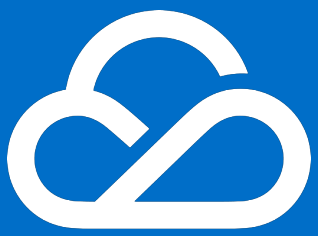
维纳斯

开发指南

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

iOS接入指南

1、Sdk目录结构说明

WnsSDK:包含WnsSDK.framework

WnsSDKDemo:示例工程,演示了如何使用WnsSDK

Doc:

WnsSDK使用说明:本文档

http端接入说明:从控制台配置到sdk的流程来演示http服务的接入

2、系统环境要求和限制

适用于iOS 5.0 及以上的系统版本

sdk接口字符类型参数限制长度为256字节

sdk发送数据限制长度为512K字节

sdk接受数据限制长度为512K字节

应用程序工程需要包含以下库(可以参考demo):

3、Sdk使用说明

3.1、名词解释

appID:为开发商在控制台申请的应用ID,

appVersion:是开发商应用程序的版本号, 如"1.0"等,

appChannel:是开发商用来区分发布的渠道的, 区分各种下载渠道，比如appstore、应用宝、百度手机助手。赋值如"appstore"。

uid:业务用户的唯一标识。
wid:wns为每个终端分配的唯一标识。
appVersion和appChannel是使用在上报统计和服务质量监控的。

终端使用Sdk主要包括下面几个步骤

3.2、Sdk初始化

调用接口initWithAppID完成系统初始化。

WnsSDK在初始化后, 会和Wns后台建立并保持一个长连接, 后续的请求会通过此连接发送
所以最好尽早初始化sdk, 最好在- (BOOL)application:(UIApplication)application didFinishLaunchingWithOptions:(NSDictionary)launchOptions里进行初始化。
示例

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

    gWnsSDK = [[WnsSDK alloc] initWithAppID:1000244
                              andAppVersion:@"1.0"
                              andAppChannel:@"appstore"];
    [gWnsSDK setStatusCallback:^(WnsSDKStatus status) {
        NSLog(@"WnsSDKStatus:%ld", (long)status);
        gWnsSDKStatus = status;

    }];

    return YES;
}
```

3.3、数据收发

初始化完Sdk后, 应用即可通过Sdk来发送数据. 根据发送的数据的类型的不同, 我们提供了两组接口:

发送二进制数据和发送http数据，开发商可以根据后端的协议类型(protobuf类的二进制数据 或 http协议)选择不同的接口。

3.3.1、Http数据收发

对于发送HTTP数据, 我们提供了两种接口方式，分别如下
1、兼容系统接口方案：
基于iOS系统的URL Loading System实现的, 只需引入WnsURLProtocol.h,
然后绑定sdk实例并向系统注册，代码如下

```
[WnsURLProtocol bindSDKInstance:gWnsSDK];
[NSURLProtocol registerClass:[WnsURLProtocol class]];

//注:使用AFNetworking的AFURLSessionManager时, 需要使用以下方式注册:

NSURLSessionConfiguration *configuration = [
NSURLSessionConfiguration defaultSessionConfiguration];
NSMutableArray *protocolsArray = [NSMutableArray
 arrayWithArray:configuration.protocolClasses];
[protocolsArray insertObject:[WnsURLProtocol class] atIndex:0];
configuration.protocolClasses = protocolsArray;
AFURLSessionManager *manager = [[AFURLSessionManager alloc] initWithSessionConfi
guration:configuration];
```

然后你在需要用Wns发送的请求用NSURLProtocol的方法来给这个请求打上标志[NSURLProtocol setProperty:@(YES) forKey:ShouldUseWns inRequest:req],
然后使用系统的NSURLConnection或者AFNetworking等第三方库来发送请求即可,
Wns会对打上标志的请求使用Wns的通道来发送.

注意, 如果使用NSURLSessionDataTask或者使用了该类的第三方库(比如AFNetworking),
还需要对request做以下设置:

```
// 当请求的方法是POST时, 如果使用NSURLSessionDataTask或者使用了改类的第三方库(比如AFNetworking)时,
```

```
// 自定义的NSURLProtocol获取到的NSURLRequest的HTTPBody为nil(系统bug, 对应的radar链接: rdar://1599
3891 )
// 所以, 在使用相关类发送前, 需要加上以下语句, 这样WnsURLProtocol才能获取到可用的HTTPBody
if (request.HTTPBody)
{
    [NSURLProtocol
 setProperty:request.HTTPBody forKey:WnsHTTPBody inRequest:request];
}
```

[注意]此模式下，sdk会自动将url设置为cmd，wns会统计每个cmd的成功率等信息，对应的，需要在控制台配置url域名对应的路由。路由配置请参考：控制台说明

2、Wns Sdk接口方案

调用接口sendHTTPRequest来收发Http数据。

开发商终端需要修改老的接口，替换为Wns的收发接口(该接口不支持http的301, 302跳转)

[注意]cmd必须是细化到接口，wns会统计每个cmd的成功率等信息，对应的，需要在控制台配置域名user.qzone.qq.com对应的路由。路由配置请参考：控制台说明

