindow/mw-sdk-faq