```
            NSString *cmd = @"user.qzone.qq.com/xunren";
            NSMutableURLRequest *req = [NSMutableURLRequest
 requestWithURL:[NSURL URLWithString:@"http://user.qzone.qq.com/xunren"]

                cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData
                                    timeoutInterval:120.0];
            [req setHTTPMethod:@"GET"];
            [req setHTTPShouldHandleCookies:NO];
            [req setValue:@"application/x-
www-form-urlencoded" forHTTPHeaderField:@"Content-Type"];
            [req setValue:@"gzip" forHTTPHeaderField:@"Content-Encoding"];

            [gWnsSDK se
```

```
ndHTTPRequest:req cmd:cmd timeout:30 completion:^(NSString *cmd,
NSURLResponse* response, NSData* data, NSError* wnsError, NSError
* wnsSubError) {
                NSLog(
@"cmd = %@, data = %@, wn
sError = %@, wnsSubError = %@", cmd, data, wnsError, wnsSubError);
                if (wnsError) {
                    self
.detailDescriptionLabel.text = [NSString stringWithFormat:
@"数据接收失败!\nerror = %@", wnsError];
                } else {
                    self.detailDescriptionLabel.text = @"";
                    self.responseWebView.hidden = NO;
                    [self
.responseWebView loadData:data MIMEType:[response MIMEType] textEncodingName:[re
sponse textEncodingName] baseURL:[response URL]];
                }
            }];
```

3.3.2、二进制数据收发

调用接口sendRequest来收发二进制数据。

发送二进制数据的接口和发送http的比较类似，开发商终端需要修改原来的代码，将收发接口替换为Wns的Sdk

```
/*! @brief 发送请求(TCP协议接口)。
 *
 * @param data 第三方应用应用层数据
 * @param cmd 第三方应用应用层请求命令字
 * @param timeout 请求超时时间
 * @param completion 回调Block，参数data表示服务器应答数据，参数bizError表示第三方业务错误，参数wnsError表示Wns
内部错误，wnsSubError表示Wns的子类错误
 * @return 成功返回请求序列号，失败返回-1。
```

```
 */
- (long)sendRequest:(NSData *)data
                cmd:(NSString *)cmd
            timeout:(unsigned int)timeout
         completion:(void(^)(NSString *cmd, NSData *data, NSError
 *bizError, NSError *wnsError, NSError *wnsSubError))completion;

//□□
NSData *data = [NSData dataWithBytes:"abcdefg" length:7];
            [gWnsSDK sendRequest:data
cmd:@"wnsdemo/transfer" timeout:30 completion:^(NSString *cmd, NSData
 *data, NSError *bizError, NSError *wnsError, NSError* wnsSubError) {
                NSLog(
@"cmd = %@
, data = %@, bizErro
r = %@, wnsError = %@, wnsSubError = %@"
, cmd, data, bizError, wnsError, wnsSubError);
                if (bizError || wnsError) {
                    self
.detailDescriptionLabel.text = [NSString stringWithFormat:
@"□□□□□□□\nerror = %@", bizError ?: wnsError];
                } else {
                    self
.detailDescriptionLabel.text = [NSString stringWithFormat:
@"□□□□□□□cmd = %@□ data length = %lu", cmd, (unsigned long)[data length]];
                }
            }];
        }
```

[注意]cmd必须是细化到接口，wns会统计每个cmd的成功率等信息，对应的，需要在控制台配置模块wnsdemo对应的路由。路由配置请参考：控制台说明

3.4、用户绑定和接收push

3.4.1、用户绑定

Wns会对每一个设备记录一个设备ID来进行标识, 可以通过后端的API对某一个设备进行消息推送(PUSH).
但是应用本身可能也有自己的用户id(后面简称uid), 如果需要对某一个uid进行消息推送时,
客户端就得在用户登录和注销时调用绑定/注销的相应方法. 调用后, Wns后台会记录两者的对应关系,
此后就能对该uid进行消息推送

```
/*! @brief
 * 第三方应用层可选调用接口。如果第三方应用后台需要通过WNS服务发送针对指定用户的数据，
 * 则第三方应用终端，在用户登录成功后，需要绑定第三方用户ID到WNS服务。
 * 对新uid进行绑定前， 需要使用unbind对旧bid进行解绑
 *
 * @param uid 第三方应用的用户唯一标识
 * @param completion 回调Block
 */
- (void)bind:(NSString *)uid completion:(void(^)(NSError *error))completion;


/*! @brief 注销绑定。应用的用户注销时请调用该接口.
 *
 * @param uid 第三方应用的用户唯一标识
 * @param completion 回调Block
 */
- (void)unbind:(NSString *)uid completion:(void(^)(NSError *error))completion;
```

3.4.2、接收Push

```
/*! @brief 设置Wns Push的数据接收block。
 *
 * @param completion 回调Block，参数cmd标识服务推送数据命令字，参数data标识服务器推送数据，参数error标识错误信息
 */
- (void)setPushDataReceiveBlock:(void(^)(NSString *cmd, NSData *data,
NSError *error))completion;
```

```
/*! @brief 向服务器注册苹果的推送服务所使用的devicetoken
 *
 * @param deviceToken 用户设备Tokon°
 * @param completion 回调Block, error为nil时表示注册成功
 */
- (void)registerRemoteNotification:(NSString
 *)deviceToken completion:(void(^)(NSError *error))completion;
```

3.5、调试类接口

在接入Wns的过程中, 当遇上某些问题需要和Wns侧一起定位问题时, 可以通过以下接口设置客户端连接到Wns的测试环境，Wns的测试环境也可以连接到开发商的测试服务器环境中，这样便于快速的定位到具体的问题。要使用测试环境，开发商必须按下面来要求来设置

3.5.1、设置调试环境接入IP

WNS提供调试环境给开发商使用，服务端使用控制台可以配置访问。客户端需要指定调试环境的接入IP。
开发商必须在Wns控制台配置好开发商测试环境服务器路由信息。
在接入Wns的过程中, 当遇上某些问题需要和Wns的工作人员一起调试时,
可以通过以下接口设置客户端连接的Wns后台的IP, 方便查找问题。

```
/*! @brief 调试接口: 设置调试服务器地址。
 *
 * @param ip 为nil或@"wns.qq.com"时切换回非调试模式
 * @param port 允许为0，即默认80/443/8080/14000轮询
 *
 * @note 会触发重建链接
 */
- (void)setDebugIP:(NSString *)ip port:(unsigned short)port;
```

3.5.2、关键日志

```
/*! @brief 获取wns的关键的日志信息,用于开发中查问题
 *
 */
- (NSString *)keyLog;
```

另外, 如果出现Wns连接或者收发出错的问题时, 可以通过Sdk接口keyLog拿到提示信息并打印出来,
提供给相关工作人员查看, 以方便查找问题。

3.5.3、Wns快速验证模式

对于一些还在犹豫是否该使用Wns的业务, 我们提供了一个自动测试的模式.
在该模式下, sdk会自动地每隔一段时间就发送一个测试的数据包到后台.
你可以到控制台的监控页面看到对应的成功率和延时等情况, 作为你们是否选用Wns的参考.
使用方式:初始化后sdk后,调用下面接口即可.

```
/*! @brief 设置自动测试模式,这种模式下,sdk自身会定时发送测试数据包到后台,您可以从监控报表查看相关统计数据
 *
 * @param isEnable 是否打开自动测试模式
 *
 */
- (void)setAutoTestMode:(BOOL)isEnable;
```

Wns提供快速验证模式，开发商可以先集成Wns的Sdk，但是通信接口暂时不切换到Wns通道，该模式下，Wns Sdk会自动发一些探测包到Wns的接入服务器（探测包不会转发到开发商服务器）。这样开发商可以在控制台上看到探测包的统计数据。可以准确的评估出实际接入Wns后的服务质量。如果效果明显，开发商再后续版本才正式将服务切换到Wns通道。

使用Wns快速验证模式，开发商只需要下面的几个步骤
1、Wns控制台注册App和签名信息
2、终端集成Wns Sdk
3、终端在初始化阶段调用Sdk接口initWithAppID和setAutoTestMode

Android接入指南

Sdk目录结构说明

WnsSDKDemo:示例工程,演示了如何使用WnsSDK

获取应用签名的md5工具:用来获取打了正式签名的正式包的md5

Doc:

WnsSDK使用说明:本文档

系统环境要求和限制

适用于Android 2.3.3及以上的系统版本。

sdk接口字符类型参数限制长度为256字节

sdk发送数据限制长度为512K字节

sdk接受数据限制长度为512K字节

名词解释

appid: wns 为接入 app 分配的标识。

appVersion: app 版本号。

channelId: app 渠道号，与 appVersion 一起在 wns 监控系统中提供统计信息，

来区分各种下载渠道，比如应用宝,百度手机助手。

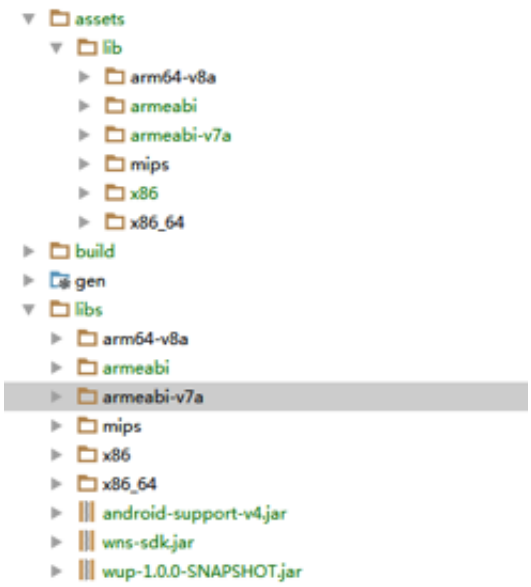
uid:业务用户的唯一标识。

wid:wns为每个终端分配的唯一标识

Sdk配置和初始化说明

引入Wns Sdk

将/libs 下的 assets 和 libs 目录分别复制到工程根目录对应的目录中，如下图



注意：需要将 jar 文件和对应的so 添加 到工程中。

这里的armeabi的so文件夹是必须添加，其它so文件夹可以按需添加，如果你的工程中有对应的so文件夹就必须加入相应文件夹的so。例如工程libs目录下已经有arm64-v8a目录，则需要添加到WNSsdk中相关的arm64-v8a文件夹下的64位系统的相关so，否则64位上的机器会崩溃。

配置 AndroidManifest.xml

wns_SDK 需要使用的系统权限如下图（可参考示例工程）：

注意：加到根目录 manifest 下面。

```
<!-- demo□□□□□□□□API□□□□□□□□□□□□□□□□□□□□sdk version -->
<uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="18" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

配置 WnsSdk服务

注册 Service：

```
<!-- service -->

<!-- WNS service,□□□□□□  :wns □□□□□□□□□□□□□□□□ -->
<service
        android:name="com.tencent.wns.service.WnsMain"
        android:exported="false"
        android:process=":wns">
    <intent-filter>
        <action android:name="com.tencent.wns.service.WnsMain"/>
    </intent-filter>
</service>

<!-- □□WNS□□□□□ -->
<receiver
        android:name="com.tencent.base.os.clock.AlarmClockReceiver"
        android:exported="false"
        android:process=":wns">
    <intent-filter>
        <action android:name="wns.heartbeat"/>
    </intent-filter>
</receiver>
```

注意:

(1) 进程名“:wns”为 wns 服务使用，请不要占用

(2) 以上部分位于 application 分支下。

初始化 APP 信息并启动服务

修改 MyBaseApplication，onCreate（ ）变为如下内容：

```
public class MyBaseApplication extends Application
{
    private final static String TAG = "BaseApplication";

    private WnsService wns = WnsClientFactory.getThirdPartyWnsService();;

    @Override
    public void onCreate()
    {
        super.onCreate();

        WnsAppInfo info = new WnsAppInfo()
            //[修改] 云平台上申请获取的APPID
             .setAppId(1000244)

//[修改] App的版本信息 – 建议填写，方便以后提供详细细分统计（这里的"1.0.0"代表版本
    号，业务可以按照自己规则填写）
             .setAppVersion("1.0.0")
             //[修改] APP的渠道信息 – 建议填写，方便以后提供详细细分统计。（这里的"yyb"代表应用宝，业务可以按照自己规则填写）
               .setChannelId("yyb");
         wns.initAndStartWns(this, info);
     }
}
```

同时在 AndroidManifes.xml 中配置实现：

```
<application android:name=".MyBaseApplication" ....
```

代码混淆

sdk 已经混淆，建议不要再次混淆。如果需要混淆，请务必在脚本中加入以下配置

```
-keep class com.tencent.wns.openssl.OpenSSLNative
{ private long pkey;
}
-keep class com.tencent.wns.network.ConnectionImpl  {*;
}
-keep class * implements com.qq.taf.jce.JceStruct {
}
-keep class com.tencent.wns.service.WnsMain
```

SDK数据收发接口

Http数据收发

使用HttpClient请求

HttpClent实例使用WnsService.getWnsHttpClient()来获取,然后使用HttpClient.execute(HttpUriRequest httpUriRequest)来发起http请求
注意：
1.发送和接受数据大小限制为512KB。
2.业务侧最好打印出 response.getFirstHeader(WnsService.KEY_HTTP_RESULT)中的数据
以便于bug定位

[注意]此模式下，sdk会自动将url设置为命令字，wns会统计每个命令字的成功率等信息，对应的，需要在控制台配置url域名对应的路由。路由配置请参考：http://www.qcloud.com/doc/product/276/控制台说明

如下图所示:

```
//[必须] 定义wns的引用，从而使用其内部方法
private final WnsService wns = WnsClientFactory.getThirdPartyWnsService();
//[注意] HTTP的收发包样例。 发送数据&接收数据大小限制为512KB
private void sendHttpReq() {
    Runnable run = new Runnable() {
        @Override
```

```
        public void run() {
          // [修改] 请求的url地址，完整地址。
          String url =
"http://user.qzone.qq.com/xunren?a=b";     //控制台上需要配置user.qzone.qq.com对应的路由
          //[必须] 用wns的接口替换系统HttpClient
            HttpClient client = wns.getWnsHttpClient();

//[两个超时总和的请求，建议60秒          client.getParams().setParameter(CoreConnectionPNames.
CONNECTION_TIMEOUT,30*1000);


   client.getParams().setParameter(CoreConnectionPNames.SO_TIMEOUT, 30*1000);
            HttpUriRequest request = new HttpGet(URI.create(url));
            try {
                //[必须] 同样的execute方法，发起http请求。
                HttpResponse response = client.execute(request);
                if
 (response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK)
                {
                    //[参考] 常见http回包解析方式，业务端可以保持自己代码不变
                    InputStream in = response.getEntity().getContent();
                    ByteArrayOutputStream out = new ByteArrayOutputStream();
                    byte[] buff = new byte[1024];
                    int len = -1;
                    while ((len = in.read(buff)) != -1) {
                        out.write(buff, 0, len);
                    }
                    in.close();

                     Log.d("wns","http code "
 + response.getStatusLine().toString());
                     Log.d("wns","http data " + out.toString().trim());
                }
                else
```

```
                {
                    Header header = response.getFirstHeader(WnsService.KEY_HTTP_
RESULT);
                    String wnsCode = "";
                    if (header != null)
                    {
                        wnsCode = header.getValue();
                    }
                    Log.d("wns","http code "
 + response.getStatusLine().toString() + ", wnscode " + wnsCode);
                }

            } catch (ClientProtocolException e) {
                e.printStackTrace();
                Log.e(TAG,"catch error:"+e.toString());
            } catch (IOException e) {
                e.printStackTrace();
                Log.e(TAG,"catch error:"+e.toString());
            }
        }
    };
    new Thread(run).start();
}
```

使用HttpUrlConnection请求

使用wns.getWnsHttpUrl()获取URL实例

注意：

1发送和接受数据大小限制为512KB。

[注意]此模式下，sdk会自动将url设置为命令字，wns会统计每个命令字的成功率等信息，对应的，需要在控制台配置url域名对应的路由。路由配置请参考：http://www.qcloud.com/doc/product/276/控制台说明

```
//[必须] 定义wns的引用，从而使用其内部方法
private final WnsService wns = WnsClientFactory.getThirdPartyWnsService();
//[注意] HTTP的收发包样例. 发送数据&接收数据大小限制为512KB
private void sendHttpUrlConnReq(final String url)
{
    Runnable run = new Runnable()
    {
        @Override
        public void run()
        {
            try
            {
                //1.构造URL，底层网络走wns通道,使用起来和URL是一样的。

  URL u = wns.getWnsHttpUrl(url);                    //需要在控制台上配置url对应域名的路由
                HttpURLConnection conn = (HttpURLConnection) u.openConnection();
                conn.addRequestProperty("User-Agent", "");

                //[两个超时总和的请求，建议60秒
                conn.setConnectTimeout(30*1000);
                conn.setReadTimeout(30*1000);

                //如果是post方法，需要设置一下post参数,可选
                //conn.setDoOutput(true);
                //conn.getOutputStream().write("a=10&b=99".getBytes());
                //获取页面内容
                int rspcode = conn.getResponseCode();
                Log.d(TAG, "rspcode = " + rspcode);
                if (HttpURLConnection.HTTP_OK == rspcode)
                {
                    InputStream in = conn.getInputStream();
                    if (in == null)
                    {
                        return;
```

```
                    }
                    byte[] buff = new byte[1024];
                    ByteArrayOutputStream out = new ByteArrayOutputStream();
                    int len = -1;
                    while ((len = in.read(buff)) != -1)
                    {
                        out.write(buff, 0, len);
                    }
                    in.close();
                    final String content = out.toString().trim();
                    Log.w(TAG,
content.substring(0, 100 > content.length() ? content.length() : 100));

        Log.w(TAG, conte
nt.substring(content.length() > 100 ? content.length() - 100
 : content.length()));

                }
            } catch (MalformedURLException e)
            {
                e.printStackTrace();
            } catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    };
    new Thread(run).start();
}
```

二进制数据收发

调用接口sendRequest来收发二进制数据。

发送二进制数据的接口和发送http的比较类似，开发商终端需要修改原来的代码，将收发接口替换为Wns的Sdk

1.发送和接受数据大小限制为512KB。

2.命令字禁止使用“wnscloud”作为前缀。

[注意]cmd必须是细化到接口，wns会统计每个cmd的成功率等信息，对应的，需要在控制台配置模块wnsdemo对应的路由。路由配置请参考：http://www.qcloud.com/doc/product/276/控制台说明

```
private final
 WnsService wns =
WnsClientFactory.getThirdPartyWnsService(); //[必须] 定义wns的引用，从而使用其内部方法



//[注意] 二进制的收发包样例. 发送数据&接收数据大小限制为512KB
private int sendReq()
{
    final String cmd = "wnsdemo/transfer";    //需要在控制台配置wnsdemo对应的路由
    final int timeout = 60 * 1000;
    final byte[] buff = "this is buff".getBytes();
    showLoading();
    int seqNo = wns.sendRequest(cmd, timeout, buff, new WnsTransferCallback()
    {

        @Override
        public void onTransferFinished(IWnsTransferResult re)
        {
            int mainErrCode=re.getWnsCode();
            int subErrCode=re.getWnsSubCode();
            int bizErrCode=re.getBizCode();
            String errMsg=re.getErrMsg();
            //mainErrCode,bizErrCode可用于提示用户，提示语需要业务自定义，subErrCode需要查问题时向wns提供
             Log.d("wns","send request result " + re + ", msg :" + errMsg);
             stopLoading();
        }
```

```
    });
    Log.d("wns","send request seqNo=" + seqNo);

    return seqNo;
}
```

PUSH接入

在 AndroidManifest.xml 中注册接收 push 的 service

```
<!-- □□ WNS push □□□ -->

<service
        android:name=
"com.example.cloudwns.push.MyPushService" android:exported="false">

        <intent-filter>

              <!-- □□ action □□□□□    wns.push.to.<package name> -->
                <action android:name="wns.push.to.com.example.cloudwns" />
         </intent-filter>

</service>
```

其中 com.example.cloudwns.push.MyPushService （类名可修改）是应用自定义的push 处 理 类 型 ， 继 承 自 WNS SDK 提 供 的 抽 象 类com.tencent.wns.ipc.AbstractPushService。

自定义处理 Push 的Service

假设类名是 com.example.cloudwns.push.MyPushService（应用可自定义名称），应用只需要实现 onPushReceived 这个方法即可。如下：

```
package com.example.cloudwns.push;


import android.util.Log;

import
 com.tencent.wns.client.data.PushData; import
 com.tencent.wns.ipc.AbstractPushService;


public class MyPushService extends AbstractPushService{

       public static final String TAG = "MyPushService";
       /**

        *子类实现的接受消息推送的方法<br>

         *<br>

        *这将在主线程执行，若要处理耗时操作，请异步。

        *

        *@param pushes 消息内容

        *@return 是否消费。暂时返回 true，保留字段。

        */

      @Override

       public boolean onPushReceived(PushData[] pushes)
       {
            long
```

```
begin = System.currentTimeMillis(); for (PushData pushData : pushes)
      {
          Log.i(TAG,"push data = " + pushData);
      }

       long end = System.currentTimeMillis();

       Log.i(TAG, "onPushReceived timecost = " + (end - begin)); return true;

     }

}
```

调试类接口

在接入Wns的过程中, 当遇上某些问题需要和Wns侧一起定位问题时, 可以通过以下接口设置客户端连接到Wns的测试环境，Wns的测试环境也可以连接到开发商的测试服务器环境中，这样便于快速的定位到具体的问题。
要使用测试环境，开发商必须按下面来要求来设置
1、终端调用Wns Sdk接口：setDebugIP，确保连接到Wns测试环境。
2、开发商必须在Wns控制台配置好，开发商测试环境服务器路由信息。

另外, 如果出现Wns连接或者收发出错的问题时, 可以通过Sdk接口keyLog拿到提示信息并打印出来,
提供给相关工作人员查看, 以方便查找问题。

Wns快速验证模式

Wns提供快速验证模式，开发商可以先集成Wns的Sdk，但是通信接口暂时不切换到Wns通道，该模式下，Wns Sdk会自动发一些探测包到Wns的接入服务器（探测包不会转发到开发商服务器）。这样开发商可以在控制台上看到探测包的统计数据。可以准确的评估出实际接入Wns后的服务质量。如果效果明显，开发商再后续版本才正式将服务切换到Wns通道。

使用Wns快速验证模式，开发商只需要下面的几个步骤
1、Wns控制台注册App和签名信息

2、终端集成Wns Sdk

3、终端在初始化阶段调用Sdk接口initWithAppID

常见问题

i) 问：为什么 WNS SDK 中有些 so 放在 assets 目录下，有些 so 放在 libs 目录下？

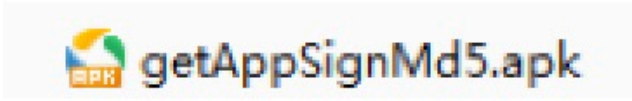
答：通过 System.loadLibrary 方法加载 so 都会去 libs 目录找相应 cpu 架构的 so，根据以往经验，一些低端机型无法加载到 so 导致 WNS 启动失败。

因此 WNS 会在 assets 目录 下也放了一份 so，当 loadLibraray 方法加载失败会尝试将 assets 目录下的 so 复制到 app 运行目录中，通过 System.load

方法加载 so，提高启动 WNS 的成功率。

i) 问：怎么样获取应用的签名

答：我们提供的 sdk zip 包中包含获取应用签名的 app 工具，安装到手机后，输入您的app包名即可获取到签名



服务端接入指南

Wns服务端部署架构

详细内容请参考新手入门->整体架构的说明。

WNS和开发商服务器的通信方式

WNS接入服务器和开发商服务器之间建立TCP长连接，目前只支持单发单收的模式，就是一个连接上，WNS发送数据到开发商服务器后，必须等开发商服务器返回响应包后，再发下一个请求。多个并发的请求就建立多个连接来并发收发数据。

WNS和开发商服务器的通信协议

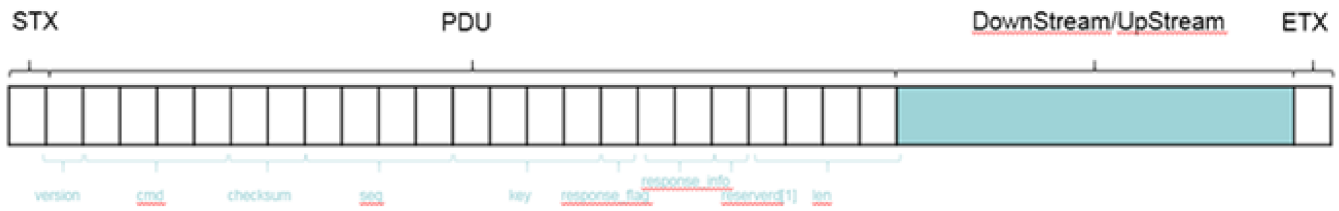
开发商接入WNS，开发商服务器需要对接WNS接入服务器的协议，目前主要有两种协议对接方式。
HTTP协议：开发商服务器不用任何改造就可以接入WNS，WNS接入服务器和开发商服务器之间的通信是标准的HTTP协议。
二进制协议：开发商服务器需要改造对接WNS接入服务器的协议，具体协议格式在下面详细介绍。

HTTP协议对接

开发商的web服务器不需要做任何改造，wns透传开发商自己的http数据包。为了开发商便于获取终端的信息，wns在http头上，会添加部分信息。具体内容如下
X-Wns-Qua : Wns Sdk的版本信息，开发商可以记录sdk版本信息，定位问题时可以便于确认具体的版本号。
X-Forwarded-For : 移动终端的IP地址，开发商可以根据终端IP做精确营销等服务。
X-Wns-DeviceInfo : 移动终端的设备信息，开发商可以根据设备信息做图片适配、差异化服务等。
X-Wns-Wid :
移动终端的唯一标识，可以根据wid做详细的问题定位，开发商也可以使用wid做push消息调用。
QVIA : 移动终端的IP，内容和X-Forwarded-For 相同。

二进制协议说明

维纳斯二进制协议主要包括协议开始标记、协议头，数据串、协议结束标记，一共四个部分，如下图所示



WNS接入服务器 转发到 开发商服务器协议： STX + pdu + UpStream + ETX

开发商服务器 返回到 WNS接入服务器协议： STX +pdu + DownStream + ETX

说明：

STX：协议开始标记位(1字节)

Pdu：WNS协议头(23字节)

UpStream： protobuf结构，客户端APP发送的buf在其中(变长字段)

DownStream：protobuf结构，开发商返回的buf在其中(变长字段)

ETX：协议结束标记位(1字节)

STX和ETX（单字节标记位）

STX = 0x04

ETX = 0x05

Pdu协议格式（二进制网络字节序）

```
struct pdu_protocol_header
{
uint8_t version;            //wns保留,开发商无需识别
uint32_t cmd;              // wns保留,开发商无需识别
uint16_t checksum;          // wns保留,开发商无需识别
uint32_t seq;              // wns保留,开发商无需识别
uint32_t key;              //填写0
uint8_t response_flag;     //填写0
uint16_t response_info;     //填写0
char reserved[1];           //预留字段
uint32_t len;                  //整个数据长度，包括STX、协议头、ETX等
 } ;
```

UpStream协议（protobuf结构）

```
message WnsAppUpstream
 {
     required uint32 seq      = 1;                  //序列号回包时对应
     required uint32 appid      = 2;                //业务的appid
     required uint64 wid      = 3;                  //wns id代表一个wns的链接可以发push用
     required string qua      = 4;                  //qua信息
     required string service_cmd = 5;          //sdk接口传入的命令字
     required string device_info      = 6;   //设备信息 imei/os/机型/分辨率
     optional WnsAppClientIpInfo ip_info     = 7;     //客户端IP信息
     required bytes busi_buff      = 8;            //sdk接口传入的业务buf
     optional string uid  = 9;                     //开发商sdk调用bind的用户id
     optional UserAgent user_agent = 10; //标识终端版本号、渠道号
 };


message WnsAppClientIpInfo
 {
     enum IpType{
         IPV4     = 1;
         IPV6     = 2;
     }//1表示IPv4 2表示IPv6
     optional IpType ip_type         = 1;
     optional int32 client_port     = 2;     //客户端端口
     optional string client_ip     = 3;
//如: IPV4:10.6.1.1 ipv6: fe80::2e0:81ff:feda:202d
 };


message UserAgent
 {
```

```
    required int32 platform         = 1;       //平台区分 0:ios 1:android
    optional string version         = 2;       //版本号
    optional string channel         = 3;        //渠道名
};
```

服务器收包逻辑简介

这里简单说明下开发商服务器如何对接Wns二进制协议，开发商服务器可以按下面思路来接收数据包

1、接收数据头：数据头长度是 STX+PDU的长度，一共是24个字节，多次收取数据直到大于等于24个字节。

2、分析协议头：根据pdu的协议，获取出len字段的长度，获取整个数据包的长度。

3、接收完整数据：根据pdu头解析出的整个数据包长度，多次收取数据，直到收完整个数据包的长度。

4、完整性检查：检查STX、ETX来简单判断数据合法性。

5、ProtoBuf解析：根据协议获取到WnsAppUpstream对应的buffer，用ProtoBuf对这段buffer做解码。

6、获取业务数据：根据解析出来的WnsAppUpstream结构，获取到业务自己的buffer数据(busi_buff字段)。

7、其他信息获取：根据WnsAppUpstream，还可以获取到终端的一些信息，比如终端IP(WnsAppClientIpInfo)、终端设备信息(device_info)、设备标识(wid)等。

DownStream协议（protobuf结构）

```
message WnsAppDownstream
{
    required uint32 seq  = 1;                  //和UpStream中seq保持一致
    required uint64 wid = 2;                  //和UpStream中wid保持一致
    required sint32 biz_code = 3;
//业务返回码 由业务设置。用于wns统计监控使用  0为处理正常，非0为异常
    required string service_cmd = 4;     //和UpStream中service_cmd保持一致
    required bytes busi_buff = 5;              //开发商返回给APP的数据buf
    optional string err_msg = 6;              //错误信息，biz_code非0的时候填写错误信息提示客户端
    optional string uid = 7;                  //  和UpStream中uid保持一致
};
```

服务器回包逻辑简介

服务器回包逻辑，只需要按二进制协议规范，将数据打包返回给Wns接入服务器即可。返回包中，几个关键数据必须正确。主要包括下面几个

1、序列号：WnsAppDownstream中seq字段，必须保证和WnsAppUpstream中seq字段相同的值。

2、设备标识：WnsAppDownstream中wid字段，必须保证和WnsAppUpstream中wid字段相同的值。

3、命令字：WnsAppDownstream中service_cmd字段，必须保证和WnsAppUpstream中service_cmd字段相同的值。

4、用户ID：WnsAppDownstream中uid字段，必须保证和WnsAppUpstream中uid字段相同的值。

接入Push指南

Push消息统一路径

消息相关的接口，统一的接口地址是http://wns.api.qcloud.com/api/
在统一地址后面加上不同的接口名字，实现响应的功能接口。

签名生成方法

sign=Base64(Hmac-sha1(plaintext, secretkey))
secretkey：摘要算法key，在腾讯云创建app时分配，
plaintext（原文）：“appid&timestamp”

OpenApi接口说明

发送消息接口

接口说明：发送在线消息通知到客户端
接口名：send_msg_new
请求方法：http post方法

请求参数:

参数名	类型	必选	说明
appid	int	是	第三方appid
secretid	string	是	第三方加密用的密钥id
sign	string	是	第三方签名
tm	int	是	时间戳，防请求重放
uid	string	是	用户唯一标识
wid	string	是	用户名下的某个设备标识，需要指定某台设备时才需填写
plat	string	否	推送目标手机平台，默认0=所有平台 1=iphone

参数名	类型	必选	说明
			2=安卓
tag	string	是	消息标签，会填在STMsg.Tag推给客户端
content	string	是	消息内容
aps	string	否	os离线消息内容，json格式，见苹果文档
only_online	string	否	1=表示只发当前在线用户，默认0在线离线都会发
expire	string	否	消息有效期，单位秒，表示从现在起经过该时间之后该消息将被丢弃

注：

i)uid/wid：填写一个即可

i)参数格式：将参数按query_string格式拼接，用http post发送到接口

响应参数：

参数名	类型	必选	说明
errno	int	是	响应码
msg	string	否	错误信息
detail	string	否	终端推送结果

注：

i)数据格式：json格式

i)errno为0时，detail是详细信息

示例：

```
{
        "errno":0,
        "detail:"
        {
            [
```

```
                {
                    ret: 0,
                    wid: 111111
                }
            ],
        }
}
```

获取在线状态接口

接口说明：获取指定用户或终端的在线状态

接口名：get_online_status

请求方法：http post方法

请求参数:

参数名	类型	必选	说明
appid	int	是	第三方appid
secretid	string	是	第三方加密用的密钥id
sign	string	是	第三方签名
tm	int	是	时间戳，防请求重放
uid	string	是	用户唯一标识
wid	string	是	用户名下的某个设备标识，需要指定某台设备时才需填写

注：

i)参数格式：json

i)uid/wid：填写一个即可，按填写的id获取在线状态

i)批量：支持批量获取，上限是100个

示例：

```
{
            "appid":65538,
```

```
            "secretid":"asdadasd",
            "sign":"SADFKLJKLJCASDK",
            "tm": 435423123132113,
            "uid":["uid1", "uid2", "uid3"],
            "wid":[wid1, wid2, wid3]
}
```

响应参数：

参数名	类型	必选	说明
ret	int	是	响应码
msg	string	否	错误信息
data	数组	是	按uid和wid维度返回的在线状态数组
uid	uid在线数组	否	每个uid的在线状态(0:离线 1:在线)
wid	wid在线数组	否	每个wid的在线状态(0:离线 1:在线)

注：

i)参数格式：json

i)uid和wid关系：wid是设备id，一个用户(uid)可以最多登录5个设备(wid)

示例：

```
{
        "ret":0,
        "msg":"",
        "data:"
        {
            "uid":
            [
                {
```

```
                    "uid1":
                    [
                        {"wid1": 0},
                        {" wid2": 1},
                        { "wid3": 0}
                    ]
                },
                {
                    "uid2":
                    [
                        { "wid1": 0},
                        { "wid2": 0}
                    ]
                }
            ],
            "wid":
            [
                {"wid1": 0},
                {"wid2": 0},
                {"wid3": 0}
            ]
    }
}
```

通知在线状态

接口说明：获取指定用户或终端的在线状态

接口名：notify_online_status

请求方法：http post方法

请求参数:

参数名	类型	必选	说明
appid	int	是	第三方appid

参数名	类型	必选	说明
secretid	string	是	第三方加密用的密钥id
sign	string	是	第三方签名
tm	int	是	时间戳，防请求重放
status	状态数组	是	按uid数组推送在线状态(0:离线 1:在线)

注：

i)参数格式：json

i)批量：支持批量获取，上限是100个

示例：

```
{
            "appid":65538,
            "secretid":"asdadasd",
            "sign":"SADFKLJKLJCASDK",
            "tm": 435423123132113,
            "status":
            [
                {
                    "uid1":
                    [
                        {"wid1": 0},
                        {"wid2": 0},
                        {"wid3": 1}
                    ]
                },
                {
                    "uid2":
                    [
                        {"wid1": 0},
                        {"wid2": 0}
                    ]
```

```
                }
            ]
    }
```